Flight Predict - Predicting Departure Delays using ML

Phase leader plan in Table format

Phase	Leader	Duration	Project Plan
1	Kowsalya Ganesan	Oct 27- Nov 2	Abstract, Data description, Machine algorithms and metrics, Machine Learning Pipelines
2	Eric Wu, Steven Au	Nov 3 - Nov 19	Abstract, Data description, Feature Engineering, EDA, Lasso, Model implementation, Hyper parameter tuning, Next steps
3	Yu-Sheng, Naresh	Nov 20 - Dec 13	Abstract, Data description, Feature Engineering, EDA, MLP, Tree Models, Model implementation, Hyper parameter tuning, Top Features, Conclusion

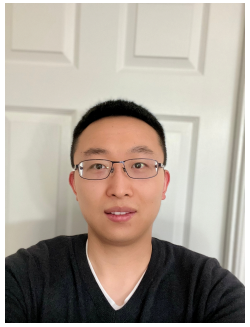
Credit assignment plan / who does/did what

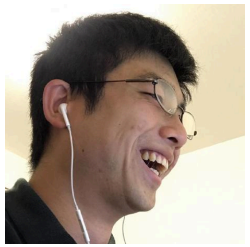
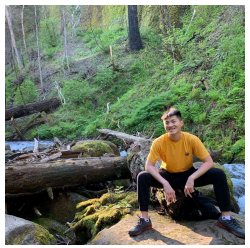
Team Member (A–Z by first name)	Responsibilities	Time Spent
Eric Wu	Logistic Regression, MLP, Feature Engineering, Hyper-parameter search, Reports	32 hours

;

Team Member (A–Z by first name)	Responsibilities	Time Spent
Kowsalya Ganesan	Data Analysis, Data Quality Check, Data Imputation Strategy, Data Quality Fixes, Including External Data, Feature Engineering , Feature Selection	42 hours
Naresh Kumar	Problem definition, Abstract preparation, Feature analysis, Lasso regularization, Pipeline Analysis, Pagerank, Data Quality Fixes, Conclusion	36 hours
Steven Au	XGBoost and Random Forest models, Feature Engineering for models, Cross Validation, Hyperparameter tuning, Model Performance Report on XG/RF	34 hours
Yu-Sheng Lee	Weather Features - HourlySkyConditions, Graph-Based Features - Airport PageRank Score Weighted By Traffic Volume, Time Features - Sine-Cosine Encoded Departure Time, Data Leakage	36 hours

Team Members

Name (A–Z by first name)	Email	photo
Eric Wu	enqiwu@berkeley.edu (mailto:enqiwu@berkeley.edu)	

Name (A–Z by first name)	Email	photo
Kowsalya Ganesan	kowsalya@berkeley.edu (mailto:kowsalya@berkeley.edu)	
Naresh Kumar	naresh_kumar@berkeley.edu (mailto:naresh_kumar@berkeley.edu)	
Steven Au	steau@berkeley.edu (mailto:steau@berkeley.edu)	
Yu-Sheng Lee	yushenglee@berkeley.edu (mailto:yushenglee@berkeley.edu)	

Project Abstract

Airports face significant operational challenges due to unpredictable flight delays (departure delays over 15 minutes than scheduled time), which cost the U.S. economy billions annually by disrupting crew schedules, gate allocations, and passenger itineraries. This project develops a data-driven predictive framework to classify flights as delayed or on-time prior to departure, enabling proactive operational planning and improved passenger experience. Using the U.S. Department of Transportation On-Time Performance dataset (2015–2019), integrated with NOAA weather data and airport metadata, we analyzed over 31 million flights and engineered more than 50 features capturing temporal patterns, congestion, weather, seasonality, operational disruptions, airline reputation, and network-based PageRank metrics.

Exploratory analysis revealed that delays were rare but highly variable,highlighting both class imbalance and the importance of network- and weather-informed predictors. **In Phase 1**, exploratory data analysis showed strong correlations between delay duration and factors such as poor weather visibility, precipitation, congestion at major hubs, and peak-hour departures. We framed delay prediction as a binary classification task and selected Logistic Regression as the baseline model, with Random Forest and XGBoost identified as advanced modeling approaches. We also identified potential improvements through temporal features, route-based clustering, and real-time streaming predictions for airport operations dashboards. Model evaluation focused on Precision, Recall, and F1-score.

In Phase 2, we performed detailed analysis on **one year of data (2015)**, consisting of 5.8 million flight records, to predict whether a departing flight would be delayed (DEP_DELAY_GROUP ≥ 1) or on-time (≤ 0). We established baseline models using a robust, multi-stage feature engineering pipeline that included data cleaning, categorical encoding, feature assembly, PCA, Lasso regularization, standardized scaling, and model training. **Logistic Regression achieved a baseline performance of 27% precision, 34% recall for the delayed class**, reflecting the impact of class imbalance. **Ensemble models significantly improved performance, with Random Forest achieving 78% precision, 70% recall, and XGBoost achieving 78% precision, 74% recall**. These results were obtained using three

quarters of data for training and one quarter for testing. The primary challenge identified at this stage was scaling the pipeline to five years of data while implementing multi-fold, time-aware cross-validation and holding 2019 as a final test year.

In Phase 3, we expanded the analysis to **five years of flight and weather data** and incorporated airline reputation scores and airline PageRank as network-aware features, capturing both operational reliability and exposure to structurally critical routes. We engineered features across multiple domains, including time-based patterns, seasonality, graph-based network structure, airline reputation, major events, natural disasters, airport maintenance disruptions, and weather conditions. After extensive preprocessing, we trained models including XGBoost, Random Forest, and Multilayer Perceptron (MLP), using rolling cross-validation across training windows from 2015–2018 and holding 2019 as a blind test set.

Our modeling pipeline consisted of raw data cleaning, label construction, feature engineering, time-respecting train/test splitting, class imbalance handling, feature transformation, model training, hyperparameter tuning, and final evaluation. **The final optimized XGBoost model** achieved a **66% recall and 77% precision for delayed flights**, with scheduled departure time, recent route performance, and airline network centrality emerging as the most influential predictors. These results demonstrate the value of combining traditional operational features with engineered reputation and network-aware metrics. The proposed framework provides actionable insights for airports and airlines, supporting better resource allocation, congestion management, and targeted interventions to improve on-time performance.

Data and Feature Engineering

Data Lineage & Key Transformations

Data Lineage Summary: The modeling dataset is built by progressively enriching raw flight performance records with several complementary external data sources to capture the full range of temporal, environmental, and operational factors influencing flight delays. The **On-Time Performance (OTPW) dataset** from the **U.S. Department of Transportation** serves as the central fact table at the flight level. This core dataset is augmented through a series of joins with hourly weather observations, calendar and holiday data, major event schedules, natural disaster records, and airport maintenance and disruption logs. Together, these integrations provide a unified, flight-level view that reflects both routine operating conditions and irregular system shocks.

Data Description

Dataset	Source	Description
On-Time Performance Flight Data (OTPW)	U.S. DOT Bureau of Transportation Statistics (/(https://www.transtats.bts.gov/Fields.asp?gnoyr_VQ=FGJ)	Detailed flight-level operational data (times, delays, identifiers)
Hourly Weather Data	NOAA / Databricks-provided weather tables (https://www.ncei.noaa.gov/access/metadata/landing-page/bin/iso?id=gov.noaa.ncdc:C00679)	Temperature, wind, visibility, precipitation, humidity, and other hourly metrics
Airport Reference Data	Airport Metadata (https://datahub.io/core/airport-codes)	Latitude, longitude, elevation, airport type, region

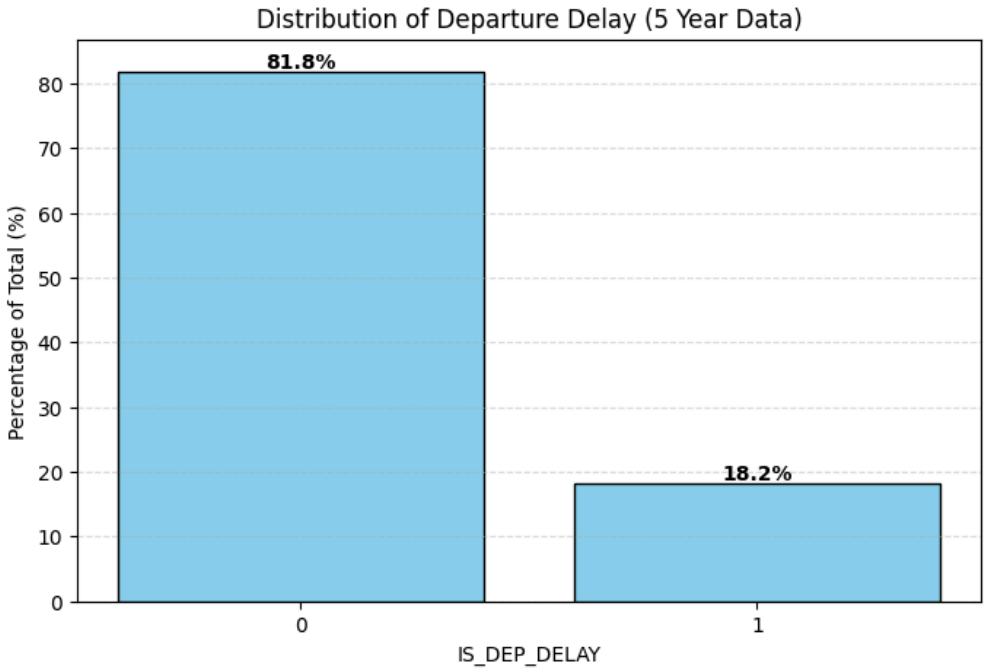
Key Transformations & Joins: No custom or external joins were introduced beyond the datasets provided in the platform. The modeling dataset was constructed using the **On-Time Performance (OTPW) flight data, hourly weather data, and airport reference** data supplied as part of the project resources. These datasets were integrated

using their standard keys (flight date, airport identifiers, and time alignment), ensuring consistent and reproducible joins. All subsequent processing focused on feature engineering and transformation rather than additional data integration.

Distribution of Target Variable

The below histogram shows the distribution of our target variable, `IS_DEP_DELAY` derived from `DEP_DELAY_GROUP`. This is a binary variable indicating if a flight departs over 15 minutes than scheduled time. The class distribution is imbalanced where the non-delay class is 18.2%. We will factor in class weight distribution into our modeling phases to address this problem.

`IS_DEP_DELAY ∈ {delayed, not_delayed}` where
`delayed = DEP_DELAY_GROUP >= 1`
`not_delayed = DEP_DELAY_GROUP <= 0`



Feature Families

Feature Family	Explanation
Time-Based Features	Capture intraday delay patterns and short-term operational pressure using cyclic time-of-day encodings and recent rolling delay trends at the route, origin, and destination levels.
Seasonality Features	Model predictable calendar-driven demand cycles using month, weekday/weekend indicators, seasonal categories, and summer peak flags.
Graph-Based Network Features	Represent the structural importance of airports within the national flight network using PageRank, capturing delay propagation and hub-related effects.

Feature Family	Explanation
Airline Reputation Features	Summarize carrier-level operational reliability over time by combining on-time performance, cancellation rates, and diversion rates into a monthly reputation score.
Event-Based Features	Capture non-routine demand surges from major scheduled events such as federal holidays, conferences, and sporting events using date-window and airport-specific indicators.
Natural Disaster Features	Model rare, high-impact disruptions (e.g., hurricanes, floods, wildfires) using binary indicators, severity scores, and temporal decay features.
Airport Maintenance & Operational Disruption Features	Capture sustained capacity constraints and infrastructure-related disruptions through severity encoding and duration-based features.
Weather Features	Represent atmospheric impacts on operations using parsed categorical sky conditions and a PCA-based Weather_Index summarizing continuous weather variables.

Time-Based Features

1. Sine-Cosine Transformed Time-Of-Day

Transforms scheduled departure time into smooth cyclic variables that capture the 24-hour periodicity of flight operations. These features help both tree-based models and neural networks learn temporal patterns without the artificial discontinuity at midnight. Our EDA shows that the delay rate follows a cyclical pattern. The delay rate fluctuates significantly during the early morning hours. Then the delay propagates through the network, and the rate increases until it peaks around 8:30-9:00 PM. After that, the delay rate declines gradually. The pattern repeats and time-of-day is circular, but the linear numeric features (e.g., hour = 23 vs hour = 0) incorrectly suggest that 23 and 0 are far apart, when in reality they represent adjacent moments. To address this, we encode time-of-day using the sine and cosine transformation, scaled by the total number of minutes in a day (1440).

- `CRS_DEP_MINUTES`: The scheduled departure time expressed as the number of minutes since midnight, computed as:

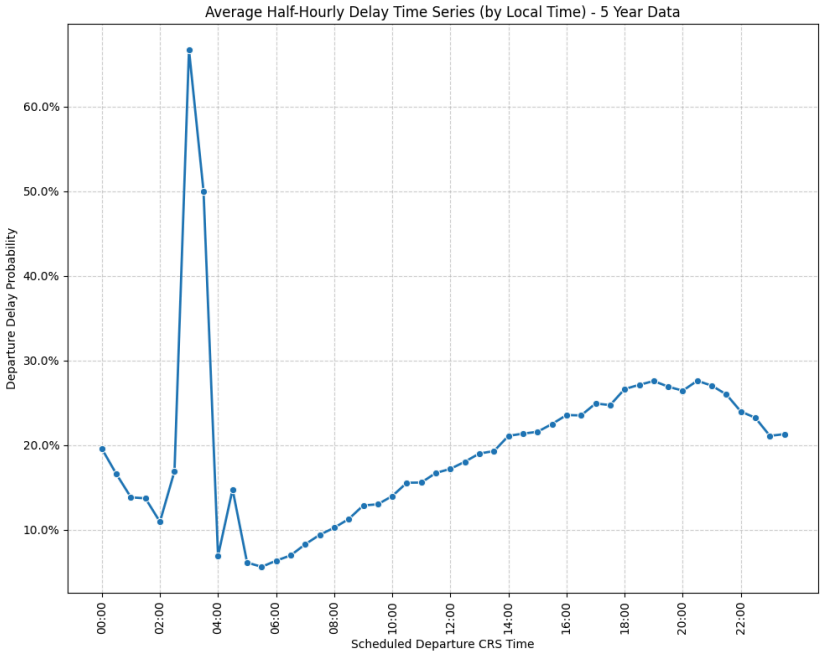
$$CRS\_DEP\_MINUTES = Hour\ of\ CRS\_DEP\_TIME \cdot 60 + Minute\ of\ CRS\_DEP\_TIME$$

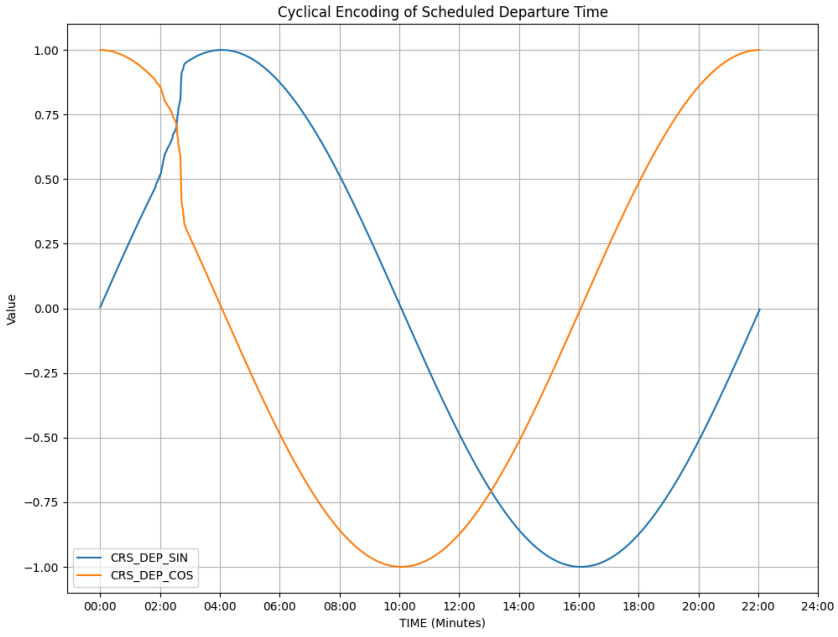
- `CRS_DEP_SIN`: The sine transformation of CRS_DEP_MINUTES, scaled by the total number of minutes in a day (1440):

$$CRS\_DEP\_SIN = sin(2\pi \cdot \frac{CRS\_DEP\_MINUTES}{1440})$$

- `CRS_DEP_COS`: The cosine transformation of CRS_DEP_MINUTES, scaled by the total number of minutes in a day (1440):

$$CRS\_DEP\_COS = cos(2\pi \cdot \frac{CRS\_DEP\_MINUTES}{1440})$$





Since every record includes the `CRS_DEP_TIME` field, generating these features simply required deriving `CRS_DEP_MINUTES` and applying the formulas above. The shape of `CRS_DEP_SIN` appears is similar to the daily delay rate shape (if it is multiplied by negative one to flip the curve). In short, sine–cosine transformations provide the model with a mathematically appropriate representation of time, resulting in more stable and accurate learning of delay trends.

2. Recent Delay Trends (7-day rolling)

To make our model smart about real-world operational pressure, we needed features that captured the immediate history of a flight route or airport, not just long-term averages. This led us to create Rolling Average Delay Features, which serve as a critical proxy for current system stress and delay propagation.

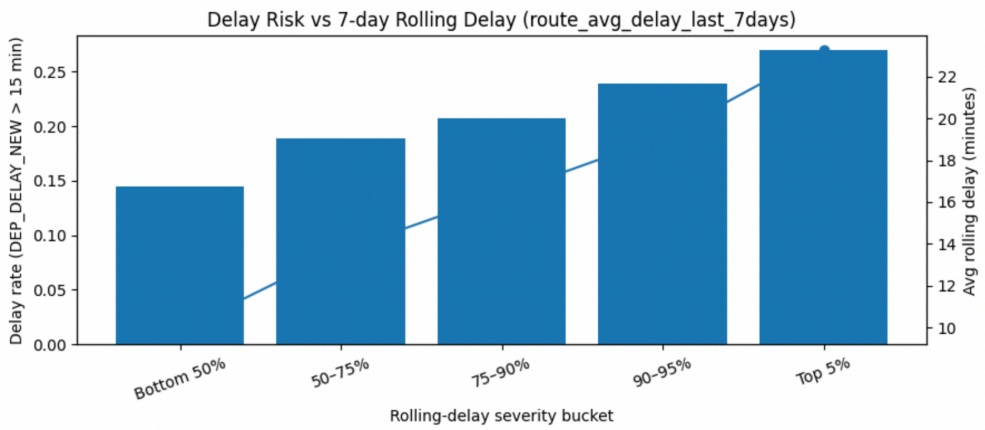
These features calculate the average departure delay (in minutes) for the past seven days, based on three different operational perspectives:

Route Average (*route_avg_delay_last_7days*): This measures how delayed a specific pairing (e.g., LAX to JFK) has been recently. If this route has been hammered all week, the score will be high, indicating high current risk.

Origin Average (*origin_avg_delay_last_7days*): This measures the overall operational stress at the departure airport (e.g., LAX) over the last seven days, regardless of the flight's destination. It tells us if the origin airport is currently struggling with congestion or ground crew issues.

Destination Average (*dest_avg_delay_last_7days*): This captures how congested or delayed the destination airport (e.g., JFK) has been. A high score suggests delays are likely to ripple through that airport's schedule, potentially affecting the inbound flight and its eventual turnaround time.

Below figure shows delay risk as a function of recent operational stress measured by the 7-day rolling route delay. Flights in higher rolling-delay quantiles exhibit a increase in delay probability, with delay rates rising from approximately 14% in the bottom 50% to over 25% in the top 5%. This confirms that rolling delay features effectively capture short-term congestion and delay propagation, providing strong predictive signal beyond static route or seasonal effects.



Seasonality Features

Seasonality features were engineered to capture predictable and recurring patterns in air travel demand and delay behavior across both the calendar year and the week. Using the flight date, we derived month- and weekday-based indicators to model annual travel cycles, weekly commuting patterns, and congestion effects that systematically influence flight performance.

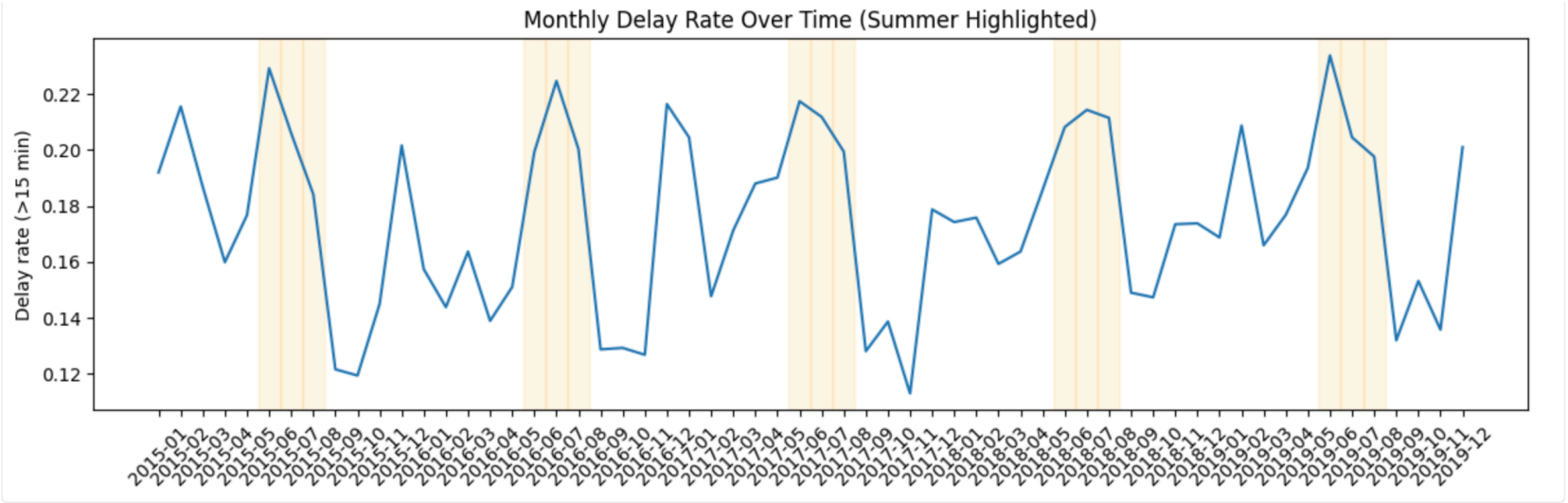
is_summer_peak: A binary indicator identifying peak summer travel months (June–August). This feature captures elevated leisure demand, increased network congestion, and tighter operational margins during the busiest travel period of the year.

season: A categorical feature assigning each flight to Winter, Spring, Summer, or Fall, modeling broad seasonal differences in travel demand and operational conditions.

is_weekday: A binary indicator distinguishing weekday travel (Monday–Friday) from weekend travel, separating business-driven traffic patterns from leisure-oriented demand.

Together, these features enable the model to learn stable and repeatable temporal effects that drive systematic variation in flight delays.

The chart below illustrates clear seasonal differences in delay rates. Summer exhibits the highest delay probability, reflecting peak travel demand and network congestion, while winter also shows elevated delays driven by adverse weather conditions. In contrast, spring and fall experience lower and more stable delay rates, with fall being the least disruptive season. These findings validate the inclusion of both granular seasonal indicators and a dedicated summer-peak feature in the model.



Graph-Based Features

Raw airport codes are nominal categorical features, which makes it difficult for a model to directly infer the relative importance or connectivity across the national flight network. Graph algorithms are good at capturing this network information. Specifically, we use PageRank to measure the relative importance of each node based on both the number and the quality of its connections. Instead of only focusing on directly connected neighbors, PageRank

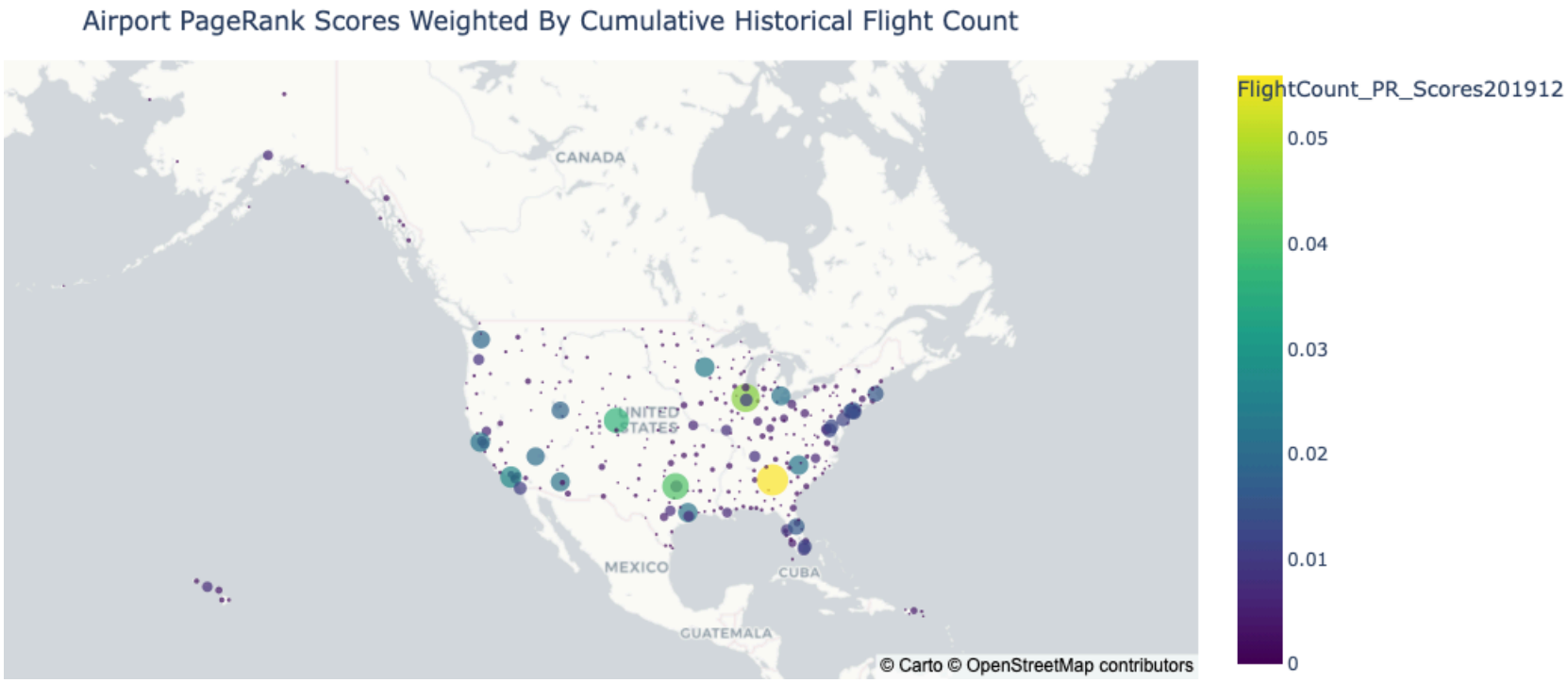
recursively incorporates the importance of upstream and downstream nodes to capture indirect influence across the airport network. This produces a more expressive structural variable that reflects how central an airport is within the broader air transportation system.

1. Airport PageRank Scores

- Unweighted Airport Page Rank Score: This score focuses more on structural connectivity. A route with 1 flight/day has the same influence as a route with 100 flights/day on the originating airport's rank distribution. Such PageRank scores are good for identifying airports that are topologically critical (bottlenecks in the structure).
- Airport Page Rank Score Weighted By All 5 Year Traffic Volume: After weighting every edge by all 5 years' flight count, high volume connections are more influential in the graph structure. This means, a route with 100 flights/day has a much greater influence on the originating airport's rank distribution. This measure is better for identifying airports that are operationally critical and where a delay is likely to affect the greatest number of subsequent flights and passengers.
- Page Rank Scores Weighted By Cumulative Flight Count: To avoid introducing future information into the model, we compute PageRank scores in a time-aware manner. Specifically, we group the data by year and month and calculate cumulative flight counts up to each year–month period. By recomputing the network structure incrementally at each time step, we obtain PageRank values that reflect only historical information available at that point in time to prevent any leakage from future activity. This PageRank score is the only version that we adopted and merged back as features based on the airport ID, year, and month. Finally, two features are created: `Origin_FlightCount_PR_Scores` and `Dest_FlightCount_PR_Scores`, and they represent the how important the airport is as of the end of the specific year-month.

We first constructed a directed graph in which each airport is a node, and each flight is an edge from the origin to the destination. We then assign edge weights using cumulative flight counts up to the year month period, so routes with higher traffic contribute proportionally more to the network structure. The key fields used to construct the graph are

the airport ID for each flight's origin and destination, and the flight count is calculated by the record count.



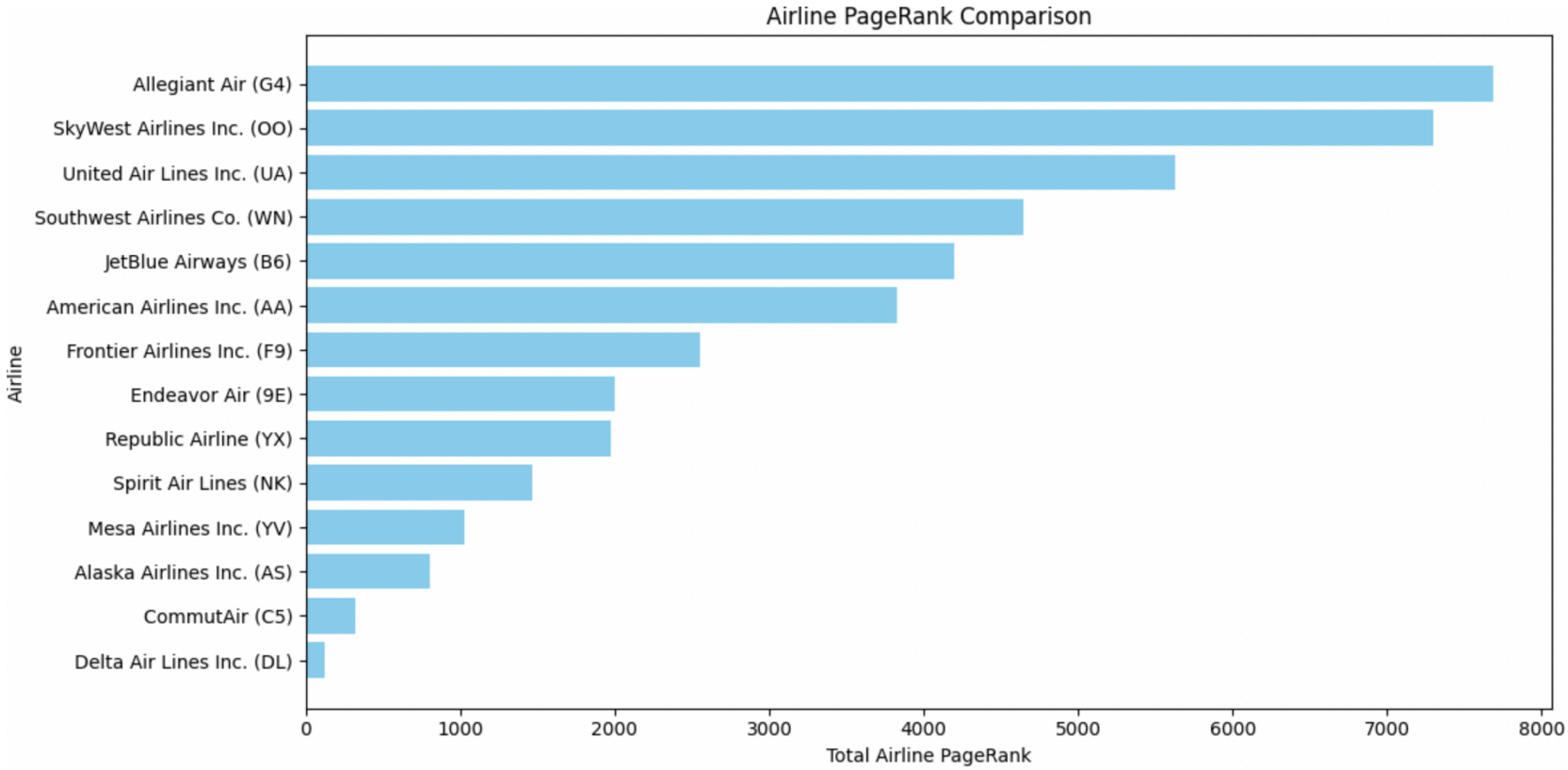
On the map, airports with higher PageRank scores appear as larger circles. As expected, the largest dots correspond to major hub airports. For example, the yellow dot represents ATL, reflecting its high connectivity and traffic volume.

2. Airline PageRank

The feature we are deriving is airline_pagerank, a network-aware metric that quantifies the importance of airlines within the flight network. By constructing a graph of flight routes and applying the PageRank algorithm, we capture an airline's influence not only by the number of flights it operates, but also by the strategic significance of the routes it serves.

We computed two metrics: *total_airline_pagerank* (sum of origin PageRank across an airline's routes) and *avg_airline_pagerank*(mean origin PageRank). These airline-level scores serve as proxies for operating in high-centrality regions of the network, capturing exposure to structurally critical airports where disruptions can cascade. Raw airport IDs are categorical and do not encode network centrality, hub behavior, or potential delay propagation, but PageRank provides a continuous signal of airline and airport importance, accounting for both the number and significance of connections. Using flight records from 2015–2019, we aggregated flights to the route level per airline and computed the number of flights.

Airline PageRank reflects both direct and indirect connectivity, offering a richer signal than simple counts or averages. This feature can enhance predictive models by better capturing operational centrality, reliability, and exposure to network disruptions, complementing traditional airline performance metrics.



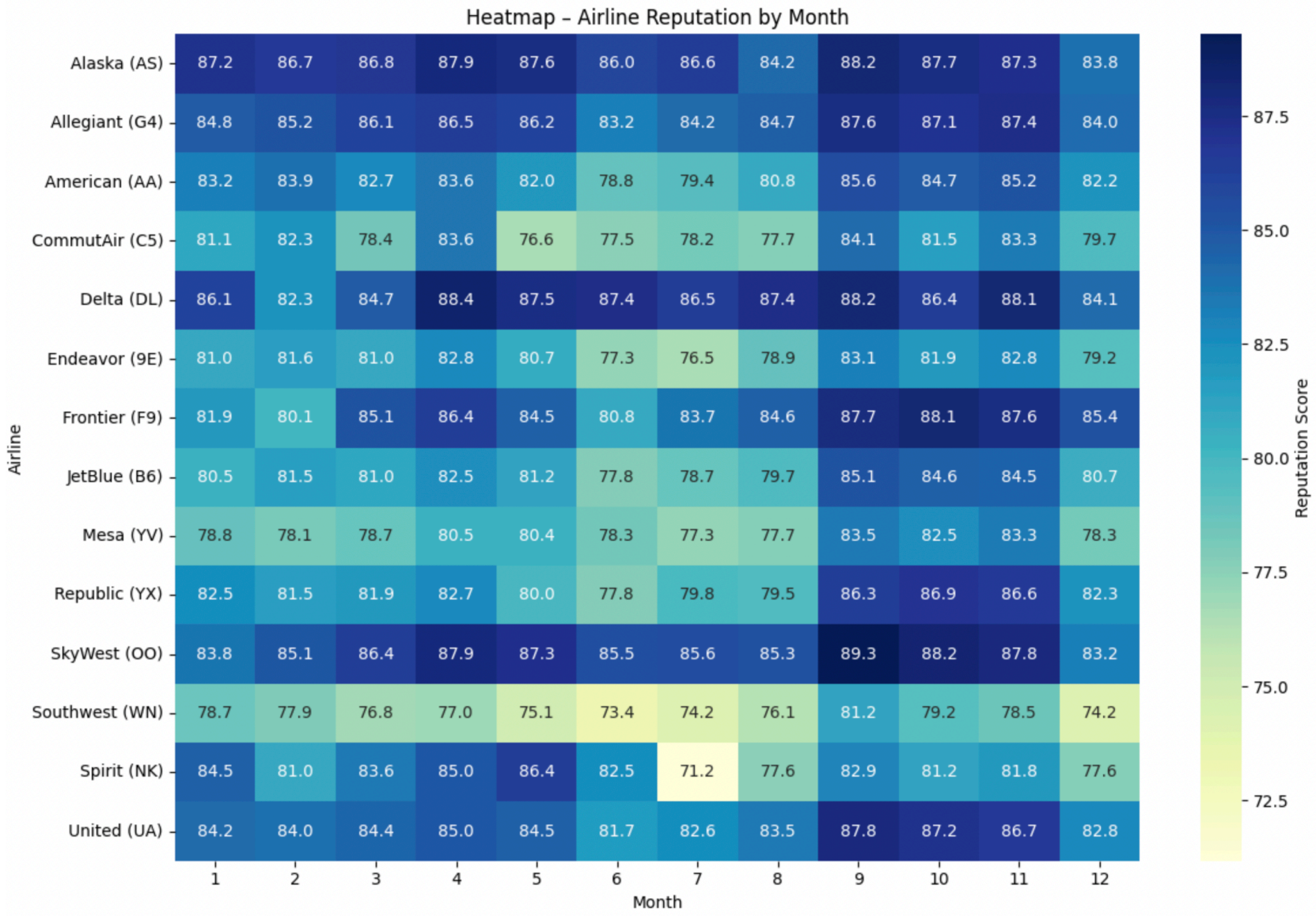
Airline Reputation

The feature we are deriving is *airline_reputation_score*, a network-agnostic metric that quantifies airline performance based on operational reliability. By aggregating monthly flight records, we compute weighted contributions from three key aspects: on-time arrivals, cancellations, and diversions.

We combine these metrics using configurable weights to produce a single score (reputation_score_0_100) that reflects overall reliability, where higher values indicate better performance. This feature provides a continuous, interpretable signal that goes beyond simple counts of flights, capturing both the frequency of disruptions and the airline's ability

to maintain scheduled operations. Using flight records from 2015–2019, we group flights by airline, year, and month to compute rates of on-time performance, cancellations, and diversions, and then combine them into a weighted score.

By summarizing operational reliability into a single numeric metric, airline reputation score serves as a powerful feature in predictive models. It can be used to forecast delays, assess operational risk, or benchmark airline performance across time periods and network segments. Since it captures the combined effect of multiple reliability indicators, it provides a richer signal than individual metrics alone and can help models better differentiate airlines that are more consistent and resilient in their operations. Additionally, it is directly interpretable for stakeholders, offering a clear, actionable view of airline reliability that can complement network-based features such as airline PageRank.


Heatmap – Airline Reputation by Month
Airline
Alaska (AS) 87.2 86.7 86.8 87.9 87.6 86.0 86.6 84.2 88.2 87.7 87.3 83.8
Allegiant (G4) 84.8 85.2 86.1 86.5 86.2 83.2 84.2 84.7 87.6 87.1 87.4 84.0
American (AA) 83.2 83.9 82.7 83.6 82.0 78.8 79.4 80.8 85.6 84.7 85.2 82.2
CommutAir (C5) 81.1 82.3 78.4 83.6 76.6 77.5 78.2 77.7 84.1 81.5 83.3 79.7
Delta (DL) 86.1 82.3 84.7 88.4 87.5 87.4 86.5 87.4 88.2 86.4 88.1 84.1
Endeavor (9E) 81.0 81.6 81.0 82.8 80.7 77.3 76.5 78.9 83.1 81.9 82.8 79.2
Frontier (F9) 81.9 80.1 85.1 86.4 84.5 80.8 83.7 84.6 87.7 88.1 87.6 85.4
JetBlue (B6) 80.5 81.5 81.0 82.5 81.2 77.8 78.7 79.7 85.1 84.6 84.5 80.7
Mesa (YV) 78.8 78.1 78.7 80.5 80.4 78.3 77.3 77.7 83.5 82.5 83.3 78.3
Republic (YX) 82.5 81.5 81.9 82.7 80.0 77.8 79.8 79.5 86.3 86.9 86.6 82.3
SkyWest (OO) 83.8 85.1 86.4 87.9 87.3 85.5 85.6 85.3 89.3 88.2 87.8 83.2
Southwest (WN) 78.7 77.9 76.8 77.0 75.1 73.4 74.2 76.1 81.2 79.2 78.5 74.2
Spirit (NK) 84.5 81.0 83.6 85.0 86.4 82.5 71.2 77.6 82.9 81.2 81.8 77.6
United (UA) 84.2 84.0 84.4 85.0 84.5 81.7 82.6 83.5 87.8 87.2 86.7 82.8
1 2 3 4 5 6 7 8 9 10 11 12
Month
Reputation Score
87.5
85.0
82.5
80.0
77.5
75.0
72.5

Event-based features

To capture non-routine travel surges that aren't fully explained by seasonality or weather, we created a unified set of event-based features covering both nationwide and location-specific events. These features help the model learn when airports and routes experience abnormal demand, congestion, and operational stress—often leading to higher delay risk.

1. **Federal Holiday Indicators** (Nationwide Demand Shifts) Federal holidays typically change travel patterns across the entire network (long-weekend spikes, return-day peaks, and shifted business travel). We encode holidays as calendar-based flags (and optionally holiday windows like ±1–3 days) to capture these broad demand effects.
2. **Major Event Windows** (Localized Demand Shocks) For large, scheduled events that drive concentrated travel into specific cities/airports, we flag flights occurring within an event travel window (e.g., ±3 days) and tied to relevant airports when applicable. Below are small number of recurring, high-impact events were included to capture predictable demand spikes while avoiding noise from irregular or low-impact events.

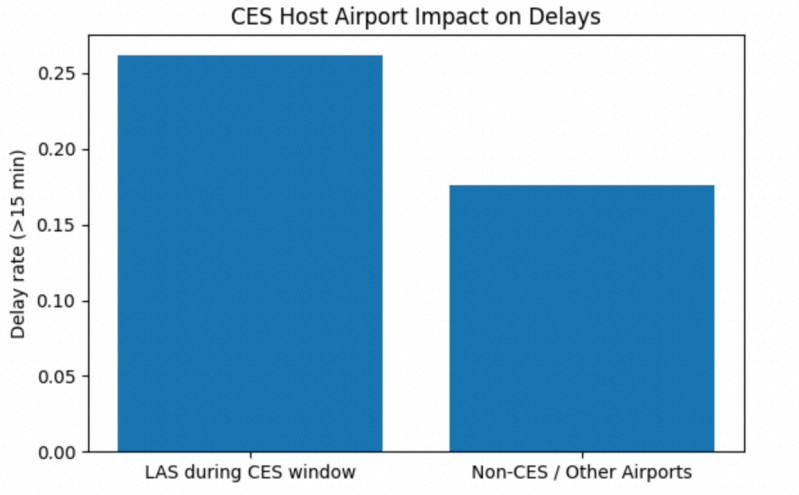
Super Bowl: This feature was selected because the Super Bowl is a recurring, nationally significant event that generates short-term, airport-specific travel demand spikes at clearly defined host locations each year. We encode this effect using a window-based indicator for flights occurring near the event date and involving host airports as either the origin or destination.

Grace Hopper Celebration (GHC): This feature was included due to the conference's consistent annual schedule and its tendency to concentrate travel into a small set of host cities, resulting in predictable, localized congestion. We capture this pattern using a window-based flag around conference dates, along with an interaction feature that highlights peak impact at known host airports.

CES (Consumer Electronics Show): This feature was chosen because CES occurs annually in Las Vegas and reliably drives a large influx of travelers into a single airport hub, producing a strong and geographically focused demand surge. We model this effect with a window-based indicator centered on CES dates, mapped specifically to Las Vegas-area airports

These event indicators provide the model with a clean, interpretable signal of travel demand spikes and congestion risk, improving prediction performance during atypical periods without requiring the model to infer event effects indirectly from noisy operational patterns.

For example Exploratory analysis of event-based features shows a pronounced increase in delay risk at the CES host airport (LAS) during the event travel window. Flights departing from LAS during CES exhibit substantially higher delay rates than flights operating outside the event window or at other airports. This confirms that major conferences create localized demand shocks and validates the use of airport-specific interaction features to capture non-routine congestion effects.



Natural disasters

Natural disasters can cause severe, localized disruptions (airport closures, airspace restrictions, staffing impacts, reroutes) that drive abnormal delays. We engineered disaster-based features to explicitly flag flights operating during known disaster periods near affected airports.

Since flights contain airport codes (e.g., ORIGIN) but disaster records are state-based, we first map each origin airport to its state abbreviation.

We created a structured disaster table with start/end dates and join it to the airport–state map to produce an airport-level disaster reference (DISASTER_ORIGIN, date range). Flights are left-joined to the disaster reference when:

ORIGIN matches the affected airport, and FL_DATE falls between the disaster START_DATE and END_DATE.

Below are the features created:

is_disaster_day (binary): 1 if the flight occurs at an affected origin airport during an active disaster period, else 0.

days_since_disaster (numeric): Number of days since the disaster started (captures ramp-up / lingering recovery effects).

disaster_severity (ordinal): A mapped severity score (e.g., hurricanes higher than storms/tornado outbreaks) to represent expected disruption intensity.

Disaster Severity Definition

According to the Natural Disaster dataset, 4 classes are specified below. Disaster severity was encoded as an ordinal scale based on historical impact, geographic scope, and operational disruption severity:

Severity 5 (Catastrophic hurricanes): Hurricane Harvey, Hurricane Maria

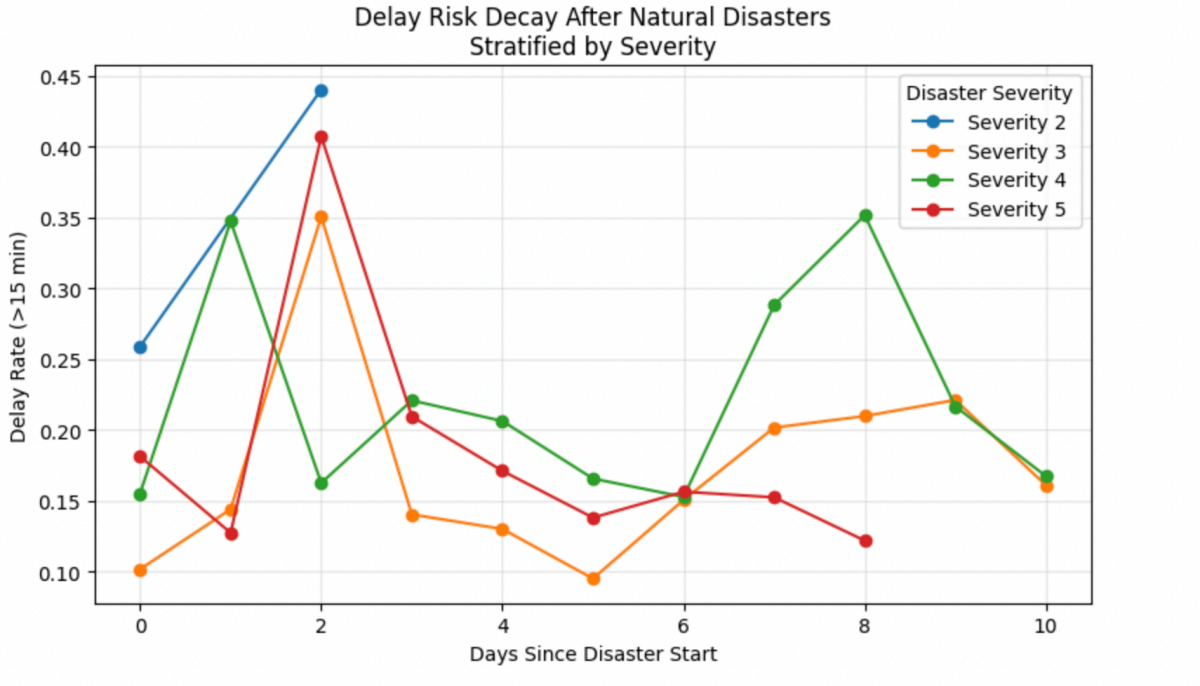
Severity 4 (Major hurricanes / large-scale disasters): Hurricane Irma, Camp Fire Wildfire, Midwest Bomb Cyclone Floods

Severity 3 (Regional flooding / moderate hurricanes): Hurricane Florence, Louisiana Floods, West Virginia Floods

Severity 2 (Localized severe weather events): Winter Storm Juno, Moore Tornado Outbreak

This mapping reflects increasing levels of infrastructure damage, airport closures, airspace restrictions, staffing disruptions, and recovery time. Below figure illustrates how delay risk evolves in the days following the onset of a natural disaster, stratified by disaster severity. Higher-severity events (severity 4–5) exhibit a pronounced spike in delay rates immediately after the disaster begins, followed by a gradual decay over subsequent days, reflecting recovery and stabilization of airport operations.

Lower-severity events show smaller initial impacts and faster normalization, indicating more limited operational disruption. This pattern validates both the inclusion of a temporal decay feature (days_since_disaster) and an ordinal severity signal (disaster_severity), as delay risk is clearly non-uniform across disaster types and persists beyond the event start date.



Airport maintenance

To model the impact of airport maintenance and operational disruptions on flight performance, the following features were derived from known disruption events and aligned at a daily level:

severity_num: A numeric encoding of disruption severity, mapped from categorical labels (low, medium, high, very high, extreme). This captures the relative intensity of operational impact on airport capacity.

days_into_disruption: The number of days since the disruption began, measuring how far the operation is into an ongoing maintenance or disruption period and capturing ramp-up and persistence effects.

event_type: Categorical indicator describing the nature of the disruption (e.g., runway construction, taxiway maintenance, power outage, ATC staff shortage, government shutdown), allowing the model to differentiate between types of operational constraints.

airport: The affected airport code (or ALL for nationwide events), used to localize the impact of the disruption.

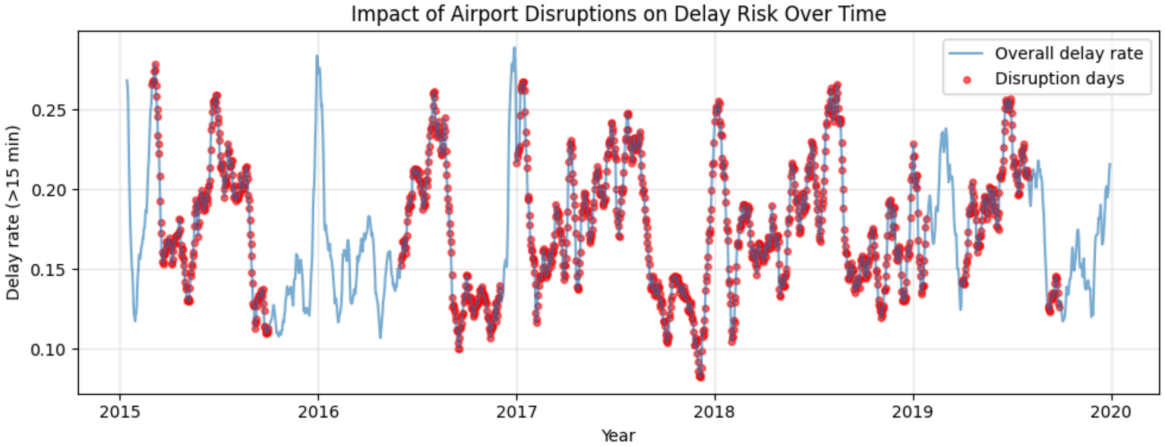
event_date: The specific calendar date during which the disruption is active, enabling precise alignment with flight-level records.

Together, these features provide structured signals that represent both the severity and temporal evolution of airport maintenance and operational events, improving the model's ability to account for sustained capacity constraints and sudden infrastructure-related disruptions.

Airport Maintenance and Operational Disruptions - EDA

The figure overlays the daily network-wide delay rate with indicators for airport disruption days. Disruption days consistently exhibit elevated delay rates relative to the underlying seasonal baseline, forming visible clusters above the normal delay envelope.

Importantly, the effect persists across multiple consecutive days, indicating that airport disruptions introduce sustained operational stress rather than isolated one-day anomalies. This pattern is observed consistently across multiple years, supporting the inclusion of disruption flags, severity indicators, and time-since-disruption features in the model.



Weather Features

Hourly Sky Condition

`HourlySkyConditions` is an informative feature because sky conditions directly influence airport operations, pilot visibility, runway capacity, and ultimately the likelihood of departure delays. However, the raw variable is difficult to use: each observation often contains multiple cloud layers concatenated into a single string. A single record can report up to three layers with distinct coverage codes and base heights—for example, "SCT020 BKN050 OVC100".

To make this variable usable for modeling, we parsed the text using `regular expressions` and extracted each cloud layer into separate columns. The coverage codes themselves carry specific operational meaning. A reference summary is provided below:

Coverage Code	Coverage Code Meaning	Layer Amount	Three Digits Meaning	Category
CLR	all clear sky	00	none expected	clear sky

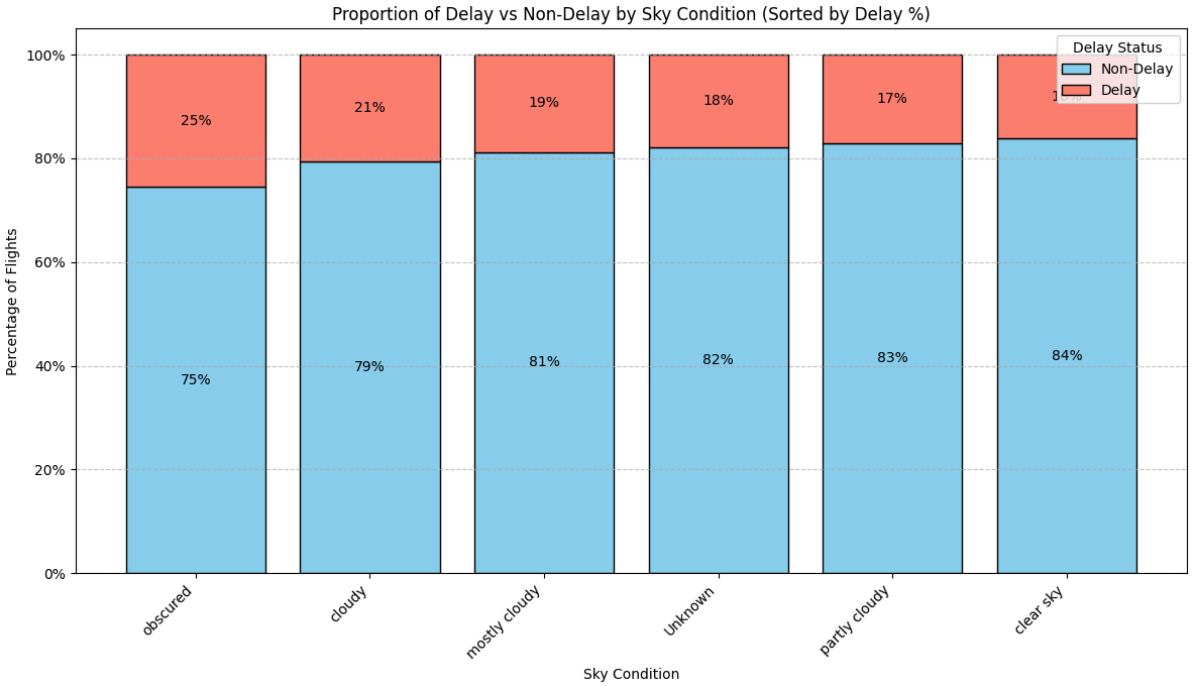
Coverage Code	Coverage Code Meaning	Layer Amount	Three Digits Meaning	Category
FEW	few clouds	01-02	cloud base height	partly cloudy
SCT	scattered clouds	03-04	cloud base height	partly cloudy
BKN	broken clouds	05-07	cloud base height	mostly cloudy
OVC	overcast	08	cloud base height	cloudy
VV	obscured sky	09	vertical visibility	obscured
10	partially obscured sky	x	x	obscured

The original Sky Condition string has three layers. `l1_sig`, `l2_sig`, and `l3_sig` are intermediate fields which represent each layer class after parsing. Each layer is then classified into a human-interpretable category. Finally, the overall operational sky condition is determined using `l1_sig` because `l1_sig` is the lowest non-clear sky layer. More details of how this logic is derived are described in the next paragraph.

HourlySkyConditions	l1_sig	l2_sig	l3_sig	overall_sky_condition
OVC:08 32	cloudy	null	null	cloudy
BKN:07 8 OVC:08 14	mostly cloudy	cloudy	null	mostly cloudy
FEW:02 65 SCT:04 250	partly cloudy	partly cloudy	null	partly cloudy
CLR:00	null	null	null	clear sky
FEW:02 50	partly cloudy	null	null	partly cloudy

According to the NOAA LCD documentation (https://www.ncei.noaa.gov/pub/data/cdo/documentation/LCD_documentation.pdf), climatologists traditionally assign the “overall” sky condition using the highest cloud layer reported. For example, `SCT020 BKN050 OVC100` would be labeled OVC because the ten-thousand-foot layer is the highest. However, this convention is not well-suited to operational delay prediction. For airport operations, the lowest significant cloud layer is far more relevant because it constraints pilot visibility minima, runway usage, and approach/departure procedures. A higher layer with light coverage (e.g., SCT or FEW) does not mitigate the operational impact of a lower layer with heavy coverage (e.g., BKN or OVC). Using the highest layer would therefore introduce noise and reduce the feature’s predictive power. For this reason, we defined the overall sky condition as the **lowest non-CLR layer** in the report. This operational definition better aligns with real-world aviation impacts and improves model signal quality. Finally, we mapped coverage codes into six classes — clear sky, partly cloudy, mostly cloudy, cloudy, obscured, and unknown.That way we can redue category sparsity while preserving meaningful distinctions for the model. Overall, we processed the original free-form text using regular expressions to extract up to three layers, transformed each layer into an interpretable category, and selected the lowest significant layer as the overall sky condition used in downstream modeling.

The stacked bar plot below summarizes how each category correlates with flight delay outcomes. The overcast condition exhibits the highest proportion of delayed departures (25%), indicating that low, dense cloud cover poses the most operational constraints for airports.



Weather Index

The Weather_Index is designed to condense multiple continuous weather measurements into a single numeric feature. This allows the model to capture overall weather severity and operating conditions without explicitly handling many highly correlated weather variables.

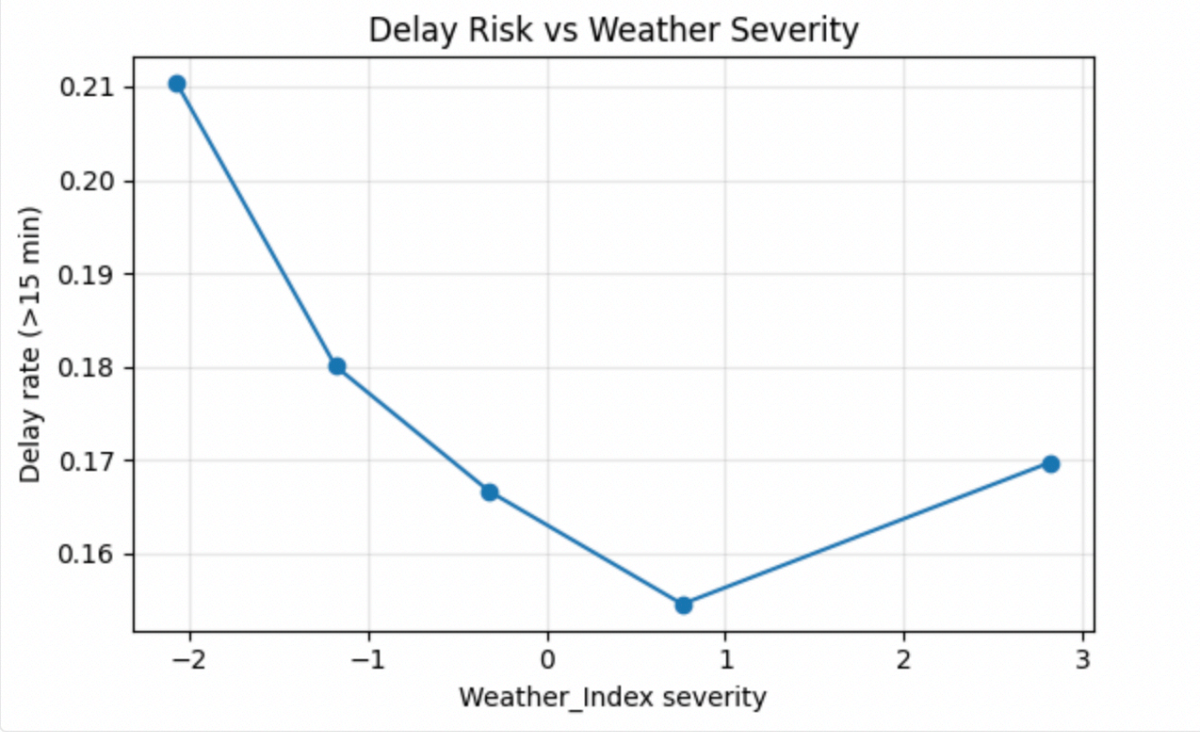
Inputs (Continuous Weather Columns):

The index is built using 10 hourly weather signals, including precipitation, pressure-related measures (sea level pressure, station pressure, altimeter), temperature-related measures (dry bulb, wet bulb, dew point), wind speed, relative humidity, and visibility.

These weather variables are first combined into a standardized feature vector to ensure all measurements are on a comparable scale. We then apply **Principal Component Analysis (PCA)** and retain only the first principal component, which represents the strongest shared pattern across all weather variables and effectively summarizes overall weather conditions.

The resulting principal component score is extracted as a single scalar feature called **Weather_Index**, and intermediate columns used during processing are removed to keep the dataset clean and model-ready.

From below EDA Delay probability varies nonlinearly with Weather_Index severity. Flights operating under moderate weather conditions exhibit the lowest delay rates, while both extreme-weather regimes—corresponding to severe conditions on either end of the index—show elevated delay risk. This U-shaped relationship reflects the fact that the Weather_Index captures multiple operational stress regimes (e.g., winter weather vs. convective storms). These results confirm that Weather_Index provides meaningful predictive signal and motivates the use of nonlinear models and quantile-based splits.



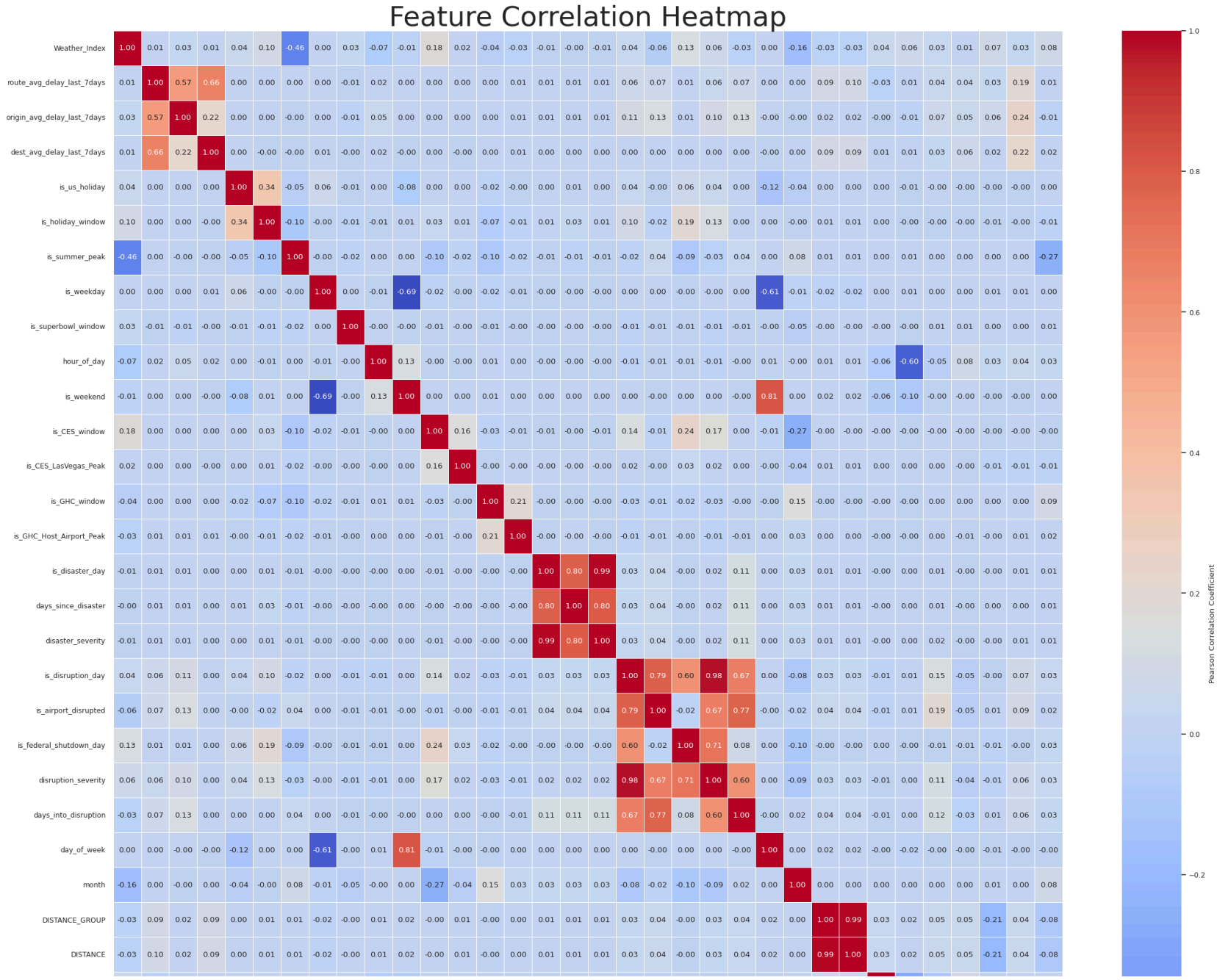
Experiments Demonstrating Feature Value:

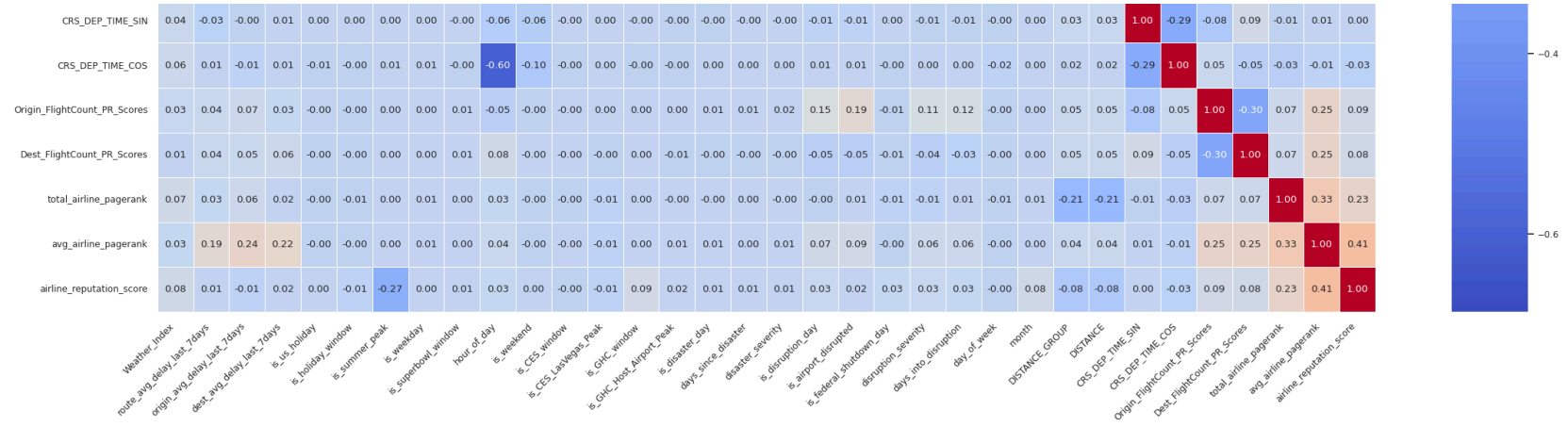
To validate the usefulness of the engineered features and guide feature selection, we conducted a series of statistical and model-based experiments aimed at identifying redundant features, assessing individual feature relevance, and confirming which feature families contribute meaningful predictive signal.

In addition to improving model quality and interpretability, these experiments were also designed to reduce overall model complexity and training runtime by eliminating redundant or low-impact features, which is particularly important given the scale of the multi-year flight dataset.

1. Correlation Analysis (Redundancy Detection)

As a first step, we performed a Pearson correlation analysis across engineered features to detect redundancy and multicollinearity. This analysis was used strictly for feature pruning rather than for measuring predictive power. Several highly correlated feature pairs were identified ($|r| > 0.95$), including disaster_severity and is_disaster_day ($r = 0.987$), DISTANCE and DISTANCE_GROUP ($r = 0.988$), and disruption_severity and is_disruption_day ($r = 0.979$). These results indicate that certain binary indicator variables encode nearly identical information to their corresponding severity or grouped features. To reduce redundancy, improve numerical stability, and simplify the feature space, we retained the more expressive variables (e.g., severity scores or continuous distance) while dropping their highly correlated counterparts. In addition to improving model interpretability, this feature consolidation step was intentionally used as a form of feature selection to reduce overall model complexity and training runtime, which is especially important given the large-scale, multi-year nature of the dataset. Overall, the correlation analysis validated our feature consolidation decisions and helped reduce multicollinearity without sacrificing predictive information.


Feature Correlation Heatmap
Weather_Index
route_avg_delay_last_7days
origin_avg_delay_last_7days
dest_avg_delay_last_7days
is_us_holiday
is_holiday_window
is_summer_peak
is_weekday
is_superbowl_window
hour_of_day
is_weekend
is_CES_window
is_CES_LasVegas_Peak
is_GHC_window
is_GHC_Host_Airport_Peak
is_disaster_day
days_since_disaster
disaster_severity
is_disruption_day
is_airport_disrupted
is_federal_shutdown_day
disruption_severity
days_into_disruption
day_of_week
month
DISTANCE_GROUP
DISTANCE
Pearson Correlation Coefficient
1.0
0.8
0.6
0.4
0.2
0.0
−0.2



2. ANOVA F-Test (Linear Separability Assessment)

We next applied one-way ANOVA F-tests to assess whether individual continuous features exhibit statistically significant mean differences between delayed and non-delayed flights.

Because the full dataset is large, we first created a representative stratified sample to keep the analysis computationally tractable while preserving class balance. Specifically, we binarized the target (DEP_DELAY_GROUP) into 0 = not delayed (original labels −1 and 0) and 1 = delayed (all other labels), then performed stratified sampling using sampleBy with the same sampling fraction (3.7%) for each class and a fixed random seed (42) for reproducibility. This produced a sample of approximately 1.12M rows, maintaining proportional coverage of both delay classes.

We then evaluated a broad set of continuous, engineered, and ordinal features (weather signals, location/distance, rolling delay trends, cyclic time encodings, disruption/disaster indicators, PageRank-based network features, and airline reputation), computing an F-statistic and p-value for each feature. The selected results showed that some variables (e.g., hour_of_day, days_since_disaster, disaster_severity, LATITUDE) did not demonstrate strong linear separability when tested independently.

Overall, the ANOVA study primarily informed modeling strategy and reinforced the need for nonlinear models, rather than serving as a strict rule for feature removal.

3. Model-Based Validation (XGBoost Feature Importance)

To assess the true predictive contribution of each engineered feature family, we examined feature importance scores from a separately trained XGBoost model used exclusively for feature validation. This is not the final production model.

Unlike univariate statistical tests, this model-based evaluation captures nonlinear relationships and complex feature interactions that are critical in flight delay prediction.

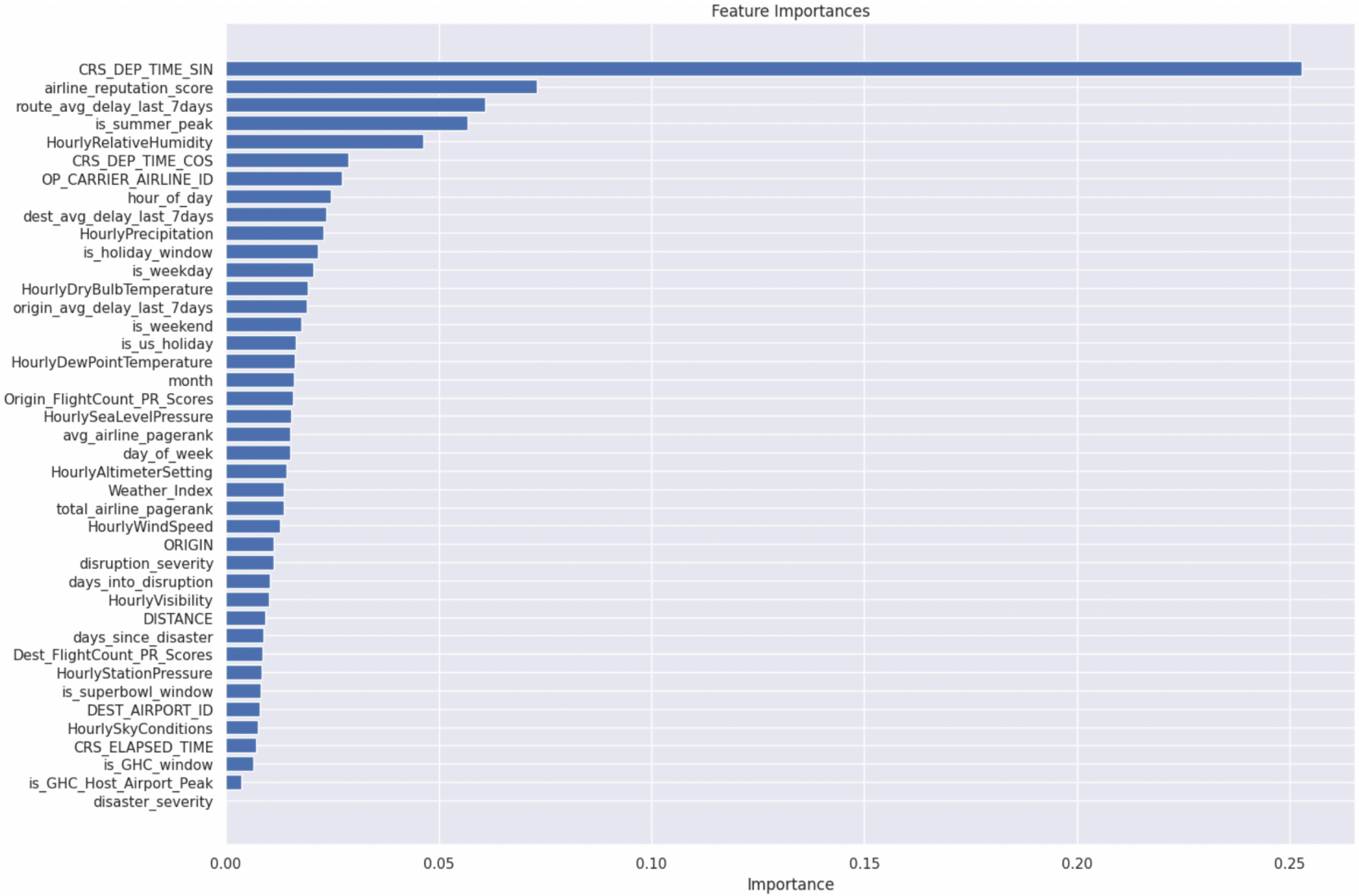
From below EDA we understand that time-based features dominate model importance, with cyclic encodings of scheduled departure time (CRS_DEP_TIME_SIN and CRS_DEP_TIME_COS) emerging as the strongest predictors, alongside hour_of_day, month, is_weekday, and is_summer_peak, confirming strong daily and seasonal delay patterns. Recent delay trend features such as route_avg_delay_last_7days, origin_avg_delay_last_7days, and dest_avg_delay_last_7days rank highly, demonstrating that short-term operational congestion is highly predictive.

Airline and network-level features, including airline_reputation_score and PageRank-based airport traffic scores, further validate the importance of airline reliability and network centrality.

In addition, weather-related features—including relative humidity, precipitation, temperature, dew point, and sea-level pressure—contribute meaningful signal, confirming that both individual meteorological variables and aggregated weather effects influence delay risk.

Finally, event and calendar features such as is_holiday_window and is_us_holiday retain non-trivial importance, capturing nationwide demand shifts not fully explained by seasonality alone.

Overall, this model confirms that all major feature families provide meaningful predictive signal and informs feature retention for the final model, rather than serving as the final trained system itself.



Leakage

Data leakage occurs when information from outside the training period or information that would not realistically be available at prediction time, this will end up in the model's training data. This causes the model to "cheat" by learning patterns that it should not have access to, leading to overly optimistic performances during training but non-optimal performance on truly unseen future data.

For example, suppose we are predicting whether a flight will be delayed. If our training dataset includes a column like "actual arrival time" or "final delay minutes", we would use it as a feature. The model effectively sees the answer during training. It will perform extremely well in validation but will fail in real world prediction, because such information is not available before the flight departs.

Our model pipeling case:

- In our time-series case, because our project involves time-dependent data, we must respect the temporal order. If using standard K-fold cross-validation, this would mix future data into the training folds as a form of leakage. Instead, we use a time series split, where each training set contains only data from the past, and the test set contains data from the future. This ensures the model is evaluated in the same way that it will be used operationally (predicting future outcomes from past data) and prevents leakage from future information.
- In our Airport PageRank graph case, we group the data by year and month and calculate cumulative flight counts up to each year–month period. By recomputing the network structure incrementally at each time step, we obtain PageRank values that reflect only historical information available at that point in time to prevent any leakage from future activity.

Therefore, all cardinal sins of ML have been addressed: we ensured proper time-series handling with no data leakage, applied PageRank without mixing future information, removed irrelevant or problematic columns, and handled missing values through careful imputation.

Modeling Pipelines

1. Pipeline overview

For our neural network model - Multilayer Perceptron (MLP), baseline logistic regression, and our subsequent ensemble tree models (Random Forest Classifier + gradient boosted XGBoost Classifier) all use the following:

- **Target label**: See the Target Variable distribution on the classification of the classes for `IS_DEP_DELAY ∈ {delayed, not_delayed}`.
- **Goal**: maximize **recall for the delayed class**, while monitoring F1 and precision to avoid a model that simply "cries delay" on every flight.

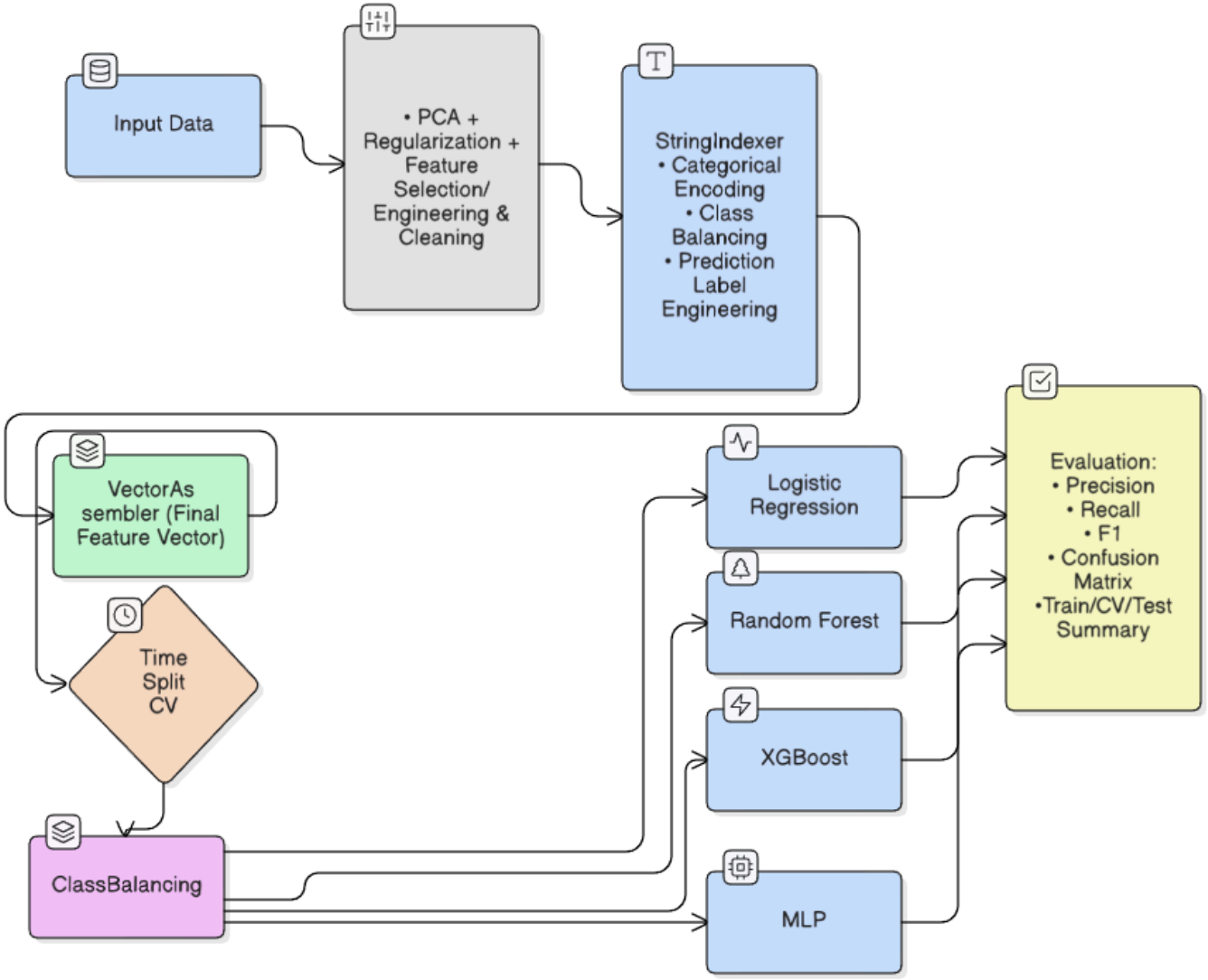
The high-level pipeline to a model is:

Stage	Purpose	Key details / examples
Raw data	Provide historical flights & context	Five years of OTPW flight records (2015–2019) joined with hourly Weather at origin airports and disruption signals
Label engineering	Define prediction target	Map `DEP_DELAY_GROUP` → {**delayed**, **not_delayed**} to build a binary delay classifier

Stage	Purpose	Key details / examples
Feature engineering	Build informative predictors	Calendar features (year, month, day-of-week, hour, season, holidays), route & distance, 7-day rolling delay stats, hourly weather, disruption / event flags
Train/validation/test split	Respect time order & avoid leakage	Train/CV on 2015–2018; **2019** held out as a blind test year
Class imbalance handling	Address strong class imbalance	Inverse-frequency weights per class / Downsample majority (`not_delayed`) per (YEAR, MONTH)
Modeling pipeline (Spark ML)	Transform features and fit model	`StringIndexer` → (low-card) `OneHotEncoder` + (high-card) indexed only → numeric `StandardScaler` → `VectorAssembler` → MLP / Random Forest / XGBoost
(Secondary branch) Evaluation (CV on 2015–2018)	Tune hyperparameters and assess generalization	Time-aware rolling CV (2015→2016, 2015–16→2017, 2015–17→2018); metrics per fold: precision / recall / F1 for **delayed**; confusion matrix
Final evaluation (2019 test)	Report blind test performance for business decisions	Train best model on 2015–2018; compute precision, recall, F1 (delayed) and confusion matrix on 2019

Note: with the five-year dataset we use **years** for time-based splits: 2015–2018 for model selection via rolling CV, and 2019 as a completely held-out test year.

Shared Diagram:



2. Input feature families

After all joins and feature engineering, the schema used for modeling is:

Feature name	Family
CRS_DEP_TIME_SIN	Time & calendar / season
CRS_DEP_TIME_COS	Time & calendar / season
hour_of_day	Time & calendar / season
day_of_week	Time & calendar / season
month	Time & calendar / season
is_weekday	Time & calendar / season
is_weekend	Time & calendar / season
is_us_holiday	Time & calendar / season
is_holiday_window	Time & calendar / season
is_summer_peak	Time & calendar / season
OP_CARRIER_AIRLINE_ID	Airline-level
airline_reputation_score	Airline-level
avg_airline_pagerank	Airline-level
total_airline_pagerank	Airline-level
route_avg_delay_last_7days	Route / historical delay

Feature name	Family
origin_avg_delay_last_7days	Route / historical delay
dest_avg_delay_last_7days	Route / historical delay
Origin_FlightCount_PR_Scores	Route / historical delay
Dest_FlightCount_PR_Scores	Route / historical delay
HourlyRelativeHumidity	Weather at origin
HourlyPrecipitation	Weather at origin
HourlyDryBulbTemperature	Weather at origin
HourlyDewPointTemperature	Weather at origin
HourlySeaLevelPressure	Weather at origin
HourlyAltimeterSetting	Weather at origin
HourlyWindSpeed	Weather at origin
HourlyVisibility	Weather at origin
HourlyStationPressure	Weather at origin
Weather_Index	Weather at origin
ORIGIN	Route / geography
DISTANCE	Route / geography

Feature name	Family
CRS_ELAPSED_TIME	Route / geography
days_into_disruption	Disruption context
disruption_severity	Disruption context

Tabulation of the above:

Feature Family	Count
Time & calendar / season	10
Airline-level	4
Route / historical delay	5
Weather at origin	10
Route / geography	3
Disruption context	2
Total features	**34**

3. Metrics used

We focused on binary metrics for the delayed class. **Our main objective is the Recall metric for delayed flights.** This addresses the question of: “If a flight is truly delayed, how often do we identify it?”

We also report the F1 score to summarize the trade-off between precision and recall and visualized on the True/False Positives/Negatives in a confusion matrix.

The following are the calculations:

- **Precision**
  - **Description:** The proportion of all classifications being actually positive

$$\text{Precision} = \frac{TP}{TP + FP}$$

- **Recall**
  - **Description:** The proportion of all actual positives that were classified correctly as positive.

$$\text{Recall} = \frac{TP}{TP + FN}$$

- **F1-Score**
  - **Description:** F1 Score calculates the harmonic mean between the Precision and Recall values.

$$\text{F1} = 2 * \frac{Precision * Recall}{Precision + Recall}$$

4. Loss functions

The following are the loss functions present in the models:

For Logistic Regression, MLP and XGBoost

- **Loss/Criteria:**

- *Log Loss:*

$$\text{LogLoss} = -\frac{1}{N}\sum_{i=1}^{N}\left[y_i \log \hat{p}_i + (1 - y_i)\log(1 - \hat{p}_i)\right]$$

For Random Forest

- **Loss/Criteria:**
  - *Entropy (per tree):*

$$H(S) = -\sum_{k=1}^{K} p_k \log p_k$$

[The tree grows by maximizing impurity reduction (information gain).]

The following are the regularization formulas used per our models:

XGBoost using both Ridge and Lasso regularization where

- λ: L2 (Ridge) regularization
- α: L1 (Lasso) regularization
- γ: Leaf penalty

$$\Omega(f_t) = \gamma T + \frac{1}{2}\lambda\sum_{j=1}^{T} w_j^2 + \alpha\sum_{j=1}^{T} |w_j|$$

5. Cluster Description

We are currently using a cluster noode of m5d.2xlarge with a minimum of 6 workers and a maximum of 12 workers. Detailed approximate exevution time for the models are described in their respective sections below.

Per Model Pipeline Setup and Configurations

Logistic Regression (LR)

Logistic Regression is a binary classification model between the 2 classes.

Handling class imbalance

To address label imbalance, we use inverse-frequency class weights on the training set:

1. Let n_c be the number of training examples in class c, $N = \sum_c n_c$ be the total number of training examples, and K be the number of classes ((K=2) here).
2. The base weight for class (c) is:

$$(w_c^{(0)} = \frac{N}{K, n_c})$$

3. We then raise weights to a tunable exponent alpha:

$$w_c = \left(w_c^{(0)}\right)^{\alpha}$$

- alpha = 1: standard inverse-frequency reweighting
- alpha < 1: softer reweighting

- alpha > 1: more aggressive up-weighting of rare classes

Cross-validation setup and number of experiments

We used the rolling cross validation for the training window (years 2015 to 2018) where year 2019 is held out for test.

- Fold 1: Train on year 2015, Validate on 2016
- Fold 2: Train on years 2015 and 2016, Validate on 2017
- Fold 3: Train on years 2015 to 2017, Validate on 2018

Here is the parameter grid search space

Hyperparameter	Values
alpha	{0.5, 1.0}
regParam	{0.0, 1e-3, 1e-2}
elasticNetParam	{0.0, 0.5}

This yields 2 * 3 * 2 = 12 logistic regression configurations. Each configuration is evaluated on 3 folds, so the grid search requires:

- 12 configurations * 3 folds
- plus the class-weighted LR with the chosen “best” hyperparameters

In total, we ran 37 logistic regression trainings in Phase 3.

Runtime

• A single full-data weighted logistic regression fit on 2015-2018 takes about 5-6 minutes.
• The full grid search (12 configurations * 3 folds) completed in about 56 minutes.

Multilayer Perceptron (MLP)

MLP is a neural network that has various architectural layers to produce the classification.

Handling class imbalance

The training data are highly imbalanced: on-time flights substantially outnumber delayed flights. To avoid the network simply predicting “not_delayed” most of the time, we applied downsampling of the majority class. We implemented a custom downsample_majority function that:

1. Creates a (YEAR, MONTH) composite key for each flight.
2. For each (year, month) bucket, counts delayed vs. not_delayed flights.
3. Samples the majority class so that, within each year-month bucket.
4. Leaves all delayed flights in place and unions them with the sampled majority.

This preserves the temporal structure of the data while reducing imbalance. A ratio of maj_to_min_ratio = 1.0 corresponds to a roughly 50:50 class balance (about one on-time flight per delayed flight).

Cross-validation setup and number of experiments

For the neural network, we trained a Multilayer Perceptron classifier on five years of OTPW data (2015-2019). We treated 2015-2018 as the training period and held out 2019 as a blind test set that was never used for model selection. Within 2015-2018 we used time-based cross-validation:

- Fold 1: train on 2015 → validate on 2016
- Fold 2: train on 2015-2016 → validate on 2017

- Fold 3: train on 2015-2017 → validate on 2018

We ran a modest grid search around the following dimensions:

- Network architecture: 1-2 hidden layers, hidden sizes such as [32], [64, 32]
- Downsampling ratio: maj_to_min_ratio ∈ {1.0, 1.2}
- Decision threshold on P(delay): we evaluated 0.3, 0.5

For each grid point we performed the 3-fold time CV described above, then averaged validation metrics across the 2016-2018 folds. We ranked configurations primarily by recall and F1 on the delayed class. The best configuration for this MLP was:

- Hidden layers: [32]
- Downsampling ratio: maj_to_min_ratio = 1.0
- Threshold: 0.5

This setting provided a reasonable trade-off: it caught a majority of true delays while keeping false positives somewhat under control.

Runtime

On our Databricks cluster, a single MLP training run for one fold (including feature pipeline fit + MLP fit) takes around 3-5 minutes.
The full grid search (8 configurations × 3 folds ≈ 24 fits) completed around 90 minutes, plus one additional fit for the final 2015–2018 model used to evaluate on 2019, whichs takes around 6 mins.

Random Forest (RF)

Random Forest uses an ensemble of decision trees for binary classification. The regularization is controlled by tree depth, number of trees, and features.

1. Class imbalance and class weights

To address label imbalance, we use inverse-frequency class weights on the training set:

1. Let n_c be the number of training examples in class c, $N = \sum_c n_c$ be the total number of training examples, and K be the number of classes ((K=2) here).
2. The base weight for class (c) is:

$$(w_c^{(0)} = \frac{N}{K, n_c})$$

3. We then raise weights to a tunable exponent alpha:

$$w_c = \left(w_c^{(0)}\right)^{\alpha}$$

- alpha = 1: standard inverse-frequency reweighting
- alpha < 1: softer reweighting
- alpha > 1: more aggressive up-weighting of rare classes

2. Cross-validation setup and number of experiments

Likewise to the MLP Model, we used the rolling cross validation for the training window (years 2015 to 2018) where year 2019 is held out for test.

- Fold 1: Train on year 2015, Validate on 2016
- Fold 2: Train on years 2015 and 2016, Validate on 2017
- Fold 3: Train on years 2015 to 2017, Validate on 2018

Here is the parameter grid search space

Hyperparameter	**Values**
numTrees	{100, 125}
maxDepth	{8}
maxBins	{32, 64}
featureSubsetStrategy	{"auto"}

Due to the use of 3 folds and a grid search space, there are a total of 2 × 1 × 2 × 1 = 4 configurations

With cross validation, 4 configurations × 3 folds = 12 CV experiments.

3. Runtime

A single Random Forest fit on the data typically taken 300-700 seconds depending on tree depth and number of estimators. The full grid search completes in under 3 hours with CPU. (9022 seconds total)

XGBoost (XG)

XGBoost uses a gradient-boosted tree ensemble with a logistic loss for binary classification. Regularization is controlled by `lambda` (L2), `alpha` (L1), and tree-specific parameters.

1. Class imbalance and class weights

To address label imbalance, we use inverse-frequency class weights on the training set:

1. Let n_c be the number of training examples in class c, $N = \sum_c n_c$ be the total number of training examples, and K be the number of classes ((K=2) here).
2. The base weight for class (c) is:

$$(w_c^{(0)} = \frac{N}{K, n_c})$$

3. We then raise weights to a tunable exponent alpha:

$$w_c = \left(w_c^{(0)}\right)^{\alpha}$$

- alpha = 1: standard inverse-frequency reweighting
- alpha < 1: softer reweighting
- alpha > 1: more aggressive up-weighting of rare classes

2. Cross-validation setup and number of experiments

Likewise to all the models described previously, we used the rolling cross validation for the training window (years 2015 to 2018) where year 2019 is held out for test.

- Fold 1: Train on year 2015, Validate on 2016
- Fold 2: Train on years 2015 and 2016, Validate on 2017
- Fold 3: Train on years 2015 to 2017, Validate on 2018

The following is the paramter grid search space:

Hyperparameter	Values
max_depth	{8, 10}
n_estimators	{150, 175}
learning_rate	{0.1}
subsample	{0.8}
colsample_bytree	{1.0}
reg_lambda	{2.0}
reg_alpha	{1.0}

Due to the use of 3 folds and a grid search space, there are a total of $2 \times 2 \times 1 \times 1 \times 1 \times 1 \times 1 = 4$ total configurations

With cross validation, 4 configurations × 3 folds = 12 CV experiments.

3. Runtime

With a GPU, a single XGBoost fit on the data typically takes 100-200 seconds depending on tree depth and number of estimators. The full grid search completes in about 1 hour with a GPU. (Around 2920 seconds) Otherwise, with a CPU, the xgboost cross validation execution will not be able complete satisfactorilly due to shared compute resources and time.

Results and Discussions

Logistic Regression

1. Experiment table

Table 1 summarizes the 12 experiments we ran, varying alpha, regParam, and elasticNetParam. Each row shows the mean metrics over the three rolling CV folds (2015–2018) for the *delayed* class.

Exp ID	alpha	regParam	elasticNetParam	mean_precision_delayed	mean_recall_delayed	mean_f1_delayed
G01	0.5	0.0	0.0	0.3603	0.1676	0.2271
G02	0.5	0.0	0.5	0.3603	0.1676	0.2271
G03	0.5	0.001	0.0	0.3617	0.1656	0.2254
G04	0.5	0.001	0.5	0.3678	0.1527	0.2140
G05	0.5	0.01	0.0	0.3665	0.1534	0.2145
G06	0.5	0.01	0.5	0.3772	0.1051	0.1639
G07	1.0	0.0	0.0	0.2648	0.6209	0.3706

Exp ID	alpha	regParam	elasticNetParam	mean_precision_delayed	mean_recall_delayed	mean_f1_delayed
G08	1.0	0.0	0.5	0.2648	0.6209	0.3706
G09	1.0	0.001	0.0	0.2649	0.6213	0.3708
G10	1.0	0.001	0.5	0.2646	0.6258	0.3714
G11	1.0	0.01	0.0	0.2652	0.6220	0.3712
G12	1.0	0.01	0.5	0.2557	0.6461	0.3662

*Table 1 - Grid search results for **delayed vs not_delayed** (rolling CV on 2015-2018) via logistic regression*

Table 2 is the runtime table for the 12 experiments.

alpha	regParam	elasticNetParam	Mean train time (s)	Mean eval time (s)
0.5	0.0	0.0	75.9	0.6
0.5	0.0	0.5	74.9	0.5
0.5	0.001	0.0	75.1	0.4
0.5	0.001	0.5	82.8	0.4
0.5	0.01	0.0	76.5	0.5
0.5	0.01	0.5	94.6	0.5
1.0	0.0	0.0	74.3	0.5
1.0	0.0	0.5	75.2	0.5

alpha	regParam	elasticNetParam	Mean train time (s)	Mean eval time (s)
1.0	0.001	0.0	77.3	0.5
1.0	0.001	0.5	85.9	0.5
1.0	0.01	0.0	77.1	0.5
1.0	0.01	0.5	85.2	0.5

*Table 2 - Runtime results for **delayed vs not_delayed** (rolling CV on 2015-2018) via logistic regression*

Using the class-weighting scheme from the imbalance section, the **best configuration by recall** on the *delayed* class is:

- `alpha = 1.0`, `regParam = 0.01`, `elasticNetParam = 0.5` (G12)
- Approximate class weights on the training window (2015–2018):
  - `delayed`: 2.82
  - `not_delayed`: 0.61

Key observations from the training / validation experiments:

- **Moderate reweighting (`alpha = 0.5`)**
  Mean recall(delayed) stays low (~0.15–0.17), with precision ≈ 0.36–0.38 and F1 ≈ 0.22.
  The model is still fairly conservative in calling flights “delayed”.

- **Stronger reweighting (`alpha = 1.0`)**
  Recall(delayed) jumps to about **0.62–0.65**, while precision drops to ≈ 0.26.
  F1 peaks around **0.37** (G10–G11). This region (`alpha = 1.0`, small `regParam`) gives the best balance between catching delays and controlling false positives on the CV years.

Overall, increasing `alpha` clearly shifts the model toward predicting more delays, as intended.

2. Blind test performance on 2019

For the final model, we retrained logistic regression on the full 2015–2018 window using the **high-recall configuration G12**:

- `alpha = 1.0`, `regParam = 0.01`, `elasticNetParam = 0.5`

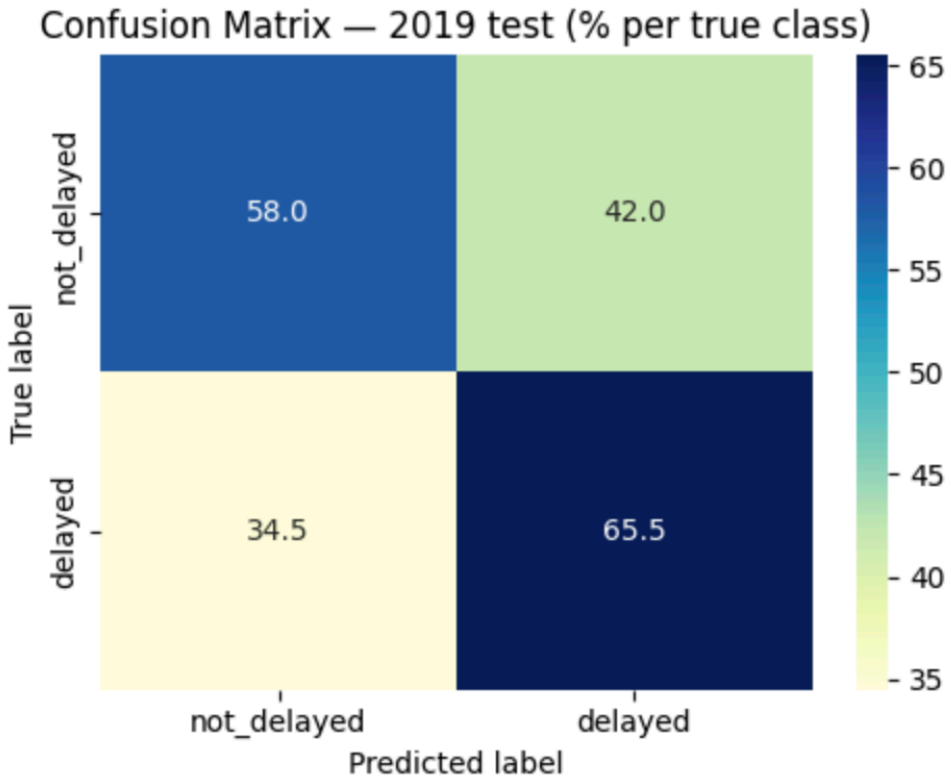
On the held-out **2019** test set, we obtain:

- **Precision (delayed) ≈ 0.2595**
- **Recall (delayed) ≈ 0.6551**
- **F1 (delayed) ≈ 0.3718**

These are very close to the CV averages for G12 (P ≈ 0.256, R ≈ 0.646, F1 ≈ 0.366), which suggests the model generalizes reasonably well across years.

The confusion matrix below is normalized **per true class** (rows sum to 100%):

- For **true not_delayed** flights:
  - 58.0% correctly predicted as not_delayed
  - 42.0% incorrectly flagged as delayed (false alarms)
- For **true delayed** flights:
  - 65.5% correctly predicted as delayed (true positives)
  - 34.5% missed and predicted as not_delayed (false negatives)



These are the final metric results with the optimal parameters.

Split	F1	Precision	Recall
Train	0.3683	0.2554	0.6606
Cross Validation	0.3662	0.2557	0.6461
Test	0.3718	0.2595	0.6551

3. Discussion

Impact of class weighting

- Without strong weighting, LR behaves like a standard accuracy-oriented classifier and mostly learns the majority pattern (*not_delayed*), leading to poor recall on delays.
- Increasing `alpha` to 1.0 meaningfully increases recall for *delayed* flights; the model becomes more "alert" to possible delays, which is desirable if missing a delay is costly.

Trade-off between recall and precision

- The chosen high-recall configuration (G12) is suitable if the system is used as a **warning tool**: better to over-alert than to miss delays.
- A slightly different setting (e.g. G10 with slightly higher F1 but a bit lower recall) would be more appropriate if operations are sensitive to **too many false positives**.

Generalization and limitations

- The 2019 results mirror the recall/precision trade-off seen in CV, so the class-weighted LR appears stable over time.
- However, LR still uses a **linear decision boundary**; complex nonlinear interactions between schedule, route, and weather are likely under-modeled.
  This motivates richer models (e.g., tree ensembles or neural nets) or additional engineered interaction features in future work.

MLP

1. Experiment table

Table 1 summarizes the 8 MLP configurations we evaluated in our time-based grid search. We varied:

- Network depth / width: hidden layers ∈ {[32], [64, 32]}

- Downsampling ratio: maj_to_min_ratio ∈ {1.0, 1.2}
- Decision threshold on P(delay): {0.3, 0.5}

The table reports the mean validation metrics for the delayed class averaged across the three folds.

Exp ID	hidden_layers	maj_to_min_ratio	threshold	mean_precision_delayed	mean_recall_delayed	mean_f1_delayed
M01	[32]	1.0	0.3	0.2535	0.6546	0.3654
M02	[32]	1.0	0.5	0.2544	0.6516	0.3660
M03	[32]	1.2	0.3	0.2716	0.5489	0.3634
M04	[32]	1.2	0.5	0.2721	0.5458	0.3632
M05	[64, 32]	1.0	0.3	0.2506	0.6599	0.3631
M06	[64, 32]	1.0	0.5	0.2498	0.6648	0.3631
M07	[64, 32]	1.2	0.3	0.2691	0.5502	0.3613
M08	[64, 32]	1.2	0.5	0.2716	0.5453	0.3625

Table 1 - Grid-search results for the MLP (mean validation metrics over 3 time-based folds).

Table 2 is the runtime table for all the folds.

Hidden layers	maj_to_min_ratio	Fold	Feature fit time (s)	Train time (s)	Eval time (s)
[32]	1.0	[2015]→2016	14.5	23.6	0.147
[32]	1.0	[2015,2016]→2017	12.9	63.6	0.135

Hidden layers	maj_to_min_ratio	Fold	Feature fit time (s)	Train time (s)	Eval time (s)
[32]	1.0	[2015,2016,2017]→2018	15.9	262.8	0.142
[64, 32]	1.0	[2015]→2016	9.7	41.8	0.125
[64, 32]	1.0	[2015,2016]→2017	11.7	124.6	0.116
[64, 32]	1.0	[2015,2016,2017]→2018	31.2	136.4	0.123
[32]	1.2	[2015]→2016	10.7	26.3	0.141
[32]	1.2	[2015,2016]→2017	12.0	38.7	0.142
[32]	1.2	[2015,2016,2017]→2018	36.1	62.0	0.139
[64, 32]	1.2	[2015]→2016	11.1	43.3	0.141
[64, 32]	1.2	[2015,2016]→2017	14.2	92.1	0.136
[64, 32]	1.2	[2015,2016,2017]→2018	15.4	160.5	0.140

Table 2 - Runtime per fold for MLP models

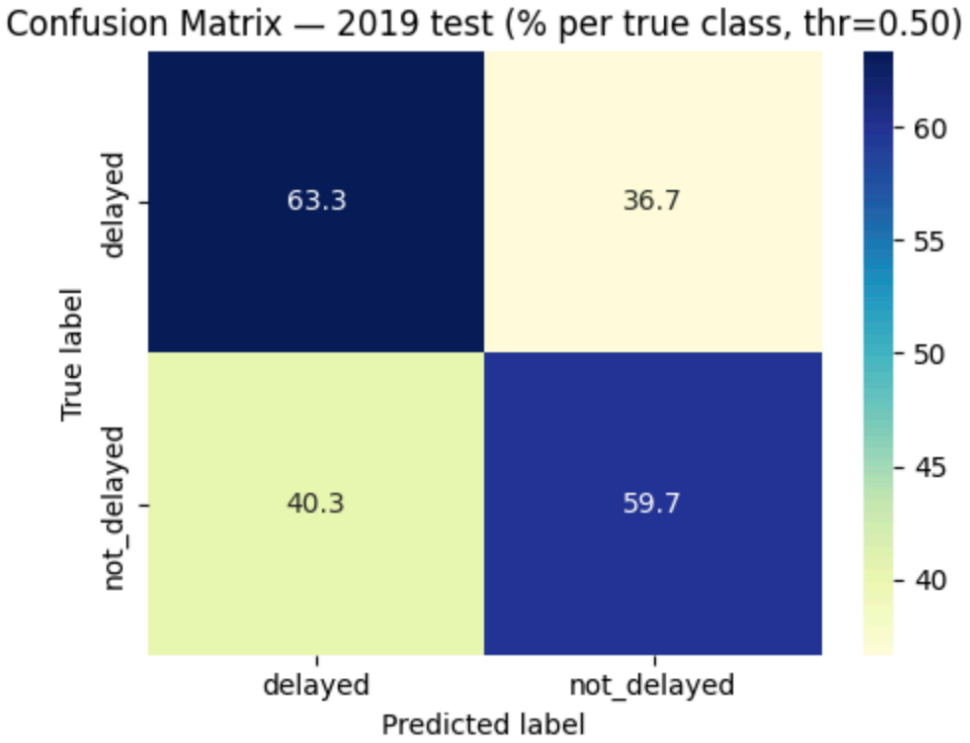
Key observations:

- All configurations land in a similar band of performance (valid F1 ≈ 0.36), suggesting that deeper architectures or slightly different downsampling ratios do not radically change results with this feature set.
- Lower thresholds (0.3) slightly increase recall at the cost of precision; thresholds of 0.5 offer a marginally more balanced trade-off.
- We selected M02 (hidden_layers = [32], maj_to_min_ratio = 1.0, threshold = 0.5) as the final MLP configuration: it has the best results of both Recall and F1 on the delayed class.

2. Blind test performance on 2019

We report metrics for the delayed class:

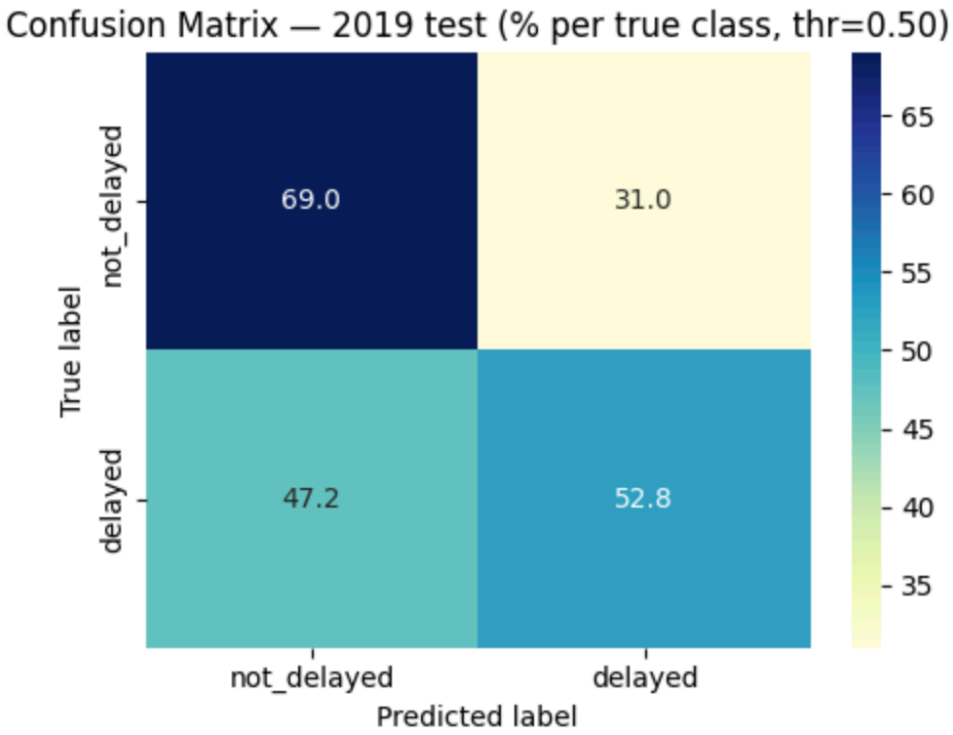
Split	F1	Precision	Recall
Train	0.6375	0.6172	0.6593
Cross Validation	0.3699	0.2577	0.6557
Test	0.3696	0.2610	0.6331



The cross-validation and test scores are nearly identical, which indicates that the MLP generalizes consistently across years and is not severely overfitting the training period. However, the precision and F1 for the delayed class remain relatively low.So operationally, the MLP behaves like a “delay-sensitive” classifier: it prioritizes catching delays at the cost of warning on many flights that ultimately depart on time.

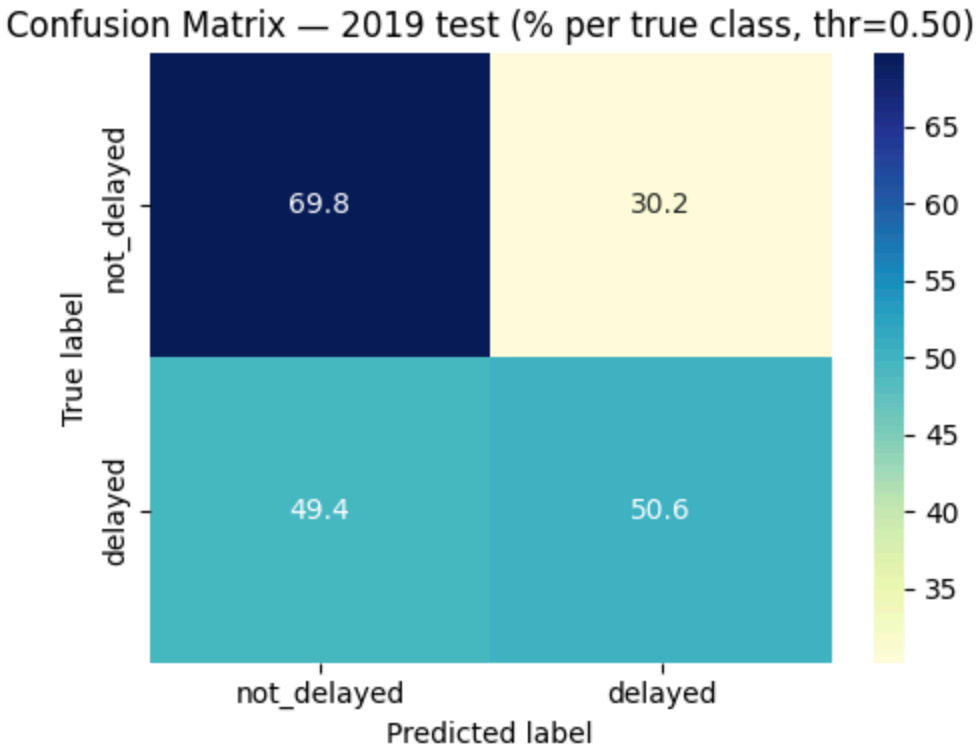
We also compared different network architectures by depth on the held-out 2019 test set, keeping the same downsampling ratio (1.2) and threshold (0.5):

- 2 hidden layers: [64, 32]
  - TRAIN (2015–2018, balanced): P=0.5945, R=0.5574, F1=0.5753
  - TEST (2019 blind): P=0.2765, R=0.5279, F1=0.3629



- 3 hidden layers: [128, 64, 32]

- TRAIN (2015–2018, balanced): P=0.5939, R=0.5296, F1=0.5599
- TEST (2019 blind): P=0.2731, R=0.5061, F1=0.3547



3. Discussion

Several factors likely limit the MLP's performance on this tabular problem:

1. Model class vs. tabular data

   MLPs are not always the best choice for high-dimensional, mixed-type tabular data. In our experiments, tree-based models (Random Forest, XGBoost) achieved substantially higher F1 and precision with the same features. We also observed that adding more hidden layers (e.g., moving from [64, 32] layer to [128, 64, 32]) did not improve and

often slightly worsened validation/test F1, likely because the deeper network overfits the downsampled training data and struggles to learn stable patterns from relatively noisy tabular features.

2. Imbalance handling
   Downsampling improves recall on the delayed class but discards many majority examples and can hurt calibration. Alternative approaches like class-weighted loss or focal loss might give a better recall-precision balance.
3. Limited hyperparameter search
   Due to compute constraints we tuned only a small set of architectures and training parameters. A wider search over depth, width, learning rate, and regularization (e.g., dropout, L2) might yield a stronger neural model, though at higher cost. If we had more time and compute, the next steps for the MLP would be:
   - Re-design the categorical handling using hashing or embeddings.
   - Calibrate the predicted probabilities and set route-specific thresholds so that high-traffic, delay-prone routes can use different operating points than very reliable ones.

Overall, the MLP provides a useful contrast: it demonstrates that even with extensive feature engineering and hyperparameter tuning, a standard neural network struggles to match tree-based methods on this flight-delay prediction task, especially in terms of precision and F1 for the delayed class.

Random Forest

1. Experiment table (Random Forest)

The following table summarizes the grid search experiments for Random Forest with varying key hyperparameters.

Exp ID	numTrees	maxDepth	maxBins	featureSubsetStrategy	train_val_split	precision	recall	f1	time in seconds
RF01	100	8	32	auto	≤2015 → 2016	0.7859	0.6496	0.6910	295

Exp ID	numTrees	maxDepth	maxBins	featureSubsetStrategy	train_val_split	precision	recall	f1	time in seconds
RF02	100	8	32	auto	≤2016 → 2017	0.7802	0.6293	0.9714	492
RF03	100	8	32	auto	≤2017 → 2018	0.7772	0.6221	0.6642	672
RF04	100	8	64	auto	≤2015 → 2016	0.7858	0.6443	0.6867	325
RF05	100	8	64	auto	≤2016 → 2017	0.7804	0.6269	0.6693	521
RF06	100	8	64	auto	≤2017 → 2018	0.7773	0.6189	0.6614	711
RF07	125	8	32	auto	≤2015 → 2016	0.7862	0.6429	0.6856	369
RF08	125	8	32	auto	≤2016 → 2017	0.7802	0.6248	0.6675	615
RF09	125	8	32	auto	≤2017 → 2018	0.7768	0.6204	0.6627	751
RF10	125	8	64	auto	≤2015 → 2016	0.7864	0.6425	0.6852	401
RF11	125	8	64	auto	≤2016 → 2017	0.7801	0.6237	0.6666	658
RF12	125	8	64	auto	≤2017 → 2018	0.7771	0.6183	0.6609	848

Table – Grid search results for delayed vs not_delayed via random forest
As previously described, the total time was 9022 seconds of the cell execution. The experiments per the above parameters were due to compute constraints considerations (i.e. out of memory errors) and their respective values were based on plausible parameters experienced in phase 2.

The final optimal parameters for random forest is (All metrics were averaged per parameters):

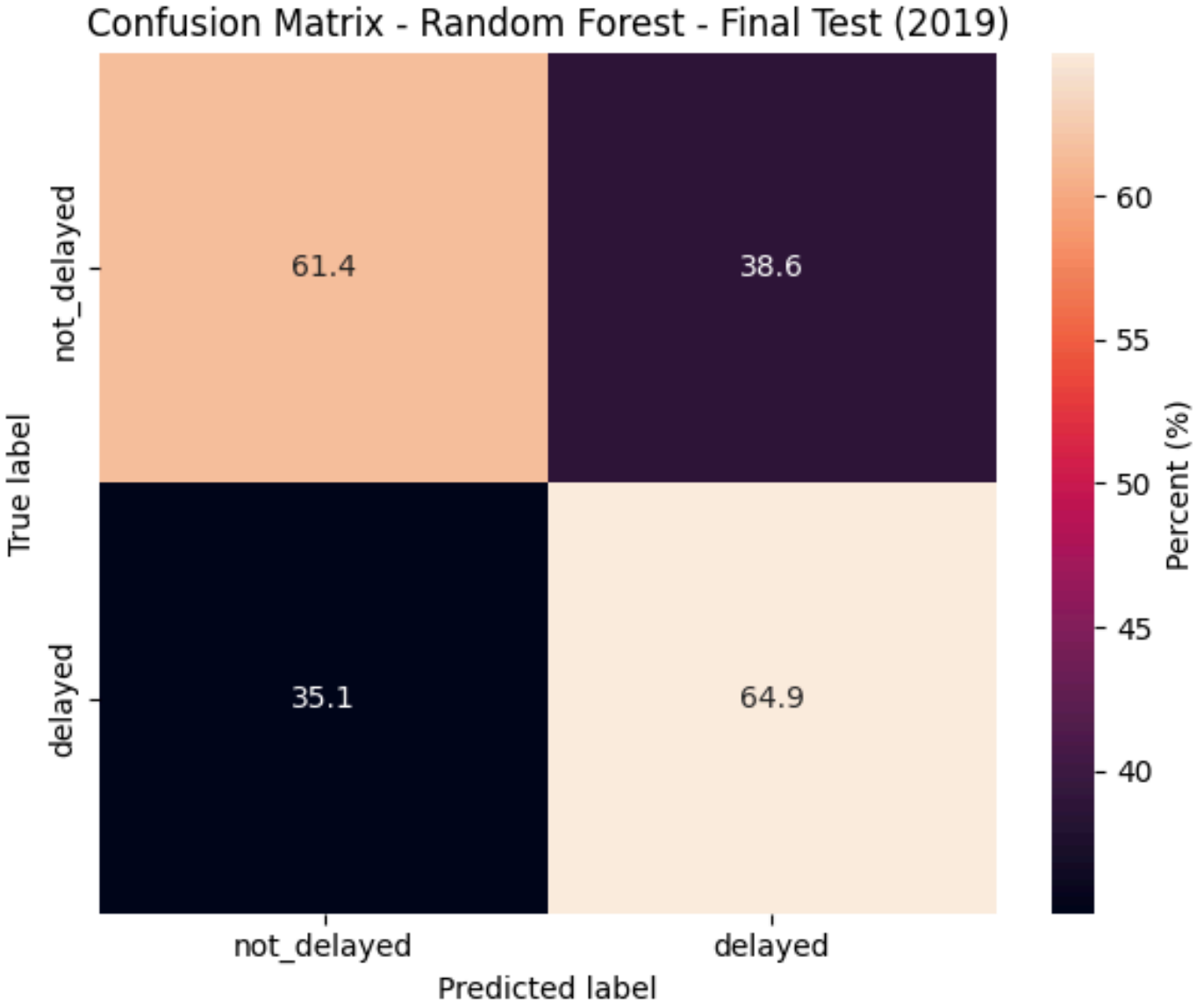
numTrees	maxDepth	maxBins	featureSubsetStrategy	Precision	Recall	F1
100	8	32	auto	0.7810	0.6336	0.6755

2. Train/Validation/Test performance with the optimal parameters

These are the final metric results with the optimal parameters.

Split	Precision	Recall	F1
Train	0.7809	0.6229	0.6660
Cross Validation	0.7810	0.6337	0.6755
Test	0.7704	0.6209	0.6618

Confusion Matrix in percentage per true class:



3. Discussion

Impact of class weighting and tree depth on detecting delayed flights

The class weighting preprocessed onto the model shows more likely results of the prediction. Otherwise, the random forest model would be biased to predict not_delayed as the result from the class imbalance. With this weighting, the combination of running the grid search and using cross validation to select the best hyperparameters had reached to a depth of 8 at 100 trees offered the most favorable balance to maintain the generalizability for a delayed flight prediction.

Trade-off between recall and precision

Our random forest model shows the trade off at predicing the flight delays. While the F1-score suggests a relative balance between precision and recall, our primary objective is to optimize recall and not precision. Our goal in selecting the best model parameters was based on the best cross validation results that had the highest recall (given our constraints of time and compute). Our random forest model, regardless, does exhibit that it is predicting the precision metric well probably due to the model being conservative at predicting "safely" delayed flights.

Generalization and limitations

Our hyperparameter tuned random forest model clearly shows that it generalizes the predictions well while maintaining yearly variation such as seasonal travel impacts and weather conditions in tact.

That is, we are surprised that the model retained favorable results within 5% (0.05) of each split per metric. If the difference between the train/cross validatiuon against the test data was greater, we could conclude that the model was overfit as random forest decision tree models are always prone to this scenario. However, our parameters and data did not fortunately allow the model to reach an overfit state. Regardless, the performance - with the class balanced data - can be due to how well non-linear models are able to make predictions to the features.

However, as much as we want to expand the parameter search space, memory compute constraints per our cluster (mentioned in the pipelines section above) restricted our ability to expand the search space for the hyperparameters yet the grid search surprisingly determined the best hyperparamter model for random forest that is reasonably generalizable.

XGBoost

1. Experiment table (XGBoost)

The following table (table 3) summarizes the grid search experiments for XGBoost with varying key hyperparameters.

Exp ID	max_depth	n_estimators	learning_rate	subsample	colsample_bytree	reg_lambda	reg_alpha	split	precision	re
X01	8	150	0.1	0.8	1.0	2.0	1.0	≤2015 → 2016	0.7896	0.6
X02	8	150	0.1	0.8	1.0	2.0	1.0	≤2016 → 2017	0.7847	0.6
X03	8	150	0.1	0.8	1.0	2.0	1.0	≤2017 → 2018	0.7812	0.6
X04	8	175	0.1	0.8	1.0	2.0	1.0	≤2015 → 2016	0.7896	0.6

Exp ID	max_depth	n_estimators	learning_rate	subsample	colsample_bytree	reg_lambda	reg_alpha	split	precision	re
X05	8	175	0.1	0.8	1.0	2.0	1.0	≤2016 → 2017	0.7849	0.6
X06	8	175	0.1	0.8	1.0	2.0	1.0	≤2017 → 2018	0.7813	0.6
X07	10	150	0.1	0.8	1.0	2.0	1.0	≤2015 → 2016	0.7886	0.7
X08	10	150	0.1	0.8	1.0	2.0	1.0	≤2016 → 2017	0.7841	0.6
X09	10	150	0.1	0.8	1.0	2.0	1.0	≤2017 → 2018	0.7809	0.6
X10	10	175	0.1	0.8	1.0	2.0	1.0	≤2015 → 2016	0.7883	0.7
X11	10	175	0.1	0.8	1.0	2.0	1.0	≤2016 → 2017	0.7840	0.6

Exp ID	max_depth	n_estimators	learning_rate	subsample	colsample_bytree	reg_lambda	reg_alpha	split	precision	re
X12	10	175	0.1	0.8	1.0	2.0	1.0	≤2017 → 2018	0.7807	0.6

Table – Grid search results for delayed vs not_delayed via xgboost
As previously described, the total time for the execution was 2920 seconds in the "cell" execution. Times reported are at the exact execution of each experiment.

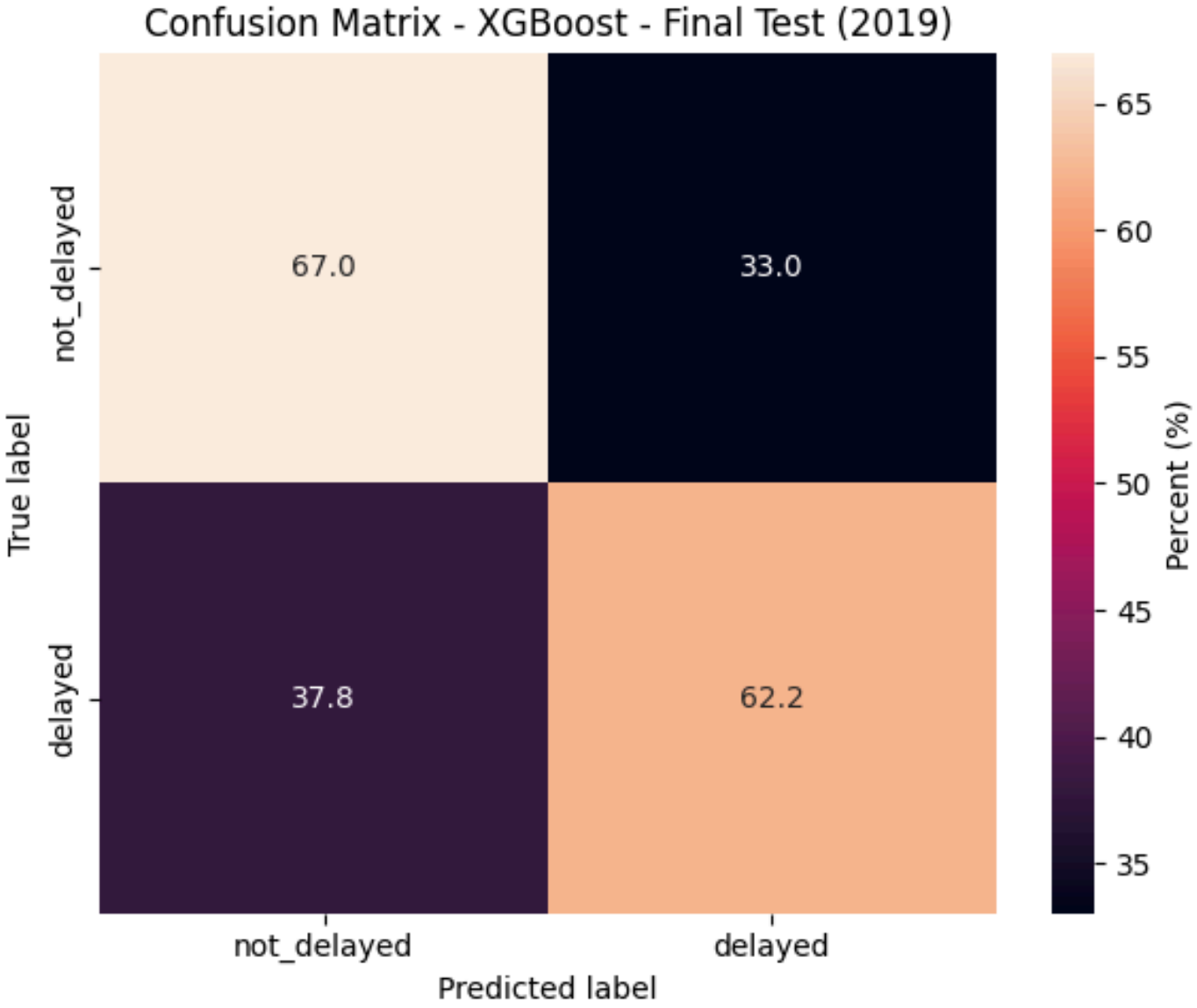
The final optimal parameters for XGBoost is (All metrics were averaged per parameters):

max_depth	n_estimators	learning_rate	subsample	colsample_bytree	lambda	alpha	precision	recall	f1
10	175	0.1	0.8	1.0	2.0	1.0	0.7843	0.6826	0.7153

2. Train/Validation/Test performance with the optimal parameters

Split	Precision	Recall	F1
Train	0.7974	0.6792	0.7140
Cross Validation	0.7843	0.6826	0.7153
Test	0.7756	0.6606	0.6957

Confusion Matrix in percentage per true class:



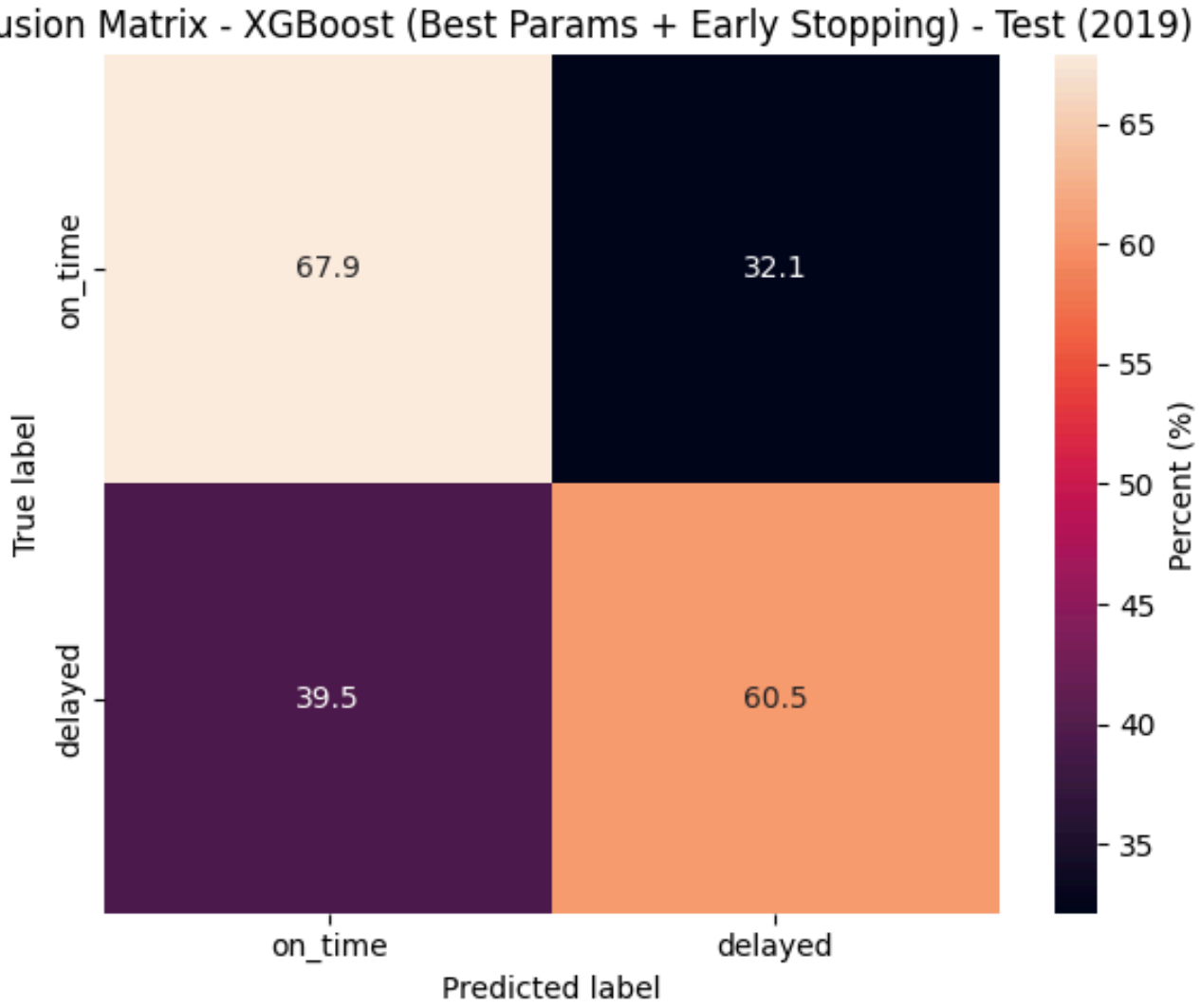
3. Additional Experiment

We have tried to use early stopping with XGBoost. With a early stopping round of 10 and a modification to the columns to signify the appropriate validation with the same best model parameters, we have arrived at the following results. Note that the training data for the model was on the 2015-2017 data and the validation was directly at the 2018 data.

Split	Precision	Recall	F1
Train (2015-2017)	0.7981	0.6800	0.7151
Validation (2018)	0.7811	0.6626	0.6984
Test (2019)	0.7732	0.6649	0.6989

However, we did not define a fixed seed so we believe that these results are purely as an exploratory analysis.

Confusion Matrix in percentage per true class:



If we are to run the cross validation experiments with early stopping again, we should arrive at the same best parameters that we elected to define as the results above are relatively similar. Therefore, early stopping would probably not help unless we have a larger grid (due time and out of memory compute constraints) of parameters to analyze.

4. Discussion

Impact of class weighting and boosting on detecting delayed flights

Just like Logistic Regression and random forest, the class weighting preprocessed onto the model is also essesntial to produce more likely results of the predictio, otherwise, XGBoost would also be biased to predict the majority class not_delayed class. With this class weighting, however, the improvement does come with a some false positives that we can see in the confusion matrix, creating the trade off between recall and precision. Per a depth of 10 instead of the shallower 8, the model is relatively deep to capture some trends in the data yet without fully being underfit to the occasional extremes or out of ordinary patterns of the dataset.

Trade-off between recall and precision

Our XGBoost model shows the trade off at predicing the flight delays. While the F1-score suggests a relative balance between precision and recall, our primary objective is to optimize recall. Although the current results show high precision metrics across each set of dataframes, our goal in our cross validation experiments was at selecting the best hyperparameters that favored recall. Per the hyperparameters as it currently stands, our XGBoost model delivers well at capturng the true delays despite the possible data drifts in time series data per the test year 2019.

Generalization and limitations

Based on the results presented above for the model, XGBoost generalizes well with just an edge against the random forest, MLP, and well above our logistic regression models. This can be due to how we have regularization as part of the input parameters for XGBoost to minimize the scenario of overfitting the model. In fact, the delta between the metrics are about 5% (0.05) of one another for the model. This signifies that the model is relatively stable at making its flight delay classification predictions.

However, the class imbalance persists. Despite the class weighting, the flight outcomes are not truly predicted well per the dataset due to noise and seasonality similar to the discussions per our other models. (Like routes, unforeseeen events, etc.)

Forthermore, even though we have tried to use more parameters in our experiments, we have indeed reached out of memory issues even with a GPU. Thus, we reduced the number of parameter configurations to the ones that are promising for our results per the time/compute constraints.

Gap Analysis

Final Results Table (Phase 2)

Model	Split	F1	Precision	Recall
	Train (Q1 to 3)	0.3963	0.2863	0.6434
Logistic Regression	Validation (Cross)	0.3830	0.2927	0.5751
	Test (Q4)	0.2986	0.2663	0.3397
	Train (Q1 to 3)	0.6686	0.7670	0.6291
Random Forest	Validation (Cross)	0.7098	0.7194	0.7014

Model	Split	F1	Precision	Recall
	Test (Q4)	0.7353	0.7768	0.7074
	Train (Q1 to 3)	0.6903	0.7836	0.6537
XGBoost	Validation (Cross)	0.6738	0.7617	0.6396
	Test (Q4)	0.7576	0.7769	0.7422

Final Results Table (Phase 3)

Model	Split	F1	Precision	Recall
	Train	0.3683	0.2554	0.6606
Logistic Regression	Validation (Cross)	0.3662	0.2557	0.6461
	Test	0.3718	0.2595	0.6551
	Train	0.6375	0.6172	0.6593
Multilayer Perceptron	Validation (Cross)	0.3699	0.2577	0.6557
	Test	0.3696	0.2610	0.6331
	Train	0.6660	0.7809	0.6229
Random Forest	Validation (Cross)	0.6755	0.7810	0.6337
	Test	0.6618	0.7704	0.6209
	Train	**0.7140**	**0.7974**	**0.6792**

Model	Split	F1	Precision	Recall
XGBoost	Validation (Cross)	**0.7153**	**0.7843**	**0.6826**
	Test	**0.6957**	**0.7756**	**0.6606**

1. Current State

Based on both the experiments from phase 2 and phase 3, XGBoost consistently emerges as our strongest performing model across the train, cross validation, and test data in our metrics. That is, in the quarterly analysis and the yearly analysis, the model maintains robustness across each of the temporal scopes.

As discussed in each of the individual model subsections that we have used above, our overall best model is XGBoost when exposed to the 5-year data. In fact, with class balancing considerations, the model stands on top at a recall percentage of 0.66 and a f1 of 0.69. In addition, the model does not appear to overfit: its performance remains stable across the train, validation, and test splits. Across these evaluations, differences in our main metric (F1) between train, validation, and test remain within about 5 percentage points (0.05). This can be due to the model being robust at capturing non-linear trends while maintaining generalizability well due to the regularization factor that we used to train the model (reg_lambda/alpha).

This consistency of XGBoost indicates that given the right feature engineered data, proper feature selection, and hyperparameter values, the model can have the right conditions to perform extremely well. Additionally, XGBoost's stability relative to the other models is still notable. This outcome is also similar to our second best model which is the random forest.

By contrast, the non-tree models struggle to reach comparable performance on the same feature space. Logistic regression maintains reasonable recall (0.65) thanks to aggressive class weighting, but precision remains low (0.26), yielding F1 scores around 0.37. MLP achieves high F1 on the downsampled training data (0.64), but its validation and

test F1 scores drop back to the 0.37 range, similar to logistic regression. This indicates that, with the current architecture and regularization, the network does not extract substantially more signal from the features than a well-tuned linear model.

2. Target State

Our ideal goal would be to achieve metrics well above 90% (0.9), but we recognize that this is likely unrealistic for this problem. As a more practical target, we instead require that performance metrics (precision, recall, and F1) remain within about 5 percentage points of each other across the validation and test splits. Fortunately, this is the case with the XGBoost model for the validation and test data in phase 3 where the difference between the two are indeed within 5% so we also do not have a case of overfitting.

Another realistic target is to aim for a 70% (0.7) recall score as this would just be the next stepping stone in the model. Unfortunately, all of our models were unable to reach this figure. XGBoost is the only model that comes close to this target, with recall in the mid-0.60s on both the Phase 3 validation folds and the held-out test set.

Once again, random forest also fulfills the criteria as our follow up second best model at being close yet missing the target of 70%. Moreover, even though logistic regression and the multilayer perceptron models do not achieve competitive absolute performance, their validation and test metrics are still within a similar 8% (0.08) band. This indicates that, despite their lower F1 and recall, their generalization behavior across time is reasonably consistent.

3. Gap State

The recall metric is still below our aspirational target of 90%, but it is reasonably close to the more realistic 70% goal. With test recall at 66%, there remains a gap of about 4 percentage points that might be recoverable through additional hyperparameter tuning and more targeted feature engineering.

Since this gap does not appear to be driven by overfitting or model instability, it suggests that further improvements will likely depend on better engineered features and refined class-balancing strategies to expose clearer patterns for the model to learn from.

In addition to our other models, the non-tree-based models underperform relative to the ensembles. Logistic regression is limited by its effectively linear decision boundary, which cannot fully capture complex interactions between time, route, and weather features. The MLP, in its current configuration, appears sensitive to class imbalance and may not be sufficiently tuned (e.g., in terms of architecture, regularization, and optimization) to extract more signal than the simpler tree-based methods. These limitations contribute to the performance gap between the best pipeline (XGBoost) and the rest.

4. Closing the Gap State

Based on the experimental models analysis and results above, a natural next step for closing the performance gap is to explore an ensemble-based approach that combines our strongest models. For example, a stacked ensemble that integrates predictions from the neural network, logistic regression, and random forest, and then uses XGBoost as a meta-learner, could potentially improve recall beyond our current ~66% and move closer toward the 70% target while maintaining a reasonable balance with F1.

Although we did not implement this stacked ensemble due to time and compute constraints, we view it as a plausible direction for future work rather than a guaranteed path to 90% recall. With additional feature engineering and hyperparameter tuning under this framework, we would expect incremental gains on unseen data, ideally without sacrificing the model's generalization performance across train, validation, and test splits.

Conclusion

Our project focused on predicting flight delays prior to departure using large-scale historical flight, weather, and operational data, addressing a critical problem that impacts airport operations. Our hypothesis was that a machine learning pipeline enriched with custom temporal, weather, operational, reputation, and network-based features could accurately predict flight departure delays. Among the evaluated models, an optimized XGBoost classifier achieved the best performance, reaching **66% recall and 77% precision for delayed flights on a held-out 2019 test set**, demonstrating the value of combining traditional operational features with network-aware and reputation-based metrics. These results show meaningful improvements over baseline models and provide insights for proactive resource planning. Future work includes extending the dataset to include **2020–2021 flights**, exploring stacked **ensemble models**, utilizing **feature selection based on final model** and perform deeper hyper-parameter tuning to ensure gains on unseen data.

Appendix

Section	Notebook Link
Data Load, Data Analysis and Data Quality Fixes	https://dbc-fae72cab-cf59.cloud.databricks.com/editor/notebooks/1792055957781638?o=4021782157704243#command/1792055957781639 (https://dbc-fae72cab-cf59.cloud.databricks.com/editor/notebooks/1792055957781638?o=4021782157704243#command/1792055957781639)
EDA - Correlation Analysis, Time Series Decomposition	https://dbc-fae72cab-cf59.cloud.databricks.com/editor/notebooks/3056786081358273?o=4021782157704243#command/3056786081358274 (https://dbc-fae72cab-cf59.cloud.databricks.com/editor/notebooks/3056786081358273?o=4021782157704243#command/3056786081358274)

Section	Notebook Link
Feature Engineering	https://dbc-fae72cab-cf59.cloud.databricks.com/editor/notebooks/4336423679577408?o=4021782157704243 (https://dbc-fae72cab-cf59.cloud.databricks.com/editor/notebooks/4336423679577408?o=4021782157704243)
Feature Selection	https://dbc-fae72cab-cf59.cloud.databricks.com/editor/notebooks/4093368127198590?o=4021782157704243#command/8173636742488469 (https://dbc-fae72cab-cf59.cloud.databricks.com/editor/notebooks/4093368127198590?o=4021782157704243#command/8173636742488469)
Logistic Regression and MLP	https://dbc-fae72cab-cf59.cloud.databricks.com/editor/notebooks/4093368127198629?o=4021782157704243 (https://dbc-fae72cab-cf59.cloud.databricks.com/editor/notebooks/4093368127198629?o=4021782157704243)
XGBoost and Random Forest - Ensemble/Gradient Boosted Trees	https://dbc-fae72cab-cf59.cloud.databricks.com/editor/notebooks/3915201981341170?o=4021782157704243#command/8516020384390782 (https://dbc-fae72cab-cf59.cloud.databricks.com/editor/notebooks/3915201981341170?o=4021782157704243#command/8516020384390782)