MACHINE LEARNING USING SPARK

SUBMITTED BY: ANKIT SINGH

Research Objective

- There is a famous quote you can not recycle waste time. In our country flight delays has become a major problem because of large number of airlines companies and travellers, so here our objective is to predict flight delays based on some features so that we can reduce the inconvenience and can create a better ecosystem.
- Here our second objective is to find the most important features which are responsible for flight delays.

Importance of our Objective

- There is a famous quote time is more valuable than money. You can get more money, but you cannot get more time.
- We all have faced a situation when our flights got delayed and because of flight delays many of us has missed important exams, meetings, functions, events and what not. Our flight prediction model can help authorities and airlines companies to understand that which flights are on the verge of getting delay and by having access of this kind of information concerned authority can plan the entire operation accordingly so that organizations can enhance the customer experience.
- This flight prediction can help increasing revenue of airlines companies, cause this model will help organizations to serve better to customers so that they can enhance customer experience.

- APPROACH
- Creating the Spark Environment
- Loading the required libraries
- Understanding the dataset
- Data Exploration
- Applying Decision tree model and predictions

Understanding the dataset

- Id → Unique id for every record
- Dofw → Day of the week
- carrier→ Name of the airlines company
- origin→ Origin of flight
- dest → Destination of flight
- crsdeptime → Flight departure time
- crsarrtime → Flight arrival time
- Cresdephour → Departure delays by hour of day
- depdelay → Flight has got delayed for how much time
- dist – Distance between origin and destination

CODING PART

To initiate the spark environment in Collab we will first need to install java

Java Installation

- Here we are installing updated java in our VM

```
[10] !apt update > /dev/null
     !apt install openjdk-8-jdk-headless -qq > /dev/null
```

- To run the spark in our colab vm we first have to install Hadoop and top of it we will install spark.

Spark Installation

- Here we are downloading spark files from the internet using !wget command and then we are installing spark in our VM on top of hadoop.

```
!wget -q http://apache.osuosl.org/spark/spark-3.1.2/spark-3.1.2-bin-hadoop3.2.tgz


!tar xf spark-3.1.2-bin-hadoop3.2.tgz

!pip install -q pyspark

import os
os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-8-openjdk-amd64"

os.environ["SPARK_HOME"] = "/content/spark-3.1.2-bin-hadoop3.2"
```

LOADING THE DATASET

```
[22]
FLIGHTS_TRAIN_DATA = '/content/drive/MyDrive/Assignment Data/flights20170102.json'
FLIGHTS_TEST_DATA = '/content/drive/MyDrive/Assignment Data/flights20170304.json'
```

Below we specify the data source, schema and class to load into a Dataset. We load the data from January and February, which we will use for training the model. (Note that specifying the schema when loading data into a DataFrame will give better performance than schema inference).

```
[23] # define the schema, corresponding to a line in the JSON data file.
schema = StructType([
    StructField("_id", StringType(), nullable=True),
    StructField("dofW", IntegerType(), nullable=True),
    StructField("carrier", StringType(), nullable=True),
    StructField("origin", StringType(), nullable=True),
    StructField("dest", StringType(), nullable=True),
    StructField("crsdephour", IntegerType(), nullable=True),
    StructField("crsdeptime", DoubleType(), nullable=True),
    StructField("depdelay", DoubleType(), nullable=True),
    StructField("crsarrtime", DoubleType(), nullable=True),
    StructField("arrdelay", DoubleType(), nullable=True),
    StructField("crselapsedtime", DoubleType(), nullable=True),
    StructField("dist", DoubleType(), nullable=True)]
  )
```

```
[24] # Load training data
train_df = spark.read.json(path=FLIGHTS_TRAIN_DATA, schema=schema)
train_df.cache()
```

LOADING THE DATASET

```
[26] train_df.show(5)

+--------------------+----+-------+------+----+-----------+----------+--------+----------+--------+--------------+------+
|                 _id|dofW|carrier|origin|dest|crsdephour|crsdeptime|depdelay|crsarrtime|arrdelay|crselapsedtime|  dist|
+--------------------+----+-------+------+----+-----------+----------+--------+----------+--------+--------------+------+
|AA_2017-01-01_ATL...|   7|     AA|   ATL| LGA|         17|    1700.0|     0.0|    1912.0|     0.0|         132.0| 762.0|
|AA_2017-01-01_LGA...|   7|     AA|   LGA| ATL|         13|    1343.0|     0.0|    1620.0|     0.0|         157.0| 762.0|
|AA_2017-01-01_MIA...|   7|     AA|   MIA| ATL|          9|     939.0|     0.0|    1137.0|    10.0|         118.0| 594.0|
|AA_2017-01-01_ORD...|   7|     AA|   ORD| MIA|         20|    2020.0|     0.0|      26.0|     0.0|         186.0|1197.0|
|AA_2017-01-01_LGA...|   7|     AA|   LGA| MIA|          7|     700.0|     0.0|    1017.0|     0.0|         197.0|1096.0|
+--------------------+----+-------+------+----+-----------+----------+--------+----------+--------+--------------+------+
only showing top 5 rows
```

```python
train_df.describe(["dist", "crselapsedtime", "depdelay", "arrdelay"]).show()
```

```
+-------+-----------------+-----------------+-----------------+-----------------+
|summary|             dist|   crselapsedtime|         depdelay|         arrdelay|
+-------+-----------------+-----------------+-----------------+-----------------+
|  count|            41348|            41348|            41348|            41348|
|   mean|1111.0529167069749|186.26264873754474|15.018719164167553|14.806907226468027|
| stddev| 568.7941212507543| 68.38149648990039| 44.52963204436135| 44.22370513266647|
|    min|            184.0|             64.0|              0.0|              0.0|
|    max|           2704.0|            423.0|           1440.0|           1442.0|
+-------+-----------------+-----------------+-----------------+-----------------+
```

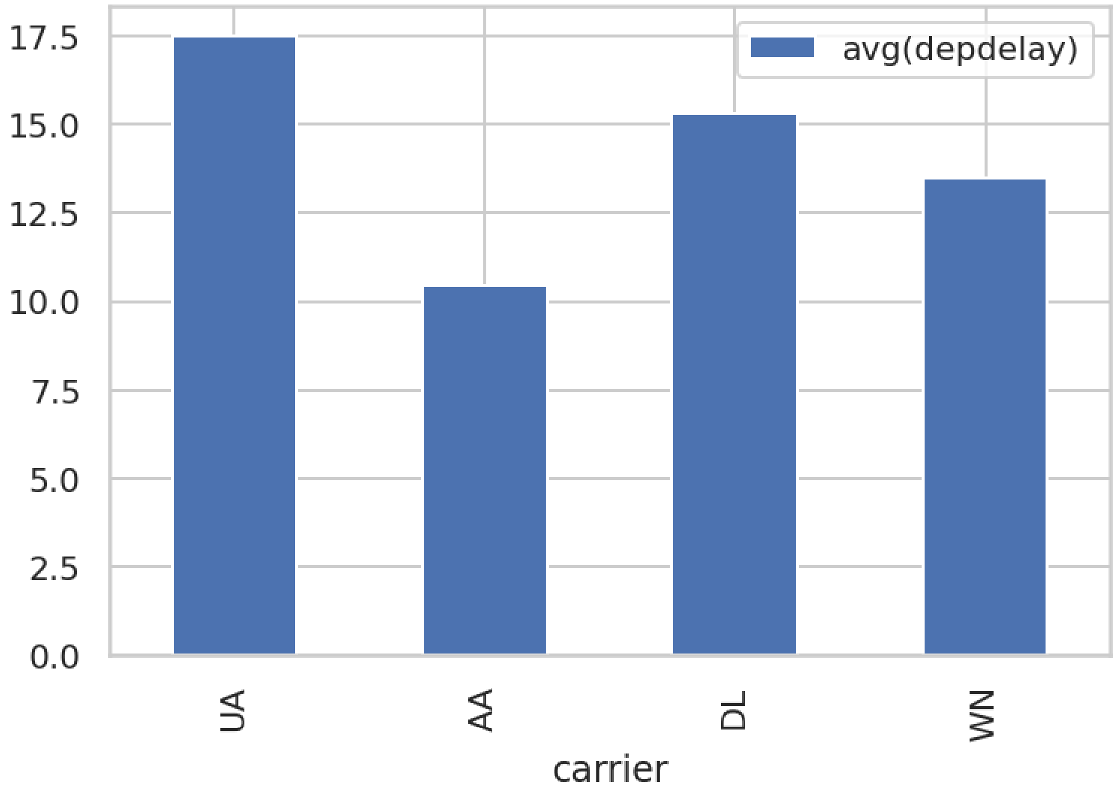
Statistics Summary

- Spark DataFrames include some built-in functions for statistical processing
- The describe() function performs summary statistical calculations on all numeric columns and returns them as a DataFrame.

TOP 5 LONGEST DEPARTURE DELAYS

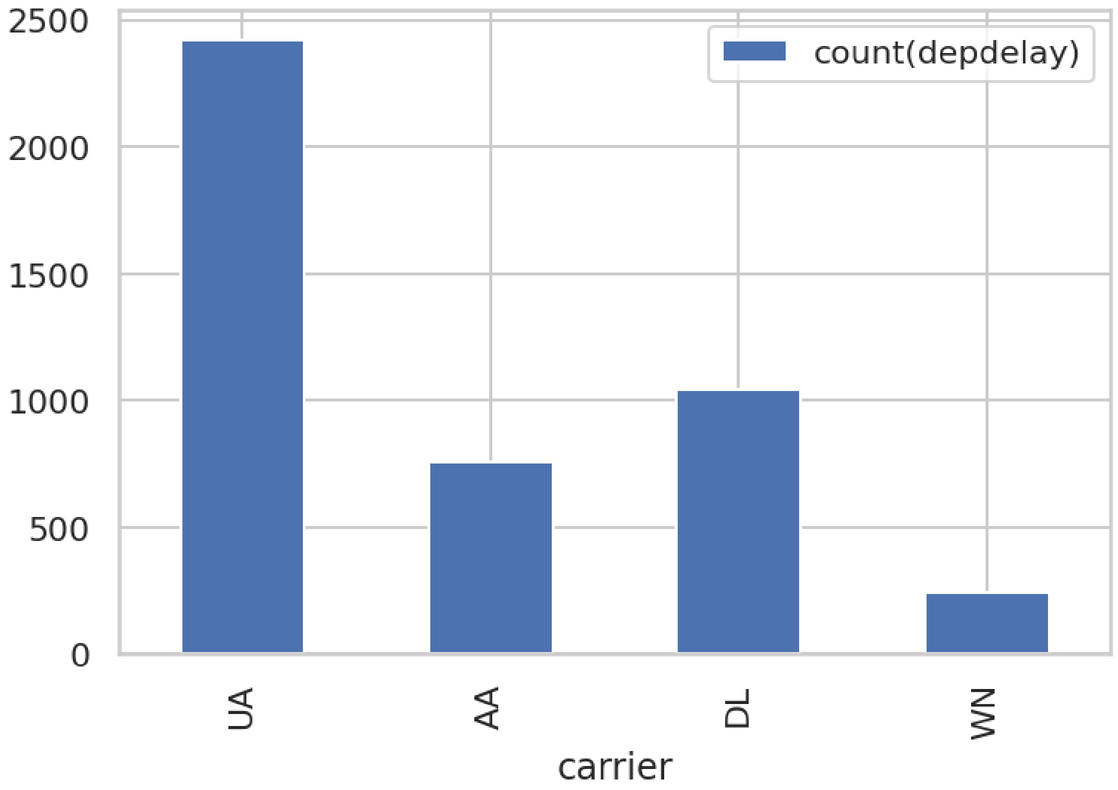
carrier	origin	dest	depdelay	crsdephour	dist	dofW
AA	SFO	ORD	1440.0	8	1846.0	3
DL	BOS	ATL	1185.0	17	946.0	6
UA	DEN	EWR	1138.0	12	1605.0	4
DL	ORD	ATL	1087.0	19	606.0	7
UA	MIA	EWR	1072.0	20	1085.0	1

AVERAGE DEPARTURE DELAY BY CARRIER



carrier	avg(depdelay)
UA	17.477878450696764
AA	10.45768118831622
DL	15.316061660865241
WN	13.491000418585182

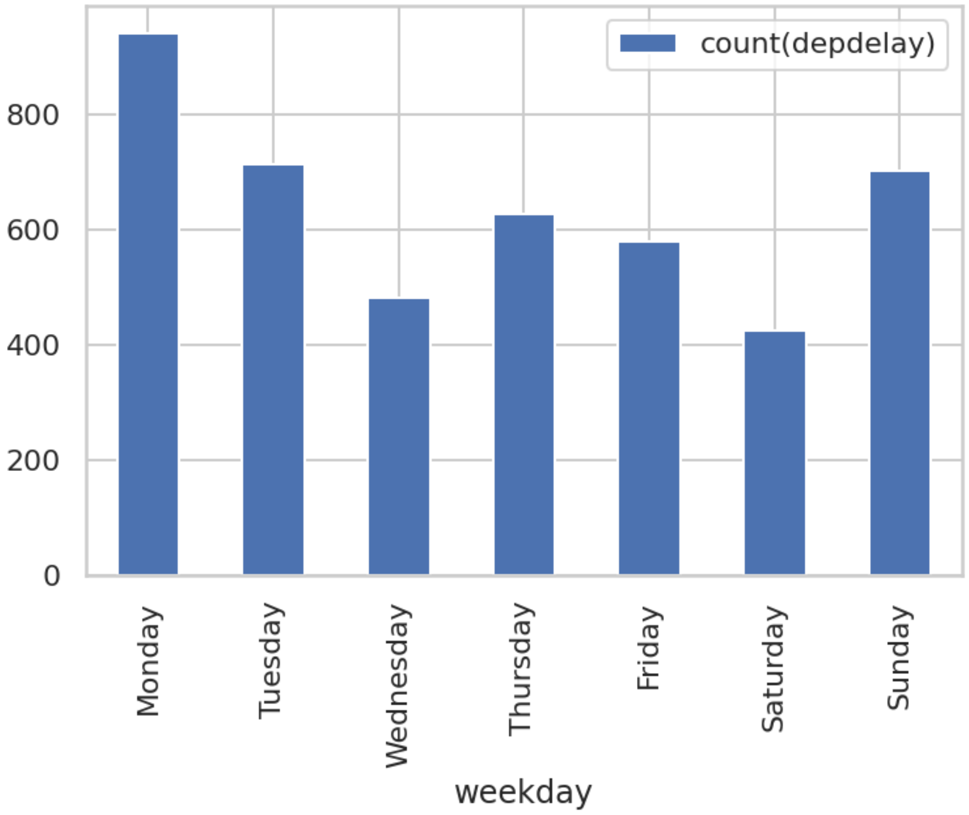
COUNT OF DEPARTURE DELAYS BY CARRIER (WHERE DELAYS >40 MINUTES)




```
sql_result_df.show()
```

```
+-------+---------------+
|carrier|count(depdelay)|
+-------+---------------+
|     UA|           2420|
|     AA|            757|
|     DL|           1043|
|     WN|            244|
+-------+---------------+
```

COUNT OF DEPARTURE DELAYS BY DAY OF THE WEEK



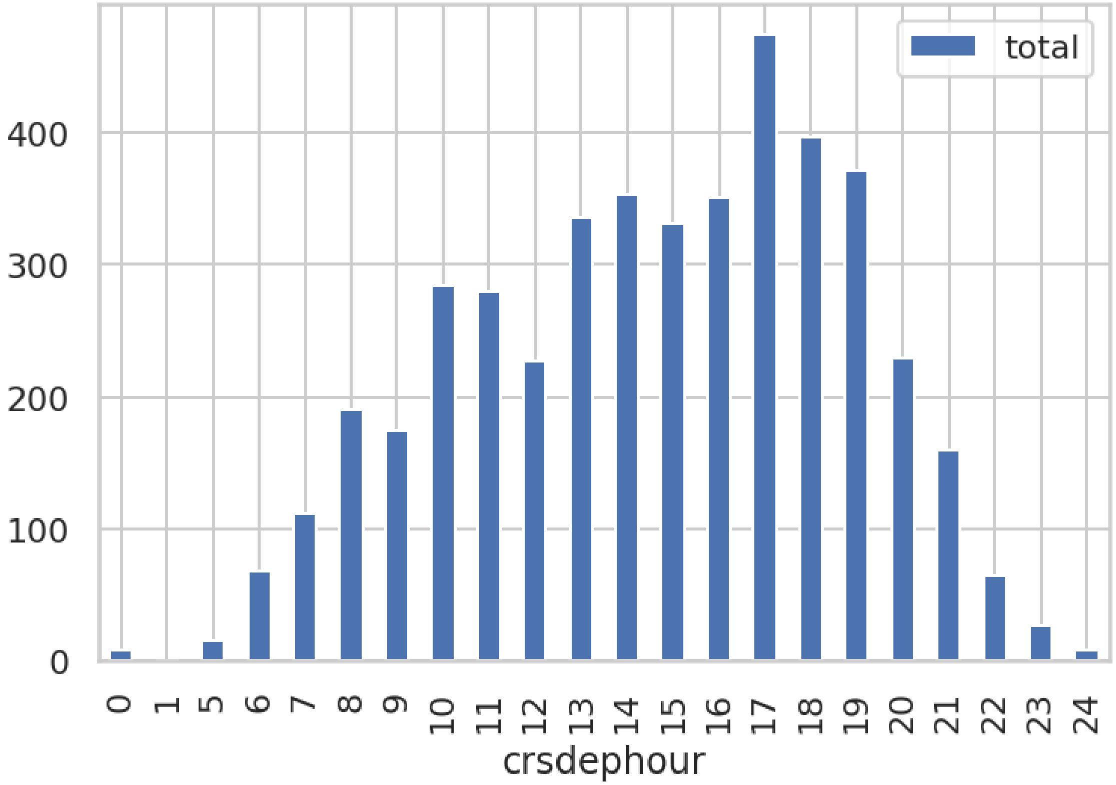

```
sql_result_df.show()
```

```
+----+---------------+---------+
|dofW|count(depdelay)|  weekday|
+----+---------------+---------+
|   1|            940|   Monday|
|   2|            712|  Tuesday|
|   3|            482|Wednesday|
|   4|            626| Thursday|
|   5|            579|   Friday|
|   6|            424| Saturday|
|   7|            701|   Sunday|
+----+---------------+---------+
```

COUNT OF DEPARTURE DELAYS BY HOUR OF DAY

```
sql_result_df.show(10)
```

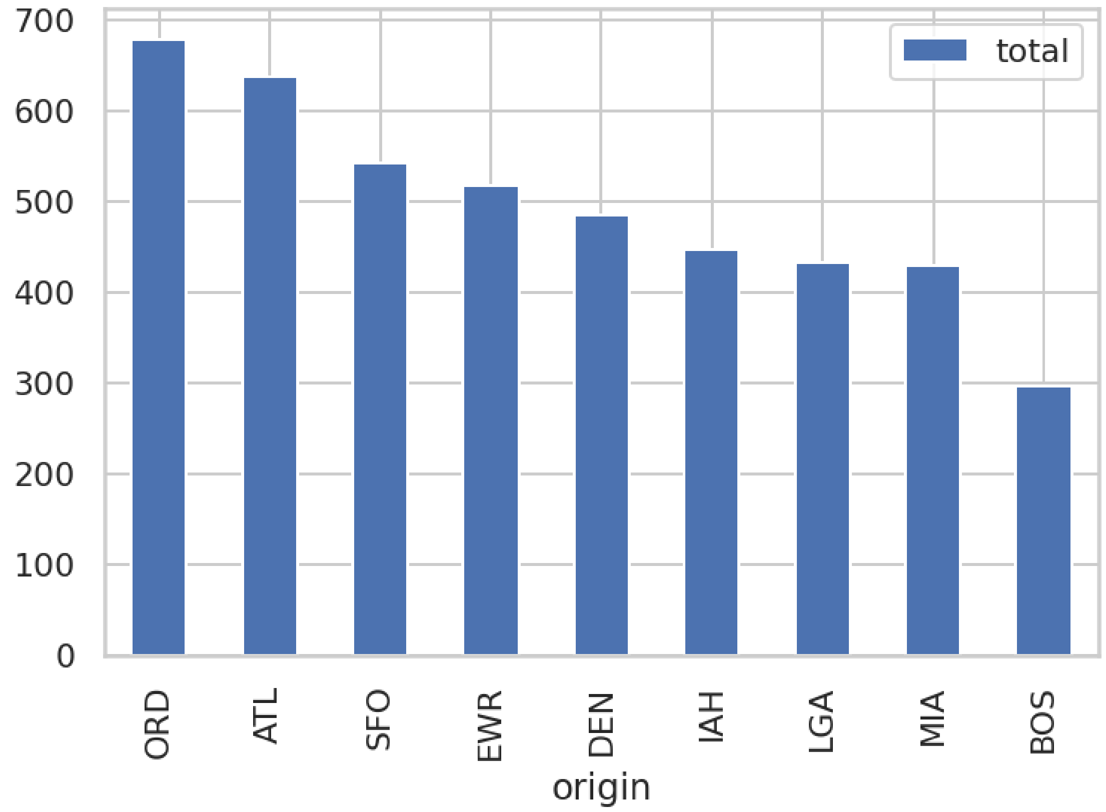
```
+----------+-----+
|crsdephour|total|
+----------+-----+
|         0|    9|
|         1|    1|
|         5|   15|
|         6|   68|
|         7|  112|
|         8|  190|
|         9|  175|
|        10|  284|
|        11|  280|
|        12|  227|
+----------+-----+
only showing top 10 rows
```



COUNT OF DEPARTURE DELAYS BY ORIGIN

```
sql_result_df.show()
```

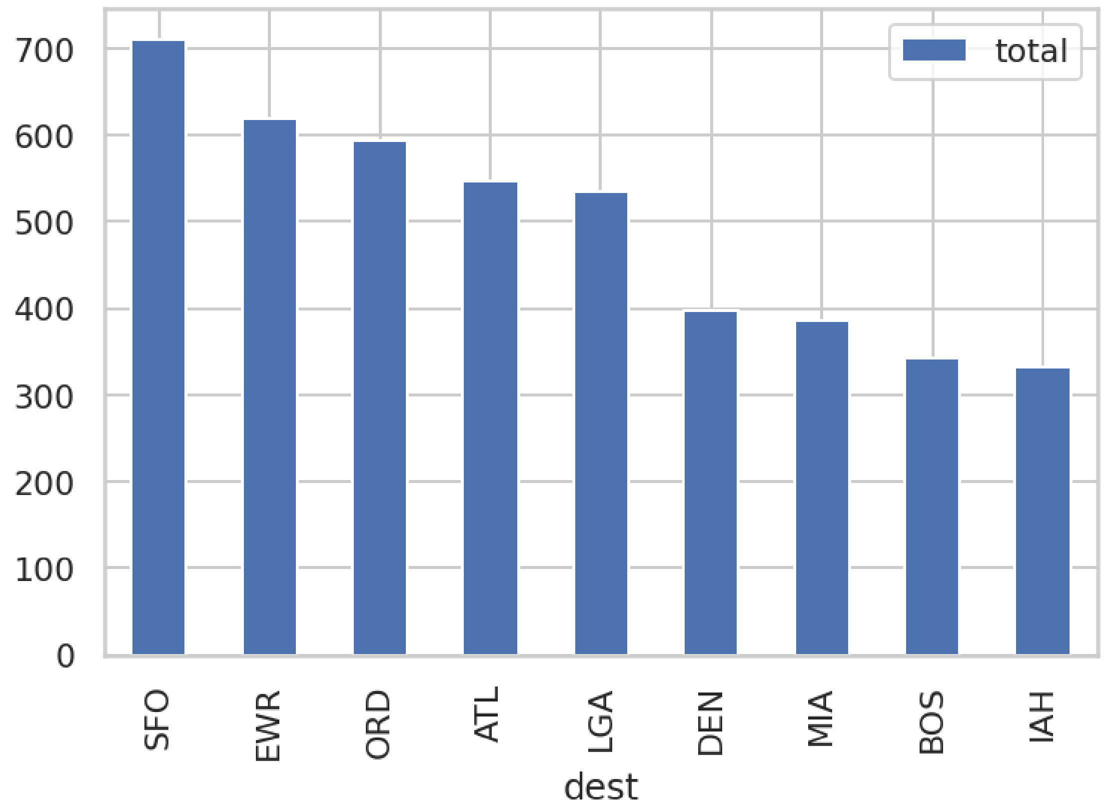
```
+------+-----+
|origin|total|
+------+-----+
|   ORD|  679|
|   ATL|  637|
|   SFO|  542|
|   EWR|  518|
|   DEN|  484|
|   IAH|  447|
|   LGA|  432|
|   MIA|  429|
|   BOS|  296|
+------+-----+
```



COUNT OF DEPARTURE DELAYS BY DESTINATION

```
sql_result_df.show()
```

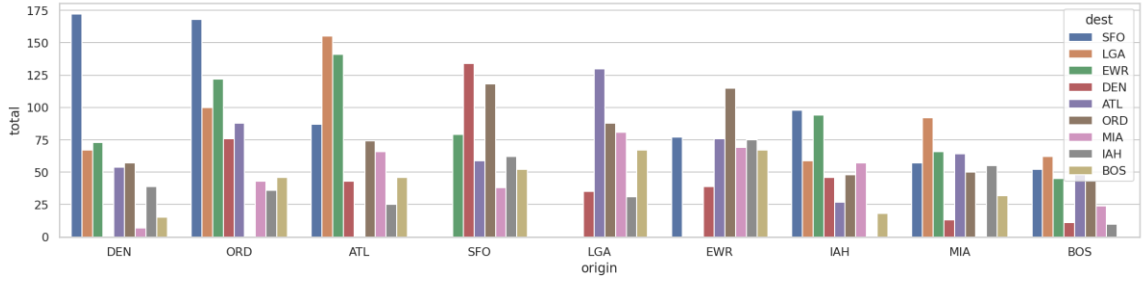
```
+----+-----+
|dest|total|
+----+-----+
| SFO|  711|
| EWR|  620|
| ORD|  593|
| ATL|  547|
| LGA|  535|
| DEN|  397|
| MIA|  385|
| BOS|  343|
| IAH|  333|
+----+-----+
```



COUNT FO DEPARTURE DELAYS BY ORIGIN & DESTINATION

```
sql_result_df.show()
```

```
+------+----+-----+
|origin|dest|total|
+------+----+-----+
|   DEN| SFO|  172|
|   ORD| SFO|  168|
|   ATL| LGA|  155|
|   ATL| EWR|  141|
|   SFO| DEN|  134|
|   LGA| ATL|  130|
|   ORD| EWR|  122|
|   SFO| ORD|  118|
|   EWR| ORD|  115|
|   ORD| LGA|  100|
|   IAH| SFO|   98|
|   IAH| EWR|   94|
|   MIA| LGA|   92|
|   ORD| ATL|   88|
|   LGA| ORD|   88|
|   ATL| SFO|   87|
|   LGA| MIA|   81|
|   SFO| EWR|   79|
|   EWR| SFO|   77|
|   ORD| DEN|   76|
+------+----+-----+
only showing top 20 rows
```



ADDING LABELS FOR DELAYED FLIGHTS AND COUNT

- We can use spark bucketizer to split the dataset into delayed and not delayed flights with a delayed 0/1 column. Then, the resulting total counts are displayed. Grouping the data by the delayed field and counting the number of instances in each group shows that there are roughly eight times as many not delayed samples as delayed samples.

```
[85] # User Bucketizer to split the data into custom buckets
     bucketizer = Bucketizer(splits=[0.0, 40.0, float("inf")], inputCol="depdelay", outputCol="delayed")
```

```
[86] train_df = bucketizer.transform(train_df).cache()
```

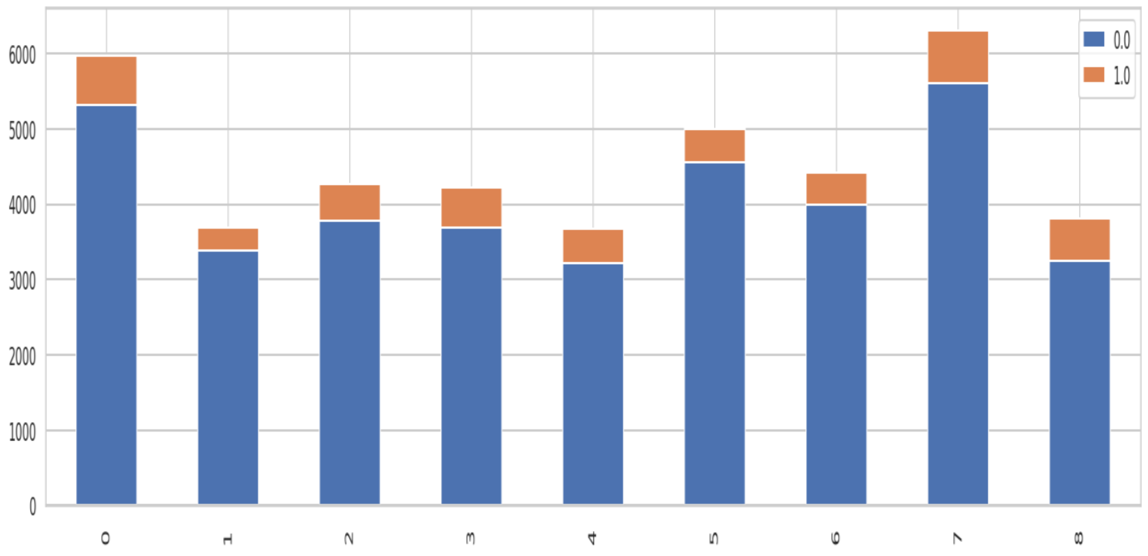
```
[87] train_df.groupBy("delayed").count().show()

     +-------+-----+
     |delayed|count|
     +-------+-----+
     |    0.0|36790|
     |    1.0| 4558|
     +-------+-----+
```

CREATE A STACKED BAR PLOT OF COUNT DEPARTURE DELAYS BY ORIGIN

```
# careful! we are using train_df and not result_df. Pivot will only work with a Grouped DF is Spark
train_df.select(["origin", "delayed"]).groupBy(["origin"]).pivot('delayed').count().orderBy('origin').show(15)
```

```
+------+----+---+
|origin| 0.0|1.0|
+------+----+---+
|   ATL|5322|649|
|   BOS|3388|306|
|   DEN|3773|496|
|   EWR|3693|526|
|   IAH|3218|455|
|   LGA|4550|442|
|   MIA|3991|434|
|   ORD|5607|693|
|   SFO|3248|557|
+------+----+---+
```



STRATIFIED SAMPLING

In order to ensure that our model is sensitive to the delayed samples, we can put the two sample types on the same footing using stratified sampling. The DataFrames sampleBy() function does this when provided with fractions of each sample type to be returned. Here, we're keeping all instances of delayed, but down sampling the not delayed instances to 29%, then displaying the results.

```
strat_train_df.show(5)
```

dofW	carrier	origin	dest	crsdephour	crsdeptime	depdelay	crsarrtime	arrdelay	crselapsedtime	dist	delayed
7	AA	ORD	DEN	13	1345.0	0.0	1527.0	0.0	162.0	888.0	0.0
7	AA	DEN	ORD	12	1235.0	0.0	1600.0	0.0	145.0	888.0	0.0
7	AA	ORD	DEN	10	1005.0	5.0	1145.0	3.0	160.0	888.0	0.0
7	AA	MIA	IAH	20	2045.0	80.0	2238.0	63.0	173.0	964.0	1.0
7	AA	MIA	LGA	18	1756.0	85.0	2049.0	73.0	173.0	1096.0	1.0

only showing top 5 rows

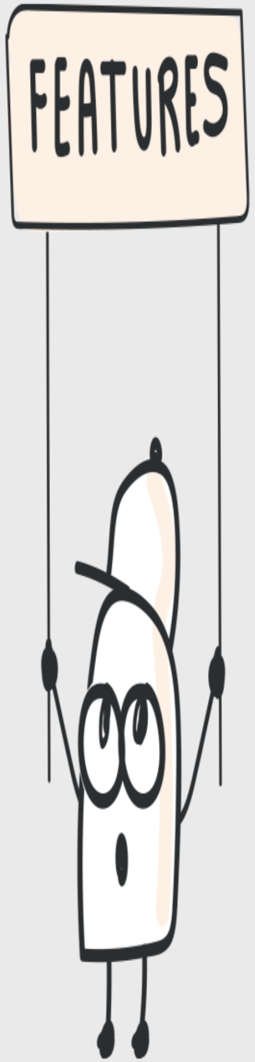
\+ Code \+ Text

```
[116] # Done with stratifying, we can drop the column
strat_train_df = strat_train_df.drop("delayed")
```

```
[117] strat_train_df.show(5)
```

dofW	carrier	origin	dest	crsdephour	crsdeptime	depdelay	crsarrtime	arrdelay	crselapsedtime	dist
7	AA	ORD	DEN	13	1345.0	0.0	1527.0	0.0	162.0	888.0
7	AA	DEN	ORD	12	1235.0	0.0	1600.0	0.0	145.0	888.0
7	AA	ORD	DEN	10	1005.0	5.0	1145.0	3.0	160.0	888.0
7	AA	MIA	IAH	20	2045.0	80.0	2238.0	63.0	173.0	964.0
7	AA	MIA	LGA	18	1756.0	85.0	2049.0	73.0	173.0	1096.0

only showing top 5 rows



FEATURES ARRAY

- -To build a classifier model, we have to extract the features that most contribute to the classification. In this scenario, we will build a tree to predict the label of delayed or not based on the following features:

    - Label:

        - delayed = 0

        - delayed = 1if delay > 40 minutes

- Features → {day of the week, scheduled departure time, scheduled arrival time, carrier, scheduled elapsed time origin, destination.
- In order for the features to be used by a machine learning algorithm, they must be transformed and put into feature vectors, which are vectors of numbers representing the value for each feature.

USING A COMBINATION OF STRINGINDEXER AND ONE HOTENCODER TO ENCODE CATEGORICAL COLUMNS – "CARRIER"

```python
# create a new "carrier_indexed" column
(indexed_df.select(["origin", "dest", "carrier", "carrier_indexed"])
                .sample(fraction=0.001, withReplacement=False, seed=rnd_seed).show())
```

```
+------+----+-------+---------------+
|origin|dest|carrier|carrier_indexed|
+------+----+-------+---------------+
|   MIA| LGA|     AA|            2.0|
|   SFO| DEN|     WN|            3.0|
|   ORD| ATL|     UA|            0.0|
|   ATL| MIA|     AA|            2.0|
|   MIA| ORD|     AA|            2.0|
|   DEN| ORD|     AA|            2.0|
|   LGA| ATL|     AA|            2.0|
|   ORD| LGA|     UA|            0.0|
|   ATL| EWR|     DL|            1.0|
|   MIA| BOS|     AA|            2.0|
|   BOS| LGA|     DL|            1.0|
|   DEN| ORD|     UA|            0.0|
|   SFO| ORD|     UA|            0.0|
|   ORD| DEN|     AA|            2.0|
|   ATL| BOS|     DL|            1.0|
|   ATL| LGA|     WN|            3.0|
|   MIA| ATL|     AA|            2.0|
|   SFO| MIA|     AA|            2.0|
+------+----+-------+---------------+
```

```python
(encoded_df.select(["origin", "dest", "carrier", "carrier_indexed", "carrier_encoded"])
                .sample(fraction=0.001, withReplacement=False, seed=rnd_seed).show())
```

```
+------+----+-------+---------------+---------------+
|origin|dest|carrier|carrier_indexed|carrier_encoded|
+------+----+-------+---------------+---------------+
|   MIA| LGA|     AA|            2.0|  (3,[2],[1.0])|
|   SFO| DEN|     WN|            3.0|      (3,[],[])|
|   ORD| ATL|     UA|            0.0|  (3,[0],[1.0])|
|   ATL| MIA|     AA|            2.0|  (3,[2],[1.0])|
|   MIA| ORD|     AA|            2.0|  (3,[2],[1.0])|
|   DEN| ORD|     AA|            2.0|  (3,[2],[1.0])|
|   LGA| ATL|     AA|            2.0|  (3,[2],[1.0])|
|   ORD| LGA|     UA|            0.0|  (3,[0],[1.0])|
|   ATL| EWR|     DL|            1.0|  (3,[1],[1.0])|
|   MIA| BOS|     AA|            2.0|  (3,[2],[1.0])|
|   BOS| LGA|     DL|            1.0|  (3,[1],[1.0])|
|   DEN| ORD|     UA|            0.0|  (3,[0],[1.0])|
|   SFO| ORD|     UA|            0.0|  (3,[0],[1.0])|
|   ORD| DEN|     AA|            2.0|  (3,[2],[1.0])|
|   ATL| BOS|     DL|            1.0|  (3,[1],[1.0])|
|   ATL| LGA|     WN|            3.0|      (3,[],[])|
|   MIA| ATL|     AA|            2.0|  (3,[2],[1.0])|
|   SFO| MIA|     AA|            2.0|  (3,[2],[1.0])|
+------+----+-------+---------------+---------------+
```

CREATING DECISION TREE ESTIMATOR, SETING LABEL AND FEATURE COLUMNS

7. Create Decision Tree Estimator, set Label and Feature Columns

```
[140] from pyspark.ml.classification import DecisionTreeClassifier
      dTree = DecisionTreeClassifier(featuresCol='features', labelCol='label', predictionCol='prediction', maxDepth=5, maxBins=7000)
```

Below, we chain the indexers and tree in a Pipeline.

7.1 Setup pipeline with feature transformers and model estimator

```
steps = stringIndexers + encoders + [labeler, assembler, dTree]
steps
```

```
[StringIndexerModel: uid=StringIndexer_839ec77dbeaa, handleInvalid=error,
 StringIndexerModel: uid=StringIndexer_5d0e4bea0f3d, handleInvalid=error,
 StringIndexerModel: uid=StringIndexer_2825b38fb0b2, handleInvalid=error,
 StringIndexerModel: uid=StringIndexer_8b77084f0408, handleInvalid=error,
 OneHotEncoder_4d944697e3c1,
 OneHotEncoder_17d93fe93c59,
 OneHotEncoder_a1480934003d,
 OneHotEncoder_8e9f2f09ec5d,
 Bucketizer_b797093a7344,
 VectorAssembler_9bf49dd775f1,
 DecisionTreeClassifier_382f1c0fecb8]
```

```
[142] pipeline = Pipeline(stages=steps)
```

TRAINIG THE MODEL

We would like to determine which parameter values of the decision tree produce the best model. A common technique for model selection is k-fold cross-validation, where the data is randomly split into k partitions. Each partition is used once as the testing dataset, while the rest are used for training. Models are then generated using the training sets and evaluated with the testing sets, resulting in k model performance measurements. The model parameters leading to the highest performance metric produce the best model.

Spark ML supports k-fold cross-validation with a transformation/estimation pipeline to try out different combinations of parameters, using a process called grid search, where you set up the parameters to test, and a cross-validation evaluator to construct a model selection workflow.

Below, we use a ParamGridBuilder to construct the parameter grid. We define an Evaluator, which will evaluate the model by comparing the test label column with the test prediction column. We use a CrossValidator for model selection.

SET UP A CROSSVALIDATOR WITH THE PARAMETERS, A TREE ESTIMATOR AND EVAULATOR

The crossvalidator uses the Estimator Pipeline, the parameter grid, and the classification Evaluator to fit the training set and returns a model

```
[143] # set param grid to search through decision tree's maxDepth parameter for best model
      # Deeper trees are potential more accurate, but are also more likely to overfit
      paramGrid = ParamGridBuilder().addGrid(dTree.maxDepth, [4, 5, 6]).build()
```

str: label

View

'6'

```
#evaluator = BinaryClassificationEvaluator(rawPredictionCol='rawPrediction', labelC          tricName="areaUnderROC")
evaluator = MulticlassClassificationEvaluator(predictionCol='prediction', labelCol="label", metricName="accuracy")
```

```
[145] # Set up 3-fold cross validation with paramGrid
      crossVal = CrossValidator(estimator=pipeline, evaluator=evaluator, estimatorParamMaps=paramGrid, numFolds=3)
```

USING CROSS VALIDATOR ESTIMATES TO FIT THE TRAINING DATASET

```
[146] cvModel = crossVal.fit(strat_train_df)
```

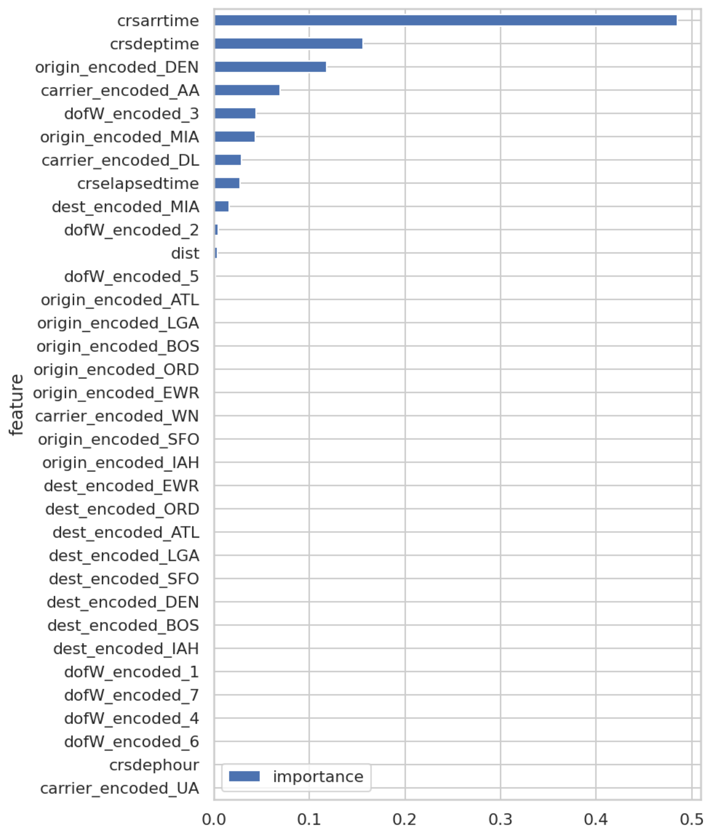
```
cvModel.bestModel.stages
```

```
[StringIndexerModel: uid=StringIndexer_839ec77dbeaa, handleInvalid=error,
 StringIndexerModel: uid=StringIndexer_5d0e4bea0f3d, handleInvalid=error,
 StringIndexerModel: uid=StringIndexer_2825b38fb0b2, handleInvalid=error,
 StringIndexerModel: uid=StringIndexer_8b77084f0408, handleInvalid=error,
 OneHotEncoderModel: uid=OneHotEncoder_4d944697e3c1, dropLast=false, handleInvalid=error,
 OneHotEncoderModel: uid=OneHotEncoder_17d93fe93c59, dropLast=false, handleInvalid=error,
 OneHotEncoderModel: uid=OneHotEncoder_a1480934003d, dropLast=false, handleInvalid=error,
 OneHotEncoderModel: uid=OneHotEncoder_8e9f2f09ec5d, dropLast=false, handleInvalid=error,
 Bucketizer_b797093a7344,
 VectorAssembler_9bf49dd775f1,
 DecisionTreeClassificationModel: uid=DecisionTreeClassifier_382f1c0fecb8, depth=5, numNodes=41, numClasses=2, numFeatures=34]
```

GET THE MOST IMPORTANT FEATURES AFFECTING THE DELAY

```
[159] feature_importance_df = pd.DataFrame({"feature":expandedFeatureCols, "importance":treeModel.featureImportances.toArray()})
      feature_importance_df
```

	feature	importance
0	carrier_encoded_UA	0.000000
1	carrier_encoded_DL	0.028929
2	carrier_encoded_AA	0.069490
3	carrier_encoded_WN	0.000000
4	origin_encoded_ORD	0.000000
5	origin_encoded_ATL	0.000000
6	origin_encoded_LGA	0.000000
7	origin_encoded_MIA	0.043412
8	origin_encoded_DEN	0.118263
9	origin_encoded_EWR	0.000000
10	origin_encoded_SFO	0.000000
11	origin_encoded_IAH	0.000000
12	origin_encoded_BOS	0.000000
13	dest_encoded_ORD	0.000000
14	dest_encoded_ATL	0.000000
15	dest_encoded_LGA	0.000000



PREDUCTIONS AND MODEL EVALUATION

The actual performance of the model can be determined using the test data set that has not been used for any training or cross-validation activities.

We transform the test Dataframe with the model pipeline, which will transform the features according to the pipeline, estimate and then return the label predictions in a column of a new dataframe.

```
[161] # transform the test set with the model pipeline,
      # which will map the features according to the same recipe
      predictions = cvModel.transform(test_df)
```

```
accuracy = evaluator.evaluate(predictions)
accuracy
```

```
0.8353502904418236
```

PREDUCTIONS AND MODEL EVALUATION

The actual performance of the model can be determined using the test data set that has not been used for any training or cross-validation activities.

We transform the test Dataframe with the model pipeline, which will transform the features according to the pipeline, estimate and then return the label predictions in a column of a new dataframe.

```
[161] # transform the test set with the model pipeline,
      # which will map the features according to the same recipe
      predictions = cvModel.transform(test_df)
```

```
accuracy = evaluator.evaluate(predictions)
accuracy
```

```
0.8353502904418236
```