IEOR Final Project Report

Jayden Zheng, Xiaolu Liu, Anyi Chomtin, Jiahui Yu, Renzhi Wang, and Carolyn Shan

I. Motivation & Impact

Air travel in the San Francisco Bay Area is a vital mode of transportation, yet the unpredictability of flight delays often disrupts travel plans. These delays cause inconvenience, as travelers are unsure whether a flight will be delayed and the duration of the delay, making it challenging to plan effectively. As college students, coordinating arrivals and departures when traveling together can be vital for ensuring safety and increasing productivity. Motivated by the desire to address this uncertainty, we designed "Flight Delay Prophet," a machine learning model designed to predict potential delays for Bay Area travelers. By allowing travelers to foresee delays from the comfort of home, we aim to empower them to make informed decisions and better manage their travel experiences – minimizing disruptions and maximizing time. While the focus of this project is on the Bay Area specifically, we expect that it has the potential to be adapted for other regions in future. Additionally, by integrating "Flight Delay Prophet" with complementary models, such as those predicting airport commute times, we can offer a comprehensive suite of tools for travelers seeking efficient travel management solutions.

II. Data

We integrated two datasets to construct our predictive model. The first dataset, sourced from Kaggle[1], includes United States domestic flight delay data spanning January 2019 to June 2019. This dataset provides detailed information about each flight, such as temporal data, aircraft details, and distance traveled. Our rationale for integrating this dataset is that by analyzing and transforming these features, we can gain valuable insights that will help us predict flight delays. The second dataset comprises daily weather indicators for each airport and was obtained from the National Centers for Environmental Information.[2] Specifically, we focused on three ZIP codes (94128, 94621, 95110)[3] corresponding to the airport areas in the San Francisco Bay Area, covering the entirety of the year 2019. These ZIP codes are exclusively tied to airport areas, underscoring their relevance to our analysis.

During the data cleaning and preprocessing phase, we executed the following tasks:

I. Selected flights originating from or destined for airports within the San Francisco Bay Area.
II. Standardized all dates to a uniform format and sorted the date column in ascending order.
III. Refined airline arrival and departure times for consistent formatting by implementing leading zero padding, ensuring all times are in a consistent four-digit format. Utilized f-strings, validated times to ensure hours and minutes fall within valid ranges, and converted time strings to datetime objects for easier manipulation when extracting specific data ranges.
IV. Incorporated six new features: ['Date', 'Airline', 'DepTime', 'DepHour', 'ArrDelay', 'DepDelay']. Removed null values, created a new dictionary to make the days of the week more readable.
V. Integrated the weather dataset with the flight delay dataset, predicated on airport locations. We pruned columns with substantial missing data and extracted city names and airport codes to streamline the merging process.

---

[1] Flight Delay and Causes Dataset
[2] National Centers for Environmental Information
[3] United States Zip Codes

VI. Implemented two new columns: 'SameDayFlightsperPlane' to track the number of flights a specific plane is scheduled for in a single day, and 'PreviousFlightDelay' to record the delay of the most recent flight that the same plane experienced earlier on the same day.

VII. Used conditional filtering to create a new dataframe where the origin of a flight is one of the three airports in the San Francisco Bay Area: SFO, SJC, or OAK.

III. Analytics Models

Variance Inflation Factor (VIF)

Our analysis examined multicollinearity using the Variance Inflation Factor (VIF) technique. Initially, we computed VIF values for each independent variable in our dataset, revealing significant multicollinearity issues, as evidenced by infinite VIF values for certain variables. We used an iterative approach to address this problem and systematically removed features with high VIF values, including "Plane_Same_Day_Most_Recent_Flight_TotalDelay" and "Avg_ArrDelay_Before". After each removal iteration, we recalculated VIF values to ensure multicollinearity mitigation. Through this process, we successfully reduced multicollinearity and stabilized our regression models. This meticulous VIF analysis improved the reliability and accuracy of our analytical outcomes.

OLS Linear Regression Modeling

We employed linear regression analysis to model the relationship between departure delays and the explanatory features. Initially, we selected a subset of features deemed relevant for predicting departure delays, excluding columns such as 'DayOfWeek_Monday,' 'Airline_Delta Air Lines Inc.,' 'Origin_OAK,' 'DepHour_0', and 'Month_1'. The model was fitted using Ordinary Least Squares (OLS) regression to minimize the sum of squared residuals between the observed and predicted departure delays. The resulting model exhibited an R-squared value of 0.254, indicating that the selected features could explain approximately 25.4% of the variance in departure delays. The adjusted R-squared value, which considers the number of predictors in the model, was 0.250. Additionally, the F-statistic of 73.51 with a corresponding p-value close to zero suggested that the overall regression model was statistically significant. However, some coefficients associated with individual features had p-values greater than the conventional threshold of 0.05, indicating insignificance. There were some multicollinearity issues observed as indicated by the large condition number of 1.21e+05, suggesting the presence of strong correlations among predictors. Further analysis revealed that certain features, such as 'Plane_Same_Day_Most_Recent_Flight_DepDelay,' 'Avg_DepDelay_Before,' and 'Flights_Before,' exhibited notable coefficients, suggesting their importance in predicting departure delays. Conversely, features like 'Average_Wind_Speed,' 'Precipitation,' and 'Maximum_Temperature' showed relatively weaker associations with departure delays.

Regression Tree Modeling

For a more exploratory approach to predicting departure delays, we employed a Decision Tree Regressor. Through grid search with cross-validation, we determined the optimal complexity parameter (ccp_alpha) for controlling model complexity. The best ccp_alpha value was found to be 0.0099, leading to the optimal decision tree estimator selection. The decision tree model was then visualized to interpret the splits and decision rules, providing insights into the predictive factors influencing departure delays. Upon evaluating the model's performance on the validation dataset, the decision tree exhibited an Out of Sample

R-squared (OSR2) value of -0.3616. Despite this negative OSR2 value, indicating that the model performs worse than a horizontal line, it's worth noting that decision trees are not evaluated solely based on R-squared. Additionally, the Mean Absolute Error (MAE) between the predicted and actual departure delays was 37.1050. While the decision tree model offered insights into departure delay patterns, it may require further refinement to improve its predictive accuracy. The negative OSR2 value suggests that the model may not be suitable for accurately predicting departure delays using the original feature data. Further investigations into feature selection, model tuning, or alternative algorithms may be necessary to build a more robust predictive model.

Logistic Regression Approach

Given the challenges of predicting the exact duration of delays via regression, we refocused our efforts on a classification problem: predicting whether a flight would encounter a significant delay (defined as greater than one hour). This categorical prediction can still offer valuable guidance for travelers. Even without knowing the exact delay length, they can be aware of the need to be timely. We developed a logistic regression model to predict significant flight delays. The target variable, "Significant_DepDelay," was defined as a binary indicator of whether a flight delay would exceed 60 minutes, with "1" indicating significant delays and "0" indicating delays shorter than 60 minutes. Features used in the logistic regression model were consistent with those in the linear regression analysis, ensuring continuity in capturing various flight and weather characteristics. To optimize the model, we employed L1-regularization, which helps identify the most relevant features while penalizing those with less predictive power. This approach reduced the impact of multicollinearity and enhanced model interpretability by focusing on the most impactful variables. Evaluation of the logistic regression model using the validation dataset yielded an accuracy of 83.56%, a precision of 83.14%, a recall of 84.32%, and an F1 score of 83.73%. The confusion matrix indicated a balanced classification of flights across both significant and non-significant delay categories, suggesting reliable predictive performance.

Classification Tree Approach

A decision tree classifier was then used to explore patterns and relationships within the data further. GridSearchCV, combined with cross-validation, was employed to fine-tune the complexity parameter (ccp_alpha), which controls model complexity. We effectively reduced overfitting by selecting the optimal ccp_alpha while maintaining predictive strength. Visualizing the decision tree allowed us to identify key decision rules and gain insights into the feature splits, ultimately providing greater model interpretability. The decision tree classifier was tested on the validation dataset and exhibited an accuracy of 78.91%, a precision of 77.02%, a recall of 81.05%, and an F1 score of 78.98%. The confusion matrix highlighted the distribution of correct and incorrect classifications, revealing the importance of specific features in predicting significant flight delays. The decision tree model's real-world performance was tested on an independent test dataset. The model achieved an accuracy of 77.62%, a precision of 76.23%, a recall of 78.45%, and an F1 score of 77.33%. This shows the model could identify significant delays with reasonable accuracy while maintaining interpretability. Despite the slightly lower scores compared to the validation dataset, the model proved useful in offering travelers accurate predictions to minimize travel disruptions.

Random Forest Approach

To deepen our understanding of the data's predictive aspects, we leveraged a Random Forest approach to forecast flight delay durations.A plot between ccp_alpha and CV R2 indicates that the optimal value of ccp_alpha is around 0.00092. This suggests that some mild pruning improves the model's generalization performance compared to no pruning (ccp_alpha = 0) or extremely aggressive pruning (larger ccp_alpha values). A second plot revealed an increasing trend in CV R2 as the "max_features" value increases from around 2.5 to around 15, after which the curve levels off slightly. The optimal value for the "max_features" parameter is around 15-17, which yields a cross-validated R-squared score of approximately 0.96 (specifically, cross-validated R2 = 0.95995 and out-of-sample R2 = 0.96608). Overall, the results indicate that setting max_features to around 15-17 strikes a good balance between capturing relevant information from the features and avoiding overfitting or unnecessary complexity in the Random Forest model for this particular dataset and problem.

Neural Network Approach
As a challenge we created a neural network model because of its strength in learning complex patterns and relationships in data especially with high dimensionality and non-linearities. Using TensorFlow and Keras, we trained a neural network over 50 epochs, utilizing the input layer of preprocessed data and hidden layers for weighted calculations to observe its performance on the validation set. After training, the model had a MAE of 26.4977 indicating that on average the model’s predictions deviate from the true departure delay values by approximately 26.5 minutes on the validation set.

IV. Findings & Further Scope

Our analysis found that certain features, such as 'Plane_Same_Day_Most_Recent_Flight_DepDelay,' 'Avg_DepDelay_Before,' and 'Flights_Before,' significantly influenced departure delay predictions, while weather conditions had weaker associations. The decision tree models we created provided clarity by visualizing decision rules but had lower predictive performance than other models. The logistic regression model performed well in classifying significant flight delays (>60 minutes), with an accuracy of 83.56%, precision of 83.14%, recall of 84.32%, and F1 score of 83.73%. The Random Forest model, with optimal hyperparameters (ccp_alpha = 0.00092, max_features = 15-17), demonstrated strong predictive power with a cross-validated R-squared score of approximately 0.96. The neural network model, trained over 50 epochs, achieved a validation MAE of 26.4977, indicating an average deviation of approximately 26.5 minutes from true departure delay values.

While our project initially focused on flight delays in the San Francisco Bay Area, the model's versatility allows for expansion to other regions in the United States by leveraging the customizable nature of our approach with Zip Code data. We should explore techniques that integrate the strengths of various models and provide a more comprehensive evaluation of flight delays. It is important to develop a user-friendly interface or application that displays real-time flight delay information. The UIUX implementation would be crucial for the success of our "Flight Delay Prophet" and could assist travelers in navigating air travel seamlessly.

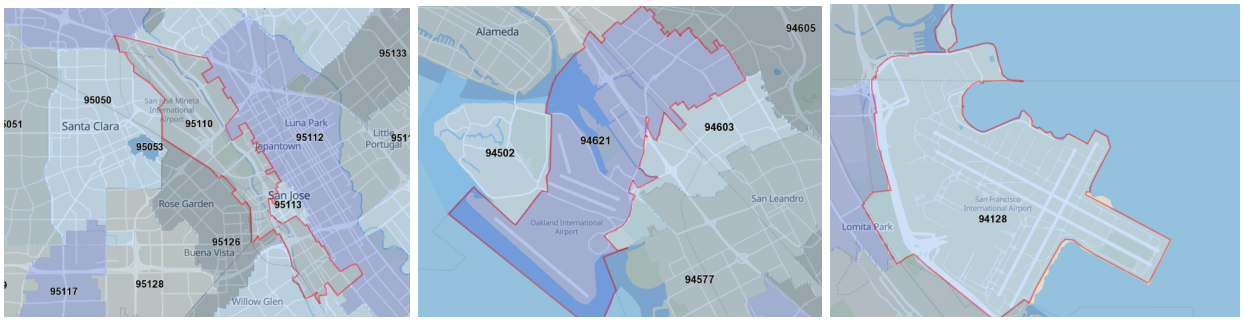
Google Collab [Link]

3 Google Notebooks at included inside this folder:

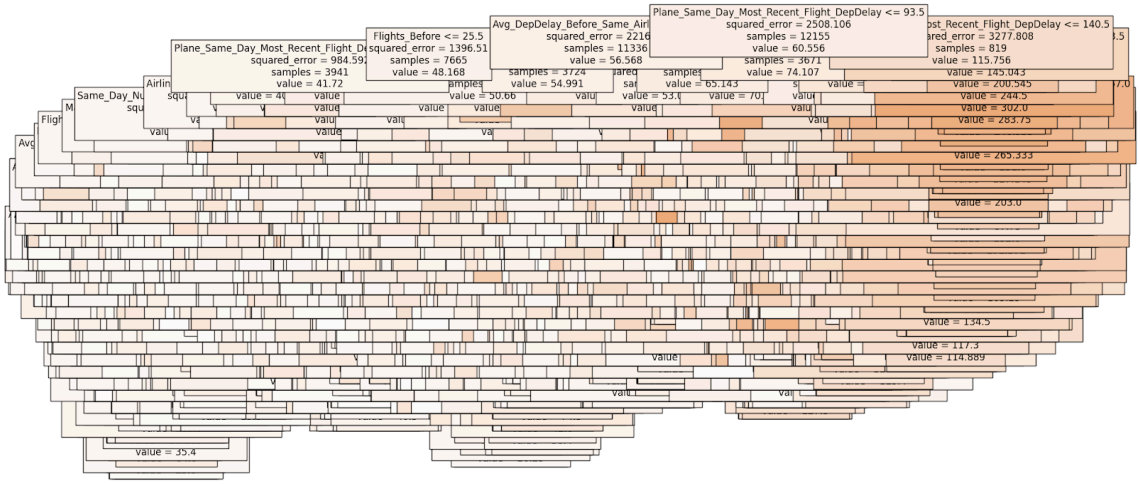
1) EDA: Dataset Stitching, Preprocessing, Cleaning
2) Modeling: Linear, Logistic, Regression Tree, Classification Tree, Random Forest
3) Neural Network: Challenge Model

Appendix

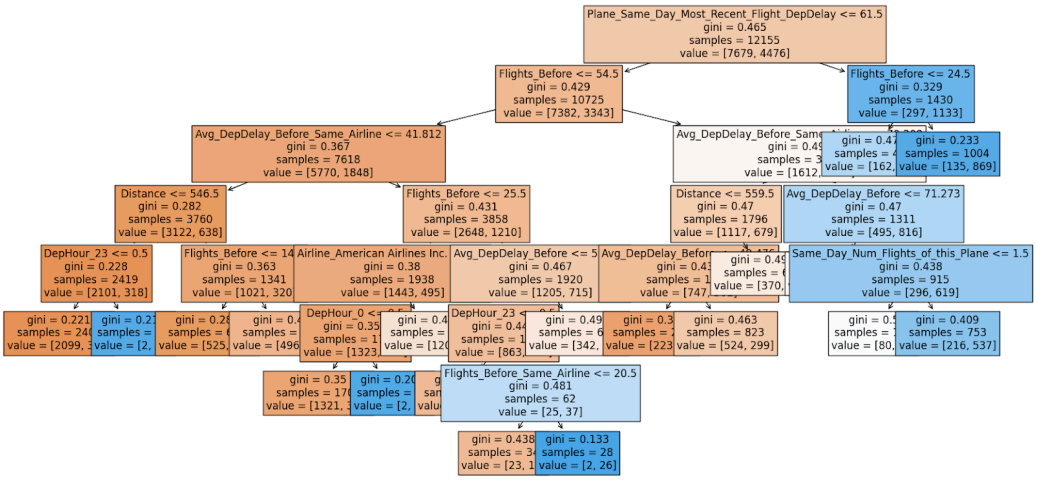
Appendix 1: Zip Codes for SFO, SJC, and OAK Airports



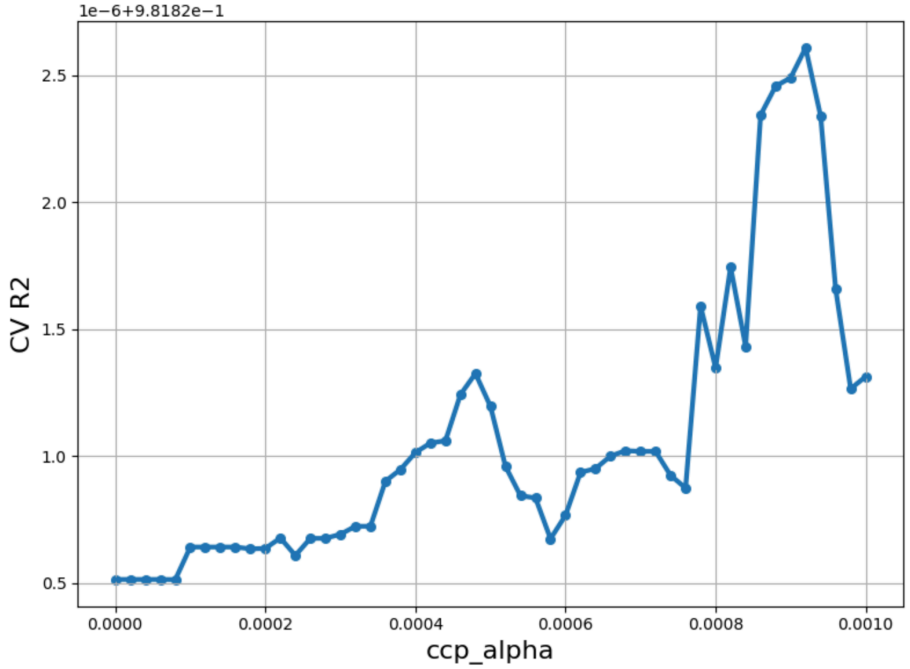
Appendix 2: Decision Tree Regressor



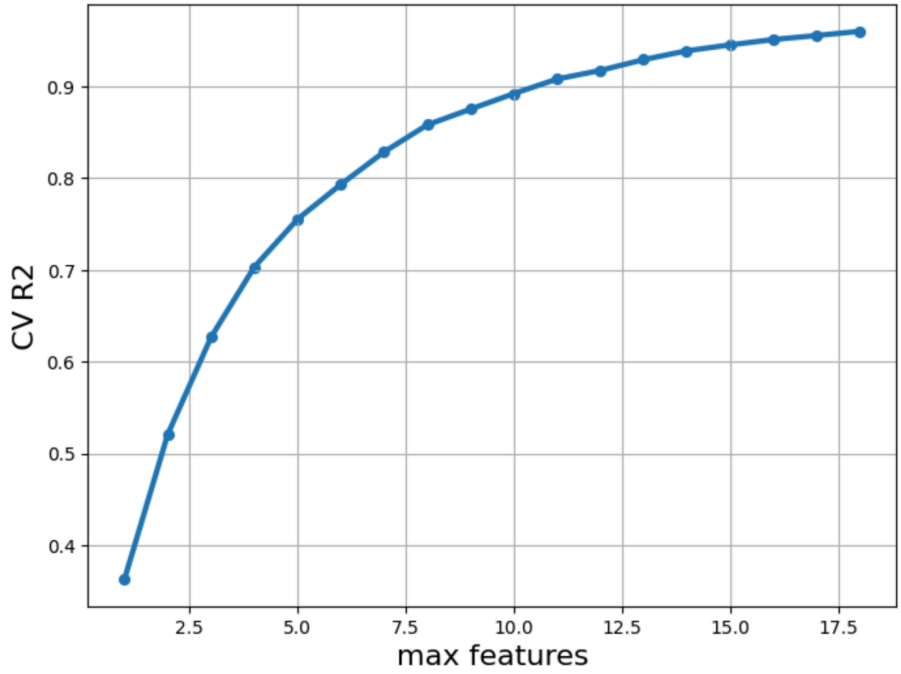
Appendix 3: Decision Tree Classifier



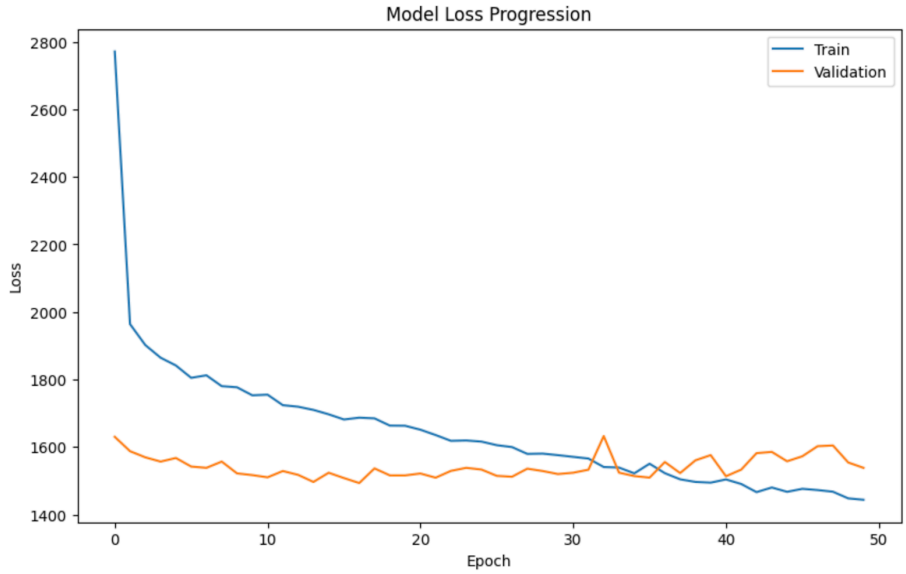
Appendix 4: Random Forest Plot 1



Appendix 5: Random Forest Plot 2



Appendix 6: Neural Network Model Loss Progression



Project Motivation

As a student living in the San Francisco Bay Area, traveling by air is a popular choice. However, there is one thing annoying about it, which is flight delays. We don't know whether there will be a delay, how long the delay might be, and we have no idea when the notification of delay will be sent out, and it can be a nightmare when we are in a cab to the airport and receive a text message saying that "your flight XXXXXX has delayed 3 hours, and we apologize for any inconvenience". Yes, that can be really incovenient if you have to spent 3 more hours waiting in the lounge. No seriously, that can actually feel like 8 hours.

Therefore, our group is interested in building a flight delay prophet, which can help travellers in the bay area to predict delays even before they leave home, so that they might be able to have a better understanding of the possibility of a delay for their flight and how bad that might be, and therefore have a wiser plan and use of their time instead of getting stuck in the excess waiting in the airport. We believe this project might be valuable since it can save travellers much time, and that can be considerably impactful if those saved time accumulated in each day. If our model is proven to provide some great insights, it might also be inspiring for other areas or even globally to research related features. Moreover, it can also be connected with models on other related areas, such as those predicting commuting time to the airport, which might provide travellers with more precise time management.

Let's get started.

Initialization

[ ] ↳ *4 cells hidden*

Importing Datasets

[ ] ↳ *4 cells hidden*

Data Pre-processing

Flight Delay Data - Data Cleaning

This section focuses on identifying flights originating from or destined for airports in the San Francisco Bay Area.

```
# This line creates a list named 'bay_area_airports' containing the three-letter cod
bay_area_airports = ['SFO', 'OAK', 'SJC']

# This line filters the 'flight_delay' DataFrame to include only flights either orig
#       - Flights where the 'Origin' column value is in the 'bay_area_airports' list
#       - Flights where the 'Dest' column value is in the 'bay_area_airports' list
bay_area_flights = flight_delay[(flight_delay['Origin'].isin(bay_area_airports)) |
                                (flight_delay['Dest'].isin(bay_area_airports))]

print(bay_area_flights.shape)
print(bay_area_flights.head())
```

```
(50660, 29)
    DayOfWeek        Date  DepTime  ArrTime  CRSArrTime UniqueCarrier  \
77          4  03-01-2019     1944     2110        1915            WN
78          4  03-01-2019     1251     1425        1345            WN
79          4  03-01-2019     1548     1728        1635            WN
80          4  03-01-2019     2216     2348        2255            WN
81          4  03-01-2019     2010     2203        2030            WN

                  Airline  FlightNum TailNum  ActualElapsedTime  ...  TaxiIn  \
77  Southwest Airlines Co.        223  N223WN                 86  ...       5
78  Southwest Airlines Co.        237  N281WN                 94  ...       3
79  Southwest Airlines Co.        280  N460WN                100  ...       4
80  Southwest Airlines Co.        900  N679AA                 92  ...       4
81  Southwest Airlines Co.        962  N378SW                113  ...      28

    TaxiOut  Cancelled  CancellationCode Diverted CarrierDelay WeatherDelay  \
77        9          0                N        0           21            0
78       15          0                N        0            2            0
79       18          0                N        0           14            0
80       10          0                N        0           11            0
81        8          0                N        0           45            0

   NASDelay  SecurityDelay  LateAircraftDelay
77        0              0                 94
78        0              0                 38
79       10              0                 29
80        2              0                 40
81       23              0                 25

[5 rows x 29 columns]
```

This section focuses on cleaning and organizing the `bay_area_flights` DataFrame, specifically the `'Date'` column.

1. **Date Format Validation:**

```
bay_area_flights[bay_area_flights['Date'].str.match(r'^\d{2}-\d{2}-\d{4}$')]
```

- This line checks the format of the `'Date'` column in the `bay_area_flights` DataFrame.
- It utilizes the `str.match` function with a regular expression (`r'^\d{2}-\d{2}-\d{4}$'`) to ensure the dates follow a specific format:
  - `^`: Matches the beginning of the string.
  - `\d{2}`: Matches two digits, representing the day of the month.
  - `-`: Matches a hyphen separator.
  - Repeated twice more for month and year, ensuring a consistent DD-MM-YYYY format.

2. **Converting Dates to DateTime:**

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

- Assuming the date format validation is successful, we proceed to convert the `'Date'` column from strings to datetime objects.
- The `pd.to_datetime` function is used for this purpose.
- This conversion allows for more advanced date and time manipulation and analysis later in the process.

3. **Sorting by Date:**

```
bay_area_flights.sort_values(by='Date')
```

- Finally, we sort the `bay_area_flights` DataFrame by the `'Date'` column in ascending order.
- Sorting by date ensures a chronological order, which is beneficial for further analysis or visualization tasks.

```
bay_area_flights = bay_area_flights[bay_area_flights['Date'].str.match(r'^\d{2}-\d{2
bay_area_flights['Date'] = pd.to_datetime(bay_area_flights['Date'], format='%d-%m-%Y
bay_area_flights = bay_area_flights.sort_values(by='Date')
print(bay_area_flights.shape)
bay_area_flights.tail()
```

```
(50660, 29)
```

	DayOfWeek	Date	DepTime	ArrTime	CRSArrTime	UniqueCarrier	Airline	Fli
434819	1	2019-06-30	1155	1243	1157	OO	Skywest Airlines Inc.	
434818	1	2019-06-30	1257	1351	1317	OO	Skywest Airlines Inc.	
434817	1	2019-06-30	1722	1856	1836	OO	Skywest Airlines Inc.	
434827	1	2019-06-30	1440	1547	1407	OO	Skywest Airlines Inc.	
434752	1	2019-06-30	1004	1123	1022	OO	Skywest Airlines Inc.	

5 rows × 29 columns

```
bay_area_flights['DepTime'].sort_values().head()
```

```
118918    1
394856    1
119795    1
209387    1
19279     1
Name: DepTime, dtype: int64
```

Cleaning Departure and Arrival Times

This section addresses the `'DepTime'` (departure time) and `'ArrTime'` (arrival time) columns in the `bay_area_flights` DataFrame. At the end we'll ensure a clean and standardized format for departure and arrival times, facilitating further analysis.

Workflow:

1. **Leading Zero Padding:**
   - Ensures a consistent four-digit format (e.g., "0915") for all times using `f-strings`.
2. **Time Validation:**
   - Checks if hours (`< 24`) and minutes (`< 60`) fall within valid ranges.
   - Invalid entries are replaced with `"0000"` for identification.

3. **Conversion to Time Objects:**
    - Converts time strings to datetime time objects using `pd.to_datetime` with format `'%H%M'`.
    - Handles potential errors (`'coerce'`) by converting them to `NaT` (Not a Time).
    - Extracts the time portion using `.dt.time` for dedicated time data.

```
bay_area_flights['DepTime'] = bay_area_flights['DepTime'].apply(lambda x: f"{int(x):
bay_area_flights['DepTime'] = bay_area_flights['DepTime'].apply(lambda x: x if int(x
bay_area_flights['DepTime'] = pd.to_datetime(bay_area_flights['DepTime'], format='%H

bay_area_flights['ArrTime'] = bay_area_flights['ArrTime'].apply(lambda x: f"{int(x):
bay_area_flights['ArrTime'] = bay_area_flights['ArrTime'].apply(lambda x: x if int(x
bay_area_flights['ArrTime'] = pd.to_datetime(bay_area_flights['ArrTime'], format='%H

print(bay_area_flights[['Date', 'Airline', 'DepTime', 'ArrTime', 'ArrDelay', 'DepDel
```

```
                Date                 Airline   DepTime   ArrTime  ArrDelay  \
    83187 2019-01-01         JetBlue Airways  18:56:00  22:33:00        23
    38707 2019-01-01   United Air Lines Inc.  23:47:00  01:11:00       237
    45314 2019-01-01  Southwest Airlines Co.  19:02:00  21:06:00        31
    45377 2019-01-01  Southwest Airlines Co.  15:48:00  17:10:00        35
    45388 2019-01-01  Southwest Airlines Co.  21:52:00  23:15:00        35

           DepDelay
    83187        31
    38707       253
    45314        42
    45377        43
    45388        47
```

```
bay_area_flights_w_new_features = bay_area_flights.copy()
```

Now, let's start adding some features of our interest based on the columns we have. First, let's extract the hour in the day that each flight is at:

```
def extract_hour(dep_time):
    return dep_time.hour
bay_area_flights_w_new_features['DepHour'] = bay_area_flights_w_new_features['DepTim
print(bay_area_flights_w_new_features.shape)
bay_area_flights_w_new_features[['Date', 'Airline', 'DepTime', 'DepHour', 'ArrDelay'
```

```
(50660, 30)
```

	Date	Airline	DepTime	DepHour	ArrDelay	DepDelay
83187	2019-01-01	JetBlue Airways	18:56:00	18	23	31
38707	2019-01-01	United Air Lines Inc.	23:47:00	23	237	253
45314	2019-01-01	Southwest Airlines Co.	19:02:00	19	31	42
45377	2019-01-01	Southwest Airlines Co.	15:48:00	15	35	43
45388	2019-01-01	Southwest Airlines Co.	21:52:00	21	35	47

Based on our observation, there seems to be a lot of 0 values for the cancellation and diverted columns, let's check it and remove them if that is the case:

```
print(bay_area_flights_w_new_features['Cancelled'].mean())
print(bay_area_flights_w_new_features['CancellationCode'].unique())
print(bay_area_flights_w_new_features['Diverted'].mean())
```

```
0.0
['N']
0.0
```

```
bay_area_flights_w_new_features = bay_area_flights_w_new_features.drop(columns=['Can
```

Transform the day of week column to make it more readable:

```
day_of_week_mapping = {
    1: 'Monday',
    2: 'Tuesday',
    3: 'Wednesday',
    4: 'Thursday',
    5: 'Friday',
    6: 'Saturday',
    7: 'Sunday'
}
```

```
bay_area_flights_w_new_features['DayOfWeek'] = bay_area_flights_w_new_features['DayO
bay_area_flights_w_new_features['Month'] = bay_area_flights_w_new_features['Date'].d
print(bay_area_flights_w_new_features[['Date', 'Month', 'DayOfWeek', 'Airline', 'Dep
```

```
             Date  Month DayOfWeek                 Airline   DepTime  ArrDelay  \
83187 2019-01-01      1   Tuesday         JetBlue Airways  18:56:00        23
38707 2019-01-01      1   Tuesday   United Air Lines Inc.  23:47:00       237
45314 2019-01-01      1   Tuesday  Southwest Airlines Co.  19:02:00        31
45377 2019-01-01      1   Tuesday  Southwest Airlines Co.  15:48:00        35
45388 2019-01-01      1   Tuesday  Southwest Airlines Co.  21:52:00        35
```

```
       DepDelay
83187        31
38707       253
45314        42
45377        43
45388        47
```

Exploring Potential Delay Dependencies

While we've cleaned and organized the flight data, further analysis can be conducted to identify potential factors influencing delays. One possibility is the delay of a plane impacting its subsequent flights on the same day.

To investigate this, we can sort the `bay_area_flights` DataFrame by the following criteria:

1. **Date:** This ensures a chronological order, allowing us to track a specific plane's flights throughout the day.
2. **Origin:** Grouping by origin helps identify flights originating from the same airport, potentially involving the same plane.
3. **Departure Time:** Sorting by departure time within each origin group facilitates tracing the sequence of a plane's flights.

By implementing this multi-level sorting, we can analyze potential correlations between a flight's delay and the delay of the same plane's earlier flight on the same day.

```
bay_area_flights_w_new_features.sort_values(by=['Date', 'Origin', 'DepTime'], inplac
print(bay_area_flights_w_new_features.shape)
print(bay_area_flights_w_new_features[['Date', 'Origin', 'DepTime', 'ArrDelay', 'Dep
```

```
(50660, 28)
            Date Origin   DepTime  ArrDelay  DepDelay
85842 2019-01-01    ATL  14:52:00        23         7
85917 2019-01-01    ATL  19:31:00        59        51
72263 2019-01-01    AUS  16:23:00        62        78
17326 2019-01-01    BOI  14:47:00        54        64
17313 2019-01-01    BOI  15:30:00        15         7
```

```
bay_area_flights_w_new_features.head()
```

	DayOfWeek	Date	DepTime	ArrTime	CRSArrTime	UniqueCarrier	Airline	Flig
85842	Tuesday	2019-01-01	14:52:00	17:21:00	1658	DL	Delta Air Lines Inc.	
85917	Tuesday	2019-01-01	19:31:00	22:03:00	2104	DL	Delta Air Lines Inc.	
72263	Tuesday	2019-01-01	16:23:00	17:52:00	1650	AA	American Airlines Inc.	
17326	Tuesday	2019-01-01	14:47:00	15:24:00	1430	OO	Skywest Airlines Inc.	
17313	Tuesday	2019-01-01	15:30:00	16:26:00	1611	OO	Skywest Airlines Inc.	

5 rows × 28 columns

Creating Feature: Same Day Flights per Plane

This section introduces a new feature, `"Same_Day_Num_Flights_of_this_Plane"`, to the `bay_area_flights` DataFrame. This feature aims to capture the number of flights a particular plane has on the same day.

Workflow:

1. **Group by Date and TailNum:**
   - We can utilize the `groupby` function on the `'Date'` and `'TailNum'` columns.
   - This groups the DataFrame by flights occurring on the same date and involving the same plane (identified by the `"TailNum"`).
2. **Count Flights per Group:**
   - Within each group, we can calculate the number of flights for that specific plane on that day.
   - The `size` attribute applied to the group object provides this count.
3. **Adding the Feature:**
   - After iterating through the groups and calculating flight counts, we can assign a new column named `"Same_Day_Num_Flights_of_this_Plane"` to the DataFrame.

- This column will contain the count values for each flight, indicating the number of flights for that plane on the same day.

```
bay_area_flights_w_new_features['Date'] = pd.to_datetime(bay_area_flights_w_new_feat
bay_area_flights_w_new_features['Same_Day_Num_Flights_of_this_Plane'] = bay_area_fli
print(bay_area_flights_w_new_features.head())
```

```
       DayOfWeek       Date   DepTime   ArrTime  CRSArrTime UniqueCarrier  \
85842   Tuesday 2019-01-01  14:52:00  17:21:00        1658            DL
85917   Tuesday 2019-01-01  19:31:00  22:03:00        2104            DL
72263   Tuesday 2019-01-01  16:23:00  17:52:00        1650            AA
17326   Tuesday 2019-01-01  14:47:00  15:24:00        1430            OO
17313   Tuesday 2019-01-01  15:30:00  16:26:00        1611            OO

                    Airline  FlightNum TailNum  ActualElapsedTime  ...  \
85842    Delta Air Lines Inc.        629  N648DL                329  ...
85917    Delta Air Lines Inc.       1155  N3762Y                332  ...
72263  American Airlines Inc.       1023  N589AA                209  ...
17326   Skywest Airlines Inc.       6383  N732SK                 97  ...
17313   Skywest Airlines Inc.       6338  N923SW                116  ...

       TaxiIn  TaxiOut  CarrierDelay  WeatherDelay NASDelay SecurityDelay  \
85842       6       20             7             0       16             0
85917       6       17             0             0        8             0
72263       2       12            25             0        0             0
17326       4        8             0             0        0             0
17313       7       19             0             0        0             0

      LateAircraftDelay DepHour  Month  Same_Day_Num_Flights_of_this_Plane
85842                 0      14      1                                   1
85917                51      19      1                                   2
72263                37      16      1                                   2
17326                54      14      1                                   3
17313                15      15      1                                   2

[5 rows x 29 columns]
```

```
bay_area_flights_w_new_features.to_csv('/content/drive/My Drive/ieor 142 final proje
```

Flight Delay Data - Adding Features

Feature Engineering: Previous Flight Delay

This section introduces a new feature, `"Previous_Flight_Delay"`, to the `bay_area_flights` DataFrame. This feature aims to capture the delay of the most recent flight the same plane had on the same day, potentially influencing the current flight's delay.

Workflow:

1. **Sorting by Date, Origin, and Departure Time:**
    - As discussed earlier, we sort the DataFrame by `'Date'`, `'Origin'`, and `'DepTime'`.
    - This ensures a chronological order within each plane's flights on a specific day.
2. **Iterating with `shift`:**
    - We can leverage the `shift` function while iterating through the DataFrame.
    - The `shift` function allows us to access values in previous rows based on a specified number of positions.
3. **Identifying Previous Flight Delay:**
    - Within the loop, we can use `df['TailNum'].shift(1)` to access the `"TailNum"` of the previous flight in the sorted order.
    - We then compare this shifted `"TailNum"` with the current row's `"TailNum"`.
    - If they match, it indicates the previous flight for the same plane on the same day.
    - In this case, we can access the corresponding delay value from the shifted row using `df['Delay'].shift(1)`.
4. **Handling First Flight:**
    - If the shifted `"TailNum"` doesn't match, it signifies the current flight is the first for the plane that day.
    - In this scenario, we assign a delay value of 0 (no previous flight on the same day).
5. **Assigning New Feature:**
    - By iterating through the DataFrame, we can create a new column named `"Previous_Flight_Delay"`.
    - This column will hold the delay of the most recent previous flight for each plane on the same day, or 0 for first flights.

By incorporating this feature, we can analyze if the delay of a plane's earlier flight on the same day might influence the delay of its current flight. This exploration helps build a more comprehensive understanding of potential factors impacting flight delays.

```
bay_area_flights_w_new_features['ArrDelay'] = pd.to_numeric(bay_area_flights_w_new_f
bay_area_flights_w_new_features['DepDelay'] = pd.to_numeric(bay_area_flights_w_new_f
bay_area_flights_w_new_features['TotalDelay'] = bay_area_flights_w_new_features['Arr
```

```
bay_area_flights_w_new_features.sort_values(by=['Date', 'TailNum', 'DepTime'], inpla
shifted_delays = bay_area_flights_w_new_features.groupby(['Date', 'TailNum']).shift(
bay_area_flights_w_new_features['Plane_Same_Day_Most_Recent_Flight_ArrDelay'] = shif
bay_area_flights_w_new_features['Plane_Same_Day_Most_Recent_Flight_DepDelay'] = shif
bay_area_flights_w_new_features['Plane_Same_Day_Most_Recent_Flight_TotalDelay'] = sh

bay_area_flights_w_new_features.fillna({'Plane_Same_Day_Most_Recent_Flight_ArrDelay'
                                        'Plane_Same_Day_Most_Recent_Flight_DepDelay'
                                        'Plane_Same_Day_Most_Recent_Flight_TotalDela

print(bay_area_flights_w_new_features.head())
```

```
       DayOfWeek       Date   DepTime   ArrTime  CRSArrTime UniqueCarrier  \
39906   Tuesday 2019-01-01  08:17:00  11:02:00         915            US
39858   Tuesday 2019-01-01  19:18:00  22:24:00        2121            US
74314   Tuesday 2019-01-01  19:25:00  21:04:00        2045            AA
45835   Tuesday 2019-01-01  20:59:00  22:12:00        2150            WN
29239   Tuesday 2019-01-01  16:42:00  23:46:00        2115            UA

                     Airline  FlightNum TailNum  ActualElapsedTime  ...  \
39906         US Airways Inc.        202  N164AW                105  ...
39858         US Airways Inc.        703  N171US                366  ...
74314  American Airlines Inc.       1303  N201AA                219  ...
45835  Southwest Airlines Co.       2285  N202WN                 73  ...
29239   United Air Lines Inc.         74  N213UA                304  ...

       NASDelay  SecurityDelay  LateAircraftDelay  DepHour Month  \
39906         0              0                  0        8     1
39858         0              0                 45       19     1
74314         0              0                  5       19     1
45835         3              0                 14       20     1
29239         9              0                136       16     1

      Same_Day_Num_Flights_of_this_Plane TotalDelay  \
39906                                  1        229
39858                                  1        146
74314                                  1         49
45835                                  1         41
29239                                  1        293

      Plane_Same_Day_Most_Recent_Flight_ArrDelay  \
39906                                        0.0
39858                                        0.0
74314                                        0.0
45835                                        0.0
29239                                        0.0

       Plane_Same_Day_Most_Recent_Flight_DepDelay  \
39906                                         0.0
39858                                         0.0
74314                                         0.0
45835                                         0.0
29239                                         0.0

       Plane_Same_Day_Most_Recent_Flight_TotalDelay
```

```
39906                                                0.0
39858                                                0.0
74314                                                0.0
45835                                                0.0
29239                                                0.0

[5 rows x 33 columns]
```

Focusing on San Francisco Bay Area Departures

This section refines the data to concentrate on flights originating from airports within the San Francisco Bay Area.

Considering Existing Filtering:

- We've previously filtered the data to include flights either departing from or arriving at a Bay Area airport (`bay_area_flights`).
- While the previous filtering considered both arrival and departure airports, our project's focus might be on understanding delays specifically for flights departing from the Bay Area.

Implementation: Conditional Filtering

- Create a new DataFrame containing only flights where `'Origin'` is in `bay_area_airports` if focusing solely on Bay Area departures is desired.

```
bay_area_airports = ['OAK', 'SFO', 'SJC']
bay_area_flights_w_new_features = bay_area_flights_w_new_features[bay_area_flights_w
print(bay_area_flights_w_new_features.shape)
print(bay_area_flights_w_new_features.head())
```

```
(23838, 33)
      DayOfWeek       Date   DepTime   ArrTime  CRSArrTime UniqueCarrier  \
39906   Tuesday 2019-01-01  08:17:00  11:02:00         915            US
45835   Tuesday 2019-01-01  20:59:00  22:12:00        2150            WN
29232   Tuesday 2019-01-01  15:20:00  18:30:00        1230            UA
17167   Tuesday 2019-01-01  11:49:00  12:26:00        1137            OO
17154   Tuesday 2019-01-01  13:45:00  14:53:00        1436            OO

                    Airline  FlightNum TailNum  ActualElapsedTime  ...  \
39906         US Airways Inc.        202  N164AW                105  ...
45835  Southwest Airlines Co.       2285  N202WN                 73  ...
29232   United Air Lines Inc.         73  N214UA                310  ...
17167   Skywest Airlines Inc.       5741  N218SW                 37  ...
17154   Skywest Airlines Inc.       5478  N218SW                 68  ...

       NASDelay  SecurityDelay  LateAircraftDelay  DepHour Month  \
39906         0              0                  0        8     1
45835         3              0                 14       20     1
29232         0              0                  0       15     1
17167         0              0                  0       11     1
```

```
17154         0               0                   17       13     1

      Same_Day_Num_Flights_of_this_Plane TotalDelay  \
39906                                  1        229
45835                                  1         41
29232                                  1        733
17167                                  4        103
17154                                  4         47

      Plane_Same_Day_Most_Recent_Flight_ArrDelay  \
39906                                        0.0
45835                                        0.0
29232                                        0.0
17167                                       40.0
17154                                       24.0

       Plane_Same_Day_Most_Recent_Flight_DepDelay  \
39906                                         0.0
45835                                         0.0
29232                                         0.0
17167                                        44.0
17154                                        31.0

       Plane_Same_Day_Most_Recent_Flight_TotalDelay
39906                                           0.0
45835                                           0.0
29232                                           0.0
17167                                          84.0
17154                                          55.0

[5 rows x 33 columns]
```

This section explores incorporating new features related to cumulative delays.

Goal:

- Understanding the overall delay patterns at airports or for airlines on a given day.

1. **Cumulative Delay by Airport:**
    - We can calculate the total delay for all flights departing from a specific airport on a given day.
    - This involves grouping the data by `'Date'` and `'Origin'`, then summing the `'Delay'` column within each group.
2. **Cumulative Delay by Airline:**
    - Similarly, grouping by `'Date'` and `'Airline'` can provide the total delay for each airline on a particular day.
    - Summing the `'Delay'` column within these groups would yield the cumulative delay per airline per day.

Implementation Using `groupby`:

- Group the data by the desired criteria (`'Date'` and `'Origin'` or `'Airline'`) and calculate the sum of delays within each group.

```
bay_area_flights_w_new_features['Cumulative_ArrDelay_Before'] = bay_area_flights_w_n
bay_area_flights_w_new_features['Cumulative_DepDelay_Before'] = bay_area_flights_w_n
bay_area_flights_w_new_features['Cumulative_ArrDep_Delay_Before'] = bay_area_flights

bay_area_flights_w_new_features['Cumulative_ArrDelay_Before_Same_Airline'] = bay_are
bay_area_flights_w_new_features['Cumulative_DepDelay_Before_Same_Airline'] = bay_are
bay_area_flights_w_new_features['Cumulative_ArrDep_Delay_Before_Same_Airline'] = bay

print(bay_area_flights_w_new_features[['Date', 'Origin', 'Airline', 'DepTime', 'ArrD
```

```
             Date Origin                 Airline   DepTime  ArrDelay  DepDelay  \
39906 2019-01-01    SJC         US Airways Inc.  08:17:00       107       122
45835 2019-01-01    SJC  Southwest Airlines Co.  20:59:00        22        19
29232 2019-01-01    SFO   United Air Lines Inc.  15:20:00       360       373
17167 2019-01-01    SFO   Skywest Airlines Inc.  11:49:00        49        54
17154 2019-01-01    SFO   Skywest Airlines Inc.  13:45:00        17        30

       Cumulative_ArrDep_Delay_Before  \
39906                               0
45835                             229
29232                               0
17167                             733
17154                             836

       Cumulative_ArrDep_Delay_Before_Same_Airline
39906                                            0
45835                                            0
29232                                            0
17167                                            0
17154                                          103
```

This section introduces additional features to the `bay_area_flights_w_new_features` DataFrame, potentially aiding in understanding how a flight's position within the day's schedule might influence delays.

Features Added:

1. **Flights Before:**
    - `'Flights_Before'` represents the number of flights departing from the same airport (`'Origin'`) on the same day (`'Date'`) that occurred before the current flight based on departure time (`'DepTime'`).
    - This is calculated using `groupby` and `cumcount` to assign a sequential number within each origin-date group.

2. **Flights Before (Same Airline):**
    - `'Flights_Before_Same_Airline'` extends the concept of `'Flights_Before'`, but considers only flights from the same airline (`'Airline'`) departing from the same airport (`'Origin'`) before the current flight.
    - This is computed similarly using `groupby` and `cumcount` within groups defined by `'Date'`, `'Airline'`, and `'Origin'`.
3. **Average Arrival Delay Before:**
    - `'Avg_ArrDelay_Before'` calculates the average arrival delay (`'ArrDelay'`) of flights departing from the same airport (`'Origin'`) on the same day (`'Date'`) that occurred before the current flight.
    - This leverages the `transform` function with a lambda expression.
        - The expression calculates the cumulative sum of arrival delays minus the current flight's delay.
        - It then divides by the rolling count of flights (excluding the current flight) within the same origin-date group using `rolling` and `count`.
4. **Average Departure Delay Before:**
    - `'Avg_DepDelay_Before'` employs the same logic as `'Avg_ArrDelay_Before'` but for departure delays (`'DepDelay'`).
5. **Average Arrival Delay Before (Same Airline):**
    - `'Avg_ArrDelay_Before_Same_Airline'` is similar to `'Avg_ArrDelay_Before'` but considers only flights from the same airline (`'Airline'`) departing from the same airport (`'Origin'`) before the current flight.
6. **Average Departure Delay Before (Same Airline):**
    - `'Avg_DepDelay_Before_Same_Airline'` mirrors `'Avg_ArrDelay_Before_Same_Airline'` for departure delays (`'DepDelay'`).

Benefits:

- These features capture the context of a flight within the day's schedule at a specific airport (both overall and for the same airline).
- They might provide insights into potential delay propagation or dependencies between flights.

```
bay_area_flights_w_new_features.sort_values(by=['Date', 'Origin', 'DepTime'], inplac
bay_area_flights_w_new_features['Flights_Before'] = bay_area_flights_w_new_features.
bay_area_flights_w_new_features['Flights_Before_Same_Airline'] = bay_area_flights_w_
bay_area_flights_w_new_features['Avg_ArrDelay_Before'] = bay_area_flights_w_new_feat
bay_area_flights_w_new_features['Avg_DepDelay_Before'] = bay_area_flights_w_new_feat
bay_area_flights_w_new_features['Avg_ArrDelay_Before_Same_Airline'] = bay_area_fligh
bay_area_flights_w_new_features['Avg_DepDelay_Before_Same_Airline'] = bay_area_fligh

print(bay_area_flights_w_new_features[['Date', 'Origin', 'Airline', 'DepTime', 'ArrD
```

```
            Date Origin                 Airline   DepTime  ArrDelay  DepDelay  \
29463 2019-01-01    OAK   United Air Lines Inc.  00:03:00        31        48
34277 2019-01-01    OAK   United Air Lines Inc.  06:28:00        16        19
45646 2019-01-01    OAK  Southwest Airlines Co.  08:22:00        18        22
80690 2019-01-01    OAK    Alaska Airlines Inc.  08:54:00        50        59
45664 2019-01-01    OAK  Southwest Airlines Co.  09:05:00        24        25

       Flights_Before  Flights_Before_Same_Airline  Avg_ArrDelay_Before  \
29463               0                            0                  NaN
34277               1                            1            31.000000
45646               2                            0            23.500000
80690               3                            0            21.666667
45664               4                            1            28.750000

       Avg_DepDelay_Before  Avg_ArrDelay_Before_Same_Airline  \
29463                  NaN                               NaN
34277            48.000000                              31.0
45646            33.500000                               NaN
80690            29.666667                               NaN
45664            37.000000                              18.0

       Avg_DepDelay_Before_Same_Airline
29463                               NaN
34277                              48.0
45646                               NaN
80690                               NaN
45664                              22.0
```

This section addresses the potential `NaN` values introduced during feature creation in `bay_area_flights_w_new_features`.

Reason for NaNs:

- These NaNs arise from the division by zero when calculating average delays before the first flight on a given day.If there are no preceding flights, there will be a zero denominator during division.

Imputation Strategy:

- We'll replace these `NaN` values with `0` since the first flight of the day wouldn't have any delays from previous flights to influence its own delay.

Implementation:

- A list named `columns_to_replace_nan` stores the column names containing potential `NaN` values.
- We iterate through the columns in the list using a `for` loop.
- Inside the loop, the `fillna(0)` method is applied to each column, replacing any `NaN` values with `0`.

Goal:

- Ensures all features have valid values for further analysis, and maintains the interpretation that the first flight of the day has no preceding flight delays.

```
columns_to_replace_nan = ['Avg_ArrDelay_Before', 'Avg_DepDelay_Before',
                          'Avg_ArrDelay_Before_Same_Airline', 'Avg_DepDelay_Before_S
for column in columns_to_replace_nan:
    bay_area_flights_w_new_features[column] = bay_area_flights_w_new_features[column

print(bay_area_flights_w_new_features[['Date', 'Origin', 'Airline', 'DepTime', 'ArrD
                       'Flights_Before', 'Flights_Before_Same_Airline', 'Avg_ArrDel
                       'Avg_DepDelay_Before', 'Avg_ArrDelay_Before_Same_Airline',
                       'Avg_DepDelay_Before_Same_Airline']].head())
```

```
             Date Origin                 Airline   DepTime  ArrDelay  DepDelay  \
29463 2019-01-01    OAK   United Air Lines Inc.  00:03:00        31        48
34277 2019-01-01    OAK   United Air Lines Inc.  06:28:00        16        19
45646 2019-01-01    OAK  Southwest Airlines Co.  08:22:00        18        22
80690 2019-01-01    OAK    Alaska Airlines Inc.  08:54:00        50        59
45664 2019-01-01    OAK  Southwest Airlines Co.  09:05:00        24        25

       Flights_Before  Flights_Before_Same_Airline  Avg_ArrDelay_Before  \
29463               0                            0             0.000000
34277               1                            1            31.000000
45646               2                            0            23.500000
80690               3                            0            21.666667
45664               4                            1            28.750000

       Avg_DepDelay_Before  Avg_ArrDelay_Before_Same_Airline  \
29463             0.000000                               0.0
34277            48.000000                              31.0
45646            33.500000                               0.0
80690            29.666667                               0.0
45664            37.000000                              18.0

       Avg_DepDelay_Before_Same_Airline
29463                               0.0
34277                              48.0
45646                               0.0
80690                               0.0
45664                              22.0
```

Let's check what features we have so far:

```
bay_area_flights_w_new_features.columns
```

```
Index(['DayOfWeek', 'Date', 'DepTime', 'ArrTime', 'CRSArrTime',
       'UniqueCarrier', 'Airline', 'FlightNum', 'TailNum', 'ActualElapsedTime',
       'CRSElapsedTime', 'AirTime', 'ArrDelay', 'DepDelay', 'Origin',
       'Org_Airport', 'Dest', 'Dest_Airport', 'Distance', 'TaxiIn', 'TaxiOut',
       'CarrierDelay', 'WeatherDelay', 'NASDelay', 'SecurityDelay',
       'LateAircraftDelay', 'DepHour', 'Month',
       'Same_Day_Num_Flights_of_this_Plane', 'TotalDelay',
       'Plane_Same_Day_Most_Recent_Flight_ArrDelay',
       'Plane_Same_Day_Most_Recent_Flight_DepDelay',
       'Plane_Same_Day_Most_Recent_Flight_TotalDelay',
       'Cumulative_ArrDelay_Before', 'Cumulative_DepDelay_Before',
       'Cumulative_ArrDep_Delay_Before',
       'Cumulative_ArrDelay_Before_Same_Airline',
       'Cumulative_DepDelay_Before_Same_Airline',
       'Cumulative_ArrDep_Delay_Before_Same_Airline', 'Flights_Before',
       'Flights_Before_Same_Airline', 'Avg_ArrDelay_Before',
       'Avg_DepDelay_Before', 'Avg_ArrDelay_Before_Same_Airline',
       'Avg_DepDelay_Before_Same_Airline'],
      dtype='object')
```

Store it in the Drive folder for better retrieval in the future:

```
bay_area_flights_w_new_features.to_csv('/content/drive/My Drive/ieor 142 final proje
bay_area_flights_w_new_features.head()
```

	DayOfWeek	Date	DepTime	ArrTime	CRSArrTime	UniqueCarrier	Airline	Flig
29463	Tuesday	2019-01-01	00:03:00	07:48:00	717	UA	United Air Lines Inc.	
34277	Tuesday	2019-01-01	06:28:00	12:33:00	1217	UA	United Air Lines Inc.	
45646	Tuesday	2019-01-01	08:22:00	09:33:00	915	WN	Southwest Airlines Co.	
80690	Tuesday	2019-01-01	08:54:00	10:23:00	933	AS	Alaska Airlines Inc.	
45664	Tuesday	2019-01-01	09:05:00	10:24:00	1000	WN	Southwest Airlines Co.	

5 rows × 45 columns

Bay Area Airports Weather Data

Incorporating Weather Data

This section explores integrating weather data from the National Centers for Environmental Information (NCEI) (https://www.ncdc.noaa.gov/cdo-web/search) into our analysis of flight delays.

Cherrypick Weather Data:

- We selected the 3 ZIP codes (94128, 94621, 95110) that the three bay area airports are in for the entire Year 2019.
- Upon closer inspection, these ZIP codes are confined to the immediate vicinity of the Bay Area airports.

Justification:

- Weather conditions can significantly impact flight operations and delays. We can identify weather-related factors influencing flight delays, specifically the geographical regions surrounding the Bay Area airports.

```
print(bay_weather.shape)
bay_weather.head()
```

```
(1095, 20)
```

	STATION	NAME	DATE	AWND	PGTM	PRCP	SNOW	SNWD	TAVG	TMAX	TMIN	WDF2
0	USW00023293	SAN JOSE, CA US	2019-01-01	5.82	NaN	0.0	NaN	NaN	NaN	55.0	36.0	50
1	USW00023293	SAN JOSE, CA US	2019-01-02	4.03	NaN	0.0	NaN	NaN	NaN	57.0	33.0	330

Next steps: View recommended plots

Let's inspect the distribution of missing values across features:

```
nan_counts = bay_weather.isna().sum()
print(nan_counts)
```

```
STATION       0
NAME          0
```

```
DATE          0
AWND          1
PGTM        997
PRCP          0
SNOW        788
SNWD        736
TAVG        730
TMAX          3
TMIN          1
WDF2          0
WDF5          7
WSF2          0
WSF5          7
WT01        763
WT02       1077
WT03       1078
WT05       1095
WT08        817
dtype: int64
```

Refining Weather Data Preprocessing

This section addresses the cleaning and preparation of weather data (`bay_weather`) for merging with the flight data.

Steps:

1. **Identifying Columns for Removal:**
   - A threshold is defined based on the total number of rows in the data (`len(bay_weather)`) divided by 5.
   - Columns with missing values (`isna().sum()`) exceeding this threshold are identified and stored in a list `cols_to_remove` .
   - These columns likely contain too many missing values to be reliable, so they are removed using `drop` .
2. **Column Renaming:**
   - A dictionary `renaming_dict` maps abbreviations used in the weather data to more descriptive names.
   - The `rename` function is applied to `bay_weather_cleaned` to replace the short codes with their corresponding human-readable names (e.g., `'PRCP'` to `'Precipitation'`).
3. **Extracting City and Airport Code:**
   - A function `extract_city` is defined to process the `'NAME'` column.
   - It checks if the city name (`name`) contains keywords like "OAKLAND", "SAN JOSE", or "SAN FRANCISCO".

- Based on the match, it returns a tuple with the city name and corresponding airport code (`'OAK'`, `'SJC'`, or `'SFO'`).
- If no match is found, it returns `np.nan` for both city and code.
- Two new columns, `'Airport City'` and `'Airport Code'`, are created using vectorized operations (`apply` and `zip`) to extract city names and airport codes from the `'NAME'` column using the `extract_city` function.

4. **Dropping Unnecessary Columns:**
   - Columns `'STATION'`, `'NAME'`, and `'Airport City'`, redundant after code extraction, are dropped using `drop`.

Improvements:

- Removing columns with excessive missing values reduces the impact of potentially unreliable data.
- Descriptive column names enhance readability and understanding of the weather data.
- Extracted city names and airport codes facilitate merging with the flight data based on airport locations.

```
threshold = len(bay_weather) // 5
cols_to_remove = [col for col in bay_weather.columns if bay_weather[col].isna().sum(
bay_weather_cleaned = bay_weather.drop(cols_to_remove, axis=1)
renaming_dict = {
    'PRCP': 'Precipitation',
    'SNWD': 'Snow_Depth',
    'SNOW': 'Snowfall',
    'TAVG': 'Average_Temperature',
    'TMAX': 'Maximum_Temperature',
    'TMIN': 'Minimum_Temperature',
    'AWND': 'Average_Wind_Speed',
    'WDF2': 'Direction_Fastest_2min_Wind',
    'WDF5': 'Direction_Fastest_5sec_Wind',
    'WSF2': 'Fastest_2min_Wind_Speed',
    'WSF5': 'Fastest_5sec_Wind_Speed',
    'PGTM': 'Peak_Gust_Time',
}

bay_weather_cleaned = bay_weather_cleaned.rename(columns=renaming_dict)

def extract_city(name):
    if 'OAKLAND' in name:
        return 'Oakland', 'OAK'
    elif 'SAN JOSE' in name:
        return 'San Jose', 'SJC'
    elif 'SAN FRANCISCO' in name:
        return 'San Francisco', 'SFO'
    return np.nan, np.nan

bay_weather_cleaned['Airport City'], bay_weather_cleaned['Airport Code'] = zip(*bay_
bay_weather_cleaned.drop(columns=['STATION', 'NAME', 'Airport City'], inplace=True)
bay_weather_cleaned.head()
```

	DATE	Average_Wind_Speed	Precipitation	Maximum_Temperature	Minimum_Temperat
0	2019-01-01	5.82	0.0	55.0	
1	2019-01-02	4.03	0.0	57.0	
2	2019-01-03	2.68	0.0	58.0	
3	2019-01-04	3.36	0.0	62.0	
4	2019-01-05	13.20	0.1	56.0	

Next steps: View recommended plots

```
bay_weather_cleaned['Airport Code'].unique()
```

```
array(['SJC', 'OAK', 'SFO'], dtype=object)
```

```
nan_counts = bay_weather_cleaned.isna().sum()
print(nan_counts)
```

```
DATE                          0
Average_Wind_Speed            1
Precipitation                 0
Maximum_Temperature           3
Minimum_Temperature           1
Direction_Fastest_2min_Wind   0
Direction_Fastest_5sec_Wind   7
Fastest_2min_Wind_Speed       0
Fastest_5sec_Wind_Speed       7
Airport Code                  0
dtype: int64
```

Here, we still have missing values for some features. But let's not worry about it now since we can simply remove the rows that contain them in the merged dataset because merging can result in other missing values as well and it will be more efficient to handle with them together:

Merging the datasets

Let's merge our transformed flight delay data and the airport weather data:

```
bay_weather_cleaned.rename(columns={'DATE': 'Date', 'Airport Code': 'Origin'}, inpla

bay_area_flights_w_new_features['Date'] = pd.to_datetime(bay_area_flights_w_new_feat
bay_weather_cleaned['Date'] = pd.to_datetime(bay_weather_cleaned['Date'])
bay_area_flights_w_new_features['Origin'] = bay_area_flights_w_new_features['Origin'
bay_weather_cleaned['Origin'] = bay_weather_cleaned['Origin'].astype(str)

merged_data = pd.merge(bay_area_flights_w_new_features, bay_weather_cleaned, on=['Da
print(merged_data.shape)
print(merged_data.head())
```

```
(23838, 53)
  DayOfWeek       Date   DepTime   ArrTime  CRSArrTime UniqueCarrier  \
0   Tuesday 2019-01-01  00:03:00  07:48:00         717            UA
1   Tuesday 2019-01-01  06:28:00  12:33:00        1217            UA
2   Tuesday 2019-01-01  08:22:00  09:33:00         915            WN
3   Tuesday 2019-01-01  08:54:00  10:23:00         933            AS
```

```
4   Tuesday 2019-01-01  09:05:00  10:24:00        1000            WN

                  Airline  FlightNum TailNum  ActualElapsedTime  ...  \
0   United Air Lines Inc.        112  N437UA                285  ...
1   United Air Lines Inc.        652  N803UA                245  ...
2  Southwest Airlines Co.       3245  N249WN                 71  ...
3    Alaska Airlines Inc.        407  N969AS                 89  ...
4  Southwest Airlines Co.       3570  N736SA                 79  ...

   Avg_ArrDelay_Before_Same_Airline  Avg_DepDelay_Before_Same_Airline  \
0                               0.0                               0.0
1                              31.0                              48.0
2                               0.0                               0.0
3                               0.0                               0.0
4                              18.0                              22.0

   Average_Wind_Speed  Precipitation Maximum_Temperature Minimum_Temperature  \
0                5.59            0.0                56.0                38.0
1                5.59            0.0                56.0                38.0
2                5.59            0.0                56.0                38.0
3                5.59            0.0                56.0                38.0
4                5.59            0.0                56.0                38.0

  Direction_Fastest_2min_Wind Direction_Fastest_5sec_Wind  \
0                          30                        30.0
1                          30                        30.0
2                          30                        30.0
3                          30                        30.0
4                          30                        30.0

   Fastest_2min_Wind_Speed  Fastest_5sec_Wind_Speed
0                     23.0                    25.9
1                     23.0                    25.9
2                     23.0                    25.9
3                     23.0                    25.9
4                     23.0                    25.9

[5 rows x 53 columns]
```

```
merged_data.columns
```

```
Index(['DayOfWeek', 'Date', 'DepTime', 'ArrTime', 'CRSArrTime',
       'UniqueCarrier', 'Airline', 'FlightNum', 'TailNum', 'ActualElapsedTime',
       'CRSElapsedTime', 'AirTime', 'ArrDelay', 'DepDelay', 'Origin',
       'Org_Airport', 'Dest', 'Dest_Airport', 'Distance', 'TaxiIn', 'TaxiOut',
       'CarrierDelay', 'WeatherDelay', 'NASDelay', 'SecurityDelay',
       'LateAircraftDelay', 'DepHour', 'Month',
       'Same_Day_Num_Flights_of_this_Plane', 'TotalDelay',
       'Plane_Same_Day_Most_Recent_Flight_ArrDelay',
       'Plane_Same_Day_Most_Recent_Flight_DepDelay',
       'Plane_Same_Day_Most_Recent_Flight_TotalDelay',
       'Cumulative_ArrDelay_Before', 'Cumulative_DepDelay_Before',
       'Cumulative_ArrDep_Delay_Before',
       'Cumulative_ArrDelay_Before_Same_Airline',
       'Cumulative_DepDelay_Before_Same_Airline',
```

```
       'Cumulative_ArrDep_Delay_Before_Same_Airline', 'Flights_Before',
       'Flights_Before_Same_Airline', 'Avg_ArrDelay_Before',
       'Avg_DepDelay_Before', 'Avg_ArrDelay_Before_Same_Airline',
       'Avg_DepDelay_Before_Same_Airline', 'Average_Wind_Speed',
       'Precipitation', 'Maximum_Temperature', 'Minimum_Temperature',
       'Direction_Fastest_2min_Wind', 'Direction_Fastest_5sec_Wind',
       'Fastest_2min_Wind_Speed', 'Fastest_5sec_Wind_Speed'],
      dtype='object')
```

Handling Missing Weather Data

This section addresses the presence of missing values in the weather data (`bay_weather_cleaned`) after merging it with the flight data (`bay_area_flights_w_new_features`) to create `merged_data`.

Justification for Removal:

- We decided to remove rows containing missing values in specific weather columns (`columns_to_check`).
- The rationale behind this approach is the assumption that eliminating these rows won't significantly impact the data distribution or introduce bias.

Implementation:

1. **Columns to Check:**
   - A list named `columns_to_check` identifies the weather-related columns where missing values will be addressed.
2. **Dropping Rows with Missing Values:**
   - The `dropna` function is applied to `merged_data`, specifying the `subset` parameter as `columns_to_check`.
   - This removes rows where any of the listed weather columns contain missing values.

```
columns_to_check = [
    'Average_Wind_Speed', 'Precipitation', 'Maximum_Temperature',
    'Minimum_Temperature', 'Direction_Fastest_2min_Wind',
    'Direction_Fastest_5sec_Wind', 'Fastest_2min_Wind_Speed',
    'Fastest_5sec_Wind_Speed'
]
merged_data_cleaned = merged_data.dropna(subset=columns_to_check)

print(merged_data_cleaned.shape)
print(merged_data_cleaned.head())
```

```
    (23506, 53)
```

Initialization

```
import os
os.getcwd()
```

```
'/content/drive/MyDrive/ieor 142 final project/Flight Delay/Transformed Data'
```

```
from google.colab import drive
drive.mount('/content/drive', force_remount=True)
new_path = '/content/drive/My Drive/ieor 142 final project/Flight Delay/Transformed
os.chdir(new_path)
```

```
Mounted at /content/drive
```

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

Feature Selection

Modeling Data

1. Data Preparation:

- In the preceding notebook, we conducted EDA and performed data pre-processing tasks. This notebook is dedicated to the modeling process. To begin, we will import the data that was saved in the previous notebook.

2. Loading Data:

- Let's start by reading the data that we prepared and stored in the other notebook.

```
merged_data_cleaned = pd.read_csv('/content/drive/My Drive/ieor 142 final project/Fl
```

```
merged_data_cleaned.columns
```

```
Index(['DayOfWeek', 'Date', 'DepTime', 'ArrTime', 'CRSArrTime',
       'UniqueCarrier', 'Airline', 'FlightNum', 'TailNum', 'ActualElapsedTime',
       'CRSElapsedTime', 'AirTime', 'ArrDelay', 'DepDelay', 'Origin',
       'Org_Airport', 'Dest', 'Dest_Airport', 'Distance', 'TaxiIn', 'TaxiOut',
```

```
 'CarrierDelay', 'WeatherDelay', 'NASDelay', 'SecurityDelay',
 'LateAircraftDelay', 'DepHour', 'Month',
 'Same_Day_Num_Flights_of_this_Plane', 'TotalDelay',
 'Plane_Same_Day_Most_Recent_Flight_ArrDelay',
 'Plane_Same_Day_Most_Recent_Flight_DepDelay',
 'Plane_Same_Day_Most_Recent_Flight_TotalDelay',
 'Cumulative_ArrDelay_Before', 'Cumulative_DepDelay_Before',
 'Cumulative_ArrDep_Delay_Before',
 'Cumulative_ArrDelay_Before_Same_Airline',
 'Cumulative_DepDelay_Before_Same_Airline',
 'Cumulative_ArrDep_Delay_Before_Same_Airline', 'Flights_Before',
 'Flights_Before_Same_Airline', 'Avg_ArrDelay_Before',
 'Avg_DepDelay_Before', 'Avg_ArrDelay_Before_Same_Airline',
 'Avg_DepDelay_Before_Same_Airline', 'Average_Wind_Speed',
 'Precipitation', 'Maximum_Temperature', 'Minimum_Temperature',
 'Direction_Fastest_2min_Wind', 'Direction_Fastest_5sec_Wind',
 'Fastest_2min_Wind_Speed', 'Fastest_5sec_Wind_Speed'],
dtype='object')
```

Feature Processing

1. Identifying Features to Remove:

- Now, we need to remove some features from our dataset that may no longer be useful for our model.

2. Criteria for Removal:

- For example, features like 'TaxiIn' and 'TaxiOut' might exhibit high correlations with delays. However, since our goal is to predict departure delays, and the 'TaxiIn' and 'TaxiOut' values are only available after boarding, they wouldn't realistically aid travelers in predicting delays.

3. Removal Process:

- We'll remove features related to cumulative delays since they may have high correlations with average previous delays and the hour of the day. This could lead to severe multicollinearity issues for linear regression.

```
modeling_data = merged_data_cleaned.copy()
columns_to_remove = ['Date', 'DepTime', 'ArrTime', 'CRSArrTime', 'UniqueCarrier', 'F
                     'ArrDelay', 'Dest', 'Dest_Airport', 'CarrierDelay', 'WeatherDel
                     'Cumulative_ArrDep_Delay_Before', 'Cumulative_ArrDelay_Before_S
                     'Cumulative_ArrDep_Delay_Before_Same_Airline', 'Cumulative_DepD
                     'ActualElapsedTime']
modeling_data.drop(columns=columns_to_remove, inplace=True)
print(modeling_data.shape)
print(modeling_data.head())
```

```
(23506, 27)
  DayOfWeek                 Airline  AirTime  DepDelay Origin  Distance  \
0   Tuesday   United Air Lines Inc.      269        48    OAK      2408
1   Tuesday   United Air Lines Inc.      216        19    OAK      1835
2   Tuesday  Southwest Airlines Co.       57        22    OAK       337
3   Tuesday    Alaska Airlines Inc.       76        59    OAK       543
4   Tuesday  Southwest Airlines Co.       65        25    OAK       371

   DepHour  Month  Same_Day_Num_Flights_of_this_Plane  TotalDelay  ...  \
0        0      1                                   1          79  ...
1        6      1                                   1          35  ...
2        8      1                                   2          40  ...
3        8      1                                   1         109  ...
4        9      1                                   2          49  ...

   Avg_ArrDelay_Before_Same_Airline  Avg_DepDelay_Before_Same_Airline  \
0                               0.0                               0.0
1                              31.0                              48.0
2                               0.0                               0.0
3                               0.0                               0.0
4                              18.0                              22.0

   Average_Wind_Speed  Precipitation  Maximum_Temperature  \
0                5.59            0.0                 56.0
1                5.59            0.0                 56.0
2                5.59            0.0                 56.0
3                5.59            0.0                 56.0
4                5.59            0.0                 56.0

   Minimum_Temperature  Direction_Fastest_2min_Wind  \
0                 38.0                           30
1                 38.0                           30
2                 38.0                           30
3                 38.0                           30
4                 38.0                           30

   Direction_Fastest_5sec_Wind  Fastest_2min_Wind_Speed  \
0                         30.0                     23.0
1                         30.0                     23.0
2                         30.0                     23.0
3                         30.0                     23.0
4                         30.0                     23.0

   Fastest_5sec_Wind_Speed
0                     25.9
```

```
1                    25.9
2                    25.9
3                    25.9
4                    25.9

[5 rows x 27 columns]
```

Outlier Removal

1. Identifying Outliers:

- Later in our analysis, we will inspect scatter plots to identify outliers.

2. Code Addition for Improved Visualization:

- We include the code to remove outliers here since it can help improve the scaling of the plot axes.

```
def remove_top_outliers(data, num_outliers=5):
    for column in data.select_dtypes(include=['float64', 'int64']).columns:
        largest_values = data[column].nlargest(num_outliers).index
        data = data.drop(index=largest_values)
    return data

modeling_data_less_outliers = remove_top_outliers(modeling_data.copy(), num_outliers
print(modeling_data_less_outliers.shape)
```

```
(23266, 27)
```

Visualizing Feature Distribution

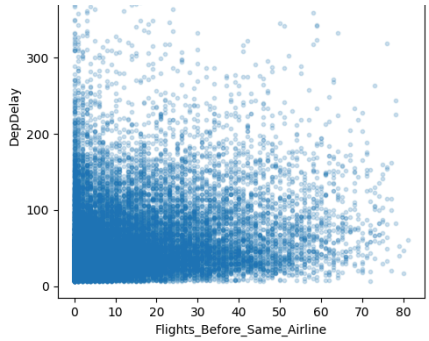
1. Selecting the Y Variable

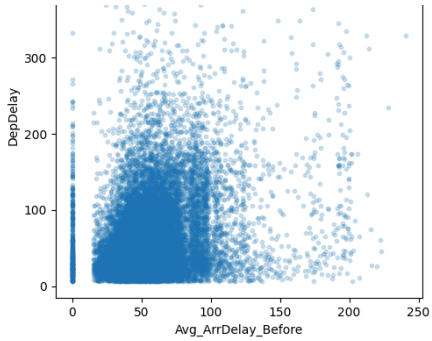
- Our Y variable of interest is departure delay ("**DepDelay**").

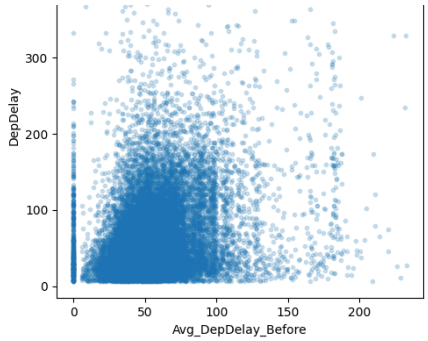
2. Visualizing Distribution

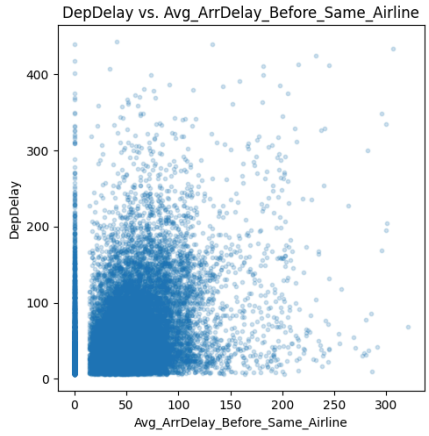
- We will create scatter plots to visualize the distribution between departure delay ("**DepDelay**") and each feature.

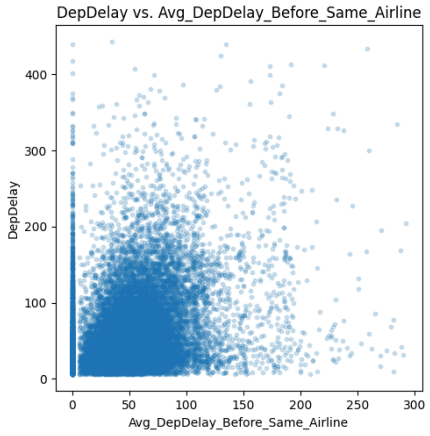
```
import math

numeric_features = [
    'AirTime', 'Distance', 'Flights_Before', 'Flights_Before_Same_Airline',
    'Avg_ArrDelay_Before', 'Avg_DepDelay_Before', 'Avg_ArrDelay_Before_Same_Airline'
    'Avg_DepDelay_Before_Same_Airline', 'Average_Wind_Speed', 'Precipitation',
    'Maximum_Temperature', 'Minimum_Temperature', 'Direction_Fastest_2min_Wind',
    'Direction_Fastest_5sec_Wind', 'Fastest_2min_Wind_Speed', 'Fastest_5sec_Wind_Spe
    'Plane_Same_Day_Most_Recent_Flight_ArrDelay', 'Plane_Same_Day_Most_Recent_Flight
    'Plane_Same_Day_Most_Recent_Flight_TotalDelay'
]

num_features = len(numeric_features)
num_cols = 3
num_rows = math.ceil(num_features / num_cols)
fig, axes = plt.subplots(nrows=num_rows, ncols=num_cols, figsize=(15, num_rows * 5))
axes = axes.ravel()

for idx, feature in enumerate(numeric_features):
    axes[idx].scatter(modeling_data_less_outliers[feature], modeling_data_less_outli
    axes[idx].set_title(f'DepDelay vs. {feature}')
    axes[idx].set_xlabel(feature)
    axes[idx].set_ylabel('DepDelay')

for idx in range(len(numeric_features), len(axes)):
    fig.delaxes(axes[idx])

plt.tight_layout()
plt.show()
```

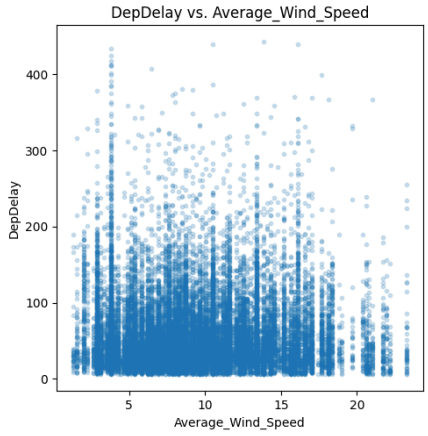

DepDelay
Flights_Before_Same_Airline

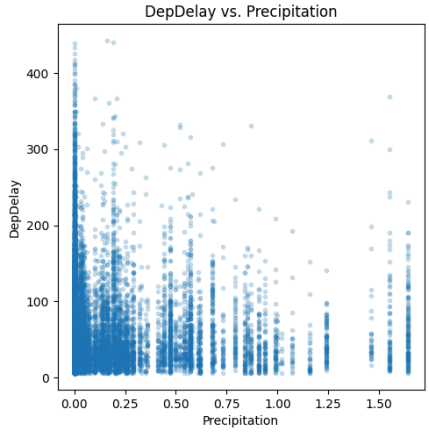

DepDelay
Avg_ArrDelay_Before

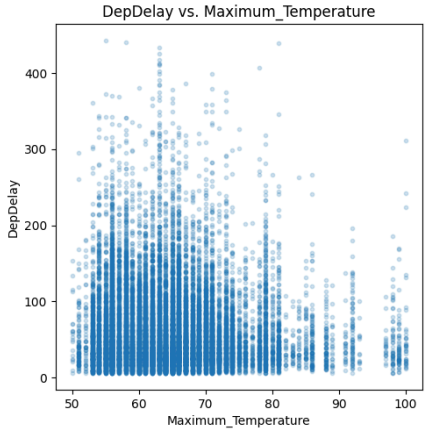

DepDelay
Avg_DepDelay_Before

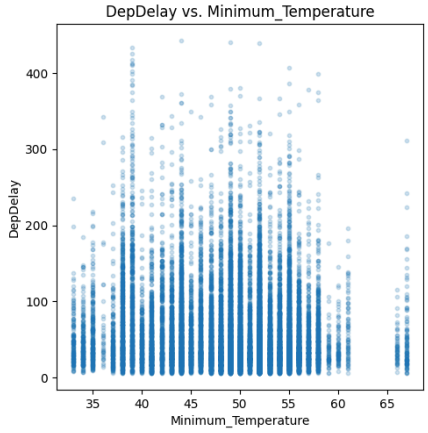

DepDelay vs. Avg_ArrDelay_Before_Same_Airline
DepDelay
Avg_ArrDelay_Before_Same_Airline

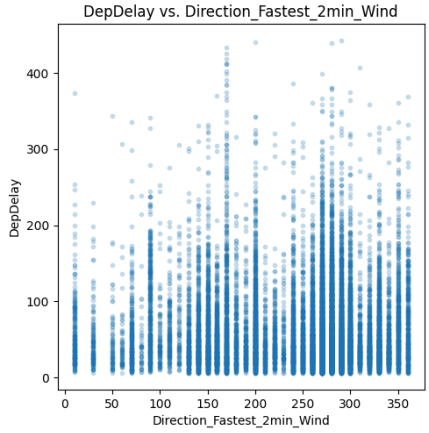

DepDelay vs. Avg_DepDelay_Before_Same_Airline
DepDelay
Avg_DepDelay_Before_Same_Airline

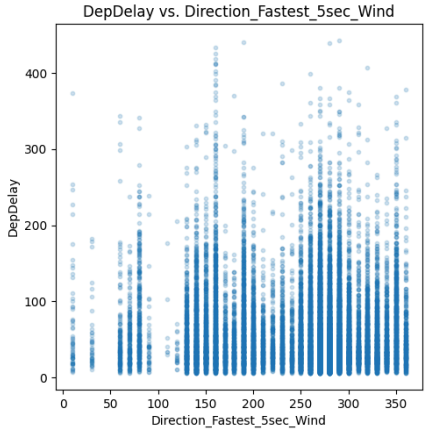

DepDelay vs. Average_Wind_Speed
DepDelay
Average_Wind_Speed

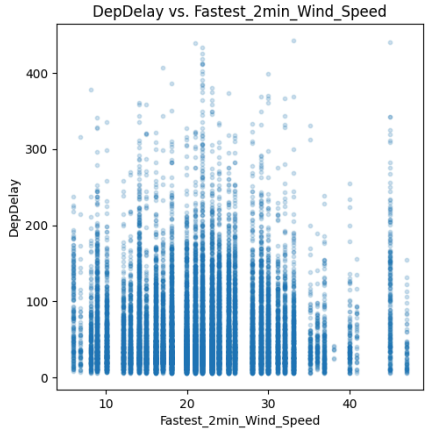

DepDelay vs. Precipitation
DepDelay
Precipitation

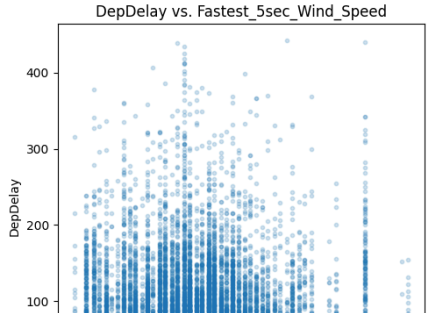

DepDelay vs. Maximum_Temperature
DepDelay
Maximum_Temperature

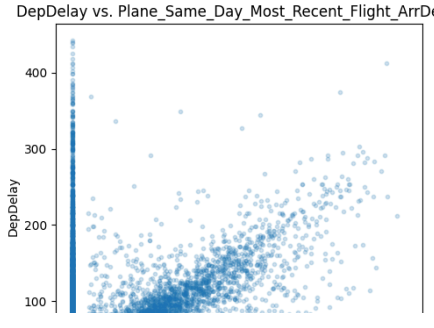

DepDelay vs. Minimum_Temperature
DepDelay
Minimum_Temperature

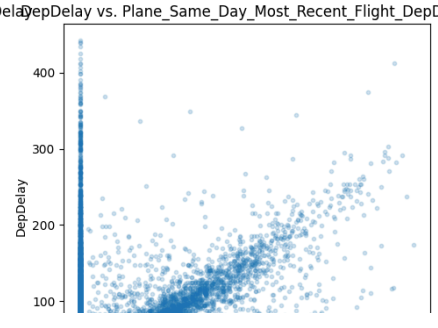

DepDelay vs. Direction_Fastest_2min_Wind
DepDelay
Direction_Fastest_2min_Wind


DepDelay vs. Direction_Fastest_5sec_Wind
DepDelay
Direction_Fastest_5sec_Wind


DepDelay vs. Fastest_2min_Wind_Speed
DepDelay
Fastest_2min_Wind_Speed


DepDelay vs. Fastest_5sec_Wind_Speed
DepDelay


DepDelay vs. Plane_Same_Day_Most_Recent_Flight_ArrDelay
DepDelay


DepDelay vs. Plane_Same_Day_Most_Recent_Flight_DepDelay
DepDelay

Identifying and Removing Zero-Value Features

1. Observing Plots

- From the plots, we notice that certain features have numerous zero values.

2. Removing Zero Values

- Upon inspection, we conclude that these zero values can be considered outliers as they lack real meaning and distort the trend significantly, then they would be removed.

```
def remove_zero_outliers(data, features):
    for feature in features:
        data = data[data[feature] != 0]
    return data

flight_features = ['AirTime', 'Distance', 'Flights_Before', 'Flights_Before_Same_Air
                   'Avg_ArrDelay_Before', 'Avg_DepDelay_Before',
                   'Avg_ArrDelay_Before_Same_Airline', 'Avg_DepDelay_Before_Same_Ai

modeling_data_less_outliers = remove_zero_outliers(modeling_data_less_outliers, flig
print(modeling_data_less_outliers.shape)
```

```
    (20259, 27)
```

Handling Missing Values

- Before regression, we need to make sure no more missing values exist

```
nan_counts = modeling_data_less_outliers.isna().sum()
print(nan_counts)
```

```
    DayOfWeek                                0
    Airline                                  0
    AirTime                                  0
    DepDelay                                 0
    Origin                                   0
    Distance                                 0
    DepHour                                  0
    Month                                    0
    Same_Day_Num_Flights_of_this_Plane       0
    TotalDelay                               0
```

```
Plane_Same_Day_Most_Recent_Flight_ArrDelay      0
Plane_Same_Day_Most_Recent_Flight_DepDelay      0
Plane_Same_Day_Most_Recent_Flight_TotalDelay    0
Flights_Before                                  0
Flights_Before_Same_Airline                     0
Avg_ArrDelay_Before                             0
Avg_DepDelay_Before                             0
Avg_ArrDelay_Before_Same_Airline                0
Avg_DepDelay_Before_Same_Airline                0
Average_Wind_Speed                              0
Precipitation                                   0
Maximum_Temperature                             0
Minimum_Temperature                             0
Direction_Fastest_2min_Wind                     0
Direction_Fastest_5sec_Wind                     0
Fastest_2min_Wind_Speed                         0
Fastest_5sec_Wind_Speed                         0
dtype: int64
```

Creating Dummy Variables for Categorical Features

1. Identification of Categorical Features

- Identify the categorical features in the dataset that need to be converted into dummy variables.

2. Dummy Variable Creation

- Using the `pd.get_dummies()` function, we will create dummy variables for these categorical features.

```
categorical_features = ['DayOfWeek', 'Airline', 'Origin', 'DepHour', 'Month']
modeling_data = pd.get_dummies(modeling_data_less_outliers, columns=categorical_feat
print(modeling_data.shape)
print(modeling_data.head())
```

```
(20259, 71)
   AirTime  DepDelay  Distance  Same_Day_Num_Flights_of_this_Plane  \
1      216        19      1835                                   1
4       65        25       371                                   2
5       64        25       407                                   2
6      220        16      1844                                   2
7       90        10       671                                   3

   TotalDelay  Plane_Same_Day_Most_Recent_Flight_ArrDelay  \
1          35                                         0.0
4          49                                         0.0
5          44                                         0.0
6          39                                         0.0
```

```
7          31                                          87.0

   Plane_Same_Day_Most_Recent_Flight_DepDelay  \
1                                         0.0
4                                         0.0
5                                         0.0
6                                         0.0
7                                        79.0

   Plane_Same_Day_Most_Recent_Flight_TotalDelay  Flights_Before  \
1                                           0.0               1
4                                           0.0               4
5                                           0.0               5
6                                           0.0               6
7                                         166.0               7

   Flights_Before_Same_Airline  ...  DepHour_20  DepHour_21  DepHour_22  \
1                            1  ...       False       False       False
4                            1  ...       False       False       False
5                            2  ...       False       False       False
6                            3  ...       False       False       False
7                            1  ...       False       False       False

   DepHour_23  Month_1  Month_2  Month_3  Month_4  Month_5  Month_6
1       False     True    False    False    False    False    False
4       False     True    False    False    False    False    False
5       False     True    False    False    False    False    False
6       False     True    False    False    False    False    False
7       False     True    False    False    False    False    False

[5 rows x 71 columns]
```

Splitting Data into Training, Validation, and Test Sets

```
from sklearn.model_selection import train_test_split

train_data, temp_data = train_test_split(modeling_data, test_size=0.4, random_state=
validation_data, test_data = train_test_split(temp_data, test_size=0.5, random_state

print("Training Set Shape:", train_data.shape)
print("Validation Set Shape:", validation_data.shape)
print("Test Set Shape:", test_data.shape)
```

```
Training Set Shape: (12155, 71)
Validation Set Shape: (4052, 71)
Test Set Shape: (4052, 71)
```

Checking Correlation between Y Variable and Features

```
correlation_matrix = train_data.corr()
dep_delay_correlations = correlation_matrix['DepDelay']
dep_delay_correlations.sort_values(ascending=False)

pd.set_option('display.max_rows', None)
print(dep_delay_correlations.sort_values(ascending=False))
pd.reset_option('display.max_rows')
```

```
DepHour_0                                  0.088616
DepHour_1                                  0.086447
DepHour_23                                 0.078319
Airline_American Airlines Inc.             0.077384
Distance                                   0.076812
Airline_United Air Lines Inc.              0.071886
AirTime                                    0.061563
DepHour_3                                  0.058703
Airline_Skywest Airlines Inc.              0.051240
DayOfWeek_Saturday                         0.048543
DepHour_17                                 0.044730
DepHour_2                                  0.043489
DepHour_4                                  0.036486
DepHour_22                                 0.035198
DepHour_18                                 0.029756
DayOfWeek_Tuesday                          0.029721
DepHour_15                                 0.028170
Month_2                                    0.023074
Fastest_2min_Wind_Speed                    0.015926
Airline_US Airways Inc.                    0.015558
Airline_Alaska Airlines Inc.               0.014334
DayOfWeek_Friday                           0.013695
DayOfWeek_Monday                           0.009171
DepHour_14                                 0.007193
Airline_JetBlue Airways                    0.006642
Fastest_5sec_Wind_Speed                    0.005910
DepHour_16                                 0.005794
DepHour_19                                 0.004195
DepHour_21                                 0.002312
Month_6                                    0.002078
Airline_Delta Air Lines Inc.               0.001348
Airline_Frontier Airlines Inc.             0.001026
DepHour_20                                -0.001667
```

```
DepHour_9                                -0.057674
Minimum_Temperature                      -0.059254
DepHour_8                                -0.064014
Month_4                                  -0.064068
DayOfWeek_Wednesday                      -0.065441
Month_3                                  -0.068853
Origin_SJC                               -0.077865
DepHour_11                               -0.084213
Origin_OAK                               -0.138706
Airline_Southwest Airlines Co.           -0.160270
Name: DepDelay, dtype: float64
```

Addressing Potential Multi-collinearity

1. Pre-Modeling Check

- Before proceeding with modeling, we need to address potential multicollinearity issues as they can impact the interpretability of our model.

2. Checking VIF

- We will calculate the Variance Inflation Factor (VIF) for each feature.

3. Note on Dummy Variables

- We will not account for the VIFs for the dummy variables here. Typically, dummy variables have high VIFs due to their correlation structure, and it's not a usual practice to assess multicollinearity for them.

```
train_data.drop(['TotalDelay'], axis=1, inplace=True)
validation_data.drop(['TotalDelay'], axis=1, inplace=True)
test_data.drop(['TotalDelay'], axis=1, inplace=True)
```

```
from statsmodels.stats.outliers_influence import variance_inflation_factor
from statsmodels.tools.tools import add_constant

X_numeric = train_data[numeric_features]
X_numeric = add_constant(X_numeric)
vif_data = pd.DataFrame()
vif_data['Feature'] = X_numeric.columns
vif_data['VIF'] = [variance_inflation_factor(X_numeric.values, i) for i in range(X_n
pd.set_option('display.max_rows', None)
print(vif_data)
```

```
                                    Feature         VIF
0                                     const  118.056769
1                                   AirTime   62.495651
2                                  Distance   62.521586
3                             Flights_Before    2.403590
```

```
4                    Flights_Before_Same_Airline    1.931827
5                            Avg_ArrDelay_Before   35.945307
6                            Avg_DepDelay_Before   35.239579
7                Avg_ArrDelay_Before_Same_Airline   26.811215
8                Avg_DepDelay_Before_Same_Airline   26.241141
9                             Average_Wind_Speed    4.497150
10                                 Precipitation    1.553839
11                           Maximum_Temperature    2.286984
12                           Minimum_Temperature    2.447970
13                   Direction_Fastest_2min_Wind    3.679891
14                   Direction_Fastest_5sec_Wind    3.732849
15                       Fastest_2min_Wind_Speed   22.474269
16                       Fastest_5sec_Wind_Speed   23.200274
17    Plane_Same_Day_Most_Recent_Flight_ArrDelay         inf
18    Plane_Same_Day_Most_Recent_Flight_DepDelay         inf
19  Plane_Same_Day_Most_Recent_Flight_TotalDelay         inf
/usr/local/lib/python3.10/dist-packages/statsmodels/stats/outliers_influence.py:
  vif = 1. / (1. - r_squared_i)
```

Iterative Removal of Features with High VIF

1. Initial VIF Check

- We will start by checking the VIF for each feature.

2. Removal Process

- For features with high VIFs that we are less interested in, we will iteratively remove them and check VIF again.

3. Repeat Until VIFs Drop Below Threshold

- We will repeat the removal process until all VIFs drop to below 5, indicating acceptable levels of multicollinearity.

```
vif_kept_features = numeric_features.copy()
vif_kept_features.remove('Plane_Same_Day_Most_Recent_Flight_TotalDelay')
vif_kept_features.remove('Plane_Same_Day_Most_Recent_Flight_ArrDelay')
vif_kept_features.remove('Fastest_5sec_Wind_Speed')
vif_kept_features.remove('Direction_Fastest_2min_Wind')
vif_kept_features.remove('Fastest_2min_Wind_Speed')
vif_kept_features.remove('Avg_ArrDelay_Before')
vif_kept_features.remove('AirTime')
vif_kept_features.remove('Avg_ArrDelay_Before_Same_Airline')

removed_features_due_to_vif = [
    'Plane_Same_Day_Most_Recent_Flight_TotalDelay',
    'Plane_Same_Day_Most_Recent_Flight_ArrDelay',
    'Fastest_5sec_Wind_Speed',
    'Direction_Fastest_2min_Wind',
    'Fastest_2min_Wind_Speed',
    'Avg_ArrDelay_Before',
    'AirTime',
    'Avg_ArrDelay_Before_Same_Airline'
]


X_numeric = train_data[vif_kept_features]
X_numeric = add_constant(X_numeric)
vif_data = pd.DataFrame()
vif_data['Feature'] = X_numeric.columns
vif_data['VIF'] = [variance_inflation_factor(X_numeric.values, i) for i in range(X_n
pd.set_option('display.max_rows', None)
print(vif_data)
```

```
                                        Feature         VIF
    0                                     const  107.351435
    1                                  Distance    1.088794
    2                             Flights_Before    2.330312
    3                  Flights_Before_Same_Airline    1.899415
    4                           Avg_DepDelay_Before    2.569609
    5              Avg_DepDelay_Before_Same_Airline    2.272588
    6                            Average_Wind_Speed    1.507619
    7                                 Precipitation    1.382270
    8                           Maximum_Temperature    2.240552
    9                           Minimum_Temperature    2.373887
    10                 Direction_Fastest_5sec_Wind    1.299229
    11  Plane_Same_Day_Most_Recent_Flight_DepDelay    1.075476
```

Removing Unnecessary Features

- We will identify the features that we do not need from the train, validation, and test datasets, and remove these features from all datasets.

```
train_data.drop(columns=removed_features_due_to_vif, inplace=True)
validation_data.drop(columns=removed_features_due_to_vif, inplace=True)
test_data.drop(columns=removed_features_due_to_vif, inplace=True)
print(train_data.head())
```

```
       DepDelay  Distance  Same_Day_Num_Flights_of_this_Plane  \
21559        33       621                                   2
3527        103       353                                   4
18804        54       329                                   4
1686         43       386                                   3
11928        96       550                                   6

       Plane_Same_Day_Most_Recent_Flight_DepDelay  Flights_Before  \
21559                                         0.0               5
3527                                         73.0              33
18804                                        58.0              71
1686                                          0.0              20
11928                                         0.0             163

       Flights_Before_Same_Airline  Avg_DepDelay_Before  \
21559                            2            65.400000
3527                             1            56.393939
18804                           22            59.985915
1686                             8            53.850000
11928                           47            69.128834

       Avg_DepDelay_Before_Same_Airline  Average_Wind_Speed  Precipitation  \
21559                         86.000000                7.61           0.00
3527                          48.000000               10.07           0.01
18804                         63.500000               10.51           0.00
1686                          35.375000               12.75           0.79
11928                         69.106383                6.93           0.00

       ...  DepHour_20  DepHour_21  DepHour_22  DepHour_23  Month_1  Month_2  \
21559  ...       False       False       False       False    False    False
3527   ...       False       False       False       False     True    False
18804  ...       False       False       False       False    False    False
1686   ...       False       False       False       False     True    False
11928  ...       False       False       False        True    False    False

       Month_3  Month_4  Month_5  Month_6
21559    False    False    False     True
3527     False    False    False    False
18804    False    False     True    False
1686     False    False    False    False
11928     True    False    False    False

[5 rows x 62 columns]
```

Now our data is ready for modeling. Let's move on.

Linear Regression Modeling

1. **Prepare Data:**

- Continuous adjustments to data types and constants are added to fit the model.

2. **Model Fitting:**

- A linear regression model is fitted using the OLS method from the statsmodels library. The summary provides detailed statistics about the model's performance and the significance of features.

```
for column in train_data.columns:
    train_data[column] = pd.to_numeric(train_data[column], errors='coerce')

bool_columns = train_data.select_dtypes(include=['bool']).columns
train_data[bool_columns] = train_data[bool_columns].astype(int)
print(train_data.dtypes)
```

```
Flights_Before_Same_Airline                   int64
Avg_DepDelay_Before                         float64
Avg_DepDelay_Before_Same_Airline            float64
Average_Wind_Speed                          float64
Precipitation                               float64
Maximum_Temperature                         float64
Minimum_Temperature                         float64
Direction_Fastest_5sec_Wind                 float64
DayOfWeek_Friday                              int64
DayOfWeek_Monday                              int64
DayOfWeek_Saturday                            int64
DayOfWeek_Sunday                              int64
DayOfWeek_Thursday                            int64
DayOfWeek_Tuesday                             int64
DayOfWeek_Wednesday                           int64
Airline_Alaska Airlines Inc.                  int64
Airline_American Airlines Inc.                int64
Airline_American Eagle Airlines Inc.          int64
Airline_Delta Air Lines Inc.                  int64
Airline_Frontier Airlines Inc.                int64
Airline_JetBlue Airways                       int64
Airline_Skywest Airlines Inc.                 int64
Airline_Southwest Airlines Co.                int64
Airline_US Airways Inc.                       int64
Airline_United Air Lines Inc.                 int64
Origin_OAK                                    int64
Origin_SFO                                    int64
Origin_SJC                                    int64
DepHour_0                                     int64
DepHour_1                                     int64
DepHour_2                                     int64
```

```
DepHour_9                                          int64
DepHour_10                                         int64
DepHour_11                                         int64
DepHour_12                                         int64
DepHour_13                                         int64
DepHour_14                                         int64
DepHour_15                                         int64
DepHour_16                                         int64
DepHour_17                                         int64
DepHour_18                                         int64
DepHour_19                                         int64
DepHour_20                                         int64
DepHour_21                                         int64
DepHour_22                                         int64
DepHour_23                                         int64
Month_1                                            int64
Month_2                                            int64
Month_3                                            int64
Month_4                                            int64
Month_5                                            int64
Month_6                                            int64
dtype: object
```

```
import statsmodels.api as sm

columns_to_drop = ['DayOfWeek_Monday', 'Airline_Delta Air Lines Inc.', 'Origin_OAK',
X_train_lin_reg = train_data.drop(['DepDelay'] + columns_to_drop, axis=1)
X_train_lin_reg = sm.add_constant(X_train_lin_reg)
y_train_lin_reg = train_data['DepDelay']

model = sm.OLS(y_train_lin_reg, X_train_lin_reg).fit()
print(model.summary())
```

```
DepHour_3                                94.7858     25.886      3.662
DepHour_4                               136.1790     43.865      3.105
DepHour_6                               -84.1826     10.181     -8.268
DepHour_7                               -57.3108      7.975     -7.186
DepHour_8                               -51.5491      7.171     -7.189
DepHour_9                               -43.5154      6.940     -6.270
DepHour_10                              -38.1312      6.867     -5.553
DepHour_11                              -39.8128      6.723     -5.922
DepHour_12                              -30.5237      6.705     -4.552
DepHour_13                              -31.7367      6.685     -4.747
DepHour_14                              -28.0655      6.697     -4.191
DepHour_15                              -24.4107      6.732     -3.626
DepHour_16                              -29.0791      6.733     -4.319
DepHour_17                              -23.9846      6.783     -3.536
DepHour_18                              -24.9894      6.796     -3.677
DepHour_19                              -28.0968      6.834     -4.112
DepHour_20                              -26.9350      6.874     -3.919
DepHour_21                              -23.7366      6.878     -3.451
DepHour_22                              -22.3206      6.957     -3.208
DepHour_23                              -19.1701      7.101     -2.700
Month_2                                   0.6178      1.397      0.442
Month_3                                  -4.0643      1.406     -2.891
Month_4                                  -6.2955      1.827     -3.446
Month_5                                  -3.7677      1.654     -2.278
Month_6                                  -6.4743      1.988     -3.256
==============================================================================
Omnibus:                     5223.533   Durbin-Watson:                   2.009
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            35771.813
Skew:                           1.931   Prob(JB):                         0.00
Kurtosis:                      10.464   Cond. No.                     1.21e+05
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
[2] The condition number is large, 1.21e+05. This might indicate that there are
strong multicollinearity or other numerical problems.
```

Regression Tree

1. **Tree Model Fitting:**

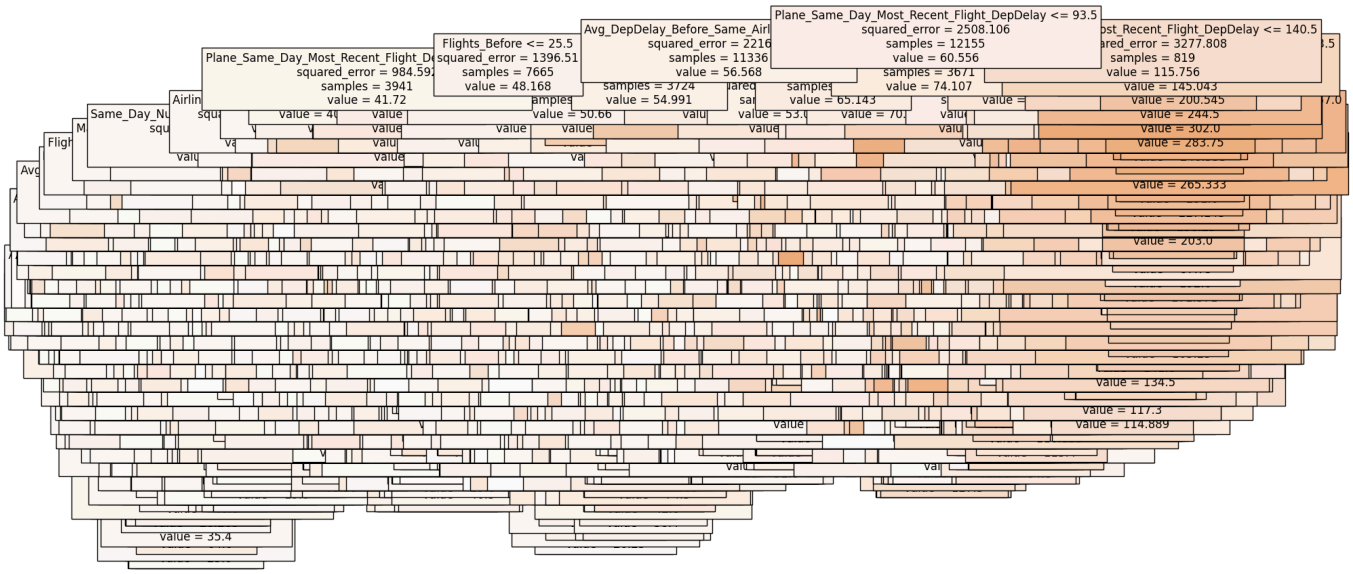
- A decision tree for regression is fitted. Parameters are tuned using GridSearchCV, which systematically works through multiple combinations of parameter tunes, cross-validating as it goes to determine which tune gives the best performance.

2. **Model Evaluation:**

- The performance of the tree model is evaluated using the validation set
- Metrics like R-squared and MAE are calculated.

```
from sklearn.tree import DecisionTreeRegressor, plot_tree
from sklearn.model_selection import train_test_split, GridSearchCV

X_train_reg_tree = train_data.drop('DepDelay', axis=1)
y_train_reg_tree = train_data['DepDelay']

tree_model = DecisionTreeRegressor(random_state=0)
param_grid = {'ccp_alpha': np.linspace(0.0001, 0.01, 100)}
grid_search = GridSearchCV(estimator=tree_model, param_grid=param_grid, cv=5, scorin
grid_search.fit(X_train_reg_tree, y_train_reg_tree)

print("Best ccp_alpha:", grid_search.best_params_['ccp_alpha'])
print("Best estimator:", grid_search.best_estimator_)
feature_names = X_train_reg_tree.columns
plt.figure(figsize=(20, 10))
plot_tree(grid_search.best_estimator_, filled=True, feature_names=feature_names, fon
plt.show()
```

```
Best ccp_alpha: 0.009899999999999999
Best estimator: DecisionTreeRegressor(ccp_alpha=0.009899999999999999, random_sta
```



```
from sklearn.metrics import r2_score, mean_absolute_error

X_val_reg_tree = validation_data.drop(['DepDelay'], axis=1)
y_val_reg_tree = validation_data['DepDelay']
tree_predictions = grid_search.best_estimator_.predict(X_val_reg_tree)
tree_OSR2 = r2_score(y_val_reg_tree, tree_predictions)
tree_MAE = mean_absolute_error(y_val_reg_tree, tree_predictions)
print(f"Decision Tree OSR2 (using the original feature data): {tree_OSR2:.4f}")
print(f"Decision Tree MAE (using the original feature data): {tree_MAE:.4f}")
```

```
Decision Tree OSR2 (using the original feature data): -0.3616
Decision Tree MAE (using the original feature data): 37.1050
```

Switching to Classification Problem

Since the regression output is not ideal, let's switch gear a bit to predict whether a flight would be significant delay (more than an hour). This might still be useful as travels can plan their time arriving at the airport more wisely.

1. **Logistic Regression:**

- Given suboptimal regression results, a shift to classification is made to predict whether a delay is significant. Logistic regression is applied, and performance metrics such as accuracy, precision, recall, and F1 score are evaluated.

2. **Classification Tree:**

- A decision tree classifier is trained, and performance metrics are again calculated to evaluate its efficacy in predicting significant delays.

3. **Testing:**

- The trained classification model is tested with the test dataset to assess its performance in a real-world scenario, aiming to provide reliable predictions on flight delays.

Logistic Regression

```
train_data['Significant_DepDelay'] = (train_data['DepDelay'] >= 60).astype(int)
validation_data['Significant_DepDelay'] = (validation_data['DepDelay'] >= 60).astype
test_data['Significant_DepDelay'] = (test_data['DepDelay'] >= 60).astype(int)
```

```
X_train_log_reg = X_train_lin_reg.copy()
y_train_log_reg = train_data['Significant_DepDelay']
model = sm.Logit(y_train_log_reg, X_train_log_reg)
result = model.fit_regularized(method='l1', alpha=0.01)
print(result.summary())
```

Flights_Before_Same_Airline	0.0106	0.003	4.109
Avg_DepDelay_Before	0.0098	0.002	6.277
Avg_DepDelay_Before_Same_Airline	0.0058	0.001	5.698
Average_Wind_Speed	-0.0032	0.007	-0.462
Precipitation	-0.2783	0.103	-2.702
Maximum_Temperature	0.0014	0.005	0.274
Minimum_Temperature	0.0016	0.006	0.268
Direction_Fastest_5sec_Wind	-7.775e-05	0.000	-0.213
DayOfWeek_Friday	-0.0986	0.075	-1.317
DayOfWeek_Saturday	0.0298	0.087	0.344
DayOfWeek_Sunday	-0.0741	0.077	-0.960
DayOfWeek_Thursday	-0.2108	0.078	-2.703
DayOfWeek_Tuesday	0.0404	0.083	0.488
DayOfWeek_Wednesday	-0.2531	0.083	-3.032
Airline_Alaska Airlines Inc.	0.2950	0.254	1.161
Airline_American Airlines Inc.	0.5244	0.229	2.292
Airline_American Eagle Airlines Inc.	-0.0331	0.306	-0.108
Airline_Frontier Airlines Inc.	0.3534	0.403	0.878
Airline_JetBlue Airways	0.0004	0.315	0.001
Airline_Skywest Airlines Inc.	-0.0803	0.235	-0.342
Airline_Southwest Airlines Co.	0.1435	0.234	0.615
Airline_US Airways Inc.	0.4878	0.270	1.806
Airline_United Air Lines Inc.	0.1169	0.227	0.515
Origin_SFO	0.3926	0.106	3.717

```
Month_5                                          -0.2007      0.088     -2.283
Month_6                                          -0.2037      0.105     -1.943
==============================================================================
/usr/local/lib/python3.10/dist-packages/statsmodels/base/l1_solvers_common.py:71
Try increasing solver accuracy or number of iterations, decreasing alpha, or swi
  warnings.warn(message, ConvergenceWarning)
/usr/local/lib/python3.10/dist-packages/statsmodels/base/l1_solvers_common.py:14
  warnings.warn(msg, ConvergenceWarning)
```

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score,

X_val_log_reg = validation_data.drop(['DepDelay', 'Significant_DepDelay'] + columns_
X_val_log_reg = sm.add_constant(X_val_log_reg)
y_val_log_reg = validation_data['Significant_DepDelay']
X_val_log_reg.fillna(0, inplace=True)
X_val_log_reg = X_val_log_reg.astype(float)

val_predictions_log_reg = result.predict(X_val_log_reg) > 0.5
accuracy_log_reg = accuracy_score(y_val_log_reg, val_predictions_log_reg)
precision_log_reg = precision_score(y_val_log_reg, val_predictions_log_reg)
recall_log_reg = recall_score(y_val_log_reg, val_predictions_log_reg)
f1_log_reg = f1_score(y_val_log_reg, val_predictions_log_reg)
conf_matrix_log_reg = confusion_matrix(y_val_log_reg, val_predictions_log_reg)

print(f"Accuracy: {accuracy_log_reg:.4f}")
print(f"Precision: {precision_log_reg:.4f}")
print(f"Recall: {recall_log_reg:.4f}")
print(f"F1 Score: {f1_log_reg:.4f}")
print("Confusion Matrix:")
print(conf_matrix_log_reg)
```

```
Accuracy: 0.7229
Precision: 0.6825
Recall: 0.4166
F1 Score: 0.5174
Confusion Matrix:
[[2327  280]
 [ 843  602]]
```

Classification Tree

```
from sklearn.tree import DecisionTreeClassifier, plot_tree

X_train_class_tree = train_data.drop(['DepDelay', 'Significant_DepDelay'], axis=1)
y_train_class_tree = train_data['Significant_DepDelay']

class_tree_model = DecisionTreeClassifier(random_state=0)
param_grid = {'ccp_alpha': np.linspace(0.0001, 0.01, 100)}
grid_search_class_tree = GridSearchCV(estimator=class_tree_model, param_grid=param_g
grid_search_class_tree.fit(X_train_class_tree, y_train_class_tree)

print("Best ccp_alpha:", grid_search_class_tree.best_params_['ccp_alpha'])
print("Best estimator:", grid_search_class_tree.best_estimator_)

feature_names = X_train_class_tree.columns
plt.figure(figsize=(20, 10))
plot_tree(grid_search_class_tree.best_estimator_, filled=True, feature_names=feature
plt.show()


X_val_class_tree = X_val_reg_tree.drop('Significant_DepDelay', axis=1)
y_val_class_tree = validation_data['Significant_DepDelay']
class_tree_predictions = grid_search_class_tree.best_estimator_.predict(X_val_class_

accuracy_class_tree = accuracy_score(y_val_class_tree, class_tree_predictions)
precision_class_tree = precision_score(y_val_class_tree, class_tree_predictions)
recall_class_tree = recall_score(y_val_class_tree, class_tree_predictions)
f1_class_tree = f1_score(y_val_class_tree, class_tree_predictions)
conf_matrix_class_tree = confusion_matrix(y_val_class_tree, class_tree_predictions)

print(f"Accuracy: {accuracy_class_tree:.4f}")
print(f"Precision: {precision_class_tree:.4f}")
print(f"Recall: {recall_class_tree:.4f}")
print(f"F1 Score: {f1_class_tree:.4f}")
print("Confusion Matrix:")
print(conf_matrix_class_tree)
```

```
X_test_class_tree = test_data.drop(['DepDelay', 'Significant_DepDelay'], axis=1)
y_test_class_tree = test_data['Significant_DepDelay']

test_tree_predictions = grid_search_class_tree.best_estimator_.predict(X_test_class_

accuracy_test_tree = accuracy_score(y_test_class_tree, test_tree_predictions)
precision_test_tree = precision_score(y_test_class_tree, test_tree_predictions)
recall_test_tree = recall_score(y_test_class_tree, test_tree_predictions)
f1_test_tree = f1_score(y_test_class_tree, test_tree_predictions)
conf_matrix_test_tree = confusion_matrix(y_test_class_tree, test_tree_predictions)

print(f"Accuracy: {accuracy_test_tree:.4f}")
print(f"Precision: {precision_test_tree:.4f}")
print(f"Recall: {recall_test_tree:.4f}")
print(f"F1 Score: {f1_test_tree:.4f}")
print("Confusion Matrix:")
print(conf_matrix_test_tree)
```

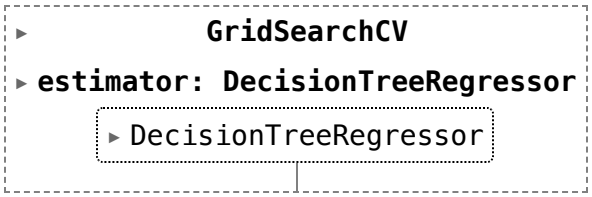
Double-click (or enter) to edit

Random Forest

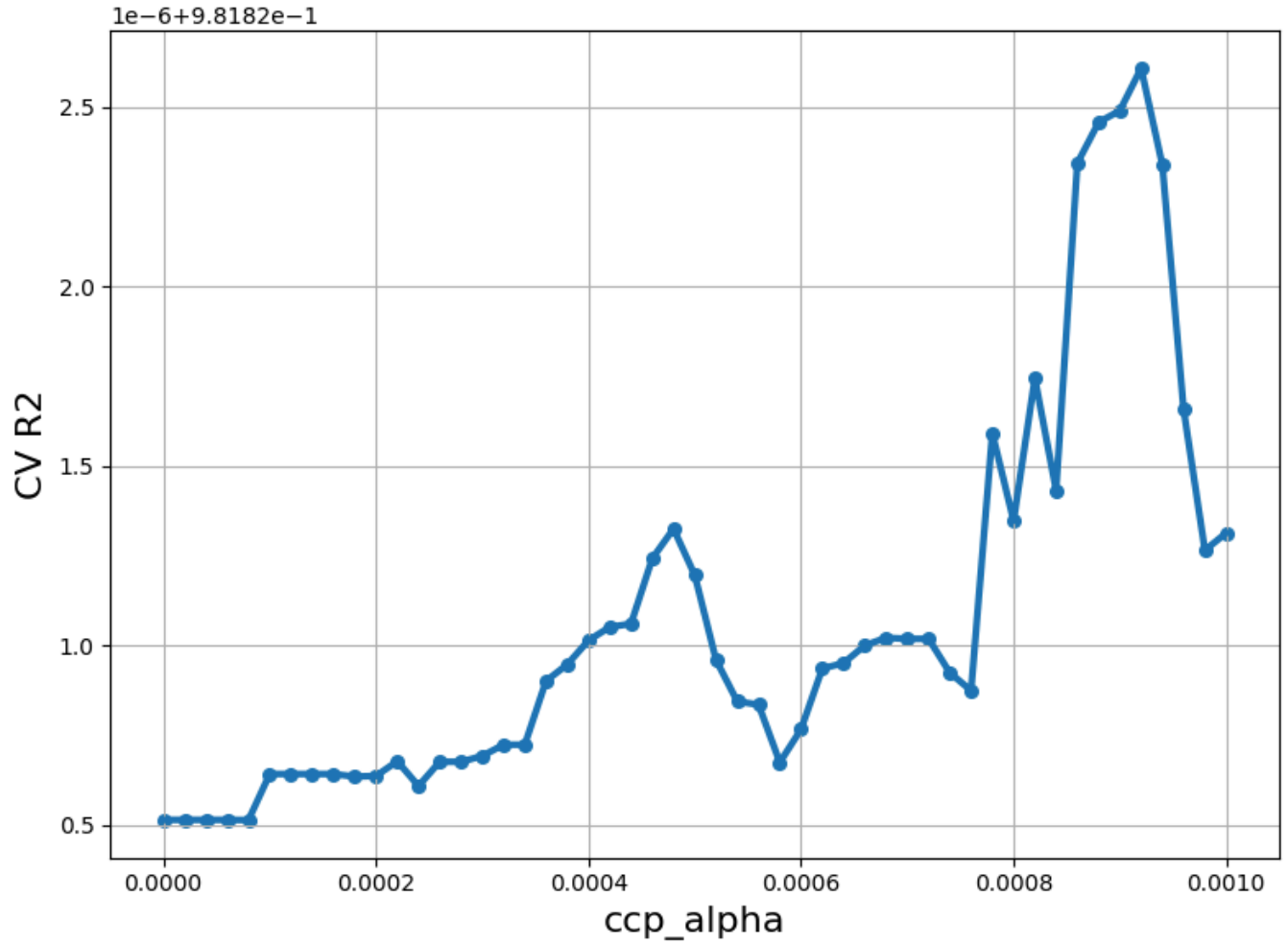
```
modeling_data.head()

X = pd.get_dummies(modeling_data.drop(['DepDelay'], axis=1))
y = modeling_data['DepDelay']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_stat
X_train.shape, X_test.shape
```

```
((14181, 70), (6078, 70))
```

```
from sklearn.model_selection import GridSearchCV
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import KFold


grid_values = {'ccp_alpha': np.linspace(0, 0.001, 51)}

dtr = DecisionTreeRegressor(min_samples_leaf=5, min_samples_split=20, random_state=8


cv = KFold(n_splits=5,random_state=1,shuffle=True)

dtr_cv = GridSearchCV(dtr, param_grid=grid_values, scoring='r2', cv=cv, verbose=0)
dtr_cv.fit(X_train, y_train)
```




```
ccp_alpha = dtr_cv.cv_results_['param_ccp_alpha'].data
R2_scores = dtr_cv.cv_results_['mean_test_score']

plt.figure(figsize=(8, 6))
plt.xlabel('ccp_alpha', fontsize=16)
plt.ylabel('CV R2', fontsize=16)
plt.scatter(ccp_alpha, R2_scores, s=30)
plt.plot(ccp_alpha, R2_scores, linewidth=3)
plt.grid(True, which='both')

plt.tight_layout()
plt.show()
print('Best ccp_alpha', dtr_cv.best_params_)
```




```
Best ccp_alpha {'ccp_alpha': 0.00092}
```

```
from sklearn.ensemble import RandomForestRegressor

rf = RandomForestRegressor(max_features=5, min_samples_leaf=5,
                           n_estimators = 500, random_state=88, verbose=2)

rf.fit(X_train, y_train)
```

Initialization

[ ] ↳ *3 cells hidden*

Feature Selection

[ ] ↳ *29 cells hidden*

Linear Regression Modeling

[ ] ↳ *2 cells hidden*

Regression Tree

[ ] ↳ *2 cells hidden*

Switching to Classification Problem

↳ *1 cell hidden*

Logistic Regression

[ ] ↳ *3 cells hidden*

Classification Tree

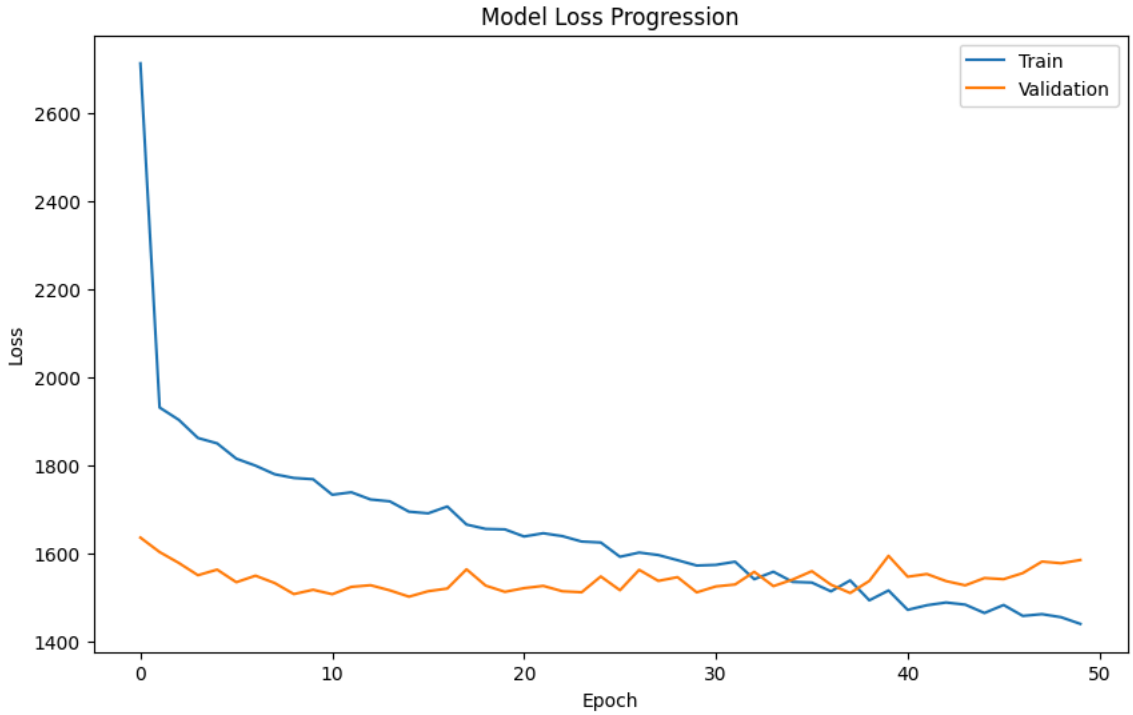
[ ] ↳ *3 cells hidden*

Neural Network

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

features = NN_train_data.drop(['DepDelay'], axis=1)
target = NN_train_data['DepDelay']

scaler = StandardScaler()
features_scaled = scaler.fit_transform(features)

X_train, X_val, y_train, y_val = train_test_split(features_scaled, target, test_size=0.2, random_state=42)
model = Sequential([
    Dense(128, activation='relu', input_shape=(X_train.shape[1],)),
    Dropout(0.2),
    Dense(64, activation='relu'),
    Dropout(0.2),
    Dense(32, activation='relu'),
    Dense(1)
])

model.compile(optimizer='adam', loss='mean_squared_error', metrics=['mean_absolute_error'])
history = model.fit(X_train, y_train, epochs=50, validation_data=(X_val, y_val), verbose=1)
val_loss, val_mae = model.evaluate(X_val, y_val, verbose=0)
print(f'Validation MAE: {val_mae:.4f}')
```

```
304/304 [==============================] - 1s 3ms/step - loss: 1595.1645 - mean_absolute_error: 27.4847 - val_loss: 1517.279
Epoch 27/50
304/304 [==============================] - 1s 3ms/step - loss: 1602.7435 - mean_absolute_error: 27.5683 - val_loss: 1563.511
Epoch 28/50
304/304 [==============================] - 1s 4ms/step - loss: 1597.0983 - mean_absolute_error: 27.6421 - val_loss: 1538.479
Epoch 29/50
304/304 [==============================] - 1s 4ms/step - loss: 1585.4150 - mean_absolute_error: 27.3965 - val_loss: 1546.948
Epoch 30/50
304/304 [==============================] - 1s 4ms/step - loss: 1573.3137 - mean_absolute_error: 27.4411 - val_loss: 1512.467
Epoch 31/50
304/304 [==============================] - 1s 4ms/step - loss: 1574.7830 - mean_absolute_error: 27.3021 - val_loss: 1525.890
Epoch 32/50
304/304 [==============================] - 1s 4ms/step - loss: 1581.9623 - mean_absolute_error: 27.1647 - val_loss: 1530.273
Epoch 33/50
304/304 [==============================] - 1s 3ms/step - loss: 1542.3278 - mean_absolute_error: 27.0055 - val_loss: 1559.025
Epoch 34/50
304/304 [==============================] - 1s 3ms/step - loss: 1559.1953 - mean_absolute_error: 27.2701 - val_loss: 1526.564
Epoch 35/50
304/304 [==============================] - 1s 2ms/step - loss: 1536.2029 - mean_absolute_error: 26.9869 - val_loss: 1541.476
Epoch 36/50
304/304 [==============================] - 1s 3ms/step - loss: 1534.6873 - mean_absolute_error: 26.9365 - val_loss: 1560.465
Epoch 37/50
304/304 [==============================] - 1s 2ms/step - loss: 1514.7477 - mean_absolute_error: 26.8610 - val_loss: 1529.834
Epoch 38/50
304/304 [==============================] - 1s 3ms/step - loss: 1539.4917 - mean_absolute_error: 26.9262 - val_loss: 1511.022
Epoch 39/50
304/304 [==============================] - 1s 3ms/step - loss: 1494.4004 - mean_absolute_error: 26.7246 - val_loss: 1538.371
Epoch 40/50
304/304 [==============================] - 1s 3ms/step - loss: 1516.7479 - mean_absolute_error: 26.8042 - val_loss: 1595.241
Epoch 41/50
304/304 [==============================] - 1s 3ms/step - loss: 1472.8102 - mean_absolute_error: 26.4517 - val_loss: 1547.883
Epoch 42/50
304/304 [==============================] - 1s 3ms/step - loss: 1483.2092 - mean_absolute_error: 26.5214 - val_loss: 1554.069
Epoch 43/50
304/304 [==============================] - 1s 3ms/step - loss: 1489.4879 - mean_absolute_error: 26.6507 - val_loss: 1538.007
Epoch 44/50
304/304 [==============================] - 1s 3ms/step - loss: 1484.6022 - mean_absolute_error: 26.4809 - val_loss: 1528.365
Epoch 45/50
304/304 [==============================] - 1s 4ms/step - loss: 1465.6346 - mean_absolute_error: 26.5249 - val_loss: 1544.878
Epoch 46/50
304/304 [==============================] - 1s 4ms/step - loss: 1483.8267 - mean_absolute_error: 26.3787 - val_loss: 1542.336
Epoch 47/50
304/304 [==============================] - 1s 5ms/step - loss: 1459.0809 - mean_absolute_error: 26.3014 - val_loss: 1556.157
Epoch 48/50
304/304 [==============================] - 1s 4ms/step - loss: 1463.0443 - mean_absolute_error: 26.3306 - val_loss: 1582.230
Epoch 49/50
304/304 [==============================] - 1s 3ms/step - loss: 1456.0455 - mean_absolute_error: 26.3197 - val_loss: 1578.496
Epoch 50/50
304/304 [==============================] - 1s 3ms/step - loss: 1440.9905 - mean_absolute_error: 26.0611 - val_loss: 1585.857
Validation MAE: 27.6242
```

```
plt.figure(figsize=(10, 6))
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Model Loss Progression')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper right')
plt.show()
```


Model Loss Progression
Train
Validation
Loss
Epoch
2600
2400
2200
2000
1800
1600
1400
0
10
20
30
40
50