Experiment 9

Aim:

1. Case study on Power BI and Apache Spark.
2. Data Visualization using Power BI, Apache Spark.

About Dataset: This is the dataset used in the second chapter of Aurélien Géron's recent book 'Hands-On Machine learning with Scikit-Learn and TensorFlow'. It serves as an excellent introduction to implementing machine learning algorithms because it requires rudimentary data cleaning, has an easily understandable list of variables and sits at an optimal size between being too toyish and too cumbersome. The data contains information from the 1990 California census. So although it may not help you with predicting current housing prices like the Zillow Zestimate dataset, it does provide an accessible introduction dataset for teaching people about the basics of machine learning.
Link:- https://www.kaggle.com/datasets/camnugent/california-housing-prices

Theory:

Power Bi:

Microsoft Power BI is a business intelligence (BI) platform that provides nontechnical business users with tools for aggregating, analyzing, visualizing and sharing data. Power BI's user interface is fairly intuitive for users familiar with Excel, and its deep integration with other Microsoft products makes it a versatile self-service tool that requires little upfront training. Users can download an application for Windows 10, called Power BI Desktop, and native mobile apps for Windows, Android and iOS devices. There is also Power BI Report Server for companies that must maintain their data and reports on premises. That version of Power BI requires a special version of the desktop app -- aptly called Power BI Desktop for Power BI Report Server.

In the Power BI service, you can use the deployment pipeline tool to test your content before you release it to your users. The deployment pipeline tool can help you deploy reports, dashboards, datasets, and paginated reports. Read about how to get started with deployment pipelines in the Power BI service. Apart from these hundreds of data sources from files (like Excel, PDF, SharePoint Folder, XML), databases (like SQL Server Database, Oracle Database, IBM databases, Amazon Redshift, Google BigQuery), other Power BI databases, Azure data connections, various online services (like Dynamics 365, Salesforce Reports, Google Analytics, Adobe Analytics, Facebook and others) can be connected into Power BI.

Power BI components

Microsoft Power BI works by connecting data sources and providing a dashboard of BI to the users. It can connect with just an Excel spreadsheet or bring together cloud-based and on-premises data warehouses. Data pulled from cloud-based sources, such as Salesforce CRM, is automatically refreshed. Included within Power BI are several components that help users create and share data reports. Those are the following:

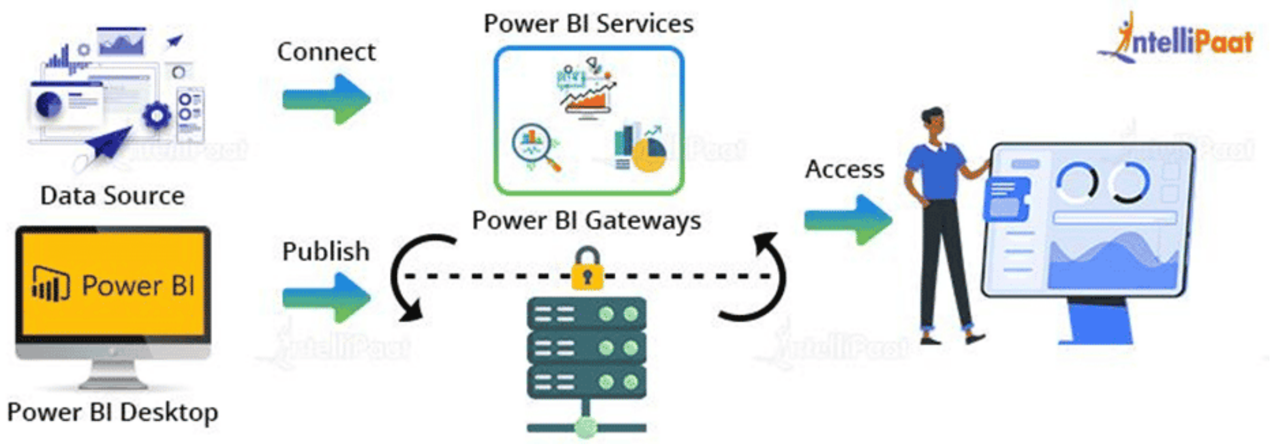
- Power Query: a data mashup and transformation tool
- Power Pivot: a memory tabular data modeling tool
- Power View: a data visualization tool
- Power Map: a 3D geospatial data visualization tool
- Power Q&A: a natural language question and answering engine

Key features of Power BI

Microsoft has added a number of data analytics features to Power BI since its inception, and continues to do so. Some of the most important features are the following:

- Artificial intelligence. Users can access image recognition and text analytics in Power BI, create machine learning models using automated ML capabilities and integrate with Azure Machine Learning.
- Hybrid deployment support. This feature provides built-in connectors that allow Power BI tools to connect with a number of different data sources from Microsoft, Salesforce and other vendors.
- Quick Insights. This feature allows users to create subsets of data and automatically apply analytics to that information.
- Common data model support. Power BI's support for the common data model allows the use of a standardized and extensible collection of data schemas (entities, attributes and relationships).
- Cortana integration. This feature, which is especially popular on mobile devices, allows users to verbally query data using natural language and access results using Cortana, Microsoft's digital assistant.
- Customization. This feature allows developers to change the appearance of default visualization and reporting tools and import new tools into the platform.

Power BI Architecture



Apache Spark:-

Apache Spark is a lightning-fast, open source data-processing engine for machine learning and AI applications, backed by the largest open source community in big data. Apache Spark (Spark) is an open source data-processing engine for large data sets. It is designed to deliver the computational speed, scalability, and programmability required for Big Data—specifically for streaming data, graph data, machine learning, and artificial intelligence (AI) applications.

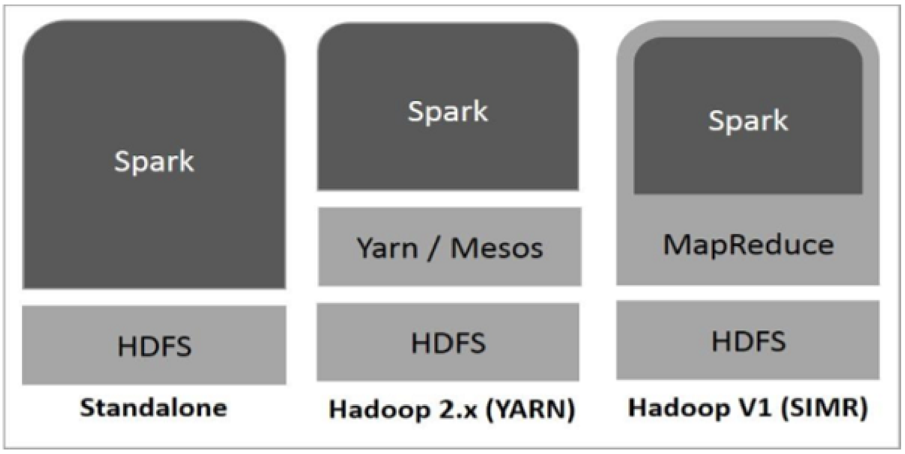
Spark's analytics engine processes data 10 to 100 times faster than alternatives. It scales by distributing processing work across large clusters of computers, with built-in parallelism and fault tolerance. It even includes APIs for programming languages that are popular among data analysts and data scientists, including Scala, Java, Python, and R. Spark is often compared to Apache Hadoop, and specifically to MapReduce, Hadoop's native data-processing component. The chief difference between Spark and MapReduce is that Spark processes and keeps the data in memory for subsequent steps—without writing to or reading from disk—which results in dramatically faster processing speeds.

Features of Apache Spark

- **Fast -** It provides high performance for both batch and streaming data, using a state-of-the-art DAG scheduler, a query optimizer, and a physical execution engine.
- **Easy to Use -** It facilitates writing applications in Java, Scala, Python, R, and SQL. It also provides more than 80 high-level operators.
- **Generality -** It provides a collection of libraries including SQL and DataFrames, MLlib for machine learning, GraphX, and Spark Streaming.
- **Lightweight -** It is a light unified analytics engine which is used for large scale data processing.
- **Runs Everywhere -** It can easily run on Hadoop, Apache Mesos, Kubernetes, standalone, or in the cloud.

Spark Built on Hadoop

The following diagram shows three ways of how Spark can be built with Hadoop components.



There are three ways of Spark deployment as explained below.

- **Standalone −** Spark Standalone deployment means Spark occupies the place on top of HDFS(Hadoop Distributed File System) and space is allocated for HDFS, explicitly. Here, Spark and MapReduce will run side by side to cover all spark jobs on cluster.
- **Hadoop Yarn −** Hadoop Yarn deployment means, simply, spark runs on Yarn without any pre-installation or root access required. It helps to integrate Spark into Hadoop ecosystem or Hadoop stack. It allows other components to run on top of stack.
- **Spark in MapReduce (SIMR) −** Spark in MapReduce is used to launch spark job in addition to standalone deployment. With SIMR, user can start Spark and uses its shell without any administrative access.

Usage of Spark

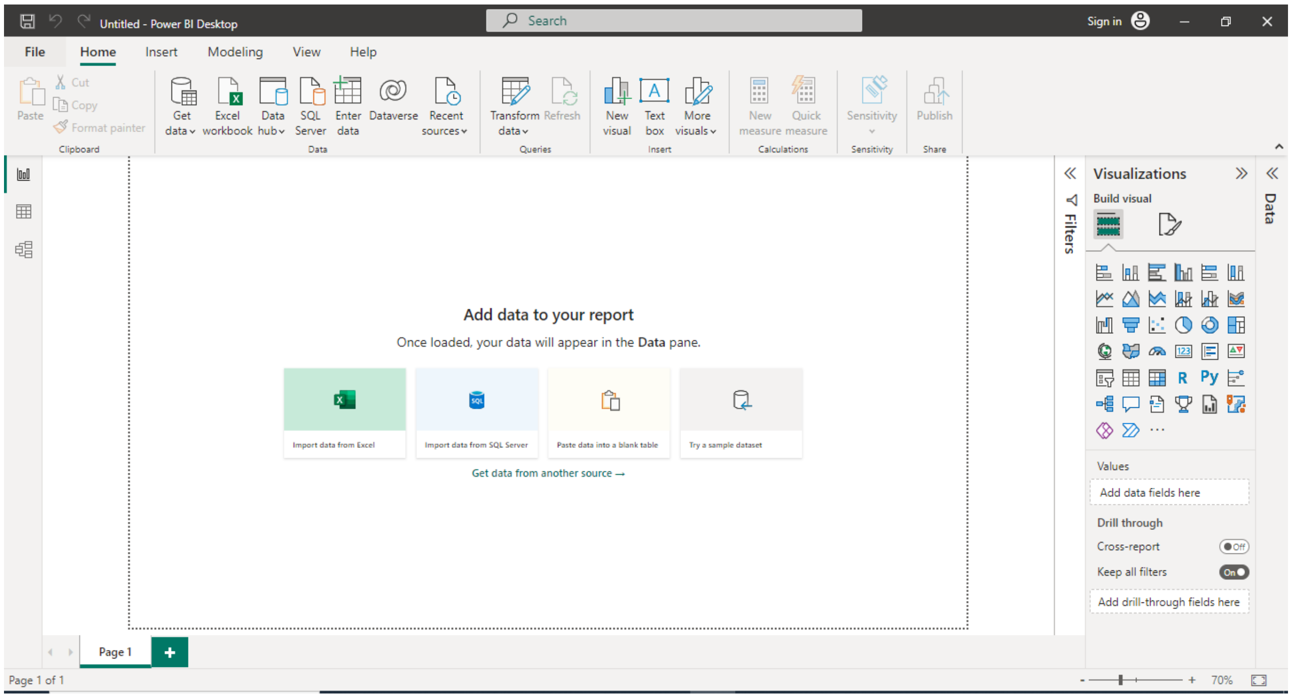
- **Data integration:** The data generated by systems are not consistent enough to combine for analysis. To fetch consistent data from systems we can use processes like Extract, transform, and load (ETL). Spark is used to reduce the cost and time required for this ETL process.
- **Stream processing:** It is always difficult to handle the real-time generated data such as log files. Spark is capable enough to operate streams of data and refuses potentially fraudulent operations.
- **Machine learning:** Machine learning approaches become more feasible and increasingly accurate due to enhancement in the volume of data. As spark is capable of storing data in memory and can run repeated queries quickly, it makes it easy to work on machine learning algorithms.
- **Interactive analytics:** Spark is able to generate responses rapidly. So, instead of running pre-defined queries, we can handle the data interactively.

What are the benefits of Apache Spark?

- **Fast:-** Through in-memory caching, and optimized query execution, Spark can run fast analytic queries against data of any size.
- **Developer friendly:-** Apache Spark natively supports Java, Scala, R, and Python, giving you a variety of languages for building your applications. These APIs make it easy for your developers, because they hide the complexity of distributed processing behind simple, high-level operators that dramatically lowers the amount of code required.
- **Multiple workloads:-** Apache Spark comes with the ability to run multiple workloads, including interactive queries, real-time analytics, machine learning, and graph processing. One application can combine multiple workloads seamlessly.

Data Visualization using Power Bi:

Importing dataset into Power Bi workspace.

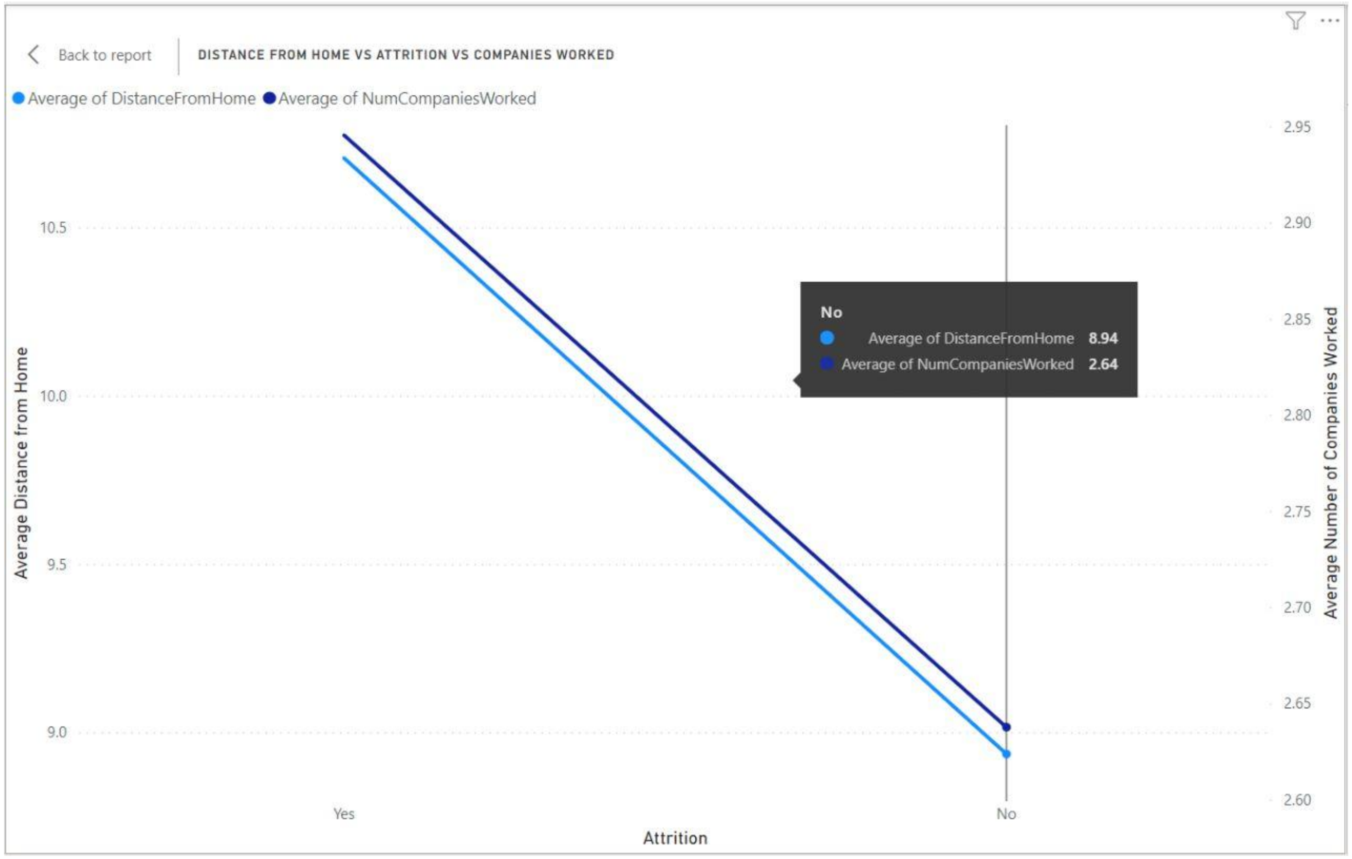


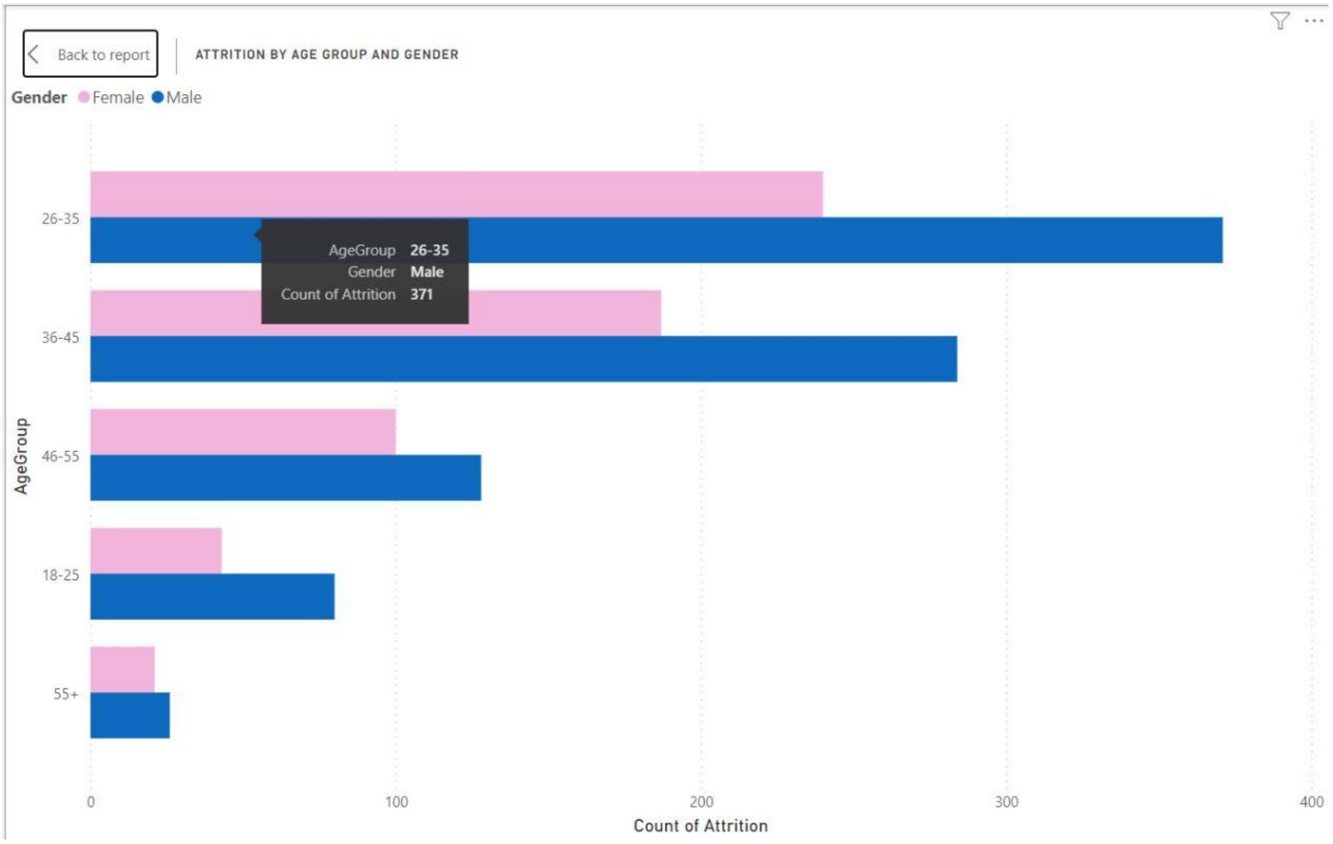
Add data to your report

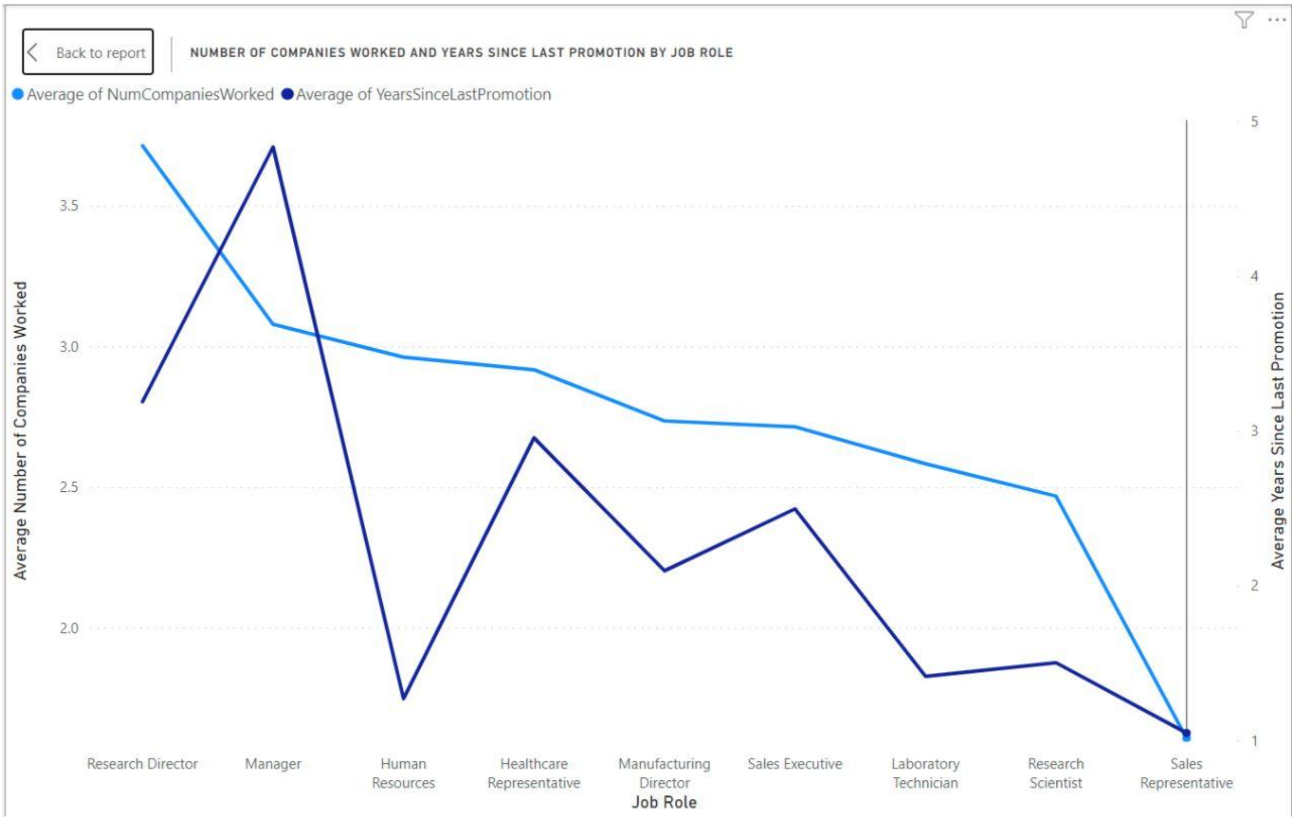
Once loaded, your data will appear in the **Data** pane.

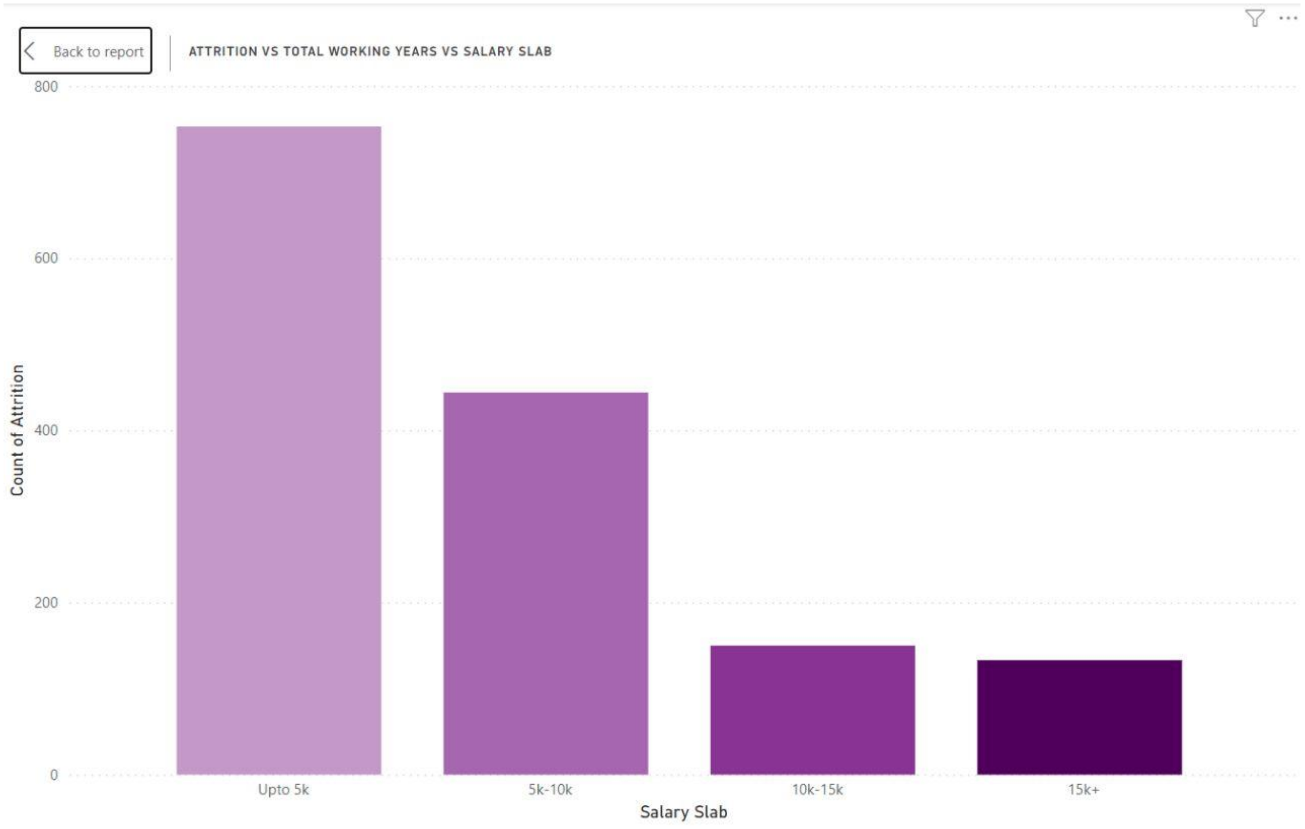
Import data from Excel | Import data from SQL Server | Paste data into a blank table | Try a sample dataset

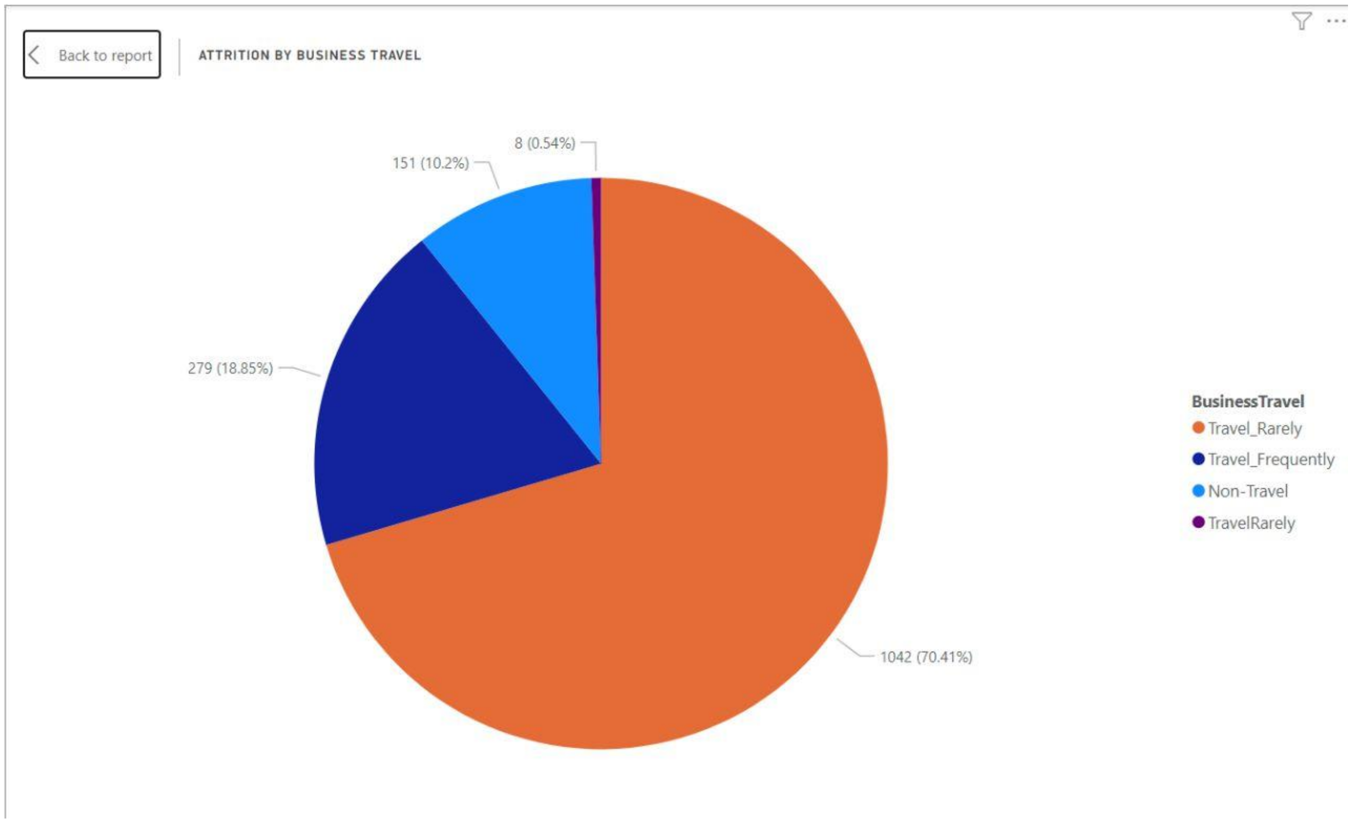
Get data from another source →

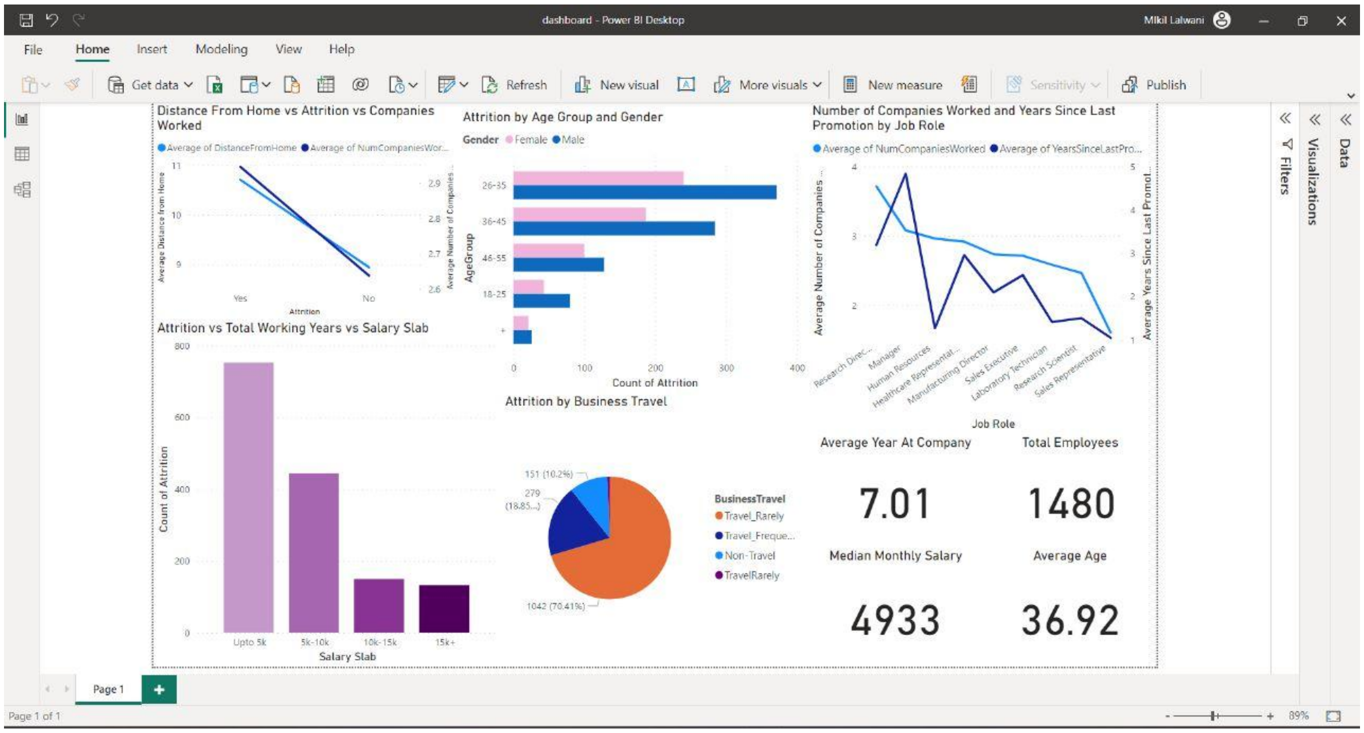



Back to report
ATTRITION BY AGE GROUP AND GENDER
Gender ● Female ● Male
AgeGroup 26-35
Gender Male
Count of Attrition 371
26-35
36-45
46-55
18-25
55+
AgeGroup
0
100
200
300
400
Count of Attrition

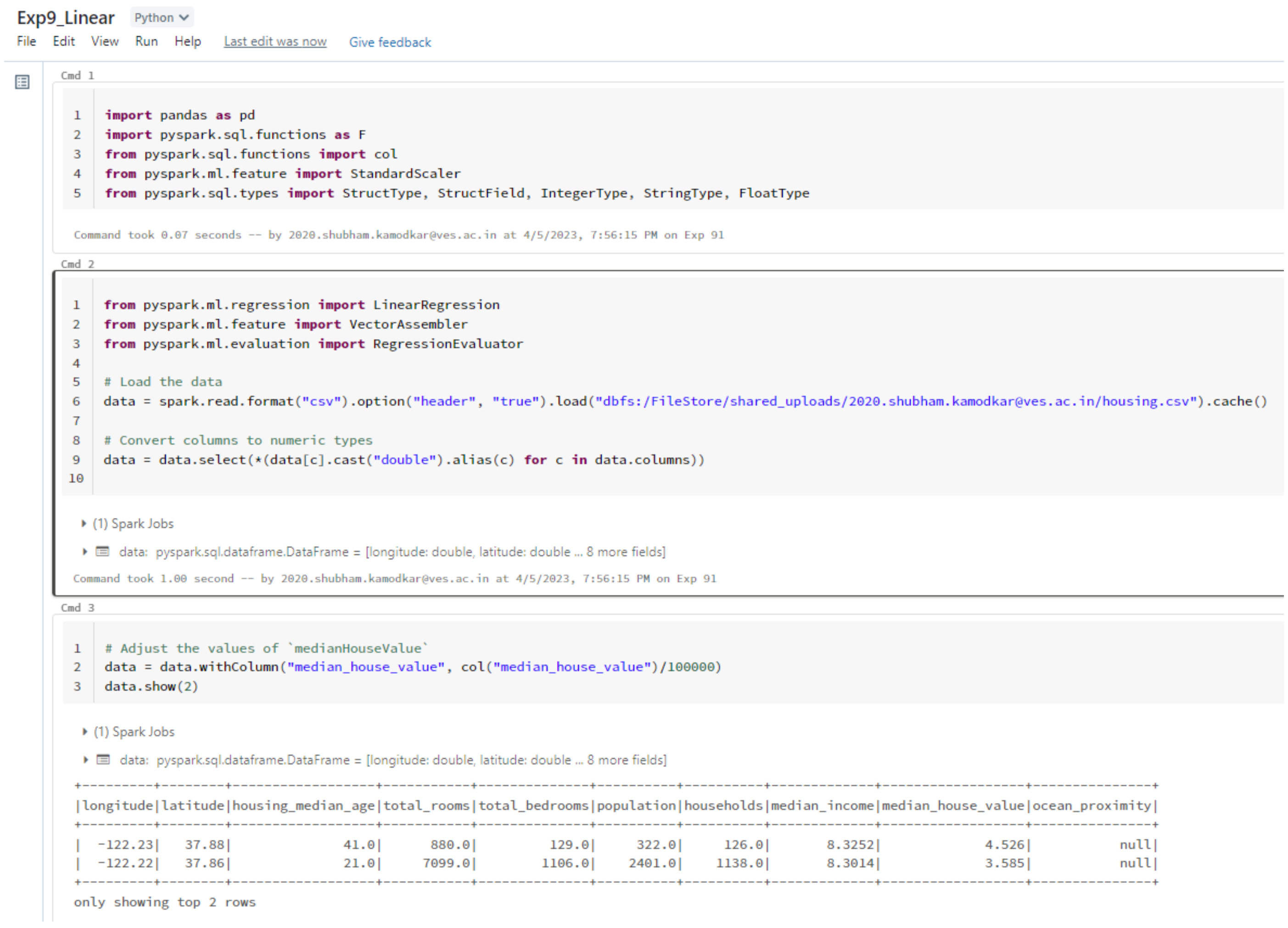

Back to report
NUMBER OF COMPANIES WORKED AND YEARS SINCE LAST PROMOTION BY JOB ROLE
● Average of NumCompaniesWorked ● Average of YearsSinceLastPromotion
Average Number of Companies Worked
3.5
3.0
2.5
2.0
5
4
3
2
1
Average Years Since Last Promotion
Research Director
Manager
Human Resources
Healthcare Representative
Manufacturing Director
Sales Executive
Laboratory Technician
Research Scientist
Sales Representative
Job Role


Back to report
ATTRITION VS TOTAL WORKING YEARS VS SALARY SLAB
800
600
400
200
0
Count of Attrition
Upto 5k
5k-10k
10k-15k
15k+
Salary Slab


Back to report
ATTRITION BY BUSINESS TRAVEL
8 (0.54%)
151 (10.2%)
279 (18.85%)
1042 (70.41%)
BusinessTravel
● Travel_Rarely
● Travel_Frequently
● Non-Travel
● TravelRarely


dashboard - Power BI Desktop
Mikil Lalwani
File
Home
Insert
Modeling
View
Help
Get data
Refresh
New visual
More visuals
New measure
Sensitivity
Publish
Filters
Visualizations
Data
Distance From Home vs Attrition vs Companies Worked
Average of DistanceFromHome
Average of NumCompaniesWor...
Average Distance From Home
Average Number of Companies...
Yes
No
Attrition
Attrition by Age Group and Gender
Gender
Female
Male
AgeGroup
26-35
36-45
46-55
18-25
Count of Attrition
Number of Companies Worked and Years Since Last Promotion by Job Role
Average of NumCompaniesWorked
Average of YearsSinceLastPro...
Average Number of Companies ...
Average Years Since Last Promot...
Research Direc...
Manager
Human Resources
Healthcare Representat...
Manufacturing Director
Sales Executive
Laboratory Technician
Research Scientist
Sales Representative
Job Role
Attrition vs Total Working Years vs Salary Slab
Count of Attrition
Upto 5k
5k-10k
10k-15k
15k+
Salary Slab
Attrition by Business Travel
151 (10.2%)
279 (18.85...)
1042 (70.41%)
BusinessTravel
Travel_Rarely
Travel_Freque...
Non-Travel
TravelRarely
Average Year At Company
7.01
Total Employees
1480
Median Monthly Salary
4933
Average Age
36.92
Page 1
Page 1 of 1
89%

Linear regression using Apache Pyspark:-



```python
import pandas as pd
import pyspark.sql.functions as F
from pyspark.sql.functions import col
from pyspark.ml.feature import StandardScaler
from pyspark.sql.types import StructType, StructField, IntegerType, StringType, FloatType
```

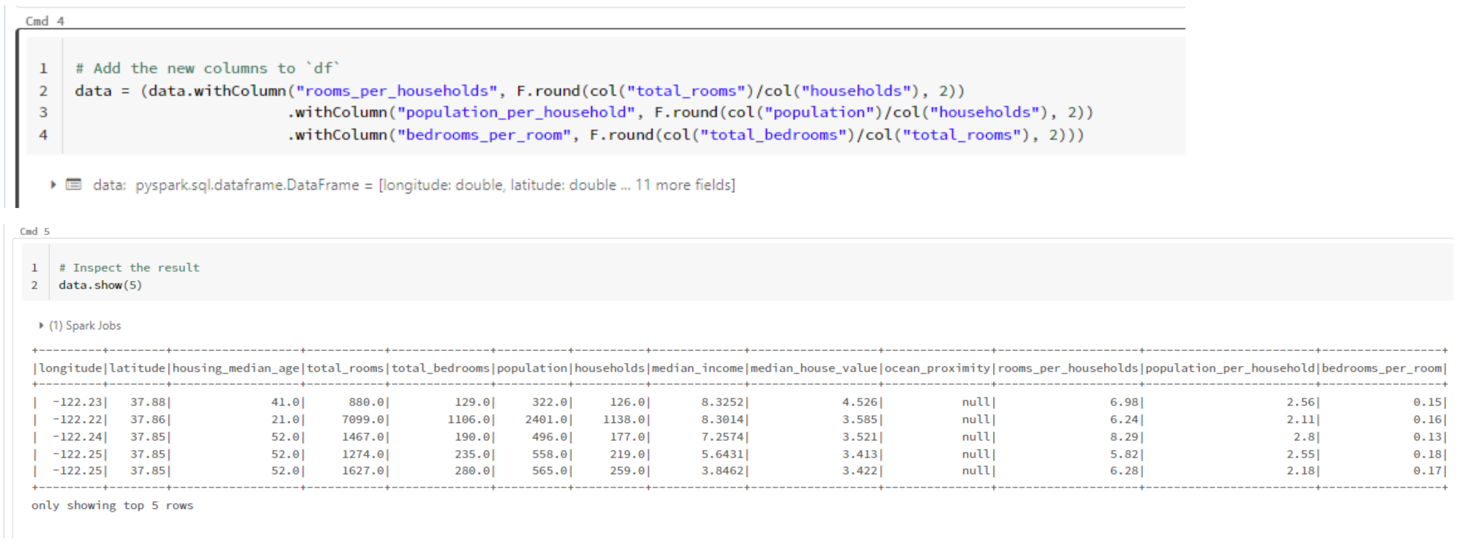
```
Command took 0.07 seconds -- by 2020.shubham.kamodkar@ves.ac.in at 4/5/2023, 7:56:15 PM on Exp 91
```

```python
from pyspark.ml.regression import LinearRegression
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.evaluation import RegressionEvaluator

# Load the data
data = spark.read.format("csv").option("header", "true").load("dbfs:/FileStore/shared_uploads/2020.shubham.kamodkar@ves.ac.in/housing.csv").cache()

# Convert columns to numeric types
data = data.select(*(data[c].cast("double").alias(c) for c in data.columns))

```

```
(1) Spark Jobs
data:  pyspark.sql.dataframe.DataFrame = [longitude: double, latitude: double ... 8 more fields]
Command took 1.00 second -- by 2020.shubham.kamodkar@ves.ac.in at 4/5/2023, 7:56:15 PM on Exp 91
```

```python
# Adjust the values of `medianHouseValue`
data = data.withColumn("median_house_value", col("median_house_value")/100000)
data.show(2)
```

```
(1) Spark Jobs
data:  pyspark.sql.dataframe.DataFrame = [longitude: double, latitude: double ... 8 more fields]
+---------+--------+------------------+-----------+--------------+----------+----------+-------------+------------------+---------------+
|longitude|latitude|housing_median_age|total_rooms|total_bedrooms|population|households|median_income|median_house_value|ocean_proximity|
+---------+--------+------------------+-----------+--------------+----------+----------+-------------+------------------+---------------+
|  -122.23|   37.88|              41.0|      880.0|         129.0|     322.0|     126.0|       8.3252|             4.526|           null|
|  -122.22|   37.86|              21.0|     7099.0|        1106.0|    2401.0|    1138.0|       8.3014|             3.585|           null|
+---------+--------+------------------+-----------+--------------+----------+----------+-------------+------------------+---------------+
only showing top 2 rows
```

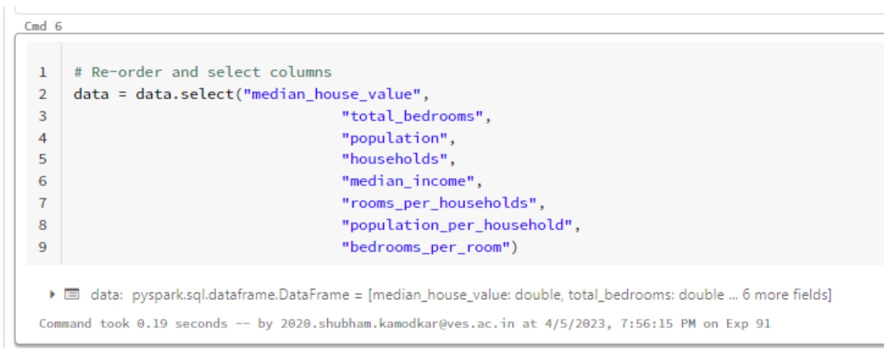


```python
# Add the new columns to `df`
data = (data.withColumn("rooms_per_households", F.round(col("total_rooms")/col("households"), 2))
                        .withColumn("population_per_household", F.round(col("population")/col("households"), 2))
                        .withColumn("bedrooms_per_room", F.round(col("total_bedrooms")/col("total_rooms"), 2)))
```

```
data:  pyspark.sql.dataframe.DataFrame = [longitude: double, latitude: double ... 11 more fields]
```

```python
# Inspect the result
data.show(5)
```

```
(1) Spark Jobs
+---------+--------+------------------+-----------+--------------+----------+----------+-------------+------------------+---------------+--------------------+------------------------+-----------------+
|longitude|latitude|housing_median_age|total_rooms|total_bedrooms|population|households|median_income|median_house_value|ocean_proximity|rooms_per_households|population_per_household|bedrooms_per_room|
+---------+--------+------------------+-----------+--------------+----------+----------+-------------+------------------+---------------+--------------------+------------------------+-----------------+
|  -122.23|   37.88|              41.0|      880.0|         129.0|     322.0|     126.0|       8.3252|             4.526|           null|                6.98|                    2.56|             0.15|
|  -122.22|   37.86|              21.0|     7099.0|        1106.0|    2401.0|    1138.0|       8.3014|             3.585|           null|                6.24|                    2.11|             0.16|
|  -122.24|   37.85|              52.0|     1467.0|         190.0|     496.0|     177.0|       7.2574|             3.521|           null|                8.29|                     2.8|             0.13|
|  -122.25|   37.85|              52.0|     1274.0|         235.0|     558.0|     219.0|       5.6431|             3.413|           null|                5.82|                    2.55|             0.18|
|  -122.25|   37.85|              52.0|     1627.0|         280.0|     565.0|     259.0|       3.8462|             3.422|           null|                6.28|                    2.18|             0.17|
+---------+--------+------------------+-----------+--------------+----------+----------+-------------+------------------+---------------+--------------------+------------------------+-----------------+
only showing top 5 rows
```

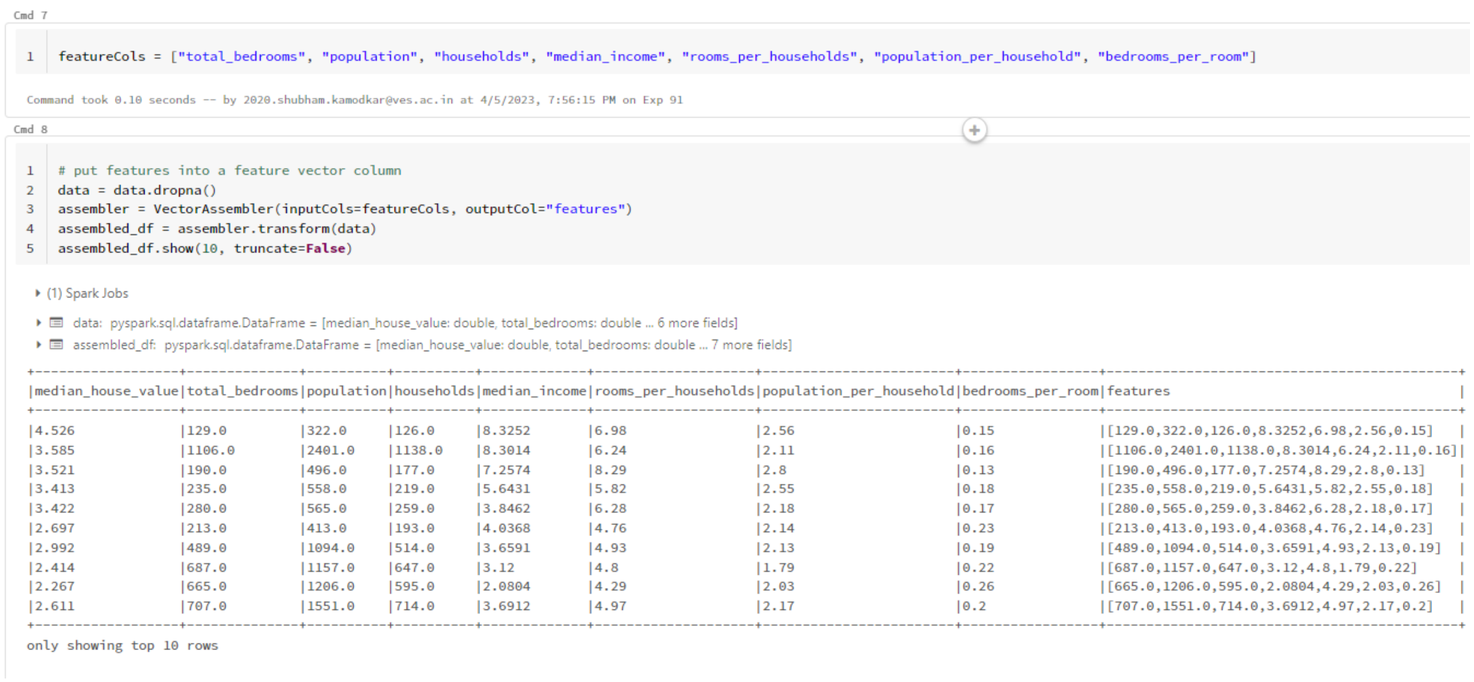


Cmd 6

```python
# Re-order and select columns
data = data.select("median_house_value",
                              "total_bedrooms",
                              "population",
                              "households",
                              "median_income",
                              "rooms_per_households",
                              "population_per_household",
                              "bedrooms_per_room")
```

▸ data: pyspark.sql.dataframe.DataFrame = [median_house_value: double, total_bedrooms: double ... 6 more fields]

Command took 0.19 seconds -- by 2020.shubham.kamodkar@ves.ac.in at 4/5/2023, 7:56:15 PM on Exp 91



Cmd 7

```python
featureCols = ["total_bedrooms", "population", "households", "median_income", "rooms_per_households", "population_per_household", "bedrooms_per_room"]
```

Command took 0.10 seconds -- by 2020.shubham.kamodkar@ves.ac.in at 4/5/2023, 7:56:15 PM on Exp 91

Cmd 8

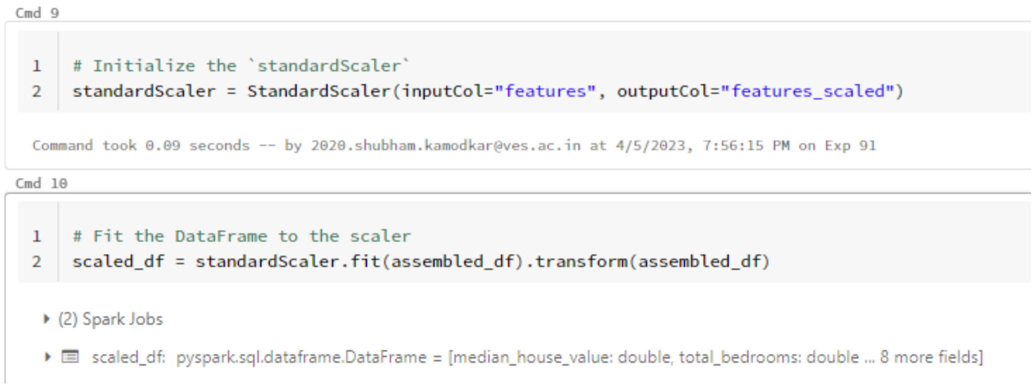
```python
# put features into a feature vector column
data = data.dropna()
assembler = VectorAssembler(inputCols=featureCols, outputCol="features")
assembled_df = assembler.transform(data)
assembled_df.show(10, truncate=False)
```

▸ (1) Spark Jobs

▸ data: pyspark.sql.dataframe.DataFrame = [median_house_value: double, total_bedrooms: double ... 6 more fields]

▸ assembled_df: pyspark.sql.dataframe.DataFrame = [median_house_value: double, total_bedrooms: double ... 7 more fields]

```
+------------------+--------------+----------+----------+-------------+--------------------+------------------------+-----------------+--------------------------------------------+
|median_house_value|total_bedrooms|population|households|median_income|rooms_per_households|population_per_household|bedrooms_per_room|features                                    |
+------------------+--------------+----------+----------+-------------+--------------------+------------------------+-----------------+--------------------------------------------+
|4.526             |129.0         |322.0     |126.0     |8.3252       |6.98                |2.56                    |0.15             |[129.0,322.0,126.0,8.3252,6.98,2.56,0.15]   |
|3.585             |1106.0        |2401.0    |1138.0    |8.3014       |6.24                |2.11                    |0.16             |[1106.0,2401.0,1138.0,8.3014,6.24,2.11,0.16]|
|3.521             |190.0         |496.0     |177.0     |7.2574       |8.29                |2.8                     |0.13             |[190.0,496.0,177.0,7.2574,8.29,2.8,0.13]    |
|3.413             |235.0         |558.0     |219.0     |5.6431       |5.82                |2.55                    |0.18             |[235.0,558.0,219.0,5.6431,5.82,2.55,0.18]   |
|3.422             |280.0         |565.0     |259.0     |3.8462       |6.28                |2.18                    |0.17             |[280.0,565.0,259.0,3.8462,6.28,2.18,0.17]   |
|2.697             |213.0         |413.0     |193.0     |4.0368       |4.76                |2.14                    |0.23             |[213.0,413.0,193.0,4.0368,4.76,2.14,0.23]   |
|2.992             |489.0         |1094.0    |514.0     |3.6591       |4.93                |2.13                    |0.19             |[489.0,1094.0,514.0,3.6591,4.93,2.13,0.19]  |
|2.414             |687.0         |1157.0    |647.0     |3.12         |4.8                 |1.79                    |0.22             |[687.0,1157.0,647.0,3.12,4.8,1.79,0.22]     |
|2.267             |665.0         |1206.0    |595.0     |2.0804       |4.29                |2.03                    |0.26             |[665.0,1206.0,595.0,2.0804,4.29,2.03,0.26]  |
|2.611             |707.0         |1551.0    |714.0     |3.6912       |4.97                |2.17                    |0.2              |[707.0,1551.0,714.0,3.6912,4.97,2.17,0.2]   |
+------------------+--------------+----------+----------+-------------+--------------------+------------------------+-----------------+--------------------------------------------+
only showing top 10 rows
```



Cmd 9

```python
# Initialize the `standardScaler`
standardScaler = StandardScaler(inputCol="features", outputCol="features_scaled")
```

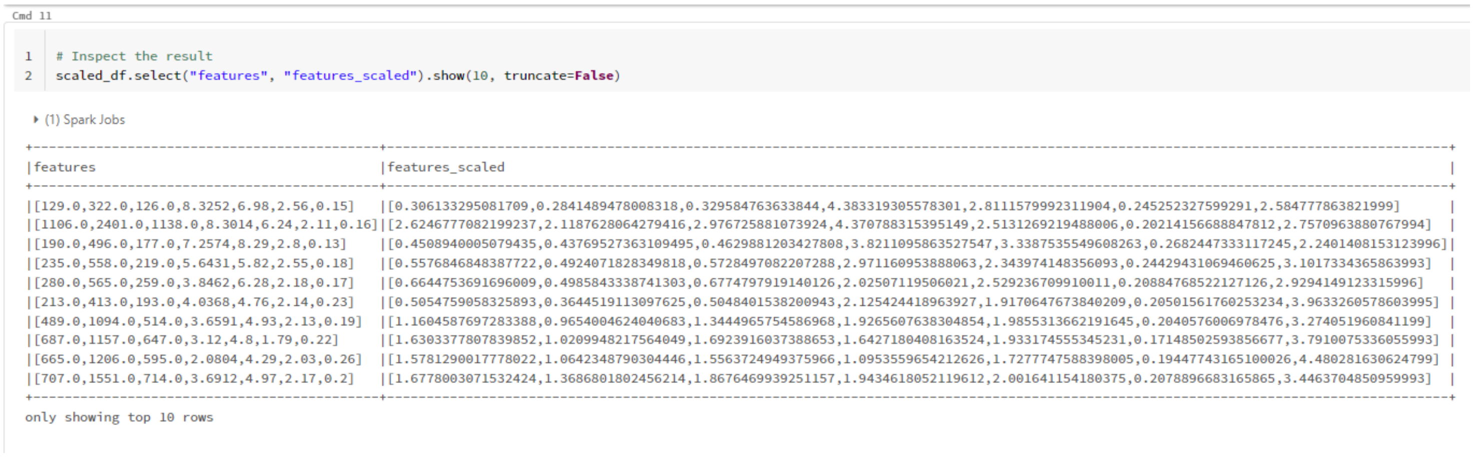
Command took 0.09 seconds -- by 2020.shubham.kamodkar@ves.ac.in at 4/5/2023, 7:56:15 PM on Exp 91

Cmd 10

```python
# Fit the DataFrame to the scaler
scaled_df = standardScaler.fit(assembled_df).transform(assembled_df)
```

▸ (2) Spark Jobs

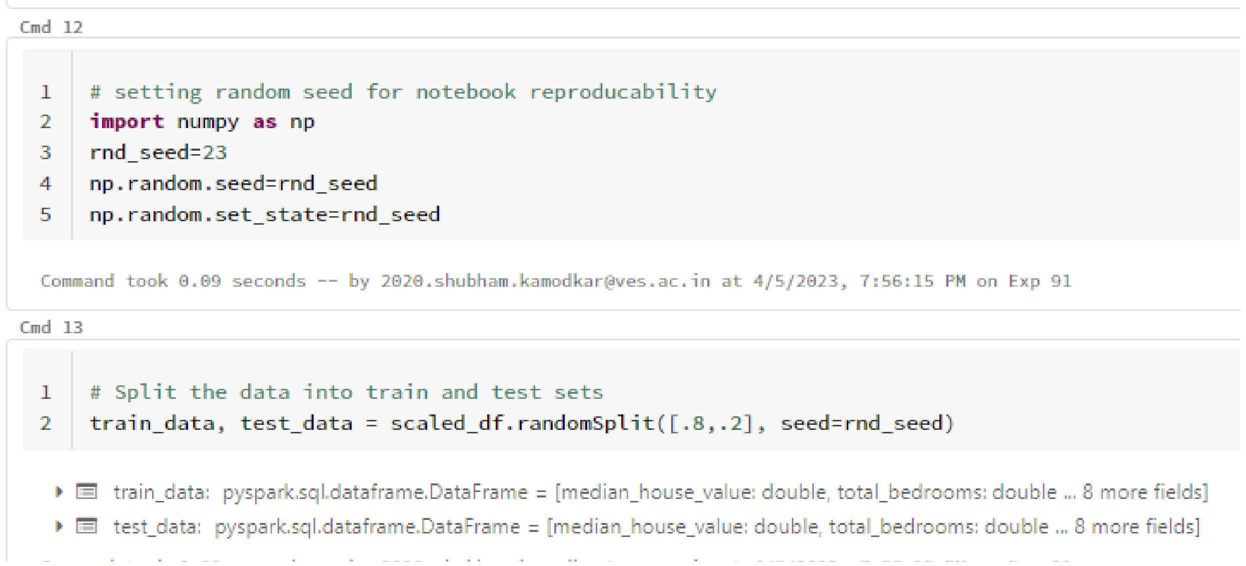
▸ scaled_df: pyspark.sql.dataframe.DataFrame = [median_house_value: double, total_bedrooms: double ... 8 more fields]


Cmd 11

```python
# Inspect the result
scaled_df.select("features", "features_scaled").show(10, truncate=False)
```

▸ (1) Spark Jobs

```
+-----------------------------------------+--------------------------------------------------------------------------------------------------------------------------------------+
|features                                 |features_scaled                                                                                                                       |
+-----------------------------------------+--------------------------------------------------------------------------------------------------------------------------------------+
|[129.0,322.0,126.0,8.3252,6.98,2.56,0.15]   |[0.306133295081709,0.2841489478008318,0.329584763633844,4.383319305578301,2.8111579992311904,0.245252327599291,2.584777863821999]      |
|[1106.0,2401.0,1138.0,8.3014,6.24,2.11,0.16]|[2.6246777082199237,2.1187628064279416,2.976725881073924,4.370788315395149,2.5131269219488006,0.20214156688847812,2.7570963880767994]  |
|[190.0,496.0,177.0,7.2574,8.29,2.8,0.13]    |[0.4508940005079435,0.43769527363109495,0.4629881203427808,3.8211095863527547,3.3387535549608263,0.2682447333117245,2.2401408153123996]|
|[235.0,558.0,219.0,5.6431,5.82,2.55,0.18]   |[0.5576846848387722,0.4924071828349818,0.5728497082207288,2.971160953888063,2.343974148356093,0.24429431069460625,3.1017334365863993]  |
|[280.0,565.0,259.0,3.8462,6.28,2.18,0.17]   |[0.6644753691696009,0.4985843338741303,0.6774797919140126,2.02507119506021,2.529236709910011,0.20884768522127126,2.9294149123315996]   |
|[213.0,413.0,193.0,4.0368,4.76,2.14,0.23]   |[0.5054759058325893,0.3644519113097625,0.5048401538200943,2.125424418963927,1.9170647673840209,0.20501561760253234,3.9633260578603995] |
|[489.0,1094.0,514.0,3.6591,4.93,2.13,0.19]  |[1.1604587697283388,0.9654004624040683,1.3444965754586968,1.9265607638304854,1.9855313662191645,0.2040576006978476,3.274051960841199]  |
|[687.0,1157.0,647.0,3.12,4.8,1.79,0.22]     |[1.6303377807839852,1.0209948217564049,1.6923916037388653,1.6427180408163524,1.933174555345231,0.17148502593856677,3.7910075336055993] |
|[665.0,1206.0,595.0,2.0804,4.29,2.03,0.26]  |[1.5781290017778022,1.0642348790304446,1.5563724949375966,1.0953559654212626,1.7277747588398005,0.19447743165100026,4.480281630624799] |
|[707.0,1551.0,714.0,3.6912,4.97,2.17,0.2]   |[1.6778003071532424,1.3686801802456214,1.8676469939251157,1.9434618052119612,2.001641154180375,0.2078896683165865,3.4463704850959993]  |
+-----------------------------------------+--------------------------------------------------------------------------------------------------------------------------------------+
only showing top 10 rows
```


Cmd 12

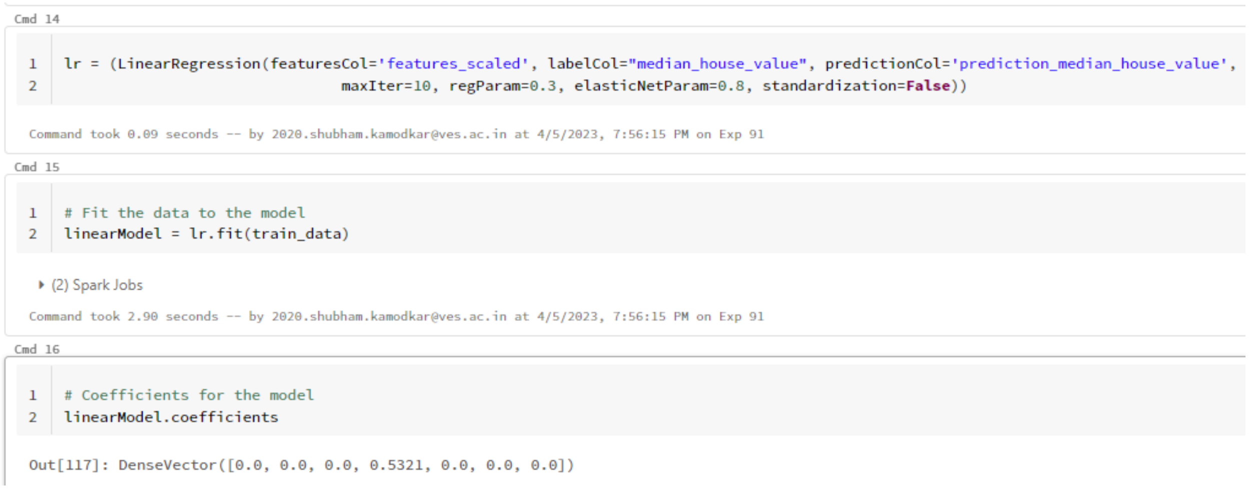
```python
# setting random seed for notebook reproducability
import numpy as np
rnd_seed=23
np.random.seed=rnd_seed
np.random.set_state=rnd_seed
```

Command took 0.09 seconds -- by 2020.shubham.kamodkar@ves.ac.in at 4/5/2023, 7:56:15 PM on Exp 91

Cmd 13

```python
# Split the data into train and test sets
train_data, test_data = scaled_df.randomSplit([.8,.2], seed=rnd_seed)
```

▸ train_data: pyspark.sql.dataframe.DataFrame = [median_house_value: double, total_bedrooms: double ... 8 more fields]
▸ test_data: pyspark.sql.dataframe.DataFrame = [median_house_value: double, total_bedrooms: double ... 8 more fields]


Cmd 14

```python
lr = (LinearRegression(featuresCol='features_scaled', labelCol="median_house_value", predictionCol='prediction_median_house_value',
                               maxIter=10, regParam=0.3, elasticNetParam=0.8, standardization=False))
```

Command took 0.09 seconds -- by 2020.shubham.kamodkar@ves.ac.in at 4/5/2023, 7:56:15 PM on Exp 91

Cmd 15

```python
# Fit the data to the model
linearModel = lr.fit(train_data)
```

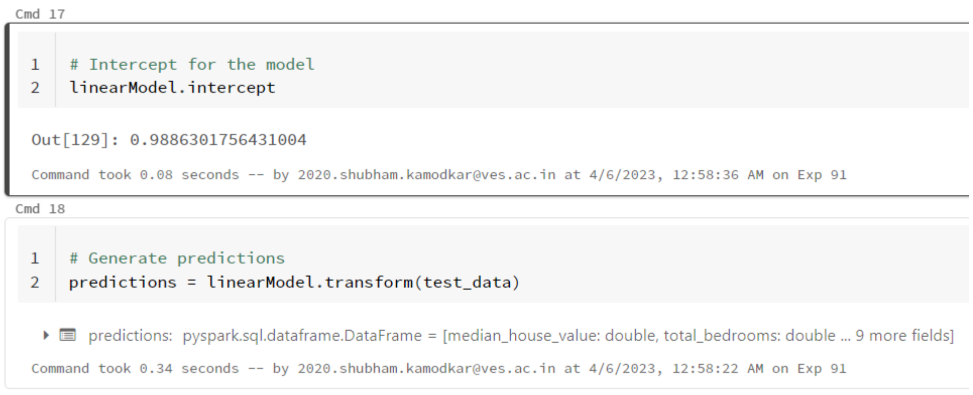
▸ (2) Spark Jobs

Command took 2.90 seconds -- by 2020.shubham.kamodkar@ves.ac.in at 4/5/2023, 7:56:15 PM on Exp 91

Cmd 16

```python
# Coefficients for the model
linearModel.coefficients
```

```
Out[117]: DenseVector([0.0, 0.0, 0.0, 0.5321, 0.0, 0.0, 0.0])
```


Cmd 17

```
# Intercept for the model
linearModel.intercept
```

```
Out[129]: 0.9886301756431004
```

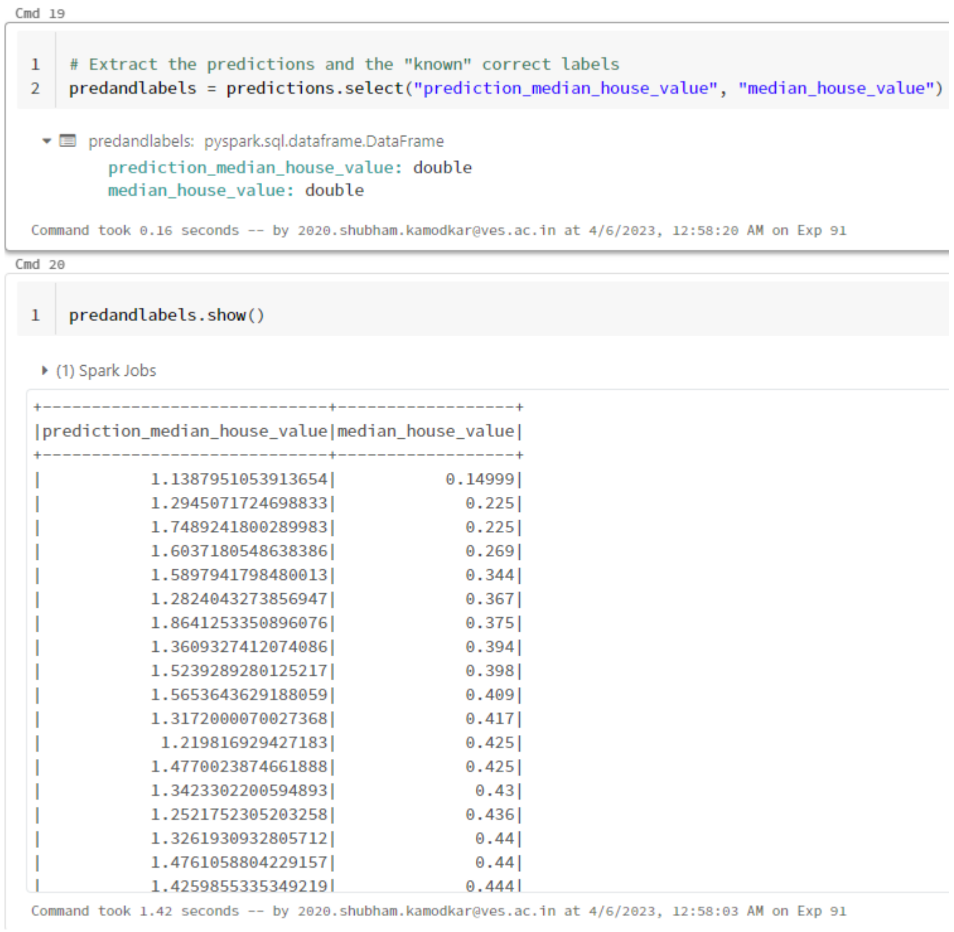
Command took 0.08 seconds -- by 2020.shubham.kamodkar@ves.ac.in at 4/6/2023, 12:58:36 AM on Exp 91

Cmd 18

```
# Generate predictions
predictions = linearModel.transform(test_data)
```

predictions: pyspark.sql.dataframe.DataFrame = [median_house_value: double, total_bedrooms: double ... 9 more fields]

Command took 0.34 seconds -- by 2020.shubham.kamodkar@ves.ac.in at 4/6/2023, 12:58:22 AM on Exp 91


Cmd 19

```
# Extract the predictions and the "known" correct labels
predandlabels = predictions.select("prediction_median_house_value", "median_house_value")
```

predandlabels: pyspark.sql.dataframe.DataFrame

```
    prediction_median_house_value: double
    median_house_value: double
```

Command took 0.16 seconds -- by 2020.shubham.kamodkar@ves.ac.in at 4/6/2023, 12:58:20 AM on Exp 91

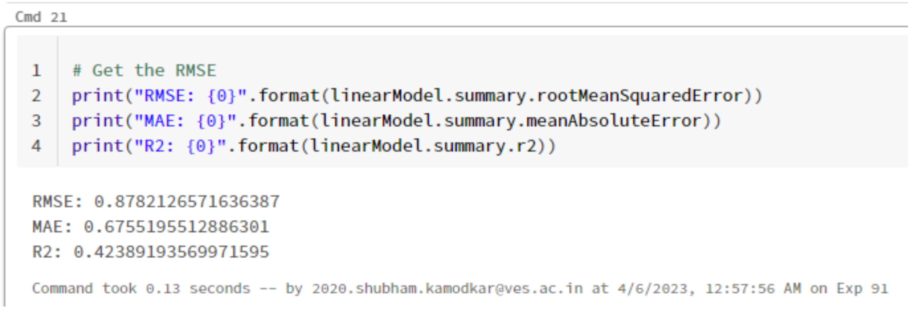
Cmd 20

```
predandlabels.show()
```

(1) Spark Jobs

```
+-----------------------------+------------------+
|prediction_median_house_value|median_house_value|
+-----------------------------+------------------+
|           1.1387951053913654|           0.14999|
|           1.2945071724698833|             0.225|
|           1.7489241800289983|             0.225|
|           1.6037180548638386|             0.269|
|           1.5897941798480013|             0.344|
|           1.2824043273856947|             0.367|
|           1.8641253350896076|             0.375|
|           1.3609327412074086|             0.394|
|           1.5239289280125217|             0.398|
|           1.5653643629188059|             0.409|
|           1.3172000070027368|             0.417|
|            1.219816929427183|             0.425|
|           1.4770023874661888|             0.425|
|           1.3423302200594893|              0.43|
|           1.2521752305203258|             0.436|
|           1.3261930932805712|              0.44|
|           1.4761058804229157|              0.44|
|           1.4259855335349219|             0.444|
```

Command took 1.42 seconds -- by 2020.shubham.kamodkar@ves.ac.in at 4/6/2023, 12:58:03 AM on Exp 91

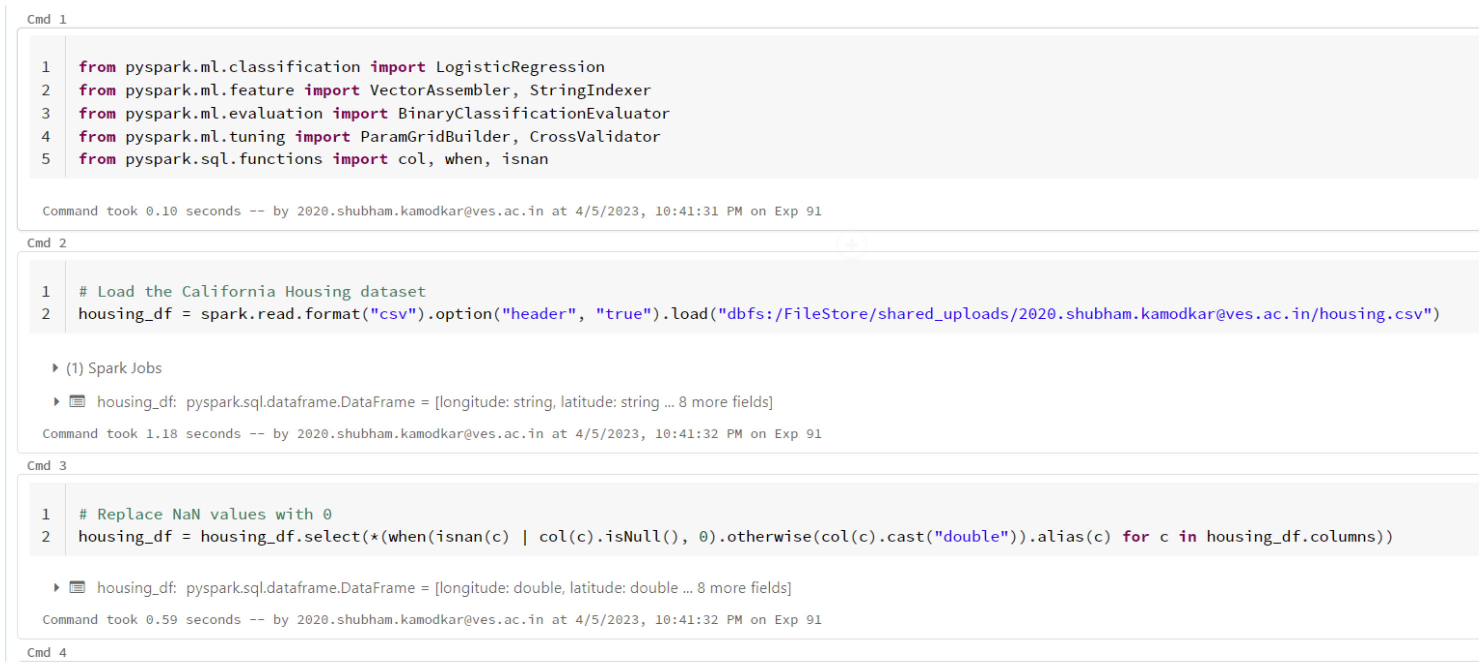

Cmd 21

```
# Get the RMSE
print("RMSE: {0}".format(linearModel.summary.rootMeanSquaredError))
print("MAE: {0}".format(linearModel.summary.meanAbsoluteError))
print("R2: {0}".format(linearModel.summary.r2))
```

```
RMSE: 0.8782126571636387
MAE: 0.6755195512886301
R2: 0.42389193569971595
```

Command took 0.13 seconds -- by 2020.shubham.kamodkar@ves.ac.in at 4/6/2023, 12:57:56 AM on Exp 91

Logistic regression using Apache Pyspark:-



Cmd 1

```
from pyspark.ml.classification import LogisticRegression
from pyspark.ml.feature import VectorAssembler, StringIndexer
from pyspark.ml.evaluation import BinaryClassificationEvaluator
from pyspark.ml.tuning import ParamGridBuilder, CrossValidator
from pyspark.sql.functions import col, when, isnan
```

```
Command took 0.10 seconds -- by 2020.shubham.kamodkar@ves.ac.in at 4/5/2023, 10:41:31 PM on Exp 91
```

Cmd 2

```
# Load the California Housing dataset
housing_df = spark.read.format("csv").option("header", "true").load("dbfs:/FileStore/shared_uploads/2020.shubham.kamodkar@ves.ac.in/housing.csv")
```

(1) Spark Jobs

housing_df: pyspark.sql.dataframe.DataFrame = [longitude: string, latitude: string ... 8 more fields]

```
Command took 1.18 seconds -- by 2020.shubham.kamodkar@ves.ac.in at 4/5/2023, 10:41:32 PM on Exp 91
```

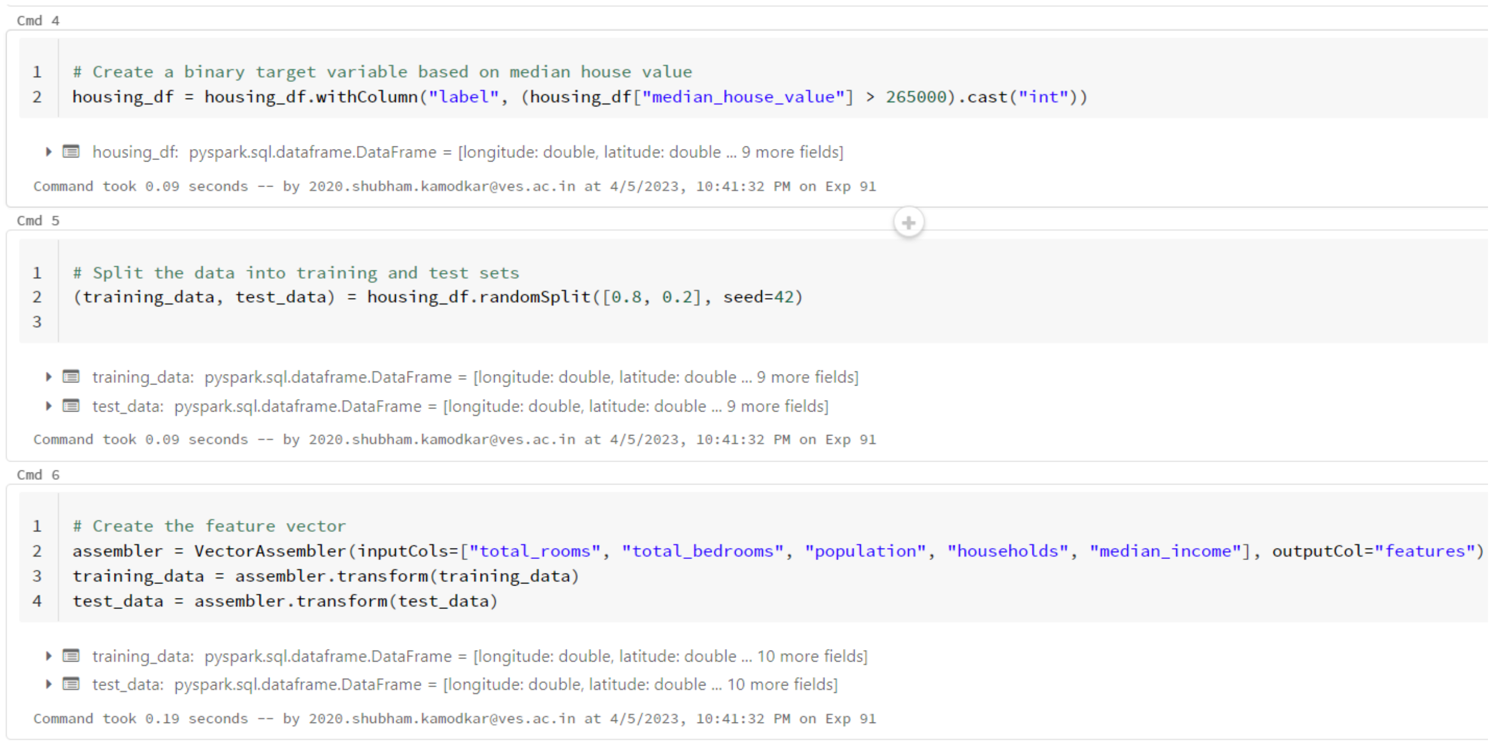
Cmd 3

```
# Replace NaN values with 0
housing_df = housing_df.select(*(when(isnan(c) | col(c).isNull(), 0).otherwise(col(c).cast("double")).alias(c) for c in housing_df.columns))
```

housing_df: pyspark.sql.dataframe.DataFrame = [longitude: double, latitude: double ... 8 more fields]

```
Command took 0.59 seconds -- by 2020.shubham.kamodkar@ves.ac.in at 4/5/2023, 10:41:32 PM on Exp 91
```

Cmd 4



Cmd 4

```
# Create a binary target variable based on median house value
housing_df = housing_df.withColumn("label", (housing_df["median_house_value"] > 265000).cast("int"))
```

housing_df: pyspark.sql.dataframe.DataFrame = [longitude: double, latitude: double ... 9 more fields]

```
Command took 0.09 seconds -- by 2020.shubham.kamodkar@ves.ac.in at 4/5/2023, 10:41:32 PM on Exp 91
```

Cmd 5

```
# Split the data into training and test sets
(training_data, test_data) = housing_df.randomSplit([0.8, 0.2], seed=42)

```

training_data: pyspark.sql.dataframe.DataFrame = [longitude: double, latitude: double ... 9 more fields]

test_data: pyspark.sql.dataframe.DataFrame = [longitude: double, latitude: double ... 9 more fields]

```
Command took 0.09 seconds -- by 2020.shubham.kamodkar@ves.ac.in at 4/5/2023, 10:41:32 PM on Exp 91
```

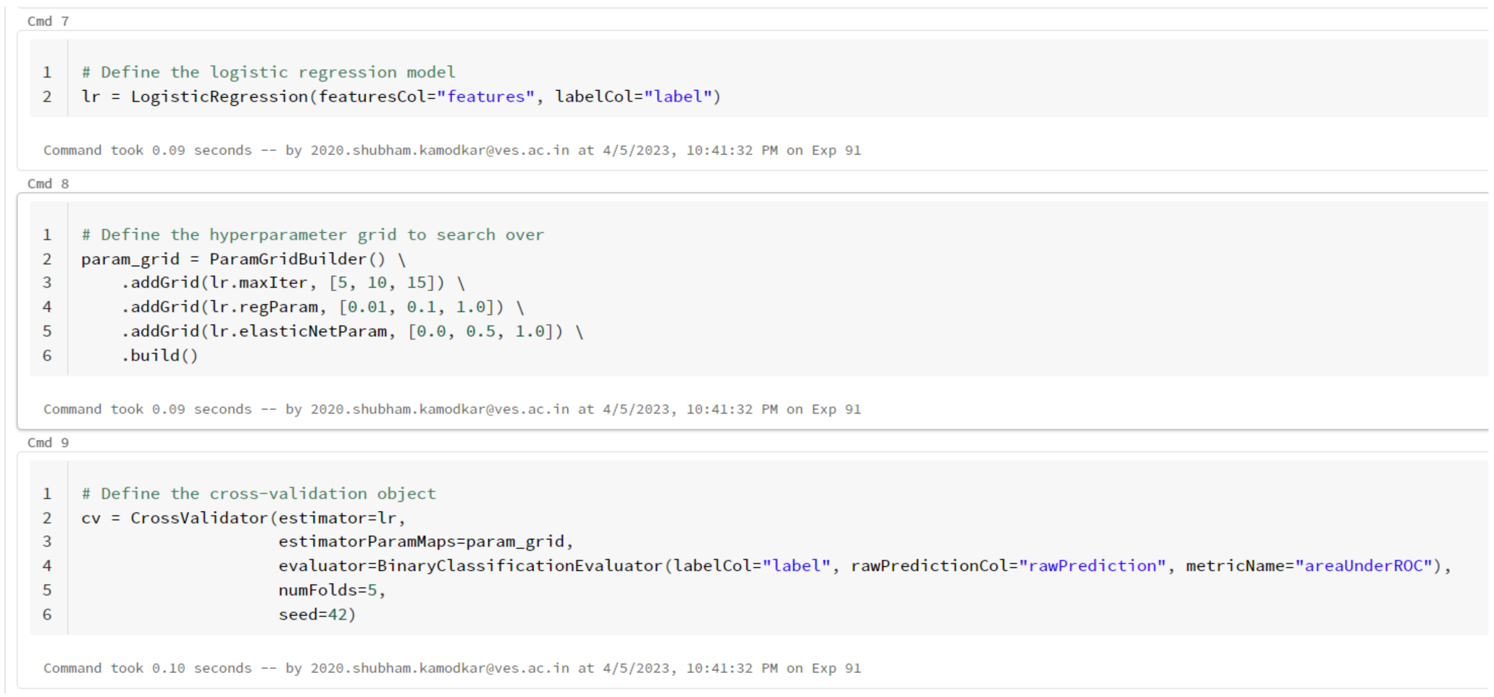
Cmd 6

```
# Create the feature vector
assembler = VectorAssembler(inputCols=["total_rooms", "total_bedrooms", "population", "households", "median_income"], outputCol="features")
training_data = assembler.transform(training_data)
test_data = assembler.transform(test_data)
```

training_data: pyspark.sql.dataframe.DataFrame = [longitude: double, latitude: double ... 10 more fields]

test_data: pyspark.sql.dataframe.DataFrame = [longitude: double, latitude: double ... 10 more fields]

```
Command took 0.19 seconds -- by 2020.shubham.kamodkar@ves.ac.in at 4/5/2023, 10:41:32 PM on Exp 91
```



Cmd 7

```python
# Define the logistic regression model
lr = LogisticRegression(featuresCol="features", labelCol="label")
```

```
Command took 0.09 seconds -- by 2020.shubham.kamodkar@ves.ac.in at 4/5/2023, 10:41:32 PM on Exp 91
```

Cmd 8

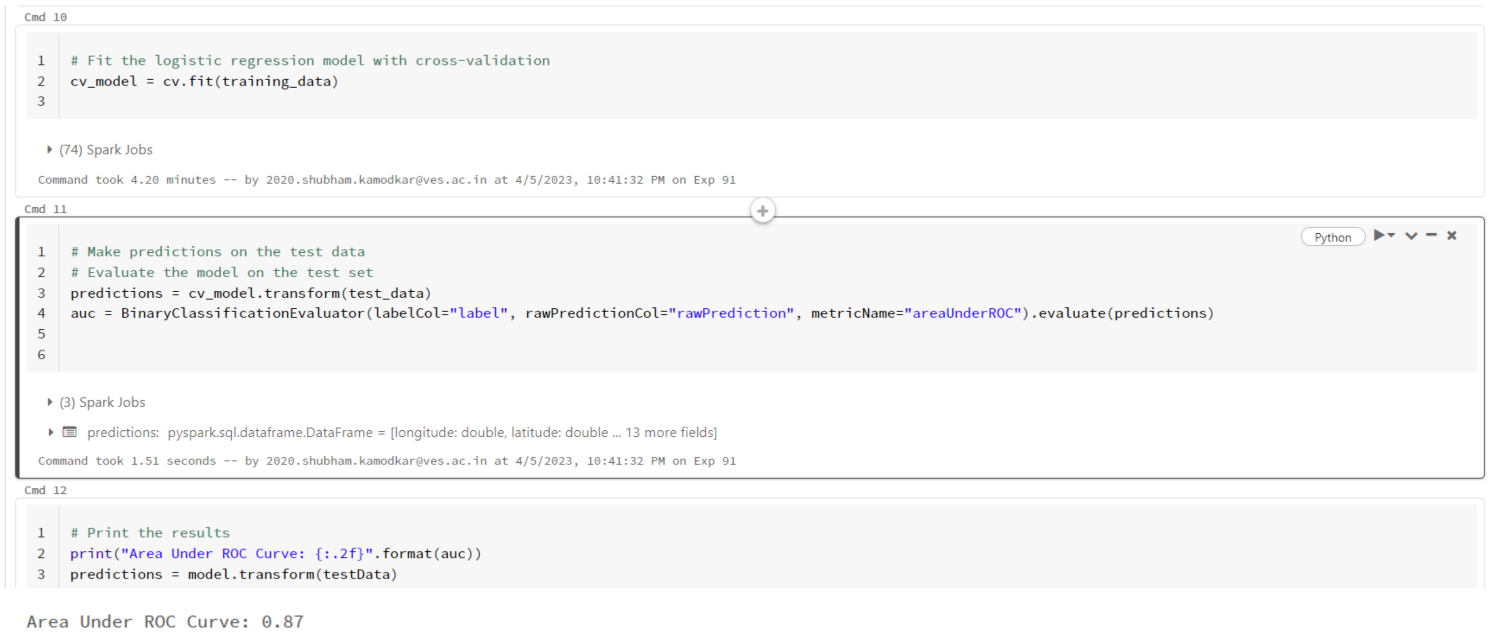
```python
# Define the hyperparameter grid to search over
param_grid = ParamGridBuilder() \
    .addGrid(lr.maxIter, [5, 10, 15]) \
    .addGrid(lr.regParam, [0.01, 0.1, 1.0]) \
    .addGrid(lr.elasticNetParam, [0.0, 0.5, 1.0]) \
    .build()
```

```
Command took 0.09 seconds -- by 2020.shubham.kamodkar@ves.ac.in at 4/5/2023, 10:41:32 PM on Exp 91
```

Cmd 9

```python
# Define the cross-validation object
cv = CrossValidator(estimator=lr,
                    estimatorParamMaps=param_grid,
                    evaluator=BinaryClassificationEvaluator(labelCol="label", rawPredictionCol="rawPrediction", metricName="areaUnderROC"),
                    numFolds=5,
                    seed=42)
```

```
Command took 0.10 seconds -- by 2020.shubham.kamodkar@ves.ac.in at 4/5/2023, 10:41:32 PM on Exp 91
```



Cmd 10

```python
# Fit the logistic regression model with cross-validation
cv_model = cv.fit(training_data)

```

▸ (74) Spark Jobs

```
Command took 4.20 minutes -- by 2020.shubham.kamodkar@ves.ac.in at 4/5/2023, 10:41:32 PM on Exp 91
```

Cmd 11

```python
# Make predictions on the test data
# Evaluate the model on the test set
predictions = cv_model.transform(test_data)
auc = BinaryClassificationEvaluator(labelCol="label", rawPredictionCol="rawPrediction", metricName="areaUnderROC").evaluate(predictions)


```

▸ (3) Spark Jobs

▸ predictions: pyspark.sql.dataframe.DataFrame = [longitude: double, latitude: double ... 13 more fields]

```
Command took 1.51 seconds -- by 2020.shubham.kamodkar@ves.ac.in at 4/5/2023, 10:41:32 PM on Exp 91
```

Cmd 12

```python
# Print the results
print("Area Under ROC Curve: {:.2f}".format(auc))
predictions = model.transform(testData)
```

```
Area Under ROC Curve: 0.87
```

Conclusion:

Using Power Bi, we visualized data from the housing dataset. We learned about big data and also learned how to use Apache Spark and how to use its libraries for preprocessing and visualization.