Predicting Flight Delays through Machine Learning Classifiers at Scale Final Presentation

W261 Fall 2022 Section 5 Group 4:
Nathan Chiu, Dominic Lim, Raul Merino, Javier Rondon

Agenda | We will cover the following topics in order

Executive Summary

Airlines should implement an **ensemble model** to better predict flight delays for resource allocation/customer service purposes

Problem	20% of flights are delayed by more than 15 minutes, costing tens of billions of dollars
Data	Flights, weather, weather station, and airport codes from government agencies
Strategy	**Select model for F2 performance and minimal run time**

Models

BASELINE	Logistic Regression for Simplicity
DECISION TREES	Efficiency and Performance at Scale
RANDOM FORESTS	Collected Decision Trees
MULTILAYER PERCEPTRON	Neural Network
ENSEMBLE	Combining Model Predictions

Modeling Steps

- Feature Engineering
- Model Pipelines
- Hyperparameter Tuning
- Run Experiments & Gap Analysis

Project Outcomes

- **F2 Score of** .558, nearly 5x the baseline
- **Sequential runtime** of 33 minutes
- Most important feature of **previous flight delay**
- **Main pipeline** ending with an ensemble model

Project Metrics

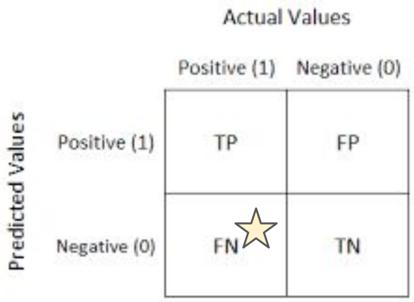
We used F2 Score as our primary metric of success because we want to focus more on recall to minimize false negatives

What is F2 Score?

$$F_2 = \frac{TP}{TP + 0.2FP + 0.8FN}$$

F2 is the weighted average of precision and recall and we selected the beta value of 2 to focus on recall, which refers to proportion of true positives that are correctly identified

Why use F2 Score?



We are using F2 for more focus on recall to **minimize false negatives**. False negatives occur when the flight is predicted to not be delayed, but ends up being delayed, which is the main driver in the costs in flight delays

With F1, the denominator would contain 0.5 FP and 0.5 FN

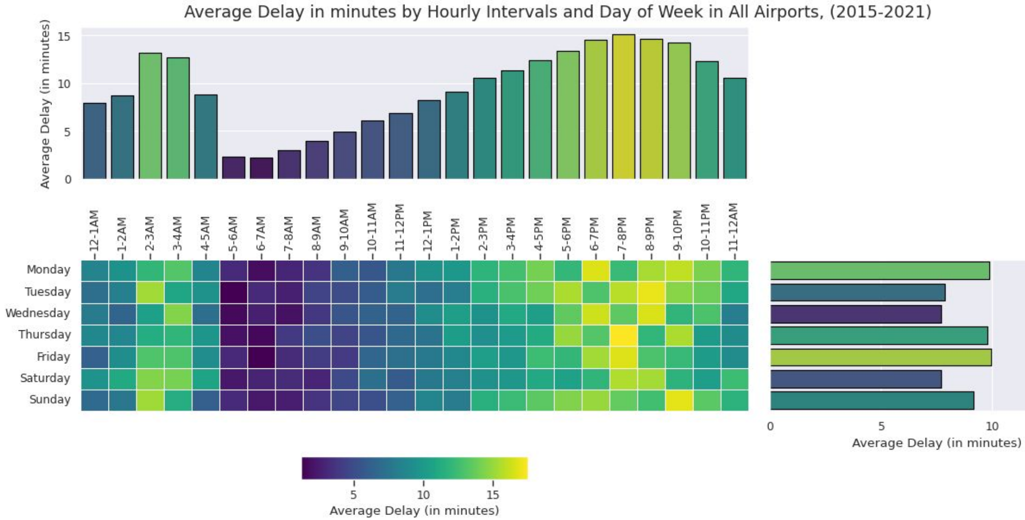
Image Source: Wikimedia Commons, Towards Data Science

EDA

We conducted an exploratory data analysis of the flight and weather datasets, focusing on computing % of missing values per feature and understanding the features' distribution, scale, and range of values

Flights Dataset EDA

YEAR	Extreme Weather (%)	NAS - Weather (%)	Late Aircraft - Weather (%)	Total Weather (%)
2015	5.37	10.37	6.87	22.61
2016	4.70	11.26	6.85	22.81
2017	4.59	11.94	7.09	23.62
2018	6.13	12.10	7.92	26.14
2019	5.95	12.52	7.97	26.43
2020	7.86	7.23	4.97	20.06
2021	7.41	6.41	5.30	19.13



- Our EDA suggests that **~20% of total delays** (in minutes) are attributable to weather conditions.
- The increasing average minutes of delay as time goes on suggests **network effects at play** i.e. delays accumulating as the day progresses and planes with delayed flights having their subsequent flights delayed

Data Lineage

We created a data lineage tracing how we turned the raw data files into our joined files and added in features that we plugged into our model pipelines

1. Joining Raw Data Files Into Joined Dataset

Join

- Find closest weather stations to airports
- Pull in airport codes and timezones
- Drop duplicates and extra weather records
- Standardize timestamps to UTC
- Create composite key (airport, timestamp) and join

Result

Final joined dataset

2. Pulling in Features Created

Join

Join in pagerank features

Join

Join in delay state features

3. Split for Cross-Validation, Train, & Test

Split

Split into parquet files for blocked time series cross-validation

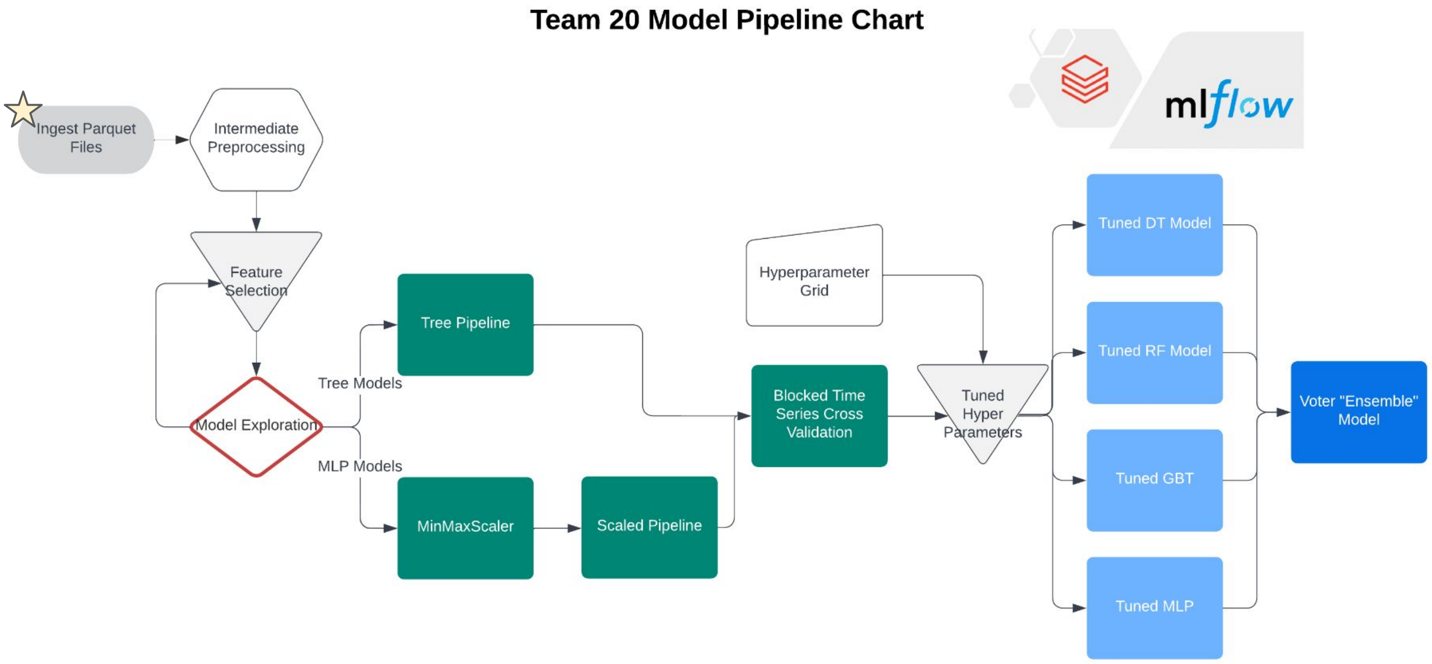
Result

Build final train and test datasets

See Appendix for Data Lineage Code Snippets

Models Pipeline

We focused on generalizing functions involved in the pipeline to make it easier to adjust parameters and run a multitude of experiments



* The modeling pipeline begins with ingesting the joined parquet files that were split by FL_DATE for our Blocked Time Series Cross Validation

Feature Engineering

We created new features to boost the predictive power of our models

Feature Name	Description
Previous Flight Delay	Airlines have a finite number of aircrafts, so each aircraft has a route that it follows every day, going from airport to airport often involving back to back scheduled flights. An earlier delay may affect subsequent flights for the same aircraft
Pagerank Features	PageRank describes an airport's importance and influence, which can describe how delays are spread throughout a network of airports.
Delay States	The delay state represents the network's delay patterns at a point in time
Weather Features	The categorical features indicate the presence of weather related to flight delays such as thunderstorms, snow, fog and ice
Average Airport Delay	We created a feature for the percentage of flights that are delayed in a given time window
Airport Capacity	The ratio of actual flights that depart over scheduled flights out of an airport

Feature Engineering EDA

We conducted an exploratory data analysis of the newly engineered features, focusing on understanding the features' distribution, scale, and range of values

Conceptual Visualization

What and why use Previous Flight Delay?

- **What:** We built the previous delay by tracking the tail number of the plane and **tagging whether or not the plane's previous flight was delayed or not**
  - E.g. Plane ABC is flying from Canada to Eastern Europe, but the previous flight from Australia to Canada was delayed would be tagged as being previously delay i.e. has a value of 1

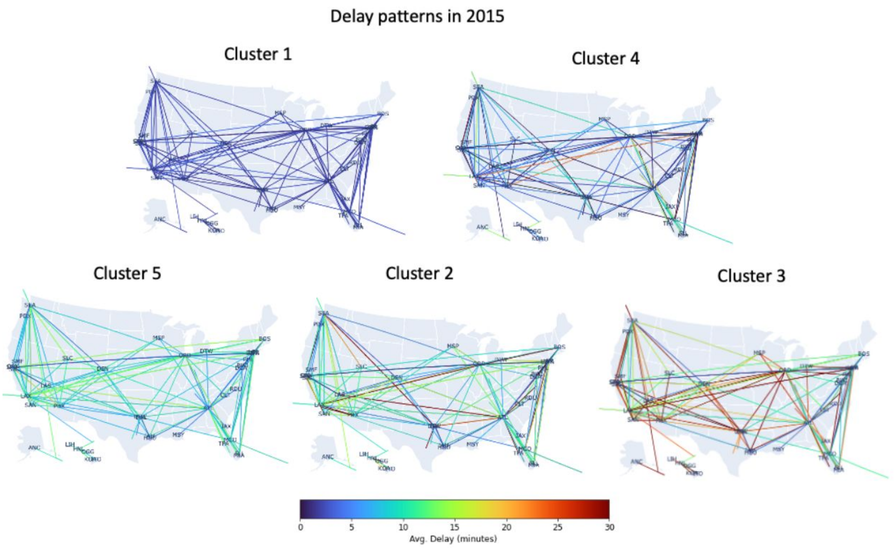
- **Why:** After an initial analysis, we saw that this had a relatively **high correlation (> 0.3)** with the current's flight delay
  - When conducting our initial EDA we thought that past flights would bear an impact on current flights so we decided to build this feature to test this hypothesis

Image Source: Adobe

Feature Engineering EDA

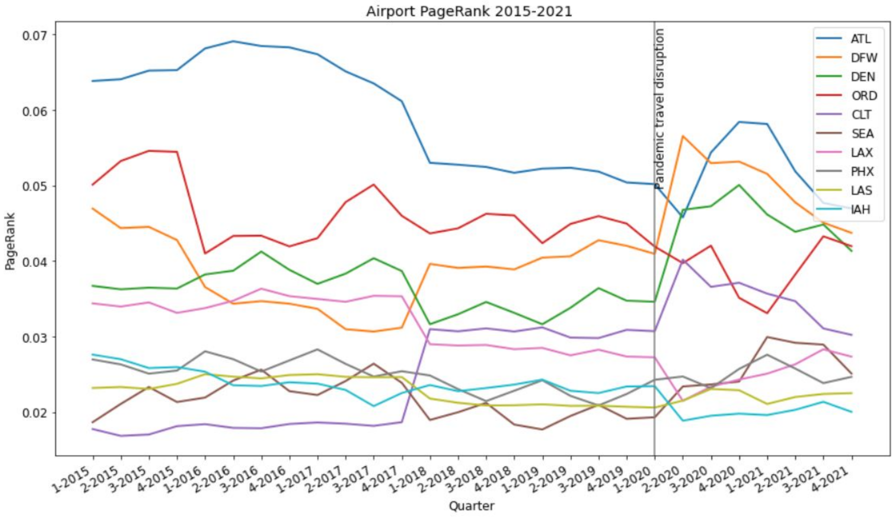
We conducted an exploratory data analysis of the newly engineered features, focusing on understanding the features' distribution, scale, and range of values

Delay State



- In the delay state cluster with the most delays stem from flights that involve **DFW, ORD and LAX**

Airport PageRank

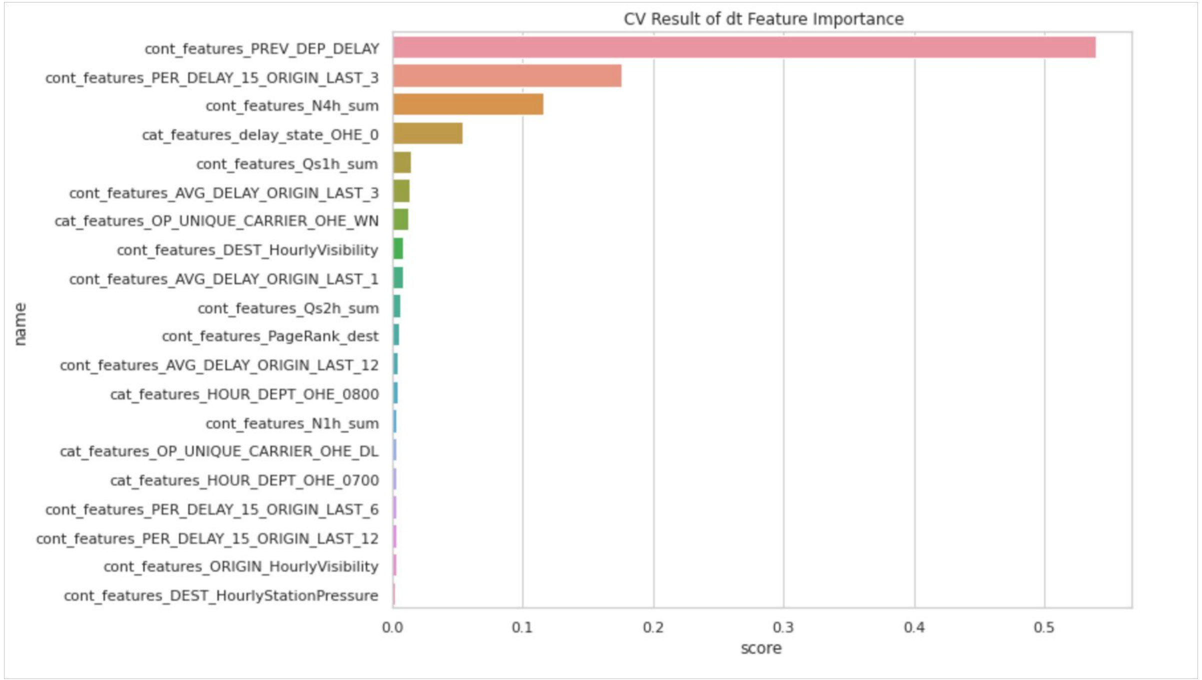


- PageRank shows that the most important airports are **ATL and DFW** and changes in rank across time

Feature Selection

We ran decision tree models with different categories of features and select top three features per category by feature importance

Determining Feature Importance Against CV Data



We selected the three most important features by feature category. Notable high performing features include:

- Previous Delay (PREV_DEP_DELAY)
- PER_DELAY_15_ORIGIN_LAST_3

Feature categories:

- Weather features
- Pagerank features
- Airport capacity
- Delay state
- Previous flight delay
- Other flight features

We utilized **feature importance**, a measure of the decrease in **node purity** weighted by the **probability of reaching that node** to score and rank features above

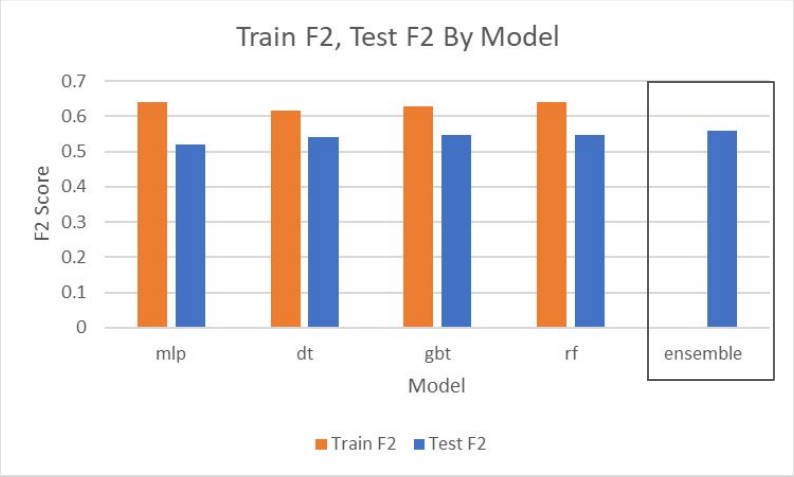
Experimental Results

We compared primary metrics of success like F2 across tuned models and the Ensemble models performed the best

Experiment Results with Hyperparameters

Model	Layers	Max Bins	Max Depth	Max Iterations	Number of Trees	Train F2	Train ROC AUC	Train Precision	Train Recall	Test F2	Test ROC AUC	Test Precision	Test Recall
MLP	[44, 44, 2]	-	-	100	-	0.641	0.748	0.716	0.619	0.519	0.755	0.388	0.589
Decision Tree	-	350	10	-	-	0.617	0.765	0.760	0.589	0.540	0.764	0.411	0.586
Gradient Boosted Tree	-	100	10	6	-	0.630	0.772	0.756	0.605	0.546	0.771	0.405	0.599
Random Forest	-	50	10	-	100	0.642	0.765	0.737	0.622	0.547	0.765	0.384	0.612
Ensemble	-	-	-	-	-	-	-	-	-	0.558	-	0.366	0.643

Model Results Bar Chart



Voting Mechanism

***One Positive Voting**: If one model suggests delay, predict DELAY

Vote by Majority: The majority prediction of DELAY or NO DELAY

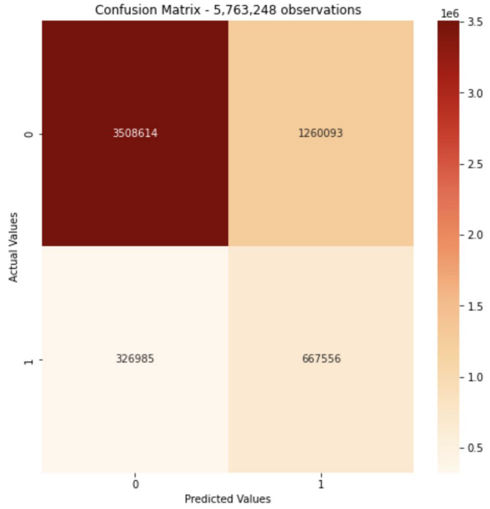
One Negative Voting: If one model suggests no-delay, predict NO DELAY

* Voting mechanism we ultimately selected for the ensemble model

New Model

We also wanted to implement novel approaches including the use of ensemble methods whereby all four models (hyper-parameterized Decision Tree, Random Forest, Gradient Boosted Tree, and Multilayer Perceptron) "vote" on the final prediction

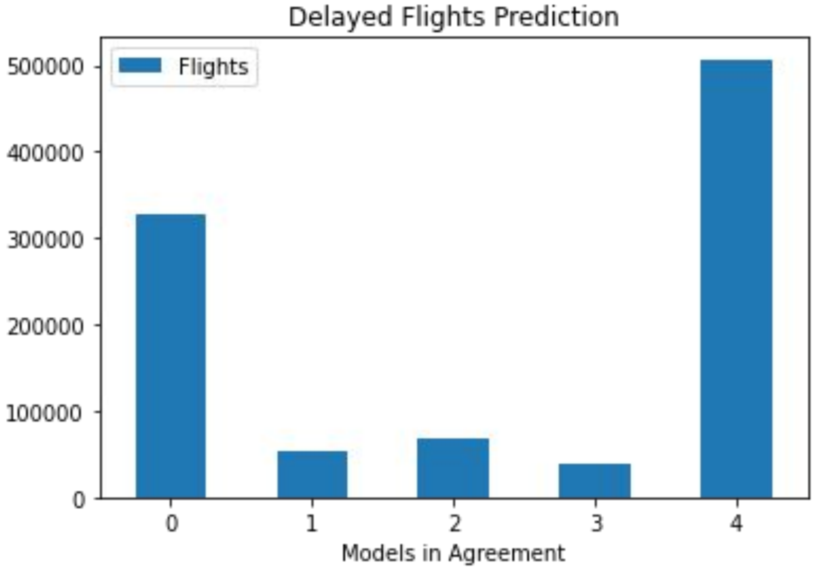
Ensemble (Best Model) Confusion Matrix



Metrics:

- **F2 Score: .558**
- **Precision: .366**
- **Recall is: .643**

Distribution of Flight Predictions by Models in Agreement



How to read chart: For 300K, none of the models correctly predicted them as delayed. For 500K flights, all models correctly predicted them as delayed

Experimental Results

We compared primary metrics of success like F2 across tuned models and the Ensemble models performed the best

Experiment Wins

1. **Model Novelty:** Experimented with ensemble model with varying voting mechanisms and found the positive voting drove the **better recall and F2 metric**
2. **Scientific Approach to Feature Selection**: Used relative importances i.e. key decision boundaries of the decision trees

Experiment Areas of Opportunity / Next Steps

1. **Altering Pipeline Inputs:** Will explore ensemble model on other models besides the four specified earlier. Models with different inputs i.e. only weather features
2. **Pull in Additional External Datasets:** We explored several external datasets like airline ratings and natural disasters that we would like to join in if given more time
3. **Further Investigate Poorly Predicted Flights:** Some flights were not predicted well by any of our models, which was also true for our training data. We can isolate these data points and create models for this data

Gap Analysis

We performed a Gap Analysis comparing our best model results with models from peers and then identified strengths and opportunities for improvement to incorporate in our models

Our Best Model Performance	Comparable Best Model Performance
.558	.750
F2 Score	F2 Score

- One particular method that we did not explore was **weighing our training data differently**. We could have determined if a particular year was more relevant than others for the 2021 data, and weighed that year higher than the rest
- Another feature that we did not focus as much as we would have liked was **departure and arrival times**. We could have turned that "continuous" feature into a categorical ones. It's unlikely the relationship with the delay was linear, instead different times of day could have had a different result (as shown in our initial EDA)
- Since we explored a variety of different models, it doesn't seem that the gap came from not trying a particular model. We could have however done **more extensive hyperparameter tuning and more experimentation**

Conclusion

Airlines should implement an **ensemble model** to better predict flight delays for resource allocation/customer service purposes

Problem	20% of flights are delayed by more than 15 minutes, costing tens of billions of dollars
Data	Flights, weather, weather station, and airport codes from government agencies
Strategy	**Select model for F2 performance and minimal run time**

Models

BASELINE	Logistic Regression for Simplicity
DECISION TREES	Efficiency and Performance at Scale
RANDOM FORESTS	Collected Decision Trees
MULTILAYER PERCEPTRON	Neural Network
ENSEMBLE	Combining Model Predictions

Modeling Steps

- Feature Engineering
- Model Pipelines
- Hyperparameter Tuning
- Run Experiments & Gap Analysis

Project Outcomes

- **F2 Score of** .558, nearly 5x the baseline
- **Sequential runtime** of 33 minutes
- Most important feature of **previous flight delay**
- **Main pipeline** ending with an ensemble model

Main Deck

Appendix

Appendix

Data Lineage

We created a data lineage tracing how we turned the raw data files into our joined files and added in features that we plugged into our model pipelines

Composite Key Code

Create composite key to join weather and flights dataset

```
def create_composite_key(code, timestamp):
  return f'{code}_{timestamp}'

create_composite_key = udf(create_composite_key)

# Create composite key for both datasets
df_weather_icao_needed_tz = df_weather_icao_needed_tz.withColumn("CODE_STATION_TIMESTAMP", create_composite_key('icao', 'HOUR_TIMESTAMP'))
flights_icao_tz = flights_icao_tz.withColumn("CODE_TIMESTAMP", create_composite_key('icao', 'HOUR_WEATHER_TIMESTAMP')) \
  .withColumn("TWO_CODE_TIMESTAMP", create_composite_key('icao', 'TWO_HOUR_WEATHER_TIMESTAMP')) \
  .withColumn("THREE_CODE_TIMESTAMP", create_composite_key('icao', 'THREE_HOUR_WEATHER_TIMESTAMP'))

# Update the datasets
flights_icao_tz.write.mode("overwrite").parquet(f"{blob_url}/flights_with_icao_tz")
df_weather_icao_needed_tz.write.mode("overwrite").parquet(f"{blob_url}/weather_with_icao_tz")

# Load the new datasets
flights_icao_tz = spark.read.parquet(f"{blob_url}/flights_with_icao_tz") # 42430592
df_weather_icao_needed_tz = spark.read.parquet(f"{blob_url}/weather_with_icao_tz") # 1057832
print(flights_icao_tz.count())

# Drop unnecessary columns
columns_to_drop = ['airport_name', 'airport_city', 'airport_country', 'airport_tz', 'year', 'airport_subd', 'country', 'elevation', 'iata', 'airport_lon', 'airport_lat', 'icao']
df_weather_icao_needed_tz = df_weather_icao_needed_tz.drop(*columns_to_drop)
```

Models Pipeline

We focused on generalizing functions involved in the pipeline to make it easier to adjust parameters and run a multitude of experiments

Model Pipeline Reusable Functions

General Functions

```
def read_clean(parquet_string):
    dataset = spark.read.parquet(f"{blob_url}/{parquet_string}")

    # Make sure the target variable is not null

    dataset = dataset.where("DEP_DEL15 is not NULL")

    dataset = dataset.withColumn("PREV_DEP_DELAY", col("PREV_DEP_DELAY").cast('int'))

    dataset= dataset.withColumnRenamed("DEP_DEL15", "label")
    dataset= dataset.withColumn("HOUR_DEPT", substring('DEP_TIME_BLK',1,4))

    for col_name in cont_feat:
        dataset = dataset.withColumn(col_name, col(col_name).cast('float'))

    dataset = dataset.na.drop(subset=["ORIGIN_HourlyStationPressure",
                                      "DEST_HourlyStationPressure",
                                      "ORIGIN_HourlyDryBulbTemperature",
                                      "DEST_HourlyDryBulbTemperature",
                                      "ORIGIN_HourlyVisibility",
                                      "DEST_HourlyVisibility"])\
                        .fillna(0, subset=["ORIGIN_HourlyPrecipitation",
                                           "ORIGIN_HourlyWindDirection",
                                           "ORIGIN_HourlyWindSpeed",
                                           "DEST_HourlyPrecipitation",
                                           "DEST_HourlyWindDirection",
                                           "DEST_HourlyWindSpeed"])

    # PLACEHOLDER TO DEAL WITH NULLS/NAs for QRN Features

    return dataset


def create_parameters(parameter_grid):
    param_names = list(parameter_grid.keys())
    param_values = parameter_grid.values()
    combinations = list(itertools.product(*param_values))
    return (param_names, combinations)

def downsample(train_df):
    n_delays = train_df.filter(f.col("label") == 1).count()
    n_no_delays = train_df.filter(f.col("label") == 0).count()

    total = n_delays + n_no_delays
    keep_percent = n_delays / n_no_delays

    train_delay = train_df.filter(f.col('label') == 1)
    train_non_delay = train_df.filter(f.col('label') == 0).sample(withReplacement=False,fraction=keep_percent,seed=741)
    train_downsampled = train_delay.union(train_non_delay)
    return train_downsampled

def get_metrics(pred_df):
    preds_mc_rdd = pred_df.select(['prediction', 'label']).rdd
    preds_b_rdd = pred_df.select('label','probability').rdd.map(lambda row: (float(row['probability'][1]), float(row['label'])))
    metrics_mc = MulticlassMetrics(preds_mc_rdd)
    metrics_b = BinaryClassificationMetrics(preds_b_rdd)
    F2 = np.round(metrics_mc.fMeasure(label=1.0, beta=2.0), 4)
    au_ROC = metrics_b.areaUnderROC
    return F2, au_ROC
```

Model Pipeline Code

Specify Model Pipelines

```
def tree_pipeline(model):

    """Pipeline for tree models - DT, RF, GBT"""

    assembler_cont = VectorAssembler(inputCols=cont_feat,
                                     outputCol="cont_features")

    #Categorical Features that need to be binned before being used as categorical variables
#     columns_to_bucketize = ['wind_dir_avg_origin', 'wind_dir_avg_dest']
#     splits = [[i for i in range(0,361,20)], [i for i in range(0,361,20)]]
#     bucketizer = Bucketizer(splitsArray=splits, inputCols=columns_to_bucketize, outputCols=[c+"_bucketized" for c in columns_to_bucketize])

    indexer = StringIndexer(inputCols=columns_categorical,
                            outputCols=[c+"_indexed" for c in columns_categorical]).setHandleInvalid("keep")

    ohe = OneHotEncoder(inputCols=[c+"_indexed" for c in columns_categorical],
                        outputCols= [c+"_OHE" for c in columns_categorical]).setHandleInvalid("keep")

    assembler_categ = VectorAssembler(inputCols= [x+"_OHE" for x in columns_categorical],
                                      outputCol="cat_features")

    assembler = VectorAssembler(inputCols= ["cat_features", "cont_features"],
                                outputCol="features")


    pipeline = Pipeline(stages=[assembler_cont, #bucketizer,
                                indexer, ohe, assembler_categ, assembler, model])
    return pipeline
```

Train Model (Decision Tree)

```
dt_best_parameters, dt_best_score = blocktimeSeriesCV(parquet_string='full',
                                                      sampling = 'down',
                                                      param_grid = paramGrid_dt,
                                                      pipeline_fn = scaled_pipeline,
                                                      model_type='dt',
                                                      k=2,
                                                      metric='f2'
)

dtModel, pred_result_dt = validation(full_train_df, full_test_df,
                                     sampling = 'down',
                                     model_type = 'dt',
                                     best_parameters = dt_best_parameters,
                                     pipeline_fn = scaled_pipeline
                                     )
```

Evaluate Model (RF)

```
rfModel, pred_result_rf = validation(full_train_df, full_test_df, sampling = 'down',
                                     model_type = 'rf',
                                     best_parameters = rf_best_parameters,
                                     pipeline_fn = tree_pipeline
                                     )
```

```
▸ (37) Spark Jobs

 CV Result of rf
            Train   Test
ROC AUC     0.715   0.712
F2 Score    0.537   0.469
Recall      0.502   0.500
Precision   0.743   0.375
Accuracy    0.664   0.770
Support for multi-dimensional indexing (e.g. `obj[:, None]`) is deprecated and will be re
```

Models Pipeline | We focused on generalizing functions involved in the pipeline to make it easier to adjust parameters and run a multitude of experiments

Feature Selection

We began by running decision tree models with different categories of features:

- Weather Features
- Airport Capacity (QRN)
- Airport PageRank
- Clustered Delay States
- Previous Flight Feature (based on Tail Number)
- Other Flight Features (Airline Carrier, Seasonality)

Hyperparameter Tuning

Once features were selected, we experimented with combinations of parameters against cross validation data

- Decision Trees / MLP: VectorAssembler, MinMaxScaler
- Decision Tree Loss Function: Gini Impurity

Model Selection

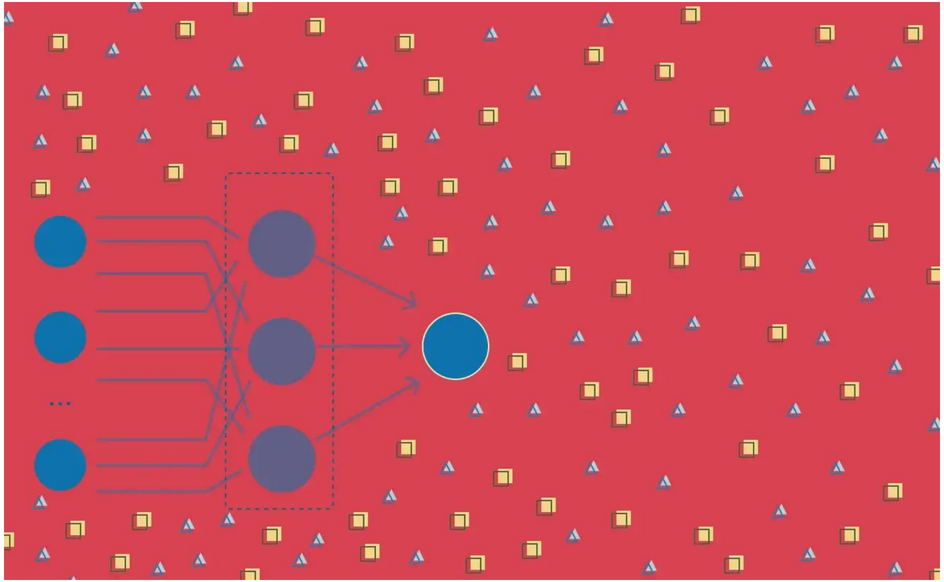
Once we selected the best hyperparameters, we compared the primary metrics like F2 score, precision, and recall across all models:

- Used average F2 score to fit the full train dataset and evaluate the full test dataset

New Model

We implemented a multilayer perceptron (MLP), a fully connected neural network easily integrated into our model pipeline

What is a Multilayer Perceptron and why use it?



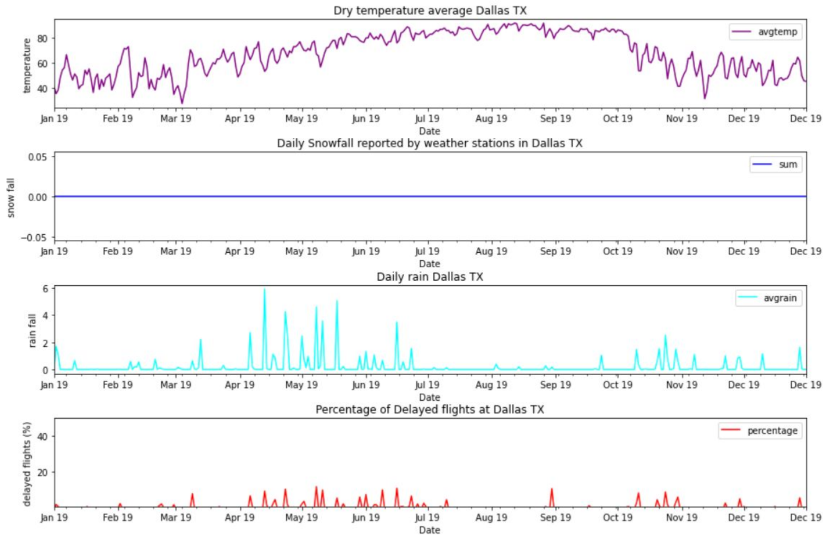
- The MLP utilizes an initial layer of *m* nodes, representing the number of features, with a final output layer of 2
- We had limited time to experiment with hyperparameter tuning, where we primarily changed the number of layers and the number of nodes per layer
- We ultimately found that a MLP architecture of (44 - Sigmoid - 44 - Sigmoid - 2 - Softmax) produced the best F2 score for our selected features.

Image Source: TowardsDataScience, Carolina Bento

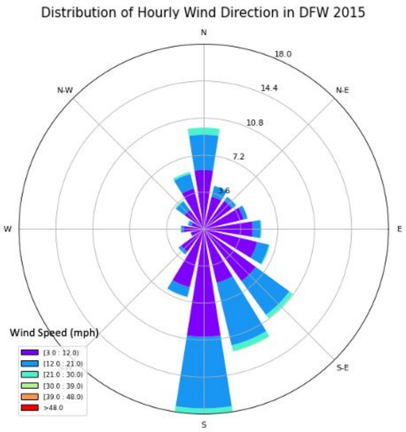
EDA

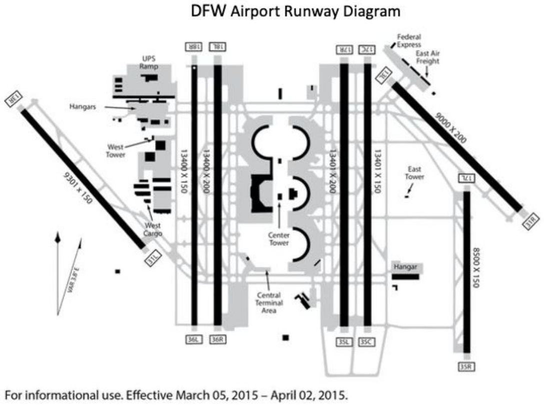
We conducted an exploratory data analysis of the flight and weather datasets, focusing on computing % of missing values per feature and understanding the features' distribution, scale, and range of values

Weather Dataset EDA



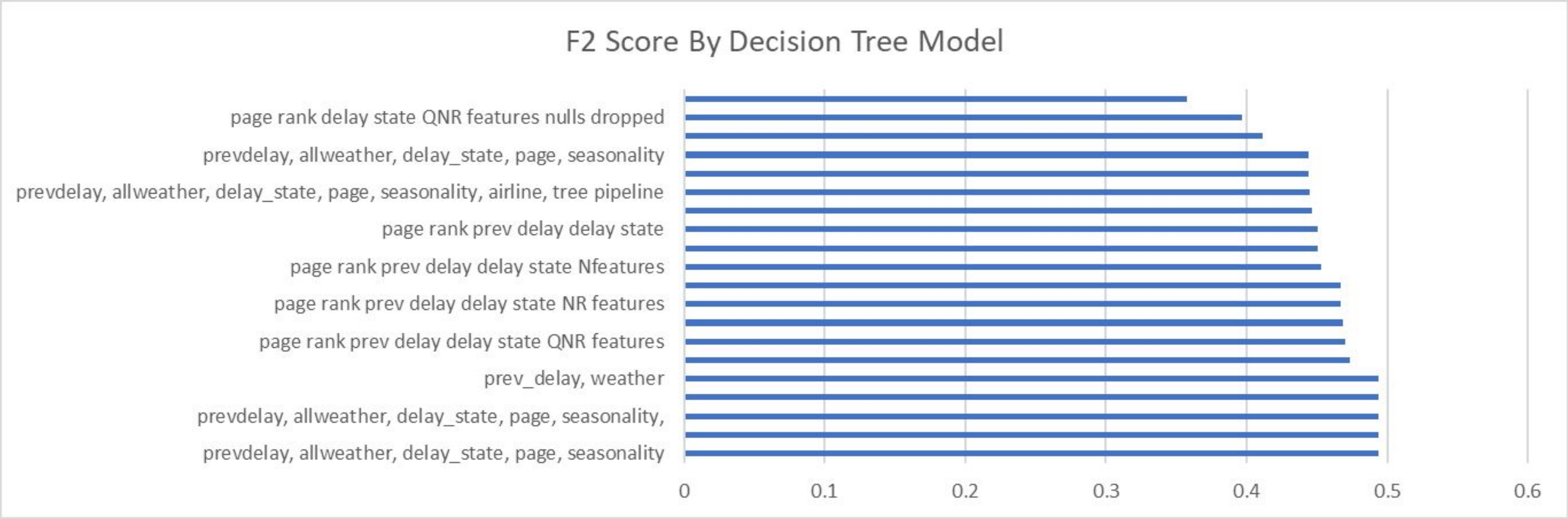
- The distributions of the temperature, snowfall, rain, and percentage of delayed flights suggest that weather feature are highly correlated to certain flight delays



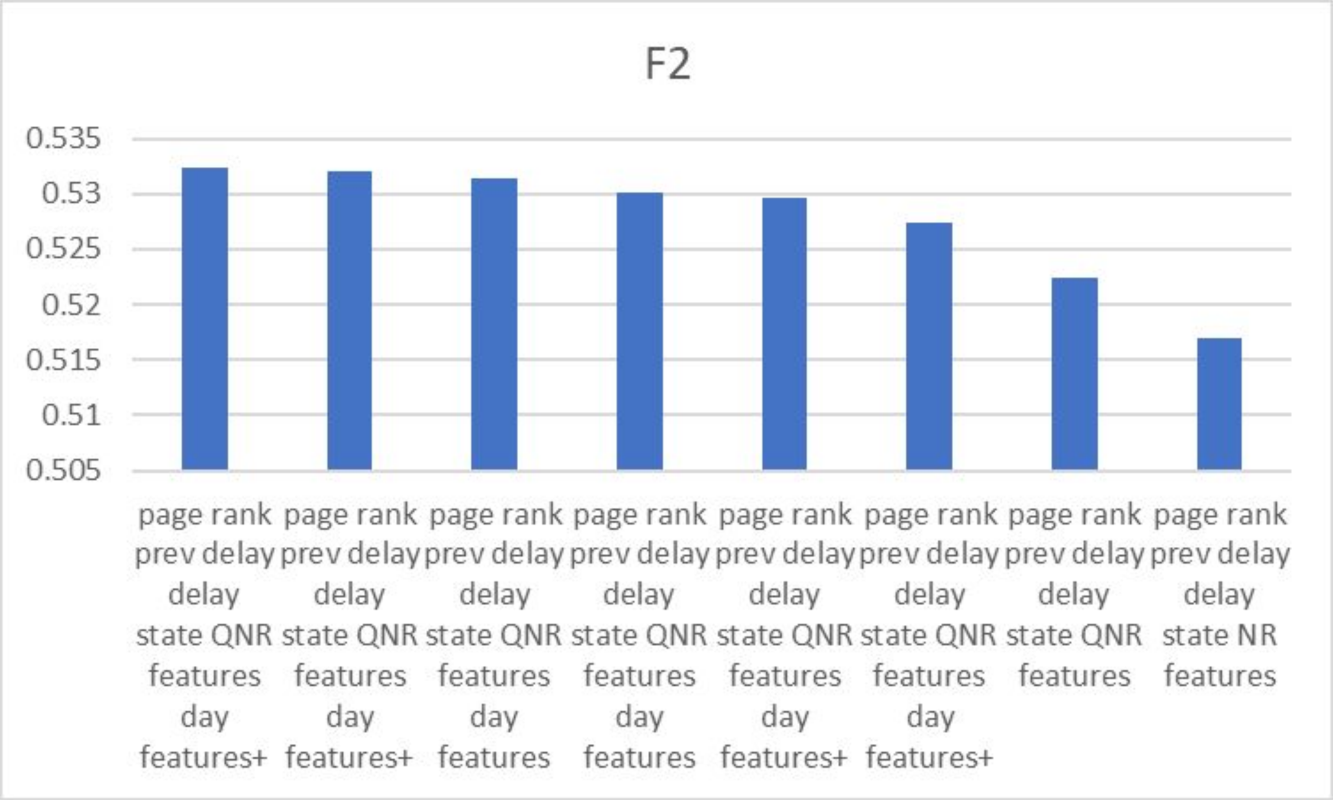


- Features such as wind speed and direction were reviewed to assess their usefulness to explain flight delays.

F2 Score by Decision Tree Model



Random Forest Overall Performance



Graveyard

Q&A

What about normalization?

Don’t need to normalize for decision trees

How are we encoding categorical features?

We will one-hot encoder from mlib Spark library

Old Flow

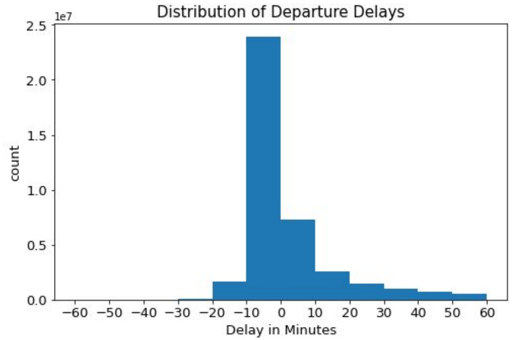
EDA	Join	Feature Engineering	Model Pipelines	Hyperparameter Tuning
Conduct an initial EDA to understand which features are important, how the datasets could be joined and how to deal with missing values.	Join all datasets into one, so each row has the necessary information for the construction of our models.	Create new features using the existing data, such as previous flight delay, airport capacity, previous average flight delay by airport and airline. Experiment and select which features should be included in the models	Create and run multiple model pipelines using the joined dataset and the new features. Measure the performance of each of model by looking at the F-2 score.	Try different combinations of hyperparameters in order to find the one that performs the best across each model type by measuring its performance against the blind test set.

EDA

We conducted an exploratory data analysis of the flight and weather datasets, focusing on computing % of missing values per feature and understanding the features' distribution, scale, and range of values

Flights Dataset Summary Statistics

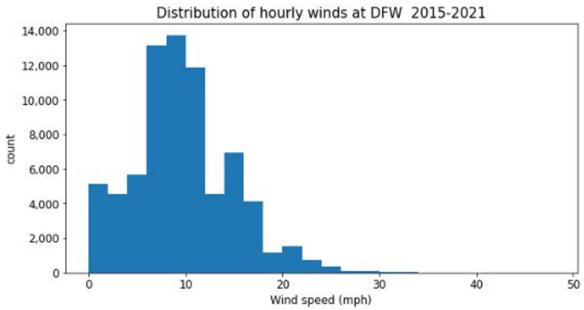
	Missing Values	% of Total Values
DIV4_TOTAL_GTIME	74177433	100.0
DIV5_AIRPORT_ID	74177433	100.0
DIV4_WHEELS_ON	74177433	100.0
DIV4_AIRPORT	74177433	100.0
DIV4_LONGEST_GTIME	74177433	100.0
...	...	...
DEST_STATE_ABR	0	0.0
DEST_CITY_NAME	0	0.0
DEST	0	0.0
DEST_CITY_MARKET_ID	0	0.0
YEAR	0	0.0



- The 6 year flights dataset filtered down to **41 M rows** and **54 features** with **~17% flights delayed**
- Dropped duplicated flight observations
- **Filtered out cancelled flights** where delay in minutes is not recorded. Negative delays reflect early departures
- Interesting features include **Origin/Destination pairs, airline carriers, and seasonality**

Weather Dataset Summary Statistics

	Missing Values	% of Total Values
HourlySkyConditions	5798250	39.827284
HourlyWetBulbTemperature	5707647	39.204946
HourlyStationPressure	5577539	38.311254
HourlyVisibility	5034969	34.584425
HourlyAltimeterSetting	5025875	34.521960
HourlyRelativeHumidity	4847037	33.293549
HourlyDewPointTemperature	4844030	33.272894
HourlyWindDirection	4034929	27.715304
REM	3931159	27.002524
HourlyWindSpeed	3744731	25.721979
HourlyDryBulbTemperature	460241	3.161324



- The 6 years weather dataset filtered down to US weather stations and hourly observations consists of **31M rows** and **379 weather stations**
- We focused on weather features that drive weather related delays such as **precipitation, temperature, pressure, wind, and visibility**
- **Removed high anomalies** and will perform **feature engineering** for changes in pressure (for incoming storms) and winds perpendicular to runways

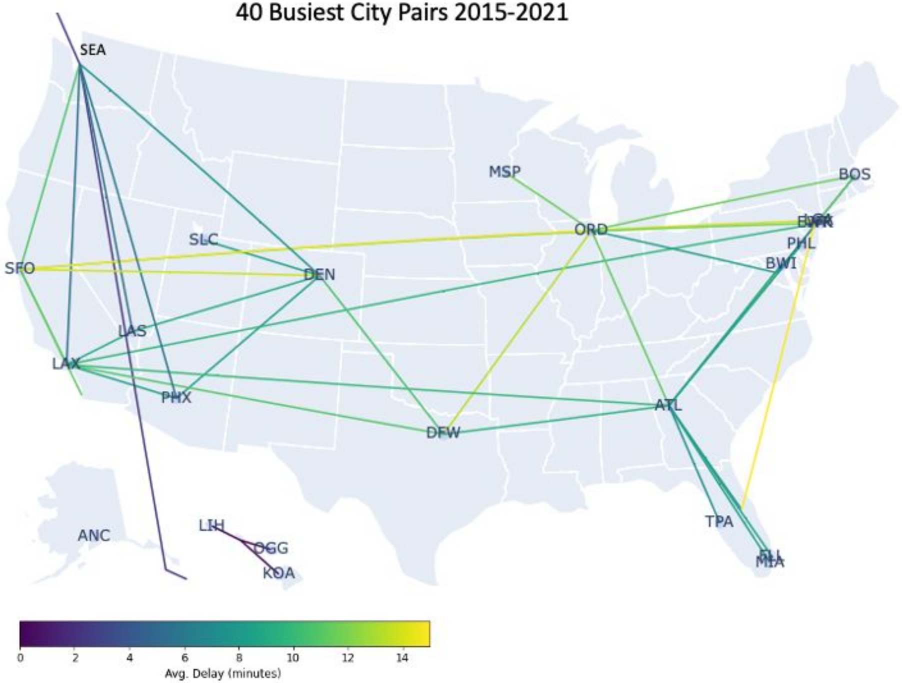
EDA

We conducted an exploratory data analysis of the flight and weather datasets, focusing on computing % of missing values per feature and understanding the features' distribution, scale, and range of values

Flights Dataset EDA

- **The EDA allowed us to visualize the geographical distribution of average delays for origin-destination pairs.**
  - **There are higher average delays in flights from ORD than in ATL.**
  - **HNL and SEA have the lowest average delays in their most frequent flights.**

40 Busiest City Pairs 2015-2021



Paths are colored according to the average departure delay in minutes

*Puerto Rico is in the flights dataset, but not the above visualization give its geographic distance and low impact

Model Features & Engineering

Our models include several other relevant features from the weather dataset and ones that we created

Origin & Destination Weather Features

- We joined in weather features from the weather dataset onto both the origin and destination airports for comprehensiveness
- Key features include:
  - Visibility
  - Ice
  - Fog
  - Snow
  - Thunderstorms
  - Wind Speeds
  - Wind Direction

Tail Number Delay

- We created a delay indicator by airplane that tells us if the airplane was delayed on a prior flight
  - This feature appears useful because it has a 30% correlation with flight delays

Modeling Pipelines

We plan on building additional models to compare performance against our baseline model of assuming all flights are on time.

Initial Model: Logistic Regression

We built our initial model running a logistic regression on several of our categorical variables. Given the simplicity of our model, the results skewed towards flights being on time.

Logistic Regression Results

Precision:	**0.8208**
Recall:	**1.0000**
F-2:	**0.9582**

Loss

Our primary task is running a binary classification task so a cross-entropy loss function makes the most sense.

$$Loss = -\frac{1}{m}\sum_{i=1}^{m}(y_i \cdot log(\hat{y}_i) + (1 - y_i) \cdot log(1 - \hat{y}_i))$$

Next Steps

Given the imbalance of data points, we need to oversample from the majority of flights (non-delayed flights) for the model to perform better.

Joins

We pulled in missing values, dropped duplicates, and created composite keys to perform the full join in 1.5 hours

Flow

Find closest weather stations to airports → Pull in airport codes and timezones → Drop duplicates and extra weather records → Standardize timestamps to UTC → Create composite key (airport, timestamp) and join

The entire join for the complete datasets took about 1.5 hours, split into these activities:

1. Removing duplicate entries in the Flight Dataset and saving to the blob: 8.09m
2. Adding the timezone to all datasets and saving to the blob: 17.48m
3. Creating all timestamps for the flight dataset and saving to the blob: 11.27m
4. Creating all timestamps for the weather dataset and saving to the blob: 2.57m
5. Creating the composite keys and saving both datasets to the blob: 4.19m
6. Joining the flight dataset with the weather dataset for the three hours window and saving to the blob 19.3m
7. Calculcating the time difference between timesamps, using it to find the closest report and saving that to the blobl 29.09m

Total: 91.99m

- The biggest timesaver in the join was the composite key of airport and timestamp down to the hour to simplify the join
- We kept weather observations within a three hour time window prior to prediction time, which covers 99.5% of flights

Feature Engineering

We created new features to boost the predictive power of our models

Feature Name	Description
Weather Features	The categorical features indicate the presence of weather related to flight delays such as thunderstorms, snow, fog and ice.
Previous Flight Delay	Airlines have a finite number of aircrafts, so each aircraft has a route that it follows every day, going from airport to airport. An aircraft being delayed in one airport likely means it will arrive late at its destination, and that may impact that aircraft's next flight's departure time.
Pagerank Features	We were interested in a feature that can indicate the importance or influence of an airport and its role in propagating delays to other flights
Delay States	The delay state represents the network's delay patterns at a point in time.
Airport Capacity	According to the literature, there is a correlation between the number of delayed connecting flights and the total number of flights out of an airport

Data Split and Baseline

We plan on using 2021 flights as the test set and 2015-2020 flights as the training set and will conduct cross-validation through a blocking split. The next steps include creating advanced models and feature engineering

Data Split & Cross-Validation Steps

01 Every flight from 2021 will be assigned to the blind test set, which will only be used for the final evaluation of the model

02 The rest of the flights (2015 to 2020) will be used for the training set. We will conduct cross validation by implementing a blocking split

03 The results of these models will be combined into a weighted average, which will be used to tune the model's hyperparameters

Baseline Models

We built the following baseline models:

- Assume \`NO DELAY\` classification

Baseline Model Results

Precision:	**0.8208**
Recall:	**1.0000**
F2:	**0.9582**