Capstone 2 Data Wrangling

Introduction

Now that I have decided upon my project, correlating airline flight delays and cancellations with various weather events, I needed to collect data and transform them into useable formats. Flight data comes from the Bureau of Transportation Statistics,[i] and weather event data comes from the "US Weather Events (2016 - 2019)" dataset on Kaggle.[ii] Some miscellaneous weather events were filled in from the Weather Underground weather service site.[iii]

Because flights and weather events do not start and end together, we have to decide how to categorize the data timewise. We settle on per-day summaries, reasoning that weather events, particularly winter weather events, can affect flights several hours before or afterwards. This may not hold up so well with events such as rain and fog, however; a severe thunderstorm that passes over an airport one evening should not affect flights that morning. I suspect that this will generate false negatives from all the flights that get weather codes yet are unaffected by poor weather that occurs at different times of the day. This means that we are probably going to underestimate the weather's effects on flights, which we should remember if we come across any significant findings.

Data wrangling consisted of three phases. In the first, we collect the weather data and compiled daily summaries for America's busiest airports. In the second, we collect the flight data, split them according to flight departures and arrivals, continued with only the data from the busiest airports, and compiled daily summaries. In the third, we merge the daily weather summaries with the daily flight summaries and began exploratory data analysis.

Three of the United States' four major airlines—American, Delta, and United—use a hub-and-spoke model, which produces significantly more air traffic at just a few airports.[1] Thus it makes sense to take just the busiest airports in America and analyze flight and weather data there. Note that airport business rankings can fluctuate from year to year, but aside from long-term disruptive events such as the coronavirus in 2020, these rankings should not change much. I could not find documentation for the busiest airports in 2019, so I decided to use the 2018 data.[iv] I arbitrarily stopped after the 24th busiest airport in 2018, Logan International Airport in Boston.

Part 1: Weather events

Gathering weather events based on hourly observations at 24 airports over four years could have required collection of almost a million data points. Even if such a repository existed that held all this data together, we would need to find a way to write an API to find and scrape all that data, and then collect usable information from each data point. Fortunately, there exists a database on Kaggle, which was based on a paper by Moosavi, Sobhan, et al.[v] that recorded and classified weather events according to the following codes:

[1] The other major airline, Southwest, uses more of a point-to-point model and has "focus cities," not hubs.

Rain: Light, Moderate, Heavy
Fog: Moderate, Severe[2]
Snow: Light, Moderate, Heavy
Precipitation: "UNK" (unclassified precipitation that will obviously require fixing)
Hail: Other
Storm: Severe
Cold: Severe

We trim the data down to our 24 airports; this cut the number of entries from over 5 million to only about 50,000. After some cleaning, we map the latitude-longitude data on Google maps to verify their accuracy. In each case tested, they not only corresponded to the respective airports but the weather instrumentation at those airports! Thus, we conclude that all the weather data are geographically correct.

Since we are going to summarize the data by day, we have to convert all the weather data from their UTC times into the local times. Otherwise, not only would the cutoffs between days not occur at midnight, they would occur at different times of the day depending on their time zone and daylight savings time. Fortunately, the weather data already comes with each cities' time zones listed, saving us the trouble of having to code that in. However, Phoenix is recorded as being in the Mountain Time Zone, which we have to change to Mountain Standard Time ("US/Arizona"), as most of Arizona, including Phoenix, does not observe daylight savings time. Then we use Python's `pytz` class to convert all the start and end times into `datetime` objects with the correct time zones. `pytz` automatically handles the transition in and out of daylight savings time, including the unique situation in Arizona.

The defines "Storm" as wind of at least 60 km/h, which is about 40 mph. We rename the "Storm" code to "Wind."

The most challenging part of this part of the data wrangling comes from the fact that the Moosavi, et al. paper made no distinction between sleet and hail. This is a problem; sleet and hail are two distinct weather phenomena. Sleet is rain that passes through a colder layer of air and freezes before reaching the ground; it rarely measures more than a couple millimeters in diameter. Hailstones are chunks of ice that grow as they are blown upwards into a supercell, too big to melt before reaching the ground. Hail can grow to several centimeters across and usually accompanies severe weather. The dataset contained 210 entries with a Hail code at our airports. Though we could look all of these up individually from historic weather data, we infer whether each code meant hail or sleet based on three assumptions:

1. Any Hail code in June, July, August, or September is probably hail. Hail is a warm-weather phenomenon, and frozen precipitation other than hail is rare if not impossible this time of the year, even in places such as Denver and Boston.
2. Any Hail code within six hours of a Snow code is probably sleet. Sleet is a form of cold-weather precipitation which can accompany or immediately precede or follow snow. A weather system that produces severe weather such as hail and then produces sleet is rare.

[2] I am not sure how fog could be classified as "severe." "Heavy" seems more appropriate.

3. Any remaining Hail code in November, December, January, or February is probably sleet. This is a less sure assumption, but during these months, sleet is much more common than hail. Hail during the winter is not impossible, however, particularly in mild climates, so we may need to revisit this assumption.

These three rules classified 153 of the 210 entries that had Hail codes. To classify the remaining 57 entries, we take the daily mean temperature per month, per city based on the climate data from NOAA via Wikipedia. We classify any Hail code corresponding to a month with an average daily temperature of under 50°F as sleet at least 50°F as hail. Though this is not a sure-fire way to impute, it is about the best we can do without either looking up these weather events manually or restarting this phase of the project. Besides, these 57 remaining events only constitute just over 0.1% of all the weather events; as we will see, other weather codes come into play much more. With these decisions, we are able to completely split the old Hail code into hail and sleet.

We convert the weather types into values for each type, consecutively ranked for severity. For instance, a Light, Moderate, or Heavy snow code enters the Snow column as a 1, 2, or 3, respectively; all codes that do not correspond to snow get a 0 in the Snow column. This led to numeric codes for fog, snow, hail, sleet, wind, rain, and cold.

One issue in the data remained: Unknown precipitation events. First, we verify that weather events had a severity code of "UNK" if and only if their corresponding type is classified as "Precipitation." Next we make sure that none of these codes are the very first or last for their respective airports; if this happens, we will need to code for it. Then we impute by taking the average of the numeric codes of the entries immediately before and after the current entry, rounding up non-integers to the nearest integer.

Finally, we merge the data into two daily summaries, one by start times and one by end times, making sure not to include any times that were outside our timeframe. We assume that weather events can affect flights at different times of the day, so we aggregate by daily maximums per code; we may have to revisit this assumption later. Then we merge these two pivot tables, again aggregating by maximum per code per day. We also compute a correlation matrix between the seven numeric codes and save both it and this final pivot table.

Part 2: Airline data

Unlike the weather data, which relied on summary codes, we can easily obtain data on individual flights from the Bureau of Transportation Statistics. A wealth of information is stored for each flight, such as scheduled departure and arrival times, actual departure and arrival times, whether the flight was cancelled, and so forth. We collect these statistics as CSV files, one for each month from January 2016 to December 2019, naming them appropriately for easy identification and importing (ex., 2016_01_T_ONTIME_REPORTING.csv for January 2016). Next we import these files into Python, using the `glob` package to automate the process of iterating over all 48 files. Then we concatenate them into one large DataFrame and removed the duplicates. In all there were over 25 million flights!

We inspect the data with a few sample flights, including a couple flights that I took that crossed time zones. Cross-referencing the flight numbers, departure times, and arrival times with my own information, we confirm that all departure and arrival times were given in their local time zones.

Next we remove any flight that neither departed from nor arrived at one of the 24 busiest airports in the United States. This only cut the number of flights down to only 22.4 million, as most domestic commercial flights travel to or from at least one of these cities. Still, it is a start. After this, we do some cleanup to the DataFrame, then we list the number of missing values per remaining column. DEP_DELAY_NEW, which indicates the departure delay, and DEP_DEL15, which indicates whether the flight departed at least 15 minutes late and thus counts as a delay, had the same count of missing values. The same was observed between ARR_DELAY_NEW and DEP_DEL15, which concern arrival delays. We confirm that there was a one-to-one match between DEP_DELAY_NEW and DEP_DEL15 and between ARR_DELAY_NEW and DEP_DEL15, so we impute all missing values as 0, which correspond to flights that were not delayed. We check CANCELLED and CANCELLATION_CODE, which respectively check whether a flight was cancelled and what its reason was, and find a one-to-one match between the rows where CANCELLED was 0 and CANCELLATION_CODE was undefined, so we impute zeroes into the CANCELLATION_CODE in these instances. There were no undefined CANCELLED codes to impute.

Cancellation Code B means the flight was delayed for weather-related reasons. Since this is the only type of cancellation we are interested in with this project, we create a new variable, WeatherCancelled, that is a 1 for every row where CANCELLATION_CODE is B and a 0 everywhere else. Similarly, we consider the column WEATHER_DELAY, which has some missing values, which we take to mean that there is no weather delay, so we impute these with 0. Then we create another column, WeatherDelayed, that is set to 1 if and only if WEATHER_DELAY is at least 15, the threshold for a flight's being considered delayed, and 0 otherwise.

We then investigate flights that were diverted to another airport or back to its origin airport. However, there was zero overlap between rows where WeatherDelayed or WeatherCancelled, and DIVERTED, were both 1. This means that weather events did not result in any flight diversions, so diversions will not be analyzed in this project.

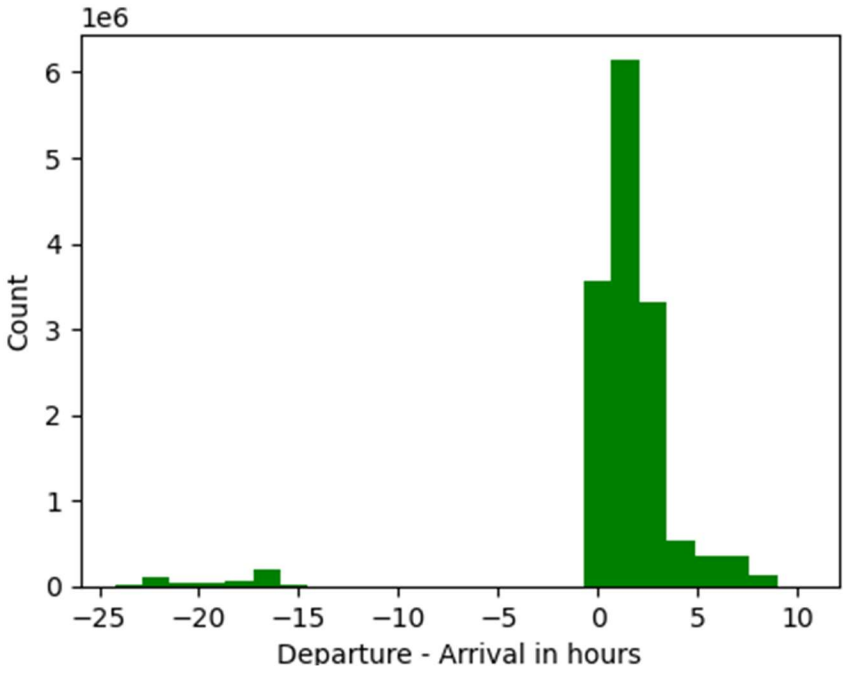
At this point, we have sixteen major airports, with up to $24^2 - 24 = 552$ unique flight paths, not including all the flight paths to and from the minor airports.[3] Even if a couple of these connections are missing, the point is the sheer number of paths to analyze, and we haven't even considered the weather effects yet! Clearly, we need to make a simplification. To do this, we duplicate the DataFrame, use one copy for departure airports and the other for arrival cities, and

[3] This assumes a "dense" graph where all 24 nodes, represented by the major airports, are bidirectionally connected to every other airport in this graph. The number is surely less than this but is still very likely several hundred.

ignore the connections between airports. Now we are down to at most 24 × 2 = 48 points to analyze.

With the DataFrame duplicated, we begin to clean both of them up. First, we trim the departing flights' DataFrame to just those who originated from one of the busiest 24 airports, and we trim the arriving flights' DataFrame to just those who arrived there.

The dates given correspond to the flights correspond to departure dates, which will affect the data of flights. To find them, we plot a histogram of the departure times minus arrival times.



The large negative differences are clear signs of a departure before midnight and an arrival after midnight. We shift these arrival dates forward by one day. Note that there are some flights whose arrival times are between zero and one hour before departure times; these correspond to short flights that travel west across time zones. In no cases did any of these flights take off after midnight and land before midnight.

Next we convert all the listed times into datetime objects in the appropriate time zones. Time zones were not listed in the original DataFrame, so we create another DataFrame that will map each city into its correct time zone. With these datetime data in place, we safely drop some of the redundant columns in the departures and arrivals DataFrames.

Finally, we use pivot tables to take daily sums per airport for the two DataFrames and save the results.

Part 3: Merging

Now that we have the parts, we need to put them together. We load our three saved DataFrames: Weather events, departures, and arrivals. We merge both the departures and arrivals

on the weather events and call the resulting DataFrames `departure_events` and `arrival_events`, respectively. Because we want to include all departures and arrivals, but some days will not have any associated weather events at some airports, both merges are left merges. Clearly, we impute all undefined weather codes as zeroes. We make some minor cleaning to the DataFrames then proceed to the exploratory data analysis.

---

[i] https://www.transtats.bts.gov/DL_SelectFields.asp

[ii] https://www.kaggle.com/sobhanmoosavi/us-weather-events

[iii] https://www.wunderground.com/

[iv] https://www.faa.gov/airports/planning_capacity/passenger_allcargo_stats/passenger/media/preliminary-cy18-commercial-service-enplanements.pdf

[v] Moosavi, et al. "Short and Long-term Pattern Discovery Over Large-Scale Geo-Spatiotemporal Data." *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2019.