CSEE5590 Big Data Programming

In Class Programming –14 Report
(Jongkook Son)

Project Overview:

MLlib is Apache Spark's scalable machine learning library, with APIs in Java, Scala, Python, and R.

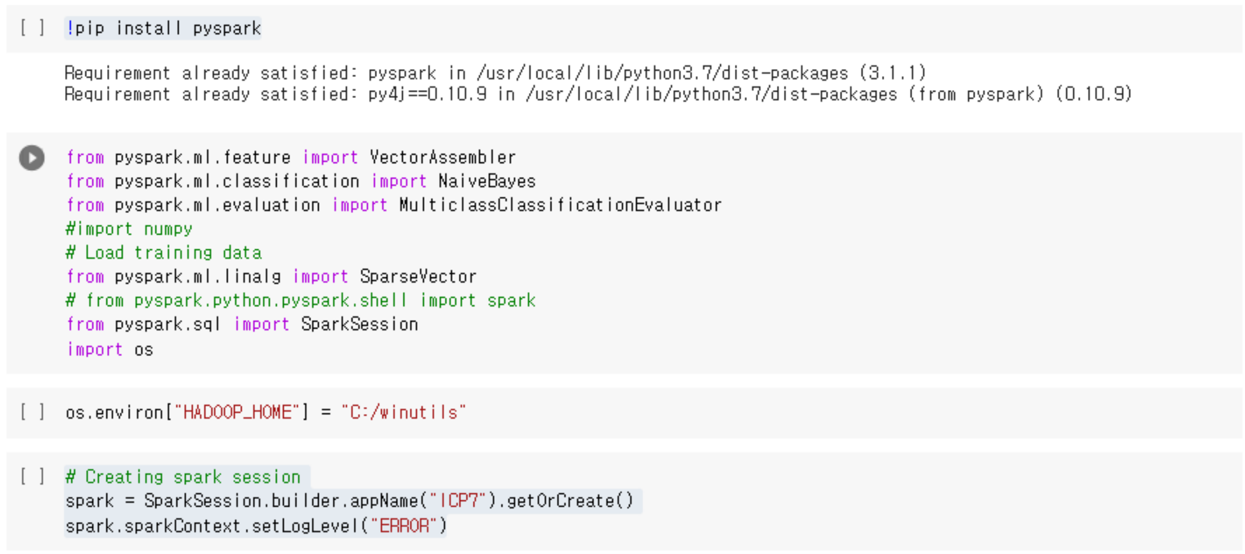
Requirements/Task(s):

In class exercise:
1. Classification:
Algorithms
a. Naïve Bayes b. Decision Tree c. Random Forest
2. Clustering:
Algorithm
a. KMeans
3. Regression:
Algorithm
a. Linear Regressionb. Logistic Regression

What I learned in ICP:

I learned How to implement some ML algrorithm using spark Mlib.Spark MLlib library provide scalable machine learning functionality which can leverage the Spark framework. ML offers the ability to make prediction based upon given datasets, and improves based upon the size of the dataset, which offers a nice synergy based upon the ability for Spark to handle very large amounts of data set. It was very meaningful to use spark for ML, which was very new to me.

Part 1 Classification



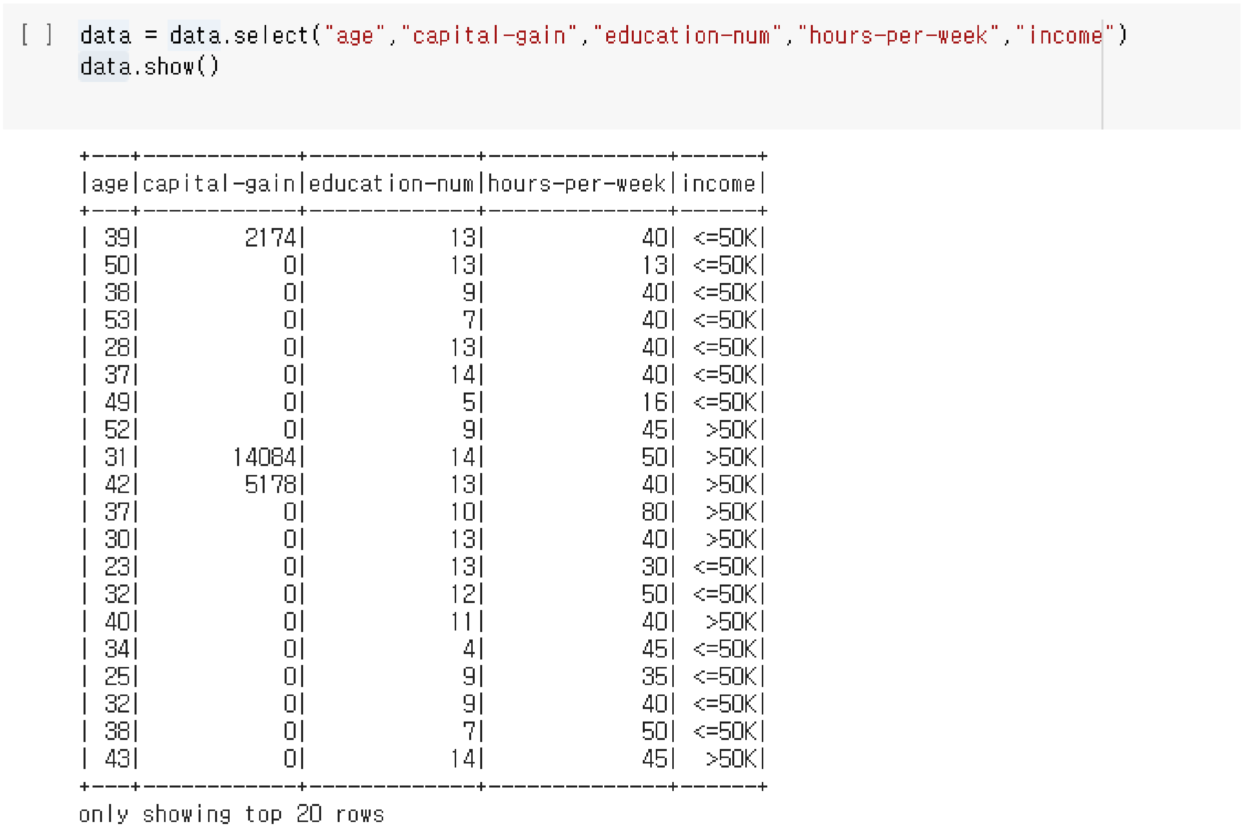
```
!pip install pyspark

Requirement already satisfied: pyspark in /usr/local/lib/python3.7/dist-packages (3.1.1)
Requirement already satisfied: py4j==0.10.9 in /usr/local/lib/python3.7/dist-packages (from pyspark) (0.10.9)
```

```
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.classification import NaiveBayes
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
#import numpy
# Load training data
from pyspark.ml.linalg import SparseVector
# from pyspark.python.pyspark.shell import spark
from pyspark.sql import SparkSession
import os
```

```
os.environ["HADOOP_HOME"] = "C:/winutils"
```

```
# Creating spark session
spark = SparkSession.builder.appName("ICP7").getOrCreate()
spark.sparkContext.setLogLevel("ERROR")
```

First import libraries and create spark session



```
data = data.select("age","capital-gain","education-num","hours-per-week","income")
data.show()
```

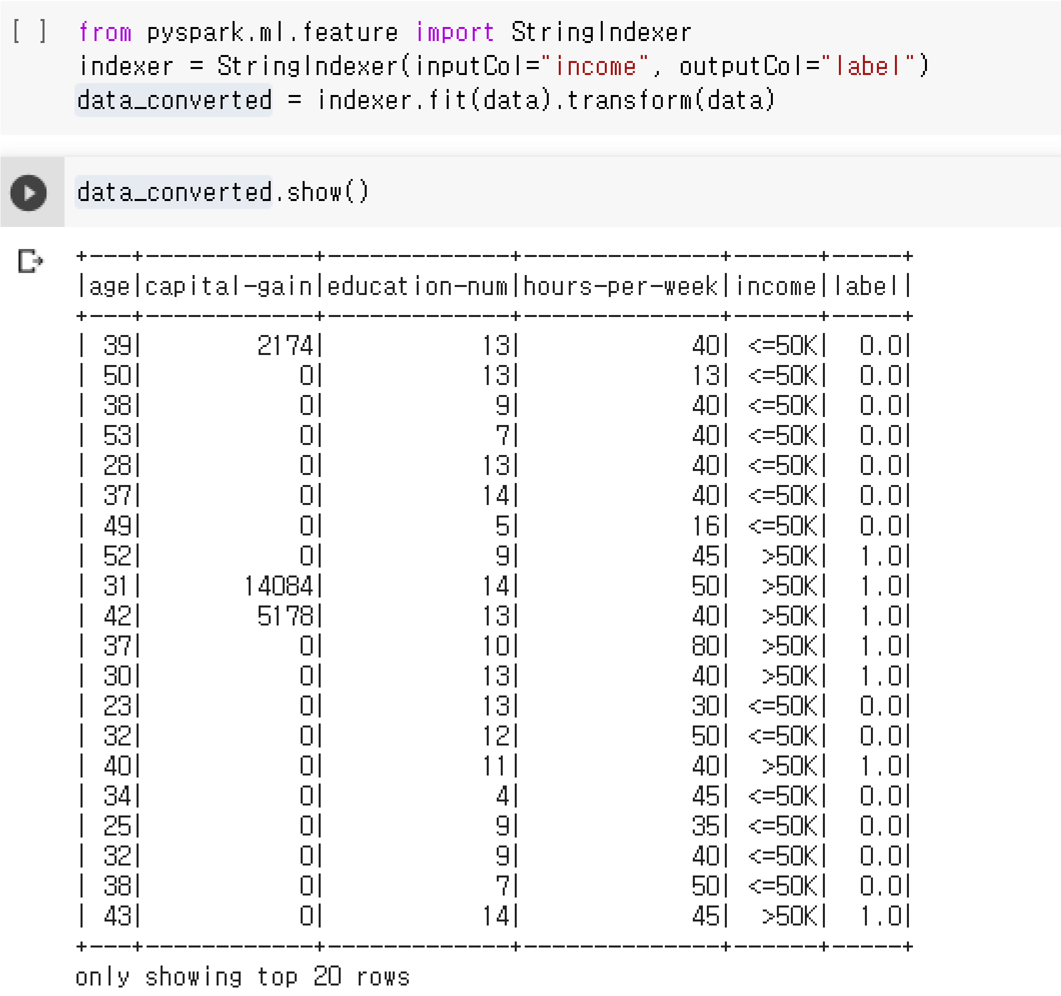
```
+---+------------+-------------+--------------+------+
|age|capital-gain|education-num|hours-per-week|income|
+---+------------+-------------+--------------+------+
| 39|        2174|           13|            40| <=50K|
| 50|           0|           13|            13| <=50K|
| 38|           0|            9|            40| <=50K|
| 53|           0|            7|            40| <=50K|
| 28|           0|           13|            40| <=50K|
| 37|           0|           14|            40| <=50K|
| 49|           0|            5|            16| <=50K|
| 52|           0|            9|            45|  >50K|
| 31|       14084|           14|            50|  >50K|
| 42|        5178|           13|            40|  >50K|
| 37|           0|           10|            80|  >50K|
| 30|           0|           13|            40|  >50K|
| 23|           0|           13|            30| <=50K|
| 32|           0|           12|            50| <=50K|
| 40|           0|           11|            40|  >50K|
| 34|           0|            4|            45| <=50K|
| 25|           0|            9|            35| <=50K|
| 32|           0|            9|            40| <=50K|
| 38|           0|            7|            50| <=50K|
| 43|           0|           14|            45|  >50K|
+---+------------+-------------+--------------+------+
only showing top 20 rows
```

Select numeric Features from data

Now, to train a mode need to have features for training which I have already chosen, and a target that I wanted to predict.

For the target, I used income feature.

However, we need it as a numeric, so I used the stringIndexer to do feature scaling as shown below.



```
from pyspark.ml.feature import StringIndexer
indexer = StringIndexer(inputCol="income", outputCol="label")
data_converted = indexer.fit(data).transform(data)
```

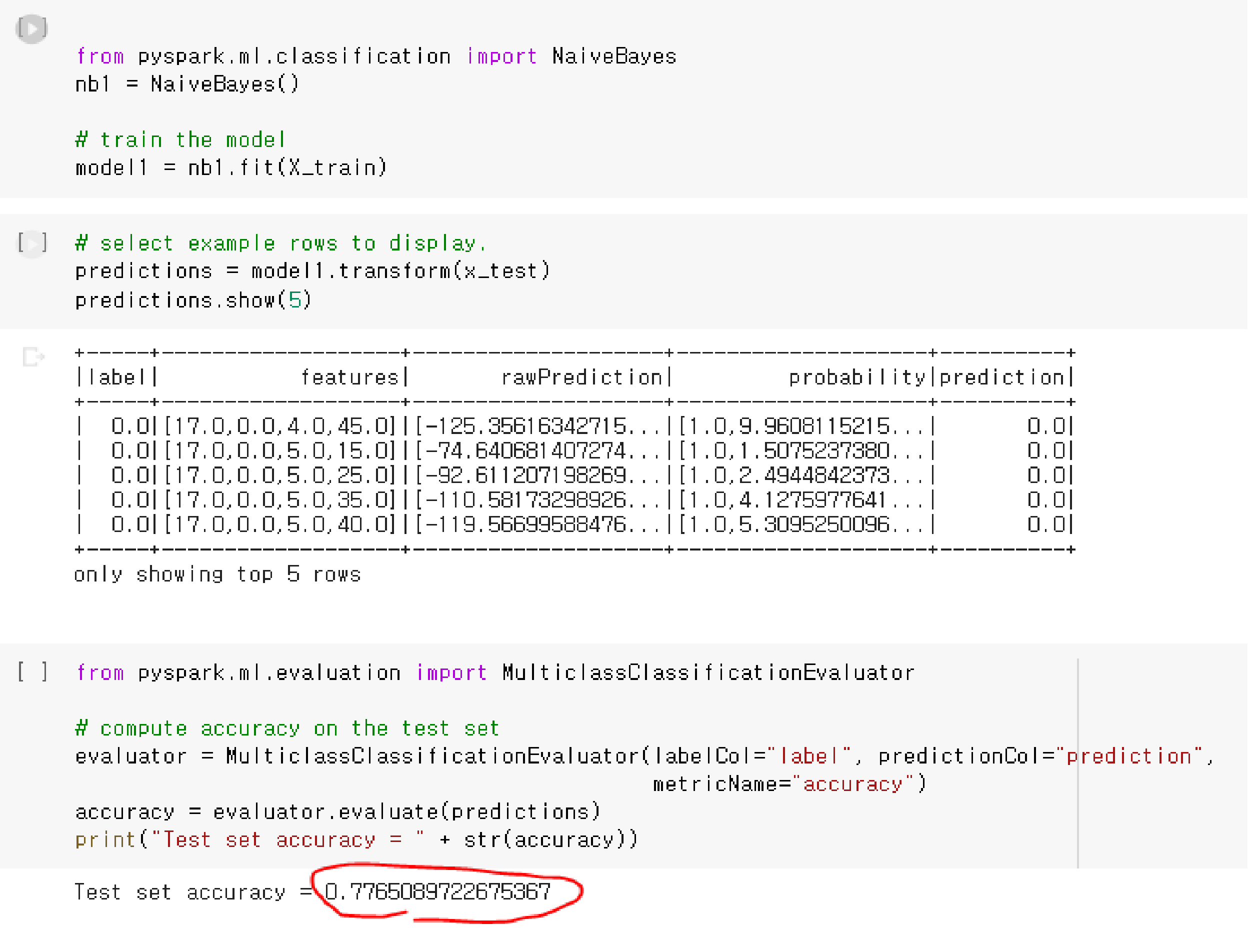
```
data_converted.show()
```

```
+---+------------+-------------+--------------+------+-----+
|age|capital-gain|education-num|hours-per-week|income|label|
+---+------------+-------------+--------------+------+-----+
| 39|        2174|           13|            40| <=50K|  0.0|
| 50|           0|           13|            13| <=50K|  0.0|
| 38|           0|            9|            40| <=50K|  0.0|
| 53|           0|            7|            40| <=50K|  0.0|
| 28|           0|           13|            40| <=50K|  0.0|
| 37|           0|           14|            40| <=50K|  0.0|
| 49|           0|            5|            16| <=50K|  0.0|
| 52|           0|            9|            45|  >50K|  1.0|
| 31|       14084|           14|            50|  >50K|  1.0|
| 42|        5178|           13|            40|  >50K|  1.0|
| 37|           0|           10|            80|  >50K|  1.0|
| 30|           0|           13|            40|  >50K|  1.0|
| 23|           0|           13|            30| <=50K|  0.0|
| 32|           0|           12|            50| <=50K|  0.0|
| 40|           0|           11|            40|  >50K|  1.0|
| 34|           0|            4|            45| <=50K|  0.0|
| 25|           0|            9|            35| <=50K|  0.0|
| 32|           0|            9|            40| <=50K|  0.0|
| 38|           0|            7|            50| <=50K|  0.0|
| 43|           0|           14|            45|  >50K|  1.0|
+---+------------+-------------+--------------+------+-----+
only showing top 20 rows
```

Then change all the features in to integer and used vector assembler

Finally I split the data 70% for train and 30% for test

Naïve Bayes



```
from pyspark.ml.classification import NaiveBayes
nb1 = NaiveBayes()

# train the model
model1 = nb1.fit(X_train)
```

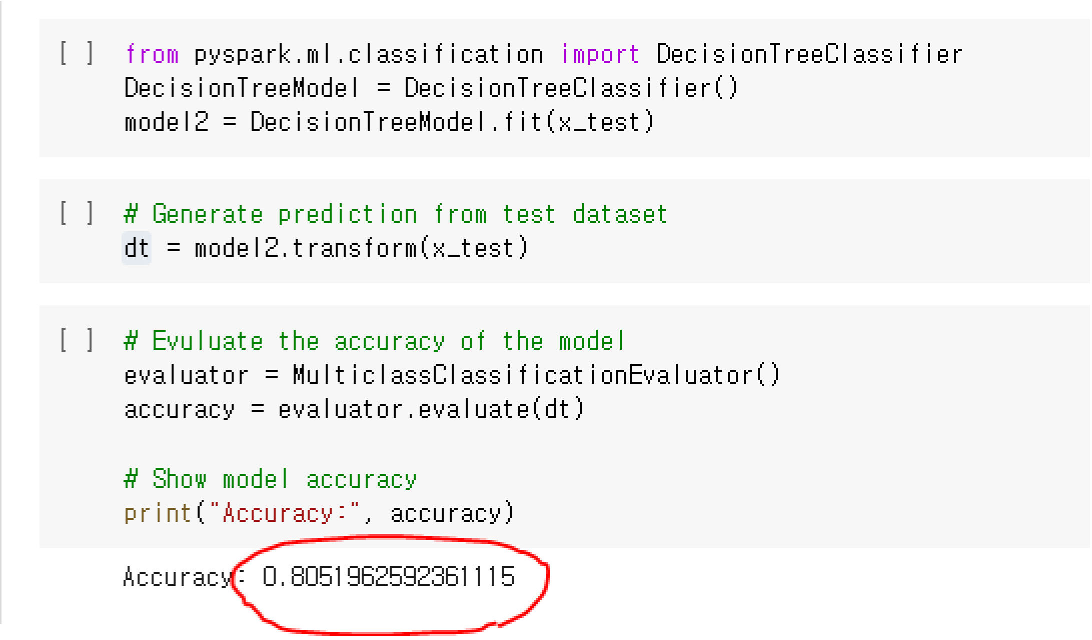
```
# select example rows to display.
predictions = model1.transform(x_test)
predictions.show(5)
```

```
+-----+--------------------+--------------------+--------------------+----------+
|label|            features|       rawPrediction|         probability|prediction|
+-----+--------------------+--------------------+--------------------+----------+
|  0.0| [17.0,0.0,4.0,45.0]|[-125.35616342715...|[1.0,9.9608115215...|       0.0|
|  0.0| [17.0,0.0,5.0,15.0]|[-74.640681407274...|[1.0,1.5075237380...|       0.0|
|  0.0| [17.0,0.0,5.0,25.0]|[-92.611207198269...|[1.0,2.4944842373...|       0.0|
|  0.0| [17.0,0.0,5.0,35.0]|[-110.58173298926...|[1.0,4.1275977641...|       0.0|
|  0.0| [17.0,0.0,5.0,40.0]|[-119.56699588476...|[1.0,5.3095250096...|       0.0|
+-----+--------------------+--------------------+--------------------+----------+
only showing top 5 rows
```

```
from pyspark.ml.evaluation import MulticlassClassificationEvaluator

# compute accuracy on the test set
evaluator = MulticlassClassificationEvaluator(labelCol="label", predictionCol="prediction",
                                              metricName="accuracy")
accuracy = evaluator.evaluate(predictions)
print("Test set accuracy = " + str(accuracy))
```

```
Test set accuracy = 0.7765089722675367
```

Decision Tree



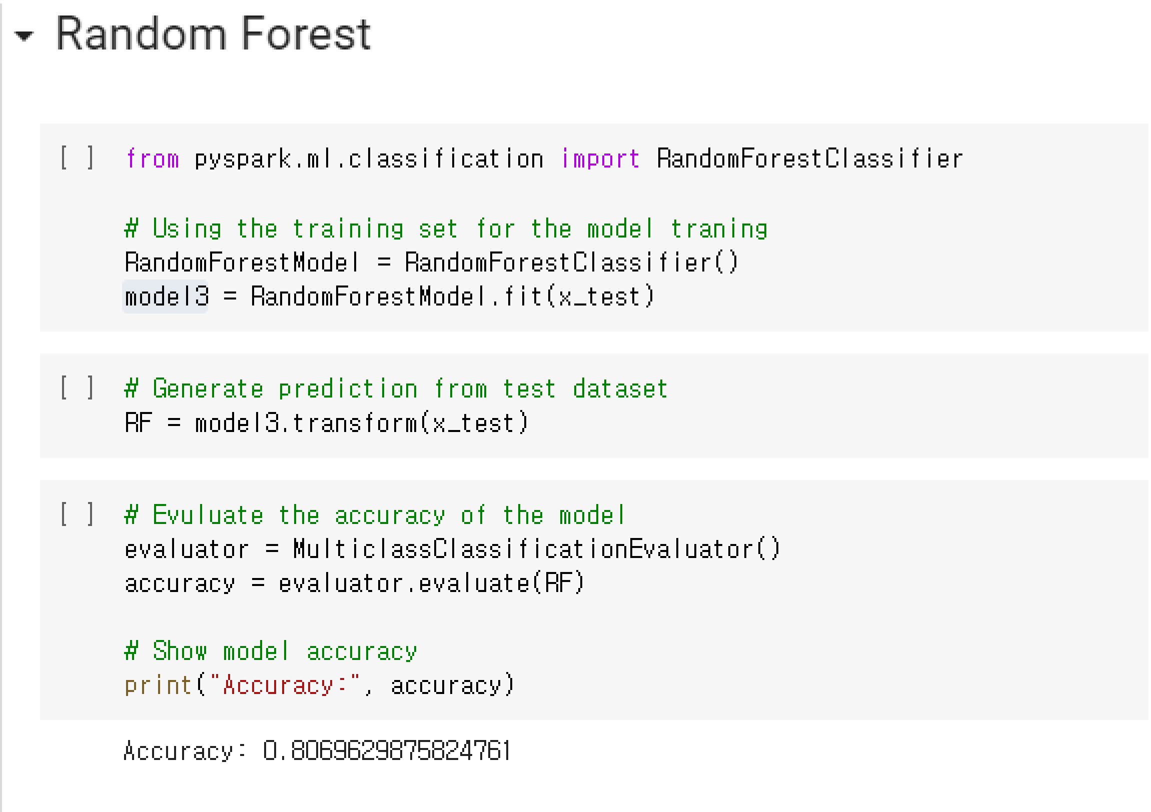
```
from pyspark.ml.classification import DecisionTreeClassifier
DecisionTreeModel = DecisionTreeClassifier()
model2 = DecisionTreeModel.fit(x_test)
```

```
# Generate prediction from test dataset
dt = model2.transform(x_test)
```

```
# Evuluate the accuracy of the model
evaluator = MulticlassClassificationEvaluator()
accuracy = evaluator.evaluate(dt)

# Show model accuracy
print("Accuracy:", accuracy)
```

```
Accuracy: 0.8051962592361115
```

Random Forest



Random Forest

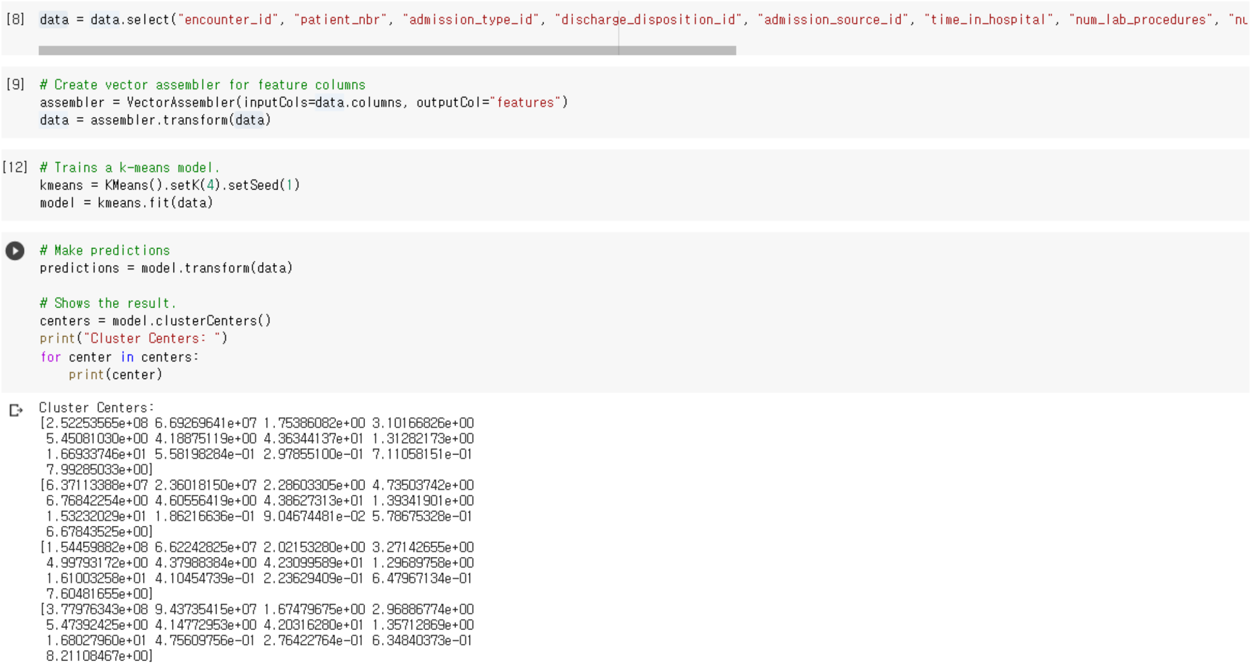
```
[ ]  from pyspark.ml.classification import RandomForestClassifier

     # Using the training set for the model traning
     RandomForestModel = RandomForestClassifier()
     model3 = RandomForestModel.fit(x_test)
```

```
[ ]  # Generate prediction from test dataset
     RF = model3.transform(x_test)
```

```
[ ]  # Evuluate the accuracy of the model
     evaluator = MulticlassClassificationEvaluator()
     accuracy = evaluator.evaluate(RF)

     # Show model accuracy
     print("Accuracy:", accuracy)
```

```
     Accuracy: 0.8069629875824761
```

Part 2 Clustering



```
[8]  data = data.select("encounter_id", "patient_nbr", "admission_type_id", "discharge_disposition_id", "admission_source_id", "time_in_hospital", "num_lab_procedures", "nu
```

```
[9]  # Create vector assembler for feature columns
     assembler = VectorAssembler(inputCols=data.columns, outputCol="features")
     data = assembler.transform(data)
```

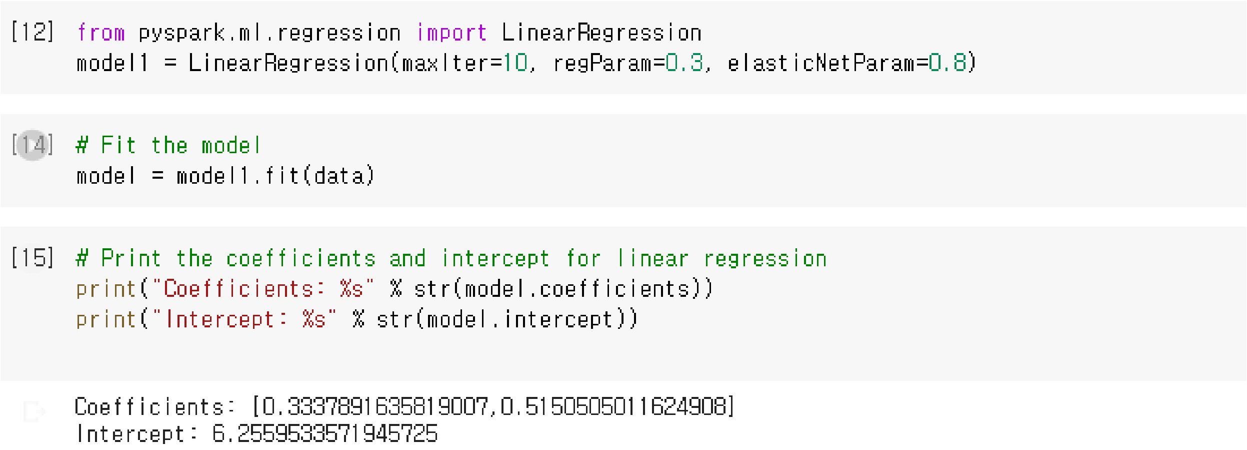
```
[12] # Trains a k-means model.
     kmeans = KMeans().setK(4).setSeed(1)
     model = kmeans.fit(data)
```

```
# Make predictions
predictions = model.transform(data)

# Shows the result.
centers = model.clusterCenters()
print("Cluster Centers: ")
for center in centers:
    print(center)
```

```
Cluster Centers:
[2.52253565e+08 6.69269641e+07 1.75386082e+00 3.10166826e+00
 5.45081030e+00 4.18875119e+00 4.36344137e+01 1.31282173e+00
 1.66933746e+01 5.58198284e-01 2.97855100e-01 7.11058151e-01
 7.99285033e+00]
[6.37113388e+07 2.36018150e+07 2.28603305e+00 4.73503742e+00
 6.76842254e+00 4.60556419e+00 4.38627313e+01 1.39341901e+00
 1.53232029e+01 1.86216636e-01 9.04674481e-02 5.78675328e-01
 6.67843525e+00]
[1.54459882e+08 6.62242825e+07 2.02153280e+00 3.27142655e+00
 4.99793172e+00 4.37988384e+00 4.23099589e+01 1.29689758e+00
 1.61003258e+01 4.10454739e-01 2.23629409e-01 6.47967134e-01
 7.60481655e+00]
[3.77976343e+08 9.43735415e+07 1.67479675e+00 2.96886774e+00
 5.47392425e+00 4.14772953e+00 4.20316280e+01 1.35712869e+00
 1.68027960e+01 4.75609756e-01 2.76422764e-01 6.34840373e-01
 8.21108467e+00]
```

Part 3 Regression

Linear Regression

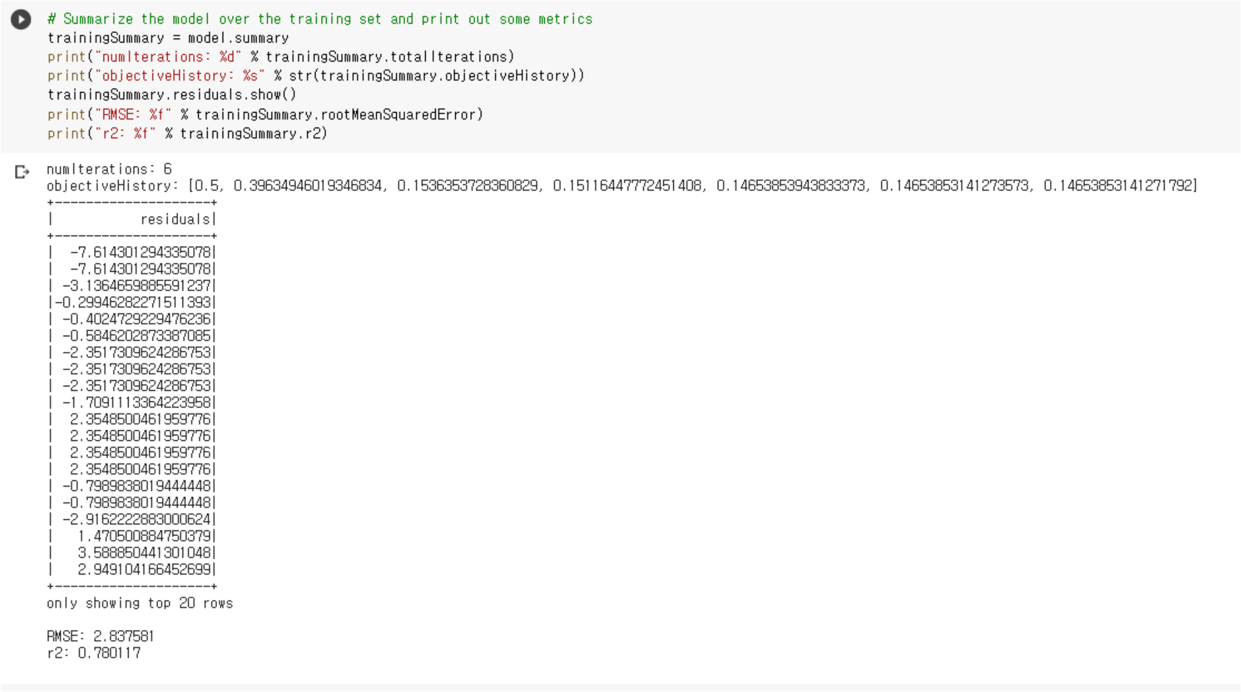


```
[12] from pyspark.ml.regression import LinearRegression
     model1 = LinearRegression(maxIter=10, regParam=0.3, elasticNetParam=0.8)
```

```
[14] # Fit the model
     model = model1.fit(data)
```

```
[15] # Print the coefficients and intercept for linear regression
     print("Coefficients: %s" % str(model.coefficients))
     print("Intercept: %s" % str(model.intercept))
```

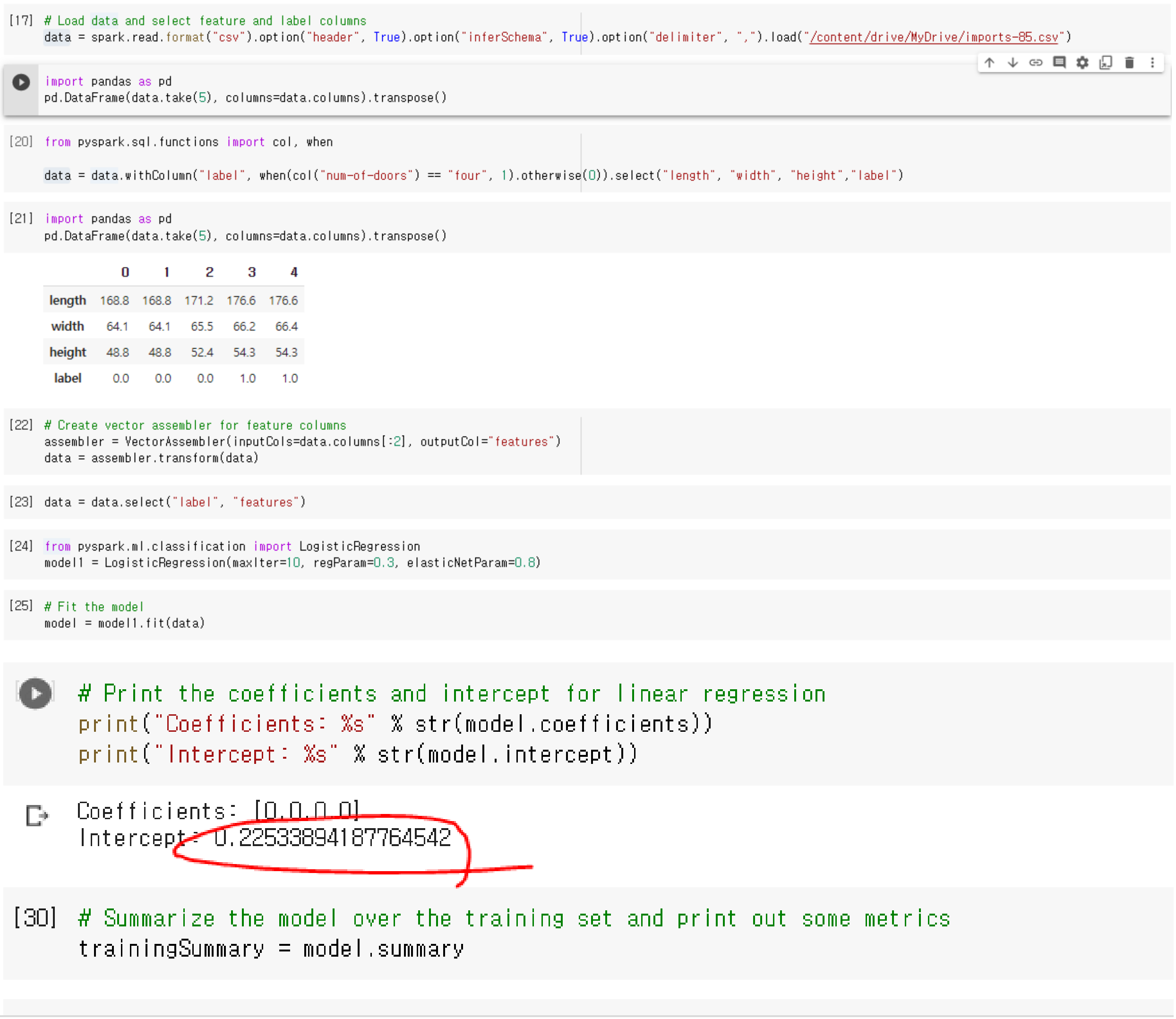
```
Coefficients: [0.3337891635819007,0.5150505011624908]
Intercept: 6.2559533571945725
```



```
# Summarize the model over the training set and print out some metrics
trainingSummary = model.summary
print("numIterations: %d" % trainingSummary.totalIterations)
print("objectiveHistory: %s" % str(trainingSummary.objectiveHistory))
trainingSummary.residuals.show()
print("RMSE: %f" % trainingSummary.rootMeanSquaredError)
print("r2: %f" % trainingSummary.r2)
```

```
numIterations: 6
objectiveHistory: [0.5, 0.39634946019346834, 0.1536353728360829, 0.15116447772451408, 0.14653853943833373, 0.14653853141273573, 0.14653853141271792]
+--------------------+
|           residuals|
+--------------------+
|  -7.614301294335078|
|  -7.614301294335078|
| -3.1364659885591237|
|-0.29946282271511393|
| -0.4024729229476236|
| -0.5846202873387085|
| -2.3517309624286753|
| -2.3517309624286753|
| -2.3517309624286753|
| -1.7091113364223958|
|  2.3548500461959776|
|  2.3548500461959776|
|  2.3548500461959776|
|  2.3548500461959776|
| -0.7989838019444448|
| -0.7989838019444448|
| -2.9162222883000624|
|   1.470500884750379|
|   3.588850441301048|
|   2.949104166452699|
+--------------------+
only showing top 20 rows

RMSE: 2.837581
r2: 0.780117
```

Logistic Regression



```
[17] # Load data and select feature and label columns
     data = spark.read.format("csv").option("header", True).option("inferSchema", True).option("delimiter", ",").load("/content/drive/MyDrive/imports-85.csv")
```

```
import pandas as pd
pd.DataFrame(data.take(5), columns=data.columns).transpose()
```

```
[20] from pyspark.sql.functions import col, when

     data = data.withColumn("label", when(col("num-of-doors") == "four", 1).otherwise(0)).select("length", "width", "height","label")
```

```
[21] import pandas as pd
     pd.DataFrame(data.take(5), columns=data.columns).transpose()
```

	0	1	2	3	4
length	168.8	168.8	171.2	176.6	176.6
width	64.1	64.1	65.5	66.2	66.4
height	48.8	48.8	52.4	54.3	54.3
label	0.0	0.0	0.0	1.0	1.0

```
[22] # Create vector assembler for feature columns
     assembler = VectorAssembler(inputCols=data.columns[:2], outputCol="features")
     data = assembler.transform(data)
```

```
[23] data = data.select("label", "features")
```

```
[24] from pyspark.ml.classification import LogisticRegression
     model1 = LogisticRegression(maxIter=10, regParam=0.3, elasticNetParam=0.8)
```

```
[25] # Fit the model
     model = model1.fit(data)
```

```
# Print the coefficients and intercept for linear regression
print("Coefficients: %s" % str(model.coefficients))
print("Intercept: %s" % str(model.intercept))
```

```
Coefficients: [0.0,0.0]
Intercept: 0.22533894187764542
```

```
[30] # Summarize the model over the training set and print out some metrics
     trainingSummary = model.summary
```