CS 548 Assignment 4

Prediction of Airline Delays

Dennis Hofmann, Avantika Shrestha, Isaac Zhao, Riddhi Thakkar, Edith Gomez

04/28/22

Contents

List of Figures

1. Introduction

1.1 Background

Everyday hundreds of millions people in the U.S. rely on public transportation to commute to work, visit family or friends, travel, and transport items. More specifically, the air transportation sub-sector provides passengers the opportunity to reach farther destinations in a shorter amount of time. It is estimated that almost 3 million people fly in and out of U.S. airports a day [1]. Although flying is very convenient, the elements required for a successful flight must accommodate certain conditions. For example, a flight cannot proceed at its scheduled time if the airplane has any mechanical failure, or if there is a tornado in the departure/arrival city. These strict flying conditions lead to numerous flight time delays or even cancellations. According to the Bureau of Transportation Statistics, in 2021, 16.85% of flights inside the United States were delayed, while 1.72% were canceled; summing up to 1.1 million flights in one year [2]. These delays and cancellations cause frustration among customers and result in airlines losing profit. Every minute of delay costs airlines about $78 [3]. Other broad problems in this industry include:

- High air traffic and airport congestion, which also affect the schedules to be unchanged.
- Fuel efficiency and cost: an economic factor that affects the prices of flights directly.
- Passengers comfort and overall experience: since air transportation is in the service sector, it is critical for airlines to maintain passengers satisfied with their services, creating a high competition within airlines.
- Climate change: aircrafts are responsible for a considerable percentage of the global carbon emissions, making it a challenge for the industry to come up with solutions for their environmental impact.

1.2 Motivation

Our goal for this project is to accurately predict if a flight will be delayed based on the data provided by the airports and weather reports in the area. Our analysis uses historical data of domestic flights [4] in the U.S. along with the weather report for the departure date, departure location, arrival date, and arrival location [5]. By creating a model that can accurately predict if a flight is delayed, airlines can notify customers beforehand to prevent frustration and loss of profit. In addition to training an accurate model, some other questions airlines would be interested in having answers for are "Which airport locations are more prone to suffer flight delays?", "Does the weather variables of a location affect the schedule of flights?", and "How much distance on average is covered on different days throughout the year?". These questions will drive decisions from airlines to optimize flight schedules and customer satisfaction. In the following sections, we will thoroughly explain our analysis and present our results.

2. Description of the Dataset

For this project we combined multiple datasets from different sources since no single dataset in particular had all the information to accurately predict a flight's delay.

2.1 Airline Reporting Carrier On-Time Performance Dataset

The Bureau of Transportation Statistics of the United States has published the Reporting Carrier On-Time Performance Dataset, which contains US domestic flights between 1987 and 2020 [4]. Approximately 200 million domestic US flights reported to the United States Bureau of Transportation Statistics. The dataset contains basic information about each flight (such as date, time, departure airport, arrival airport) and, if applicable, the amount of time the flight was delayed and information about the reason for the delay. The complete dataset takes up 81 GB of memory when downloaded. The variables utilized for this project are presented in the following table. All other variables have been removed due to missing data, redundancy, or irrelevant information to our task.

Feature	Description
Year	Year
Month	Month
DayofMonth	Day of Month
DayOfWeek	Day of Week (numeric)
FlightDate	Date of Flight
Flight_Number_Reporting_Airline	Flight Number
OriginCityName	Origin City Name
OriginState	Origin State
DestCityName	Destination City Name
DestState	Destination State
CRSArrTime	Computer Reservation System (scheduled) Arrival Time
CRSDepTime	Computer Reservation System (scheduled) Departure Time
ArrDelay	Arrival delay (minutes)
Diverted	1 = diverted
Distance	Distance between airports (miles)

Figure 1: Variables considered in Airline Dataset.

2.2 Daily Weather Data

This dataset contains a daily summary of the weather conditions from countries that are members of the World Meteorological Organization (WMO) [5]. The National Center of Environmental Information gathered historical data from 1929 to the present, with data from 1973 to the present being the most complete. The data summaries provided here are based on data exchanged under the World Meteorological Organization (WMO) World Weather Watch Program according to WMO Resolution 40 (Cg-XII) [5]. This dataset allows for us to understand the weather at any given location in the U.S. for any day within the 1929 to the present. The dataset is conformed by the following variables.

Feature	Description
NAME	Airport name, state, and country
COORDINATES	Longitude and latitude
TEMP	Mean temperature (.1 Fahrenheit)
DEWP	Mean dew point (.1 Fahrenheit)
SLP	Mean sea level pressure (.1 mb)
STP	Mean station pressure (.1 mb)
VISIB	Mean visibility (.1 miles)
WDSP	Mean wind speed (.1 knots)
MXSPD	Maximum sustained wind speed (.1 knots)
GUST	Maximum wind gust (.1 knots)
MAX	Maximum temperature (.1 Fahrenheit)
MIN	Minimum temperature (.1 Fahrenheit)
PRCP	Precipitation amount (.01 inches)
SNDP	Snow depth (.1 inches)
FRSHTT	Indicator for occurrence of: Fog, Rain or Drizzle, Snow or Ice Pellets, Hail, Thunder, Tornado/Funnel Cloud

Figure 2: Variables considered in Weather Dataset.

Variable	n_missing	pct_missing
GUST	7826989	46.83
SLP	7159430	42.84
VISIB	2332891	13.96
DEWP	1909346	11.42
MXSPD	949957	5.68
WDSP	813186	4.87
CITY	33048	0.20
MAX	16372	0.10
MIN	7467	0.04

Figure 3: Percent of missing values in weather predictors. Gust and sea level pressure were removed due to high percent of missing values.

2.3 Airport Location Dataset

With this dataset [6] information about the location for each airport within the United States was obtained. Each row in this dataset represents the record for a single airport. The complete dataset was obtained from the OurAirports website, containing the following variables.

Feature	Description
name	The official airport name, including "Airport", "Airstrip", etc.
latitude_deg	The airport latitude in decimal degrees (positive for north).
longitude_deg	The airport longitude in decimal degrees (positive for east).
iso_country	The two-character ISO 3166:1-alpha2 code for the country where the airport is (primarily) located. A handful of unofficial, non-ISO codes are also in use, such as "XK" for Kosovo. Points to the code column in countries.csv.
iso_region	An alphanumeric code for the high-level administrative subdivision of a country where the airport is primarily located (e.g. province, governorate), prefixed by the ISO2 country code and a hyphen. OurAirports uses ISO 3166:2 codes whenever possible, preferring higher administrative levels, but also includes some custom codes. See the documentation for regions.csv.

Figure 4: Variables considered in Airport Location Dataset.

NAME.x	LATITUDE	LONGITUDE	LATITUDE2	LONGITUDE2	NAME.y	STATE	CITY
ALTOONA BLAIR CO AIRPORT, PA US	40.29639	78.32028	40.29640	78.32000	Altoona Blair County Airport	PA	Altoona
ALTURAS MUNICIPAL AIRPORT, CA US	41.49139	120.56444	41.48300	120.56500	Alturas Municipal Airport	CA	Alturas
ALTUS AFB, OK US	34.65000	99.26667	34.66710	99.26670	Altus Air Force Base	OK	Altus
ALVA REGIONAL AIRPORT, OK US	36.77306	98.66972	36.77320	98.66990	Alva Regional Airport	OK	Alva
AMARILLO AIRPORT, TX US	35.22950	101.70420	35.21940	101.70600	Rick Husband Amarillo International Airport	TX	Amarillo
AMBLER AIRPORT, AK US	67.10000	157.85000	67.10630	157.85699	Ambler Airport	AK	Ambler
AMELIA LAKE PALOURD, LA US	29.70000	91.10000	29.69330	91.09870	Lake Palourde Base Heliport	LA	Amelia
AMERADA PASS, LA US	29.45000	91.33333	29.66330	91.24070	Berwick Shore Base Heliport	LA	Berwick
AMES MUNICIPAL AIRPORT, IA US	41.99056	93.61889	41.99200	93.62180	Ames Municipal Airport	IA	Ames
ANAKTUVUK AUTO, AK US	68.16667	151.76667	68.13360	151.74300	Anaktuvuk Pass Airport	AK	Anaktuvuk Pass
ANCHORAGE ELMENDORF AFB, AK US	61.25306	149.79361	61.25100	149.80701	Elmendorf Air Force Base	AK	Anchorage

Figure 5: Results of combining Daily Weather Dataset and Airport Location Datasets with different Airport names and different coordinates. Matched by euclidian distance for nearest neighbor.

3. Data Cleaning

3.1 Airline Reporting Carrier On-Time Performance Dataset

We started by first determining which variables to remove and found lots of redundant variables. For each airport, we had the airport id, airport sequence id, airport market id, airport fips code, airport wac code, etc. that conveyed essentially the same information. There were other variables that contributed directly to the delay outcome we wanted to predict, so it was very important to remove them otherwise we would end up with perfect accuracy. For example, departure delay time is calculated by the scheduled departure time - the actual departure time. Departure delay time is also calculated by CarrierDelay, WeatherDelay, NASDelay, SecurityDelay, LateAircraftDelay.

Further, we found that some of the values would be misleading if they were left as it is, and would run into trouble while dealing with them later. For instance, the "Scheduled Departure Time" was originally presented in xxxx format, where the first two digits represented the hour and the last two digits represented the minutes. This means that 10:59 would be represented as 1059 and 11:00 would be represented as 1100. We would quickly run into a problem while performing any kind of operations with this time format because while calculating the difference between 1059 and 1100 we would end up with 41 whereas it actually should be 1. To solve this problem, we converted the given time to minutes after midnight.

Taking into consideration the fact that for this project we were only interested in predicting whether or not there would be a delay in a particular flight, we removed all the records of canceled flights as they did not have information about departure or arrival delay times.

We also observed that many different variables almost entirely comprised of missing values such as information about the flights that were originally supposed to land in a particular airport but were diverted to a different airport due to weather conditions or traffic. We removed these variables as well.

One issue we encountered while working with this dataset was that due to its size, Turing would often time out due to memory issues. We counteracted this problem by utilizing pandas chunksize parameter in the read_csv function. This parameter determines how many rows are read in one iteration and it iterates through the dataset. The object returned is then combined together using the concat function. We also used the usecols parameter to select only the attributes we wanted.

```
chunksize = 100000
tfr = pd.read_csv('airline.csv', usecols=['Year', 'Month', 'DayofMonth', 'DayOfWeek',
↪  'FlightDate', 'Flight_Number_Reporting_Airline', 'OriginCityName', 'OriginState',
↪  'DestCityName', 'DestState', 'CRSDepTime',  'DepDelay',   'TaxiOut',  'WheelsOff',
↪  'WheelsOn', 'TaxiIn', 'CRSArrTime', 'ArrDelay', 'Diverted',
 'CRSElapsedTime', 'ActualElapsedTime', 'AirTime', 'Distance', 'Cancelled'],
 ↪  chunksize=chunksize, iterator=True, encoding='latin-1')
df = pd.concat(tfr, ignore_index=True)
```

For our purposes, we have established that if a flight has a difference of departure and scheduled time greater than 15 minutes or a difference of arrival and scheduled time greater than 15 minutes, then it is going to be considered as delayed (two classes for our target variable, delayed and non-delayed). There are rare cases where a flight departs before the scheduled time, these were considered for the purposes of this analysis, as on-time flights. Additional to flight information, this project includes weather data that provides characteristics of the temperature, rainfall, snow, etc on the days and locations of the considered flights.

3.2 Daily Weather Dataset

This dataset was given in one large zip file with a folder corresponding to each year since 1987. Within each file, there were thousands of individual csv files that corresponded to individual airport station data of flights that arrived and departed for that year. The airport stations were international, but our project scope was limited to domestic flights. Since the station names (sequence of numbers) were not useful for distinguishing stations within or outside of the USA, we would have to combine all files for all years then filter for the US using the NAME variable.

That would become too cumbersome, so instead we decided to row bind all the csv files for one recent year. After that all the records for the USA were extracted and then saved the station names. For our further analysis, station names that we wanted to be read in were specified, i.e, the US station names. By filtering out only the files that were required, we were able to save significant computational costs. Then all the columns that were relevant were picked and tossed out the corresponding variables with "_ATTRIBUTES " in their name that each column had as it represented the number of observations used in calculating that variable which was useless.

Once this was done, the NAME variable was split into Airport name, state and country. It was crucial in this step to make sure that the split was done correctly and it made sense, and the state and country

were extracted out of it with no mistakes. Then we went on to check for missing values and got rid of all the weather predictors with a lot of missing values. For instance, we removed GUST which had 46.83% missing values and SLP which had 42.84% missing values. After that, the variable codes from FRSHTT were converted to making 6 new variables using one-hot encoding (0=no, 1=yes): Clear, Fog, Rain, Snow, Hail, Thunder, Tornado. Results were then verified and the conversion was validated. For example, a record with FRSHTT = 110010 should correspond to fog = 1, rain = 1, and thunder = 1, and 0 for the rest.

The re-coded values consisted of missing values. Some variables like TEMP, DEWP represented missing as 9999.9, while some others represent missing as 99.99, 999.9, etc. It was alright to impute 0 in some variables such as PRCP and SNDP that, based on the documentation, means none was present. After re-coding, we looked at summary stats (min, median, max, etc.) to see if the values after re-coding made sense or not. We found inconsistencies in the reported longitude and latitude. Normally, the format for reporting coordinates is latitude, longitude. But comparing some records with what google shows, it was found that the values were actually longitude, latitude. Furthermore, some values were negative for some of the longitudes and latitudes for US stations, which is impossible since all US longitudes and latitudes are positive so we converted them to absolute values to better match the true coordinates.

3.3 Airport Location Dataset

For this dataset, we decided it would be best to split iso_region into country and state as it would be easier to join the datasets if state was its own variable. Again, we came across inconsistencies in the reported longitude and latitude. These were resolved by again converted longitude and latitude to absolute values, since the negative values for US stations were causing the same problem

3.4 Joining Daily Weather Dataset with Airport Location Dataset

In order to perform our experiments, we needed to first join the daily weather dataset with our airline dataset. This is where we run into some issues. The airline dataset only had the airport city but not the airport name while the daily weather dataset has the airport name available but not the city. Thus, we join the weather dataset with a mediary airport location dataset in order to obtain the city for the weather dataset. The airport location dataset provided us with latitude and longitude, which is also present in the weather dataset.

A challenge with this was that the airport names in the weather dataset and airport location dataset did not match. For example, one dataset says ALTUS AFB, while the other says Altus Air Force Base. As the two datasets did not have matching airport names. Therefore, we had to rely on the longitude and latitude values in order to join the dataset. However, we face another issue of the longitude and latitude in the weather dataset not matching the values in the airport location dataset. Our solution to this problem was to use both sets of longitude and latitude to calculate the euclidean distance between each record. We then joined the datasets using the smallest euclidean distance between records. In order to do this, we used the matchpt function in R to get the closest nearest neighbor for each airport station based on euclidean distance.

3.5 Joining Airline Performance Dataset with Daily Weather Dataset + Airport Location Dataset

After joining the airport location dataset to the weather dataset, the new dataset now had an attribute for city. This was required for us to join the weather dataset to the airline performance dataset. Now, using the city attribute we were able to join the new dataset to the airline performance dataset.

Since the airline dataset contains departures and arrivals we must consider the weather at both arrival and departure locations. This requires us to conduct 2 joins. First, we joined the weather dataset with the airport locations (from the last step) with the airlines dataset on date, departure city, and departure state. We then repeated this process but instead of joining on departure city and departure state we used arrival city and arrival state. This way for each record (flight) we have the weather at the departing airport along with that of the arrival airport.

```
full_df_origin = pd.merge(weather_df, airline_df, left_on=["DATE", "CITY", "STATE"],
↪  right_on=["Date", "OriginCityName", "OriginState"], how='inner')

full_df = pd.merge(weather_df, full_df_origin, left_on=["DATE", "CITY", "STATE"],
↪  right_on=["Date", "DestCityName", "DestState"], how='inner')
```

4. Methods

We chose to use Random Forest as our model since Random Forest is typically faster, performs better, and is more interpretable than other models. Since we are using a relatively large dataset, and are on a timeline, we required a faster model. In addition, our work is meant for airline companies and therefore we aim to have our results easily understood by a non-data science expert. Taking these requirements into consideration Random Forest was the clear choice.

4.1 Methods: Model Tuning

There were a few factors we had to take into consideration when training the Random Forest model. The first was the class imbalance in our target value and the second was tuning our model.

In our dataset, our target variable consisted of two classes (whether the flight was delayed or not). The non-delayed class represented about 80% of our data, and the delayed class represented about 20%. To tackle this class imbalance issue we resorted to undersampling the non-delayed class as undersampling would decrease our runtime to train our model. We were interested in finding the optimal percent to undersample the class but due to the size of our dataset we resorted to 4 runs with varying unsersmaling percentages. We trained an off the shelf Random Forest model with 100%, 75%, 50% and 25% of the non-delayed data while keeping the size of the delayed class constant. Doing so, we could observe the F1 score, accuracy, and time for each undersample size. The results can be seen in the chart below.

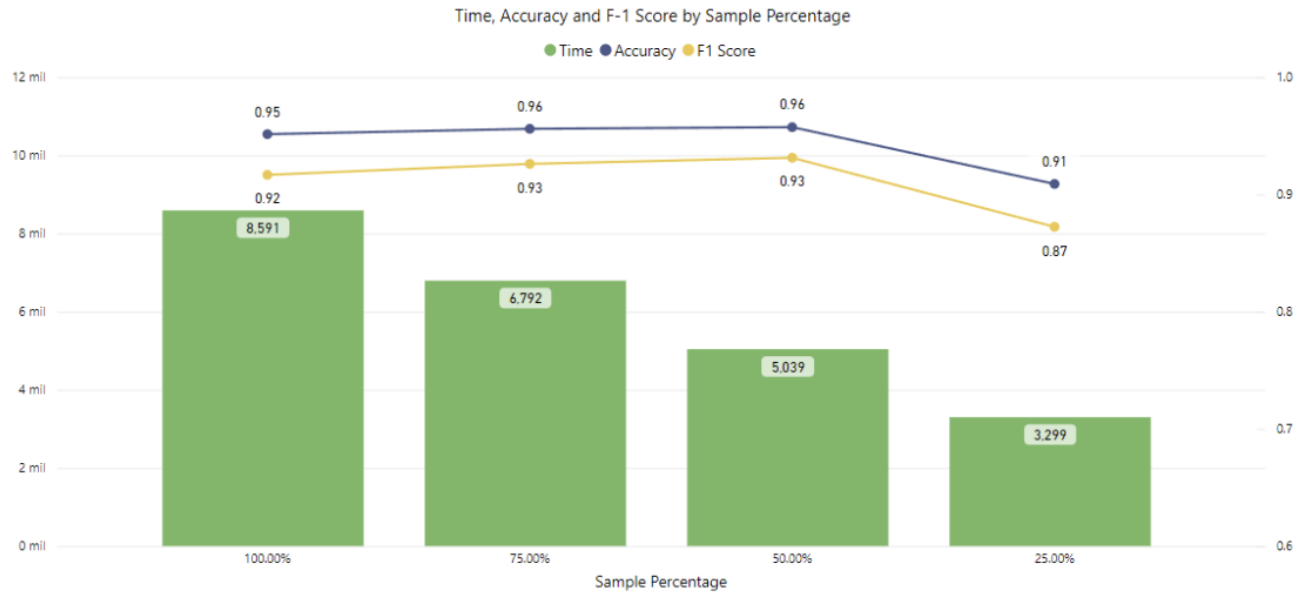

Figure 6: Percent of class undersampling while training Random Forest Model vs performance.

Using the undersampling experiments we conducted, we found that undersampling the non-delayed class by

50% provides the highest performance while also reducing the run time to train our model.

Now that we have improved the class imbalance issue, we could tune our Random Forest model. To tune our model, we leveraged Scikit-Learn's RandomizedSearchCV package. This allowed us to create a grid of different hyperparameter values, and then at training randomly sample from the grid to tune the Random Forest model. Using RandomizedSearchCV frees us from running each hyperparameter combination, which would be infeasible with the size of our dataset. We repeated the sampling process 15 times and compared the performance of each tuned model. The Random Forest hyperparameters we included in our grid are seen in the table below.

Hyperparamiter	**Meaning**	**Values**
n_estimators	Number of trees	50, 100, 150, 200, 250, 300, 350, 400, 450, 500
max_features	number of features to consider when looking for the best split	'auto', 'sqrt', 'log2'
max_depth	The maximum depth of the tree	20, 40, 60, 80, 90, 100, None
min_samples_split	The minimum number of samples required to split an internal node	2, 3, 4, 5, 6
criterion	Function to measure the quality of the split	'gini', 'entropy'

Figure 7: Table of hyperparameters tuned for training Random Forest model.

4.2 Methods: Model Evaluation

To evaluate our models we leveraged accuracy, and F1 score. Accuracy was used since our work should be easily interpretable for non-data science experts and F1 score was used to provide a better metric to consider the class imbalance issue with our target variable.

5. Results

5.1 Exploratory Analysis

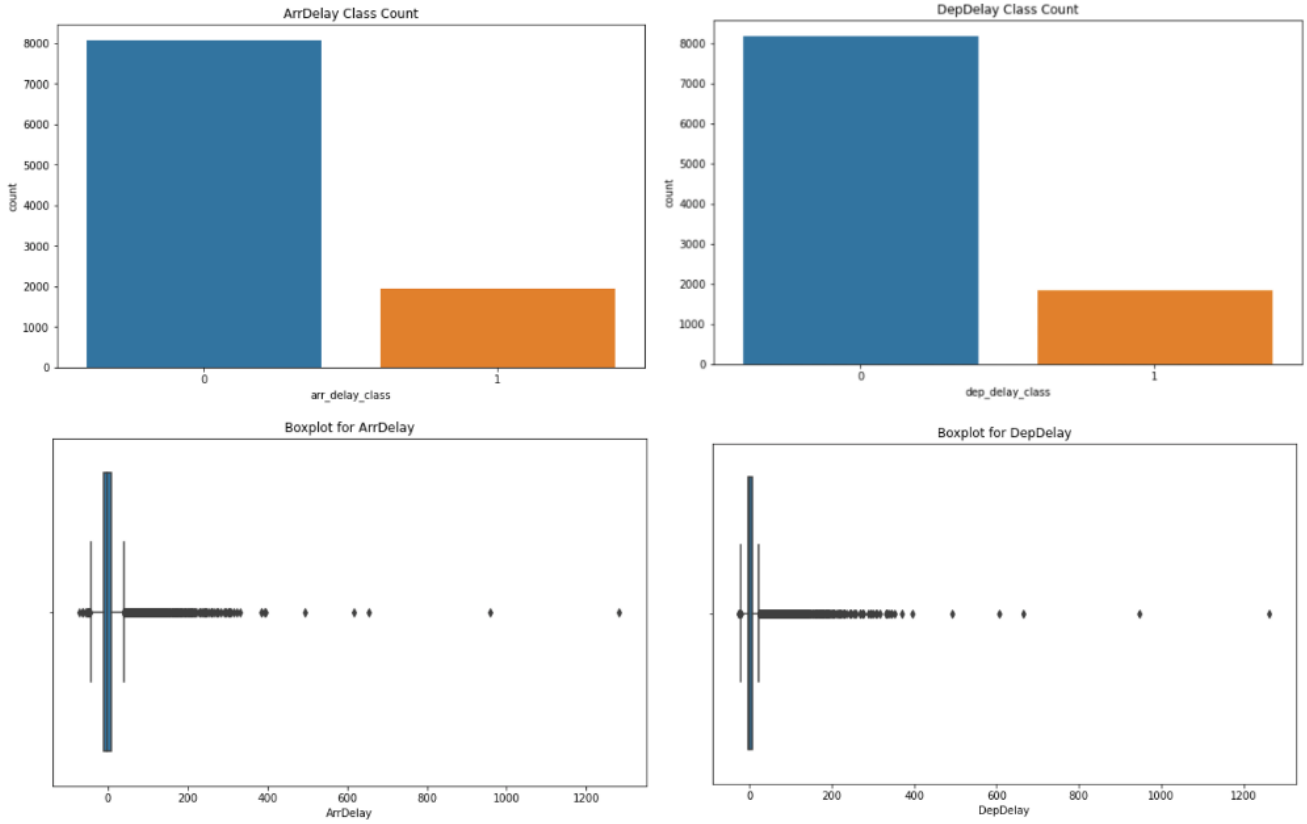

Figure 8: Distribution of Arrival and Departure Delays, where cutoff for delay is >15 min.

With the graph above we observe that there is an imbalance between the two considered classes, delayed flights and on-time flights. Additionally, the distribution of the arrival delays and departure delays have little variation, with multiple observations above the 4th quartile. Further exploration is needed to define these cases as outliers or not.

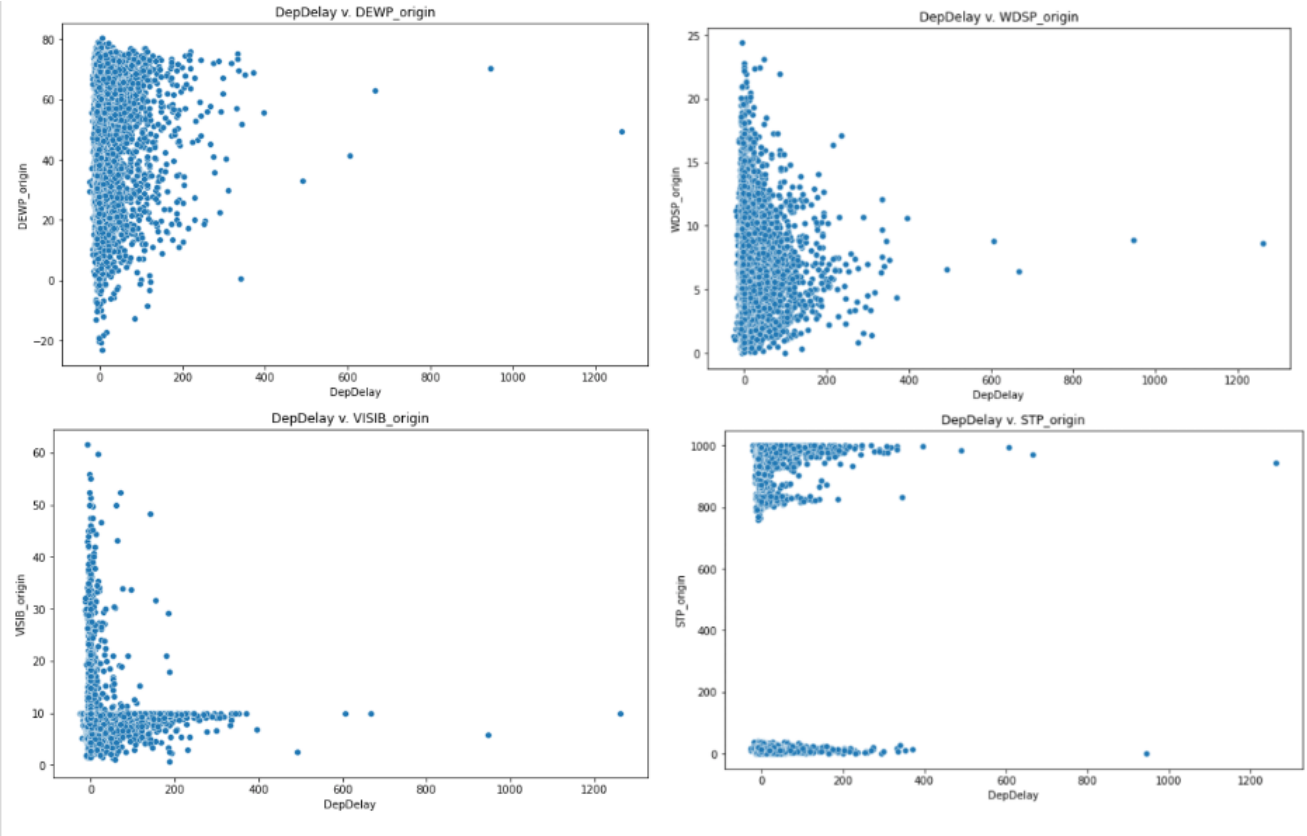

Figure 9: Correlation between weather predictors and delay time.

In the scatter plots above, we can see the relation between departure delay and the mean dew point at the origin airport, the mean wind speed at the origin airport, the mean visibility at the origin airport and the mean station pressure at the origin airport (from left to right, top to bottom). From our analysis, we can see that as the mean dew point increases, there seems to be slightly more delays. There also seems to be a relationship between visibility and delays as more delay points cluster around lower visibility. Another relation we can explore is the relation between mean wind speed and delays, as it seems that lower wind speeds relate to more delays. We further explore our variables later on using a feature importance graph.

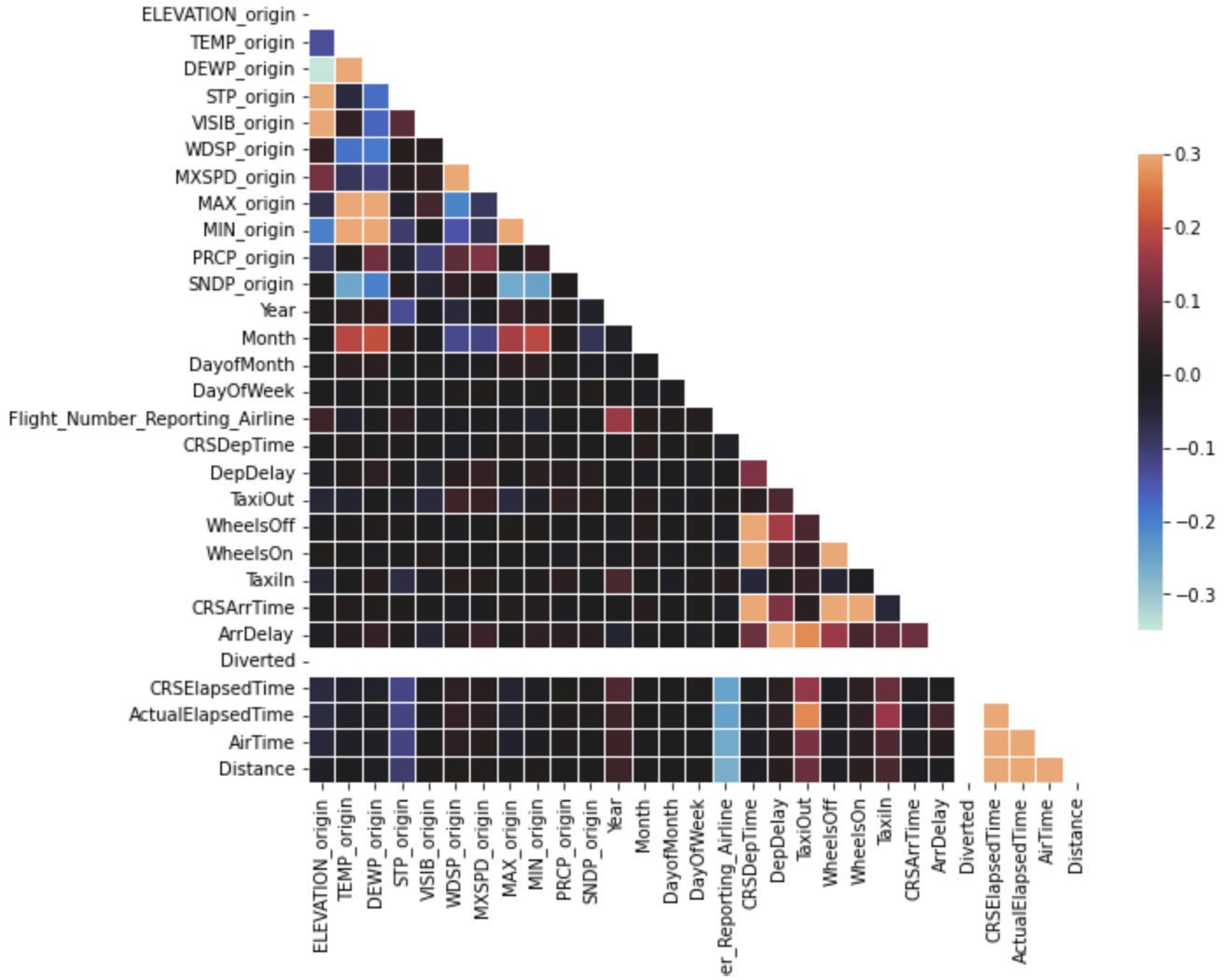

Figure 10: Correlation between all variables.

Here we see that temperature, dew point, pressure, visibility, precipitation, and wind speed are all correlated with one another.

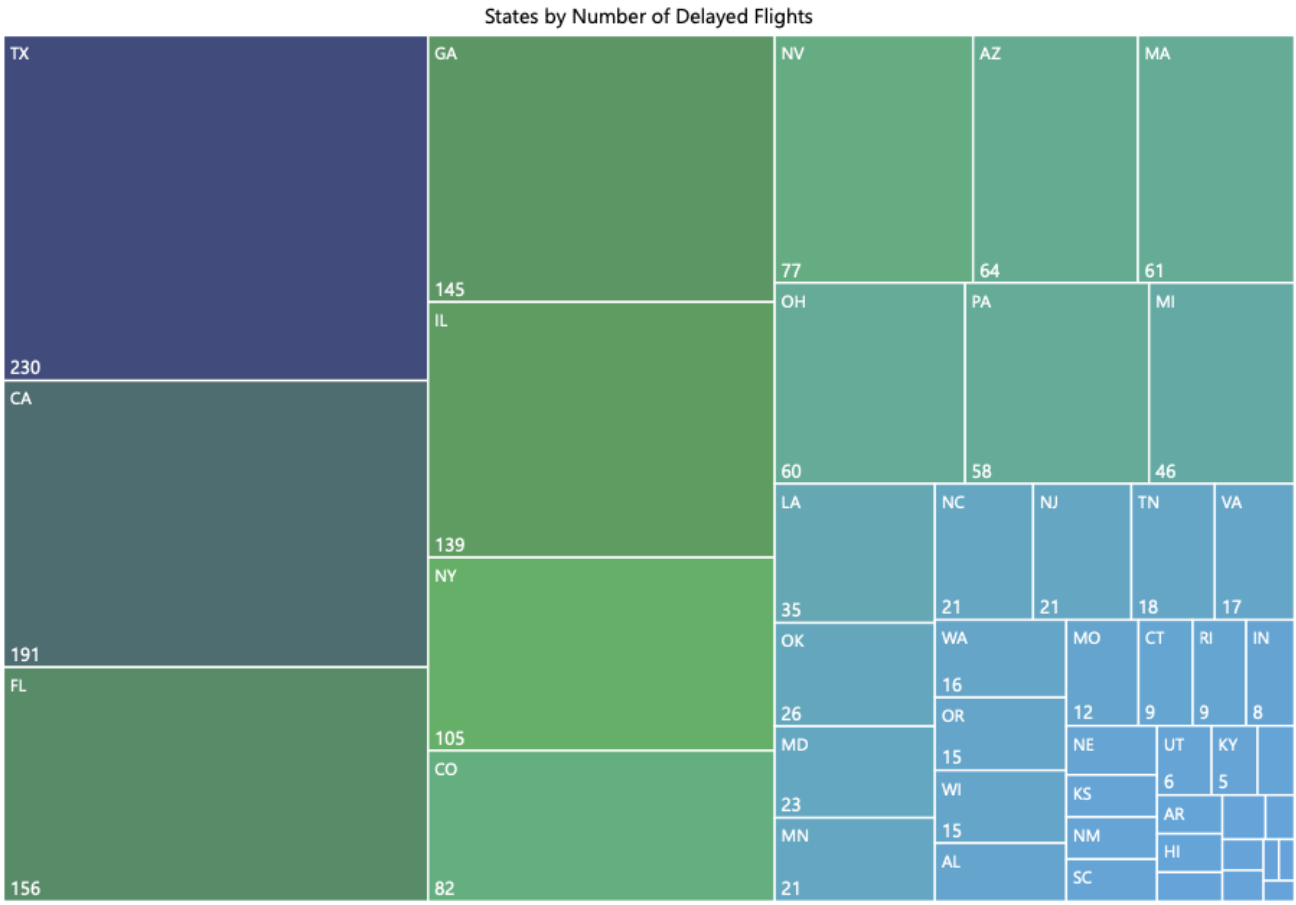

Figure 11: Number of delays by state.

As we can see in the chart above, the particular States that have the highest number of delayed flights are Texas, California and Florida, followed by Georgia, Illinois and New York. Although the most affected states are not located on the east coast, the following states are; meaning that the east region is more affected by flight delays more often; a conjecture that was also observed in previous graphs.

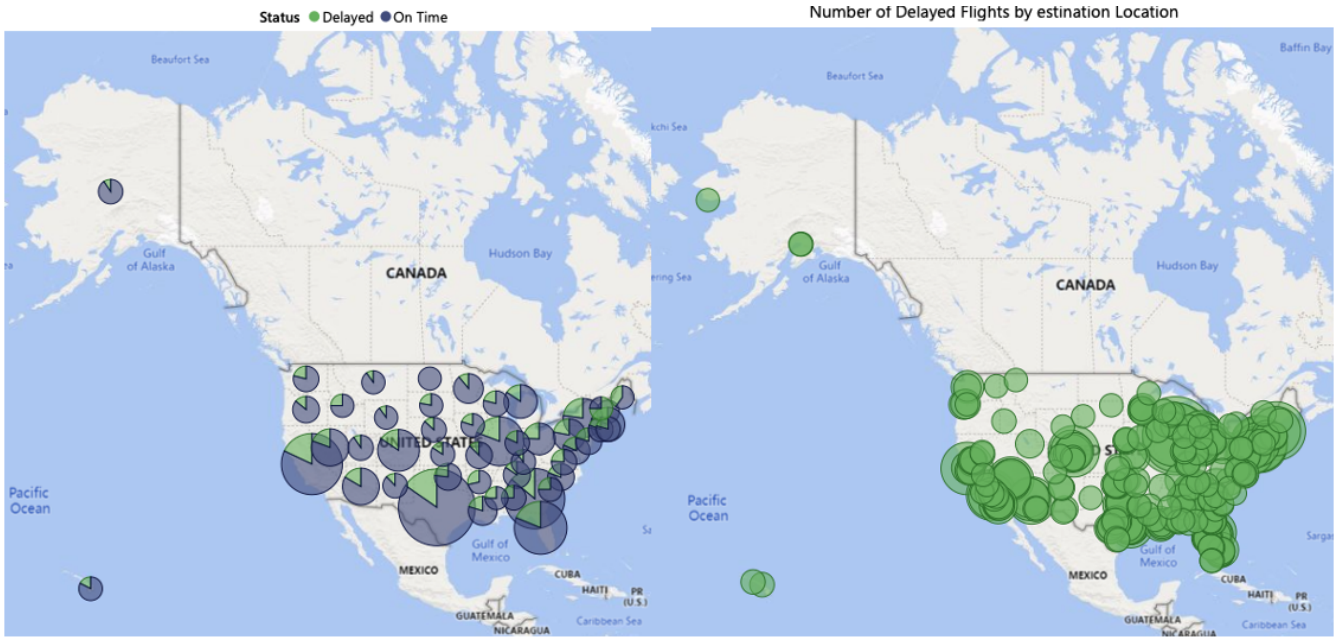

Figure 12: Percent (left) and number (right) of delays by location.

By looking at the flights by destination location and status, it is clear to see that all states within the United States experience flight delays no matter the location. However, the east coast has a higher number of incidents than the rest of the regions in the country. At the same time, by looking at the map on the left side, we can see that for all states, the number of on-time flights surpassees the number of delayed flights, which is a good signal on the correct functioning of the commercial airlines.

Figure 13: Visibility by date.

The Average Visibility by Date line chart shows us how the visibility of domestic flights within the U.S. have

developed over time. We observe there is no significant visibility difference between the delayed flights and the on-time flights, making us suspect that this variable may not be a good indicator to predict the status class of a flight.

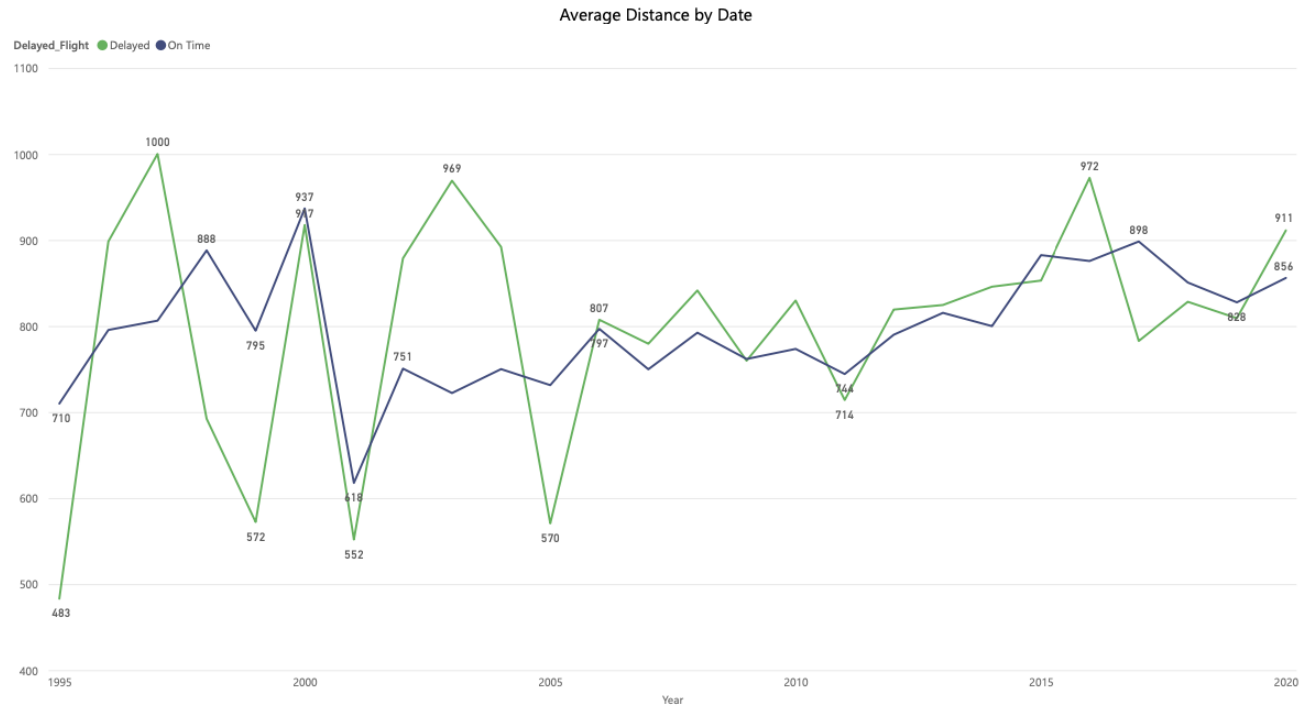

Figure 14: Average flight distance by date.

Based on the graphs above, we can say that the average distance traveled with the flights have increased with time since before 2005 the average had more variance. And if we analyze the delayed flights vs the on-time flights, there was a higher distance gap that could justify the reason for the delays suffered. However, after 2005 the distance difference reduced between the two classes, making us think that the importance of the distance traveled affecting the departure or arrival actual time reduced significantly.

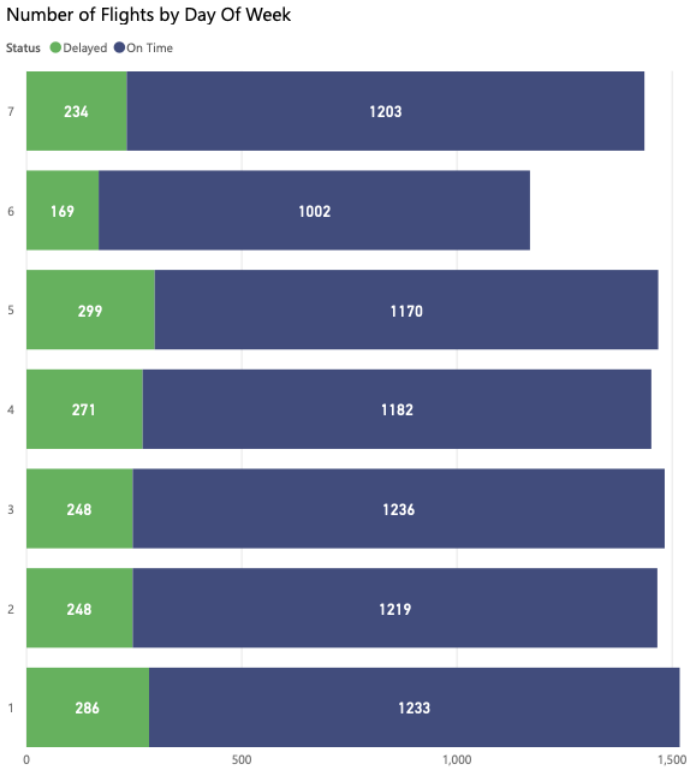

Figure 15: Number of flight delays per day. 1=Mon, 2=Tues, etc.

With the chart above we see the number of flights by day of the week, in the color green we have the delayed flights, and in blue we observe the on-time flights. At first glance we can see that the day that has the least number of flights is saturday, while the rest of the days have a similar amount of flights. Across all weekdays, the distribution of on-time and delayed flights remain similar.

5.2 Random Forest Performance

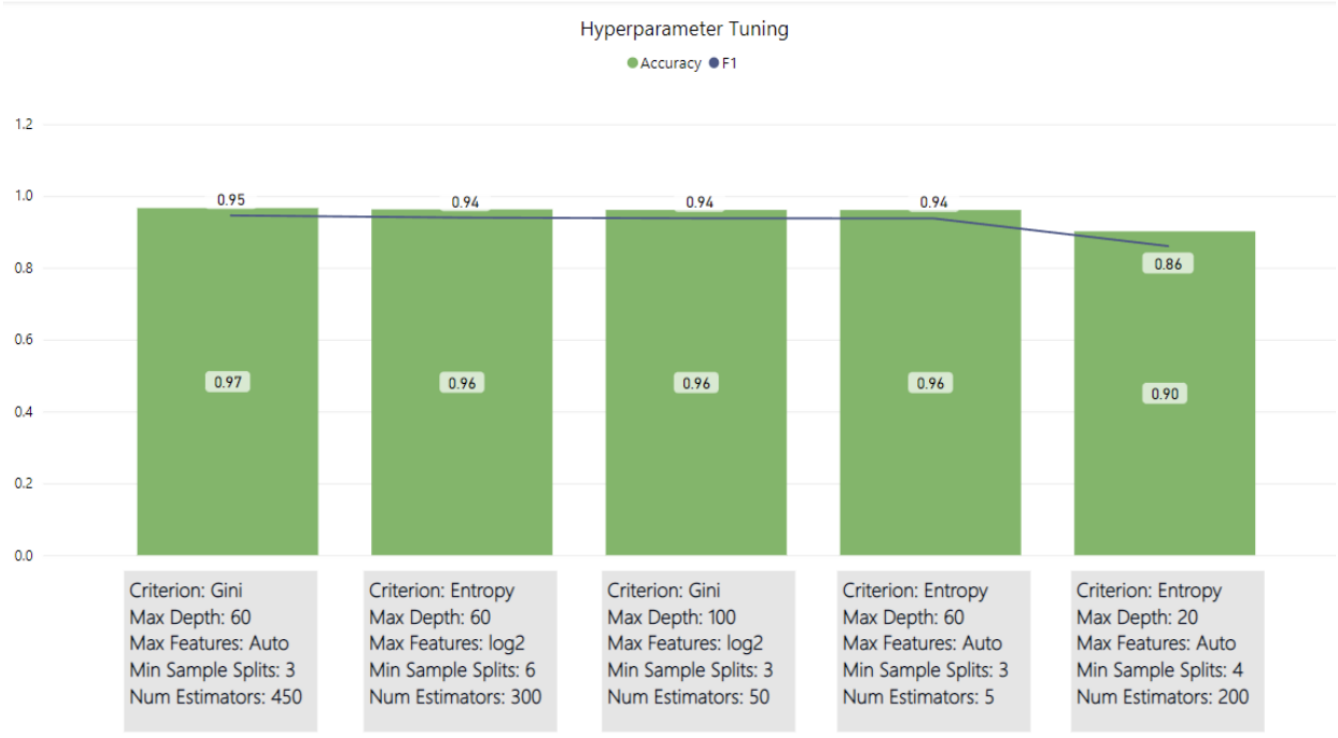

Figure 16: Performance of Random Forest predicting Delay using different hyperparameters.

After conducting the 15 runs with various hyperparameter configurations, we found the best hyperparameter combinations to be n_estimators: 450, min_samples_split: 3, max_features: auto, max_depth: 60, and criterion: gini. These hyperparameter configurations returned an accuracy of 0.97 and an F1 score of 0.95. This is a .02 increase of the F1 score from the off the shelf model. In the hyperparameter tuning chart above, we show the best combinations of parameters for each 5 runs. Each run samples from the grid 3 times so in total we have 15 runs. We also produced a feature importance chart to provide insights on which variables are important in predicting if a flight was delayed.

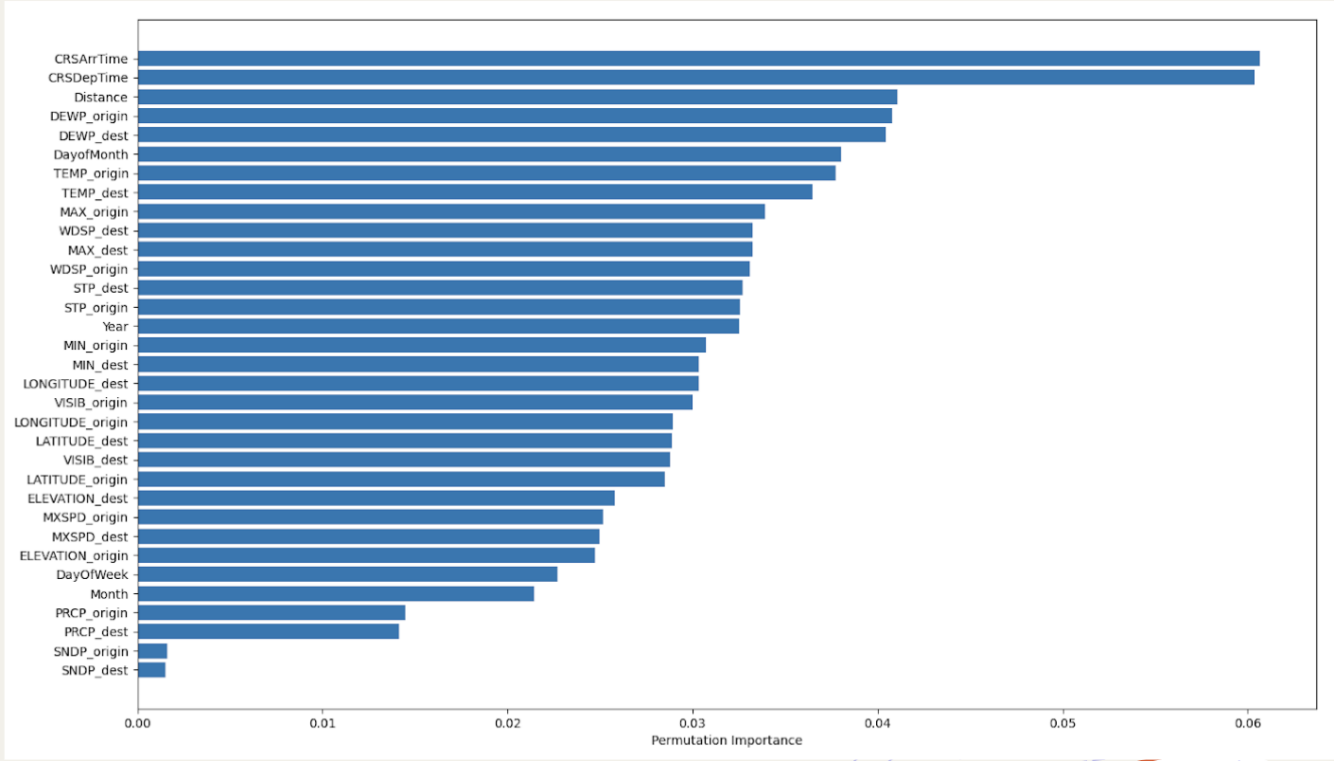

Figure 17: Variable Importance plot of best performing Random Forest model.

The variable importance plot tells us that the most important variables that affect the probability of flight delays are time of flight, distance of flight, dew point, and temperature.

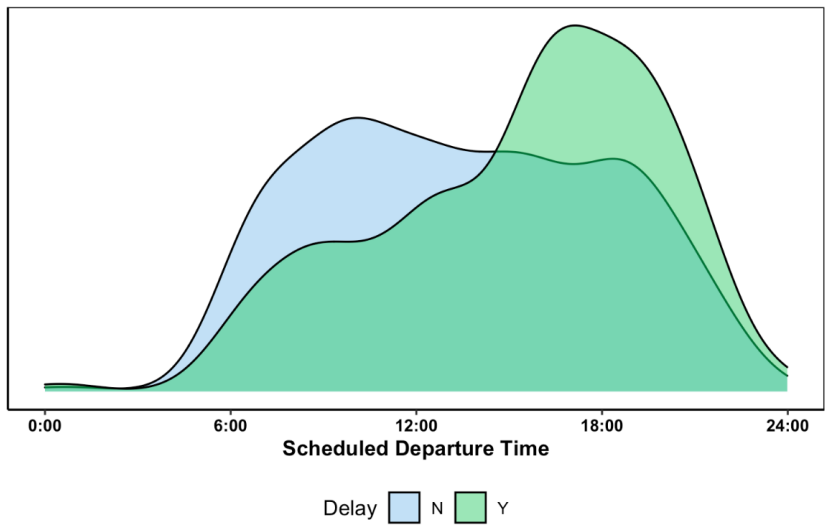

Figure 18: Distribution of departure delay vs non delay flights by departure time.

We can see that more delayed flights occur during later departure times, perhaps due to the fact that weather affects visibility much more at night time and sudden drop in temperature which could lead to unsafe conditions.

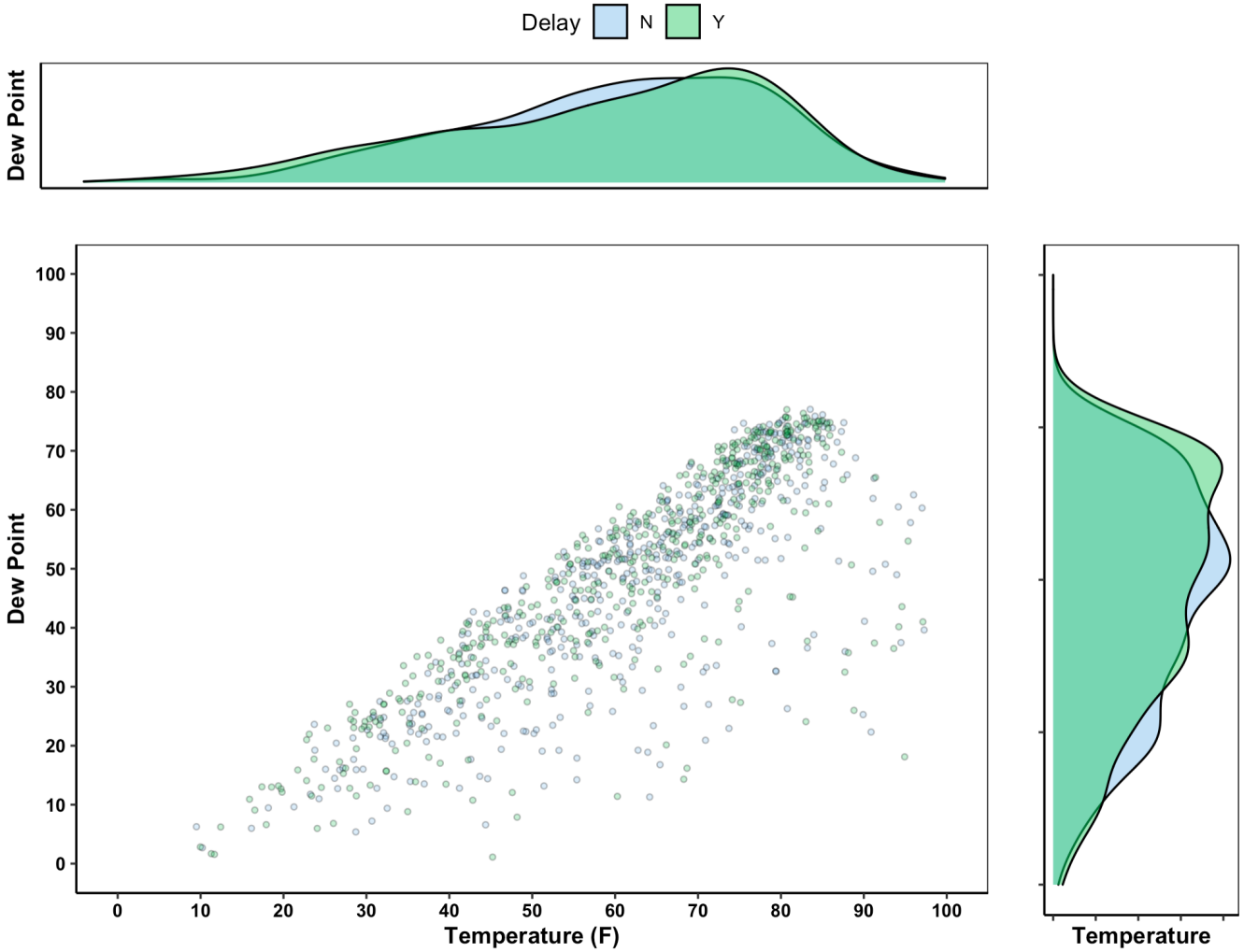

Figure 19: Distribution of departure delay vs non delay flights by dew point and temperature.

Although the variable importance tells us that dew point and temperature are the top few most important predictors for delay, the scatterplot is hard to visually distinguish differences. Although, the combination of high dew point and high temperature leads to more fronts and unstable air masses. High dew points are also correlated with a higher probability and severity of thunderstorms, which would make sense in this context.

6. Challenges

For us to be able to work as a team and divide up jobs, we had to be easily able to transfer files and load them in fast. After each teammate finished their part in data manipulation, we saved the results as a tar.gz or .RDS (R file format for compression) to save space and read speed so that other team members could work off that manipulated dataset.

We spent a lot of time searching for relevant datasets that could be joined together. We had to search through many sources to find exact formats that must meet criteria of having data from at least the year 1987 to 2020, and have data for airport stations per day. Data cleaning took most of the time for our project since we had to use three separate datasets from different sources, and each source had different formats for airport name, city name, etc.

Once we obtained our data, the biggest challenge we had was figuring out how to join them together. As

discussed in detail in the data cleaning section, we discovered that the airport dataset only had the airport city but did not have airport name, while the weather dataset had airport name, latitude, longitude, but no city. We had to find a third dataset with information about airport name, latitude, longitude, and city so that the weather dataset could be joined with the airport dataset by city. Since the weather dataset and the third dataset did not have matching airport names nor matching latitude or longitude, we had to find the airports from both datasets with the closest euclidean distance by considering all possible pairs.

Another big challenge we had was reading in large data into the Turing Cluster. The cluster would reach memory limits just loading in the data. Our solution was to specify specific columns to select and read in rows in chunks during the read csv phase. We also found it hard to work with the cluster for data cleaning since every little step for manipulation and checking results had to be submitted to the cluster. Instead, we took advantage of the 1 percent dataset provided to write scripts to clean data, run diagnostics, and make sure the results are what was desired before submitting to the cluster.

7. Conclusions

Using our combined dataset of airline data, weather data and airport location data, we were successfully able to design a model that can predict whether or not a flight will be delayed. We achieved an accuracy of 0.97 and an F1 score of 0.95 using our tuned model, which gave us an increase of 2% in our F1 score from using an off-the-shelf model.

We faced many challenges during the process, especially during the data cleaning process. We struggled with memory issues due to the size of our dataset. However, we were able to overcome these challenges.

For future work, the variable importance can play a stronger role in identifying the problematic factors that affect our study such as if certain airline companies, airports, days, or flight times are commonly involved with delays. This dataset can also be used to train a regression model instead of a classification model in future directions in order to predict the exact amount of delay an airline would suffer as delay times range from 15 mins to at most 10 hours. Webscarping could also be an interesting direction, as we can webscrape real time flight and weather data, scheduled up to a week in the future and weather forecasts for those dates. This can be utilized to run models and notify passengers, airlines and travel booking agencies of potentially delayed flights. In addition, due to the size of our dataset and limited timeframe, we were only able to run one model (Random Forest). Going forward we would like to experiment with other models such as Logistic regression, SVM, and some deep learning models.

8. Team Member's Contributions

For this project, the team chose to assign different tasks to each one of the members. The project was divided into multiple steps, some of them could be done simultaneously, others were in a sequence. This strategy was chosen so that the work-flow kept its curse. In the early stages, all team members gathered to discuss and look at the multiple possible datasets that could be used for this project. After deliberation, the project theme and dataset were settled and the execution of the project began.

After the previous steps, the project tasks were divided as follows: Isaac first worked on the data cleaning and preprocessing needed to join the three datasets used in this project. During this process, Avantika also took part in the cleaning process, accelerating the pace of this step. When these steps were finalized, it was time to perform some exploratory data analysis; this task was performed by Edith and Avantika, which created a series of informative graphs that gave more insights of the distribution of the data. With the gathered information, Dennis and Riddhi proceed to create the code necessary to fit the Machine Learning model chosen by the team.

The next step consisted of tuning the Random Forest Model used to predict the flight delay, as well as evaluating this model. This part of the project was in charge of Dennis as well. At the same time, the outputs of the model, like F-1 score, running time and accuracy were combined into a single file to create a consolidated graph to visualize the performance of the model, a task that was done by Edith.

While the hyperparameters of the model were being tuned, Edith, Isaac and Riddhi started working on the final report to save time before the final due date. This helped the rest of the team members focus on the coding part of the project. After all previous steps were finalized and verified, a final team meeting was scheduled and all members collaborated together to finalize the report, while Riddhi prepared the presentation slides.

Throughout the entire project, communication played a major role in the team's dynamic, every time one of the team members faced an obstacle, the rest of the group members were open to discussion to find a solution to the current problem. This made the project a positive experience for everyone.

9. References

[1] "Chart of the day: Airlines losing $78/minute today?," CNBC, 27-Nov-2013. [Online]. Available: https://www.cnbc.com/2013/11/27/chart-of-the-day-airlines-losing-78minute-today.html. [Accessed: 27-Apr-2022].

[2] US Bureau of Transportation Statistics. (n.d.). On-Time Performance - Reporting Operating Carrier Flight Delays at a Glance [Online]. Available: https://www.transtats.bts.gov/HomeDrillChart.asp. [Accessed: 26-Apr-2022].

[3] Air traffic by the numbers, 18-Mar-2022. [Online]. Available: https://www.faa.gov/air_traffic/by_the_numbers/. [Accessed: 27-Apr-2022].

[4] US Bureau of Transportation Statistics, "Airline Reporting Carrier On-Time Performance Dataset," IBM developer, 25-Jun-2020. [Online]. Available: https://developer.ibm.com/exchanges/data/all/airline/?mhsrc=ibmsearch_a&mhq=+delay. [Accessed: 27-Apr-2022].

[5] NCEI, "Global Summary of the Day," National Centers for Environmental Information (NCEI). [Online]. Available: https://www.ncei.noaa.gov/access/search/data-search/global-summary-of-the-day. [Accessed: 27-Apr-2022].

[6] "OurAirports dataset formats," Dataset formats @ OurAirports. [Online]. Available: https://ourairports.com/help/data-dictionary.html. [Accessed: 27-Apr-2022].

10. Appendix

10.1 Cleaning Airport Data

```
# 1 % data csv is 880 mb. 100% data is 88
# gb

df_read = as.data.frame(read_csv("Data/airline.csv")) %>%
    filter(Cancelled == 0)

df_clean = df_read %>% select(-c(OriginAirportSeqID,
    OriginCityMarketID, OriginAirportID,
    DestAirportID, IATA_CODE_Reporting_Airline,
    DestCityMarketID, Tail_Number, Reporting_Airline,
    Quarter, DOT_ID_Reporting_Airline, DestAirportSeqID,
    Origin, OriginStateName, OriginStateFips,
    OriginWac, DepDel15, DepartureDelayGroups,
    ArrDel15, ArrivalDelayGroups, Dest, DestStateName,
    DestStateFips, DestWac, DepDelayMinutes,
    DepTimeBlk, ArrDelayMinutes, ArrTimeBlk,
    CancellationCode, Flights, DistanceGroup,
```

```
    LongestAddGTime, DepTime, ArrTime, Cancelled,
    FirstDepTime, TotalAddGTime, DivAirportLandings,
    CarrierDelay, WeatherDelay, NASDelay,
    SecurityDelay, LateAircraftDelay, DivReachedDest,
    DivActualElapsedTime, DivArrDelay, DivDistance,
    Div1Airport, Div1AirportID, Div1AirportSeqID,
    Div1WheelsOn, Div1TotalGTime, Div1LongestGTime,
    Div1WheelsOff, Div1TailNum, Div2Airport,
    Div2AirportID, Div2AirportSeqID, Div2WheelsOn,
    Div2TotalGTime, Div2LongestGTime, Div2WheelsOff,
    Div2TailNum, Div3Airport, Div3AirportID,
    Div3AirportSeqID, Div3WheelsOn, Div3TotalGTime,
    Div3LongestGTime, Div3WheelsOff, Div3TailNum,
    Div4Airport, Div4AirportID, Div4AirportSeqID,
    Div4WheelsOn, Div4TotalGTime, Div4LongestGTime,
    Div4WheelsOff, Div4TailNum, Div5Airport,
    Div5AirportID, Div5AirportSeqID, Div5WheelsOn,
    Div5TotalGTime, Div5LongestGTime, Div5WheelsOff,
    Div5TailNum)) %>% mutate(Flight_Number_Reporting_Airline =
↪   as.character(Flight_Number_Reporting_Airline)) %>%
    rename(DATE = FlightDate)

saveRDS(df_clean, file = "Data/airline_clean.RDS")
```

10.2 Get US Airport Station Codes

```
year = "2015"


all_files = list.files(year)
# 12100 csvs get only USA stations
i = 1
combined_csv = data.frame()
for (i in 1:length(all_files)) {
    current_file = all_files[i]
    current_csv = as.data.frame(read_csv(paste0(year,
        "/", current_file), show_col_types = FALSE,
        col_select = c("STATION", "NAME",
            "LATITUDE", "LONGITUDE")))
    combined_csv = rbind(combined_csv, current_csv)
    print(paste0("year ", year, ": file ",
        i, " out of ", length(all_files)))
}
saveRDS(combined_csv, "~/Desktop/WPI/Classes/CS_548/Assignments/Assignment
↪   4/Data/Station_Airport.RDS")
combined_csv = readRDS("~/Desktop/WPI/Classes/CS_548/Assignments/Assignment
↪   4/Data/Station_Airport.RDS")
test2 = as.data.frame(read_csv(paste0(2015,
    "/72040800136.csv"), show_col_types = FALSE))


combined_csv_us = combined_csv %>% separate(col = "NAME",
```

```
    sep = ", ", into = c("Airport", "State_Country")) %>%
    filter(grepl(" US", State_Country)) %>%
    separate(col = "State_Country", sep = " ",
        into = c("State", "Country"))  #%>%
# mutate(Airport = gsub('AIRPORT'))

rm(combined_csv)
test = combined_csv_us %>% group_by(State,
    Airport) %>% slice(1)

saveRDS(combined_csv_us, "~/Desktop/WPI/Classes/CS_548/Assignments/Assignment
↪  4/Data/Station_Airport_US.RDS")
write_csv(combined_csv_us, "~/Desktop/WPI/Classes/CS_548/Assignments/Assignment
↪  4/Data/Station_Airport_US.csv")


paste0(unique(combined_csv_us$STATION), collapse = "', '")
```

10.3 Combining Weather Data

```
# csv files of just us stations
us_stations_df = readRDS("Station_Airport_US.RDS")
us_stations = unique(us_stations_df$STATION)

# location to save combined files
save_loc = "Raw_Weather"

all_years = 1987:2020

for (j in 1:length(all_years)) {
    year = all_years[j]
    # all files in year folder
    all_files = list.files(year)
    # all files in folder that match given US
    # station files
    matching_files = intersect(all_files,
        paste0(us_stations, ".csv"))
    print(paste0("year ", year, ": matching files in USA: ",
        length(matching_files), " out of 2703"))

    # combine all csvs for given year
    combined_csv = data.frame()
    for (i in 1:length(matching_files)) {
        current_file = matching_files[i]
        current_csv = as.data.frame(read_csv(paste0(year,
            "/", current_file), show_col_types = FALSE))
        combined_csv = rbind(combined_csv,
            current_csv)
        print(paste0("year ", year, ": file ",
            i, " out of ", length(matching_files)))
    }
    saveRDS(combined_csv, paste0(save_loc,
```

```
        "/", year, ".RDS"))
}
```

10.4 Cleaning Weather Data

```
all_years = 1987:2020

i = 1
# combine all weather years together
combined_df = data.frame()
for (i in 1:length(all_years)) {
    current_year = all_years[i]
    current_file = readRDS(paste0("Data/Raw_Weather/",
        current_year, ".RDS"))
    combined_df = rbind(combined_df, current_file)
    print(paste0("year ", current_year, ": file ",
        i, " out of ", length(all_years)))
}
saveRDS(combined_df, "Data/Raw_Weather/All_Years_Raw.RDS")

weather_df_read = readRDS("Data/Raw_Weather/All_Years_Raw.RDS") %>%
    mutate(LATITUDE = abs(as.numeric(LATITUDE)),
        LONGITUDE = abs(as.numeric(LONGITUDE)))


# split variables and fix latitude and
# longitude
airport_lat_long = as.data.frame(read_csv("Data/airport_codes.csv")) %>%
    filter(iso_country == "US") %>% separate(iso_region,
    into = c("Country", "State"), sep = "-") %>%
    select(name, State, municipality, coordinates) %>%
    rename(NAME = name, STATE = State, CITY = municipality) %>%
    separate(coordinates, into = c("LONGITUDE",
        "LATITUDE"), sep = ", ") %>% mutate(LATITUDE = abs(as.numeric(LATITUDE)),
    LONGITUDE = abs(as.numeric(LONGITUDE)))

# find closest matching latitude and
# longitude nearest neighbor
airport_lat_long_matrix = as.matrix(airport_lat_long %>%
    select(LATITUDE, LONGITUDE))

snap <- function(point) {
    d <- matchpt(airport_lat_long_matrix,
        as.matrix(data.frame(LATITUDE = point$LATITUDE +
            1e-04, LONGITUDE = point$LONGITUDE +
            1e-04)))  # to the 'northwest' criteria correct

    min_row <- as.numeric(rownames(d[d$distance ==
        min(d$distance), ]))

    LATITUDE_snap <- unique(airport_lat_long_matrix[min_row,
        "LATITUDE"])
```

```
    LONGITUDE_snap <- unique(airport_lat_long_matrix[min_row,
        "LONGITUDE"])

    return(c(LATITUDE_snap, LONGITUDE_snap))
}

all_airports_weather = weather_df_read %>%
    select(NAME, LATITUDE, LONGITUDE) %>%
    group_by_all() %>% slice(1) %>% ungroup() %>%
    as.data.frame()

i = 1
for (i in 1:nrow(all_airports_weather)) {
    current_row = all_airports_weather[i,
        ]
    closest_lat_long_match = snap(current_row)
    all_airports_weather$LATITUDE2[i] = closest_lat_long_match[1]
    all_airports_weather$LONGITUDE2[i] = closest_lat_long_match[2]

}

all_airports_weather = all_airports_weather %>%
    left_join(airport_lat_long, by = c(LATITUDE2 = "LATITUDE",
        LONGITUDE2 = "LONGITUDE"))


# convert missing value codes to missing,
# create one hot encoding
weather_df_clean = weather_df_read %>% mutate(TEMP = ifelse(TEMP ==
    9999.9, NA_real_, TEMP), DEWP = ifelse(DEWP ==
    9999.9, NA_real_, DEWP), SLP = ifelse(SLP ==
    9999.9, NA_real_, DEWP), STP = as.numeric(STP),
    STP = ifelse(STP == 9999.9, NA_real_,
        STP), VISIB = ifelse(VISIB == 999.9,
        NA_real_, VISIB), WDSP = ifelse(WDSP ==
        999.9, NA_real_, WDSP), MXSPD = ifelse(MXSPD ==
        999.9, NA_real_, MXSPD), GUST = ifelse(GUST ==
        999.9, NA_real_, GUST), MAX = ifelse(MAX ==
        9999.9, NA_real_, MAX), MIN = ifelse(MIN ==
        9999.9, NA_real_, MIN), PRCP = ifelse(PRCP ==
        99.99, 0, PRCP), SNDP = ifelse(SNDP ==
        999.9, 0, SNDP), Clear = ifelse(FRSHTT ==
        "000000", 1, 0), Fog = ifelse(substr(FRSHTT,
        1, 1) == "1", 1, 0), Rain = ifelse(substr(FRSHTT,
        2, 2) == "1", 1, 0), Snow = ifelse(substr(FRSHTT,
        3, 3) == "1", 1, 0), Hail = ifelse(substr(FRSHTT,
        4, 4) == "1", 1, 0), Thunder = ifelse(substr(FRSHTT,
        5, 5) == "1", 1, 0), Tornado = ifelse(substr(FRSHTT,
        6, 6) == "1", 1, 0), Clear = as.character(Clear),
    Fog = as.character(Fog), Rain = as.character(Rain),
    Snow = as.character(Snow), Hail = as.character(Hail),
    Thunder = as.character(Thunder), Tornado = as.character(Tornado)) %>%
    select(-FRSHTT) %>% left_join(all_airports_weather %>%
```

```
    select("NAME.x", "STATE", "CITY"), by = c(NAME = "NAME.x"))

saveRDS(weather_df_clean, "Data/weather_df_clean.RDS")
```

10.5 Join Datasets

```
import pandas as pd
import pyreadr

chunksize = 10000

##### Read in our data #####
print("Starting the joining process")
# weather data
result = pyreadr.read_r('weather_df_clean.RDS')
weather_df = result[None]
print("read in weather data")

# airline data
#airline_df = pd.read_csv('0.2airline.csv', encoding='latin-1', on_bad_lines='skip')
#print("read in airline data")
#############################################################################################

##### Clean data #####
# Airline Drop cancelled rows
# airline_df = airline_df[airline_df['Cancelled'] == 0]
# airline_df = airline_df.drop('Cancelled', axis=1)

# Weather drop GUST and SLP and name
weather_df = weather_df.drop('GUST', axis=1)
weather_df = weather_df.drop('SLP', axis=1)
weather_df = weather_df.drop('NAME', axis=1)


# Drop everything after comman including comma airline city names
# airline_df["OriginCityName"] = airline_df["OriginCityName"].replace(',.*$','',
↪  regex=True)
# airline_df["DestCityName"] = airline_df["DestCityName"].replace(',.*$','', regex=True)

# Change date cols to datatime objects
weather_df["DATE"] = pd.to_datetime(weather_df["DATE"], errors='coerce',
↪  infer_datetime_format=True)
# airline_df["Date"] = pd.to_datetime(airline_df["Date"], errors='coerce',
↪  infer_datetime_format=True)


#############################################################################################
##### Join our two datasets into one full one #####

i = 0
def preprocess(airline_df):
    global i
```

```
    # remove cancelled flights and airprot name
    airline_df = airline_df[airline_df['Cancelled'] == 0]
    airline_df = airline_df.drop('Cancelled', axis=1)

    # drop everything after commas
    airline_df["OriginCityName"] = airline_df["OriginCityName"].replace(',.*$','',
↪   regex=True)
    airline_df["DestCityName"] = airline_df["DestCityName"].replace(',.*$','',
↪   regex=True)

    # convert to datetime
    airline_df["Date"] = pd.to_datetime(airline_df["Date"], errors='coerce',
↪   infer_datetime_format=True)

    # join for origin
    full_df_origin = pd.merge(weather_df, airline_df, left_on=["DATE", "CITY", "STATE"],
↪   right_on=["Date", "OriginCityName", "OriginState"], how='inner')
    # rename weather columns for origin
    full_df_origin.rename(columns={'ELEVATION': 'ELEVATION_origin', 'TEMP':
↪   'TEMP_origin', 'DEWP': 'DEWP_origin', 'STP': 'STP_origin', 'VISIB': 'VISIB_origin',
↪   'WDSP': 'WDSP_origin',
    'MXSPD': 'MXSPD_origin', 'MAX': 'MAX_origin', 'MIN': 'MIN_origin', 'PRCP':
    ↪   'PRCP_origin', 'SNDP': 'SNDP_origin', 'LATITUDE': 'LATITUDE_origin', 'LONGITUDE':
    ↪   'LONGITUDE_origin',
    'Clear': 'Clear_origin', 'Fog': 'Fog_origin', 'Rain': 'Rain_origin', 'Snow':
    ↪   'Snow_origin', 'Hail': 'Hail_origin', 'Thunder': 'Thunder_origin', 'Tornado':
    ↪   'Tornada_origin'}, inplace=True)

    # join dest data
    full_df = pd.merge(weather_df, full_df_origin, left_on=["DATE", "CITY", "STATE"],
↪   right_on=["Date", "DestCityName", "DestState"], how='inner')
    # rename weather columns for dest
    full_df.rename(columns={'ELEVATION': 'ELEVATION_dest', 'TEMP': 'TEMP_dest', 'DEWP':
↪   'DEWP_dest', 'STP': 'STP_dest', 'VISIB': 'VISIB_dest', 'WDSP': 'WDSP_dest',
    'MXSPD': 'MXSPD_dest', 'MAX': 'MAX_dest', 'MIN': 'MIN_dest', 'PRCP': 'PRCP_dest',
    ↪   'SNDP': 'SNDP_dest', 'LATITUDE': 'LATITUDE_dest', 'LONGITUDE': 'LONGITUDE_dest',
    'Clear': 'Clear_dest', 'Fog': 'Fog_dest', 'Rain': 'Rain_dest', 'Snow': 'Snow_dest',
    ↪   'Hail': 'Hail_dest', 'Thunder': 'Thunder_dest', 'Tornado': 'Tornada_dest'},
    ↪   inplace=True)

    # on first run, add header. Else dont add header
    if i == 0:
        full_df.to_csv("full_df_2.csv",mode="a",header=True,index=False)
    else:
        full_df.to_csv("full_df_2.csv",mode="a",header=False,index=False)
    i = i + 1

reader = pd.read_csv('0.2airline.csv', encoding='latin-1', on_bad_lines='skip', chunksize
↪   = chunksize) # chunksize depends with you colsize

#temp_airline = pd.read_csv('0.2airline.csv', encoding='latin-1', on_bad_lines='skip',
↪   nrows=1) # chunksize depends with you colsize
```

```
#temp_header = pd.merge(weather_df, temp_airline, left_on=["CITY"],
↪  right_on=["DestState"], how='inner')  # incorrect join to just get headers
#temp_header.to_csv("full_df_2.csv",mode="a",header=True,index=False)  # add headers to
↪  CSV


[preprocess(r) for r in reader]  # we need to read in by chuck size since data takes up
↪  too much memory



#################################################################################################
##### Convert combined full dataset to compressed file types #####
#importing modules
from random import sample
import pyreadr
import pandas as pd

#reading in csv file
#samp_perc = 0.02
chunksize = 100000

tfr = pd.read_csv('full_df_2.csv', chunksize=chunksize, iterator=True,
↪  encoding='latin-1')

df = pd.concat(tfr, ignore_index=True)
#df = df.sample(n=round(len(df)*samp_perc), random_state=15)
#print(samp_perc, "of file is being used")
print("Read in data")

#saving as RDS file
pyreadr.write_rds('full_data.RDS', df, compress="gzip")
print('RDS dowloaded!')
```

10.6a Exploratory Analysis

```
df = readRDS("Data/full_data_clean_n10000.RDS")

df1 = df %>% select(CRSDepTime, DEWP_origin,
    TEMP_origin, DepDelay) %>% mutate(Delay = ifelse(DepDelay >=
    15, "Y", "N"))

df1_n = df1 %>% filter(Delay == "N") %>%
    slice(sample(1:n(), 500))
df1_y = df1 %>% filter(Delay == "Y") %>%
    slice(sample(1:n(), 500))
df2 = df1_n %>% bind_rows(df1_y)

library(gridExtra)
library(grid)

plot1 = ggplot(df2, aes(TEMP_origin)) + geom_density(aes(fill = Delay),
```

```
    alpha = 0.5) + mytheme + scale_x_continuous("",
    limits = c(0, 100)) + ylab("Dew Point") +
    theme(axis.text = element_blank(), axis.ticks = element_blank())

plot2 = grid.rect(gp = gpar(col = "white"))


plot3 = ggplot(df2, aes(TEMP_origin, DEWP_origin)) +
    geom_jitter(aes(fill = Delay), pch = 21,
        alpha = 0.3, size = 1) + scale_y_continuous("Dew Point",
    breaks = pretty_breaks(n = 10), limits = c(0,
        100)) + scale_x_continuous("Temperature (F)",
    breaks = pretty_breaks(n = 10), limits = c(0,
        100)) + mytheme

plot4 = ggplot(df2, aes(DEWP_origin)) + geom_density(aes(fill = Delay),
    alpha = 0.5) + coord_flip() + mytheme +
    scale_x_continuous("", limits = c(0,
        100)) + ylab("Temperature") + theme(axis.text = element_blank())

ggarrange(plot1, plot2, plot3, plot4, ncol = 2,
    nrow = 2, widths = c(0.8, 0.2), heights = c(0.2,
        0.8), common.legend = T)

ggplot(df2, aes(CRSDepTime)) + geom_density(aes(fill = Delay),
    alpha = 0.5) + mytheme + ylab("") + scale_x_continuous("Scheduled Departure Time",
    breaks = seq(0, 1440, 360), labels = c("0:00",
        "6:00", "12:00", "18:00", "24:00")) +
    theme(axis.text.y = element_blank(),
        axis.ticks.y = element_blank())
```

10.6b Exploratory Analysis

```
import pandas as pd
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt
df = pd.read_csv('full_data_clean_n10000.csv')
df.head()

plt.subplots(figsize=(10,6))
sns.countplot(x='dep_delay_class', data=df).set(title="DepDelay Class Count")
plt.show()

plt.subplots(figsize=(10,6))
sns.boxplot(x=df['DepDelay']).set(title="Boxplot for DepDelay")
plt.show()

plt.subplots(figsize=(10,6))
sns.scatterplot(x=df['ArrDelay'], y=df['TEMP_dest']).set(title="ArrDelay v. TEMP_dest")
plt.show()
```

```
df1 = df[['ELEVATION_origin', 'TEMP_origin', 'DEWP_origin',
          'STP_origin', 'VISIB_origin', 'WDSP_origin', 'MXSPD_origin',
          'MAX_origin', 'MIN_origin', 'PRCP_origin', 'SNDP_origin',
          'Year', 'Month', 'DayofMonth', 'DayOfWeek', 'Date',
          'Flight_Number_Reporting_Airline', 'OriginCityName', 'OriginState',
          'DestCityName', 'DestState', 'CRSDepTime', 'DepDelay', 'TaxiOut',
          'WheelsOff', 'WheelsOn', 'TaxiIn', 'CRSArrTime', 'ArrDelay', 'Diverted',
          'CRSElapsedTime', 'ActualElapsedTime', 'AirTime', 'Distance']]

corr = df1.corr()

mask = np.triu(np.ones_like(corr, dtype=bool))

plt.subplots(figsize=(10, 10))

sns.heatmap(corr, mask=mask, vmax=.3, center=0,
            square=True, linewidths=.5, cbar_kws={"shrink": .5})
```

10.7 Tune Random Forest Grid Search

```
from email import header
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn import metrics
from matplotlib import pyplot as plt
from matplotlib.pyplot import figure
import pyreadr
import timeit
import numpy as np
import csv
from sklearn.model_selection import RandomizedSearchCV

#### Read in our data ####
result = pyreadr.read_r('full_data_clean.RDS')
df = result[None]

#df = pd.read_csv('full_data_clean_n10000.csv')

df_temp = pd.DataFrame(columns=['sample_perc', 'time', 'f1', 'accuracy'])  # fake empty
↪  dataset to add header to csv
df_temp.to_csv("results.csv", mode="w", header=True, index=False)


#### Data Cleaning ####
numerics = ['int16', 'int32', 'int64', 'float16', 'float32', 'float64']  # we only want
↪  numeric data
df = df.select_dtypes(include=numerics)


df.drop('TaxiOut', axis=1, inplace=True) # drop taxiout
df.drop('WheelsOff', axis=1, inplace=True) # drop WheelsOff
```

```
df.drop('WheelsOn', axis=1, inplace=True) # drop WheelsOn
df.drop('TaxiIn', axis=1, inplace=True) # drop taxiout
df.drop('ActualElapsedTime', axis=1, inplace=True) # drop ActualElapsedTime

df['delay'] = np.where(df['DepDelay'] >= 15.0, '1', '0')  # if dept time delyed more then
↪  15 min, mark it as delayed
df.drop('DepDelay', axis=1, inplace=True) # drop DepDelay

df['delay'] = np.where(df['ArrDelay'] >= 15.0, '1', '0')  # if arrive time delyed more
↪  then 15 min, mark it as delayed
df.drop('ArrDelay', axis=1, inplace=True) # drop ArrDelay

print(df['delay'].value_counts())  # examine class imbalence


#### Now time for the fun part, modeling!! ####
sample_percs = [.50]  # what perc do we want to undersample ontime class by

for sample_perc in sample_percs:
    train, test = train_test_split(df, test_size=0.2)  # Split data 20% test, 80% train

    # we will only under sample not delyed since we will have a faster run time
    delayed = train.loc[(train['delay'] == '1')]  # all delayed instances
    on_time = train.loc[(train['delay'] == '0')]  # all on time instances

    temp_on_time = on_time.sample(n=round(len(on_time) * sample_perc))  # randomly sample
    frames = [delayed, temp_on_time]
    train = pd.concat(frames)

    y_train = train['delay']
    train.drop('delay', axis=1, inplace=True) # drop target var
    y_test = test['delay']
    test.drop('delay', axis=1, inplace=True)  # drop target var

    #### Now we want to implament gridsearch ####

    # set our paramiter ranges
    n_estimators = [50, 100, 150, 200, 250, 300, 350, 400, 450, 500]  # Number of trees
↪  in random forest
    max_features = ['auto', 'sqrt', 'log2']  # The number of features to consider when
↪  looking for the best split
    max_depth = [20, 40, 60, 80, 90, 100, None]  # The maximum depth of the tree
    min_samples_split = [2, 3, 4, 5, 6]  # The minimum number of samples required to
↪  split an internal node:
    criterion = ['gini', 'entropy']  # The function to measure the quality of a split

    random_grid = {'n_estimators': n_estimators,
                'max_features': max_features,
                'max_depth': max_depth,
                'min_samples_split': min_samples_split,
                'criterion': criterion}

    #start = timeit.default_timer()  # start timer
```

```
    n_iter = 3  # n_iter was 100
    cv = 2

    clf = RandomForestClassifier()  # our RF model. Off the shelf
    rf_random = RandomizedSearchCV(estimator = clf, param_distributions = random_grid,
↪   n_iter = n_iter, cv = cv, verbose=2, n_jobs = -1)


    rf_random.fit(train, y_train)  # training
    y_pred = rf_random.predict(test)  # predictions
    f1 = metrics.f1_score(y_test, y_pred, average='macro')
    acc = metrics.accuracy_score(y_test, y_pred)


    ↪  print('##########################################################################################
    print(f'N_iter: {n_iter}')
    print(f'CV: {cv}')
    print(f'Acc: {acc}')
    print(f'F1: {f1}')
    print(f'Best params: {rf_random.best_params_}')



    # y_pred = clf.predict(test)  # predictions

    # f1 = metrics.f1_score(y_test, y_pred, average='macro')
    # acc = metrics.accuracy_score(y_test, y_pred)

    # stop = timeit.default_timer()  # end timer

    # # write results to csv
    # with open(f"results.csv", "a+", newline='') as csvfile:
    #     csvwriter= csv.writer(csvfile)
    #     csvwriter.writerows([[sample_perc, stop - start, f1, acc]])

    # # create feature importance chart
    # sorted = rf_random.feature_importances_.argsort()
    # plt.figure(figsize=(17, 10))
    # plt.barh(train.columns[sorted], clf.feature_importances_[sorted])
    # plt.xlabel("Permutation Importance")
    # plt.savefig(f'Permutation_Importance_{cv}_{acc}.png')

    # return sample_perc || this tells us the percent we undersampled the on time class
    ↪  by
```

10.8 Run Random Forest

```
from email import header
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn import metrics
```

```
from matplotlib import pyplot as plt
from matplotlib.pyplot import figure
import pyreadr
import timeit
import numpy as np
import csv

#### Read in our data ####
result = pyreadr.read_r('full_data_clean.RDS')
df = result[None]

#df = pd.read_csv('full_data_clean_n10000.csv')

df_temp = pd.DataFrame(columns=['sample_perc', 'time', 'f1', 'accuracy'])  # fake empty
↪  dataset to add header to csv
df_temp.to_csv("results.csv", mode="w", header=True, index=False)


#### Data Cleaning ####
numerics = ['int16', 'int32', 'int64', 'float16', 'float32', 'float64']  # we only want
↪  numeric data
df = df.select_dtypes(include=numerics)

df.drop('CRSElapsedTime', axis=1, inplace=True) # drop taxiout
df.drop('AirTime', axis=1, inplace=True) # drop taxiout
df.drop('TaxiOut', axis=1, inplace=True) # drop taxiout
df.drop('WheelsOff', axis=1, inplace=True) # drop WheelsOff
df.drop('WheelsOn', axis=1, inplace=True) # drop WheelsOn
df.drop('TaxiIn', axis=1, inplace=True) # drop taxiout
df.drop('ActualElapsedTime', axis=1, inplace=True) # drop ActualElapsedTime

df['delay'] = np.where(df['DepDelay'] >= 15.0, '1', '0')  # if dept time delyed more then
↪  15 min, mark it as delayed
df.drop('DepDelay', axis=1, inplace=True) # drop DepDelay

df['delay'] = np.where(df['ArrDelay'] >= 15.0, '1', '0')  # if arrive time delyed more
↪  then 15 min, mark it as delayed
df.drop('ArrDelay', axis=1, inplace=True) # drop ArrDelay

print(df['delay'].value_counts())  # examine class imbalence


#### Now time for the fun part, modeling!! ####
sample_percs = [1, .75, .50, .25]  # what perc do we want to undersample ontime class by

for sample_perc in sample_percs:
    train, test = train_test_split(df, test_size=0.2, random_state=42)  # Split data 20%
↪  test, 80% train

    # we will only under sample not delyed since we will have a faster run time
    delayed = train.loc[(train['delay'] == '1')]  # all delayed instances
    on_time = train.loc[(train['delay'] == '0')]  # all on time instances
```

```
    temp_on_time = on_time.sample(n=round(len(on_time) * sample_perc))  # randomly sample
    frames = [delayed, temp_on_time]
    train = pd.concat(frames)

    y_train = train['delay']
    train.drop('delay', axis=1, inplace=True) # drop target var
    y_test = test['delay']
    test.drop('delay', axis=1, inplace=True)  # drop target var

    start = timeit.default_timer()  # start timer
    clf = RandomForestClassifier()  # our RF model. Off the shelf
    clf.fit(train, y_train)  # training
    y_pred = clf.predict(test)  # predictions

    f1 = metrics.f1_score(y_test, y_pred, average='macro')
    acc = metrics.accuracy_score(y_test, y_pred)

    stop = timeit.default_timer()  # end timer

    # write results to csv
    with open(f"results.csv", "a+", newline='') as csvfile:
        csvwriter= csv.writer(csvfile)
        csvwriter.writerows([[sample_perc, stop - start, f1, acc]])

    # create feature importance chart
    sorted = clf.feature_importances_.argsort()
    plt.figure(figsize=(17, 10))
    plt.barh(train.columns[sorted], clf.feature_importances_[sorted])
    plt.xlabel("Permutation Importance")
    plt.savefig(f'Permutation_Importance_{sample_perc * 100}.png')

    # return sample_perc || this tells us the percent we undersampled the on time class
    ↪  by
```