Final Project

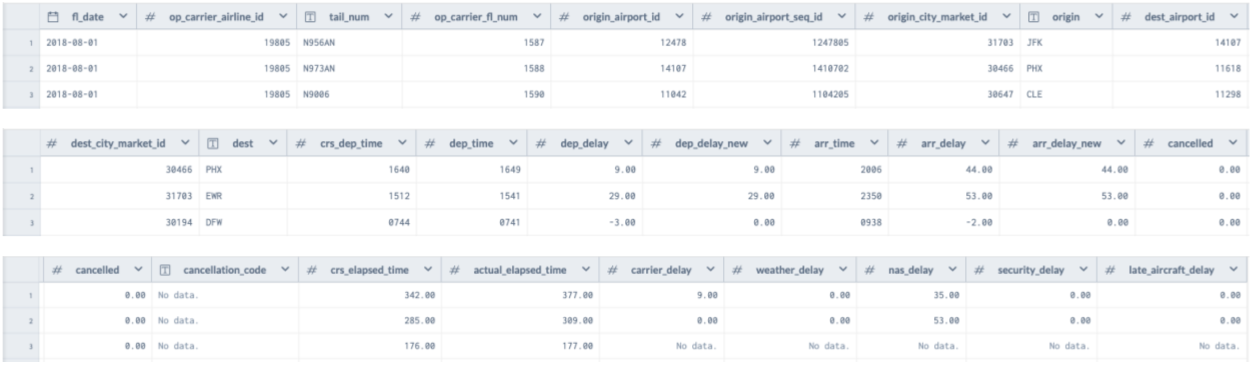
MGMT 59000- Computational Complexity

Ryan Egbert, Jai Woo Lee

Description of Data

The dataset we used to perform data analysis is National Airline On-Time Performance Statistics in August 2017[1] and August 2018[2] and from the US Department of Transformation. Each dataset consists of 29 columns and over 500,000 observations. Each row contains travel information of flight, including dates, names and IDs of origin and destination airports, schedule and actual flight time, delayed time and types of the delays, whether the flights are canceled, and types of cancellations.

[Exhibit 1: Dataset – Airport On-Time Performance Statistics in August 2018]



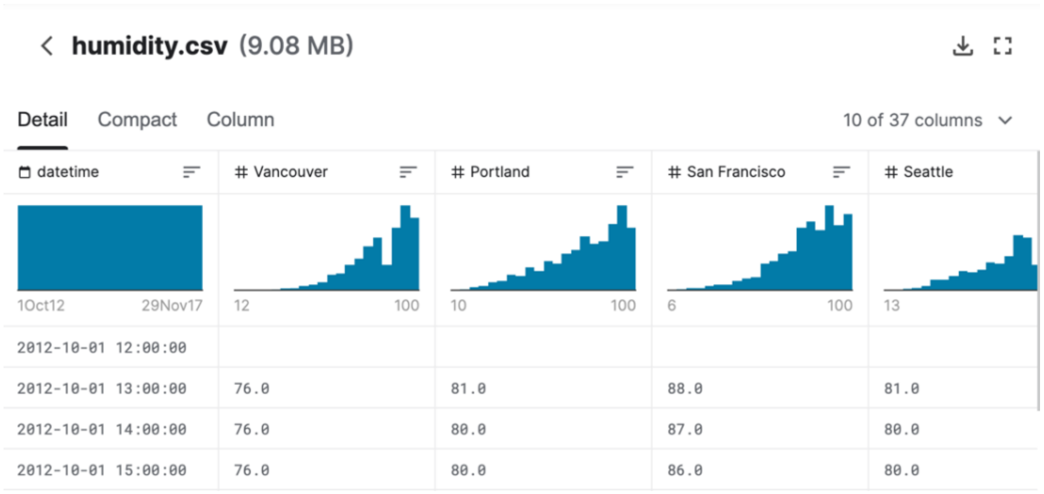
	fl_date	op_carrier_airline_id	tail_num	op_carrier_fl_num	origin_airport_id	origin_airport_seq_id	origin_city_market_id	origin	dest_airport_id
1	2018-08-01	19805	N956AN	1587	12478	1247805	31703	JFK	14107
2	2018-08-01	19805	N973AN	1588	14107	1410702	30466	PHX	11618
3	2018-08-01	19805	N9006	1590	11042	1104205	30647	CLE	11298

	dest_city_market_id	dest	crs_dep_time	dep_time	dep_delay	dep_delay_new	arr_time	arr_delay	arr_delay_new	cancelled
1	30466	PHX	1640	1649	9.00	9.00	2006	44.00	44.00	0.00
2	31703	EWR	1512	1541	29.00	29.00	2350	53.00	53.00	0.00
3	30194	DFW	0744	0741	-3.00	0.00	0938	-2.00	0.00	0.00

	cancelled	cancellation_code	crs_elapsed_time	actual_elapsed_time	carrier_delay	weather_delay	nas_delay	security_delay	late_aircraft_delay
1	0.00	No data.	342.00	377.00	9.00	0.00	35.00	0.00	0.00
2	0.00	No data.	285.00	309.00	0.00	0.00	53.00	0.00	0.00
3	0.00	No data.	176.00	177.00	No data.	No data.	No data.	No data.	No data.

We also used hourly weather data for several different cities around the country. The data included information on humidity, pressure, temperature, wind speed and direction, and a general weather description. The data was pulled from Kaggle[3] and consisted of 7 different tables.

[Exhibit 2: Dataset – Historic Hourly Weather Dataset, Kaggle]



humidity.csv (9.08 MB)

Detail Compact Column — 10 of 37 columns

datetime	Vancouver	Portland	San Francisco	Seattle
1Oct12 – 29Nov17	12 – 100	10 – 100	6 – 100	13
2012-10-01 12:00:00				
2012-10-01 13:00:00	76.0	81.0	88.0	81.0
2012-10-01 14:00:00	76.0	80.0	87.0	80.0
2012-10-01 15:00:00	76.0	80.0	86.0	80.0

[1] Data source: https://www.transtats.bts.gov/DL_SelectFields.asp?gnoyr_VQ=FGJ

[2] Data source: https://data.world/dot/airline-on-time-performance-statistics/workspace/file?filename=August+2018+Nationwide.csv

[3] Data source: https://www.kaggle.com/selfishgene/historical-hourly-weather-data

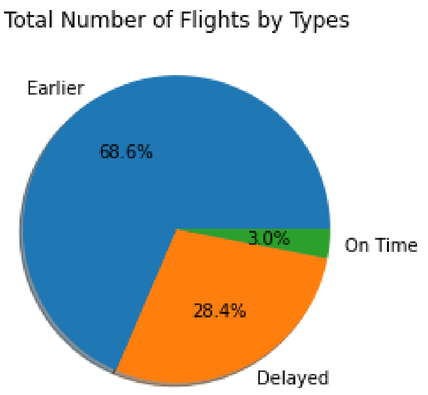
Objective #1: Performing EDA

Technique #1: Data manipulation and visualization

Our first objective is to better understand the dataset by performing EDA based on the techniques we learned from the class, such as data manipulation using the Pandas library and visualization using the Matplotlib library.

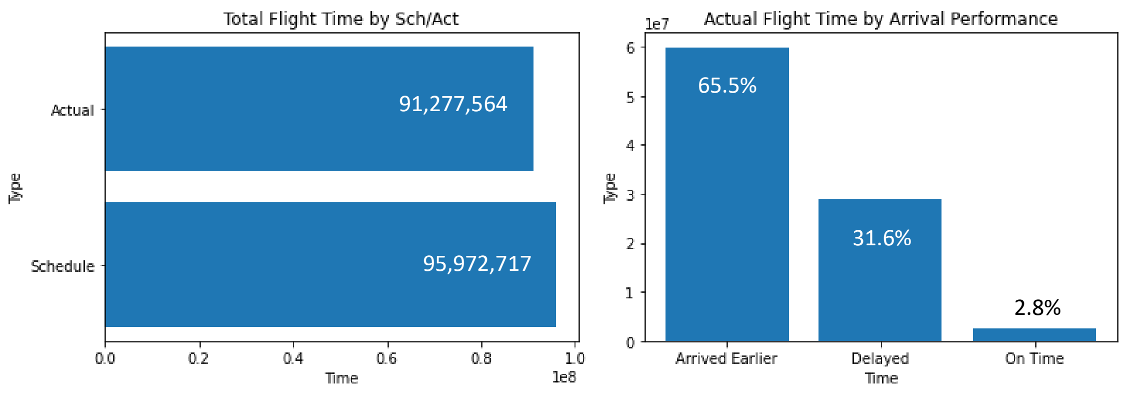
First, the total number of flights in August 2018 is 701,352. Out of the total number, the number of flights that arrived earlier than scheduled is 469,586 (66.95%). In addition, the number of flights that are delayed is 194,395 (27.72%). The rest indicates on-time or are unrecorded.

[Exhibit 3: Total Number of Flights by Arrival Performance]



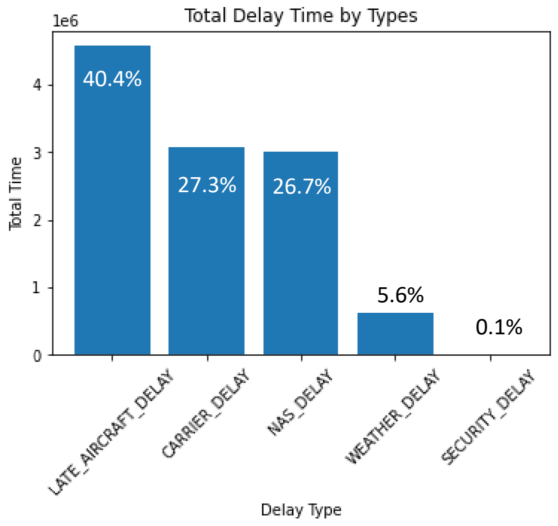
The total scheduled flight time in the same period is 95,972,717 minutes, whereas the total actual flight time is 91,277,564. In other words, customers arrived at the destination 5% earlier than the scheduled time on average. Of the actual flight time, the portion of arrived earlier, delayed, and on-time are 65.54%, 31.63%, and 2.83%, respectively.

[Exhibit 4: Total Number of Flights by Arrival Performance]



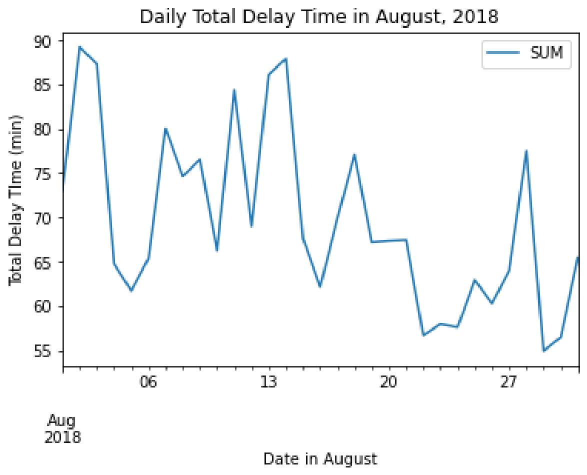
Then we looked at the total delay time more closely by identifying how much each type of delay affects the total delay time. The type that has the most impact is Late Aircraft delay(40.4%, delay from arriving airplane), followed by Carrier delay(27.3%), NAS delay(26.7%, National Aviation System delay, related to air traffic volume and control issues), Weather delay(5.6%), and Security delay(0.1%).

[Exhibit 5: Total Delay Time by Types]



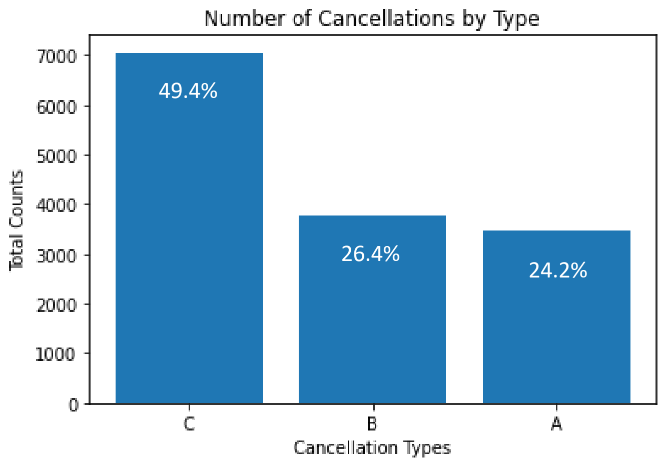
We analyzed a daily trend of total delay time. The total delay time for each day is very volatile. The high and low for the total delay time in the first half of the month is more volatile than those in the rest half. This may have been caused due to more customers for travels, thus more traffics of the airplanes, since the Late Airport delay and the Carrier delay are the top2 delay types.

[Exhibit 6: Daily Total Delay Time in August 2018]



The last EDA we performed is to identify the total number of cancellation types. Type A, B, and C cancellations indicate Carrier Caused, Extreme Weather, and NAS cancellations, respectively. In August 2018, the most common cancellation type was type C, recording 7,056 counts, approximately twice that of type B and A.

[Exhibit 7: Number of Cancellation by Types]

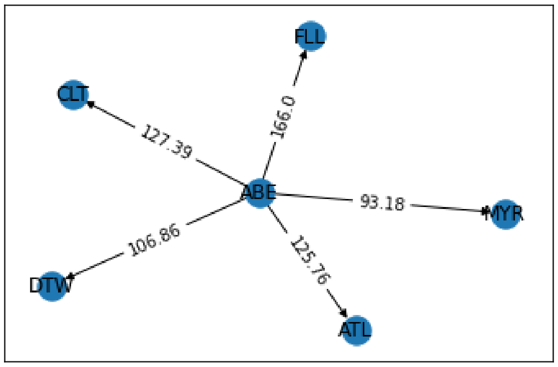


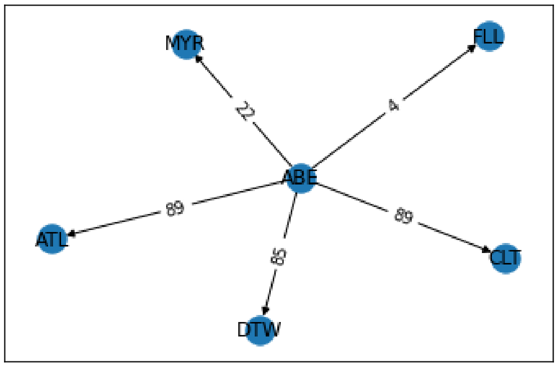
Graph building process to perform advanced techniques

This dataset can also be represented in a graph structure rather than a data frame structure, using each start and endpoint as vertex and distance or number of travels for each route as a weight. This will not only allow us to create a more intuitive visual representation of the dataset but also perform advanced techniques in our analyses, such as network algorithm and dynamic programming, which will be discussed further.

We first created two different graph structures using different types of weights: one with the distance and the other with the number of flights between two nodes. The first one will be used to get the shortest distance between airports, while the second one will be used to gain popularity of each airport. To create these two graphs, we used subsetting syntax to extract columns that we needed, and the codes related to building graph structures that we learned from the class. Please note that below exhibits are examples with only 5 observations, as the full graphs are massive. There are 365 nodes and 5902 edges in total in our graphs.

[Exhibit 8: Graphs with five observations (different weights, Left: distance | Right: Number of Trips)]





Objective #2: Getting the shortest distance between airports and popularity of cities

Technique #2: Graph/Network Algorithm – Bellman-Ford

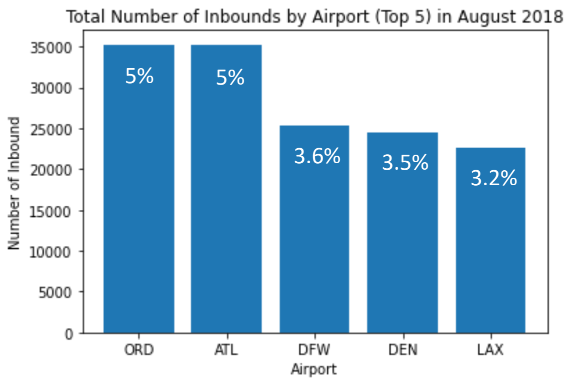
We used the Bellman-Ford algorithm to find the shortest distance between two cities. The inputs are a graph-structured dataset, start node, and the end note in the function. Within the function, we first stored distances(weights) between all neighboring cities and implemented the relax algorithm. The exhibit below shows examples of the shortest distance between cities.

[Exhibit 9: Examples of the shortest distance between two cities based on Bellman-Ford]

```
1. Time it takes from IND to HNL is: 594.92 minutes
2. Time it takes from BOS to HNL is: 704.78 minutes
3. Time it takes from GEG to ORH is: 415.06 minutes
4. Time it takes from PBI to LEX is: 185.6 minutes
5. Time it takes from YKM to JNU is: 196.3 minutes
```

Based on the graph structures, we could also get insights regarding the popularity of all cities by calculating the number of inbounds and outbounds for each airport. The bar chart below shows the five most popular airports based on inbound flights.

[Exhibit 10: Five most popular Airports in August 2018]



Objective #3: Getting the shortest path between airports

Technique #3: Dynamic Programming

We have already established a way to find the shortest path between two cities using a well-known algorithm. But is there a way to create a similar solution using dynamic programming? In order to figure out a viable algorithm, we followed the four-step process discussed in class:

1) Subproblems

- Let the shortest path between two nodes x and y be c(x,y).

2) Problem

- Given i,j for all x,y the answer to our problem is c(i,j).

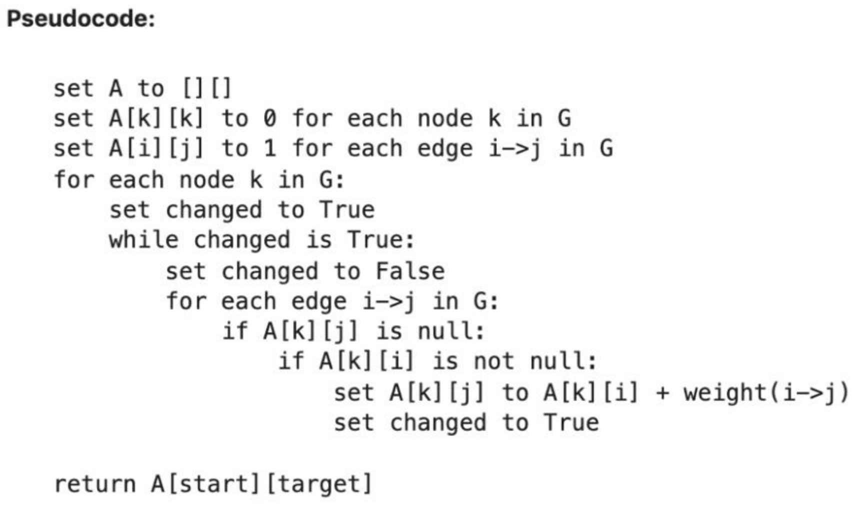
3) Base Cases

- c(a,b) = *null* for all a,b
- c(k,k) = 0 for all vertices k in a graph G
- c(i,j) = 1 for all edges i,j in a graph G

4) Solution

- For each vertex k in a graph G, we need to iterate through each edge (i,j) in G and determine if there is a possible path from k to j.
- If c(k,j) is null and c(k,i) is not null, then c(k,j) = c(k,i) + w(i,j) where w(i,j) is the weight of the edge (i,j). Repeat this process for each node until no further changes can be made.

[Exhibit 11: pseudocode of dynamic programming shortest path algorithm]



Pseudocode:

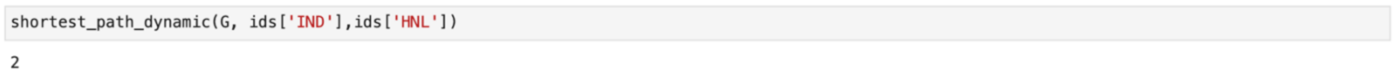
```
set A to [][]
set A[k][k] to 0 for each node k in G
set A[i][j] to 1 for each edge i->j in G
for each node k in G:
    set changed to True
    while changed is True:
        set changed to False
        for each edge i->j in G:
            if A[k][j] is null:
                if A[k][i] is not null:
                    set A[k][j] to A[k][i] + weight(i->j)
                    set changed to True

return A[start][target]
```

Implementing this algorithm, we succeeded in generating the shortest path, not only with small-scale graphs but with a network as large as the flight graph we generated from our data.

Example Use

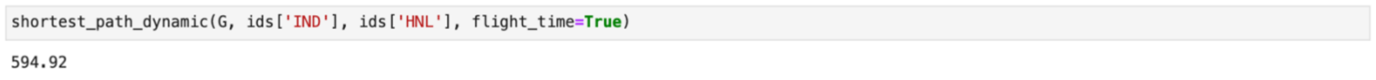
Say, for instance, we would like to take a trip to Hawaii and depart from the Indianapolis airport. However, we don't want to make the trip if there are two or more layovers along the way.
In order to determine how many layovers occur during the travel, we can simply plug in our airport values into our function and find out.



```
shortest_path_dynamic(G, ids['IND'],ids['HNL'])

2
```

Lucky for us, it appears that there would only be one layover on our journey to Hawaii. Now that we know how many stops we would be required to make, we want to prepare for the amount of time we would spend in the air.



```
shortest_path_dynamic(G, ids['IND'], ids['HNL'], flight_time=True)

594.92
```

594 minutes is nearly 10 hours! We'd better be prepared with a book or two if we are going to sit on a plane for that long.

Machine Learning – Data Cleaning and Merge

After some preliminary work with descriptive and exploratory data analysis, it appears that there are interesting insights to be gained by looking at flight activity performance. Nearly a quarter of all flights were delayed in August 2018 [Exhibit 2]. In order to prepare airlines for possible delays, we created multiple predictive models that would predict the probability of delay using flight information as well as weather data.
Rather than collect weather data for every airport in our dataset, however, we decided to narrow our model to five cities: Los Angeles, Chicago, Boston, Phoenix, and Denver.

Step 1: Clean and merge data

Just like with most analytical problems, the cleaning step took the longest in our process. Before we could use our data in a predictive model, our data needed to be cleaned and then merged with the matching weather data.

Weather data was matched with the flight data by using the departure time and the time of the weather information.

Step 2: Prepare data for models

After the dataset had been cleaned and merged, we prepared it to be fed into different models. This consisted of creating dummy variables for our categorical variables, dropping unneeded columns, and converting the data to a numerical array.

Step 3: Train/Test Split

Using the data that had been prepared during Step 2, we split the data into training and testing sets using a 60/40 split.

Step 4: Model Building

Finally, we were able to build two different models: random forest and neural network.

Step 5: Model Validation

Testing the accuracy of the models is how we determined the efficacy of each of their performances.

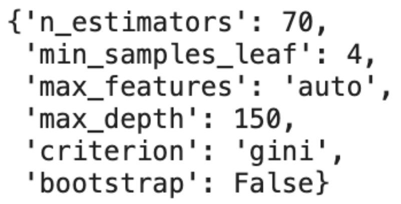
Objective #4: Predicting delays

Technique #4: Machine Learning – Random Forest

The random forest model was built as a RandomForestClassifier and tuned with cross-validation using a RandomizedSearchCV object.

Ultimately the best parameters for our model were the following:

[Exhibit 12: Best parameters for random forest]



```
{'n_estimators': 70,
 'min_samples_leaf': 4,
 'max_features': 'auto',
 'max_depth': 150,
 'criterion': 'gini',
 'bootstrap': False}
```

The random forest was completed with an accuracy of **69.24%**.

Technique #5: Machine Learning – Neural Network

The neural network was built as a multilayer perceptron model with 3 hidden layers of sizes 20, 50, and 20.

The neural network was completed with an accuracy of **65.88%**.

Final Thoughts on Models

There is obviously still work to be done in terms of improving the accuracy of our predictive models. At the core, we need to improve the quality of our data. This could mean finding additional data, adding additional variables, or even removing some variables that are not statistically significant. We could also use data over the course of an entire year rather than just one month.

In taking these steps, we would have the ability to improve our model to the point that it would be usable for airports and airlines to predict when delays will occur. By having this information, these users could adequately plan and prepare for (and in some cases, prevent) delays from occurring.