Report

1 Introduction:

The main aim in this project is to analyze Housing dataset and predict price of house in USA, and to achieve that the objectives were set:

1. Use PySpark to load and process data

2. Explore and analyze data using appropriate techniques

3. Visualize data using Tableau or Python libraries and discuss the findings

4. Identify appropriate machine learning techniques to predict PM2.5

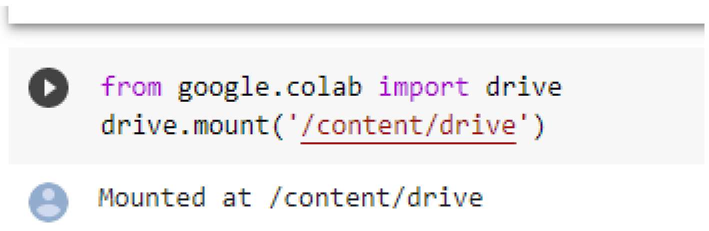
5. Discuss findings and conclude

The following section 2 explains PySpark set up, the dataset, data analysis and Machine Learning implementation

2 Installation and Importing:

Pyspark installation:

The first thing you want to do when you are working on Colab is mounting your Google Drive. This will enable you to access any directory on your Drive inside the Colab notebook.



```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

Setting up PySpark in Colab

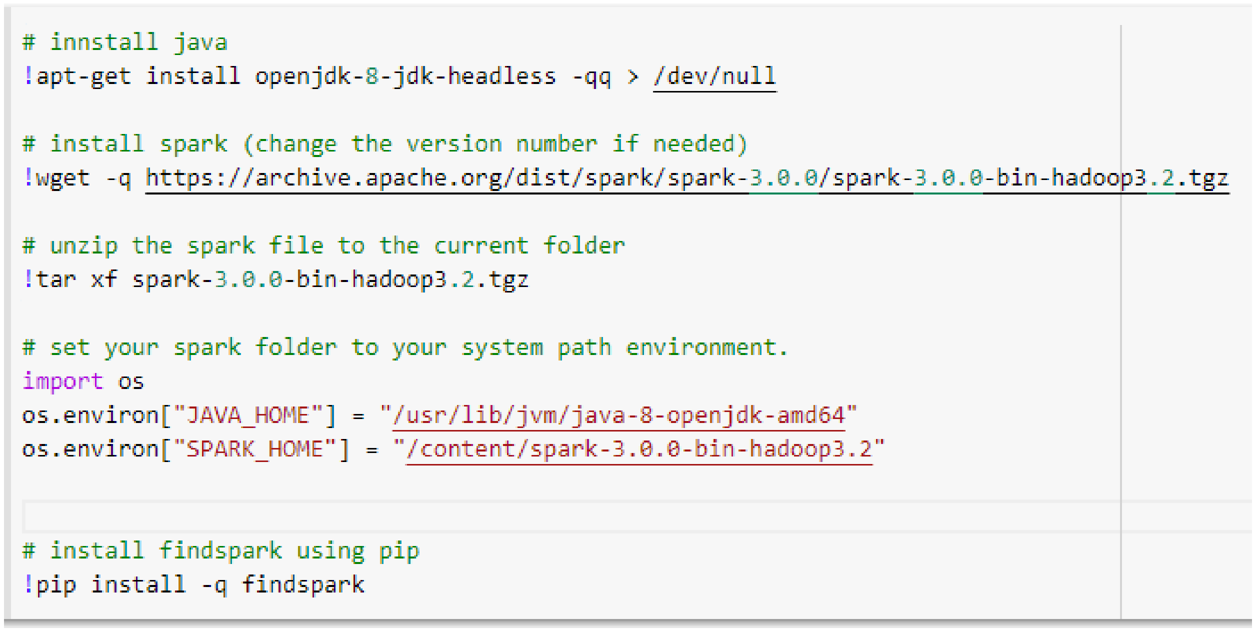
Spark is written in the Scala programming language and requires the Java Virtual Machine (JVM) to run. Therefore, our first task is to download Java.

Next, we will install Apache Spark 3.0.0 with Hadoop 3.2

Then, we just need to unzip that folder.

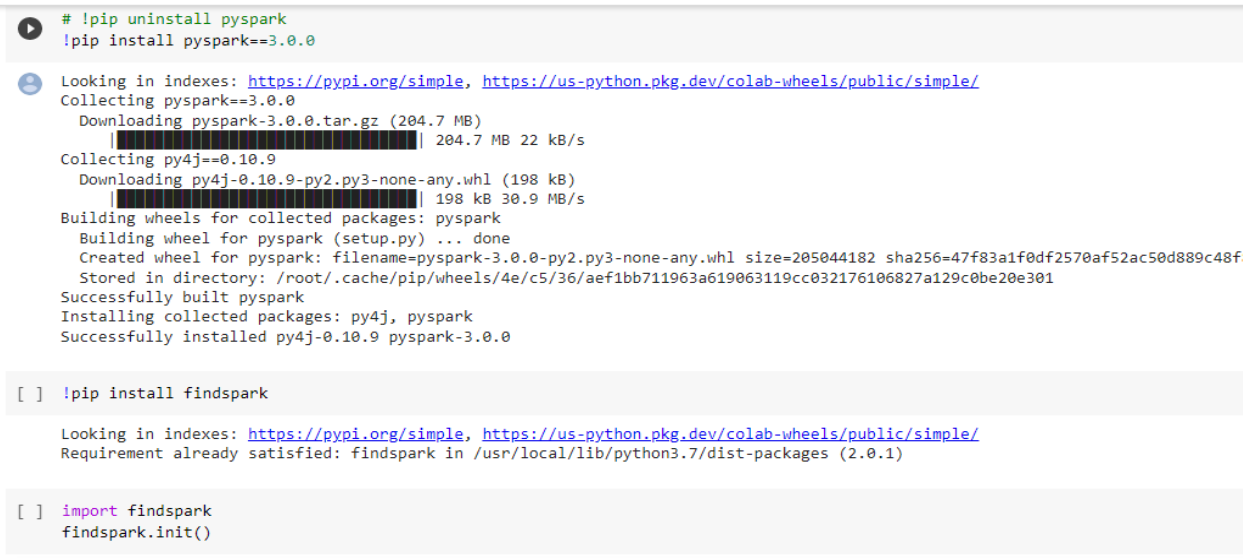
There is one last thing that we need to install and that is the findspark library. It will locate Spark on the system and import it as a regular library.

Now that we have installed all the necessary dependencies in Colab, it is time to set the environment path. This will enable us to run Pyspark in the Colab environment.



```
# innstall java
!apt-get install openjdk-8-jdk-headless -qq > /dev/null

# install spark (change the version number if needed)
!wget -q https://archive.apache.org/dist/spark/spark-3.0.0/spark-3.0.0-bin-hadoop3.2.tgz

# unzip the spark file to the current folder
!tar xf spark-3.0.0-bin-hadoop3.2.tgz

# set your spark folder to your system path environment.
import os
os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-8-openjdk-amd64"
os.environ["SPARK_HOME"] = "/content/spark-3.0.0-bin-hadoop3.2"


# install findspark using pip
!pip install -q findspark
```

Download pyspark 3.0.0 Then We need to locate Spark in the system. For that, we import findspark and use the findspark.init() method.

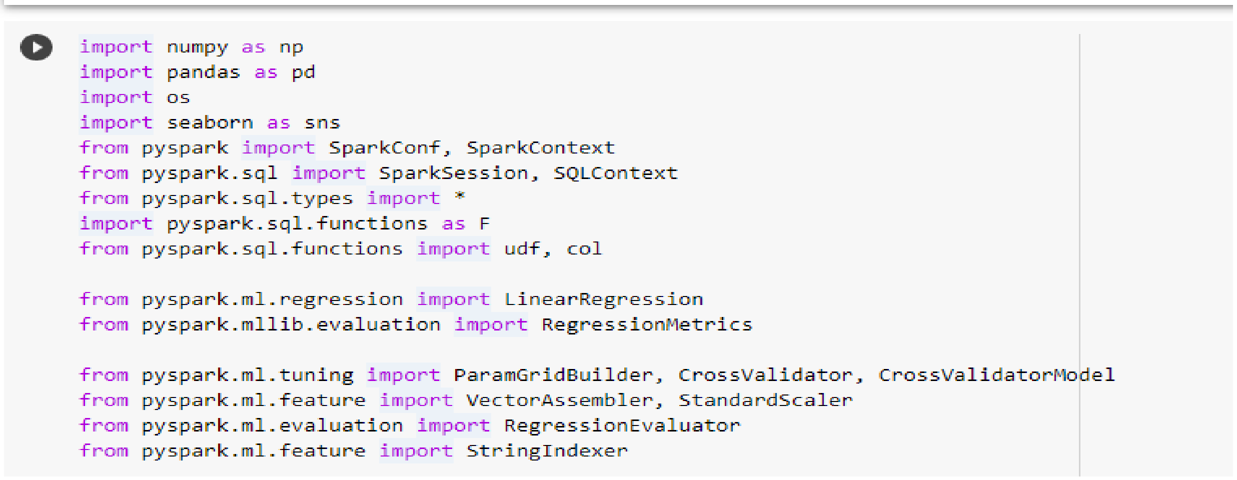


```
# !pip uninstall pyspark
!pip install pyspark==3.0.0

Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting pyspark==3.0.0
  Downloading pyspark-3.0.0.tar.gz (204.7 MB)
     |████████████████████████████████| 204.7 MB 22 kB/s
Collecting py4j==0.10.9
  Downloading py4j-0.10.9-py2.py3-none-any.whl (198 kB)
     |████████████████████████████████| 198 kB 30.9 MB/s
Building wheels for collected packages: pyspark
  Building wheel for pyspark (setup.py) ... done
  Created wheel for pyspark: filename=pyspark-3.0.0-py2.py3-none-any.whl size=205044182 sha256=47f83a1f0df2570af52ac50d889c48f
  Stored in directory: /root/.cache/pip/wheels/4e/c5/36/aef1bb711963a619063119cc032176106827a129c0be20e301
Successfully built pyspark
Installing collected packages: py4j, pyspark
Successfully installed py4j-0.10.9 pyspark-3.0.0

[ ] !pip install findspark

Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: findspark in /usr/local/lib/python3.7/dist-packages (2.0.1)

[ ] import findspark
    findspark.init()
```

Import libraries:

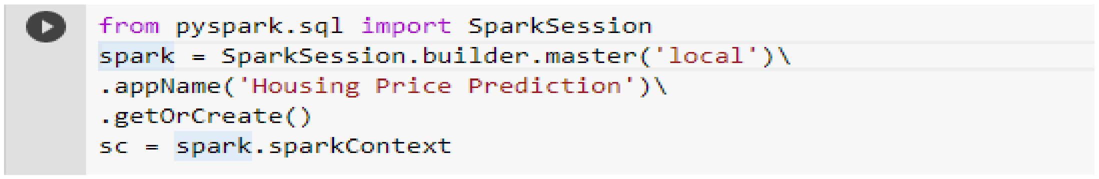
I import all the necessary libraries that is use in our code. Sea-born and matplotlib is use for visualization, pandas is use for data loading and data manipulation. Import spark session, SQL context because both is use as a entry point to spark and spark SQL.

I import linear regression, regression metrics for evaluation and string indexer to handle categorical features and vector assembler is use to combine all features into one features. Then standard scaler is use to scale the features, this make a huge impact in performance of model.



```
import numpy as np
import pandas as pd
import os
import seaborn as sns
from pyspark import SparkConf, SparkContext
from pyspark.sql import SparkSession, SQLContext
from pyspark.sql.types import *
import pyspark.sql.functions as F
from pyspark.sql.functions import udf, col

from pyspark.ml.regression import LinearRegression
from pyspark.mllib.evaluation import RegressionMetrics

from pyspark.ml.tuning import ParamGridBuilder, CrossValidator, CrossValidatorModel
from pyspark.ml.feature import VectorAssembler, StandardScaler
from pyspark.ml.evaluation import RegressionEvaluator
from pyspark.ml.feature import StringIndexer
```

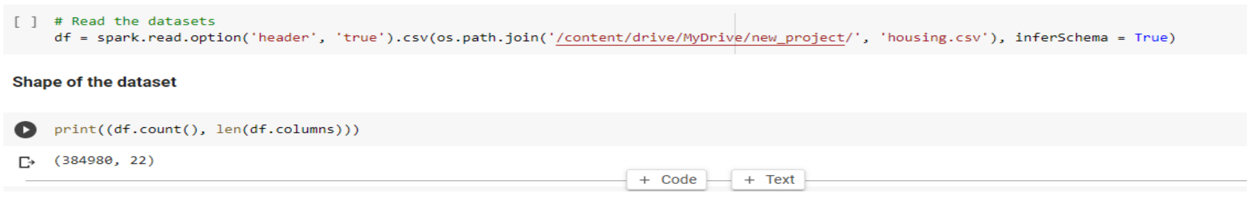
Then I create a Spark Session. Spark Session is an entry point to Spark to work with RDD, Data Frame, and Dataset. To create Spark Session in Python, we need to use the builder() method and calling getOrCreate() method. If Spark Session already exists it returns otherwise create a new Spark Session.



```
from pyspark.sql import SparkSession
spark = SparkSession.builder.master('local')\
.appName('Housing Price Prediction')\
.getOrCreate()
sc = spark.sparkContext
```

Load the datasets:

I load the datasets from spark.read and make header and infer schema is True.



```
[ ] # Read the datasets
    df = spark.read.option('header', 'true').csv(os.path.join('/content/drive/MyDrive/new_project/', 'housing.csv'), inferSchema = True)
```

Shape of the dataset

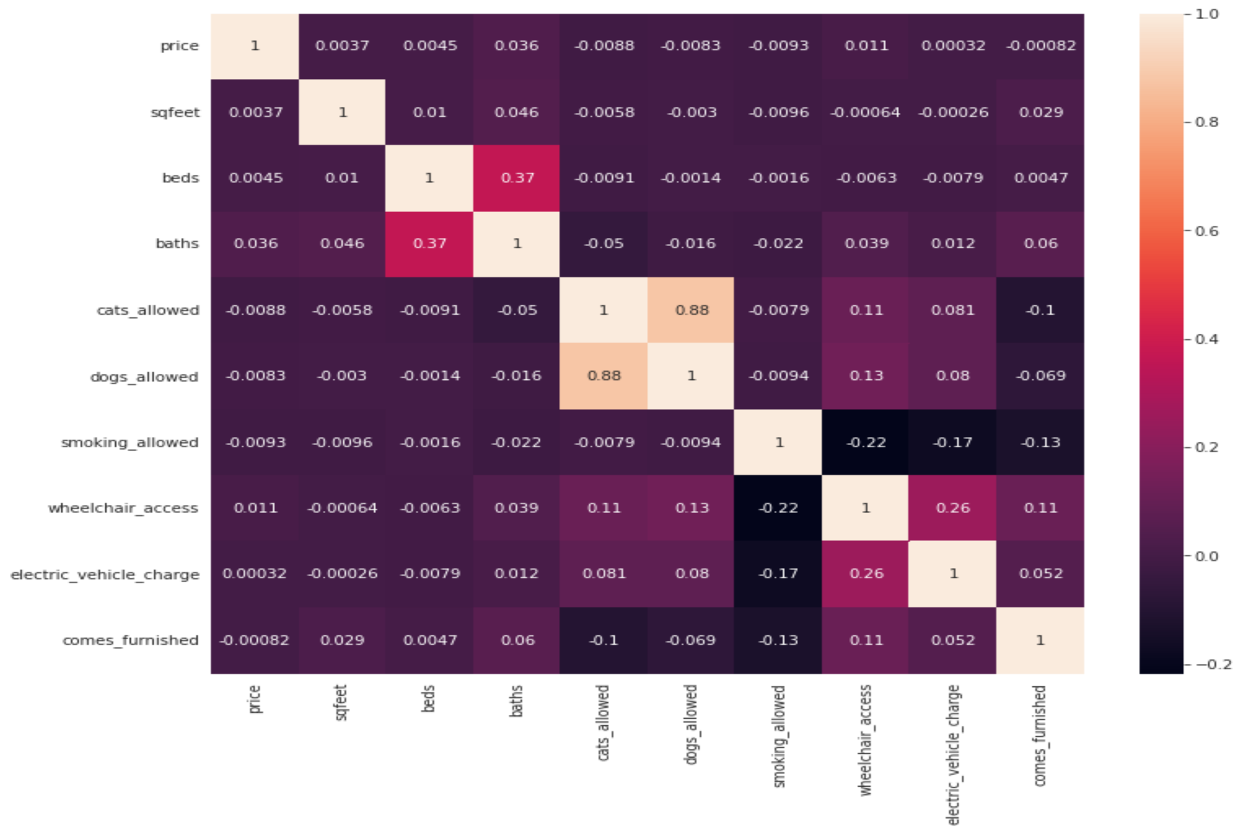
```
print((df.count(), len(df.columns)))
(384980, 22)
```

3 Data Analysis and Visualization:

Heat Map:

Heatmap is defined as a graphical representation of data using colors to visualize the value of the matrix. In this, to represent more common values or higher activities brighter colors basically reddish colors are used and to represent less common or activity values, darker colors are preferred. Heatmap is also defined by the name of the shading matrix. Heatmaps in Seaborn can be plotted by using the seaborn.heatmap() function.[1]

Heat-map is the correlation Map that is use find the correlation between features. So by using heat-map I find the important features that make a good impact on dependent variables price. So I use heat-map for feature selection.

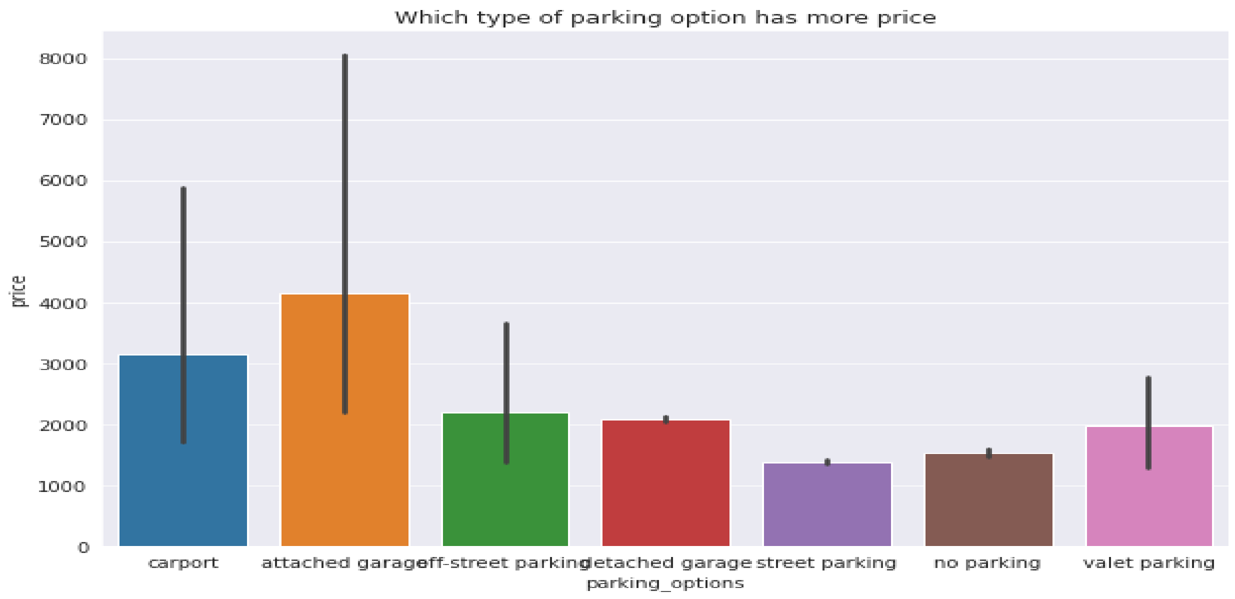


Bar-plot:

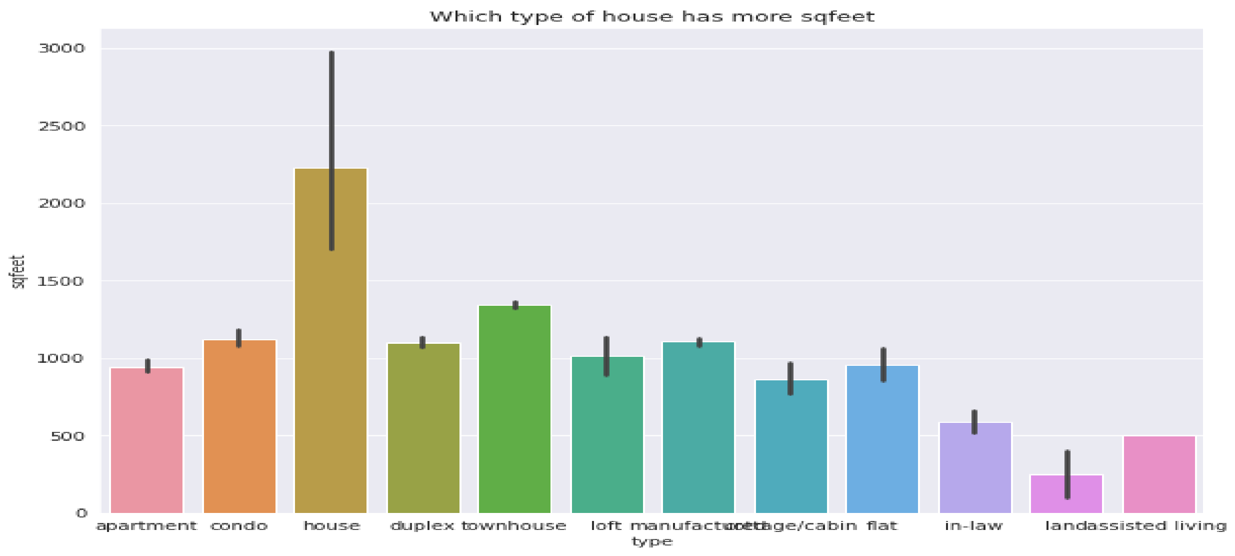
A bar plot is basically used to aggregate the categorical data according to some methods and by default it's the mean. It can also be understood as a visualization of the group by action. To use this plot we choose a categorical column for the x-axis and a numerical column for the y-axis, and we see that it creates a plot taking a mean per categorical column. [2]

Results:

In housing dataset there are many features that is relevant to dependent variable price so I do bivariate analysis in bar plot . Out of all parking option the attached garage has more price so customer want a attached garage instead of street parking and valet parking so that's why attach garage has more price. People pay very less for homes that have street parking. And people also like carport, they also pay more that have carport option for parking.

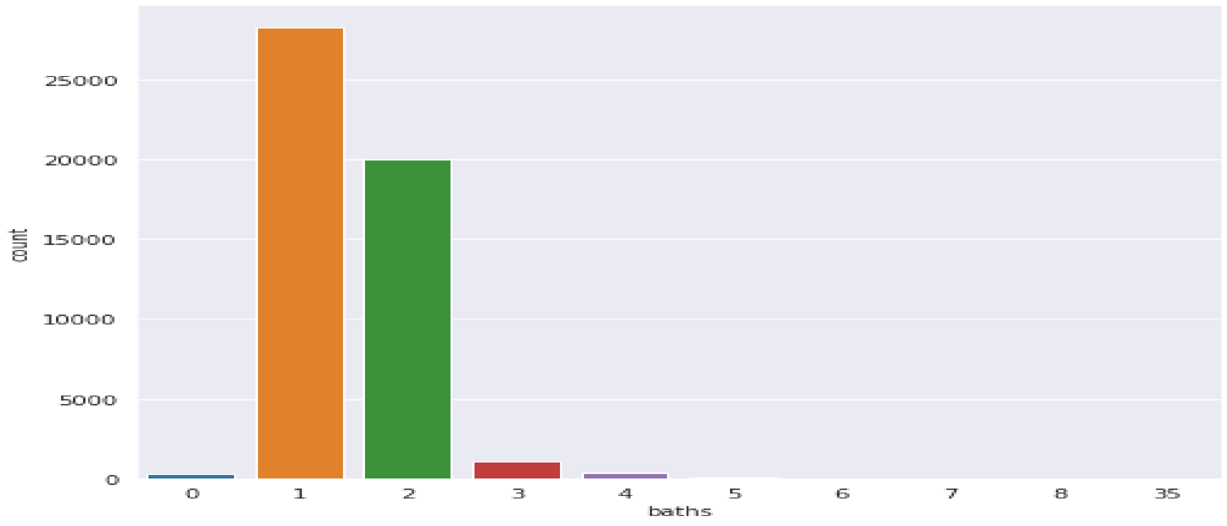


In below graph I plot the type of all houses and their sq feet. So homes take more sq feet as compared to flat, apartment and cabin. Sq feet means area so area is most important features to predict the car price. Because area is increase the price is also increase so that's why price of homes is very much high as compared to others.

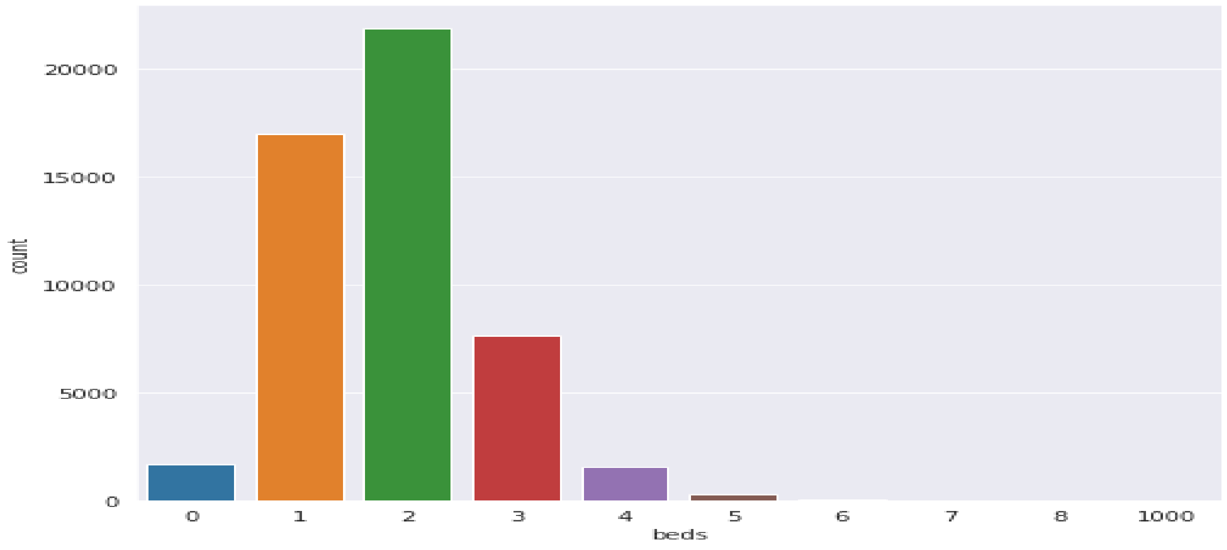


Count plot:

Count plot method is used to Show the counts of observations in each categorical bin using bars. It is used for univariate analysis so there are so many house that has only 2 and 3 baths and there are only few house that has 3 or 4 plots. This is obvious because there are many flat, apartment and cabins in our dataset so that's why there are large number of 1 and 2 plots in this.



Now we discussed the bedrooms. SO 22000 people living in 2 bedrooms because family wants small but furnished house so that's why people bought most of the houses that have 2 bed-rooms. Further 17000 peoples living in 1 beds-rooms and 7000 peoples living in 3 bedrooms so this is the trend of our data. And only few people wants 4,5 bedrooms in their homes. These are the large families so thatswhy they liked 4,5 beds rooms.



15000 houses not allowed allowed dogs in their houses and 34000 houses allowed the dogs in the houses.

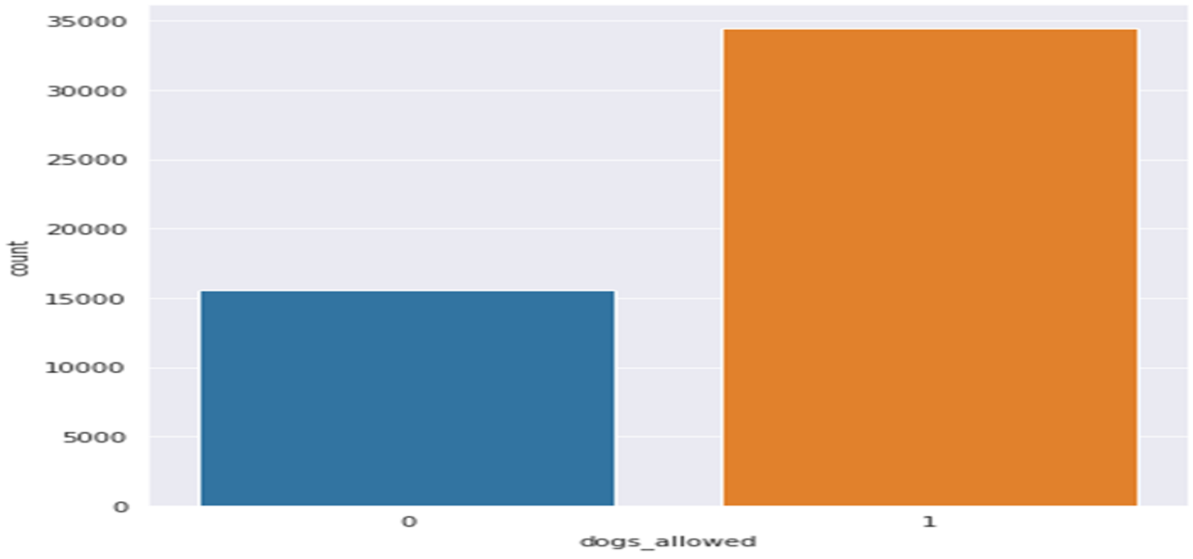
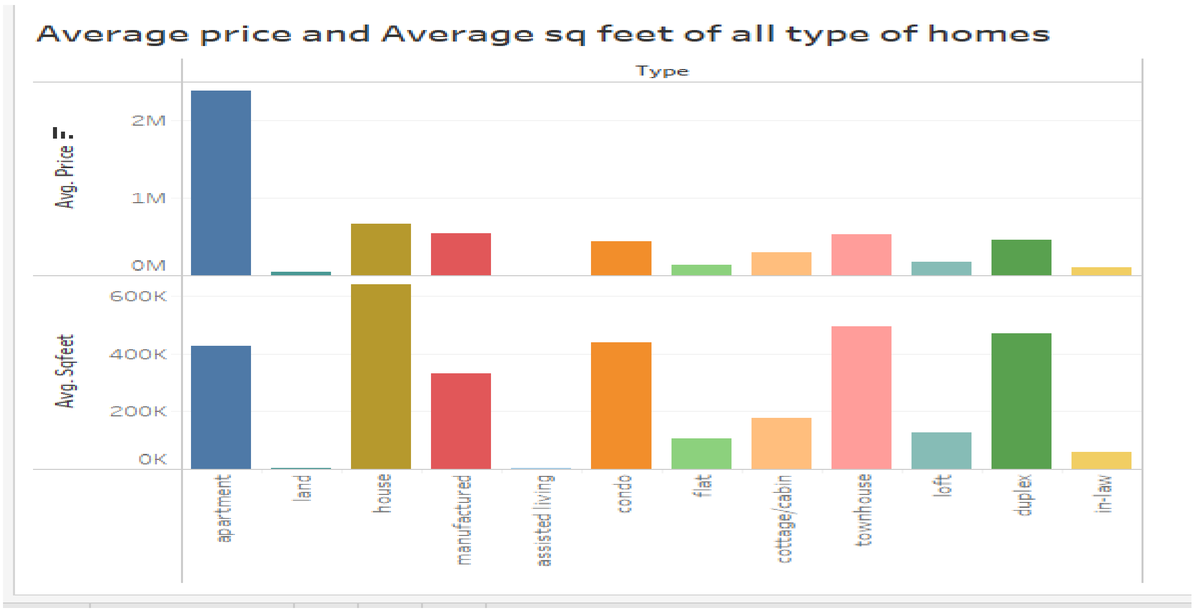
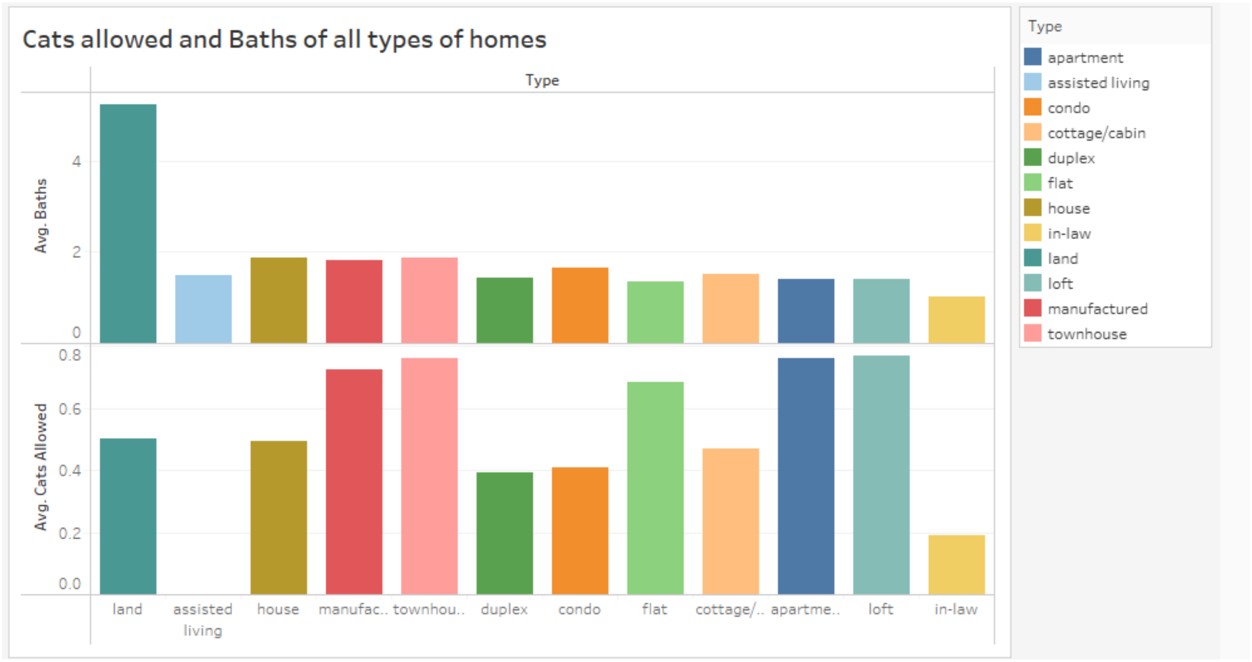


Tableau Chart:

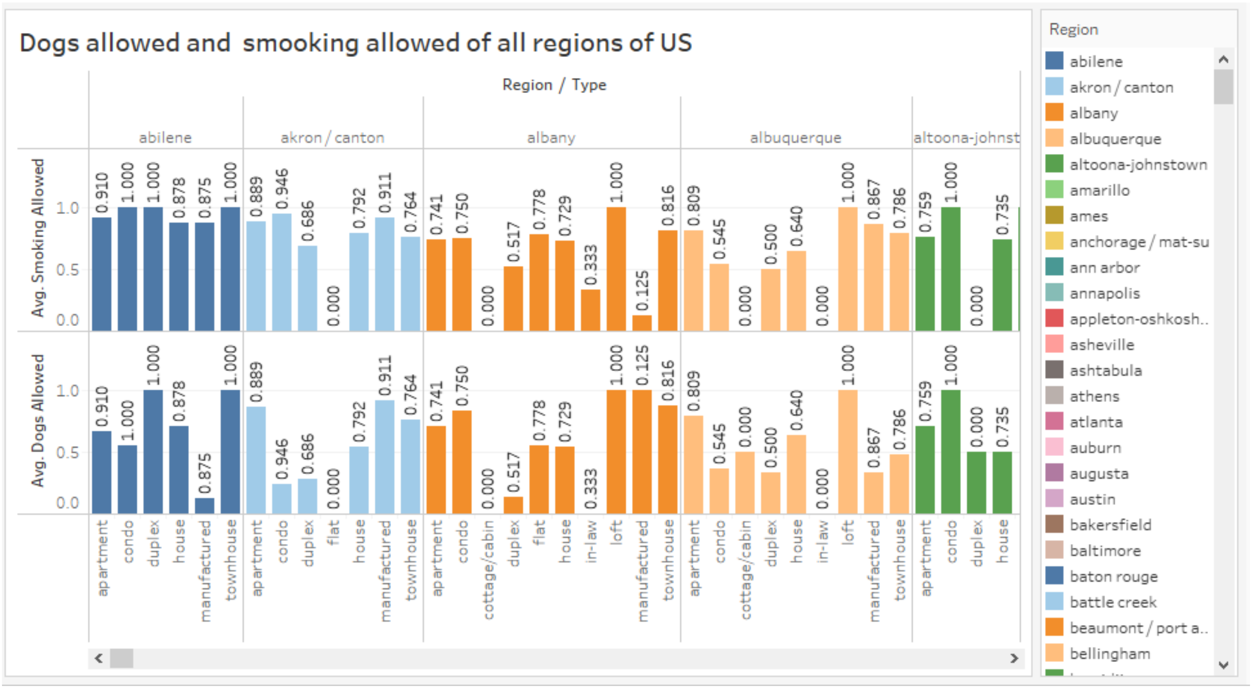
This is the tableau chart in which we see the trend of price and sq feet of all homes. I already mentioned that Price and sq feet is highly correlated and this things is also verify from below chart because when we increase the sqfeet the price is also increase because more area of houses have more prices. Average price of apartment is 4.2M and sq feet of is also 400k so similarly from below chart we see the average price and average sq feet of all cabins, homes, and flat etc.



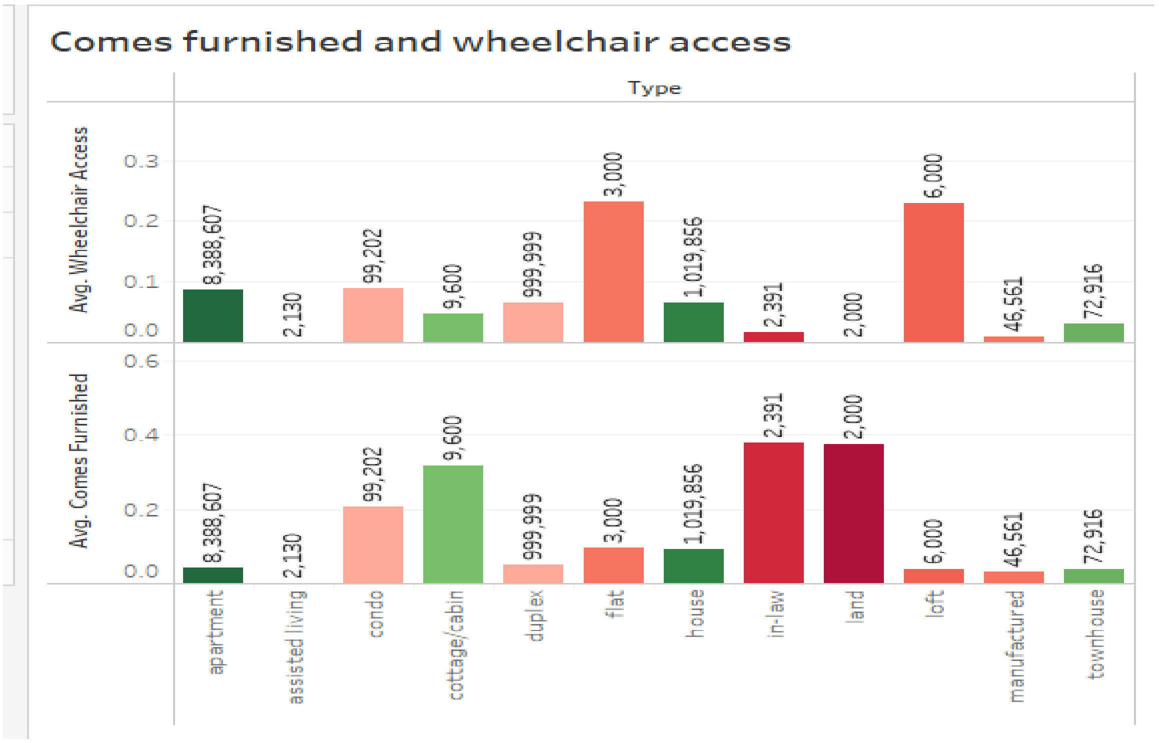
So in below tableau chart we see the Average cats allowed and Average baths of all types of homes. So there is no relationship between cats allowed and beds while we clearly see that



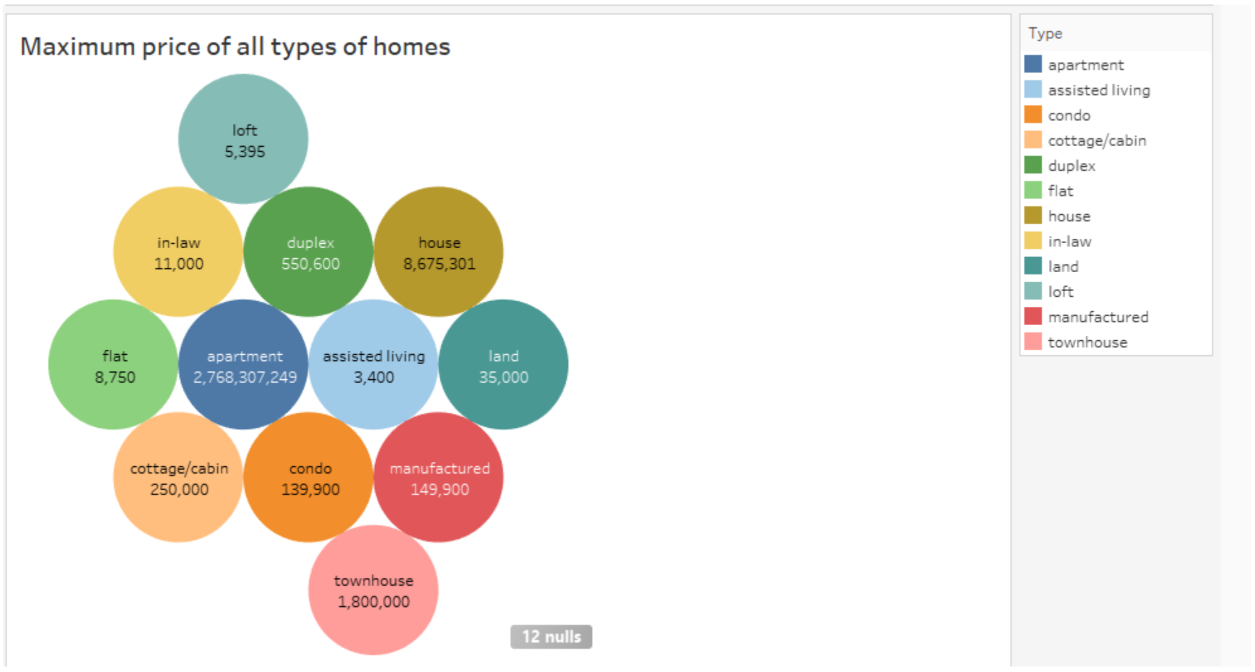
In below tableau chart we want to see the trend of dogs allowed and smoking allowed of all region of US.



In below tableau plot I see the trend of comes furnished houses and houses that have wheel chair access. I plot comes furnished and wheel chair access of all type of house.



In below chart I plot the maximum price of all types of homes in bubble chart so we clearly seen that maximum price of apartment is very high and maximum price of land and flat is very low as compared to apartment and cottage.



4 Data Preprocessing:

Data preprocessing is a step in the data mining and data analysis process that takes raw data and transforms it into a format that can be understood and analyzed by computers and machine learning.

Raw, real-world data in the form of text, images, video, etc., is messy. Not only may it contain errors and inconsistencies, but it is often incomplete, and doesn't have a regular, uniform design.

Machines like to process nice and tidy information – they read data as 1s and 0s. So calculating structured data, like whole numbers and percentages is easy. However, unstructured data, in the form of text and images must first be cleaned and formatted before analysis.

Let's take a look at the established steps you'll need to go through to make sure your data is successfully preprocessed.

1. Data Cleaning
2. Data Transformation
3. Data Reduction

1 Data Cleaning:

Data Cleaning is the process of adding missing data and correcting, repairing, or removing incorrect or irrelevant data from a data set. Dating cleaning is the most important step of preprocessing because it will ensure that your data is ready to go for your downstream needs.

Data cleaning will correct all of the inconsistent data you uncovered in your data quality assessment. Depending on the kind of data you're working with, there are number of possible cleaners you'll need to run your data through.

Missing data

There are a number of ways to correct for missing data, but the two most common are:

- **Ignore the tuples:** A tuple is an ordered list or sequence of numbers or entities. If multiple values are missing within tuples, you may simply discard the tuples with that missing information. This is only recommended for large data sets, when a few ignored tuples won't harm further analysis.
- **Manually fill in missing data:** This can be tedious, but is definitely necessary when working with smaller data sets.

2 Data Transformation

With data cleaning, we've already begun to modify our data, but data transformation will begin the process of turning the data into the proper format(s) you'll need for analysis and other downstream processes.

This generally happens in one or more of the below:

1. Aggregation
2. Normalization
3. Feature selection

- **Aggregation:** Data aggregation combines all of your data together in a uniform format.
- **Normalization:** Normalization scales your data into a regularized range so that you can compare it more accurately. For example, if you're comparing employee loss or gain within a number of companies (some with just a dozen employees and some with 200+), you'll have to scale them within a specified range, like -1.0 to 1.0 or 0.0 to 1.0.
- **Feature selection:** Feature selection is the process of deciding which variables (features, characteristics, categories, etc.) are most important to your analysis. These features will be used to train ML models. It's important to remember, that the more features you choose to use, the longer the training process and, sometimes, the less accurate your results, because some feature characteristics may overlap or be less present in the data.

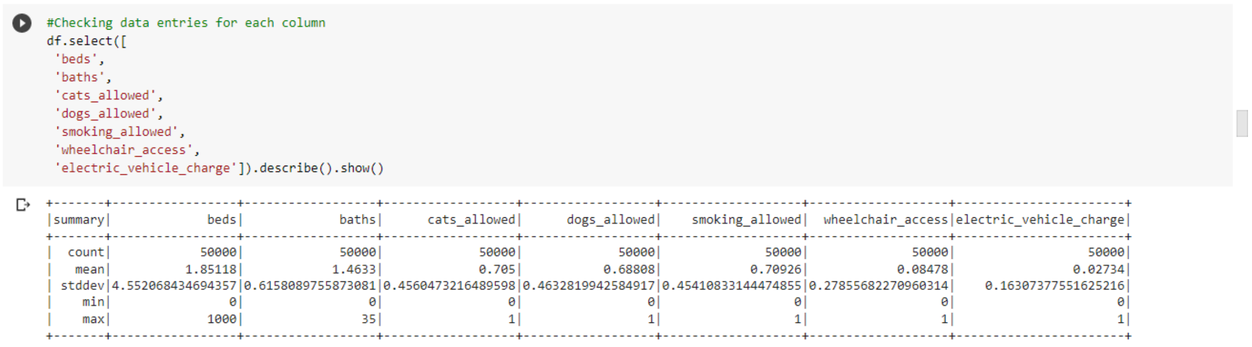
3 Data Reduction:

The more data you're working with, the harder it will be to analyze, even after cleaning and transforming it. Depending on your task at hand, you may actually have more data than you need. Especially when working with text analysis, much of regular human speech is superfluous or irrelevant to the needs of the researcher. Data reduction not only makes the analysis easier and more accurate, but cuts down on data storage.

It will also help identify the most important features to the process at hand.

- **Attribute selection:** Similar to discreditization, attribute selection can fit your data into smaller pools. It, essentially, combines tags or features, so that tags like *male/female* and *professor* could be combined into *male professor/female professor*.
- **Numerosity reduction:** This will help with data storage and transmission. You can use a regression model, for example, to use only the data and variables that are relevant to your analysis.
- **Dimensionality reduction:** This, again, reduces the amount of data used to help facilitate analysis and downstream processes. Algorithms like KNN use pattern recognition to combine similar data and make it more manageable. [3]

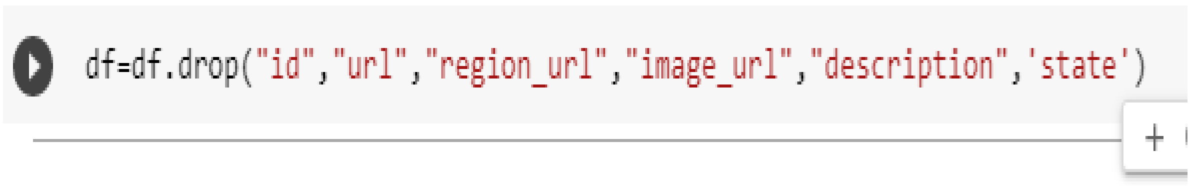
Statistics of the dataset



```
#Checking data entries for each column
df.select([
 'beds',
 'baths',
 'cats_allowed',
 'dogs_allowed',
 'smoking_allowed',
 'wheelchair_access',
 'electric_vehicle_charge']).describe().show()
```

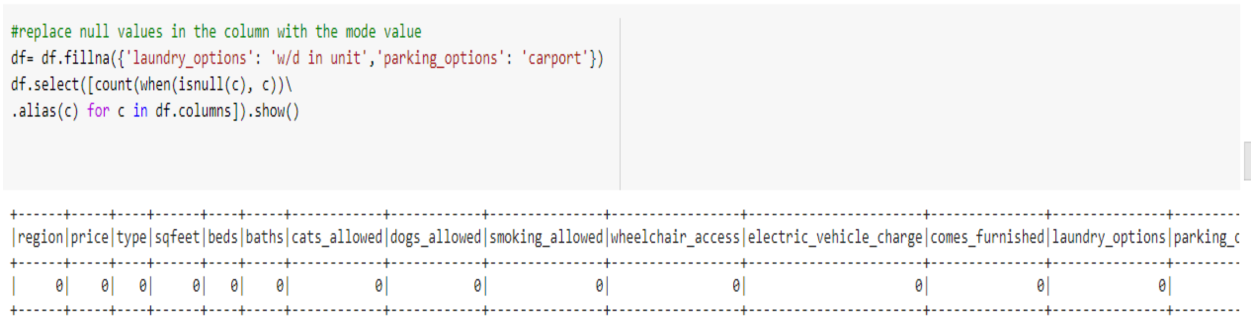
```
+-------+-----------------+------------------+------------------+------------------+-------------------+-------------------+-----------------------+
|summary|             beds|             baths|      cats_allowed|      dogs_allowed|    smoking_allowed|  wheelchair_access|electric_vehicle_charge|
+-------+-----------------+------------------+------------------+------------------+-------------------+-------------------+-----------------------+
|  count|            50000|             50000|             50000|             50000|              50000|              50000|                  50000|
|   mean|          1.85118|            1.4633|             0.705|           0.68808|            0.70926|            0.08478|                0.02734|
| stddev|4.552068434694357|0.6158089755873081|0.4560473216489598|0.4632819942584917|0.45410833144474855|0.27855682270960314|    0.16307377551625216|
|    min|                0|                 0|                 0|                 0|                  0|                  0|                      0|
|    max|             1000|                35|                 1|                 1|                  1|                  1|                      1|
+-------+-----------------+------------------+------------------+------------------+-------------------+-------------------+-----------------------+
```

Remove unwanted columns



```
df=df.drop("id","url","region_url","image_url","description",'state')
```

Handle missing values



```
#replace null values in the column with the mode value
df= df.fillna({'laundry_options': 'w/d in unit','parking_options': 'carport'})
df.select([count(when(isnull(c), c))\
.alias(c) for c in df.columns]).show()
```

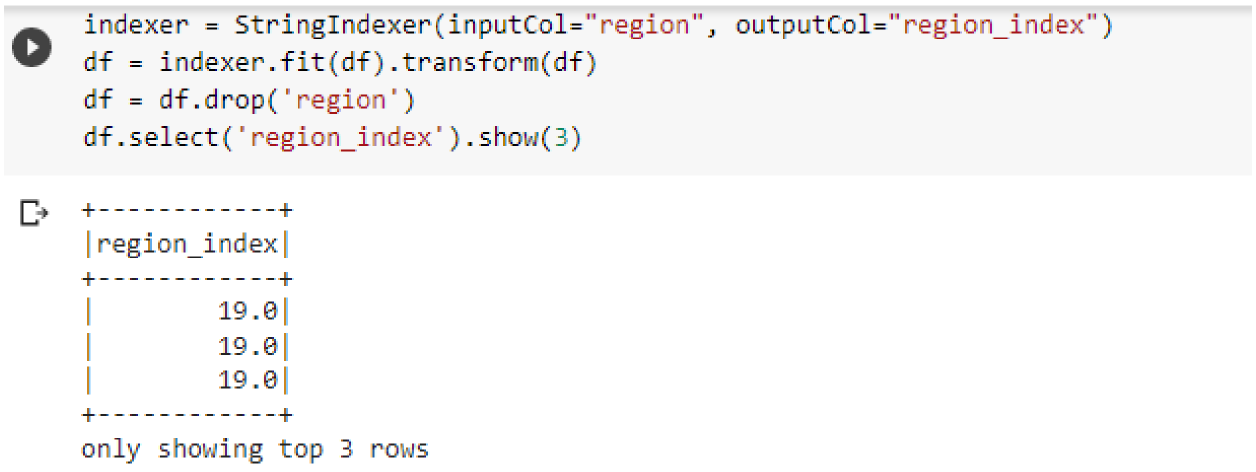
```
+------+-----+----+------+----+-----+------------+------------+---------------+-----------------+-----------------------+--------------+---------------+--------
|region|price|type|sqfeet|beds|baths|cats_allowed|dogs_allowed|smoking_allowed|wheelchair_access|electric_vehicle_charge|comes_furnished|laundry_options|parking_c
+------+-----+----+------+----+-----+------------+------------+---------------+-----------------+-----------------------+--------------+---------------+--------
|     0|    0|   0|     0|   0|    0|           0|           0|              0|                0|                      0|             0|              0|
+------+-----+----+------+----+-----+------------+------------+---------------+-----------------+-----------------------+--------------+---------------+--------
```

String indexer:

Class pyspark.ml.feature.StringIndexer(inputCol=None, outputCol=None, inputCols=None, outputCols=None, handleInvalid='error', stringOrderType='frequencyDesc') — StringIndexer encodes a string column of labels to a column of label indices. If the input column is numeric, we cast it to string and index the string values. The indices are in [0, numLabels).[4] By default, this is ordered by label frequencies so the most frequent label gets index 0. The ordering behavior is controlled by setting stringOrderType. Its default value is 'frequencyDesc'. In case of equal frequency when under frequencyDesc/Asc, the strings are further sorted alphabetically [4]

Label Encoding with string indexer:

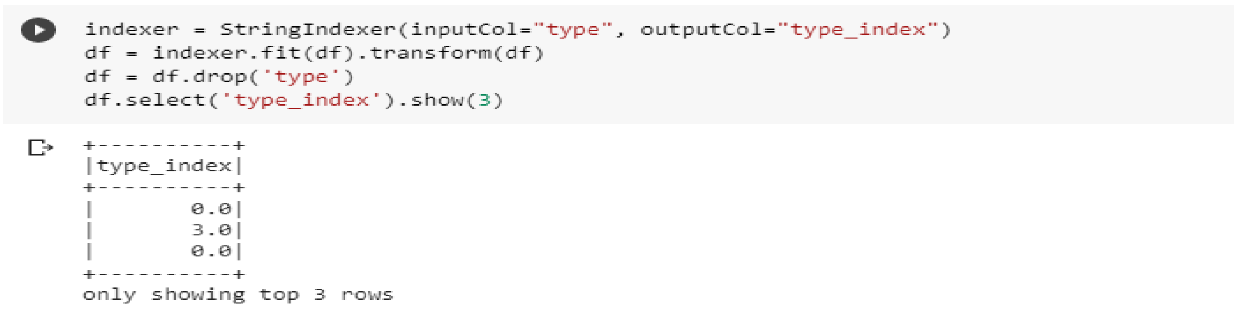
Apply String indexer to region columns:



```
indexer = StringIndexer(inputCol="region", outputCol="region_index")
df = indexer.fit(df).transform(df)
df = df.drop('region')
df.select('region_index').show(3)
```

```
+------------+
|region_index|
+------------+
|        19.0|
|        19.0|
|        19.0|
+------------+
only showing top 3 rows
```

Apply string indexer to Type columns:



```
indexer = StringIndexer(inputCol="type", outputCol="type_index")
df = indexer.fit(df).transform(df)
df = df.drop('type')
df.select('type_index').show(3)
```

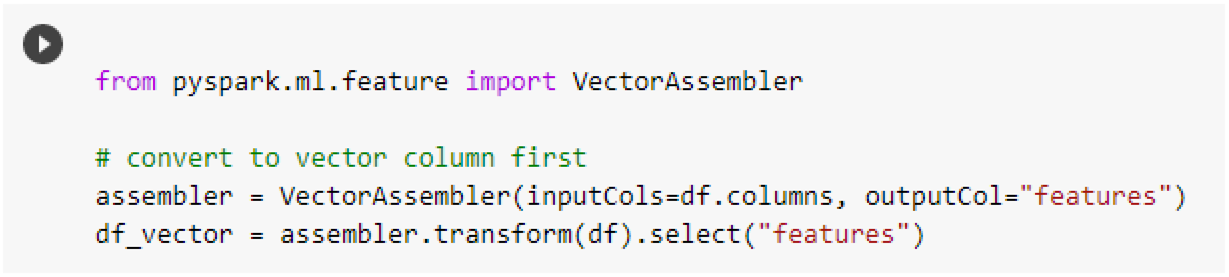
```
+----------+
|type_index|
+----------+
|       0.0|
|       3.0|
|       0.0|
+----------+
only showing top 3 rows
```

Vector Assembler:

VectorAssembler is a transformer that combines a given list of columns into a single vector column. It is useful for combining raw features and features generated by different feature transformers into a single feature vector, in order to train ML models like logistic regression and decision trees. VectorAssembler accepts the following input column types: all numeric types, boolean type, and vector type. In each row, the values of the input columns will be concatenated into a vector in the specified order [5]

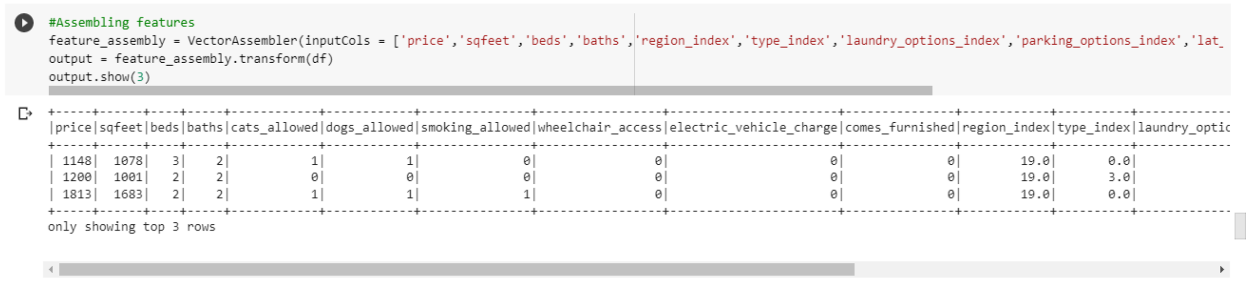
class pyspark.ml.feature.VectorAssembler(inputCols=None, outputCol=None, handleInvalid='error'): VectorAssembler is a transformer that combines a given list of columns into a single vector column.It is useful for combining raw features and features generated by different feature transformers into a single feature vector, in order to train ML models like logistic regression and decision trees. VectorAssembler accepts the following input column types: all numeric types, boolean type, and vector type. In each row, the values of the input columns will be concatenated into a vector in the specified order.

Note: For VectorAssembler, we do not need StringIndexer and OneHotEncoder, if your data have all numeric values. In this example we have string columns, so we are using StringIndexer and OneHotEncoder. [6]



```
from pyspark.ml.feature import VectorAssembler

# convert to vector column first
assembler = VectorAssembler(inputCols=df.columns, outputCol="features")
df_vector = assembler.transform(df).select("features")
```

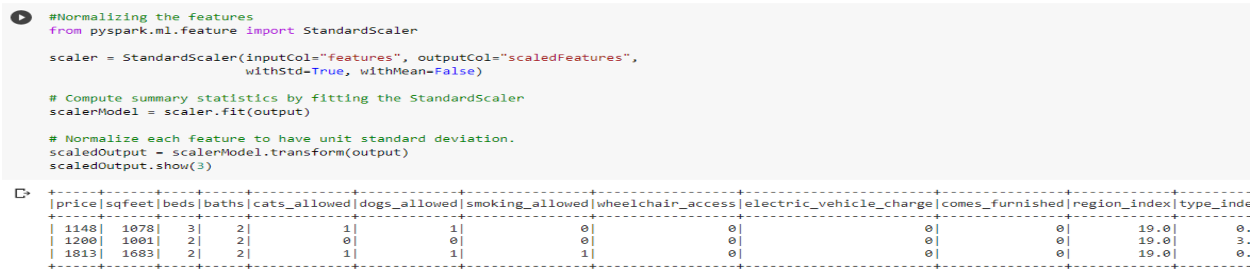
Assembling relevant features:



```
#Assembling features
feature_assembly = VectorAssembler(inputCols = ['price','sqfeet','beds','baths','region_index','type_index','laundry_options_index','parking_options_index','lat_
output = feature_assembly.transform(df)
output.show(3)
```

```
+-----+------+----+-----+------------+------------+---------------+-----------------+-----------------------+---------------+------------+----------+------------
|price|sqfeet|beds|baths|cats_allowed|dogs_allowed|smoking_allowed|wheelchair_access|electric_vehicle_charge|comes_furnished|region_index|type_index|laundry_optic
+-----+------+----+-----+------------+------------+---------------+-----------------+-----------------------+---------------+------------+----------+------------
| 1148|  1078|   3|    2|           1|           1|              0|                0|                      0|              0|        19.0|       0.0|
| 1200|  1001|   2|    2|           0|           0|              0|                0|                      0|              0|        19.0|       3.0|
| 1813|  1683|   2|    2|           1|           1|              1|                0|                      0|              0|        19.0|       0.0|
+-----+------+----+-----+------------+------------+---------------+-----------------+-----------------------+---------------+------------+----------+------------
only showing top 3 rows
```

Data Normalization:



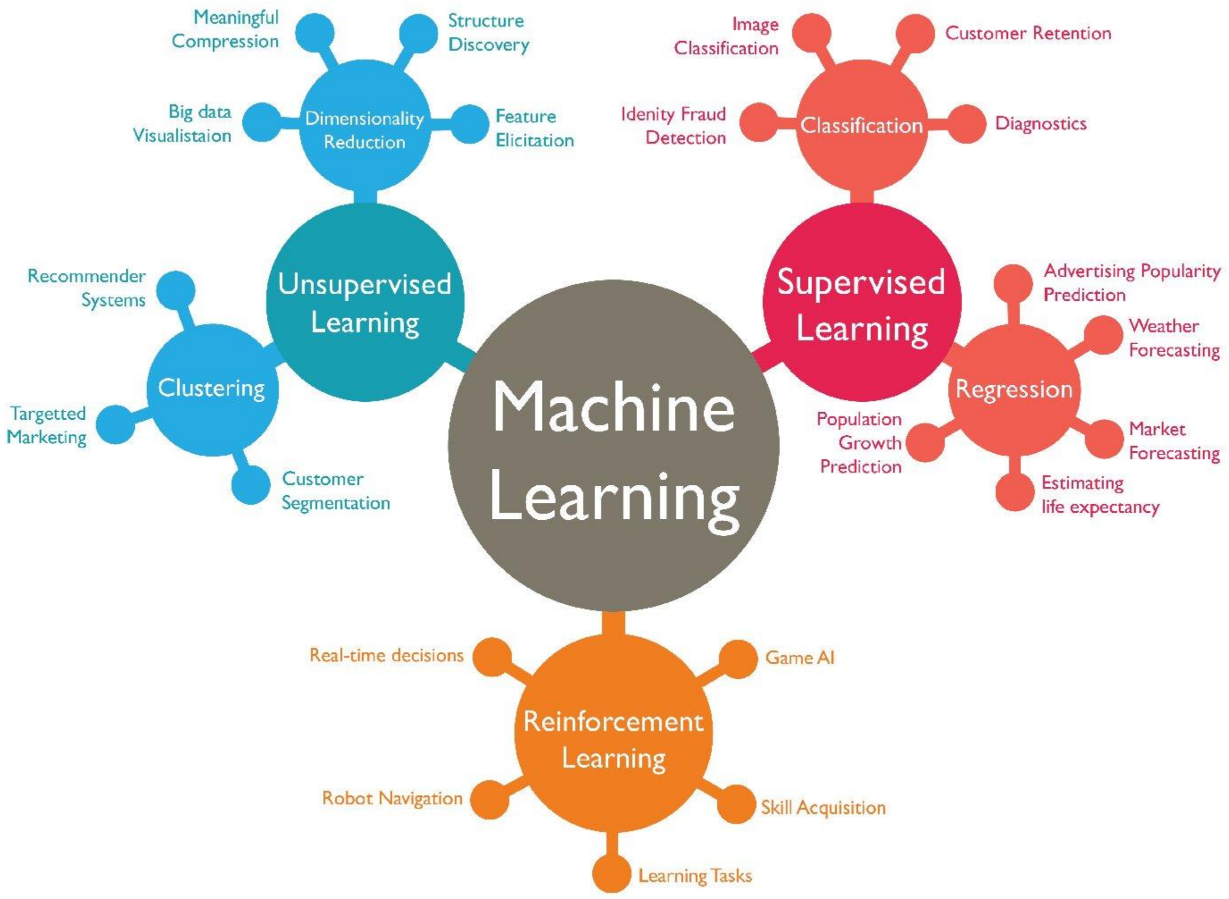
```
#Normalizing the features
from pyspark.ml.feature import StandardScaler

scaler = StandardScaler(inputCol="features", outputCol="scaledFeatures",
                        withStd=True, withMean=False)

# Compute summary statistics by fitting the StandardScaler
scalerModel = scaler.fit(output)

# Normalize each feature to have unit standard deviation.
scaledOutput = scalerModel.transform(output)
scaledOutput.show(3)
```

```
+-----+------+----+-----+------------+------------+---------------+-----------------+-----------------------+---------------+------------+---------
|price|sqfeet|beds|baths|cats_allowed|dogs_allowed|smoking_allowed|wheelchair_access|electric_vehicle_charge|comes_furnished|region_index|type_inde
+-----+------+----+-----+------------+------------+---------------+-----------------+-----------------------+---------------+------------+---------
| 1148|  1078|   3|    2|           1|           1|              0|                0|                      0|              0|        19.0|       0.
| 1200|  1001|   2|    2|           0|           0|              0|                0|                      0|              0|        19.0|       3.
| 1813|  1683|   2|    2|           1|           1|              1|                0|                      0|              0|        19.0|       0.
+-----+------+----+-----+------------+------------+---------------+-----------------+-----------------------+---------------+------------+---------
```

5 Machine Learning

It is now possible to Train Machines with a **Data-Driven** approach. On a wider spectrum, if you think of Artificial Intelligence as the main umbrella, Machine Learning is a subset of Artificial Intelligence. Machine Learning, a set of Algorithms, gives Machines or Computers the ability to learn from data on their own without any human intervention.

The idea behind Machine Learning is that you teach and Train Machines by feeding them data and defining features. Computers **learn, grow, adapt, and develop** by themselves when they are fed with new and relevant data, without relying on explicit programming. Without data, there is very little that Machines can learn. The Machine observes the dataset, identifies patterns in it, learns automatically from the behavior, and makes predictions.



https://www.wordstream.com/blog/ws/2017/07/28/machine-learning-applications

It is the Machine Learning technology that Online Recommendation Engines use to offer relevant recommendations to the user, be it **YouTube Video Recommendations** or **Facebook Friend Recommendations**. One of the most recent technologies, **Google's Self Driving Car** also makes use of Machine Learning Algorithms to understand the patterns and definitions, learn automatically, and execute the operation. [7]

Supervised Learning VS Unsupervised Learning:

The main difference between supervised vs unsupervised learning is the need for labelled training data. Supervised machine learning relies on labelled input and output training data, whereas unsupervised learning processes un-labelled or raw data. In supervised machine learning the model learns the relationship between the labelled input and output data. Models are finetuned until they can accurately predict the outcomes of unseen data. However, labelled training data will often be resource intensive to create. Unsupervised machine learning on the other hand learns from un-labelled raw training data. An unsupervised model will learn relationships and patterns within this unlabelled dataset, so is often used to discover inherent trends in a given dataset.

So overall, supervised and unsupervised machine learning are different in the approach to training and the data the model learns from. But as a result, they also differ in their final application and specific strengths. Supervised machine learning models are generally used to predict outcomes for unseen data. This could be predicting fluctuations in house prices or understanding the sentiment of a message.

Models are also used to classify unseen data against learned patterns. On the other hand, unsupervised machine learning techniques are generally used to understand patterns and trends within unlabelled data. This could be clustering data due to similarities or differences, or identifying underlying patterns within datasets. Unsupervised machine learning can be used to cluster customer data in marketing campaigns, or to detect anomalies and outliers.

Examples of supervised machine learning include:

- Classification, identifying input data as part of a learned group.
- Regression, predicting outcomes from continuously changing data.

Examples of unsupervised machine learning include:

- Clustering, grouping together data points with similar data.
- Association, understanding how certain data features connect with other features. [8]

Regression:

The common use of supervised machine learning models is in predictive analytics. Regression is commonly used as the process for a machine learning model to predict continuous outcomes. A supervised machine learning model will learn to identify patterns and relationships within a labelled training dataset. Once the relationship between input data and expected output data is understood, new and unseen data can be processed by the model. Regression is therefore used in predictive machine learning models, which could be used to:

- Forecast stock or trading outcomes and market fluctuations, a key role of machine learning in finance.
- Predict the success of marketing campaigns so organizations can assign and refine resources.
- Forecast changes in market value in sectors like retail or the housing market.

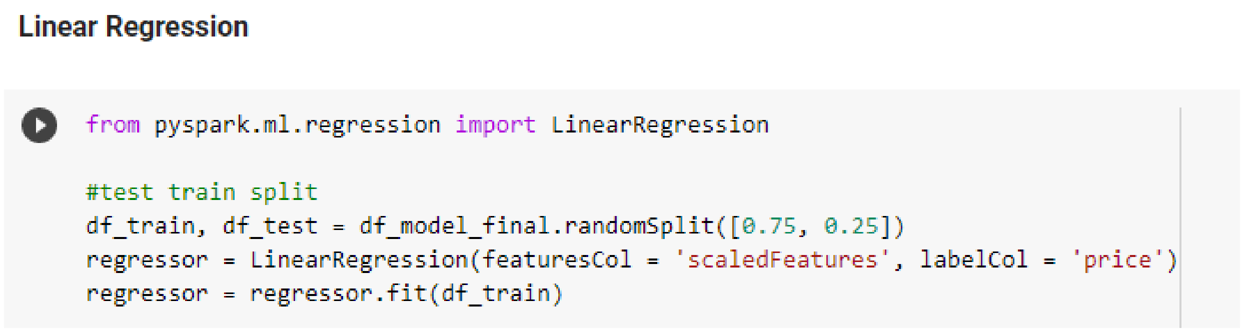
- Predict changes in health trends in a demographic or area.

Common algorithms used in supervised learning regression include:

1. Multiple Linear regression
2. Decision tree Regressor
3. Random Forest Regressor etc

Multiple Linear Regression

Multiple Linear Regression is a popular type of regression approach and is used to predict target output from more than one input variable. A linear connection between the input and target output should be present. Once a model has been trained on the relationship between the input and target output, it can be used to make predictions on new data. Examples might be predicting salary based on age, Children, Martial status and gender. [8]



Linear Regression

```
from pyspark.ml.regression import LinearRegression

#test train split
df_train, df_test = df_model_final.randomSplit([0.75, 0.25])
regressor = LinearRegression(featuresCol = 'scaledFeatures', labelCol = 'price')
regressor = regressor.fit(df_train)
```

Evaluation Metrics:

Model evaluation is very important in data science. It helps you to understand the performance of your model and makes it easy to present your model to other people. There are many different evaluation metrics out there but only some of them are suitable to be used for regression. This article will cover the different metrics for the regression model and the difference between them. Hopefully, after you read this post, you are clear on which metrics to apply to your future regression model.

Every time when I tell my friends: “Hey, I have built a machine learning model to predict XXX.” Their first reaction would be: “Cool, so what is the accuracy of your model prediction?” Well, unlike classification, accuracy in a regression model is slightly harder to illustrate. It is impossible for you to predict the exact value but rather **how close your prediction is against the real value**.

There are 3 main metrics for model evaluation in regression:

1. R Square/Adjusted R Square

2. Mean Square Error(MSE)/Root Mean Square Error(RMSE)

3. Mean Absolute Error(MAE)

R Square/Adjusted R Square

R Square measures how much variability in dependent variable can be explained by the model. It is the square of the Correlation Coefficient(R) and that is why it is called R Square.

$$R^2 = 1 - \frac{SS_{Regression}}{SS_{Total}} = 1 - \frac{\sum_i (y_i - \hat{y}_i)^2}{\sum_i (y_i - \bar{y})^2}$$

R square formula

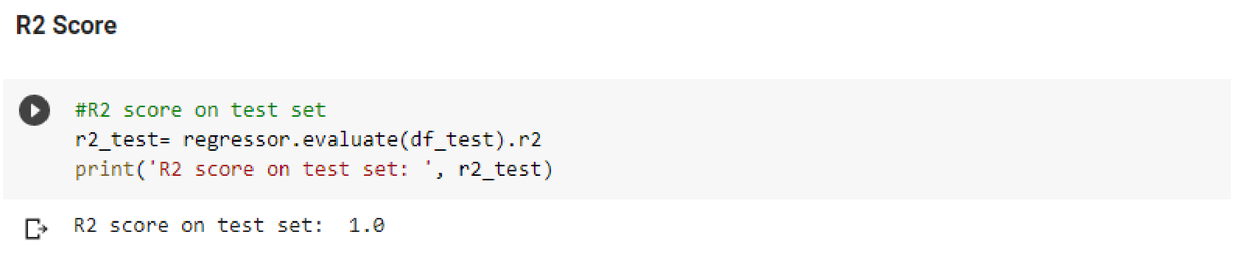
R Square is calculated by the sum of squared of prediction error divided by the total sum of the square which replaces the calculated prediction with mean. R Square value is between 0 to 1 and a bigger value indicates a better fit between prediction and actual value.
R Square is a good measure to determine how well the model fits the dependent variables. **However, it does not take into consideration of overfitting problem**. If your regression model has many independent variables, because the model is too complicated, it may fit very well to the training data but performs badly for testing data
While R Square is a relative measure of how well the model fits dependent variables, Mean Square Error is an absolute measure of the goodness for the fit.

$$MSE = \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2$$

Mean Square Error formula

MSE is calculated by the sum of square of prediction error which is real output minus predicted output and then divide by the number of data points. It gives you an absolute number on how much your predicted results deviate from the actual number. You cannot interpret many insights from one single result but it gives you a real number to compare against other model results and help you select the best regression model.

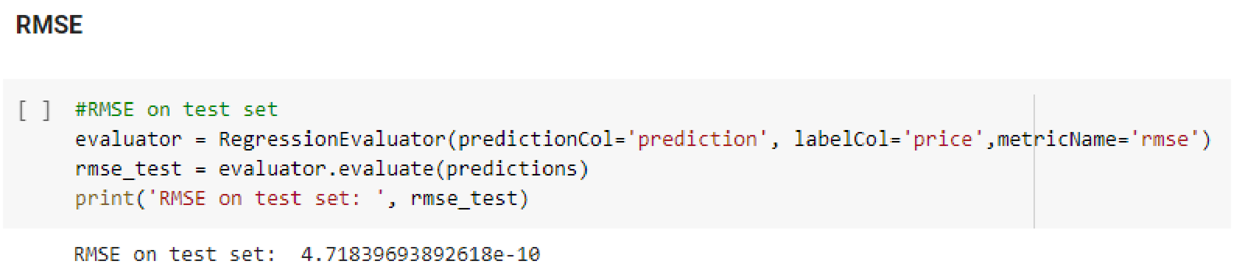
Root Mean Square Error(RMSE) is the square root of MSE. It is used more commonly than MSE because firstly sometimes MSE value can be too big to compare easily. [9]



R2 Score

```
#R2 score on test set
r2_test= regressor.evaluate(df_test).r2
print('R2 score on test set: ', r2_test)
```

```
R2 score on test set:  1.0
```



RMSE

```
#RMSE on test set
evaluator = RegressionEvaluator(predictionCol='prediction', labelCol='price',metricName='rmse')
rmse_test = evaluator.evaluate(predictions)
print('RMSE on test set: ', rmse_test)
```

```
RMSE on test set:  4.71839693892618e-10
```

Unsupervised Learning:

We already discussed in detail what is unsupervised learning.

Clustering:

Clustering is the grouping together of data points into a determined number of categories depending on similarities (or differences) between data points. This way raw and unlabelled data can be processed and clustered depending on the patterns within the dataset. Hyperparameters set by the data scientist will usually define the overall count of clusters.

Clustering is a popular use of unsupervised learning models and can be used to understand trends and groupings in raw data. The approach can also highlight data points that sit outside of the groupings, making it an important tool for anomaly detection. [8]

Clustering as an approach can be used to:

- Segment audience or customer data into groups in marketing environments.
- Perform initial exploratory analysis on raw datasets to understand the grouping of data points.
- Detect outliers and anomalies that sit outside of clustered data.

Common approaches to unsupervised learning clustering include:

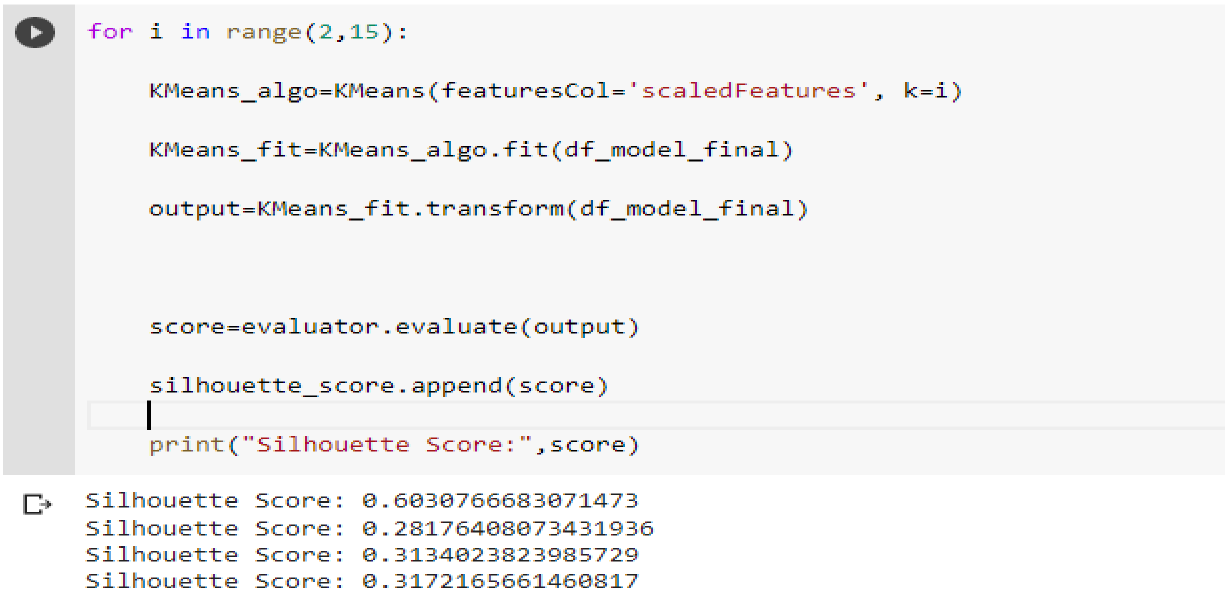
- K-means clustering

- Gaussian Mixture Models

K means Clustering:

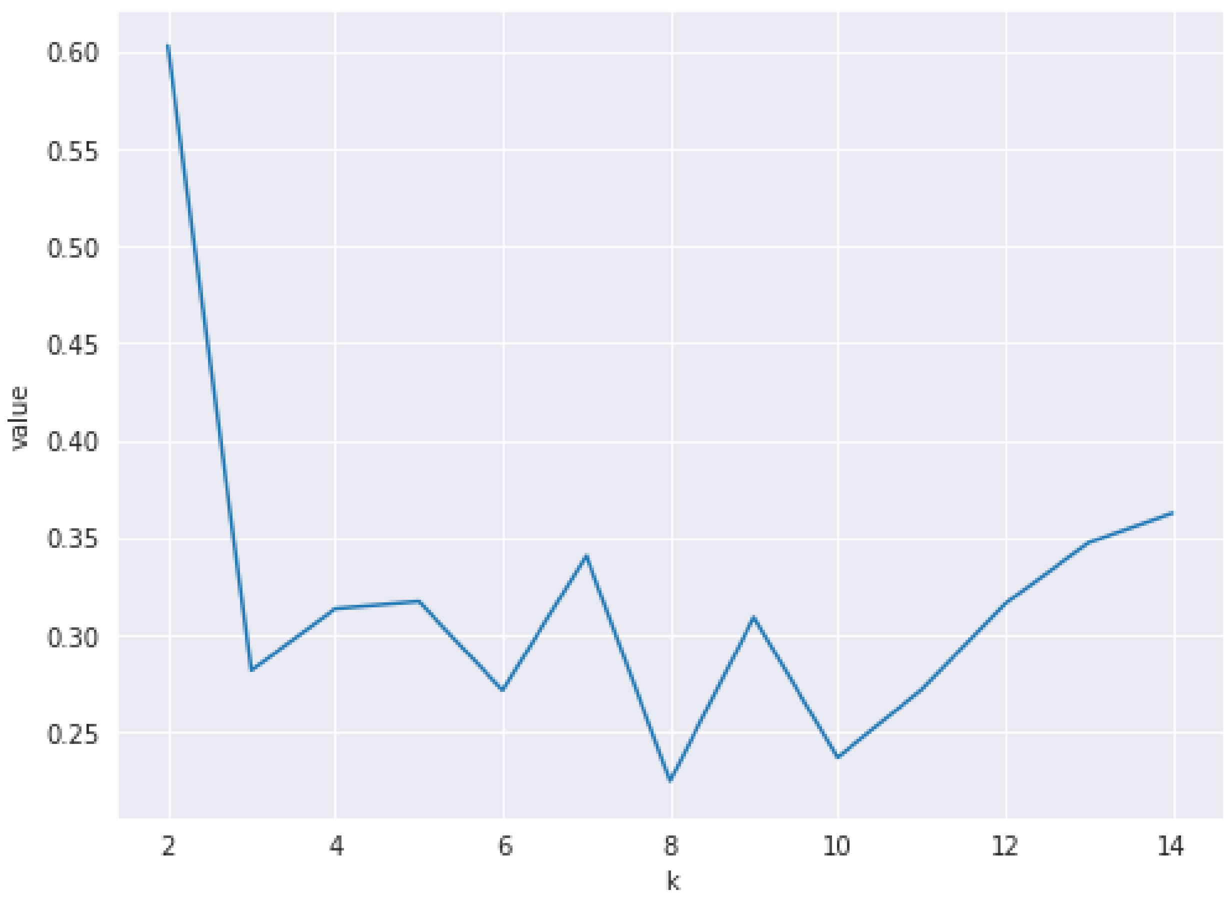
K-means clustering is a popular method for clustering data. K represents the count of clusters, set by the data scientist. Clusters are defined by the distance from the center of each grouping. A higher count of clusters means more granular groupings, and a lower count of clusters means less granular groupings. This method can be used to identify exclusive or overlapping clusters. Exclusive clustering means each data point can belong to only one cluster. Overlapping clustering means data can be within multiple clusters. [8]

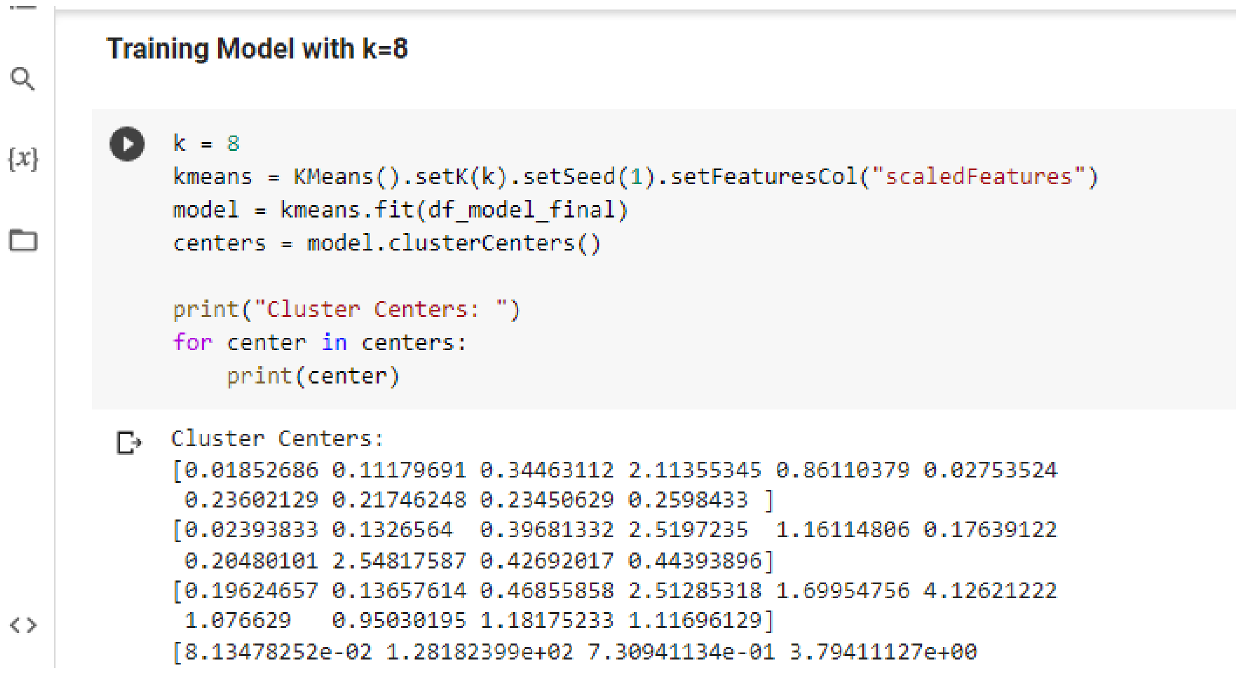
Find best K value:



```
for i in range(2,15):

    KMeans_algo=KMeans(featuresCol='scaledFeatures', k=i)

    KMeans_fit=KMeans_algo.fit(df_model_final)

    output=KMeans_fit.transform(df_model_final)



    score=evaluator.evaluate(output)

    silhouette_score.append(score)

    print("Silhouette Score:",score)
```

```
Silhouette Score: 0.6030766683071473
Silhouette Score: 0.28176408073431936
Silhouette Score: 0.3134023823985729
Silhouette Score: 0.3172165661460817
```

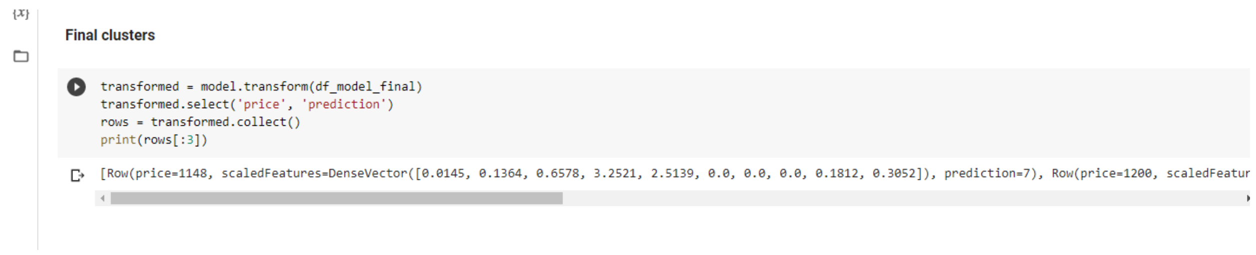




Training Model with k=8

```
k = 8
kmeans = KMeans().setK(k).setSeed(1).setFeaturesCol("scaledFeatures")
model = kmeans.fit(df_model_final)
centers = model.clusterCenters()

print("Cluster Centers: ")
for center in centers:
    print(center)
```

```
Cluster Centers:
[0.01852686 0.11179691 0.34463112 2.11355345 0.86110379 0.02753524
 0.23602129 0.21746248 0.23450629 0.2598433 ]
[0.02393833 0.1326564  0.39681332 2.5197235  1.16114806 0.17639122
 0.20480101 2.54817587 0.42692017 0.44393896]
[0.19624657 0.13657614 0.46855858 2.51285318 1.69954756 4.12621222
 1.076629   0.95030195 1.18175233 1.11696129]
[8.13478252e-02 1.28182399e+02 7.30941134e-01 3.79411127e+00
```



Final clusters

```
transformed = model.transform(df_model_final)
transformed.select('price', 'prediction')
rows = transformed.collect()
print(rows[:3])
```

```
[Row(price=1148, scaledFeatures=DenseVector([0.0145, 0.1364, 0.6578, 3.2521, 2.5139, 0.0, 0.0, 0.0, 0.1812, 0.3052]), prediction=7), Row(price=1200, scaledFeatur
```

Word count: 3606

Link to notebooks:

https://colab.research.google.com/drive/1nb2lspZcf1sJvRGIdmO27XcPFTdktrSY?usp=sharing

References:

[1] **https://www.geeksforgeeks.org/seaborn-heatmap-a-comprehensive-guide/**

[2] **https://www.geeksforgeeks.org/seaborn-barplot-method-in-python/**

[3] https://monkeylearn.com/blog/data-preprocessing/

[4] https://medium.com/@nutanbhogendrasharma/role-of-stringindexer-and-pipelines-in-pyspark-ml-feature-b79085bb8a6c

[5] https://george-jen.gitbook.io/data-science-and-apache-spark/vectorassembler

[6] https://medium.com/@nutanbhogendrasharma/feature-transformer-vectorassembler-in-pyspark-ml-feature-part-3-b3c2c3c93ee9

[7] https://hevodata.com/learn/machine-learning-in-data-science-2/#mac

[8] https://www.seldon.io/supervised-vs-unsupervised-learning-explained#:~:text=The%20main%20difference%20between%20supervised,processes%20unlabelled%20or%20raw%20data.

[9] https://towardsdatascience.com/what-are-the-best-metrics-to-evaluate-your-regression-model-418ca481755b