BSc. (Hons.) Data Science Degree.

7081 CEM - Business Simulation.

Module leader: Mr. Chameera De Silva

ADDS21.2P

11 Feb 2023

Jayasanka Hettiarachchi
Member ID: 017

ACKNOWLEDGEMENT

I would like to express my sincere gratitude to to Mr. Chameera De Silva (B.Eng. (Hons.), Chartered Engineer (SL)), our ML lecturer, my supervisor, for his invaluable guidance, support, and encouragement throughout the duration of learning ML01 & ML 02, His passion for teaching and deep knowledge of the subject made every session both enjoyable and enlightening. I am also grateful for his unwavering support and his belief in me which allowed me to complete this report.

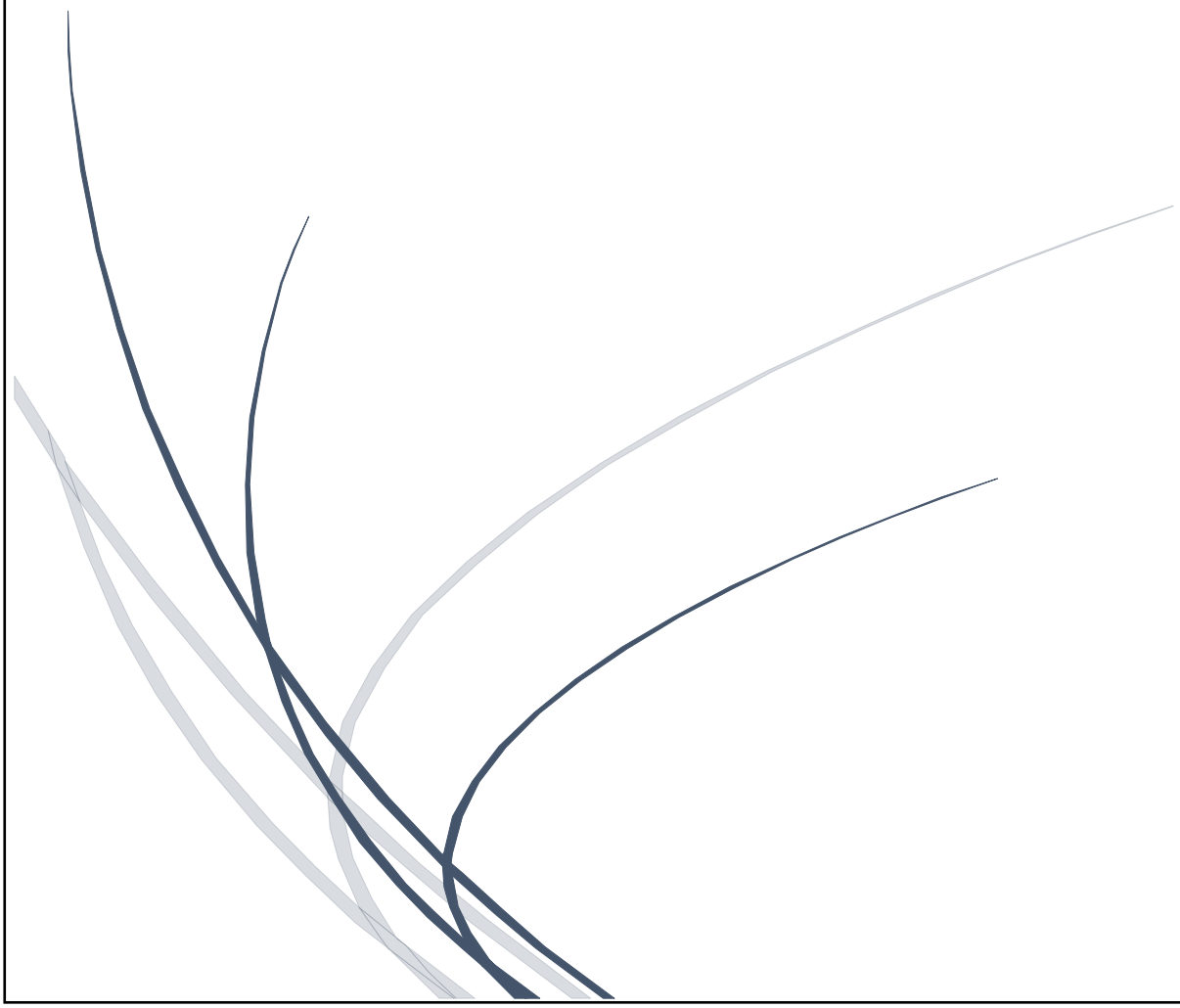

TABLE OF CONTENTS

FIGURE OF CONTENTS

House Price Analysis

House price analysis is an important aspect of the real estate industry and it involves evaluating and forecasting the worth of a property based on various features and characteristics. The use of machine learning algorithms plays a crucial role in this analysis as they offer a systematic way to examine and understand the connections between various variables and the house price. By utilizing historical data, these algorithms can be trained to make precise predictions about the price of a home based on its attributes.

There are various machine learning methods that are applied in house price analysis, including regression algorithms such as linear regression, decision trees, random forests and support vector regression. These algorithms can be utilized to construct predictive models that consider factors such as location, size, number of bedrooms and bathrooms and the age of the property. Moreover, these models can be refined by adding external factors such as economic indicators and crime rates, to the analysis.

The integration of machine learning in house price analysis provides a potent method for forecasting the value of a property, making it a valuable resource for real estate agents, buyers and sellers. With the ongoing advancements in artificial intelligence, it is anticipated that these techniques will become even more advanced, offering even more accurate and dependable predictions.

There are several reasons why a machine learning model is desirable for real estate price analysis.

- Accuracy
- Efficiency
- Reliability
- Better decision making
- Improved market understanding

By training on extensive data, machine learning models are capable of recognizing complex relationships between various factors and the price of a property. This leads to more accurate price predictions compared to conventional methods. Based on the efficiency, automating the analysis process through machine learning models enables quicker and more efficient analysis of large data sets, conserving time and resources. As well as the reliability, Machine learning algorithms are less prone to human error and can consistently produce results, even when analyzing vast amounts of data.When considering better decision making, The insights and predictions produced by machine learning models can assist real estate agents, buyers, and sellers in making better informed decisions and negotiating more effectively.Finally about the improved market understanding, Machine learning models can provide a deeper understanding of the real estate market by uncovering patterns and relationships between variables that may not have been previously identified.

Trough this project,the real estate price analysis relies on technical analysis, which involves using mathematical and statistical methods to uncover patterns and trends in real estate data. Machine learning algorithms play a critical role in this process, enabling the analysis of large datasets and making accurate predictions about the value of a property.

Some of the most widely used machine learning algorithms for real estate price analysis include linear regression - This is a simple regression algorithm that models the relationship between a dependent variable (house price) and one or more independent variables (features such as location, size, number of bedrooms, etc.) as well as lasso regression-This is a regularization technique that can be used to simplify a model by reducing the number of features. It encourages the model to have sparse coefficients by adding a penalty term to the loss function, zeroing out features that do not significantly impact the real estate price & Ridge Regression – This is a regularization technique that helps to control the impact of highly correlated features on the price of a property in real estate price analysis. It adds a penalty term to the loss function, which encourages the coefficients to be small in magnitude, reducing the impact of multicollinearity, where two or more features are highly correlated. These algorithms model the relationship between the price of a property and various factors such as location, size, number of bedrooms, and other attributes.

By training these algorithms on historical data, they are capable of providing reliable and accurate price predictions. Additionally, incorporating external factors like economic indicators and crime rates can further improve the accuracy of these models.

2. Literature review

The growth of technology has had a major impact on the field of Real Estate Price Analysis. The introduction of innovative algorithms and models, such as linear regression, lasso, and ridge, has facilitated a deeper understanding of the factors that impact real estate prices. The combination of various data sources, such as property listings, demographic information, and geographic data, has also helped to create a more comprehensive analysis of real estate prices. The integration of big data analytics and machine learning has made it possible to streamline the process of analyzing real estate prices, resulting in a more efficient and automated approach. Overall, the advancements in technology have greatly improved the accuracy and effectiveness of Real Estate Price Analysis, making it a valuable tool for professionals in the real estate industry.

The issue of real estate house prices is a major concern due to its direct impact on the economy and the lives of individuals. Accurate house prices are critical for both buyers and sellers in the real estate market as it determines their decisions regarding buying and selling properties. This is why it is important to have a comprehensive and reliable method for determining house prices.

However determining accurate house prices is a complex task as there are many factors that influence real estate prices such as location, property type, size, age, and many more. In the past real estate price analysis was mostly based on manual and subjective methods which were prone to human error and bias. With the advancement of technology, the use of various algorithms, models, data sources, big data analytics and machine learning techniques has allowed for a more accurate and efficient approach to real estate price analysis. By using these technological tools the real estate industry can make more informed decisions, minimize risks and improve overall performance.

"The Impact of Economic Indicators on Real Estate Prices: Evidence from the Turkish Housing Market" by Alper Aydın and Merve Şahin (2018) – This study investigates the relationship between real estate prices and various economic indicators in Turkey. The authors use regression analysis and time series methods to analyze the impact of macroeconomic indicators such as inflation, gross domestic product (GDP), and interest rates on housing prices.

"Real Estate Price Forecasting using Machine Learning Algorithms" by Jivitesh Dhaliwal, Nitin Kumar, and Rajat Agarwal (2017) – This research explores the use of machine learning algorithms for real estate price forecasting. The authors compare the performance of various algorithms such as decision trees, support vector regression and artificial neural networks. They conclude that machine learning algorithms can provide more accurate and efficient predictions compared to traditional statistical models.

"Spatial Autoregressive Model for Housing Price Analysis" by Dan Li and Guofeng Song (2020) – This study proposes a spatial autoregressive model for housing price analysis. The authors argue that the spatial dependence between housing prices in neighboring regions needs to be considered in the analysis. They demonstrate that the proposed model outperforms traditional regression models in terms of predictive accuracy.

3. Methodology

In this project, the goal is to perform data analysis on Bengaluru house prices using machine learning algorithms and the Python programming language. The analysis is performed using Google Colab, which is a free cloud-based platform that supports Jupyter notebooks. The data set used for the analysis was obtained from Kaggle, which is a website that provides a repository of datasets for various applications.

Bengaluru House Price dataset is a collection of data that pertains to the prices of houses in the city of Bengaluru, India. The data may include information such as the location of the property, the size of the property, the number of bedrooms and bathrooms, and other relevant factors that may impact the price of a house. The machine learning algorithms used for this analysis as mentioned above linear regression, lasso regression, and ridge regression among others. The choice of algorithm will depend on the nature of the data and the objective of the analysis.

Once the data is pre-processed and the appropriate machine learning algorithm is selected, the model can be trained on the historical data. The results of the analysis can then be used to make predictions about the price of a property based on its characteristics. Additionally, This data can be analyzed and used to gain a better understanding of the real estate market in Bengaluru, helping real estate agent s, buyers, and sellers make informed decisions.

3.1 Libraires used for the Data Analysis here

House Price Analysis

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

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
[ ] import numpy as np
    import pandas as pd
    import matplotlib.pyplot as plt
    import seaborn as sns
    import statsmodels.api as sm
    import os
    sns.set()
    from math import *
    import warnings
    warnings.filterwarnings('ignore')
```

```
[ ] data = pd.read_csv('/content/drive/MyDrive/ML - 02/Bengaluru_House_Data.csv')
```

Figure 1.Used Libraires

The above libraries are imported for use in the analysis of the Bengaluru House Price dataset.

- Numpy library is used for mathematical functions and working with arrays.
- Pandas library is used for reading the dataset and for data analysis purposes.
- Matplotlib.pyplot is imported for creating figures for data visualization.
- Seaborn is also imported for high-level data visualization.
- Statistical models are used, and the statmodels.api library is imported for that purpose.
- The OS library is imported to use the Python language.

3.2 Headings of Data Set

area_type	availability	location	size	society	total_sqft	bath	balcony	price
Super built-up Area	19-Dec	Electronic City Phase II	2 BHK	Coomee	1056	2	1	39.07
Plot Area	Ready To Move	Chikka Tirupathi	4 Bedroom	Theanmp	2600	5	3	120
Built-up Area	Ready To Move	Uttarahalli	3 BHK		1440	2	3	62
Super built-up Area	Ready To Move	Lingadheeranahalli	3 BHK	Soiewre	1521	3	1	95
Super built-up Area	Ready To Move	Kothanur	2 BHK		1200	2	1	51
Super built-up Area	Ready To Move	Whitefield	2 BHK	DuenaTa	1170	2	1	38
Super built-up Area	18-May	Old Airport Road	4 BHK	Jaades	2732	4		204

Figure 2.Heading of Data set

The Bengaluru House Price dataset includes the following columns or headings:

- area_type: This column provides information about the type of area in which the property is located. For example, it could be a residential area, a commercial area, etc.
- availability: This column indicates whether the property is available for purchase or has already been sold.
- location: This column provides the location of the property within the city of Bengaluru.

- size: This column provides information about the size of the property, typically in square feet.
- society: This column provides information about the society or community in which the property is located.
- total_sqft: This column provides the total area of the property in square feet.
- bath: This column provides information about the number of bathrooms in the property.
- balcony: This column provides information about the number of balconies in the property.
- price: This column provides information about the price of the property, typically in rupees.

These headings are important for analyzing the data and determining the factors that impact the price of a house in Bengaluru. They help provide a comprehensive picture of the properties and their respective prices, which can be useful for real estate professionals, buyers, and sellers.

4. Implementation

4.1 Loading the Raw Data

Loading the Raw Data

```
data.head()
```

	area_type	availability	location	size	society	total_sqft	bath	balcony	price
0	Super built-up Area	19-Dec	Electronic City Phase II	2 BHK	Coomee	1056	2.0	1.0	39.07
1	Plot Area	Ready To Move	Chikka Tirupathi	4 Bedroom	Theanmp	2600	5.0	3.0	120.00
2	Built-up Area	Ready To Move	Uttarahalli	3 BHK	NaN	1440	2.0	3.0	62.00
3	Super built-up Area	Ready To Move	Lingadheeranahalli	3 BHK	Soiewre	1521	3.0	1.0	95.00
4	Super built-up Area	Ready To Move	Kothanur	2 BHK	NaN	1200	2.0	1.0	51.00

```
[ ] data.tail()
```

	area_type	availability	location	size	society	total_sqft	bath	balcony	price
13315	Built-up Area	Ready To Move	Whitefield	5 Bedroom	ArsiaEx	3453	4.0	0.0	231.0
13316	Super built-up Area	Ready To Move	Richards Town	4 BHK	NaN	3600	5.0	NaN	400.0
13317	Built-up Area	Ready To Move	Raja Rajeshwari Nagar	2 BHK	Mahla T	1141	2.0	1.0	60.0
13318	Super built-up Area	18-Jun	Padmanabhanagar	4 BHK	SollyCl	4689	4.0	1.0	488.0
13319	Super built-up Area	Ready To Move	Doddathoguru	1 BHK	NaN	550	1.0	1.0	17.0

```
[ ] data.shape
(13320, 9)
```

Figure3.Loading the Raw Data

Here loading the raw data. Getting a quick overview as above, The first five rows & last five rows represent a small sample of the data, so this function can give you a general sense of the data's content and structure. As well as understanding the size of the data set and making sure that the data has been loaded correctly. It can also be helpful for ensuring that the data set has enough data to support the analysis. In this data set, There are 09 columns & 13320 rows as the data.shape function returns.

```
data.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 13320 entries, 0 to 13319
Data columns (total 9 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   area_type     13320 non-null  object
 1   availability  13320 non-null  object
 2   location      13319 non-null  object
 3   size          13304 non-null  object
 4   society       7818 non-null   object
 5   total_sqft    13320 non-null  object
 6   bath          13247 non-null  float64
 7   balcony       12711 non-null  float64
 8   price         13320 non-null  float64
dtypes: float64(3), object(6)
memory usage: 936.7+ KB
```

Figure 4. Getting a quick overview

Here getting a quick overview of the data and to check for any missing or inconsistent values. It can also be used to verify that the data types of each column are correctly set, which is important for the analysis and modeling steps. By running data.info() on the Bengaluru House Price dataset, you can quickly identify any potential issues with the data and take steps to address them before proceeding with your analysis.

4.2 Data preprocessing

Preprocessing

```
data.describe(include = 'all')
```

	area_type	availability	location	size	society	total_sqft	bath	balcony	price
count	13320	13320	13319	13304	7818	13320	13247.000000	12711.000000	13320.000000
unique	4	81	1305	31	2688	2117	NaN	NaN	NaN
top	Super built-up Area	Ready To Move	Whitefield	2 BHK	GrrvaGr	1200	NaN	NaN	NaN
freq	8790	10581	540	5199	80	843	NaN	NaN	NaN
mean	NaN	NaN	NaN	NaN	NaN	NaN	2.692610	1.584376	112.565627
std	NaN	NaN	NaN	NaN	NaN	NaN	1.341458	0.817263	148.971674
min	NaN	NaN	NaN	NaN	NaN	NaN	1.000000	0.000000	8.000000
25%	NaN	NaN	NaN	NaN	NaN	NaN	2.000000	1.000000	50.000000
50%	NaN	NaN	NaN	NaN	NaN	NaN	2.000000	2.000000	72.000000
75%	NaN	NaN	NaN	NaN	NaN	NaN	3.000000	2.000000	120.000000
max	NaN	NaN	NaN	NaN	NaN	NaN	40.000000	3.000000	3600.000000

Figure 5.Data describing

From here Data preprocessing starts. Data preprocessing refers to the techniques used to clean and transform raw data into a format that is suitable for analysis. In the context of the Bengaluru House Price dataset, data preprocessing may involve the following steps. Handling Missing Data, Removing Duplicates, Data Cleaning, Feature Engineering, Encoding Categorical Variables, Normalizing Numeric Variables & Splitting the Dataset.

As above figure , the function is utilized to produce descriptive statistics for the dataset. By setting the "include" parameter to "all", the function computes statistics for all columns in the dataset, regardless of their data type. Here showing a summary of the numerical data distribution in the dataset, including count, mean, standard deviation, minimum value, first quartile, median, third quartile, and maximum value.

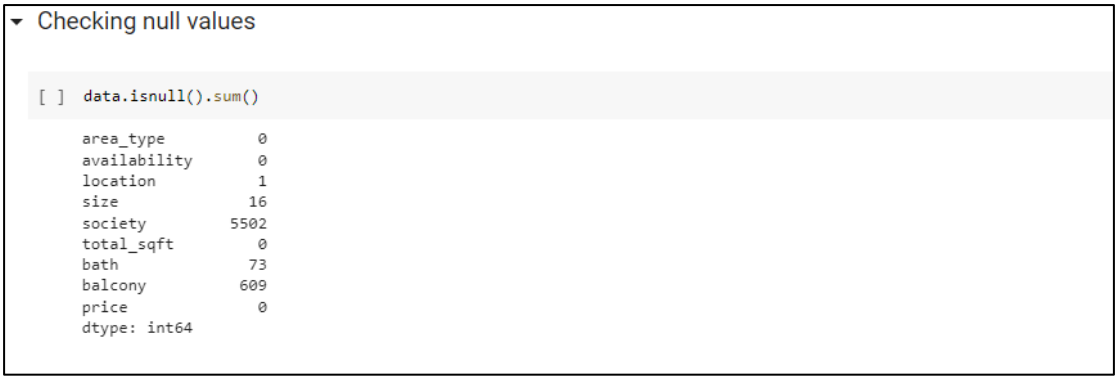

Figure 6.Checking null values

The output of above function provides information on the number of missing values in each column which can be useful in determining if and how missing values should be addressed before modeling.as above figure,getting a summary of the number of missing values in each column of the Bengaluru House Price dataset. This information can then be used to decide how to handle the missing values & it leads to prevent for producing unreliable or incorrect results.

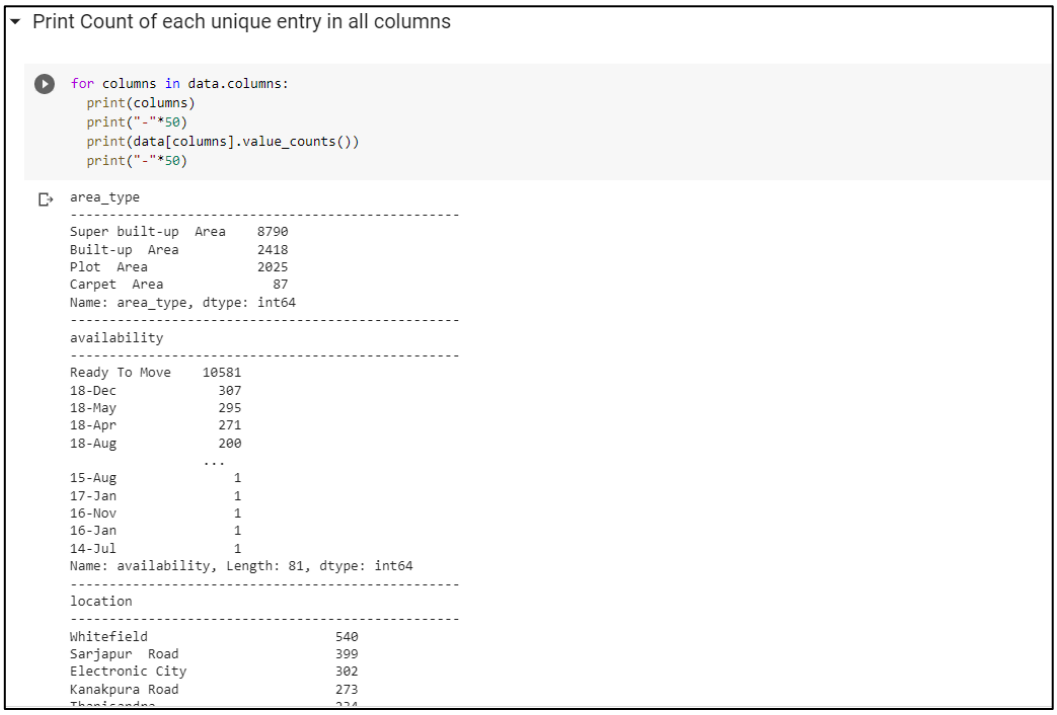

Figure 7.Printing count of each unique entry in all columns

Above code is useful for understanding the distribution of data within each column and identifying any potential issues or irregularities that may need to be addressed before modeling. There printing a line of 50 dashes to visually separate the output for each column & the value counts of each unique value in the column using the value_counts() method.after that printing another line of 50 dashes to visually separate the output for each column.

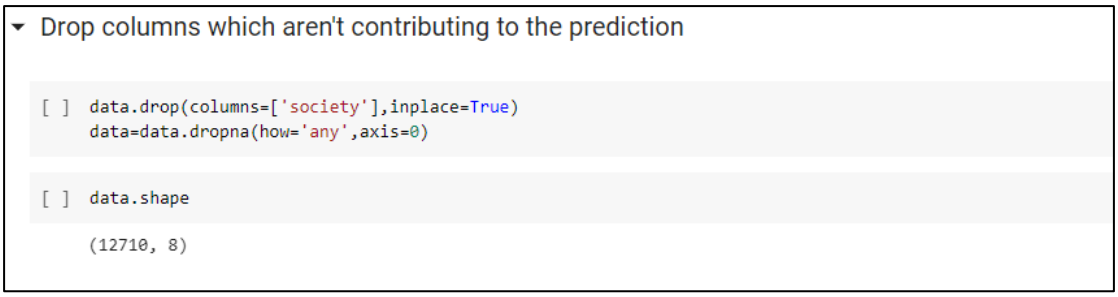

Figure 8.Dropping a column

The above lines of code are a part of the data preprocessing phase for the Bengaluru House Price dataset. The first line is used to remove the "society" column from the dataset, using the "drop" method. The "columns" parameter is set to a list containing the name of the column to be dropped, and the "inplace" parameter is set to "True," indicating that the change should be made directly to the original dataset instead of creating a new one. The second line of code uses the "dropna" method to remove any rows that contain missing values. The "how" parameter is set to "any," meaning that any row with at least one missing value will be removed, and the "axis" parameter is set to "0," which indicates that the operation should be performed on rows.

4.3 Dealing with outliers

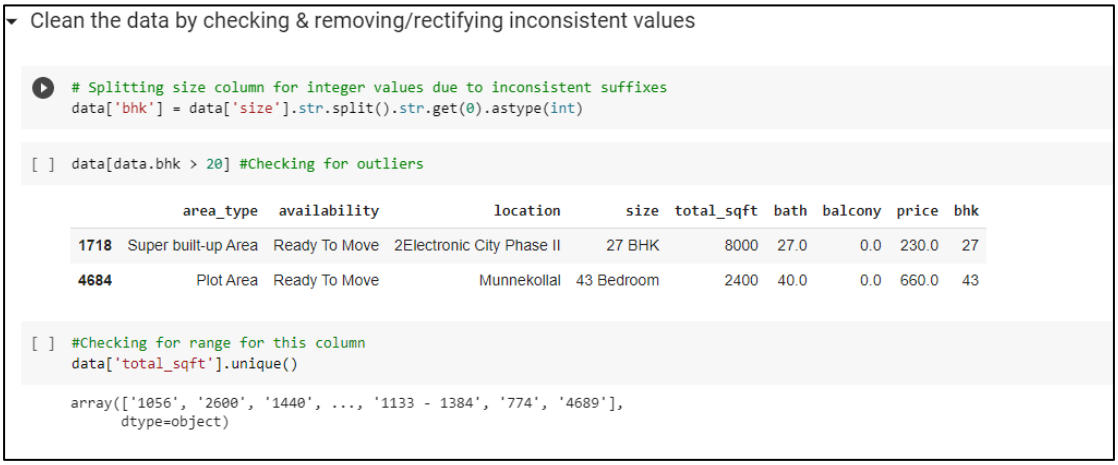

Figure 9.Splitting size columns

First line of code performs preprocessing on the Bengaluru House Price dataset by creating a new column named 'bhk' by representing bedrooms. The number of bedrooms is obtained from the 'size' column of the dataset. The 'str.split' method splits the values in the 'size' column based on spaces, and the 'str.get(0)' method retrieves the first value which is the number of bedrooms. The '.astype(int)' method converts the extracted value to an integer data type. Having this new column can be important in analyzing the Bengaluru House Price dataset as the number of bedrooms in a house often has a significant impact on its price.

When considering the second line, that filter is used to identify any outliers in the data that may need to be addressed before modeling. There found 02 number of outliers as above figure.

Third line can be used to examine any discrepancies or inaccuracies in the data, such as incorrect or inconsistent units of measurement or to find potential outliers & it is showing the the range of the column.

Remove outliers

```
[ ]  # Creating a function to deal with the ranges given in some cells in "total_sqft" column

[ ]  def Covt_Range(x):
       temp=x.split('-')
       if len(temp) == 2:
         return (float(temp[0])+ float(temp[1]))/2
       try:
           return float(x)
       except:
           return None

[ ]  #Applying "Covt_Range" to "total_sqft" column

[ ]  data['total_sqft']=data['total_sqft'].apply(Covt_Range)

[ ]  data['total_sqft'].unique()

     array([1056. , 2600. , 1440. , ..., 1258.5,  774. , 4689. ])

[ ]  data['area_type'].unique()

     array(['Super built-up  Area', 'Plot  Area', 'Built-up  Area',
            'Carpet  Area'], dtype=object)

[ ]  data.drop(columns='area_type',inplace = True)

[ ]  data.drop(columns='availability',inplace = True)
```

Figure 10.Removing outliers

As above figure, First function can be used to convert the values in the 'total_sqft' column to float values. It cleans and preprocesses the 'total_sqft' column of the Bengaluru House Price dataset by converting the area range into a single value which can be used for more accurate analysis. After that second function is to standardize the values in the 'total_sqft' column by converting the values in the format to their average and converting other non-numeric values to None. This will help in cleaning and preparing the data for further analysis.

Next line of code is used to verify the results of the processing performed on the 'total_sqft' column by the 'Covt_Range' function. By calling the 'unique()' method on the 'total_sqft' column, it retrieves an array of unique values in the column. This helps to determine if the function has successfully converted all the values in the 'total_sqft' column to a float data type & checking the different types of areas in the dataset and ensure that there are no inconsistencies or inaccuracies in the data from other line.After that dropping 'area_type' & ‘availability’ columns from the Bengaluru House Price dataset.

4.4 Checking correlation among different columns

```
[ ]  data.head()
```

	location	size	total_sqft	bath	balcony	price	bhk
0	Electronic City Phase II	2 BHK	1056.0	2.0	1.0	39.07	2
1	Chikka Tirupathi	4 Bedroom	2600.0	5.0	3.0	120.00	4
2	Uttarahalli	3 BHK	1440.0	2.0	3.0	62.00	3
3	Lingadheeranahalli	3 BHK	1521.0	3.0	1.0	95.00	3
4	Kothanur	2 BHK	1200.0	2.0	1.0	51.00	2

```
[ ]  correlation=data.corr()
     sns.heatmap(correlation,xticklabels=correlation.columns,yticklabels=correlation.columns,annot=True)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7ff0f6ee1eb0>
```

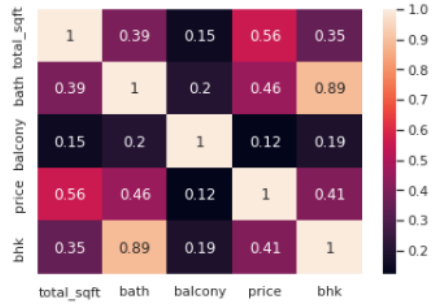

Figure 11.Checking correlation

The above code is used to calculate the correlation between the variables in the Bengaluru House Price dataset and to visualize the results. The 'corr' method is used to calculate the correlation and the result is stored in the 'correlation' variable. The 'heatmap' function from the seaborn library is then used to visualize the correlation matrix with the 'xticklabels' and 'yticklabels' parameters set to the column labels of the correlation matrix. The 'annot' parameter is set to True, which means that the values of the correlation matrix will be displayed on the heatmap. This visual representation of the correlation between the variables helps in identifying any strong relationships between them which can be useful in further analysis and modeling.

The above correlation heat map is a visual representation of the correlation between multiple variables. In the heat map, each square represents the correlation between two variables. The color of the square represents the strength of the correlation, with darker colors indicating a stronger correlation and lighter colors indicating a weaker correlation. In this case, all the variables have a positive correlation,meaning that as one variable increases the other variables also tend to increase. The strength of the correlation can vary with some variables having a low correlation, others having a moderate correlation and others having a high correlation. This information can be useful in understanding the relationship between variables and in predicting the behavior of one variable based on the behavior of another

Remove one of the columns form the ones highly correlated

+ Code

```
[ ] data['bath'].unique()
```

```
array([ 2.,  5.,  3.,  4.,  1.,  8.,  7.,  6.,  9., 27., 11., 12., 10.,
       40., 15., 13.])
```

```
[ ] data['balcony'].unique()
```

```
array([1., 3., 2., 0.])
```

```
[ ] data['bhk'].unique()
```

```
array([ 2,  4,  3,  1,  6,  8,  7,  5, 11,  9, 27, 43, 14, 12, 10, 13])
```

```
[ ] #"bath" and "bhk" columns are higly correlated, we need to drop one of them
    data.drop(columns=['bath'], inplace=True)
```

```
[ ] data.head()
```

	location	size	total_sqft	balcony	price	bhk
0	Electronic City Phase II	2 BHK	1056.0	1.0	39.07	2
1	Chikka Tirupathi	4 Bedroom	2600.0	3.0	120.00	4
2	Uttarahalli	3 BHK	1440.0	3.0	62.00	3
3	Lingadheeranahalli	3 BHK	1521.0	1.0	95.00	3
4	Kothanur	2 BHK	1200.0	1.0	51.00	2

Figure 12. Removing the columns which are highly correlated

It is observed that the "bath" and "bhk" columns are highly correlated. The correlation coefficient between these two variables is close to 1, which indicates a strong positive linear relationship. As a result, it is decided to drop one column as they are redundant and do not add much value to the analysis. By removing 'bath' column it can help reduce the noise in the data and avoid overfitting.

```
[ ] #Finding Price per Sq.ft
    data['price_per_sqft']=data['price']*100000/data['total_sqft']
```

```
[ ] data['price_per_sqft']=data['price_per_sqft'].round(2)
    data['price_per_sqft']
```

```
0         3699.81
1         4615.38
2         4305.56
3         6245.89
4         4250.00
           ...
13314     6530.61
13315     6689.83
13317     5258.55
13318    10407.34
13319     3090.91
Name: price_per_sqft, Length: 12710, dtype: float64
```

Figure 13.Finding House Price per Sq.ft

Above first line of code creates a new column in the Bengaluru House Price dataset called 'price_per_sqft' which provides a uniform measure of the property price that takes into consideration the variations in property size. The calculation is achieved by dividing the 'price' column by the 'total_sqft' column and then multiplying that result by 100,000. This column will be useful in analyzing and comparing property prices during further analysis.

And the second line of code rounds the values in the 'price_per_sqft' column to 2 decimal places. It can be useful for better readability and to avoid presenting excessively precise or unnecessarily large numbers. The line "data['price_per_sqft']" then retrieves the rounded 'price_per_sqft' column.

Print the desciptive statistics of the dataset

```
data.describe()
```

	total_sqft	balcony	price	bhk	price_per_sqft
count	12668.000000	12710.000000	12710.000000	12710.000000	1.266800e+04
mean	1511.835167	1.584343	106.060778	2.737136	6.876277e+03
std	1162.097276	0.817287	131.766089	1.205097	2.263354e+04
min	5.000000	0.000000	8.000000	1.000000	2.678300e+02
25%	1100.000000	1.000000	49.030000	2.000000	4.242720e+03
50%	1260.000000	2.000000	70.000000	3.000000	5.376340e+03
75%	1640.000000	2.000000	115.000000	3.000000	7.142860e+03
max	52272.000000	3.000000	2912.000000	43.000000	2.300000e+06

Figure 14.Printing the descriptive statistics

Retrieving the descriptive statistics of the dataset once again after applying the previously discussed modifications. This will give insight into the count, mean, standard deviation, minimum, 25th percentile, median, 75th percentile and maximum values for the available columns at this stage, showcasing any significant changes.

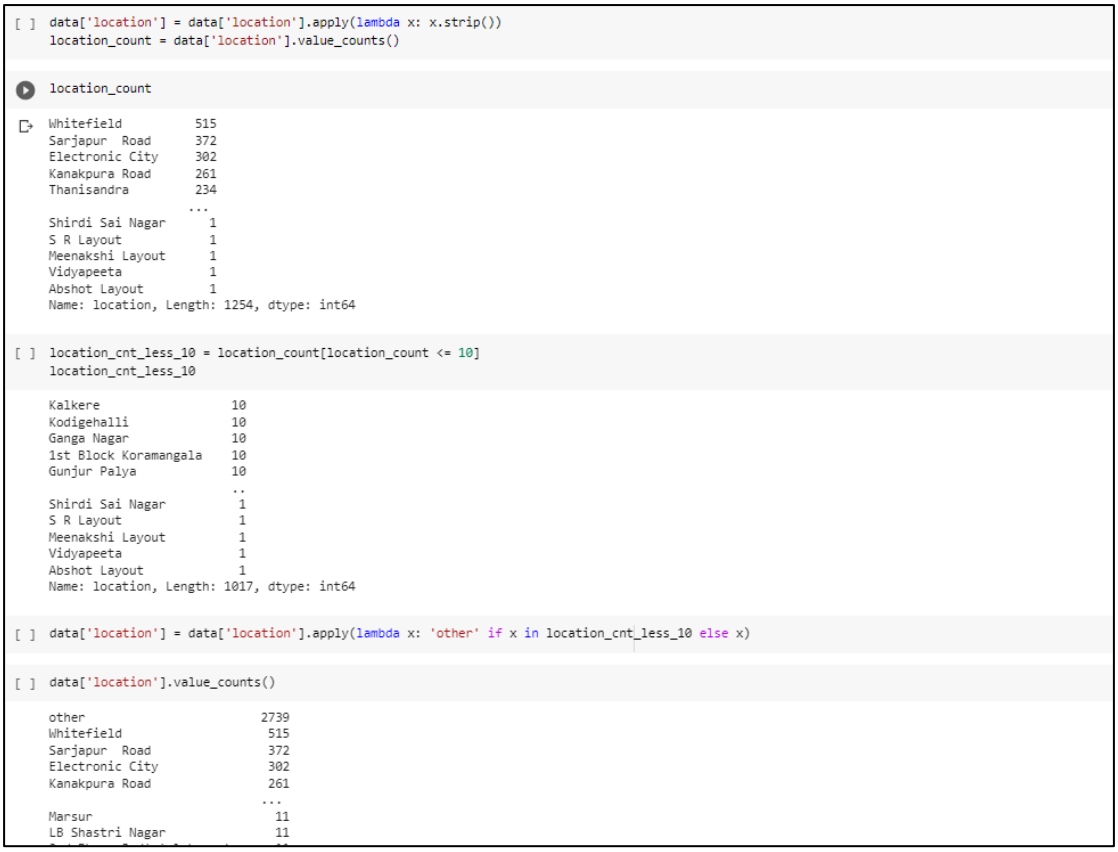

Figure 15.Creating a new column as location

First code in above figure creates a new column in the Bengaluru House Price dataset called 'location' and removes any extra spaces in the values of this column. Then it calculates the number of occurrences of each location using the 'value_counts' method and saves the result in a new variable called 'location_count'.

Second line of code is checking the number of times each location appears in the dataset. Locations with a count of 10 or less are stored in the "location_cnt_less_10" variable.

After that next code replaces the location names that appear 10 or less times in the dataset with the word "other." This is done to simplify the data and reduce the number of unique location names in the dataset.

Finally in this stage, getting the count of each unique location in the 'location' column of the data. It shows how many times each location appears in the dataset after the locations with count less than or equal to 10 were grouped together and labeled as "other".

```
# Removing outliers from "total_sqft" column
(data['total_sqft']/data['bhk']).describe() #*******************************************
```

```
count    12668.000000
mean       570.060291
std        380.298999
min          0.714286
25%        473.333333
50%        550.000000
75%        622.500000
max      26136.000000
dtype: float64
```

```
[ ] data=data[((data['total_sqft']/data['bhk']) >= 300 )] #estimation is 300
    data.describe()
```

	total_sqft	balcony	price	bhk	price_per_sqft
count	12013.000000	12013.000000	12013.000000	12013.000000	12013.000000
mean	1542.315982	1.587613	105.003648	2.607259	6206.082361
std	1181.094228	0.808867	134.205666	0.922985	3985.518849
min	300.000000	0.000000	9.000000	1.000000	267.830000
25%	1107.000000	1.000000	48.450000	2.000000	4199.360000
50%	1285.000000	2.000000	68.000000	2.000000	5252.530000
75%	1660.000000	2.000000	110.000000	3.000000	6823.530000
max	52272.000000	3.000000	2912.000000	13.000000	176470.590000

```
[ ] data.shape
```

```
(12013, 7)
```

Figure 16.Removing outliers from total_sqft

Above first line of code calculates the average size of each room in square feet in the properties in the Bengaluru House Price dataset by dividing the total square footage of each property by the number of rooms ‘bhk’ in each property. After that getting the summary statistics for this calculated average room size.

After that next code is used to remove any rows from the dataset that have an estimated square footage per bedroom of less than 300. This is done by dividing the 'total_sqft' column by the 'bhk' column and checking if the result is less than 300. Any rows that meet this criteria are dropped from the dataset. After removing ,descriptive statistics are calculated again to see the changes in the dataset.As well as checking the number of rows & columns available at this stage using ‘data.shape’.

```
[ ] # Removing the outlier from "price_per_sqft" column
    q = data['price_per_sqft'].quantile(0.99)
    data_1 = data[data['price_per_sqft'] < q]
    data_1.describe()
```

	total_sqft	balcony	price	bhk	price_per_sqft
count	11892.000000	11892.000000	11892.000000	11892.000000	11892.000000
mean	1529.214625	1.586865	98.261674	2.593508	5977.170620
std	1162.286734	0.807795	102.325976	0.907505	2868.665841
min	300.000000	0.000000	9.000000	1.000000	267.830000
25%	1105.000000	1.000000	48.000000	2.000000	4189.640000
50%	1280.000000	2.000000	68.000000	2.000000	5233.565000
75%	1650.000000	2.000000	110.000000	3.000000	6760.560000
max	52272.000000	3.000000	2200.000000	13.000000	20645.160000

```
[ ] data.bhk.describe()
```

```
count    12013.000000
mean         2.607259
std          0.922985
min          1.000000
25%          2.000000
50%          2.000000
75%          3.000000
max         13.000000
Name: bhk, dtype: float64
```

Figure 17.Removing outliers from price_per_sqft

Above code is calculating the 99th percentile value of the 'price_per_sqft' column in the Bengaluru House Price dataset. Then it is creating a new dataset called 'data_1' by selecting only the rows from the original dataset where the 'price_per_sqft' is less than the 99th percentile value. Finally the describe method is being used to get the descriptive statistics for the new dataset 'data_1'. This is done to remove the extreme outliers in the 'price_per_sqft' column and get a more accurate picture of the distribution of property prices in Bengaluru.

Remove remaining outliers identified by descriptive statistics

```
[ ] #Removing outliers or invalid data points from 'bhk' column

    def bhk_outliers(df):
      exclude_indices = np.array([]) # Creating an empty array which will carry all the values we need to remove
      for location,location_df in df.groupby('location'):
        bhk_stats = {}
        for bhk, bhk_df in location_df.groupby('bhk'):
          bhk_stats[bhk]={
              'mean' : np.mean(bhk_df.price_per_sqft),
              'std' : np.std(bhk_df.price_per_sqft),
              'count': bhk_df.shape[0]
          }
        for bhk, bhk_df in location_df.groupby('bhk'):
          stats = bhk_stats.get(bhk-1)
          if stats and stats['count'] >5: #checking for threshold no. of data points to get accurate mean
              exclude_indices= np.append(exclude_indices,bhk_df[bhk_df.price_per_sqft < (stats['mean'])].index.values)
      return df.drop(exclude_indices,axis='index')
```

```
[ ] data = bhk_outliers(data)
    data.shape

    (8320, 7)
```

```
[ ] data.head()
```

	location	size	total_sqft	balcony	price	bhk	price_per_sqft
0	Electronic City Phase II	2 BHK	1056.0	1.0	39.07	2	3699.81
1	Chikka Tirupathi	4 Bedroom	2600.0	3.0	120.00	4	4615.38
2	Uttarahalli	3 BHK	1440.0	3.0	62.00	3	4305.56
3	Lingadheeranahalli	3 BHK	1521.0	1.0	95.00	3	6245.89
4	Kothanur	2 BHK	1200.0	1.0	51.00	2	4250.00

Figure 18.Removing outliers or invalid data

Above function removes outliers from ‘bhk’ column. The function starts by creating an empty array called exclude_indices to store the indices of the data points that need to be removed.

The function then groups the data by location and by the number of bedrooms 'bhk'. For each group of data the function calculates the mean and standard deviation of the price per square foot. It only keeps the information for groups with more than 5 data points as this is considered a threshold number to get an accurate mean.Finally the function removes any data points that have a price per square foot lower than the mean of a similar group with one fewer bedrooms. The indices of these data points are added to the exclude_indices array and the function returns the original dataframe with these data points removed.

Next line of code applies a function called "bhk_outliers" to the dataset which helps identify and remove any outliers in the data. The function groups the data by location and number of bedrooms ‘bhk’ and calculates the mean and standard deviation of the price per square foot for each group. If there are less than 5 data points in a group the function does not consider that group. If there are more than 5 data points, the function checks if any data points in that group have a price per square foot that is less than the mean of the previous group with one less bedroom. If any such data points are found the function removes those indices from the dataset. Finally the modified dataset is returned and its shape is printed to show the number of rows and columns in the new dataset as 8320 rows & 07 columns at this stage.

4.5 Splitting the target & independent variables

Split the target and independent variables from the dataset

```
[ ] # Removing columns which are not necessary for the model
    data.drop(columns=['size' , 'price_per_sqft'] , inplace=True)
```

```
[ ] data.head()
```

	location	total_sqft	balcony	price	bhk
0	Electronic City Phase II	1056.0	1.0	39.07	2
1	Chikka Tirupathi	2600.0	3.0	120.00	4
2	Uttarahalli	1440.0	3.0	62.00	3
3	Lingadheeranahalli	1521.0	1.0	95.00	3
4	Kothanur	1200.0	1.0	51.00	2

```
[ ] data.info()

<class 'pandas.core.frame.DataFrame'>
Int64Index: 8320 entries, 0 to 13319
Data columns (total 5 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   location    8320 non-null   object
 1   total_sqft  8320 non-null   float64
 2   balcony     8320 non-null   float64
 3   price       8320 non-null   float64
 4   bhk         8320 non-null   int64
dtypes: float64(3), int64(1), object(1)
memory usage: 390.0+ KB
```

```
[ ] data.to_csv("Cleaned_data.csv")
```

```
[ ] x = data.drop(columns = ['price']) # independent features
    y = data['price']  # feature which we are predicting through the model
```

Figure 19.Splitting the target & independent variables

First line of code removes the columns 'size' and 'price_per_sqft' from the dataset. The changes are made directly in the original data and not on a copy.after that checking the data set by getting quick view as above figure.As well as the data is saved to a new file called "Cleaned_data.csv". This file will contain the data that has been cleaned and processed with any unwanted columns or values removed.

After that last code in above figure separate the data into two parts.one for input and one for the target. The input data contains all the columns except for the "price" column which did store in a separate variable called "y". The input data stored in the "x" variable.

4.6 Performing the test & train split

Figure 20.Performing the test &train split

After preprocessing the data and creating the variables 'x' and 'y', the next step is to split the data into training and testing sets. This is a common step in building machine learning models to evaluate how well the model will perform on unseen data. The training data is used to train the model and the testing data is used to evaluate the model's performance.

Above figure shows the importing new libraries for performing the test & train split.when getting one by one as follow,

- Splitting data into training and test sets(train_test_split)
- Implementing linear regression algorithms (LinearRegression, Lasso, and Ridge)
- Preprocessing data (OneHotEncoder and StandardScaler)
- Combining multiple preprocessing steps into a single process (make_column_transformer)
- Simplifying the pipeline for implementing machine learning algorithms (make_pipeline)
- Evaluating the performance of the model using a metric called R-squared (r2_score)

After that next line splits the data into two parts as training and testing. 80% of the data is used for training and 20% for testing.

Next checking the shapes of the training and test data sets.X train data has 6656 rows & 4 columns. X test has 1664 rows & 4 columns.

5. Result

5.1 Creating Model & generating r2_score from the models

Model 1: Linear Regression

```
#Create the linear regression model and print the r2_score for it.

column_trans = make_column_transformer((OneHotEncoder(sparse=False),['location']), remainder='passthrough')

scaler = StandardScaler()

lr = LinearRegression(normalize=True)

pipe=make_pipeline(column_trans, scaler,lr)

pipe.fit(x_train, y_train)

Pipeline(steps=[('columntransformer',
                 ColumnTransformer(remainder='passthrough',
                                   transformers=[('onehotencoder',
                                                  OneHotEncoder(sparse=False),
                                                  ['location'])])),
                ('standardscaler', StandardScaler()),
                ('linearregression', LinearRegression(normalize=True))])

y_pred_lr = pipe.predict(x_test)

r2_score(y_test, y_pred_lr)

0.706275732826672
```

Figure 21.Model 1: Linear Regression

In this code block, creating the linear regression model & printing the r2_score.How it is working is a column transformer object is created using the make_column_transformer function. This object is used to apply one hot encoding to the 'location' column in the data. The remainder of the data is left unchanged.

After that a standard scaler object is created which is used to standardize the data so that it has a mean of 0 and a standard deviation of 1.

Next a linear regression model is created using the LinearRegression class with the normalize parameter set to True.Finally a pipeline object is created using the make_pipeline function. This pipeline object chains together the column transformer, scaler and linear regression model into one end-to-end process. The pipeline is then fit to the training data and used to make predictions on the test data. The R2 score is then calculated to evaluate the performance of the model.

The R2 value of 0.706275732826672 is a measure of how well the model fits the data. R2 ranges from 0 to 1, where a value of 1 means the model perfectly predicts the data, and a value of 0 means the model does not explain the variation in the data at all. A value of 0.706275732826672 means that the model explains about 70.63% of the variation in the data. The higher the R2 value, the better the model fits the data. However a high R2 value alone does not guarantee a good model as it may indicate overfitting or not generalizing well to unseen data.

Figure 22.Model 2:Lasso

The Lasso model is trained using a pipeline which consists of three steps as column transformation, scaling and Lasso model training. The pipeline is created using the make_pipeline function from scikit-learn. The Lasso model is fit to the training data using the fit method and predictions are made on the test data using the predict method. Finally the coefficient of determination (R^2) score which is a measure of how well the model fits the data is calculated using the r2_score function from scikit-learn and the result is returned.

The R2 value of 0.7064964807885676 for the Lasso model indicates a relatively good performance of the model. R2 is a metric that measures the goodness of fit of a regression model with values ranging from 0 to 1. A value closer to 1 indicates a better fit of the model to the data while a value closer to 0 indicates a weaker fit. In this case a value of 0.7064964807885676 suggests that the Lasso model is able to explain about 70.6% of the variation in the target variable (price) based on the independent variables. Although it is not a perfect fit.It is still a reasonably good fit for the data.

Model 3: Ridge

```
[ ] ridge =Ridge()

[ ] pipe = make_pipeline(column_trans, scaler, ridge)

    pipe.fit(x_train, y_train)

    Pipeline(steps=[('columntransformer',
                     ColumnTransformer(remainder='passthrough',
                                       transformers=[('onehotencoder',
                                                      OneHotEncoder(sparse=False),
                                                      ['location'])])),
                    ('standardscaler', StandardScaler()), ('ridge', Ridge())])

[ ] y_pred_ridge = pipe.predict(x_test)
    r2_score(y_test, y_pred_ridge)

    0.7062667131366189
```

Figure 23.Model 3:Ridge

As above figure the next step involves training the Ridge regression model. The Ridge model is similar to the Linear Regression and Lasso models with a slight difference in the way it deals with overfitting. Ridge regression adds a penalty term to the loss function to reduce the complexity of the model and prevent overfitting. A pipeline is created that consists of the column transformer, scaler and the Ridge model. The pipeline is then fitted to the training data and the resulting model is used to make predictions on the test data. The performance of the model is evaluated by calculating the R-squared value between the actual and predicted test target values.

The R2 value is a measure of how well the model fits the data. It ranges from 0 to 1, where a value of 1 means the model fits the data perfectly. The R2 value of 0.7062667131366189 indicates that the Ridge model explains approximately 70.6% of the variance in the data.This is a good R2 value but it is not perfect. There is still room for improvement in the model. It is important to keep in mind that the R2 value is just one measure of model performance and it is necessary to consider other metrics and techniques to determine the overall effectiveness of the model.

```
[ ] print("Linear Regression: ", r2_score(y_test, y_pred_lr))
    print("Lasso: ", r2_score(y_test, y_pred_lasso))
    print("Ridge: ", r2_score(y_test, y_pred_ridge))

    Linear Regression:  0.706275732826672
    Lasso:  0.7064964807885676
    Ridge:  0.7062667131366189
```

Figure 24.Printing r2_score of 03 Model

Printing r2_score of 03 regression models applied here as above figure.

6. Conclusion

Here various regression models have been applied to a given real estate dataset. Firstly preprocessed the data to clean and prepare it for modeling. This involved removing outliers, handling missing values and converting categorical data into numerical data. After that split the data into training and testing sets with 80% of the data used for training and 20% used for testing.

Then three different regression models are applied, Linear Regression, Lasso and Ridge, to the training data and made predictions on the test data. The performance of each model was evaluated using the R-squared metric which measures the proportion of variation in the target variable that is explained by the predictors.

The results showed that all three models had similar R-squared values, with Linear Regression having a value of 0.7062, Lasso having a value of 0.7064, and Ridge having a value of 0.7062. This suggests that all three models perform similarly well in explaining the variation in the target variable.

In conclusion the choice of which model to use would depend on various factors such as the complexity of the data, the presence of multicollinearity among the predictors and the requirement for a sparse solution.

7. References

[1] Alper Aydın and Merve Şahin (2018), "The Impact of Economic Indicators on Real Estate Prices: Evidence from the Turkish Housing Market"

[2] Jivitesh Dhaliwal, Nitin Kumar, and Rajat Agarwal (2017),” Real Estate Price Forecasting using Machine Learning Algorithms”

[3] Dan Li and Guofeng Song (2020),” Spatial Autoregressive Model for Housing Price Analysis”

[4] Nor Hamizah Zulkifley, Shuzlina Abdul Rahman, Nor Hasbiah Ubaidullah and Ismail Ibrahim(2020),” House Price Prediction using a Machine Learning Model: A Survey of Literature” 2020 MECS