BIG DATA ANALYTICS

SPRING – 2022

MS(DS)

PROJECT 2

SPARK MLIB vs APACHE MAHOUT

PROJECT REPORT

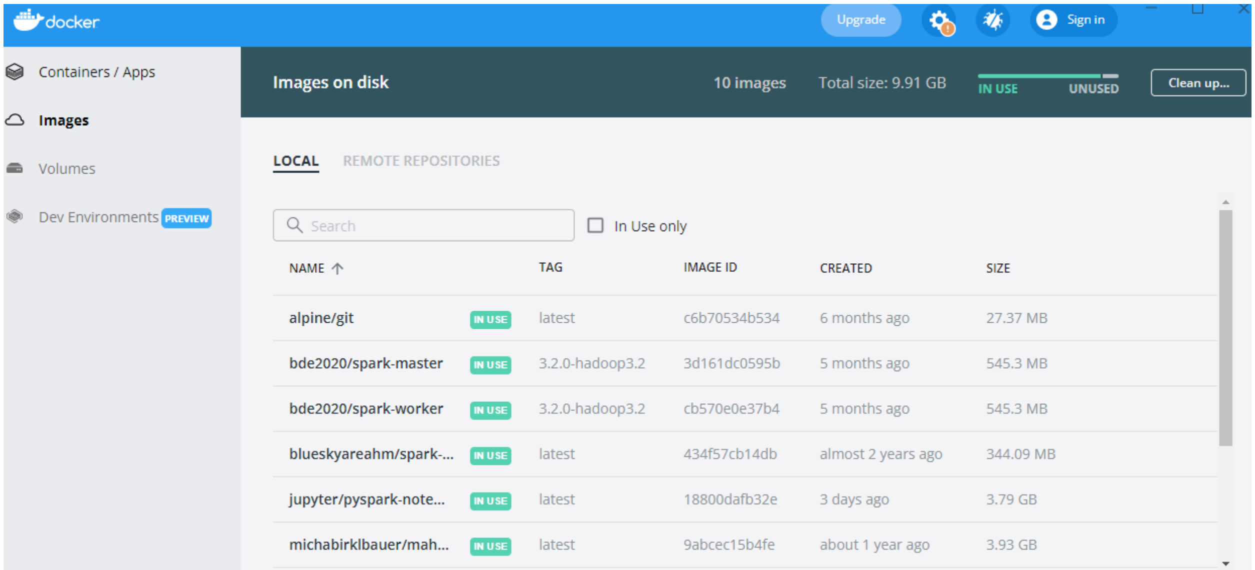
SUBMITTED BY : FAIZAN JALIL-(25370)

- **SYSTEM CONFIGURATION**

System Information

Item	Value
OS Name	Microsoft Windows 10 Enterprise
Version	10.0.19042 Build 19042
Other OS Description	Not Available
OS Manufacturer	Microsoft Corporation
System Name	DESKTOP-DT1VIOE
System Manufacturer	Hewlett-Packard
System Model	HP 14 Notebook PC
System Type	x64-based PC
System SKU	L5E61EA#ABV
Processor	Intel(R) Core(TM) i5-5200U CPU @ 2.20GHz, 2201 Mhz, 2 Core(s), 4 Logical Pr...
BIOS Version/Date	Insyde F.36, 12/18/2014
SMBIOS Version	2.7
Embedded Controller Version	5.24
BIOS Mode	UEFI
BaseBoard Manufacturer	Hewlett-Packard
BaseBoard Product	2335
BaseBoard Version	05.24
Platform Role	Mobile
Secure Boot State	Off
PCR7 Configuration	Binding Not Possible
Windows Directory	C:\WINDOWS
System Directory	C:\WINDOWS\system32
Boot Device	\Device\HarddiskVolume2
Locale	United States
Hardware Abstraction Layer	Version = "10.0.19041.1566"
User Name	DESKTOP-DT1VIOE\Administrator
Time Zone	Pakistan Standard Time
Installed Physical Memory (RAM)	8.00 GB
Total Physical Memory	7.92 GB
Available Physical Memory	2.87 GB
Total Virtual Memory	9.92 GB
Available Virtual Memory	4.56 GB
Page File Space	2.00 GB
Page File	C:\pagefile.sys
Kernel DMA Protection	Off
Virtualization-based security	Running
Virtualization-based security Req...	
Virtualization-based security Avail...	Base Virtualization Support, DMA Protection
Virtualization-based security Servi...	
Virtualization-based security Servi...	
Device Encryption Support	Reasons for failed automatic device encryption: TPM is not usable, PCR7 bindi...
A hypervisor has been detected. ...	

- **DOCKER DESKTOP SETUP & CONFIGURATION**



- **Pulling Spark Master using the Docker Compose File**

```
D:\spark>docker-compose -f docker-compose.yml up -d
Pulling spark-master (bde2020/spark-master:3.2.0-hadoop3.2)...
3.2.0-hadoop3.2: Pulling from bde2020/spark-master
396c31837116: Pull complete
e54fd3abc483: Pull complete
99fe4892ef97: Pull complete
77fc61a0e20d: Pull complete
fc2ead407a6b: Pull complete
7157b1c96512: Pull complete
Digest: sha256:cb7cca9cec663f4dec6b6a6fb2d83303fdca565dc6457db65de0a88eee1d7244
Status: Downloaded newer image for bde2020/spark-master:3.2.0-hadoop3.2
Pulling spark-worker-1 (bde2020/spark-worker:3.2.0-hadoop3.2)...
3.2.0-hadoop3.2: Pulling from bde2020/spark-worker
396c31837116: Already exists
e54fd3abc483: Already exists
99fe4892ef97: Already exists
77fc61a0e20d: Already exists
fc2ead407a6b: Already exists
c8decaab2964: Pull complete
Digest: sha256:8f241ff39526d939d666917e628e2096f302e7b00b05635f837582a42949688f
Status: Downloaded newer image for bde2020/spark-worker:3.2.0-hadoop3.2
Creating spark-master ... done
Creating spark-worker-1 ... done
```

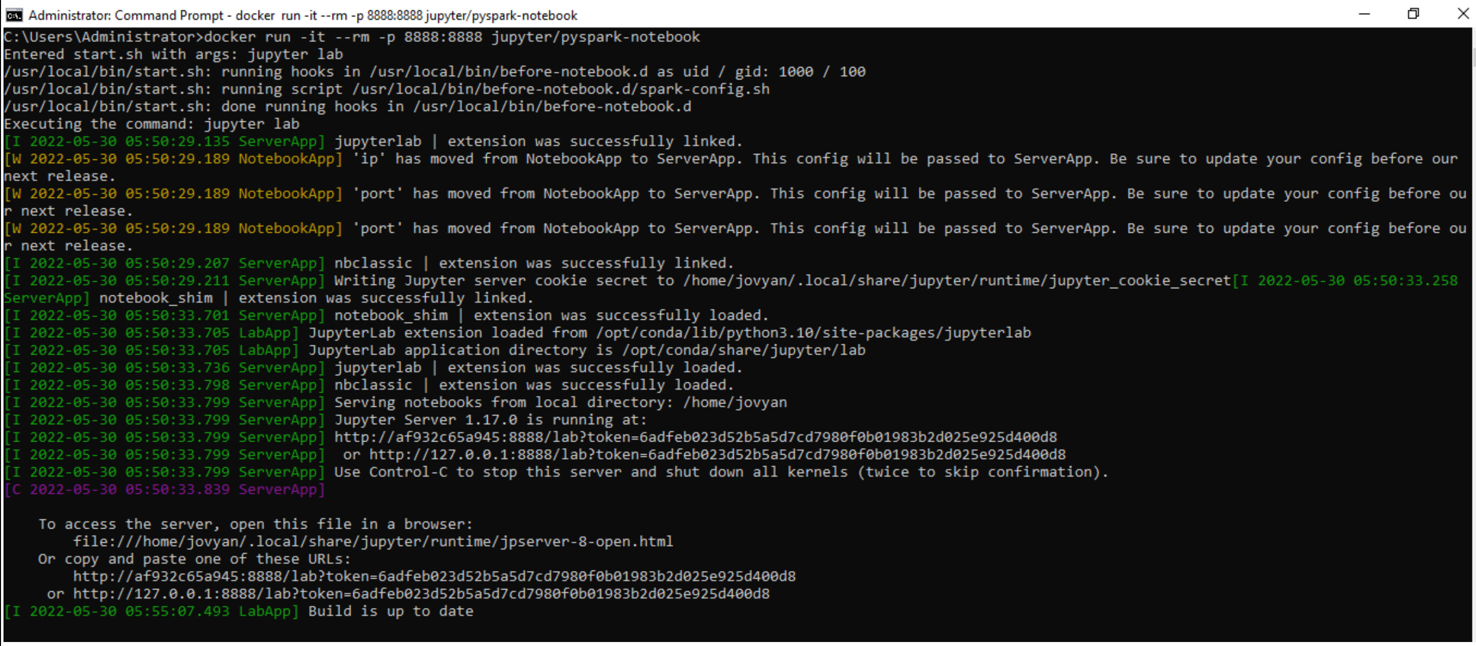
INTRODUCTION TO SPARK MLIB (MACHINE LEARNING LIBRARY) & APACHE MAHOUT

- **Spark ML** is not an official name but occasionally used to refer to the MLlib DataFrame-based API. This is majorly due to the org.apache.spark.ml Scala package name used by the DataFrame-based API, and the "Spark ML Pipelines" term is used to emphasize the pipeline concept.
- The primary Machine Learning API for Spark is the DataFrame-based API in the spark.ml package.
- Spark's machine learning (ML) library. Its goal is to make practical machine learning scalable and easy. At a high level, it provides tools such as: ML Algorithms: common learning algorithms such as classification, regression, clustering, and collaborative filtering.
- It is a scalable machine learning library consisting of common learning algorithms and utilities, including classification, regression, clustering, collaborative filtering, dimensionality reduction, and underlying optimization primitives.
- PySpark is the Python API for Apache Spark, an open source, distributed computing framework and set of libraries for real-time, large-scale data processing. If you're already familiar with Python and libraries such as Pandas, then PySpark is a good language to learn to create more scalable analyses and pipelines.
- PySpark is a Python-based API for utilizing the Spark framework in combination with Python.
- Apache Spark is written in Scala programming language. PySpark has been released in order to support the collaboration of Apache Spark and Python, it actually is a Python API for Spark. In addition, PySpark, helps you interface with Resilient Distributed Datasets (RDDs) in Apache Spark and Python programming language. It works on distributed systems for data analysis.
- **Apache Mahout** is a distributed linear algebra framework and mathematically expressive Scala DSL designed to let mathematicians, statisticians, and data scientists quickly implement their own algorithms. Apache Spark is the recommended out-of-the-box distributed back-end, or can be extended to other distributed backends.
- It has Mathematically Expressive Scala DSL. It Support for Multiple Distributed Backends (including Apache Spark). It has Modular Native Solvers for CPU/GPU/CUDA Acceleration
- Apache Mahout is a highly scalable machine learning library that enables developers to use optimized algorithms. Mahout implements popular machine learning techniques such as recommendation, classification, and clustering.

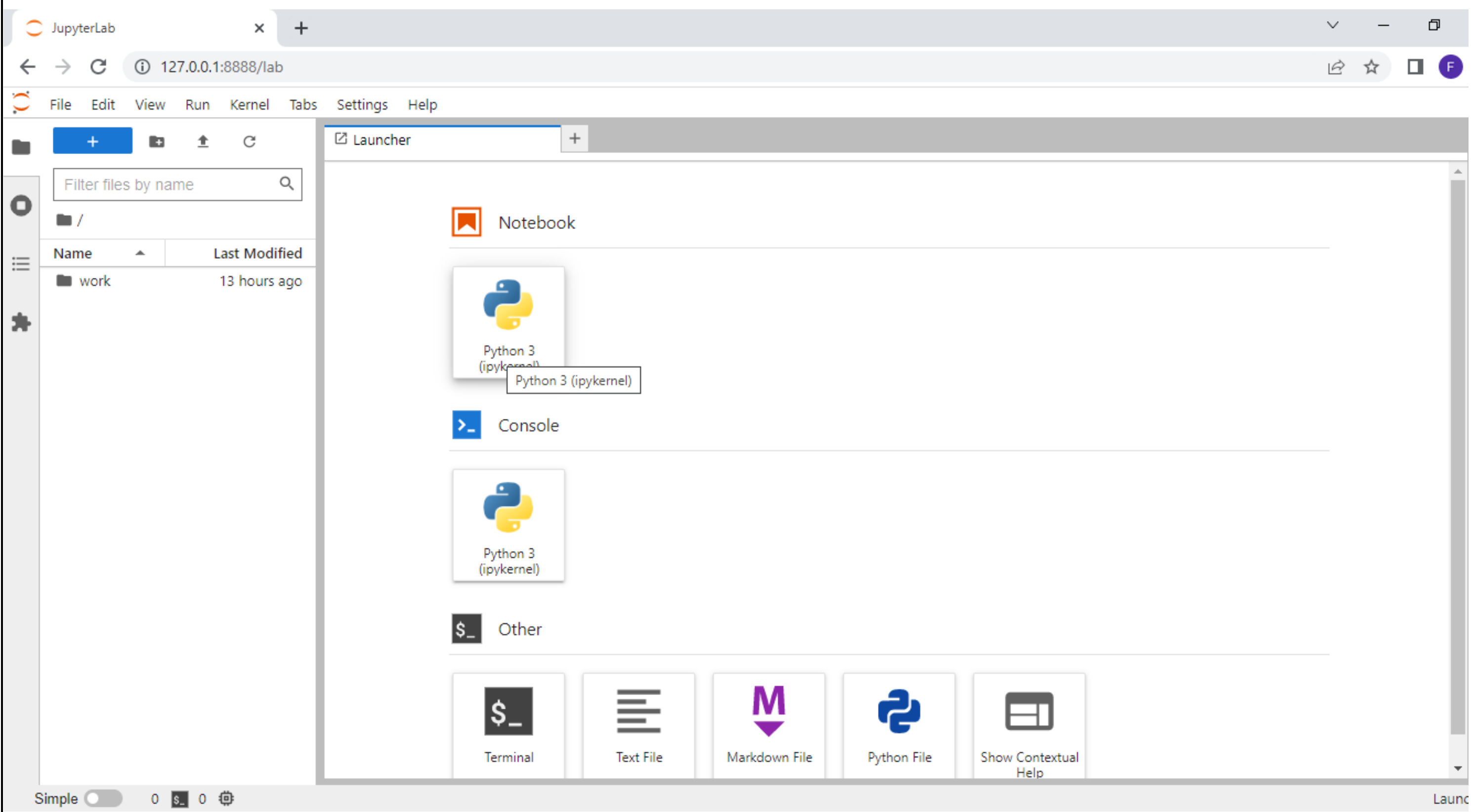
- **<u>RUNNING JUPYTER PYSPARK NOTEBOOK IN DOCKER – MLIB USING SPARK</u>**

Using the following command the jupyter lab extension is linked with docker and further link is achieved to run the pyspark-notebook on the web browser

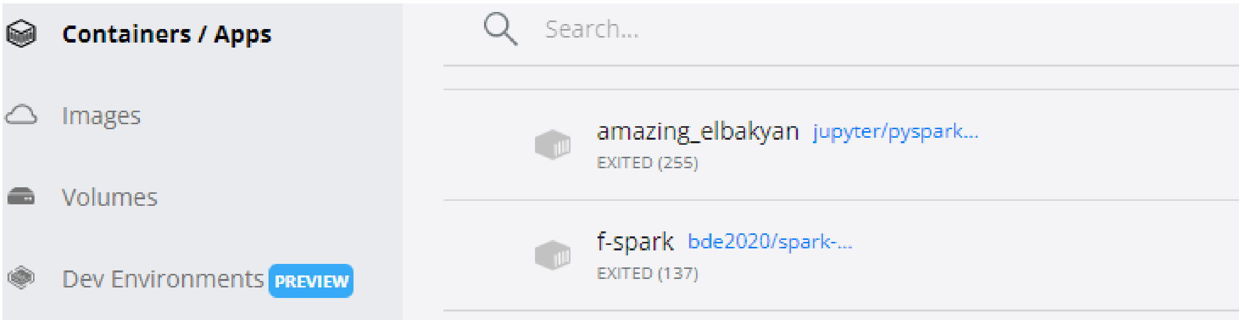
- Docker run –it –rm –p 8888:8888 juyter/pyspark-notebook



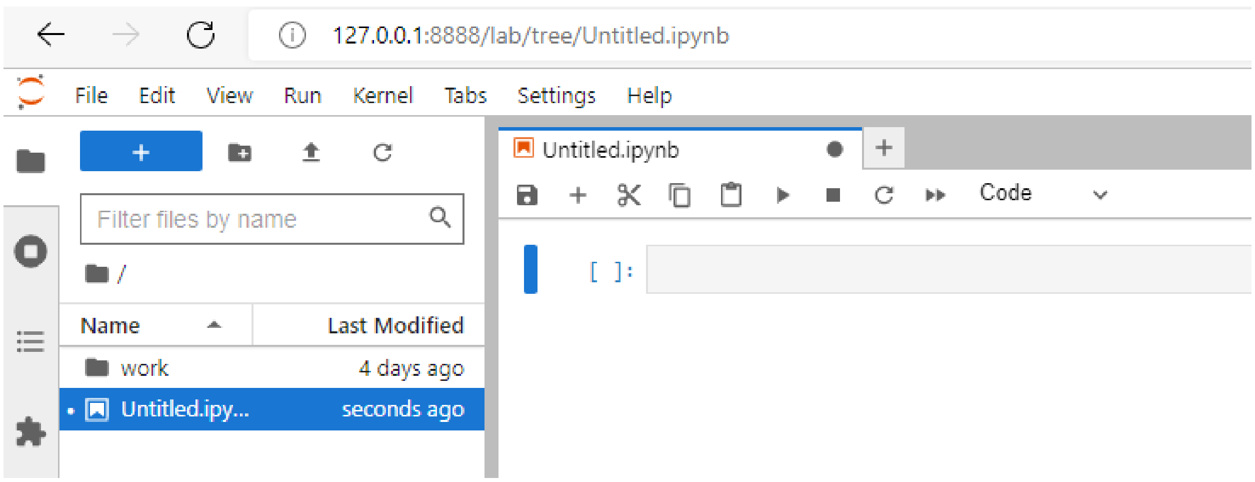
- Using the local host server access link following linked Jupyter Lab is accessed



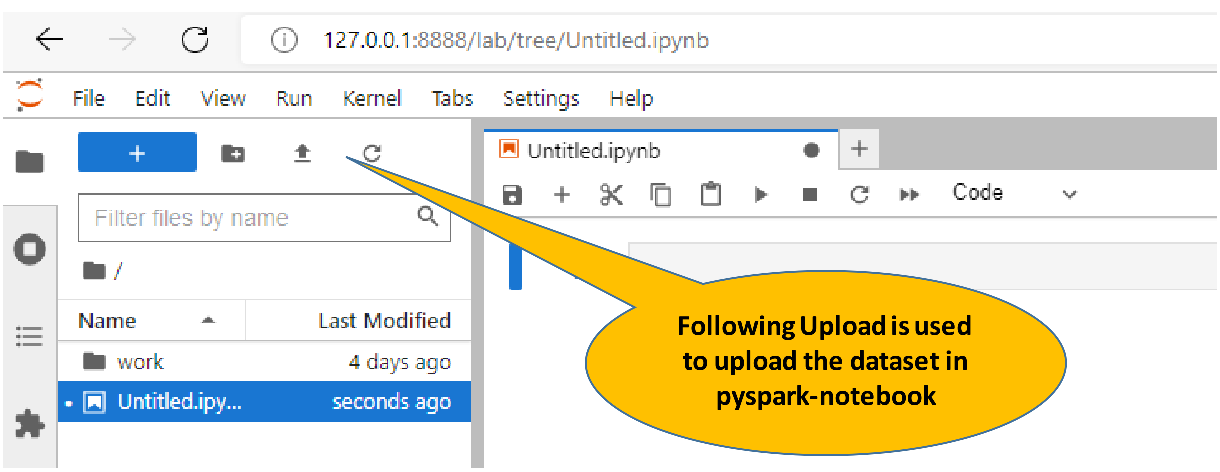
- A new Python 3 notebook is opened for further working on Docker based Jupyter Notebook
- A container of Jupyter/Pyspark starts to run in docker



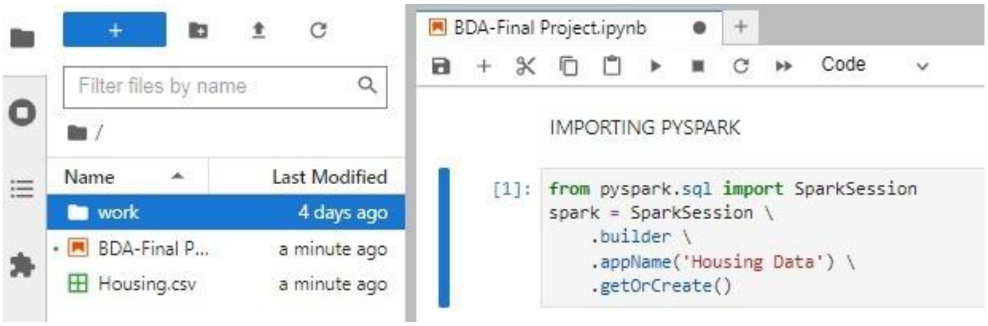
- The work environment ("Notebook") is ready for implementation of code



- The data is then uploaded on the Pyspark-Notebook



- Housing Data consisting of over 400mb and 79 several Features is uploaded in the Jupyter-Pyspark Notebook. For further working of spark.ml.regression



- <u>**Getting Started with Code Implementation of Machine Learning Algorithms**</u>
    - Using pyspark.sql to Import SparkSession in the Jupyter-pyspark Notebook

- <u>**LOADING DATASET**</u>
    - The dataset in Jupyter-pyspark notebook is then Read and Loaded using spark
    - The data is a Housing Data set consisting of prediction of Price of Houses based on different features

- **DATAFRAME HEAD & DATA TYPES**
    - The head of dataframe is visualized using the following commands

DATAFRAME HEAD & DATA TYPES

```
[3]: df.head()
```

```
[3]: Row(id='7129300520', date='20141013T000000', price='221900', bedrooms='3', bathrooms='1', sqft_living='1180', sqft_lot='5650', floors='1', waterfront='0', view='0', condition='3', grade='7', sqft_above='1180', sqft_basement='0', yr_built='1955', yr_renovated='0', zipcode='98178', lat='47.5112', long='-122.257', sqft_living15='1340', sqft_lot15='5650', sqft_living1='1340', sqft_lot1='5650', sqft_living2='1340', sqft_lot2='5650', sqft_living3='1340', sqft_lot3='5650', sqft_living4='1340', sqft_lot4='5650', sqft_living5='1340', sqft_lot5='5650', sqft_living6='1340', sqft_lot6='5650', sqft_living7='1340', sqft_lot7='5650', sqft_living8='1340', sqft_lot8='5650', sqft_living9='1340', sqft_lot9='5650', sqft_living10='1340', sqft_lot10='5650', sqft_living11='1340', sqft_lot11='5650', sqft_living12='1340', sqft_lot12='5650', sqft_living13='1340', sqft_lot13='5650', sqft_living14='1340', sqft_lot14='5650', sqft_living16='1340', sqft_lot16='5650', sqft_living17='1340', sqft_lot17='5650', sqft_living18='1340', sqft_lot18='5650', sqft_living19='1340', sqft_lot19='5650', sqft_living20='1340', sqft_lot20='5650', sqft_living21='1340', sqft_lot21='5650', sqft_living22='1340', sqft_lot22='5650', sqft_living23='1340', sqft_lot23='5650', sqft_living24='1340', sqft_lot24='5650', sqft_living25='1340', sqft_lot25='5650', sqft_living26='1340', sqft_lot26='5650', sqft_living27='1340', sqft_lot27='5650', sqft_living28='1340', sqft_lot28='5650', sqft_living29='1340', sqft_lot29='5650', sqft_living30='1340', sqft_lot30='5650')
```

- Visualizing the datatypes for further type casting

```
[4]: df.dtypes
```

```
[4]: [('id', 'string'),
      ('date', 'string'),
      ('price', 'string'),
      ('bedrooms', 'string'),
      ('bathrooms', 'string'),
      ('sqft_living', 'string'),
      ('sqft_lot', 'string'),
      ('floors', 'string'),
      ('waterfront', 'string'),
      ('view', 'string'),
      ('condition', 'string'),
      ('grade', 'string'),
      ('sqft_above', 'string'),
      ('sqft_basement', 'string'),
      ('yr_built', 'string'),
      ('yr_renovated', 'string'),
      ('zipcode', 'string'),
      ('lat', 'string'),
      ('long', 'string'),
      ('sqft_living15', 'string'),
      ('sqft_lot15', 'string')]
```

- **SUMMARY – DATAFRAME**
  - The insights and mathematical deviation of all the columns are calculated and visualized using
    - .describe().toPandas().transpose()

SUMMARY - DATAFRAME FEATURES

```
[5]: df.describe().toPandas().transpose()
```

[5]:

	0	1	2	3	4
summary	count	mean	stddev	min	max
id	1048574	4.579861275310459E9	2.876391651955958 4E9	1000102	999000215
date	1048574	None	None	20140502T000000	20150527T000000
price	1048574	540000.9407280745	367160.25436746876	1.00E+06	999999
bedrooms	1048574	3.3707616248352523	0.9298424990258323	0	9
bathrooms	1048574	2.114187935233946	0.7700672393270263	0	8
sqft_living	1048574	2079.600361061785	918.2777039636362	1000	998
sqft_lot	1048574	15114.805397616192	41446.225024989784	1000	9999
floors	1048574	1.4936575768615281	0.5397294437285176	1	3.5
waterfront	1048574	0.007542624554871664	0.08652017399813725	0	1
view	1048574	0.2343306242573247	0.7662741145124068	0	4
condition	1048574	3.4098194309605234	0.6508781415110932	1	5
grade	1048574	7.6562483906715215	1.1752864651032575	1	9
sqft_above	1048574	1787.9822511334442	827.8630287542384	1000	998
sqft_basement	1048574	291.6181099283408	442.6381938292455	0	990
yr_built	1048574	1970.9668282829823	29.36000258050457	1900	2015
yr_renovated	1048574	84.46046058742635	401.8029910667028	0	2015
zipcode	1048574	98077.9390419751	53.50842796395778	98001	98199
lat	1048574	47.560051383497864	0.1385778728621862	47.1559	47.7776
long	1048574	-122.21389457396674	0.14081404352083712	-121.315	-122.519
sqft_living15	1048574	1986.4422167629561	685.231818483224	1000	998
sqft_lot15	1048574	12772.798134418745	27296.499607930138	10000	9999

- FEATURE IMPORTANCE

Using Feature Importance to identify the importance and most correlated features

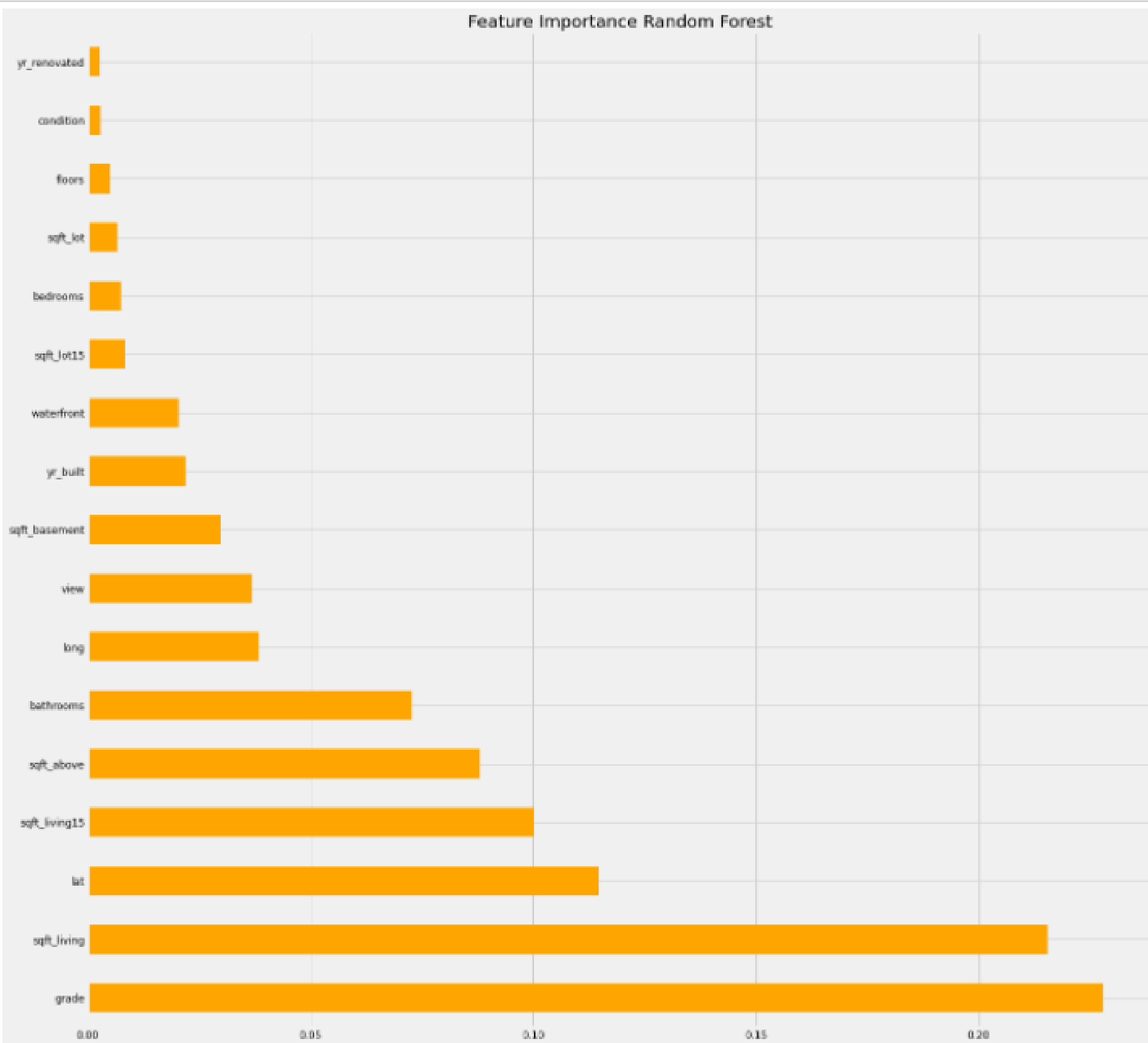
```
[60]: modelRF.featureImportances
```

```
[60]: SparseVector(16, {0: 0.0021, 1: 0.0565, 2: 0.2385, 3: 0.002, 4: 0.0002, 5: 0.0378, 6: 0.0208, 7: 0.0001, 8: 0.3165, 9: 0.0016, 10:
0.0292, 11: 0.0012, 12: 0.1324, 13: 0.0414, 14: 0.1146, 15: 0.005})
```

```
[61]: training_data.take(1)
```

```
[61]: [Row(price=75000.0, bedrooms=1, bathrooms=0.0, sqft_living=670, sqft_lot=43377, floors=1.0, waterfront=0, view=0, condition=3, gra
de=3, sqft_basement=0, yr_built=1966, yr_renovated=0, lat=47.26380157470703, long=-121.90599822998047, sqft_living15=1160, sqft_lo
t15=42882, features=DenseVector([1.0, 0.0, 670.0, 43377.0, 1.0, 0.0, 0.0, 3.0, 3.0, 0.0, 1966.0, 0.0, 47.2638, -121.906, 1160.0, 4
2882.0]))]
```

```
regRF = RandomForestRegressor(max_depth=6, max_features=4, min_samples_split=8,
                              n_estimators=300)#, random_state=0)
regRF.fit(X_train, Y_train)
importance_rf = pd.Series(regRF.feature_importances_, index = X_train.columns)
importance_rf_sorted = importance_rf.sort_values()
importance_rf_sorted.nlargest(20).plot(kind='barh', color='orange')
plt.title("Feature Importance Random Forest")
plt.show()
```



- **SELECTING COLUMNS & CHANGING DATATYPES OF FEATURES**

```python
[6]: from pyspark.sql.functions import col
     dataset = df.select(col('price').cast('float'),
                          col('bedrooms').cast('int'),
                          col('bathrooms').cast('float'),
                          col('sqft_living').cast('int'),
                          col('sqft_lot').cast('int'),
                          col('floors').cast('float'),
                         col('waterfront').cast('int'),
                         col('view').cast('int'),
                         col('condition').cast('int'),
                         col('grade').cast('int'),
                         col('sqft_basement').cast('int'),
                         col('yr_built').cast('int'),
                         col('yr_renovated').cast('int'),
                         col('lat').cast('float'),
                         col('long').cast('float'),
                         col('sqft_living15').cast('int'),
                         col('sqft_lot15').cast('int')
                         )
     dataset.show()
```

```
+---------+--------+---------+-----------+--------+------+----------+----+---------+-----+-------------+-----
---+------------+-------+--------+------------+----------+
|    price|bedrooms|bathrooms|sqft_living|sqft_lot|floors|waterfront|view|condition|grade|sqft_basement|yr_bu
ilt|yr_renovated|    lat|    long|sqft_living15|sqft_lot15|
+---------+--------+---------+-----------+--------+------+----------+----+---------+-----+-------------+-----
---+------------+-------+--------+------------+----------+
| 221900.0|       3|      1.0|       1180|    5650|   1.0|         0|   0|        3|    7|            0|    1
955|           0|47.5112|-122.257|        1340|      5650|
| 538000.0|       3|     2.25|       2570|    7242|   2.0|         0|   0|        3|    7|          400|    1
951|        1991| 47.721|-122.319|        1690|      7639|
| 180000.0|       2|      1.0|        770|   10000|   1.0|         0|   0|        3|    6|            0|    1
933|           0|47.7379|-122.233|        2720|      8062|
| 604000.0|       4|      3.0|       1960|    5000|   1.0|         0|   0|        5|    7|          910|    1
965|           0|47.5208|-122.393|        1360|      5000|
| 510000.0|       3|      2.0|       1680|    8080|   1.0|         0|   0|        3|    8|            0|    1
987|           0|47.6168|-122.045|        1800|      7503|
|1230000.0|       4|      4.5|       5420|  101930|   1.0|         0|   0|        3|   11|         1530|    2
001|           0|47.6561|-122.005|        4760|    101930|
| 257500.0|       3|     2.25|       1715|    6819|   2.0|         0|   0|        3|    7|            0|    1
995|           0|47.3097|-122.327|        2238|      6819|
| 291850.0|       3|      1.5|       1060|    9711|   1.0|         0|   0|        3|    7|            0|    1
963|           0|47.4095|-122.315|        1650|      9711|
| 229500.0|       3|      1.0|       1780|    7470|   1.0|         0|   0|        3|    7|          730|    1
960|           0|47.5123|-122.337|        1780|      8113|

+---------+--------+---------+-----------+--------+------+----------+----+---------+-----+-------------+-----
---+------------+-------+--------+------------+----------+-------------------+
only showing top 20 rows
```

- **Using Vector Assembler**
    - Using .pyspark.Vector Assembler() to select the input and output columns features based on the required features

MODEL TRAINING

```
[8]: # Assemble all the features with VectorAssembler
required_features = ['bedrooms','bathrooms','sqft_living','sqft_lot','floors','waterfront','view','condition',
                     'yr_built','yr_renovated','lat','long','sqft_living15','sqft_lot15']
from pyspark.ml.feature import VectorAssembler
assembler = VectorAssembler(inputCols=required_features, outputCol='features')
transformed_data = assembler.transform(dataset)
# transformed_data.select('features').show()
transformed_data.show()
```

```
+---------+--------+---------+-----------+--------+------+----------+----+---------+-----+-------------+-----
---+------------+-------+--------+------------+----------+--------------------+
|    price|bedrooms|bathrooms|sqft_living|sqft_lot|floors|waterfront|view|condition|grade|sqft_basement|yr_bu
ilt|yr_renovated|    lat|    long|sqft_living15|sqft_lot15|            features|
+---------+--------+---------+-----------+--------+------+----------+----+---------+-----+-------------+-----
---+------------+-------+--------+------------+----------+--------------------+
| 221900.0|       3|      1.0|       1180|    5650|   1.0|         0|   0|        3|    7|            0|    1
955|           0|47.5112|-122.257|        1340|      5650|[3.0,1.0,1180.0,5...|
| 538000.0|       3|     2.25|       2570|    7242|   2.0|         0|   0|        3|    7|          400|    1
951|        1991| 47.721|-122.319|        1690|      7639|[3.0,2.25,2570.0,...|
| 180000.0|       2|      1.0|        770|   10000|   1.0|         0|   0|        3|    6|            0|    1
933|           0|47.7379|-122.233|        2720|      8062|[2.0,1.0,770.0,10...|
| 604000.0|       4|      3.0|       1960|    5000|   1.0|         0|   0|        5|    7|          910|    1
965|           0|47.5208|-122.393|        1360|      5000|[4.0,3.0,1960.0,5...|
| 510000.0|       3|      2.0|       1680|    8080|   1.0|         0|   0|        3|    8|            0|    1
987|           0|47.6168|-122.045|        1800|      7503|[3.0,2.0,1680.0,8...|

+---------+--------+---------+-----------+--------+------+----------+----+---------+-----+-------------+-----
---+------------+-------+--------+------------+----------+--------------------+
only showing top 20 rows
```

- **SPLITTING DATASET USING TRAIN/TEST SPLIT**

- The selected dataset is then randomSplit into training data and testing data with 70:30 ratio

```
[9]: (training_data, test_data) = transformed_data.randomSplit([0.7,0.3])
```

- This will split the dataframe into Testing Data and Training Data which would be further used in fitting and testing the model through

- **Visualizing the training data**

```
[10]: training_data.show()
```

```
+-------+--------+---------+-----------+--------+------+----------+----+---------+-----+-------------+-------
-+------------+-------+--------+------------+---------+--------------------+
|  price|bedrooms|bathrooms|sqft_living|sqft_lot|floors|waterfront|view|condition|grade|sqft_basement|yr_buil
t|yr_renovated|    lat|    long|sqft_living15|sqft_lot15|            features|
+-------+--------+---------+-----------+--------+------+----------+----+---------+-----+-------------+-------
-+------------+-------+--------+------------+---------+--------------------+
|75000.0|       1|      0.0|        670|   43377|   1.0|         0|   0|        3|    3|            0|    196
6|           0|47.2638|-121.906|        1160|    42882|[1.0,0.0,670.0,43...|
|75000.0|       1|      0.0|        670|   43377|   1.0|         0|   0|        3|    3|            0|    196
6|           0|47.2638|-121.906|        1160|    42882|[1.0,0.0,670.0,43...|
|75000.0|       1|      0.0|        670|   43377|   1.0|         0|   0|        3|    3|            0|    196
6|           0|47.2638|-121.906|        1160|    42882|[1.0,0.0,670.0,43...|
|75000.0|       1|      0.0|        670|   43377|   1.0|         0|   0|        3|    3|            0|    196
6|           0|47.2638|-121.906|        1160|    42882|[1.0,0.0,670.0,43...|
|75000.0|       1|      0.0|        670|   43377|   1.0|         0|   0|        3|    3|            0|    196
6|           0|47.2638|-121.906|        1160|    42882|[1.0,0.0,670.0,43...|

+---------+--------+---------+-----------+--------+------+----------+----+---------+-----+-------------+-----
---+------------+-------+--------+------------+---------+--------------------+
only showing top 20 rows
```

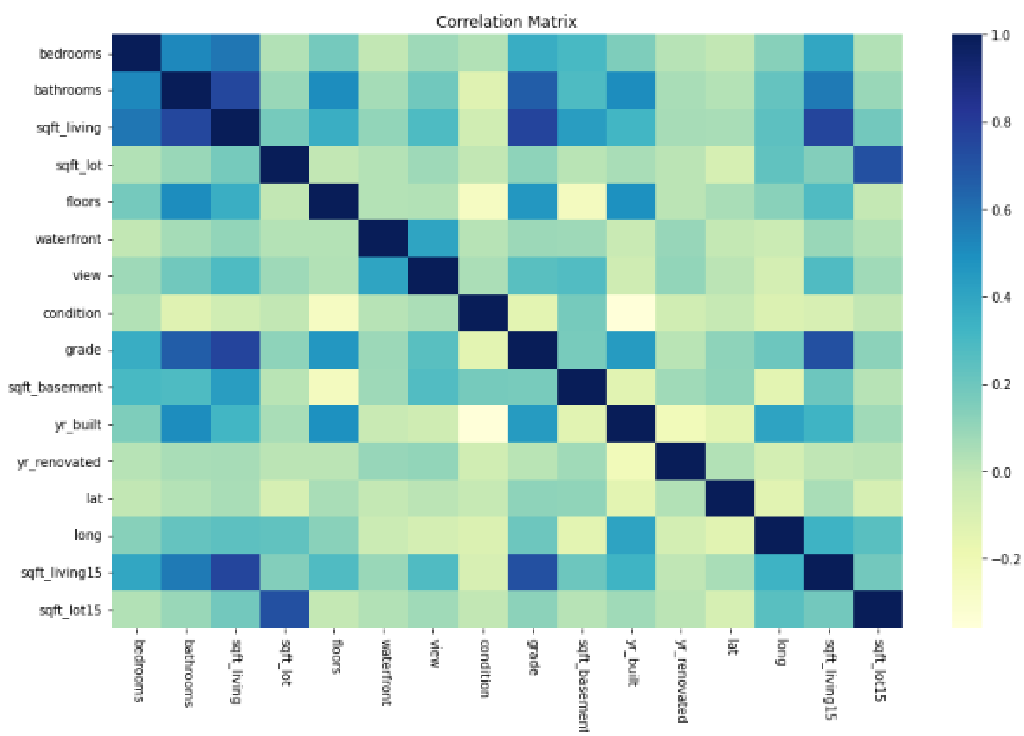
- **CORRELATION MATRIX**

  The Correlation Matrix identifies the correlation and relevancy between features, the seaborn is imported to plot the matrix with Features on Y-Axis & relevance on X-Axis.

```
[30]: from pyspark.ml.stat import Correlation

matrix = Correlation.corr(training_data.select('features'), 'features')
matrix_np = matrix.collect()[0]["pearson({})".format('features')].values
```

```
[31]: from IPython.core.display import display, HTML # my imports

# annot = dataset.display_image(21, use_url=False) #my function return a html page

# HTML(annot) # used for displaying the page

%matplotlib widget
%matplotlib inline
```

```
[32]: import seaborn as sns

matrix_np = matrix_np.reshape(len(required_features),len(required_features))

fig, ax = plt.subplots(figsize=(12,8))
ax = sns.heatmap(matrix_np, cmap="YlGnBu")
ax.xaxis.set_ticklabels(required_features, rotation=270)
ax.yaxis.set_ticklabels(required_features, rotation=0)
ax.set_title("Correlation Matrix")
plt.tight_layout()
plt.show()
```



TRAINING MODELS

Have implemented several Machine Learning Models using spark ml libraries

1) RANDOM FOREST REGRESSION

- **IMPORTING LIBRARIES & FUNCTION WITH PARAMETERS**

```
[15]: from pyspark.ml.regression import RandomForestRegressor
      rf = RandomForestRegressor(labelCol="price", featuresCol="features")
```

- MODEL FIT

```
[16]: import time
```

```
[17]: start_timeRF = time.perf_counter()

      modelRF = rf.fit(training_data)
      predictionsRF = modelRF.transform(test_data)

      end_timeRF = time.perf_counter()
      training_timeRF = end_timeRF - start_timeRF
```

- **RANDOM FOREST REGRESSION – TRAINING TIME**

- EXECUTION TIME & PERFORMANCE

```
[18]: print('Random Forest Model Start Time = ', start_timeRF)
      print('Random Forest Model End Time = ', end_timeRF)
      print('Random Forest Model Total Training Time = ', training_timeRF)

      Random Forest Model Start Time =  4276.7897
      Random Forest Model End Time =  4378.5083438
      Random Forest Model Total Training Time =  101.71864379999988
```

- **RMSE & R SQUARED – TEST ACCURACIES**

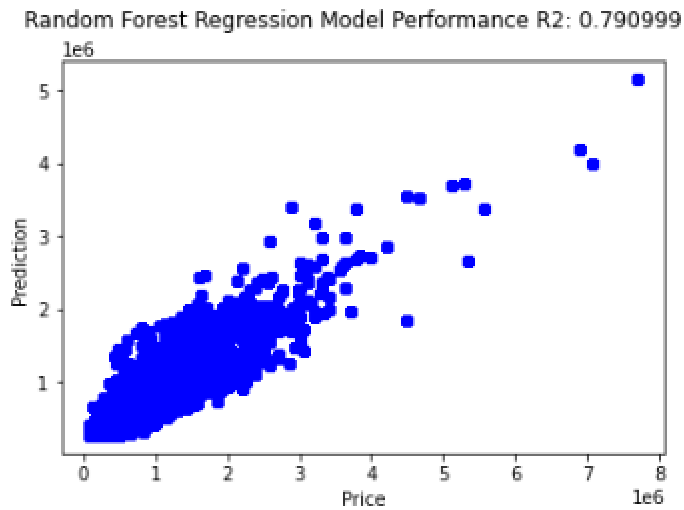
- The Test Accurancy using R2 and RMSE score in Random Forest is calculated using the .pysparl.ml.evaluation by importing RegressionEvaluator

```
[19]: # Evaluate our model
      from pyspark.ml.evaluation import RegressionEvaluator, MulticlassClassificationEvaluator
      #evaluator = MulticlassClassificationEvaluator(
      #    labelCol='price',
      #    predictionCol='prediction',
      #    metricName='accuracy')
      evaluatorrf1 = RegressionEvaluator(labelCol="price", predictionCol="prediction", metricName="rmse")
      evaluatorrf2 = RegressionEvaluator(labelCol="price", predictionCol="prediction", metricName="r2")
```

```
[20]: accuracy1 = evaluatorrf1.evaluate(predictionsRF)
      accuracy2 = evaluatorrf2.evaluate(predictionsRF)
      print('Random Forest Regression RMSE Test Accuracy = ', accuracy1)
      print('Random Forest Regression R2 Test Accuracy = ', accuracy2)

      Random Forest Regression RMSE Test Accuracy =  167386.04013005417
      Random Forest Regression R2 Test Accuracy =  0.7909986431832868
```

- **PERFORMANCE VISUALIZATION – RANDOM FOREST REGRESSION**

[21]:
```
import matplotlib.pyplot as plt

rmse = evaluatorrf1.evaluate(predictionsRF)
r2 = evaluatorrf2.evaluate(predictionsRF)
rfPred = modelRF.transform(test_data)
rfResult = rfPred.toPandas()
plt.plot(rfResult.price, rfResult.prediction, 'bo')
plt.xlabel('Price')
plt.ylabel('Prediction')
plt.suptitle("Random Forest Regression Model Performance RMSE: %f" % rmse)
plt.suptitle("Random Forest Regression Model Performance R2: %f" % r2)
plt.show()
```



2) LINEAR REGRESSION

- Using the pyspark.ml.regression to Import Linear Regression
- Setting the Parameters
- Fitting the Model & Calculating the Training Time (Execution Time)

```
[22]: from pyspark.ml.regression import LinearRegression
lr = LinearRegression(featuresCol = 'features', labelCol='price', maxIter=10, regParam=0.3, elasticNetParam=0.8)
```

```
[23]: start_timeLR = time.perf_counter()

lr_model = lr.fit(training_data)
predictionsLR = lr_model.transform(test_data)

end_timeLR = time.perf_counter()

training_timeLR = end_timeLR - start_timeLR
```

```
[24]: print('Linear Regression Model Start Time = ', start_timeLR)
print('Linear Regression Model End Time = ', end_timeLR)
print('Linear Regression Model Total Training Time = ', training_timeLR)
```

```
Linear Regression Model Start Time =  4569.8371016
Linear Regression Model End Time =  4620.4712325
Linear Regression Model Total Training Time =  50.634130899999946
```

- **CALCULATING COEFFICIENTS & INTERCEPTS**

- Calculating coefficient and intercept for the model

```
[25]: print("Coefficients: " + str(lr_model.coefficients))
print("Intercept: " + str(lr_model.intercept))
```

```
Coefficients: [-33993.80847295216,44175.53907306952,166.3422434132397,0.16847942501952068,3045.154741430932,628356.2302689436,4503
8.93845860079,30501.123812705846,102303.4260319803,-20.5419649329964,-2591.5971762995473,20.331834883862022,556023.1578516479,-126
482.23879823307,34.512934314395274,-0.44930466229975813]
Intercept: -37547005.15196078
```

- **Using Regression Evaluator to calculate R2 & RMSE Score for Linear Regression**

- The Test Accurancy using R2 and RMSE in using Linear Regression is calculated using the .pysparl.ml.evaluation by importing RegressionEvaluator

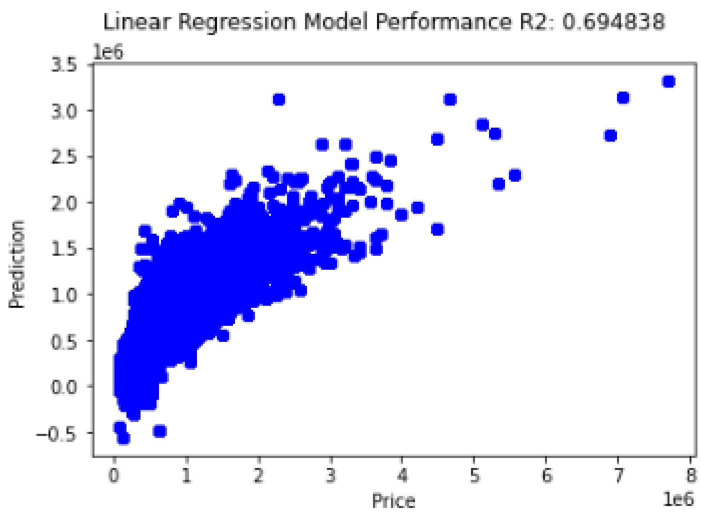
```
[26]: # Evaluate our model
from pyspark.ml.evaluation import RegressionEvaluator, MulticlassClassificationEvaluator

evaluatorLR1 = RegressionEvaluator(labelCol="price", predictionCol="prediction", metricName="rmse")
evaluatorLR2 = RegressionEvaluator(labelCol="price", predictionCol="prediction", metricName="r2")
```

```
[27]: accuracy3 = evaluatorLR1.evaluate(predictionsLR)
accuracy4 = evaluatorLR2.evaluate(predictionsLR)
print('Linear Regression RMSE Test Accuracy = ', accuracy3)
print('Linear Regression R2 Test Accuracy = ', accuracy4)
```

```
Linear Regression RMSE Test Accuracy =  202260.07375045063
Linear Regression R2 Test Accuracy =  0.6948376452815955
```

- **MODEL - PERFORMANCE VISUALIZATION – LINEAR REGRESSION**

[28]:
```
import matplotlib.pyplot as plt

rmse = evaluatorLR1.evaluate(predictionsLR)
r2 = evaluatorLR2.evaluate(predictionsLR)
rfPred = lr_model.transform(test_data)
rfResult = rfPred.toPandas()
plt.plot(rfResult.price, rfResult.prediction, 'bo')
plt.xlabel('Price')
plt.ylabel('Prediction')
plt.suptitle("Linear Regression Model Performance RMSE: %f" % rmse)
plt.suptitle("Linear Regression Model Performance R2: %f" % r2)
plt.show()
```



3) GRADIENT BOOSTING REGRESSION

- Importing important Libraries for GBT Regressor
- Using the pyspark.ml.regression to Import GBT Regressor

GRADIENT BOOSTING REGRESSOR

```
[40]: import pyspark
from pyspark.sql import SparkSession
from pyspark.ml.regression import GBTRegressor
from pyspark.ml.feature import StringIndexer, VectorAssembler
from pyspark.ml.feature import VectorAssembler
from pyspark.ml import Pipeline
from pyspark.ml.evaluation import BinaryClassificationEvaluator
conf = pyspark.SparkConf().setAppName("Gradient Boosted Tree Regressor")
```

- Setting the Parameters with different depth

```
[41]: from pyspark.ml.regression import RandomForestRegressor, GBTRegressor
gbtregressor = GBTRegressor(featuresCol = 'features', labelCol = 'price', maxDepth = 30)

# maxIter=10
```

- Fitting the Model & Calculating the Training Time (Execution Time)

```
[42]: start_timeGB = time.perf_counter()

modelGB = gbtregressor.fit(training_data)
predictionsGB = modelGB.transform(test_data)

end_timeGB = time.perf_counter()

training_timeGB = end_timeGB - start_timeGB
```

```
[43]: print('Gradient Boosting Regression Model Start Time = ', start_timeGB)
print('Gradient Boosting Regression Model End Time = ', end_timeGB)
print('Gradient Boosting Regression Model Training Time = ', training_timeGB)
```

```
Gradient Boosting Regression Model Start Time =  5864.9711885
Gradient Boosting Regression Model End Time =  6647.7953575
Gradient Boosting Regression Model Training Time =  782.8241690000004
```

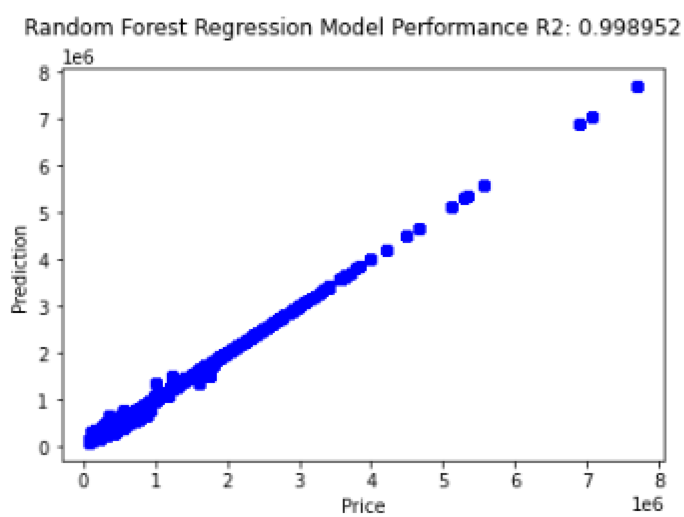
- **Using Regression Evaluator to calculate R2 & RMSE Score for Gradient Boosting Regressor**
    - The Test Accurancy using R2 and RMSE score using Linear Regression is calculated using the .pysparl.ml.evaluation importing RegressionEvaluator

```
[44]: from pyspark.ml.evaluation import RegressionEvaluator, MulticlassClassificationEvaluator

evaluatorGB1 = RegressionEvaluator(labelCol="price", predictionCol="prediction", metricName="rmse")
evaluatorGB2 = RegressionEvaluator(labelCol="price", predictionCol="prediction", metricName="r2")
```

```
[45]: accuracy7 = evaluatorGB1.evaluate(predictionsGB)
accuracy8 = evaluatorGB2.evaluate(predictionsGB)
print('Gradient Boosting Regression RMSE Test Accuracy = ', accuracy7)
print('Gradient Boosting Regression R2 Test Accuracy = ', accuracy8)

Gradient Boosting Regression RMSE Test Accuracy =  11850.855495781418
Gradient Boosting Regression R2 Test Accuracy =  0.9989523649015711
```

- **<u>MODEL - PERFORMANCE VISUALIZATION – GRADIENT BOOSTING REGRESSION</u>**

```
[46]: import matplotlib.pyplot as plt

rmse = evaluatorGB1.evaluate(predictionsGB)
r2 = evaluatorGB2.evaluate(predictionsGB)
rfPred = modelGB.transform(test_data)
rfResult = rfPred.toPandas()
plt.plot(rfResult.price, rfResult.prediction, 'bo')
plt.xlabel('Price')
plt.ylabel('Prediction')
plt.suptitle("Random Forest Regression Model Performance RMSE: %f" % rmse)
plt.suptitle("Random Forest Regression Model Performance R2: %f" % r2)
plt.show()
```



4) DECISION TREE REGRESSION

- Importing important Libraries for DecisionTreeRegressor
- Using the pyspark.ml.regression to Import DecisionTreeRegressor
- Setting the Parameters with different depth

```
[47]: import pyspark
      from pyspark.ml.regression import DecisionTreeRegressionModel, DecisionTreeRegressor
```

```
[54]: DecisionTree = DecisionTreeRegressor(featuresCol = 'features', labelCol = 'price', maxDepth = 10, maxBins=32)
```

- Fitting the Model & Calculating the Training Time (Execution Time)

```
[55]: start_timeDT = time.perf_counter()

      modelDT = DecisionTree.fit(training_data)
      predictionsDT = modelDT.transform(test_data)

      end_timeDT = time.perf_counter()

      training_timeDT = end_timeDT - start_timeDT
```

```
[56]: print('Decision Tree Regression Model Start Time = ', start_timeDT)
      print('Decision Tree Boosting Regression Model End Time = ', end_timeDT)
      print('Decision Tree Regression Model Training Time = ', training_timeDT)

      Decision Tree Regression Model Start Time =  9742.6451198
      Decision Tree Boosting Regression Model End Time =  9781.4423352
      Decision Tree Regression Model Training Time =  38.797215399999914
```

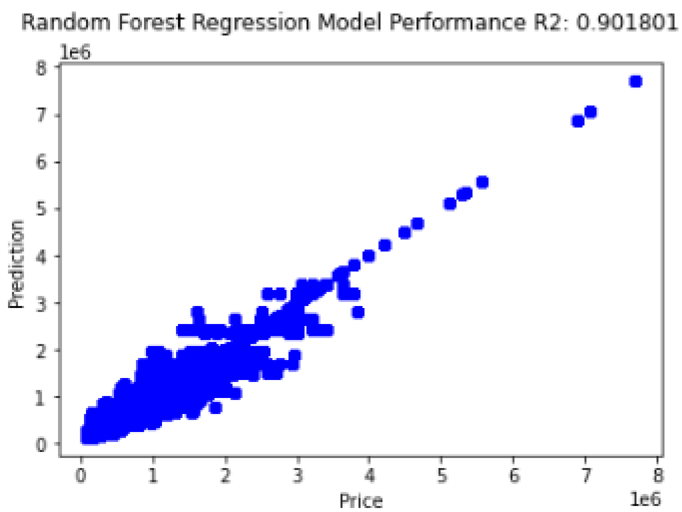
- **Using Regression Evaluator to calculate R2 & RMSE Score for Decision Tree Regression**
  - The Test Accurancy using R2 and RMSE score using Decision Tree Regression is calculated using the .pysparl.ml.evaluation importing RegressionEvaluator

```
[57]: from pyspark.ml.evaluation import RegressionEvaluator, MulticlassClassificationEvaluator

      evaluatorDT1 = RegressionEvaluator(labelCol="price", predictionCol="prediction", metricName="rmse")
      evaluatorDT2 = RegressionEvaluator(labelCol="price", predictionCol="prediction", metricName="r2")
```

```
[58]: accuracy9 = evaluatorDT1.evaluate(predictionsDT)
      accuracy10 = evaluatorDT2.evaluate(predictionsDT)
      print('Decision Tree Regression RMSE Test Accuracy = ', accuracy9)
      print('Decision Tree Regression R2 Test Accuracy = ', accuracy10)

      Decision Tree Regression RMSE Test Accuracy =  114735.3862947077
      Decision Tree Regression R2 Test Accuracy =  0.9018013767034312
```

- **MODEL - PERFORMANCE VISUALIZATION – GRADIENT BOOSTING REGRESSION**

[59]:
```
import matplotlib.pyplot as plt

rmse = evaluatorDT1.evaluate(predictionsDT)
r2 = evaluatorDT2.evaluate(predictionsDT)
rfPred = modelDT.transform(test_data)
rfResult = rfPred.toPandas()
plt.plot(rfResult.price, rfResult.prediction, 'bo')
plt.xlabel('Price')
plt.ylabel('Prediction')
plt.suptitle("Random Forest Regression Model Performance RMSE: %f" % rmse)
plt.suptitle("Random Forest Regression Model Performance R2: %f" % r2)
plt.show()
```



5) **K-MEANS ALGORITHM**

- Importing important Libraries for K-Means Algorithm

```
[63]: from pyspark.ml.clustering import KMeans
```

- Setting the Parameters with different depth

```
[64]: kmeans2 = KMeans(featuresCol = 'features', k=2)
      kmeans3 = KMeans(featuresCol = 'features', k=3)
      kmeans10 = KMeans(featuresCol = 'features', k=10)
```

- Fitting the Model & Calculating the Training Time (Execution Time)

```
[58]: start_timeKM = time.perf_counter()

      model_k2 = kmeans2.fit(transformed_data)
      model_k3 = kmeans3.fit(transformed_data)
      model_k10 = kmeans10.fit(transformed_data)

      end_timeKM = time.perf_counter()

      training_timeKM = end_timeKM - start_timeKM
```

```
[59]: print('Decision Tree Regression Model Start Time = ', start_timeKM)
      print('Decision Tree Boosting Regression Model End Time = ', end_timeKM)
      print('Decision Tree Regression Model Training Time = ', training_timeKM)

      Decision Tree Regression Model Start Time =  16457.578523
      Decision Tree Boosting Regression Model End Time =  16543.7923473
      Decision Tree Regression Model Training Time =  86.21382430000085
```

- The Distance Accuracy using KMeans is calculated using different KMeans Parameters are as below:

```
[66]: # Make predictions
      predictionsK2 = model_k2.transform(training_data)
      from pyspark.ml.evaluation import ClusteringEvaluator
      # Evaluate clustering by computing Silhouette score
      evaluator1 = ClusteringEvaluator()
      silhouette1 = evaluator1.evaluate(predictionsK2)
      print("Silhouette with squared euclidean distance = " + str(silhouette1))

      Silhouette with squared euclidean distance = 0.979522914998058
```

```
[67]: # Make predictions
predictionsK3 = model_k3.transform(training_data)
from pyspark.ml.evaluation import ClusteringEvaluator
# Evaluate clustering by computing Silhouette score
evaluator2 = ClusteringEvaluator()
silhouette2 = evaluator2.evaluate(predictionsK3)
print("Silhouette with squared euclidean distance = " + str(silhouette2))
```

```
Silhouette with squared euclidean distance = 0.9754691847675738
```

```
[68]: # Make predictions
predictionsK10 = model_k10.transform(training_data)
from pyspark.ml.evaluation import ClusteringEvaluator
# Evaluate clustering by computing Silhouette score
evaluator3 = ClusteringEvaluator()
silhouette3 = evaluator3.evaluate(predictionsK10)
print("Silhouette with squared euclidean distance = " + str(silhouette3))
```

```
Silhouette with squared euclidean distance = 0.628720954966732
```

PERFORMANCE COMPARISION OF IMPLEMENTED MACHINE LEARNING ALGORITHM

MACHINE LEARNING ALGORITHMS (PERFORMANCE COMPARISION)			
Name	**Execution Time(sec)**	**RMSE**	**R2**
Linear Regression	51	202260.07	0.69
Random Forest Regression	102	167386.04	0.79
Gradient Boosting Regression	782	11850.85	0.99
Decision Tree Regression	39	114735.38	0.90
K-Means	87	39850.85	0.97

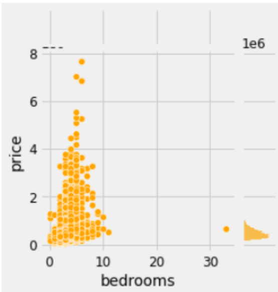
- Here all the noted statistics are visualized in the table for the implemented Machine Learning Algorithms.
- The most Training & Execution time taken was from Gradient Boosting based on its parametric fitting of model of maxDepth of 30. Hence taking observable time.

Data Visualization

```
[70]: display(df.select("bedrooms", "price"))
```

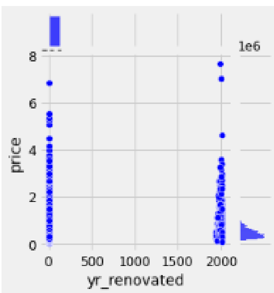
```
DataFrame[bedrooms: string, price: string]
```

```
b]: <seaborn.axisgrid.JointGrid at 0x269b1e9f0d0>
```



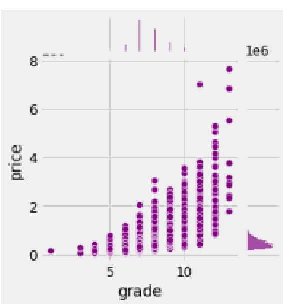

```
[71]: display(df.select("yr_renovated", "price"))
```

```
DataFrame[yr_renovated: string, price: string]
```



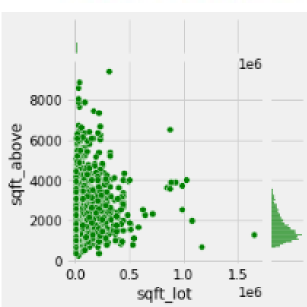

```
[72]: display(df.select("grade", "price"))
```

```
DataFrame[grade: string, price: string]
```




```
[73]: display(df.select("sqft_lot", "sqft_above"))
```

```
DataFrame[sqft_lot: string, sqft_above: string]
```



RUNNING APACHE MAHOUT IN DOCKER

- Pulling the Mahout Image from the following command:
    - It pulls the Container and Image of the Mahout
    - docker pull michabirklbauer/mahout:latest

Administrator: Command Prompt

```
D:\BDAML>docker pull michabirklbauer/mahout:latest
latest: Pulling from michabirklbauer/mahout
Digest: sha256:0234fa0dcc1f86eedff78bdc64dba947641cfcbf52ee8241507cf143d482e2f4
Status: Image is up to date for michabirklbauer/mahout:latest
docker.io/michabirklbauer/mahout:latest
```

```
latest: Pulling from michabirklbauer/mahout
0ecb575e629c: Pull complete
7467d1831b69: Pull complete
feab2c490a3c: Pull complete
f15a0f46f8c3: Pull complete
26cb1dfcbebb: Pull complete
5b224ce6d4ea: Pull complete
c932fe81bb40: Pull complete
0c39e3902a25: Pull complete
18fadcbee53b: Pull complete
ee3d54adf2d0: Pull complete
8e3d56b510ec: Pull complete
eee173c37d0f: Pull complete
2abd8bfd3e61: Pull complete
76866a3d54fd: Pull complete
bfb87913f780: Pull complete
2bfe793a1e2d: Pull complete
912fcec458b0: Pull complete
dec3180c7883: Pull complete
f643bf1aed1b: Pull complete
3c6258235011: Pull complete
536e64712f6c: Pull complete
e47d855c16f2: Pull complete
0aa6c9a8bbf4: Pull complete
8d6493e48cb8: Pull complete
a51c2f69bd7b: Pull complete
Digest: sha256:0234fa0dcc1f86eedff78bdc64dba947641cfcbf52ee8241507cf143d482e2f4
Status: Downloaded newer image for michabirklbauer/mahout:latest
docker.io/michabirklbauer/mahout:latest
```

- The Mahout Shell Session for Spark is run using
    - docker run -it michabirklbauer/mahout:latest

Administrator: Command Prompt - docker run -it michabirklbauer/mahout:latest

```
D:\BDAML>docker run -it michabirklbauer/mahout:latest
Adding lib/ to CLASSPATH
22/06/01 17:52:35 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java cl
asses where applicable
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
Spark context Web UI available at http://83a1c973b712:4040
Spark context available as 'sc' (master = local[*], app id = local-1654105971050).
Spark session available as 'spark'.

                                                    version 0.14

warning: File `spark-shell' does not exist.
Welcome to
                                    version 3.1.1

Using Scala version 2.12.10 (OpenJDK 64-Bit Server VM, Java 1.8.0_282)
Type in expressions to have them evaluated.
Type :help for more information.

scala> import org.apache.spark.ml.regression.LinearRegression
```

- <u>**OPENING MAHOUT SPARK SESSION SHELL**</u>

Administrator: Command Prompt - docker run -it michabirklbauer/mahout:latest

```
D:\BDAML>docker run -it michabirklbauer/mahout:latest
Adding lib/ to CLASSPATH
22/06/01 17:52:35 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java cl
asses where applicable
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
Spark context Web UI available at http://83a1c973b712:4040
Spark context available as 'sc' (master = local[*], app id = local-1654105971050).
Spark session available as 'spark'.

                 _                 _
 _ __ ___   __ _| |__   ___  _   _| |_
| '_ ` _ \ / _` | '_ \ / _ \| | | | __|
| | | | | | (_| | | | | (_) | |_| | |_
|_| |_| |_|\__,_|_| |_|\___/ \__,_|\__|  version 0.14


warning: File `spark-shell' does not exist.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 3.1.1
      /_/

Using Scala version 2.12.10 (OpenJDK 64-Bit Server VM, Java 1.8.0_282)
Type in expressions to have them evaluated.
Type :help for more information.

scala> import org.apache.spark.ml.regression.LinearRegression
```

- Importing Linear Regression in Mahout Spark Shell

```
scala> import org.apache.spark.ml.regression.LinearRegression
import org.apache.spark.ml.regression.LinearRegression
```

- To Load the data set to the Container we use the following command
  - docker cp [datasetname] [container name]:/apache

Administrator: Command Prompt

```
Microsoft Windows [Version 10.0.19042.1706]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Administrator>cd..

C:\Users>cd..

C:\>D:

D:\>cd BDAML

D:\BDAML>docker cp kc.csv quirky_elgamal:/apache

D:\BDAML>docker cp kc.csv loving_wing:/apache

D:\BDAML>docker cp kc_house_data.csv loving_wing:/apache
```

```
D:\BDAML>docker cp Housing.csv xenodochial_jepsen:/apache

D:\BDAML>
```

- **<u>Implementation of Simple Linear Regression</u>**

```
scala> import org.apache.mahout.math.algorithms.regression.OrdinaryLeastSquares
import org.apache.mahout.math.algorithms.regression.OrdinaryLeastSquares
```

  - The data loaded in dataset is paralleized and dense using following command:
  - Val drmDataa = drmParallelize(dense(dfset), numPartitions = 2);

```
     | (3, 3, 2990, 9773, 2, 0, 0, 4, 8, 2990, 0, 1973, 0, 98005, 2230, 11553, 849950),
     | (3, 2, 1350, 6000, 1, 0, 2, 3, 7, 900, 450, 1950, 0, 98136, 1730, 6012, 525000),
     | (4, 3, 4860, 181319, 3, 0, 0, 3, 9, 4860, 0, 1993, 0, 98074, 3850, 181319, 1385000),
     | (4, 3, 2160, 7725, 1, 0, 0, 4, 8, 1460, 700, 1978, 0, 98023, 2060, 8250, 295000),
     | (6, 4, 4860, 11793, 2, 0, 0, 3, 11, 3860, 1000, 1998, 0, 98006, 3600, 11793, 1067000),
     | (2, 2, 890, 5000, 1, 0, 0, 3, 6, 890, 0, 1917, 0, 98118, 1860, 5000, 207950),
     | (4, 3, 2810, 7302, 2, 0, 0, 3, 9, 2810, 0, 2002, 0, 98075, 2820, 7302, 699900),
     | (5, 3, 3400, 9500, 2, 0, 1, 4, 8, 3400, 0, 1977, 0, 98027, 3080, 11081, 1280000),
     | (4, 2, 1580, 7350, 1, 0, 0, 4, 7, 960, 620, 1963, 0, 98052, 1560, 7350, 452000),
     | (3, 2, 1680, 5036, 1, 0, 1, 4, 7, 1680, 0, 1987, 0, 98055, 1680, 4921, 370000),
     | (3, 2, 1300, 11230, 1, 0, 0, 5, 7, 1300, 0, 1968, 0, 98042, 1300, 10794, 232000),
     | (5, 3, 2820, 14062, 2, 0, 0, 4, 7, 2380, 440, 1960, 0, 98034, 1910, 10392, 669950),
     | (3, 2, 1510, 6710, 1, 0, 0, 3, 7, 1070, 440, 1972, 0, 98034, 1660, 6600, 397500),
     | (2, 3, 1230, 1391, 2, 0, 0, 3, 8, 870, 360, 2004, 0, 98112, 1240, 1350, 490000),
     | (4, 2, 2110, 4140, 2, 0, 0, 3, 9, 1710, 400, 1925, 2003, 98116, 1440, 4420, 725000),
     | (3, 3, 3080, 19635, 1, 0, 2, 4, 7, 1610, 1470, 1958, 0, 98032, 2424, 12410, 299000),
     | (2, 2, 1490, 5750, 2, 0, 0, 4, 7, 1190, 300, 1900, 0, 98116, 1590, 4025, 625000),
     | (3, 3, 2120, 4500, 2, 0, 0, 3, 7, 2120, 0, 2000, 0, 98065, 2530, 4816, 437500)),
     | numPartitions = 2);
drmDataa: org.apache.mahout.math.drm.CheckpointedDrm[Int] = org.apache.mahout.sparkbindings.drm.CheckpointedDrmSpark@36a
70b1e
```

  - The data set has the following Data Types (17 Features)

```
scala> dfset.dtypes.foreach(f=>println(f._1+","+f._2))
bedrooms,IntegerType
bathrooms,IntegerType
sqft_living,IntegerType
sqft_lot,IntegerType
floors,IntegerType
waterfront,IntegerType
view,IntegerType
condition,IntegerType
grade,IntegerType
sqft_above,IntegerType
sqft_basement,IntegerType
yr_built,IntegerType
yr_renovated,IntegerType
zipcode,IntegerType
sqft_living15,IntegerType
sqft_lot15,IntegerType
price,IntegerType
```

  - The X (Independent Variable) – "**drmX**" & Y (Dependent Variable) – "**y**"
  - The First 16 Columns are put in "**drmX**" & (Taking all rows is denoted by **::**)
  - The Last Column **"Price"** is the Target Variable are put in – **"y"**

```
scala> val drmX = drmDataa(::, 0 until 16)
drmX: org.apache.mahout.math.drm.DrmLike[Int] = OpMapBlock(org.apache.mahout.sparkbindings.drm.CheckpointedDr
mSpark@36a70b1e,org.apache.mahout.math.drm.DrmLikeOps$$Lambda$2106/534559930@7130314b,16,-1,true)

scala> val y = drmDataa.collect(::, 16)
y: org.apache.mahout.math.Vector = {0:221900.0,1:538000.0,2:180000.0,3:604000.0,4:510000.0,5:1225000.0,6:2575
00.0,7:291850.0,8:229500.0,9:323000.0,10:662500.0,11:468000.0,12:310000.0,13:400000.0,14:530000.0,15:650000.0
,16:395000.0,17:485000.0,18:189000.0,19:230000.0,20:385000.0,21:2000000.0,22:285000.0,23:252700.0,24:329000.0
,25:233000.0,26:937000.0,27:667000.0,28:438000.0,29:719000.0,30:580500.0,31:280000.0,32:687500.0,33:535000.0,
34:322500.0,35:696000.0,36:550000.0,37:640000.0,38:240000.0,39:605000.0,40:625000.0,41:775000.0,42:861990.0,4
3:685000.0,44:309000.0,45:488000.0,46:210490.0,47:785000.0,48:450000.0,49:1350000.0,50:228000.0,51:345000.0,5
2:600000.0,53:585000.0,54:920000.0,55:885000.0,56:292500.0,57:301000.0,58:951000.0,59:430000.0,60:650000.0,61
:289000....
```

- Mahout’s DSL automatically optimizes and parallelizes all operations on DRMs and runs them on Apache Spark
- We extract the target variable vector y, the Last column “Price” of the data matrix. We assume this one fits into our driver machine, so we fetch it into memory using **collect:**
- A simple textbook approach is ordinary least squares (OLS), which minimizes the sum of residual squares between the true target variable and the prediction of the target variable.
- Mahout’s scala DSL maps directly to the mathematical formula. The operation **.t()** transposes a matrix and analogous to R **%*%** denotes matrix multiplication.
- X lives in the cluster, while is y in the memory of the driver, and the result is a DRM again.

```
scala> val drmXtX = drmX.t %*% drmX
drmXtX: org.apache.mahout.math.drm.DrmLike[Int] = OpABAnyKey(OpAt(OpMapBlock(org.apache.mahout.sparkbindings.
drm.CheckpointedDrmSpark@36a70b1e,org.apache.mahout.math.drm.DrmLikeOps$$Lambda$2106/534559930@7130314b,16,-1
,true)),OpMapBlock(org.apache.mahout.sparkbindings.drm.CheckpointedDrmSpark@36a70b1e,org.apache.mahout.math.d
rm.DrmLikeOps$$Lambda$2106/534559930@7130314b,16,-1,true))

scala> val drmXty = drmX.t %*% y
drmXty: org.apache.mahout.math.drm.DrmLike[Int] = OpAx(OpAt(OpMapBlock(org.apache.mahout.sparkbindings.drm.Ch
eckpointedDrmSpark@36a70b1e,org.apache.mahout.math.drm.DrmLikeOps$$Lambda$2106/534559930@7130314b,16,-1,true)
),{0:221900.0,1:538000.0,2:180000.0,3:604000.0,4:510000.0,5:1225000.0,6:257500.0,7:291850.0,8:229500.0,9:3230
00.0,10:662500.0,11:468000.0,12:310000.0,13:400000.0,14:530000.0,15:650000.0,16:395000.0,17:485000.0,18:18900
0.0,19:230000.0,20:385000.0,21:2000000.0,22:285000.0,23:252700.0,24:329000.0,25:233000.0,26:937000.0,27:66700
0.0,28:438000.0,29:719000.0,30:580500.0,31:280000.0,32:687500.0,33:535000.0,34:322500.0,35:696000.0,36:550000
.0,37:640000.0,38:240000.0,39:605000.0,40:625000.0,41:775000.0,42:861990.0,43:685000.0,44:309000.0,45:488000.
0,46:210...
```

- Mahout provides the an analog to R’s **solve()** for that which computes beta, our OLS estimate of the parameter vector.
- We have a implemented a distributed linear regression algorithm.

```
scala> val XtX = drmXtX.collect
XtX: org.apache.mahout.math.Matrix =
{
 0 =>   {0:6019.0,1:3976.0,2:3753128.0,3:2.6039296E7,4:2556.0,5:17.0,6:427.0,7:5811.0,8:13089.0,9:3133389.0,1
0:619739.0,11:3315038.0,12:181717.0,13:1.65052628E8,14:3523039.0,15:2.3570396E7}
 1 =>   {0:3976.0,1:2923.0,2:2648371.0,3:1.8962908E7,4:1788.0,5:17.0,6:331.0,7:3887.0,8:9012.0,9:2222172.0,10
:426199.0,11:2240569.0,12:131904.0,13:1.11307132E8,14:2476755.0,15:1.7258036E7}
 2 =>   {0:3753128.0,1:2648371.0,2:2.618114989E9,3:1.899731786E10,4:1641043.0,5:15133.0,6:349676.0,7:3611490.
0,8:8457289.0,9:2.160327625E9,10:4.57787364E8,11:2.071153767E9,12:1.2834833E8,13:1.02973299606E11,14:2.379513
964E9,15:1.7145863025E10}
 3 =>   {0:2.6039296E7,1:1.8962908E7,2:1.899731786E10,3:5.3985417637E11,4:1.1340383E7,5:138855.0,6:1871939.0,
7:2.5973...

scala> val Xty = drmXty.collect(::, 0)
Xty: org.apache.mahout.math.Vector = {0:9.31483282E8,1:6.61958168E8,2:6.60549652302E11,3:4.820186782243E12,4:
4.12493232E8,5:8554900.0,6:1.269342E8,7:9.05879666E8,8:2.145029272E9,9:5.44156030242E11,10:1.1639362206E11,11
:5.17516043814E11,12:4.348950845E10,13:2.5767532164145E13,14:6.114915231E11,15:4.286763928005E12}

scala>

scala> val beta = solve(XtX, Xty)
beta: org.apache.mahout.math.Vector = {0:-214354.15394986846,1:-2.6616069390283264E7,2:1.25144580525106534E18
,3:91.2375906894262,4:4336734.264176747,5:726140.8028051739,6:3143933.0633249683,7:1.5837423773057139E7,8:-12
62633.3706173836,9:-1.2514458052510656E18,10:-1.2514458052510505E18,11:1496606.7965678072,12:32797.6853432466
7,13:-29937.71043420865,14:-3349.6674021020667,15:-150.77902673458686}
```

```
scala> beta
res15: org.apache.mahout.math.Vector = {0:-214354.15394986846,1:-2.6616069390283264E7,2:1.25144580525106534E1
8,3:91.2375906894262,4:4336734.264176747,5:726140.8028051739,6:3143933.0633249683,7:1.5837423773057139E7,8:-1
262633.3706173836,9:-1.2514458052510656E18,10:-1.2514458052510505E18,11:1496606.7965678072,12:32797.685343246
67,13:-29937.71043420865,14:-3349.6674021020667,15:-150.77902673458686}
```

- To check how well our model fits its training data. First, we multiply the **feature matrix (drmX)** by **estimate beta**. Then, we look at the difference (via L2-norm) of the target variable to the fitted target variable

```
scala> val yFitted = (drmX %*% beta).collect(::, 0)
yFitted: org.apache.mahout.math.Vector = {0:-2707919.5395412953,1:4.033626037069394E7,2:-3.4293824526286095E7
,3:5026865.283709168,4:1.868951529428063E7,5:-3.8145732302340515E7,6:3.6666805639817566E7,7:-1.94610738979032
04E7,8:1.489794677742044E7,9:2.0492911652965095E7,10:-1.5128499041997582E7,11:543439.8770963444,12:-2.1162838
736631505E7,13:2.2057451044189245E7,14:-1.0564861403659609E8,15:4549191.99309601,16:3.5557853518771216E7,17:-
4.0132458835595556E7,18:-3.078685235399956E7,19:3.9455090096970424E7,20:-1.2778175038757985E7,21:-1.971153063
4685867E7,22:6442485.1773958765,23:2.039609417286256E7,24:3.595497901846584E7,25:-6986878.580153577,26:-7.526
890275698993E7,27:-3.6969306663267076E7,28:-2.7980021221178655E7,29:2.1447303940795857E7,30:1.967851036731929
E7,31:5....

scala> (y - yFitted).norm(2)
res11: Double = 6.744315029068638E8
```

- Putting all the commands for ordinary least squares into a function ols

```
scala> def ols(drmX: DrmLike[Int], y: Vector) =
     |   solve(drmX.t %*% drmX, drmX.t %*% y)(::, 0)
ols: (drmX: org.apache.mahout.math.drm.DrmLike[Int], y: org.apache.mahout.math.Vector)org.apache.mahout.math.
Vector
```

- A function named goodnessOfFit is defined that tells how well a model fits the target variable:

```
scala> def goodnessOfFit(drmX: DrmLike[Int], beta: Vector, y: Vector) = {
     |   val fittedY = (drmX %*% beta).collect(::, 0)
     |   (y - fittedY).norm(2)
     | }
goodnessOfFit: (drmX: org.apache.mahout.math.drm.DrmLike[Int], beta: org.apache.mahout.math.Vector, y: org.ap
ache.mahout.math.Vector)Double
```

- Adding a bias column

```
scala> val drmXwithBiasColumn = drmX cbind 1
drmXwithBiasColumn: org.apache.mahout.math.drm.DrmLike[Int] = OpCbindScalar(OpMapBlock(org.apache.mahout.spar
kbindings.drm.CheckpointedDrmSpark@36a70b1e,org.apache.mahout.math.drm.DrmLikeOps$$Lambda$2106/534559930@7130
314b,16,-1,true),1.0,false)
```

- The newly created DRM **drmXwithBiasColumn** to our model fitting method **ols** and see how well the resulting model fits the training data with **goodnessOfFit**

```
scala> val betaWithBiasTerm = ols(drmXwithBiasColumn, y)
betaWithBiasTerm: org.apache.mahout.math.Vector = {0:1.8360252330403206E8,1:-1.74006131067264E8,2:2.262911912
13107738E18,3:2778.274306301623,4:-2.7229047690598434E8,5:-2.363028447574347E8,6:-1.2839202535793391E8,7:2.47
41737426158023E8,8:-2.095971394656018E8,9:-2.26291191213104973E18,10:-2.2629119121311337E18,11:2.806802934624
564E7,12:230306.50186903434,13:3.1859784855040707E7,14:748139.0861133145,15:-1854.249458800011,16:-3.18045023
08055356E12}

scala> goodnessOfFit(drmXwithBiasColumn, betaWithBiasTerm, y)
res12: Double = 3.2193108163740654E10

scala> val cachedDrmX = drmXwithBiasColumn.checkpoint()
cachedDrmX: org.apache.mahout.math.drm.CheckpointedDrm[Int] = org.apache.mahout.sparkbindings.drm.Checkpointe
dDrmSpark@830ee01
```

- As a further optimization, we can make use of the DSL's caching functionality. We use **drmXwithBiasColumn** repeatedly as input to a computation, so it might be beneficial to cache it in memory. This is achieved by calling **checkpoint()**.

```
scala> val betaWithBiasTerm = ols(cachedDrmX, y)
betaWithBiasTerm: org.apache.mahout.math.Vector = {0:1.8360252330403206E8,1:-1.74006131067264E8,2:2.262911912
13107738E18,3:2778.274306301623,4:-2.7229047690598434E8,5:-2.363028447574347E8,6:-1.2839202535793391E8,7:2.47
41737426158023E8,8:-2.095971394656018E8,9:-2.262911912131O4973E18,10:-2.2629119121311337E18,11:2.806802934624
564E7,12:230306.50186903434,13:3.1859784855040707E7,14:748139.0861133145,15:-1854.249458800011,16:-3.18045023
08055356E12}

scala> val goodness = goodnessOfFit(cachedDrmX, betaWithBiasTerm, y)
goodness: Double = 3.2193108163740654E10
```

- In the end, we remove it from the cache with **uncache**:

```
scala> cachedDrmX.uncache()
res13: cachedDrmX.type = org.apache.mahout.sparkbindings.drm.CheckpointedDrmSpark@830ee01

scala>

scala> goodness
res14: Double = 3.2193108163740654E10
```

- <u>**Implementing OrdinaryLeastSquares from spark.mahout**</u>

- As per the parameters of **OrdinaryLeastSquares**
    - **'calcCommonStatistics** when set to **false** it only calculates the common statistics

```
scala> val model = new OrdinaryLeastSquares[Int]().fit(drmX, drmY, 'calcCommonStatistics → false)
model: org.apache.mahout.math.algorithms.regression.OrdinaryLeastSquaresModel[Int] = org.apache.mahout.math.algorithms.r
egression.OrdinaryLeastSquaresModel@46cea1c5
```

```
scala> println(model.summary)

Coef.           Estimate                Std. Error              t-score                 Pr(Beta=0)
X0              +4422.12909             +660392.36171           +0.00670                +0.99466
X1              -3312332793.14834               +1270586673.13633               -2.60693                +0.00989
X2              +2022751767.73267               +1929326610.52863               +1.04842                +0.29583
X3              +1568358.13303          +2930828.75867          +0.53512                +0.59321
X4              -16643.95136              NaN             NaN             NaN
X5              +1749901188.93858               +2136603258.62353               +0.81901                +0.41385
X6              -5253029249.07306               +13204452096.63043              -0.39782                +0.69122
X7              +1002633833.88555               +1602760581.87386               +0.62557                +0.53238
X8              -1047410012.90935               +1496060504.98083               -0.70011                +0.48475
X9              +16782294684421080000000.00000          NaN             NaN             NaN
X10             -16782294684412230000000.00000          +30081566427796100.00000                -55789.29782            +
0.00000
X11             -720328.31316           +2920673.67752          -0.24663                +0.80547
X12             -184874351.56143                +49260689.86475         -3.75298                +0.00023
X13             -7854543.91283          +2681985.20573          -2.92863                +0.00384
X14             -255320157.81677                NaN             NaN             NaN
X15             -8394925.09253          +1921486.22102          -4.36897                +0.00002
X16             +25416684293904.29300           +1878846369199.23830            +13.52781               +0.00000
```

    - **'calcCommonStatistics** when set to **True** it only calculates the common statistics
    - Calculates F-Stats / P-Value / Mean Squared Error / R Squared Error

```
scala> val model = new OrdinaryLeastSquares[Int]().fit(drmX, drmY, 'calcCommonStatistics → true)
model: org.apache.mahout.math.algorithms.regression.OrdinaryLeastSquaresModel[Int] = org.apache.mahout.math.a
lgorithms.regression.OrdinaryLeastSquaresModel@1af3c99b

scala> println(model.summary)

Coef.           Estimate                Std. Error              t-score                 Pr(Beta=0)
X0              +184899381.26605                +98411649.51270         +1.87884                +0.06087
X1              -173593418.34597                +134804747.49437                -1.28774                +0.19
845
X2              +2262911912131076610.00000              +71365343932.50372              +31708834.95316
   +0.00000
X3              +2779.16523             +5836.27158             +0.47619                +0.63416
X4              -273562321.82695                +167068341.80751                -1.63743                +0.10
219
X5              -228874811.52999                +694909224.22761                -0.32936                +0.74
203
X6              -129404177.01311                +104470683.10746                -1.23866                +0.21
607
X7              +246248115.33626                +103419120.27011                +2.38107                +0.01
765
X8              -209253430.94342                +104384421.09205                -2.00464                +0.04
556
X9              -2262911912131049220.00000              NaN             NaN             NaN
X10             -2262911912131133180.00000              +24893050060.40104              -90905369.43606
   +0.00000
X11             +28068029.34625         +3563902.38572          +7.87564                +0.00000
X12             +230306.50187           +168625.66224           +1.36579                +0.17264
X13             +31859784.85504         +2163154.60301          +14.72839               +0.00000
X14             +748139.08611           +191853.93241           +3.89952                +0.00011
X15             -1854.24946             +3523.55761             -0.52624                +0.59896
X16             -3180450230805.53470            +144745742440.76355             -21.97267               +0.00
000
F-statistic: -30.18749821887496 on 16 and 483 DF,  p-value: 0.009545

Mean Squared Error: 2.0728644686462392 3E18
R^2: [redacted].6948556386865076E7
```

- **Reading Using .Spark ReadData and Creating Pipeline and applying Machine Learning Algorithm**

```
scala> val training = spark.read.option("inferSchema","true").option("header","true").csv("kc_house_data.csv")
training: org.apache.spark.sql.DataFrame = [id: bigint, date: string ... 19 more fields]
```

```
scala> import org.apache.spark.ml.feature.VectorAssembler
import org.apache.spark.ml.feature.VectorAssembler

scala> import org.apache.spark.ml.linalg.Vectors
import org.apache.spark.ml.linalg.Vectors
```

```
scala> val assembler = new VectorAssembler().setInputCols(Array("bedrooms", "bathrooms", "sqft_living", "sqft_lot", "floors"
, "waterfront", "view", "condition", "grade", "sqft_above", "sqft_basement", "yr_built", "yr_renovated", "lat", "long", "sqf
t_living15", "sqft_lot15")).setOutputCol("features")
assembler: org.apache.spark.ml.feature.VectorAssembler = VectorAssembler: uid=vecAssembler_eaaab29093f4, handleInvalid=error
, numInputCols=17
```

```
scala> val lr = new LinearRegression()
lr: org.apache.spark.ml.regression.LinearRegression = linReg_9896b3c56f5b

scala> .setMaxIter(10)
res1: lr.type = linReg_9896b3c56f5b

scala> .setRegParam(0.3)
res2: res1.type = linReg_9896b3c56f5b

scala> .setElasticNetParam(0.8)
res3: res2.type = linReg_9896b3c56f5b
```

```
scala> .setFeaturesCol("features")
res4: org.apache.spark.ml.regression.LinearRegression = linReg_9896b3c56f5b

scala> .setLabelCol("price")
res5: org.apache.spark.ml.regression.LinearRegression = linReg_9896b3c56f5b
```

```
scala> import org.apache.spark.ml.Pipeline
import org.apache.spark.ml.Pipeline

scala> val pipeline = new Pipeline().setStages(Array(assembler,lr))
pipeline: org.apache.spark.ml.Pipeline = pipeline_e82cba068cb0
```

```
scala> val lrModel = pipeline.fit(training)
22/06/01 17:54:58 WARN BLAS: Failed to load implementation from: com.github.fommil.netlib.NativeSystemBLAS
22/06/01 17:54:58 WARN BLAS: Failed to load implementation from: com.github.fommil.netlib.NativeRefBLAS
lrModel: org.apache.spark.ml.PipelineModel = pipeline_e82cba068cb0
```

CONCLUSION

The main difference lies in their framework. For Mahout, it is Hadoop MapReduce and in the case of MLib, Spark is the framework. Mahout has proven capabilities that Spark's MlLib lacks. Apache Mahout is mature and comes with many ML algorithms to choose from and it is built atop MapReduce.

MLlib was built on top of Spark to take advantage of Spark's efficiency when running iterative Machine Learning algorithms. Its algorithms end up being much faster than Mahout equivalents

We have compared the performance of Spark Mlib and Apache Mahout on different Machine Learning Algorithms and record their performance based on execution time and measuring statistics and error