Improving Customers' Flight Experience

Team 14

Carlos Moreno

Elizabeth Khan

Jagan Lakshmipathy

Ziling Huang

Agenda

- Business Case
- Dataset
- EDA & Feature Engineering
- Cross Validation & Evaluation Metrics
- Performance & Scalability Concerns
- Pipeline Errors & Debugging Experience
- Limitations, Challenges, & Future Work

Business Case

Customer

Mid-Size Online Booking Agency

Travel Industry in 2019[1]:

- **2.3 billion** person-trips
- **$2.6 Trillion** Travel Output
- **15.8 million** US Jobs supported
- **1 in 10** U.S. non-farm jobs directly/indirectly relying on travel

Challenge

Cost goes beyond passenger irritation…

Assuming **$47** per hour as the average value of a passenger's time, FAA/Nextor estimated the annual costs of delays (direct cost to airlines and passengers, lost demand, and indirect costs) in 2018 to be **$28 billion.**[2]

** Cost extends to jobs relying on travel industries

Solution

Leveraging the Power of Machine Learning:

To help customers reach their destination by alerting them of potential delays two to three hours before their flight.

References in slide note:

Dataset & EDA

Flights

- 5-year flight data (2015-2019)
- Understand Flights, Airports, Timing, Root Causes
- Data cleaning and identification of missing data
- Data imputation
- Outcome variables
- Investigate data correlations
- Identify important airports and routes
- Time zone synchronization

Weather

- 5-year hourly weather data (2015-2019)
- Understand weather conditions for flight take-off and landing safety
- Parse, Data cleaning for ‘9999’ data
- Multiple records in an hour
- Investigate data correlations
- Identify weather conditions associated with delays

Final Joined Dataset: **31,254,049 records**

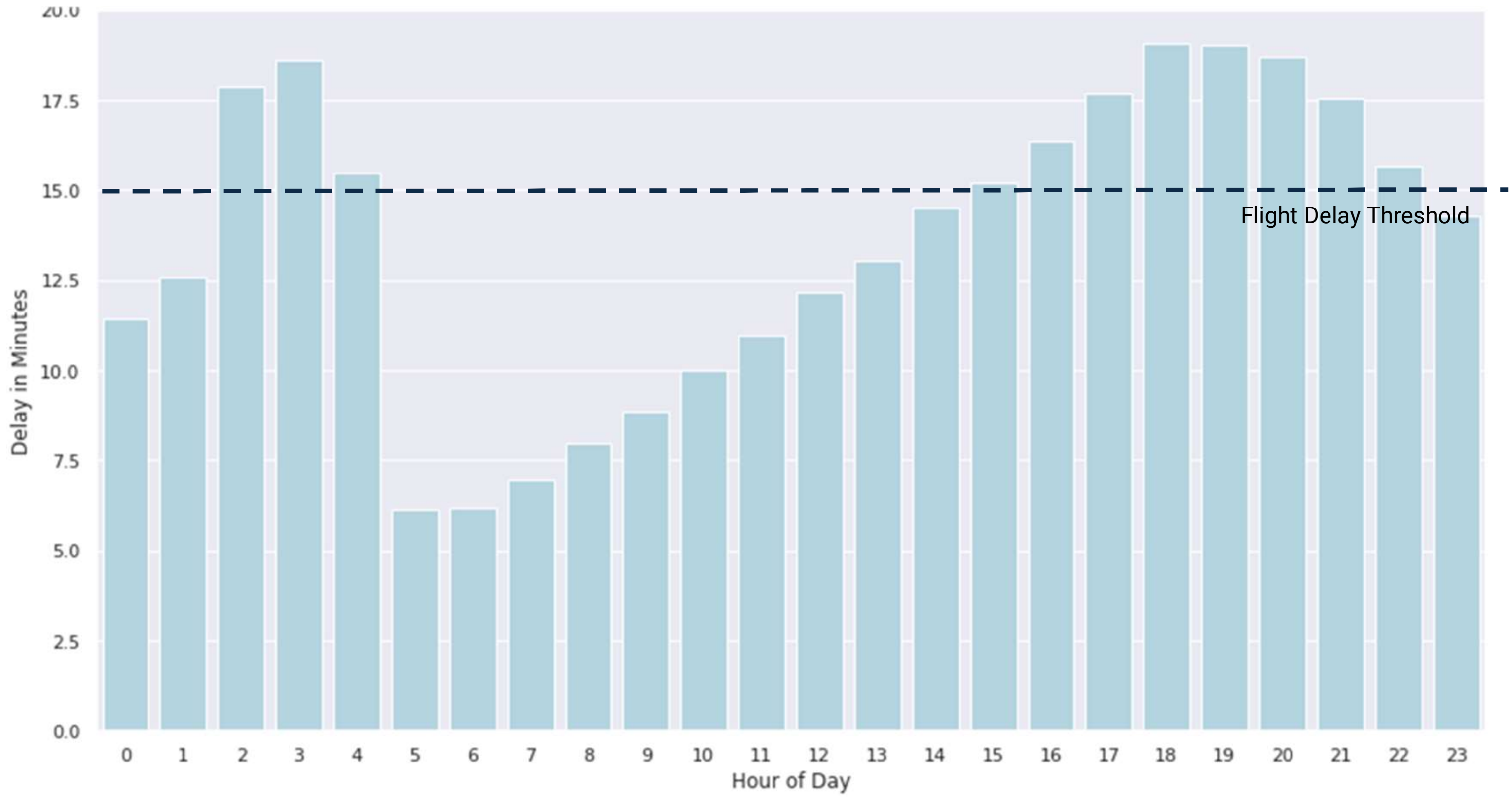
** We excluded cancelled flights and small airports (15235,11719,10590, 11468,16869) from the final joined dataset

*** Weather data only includes FM-15 routine weather reports

Airline Delays by Time of Day

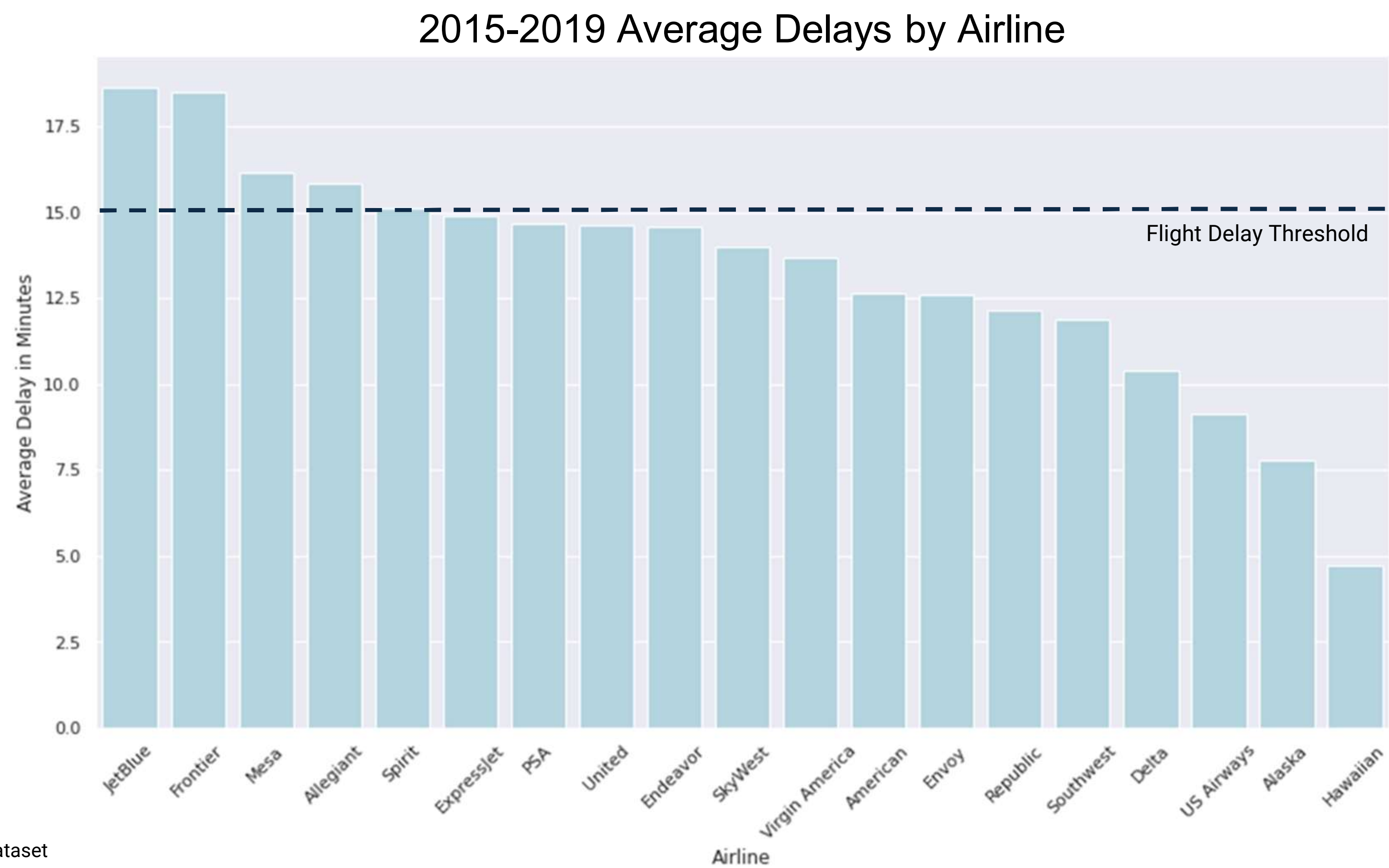
We investigated how departure time hour impacts delays

2015-2019 Delays by Time of Day



Airline Delays

We explored how delays varied across airlines

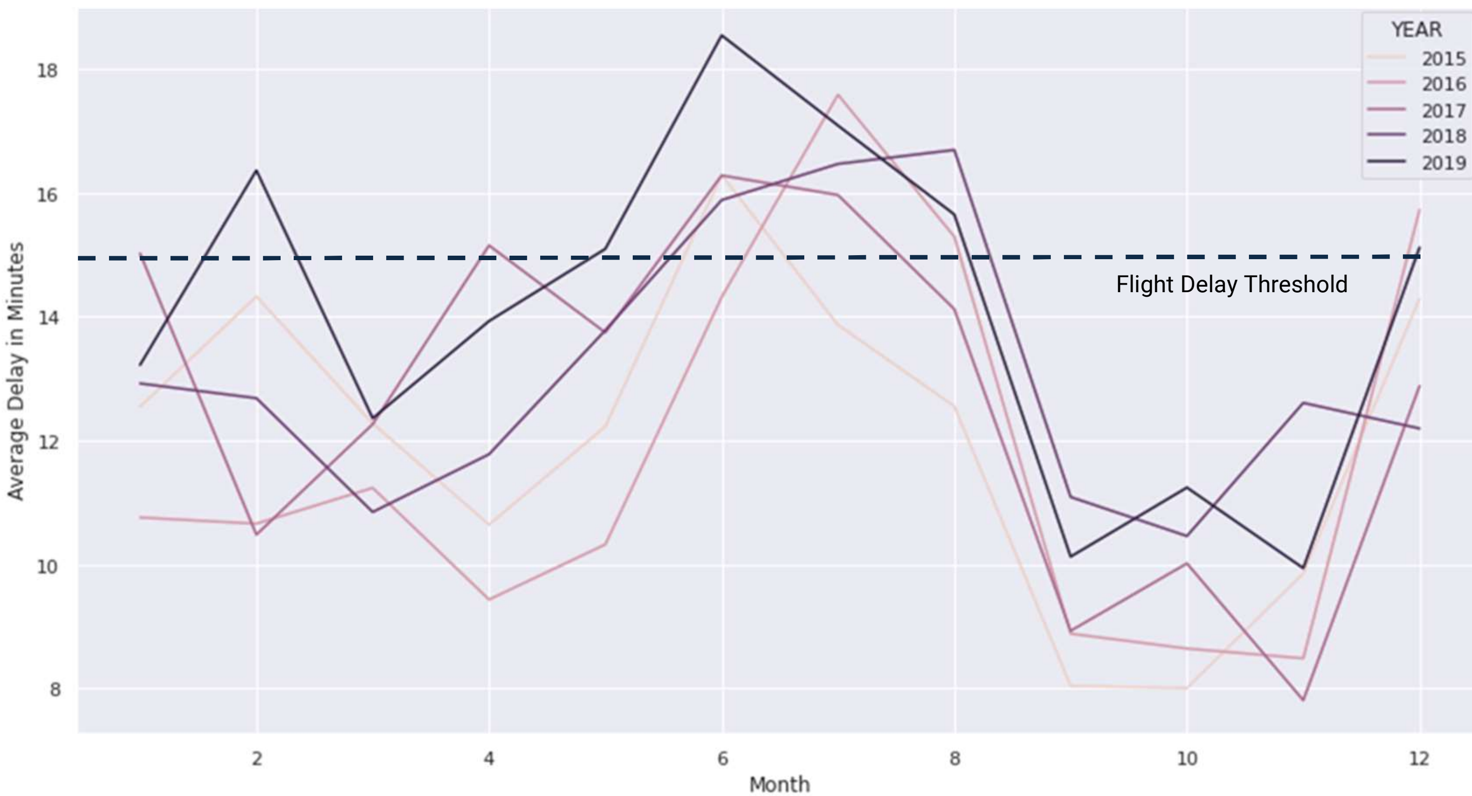


* Source: Joined Dataset

Monthly Airline Delays by Year

We examined the seasonality trends in flight delays

2015-2019 Average Monthly Flight Delays

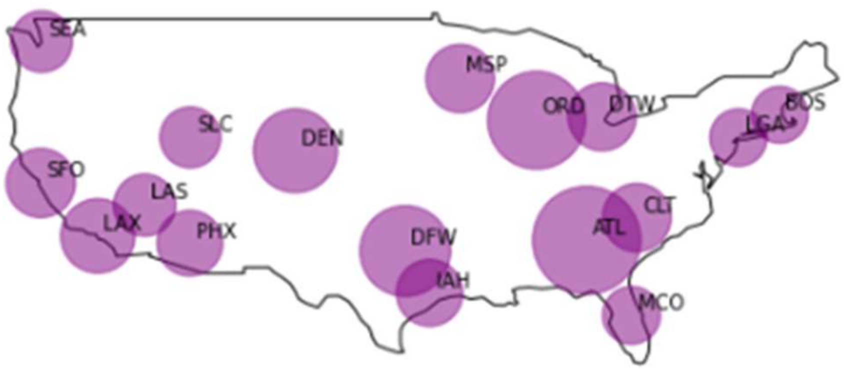


* Source: Joined Dataset

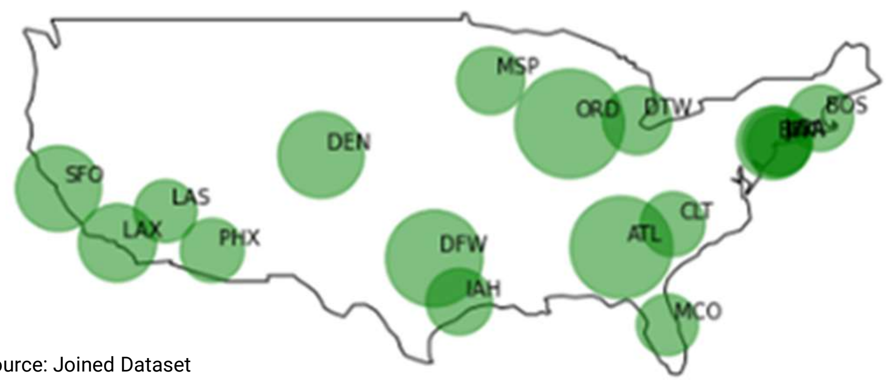
Graph Analytics - top 5% (17 airports)

Page Rank weighted by flights and delays

Top 5% Airport Connections Page Rank



Top 5% Airport Delays Page Rank



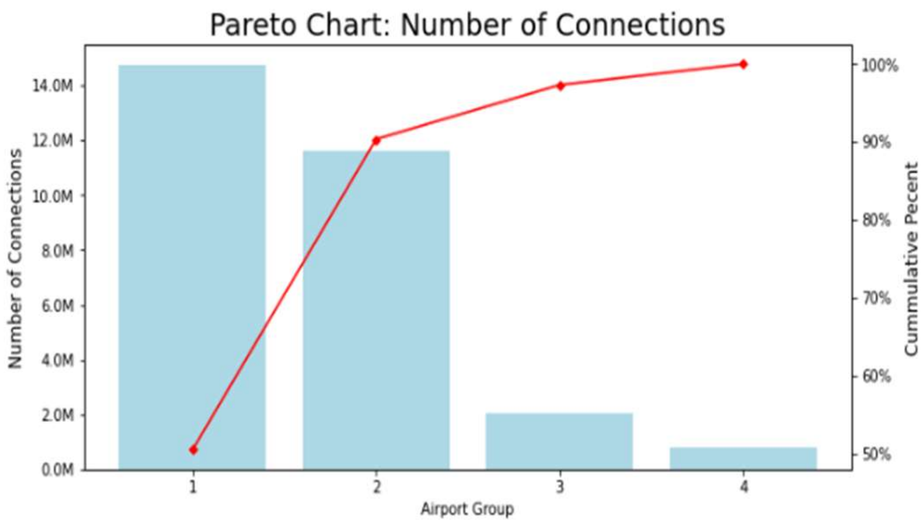

```python
def pgrnk_weighted_conn(df):
    df = df.select("ORIGIN_AIRPORT_ID", "DEST_AIRPORT_ID")
    df = df.withColumn("connection", lit(1))
    airportsRDD_conn = connections_weight(df).cache()

    arpcnn = airportsRDD_conn.flatMap(lambda x: calc_cnn(x)).cache()
    colnames = ["airport", "N_Air_to", "Num_connection"]
    airport_conn = arpcnn.toDF(colnames)
    Nnodes_delay = count_nodes(airportsRDD_conn)

    # Initialize Graph
    init_graph_connections = initGraph(airportsRDD_conn)

    # Get Page Rank
    nIter = 10
    start = time.time()
    graph_connections = runPageRank(init_graph_connections, alpha = 0.15, maxIter = nIter, verbose = True)
    airportConnRank = graph_connections.toDF(["airport", "Conn_Ranking"])
    return airportConnRank
```

* Source: Joined Dataset

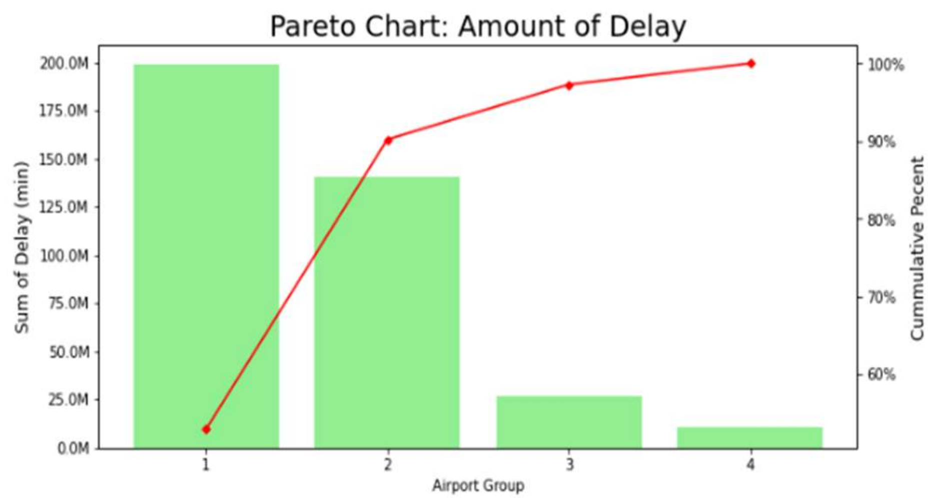
Pareto Distribution of Airport Influence

Flight data is a classic example of Power Law Distribution where a small number of airports hubs hold disproportionately high influence on the transport network delays and traffic

328 Airports



17 65 82 164



17 65 82 164

* Source: Joined Dataset

Outcome and Feature Engineering

* Weather measure within a 2 hours horizon window.

Outcome = **Features:** Information available <u>two hours</u> before scheduled Departure**

33

1 **Delayed or Not (by >15 min)**

2 **Time of Delay (Multi Task) (in Minutes)**

Flight Related (F_i)	Weather Related (W_j)* (Departure and Destination)	Others (O_k)
• Airport (Origin, Destin.) • Airline Carrier • Flight Number • Departure Time • Date Related Variables: Year, Quarter, Month, Day of Month, **Season, Weekday, Time of Day** • Length of flight	• Wind Angle (0 to 360) • Wind Speed (m/s - scale 10) • CIG - Ceiling Height Dimension (km) • Visibility (meters) • DEW point • Atmospheric pressure • Precipitation hour (mm / six hours)	• **Airport Traffic Page Rank(Unweighted)** • **Lagged Weighted Page Rank by Delay and Flights** • **Origin/Destination Delay Pairs** • **Rolling Ninety Day Average Delay**

1 $$P(\text{Delay}) = \frac{1}{1 + e^{-(\beta_0 + \sum_{i=1}^{n}(\beta i * \text{Flight}_i) + \sum_{j=1}^{m}(\beta j * \text{Weather}_j) + \sum_{k=1}^{l}(\beta k * \text{Others}_k))}}$$

2 **Gradient Boosted Trees / Random Forest / Linear Support Vector Machine**

3 **Ensemble Learning (Stacking)**

4 **Multi Task Algorithm**

Key Features:

~95% Weight

33 Features → 19 Features

Accuracy = 64.9%
F1_score = 41%

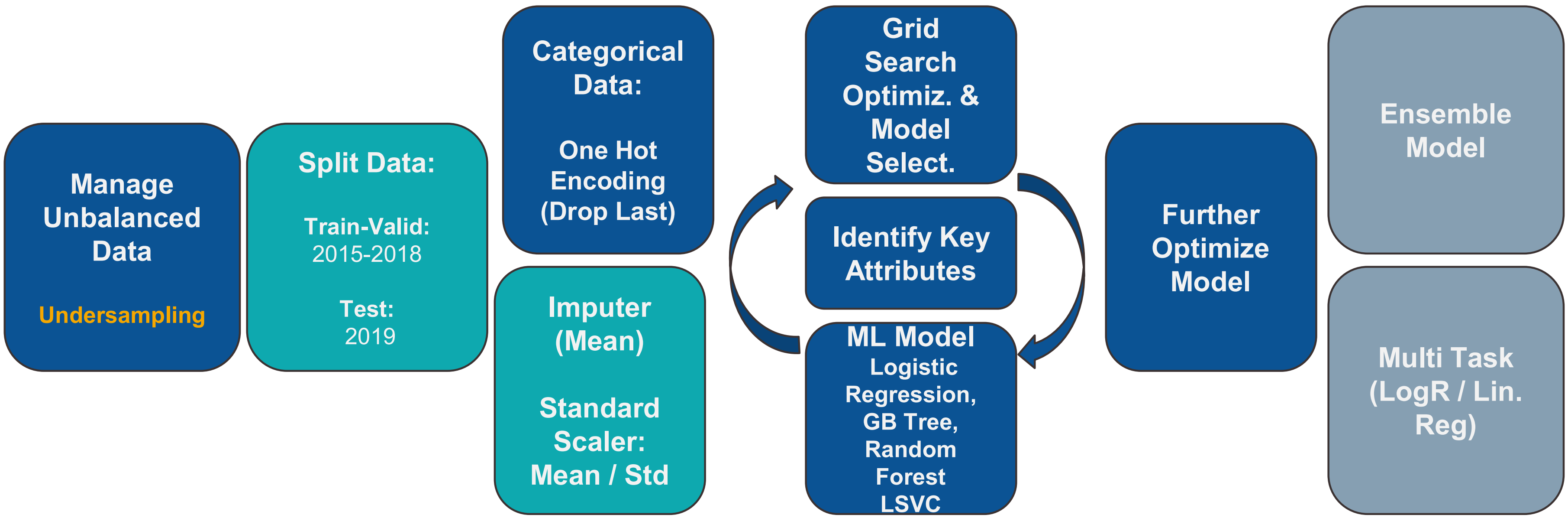
Accuracy = 64.8%
F1_score = 40.8%

Num Feat.	Type of Feature	Importance
10	Engineered Feature	53.7%
5	Time Related Feature	21.7%
14	**Weather Related Feature**	**18.4%**
1	Airline	3.5%
2	Airport Related	2.4%
1	Distance	0.3%
33	TOTAL	100.0%

Engineered Feature	Importance
OD_delay_pair	**23.1%**
time_of_day_int	**22.1%**
Season	3.5%
Page Rank Related	**3.3%**
Others	1.7%
Total	53.7%

Attribute	importance
DEPARTURE_Hour_CRS	**19.6%**

Pipeline Structure:



ML Evaluation Metrics:

Emphasis on F2_score/ Recall - Plus Accuracy

Positive: Flight Is Delayed

False Positive (FP):
"Predicted Delay, when it was NOT delayed"

- Unnecessary Stress for customer
- Customer may look for alternatives
- Negative about service before hand
- (+) May be pleasantly surprised

Recommendations:
- Provide a sense of the expected time of delay.
- Provide a sense of the likelihood of the event.

False Negative (FN):
"Predicted No Delay, when there was a Delay"

- Frustrated Passenger (helpless)
- No time to look for alternatives
- High reputation cost for service providers

Given the potential high cost, we seek to reduce "False Negatives"

Emphasis on f2_score
(Balance f_score with more weight to Recall)

$$Fbeta = \frac{((1 + beta^2) * Precision * Recall)}{(beta^2 * Precision) + Recall)}$$

ML Algorithms Explored:

1 Delayed or Not (by >15 min)

Outcome Variable

Train-Val data (balanced): 2015-2018; Test Data: 2019 (Imbalanced)
Unbalance Data Approach: Undersampling.

Log Reg

- Max Iterations = 20
- regParam= 0.01
- elasticNetParam= 0.5
- weightCol= "weight"

GB Trees

- lossType = "logistic"
- maxIter=20
- maxDepth=5
- weightCol= 'weight'

LSVC Trees

- regParam = 0.01

Random Forest

- Number of Trees = 110
- Max Depth = 16
- Weight = 'weight'

Ensemble Model

- Models used: RF (110 trees, 16 max depth), GBT(maxIter=20, maxDepth=5), & LSVC(regParam = 0.01),

Metric	Baseline	LR_test	GBT_test	LSVC test	RF_test	Ensemble Test
Accuracy	82.0%	59.6%	68.3%	57.7%	**65.7%**	65.2%
Precision	0.0%	27.1%	31.7%	26.1%	**30.2%**	30.0%
Recall	0.0%	68.7%	60.3%	69.1%	**63.8%**	64.8%
Specificity	NA	57.5%	70.1%		**66.2%**	
F1_Score	NA	38.9%	41.5%	37.9%	**41.0%**	41.0%
F05_Score	NA	30.8%	35.0%	29.8%	**33.8%**	33.6%
F2_Score	NA	52.6%	51.1%	52.0%	**52.2%**	52.6%

Able to Predict

$4.1B out of $6.4B

Opportunity for Independents Models

Time of Day

Time of Day	% of Total	% Delayed	Accuracy
Morning	20.5%	8.6%	69.85%
Mid Morning	12.2%	13.8%	62.31%
Mid Day	**18.2%**	**17.5%**	**60.00%**
Early Afternoon	**17.6%**	**22.1%**	**62.21%**
Late Afternoon	17.7%	26.1%	64.85%
Evening	10.4%	26.0%	66.96%
Night	3.4%	10.6%	69.51%
Total	100.0%	18.2%	

Airport Group

Airport Group	# Airports	% of Total Conn.	% Delayed	Accuracy
Group 1	**17**	**51%**	**18.9%**	**63.9%**
Group 2	65	40%	18.0%	65.7%
Group 3	82	7%	15.5%	64.3%
Group 4	164	3%	14.3%	62.7%
Total	328	100%	18.2%	

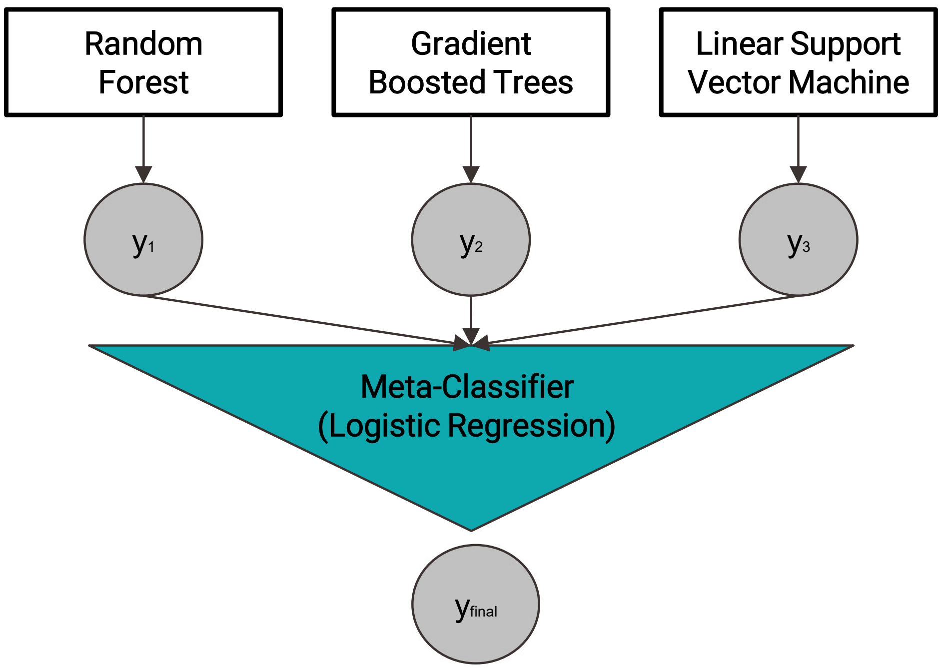
Season

Season	% of Total	% Delayed	Accuracy
Spring	25.3%	17.8%	64.9%
Summer	**26.5%**	**21.6%**	**65.3%**
Fall	25.1%	14.3%	63.1%
Winter	23.2%	19.0%	64.2%
Total	100.0%	18.2%	

Ensemble Learning Algorithm

Ensemble learning combines several model predictions

Ensemble Stacking Model



PySpark Implementation

```python
def ensemble_learning_model(gbt_model, rf_model,svc_model, train_val, meta_features):
  """
    Return predictions of Stacking Ensemble Learning Model in Spark DataFrame.
    Args:
        gbt_model - Gradient Boosted Tree spark fitted model
        rf_model - Random Forest spark fitted model
        meta_classifier - Logistic Regression fitted model
        meta_features - (list) meta feature prediction column names
        train_val - (spark DataFrame) training data
  """
  # get predictions from each model
  data1 = rf_model.transform(train_val).select(categoricals+numerics+ ["prediction","label"]) \
                .withColumnRenamed('prediction', 'rf_pred')
  data2 = gbt_model.transform(data1).select(categoricals+numerics+["prediction","label","rf_pred"]) \
                                  .withColumnRenamed('prediction', 'gbt_pred')
  data3 = svc_model.transform(data2).select(categoricals+numerics+["label", "prediction","rf_pred", 'gbt_pred']) \
                                  .withColumnRenamed('prediction', 'svc_pred')
  # pre-process meta-features
  preds = preprocess_data(data3, meta_features).cache()

  # create meta-classifier
  lr = LogisticRegression(featuresCol='meta_features', labelCol='label', predictionCol='meta_pred', maxIter=20,
                         regParam=1., elasticNetParam=0)
  meta_classifier = lr.fit(preds)
  meta_preds = meta_classifier.transform(preds)
  return meta_classifier, meta_preds
```

Multi-Task Algorithm From Scratch

Multi-task algorithm combines the Logistic & Linear Regressions

Task 1 Prediction (Linear Regression)	Task 2 Prediction (Logistic Regression)

Multi-Task Loss Function
β(MSE) + (1-β) (LogLoss)

Dataset	Multi - Linear (RMSE)	Linear (RMSE)
Test (Full 2019)	49.820668	52.296001
Train (2.5%)	70.557089	62.080181

Dataset	Multi - Logistic (Accuracy)	Logistic (Accuracy)
Test (Full 2019)	0.586973	0.587091

* Trained on 2.5% of the data with beta = .4

PySpark Implementation

```python
def MTLoss(data, modelLR, modellogr, beta):
    """
    Compute multi-task loss function
    Args:
        data     - each record is a tuple of (features_array, y)
        modelLR  - (array) Linear Regression model coefficients with bias at index 0
        modellogr - (array) Logistic Regression model coefficients with bias
        beta - (numeric) weight for the loss function of each
    """
    augmentedDF = data.map(lambda x: (np.append([1.0], x[0]), x[1]))
    lossLR = None

    multi_loss = augmentedDF.map(lambda x:( (np.dot(x[0],modelLR) - x[1][1])**2, ((x[1][0] * np.log(sigmoid(np.dot(x[0],
modellogr))))+ ((1-x[1][0])*np.log(1-sigmoid(np.dot(x[0], modellogr)))))        ))

    lossLogR = multi_loss.map(lambda x: x[1]).mean()
    lossLogR = -lossLogR
    lossLR = multi_loss.map(lambda x: x[0]).mean()

    # Multi-task Loss
    MTLossVal = lossLR*beta + (1-beta)*lossLogR

    return MTLossVal
```

GAP Analysis

We achieved a moderately performance based on a benchmark comparison to the leaderboard.

Our Model - Team 14	LeaderBoard
Model Performance: Accuracy 65.7%; F1-Score 41.0%; F2-Score 52.2%; Recall 63.8% (Random Forest)	Team 7 Accuracy 85%; Team 11 F1-Score 80% (XGBoost)
Join Time: 2 hours	Team 25: 4.5 minutes
Training and Test Time: 30 minutes	Team Super Mario: 7 minutes

Performance & Scalability:

- Our best model was Random Forest (110 trees / 16 maxDepth) - the more trees, the longer the training time. Though, feature reduction helped significantly.

Limitations, Challenges and Future Work:

- Any exceptions in the Spark pipeline stages (e.g. OHE, Scaler etc.) are challenging to track and troubleshoot
- Accessing Hyper parameters and variables inside the pipeline stages can pose debugging challenges. The optimization is by Accuracy and not by F2-Score.
- An opportunity exists to improve our model performance by creating custom models for certain seasons (Fall) and Airport Buckets (Group 1- top airports)
- We could fine-tune the model further if given more time or explore more sophisticated algorithms (i.e. neural networks, XGBoost)
- Explore additional features

Thank you

Any Questions?

Slide Links

Pipeline Structure: Code Example

- Define the model – example for Random Forest

```
rf = RandomForestClassifier(featuresCol = 'scaledFeatures', labelCol = 'label',
                            featureSubsetStrategy='auto',
                            impurity='gini',
                            seed=123)
```

- Set the Grid Search set of parameters

```
grid = ParamGridBuilder()\
            .addGrid(md.maxDepth, [5, 8, 10, 14])\
            .addGrid(md. numTrees, [50, 110, 150, 200, 250])\
            .build()
```

- Define Pipeline for the model

```
pipeline = Pipeline(stages=model_matrix_stages+[scaler]+[rf])
```

- Define evaluator

```
evaluator = BinaryClassificationEvaluator()
```

- Define CrossValidator for model tuning

```
crossval = CustomCrossValidator(estimator=pipeline,
                                estimatorParamMaps=grid,
                                evaluator=evaluator,
                                splitWord = ('train', 'test'),
                                cvCol = 'cv',
                                parallelism=4)
```

- Fit the Cross Validation to the Data Segments (folds), and select the best model

```
Pipeline_rf = crossval.fit(data_segments)

rf_model = Pipeline_rf.bestModel
```

- Generate Predictions from model using test data

```
pred = rf_model.transform(test).select("DEP_DEL15", "prediction")
```

- Get metrics based on predictions

```
metricsT = MulticlassMetrics(pred.rdd.map(lambda x: (x[1], x[0])))
```

Backup

Confusion Matrix:

While **Training** and **Validating** on **Balanced** data, when **Testing** on **Imbalanced** Test Data - **Precision** goes Down vs. Recall driving F1 and F2 scores down.

Metric	RF 110/16
Accuracy	65.7%
Precision	30.2%
Recall	63.8%
F1_Score	41.0%
F05_Score	33.8%
F2_Score	52.2%

Label	Pr (0)	Pr (1)
0	3,715,598	1,899,150
1	466,926	823,411

Label	Pr (0)	Pr (1)
0	66.2%	33.8%
1	36.2%	63.8%

Backup

	PREDICTION	
	0	1
0	82	0
1	18	0

	P(0)	P(1)
0	TN	FP
1	FN	TP

$$R = \frac{TP}{TP+FN}$$

$$P = \frac{TP}{TP+FP}$$

	0	1
0	82	0
1	18	0

JOINS

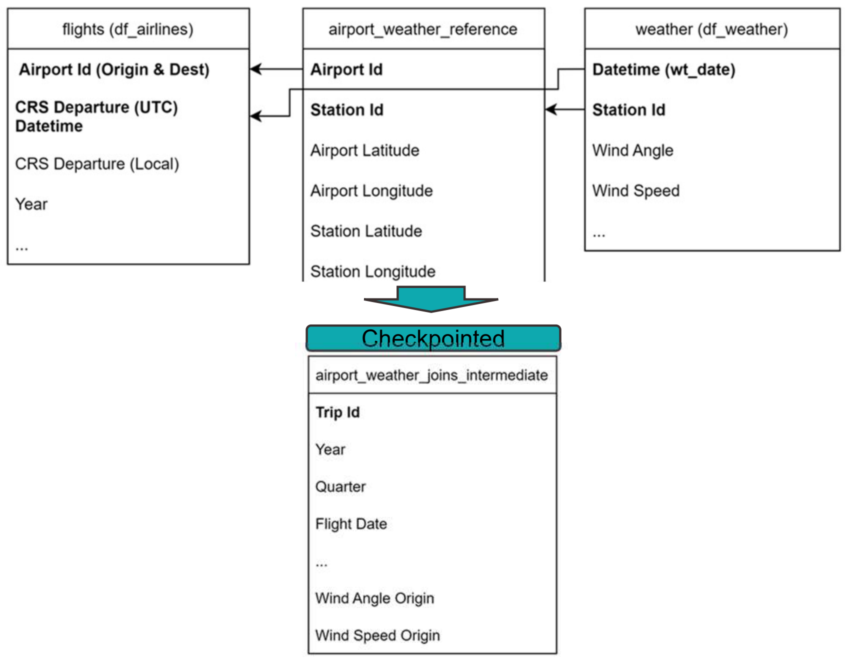
1 Create Reference Table

airport_weather_reference
Airport Id
Station Id
Airport Latitude
Airport Longitude
Station Latitude
Station Longitude
Distance Kilometers
Timezone

.183 hours

*Haversine Distance Formula used to find 5 closest Weather Stations

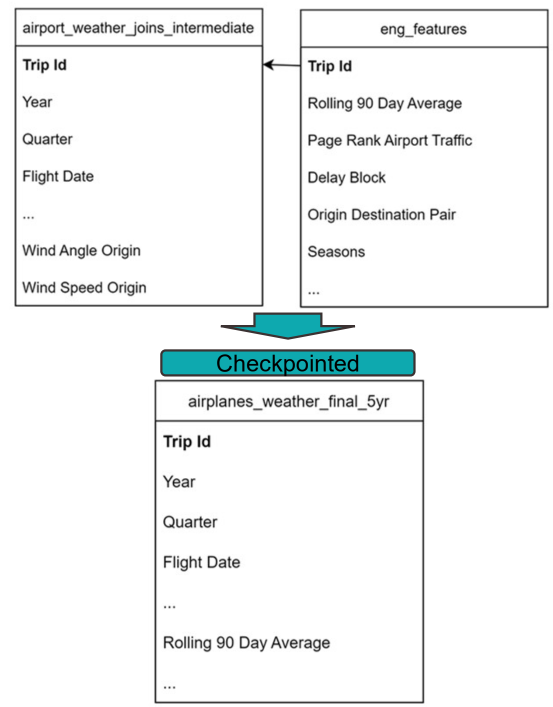
2 Join weather and flights data



3.21 hours

** Departure Times were converted to UTC to join with weather table
*** Added in weather data that is in window of 2-3 hours before flight departure from 3 closest airports
**** Added weather for origin and destination airports

3 Join engineered features



.183 hours

Flight Duplicates

Figure 3.1, demonstrates there are duplicate records in the flight dataset caused by the `airlines_size_test.parquet` file. With this file included, the total records are 63,493,682. We will need to remove these duplicates from the flights dataset prior to joining to the weather dataset to ensure we are only including the 31,746,841 flights.

Figure 3.1 Duplicate Records in Flights Table

Name	Year	Records	Unique Records
2015.parquet	2015	5,819,079.00	5,819,079.00
2016.parquet	2016	5,617,658.00	5,617,658.00
2017.parquet	2017	5,674,621.00	5,674,621.00
2018.parquet	2018	7,213,446.00	7,213,446.00
2019.parquet	2019	7,422,037.00	7,422,037.00
airlines_size_test.parquet	Additional File	31,746,841.00	31,746,841.00

Metrics:

- *Accuracy = (TP+TN)/(TP+FP+FN+TN)*
- *Precision = TP/(TP+FP)*
- *Recall = TP/(TP+FN) (sensitivity)*
- *F1 Score = 2*(Recall * Precision) / (Recall + Precision)*
- *Specificity = TN/(TN+FP)*
- *Fbeta = ((1 + beta^2) * Precision * Recall) / (beta^2 * Precision + Recall)*
    a. *F0.5-Measure (beta=0.5): More weight on precision, less weight on recall.*
    b. *F1-Measure (beta=1.0): Balance the weight on precision and recall.*
    c. *F2-Measure (beta=2.0): Less weight on precision, more weight on recall*

Confusion Matrix		Prediction		
		0	1	
Label	0	62.7%	37.3%	100.0%
	1	**41.2%**	58.8%	100.0%

https://towardsdatascience.com/accuracy-recall-precision-f-score-specificity-which-to-optimize-on-867d3f11124
https://machinelearningmastery.com/fbeta-measure-for-machine-learning/#:~:text=The%20F2%2Dmeasure%20is%20calculated,(4%20*%20Precision%20%2B%20Recall)

Did Stacking Improve My PySpark Churn Prediction Model?:

The base models are chosen to be different algorithms which achieve a wide range of results. The meta model is usually logistic regression. The base models are fit on the train set and they make predictions on the validation set. The meta model is trained on the dataset of predictions of the base models, and it then predicts on the test set.

The methodology in this case is outlined below:

- the features set dataframe is prepared as in the first blog,
- stratified split the dataset into a train set, a validation set and a test set with ratios 4:4:2,
- process the data for modelling,
- fit 5 base classifiers (LR, RF, GBT, MLPC, LSVM) on the train set,
- use the base models to make predictions on the validation set,
- create a meta features dataset that includes the predictions and the probabilities (when available) from the base classifiers, as well as the label column,
- train and fine tune a Logistic Regression meta model, using 5-fold cross validation and grid search on the meta features dataset,
- the tuned meta classifier makes predictions on the meta features set build from the test set.

https://medium.com/@silviaonofrei/did-stacking-improve-my-pyspark-churn-prediction-model-230bdf3b9675

Managing Unbalanced Data

Explored Alternatives with 2.5% of full data:

23.9M

No Delayed 82%

Delayed 18%

5.3M

F2_score

Oversampling +64%: 23.9 / 23.9 — ~53%

Undersampling -64%: 5.3 / 5.3 — ~60%

Balanced ~Same Size: 14.6 / 14.6 — ~55%

Other Alternative:

Leveraging Weight Parameter for ML Algorithm

$$w_i := \frac{n}{n_i * C}$$

W_i = Weight for Class i. n = Total Observations, n_i = number of observations for class i, C = number of classes

ML Algorithms Explored:

1 Delayed or Not (by >15 min)

Outcome Variable

Train-Val data: 2015-2018; Test Data: 2019 (Balanced)
Unbalance Data Approach: Undersampling, Oversampling, Balanced

Log Reg

- Max Iterations = 20
- regParam= 0.01
- elasticNetParam= 0.5
- weightCol= "weight"

GB Trees

- lossType = "logistic"
- maxIter=20
- maxDepth=5
- weightCol= 'weight'

LSVC Trees

- regParam = 0.01

Random Forest

- Number of Trees = 110
- Max Depth = 16
- Weight = 'weight'

Ensemble Model

- Models used: RF (110 trees, 16 max depth), GBT(maxIter=20, maxDepth=5), & LSVC(regParam = 0.01),

Metric	Ensemble	GBT	LSVC	LogR	RF19-110/16
Accuracy	**65.8%**	**65.5%**	**63.3%**	**63.5%**	**65.0%**
Precision	65.3%	67.9%	62.4%	62.8%	66.0%
Recall	**66.7%**	**60.6%**	**69.4%**	**68.9%**	**64.1%**
F1_Score	66.0%	64.0%	65.8%	65.7%	65.0%
F05_Score	65.6%	66.3%	63.7%	63.9%	65.6%
F2_Score	**66.4%**	**61.9%**	**67.9%**	**67.6%**	**64.5%**

Label	Pred (0)	Pred (1)
0	815,460	434,696
1	455,084	835,253

Label	Pred (0)	Pred (1)
0	65%	35%
1	35%	65%

Top 17 Airports Ranked by Connections Page Rank

5% of Airport, 50% of all Connections

Rank	airport	Connec_Ranking	Num_connection	ORIGIN	ORIGIN_CITY_NAME
1	**10397**	0.05949	1,865,310	ATL	Atlanta, GA
2	**13930**	0.04876	1,428,494	ORD	Chicago, IL
3	**11298**	0.04188	1,158,829	DFW	Dallas/Fort Worth, TX
4	**11292**	0.03573	1,090,882	DEN	Denver, CO
5	**12892**	0.02811	1,045,707	LAX	Los Angeles, CA
6	**14771**	0.02436	819,507	SFO	San Francisco, CA
7	**11057**	0.02386	759,787	CLT	Charlotte, NC
8	**13487**	0.02372	659,920	MSP	Minneapolis, MN
9	**11433**	0.02351	657,040	DTW	Detroit, MI
10	**12266**	0.02267	744,566	IAH	Houston, TX
11	**14107**	0.02201	787,700	PHX	Phoenix, AZ
12	**12889**	0.01964	731,648	LAS	Las Vegas, NV
13	**14747**	0.01928	642,030	SEA	Seattle, WA
14	**14869**	0.01888	501,813	SLC	Salt Lake City, UT
15	**13204**	0.01697	623,649	MCO	Orlando, FL
16	**12953**	0.01677	605,224	LGA	New York, NY
17	**10721**	0.01645	630,955	BOS	Boston, MA

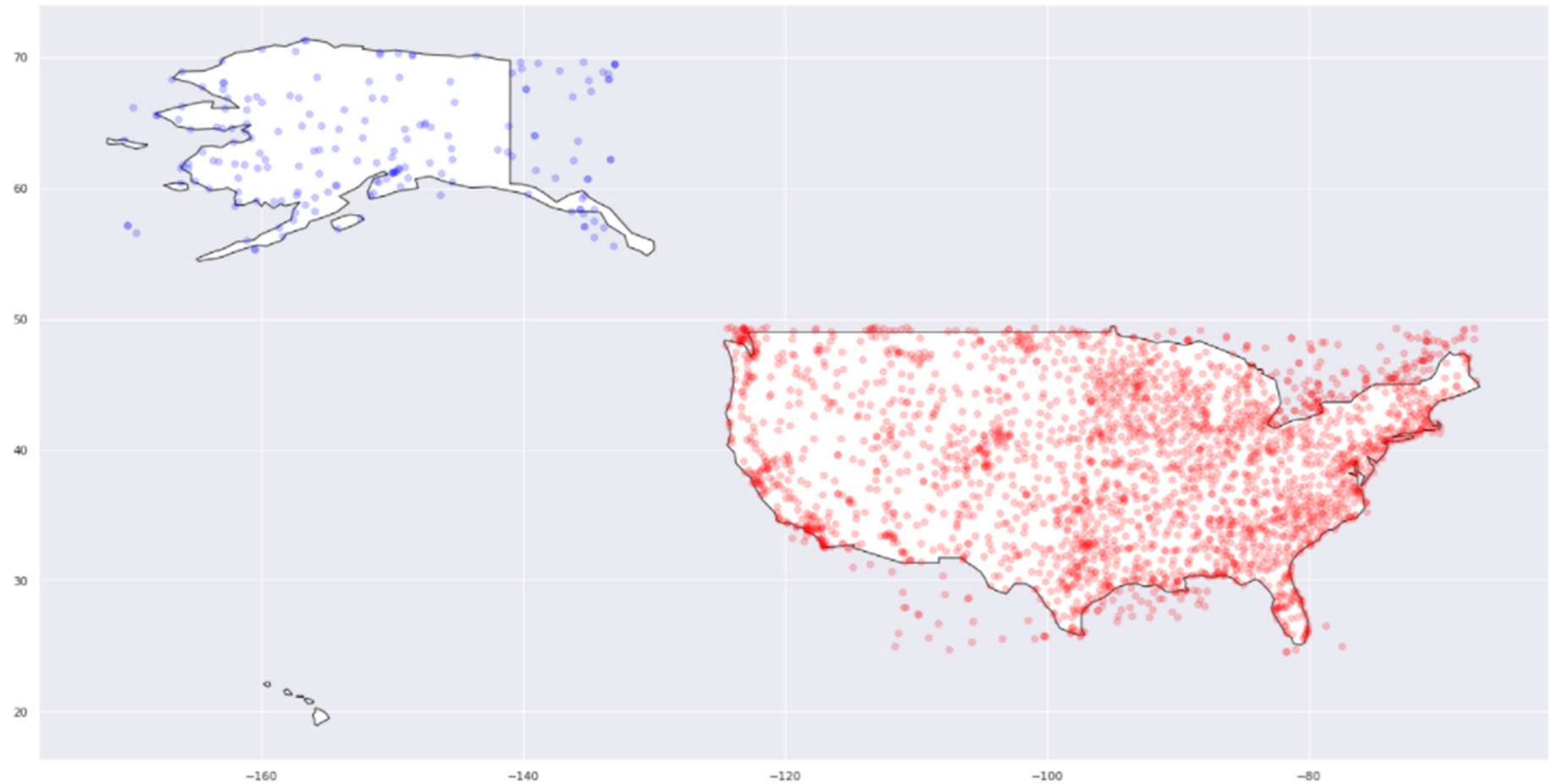
Limitations, Challenges, & Future Work

- Data Quality (missing values, duplicates, inconsistent primary keys and timestamps)
- Feature Engineering (No historical data for beyond 2015)
- Time Consuming Joins and Preprocessing
  - 32 million records of flight data

Geo Maps:

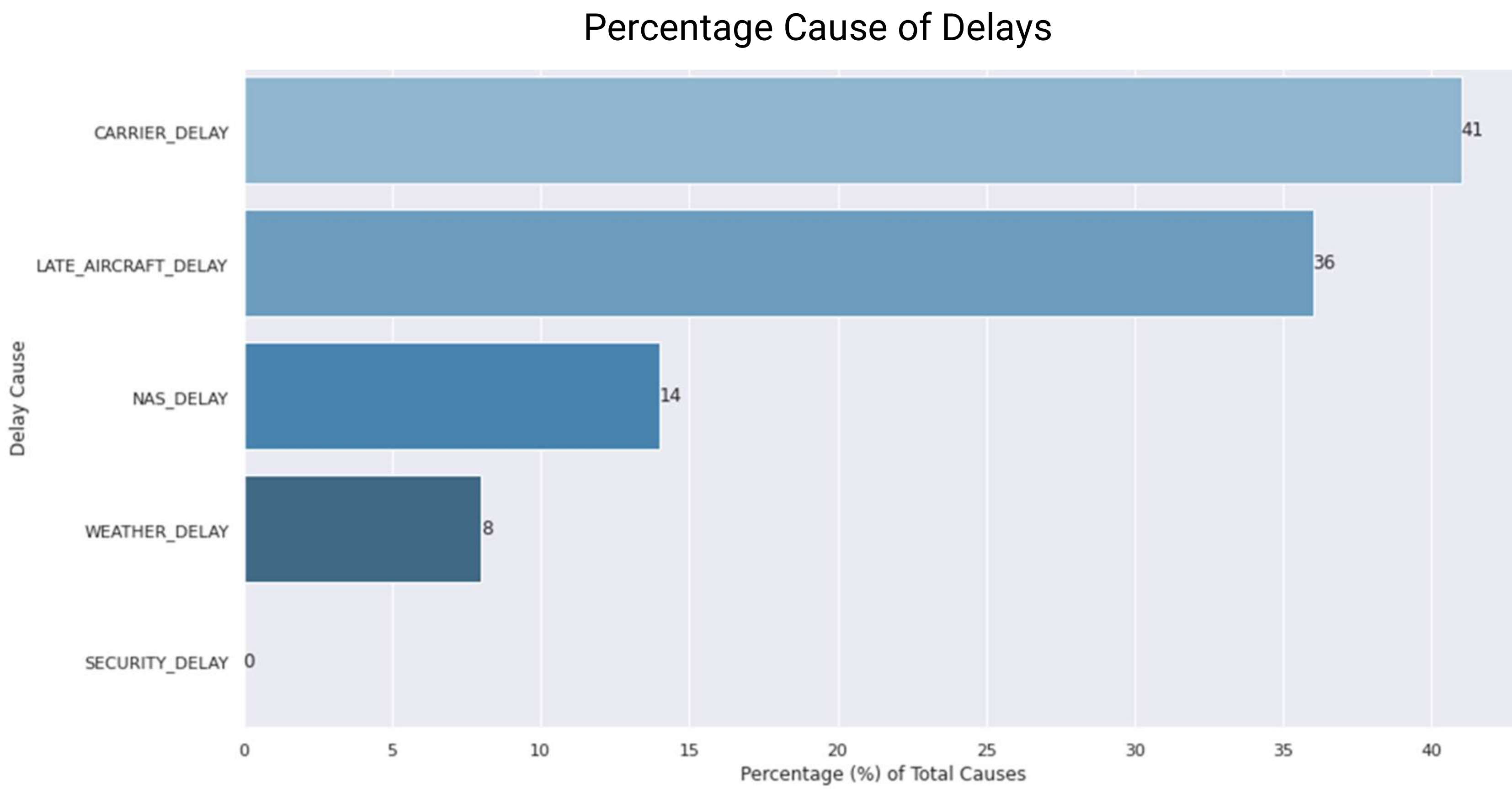
We mapped closest weather station to the Airport

U.S. Weather Station Locations



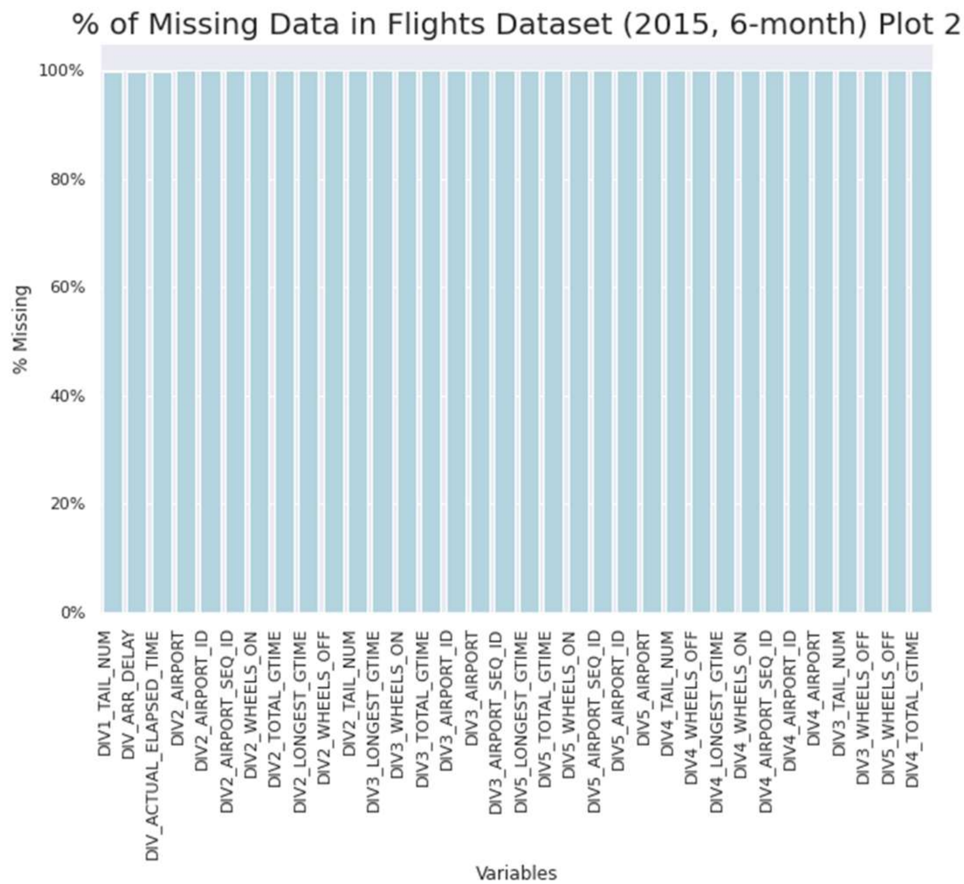
Causes of Delays:

Late Aircraft Delay & Carrier Delay are the most important factors



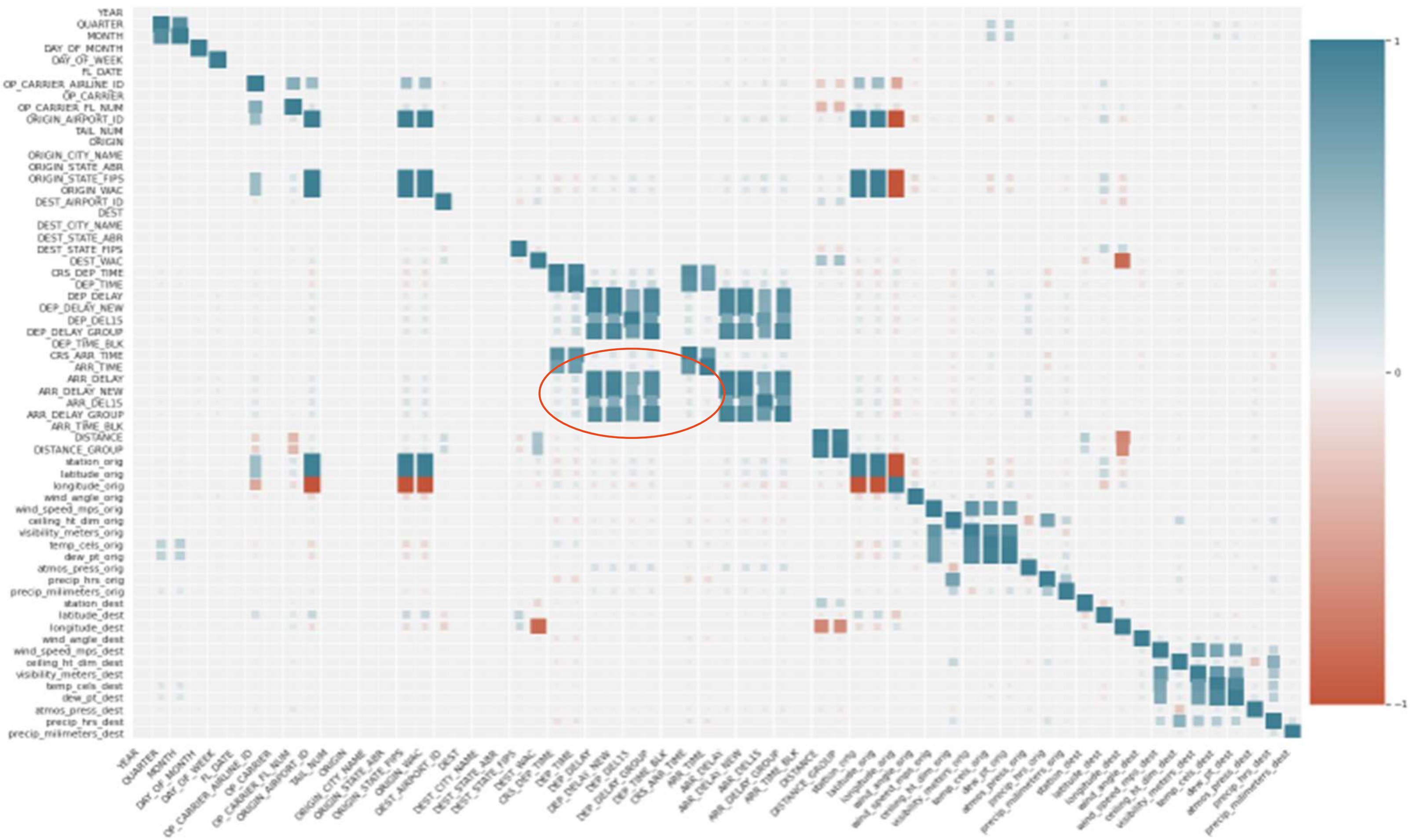
Data Cleaning:

Took out features with high proportion of missing data



- High Proportion of Missing Data in Diversion-related Fields

Correlation Heatmap



- Arrivals vs Delay

Top 10 and Bottom 10 Routes

Illinois and New York are one of the busiest cities.

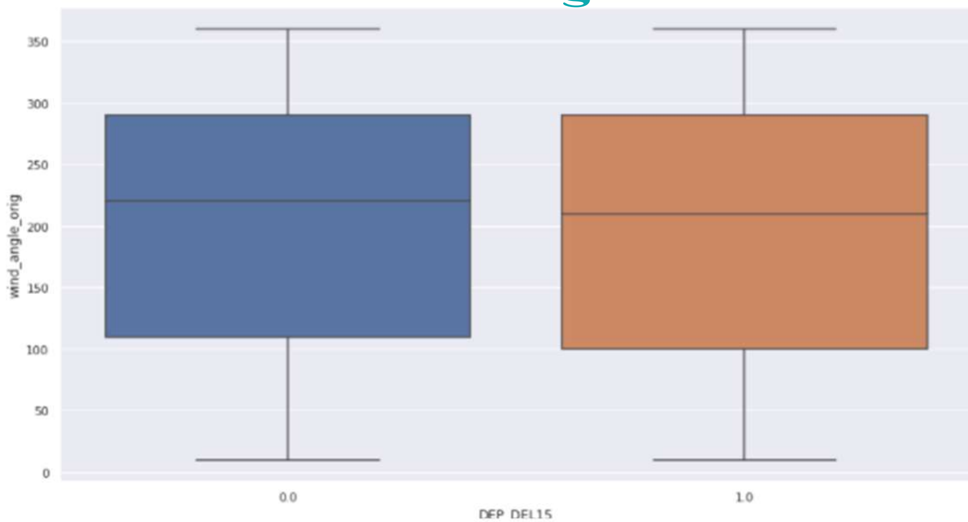
	src	dst	count_trips
1	ORD	LGA	5224
2	ATL	LGA	4545
3	ATL	MCO	4386
4	ORD	DFW	4122
5	ORD	LAX	4109
6	ATL	FLL	4070
7	ATL	DFW	3938
8	ATL	TPA	3742
9	ORD	SFO	3589
10	ATL	DCA	3490

COD is a county airport in Colorado with only one runway.

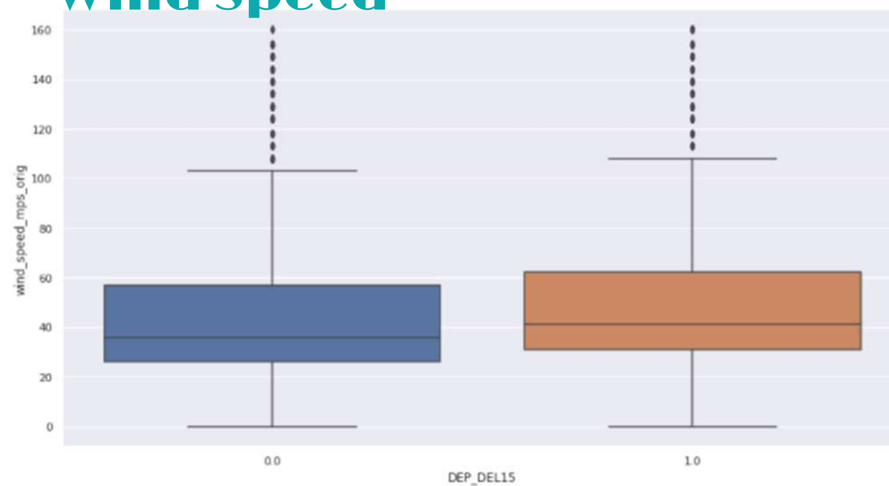
	src	dst	count_trips
1	ORD	COD	2
2	ATL	OAK	2
3	ATL	ONT	3
4	ATL	ELM	4
5	ATL	MSO	6
6	ATL	FCA	7
7	ATL	RAP	7
8	ATL	TVC	9
9	ATL	LAN	13
10	ATL	MBS	15

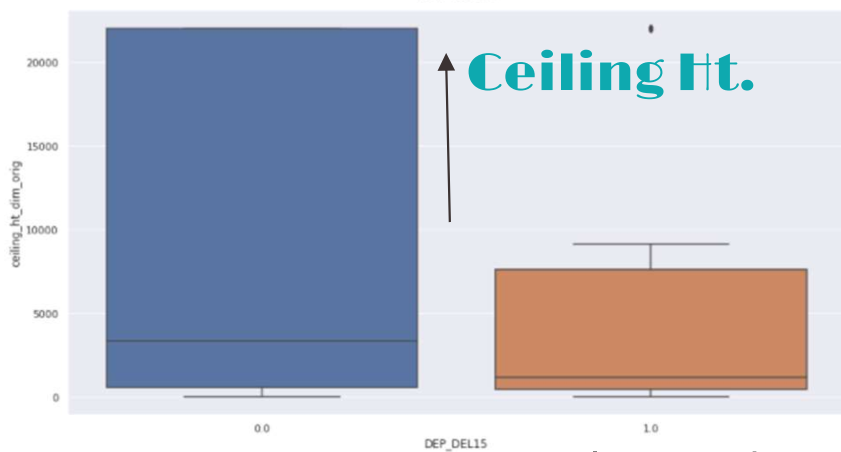
Distribution of Weather Params vs Delay

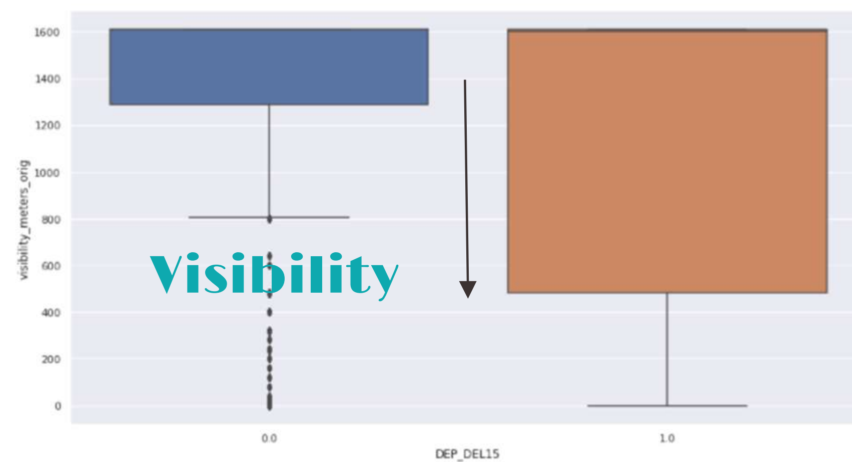
Wind Angle



Wind Speed

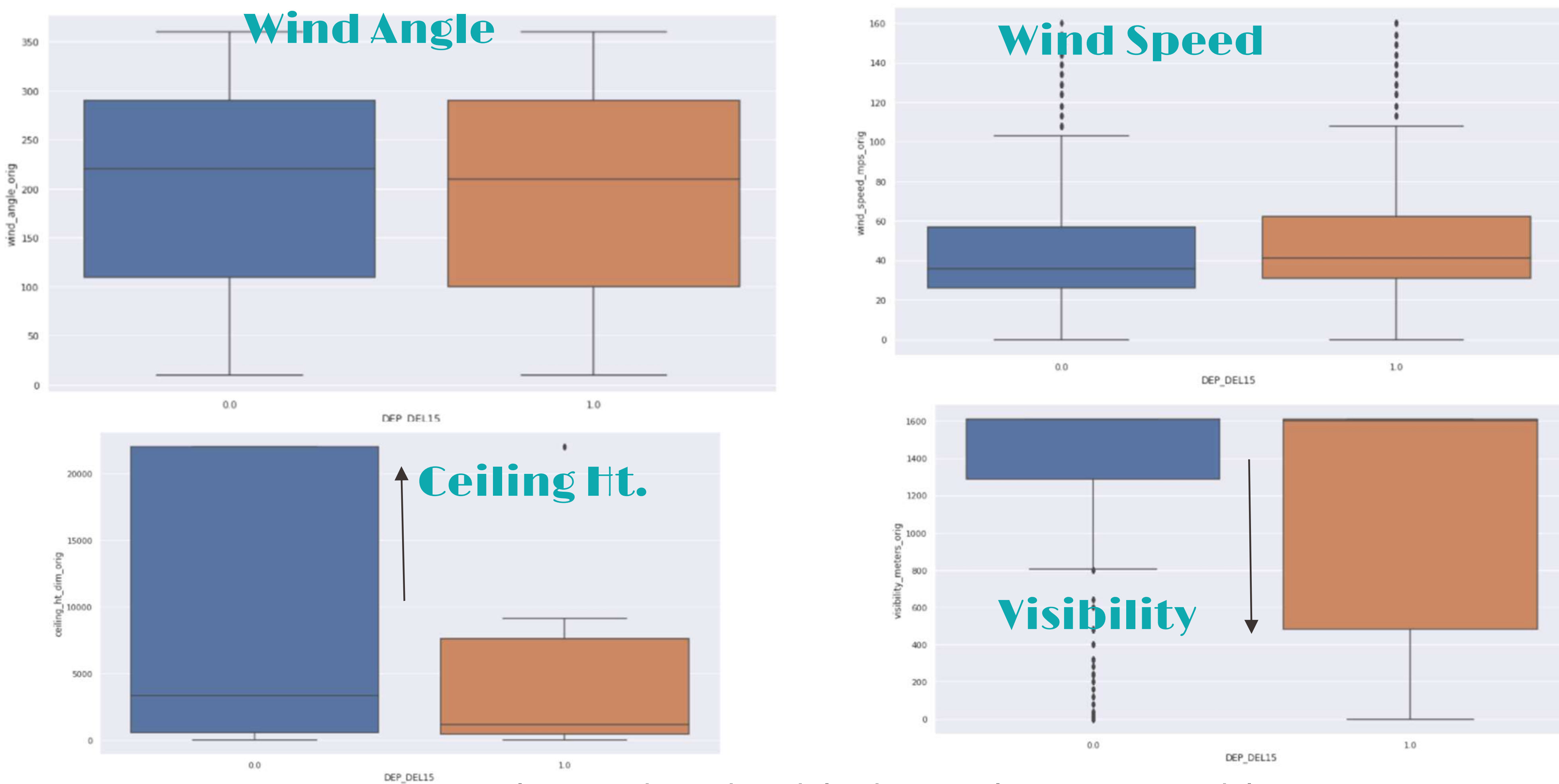






- Ceiling Height and Visibility have a clear impact on delays

Distribution of Weather Params vs Delay



- Ceiling Height and Visibility have a clear impact on delays

Missing Data Overview

We investigated why the weather and flights data was missing.

Flights Dataset	Weather Dataset *FM-15 Metar Aviation Routine Report*
● High proportion of data missing for flight diversion related fields	● High proportion of data missing for atmospheric pressure (not a mandatory value in this report) ● Weather Stations subjected to to Quality Control (V03) had fewer missing values than automated quality control Weather Stations (V02) ● The Wind Angle values are missing when the Wind Speed is equal to 0 ● Some of the values that were missing did not meet quality assurance check, were suspect, or erroneous

* Only includes weather stations in station table
** Weather data includes only the first 6 months of 2015

Missing Weather Data

Figure 2.1: Missing Weather Values by Quality

Quality	Wind Angle	Wind Speed	Ceiling Height	Visibility	Temperature	Dew Point	Atmos. Pressure	Precipitation Hours	Precipitation
1	96824	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0
5	3303454	0	0	0	0	0	0	0	0
6	972	0	0	0	0	0	0	0	0
7	66	0	0	0	0	0	0	0	2
9	198265	198612	434460	0	173869	246804	11924476	0	2292
A	27	0	0	0	0	0	0	0	0
I	9	0	0	0	0	0	0	0	0
P	107	0	0	0	0	0	0	0	0
U	10	0	0	0	0	0	0	0	0

0 = Passed gross limits check
1 = Passed all quality control checks
2 = Suspect
3 = Erroneous
4 = Passed gross limits check, data originate from an NCEI data source
5 = Passed all quality control checks, data originate from an NCEI data source
6 = Suspect, data originate from an NCEI data source

7 = Erroneous, data originate from an NCEI data source
9 = Passed gross limits check if element is present
A – Data value flagged as suspect, but accepted as good value
I – Data value not originally in data, but inserted by validator
P – Data value not originally flagged as suspect, but replaced by validator
U – Data value replaced with edited value

* Only includes weather stations in station table
** Weather data includes only the first 6 months of 2015

Missing Weather Data

Figure 2.2: Missing Weather Values by Station

Quality Control	Average Number Missing *Excludes Wind Angle*	Average Number Missing *Excludes Wind Angle & Atmos. Pressure*
V020	5860.172613	116.4778206
V030	42.18965517	16.21336207

V01 = No A or M Quality Control applied
V02 = Automated Quality Control
V03 = subjected to Quality Control

* Only includes weather stations in station table
** Weather data includes only the first 6 months of 2015

Note: This is the average of the total missing values for the columns Wind Angle, Wind Speed, Ceiling Height, Visibility, Temperature, Dew Point, Atmos. Pressure, Precipitation Hours, Precipitation combined by weather station

Closest Weather Station

Haversine Distance Formula used to calculate the shortest distance between two points

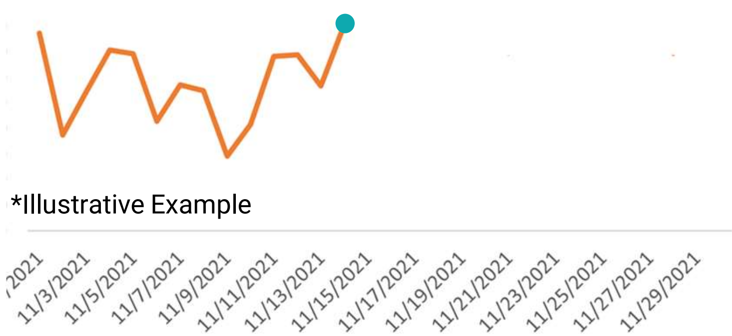
Formula	
$distance\ kilometers = 2r\sin^{-1}\sqrt{(\sin^2\frac{\Phi_2 - \Phi_1}{2}) + \cos(\Phi_1)\cos(\Phi_2)\sin^2(\frac{\lambda_2 - \lambda_1}{2})}$	** Phi is the latitude of the points *** Lambda is the longitude of the points **** The radius r is the radius of the earth (6371 kilometers)

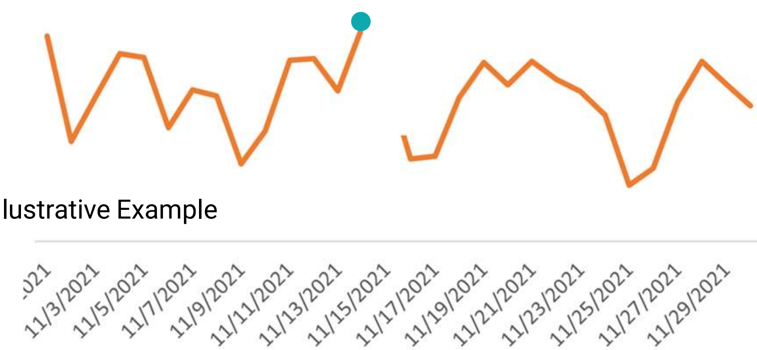
Actual Query Implementation:

```
select distinct t1.airport_id,
                t2.station_id,
                t1.Lat as air_Lat,
                t1.Long as air_Long,
                t2.lon,
                t2.lat,
                acos(
                sin(radians(t1.Lat)) * sin(radians(t2.lat)) +
                cos(radians(t1.Lat)) * cos(radians(t2.lat)) *
                cos(radians(t1.Long) - radians(t2.lon))
                ) * (6371.0) as dist_km,
                t3.tz_database as time_zone
from air_locs as t1
left join w_stats as t2 on t1.state = t2.state
```

Imputation Methods

Missing Weather data imputed using Last Observation Carried Forward and nearby weather station data

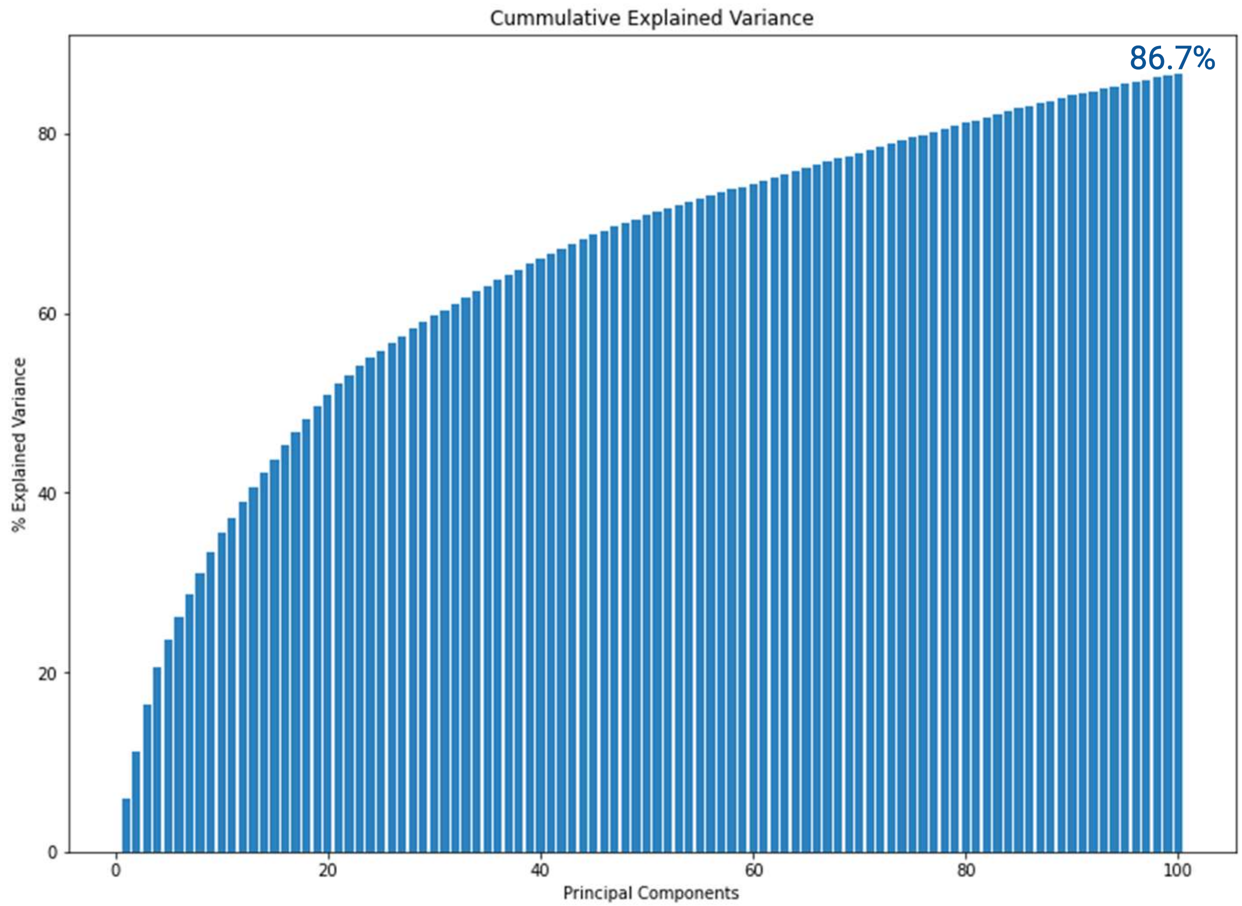




Real-Time Prediction **(Past Data Only)**	**Non Real-Time Prediction** **(Past and Future Data)**
Nearby Weather Station **Last Observed Carried Forward** Mean (Median) Random Empirical Base	Linear Interpolation Moving Average K-Nearest Neighbors Random Forest

PCA - Analysis

Preliminary Results - 100 Principal Components Explain 86.7% of variance



Code - Multi-Task Algorithm from Scratch

Multi-task loss function

```
def MTLoss(data, modelLR, modellogr, beta):
    """
    Compute multi-task loss function
    Args:
        data     - each record is a tuple of (features_array, y)
        modelLR  - (array) Linear Regression model coefficients with bias at index 0
        modellogr - (array) Logistic Regression model coefficients with bias
        beta - (numeric) weight for the loss function of each
    """
    augmentedDF = data.map(lambda x: (np.append([1.0], x[0]), x[1]))
    lossLR = None

    multi_loss = augmentedDF.map(lambda x:( (np.dot(x[0],modelLR) - x[1][1])**2, ((x[1][0] *
np.log(sigmoid(np.dot(x[0], modellogr))))+ ((1-x[1][0])*np.log(1-sigmoid(np.dot(x[0], modellogr)))))        ))



    lossLogR = multi_loss.map(lambda x: x[1]).mean()
    lossLogR = -lossLogR
    lossLR = multi_loss.map(lambda x: x[0]).mean()

    # Multi-task loss
    MTLossVal = lossLR*beta + (1-beta)*lossLogR

    return MTLossVal
```

Code - Multi-Task Algorithm from Scratch

Gradient Update

```
#helper function
def get_reg( W, regType, regParam):
    w = np.append([0.],W[1:])
    if regType =='ridge':
        reg = regParam * 2 * w
    elif regType == 'lasso':
        reg = W * 1
        reg = (reg>0) * 2- 1
        reg[0] = 0
        reg = reg * regParam
    else:
        reg = np.float(0)
    return reg


modellogr_broadcast = sc.broadcast(modellogr)
modelLR_broadcast = sc.broadcast(modelLR)
  augmentedDF =data.map(lambda x: (np.append([1.0], x[0]), x[1])).cache()

  #gradLR = augmentedLR.map(lambda d: np.dot(d[0], ( np.dot(d[0], modelLR) - d[1] )  )).mean()*2
  grads = augmentedDF.map(lambda d: (np.dot(d[0], ( np.dot(d[0], modelLR_broadcast.value) - d[1][1] )  )
,np.dot(d[0], (sigmoid(np.dot(d[0],modellogr_broadcast.value))-d[1][0]))))).cache()

  # Get regularization for each model
  lrReg = get_reg(modelLR_broadcast.value, LRregType, LRregParam)
  logReg = get_reg(modellogr_broadcast.value, logregType, logregParam)

  # add regularization to the gradients
  gradLR = grads.map(lambda x: x[0]).mean()*2 + lrReg
  grad_LogR = grads.map(lambda x: x[1]).mean() + logReg

  new_model_LR = modelLR - (learningRate * gradLR * beta)

  new_model_logR = modellogr_broadcast.value - (learningRate * grad_LogR * (1- beta))

  return new_model_LR, new_model_logR
```

Multi-Task Algorithm Formula

Simplified Multi-Task Loss Function

$$Multi - Task\ Loss = (1-\beta)LogLoss + (\beta)MSE$$

Multi-Task Loss Function

$$Multi - Task\ Loss = (1-\beta)\left(-\frac{1}{m}\sum_{i=1}^{m} y^{(i)} log h_\theta\left(x^{(i)}\right) + (1-y(i))log\left(1-h_\theta\left(x^{(i)}\right)\right)\right) + (\beta)\frac{1}{m}\sum_{i=1}^{m}\left(y^{(i)}-\hat{y}^{(i)}\right)^2$$

*where **β**=weight of each loss function*

Like the single-task (Logistic and Linear regression) algorithms, we used a gradient descent update which was calculated by taking the partial derivative of the combined loss function with respect to the feature vector (X). Finally, we apply regularization by adding in the regularization penalties to the gradient prior to multiplying by the learning rate.

Multi-Task Gradient

$$Gradient = (1-\beta) * \sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)x_j^{(i)} + (\beta)\frac{2}{m}\sum_{i=1}^{m}\left(\left[\theta^T \cdot x_i' - y_i\right]^* x_i'\right)$$

Solving for Multi-Task Gradient

$$\frac{\partial}{\partial x}(1-\beta)\left(-\frac{1}{m}\sum_{i=1}^{m} y^{(i)} log h_\theta\left(x^{(i)}\right) + (1-y(i))log\left(1-h_\theta\left(x^{(i)}\right)\right)\right) + (\beta)\frac{1}{m}\sum_{i=1}^{m}\left(y^{(i)}-\hat{y}^{(i)}\right)^2$$

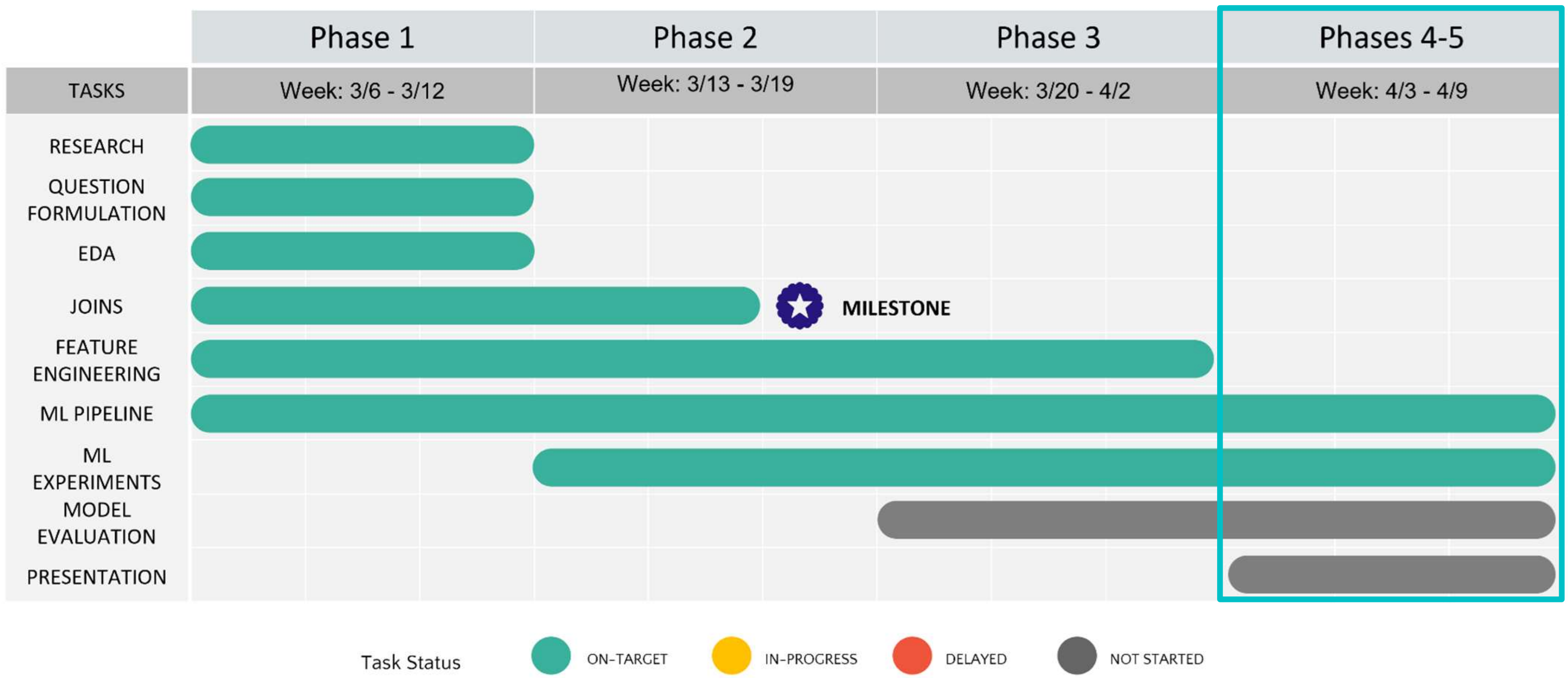
Cost to Passenger Estimate

Random Forest Model: Estimated Cost to Passengers in 2019

	True Positives	False Negatives	Total
Cost x Person per min (equiv. to $47 per Hour)	0.783333333	0.783333333	
2019 Flights Delayed	823,411	466,926	***1,290,337***
Average Passengers Per Flight (2015-2019) from BTS	90	90	
Estimated Predicted Passengers Delayed	74,106,990	42,023,340	***116,130,330***
Average Delay in Minutes 2019 from flights data	70.30548011	70.30548011	
Estimated Cost to Delayed Passengers in 2019	***$ 4,081,266,550.47***	***$ 2,314,335,690.62***	***$ 6,395,602,241.09***

* Passenger Cost of $47 from airlines.org
** Average Passengers Per Flight is the Average of Passengers for Domestic Flights using the Bureau of Transportation Statistics TransStats report
*** flights dataset used to estimate the average minutes of a delayed flight in 2019 (where DEP_DEL15=1)

TEAM 14:
GANTT CHART



- Trivago is a mid size on-line booking agency looking to offer better customer experience to passenger.
- Flight booking sites are very similar, and margins are small
- However, passengers appreciate to manage flight delays

Random Forest

	RF_train	RF_test
Accuracy	65.7%	64.8%
Precision	29.7%	29.7%
Recall	65.4%	64.8%
Specificity	65.8%	64.8%
F1_Score	40.8%	40.8%
F05_Score	33.3%	33.4%
F2_Score	52.7%	52.5%

	GBT_train	GBT_test
Accuracy	66.1%	68.3%
Precision	67.7%	31.7%
Recall	60.9%	60.3%
Specificity	71.2%	70.1%
F1_Score	64.1%	41.5%
F05_Score	66.2%	35.0%
F2_Score	62.1%	51.1%