(https://databricks.com)

Imports and Setup

Show code

```
Welcome to the W261 final project!
```

Read Delta Lake dataset into Dataframes

```
# The following blob storage is accessible to team members only (read and write)
# access key is valid til TTL
# after that you will need to create a new SAS key and authenticate access again via DataBrick command line
blob_container  = "w261storage"       # The name of your container created in https://portal.azure.com
storage_account = "w261rtang"  # The name of your Storage account created in https://portal.azure.com
secret_scope    = "team_2_1_scope"           # The name of the scope created in your local computer using the Databricks CLI
secret_key      = "team_2_1_key"             # The name of the secret key created in your local computer using the Databricks CLI
team_blob_url   = f"wasbs://{blob_container}@{storage_account}.blob.core.windows.net"  #points to the root of your team storage bucket

# the 261 course blob storage is mounted here.
mids261_mount_path      = "/mnt/mids-w261"

# SAS Token: Grant the team limited access to Azure Storage resources
spark.conf.set(
  f"fs.azure.sas.{blob_container}.{storage_account}.blob.core.windows.net",
  dbutils.secrets.get(scope = secret_scope, key = secret_key)
)

## Configure Path
DELTALAKE_OTPW_3M_2015 = f'{team_blob_url}/OTPW_3M_2015.parquet'
DELTALAKE_OTPW_3M = f'{team_blob_url}/OTPW_3M.parquet'
DELTALAKE_OTPW_12M = f'{team_blob_url}/OTPW_12M.parquet'
DELTALAKE_OTPW_36M = f'{team_blob_url}/OTPW_36M.parquet'
DELTALAKE_OTPW_60M = f'{team_blob_url}/OTPW_60M.parquet'
NEW_FEATURE_36M = f'{team_blob_url}/FeatEng_ALL_3yr_with_cancelled.deltalake'
NEW_FEATURE_60M = f'{team_blob_url}/FeatEng_ALL_5yr_w_Holiday_LagFeat.deltalake'

## Re-read as Delta Lake
delta_otpw_3m_2015 = spark.read.format("delta").load(DELTALAKE_OTPW_3M_2015).cache()
delta_otpw_3m = spark.read.format("delta").load(DELTALAKE_OTPW_3M).cache()
delta_otpw_1yr = spark.read.format("delta").load(DELTALAKE_OTPW_12M).cache()
delta_otpw_3yr = spark.read.format("delta").load(DELTALAKE_OTPW_36M).cache()
delta_otpw_5yr = spark.read.format("delta").load(DELTALAKE_OTPW_60M).cache()
delta_new_feature_3yr = spark.read.format("delta").load(NEW_FEATURE_36M).cache()
delta_new_feature_5yr = spark.read.format("delta").load(NEW_FEATURE_60M).cache()
```

Work with these set of features after EDA

Show code

2. Start ML Pipeline

- Data Imputation
- Stand Scalar
- String Indexing
- One Hot Encoding
- Logistic Regression

TimeSeriesCrossValidator - Custom CrossValidator class

Show code

```
!pip install --no-deps sparkxgb
```

```
Collecting sparkxgb
  Downloading sparkxgb-0.1.tar.gz (3.6 kB)
  Preparing metadata (setup.py) ... done
Building wheels for collected packages: sparkxgb
  Building wheel for sparkxgb (setup.py) ... done
  Created wheel for sparkxgb: filename=sparkxgb-0.1-py3-none-any.whl size=5630 sha256=69492b4bdbd36d6996aa785f981574daa37368bc43177190e37f7ed7d4af48be
  Stored in directory: /root/.cache/pip/wheels/b7/0c/a1/786408e13056fabeb8a72134e101b1e142fc95905c7b0e2a71
Successfully built sparkxgb
Installing collected packages: sparkxgb
Successfully installed sparkxgb-0.1

[notice] A new release of pip available: 22.2.2 -> 23.3.1
[notice] To update, run: pip install --upgrade pip
```

```
!pip install --upgrade pip
!pip install sparkxgb
```

```
Requirement already satisfied: pip in /local_disk0/.ephemeral_nfs/envs/pythonEnv-e702cde6-f342-4ead-91dc-05bf759052a3/lib/python3.10/site-packages (22.2.2)
Collecting pip
  Downloading pip-23.3.1-py3-none-any.whl (2.1 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.1/2.1 MB 12.7 MB/s eta 0:00:00a 0:00:01
Installing collected packages: pip
  Attempting uninstall: pip
    Found existing installation: pip 22.2.2
    Uninstalling pip-22.2.2:
      Successfully uninstalled pip-22.2.2
Successfully installed pip-23.3.1
Requirement already satisfied: sparkxgb in /local_disk0/.ephemeral_nfs/envs/pythonEnv-e702cde6-f342-4ead-91dc-05bf759052a3/lib/python3.10/site-packages (0.1)
```

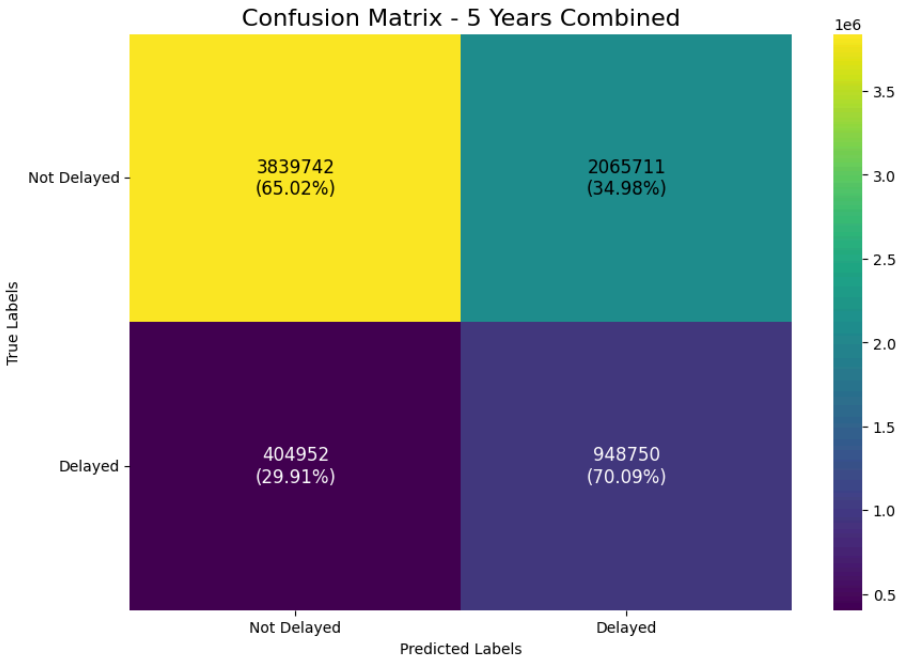
```
Collecting pyspark==3.1.1 (from sparkxgb)
  Downloading pyspark-3.1.1.tar.gz (212.3 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 212.3/212.3 MB 6.4 MB/s eta 0:00:0000:0100:01
  Preparing metadata (setup.py) ... done
Collecting py4j==0.10.9 (from pyspark==3.1.1->sparkxgb)
  Downloading py4j-0.10.9-py2.py3-none-any.whl (198 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 198.6/198.6 kB 14.1 MB/s eta 0:00:00
Building wheels for collected packages: pyspark
  Building wheel for pyspark (setup.py) ... done
```

XGBoost with Bayesian Optimization

```
from sparkxgb import XGBoostClassifier
from hyperopt import fmin, tpe, hp, STATUS_OK, Trials
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
from pyspark.sql.types import DoubleType
from pyspark.sql import SparkSession
from pyspark.ml import Pipeline
from pyspark.ml.feature import Imputer, StandardScaler, StringIndexer, VectorAssembler
from pyspark.ml.classification import MultilayerPerceptronClassifier
import pandas as pd
from sklearn.metrics import precision_score, recall_score, f1_score, roc_auc_score
from sklearn.metrics import fbeta_score
from pyspark.mllib.evaluation import BinaryClassificationMetrics
import logging
from xgboost import XGBClassifier, callback

# ============================================================================
#     5-YEAR Data Load
# ============================================================================

DELTALAKE_CLEANED_TRAIN = f'{team_blob_url}/UNDERSAMPLE_4yr_TRAIN_SET_LAG_FEATURE_60M.deltalake'      # Training: 2015-2018 (Undersampled)
# DELTALAKE_CLEANED_VAL = f'{team_blob_url}/VAL_SET_2018_LAG_FEATURE_60M.deltalake'                   # Validation: 2018
DELTALAKE_CLEANED_TEST = f'{team_blob_url}/TEST_SET_LAG_FEATURE_60M.deltalake'                       # Testing: 2019
# UNDERSAMPLE_4yr_TRAIN_SET_LAG_FEATURE_60M.deltalake

df_split_train = spark.read.format("delta").load(DELTALAKE_CLEANED_TRAIN).cache()
# df_split_val = spark.read.format("delta").load(DELTALAKE_CLEANED_VAL).cache()
df_split_test = spark.read.format("delta").load(DELTALAKE_CLEANED_TEST).cache()

# ============================================================================
#     Define which features to perform which types of spark transformation
# ============================================================================

# ####### EXISTIN FEATURE SETUP #######

imputer_feature_cols = ['HourlyPrecipitation', 'HourlyRelativeHumidity', 'HourlyVisibility', 'HourlyWindSpeed']
scaler_feature_cols = ['CRS_DEP_TIME', 'DISTANCE', 'ELEVATION', 'DAY_OF_WEEK_sin', 'DAY_OF_WEEK_cos', 'CRS_DEP_HOUR_sin', 'CRS_DEP_HOUR_cos', 'DAY_HOUR_interaction', 'TIME_INTERVAL_OF_DAY']
indexer_feature_cols = ['DAY_OF_MONTH','MONTH','OP_UNIQUE_CARRIER','ORIGIN','DEST','TAIL_NUM','CARRIER_SIZE','DAY_TYPE', 'DEGREE_VISIBILITY', 'FL_DISTANCE_GROUP', '5_DAYS_DIST_FROM_Independe
'7_DAYS_DIST_FROM_Christmas', '7_DAYS_DIST_FROM_NewYear','DIVERTED','dep_del15_2hr_before']
one_hot_encode_cols = ['DIVERTED','dep_del15_2hr_before']

# Define the stages for the pipeline

# Impute missing values
imputer = Imputer(inputCols=imputer_feature_cols,
                  outputCols=['imputed_' + col for col in imputer_feature_cols])

# Apply StandardScaler
vector_assembler_scaler = VectorAssembler(inputCols=scaler_feature_cols, outputCol='scaled_features')
scaler = StandardScaler(inputCol='scaled_features', outputCol='scaled_features_final', withStd=True, withMean=True)

# Apply StringIndexer
indexer = [StringIndexer(inputCol=col, outputCol='indexed_' + col, handleInvalid='keep') for col in indexer_feature_cols]

# Apply OneHotEncoder
one_hot_encoder = [OneHotEncoder(inputCol='indexed_' + col, outputCol='onehot_' + col, handleInvalid='keep') for col in one_hot_encode_cols]

# Assemble features for each encoder
vector_assembler_imputer = VectorAssembler(inputCols=['imputed_' + col for col in imputer_feature_cols], outputCol='imputed_features')
vector_assembler_indexer = VectorAssembler(inputCols=['indexed_' + col for col in indexer_feature_cols], outputCol='indexed_features')
vector_assembler_oher = VectorAssembler(inputCols=['onehot_' + col for col in one_hot_encode_cols], outputCol='onehot_features')

# Combine all feature columns
regular_cols = ['DIVERTED', 'YEAR', 'DAY_OF_WEEK_sin', 'DAY_OF_WEEK_cos', 'CRS_DEP_HOUR_sin', 'CRS_DEP_HOUR_cos', 'DAY_HOUR_interaction', 'dep_del15_2hr_before']
all_feature_cols = ['imputed_features', 'scaled_features_final', 'indexed_features', 'onehot_features'] + regular_cols

# Assemble all features into a single vector
vector_assembler_final = VectorAssembler(inputCols=all_feature_cols, outputCol='features', handleInvalid='skip')

# ============================================================================
#     BUILD XGBoost with Bayesian Optimization Hyperparameter Tuning
# ============================================================================

space = {
    'eta': hp.choice('eta', [0.01, 0.05, 0.1]),
    'alpha': hp.choice('alpha', [0.1, 0.3, 0.7, 1.0]),
    'maxDepth': hp.choice('maxDepth', [6, 7, 8, 9,10]),
    'numRound': hp.choice('numRound', [100, 150, 200, 250, 300]),
}

def objective(hyperparams, df_split_train, df_split_test):
  # Build the model with the given hyperparams
  xgb = XGBoostClassifier(
                eta=hyperparams['eta'],
                maxDepth=int(hyperparams['maxDepth']),
                numRound=int(hyperparams['numRound']),
                alpha=hyperparams['alpha'],
                numEarlyStoppingRounds=100,
                objective='binary:logistic',
                featuresCol='features',
                labelCol='label',
                missing=0.0,
                seed=1234
            )

  hyperparams['maxDepth'] = int(hyperparams['maxDepth'])
  hyperparams['numRound'] = int(hyperparams['numRound'])

  # ============================================================================
  #     BUILD Pipeline with 5 Fold Cross-Validation
```

```
    # ========================================================================

    paramGrid = ParamGridBuilder().build()

    # Build our ML pipeline
    pipeline = Pipeline(stages=[imputer, vector_assembler_imputer, vector_assembler_scaler, scaler] + indexer + one_hot_encoder +
                                [vector_assembler_indexer, vector_assembler_oher, vector_assembler_final, xgb])

    evaluator = MulticlassClassificationEvaluator(metricName='fMeasureByLabel', beta=2.0, labelCol='label',  predictionCol='prediction', metricLabel=1)

    tscv = TimeSeriesCrossValidator(estimator=pipeline,
                                estimatorParamMaps=paramGrid,
                                evaluator=evaluator,
                                numFolds=5,
                                collectSubModels=True)  # use 3+ folds in practice

    # Fit the pipeline including time series cross validator on training data. Run cross-validation, and choose the best set of parameters.
    tsCvModel = tscv.fit(df_split_train)

# ========================================================================
#     EVALUATION ON TRAIN AND HELD-OUT TEST
# ========================================================================

    evaluator = MulticlassClassificationEvaluator().setPredictionCol("prediction")

    train_predictions = tsCvModel.transform(df_split_train).cache()
    trainScoreAndLabels = train_predictions.select(F.col('prediction'), F.col('label')).cache()
    train_f125_score = evaluator.evaluate(trainScoreAndLabels, {evaluator.metricName: "fMeasureByLabel", evaluator.beta: 1.25})
    train_f15_score = evaluator.evaluate(trainScoreAndLabels, {evaluator.metricName: "fMeasureByLabel", evaluator.beta: 1.5})
    train_f2_score = evaluator.evaluate(trainScoreAndLabels, {evaluator.metricName: "fMeasureByLabel", evaluator.beta: 2.0})
    train_precision =evaluator.evaluate(trainScoreAndLabels, {evaluator.metricName: "precisionByLabel"})
    train_recall = evaluator.evaluate(trainScoreAndLabels, {evaluator.metricName: "recallByLabel"})
    train_tpr =evaluator.evaluate(trainScoreAndLabels, {evaluator.metricName: "truePositiveRateByLabel"})
    train_fpr =evaluator.evaluate(trainScoreAndLabels, {evaluator.metricName: "falsePositiveRateByLabel"})

    # Evaluate best model on held-out validation/test
    test_predictions = tsCvModel.transform(df_split_test).cache()
    testScoreAndLabels = test_predictions.select(F.col('prediction'), F.col("label")).cache()
    test_f125_score = evaluator.evaluate(testScoreAndLabels, {evaluator.metricName: "fMeasureByLabel", evaluator.beta: 1.25})
    test_f15_score = evaluator.evaluate(testScoreAndLabels, {evaluator.metricName: "fMeasureByLabel", evaluator.beta: 1.5})
    test_f2_score = evaluator.evaluate(testScoreAndLabels, {evaluator.metricName: "fMeasureByLabel", evaluator.beta: 2.0})
    test_precision =evaluator.evaluate(testScoreAndLabels, {evaluator.metricName: "precisionByLabel"})
    test_recall = evaluator.evaluate(testScoreAndLabels, {evaluator.metricName: "recallByLabel"})
    test_tpr =evaluator.evaluate(testScoreAndLabels, {evaluator.metricName: "truePositiveRateByLabel"})
    test_fpr =evaluator.evaluate(testScoreAndLabels, {evaluator.metricName: "falsePositiveRateByLabel"})

    # Track train metrics for current iteration in ML Flow
    mlflow.log_metric('train_f125_score', train_f125_score)
    mlflow.log_metric('train_f15_score', train_f15_score)
    mlflow.log_metric('train_f2_score', train_f2_score)
    mlflow.log_metric('train_precision', train_precision)
    mlflow.log_metric('train_recall', train_recall)
    mlflow.log_metric('train_tpr', train_tpr)
    mlflow.log_metric('train_fpr', train_fpr)

    # Track test metrics for current iteration in ML Flow
    mlflow.log_metric('test_f125_score', test_f125_score)
    mlflow.log_metric('test_f15_score', test_f15_score)
    mlflow.log_metric('test_f2_score', test_f2_score)
    mlflow.log_metric('test_precision', test_precision)
    mlflow.log_metric('test_recall', test_recall)
    mlflow.log_metric('test_tpr', test_tpr)
    mlflow.log_metric('test_fpr', test_fpr)

    # Convert to RDD to use with MulticlassMetrics
    predictionAndLabels = test_predictions.select(['prediction', 'label']).rdd.map(lambda r: (float(r['prediction']), float(r['label'])))

    metrics = MulticlassMetrics(predictionAndLabels)
    precision = metrics.precision(label=1.0)
    recall = metrics.recall(label=1.0)
    f2_score = evaluator.evaluate(test_predictions, {evaluator.metricLabel: 1.0})  # 1.0 = delayed

    # ========================================================================
    #     Plot Confusion Matrix
    # ========================================================================

    conf_matrix = metrics.confusionMatrix().toArray()

    # Calculate the percentage for each cell
    conf_matrix_percentages = conf_matrix / np.sum(conf_matrix, axis=1, keepdims=True)

    # Plot the heatmap
    plt.figure(figsize=(10, 7))
    sns.heatmap(conf_matrix, annot=False, fmt='', cmap='viridis')  # Turn off default annotations

    # Annotate with the count and percentage
    for i in range(conf_matrix.shape[0]):
        for j in range(conf_matrix.shape[1]):
            count = int(conf_matrix[i, j])
            percentage = conf_matrix_percentages[i, j]
            plt.text(j+0.5, i+0.5, f"{count}\n({percentage:.2%})",
                    ha='center', va='center',
                    color='black' if conf_matrix[i, j] > conf_matrix.max() / 2 else 'white',
                    fontsize=12 if count != 0 else 10)  # Increase fontsize for count

    plt.xlabel('Predicted Labels')
    plt.ylabel('True Labels')
    plt.title('Confusion Matrix - 5 Years Combined', fontsize=16)
    plt.xticks([0.5, 1.5], ['Not Delayed', 'Delayed'])
```

```
    plt.yticks([0.5, 1.5], ['Not Delayed', 'Delayed'], rotation=0)
    plt.show()

    # return {'loss': -f2_score, 'status': STATUS_OK}
    return {'loss': -f2_score,
          'status': STATUS_OK,
          'train_f125_score':train_f125_score,
          'train_f15_score': train_f15_score,
          'train_f2_score': train_f2_score,
          'train_precision': train_precision,
          'train_recall': train_recall,
          'train_tpr': train_tpr,
          'train_fpr': train_fpr,
          'test_f125_score':test_f125_score,
          'test_f15_score': test_f15_score,
          'test_f2_score': test_f2_score,
          'test_precision': test_precision,
          'test_recall': test_recall,
          'test_tpr': test_tpr,
          'test_fpr': test_fpr}

fold: 0, start: 1, mid: 1379629, stop: 1724537
fold: 1, start: 1379630, mid: 2759258, stop: 3104166
fold: 2, start: 2759259, mid: 4138887, stop: 4483795
fold: 3, start: 4138888, mid: 5518516, stop: 5863424
fold: 4, start: 5518517, mid: 6898145, stop: 7243053
Here are all metrics per fold:
================================
{}
Detailed F2 Score Per Fold [0.6843581554665982, 0.6671492231102704, 0.6835301439619246, 0.5965751964602587, 0.5914421524623402]
Avg Score 0.6446109742922784

Here are the best model metrics with best hyperparameters:
Best Model:
{}
Detailed Score [0.6843581554665982, 0.6671492231102704, 0.6835301439619246, 0.5965751964602587, 0.5914421524623402]
Avg Score 0.6446109742922784
  0%|          | 0/1 [1:25:48<?, ?trial/s, best loss=?]
```




```
100%|██████████| 1/1 [3:26:12<00:00, 12372.88s/trial, best loss: -0.6965048121267188]

BEST PARAMETERS:  {'alpha': 0, 'eta': 1, 'maxDepth': 4, 'numRound': 4}

METRICS ON TRAIN
==================
train_f125_score: 0.6873999417786456
train_f15_score: 0.6818847742191576
train_f2_score: 0.6748216403497164
train_precision: 0.7310757319163437
train_recall: 0.6620852698986802
train_tpr: 0.6620852698986802
train_fpr: 0.24337475442624024

METRICS ON TEST
==================
test_f125_score: 0.730356661193835
test_f15_score: 0.7117948698643319
test_f2_score: 0.6889528956375084
test_precision: 0.904598069966881
test_recall: 0.650202787152823
```