FINAL Team_4-2_Phase_3_Report (Python)

Import notebook

SkyAlliance Presents:

Predicting Flight Delay Severity

A Scalable Classification Approach to Efficient Operations and Happy Customers

Phase 3 Report

Team 4-2

August 9, 2025

Table of Contents

1. Abstract

Weather, congestion, and mechanical issues often cause flight delays that can wreak havoc across flight networks, leading to missed connections, extended gate times, and overtime labor costs. According to Airlines for America [1], these disruptions cost both airlines and passengers billions of dollars annually, reducing both customer satisfation and operational resilience. To address this, we propose a predictive modeling framework for airlines joining SkyAlliance, a new airline alliance open to all US-based airlines. SkyAlliance specializes in utilizing data science and machine learning to provide unique value in the realms of customer experience, staff satisfaction, and airport logistics. By leveraging cutting edge modeling techniques, SkyAlliance will provide a data driven approach to managing disruptions caused by flight delays.

Our core dataset contains over 60 million flights from the U.S. Department of Transportation (DOT). In order to maximize our models' ability to predict delays, we joined a number of supplemental datasets to our core data. The most notable of these include datasets from the National Oceanic and Atmospheric Administration (NOAA), Federal Flights Administration (FAA), Federal Emergency Management Agency (FEMA), and supplemental datasets from DOT.

According to Airlines for America, the average cost of a flight delay is $100.80 per minute [1]. Thus we initially, we framed the challenge of predicting flight delays as a multi-class classification problem by binning delays into 4 categories based on delay length:

- No Delay (<= 15 min),
- Small Delay (15-30 min),
- Medium Delay (30-60 min), and
- Large Delay (60+ min)

By distinguishing between flight severity, our goal was to capture the necessary granularity required to differentiate between the significant operational and financial costs between a 15-minute inconvenience and a 3-hour disruption. However, experimentation revealed to us that our multi-class models struggled to generalize and thus add business value. To ensure we are meeting the needs of our partner airlines, we reframed our approach into a binary classification problem:

- No Delay (<= 15 min)
- Delay (> 15 min)

Failing to predict a delay leads to insufficient staffing, customer frustration, and cascading operational issues, while incorrectly predicting a delay has little impact on KPIs related to customer experience and airport logistics. A false positive simply ensures that airline staff is never caught off guard. Recall is thus our most important metric. While we take into consideration precision, accuracy, and f1 score, given SkyAlliance's business context, we chose to train and tune our models to maximize recall.

To maximize recall, we engineered a powerful set of features to help our models catch as many delays as possible. These features include holiday indicators, airline delay reputation, airport traffic over the previous week leading up to the flight, and whether a prior flight exists. Additionally, we ran a graph neural net to build to a 32-vector feature containing airport-specific information, number of runways and runway length.

At the heart of our feature set is a powerful graph neural network that maps the complex web of connections between airports, flights, and airlines. It distills these relationships into a 128-dimensional vector: a dense, information-rich signal that is then passed to our models. By training on our engineered features in addition to the graph vector, our models gain a deeper, more connected understanding of the flight network. Our other important features include departure hour (cyclically encoded), whether the flight departed at night, number of runways present at the airport, average daily delay minutes at the airport over the previous seven days, and whether the prior flight was delayed (respecting the two-hour mask).

To demonstrate clear value, we evaluated a suite of powerful classification models (Random Forest, XGBoost, Feedforward Neural Network (FNN) and Graph Neural Network) by benchmarking their performance against a binary logistic regression model, which serves as our baseline model.

Recall for Logistic Model			
	Train	Val	Test

Recall for Logistic Model			
No Delay	0.5849	0.6021	0.6117
Delay	0.6570	0.6132	0.6125

Our strategy is designed to create value for our partner airlines by enhancing passenger travel experience and reducing significant costs caused by unexpected delays. This report goes through the build process to generate the above results.

Thank you for partnering with SkyAlliance!

2. Data and Feature Engineering

While raw data provides the foundation for predictive modeling, feature engineering is what unlocks real performance. Simply relying on basic flight schedules and weather readings is not enough to capture the complex, interconnected drivers of flight delays. To build a high-performing model capable of generalizing to future disruptions, we engineered a rich, domain-informed feature set designed to surface signals related to weather, infrastructure, operations, and broader systemic stress.

Our model is built on two primary data sources:

- **On-Time Performance (OTP) data from the U.S. Department of Transportation,** which includes detailed information on flight times, routes, and delays.
- **Climatological records from NOAA's Integrated Surface Dataset**, which provides hourly, station-level weather data near airports.

To expand the predictive power of our features, we integrated multiple supplemental datasets:

- **FAA airport infrastructure data**, which includes characteristics such as runway count, average runway length, and the presence of operational facilities like repair services or control towers.

- **Timezone normalization tables**, used to standardize all arrival and departure timestamps to UTC, ensuring consistency across airports in different regions.
- **FEMA disaster declarations**, used to flag airports experiencing recent extreme weather or regional emergencies that could impact operations.
- **Federal holiday calendars**, to account for increased passenger volume and staffing pressure during peak travel periods.
- **DOT annual airline rankings**, which provide an aggregate measure of each carrier's historical performance, enabling us to represent reputation and systemic delay likelihood.

Collectively, this enriched feature space allows our models to capture both short-term operational pressures and long-term structural factors. In addition to these engineered features, we later leveraged a Graph Neural Network (GNN) to encode spatial and relational information between airports into a 128-dimensional embedding vector, further improving model performance through learned representations of the flight network structure.

Below we outline specifics related to the datasets we used and our feature engineering process.

2.1 Datasets

Reporting Carrier On-Time Performance (OTP)

- Source: Bureau of Transportation Statistics (BTS)
- Description: According to BTS, airline on-time performance data contains "scheduled and actual departure and arrival times", which are reported by "certified U.S. air carriers." If the air carrier accounts for at least 0.05% of domestic scheduled passenger revenues, its data is included in the dataset.
- Link: https://www.transtats.bts.gov/Fields.asp?gnoyr_VQ=FGJ (https://www.transtats.bts.gov/Fields.asp?gnoyr_VQ=FGJ)
- Snapshot of Raw OTP Month Dataset (2015-2019):

- Original number of samples: **74,177,433**
- Original number of features: **109**
- Raw dataset size: **2.74 GB**

Quality Controlled Local Climatological Data (QCLCD) Publication

- Source: National Oceanic and Atmospheric Administration (NOAA)
- Description: According to NOAA, QCLCD contains weather information from major airport weather stations in daily, hourly, and monthly intervals. The data have a large scope and include various measurements for temperature, precipitation, pressure, visibility, and atmospheric conditions.
- Link: https://www.ncei.noaa.gov/pub/data/cdo/documentation/LCD_documentation.pdf (https://www.ncei.noaa.gov/pub/data/cdo/documentation/LCD_documentation.pdf)
- Snapshot of Raw QCLCD Dataset (2015-2019):
  - Original number of samples: **898,983,399**
  - Original Number of features: **124**
  - Raw dataset size: **32.64 GB**

FAA's Airport Data and Information Portal (FAA-ADIP)

- Source: Federal Aviation Administration
- Description: ADIP contains centralized airport-related metadata, including data relevant to airport operations. The dataset conforms to FAA business rules and standards, and can be used to enrich features by providing relevant airport characteristics, such as runway length, latitude, and longitude. From this source

we pulls the data on runways and airports, which correspond to information on each of the runways at the airports or the basic information on the airport respectively.

- Link: https://adip.faa.gov/agis/public/#/public (https://adip.faa.gov/agis/public/#/public)
- Snapshot of Raw FAA-ADIP Runway Dataset:
  - Number of samples: **16,389**
  - Number of features: **135**
  - Raw dataset size: **5.62 MB**
- Snapshot of Raw FAA-ADIP Airport Dataset:
  - Number of samples: **13,223**
  - Number of features: **106**
  - Raw dataset size: **7.25 MB**

Airport Timezones

- Source: Github - Matt Johnson-Pint, Staff Software Engineer at Hypermode Inc.
- Description: This dataset was built using open source tools, including Timezone Boundary Builder (https://github.com/evansiroky/timezone-boundary-builder) and Open Street Map (https://www.openstreetmap.org/#map=5/38.01/-95.84). It includes a unique IATA code for airports across the globe, and each airport's IANA timezone, which is the format required to use pyspark.sql's to_utc_timestamp function. This dataset is available free of charge, under the terms of the MIT license.
- Link: https://gist.github.com/mattjohnsonpint/6d219c48697c550c2476 (https://gist.github.com/mattjohnsonpint/6d219c48697c550c2476)
- Snapshot of Raw Airport Timezones Dataset:

- Number of samples: **8,876**
- Number of features: **3**
- Raw dataset size: **0.38 MB**

FEMA Disaster Declaration

- Source: Federal Emergency Management Agency
- Description: Disaster Declarations Summaries provides a centralized record of all federally declared disasters, as documented by FEMA since 1953. The dataset includes all three declaration types—major disaster, emergency, and fire management assistance—and can be used to enrich features with historical and categorical information about disaster events. From this source, we pull data on disaster type, declaration date, and affected areas to incorporate context on potential operational disruptions.
- Link: https://www.fema.gov/openfema-data-page/disaster-declarations-summaries-v2 (https://www.fema.gov/openfema-data-page/disaster-declarations-summaries-v2)
- Number of samples: **68,417**
- Number of features: **28**
- Raw dataset size: **21.515 MB**

Annual Airline On-Time Rankings

- Department of Transportation
- Description: The Annual Airline On-Time Rankings dataset provides yearly on-time arrival performance percentages for U.S. marketing airlines from 2003 to 2024. Rankings include individual carrier performance as well as the industry-wide average, with certain airlines' results incorporating branded code-share partners. From this source, we pulled carrier-level on-time arrival percentages (from the previous year) to incorporate historical reliability as a feature in our models.

- Link: https://www.bts.gov/topics/airlines-and-airports/annual-airline-time-rankings-2003-2024 (https://www.bts.gov/topics/airlines-and-airports/annual-airline-time-rankings-2003-2024)
- Number of samples: **90**
- Number of features: **4**
- Raw dataset size: **0.002 MB**

US Holiday Schedule

- Kaggle
- Description: The U.S. Holidays dataset contains 18 years of holiday dates, spanning January 1, 2004, to December 31, 2021. Each record includes the holiday name, date, weekday, month, day, and year, covering major federal and cultural holidays such as Independence Day, Thanksgiving, Christmas, and Juneteenth. From this source, we pull holiday indicators to capture temporal patterns and assess the impact of holidays on travel behavior and flight delays.
- Link: https://www.kaggle.com/datasets/donnetew/us-holiday-dates-2004-2021?resource=download (https://www.kaggle.com/datasets/donnetew/us-holiday-dates-2004-2021?resource=download)
- Number of samples: **342**
- Number of features: **6**
- Raw dataset size: **0.015 MB**

2.2 Dataset Join

- Main Datasets: Carrier OTP, QCLCD Publication
- Supporting Datasets: FAA-ADIP and Airport Timezones provide features pivotal to performing our join.
- Description:

    - To create the joined dataset, we first loaded flight data from OTP, weather data from QCLCD, and airport data from FAA-ADIP. Then, we standardized and cleaned the data and performed a smaller join to incorporate the Airport Timezone dataset into the flight data. This enabled us to build the UTC timestamps, which we used along with tail number, flight date, and other fields to build unique IDs for each flight. These unique IDs and the features used in their construction are pivotal for determining whether a previous flight exists and if it was delayed. Next, we cleaned the runway data and airport data (both from FAA-ADIP), performed feature engineering, and joined these features to our main dataset.

    - We then joined weather information with our main dataset. First, we isolated station-related features such as latitude and longitude from the QCLCD weather dataset and airport latitude and longitude from the FAA-APID. We transformed these features into radians and built a haversine vector function to determine the weather station nearest to each airport. Combined with flight date, scheduled departure time, and airport site ID, we were able to use this information to incorporate weather features from the appropriate weather station into our dataset for each flight.

  Data Augmentation note:
    - The flight data from OTP uses the IATA airport code system, which is not always the same at the FAA airport location system. We manually recorded 11 airports to have the FAA code. The airport code was used to join the FAA and OTP data to add the airport longtitue and latitude so that we could join the weather data to the flight data.

- Link: 261 Phase II - Data Join (https://dbc-fae72cab-cf59.cloud.databricks.com/editor/notebooks/819275088972070?o=4021782157704243)

- Source for haversine calculation: Geospatial Nearest Neighbor Search (https://www.vibhuagrawal.com/blog/geospatial-nearest-neighbor-search)

- Snapshot of Raw Dataset:

    - Number of samples: **5,727,454**

- Number of features: **35**
- Raw dataset size: **0.429694 GB**

2.3 Feature Families and EDA of Each Family

PREDICTION/TARGET VARIABLE

`dep_delay_group`: number of minutes a flight is delayed at departure, binned into 2 classes:

- class 0: no delay (0 to 15 min)
- class 1: delay (>15 min)

Below is the distribution of our target variable on the 1 year dataset. It only includes data from 2015.

FLIGHT & AIRPORT FEATURES

`ORIGIN_AIRPORT_ID`: An identification number assigned by US DOT to identify a unique airport. Designates the airport at which flights originate.

`DEST_AIRPORT_ID`: Destination Airport, Airport ID. An identification number assigned by US DOT to identify a unique airport.

`TAIL_NUM`: Aircraft registration number is a unique alphanumeric identifier assigned to a specific aircraft.

`FL_DATE`: Flight date

`CRS_DEP_TIME`: scheduled departure time, local time zone (format: hhmm)

`CRS_ARR_TIME`: scheduled arrival time, local time zone (format: hhmm)

`DEP_TIME`: actual departure time, local time zone (format: hhmm)

`DEP_DEL15` : Departure Delay Indicator, 15 Minutes or More (1=Yes), 0 = on time, and null is cancelled

`DEP_DELAY_NEW` : Difference in minutes between scheduled and actual departure time. Early departures set to 0.

`CRS_ELAPSED_TIME` : scheduled/anticipated flight length, in minutes

OTPW Summary Metrics - Full Dataset (2015-2019)

Metric	Value
Min Date	2015-01-01
Max Date	2019-12-31
Unique Flights	41,557,594
Airline Carriers	14
Number of Airports	387
Total Delayed Flights	7,117,584
Percentage of Delayed Flights	17.13%
Total count of dropped rows	11,940,262
Total Flights after Dropped Data	29,617,332

EDA ON ONE YEAR DATASET (2015)

The combined dataset includes 5,819,079 flights across 321 airports spanning from January 1, 2015, to December 31, 2015. Focusing on on departure delays, examining patterns by origin airport and airline carrier. Southwest had the highest total number of delayed flights, followed by Delta and United. In terms of airport-specific delays, ORD (Chicago O'Hare), ATL (Atlanta), DFW (Dallas-Fort Worth), DEN (Denver), and LAX (Los Angeles) stood out. This raised the question of whether delays at these airports are primarily due to high traffic volume, weather, or internal operations.

While the majority of flights were arriving on time, approximately 20% experienced a delay of 15 minutes or more. Among those delayed, most were held up for 15 minutes or less, indicating that short delays are the most common.

Looking across airport sizes, we found that the average delay percentages are relatively similar, but medium-sized airports show much greater variability, possibly due to less buffer capacity or operational inefficiencies. We also noticed state-level differences: 7 states or territories have over 20% delayed flights, with Delaware leading at around 39%. This implies that state-level factors—such as regional weather or airspace congestion—might be relevant to include in the model.

Finally, we looked at route-level data. Among the most frequently traveled routes, delay rates ranged from 6% to 26%. The most traveled route—Los Angeles (LAX) to San Francisco (SFO) and vice versa—has about 22% and 24% of flights delayed in both directions. The route with the highest delay rate was ORD to LAX (26%), while the lowest was HNL to OGG (6%). This points to the value of engineering a route-level variable, as it captures origin-destination dynamics and potentially factors like air traffic and weather corridors.

WEATHER FEATURES

`STATION`: Station ID (string)

`DATE` (string): date with the day of month given in two digits (need for joins; cast to datetime), used for join.

`HourlyVisibility`: The horizontal distance an object can be seen and identified given in whole miles. Note visibilities less than 3 miles are usually given in smaller increments (e.g. 2.5)

`HourlyDewPointTemperature`: the temperature in whole degrees Fahrenheit at which air becomes saturated with moisture

`HourlyDryBulbTemperature`: the temperature of air measured in whole degrees Fahrenheit by a thermometer that is not affected by the moisture in the air

`HourlyWetBulbTemperature`: the lowest temperature air can reach through evaporation alone measured in whole degrees Fahrenheit. a measure of both heat and humidity

`HourlyRelativeHumidity`: a measure of how much water vapor is in the air compared to the maximum amount the air can hold at that temperature, given to the nearest whole percentage.

`HourlyWindSpeed`: Speed of the wind at the time of observation given in miles per hour (mph).

Among the weather-related variables, humidity showed the largest difference in average values between delayed and non-delayed flights. While the overall variability was similar across both groups, delayed flights tended to occur in more humid conditions on average—about 3.5 points higher. Additionally, the interquartile range (25th to 75th percentile) for delayed flights was consistently higher than that of on-time flights, suggesting that elevated humidity may be a subtle but consistent contributor to flight delays. In addition to humidity, HourlyDryBulbTemperature, HourlyWetBulbTemperature, and HourlyDewPointTemperature also exhibited relatively large differences in averages between delayed and non-delayed flights. These variables may indirectly reflect the presence of air masses or storm systems, which can impact flight operations and contribute to delays.

EDA ON ONE YEAR DATASET (2015)

The first plot shows that delays spike when visibility is at its lowest, with the highest delay counts clustered in the poorest-visibility category. The second plot reveals a clear peak in delays between 6 pm and 9 pm, after which the frequency tapers off. Overall, the hourly delay curve looks roughly bell-shaped—centered in the late evening - but with a longer "tail" stretching into the pre-dawn hours.

AIRPORT FEATURES

FAA Airport Data (FAA-APID) (join on airport code using Loc_id)

`Site_Id` : Landing facility site number. The unique identifier for the landing site.

`Loc_Id` : Location identifier—unique 3–4 character alphanumeric ID assigned to the airport.

`Longitude` : The Longitude of the airport to 7 decimal points

`Latitude` : The Latitude of the airport ot 7 decimal points

`Beacon` : Beacon lighting schedule value at the airport facilities

`Bulk Oxygen` : The type of bulk oxygen available at the airport

FAA Runway Data (FAA-APID) (join on airport code using Loc_id)

`Site_Id` : Landing facility site number. The unique identifying number of the airport whose runway is being described. Together with the runway id field, this provides the unique key to a runway record.

`Loc_Id` : Location identifier unique 3-4 character alphanumeric identifier assigned to the landing facility. (ex. 'ORD' for Chicago O'Hare)

`Runway_Id` : Runway identification. Ex. 01/19; 18L/36R (parallel runways); H1 (helipad); N/S (north/south); ALL/WAY (seaplane); B1 (balloonport)

`Length` : Physical runway length (nearest foot) (ex. 3500)

`Width` : Physical runway width (nearest foot) (ex. 100)

`Base_Obstacle_Clearance_Slope` : The clearance that is available to approaching aircraft

`Base LDA` : Declared distances Landing Distance Available, in feet.

`Base TORA` : Declared distances Take Off Run Available, in feet.

ENGINEERED FEATURES

`prior_flight_exists`: a boolean value that tells us if there was a prior incoming flight. This was based on `lag()` function for flights based on scheduled departure flight and tail number

`avg_obstacle_clearance_slope`: Based on the airports runways take the average value of the value of `base_obstacle_clearance_slope` across all runways at the departing airport in degrees.

`avg_length_of_runway`: The average of the length in feet of the runways available at departing airport

`avg_width_of_runway`: Average width of all runways in feet of the available runways at the departing airprot

`avg_landing_distance_available`: The average available landing distance for land across all runways at the departing airport in feet

`avg_take_off_distance_available`: The average available take-off distance across all runways at the departing airport in feet

`number_of_runways`: Unique count of runways at the airport

`avg_flights_last_7_day`: Average flights departed from origin in last 7 days

`avg_delays_last_7_day`: Average number of delayed flights last 7 days at orgin

`avg_arrial_last_7_day`: Average number of flights arrived at origin last 7 days

`fema_at_origin`: Boolean flag of if there was a FEMA disaster declared in the State of origin in the last 5 days

`sched_arr_night`: Boolean flag for whether a flight's scheduled arrival time was between 12:00 AM and 5:59 AM local time.

`sched_dep_night`: Boolean flag for whether a flight's scheduled departure time was between 12:00 AM and 5:59 AM local time.

`sched_arr_weekend`: Boolean flag for whether a flight's scheduled arrival day falls on a weekend (Saturday or Sunday).

`holiday_flag` : Boolean flag if there is a match to the date from US Holiday data set

`dep_hr_sin` : Sine transformation of the scheduled departure hour, scaled to a 24-hour cycle:

`arr_hr_sin` : Sine transformation of the scheduled arrival hour, using the same 24-hour cycle formula as above.

`quarter_cos` : Cosine transformation of the quarter of the year

`month_cos` : Cosine transformation of the month (1–12) into a cyclical representation

`month_sin` : Sine transformation of the day of week (1–7) into a cyclical representation

`dow_sin` : Sine transformation of the day of week (1–7) into a cyclical representation

Note: All EDA ran on 1 year of data from 2015.

2.5 Reducing Features

To understand which variables had the greatest impact on predicting flight delays, we trained a logistic regression model with Elastic Net regularization. We performed a custom grid search over both the Elastic Net mixing parameter (ranging from 0 to 1) and the regularization strength (lambda values of 0.001, 0.01, 0.1, 1, and 10). This approach allowed us to balance feature selection (via L1 regularization) with coefficient shrinkage (via L2 regularization) to prevent overfitting. By tuning across these parameters, we identified the combination that yielded the best performance on our validation sets and examined the resulting coefficients to rank features by importance. Positive coefficients indicate a higher likelihood of delay, while negative coefficients indicate a reduced likelihood of delay.

Model Tuning Approach

- **Elastic Net Tuning**
  - Range: 0 to 1 in increments of 0.1

  - 0 = Ridge (L2) Regularization
  - 1 = Lasso (L1) Regularization
- **Regularization Strength Tuning**
  - Lambda values: `[0.001, 0.01, 0.1, 1, 10]`

Best Model:

- Elastic Net = **0.1** (much closer to Ridge than to Lasso)
- Lambda = **0.001**

Important Features

Feature	Coefficient
Is there a prior flight?	0.0723
Hourly Visibility	-0.0578
Hourly Wind Speed	-0.0359
Hourly Dew Point Temperature	0.0348
Is the flight scheduled to arrive at night?	-0.0345
Hourly Dry Bulb Temperature	0.0215
Departure hour (cyclically encoded via sine)	0.0145
Avg delays at airport the previous week	-0.0126
Is the flight scheduled to depart at night?	0.0121
Number of runways at airport	0.0120

Unimportant Features

Feature	Coefficient
Day of week flight departs	-0.0012
repair_service	0.0012

Feature	Coefficient
avg_landing_distance_available	-0.0011
Previous year's OTP percentage	0.0010
Is flight scheduled to depart on weekend?	0.0008

2.6 Null & Duplicate Handling

For any predictive modeling effort to be trusted and deployed, data quality is paramount. Before feature engineering or modeling, we needed to ensure our data was clean, consistent, and free of redundancies. Duplicate records or mismatched flight counts can introduce misleading patterns that weaken model performance and damage stakeholder confidence. As part of our pipeline, we audited the data to verify flight uniqueness and handle any structural issues.

When comparing the data that is coming directly from `flights` in the raw data we found there to be 74,177,433 rows but only 42,430,589 unique flights. Based on this discrepancy, we investigated further by creating a unique id per row by combining the following fields to identify unique flights: `FL_DATE, OP_UNIQUE_CARRIER, TAIL_NUM,OP_CARRIER_FL_NUM, ORIGIN_AIRPORT_SEC`. This unique id would show all unique flights and account for any routes that happen more than once a day.

After creating this unique identifier in both datasets we see that the raw `flights` data is perfectly duplicated and so we ran .dropDuplicates() to get a unique flight count of 42,430,589.

Throughout this phase, each team member conducted independent exploratory data analysis (EDA) on the flight and weather datasets. This diversity of approaches allowed us to cross-validate findings and surface a broader set of patterns. In the remainder of this EDA section, "OTPW" refers to our joined Ontime Performance and Weather dataset, created by merging flight records with station-level weather data based on airport proximity and departure timing.

Nulls & Data Cleaning in Flight Data

Because we did our own join we dropped the nulls before joining in the weather data.

Feature	Null Count	Dropped/Kept	Note
ORIGIN_AIRPORT_ID	0	N/A	
DEST_AIRPORT_ID	0	N/A	
TAIL_NUM	242,827	Dropped	
FL_DATE	0	N/A	
CRS_DEP_TIME	0	N/A	
CRS_ARR_TIME	0	N/A	
DEP_TIME	852,812	Dropped	Corresponds to Cancellation
DEP_DEL15,	857,939	Dropped	Corresponds to Cancellation
DEP_DELAY_NEW	857,939	Dropped	Corresponds to Cancellation
CRS_ELAPSED_TIME	170	Dropped	Corresponds to Cancellation
DISTANCE	0	N/A	Corresponds to Cancellation

Additionally from research we dropped Flights from ISN (Sloulin Field International Airport) because it perpanantly closed in 2019, which was drop of 7,594

Nulls & Data Cleaning in Weather Data

The weather data was also cleaned after the join to the flight data

Feature	Null Count	Dropped/Kept	Note
station	0	N/A	
date	0	N/A	
HourlyVisibility	10,640,927	Dropped	Smaller Stations don't always report
HourlyDewPointTemperature	10,656,042	Dropped	Smaller Stations don't always report

Feature	Null Count	Dropped/Kept	Note
HourlyDryBulbTemperature	10,641,300	Dropped	Smaller Stations don't always report
HourlyWetBulbTemperature	121,017	Dropped	Smaller Stations don't always report
HourlyRelativeHumidity	11,272,772	Dropped	Smaller Stations don't always report
HourlyWindSpeed	10,652,741	Dropped	Smaller Stations don't always report

Nulls & Data Cleaning in Airport Data

Feature	Null Count	Dropped/Kept	Note
site_id	0		
loc_id	0		
longitude	0		
latitude	0		

Nulls & Data Cleaning in Runway Data

The features that are numeric they were all used for averages so the nulls didn't factor into the calculation

Feature	Null Count	Dropped/Kept	Note
Site_Id	0		
Loc_Id	0		
Runway_Id	0		
Length	0		
Width	0		

Feature	Null Count	Dropped/Kept	Note
Base_Obstacle_Clearance_Slope	9009	Dropped	Not all runways have obstacles
Base LDA	14,983	Dropped	Some smaller airports don't report to this level
Base TORA	14,981	Dropped	Some smaller airports don't report to this level

2.7 Train, Validation, & Test Dataset Summary

Below outlines the final total counts

Extra Credit Dataset Join

- rows: 60,186,267
- columns: 71

Training Dataset

Prior to post rebalancing

- rows: 8,007,608 (64%)
- features: 153

Validation Datasets (built from rebalanced training set)

fold 1:

- dim fold 1 train: (1,355,198, 4)
- dim fold 1 val: (285,775, 4)

fold 2:

- dim fold 2 train: (1,238,296, 4)
- dim fold 2 val: (339,218, 4)

fold 3:

- dim fold 3 train: (1,263,828, 4)
- dim fold 3 val: (348,734, 4)

fold 4:

- dim fold 4 train: (1,187,590, 4)
- dim fold 4 val: (294,570, 4)

fold 5:

- dim fold 5 train: (1,354,927, 4)
- dim fold 5 val: (331,244, 4)

Testing Dataset

- rows: 6,861,207 (31%)
- features: 153

3. Modeling Pipeline

To build robust and scalable machine learning models, we designed a modular pipeline that prioritizes reproducibility, data integrity, and experimentation flexibility. This structured process allowed us to move from raw flight data to deployable models in a controlled and transparent manner, ensuring that each stage was verifiable and contributed meaningfully to predictive performance.

Our pipeline spans the full machine learning workflow:

- It begins with loading and cleansing the data, ensuring nulls and duplicates are handled and data types are properly cast.
- Next, we perform domain-driven feature engineering, extracting key signals such as prior flight delay status, cyclical time encodings, and airport operational characteristics.
- We then apply encoding and balancing strategies to prepare the dataset for training, while preserving interpretability and fairness across classes.
- Finally, we vectorize, train, evaluate, and tune a suite of models, including Random Forest, XGBoost, Feedforward Neural Networks, and a Graph Neural Network for embedding structural airport data.

Each major step includes intermediate checkpoints to enable versioning, error recovery, and experimentation. This pipeline allowed us to build high-performing models while maintaining transparency and traceability — two key pillars in any operational machine learning deployment. This is the area in which we did the most experimentation as well.

3.1 Modeling Pipeline Steps & Diagram

This section includes planned pipeline steps.

- Load Data
- Data Processing
  - Feature Selection
  - Drop Duplicates & Nulls
  - Combine Airlines
  - Typecasting
  - UTC Timestamp Conversion
  - **Checkpoint Process Data**
- Feature Egineering
  - Previous Flight Delay
  - Time-Based Features
  - Predicted Delay Category
  - Drop un-needed Features
  - **Checkpoint Data**
- Feature Encoding
- Encode Target Variable
- One-Hot Encoding
- Binarize Tail Number
- Cyclical Encoding
- Select Features
- **Checkpoint Encoded Data**
- Train/Test Split
- Scale Numeric Values
  - **Checkpoint train/test split**
- Rebalance Dataset
  - Downsample Majority Class

    - Upsample Delay Classes
    - Check Balanced DF
    - **Checkpoint Balanced Dataset**
- Vectorize Features
    - **Checkpoint FLAT train/test Datasets**
    - **Checkpoint Vectorized Data**
- Train Models
- Test and Validate Models
- Build Models
- Tune Models
- Pass to Deployment Team

3.2 Training & Cross-Validation Data Splits

To simulate real-world deployment and avoid future-looking bias, we trained our models on flight and weather data from 2015–2018 and tested them on a holdout set from 2019 (6.9M examples). This time-aware split mirrors how airlines must make predictions on upcoming flights based only on past patterns.

Within the training window, we used a Blocked Time Series Cross Validation strategy to validate model performance over time. By dividing the training data into 5, non-overlapping, time-ordered folds (80% train, 20% validation), we preserved the natural temporal structure of flight delays. This approach ensures we evaluate models in the same way they would be used in production — predicting future events without hindsight.

Each fold was scaled and encoded independently and downsampled to address the natural class imbalance (82% on-time, 18% delayed). This rigorous approach helped us select the most generalizable models, reduce overfitting, and improve confidence in real-world performance.

3.3 Feature Scaling & Standardization

To ensure our models treat all numeric features fairly — regardless of their original scale — we applied two different types of standardization: MinMax Scaling and Z-Score Normalization, depending on the feature's context and distribution.

- **MinMax Scaling** rescales values to a [0, 1] range and was applied to infrastructure-related features like:

  `ELEVATION`
  `avg_obstacle_clearance_slope`
  `number_of_runways`

These features benefit from normalization without altering relative relationships.

- **Z-Score Standardization** (mean-centered with unit variance) was applied to operational and weather-related variables with wider ranges and natural variability:

  Weather:
  `HourlyDewPointTemperature`, `HourlyDryBulbTemperature`, `HourlyRelativeHumidity`, `HourlyVisibility`, `HourlyWetBulbTemperature`, `HourlyWindSpeed`

  Flight: `CRS_ELAPSED_TIME`, `DISTANCE`

  Runway: `avg_length_of_runway`, `avg_width_of_runway`, `avg_landing_distance_available`, `avg_take_off_distance_available`

To avoid data leakage and preserve time-based integrity, we scaled the training and validation data separately within each cross-validation fold. We then applied the final scaler from the full training data to the test (and extra credit) sets, ensuring a consistent and realistic evaluation environment.

This step is essential for both model fairness and performance — allowing our models to learn true signal rather than being skewed by differing feature scales.

3.4 Feature Engineering: Graph Neural Net

Reference Paper (https://medium.com/stanford-cs224w/graphing-the-skies-timely-flight-delay-prediction-using-spatio-temporal-gnns-879854f204ed)

We then used this information to build a Graph Neural Net, a tool at our disposal that is designed for measuring relationships between data points. On this Graph we ran a Neural Network to allow us to enhance our features about airports and flight patterns instead of using the data directly. While relationship could be modeled with specific features we wanted to leverage a Graph to model connections between airports, flights and carriers

In our case we used Airports as Nodes and Flights as Edges, here is a single example of a Node and Edge to show how it works under the hood.

Our specific implementation focused on edge classification, predicting the likelihood of delay for a given flight based on both node features (airport characteristics) and edge features (flight-specific details). The output of the GNN was a set of learned embeddings that summarize network-level patterns, which we then integrated into downstream models to enhance predictive performance, rather than using the GNN as the sole predictor.

Loss Function

Binary Cross-entropy (log-loss) function for the two classes “No Delay” (0) and “Delay” (1):

$$\mathcal{L} = -\frac{1}{N}\sum_{i=1}^{N}\left[y_i \log(p_i) + (1 - y_i)\log(1 - p_i)\right]$$

Model Configuration

- Optimizer: Adam
- Device: CPU
- Epochs: 80
- Loss: 0.472

- Hidden LayerS: 64

Output

A 128-dimension layer of patterns. This was then mapped back to train and test dataset. If that route was not present in the test dataset an average across the dimensions was taken and applied.

3.5 Dataset Rebalancing

Downsampling

In our training dataset, the target variable `target_bin` (0 = No Delay, 1 = Delay) was imbalanced, with significantly more "No Delay" flights than "Delay" flights. Class imbalance can bias the model toward predicting the majority class, resulting in poor recall for the minority class (delays).

To address this, we applied **random downsampling** to the majority class:

1. **Class distribution check** – We calculated the number of records in each class and their percentages.
2. **Identify majority/minority** –
   - Majority label: `0` (No Delay)
   - Minority label: `1` (Delay)
3. **Down-sample ratio** – We computed the ratio of minority to majority records and used it as the sampling fraction for the majority class.
4. **Random downsampling** – We randomly sampled from the majority class (without replacement) to match the number of minority records.
5. **Combine datasets** – The downsampled majority set was combined with the minority set to create a **balanced dataset** for model training.

This ensured that each class contributed equally during training, improving the model's ability to detect delays while avoiding an over-representation of "No Delay" flights.

3.6 Feature Vectorization

Several features in our dataset were stored as Boolean values (True/False). Since machine learning models in Spark require numerical inputs, we cast these Boolean columns to numeric (`0.0` or `1.0`) for compatibility:

- `holiday_flag` – Indicates if the flight date was a holiday.
- `prior_flight_exists` – Whether an incoming flight to the same aircraft existed.
- `prior_flight_dep_delay` – Whether the prior flight was delayed at departure.
- `fema_at_origin` – Whether a FEMA disaster declaration was in effect at the origin airport.

Once all features were numeric, we used Spark's `VectorAssembler` to combine the individual feature columns into a single vector column called `features`. This vectorization step is required by Spark ML models, as they expect the feature set to be stored in a single vector field rather than multiple separate columns.

Vectorization improves pipeline efficiency by:

1. **Standardizing input format** – All models can be trained with a consistent single `features` column.
2. **Enabling scalable transformations** – Vector operations can be performed in parallel across the cluster.
3. **Preserving schema consistency** – Ensures that training, validation, and test sets share the exact same feature ordering.

3.7 Data Leakage Prevention

Data leakage occurs when a machine learning model is accidentally trained on information that it will not have access to when it makes predictions in the real world. It's like a student cheating on a test. If they already saw the answers, they might ace the exam, but they won't perform well in the real world. Models are the same. This leads to the development of models that appear highly accurate during testing but fail to perform when deployed, resulting in wasted resources and a loss of confidence in the model's utility.

Data leakage can be categorized into two distinct types:

1. **Target Leakage (Leaky Predictors)**
   Target leakage occurs when a predictor feature in the dataset contains information that is directly tied to the target variable but would not be known at the time of an actual prediction. The leak comes from the data itself. This form of leakage is characterized by the inclusion of data that was only available after the outcome you are trying topredict has already occurred. The model learns a shortcut by correlating this "post-event" data with the outcome, rather than learning the true underlying patterns from the predictive features.

2. **Train-Test Contamination (Leaky Validation)**
   Train-Test Contamination is a methodological error where the training process inadvertently learns from the data that is supposed to be reserved for testing and validation. The leak comes from the modeling procedure, not the features themselves. This leakage stems from performing data preparation and transformation steps before splitting the data into separate training and testing sets. When this happens, statistical information from the test set (e.g., its mean, minimum, or maximum values) bleeds into the training set, giving the model an unrealistic preview of the data it will be tested on.

Data Leakage Prevention Strategies Used:

To ensure the integrity of the model and the reliability of its performance metrics, our data pipeline incorporates a rigorous, multi-layered strategy to prevent data leakage. The following methods were systematically applied:

Technique	Implementation Details	Purpose & Impact
Chronological Data Splitting	The dataset was partitioned by date: the training set contains data from 2015-2018, and the test set contains data from 2019.	Simulates a real-world prediction task (past predicting future) and prevents the model from learning from the evaluation period.
Isolated Preprocessing	Scalers (`MinMaxScaler`, `StandardScaler`) and the GNN model were fit/trained exclusively on the training data, then used to transform all data splits.	Prevents train-test contamination, ensuring no statistical information from the test set influences the model's training.

Technique	Implementation Details	Purpose & Impact
Time-Aware Cross-Validation	The training data was divided into 5 folds ordered by time, with validation always occurring on data chronologically after the training data within each fold.	Ensures that the hyperparameter tuning process is robust and free from leakage, maintaining the temporal integrity of the data.

No data leakage occurred for the 2019 blind test set. However, during EDA on the 3 month (Jan - March 2015) and 1 year (2015) datasets, we performed EDA on the entire data before splitting it into separate training, validation, and test sets. Based on our EDA of these time periods, we decided to combine airlines that were bought by others into a single category. For instance, Virgin America was acquired by Alaska Air Group in April 2016 [2], so we reassigned all Virgin America to Alaska Airlines. Similarly, American Airlines and US Airways merged in 2013 [2], so we combined these air carriers under American Airlines in our dataset. While the choice to combine airlines in this way is not incorrect, it is still technically an example of data leakage because these decisions were not exclusively made using an isolated training set. However, we include an unknown airline category in our data processing pipeline to ensure the pipeline can handle new, unseen data. Additionally, the final test set (2019) was not included in this decision making process.

4. Our Alogirthms

4.1 Cluster Size

All algorithms for phase 3 were trained and evaluated on data from 2015-2019 using a cluster with the following specifications:

- Driver: Databricks m5d8XL (128GB, 32 vCPUs)
- Workers: 6-12 r5dxl workers (32 GB, 4 vCPUs each)
- Total Cluster Size: 320GB - 512 GB RAM with 56-80 vCPUs

4.2 Baseline Model: Binary Logistic Regression

Overview

Given the high cost of missing a delayed flight, our primary evaluation metric is recall, particularly for the delayed category. This makes a most-frequent-class baseline model (which would always predict "no delay") inappropriate for our use case, as it would achieve zero recall on the class we care about most. Instead, we chose to define our baseline using **binary logistic regression**, a simple yet effective linear classifier that enables learning class probabilities and separating decision boundaries across multiple delay categories.

Model Run Time

- The 5-fold cross-validation baseline model took 23 minutes to run on the full training with cross validation folds dataset.
- The test baseline model took 3 minutes to run on the full test dataset.

Loss Function

We trained our baseline model to minimize the binary log-loss, defined as:

$$\mathcal{L} = -\frac{1}{N}\sum_{i=1}^{N}[y_i \cdot \log(p_i) + (1 - y_i) \cdot \log(1 - p_i)]$$

Where:

- N is the number of samples
- y_i is the true label for sample i (0 for *No Delay*, 1 for *Delay*)
- p_i is the predicted probability that sample i belongs to the *Delay* class This formulation ensures the model is trained to **maximize the probability of correct classification**, while penalizing confident wrong predictions more heavily.

Experiment: Regularization

Class	Metric	Vanilla Regression	ElasticNet (0.5, 0.01)	% Change
0	Recall	0.6117	0.5356	-12.45%

Class	Metric	Vanilla Regression	ElasticNet (0.5, 0.01)	% Change
0	Precision	0.8764	0.6057	-30.89%
0	F1	0.7205	0.5685	-21.09%
1	Recall	0.6125	0.6990	+14.11%
1	Precision	0.2600	0.6356	+144.46%
1	F1	0.3650	0.6658	+82.14%

Switching from vanilla logistic regression to ElasticNet (α=0.5, λ=0.01) produces clear trade-offs. Results shown are performance on the test dataset (see notebooks in appendix for CV/test results). For the No Delay class, all metrics decline — recall (–12.45%), precision (–30.89%), and F1 (–21.09%) — indicating weaker performance in correctly identifying no-delay flights. For the Delay class, recall improves (+14.11%) and precision more than doubles (+144.46%), resulting in a substantial F1 gain (+82.14%). Overall, ElasticNet greatly strengthens the model's ability to accurately identify delayed flights, but at the cost of reduced accuracy for no-delay predictions.

Experiment: Threshold

Class	Metric	Threshold = 0.5	Threshold = 0.4	% Change
0	Recall	0.6117	0.4042	-33.93%
0	Precision	0.8764	0.9007	+2.77%
0	F1	0.7205	0.5580	-22.56%
1	Recall	0.6125	0.8000	+30.58%
1	Precision	0.2600	0.2302	-11.46%
1	F1	0.3650	0.3575	-2.05%

Lowering the decision threshold from 0.5 to 0.4 shifts the model toward predicting more delays. Results shown are performance on the test dataset (see notebooks in appendix for CV/test results). For the No Delay class, recall drops sharply

(–33.93%) and F1 declines (–22.56%), while precision improves slightly (+2.77%), indicating a stricter standard for labeling flights as no delay. For the Delay class, recall improves substantially (+30.58%), but precision decreases (–11.46%) and F1 falls slightly (–2.05%). Overall, lowering the threshold increases sensitivity to delays at the expense of more false positives for the delay class and more missed no-delay flights.

More information regarding feature-related experiments is available in sections 5.4 and 8.1

4.3 XGBoost Classifier

Implementation Overview: The XGBoost model builds an ensemble of decision trees, where each new tree learns from the mistakes of the previous ones—gradually improving overall performance. It includes both L1 and L2 regularization to help prevent overfitting, making it well-suited for complex, high-dimensional data. XGBoost also handles sparse data efficiently, which aligns well with our feature engineering approach that included one-hot encoding of categorical variables. To further improve model performance, we applied hyperparameter tuning, specifically optimizing the learning rate and tree depth.

Loss Function

We use the `binary:logistic` loss function in XGBoost for this task, as our target variable is now a **binary classification problem**: *No Delay* vs. *Delay*. This loss function is based on **binary cross-entropy (log loss)**, which is commonly used when the target has only two classes. It works by minimizing the difference between the predicted probabilities and the actual binary labels, encouraging the model to assign high probability to the correct class. The **binary log-loss** is defined as:

$$\mathcal{L} = -\frac{1}{N}\sum_{i=1}^{N}[y_i \cdot \log(p_i) + (1 - y_i) \cdot \log(1 - p_i)]$$

Where:

- N is the number of samples
- y_i is the true label for sample i (0 for *No Delay*, 1 for *Delay*)
- p_i is the predicted probability that sample i belongs to the *Delay* class This formulation ensures the model is trained to **maximize the probability of correct classification**, while penalizing confident wrong predictions more heavily.

Experiments

We ran 99 experiments (not all of the possible combinations were run due to lengthy computation time) during hyperparameter tuning to identify the optimal model configuration, using the dataset from 2015. The initial tuning phase took ~3 hours, with 3-fold cross-validation over the following parameter grid:

- "max_depth": [8]
- "learning_rate": [0.05, 0.07]
- "subsample": [0.75, 0.95]
- "gamma": [0.0, 2.5, 5.0],
- "reg_alpha": [0.15, 0.5, 0.95],
- "reg_lambda": [2.5, 5, 7.5]

Best configuration:

- "max_depth": [8]
- "learning_rate": [0.05]
- "subsample": [0.75]
- "gamma": [2.5],
- "reg_alpha": [0.95],
- "reg_lambda": [2.5]

The best configuration was then trained on the data spanning 2015 to 2018, and evaluated on the 2019 data. All experiments used the same input feature set, vectorized via VectorAssembler prior to training. It is important to note that the additional 128 dimension vector from the Graph Neural Net was included in this feature set. No feature subsets were altered between models or hyperparameter tuning.

Training Time

Training the best model configuration on the 4 year 2015-2018 training set and evaluating on the 4 year 2015-2018 training set and the 1 year 2019 testing set took 5 minutes.

4.4 Feed Forward Neural Network

Feed forward neural networks are effective at capturing non-linear patterns in the data which may not be apparent through otherwise capable linear methods like boosting. Unlike recurrent neural networks and long short-term memory networks, FFNNs can be trained in parallel which makes them more appropriate for this problem.

Loss function used:

$$\mathcal{L} = -\frac{1}{N}\sum_{i=1}^{N}[y_i \cdot \log(p_i) + (1 - y_i) \cdot \log(1 - p_i)]$$

Where:

- N is the number of samples
- y_i is the true label for sample i (0 for *No Delay*, 1 for *Delay*)
- p_i is the predicted probability that sample i belongs to the *Delay* class This formulation ensures the model is trained to **maximize the probability of correct classification**, while penalizing confident wrong predictions more heavily.

Number of experiments conducted:

Fixed Parameters:

- tol = 1e-5, features used
- Justifications:
  - Used same features as described earlier in the report, in order to have meaningful performance comparison against other types of models.

Adjusted Parameters:

- Epochs Trained: 10, 25, 50.
- stepSize = 0.003, 0.001, 0.0001

- blockSize = 512, 1024
- Layer Structures: [Input, Input//2, Output], [Input, Input, Output], [Input, Input//2, Input//4, Output], [Input, Input, Input//2, Output]

Experiments: (Epochs, Tolerance, BlockSize, Learning Rate, Layer Structure)

- (50, 1e-05, 512, 0.003, [154, 77, 2])
- (50, 1e-05, 512, 0.003, [154, 154, 2])
- (25, 1e-06, 1024, 0.001, [154, 154, 2])
- (50, 1e-05, 1024, 0.0001, [154, 154, 2])
- (10, 1e-05, 1024, 0.0001, [154, 154, 2]) **BEST MODEL**
- (50, 1e-05, 512, 0.003, [154, 77, 38, 2])
- (25, 1e-06, 512, 0.003, [154, 154, 77, 2])
- (50, 1e-05, 1024, 0.0001, [154, 154, 77, 2])

Training Time:
- Approximately 3 minutes for CV on the full dataset.
- 1 minute to fit all training data without CV.
- 20s to make predictions on test

More information on the best model can be found here: https://dbc-fae72cab-cf59.cloud.databricks.com/editor/notebooks/3600546862782352?o=4021782157704243#command/3600546862782381 (https://dbc-fae72cab-cf59.cloud.databricks.com/editor/notebooks/3600546862782352?o=4021782157704243#command/3600546862782381)

Hyperparameters were tuned manually, saving each experiment as an independent notebook. The process was the start running all 4 layer architectures using identical hyperparameters (50, 1e-05, 512, 0.003), and evaluate which models held the most promise. After initial exploration, architectures [154, 154, 2] and [154, 154, 77, 2] were the best performers. Further experiments were conducted on these two by

increaseing the blocksize, and varying both learning rate and number of training epochs. Finally [154, 154, 2] displayed the best performance with a small learning rate of 0.0001 and limiting training epochs to 10 in order to mitigate overfitting.

4.5 Experimentation with Other Models

We tried to build out other models but due to time and results we decided to focus on other models. But below outlines the process used for DBSCAN and Coefficient Weighted Random Forest that was attempted. This was only done on the multi-categorical stage.

DBSCAN > COWRF for multi-categorical classification

Step 1 (to be implemented in Phase 3): Using Graph Neural Net (this would be our graph component) to generate a 32-dim vectory of airports Nodes: Airports Edges = flight connections, shared aircraft, delay propagation, other components

supporting sources: https://medium.com/stanford-cs224w/graphing-the-skies-timely-flight-delay-prediction-using-spatio-temporal-gnns-879854f204ed (https://medium.com/stanford-cs224w/graphing-the-skies-timely-flight-delay-prediction-using-spatio-temporal-gnns-879854f204ed) https://medium.com/@ajmal.t.aziz/using-mlflow-to-deploy-graph-neural-networks-for-monitoring-supply-chain-risk-644c87e5259e (https://medium.com/@ajmal.t.aziz/using-mlflow-to-deploy-graph-neural-networks-for-monitoring-supply-chain-risk-644c87e5259e)

This would require the use of graph-frames, Pytorch + PyTorch Geometric, or GDL

Output: 32-dim vector to store in a dataframe

Training Rows: 2,840,435

Step 2: DBSCAN

Supporting source: https://www.nature.com/articles/s41598-024-55217-z (https://www.nature.com/articles/s41598-024-55217-z)

To make this run what I had to do was to create multiple partitions in the data to avoid a full Cartesian join. I used the longitude and latitude of the airports and then made sure they were balanced by category which I labeled `fine_partition_key`. This allowed us to generate the clusters more efficients.

- Epsilon for Neighborhood Radius: 1.0
- Minimum neighbors = 10
- If fewer than 5 rows per group drop the group

To then use the clusters on new data a function was created to create the centroid of each cluster, then when unseen data is brought in the centroid is calculated and the cluster applied by finding the nearest cluster.

Because DBSCAN creates groupings of data we get more rows afterwards **After DBSCAN rows: 7,410,631**

After Removing Data that doesn't fit into cluster: 7,070,644

Step 3: COWRF (Cluster-Oriented Weighted Random Forest)

Supporting source: https://www.nature.com/articles/s41598-024-55217-z (https://www.nature.com/articles/s41598-024-55217-z)

The COWRT is designed to run by each cluster. In this case we have 4 clusters created from DBSCAN and each of those then predicts into the 4 delay categories, the Weighted Random Forest runs on each clusters and predicts each of the target classes by cluster. The outcomes are then joined toghether to be able to evaluate the materics by each of the target classifications.

Number of Trees: 50

5. Results and Discussion

Our models will be compared by their validation-set performance on the binary cross-entropy (log-loss) function for the two classes “No Delay” (0) and “Delay” (1):

$$\mathcal{L} = -\frac{1}{N}\sum_{i=1}^{N}\left[y_i \log(p_i); +; (1 - y_i) \log(1 - p_i)\right]$$

While we report recall, precision, and F1 score for both classes, we will focus on recall for the **Delay** class as our key interpretation metric.

Recall (Sensitivity or True Positive Rate) for each class is defined as:

$$\text{Recall}_{\text{No Delay}} = \frac{\text{True Negatives}}{\text{True Negatives} + \text{False Positives}}, \quad \text{Recall}_{\text{Delay}} = \frac{}{\text{True P}}$$

In evaluating our classification models, we aim to maximize the recall for the delay class since our business goal is to catch as many delayed flights as possible so the airline alliance can proactively allocate customer service resources (rebooking agents, ground staff, communications, etc.).

5.1 Results for Training Dataset (2015-2018)

The best model configurations were trained on the 4 years of training data between 2015 to 2018. They were then evaluated on the same training set.

Model	Classification	Recall	Precision	F1
Logistic Regression	No Delay	0.5849	0.6304	0.6068
	Delay	0.6570	0.6127	0.6341
XGBoost	No Delay	0.6307	0.6448	0.6377
	Delay	0.6532	0.6392	0.6461
Feed Forward Neural Net	No Delay	0.1583	0.5191	0.2426
	Delay	0.8533	0.5033	0.6332

5.2 Results for 5-Fold Cross Validation

The best model configurations were trained on the 4 years of training data between 2015 to 2018. They were then evaluated on the same training set.

Model	Classification	Recall	Precision	F1
Logistic Regression	No Delay	0.6021	0.6094	0.6057
	Delay	0.6132	0.6059	0.6095
XGBoost	No Delay	0.6315	0.6316	0.6265
	Delay	0.6137	0.6227	0.6111
Feed Forward Neural Net [154, 154, 2]	No Delay	0.4648	0.5231	0.3840
	Delay	0.5415	0.4838	0.3683

5.3 Results for Test Dataset (2019)

The best models were then tested on the data spanning the year 2019. The discussion of these results are below.

Model	Classification	Recall	Precision	F1
Logistic Regression	No Delay	0.6117	0.8764	0.7205
	Delay	0.6125	0.2600	0.3650
XGBoost	No Delay	0.4553	0.8976	0.6041
	Delay	0.7668	0.2387	0.3640
Feed Forward Neural Net	No Delay	0.1655	0.8336	0.2762
	Delay	0.8517	0.1852	0.3042

5.4 Results Analysis and Discussion

Throughout our pipeline, we prioritized Weighted Recall as the primary evaluation metric, as it most directly supports SkyAlliance's objective: predicting whether a flight will be delayed to ensure operational efficiency. Recall represents the percentage of flights that were actually delayed, out of all the flights the model predicted to be delayed. In the context of SkyAlliance, it's crucial for the model to

achieve strong recall across all delay categories to support accurate forecasting and preparation. We placed particular emphasis on recall for delay class, since this is what will most likely trigger high-cost service interventions such as rebooking, lounge access, or additional customer support. By prioritizing performance in this categories, we aim to ensure the model supports real-world operational decisions and enables more proactive, efficient resource planning.

Logistic Regression

We began by evaluating the Logistic Regression model as our baseline. On the training set, the model demonstrated reasonably balanced performance across both classes, with recall of 58% for No Delay and 66% for Delay. This pattern held relatively steady during cross-validation, where the model achieved recall values of 60% and 61% for No Delay and Delay, respectively—suggesting it generalized well and avoided overfitting. On the 2019 test set, the model maintained strong performance. It correctly identified 61% of on-time flights (No Delay) with an impressive precision of nearly 88%, meaning when the model predicts that a flight will not be delayed, it is highly likely to be correct. For actual delays, the model achieved a recall of 61%, indicating it successfully flagged the majority of delayed flights. While the precision for this class was lower at 26%, this tradeoff is acceptable within the SkyAlliance context, where the cost of missing a delay (false negative) outweighs that of over-preparing (false positive). **Overall, the Logistic Regression model offers strong generalizability, reliable recall in both classes, and high precision when predicting on-time flights. These attributes make it a dependable and interpretable option for deployment—particularly in scenarios where clarity and robustness are priorities.**

Feature Importance for Logistic Regression

Since Linear Regression turned out to be our recommended model for deployment, we wanted to understand which features are most important for this model in particular. Unfortunately, due to limitations of MLLib, the only way to understand which features are most important to the model is to reverse-engineer the VectorAssembler. This makes it very difficult to understand the importance of the

graph neural net vector for this model. However, when it comes to the other features we engineered, the ranking is as follows (in descending order by absolute value):

Feature	Coefficient
prior_flight_exists	-0.5784
dep_hr_sin	-0.5570
sched_dep_night	-0.3097
sched_arr_night	0.3050
prior_flight_dep_delay	-0.3046
month_cos	-0.2662
HourlyRelativeHumidity	0.2445
arr_hr_sin	-0.2337
holiday_flag	-0.1311
HourlyVisibility	-0.1166
fema_at_origin	0.1135
HourlyDewPointTemperature	-0.1099
quarter_cos	0.08904
HourlyWindSpeed	0.07359
arr_hr_cos	0.06627
ELEVATION	0.05656
HourlyWetBulbTemperature	-0.05621
HourlyDryBulbTemperature	0.04692
dow_sin	-0.03989
month_sin	0.02208

Feature	Coefficient
DISTANCE	0.01897
quarter_sin	-0.01765
CRS_ELAPSED_TIME	0.01422
avg_flights_last_7_days	-0.0002599
avg_delays_last_7_days	0.00006203
avg_arrivals_last_7_days	-0.000000477

Interestingly, whether a prior flight exists is a stronger indicator than whether that flight was delayed. This likely is due to the two-hour mask, which hides most of the prior flight delays from the model. Departure hour (cyclically encoded) is also a very strong indicator, as well as flights that are scheduled to depart or arrive at night. Night time conditions contribute heavily to delays, possibly due to visibility conditions and/or staffing shortages during typical off-hours. Noticeably, an airport's historical delay context over the past week does not contribute meaningfully to the model. However, after further analysis, this is likely because `avg_flights_last_7_days`, `avg_delays_last_7_days`, and `avg_arrivals_last_7_days` were not scaled properly in our data processing pipeline. We attempted to go back and fix this error, but ran out of time to implement the fix for all models. When running a regularized logistic regression model with these features properly scaled (but without the graph feature included in the run), the absolute values of the coefficients for these features grow dramatically:

Feature	Coefficient
avg_delays_last_7_days	0.2402
avg_arrivals_last_7_days	-0.1377

In this experiment, the coefficient for `avg_flights_last_7_days` was pushed to 0 due to regularizaiton.

In order to overcome the obstacle of reverse-engineering VectorAssembler output on a 128-dimension vector while still getting a sense for the importance of our graph feature, we ran an identical logistic regression model configuration with and without the graph feature.

Class	Metric	With Graph	Without Graph	% Change
0	Recall	0.6117	0.5559	-9.12%
0	Precision	0.8764	0.8830	+0.75%
0	F1	0.7205	0.6823	-5.30%
1	Recall	0.6125	0.6693	+9.27%
1	Precision	0.2600	0.2513	-3.35%
1	F1	0.3650	0.3654	+0.11%

Overall, not including the graph feature shifts toward better detection of delays at the cost of missing more no-delay flights, with minimal change in overall balance for the delay class.

XGBoost

The XGBoost model showed strong, balanced performance on the training set, with recall of 63% for No Delay and 65% for Delay, and precision around 64% for both. Cross-validation results were similar, suggesting the model captured patterns without overfitting. On the test set, XGBoost improved Delay recall to 77%, outperforming the baseline Logistic Regression model (61%). However, this gain came at the cost of lower Delay precision (24%), meaning it frequently predicted delays when none occurred—leading to potential over-preparation. While this may increase unnecessary costs, the high sensitivity is valuable for ensuring SkyAlliance is prepared when delays happen. For No Delay, recall fell to 45% but precision rose to nearly 90%, indicating the model predicts on-time departures only when highly confident. Compared to the baseline, XGBoost detects more true delays but with more false positives. The drop in consistency from training/validation to test results suggests possible overfitting, limiting confidence in its stability for future years. For this reason, we do not recommend deploying XGBoost in its current form.

Feedforward Neural Network

The Feedforward Neural Network achieved the highest recall for the Delay class on the test set (85%), surpassing both the baseline Logistic Regression model (61%) and XGBoost (77%). However, its Delay precision was just 18.5%—essentially equivalent to random guessing delays (18.2%). Performance on the No Delay class was also weak, with recall at only 17% and precision at 83%, meaning the model rarely predicts on-time flights and does so only when highly confident. Compared to the baseline, FFNN creates a sharper imbalance between classes: it excels at detecting delays but offers poor reliability for identifying flights that will arrive on time. Given this asymmetry, the low Delay precision, and the added model complexity, FFNN is not well-suited for SkyAlliance's needs, as its inability to accurately flag No Delay cases would drive significant unnecessary operational costs and undermine the goal of improving efficiency during true delays.

6. Conclusion

6.1 Major Takeaways

This project focuses on predicting airline delays and their severity, enabling carriers to shift from reactive problem-solving to a predictive operations model. By anticipating the scale of a delay, airlines can take proactive measures to minimize customer frustration, such as providing timely updates, offering lounge access or meal credits, and optimizing baggage routing and scheduling desk staffing.

Our hypothesis was that by framing delays as a time series problem, machine learning algorithms could forecast the occurrence and severity of flight delays with reasonable accuracy. Since our goal was to enable airlines to take preemptive action, we focused on predicting multiple categorical outcomes for our models. We initially attempted to predict four categories of delay (No Delay, Small Delay, Medium Delay, Large Delay). After limited success, we pivoted to a binary classification of Delay versus No Delay, where a delay was defined as any flight departing more than 15 minutes after its scheduled departure time.

We cleaned and enriched our dataset with external sources to engineer important signal features for our models. We experimented with various algorithms, including Logistic Regression, XGBoost, Random Forest, Feedforward Neural Networks, DBSCAN, and Weighted Coefficient Random Forest. After initial experimentation, we narrowed our focus to fine-tuning Logistic Regression, XGBoost, and Feedforward Neural Network models. We also incorporated a Graph Neural Network at the feature level to generate pattern embeddings based on airport and flight connections, enhancing the feature set with network behavior patterns beyond traditional flight and airport attributes.

Model comparison based on recall showed that Logistic Regression currently has the most reliable performance. For predicting actual delays, the model achieved a recall of 61%, successfully flagging the majority of delayed flights.

6.2 Gap Analysis

Our next iteration targets surpassing 80% recall, with a long-term objective of reaching 98% within five iterations. During experimentation, we achieved the 80% recall threshold by adjusting the decision threshold in our logistic regression model; however, this came at a steep cost to precision. In these scenarios, the model largely defaulted to predicting delays, reducing its ability to meaningfully distinguish between delayed and on-time flights. Once we meet the recall target, our subsequent priority will be to improve No Delay precision to 90%.

The primary gap stems from the significant class imbalance and the broad scope of the prediction task. Because the majority of flights arrive on time, models are inherently biased toward predicting No Delay. The next steps outlined below are designed to address these limitations and improve overall predictive balance.

6.3 Recommended Improvements & Next Steps

1. **Revisit Clustering Methods to Segment Flights**
    - Treating all flights identically is too broad; clustering could enable more tailored analysis and robust results.
    - Segment flights by route, seasonality, or operational characteristics to uncover hidden patterns.

2. **Refine Assumptions & Expand Data Sources**
    - Assess the feasibility of predicting delays 1 hour ahead instead of 2.
    - Incorporate advanced, real-time data such as:
        - Staffing and crew schedules
        - Airport congestion levels

3. **Limit Scope by Route or Region**
    - Restrict the model to specific airports or regions to create a more specialized and accurate predictor.
    - This can reduce noise introduced by over-generalization.

4. **Explore Ensemble Modeling**
    - Combine predictions from Logistic Regression, XGBoost, and Feedforward Neural Networks (FFNN) to leverage the complementary strengths of each model.

7 References

1. Author Unknown. (2024, July 12). *U.S. Passenger Carrier Delay Costs.* Airlines for America. https://www.airlines.org/dataset/u-s-passenger-carrier-delay-costs/#:~:text=In%202023%2C%20the%20average%20cost,percent%20to%20%2432.68%20per%20minute (https://www.airlines.org/dataset/u-s-passenger-carrier-delay-costs/#:~:text=In%202023%2C%20the%20average%20cost,percent%20to%20%2432.68%20per%20minute)

2. Author Unknown. (Accessed 2025, August 9). *List of airline mergers and acquisitions* Wikipedia. https://en.wikipedia.org/wiki/List_of_airline_mergers_and_acquisitions (https://en.wikipedia.org/wiki/List_of_airline_mergers_and_acquisitions)

3. Baran, M. (2024, October 17). *These Are the Busiest Travel Days of the Year, According to TSA* AFAR. https://www.afar.com/magazine/tsa-just-had-its-busiest-week-ever (https://www.afar.com/magazine/tsa-just-had-its-busiest-week-ever)

4. Boogaard, K. (2024, September 12). *How to write smart goals (with examples)*. Work Life by Atlassian. https://www.atlassian.com/blog/productivity/how-to-write-smart-goals (https://www.atlassian.com/blog/productivity/how-to-write-smart-goals)

5. Dai, M. (2024, February 26). A hybrid machine learning-based model for predicting flight delay through Aviation Big Data. Nature News. https://www.nature.com/articles/s41598-024-55217-z (https://www.nature.com/articles/s41598-024-55217-z)

6. Karabiber, F. (n.d.). Gini impurity. Learn Data Science - Tutorials, Books, Courses, and More. https://www.learndatasci.com/glossary/gini-impurity/ (https://www.learndatasci.com/glossary/gini-impurity/)

7. Kim, J. (2025, January 3). Graphing the Skies: Timely Flight Delay Prediction using Spatio-Temporal GNNs. Medium. https://medium.com/stanford-cs224w/graphing-the-skies-timely-flight-delay-prediction-using-spatio-temporal-gnns-879854f204ed (https://medium.com/stanford-cs224w/graphing-the-skies-timely-flight-delay-prediction-using-spatio-temporal-gnns-879854f204ed)

8. *Make a great first impression: 6 tips for writing a good abstract*. AJE. (2014, September 14). https://www.aje.com/arc/make-great-first-impression-6-tips-writing-strong-abstract/ (https://www.aje.com/arc/make-great-first-impression-6-tips-writing-strong-abstract/)

9. National Climatic Data Center (NCDC). (2023, June 23). *Quality Controlled Local Climatological Data (QCLCD) Publication*. National Centers for Environmental Information(NCEI)/National Oceanic and Atmospheric Administration(NOAA). https://www.ncei.noaa.gov/access/metadata/landing-page/bin/iso?id=gov.noaa.ncdc%3AC00679 (https://www.ncei.noaa.gov/access/metadata/landing-page/bin/iso?id=gov.noaa.ncdc%3AC00679)

10. Nguyen, M. D. (2023, August 18). Ensembling: Voting and Stacking. Medium. https://medium.com/@minhducnguyen20/ensembling-voting-and-stacking-fd0d37ef9791 (https://medium.com/@minhducnguyen20/ensembling-voting-

and-stacking-fd0d37ef9791)

11. Pedregosa et al. (2011). Scikit-Learn: Ensembles: Gradient boosting, random forests, bagging, voting, stacking. scikit-learn. https://scikit-learn.org/stable/modules/ensemble.html (https://scikit-learn.org/stable/modules/ensemble.html)

12. Pelletier, H. (2024, May 3). *Cyclical Encoding: An Alternative to One-Hot Encoding for Time Series Features*. Towards Data Science. https://towardsdatascience.com/cyclical-encoding-an-alternative-to-one-hot-encoding-for-time-series-features-4db46248ebba/ (https://towardsdatascience.com/cyclical-encoding-an-alternative-to-one-hot-encoding-for-time-series-features-4db46248ebba/)

13. U.S. Departement of Transportation. (n.d.). Bureau of Transportation Statistics. https://www.transtats.bts.gov/Fields.asp?gnoyr_VQ=FGJ (https://www.transtats.bts.gov/Fields.asp?gnoyr_VQ=FGJ)

14. Yamini. (2021, April 23). *Random Forest-Ensemble Method*. Medium. https://medium.com/geekculture/random-forest-ensemble-method-860aaf4fcd16 (https://medium.com/geekculture/random-forest-ensemble-method-860aaf4fcd16)

8 Appendix

8.1 Pipeline Improvements and Preliminary Results

Based on the feedback provided during our presentation we wanted to improve on our models. Below outlines additional work we did and the preliminary results.

Data Processing

We updated our pipeline to rescale (using z-score scaling) some features that had previously been missed:

- `avg_flights_last_7_days`
- `avg_delays_last_7_days`

- `avg_arrivals_last_7_days`

Previously, because we could not measure the importance of one-hot-encoded features, we did not include categorical features in our pipeline. We updated our encoding method for the following features by calculating and then scaling their log frequency:

- `origin_airport_code`
- `dest_airport_code`
- `origin_timezone`
- `dest_timezone`
- `OP_UNIQUE_CARRIER`

In order for more the model to interpret more recent data as more relevant, we encoded `YEAR` by calculating the max year in the training set, assigning it a value of 1, and then decaying at a rate of 0.5 for each previous year. We attempted to z-score scale this encoding, but this created many nulls, so we ended up leaving the original encoding intact.

We also re-wrote our scaling pipeline wihtout interim VectorAssembler steps, in order to more easily decode feature importance in future model runs.

Unfortunately, due to time constraints, we were unable to include our graph vector feature in this updated data processing pipeline. Therefore, in the results below, Old Pipeline refers to the old data pipeline WITH the graph vector feature, while New Pipeline refers to the new data pipeline WITHOUT the graph vector feature.

Logistic Regression Results

Class	Metric	Old Pipeline	New Pipeline	% Change
0	Recall	0.6117	0.5830	-4.69%
0	Precision	0.8764	0.6161	-29.67%
0	F1	0.7205	0.5991	-16.86%
1	Recall	0.6125	0.6356	+3.77%
1	Precision	0.2600	0.6031	+131.19%

Class	Metric	Old Pipeline	New Pipeline	% Change
1	F1	0.3650	0.6189	+69.57%

Even without the graph feature, the updated pipeline shows substantial improvement in precision and F1 for the Delay class, along with a slight gain in recall. This indicates the new pipeline is much better at correctly identifying delayed flights without a large trade-off in missed delays. However, performance for the No Delay class drops moderately across all metrics, suggesting the model is now more prone to predicting delays where there are none. This shift reflects a rebalancing of the model toward capturing more true delays at the expense of some false positives for no-delay flights.

Feedforward Neural Network Results

Class	Metric	Old Pipeline	New Pipeline	% Change
0	Recall	0.1655	0.5586	+237.46%
0	Precision	0.8336	0.9064	+8.73%
0	F1	0.2762	0.6912	+150.23%
1	Recall	0.8517	0.6646	-22.00%
1	Precision	0.1852	0.2056	+11.01%
1	F1	0.3042	0.3140	+3.22%

The new FNN pipeline delivers dramatic gains for the No Delay class, with recall surging by over +237% and F1 more than doubling (+150%). Precision also improves slightly (+8.73%), indicating stronger overall balance when predicting no-delay flights. For the Delay class, recall drops noticeably (–22.00%), suggesting more missed delays, but both precision (+11.01%) and F1 (+3.22%) see modest gains. Overall, the new FNN pipeline trades some recall on delayed flights for much stronger performance on no-delay predictions, resulting in a far more balanced model.

8.2 Ensemble Model Experiments:

To evaluate the impact of different ensemble strategies on predictive performance, we compared Stacked Ensemble and Soft Voting Ensemble against their individual base learners: Feed-Forward Neural Network (FFNN), Logistic Regression, and XGBoost. Performance metrics were calculated for both the delay and no delay classes using the F1 Score, Precision, and Recall.

Model: Stacked

Key observations:

For the delay class, Logistic Regression achieved the highest F1 (0.720) and recall (0.612), while the Stacked Ensemble matched XGBoost in precision (0.895) but did not surpass the best base model in F1.

For the no delay class, the Stacked Ensemble matched XGBoost exactly, suggesting the meta-learner heavily weighted the XGBoost predictions.

Why the Stacked Ensemble matched XGBoost exactly:

This occurs when one base model (In this case: XGBoost) consistently outperforms the others on the training folds. The meta-learner (Logistic Regression) learns to place nearly all its weight on XGBoost's predictions, effectively copying them. Without measures to encourage diversity (e.g., regularizing the meta-learner, calibrating probabilities, or ensuring more varied base models), stacking can collapse into simply replicating the dominant model.

Model: Soft_Voting

Key observations:

The Soft Voting Ensemble achieved the highest F1 score for the delay class (0.366) by balancing the recall advantage of FFNN with the precision of Logistic Regression and XGBoost.

For the no delay class, Soft Voting achieved precision (0.881) close to XGBoost (0.886) while improving recall over both Logistic Regression and XGBoost.

Soft Voting provided more balanced performance across both classes compared to any single base learner.

Conclusion:

While the Stacked Ensemble did not consistently outperform the best base learner, the Soft Voting Ensemble offered modest gains in F1 score for the delay class and improved recall for the no-delay class, suggesting it better leveraged complementary strengths of the base models without overfitting to a single dominant model.

8.3 Notebooks

Phase 1

- Combined EDA Notebook (https://dbc-fae72cab-cf59.cloud.databricks.com/editor/notebooks/461183347491767?o=4021782157704243#command/5845660857438691)
  - Clara EDA (https://dbc-fae72cab-cf59.cloud.databricks.com/editor/notebooks/1473666391937765?o=4021782157704243)
  - Ainsley EDA (https://dbc-fae72cab-cf59.cloud.databricks.com/editor/notebooks/2307011031255678?o=4021782157704243#command/8364543520232348)
  - Katya EDA (https://dbc-fae72cab-cf59.cloud.databricks.com/editor/notebooks/1422956527184024?o=4021782157704243#command/5263727993576205)
  - Monica EDA (https://dbc-fae72cab-cf59.cloud.databricks.com/editor/notebooks/1473666391937783?o=4021782157704243)

    - Safiya EDA (https://dbc-fae72cab-cf59.cloud.databricks.com/editor/notebooks/1007317598244412?o=4021782157704243#command/5263727993576494)
- Raw Data Dimensions & Sizes (https://dbc-fae72cab-cf59.cloud.databricks.com/editor/notebooks/819275088972559?o=4021782157704243#command/8238339641868913)
- Logistic Regression - Phase 1 (https://dbc-fae72cab-cf59.cloud.databricks.com/editor/notebooks/3631386026268697?o=4021782157704243#command/8794488278568243)

Phase 2

- Phase 2 Pipeline Notebook - Draft 1 (https://dbc-fae72cab-cf59.cloud.databricks.com/editor/notebooks/921730179317256?o=4021782157704243#command/7278974534762671)
- Phase 2 Pipeline Notebook - Draft 2 (https://dbc-fae72cab-cf59.cloud.databricks.com/editor/notebooks/819275088972739?o=4021782157704243#command/8238339641869844)
- Join Notebook (https://dbc-fae72cab-cf59.cloud.databricks.com/editor/notebooks/819275088972070?o=4021782157704243#command/8238339641868402)
- Logistic Regression - Phase 2 (https://dbc-fae72cab-cf59.cloud.databricks.com/editor/notebooks/819275088972570?o=4021782157704243)
- XGBoost Model Notebook (https://dbc-fae72cab-cf59.cloud.databricks.com/editor/notebooks/851535786085675?o=4021782157704243)
- FFNN Notebook (https://dbc-fae72cab-cf59.cloud.databricks.com/editor/notebooks/3631386026269202?o=4021782157704243#command/8794488278569540)

- DBSCAN + COWRF Notebook (https://dbc-fae72cab-cf59.cloud.databricks.com/editor/notebooks/2135364053139113?o=4021782157704243#command/8293166983681721)
- Random Forest Classifer (https://dbc-fae72cab-cf59.cloud.databricks.com/editor/notebooks/851535786086602?o=4021782157704243#command/851535786086636)

Phase 3

- Phase 3 - Determine Important Features (https://dbc-fae72cab-cf59.cloud.databricks.com/editor/notebooks/3831723723990428?o=4021782157704243)
- Phase 3 Data Processing Pipeline (https://dbc-fae72cab-cf59.cloud.databricks.com/editor/notebooks/3831723723988220?o=4021782157704243)
- We reused phase 2 notebooks for modeling

Phase 4

(After Phase 3 Presentation)

New Data Processing Pipeline (https://dbc-fae72cab-cf59.cloud.databricks.com/editor/notebooks/3600546862783548?o=4021782157704243)

Determine Graph Feature Importance for LR via Its Absence (https://dbc-fae72cab-cf59.cloud.databricks.com/editor/notebooks/2431760182593865?o=4021782157704243)

LR Regularization Experiment (https://dbc-fae72cab-cf59.cloud.databricks.com/editor/notebooks/3600546862782593?o=4021782157704243)

Other LR Experiments (https://dbc-fae72cab-cf59.cloud.databricks.com/editor/notebooks/2431760182593635?o=4021782157704243)

Failed attempt to integrate GNN into new data pipeline (https://dbc-fae72cab-cf59.cloud.databricks.com/editor/notebooks/2431760182592726?o=4021782157704243)

Determine important features for LR without graph vector (https://dbc-fae72cab-cf59.cloud.databricks.com/editor/notebooks/2431760182592563?o=4021782157704243)

Soft_Voting_Ensemble (https://dbc-fae72cab-cf59.cloud.databricks.com/editor/notebooks/2431760182594801?o=4021782157704243#command/6326987468980879)

Stacked_Ensemble (https://dbc-fae72cab-cf59.cloud.databricks.com/editor/notebooks/2431760182593664?o=4021782157704243#command/6326987468974116)

8.4 EXTRA CREDIT

8.4.1 MLFlow Logging Troubleshooting and Workaround

Throughout our project, we encountered persistent and confusing issues with MLflow experiment tracking. Specifically, attempts to log experiments from our Databricks notebooks repeatedly failed with the following errors:

- `RESOURCE_DOES_NOT_EXIST`
- `UNKNOWN_EXPERIMENT_ID`

Diagnosis

After extensive debugging, we discovered that the issue stemmed from changes introduced in MLflow version 3.0, released in June 2025. This version appeared to alter the default behavior of autologging and experiment registration in Databricks,

breaking the automatic linkage between notebook sessions and experiment directories.

To identify the scope of the problem, we used the following diagnostic code to enumerate accessible experiments:

```
for exp in mlflow.search_experiments():
    print(exp.experiment_id, exp.name, exp.artifact_location)
```

We found that our cluster only had permissions to log to experiments owned by a small number of users (e.g., `/Users/ucberkeley2023@berkeley.edu/`, `/Users/jasonscott@berkeley.edu/`, and `/Users/ketadesai@berkeley.edu/`). Any attempt to create or log to an experiment under a different path—such as one owned by our own users (e.g., `/Users/katya_aukamp@berkeley.edu/`)—resulted in resource errors.

Root Cause

Our investigation suggests that MLflow 3.0 no longer reliably autoconfigures notebook-based experiment paths unless:

1. The notebook's workspace user matches the owner of the experiment path, and
2. The cluster has explicit write permissions to the experiment's artifact location.

This appears to be a regression from previous versions of MLflow, where notebook contexts were automatically tied to a valid default experiment.

Workaround

After trying multiple failed workarounds—including creating experiments manually via the Experiments UI, setting experiment IDs explicitly, and using personal access tokens—we ultimately found a temporary solution:

We set our experiment paths to one of the known working directories (e.g., `/Users/ucberkeley2023@berkeley.edu/`) where our cluster had confirmed access. While this allowed us to log new runs, it required impersonating a user identity we didn't own and had no visibility into the setup of. We could not log to our own experiment directories, and attempts to write to other directories consistently failed.

Outcome

This workaround enabled us to continue tracking experiments, but only after hours of unnecessary effort. The fact that we could see and rerun others' experiments—but not create our own—suggests a misalignment between workspace user permissions, notebook contexts, and experiment artifact locations under MLflow 3.0.

OTHER SCREENSHOTS HERE (https://github.com/cybrckr/create_image_links/tree/main)

8.4.2 Dataset Join

We joined OTP flight data with ISD weather data ourselves (alongside many smaller-scale joins). Please see section `2.2 Dataset Join` for more details.

8.4.3 Training a Second Deep Learning Model

In addition to the MLP requested by the client, we also trained a second neural network to produce our graph feature. This deep learning technique produced an information-rich vector of relationships between airport-specific features and flight-specific features which we then fed to our other models. For more information, please see section `3.4 Feature Engineering: Graph Neural Net`.

8.5 Phase Leader Plan:

Week	Phase	Leader
Mon Jul 7 - Sun Jul 13	I	Monica
Mon Jul 14 - Sun Jul 20	II	Ainsley
Mon Jul 21 - Sun Jul 27	II	Safiya
Mon Jul 28 - Sun Aug 3	III	Clara
Mon Aug 4 - Sat Aug 9	**III**	**Katya**

8.6 Credit Assignment Plan

Katya Aukamp / katya_aukamp@berkeley.edu

TOTAL TIME PHASE 3: 32 HRS

- Build Graph Neural Net
  - Budgeted time : 5 hours
  - Actual Time: 8 hours
  - Dates: 8/2 - 8/4

  Description: Using a resource online build the graph neural net using torch geometric package

- Updated pipeine to run on full data
  - Budgeted time : 3 hours
  - Actual Time: 10 hours
  - Dates: 8/3 - 8/4

  Description: Combined notebooks so that the joins, the full data run, graph neural net and mapping all ran from beginning to end

- Built Skeleton of final presentation
  - Budgeted time : 2 hours
  - Actual Time: 5 hours
  - Dates: 8/5

  Description: Put together the foundations of the final presentation, speaker notes, transitions, making sure the story was there

- Helped debug MlFlow
  - Budgeted time : 30 min
  - Actual Time: 2 hours
  - Dates: 8/1

Description: Worked with teammates to figure out the issue with MLFlow by checking what workplaces we had access too, and debugged why the experiments wouldn't log correctly

- Formatted and Wrote the project story for final phase
  - Budgeted time : 3
  - Actual Time: 7
  - Dates: 8/7 - 8/9

  Description: Updated the notebook to read as the final project story, with explanation focused toward business stakeholders

Ainsley / wbock@berkeley.edu

TOTAL TIME PHASE 3: 52 HRS

- Rebuilt the data processing pipeline twice
  - Budgeted time: 0 HRS
  - Actual Time: 12 HRS

  Description: Once for independent scaling of CV folds, once to support integration of some extra data set or last minute change we made.

- Implemented independent scaling of cross-validation folds to prevent data leakage.
  - Budgeted time: 2
  - Actual Time: 4

  Description: Change required by instructor.

- Debugged ML Flow and configured a working instance despite Databricks misconfiguration.
  - Budgeted time: 0 HRS
  - Actual Time: 12 HRS

Description: Spent a day on this at beginning of phase 3 because I couldn't get the training-loss history out of MLLib - and both instructors, a TA, and 3 LLMs said MLib didn't support loss history. No luck the first time because databricks was misconfigured. Katya was working the issue later in phase 3, I was discussing it with her, and began exploring it again. I found I could write to an old student directory from a prior semester and create/link to MLFlow experiments there. More information in section 8.3 above. LOTS of spinning my wheels on this one. Also Safiya found the exact MLLib API call I was looking for, burried in the documentation - so she's why we were able to make nice training loss vs epoch plots. Girl came in clutch.

- Update FFNN training pipeline to perform binary classification instead of multiclass.
  - Budgeted Time: 1 Hr
  - Actual Time: 3 Hr

  Description:

- Built and tuned four different FFNN architectures for experimentation and performance comparison.
  - Budgeted time: 8 HRS
  - Actual Time: 12 HRS

  Description: Manual hyperparameter tuning of 1 and 2 layer FFNN architectures on various feature selections. See section 4.4.3 for more info.

- Analyzed and interpreted model performance results to guide conclusions.
  - Budgeted time: 2 HRS
  - Actual Time: 2 HRS

  Description: Statistical and real world interpretations of precision, recall, and f1 scores on a per class basis.

- Presentation Slide Updates and Speaker Notes
  - Budgeted: 1
  - Actual: 6

    - Description: Slides 14, 15, 22, 24-29. Wrote discussion of cross-validation. Collaborated with others extensively, b/c I cannot make a slide layout to save my life.

- Report contributions: authored neural network section and provided review feedback.
    - Budgeted time : 1 HRS
    - Actual Time: 1 HRS

Monica / monicaj_martin@berkeley.edu

TOTAL TIME PHASE 3: 20.5 HRS

- Credit Assignment Plan (Previous Phase 3 Notebook)
    - Budgeted time : 2
    - Actual Time: 1.5

  Description:
    - Reformatted project tracking table to include "Start" and "End" date columns and "Budgeted" hours. Ultimately **scrapped** after determining the format did not add analytical value for Phase 3 reporting.

- Ensemble Model Section (Report)
    - Budgeted time : 1.5 HRS
    - Actual Time: 1.5 HRS

  Description:
    - Drafted and refined the Ensemble Model section of the final report, including an overview of methods, rationale for model selection, and a summary of comparative performance results.

- Data Leakage Dicussion (Report)
    - Budgeted time : 2 HRS
    - Actual Time: 2 HRS

  Description:

    - Defined data leakage in the context of predictive modeling, outlined its potential impacts on model validity, and detailed the cross-validation and preprocessing strategies implemented in the notebook to mitigate leakage.

- Stacked Ensemble Model (Notebook)
    - Budgeted time : 8 HRS
    - Actual Time: 10 HRS

  Description:
    - Implemented and evaluated a stacked ensemble in Spark ML, integrating FFNN, Logistic Regression, and XGBoost as base learners. Developed a meta-learner using Logistic Regression, performed hyperparameter tuning, and assessed per-class performance.

- Soft_Voting Ensemble Model (Notebook)
    - Budgeted time : 4 HRS
    - Actual Time: 5 HRS

  Description:
    - Developed a Spark-based soft voting ensemble to combine base model probabilities without stacking. Implemented equal-weight and weighted approaches, tuned weights via simplex grid search, and optimized thresholds for F1 and recall.

- Reference (Report)
    - Budgeted time : .5 HRS
    - Actual Time: .5 HRS

  Description:
    - Added additional reference and updated entries to alphabetical order in accordance with APA style guidelines.

Clara / clara.rhoades@berkeley.edu

TOTAL TIME PHASE 3: 61 HRS

- Phase 3 Processing Pipeline
  - Budgeted time : 7
  - Actual Time: 14

  Description:
  - established data processing notebook organizational structure to ensure steps happen in proper order
  - established checkpointing strategy
  - rewrote data pipeline to be able to reverse engineer VectorAssembler to determine important features

- Supplemental dataset research to support Katya's join
  - Budgeted time : 0
  - Actual Time: 5

  Description:
  - FAA Registry to try to get aircraft age based on tail number
  - Airline reputation dataset research/EDA
  - Aggregation of rankings data for Katya's join
  - responsible for finding FEMA dataset
  - wrote code to get sizes of all datasets

- Built Baseline Models
  - Budgeted time : 3 hrs
  - Actual Time: 5 hrs

  Description:
  - Built most frequent class and logistic regression models

- Important Feature Selection
  - Budgeted time : 3 hrs
  - Actual Time: 10 hrs

  Description:
  - Initiated final feature selection via regularization and custom grid search
  - reverse-engineered vector assembler

- MLFlow

    - Budgeted time : 1 hrs
    - Actual Time: 8 hrs

  Description:
    - research, troubleshooting, and documentation
    - wrote MLFlow part of appendix
    - surfaced issue with instructor multiple times

- Logistic Regression Experimentation
    - Budgeted time : 5 hrs
    - Actual Time: 1 hr

  Description:
    - regularization
    - threshold

- Phase 4 Data Cleaning
    - Budgeted time : 0 hrs
    - Actual Time: 14 hrs

  Description:
    - performed all steps outlined in section 8.1
    - built new versions of all datasets (w/o GNN)

- Phase 3 Presentation
    - Budgeted time : 2 hrs
    - Actual Time: 2 hrs

  Description:
    - wrote dataset, abstract, outline, feature selection slides

- Phase 3 Report
    - Budgeted time : 5
    - Actual Time: 12

  Description:
    - rewrote abstract for binary classification
    - updated team picture, added logo

- wrote Logistic Regression algorithm section
- wrote feature importance section for LR in 5.4
- wrote section 8.1 on phase 4
- renamed all sections and added correct headings
- reviewed content and rewrote as needed for grammar, clarity, and accuracy
- compared with final rubric to ensure all requirements are met

Safiya / safiyaalavi@berkeley.edu

TOTAL TIME PHASE 3: 20 HRS

- Coded the XGBoost model
  - Budgeted time : 5 HRS
  - Actual Time: 6 HRS
  - 7/29 - 8/7

  Description: Experimentation of the XGBoost hyperparameters, finalizing model, and testing on testing set for both Phase 3.

- Phase 3 Notebook Assignments
  - Budgeted time : 4 HRS
  - Actual Time: 4 HRS
  - 8/1 - 8/8

  Description: Ensured shell of Phase 3 addresses rubric. Wrote the XGBoost sections of the report and the Results portion. Edited the EDA section to be combined with the data dictionary.

- Assisting Others
  - Budgeted time : 2 HRS
  - Actual Time: 2 HRS
  - 8/5

  Description: Assisted with making epoch/loss figure for neural net.

- Presentation slides for Phase 3

- Budgeted time : 3 HRS
- Actual Time: 3 HRS
- 8/1 - 8/6

Description: Created slides related to XGBoost, implemented Phase 2 feedback into Phase 3 presentation.

- Group Discussions
  - Budgeted time : 5 HRS
  - Actual Time: 5 HRS
  - 7/29 - 8/7

Description: Numerous meetings throughout Phase 3 of this project to produce final result (models, presentation and report).