Predicting Temperature and Precipitation in NYC

Alexander Wendelborn
David Freitag
Kimberly Yee Tan
Neetu Kachawa
Tshering Sherpa

CIS 9660 - Data Mining for Business Analytics
Final Project - Group #7
Professor Andrew Treadway
Baruch College - Spring 2021

Table of Contents

Problem Statement **4**

Background of Data Used for the Project **4**
- New York City Weather Dataset Components 4
- Multiple Locations 5

Issues Encountered and Cleaning Done on the NYC Weather Data Set **5**

Data Exploration and Patterns **6**

Predicting Weekly Temperature Using XGBoost from a Single Location **8**
- Analysis and Modeling Work Done on NYC Weather Data 8
- Modeling Approaches and Hyperparameter Tuning 8
- Possible Improvements if Given More Time 9
- Model Evaluation 10

Predicting Weekly Temperature Using XGBoost from Multiple Locations **12**
- Analysis and Modeling 12
- Feature Engineering 12
- Modeling Approaches and Tuning 13
- Improvements Given Additional Time 13
- Evaluating the Model 13

Predicting the Absence or Presence of Precipitation for a Given Day - Classification **14**
- Analysis and Modeling 14
- Issues Encountered in the Data/Cleaning 14
- Modeling Approaches and Tuning 14
- Improvements Given Additional Time 15
- Evaluating the Model 15

Forecasting Temperature with Facebook Prophet **16**
- Analysis and Modeling 16
- Issues Encountered in the Data/Cleaning 16
- Modeling Approaches and Tuning 16
- Improvements Given Additional Time 16
- Evaluating the Model 17

Predicting the Temperature on a Given Day - Linear Regression **18**
- Analysis and Modeling 18
- Issues Encountered in the Data/Cleaning 18
- Modeling Approaches and Tuning 18

Evaluating the Model 19

Improvements Given Additional Time 19

Predicting Weekly Precipitation Using Decision Tree Regression 20

Analysis and Modeling Work Done on NYC Weather Data 20

Evaluating the Model 20

Modeling Approaches and Tuning 20

Model Evaluation 22

Improvements Given Additional Time 22

Top Performing Models 23

Putting the Model Into Production 23

Appendix: Jupyter Notebook Files for Models 24

Jupyter Notebooks 24

Datasets 24

Data Cleaning Code 24

Data Dictionary 24

Problem Statement

We will attempt to predict the weather in NYC: temperature (predict value) and precipitation (predict raining/not raining) based on historical data.

Background of Data Used for the Project

Background of the data used for the project – where did it come from? What does each observation in the data represent? Why did your group choose this data?

The source of the New York weather dataset our group used for this project originates from the National Centers for Environmental Information (NOAA)'s weather database.

New York City Weather Dataset Components

The NYC weather dataset consists of data from the NY City Central Park station and Laguardia Airport station ranging from 05-02-2011 to 04-19-2021, where both have 100% coverage. We chose to use data from these two locations due to their comprehensive coverage and because we wanted to see if New York City data is sufficient to predict New York City weather.

The final dataset was formed from multiple datasets using two different methods of retrieval: API and ordering of datasets from NOAA's Local Climatological Database.

Using the API, I obtained daily weather data ("Daily Summaries") from Central Park and Laguardia Airport, consisting of the following features which pertain to precipitation levels, maximum and minimum temperature, wind direction, and wind speed (details in attached data dictionary):

"AWND(metres/sec)", "PRCP(cm)", "SNOW(cm)", "SNWD(cm)", "TMAX(celsius)", "TMIN(celsius)", "WDF2(degrees)", "WDF5(degrees)", "WSF2(metres/sec)", "WSF5(metres/sec)".

By manually ordering datasets from NOAA, I obtained hourly weather information from Central Park and Laguardia Airport that included variables not found in Daily Summaries. These variables included our label, DailyDryBulbTemperature(celsius) and DailyPrecipitation(cm), as well as other temperature-related, humidity, visibility and pressure-related variables.

After doing some data cleaning, the mean of the remaining variables deemed useful was obtained to be used as daily weather data. By merging this data frame with the data frame obtained through API on ["YEAR", "MONTH","DAY"], the final New York City weather dataset was formed. Each observation represents measurements of the various indicators for one day.

Multiple Locations

We also attempted to pull in data for multiple locations using the GHCND API. We downloaded minimum temperature, maximum temperature, and precipitation for New York, Buffalo, Houston, Miami, and Minneapolis. A number of data transformations were done to the dataset. We standardized the data to remove seasonality, lagged it, and averaged it. More information is included in the section titled: Predicting Weekly Temperature Using XGBoost from Multiple Locations.

Issues Encountered and Cleaning Done on the NYC Weather Data Set

The source data, datasets that were used to construct the final NYC weather data set, initially contained multiple columns filled with NAs. As much of the data was missing for these variables/columns, these variables would not contribute to helping the model predict with better accuracy. Hence, columns with too many missing data (more than 50% of data missing) were identified and removed. This percentage could have been higher. However, prior scanning of the source data informed us that mostly columns were either mostly filled with data or completely filled with NAs. Hence, 50 % was sufficient to weed out the unneeded columns/variables.

Another significant issue that was encountered when data cleaning was the conversion of all measurements to floats. Since we are running a regression analysis, all predictors have to be numerical. However, prior to data cleaning, most columns were of the object data type. It required some effort by referring to the data dictionary as well as using value_counts() to identify strings and alpha characters that prevented the conversion of the data to floats using the .map method. Using the .rstrip and map method, we were able to remove these characters or replace them with 0.

However, this was insufficient because this method resulted in a lot more NAs than in the original data, which we kept track through isna().sum() for each column before and after conversion. It took some time before we realized that the columns contained multiple data types and it was necessary to convert all data to string before employing the above measures.

Data Exploration and Patterns

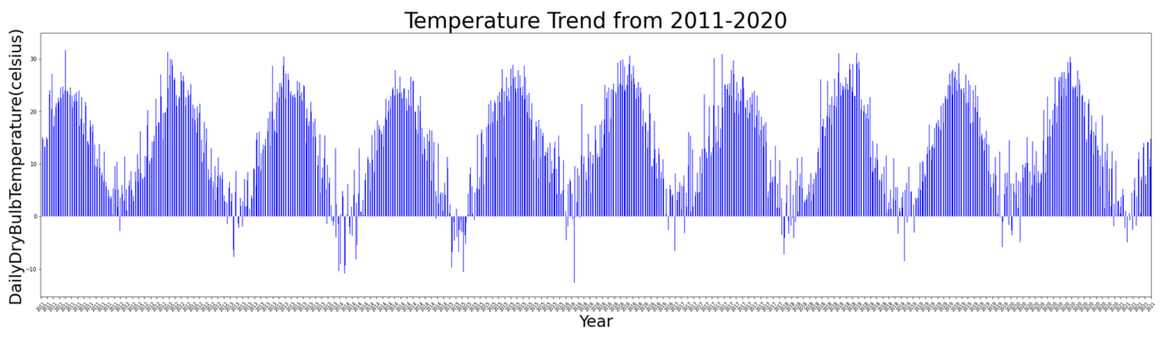
Data exploration was useful for deciding how to structure the features which were input into the model. Temperature data is highly seasonal, and the same time each year is usually similar. Also, there is some autoregressive behavior in the dataset where each day's temperature is related to the previous and following day's temperature.

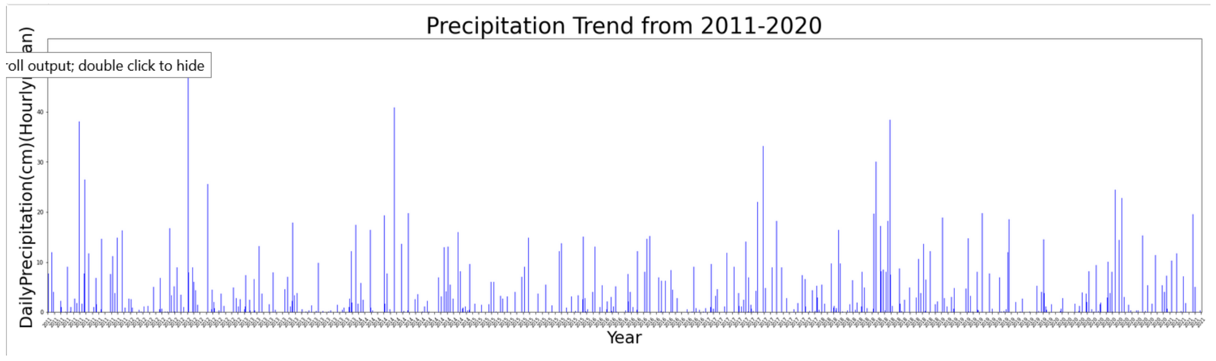
We explored the data by month to observe patterns in the number of days with precipitation, the total amount of precipitation, the dew point temperature (at what temperature water condenses, and wet bulb temperature (warmer if the humidity is higher). The pattern of the plots suggest that we would see more precipitation by volume in the summer months (June through August), but the number of days with precipitation is highest for February and December (with another high point in May). This suggests that it rains less often in the summer months than in the winter months, but it rains more in overall volume in the summer when there is rain at all.

When plotting the distribution of each predictors/variables, it was discovered that certain variables have a skewed distribution. These are used later to determine whether certain variables should be binned or square-rooted to improve the model's predictive ability.

Skewed variables include:
AWND (metres/sec), PRCP (cm), SNWD (cm), TMIN (celsius), WSF2 (metres/sec), WSF5 (metres/sec), DailyDewPointTemperature (celsius), DailyPrecipitation(cm)(Hourlymean), Daily Visibility, Daily Wind Speed (miles/hr).

We also tried plotting the y-variables separately to find any trend in the data. We noticed that there seems to be some annual seasonality for both target variables.





However, the annual temperature trend seems to be more consistent than precipitation, having seemingly identical temperatures per year. Hence, it may be easier to predict than precipitation.

Predicting Weekly Temperature Using XGBoost from a Single Location

By Kimberly Tan

Analysis and Modeling Work Done on NYC Weather Data

As a first step, the original New York City weather dataset was split to form the training and validation datasets with a 70/30 ratio.
Among the various types of models we have learned, we applied XGBoost Regression to the New York City weather dataset. It has been determined that we will use Root Mean Square Error as a metric for evaluating model performance. In order to decide what RMSE value would be acceptable to conclude whether a model performed well, the distribution of the label/y-variable (DailyDryBulbTemperature(celsius) and DailyPrecipitation(cm)(Hourlymean)) was obtained via .describe().

The following was discovered:

- DailyDryBulbTemperature(celsius) Training Data Set
    - Range of 45.367063
    - Range of 16.101009 between the 25th and 75th percentile
- DailyPrecipitation(cm)(Hourlymean)
    - Range of 47.394091
    - Range of 0.994437 between the 25th and 75th percentile

While trying to obtain an RMSE value that is as small as possible, an RMSE value within the range difference between the 25th and 75th percentile and is significantly smaller is ideal.

DailyDryBulbTemperature(celsius) and DailyPrecipitation(cm)(Hourlymean) were removed from the original dataset to obtain a dataset with only predictors, x_train. XGBoost Regression was then applied to the predictor dataset to obtain RMSE values to serve as a baseline for evaluating whether subsequent model performance is improving in prediction accuracy.

Modeling Approaches and Hyperparameter Tuning

XGBoosting Regression was tried to obtain continuous numerical temperature and precipitation predictions. To tune the XGB model, we used hyperparameter tuning using Randomized Search. Grid Search took too long.

The following parameters were used for Randomized Search on all XGBoost models tried:

```
parameters = {
        'max_depth': range(2, 6),
        'n_estimators': [50, 100, 150, 200, 250, 300],
        'subsample': [0.6, 0.7, 0.8],
        'colsample_bytree': [0.2, 0.4, 0.5, 0.6, 0.7, 0.8, 1],
        'colsample_bynode': [0.2, 0.4, 0.5, 0.6, 0.7, 0.8, 1],
        'gamma': [0, 5, 10, 15, 20],
        'learning_rate': [0.05, 0.1, 0.15, 0.2, 0.25, 0.3],
        'alpha': [10, 25, 50, 75, 100]
}
```

For all models, the following parameters were used:

```
clf = RandomizedSearchCV(xgb.XGBRegressor(eval_metric = 'rmse'), parameters, cv=5,
n_jobs=4, scoring = "neg_root_mean_squared_error", n_iter = 200)
```

Modeling was approached using several different predictor/feature sets:

1. The original dataset features minus the "DailyDryBulbTemperature(celsius)" and "DailyPrecipitation(cm)(Hourlymean)" labels
2. Reduced set of predictor features obtained by removing certain features with feature importance below a certain threshold (0.000301 for DryBulbTemperature and below 0.012940 for DailyPrecipitation)
3. Features in 2 and engineered features
4. Features in 3 and 1 week rolling average variable
5. Features in 4 and 1 day and 1 week lagged variable

Possible Improvements if Given More Time

If I were given more time, I would experiment with more window/period sizes for creating lagged and rolling average variables to see which window/period size works better in improving the models' ability to predict (if at all).

More trials for engineering new features and figuring out which variables should be kept or removed to optimize the models' ability to predict with what is available would potentially be helpful in getting better performance.

Additionally, more time to research best practices for predicting weather-related measurements could better inform me how to best approach this problem and/or optimize my models.

Model Evaluation

The RMSE values for the training and validation data set of each model was used to evaluate which models performed the best for predicting the Daily Dry Bulb Temperature and Daily Precipitation.

The RMSE values were evaluated in two ways:

1. The size of the RMSE value (the smaller the better due to smaller size of error)
2. The difference between the RMSE value of the training data set and the validation dataset (the smaller the difference, the better as there is less overfitting and better generalization ability to unseen data)

For DailyDryBulbTemperature(celsius), the XGBReggressor model with reduced features and the engineered features of TMIN_DPTemp_interaction, TMIN_low, TMIN_mid, TMIN_high seemed to perform the best. Although there are signs of overfitting and the model does not generalize as well to unseen data compared to some other models, the model achieved the lowest RMSE values for the training and validation data sets:

```
RMSE for Training Dataset: 0.3395105967407719
RMSE for Validation Dataset 0.4912868005687755
```

Compared to the wide range difference between the 25th and 75th percentile of the temperature variable's data, this RMSE value can be considered good. The model can be said to predict relatively well.

For DailyPrecipitation(cm), the XGBReggressor model with reduced features and the engineered features of TMIN_DPTemp_interaction, TMIN_low, TMIN_mid, TMIN_high and the rolling average of a 1 week period, seemed to perform the best. Although there are signs of overfitting and the model does not generalize extremely well to unseen data, the model achieved the lowest RMSE values for the training and validation data sets:

```
RMSE for Training Dataset: 1.984782809350188
RMSE for Validation Dataset 3.0630736064485493
```

It is also the model that generalizes to unseen data the best among all the models we tried. However, compared to the small range difference between the 25th and 75th percentile of the precipitation variable's data, the RMSE value is too high. Hence, the model is not suitable for production and does not predict well.

Predicting Weekly Temperature Using XGBoost from Multiple Locations

By Alexander Wendelborn

Analysis and Modeling

We wanted to see if we could predict the weekly average temperature in New York using historical weather data from a number of locations. We used meteorological data from five locations: New York, Buffalo, Houston, Miami, and Minneapolis. We used the minimum and maximum temperature as well as precipitation data for each day going back to 2007. The data was standardized to remove seasonality and lagged to get data in the past. After data cleaning and feature engineering, an XGBoost model was fitted to the data using a randomized grid search.

Feature Engineering

In order to clean and engineer the features, a few data transformations were done. First, the maximum temperature and minimum temperature were added together and divided by 2 to get the average daily temperature. A seven-day rolling average was taken to get weekly temperatures. Second, the precipitation data was converted into dummy variables to get 1's and 0's representing whether or not there was precipitation on that day. The precipitation was then summed on a weekly basis to get a value between 0 and 7. The data was then separated into the training and validation datasets. Next, the average and standard deviation of the same week each year were taken from the training subset. To account for the seasonality, the dataset was standardized using the same time each year averages and standard deviations. This resulted in features which showed the variance from the mean for that particular week in the year. To capture the autoregressive component, lagged data for each week was added as a separate column. This resulted in features which were essentially the temperature and precipitation into the past for each of the locations. The lags went back one year. We combined the older lags by averaging them together. The independent variables were standardized weekly temperatures and precipitation for each location going back weekly lags of up to 1 year. Additionally, we did a series of averages for data going back further than one week.

Modeling Approaches and Tuning

In order to tune the XGBoost model, the RandomizedSearchCV function was utilized. We tried a wide variety of different parameters to try and find the optimal subset. We focused on tuning max_depth, n_estimators, gamma, learning_rate, and alpha.

Improvements Given Additional Time

Given additional time, a number of improvements could be made. We could have included more locations as independent variables and used more years of data. We could have tried a more granular time scale (daily). Also, we could have included various other variables from a number of different sources such as CO_2 emissions, solar luminosity, etc.

Evaluating the Model

We wanted to minimize the prediction mean squared error. The mean squared error of this model was found for both training and validation datasets. In order to get the temperature predictions, the data output by the model had to be un-standardized. Using the average temperature as the prediction for that day in the year, the RMSE was 2.55 degrees C for the training set and 2.68 degrees C for the validation set. The model prediction root mean squared error on the training dataset was 1.99 degrees C. The model prediction root mean squared error on the validation dataset was 2.36 degrees C. This model modestly increased the prediction accuracy over the average temperature. The most important variables in the model for predicting the weekly temperature in New York ended up being the weekly temperatures lagged one week in Minneapolis and Buffalo. Based on our limited knowledge of meteorology, this makes sense. Cold fronts tend to move west to east from these locations and weather there one week ago would affect weather in New York.

Predicting the Absence or Presence of Precipitation for a Given Day - Classification

By David Freitag

Analysis and Modeling

To produce this model, we included a number of relevant data points gathered from NOAA including metrics like temperature, humidity, pressure, dew point temperature, visibility, and wind speed. Daily precipitation ('PRCP') is our target variable. After importing the data, the next step is to lag the data by one day. The purpose of this is to create a new row (or multiple rows), where the data points represent yesterday's weather observations (or last week's observations, or last month's, etc.). Then, we predict the absence or presence of precipitation using yesterday's weather observations. This effectively allows us to predict whether or not it will rain tomorrow (or next week, etc.) given today's weather. We tried a logistic regression model and a decision tree model to make this prediction.

Issues Encountered in the Data/Cleaning

For this task, the primary work of dealing with this data came from creating the time lags effectively for the dataset. By nature of creating a lag in the data, the values for the first row become null (because the values of every row represent yesterday's weather observations). When dealing with lagged data, it is key to remove those rows with null values, as they are not helpful for prediction.

Modeling Approaches and Tuning

After creating the lagged dataset, we applied a logistic regression model with L1 the regularization parameter ('C') set to 1.0, indicating the default level of regularization strength. This number was arrived at after testing a few different values of C and using the AUC score of the validation set as the evaluation metric. For the decision tree model, we used a parameter grid and performed a grid search to find the optimal set of parameters.

Improvements Given Additional Time

Given additional time, we could further tune the tree model with a larger parameter grid, and we could try to implement a random forest instead of a single tree. We could also test adding multiple lag periods into the same dataset. We could also try adding multiple lags to the dataset, instead of just a single set of lagged data, for example, 7-day lagged data and 1-day lagged data instead of just 1-day lagged data.

Evaluating the Model

To evaluate the models, we calculated the AUC score for each. Since the goal is to identify true positives and true negatives (predict rain on days when it rained, predict no rain on days when it did not rain), the AUC score accounts for this. We also produced a confusion matrix to show at a granular level true positives/false positives and true negatives/false negatives. It is important to note that the effectiveness of both models decreased significantly when trying to predict values farther into the future (i.e. using a time lag of 7 or 14 days compared to a time lag of 1 day). As a result, the models included in the notebook are only set for a lag of 1 day.

Forecasting Temperature with Facebook Prophet

By David Freitag

Analysis and Modeling

For this model, we used Facebook's Prophet library, which can be used to produce forecasts based on historical data relatively quickly. As an input, Prophet takes a dataframe with two columns: ds (date) and y (value). Using this historical data, Prophet can generate a forecast for an upcoming "future period" along with an uncertainty interval.

Issues Encountered in the Data/Cleaning

To prepare the data from Prophet to work with, we needed to take our main weather dataset and generate a date column that could be interpreted by Prophet from the year, month, and day fields. Additionally, the dry bulb temperature field was selected for the 'y' column in the prepared dataframe since this is the target variable we are seeking to predict.

Modeling Approaches and Tuning

After providing Prophet with the properly formatted dataframe, we ran the model with the default parameters and generated a Plotly visualization of the results. To tune the model, we ran a grid search on a grid of parameters--this revealed that the optimal parameters for the model (as calculated by what produces the lowest RMSE) are changepoint_prior_scale = 0.01, and seasonality_prior_scale = 0.01. The first parameter, changepoint_prior_scale, refers to the number of changepoints (i.e. how often does the identified trend change), with a smaller value signifying fewer changepoints. The second parameter, seasonality_prior_scale, refers to the strength of the seasonality, with smaller values reducing the effect of the identified seasonality on the prediction.

Improvements Given Additional Time

Given additional time, we could improve the model by tuning hyperparameters on a larger grid and by adding additional regressors to the model. Adding additional regressors could help the model to generate a more accurate forecast, since other attributes of the weather (amount of precipitation, wind speed, etc.) may have an effect on the temperature.

Evaluating the Model

To evaluate the model, we performed cross validation on a series of 8 forecasts and calculated the mean RMSE based on these forecasts. The result was an RMSE of 3.969, which underperforms compared to some of our other models. However, with additional regressors and feature engineering for these regressors, it is possible we could improve the model's performance.

Predicting the Temperature on a Given Day - Linear Regression

By Tshering Sherpa

Analysis and Modeling

We used the lasso and ridge regression model to predict the temperature. Initially, we trained the model using data gathered from NOAA to document the model performance before engineering any new features. Our y variable was the “daily dry bulb temperature” and our x variables were all the remaining columns such as snow, precipitation, wind, sea level pressure and humidity. For our lasso and ridge regression model, we tried a number of feature engineering such as calculating the weekly lag data based on the “daily dry bulb temperature”, where the data points represent previous week’s temperature. The weekly lag data created a number of new features which we used as our x variables to train our models. We also calculated the weekly rolling average of the “daily dry bulb temperature”, in addition to which we included some features from the initial dataset that had high coefficient value when running our first lasso regression model. Lastly, we added new features to the original dataset from NOAA, by transforming some variables using log, square root, one hot encoding and calculating ratios using two related variables.

Issues Encountered in the Data/Cleaning

While creating lagged data for our feature engineering some null values were also created. We dealt with the null values by dropping the rows that contained null values.

Modeling Approaches and Tuning

For our lasso regression model (L1 penalty) we performed k-fold cross validation by setting the k value equal to 5. We did not set the alpha parameter for lasso, therefore the model tried a random number of alpha values to find the optimal value. Whereas, for the ridge regression model (L2 penalty) we also performed a k-fold cross validation by setting the k value equal to 5 but set the alpha parameter to 0.0001, 0.001, .01, .05, 0.1, 0.2, 0.3, 0.4, 0.5, 0.75, 1.

Evaluating the Model

We used the mean squared error (MSE) to evaluate the performance of our model. We wanted a baseline metric score to compare the performance of the model, hence we ran the lasso and ridge regression model on the NOAA data before applying any feature engineering. The lasso model resulted in a score of 0.3905 on the training set and 0.3877 on the validation set. And the ridge regression model had an MSE score of 0.3797 on the training set and 0.3828 on the validation set. Because we had a baseline metric score, we found out that our weekly lag data and rolling average data did not improve the performance of the model and therefore, we tried additional feature engineering such as the log transformation, one hot encoding, square root, etc.

Improvements Given Additional Time

Because our weekly lag data and rolling average did not perform better than our baseline model, we would like to try additional feature engineering to train the model. We would like to try a combination of the rolling average data and some of the new features created using the log transformation, square root, ratios and one hot encoding. This may be helpful because the rolling average is created using the "daily dry bulb temperature" and may add useful information to the transformed data that saw improvement in performance compared to our initial model.

Predicting Weekly Precipitation Using Decision Tree Regression

By Neetu Kachawa

Analysis and Modeling Work Done on NYC Weather Data

For this model, we used Decision Tree Regression to predict weekly precipitation. We used the class notes and the scikit-learn org site to understand how to train a Decision Tree Regressor model and learn more about the parameters and the metric evaluations.

As a first step, we imported a few libraries to run this model. We then split the original New York City weather dataset (finalproject_dataset.csv) in the training and validation datasets with a 70/30 ratio. We identified 2 labels: 'DailyPrecipitation(cm)(Hourlymean)' and 'DailyDryBulbTemperature(celsius)' but for this model, we will be using only the 'DailyPrecipitation(cm)(Hourlymean)' label.

Evaluating the Model

We decided to use Root Mean Square Error (RMSE) as a metric for evaluating model performance. To determine the ideal RMSE value, we saw the distribution of the y-label using the describe() function.

- DailyPrecipitation(cm)(Hourlymean)
  - Range of 51.987337
  - Range of 1.069474 between the 25th and 75th percentile

Based on this observation, we tried to obtain a smaller RMSE value. Since the range between the 25th and 75th percentile is significantly smaller.

Modeling Approaches and Tuning

We trained an initial DecisionTreeRegressor by removing the labels from the original dataset and obtained a dataset (x_train) to know the RMSE value which will serve as a baseline for evaluating the model performance.

```
RMSE for Training Dataset: 1.1146195109936006
RMSE for Validation Dataset 1.6165658828773297
```

This is a good (smaller) RMSE value with very less overfitting.

Decision Tree Regression was trained to obtain the optimal parameters. We used GridSearchCV to tune the hyperparameters of our model.

Following parameters were used on all the models that we trained:

```
parameters = {
        "max_depth":range(2, 10),
        "min_samples_leaf": range(5, 55, 5),
        "min_samples_split": range(5, 110, 5)
                }
```

```
regp1 = GridSearchCV(DecisionTreeRegressor(), parameters, n_jobs=4, scoring =
"neg_root_mean_squared_error")
```

This time we received increased RMSE values compared to the model without hyperparameter tuning.

```
RMSE for Training Dataset: 1.2042479780737443
RMSE for Validation Dataset 1.6593106910108815
```

We then performed feature reduction and feature engineering to see if our model improves.

Feature reduction:

```
RMSE for Training Dataset: 2.8849144103783653
RMSE for Validation Dataset 3.989389264965367
```

Feature engineering:

```
RMSE for Training Dataset: 3.4861706739514995
RMSE for Validation Dataset 4.050208201796241
```

The models have shown an increase in the RMSE values instead.

We performed additional feature engineering on the same dataset by adding a rolling average for 1 week and observed our model perform better with very less to no overfitting.

Feature engineering 2:

RMSE for Training Dataset: 2.8329157117527726
RMSE for Validation Dataset 2.9600148495378855

An additional feature engineering was performed by adding a lagged variable for 1 day and 1 week, but the model has not improved.

Feature engineering 3:

RMSE for Training Dataset: 3.521813221375177
RMSE for Validation Dataset 3.911680043854125

Model Evaluation

We used the RMSE values for the training and validation data set of each model to evaluate which models performed the best for predicting the Daily Precipitation.

The RMSE values were evaluated in two ways :

- RMSE value size - The smaller the size, the better value
- Difference between the RMSE values of Training and Validation set - The smaller the difference, signifies less overfitting and better generalization to unseen data

Out of all the models performed (with hyperparameter tuning, feature reduction, feature engineering) to predict the Daily precipitation, we selected the model with feature engineering 2, where we reduced certain features and added a rolling average for one week.

RMSE for Training Dataset: 2.8329157117527726
RMSE for Validation Dataset 2.9600148495378855

The RMSE values are smaller compared to other models and the difference between the RMSE for Training and Validation datasets is very small.

Improvements Given Additional Time

Given more time, we would experiment with additional period sizes for rolling average variables to see which period size works better in improving the models' ability to predict Precipitation. We would also experiment with more trials for engineering new features and discover what variables should be kept or removed to optimize the models' ability to perform better with given data.

In addition, we would research best practices and preferred modelling techniques for weather-related predictions. Additional research would help in approaching and experimenting different ways to optimize the models or creating new models to give us better predictions.

Top Performing Models

To predict the temperature on a given day, we used different models such as Lasso regression model, XGBoost model on NYC weather data, and XGBoost model on weather data from multiple locations. For each model, the RMSE value was calculated to compare the models and we found that our lasso regression model performed the best out of the three models. Specifically for lasso regression, we engineered new features using log and square root transformation, one hot encoding, and calculated ratios of related variables, to train the model. This resulted in the lasso model outputting a RMSE value of 0.3428 on the training set and 0.3503 on the validation set.

To predict the daily precipitation, we performed 2 models on NYC weather data: XGBoost and Decision Tree Regression. For each model, the RMSE value was calculated to compare the model performance and we observed that the Decision tree regression model with certain feature engineering performed best out of all the models to predict the daily precipitation. We removed a few features from the dataset and engineered new features to obtain the lower RMSE of 2.83 on the training dataset and an RMSE value of 2.96 on the validation set. Feature reduction involved removal of Precipitation values, Snow, wind and sea level pressure variables based on the feature importance. Feature engineering was performed on the skewed variables by binning them, few of these variables involved Minimum Temperature, Snow depth, and Daily Visibility. This did not really improve the model performance so we further performed feature engineering by adding a rolling average of 1 week and observed the model performed much better than all other models. We can further experiment by adding the rolling average for varying time periods.

Putting the Model Into Production

To implement our model in production, we would need to make some changes to how our model is run and how it returns results.

For the model to be available to the public (or to whomever we want to access it), we would need to take the code for the model and encapsulate it in a runnable Python script that would be hosted on a cloud server. This script would take in the weather data as an input, and output our predictions as the output. Because weather data is updated daily (or sometimes sooner), we would include code that would run every night to pull in yesterday's new data, and then rerun the model to generate predictions given the expanded dataset. Additionally, to make the predictions available publicly, we would provide a HTTP API endpoint that could be accessed by users to retrieve a particular prediction. This could be done using the Python Flask library.

Using the API endpoint, we could then export our predictions through a data pipeline to either generate a daily report that could be provided to a user (for example, through email), or we could connect the data to a business intelligence dashboard where the predictions could be visualized or incorporated with other metrics.

Appendix: Jupyter Notebook Files for Models

Jupyter Notebooks

- Logistic Regression - Tree Model - Rain Tomorrow Classification.ipynb
- Prophet Forecasting Model.ipynb
- Lasso and Ridge Regression - predicting temperature.ipynb
- XGBoost Weekly Lagged Data with multiple locations.ipynb
- XGBoosting - NYC Dataset.ipynb
- Decisiontree_Regression_Precipitation.ipynb

Datasets

- finalproject_dataset.csv - main NYC weather dataset
    - Central_park_data.csv - used for creating finalproject_dataset.csv
    - Central_park_data2.csv - used for creating finalproject_dataset.csv
    - Laguardia_data.csv - used for creating finalproject_dataset.csv
- Alex_Weather_Data.csv - weather data including cities outside NYC

Data Cleaning Code

- Kimberly Data Cleaning Code.ipynb
- Extracting Weather Data Function.ipynb

Data Dictionary

- Data Dictionary 1.pdf
- Data Dictionary 2.pdf