CIS5560 Term Project Tutorial

Authors: Anusha Valasapalli
Naga Sai Lohitha Karmuru
Rishabh Shah
Wethanie Law

Instructor: Jongwook Woo
Date: 05/22/2023

Lab Tutorial

Anusha Valasapalli (avalasa@calstatela.edu)
Naga Sai Lohitha Karmuru (nkarmur@calstatela.edu)
Rishabh shah (rshah39@calstatela.edu)
Wethanie Law (wlaw4@calstatela.edu)

USA Used Car Dataset

Predictive Analysis using machine learning models in Spark ML

Objectives

The objective of the lab is to build a model that predicts the price of the used cars considering the features of the details of the vehicle using the following machine learning algorithms:

Used Car Price Prediction

Linear Regression
Random Forest Regression
Gradient Boost Tree Regression

Factorization Machines Regression
Logistic Regression Classification

Predicting whether the customer can buy this car or not?

Logistic Regression

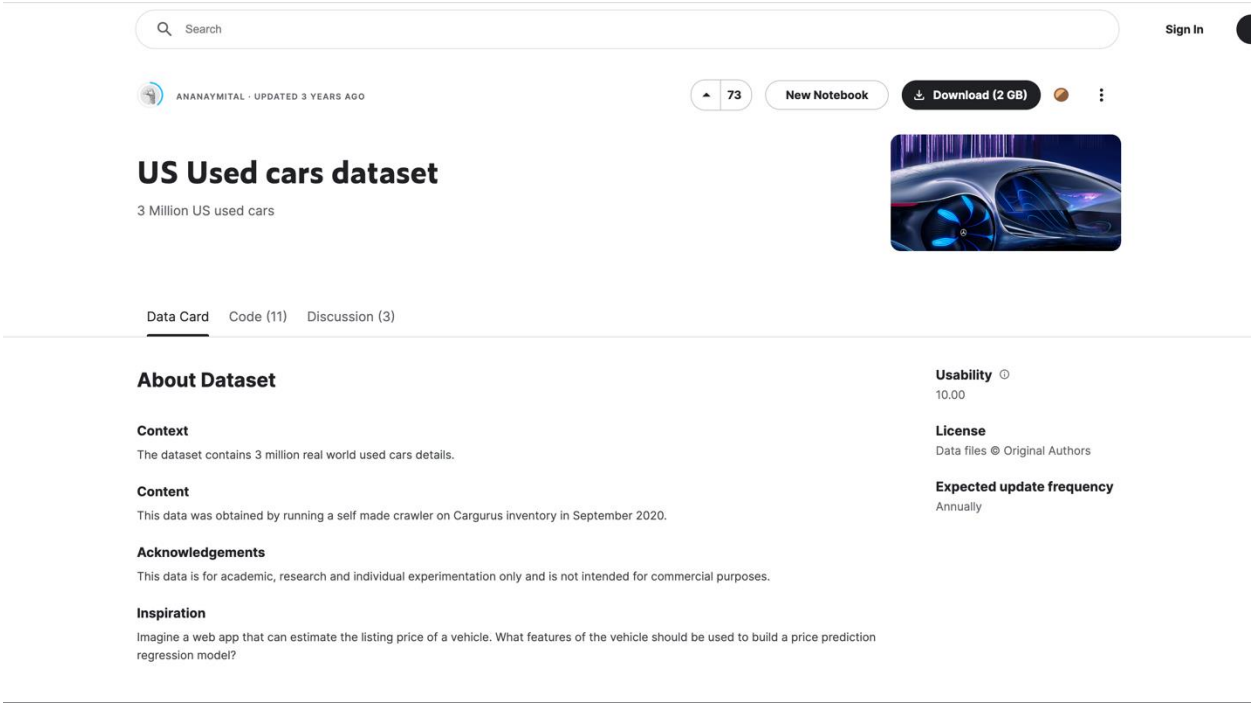
You will learn how to build the above-listed regressions to predict the price of the car and whether the price is a fair deal.

Platform Specifications: -

Version: 3.2.1
CPU Speed: 2.40 GHz
# of CPU cores: 4
 # Number of nodes: 3
Total Memory Size: 481GB

Dataset Specifications: -

Dataset Name: US Used Car Dataset
Dataset Size: 2 GB
Dataset URL: https://www.kaggle.com/datasets/ananaymital/us-used-cars-dataset
Dataset Format: CSV



Step 1: Get data manually from the Data source.

We first need to Download the dataset from Kaggle.

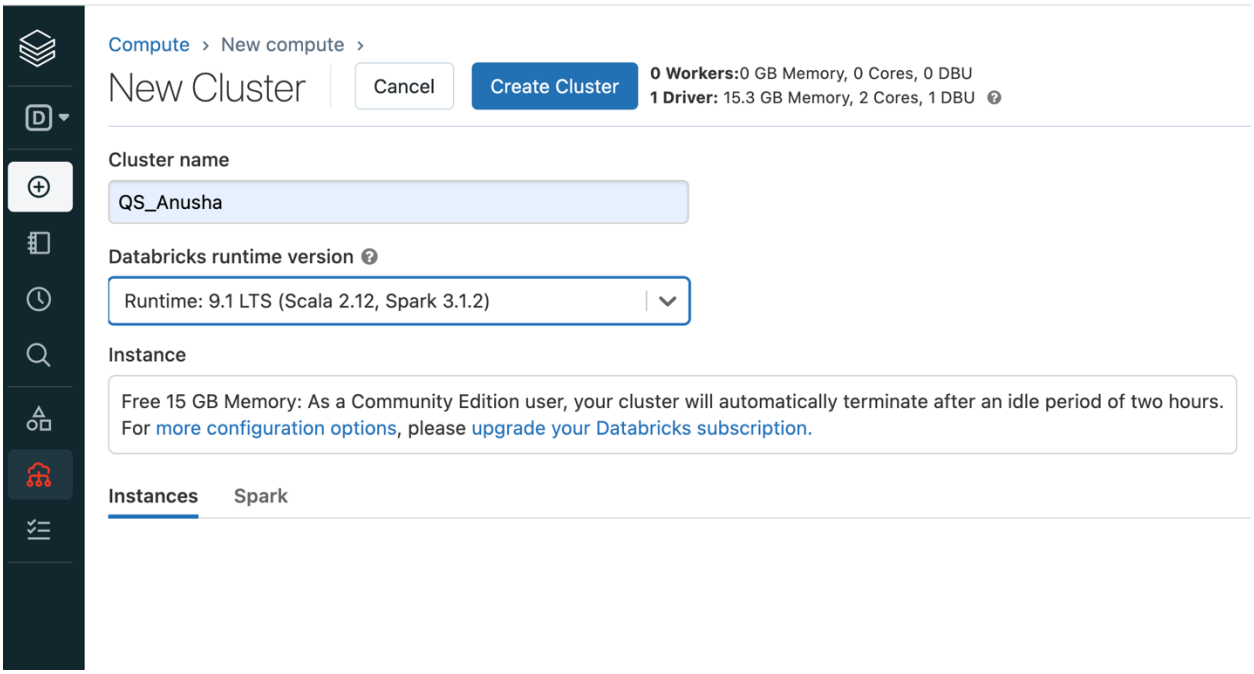
1. Log in to Kaggle.
2. It is showing as a 2GB dataset with 9.9 GB. I downloaded the 9.9 GB csv dataset file from Kaggle.
3. Using the below code, slice the 33 MB of data from the above 9.9 GB csv and use that 33 MB file in the Databricks.
4. Using the below code, remove 2 GB of data from the 9.9 GB csv and use that 2 GB file in the spark-submit.

```
split -l 1000000 -d pricedata.csv ppd-
```

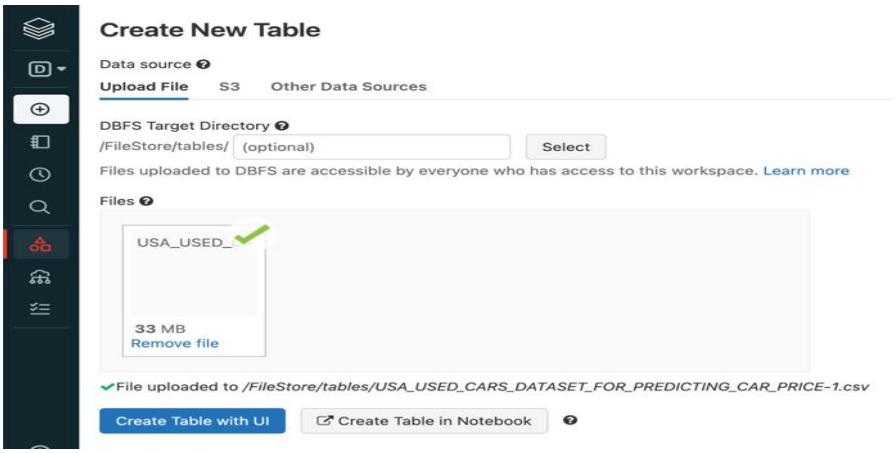
Step 2: Upload data to Databricks.

1)Log into data bricks using the below URL.
https://community.cloud.databricks.com/?o=999471079023550#

Step 3: Create a cluster in the data bricks.



Step 4: Upload the US Used Car dataset CSV file into the Databricks data and click Create notebook.



Once the file is uploaded, you can see Create Table in Notebook. Select “Create Table in Notebook.” It automatically generates a Spark and markdown codes in the notebook, which reads the data.
File and display it.

```
# File location and type
file_location = "/FileStore/tables/USA_USED_CARS_DATASET_FOR_PREDICTING_CAR_PRICE.csv"
file_type = "csv"

# CSV options
infer_schema = "true"
first_row_is_header = "true"
delimiter = ","

# The applied options are for CSV files. For other file types, these will be ignored.
df = spark.read.format(file_type) \
  .option("inferSchema", infer_schema) \
  .option("header", first_row_is_header) \
  .option("sep", delimiter) \
  .load(file_location)

#display(df)
df.show()
```

Step 5: Create a temporary view of the table to use as a table name of Spark SQL.

temp_table_name = "USA_USED_CARS_DATASET_FOR_PREDICTING_CAR_PRICE_csv"
df. createOrReplaceTempView(temp_table_name)

Cmd 4

```
1
2  temp_table_name = "USA_USED_CARS_DATASET_FOR_PREDICTING_CAR_PRICE_csv"
3
4  df.createOrReplaceTempView(temp_table_name)
```

Command took 0.12 seconds -- by avalasa@calstatela.edu at 5/14/2023, 9:05:25 AM on QS_ANUSHA

Step 6: To see the data in the CSV file, use this code df. take(10)

df.take(10)

```
1  df.take(10)
```

Python

▸ (1) Spark Jobs

Out[3]: [Row(vin='1N4BL4CV9LC274871', back_legroom='35.2 in', bed=None, bed_height=None, bed_length=None, body_type='Sedan', cabin=None, city='San Antonio', city_fuel_economy='27', combine_fuel_economy=None, daysonmarket='0', dealer_zip='78233', description='Pearl White Tricoat 2020 Nissan Altima 2.5 SR FWD CVT with Xtronic 2.5L 4-Cylinder DOHC 16V Recent Arrival! 27/37 City/Highway MPG Premium Package (Heated Front Seats, Heated Outside Mirrors, and Single Panel Moonroof), 4-Wheel Disc Brakes, 6 Speakers, ABS brakes, Air Conditioning, Alloy wheels, AM/FM radio: SiriusXM, Auto High-beam Headlights, Blind Spot Warning, Body-Colored Splash Guards, Brake assist, Bumpers: body-color, Chrome Bumper Protector, Delay-off headlights, Driver door bin, Driver vanity mirror, Dual front impact airbags, Dual front side impact airbags, Electronic Stability Control, Four wheel independent suspension, Front anti-roll bar, Front Bucket Seats, Front Center Armrest, Front reading lights, Front Sport Bucket Seats, Fully automatic headlights, Illuminated entry, Knee airbag, Leather Shift Knob, Low tire pressure warning, NissanConnect featuring Apple CarPlay and Android Auto, Occupant sensing airbag, Outside temperature display, Overhead airbag, Overhead console, Panic alarm, Passenger door bin, Passenger vanity mirror, Power door mirrors, Power driver seat, Power steering, Power windows, Premium Paint - Pearl White, Radio data system, Radio: AM/FM Audio System, Rear anti-roll bar, Rear Parking Sensors, Rear reading lights, Rear seat center armrest, Rear side impact airbag, Rear window defroster, Remote keyless entry, Security system, Speed control, Speed-sensing steering, Speed-Sensitive Wipers, Split folding rear seat, Sport Seat Trim, Sport steering wheel, SR Floor Mats/Trunk Mat/Hideaway Nets, Steering wheel mounted audio controls, Tachometer, Telescoping steering wheel, Tilt steering wheel, Traction control, Trip computer, Trunk Organizer Tray, and Variably intermittent wipers. Incentives have Residency Restrictions. If you do not qualify Price includes: $2250 - Nissan Customer Cash - National. Exp. 09/30/2020[!@@Additional Info@@!]Front Sport Bucket Seats,Sport Seat Trim,Radio: AM/FM Audio System,Body-Colored Splash Guards,Premium Paint - Pearl White,Premium Package,Chrome Bumper Protector,SR Floor Mats/Trunk Mat/Hideaway Nets,Trunk Organizer Tray,Heated Front Seats,Heated Outside Mirrors,Single Panel Moonroof,4-Wheel Disc Brakes,6 Speakers,Air Conditioning,Electronic Stability Control,Front Bucket Seats,Front Center Armrest,Leather Shift Knob,Rear Parking Sensors,Tachometer,ABS brakes,Alloy wheels,Brake assist,Bumpers: body-color,Delay-off headlights,Driver door bin,Driver vanity mirror,Dual front impact airbags,Dual front side impact airbags,Four wheel independent suspension,Front anti-roll bar,Front reading lights,Fully automatic headlights,Illuminated entry,Knee airbag,Low tire pressure warning,Occupant sensing airbag,Outside temperature display,Overhead airbag,Overhead console,Panic alarm,Passenger door bin,Passenger vanity mirror,Power door mirrors,Power driver seat,Power steering,Power windows,Radio data system,
Rear anti-roll bar,Rear reading lights,Rear seat center armrest,Rear side impact airbag,Rear window defroster,Remote keyless entry,Security system,Speed contr

Step 7: To check the data type of the parameter, use the below code

df.dtypes

Step 8: Import the statements

```
import pyspark
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, lit
from pyspark.sql.types import StructType, StructField, StringType,IntegerType
from pyspark.sql.types import *
from pyspark.sql.functions import *
from pyspark.ml.regression import RandomForestRegressor, LinearRegression, GBTRegressor
from pyspark.ml.linalg import Vectors
from pyspark.ml import Pipeline
from pyspark.ml.tuning import ParamGridBuilder,TrainValidationSplit, CrossValidator
from pyspark.ml.tuning import CrossValidator
from pyspark.ml.classification import LogisticRegression
from pyspark.ml.feature import VectorAssembler, StringIndexer, VectorIndexer, MinMaxScaler
from pyspark.ml.evaluation import BinaryClassificationEvaluator
```

```
import pyspark
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, lit
from pyspark.sql.types import StructType, StructField, StringType,IntegerType
from pyspark.sql.types import *
from pyspark.sql.functions import *
from pyspark.ml.regression import RandomForestRegressor, LinearRegression, GBTRegressor, FMRegressor
from pyspark.ml.linalg import Vectors
from pyspark.ml import Pipeline
from pyspark.ml.tuning import ParamGridBuilder,TrainValidationSplit, CrossValidator
from pyspark.ml.tuning import CrossValidator
from pyspark.ml.classification import DecisionTreeClassifier, LogisticRegression
from pyspark.ml.feature import VectorAssembler, StringIndexer, VectorIndexer, MinMaxScaler
from pyspark.ml.evaluation import BinaryClassificationEvaluator
from pyspark.sql.functions import when, col

from pyspark.context import SparkContext
from pyspark.sql.session import SparkSession
from pyspark.ml.evaluation import RegressionEvaluator
from pyspark.ml.tuning import ParamGridBuilder
import pandas as pd


```

Command took 0.92 seconds -- by avalasa@calstatela.edu at 5/14/2023, 9:05:25 AM on QS_ANUSHA

Step 9: Typecast string to int or float

df2 = df.withColumn("year",col("year").cast("int")).withColumn("city_fuel_economy",col("city_fuel_economy").cast("float")).withColumn("price",col("price").cast("float")).withColumn("mileage",col("mileage").cast("float")).withColumn("owner_count",col("owner_count").cast("int")).withColumn("latitude",col("latitude").cast("float")).withColumn("highway_fuel_economy",col("highway_fuel_economy").cast("float")).withColumn("daysonmarket",col("daysonmarket").cast("int")).withColumn("dealer_zip",col("dealer_zip").cast("int")).withColumn("engine_displacement",col("engine_displacement").cast("int")).withColumn("listing_id",col("listing_id").cast("int")).withColumn("savings_amount",col("savings_amount").cast("int")).withColumn("sp_id",col("sp_id").cast("int")).withColumn("seller_rating",col("seller_rating").cast("int"))

Typecast string to int or float

Python

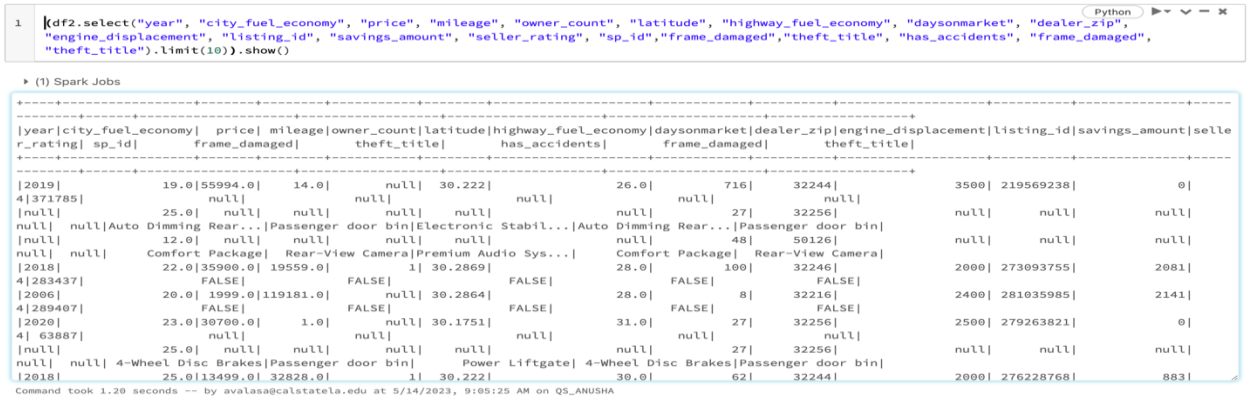
```
df2 =
df.withColumn("year",col("year").cast("int")).withColumn("city_fuel_economy",col("city_fuel_economy").cast("float")).withColumn("price",col("price").cast("float")).withColumn("mileage",col("mileage").cast("float")).withColumn("owner_count",col("owner_count").cast("int")).withColumn("latitude",col("latitude").cast("float")).withColumn("highway_fuel_economy",col("highway_fuel_economy").cast("float")).withColumn("daysonmarket",col("daysonmarket").cast("int")).withColumn("dealer_zip",col("dealer_zip").cast("int")).withColumn("engine_displacement",col("engine_displacement").cast("int")).withColumn("listing_id",col("listing_id").cast("int")).withColumn("savings_amount",col("savings_amount").cast("int")).withColumn("sp_id",col("sp_id").cast("int")).withColumn("seller_rating",col("seller_rating").cast("int"))

```

Command took 0.61 seconds -- by avalasa@calstatela.edu at 5/14/2023, 9:05:25 AM on QS_ANUSHA

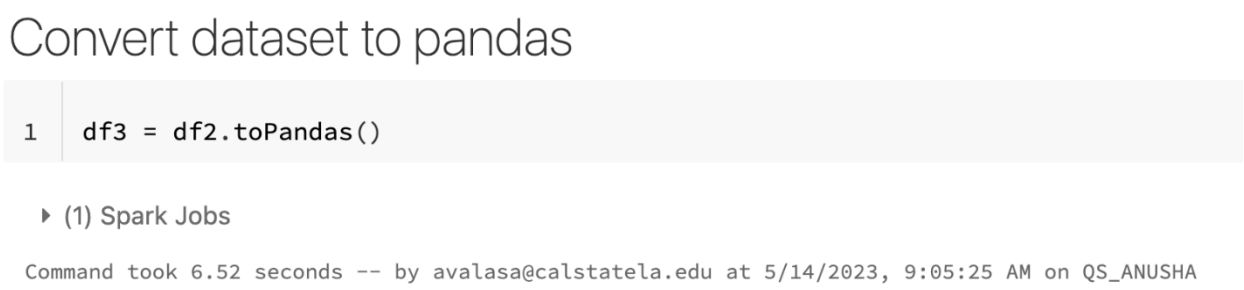
Step 10: After converting the string to Int and float, see the results with the following command.

(df2.select("year", "city_fuel_economy", "price", "mileage", "owner_count", "latitude", "highway_fuel_economy", "daysonmarket", "dealer_zip", "engine_displacement", "listing_id", "savings_amount", "seller_rating", "sp_id","frame_damaged","theft_title", "has_accidents", "frame_damaged", "theft_title").limit(10)).show()



Step 11: Convert dataset to pandas to do the modifications

df3 = df2.toPandas()



Step 12:Remove the null values and filter the price to more than 100,000

df2.na.drop(subset=["price", "year", "city_fuel_economy", "mileage", "owner_count", "highway_fuel_economy", "daysonmarket", "dealer_zip", "engine_displacement", "listing_id", "savings_amount", "seller_rating", "sp_id", "has_accidents", "frame_damaged", "theft_title"])
df222 = df2.filter(col("price") < 100000)
df22 = df222.select("year", "city_fuel_economy", col("price").alias("label"), "mileage", "owner_count", "latitude", "highway_fuel_economy", "daysonmarket", "dealer_zip", "engine_displacement", "listing_id", "savings_amount", "seller_rating", "sp_id", "has_accidents", "frame_damaged", "theft_title")
df22.show()

```
df2.na.drop(subset=["price", "year", "city_fuel_economy", "mileage", "owner_count", "highway_fuel_economy", "daysonmarket", "dealer_zip",
"engine_displacement", "listing_id", "savings_amount", "seller_rating", "sp_id", "has_accidents", "frame_damaged", "theft_title"])
df222 = df2.filter(col("price") < 100000)
df22 = df222.select("year", "city_fuel_economy", col("price").alias("label"), "mileage", "owner_count", "latitude", "highway_fuel_economy", "daysonmarket",
"dealer_zip", "engine_displacement", "listing_id", "savings_amount", "seller_rating", "sp_id", "has_accidents", "frame_damaged", "theft_title")
df22.show()
```

▸ (1) Spark Jobs

```
+----+----------------+-------+--------+----------+--------+--------------------+------------+---------+------------------+----------+--------------+-----
--------+------+-------------+-------------+-----------+
|year|city_fuel_economy|  label| mileage|owner_count|latitude|highway_fuel_economy|daysonmarket|dealer_zip|engine_displacement|listing_id|savings_amount|selle
r_rating| sp_id|has_accidents|frame_damaged|theft_title|
+----+----------------+-------+--------+----------+--------+--------------------+------------+---------+------------------+----------+--------------+-----
--------+------+-------------+-------------+-----------+
|2019|            19.0|55994.0|    14.0|      null|  30.222|                26.0|         716|     32244|              3500| 219569238|             0|
4|371785|         null|         null|       null|
|2018|            22.0|35900.0| 19559.0|         1| 30.2869|                28.0|         100|     32246|              2000| 273093755|          2081|
4|283437|        FALSE|        FALSE|      FALSE|
|2006|            20.0| 1999.0|119181.0|      null| 30.2864|                28.0|           8|     32216|              2400| 281035985|          2141|
4|289407|        FALSE|        FALSE|      FALSE|
|2020|            23.0|30700.0|     1.0|      null| 30.1751|                31.0|          27|     32256|              2500| 279263821|             0|
4| 63887|         null|         null|       null|
|2018|            25.0|13499.0| 32828.0|         1|  30.222|                30.0|          62|     32244|              2000| 276228768|           883|
4|371785|        FALSE|        FALSE|      FALSE|
|2020|            18.0|60015.0|    13.0|      null| 30.2025|                25.0|         209|     32244|              3600| 265854322|             0|
4|290529|         null|         null|       null|
|2020|            20.0|56532.0|     7.0|      null|  30.222|                27.0|         230|     32244|              3500| 264019553|             0|
4|371785|         null|         null|       null|
```

STEP 13:Split the data into Train and Test datasets.

splits = df22.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)

Split the data into Train and Test datasets.

```
splits = df22.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: 2998 TESTING ROWS: 1291
```

Command took 8.59 seconds -- by avalasa@calstatela.edu at 5/14/2023, 9:07:16 AM on QS_ANUSHA

Working with Machine Learning Algorithms: -

We did tune all algorithms with cross-validation and Train Validation Split

Algorithm 1: Linear Regression

LINEAR REGRESSION CROSS-VALIDATION

1) Create a vector assembler with input columns

lr_assembler = VectorAssembler(inputCols = ["year", "city_fuel_economy", "mileage", "owner_count", "highway_fuel_economy", "engine_displacement", "savings_amount"], outputCol="features").setHandleInvalid("skip")

```
1  xlr_assembler = VectorAssembler(inputCols = ["year", "city_fuel_economy", "mileage", "owner_count", "highway_fuel_economy", "engine_displacement",
   "savings_amount"], outputCol="features").setHandleInvalid("skip")
```

Command took 0.05 seconds -- by avalasa@calstatela.edu at 5/14/2023, 9:07:23 AM on QS_ANUSHA

2) Train a Regression Model

lr= LinearRegression(labelCol="label") #,featuresCol="features",maxIter=10

```
1  lr= LinearRegression(labelCol="label") #,featuresCol="features",maxIter=10
```

Command took 0.16 seconds -- by avalasa@calstatela.edu at 5/14/2023, 9:07:26 AM on QS_ANUSHA

3) Create a parameter combination, which is to tune the model.

```
1  paramGrid1_CV_LR = ParamGridBuilder()\
2  .addGrid(lr.maxIter, [10, 30])\
3  .addGrid(lr.regParam, [0, 0.1])\
4  .build()
```

Command took 0.03 seconds -- by avalasa@calstatela.edu at 5/14/2023, 9:07:30 AM on QS_ANUSHA

4) Create an evaluator to evaluate a model with R2

from pyspark.ml.evaluation import RegressionEvaluator
lr_CV_evaluator = RegressionEvaluator(predictionCol="prediction", \
labelCol="label",metricName="r2")

```
from pyspark.ml.evaluation import RegressionEvaluator
lr_CV_evaluator = RegressionEvaluator(predictionCol="prediction", \
labelCol="label",metricName="r2")

```

Command took 0.05 seconds -- by avalasa@calstatela.edu at 5/14/2023, 9:07:33 AM on QS_ANUSHA

5) Create a pipeline, which is to sequence the tasks.

#we can use same pipeline_lr for both cv and tvs
pipeline_lr = Pipeline(stages=[lr_assembler, lr])

```
#we can use same pipeline_lr for both cv and tvs
pipeline_lr = Pipeline(stages=[lr_assembler, lr])
```

Command took 0.02 seconds -- by avalasa@calstatela.edu at 5/14/2023, 9:07:36 AM on QS_ANUSHA

6) **Create a cross-validator to tune the model for the generalization. The default folds are 3.**

cv_lr = CrossValidator(estimator=pipeline_lr, evaluator=lr_CV_evaluator, estimatorParamMaps=paramGrid1_CV_LR)

```
cv_lr = CrossValidator(estimator=pipeline_lr, evaluator=lr_CV_evaluator, estimatorParamMaps=paramGrid1_CV_LR)
```

Command took 0.02 seconds -- by avalasa@calstatela.edu at 5/14/2023, 9:07:39 AM on QS_ANUSHA

7)Build a model with the train Dataframe of cross validator

model_CV_LR = cv_lr.fit(train)

```
model_CV_LR = cv_lr.fit(train)
```

▸ (38) Spark Jobs

/databricks/spark/python/pyspark/ml/util.py:886: UserWarning: Cannot find mlflow module. To enable MLflow logging, install mlflow from PyPI.
  warnings.warn(_MLflowInstrumentation._NO_MLFLOW_WARNING)

Command took 58.50 seconds -- by avalasa@calstatela.edu at 5/14/2023, 9:07:42 AM on QS_ANUSHA

8)Predict price using Linear Regression with Cross Validation

prediction_CV_lR = model_CV_LR.transform(test)
predicted_CV_lR = prediction_CV_lR.select("features", "prediction", "label")
predicted_CV_lR.show(10)

```
prediction_CV_lR = model_CV_LR.transform(test)
predicted_CV_lR = prediction_CV_lR.select("features", "prediction", "label")
predicted_CV_lR.show(10)
```

▸ (1) Spark Jobs

```
+--------------------+------------------+-------+
|            features|        prediction|  label|
+--------------------+------------------+-------+
|[1991.0,13.0,7583...| 4815.534744219389| 8995.0|
|[1994.0,15.0,6960...| 6527.798986880574| 9999.0|
|[2002.0,14.0,1712...|  8457.52107790159| 8950.0|
|[2004.0,16.0,1434...| 2238.519381177146| 4889.0|
|[2006.0,13.0,1483...|15546.763351364061| 7489.0|
|[2006.0,17.0,1276...| 5836.667753734626| 4488.0|
|[2006.0,17.0,1365...| 5872.726446398534| 5589.0|
|[2007.0,12.0,1574...|15185.395905680954|11850.0|
|[2007.0,16.0,6251...|32209.607766159344|34998.0|
|[2011.0,16.0,7325...|  19525.6638359949|11989.0|
+--------------------+------------------+-------+
only showing top 10 rows
```

Command took 1.09 seconds -- by avalasa@calstatela.edu at 5/14/2023, 9:07:45 AM on QS_ANUSHA

9) R2 and RMSE For LR_CV

from pyspark.ml.evaluation import RegressionEvaluator
lr_evaluator_CV_r2 = RegressionEvaluator(predictionCol="prediction", \
labelCol="label",metricName="r2")
lr_evaluator_CV_rmse = RegressionEvaluator(labelCol="label", predictionCol="prediction", metricName="rmse")
#R2 and RMSE For LR_CV
print("r2 For LR_CV: %.2f" % lr_evaluator_CV_r2.evaluate(prediction_CV_lR))
print("RMSE For LR_CV: %.2f" % lr_evaluator_CV_rmse.evaluate(prediction_CV_lR))

```

from pyspark.ml.evaluation import RegressionEvaluator
lr_evaluator_CV_r2 = RegressionEvaluator(predictionCol="prediction", \
                labelCol="label",metricName="r2")

lr_evaluator_CV_rmse = RegressionEvaluator(labelCol="label", predictionCol="prediction", metricName="rmse")
#R2 and RMSE For LR_CV
print("r2 For LR_CV: %.2f" % lr_evaluator_CV_r2.evaluate(prediction_CV_lR))
print("RMSE For LR_CV: %.2f" % lr_evaluator_CV_rmse.evaluate(prediction_CV_lR))
```

▸ (2) Spark Jobs

```
r2 For LR_CV: 0.61
RMSE For LR_CV: 7693.73
```

Command took 5.24 seconds -- by avalasa@calstatela.edu at 5/14/2023, 9:07:56 AM on QS_ANUSHA

RESULT: R2 For LR_CV: 0.61
RMSE For LR_CV: 7693.73
LR_TV_Data_Training_Time = 58.50 seconds

LINEAR REGRESSION TRAIN VALIDATION:

Did the same steps as above with the Cross Validation and used the same lr and lr-assembler which are identical for both cross-validation and train validation split.

1) Create a parameter combination to tune the model.

paramGrid2_TV_LR = ParamGridBuilder()\
.addGrid(lr.maxIter, [10, 40])\
.addGrid(lr.regParam, [0.1, 1.0])\
.build()

```
paramGrid2_TV_LR = ParamGridBuilder()\
.addGrid(lr.maxIter, [10, 40])\
.addGrid(lr.regParam, [0.1, 1.0])\
.build()
```

Command took 0.02 seconds -- by avalasa@calstatela.edu at 5/14/2023, 9:10:40 AM on QS_ANUSHA

2) Create an evaluator to evaluate a model with R2

from pyspark.ml.evaluation import RegressionEvaluator
lr_tv_evaluator = RegressionEvaluator(predictionCol="prediction", \
labelCol="label",metricName="r2")

```
from pyspark.ml.evaluation import RegressionEvaluator
lr_tv_evaluator = RegressionEvaluator(predictionCol="prediction", \
labelCol="label",metricName="r2")
```

Command took 0.03 seconds -- by avalasa@calstatela.edu at 5/14/2023, 9:10:42 AM on QS_ANUSHA

3) Create a Tarin_Validation Split to tune the model for the generalization. The default folds are 3.

tv_lr = TrainValidationSplit(estimator=pipeline_lr, evaluator=lr_tv_evaluator, estimatorParamMaps=paramGrid2_TV_LR, trainRatio=0.8)

```
tv_lr = TrainValidationSplit(estimator=pipeline_lr, evaluator=lr_tv_evaluator, estimatorParamMaps=paramGrid2_TV_LR, trainRatio=0.8)

```

Command took 0.03 seconds -- by avalasa@calstatela.edu at 5/14/2023, 9:10:45 AM on QS_ANUSHA

5) Build a model with the train Dataframe of cross validator

model1_tv_lr = tv_lr.fit(train)

```
1  model1_tv_lr = tv_lr.fit(train)
```

▸ (14) Spark Jobs

```
Command took 20.46 seconds -- by avalasa@calstatela.edu at 5/14/2023, 9:10:48 AM on QS_ANUSHA
```

6) Predicted price using Linear Regression Train Validation

prediction_tv_lr = model1_tv_lr.transform(test)
predicted_tv_lr = prediction_tv_lr.select("features", "prediction", "label")
predicted_tv_lr.show(10)

```
1  prediction_tv_lr = model1_tv_lr.transform(test)
2  predicted_tv_lr = prediction_tv_lr.select("features", "prediction", "label")
3  predicted_tv_lr.show(10)
```

▸ (1) Spark Jobs

```
+--------------------+--------------------+-------+
|            features|          prediction|  label|
+--------------------+--------------------+-------+
|[1991.0,13.0,7583...| 4815.534744219389| 8995.0|
|[1994.0,15.0,6960...| 6527.798986880574| 9999.0|
|[2002.0,14.0,1712...|  8457.52107790159| 8950.0|
|[2004.0,16.0,1434...| 2238.519381177146| 4889.0|
|[2006.0,13.0,1483...|15546.763351364061| 7489.0|
|[2006.0,17.0,1276...| 5836.667753734626| 4488.0|
|[2006.0,17.0,1365...| 5872.726446398534| 5589.0|
|[2007.0,12.0,1574...|15185.395905680954|11850.0|
|[2007.0,16.0,6251...|32209.607766159344|34998.0|
|[2011.0,16.0,7325...|  19525.6638359949|11989.0|
+--------------------+--------------------+-------+
only showing top 10 rows

Command took 1.07 seconds -- by avalasa@calstatela.edu at 5/14/2023, 9:10:53 AM on QS_ANUSHA
```

7) R2 and RMSE For Linear Regression Train Validation

```
from pyspark.ml.evaluation import RegressionEvaluator
lr_tv_evaluator_r2 = RegressionEvaluator(predictionCol="prediction", \
            labelCol="label",metricName="r2")

lr_tv_evaluator_rmse = RegressionEvaluator(labelCol="label", predictionCol="prediction",
metricName="rmse")
#R2 and RMSE For LR_TV
print("r2 For LR CV: %.2f" % lr_tv_evaluator_r2.evaluate(prediction_tv_lr))
print("RMSE for LT TV: %.2f" % lr_tv_evaluator_rmse.evaluate(prediction_tv_lr))
```

```
from pyspark.ml.evaluation import RegressionEvaluator
lr_tv_evaluator_r2 = RegressionEvaluator(predictionCol="prediction", \
                labelCol="label",metricName="r2")

lr_tv_evaluator_rmse = RegressionEvaluator(labelCol="label", predictionCol="prediction", metricName="rmse")
#R2 and RMSE For LR_TV
print("r2 For LR CV: %.2f" % lr_tv_evaluator_r2.evaluate(prediction_tv_lr))
print("RMSE for LT TV: %.2f" % lr_tv_evaluator_rmse.evaluate(prediction_tv_lr))

```

```
▸ (2) Spark Jobs
r2 For LR CV: 0.61
RMSE for LT TV: 7693.73
Command took 5.98 seconds -- by avalasa@calstatela.edu at 5/14/2023, 9:11:00 AM on QS_ANUSHA
```

Result: R2 For LR TV: 0.61
RMSE for LT TV: 7693.73
LR_TV_Data_Training_Time = 20.46 seconds

Algorithm 2: Gradient Boost Tree(GBT)

GBT Cross Validation:

1)Create a vector assembler with input columns

gbt_assembler = VectorAssembler(inputCols = ["year", "city_fuel_economy", "mileage", "owner_count", "highway_fuel_economy", "engine_displacement", "savings_amount"], outputCol="features").setHandleInvalid("skip")

```
gbt_assembler = VectorAssembler(inputCols = ["year", "city_fuel_economy", "mileage", "owner_count", "highway_fuel_economy", "engine_displacement",
"savings_amount"], outputCol="features").setHandleInvalid("skip")
```

```
Command took 0.05 seconds -- by avalasa@calstatela.edu at 5/14/2023, 9:11:58 AM on QS_ANUSHA
```

2) Create GBT REgressor

gbt = GBTRegressor(labelCol="label") #,featuresCol="features",maxIter=10

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

```
Command took 0.04 seconds -- by avalasa@calstatela.edu at 5/14/2023, 9:12:02 AM on QS_ANUSHA
```

3) Create ParamGridBuilder with different parameters

paramGrid_CV_GBT = ParamGridBuilder()\
.addGrid(gbt.maxDepth, [2, 5])\
.addGrid(gbt.maxIter, [10, 20])\
.addGrid(gbt.minInfoGain, [0,0])\
.build()

```
paramGrid_CV_GBT = ParamGridBuilder()\
.addGrid(gbt.maxDepth, [2, 5])\
.addGrid(gbt.maxIter, [10, 20])\
.addGrid(gbt.minInfoGain, [0,0])\
.build()
```

Command took 0.02 seconds -- by avalasa@calstatela.edu at 5/14/2023, 9:12:04 AM on QS_ANUSHA

4) Create an evaluator to evaluate a model with R2

gbt_evaluator = RegressionEvaluator(predictionCol="prediction", \
labelCol="label",metricName="r2")

```
gbt_evaluator = RegressionEvaluator(predictionCol="prediction", \
labelCol="label",metricName="r2")
```

Command took 0.04 seconds -- by avalasa@calstatela.edu at 5/14/2023, 9:12:06 AM on QS_ANUSHA

5) Creating the Pipeline

pipeline_GBT = Pipeline(stages=[gbt_assembler, gbt])

```
pipeline_GBT = Pipeline(stages=[gbt_assembler, gbt])
```

Command took 0.02 seconds -- by avalasa@calstatela.edu at 5/14/2023, 9:12:09 AM on QS_ANUSHA

6) create a cross-validator to tune the model for the generalization.

GBT_CV = CrossValidator(estimator=pipeline_GBT, evaluator=gbt_evaluator, estimatorParamMaps=paramGrid_CV_GBT, numFolds = 3)

```
1  GBT_CV = CrossValidator(estimator=pipeline_GBT, evaluator=gbt_evaluator, estimatorParamMaps=paramGrid_CV_GBT, numFolds = 3)
```

Command took 0.01 seconds -- by avalasa@calstatela.edu at 5/14/2023, 9:12:12 AM on QS_ANUSHA

7) creating a model to train

model_GBT = GBT_CV.fit(train)

```
1  model_GBT = GBT_CV.fit(train)
```

▸ (50) Spark Jobs

Command took 12.81 minutes -- by avalasa@calstatela.edu at 5/14/2023, 9:12:15 AM on QS_ANUSHA

8) Predicting the Price

prediction_GBT = model_GBT.transform(test)
predicted_GBT = prediction_GBT.select("features", "prediction", "label")
predicted_GBT.show(10)

```
1  prediction_GBT = model_GBT.transform(test)
2  predicted_GBT = prediction_GBT.select("features", "prediction", "label")
3  predicted_GBT.show(10)
```

▸ (1) Spark Jobs

```
+--------------------+------------------+-------+
|            features|        prediction|  label|
+--------------------+------------------+-------+
|[1991.0,13.0,7583...| 7802.411208415451| 8995.0|
|[1994.0,15.0,6960...| 9705.584285513498| 9999.0|
|[2002.0,14.0,1712...|12229.696964204248| 8950.0|
|[2004.0,16.0,1434...| 5727.033835482392| 4889.0|
|[2006.0,13.0,1483...| 6224.394710595671| 7489.0|
|[2006.0,17.0,1276...| 6217.972778736959| 4488.0|
|[2006.0,17.0,1365...| 6342.372046874887| 5589.0|
|[2007.0,12.0,1574...| 8311.900035425657|11850.0|
|[2007.0,16.0,6251...| 11033.35257580522|34998.0|
|[2011.0,16.0,7325...|  9456.82107384449|11989.0|
+--------------------+------------------+-------+
only showing top 10 rows
```

Command took 1.77 seconds -- by avalasa@calstatela.edu at 5/14/2023, 9:12:18 AM on QS_ANUSHA

9) R2 and RMSE for GBT Cross Validation

gbt_evaluator_r2 = RegressionEvaluator(labelCol="label", predictionCol="prediction", metricName="rmse")
print("RMSE_GBT_CV: %.2f" % gbt_evaluator_r2.evaluate(prediction_GBT))
gbt_evaluator_rmse = RegressionEvaluator(predictionCol="prediction", \
labelCol="label",metricName="r2")
print("R2_GBT_CV: %.2f" % gbt_evaluator_rmse.evaluate(prediction_GBT)

```
gbt_evaluator_r2 = RegressionEvaluator(labelCol="label", predictionCol="prediction", metricName="rmse")
print("RMSE_GBT_CV: %.2f" % gbt_evaluator_r2.evaluate(prediction_GBT))

gbt_evaluator_rmse = RegressionEvaluator(predictionCol="prediction", \
                labelCol="label",metricName="r2")
print("R2_GBT_CV: %.2f" % gbt_evaluator_rmse.evaluate(prediction_GBT))

```

▸ (2) Spark Jobs

```
RMSE_GBT_CV: 7059.37
R2_GBT_CV: 0.67
```

Command took 5.63 seconds -- by avalasa@calstatela.edu at 5/14/2023, 9:12:25 AM on QS_ANUSHA

RMSE_GBT_CV: 7059.37
R2_GBT_CV: 0.67
Training time: 12.81 minutes

GBT_Train_Validation: -

1)Create a Param Grid with different parameters

paramGrid_TV_GBT = ParamGridBuilder()\
.addGrid(gbt.maxDepth, [2, 5])\
.addGrid(gbt.maxIter, [20, 40])\
.addGrid(gbt.maxBins, [15, 30])\
.addGrid(gbt.stepSize, [0.05, 0.1])\
.addGrid(gbt.minInfoGain, [0,0])\
.build()

```
paramGrid_TV_GBT = ParamGridBuilder()\
.addGrid(gbt.maxDepth, [2, 5])\
.addGrid(gbt.maxIter, [20, 40])\
.addGrid(gbt.maxBins, [15, 30])\
.addGrid(gbt.stepSize, [0.05, 0.1])\
.addGrid(gbt.minInfoGain, [0,0])\
.build()
```

Command took 0.02 seconds -- by avalasa@calstatela.edu at 5/14/2023, 9:12:30 AM on QS_ANUSHA

2) Create Train Validation split for GBT

GBT_TV = TrainValidationSplit(estimator=pipeline_GBT, evaluator=gbt_evaluator, estimatorParamMaps=paramGrid_TV_GBT, trainRatio=0.8)

```
GBT_TV = TrainValidationSplit(estimator=pipeline_GBT, evaluator=gbt_evaluator, estimatorParamMaps=paramGrid_TV_GBT, trainRatio=0.8)

```

Command took 0.01 seconds -- by avalasa@calstatela.edu at 5/14/2023, 9:12:33 AM on QS_ANUSHA

3) Train the Data

model_GBT_TV = GBT_TV.fit(train)

```
1  model_GBT_TV = GBT_TV.fit(train)
```

▸ (50) Spark Jobs

Command took 53.94 minutes -- by avalasa@calstatela.edu at 5/14/2023, 9:12:36 AM on QS_ANUSHA

4) Predicting the Price

prediction_GBT = model_GBT_TV.transform(test)
predicted_GBT = prediction_GBT.select("features", "prediction", "label")
predicted_GBT.show(10)

```
1  prediction_GBT = model_GBT_TV.transform(test)
2  predicted_GBT = prediction_GBT.select("features", "prediction", "label")
3  predicted_GBT.show(10)
```

▸ (1) Spark Jobs

```
+------------------+-----------------+-------+
|          features|       prediction|  label|
+------------------+-----------------+-------+
|[1991.0,13.0,7583...| 6939.270627278039| 8995.0|
|[1994.0,15.0,6960...| 8592.925995893613| 9999.0|
|[2002.0,14.0,1712...|31973.636533407356| 8950.0|
|[2004.0,16.0,1434...| 3690.563577434275| 4889.0|
|[2006.0,13.0,1483...| 5799.857631250606| 7489.0|
|[2006.0,17.0,1276...| 5410.587937303878| 4488.0|
|[2006.0,17.0,1365...| 4786.050468158109| 5589.0|
|[2007.0,12.0,1574...| 9136.380471439339|11850.0|
|[2007.0,16.0,6251...| 72125.95353113592|34998.0|
|[2011.0,16.0,7325...|11526.751620197414|11989.0|
+------------------+-----------------+-------+
only showing top 10 rows
```

Command took 2.01 seconds -- by avalasa@calstatela.edu at 5/14/2023, 9:12:39 AM on QS_ANUSHA

GBT Train Validation Split RMSE & R2

gbt_evaluator_rmse = RegressionEvaluator(labelCol="label", predictionCol="prediction", metricName="rmse")

print("RMSE_GBT_TV: %.2f" % gbt_evaluator_rmse.evaluate(prediction_GBT))
gbt_evaluator_r2 = RegressionEvaluator(predictionCol="prediction", \
labelCol="label",metricName="r2")
print("R2_GBT_TV: %.2f" % gbt_evaluator_r2.evaluate(prediction_GBT))

```
gbt_evaluator_rmse = RegressionEvaluator(labelCol="label", predictionCol="prediction", metricName="rmse")
print("RMSE_GBT_TV: %.2f" % gbt_evaluator_rmse.evaluate(prediction_GBT))


gbt_evaluator_r2 = RegressionEvaluator(predictionCol="prediction", \
                labelCol="label",metricName="r2")
print("R2_GBT_TV: %.2f" % gbt_evaluator_r2.evaluate(prediction_GBT))

```

▸ (2) Spark Jobs

```
RMSE_GBT_TV: 7009.82
R2_GBT_TV: 0.67
```

Command took 4.22 seconds -- by avalasa@calstatela.edu at 5/14/2023, 9:12:47 AM on QS_ANUSHA

RMSE_GBT_TV: 7009.82
R2_GBT_TV: 0.67
GBT TV Data Training Time = 53.94 Minutes

Algorithm 3: Random Forest Regression

Random Forest with Cross Validation: -

1)Create a vector assembler with input columns

assembler_Rf = VectorAssembler(inputCols = ["year", "city_fuel_economy", "mileage", "owner_count", "highway_fuel_economy", "engine_displacement", "savings_amount"], outputCol="features").setHandleInvalid("skip")

```
assembler_Rf = VectorAssembler(inputCols = ["year", "city_fuel_economy", "mileage", "owner_count", "highway_fuel_economy", "engine_displacement",
"savings_amount"], outputCol="features").setHandleInvalid("skip")
```

Command took 0.06 seconds -- by avalasa@calstatela.edu at 5/14/2023, 9:13:00 AM on QS_ANUSHA

2) Train the Model

training_Rf = assembler_Rf.transform(train)

```
training_Rf = assembler_Rf.transform(train)
#training.show()
```

Command took 0.06 seconds -- by avalasa@calstatela.edu at 5/14/2023, 9:13:04 AM on QS_ANUSHA

3) Creating a regressor

rf = RandomForestRegressor(labelCol='label') #,featuresCol="features", numTrees=10, maxDepth=5

```
1  rf = RandomForestRegressor(labelCol='label') #,featuresCol="features", numTrees=10, maxDepth=5
```

```
Command took 0.03 seconds -- by avalasa@calstatela.edu at 5/14/2023, 9:13:07 AM on QS_ANUSHA
```

4) **Create a parameter combination to tune the model.**

paramGrid_CV_Rf = ParamGridBuilder()\
.addGrid(rf.maxDepth, [5, 10, 15])\
.addGrid(rf.numTrees, [20, 50, 100])\
.addGrid(rf.minInfoGain, [0.0])\
.build()

```
paramGrid_CV_Rf = ParamGridBuilder()\
    .addGrid(rf.maxDepth, [5, 10, 15])\
    .addGrid(rf.numTrees, [20, 50, 100])\
    .addGrid(rf.minInfoGain, [0.0])\
    .build()
```

```
ommand took 0.03 seconds -- by avalasa@calstatela.edu at 5/14/2023, 9:13:10 AM on QS_ANUSHA
```

5) Create an evaluator to evaluate a model with R2

rf_evaluator = RegressionEvaluator(predictionCol="prediction", \
labelCol="label",metricName="r2")

```
1  rf_evaluator = RegressionEvaluator(predictionCol="prediction", \
2  labelCol="label",metricName="r2")
```

```
Command took 0.03 seconds -- by avalasa@calstatela.edu at 5/14/2023, 9:13:14 AM on QS_ANUSHA
```

6) **Create a pipeline, which is to sequence the tasks.**
pipeline_rf = Pipeline(stages=[assembler_Rf, rf])

```
1  pipeline_rf = Pipeline(stages=[assembler_Rf, rf])
```

```
Command took 0.01 seconds -- by avalasa@calstatela.edu at 5/14/2023, 9:13:16 AM on QS_ANUSHA
```

7) Create a cross-validator to tune the model for the generalization.
cv_rf = CrossValidator(estimator=pipeline_rf, evaluator=rf_evaluator, estimatorParamMaps=paramGrid_CV_Rf)

```
1  cv_rf = CrossValidator(estimator=pipeline_rf, evaluator=rf_evaluator, estimatorParamMaps=paramGrid_CV_Rf)
```

```
Command took 0.02 seconds -- by avalasa@calstatela.edu at 5/14/2023, 9:13:19 AM on QS_ANUSHA
```

8) Train the Model
model_CV_rf = cv_rf.fit(train)

```
1  model_CV_rf = cv_rf.fit(train)
```

```
▸ (52) Spark Jobs

Command took 32.08 minutes -- by avalasa@calstatela.edu at 5/14/2023, 9:13:21 AM on QS_ANUSHA
```

9) Price Predicton
prediction_CV_rf = model_CV_rf.transform(test)
predicted_CV_rf = prediction_CV_rf.select("features", "prediction", "label")
predicted_CV_rf.show(10)

```
1  prediction_CV_rf = model_CV_rf.transform(test)
2  predicted_CV_rf = prediction_CV_rf.select("features", "prediction", "label")
3  predicted_CV_rf.show(10)
```

```
▸ (1) Spark Jobs
+--------------------+------------------+--------+
|            features|        prediction|   label|
+--------------------+------------------+--------+
|[1991.0,13.0,7583...|          10713.51|  8995.0|
|[1994.0,15.0,6960...|          12771.38|  9999.0|
|[2002.0,14.0,1712...| 9106.353333333334|  8950.0|
|[2004.0,16.0,1434...| 5547.738690476191|  4889.0|
|[2006.0,13.0,1483...| 7960.288333333333|  7489.0|
|[2006.0,17.0,1276...| 6426.022777777777|  4488.0|
|[2006.0,17.0,1365...|6088.0498611111125|  5589.0|
|[2007.0,12.0,1574...| 8685.243333333334|11850.0|
|[2007.0,16.0,6251...|40237.435333333335|34998.0|
|[2011.0,16.0,7325...|15776.226499999999|11989.0|
+--------------------+------------------+--------+
only showing top 10 rows

Command took 2.36 seconds -- by avalasa@calstatela.edu at 5/14/2023, 9:13:24 AM on QS_ANUSHA
```

10) R2 and RMSE for Random Forest Cross Validation

```
rf_evaluator_CV_rmse = RegressionEvaluator(labelCol="label", predictionCol="prediction",
metricName="rmse")
print("RMSE for CV_RF: %.2f" % rf_evaluator_CV_rmse.evaluate(prediction_CV_rf))
 rf_evaluator_CV_r2 = RegressionEvaluator(predictionCol="prediction", \
              labelCol="label",metricName="r2")
print("R2 for CV_RF: %.2f" % rf_evaluator_CV_r2.evaluate(prediction_CV_rf))
```

```
1  rf_evaluator_CV_rmse = RegressionEvaluator(labelCol="label", predictionCol="prediction", metricName="rmse")
2  print("RMSE for CV_RF: %.2f" % rf_evaluator_CV_rmse.evaluate(prediction_CV_rf))
3
4
5  rf_evaluator_CV_r2 = RegressionEvaluator(predictionCol="prediction", \
6                 labelCol="label",metricName="r2")
7  print("R2 for CV_RF: %.2f" % rf_evaluator_CV_r2.evaluate(prediction_CV_rf))
```

```
▸ (2) Spark Jobs
RMSE for CV_RF: 6197.13
R2 for CV_RF: 0.74
Command took 4.42 seconds -- by avalasa@calstatela.edu at 5/14/2023, 9:13:32 AM on QS_ANUSHA
```

RMSE for CV_RF: 6197.13
R2 for CV_RF: 0.74
Random Forest CV data training time: 32.08 minutes

Random Forest with Train Validation: -

1)Creating ParamGridBuilder along with different parameters

```
paramGrid_TV_Rf = ParamGridBuilder()\
   .addGrid(rf.maxDepth, [5, 10, 15])\
   .addGrid(rf.numTrees, [20, 50, 100])\
   .addGrid(rf.featureSubsetStrategy, ['auto', 'sqrt', 'log2'])\
   .addGrid(rf.minInfoGain, [0.0])\
   .build()
```

```
1  paramGrid_TV_Rf = ParamGridBuilder()\
2      .addGrid(rf.maxDepth, [5, 10, 15])\
3      .addGrid(rf.numTrees, [20, 50, 100])\
4      .addGrid(rf.featureSubsetStrategy, ['auto', 'sqrt', 'log2'])\
5      .addGrid(rf.minInfoGain, [0.0])\
6      .build()
7
```

```
Command took 0.02 seconds -- by avalasa@calstatela.edu at 5/14/2023, 9:14:38 AM on QS_ANUSHA
```

2)Creating TrainValidation Split

Tv_rf = TrainValidationSplit(estimator=pipeline_rf, evaluator=rf_evaluator, estimatorParamMaps=paramGrid_TV_Rf, trainRatio=0.9)

```
Tv_rf = TrainValidationSplit(estimator=pipeline_rf, evaluator=rf_evaluator, estimatorParamMaps=paramGrid_TV_Rf, trainRatio=0.9)
```

Command took 0.01 seconds -- by avalasa@calstatela.edu at 5/14/2023, 9:14:40 AM on QS_ANUSHA

3)Training the Model

model_tv_rf = Tv_rf.fit(train)

```
model_tv_rf = Tv_rf.fit(train)
```

▸ (52) Spark Jobs

```
/databricks/spark/python/pyspark/ml/util.py:886: UserWarning: Cannot find mlflow module. To enable MLflow logging, install mlflow from PyPI.
  warnings.warn(_MLflowInstrumentation._NO_MLFLOW_WARNING)
```

Command took 35.74 minutes -- by avalasa@calstatela.edu at 5/16/2023, 12:45:20 AM on QS_Anusha

4)Predicting the Price

prediction_tv_rf = model_tv_rf.transform(test)
predicted_tv_rf = prediction_tv_rf.select("features", "prediction", "label")
predicted_tv_rf.show(10)

```
prediction_tv_rf = model_tv_rf.transform(test)
predicted_tv_rf = prediction_tv_rf.select("features", "prediction", "label")
predicted_tv_rf.show(10)
```

▸ (1) Spark Jobs

```
+--------------------+------------------+-------+
|            features|        prediction|  label|
+--------------------+------------------+-------+
|[1997.0,17.0,1645...|           4841.15| 1500.0|
|[2002.0,14.0,1712...| 6577.741666666667| 8950.0|
|[2003.0,18.0,1482...|           3866.85| 2800.0|
|[2005.0,15.0,2820...|30949.783333333336|24900.0|
|[2006.0,15.0,1282...|            5755.7| 4995.0|
|[2006.0,17.0,1365...|           6358.25| 5589.0|
|[2006.0,20.0,8719...|7816.9857142857145| 5900.0|
|[2008.0,15.0,310....| 44642.00833333334|48999.0|
|[2008.0,17.0,1484...|          6421.875| 5495.0|
|[2008.0,22.0,1058...| 6423.666666666666| 5995.0|
+--------------------+------------------+-------+
only showing top 10 rows
```

Command took 3.90 seconds -- by avalasa@calstatela.edu at 5/16/2023, 12:45:25 AM on QS_Anusha

prediction_tv_rf.limit(1000).show()

```
prediction_tv_rf.limit(1000).show()
```

▸ (1) Spark Jobs

```
+----+----------------+-------+--------+-----------+--------+--------------------+------------+----------+-----------------+----------+--------------+----
--------+------+------------+-------------+-----------+--------------------+-----------------+
|year|city_fuel_economy|  label| mileage|owner_count|latitude|highway_fuel_economy|daysonmarket|dealer_zip|engine_displacement|listing_id|savings_amount|sell
r_rating| sp_id|has_accidents|frame_damaged|theft_title|            features|       prediction|
+----+----------------+-------+--------+-----------+--------+--------------------+------------+----------+-----------------+----------+--------------+----
--------+------+------------+-------------+-----------+--------------------+-----------------+
|1997|            17.0| 1500.0|164524.0|          2|  42.459|                27.0|          84|     50613|             3800| 274364588|             0|
4|284048|        FALSE|        FALSE|      FALSE|[1997.0,17.0,1645...|          4841.15|
|2002|            14.0| 8950.0|171277.0|          8| 30.2866|                17.0|           2|     32246|             4000| 281585572|          1715|
4|379865|        FALSE|        FALSE|      FALSE|[2002.0,14.0,1712...| 6577.741666666667|
|2003|            18.0| 2800.0|148234.0|          3|  42.459|                25.0|          77|     50613|             3000| 274961437|             0|
4|284048|        FALSE|        FALSE|      FALSE|[2003.0,18.0,1482...|          3866.85|
|2005|            15.0|24900.0| 28200.0|          3| 30.2869|                23.0|          20|     32246|             4600| 279972616|          3234|
4|283437|         TRUE|        FALSE|      FALSE|[2005.0,15.0,2820...|30949.783333333336|
|2006|            15.0| 4995.0|128256.0|          3|  40.737|                21.0|           8|     51632|             3500| 281055812|             0|
5|335614|         TRUE|        FALSE|      FALSE|[2006.0,15.0,1282...|           5755.7|
|2006|            17.0| 5589.0|136583.0|          3| 30.2864|                23.0|          12|     32216|             3500| 280698144|            32|
4|289407|        FALSE|        FALSE|      FALSE|[2006.0,17.0,1365...|          6358.25|
|2006|            20.0| 5900.0| 87199.0|          3| 30.2586|                28.0|          12|     32217|             1800| 280740419|          1023|
null|438462|        FALSE|        FALSE|      FALSE|[2006.0,20.0,8719...|7816.9857142857145|
|2008|            15.0|48999.0|   310.0|          2| 30.2869|                24.0|         333|     32246|             7000| 255096079|          4253|
Command took 1.23 seconds -- by avalasa@calstatela.edu at 5/16/2023, 12:45:29 AM on QS_Anusha
```

5)R2 and RMSE for RF Train Validation:-

rf_tv_evaluator_rmse = RegressionEvaluator(labelCol="label", predictionCol="prediction", metricName="rmse")
print("RMSE for RF_TV: %.2f" % rf_tv_evaluator_rmse.evaluate(prediction_tv_rf))
rf_tv_evaluator_r2 = RegressionEvaluator(predictionCol="prediction", \
labelCol="label",metricName="r2")
print("R2 for RF_TV: %.2f" % rf_tv_evaluator_r2.evaluate(prediction_tv_rf))

```
1  rf_tv_evaluator_rmse = RegressionEvaluator(labelCol="label", predictionCol="prediction", metricName="rmse")
2  print("RMSE for RF_TV: %.2f" % rf_tv_evaluator_rmse.evaluate(prediction_tv_rf))
3
4
5  rf_tv_evaluator_r2 = RegressionEvaluator(predictionCol="prediction", \
6                  labelCol="label",metricName="r2")
7  print("R2 for RF_TV: %.2f" % rf_tv_evaluator_r2.evaluate(prediction_tv_rf))
```

▸ (2) Spark Jobs

```
RMSE for RF_TV: 6758.30
R2 for RF_TV: 0.73
Command took 6.52 seconds -- by avalasa@calstatela.edu at 5/16/2023, 12:45:32 AM on QS_Anusha
```

RMSE for TV_RF: 6758.30
R2 for TV_RF: 0.73
Random Forest CV data training time: 35.74 minutes

Finding the R2 and RMSE without tuning the dataset

1)Training the model

model_RF = pipeline_rf.fit(train)

```
1  model_RF = pipeline_rf.fit(train)
```

▸ (8) Spark Jobs

Command took 10.73 seconds -- by avalasa@calstatela.edu at 5/16/2023, 12:45:35 AM on QS_Anusha

2)Creating Model

rfModel = model_RF.stages[-1]
#print(rfModel.toDebugString)

```
1  rfModel = model_RF.stages[-1]
2  #print(rfModel.toDebugString)
```

Command took 0.02 seconds -- by avalasa@calstatela.edu at 5/16/2023, 12:45:38 AM on QS_Anusha

3)Feature Importance

featureImp = pd.DataFrame(list(zip(assembler_Rf.getInputCols(), rfModel.featureImportances)), columns=["feature", "importance"])
featureImp.sort_values(by="importance", ascending="False")

```
1  featureImp = pd.DataFrame(list(zip(assembler_Rf.getInputCols(), rfModel.featureImportances)), columns=["feature", "importance"])
2  featureImp.sort_values(by="importance", ascending="False")
```

	feature	importance
3	owner_count	0.044117
6	savings_amount	0.065726
1	city_fuel_economy	0.092745
4	highway_fuel_economy	0.118275
5	engine_displacement	0.145099
0	year	0.230173
2	mileage	0.303864

Command took 0.91 seconds -- by avalasa@calstatela.edu at 5/16/2023, 12:45:41 AM on QS_Anusha

4)Creating RF regression

rf = RandomForestRegressor(labelCol="label", featuresCol="features", numTrees=30, maxDepth=15)
# combine stages into pipeline
pipeline_RF_1 = Pipeline(stages= [assembler_Rf, rf])
model_rf = pipeline_RF_1.fit(train)

```
rf = RandomForestRegressor(labelCol="label", featuresCol="features", numTrees=30, maxDepth=15)

# combine stages into pipeline
pipeline_RF_1 = Pipeline(stages= [assembler_Rf, rf])
model_rf = pipeline_RF_1.fit(train)
```

▸ (18) Spark Jobs

Command took 1.79 minutes -- by avalasa@calstatela.edu at 5/16/2023, 12:45:47 AM on QS_Anusha

5)Predicting the Price

prediction_rf1 = model_rf.transform(test)
predicted_rf1 = prediction_rf1.select("features", "prediction", "label")
predicted_rf1.show(10)

```
prediction_rf1 = model_rf.transform(test)
predicted_rf1 = prediction_rf1.select("features", "prediction", "label")
predicted_rf1.show(10)
```

▸ (1) Spark Jobs

```
+--------------------+------------------+-------+
|            features|        prediction|  label|
+--------------------+------------------+-------+
|[1997.0,17.0,1645...|3879.0555555555557| 1500.0|
|[2002.0,14.0,1712...|10613.944444444445| 8950.0|
|[2003.0,18.0,1482...|3778.5333333333333| 2800.0|
|[2005.0,15.0,2820...|31339.666666666668|24900.0|
|[2006.0,15.0,1282...| 6070.874358974358| 4995.0|
|[2006.0,17.0,1365...|5875.3076923076915| 5589.0|
|[2006.0,20.0,8719...|            7449.1| 5900.0|
|[2008.0,15.0,310....|           48372.8|48999.0|
|[2008.0,17.0,1484...| 6434.266666666666| 5495.0|
|[2008.0,22.0,1058...| 6625.144444444444| 5995.0|
+--------------------+------------------+-------+
only showing top 10 rows
```

Command took 3.67 seconds -- by avalasa@calstatela.edu at 5/16/2023, 12:45:50 AM on QS_Anusha

predicted_rf1.limit(1000).show()

```
2  predicted_rf1.limit(1000).show()
```

▸ (1) Spark Jobs

```
+--------------------+------------------+-------+
|            features|        prediction|  label|
+--------------------+------------------+-------+
|[1997.0,17.0,1645...|3879.0555555555557| 1500.0|
|[2002.0,14.0,1712...|10613.944444444445| 8950.0|
|[2003.0,18.0,1482...|3778.5333333333333| 2800.0|
|[2005.0,15.0,2820...|31339.666666666668|24900.0|
|[2006.0,15.0,1282...| 6070.874358974358| 4995.0|
|[2006.0,17.0,1365...|5875.3076923076915| 5589.0|
|[2006.0,20.0,8719...|            7449.1| 5900.0|
|[2008.0,15.0,310....|           48372.8|48999.0|
|[2008.0,17.0,1484...| 6434.266666666666| 5495.0|
|[2008.0,22.0,1058...| 6625.144444444444| 5995.0|
|[2009.0,18.0,2781...| 6309.433333333333| 2300.0|
|[2011.0,13.0,1299...|13357.269999999999|17900.0|
|[2012.0,14.0,9504...|18580.633333333335|19391.0|
|[2012.0,15.0,1668...|           14699.9| 8389.0|
|[2012.0,18.0,7600...|11787.650584795321|12000.0|
|[2012.0,21.0,5665...|14257.787777777778|11000.0|
|[2012.0,25.0,1147...| 8180.689040404041| 9489.0|
|[2013.0,15.0,1632...|           28282.5|21500.0|
```

Command took 0.74 seconds -- by avalasa@calstatela.edu at 5/16/2023, 12:45:56 AM on QS_Anusha

R2 and RMSE for without tuning the RF: -

rf_evaluator_rmse = RegressionEvaluator(labelCol="label", predictionCol="prediction", metricName="rmse")
print("RMSE for only RF: %.2f" % rf_evaluator_rmse.evaluate(prediction_rf1))
rf_evaluator_r2 = RegressionEvaluator(predictionCol="prediction", \
labelCol="label",metricName="r2")
print("R2 for only RF: %.2f" % rf_evaluator_r2.evaluate(prediction_rf1))

```
1  rf_evaluator_rmse = RegressionEvaluator(labelCol="label", predictionCol="prediction", metricName="rmse")
2  print("RMSE for only RF: %.2f" % rf_evaluator_rmse.evaluate(prediction_rf1))
3
4
5  rf_evaluator_r2 = RegressionEvaluator(predictionCol="prediction", \
6                  labelCol="label",metricName="r2")
7  print("R2 for only RF: %.2f" % rf_evaluator_r2.evaluate(prediction_rf1))
8
```

▸ (2) Spark Jobs

```
RMSE for only RF: 6937.48
R2 for only RF: 0.72
```

Command took 5.84 seconds -- by avalasa@calstatela.edu at 5/16/2023, 12:46:00 AM on QS_Anusha

RMSE for TV_RF: 6937.48
R2 for TV_RF: 0.72
Random Forest CV data training time: 10.73 seconds

Algorithm 4: Factorization Machine Regression

Factorization Machine Regression Cross Validation:

1)Creating a vector Assembler

assembler_fm = VectorAssembler(inputCols = ["year", "city_fuel_economy", "mileage", "owner_count", "highway_fuel_economy", "engine_displacement", "savings_amount"], outputCol="features").setHandleInvalid("skip")

```
assembler_fm = VectorAssembler(inputCols = ["year", "city_fuel_economy", "mileage", "owner_count", "highway_fuel_economy", "engine_displacement",
"savings_amount"], outputCol="features").setHandleInvalid("skip")
```

Command took 0.10 seconds -- by avalasa@calstatela.edu at 5/14/2023, 12:35:57 AM on QS_Anusha

2)Training Data Frame

training_fm = assembler_fm.transform(train).select(col("features"))

```
training_fm = assembler_fm.transform(train).select(col("features"))
#training.show()
```

3)Creating the Regressor

fm = FMRegressor(labelCol='label') #,featuresCol="features", numTrees=10, maxDepth=5

```
fm = FMRegressor(labelCol='label') #,featuresCol="features", numTrees=10, maxDepth=5
```

Command took 0.05 seconds -- by avalasa@calstatela.edu at 5/14/2023, 12:36:04 AM on QS_Anusha

4)Creating the Param Grid

paramGrid_CV_fm = ParamGridBuilder() \
  .addGrid(fm.stepSize, [0.1, 1]) \
  .build()

```
# paramGrid_CV_fm = ParamGridBuilder()\
#   .addGrid(fm.factorSize, [4, 7]) \
#   .addGrid(fm.stepSize, [50, 90]) \
#   .addGrid(fm.regParam, [0.0]) \
# .build()


paramGrid_CV_fm = ParamGridBuilder() \
  .addGrid(fm.stepSize, [0.1, 1]) \
  .build()
```

Command took 0.03 seconds -- by avalasa@calstatela.edu at 5/14/2023, 12:39:00 AM on QS_Anusha

5)Creating Evaluator

from pyspark.ml.evaluation import RegressionEvaluator
fm_evaluator = RegressionEvaluator(predictionCol="prediction", \
labelCol="label",metricName="r2")

```
1  from pyspark.ml.evaluation import RegressionEvaluator
2  fm_evaluator = RegressionEvaluator(predictionCol="prediction", \
3  labelCol="label",metricName="r2")
```

```
Command took 0.04 seconds -- by avalasa@calstatela.edu at 5/14/2023, 12:39:03 AM on QS_Anusha
```

7)Creating Pipeline

pipeline_fm = Pipeline(stages=[assembler_fm, fm])

```
1  pipeline_fm = Pipeline(stages=[assembler_fm, fm])
```

```
Command took 0.02 seconds -- by avalasa@calstatela.edu at 5/14/2023, 12:39:06 AM on QS_Anusha
```

8)Creating the Cross Validator

cv_fm = CrossValidator(estimator=pipeline_fm, evaluator=fm_evaluator,
estimatorParamMaps=paramGrid_CV_fm)

```
1  cv_fm = CrossValidator(estimator=pipeline_fm, evaluator=fm_evaluator, estimatorParamMaps=paramGrid_CV_fm)
```

```
Command took 0.02 seconds -- by avalasa@calstatela.edu at 5/14/2023, 12:39:09 AM on QS_Anusha
```

9)Training Cross Validator FM
model_fm = cv_fm.fit(train)

```
1  model_fm = cv_fm.fit(train)
```

▸ (50) Spark Jobs

```
Command took 3.88 minutes -- by avalasa@calstatela.edu at 5/14/2023, 12:39:12 AM on QS_Anusha
```

10)Predicting the Price
prediction_fm = model_fm.transform(test)
predicted_fm = prediction_fm.select("features", "prediction", "label")

predicted_fm.show(10)

Predict price with actual price using Lr

```
prediction_fm = model_fm.transform(test)
predicted_fm = prediction_fm.select("features", "prediction", "label")
predicted_fm.show(10)
```

▸ (1) Spark Jobs

```
+--------------------+-------------------+-------+
|            features|         prediction|  label|
+--------------------+-------------------+-------+
|[2002.0,14.0,1712...|-298262.64678397554| 8950.0|
|[2003.0,18.0,1482...| -84135.67164692322| 2800.0|
|[2004.0,16.0,1434...|-107169.90338720895| 4889.0|
|[2006.0,17.0,1276...| -63605.10398977945| 4488.0|
|[2007.0,17.0,2449...| -358831.7283427908| 4795.0|
|[2008.0,13.0,846....| 489369.83487343596|34900.0|
|[2008.0,15.0,310....| 443561.60182840435|48999.0|
|[2008.0,22.0,1058...|-22214.505959814876| 5995.0|
|[2011.0,18.0,8840...| -64012.23207307626|12921.0|
|[2011.0,18.0,3699...| 45026.773507431964|18950.0|
+--------------------+-------------------+-------+
only showing top 10 rows
```

Command took 0.98 seconds -- by avalasa@calstatela.edu at 5/14/2023, 12:39:17 AM on QS_Anusha

10)RMSE AND R2 For FM Cross Validation

fm_cv_evaluator_rmse = RegressionEvaluator(labelCol="label", predictionCol="prediction", metricName="rmse")
print("RMSE for FM_CV: %.2f" % fm_cv_evaluator_rmse.evaluate(prediction_fm))
fm_cv_evaluator_r2 = RegressionEvaluator(predictionCol="prediction", \
labelCol="label",metricName="r2")
print("R2 for FM_CV: %.2f" % fm_cv_evaluator_r2.evaluate(prediction_fm))

```
fm_cv_evaluator_rmse = RegressionEvaluator(labelCol="label", predictionCol="prediction", metricName="rmse")
print("RMSE for FM_CV: %.2f" % fm_cv_evaluator_rmse.evaluate(prediction_fm))

fm_cv_evaluator_r2 = RegressionEvaluator(predictionCol="prediction", \
                labelCol="label",metricName="r2")
print("R2 for FM_CV: %.2f" % fm_cv_evaluator_r2.evaluate(prediction_fm))

```

▸ (2) Spark Jobs

```
RMSE for FM_CV: 95536.98
R2 for FM_CV: -48.92
```

Command took 4.08 seconds -- by avalasa@calstatela.edu at 5/14/2023, 12:39:24 AM on QS_Anusha

RMSE for FM_CV: 95536.98
R2 for FM_CV: -48.92
Training time: 3.88 minutes

FM_TrainValidationSplit: -

1)Creating a Paramgrid builder with parameters

paramGrid_TV_fm = ParamGridBuilder()\
  .addGrid(fm.factorSize, [8]) \
  .addGrid(fm.stepSize, [0.01, 0.1]) \
  .addGrid(fm.regParam, [0.01, 0.1]) \
.build()

```
paramGrid_TV_fm = ParamGridBuilder()\
  .addGrid(fm.factorSize, [8]) \
  .addGrid(fm.stepSize, [0.01, 0.1]) \
  .addGrid(fm.regParam, [0.01, 0.1]) \
.build()
```

```
Command took 0.03 seconds -- by avalasa@calstatela.edu at 5/14/2023, 12:40:46 AM on QS_Anusha
```

2)Creating the Train Validation Split

fm_tv = TrainValidationSplit(estimator=pipeline_fm, evaluator=fm_evaluator, estimatorParamMaps=paramGrid_TV_fm, trainRatio=0.8)

```
fm_tv = TrainValidationSplit(estimator=pipeline_fm, evaluator=fm_evaluator, estimatorParamMaps=paramGrid_TV_fm, trainRatio=0.8)

```

```
Command took 0.02 seconds -- by avalasa@calstatela.edu at 5/14/2023, 12:40:49 AM on QS_Anusha
```

3)Training the model

model_tv_fm = fm_tv.fit(train)

```
model_tv_fm = fm_tv.fit(train)
```

```
▸ (50) Spark Jobs

Command took 2.63 minutes -- by avalasa@calstatela.edu at 5/14/2023, 12:40:52 AM on QS_Anusha
```

4)Predicting the Price

prediction_tv_fm = model_tv_fm.transform(test)
predicted_tv_fm = prediction_tv_fm.select("features", "prediction", "label")
predicted_tv_fm.show(10)

```
prediction_tv_fm = model_tv_fm.transform(test)
predicted_tv_fm = prediction_tv_fm.select("features", "prediction", "label")
predicted_tv_fm.show(10)
```

▸ (1) Spark Jobs

```
+--------------------+------------------+-------+
|            features|        prediction|  label|
+--------------------+------------------+-------+
|[2002.0,14.0,1712...| 51432.75495778801| 8950.0|
|[2003.0,18.0,1482...|  7414.746483148912| 2800.0|
|[2004.0,16.0,1434...|11052.810717561719| 4889.0|
|[2006.0,17.0,1276...|  6903.443750543498| 4488.0|
|[2007.0,17.0,2449...|  43637.93900190329| 4795.0|
|[2008.0,13.0,846....|28289.174439077844|34900.0|
|[2008.0,15.0,310....|25074.657019378872|48999.0|
|[2008.0,22.0,1058...|  2827.426917106175| 5995.0|
|[2011.0,18.0,8840...|23677.419623396643|12921.0|
|[2011.0,18.0,3699...|13887.143528103643|18950.0|
+--------------------+------------------+-------+
only showing top 10 rows
```

Command took 0.99 seconds -- by avalasa@calstatela.edu at 5/14/2023, 12:40:55 AM on QS_Anusha

5)R2 and RMSE for FM TV: -

fm_tv_evaluator_rmse = RegressionEvaluator(labelCol="label", predictionCol="prediction", metricName="rmse")
print("RMSE for FM_TV: %.2f" % fm_tv_evaluator_rmse.evaluate(prediction_tv_fm))
fm_tv_evaluator_r2 = RegressionEvaluator(predictionCol="prediction", \
labelCol="label",metricName="r2")
print("R2 for FM_CV: %.2f" % fm_tv_evaluator_r2.evaluate(prediction_tv_fm))

```
fm_tv_evaluator_rmse = RegressionEvaluator(labelCol="label", predictionCol="prediction", metricName="rmse")
print("RMSE for FM_TV: %.2f" % fm_tv_evaluator_rmse.evaluate(prediction_tv_fm))

fm_tv_evaluator_r2 = RegressionEvaluator(predictionCol="prediction", \
                labelCol="label",metricName="r2")
print("R2 for FM_CV: %.2f" % fm_tv_evaluator_r2.evaluate(prediction_tv_fm))
```

▸ (2) Spark Jobs

```
RMSE for FM_TV: 19841.19
R2 for FM_CV: -1.15
```

Command took 3.67 seconds -- by avalasa@calstatela.edu at 5/14/2023, 12:41:03 AM on QS_Anusha

RMSE for TV_RF: 19841.19
R2 for TV_RF: -1.15
Random Forest CV data training time: 2.63 minutes

Algorithm 5: Logistic Regression

1)Creating the Schema

```
flightSchema = StructType([
  StructField("year", IntegerType(), False),
  StructField("price", IntegerType(), False),
  StructField("mileage", StringType(), False),
  StructField("owner_count", IntegerType(), False),
  StructField("seller_rating", IntegerType(), False),
  StructField("city_fuel_economy", IntegerType(), False),
```

StructField("highway_fuel_economy", IntegerType(), False),
StructField("daysonmarket", IntegerType(), False),
StructField("dealer_zip", IntegerType(), False),
StructField("engine_displacement", IntegerType(), False),
StructField("listing_id", IntegerType(), False),
StructField("savings_amount", IntegerType(), False),
StructField("seller_rating", IntegerType(), False),
StructField("sp_id", IntegerType(), False),

])

```
flightSchema = StructType([
  StructField("year", IntegerType(), False),
  StructField("price", IntegerType(), False),
  StructField("mileage", StringType(), False),
  StructField("owner_count", IntegerType(), False),
  StructField("seller_rating", IntegerType(), False),
  StructField("city_fuel_economy", IntegerType(), False),
  StructField("highway_fuel_economy", IntegerType(), False),
  StructField("daysonmarket", IntegerType(), False),
  StructField("dealer_zip", IntegerType(), False),
  StructField("engine_displacement", IntegerType(), False),
  StructField("listing_id", IntegerType(), False),
  StructField("savings_amount", IntegerType(), False),
  StructField("seller_rating", IntegerType(), False),
  StructField("sp_id", IntegerType(), False),

])
```

```
Command took 0.02 seconds -- by avalasa@calstatela.edu at 5/14/2023, 12:03:54 AM on QS_Anusha
```

2)Selecting the input columns which are different from the above algorithms as here predicting yes or no

(df2.select("year", "city_fuel_economy", col("price").alias("label"), "mileage", "owner_count", "latitude", "highway_fuel_economy", "daysonmarket", "dealer_zip", "engine_displacement", "listing_id", "savings_amount", "seller_rating","sp_id").limit(10)).show()

```
(df2.select("year", "city_fuel_economy", col("price").alias("label"), "mileage", "owner_count", "latitude", "highway_fuel_economy", "daysonmarket",
"dealer_zip", "engine_displacement", "listing_id", "savings_amount", "seller_rating","sp_id").limit(10)).show()
```

▸ (1) Spark Jobs

```
+----+----------------+-------+--------+-----------+--------+--------------------+------------+----------+------------------+----------+--------------+---
-------+------+
year|city_fuel_economy|  label| mileage|owner_count|latitude|highway_fuel_economy|daysonmarket|dealer_zip|engine_displacement|listing_id|savings_amount|sel
_rating| sp_id|
+----+----------------+-------+--------+-----------+--------+--------------------+------------+----------+------------------+----------+--------------+---
-------+------+
2019|            19.0|55994.0|    14.0|       null|  30.222|                26.0|         716|     32244|              3500| 219569238|             0|
|371785|
null|            25.0|   null|    null|       null|    null|                null|          27|     32256|              null|      null|          null|
ull|  null|
null|            12.0|   null|    null|       null|    null|                null|          48|     50126|              null|      null|          null|
ull|  null|
2018|            22.0|35900.0| 19559.0|          1| 30.2869|                28.0|         100|     32246|              2000| 273093755|          2081|
|283437|
2006|            20.0| 1999.0|119181.0|       null| 30.2864|                28.0|           8|     32216|              2400| 281035985|          2141|
|289407|
2020|            23.0|30700.0|     1.0|       null| 30.1751|                31.0|          27|     32256|              2500| 279263821|             0|
| 63887|
null|            25.0|   null|    null|       null|    null|                null|          27|     32256|              null|      null|          null|
ull|  null|
2018|            25.0|13499.0| 32828.0|          1|  30.222|                30.0|          62|     32244|              2000| 276228768|           883|
```

ommand took 0.48 seconds -- by avalasa@calstatela.edu at 5/14/2023, 12:03:54 AM on QS_Anusha

df2 = df2.filter(col("price") < 100000)
#display(df2.limit(10))
(df2.limit(10)).show()
df23 = df2.select("year", "city_fuel_economy", ((col("price") > 10000).cast("Double").alias("label")), "mileage", "owner_count", "highway_fuel_economy", "daysonmarket", "dealer_zip", "engine_displacement", "listing_id", "savings_amount", "seller_rating","sp_id")

```
df2 = df2.filter(col("price") < 100000)
#display(df2.limit(10))
(df2.limit(10)).show()
df23 = df2.select("year", "city_fuel_economy", ((col("price") > 10000).cast("Double").alias("label")), "mileage", "owner_count", "highway_fuel_economy",
"daysonmarket", "dealer_zip", "engine_displacement", "listing_id", "savings_amount", "seller_rating","sp_id")
```

3)Dropping the null values

df23.na.drop(subset=["label", "year", "city_fuel_economy", "mileage","owner_count", "highway_fuel_economy", "daysonmarket", "dealer_zip", "engine_displacement", "listing_id", "savings_amount", "seller_rating","sp_id"])
df23.show()

```
1  evaluator = BinaryClassificationEvaluator(labelCol="trueLabel", rawPredictionCol="rawPrediction", metricName="areaUnderROC"
2  auc_cv = evaluator.evaluate(prediction_cv_logr)
3  print("AUC = ", auc_cv)
```

▸ (4) Spark Jobs

AUC = 0.9795442908346135

Command took 2.97 seconds -- by avalasa@calstatela.edu at 5/14/2023, 12:03:55 AM on QS_Anusha

```
df23.na.drop(subset=["label", "year", "city_fuel_economy", "mileage","owner_count", "highway_fuel_economy", "daysonmarket", "dealer_zip",
"engine_displacement", "listing_id", "savings_amount", "seller_rating","sp_id"])
df23.show()
```

(1) Spark Jobs

```
--+-----------------+-----+---------+-----------+-------------------+------------+----------+-------------------+----------+--------------+-------------+--
--+
ar|city_fuel_economy|label| mileage|owner_count|highway_fuel_economy|daysonmarket|dealer_zip|engine_displacement|listing_id|savings_amount|seller_rating| s
d|
--+-----------------+-----+---------+-----------+-------------------+------------+----------+-------------------+----------+--------------+-------------+--
--+
019|             19.0|  1.0|    14.0|       null|               26.0|         716|     32244|               3500| 219569238|             0|            4|37
35|
018|             22.0|  1.0| 19559.0|          1|               28.0|         100|     32246|               2000| 273093755|          2081|            4|28
17|
006|             20.0|  0.0|119181.0|       null|               28.0|           8|     32216|               2400| 281035985|          2141|            4|28
17|
020|             23.0|  1.0|     1.0|       null|               31.0|          27|     32256|               2500| 279263821|             0|            4| 6
17|
018|             25.0|  1.0| 32828.0|          1|               30.0|          62|     32244|               2000| 276228768|           883|            4|37
35|
020|             18.0|  1.0|    13.0|       null|               25.0|         209|     32244|               3600| 265854322|             0|            4|29
39|
020|             20.0|  1.0|     7.0|       null|               27.0|         230|     32244|               3500| 264019553|             0|            4|37
35|
015|             24.0|  1.0| 65295.0|          1|               30.0|          28|     32244|               1600| 279176979|           646|            4|29
mand took 0.67 seconds -- by avalasa@calstatela.edu at 5/14/2023, 12:03:54 AM on QS_Anusha
```

4) Splitting the data

splits = df23.randomSplit([0.7, 0.3])
train = splits[0]
test = splits[1].withColumnRenamed("label", "trueLabel")
train_rows = train.count()
test_rows = test.count()
print("Training Rows:", train_rows, " Testing Rows:", test_rows)

```
1  splits = df23.randomSplit([0.7, 0.3])
2  train = splits[0]
3  test = splits[1].withColumnRenamed("label", "trueLabel")
4  train_rows = train.count()
5  test_rows = test.count()
6  print("Training Rows:", train_rows, " Testing Rows:", test_rows)
```

▸ (4) Spark Jobs

```
Training Rows: 2971  Testing Rows: 1318

Command took 4.89 seconds -- by avalasa@calstatela.edu at 5/14/2023, 12:03:54 AM on QS_Anusha
```

5) Creating the vector assembler and adding all the input features like vector indexer and min max scaler to the vector assembler. Creating the logistic regression and pipeline

catVect = VectorAssembler(inputCols = ["mileage","year", "owner_count",
"city_fuel_economy", "highway_fuel_economy", "daysonmarket", "dealer_zip",
"engine_displacement", "listing_id", "savings_amount", "seller_rating","sp_id"],
outputCol="catFeatures").setHandleInvalid("skip")
catIdx = VectorIndexer(inputCol = catVect.getOutputCol(), outputCol =
"idxCatFeatures").setHandleInvalid("skip")

```
# the following columns are categorical number such as ID so that it should be Category features
catVect = VectorAssembler(inputCols = ["mileage","year", "owner_count", "city_fuel_economy", "highway_fuel_economy", "daysonmarket", "dealer_zip",
"engine_displacement", "listing_id", "savings_amount", "seller_rating","sp_id"], outputCol="catFeatures").setHandleInvalid("skip")
catIdx = VectorIndexer(inputCol = catVect.getOutputCol(), outputCol = "idxCatFeatures").setHandleInvalid("skip")
```

Command took 0.17 seconds -- by avalasa@calstatela.edu at 5/14/2023, 12:03:54 AM on QS_Anusha

numVect = VectorAssembler(inputCols = ["mileage"], outputCol="numFeatures")
# number vector is normalized
minMax = MinMaxScaler(inputCol = numVect.getOutputCol(), outputCol="normFeatures")
featVect = VectorAssembler(inputCols=["idxCatFeatures", "normFeatures"],
outputCol="features") #="features1")
# number vector is normalized: this changes the accuracy of precision a little
# minMax2 = MinMaxScaler(inputCol = featVect.getOutputCol(), outputCol="features")
lr = LogisticRegression(labelCol="label",featuresCol="features",maxIter=10,regParam=0.3)

# Pipeline process the series of transformation above, which is 7 transformation
pipeline = Pipeline(stages=[catVect, catIdx, numVect, minMax, featVect, lr]) #minMax2, lr])

```
# number is meaningful so that it should be number features
numVect = VectorAssembler(inputCols = ["mileage"], outputCol="numFeatures")
# number vector is normalized
minMax = MinMaxScaler(inputCol = numVect.getOutputCol(), outputCol="normFeatures")

featVect = VectorAssembler(inputCols=["idxCatFeatures", "normFeatures"], outputCol="features")  #="features1")
# number vector is normalized: this changes the accuracy of precision a little
# minMax2 = MinMaxScaler(inputCol = featVect.getOutputCol(), outputCol="features")

#dt = DecisionTreeClassifier(labelCol="label", featuresCol="features",maxIter=10,regParam=0.3)
lr = LogisticRegression(labelCol="label",featuresCol="features",maxIter=10,regParam=0.3)

# Pipeline process the series of transformation above, which is 7 transformation
pipeline = Pipeline(stages=[catVect, catIdx, numVect, minMax, featVect, lr]) #minMax2, lr])
```

Command took 0.17 seconds -- by avalasa@calstatela.edu at 5/14/2023, 12:03:54 AM on QS_Anusha

6) Creating the Param grid with different parameters

paramGrid = (ParamGridBuilder() \
.addGrid(lr.regParam, [0.01, 0.5]) \
.addGrid(lr.elasticNetParam, [0.0, 0.5]) \
.addGrid(lr.maxIter, [1, 5]) \
.build())
#paramGrid = ParamGridBuilder().addGrid(lr.regParam, [0.3, 0.1, 0.01]).addGrid(lr.maxIter, [10, 5]).addGrid(lr.threshold, [0.35, 0.30]).build()

```
paramGrid = (ParamGridBuilder() \
             .addGrid(lr.regParam, [0.01, 0.5]) \
             .addGrid(lr.elasticNetParam, [0.0, 0.5]) \
             .addGrid(lr.maxIter, [1, 5]) \
             .build())
#paramGrid = ParamGridBuilder().addGrid(lr.regParam, [0.3, 0.1, 0.01]).addGrid(lr.maxIter, [10, 5]).addGrid(lr.threshold, [0.35, 0.30]).build()
```

Command took 0.02 seconds -- by avalasa@calstatela.edu at 5/14/2023, 12:03:55 AM on QS_Anusha

7) Creating the Evaluator

cv_logr = CrossValidator(estimator=pipeline, evaluator=BinaryClassificationEvaluator(), estimatorParamMaps=paramGrid)

model_Cv_logr = cv_logr.fit(train)

```
cv_logr = CrossValidator(estimator=pipeline, evaluator=BinaryClassificationEvaluator(), estimatorParamMaps=paramGrid)
model_Cv_logr = cv_logr.fit(train)
```

▸ (58) Spark Jobs

Command took 2.89 minutes -- by avalasa@calstatela.edu at 5/14/2023, 12:03:55 AM on QS_Anusha

8) Predicting the data with input columns

prediction_cv_logr = model_Cv_logr.transform(test)
predicted_cv_logr = prediction_cv_logr.select("features", "prediction", "probability", "trueLabel")
predicted_cv_logr.show(100, truncate=False)

```
prediction_cv_logr = model_Cv_logr.transform(test)
predicted_cv_logr = prediction_cv_logr.select("features", "prediction", "probability", "trueLabel")

predicted_cv_logr.show(100, truncate=False)
```

Python

▸ (2) Spark Jobs

```
+----------------------------------------------------------------------------------------------------------+----------+----------------------------------------
-+---------+
|features                                                                                                  |prediction|probability
|trueLabel|
+----------------------------------------------------------------------------------------------------------+----------+----------------------------------------
-+---------+
|[164524.0,1997.0,1.0,17.0,27.0,84.0,50613.0,3800.0,2.74364588E8,0.0,1.0,284048.0,0.5191587384231868]    |0.0       |[0.9922863937205131,0.007713606279486918]
|0.0      |
|[148234.0,2003.0,2.0,18.0,25.0,77.0,50613.0,3000.0,2.74961437E8,0.0,1.0,284048.0,0.46775532099525097]   |0.0       |[0.9670723363322165,0.03292766366778355]
|0.0      |
|[105733.0,2004.0,1.0,15.0,21.0,12.0,32216.0,3500.0,2.80686438E8,1458.0,1.0,289407.0,0.33364257427304717]|0.0       |[0.6494344318173253,0.3505655681826747]
|0.0      |
|[127669.0,2006.0,1.0,17.0,22.0,10.0,32210.0,3000.0,2.80854198E8,0.0,0.0,369498.0,0.4028620564522491]    |0.0       |[0.8623852332264639,0.1376147667735361 2]
|0.0      |
|[157445.0,2007.0,1.0,12.0,18.0,195.0,51503.0,6200.0,2.67212199E8,149.0,2.0,299674.0,0.49682081380855464]|1.0       |[0.36153477546928825,0.6384652245307118]
|1.0      |
|[244956.0,2007.0,1.0,17.0,24.0,28.0,51463.0,3500.0,2.79236584E8,998.0,1.0,137510.0,0.772963506413594]   |0.0       |[0.9481569189758158,0.05184308102418422]
|0.0      |
|[846.0,2008.0,1.0,13.0,18.0,89.0,32246.0,6100.0,2.73991981E8,5794.0,1.0,283437.0,0.002669569744876225]  |1.0       |[0.0015566987071045783,0.998443301292895
4]|1.0      |
|[310.0,2008.0,1.0,15.0,24.0,333.0,32246.0,7000.0,2.55096079E8,4253.0,1.0,283437.0,9.782111358293496E-4] |1.0       |[0.002055485802576521,0.9979445141974235]
```

Command took 1.17 seconds -- by avalasa@calstatela.edu at 5/14/2023, 12:03:55 AM on QS_Anusha

9)Predicting the data which has True Positives, False Positives, True Negatives, False negatives

tp = float(predicted_cv_logr.filter("prediction == 1.0 AND truelabel == 1").count())
fp = float(predicted_cv_logr.filter("prediction == 1.0 AND truelabel == 0").count())
tn = float(predicted_cv_logr.filter("prediction == 0.0 AND truelabel == 0").count())
fn = float(predicted_cv_logr.filter("prediction == 0.0 AND truelabel == 1").count())
metrics = spark.createDataFrame([
("TP", tp),
("FP", fp),
("TN", tn),
("FN", fn),
("Precision", tp / (tp + fp)),
("Recall", tp / (tp + fn))],["metric", "value"])

```
tp = float(predicted_cv_logr.filter("prediction == 1.0 AND truelabel == 1").count())
fp = float(predicted_cv_logr.filter("prediction == 1.0 AND truelabel == 0").count())
tn = float(predicted_cv_logr.filter("prediction == 0.0 AND truelabel == 0").count())
fn = float(predicted_cv_logr.filter("prediction == 0.0 AND truelabel == 1").count())
metrics = spark.createDataFrame([
 ("TP", tp),
 ("FP", fp),
 ("TN", tn),
 ("FN", fn),
 ("Precision", tp / (tp + fp)),
 ("Recall", tp / (tp + fn))],["metric", "value"])
```

▸ (8) Spark Jobs

Command took 10.43 seconds -- by avalasa@calstatela.edu at 5/14/2023, 12:03:55 AM on QS_Anusha

10) To see the metrics
metrics.show()

```
metrics.show()
```

▸ (3) Spark Jobs

```
+---------+------------------+
|   metric|             value|
+---------+------------------+
|       TP|             417.0|
|       FP|              25.0|
|       TN|              68.0|
|       FN|               3.0|
|Precision|0.9434389140271493|
|   Recall|0.9928571428571429|
+---------+------------------+
```

Command took 1.03 seconds -- by avalasa@calstatela.edu at 5/14/2023, 12:03:55 AM on QS_Anusha

11) Predicting whether user can buy this or not
prediction_cv_logr.select("rawPrediction", "probability", "prediction", "trueLabel").show(100, truncate=False)

```
prediction_cv_logr.select("rawPrediction", "probability", "prediction", "trueLabel").show(100, truncate=False)
```

(2) Spark Jobs

```
+----------------------------------------+----------------------------------------+----------+---------+
|rawPrediction                           |probability                             |prediction|trueLabel|
+----------------------------------------+----------------------------------------+----------+---------+
|[4.857025950216553,-4.857025950216553]  |[0.9922863937205131,0.007713606279486918]|0.0       |0.0      |
|[3.3799601521205886,-3.3799601521205886]|[0.9670723363322165,0.03292766366778355] |0.0       |0.0      |
|[0.6165541198317896,-0.6165541198317896]|[0.6494344318173253,0.3505655681826747]  |0.0       |0.0      |
|[1.8352438407166574,-1.8352438407166574]|[0.8623852332264639,0.13761476677353612] |0.0       |0.0      |
|[-0.5687089749936831,0.5687089749936831]|[0.36153477546928825,0.6384652245307118] |1.0       |1.0      |
|[2.90629853128803,-2.90629853128803]    |[0.9481569189758158,0.05184308102418422] |0.0       |0.0      |
|[-6.463630001837373,6.463630001837373]  |[0.0015566987071045783,0.9984433012928954]|1.0       |1.0      |
|[-6.185185457478724,6.185185457478724]  |[0.002055485802576521,0.9979445141974235]|1.0       |1.0      |
|[-0.3153824534249452,0.3153824534249452]|[0.4218014896762935,0.5781985103237065]  |1.0       |1.0      |
|[-1.8460595210315205,1.8460595210315205]|[0.13633621856032985,0.8636637814396702] |1.0       |0.0      |
|[-1.089506368113689,1.089506368113689]  |[0.25171124389232336,0.7482887561076766] |1.0       |1.0      |
|[-2.405220751737261,2.405220751737261]  |[0.08277545343864397,0.917224546561356]  |1.0       |1.0      |
|[-0.9724801372185539,0.9724801372185539]|[0.2743864343593519,0.7256135656406482]  |1.0       |0.0      |
|[-3.3326201115004324,3.3326201115004324]|[0.034468924392389494,0.9655310756076105]|1.0       |1.0      |
|[-2.887997095837136,2.887997095837136]  |[0.05275010901433451,0.9472498909856655] |1.0       |1.0      |
|[-0.8512167574778005,0.8512167574778005]|[0.2991776771830927,0.7008223228169073]  |1.0       |0.0      |
|[-1.698871385819757,1.698871385819757]  |[0.1546127260720876,0.8453872739279125]  |1.0       |1.0      |
|[-1.2325121939003338,1.2325121939003338]|[0.22574203552300406,0.7742579644769959] |1.0       |1.0      |
```

Command took 1.25 seconds -- by avalasa@calstatela.edu at 5/14/2023, 12:03:55 AM on QS_Anusha

12) Finding the AUC

evaluator = BinaryClassificationEvaluator(labelCol="trueLabel", rawPredictionCol="rawPrediction", metricName="areaUnderROC")
auc_cv = evaluator.evaluate(prediction_cv_logr)
print("AUC = ", auc_cv)

```
evaluator = BinaryClassificationEvaluator(labelCol="trueLabel", rawPredictionCol="rawPrediction", metricName="areaUnderROC")
auc_cv = evaluator.evaluate(prediction_cv_logr)
print("AUC = ", auc_cv)
```

(4) Spark Jobs

```
AUC =  0.9795442908346135
```

Command took 2.97 seconds -- by avalasa@calstatela.edu at 5/14/2023, 12:03:55 AM on QS_Anusha

AUC = 0.97
Training time: 3 Minutes

TrainValidationSplit For Logistic Regression: -

1)Creating the Train Validation for logistic Regression

cv = TrainValidationSplit(estimator=pipeline, evaluator=BinaryClassificationEvaluator(), estimatorParamMaps=paramGrid, trainRatio=0.8)

model = cv.fit(train)

```
cv = TrainValidationSplit(estimator=pipeline, evaluator=BinaryClassificationEvaluator(), estimatorParamMaps=paramGrid, trainRatio=0.8)

model = cv.fit(train)
```

(49) Spark Jobs

Command took 57.98 seconds -- by avalasa@calstatela.edu at 5/14/2023, 12:03:55 AM on QS_Anusha

2)Predicting the probability

prediction = model.transform(test)
predicted = prediction.select("features", "prediction", "probability", "trueLabel")
predicted.show(100, truncate=False

```
prediction = model.transform(test)
predicted = prediction.select("features", "prediction", "probability", "trueLabel")

predicted.show(100, truncate=False)
```

(2) Spark Jobs

```
+----------------------------------------------------------------------------------------------------+----------+----------------------------------------
-+---------+
|features                                                                                            |prediction|probability
|trueLabel|
+----------------------------------------------------------------------------------------------------+----------+----------------------------------------
-+---------+
|[164524.0,1997.0,1.0,17.0,27.0,84.0,50613.0,3800.0,2.74364588E8,0.0,1.0,284048.0,0.5191587384231868]    |0.0       |[0.9869464838835124,0.01305351611648764]
|0.0      |
|[148234.0,2003.0,2.0,18.0,25.0,77.0,50613.0,3000.0,2.74961437E8,0.0,1.0,284048.0,0.46775532099525097]   |0.0       |[0.9556442943448961,0.04435570565510394]
|0.0      |
|[105733.0,2004.0,1.0,15.0,21.0,12.0,32216.0,3500.0,2.80686438E8,1458.0,1.0,289407.0,0.33364257427304717]|0.0       |[0.6644182815204104,0.3355817184795896]
|0.0      |
|[127669.0,2006.0,1.0,17.0,22.0,10.0,32210.0,3000.0,2.80854198E8,0.0,0.0,369498.0,0.4028620564522491]    |0.0       |[0.822335312156212,0.17766468784378797]
|0.0      |
|[157445.0,2007.0,1.0,12.0,18.0,195.0,51503.0,6200.0,2.67212199E8,149.0,2.0,299674.0,0.49682081380855464]|1.0       |[0.42982606085218644,0.5701739391478136]
|1.0      |
|[244956.0,2007.0,1.0,17.0,24.0,28.0,51463.0,3500.0,2.79236584E8,998.0,1.0,137510.0,0.772963506413594]   |0.0       |[0.9316623887535664,0.06833761124643356]
|0.0      |
|[846.0,2008.0,1.0,13.0,18.0,89.0,32246.0,6100.0,2.73991981E8,5794.0,1.0,283437.0,0.002669569744876225]  |1.0       |[0.004936138837280451,0.9950638611627195
|1.0      |
|[310.0,2008.0,1.0,15.0,24.0,333.0,32246.0,7000.0,2.55096079E8,4253.0,1.0,283437.0,9.782111358293496E-4] |1.0       |[0.005964180974322603,0.9940358190256774
```

Command took 1.61 seconds -- by avalasa@calstatela.edu at 5/14/2023, 12:03:55 AM on QS_Anusha

3) Predicting the data which has True Positives, False Positives, True Negatives, False negatives

tp = float(predicted.filter("prediction == 1.0 AND truelabel == 1").count())
fp = float(predicted.filter("prediction == 1.0 AND truelabel == 0").count())
tn = float(predicted.filter("prediction == 0.0 AND truelabel == 0").count())
fn = float(predicted.filter("prediction == 0.0 AND truelabel == 1").count())
metrics = spark.createDataFrame([
 ("TP", tp),
 ("FP", fp),
 ("TN", tn),
 ("FN", fn),
 ("Precision", tp / (tp + fp)),
 ("Recall", tp / (tp + fn))],["metric", "value"])

```
tp = float(predicted.filter("prediction == 1.0 AND truelabel == 1").count())
fp = float(predicted.filter("prediction == 1.0 AND truelabel == 0").count())
tn = float(predicted.filter("prediction == 0.0 AND truelabel == 0").count())
fn = float(predicted.filter("prediction == 0.0 AND truelabel == 1").count())
metrics = spark.createDataFrame([
 ("TP", tp),
 ("FP", fp),
 ("TN", tn),
 ("FN", fn),
 ("Precision", tp / (tp + fp)),
 ("Recall", tp / (tp + fn))],["metric", "value"])
```

▸ (8) Spark Jobs

Command took 8.71 seconds -- by avalasa@calstatela.edu at 5/14/2023, 12:03:55 AM on QS_Anusha

4) To see the metrics

metrics.show()

```
metrics.show()
```

▸ (3) Spark Jobs

```
+---------+------------------+
|   metric|             value|
+---------+------------------+
|       TP|             415.0|
|       FP|              28.0|
|       TN|              65.0|
|       FN|               5.0|
|Precision|0.9367945823927766|
|   Recall|0.9880952380952381|
+---------+------------------+
```

Command took 0.31 seconds -- by avalasa@calstatela.edu at 5/14/2023, 12:03:55 AM on QS_Anusha

5) Predicting the probability of wether the user can buy this car or not

prediction.select("rawPrediction", "probability", "prediction", "trueLabel").show(100, truncate=False)

```
prediction.select("rawPrediction", "probability", "prediction", "trueLabel").show(100, truncate=False)
```

▸ (2) Spark Jobs

```
+----------------------------------------+----------------------------------------+----------+---------+
|rawPrediction                           |probability                             |prediction|trueLabel|
+----------------------------------------+----------------------------------------+----------+---------+
|[4.325558285282682,-4.325558285282682]  |[0.9869464838835124,0.01305351611648764]|0.0       |0.0      |
|[3.070144415665254,-3.070144415665254]  |[0.9556442943448961,0.04435570565510394]|0.0       |0.0      |
|[0.6830463942228562,-0.6830463942228562]|[0.6644182815204104,0.3355817184795896] |0.0       |0.0      |
|[1.532250236613777,-1.532250236613777]  |[0.822335312156212,0.17766468784378797] |0.0       |0.0      |
|[-0.28256085347209137,0.28256085347209137]|[0.42982606085218644,0.5701739391478136]|1.0       |1.0      |
|[2.6125102131156837,-2.6125102131156837]|[0.9316623887535664,0.06833761124643356]|0.0       |0.0      |
|[-5.3062235034766445,5.3062235034766445]|[0.004936138837280451,0.9950638611627195]|1.0       |1.0      |
|[-5.116001500348489,5.116001500348489]  |[0.005964180974322603,0.9940358190256774]|1.0       |1.0      |
|[0.14341008660051102,-0.14341008660051102]|[0.5357912011615971,0.46420879883840294]|0.0       |1.0      |
|[-1.4200847465767765,1.4200847465767765]|[0.19464829834575179,0.8053517016542482]|1.0       |0.0      |
|[-0.7021297495660974,0.7021297495660974]|[0.3313402041924037,0.6686597958075964] |1.0       |1.0      |
|[-2.1214885161568873,2.1214885161568873]|[0.10702572794240765,0.8929742720575924]|1.0       |1.0      |
|[-0.8501257775528757,0.8501257775528757]|[0.2994064734795176,0.7005935265204823] |1.0       |0.0      |
|[-2.561114241261521,2.561114241261521]  |[0.07168335984397359,0.9283166401560264]|1.0       |1.0      |
|[-2.657956371598175,2.657956371598175]  |[0.06550031294510526,0.9344996870548947]|1.0       |1.0      |
|[-0.8267573424991497,0.8267573424991497]|[0.30433114928680505,0.695668850713195] |1.0       |0.0      |
|[-1.5084747920378732,1.5084747920378732]|[0.18116493889672788,0.8188350611032721]|1.0       |1.0      |
|[-0.9371091209027327,0.9371091209027327]|[0.28148465638205883,0.7185153436179412]|1.0       |1.0      |
```

Command took 0.98 seconds -- by avalasa@calstatela.edu at 5/14/2023, 12:03:55 AM on QS_Anusha

6)Finding the AUC

evaluator = BinaryClassificationEvaluator(labelCol="trueLabel",
rawPredictionCol="rawPrediction", metricName="areaUnderROC")
auc = evaluator.evaluate(prediction)
print("AUC = ","{:.2f}".format(auc))

```
evaluator = BinaryClassificationEvaluator(labelCol="trueLabel", rawPredictionCol="rawPrediction", metricName="areaUnderROC")
auc = evaluator.evaluate(prediction)
print("AUC = ","{:.2f}".format(auc))
```

▸ (4) Spark Jobs

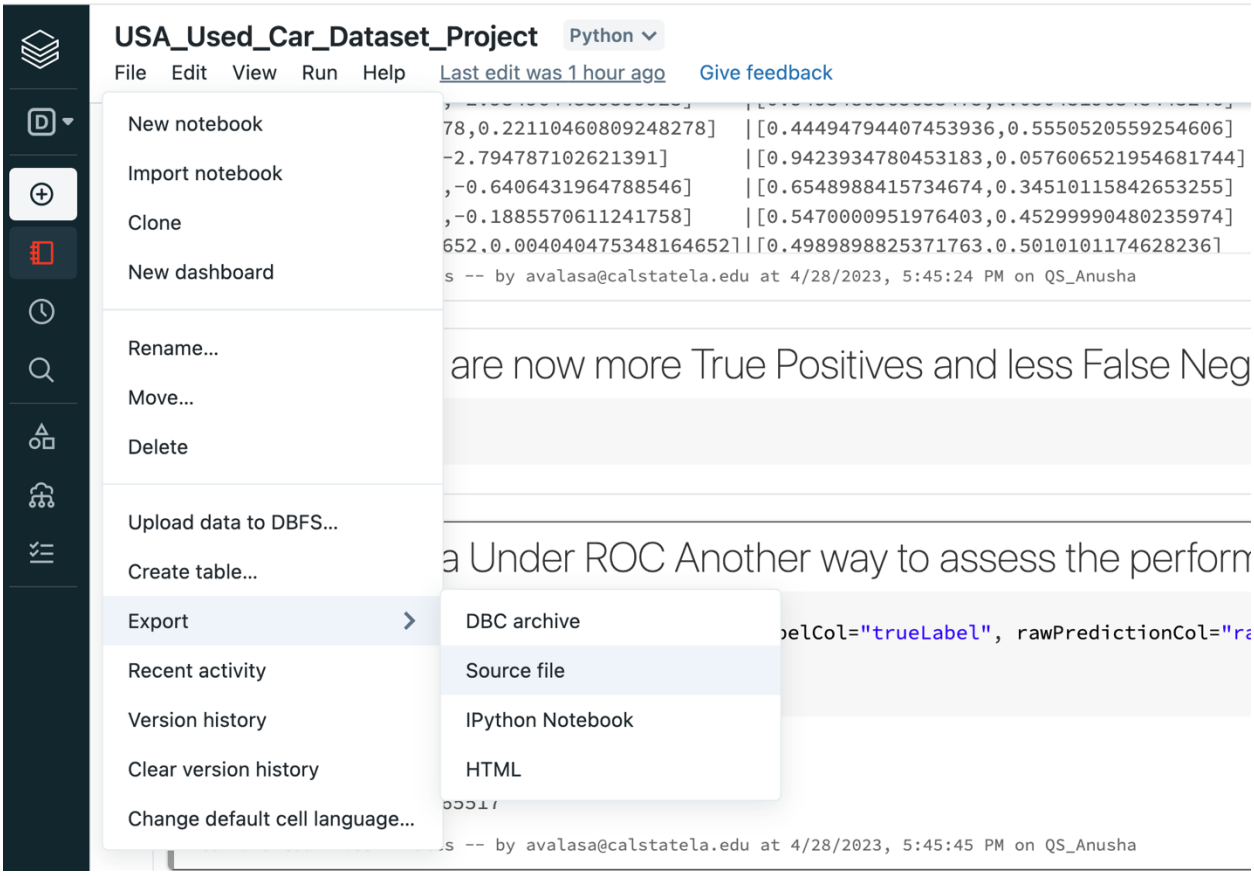
```
AUC =  0.98
```

Command took 2.91 seconds -- by avalasa@calstatela.edu at 5/14/2023, 12:03:56 AM on QS_Anusha

AUC = 0.98
Training time = 1minute

To run the whole 2GB used_cars_data_set.csv with Pyspark Submit.

1)Download .py file from the Databricks



PYSPARK SUBMIT

STEP 1: Connect to the Hadoop Spark cluster.

If you use Linux or Mac computers, you can use the Finder app to search the terminal with the keyword Term to find it. Then it will show you the terminal that you can use.

If you are working with a Windows laptop, you must use git bash and paste the ssh command to connect to the Hadoop Spark cluster.

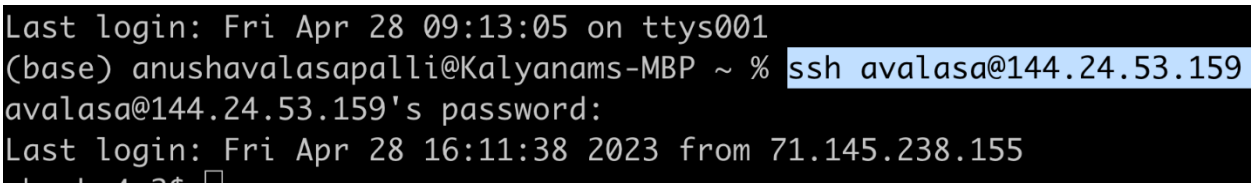
You can download and install Git Bash using this link: https://git-scm.com/downloads

You must use your Cal State LA email account name (avalasa) to log in to the cluster; in a terminal, you must type in the following ssh shell command to connect.

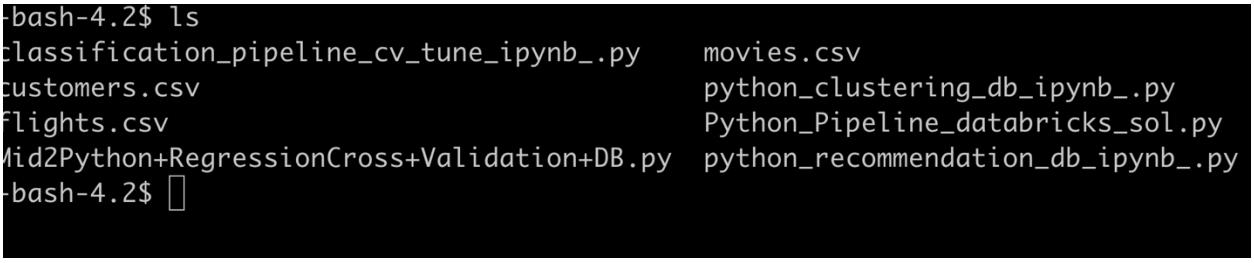
NOTE: Use your username "avalasa" to login into the Oracle big data server

ssh avalasa@144.24.53.159

To enter the password, type in your username as password and press enter.



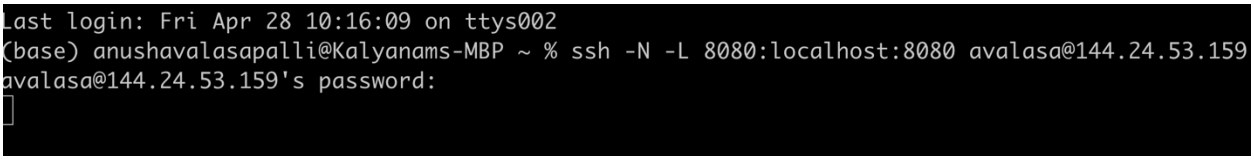
Now you will be logged in.



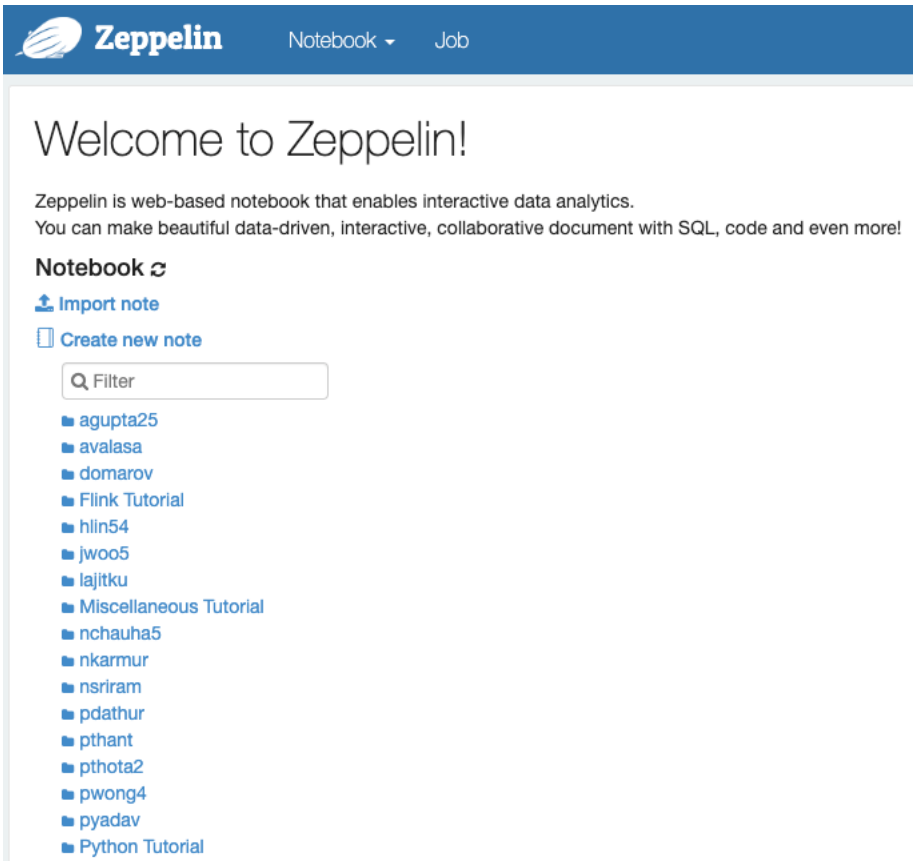
ZEPPELIN:

Suppose you use Zeppelin to write and run the code instead of data bricks. You have to open another window in the terminal or git bash, add the below command, and give the password the same as the user name.

ssh -N -L 8080:localhost:8080 avalasa@144.24.53.159



Click on this URL to connect to the port 8080 for Zeppelin http://localhost:8080/ to import the files.



STEP 2: transfer files from the local to Hadoop local server; open a new window and paste the below command.
scp downloads/US_Used_Cars_Data.csv avalasa@144.24.53.159:~

```
Last login: Sat Apr 29 09:50:25 on ttys005
(base) anushavalasapalli@Kalyanams-MBP ~ % scp downloads/US_Used_Cars_Data.csv avalasa@144.24.53.159:~
avalasa@144.24.53.159's password:
US_Used_Cars_Data.csv                                         100% 1921MB  35.8MB/s   00:53
(base) anushavalasapalli@Kalyanams-MBP ~ %
```

STEP 3: Use the below command to upload the US_Used_Cars_Data.csv file from the local server to HDFS.

hdfs dfs -put US_Used_Cars_Data.csv.

```
-bash-4.2$ hdfs dfs -put US_Used_Cars_Data.csv CIS-5560
-bash-4.2$ hdfs dfs -ls CIS-5560
Found 2 items
-rw-r--r--   3 avalasa hdfs      17789 2023-04-29 01:00 CIS-5560/USA_Used_Car_Dataset_Project_file.py
-rw-r--r--   3 avalasa hdfs 2014251349 2023-04-29 16:59 CIS-5560/US_Used_Cars_Data.csv
```

```
-bash-4.2$ hdfs dfs -put used_cars_data_set.csv
```

To check that the file is uploaded in the hdfs, use the hdfs dfs -ls command

Now we can see the US_Used_Cars_Data.csv file in the hdfs.

<u>STEP 4:</u> Create a directory, "CIS5560" to put the file to HDFS.

a. Run the following HDFS command to create the directory in HDFS.

hdfs dfs -mkdir CIS-5560

```
bash-4.2$ hdfs dfs -mkdir CIS-5560
```

b. Next, you can run the following shell command to put the file in the respective directory.
hdfs dfs -put US_Used_Cars_Data.csv CIS-5560/

```
-bash-4.2$ hdfs dfs -put US_Used_Cars_Data.csv   CIS-5560
```

To Check that the US_Used_Cars_Data.csv file is uploaded into the CIS-5560

```
-bash-4.2$ hdfs dfs -ls CIS-5560
Found 2 items
-rw-r--r--   3 avalasa hdfs 2014251349 2023-04-29 16:59 CIS-5560/US_Used_Cars_Data.csv
```

Now we have to upload the .py file to the hdfs which is the same as how we did for US_Used_Cars_Data.csv

scp downloads/ usa_used_car_dataset_project.py avalasa@144.24.53.159:~
hdfs dfs -put usa_used_car_dataset_project.py.
hdfs dfs -put usa_used_car_dataset_project.py CIS-5560/
hdfs dfs -ls CIS-5560

```
-bash-4.2$ hdfs dfs -ls CIS-5560
Found 2 items
-rw-r--r--   3 avalasa hdfs 2014251349 2023-04-29 16:59 CIS-5560/US_Used_Cars_Data.csv
-rw-r--r--   3 avalasa hdfs      17586 2023-04-29 18:10 CIS-5560/usa_used_car_dataset_project.py
```

To open the usa_used_car_dataset_project.py file in the hdfs
vi usa_used_car_dataset_project.py
Changing PYSPARK_CLI = True

<u>STEP 5:</u> To run the usa_used_car_dataset_project.py in spark-submit

spark-submit --conf spark.driver.memory=8g --conf spark.executor.memory=8g usa_used_car_dataset_project.py

```
spark-submit --conf spark.driver.memory=8g --conf spark.executor.memory=8g usa_used_car_dataset_project.py
```

Change the file_location as "/user/avalasa/ US_Used_Cars_Data.csv "

file_location = "/user/avalasa/ US_Used_Cars_Data.csv "

Results From Pyspark Submit

1)Linear Regression

a) **Cross Validation**

```
r2 For LR_CV: 0.63
RMSE For LR_CV: 6790.33
```

b) Train Validation

```
r2 For LR CV: 0.63
RMSE for LT TV: 6790.33
```

2)Gradient Boost Tree Regression

a) Cross Validation

```
RMSE_GBT_CV: 5427.82
R2_GBT_CV: 0.77
```

b) Train Validation

```
RMSE_GBT_TV: 5041.78
R2_GBT_TV: 0.80
```

3)Random Forest Regression

a) Cross Validation

```
RMSE for CV_RF: 4115.92
R2 for CV_RF: 0.87
```

b) Train Validation

```
RMSE for RF_TV: 4123.82
R2 for RF_TV: 0.86
```

4)Factorization Machine Regressor

a) Cross Validation

```
RMSE for FM_CV: 114680.51
R2 for FM_CV: -103.69
```

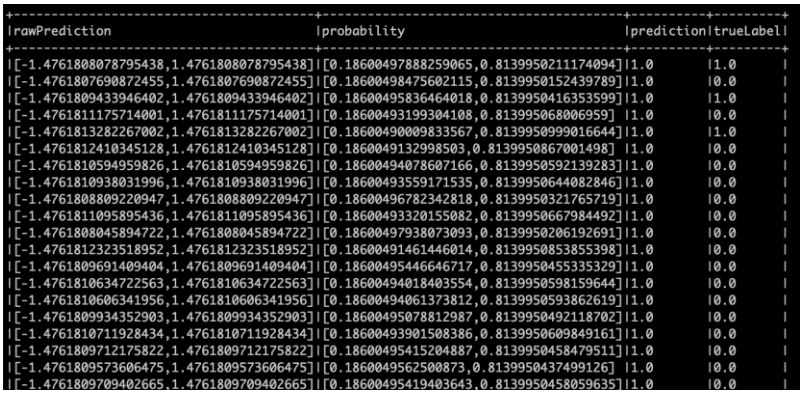
b) Train Validation

```
RMSE for FM_TV: 17640.81
R2 for FM_CV: -1.48
```

5) Logistic Regression

a) Cross Validation

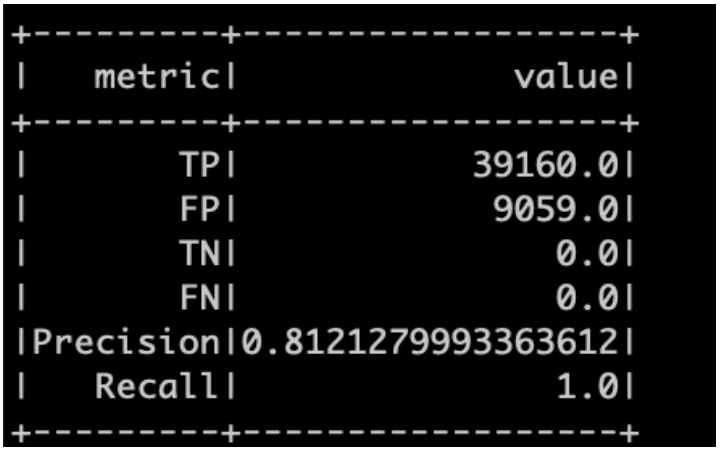
metric	value
TP	39160.0
FP	9059.0
TN	0.0
FN	0.0
Precision	0.8121279993363612
Recall	1.0



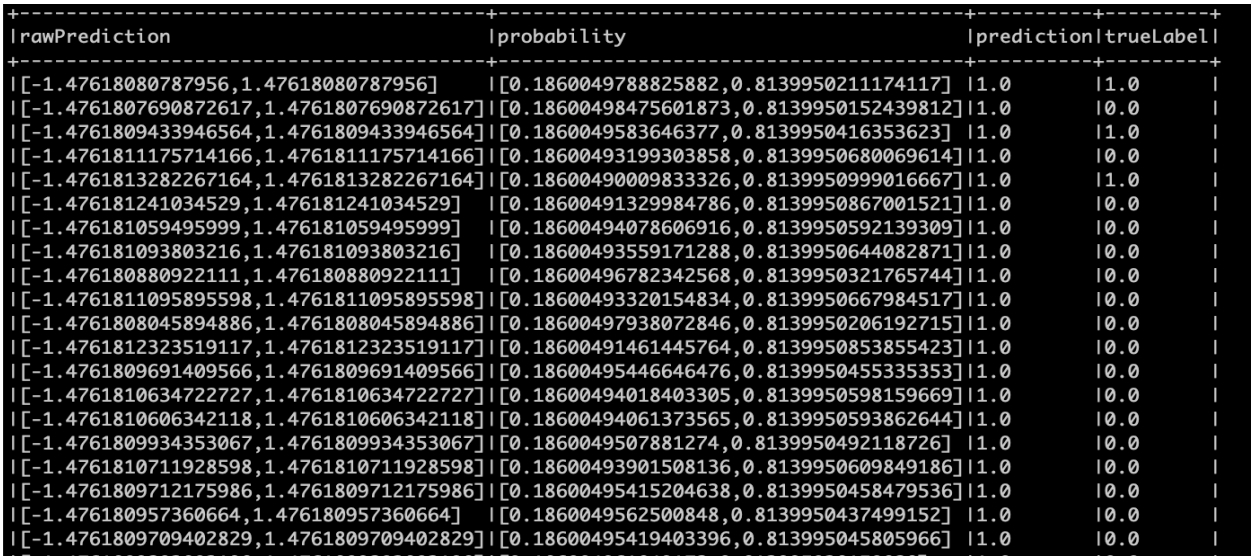
rawPrediction	probability	prediction	trueLabel
[-1.4761808078795438,1.4761808078795438]	[0.18600497888259065,0.8139950211174094]	1.0	1.0
[-1.4761807690872455,1.4761807690872455]	[0.1860049847560211 5,0.8139950152439789]	1.0	0.0
[-1.4761809433946402,1.4761809433946402]	[0.1860049583646401 8,0.8139950416353599]	1.0	1.0
[-1.4761811175714001,1.4761811175714001]	[0.1860049319930410 8,0.813995068006959]	1.0	0.0
[-1.4761813282267002,1.4761813282267002]	[0.1860049000983356 7,0.8139950999016644]	1.0	1.0
[-1.4761812410345128,1.4761812410345128]	[0.186004913299850 3,0.813995086700149 8]	1.0	0.0
[-1.4761810594959826,1.4761810594959826]	[0.1860049407860716 6,0.8139950592139283]	1.0	0.0
[-1.4761810938031996,1.4761810938031996]	[0.1860049355917153 5,0.8139950644082846]	1.0	0.0
[-1.4761808809220947,1.4761808809220947]	[0.1860049678234281 8,0.8139950321765719]	1.0	0.0
[-1.4761811095895436,1.4761811095895436]	[0.1860049332015508 2,0.8139950667984492]	1.0	0.0
[-1.4761808045894722,1.4761808045894722]	[0.1860049793807309 3,0.8139950206192691]	1.0	0.0
[-1.4761812323518952,1.4761812323518952]	[0.1860049146144601 4,0.8139950853855398]	1.0	0.0
[-1.4761809691409404,1.4761809691409404]	[0.1860049544664671 7,0.8139950455335329]	1.0	0.0
[-1.4761810634722563,1.4761810634722563]	[0.1860049401840355 4,0.8139950598159644]	1.0	0.0
[-1.4761810606341956,1.4761810606341956]	[0.1860049406137381 2,0.8139950593862619]	1.0	0.0
[-1.4761809934352903,1.4761809934352903]	[0.1860049507881298 7,0.8139950492118702]	1.0	0.0
[-1.4761810711928434,1.4761810711928434]	[0.1860049390150838 6,0.8139950609849161]	1.0	0.0
[-1.4761809712175822,1.4761809712175822]	[0.1860049541520488 7,0.8139950458479511]	1.0	0.0
[-1.4761809573606475,1.4761809573606475]	[0.186004956250087 3,0.813995043749912 6]	1.0	0.0
[-1.4761809709402665,1.4761809709402665]	[0.1860049541940364 3,0.8139950458059635]	1.0	0.0

```
AUC =  0.9546786242181969
```

a) Train Validation



metric	value
TP	39160.0
FP	9059.0
TN	0.0
FN	0.0
Precision	0.8121279993363612
Recall	1.0



rawPrediction	probability	prediction	trueLabel
[-1.47618080787956,1.47618080787956]	[0.1860049788825882,0.8139950211174117]	1.0	1.0
[-1.4761807690872617,1.4761807690872617]	[0.18600498475601873,0.8139950152439812]	1.0	0.0
[-1.4761809433946564,1.4761809433946564]	[0.1860049583646377,0.8139950416353623]	1.0	1.0
[-1.4761811175714166,1.4761811175714166]	[0.18600493199303858,0.8139950680069614]	1.0	0.0
[-1.4761813282267164,1.4761813282267164]	[0.18600490009833326,0.8139950999016667]	1.0	1.0
[-1.476181241034529,1.476181241034529]	[0.1860049132998478 6,0.8139950867001521]	1.0	0.0
[-1.476181059495999,1.476181059495999]	[0.18600494078606916,0.8139950592139309]	1.0	0.0
[-1.476181093803216,1.476181093803216]	[0.1860049355917128 8,0.8139950644082871]	1.0	0.0
[-1.476180880922111,1.476180880922111]	[0.18600496782342568,0.8139950321765744]	1.0	0.0
[-1.4761811095895598,1.4761811095895598]	[0.1860049332015483 4,0.8139950667984517]	1.0	0.0
[-1.4761808045894886,1.4761808045894886]	[0.18600497938072846,0.8139950206192715]	1.0	0.0
[-1.4761812323519117,1.4761812323519117]	[0.18600491461445764,0.8139950853855423]	1.0	0.0
[-1.4761809691409566,1.4761809691409566]	[0.18600495446646476,0.8139950455335353]	1.0	0.0
[-1.4761810634722727,1.4761810634722727]	[0.18600494018403305,0.8139950598159669]	1.0	0.0
[-1.4761810606342118,1.4761810606342118]	[0.18600494061373565,0.8139950593862644]	1.0	0.0
[-1.4761809934353067,1.4761809934353067]	[0.1860049507881274,0.8139950492118726]	1.0	0.0
[-1.4761810711928598,1.4761810711928598]	[0.18600493901508136,0.8139950609849186]	1.0	0.0
[-1.4761809712175986,1.4761809712175986]	[0.18600495415204638,0.8139950458479536]	1.0	0.0
[-1.476180957360664,1.476180957360664]	[0.1860049562500848,0.8139950437499152]	1.0	0.0
[-1.4761809709402829,1.4761809709402829]	[0.18600495419403396,0.813995045805966]	1.0	0.0

```
only showing top 100 rows

AUC =  0.95
```

Compare the Results

Once you run all the Regression models, you can compare the results of all the Regression models. The Results should look something like the below. According to the Comparison table below. We can arrange various Regression Algorithms in the below order.

Based on time:
GBT> RF>LR>FM> LGR (GBT taking the most time and Logistic Regression taking the least)

Based on accuracy:
RF> LGR> GBT>LR> FM (RF having the best accuracy and FM having the least)

Algorithm	**Results CV**	**Results TV**	**Time is taken to fit the model. For CV (For data bricks)**	**Time is taken to fit the model. For TV (For data bricks)**
Linear Regression (LR)	Databricks R2: 0.61 RMSE:7693.73 Spark Submit R2: 0.63 RMSE: 6790.33	Databricks R2: 0.61 RMSE:7693.73 Spark Submit R2:0.63 RMSE:6790.33	58.50 seconds	20.46 seconds
Random Forest Regression (RF)	Databricks R2: 0.74 RMSE:6197.13 Spark Submit R2: 0.87 RMSE:4115.92	Databricks R2: 0.73 RMSE:6758.30 Spark Submit R2: 0.86 RMSE: 4123.82	32.08 minutes	35.74 minutes

Gradient Boost Tree Regression (GBT)	Databricks R2: 0.67 RMSE: 7059.37 Spark Submit R2: 0.77 RMSE:5427.82	Databricks R2: 0.67 RMSE: 7009.82 Spark Submit R2: 0.80 RMSE:5041.78	12.81 Minutes	53.94 Minutes
Factorization Machine Regression	Databricks R2: -48.92 RMSE:95536.98 Spark Submit R2: -103.69 RMSE:114680.51	Databricks R2: -1.15 RMSE:19841.19 Spark Submit R2: -1.48 RMSE:17640.81	3.88 minutes	2.63 minutes
Logistic Regression	Databricks Precision: 0.94 Recall:0.97 AUC:0.99 Spark Submit Precision: 0.81 Recall: 1.0 AUC:0.95	Databricks Precision: 0.93 Recall:0.98 AUC:0.98 Spark Submit Precision: 0.8 Recall: 1.0 AUC: 0.95	3 Minutes	1Minute

Result:

After working with all the algorithms in Spark submit to predict the price of the used cars in the USA, we compared the RMSE and R2 results and decided that the Random Forest regressor is the best algorithm for our dataset. When we worked with the Random Forest regressor, we got the R2 value as 0.8 and RMSE as 4115. The second-best algorithm to predict the used car price is the Gradient Boost Tree Regression (GBT); we got the R2 value as 0.8 and RMSE as 5041. For the Linear Regression (LR), we got the R2 value as 0.6 and RMSE as 6790. The Factorization Machine Regressor model is the only model not a suggestible algorithm for our dataset. The Logistic regressor model is helpful with the recall, precision, and AUC values being accurate, allowing the customer to decide if they are getting a fair deal. Compared to the data bricks results, the Spark submit results are more precise, the R2 values are almost near 1, and the RMSE values are also low for all the algorithms.
Based on Spark Submit results, we arranged various Regression Algorithms below.

References:

1. URL of Data Source:
https://www.kaggle.com/datasets/ananaymital/us-used-cars-dataset

2. URL of your GitHub:

https://github.com/anushavalasapalli-97/CIS-5560-US_USED_CARS_PROJECT-

3. URL of References:

https://spark.apache.org/docs/2.2.0/ml-pipeline.html
https://spark.apache.org/docs/latest/ml-tuning.html

4. Databricks Html File URL:

file:///Users/anushavalasapalli/Downloads/US_Used_Cars_Dataset_Project.html