# Predicting Flight Delays

[W261 Final Project Link]

Team 28: Aditya Bajaj, Gerrit Lensink, Ruby Han, Tsun Tsai

# Outline

01

## Project Description

02

## Pipeline

03

## Performance and Gap Analysis

04

## Limitations, Challenges & Future Work

# Project Description

**$30 billion**

Industry losses as a result of flight delays

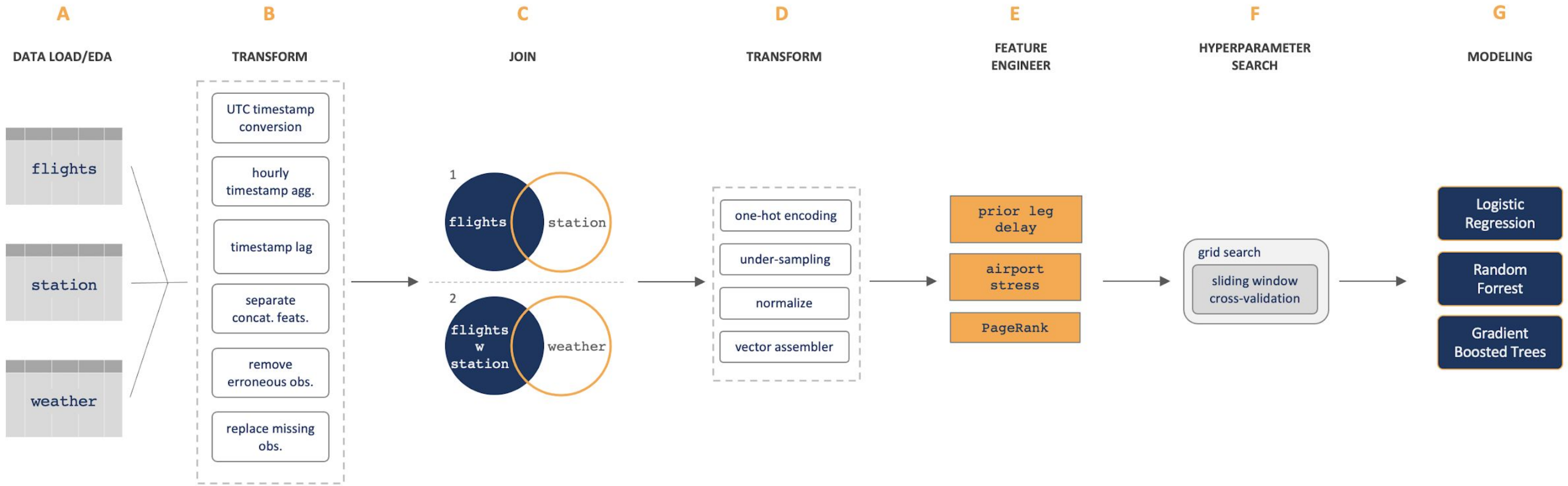
**Delay reduction**

Motivation for airlines to reduce flight delays

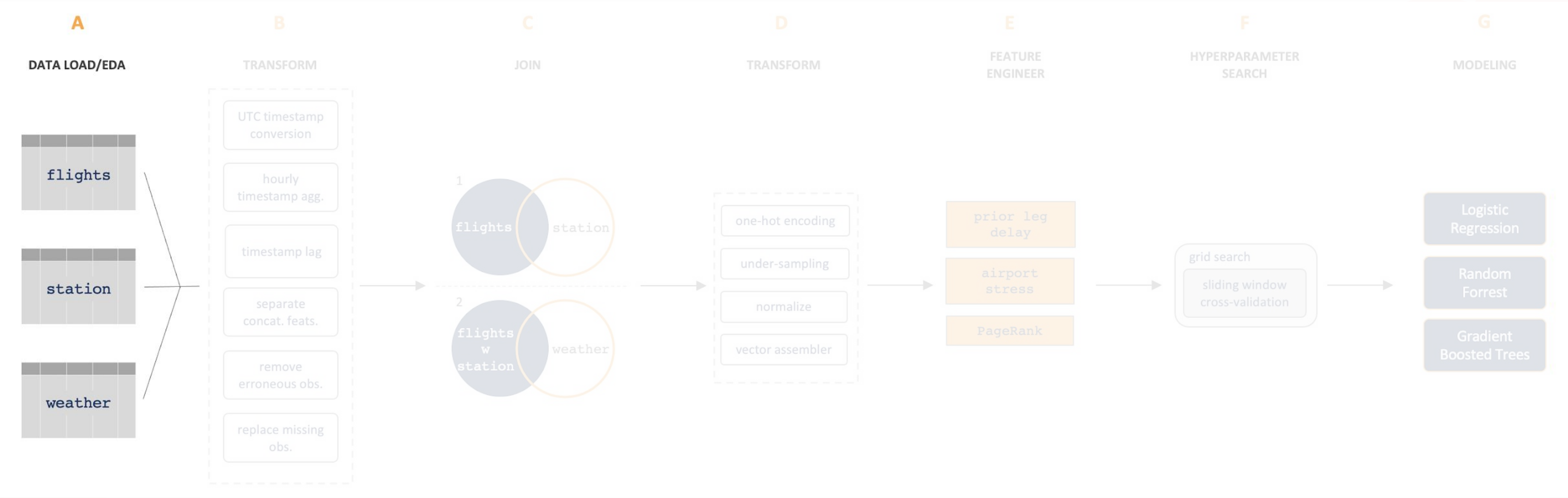
**Airlines**

Create a model for stakeholder to predict flight delays, when unavoidable

# Pipeline

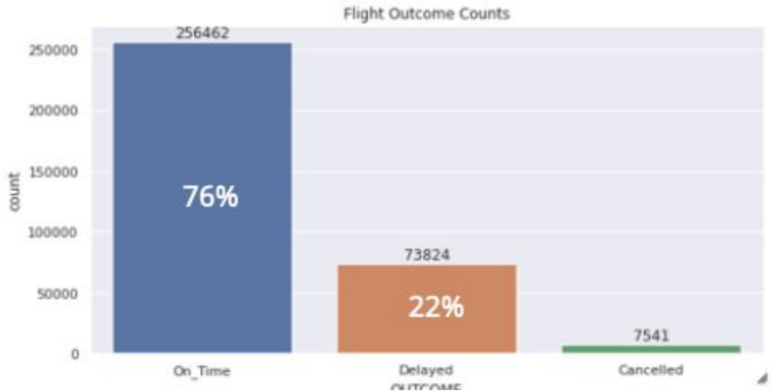

A
DATA LOAD/EDA
B
TRANSFORM
C
JOIN
D
TRANSFORM
E
FEATURE ENGINEER
F
HYPERPARAMETER SEARCH
G
MODELING
flights
station
weather
UTC timestamp conversion
hourly timestamp agg.
timestamp lag
separate concat. feats.
remove erroneous obs.
replace missing obs.
1
flights
station
2
flights w station
weather
one-hot encoding
under-sampling
normalize
vector assembler
prior leg delay
airport stress
PageRank
grid search
sliding window cross-validation
Logistic Regression
Random Forrest
Gradient Boosted Trees

# Pipeline - Data Load/EDA

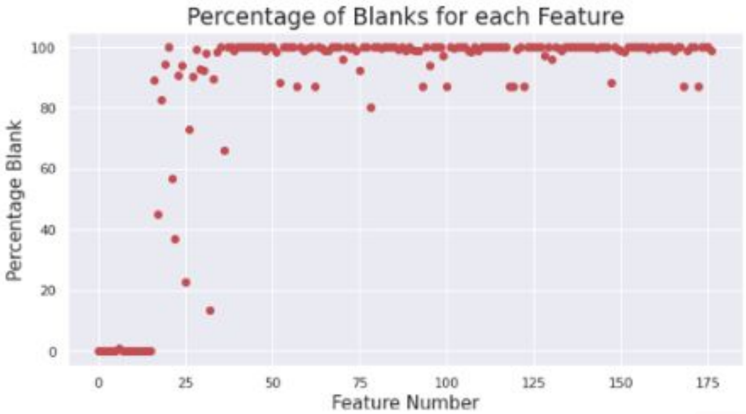


# Pipeline - Data Load/EDA

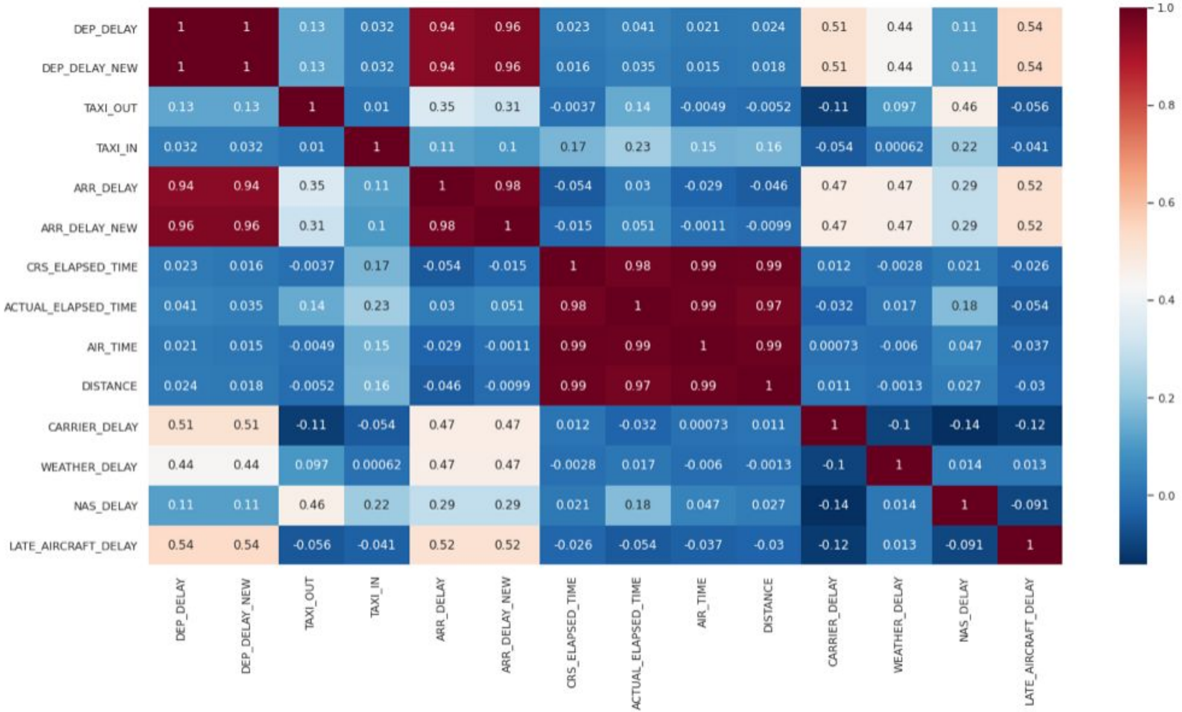
## CLASS IMBALANCE



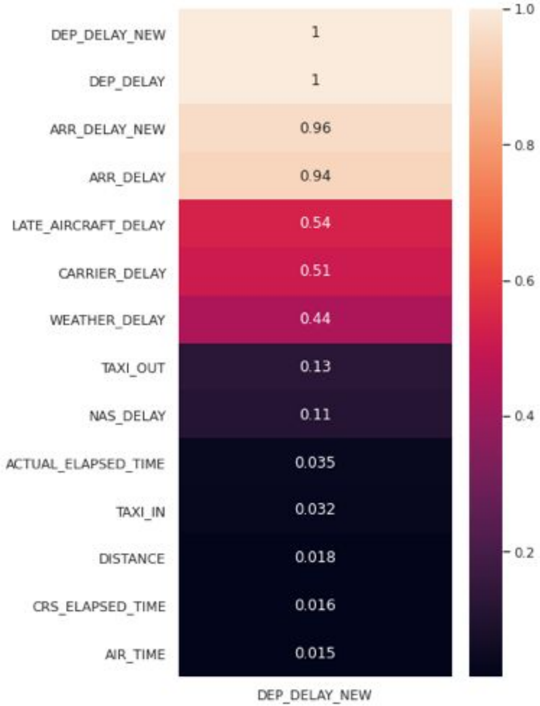

## NULL DATA



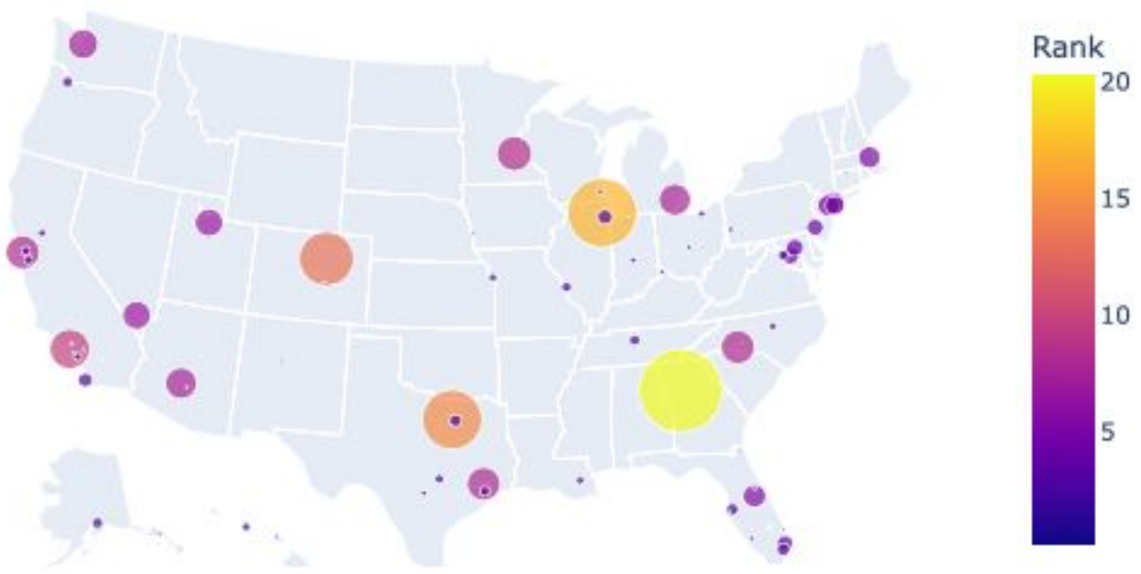
# Pipeline - Data Load/EDA



# Pipeline - Data Load/EDA



# Pipeline - Data Load/EDA



# Pipeline - Transform I

# Pipeline - Transform I

| | | Airlinine | | | | Weather | |
|---|---|---|---|---|---|---|---|
| Case | Strategy | UTC Time | "Hourly" UTC Time | Hour Lag | "Hourly" UTC Time (Lagged) | Worst Case UTC Time | Difference in Actual UTC Times |
| 1 | Round-down | 1:59 PM<br>1:01 PM | 1:00 PM | 2 | 11:00 AM | 11:59 AM | **2 hr 0 min**<br>**1hr 2 min** |
| 2 | Round-down | 1:59 PM<br>1:01 PM | 1:00 PM | 3 | 10:00 AM | 10:59 AM | **3 hr 0 min**<br>**2 hr 2 min** |

# Pipeline - Transform I + II

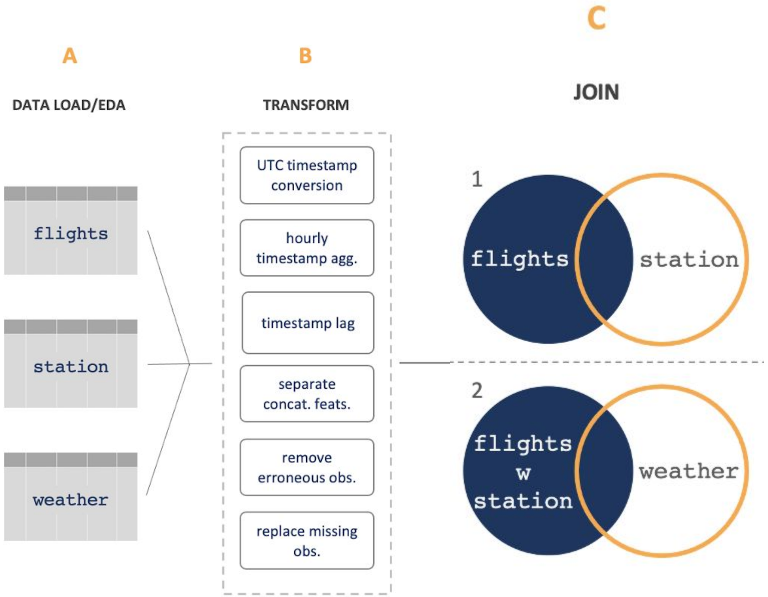
## TRANSFORMATION

- UTC timestamp conversion
- **Hourly timestamp aggregation**
- **Timestamp lag**
- Cross-map to fill `station_id` missing values
- Parse concatenated features
- Remove erroneous observations
- **Replace missing observations**
    - *Seven-day look back*
- **One-hot encoding**
- **Undersampling**
- Scaling continuous features

## ENGINEERED FEATURES

- Prior leg departure/arrival delay
- Airport stress @ departure/arrival time
- PageRank (airport traffic score)
- Dew & air temperature difference
- Vectorized feature list

# Pipeline - Join




Keys:
- Origin_ICAO
- Dest_ICAO

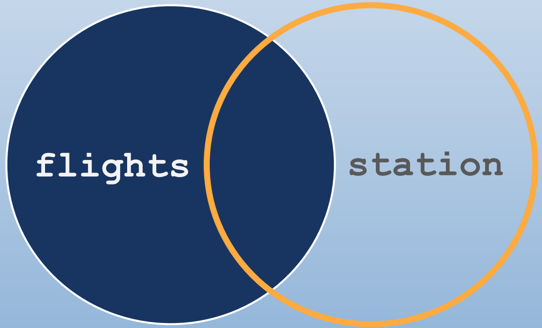
Added Features:
- Org_weather_station
- dest_weather_station

Keys:
- Org_weather_station
- Dest_weather_station
- Org_Dep_UTC_lag

Added Features:
- Assorted weather features, Dest and Org.

# Data - Join datasets

### JOIN 1:
flights_w_station




Keys:
- Origin_ICAO
- Dest_ICAO

Added Features:
- Org_weather_station
- dest_weather_station

### JOIN 2:
flights_w_weather

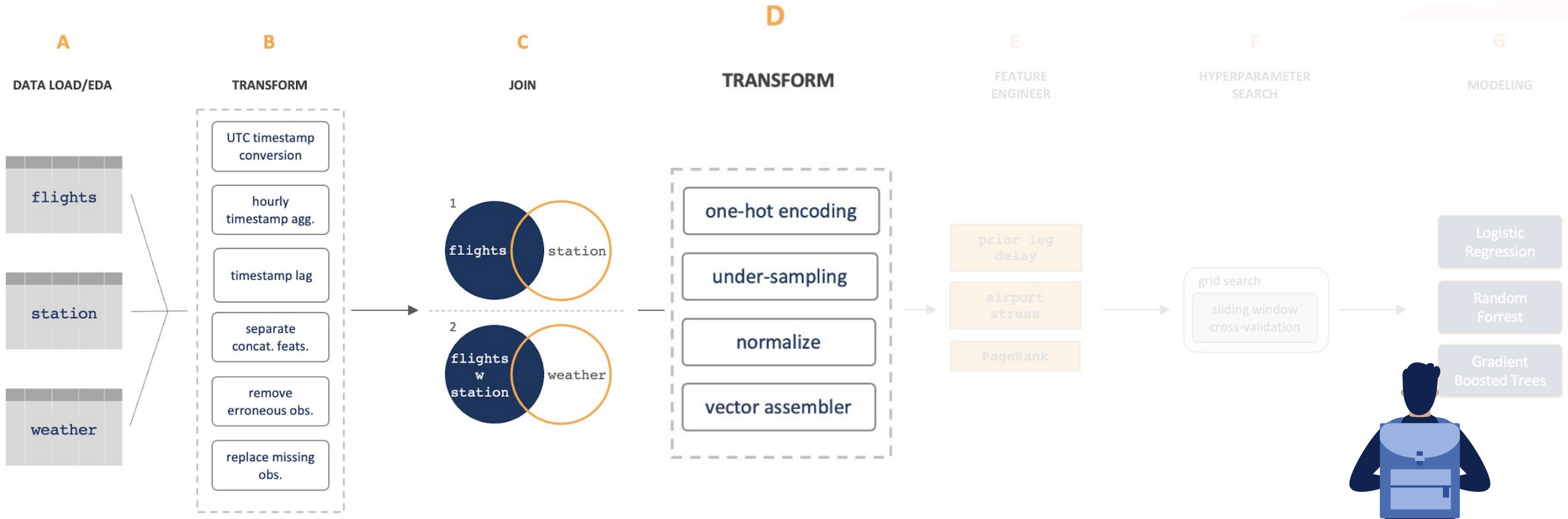

Keys:
- Org_weather_station
- Dest_weather_station
- Org_Dep_UTC_lag

Added Features:
- Assorted weather features, Dest and Org.

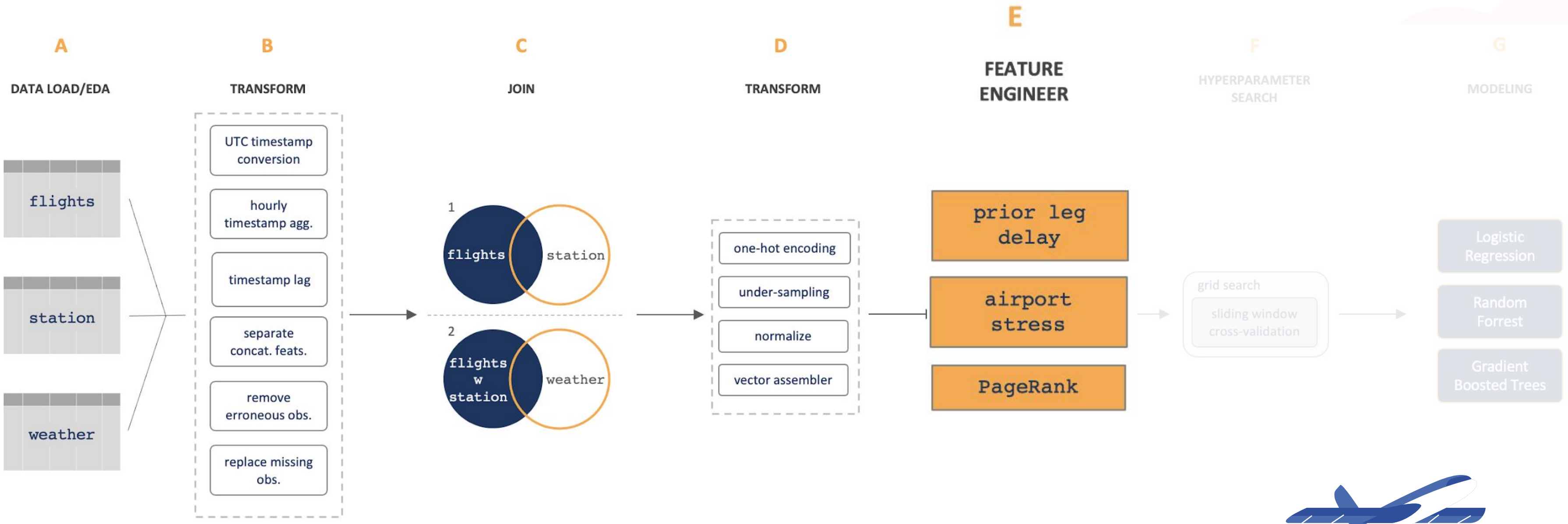
# Data - Join datasets

```
# Second, we need to create a new timestamp to join weather on - which is lagged by 2 hours.
# Due to the nature of our timestamps - we will actually have to subtract 3 hours to avoid leakage. 10800 is equivalent to 3 hours
flights_w_stations = flights_w_stations.withColumn('CRS_DEP_TIME_UTC_LAG', to_timestamp(f.col('CRS_DEP_TIME_UTC_HOUR').cast('long')
- 10800))

# Next, prepend an origin or destination prefix to the weather columns, so when we join we know which weather set we're looking at
origin_weather = weather.select([f.col(weather_feat).alias('ORIGIN_WEATHER_'+weather_feat) for weather_feat in weather.columns])
dest_weather = weather.select([f.col(weather_feat).alias('DEST_WEATHER_'+weather_feat) for weather_feat in weather.columns])

# join flights to ORIGIN weather on station_id and our lagged ORIGIN time variable
flights_w_weather_temp = flights_w_stations.join(origin_weather, (flights_w_stations.ORIGIN_WEATHER_STATION_ID ==
origin_weather.ORIGIN_WEATHER_STATION) & (flights_w_stations.CRS_DEP_TIME_UTC_LAG == origin_weather.ORIGIN_WEATHER_HOUR), 'left')

# Finally, join flights to DESTINATION weather on station_id and lagged ORIGIN time variable
flights_w_weather = flights_w_weather_temp.join(dest_weather, (flights_w_weather_temp.DEST_WEATHER_STATION_ID ==
dest_weather.DEST_WEATHER_STATION) & (flights_w_weather_temp.CRS_DEP_TIME_UTC_LAG == dest_weather.DEST_WEATHER_HOUR), 'left')
```

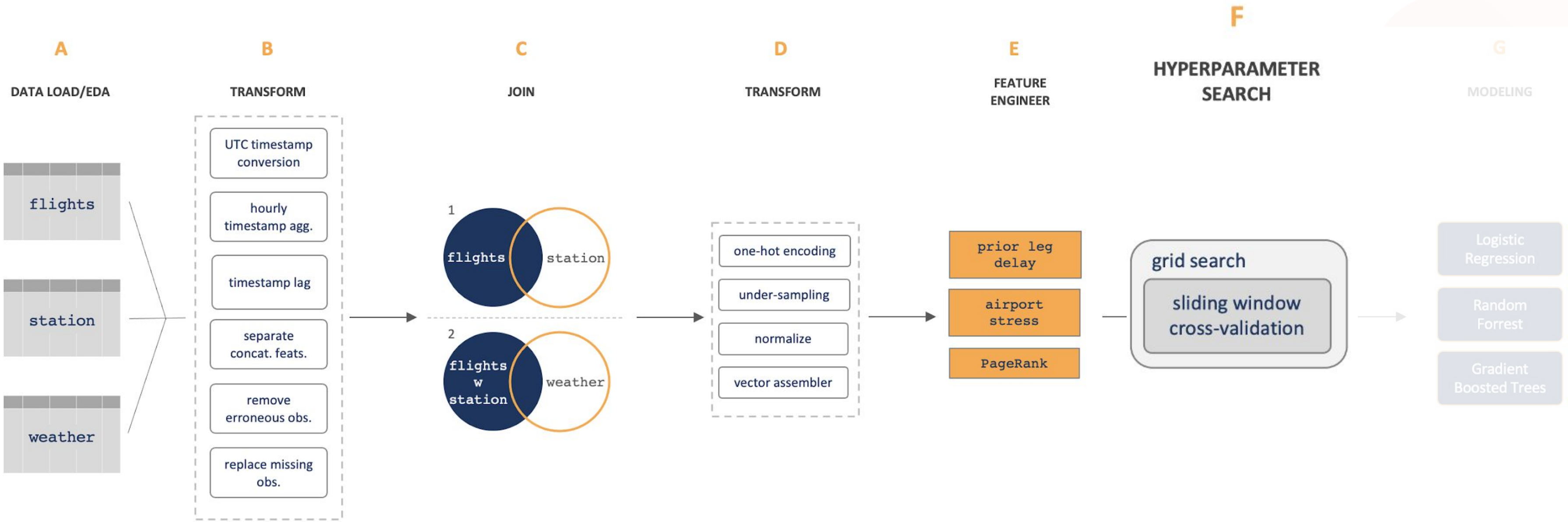
# Pipeline - Transform II



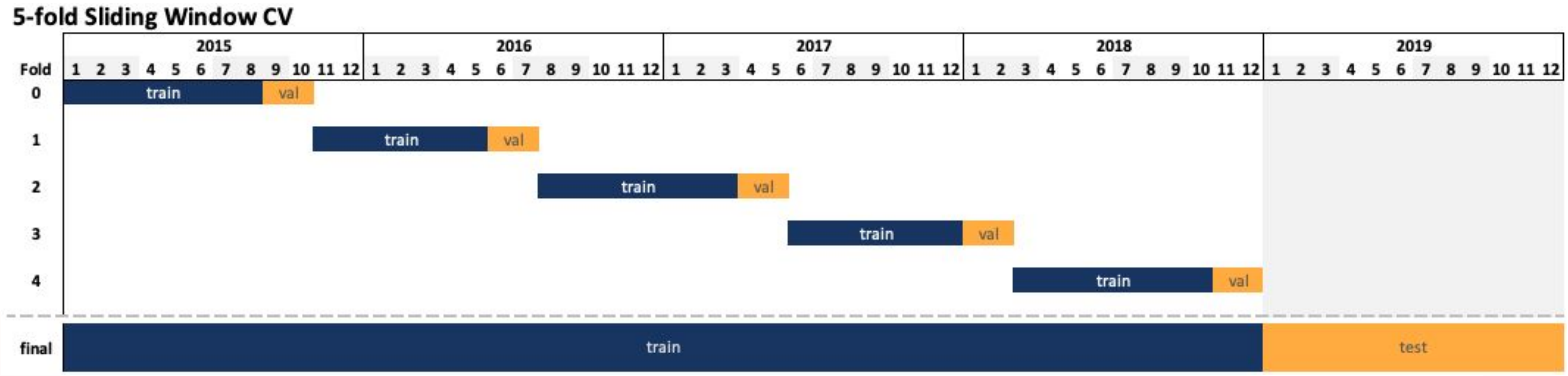
# Pipeline - Feature Engineering



# Pipeline - Hyperparameter Search



# Pipeline - Hyperparameter Search




Model

# Model - CV and Grid Search

```
def cv_folds(self):

        # Create simple folds by full train year, val year
        if self.simple_splits is True:
            for year in [2015, 2016, 2017]:

                train_df = self.df.where(f.col('CRS_DEP_TIME_UTC_HOUR') == year)
                val_df = self.df.where(f.col('CRS_DEP_TIME_UTC_HOUR') == (year + 1))

                self.train_df.append(train_df)
                self.test_df.append(val_df)

        # Create folds by number of folds, train/test split %
        else:
            data_size = self.df.count()

            for k in range(self.folds):

                 # Calculate total size of each fold
                fold_size = data_size/self.folds

                # Find range of `SPLIT_ID` for train and val
                train_ids = ((fold_size * k) + 0,
                             int(fold_size * k + fold_size*self.train_percent))

                val_ids = (train_ids[1] + 1,
                           fold_size * k + fold_size)

                # Split the data
                train_df = self.df.where(f.col('SPLIT_ID').between(train_ids[0], train_ids[1]))
                val_df = self.df.where(f.col('SPLIT_ID').between(val_ids[0], val_ids[1]))

                # store data
                train_df = DataSplit.class_balance(train_df, 'undersample')
                self.train_df.append(train_df)
                self.test_df.append(val_df)

         return train_dfs, val_dfs
```

# Model - CV and Grid Search

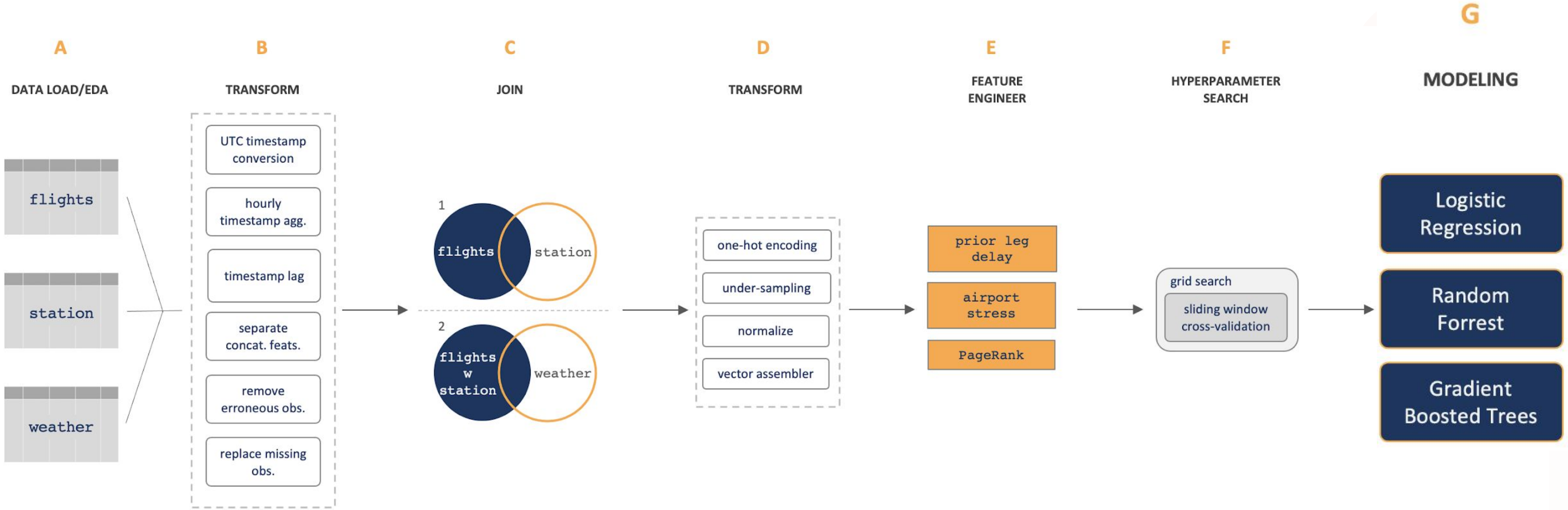
```
def gridsearch(full_data, k_folds, param_list, param_names, random_shuffle_top_N=10, threshold_plot=False, model_type='lr'):

    # Assemble train/test from cv_folds
    model = Model(df=full_data, folds = k_folds, train_percent = .8, simple_splits = False)
    model.cv_folds()


    # Assemble Gridsearch
    params = list(itertools.product(*param_list))
    random.shuffle(params)

    # Run Models in folds
    for param in params[:random_shuffle_top_N]:

        print(f"\nLooping through param:{list(zip(param_names,param))}")
        if model_type == 'lr':
            model.get_logistic_regression(**{'maxIter':param[0],'elasticNetParam':param[1],'regParam':param[2]})
        if model_type == 'rf':
            model.get_random_forest(**{'numTrees':param[0],'maxDepth':param[1],'maxBins':param[2]})
        if model_type == 'gbt':
            model.get_GBT(**{'maxIter':param[0],'maxDepth':param[1],'maxBins':param[2],'stepSize':param[3]})

    print(f'GridSearch\nBest Valid f0.5 Score: {model.best_score:.3f}\nBest Parameters: {model.best_params}')

    if threshold_plot:
        preds_valid_PR = model.tune_threshold()
        model.threshold_plot(preds_valid_PR)
```

# Model - CV and Grid Search

```python
class Model(DataSplit):

#     def __init__(self,train_df, test_df, label_col=['DEP_DEL15'],feature_col=['ALL_FEATURES_VA'],
**kwargs):
    def __init__(self, df,label_col=['DEP_DEL15'],feature_col=['ALL_FEATURES_VA'], train_percent=0.8,
timestamp_col='CRS_DEP_TIME_UTC', folds = 1, simple_splits = False):

        super().__init__(df, train_percent,timestamp_col, folds,simple_splits)
        self.label_col = label_col
        self.feature_col = feature_col
        self.model = None
        self.best_score = 0
        self.best_params = None
        self.best_preds_test = None

    def get_logistic_regression(self,**kwargs):
        model_args = kwargs
        lr = LogisticRegression(labelCol = self.label_col[0],
                                featuresCol=self.feature_col[0],
                                elasticNetParam = model_args.get('elasticNetParam',1),
                                standardization = model_args.get('standardization',True),
                                maxIter=model_args.get('maxIter',10),
                                regParam=model_args.get('regParam',0.001))

        for index in range(len(self.train_df)):
            print(f'\n{index}-Fold:\n')
            self.model = lr.fit(self.train_df[index])
            self.get_predictions(index,model_args)

    def get_random_forest(self,**kwargs):
        model_args = kwargs
        rf = RandomForestClassifier(labelCol = self.label_col[0],
                                    featuresCol=self.feature_col[0],
                                    numTrees=model_args.get('numTrees',1),
                                    maxDepth=model_args.get('maxDepth',5),
                                    maxBins=model_args.get('maxBins',32))

        for index in range(len(self.train_df)):
            print(f'\n{index}-Fold:\n')
            self.model = rf.fit(self.train_df[index])
            self.get_predictions(index,model_args)
```

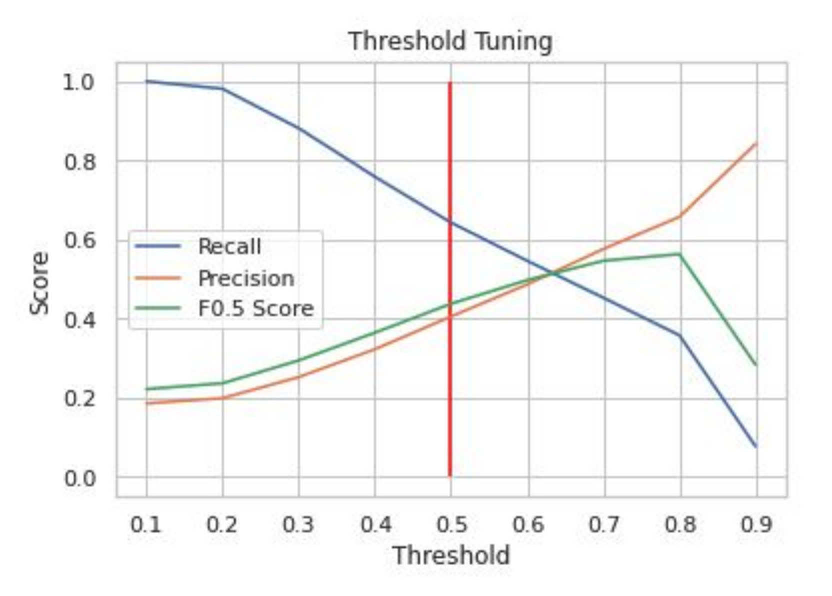
```python
    def get_GBT(self,**kwargs):
        model_args = kwargs
        GBT = GBTClassifier(labelCol = self.label_col[0],
                            featuresCol=self.feature_col[0],
                            maxIter=model_args.get('maxIter',1),
                            maxDepth=model_args.get('maxDepth',5),
                            maxBins=model_args.get('maxBins',32),
                            stepSize=model_args.get('stepSize',0.1))

        for index in range(len(self.train_df)):
            print(f'\n{index}-Fold:\n')
            self.model = GBT.fit(self.train_df[index])
            self.get_predictions(index,model_args)

def get_metrics(self,preds_train, preds_test,model_args):

        train_rdd = preds_train.select(['prediction', 'DEP_DEL15']).rdd
        train_metrics = MulticlassMetrics(train_rdd)
        valid_rdd = preds_test.select(['prediction', 'DEP_DEL15']).rdd
        valid_metrics = MulticlassMetrics(valid_rdd)
        valid_score = valid_metrics.fMeasure(1.0, 0.5)
        if valid_score > self.best_score:
            self.best_score = valid_score
            self.best_params = model_args
            self.best_preds_test = preds_test
        print(' \t\tTrain Metrics \t Validation Metrics\n')
        print(f'Recall: \t\t{train_metrics.recall(label=1):.3f} \t\t
{valid_metrics.recall(label=1):.3f}')
        print(f'Precision: \t\t{train_metrics.precision(1):.3f} \t\t
{valid_metrics.precision(1):.3f}')
        print(f'Accuracy: \t\t{train_metrics.accuracy:.3f} \t\t {valid_metrics.accuracy:.3f}')
        print(f'F0.5 score: \t\t{train_metrics.fMeasure(1.0, 0.5):.3f} \t\t
{valid_metrics.fMeasure(1.0, 0.5):.3f}')
        print(f'F2 score: \t\t{train_metrics.fMeasure(1.0, 2.0):.3f} \t\t {valid_metrics.fMeasure(1.0,
2.0):.3f}')
        print(f'F1 score: \t\t{train_metrics.fMeasure(1.0):.3f} \t\t
{valid_metrics.fMeasure(1.0):.3f}')
```

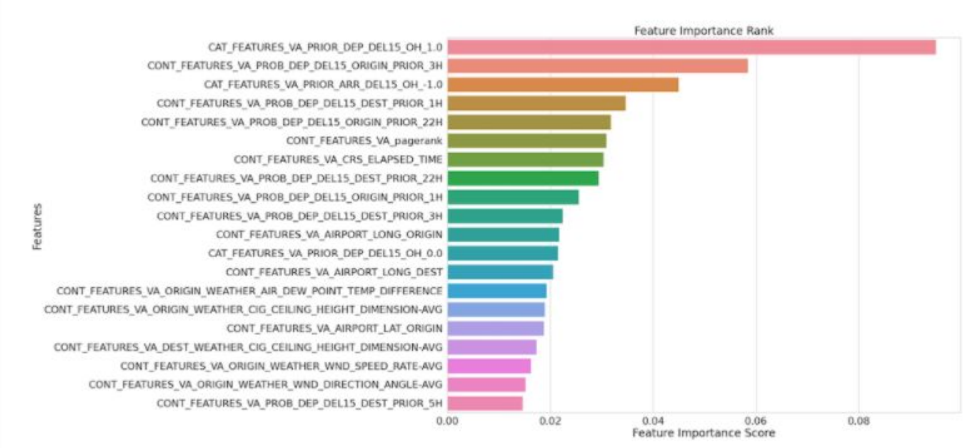
# Pipeline - Modeling



# Model Results

# Pipeline - Modeling



# Pipeline - Modeling

| Phase | Model # | Algorithm | F0.5 | CV Folds | GridSearch | Param Combos | RunTime | Change log | Train | Val | Test |
|---|---|---|---|---|---|---|---|---|---|---|---|
| II | Initial | LR | 0.049 | 1 | N | - | - | Did not account for class imbalance, regularization, OHE, VA | Jan - Sep 2015 | Oct - Dec 2015 | N/A |
| | **Baseline** | LR | **0.253** | 1 | N | - | - | Add undersampling (~20% sampled on on-time flights) | | | |
| | 1 | LR | 0.453* | 1 | N | - | - | Add regularization, OHE, VA | | | |
| | 2a | LR | 0.208* | 1 | N | - | - | Add undersampling | | | |
| | 2b | RF | 0.244 | 1 | N | - | - | | | | |
| III | 3 | LR | 0.284 | 1 | N | - | - | Change Train/Val | 2017 | 2018 | N/A |
| | **4a** | LR | **0.304** | 1 | N | - | - | **Add PRIOR_ARR_DEL15** | | | |
| | 4b | RF | 0.281 | 1 | N | - | - | | | | |
| | **5a** | LR | **0.384** | 1 | N | - | - | **Add PRIOR_DEP_DEL15** | | | |
| | 5b | RF | 0.362 | 1 | N | - | - | | | | |
| | 5c | GBT | 0.409 | 1 | N | - | - | | | | |
| | **6a** | LR | **0.667** | 2 | N | - | 0h 1m | Add CV, **incorrect** undersampling | 2017-2018 | | N/A |
| | **6b** | RF | **0.609** | 2 | N | - | 0h 25m | | | | |
| | **6c** | GBT | **0.684** | 2 | N | - | **12h 20m** | | | | |
| | 7a | LR | 0.4 | 2 | N | - | 0h 1m | Add CV, **PROB_DEP_DEL15_ORIGIN_PRIOR_1H** | | | |
| | 7b | RF | 0.37 | 2 | N | - | 0h 2m | | | | |
| | 7c | GBT | 0.438 | 2 | N | - | | | | | |
| | **8** | LR | **0.479** | **5** | **Y** | **27** | **3h 52m** | Expand CV, add gridsearch on max iter, alpha, lambda | | | |
| | **9a** | LR | **0.445** | **5** | **Y** | **27** | **6h 30m** | **Further Expand CV** | **2015-2018** | | N/A |
| | **9b** | LR | **0.477** | N/A | N/A | N/A | **0h 12m** | **First true Test \| iter = 64, alpha = .2, lambda = .01 \| P = .671** | | | **2019** |
| IV | **10a** | LR | **0.524** | N/A | N/A | 256 | **0h 14m** | iter = 64, alpha = .2, lambda = .01, threshold = .7<br>- Add airport stress probability variations | **2015-2018** | | **2019** |
| | **10b** | RF | **0.373** | N/A | N/A | 27 | **0h 24m** | num_trees = 128, max_depth = 10, max_bins = 32<br>- Add pagerank | | | |
| | **10c** | GBT | **0.563** | N/A | N/A | 3125 | **1h 5m** | iter = 6, max_depth = 10, max_bins = 256, step_size = .2, threshold = .77<br>- Add weather dew and air temperature ')<br>All hyperparam combos found through prior gridsearch | | | |

# Performance

## Model Performance

**Logistic Regression**

| | | Predicted 0 | Predicted 1 |
|---|---|---|---|
| Actual | 0 | 0.93 | 0.07 |
| | 1 | 0.60 | 0.40 |

F0.5: **0.524**

**Random Forest**

| | | Predicted 0 | Predicted 1 |
|---|---|---|---|
| Actual | 0 | 0.69 | 0.31 |
| | 1 | 0.32 | 0.68 |

F0.5: **0.373**

**Gradient Boosted Trees**

| | | Predicted 0 | Predicted 1 |
|---|---|---|---|
| Actual | 0 | 0.95 | 0.05 |
| | 1 | 0.61 | 0.39 |

F0.5: **0.563**

## Pipeline Performance

- 80GB - 20 Cores
- Join: **5m**
- Caching: **5-10m** x checkpoint
- Total Join pipeline: **1hr**

# Gap Analysis & Future Work

- Expanded Domain Knowledge
  - Aide in best feature selection/creation
- Augment with additional data sources
- Algorithmic Feature Selection/Dimensionality Reduction
  - Are we providing the model with the best information?
- Additional EDA to better inform Transformations
- Seasonality

# Limitations, Challenges & Future Work

- Limitations/Challenges
  - Would like to spend more time creating/tuning engineered features
  - F0.5 2x baseline score
    - Still low after all feat eng/transformations (.563)
  - Addressing Seasonality
  - No GPU = No Neural Nets
- Future Work
  - Additional EDA to inform specific transformations
  - Interview with industry experts to gain domain knowledge
  - Address seasonality


Questions?