Capstone Project - Group 9 - Climate Change

John Moen
Carmlina Sandoval
Amer Lam
Sanil Veeravu

The Heat Map - Analyzing Climate Change
Final Paper

Introduction

Climate Change is one of the most critical issues of our time. From change in weather patterns that impact food production, to rising sea levels that increase the risk of catastrophic flooding, spreading wildfires the impacts of climate change are global in scope and unprecedented in scale.

In creating The Heat Map we attempted to demonstrate a rise in temperatures or other extreme weather patterns using historical data and also build a machine learning model which could predict future temperatures. In addition to the model, two Tableau dashboards and an interactive database would be built to support our data story.

Through this research using historical climate change data from NOAA, we accomplished our task to research and predict the climate change expected by US states/countries for future years, and to show the possible impact to our environment.

Data Collection

The National Centers of Environmental Information maintains data from different stations all over the world with different metrics like temperature max/min, snow depth, precipitation and many more. They also provide data with details of stations. All this data was available in this URL - https://www.ncei.noaa.gov/pub/data/ghcn/daily/.

Data was available in different formats and the station data involved data for each month in a row with a column for each day of the month. This data was in a fixed-width-file format which needed to be converted to csv. Initial work was done to convert this data into a row for each day, but with additional research we identified a similar dataset was available in the yearly folder that was already in csv format.

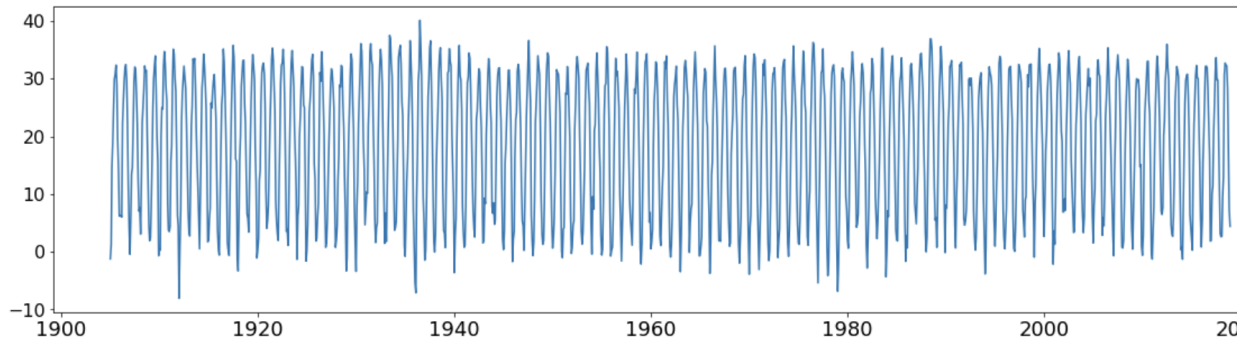
With the data being so huge, we build a loop to download one year at a time using python requests, perform the necessary aggregations/lookup and store only the required data into the required metric files

```
for i in range(118):
    yr = i + 1905
    print(f"processing {yr}")
    remote_url = f'https://www.ncei.noaa.gov/pub/data/ghcn/daily/by_year/{yr}.csv.gz'
    if os.path.exists('local_copy.csv.gz'):
        os.remove('local_copy.csv.gz')
    if os.path.exists('local_copy.csv'):
        os.remove('local_copy.csv')
    #time.sleep(10)
    print(f"Going to Run:{remote_url}")
    r = requests.get(remote_url, headers={'User-Agent': 'Mozilla/5.0'})
    open('local_copy.csv.gz', 'wb').write(r.content)
    #time.sleep(10)
    print("Copy Complete")
    with gzip.open('local_copy.csv.gz', 'rb') as f_in:
        with open('local_copy.csv', 'wb') as f_out:
            shutil.copyfileobj(f_in, f_out)
    print("Unzip Complete")
    df = pd.read_csv('local_copy.csv', names=["station_id", "date", "metric", "value", "measurement_flag", "quality
    df = df[df.station_id.str[:2] == "US"]
    df = df[(df.metric == "SNOW")|(df.metric == "PRCP")|(df.metric == "TMIN")|(df.metric == "TMAX")]
    print("Filtering Complete")
    df['date'] = pd.to_datetime(df['date'], format='%Y%m%d')
    df = df[df.quality_flag.isnull()]
    print("Cleanup Complete")
    df = pd.merge(df, stations_df, how='inner', on = 'station_id')
    df['value'] = df.apply(lambda x: x['value'] if x['metric']=="SNOW" else x['value']/10, axis=1)
    df.drop(["station_id","station_name","measurement_flag","quality_flag","source_flag","observation_time","latitu
    print("Conversion Complete")
    tmindata = df[df.metric=="TMIN"].groupby(["state_code", "date"]).agg("min").reset_index()
    tmaxdata = df[df.metric=="TMAX"].groupby(["state_code", "date"]).agg("max").reset_index()
    prcpdata = df[df.metric=="PRCP"].groupby(["state_code", "date"]).agg("max").reset_index()
    snowdata = df[df.metric=="SNOW"].groupby(["state_code", "date"]).agg("max").reset_index()
    tmindata.drop(["metric"], axis=1, inplace=True)
    tmaxdata.drop(["metric"], axis=1, inplace=True)
    prcpdata.drop(["metric"], axis=1, inplace=True)
    snowdata.drop(["metric"], axis=1, inplace=True)
    print("Aggregation Complete")
    tmindata.to_csv('../data/cleaned_data/tmindata.csv', mode='a', index=False, header=False)
    tmaxdata.to_csv('../data/cleaned_data/tmaxdata.csv', mode='a', index=False, header=False)
    prcpdata.to_csv('../data/cleaned_data/prcpdata.csv', mode='a', index=False, header=False)
    snowdata.to_csv('../data/cleaned_data/snowdata.csv', mode='a', index=False, header=False)
```

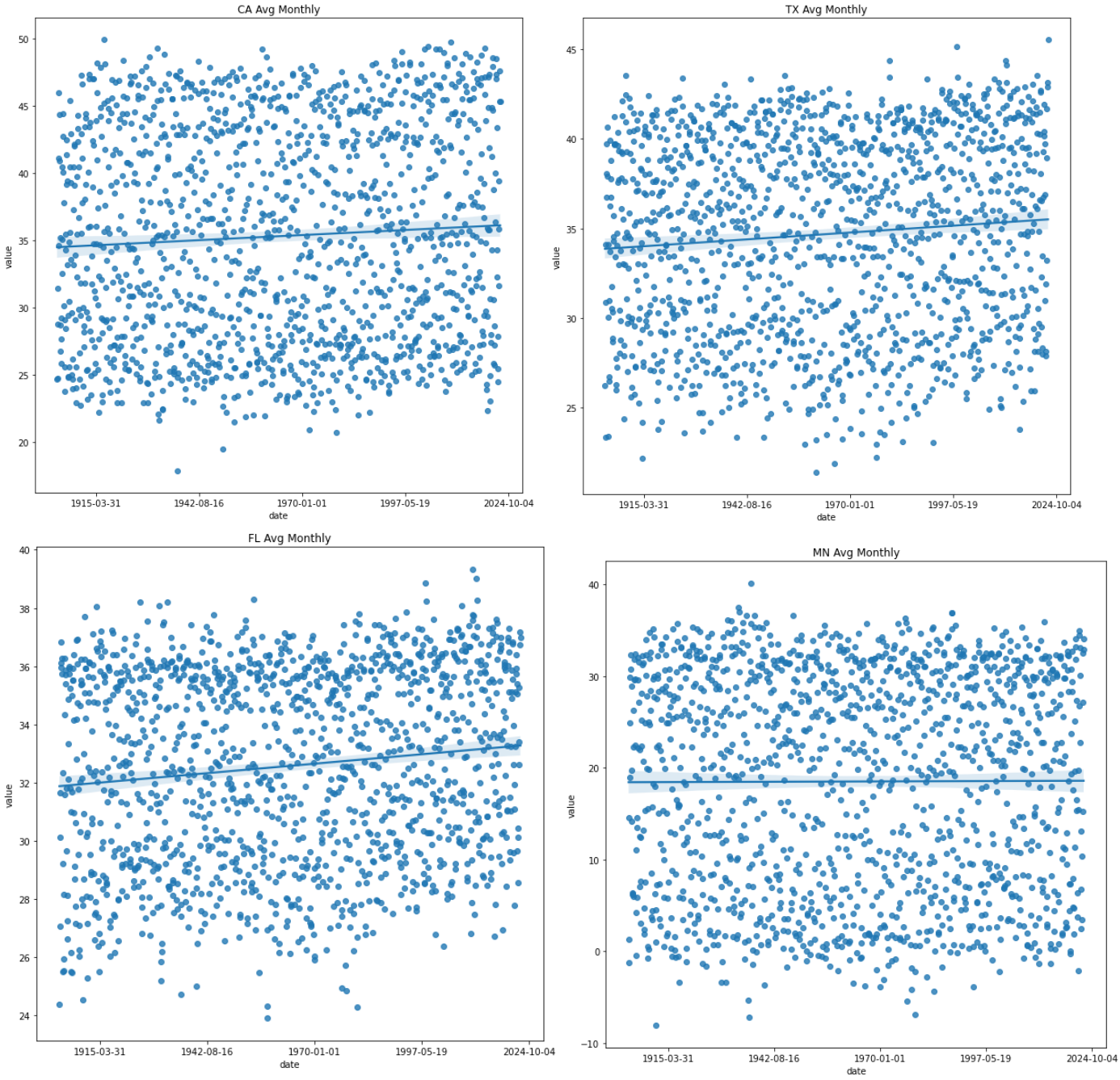
We were able to condense the data from multiple GBs to 34 MB files.

One key challenge that came during this process was the quality of the data. The initial graph was all around the place with temperature upto 5000 C. After more data exploration identified a column called quality flag which provides certification of the data and once this filter was applied we got a smooth dataset for metrics.

```
WARNING:matplotlib.legend:No handles with labels found to put in legend.
```



During the initial data analysis we looked at the data in many different ways, including by min/max temperatures, number of data points above a certain threshold over time, and temperature average. Simple linear regression plots began to show the upward trend in temperatures we were expecting when we analyzed average temperatures by state over the



years. Above are four plots showing the regression line for states Texas, California, Florida, and Minnesota.

Database

Initial data was built using daily data from stations. This had the following structure:

```
station_id varchar pk fk >- stations.station_id
date date pk
metric varchar
value float
```

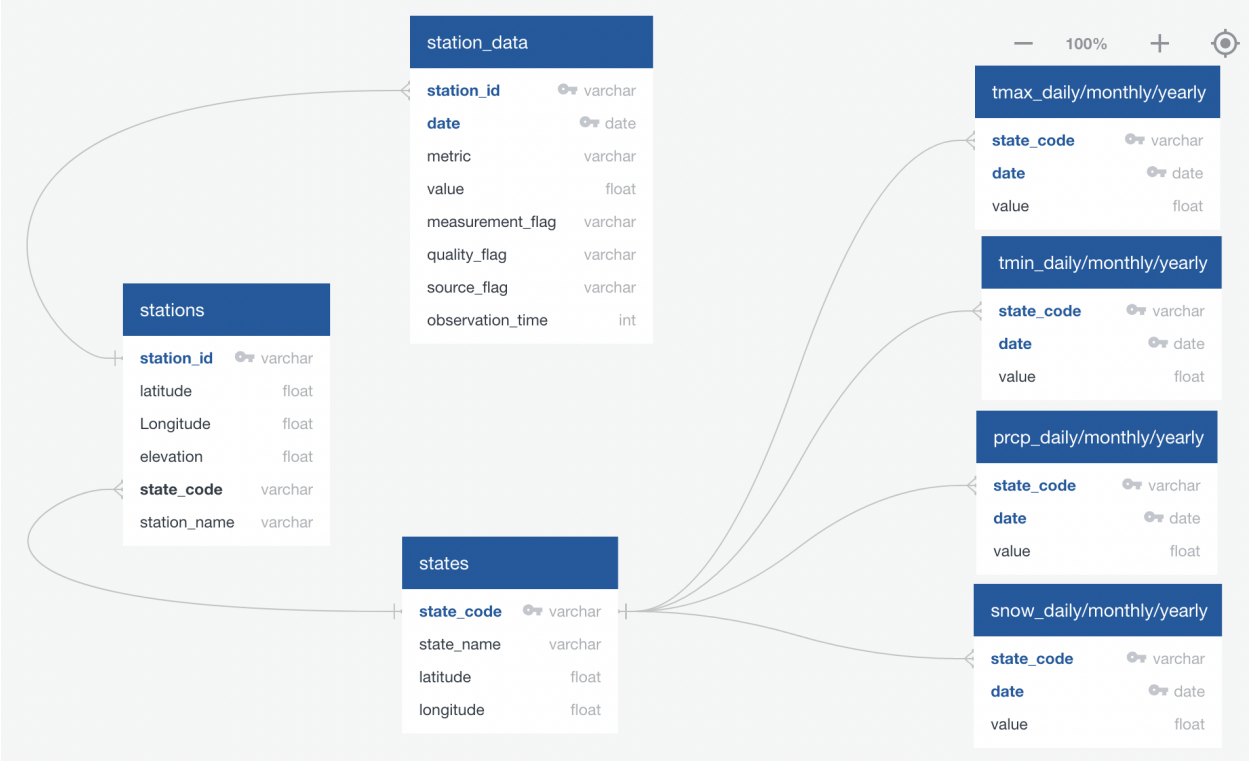
```
measurement_flag varchar
quality_flag varchar
source_flag varchar
observation_time int
```

To get to the values by state, a lookup table was built referring station code to state code

```
station_id varchar pk
latitude float
Longitude float
elevation float
state_code varchar FK >- states.state_code
station_name varchar
```

Now selecting just the required states and country based on station id name, data was aggregated to daily , monthly and yearly levels for our analysis.

Below is the ERD of tables used for our research.



Creating the SQlite database - using a jupyter notebook, pandas and SQLalchemy we were able to construct our database. First we loaded the CSV into the notebook, creating a dataframe for each soon to be table for our database. In total, these was four files and data frames created.

```python
data = "../data/cleaned_data/tmaxdata_max_with_lat_long_yearly.csv"
tmax_df = pd.read_csv(data)
tmax_df.head()
```

Python

	state_code	date	value	state_name	latitude	longitude
0	AK	1905-01-01	36.1	Alaska	66.160507	-153.369141
1	AK	1906-01-01	36.1	Alaska	66.160507	-153.369141
2	AK	1907-01-01	37.8	Alaska	66.160507	-153.369141
3	AK	1908-01-01	31.7	Alaska	66.160507	-153.369141
4	AK	1909-01-01	34.4	Alaska	66.160507	-153.369141

```python
data = "../data/cleaned_data/tmindata_min_with_lat_long_yearly.csv"
tmin_df = pd.read_csv(data)
tmin_df.head()
```

Python

Below were few sample join queries used:

1. To get the tmax and tmin information from one query

In [35]:
```python
conn = engine.connect()
query = """
        SELECT
            tmax.state_code,tmax.state_name,tmax.date,tmax.value temp_max,tmin.value temp_min
        fROM
            tmax tmax inner join tmin tmin on (tmax.state_code=tmin.state_code and tmax.date=tmin.date) ;
        """
climate_data_df = pd.read_sql(query, conn)
climate_data_df.head()
```

Out[35]:

	state_code	state_name	date	temp_max	temp_min
0	AK	Alaska	1905-01-01	36.1	-52.8
1	AK	Alaska	1906-01-01	36.1	-58.9
2	AK	Alaska	1907-01-01	37.8	-54.4
3	AK	Alaska	1908-01-01	31.7	-54.4
4	AK	Alaska	1909-01-01	34.4	-56.1

2. To get the latitude, longitude and state code using state lookup table for tmax variance

In [51]:
```python
conn = engine.connect()
query = """
        SELECT
             sl.state_code,sl.state_name,tv.year,tv.value tmax_var,sl.latitude,sl.longitude
        fROM
            tmax_var tv inner join states_lookup sl
            on (tv.state_name=sl.state_name) ;
        """
tmax_var_with_lat_long = pd.read_sql(query, conn)
tmax_var_with_lat_long.head()
```

Out[51]:

	state_code	state_name	year	tmax_var	latitude	longitude
0	AL	Alabama	1960	0.977391	32.318230	-86.902298
1	AK	Alaska	1960	1.802526	66.160507	-153.369141
2	AZ	Arizona	1960	0.908129	34.048927	-111.093735
3	AR	Arkansas	1960	0.876404	34.799999	-92.199997
4	CA	California	1960	1.664075	36.778259	-119.417931

Next, we needed to connect to the database and create the tables in the database using the below code.

```
# get all tables
inspector_gadget = inspect(engine)
tables = inspector_gadget.get_table_names()
for table in tables:
    print(table)

    # get all columns in table
    columns = inspector_gadget.get_columns(table)
    for column in columns:
        print(column)
    print()
```

```
database_path = "weather.sqlite"
conn_string = f"sqlite:///{database_path}"

# Create an engine that can talk to the database
engine = create_engine(conn_string)
```

```
def create_metric_table(metric_name):
    sql=f""" CREATE TABLE {metric_name} (
        state_code varchar,
        date int,
        value float,
        state_name varchar,
        latitude float,
        longitude float
    )"""
    engine.execute(sql)
```

```
create_metric_table("tmax")
create_metric_table("tmin")
create_metric_table("snow")
create_metric_table("prcp")
```

Next, we stored our tables in the database and ran a SQL query out of the database for each table.

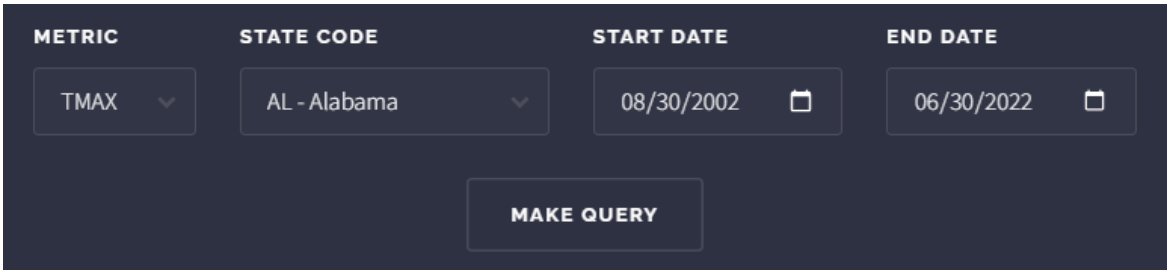
```
tmax_df.to_sql("tmax", con=engine, method="multi", index=False, if_exists="replace")
```

```
tmin_df.to_sql("tmin", con=engine, method="multi", index=False, if_exists="replace")
snow_df.to_sql("snow", con=engine, method="multi", index=False, if_exists="replace")
prcp_df.to_sql("prcp", con=engine, method="multi", index=False, if_exists="replace")
```

```
conn = engine.connect()

query = """
        SELECT
            *
        FROM
            TMAX;
        """
tmax_final = pd.read_sql(query, conn)
tmax_final.head()
```

The sqlite database needed to be accessed by a query built by the user's inputs, which was work that needed to be split across several different files. Firstly the user needs to select the desired query parameters using the form below.



An event listener in a javascript file waited for the query button to be pressed, then pulled the values of the entry fields into a payload of information. The state selector was populated using an AJAX 'GET' and like the metric value, was already formatted appropriately for the query. The date needed to be in our YYYY-MM-DD format and we so it had to be formatted to fit our database, which was done with the code snippet below:

```
function format_date(d) {
    return (d.getFullYear() + "-" + ("0"+(d.getMonth()+1)).slice(-2) + "-" + ("0" + d.getDate()).slice(-2));
  }
```

The Javascript file then makes an AJAX call to the /getSQL route, which runs on app.py but makes use of a sql helper file to access the database.

```
@app.route("/getSQL", methods=["POST"])
def get_sql():
    content = request.json["data"]
    print(content)

    # parse
    metric = content["metric"]
    state_code = content["state_code"]
    min_date = content["min_date"]
    max_date = content["max_date"]
    df = sqlHelper.getDataFromDatabase(metric, state_code, min_date, max_date)
    return(jsonify(json.loads(df.to_json(orient="records"))))
```

This data is then put into the final query pictured below

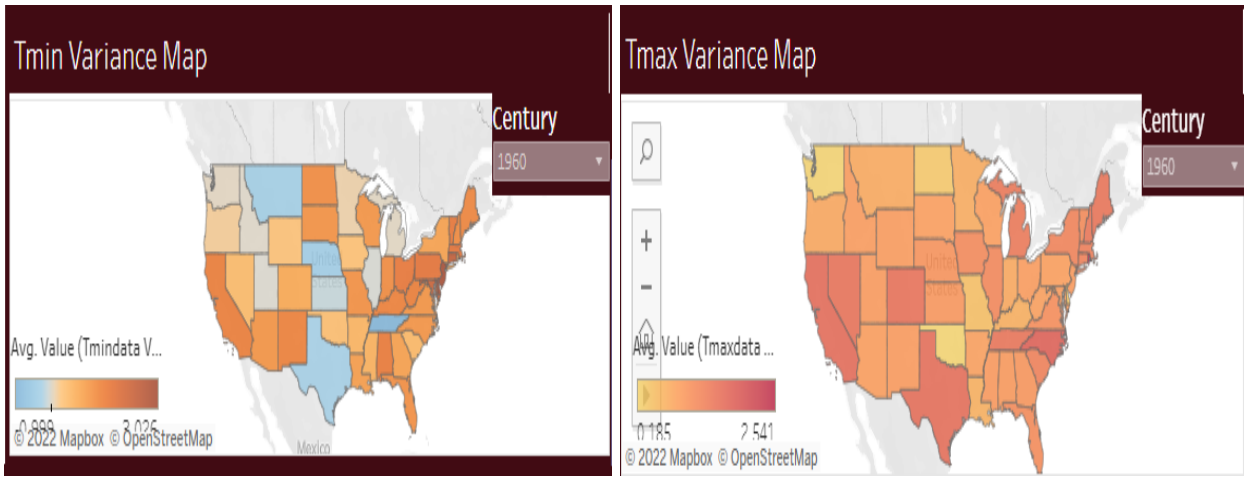
```
def getDataFromDatabase(self, metric, state_code, min_date, max_date):

    query = f"""
            SELECT
                state_name,
                date,
                value
            FROM
                {metric}
            WHERE
                state_code = '{state_code}'
                AND date >= '{min_date}'
                AND date <= '{max_date}'
                """
```

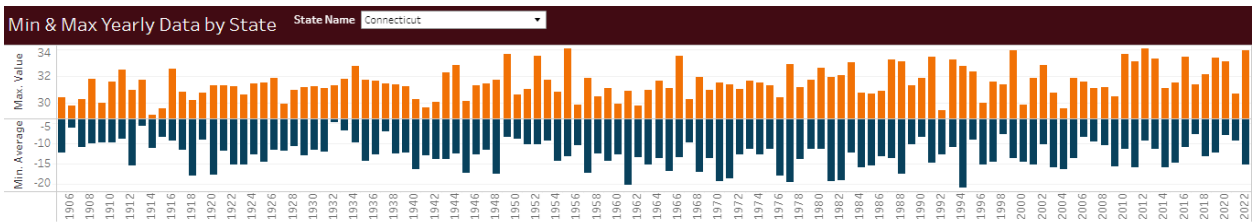
After the query is built the database is accessed and the results are returned and populate the empty table on the page.

Visualizations

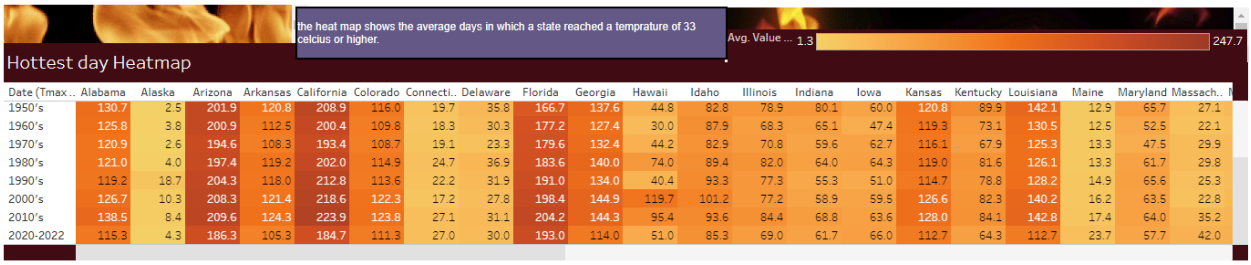
Using Tableau Public, we created two dashboards to better represent the change in temperatures, snow and rainfall in the US. The first dashboard focuses on temperature changes, maximum and minimum temperature by state. Our first visualization is a map of the US that shows the temperature and uses variability as a measure to display the change in temperature, both the high and lows recorded, from the selected centuries compared to our most recent century (2010-2020).



To drill down on the average maximum and minimum temperature per state, we also used a bar graph to represent the averages for every year.

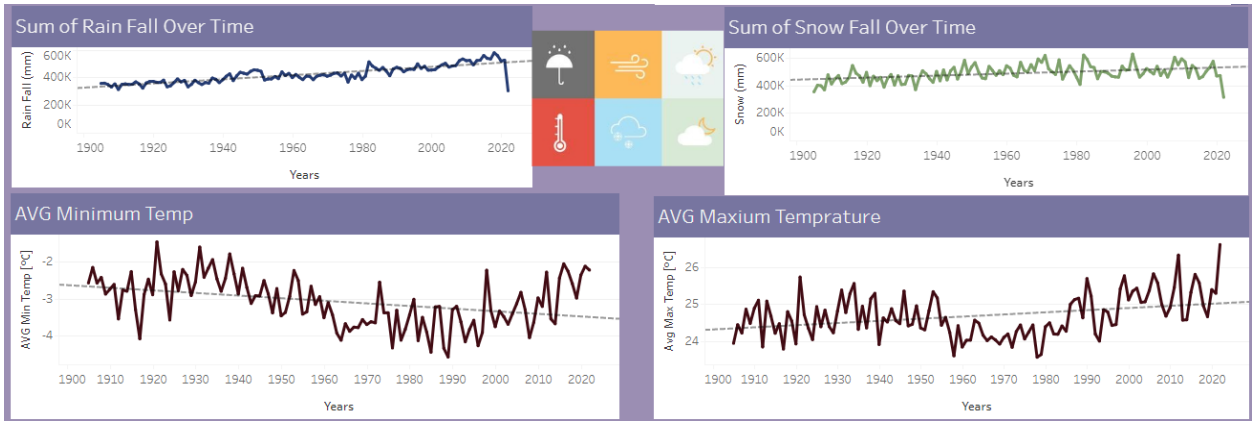


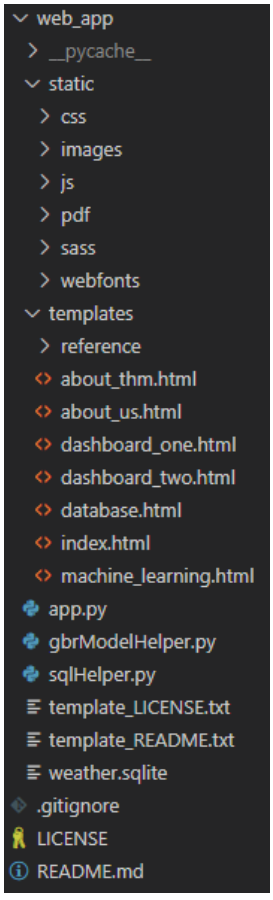
Lastly, to help show the story of temperature change, we created a heat map that displays the average number of days recorded where the temperature was above 33°C, grouped in centuries, for each state. We can see many states saw an increase in the amount of days each state is experiencing.



Date (Tmax..	Alabama	Alaska	Arizona	Arkansas	California	Colorado	Connecti..	Delaware	Florida	Georgia	Hawaii	Idaho	Illinois	Indiana	Iowa	Kansas	Kentucky	Louisiana	Maine	Maryland	Massach..
1950's	130.7	2.5	201.9	120.8	208.9	116.0	19.7	35.8	166.7	137.6	44.8	82.8	78.9	80.1	60.0	120.8	89.9	142.1	12.9	65.7	27.1
1960's	125.8	3.8	200.9	112.5	200.4	109.8	18.3	30.3	177.2	127.4	30.0	87.9	68.3	65.1	47.4	119.3	73.1	130.5	12.5	52.5	22.1
1970's	120.9	2.6	194.6	108.3	193.4	108.7	19.1	23.3	179.6	132.4	44.2	82.9	70.8	59.6	62.7	116.1	67.9	125.3	13.3	47.5	29.9
1980's	121.0	4.0	197.4	119.2	202.0	114.9	24.7	36.9	183.6	140.0	74.0	89.4	82.0	64.0	64.3	119.0	81.6	126.1	13.3	61.7	29.8
1990's	119.2	18.7	204.3	118.0	212.8	113.6	22.2	31.9	191.0	134.0	40.4	93.3	77.3	55.3	51.0	114.7	78.8	128.2	14.9	65.6	25.3
2000's	126.7	10.3	208.3	121.4	218.6	122.3	17.2	27.8	198.4	144.9	119.7	101.2	77.2	58.9	59.5	126.6	82.3	140.2	16.2	63.5	22.8
2010's	138.5	8.4	209.6	124.3	223.9	123.8	27.1	31.1	204.2	144.3	95.4	93.6	84.4	68.8	63.6	128.0	84.1	142.8	17.4	64.0	35.2
2020-2022	115.3	4.3	186.3	105.3	184.7	111.3	27.0	30.0	193.0	114.0	51.0	85.3	69.0	61.7	66.0	112.7	64.3	112.7	23.7	57.7	42.0

Dashboard 2 was more focused on the overall averages over time in the US rather than each state. We created line graphs for each of the temperature data we compiled, (max/ min temperature, snowfall and rainfall) included trend lines in each graph.





We are able to clearly see that we are seeing an increase in our max temperatures, snowfall, rainfall and our minimum temperatures are steadily getting colder as well.

Web Application

The foundation of the web application was a free template from html5up.net, https://html5up.net/solid-state. For the color scheme we decided to look through themes that were darker and more foreboding than others, looking to bring depth to our data storytelling and this template brought the correct atmosphere. It also had built in css for data tables and input forms which we would need for our database and machine learning pages.

After populating the template with all the required pages, we needed to convert the basic html site structure into a Flask app for Heroku development by placing all the web page html files into a template folder and all of the images and other files into the static folder. After some debugging and correcting links the Flask app was running with our

template. A few further cosmetic changes were made including adding photos to the main page and about page and the website was ready. The final file structure is pictured to the left.

While the template included Javascript files for page responsiveness and other features, we would need to add our own Javascript in order to integrate the database and machine learning pages. Both pages would also require python files to manage the database queries and machine learning calls, as well as new routes to be created in the app.py file.

The final version of the data would only have values for each state by year, but we deployed the database in the web app with the functionality of choosing specific months and even days. This would make adding such a database easier later and paired the database better with the machine learning app which can predict by day.

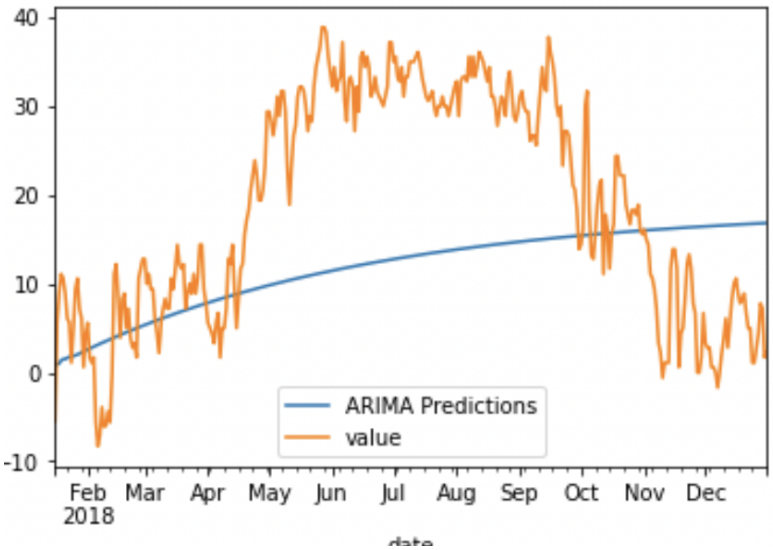
For deployment to Heroku a few things needed to happen, starting with making a copy of our web app and putting it in a separate Heroku deployment folder. By separating them we can be sure that any complications with the deployment could be easily addressed without compromising our main repository. We also needed to add a procfile, a heroku specific necessity, and a requirements file to ensure the libraries our app would need would be installed on the site. Getting the exact version of each library would prove to be very important as the machine learning algorithm would not run unless they were perfect.

Machine Learning

For machine learning the goal was being able to predict the future weather as accurately as possible. This required us to look more into regression algorithms vs classifier as this data is more time series related.

Initially we started with simple linear regression, though we saw an upward trend, it was not able to get to a right prediction value .

Next we looked at ARIMA as it was a good option for time series forecasting. But due to the seasonality of the data it couldn't build a good prediction output.



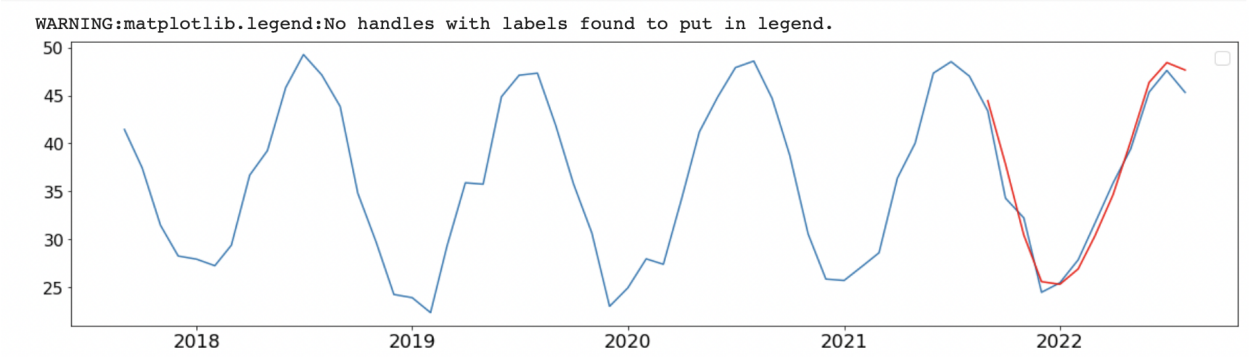
Code to this research is available here -
https://github.com/MoenJohn/capstone_climate_project/blob/main/model/arimatest.ipynb

We looked at tensor flow and realized the input cannot be the dates, but historical data points for the algorithm to understand the trends. Here we have to consider time series forecasting as a sequential machine learning regression problem, and the time series data is converted into a set of feature values and the corresponding target value. Since regression is a supervised learning problem, we need the target value, in which the lagged time series data becomes the feature values like this:

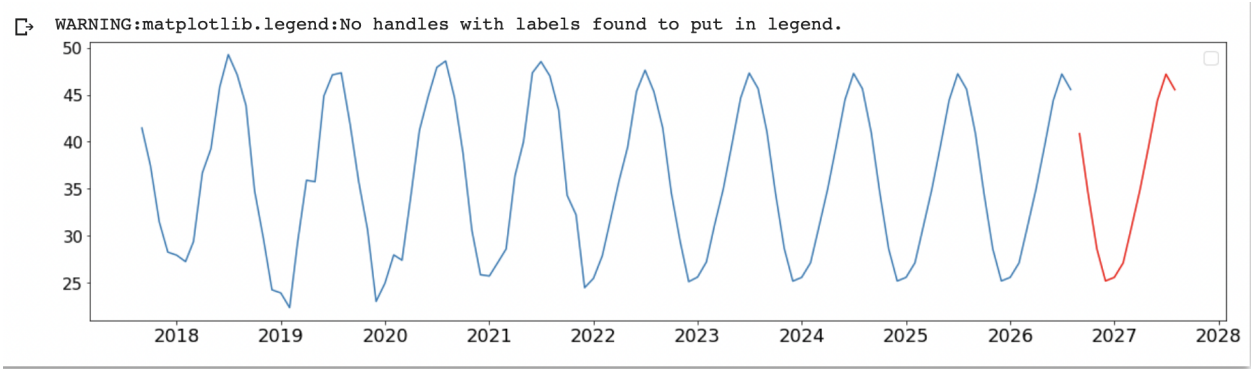
1	4	9	16	25
4	9	16	25	36
9	16	25	36	49

We will follow a window or buffer approach in which we would have to consider an appropriate window size. Then we would move the window from left to right of the sequence or the series data. We will consider the value immediately to the right of the window frame as target or the true value. So, each time-step we will shift or move the window so as to get a new row of features values and target value pairs. In this way we form the training data and training labels. In a similar way, we form the test and the validation dataset.

We did this exercise with LSTM in tensor flow and the algorithm was able to predict the required data points.



The prediction was good and we were able to predict far into the future too.



Code to this research is available here - https://github.com/MoenJohn/capstone_climate_project/blob/main/model/lstmnewtest.ipynb

One drawback to this approach was that the input required historic data points to predict we had to probably predict and keep the data for future days in a batch job. For example if prediction was needed for 2025 and batch size is 12, we need to predict from 2022 to 2023 to 2024 to 2025 and then pick the required date making it complex.

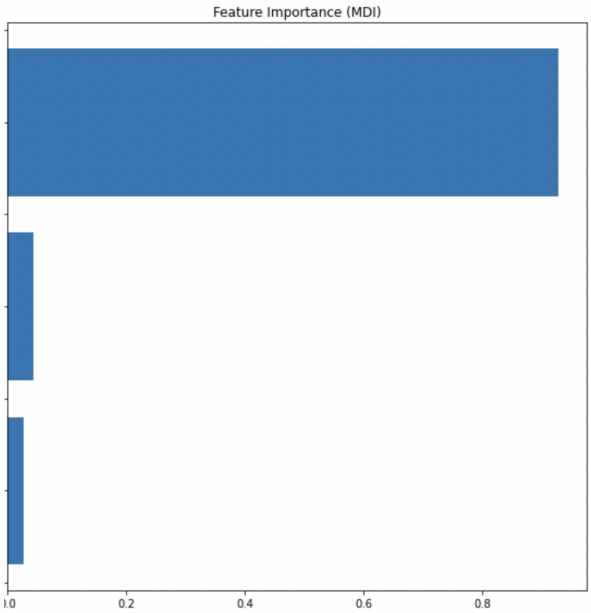
Work cited : https://github.com/gianfelton/12-Month-Forecast-With-LSTM/blob/master/12-Month%20Forecast%20With%20LSTM.ipynb

With this we came across gradient boosting regression. Gradient boosting refers to a class of ensemble machine learning algorithms that can be used for classification or regression predictive modeling problems. Ensembles are constructed from decision tree models. Trees are added one at a time to the ensemble and fit to correct the prediction errors made by prior models. This is a type of ensemble machine learning model referred to as boosting.

Models are fit using any arbitrary differentiable loss function and gradient descent optimization algorithm. This gives the technique its name, “gradient boosting,” as the loss gradient is minimized as the model is fit, much like a neural network.

Using Gradient Boosting for Regression we were able to split our input features into year, month and day. In the analysis by taking a specific state and metric we were able to predict both for past and future dates. Analysis Code here - GradientBoostingRegressorAnalysis.ipynb

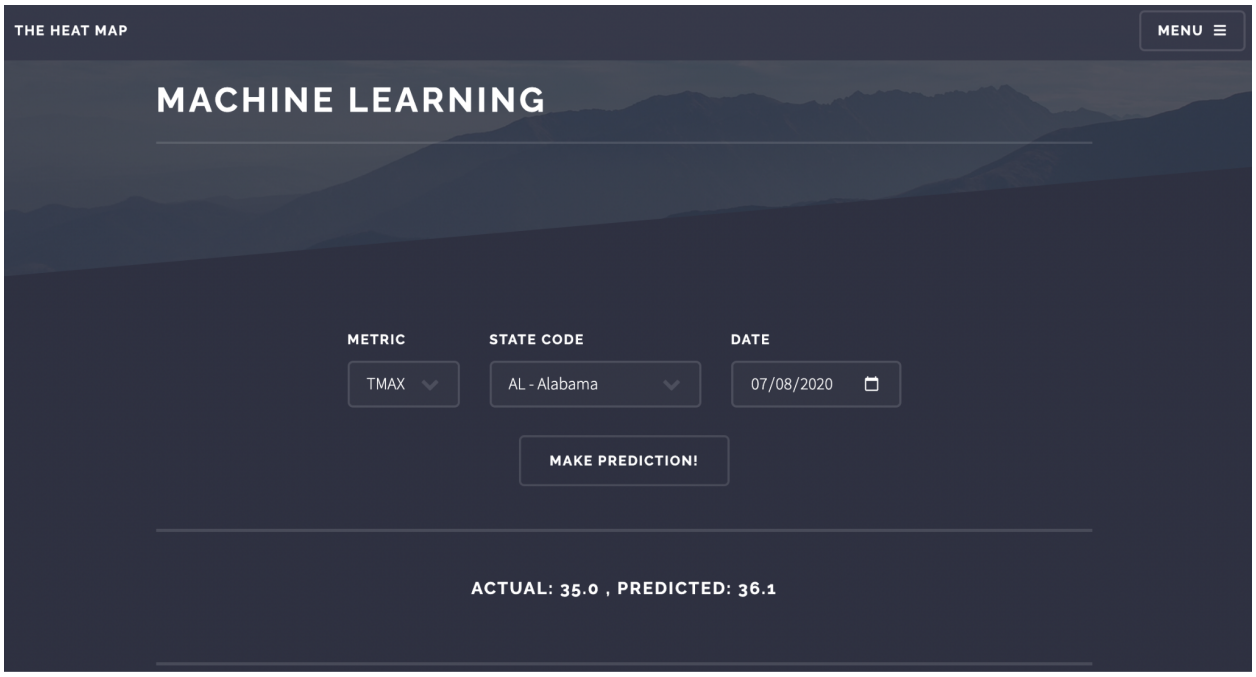
Month had the highest feature importance in this research.



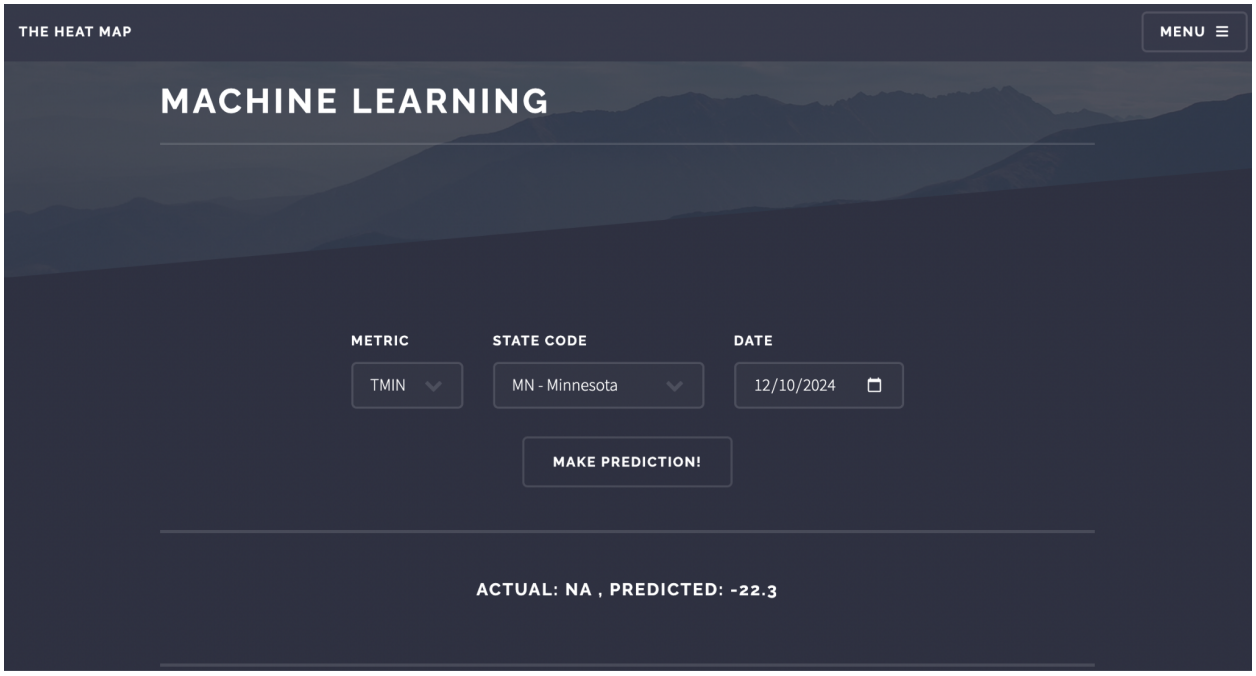
With this we built a loop to create a model and scalar for every state and metric in separate model objects. With this we were able to create a predicting process to use the required model taking the metric and state as input. Code here -

https://github.com/MoenJohn/capstone_climate_project/blob/main/web_app/gbrModelHelper.py

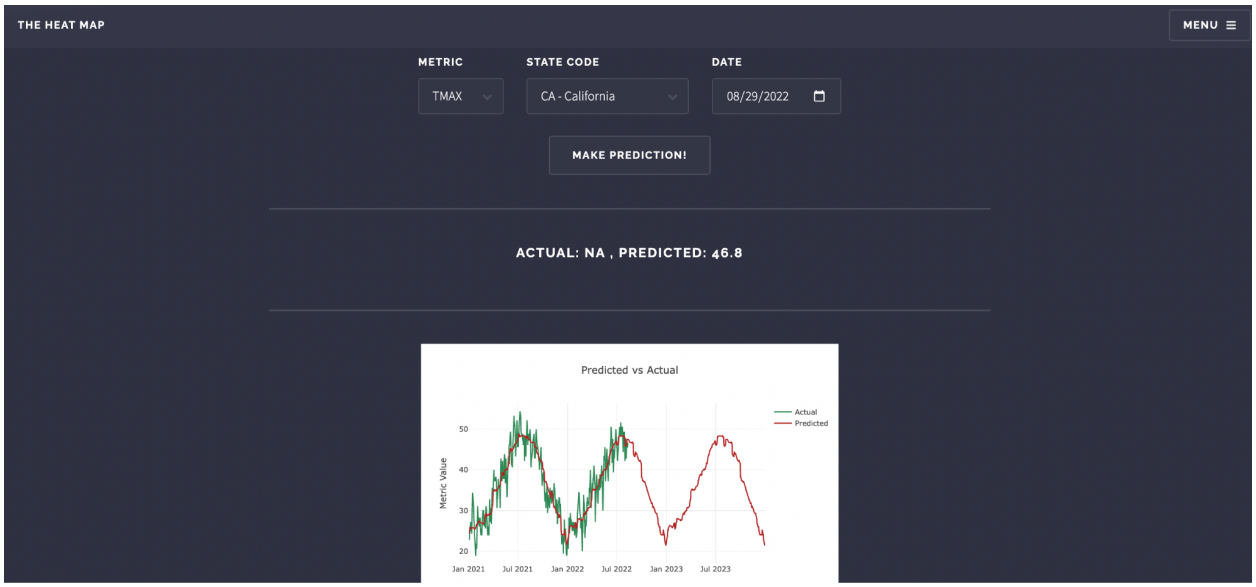
With this we embedded the feature to our webpage to enable prediction. Here is an example prediction for the past.



And below is an example for the future:



The machine learning page is also embedded with a ploty graph to view the trend from previous year to next year.



Work Cited:

SOURCE DATA
Metrics data by Station by Year- https://www.ncei.noaa.gov/pub/data/ghcn/daily/by_year/
Station data - https://www.ncei.noaa.gov/pub/data/ghcn/daily/ghcnd-stations.txt

DASHBOARD INSPIRATION
https://public.tableau.com/app/profile/matt.sorenson/viz/WeatherDataDashboard/DashboardSmaller

MACHINE LEARNING
Arima - https://machinelearningmastery.com/arima-for-time-series-forecasting-with-python/
Tensor Flow LSTM - https://github.com/gianfelton/12-Month-Forecast-With-LSTM/blob/master/12-Month%20Forecast%20With%20LSTM.ipynb
Gradient Boosting Regression - https://machinelearningmastery.com/gradient-boosting-machine-ensemble-in-python/

WEB APPLICATION
HTML Template - https://html5up.net/