```
pip install pyspark
```

```
Collecting pyspark
  Downloading pyspark-3.2.1.tar.gz (281.4 MB)
     |████████████████████████████████| 281.4 MB 36 kB/s
Collecting py4j==0.10.9.3
  Downloading py4j-0.10.9.3-py2.py3-none-any.whl (198 kB)
     |████████████████████████████████| 198 kB 31.1 MB/s
Building wheels for collected packages: pyspark
  Building wheel for pyspark (setup.py) ... done
  Created wheel for pyspark: filename=pyspark-3.2.1-py2.py3-none-any.whl size=281853642
  Stored in directory: /root/.cache/pip/wheels/9f/f5/07/7cd8017084dce4e93e84e92efd1e1d5
Successfully built pyspark
Installing collected packages: py4j, pyspark
Successfully installed py4j-0.10.9.3 pyspark-3.2.1
```

```
import pyspark
from pyspark import SparkContext
from pyspark.sql import SQLContext
from pyspark.sql.functions import col

sc = SparkContext.getOrCreate()
sqlCtx = SQLContext(sc)
```

```
/usr/local/lib/python3.7/dist-packages/pyspark/sql/context.py:79: FutureWarning: Depreca
  FutureWarning
```

```
import pandas as pd
from matplotlib import pyplot as plt
import numpy as np
from pyspark.sql.functions import col
from pyspark.ml.feature import VectorAssembler
from pyspark.mllib.linalg import Vectors
from pyspark.mllib.regression import LabeledPoint
from pyspark.sql.types import DoubleType
from pyspark.ml import Pipeline
from pyspark.ml.feature import OneHotEncoder
```

```
flights_df = pd.read_csv("flight.csv")
```

```
flights_df.isnull().sum()
```

```
YEAR                      0
MONTH                     0
DAY                       0
DAY_OF_WEEK               0
AIRLINE                   0
```

```
     FLIGHT_NUMBER             0
     TAIL_NUMBER              13
     ORIGIN_AIRPORT            0
     DESTINATION_AIRPORT       0
     SCHEDULED_DEPARTURE       0
     DEPARTURE_TIME          388
     DEPARTURE_DELAY         388
     TAXI_OUT                389
     WHEELS_OFF              389
     SCHEDULED_TIME            0
     ELAPSED_TIME            407
     AIR_TIME                407
     DISTANCE                  0
     WHEELS_ON               395
     TAXI_IN                 395
     SCHEDULED_ARRIVAL         0
     ARRIVAL_TIME            395
     ARRIVAL_DELAY           407
     DIVERTED                  0
     CANCELLED                 0
     CANCELLATION_REASON    9607
     AIR_SYSTEM_DELAY       8272
     SECURITY_DELAY         8272
     AIRLINE_DELAY          8272
     LATE_AIRCRAFT_DELAY    8272
     WEATHER_DELAY          8272
     dtype: int64
```

```
flights_agg = flights_df[['MONTH','DAY','DAY_OF_WEEK','AIRLINE','ORIGIN_AIRPORT',
                          'SCHEDULED_DEPARTURE','SCHEDULED_TIME',
                          'DISTANCE','SCHEDULED_ARRIVAL','DEPARTURE_DELAY']].copy()
flights_agg = flights_agg.dropna(axis=0, how = "any")
```

```
flights_agg.isnull().sum()
```

```
     MONTH                  0
     DAY                    0
     DAY_OF_WEEK            0
     AIRLINE                0
     ORIGIN_AIRPORT         0
     SCHEDULED_DEPARTURE    0
     SCHEDULED_TIME         0
     DISTANCE               0
     SCHEDULED_ARRIVAL      0
     DEPARTURE_DELAY        0
     dtype: int64
```

```
flights_agg['DELAY'] = np.where(flights_agg['DEPARTURE_DELAY'] <= 0, 0, 1)
```

```
no_delay = (flights_agg['DELAY'] == 0).sum()
nobs = len(flights_agg['DELAY'])
no_delay_perc = float(no_delay)/nobs
```

```
delay_perc = 1 - no_delay_perc
print(no_delay_perc, delay_perc)
```

```
    0.6097180314223286 0.3902819685776714
```

```
delay = nobs - no_delay
no_delay_indices = flights_agg[flights_agg.DELAY == 0].index
#undersamples no delays to equal same number of delays
np.random.seed(5)
random_indices = np.random.choice(no_delay_indices, delay, replace=False)
no_delay_sample = flights_agg.loc[random_indices]
```

```
no_delay_sample[:10]
```

	MONTH	DAY	DAY_OF_WEEK	AIRLINE	ORIGIN_AIRPORT	SCHEDULED_DEPARTURE	SCHEDULED_
1781	1	1	4	WN	JAX	810	
3774	1	1	4	WN	BOI	1030	
4679	1	1	4	US	CLT	1130	
3506	1	1	4	DL	LAX	1010	
8837	1	1	4	OO	DTW	1550	
7662	1	1	4	EV	JAN	1436	
65	1	1	4	NK	BOS	510	
60	1	1	4	HA	HNL	502	
9992	1	1	4	HA	LAX	1705	
8270	1	1	4	DL	ATL	1515	

```
delay_sample = flights_agg[flights_agg.DELAY == 1]
flights_agg_balanced = delay_sample.append(no_delay_sample)
```

```
n = int(len(flights_agg_balanced)*0.10)
flights_new_bal = flights_agg_balanced.sample(n, random_state = 314)
```

```
no_delay_bal = (flights_new_bal['DELAY'] == 0).sum()
no_delay_perc_bal = float(no_delay_bal)/n
delay_perc_bal = 1 - no_delay_perc_bal
print(no_delay_perc_bal, delay_perc_bal)
```

```
    0.49733333333333335 0.5026666666666666
```

```
flights = pd.read_csv("flight.csv")
```

```
flight_df = sqlCtx.createDataFrame(flights_new_bal)
flight_df.show(5)
```

```
+-----+---+-----------+-------+--------------+-------------------+--------------+-------
|MONTH|DAY|DAY_OF_WEEK|AIRLINE|ORIGIN_AIRPORT|SCHEDULED_DEPARTURE|SCHEDULED_TIME|DISTANC
+-----+---+-----------+-------+--------------+-------------------+--------------+-------
|    1|  1|          4|     DL|           ATL|               1539|            74|     25
|    1|  1|          4|     EV|           RIC|               1047|            53|     16
|    1|  1|          4|     EV|           IAH|               1032|            67|     21
|    1|  1|          4|     UA|           EWR|                758|           181|     99
|    1|  1|          4|     EV|           MCI|                550|           135|     64
+-----+---+-----------+-------+--------------+-------------------+--------------+-------
only showing top 5 rows
```

```
flight_df.write.parquet("flight_df.parquet")
```

```
flight_df = sqlCtx.read.parquet("flight_df.parquet")
```

```
flight_df.show(4)
```

```
+-----+---+-----------+-------+--------------+-------------------+--------------+-------
|MONTH|DAY|DAY_OF_WEEK|AIRLINE|ORIGIN_AIRPORT|SCHEDULED_DEPARTURE|SCHEDULED_TIME|DISTANC
+-----+---+-----------+-------+--------------+-------------------+--------------+-------
|    1|  1|          4|     DL|           ATL|               1539|            74|     25
|    1|  1|          4|     EV|           RIC|               1047|            53|     16
|    1|  1|          4|     EV|           IAH|               1032|            67|     21
|    1|  1|          4|     UA|           EWR|                758|           181|     99
+-----+---+-----------+-------+--------------+-------------------+--------------+-------
only showing top 4 rows
```

```
# Use OneHotEncoder to map categorical variables to binary vectors
cat_columns = ['MONTH','DAY','DAY_OF_WEEK']
encoders = [OneHotEncoder(inputCol=column, outputCol=column+"_vec") for column in cat_columns
pipelineOHE = Pipeline(stages=encoders)
flight_df2 = pipelineOHE.fit(flight_df).transform(flight_df)
```

```
flight_df2.show(2)
```

```
+-----+---+-----------+-------+--------------+-------------------+--------------+-------
|MONTH|DAY|DAY_OF_WEEK|AIRLINE|ORIGIN_AIRPORT|SCHEDULED_DEPARTURE|SCHEDULED_TIME|DISTANC
+-----+---+-----------+-------+--------------+-------------------+--------------+-------
|    1|  1|          4|     DL|           ATL|               1539|            74|     25
|    1|  1|          4|     EV|           RIC|               1047|            53|     16
```

```
+-----+---+----------+------+-------------+------------------+------------+-------
only showing top 2 rows
```

```
assembler = VectorAssembler(inputCols=['MONTH_vec', 'DAY_vec', 'DAY_OF_WEEK_vec',
                                       'SCHEDULED_DEPARTURE', 'SCHEDULED_TIME', 'DISTANCE',
                                       'SCHEDULED_ARRIVAL'], outputCol="features")
```

```
# Apply vector assembler to data
transformed = assembler.transform(flight_df2)
```

```
transformed.select(['DELAY', 'features']).show(5)
```

```
+-----+--------------------+
|DELAY|            features|
+-----+--------------------+
|    0|(10,[6,7,8,9],[15...|
|    0|(10,[6,7,8,9],[10...|
|    0|(10,[6,7,8,9],[10...|
|    0|(10,[6,7,8,9],[75...|
|    0|(10,[6,7,8,9],[55...|
+-----+--------------------+
only showing top 5 rows
```

```
# Convert to RDD
dataRDD = transformed.select(['DELAY','features']).rdd.map(tuple)
```

```
# Map label to binary values, then convert to LabeledPoint
lp = dataRDD.map(lambda row : (0 if row[0] == 0 else 1, Vectors.dense(row[1])))    \
            .map(lambda row : LabeledPoint(row[0], row[1]))
```

```
lp.take(5)
```

```
[LabeledPoint(0.0, [0.0,0.0,0.0,0.0,0.0,0.0,1539.0,74.0,259.0,1653.0]),
 LabeledPoint(0.0, [0.0,0.0,0.0,0.0,0.0,0.0,1047.0,53.0,100.0,1140.0]),
 LabeledPoint(0.0, [0.0,0.0,0.0,0.0,0.0,0.0,1032.0,67.0,216.0,1139.0]),
 LabeledPoint(0.0, [0.0,0.0,0.0,0.0,0.0,0.0,758.0,181.0,997.0,1059.0]),
 LabeledPoint(0.0, [0.0,0.0,0.0,0.0,0.0,0.0,550.0,135.0,643.0,805.0])]
```

```
split = lp.randomSplit([0.8, 0.2], 314)
training = split[0]
test = split[1]
```

LOGISTIC REGRESSION

```
from pyspark.mllib.classification import LogisticRegressionWithLBFGS, LogisticRegressionModel


# Build model
LR_model = LogisticRegressionWithLBFGS.train(training)


# Evaluate model on training data
LR_LAPtrain = training.map(lambda lp: (float(LR_model.predict(lp.features)), lp.label))


# Print training accuracy
LR_accTrain = 1.0 * LR_LAPtrain.filter(lambda x:x[0] == x[1]).count()/training.count()
print(LR_accTrain)
```

```
     0.5830564784053156
```

```
# Evaluate model on test data
LR_LAP = test.map(lambda lp: (float(LR_model.predict(lp.features)), lp.label))



# Print test accuracy
LR_acc = 1.0 * LR_LAP.filter(lambda x:x[0] == x[1]).count()/test.count()
print(LR_acc)
```

```
     0.5878378378378378
```

RANDOM FOREST

```
from pyspark.mllib.tree import RandomForest, RandomForestModel
from pyspark.mllib.util import MLUtils
from pyspark.ml.feature import StringIndexer
from pyspark.ml.classification import RandomForestClassifier


# Build model
RF_model = RandomForest.trainClassifier(training, numClasses = 2,
                                       categoricalFeaturesInfo = {},
                                       numTrees = 5, featureSubsetStrategy = "auto",
                                       impurity = 'gini', maxDepth = 4, maxBins = 32)


# Evaluate model on training data
RF_predtrain = RF_model.predict(training.map(lambda x: x.features))
RF_LAPtrain = training.map(lambda lp: lp.label).zip(RF_predtrain)


# Print training accuracy
RF_trainAcc = RF_LAPtrain.filter(lambda x: x[0] == x[1]).count() / float(training.count())
print(RF_trainAcc)
```

```
     0.6710963455149501
```

```
# Evaluate model on test data
RF_pred = RF_model.predict(test.map(lambda x: x.features))
RF_LAP = test.map(lambda lp: lp.label).zip(RF_pred)
```

```
# Print test accuracy
RF_testAcc = RF_LAP.filter(lambda x: x[0] == x[1]).count() / float(test.count())
print(RF_testAcc)
```

```
     0.5743243243243243
```

CROSS VALIDATION

```
# Prepare data for modeling
flight_cv = transformed.select(['DELAY', 'features'])
flight_cv = flight_cv.withColumnRenamed('DELAY', 'label')
flight_cv = flight_cv.select(flight_cv.label.cast(DoubleType()).alias('label'),
                                 'features')
flight_cv.show(5)
```

```
     +-----+--------------------+
     |label|            features|
     +-----+--------------------+
     |  0.0|(10,[6,7,8,9],[15...|
     |  0.0|(10,[6,7,8,9],[10...|
     |  0.0|(10,[6,7,8,9],[10...|
     |  0.0|(10,[6,7,8,9],[75...|
     |  0.0|(10,[6,7,8,9],[55...|
     +-----+--------------------+
     only showing top 5 rows
```

```
train_cv, test_cv = flight_cv.randomSplit([0.8, 0.2], 314)
```

```
from pyspark.ml.classification import LogisticRegression
from pyspark.ml.evaluation import BinaryClassificationEvaluator
from pyspark.ml.tuning import CrossValidator, ParamGridBuilder
```

```
# Build model
lr_k = LogisticRegression()
```

```
# Create grid of parameters
grid_k = ParamGridBuilder().addGrid(lr_k.maxIter, [0, 1, 5, 10, 25]) \
                           .addGrid(lr_k.regParam, [0.1,0.01]) \
                           .addGrid(lr_k.fitIntercept, [False, True])\
                           .addGrid(lr_k.elasticNetParam, [0.0,0.3, 0.5,0.8, 1.0])\
                           .build()
```

```
evaluator_k = BinaryClassificationEvaluator()
```

```
cv_lr = CrossValidator(estimator = lr_k, estimatorParamMaps = grid_k, evaluator = evaluator_k
```

```
# Run cross-validation
cvmodel_lr = cv_lr.fit(train_cv)
```

```
# Evaluate tuned model on training data
evaluator_k.evaluate(cvmodel_lr.transform(train_cv))
```

```
     0.6401938851603286
```

```
# Evaluate tuned model on test data
evaluator_k.evaluate(cvmodel_lr.transform(test_cv))
```

```
     0.5960702815038731
```

```
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
from pyspark.ml.feature import VectorIndexer, IndexToString
```

```
labelIndexer = StringIndexer(inputCol = "label",
                             outputCol = "indexedLabel").fit(flight_cv)
```

```
featureIndexer = VectorIndexer(inputCol="features",
                              outputCol="indexedFeatures",
                              maxCategories=4).fit(flight_cv)
```

```
labelConverter = IndexToString(inputCol="prediction",
                               outputCol="predictedLabel",
                               labels=labelIndexer.labels)
```

```
rf_k = RandomForestClassifier(labelCol = "indexedLabel",
                              featuresCol = "indexedFeatures")
```

```
evaluator_rf = MulticlassClassificationEvaluator(labelCol="indexedLabel",
                                                 predictionCol="prediction",
                                                metricName="accuracy")
numFolds = 5
```

```
# Create grid of parameters
grid_k_rf = ParamGridBuilder().addGrid(rf_k.numTrees, [5,10,25])\
                           .addGrid(rf_k.maxDepth, [3, 5,10,15])\
                           .addGrid(rf_k.maxBins, [5, 10, 20, 30])\
                           .build()
```

```
# Create pipeline of transformers and estimators
pipeline_rf = Pipeline(stages=[labelIndexer,
                               featureIndexer,
                               rf_k,
                               labelConverter])
```

```
# Treat pipeline as estimator in a CrossValidator instance.
cv_rf = CrossValidator(estimator = pipeline_rf,
                       estimatorParamMaps = grid_k_rf,
                       evaluator = evaluator_rf,
                       numFolds = numFolds)
```

```
# Run cross-validation
cvmodel_rf = cv_rf.fit(train_cv)
```

```
# Evaluate tuned model on training data
predictions_rf_train = cvmodel_rf.transform(train_cv)
evaluator_rf.evaluate(predictions_rf_train)
```

```
0.6374172185430463
```

```
# Evaluate tuned model on test data
predictions_rf = cvmodel_rf.transform(test_cv)
```

```
predictions_rf.select("predictedLabel", "label", "features").show(5)
```

```
+--------------+-----+--------------------+
|predictedLabel|label|            features|
+--------------+-----+--------------------+
|           0.0|  0.0|(10,[6,7,8,9],[60...|
|           0.0|  0.0|(10,[6,7,8,9],[62...|
|           0.0|  0.0|(10,[6,7,8,9],[64...|
|           0.0|  0.0|(10,[6,7,8,9],[72...|
|           0.0|  0.0|(10,[6,7,8,9],[73...|
+--------------+-----+--------------------+
only showing top 5 rows
```

```
evaluator_rf.evaluate(predictions_rf)
```

```
0.541095890410959
```

```
results = pd.DataFrame(data={'Logistic Regression': [0.5898583146905294,0.6129032258064516],
                             'Random Forests': [0.6167039522744221,0.5513196480938416],
                             },
                       index={'Training Accuracy',
                              'Test Accuracy'})
results
```

	Logistic Regression	Random Forests
Test Accuracy	0.589858	0.616704
Training Accuracy	0.612903	0.551320

```
results_kfold = pd.DataFrame(data={'Logistic Regression': [0.6401754040204967,0.6146659497244
                             'Random Forests': [0.7105459985041137,0.5565217391304348],
                             },
                       index={'Training Accuracy',
                              'Test Accuracy'})
results_kfold
```

	Logistic Regression	Random Forests
Test Accuracy	0.640175	0.710546
Training Accuracy	0.614666	0.556522

✓ 0s completed at 11:29 PM ● ×