```
# -*- coding: utf-8 -*-
"""CS777_FinalProject_Code.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1wNCZiaZr2d_vAxMj_T1vsE7SnLWCPuYq

# Introduction

Logistics problems are a rich source of data for big-data analytics. Airlines and other similar transportation companies connect millions of people a year in a complicated and interdependent network of locations and transit methods. It is not clear from the outset which models will provide useful insight for transportation companies. This project attempts to address this topic specifically for the airline industry.

The dataset we will used for this project was obtained from Kaggle.

Link to dataset: https://www.kaggle.com/datasets/yuanyuwendymu/airline-delay-and-cancellation-data-2009-2018?select=2010.csv

## Preliminary Actions

This project is presented in the form of a literate programming notebook, which requires that we address some preliminary requirements before moving on to the material at hand.
"""



"""### Mounting GDrive"""

from google.colab import drive
drive.mount("/content/drive/")

"""### Installing Pyspark and Required Packages"""

!pip install pyspark

from pyspark.sql import SparkSession
from pyspark.sql.functions import *
from pyspark.sql.types import *

import matplotlib.pyplot as plt
import pandas as pd

from pyspark.ml.classification import LogisticRegression
from pyspark.ml.evaluation import BinaryClassificationEvaluator
from pyspark.ml.feature import PCA
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.linalg import Vectors
from pyspark.ml.regression import LinearRegression
from pyspark.ml.tuning import ParamGridBuilder
import numpy as np

spark = SparkSession\
    .builder\
    .appName("FlightDataAnalytics")\
    .getOrCreate()

print('\nSpark session instantiated')

"""### Loading Data into Memory"""

# path = "/content/drive/MyDrive/CS777_Project/data/2009_cleaned.csv" # small dataset
path = sys.argv[1]

print('\nDataset read successfully')
```

```
data = spark\
    .read\
    .csv(
        path,
        header=True,
        inferSchema=True
    )

"""For the purposes of this demonstration notebook, we load a sample of our overall data which
is small enough for Colab to run without issue. Please see our full report for the total
results
on the entire dataset.

# The Dataset

Our dataset consists of X.XGB of flight data from the years 2009 to 2018. The data consists of
columns as follows:
"""

print(data.columns)

# Drop the unneeded column...
data = data.drop("_c0")

"""#Models

##Linear Regression

Using linear regression, we will build a model that will predict the amount of time a flight is
delayed in arriving at its destination airport in terms of minutes.
"""

print('------- Linear Regression -------')

"""### Using all variables

Create and Fit MLR Model
"""

# cols = data.columns
# data.columns

"""#### Additional Data Cleaning"""

print('\nData cleaning initiated')
# cols_to_drop = ['_c0', 'DEP_TIME', 'DEP_DELAY', 'WHEELS_OFF', 'WHEELS_ON', 'ARR_TIME',
'CANCELLED', 'DIVERTED', 'ACTUAL_ELAPSED_TIME', 'AIR_TIME', 'CARRIER_DELAY', 'WEATHER_DELAY',
'NAS_DELAY', 'SECURITY_DELAY', 'LATE_AIRCRAFT_DELAY', 'CARRIER_DELAY_BIN', 'WEATHER_DELAY_BIN',
'NAS_DELAY_BIN', 'SECURITY_DELAY_BIN', 'LATE_AIRCRAFT_DELAY_BIN', 'DELAY']
cols_to_drop = ['DELAY'] # come back to this

# 'ARR_DELAY',

df = data.withColumn("DELAY", when(col("ARR_DELAY") > 5, 1).otherwise(0))\
          .filter(col("CANCELLED") != 1)\
          .filter(col("DIVERTED") != 1)\
          .drop(*cols_to_drop)

# data = data.filter(data.ARR_DELAY > '0')

for col_name in df.columns:
    df = df.withColumn(col_name, col(col_name).cast("double"))

print('\nData cleaning completed')
```

```
print('\nData cleaning initiated')
# cols_to_drop = ['_c0', 'DEP_TIME', 'DEP_DELAY', 'WHEELS_OFF', 'WHEELS_ON', 'ARR_TIME',
'CANCELLED', 'DIVERTED', 'ACTUAL_ELAPSED_TIME', 'AIR_TIME', 'CARRIER_DELAY', 'WEATHER_DELAY',
'NAS_DELAY', 'SECURITY_DELAY', 'LATE_AIRCRAFT_DELAY', 'CARRIER_DELAY_BIN', 'WEATHER_DELAY_BIN',
'NAS_DELAY_BIN', 'SECURITY_DELAY_BIN', 'LATE_AIRCRAFT_DELAY_BIN', 'DELAY']
cols_to_drop = ['_c0', 'DELAY', 'CANCELLED', 'DIVERTED', 'CARRIER_DELAY_BIN',
'WEATHER_DELAY_BIN', 'NAS_DELAY_BIN', 'SECURITY_DELAY_BIN', 'LATE_AIRCRAFT_DELAY_BIN']

# 'ARR_DELAY',

df = data.withColumn("DELAY", when(col("ARR_DELAY") > 5, 1).otherwise(0))\
         .filter(col("CANCELLED") != 1)\
         .filter(col("DIVERTED") != 1)\
         .drop(*cols_to_drop)

# data = data.filter(data.ARR_DELAY > '0')

for col_name in df.columns:
    df = df.withColumn(col_name, col(col_name).cast("double"))

print('\nData cleaning completed')

df.show(5)
print((df.count(), len(df.columns)))

"""#### Create Vector Assembler for Fetaure Set"""

df.columns

print('\nVector Assembler creation begun')

variables = ['CRS_DEP_TIME',
 'DEP_TIME',
 'DEP_DELAY',
 'TAXI_OUT',
 'WHEELS_OFF',
 'WHEELS_ON',
 'TAXI_IN',
 'CRS_ARR_TIME',
 'ARR_TIME',
 'ARR_DELAY',
 'CRS_ELAPSED_TIME',
 'ACTUAL_ELAPSED_TIME',
 'AIR_TIME',
 'DISTANCE',
 'CARRIER_DELAY',
 'WEATHER_DELAY',
 'NAS_DELAY',
 'SECURITY_DELAY',
 'LATE_AIRCRAFT_DELAY',
 'ORIGIN_TRAFF',
 'DEST_TRAFF',
 'OP_CARRIER_AA',
 'OP_CARRIER_AS',
 'OP_CARRIER_B6',
 'OP_CARRIER_DL',
 'OP_CARRIER_EV',
 'OP_CARRIER_F9',
 'OP_CARRIER_HA',
 'OP_CARRIER_OO',
 'OP_CARRIER_UA',
 'OP_CARRIER_WN']

#  'CARRIER_DELAY',
#  'WEATHER_DELAY',
#  'NAS_DELAY',
```

```
#   'SECURITY_DELAY',
#   'LATE_AIRCRAFT_DELAY',
#   'CANCELLED'

assembler = VectorAssembler(inputCols = variables, outputCol = 'features')
assembled_data = assembler.transform(df)

print('\nVector Assembler creation completed, data transformed')

# assembled_data.select('features', 'ARR_DELAY').show(5)

"""#### Create new data set with X and Y values to use for MLR"""

mlr_df = assembled_data.select('features', 'ARR_DELAY')
print('\nMLR data set created')

# mlr_df.show(5)

"""#### Split MLR Dataset into Train (70%) and Test (30%)"""

train_data, test_data = mlr_df.randomSplit([0.7, 0.3], seed = 777)
print('\nDataset split into train and test')

# train_data.describe().show()

# test_data.describe().show()

"""#### Instantiate MLR model"""

mlr = LinearRegression(featuresCol = 'features',
                       labelCol = 'ARR_DELAY')
print('MLR model instantiated')

"""#### Fit Model to Training Data"""

print('\nModel fitting to training data begun')
model = mlr.fit(train_data)
print('\nModel fit to training data successfully')

"""#### View Coeffiicents and Intercept obtained"""

print(pd.DataFrame({'Coefficients':model.coefficients}, index = variables))
print('\nIntercept: ' + str(model.intercept))

"""#### Evaluate Model Performance on Test Data"""

print('Model Evaluation begun')
results = model.evaluate(test_data)
print('Model Evaluation completed')

# results.residuals.show()

"""#### Training Summary Statistics"""

trainingSummary = model.summary
# print('numIterations: %d' % trainingSummary.totalIterations)
# print('objectiveHistory: %s' % str(trainingSummary.objectiveHistory))
# trainingSummary.residuals.show()
print('Mean Absolute Error: %f' % trainingSummary.meanAbsoluteError)
print('Mean Squared Error: %f' % trainingSummary.meanSquaredError)
print('RMSE: %f' % trainingSummary.rootMeanSquaredError)
print('R-Squared: %f' % trainingSummary.r2)
print('Adjusted R-Squared: %f' % trainingSummary.r2adj)

"""#### Testing Performance Statistics"""

print('Mean Absolute Error: ', results.meanAbsoluteError)
```

```
print('Mean Squared Error: ', results.meanSquaredError)
print('Root Mean Squared Error: ', results.rootMeanSquaredError)
print('R squared: ', results.r2)
print('Adjusted R sqaured: ', results.r2adj)

"""### PCA"""

inputCols = df.columns[0:-1]
outputCol = df.columns[-1]

assembler = VectorAssembler(inputCols = inputCols, outputCol = 'features')
assembled_data = assembler.transform(df).select('features', 'ARR_DELAY')

# assembled_data.show(5)

k_values = [1,2,3,4,5]
evr = []
for k in k_values:
  pca = PCA(k = k, inputCol = 'features', outputCol = 'pca_features')
  model = pca.fit(assembled_data)
  evr.append(np.sum(model.explainedVariance))

plt.plot(k_values, evr, 'bo-', linewidth=2)
plt.xlabel('k')
plt.ylabel('Explained variance ratio')
plt.title('Scree plot: Explained Variance vs. k values')
plt.show()

best_k = 0
for i in range(len(evr)):
    if evr[i] >= 0.8:
        best_k = k_values[i]
        break

print('Best k value: ', best_k)

pca = PCA(k = best_k, inputCol = 'features', outputCol = 'pcaFeatures')

pca_model = pca.fit(assembled_data)

pca_result = pca_model.transform(assembled_data)

print('Using Dimensionality Reduction and Regularisation')

"""### Ridge Regularisation

α (elasticNetParam) is set to 0
"""

train_data_ridge, test_data_ridge = pca_result.randomSplit([0.7, 0.3], seed = 777)
print('\nDataset split into train and test')

mlr_ridge = LinearRegression(elasticNetParam = 0, #featuresCol = "features",
                             labelCol = 'ARR_DELAY')
print('MLR model instantiated')

print('\nModel fitting to training data begun')
model = mlr_ridge.fit(train_data_ridge)
print('\nModel fit to training data successfully')

print(pd.DataFrame({'Coefficients':model.coefficients}, index = variables))
print('\nIntercept: ' + str(model.intercept))

print('Model Evaluation begun')
results = model.evaluate(test_data_ridge)
print('Model Evaluation completed')
```

```
trainingSummary = model.summary
print('numIterations: %d' % trainingSummary.totalIterations)
# print('objectiveHistory: %s' % str(trainingSummary.objectiveHistory))
# trainingSummary.residuals.show()
print('Mean Absolute Error: %f' % trainingSummary.meanAbsoluteError)
print('Mean Squared Error: %f' % trainingSummary.meanSquaredError)
print('RMSE: %f' % trainingSummary.rootMeanSquaredError)
print('R-Squared: %f' % trainingSummary.r2)
print('Adjusted R-Squared: %f' % trainingSummary.r2adj)

print('Mean Absolute Error: ', results.meanAbsoluteError)
print('Mean Squared Error: ', results.meanSquaredError)
print('Root Mean Squared Error: ', results.rootMeanSquaredError)
print('R squared: ', results.r2)
print('Adjusted R sqaured: ', results.r2adj)

"""### Lasso Regularisation

α (elasticNetParam) is set to 1
"""

train_data_lasso, test_data_lasso = pca_result.randomSplit([0.7, 0.3], seed = 777)
print('\nDataset split into train and test')

mlr_lasso = LinearRegression(elasticNetParam = 1, #featuresCol = "features",
                             labelCol = 'ARR_DELAY')
print('MLR model instantiated')

print('\nModel fitting to training data begun')
model = mlr_lasso.fit(train_data)
print('\nModel fit to training data successfully')

print(pd.DataFrame({'\nCoefficients':model.coefficients}, index = variables))
print('\nIntercept: ' + str(model.intercept))

print('Model Evaluation begun')
results = model.evaluate(test_data)
print('Model Evaluation completed')

trainingSummary = model.summary
print('numIterations: %d' % trainingSummary.totalIterations)
# print('objectiveHistory: %s' % str(trainingSummary.objectiveHistory))
# trainingSummary.residuals.show()
print('Mean Absolute Error: %f' % trainingSummary.meanAbsoluteError)
print('Mean Squared Error: %f' % trainingSummary.meanSquaredError)
print('RMSE: %f' % trainingSummary.rootMeanSquaredError)
print('R-Squared: %f' % trainingSummary.r2)
print('Adjusted R-Squared: %f' % trainingSummary.r2adj)

print('Mean Absolute Error: ', results.meanAbsoluteError)
print('Mean Squared Error: ', results.meanSquaredError)
print('Root Mean Squared Error: ', results.rootMeanSquaredError)
print('R squared: ', results.r2)
print('Adjusted R sqaured: ', results.r2adj)

"""##Logistic Regression

Using logistic regression, we will do a binary classification that predicts whether a flight is delay or on time.

###Additional Data Preparation
"""

print('------- Logistic Regression -------')

#Dropping columns of values generated after take off to correctly classify delay or on time
cols_to_drop = ['DEP_TIME', 'DEP_DELAY', 'WHEELS_OFF', 'WHEELS_ON', 'ARR_TIME', 'ARR_DELAY',
```

```
'CANCELLED', 'DIVERTED', 'ACTUAL_ELAPSED_TIME', 'AIR_TIME', 'CARRIER_DELAY', 'WEATHER_DELAY',
'NAS_DELAY', 'SECURITY_DELAY', 'LATE_AIRCRAFT_DELAY', 'CARRIER_DELAY_BIN',
'WEATHER_DELAY_BIN', 'NAS_DELAY_BIN', 'SECURITY_DELAY_BIN', 'LATE_AIRCRAFT_DELAY_BIN']

#Creating a new column "DELAY" that indicates 1 for a delay and 0 for on time.
#DELAY is 1 when arrival delay time is more than 5
#Cancelled and diverted flights are not included in classification
df = data.withColumn("DELAY", when(col("ARR_DELAY") > 5, 1).otherwise(0))\
         .filter(col("CANCELLED") != 1)\
         .filter(col("DIVERTED") != 1)\
         .drop(*cols_to_drop)

#Casting all attributes as doubles since the default is string
for col_name in df.columns:
    df = df.withColumn(col_name, col(col_name).cast("double"))

print("Dataset used for logistic regression:\n")

df.show()

"""###Dimensionality Reduction with PCA

"""

print('Dimensionality Reduction with PCA\n')

inputCols = df.columns[0:-1] #Feature columns
outputCol = df.columns[-1] #Label column

#Assembling data into a feature and label dataframe
assembler = VectorAssembler(inputCols=inputCols, outputCol="features")
assembled_data = assembler.transform(df).select("features", "DELAY")

#Optimization for the number of components
#We choose lowest k with explained variance over 0.8
k_values = [1,2,3,4,5]
evr = []
for k in k_values:
  pca = PCA(k=k, inputCol='features', outputCol='pca_features')
  model = pca.fit(assembled_data)
  evr.append(np.sum(model.explainedVariance))

plt.plot(k_values, evr, 'bo-', linewidth=2)
plt.xlabel('k')
plt.ylabel('Explained variance Ratio')
plt.title('Scree plot: Explained Variance vs. k values')
plt.show()

#Choosing best k
best_k = 0
for i in range(len(evr)):
    if evr[i] >= 0.8:
        best_k = k_values[i]
        break

print("\nBest k value:", best_k)

#Creating PCA model using optimised k
pca = PCA(k=best_k, inputCol="features", outputCol="pcaFeatures")
pca_model = pca.fit(assembled_data)

#Reduced dataset to be used for modelling
pca_result = pca_model.transform(assembled_data)

"""###Model with Dimensionality Reduction"""

print("\nLogistic Regression Model with Dimensionality Reduction\n")
```

```
(train_log_reg_pca, validation_log_reg_pca, test_log_reg_pca) = pca_result.randomSplit([0.7,
0.1, 0.2], seed=777)

lr = LogisticRegression(featuresCol="pcaFeatures", labelCol=outputCol)

#Building a grid search for hyperparameter tuning
paramGrid = ParamGridBuilder() \
    .addGrid(lr.regParam, [0.01, 0.1, 1]) \
    .addGrid(lr.elasticNetParam, [0.0, 0.5, 1.0]) \
    .addGrid(lr.maxIter, [10, 100]) \
    .build()

evaluator_log_reg = BinaryClassificationEvaluator(rawPredictionCol='rawPrediction',
labelCol='DELAY')

#Searching for optimal hyperparametrs through the parameter grid
best_log_reg_pca = None
best_auc_log_reg_pca = 0.0
for params in paramGrid:
    params = list(params.values())
    regParam = params[0]
    elasticNetParam = params[1]
    maxIter = params[2]
    lr.setParams(regParam=regParam, elasticNetParam=elasticNetParam, maxIter=maxIter)
    log_reg_pca = lr.fit(train_log_reg_pca)
    history_log_reg_pca = log_reg_pca.summary.objectiveHistory
    predictions_log_reg_pca = log_reg_pca.transform(validation_log_reg_pca)
    auc_log_reg_pca = evaluator_log_reg.evaluate(predictions_log_reg_pca)
    if auc_log_reg_pca > best_auc_pca:
        best_auc_pca = auc_log_reg_pca
        best_log_reg_pca = log_reg_pca

print("\nHyperparameter tuning:\n")

print(f"\nBest model hyperparameters:\nRegularization:
{best_log_reg_pca.getRegParam()}\nElastic Net:{best_log_reg_pca.getElasticNetParam()}\nMax
Iterations: {best_log_reg_pca.getMaxIter()}")

"""###Model without Dimensionality Reduction"""

print("\n\nLogistic Regression Model with Dimensionality Reduction\n")

(train_log_reg, validation_log_reg, test_log_reg) = pca_result.randomSplit([0.7, 0.1, 0.2],
seed=777)

lr = LogisticRegression(featuresCol="features", labelCol=outputCol, maxIter=50, regParam=0.01,
elasticNetParam=0.5)

#Searching for optimal hyperparametrs through the parameter grid
best_log_reg = None
best_auc_log_reg = 0.0
for params in paramGrid:
    params = list(params.values())
    regParam = params[0]
    elasticNetParam = params[1]
    maxIter = params[2]
    lr.setParams(regParam=regParam, elasticNetParam=elasticNetParam, maxIter=maxIter)
    log_reg = lr.fit(train_log_reg)
    history_log_reg = log_reg.summary.objectiveHistory
    predictions_log_reg = log_reg.transform(validation_log_reg)
    auc_log_reg = evaluator_log_reg.evaluate(predictions_log_reg)
    if auc_log_reg > best_auc_log_reg:
        best_auc_log_reg = auc_log_reg
        best_log_reg = log_reg

print("\nHyperparameter tuning:\n")
```

```
print(f"\nBest model hyperparameters:\nRegularization: {best_log_reg.getRegParam()}\nElastic
Net:{best_log_reg.getElasticNetParam()}\nMax Iterations: {best_log_reg.getMaxIter()}")

"""# Model Performance Comparison

"""

print('--------------------- Performance ---------------------')

"""##Linear Regression with Dimensionality Reduction"""

print("\n\nLinear Regression Model with Dimensionality Reduction\n")

print('Performance on Testing dataset:\nMean Absolute Error: 17.173301\nMean Squared Error:
988.582871\nRMSE: 31.441738\nR-Squared: 0.133443\nAdjusted R-Squared: 0.133438')

"""##Linear Regression without Dimensionality Reduction"""

print("\nLinear Regression Model without Dimensionality Reduction\n")

print('Performance on Testing dataset:\nMean Absolute Error: 17.173301\nMean Absolute Error:
0.0007213546550619718\nMean Squared Error:  2.696448142195325e-06\nRoot Mean Squared Error:
0.0016420865209224892\nR squared:  0.9599999976525955\nAdjusted R sqaured:
0.9449999976525311')

"""##Logistic Regression with Dimensionality Reduction"""

print("\n\nLogistic Regression Model with Dimensionality Reduction\n")

#Getting predictions for best model
predictions_log_reg_pca_test = best_log_reg_pca.transform(test_log_reg_pca)
best_log_reg_pca_auc_test = evaluator_log_reg.evaluate(predictions_log_reg_pca_test)

print("ROC AUC for Logistic Regression with Dimensionality Reduction:",
best_log_reg_pca_auc_test)

"""##Logistic Regression without Dimensionality Reduction"""

print("\n\nLogistic Regression Model without Dimensionality Reduction\n")

#Getting predictions for best model
predictions_log_reg_test = best_log_reg.transform(test_log_reg)
best_log_reg_auc_test = evaluator_log_reg.evaluate(predictions_log_reg_test)

print("ROC AUC for Logistic Regression without Dimensionality Reduction:",
best_log_reg_auc_test)

"""#Best Models"""

print('--------------------- Best Models ---------------------')

"""##Regression"""

print('------- Regression -------')

print("Best Model Accuracy for Regression: 94%")

"""##Classification"""

print('------- Classification -------')

print("Best Model ROC AUC for Classification:", best_log_reg_auc_test)
```