DATA ANALYTICS IN CLIMATE CHANGE STUDIES

ESHTA BHARDWAJ

A THESIS SUBMITTED TO

THE FACULTY OF GRADUATE STUDIES

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

MASTER OF ARTS

GRADUATE PROGRAM IN INFORMATION SYSTEMS AND TECHNOLOGY

YORK UNIVERSITY

TORONTO, ONTARIO

APRIL 2022

Abstract

The observed trends of climate change have wide ramifications to the sustainability of a normal lifestyle. The application of data analysis techniques can increase the knowledge around climate studies while introducing additional research methods. The proposed research showcases the development of a novel data analytics framework to address the gap in data modeling for the comparison of climate model data and observational data. A detailed data pipeline for data collection, extraction, wrangling, analysis, and visualization is discussed. The practical implementation of the framework is presented through a visualization tool, Weather Analysis Regional Model, to aid researchers and practitioners in their analysis of climate data.

Acknowledgements

I would firstly like to thank Prof. Peter Khaiter who has been an incredibly encouraging, supportive, and knowledgeable supervisor during my master's studies. Prof. Khaiter provided guidance and expertise which helped me explore and understand the use of data analytics in the climate change domain which was novel for me. Under his supervision, I have been able to not only complete my work but grow as a researcher. I would also like to thank the members of my examination committee, Prof. Manar Jammal and Prof. Iris Epstein for their time, expertise, and feedback which helped improve this research endeavor. Lastly, I would like to thank my family and friends for their continuous support allowing me to complete this milestone successfully.

Table of Contents

List of Tables

List of Figures

Chapter 1: Introduction

Data analysis and visualization is valuable in all fields and industries as it provides data-driven results that can summarize and describe the data but also provide an overview of trends. Applying these techniques to climate change studies can increase the knowledge and accessibility of climate data. Accordingly, the objective of this thesis is to approach climate change studies from a data analysis perspective. This is demonstrated by applying each step of the data analysis process to compare data from a climate model against observed data for a common region and timespan. The comparison of climate data in such a way is necessary due to the extensive usage of climate models and climate datasets aimed at predicting future climate variability and its impacts on the environment, economy, and society. Ultimately, the proposed tool will be a means for researchers to conduct their own analyses based on the results of the comparisons.

Researchers can use the tool/framework to compare climate models and observed climate datasets on a regional level to inform their own research in a few ways. The field of impact modelling uses predictions on future climate variability and simulates the consequences to various industries. However, climate models provide climate predictions on a large geographic scale. Therefore, to aid policy makers with regional decision-making, isolating data for regional use and comparing it to observed values, will lead to greater accuracy and performance of impact models. There are also ongoing efforts in developing machine learning models that predict climate or perform climate change risk assessment. Both of these endeavours can be supported and improved by incorporating the results of a comparison tool that demonstrates the exact

difference in the daily values predicted and observed. The tool also presents opportunities for additional research to speculate whether climate models predict values more accurately on a regional level in certain geographic climates versus others.

Some considerations are introduced prior to discussing the research methodology to position the scope of this thesis. Firstly, the contributions of the comparative visualization tool are not equivalent to climate model validation. Climate model validation uses the technique of hindcasting (predicting past climate) to validate the accuracy of a model (NOAA, n.d.). However, climate models are made based on various assumptions, such as future greenhouse gas emissions. These and other assumptions cause variability in the predictions that cannot be discovered through hindcasting. The tool presented in this thesis does not attempt to validate or provide accuracy measures for climate models, but rather provide the differences in the predicted and observed values to aid in adjusting or tuning additional modelling or decision-making that is based on the results of climate models. Secondly, this tool is not meant to aid with developing reanalysis data. Climate reanalysis is the combination of historical weather observations and climate model data to create a synthesized picture of the past weather to eliminate any gaps in the weather record (ECMWF, n.d.). This tool does not contain the complexity required to contribute to reanalysis. Lastly, the tool does not perform any statistical downscaling (this concept is discussed in Chapter 2) because it uses a regional climate model for comparison. Since regional climate models perform this technique inherently within the development to ensure that the outputs are relevant to regional climate studies, no additional downscaling processes were performed.

The key research questions of this study are in Table 1.1.

Table 1.1: Research questions and motivations

N	Research Question	Motivation
RQ 1	To formulate a data analytics framework to compare climate datasets.	The climate studies domain has distinct factors which must be considered when conducting a data analysis project therefore requiring a novel framework.
RQ 2	To conduct a literature review and bibliometric analysis for the requirements gathering phase of the framework.	A broad literature review aids in learning the climate change domain. Bibliometric methods summarize and critique existing data analysis tools for climate data.
RQ 3	To determine sources for data collection and perform extraction.	Climate data requires specific and extensive data retrieval due to its size, format, and technical complexities.
RQ 4	To formulate a standardized data pipeline which performs data wrangling (i.e., cleaning, pre-processing, transforming, *etc.*).	There are numerous available climate datasets, each with varying formats. A standardized method of data preparation provides a roadmap for future data analytics projects using climate data.
RQ 5	To implement a visualization tool that showcases the climate studies data analytics framework.	A visualization tool assists future users and researchers with climatic analysis.

The corresponding research contributions are as follows:

RC 1: The design and development of a novel data analytics framework tailored for the comparison of climate datasets.

RC 2: An extensive literature review covering the broad scope of the climate change domain along with the use of bibliometric methods to perform state-of-the-art analysis on climate change analytics visualization tools.

RC 3: A stepwise guideline for collecting and extracting relevant climate data from large datasets.

RC 4: A standardized pipeline for automated data preparation in climate studies.

RC 5: The design and development of a visualization tool that applies the novel data analytics framework to aid in visual climatic analysis.

The rest of the paper is organized as follows. Chapter 2 overviews the climate change terminology and concepts pertinent to the analysis performed in this paper. It also presents a bibliometric review of prior analytics work comparing climate model data with observed data through visualization-related tools. Chapter 3 formally defines the analysis framework. Particularly, it outlines the scope of the data collected, the details of the dataset sources, the preprocessing steps, and the transformation of the data to perform comparisons. It also presents the visualization tool and various details on how it was developed. Chapter 4 contains a case study to demonstrate how the visualization tool can be used and its various outputs. Chapter 5 considers how results from the tool can be used to discover research questions and other uses of the framework and tool. Lastly, Chapter 6 concludes the thesis with a summary of the work performed, a discussion of the challenges faced, and potential future work.

Chapter 2: Literature Review

2.1: Definition of Climate Change

The field of data analytics involves the investigation, interpretation, and evaluation of a given dataset to verify a series of hypotheses or perform exploratory tasks with the goal of discovery. This field can be further expanded to include data visualization; as without a graphical representation, the use of complex data and analyses has a limited audience. The combination of data analytics and visualization can be applied to a plethora of industries to enhance their usage of data in making data-driven decisions. The focus of Chapter 2.1 is to obtain domain-knowledge expertise of climate studies in order to apply data analytics and visualization techniques. Accordingly, it serves as the requirements gathering phase (step 1) of the data analytics framework and can be mapped to research contribution 1 as outlined in the introduction.

2.1.1: Overview

Climate refers to the long-term weather patterns observed for a given region (CCCS, 2020; Denchak, 2017). Therefore, climate change is the drastic shift in expected weather conditions over time (CCCS, 2020; Denchak, 2017). Climate forcing is a process that alters the expected climate through various climate drivers. The climate system is composed of the hydrosphere, land surface, cryosphere, biosphere, and atmosphere; climate change factors can be considered internal or external based on whether the forcings exist inside or outside the climate system (Colose et al., 2020). Internal mechanisms that cause climate change exist completely within the climate system and are due to interactions and dynamics between its components (Baede et al., 2018). On the other hand, external forcings can occur due to natural causes or by

anthropogenic activities, both of which occur separately from the processes of the climate system (CCCS, 2020; Colose et al., 2020; Baede et al., 2018).

Earth's energy budget is a key concept in understanding external drivers of climate change; it measures the incoming and outgoing radiation energy from the Sun to Earth and the energy emitted from Earth back into space (Afework et al., 2018; Afework et al., 2019; Baede et al., 2018; Colose et al., 2020). The energy emitted from the Sun interacts with Earth in various ways, such as: being absorbed by the atmosphere, moving through the atmosphere towards the surface, or emitting back into space (NOAAa, n.d.). The energy that reaches the Earth can create a large impact on the climate based on the energy flows that occur (Afework et al., 2019). Energy flows describe how energy is distributed within the climate system and its interactions with other objects such as plants, animals, water bodies, etc. As a result, the impact on Earth's climate can be summarized by measuring the planet's radiative forcing which is the difference between the incoming energy from the Sun and the outgoing energy from the Earth (NOAAa, n.d.) The first law of thermodynamics states that energy cannot be created or destroyed, therefore the Earth's energy budget is considered to be at equilibrium if a balance is achieved between incoming and outgoing energy (Afework et al., 2019). However, if incoming energy is greater than outgoing energy, this causes positive radiative forcing. As a whole, Earth's climate depends on this radiative balance (Baede et al., 2018). Any activities (natural or anthropogenic) that create a warming effect are of interest when measuring climate change.

2.1.2: Measurement of Climate Change

Climatologists measure climate change using a variety of observations from past and present climate change key indicators (GCB, n.d.). Earth-orbiting satellites, meteorological stations, ocean buoys, and more are used for measuring and monitoring present climate data (Denchak, 2017).

The concept of essential climate variables (ECVs) was developed by The Global Climate Observing System to create a baseline of variables that should be used widely to measure, monitor, and understand past, present, and future climate variability. The ECVs are created based on three criteria: relevance (whether a given variable is an identifier or definition of climate), feasibility (whether the variable can be observed in a practical manner with scientifically proven methods), and cost effectiveness (whether creating and maintaining data on the variable is economical) (WMO, n.d.). The essential climate variables are divided into 3 categories including: atmospheric, oceanic, and terrestrial with further subcategories as seen in Table 2.1.

Table 2.1: Summary of essential climate variables (Bojinski et al., 2014)

Atmospheric	Surface:[a]	Air temperature, wind speed and direction, water vapor, pressure, precipitation, surface radiation budget
	Upper air:[b]	Temperature, wind speed and direction, water vapor, cloud properties, Earth radiation budget (including solar irradiance)
	Composition:	Carbon dioxide, methane, other long-lived greenhouse gases,[c] ozone and aerosol supported by their precursors[d]
Oceanic	Surface:[e]	Sea surface temperature, sea surface salinity, sea level, sea state, sea ice, surface current, ocean color, carbon dioxide partial pressure, ocean acidity, phytoplankton
	Subsurface:	Temperature, salinity, current, nutrients, carbon dioxide partial pressure, ocean acidity, oxygen, tracers
Terrestrial		River discharge, water use, groundwater, lakes, snow cover, glaciers and ice caps, ice sheets, permafrost, albedo, land cover (including vegetation type), fraction of absorbed photosynthetically active radiation, leaf area index, above-ground biomass, soil carbon, fire disturbance, soil moisture

The definition of climate change is multifaceted. Although emphasis is placed on global temperature as the main indicator of climate, with the focus being on the rising

surface temperature over the last 150 years; a more accurate representation can be obtained using an all-rounded view (Williams & Eggleston, 2017). An ideal shortlist is presented by Williams and Eggleston (2017), by narrowing down the 54 ECVs to 6 key indicators. While the suggestions presented in the article aim to reduce the scope and complexity of measuring climate change, over-simplification of some indicators also causes inaccuracies and uncertainties. Specifically, examining global precipitation changes at the local level can be highly variable. According to the article, a deeper analysis of annual, multi-year, and long-term changes in solid and liquid precipitation, rainstorms, drought, and more would be required for precipitation to be a useful and accurate measurement of climate change (Williams & Eggleston, 2017).

Reyes-Garcia et al. (2016) place importance on the local knowledge and observation of indicators of climate change. Their claim is that local knowledge can be leveraged to understand local indicators of climate change and its impacts on a more grounded and narrow level. The authors gathered data from documents published from 2014 and onwards about indigenous knowledge about climate change and categorized indicators into: local observations of climate change, observed impacts of physical systems, observed impacts on biological systems, and observed impacts on socio-economic systems (Reyes-Garcia et al., 2016). Focussing on the first two categories of indicators gives an idea of local indicators of climate change, whereas the last two cover the impacts it causes. Variables including temperature, precipitation, wind, hydrological variables, and cryosphere variables are covered in these categories.

Patricia McCarney's (2012) paper highlights a framework for cities to perform planning, implement policies and management, and investment based on climate

metrics at a city level. As part of the framework established, she highlights 4 main areas of "urban vulnerabilities associated with climate change" (McCarney, 2012). These include variabilities in temperature, increase in precipitation, drought, change in sea level, and increase in storm activity. Each of these is then mapped to various city impacts. By narrowing down the focus to key local indicators and specifically highlighting individual variables within large groups (such as drought within the precipitation category), more focussed and accessible analytics can be performed about regional climate change.

While investigating climate change at a regional level can be a challenging task to perform with accuracy, the concept of ECVs provides a standardized method for describing and evaluating climate variables that is widely accepted and used. This paper narrows down the scope of the variables used to evaluate climate change based on guidelines provided by previous literature and the availability of datasets with equivalent or comparable variables. This is discussed further in Chapter 3.

2.1.3: Climate Projections

Climate models use mathematical equations to emulate the processes and interactions of the climate system. This includes dynamics of each component of the climate system, the interactions between components of the climate system, and interactions between the climate system and external factors. The use of these mathematical equations creates a simulation of the energy transfer that drives global climate which is then used to predict future climate. On a high-level, building a climate model is comprised of the following steps: 1) identifying processes of the Earth's climate system, 2) quantifying the systems mathematically, 3) representing the exchange of

energy using equations, 4) creating base conditions and conditions that will change over time, and 5) solve the equations using the base and subsequent conditions to predict future climate (NOAAb, n.d.).

To perform the mathematical quantification of the global climate system, climate models do not consider the energy transfer between each cubic meter. Instead, the Earth is divided into grid cells (also referred to as tiles in this study) and the exchange of matter and energy between neighbouring cells is modelled. The resolution of a grid cell is defined by its size in longitude and latitude. A coarse global climate model would have a spatial resolution of 100kmx100km, however higher resolution models have a smaller area per tile and greater number of boxes. Climate models also include a time step factor which signifies how often the model will calculate the energy transfer. The time granularity and spatial resolution influence the computing power required to run the model with smaller time steps and grid cells requiring higher computing power (NOAAb, n.d.).

One of the vital inputs for climate models include how much energy from the Sun is absorbed or trapped by the Earth's atmosphere; a method of measuring this is through greenhouse gas (GHG) emissions. Since GHGs are highly affected by anthropogenic activities, it becomes important to accurately represent the future changes in GHG emissions caused by human behaviour (McSweeney & Hausfather, 2018). Since there is no method to predict societal development, probable scenarios are being created to simulate human patterns in the future. These scenarios ensure standardization across various climate science projects and research. In general, each scenario takes GHG emissions, population growth, economic development,

technological change, and other drivers into consideration to aptly model potential futures (CCCS, 2018; CCCMA, 2010). In 2000, the International Panel on Climate Change (IPCC) developed 40 scenarios each of which was categorized in groups: A1, A2, B1, B2. However, in 2014, Representative Concentration Pathways (RCPs) were created by the IPCC to supersede the SRES scenarios (Wayne, 2013). There are four main pathways: RCP8.5, RCP6, RCP4.5 and RCP2.6. The Government of Canada uses RCP8.5, RCP4.5 and RCP2.6. RCP8.5 is a high global emission scenario that indicates severe global average warming levels, it is used as the basis for simulating worst-case climate scenarios. RCP4.5 is a medium scenario with measures to reduce climate change and indicates medium global warming levels. Lastly, RCP2.6 indicates very low global average warming levels but would require strong mitigation actions (CCCS, 2018).

2.1.4: Types of Climate Models

There are three main types of climate models: energy balance models (EBMs), intermediate complexity models, and general circulation models (GCMs). EBMs are the basic numerical climate models which use the balance of Earth's energy budget as input and generate surface temperature as its only output (McSweeney & Hausfather, 2018). Intermediate complexity models are made up of 2D-EBMs which also include energy and moisture components. They may also include bio-geochemistry and land ice components (land, oceans, ice features, *etc*.). Due to the larger variety of inputs as compared to EBMs, intermediate complexity models can also model ocean current shifts and atmospheric components (McSweeney & Hausfather, 2018). Lastly, GCMs are the most complex of the three types of models and aim to simulate the physics of

the entire climate system as its input. This includes atmospheric chemistry data, glacial makeup, air flow processes, oceanic components, heat transfer, and more (Schmittner, 2018). Furthermore, atmosphere-ocean GCMs (AOGCMs) use data on the exchange of heat between upper-air, land, and ocean surfaces. The latest GCM models also simulate biogeochemical cycles and its interactions with the climate system; these models are called Earth system models (ESMs). Biogeochemistry is used to factor in processes like the carbon cycle, nitrogen cycle, changes in vegetation, *etc*. to be able to better simulate future climate which includes the effect of anthropogenic emission activities (Schmittner, 2018).

With the use of GCMs, regional climate models (RCMs) were created to allow climate modelling on a regional level (Tapiador et al., 2020). Climate change and its impacts can become personal and substantive to governments, organizations, and even individuals when thought about as a local phenomenon. Regional climate modelling uses the technique of downscaling, which can be statistical or dynamic (Rummukainen, 2009). Statistical downscaling maps statistical relationships between large climate variables and local ones; this accordingly maps GCM results to refined results which can be used as input into regional climate studies. An example of this approach is mapping mean sea level pressure field to temperature or precipitation (Rummukainen, 2009). Dynamic downscaling is applying data from higher resolution global model to a specific region to increase the granularity of the data outputs while reserving computational complexity. Tapiador et al. (2020) present a comprehensive list of regional climate models developed over the last 30 years.

2.1.5: Evaluation and Reliability

Since the initial development of RCMs in the late 1980s, there has been debate about the value of information provided by the models. As innovation and advancement progress with GCMs, there is a possibility that further finer resolutions can be accomplished within the next 5-10 years rendering the current usefulness of RCMs obsolete. A study by Giorgi (2019) highlights the achievements and issues with the RCMs reviewing the progress made since its development. A key point made in this study is the steady increase in literature, research, and use of RCMs to date. Not only are RCMs being used to for research and validation purposes but are being applied to a variety of climate study problems. Furthermore, the conception of regional earth system models (RESMs) provides increased data at increased resolutions. Lastly, the Coordinated Regional Climate Downscaling Experiment (CORDEX) has been developed for the climate science to achieve standardized methods of evaluation, while cultivating knowledge transfer between experts to better understand and improve the use of RCMs (Giorgi, 2019). Conversely, there are a variety of issues highlighted in Giorgi's study (2019). An issue surrounded by debate is regarding the added value (AV) of dynamic downscaling used within RCMs. Since RCMs are developed using coarse GCMs with finer resolutions over specific regions, there is debate as to the added value of the downscaling technique as compared to the performance of the climate model. For a given RCM, the performance may be attributed to the downscaling technique applied, when it could be a cause of the global climate model its derived from. Overall, the AV concept requires specific, quantitative methods of measurement that are applicable across RCM models and their settings (Giorgi, 2019). Furthermore, the sensitivity of the

experiment configuration can cause a high degree of variability in the RCM results. A standardized framework (or set of frameworks/procedures/best practices) is required to ensure that the evaluation of RCMs accurately reflects the decisions made in the domain selection, LBC technique, initial conditions, and validations made for each individual RCM (Giorgi, 2019).

Although there is a level of uncertainty with RCMs, the creation of RCMs is only complete after performing validation tests using a technique called hind-casting. Hind-casting involves using the completed climate model to generate outputs for a past time period. The results can then be compared with the observed climate values to determine the accuracy of the climate model produced. Upon completing hind-casting validations, climate models are then used to perform future climate simulations (NOAAb, n.d.) Furthermore, organizations like the IPCC, a United Nations body, issue periodic evaluation reports of a large variety of climate models and review the progress and development of climate modelling globally.

2.2: Climate Data Modeling and Comparison

Gagnon et al. (2009) performed regional validation of the CRCM 4.1 climate model. They examined 4 variables: minimum and maximum daily temperatures, total precipitation, and total runoff for two watersheds in Quebec, Canada. Their aim was to compare projected numbers against the observations made for the same variables to show whether predictions can be used as a reliable source to model future climate change (Gagnon et al., 2009). In their study, they considered that comparing observed values with climate models' simulated values is not recommended since an observation records a single point of location whereas climate models cover the entirety of one tile

(45km x 45km for CRCM 4.1). Therefore, they transformed the predicted climate value for a given time granularity to the weighted mean of all the tiles that span a single watershed. The same was done for the observed values (Gagnon et al., 2009). In this way, each of the 2 watersheds only had one observed and predicted value for each of the 4 climate variables. This paper shows one of the few to address how to compare regional climate model data with observed climate. Furthermore, a large time span is covered, and the geographic area is within Canada, both of which are valuable contributions. The work presented in this thesis continues the research presented by Gagnon et al. (2009) by examining a larger variety of climate variables recorded by climate models. Additionally, the predicted variables are mapped to ECVs to ensure that the results can be used to make a commentary on the useability of climate model projections. Furthermore, a visualization and data download tool are presented, so that other researchers, students, and educators can benefit from the collected data and research.

The following visualization tools discussed do not have an accompanying research paper published and therefore were not captured in the bibliometric analysis presented in Chapter 2.3. The Power Analytics and Visualization for Climate Science (PAVICS) tool is a virtual laboratory that provides access to climate studies data and tools for analysing and visualizing (PAVICS, n.d.). There are 4 datasets provided. The first one is climate projection data for precipitation, minimum temperature, and maximum temperature. The model used is the CMIP5 with data from 1950 to 2100. The second dataset is observed data for the same variables from 1950 to 2017. The third dataset reanalyses data for precipitation and maximum temperature from 1979 to 2020.

Lastly, a current weather forecast dataset is provided for precipitation and maximum temperature. These datasets can be downloaded from the THREDDS data server hosted by PAVICS. At the same time, a tutorial is provided for accessing multiple datasets and splicing them for the information of interest (based on latitude, longitude, and time). Further tutorials are provided for calculating climate indices, ensemble statistics, and visualizations. The visualization tutorial includes a topic on creating a PAVICS dashboard. The interactive dashboard allows end-users to choose between two temperature variables, the season, region, and year range. The tools presented by PAVICS certainly allow the researchers to recreate visualizations and perform analyses on the 4 hosted datasets, however there is a limitation in the regions and variables available to explore. Specifically, this tool only contains data for precipitation, minimum temperature, and maximum temperature and these variables have the most commonly available data and many resources that have performed analysis on them. Furthermore, a comparison cannot be made between the projected and observed variables since the data is not presented together and therefore an analysis is not possible. For users to compare the provided data, they must access the data (through the method described in the tutorials) but create their own visualizations and perform their own analyses. The visualization tool in this study aims to bridge this gap by allowing end-users to choose from a larger set of climate variables and compare predicted and observed values for the same region and time periods.

The Program for Climate Model Diagnosis and Intercomparison (PCMDI) also holds a large majority of the CMIP models' data and provides tools to facilitate support and analysis for these datasets (PCMDI, n.d.) The PCMDI Metrics package, in

particular, provides the ability to compare results from the climate model with observations and various statistical tests (PCMDI, n.d.) However, both PAVICS and PCMDI, can only provide data and analytics for GCMs. Therefore, there remains a gap in the analysis tools available for research and decision-making on a regional level.

The United States Environment Protection Agency (US EPA) presents its own list of climate change indicators that differ from the standard ECVs (US EPA, n.d.a). These variables are derived from contributors in government agencies, academic institutions, *etc*. and are used as input to the National Climate Assessment which is a quadrennial assessment completed by the US Global Change Research Program; they are also considered federal indicators. The US EPA's indicators are presented in 6 categories and are summarized in table 2.2. Although the list of variables is presented as indicators of climate change, some variables, particularly in the health and society category, present effects of climate change. The focus of this study is primarily to examine variables that are used to measure climate change. The variables under consideration must face change internally or externally (to the climate system), and the variability of the measurements is studied to analyze climate change.

Table 2.2: Summary of US EPA's climate change indicators (US EPA, n.d.)

Greenhouse Gases	U.S. Greenhouse Gas Emissions, Global Greenhouse Gas Emissions, Atmospheric Concentration of Greenhouse Gases, Climate Forcing
Weather and Climate	U.S. and Global Temperature, Seasonal Temperature, High and Low Temperatures, Heat Waves, U.S. and Global Precipitation, Heavy Precipitation, Tropical Cyclone Activity, River Flooding, Drought
Oceans	Ocean Heat, Sea Surface Temperature, Sea Level, Coastal Flooding, Ocean Acidity

Snow and Ice	Arctic Sea Ice, Antarctic Sea Ice, Ice Sheets, Glaciers, Lake Ice, Snowfall, Snow Cover, Snowpack, Permafrost, Freeze-Thaw Conditions
Health and Society	Heat-Related Deaths, Heat-Related Illnesses, Cold-Related Deaths, Heating and Cooling Degree Days, Residential Energy Use, Lyme Disease, West Nile Virus, Length of Growing Season, Growing Degree Days, Ragweed Pollen Season
Ecosystems	Wildfires, Streamflow, Stream Temperature, Lake Temperature, Great Lakes Water Levels and Temperatures, Bird Wintering Ranges, Marine Species Distribution, Leaf and Bloom Dates

The US EPA's climate change indicators portal also contains: 'discover pages' for all the indicators, a climate indicators explorer, and an indicators overview story map (US EPA, n.d.). The visualizations provided in the 'discover pages' are only for the U.S. with the starting years anywhere between the early 1900s to the late 1990s and ending near 2020. Each climate variable has a minimum of one visualization provided, with the option to download the image as well as the data (US EPA, n.d.). One of the key contributions of the US EPA's climate change resource is the wide variety of variables collected. Each variable has data associated with it that is easily accessible to an end-user. There is detailed information provided about the importance of the variable: why it is collected, various past trends, knowledge transfer about the data, its source, the collection method, and any technical documentation. However, the data collected is only for the U.S. and the data download only denotes the values already visualized in the graph meaning that the raw data is not available for download. Therefore, for researchers, students, and educators, this resource does not contain enough granularity for independent and in-depth research and analysis. The climate indicators explorer (as compared to the discovery pages) presents only the visualizations for various climate indicators in tab-view style webpage (with the option to download data for each graph).

This would be an extremely effective tool if a larger geographic range was covered and if the data available for download was at a finer time and geographic granularity (US EPA, n.d.). Lastly, the indicators' overview story map is a tab-style presentation tool with 7 tabs; an introduction tab, 5 tabs for indicator categories, and a conclusion tab that presents methods for reducing the anthropogenic effects of climate change (US EPA, n.d.). Each individual tab is akin to a "scrollytelling" dashboard, in which a few visualizations are provided alongside small blurbs of text for explanation. The climate indicators explorer is effective as a presentation tool because it provides a high-level overview with compelling graphics and easy-to-understand text. However, it is not an effective tool for researchers aiming to perform their own climate studies and analysis, whereas the visualization tool presented in this thesis fills the gap between high-level aggregated data and complex raw data readable only by experts in climatology.

2.3: Bibliometric Analysis

Bibliometric analyses are the quantitative evaluation of research publications (Ellegaard et al., 2015; Szomszor et al., 2021). Bibliometric methods allow a large volume of literature to be reviewed and summarized for any chosen subject matter that has sufficient publication data. A bibliometric analysis can serve many purposes. Primarily, it can be used to summarize the research activity of a given topic for a large period of time (i.e., to map the state of the art). The summary can be further used to identify trends in the research area, analyze scientific contribution, and discover gaps in the literature which can be used to derive areas that need further exploration (Donthu et al., 2021).

One of the key research contributions of this thesis is the creation of a visualization tool for comparing climate model data with observed climate data. To position this contribution as novel within the field, a bibliometric analysis was conducted. The stepwise approaches outlined in by De Oliveira et al. (2019) and Zare et al. (2017) were used as guiding sources to frame the methodology but revised to fit the research topic and data at hand.

2.3.1: Approach

The first task performed was establishing the purpose(s) of conducting the analysis, henceforth referred to as step 1. The outcome of this step directly influenced the criteria of the search strategy for retrieving the research publications within scope (step 2). The third step was the identification of bibliometric methods that should be applied in order to answer the questions formed in step 1. The fourth and most important step was the data wrangling to create subsets or aggregations of data in order to apply the identified bibliometric methods. Lastly, the results were analyzed to glean insights and satisfy the purpose of conducting a bibliometric analysis. The overview of the approach is summarized in Figure 2.1.

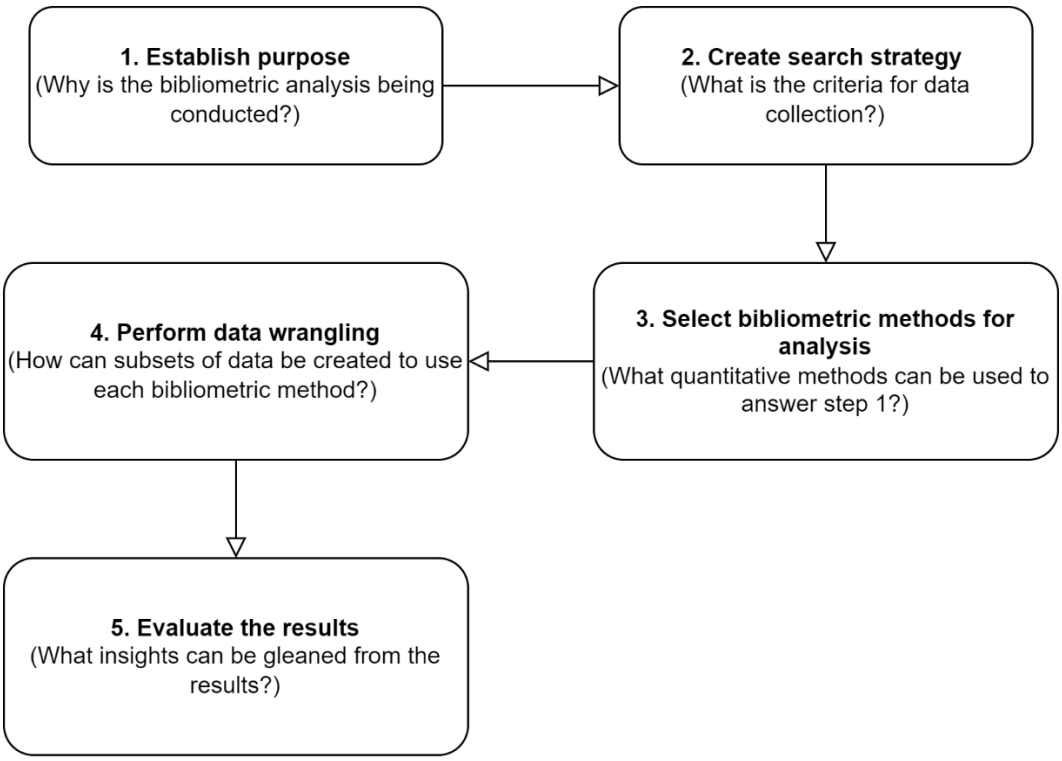


Figure 2.1: Overview of approach

2.3.2: Methodology

Step 1: Establish Purpose

The primary purpose of conducting a bibliometric analysis was to demonstrate that the development of the climatic visualization tool is novel. In order to do so, it was first necessary to obtain an understanding of the visualization tools developed over time that perform climatic or climate-adjacent analysis. For this reason, additional questions were proposed in step 1 so that the bibliometric analysis could aid in mapping the state-of-the-art. Hence, answering the list of questions in Figure 2.2 served as the purpose of conducting the bibliometric analysis.

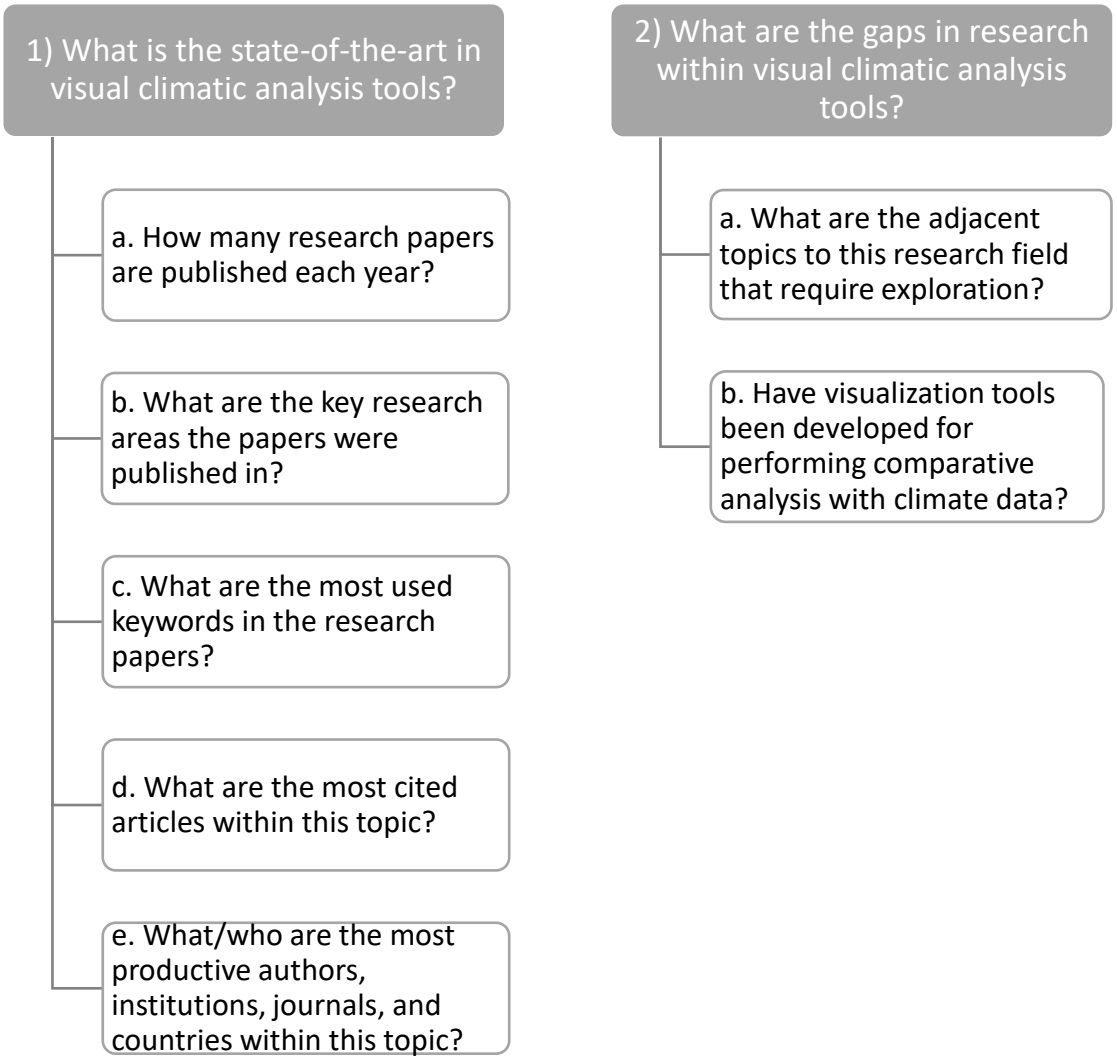


Figure 2.2: Purpose of bibliometric analysis

Step 2: Create search strategy

To answer the questions outlined in Figure 2.2, the Web of Science core collection databased was used. Web of Science also has multiple tools available that simplify the data mining process. Only one database was chosen to limit the data integration and preprocessing steps required to merge results from two or more data sources.

The next part of the search strategy was selecting search criteria that would reflect the topic accurately. The search query entered was:

> TS=((("data science" OR "data analytics" OR "data analysis") AND ("climate change" OR "climate model" OR "weather") AND ("tool" OR "system" OR "decision support" OR "visual" OR "interactive"))) and Articles or Review Articles (Document Types) and English (Languages) and 2022 (Exclude – Publication Years)

The topic of interest was "climate change data analysis visualization tools". Each word or phrase within this topic was reflected in the creation of the query. The first group of keywords was relating to data analysis, the second was relating to climate change, and the third was relating to the presentation of climate data analysis through a type of tool. These keywords were searched in the topic (indicated by TS) which includes the article title, abstract, and keywords. The remainder of the search query reflects inclusion and exclusion criteria. Based on the recommendations made by De Oliveira et al., the document type was limited to articles and review articles since those are usually peer-reviewed (2019). Only documents in English were included in the scope of analysis and the year range selected was 1990-2021 inclusive. There were 746 articles under analysis.

Step 3: Select bibliometric methods

A bibliometric parameter or method was assigned to each of the questions in step 1 to reach a quantitative answer (except 2b). This mapping is summarized in Table 2.3.

Table 2.3: Bibliometric methods

Question from Step 1	Bibliometric Method/Parameter
1a. How many research papers are published each year?	Evolution of publications (summary)
1b. What are the key research areas the papers were published in?	Most productive research areas (summary)
1c. What are the most used keywords in the research papers?	Most used keywords (summary)
1d. What are the most cited articles within this topic?	Evolution of citations (summary) Top 10 cited articles (ranking)
1e. What/who are the most productive authors, institutions, journals, and countries within this topic?	Most productive authors, institutions, journals and countries (summary)
2a. What are the adjacent topics to this research field that require exploration?	Co-word analysis / co-occurrence analysis
2b. Have visualization tools been developed for performing comparative analysis with climate data?	Manual analysis to determine relevance and novelty

Step 4: Perform data wrangling

The evolution of publications seen in Figure 2.3 shows 0 publications in the years 1990-1993. From 1994 onwards, there was a gradual increase in the number of publications, increasing by 2-5 articles each year. In 2016, the number of publications increased from 39 to 69 indicating a 56% growth. The overall trend displays a rapid annual growth in publications from 2016 onwards.

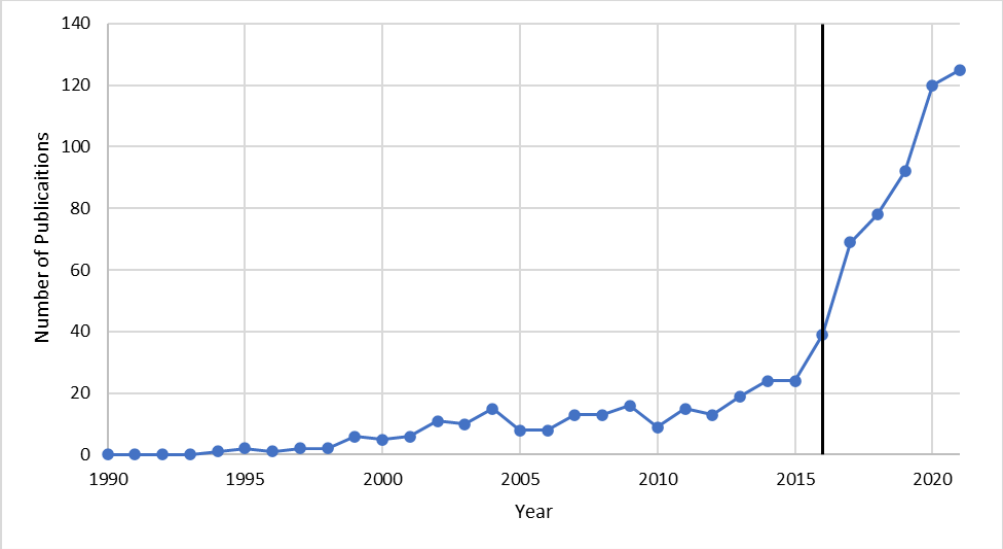


Figure 2.3: Evolution of publications

There are over 80 research areas with publications relating to visual climatic analysis. The top 25 research areas are shown in Figure 2.4. The top 3 research areas include: environmental sciences ecology, engineering, and meteorology atmospheric sciences. It is understandable that these 3 areas are the most researched as they represent "core" climate change-related topics. However, climate change impacts every aspect of daily living, and the use of visual climatic analysis needs to be further developed in underexplored research areas such as: transportation, forestry, water resources, *etc.*

To create the word cloud seen in Figure 2.5, the author keywords were extracted from the Web of Science corpus of 746 documents. To ensure the word cloud would consider the frequency of each phrase and not individual words, multiple preprocessing steps were performed. The final list contained each phrase separated by commas and words within a phrase separated by a tilde. The website wordclouds.com, published by Zygomatic, was used to generate the word cloud image. After loading the words into

the platform, the keywords' list was edited to remove phrases that contained keywords used in the search query (as those would evidently have the highest frequency). The top 10 most frequently used keywords included (in order): "machine learning", "internet of things", "sustainability", "agriculture", "artificial intelligence", "energy efficiency", "radio occultation", "adaptation", and "atmospheric effects". However, the frequency of the most used commonly used phrase was only 27 (out of 746 total documents). This indicates that none of the related research areas have been sufficiently explored or over-explored. Furthermore, a total of 1499 phrases were part of the analysis (after pruning the list), only 17 of those phrases had greater than 5 occurrences. This indicates that the research performed within this topic is extremely varied.

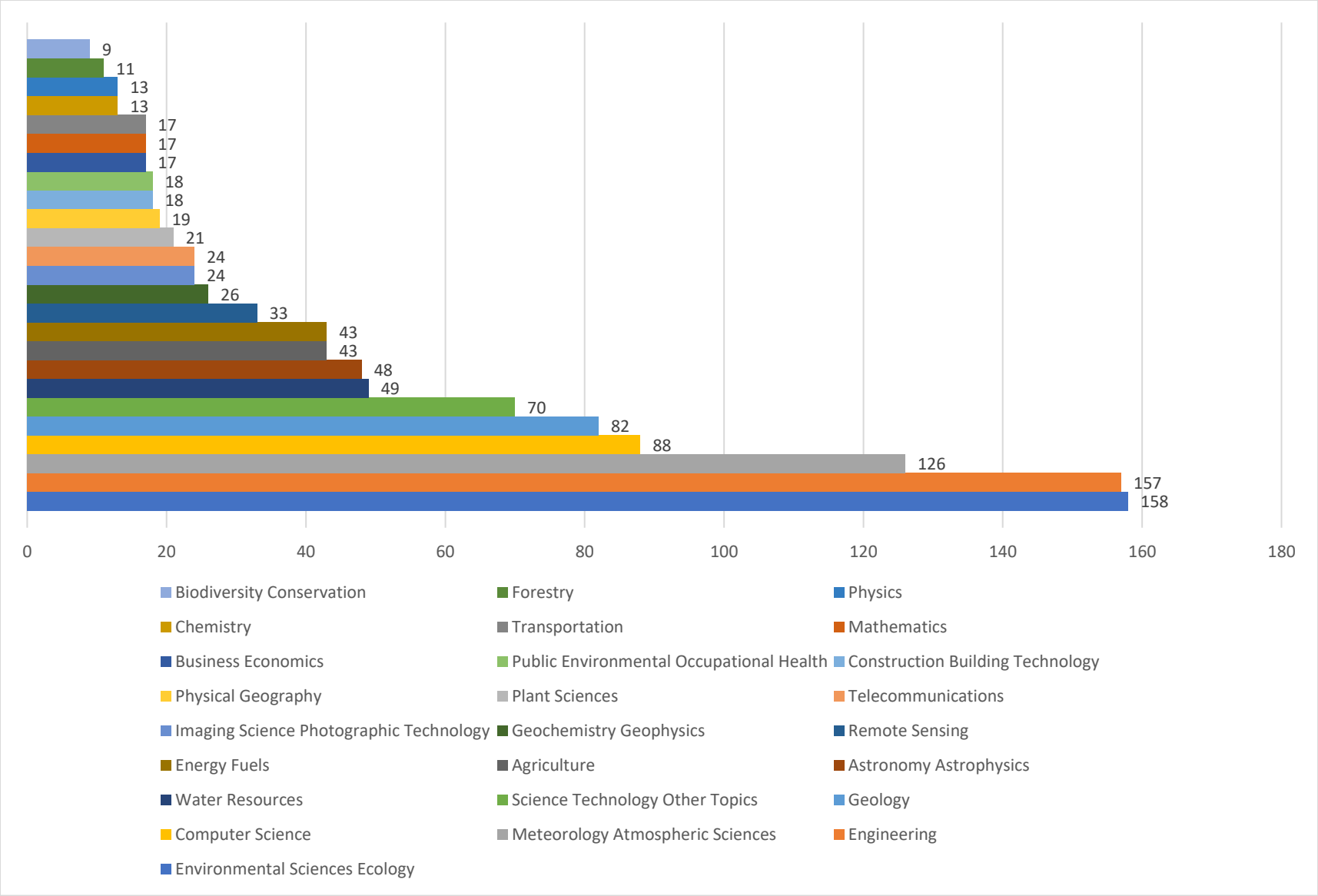


Figure 2.4: Most productive research areas

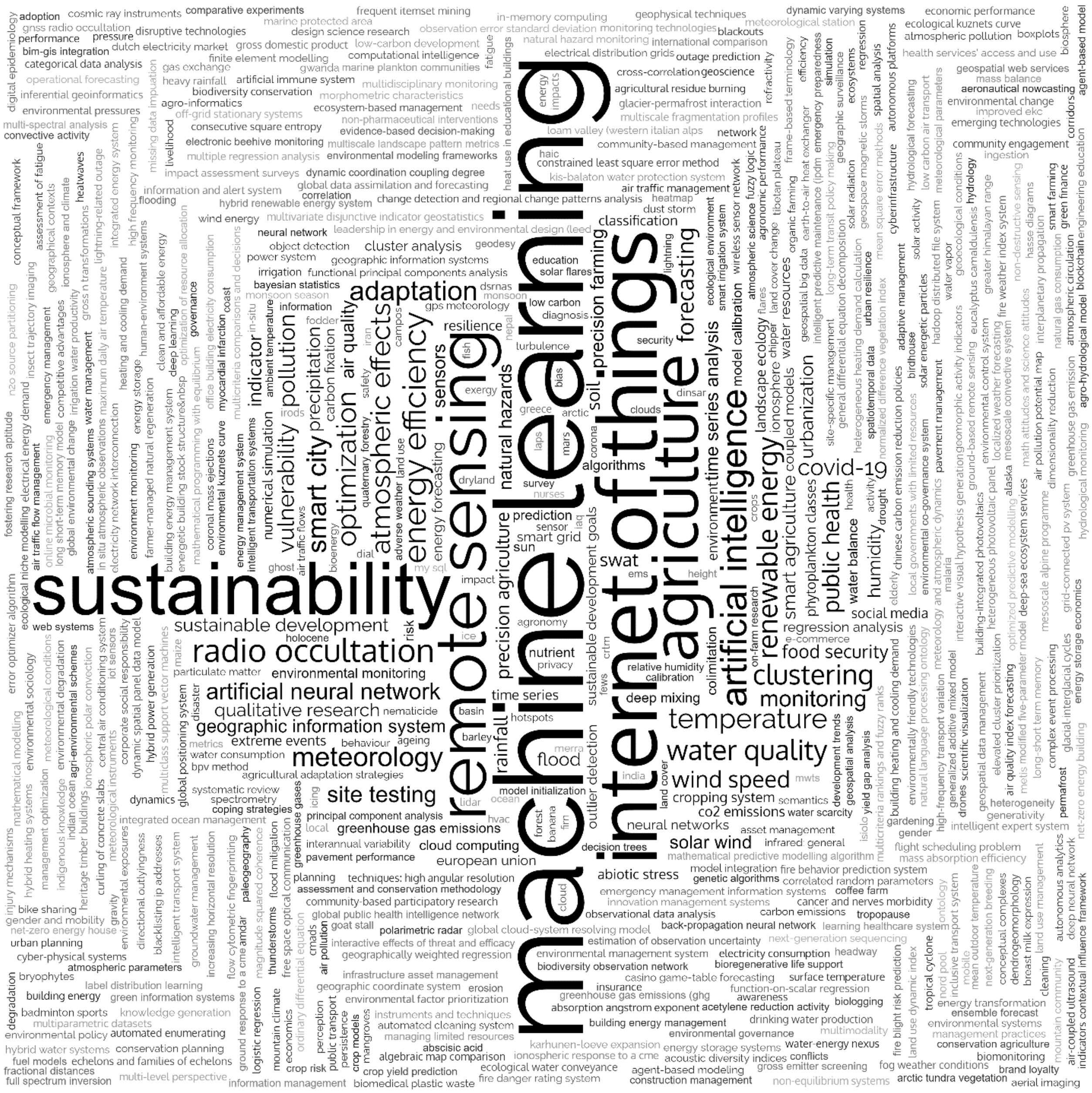


Figure 2.5: Word cloud of most used words/phrases

The evolution of citations is mapped with dual y-axes alongside the number of publications in Figure 2.6. This visualization demonstrates that the number of citations grew in a similar trend as the publications. For the 746 articles published, there were a total of 13658 citations, averaging roughly 18 citations per publication. The median is

approximately 6 citations per publication, ranging from 0-615 citations. The h-index is 55 meaning that 55 publications in the corpus have been cited 55 or more times.

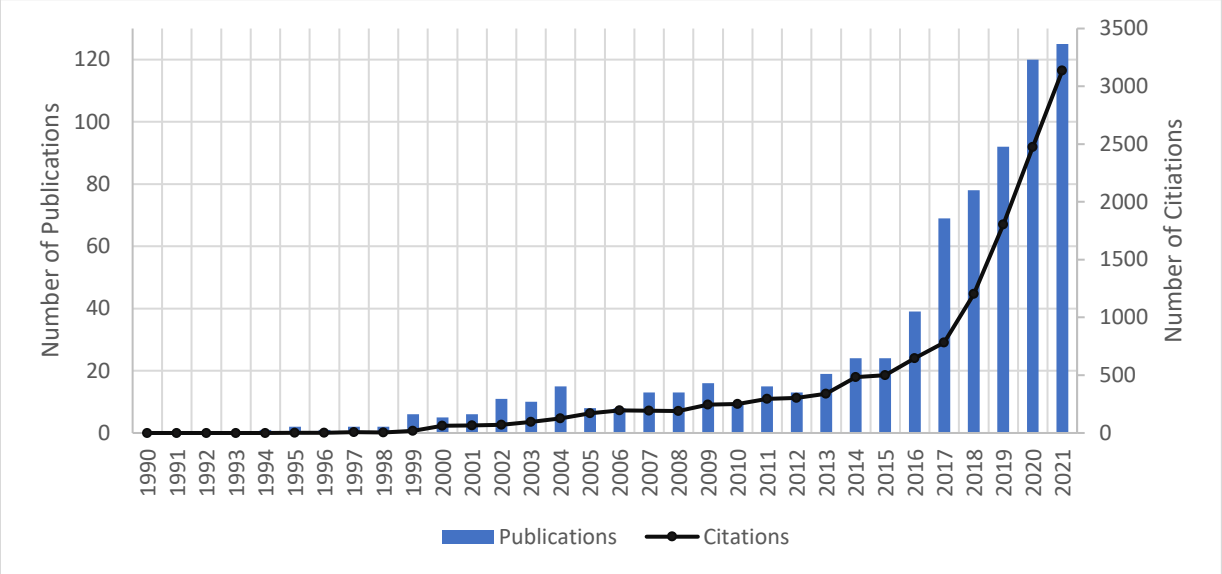
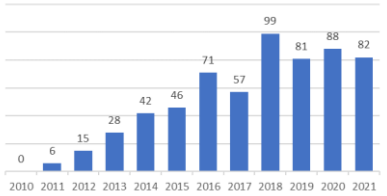
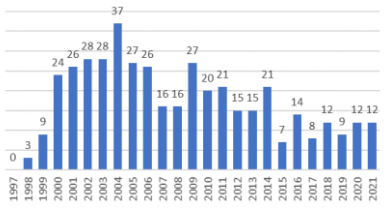
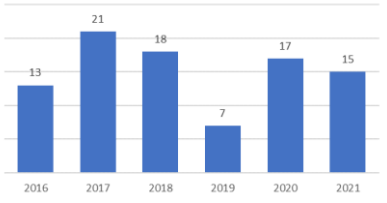
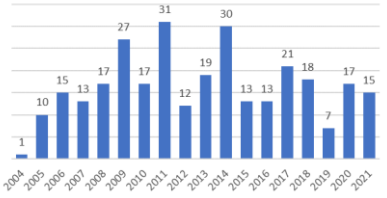
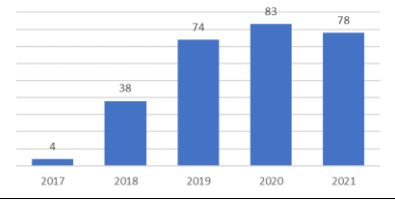
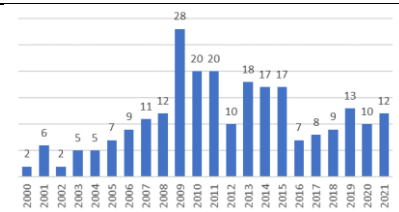
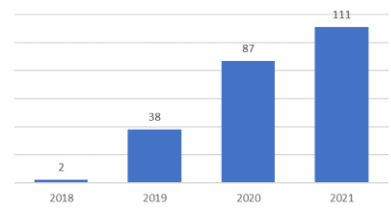
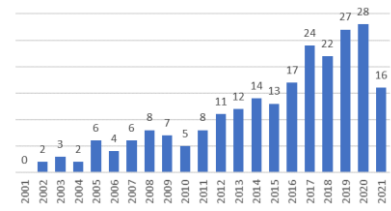
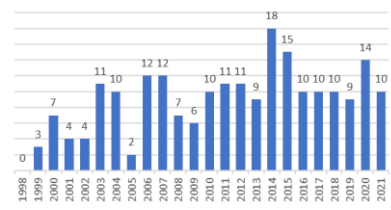
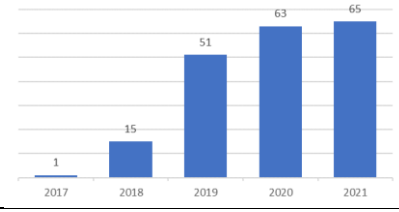


Figure 2.6: Evolution of citations

The top 10 cited articles according to total number of citations within the research topic are presented in Table 2.4. These articles are the prominent research contributions in the year range 1990-2021. There are no repeated authors or journals within the list which further emphasizes the diversity of research within this topic. However, the research areas of half the publications include Meteorology & Atmospheric Sciences as one of the research areas. This is understandable due to the meteorological nature of the climate change domain. The journal impact factor measure is also included in the top 10 summary to provide additional details on the research contribution of each article. The journal impact factor is measured by the number of times articles from a journal are cited within a year. The impact factor concludes that a journal is more important within its field given that publications from that journal are highly cited.

Table 2.4: Top 10 most cited articles

#	Article Title	Author(s) (Year)	Country	Total Citations	Average Citations per Year	Journal Title	Journal Impact Factor	Evolution of Citations per Year
1	Review article 'Assessment of economic flood damage'	Merz et al., (2010)	Germany and Austria	615	51.25	Natural Hazards and Earth System Sciences	4.345	
2	Analysis and validation of GPS/MET data in the neutral atmosphere	Rocken et al., (1997)	USA and Russia	433	17.32	Journal of Geophysical Research-Atmospheres	4.261	
3	Urban planning and building smart cities based on the Internet of Things using Big Data analytics	Rathore et al., (2016)	South Korea	340	56.67	Computer Networks	4.474	
4	Inversion and error estimation of GPS radio occultation data	Kuo et al., (2004)	USA	296	16.44	Journal of the Meteorological Society of Japan	2.236	
5	A review on time series forecasting techniques for building energy consumption	Deb et al., (2017)	Singapore	277	55.40	Renewable & Sustainable Energy Reviews	14.982	

#	Article Title	Author(s) (Year)	Country	Total Citations	Average Citations per Year	Journal Title	Journal Impact Factor	Evolution of Citations per Year
6	COSMIC system description	Rocken et al., (2000)	USA	248	11.27	Terrestrial Atmospheric and Oceanic Sciences	1.009	
7	An Overview of Internet of Things (IoT) and Data Analytics in Agriculture: Benefits and Challenges	Elijah et al., (2018)	Malaysia and South Korea	238	59.50	IEEE Internet Of Things Journal	9.471	
8	Ability of a poor man's ensemble to predict the probability and distribution of precipitation	Ebert (2001)	Australia	235	11.19	Monthly Weather Review	3.735	
9	Clouds and the Earth's Radiant Energy System (CERES): Algorithm overview	Wielicki et al., (1998)	USA, Belgium, and France	215	8.96	IEEE Transactions on Geoscience and Remote Sensing	5.6	
10	Plant Phenomics, From Sensors to Knowledge	Tardieu et al., (2017)	France and England	195	39.00	Current Biology	10.834	

The 20 most productive authors are captured in Figure 2.7. The highest number of publications per author are only 5 out of 746. Therefore, there are no particularly active authors within this research subject. A large majority of the papers are multi-authored and there are 3413 total authors across the entire corpus. From the 3413, approximately 95% of the authors have only written one article.

Figure 2.7: Most productive authors (Last Name First Initial)

The institutions and journals with 10 or more publications are presented in Figures 2.8 and 2.9, respectively. The maximum number of publications made by any one institution is 31 by the Chinese Academy of Sciences. The most productive journal is Elsevier which has published 212 of the 746 articles (approximately 28%) which is far greater than the remaining journals and almost double the number of articles published by the journal in second place. It can be concluded that Elsevier is a key contributor within the field and studying the research evolution of this topic must include a review of the publications from Elsevier.

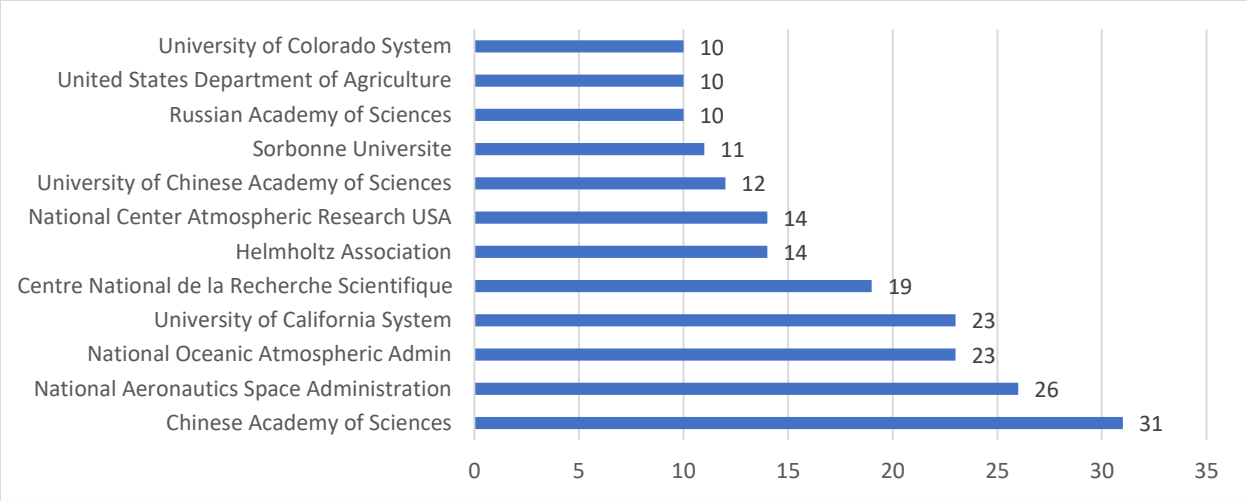


Figure 2.8: Most productive institutions

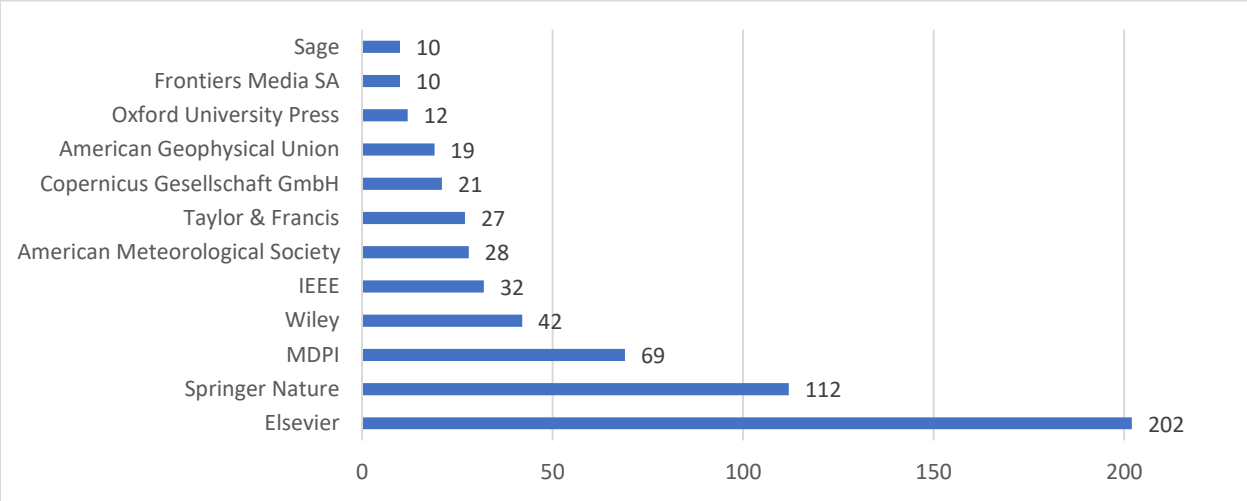


Figure 2.9: Most productive journals

The most productive countries, in order, include the USA, China, Italy, England, Germany, France, India, Spain, Canada, Australia, and Japan. The USA (28% of the articles) has published almost twice as much as China in second place. There are a total of 92 countries that have contributed within this research topic and a third of those countries have one publication between 1990-2021. The visualization of the most productive countries is summarized in Figure 2.10.

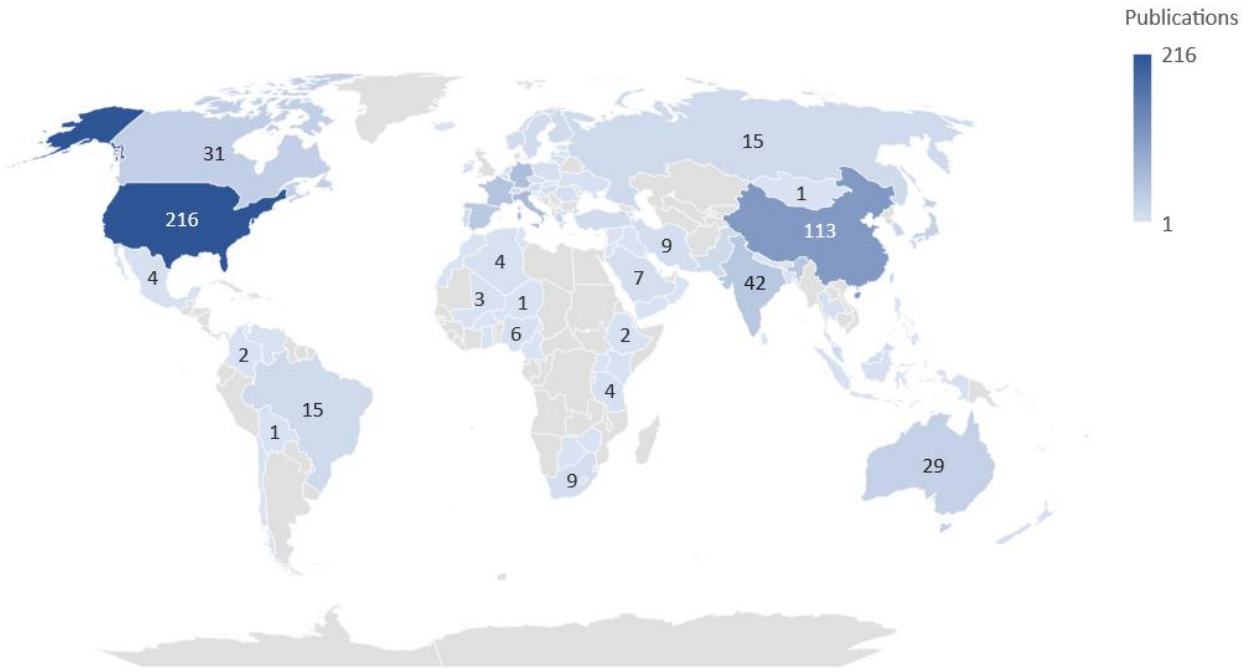


Figure 2.10: Most productive countries

The keyword co-occurrence network was developed using the VosViewer software. Co-occurrence network analyses allow the visualization of the occurrence of keywords (through the size of the node) while mapping its proximity to other keywords that appear in the same document (through the link). The results from Web of Science were exported to VosViewer and a map was generated based on the bibliographic data. To create the map, each phrase had to occur a minimum of 2 times. From a total of 2725 phrases, only 316 phrases met this requirement. These 316 phrases were pruned further to remove words from the original search query (such as data science, data analytics, climate change, weather, decision support, visual, tool, *etc.*). Additionally, any words that were repeated or were abbreviations of other phrases were also removed. There were 215 phrases remaining after this processing. Keyword co-occurrences can be visualized in 3 ways in VosViewer, 2 of those are included in Figure 2.11 and 2.12. The network display of keyword co-occurrence shows various clusters which helps to

indicate whether a particular domain has been researched. The overlay display additionally shows the evolution research clusters over time. This aids in identifying the areas that were being explored but are no longer relevant as well as the current trends. For example, in Figure 2.11, it can be seen that there are 9 main clusters. The red cluster (cluster 1) includes keywords like atmosphere, global positioning system, forecasts, space weather, and temperature which is meteorological terminology. The green cluster (cluster 2) includes terms such as trends, precipitation, wind speed, and vegetation. Although these are still meteorological terms, it seems that it is a separation of proximity between articles written about atmospheric climate such as temperature and radiation fluxes as compared to precipitation, rainfall, and vegetation. The third cluster in blue is focused on carbon dioxide emissions, energy consumption, electricity, greenhouse gas emissions, sustainability and renewable energy. The overall theme of this cluster is the impact of climate change on the energy supply sector and its relation to greenhouse gas emissions. The yellow cluster (cluster 4) has keywords such as precision farming, sensors, internet of things, *etc.* and is therefore very agriculture focussed. The blue (cluster 3) and yellow (cluster 4) clusters both represent major industries that are impacted by climate change and therefore it is logical that publications within climate change visualization tools would be made under these domains. This is similarly seen with the purple cluster (cluster 5) which includes the keywords: land, food security, public health, and risk and the light blue cluster (cluster 6) which includes the keywords: policy, ecosystems, water quality, river-basin, streamflow, and biomass. The remaining 3 clusters (orange, brown, and pink) are the smallest of the 9 clusters and include variations and/or synonyms of keywords from other clusters. A

more intensive pruning process would eliminate the last 3 clusters and further solidify the earlier established co-occurrences.

Figure 2.11 contains the same clusters and associations as Figure 2.12, however it presents a trend analysis for these clusters. Cluster 1 (previously the red cluster) and cluster 2 (previously the green cluster) which both represented the meteorological-themed research areas were actually the most researched in 2012-2014. The most recent trend within this field is focussed on impacts of climate change to various industries, organizations, and entities such as energy consumption, agriculture practices, food security, land-use practices, water resource management, *etc.*

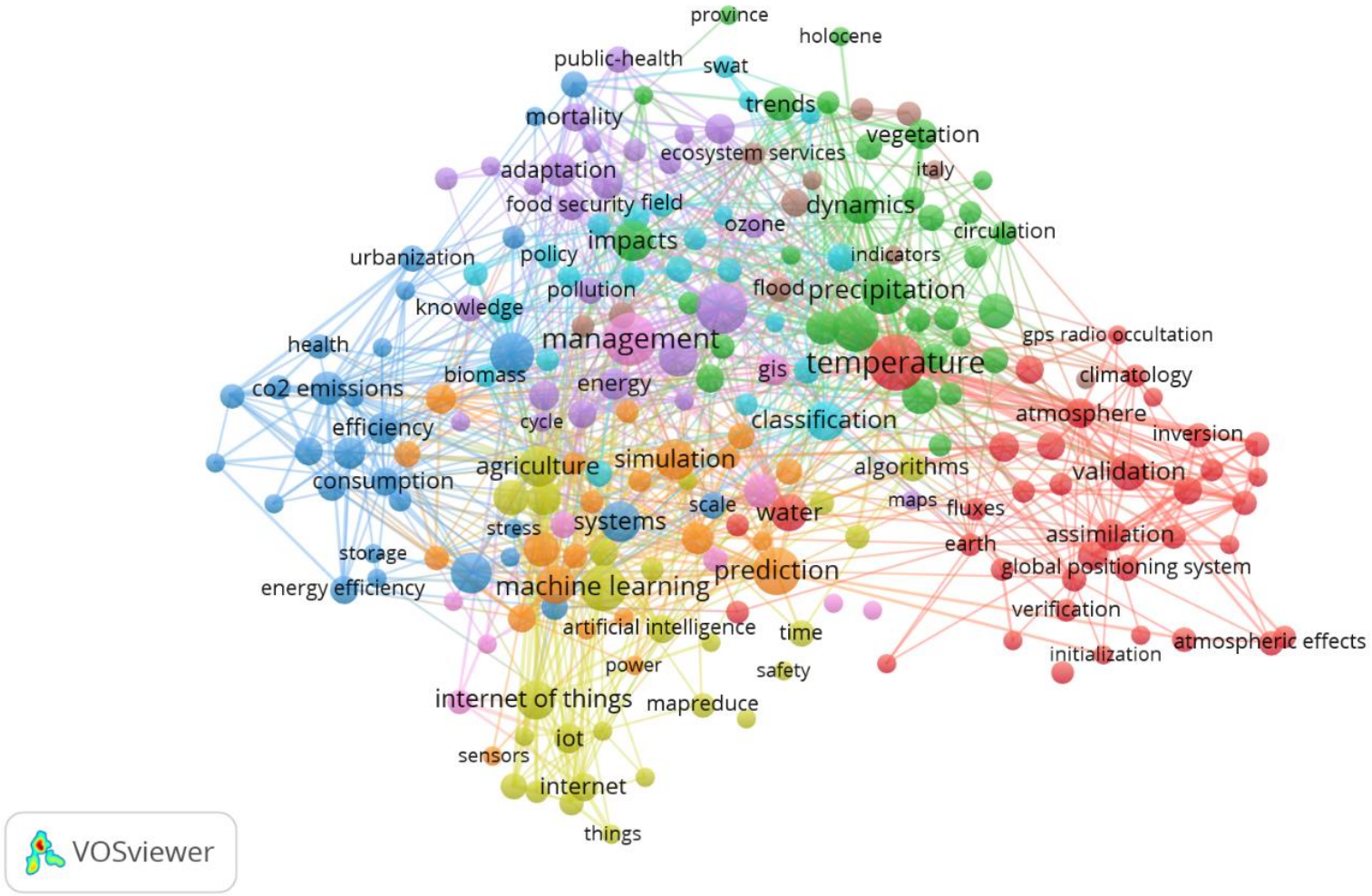


Figure 2.11: Keyword co-occurrence network

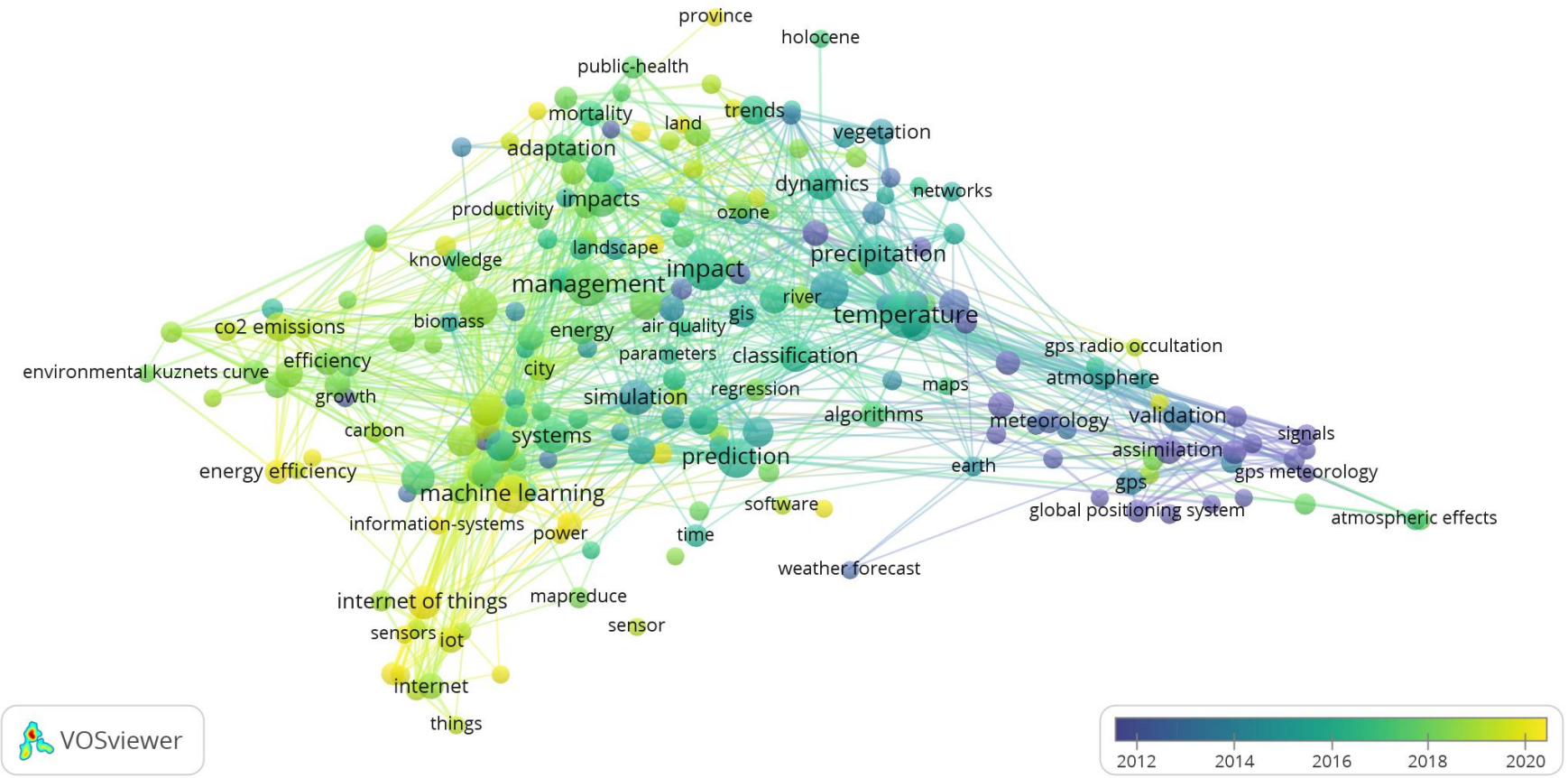


Figure 2.12: Keyword co-occurrence overlay

The abovementioned bibliometric methods are performance analysis measures that help outline the various contributions made in the research field. To position the development of a novel visual climatic analysis tool within this field, a "classification of relevance" was performed for all 746 documents in the corpus. The abstract of each article was skimmed to determine whether it was relevant to the thesis topic. An article was determined relevant if it mentioned performing comparative analysis between any 2 climate datasets or the visualization of climate data. After the first pass through, 18 articles were determined as relevant, 697 were determined as not relevant, and 31 articles needed further review. The 31 articles' abstract was read individually to analyze relevance and 19 articles were determined as relevant, 715 were determined as not relevant, and 12 articles still required review. For the remaining 12 articles, the full research paper was reviewed, and these were all determined as not relevant. Finally, only 19 articles were determined relevant (and 721 were not). The title, author(s), publication year, and journal title of these 19 articles is summarized in Table 2.5. Each of the articles was read in detail to review the research objective as it pertains to the "relevance" to this thesis (performs climatic comparative analysis or climatic visualization). Additionally, gaps in the literature are described in Table 2.5 to highlight the need for the tool presented in this work.

Table 2.5: Articles from bibliometric analysis related to thesis topic

#	Article Title	Author(s) (Year)	Journal Title	Objective of article as it pertains to thesis topic	Literature gaps
1	Predicting patterns of near-surface air temperature using empirical data	Anisimov (2001)	Springer	Data is collected from 2000 weather stations and GCMs to predict regional climate sensitivity.	Does not directly compare climate model and predicted data.
2	Software to Identify Climate Change Trends at the Local Level: A Study Case in Yucatan, Mexico	Bautista et al. (2013)	Universidad Autónoma Chapingo	Tool is created to identify local climate change trends based on agroclimatic data.	Is a information processing tool and does not have visualization and comparative capabilities.
3	Modeling the future impacts of climate change on water availability in the Karnali River Basin of Nepal Himalaya	Dahal et al. (2020)	Academic Press Inc., Elsevier Science	Analyzes hydro-climate data from CORDEX climate models for a specific region, uses the observed climate data as a baseline to produce a SWAT model.	Does not perform a comparative analysis between climate model and historical climate data.
4	Indian Ocean warming during 1958-2004 simulated by a climate system model and its mechanism	Dong et al. (2014)	Springer	Observed data and the climate system model, FGOALS-gl, are compared for sea surface temperature.	A comparative visualization tool is not developed.
5	Model-based climatology of diurnal variability in stratospheric ozone as a data analysis tool	Frith et al. (2020)	Copernicus Gesellschaft GmbH	Integrates various data sources to create climatological representation of diurnal variations.	Data integration of various data sources is performed to generate a model. The model/framework is not intended for visualization or comparison.
6	Evaluating the impact of climate change on stream flow: integrating GCM, hydraulic modelling and functional data analysis	Ghumman et al. (2020)	Springer Heidelberg	Precipitation and temperature data from three GCS are aggregate to evaluate climate change impact and perform streamflow analysis.	Does not perform a comparative analysis between climate model and historical climate data.
7	Applied Climate-Change Analysis: The Climate Wizard Tool	Girvetz et al. (2009)	Public Library Science	Provides visualization of precipitation and temperature trends historically and future predictions for any geographic region.	Does not provide comparative visualization for historical predictions. The climate variables are limited to precipitation and temperature.

#	Article Title	Author(s) (Year)	Journal Title	Objective of article as it pertains to thesis topic	Literature gaps
8	Web processing service for climate impact and extreme weather event analyses. Flyingpigeon (Version 1.0)	Hempelmann et al. (2018)	Pergamon-Elsevier Science Ltd.	A web processing service is developed to allow users to process large climate model data and perform a variety of analyses and visualizations.	Does not have the functionality to compare climate model and observed data directly.
9	Spatiotemporal climate model validation - Case studies for MM(5) over Northwestern Canada and Alaska	Herzfeld et al. (2007)	American Meteorological Society	Compares various climate models with various observed climate datasets to evaluate similarity.	Does not provide a tool for visualization.
10	The ANU translator: Facilitating computer visualization and data analysis of climate model outputs	Hyman et al. (1996)	Elsevier Sci Ltd.	Presents a translation tool to convert data from climate model format into readable format for visualizations.	Does not perform comparative analysis.
11	Climate-Smart Agro-Hydrological Model for a Large Scale Rice Irrigation Scheme in Malaysia	Ismail et al. (2020)	MDPI	Climate model data from 10 GCMs and daily climate data is collected and analyzed to develop a water resource management model for large-scale rice irrigation.	A VBA-based tool agro-hydrological tool is developed which does not visualization functions.
12	Online weather data analysis and visualization tools for applications in ecoinformatics	Jaroensutasinee et al. (2014)	Springer Heidelberg	Collects weather data from various weather stations and provides data analysis and visualization services.	Does not perform comparative analysis between climate models and observed climate.
13	Hypothesis Generation in Climate Research with Interactive Visual Data Exploration	Kehrer et al. (2008)	IEEE Computer Society	Climate model and reanalysis datasets are explored through interactive visualization.	Does not present a tool to allow external users to interactive visualization tool.
14	Regional Climate Model Evaluation System powered by Apache Open Climate Workbench v1.3.0: an enabling tool for facilitating regional climate studies	Lee et al. (2018)	Copernicus Gesellschaft GmbH	Provides a framework for comparing and visualizing climate model simulations with observed data to perform climate model evaluation.	The RCMES tool is not supported actively and has various features that are non-functional. Users must use command-line inputs to interact with the tool.

#	Article Title	Author(s) (Year)	Journal Title	Objective of article as it pertains to thesis topic	Literature gaps
15	Precipitable water vapour forecasting: a tool for optimizing IR observations at Roque de los Muchachos Observatory	Perez-Jordan et al. (2018)	Oxford University Press	Data validation is performed for the Weather Research and Forecasting model for precipitation water vapour by comparing model outputs with observed values.	A comparative visualization tool is not developed.
16	A 2000 year-long proxy and observational reconstruction of Central Asian climate	Rodda et al. (2019)	Pergamon-Elsevier Science Ltd.	Paleo-climate records and climate reanalyses are compared to make regional assessments on climate change impacts.	A comparative visualization tool is not developed.
17	Impact of cloud analysis on numerical weather prediction in the Galician region of Spain	Souto et al. (2003)	American Meteorological Society	The Advanced Regional Prediction System (ARPS) predictions are compared with local observations to explore precipitation processes.	A comparative visualization tool is not developed.
18	Assessment of Water Balance for Russian Subcatchment of Western Dvina River Using SWAT Model	Terskii et al. (2019)	Frontiers Media SA	Observed climate data and reanalysis data are integrated to assess water balance using the SWAT tool.	Reanalysis and observed climate data are integrated but not compared.
19	Web-system for processing and visualization of meteorological data for Siberian environment research	Titov et al. (2009)	Taylor & Francis Ltd.	Presents web interface to perform data processing and visualization of reanalysis datasets.	Comparative visualization analysis functionality is provided but there are limited datasets and variables available.

Step 5: Evaluate the results

From the 746 articles reviewed as part of the bibliometric analysis, only 19 were determined to have similar research topics and contributions as the ones presented in this thesis. A summary of the gaps of the 19 research articles are in Table 2.6.

Table 2.6: Summary of literature gaps

Literature gap	**Number of articles**
Performs comparative analysis but it is not between climate model data and observed climate data, or Historical data and climate model is collected but is not compared.	6
Performs comparative analysis but no visualization tool.	2
Model/tool is developed but it does not have visualization or comparative analysis functionalities.	6
Visualization of comparative analyses are presented but a tool is not developed.	2
A visualization tool is developed but comparison of 2 or more datasets cannot be performed.	1
A tool is developed for visual climatic analysis but is not user-friendly, supported actively, or has limited functionalities.	2

From the summary, it is evident that no research paper has developed a tool that provides the ability to compare and visualize climate model data with observed data. The tool presented in this thesis can be used simply in the meteorological domain to assess climate datasets but can also be expanded to integrate, compare, and visualize other datasets relevant to agriculture, energy, land use, *etc.* as is the current trend in climate change research.

Chapter 3: Research Methodology

The standard data analytics framework can be represented through a 6-step process, requirements gathering, data collection, data extraction, data wrangling, data analysis, and data visualization, as shown in Figure 3.1. However, working with climate data poses unique challenges. Each climate data source collects, documents, and presents its data in a different way requiring thoughtful considerations in each step of the analytics process. Ensuring that the comparison of 2 sets of data is "apples to apples" requires standardization across all criteria in the scope. For this work, the criteria consisted of the geographic region, year range, data granularity, variables in consideration, and spatial resolution of the climate model.

Step 1 of the framework, requirements gathering, has been discussed thoroughly in the literature review. Steps 2-6 will be discussed in the following subsections of this chapter and can be mapped to the research contributions (RCs) 3-6 as outlined in the introduction.

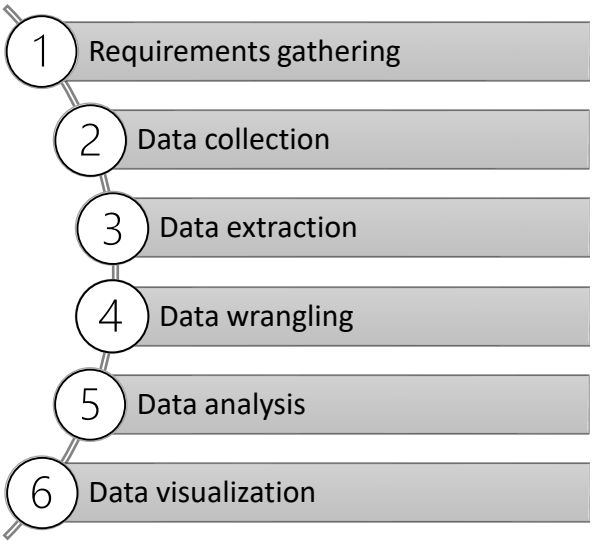


Figure 3.1: Conventional data analytics framework

3.1: Data Collection

The process of data collection can only occur once the objective of the analysis has been established. In this case, the overall objective was to perform a comparative analysis between predicted and observed climate data for a specific year range and region. To add specificity to this objective, the scope under consideration had to be narrowed while conducting the data collection process.

3.1.1: Geographic Region

The first task of data collection was choosing a region to perform analysis for, as that would focus the search for sources of data. To aid with understanding the data better due to familiarity, Toronto was chosen as the region of analysis. Additionally, it was presumed that Toronto as a major city within Ontario would have a greater variety of available data sources.

3.1.2: Date Range

The year range 1986-2005 was selected to focus the comparative analysis. A wide range of years was chosen to gather sufficient data to make suitable comparisons. A historical range was chosen to increase the availability of data.

3.1.3: Data Granularity

Daily data granularity was chosen for the observed and predicted datasets to reduce loss of information when performing comparisons. Furthermore, if data was aggregated to a monthly or annual visualization and a trend required investigation, the data would have the capacity to show detailed values. A daily granularity would also allow for a day-by-day calculation of the difference in values between the climate model's projection and the observed climate value.

3.1.4: Climate Variables

The next task in the data collection was the selection of climate variables of interest which had to be those within the essential climate variables' definition (as ECVs are used as the baseline to measure climate variability) (Bojinski et al., 2014). To narrow the scope of this research, the surface atmospheric climate variables were considered feasible to explore for the comparative analysis.

For most of the observed data, Weather Stats was chosen as the data source since it derives its information from Environment and Climate Change Canada by integrating data from two Toronto-based climate stations (Weather Stats, 2021). The integration performed by this platform helps to ensure that the data is complete to a high degree while maintaining its reliability by using reputable data sources. There were 69 climate variables available in the Weather Stats platform of which 28 variables did not have any data recorded for Toronto. The remaining data was various measures of temperature, windchill, relative humidity, dew point, wind speed, wind gust, sea pressure, station pressure, visibility, degree days, precipitation, rain, and snow. Since the solar radiation variable had poor data quality from the Weather Stats source, data from NASA's Prediction of Worldwide Energy Resources (POWER) project was collected (POWER, 2021). Their Earth Science research program provides data for the study of climate and climate processes particularly for radiation variables. The data has been shown to be sufficiently accurate to provide reliable solar data over regions where surface measurements are sparse. The data from the POWER project has 10 radiation variables that can be selected, and all the variables had 100% completeness for the Toronto region. The subsequent aspect of this process was mapping the available

observed climate variables with ones in the climate model. The chosen climate model was the CanRCM4 model (CCCMA, 2020). This model is part of the Coordinated Regional Climate Downscaling experiment (CORDEX) responsible for advancing regional climate downscaling models. The NAM version was chosen because it contained data for all North America from which Toronto data could be extracted. The CanRCM4 is derived from the ESM2 global model developed by the Canadian Centre for Climate Modelling and Analysis. The CanRCM4 model contains over 50 atmospheric climate variables. However, there were only 8 variables that mapped equally to the abovementioned 28 variables from Weather Stats and 10 radiation variables from the POWER data repository. The summary of the selected variables and mapping between observed and predicted datasets is presented in Figure 3.2.

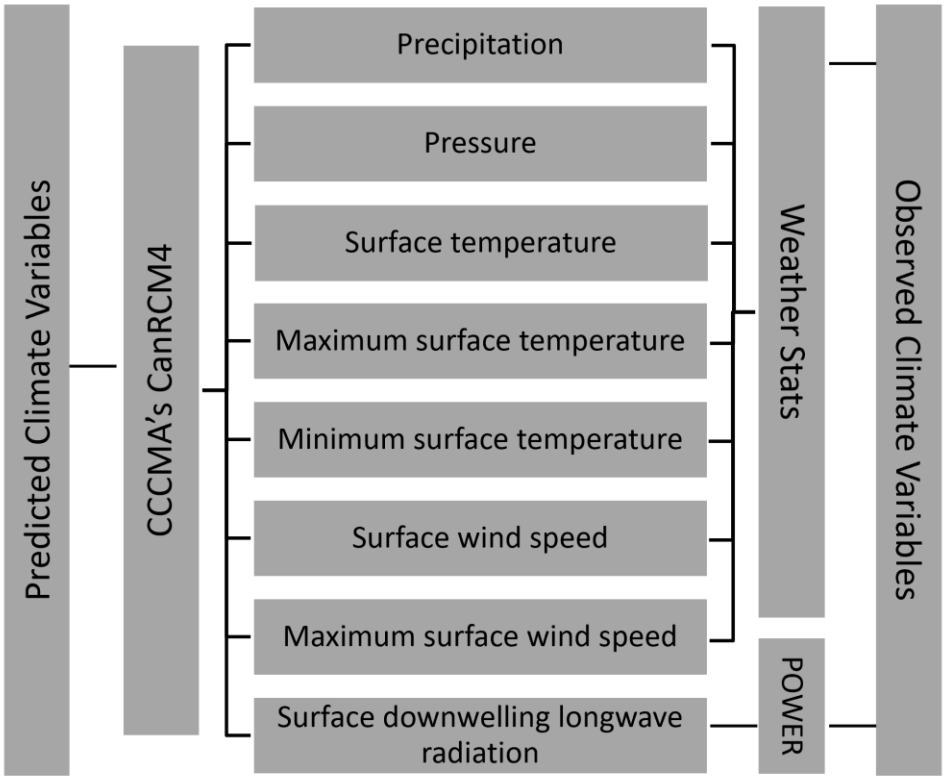


Figure 3.2: Summary of variables in scope for comparison

3.1.5: Spatial Resolution

For the chosen climate model, CanRCM4, there were two options for spatial resolution, 50km x 50km or 25km x 25km grid tiles. The larger spatial resolution would cover the area of Toronto within less grid tiles and therefore generate less data and require less computational resources. However, comparing the model data with a climate station situated in a specific location within the 50km^2 square would be less accurate. Therefore, the 25km x 25km grid resolution was chosen to make the data more comparable.

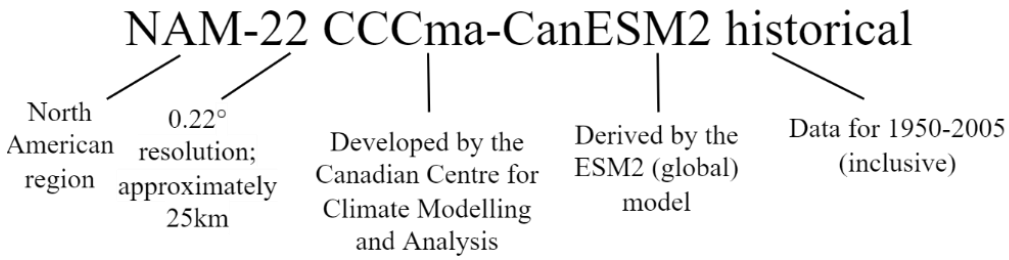


Figure 3.3: Summary of attributes of chosen climate model, CanRCM4

3.1.6: Summary of the Scope

The scope thus established the inclusion criteria relevant to perform the comparison between the two datasets, as summarized in Table 3.1.

Table 3.1: Summary of inclusion criteria

Attribute	**Inclusion Criteria**
Region	Toronto, Canada
Year range	1986 – 2005
Data granularity	Daily
Climate variables	8, as per Figure 3.2
Spatial resolution	25km x 25km grid tile

The last step of the data collection process was downloading the data from each source. For the Weather Stats dataset, there was a simple data interface to download data in a csv format, given inputs for the year range of interest. Once downloaded, the raw Weather Stats data was in a tabular format including all 69 climate variables, and one field for the date in YYYY-MM-DD format, with one record per day from 1986-01-01 to 2005-12-31.

For the POWER dataset, the inputs required the specific latitude and longitude of the location specified. The latitude and longitude coordinates for Toronto, if narrowed down to one point are 43.6532, -79.3832 respectively. The climate station data from Weather Stats does not have the same coordinates as the single point coordinates for Toronto and is instead near the Toronto Pearson International Airport. The input for the POWER dataset was entered as the same coordinates as the Weather Stats data to maintain consistency, pictured in Figure 3.4.

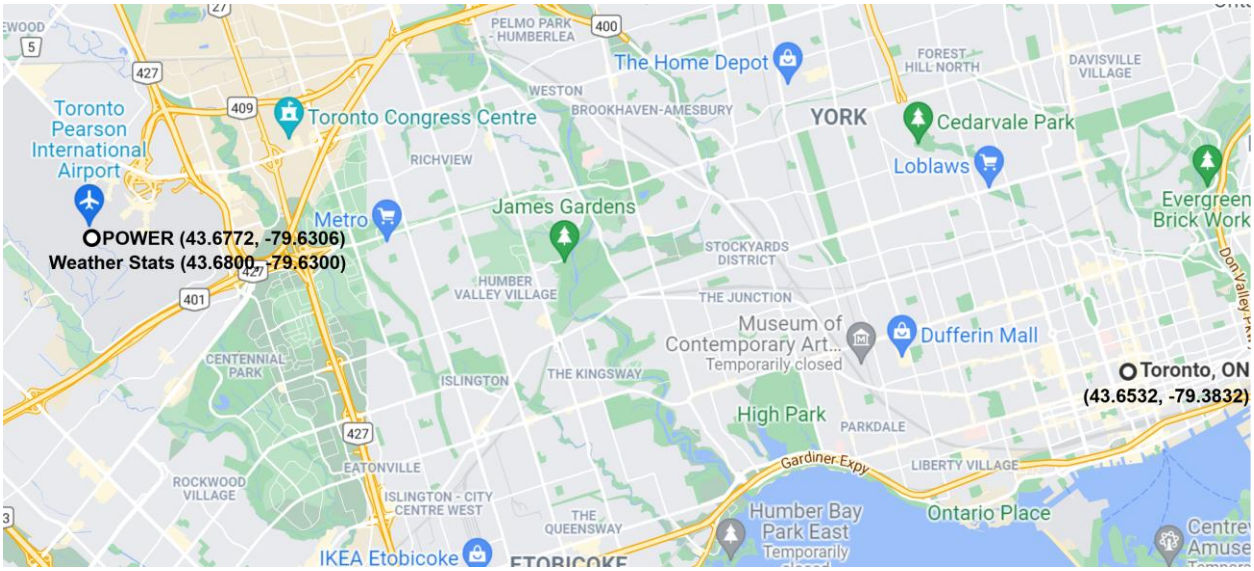


Figure 3.4: Coordinates' comparison for Toronto and observed climate datasets

Although the coordinates were not the same, it is important to note that the larger bounding box for Toronto would include the data source location. The bounding box for Toronto is shown in Figure 3.5. Similar to the Weather Stats dataset, once download the POWER dataset was in a tabular format as a csv with 1 climate variable (radiation) and 3 fields for the date format with year, month, and day. There were a few rows of meta-data provided at the top of the spreadsheet; these were deleted to prepare the file for the next step in the data pipeline.

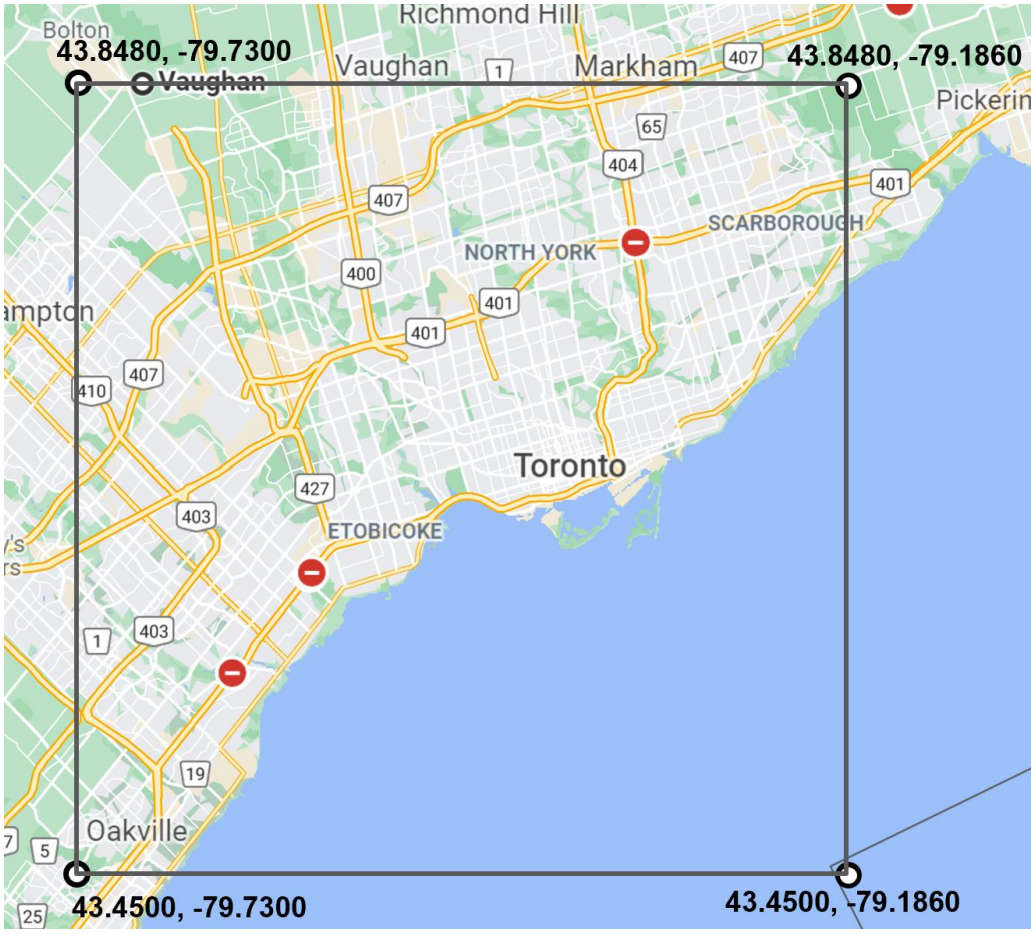


Figure 3.5: Bounding box for Toronto, Ontario

The climate model data was available for download in .netcdf format. Furthermore, each variable's data was a separate file which contained climate records

for a 5-year period. As a result, to collect data for 8 variables across a 20-year time period, a total of 32 .netcdf files were downloaded, each with a size of over 0.5gb.

3.2: Data Extraction

The data extraction portion for this work was only relevant to the climate model dataset since the Weather Stats and POWER datasets were already in tabular Excel spreadsheet format with one day's climate values per line (i.e., one record per line). Therefore for the climate model dataset, the extraction process had to end with a tabular dataset describing the climate predictions for Toronto for 8 variables as one record per line. The list of tasks performed to achieve this are described in detail below.

3.2.1: Convert .netcdf files to Excel Files

NetCDF (network Common Data Form) is a file format that stores scientific data as an array with each dimension as a separate layer. Each of these layers can be individually converted to a tabular format using a software like Panoply Data Viewer (Panoply, 2021). The dimensions of interest for extraction were time and the climate variable data. Since the 4 time-related files contained consistent data across the variables, they only needed to be converted to excel once (rather than for each climate variable). However, each of the 32 .netcdf files' climate variable data had to be converted to Excel one-by-one since they contained unique information.

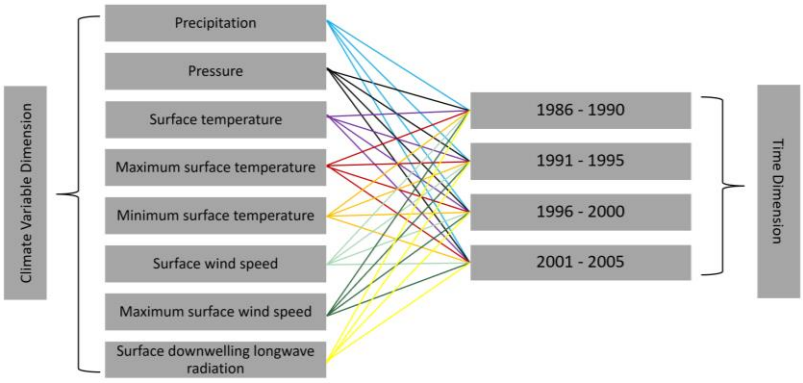


Figure 3.6: Combination of 32 files downloaded for climate model

3.2.2: Determine Grid Tile Corresponding to Region

As mentioned in the literature review, climate models predict data for each grid tile within a region (North America in this example). To extract data for Toronto, it was important to narrow down the one grid tile that would be closest to the climate station locations of the observed climate datasets. However, CORDEX files provide location data as rotated latitude and longitude (latitude and longitude with the north and south pole rotated). Therefore, the latitude and longitude of the observed datasets had to be converted to rotated latitude and longitude first before deriving the grid tile values. The online tool, Rotation of Coordinates Based On CORDEX Domains, was used for this task (Kolsoumi & Salehnia, 2019). The inputs entered in this tool were: the CORDEX domain as North America and the latitude and longitude for the POWER dataset. The rotated latitude and longitude for Toronto was -2.43, 12.48. Using these outputs, the grid tile of interest could be deduced. To do this, Panoply was used again to generate options for the closest rotated latitude and longitude for Toronto. It is important to note that the coordinate data for the exact same location could not be deduced due to the non-continuous format the grid tile structure of the CORDEX files. In Panoply, the Create Plot function allows users to visualize data through different types of plots and investigate the underlying values. Using this feature, a georeferenced longitude-latitude colour contour plot was generated for both of the latitude and longitude variables. The data tables of these plots are shown in Figure 3.7 and 3.8.

Figure 3.7: Georeferenced longitude-latitude colour contour plot for latitude

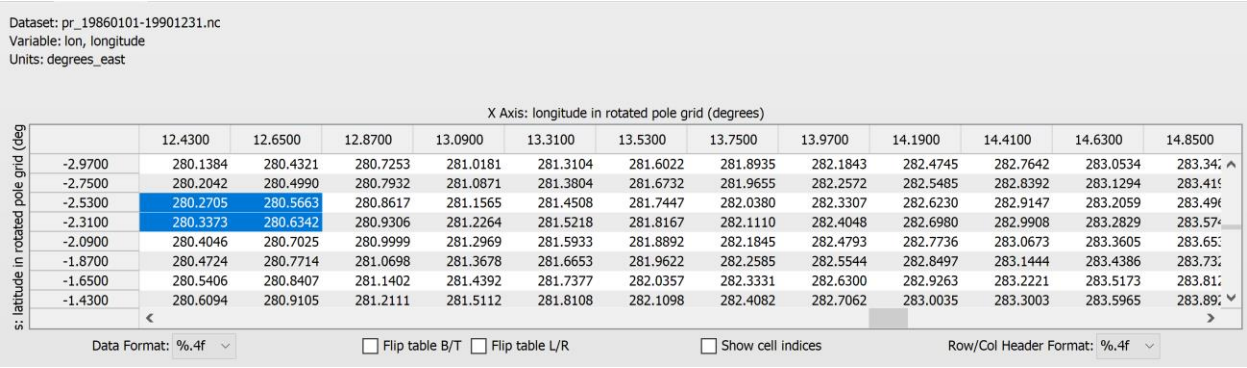


Figure 3.8: Georeferenced longitude-latitude colour contour plot for longitude

As shown in these figures, the y axis contains values for the rotated latitude and x axis contains values for the rotated longitude. The intersection point of the axes is the value of latitude (in Figure 3.7) and longitude (in Figure 3.8). Comparing these to the calculated rotated latitude and longitude for Toronto showed that there were a few options close to the location of the observed datasets. These options are presented in Table 3.2.

Table 3.2: Latitude longitude options within CORDEX that are close to observed climate stations

Rotated Latitude / Rotated Longitude	**12.43**	**12.65**
-2.53	43.5883, -79.7295	43.5397, -79.4337

-2.31	43.8029, -79.6627	43.7542, -79.3658

To select the most suitable set of coordinates, the POWER dataset coordinates and each of the 4 options were plugged into mapping software to determine the pair of coordinates that were within 25 km of distance of each other (as this was the spatial resolution of each grid tile. Additionally, the chosen location had to be within the bounding box of Toronto. Of the four options, the only set of coordinates that fulfilled these conditions was 43.5883, -79.7295 which mapped to -2.53, 12.43 in rotated latitude and longitude. The last step was to use this information to derive the grid tile values for Toronto. This was done using the Panoply line plot with time for the horizontal axis feature. The rotated latitude and longitude values were plugged into the dropdown selections which generated the grid cell value, as seen in Figure 3.9.

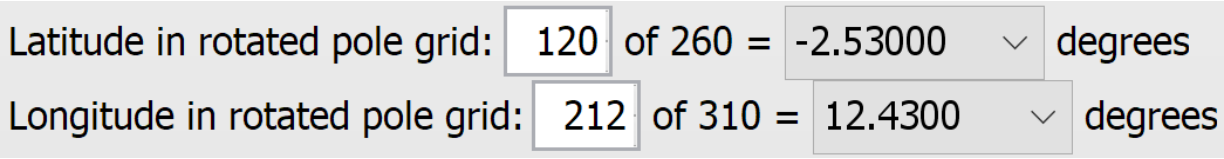


Figure 3.9: Grid tile value according to rotated latitude and longitude

3.2.3: Extract Region-specific Data Based on Grid Tile

The raw version of the climate model in csv files contained the entire data for North America. Each cell in the spreadsheet contained the value of the climate variable for the location intersection of a grid tile. The tabular file had 310 total columns (indicating longitude as a grid value) and 260 total rows (indicating latitude as a grid value). Therefore, it was understood that the entire North America grid in this model was made up of a 260 x 310 grid where each grid tile is 25km x 25km. The grid specific to Toronto was 120, 212 which corresponds the cell address HD120. The contents this cell

address contained the climate value for the first date in the year range of the file. For example, cell HD120 for the file surface temperature in the year 2001-2005 would contain the temperature on 2001-01-01. Each file had 1825 (365 days * 5 years) 260x310 matrices. Each of the 1825 represented 1 day of data for every tile in the North America grid. The "HD120" or 120, 212 intersection of each matrix held data for Toronto that had to be extracted. Using the grid tile values allowed for extraction for Toronto, Ontario programmatically using Python. Two files were required for input, one for the climate variable dimension and another for the time dimension. Therefore using Python, the grid cell intersection was provided as input to loop through the spreadsheet and extract 1825 instances of data into a pandas dataframe. The next step was loading the data from the time spreadsheet. However, the units for time from the climate model were given as "days since 1949-12-01". This had to be converted by adding the number of days indicated in the time field to 1949-12-01 to generate the date in YYYY-MM-DD format. Since the inbuilt Python function used to perform this calculation, pandas to_datetime, accounts for leap years while the CORDEX data ignores them, it was important to offset the time variable accordingly. With this last step performed, the goal of establishing a tabular dataset describing the climate predictions for Toronto for each variables as one record per line was achieved, as pictured in Figure 3.10.

	Date	Variable	Date_2
0	18646.5	266.12506	2001-01-01
1	18647.5	267.93008	2001-01-02
2	18648.5	270.53894	2001-01-03
3	18649.5	276.02680	2001-01-04
4	18650.5	272.79110	2001-01-05
...	...	...	...
1820	20466.5	272.24738	2005-12-27
1821	20467.5	272.01850	2005-12-28
1822	20468.5	274.22223	2005-12-29
1823	20469.5	273.85046	2005-12-30
1824	20470.5	273.55300	2005-12-31

Figure 3.10: Sample dataframe output from Python

The data extraction steps are summarized in Figure 3.11.

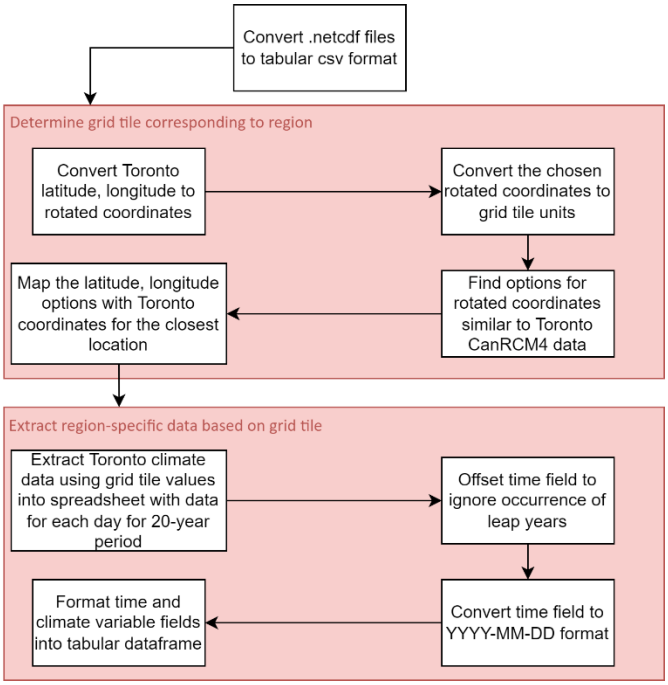


Figure 3.11: Summary of data extraction steps

3.3: Data Wrangling

3.3.1: Data Cleaning of Observed Climate Datasets

The next phase of the data pipeline involved data wrangling. While the climate model data was specifically downloaded for the 8 variables of interest, the POWER and Weather Stats datasets contained all possible variables that the source hosted. Therefore, the first step was dropping all other fields other than the 8 climate variables of interest and the time field. Additionally, there was no leap year data in the climate model dataset, the records holding climate data for leap years in the observed datasets were dropped. The last step of the extraction process established 32 Excel files for the climate model data with 2 attributes (time, climate variable value). There were 4 files per the (8) variables, each containing 5 years of data. To reduce the number of files that had to be worked with, the data was consolidated into 1 file per variable (programmatically using Python) holding the data across 1986-2005 for a total of 8 climate model datasets.

3.3.2: Data Quality

One of the major steps of the data wrangling phase is checking the data quality of the datasets being used. While data quality is multi-dimensional, the only attribute of interest for this comparative analysis was completeness. Given that a climate model is a mathematical derivation of climate predictions, there were no completeness-related issues. However, the observed climate datasets had to be investigated to determine its data quality, this was performed programmatically in Python using the isnull() function. The POWER dataset (containing 1 radiation variable) had 100% completeness and the

Weather Stats had less than 100% completeness for only the precipitation variable. The summary of the data quality check is summarized in Table 3.3.

Table 3.3: Data completeness of the observed precipitation variable

Year	**Precipitation (total number missing annually)**	**Percentage (annual)**
1992	5	1.37%
1993	55	15.07%

To fill the missing records of precipitation, records from nearby climate stations within 25km of the Toronto climate station were used (Egigu, 2020; Sattari et al., 2017). Since the data availability was not consistent with just one additional climate station, two stations (Brampton MOE and Lakeview MOE) had to be used to fill in the data. The first step in this process was downloading the daily data report for the Brampton MOE or Lakeview MOE station for the months with missing data in 1992 and 1993. Then, climate variables that were not needed and dates that were not blank in the Toronto dataset was deleted. At the end, there was one Excel file with the specific dates that had missing data with the precipitation value from a nearby climate station. Then the Toronto dataset and the dataset with the values to fill in were read into Python. A loop was created to go through the Toronto dataset and fill the data from the secondary dataset if the precipitation was null, while checking the equivalency of date in both datasets prior to filling the value in.

3.3.3: Unit Conversion

One aspect of ensuring that the comparative analysis between the observed and predicted datasets is "apples to apples" is to ensure that the comparison is being made

between the same units for each climate variable. Since the units of the observed datasets are more readable and familiar in terms of how weather information is consumed by any user, the climate model variables' units were converted to match those of the observed dataset. The units of the variables in each dataset is summarized in Table 3.4.

Table 3.4: Climate variables' units in each dataset

Climate variable	**Units in Predicted Data**	**Units in Observed Data**
Precipitation	$\frac{kg^2 \times m^2}{s^2}$	$\frac{mm}{day}$
Pressure	Pa	kPa
Surface temperature	K	℃
Maximum surface temperature	K	℃
Minimum surface temperature	K	℃
Surface wind speed	$\frac{m}{s}$	$\frac{km}{h}$
Maximum surface wind speed	$\frac{m}{s}$	$\frac{km}{h}$
Surface downwelling longwave radiation	$\frac{W}{m^2}$	$\frac{W}{m^2}$

3.3.4: Data Model

The final step in the data wrangling process was creating a data model to serve as the basis for analysis and visualization. The data model represents the final, cleaned version of the dataset to which no further modifications are made. In this case, there were 8 data models, one for each climate variable. Python was used to read each of the observed climate variable and map it to the same variable in the climate model dataset,

with the date field. Therefore, the final data model for each variable consisted of the date field, the predicted climate variable value, and the observed climate variable.

	Date	Observedpr	Modelpr
0	1986-01-01	0.0000	12.3433
1	1986-01-02	0.0000	0.0038
2	1986-01-03	1.4000	0.0027
3	1986-01-04	0.8000	0.0000
4	1986-01-05	0.2000	0.0010
...	...	...	...
7295	2005-12-27	0.0000	0.0000
7296	2005-12-28	7.0000	0.0032
7297	2005-12-29	2.8000	3.4819
7298	2005-12-30	0.0000	0.7093
7299	2005-12-31	3.8000	11.2188

[7300 rows x 3 columns]

Figure 3.12: Data model for precipitation variable

The data wrangling steps are summarized in Figure 3.13.

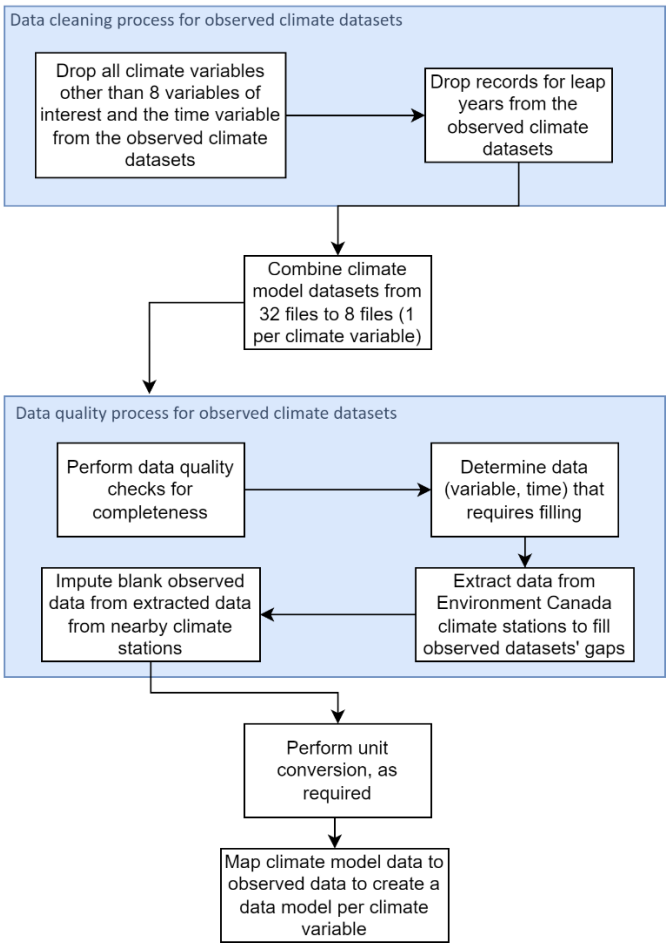


Figure 3.13: Summary of data wrangling steps

3.4: Data Analysis

The purpose of this thesis to develop a framework for comparing climate model data with observed climate data. The framework is implemented a visualization tool which also allows researchers to download the data models. However to provide additional aid to researchers, the tool also contains data tables and additional visualizations to perform exploratory data analysis. The inclusion of EDA can also be used as a method to discover potential areas that require further analysis or provoke research questions around the interpretation of the data. While the subsequent Case Studies chapter will go over examples of analyses output and how they can be interpreted, this subsection focuses on the selection of the data analysis techniques, as summarized in Table 3.5.

Table 3.5: Summary of analyses per climate variable

Climate Variable: Precipitation	
Sum	Total monthly (data table) Total annual (data table)
Central Tendency	Monthly mean (data table) Monthly median (data table)
Variation	Monthly standard deviation (data table)
Trends	Monthly 5 number-summary and boxplot (data table and visualization) Annual 5 number-summary and boxplot (data table and visualization)
Seasonality	Seasonality graph
Bias	Number of instances where the bias is $\pm$ 1mm (data table) Line graph of observed precipitation with bias as secondary line graph with area (visualization)
Climate Variable: Surface Pressure	
Central Tendency	Monthly mean (data table) Monthly median (data table)
Variation	Monthly standard deviation (data table)

Trends	Monthly 5 number-summary and boxplot (data table and visualization) Annual 5 number-summary and boxplot (data table and visualization)
Seasonality	Seasonality graph
Bias	Number of instances where the bias is $\pm$ 0.5kPa (data table) Line graph of observed precipitation with bias as secondary line graph with area (visualization)
Climate Variable: Surface Temperature, Maximum/Minimum Surface Temperature	
Central Tendency	Monthly mean (data table) Monthly median (data table)
Variation	Monthly standard deviation (data table)
Trends	Monthly 5 number-summary and boxplot (data table and visualization) Annual 5 number-summary and boxplot (data table and visualization)
Seasonality	Seasonality graph
Bias	Number of instances where the bias is $\pm$ 1℃ (data table) Line graph of observed precipitation with bias as secondary line graph with area (visualization)
Climate Variable: Wind Speed, Maximum Wind Speed	
Central Tendency	Monthly mean (data table) Monthly median (data table)
Variation	Monthly standard deviation (data table)
Trends	Monthly 5 number-summary and boxplot (data table and visualization) Annual 5 number-summary and boxplot (data table and visualization)
Seasonality	Seasonality graph
Bias	Number of instances where the bias is $\pm$ 5km/h (data table) Line graph of observed precipitation with bias as secondary line graph with area (visualization)
Climate Variable: Surface Downwelling Longwave Radiation	
Central Tendency	Monthly mean (data table) Monthly median (data table)
Variation	Monthly standard deviation (data table)
Trends	Monthly 5 number-summary and boxplot (data table and visualization)

	Annual 5 number-summary and boxplot (data table and visualization)
Seasonality	Seasonality graph
Bias	Line graph of observed precipitation with bias as secondary line graph with area (visualization)

The measure sum, used only for precipitation, was created because a large majority of the days incur no precipitation. Therefore, to make any comparison between the predicted and observed data, the values have to be aggregated. Both monthly mean and monthly median were considered important measures of central tendency. Having both measures would give a better idea of the data distribution and whether the data is skewed. If there any many outliers for a particular climate variable, the median measure would demonstrate the true central tendency without adverse impact. Standard deviation was chosen as the measure of variation because it shows the dispersion of the data relative to the mean. A low monthly standard deviation would indicate that the values for the climate variable were very close to the mean value. Therefore, the combination of viewing the mean and standard deviation of a climate variable for a particular month would a very quick way to get an understanding of the data. The 5-number summary and its visual partner, boxplots, were chosen to understand trends over time. A monthly boxplot shows the entire range of climate variable within a month but comparing it to adjacent months also helps determine trends over time. For understanding climate change patterns, comparing the boxplots from 1986 with 2005 may help glean some insights. Some variables have seasonality, like temperature, while others do not such as precipitation (in a non-monsoon region such as Toronto). However, the visualization was included because it helps in understanding whether there are any distinct patterns within a year and whether those patterns are shifting

year-over-year. Lastly, the visualization of bias was created since it is a fast way to consume whether the predicted climate data was similar to the observed climate.

3.5: Data Visualization

Data visualization is an important aspect for all data analytics projects as the visual summary of data allows users to understand the data, identify patterns, and comprehend the larger context of the data in an efficient manner. In this thesis, an interactive tool serves as the implementation of the data analytics framework to compare two climate datasets and is also a method for visualizing the resulting data. The visualization tool, Weather Analysis Regional Model (WARM), contains various useful features. Primarily, it allows users to compare the predicted and observed climate data for Toronto for any date range between 1986-2005. Additionally, users can perform a comparative day-by-day analysis through a time series graph and view the results of a detailed EDA. Users also have the ability to download the underlying data model, along with all the data for the EDA.

3.5.1: Home Page

The WARM tool was created using Dash which is a Python framework created by Plotly. Dash allows the development of data visualization interfaces through the use of Python code exclusively. There are two main libraries required to generate the layout of the application interface. The first is dash_html_components which uses HTML tags, keywords, and attributes to generate the basic layout of the webpage. The second library, dash_core_components (DCC) is used to create controls and graphs.

The layout of the WARM tool was designed to represent a dashboard experience. The home page, pictured in Figure 3.14, is mostly static with a few logos,

design elements, and an explanation of the visualization tool. The vertical panel on the left represents buttons that users can click on to navigate to different pages within the web application.

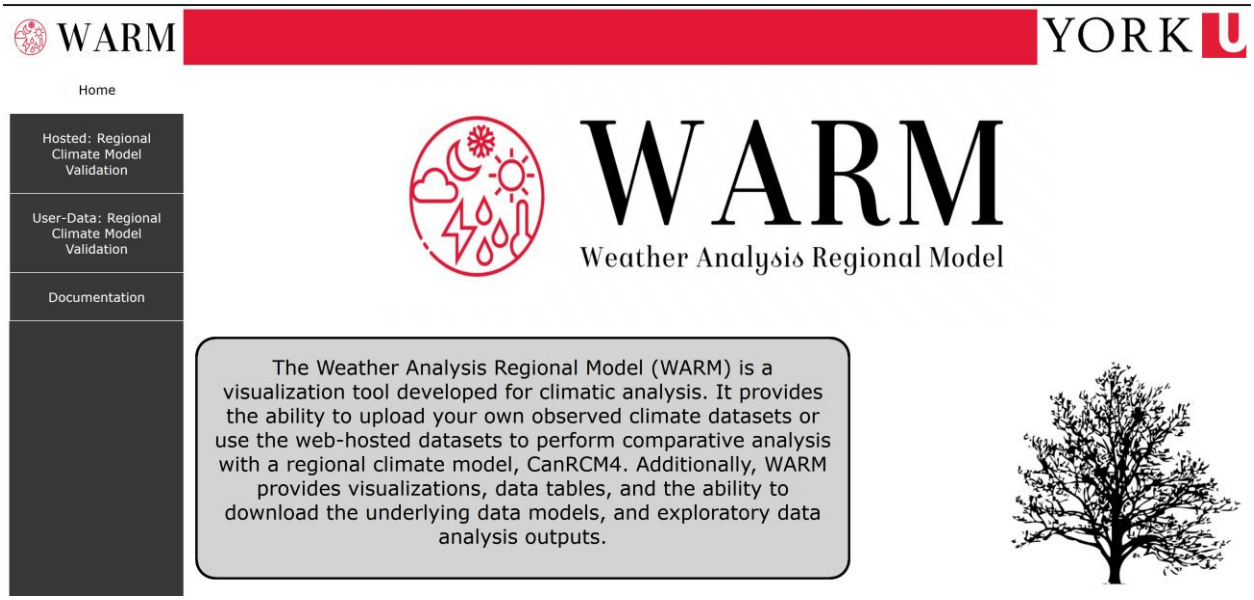


Figure 3.14: Home page of WARM tool

Each button on the vertical panel operates through the use of callback functions. Callback functions facilitate user interaction by collecting user input (properties) on an input component (the buttons on the vertical panel) and displaying an output (navigating to a different page) when the properties are changed (clicking on a button). From the home page, the user can navigate to any of the 3 other pages. The following subsections discuss the functionality of each page, design components, elements the user can interact with, and relevant programmatic details. The details of the User-Data: Regional Climate Model Validation page will be discussed in the Case Studies chapter.

3.5.2: Hosted: Regional Climate Model Validation

The Hosted: Regional Climate Model Validation page of the tool allows users to visualize and download the comparison climate data for Toronto. There were 5 visual

components created in this page. The first component was a continuation of the static space occupied by the vertical navigation panel. Although it is evident that clicking on the button generates a new page, the background colour of the button was changed to provide feedback on click. The second static space created in this page was the top panel with the logos to maintain consistency in the design. The third component that was added is the instructions panel to provide details on how to interact with the form. The fourth component developed was the form which the user interacts with. The first 2 fields were designed with dropdowns for which the user can choose the region (only Toronto is available) and the variable (8 options). The third field was designed with 3 parts; the first is a slider that allows the user to select a year range of interest. This was added so that the user did not have to click the left/right button repeatedly to navigate through multiple years one month at-a-time in the calendar widget. After selecting the year range, the calendar widget was added so users could specify the months or dates of interest. Lastly, a text output of the date range selected by the user was provided to allow the user to make any corrections if needed. The submit button at the end of the form initiates the creation of the various visualizations and data tables as outputs. It also automatically switches the view to the next tab within the Regional Climate Model Validation page called "Analyze", which is component 5. Since users cannot navigate to the Analyze tab (other than by filling the form and clicking submit), the Analyze tab was made grey to indicate it is not available for interaction. The design of the Load tab is in Figure 3.15.

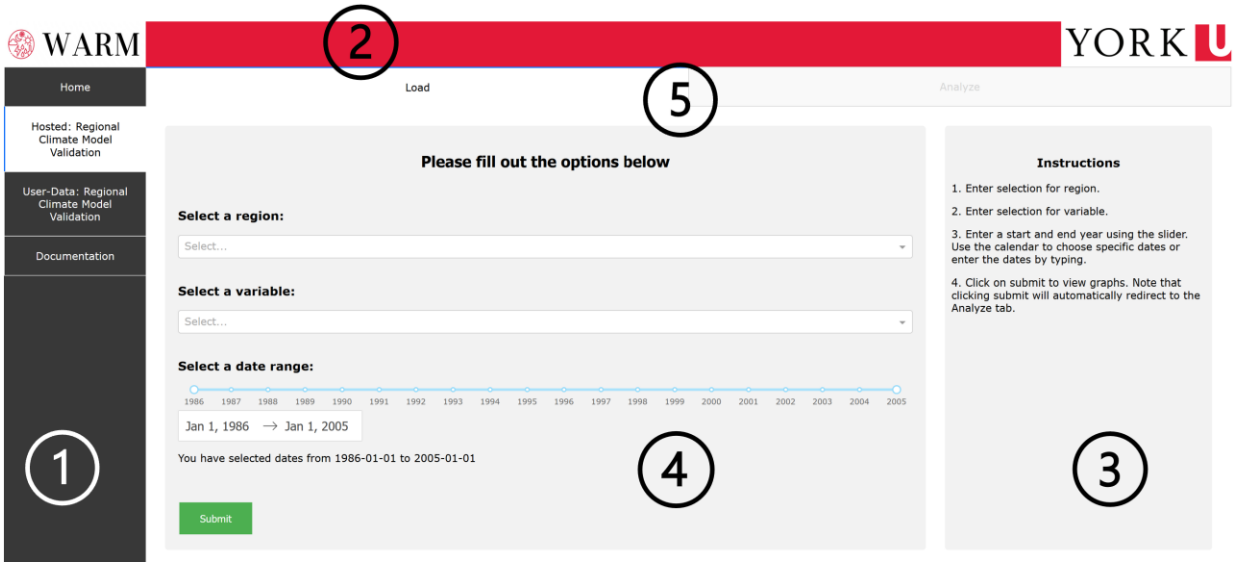


Figure 3.15: Layout and components of the Load tab of Hosted: Regional Climate Model Validation page

For the outputs to be rendered, the code processes the input by using a callback function to save each of the user inputs into local variables. Depending on the climate variable selected by the user, the corresponding csv file with the data model is loaded into a pandas dataframe. Using the user's date range input, the dataframe may need modification to keep only the data that is within the required date range. Subsequently, the primary visualization (time series graph) is created using the data model. For the exploratory data analysis, the original data model is manipulated to generate 5-number summaries, monthly means, *etc.* and these variations are stored in dataframes. These dataframes are then used to provide the data to generate the EDA visualizations. Each visualization, dataframe, or header is stored within a div. The divs contain additional styling attributes like position, size, *etc.* All of these divs are placed inside another div which is returned to the callback function as outputs.

The page automatically switches to the Analyze tab when the form is submitted. Numerous steps were taken to ensure proper spacing was provided between each graph and/or data table especially for large date range inputs to avoid overlapping. On the Analyze tab, users can interact with the content in multiple ways. The first is by interacting with the graphs. Plotly graphs have in-built features to allow users to perform functions such as zoom, pan, box select, pan select, and autoscale. The other interactions are clicking on the buttons to download the data model and the data table summaries for the exploratory data analysis. The download functionality was created by returning further information to the same callback function initiated when the user clicks submit. Each dataframe made for the various EDA data is returned to the callback by storing the data into DCC.Stores (global variables). These global variables, storing the dataframes, can be called and appended to csv files for the user to download. The backend processes to generate the visualization outputs and downloadable files is summarized in Figure 3.16.

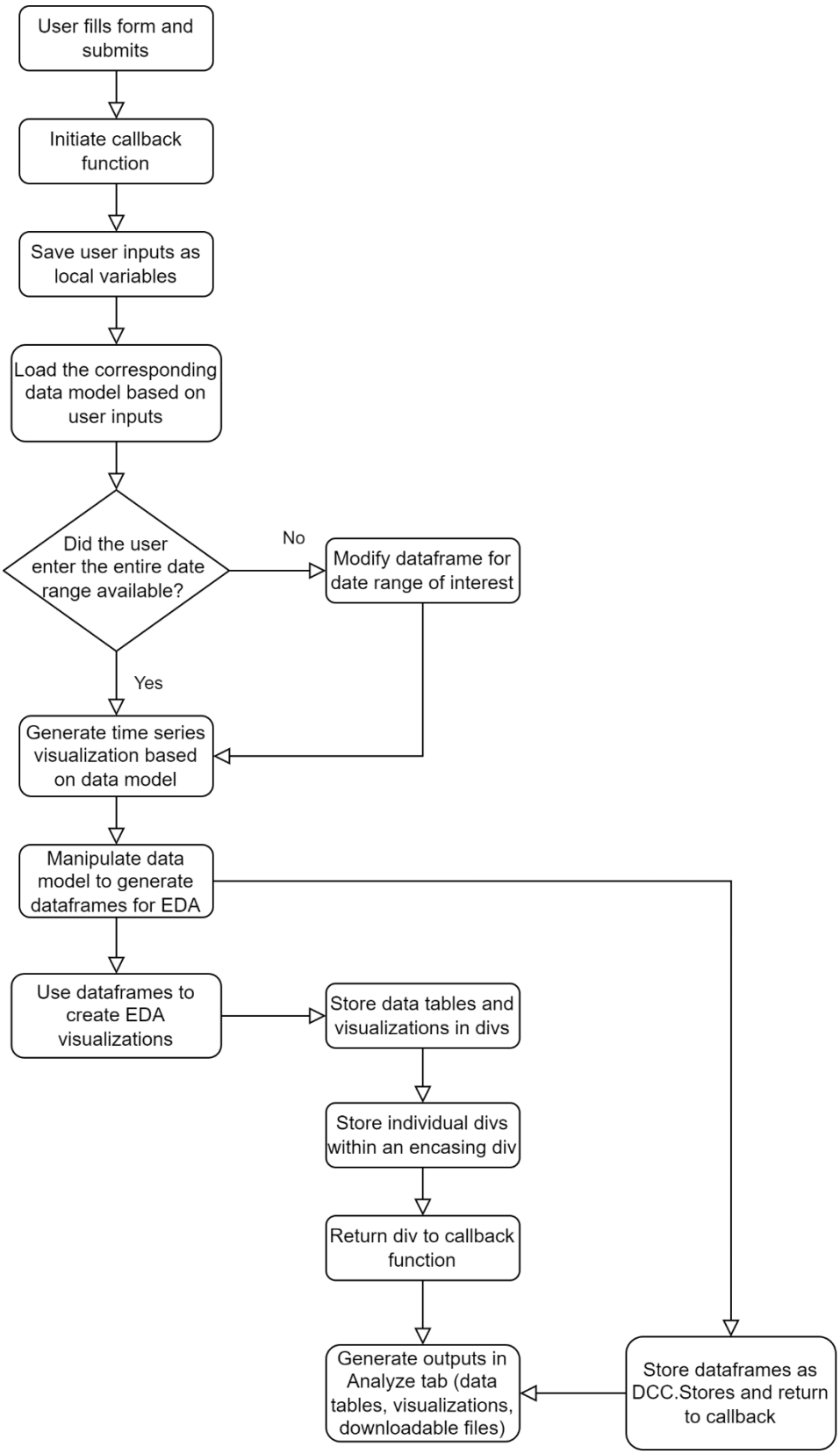


Figure 3.16: Flowchart of process to generate outputs on Analyze tab (Hosted Data)

3.5.3: Summary

A flowchart of user interaction to view and download a sample comparative analysis is in Figure 3.17. Figures 3.18 – 3.21 are images from the WARM tool corresponding to each step in the example user interaction flowchart.

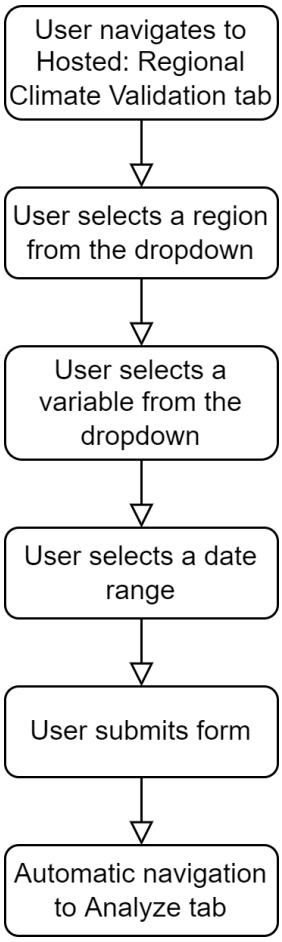


Figure 3.17: Flowchart of user interaction

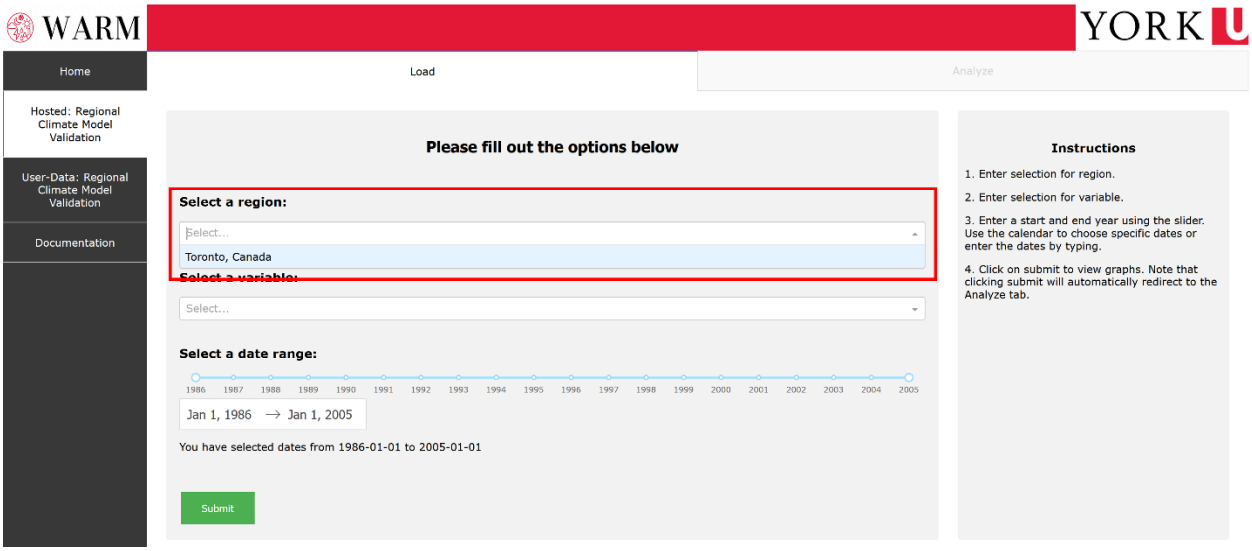


Figure 3.18: Region options in WARM tool

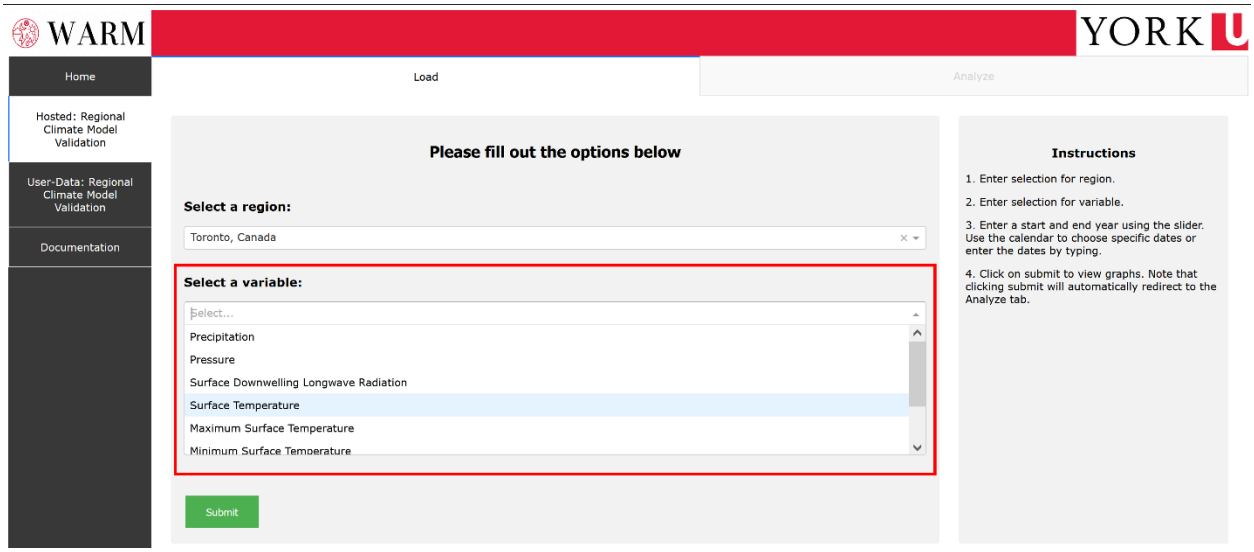


Figure 3.19: Climate variable options in WARM tool

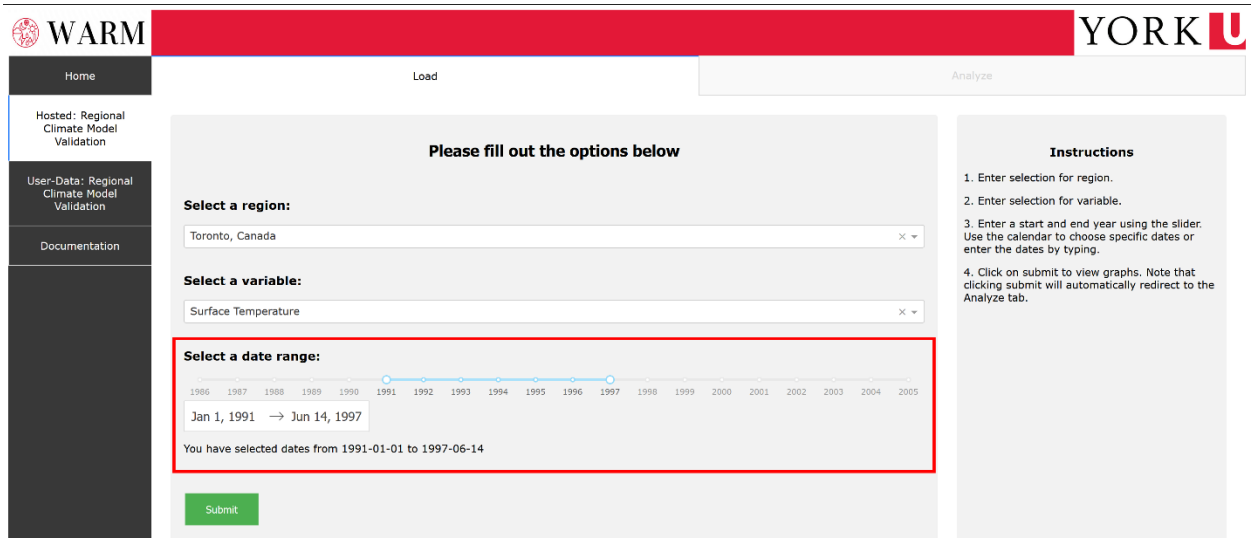


Figure 3.20: Date range selection in WARM tool

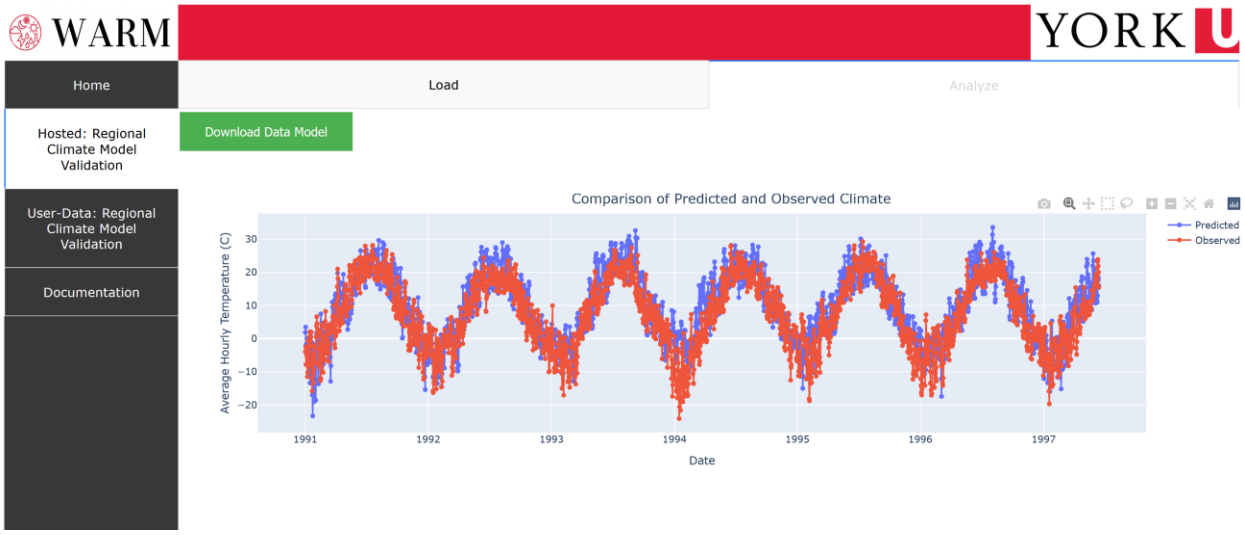


Figure 3.21: Date range selection in WARM tool

Chapter 4: Case Studies

To expand the functionality of the WARM tool, a feature was added that would allow users to view a comparative analysis of any 25 km x 25 km region within North America (as defined in the CORDEX datasets) between 1986-2005. Since the CORDEX data was already collected and extracted for the North America region for the 20-year time period, it was feasible to add this additional feature by making some programmatic changes to how the CORDEX data was extracted and mapped. The backend process of the User-Data: Regional Climate Model Validation tab is in Figure 4.1. The earlier functionality (discussed in chapter 3) used an observed dataset that is hosted by the WARM tool to compare data for Toronto. The datasets hosted by the tool provide the benefits of pre-collected data that is processed in the correct format and evaluated for various data quality issues. In the subsequent section, details on how the programmatic comparison was developed are discussed. The section also includes a resource on how users can collect and format their observed climate datasets and a stepwise guideline on how to derive the equivalent CORDEX latitude and longitude grid tile values from regular latitude and longitude degrees.

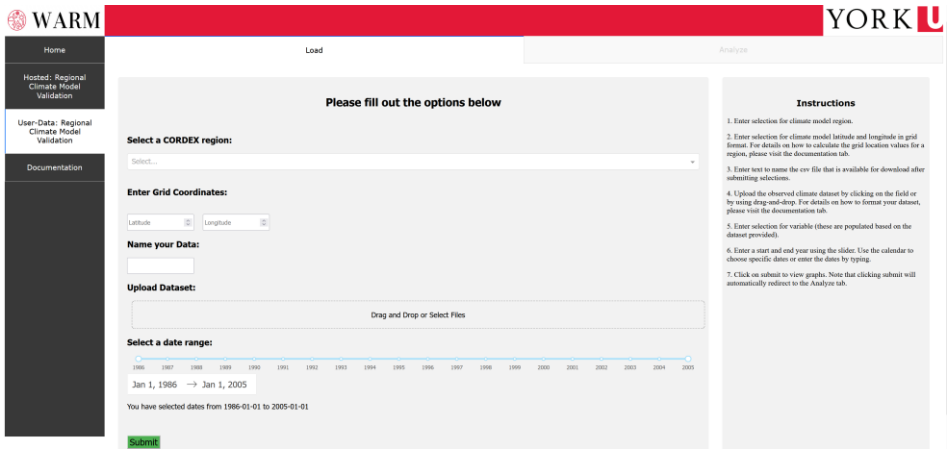


Figure 4.1: Design of the user-data tab

4.1: Development of the Programmatic Comparison of Climate Data

4.1.1: Programmatic Differences

The programmatic differences between the development of the comparison functionality that uses data hosted by the tool versus data provided by the user comes down to the creation of the data model. In the hosted-data version, each step of the data preprocessing occurs within an individual Jupyter Notebook file. The output of each file is an intermediate data model (in the format of a locally saved csv file) that is used as the input of the following step. Since the region is static in the hosted version (Toronto), the intermediate data models never change and are only generated once. The final data model is used to extract the specific climate variable and date range that the user provides. In the case of user-provided data, the dataset can contain information for any region. Hence, each time the inputs are submitted, the preprocessing has to occur to develop new intermediate data models to generate the final data model used as input for visualizations. In Figure 4.2., the left side shows the same process as Figure 3.16. The grey boxes indicate steps that were changed for the user-data functionality. The steps highlighted in orange are the general phases of the process that were updated. The first step that changed was the input process for the user, as they now have to input a greater amount of information. This is covered in more detail in chapter 4.1.2. Once the form is submitted, the dataset is saved and the other inputs are also saved in a local csv file (hence called selections). As mentioned for the hosted data, each phase of the preprocessing steps was performed with an individual Jupyter Notebook file. Therefore for the user-data functionality, inline Jupyter notebook command-line prompts were used to run other Jupyter notebook files (%run). The first

file performed the data extraction steps by loading the selections and extracting climate model data based on the grid tile inputs. The extracted data was the input for the next step. As mentioned in Chapter 3, the climate model spanned across 4 files, therefore the next Jupyter notebook combined the climate model data into 1 file in the second file called by the %run command. At this time, any previous files from running the WARM tool were deleted. The third file performed the unit conversion to the climate model data to match the observed dataset. In the final file, the observed and projected climate data was mapped together into one file to create the final data model. The process from here onwards was the same as for the hosted data.

4.1.2: Data Collection and Format of Observed Climate Datasets

The documentation page of the WARM tool provides users with a resource from which they can collect observed climate data. The POWER project has data for the 8 climate variables available to compare in the WARM tool (POWER, 2021). The data download portal allows users to enter an exact location using longitude and latitude which is ideal for regional climate comparison. Since the POWER data access viewer is the recommended data source provided to users, they are also asked to format the dataset they input in the format of the POWER csv downloads. Users must provide the dataset in a csv format with a minimum of 4 columns; 3 of the columns are for year, date, and month; the 4th column must be one of the eight climate variables available for comparison. The maximum number of columns the dataset can have is 11, 3 for date variables and 8 for climate variables. The maximum number of rows is 7306 which would contain data from 1986-01-01 to 2005-12-31 and a header row. The headers and units must be formatted as described in Table 4.1.

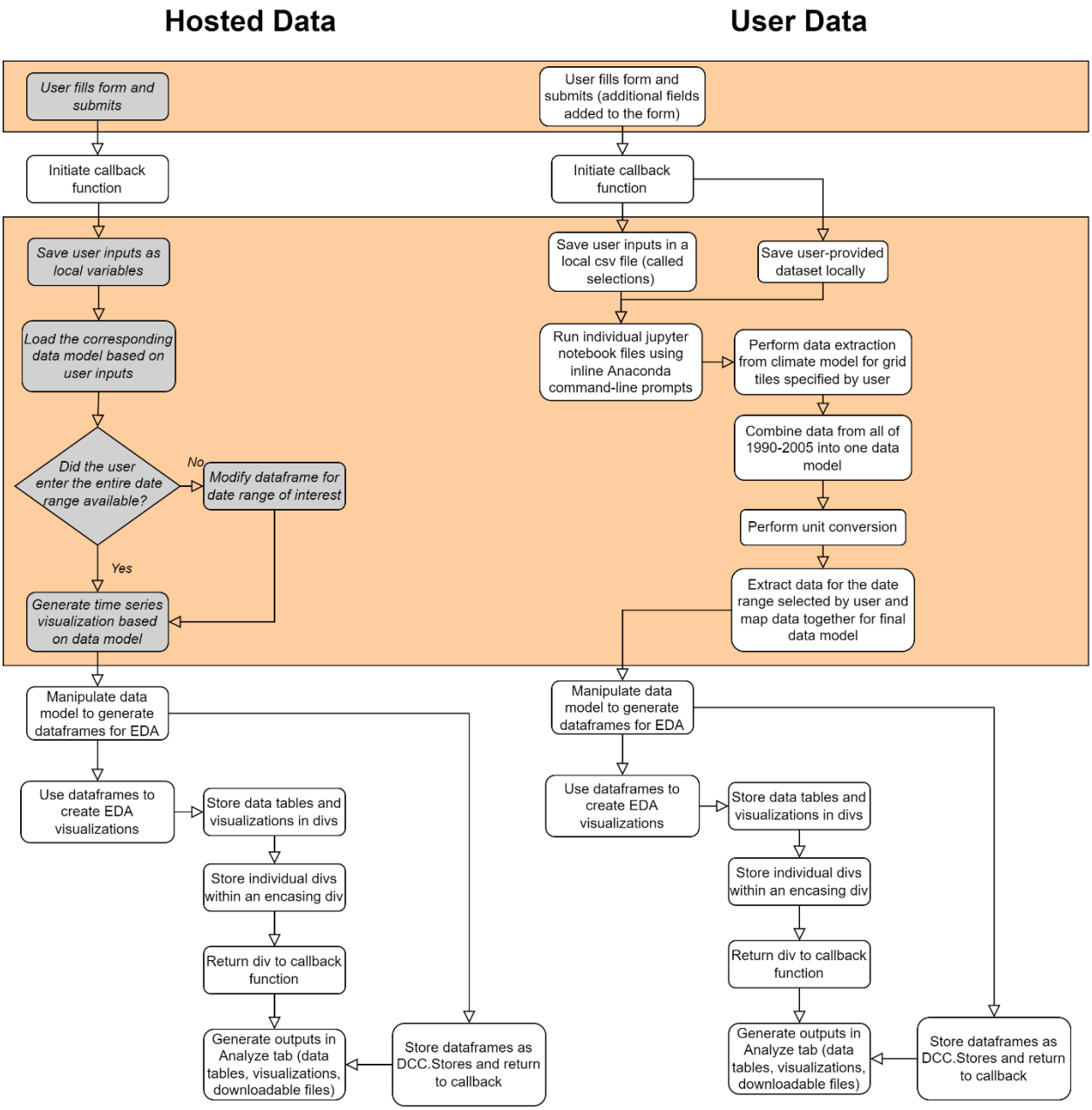


Figure 4.2: Flowchart of process to generate outputs on Analyze tab (User Data)

Table 4.1: Format of observed climate dataset input by user

Description	**Header Name**	**Units**
Year	YEAR	xxxx
Month	MO	xx
Day	DY	xx

Surface downwelling longwave radiation	ALLSKY_SFC_LW_DWN	$\frac{W}{m^2}$
Surface temperature	T2M	℃
Maximum surface temperature	T2M_MAX	℃
Minimum Surface Temperature	T2M_MIN	℃
Maximum surface wind speed	WS10M_MAX	$\frac{km}{h}$
Surface wind speed	WS10M	$\frac{km}{h}$
Surface pressure	PS	kPa
Precipitation	PRECTOTCORR	$\frac{mm}{day}$

Additionally, the POWER dataset contains rows with meta data with the variable description, these must be deleted by the users prior to uploading the data. The users are also responsible for any data quality-related issues with the dataset, such as the use of -999 to indicate missing climate values. While the WARM tool will not prevent users from visualizing a dataset with poor data quality, feedback will be given when users upload a dataset with blanks or null values such as -999. A summary of the formatting guidelines that users must follow is included on the Documentation page of the WARM tool and within Appendix A.

4.1.3: Deriving Grid Tile Values

A detailed description of the process used to extract data for a specific region from the CORDEX files was discussed in Chapter 3. A stepwise guideline summarizing this process is provided for users so that they can derive the grid tile values for any

location. This method is included on the Documentation page of the WARM tool and within Appendix A.

4.2: Case Study 1: Miami, Florida

To demonstrate the functionality of the hosted data page, data was downloaded from the POWER data access viewer (POWER, 2021) for Miami, Florida. The climate of Miami is hot and humid during the summer and mild/warm in the winter. In order to confirm whether this trend was generally true for the climate model predictions, the temperature variable was chosen for examination in the years 1986-1990 for the case study.

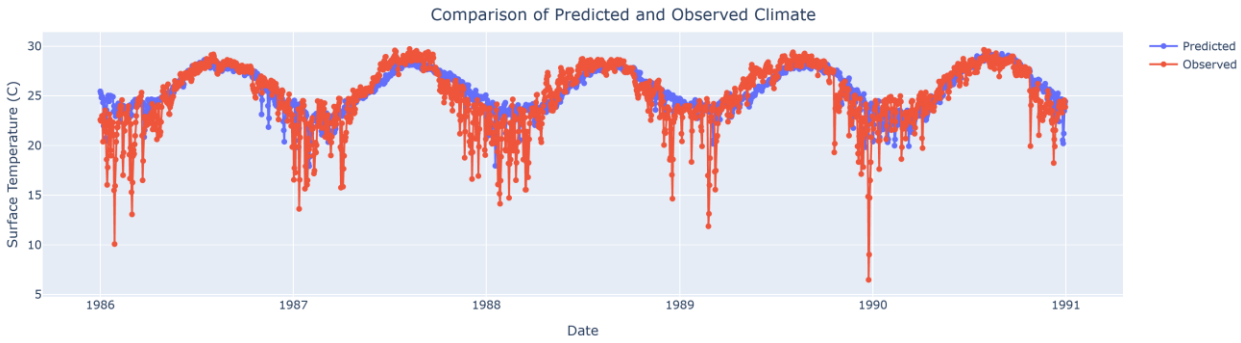


Figure 4.3: Output from WARM tool for daily temperature (°C) in Miami, Florida (1986-1990)

From the primary output of the WARM tool in Figure 4.3, it can be seen that both the observed and predicted values follow a similar trend of mid to low 20 degrees temperature in the winter and the low 30s in the summer. There are various outliers in the winter months where the temperature is cooler (usually around December – February). However, there are no visible outliers in the warmer range. It can also be seen that the variation of the outliers is not captured with complete accuracy by the climate model predictions. In some winter months, the climate model does predict lower

temperatures than the usual mid-20s such as in winter 1987, 198, and 1990, however the actual values for those months are even cooler.

Table 4.2: Observed and predicted monthly mean surface temperature (°C)

Observed Surface Temperature: Monthly Means

Month	1986	1987	1988	1989	1990
January	20.54	20.45	20.96	23.07	22.92
February	21.95	21.76	20.64	21.95	23.36
March	21.55	22.31	21.56	22.95	23.23
April	22.88	22.09	24.16	24.59	23.99
May	25.16	25.5	25.48	26.77	26.13
June	27.14	27.56	27.17	27.28	27.54
July	28.13	28.56	28.04	28.11	28.31
August	28.23	28.96	28.04	28.69	28.57
September	27.99	28.36	28.09	28.32	28.27
October	26.73	25.71	26.09	25.98	26.86
November	26.11	24.23	25.09	24.67	24.18
December	23.8	21.85	22.03	20	23.13
Annual	25.03	24.79	24.8	25.21	25.55

Predicted Surface Temperature: Monthly Means

Month	1986	1987	1988	1989	1990
January	24.1	22.79	23.03	24.19	22.6
February	23.66	22.13	23.1	23.73	22.57
March	23.81	22.4	23.47	23.46	22.84
April	24.46	23.83	23.64	24.2	24
May	25.5	25.18	24.77	25.07	25.6
June	27.19	26.27	26.28	26.29	27.29
July	28.23	27.76	27.45	27.44	28.53
August	27.93	28.21	27.91	28.15	28.84
September	27.86	27.82	27.79	28	28.68
October	27.15	27.09	26.95	27.52	27.42
November	25.04	26	25.45	25.47	25.95
December	24.08	24.8	24.63	23.88	23.94
Annual	25.76	25.38	25.39	25.63	25.71

To discover trends in the data, various summaries were created on a monthly or annual basis. The first is a summary of the monthly means in Table 4.2. The observed means by month range from 20°C to 28°C in 1986-1989. However, the means for 1990 are between 23°C and 28°C. Comparatively, the predictions demonstrate means of 23°C to 28°C for the entire 5-year period. On average, annually, the mean is 25°C for the predicted and observed data. The WARM tool also presents the same summaries for medians as well, as seen in Table 4.3. Since the temperature in Miami is relatively mild year-round and has a symmetric distribution, the medians do not showcase any additional insights. This is evident for both observed and predicted median temperatures.

Table 4.3: Observed and predicted monthly median surface temperature (°C)

Observed Surface Temperature: Monthly Medians

Month	1986	1987	1988	1989	1990
January	21.58	21.24	21.63	23.51	23.2
February	22.48	22.15	20.7	23.6	24.24
March	22.3	22.3	22.65	23.66	23.2
April	22.76	22.79	24.16	24.57	24.14
May	24.94	25.48	25.42	26.77	26.43
June	27.15	27.72	27.23	27.38	27.66
July	28.12	28.56	28.06	28.29	28.27
August	28.33	28.97	28.05	28.7	28.69
September	28.04	28.48	28.26	28.35	28.37
October	26.75	25.69	25.92	27.34	27.41
November	26.14	24.56	25.41	25.04	24.3
December	23.75	22.3	22.95	21.14	23.76
Annual	25.71	25.21	25.5	25.71	25.27

Predicted Surface Temperature: Monthly Medians

Month	1986	1987	1988	1989	1990
January	24.59	23.39	23.27	24.17	23.1
February	23.72	22.47	23.23	23.79	22.65
March	24.13	22.72	23.71	23.68	22.94
April	24.53	23.83	23.73	24.22	24.12
May	25.35	25.18	24.9	25.02	25.91
June	27.25	26.31	26.33	26.25	27.39
July	28.25	27.83	27.48	27.44	28.6
August	27.9	28.22	27.87	28.2	28.85
September	27.88	27.83	27.73	27.95	28.63
October	27.25	27.09	27.19	27.47	27.66
November	25.36	26.08	25.43	25.68	25.9
December	24.32	24.85	24.71	24.05	24.2
Annual	25.49	25.48	25.29	25.31	25.9

The standard deviation summaries in Table 4.4 show that the observed temperatures have more spread or dispersion than the predicted values. The individual predictions tend to be closer to the predicted mean temperature. This is understandeable because the actual observations will have more variability whereas climate models will capture the general climate of the area with higher accuracy as compared to the exact values.

Table 4.4: Observed and predicted monthly standard deviation for surface temperature (°C)

Observed Surface Temperature: Monthly Standard Deviation

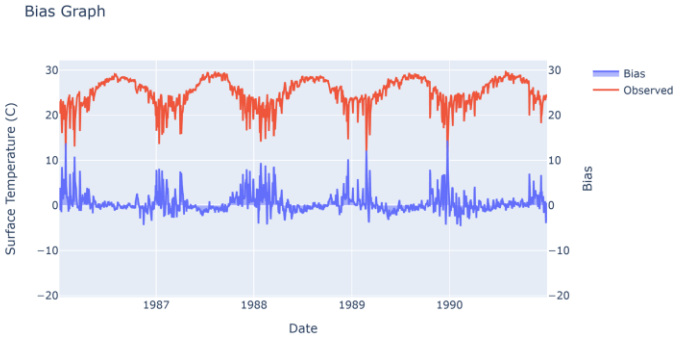
Month	1986	1987	1988	1989	1990
January	2.96	3.03	2.51	1.21	1.65
February	2.1	1.97	2.29	3.46	1.63
March	3.13	1.51	2.45	2.36	1.03
April	1.28	2.69	1.87	0.93	1.2
May	0.96	0.35	1.34	0.9	0.8
June	0.44	0.75	0.71	0.65	0.67
July	0.45	0.46	0.39	0.66	0.55
August	0.48	0.34	0.41	0.38	0.57
September	0.33	0.7	0.43	0.36	0.51
October	1.02	0.89	0.84	2.31	1.72
November	0.43	1.91	1.15	1.32	1.17
December	1.21	2.2	2.3	4.22	1.47
Annual	3.07	3.37	3.16	3.3	2.47

Predicted Surface Temperature: Monthly Standard Deviation

Month	1986	1987	1988	1989	1990
January	1.27	1.72	1.28	0.24	1.29
February	0.39	1.04	1.04	0.38	0.82
March	0.85	0.89	0.89	1.09	0.87
April	0.3	0.54	0.56	0.54	0.58
May	0.51	0.29	0.51	0.47	0.67
June	0.52	0.44	0.42	0.4	0.54
July	0.2	0.42	0.38	0.44	0.32
August	0.22	0.17	0.18	0.19	0.25
September	0.21	0.26	0.27	0.21	0.2
October	0.36	0.26	0.66	0.3	0.78
November	1.09	0.79	0.64	1.17	0.48
December	1.19	0.42	0.62	1.32	1.36
Annual	1.85	2.21	1.91	1.84	2.46

The bias graph in the WARM tool was developed as a combination of a line graph and an area graph. The line demonstrates the observed daily temperature, while the bias is the difference between the observed and predicted value. A positive bias indicates that the predicted value is greater than the actual while a negative bias is the opposite. The data table in Figure 4.4 has 2 values for each month-year. The first value

is the number of days where the bias is positive, and the second value is the number of days where the bias is negative. This excludes any days where the bias was between -1 to +1, as that is considered to be within a reasonable range to the observed value. In Figure 4.4, it can be seen that the climate model tends to have more positive bias than negative bias meaning it predicts warmer temperatures than the observed values. However, in July, August, September, and October, the climate predictions are fairly accurate on most days and completely accurate (within a 1°C difference) in some months. Further investigation would be required to test whether the performance of the climate model is high due to the low variability in temperature year-round in Miami.



Month	1986	1987	1988	1989	1990
January	27/2	18/2	17/3	14/0	7/12
February	14/1	7/9	20/3	10/1	4/15
March	15/2	9/9	15/3	7/5	3/6
April	21/0	14/2	7/14	2/9	5/5
May	4/1	0/0	2/13	0/26	1/8
June	1/0	0/21	0/15	1/18	3/6
July	0/1	0/14	0/6	0/11	3/1
August	0/0	0/5	0/0	0/7	3/0
September	0/0	0/12	0/0	1/1	5/0
October	9/0	18/0	11/0	13/1	6/0
November	0/13	20/0	9/7	12/4	21/0
December	10/6	22/0	28/0	26/3	13/5

Figure 4.4: Bias graph and data table for observed and predicted surface temperature (°C)

The month-wise box plots, in Figure 4.5., show the overall trend month-by-month for 5 years. It can be seen that there is a greater degree of variability in the observed temperature (as was noticed with the monthly medians as well). For example, in December 1989, there was one day with temperature of 6.9°C whereas in all the predictions made by the climate model, the lowest temperature is 17.92°C for January 1987.

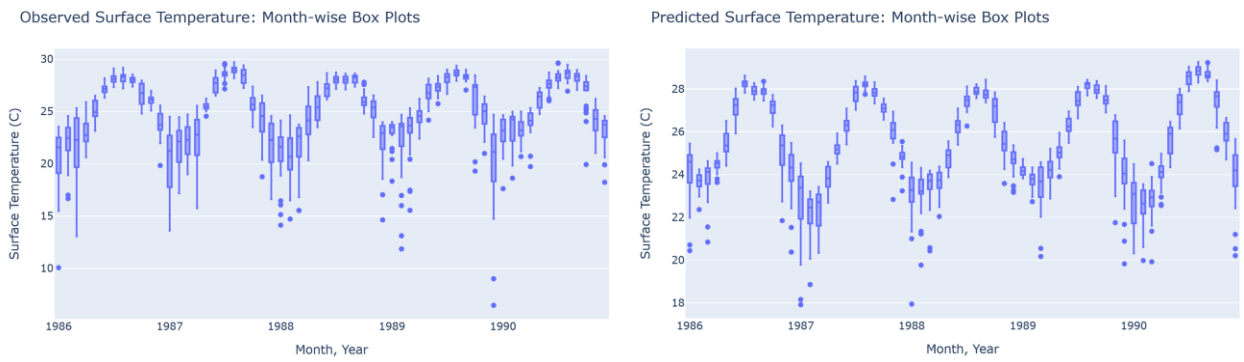
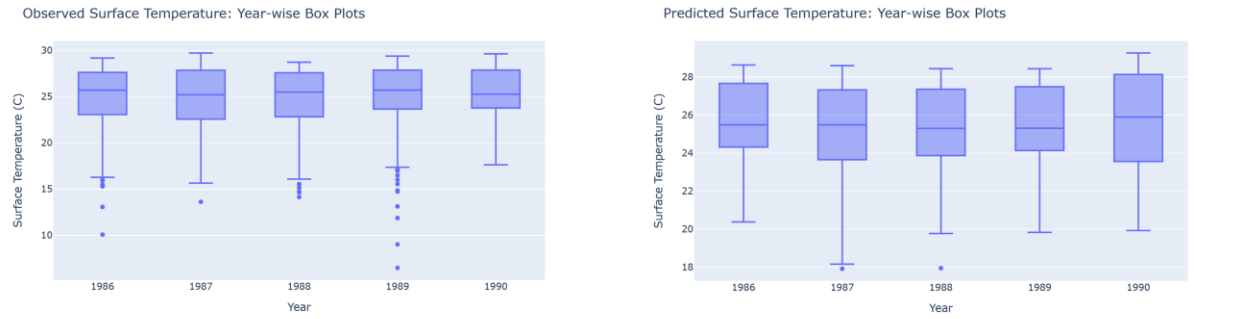


Figure 4.5: Month-wise box plots for observed and predicted surface temperature (°C)

The year-wise boxplots reveal that the predictions made by the climate model tend to be warmer than the actual minimum temperature and cooler than the actual maximum temperature. The overall range of the observed temperature is between 15-30°C, however the range for the predictions is between 18-29°C. Another interesting insight is that the observed values have a greater number of outliers than the predicted observations. This may allude that climate models have poorer performance in detecting "extreme" or "outlier" weather events.



Date	Minimum	First Quartile	Median	Third Quartile	Maximum
1986	16.28	23.07	25.71	27.63	29.18
1987	15.65	22.58	25.21	27.86	29.72
1988	16.08	22.83	25.5	27.58	28.73
1989	17.37	23.66	25.71	27.87	29.39
1990	17.63	23.78	25.27	27.87	29.64

Date	Minimum	First Quartile	Median	Third Quartile	Maximum
1986	20.38	24.32	25.49	27.66	28.63
1987	18.17	23.66	25.48	27.32	28.59
1988	19.78	23.87	25.29	27.35	28.43
1989	19.83	24.13	25.31	27.47	28.43
1990	19.93	23.56	25.9	28.12	29.26

Figure 4.6: Year-wise boxplots for observed and surface predicted temperature (°C)

The seasonality plots provide similar information as the time series graph however with each year plotted against each other. Using this view, any visible trends

that are changing year-to-year can be visualized. Although there are no insights that can be gleaned regarding climate variability over the 5 years, it can be seen that the surface temperature values for the predictions are much smoother indicating similar trends were predicted seasonally. However, the observed values are more varied, which aligns with previously mentioned analyses.

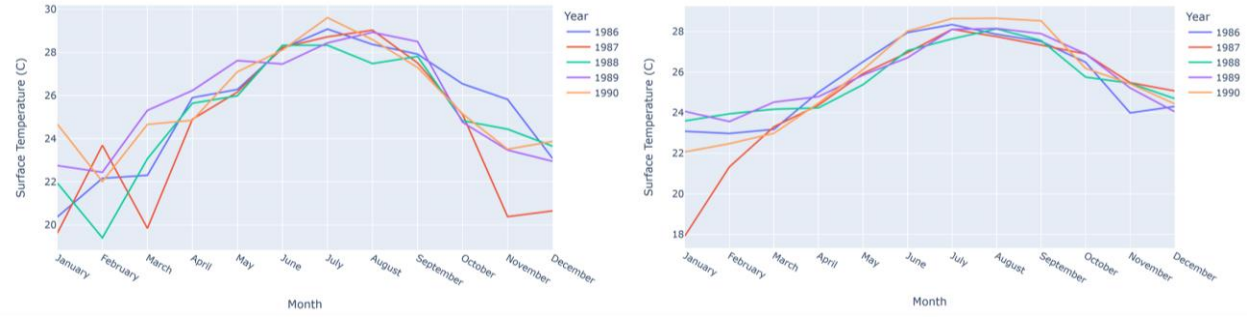


Figure 4.7: Seasonal plots for observed and predicted surface temperature (°C)

4.3: Case Study 2: Bismarck, North Dakota

The general climate of Bismarck is warm during the summer and cold with rain, snow, and high winds in the winter. North Dakota is known to be a windy region and therefore the case study was focussed on investigating the surface wind speed between 1991-1995.

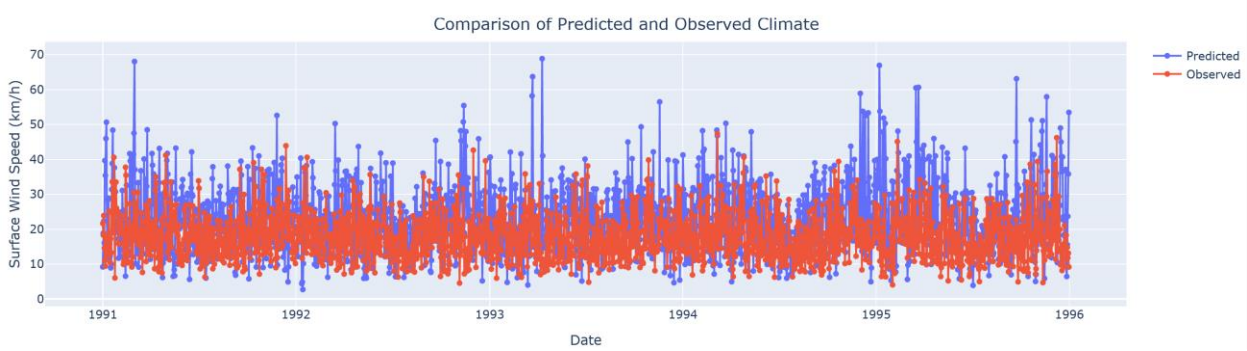


Figure 4.8: Output from WARM tool for surface wind speed ($\frac{km}{h}$) in Bismarck, North Dakota (1991-1995)

The primary output of the graph, in Figure 4.8, displays the time series trend of predicted and observed surface wind speed. The observations are within the range of 4 $\frac{km}{h}$ and 50 $\frac{km}{h}$, however the predictions range from 2 $\frac{km}{h}$ to 70 $\frac{km}{h}$. The predictions greater than 50 $\frac{km}{h}$ only make up 1.42% of the records from the 5 years of data. The overall predictions made by the climate model seem fairly similar to the observed values, but this will be further examined when investigating the bias values.

The comparison of the observed and predicted mean surface wind speed in Table 4.5 shows that the predictions made by the climate model are higher than the observed values. It is known that the higher values which are skewing the means are only 26 records out of 1825 in total.

Table 4.5: Observed and predicted monthly mean surface wind speed ($\frac{km}{h}$)

Observed Surface Wind Speed: Monthly Means

Month	1991	1992	1993	1994	1995
January	19.85	20.53	18.87	16.97	15.84
February	18.33	18.33	16.32	19.02	20.71
March	19.77	17.42	20.21	21.35	19.72
April	20.07	18.86	15.44	19.81	17.93
May	19.26	18.94	17.18	18.03	18.1
June	18.11	18.42	19.04	18.23	15.37
July	16.77	14.65	16.69	15.23	15.3
August	16.4	15.82	16.14	15.28	19.45
September	17.69	19.2	17.41	15.84	15.58
October	20.18	17.03	19.29	20.83	17.84
November	19.71	16.65	19.39	20.65	19.91
December	20.87	18.99	18.03	17.64	18.24
Annual	18.92	17.9	17.85	18.23	17.82

Predicted Surface Wind Speed: Monthly Means

Month	1991	1992	1993	1994	1995
January	24.33	22.15	22.33	22.03	27.82
February	23.02	20.47	19.61	24.03	26.58
March	25.17	21.19	22.4	26.04	27.66
April	23.98	23.31	22.86	24.25	26.72
May	22.72	18.5	21.99	24.16	25.46
June	22.58	23.24	22.5	20.78	19.86
July	20.24	17.94	21.25	14.58	17.3
August	22.71	18.72	21.49	20.68	20.09
September	22.08	22.45	24.25	23.48	23.52
October	22.38	22.02	23.96	22.58	23.74
November	24.5	29.49	23.11	27.31	27.8
December	22.93	22.45	21.11	28.11	27.08
Annual	23.05	21.81	22.25	23.15	24.45

The surface wind speed is less skewed for the predicted values when looking at the medians in Table 4.6 as that ignores the outliers. Similarly for the standard deviation in Table 4.7, it can be seen that the deviation from the mean is greater for the predicted values due to a greater range of values and outliers.

Table 4.6: Observed and predicted monthly median surface wind speed ($\frac{km}{h}$)

Observed Surface Wind Speed: Monthly Medians

Month	1991	1992	1993	1994	1995
January	18.32	19.87	18.94	17.03	13.9
February	18.99	18.04	18.05	19.01	20.83
March	19.04	17.03	21.17	20.84	18.83
April	19.35	18.41	13.16	18.38	16.42
May	18.86	18.5	17.39	16.38	18.07
June	17.71	17.8	18.31	16.18	13.16
July	15.59	13.75	15.01	13.32	14.29
August	16.27	14.29	15.88	14.26	20.45
September	17.26	17.82	16.04	15.1	15.14
October	19.48	15.62	18.68	19.3	15.7
November	17.96	15.88	18.65	19.44	18.13
December	19.22	17.93	17.17	17.82	15.77
Annual	18.25	17.32	17.24	17.68	16.88

Predicted Surface Wind Speed: Monthly Medians

Month	1991	1992	1993	1994	1995
January	21.47	22.56	22.11	21.91	22.76
February	21.99	18.92	17.36	20.78	24.57
March	21.23	16.97	21.45	24.84	24.82
April	23.26	22.71	19.95	23.56	24.3
May	21.55	18.38	21.26	24.79	26.15
June	22.68	23.18	22.29	20.12	18.66
July	20.64	20.06	20.19	14.55	17.7
August	21.75	17.6	22.59	20.49	18.4
September	21.77	19.67	22.72	24.47	21.06
October	20.97	21.64	24.41	22.96	22.7
November	23.44	27.65	22.23	27.36	25.43
December	24.09	21.99	19.1	27.44	23.65
Annual	21.75	20.85	21.57	22.3	21.91

Table 4.7: Observed and predicted monthly standard deviation for surface wind speed $(\frac{km}{h})$

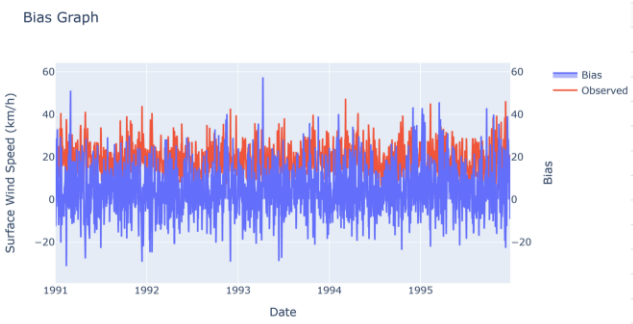
Observed Surface Wind Speed: Monthly Standard Deviation

Month	1991	1992	1993	1994	1995
January	7.61	6.4	5.76	4.82	6.36
February	7.09	5.06	4.94	6.71	7.51
March	5.64	5.35	7.18	7.45	6.37
April	9.16	7.55	6.78	8.34	7.08
May	6.41	6.85	5.4	6.47	7.12
June	4.73	4.93	7.7	7.52	6.72
July	6.63	5.17	7.34	5.6	5.97
August	3.79	5.62	4.63	4.81	5.66
September	6.98	6.09	6.22	5.95	5.16
October	7.27	6.59	7.33	7.88	8.39
November	7	7.07	5.64	6.17	7.6
December	6.98	6.97	6.69	3.91	8.76
Annual	6.77	6.31	6.45	6.66	7.1

Predicted Surface Wind Speed: Monthly Standard Deviation

Month	1991	1992	1993	1994	1995
January	12.01	9.22	8.77	9.68	15.21
February	9.59	8.21	9.49	10.3	10.07
March	13.16	10.02	13.16	10.98	13.61
April	9.6	10.55	11.26	10.96	10.16
May	10.34	8.61	8.72	9.28	8.79
June	8.55	8.49	8.34	6.55	8.87
July	7.55	7.4	7.11	5.65	6.16
August	6.54	6.66	7.32	6.33	7.91
September	8.13	9.02	7.75	9.84	11.61
October	9.5	7.54	7.74	7.89	10.47
November	10.21	11.27	9.48	8.23	11.8
December	8.66	8.74	8.87	13.42	12.06
Annual	9.59	9.23	9.09	9.82	11.22

The bias data table in Figure 4.9 displays the number of days where the bias is positive and the number of days where the bias is negative with the exclusion of days where the bias was within -5 to +5 $\frac{km}{h}$. Approximately 20-25 days within each month out of the 5 years had predicted values outside the acceptable prediction range. This indicates that although the overall trend of predictions was fairly close to the observed values, the day-to-day comparison was not precise.



Month	1991	1992	1993	1994	1995
January	11/7	13/6	15/8	13/5	18/4
February	14/4	10/7	11/6	11/7	16/3
March	14/8	14/7	12/10	15/6	17/7
April	16/9	16/7	18/4	13/12	18/2
May	15/10	8/11	17/7	16/5	19/3
June	17/8	14/5	14/6	12/8	14/3
July	12/4	14/6	14/5	8/11	13/6
August	18/4	10/2	18/5	17/4	11/10
September	15/8	11/6	18/4	17/4	15/3
October	10/8	16/4	15/6	14/5	15/5
November	15/4	21/3	10/6	18/6	16/5
December	14/8	14/8	15/7	21/3	19/7

Figure 4.9: Bias graph and data table for observed and predicted surface wind speed $(\frac{km}{h})$

The month-wise and year-wise boxplots also reinforce the previous discussion on the medians and range of the observed and predicted values. The graphs in Figure 4.10 and Figure 4.11 also showcase the outliers which are relatively equal in quantity in both cases. It can also be seen that the outliers are in the 40 $\frac{km}{h}$ range for the observed values but the 60-70 $\frac{km}{h}$ for the predictions. There is only one outlier in the lower range for both types of records in the month-wise boxplots.

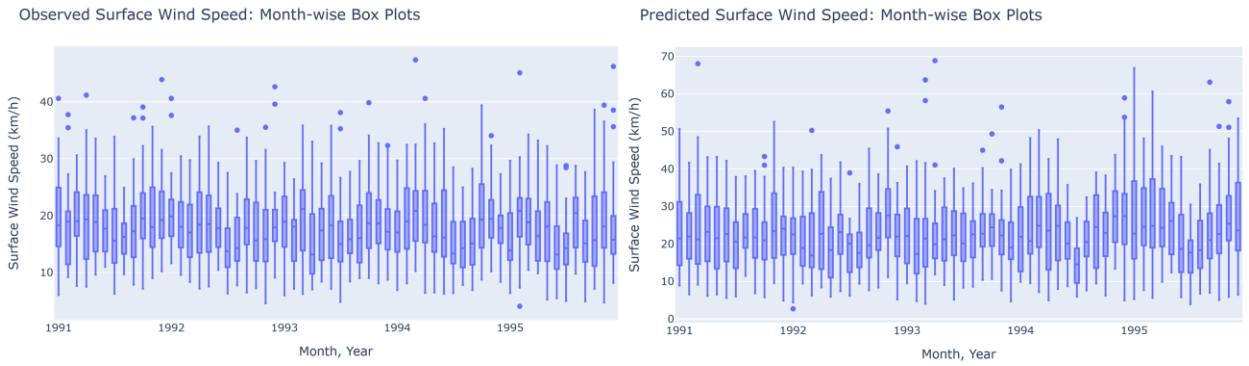
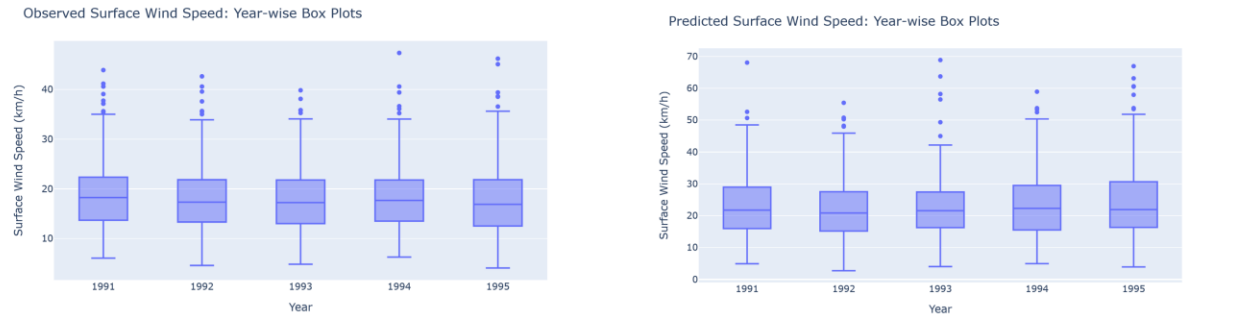


Figure 4.10: Month-wise boxplots for observed and predicted surface wind speed $(\frac{km}{h})$



Date	Minimum	First Quartile	Median	Third Quartile	Maximum
1991	6.05	13.68	18.25	22.32	35.03
1992	4.57	13.32	17.32	21.78	33.91
1993	4.82	13.03	17.24	21.78	34.09
1994	6.26	13.54	17.68	21.78	34.06
1995	4.07	12.53	16.88	21.82	35.64

Date	Minimum	First Quartile	Median	Third Quartile	Maximum
1991	4.91	15.96	21.75	28.97	48.49
1992	2.73	15.17	20.85	27.47	48.22
1993	4.02	16.27	21.57	27.42	44.99
1994	4.96	15.52	22.3	29.47	48.24
1995	3.92	16.35	21.91	30.57	48.12

Figure 4.11: Year-wise boxplots for observed and predicted surface wind speed ($\frac{km}{h}$)

The seasonal plots are an effective method for visualizing recurring patterns year over year. For example, the Miami use case for temperature had seasonal variations between summer and winter months. Although these were slight changes due to the year-round mild temperature in Miami, it was still a seasonal pattern. However, for variables like wind speed, there is no observable seasonality as seen in Figure 4.12.

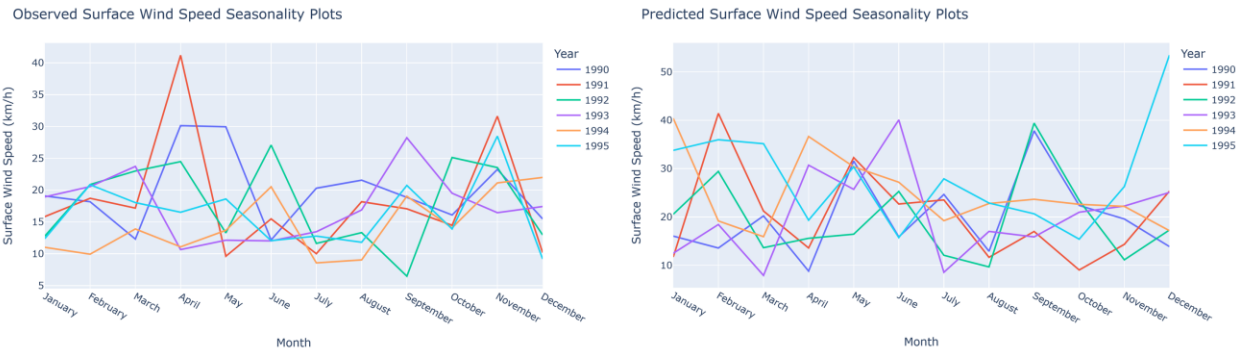


Figure 4.12: Seasonal plots for observed and predicted surface wind speed ($\frac{km}{h}$)

4.4: Case Study 3: San Diego, California

The general climate of San Diego is dry and sunny during the summer and cold and wet during the winter. Additionally, San Diego only receives approximately 42 days

of rainfall annually. To explore this trend further, the case study for San Diego was based on the precipitation variable between 2001-2005. Figure 4.13 shows that the most common value is 0mm of precipitation. For the observed values, only 8.71% of records in the 5-year range have a value of greater than 10mm. Similarly, only 8.22% of the predicted values are greater than 10mm. The ranges for both types of records are also similar. The observed precipitation ranges from 0-40mm approximately and the predicted precipitation ranges from 0-46mm, with the exception of one record of 71mm. The 8% of records that had more than 10mm of precipitation tend to occur in the colder months including: January, February, October, November, and December.

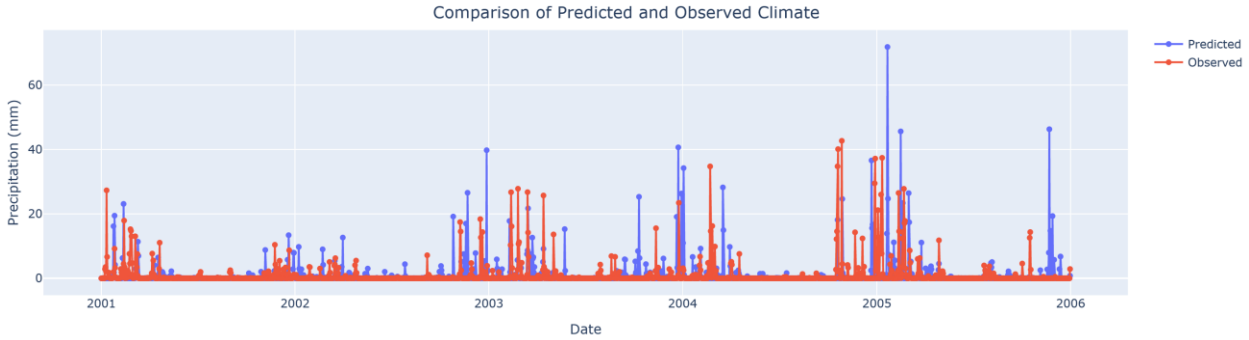


Figure 4.13: Output from WARM tool for daily precipitation ($\frac{mm}{day}$) in San Diego, California (2001-2005)

The comparison in Table 4.8 between the observed and predicted total annual precipitation is varying each year. The observed precipitation is 100mm greater than the predicted in 2001, 50mm greater in 2003, and 150mm greater in 2004. The predicted total precipitation is approximately 70mm greater in 2002 and 40mm in 2005.

Table 4.8: Total observed and predicted annual precipitation ($\frac{mm}{day}$)

Annual Precipitation

Year	Observed	Projected
2001	292.29	197.22
2002	175.09	241.56
2003	337.85	395.66
2004	432.8	279.23
2005	412.92	449.91

The monthly total precipitation reveals that the summer months between May to September are usually dry as compared to the remainder of the year. This trend is true for both the observed and predicted values. There are some exceptions to this trend however those exceptions do not align when comparing the observed and predicted values in all but one case. The predicted monthly precipitation, 21.56mm, in May 2003 is relatively precise to the observed value which is 19.54mm.

Table 4.9: Total observed (left) and predicted (right) monthly precipitation ($\frac{mm}{day}$)

Month	2001	2002	2003	2004	2005
January	61.31	11.1	5.92	10.68	147.1
February	100.69	4.08	117.04	97.46	146.2
March	27.88	27.97	55.31	14.21	36.92
April	34.22	15.2	42.21	22.79	15.43
May	1.39	0.32	19.55	0.24	4
June	0.04	0.27	1	0.01	0.27
July	5.61	0.9	10.46	1.15	13.18
August	2.49	0.07	19.58	2.26	8.56
September	4.84	11.38	4.21	3.86	2.43
October	0.38	0.95	1.09	156.72	34.48
November	25.94	45.91	26.66	27.59	0.73
December	27.5	56.94	34.82	95.83	3.62

Month	2001	2002	2003	2004	2005
January	43.93	18.68	11.94	58.68	119.54
February	34.55	16.55	27.75	15.68	94.91
March	32.02	13.18	91.21	63.91	65.82
April	19.48	19.64	19.56	7.46	28.58
May	6.44	5.61	21.56	7.60	2.04
June	1.16	3.68	0.84	1.93	0.43
July	0.02	8.43	0.41	1.83	1.22
August	0.83	0.29	3.22	0.46	16.77
September	1.79	4.17	17.41	3.33	6.47
October	5.32	27.49	54.46	44.90	0.90
November	11.27	60.00	11.87	0.76	96.64
December	40.40	63.85	135.44	72.68	16.58

The monthly means are slightly skewed by the few days in the 5-year range that have heavy rain as seen in Table 4.10. The monthly medians, in Table 4.11, accurately showcase that the precipitation values are generally 0.

Table 4.10: Observed and predicted monthly mean daily precipitation ($\frac{mm}{day}$)

Observed Precipitation: Monthly Means

Month	2001	2002	2003	2004	2005
January	1.98	0.36	0.19	0.34	4.75
February	3.6	0.15	4.18	3.48	5.22
March	0.9	0.9	1.78	0.46	1.19
April	1.14	0.51	1.41	0.76	0.51
May	0.04	0.01	0.63	0.01	0.13
June	0	0.01	0.03	0	0.01
July	0.18	0.03	0.34	0.04	0.43
August	0.08	0	0.63	0.07	0.28
September	0.16	0.38	0.14	0.13	0.08
October	0.01	0.03	0.04	5.06	1.11
November	0.86	1.53	0.89	0.92	0.02
December	0.89	1.84	1.12	3.09	0.12
Annual	0.8	0.48	0.93	1.19	1.13

Predicted Precipitation: Monthly Means

Month	2001	2002	2003	2004	2005
January	1.42	0.6	0.39	1.89	3.86
February	1.23	0.59	0.99	0.56	3.39
March	1.03	0.43	2.94	2.06	2.12
April	0.65	0.65	0.65	0.25	0.95
May	0.21	0.18	0.7	0.25	0.07
June	0.04	0.12	0.03	0.06	0.01
July	0	0.27	0.01	0.06	0.04
August	0.03	0.01	0.1	0.01	0.54
September	0.06	0.14	0.58	0.11	0.22
October	0.17	0.89	1.76	1.45	0.03
November	0.38	2	0.4	0.03	3.22
December	1.3	2.06	4.37	2.34	0.53
Annual	0.54	0.66	1.08	0.77	1.23

Table 4.11: Observed and predicted monthly median daily precipitation $(\frac{mm}{day})$

Observed Precipitation: Monthly Medians

Month	2001	2002	2003	2004	2005
January	0.17	0	0	0	0.08
February	1.47	0	0.24	0.29	0.85
March	0	0.01	0	0	0.01
April	0.12	0.01	0.06	0	0
May	0	0	0	0	0
June	0	0	0	0	0
July	0	0	0	0	0
August	0	0	0.01	0	0
September	0	0.02	0	0	0
October	0	0	0	0	0
November	0.02	0	0	0	0
December	0.07	0.01	0.01	0	0
Annual	0	0	0	0	0

Predicted Precipitation: Monthly Medians

Month	2001	2002	2003	2004	2005
January	0	0	0	0	0
February	0	0	0.01	0	0.03
March	0	0	0.1	0.01	0.07
April	0.04	0.01	0.03	0.01	0.07
May	0.02	0	0.01	0	0
June	0.01	0	0	0.02	0
July	0	0.01	0	0	0
August	0	0	0	0	0
September	0.01	0.05	0.05	0.01	0.03
October	0	0	0.01	0	0
November	0	0.03	0.01	0	0
December	0	0.01	0	0	0
Annual	0	0	0	0	0

The monthly standard deviation values for both observed and predicted values are very similar. Due to the overall trend of low precipitation, the comparison between the deviations is fairly accurate.

Table 4.12: Observed and predicted monthly standard deviation for daily precipitation $(\frac{mm}{day})$

Observed Precipitation: Monthly Standard Deviation

Month	2001	2002	2003	2004	2005
January	5.16	0.89	0.54	0.74	8.97
February	5.24	0.58	7.81	7.4	8.46
March	2.69	1.74	5.44	1.81	2.4
April	2.55	1.25	4.69	1.92	2.15
May	0.1	0.04	2.46	0.03	0.47
June	0	0.02	0.08	0	0.03
July	0.46	0.07	0.92	0.11	0.95
August	0.35	0.01	1.69	0.25	0.74
September	0.55	1.31	0.39	0.36	0.26
October	0.03	0.08	0.13	11.9	3.44
November	2.08	4.1	2.88	2.76	0.09
December	1.9	4.57	4.22	8.52	0.52
Annual	2.65	2.01	3.59	5.06	4.12

Predicted Precipitation: Monthly Standard Deviation

Month	2001	2002	2003	2004	2005
January	4.48	1.86	1.26	6.43	13.62
February	4.47	1.84	3.48	1.79	9.59
March	2.64	1.2	4.89	5.81	5.83
April	1.4	2.31	1.88	0.62	1.37
May	0.45	0.6	2.76	0.45	0.15
June	0.05	0.37	0.05	0.1	0.03
July	0	0.8	0.04	0.29	0.12
August	0.06	0.03	0.42	0.04	1.42
September	0.09	0.38	1.49	0.25	0.46
October	0.48	3.49	4.86	5.4	0.06
November	1.65	5.77	1.22	0.06	9.29
December	3.15	7.23	9.96	7.53	1.62
Annual	2.28	3.1	4	3.78	5.86

The bias data table also reinforces that a large number of records are fairly close to the acceptable prediction range between -1 to +1mm. Although some months had nearly 20 days outside this range, the majority of months had fewer days with precipitation outside the thresholds. However, the bias graph demonstrates that the days on which the predicted precipitation did not match the observed value, the difference between the values was large. The number of observations with biases greater than 10 or less than -10mm were 87, meaning approximately 4.77% of all observations in the 5 years.

Month	2001	2002	2003	2004	2005
January	4/6	5/3	3/2	5/2	5/13
February	3/12	3/1	3/8	3/10	7/12
March	4/2	3/6	10/2	6/3	5/7
April	4/5	4/4	2/5	3/4	8/2
May	1/0	2/0	3/3	3/0	0/2
June	0/0	1/0	0/0	0/0	0/0
July	0/2	1/0	0/4	1/0	0/5
August	0/1	0/0	1/5	0/1	4/3
September	0/2	1/3	3/1	1/1	3/1
October	2/0	3/0	7/0	2/8	0/4
November	2/6	5/5	2/3	0/5	7/0
December	6/6	6/7	6/3	5/8	3/1

Figure 4.14: Bias graph and data table for observed and predicted daily precipitation $(\frac{mm}{day})$

As mentioned earlier, San Diego has a very dry climate with few days of rainfall annually. This can be visualized in the monthly and yearly boxplots in Figures 4.15 and 4.16. In many cases, the minimum, quartile 1, and median values are 0 with the quartile 3 and maximum values less than 0.5mm.

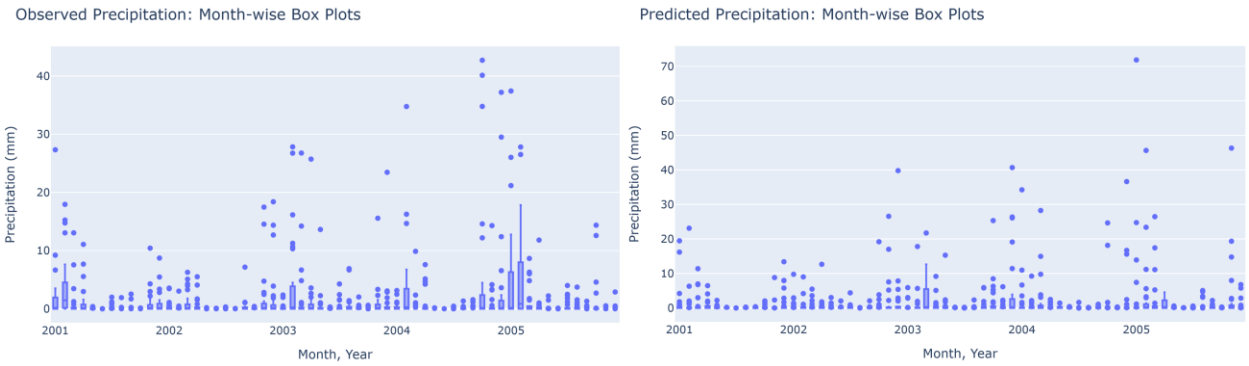
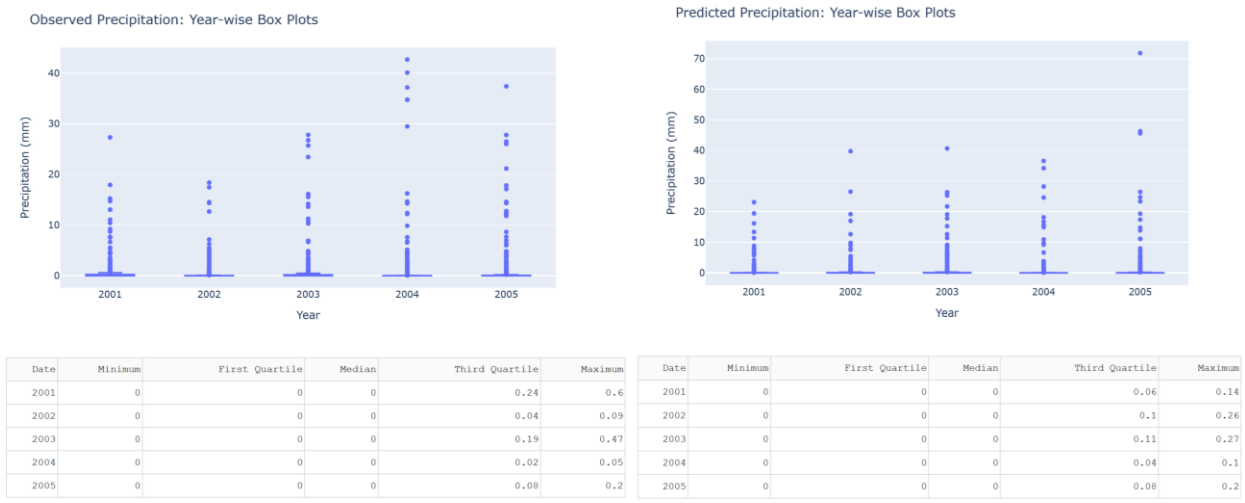


Figure 4.15: Month-wise boxplots for observed and predicted daily precipitation ($\frac{mm}{day}$)



Date	Minimum	First Quartile	Median	Third Quartile	Maximum
2001	0	0	0	0.24	0.6
2002	0	0	0	0.04	0.09
2003	0	0	0	0.19	0.47
2004	0	0	0	0.02	0.05
2005	0	0	0	0.08	0.2

Date	Minimum	First Quartile	Median	Third Quartile	Maximum
2001	0	0	0	0.06	0.14
2002	0	0	0	0.1	0.26
2003	0	0	0	0.11	0.27
2004	0	0	0	0.04	0.1
2005	0	0	0	0.08	0.2

Figure 4.16: Year-wise boxplots for observed and predicted daily precipitation ($\frac{mm}{day}$)

Seasonal plots, such as the ones in Figure 4.17, allow the visualization and comparison of patterns over multiple years at once. For the precipitation variable, this would be particularly useful for regions that experience monsoons, however this is not applicable to San Diego, California.

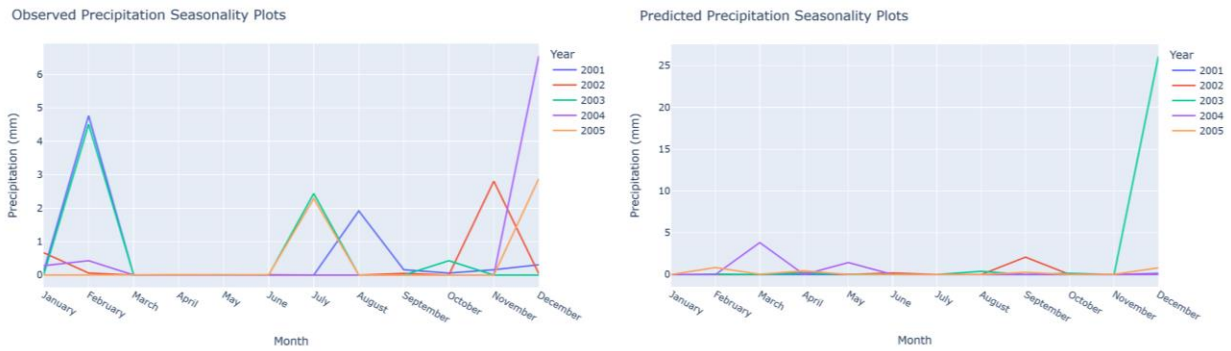


Figure 4.17: Seasonal plots for observed and predicted daily precipitation ($\frac{mm}{day}$)

Chapter 5: Conclusion and Future Work

The summary of the work performed in this thesis is as follows:

- An overall data analytics framework for comparing climate datasets was presented. This framework was specifically applied to compare climate model predictions with observed values. Each of the points below summarize a phase of the analytics framework that was applied.
- The requirements gathering phase was completed through an in depth literature review to understand the key climate concepts and terminology. A bibliometric analysis was conducted to review past climate visualization tools. The requirements gathering phase highlighted that analytical tools that visualize and compare climate modelling data with their historically observed values remain by far an underexplored area within the domain.
- Data was collected from multiple sources and extracted using conversion methods, manual plotting, and other techniques to ensure an equal comparison was being made between both datasets.
- A standardized data pipeline was established to perform data wrangling as climate datasets have widely varying formats and structures. A number of complexities were addressed during this process, one of which included data quality.
- A visualization tool, WARM, was developed that used the data model generated through the data pipeline to construct visualizations and data tables for end-users. The tool provides the functionalities to compare the results from the

climate model and observed climate and view trends over time for 8 climate variables.

Each phase of the data analytics process posed its own challenges. As a data analytics project, it was important to understand the climate change domain and its technicalities before pursuing the development of a framework. Therefore, the requirements gathering phase of the project was lengthy and required compiling information from many resources to gain a meaningful understanding of the problem at hand. The data collection was a challenging process due to the numerous observational climate datasets available online. Each dataset contained varying climate variables and were captured for different periods of time. Additionally, data was captured at different granularities (hourly, daily, weekly) and each dataset posed its own data quality issues. A large number of datasets had to be perused to arrive at the datasets that are used in this study. As for the climate model dataset, the format and amount of data posed a computational complexity. Although the amount of data required for the final comparison was smaller, the process of understanding the file format and locating the values required for extraction was onerous. The data wrangling steps had many coding-related and logical aspects that had to be developed continuously to reach a balance of efficient and accurate modelling. Lastly, the visualization dashboard was created keeping useability at the forefront which involved handling all the complexities within the programmatic features of the tool and presenting a simple user interface which ultimately resulted in lengthy source code.

There are two streams of future work for this research. The first stream is related to expanding the functionality of the tool. Firstly, in the hosted-data tab, only the Toronto

region datasets are hosted by the tool. More datasets can be added to this tab, as the hosted datasets incorporate data quality processes and perform any preprocessing and formatting steps, which would provide an easier user experience. For the user-data tab, a feature that programmatically performs data preprocessing and data quality steps can be added. Additionally, the ability to upload datasets from other CORDEX regions can be incorporated to expand the use of the tool. The CORDEX datasets contain data for over 50 climate variables, each of these is a potential climate variable that can be added to the tool, given that an equivalent observational dataset can be sourced. It would also be a big improvement for the tool to allow the comparison of multiple climate variables together. This would allow researchers to get a broader understanding of the comparison between the climate model and observed data. An accompanying addition could be the development of statistical analyses for understanding the relationships between the climate variables. Lastly, the most important aspect of improvement would the availability of the tool online for public use. The second stream of future work is in regards to growing the purpose of the tool. With the existing framework, a comparison between predicted extreme weather events against their actual occurrence can be performed. This can be developed with the classification of extreme weather events based on threshold values of climate variables. The classification algorithm would be applied to both climate model datasets and historical climate datasets. Then a comparison can be made regarding the prediction of extreme weather events made by climate models against daily climate data. Other uses for the existing framework include using the comparison between climate models and daily climate data and adding in other datasets to view the impact of future climate variability. For example, incorporating

data on soil, yield, and agricultural variables with predicted and observed climate data can allow for an understanding of the impacts on crops due to climate change.

References

Afework, B., Hanania, J., Stenhouse, K., & Donev, J. (2018). *Earth's energy budget.* Retrieved from https://energyeducation.ca/encyclopedia/Earth%27s_energy_budget

Afework, B., Hanania, J., Lloyd, E., Meyer, R., Sheardown, A., Stenhouse, K., … Donev, J. (2019). *Earth's energy flow.* Retrieved from https://energyeducation.ca/encyclopedia/Earth%27s_energy_flow

Anisimov, O. A. (2001). Predicting patterns of near-surface air temperature using empirical data. *Climatic Change*, *50*(3), 297-315. doi: 10.1023/A:1010658014439

Baede, A.P.M., Ahlonsou, E., Ding, Y., & Schimel, D. (2018). The Climate System: an Overview. *IPCC 2018*, 87-98.

Bautista, F., Bautista-Hernández, D. A., Álvarez, O., Anaya Romero, M., & Rosa, D. D. L. (2013). Software to identify climate change trends at the local level: A study case in Yucatán, México.

Bojinski, S., Verstraete, M., Peterson, T.C., Richter, C., Simmons, A., & Zemp, M. (2014). The Concept of Essential Climate Variables in Supper of Climate Research, Applications, and Policy. *Bulletin of the American Meteorological Society, 95*(1), 1431-1443. doi: 10.1175/BAMS-D-13-00047.1

Canadian Centre for Climate Modelling and Analysis (CCCMA). (2020). *CanRCM4 Model Output.* [Data file]. Retrieved from https://climate-modelling.canada.ca/climatemodeldata/canrcm/CanRCM4/index_cordex.shtml

Canada Centre for Climate Services (CCCS). (2020). *Climate Change Concepts.* Retrieved from https://www.canada.ca/en/environment-climate-change/services/climate-change/canadian-centre-climate-services/basics/concepts.html

Canada Centre for Climate Services (CCCS). (2018). *Scenarios and Climate Models.* Retrieved from https://www.canada.ca/en/environment-climate-change/services/climate-change/canadian-centre-climate-services/basics/scenario-models.html

Canadian Centre for Climate Modelling and Analysis (CCCMA). (2010). *CGCM3 Forcing: SRES Storylines and Scenario Families.* Retrieved from https://climate-modelling.canada.ca/data/cgcm3/cgcm3_forcing.shtml

Colose, C., Hanania, J., Stenhouse, K., & Donev, J. (2020). *Climate Forcing*. Retrieved from https://energyeducation.ca/encyclopedia/Climate_forcing

Dahal, P., Shrestha, M. L., Panthi, J., & Pradhananga, D. (2020). Modeling the future impacts of climate change on water availability in the Karnali River Basin of

Nepal Himalaya. *Environmental Research*, *185*, 109430. doi: 10.1016/j.envres.2020.109430

De Oliveira, O. J., da Silva, F. F., Juliani, F., Barbosa, L. C. F. M., & Nunhes, T. V. (2019). Bibliometric method for mapping the state-of-the-art and identifying research gaps and trends in literature: an essential instrument to support the development of scientific projects. In *Scientometrics recent advances*. IntechOpen.

Deb, C., Zhang, F., Yang, J., Lee, S. E., & Shah, K. W. (2017). A review on time series forecasting techniques for building energy consumption. *Renewable and Sustainable Energy Reviews*, *74*, 902-924. doi: 10.1016/j.rser.2017.02.085

Denchak, M. (2017). *Global Climate Change: What You Need to Know.* Retrieved from https://www.nrdc.org/stories/global-climate-change-what-you-need-know

Dong, L., Zhou, T., & Wu, B. (2014). Indian Ocean warming during 1958–2004 simulated by a climate system model and its mechanism. *Climate Dynamics*, *42*(1), 203-217. doi: 10.1007/s00382-013-1722-z

Donthu, N., Kumar, S., Mukherjee, D., Pandey, N., & Lim, W. M. (2021). How to conduct a bibliometric analysis: An overview and guidelines. *Journal of Business Research*, *133*, 285-296.

Ebert, E. E. (2001). Ability of a poor man's ensemble to predict the probability and distribution of precipitation. *Monthly Weather Review*, *129*(10), 2461-2480. doi: 10.1175/1520-0493(2001)129<2461:AOAPMS>2.0.CO;2

Egigu, M. L. (2020) Techniques of Filling Missing Values of Daily and Monthly Rain Fall Data: A Review. SF J Environ Earth Sci. 2020; 3 (1), 1036.

Elijah, O., Rahman, T. A., Orikumhi, I., Leow, C. Y., & Hindia, M. N. (2018). An overview of Internet of Things (IoT) and data analytics in agriculture: Benefits and challenges. *IEEE Internet of things Journal*, *5*(5), 3758-3773. doi: 10.1109/JIOT.2018.2844296

Ellegaard, O., & Wallin, J. A. (2015). The bibliometric analysis of scholarly production: How great is the impact? *Scientometrics*, *105*(3), 1809-1831.

Environment and Climate Change Canada. (2021). *Historical Data – Daily Data Report.* [Data file]. Retrieved from: https://climate.weather.gc.ca/historical_data/search_historic_data_e.html

Environment and Climate Change Canada. (2021). *1981–2010 Climate Normals & Averages.* [Data file]. Retrieved from: https://climate.weather.gc.ca/climate_normals/index_e.html

Essential Climate Variables: World Meteorological Organization (WMO) (n.d.). Retrieved from https://public.wmo.int/en/programmes/global-climate-observing-system/essential-climate-variables

European Centre for Medium-Range Weather Forecasts (ECMWF) (n.d.). Climate reanalysis. Retrieved from https://www.ecmwf.int/en/research/climate-reanalysis

Frith, S. M., Bhartia, P. K., Oman, L. D., Kramarova, N. A., McPeters, R. D., & Labow, G. J. (2020). Model-based climatology of diurnal variability in stratospheric ozone as a data analysis tool. *Atmospheric Measurement Techniques*, *13*(5), 2733-2749. doi: 10.5194/amt-13-2733-2020

Gagnon, P., Konan, B., Rousseau, A.N., & Slivitzky, M. (2009). Hydrometeorological validation of a Canadian Regional Model simulation within the Chaudière and Châteauguay watersheds (Québec, Canada). *Canadian Journal of Civil Engineering, 36*(2), 253-266. doi: 10.1139/L08-125

Ghumman, A. R., Alodah, A., Haider, H., & Shafiquzzaman, M. (2020). Evaluating the impact of climate change on stream flow: integrating GCM, hydraulic modelling and functional data analysis. *Arabian Journal of Geosciences*, *13*(17), 1-15. doi: 10.1007/s12517-020-05881-y

Giorgi, F. (2019). Thirty Years of Regional Climate Modeling: Where Are We and Where Are We Going Next? *Journal of Geophysical Research: Atmospheres, 124*(11), 5696-5723. doi: 10.1029/2018JD030094

Girvetz, E. H., Zganjar, C., Raber, G. T., Maurer, E. P., Kareiva, P., & Lawler, J. J. (2009). Applied climate-change analysis: the climate wizard tool. *PLoS One*, *4*(12), e8320. doi: 10.1371/journal.pone.0008320

Global Climate Change Vital Signs of the Planet (GCB) (n.d.). *Global Warming vs. Climate Change.* Retrieved from https://climate.nasa.gov/resources/global-warming-vs-climate-change/

Hempelmann, N., Ehbrecht, C., Alvarez-Castro, C., Brockmann, P., Falk, W., Hoffmann, J., ... & Yiou, P. (2018). Web processing service for climate impact and extreme weather event analyses. Flyingpigeon (Version 1.0). *Computers & Geosciences*, *110*, 65-72. doi: 10.1016/j.cageo.2017.10.004

Herzfeld, U. C., Drobot, S., Wu, W., Fowler, C., & Maslanik, J. (2007). Spatiotemporal climate model validation—case studies for MM5 over northwestern Canada and Alaska. *Earth Interactions*, *11*(20), 1-23. doi: 10.1175/EI208.1

Hyman, D. E., Whitehouse, D. R., Taylor, J. A., Larson, J. W., Hansen, D. P., & Lindesay, J. A. (1996). The ANU translator: facilitating computer visualization and data analysis of climate model outputs. *Environmental Software*, *11*(1-3), 65-72. doi: 10.1016/S0266-9838(96)00020-2

Ismail, H., Kamal, M. R., bin Abdullah, A. F., & bin Mohd, M. S. F. (2020). Climate-smart agro-hydrological model for a large scale rice irrigation scheme in Malaysia. *Applied Sciences*, *10*(11), 3906. doi: 10.3390/app10113906

Jaroensutasinee, K., Pheera, W., & Jaroensutasinee, M. (2014). Online weather data analysis and visualization tools for applications in ecoinformatics. *Earth Science Informatics*, *7*(3), 205-213. doi: 10.1007/s12145-013-0138-y

Kehrer, J., Ladstädter, F., Muigg, P., Doleisch, H., Steiner, A., & Hauser, H. (2008). Hypothesis generation in climate research with interactive visual data exploration. *IEEE Transactions on Visualization and Computer Graphics*, *14*(6), 1579-1586. doi: 10.1109/TVCG.2008.139

Kolsoumi, S. & Salehnia, N. (2019). Rotation of Coordinates Based On CORDEX Domains (1.0). Zenodo. https://doi.org/10.5281/zenodo.2590812

Kuo, Y. H., Wee, T. K., Sokolovskiy, S., Rocken, C., Schreiner, W., Hunt, D., & Anthes, R. A. (2004). Inversion and error estimation of GPS radio occultation data. *Journal of the Meteorological Society of Japan. Ser. II*, *82*, 507-531. doi: 10.2151/jmsj.2004.507

Lee, H., Goodman, A., McGibbney, L., Waliser, D. E., Kim, J., Loikith, P. C., ... & Massoud, E. C. (2018). Regional Climate Model Evaluation System powered by Apache Open Climate Workbench v1. 3.0: an enabling tool for facilitating regional climate studies. *Geoscientific Model Development*, *11*(11), 4435-4449. doi: 10.5194/gmd-11-4435-2018

McCarney, P.L. (2012). City Indicators on Climate Change: Implications for Governance. *Environment and Urbanization Asia, 3*(1), 1-39. doi: 10.1177/097542531200300102

McSweeney, R. & Hausfather, Z. (2018). *How do climate models work?* Retrieved from https://www.carbonbrief.org/qa-how-do-climate-models-work

Merz, B., Kreibich, H., Schwarze, R., & Thieken, A. (2010). Assessment of economic flood damage. *Natural Hazards and Earth System Sciences*, 10(8), 1697-1724. doi: 10.5194/nhess-10-1697-2010

National Oceanic and Atmospheric Administration (NOAAa) (n.d). *Climate Forcing.* Retrieved from https://www.climate.gov/maps-data/primer/climate-forcing

National Oceanic and Atmospheric Administration (NOAAb) (n.d.). *Climate Models.* Retrieved from https://www.climate.gov/maps-data/primer/climate-models

Panoply Data Viewer Version 4.12.7 [Computer software]. (2021). Retrieved from https://www.giss.nasa.gov/tools/panoply/credits.html

Pérez-Jordán, W., Castro-Almazan, J. A., & Munoz-Tunon, C. (2018). Precipitable water vapour forecasting: a tool for optimizing IR observations at Roque de los Muchachos Observatory. *Monthly Notices of the Royal Astronomical Society*, *477*(4), 5477-5485. doi: 10.1093/mnras/sty943

Power Analytics and Visualization for Climate Science (PAVICS) (n.d.). Retrieved from https://pavics.ouranos.ca/index.html

Program for Climate Model Diagnosis & Intercomparison (PCMDI) (n.d.) Retrieved from https://pcmdi.llnl.gov/index.html

Rathore, M. M., Ahmad, A., Paul, A., & Rho, S. (2016). Urban planning and building smart cities based on the internet of things using big data analytics. *Computer networks*, *101*, 63-80. doi: 10.1016/j.comnet.2015.12.023

Reyes-García, V., Fernández-Llamazares, A., Guèze, M., Garcés, A., Mallo, M., Vila-Gómez, M., & Vilaseca, M. (2016). Local indicators of climate change: The potential contribution of local knowledge to climate research. *Wiley Interdiscip Rev Clim Change, 7*(1), 109-124. doi: 10.1002/wcc.374.

Rocken, C., Anthes, R., Exner, M., Hunt, D., Sokolovskiy, S., Ware, R., ... & Zou, X. (1997). Analysis and validation of GPS/MET data in the neutral atmosphere. *Journal of Geophysical Research: Atmospheres*, *102*(D25), 29849-29866. doi: 10.1029/97JD02400

Rocken, C., Ying-Hwa, K., Schreiner, W. S., Hunt, D., Sokolovskiy, S., & McCormick, C. (2000). COSMIC system description. *Terrestrial Atmospheric and Oceanic Sciences*, *11*(1), 21-52.

Rodda, C., Birkel, S., & Mayewski, P. (2019). A 2000 year-long proxy and observational reconstruction of Central Asian climate. *Quaternary Science Reviews*, *223*, 105847. doi: 10.1016/j.quascirev.2019.07.029

Rummukainen, M. (2009). State-of-the-art with regional climate models. *WIREs Climate Change, 1*(1), 82-96. doi: 10.1002/wcc.8

Sattari, M. T., Rezazadeh-Joudi, A., & Kusiak, A. (2017). Assessment of different methods for estimation of missing data in precipitation studies. Hydrology Research, 48(4), 1032-1044.

Schmittner, A. (2018). *Introduction to Climate Change.* Corvallis, OR: Oregon State University.

Souto, M. J., Balseiro, C. F., Pérez-Muñuzuri, V., Xue, M., & Brewster, K. (2003). Impact of cloud analysis on numerical weather prediction in the Galician region of Spain. *Journal of Applied Meteorology*, *42*(1), 129-140. doi: 10.1175/1520-0450(2003)042<0129:IOCAON>2.0.CO;2

Szomszor, M., Adams, J., Fry, R., Gebert, C., Pendlebury, D. A., Potter, R. W., & Rogers, G. (2021). Interpreting bibliometric data. *Frontiers in Research Metrics and Analytics*, *5*, 30.

Tapiador, F.J., Navarro, A., Moreno, R., Sánchez, J.L., & García-Ortega, E. (2020). Regional climate models: 30 years of dynamical downscaling. *Atmospheric Research, 235*. doi: 10.1016/j.atmosres.2019.104785

Tardieu, F., Cabrera-Bosquet, L., Pridmore, T., & Bennett, M. (2017). Plant phenomics, from sensors to knowledge. *Current Biology*, *27*(15), R770-R783. doi: 10.1016/j.cub.2017.05.055

Terskii, P., Kuleshov, A., Chalov, S., Terskaia, A., Belyakova, P., Karthe, D., & Pluntke, T. (2019). Assessment of water balance for Russian subcatchment of western dvina river using SWAT model. *Frontiers in Earth Science*, 241. doi: 10.3389/feart.2019.00241

Titov, A., Gordov, E., Okladnikov, I., & Shulgina, T. (2009). Web-system for processing and visualization of meteorological data for Siberian environment research. *International Journal of Digital Earth*, *2*(S1), 105-119. doi: 10.1080/17538940902866187

The Prediction of Worldwide Energy Resources (POWER) Project. (2021). *FLASHFlux 4 Model Output.* [Data file]. Retrieved from: https://power.larc.nasa.gov/data-access-viewer/

United States Environmental Protection Agency (US EPA) (n.d.). *Climate Change Indicators in the United States.* Retrieved from https://www.epa.gov/climate-indicators

Wayne, G. (2013). *The Beginner's Guide to Representative Concentration Pathways.* Retrieved from https://skepticalscience.com/rcp.php?t=1

Weather Stats. (2021). *Climate Daily/Forecast/Sun Output.* [Data file]. Retrieved from: https://toronto.weatherstats.ca/download.html

Wielicki, B. A., Barkstrom, B. R., Baum, B. A., Charlock, T. P., Green, R. N., Kratz, D. P., ... & Welch, R. M. (1998). Clouds and the Earth's Radiant Energy System (CERES): algorithm overview. *IEEE Transactions on Geoscience and Remote Sensing*, *36*(4), 1127-1141. doi: 10.1109/36.701020

Williams, M. & Eggleston, S. (2017). *Using indicators to explain our changing climate to policymakers and the public.* Retrieved from https://public.wmo.int/en/resources/bulletin/using-indicators-explain-our-changing-climate-policymakers-and-public

Zare, F., Elsawah, S., Iwanaga, T., Jakeman, A. J., & Pierce, S. A. (2017). Integrated water assessment and modelling: A bibliometric analysis of trends in the water resource sector. *Journal of Hydrology*, *552*, 765-778.

Appendices

Appendix A: Content of the Documentation Page

The following is the content of the documentation page of the WARM tool.

A.1: Dataset Format

Please use the following stepwise guideline as a checklist when uploading a dataset for comparative analysis:

1. Obtain observed climate dataset for region of interest. A sample resource is the POWER Data Access Viewer: https://power.larc.nasa.gov/data-access-viewer/ (POWER, 2021).
2. The downloaded dataset can be a maximum of 11 columns and a minimum of 4 columns. The columns must compose of 3 columns for date (year, month and day), and one column for the climate variable.
3. The WARM tool provides analysis abilities for the year range 1986-2005 inclusive therefore the downloaded dataset can be at maximum 7306 rows (including a header row).
4. The WARM tool only provides analysis for 8 climate variables, other variables will not appear as options if present in the dataset.
5. The headers and units of each of the columns must be as follows:

Description	**Header Name**	**Units**
Year	YEAR	xxxx
Month	MO	xx
Day	DY	xx
Surface downwelling longwave radiation	ALLSKY_SFC_LW_DWN	$\frac{W}{m^2}$
Surface temperature	T2M	℃

Maximum surface temperature	T2M_MAX	℃
Minimum Surface Temperature	T2M_MIN	℃
Maximum surface wind speed	WS10M_MAX	$\frac{km}{h}$
Surface wind speed	WS10M	$\frac{km}{h}$
Surface pressure	PS	kPa
Precipitation	PRECTOTCORR	$\frac{mm}{day}$

6. The data quality of the dataset is not evaluated by the WARM tool. Therefore, the analysis and visualizations will reflect the data quality as is. Please check if your dataset contains nulls, blanks, or cells with the value -999.
7. The dataset must be in csv format and start with the header row (as row 1). Please remove any meta-data if present in the dataset.

The Prediction of Worldwide Energy Resources (POWER) Project. (2021). *FLASHFlux 4 Model Output.* [Data file]. Retrieved from: https://power.larc.nasa.gov/data-access-viewer/

A.2: Deriving Grid Cell Coordinates for CORDEX

In this stepwise guideline, the process of deriving the grid tile latitude and longitude values will be explained. The location for Toronto, Ontario, Canada is shown as an example.

1. Determine the latitude and longitude of the location of interest.

Toronto, Ontario, Canada has a latitude of 43.6532 N and 79.3832 W.

2. Use the online tool, Rotation of Coordinates Based On CORDEX Domains, to determine the rotated latitude and longitude (Kolsoumi & Salehnia, 2019). Link: https://agrimetsoft.com/Cordex%20Coordinate%20Rotation

Change the CORDEX domain to North America. Enter the latitude and longitude for Toronto and select the option to convert non-rotated to rotated. (The latitude and longitude of axis will be generated automatically based on the CORDEX domain). Enter the coordinates with North and East as + and South and West as -. Therefore, the latitude will be 43.6532 and the longitude will be -79.3832. The converted latitude should be -2.43 and the converted longitude will be 12.48. These are known as the rotated latitude and longitude.

3. Download the software Panoply: https://www.giss.nasa.gov/tools/panoply/credits.html (Panoply, 2021).
4. Download any one CORDEX data model file and view it using Panoply. (CCCMA, 2020)

For this example, the 1950 data for the CanRCM4 model was downloaded for the surface temperature variable, which can be found at this link: https://climate-modelling.canada.ca/climatemodeldata/canrcm/CanRCM4/NAM-22_CCCma-CanESM2_historical/day/atmos/tas/index.shtml.

5. Open the file in Panoply by loading the .netcdf file. Use the Create Plot function to create 2 graphs. For the first graph, click on the latitude variable, then click

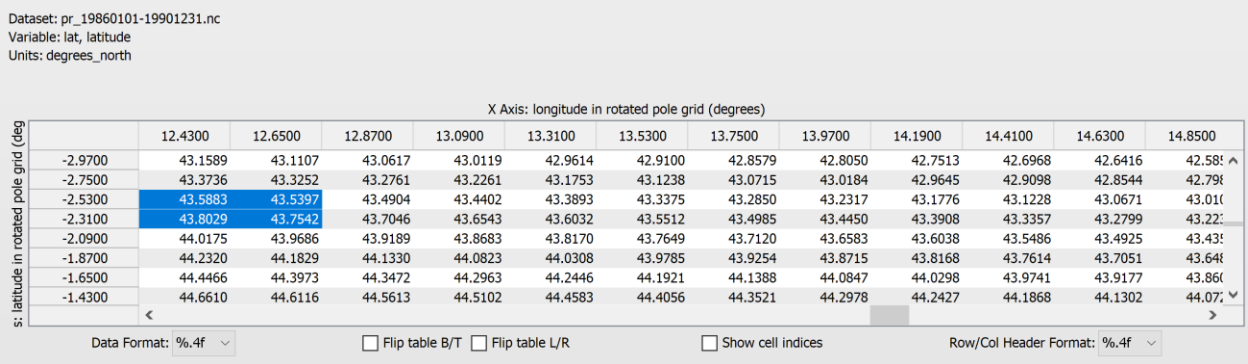
Create Plot, then click georeferenced longitude-latitude colour contour plot. Repeat the same for the longitude variable.

6. In both plots, navigate to the Array 1 section. The y axis contains values for the rotated latitude and x axis contains values for the rotated longitude. Using the rotated coordinates from step 2, find the closest values on the x and y axes in both data tables.

For the Toronto example, the rotated latitude and rotated longitude are -2.43 and 12.48, respectively. The closest values on the y axes are -2.31 and -2.53 and the closest on the x axes are 12.43 and 12.65.

7. The intersection of the closest rotated latitude and longitude values contain decimal degree latitude and longitude. Create combinations of rotated latitude and longitude and their equivalent decimal degree value.

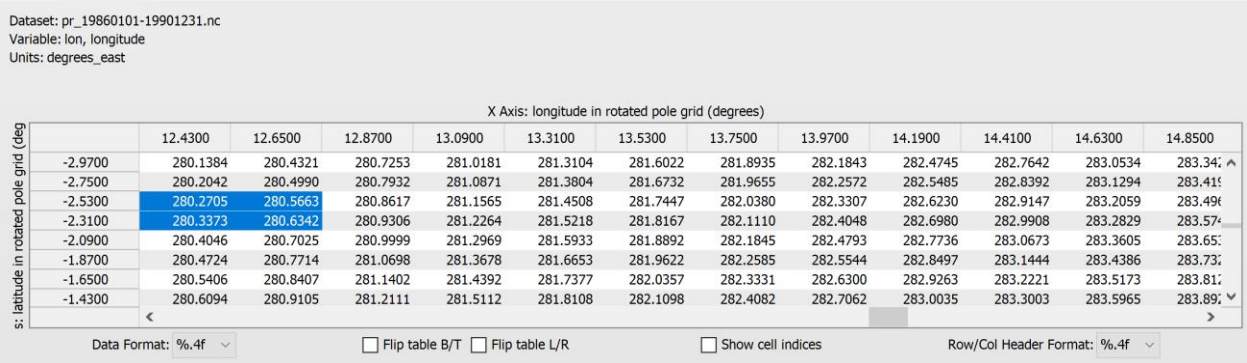
For Toronto, the 2 arrays are:


Dataset: pr_19860101-19901231.nc
Variable: lat, latitude
Units: degrees_north

X Axis: longitude in rotated pole grid (degrees)

s: latitude in rotated pole grid (deg	12.4300	12.6500	12.8700	13.0900	13.3100	13.5300	13.7500	13.9700	14.1900	14.4100	14.6300	14.8500
-2.9700	43.1589	43.1107	43.0617	43.0119	42.9614	42.9100	42.8579	42.8050	42.7513	42.6968	42.6416	42.58
-2.7500	43.3736	43.3252	43.2761	43.2261	43.1753	43.1238	43.0715	43.0184	42.9645	42.9098	42.8544	42.79
-2.5300	43.5883	43.5397	43.4904	43.4402	43.3893	43.3375	43.2850	43.2317	43.1776	43.1228	43.0671	43.01
-2.3100	43.8029	43.7542	43.7046	43.6543	43.6032	43.5512	43.4985	43.4450	43.3908	43.3357	43.2799	43.22
-2.0900	44.0175	43.9686	43.9189	43.8683	43.8170	43.7649	43.7120	43.6583	43.6038	43.5486	43.4925	43.43
-1.8700	44.2320	44.1829	44.1330	44.0823	44.0308	43.9785	43.9254	43.8715	43.8168	43.7614	43.7051	43.64
-1.6500	44.4466	44.3973	44.3472	44.2963	44.2446	44.1921	44.1388	44.0847	44.0298	43.9741	43.9177	43.86
-1.4300	44.6610	44.6116	44.5613	44.5102	44.4583	44.4056	44.3521	44.2978	44.2427	44.1868	44.1302	44.07

Data Format: %.4f Flip table B/T Flip table L/R Show cell indices Row/Col Header Format: %.4f



Therefore, there are 4 rotated latitude and longitude combinations. These are as follows (with the equivalent decimal degree values.

Rotated Lat/Lon	**Decimal Degree Lat/Lon**	**Converted Lat/Lon**
-2.53, 12.43	43.5883, 280.2705	43.5883, -79.7295
-2.31, 12.43	43.8029, 280.3373	43.8029, -79.6627
-2.53, 12.65	43.5397, 280.5663	43.5397, -79.4337
-2.31, 12.65	43.7542, 280.6342	43.7542, -79.3658

8. Use mapping software to determine the closest location (at minimum within 25km of distance) to the original location.

Using Google Maps, the location for Toronto (43.6532, -79.3832) was mapped to each of the 4 options in the table above. The option with the least distance was 43.5883, -79.7295. This maps to -2.53, 12.43 in rotated decimal degrees.

9. Convert the rotated decimal degrees to rotated pole grid values. Navigate to panoply and select the climate variable downloaded earlier. Choose the line plot using time for the horizontal axis graph. In the array section in the lower portion

of the graph, there will be a dropdown for latitude and longitude. Choose the values found from step 8 from the dropdowns. The field will be populated with the grid values.

For Toronto, the grid values are 120 and 212 for latitude and longitude, respectively.

10. Use the grid values as input when using the WARM tool.

Canadian Centre for Climate Modelling and Analysis (CCCMA). (2020). *CanRCM4 Model Output.* [Data file]. Retrieved from https://climate-modelling.canada.ca/climatemodeldata/canrcm/CanRCM4/index_cordex.shtml

Kolsoumi, S. & Salehnia, N. (2019). Rotation of Coordinates Based On CORDEX Domains (1.0). Zenodo. https://doi.org/10.5281/zenodo.2590812

Panoply Data Viewer Version 4.12.7 [Computer software]. (2021). Retrieved from https://www.giss.nasa.gov/tools/panoply/credits.html

Appendix B: Code

The code for the hosted and user data tab is different for the following steps:

- Data quality
- Data extraction from climate model
- Combining data from climate model into 1 file
- Unit conversion
- Mapping observed climate data to climate model predictions
- Calculating bias between observed and predicted values

Code for Hosted: Regional Climate Model Validation

```
#!/usr/bin/env python
# coding: utf-8

# data quality check

import pandas as pd
import datetime as dt
import numpy as np
import csv
import calendar

import dash
import dash_core_components as dcc
import dash_html_components as html

import os,sys,inspect
from dash.dependencies import Input, Output

current_dir =
os.path.dirname(os.path.abspath(inspect.getfile(inspect.currentframe())))
parent_dir = os.path.dirname(current_dir)
sys.path.insert(0, parent_dir)
from environmentVariables import environment_variables

climate=pd.read_excel(user_key['observed_raw_unclean']+'climate-daily-
weatherstats.xlsx',sheet_name=1)
power=pd.read_csv(user_key['observed_raw_unclean']+'POWER-
airportlocation.csv')

#drop all columns in climate except for,
#Date, precipitation, avg_hourly_pressure_station,All Sky Surface Longwave
Downward Irradiance
#ALLSKY_SFC_LW_DWN, avg_hourly_wind_speed,max_wind_speed,
avg_hourly_temperature, max_temperature, min_temperature
```

```
climate_new = pd.DataFrame()
climate_new['Date'] = climate['date']
climate_new['precipitation'] = climate['precipitation']
climate_new['avg_hourly_pressure_station'] =
climate['avg_hourly_pressure_station']
climate_new['avg_hourly_wind_speed'] = climate['avg_hourly_wind_speed']
climate_new['max_wind_speed'] = climate['max_wind_speed']
climate_new['avg_hourly_temperature'] = climate['avg_hourly_temperature']
climate_new['max_temperature'] = climate['max_temperature']
climate_new['min_temperature'] = climate['min_temperature']

power_new = pd.DataFrame()
power_new['Date'] =
pd.to_datetime((power.YEAR*10000+power.MO*100+power.DY).apply(str),format='%Y
%m%d')
power_new['ALLSKY_SFC_LW_DWN'] = power['ALLSKY_SFC_LW_DWN']


#power_new.ALLSKY_SFC_LW_DWN.count(-999)
power_new.eq(0).sum()
power_new.isnull().sum()

climate_new.isnull().sum()

#now we delete leap year columns
climate_new = climate_new[~((climate_new.Date.dt.month == 2) &
(climate_new.Date.dt.day == 29))]

power_new = power_new[~((power_new.Date.dt.month == 2) &
(power_new.Date.dt.day == 29))]

#sort the dates

climate_new = climate_new.sort_values(by="Date")
power_new = power_new.sort_values(by="Date")

climate_new.to_csv(user_key['observed_raw_clean']+'climate-daily-
weatherstats_Cleaned.csv', index=None)
power_new.to_csv(user_key['observed_raw_clean']+'POWER-
airportlocation_Cleaned.csv', index=None)


                    # step1.csv_data_extraction_CRCM

import pandas as pd
import datetime as dt
import numpy as np
import csv
import calendar

import os,sys,inspect
current_dir =
os.path.dirname(os.path.abspath(inspect.getfile(inspect.currentframe())))
parent_dir = os.path.dirname(current_dir)
sys.path.insert(0, parent_dir)
from environmentVariables import environment_variables
```

```
# read files

data=pd.read_csv(user_key['path_in']+'tasmin/tasmin_20010101-20051231.csv',
header=None)
data_time = pd.read_csv(user_key['path_in']+'tasmin/time_20010101-
20051231.csv', header=None)

print(len(data))
# print(data)

# first position for toronto
var = data.iloc[119, 211]
print(var)

# Calculate num of leap years

# Input list initialization
input_list = [1950, 1951, 1952, 1953, 1954, 1955,
         1956, 1957, 1958, 1959, 1960,
         1961, 1962, 1963, 1964, 1965,
         1966, 1967, 1968, 1969, 1970,
         1971, 1972, 1973, 1974, 1975,
         1976, 1977, 1978, 1979, 1980,
         1981, 1982, 1983, 1984, 1985
        ,1986, 1987, 1988, 1989, 1990
        ,1991, 1992, 1993, 1994, 1995
        ,1996, 1997, 1998, 1999, 2000
        ]

def findLeapYear(input_list):
    ans = 0
    for elem in input_list:
        if calendar.isleap(int(elem)):
            ans = ans + 1
    return ans

numleap = findLeapYear(input_list)
print(numleap)

# Extract based on rlat and rlon

rlon_x = 211 # adjusted from 212 since headers start at 0
rlat_y = 119 # adjusted from 120 since index starts at 0
counter = 0

list_of_vars = []


while (rlat_y+counter) < len(data)-1:
    var = data.iloc[rlat_y+counter, rlon_x]
    list_of_vars.append(var)

    counter+=260

len(list_of_vars) #should always be 1825
```

```
#read data_time and transpose into column and append variable and date info
to new df
data_all = data_time.transpose()
data_all.insert(1, 'Variable', list_of_vars)
data_all = data_all.rename(columns={0: 'Date'})

# Change time units

startdate = "12/01/1949"
temp = pd.to_datetime(startdate) + pd.DateOffset(days= 13171.5 +numleap)


# change time units from 'days since 12/01/1949' to actual date
startdate = "12/01/1949"
row_iterator = data_all.iterrows()
counter = 1

for i, row in row_iterator:
    if counter != len(data_all)+1:
        #temp = pd.to_datetime(startdate) + pd.DateOffset(days=data_all.at[i,
'Date']+9)
        #if temp.month != 2 and temp.day != 29:
            #data_all.at[i, 'Date_new'] = pd.to_datetime(temp)
        data_all.at[i, 'Date_new'] = pd.to_datetime(startdate) +
pd.DateOffset(days=data_all.at[i, 'Date']+numleap)
            #data_all.at[i, 'Date_new2'] = pd.to_datetime(data_all.at[i,
'Date_new']) + pd.DateOffset(days=9)

    counter += 1

data_all['Date_new'] = pd.to_datetime(data_all['Date_new']).dt.date

# flag leap years

row_iterator = data_all.iterrows()
counter = 1

for i, row in row_iterator:
    if counter != len(data_all)+1:
        if data_all.at[i, 'Date_new'].month == 2 and data_all.at[i,
'Date_new'].day == 29:
             data_all.at[i, 'flag'] = 'flag'

    counter += 1

# if flag, change flagged date to +1

row_iterator = data_all.iterrows()
counter2 = 1

for i, row in row_iterator:
    if counter2 != len(data_all)+1:
        if data_all.at[i, 'flag'] == 'flag':
            data_all.at[i, 'Date_2'] = data_all.at[i+1, 'Date_new']
    counter2+=1
```

```
# copy over date from before if Date_2 is null
# otherwise keep the value in Date_2

row_iterator = data_all.iterrows()
counter3 = 1
numberofflags = 0

for i, row in row_iterator:
    if counter3 != len(data_all)+1:
        if data_all.at[i, 'flag'] == 'flag':
            numberofflags+=1
        temp = data_all.at[i, 'Date_new'] + pd.DateOffset(days=numberofflags)
        if (temp.month == 2 and temp.day == 29):
            data_all.at[i, 'Date_2'] = data_all.at[i, 'Date_new'] +
pd.DateOffset(days=numberofflags+1)
        else:
            data_all.at[i, 'Date_2'] = data_all.at[i, 'Date_new'] +
pd.DateOffset(days=numberofflags)


    counter3+=1

data_all['Date_2'] = pd.to_datetime(data_all['Date_2']).dt.date

# delete unnecessary columns
del data_all['Date']
del data_all['Date_new']
del data_all['flag']

# rename column
data_all = data_all.rename(columns={'Date_2': 'Date'})

# reorder column
data_all = data_all[['Date', 'Variable']]


data_all.to_csv(user_key['model_raw_unclean']+'tasmin/tasmin_extrTO_2001-
2005.csv', index=None)

data_all.to_csv(user_key['model_raw_unclean']+'tasmin/tasmin_extrTO_2001-
2005.csv', index=None)

                    # step2.combine_years_into_one_var_file

import pandas as pd
import datetime as dt
import numpy as np
import csv
import calendar

import os,sys,inspect
current_dir =
os.path.dirname(os.path.abspath(inspect.getfile(inspect.currentframe())))
parent_dir = os.path.dirname(current_dir)
sys.path.insert(0, parent_dir)
from environmentVariables import environment_variables
# read files
```

```
data_one=pd.read_csv(user_key['model_raw_unclean']+'pr/pr_extrTO_1986-
1990.csv')
data_two=pd.read_csv(user_key['model_raw_unclean']+'pr/pr_extrTO_1991-
1995.csv')
data_three=pd.read_csv(user_key['model_raw_unclean']+'pr/pr_extrTO_1996-
2000.csv')
data_four=pd.read_csv(user_key['model_raw_unclean']+'pr/pr_extrTO_2001-
2005.csv')

# MERGE
frames = [data_one, data_two, data_three, data_four]
result = pd.concat(frames)

result.to_csv(user_key['model_raw_clean']+'pr/pr_all_extrTO_1986-2005.csv',
index=None)
```

```
                            # step3.Unit_Conversion

import pandas as pd
import datetime as dt
import numpy as np
import csv
import calendar

import os,sys,inspect
current_dir =
os.path.dirname(os.path.abspath(inspect.getfile(inspect.currentframe())))
parent_dir = os.path.dirname(current_dir)
sys.path.insert(0, parent_dir)
from environmentVariables import environment_variables

# read files
data=pd.read_csv(user_key['model_raw_clean']+'pr/pr_all_extrTO_1986-
2005.csv')

# conversion equation based on variable name
variableName = "pr"

# loop through the rows creating a new converted column
row_iterator = data.iterrows()
for i, row in row_iterator:
    currVar = (data.at[i,'Variable'])
    if (currVar != "Variable" ):
        if (variableName == "pr"):
            #1 kg/m2/s = 86400 mm/day
            data.at[i, 'Variable_converted'] = (float(currVar) *  (86400))
        elif (variableName == "ps"):
            #1 pa 0.001 kP
             data.at[i, 'Variable_converted'] = (float(currVar) *  (0.001))
        elif (variableName == "rlds"):
            data.at[i, 'Variable_converted'] = (float(currVar))
        elif (variableName == "sfcWind"):
            #1 m/s      3.6km/h
            data.at[i, 'Variable_converted'] = (float(currVar) *  (3.6))
        elif (variableName == "sfcWindmax"):
            #1 m/s      3.6km/h
```

```
            data.at[i, 'Variable_converted'] = (float(currVar) *  (3.6))
        elif (variableName == "tas"):
            # 1K = -273 degrees celsius
            data.at[i, 'Variable_converted'] = (float(currVar) - (273.15))
        elif (variableName == "tasmax"):
            # 1K = -273 degrees celsius
            data.at[i, 'Variable_converted'] = (float(currVar) - (273.15))
        elif (variableName == "tasmin"):
            # 1K = -273 degrees celsius
            data.at[i, 'Variable_converted'] = (float(currVar) - (273.15))


print(data)
# delete unnecessary columns
del data['Variable']

# rename column
data = data.rename(columns={'Variable_converted': 'Variable'})

data.to_csv(user_key['model_raw_clean']+'pr/pr_allCONV_extrTO_1986-2005.csv',
index=None)

                              # step3.5_dq_fillin

import pandas as pd
import datetime as dt
import numpy as np
import csv
import calendar

import os,sys,inspect
current_dir =
os.path.dirname(os.path.abspath(inspect.getfile(inspect.currentframe())))
parent_dir = os.path.dirname(current_dir)
sys.path.insert(0, parent_dir)
from environmentVariables import environment_variables

# read file created by step 3
data=pd.read_csv(user_key['observed_raw_clean']+'climate-daily-
weatherstats_Cleaned.csv')

# read fill in data observed data
dataFillIn=pd.read_csv(user_key['observed_raw_clean']+'DQ
Report/projectedvsobserved_fillin.csv')
# loop through the data file, if the precipitation is null then
# look through the fill in file if it has that date, if so fill in the value
from fill in to the data

# make the date column the index for searching
dataFillIn.set_index('Date',inplace = True)

row_iterator = data.iterrows()
for i, row in row_iterator:
    if (pd.isnull(data.at[i,"precipitation"])):
        print(data.at[i,"Date"])
       # print(dataFillIn.index.values)
        #if null then replace with variable from fill in
```

```
        data.at[i,"precipitation"] =
dataFillIn.loc[data.at[i,"Date"],"precipitation"]
        print(dataFillIn.loc[data.at[i,"Date"],"precipitation"])

#write to file
data.to_csv(user_key['observed_raw_clean']+'climate-daily-
weatherstats_Cleaned.csv', index=None)

                      # step4.mapping_model_to_observed

import pandas as pd
import datetime as dt
import numpy as np
import csv
import calendar

import dash
import dash_core_components as dcc
import dash_html_components as html

import os,sys,inspect
from dash.dependencies import Input, Output

current_dir =
os.path.dirname(os.path.abspath(inspect.getfile(inspect.currentframe())))
parent_dir = os.path.dirname(current_dir)
sys.path.insert(0, parent_dir)
from environmentVariables import environment_variables

#variable (change to read user input)
variable = 'pr'

#use inputs to select file for climate model

# need to figure out the time

data=pd.read_csv(user_key['model_raw_clean']+variable+'/'+variable+'_allCONV_
extrTO_1986-2005.csv')

# read observational climate csv

if variable =='rlds':
    observeData=pd.read_csv(user_key['observed_raw_clean']+'POWER-
airportlocation_Cleaned.csv')
else:
    observeData=pd.read_csv(user_key['observed_raw_clean']+'climate-daily-
weatherstats_Cleaned.csv')

data['Date'] =  pd.to_datetime(data['Date'], format='%Y-%m-%d')

# read column based on the variable


if variable == 'tas':
            #then it is TEMP in the data, find the column
    column = 'avg_hourly_temperature'
elif variable == 'pr' :
```

```
    column = 'precipitation'

elif variable == 'ps' :
    column = 'avg_hourly_pressure_station'

elif variable == 'sfcWind' :
    column = 'avg_hourly_wind_speed'

elif variable == 'sfcWindmax' :
    column = 'max_wind_speed'

elif variable == 'tasmax' :
    column = 'max_temperature'

elif variable == 'tasmin' :
    column = 'min_temperature'

elif variable == 'rlds':
    column = 'ALLSKY_SFC_LW_DWN'


row_iterator = observeData.iterrows()
counter =0

dates = []
observedData = []
modelData = []

#the for loop will loop through the observed data file (weatherstats)
for i, row in row_iterator:
        if counter != len(observeData):
            dates.append(observeData.at[i,'Date'])
            observedData.append(observeData.at[i,column])
            counter += 1

row_ittr2 = data.iterrows()
counter =0
#the for loop will loop through the model data file
for i, row in row_ittr2:
         if counter != len(data):
            modelData.append(data.at[i,'Variable'])
            counter += 1

#now we have all the variables in array now we create a dataframe
data_all = pd.DataFrame()
data_all.insert(0, 'Date', dates)
data_all.insert(1, 'Observed'+variable, observedData)
data_all.insert(2, 'Model'+variable, modelData)

print(data_all)




data_all.to_csv(user_key['projectedvsobserved_mapped']+
variable+'_comparison_output.csv', index=None)
```

```
                                # Step5.biascalc

import pandas as pd
import datetime as dt
import numpy as np
import csv
import calendar

import dash
import dash_core_components as dcc
import dash_html_components as html

import os,sys,inspect
from dash.dependencies import Input, Output

current_dir =
os.path.dirname(os.path.abspath(inspect.getfile(inspect.currentframe())))
parent_dir = os.path.dirname(current_dir)
sys.path.insert(0, parent_dir)
from environmentVariables import environment_variables

variables = ['pr','ps','rlds','sfcWind','sfcWindmax','tas','tasmax','tasmin']

for variable in variables:
    data=pd.read_csv(user_key['projectedvsobserved_mapped']+
variable+'_comparison_output.csv')

    biasData = data['Model'+variable] - data['Observed'+variable]

    data["Bias"] = biasData
    data.to_csv(user_key['projectedvsobservedbias_mapped']+
variable+'_comparison_bias_output.csv', index=None)
```

Code for User-Data: Regional Climate Model Validation

```
#!/usr/bin/env python
# coding: utf-8

import pandas as pd
import datetime as dt
import numpy as np
import csv
import calendar

import os,sys,inspect
current_dir =
os.path.dirname(os.path.abspath(inspect.getfile(inspect.currentframe())))
parent_dir = os.path.dirname(current_dir)
sys.path.insert(0, parent_dir)
from environmentVariables import environment_variables


Userdata=pd.read_csv('AutomatedSteps/userInput.csv', header=[0])
SummaryDF = pd.DataFrame(columns=['Variable'])
```

```
directory_list = list()

for root, dirs, files in os.walk(user_key['path_in'], topdown=False):
    for name in dirs:
        directory_list.append(name)

print(directory_list)

for col in Userdata.columns:
    #if the column name exists in the data variables then add to list

    if col == 'T2M':
        col = 'tas'
    elif col == 'PRECTOTCORR' :
        col = 'pr'
    elif col == 'PS' :
        col = 'ps'
    elif col == 'WS10M' :
        col = 'sfcWind'
    elif col == 'WS10M_MAX' :
        col = 'sfcWindmax'
    elif col == 'T2M_MAX' :
        col = 'tasmax'
    elif col == 'T2M_MIN' :
        col = 'tasmin'
    elif col == 'ALLSKY_SFC_LW_DWN':
        col = 'rlds'
    #col = col.lower()
    print(col)
    if col in directory_list:

        SummaryDF = SummaryDF.append({'Variable':col}, ignore_index=True)

print(SummaryDF)

#logic to create an output for data
#create counts
outputDF = pd.DataFrame(columns=['Column'])
NA = []
blanks = []

for col in Userdata.columns:
    #use each column

    tempNA = 0
    tempBlanks = 0
    outputDF = outputDF.append({'Column': col}, ignore_index=True)
    for index, row in Userdata.iterrows():
        #look at each item in each column
        if Userdata.loc[index,col] == -999 :
            tempNA = tempNA + 1

        if Userdata.loc[index,col] == '' :
            tempBlanks = tempBlanks + 1

    print(tempNA)
    NA.append(tempNA)
```

```
    blanks.append(tempBlanks)

print(NA)
print(outputDF)
print(blanks)
outputDF['Nulls'] = NA
outputDF['Blanks'] = blanks

SummaryDF.to_csv('AutomatedSteps/variableList.csv', index=None)
outputDF.to_csv('AutomatedSteps/outputFeedback.csv', index=None)


                        # step1.csv_data_extraction_CRCM

import pandas as pd
import datetime as dt
import numpy as np
import csv
import calendar

import os,sys,inspect
current_dir =
os.path.dirname(os.path.abspath(inspect.getfile(inspect.currentframe())))
parent_dir = os.path.dirname(current_dir)
sys.path.insert(0, parent_dir)
from environmentVariables import environment_variables

# read files

#we must automate all the files here

#the variable decides which file to select as data
#load in the csv with the selected options
selections=pd.read_csv('AutomatedSteps/optionSelection.csv', header=[0])
#now we allocate all variables (rlat,rlong,variable)
variable = ""
rlat = 0
rlong = 0
for index, row in selections.iterrows():
    if selections.at[index,'option'] == 'rlat':
        rlat = selections.at[index,'selection']
    if selections.at[index,'option'] == 'rlong':
        rlong = selections.at[index,'selection']
    if selections.at[index,'option'] == 'variable':
        variable = selections.at[index,'selection']
    if selections.at[index,'option'] == 'startDate':
        startDate = selections.at[index,'selection']
    if selections.at[index,'option'] == 'endDate':
        endDate = selections.at[index,'selection']
    if selections.at[index,'option'] == 'currYear':
        currYear = selections.at[index,'selection']

#now that selections are set we continue
#decide file based on date
rlat = int(rlat)
rlong = int(rlong)
#now based on how many files exist with variable name and date
```

```
if currYear == '1986':
    print("1986 here")
    dtStr = "19860101-19901231"
    dtOutStr = "1986-1990"
elif currYear == '1991':
    dtStr = "19910101-19951231"
    dtOutStr = "1991-1995"
elif currYear == '1996':
    dtStr = "19960101-20001231"
    dtOutStr = "1996-2000"
elif currYear == '2001':
    dtStr = "20010101-20051231"
    dtOutStr = "2001-2005"

data=pd.read_csv(user_key['path_in']+ variable + '/' + variable + '_' + dtStr
+ '.csv', header=None)
data_time = pd.read_csv(user_key['path_in']+ variable + '/' +'time_' + dtStr
+ '.csv', header=None)

# Input list initialization
input_list = [1950, 1951, 1952, 1953, 1954, 1955,
         1956, 1957, 1958, 1959, 1960,
         1961, 1962, 1963, 1964, 1965,
         1966, 1967, 1968, 1969, 1970,
         1971, 1972, 1973, 1974, 1975,
         1976, 1977, 1978, 1979, 1980,
         1981, 1982, 1983, 1984, 1985
        ,1986, 1987, 1988, 1989, 1990
        ,1991, 1992, 1993, 1994, 1995
        ,1996, 1997, 1998, 1999, 2000
        ]

def findLeapYear(input_list):
    ans = 0
    for elem in input_list:
        if calendar.isleap(int(elem)):
            ans = ans + 1
    return ans

numleap = findLeapYear(input_list)
print(numleap)

# Extract based on rlat and rlon


rlon_x = rlong
rlat_y = rlat
counter = 0

list_of_vars = []


while (rlat_y+counter) < len(data)-1:
    var = data.iloc[rlat_y+counter, rlon_x]
    list_of_vars.append(var)

    counter+=260
```

```
len(list_of_vars)

#read data_time and transpose into column and append variable and date info
to new df
data_all = data_time.transpose()
data_all.insert(1, 'Variable', list_of_vars)
data_all = data_all.rename(columns={0: 'Date'})

# Change time units
# change time units from 'days since 12/01/1949' to actual date
startdate = "12/01/1949"
row_iterator = data_all.iterrows()
counter = 1

for i, row in row_iterator:
    if counter != len(data_all)+1:
        #temp = pd.to_datetime(startdate) + pd.DateOffset(days=data_all.at[i,
'Date']+9)
        #if temp.month != 2 and temp.day != 29:
            #data_all.at[i, 'Date_new'] = pd.to_datetime(temp)
        data_all.at[i, 'Date_new'] = pd.to_datetime(startdate) +
pd.DateOffset(days=data_all.at[i, 'Date']+numleap)
            #data_all.at[i, 'Date_new2'] = pd.to_datetime(data_all.at[i,
'Date_new']) + pd.DateOffset(days=9)

    counter += 1

data_all['Date_new'] = pd.to_datetime(data_all['Date_new']).dt.date


# flag leap years

row_iterator = data_all.iterrows()
counter = 1

for i, row in row_iterator:
    if counter != len(data_all)+1:
        if data_all.at[i, 'Date_new'].month == 2 and data_all.at[i,
'Date_new'].day == 29:
             data_all.at[i, 'flag'] = 'flag'

    counter += 1

# if flag, change flagged date to +1

row_iterator = data_all.iterrows()
counter2 = 1

for i, row in row_iterator:
    if counter2 != len(data_all)+1:
        if data_all.at[i, 'flag'] == 'flag':
            data_all.at[i, 'Date_2'] = data_all.at[i+1, 'Date_new']
    counter2+=1

# copy over date from before if Date_2 is null
# otherwise keep the value in Date_2
```

```
row_iterator = data_all.iterrows()
counter3 = 1
numberofflags = 0

for i, row in row_iterator:
    if counter3 != len(data_all)+1:
        if data_all.at[i, 'flag'] == 'flag':
            numberofflags+=1
        temp = data_all.at[i, 'Date_new'] + pd.DateOffset(days=numberofflags)
        if (temp.month == 2 and temp.day == 29):
            data_all.at[i, 'Date_2'] = data_all.at[i, 'Date_new'] +
pd.DateOffset(days=numberofflags+1)
        else:
            data_all.at[i, 'Date_2'] = data_all.at[i, 'Date_new'] +
pd.DateOffset(days=numberofflags)


    counter3+=1

data_all['Date_2'] = pd.to_datetime(data_all['Date_2']).dt.date


# OUTPUT
# delete unnecessary columns
del data_all['Date']
del data_all['Date_new']
del data_all['flag']

# rename column
data_all = data_all.rename(columns={'Date_2': 'Date'})

# reorder column
data_all = data_all[['Date', 'Variable']]

data_all.to_csv('AutomatedSteps/' + variable +'_'+dtOutStr +'.csv',
index=None)

                    # step2.combine_years_into_one_var_file

import pandas as pd
import datetime as dt
import numpy as np
import csv
import calendar

import os,sys,inspect
current_dir =
os.path.dirname(os.path.abspath(inspect.getfile(inspect.currentframe())))
parent_dir = os.path.dirname(current_dir)
sys.path.insert(0, parent_dir)
from environmentVariables import environment_variables

# read files

#this will be based on variable in the user input
```

```
selections=pd.read_csv('AutomatedSteps/optionSelection.csv', header=[0])


for index, row in selections.iterrows():
    if selections.at[index,'option'] == 'variable':
        variable = selections.at[index,'selection']

data_one=pd.read_csv('AutomatedSteps/'+ variable + '_1986-1990.csv')
data_two=pd.read_csv('AutomatedSteps/'+ variable + '_1991-1995.csv')
data_three=pd.read_csv('AutomatedSteps/'+ variable + '_1996-2000.csv')
data_four=pd.read_csv('AutomatedSteps/'+ variable + '_2001-2005.csv')

# merge
frames = [data_one, data_two, data_three, data_four]
result = pd.concat(frames)


result.to_csv('AutomatedSteps/'+ variable + '.csv', index=None)

#clean up and delete those files
os.remove('AutomatedSteps/'+ variable + '_1986-1990.csv')
os.remove('AutomatedSteps/'+ variable + '_1991-1995.csv')
os.remove('AutomatedSteps/'+ variable + '_1996-2000.csv')
os.remove('AutomatedSteps/'+ variable + '_2001-2005.csv')


                          # step3.Unit_Conversion

import pandas as pd
import datetime as dt
import numpy as np
import csv
import calendar

import os,sys,inspect
current_dir =
os.path.dirname(os.path.abspath(inspect.getfile(inspect.currentframe())))
parent_dir = os.path.dirname(current_dir)
sys.path.insert(0, parent_dir)

# read files
selections=pd.read_csv('AutomatedSteps/optionSelection.csv', header=[0])
for index, row in selections.iterrows():
    if selections.at[index,'option'] == 'variable':
        variable = selections.at[index,'selection']


data=pd.read_csv('AutomatedSteps/'+variable +'.csv')
observedData=pd.read_csv('AutomatedSteps/userInput.csv')

# conversion equation based on variable name
variableName = variable


# loop through the rows creating a new converted column
row_iterator = data.iterrows()
```

```
for i, row in row_iterator:
    currVar = (data.at[i,'Variable'])
    if (currVar != "Variable" ):
        if (variableName == "pr"):
            #1 kg/m2/s = 86400 mm/day
            data.at[i, 'Variable_converted'] = (float(currVar) *  (86400))
        elif (variableName == "ps"):
            #1 pa 0.001 kP
             data.at[i, 'Variable_converted'] = (float(currVar) *  (0.001))
        elif (variableName == "rlds"):
            data.at[i, 'Variable_converted'] = (float(currVar))
        elif (variableName == "sfcWind"):
            #1 m/s      3.6km/h
            data.at[i, 'Variable_converted'] = (float(currVar) *  (3.6))
        elif (variableName == "sfcWindmax"):
            #1 m/s      3.6km/h
            data.at[i, 'Variable_converted'] = (float(currVar) *  (3.6))
        elif (variableName == "tas"):
            # 1K = -273 degrees celsius
            data.at[i, 'Variable_converted'] = (float(currVar) - (273.15))
        elif (variableName == "tasmax"):
            # 1K = -273 degrees celsius
            data.at[i, 'Variable_converted'] = (float(currVar) - (273.15))
        elif (variableName == "tasmin"):
            # 1K = -273 degrees celsius
            data.at[i, 'Variable_converted'] = (float(currVar) - (273.15))


#print(data)

row_iterator2 = observedData.iterrows()
for i, row in row_iterator2:
    if (variableName == "sfcWind"):
        currVar = (observedData.at[i,'WS10M'])
        #1 m/s    3.6km/h
        observedData.at[i, 'WS10M'] = (float(currVar) *  (3.6))
    elif (variableName == "sfcWindmax"):
        currVar = (observedData.at[i,'WS10M_MAX'])
        #1 m/s    3.6km/h
        observedData.at[i, 'WS10M_MAX'] = (float(currVar) *  (3.6))

#print(observedData)

# delete unnecessary columns
del data['Variable']

# rename column
data = data.rename(columns={'Variable_converted': 'Variable'})

data.to_csv('AutomatedSteps/'+variable +'_converted.csv', index=None)
observedData.to_csv('AutomatedSteps/userInput.csv', index=None)

                      # step4.mapping_model_to_observed

import pandas as pd
import datetime as dt
import numpy as np
```

```
import csv
import calendar

import dash
import dash_core_components as dcc
import dash_html_components as html

import os,sys,inspect
from dash.dependencies import Input, Output

current_dir =
os.path.dirname(os.path.abspath(inspect.getfile(inspect.currentframe())))
parent_dir = os.path.dirname(current_dir)
sys.path.insert(0, parent_dir)
from environmentVariables import environment_variables

#use inputs to select file for climate model
selections=pd.read_csv('AutomatedSteps/optionSelection.csv', header=[0])
for index, row in selections.iterrows():
    if selections.at[index,'option'] == 'variable':
        variable = selections.at[index,'selection']


data=pd.read_csv('AutomatedSteps/'+variable+'_converted.csv')

# read observational climate csv provided by user

observeData=pd.read_csv('AutomatedSteps/userInput.csv')

data['Date'] =  pd.to_datetime(data['Date'], format='%Y-%m-%d')

# read column based on the variable


if variable == 'tas':
            #then it is TEMP in the data, find the column
    column = 'T2M'

elif variable == 'pr' :
    column = 'PRECTOTCORR'

elif variable == 'ps' :
    column = 'PS'

elif variable == 'sfcWind' :
    column = 'WS10M'

elif variable == 'sfcWindmax' :
    column = 'WS10M_MAX'

elif variable == 'tasmax' :
    column = 'T2M_MAX'

elif variable == 'tasmin' :
    column = 'T2M_MIN'

elif variable == 'rlds':
```

```
    column = 'ALLSKY_SFC_LW_DWN'


row_iterator = observeData.iterrows()
counter =0

dates = []
observedData = []
modelData = []

#the for loop will loop through the observed data file (weatherstats)
for i, row in row_iterator:
        if counter != len(observeData):
            dtTemp = str(observeData.at[i,'YEAR']) + "-" +
str(observeData.at[i,'MO']) + "-" + str(observeData.at[i,'DY'])
            if(str(observeData.at[i,'MO']) == '2' and
str(observeData.at[i,'DY']) == '29'):
                print("skipped : " + dtTemp)
            else:
                dates.append(dtTemp)
                observedData.append(observeData.at[i,column])
                counter += 1

row_ittr2 = data.iterrows()
counter =0
#the for loop will loop through the model data file
for i, row in row_ittr2:
         if counter != len(data):
            modelData.append(data.at[i,'Variable'])
            counter += 1

#now we have all the variables in array now we create a dataframe
data_all = pd.DataFrame()
data_all.insert(0, 'Date', dates)
data_all.insert(1, 'Observed'+variable, observedData)
data_all.insert(2, 'Model'+variable, modelData)

#print(data_all)

data_all.to_csv('AutomatedSteps/'+variable+'_comparison.csv', index=None)

                                # Step5.biascalc

import pandas as pd
import datetime as dt
import numpy as np
import csv
import calendar

import dash
import dash_core_components as dcc
import dash_html_components as html

import os,sys,inspect
from dash.dependencies import Input, Output
```

```
current_dir =
os.path.dirname(os.path.abspath(inspect.getfile(inspect.currentframe())))
parent_dir = os.path.dirname(current_dir)
sys.path.insert(0, parent_dir

selections=pd.read_csv('AutomatedSteps/optionSelection.csv', header=[0])
for index, row in selections.iterrows():
    if selections.at[index,'option'] == 'variable':
        variable = selections.at[index,'selection']

data=pd.read_csv('AutomatedSteps/'+variable+'_comparison.csv')

biasData = data['Model'+variable] - data['Observed'+variable]

data["Bias"] = biasData
data.to_csv('AutomatedSteps/'+variable+'_comparison_bias.csv', index=None)
```

Code for WARM Interface

```
#!/usr/bin/env python
# coding: utf-8

                                # step6.dash_v1

import pandas as pd
import numpy as np
import csv
import calendar
import plotly.graph_objs as go
import plotly.express as px
from plotly.subplots import make_subplots
import base64

import io
from io import StringIO
from scipy import stats

import dash
import dash_table
import dash_core_components as dcc
import dash_html_components as html
import datetime as dt
import plotly.io as pio
import time

import os,sys,inspect
from dash.dependencies import Input, Output, State
from pandas import ExcelWriter
current_dir =
os.path.dirname(os.path.abspath(inspect.getfile(inspect.currentframe())))
#parent_dir = os.paths.dirname(current_dir)
sys.path.insert(0, current_dir)
from environmentVariables import environment_variables
import plotly
plotly.__version__
```

```
# sys.path.insert(0, 'calculation_folder') # Note: if this relavtive path
doesn't work or produces errors try replacing it with an absolute path
# import calculation


app = dash.Dash(__name__)

#for external style sheets replace ___name__ with the below format
#external_stylesheets = ['https://codepen.io/chriddyp/pen/bWLwgP.css']

#df =
pd.read_csv(user_key['extracted_loc_raw']+'/tas/tas_comparison_output_1991-
1995.csv')


colors = {
    'background': '#E31837',
    'text': '#111111'
}


MIN_YR = 1986
MAX_YR = 2006
#This is the inital layout of the Website
app.layout = html.Div(style={'top-margin' : '0px'}, children=[
    dcc.Store(id='main_data'),
    dcc.Store(id='observed_normals'),
    dcc.Store(id='projected_normals'),
    dcc.Store(id='annual_data'),
    dcc.Store(id='observed_mean'),
    dcc.Store(id='projected_mean'),
    dcc.Store(id='observed_deviation'),
    dcc.Store(id='projected_deviation'),
    dcc.Store(id='observed_variance'),
    dcc.Store(id='projected_variance'),
    dcc.Store(id='bias_data'),
    dcc.Store(id='observed_monthBox'),
    dcc.Store(id='projected_monthBox'),
    dcc.Store(id='observed_yearBox'),
    dcc.Store(id='projected_yearBox'),
    dcc.Store(id='mainFig'),
    dcc.Store(id='projected_median'),
    dcc.Store(id='observed_median'),


    dcc.Store(id='main_data_auto'),
    dcc.Store(id='observed_normals_auto'),
    dcc.Store(id='projected_normals_auto'),
    dcc.Store(id='annual_data_auto'),
    dcc.Store(id='observed_mean_auto'),
    dcc.Store(id='projected_mean_auto'),
    dcc.Store(id='observed_deviation_auto'),
    dcc.Store(id='projected_deviation_auto'),
    dcc.Store(id='observed_variance_auto'),
    dcc.Store(id='projected_variance_auto'),
    dcc.Store(id='bias_data_auto'),
    dcc.Store(id='observed_monthBox_auto'),
```

```
    dcc.Store(id='projected_monthBox_auto'),
    dcc.Store(id='observed_yearBox_auto'),
    dcc.Store(id='projected_yearBox_auto'),
    dcc.Store(id='mainFig_auto'),
    dcc.Store(id='projected_median_auto'),
    dcc.Store(id='observed_median_auto'),


    html.Div([

html.Img(src='https://github.com/eshtab/york_thesisrepo_public/blob/main/Warm
Logo3.jpeg?raw=true',style={'width': '220px','height' : '70px','float' :
'left','margin-left':'0px','padding':'0px'}),

html.Img(src='https://github.com/eshtab/york_thesisrepo_public/blob/main/york
logo2.png?raw=true',style={'height' : '70px','float' : 'right','margin-
right':'0px'})
    ],style = {'backgroundColor' : colors['background'], 'height' : '70px'}
),

    html.Div([
        dcc.Tabs(id='navbar', vertical=True, value='home',  children=[
        dcc.Tab(id="home",label='Home', value='home'),
        dcc.Tab(id="porj",label='Hosted: Regional Climate Model Validation',
value='projVobs'),
        dcc.Tab(id="automate",label='User-Data: Regional Climate Model
Validation', value='auto'),
        dcc.Tab(id="docs",label='Documentation', value='howto'),
        ],parent_style={'float': 'left', 'height' : '90vh', 'width': '220px',
'backgroundColor': '#383838'}, style = {'height' : '600px','color' :
'white','font-family': 'Verdana'},
        colors={
        "border": "white",
        "primary": "white",
        "background": "#383838"
    })


    ]),
    html.Div(id='nav-content')

])

#This is for the vertical tabs, like Home, Docs, etc.

@app.callback(Output('nav-content', 'children'),
              Input('navbar', 'value'))
def render_content(tab):
    if tab == 'home':
        return html.Div([
            html.Img(id='image',src =
'https://github.com/eshtab/york_thesisrepo_public/blob/main/WarmLogoFull.jpeg
?raw=true',style={'height' : '50%','width' : '55%','margin' :'auto','display'
: 'block'} ),
            html.P('The Weather Analysis Regional Model (WARM) is a
visualization tool developed for climatic analysis. It provides the ability
to upload your own observed climate datasets or use the web-hosted datasets
```

```
to perform comparative analysis with a regional climate model, CanRCM4.
Additionally, WARM provides visualizations, data tables, and the ability to
download the underlying data models, and exploratory data analysis
outputs.',style= {'float' : 'left','margin': '15px','color' : 'black','width'
: '50%','height' : '260px','background-color': '#D3D3D3','border-style' :
'solid','border-radius': '25px','font-family': 'Verdana','font-size':
'25px','padding':'20px','text-align':'center','vertical-align': 'middle'})
        ],style = {'background':
'url(https://github.com/eshtab/york_thesisrepo_public/blob/main/FPBackground.
png?raw=true)', 'background-size': 'contain','height': '88vh','background-
position' : 'right','background-repeat' : 'no-repeat'})



    elif tab == 'projVobs':
        return html.Div([
            dcc.Tabs(id='tabs', value='tabs-1', children=[
            dcc.Tab(id="load",label='Load', value='Load'),
            dcc.Tab(id="analyze",label='Analyze',
value='Analyze',disabled=True)
            ],style = {'font-family': 'Verdana'}),
            html.Div(id='tabs-content')
            ])

    elif tab == 'auto':
        return html.Div([
            dcc.Tabs(id='tabs-auto', value='tabs-3', children=[
            dcc.Tab(id="load",label='Load', value='Load'),
            dcc.Tab(id="analyze-auto",label='Analyze', value='Analyze-
auto',disabled=True)
            ],style = {'font-family': 'Verdana'}),
            html.Div(id='tabs-content-auto')
            ])

    elif tab == 'howto':
        return html.Div([
             html.Iframe(src =
"https://docs.google.com/viewer?embedded=true&url=https://github.com/eshtab/y
ork_thesisrepo_public/raw/main/documentationDASH.pdf",
                         style={'height' : '100vh','width' : '80%'})
        ])



#THIS is for Proj v Obsv the horizontal tabs such as Load,Analyze
@app.callback(Output('tabs-content', 'children'),
              Input('tabs', 'value'))
def render_content(tab):
    if tab == 'Load':
        return html.Div([
            html.Form([
                html.H2('Please fill out the options below',style = {'text-
align' : 'Center','font-family': 'Tahoma'}),
                html.Br([]),
                html.H3('Select a region:'),
                 dcc.Dropdown(
```

```
                    id='region',
                    options=[
                            {'label': 'Toronto, Canada', 'value': 'TOR'}
                ]),
                html.Br([]),
                html.H3('Select a variable:'),
                dcc.Dropdown(
                        id='variable',
                        options=[
                            {'label': 'Precipitation', 'value': 'pr'},
                            {'label': 'Pressure', 'value': 'ps'},
                            {'label': 'Surface Downwelling Longwave
Radiation', 'value': 'rlds'},
                            {'label': 'Surface Temperature', 'value': 'tas'},
                            {'label': 'Maximum Surface Temperature', 'value':
'tasmax'},
                            {'label': 'Minimum Surface Temperature', 'value':
'tasmin'},
                            {'label': 'Surface Wind Speed', 'value':
'sfcWind'},
                            {'label': 'Maximum Surface Wind Speed', 'value':
'sfcWindmax'}
                ]),
                html.Br([]),
                html.H3('Select a date range:'),
                dcc.RangeSlider(
                    id="date_slider",
                    min=MIN_YR,
                    max=MAX_YR,
                    count=1,
                    step=1,
                    value=[MIN_YR, MAX_YR],
                    marks={yr: str(yr) for yr in range(MIN_YR, MAX_YR + 1)},
                ),
                dcc.DatePickerRange(
                    id='date-picker-range',
                    min_date_allowed=dt.datetime(1986, 1, 1),
                    max_date_allowed=dt.datetime(2006, 1, 1),
                    display_format="MMM D, YYYY",
                    style={'width' : '100%'}
                    #observed vs projected #daterange should be diff based on
variable

                ),
                html.P(id = "displayDate"),
                html.Br([]),
                html.Br([]),
                html.Button('Submit', id='submit-val', type = 'button',
n_clicks=0,style={'background-color': '#4CAF50','border': 'none','color':
'white','padding': '15px 32px','text-align': 'center','text-decoration':
'none','display': 'inline-block','font-size': '16px','margin': '4px 2px'})


        ],id = 'LoadForm'),

        html.Form([ html.H3('Instructions',style={'font-family': 'Verdana'}),
```

```
                html.P('1. Enter selection for region.',style = {'text-align'
: 'Left'}),
                html.P('2. Enter selection for variable.',style = {'text-
align' : 'Left'}),
                html.P('3. Enter a start and end year using the slider. Use
the calendar to choose specific dates or enter the dates by typing.',style =
{'text-align' : 'Left'}),
                html.P('4. Click on submit to view graphs. Note that clicking
submit will automatically redirect to the Analyze tab.',style = {'text-align'
: 'Left'})

                  ],id = 'LoadInstruction')
        ],id = 'LoadDiv' )


#this is for the automation tab
@app.callback(Output('tabs-content-auto', 'children'),
              Input('tabs-auto', 'value'))
def render_content(tab):
    if tab == 'Load':
        return html.Div([
            html.Form([
                html.H2('Please fill out the options below',style = {'text-
align' : 'Center','font-family': 'Tahoma'}),
                html.Br([]),
                html.H3('Select a CORDEX region:'),
                dcc.Dropdown(
                    id='region-auto',
                    options=[
                            {'label': 'North America', 'value': 'NA'}
                ]),
                html.Br([]),
                html.H3('Enter Grid Coordinates:'),
                html.Br([]),
                dcc.Input(
                    id = 'lat',
                    type="number",
                    step = 1,
                    placeholder="Latitude",
                    style = {'height' : '30px'}),
                dcc.Input(
                    id = 'long',
                    type="number",
                    step = 1,
                    placeholder="Longitude",
                    style = {'margin-left' : '20px','height' : '30px'}),

                html.Br([]),
                html.H3('Name your Data:'),
                dcc.Input(
                    id='nameData-auto',
                    type="text",
                    style = {'height' : '30px'}
                ),
                html.Br([]),
                html.H3('Upload Dataset:'),
                dcc.Upload(
```

```
                    id='upload-data-auto',
                    children=html.Div([
                        'Drag and Drop or ',
                        html.A('Select Files')
                    ]),
                    style={
                        'width': '100%',
                        'height': '60px',
                        'lineHeight': '60px',
                        'borderWidth': '1px',
                        'borderStyle': 'dashed',
                        'borderRadius': '5px',
                        'textAlign': 'center',
                        'margin': '10px'
                    },
                    # Allow multiple files to be uploaded
                    multiple=False
                ),
                html.Div(id='upload-Output'),
                html.H3('Select a date range:'),
                dcc.RangeSlider(
                    id="date_slider-auto",
                    min=MIN_YR,
                    max=MAX_YR,
                    count=1,
                    step=1,
                    value=[MIN_YR, MAX_YR],
                    marks={yr: str(yr) for yr in range(MIN_YR, MAX_YR + 1)},
                ),
                dcc.DatePickerRange(
                    id='date-picker-range-auto',
                    min_date_allowed=dt.datetime(1986, 1, 1),
                    max_date_allowed=dt.datetime(2006, 1, 1),
                    display_format="MMM D, YYYY",
                    style={'width' : '100%'}
                    #observed vs projected #daterange should be diff based on
variable

                ),
                html.P(id = "displayDate-auto"),
                html.Br([]),
                html.Br([]),
                #after the submit button is hit
                #the code will execute the steps  to extract the rlat and
rlong and everything else using the variable chosen and provided rlat and
rlong
                #they will then provide the proper excel output to be
displayed
                html.Button('Submit', id='submit-val-auto', type = 'button',
n_clicks=0,style = {'background-color': '#4CAF50','font-size' : '20px'})


        ],id = 'LoadForm',style = {'font-family': 'Tahoma'}),

        html.Form([ html.H3('Instructions',style={'font-family': 'Verdana'}),
                html.P('1. Enter selection for climate model region.',style =
{'text-align' : 'Left'}),
```

```
                html.P('2. Enter selection for climate model latitude and
longitude in grid format. For details on how to calculate the grid location
values for a region, please visit the documentation tab.',style = {'text-
align' : 'Left'}),
                html.P('3. Enter text to name the csv file that is available
for download after submitting selections.',style = {'text-align' : 'Left'}),
                html.P('4. Upload the observed climate dataset by clicking on
the field or by using drag-and-drop. For details on how to format your
dataset, please visit the documentation tab.',style = {'text-align' :
'Left'}),
                html.P('5. Enter selection for variable (these are populated
based on the dataset provided).',style = {'text-align' : 'Left'}),
                html.P('6. Enter a start and end year using the slider. Use
the calendar to choose specific dates or enter the dates by typing.',style =
{'text-align' : 'Left'}),
                html.P('7. Click on submit to view graphs. Note that clicking
submit will automatically redirect to the Analyze tab.',style = {'text-align'
: 'Left'})

                  ],id = 'LoadInstruction')
        ],id = 'LoadDiv' )



@app.callback(Output('upload-Output', 'children'),
              Input('upload-data-auto', 'contents'),
              State('upload-data-auto', 'filename'),
              State('upload-data-auto', 'last_modified'))
def update_output(list_of_contents, filename, list_of_dates):
    #print(1)
    if list_of_contents is not None:
        #print(2)
        content_type, content_string = list_of_contents.split(',')
        decoded = base64.b64decode(content_string)
        try:

            if 'csv' in filename:

                 #Delete all csv files in the Automatedsteps folder
                dir_name = "AutomatedSteps/"
                files = os.listdir(dir_name)

                for file in files:
                    if file.endswith(".csv"):
                        os.remove(os.path.join(dir_name, file))

                #IF user uploads a csv then we continue with steps
                #next step is to save the uploaded file
                #the steps will run and create the needed csvs
                #then there will be a printed out csv with results from the
upload which we can display below
                #os.remove("AutomatedSteps/userInput.csv")
                #first save the user upload to file
                df = pd.read_csv(io.StringIO(decoded.decode('utf-8')))
                #print(df)
                df.to_csv('AutomatedSteps/userInput.csv', index=None)
```

```
                #this step will run and extract columns
                get_ipython().run_line_magic('run',
'./AutomatedSteps/Step0.ipynb')

                display = pd.read_csv('AutomatedSteps/variableList.csv')
                display2 = pd.read_csv('AutomatedSteps/outputFeedback.csv')

                #finally display the output
                return html.Div([
                html.H3('Select a variable:'),
                #this variable selection is made by the user input csv
                dcc.Dropdown(
                        id='variable-auto',
                        options = [{'label': row_value, 'value': row_value}
for i,row_value in display['Variable'].iteritems()]
                ),
                html.H3('Data Quality of Input:'),
                dash_table.DataTable(
                id='tableFeedback',
                columns=[{"name": i, "id": i} for i in display2.columns],
                data=display2.to_dict('records')
            )
                ])
            else:

                return html.Div([
                'The file is not a csv, please upload a csv file'
            ])
        except Exception as e:
            print(e)
            return html.Div([
                'There was an error processing this file: '+ filename
            ])

#This is the AUtomation BUTTON
#This is here for calculating the Model vs Observed graph and outputting it
to the DIV
@app.callback(
    Output("tabs-auto", "value"),
    Output("analyze-auto", "children"),
    Output('main_data_auto','data'),
    Output('observed_normals_auto','data'),
    Output('projected_normals_auto','data'),
    Output('annual_data_auto','data'),
    Output('observed_mean_auto','data'),
    Output('projected_mean_auto','data'),
    Output('observed_deviation_auto','data'),
    Output('projected_deviation_auto','data'),
    Output('observed_variance_auto','data'),
    Output('projected_variance_auto','data'),
    Output('bias_data_auto','data'),
    Output('observed_monthBox_auto','data'),
    Output('projected_monthBox_auto','data'),
    Output('observed_yearBox_auto','data'),
    Output('projected_yearBox_auto','data'),
    Output('mainFig_auto','data'),
```

```
    [Input("submit-val-auto", "n_clicks")],
    state=[State(component_id='region-auto',
component_property='value'),State('variable-auto', 'value'), State('date-
picker-range-auto', 'start_date'),State('date-picker-range-auto',
'end_date'),State('lat', 'value'),State('long', 'value')])
def
tab_resources(click,region,variable,startDatePicked,endDatePicked,rlat,rlong)
:
    #print("testing1234")
    if click:
        startDate = dt.datetime.strptime(startDatePicked,'%Y-%m-%d')

        endDate = dt.datetime.strptime(endDatePicked,'%Y-%m-%d') -
dt.timedelta(days=1)


        #HERE WE RUN THE STEPS
        #we need to use rlat rlong and all other variables provided
        #in order to run the steps
        #lets save a csv file with the variables
        selections = pd.DataFrame(columns=['option','selection'])
        selections = selections.append({'option':'rlat','selection':rlat},
ignore_index=True)
        selections = selections.append({'option':'rlong','selection':rlong},
ignore_index=True)
        selections =
selections.append({'option':'variable','selection':variable},
ignore_index=True)
        selections =
selections.append({'option':'startDate','selection':startDate},
ignore_index=True)
        selections =
selections.append({'option':'endDate','selection':endDate},
ignore_index=True)
        selections = selections.append({'option':'currYear','selection':0},
ignore_index=True)

        #loop 4 times and create the files
        for i in range(1986,2005,5):
            #create the csv with variables
            #selections.loc[selections['option'] = 'currYear', ['selection']]
= i
            selections.at[5,'selection']=i
            #print(selections)
            selections.to_csv('AutomatedSteps/optionSelection.csv',
index=None)
            get_ipython().run_line_magic('run',
'./AutomatedSteps/step1.csv_data_extraction_CRCM.ipynb')

        #run step 2- merge files
        get_ipython().run_line_magic('run',
'./AutomatedSteps/step2.combine_years_into_one_var_file.ipynb')

        #run step 3 unit conversion
        get_ipython().run_line_magic('run',
'./AutomatedSteps/step3.Unit_Conversion.ipynb')
```

```
        #run step 4 model mapped to observed
        get_ipython().run_line_magic('run',
'./AutomatedSteps/step4.mapping_model_to_observed.ipynb')

        #step 5 add bias
        get_ipython().run_line_magic('run',
'./AutomatedSteps/Step5.biascalc.ipynb')

        #variable (change to read user input)
        variable = variable


        if variable == 'tas':

            column = 'T2M'
            columnName = 'Surface Temperature'
            measurement = '(C)'

        elif variable == 'pr' :
            column = 'PRECTOTCORR'
            columnName = 'Precipitation'
            measurement = '(mm)'

        elif variable == 'ps' :
            column = 'PS'
            columnName = 'Surface Pressure'
            measurement = '(kPa)'

        elif variable == 'sfcWind' :
            column = 'WS10M'
            columnName = 'Surface Wind Speed'
            measurement = '(km/h)'

        elif variable == 'sfcWindmax' :
            column = 'WS10M_MAX'
            columnName = 'Maximum Surface Wind Speed'
            measurement = '(km/h)'
        elif variable == 'tasmax' :
            column = 'T2M_MAX'
            columnName = 'Maximum Surface Temperature'
            measurement = '(C)'
        elif variable == 'tasmin' :
            column = 'T2M_MIN'
            columnName = 'Minimum Surface Temperature '
            measurement = '(C)'
        elif variable == 'rlds':
            column = 'ALLSKY_SFC_LW_DWN'
            columnName = 'Surface Downwelling Longwave Radiation'
            measurement = '(W m-2)'



        #THIS IS WHERE WE AUTOMATE IT, use the automated generated file
        data = pd.read_csv('AutomatedSteps/'+variable+'_comparison_bias.csv')

#         mask = (data['Date'] >= str(startDate.date())) & (data['Date'] <=
str(endDate.date()))
```

```
#         #only use the data in between the user given range
#         data_all = data.loc[mask]
#        mask = (data['Date'] >= str(startDate.date())) & (data['Date'] <=
str(endDate.date()))

#        data_all = data.loc[mask]
        data['Date'] = pd.to_datetime(data.Date)
        data['Date'] = data['Date'].dt.strftime('%Y-%m-%d')
        print(data)

        data_all = data[data['Date'].between(str(startDate.date()),
str(endDate.date()),inclusive='both')]
        print(data_all)
        #create the plot figures
        fig = make_subplots(rows=2, cols=1, shared_xaxes=False)

        fig.add_trace(go.Scatter(x=data_all['Date'],
y=data_all['Model'+variable],mode = 'lines+markers', name =
"Predicted",xhoverformat="%d %b, %Y" ,yhoverformat = ".2f"))
        fig.add_trace(go.Scatter(x=data_all['Date'],
y=data_all['Observed'+variable],mode = 'lines+markers',name =
"Observed",xhoverformat="%d %b, %Y" ,yhoverformat = ".2f"))

        #Update the plot layot with titles
        fig.update_layout(
            title_text="Comparison of Predicted and Observed Climate",
            title_x=0.5,
            xaxis_title="Date",
            yaxis_title=columnName + ' ' + measurement,
            showlegend=True,
            autosize=False,
            width=1300,
            height=700,

#             margin=dict(l=5,r=5,b=0,t=5,pad=5)
            margin=go.layout.Margin(
                l=0,
                r=0,
                b=25,
                t=25,
                pad = 0
                )
            )


        #Create the Analysis to be placed under the graph
        #observed frequency
        stats1 = pd.DataFrame(columns=['Month'])

        #projected frequency
        stats2 = pd.DataFrame(columns=['Month'])

        #observed tendency
        stats3 = pd.DataFrame(columns=['Month'])

        #projected tendency
        stats4 = pd.DataFrame(columns=['Month'])
```

```
        #seasonal observed
        seasonalDF = pd.DataFrame(columns=['Month'])

        #seasonal project
        seasonalDF2 = pd.DataFrame(columns=['Month'])

        #standardDeviation Observed
        stats5 = pd.DataFrame(columns=['Month'])

        #standard deviation Projected
        stats6 = pd.DataFrame(columns=['Month'])


        #variance observed
        stats7 = pd.DataFrame(columns=['Month'])

        #variance projected
        stats8 = pd.DataFrame(columns=['Month'])


        #Bias
        biasDF = pd.DataFrame(columns=['Month'])

       #monthly median stats observed
        medianMonthlyObs = pd.DataFrame(columns=['Month'])

        #monthly median stats predicted
        medianMonthlyProj = pd.DataFrame(columns=['Month'])


        #get number of years in the date range and create them as columns
        for i in range(startDate.year,(endDate.year+1)):
            stats1[i] = ""
            stats2[i] = ""
            stats3[i] = ""
            stats4[i] = ""
            stats5[i] = ""
            stats6[i] = ""
            stats7[i] = ""
            stats8[i] = ""
            medianMonthlyObs[i] = ""
            medianMonthlyProj[i] = ""
            biasDF[i] = ""
            seasonalDF[i] = ""
            seasonalDF2[i] = ""

        #fill in the months Column
        months =
['January','February','March','April','May','June','July','August','September
','October','November','December']
        for i in months:
            stats1 = stats1.append({'Month': i}, ignore_index=True)
            stats2 = stats2.append({'Month': i}, ignore_index=True)
            stats3 = stats3.append({'Month': i}, ignore_index=True)
            stats4 = stats4.append({'Month': i}, ignore_index=True)
```

```
            stats5 = stats5.append({'Month': i}, ignore_index=True)
            stats6 = stats6.append({'Month': i}, ignore_index=True)
            stats7 = stats7.append({'Month': i}, ignore_index=True)
            stats8 = stats8.append({'Month': i}, ignore_index=True)
            medianMonthlyObs = medianMonthlyObs.append({'Month': i},
ignore_index=True)
            medianMonthlyProj = medianMonthlyProj.append({'Month': i},
ignore_index=True)
            biasDF = biasDF.append({'Month': i}, ignore_index=True)
            seasonalDF = seasonalDF.append({'Month': i}, ignore_index=True)
            seasonalDF2 = seasonalDF2.append({'Month': i}, ignore_index=True)

        #fill in the rows with 0/0

        stats1.fillna("0/0", inplace=True)
        stats2.fillna("0/0", inplace=True)
        stats3.fillna("0", inplace=True)
        stats4.fillna("0", inplace=True)
        stats5.fillna("0", inplace=True)
        stats6.fillna("0", inplace=True)
        stats7.fillna("0", inplace=True)
        stats8.fillna("0", inplace=True)
        medianMonthlyObs.fillna("0", inplace=True)
        medianMonthlyProj.fillna("0", inplace=True)
        biasDF.fillna("0/0", inplace=True)
        seasonalDF.fillna("0", inplace=True)
        seasonalDF2.fillna("0", inplace=True)


        #now loop through each row in data_all, and
        #THIS IS FOR FREQUENCY for Stats 1 and 2
        rowit = data_all.iterrows()

        for i, row in rowit:

            #now for each row we check the month
            #then we declare our normal based on the month
            #then we compare with the normal to determine which count it goes
in
            currDate = dt.datetime.strptime(data_all.at[i,'Date'], '%Y-%m-
%d')

            currMonth = currDate.month - 1

            currYear = currDate.year

            newValObs = data_all.at[i,'Observed'+variable]
            newValProj = data_all.at[i,'Model'+variable]

            newBias = data_all.at[i,'Bias']


            if(currMonth == 0):
                normal = 48.67
            elif(currMonth == 1):
                normal = 47.7
            elif(currMonth == 2):
```

```
    normal = 49.8
elif(currMonth == 3):
    normal = 68.5
elif(currMonth == 4):
    normal = 74.3
elif(currMonth == 5):
    normal = 71.5
elif(currMonth == 6):
    normal = 75.7
elif(currMonth == 7):
    normal = 78.1
elif(currMonth == 8):
    normal = 74.5
elif(currMonth == 9):
    normal = 61.1
elif(currMonth == 10):
    normal = 75.1
elif(currMonth == 11):
    normal = 57.9


currValObs = stats1.at[(currMonth),currYear]

overObs = currValObs.split("/",1)[0]
underObs = currValObs.split("/",1)[1]

currValProj = stats2.at[(currMonth),currYear]

overProj = currValProj.split("/",1)[0]
underProj = currValProj.split("/",1)[1]


currentBias = biasDF.at[(currMonth),currYear]
posBias = currentBias.split("/",1)[0]
negBias = currentBias.split("/",1)[1]

#the bias threshold is based on variable
biasThreshold = 0

if variable == 'tas':
    biasThreshold = 1
elif variable == 'pr' :
    biasThreshold = 1

elif variable == 'ps' :
    biasThreshold = 0.5

elif variable == 'sfcWind' :
    biasThreshold = 5

elif variable == 'sfcWindmax' :
    biasThreshold = 5

elif variable == 'tasmax' :
    biasThreshold = 1

elif variable == 'tasmin' :
```

```
                biasThreshold = 1



            if newBias > biasThreshold:
                posBias = int(posBias) + 1
            elif newBias < (biasThreshold * -1):
                negBias = int(negBias) + 1
            else:
                posBias = int(posBias)
                negBias = int(negBias)


            if newValObs > normal:
                overObs = int(overObs) + 1
            else:
                underObs = int(underObs) + 1

            if newValProj > normal:
                overProj = int(overProj) + 1
            else:
                underProj = int(underProj) + 1


            #overwrite the value
            stats1.at[(currMonth),currYear] = str(overObs) + "/" +
str(underObs)
            stats2.at[(currMonth ),currYear] = str(overProj) + "/" +
str(underProj)


            #seasonal data
            seasonalDF.at[(currMonth ),currYear] = newValObs
            seasonalDF2.at[(currMonth ),currYear] = newValProj

            #bias
            biasDF.at[(currMonth ),currYear] = str(posBias) + "/" +
str(negBias)


        #CENTRAL TENDENCY

        temp_monthly = data_all
        temp_monthly['Date'] = pd.to_datetime(temp_monthly['Date'])
        temp_monthly = (temp_monthly.set_index('Date').resample('M').mean())

        temp_monthly_median = data_all
        temp_monthly_median['Date'] =
pd.to_datetime(temp_monthly_median['Date'])
        temp_monthly_median =
(temp_monthly_median.set_index('Date').resample('M').median())


        temp_yearly = data_all
        temp_yearly['Date'] = pd.to_datetime(temp_yearly['Date'])
```

```
        temp_yearly = (temp_yearly.set_index('Date').resample('Y').mean())

        temp_yearly_median = data_all
        temp_yearly_median['Date'] =
pd.to_datetime(temp_yearly_median['Date'])
        temp_yearly_median =
(temp_yearly_median.set_index('Date').resample('Y').median())

        #now that we have monthly and yearly resampled, place them in matrix
        stats3 = stats3.append({'Month': 'Annual'}, ignore_index=True)
        stats4 = stats4.append({'Month': 'Annual'}, ignore_index=True)

        #monthly median in dataframe
        medianMonthlyObs = medianMonthlyObs.append({'Month': 'Annual'},
ignore_index=True)
        medianMonthlyProj = medianMonthlyProj.append({'Month': 'Annual'},
ignore_index=True)

        rowit = temp_monthly.iterrows()

        row_median = temp_monthly_median.iterrows()

        temp_monthly['Date'] = temp_monthly.index
        temp_monthly_median['Date'] = temp_monthly_median.index

        temp_monthly = temp_monthly.round({'Observed'+variable: 2,
'Model'+variable: 2})
        temp_monthly_median = temp_monthly_median.round({'Observed'+variable:
2, 'Model'+variable: 2})
        temp_yearly = temp_yearly.round({'Observed'+variable: 2,
'Model'+variable: 2})
        temp_yearly_median = temp_yearly_median.round({'Observed'+variable:
2, 'Model'+variable: 2})


        #add monthly means
        for i, row in rowit:
            currDate = temp_monthly.at[i,'Date']

            currMonth = currDate.month

            currYear = currDate.year


            stats3.at[(currMonth),currYear] =
temp_monthly.at[i,'Observed'+variable]
            stats4.at[(currMonth),currYear] =
temp_monthly.at[i,'Model'+variable]



         #add monthly median
        for i, row in row_median:
            currDate = temp_monthly_median.at[i,'Date']

            currMonth = currDate.month
```

```
            currYear = currDate.year


            medianMonthlyObs.at[(currMonth ),currYear] =
temp_monthly_median.at[i,'Observed'+variable]
            medianMonthlyProj.at[(currMonth),currYear] =
temp_monthly_median.at[i,'Model'+variable]


        temp_yearly['Date'] = temp_yearly.index
        rowitYear = temp_yearly.iterrows()



        #add monthly means
        for i, row in rowitYear:
            currDate = temp_yearly.at[i,'Date']

            currMonth = currDate.month

            currYear = currDate.year


            stats3.at[12,currYear] = temp_yearly.at[i,'Observed'+variable]
            stats4.at[12,currYear] = temp_yearly.at[i,'Model'+variable]







        temp_yearly_median['Date'] = temp_yearly_median.index
        rowitYear_median = temp_yearly_median.iterrows()
         #add yearly medians
        for i, row in rowitYear_median:
            currDate = temp_yearly_median.at[i,'Date']

            currMonth = currDate.month

            currYear = currDate.year


            medianMonthlyObs.at[12,currYear] =
temp_yearly_median.at[i,'Observed'+variable]
            medianMonthlyProj.at[12,currYear] =
temp_yearly_median.at[i,'Model'+variable]


        #STANDARD DEVIATION

        stats5 = stats5.append({'Month': 'Annual'}, ignore_index=True)
        stats6 = stats6.append({'Month': 'Annual'}, ignore_index=True)

        temp_monthly_std = data_all
        temp_monthly_std['Date'] = pd.to_datetime(temp_monthly_std['Date'])
```

```
        temp_monthly_std =
(temp_monthly_std.set_index('Date').resample('M').std())


        temp_yearly_std = data_all
        temp_yearly_std['Date'] = pd.to_datetime(temp_yearly_std['Date'])
        temp_yearly_std =
(temp_yearly_std.set_index('Date').resample('Y').std())


        rowit = temp_monthly_std.iterrows()

        temp_monthly_std['Date'] = temp_monthly_std.index

        temp_monthly_std = temp_monthly_std.round({'Observed'+variable: 2,
'Model'+variable: 2})
        temp_yearly_std = temp_yearly_std.round({'Observed'+variable: 2,
'Model'+variable: 2})

        #add monthly means
        for i, row in rowit:
            currDate = temp_monthly_std.at[i,'Date']

            currMonth = currDate.month

            currYear = currDate.year


            stats5.at[(currMonth),currYear] =
temp_monthly_std.at[i,'Observed'+variable]
            stats6.at[(currMonth),currYear] =
temp_monthly_std.at[i,'Model'+variable]

        temp_yearly_std['Date'] = temp_yearly_std.index
        rowitYear = temp_yearly_std.iterrows()
        #add monthly means
        for i, row in rowitYear:
            currDate = temp_yearly_std.at[i,'Date']

            currMonth = currDate.month

            currYear = currDate.year


            stats5.at[12,currYear] =
temp_yearly_std.at[i,'Observed'+variable]
            stats6.at[12,currYear] = temp_yearly_std.at[i,'Model'+variable]


        #add monthly means
        for i, row in rowit:
            currDate = temp_monthly_var.at[i,'Date']

            currMonth = currDate.month

            currYear = currDate.year
```

```
            stats7.at[(currMonth),currYear] =
temp_monthly_var.at[i,'Observed'+variable]

        #Anual Precip
        temp_yearly_precip = data_all
        temp_yearly_precip['Date'] =
pd.to_datetime(temp_yearly_precip['Date'])
        temp_yearly_precip =
(temp_yearly_precip.set_index('Date').resample('Y').sum())

        temp_yearly_precip['Date'] = temp_yearly_precip.index

        #anual precip observed
        anualPrecip = pd.DataFrame()

        anualPrecip['Year'] = temp_yearly_precip['Date'].dt.year
        anualPrecip['Observed'] =  temp_yearly_precip['Observed'+variable]
        anualPrecip['Projected'] =  temp_yearly_precip['Model'+variable]

        anualPrecip = anualPrecip.round({'Observed': 2, 'Projected': 2})

        #box plots
        #create a df for both positive and negative

        #firstly pull the data
        bias=pd.read_csv(user_key['projectedvsobservedbias_mapped']+
variable+'_comparison_bias_output.csv')

        bias['Date'] = pd.to_datetime(bias['Date'])
        bias.reset_index(inplace=True)


        # do bias data
        bias['year'] = [dt.datetime.year for d in bias.Date]
        bias['month'] = [dt.datetime.month for d in bias.Date]
        years = bias['year'].unique()

        # Draw box plots
        #to do this we must create a new column listing month and year
        #this is the monthly breakdown
        temp_box = data_all

        temp_box['month_year'] = temp_box['Date'].dt.strftime('%Y-%m')

        boxfig1 = px.box(temp_box,x = "month_year", y = "Observed" +variable)

        boxfig2 = px.box(temp_box,x = "month_year", y = "Model" +variable)

        #this is for the year
        temp_box_year = data_all
        temp_box_year['year'] = temp_box_year['Date'].dt.strftime('%Y')

        boxfig3 = px.box(temp_box_year,x = "year", y = "Observed" +variable)

        boxfig4 = px.box(temp_box_year,x = "year", y = "Model" +variable)
```

```
#print(boxfig4.hoverinfo)

boxfig1.update_xaxes(
    title = "Month, Year"
)
boxfig1.update_yaxes(
    title = columnName + " " +   measurement
)


boxfig2.update_xaxes(
    title = "Month, Year"
)
boxfig2.update_yaxes(
    title = columnName + " " +   measurement
)



boxfig3.update_xaxes(
    title = "Year"
)
boxfig3.update_yaxes(
    title = columnName + " " +   measurement
)

boxfig4.update_xaxes(
    title = "Year"
)
boxfig4.update_yaxes(
    title = columnName + " " +   measurement
)

#print(boxfig3.hover_data())


tempdataMonth = data_all
tempdataMonth['Date'] = pd.to_datetime(tempdataMonth['Date'])
tempdataMonth = (tempdataMonth.set_index('Date').resample('M').var())
tempdataMonth['Date'] = tempdataMonth.index
#create the data for the box figures


#use temp_box for monthwise
temp_monthly_box = pd.DataFrame()

temp_box = data_all
temp_box['Date'] = pd.to_datetime(temp_box['Date'])
temp_box = (temp_box.set_index('Date'))
temp_box['Date'] = temp_box.index

date_monthly_obs = tempdataMonth['Date']
```

```
        median_monthly = temp_box["Observed"
+variable].groupby(pd.Grouper(freq="M")).median()
        first_monthly = temp_box["Observed"
+variable].groupby(pd.Grouper(freq="M")).quantile(0.25)
        third_monthly = temp_box["Observed"
+variable].groupby(pd.Grouper(freq="M")).quantile(0.75)
        tempOBSBOX = temp_box["Observed"
+variable].groupby(pd.Grouper(freq="M"))

        median_monthly_proj = temp_box["Model"
+variable].groupby(pd.Grouper(freq="M")).median()
        first_monthly_proj = temp_box["Model"
+variable].groupby(pd.Grouper(freq="M")).quantile(0.25)
        third_monthly_proj = temp_box["Model"
+variable].groupby(pd.Grouper(freq="M")).quantile(0.75)


        median_yearly = temp_box["Observed"
+variable].groupby(pd.Grouper(freq="Y")).median()
        first_yearly = temp_box["Observed"
+variable].groupby(pd.Grouper(freq="Y")).quantile(0.25)
        third_yearly = temp_box["Observed"
+variable].groupby(pd.Grouper(freq="Y")).quantile(0.75)

        median_yearly_proj = temp_box["Model"
+variable].groupby(pd.Grouper(freq="Y")).median()
        first_yearly_proj = temp_box["Model"
+variable].groupby(pd.Grouper(freq="Y")).quantile(0.25)
        third_yearly_proj = temp_box["Model"
+variable].groupby(pd.Grouper(freq="Y")).quantile(0.75)

        iqr = third_monthly - first_monthly

        fence_low  = first_monthly - (1.5*iqr)
        fence_high = third_monthly + (1.5*iqr)

        fence_low = fence_low.to_frame()

        fence_high = fence_high.to_frame()

        #print(fence_high)
        currMonth = 0
        prevMonth = 0
        tempx = 0 #this will count the fence_low and fence_high, incrementing
when month changed

        newDf = temp_box
        #pd.set_option("display.max_rows", None, "display.max_columns", None)
        #print(newDf)

        for index, row  in newDf.iterrows():
            #go over each row
            #check which month and year and based on that grab the proper
fences
            #once you have them
            currDate = newDf.at[index,'Date']
```

```
            #now grab the year and month of the date
            #currDate = dt.datetime.strptime(currDate,'%Y-%m-%d')

            #now get current month
            currMonth = currDate.month


            #if curr Month and previous month are not same then we increase
tempx
            #and make previous month into current
            if prevMonth != currMonth:
                if (prevMonth != 0):
                    tempx = tempx + 1

                prevMonth = currMonth

            currLow = fence_low.iat[tempx,0]
            currHigh = fence_high.iat[tempx,0]
            #print ("tempx : " + str(tempx) + " currHigh : " + str(currHigh)
)
            #now we remove rows from newDf based on low and high
            if (newDf.at[index,"Observed" +variable] > currHigh) or
(newDf.at[index,"Observed" +variable] < currLow ):
                #drop if the value is outside the range
                #print( "current newDf var: " + str(newDf.at[index,"Observed"
+variable]) +  " current high : " + str(currHigh) + " current Date:" +
str(currDate))
                newDf.drop(index, inplace=True)

        #df_out = temp_box.query('(@first_monthly - 1.5 * @iqr) <= ' +
"Observed" +variable + ' <= (@third_monthly + 1.5 * @iqr)')


        #df_out = (temp_box["Observed" +variable] >= fence_low) &
(temp_box["Observed" +variable] <= fence_high)
        #return df.loc[filter]

        #df_out = temp_box.loc[(temp_box["Observed"
+variable].groupby(pd.Grouper(freq="M")).min() > fence_low) &
(temp_box["Observed" +variable].groupby(pd.Grouper(freq="M")).min() <
fence_high)]

         #df_out = temp_box[(np.abs(stats.zscore(temp_box[1])) < 3)]
        max_monthly = newDf["Observed"
+variable].groupby(pd.Grouper(freq="M")).max()
        min_monthly = newDf["Observed"
+variable].groupby(pd.Grouper(freq="M")).min()

        temp_monthly_box.insert(0,'Date',date_monthly_obs)
        temp_monthly_box.insert(1, 'Minimum', min_monthly)
        temp_monthly_box.insert(2, 'First Quartile', first_monthly)
        temp_monthly_box.insert(3, 'Median', median_monthly)
        temp_monthly_box.insert(4, 'Third Quartile', third_monthly)
        temp_monthly_box.insert(5, 'Maximum', max_monthly)
```

```
        temp_monthly_box['Date'] = temp_monthly_box['Date'].dt.strftime('%b,
%Y')
        #projected
        temp_monthly_proj = pd.DataFrame()

        date_monthly_proj = tempdataMonth['Date']
        #max_monthly_proj = temp_box["Model"
+variable].groupby(pd.Grouper(freq="M")).max()
        #min_monthly_proj = temp_box["Model"
+variable].groupby(pd.Grouper(freq="M")).min()
#         median_monthly_proj = temp_box["Model"
+variable].groupby(pd.Grouper(freq="M")).median()
#         first_monthly_proj = temp_box["Model"
+variable].groupby(pd.Grouper(freq="M")).quantile(0.25)
#         third_monthly_proj = temp_box["Model"
+variable].groupby(pd.Grouper(freq="M")).quantile(0.75)

        iqr2 = third_monthly_proj - first_monthly_proj
        fence_low2  = first_monthly_proj - (1.5*iqr2)
        fence_high2 = third_monthly_proj + (1.5*iqr2)
        fence_low2 = fence_low2.to_frame()

        fence_high2 = fence_high2.to_frame()

        #print(fence_high)
        currMonth = 0
        prevMonth = 0
        tempx = 0 #this will count the fence_low and fence_high, incrementing
when month changed
        currLow = 0
        currHigh = 0
        newDf2 = temp_box
        #pd.set_option("display.max_rows", None, "display.max_columns", None)
        #print(newDf)

        for index, row  in newDf2.iterrows():
            #go over each row
            #check which month and year and based on that grab the proper
fences
            #once you have them
            currDate = newDf2.at[index,'Date']

            #now grab the year and month of the date
            #currDate = dt.datetime.strptime(currDate,'%Y-%m-%d')

            #now get current month
            currMonth = currDate.month


            #if curr Month and previous month are not same then we increase
tempx
            #and make previous month into current
            if prevMonth != currMonth:
                if (prevMonth != 0):
                    tempx = tempx + 1

                prevMonth = currMonth
```

```
            currLow = fence_low2.iat[tempx,0]
            currHigh = fence_high2.iat[tempx,0]
            #print ("tempx : " + str(tempx) + " currHigh : " + str(currHigh)
)
            #now we remove rows from newDf based on low and high
            if (newDf2.at[index,"Model" +variable] > currHigh) or
(newDf2.at[index,"Model" +variable] < currLow ):
                #drop if the value is outside the range
                #print( "current newDf var: " + str(newDf.at[index,"Observed"
+variable]) +  " current high : " + str(currHigh) + " current Date:" +
str(currDate))
                newDf2.drop(index, inplace=True)
        pd.set_option("display.max_rows", None, "display.max_columns", None)
        #print(newDf2)
        max_monthly_proj = newDf2["Model"
+variable].groupby(pd.Grouper(freq="M")).max()
        min_monthly_proj = newDf2["Model"
+variable].groupby(pd.Grouper(freq="M")).min()
        temp_monthly_proj.insert(0,'Date',date_monthly_proj)
        temp_monthly_proj.insert(1, 'Minimum', min_monthly_proj)
        temp_monthly_proj.insert(2, 'First Quartile', first_monthly_proj)
        temp_monthly_proj.insert(3, 'Median', median_monthly_proj)
        temp_monthly_proj.insert(4, 'Third Quartile', third_monthly_proj)
        temp_monthly_proj.insert(5, 'Maximum', max_monthly_proj)

        temp_monthly_proj['Date'] =
temp_monthly_proj['Date'].dt.strftime('%b, %Y')


        tempdataYear = data_all
        tempdataYear['Date'] = pd.to_datetime(tempdataYear['Date'])
        tempdataYear = (tempdataYear.set_index('Date').resample('Y').var())
        tempdataYear['Date'] = tempdataYear.index

        temp_yearly_box = pd.DataFrame()

        #temp_yearly_box['Date'] = temp_yearly_box.index

        date_yearly = tempdataYear['Date']
        #max_yearly = temp_box["Observed"
+variable].groupby(pd.Grouper(freq="Y")).max()
        #min_yearly = temp_box["Observed"
+variable].groupby(pd.Grouper(freq="Y")).min()

        iqr3 = third_yearly - first_yearly
        fence_low3  = first_yearly - (1.5*iqr3)
        fence_high3 = third_yearly + (1.5*iqr3)

        fence_low3 = fence_low3.to_frame()

        fence_high3 = fence_high3.to_frame()

        #print(fence_high)
        currYear = 0
        prevYear = 0
        tempx = 0 #this will count the fence_low and fence_high, incrementing
when month changed
```

```
        newDf3 = temp_box
        #pd.set_option("display.max_rows", None, "display.max_columns", None)
        #print(newDf)

        for index, row  in newDf3.iterrows():
            #go over each row
            #check which month and year and based on that grab the proper
fences
            #once you have them
            currDate = newDf3.at[index,'Date']

            #now grab the year and month of the date
            #currDate = dt.datetime.strptime(currDate,'%Y-%m-%d')

            #now get current year
            currYear = currDate.year


            #if curr Month and previous month are not same then we increase
tempx
            #and make previous month into current
            if prevYear != currYear:
                if (prevYear != 0):
                    tempx = tempx + 1

                prevYear = currYear



            currLow = fence_low3.iat[tempx,0]
            currHigh = fence_high3.iat[tempx,0]
            #print ("tempx : " + str(tempx) + " currHigh : " + str(currHigh)
)
            #now we remove rows from newDf based on low and high
            if (newDf3.at[index,"Observed" +variable] > currHigh) or
(newDf3.at[index,"Observed" +variable] < currLow ):
                #drop if the value is outside the range
                #print( "current newDf var: " + str(newDf.at[index,"Observed"
+variable]) +  " current high : " + str(currHigh) + " current Date:" +
str(currDate))
                newDf3.drop(index, inplace=True)



        max_yearly = newDf3["Observed"
+variable].groupby(pd.Grouper(freq="Y")).max()
        min_yearly = newDf3["Observed"
+variable].groupby(pd.Grouper(freq="Y")).min()


        temp_yearly_box.insert(0,'Date',date_yearly)
        temp_yearly_box.insert(1, 'Minimum', min_yearly)
        temp_yearly_box.insert(2, 'First Quartile', first_yearly)
        temp_yearly_box.insert(3, 'Median', median_yearly)
        temp_yearly_box.insert(4, 'Third Quartile', third_yearly)
        temp_yearly_box.insert(5, 'Maximum', max_yearly)
```

```
        temp_yearly_box['Date'] = temp_yearly_box['Date'].dt.strftime('%Y')

        temp_yearly_proj = pd.DataFrame()

        date_yearly_proj = tempdataYear['Date']


        iqr4 = third_yearly_proj - first_yearly_proj
        fence_low4  = first_yearly_proj - (1.5*iqr4)
        fence_high4 = third_yearly_proj + (1.5*iqr4)

        fence_low4 = fence_low4.to_frame()

        fence_high4 = fence_high4.to_frame()

        #print(fence_high)
        currYear = 0
        prevYear = 0
        tempx = 0 #this will count the fence_low and fence_high, incrementing
when month changed

        newDf4 = temp_box
        #pd.set_option("display.max_rows", None, "display.max_columns", None)
        #print(newDf)

        for index, row  in newDf4.iterrows():
            #go over each row
            #check which month and year and based on that grab the proper
fences
            #once you have them
            currDate = newDf4.at[index,'Date']

            #now grab the year and month of the date
            #currDate = dt.datetime.strptime(currDate,'%Y-%m-%d')

            #now get current month
            currMonth = currDate.year


            #if curr Month and previous month are not same then we increase
tempx
            #and make previous month into current
            if prevYear != currYear:
                if (prevYear != 0):
                    tempx = tempx + 1

                prevYear = currYear



            currLow = fence_low4.iat[tempx,0]
            currHigh = fence_high4.iat[tempx,0]
            #print ("tempx : " + str(tempx) + " currHigh : " + str(currHigh)
)
            #now we remove rows from newDf based on low and high
```

```
            if (newDf4.at[index,"Model" +variable] > currHigh) or
(newDf4.at[index,"Model" +variable] < currLow ):
                #drop if the value is outside the range
                #print( "current newDf var: " + str(newDf.at[index,"Observed"
+variable]) +  " current high : " + str(currHigh) + " current Date:" +
str(currDate))
                newDf4.drop(index, inplace=True)


        pd.set_option("display.max_rows", None, "display.max_columns", None)
        print(newDf4["Model" +variable].groupby(pd.Grouper(freq="Y")))
        print("AHHHHHHHHH NEW UHSAUDUH UISHAUI DH")

        max_yearly_proj = newDf4["Model"
+variable].groupby(pd.Grouper(freq="Y")).max()
        min_yearly_proj = newDf4["Model"
+variable].groupby(pd.Grouper(freq="Y")).min()


        pd.set_option("display.max_rows", None, "display.max_columns", None)
        print(min_yearly_proj)
        temp_yearly_proj.insert(0,'Date',date_yearly_proj)
        temp_yearly_proj.insert(1, 'Minimum', min_yearly_proj)
        temp_yearly_proj.insert(2, 'First Quartile', first_yearly_proj)
        temp_yearly_proj.insert(3, 'Median', median_yearly_proj)
        temp_yearly_proj.insert(4, 'Third Quartile', third_yearly_proj)
        temp_yearly_proj.insert(5, 'Maximum', max_yearly_proj)

        temp_yearly_proj['Date'] = temp_yearly_proj['Date'].dt.strftime('%Y')


        #use temp_box_year for year wise


        #seasonal plot here
        #have to creat a DF where each year is a column and one column is the
month
        #under the month column is simply months, and under the year column
is the values


        temp_yearly_proj = temp_yearly_proj.round(2)
        temp_yearly_box = temp_yearly_box.round(2)

        temp_monthly_proj = temp_monthly_proj.round(2)
        temp_monthly_box = temp_monthly_box.round(2)


        seasonalfig = px.line(seasonalDF, x="Month", y=seasonalDF.columns)


        seasonalfig.update_yaxes(
            title = columnName + " " +   measurement
        )
        seasonalfig.update_layout(title = "Observed "+ columnName+ "
Seasonality Plots",legend_title=dict(text = "Year"))
```

```
        seasonalfig2 = px.line(seasonalDF2, x="Month", y=seasonalDF.columns)

        boxfig3.update_layout(title = 'Observed ' + columnName + ': Year-wise
Box Plots')

        boxfig4.update_layout(title = 'Predicted ' + columnName + ': Year-
wise Box Plots')

        boxfig1.update_layout(title = 'Observed ' + columnName + ': Month-
wise Box Plots')

        boxfig2.update_layout(title = 'Predicted ' + columnName + ': Month-
wise Box Plots')

        seasonalfig2.update_yaxes(
            title = columnName + " " +   measurement
        )
        seasonalfig2.update_layout(title = "Predicted "+ columnName+ "
Seasonality Plots",legend_title=dict(text = "Year"))


        tempfig = make_subplots(specs=[[{"secondary_y": True}]])



        tempfig.add_trace(go.Scatter(x=data_all['Date'], y=data_all['Bias'],
fill='tozeroy', name = "Bias"),secondary_y=True) # fill to trace0 y
        tempfig.add_trace(go.Scatter(x=data_all['Date'],
y=data_all["Observed" +variable], name = "Observed"),secondary_y=False) #
fill down to xaxis
        tempfig.update_layout(
            title_x=0.5,
            xaxis_title="Date",
            yaxis_title=columnName + ' ' + measurement,
            showlegend=True,
            yaxis_range=[-75,150],
#             margin=dict(l=5,r=5,b=0,t=5,(specs=[[{"secondary_y":
True}]])pad=5)
            )
        tempfig.update_yaxes(title = "Bias", range = [-75,150],secondary_y =
True)
        tempfig.update_layout(title = 'Bias Graph')

#        pio.kaleido.scope.mathjax = None
        #print out all figs
#         fig.write_image("Images/MainGraph.png",engine='orca')
#         tempfig.write_image("Images/Biasfig.png",engine='orca')
#         boxfig1.write_image("Images/Boxfig1.png",engine='orca')
#         boxfig2.write_image("Images/Boxfig2.png",engine='orca')
#         boxfig3.write_image("Images/Boxfig3.png",engine='orca')
#         boxfig4.write_image("Images/Boxfig4.png",engine='orca')
#         seasonalfig.write_image("Images/Seasonalfig1.png",engine='orca')
#         seasonalfig2.write_image("Images/Seasonalfig2.png",engine='orca')

        print(temp_yearly_proj)
        print(boxfig4)
        if variable == 'pr':
```

```
            percipValue ={'float' : 'left','margin' : '10px','width':
'800px', 'display': 'inline-block'}
            percipValueRight ={'float' : 'right','margin' : '10px','width':
'800px', 'display': 'inline-block'}
        else:
            percipValue ={'float' : 'left','margin' : '10px','width':
'800px', 'display': 'none'}
            percipValueRight ={'float' : 'right','margin' : '10px','width':
'800px', 'display': 'none'}

        if variable == 'rlds':
            biasShow ={'float' : 'right','margin' : '10px','width': '800px',
'display': 'none'}
        else:
            biasShow ={'float' : 'right','margin' : '10px','width': '800px',
'display': 'inline-block'}

   #     print(str(percipValue))
#         biasfig = go.Figure(data=go.Scatter(
#                     x=data_all['Date'],
#                     y=data_all["Observed" +variable],
#                     xhoverformat="%d %b, %Y",
#                     yhoverformat = ".2f",
#                     error_y=dict(
#                         type='data',
#                         array=data_all['Bias']
#                     )))

#         biasfig.update_yaxes(title_text="Observed Value of " + columnName,
secondary_y=False)
#         biasfig.update_yaxes(title_text="Error Bar Value",
secondary_y=True)

        #create the output that will go on the page

        graph = html.Div([
            html.Div(id = 'graphDIV', children =[
                html.Button("Download Data Model",
id="btn_csv_auto",style={'background-color': '#4CAF50','border':
'none','color': 'white','padding': '15px 32px','text-align': 'center','text-
decoration': 'none','display': 'inline-block','font-size': '16px','margin':
'4px 2px'}),
                dcc.Download(id="download-dataframe-csv-auto"),
                (dcc.Graph(
                   id='Graph1',
                   figure=fig,
                    style={
                        'height' : '300px',
                        'margin' : '50px'
                    }
                ))
            ],style = {'display': 'inline-block'}),


            html.Div(id = 'frequencyObs', children=[
            html.Div([
```

```
                html.Button("Download Analysis",
id="btn_analysis_auto",style={'background-color': '#4CAF50','border':
'none','color': 'white','padding': '15px 32px','text-align': 'center','text-
decoration': 'none','display': 'inline-block','font-size': '16px','margin':
'4px 2px'}),
                dcc.Download(id="download-analysis-auto"),
                    ],style={'float' : 'left','margin' : '10px','margin-top':
'350px','width': '100%', 'display': 'inline-block'}),

            html.Div([
            html.H3('Number of days where observed ' + columnName + ' is >/<
than  ECCC 30 year normals', style={'width': '800px','margin' : '10px','font-
size': '12px', 'float': 'left', 'font-family': 'Verdana'}),
            dash_table.DataTable(
                id='table',
                columns=[{"name": i, "id": i} for i in stats1.columns],
                data=stats1.to_dict('records'),
            )
            ],style=percipValue),

             html.Div([
                 html.H3('Number of days where Predicted ' + columnName + '
is >/< than  ECCC 30 year normal', style={'width': '800px','margin' :
'10px','font-size': '12px', 'float': 'left', 'font-family': 'Verdana'}),
                 dash_table.DataTable(
                id='table',
                columns=[{"name": i, "id": i} for i in stats2.columns],
                data=stats2.to_dict('records'),
            )
             ],style=percipValueRight),

            html.Div([
                html.H3('Annual ' + columnName + '', style={'width':
'800px','font-size': '12px','margin' : '10px', 'font-family': 'Verdana'}),
                dash_table.DataTable(
                id='table',
                columns=[{"name": i, "id": i} for i in anualPrecip.columns],
                data=anualPrecip.to_dict('records'),
                style_cell={
                    'minWidth': '10px', 'width': '10px', 'maxWidth': '10px'
                }
            ),
            ],style=percipValue),

            html.Div([],style={'float' : 'left','margin' : '10px','width':
'100%', 'display': 'inline-block'}),
            html.Div([
                html.H3('Observed ' + columnName + ': Monthly Means',
style={'width': '800px','margin' : '10px', 'font-size': '12px', 'float':
'left', 'font-family': 'Verdana'}),
                dash_table.DataTable(
                id='table',
                columns=[{"name": i, "id": i} for i in stats3.columns],
                data=stats3.to_dict('records')
            ),
            ],style={'float' : 'left','margin' : '10px','width': '800px',
'display': 'inline-block'}),
```

```
            html.Div([
                html.H3('Predicted ' + columnName + ': Monthly Means',
style={'width': '800px','margin' : '10px','font-size': '12px', 'float':
'left', 'font-family': 'Verdana'}),
                dash_table.DataTable(
                id='table',
                columns=[{"name": i, "id": i} for i in stats4.columns],
                data=stats4.to_dict('records')

            ),
            ],style={'float' : 'right','margin' : '10px','width': '800px',
'display': 'inline-block'}),
             html.Div([
                html.H3('Observed ' + columnName + ': Monthly Medians',
style={'width': '800px','margin' : '10px', 'font-size': '12px', 'float':
'left', 'font-family': 'Verdana'}),
                dash_table.DataTable(
                id='table',
                columns=[{"name": i, "id": i} for i in
medianMonthlyObs.columns],
                data=medianMonthlyObs.to_dict('records')
            ),
            ],style={'float' : 'left','margin' : '10px','width': '800px',
'display': 'inline-block'}),
            html.Div([
                html.H3('Predicted ' + columnName + ': Monthly Medians',
style={'width': '800px','margin' : '10px', 'font-size': '12px', 'float':
'left', 'font-family': 'Verdana'}),
                dash_table.DataTable(
                id='table',
                columns=[{"name": i, "id": i} for i in
medianMonthlyProj.columns],
                data=medianMonthlyProj.to_dict('records')
            ),
            ],style={'float' : 'right','margin' : '10px','width': '800px',
'display': 'inline-block'}),
            html.Div([

                html.H3('Observed ' + columnName + ': Monthly Standard
Deviation', style={'width': '800px','margin' : '10px','font-size': '12px',
'float': 'left', 'font-family': 'Verdana'}),
                dash_table.DataTable(
                id='table',
                columns=[{"name": i, "id": i} for i in stats5.columns],
                data=stats5.to_dict('records')
            ),
            ],style={'float' : 'left','margin' : '10px','width': '800px',
'display': 'inline-block'}),

            html.Div([
                html.H3('Predicted ' + columnName + ': Monthly Standard
Deviation', style={'width': '800px','margin' : '10px','font-size': '12px',
'float': 'left', 'font-family': 'Verdana'}),
                dash_table.DataTable(
                id='table',
                columns=[{"name": i, "id": i} for i in stats6.columns],
                data=stats6.to_dict('records')
```

```
            )

            ],style={'float' : 'right','margin' : '10px','width': '800px',
'display': 'inline-block'}),



            html.Div([],style={'float' : 'left','margin' : '10px','width':
'100%', 'display': 'inline-block'}),

            html.Div([

                #html.H3('Bias Graph
', style={'width': 'auto','margin' : '10px','font-size': '12px', 'float':
'left', 'font-family': 'Verdana'}),
                (dcc.Graph(
                   id='Graph1',
                   figure=tempfig
                   ))

            ],style={'float' : 'left','margin' : '10px','width': '800px',
'display': 'inline-block'}),

              html.Div([
                html.H3('Bias between Predicted and Observed' + columnName +
' ', style={ 'width': '800px','margin' : '10px','font-size': '12px', 'float':
'left', 'font-family': 'Verdana'}),
                dash_table.DataTable(
                id='table',
                columns=[{"name": i, "id": i} for i in biasDF.columns],
                data=biasDF.to_dict('records'))


            ],style=biasShow),

               html.Div([],style={'float' : 'left','margin' : '10px','width':
'100%', 'display': 'inline-block'}),

                html.Div([
          #       html.H3('Observed ' + columnName + ': Month-wise Box Plots',
style={'width': '800px','margin' : '10px','font-size': '12px', 'float':
'left', 'font-family': 'Verdana'}),
                 (dcc.Graph(
                id='Graph1',
                figure=boxfig1
                 )),

            ],style={'float' : 'left','margin' : '10px','width': '800px',
'display': 'inline-block'}),

            html.Div([

       #    html.H3('Projected ' + columnName + ': Month-wise Box Plots',
style={ 'width': '800px','margin' : '10px','font-size': '12px', 'float':
'left', 'font-family': 'Verdana'}),
            (dcc.Graph(
               id='Graph1',
```

```
                figure=boxfig2
            )),

            ],style={'float' : 'right','margin' : '10px','width': '800px',
'display': 'inline-block'}),


            html.Div([
                dash_table.DataTable(
                id='table',
                columns=[{"name": i, "id": i} for i in
temp_monthly_box.columns],
                data=temp_monthly_box.to_dict('records')
            ),

            ],style={'float' : 'left','margin' : '10px','width': '800px',
'display': 'inline-block'}),

            html.Div([
                 dash_table.DataTable(
                id='table',
                columns=[{"name": i, "id": i} for i in
temp_monthly_proj.columns],
                data=temp_monthly_proj.to_dict('records')
            ),
            ],style={'float' : 'right','margin' : '10px','width': '800px',
'display': 'inline-block'}),


            html.Div([
#               html.H3('Observed ' + columnName + ': Year-wise Box Plots',
style={'width': '800px','margin' : '10px','font-size': '12px', 'float':
'left', 'font-family': 'Verdana'}),
                (dcc.Graph(
                   id='Graph1',
                   figure=boxfig3
               )),

            ],style={'float' : 'left','width': '800px', 'display': 'inline-
block'}),

              html.Div([
            #html.H3('Projected ' + columnName + ': Year-wise Box Plots',
style={'font-size': '12px', 'float': 'right', 'font-family': 'Verdana'}),


            (dcc.Graph(
               id='Graph1',
               figure=boxfig4
           )),

            ],style={'float' : 'right','width': '800px', 'display': 'inline-
block'}),


            html.Div([
               dash_table.DataTable(
```

```
                id='table',
                columns=[{"name": i, "id": i} for i in
temp_yearly_box.columns],
                data=temp_yearly_box.to_dict('records')
            ),

            ],style={'float' : 'left','margin' : '10px','width': '800px',
'display': 'inline-block'}),

              html.Div([

            dash_table.DataTable(
                id='table',
                columns=[{"name": i, "id": i} for i in
temp_yearly_proj.columns],
                data=temp_yearly_proj.to_dict('records')
            ),

            ],style={'float' : 'right','margin' : '10px','width': '800px',
'display': 'inline-block'}),


            html.Div([
#           html.H3('Observed and Projected Precipitation Seasonality Plots',
style={'font-size': '12px', 'font-family': 'Verdana'}),
                 (dcc.Graph(
               id='Graph1',
               figure=seasonalfig
           )),

            ],style={'float' : 'left','width': '800px', 'display': 'inline-
block'}),

              html.Div([
                (dcc.Graph(
                   id='Graph1',
                   figure=seasonalfig2,

               ))

            ],style={'float' : 'right','width': '800px', 'display': 'inline-
block'}),




            ],style = {'display':'inline-block','width' : '2000px'})



        ])

        return 'Analyze-auto',graph,data_all.to_json(date_format='iso',
orient='split'),stats1.to_json(date_format='iso',
orient='split'),stats2.to_json(date_format='iso',
orient='split'),anualPrecip.to_json(date_format='iso',
```

```
orient='split'),stats3.to_json(date_format='iso',
orient='split'),stats4.to_json(date_format='iso',
orient='split'),stats5.to_json(date_format='iso',
orient='split'),stats6.to_json(date_format='iso',
orient='split'),stats7.to_json(date_format='iso',
orient='split'),stats8.to_json(date_format='iso',
orient='split'),biasDF.to_json(date_format='iso',
orient='split'),temp_monthly_box.to_json(date_format='iso',
orient='split'),temp_monthly_proj.to_json(date_format='iso',
orient='split'),temp_yearly_box.to_json(date_format='iso',
orient='split'),temp_yearly_proj.to_json(date_format='iso',
orient='split'),fig

#This is here for calculating the Model vs Observed graph and outputting it
to the DIV
@app.callback(
    Output("tabs", "value"),
    Output("analyze", "children"),
    Output('main_data','data'),
    Output('observed_normals','data'),
    Output('projected_normals','data'),
    Output('annual_data','data'),
    Output('observed_mean','data'),
    Output('projected_mean','data'),
    Output('observed_deviation','data'),
    Output('projected_deviation','data'),
    Output('observed_variance','data'),
    Output('projected_variance','data'),
    Output('bias_data','data'),
    Output('observed_monthBox','data'),
    Output('projected_monthBox','data'),
    Output('observed_yearBox','data'),
    Output('projected_yearBox','data'),
    Output('mainFig','data'),
    [Input("submit-val", "n_clicks")],
    state=[State(component_id='region', component_property='value'),
State('variable', 'value'),State('date-picker-range',
'start_date'),State('date-picker-range', 'end_date')])
def tab_resources(click,region,variable,startDatePicked,endDatePicked):
    #print("testing1234")
    if click:

        #this is where we calculate stuff
        #Insert Inputs HERE

        #lat and long are currently not relevant

        #date range (change to read user input)
        #print(startDatePicked)
        startDate = dt.datetime.strptime(startDatePicked,'%Y-%m-%d')

        endDate = dt.datetime.strptime(endDatePicked,'%Y-%m-%d') -
dt.timedelta(days=1)

        print (startDate)
        print(endDate)
        #variable (change to read user input)
```

```
        variable = variable


# use this to determine full variable name and measurement unit
        if variable == 'tas':
                    #then it is TEMP in the data, find the column
            column = 'avg_hourly_temperature'
            columnName = 'Surface Temperature'
            measurement = '(C)'
        elif variable == 'pr' :
            column = 'precipitation'
            columnName = 'Precipitation'
            measurement = '(mm)'

        elif variable == 'ps' :
            column = 'avg_hourly_pressure_station'
            columnName = 'Surface Pressure'
            measurement = '(kPa)'

        elif variable == 'sfcWind' :
            column = 'avg_hourly_wind_speed'
            columnName = 'Surface Wind Speed'
            measurement = '(km/h)'

        elif variable == 'sfcWindmax' :
            column = 'max_wind_speed'
            columnName = 'Maximum Surface Wind Speed'
            measurement = '(km/h)'

        elif variable == 'tasmax' :
            column = 'max_temperature'
            columnName = 'Maximum Surface Temperature'
            measurement = '(C)'

        elif variable == 'tasmin' :
            column = 'min_temperature'
            columnName = 'Minimum Surface Temperature '
            measurement = '(C)'

        elif variable == 'rlds':
            column = 'ALLSKY_SFC_LW_DWN'
            columnName = 'Surface Downwelling Longwave Radiation'
            measurement = '(W m-2)'



        #now we call the mapped csv for this variable
        data =
pd.read_csv(user_key['projectedvsobservedbias_mapped']+variable+'_comparison_
bias_output.csv')

#         mask = (data['Date'] >= str(startDate.date())) & (data['Date'] <=
str(endDate.date()))
#         #only use the data in between the user given range
#         data_all = data.loc[mask]
        data_all = data[data['Date'].between(str(startDate.date()),
str(endDate.date()),inclusive='both')]
```

```
        #create the plot figures
        fig = make_subplots(rows=2, cols=1, shared_xaxes=False)

        fig.add_trace(go.Scatter(x=data_all['Date'],
y=data_all['Model'+variable],mode = 'lines+markers', name =
"Predicted",xhoverformat="%d %b, %Y" ,yhoverformat = ".2f"))
        fig.add_trace(go.Scatter(x=data_all['Date'],
y=data_all['Observed'+variable],mode = 'lines+markers',name =
"Observed",xhoverformat="%d %b, %Y" ,yhoverformat = ".2f"))

        #Update the plot layot with titles
        fig.update_layout(
            title_text="Comparison of Predicted and Observed Climate",
            title_x=0.5,
            xaxis_title="Date",
            yaxis_title=columnName + ' ' + measurement,
            showlegend=True,
            autosize=False,
            width=1300,
            height=700,

#             margin=dict(l=5,r=5,b=0,t=5,pad=5)
            margin=go.layout.Margin(
                l=0,
                r=0,
                b=25,
                t=25,
                pad = 0
                )
            )


        #Create the Analysis to be placed under the graph
        #observed frequency
        stats1 = pd.DataFrame(columns=['Month'])

        #projected frequency
        stats2 = pd.DataFrame(columns=['Month'])

        #observed tendency
        stats3 = pd.DataFrame(columns=['Month'])

        #projected tendency
        stats4 = pd.DataFrame(columns=['Month'])


        #seasonal observed
        seasonalDF = pd.DataFrame(columns=['Month'])

        #seasonal project
        seasonalDF2 = pd.DataFrame(columns=['Month'])

        #standardDeviation Observed
        stats5 = pd.DataFrame(columns=['Month'])

        #standard deviation Projected
```

```
        stats6 = pd.DataFrame(columns=['Month'])


        #variance observed
        stats7 = pd.DataFrame(columns=['Month'])

        #variance projected
        stats8 = pd.DataFrame(columns=['Month'])


        #Bias
        biasDF = pd.DataFrame(columns=['Month'])

       #monthly median stats observed
        medianMonthlyObs = pd.DataFrame(columns=['Month'])

        #monthly median stats predicted
        medianMonthlyProj = pd.DataFrame(columns=['Month'])


        #get number of years in the date range and create them as columns
        for i in range(startDate.year,(endDate.year+1)):
            stats1[i] = ""
            stats2[i] = ""
            stats3[i] = ""
            stats4[i] = ""
            stats5[i] = ""
            stats6[i] = ""
            stats7[i] = ""
            stats8[i] = ""
            medianMonthlyObs[i] = ""
            medianMonthlyProj[i] = ""
            biasDF[i] = ""
            seasonalDF[i] = ""
            seasonalDF2[i] = ""

        #fill in the months Column
        months =
['January','February','March','April','May','June','July','August','September
','October','November','December']
        for i in months:
            stats1 = stats1.append({'Month': i}, ignore_index=True)
            stats2 = stats2.append({'Month': i}, ignore_index=True)
            stats3 = stats3.append({'Month': i}, ignore_index=True)
            stats4 = stats4.append({'Month': i}, ignore_index=True)
            stats5 = stats5.append({'Month': i}, ignore_index=True)
            stats6 = stats6.append({'Month': i}, ignore_index=True)
            stats7 = stats7.append({'Month': i}, ignore_index=True)
            stats8 = stats8.append({'Month': i}, ignore_index=True)
            medianMonthlyObs = medianMonthlyObs.append({'Month': i},
ignore_index=True)
            medianMonthlyProj = medianMonthlyProj.append({'Month': i},
ignore_index=True)
            biasDF = biasDF.append({'Month': i}, ignore_index=True)
            seasonalDF = seasonalDF.append({'Month': i}, ignore_index=True)
            seasonalDF2 = seasonalDF2.append({'Month': i}, ignore_index=True)
```

```
        #fill in the rows with 0/0

        stats1.fillna("0/0", inplace=True)
        stats2.fillna("0/0", inplace=True)
        stats3.fillna("0", inplace=True)
        stats4.fillna("0", inplace=True)
        stats5.fillna("0", inplace=True)
        stats6.fillna("0", inplace=True)
        stats7.fillna("0", inplace=True)
        stats8.fillna("0", inplace=True)
        medianMonthlyObs.fillna("0", inplace=True)
        medianMonthlyProj.fillna("0", inplace=True)
        biasDF.fillna("0/0", inplace=True)
        seasonalDF.fillna("0", inplace=True)
        seasonalDF2.fillna("0", inplace=True)


        print(data_all)
        #now loop through each row in data_all, and
        #THIS IS FOR FREQUENCY for Stats 1 and 2
        rowit = data_all.iterrows()

        for i, row in rowit:

            #now for each row we check the month
            #then we declare our normal based on the month
            #then we compare with the normal to determine which count it goes
in
            currDate = dt.datetime.strptime(data_all.at[i,'Date'], '%Y-%m-
%d')

            currMonth = currDate.month - 1

            currYear = currDate.year

            newValObs = data_all.at[i,'Observed'+variable]
            newValProj = data_all.at[i,'Model'+variable]

            newBias = data_all.at[i,'Bias']


            if(currMonth == 0):
                normal = 48.67
            elif(currMonth == 1):
                normal = 47.7
            elif(currMonth == 2):
                normal = 49.8
            elif(currMonth == 3):
                normal = 68.5
            elif(currMonth == 4):
                normal = 74.3
            elif(currMonth == 5):
                normal = 71.5
            elif(currMonth == 6):
                normal = 75.7
            elif(currMonth == 7):
                normal = 78.1
```

```
elif(currMonth == 8):
    normal = 74.5
elif(currMonth == 9):
    normal = 61.1
elif(currMonth == 10):
    normal = 75.1
elif(currMonth == 11):
    normal = 57.9


currValObs = stats1.at[(currMonth),currYear]

overObs = currValObs.split("/",1)[0]
underObs = currValObs.split("/",1)[1]

currValProj = stats2.at[(currMonth),currYear]

overProj = currValProj.split("/",1)[0]
underProj = currValProj.split("/",1)[1]


currentBias = biasDF.at[(currMonth),currYear]
posBias = currentBias.split("/",1)[0]
negBias = currentBias.split("/",1)[1]

#the bias threshold is based on variable
biasThreshold = 0

if variable == 'tas':
    biasThreshold = 1
elif variable == 'pr' :
    biasThreshold = 1

elif variable == 'ps' :
    biasThreshold = 0.5

elif variable == 'sfcWind' :
    biasThreshold = 5

elif variable == 'sfcWindmax' :
    biasThreshold = 5

elif variable == 'tasmax' :
    biasThreshold = 1

elif variable == 'tasmin' :
    biasThreshold = 1




if newBias > biasThreshold:
    posBias = int(posBias) + 1
elif newBias < (biasThreshold * -1):
    negBias = int(negBias) + 1
else:
```

```
                posBias = int(posBias)
                negBias = int(negBias)


            if newValObs > normal:
                overObs = int(overObs) + 1
            else:
                underObs = int(underObs) + 1

            if newValProj > normal:
                overProj = int(overProj) + 1
            else:
                underProj = int(underProj) + 1


            #overwrite the value
            stats1.at[(currMonth),currYear] = str(overObs) + "/" +
str(underObs)
            stats2.at[(currMonth ),currYear] = str(overProj) + "/" +
str(underProj)


            #seasonal data
            seasonalDF.at[(currMonth ),currYear] = newValObs
            seasonalDF2.at[(currMonth ),currYear] = newValProj

            #bias
            biasDF.at[(currMonth ),currYear] = str(posBias) + "/" +
str(negBias)


        #CENTRAL TENDENCY

        temp_monthly = data_all
        temp_monthly['Date'] = pd.to_datetime(temp_monthly['Date'])
        temp_monthly = (temp_monthly.set_index('Date').resample('M').mean())

        temp_monthly_median = data_all
        temp_monthly_median['Date'] =
pd.to_datetime(temp_monthly_median['Date'])
        temp_monthly_median =
(temp_monthly_median.set_index('Date').resample('M').median())


        temp_yearly = data_all
        temp_yearly['Date'] = pd.to_datetime(temp_yearly['Date'])
        temp_yearly = (temp_yearly.set_index('Date').resample('Y').mean())

        temp_yearly_median = data_all
        temp_yearly_median['Date'] =
pd.to_datetime(temp_yearly_median['Date'])
        temp_yearly_median =
(temp_yearly_median.set_index('Date').resample('Y').median())

        #now that we have monthly and yearly resampled, place them in matrix
        stats3 = stats3.append({'Month': 'Annual'}, ignore_index=True)
        stats4 = stats4.append({'Month': 'Annual'}, ignore_index=True)
```

```
        #monthly median in dataframe
        medianMonthlyObs = medianMonthlyObs.append({'Month': 'Annual'},
ignore_index=True)
        medianMonthlyProj = medianMonthlyProj.append({'Month': 'Annual'},
ignore_index=True)

        rowit = temp_monthly.iterrows()

        row_median = temp_monthly_median.iterrows()

        temp_monthly['Date'] = temp_monthly.index
        temp_monthly_median['Date'] = temp_monthly_median.index

        temp_monthly = temp_monthly.round({'Observed'+variable: 2,
'Model'+variable: 2})
        temp_monthly_median = temp_monthly_median.round({'Observed'+variable:
2, 'Model'+variable: 2})
        temp_yearly = temp_yearly.round({'Observed'+variable: 2,
'Model'+variable: 2})
        temp_yearly_median = temp_yearly_median.round({'Observed'+variable:
2, 'Model'+variable: 2})


        #add monthly means
        for i, row in rowit:
            currDate = temp_monthly.at[i,'Date']

            currMonth = currDate.month

            currYear = currDate.year


            stats3.at[(currMonth),currYear] =
temp_monthly.at[i,'Observed'+variable]
            stats4.at[(currMonth),currYear] =
temp_monthly.at[i,'Model'+variable]

         #add monthly median
        for i, row in row_median:
            currDate = temp_monthly_median.at[i,'Date']

            currMonth = currDate.month

            currYear = currDate.year


            medianMonthlyObs.at[(currMonth ),currYear] =
temp_monthly_median.at[i,'Observed'+variable]
            medianMonthlyProj.at[(currMonth),currYear] =
temp_monthly_median.at[i,'Model'+variable]


        temp_yearly['Date'] = temp_yearly.index
        rowitYear = temp_yearly.iterrows()
```

```
        #add monthly means
        for i, row in rowitYear:
            currDate = temp_yearly.at[i,'Date']

            currMonth = currDate.month

            currYear = currDate.year


            stats3.at[12,currYear] = temp_yearly.at[i,'Observed'+variable]
            stats4.at[12,currYear] = temp_yearly.at[i,'Model'+variable]

        temp_yearly_median['Date'] = temp_yearly_median.index
        rowitYear_median = temp_yearly_median.iterrows()
         #add yearly medians
        for i, row in rowitYear_median:
            currDate = temp_yearly_median.at[i,'Date']

            currMonth = currDate.month

            currYear = currDate.year


            medianMonthlyObs.at[12,currYear] =
temp_yearly_median.at[i,'Observed'+variable]
            medianMonthlyProj.at[12,currYear] =
temp_yearly_median.at[i,'Model'+variable]

        #STANDARD DEVIATION

        stats5 = stats5.append({'Month': 'Annual'}, ignore_index=True)
        stats6 = stats6.append({'Month': 'Annual'}, ignore_index=True)

        temp_monthly_std = data_all
        temp_monthly_std['Date'] = pd.to_datetime(temp_monthly_std['Date'])
        temp_monthly_std =
(temp_monthly_std.set_index('Date').resample('M').std())



        temp_yearly_std = data_all
        temp_yearly_std['Date'] = pd.to_datetime(temp_yearly_std['Date'])
        temp_yearly_std =
(temp_yearly_std.set_index('Date').resample('Y').std())

        rowit = temp_monthly_std.iterrows()

        temp_monthly_std['Date'] = temp_monthly_std.index

        temp_monthly_std = temp_monthly_std.round({'Observed'+variable: 2,
'Model'+variable: 2})
        temp_yearly_std = temp_yearly_std.round({'Observed'+variable: 2,
'Model'+variable: 2})


        #add monthly means
```

```
        for i, row in rowit:
            currDate = temp_monthly_std.at[i,'Date']

            currMonth = currDate.month

            currYear = currDate.year


            stats5.at[(currMonth),currYear] =
temp_monthly_std.at[i,'Observed'+variable]
            stats6.at[(currMonth),currYear] =
temp_monthly_std.at[i,'Model'+variable]

        temp_yearly_std['Date'] = temp_yearly_std.index
        rowitYear = temp_yearly_std.iterrows()
        #add monthly means
        for i, row in rowitYear:
            currDate = temp_yearly_std.at[i,'Date']

            currMonth = currDate.month

            currYear = currDate.year


            stats5.at[12,currYear] =
temp_yearly_std.at[i,'Observed'+variable]
            stats6.at[12,currYear] = temp_yearly_std.at[i,'Model'+variable]
                #add monthly means
        for i, row in rowit:
            currDate = temp_monthly_var.at[i,'Date']

            currMonth = currDate.month

            currYear = currDate.year


            stats7.at[(currMonth),currYear] =
temp_monthly_var.at[i,'Observed'+variable]


        #Anual Precip
        temp_yearly_precip = data_all
        temp_yearly_precip['Date'] =
pd.to_datetime(temp_yearly_precip['Date'])
        temp_yearly_precip =
(temp_yearly_precip.set_index('Date').resample('Y').sum())

        temp_yearly_precip['Date'] = temp_yearly_precip.index

        #anual precip observed
        anualPrecip = pd.DataFrame()

        anualPrecip['Year'] = temp_yearly_precip['Date'].dt.year
        anualPrecip['Observed'] =  temp_yearly_precip['Observed'+variable]
        anualPrecip['Projected'] =  temp_yearly_precip['Model'+variable]

        anualPrecip = anualPrecip.round({'Observed': 2, 'Projected': 2})
```

```
        #box plots
        #create a df for both positive and negative

        #firstly pull the data
        bias=pd.read_csv(user_key['projectedvsobservedbias_mapped']+
variable+'_comparison_bias_output.csv')

        bias['Date'] = pd.to_datetime(bias['Date'])
        bias.reset_index(inplace=True)



        # do bias data
        bias['year'] = [dt.datetime.year for d in bias.Date]
        bias['month'] = [dt.datetime.month for d in bias.Date]
        years = bias['year'].unique()

        # Draw box plots

        #to do this we must create a new column listing month and year

        #this is the monthly breakdown
        temp_box = data_all


        temp_box['month_year'] = temp_box['Date'].dt.strftime('%Y-%m')

        boxfig1 = px.box(temp_box,x = "month_year", y = "Observed" +variable)

        boxfig2 = px.box(temp_box,x = "month_year", y = "Model" +variable)

        #this is for the year

        temp_box_year = data_all

        temp_box_year['year'] = temp_box_year['Date'].dt.strftime('%Y')

        boxfig3 = px.box(temp_box_year,x = "year", y = "Observed" +variable)

        boxfig4 = px.box(temp_box_year,x = "year", y = "Model" +variable)
        #print(boxfig4.hoverinfo)

        boxfig1.update_xaxes(
            title = "Month, Year"
        )
        boxfig1.update_yaxes(
            title = columnName + " " +   measurement
        )


        boxfig2.update_xaxes(
            title = "Month, Year"
        )
        boxfig2.update_yaxes(
            title = columnName + " " +   measurement
        )
```

```
        boxfig3.update_xaxes(
            title = "Year"
        )
        boxfig3.update_yaxes(
            title = columnName + " " +   measurement
        )


        boxfig4.update_xaxes(
            title = "Year"
        )
        boxfig4.update_yaxes(
            title = columnName + " " +   measurement
        )

        #print(boxfig3.hover_data())

        tempdataMonth = data_all
        tempdataMonth['Date'] = pd.to_datetime(tempdataMonth['Date'])
        tempdataMonth = (tempdataMonth.set_index('Date').resample('M').var())
        tempdataMonth['Date'] = tempdataMonth.index
        #create the data for the box figures


        #use temp_box for monthwise
        temp_monthly_box = pd.DataFrame()

        temp_box = data_all
        temp_box['Date'] = pd.to_datetime(temp_box['Date'])
        temp_box = (temp_box.set_index('Date'))
        temp_box['Date'] = temp_box.index

        date_monthly_obs = tempdataMonth['Date']

        median_monthly = temp_box["Observed"
+variable].groupby(pd.Grouper(freq="M")).median()
        first_monthly = temp_box["Observed"
+variable].groupby(pd.Grouper(freq="M")).quantile(0.25)
        third_monthly = temp_box["Observed"
+variable].groupby(pd.Grouper(freq="M")).quantile(0.75)
        tempOBSBOX = temp_box["Observed"
+variable].groupby(pd.Grouper(freq="M"))

        median_monthly_proj = temp_box["Model"
+variable].groupby(pd.Grouper(freq="M")).median()
        first_monthly_proj = temp_box["Model"
+variable].groupby(pd.Grouper(freq="M")).quantile(0.25)
        third_monthly_proj = temp_box["Model"
+variable].groupby(pd.Grouper(freq="M")).quantile(0.75)


        median_yearly = temp_box["Observed"
+variable].groupby(pd.Grouper(freq="Y")).median()
        first_yearly = temp_box["Observed"
+variable].groupby(pd.Grouper(freq="Y")).quantile(0.25)
```

```
        third_yearly = temp_box["Observed"
+variable].groupby(pd.Grouper(freq="Y")).quantile(0.75)

        median_yearly_proj = temp_box["Model"
+variable].groupby(pd.Grouper(freq="Y")).median()
        first_yearly_proj = temp_box["Model"
+variable].groupby(pd.Grouper(freq="Y")).quantile(0.25)
        third_yearly_proj = temp_box["Model"
+variable].groupby(pd.Grouper(freq="Y")).quantile(0.75)


        iqr = third_monthly - first_monthly

        fence_low  = first_monthly - (1.5*iqr)
        fence_high = third_monthly + (1.5*iqr)

        fence_low = fence_low.to_frame()

        fence_high = fence_high.to_frame()

        #print(fence_high)
        currMonth = 0
        prevMonth = 0
        tempx = 0 #this will count the fence_low and fence_high, incrementing
when month changed

        newDf = temp_box
        #pd.set_option("display.max_rows", None, "display.max_columns", None)
        #print(newDf)

        for index, row  in newDf.iterrows():
            #go over each row
            #check which month and year and based on that grab the proper
fences
            #once you have them
            currDate = newDf.at[index,'Date']

            #now grab the year and month of the date
            #currDate = dt.datetime.strptime(currDate,'%Y-%m-%d')

            #now get current month
            currMonth = currDate.month


            #if curr Month and previous month are not same then we increase
tempx
            #and make previous month into current
            if prevMonth != currMonth:
                if (prevMonth != 0):
                    tempx = tempx + 1

                prevMonth = currMonth



            currLow = fence_low.iat[tempx,0]
            currHigh = fence_high.iat[tempx,0]
```

```
            #print ("tempx : " + str(tempx) + " currHigh : " + str(currHigh)
)
            #now we remove rows from newDf based on low and high
            if (newDf.at[index,"Observed" +variable] > currHigh) or
(newDf.at[index,"Observed" +variable] < currLow ):
                #drop if the value is outside the range
                #print( "current newDf var: " + str(newDf.at[index,"Observed"
+variable]) +  " current high : " + str(currHigh) + " current Date:" +
str(currDate))
                newDf.drop(index, inplace=True)


        #df_out = temp_box.query('(@first_monthly - 1.5 * @iqr) <= ' +
"Observed" +variable + ' <= (@third_monthly + 1.5 * @iqr)')


        #df_out = (temp_box["Observed" +variable] >= fence_low) &
(temp_box["Observed" +variable] <= fence_high)
        #return df.loc[filter]

        #df_out = temp_box.loc[(temp_box["Observed"
+variable].groupby(pd.Grouper(freq="M")).min() > fence_low) &
(temp_box["Observed" +variable].groupby(pd.Grouper(freq="M")).min() <
fence_high)]

         #df_out = temp_box[(np.abs(stats.zscore(temp_box[1])) < 3)]
        max_monthly = newDf["Observed"
+variable].groupby(pd.Grouper(freq="M")).max()
        min_monthly = newDf["Observed"
+variable].groupby(pd.Grouper(freq="M")).min()
        temp_monthly_box.insert(0,'Date',date_monthly_obs)
        temp_monthly_box.insert(1, 'Minimum', min_monthly)
        temp_monthly_box.insert(2, 'First Quartile', first_monthly)
        temp_monthly_box.insert(3, 'Median', median_monthly)
        temp_monthly_box.insert(4, 'Third Quartile', third_monthly)
        temp_monthly_box.insert(5, 'Maximum', max_monthly)


        temp_monthly_box['Date'] = temp_monthly_box['Date'].dt.strftime('%b,
%Y')
        #projected
        temp_monthly_proj = pd.DataFrame()

        date_monthly_proj = tempdataMonth['Date']
        #max_monthly_proj = temp_box["Model"
+variable].groupby(pd.Grouper(freq="M")).max()
        #min_monthly_proj = temp_box["Model"
+variable].groupby(pd.Grouper(freq="M")).min()
#         median_monthly_proj = temp_box["Model"
+variable].groupby(pd.Grouper(freq="M")).median()
#         first_monthly_proj = temp_box["Model"
+variable].groupby(pd.Grouper(freq="M")).quantile(0.25)
#         third_monthly_proj = temp_box["Model"
+variable].groupby(pd.Grouper(freq="M")).quantile(0.75)

        iqr2 = third_monthly_proj - first_monthly_proj
        fence_low2  = first_monthly_proj - (1.5*iqr2)
```

```
        fence_high2 = third_monthly_proj + (1.5*iqr2)
        fence_low2 = fence_low2.to_frame()

        fence_high2 = fence_high2.to_frame()

        #print(fence_high)
        currMonth = 0
        prevMonth = 0
        tempx = 0 #this will count the fence_low and fence_high, incrementing
when month changed
        currLow = 0
        currHigh = 0
        newDf2 = temp_box
        #pd.set_option("display.max_rows", None, "display.max_columns", None)
        #print(newDf)

        for index, row  in newDf2.iterrows():
            #go over each row
            #check which month and year and based on that grab the proper
fences
            #once you have them
            currDate = newDf2.at[index,'Date']

            #now grab the year and month of the date
            #currDate = dt.datetime.strptime(currDate,'%Y-%m-%d')

            #now get current month
            currMonth = currDate.month


            #if curr Month and previous month are not same then we increase
tempx
            #and make previous month into current
            if prevMonth != currMonth:
                if (prevMonth != 0):
                    tempx = tempx + 1

                prevMonth = currMonth



            currLow = fence_low2.iat[tempx,0]
            currHigh = fence_high2.iat[tempx,0]
            #print ("tempx : " + str(tempx) + " currHigh : " + str(currHigh)
)
            #now we remove rows from newDf based on low and high
            if (newDf2.at[index,"Model" +variable] > currHigh) or
(newDf2.at[index,"Model" +variable] < currLow ):
                #drop if the value is outside the range
                #print( "current newDf var: " + str(newDf.at[index,"Observed"
+variable]) +  " current high : " + str(currHigh) + " current Date:" +
str(currDate))
                newDf2.drop(index, inplace=True)



        pd.set_option("display.max_rows", None, "display.max_columns", None)
```

```
        #print(newDf2)
        max_monthly_proj = newDf2["Model"
+variable].groupby(pd.Grouper(freq="M")).max()
        min_monthly_proj = newDf2["Model"
+variable].groupby(pd.Grouper(freq="M")).min()


        temp_monthly_proj.insert(0,'Date',date_monthly_proj)
        temp_monthly_proj.insert(1, 'Minimum', min_monthly_proj)
        temp_monthly_proj.insert(2, 'First Quartile', first_monthly_proj)
        temp_monthly_proj.insert(3, 'Median', median_monthly_proj)
        temp_monthly_proj.insert(4, 'Third Quartile', third_monthly_proj)
        temp_monthly_proj.insert(5, 'Maximum', max_monthly_proj)

        temp_monthly_proj['Date'] =
temp_monthly_proj['Date'].dt.strftime('%b, %Y')


        tempdataYear = data_all
        tempdataYear['Date'] = pd.to_datetime(tempdataYear['Date'])
        tempdataYear = (tempdataYear.set_index('Date').resample('Y').var())
        tempdataYear['Date'] = tempdataYear.index

        temp_yearly_box = pd.DataFrame()

        #temp_yearly_box['Date'] = temp_yearly_box.index

        date_yearly = tempdataYear['Date']
        #max_yearly = temp_box["Observed"
+variable].groupby(pd.Grouper(freq="Y")).max()
        #min_yearly = temp_box["Observed"
+variable].groupby(pd.Grouper(freq="Y")).min()

        iqr3 = third_yearly - first_yearly
        fence_low3  = first_yearly - (1.5*iqr3)
        fence_high3 = third_yearly + (1.5*iqr3)

        fence_low3 = fence_low3.to_frame()

        fence_high3 = fence_high3.to_frame()

        #print(fence_high)
        currYear = 0
        prevYear = 0
        tempx = 0 #this will count the fence_low and fence_high, incrementing
when month changed

        newDf3 = temp_box
        #pd.set_option("display.max_rows", None, "display.max_columns", None)
        #print(newDf)

        for index, row  in newDf3.iterrows():
            #go over each row
            #check which month and year and based on that grab the proper
fences
            #once you have them
            currDate = newDf3.at[index,'Date']
```

```
            #now grab the year and month of the date
            #currDate = dt.datetime.strptime(currDate,'%Y-%m-%d')

            #now get current year
            currYear = currDate.year


            #if curr Month and previous month are not same then we increase
tempx
            #and make previous month into current
            if prevYear != currYear:
                if (prevYear != 0):
                    tempx = tempx + 1

                prevYear = currYear



            currLow = fence_low3.iat[tempx,0]
            currHigh = fence_high3.iat[tempx,0]
            #print ("tempx : " + str(tempx) + " currHigh : " + str(currHigh)
)
            #now we remove rows from newDf based on low and high
            if (newDf3.at[index,"Observed" +variable] > currHigh) or
(newDf3.at[index,"Observed" +variable] < currLow ):
                #drop if the value is outside the range
                #print( "current newDf var: " + str(newDf.at[index,"Observed"
+variable]) +  " current high : " + str(currHigh) + " current Date:" +
str(currDate))
                newDf3.drop(index, inplace=True)



        max_yearly = newDf3["Observed"
+variable].groupby(pd.Grouper(freq="Y")).max()
        min_yearly = newDf3["Observed"
+variable].groupby(pd.Grouper(freq="Y")).min()


        temp_yearly_box.insert(0,'Date',date_yearly)
        temp_yearly_box.insert(1, 'Minimum', min_yearly)
        temp_yearly_box.insert(2, 'First Quartile', first_yearly)
        temp_yearly_box.insert(3, 'Median', median_yearly)
        temp_yearly_box.insert(4, 'Third Quartile', third_yearly)
        temp_yearly_box.insert(5, 'Maximum', max_yearly)

        temp_yearly_box['Date'] = temp_yearly_box['Date'].dt.strftime('%Y')

        temp_yearly_proj = pd.DataFrame()

        date_yearly_proj = tempdataYear['Date']


        iqr4 = third_yearly_proj - first_yearly_proj
        fence_low4  = first_yearly_proj - (1.5*iqr4)
        fence_high4 = third_yearly_proj + (1.5*iqr4)
```

```
        fence_low4 = fence_low4.to_frame()

        fence_high4 = fence_high4.to_frame()

        #print(fence_high)
        currYear = 0
        prevYear = 0
        tempx = 0 #this will count the fence_low and fence_high, incrementing
when month changed

        newDf4 = temp_box
        #pd.set_option("display.max_rows", None, "display.max_columns", None)
        #print(newDf)

        for index, row  in newDf4.iterrows():
            #go over each row
            #check which month and year and based on that grab the proper
fences
            #once you have them
            currDate = newDf4.at[index,'Date']

            #now grab the year and month of the date
            #currDate = dt.datetime.strptime(currDate,'%Y-%m-%d')

            #now get current month
            currMonth = currDate.year


            #if curr Month and previous month are not same then we increase
tempx
            #and make previous month into current
            if prevYear != currYear:
                if (prevYear != 0):
                    tempx = tempx + 1

                prevYear = currYear



            currLow = fence_low4.iat[tempx,0]
            currHigh = fence_high4.iat[tempx,0]
            #print ("tempx : " + str(tempx) + " currHigh : " + str(currHigh)
)
            #now we remove rows from newDf based on low and high
            if (newDf4.at[index,"Model" +variable] > currHigh) or
(newDf4.at[index,"Model" +variable] < currLow ):
                #drop if the value is outside the range
                #print( "current newDf var: " + str(newDf.at[index,"Observed"
+variable]) +  " current high : " + str(currHigh) + " current Date:" +
str(currDate))
                newDf4.drop(index, inplace=True)


        pd.set_option("display.max_rows", None, "display.max_columns", None)
        print(newDf4["Model" +variable].groupby(pd.Grouper(freq="Y")))
        print("AHHHHHHHHH NEW UHSAUDUH UISHAUI DH")
```

```
        max_yearly_proj = newDf4["Model"
+variable].groupby(pd.Grouper(freq="Y")).max()
        min_yearly_proj = newDf4["Model"
+variable].groupby(pd.Grouper(freq="Y")).min()


        pd.set_option("display.max_rows", None, "display.max_columns", None)
        print(min_yearly_proj)

        temp_yearly_proj.insert(0,'Date',date_yearly_proj)
        temp_yearly_proj.insert(1, 'Minimum', min_yearly_proj)
        temp_yearly_proj.insert(2, 'First Quartile', first_yearly_proj)
        temp_yearly_proj.insert(3, 'Median', median_yearly_proj)
        temp_yearly_proj.insert(4, 'Third Quartile', third_yearly_proj)
        temp_yearly_proj.insert(5, 'Maximum', max_yearly_proj)

        temp_yearly_proj['Date'] = temp_yearly_proj['Date'].dt.strftime('%Y')


        #use temp_box_year for year wise


        #seasonal plot here
        #have to creat a DF where each year is a column and one column is the
month
        #under the month column is simply months, and under the year column
is the values


        temp_yearly_proj = temp_yearly_proj.round(2)
        temp_yearly_box = temp_yearly_box.round(2)

        temp_monthly_proj = temp_monthly_proj.round(2)
        temp_monthly_box = temp_monthly_box.round(2)


        seasonalfig = px.line(seasonalDF, x="Month", y=seasonalDF.columns)


        seasonalfig.update_yaxes(
            title = columnName + " " +   measurement
        )
        seasonalfig.update_layout(title = "Observed "+ columnName+ "
Seasonality Plots",legend_title=dict(text = "Year"))

        seasonalfig2 = px.line(seasonalDF2, x="Month", y=seasonalDF.columns)

        boxfig3.update_layout(title = 'Observed ' + columnName + ': Year-wise
Box Plots')

        boxfig4.update_layout(title = 'Predicted ' + columnName + ': Year-
wise Box Plots')

        boxfig1.update_layout(title = 'Observed ' + columnName + ': Month-
wise Box Plots')
```

```
        boxfig2.update_layout(title = 'Predicted ' + columnName + ': Month-
wise Box Plots')

        seasonalfig2.update_yaxes(
            title = columnName + " " +   measurement
        )
        seasonalfig2.update_layout(title = "Predicted "+ columnName+ "
Seasonality Plots",legend_title=dict(text = "Year"))


        tempfig = make_subplots(specs=[[{"secondary_y": True}]])



        tempfig.add_trace(go.Scatter(x=data_all['Date'], y=data_all['Bias'],
fill='tozeroy', name = "Bias"),secondary_y=True) # fill to trace0 y
        tempfig.add_trace(go.Scatter(x=data_all['Date'],
y=data_all["Observed" +variable], name = "Observed"),secondary_y=False) #
fill down to xaxis
        tempfig.update_layout(
            title_x=0.5,
            xaxis_title="Date",
            yaxis_title=columnName + ' ' + measurement,
            showlegend=True,
            yaxis_range=[-75,150],
#             margin=dict(l=5,r=5,b=0,t=5,(specs=[[{"secondary_y":
True}]])pad=5)
            )
        tempfig.update_yaxes(title = "Bias", range = [-75,150],secondary_y =
True)
        tempfig.update_layout(title = 'Bias Graph')




#        pio.kaleido.scope.mathjax = None
        #print out all figs
#         fig.write_image("Images/MainGraph.png",engine='orca')
#         tempfig.write_image("Images/Biasfig.png",engine='orca')
#         boxfig1.write_image("Images/Boxfig1.png",engine='orca')
#         boxfig2.write_image("Images/Boxfig2.png",engine='orca')
#         boxfig3.write_image("Images/Boxfig3.png",engine='orca')
#         boxfig4.write_image("Images/Boxfig4.png",engine='orca')
#         seasonalfig.write_image("Images/Seasonalfig1.png",engine='orca')
#         seasonalfig2.write_image("Images/Seasonalfig2.png",engine='orca')

        print(temp_yearly_proj)
        print(boxfig4)
        if variable == 'pr':
            percipValue ={'float' : 'left','margin' : '10px','width':
'800px', 'display': 'inline-block'}
            percipValueRight ={'float' : 'right','margin' : '10px','width':
'800px', 'display': 'inline-block'}
        else:
            percipValue ={'float' : 'left','margin' : '10px','width':
'800px', 'display': 'none'}
```

```
            percipValueRight ={'float' : 'right','margin' : '10px','width':
'800px', 'display': 'none'}

        if variable == 'rlds':
            biasShow ={'float' : 'right','margin' : '10px','width': '800px',
'display': 'none'}
        else:
            biasShow ={'float' : 'right','margin' : '10px','width': '800px',
'display': 'inline-block'}

        #print(str(percipValue))
#         biasfig = go.Figure(data=go.Scatter(
#                     x=data_all['Date'],
#                     y=data_all["Observed" +variable],
#                     xhoverformat="%d %b, %Y",
#                     yhoverformat = ".2f",
#                     error_y=dict(
#                         type='data',
#                         array=data_all['Bias']
#                     )))

#         biasfig.update_yaxes(title_text="Observed Value of " + columnName,
secondary_y=False)
#         biasfig.update_yaxes(title_text="Error Bar Value",
secondary_y=True)

        #create the output that will go on the page
        graph = html.Div([
            html.Div(id = 'graphDIV', children =[
                html.Button("Download Data Model",
id="btn_csv",style={'background-color': '#4CAF50','border': 'none','color':
'white','padding': '15px 32px','text-align': 'center','text-decoration':
'none','display': 'inline-block','font-size': '16px','margin': '4px 2px'}),
                dcc.Download(id="download-dataframe-csv"),
                (dcc.Graph(
                   id='Graph1',
                   figure=fig,
                    style={
                        'height' : '300px',
                        'margin' : '50px'
                    }
               ))
            ],style = {'display': 'inline-block'}),


            html.Div(id = 'frequencyObs', children=[
            html.Div([
                html.Button("Download Analysis",
id="btn_analysis",style={'background-color': '#4CAF50','border':
'none','color': 'white','padding': '15px 32px','text-align': 'center','text-
decoration': 'none','display': 'inline-block','font-size': '16px','margin':
'4px 2px'}),
                dcc.Download(id="download-analysis"),
                    ],style={'float' : 'left','margin' : '10px','margin-top':
'350px','width': '100%', 'display': 'inline-block'}),

            html.Div([
```

```
            html.H3('Number of days where observed ' + columnName + ' is >/<
than  ECCC 30 year normals', style={'width': '800px','margin' : '10px','font-
size': '12px', 'float': 'left', 'font-family': 'Verdana'}),
            dash_table.DataTable(
                id='table',
                columns=[{"name": i, "id": i} for i in stats1.columns],
                data=stats1.to_dict('records'),
            )
            ],style=percipValue),

             html.Div([
                 html.H3('Number of days where Predicted ' + columnName + '
is >/< than  ECCC 30 year normal', style={'width': '800px','margin' :
'10px','font-size': '12px', 'float': 'left', 'font-family': 'Verdana'}),
                 dash_table.DataTable(
                id='table',
                columns=[{"name": i, "id": i} for i in stats2.columns],
                data=stats2.to_dict('records'),
            )
             ],style=percipValueRight),

            html.Div([
                html.H3('Annual ' + columnName + '', style={'width':
'800px','font-size': '12px','margin' : '10px', 'font-family': 'Verdana'}),
                dash_table.DataTable(
                id='table',
                columns=[{"name": i, "id": i} for i in anualPrecip.columns],
                data=anualPrecip.to_dict('records'),
                style_cell={
                    'minWidth': '10px', 'width': '10px', 'maxWidth': '10px'
                }
            ),
            ],style=percipValue),

            html.Div([],style={'float' : 'left','margin' : '10px','width':
'100%', 'display': 'inline-block'}),
            html.Div([
                html.H3('Observed ' + columnName + ': Monthly Means',
style={'width': '800px','margin' : '10px', 'font-size': '12px', 'float':
'left', 'font-family': 'Verdana'}),
                dash_table.DataTable(
                id='table',
                columns=[{"name": i, "id": i} for i in stats3.columns],
                data=stats3.to_dict('records')
            ),
            ],style={'float' : 'left','margin' : '10px','width': '800px',
'display': 'inline-block'}),
            html.Div([
                html.H3('Predicted ' + columnName + ': Monthly Means',
style={'width': '800px','margin' : '10px','font-size': '12px', 'float':
'left', 'font-family': 'Verdana'}),
                dash_table.DataTable(
                id='table',
                columns=[{"name": i, "id": i} for i in stats4.columns],
                data=stats4.to_dict('records')

            ),
```

```
            ],style={'float' : 'right','margin' : '10px','width': '800px',
'display': 'inline-block'}),
             html.Div([
                html.H3('Observed ' + columnName + ': Monthly Medians',
style={'width': '800px','margin' : '10px', 'font-size': '12px', 'float':
'left', 'font-family': 'Verdana'}),
                dash_table.DataTable(
                id='table',
                columns=[{"name": i, "id": i} for i in
medianMonthlyObs.columns],
                data=medianMonthlyObs.to_dict('records')
            ),
            ],style={'float' : 'left','margin' : '10px','width': '800px',
'display': 'inline-block'}),
            html.Div([
                html.H3('Predicted ' + columnName + ': Monthly Medians',
style={'width': '800px','margin' : '10px', 'font-size': '12px', 'float':
'left', 'font-family': 'Verdana'}),
                dash_table.DataTable(
                id='table',
                columns=[{"name": i, "id": i} for i in
medianMonthlyProj.columns],
                data=medianMonthlyProj.to_dict('records')
            ),
            ],style={'float' : 'right','margin' : '10px','width': '800px',
'display': 'inline-block'}),
            html.Div([

                html.H3('Observed ' + columnName + ': Monthly Standard
Deviation', style={'width': '800px','margin' : '10px','font-size': '12px',
'float': 'left', 'font-family': 'Verdana'}),
                dash_table.DataTable(
                id='table',
                columns=[{"name": i, "id": i} for i in stats5.columns],
                data=stats5.to_dict('records')
            ),
            ],style={'float' : 'left','margin' : '10px','width': '800px',
'display': 'inline-block'}),

            html.Div([
                html.H3('Predicted ' + columnName + ': Monthly Standard
Deviation', style={'width': '800px','margin' : '10px','font-size': '12px',
'float': 'left', 'font-family': 'Verdana'}),
                dash_table.DataTable(
                id='table',
                columns=[{"name": i, "id": i} for i in stats6.columns],
                data=stats6.to_dict('records')
            )

            ],style={'float' : 'right','margin' : '10px','width': '800px',
'display': 'inline-block'}),



            html.Div([],style={'float' : 'left','margin' : '10px','width':
'100%', 'display': 'inline-block'}),
```

```
            html.Div([

                #html.H3('Bias Graph
', style={'width': 'auto','margin' : '10px','font-size': '12px', 'float':
'left', 'font-family': 'Verdana'}),
                (dcc.Graph(
                   id='Graph1',
                   figure=tempfig
                   ))

            ],style={'float' : 'left','margin' : '10px','width': '800px',
'display': 'inline-block'}),

              html.Div([
                html.H3('Bias between Observed and Projected ' + columnName +
'', style={ 'width': '800px','margin' : '10px','font-size': '12px', 'float':
'left', 'font-family': 'Verdana'}),
                dash_table.DataTable(
                id='table',
                columns=[{"name": i, "id": i} for i in biasDF.columns],
                data=biasDF.to_dict('records'))


            ],style=biasShow),

               html.Div([],style={'float' : 'left','margin' : '10px','width':
'100%', 'display': 'inline-block'}),

                html.Div([
          #       html.H3('Observed ' + columnName + ': Month-wise Box Plots',
style={'width': '800px','margin' : '10px','font-size': '12px', 'float':
'left', 'font-family': 'Verdana'}),
                 (dcc.Graph(
                id='Graph1',
                figure=boxfig1
                 )),

            ],style={'float' : 'left','margin' : '10px','width': '800px',
'display': 'inline-block'}),

            html.Div([

       #    html.H3('Projected ' + columnName + ': Month-wise Box Plots',
style={ 'width': '800px','margin' : '10px','font-size': '12px', 'float':
'left', 'font-family': 'Verdana'}),
            (dcc.Graph(
               id='Graph1',
               figure=boxfig2
           )),

            ],style={'float' : 'right','margin' : '10px','width': '800px',
'display': 'inline-block'}),


            html.Div([
                dash_table.DataTable(
                id='table',
```

```
                columns=[{"name": i, "id": i} for i in
temp_monthly_box.columns],
                data=temp_monthly_box.to_dict('records')
            ),

            ],style={'float' : 'left','margin' : '10px','width': '800px',
'display': 'inline-block'}),

            html.Div([
                 dash_table.DataTable(
                id='table',
                columns=[{"name": i, "id": i} for i in
temp_monthly_proj.columns],
                data=temp_monthly_proj.to_dict('records')
            ),
            ],style={'float' : 'right','margin' : '10px','width': '800px',
'display': 'inline-block'}),


            html.Div([
#               html.H3('Observed ' + columnName + ': Year-wise Box Plots',
style={'width': '800px','margin' : '10px','font-size': '12px', 'float':
'left', 'font-family': 'Verdana'}),
                (dcc.Graph(
                   id='Graph1',
                   figure=boxfig3
               )),

            ],style={'float' : 'left','width': '800px', 'display': 'inline-
block'}),

              html.Div([
            #html.H3('Projected ' + columnName + ': Year-wise Box Plots',
style={'font-size': '12px', 'float': 'right', 'font-family': 'Verdana'}),


            (dcc.Graph(
               id='Graph1',
               figure=boxfig4
           )),

            ],style={'float' : 'right','width': '800px', 'display': 'inline-
block'}),


            html.Div([
               dash_table.DataTable(
                id='table',
                columns=[{"name": i, "id": i} for i in
temp_yearly_box.columns],
                data=temp_yearly_box.to_dict('records')
            ),

            ],style={'float' : 'left','margin' : '10px','width': '800px',
'display': 'inline-block'}),

              html.Div([
```

```
            dash_table.DataTable(
                id='table',
                columns=[{"name": i, "id": i} for i in
temp_yearly_proj.columns],
                data=temp_yearly_proj.to_dict('records')
            ),

            ],style={'float' : 'right','margin' : '10px','width': '800px',
'display': 'inline-block'}),


            html.Div([
#           html.H3('Observed and Projected Precipitation Seasonality Plots',
style={'font-size': '12px', 'font-family': 'Verdana'}),
                 (dcc.Graph(
               id='Graph1',
               figure=seasonalfig
           )),

            ],style={'float' : 'left','width': '800px', 'display': 'inline-
block'}),

              html.Div([
                (dcc.Graph(
                   id='Graph1',
                   figure=seasonalfig2,

               ))

            ],style={'float' : 'right','width': '800px', 'display': 'inline-
block'}),




            ],style = {'display':'inline-block','width' : '2000px'})



        ])

        return 'Analyze',graph,data_all.to_json(date_format='iso',
orient='split'),stats1.to_json(date_format='iso',
orient='split'),stats2.to_json(date_format='iso',
orient='split'),anualPrecip.to_json(date_format='iso',
orient='split'),stats3.to_json(date_format='iso',
orient='split'),stats4.to_json(date_format='iso',
orient='split'),stats5.to_json(date_format='iso',
orient='split'),stats6.to_json(date_format='iso',
orient='split'),stats7.to_json(date_format='iso',
orient='split'),stats8.to_json(date_format='iso',
orient='split'),biasDF.to_json(date_format='iso',
orient='split'),temp_monthly_box.to_json(date_format='iso',
orient='split'),temp_monthly_proj.to_json(date_format='iso',
orient='split'),temp_yearly_box.to_json(date_format='iso',
```

```
orient='split'),temp_yearly_proj.to_json(date_format='iso',
orient='split'),fig



@app.callback(
    Output("download-dataframe-csv", "data"),
    inputs = [Input("btn_csv", "n_clicks"), Input('main_data','data')],
    prevent_initial_call=True,
)
def func(n_clicks,data_all ):

    if n_clicks:
        df = pd.read_json(data_all, orient='split')
        return dcc.send_data_frame(df.to_csv, "mydf.csv")



@app.callback(
    Output("download-analysis", "data"),
    inputs = [Input("btn_analysis", "n_clicks"),
Input('observed_normals','data'),

Input('projected_normals','data'),Input('annual_data','data'),Input('observed
_mean','data'),

Input('projected_mean','data'),Input('observed_deviation','data'),Input('proj
ected_deviation','data'),

Input('observed_variance','data'),Input('projected_variance','data'),Input('b
ias_data','data'),

Input('observed_monthBox','data'),Input('projected_monthBox','data'),Input('o
bserved_yearBox','data'),
             Input('projected_yearBox','data'),Input('mainFig','data')],
    prevent_initial_call=True,
)
def
func(n_clicks,observedNormal,projectedNormal,annualData,observedMean,projecte
dMean,observedDeviation,projectedDeviation,observedVairance,projectedVariance
,biasData,observedMonthBox,projectedMonthBox,observedYearBox,projectedYearBox
,fig):

    if n_clicks:

        df1 = pd.read_json(observedNormal, orient='split')
        df2 = pd.read_json(projectedNormal, orient='split')
        df3 = pd.read_json(annualData, orient='split')
        df4 = pd.read_json(observedMean, orient='split')
        df5 = pd.read_json(projectedMean, orient='split')
        df6 = pd.read_json(observedDeviation, orient='split')
        df7 = pd.read_json(projectedDeviation, orient='split')
        df8 = pd.read_json(observedVairance, orient='split')
        df9 = pd.read_json(projectedVariance, orient='split')
        df10 = pd.read_json(biasData, orient='split')
        df11 = pd.read_json(observedMonthBox, orient='split')
        df12 = pd.read_json(projectedMonthBox, orient='split')
```

```
        df13 = pd.read_json(observedYearBox, orient='split')
        df14 = pd.read_json(projectedYearBox, orient='split')




        df3.reset_index(drop=True, inplace=True)
        df11.reset_index(drop=True, inplace=True)
        df12.reset_index(drop=True, inplace=True)
        df13.reset_index(drop=True, inplace=True)
        df14.reset_index(drop=True, inplace=True)




        with pd.ExcelWriter('DataAnalysis.xlsx') as writer:

            workbook = writer.book
            writer.sheets={'Project vs Observed
Graph':workbook.add_worksheet()}
            worksheet1 = writer.sheets['Project vs Observed Graph']
            worksheet1.insert_image('D3', 'Images/MainGraph.png')


            df1.to_excel(writer, sheet_name='Observed Normal',index=False)
            df2.to_excel(writer, sheet_name='Projected Normal',index=False)
            df3.to_excel(writer, sheet_name='Annual Data',index=False)
            df4.to_excel(writer, sheet_name='Observed Mean',index=False)
            df5.to_excel(writer, sheet_name='Projected Mean',index=False)
            df6.to_excel(writer, sheet_name='Observed Deviation',index=False)
            df7.to_excel(writer, sheet_name='Projected
Deviation',index=False)
            df8.to_excel(writer, sheet_name='Observed Variance',index=False)
            df9.to_excel(writer, sheet_name='Projected Variance',index=False)
            df10.to_excel(writer, sheet_name='Bias Data',index=False)
            df11.to_excel(writer, sheet_name='Observed Monthly Box
Data',index=False)
            df12.to_excel(writer, sheet_name='Projected Monthly Box
Data',index=False)
            df13.to_excel(writer, sheet_name='Observed Yearly Box
Data',index=False)
            df14.to_excel(writer, sheet_name='Projected Yearly Box
Data',index=False)


            writer.save()

        return dcc.send_file(writer)

#THIS IS TO DOWNLOAD THE AUTOMATED DATA to CSV


@app.callback(
    Output("download-dataframe-csv-auto", "data"),
    inputs = [Input("btn_csv_auto", "n_clicks"),
Input('main_data_auto','data')],
    prevent_initial_call=True,
```

```
)
def func(n_clicks,data_all ):

    if n_clicks:
        df = pd.read_json(data_all, orient='split')
        return dcc.send_data_frame(df.to_csv, "mydf.csv")

@app.callback(
    Output("download-analysis-auto", "data"),
    inputs = [Input("btn_analysis_auto", "n_clicks"),
Input('observed_normals_auto','data'),

Input('projected_normals_auto','data'),Input('annual_data_auto','data'),Input
('observed_mean_auto','data'),

Input('projected_mean_auto','data'),Input('observed_deviation_auto','data'),I
nput('projected_deviation_auto','data'),

Input('observed_variance_auto','data'),Input('projected_variance_auto','data'
),Input('bias_data_auto','data'),

Input('observed_monthBox_auto','data'),Input('projected_monthBox_auto','data'
),Input('observed_yearBox_auto','data'),

Input('projected_yearBox_auto','data'),Input('mainFig_auto','data')],
    prevent_initial_call=True,
)
def
func(n_clicks,observedNormal,projectedNormal,annualData,observedMean,projecte
dMean,observedDeviation,projectedDeviation,observedVairance,projectedVariance
,biasData,observedMonthBox,projectedMonthBox,observedYearBox,projectedYearBox
,fig):

    if n_clicks:

        df1 = pd.read_json(observedNormal, orient='split')
        df2 = pd.read_json(projectedNormal, orient='split')
        df3 = pd.read_json(annualData, orient='split')
        df4 = pd.read_json(observedMean, orient='split')
        df5 = pd.read_json(projectedMean, orient='split')
        df6 = pd.read_json(observedDeviation, orient='split')
        df7 = pd.read_json(projectedDeviation, orient='split')
        df8 = pd.read_json(observedVairance, orient='split')
        df9 = pd.read_json(projectedVariance, orient='split')
        df10 = pd.read_json(biasData, orient='split')
        df11 = pd.read_json(observedMonthBox, orient='split')
        df12 = pd.read_json(projectedMonthBox, orient='split')
        df13 = pd.read_json(observedYearBox, orient='split')
        df14 = pd.read_json(projectedYearBox, orient='split')




        df3.reset_index(drop=True, inplace=True)
        df11.reset_index(drop=True, inplace=True)
        df12.reset_index(drop=True, inplace=True)
        df13.reset_index(drop=True, inplace=True)
```

```
        df14.reset_index(drop=True, inplace=True)



        with pd.ExcelWriter('DataAnalysis.xlsx') as writer:

            workbook = writer.book
            writer.sheets={'Project vs Observed
Graph':workbook.add_worksheet()}
            worksheet1 = writer.sheets['Project vs Observed Graph']
            worksheet1.insert_image('D3', 'Images/MainGraph.png')


            df1.to_excel(writer, sheet_name='Observed Normal',index=False)
            df2.to_excel(writer, sheet_name='Projected Normal',index=False)
            df3.to_excel(writer, sheet_name='Annual Data',index=False)
            df4.to_excel(writer, sheet_name='Observed Mean',index=False)
            df5.to_excel(writer, sheet_name='Projected Mean',index=False)
            df6.to_excel(writer, sheet_name='Observed Deviation',index=False)
            df7.to_excel(writer, sheet_name='Projected
Deviation',index=False)
            df8.to_excel(writer, sheet_name='Observed Variance',index=False)
            df9.to_excel(writer, sheet_name='Projected Variance',index=False)
            df10.to_excel(writer, sheet_name='Bias Data',index=False)
            df11.to_excel(writer, sheet_name='Observed Monthly Box
Data',index=False)
            df12.to_excel(writer, sheet_name='Projected Monthly Box
Data',index=False)
            df13.to_excel(writer, sheet_name='Observed Yearly Box
Data',index=False)
            df14.to_excel(writer, sheet_name='Projected Yearly Box
Data',index=False)


            writer.save()

        return dcc.send_file(writer)



@app.callback(
    [Output("date-picker-range", "start_date"), Output("date-picker-range",
"end_date")],
    [Input("date_slider", "value")],
    [State("date-picker-range", "start_date"), State("date-picker-range",
"end_date")],
)
def update_date_range(slider_dates, date_range_start, date_range_end):
    start_yr, end_yr = slider_dates[0], slider_dates[1]

    if date_range_start is not None:
        date_range_start = str(start_yr) + date_range_start[4:]
    else:
        date_range_start = str(start_yr) + '-01-01'

    if date_range_end is not None:
```

```
        date_range_end = str(end_yr) + date_range_end[4:]
    else:
        date_range_end = str(end_yr) + '-01-01'

    return date_range_start, date_range_end


@app.callback(
    Output("displayDate", "children"),
    [Input("date-picker-range", "start_date"), Input("date-picker-range",
"end_date")],
)
def update_date_range(date_range_start, date_range_end):
    return f"You have selected dates from {date_range_start} to
{date_range_end}"



@app.callback(
    [Output("date-picker-range-auto", "start_date"), Output("date-picker-
range-auto", "end_date")],
    [Input("date_slider-auto", "value")],
    [State("date-picker-range-auto", "start_date"), State("date-picker-range-
auto", "end_date")],
)
def update_date_range(slider_dates, date_range_start, date_range_end):
    start_yr, end_yr = slider_dates[0], slider_dates[1]

    if date_range_start is not None:
        date_range_start = str(start_yr) + date_range_start[4:]
    else:
        date_range_start = str(start_yr) + '-01-01'

    if date_range_end is not None:
        date_range_end = str(end_yr) + date_range_end[4:]
    else:
        date_range_end = str(end_yr) + '-01-01'

    return date_range_start, date_range_end


@app.callback(
    Output("displayDate-auto", "children"),
    [Input("date-picker-range-auto", "start_date"), Input("date-picker-range-
auto", "end_date")],
)
def update_date_range(date_range_start, date_range_end):
    return f"You have selected dates from {date_range_start} to
{date_range_end}"




if __name__ == '__main__':
    app.run_server()
```