CIS5560 Term Project Tutorial

Lab Tutorial

Ramdas, Priya (pramdas2@calstatela.edu), Parekh, Heta (hparekh2@calstatela.edu),
Kang, Jo In(kjikji956@gmail.com)
05/14/2022

Cargurus Used Cars Analysis Using Machine Learning Models

Authors: Kang, Jo In; Parekh, Heta; Ramdas Priya
Instructor: Jongwook Woo
Date: 05/14/2022

Objectives:

This hands-on lab is divided into two parts:

1. Build machine learning models (*Linear Regression, Recommendation Model, Random Forest, Gradient Boost Tree, Factorization Machines learning*) on Databricks using the sample size.
2. Evaluate the models with Coefficient of Determination and Root Mean Square Error on Spark CLI with the whole dataset.

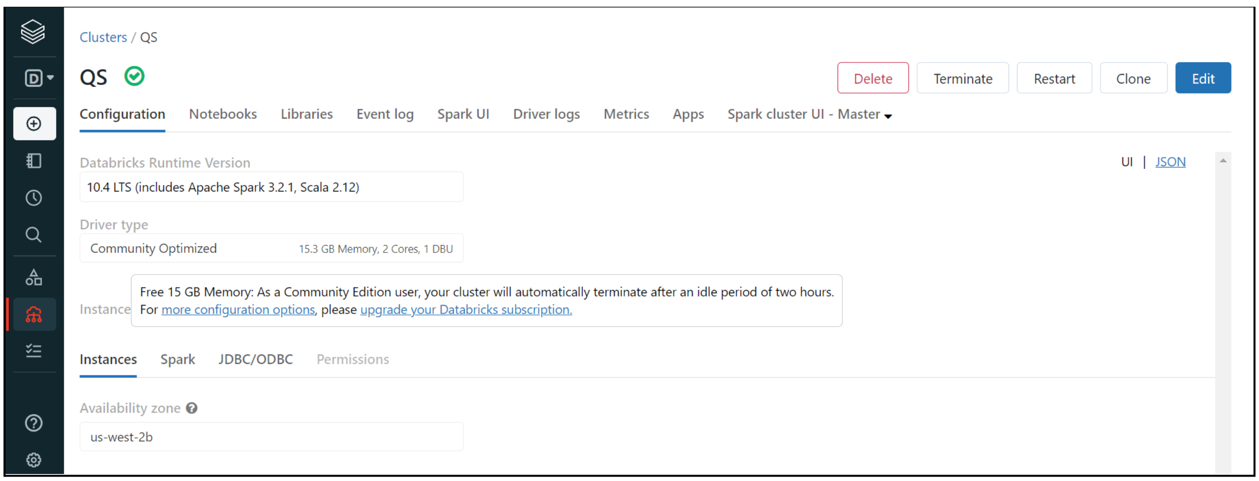
Platform Specification

Cluster Version	Hadoop 3.2.1-amazon-3.1
Number of Nodes	5
Memory size	30874 KB
CPU	8
CPU Speed	2.20 GHz
HDFS capacity	147 GB
Storage	481 GB

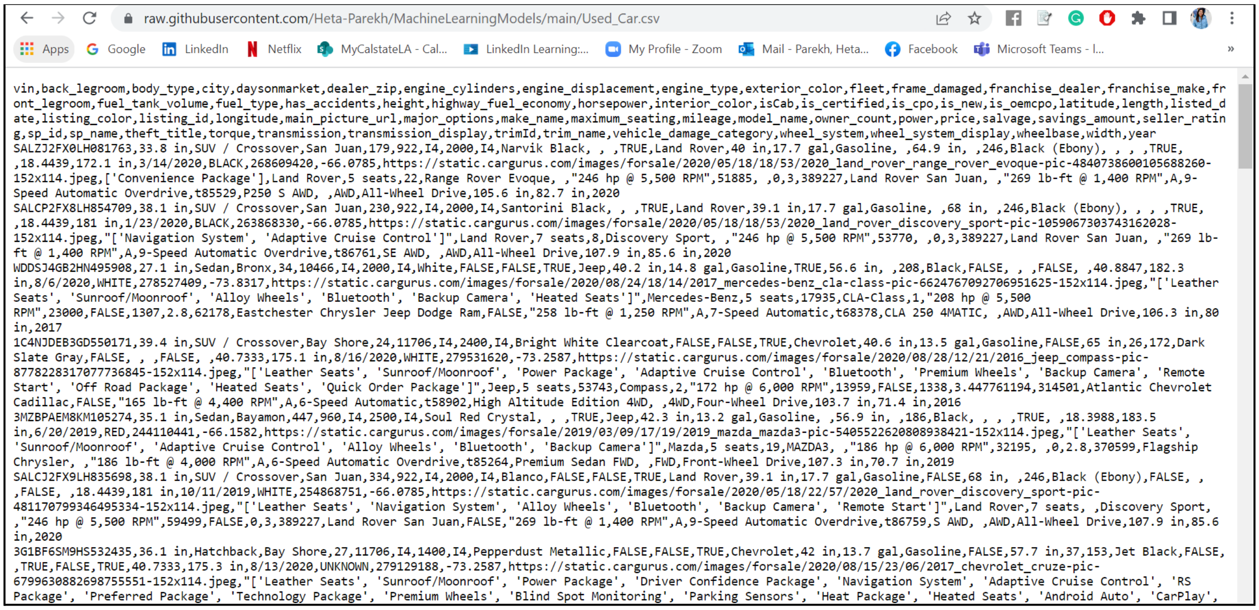
Part 1: Build machine learning models on Databricks using the sample size.

General Setup required for all the regression models:

1. Sign up on Databricks Community Edition
    1) You need to sign in https://community.cloud.databricks.com/ to run Spark on Data Bricks Cloud Computing.
    2) In the sidebar of the page above, click Compute.
    3) On the Compute page, click Create Cluster.
        - Create a cluster [1] with "select 9.1 LTS (Scala 2.12, Spark 3.1.2)".
        - It may take 5-10 minutes to see the cluster.

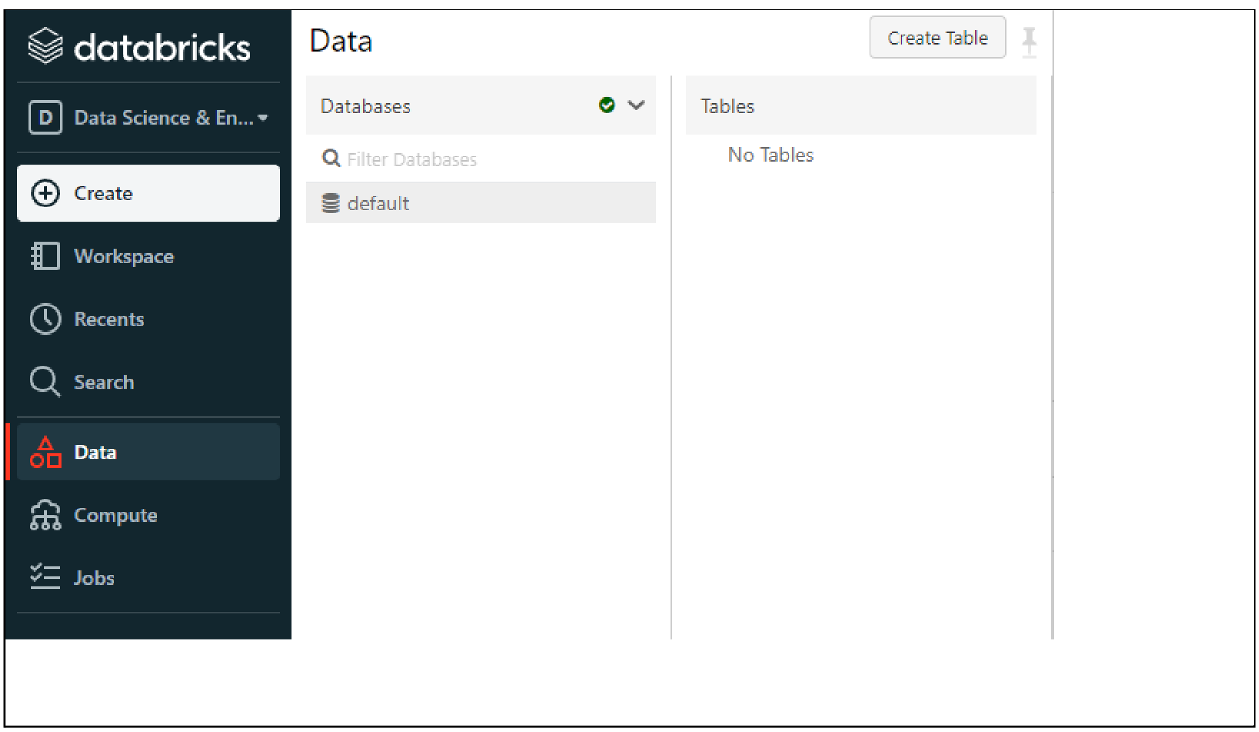


2. Create a notebook and Load Data File.
    1) Open a web browser. Select and go to the data link on your web. Browse https://raw.githubusercontent.com/Heta-Parekh/MachineLearningModels/main/Used_Car.csv. Then download the file "Used_Car.csv" to your computer using the right mouse button > Save As



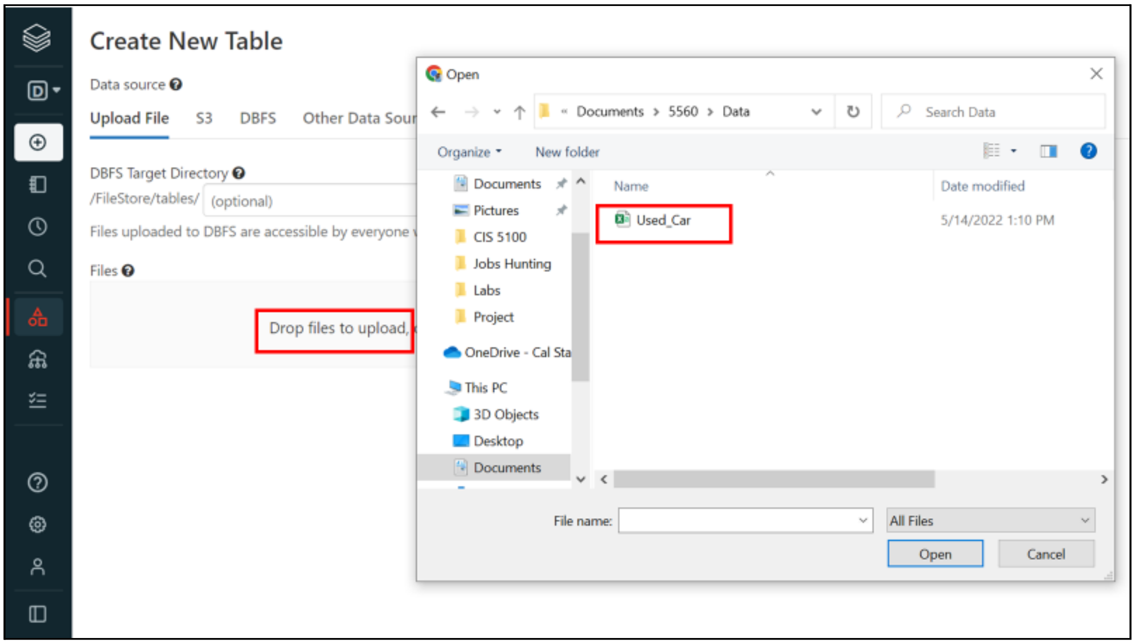
You must remember the data file location on your computer to which you downloaded the file.

2) Go to your Databricks page and select the Data menu in the left menu bar.

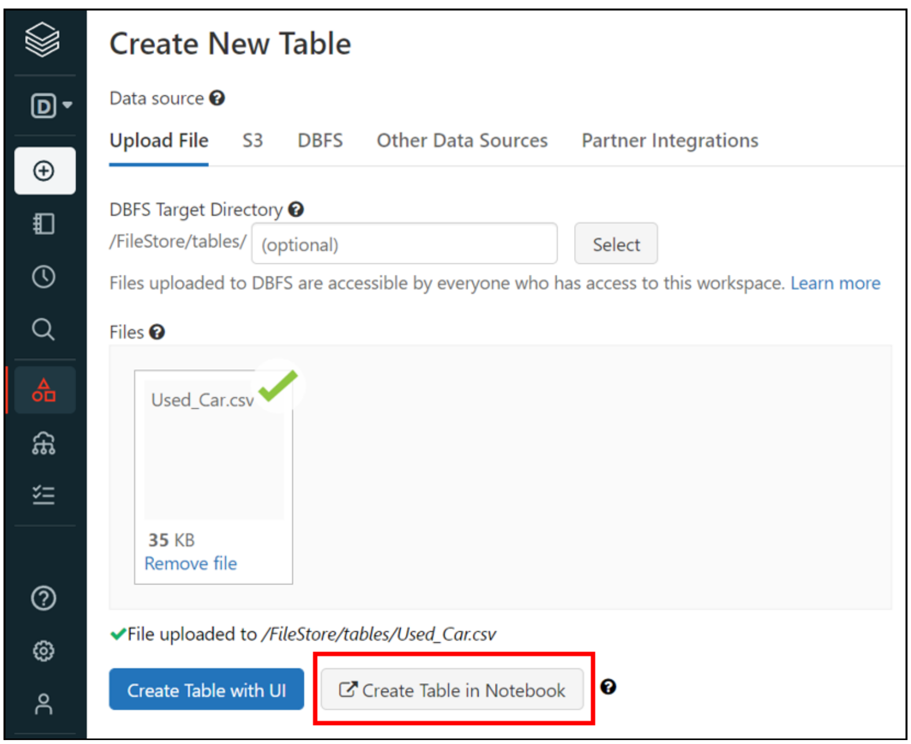


3) Then select "Drop files to upload or click to browse." Once a file explorer opens, you have to explore your PC to find out and select the "*Used_Car.csv*" file that you downloaded above.

Then, select Open



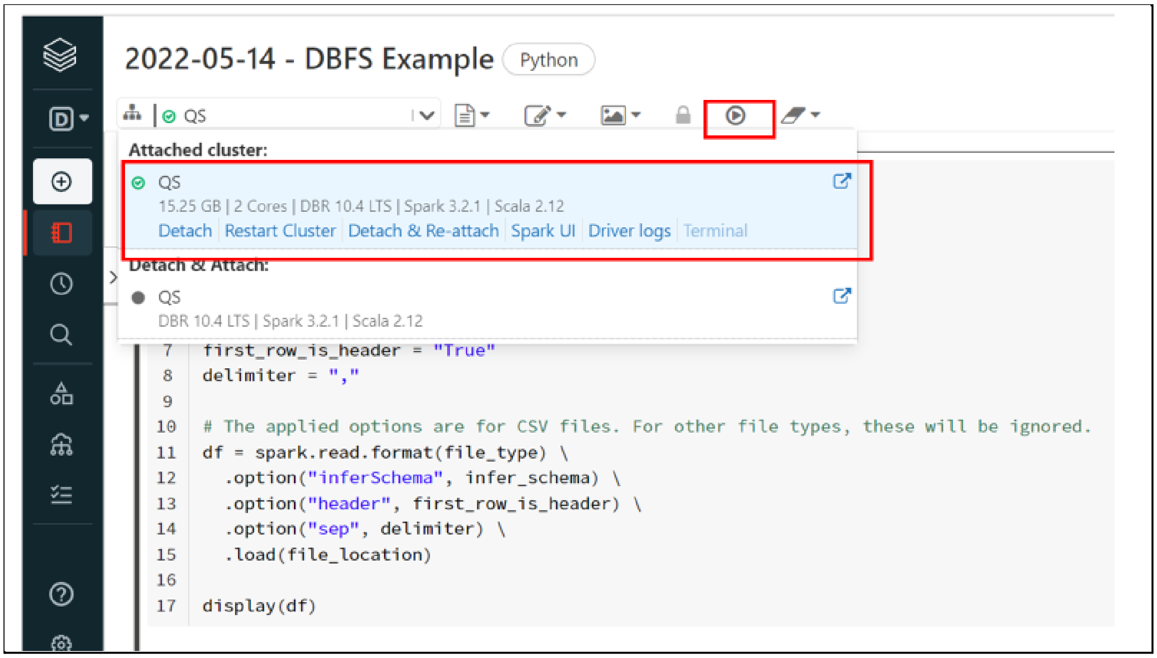
4) Select "Create Table in Notebook."



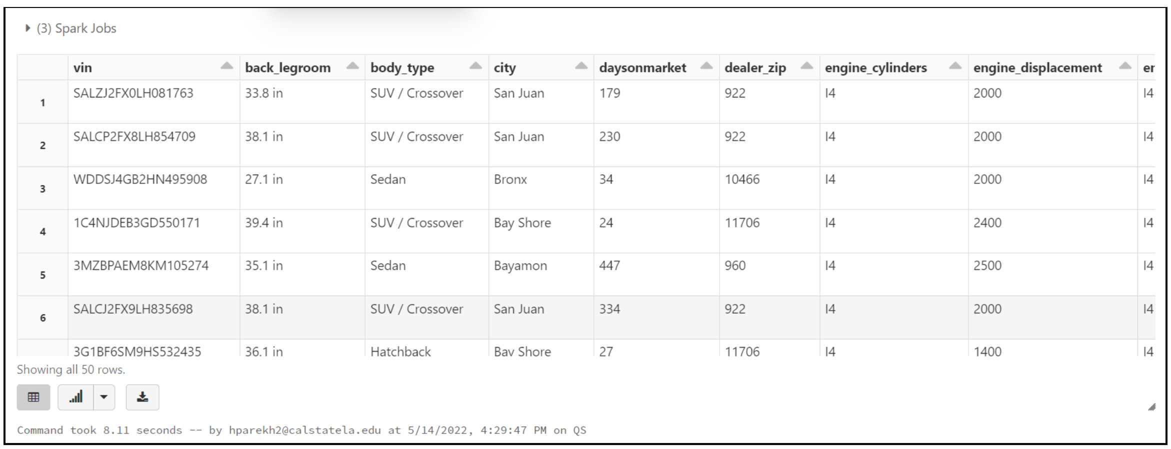
5) Change the code in the cell to true from false as follows:

```
# CSV options
infer_schema = "True"
first_row_is_header = "True"
```

6) Attach the cluster and select "Run all" to run the entire cells.



7) You can see the following output after command 2:


(3) Spark Jobs

	vin	back_legroom	body_type	city	daysonmarket	dealer_zip	engine_cylinders	engine_displacement	er
1	SALZJ2FX0LH081763	33.8 in	SUV / Crossover	San Juan	179	922	I4	2000	I4
2	SALCP2FX8LH854709	38.1 in	SUV / Crossover	San Juan	230	922	I4	2000	I4
3	WDDSJ4GB2HN495908	27.1 in	Sedan	Bronx	34	10466	I4	2000	I4
4	1C4NJDEB3GD550171	39.4 in	SUV / Crossover	Bay Shore	24	11706	I4	2400	I4
5	3MZBPAEM8KM105274	35.1 in	Sedan	Bayamon	447	960	I4	2500	I4
6	SALCJ2FX9LH835698	38.1 in	SUV / Crossover	San Juan	334	922	I4	2000	I4
	3G1BF6SM9HS532435	36.1 in	Hatchback	Bay Shore	27	11706	I4	1400	I4

Showing all 50 rows.

Command took 8.11 seconds -- by hparekh2@calstatela.edu at 5/14/2022, 4:29:47 PM on QS

8) Import the following libraries in a new cell and run the command:

```
from pyspark.sql.functions import col
from functools import reduce
import pyspark.sql.functions as F
from pyspark.sql.types import DoubleType,IntegerType
```

```
Cmd 6

from pyspark.sql.functions import col
from functools import reduce
import pyspark.sql.functions as F
from pyspark.sql.types import DoubleType,IntegerType

Command took 0.04 seconds -- by hparekh2@calstatela.edu at 5/14/2022, 4:35:42 PM on QS
```

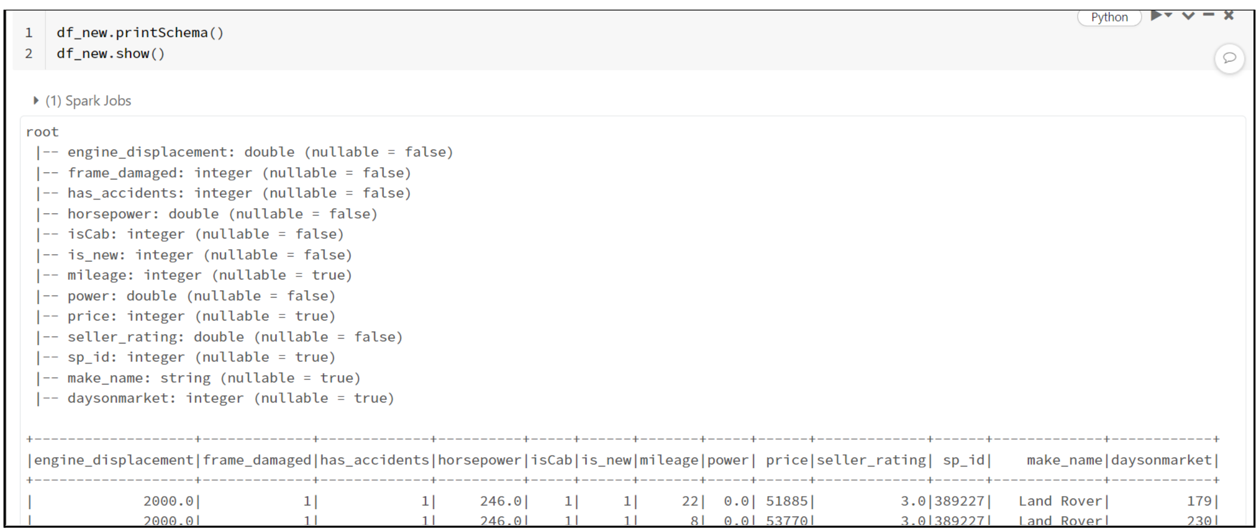
9) Copy the below code to a new cell and run them to prepare and cleanse the data and convert them to desired datatypes and run it.

```
df_new = df.select (col('engine_displacement'), col('frame_damaged')
,col('has_accidents')
,col('horsepower'),col('isCab'),col('is_new'),col('mileage'),col('power'),col('price'),col('se
ller_rating'),col('sp_id'),col('make_name'),col('daysonmarket'))
df_new = df_new.withColumn ("engine_displacement",
col("engine_displacement").cast(DoubleType()))
df_new = df_new.withColumn ("horsepower", col("horsepower").cast(DoubleType()))
df_new = df_new.withColumn ("power", col("power").cast(DoubleType()))
df_new = df_new.withColumn ("mileage", col("mileage").cast(IntegerType()))
df_new = df_new.withColumn ("price",col("price").cast(IntegerType()))
df_new = df_new.withColumn ("seller_rating",col("seller_rating").cast(DoubleType()))

cols = ['is_new']
col2 = ['frame_damaged','has_accidents','isCab']

df_new= reduce(lambda df_new, c: df_new.withColumn(c, F.when(df_new[c] ==
'False', 0).otherwise(1)), cols, df_new)
df_new=  df_new.na.fill(value=0,subset=["mileage"])
df_new = reduce(lambda df_new, c: df_new.withColumn(c, F.when(df_new[c]==
'False', 2).when(df_new[c]== 'True', 0).otherwise(1)), col2, df_new)
df_new= df_new.na.fill(value=0,subset=["engine_displacement"])
df_new= df_new.na.fill(value=0,subset=["horsepower"])
df_new= df_new.na.fill(value=0,subset=["power"])
df_new= df_new.na.fill(value=0,subset=["seller_rating"])
df_new= df_new.na.fill(value=0,subset=["price"])
```

```
df_new = df_new.withColumn ("is_new",col("is_new").cast(IntegerType()))
df_new = df_new.withColumn
("frame_damaged",col("frame_damaged").cast(IntegerType()))
df_new = df_new.withColumn
("has_accidents",col("has_accidents").cast(IntegerType()))
df_new = df_new.withColumn ("isCab",col("isCab").cast(IntegerType()))
df_new = df_new.select('*').where(col("price")>100)
df_new = df_new.select('*').where(col("price")<100000)
df_new = df_new.select('*').where(col("engine_displacement")>0)
df_new = df_new.select('*').where(col("horsepower")>0)
```

10) To view the output, run the following code in a new cell:

```
df_new.printSchema()
df_new.show()
```



Linear Regression Model:

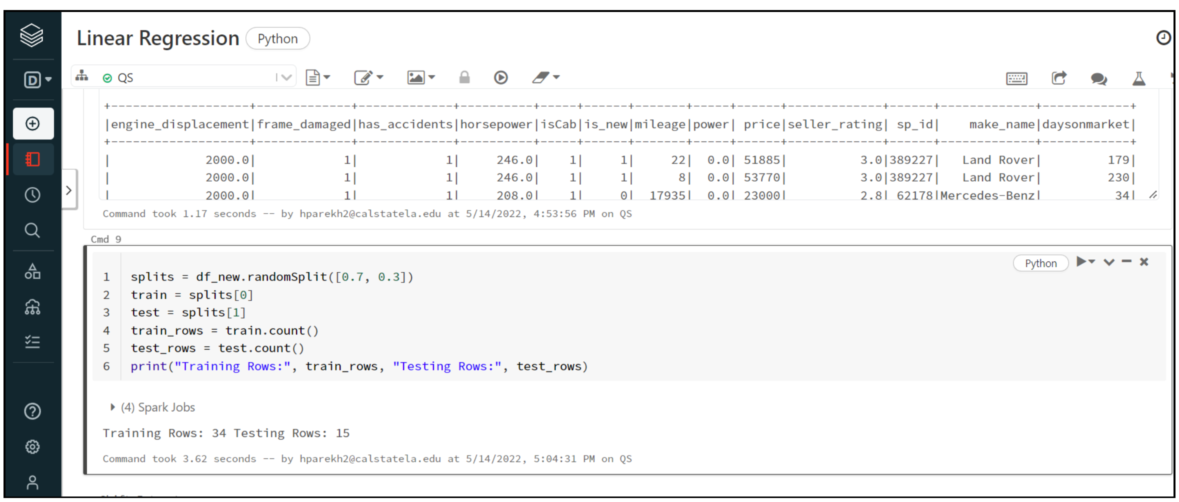
A) CRun the steps 1 - 10 as shown above and rename the notebook as "Linear Regression"

B) Import this libraries and so, copy this code to the existing cell no. 6 and run it:

```
from pyspark.sql.functions import col
from functools import reduce
import pyspark.sql.functions as F
from pyspark.sql.types import DoubleType,IntegerType
from pyspark.sql.types import *
from pyspark.sql.functions import *
from pyspark.ml.regression import LinearRegression
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.linalg import Vectors
from pyspark.ml.evaluation import RegressionEvaluator
from pyspark.ml import Pipeline
```

C) Copy the below code to split the data to 70 and 30%

```
splits = df_new.randomSplit([0.7, 0.3])
train = splits[0]
test = splits[1]
train_rows = train.count()
test_rows = test.count()
print("Training Rows:", train_rows, "Testing Rows:", test_rows)
```



D) Copy the below code to define the pipeline and run it.

```
assembler = VectorAssembler(inputCols =["engine_displacement","is_new",
```

```
"mileage", "frame_damaged", "has_accidents",
"seller_rating","isCab","horsepower"], outputCol="features")
```

Cmd 10

```
assembler = VectorAssembler(inputCols =["engine_displacement","is_new", "mileage", "frame_damaged", "has_accidents",
"seller_rating","isCab","horsepower"], outputCol="features")
```

Command took 0.10 seconds -- by hparekh2@calstatela.edu at 5/14/2022, 5:12:57 PM on QS

E) Copy the below code and run it to train the linear regression model.

```
lr = LinearRegression (labelCol="price", featuresCol="features",maxIter=10,
regParam=0.8)
pipeline = Pipeline(stages=[assembler, lr])
model = pipeline.fit(train)
```

Cmd 11

```
lr = LinearRegression (labelCol="price", featuresCol="features",maxIter=10, regParam=0.8)
pipeline = Pipeline(stages=[assembler, lr])
model = pipeline.fit(train)
```

▸ (2) Spark Jobs

Command took 3.77 seconds -- by hparekh2@calstatela.edu at 5/14/2022, 5:21:16 PM on QS

F) Copy the below code and run it to test the model.

```
prediction = model.transform(test)
predicted = prediction.select("features", "prediction", "price")
predicted = predicted.drop("features")
predicted.show()
```

```
prediction = model.transform(test)
predicted = prediction.select("features", "prediction", "price")
predicted = predicted.drop("features")
predicted.show()
```

(1) Spark Jobs

```
+-----------------+------+
|       prediction| price|
+-----------------+------+
|16985.487552547333| 14224|
| 38837.37957923651| 29485|
| 5595.265708255223| 17926|
| 6703.964479657365|  7703|
|22785.861873461345| 23000|
|23360.771853510818| 18900|
|27188.971474350605| 21995|
|32087.837416581046| 59499|
| 6735.868672735125|  8999|
|  9348.86785923529| 13959|
|13049.529564719118| 21300|
| 32112.14855067952| 21595|
|  69995.5144137861|101737|
| 42566.44545807682| 37937|
| 19479.70291811989| 41823|
+-----------------+------+
```

G) Copy the below code and run it to calculate the Root Mean Square Error (RMSE) and Coefficient of Determination (R2) for the linear regression model.

```
lr_evaluator = RegressionEvaluator
(predictionCol="prediction",labelCol="price",metricName="r2")
print("R Squared (R2) on test data = %g" %lr_evaluator.evaluate(prediction))
lr_evaluator = RegressionEvaluator(labelCol="price",
predictionCol="prediction", metricName="rmse")
print("RMSE: %f" % lr_evaluator.evaluate(prediction))
```

Cmd 13

```
lr_evaluator = RegressionEvaluator (predictionCol="prediction",labelCol="price",metricName="r2")
print("R Squared (R2) on test data = %g" %lr_evaluator.evaluate(prediction))
lr_evaluator = RegressionEvaluator(labelCol="price", predictionCol="prediction", metricName="rmse")
print("RMSE: %f" % lr_evaluator.evaluate(prediction))
```

(2) Spark Jobs

```
R Squared (R2) on test data = 0.660423
RMSE: 13612.895275
```

Command took 1.74 seconds -- by hparekh2@calstatela.edu at 5/14/2022, 5:30:09 PM on QS

Random Forest Regression

A) Run the steps 1 - 10 as shown above and rename the notebook as "Random_Forest_Regression"

B) Import this libraries and so, copy this code to the existing cell no. 6 and run it:

```
from pyspark.ml.regression import RandomForestRegressor
```

C) Copy the below code to split the data to 70 and 30%

```
splits = df_new.randomSplit([0.7, 0.3])
train = splits[0]
test = splits[1]
train_rows = train.count()
test_rows = test.count()
print("Training Rows:", train_rows, "Testing Rows:", test_rows)
```

```
Python
splits = df_new.randomSplit([0.7, 0.3])
train = splits[0]
test = splits[1]
train_rows = train.count()
test_rows = test.count()
print("Training Rows:", train_rows, "Testing Rows:", test_rows)

▸ (4) Spark Jobs
Training Rows: 36 Testing Rows: 13
Command took 4.60 seconds -- by pramdas2@calstatela.edu at 18/05/2022, 17:23:06 on My Cluster
```

D) Copy the below code to define the pipeline and run it.

```
assembler = VectorAssembler(inputCols =["engine_displacement","is_new",
"mileage", "frame_damaged", "has_accidents", "seller_rating","isCab","horsepower"],
outputCol="features")
```

```
Python
"""## Define the Pipeline"""
assembler = VectorAssembler(inputCols =["engine_displacement","is_new", "mileage", "frame_damaged", "has_accidents",
"seller_rating","isCab","horsepower"], outputCol="features")

Command took 0.21 seconds -- by pramdas2@calstatela.edu at 18/05/2022, 17:23:06 on My Cluster
```

E) Copy the below code and run it to train the linear regression model.

```
rf = RandomForestRegressor(labelCol="price",featuresCol="features",numTrees=10,
```

```
maxDepth=5)
pipeline = Pipeline(stages=[assembler, rf])
model = pipeline.fit(train)
```

```
"""## Train a Regression Model"""
rf = RandomForestRegressor(labelCol="price",featuresCol="features",numTrees=10, maxDepth=5)
pipeline = Pipeline(stages=[assembler, rf])
model = pipeline.fit(train)
```

▸ (8) Spark Jobs

Command took 6.73 seconds -- by pramdas2@calstatela.edu at 18/05/2022, 17:23:06 on My Cluster

F) Copy the below code and run it to test the model.

```
prediction = model.transform(test)
predicted = prediction.select("features", "prediction", "price")
predicted = predicted.drop("features")
predicted.show()
```

```
prediction = model.transform(test)
predicted = prediction.select("features", "prediction", "price")
predicted = predicted.drop("features")
predicted.show()
```

▸ (1) Spark Jobs

```
+------------------+------+
|        prediction| price|
+------------------+------+
|12145.971710526317|  2999|
| 17684.18726608187| 17926|
|11413.371710526317|  5499|
| 19856.89837719298| 13295|
| 19085.09337719298| 32439|
| 18216.46726608187|  8999|
| 25223.29837719298| 21681|
|27648.021710526315| 32195|
|22204.548377192983| 29800|
|23140.471710526315| 21495|
| 38971.53587719299|101737|
|10274.421710526316|  9750|
|11888.005043859648|  5250|
+------------------+------+
```

Command took 3.10 seconds -- by pramdas2@calstatela.edu at 18/05/2022, 17:23:06 on My Cluster

G) Copy the below code and run it to calculate the Root Mean Square Error (RMSE) and Coefficient of Determination (R2) for the regression model.

```
rf_evaluator =
RegressionEvaluator(predictionCol="prediction",labelCol="price",metricName
="r2")
print("R Squared (R2) on test data = %g" %rf_evaluator.evaluate(prediction))

rf_evaluator = RegressionEvaluator(labelCol="price",
predictionCol="prediction", metricName="rmse")
```

```
print("RMSE: %f" % rf_evaluator.evaluate(prediction))
```

```
rf_evaluator = RegressionEvaluator(predictionCol="prediction",labelCol="price",metricName="r2")
print("R Squared (R2) on test data = %g" %rf_evaluator.evaluate(prediction))

rf_evaluator = RegressionEvaluator(labelCol="price", predictionCol="prediction", metricName="rmse")
print("RMSE: %f" % rf_evaluator.evaluate(prediction))
```

(2) Spark Jobs

```
R Squared (R2) on test data = 0.432578
RMSE: 18610.409329
```

Command took 1.83 seconds -- by pramdas2@calstatela.edu at 18/05/2022, 17:23:06 on My Cluster

Recommendation Model:

A) Run the steps 1 - 10 as shown above and rename the notebook as "Recommendation_Regression"

B) Import this libraries and so, copy this code to the existing cell no. 6 and run it:

```
from pyspark.ml.recommendation import ALS
```

C) Copy the below code to select only required fields for regression.

```
data =
df_new.select("sp_id","make_name","daysonmarket","seller_rating").distinct()

data.show()
```

```
data = df_new.select("sp_id","make_name","daysonmarket","seller_rating").distinct()
display(data)
```

(2) Spark Jobs

	sp_id	make_name	daysonmarket	seller_rating
1	432605	BMW	18	2.963636364
2	314501	Chevrolet	27	3.447761194
3	351457	Chevrolet	21	3.577777778
4	351457	Honda	3	3.577777778
5	389227	Land Rover	334	3
6	339626	Kia	19	3.647058824

Showing all 48 rows.

Command took 1.59 seconds -- by pramdas2@calstatela.edu at 18/05/2022, 17:48:37 on My Cluster

D) Copy the below code to split the data to 70 and 30%

```
splits = data.randomSplit([0.7, 0.3])
train = splits[0].withColumnRenamed("seller_rating","label")
test = splits[1].withColumnRenamed("seller_rating","trueLabel")
train_rows = train.count()
test_rows = test.count()
print("Training Rows:", train_rows, "Testing Rows:", test_rows)
```

Python

```
splits = data.randomSplit([0.7, 0.3])
train = splits[0].withColumnRenamed("seller_rating","label")
test = splits[1].withColumnRenamed("seller_rating","trueLabel")
train_rows = train.count()
test_rows = test.count()
print("Training Rows:", train_rows, "Testing Rows:", test_rows)
```

▸ (6) Spark Jobs

```
Training Rows: 37 Testing Rows: 11
```

Command took 2.40 seconds -- by pramdas2@calstatela.edu at 18/05/2022, 17:48:37 on My Cluster

E) Copy the below code to define the ALS and then, select the "Run Cell"

```
als = ALS(userCol = "sp_id",itemCol = "daysonmarket", ratingCol = "label")
```

Command took 0.20 seconds -- by pramdas2@calstatela.edu at 18/05/2022, 17:48:37 on My Cluster

F) Create a new cell and add the following code to create a parameter combination, which is to tune the model. Then, select "Run Cell":

```
paramGrid = ParamGridBuilder() \
                .addGrid(als.rank, [1]) \
                .addGrid(als.maxIter, [5]) \
                .addGrid(als.regParam, [0.3]) \
                .addGrid(als.alpha, [2.0]) \
                .build()
```

Python

```
# Commented out IPython magic to ensure Python compatibility.
# %pyspark
paramGrid = ParamGridBuilder() \
                    .addGrid(als.rank, [1]) \
                    .addGrid(als.maxIter, [5]) \
                    .addGrid(als.regParam, [0.3]) \
                    .addGrid(als.alpha, [2.0]) \
                    .build()
```

Command took 0.03 seconds -- by pramdas2@calstatela.edu at 18/05/2022, 17:48:37 on My Cluster

G) Create the new cell and add the below cell to add TrainValidationSplit.

```
cv = TrainValidationSplit(estimator=als, evaluator=RegressionEvaluator(),
estimatorParamMaps=paramGrid, trainRatio=0.8)
model = cv.fit(train)
```

```
cv = TrainValidationSplit(estimator=als, evaluator=RegressionEvaluator(), estimatorParamMaps=paramGrid,
trainRatio=0.8)
model = cv.fit(train)
```

Python

▸ (11) Spark Jobs

Command took 57.55 seconds -- by pramdas2@calstatela.edu at 18/05/2022, 17:48:37 on My Cluster

H) Copy the below code and run it to test the model.

```
prediction = model.transform(test)

# Remove NaN values from prediction (due to SPARK-14489) [1]
prediction = prediction.filter(prediction.prediction != float('nan'))

# Round floats to whole numbers
prediction = prediction.withColumn("prediction",
F.abs(F.round(prediction["prediction"],0)))

prediction.select("sp_id", "make_name", "prediction", "trueLabel").show(30,
truncate=False)
```

```
prediction = model.transform(test)

# Remove NaN values from prediction (due to SPARK-14489) [1]
prediction = prediction.filter(prediction.prediction != float('nan'))

# Round floats to whole numbers
prediction = prediction.withColumn("prediction", F.abs(F.round(prediction["prediction"],0)))

prediction.select("sp_id", "make_name", "prediction", "trueLabel").show(30, truncate=False)
```

Python

▸ (6) Spark Jobs

```
+------+-------------+----------+-----------+
|sp_id |make_name    |prediction|trueLabel  |
+------+-------------+----------+-----------+
|62178 |Jeep         |3.0       |2.8        |
|432605|Mercedes-Benz|3.0       |2.963636364|
+------+-------------+----------+-----------+
```

Command took 4.77 seconds -- by pramdas2@calstatela.edu at 18/05/2022, 17:48:37 on My Cluster

I) Copy the below code and run it to calculate the Root Mean Square Error (RMSE) and Coefficient of Determination (R2) for the regression model.

```
evaluator = RegressionEvaluator(labelCol="trueLabel", predictionCol="prediction",
metricName="rmse")
rmse = evaluator.evaluate(prediction)
print ("Root Mean Square Error (RMSE):", rmse)


evaluator = RegressionEvaluator(labelCol="trueLabel", predictionCol="prediction",
metricName="r2")
r2 = evaluator.evaluate(prediction)
print ("Coefficient of Determination (R2):", r2)
```

Python

```
evaluator = RegressionEvaluator(labelCol="trueLabel", predictionCol="prediction", metricName="rmse")
rmse = evaluator.evaluate(prediction)
print ("Root Mean Square Error (RMSE):", rmse)

evaluator = RegressionEvaluator(labelCol="trueLabel", predictionCol="prediction", metricName="r2")
r2 = evaluator.evaluate(prediction)
print ("Coefficient of Determination (R2):", r2)
```

(8) Spark Jobs

```
Root Mean Square Error (RMSE): 0.14373989359802058
Coefficient of Determination (R2): -2.086419737393685
```

Command took 6.26 seconds -- by pramdas2@calstatela.edu at 18/05/2022, 17:48:37 on My Cluster

Gradient Boost Tree:

A) Run the steps 1 - 10 as shown above and rename the notebook as “Gradient Boost Tree”

B) Import this libraries and so, copy this code to the existing cell no. 6 and run it:

```
from pyspark.sql.functions import col
from pyspark.sql import functions as F
from pyspark.sql.types import *
from pyspark.sql.functions import *
from functools import reduce
from pyspark.sql.types import DoubleType,IntegerType
from pyspark.context import SparkContext
from pyspark.sql.session import SparkSession
```

```
from pyspark.storagelevel import StorageLevel
from pyspark.ml.regression import GBTRegressor
from pyspark.ml import Pipeline
from pyspark.ml.evaluation import RegressionEvaluator
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.linalg import Vectors
```

C) Copy the below code to split the data to 70 and 30%

```
splits = df_new.randomSplit([0.7, 0.3])
train = splits[0]
test = splits[1]
train_rows = train.count()
test_rows = test.count()
print("Training Rows:", train_rows, "Testing Rows:", test_rows)
```

Cmd 4

```
splits = df_new.randomSplit([0.7, 0.3])
train = splits[0]
test = splits[1]
train_rows = train.count()
test_rows = test.count()
print("Training Rows:", train_rows, "Testing Rows:", test_rows)
```

▸ (4) Spark Jobs

```
Training Rows: 1306157 Testing Rows: 559210
```

Command took 30.11 seconds -- by kjikji956@gmail.com at 2022. 5. 17. 오전 12:00:08 on jjj

D) Copy the below code to define the pipeline and run it.

```
assembler = VectorAssembler(inputCols =["engine_displacement","is_new",
"mileage", "frame_damaged", "has_accidents",
"seller_rating","isCab","horsepower"], outputCol="features")
```

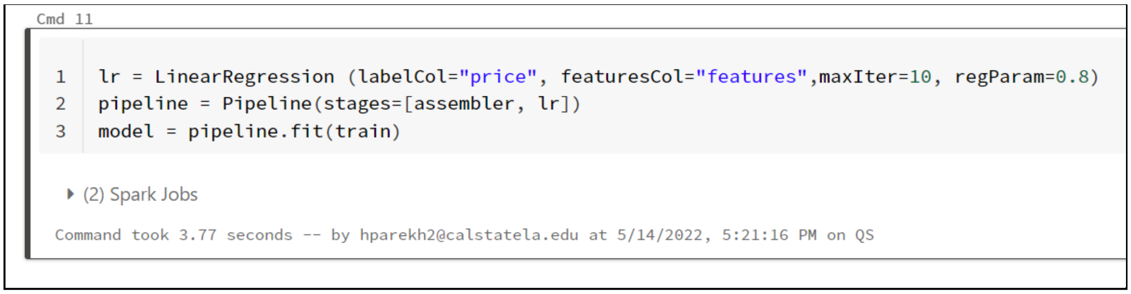
Cmd 5

```
assembler = VectorAssembler(inputCols =["engine_displacement","is_new", "mileage",
"frame_damaged", "has_accidents", "seller_rating","isCab","horsepower"], outputCol="features")
```

Command took 0.15 seconds -- by kjikji956@gmail.com at 2022. 5. 17. 오전 12:00:58 on jjj

E) Copy the below code and run it to train the Gradient Boost Tree.

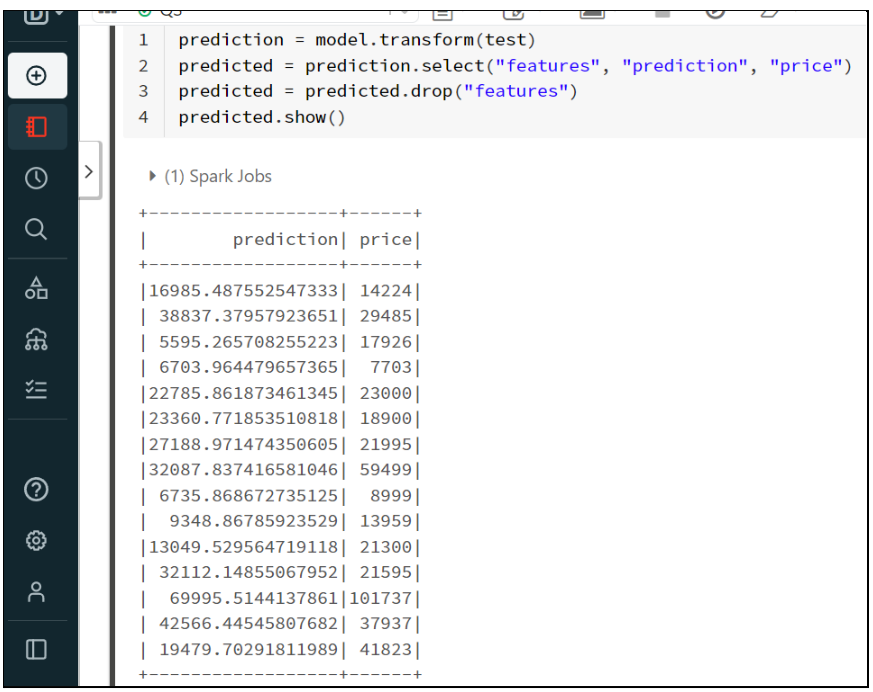
```
gbt = GBTRegressor(labelCol="price",featuresCol="features", maxIter=10)
pipeline = Pipeline(stages=[assembler, gbt])
model = pipeline.fit(train)
```



```
Cmd 11

lr = LinearRegression (labelCol="price", featuresCol="features",maxIter=10, regParam=0.8)
pipeline = Pipeline(stages=[assembler, lr])
model = pipeline.fit(train)

(2) Spark Jobs

Command took 3.77 seconds -- by hparekh2@calstatela.edu at 5/14/2022, 5:21:16 PM on QS
```

F) Copy the below code and run it to test the model.

```
prediction = model.transform(test)
predicted = prediction.select("features", "prediction", "price")
predicted = predicted.drop("features")
predicted.show()
```



```
prediction = model.transform(test)
predicted = prediction.select("features", "prediction", "price")
predicted = predicted.drop("features")
predicted.show()

(1) Spark Jobs

+------------------+------+
|        prediction| price|
+------------------+------+
|16985.487552547333| 14224|
| 38837.37957923651| 29485|
| 5595.265708255223| 17926|
| 6703.964479657365|  7703|
|22785.861873461345| 23000|
|23360.771853510818| 18900|
|27188.971474350605| 21995|
|32087.837416581046| 59499|
| 6735.868672735125|  8999|
|  9348.86785923529| 13959|
|13049.529564719118| 21300|
| 32112.14855067952| 21595|
|  69995.5144137861|101737|
| 42566.44545807682| 37937|
| 19479.70291811989| 41823|
+------------------+------+
```

G) Copy the below code and run it to calculate the Root Mean Square Error (RMSE) and Coefficient of Determination (R2) for the regression model.

```
gbt_evaluator =
RegressionEvaluator(predictionCol="prediction",labelCol="price",metricName
="r2")
print("R Squared (R2) on test data = %g" %gbt_evaluator.evaluate(prediction))
gbt_evaluator = RegressionEvaluator(labelCol="price",
predictionCol="prediction", metricName="rmse")
print("RMSE: %f" % gbt_evaluator.evaluate(prediction))
```

```
Cmd 13

lr_evaluator = RegressionEvaluator (predictionCol="prediction",labelCol="price",metricName="r2")
print("R Squared (R2) on test data = %g" %lr_evaluator.evaluate(prediction))
lr_evaluator = RegressionEvaluator(labelCol="price", predictionCol="prediction", metricName="rmse")
print("RMSE: %f" % lr_evaluator.evaluate(prediction))

▸ (2) Spark Jobs
R Squared (R2) on test data = 0.660423
RMSE: 13612.895275

Command took 1.74 seconds -- by hparekh2@calstatela.edu at 5/14/2022, 5:30:09 PM on QS
```

Gradient Boost Tree with Train Validation Split:

A) Run the steps 1 - 10 as shown above and rename the notebook as “Gradient Boost Tree with Validation Split”

B) To add Train Validation Split with Gradient Boost Tree, Run the steps A) - D) and replace this step from the E) of the Gradient Boost Tree. Copy the below code and run it.

```
from pyspark.ml.tuning import ParamGridBuilder,TrainValidationSplit,
CrossValidator

#gbt =
GBTRegressor(labelCol="price",featuresCol="features",maxIter=10,maxDepth
=10)
```

```
gbt =
GBTRegressor(labelCol="price",featuresCol="features",maxIter=3,maxDepth=
5)
#paramGrid =
ParamGridBuilder().addGrid(gbt.maxDepth,[15]).addGrid(gbt.maxIter,[5]).buil
d()
paramGrid =
ParamGridBuilder().addGrid(gbt.maxDepth,[5]).addGrid(gbt.maxIter,[5]).build
()
gbt_evaluator = RegressionEvaluator(predictionCol="prediction", \
labelCol="price",metricName="r2")
pipeline = Pipeline(stages=[assembler, gbt])
tv = TrainValidationSplit(estimator=pipeline,evaluator=gbt_evaluator,
estimatorParamMaps=paramGrid,trainRatio=0.8,parallelism=2)
model = tv.fit(train)
```

```
from pyspark.ml.tuning import ParamGridBuilder,TrainValidationSplit, CrossValidator

#gbt = GBTRegressor(labelCol="price",featuresCol="features",maxIter=10,maxDepth=10)
gbt = GBTRegressor(labelCol="price",featuresCol="features",maxIter=3,maxDepth=5)
#paramGrid = ParamGridBuilder().addGrid(gbt.maxDepth,[15]).addGrid(gbt.maxIter,[5]).build()
paramGrid = ParamGridBuilder().addGrid(gbt.maxDepth,[5]).addGrid(gbt.maxIter,[5]).build()
gbt_evaluator = RegressionEvaluator(predictionCol="prediction", \
labelCol="price",metricName="r2")
pipeline = Pipeline(stages=[assembler, gbt])
tv = TrainValidationSplit(estimator=pipeline,evaluator=gbt_evaluator, estimatorParamMaps=paramGrid,trainRatio=0.8,parallelism=2)
model = tv.fit(train)
```

▸ (51) Spark Jobs

Command took 2.19 minutes -- by kjikji956@gmail.com at 2022. 5. 17. 오전 3:02:25 on jjj

C) Copy the below code and run it to test the model.

```
prediction = model.transform(test)
predicted = prediction.select("features", "prediction", "price")
predicted = predicted.drop("features")
predicted.show()
```

```
prediction = model.transform(test)
predicted = prediction.select("features", "prediction", "price")
predicted = predicted.drop("features")
predicted.show()
```

```
▸ (1) Spark Jobs

+------------------+-----+
|        prediction|price|
+------------------+-----+
| 30383.72376014185| 4995|
|   37570.452429514|56997|

Command took 3.22 seconds -- by kjikji956@gmail.com at 2022. 5. 17. 오전 3:02:26 on jjj
```

D) Copy the below code and run it to calculate the Root Mean Square Error (RMSE) and Coefficient of Determination (R2) for the linear regression model.

```
gbt_evaluator =
RegressionEvaluator(predictionCol="prediction",labelCol="price",metricName
="r2")
print("R Squared (R2) on test data = %g" %gbt_evaluator.evaluate(prediction))
gbt_evaluator = RegressionEvaluator(labelCol="price",
predictionCol="prediction", metricName="rmse")
print("RMSE: %f" % gbt_evaluator.evaluate(prediction))
```

```
gbt_evaluator =
RegressionEvaluator(predictionCol="prediction",labelCol="price",metricName="r2")
print("R Squared (R2) on test data = %g" %gbt_evaluator.evaluate(prediction))
gbt_evaluator = RegressionEvaluator(labelCol="price", predictionCol="prediction",
metricName="rmse")
print("RMSE: %f" % gbt_evaluator.evaluate(prediction))
```

```
▸ (2) Spark Jobs

R Squared (R2) on test data = 0.650157
RMSE: 11545.600866

Command took 29.45 seconds -- by kjikji956@gmail.com at 2022. 5. 17. 오전 12:12:48 on jjj
```

Gradient Boost Tree with Cross Validation Split:

A) Run the steps 1 - 10 as shown above and rename the notebook as “Gradient Boost Tree with Cross Validation Split”

B) To add Cross Validation Split with Gradient Boost Tree, Run the steps A) - D) and replace this step from the E) of the Gradient Boost Tree. Copy the below code and run it.

```
from pyspark.ml.tuning import ParamGridBuilder,TrainValidationSplit, CrossValidator

#gbt = GBTRegressor(labelCol="price",featuresCol="features",maxIter=10,maxDepth=10)
gbt = GBTRegressor(labelCol="price",featuresCol="features",maxIter=3,maxDepth=3)
#paramGrid = ParamGridBuilder().addGrid(gbt.maxDepth,[15]).addGrid(gbt.maxIter,[5]).build()
paramGrid = ParamGridBuilder().addGrid(gbt.maxDepth,[5]).addGrid(gbt.maxIter,[5]).build()
gbt_evaluator = RegressionEvaluator(predictionCol="prediction", \
labelCol="price",metricName="r2")
pipeline = Pipeline(stages=[assembler, gbt])
cv = CrossValidator(estimator=pipeline, evaluator=gbt_evaluator,estimatorParamMaps=paramGrid,parallelism=4,numFolds=4)
model = cv.fit(train)
```

```
from pyspark.ml.tuning import ParamGridBuilder,TrainValidationSplit, CrossValidator

#gbt = GBTRegressor(labelCol="price",featuresCol="features",maxIter=10,maxDepth=10)
gbt = GBTRegressor(labelCol="price",featuresCol="features",maxIter=3,maxDepth=3)
#paramGrid = ParamGridBuilder().addGrid(gbt.maxDepth,[15]).addGrid(gbt.maxIter,[5]).build()
paramGrid = ParamGridBuilder().addGrid(gbt.maxDepth,[5]).addGrid(gbt.maxIter,[5]).build()
gbt_evaluator = RegressionEvaluator(predictionCol="prediction", \
labelCol="price",metricName="r2")
pipeline = Pipeline(stages=[assembler, gbt])
cv = CrossValidator(estimator=pipeline, evaluator=gbt_evaluator,estimatorParamMaps=paramGrid,parallelism=4,numFolds=4)
model = cv.fit(train)
```

▸ (51) Spark Jobs

Command took 5.47 minutes -- by kjikji956@gmail.com at 2022. 5. 17. 오전 3:08:28 on jjj

C) Copy the below code and run it to test the model in the new cell.

```
prediction = model.transform(test)
predicted = prediction.select("features", "prediction", "price")
predicted = predicted.drop("features")
predicted.show()
```

```
prediction = model.transform(test)
predicted = prediction.select("features", "prediction", "price")
predicted = predicted.drop("features")
predicted.show()
```

▸ (1) Spark Jobs

```
+------------------+-----+
|        prediction|price|
+------------------+-----+
| 36359.59004660151|27888|
|27671.946402546946|24975|
|27671.946402546946| 8975|
|27671.946402546946|32995|
|27671.946402546946|24995|
|12612.133363225634|18975|
```

Command took 3.12 seconds -- by kjikji956@gmail.com at 2022. 5. 17. 오전 3:08:28 on jjj

D) Copy the below code and run it to calculate the Root Mean Square Error (RMSE) and Coefficient of Determination (R2) for the linear regression model.

```
gbt_evaluator =
RegressionEvaluator(predictionCol="prediction",labelCol="price",metricName
="r2")
print("R Squared (R2) on test data = %g" %gbt_evaluator.evaluate(prediction))
gbt_evaluator = RegressionEvaluator(labelCol="price",
predictionCol="prediction", metricName="rmse")
print("RMSE: %f" % gbt_evaluator.evaluate(prediction))
```

```
gbttv_evaluator = RegressionEvaluator(predictionCol="prediction",labelCol="price",metricName="r2")
print("R Squared (R2) on test data = %g" %gbttv_evaluator.evaluate(prediction))
gbttv_evaluator = RegressionEvaluator(labelCol="price", predictionCol="prediction", metricName="rmse")
print("RMSE: %f" % gbttv_evaluator.evaluate(prediction))
```

▸ (2) Spark Jobs

```
R Squared (R2) on test data = 0.595013
RMSE: 12457.466001
```

Command took 24.61 seconds -- by kjikji956@gmail.com at 2022. 5. 17. 오전 3:08:28 on jjj

Factorization Machines learning:

A) Run the steps 1 - 10 as shown above and rename the notebook as "Factorization Machines learning"

B) Import this libraries and so, copy this code to the existing cell no. 6 and run it:

```
from kiwisolver import Solver
from pyspark.sql.functions import col
from pyspark.sql import functions as F
from pyspark.sql.types import *
from pyspark.sql.functions import *
from functools import reduce
from pyspark.sql.types import DoubleType,IntegerType
from pyspark.context import SparkContext
from pyspark.sql.session import SparkSession
from pyspark.ml import Pipeline
from pyspark.ml.evaluation import RegressionEvaluator
from pyspark.ml.feature import VectorAssembler
from sklearn import metrics
from pyspark.ml.regression import FMRegressor
from pyspark.ml.feature import MinMaxScaler
```

C) Copy the below code to split the data to 70 and 30%

```
splits = df_new.randomSplit([0.7, 0.3])
train = splits[0]
```

```
test = splits[1]
train_rows = train.count()
test_rows = test.count()
print("Training Rows:", train_rows, "Testing Rows:", test_rows)
```

```
Cmd 4

splits = df_new.randomSplit([0.7, 0.3])
train = splits[0]
test = splits[1]
train_rows = train.count()
test_rows = test.count()
print("Training Rows:", train_rows, "Testing Rows:", test_rows)

(4) Spark Jobs
Training Rows: 1306157 Testing Rows: 559210
Command took 30.11 seconds -- by kjikji956@gmail.com at 2022. 5. 17. 오전 12:00:08 on jjj
```

D) Copy the below code to define the pipeline and run it.

```
assembler = VectorAssembler(inputCols =["engine_displacement","is_new", "mileage", "frame_damaged", "has_accidents", "seller_rating","isCab","horsepower"], outputCol="features")
featureScaler = MinMaxScaler(inputCol="features", outputCol="scaledFeatures")
```

```
Cmd 5

assembler = VectorAssembler(inputCols =["engine_displacement","is_new", "mileage", "frame_damaged", "has_accidents", "seller_rating","isCab","horsepower"], outputCol="features")
featureScaler = MinMaxScaler(inputCol="features", outputCol="scaledFeatures")

Command took 0.15 seconds -- by kjikji956@gmail.com at 2022. 5. 17. 오전 3:21:10 on jjj
```

E) Copy the below code and run it to train the Factorization Machines learning.

```
#FM = FMRegressor(labelCol="price",featuresCol="scaledFeatures",
stepSize=2,factorSize=32,maxIter=300)
FM = FMRegressor(labelCol="price",featuresCol="scaledFeatures",
stepSize=2,factorSize=8,maxIter=50)
pipeline = Pipeline(stages=[assembler, featureScaler,FM])
model = pipeline.fit(train)
```

```
#FM = FMRegressor(labelCol="price",featuresCol="scaledFeatures", stepSize=2,factorSize=32,maxIter=300)
FM = FMRegressor(labelCol="price",featuresCol="scaledFeatures", stepSize=2,factorSize=8,maxIter=50)
pipeline = Pipeline(stages=[assembler, featureScaler,FM])
model = pipeline.fit(train)
```

▸ (50) Spark Jobs

Command took 7.15 minutes -- by kjikji956@gmail.com at 2022. 5. 17. 오전 3:21:42 on jjj

F) Copy the below code and run it to test the model.

```
prediction = model.transform(test)
predicted = prediction.select("features", "prediction", "price")
predicted = predicted.drop("features")
predicted.show()
```

```
prediction = model.transform(test)
predicted = prediction.select("features", "prediction", "price")
predicted = predicted.drop("features")
predicted.show()
```

▸ (1) Spark Jobs

```
+------------------+-----+
|        prediction|price|
+------------------+-----+
|  47.6527534014271| 4599|
|  49.4027328347336|13975|
| 46.48841365973088|28905|
| 32.75123312453502| 8975|
|19.182607173200978|24995|
```

Command took 2.62 seconds -- by kjikji956@gmail.com at 2022. 5. 17. 오전 3:43:17 on jjj

G) Copy the below code and run it to calculate the Root Mean Square Error (RMSE) and Coefficient of Determination (R2) for the linear regression model.

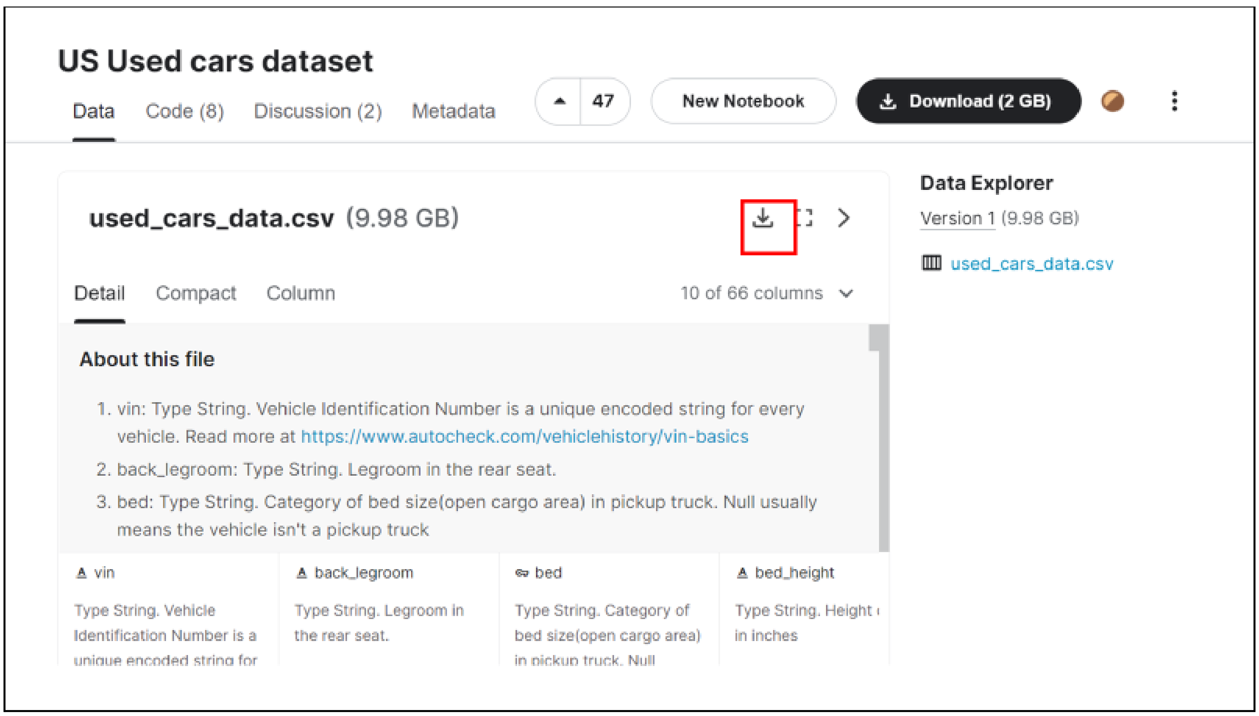
```
fm_evaluator =
RegressionEvaluator(predictionCol="prediction",labelCol="price",metricName
="r2")
print("R Squared (R2) on test data = %g" %fm_evaluator.evaluate(prediction))
fm_evaluator = RegressionEvaluator(labelCol="price",
predictionCol="prediction", metricName="rmse")
print("RMSE: %f" % fm_evaluator.evaluate(prediction))
```

```
fm_evaluator = RegressionEvaluator(predictionCol="prediction",labelCol="price",metricName="r2")
print("R Squared (R2) on test data = %g" %fm_evaluator.evaluate(prediction))
fm_evaluator = RegressionEvaluator(labelCol="price", predictionCol="prediction", metricName="rmse")
print("RMSE: %f" % fm_evaluator.evaluate(prediction))
```

```
(2) Spark Jobs
R Squared (R2) on test data = 0.177479
RMSE: 17710.377359
Command took 32.76 seconds -- by kjikji956@gmail.com at 2022. 5. 17. 오전 3:49:56 on jjj
```

Part 2: Run the Models on the Spark CLI to calculate the RMSE and R2 for the entire dataset.

1. Download the data file from https://www.kaggle.com/datasets/ananaymital/us-used-cars-dataset to your computer.



2. Use SCP to download the file to the Linux server by using the below code.

   Change the username:

Scp used_cars_data.csv pramdas2@129.146.154.176:~

```
(base) Apples-MacBook-Pro:~ priya$ scp /Users/priya/CalStateLA/1st\ Sem/5200-Big\ data/Project/used_cars_
data.csv pramdas2@129.146.154.176:~
pramdas2@129.146.154.176's password:
used_cars_data.csv                              100% 9518MB   4.7MB/s   33:58
```

3. Download the python scripts for the above code from the GitHub link: https://github.com/Heta-Parekh/MachineLearningModels

Heta-Parekh Add files via upload		d883bf2 4 days ago 7 commits
LinearRegressionModel.py	Add files via upload	9 days ago
Team2_Presentation (1).pptx	Add files via upload	9 days ago
Used_Car.csv	Add files via upload	4 days ago
gbt_crossvalidationsplit.py	Add files via upload	9 days ago
gbt_trainvalidationsplit.py	Add files via upload	9 days ago
gradientboosttree.py	Add files via upload	9 days ago
randomforest.py	Add files via upload	9 days ago
recommendation.py	Add files via upload	9 days ago
ridgeregression.py	Add files via upload	9 days ago

4. Upload all the python scripts on SCP and run them using the codes below:
   a) Upload the downloaded files from GitHub and upload them using the SCP command.

```
scp FilePath/*.py pramdas2@129.146.154.176:~
```

```
(base) Apples-MBP:~ priya$ scp /Users/priya/CalStateLA/4th\ Sem/5560\ -\ INTRO\
TO\ BIG\ DATA\ SCIENCE/Project/Team_2\ Code/*.py pramdas2@129.146.154.176:~
pramdas2@129.146.154.176's password:
FM.py                                         100% 5177   166.6KB/s   00:00
LinearRegressionModel.py                      100% 5077   304.1KB/s   00:00
gbt_crossvalidationsplit.py                   100% 5552   334.4KB/s   00:00
gbt_trainvalidationsplit.py                   100% 5574   174.5KB/s   00:00
gradientboosttree.py                          100% 5118   307.8KB/s   00:00
randomforest.py                               100% 5185   311.9KB/s   00:00
recommendation.py                             100% 8186   243.1KB/s   00:00
(base) Apples-MBP:~ priya$
```

   b) Connect to Oracle Cloud: Big Data Compute

   You need to remotely access your Oracle Big Data that you executed in your Oracle Cloud account using ssh. Your CalStateLA username(syadav5) should be a username/password to connect to the Hadoop cluster at BDCE as follows:

   Note: Change the username (pramdas2) before executing.

```
ssh pramdas2@129.146.154.176
```

c) Copy the used_cars_data.csv file to the /tmp folder

```
hdfs dfs -put used_cars_data.csv /tmp
```

d) Check of the file is present in the /tmp folder

```
hdfs dfs -ls /tmp
```

```
-bash-4.2$ hdfs dfs -ls /tmp
Found 4 items
drwxr-xr-x   - hdfs     hdfs          0 2022-05-09 03:29 /tmp/entity-file-histor
y
drwx-wx-wx   - hive     hdfs          0 2022-05-15 21:54 /tmp/hive
drwx-wx-wx   - spark    hdfs          0 2022-05-09 03:30 /tmp/spark
-rw-r--r--   3 pramdas2 hdfs 9980208148 2022-05-18 21:25 /tmp/used_cars_data.csv
-bash-4.2$
```

e) Check if all the .py files are been uploaded to the server by using the below code

```
ls
```

```
-bash-4.2$ ls
FM.py                          gbt_trainvalidationsplit.py   recommendation.py
LinearRegressionModel.py       gradientboosttree.py          used_cars_data.csv
gbt_crossvalidationsplit.py    randomforest.py
-bash-4.2$
```

f) Using the below command run regression models.

Note: Replace the highlighted text with the specific .py file name.

```
spark-submit fileName.py
```

g) To execute the **Linear Regression** model and to get the model result.

```
spark-submit LinearRegressionModel.py
```

Result:

```
22/05/19 19:03:07 INFO DAGScheduler: Job 12 finished: treeAggregate at Statistics.scala:58, took 35.776772 s
R Squared (R2) on test data = 0.738957
22/05/19 19:03:07 INFO FileSourceStrategy: Pushed Filters:
```

```
22/05/19 19:03:42 INFO YarnScheduler: Killing all running tasks in stage 20: Stage finished
22/05/19 19:03:42 INFO DAGScheduler: Job 13 finished: treeAggregate at Statistics.scala:58, took 35.426818 s
RMSE: 7714.316869
```

h) To execute the **Random Forest** model and to get the model result.

```
spark-submit randomforest.py
```

Result:

```
22/05/19 19:18:04 INFO YarnScheduler: Killing all running tasks in stage 26: Stage finished
22/05/19 19:18:04 INFO DAGScheduler: Job 17 finished: treeAggregate at Statistics.scala:58, took 73.243917 s
R Squared (R2) on test data = 0.768045
```

```
22/05/19 19:19:16 INFO YarnScheduler: Killing all running tasks in stage 28: Stage finished
22/05/19 19:19:16 INFO DAGScheduler: Job 18 finished: treeAggregate at Statistics.scala:58, took 72.651221 s
RMSE: 7280.555943
```

i) To execute the **Recommendation model** and to get the model result.

```
spark-submit recommendation.py
```

Result:

```
22/05/19 19:38:49 INFO YarnScheduler: Killing all running tasks in stage 154: Stage finished
22/05/19 19:38:49 INFO DAGScheduler: Job 22 finished: treeAggregate at Statistics.scala:58, took 132.921342 s
Root Mean Square Error (RMSE): 0.39675011824784534
```

```
22/05/19 19:03:07 INFO YarnScheduler: Killing all running tasks in stage 18: Stage finished
22/05/19 19:03:07 INFO DAGScheduler: Job 12 finished: treeAggregate at Statistics.scala:58, took 35.776772 s
R Squared (R2) on test data = 0.738957
```

j) To execute the **Gradient Booster** model and to get the model result.

```
spark-submit gradientboosttree.py
```

Result:

```
22/05/19 20:10:09 INFO YarnScheduler: Killing all running tasks in stage 118: St
age finished
22/05/19 20:10:09 INFO DAGScheduler: Job 63 finished: treeAggregate at Statistic
s.scala:58, took 73.582141 s
RMSE: 6676.712299
```

```
22/05/19 20:08:55 INFO DAGScheduler: Job 62 finished: treeAggregate at Statistics.scala:58,
 took 74.111902 s
R Squared (R2) on test data = 0.805031
```

k) To execute the **Gradient Booster with Train Validation Split** model and to get the model result.

```
spark-submit gbt_trainvalidationsplit.py
```

Result:

```
22/05/19 20:52:46 INFO YarnScheduler: Killing all running tasks in stage 662: Stage finishe
d
22/05/19 20:52:46 INFO DAGScheduler: Job 339 finished: treeAggregate at Statistics.scala:58
, took 36.028047 s
RMSE: 6470.908244
```

```
22/05/19 20:52:09 INFO YarnScheduler: Killing all running tasks in stage 660: Stage finishe
d
22/05/19 20:52:09 INFO DAGScheduler: Job 338 finished: treeAggregate at Statistics.scala:58
, took 36.385745 s
R Squared (R2) on test data = 0.817787
```

l) To execute the **Gradient Booster with Cross-Validation Split** model and to get the model result.

```
spark-submit gbt_crossvalidationsplit.py
```

Result:

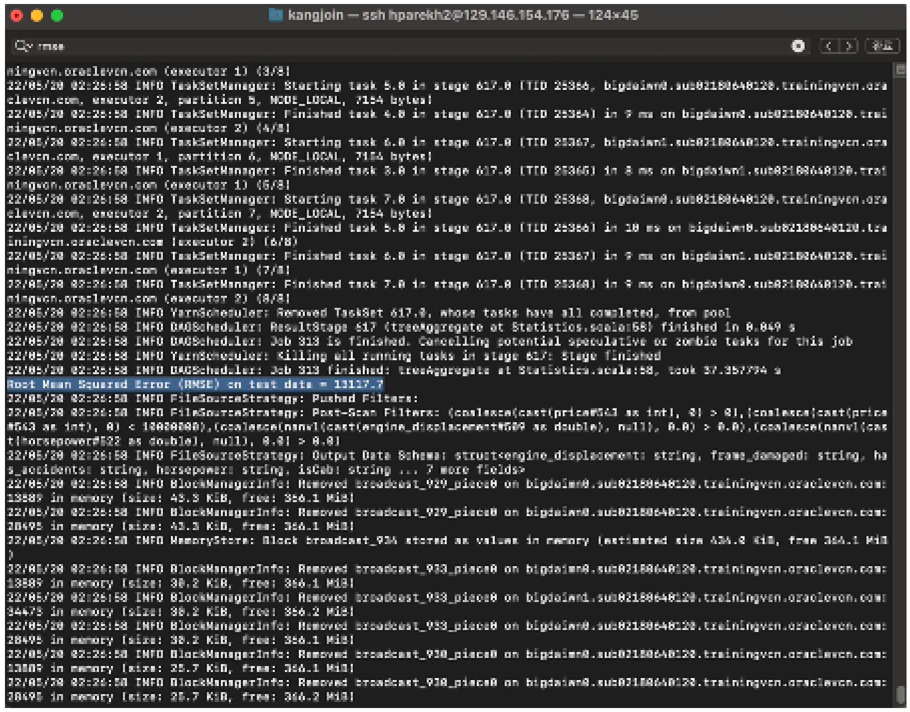
```
22/05/19 20:52:46 INFO YarnScheduler: Killing all running tasks in stage 662: Stage finishe
d
22/05/19 20:52:46 INFO DAGScheduler: Job 339 finished: treeAggregate at Statistics.scala:58
, took 36.028047 s
RMSE: 6470.908244
```

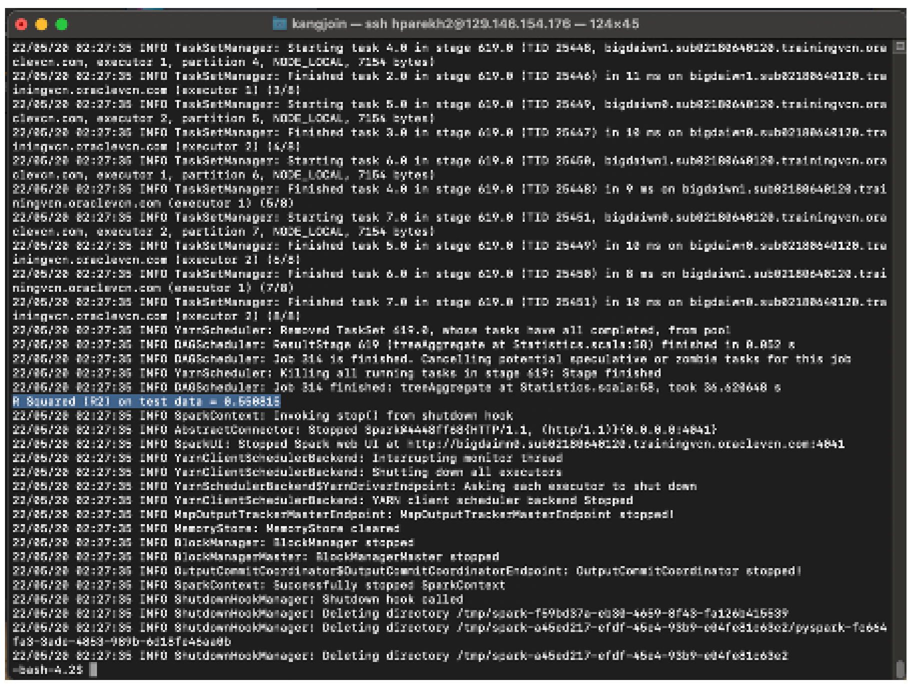
```
22/05/19 20:52:09 INFO YarnScheduler: Killing all running tasks in stage 660: Stage finishe
d
22/05/19 20:52:09 INFO DAGScheduler: Job 338 finished: treeAggregate at Statistics.scala:58
, took 36.385745 s
R Squared (R2) on test data = 0.817787
```

m) To execute the **Factorization Machines** model and to get the model result.

```
spark-submit FM.py
```

Result:


kangjoin — ssh hparekh2@129.146.154.176 — 124×45
Root Mean Squared Error (RMSE) on test data = 13117.7


kangjoin — ssh hparekh2@129.146.154.176 — 124×45

References:

[1] https://www.slideshare.net/YashIyengar/big-data-analysis-of-second-hand-car-sales

[2] https://github.com/clrife/CarPriceAnalysis

[3] https://github.com/Heta-Parekh/MachineLearningModels

[4] https://www.kaggle.com/ananaymital/us-used-cars-dataset

[5] Mason, L., Baxter, J., Bartlett, P. and Frean, M. (2000). Boosting algorithms as gradient descent. In *Advances in Neural Information Processing Systems* **12** 512--518. MIT Press, Cambridge, MA.