UNIVERSITY OF
LINCOLN

CMP9781 Big Data Analytics & Modelling Assessment 1

School of Engineering and Physical Sciences

25847208@students.lincoln.ac.uk

Tarteel Alkaraan (25847208)

University of Lincoln

Table of Contents

1. Description of Distributed Big Data Processing Ecosystem

1.1 5 V's

1.1.1 Volume

Volume of data that needs to be processed is increasing rapidly more storage capacity, computation and tools and techniques (Yu, 2025b). Think about how many devices there are currently producing data, compared to say 10 or 20 years ago. Think about what amount of data would be produced in 10 or 20 years from now. Think about the rate of growth of data (Al-Khafajiy, 2023a). Refers to amount of big data being generated at a minimum, many terabytes but also as much as petabytes (segment, 2024). It is like base of big data as it is initial size and amount of data that is collected. If volume of data is large enough, it can be considered big data (Robinson, 2023). Volume of big data refers to the size of the data sets that need to be analysed and processed, that are regularly larger than terabytes and petabytes (Framework, 2024).

1.1.2 Velocity

Big data is being generated fast everyday 900 million photos are uploaded to Facebook, 500 million tweets are uploaded to Twitter, 3.5 billion searches are performed on Google daily, and big data needs to be processed fast in which search results need to be returned immediately (Yu, 2025b). Data is being produced an at alarming rate but think about how we need to analyse it. We need to be timely in our analysis, which means we often must have a real-time data. Funnily enough companies used to run "batch processes" overnight to do some of this. Think about how we efficiently analyse data streams Al-Khafajiy, 2023). Velocity refers to how quickly data is generated and how fast it moves. This is a crucial aspect for companies that require their data to run fast, therefore its available at the correct times to make the perfect organisation verdicts probable (Robinson, 2023).

1.1.3 Variety

Various formats, types, and structures like text, numerical, images, audio, video, sequences, time series, social media data, and multi-dimensional arrays. Static versus streaming data, a single application can be generating and or collecting many types of data. To extract knowledge all these types of data need to be linked together (Yu, 2025b). Think about all different sensors out in world, all producing data. Consider what forms of data might be used, things like cameras, pressure sensors, and GPS trackers. Also think about how we might need not just raw data, but also data that draws in associated data from other sources (Al-Khafajiy, 2023a). Variety in big data refers to all the structured and unstructured data that has the possibility of getting generated either by humans or by machines. The most added are structured texts, tweets, pictures and videos (Coforge, 2024).

1.1.4 Value

The most important V of big data, it is all well and good having access to big data but unless we can turn it into value it is useless. It is important that businesses clearly understand costs and benefits associated with big data (Yu, 2025b). Data does not have any reason to exist if we are not going to use it. But clearly we can use data to gain a variety of insights given some thought. Worth thinking about ethics at this point (Al-Khafajiy, 2023a). Refers to actionable insight that can be derived from big data sets (segment, 2024). Also, refers to benefits that big data can provide, and it relates directly to what organisations can do with that collected data (Robinson, 2023). Value refers to the usefulness or organisation effect that could be derived from data. Whereas businesses are gathering huge quantities of data, the actual question is whether they could excerpt valuable perceptions from it (Jha, 2024).

1.1.5 Veracity

Refers to messiness or trustworthiness of data. With many forms of big data, quality and accuracy are less controllable (think of twitter posts with hashtags, abbreviations, typos, colloquial speech, reliability, and accuracy of content). But big data and analytics technology now allows us to work with these types of data (Yu, 2025b). Refers to inconsistencies and uncertainty in data, that is data which is available can sometimes get messy and quality and accuracy are difficult to control. Think about what would happen if we allowed intrusion into GPS network and flight systems were entirely automated. Think about how we determine valid data from invalid data (Al-Khafajiy, 2023a). Veracity refers to the quality, accuracy, integrity, and credibility of data. Collected data can have missing pieces, maybe imprecise or may not be able to give actual, valuable perception (Robinson, 2023).

1.2 Fault Tolerance & Resilience

1.2.1 Fault Tolerance

Fault Tolerance is a process that enables an operating system to respond to a failure in hardware or software. This fault-tolerance definition refers to system's ability to continue operating despite failures or malfunctions (Fortinet, 2025). Fault Tolerance refers to ability of a system (computer, network, and cloud cluster) to continue operating without interruption when one or more of its components fail (Imperva, 2024). Fault Tolerance is the ability for a system or application to continue service without interruption in the event of a hardware or software failure. The goal of fault tolerance is to prevent a fault from manifesting itself as a failure (Lahti, 2005). Fault Tolerance is the capability to resume operating effortlessly despite failures or errors in one or more of its components. This resilience is necessary for maintaining system dependability, obtainability, and consistency (Geeksforgeeks, 2024).

1.2.2 Fault Resilience

Fault Resilience is ability of a software system to recover from failures and continue to function correctly. It doesn't mean avoiding failures but rather minimising their impact and frequency (Linkedin, 2025). Fault Resilience points towards systems that will respond gracefully to not yet known failure modes, develop emergent resilient behaviour, adapt to changing failure surfaces or even transform based on them (Friedrichsen, 2022). Fault Resilience is the ability of a system to withstand adversities and bounce back to a normal state. In short, it refers to the ability to adapt to difficulties and recover from unexpected events (Kafka, 2024). Fault Resilience system adjusts to error, maintaining service however admitting a specific impact on performance (Wikipedia, 2025). Fault Resilience is capability we develop to withstand adversity and permits us to recover rapidly from tough events (Solomons, 2024).

1.3 Hadoop Distributed File Systems (HDFS)

HDFS is specially designed for storing huge datasets in commodity hardware. HDFS contains two types of nodes name and data. In HDFS a large file is split into one or more blocks and these blocks are stored in a set of DataNodes (Yu, 2025a). HDFS is a distributed file system, designed to be highly fault tolerant. HDFS splits data into chunks and distributes it across the machines. HDFS is designed to detect and recover from faults quickly, and assumes every node is unreliable (Wingate, 2023b). HDFS is a file system that manages large data sets that can run on commodity hardware. HDFS is most popular data storage system for Hadoop and can be used to scale single Apache Hadoop cluster (Holdsworth, 2024). HDFS acts as the primary component of the Hadoop ecosystem. HDFS is a distributed file system in which individual Hadoop nodes operate on data that lives in their local storage (Cloud, 2025b).

1.4 Resource Manager & Scheduler

1.4.1 Resource Manager

Resource Manager is defined as the component responsible for managing the interaction with resources across multiple clouds in a federation. It abstracts and maps resources as a unified collection, utilising APIs provided by cloud providers or open standards for management (Assis, 2016). Resource Manager acts as a scheduler that schedules and prioritises processing requests according to individual processing workload requirements (Arcitura, 2025). Resource Manager is responsible for keeping track of the data with the data structures of the database, to allow for efficient retrieval of data from storage via the buffer and storage manager (Noergaard, 2010). Resource Manager arbitrates resources among all the applications in the system (Yu, 2025d). A resource manager is a specialist that helps organisations assign their resources effectively (Indeed, 2024).

1.4.2 Resource Scheduler

Resource Scheduler is process of identifying when project resources are needed and allocating them based on factors like capacity planning or resource availability. Main purpose of resource scheduler is to guarantee that there's no over or under-allocation of resources at any point of project (Landau, 2024). Resource Scheduler uses a set of APIs to create work requests to complete specific tasks such as creating, updating, starting, stopping, enabling, disabling, and deleting schedules (Oracle, 2024). Resource Scheduler helps allocate tasks, manage time, optimise use of available resources, ensuring that projects run efficiently and meet deadlines (Świtek, 2025). Resource scheduling is practice of scheduling resources to customers and projects, whereas meeting project deadlines within financial plan. Resource scheduling software maintains everything automated and streamlined (Skuse, 2025).

1.5 YARN

Apache Hadoop Yet Another Resource Negotiator (YARN) is a resource management system. It provides unified resource management and scheduling for upper-layer applications, remarkably improving cluster resource utilisation, unified resource management, and data sharing (Al-Khafajiy, 2023b). YARN introduced in Hadoop 2 for better resource allocation and scalability. YARN splits resource management and job scheduling/monitoring into separate processes. This enables Hadoop to move beyond MapReduce and do all sorts of things (Wingate, 2023b). YARN is resource management and job scheduling technology in open source Hadoop distributed processing framework (Stedman, 2020). Yarn technology is designed for cluster/resources management. YARN is an Apache Hadoop technology (Yu, 2025d).

1.6 Apache Spark, Hive, Flume & Kafka

1.6.1 Apache Spark

Apache Hadoop/Spark is a framework that allows for distributed processing of large data sets across clusters of computers using simple programming models. Apache designed to scale up from single servers to thousands of machines, each offering local computation and storage. Rather than rely on hardware to deliver high availability, the library itself is designed to detect and handle failures at the application layer, so delivering a highly available service on top of a cluster of computers, each of which may be prone to failures. Apache spark is a unified analytics engine for large-scale data processing (Al-Khafajiy, 2023b). Apache Spark is an open-source data processing engine to store and process data in real-time across numerous clusters of computers using simple programming constructs (Yu, 2025c).

1.6.2 Apache Hive

Apache Hive data warehouse software facilitates reading, writing, and managing large datasets residing in distributed storage using SQL. Structure can be projected onto data already in storage. A command line tool and JDBC driver are provided to connect users to Hive (Al-Khafajiy, 2023b). Apache Hive is an open-source data warehouse software designed to read, write, and manage large datasets extracted from HDFS (Databricks, 2025a). Apache Hive is a distributed, fault-tolerant data warehouse system that enables analytics data at a massive scale. Hive is built on top of Apache Hadoop and allows users to read, write, and manage petabytes of data using SQL (Apache, 2023). Apache Hive is an open-source data warehouse software for reading, writing, and managing large data set files that are stored directly in either HDFS or other data storage systems like Apache HBase (Yu, 2025c).

1.6.3 Apache Flume

Apache Flume is a distribute, reliable, available service for efficiently collecting, aggregating, and moving large amounts of log data. It has a simple and flexible architecture based on streaming data flows. It is robust and fault tolerant with tunable reliability mechanisms and many failover and recovery mechanisms (Flume, 2023). Flume is a tool/service/data ingestion mechanism for collecting, aggregating, and transporting large amounts of streaming data from various sources to a centralised data store. Flume is a highly reliable, distributed, and configurable tool. It is designed to copy streaming data from various web servers to HDFS (TutorialsPoint, 2025). Apache Flume is a distributed, reliable, and available system for efficiently collecting, aggregating, and moving large amounts of log data from many different sources to a centralised data store (Yu, 2025c).

1.6.4 Apache Kafka

Apache Kafka is an open-source distributed event streaming platform used by thousands of companies for high-performance data pipelines, streaming analytics, data integration, mission-critical applications. More than 80 percent of all Fortune 100 companies trust and use Kafka (Al-Khafajiy, 2023b). Kafka is an example of what is called a publisher/subscriber messaging system aka pubsub. Point of Apache Kafka is to act as a central event/message queue for real-time apps. Kafka messages aka events are separated into topics. Kafka topics are also partitioned so they can reside across a cluster and be parallelised (Al-Khafajiy, 2023d). Kafka is an open-source stream-processing software platform developed by Apache Software Foundation, written in Scala and Java. project aims to provide a unified, high-throughput, low-latency platform for handling real-time data feeds (Yu, 2025c).

1.7 Resilient Distributed Datasets

A RDD is a read-only collection of data in spark that can be partitioned across multiple machines in a cluster, allowing for parallel computation and fault tolerance through lineage reconstruction (Bonner, 2017). RDDs are primary data structure in Spark. RDDs are reliable and memory efficient when it comes to parallel processing. By storing and processing data in RDDs, Spark speeds up MapReduce processes (Dancuk, 2022). RDD is an immutable, fault-tolerant collection of elements that can be distributed across multiple cluster nodes to be processed in parallel. RDDs are basic data structure within open-source data processing engine Apache Spark (James Holdsworth, 2025). RDDs read-only dataset partitioned across a set of machines. RDD can be rebuilt if a partition is lost. RDD saves time taken in reading, writing operations and so it runs nearly ten to hundred times faster than Hadoop (Yu, 2025c).

1.8 Data Lakes

Data lake is a central location that holds a large amount of data in its native, raw format. Compared to a hierarchical data warehouse, which stores data in files or folders, a data lake uses a flat architecture and object storage to store data (Databricks, 2025b). Data lake is a centralised repository designed to store, process, and secure large amounts of structured, semi-structured, and unstructured data. It can store data in its native format and process any variety to it, ignoring size limits (Cloud, 2025). Data lake is a place to store your structured and unstructured data, as well as method for organising large volumes of highly diverse data from diverse sources (Oracle, 2022). Data lake is a centralised repo that ingests, stores, and permits for processing of huge volumes of data in its original form. It could accommodate all forms of data, that's used to power big data analytics and ML (Azure, 2025).

1.9 Spark ML

Spark ML is the machine learning module of Apache Spark, an open-source distributed computing system designed to process large-scale data. And PySpark ML is the Python library for Spark ML, which makes it simpler to work with Spark using Python. It provides high-level APIs for common machine learning tasks like classification, regression, and clustering (Barjatiya, 2023). Spark ML a scalable and easy-to-use framework for building machine learning models on large datasets. It is designed to simplify the process of developing machine learning applications by offering a high-level API that integrates seamlessly with Sparks core functionalities (Aljobs, 2024). Spark ML is a low-level machine learning library that is simple to use, is scalable, and compatible with various programming languages. Spark ML eases the deployment and development of scalable ML algorithms (Yu, 2025c).

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

Recommendation aims to predict what items a user may like and how much a user may like or dislike these items. This is solely based on preferences and associations of the user with previous items or similarities among items. Item based recommendation find similarities between items. If user a likes item x and item x is like item y, recommend y to a. Item based recommender does not scale well with the number of items and a user-based recommenders does not scale with the number of users. We use item-based recommenders when the number of items is relatively low compared with number of users and vice versa. Items are less subject to change, therefore the more data we acquire the better estimates of the similarities will converge (Al-Khafajiy, 2023c).

Recommendation system recommends products, services, and data to customers based on analysis of collected information. Recommendation system, often named recommendation engine, represents a form of information filtering tool which uses Machine learning algorithms and AI to suggest the most relevant product to a particular client at the correct moment. Additional information an organisation has, it would be simpler to excel in the development of a recommendation engine. Therefore, it could make suggestions that would make additional income and upsurge client satisfaction because of appropriate discounts and a tailored shopping experience (Ana Jacimovic, 2021).

Three major forms of recommendation systems are content-based filtering, collaborative filtering, and hybrid recommender systems. These forms could depend on user behaviour information, consisting of activities, preferences, and likes, or could consider description of a product that a user prefer, or both. Content-based filtering this form acts based on the properties of the products that every user likes, discovering what else the user might like. It considers multiple keywords. In addition, a user profile is designed to give comprehensive data on the products that a user prefers. The system next suggests some comparable products that users might additionally would like to buy (Valeryia Shchutskaya, 2021).

Collaborative filtering recommendations engines could depend on likes and desires of other users to calculate a similarity index among users and suggest products to them consequently. This form of filtering depends on user view instead of machine analysis to precisely suggest complex products, like movies or music tracks. Collaborative filtering algorithm has some details. System could look for similar users, that would be user-user collaborative filtering. Hence, recommendations would rely on a user profile. However, such an approach needs a huge number of computational resources and would be complex to apply for wide-range databases. A different choice is item-item collaborative filtering. (Valeryia Shchutskaya, 2021).

System would discover comparable products and suggest these products to a user on a case-by-case basis. It’s a resource-saving method and companies uses it to involve clients and increase sales volumes. Hybrid recommender systems its additionally probable to mix both types to construct a more prosperous recommendation engine. This approach is used to produce collaborative and scenario-specific predictions and drag them all together to improve performance. Company providers of media services utilises a hybrid system to earn client loyalty. Users receive movie suggestions based on their habits and characteristics of content they prefer (Valeryia Shchutskaya, 2021).

When it comes to volume the word itself says it all, Big Data is precisely that large. When we talk about velocity, we talk about how quick we could collect the information we require its hugely necessary to have live information at any point to do better business decisions quicker. Variety when we have accumulated information, we need to understand the diverse kinds of information there are. Veracity when we have the information it needs to pass a credibility and quality test. Finally, value information musts be convenient for your company it needs to provide you data that would give several advantages to your company (Tudor, 2021).

Amazon excels in the e-commerce business because of the excellent mix of big data usages and their recommendation system models. At each step of client's journey, Amazon targets their clients by giving applicable and convenient product suggestions. That makes buying on a website completely tailored and takes benefit of organised, but also impulsive buying. The value of this method is that the store could be altered due to clients' interests and provide an excellent product with excellent timing (Ana Jacimovic, 2021).

Amazon is identified as the number 1 e-commerce shop at this moment, and they have their database to thank for that. They are always using big data to improve their client experience. The dynamic pricing is implemented on Amazon's website. However, what you potentially didn't know is that they alter their prices up to 2.5 million times every day. Product recommendation it's not important if the client buys the products, add it to the basket, or even just sees it amazon would utilise that information. In which they could learn what every client needs and likes and could suggest that exact product or comparable ones to them when they come back to the store (Tudor, 2021).

Amazon has flourished by adapting an everything under one roof model. But when challenged with such a big variety of choices, clients could frequently feel overcome. They successfully become information-gain, with lots of choices, however, insight-poor, with small idea around what could be the perfect buying verdict for them. To solve this, amazon utilises big data collected from clients while they search to create and fine-tune its recommendation engine. The more amazon knows about you, the greater it could foresee what you want to purchase. And when the seller knows what you would need, it could rationalise the procedure of influencing you to purchase it (Marr, 2021).

3. Linear Regression Analysis

Data cleansing is process of detecting and correcting/removing certain data from an overall set to leave only high-quality data behind. Typically, a key component of pre-processing prior to using data for its intended purpose. Data cleansing is techniques used for data cleansing are context-dependent, but the overall aim is mostly the same: identify incorrect data and remove, flag, or impute. Data profiling is the general identification of data types, missing and unique values, and relationships. Visualisation is the closely tied to data profiling, analysis of data via visual methods like graphs. Dedicated software is software that allows definition of constraints that can then be applied to a set of data (Wingate, 2023a).

We can think of data quality as being measured into five distinct categories. Validity is the degree to which data conforms to rules or constraints defined by the use case. Accuracy is the degree to which the data is close to true or possible values. Completeness is the degree to which all required data is known. Consistency is the degree to which the data is consistent either within the same set or across multiple sets. Uniformity is the degree to which the data is specified using the same units of measure. irrelevant data is data that doesn't fit under intended use case for data. Duplicate data is data that is a copy of another data point which doesn't need to be exact and can be partial. Syntactically inconsistent data is data that is incorrectly formatted compared to other records in column. (Wingate, 2023a).

Missing data is data that has NaN, empty, or equivalent value recorded. Missing data is a very common occurrence and often must be solved at data cleansing stage rather than by other means. Missing Completely at Random (MCAR) is probability of data being missing is completely independent of data. Missing at Random (MAR) is probability of data being missing is independent of missing values given observed data. Finally, Missing Not at Random (MNAR) is probability of data being missing is independent of observed data. We can predict how likely data is to be missing based on non-missing data. Probability of data being missing depends on missing values themselves (Wingate, 2023a).

Dataset that was selected to develop a pipeline for running a simple linear regression is house prices dataset from Kaggle (Kaggle, 2022). First step is to import necessary libraries to load dataset file using pandas which can be seen in snippet 1 below. Second step would be to check data information as seen below in snippet 2. Next step is to perform descriptive statistics as seen below in snippet 3. After we check unique values of every row for every column that can be seen in snippet 4 below. Following that we count NaN values in data which can be seen below in snippet 5. Now we change all the categorical variables into numerical as seen in snippet 6. Next we normalise data in dataset that can be seen in snippet 7 below.

Now we prepare model training and testing data from dataset which can be seen below in snippet 8. After we perform model training and testing as seen in snippet 9 below. After that we perform model evaluation which you can see below in snippet 10. For test set used from dataset generated table showing predicted vs actual price values as seen in snippet 11 below. Then we display heatmap visualisation that can be seen below in snippet 12 and figure 1. Finally, we perform a visualisation of our data modelling as a scatter plot showing predicted vs actual price values as seen in snippet 13 and figure 2 below.

```python
#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
```

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

Snippet 1 Import libraries and Read Data

```python
#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:  0.009256315924960958
Mean Absolute Error:  0.07113237208359516
Root Mean Square Error:  0.0962097496356838
```

```
#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:  0.008171583154797674
Mean Absolute Error:  0.06684206754287085
Root Mean Square Error:  0.09039680942819649
```

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
327  0.193939   0.280480
339  0.184848   0.258977
399  0.151515   0.200457
341  0.181818   0.233682
534  0.030303   0.144210
..        ...        ...
80   0.422424   0.339816
351  0.175758   0.093425
511  0.066667   0.051661
208  0.272727   0.173816
118  0.363636   0.346527
```

Snippet Table Displaying Actual vs Predicted Price Values

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

Snippet 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 Scatter Plot Visualisation

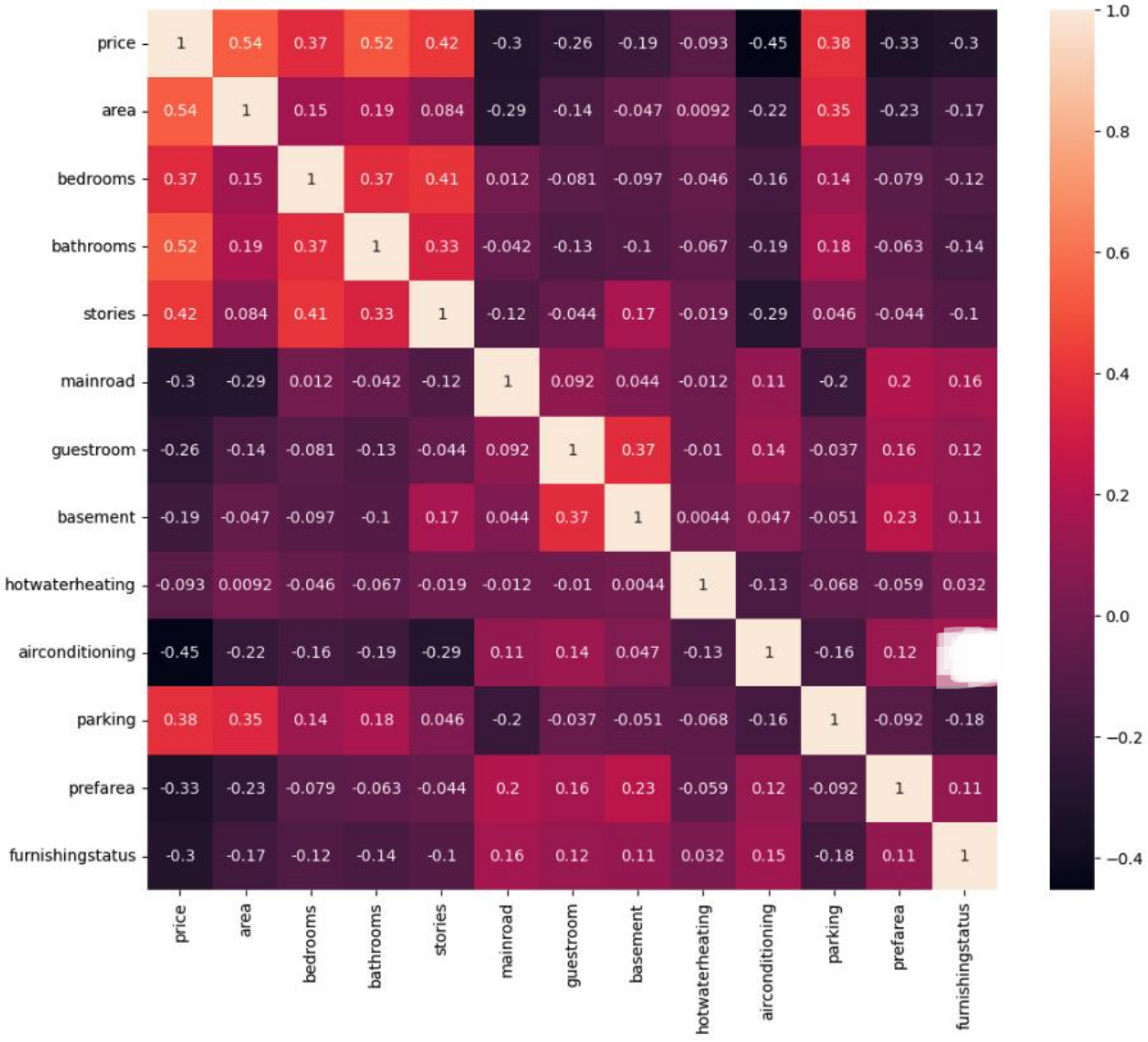

Figure 1 Heatmap Visualisation for Dataset

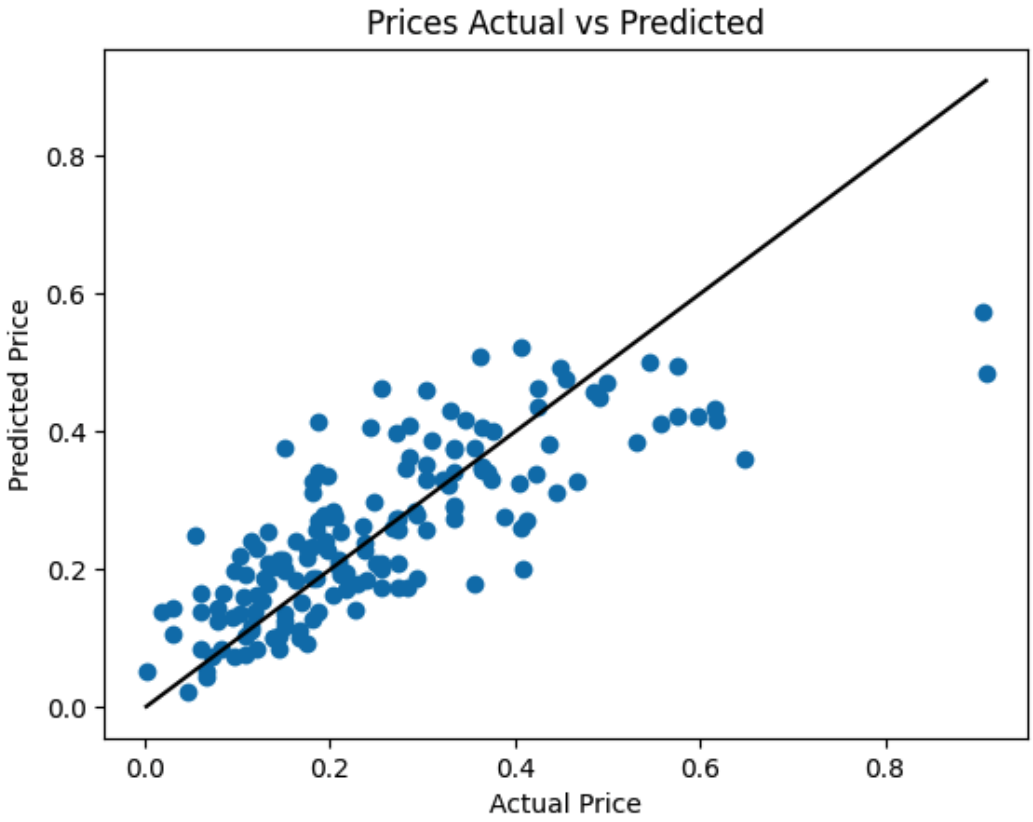

Figure 2 Scatter Plot for Predicted vs Actual Price Values

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/.

Ana Jacimovic, A. V. (2021) *How big data is used in recommendation systems to increase sales?* Available from https://selectacrm.app/blog/how-big-data-is-used-in-recommendation-systems-to-increase-sales/.

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/.

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.

Linkedin (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.

Marr, B. (2021) *Amazon: Using Big Data to understand customers*. Available from https://bernardmarr.com/amazon-using-big-data-to-understand-customers/.

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.

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/.
Tudor, N. (2021) *7 REAL-WORLD EXAMPLES OF HOW BRANDS ARE USING BIG DATA ANALYTICS*. Available from https://www.bornfight.com/blog/7-real-world-examples-of-how-brands-are-using-big-data-analytics/.
Tutorialspoint (2025) *Apache Flume - Quick Guide*. Available from https://www.tutorialspoint.com/apache_flume/apache_flume_quick_guide.htm.
Valeryia Shchutskaya, K. S. (2021) *Big Data Behind Recommender Systems*. Available from https://indatalabs.com/blog/big-data-behind-recommender-systems.
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.

5. Appendix

Housing.ipynb

```
#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()
```