
UNIVERSITY OF
LINCOLN

# CMP9781 Big Data Analytics and Modelling Assessment 1

**School of Engineering and Physical Sciences**

**25847208@students.lincoln.ac.uk**

**Tarteel Alkaraan (25847208)**

**University of Lincoln**

**Word count: 4000 (excluding table of content and references)**

## Table of Contents

# 1. Distributed Big Data Processing Ecosystem

## 1.1 Five V's

### 1.1.1 Volume

The volume of data that requires processing is increasing at a rapid pace, necessitating greater storage capacity, computational power, and advanced tools and techniques (Yu, 2025b). currently, there is a significant number of devices generating data, especially when compared to 10 or 20 years ago. Furthermore, the amount of data produced in the next 10 or 20 years is likely to increase substantially. The growth rate of data is another critical factor to consider (Al-Khafajiy, 2023a). This refers to the sheer volume of big data being generated, which ranges from several terabytes to potentially petabytes (segment, 2024). When the volume of data reaches a sufficiently large scale, it qualifies as big data (Robinson, 2023). The volume of big data refers to the size of dataset that require analysis and processing, often exceeding terabytes and petabytes (Framework, 2024).

### 1.1.2 Velocity

Big data is being generated at an unprecedented rate. Every day, approximately 900 million photos are uploaded to Facebook, 500 million tweets are posted on Twitter, and 3.5 billion searches are conducted on Google. Given this massive amount of data, rapid processing is essential to ensure that search results and other real-time applications respond instantaneously (Yu, 2025b). The exponential growth of data presents significant challenges in analysis. Timely data processing is crucial, necessitating real-time analytics. Historically, companies relied on overnight batch processing to handle large-scale data operations. However, with the increasing demand for immediate insights, organizations must now develop efficient methods for processing data streams(Al-Khafajiy, 2023a). So velocity, a key characteristic of big data, refers to the speed at which data is generated and transmitted. This aspect is particularly critical for businesses that rely on real-time data availability to make informed decisions. Ensuring that data is processed efficiently allows organizations to optimize decision-making processes and enhance operational effectiveness (Robinson, 2023).

### 1.1.3 Variety

Big data encompasses various formats, types, and structures, including text, numerical data, images, audio, video, sequences, time series, social media data, and multi-dimensional arrays (Yu, 2025b). Additionally, data can be categorized as either static or streaming, with a single application often generating or collecting multiple types of data simultaneously. To extract meaningful insights, these diverse data types must be integrated and analysed collectively (Yu, 2025b). Sensors deployed worldwide, for instance, each continuously generating data and these involve devices such as cameras, pressure sensors, and GPS trackers. Beyond raw data, there is often a need to incorporate related data from external sources to provide a more comprehensive analytical perspective (Al-Khafajiy, 2023a). The concept of variety in big data refers to the extensive range of structured and unstructured data generated by both humans and machines. Among the most produced data formats are structured text, tweets, images, and videos (Coforge, 2024).

### 1.1.4 Value

Among the key characteristics of big data, value is arguably the most critical (Yu, 2025b). While access to vast amounts of data is beneficial, its true significance lies in the ability to extract meaningful insights (Yu, 2025b). Without the capacity to derive value from data, its existence becomes redundant. Therefore, businesses must clearly understand the costs and benefits associated with big data to ensure its effective utilization (Yu, 2025b). Data has no inherent purpose unless it is analysed and applied to generate insights. With careful consideration, organizations can leverage data for various strategic advantages. At this point, it is also essential to reflect on the ethical implications of data usage (Al-Khafajiy, 2023a). The concept of value in big data refers to the actionable insights that can be extracted from large datasets (segment, 2024). It also encompasses the benefits that data can provide, particularly in terms of how organizations utilize collected information to drive decision-making and improve operations (Robinson, 2023). While businesses are accumulating massive volumes of data, the fundamental question remains whether they can effectively extract meaningful and valuable insights from it (Jha, 2024).

### 1.1.5 Veracity

Veracity refers to the reliability, accuracy, and overall trustworthiness of data(Yu, 2025b). Given the diverse forms of big data, ensuring quality and precision can be challenging. For example, social media posts on platforms such as Twitter often contain hashtags, abbreviations, typos, and colloquial expressions, making it difficult to verify the reliability and accuracy of the content. However, advancements in big data analytics now enable the processing and analysis of such unstructured and inconsistent data (Yu, 2025b). Veracity also encompasses the inconsistencies and uncertainties present in data. In many cases, data can be messy, and maintaining quality control is complex. Consider the potential consequences of unauthorized access to GPS networks or fully automated flight systems. Ensuring data validity and distinguishing between accurate and erroneous information are critical challenges in such scenarios (Al-Khafajiy, 2023a). In essence, veracity pertains to the quality, integrity, and credibility of collected data. Data may contain missing values, inaccuracies, or ambiguities, limiting its ability to provide meaningful and actionable insights (Robinson, 2023).

## 1.2 Fault Tolerance and Resilience

### 1.2.1 Fault Tolerance

Fault tolerance is a process that enables an operating system to respond effectively to hardware or software failures (Fortinet, 2025). It refers to a system's ability to continue functioning despite malfunctions or disruptions (Fortinet, 2025). More specifically, fault tolerance describes the capability of a system—whether a computer, network, or cloud infrastructure—to maintain uninterrupted operations even when one or more of its components fail (Imperva, 2024). The fundamental objective of fault tolerance is to ensure that systems and applications can sustain their services without interruption in the event of hardware or software failures. By implementing fault-tolerant mechanisms, organizations can prevent faults from escalating into complete system failures (Lahti, 2005). Additionally, fault tolerance is characterized by a system's ability to recover and continue functioning seamlessly despite errors or component failures. This resilience is crucial for maintaining system reliability, availability, and consistency, ensuring that critical processes remain operational (Geeksforgeeks, 2024).

### 1.2.2 Fault Resilience

Fault resilience refers to the ability of a software system to recover from failures while continuing to function correctly. Unlike fault tolerance, which focuses on preventing failures, fault resilience emphasizes minimizing their impact and frequency rather than eliminating them entirely (Hariom, 2025). Fault resilience pertains to systems designed to respond gracefully to previously unknown failure modes, exhibit emergent resilient behaviours, and adapt to evolving failure conditions. In some cases, these systems can even transform in response to changing failure landscapes (Friedrichsen, 2022). More broadly, fault resilience describes a system's capacity to endure adversities and restore itself to a normal operational state. It involves adapting to challenges and recovering from unexpected disruptions (Kafka, 2024). A fault-resilient system adjusts to errors while maintaining service availability, though it may experience some performance degradation as a result (Wikipedia, 2025). Ultimately, fault resilience is a crucial capability that enables both systems and organizations to withstand adversity and recover rapidly from disruptive events (Solomons, 2024).

## 1.3 Hadoop Distributed File Systems (HDFS)

The HDFS is specifically designed for storing vast datasets on commodity hardware. It operates using two primary types of nodes: NameNodes and DataNodes. In HDFS, large files are divided into multiple blocks, which are then distributed across a set of DataNodes for storage (Yu, 2025a). HDFS is a fault-tolerant distributed file system that partitions data into chunks and distributes them across multiple machines. It is designed to detect and recover from failures efficiently, operating under the assumption that individual nodes are inherently unreliable (Wingate, 2023b). As a file system optimized for managing large datasets, HDFS is widely recognized as the most prevalent storage solution for Apache Hadoop. It facilitates the scalability of a single Hadoop cluster, allowing it to efficiently handle extensive data workloads (Holdsworth, 2024). HDFS serves as a fundamental component of the Hadoop ecosystem, enabling distributed data processing. Within this system, individual Hadoop nodes process data stored in their local storage, ensuring optimized performance and reliability (Cloud, 2025b).

## 1.4 Resource Manager and Scheduler

### 1.4.1 Resource Manager

The Resource Manager is a critical component responsible for managing resource interactions across multiple cloud environments within a federation. It abstracts and integrates resources into a unified collection by utilizing Application Programming Interfaces (APIs) provided by cloud service providers or open standards for resource management (Assis, 2016). The Resource Manager allocates and prioritizes processing requests based on specific workload requirements, ensuring optimal resource utilization (Arcitura, 2025). Additionally, it maintains the data structures of the database, facilitating efficient data retrieval through the buffer and storage manager (Noergaard, 2010). The Resource Manager also arbitrates resource distribution across all applications within the system, ensuring balanced allocation and efficiency (Yu, 2025d). In a broader organizational context, a resource manager plays a strategic role in overseeing resource allocation, optimizing their use to enhance productivity and operational effectiveness (Indeed, 2024).

### 1.4.2 Resource Scheduler

The Resource Scheduler is the process of determining when project resources are required and allocating them based on factors such as capacity planning and resource availability. Its primary objective is to ensure an optimal balance in resource distribution, preventing both overallocation and underutilization at any stage of the project (Landau, 2024). A Resource Scheduler utilizes a set of APIs to generate work requests for specific tasks, including creating, updating, starting, stopping, enabling, disabling, and deleting schedules (Oracle, 2024). By effectively allocating tasks, managing time, and optimizing resource utilization, a Resource Scheduler ensures that projects run efficiently and meet their deadlines (Świtek, 2025). Resource scheduling refers to the structured allocation of resources to projects and clients, with a focus on meeting project deadlines while staying within budget constraints. Modern resource scheduling software automates and streamlines this process, enhancing efficiency and resource management (Skuse, 2025).

## 1.5 Apache Hadoop Yet Another Resource Negotiator (YARN)

YARN is a resource management system that provides unified resource management and scheduling for upper-layer applications. It significantly enhances cluster resource utilization, centralized resource management, and data sharing (Al-Khafajiy, 2023b). YARN was introduced in Hadoop 2 to improve resource allocation and scalability. It separates resource management from job scheduling and monitoring, allowing Hadoop to extend beyond MapReduce and support a diverse range of processing models (Wingate, 2023b). YARN serves as the resource management and job scheduling component of the open-source Hadoop distributed processing framework (Stedman, 2020). Designed specifically for cluster and resource management, YARN plays a crucial role in optimizing the efficiency of Apache Hadoop (Yu, 2025d).

## 1.6 Apache Spark, Hive, Flume and Kafka

### 1.6.1 Apache Spark

Apache Hadoop and Apache Spark are frameworks that enable distributed processing of large datasets across clusters of computers using simple programming models. Apache Hadoop is designed to scale from a single server to thousands of machines, each providing local computation and storage. Instead of relying on hardware-based high availability, the framework is built to detect and manage failures at the application layer, ensuring a highly available service despite potential failures within the cluster (Al-Khafajiy, 2023b). Apache Spark, on the other hand, is a unified analytics engine for large-scale data processing. It is an open-source data processing engine that enables real-time data storage and processing across multiple clusters of computers, utilizing simple programming constructs (Yu, 2025c).

### 1.6.2 Apache Hive

Apache Hive is an open-source data warehouse software designed to facilitate the reading, writing, and management of large datasets stored in distributed storage systems using SQL-based querying. Its architecture allows structured data representation to be projected onto pre-existing data in storage. Additionally, it provides a command-line interface and a JDBC driver to enable user interaction with Hive (Al-Khafajiy, 2023b). Apache Hive is specifically designed to extract, process, and manage large datasets stored in HDFS (Databricks, 2025a). It is a distributed, fault-tolerant data warehouse system that facilitates large-scale data analytics. Built on Apache Hadoop, Hive enables users to query and manage petabytes of data using SQL-based commands (Apache, 2023). Moreover, Apache Hive supports integration with various storage systems, including HDFS and Apache HBase, allowing for efficient querying and data management across multiple platforms (Yu, 2025c).

### 1.6.3 Apache Flume

Apache Flume is a distributed, reliable, and highly available service designed for efficiently collecting, aggregating, and transferring large volumes of data. Its architecture is simple and flexible, based on streaming data flows. The system is fault-tolerant and incorporates tunable reliability mechanisms, including failover and recovery features to ensure data integrity and continuous operation (Flume, 2023). Flume functions as a data ingestion mechanism**,** enabling the collection, aggregation, and transportation of large-scale streaming data from multiple sources to a centralized data store**.** It is a highly configurable and distributed tool**,** specifically designed to stream and transfer data from web servers to HDFS (TutorialsPoint, 2025). Furthermore, Apache Flume ensures scalability and robustness**,** facilitating the seamless ingestion of real-time data from diverse sources into centralized storage solutions**,** making it a crucial component in big data processing pipelines (Yu, 2025c).

### 1.6.4 Apache Kafka

Apache Kafka is an open-source, distributed event streaming platform utilized by numerous enterprises for high-performance data pipelines, real-time streaming analytics, data integration, and mission-critical applications. It is widely adopted, with over 80% of Fortune 100 companies leveraging Kafka for their data processing needs (Al-Khafajiy, 2023b). Kafka operates as a publisher-subscriber (pub/sub) messaging system**,** serving as a centralized event/message queue for real-time applications**.** In Kafka**,** messages, also referred to as events, are categorized into topics**.** These topics are further partitioned**,** enabling distributed storage across multiple nodes within a cluster**,** thereby enhancing scalability and parallel processing (Al-Khafajiy, 2023d). Developed by the Apache Software Foundation**,** Kafka is a stream-processing software platform written in Scala and Java. The project aims to provide a unified, high-throughput, and low-latency solution for managing real-time data streams, making it a fundamental component of modern data infrastructure (Yu, 2025c).

## 1.7 Resilient Distributed Datasets (RDDs)

A RDD is a read-only collection of data in Apache Spark that can be partitioned across multiple machines within a cluster, facilitating parallel computation and fault tolerance through lineage reconstruction (Bonner, 2017). RDDs serve as the primary data structure in Spark, offering reliability and memory efficiency for parallel processing. By storing and processing data within RDDs, Spark enhances the performance of MapReduce operations (Dancuk, 2022). An RDD is an immutable, fault-tolerant collection of elements that can be distributed across multiple cluster nodes for parallel processing. It is the fundamental data structure within the Apache Spark open-source data processing engine (Holdsworth, 2025). An RDD represents a read-only dataset partitioned across a set of machines and can be reconstructed in the event of partition loss. Furthermore, RDDs significantly reduce the time required for reading and writing operations, enabling Spark to run ten to one hundred times faster than Hadoop (Yu, 2025c).

## 1.8 Data Lakes

A data lake is a centralized repository that stores large volumes of data in its native, raw format. Unlike a hierarchical data warehouse, which organizes data into files or folders, a data lake employs a flat architecture and object storage for data management (Databricks, 2025b). A data lake is designed to store, process, and secure vast amounts of structured, semi-structured, and unstructured data. It allows data to be stored in its original format while eliminating size constraints and accommodating a variety of data types (Cloud, 2025). Additionally, a data lake serves as both a storage solution and an organizational framework for managing highly diverse datasets from multiple sources (Oracle, 2022). A data lake functions as a centralized repository that ingests, stores, and facilitates the processing of large-scale datasets in their original form. It can handle all types of data and serves as a foundation for big data analytics and machine learning applications (Azure, 2025).

## 1.9 Spark ML

Spark ML is the machine learning module of Apache Spark, an open-source distributed computing system designed for processing large-scale datasets. PySpark ML is the corresponding Python library for Spark ML, enabling seamless integration with Python and simplifying machine learning workflows. It provides high-level APIs for fundamental machine learning tasks such as classification, regression, and clustering (Barjatiya, 2023). Spark ML serves as a scalable and user-friendly framework for developing machine learning models on large datasets. It streamlines the development of machine learning applications by offering a high-level API that integrates seamlessly with Spark's core functionalities (Aljobs, 2024). It facilitates the development and deployment of scalable machine learning algorithms (Yu, 2025c).

# 2. Real-Life Use Case for Big Data Analytics

## 2.1 Customer Churn Analysis

Churn analysis refers to the assessment of a company's customer attrition rate, with the objective of reducing it by evaluating customer interactions, product usage patterns, and overall engagement (paddle, 2025).

## 2.2 Five V's

The analysis typically involves developing models using extensive historical data comprising multidimensional records of customer engagement, usage patterns, behavioural trends, and customer segmentation. Generally, larger datasets contribute to more accurate models. However, since historical data can span extended timeframes and undergo various transformations before integration with company data, the overall data volume increases significantly. Consequently, effective data management is a critical requirement for conducting churn analysis (Polackoff, 2024). Customer churn analysis is typically conducted on either static or dynamically changing data. The speed at which data is generated, collected, processed, and analysed is crucial, as it directly influences a company's ability to take timely action to mitigate customer churn (Zdravevski, 2020).

In customer churn analysis, data variety refers to the different formats in which customer data is represented. Examples include structured data, such as customer demographics and transaction history; unstructured data, such as text, audio, and video derived from sources like social media comments, call centre recordings, and emails; and semi-structured data, such as web browsing behaviour and mobile app interactions. Incorporating diverse data formats can enhance model performance, improve service personalization, and uncover insights that structured data alone may not reveal. In customer churn analysis, the value of data refers to its usefulness and impact in making accurate predictions and reducing customer attrition (Zdravevski, 2020).

The objective is to identify and leverage high-value data to enhance decision-making and customer retention strategies while minimizing noise and the risk of overfitting. For instance, information such as the frequency of password changes may be irrelevant in predicting customer churn. However, data on customer complaints is likely to be more valuable in mitigating customer attrition. Customer churn analysis relies on diverse data sources, making it essential to ensure accuracy, consistency, and completeness. High data veracity enhances the reliability of predictions, informs effective customer retention strategies, and optimizes resource allocation. Conversely, incomplete customer call recordings exemplify low-veracity data, which can undermine analytical accuracy (Zdravevski, 2020).

**How can big data technologies be used in customer churn analysis?** Big data technologies enable companies to process, analyse, and predict customer churn from large and complex datasets by providing scalability, high processing speed, accuracy, and advanced analytical capabilities (Polackoff, 2024). Distributed data storage and processing technologies, such as Hadoop (HDFS) and Apache Spark, facilitate the management of large-scale datasets across distributed computing clusters, enabling efficient storage and parallel processing. These technologies allow for the analysis of extensive data sources, including call records, billing information, and service complaints from millions of customers. Real-time data streaming platforms, such as Apache Kafka, capture live customer data streams, such as login frequency, to identify potential indicators of future customer churn. Additionally, machine learning frameworks for predictive analytics, such as Apache Spark ML, can be leveraged to develop predictive models that detect churn patterns within vast datasets, enhancing the accuracy of customer retention strategies (Polackoff, 2024).

**Can distributed learning revolutionise way companies drive their decision-making processes?** Distributed learning has the potential to transform how companies enhance their decision-making processes by enabling faster, more scalable, and data-driven insights. This approach allows organizations to make real-time decisions through the efficient storage, management, and processing of data (Polackoff, 2024).

# 3. Linear Regression Analysis

Dataset that was selected to develop a pipeline for running a simple linear regression is house prices dataset from Kaggle (Kaggle, 2022). The dataset contains 545 rows and 13 columns. Before running the analysis, data preparation was run to ensure the data is cleaned and ready for analysis. Data cleansing is the process of detecting and correcting or removing inaccurate data from a dataset to ensure high-quality data remains. It is typically a critical component of data pre-processing before utilizing the data for its intended purpose. The specific techniques employed for data cleansing vary depending on the context; however, the primary objective remains consistent: identifying inaccurate data and either removing, flagging, or imputing it. Data profiling involves the systematic identification of data types, missing values, unique values, and relationships within a dataset (Wingate, 2023a).

Data quality can be assessed across five distinct dimensions. Validity refers to the extent to which data conforms to predefined rules or constraints relevant to a specific use case. Accuracy measures how closely data aligns with true or expected values. Completeness indicates the extent to which all required data is available. Consistency assesses whether data remains uniform within a single dataset or across multiple datasets. Uniformity refers to the standardization of data using consistent units of measurement. Several factors can negatively impact data quality. Irrelevant data refers to information that does not align with the intended use case. Duplicate data consists of redundant records, which may be exact copies or partial duplicates. Syntactically inconsistent data arises when data is improperly formatted compared to other records within the same column (Wingate, 2023a).

Missing data refers to instances where values are recorded as NaN, empty, or an equivalent placeholder. It is a common issue that typically needs to be addressed during the data cleansing stage rather than through alternative methods. There are three primary types of missing data: Missing Completely at Random (MCAR): The probability of data being missing is entirely independent of both observed and missing values. Missing at Random (MAR): The probability of data being missing is independent of the missing values themselves but may depend on observed data. Missing Not at Random (MNAR)**:** The probability of data being missing is dependent on the missing values themselves and cannot be explained solely by observed data. In some cases, it is possible to predict the likelihood of missing data based on non-missing values (Wingate, 2023a).

After importing necessary libraries to load dataset file which can be seen in snippet 1 below (code is from lecture 2). A data preparation was performed to check the data quality starting with checking the structure of the data as seen below in snippet 2. Then running descriptive statistics as seen below in snippet 3. After that unique values of every row for every column was examined that can be seen in snippet 4 below. Following that we counted the NaN values (none is found) in data which can be seen below in snippet 5. Now we change the format of all the categorical binary variables into numerical as seen in snippet 6. And final step in the data preparation was standardising the data as can be seen in snippet 7 below using MinMaxScaler method.

Analysis stage started by splitting data into 70% training data, and 30% testing data and setting price as outcome for model as can be seen below in snippet 8. As response variable was continuous variable we choose linear regression to model data by running the model on data as seen in snippet 9 below. After that we perform model evaluation checking the performance of the model on training and testing data which you can see below in snippet 10.

Mean Absolute Erro (MAE) takes absolute value of the differences between predicted and actual values, where 0 represent a perfect fit model. MSE measures the average of squared differences between predicted and actual values. Root Mean Squared Error (RMSE) is square root of the Mean Squared Error (MSE). The smaller MSE, better model's predictive accuracy. Values of measures were very small which indicate a good performance, with expected better performance on training data. For test set used from dataset generated table showing predicted vs actual price values as seen in snippet 11 below. Also, a scatter plot showing predicted vs actual price values in the testing data is performed as seen in snippet 12 and figure 1 below. As extra work for extra marks, a heatmap visualisation showing the correlation between the variables in the whole dataset was displayed as can be seen below in snippet 13 and figure 2.

```
#Import Libraries
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, mean_absolute_error
```

```
#Read Data
Data = pd.read_csv('Housing.csv')
Data
```

Snippet 1 Import libraries and Read Data

```
#Check Data Information
Data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 545 entries, 0 to 544
Data columns (total 13 columns):
 #   Column            Non-Null Count  Dtype 
---  ------            --------------  ----- 
 0   price             545 non-null    int64 
 1   area              545 non-null    int64 
 2   bedrooms          545 non-null    int64 
 3   bathrooms         545 non-null    int64 
 4   stories           545 non-null    int64 
 5   mainroad          545 non-null    object
 6   guestroom         545 non-null    object
 7   basement          545 non-null    object
 8   hotwaterheating   545 non-null    object
 9   airconditioning   545 non-null    object
 10  parking           545 non-null    int64 
 11  prefarea          545 non-null    object
 12  furnishingstatus  545 non-null    object
dtypes: int64(6), object(7)
memory usage: 55.5+ KB
```

Snippet 2 Checking Data

```
#Perform Descriptive Statistics
Data.describe()
```

| | price | area | bedrooms | bathrooms | stories | parking |
|---|---|---|---|---|---|---|
| **count** | 5.450000e+02 | 545.000000 | 545.000000 | 545.000000 | 545.000000 | 545.000000 |
| **mean** | 4.766729e+06 | 5150.541284 | 2.965138 | 1.286239 | 1.805505 | 0.693578 |
| **std** | 1.870440e+06 | 2170.141023 | 0.738064 | 0.502470 | 0.867492 | 0.861586 |
| **min** | 1.750000e+06 | 1650.000000 | 1.000000 | 1.000000 | 1.000000 | 0.000000 |
| **25%** | 3.430000e+06 | 3600.000000 | 2.000000 | 1.000000 | 1.000000 | 0.000000 |
| **50%** | 4.340000e+06 | 4600.000000 | 3.000000 | 1.000000 | 2.000000 | 0.000000 |
| **75%** | 5.740000e+06 | 6360.000000 | 3.000000 | 2.000000 | 2.000000 | 1.000000 |
| **max** | 1.330000e+07 | 16200.000000 | 6.000000 | 4.000000 | 4.000000 | 3.000000 |

Snippet 3 Descriptive Statistics

```
#Check Unique Values of every row for every column
N = Data.nunique(axis = 0)
print(f'Number of Unique Values in Every Column:\n{N}\n')
```

```
Number of Unique Values in Every Column:
price               219
area                284
bedrooms              6
bathrooms             4
stories               4
mainroad              2
guestroom             2
basement              2
hotwaterheating       2
airconditioning       2
parking               4
prefarea              2
furnishingstatus      3
dtype: int64
```

Snippet 4 Checking Unique Values

```
#Count NaN Values in Data
CountNaN = Data.isna().sum()
print(f'Number of Missing Values in Every Variable:\n{CountNaN}\n')
```

```
Number of Missing Values in Every Variable:
price             0
area              0
bedrooms          0
bathrooms         0
stories           0
mainroad          0
guestroom         0
basement          0
hotwaterheating   0
airconditioning   0
parking           0
prefarea          0
furnishingstatus  0
dtype: int64
```

Snippet 5 Counting NaN Values

```
#Change Categorical Variable mainroad into Numerical
Labels = {'yes': 1, 'no': 2}
Data['mainroad'] = Data['mainroad'].map(Labels)
```

```
#Change Categorical Variable guestroom into Numerical
Labels = {'yes': 1, 'no': 2}
Data['guestroom'] = Data['guestroom'].map(Labels)
```

```
#Change Categorical Variable basement into Numerical
Labels = {'yes': 1, 'no': 2}
Data['basement'] = Data['basement'].map(Labels)
```

```
#Change Categorical Variable hotwaterheating into Numerical
Labels = {'yes': 1, 'no': 2}
Data['hotwaterheating'] = Data['hotwaterheating'].map(Labels)
```

```
#Change Categorical Variable airconditioning into Numerical
Labels = {'yes': 1, 'no': 2}
Data['airconditioning'] = Data['airconditioning'].map(Labels)
```

```
#Change Categorical Variable prefarea into Numerical
Labels = {'yes': 1, 'no': 2}
Data['prefarea'] = Data['prefarea'].map(Labels)
```

```
#Change Categorical Variable furnishingstatus into Numerical
Labels = {'furnished': 1, 'semi-furnished': 2, 'unfurnished': 3}
Data['furnishingstatus'] = Data['furnishingstatus'].map(Labels)
Data.head()
```

Snippet 6 Changing Categorical Variables into Numerical

```
#Data Normalisation
Scaler = MinMaxScaler()
SC = ['price', 'area']
Data[SC] = Scaler.fit_transform(Data[SC])
Data
```

Snippet 7 Data Normalisation

```
#Preparing Model Training/Testing Data
X = Data.drop('price', axis = 1)
X = pd.get_dummies(X, drop_first = True)
Y = Data['price']

X_Train, X_Test, Y_Train, Y_Test = train_test_split(X, Y, test_size = 0.3)
```

Snippet 8 Preparing Model Training and Testing Data

```
#Model Training and Testing
Regressor = LinearRegression()
Regressor.fit(X_Train, Y_Train)

Y_Tr_Pred = Regressor.predict(X_Train)
Y_Te_Pred = Regressor.predict(X_Test)
```

Snippet 9 Model Training and Testing

```
#Train and Test Data Model Evaluation
Test_MSE = mean_squared_error(Y_Test, Y_Te_Pred)
Test_MAE = mean_absolute_error(Y_Test, Y_Te_Pred)
Test_RMSE = np.sqrt(Test_MSE)

Train_MSE = mean_squared_error(Y_Train, Y_Tr_Pred)
Train_MAE = mean_absolute_error(Y_Train, Y_Tr_Pred)
Train_RMSE = np.sqrt(Train_MSE)
```

```
#Print Test Set Model Evaluation
print('Test Set:')
print('\nMean Squared Error: ', Test_MSE)
print('Mean Absolute Error: ', Test_MAE)
print('Root Mean Square Error: ', Test_RMSE)
```

```
Test Set:

Mean Squared Error:  0.010536340467680354
Mean Absolute Error:  0.07243960505273087
Root Mean Square Error:  0.10264667782096192
```

```
#Print Train Set Model Evaluation
print('Train Set:')
print('\nMean Squared Error: ', Train_MSE)
print('Mean Absolute Error: ', Train_MAE)
print('Root Mean Square Error: ', Train_RMSE)
```

```
Train Set:

Mean Squared Error:  0.007606667924293087
Mean Absolute Error:  0.06567437806954958
Root Mean Square Error:  0.08721621365487661
```

Snippet 10 Model Evaluation

```
#Make Table Displaying Predicted vs Actual Price Values
Table = pd.DataFrame({'Actual': Y_Test, 'Predicted': Y_Te_Pred})
print(Table)
```

```
       Actual  Predicted
163  0.318182   0.361022
221  0.261212   0.247388
268  0.227879   0.265842
55   0.484848   0.273301
515  0.060606   0.160145
..        ...        ...
394  0.151515   0.112052
410  0.145455   0.103534
409  0.145455   0.110319
347  0.180606   0.129495
156  0.326667   0.290924

[164 rows x 2 columns]
```

Snippet 11 Table Displaying Actual vs Predicted Price Values

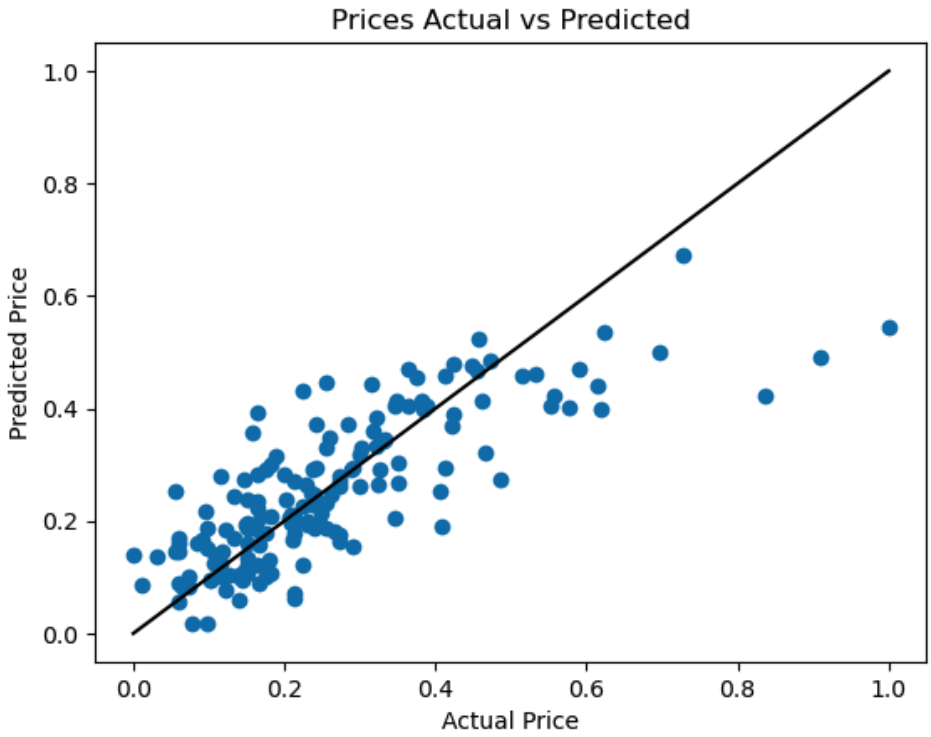

Figure 1 Scatter Plot for Predicted vs Actual Price Values

```
#Heatmap Visualisation
plt.figure(figsize=(12,10))
Correlation = Data.corr()
sns.heatmap(Correlation, annot = True)
plt.show()
```

Snippet 12 Heatmap Visualisation

```
#Scatter Plot Visualisation
plt.scatter(Y_Test, Y_Te_Pred)
plt.plot([Y_Test.min(), Y_Test.max()], [Y_Test.min(), Y_Test.max()], color = 'black')
plt.xlabel('Actual Price')
plt.ylabel('Predicted Price')
plt.title("Prices Actual vs Predicted")
plt.show()
```

Snippet 13 Scatter Plot Visualisation

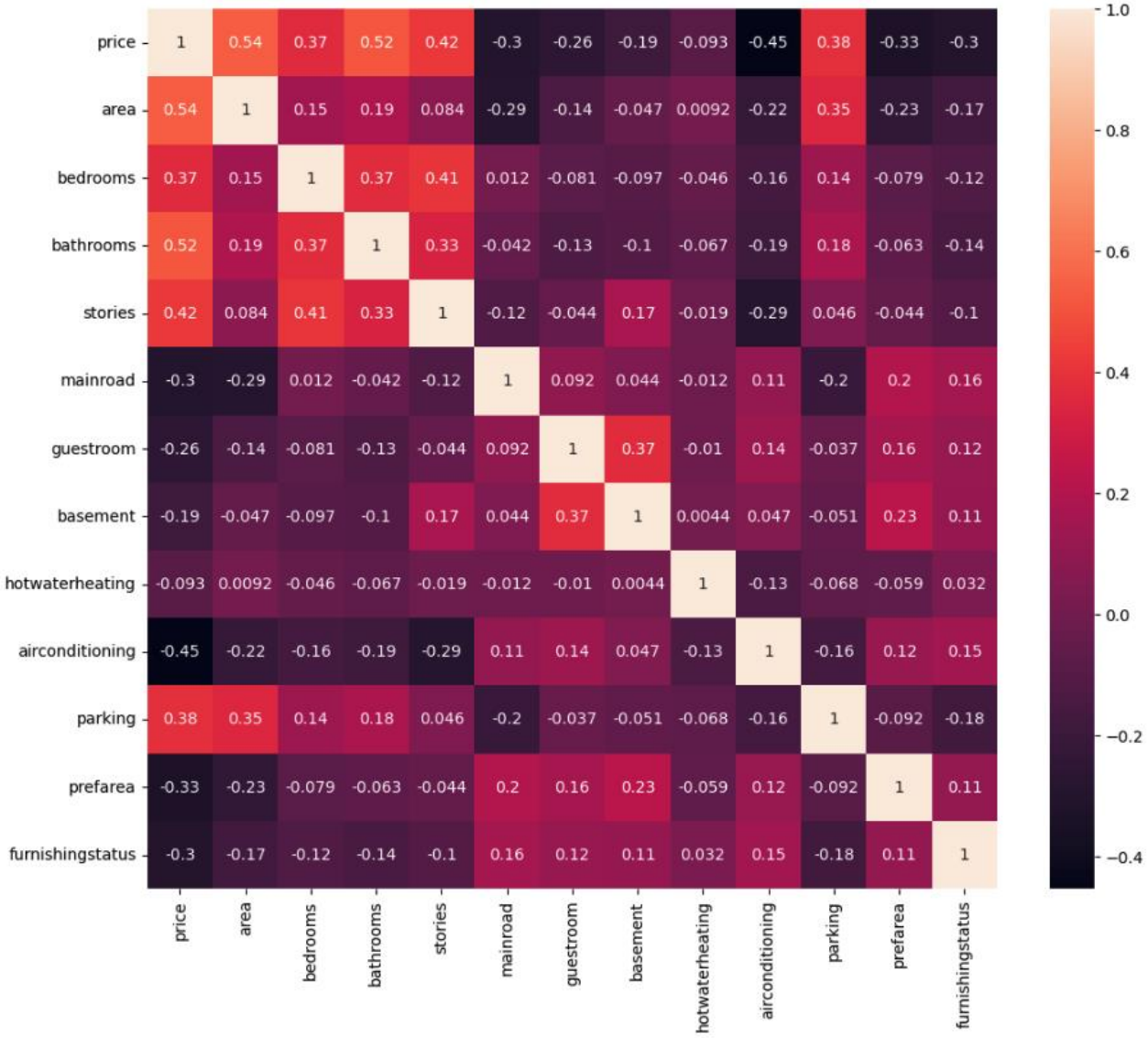


Figure 2 Heatmap Visualisation for Dataset

# 4. References


Al-Khafajiy, D. M. (2023a) Lecture 1 Introduction to Big Data.

Al-Khafajiy, D. M. (2023b) Lecture 2 Frameworks.

Al-Khafajiy, D. M. (2023c) Lecture 4 Recommender Engines.

Al-Khafajiy, D. M. (2023d) Lecture 7 Data Stream Mining.

Aljobs (2024) *SparkML explained*. Available from https://aijobs.net/insights/sparkml-explained/.

Apache (2023) *Apache Hive*. Available from https://hive.apache.org/.

Arcitura (2025) *Resource Manager*. Available from http://patterns.arcitura.com/big-data-patterns/mechanisms/resource_manager#:~:text=A%20resource%20manager%20acts%20as,the%20example%20in%20Figure%201.

Assis, M. R. M., Bittencourt, L. F. (2016) A survey on cloud federation architectures: Identifying functional and non-functional properties. *Journal of Network and Computer Applications*, 72 51-71. Available from https://www.sciencedirect.com/science/article/pii/S1084804516301436 [accessed 2016/09/01/].

Azure, M. (2025) *What is a Data Lake?* Available from https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-a-data-lake.

Barjatiya, P. (2023) *Unleashing the Power of Machine Learning with Spark ML and PySpark ML*. Available from https://medium.com/data-and-beyond/unleashing-the-power-of-machine-learning-with-spark-ml-and-pyspark-ml-9120697c0745#:~:text=Spark%20ML%20is%20the%20machine,work%20with%20Spark%20using%20Python.

Bonner, S., Kureshi, Ibad Brennan, John Theodoropoulos, Georgios (2017) Chapter 14 - Exploring the Evolution of Big Data Technologies. In: I. Mistrik, R. Bahsoon, N. Ali, M. Heisel and B. Maxim (eds.) *Software Architecture for Big Data and the Cloud*. Boston: Morgan Kaufmann, 253-283.

Cloud, G. (2025) *What is a Data Lake?* Available from https://cloud.google.com/learn/what-is-a-data-lake.

Cloud, G. (2025b) *What is Apache Hadoop?* Available from https://cloud.google.com/learn/what-is-hadoop.

Coforge (2025) *Understanding the 3 Vs of Big Data - Volume, Velocity and Variety*. Available from https://www.coforge.com/what-we-know/blog/understanding-the-3-vs-of-big-data-volume-velocity-and-variety#:~:text=Variety%20in%20Big%20Data%20refers,%2C%20tweets%2C%20pictures%20%26%20videos.

Dancuk, M. (2022) *Resilient Distributed Datasets (Spark RDD)*. Available from https://phoenixnap.com/kb/resilient-distributed-datasets#:~:text=Resilient%20Distributed%20Datasets%20(RDDs)%20are,Spark%20speeds%20up%20MapReduce%20processes.

Databricks (2025a) *Apache Hive*. Available from https://www.databricks.com/glossary/apache-hive.

Databricks (2025b) *Introduction to Data Lakes*. Available from https://www.databricks.com/discover/data-lakes.

Flume (2023) *Apache Flume*. Available from https://flume.apache.org/.

Fortinet (2025) *What Is Fault Tolerance?* Available from https://www.fortinet.com/uk/resources/cyberglossary/fault-tolerance#:~:text=Fault%20tolerance%20is%20a%20process,operating%20despite%20failures%20or%20malfunctions.

Framework, B. D. (2024) *The Four V's of Big Data*. Available from https://www.bigdataframework.org/the-four-vs-of-big-data/#:~:text=Volume%20of%20Big%20Data,traditional%20storage%20and%20processing%20capabilities.

Friedrichsen, U. (2022) *Resilience vs. Fault tolerance*. Available from https://www.ufried.com/blog/resilience_vs_fault_tolerance/.

Hariom Singh, H. M., Bryan Conley (2025) *How do you test and monitor the resilience of your software systems?* Available from https://www.linkedin.com/advice/0/how-do-you-test-monitor-resilience-your-software#:~:text=Resilience%20does%20not%20mean%20avoiding,to%20changing%20conditions%20and%20requirements.

Geeksforgeeks (2024) *Fault Tolerance in Distributed System*. Available from https://www.geeksforgeeks.org/fault-tolerance-in-distributed-system/.

Holdsworth, J. (2024) *What is Hadoop Distributed File System (HDFS)?* Available from https://www.ibm.com/think/topics/hdfs.

Imperva (2024) *Fault Tolerance*. Available from https://www.imperva.com/learn/availability/fault-tolerance/.

Indeed (2024) *What Is a Resource Manager? (With Responsibilities)*. Available from https://www.indeed.com/career-advice/finding-a-job/what-is-resource-manager#:~:text=A%20resource%20manager%20is%20a,members%20and%20resources%20they%20need.

James Holdsworth, M. K. (2025) *What is a Resilient Distributed Dataset (RDD)?* Available from https://www.ibm.com/think/topics/resilient-distributed-dataset.

Jha, G. (2024) *Understanding the 6 Vs of Big Data: Unraveling the Core Characteristics of Data-Driven Success*. Available from https://medium.com/@post.gourang/understanding-the-6-vs-of-big-data-unraveling-the-core-characteristics-of-data-driven-success-38e883e4c36b.

Kafka, A. (2024) *Fault Tolerance and Resiliency in Apache Kafka: Safeguarding Data Streams*. Available from https://www.ksolves.com/blog/big-data/apache-kafka/fault-tolerance-and-resiliency-in-apache-kafka.

Kaggle (2022) Housing Prices Dataset. Available from https://www.kaggle.com/datasets/yasserh/housing-prices-dataset?resource=download.

Lahti, C. B., Peterson, Roderick (2005) Chapter 7 - Domain III: Delivery and Support. *Sarbanes-Oxley IT Compliance Using COBIT and Open Source Tools*. Burlington: Syngress, 175-221.

Landau, P. (2024) *Resource Scheduling in Project Management*. Available from https://www.projectmanager.com/blog/better-resource-scheduling#:~:text=Resource%20scheduling%20is%20the%20process,any%20point%20of%20the%20project.

Noergaard, T. (2010) Chapter 7 - An Introduction to the Fundamentals of Database Systems. In: T. Noergaard (ed.) *Demystifying Embedded Systems Middleware*. Burlington: Newnes, 305-328.

Oracle (2022) *What Is a Data Lake?* Available from https://www.oracle.com/uk/big-data/data-intelligence-platform/what-is-data-lake/.

Oracle (2024) *Resource Scheduler Overview*. Available from https://docs.oracle.com/en-us/iaas/Content/resource-scheduler/concepts/resourcescheduler-overview.htm#:~:text=Resource%20Scheduler%20uses%20a%20set,%2C%20disabling%2C%20and%20deleting%20schedules.

Polackoff, B. (2024) *How to Build a SaaS Churn Prediction Model for Customer Churn Analysis*. Avaiable from https://churnassassin.com/blog/how-to-build-a-saas-churn-prediction-model-for-customer-churn-analysis?hs_amp=true.

Paddle (2025) *Customer churn analysis: One of SaaS' most important processes*. Available from https://www.paddle.com/resources/customer-churn-analysis.

Robinson, S. (2023) *5V's of big data*. Available from https://www.techtarget.com/searchdatamanagement/definition/5-Vs-of-big-data.

Segment, T. (2024) *Understanding the 5 V's of Big Data*. Available from https://segment.com/data-hub/big-

data/characteristics/#:~:text=Big%20data%20is%20often%20defined,variety%2C%20veracity%2C%20and%20value.
Solomons, K. (2021) *Resilience, stress tolerance & flexibility*. Available from https://issuu.com/uctcareers/docs/uct_guide_2021_v1.2/s/12379273#:~:text=Resilience%20is%20the%20ability%20we,ability%20to%20adapt%20to%20change.
Stedman, C. (2020) *Apache Hadoop YARN*. Available from https://www.techtarget.com/searchdatamanagement/definition/Apache-Hadoop-YARN-Yet-Another-Resource-Negotiator.
Skuse, J. (2025) *Resource scheduling software: Benefits, features, and how to choose*. Available from https://www.retaininternational.com/blog/resource-scheduling-software#:~:text=Resource%20scheduling%20is%20the%20practice,and%20adapt%20to%20changing%20conditions.
Swilam, M. M. (2025) *predict*. Available from https://www.kaggle.com/code/mayarmohamedswilam/predict.
Świtek, M. (2025) *Resource Scheduler*. Available from https://teamdeck.io/resources/resource-scheduler/.
Tutorialspoint (2025) *Apache Flume - Quick Guide*. Available from https://www.tutorialspoint.com/apache_flume/apache_flume_quick_guide.htm.
Wikipedia (2025) *Fault tolerance*. Available from https://en.wikipedia.org/wiki/Fault_tolerance#:~:text=In%20resilience%2C%20the%20system%20adapts,despite%20hardware%20or%20software%20issues.
Wingate, J. (2023a) Lecture 7 Big Data Cleansing.
Wingate, J. (2023b) Lecture 8 Apache Hadoop.
Yu, D. M. (2025a) Hadoop Part 1.
Yu, D. M. (2025b) Introduction of Big Data.
Yu, D. M. (2025c) Apache Spark System.
Yu, D. M. (2025d) Hadoop Part 2.
Yu, D. M. (2025e) Data analytic and modelling flow.
Zdravevski, E., Lameski, P., Apanowicz, C. and Ślęzak, D. (2020) From Big Data to business analytics: The case study of churn prediction. *Applied Soft Computing*, 90 106164. Available from https://www.sciencedirect.com/science/article/pii/S1568494620301046 [accessed 2020/05/01/].

# 5. Appendix

Housing.ipynb

```
#Author: Tarteel Alkaraan (25847208)
#Last Updated: 07/03/2025
#This is an edited and extended version of (Yu, 2025e)
#Import Libraries
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, mean_absolute_error

#Read Data
Data = pd.read_csv('Housing.csv')
Data

#Check Data Information
Data.info()

#Perform Descriptive Statistics
Data.describe()

#Check Unique Values of every row for every column
N = Data.nunique(axis = 0)
print(f'Number of Unique Values in Every Column:\n{N}\n')

#Count NaN Values in Data
CountNaN = Data.isna().sum()
print(f'Number of Missing Values in Every Variable:\n{CountNaN}\n')

#Change Categorical Variable mainroad into Numerical
Labels = {'yes': 1, 'no': 2}
Data['mainroad'] = Data['mainroad'].map(Labels)

#Change Categorical Variable guestroom into Numerical
Labels = {'yes': 1, 'no': 2}
Data['guestroom'] = Data['guestroom'].map(Labels)

#Change Categorical Variable basement into Numerical
Labels = {'yes': 1, 'no': 2}
Data['basement'] = Data['basement'].map(Labels)

#Change Categorical Variable hotwaterheating into Numerical
Labels = {'yes': 1, 'no': 2}
Data['hotwaterheating'] = Data['hotwaterheating'].map(Labels)

#Change Categorical Variable airconditioning into Numerical
```

```
Labels = {'yes': 1, 'no': 2}
Data['airconditioning'] = Data['airconditioning'].map(Labels)

#Change Categorical Variable prefarea into Numerical
Labels = {'yes': 1, 'no': 2}
Data['prefarea'] = Data['prefarea'].map(Labels)

#Change Categorical Variable furnishingstatus into Numerical
Labels = {'furnished': 1, 'semi-furnished': 2, 'unfurnished': 3}
Data['furnishingstatus'] = Data['furnishingstatus'].map(Labels)
Data.head()

#Data Normalisation
Scaler = MinMaxScaler()
SC = ['price', 'area']
Data[SC] = Scaler.fit_transform(Data[SC])
Data

#Preparing Model Training/Testing Data
X = Data.drop('price', axis = 1)
X = pd.get_dummies(X, drop_first = True)
Y = Data['price']

X_Train, X_Test, Y_Train, Y_Test = train_test_split(X, Y, test_size = 0.3)

#Model Training and Testing
Regressor = LinearRegression()
Regressor.fit(X_Train, Y_Train)

Y_Tr_Pred = Regressor.predict(X_Train)
Y_Te_Pred = Regressor.predict(X_Test)

#Train and Test Data Model Evaluation
Test_MSE = mean_squared_error(Y_Test, Y_Te_Pred)
Test_MAE = mean_absolute_error(Y_Test, Y_Te_Pred)
Test_RMSE = np.sqrt(Test_MSE)

Train_MSE = mean_squared_error(Y_Train, Y_Tr_Pred)
Train_MAE = mean_absolute_error(Y_Train, Y_Tr_Pred)
Train_RMSE = np.sqrt(Train_MSE)

#Print Test Set Model Evaluation
print('Test Set:')
print('\nMean Squared Error: ', Test_MSE)
print('Mean Absolute Error: ', Test_MAE)
print('Root Mean Square Error: ', Test_RMSE)
Test Set:

#Print Train Set Model Evaluation
print('Train Set:')
print('\nMean Squared Error: ', Train_MSE)
```

```
print('Mean Absolute Error: ', Train_MAE)
print('Root Mean Square Error: ', Train_RMSE)

#Make Table Displaying Predicted vs Actual Price Values
Table = pd.DataFrame({'Actual': Y_Test, 'Predicted': Y_Te_Pred})
print(Table)

#Heatmap Visualisation
plt.figure(figsize=(12,10))
Correlation = Data.corr()
sns.heatmap(Correlation, annot = True)
plt.show()

#Scatter Plot Visualisation
plt.scatter(Y_Test, Y_Te_Pred)
plt.plot([Y_Test.min(), Y_Test.max()], [Y_Test.min(), Y_Test.max()], color = 'black')
plt.xlabel('Actual Price')
plt.ylabel('Predicted Price')
plt.title("Prices Actual vs Predicted")
plt.show()
```