Assignment 1 PORTFOLIO

BEng in Software Engineering

Table of Contents

List of Figure

List of Table

1.0 Introduction:

1.1 Background:

In the last two-decade machine learning has gain exponential popularity. The main reason being abundance of data which is a bi product of our digital footprint on the back of the internet revolution. Unlike traditional programming machine learning not rule based and human centric rather data driven and has the flexibility of adaptation and generalisation.

1.2 How the documents designed:

The study will layout theory on its first task and practice on the second task. In task 1 the concept is taken one step further by showing mathematical and python coding implementation. The core algorithm that makes Machine learning with endless real life use case is the prime consideration in the subsequent parts of this document.

1.3 Tools and technologies used:

For Task 1, which delves into the theoretical dimensions of the study, Jupyter Notebook employed. This choice was guided by the platform's intuitive interface and its seamless compatibility with the hardware, a MacBook Air equipped with 8 GB of RAM and running on Mac OS 14.0. The utilization of Jupyter Notebook in this phase highlights a strategic alignment of the software's functionalities with the task's specific requirements and the existing hardware setup.

Moving to Task 2, cantered on housing price analysis, strategically opted for Google Colab. This decision was primarily influenced by the opportunity to access Google Colab's complimentary GPU resources. While the use of such advanced computational resources was not a fundamental requirement for the task, the initiative is to employ them demonstrates a proactive approach in experimenting with larger datasets.

2.0 Task 1:

2.1 Selection and Justification of ML Approach

In this segment, the task at hand involves the careful selection of the most suitable machine learning (ML) type, tailored to the specifics of the given scenario. This process will be accompanied by a thorough rationale, providing a clear justification for the chosen ML approach. (Brownlee, 2019)

2.1.1 Real Estate Pricing Forecast:

Supervised Learning would be the appropriate selection in this scenario, for constructing a predictive model. The abundance of labelled data - historical property data (features such as square footage, location, amenities, etc.) along with the corresponding actual sale prices. Supervised learning algorithms has the ability utilize this labelled dataset to learn the complex relationships between property features and pricing. By training on this data, the model can effectively forecast home prices. The model's ability to make predictions will highly depend upon the feedback loop established during training, the model learns to adjust its internal parameters to minimize prediction errors, ultimately providing prospective buyers and sellers with a reliable estimation of property values.

2.1.2 Customer Segmentation for Retail:

For the retail sector for customer segmentation indicates unsupervised learning to be the ideal choice. As the name suggest the task aligned with clustering techniques such as K-Means, should be the idea candidate and Unsupervised Learning sublime for this context. In Unsupervised learning the K-Means algorithm able to extract naturally occurring patterns or clusters within the customer dataset on the basis of shared traits or behaviours. This approach is extremely fitting for grouping customers into segments with identical consumer behaviour that indicates the specific purchasing habits. The marketing team can ultimately use those insights that eventually aiding in more effective marketing campaigns and customer targeting.

2.1.3 Email Classification for an Email Service Provider:

This task specific to binary classification. Main objective is to automatically classifying emails as spam or non-spam. The task associates with Supervised Learning that supplies labelled dataset containing examples of both spam and non-spam emails. Popular learning models in labelled data, like logistic regression or support vector machines, can be trained on dataset to learn the classifiable patterns and characteristics that differentiate spam from non-spam emails. This method affrim that the algorithm grasps explicit feedback through labelled data, enabling it to make accurate classifications and effectively filter out unwanted emails from users' inboxes.

2.1.4 Autonomous Driving Agent for Self-Driving Car:

Reinforcement Learning would be the ideal approach for creating an autonomous driving agent. In this scenario, the agent interacts with its ecosystem (the road network) just like the player interact with the world in a gaming context and learns from these interactions. Reinforcement Learning is the way that leads the agent to take actions (e.g., steer, brake or accelerating and slowing down) based on the current state (e.g., vehicle position, speed) and receive feedback either in rewards or penalties (e.g., reaching the finish point without having collisions). In the learning path through trial and error; the agent optimizes its decision-making process, ultimately learning how to navigate complex road networks, adhere to traffic laws, and make informed real-time decisions that will assist to reach the ultimate goal that is passenger safety.

2.2 K-Means Clustering vs. K Nearest Neighbours (KNN) Classification:

The key dissimilarity between K-Means clustering and KNN classification grounded on their fundamental purpose and methodology. The following table shall elucidate the clear difference by sorting by importance, considering the significance of each aspect between K-Means Clustering and K Nearest Neighbours (KNN) Classification:

Aspect	*K-Means Clustering*	*K Nearest Neighbours (KNN) Classification*
Learning Type	Unsupervised Learning.	Supervised Learning.
Data Requirement	Unlabelled data.	Labelled data.
Purpose	Grouping data points into clusters.	Classifying data points into classes.
Algorithm Objective	Minimize intra-cluster variance.	Assign class labels based on neighbours.
Training	No explicit training needed; clustering completed based on data distribution.	Requires training on labelled data for classification
Use Cases	Customer segmentation, Anomaly detection, Image compression	Handwriting recognition, Spam email classification, Recommender systems
Number of Clusters/Neighbours (K)	Must be specified beforehand	Must be specified beforehand.
Distance Metric	Measures likeness between data points (e.g., Euclidean distance)	Measures likeness between data points based on nearest neighbours
Decision Boundary	Doesn't define clear decision boundaries; clusters are regions of similar data points.	Defines decision boundaries based on K nearest data points.
Output	Cluster assignments for each data point.	Class labels for each data point.
Real-Life Examples	Social Media Friend Grouping, Identifying Customer Behaviour Patterns, Document Clustering	Disease Diagnosis, Autonomous Vehicles, Predictive Maintenance

Table 1 Distinction between K means Cluttering and KNN

Real Python (2020)

2.3 Calculating Loss in Machine Learning for Linear Regression:

2.3.1 Theoretical brief

'Loss' or 'cost' functions in machine learning are widely used to measure how well a model's predictions align with the actual target values. The loss function measures the difference or error between the predicted values generated by the model essentially the difference between true & target values. The aim is to minimize this loss during training, since a lower loss suggests that the model is making more accurate predictions.

2.3.2 Implementation

Taking the step further below the concept is implemented in python code inserted below:

In [1]:
```
# Import necessary libraries
import numpy as np
import matplotlib.pyplot as plt

# Generate random data
np.random.seed(0)
X = 2 * np.random.rand(100, 1)
y = 4 + 3 * X + np.random.randn(100, 1)
```

In [2]:
```
# Define the Linear Regression model class
class LinearRegression:
    def __init__(self):
        self.theta = None

    def fit(self, X, y):
        X_b = np.c_[np.ones((X.shape[0], 1)), X]
        self.theta = np.linalg.inv(X_b.T.dot(X_b)).dot(X_b.T).dot(y

    def predict(self, X):
        X_b = np.c_[np.ones((X.shape[0], 1)), X]
        return X_b.dot(self.theta)
```

In [3]:
```
# Create a Linear Regression model instance
lin_reg = LinearRegression()
```

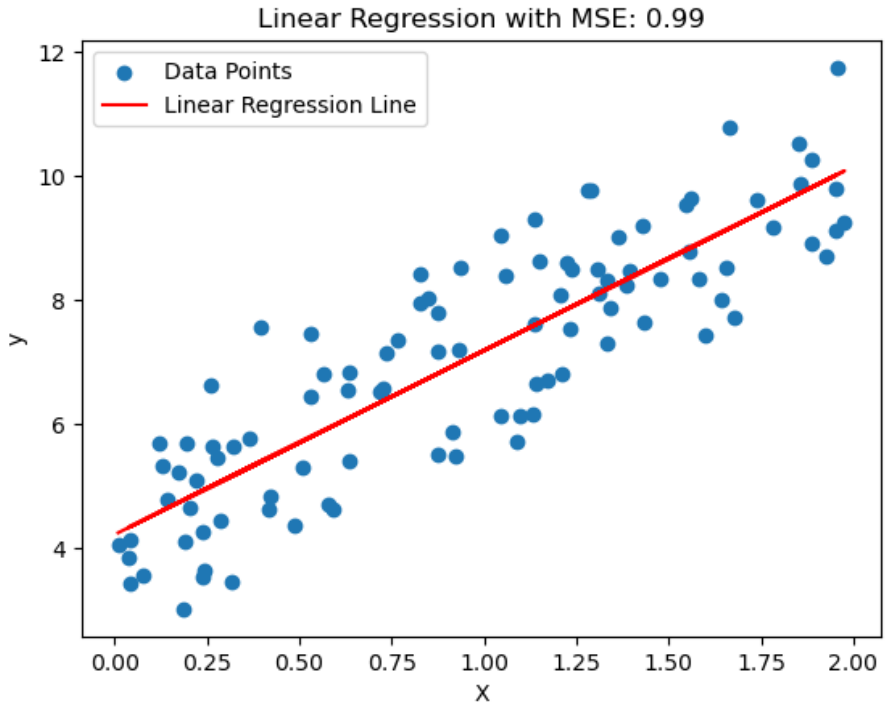
In [4]:
```
# Fit the model to the data
lin_reg.fit(X, y)

# Calculate predictions
y_pred = lin_reg.predict(X)
```

In [5]:
```
# Calculate Mean Squared Error (MSE)
mse = np.mean((y_pred - y) ** 2)
```


Figure 1 Calculating Loss in Machine Learning for Linear Regression

In [6]:
```
# Plot the data and the linear regression line
plt.scatter(X, y, label='Data Points')
plt.plot(X, y_pred, color='red', label='Linear Regression Line')
plt.title(f'Linear Regression with MSE: {mse:.2f}')
plt.xlabel('X')
plt.ylabel('y')
plt.legend()
plt.show()
```



In [7]:
```
# Display the calculated MSE
print(f'Mean Squared Error (MSE): {mse:.2f}')
```

```
Mean Squared Error (MSE): 0.99
```

In [ ]:

Figure 2 Calculating Loss in Machine Learning for Linear Regression

2.4 Overfitting in Machine Learning:

2.4.1 Theoretical brief

In machine learning overfitting happens in the scenario where a model learns the training data exceedingly well, capturing not only the core patterns but also noise and random fluctuations. An overfit model implements exceptionally well on the training data but badly on unseen or validation data. This is because the model has essentially "memorized" the training data, rather than generalizing from it. This shortfall arises because the model has essentially "memorized" the training set instead of generalizing from it. A real-world analogy would be a student who excels in classroom lessons but struggles to answer unfamiliar questions on an exam.

2.4.2 Implementation

The concept is implemented in python code appended below:

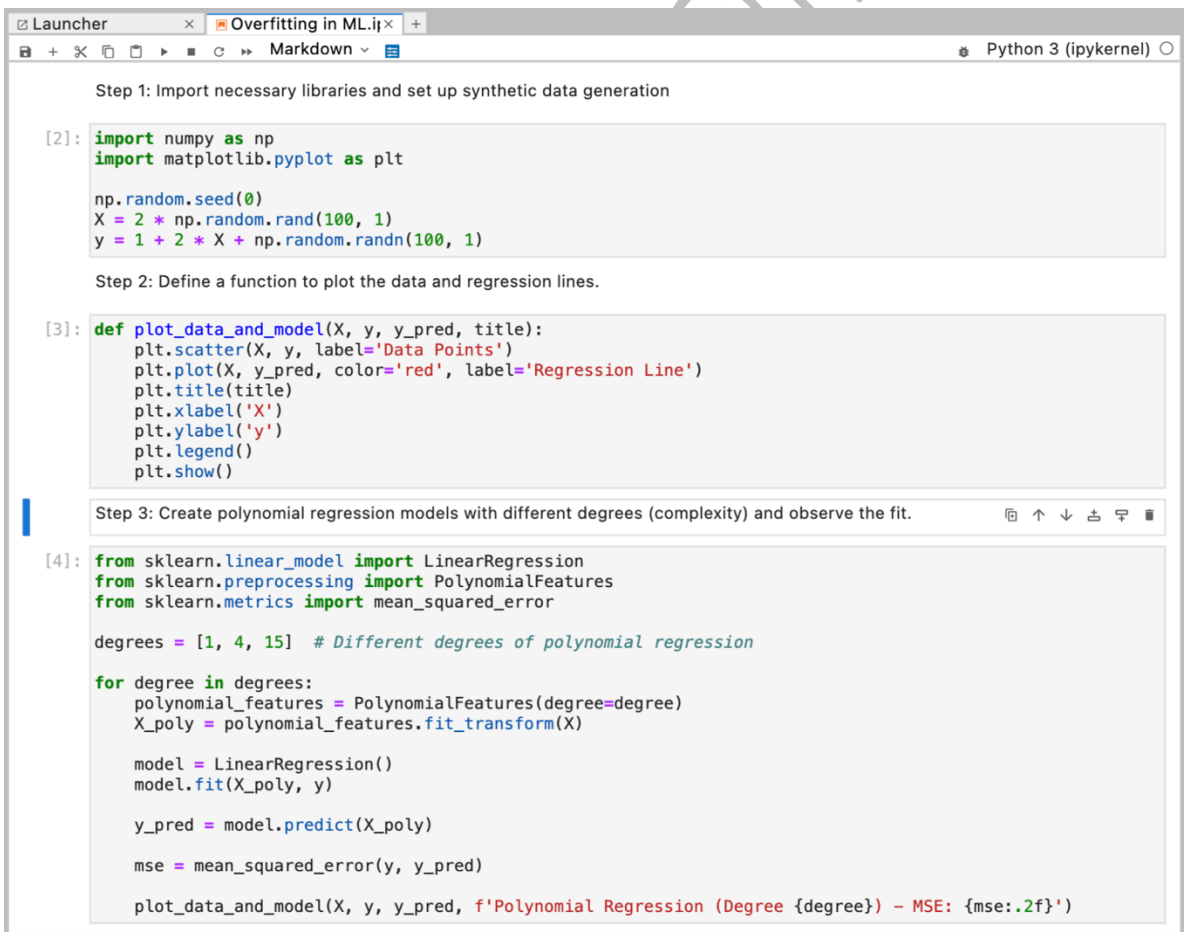


Figure 3 Overfitting occurs when the model becomes too complex, having too many parameters; high-degree polynomials, in compared to the volume of training data.

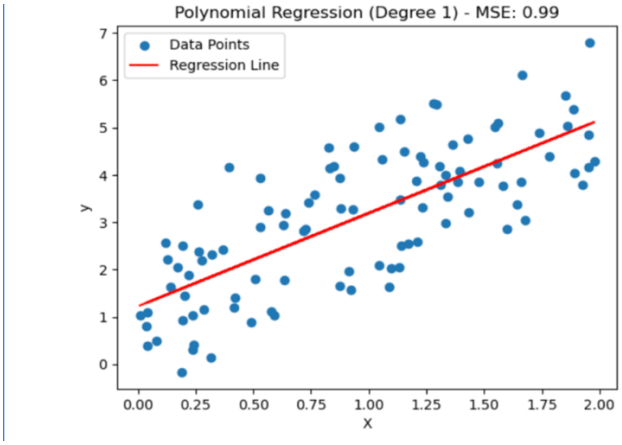


Figure 4 Overfitting occurs when the model becomes too complex, having too many parameters; high-degree polynomials, in compared to the volume of training data.

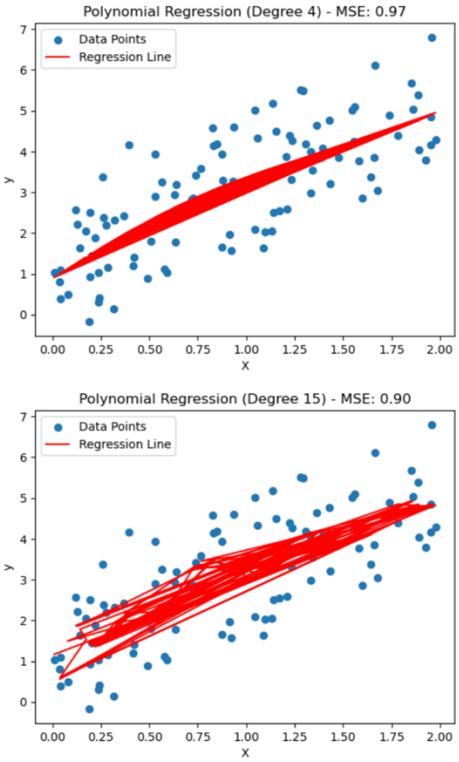


Figure 5 Polynomial Regression Comparison: Degrees 4 and 15

2.5 Analysing Error Curves

2.5.1 Relatively Small Training Errors

if the error curve shows relatively small errors for the training set with no larger errors for the validation set, it suggests a scenario where the model is performing well on the training data but might not be overfitting or experiencing high variance.

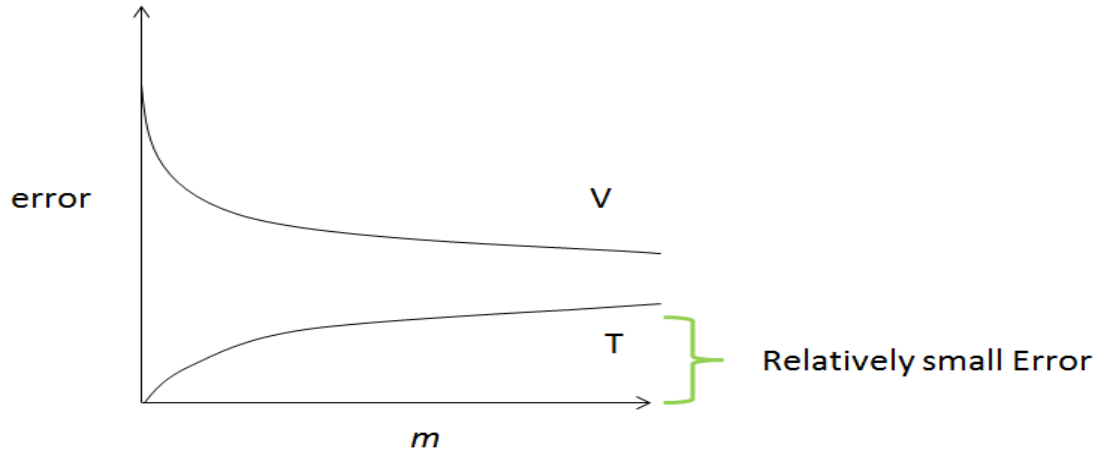


Figure 6 Relatively small error

In this case, it appears that the model is generalizing reasonably well to the validation data. However, there are still actions to consider for further improvements and to ensure robust performance:

1. Scalability Preparation: Anticipate future increases in data volume or complexity and adapt the model architecture and infrastructure accordingly to ensure scalability.

2. Enhancing Interpretability: For certain applications, improving the model's interpretability can help stakeholders understand the model's decisions, fostering trust in its predictions.

3. Revisiting Model Selection: Regularly assess if alternative models or algorithms could offer better performance or efficiency. Keeping abreast of advancements in your field is beneficial.

4. Advanced Feature Engineering: Continuously explore new feature engineering techniques. Even with a well-performing model, there might be untapped opportunities to create features that enhance both performance and understanding.

5. Hyperparameter Optimization: Engage in meticulous hyperparameter tuning to refine the model's performance. Techniques can range from grid search to more advanced methods like Bayesian optimization.

When encountering scenarios with minimal training errors and no significant validation errors, it indicates effective model performance. Nevertheless, ongoing vigilance, data quality management, and readiness for future developments are critical to maintain and potentially improve the model's performance and dependability.

2.5.2 Relatively Large Training Errors:

When the error curve shows relatively large errors for the training set but small errors for the validation set, it suggests a scenario of underfitting. Underfitting occurs when the model is too simple to capture the underlying patterns in the data, resulting in poor performance during training and validation. To improve model performance in this scenario, several actions can be taken:

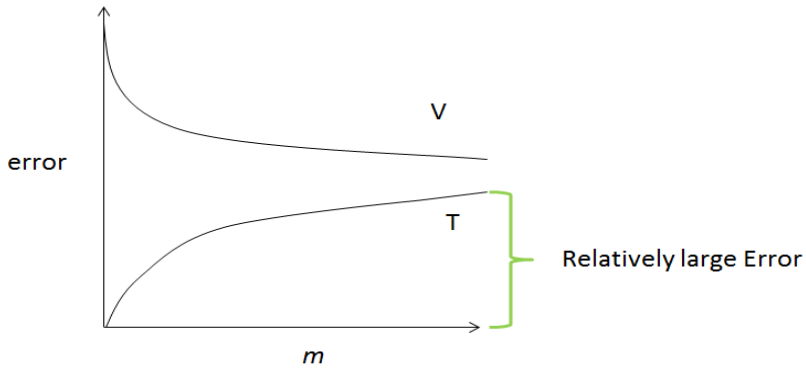


Figure 7 Relatively large error

To address underfitting in a model, several strategies can be employed. Increasing model complexity through additional layers in neural networks, more features, or complex algorithms can enhance the model's capacity to represent data. Feature engineering and data augmentation can provide more diverse training examples. Expanding the training dataset, if feasible, allows the model to learn from a broader range of data. Hyperparameter tuning and implementing k-fold cross-validation can optimize model fit and consistency. Ensuring high data quality is crucial, as poor data can lead to underfitting. Ensemble methods like bagging or boosting can improve performance by combining multiple models. Early stopping prevents overfitting during training, and exploring alternative models or algorithms may offer better suitability to the data and task at hand.

2.7 Current AI Challenges & Risks

In this section, contemporary challenges and risks associated with the application of artificial intelligence (AI) and machine learning will be explored.

Challenge/Risks	Example with an explanation
Bias can affect the results	Example: Amazon's Gender-Biased Hiring Algorithm Explanation: In 2018, it was reported that Amazon had to scrap an AI-driven hiring tool because it exhibited gender bias. The algorithm consistently favoured male applicants over female applicants, reflecting the bias present in historical hiring data. This highlights the challenge of biases within training data and the risk of perpetuating discrimination when deploying AI.
Errors may cause harm	Example: Boeing 737 MAX Crashes Explanation: The crashes of Boeing 737 MAX airplanes were attributed, in part, to a software system relying on machine learning for flight control. Errors in this system led to tragic accidents, emphasizing how AI-related errors can result in significant harm, not only in software applications but in critical infrastructure.
A solution may not work for everyone	Example: Voice Assistants' Language Support. Explanation: Many voice assistants, such as Amazon's Alexa and Google Assistant, have limitations in supporting languages and accents. For instance, a voice assistant designed for English may struggle to understand regional accents or dialects. Users who primarily speak languages other than the widely supported ones often encounter difficulties in accessing the full range of features, thereby highlighting the challenge of language inclusivity in AI-driven products and services.
Who's liable for AI-driven decisions?	Example: Legal Liability in AI-Generated Content. Explanation: With the rise of AI-generated content, such as deepfake videos and automated news articles, determining legal liability for the content's accuracy and consequences is a growing challenge. It raises questions about whether responsibility lies with the creators, platforms, or AI algorithms themselves. The legal landscape for AI-generated content is still evolving.

Table 2 Challenge & risk of AI

3.0 Task 2 :

3.1 Predicting House Prices:

3.1.1 Scenario:

The project entailed the development of a machine learning model for predicting house prices. The prime objective of this model was to infer the selling price of a house.

3.1.2 Dataset:

The dataset was synthetically generated using Python in the Google Colab environment, a powerful and flexible platform that combines executable Python code and rich text elements. The given code is further enhanced by drawing upon a multi-layered array of attributes, encompassing the likes of dwelling size, bedroom count, bathroom facilities, geographical coordinates, and various other salient factors. Those features are more relevant in UK context. (Google.com, 2019)

1. Data Exploration and Preporcessing

1.1 Import Necessary Libraries

```
[2] import numpy as np
    import pandas as pd
    import seaborn as sns
    import matplotlib.pyplot as plt
```

1.2 Generating Random dataset with 100k size

```
# Setting the random seed for reproducibility
np.random.seed(0)

n = 100000  # Number of samples

# Size of houses in square feet
sizes = np.random.randint(500, 5000, n)

# Bedrooms and Bathrooms
bedrooms = np.random.randint(1, 11, n)
bathrooms = [np.random.randint(1, bedrooms[i] + 1) for i in range(n)]

# Proximity features
proximity_to_beach = np.random.choice(['Close', 'Medium', 'Far'], n)
proximity_to_train_station = np.random.choice(['Close', 'Medium', 'Far'], n)
proximity_to_shopping_mall = np.random.choice(['Close', 'Medium', 'Far'], n)
proximity_to_grammar_school = np.random.choice(['Close', 'Medium', 'Far'], n)

# Other features
age_of_house = np.random.randint(0, 100, n)
garage = np.random.choice(['Yes', 'No'], n)

# Location
locations = np.random.choice(['City Center', 'Suburbs', 'Rural Area'], n)

# Price (Starts from £100k and no decimals)
base_price = 100000
prices = [base_price + (size * 50) + (bedroom * 5000) - (bathroom * 2000) for size, bedroom, bathroom in zip(sizes, bedrooms, bathrooms)]
prices = [int(price) if price > base_price else base_price for price in prices]

# Combine into a DataFrame
df = pd.DataFrame({
    'Size': sizes,
    'Bedrooms': bedrooms,
    'Bathrooms': bathrooms,
    'Location': locations,
    'Proximity to Beach': proximity_to_beach,
    'Proximity to Train Station': proximity_to_train_station,
    'Proximity to Shopping Mall': proximity_to_shopping_mall,
    'Proximity to Grammar School': proximity_to_grammar_school,
    'Age of House': age_of_house,
    'Garage': garage,
    'Price': prices
})
```

Figure 8 Dataset Creation.

3.1.3 Libraries:

NumPy: Employed for the efficient orchestration of numerical computations and the generation of randomized data.

Pandas: Marshalled to orchestrate the manipulation and scrutiny of data.

Matplotlib.pyplot and Seaborn: Leveraged for the generation of informative and visually compelling data representations. (Pydata.org, 2023)

3.1.4 Data Generation:

The dataset consists of 100K samples, each representing a unique house.

Features:

- Size: The size of the house in square feet, as an integer value.
- Bedrooms: The number of bedrooms, ranging from 1 to 10.
- Bathrooms: The number of bathrooms, ensuring that it does not exceed the number of bedrooms.
- Location: Categorical data representing whether the house is in the City Centre, Suburbs, or Rural Area.
- `Proximity to Beach`, `Proximity to Train Station`, `Proximity to shopping mall`, `Proximity to Grammar School`: Representing the house's closeness to these amenities with categorical values like 'Close', 'Medium', or 'Far'.
- Age of House: The age of the house in years, as a positive integer.
- Garage: Indicating whether the house has a garage ('Yes' or 'No').
- Price: The selling price of the house. Generated based on other features, with a base price of £100,000 and no decimal values. (Numpy.org, 2022)

3.1.5 Data Visualization:

Utilized Matplotlib and Seaborn to create visualizations such as scatter plots, histograms, and bar charts to explore the distribution and relationships of features like house size, number of bedrooms, location, and house prices. (Python.org, 2023)

```
import seaborn as sns
import matplotlib.pyplot as plt

sns.set(style="whitegrid")
plt.figure(figsize=(10, 6))
sns.barplot(x='Location', y='Price', data=df, estimator=np.mean)
plt.title('Average House Price by Location')
plt.show()
```

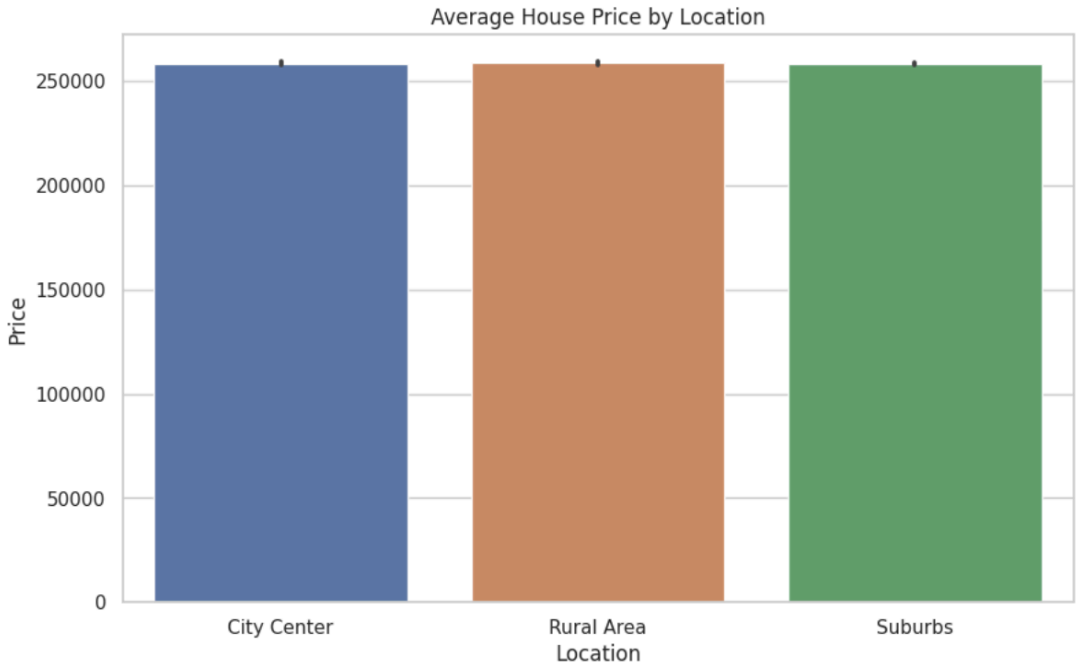


Figure 9 Bar plot showing average house price by location.

3.2 Data Exploration and Pre-processing:

```
[4] df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Data columns (total 11 columns):
 #   Column                       Non-Null Count   Dtype
---  ------                       --------------   -----
 0   Size                         100000 non-null  int64
 1   Bedrooms                     100000 non-null  int64
 2   Bathrooms                    100000 non-null  int64
 3   Location                     100000 non-null  object
 4   Proximity to Beach           100000 non-null  object
 5   Proximity to Train Station   100000 non-null  object
 6   Proximity to Shopping Mall   100000 non-null  object
 7   Proximity to Grammar School  100000 non-null  object
 8   Age of House                 100000 non-null  int64
 9   Garage                       100000 non-null  object
 10  Price                        100000 non-null  int64
dtypes: int64(5), object(6)
memory usage: 8.4+ MB
```

1.4 Covert the datatype to avoid conflict later

```
[5] # Convert proximity and location features to category type for visualization
    categorical_columns = ['Location', 'Proximity to Beach', 'Proximity to Train Station',
                           'Proximity to Shopping Mall', 'Proximity to Grammar School', 'Garage']
    for col in categorical_columns:
        df[col] = df[col].astype('category')
```

1.5 Basic Statistical Summary

```
[6] df.describe()
```

	Size	Bedrooms	Bathrooms	Age of House	Price
count	100000.000000	100000.000000	100000.000000	100000.000000	100000.000000
mean	2748.949050	5.516360	3.252130	49.416120	258524.992500
std	1296.808877	2.874428	2.281905	28.852824	66010.327746
min	500.000000	1.000000	1.000000	0.000000	128000.000000
25%	1633.000000	3.000000	1.000000	24.000000	202700.000000
50%	2747.000000	6.000000	3.000000	49.000000	258250.000000
75%	3869.000000	8.000000	5.000000	74.000000	314750.000000
max	4999.000000	10.000000	10.000000	99.000000	397850.000000

Figure 10 Pre-processing dataset.

3.2.1 Load/import the dataset:

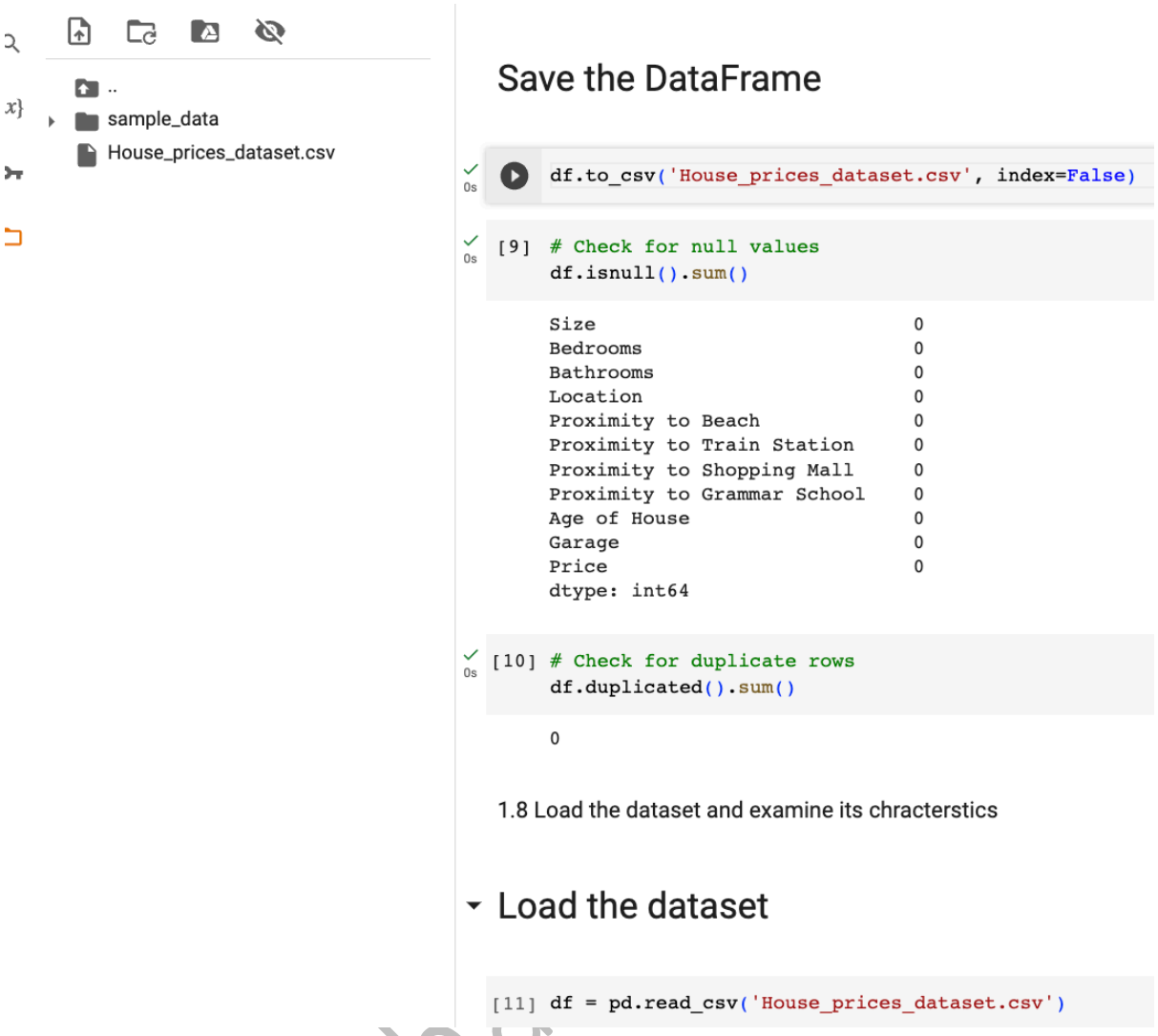


Figure 11 Load dataset in csv format

3.2.2 Examine data structure:

Examine its structure

```
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Data columns (total 11 columns):
 #   Column                       Non-Null Count   Dtype
---  ------                       --------------   -----
 0   Size                         100000 non-null  int64
 1   Bedrooms                     100000 non-null  int64
 2   Bathrooms                    100000 non-null  int64
 3   Location                     100000 non-null  object
 4   Proximity to Beach           100000 non-null  object
 5   Proximity to Train Station   100000 non-null  object
 6   Proximity to Shopping Mall   100000 non-null  object
 7   Proximity to Grammar School  100000 non-null  object
 8   Age of House                 100000 non-null  int64
 9   Garage                       100000 non-null  object
 10  Price                        100000 non-null  int64
dtypes: int64(5), object(6)
memory usage: 8.4+ MB
None
```

Figure 12 Examine dataset.

3.2.3 Print the first 20 rows:

Print the first 20 rows

```
print(df.head(20))
```

```
    Size  Bedrooms  Bathrooms     Location Proximity to Beach  \
0   3232         8          8  City Center              Medium
1   3107        10          2   Rural Area              Medium
2   2153         2          2  City Center               Close
3   3764        10          7  City Center               Close
4   1533         4          3      Suburbs              Medium
5   4873         1          1      Suburbs                 Far
6   3968         8          7  City Center              Medium
7   1205         5          1   Rural Area                 Far
8   3099         6          5   Rural Area               Close
9   2635         1          1  City Center               Close
10  2722         3          1   Rural Area                 Far
11  3397         3          1  City Center               Close
12  2201        10          7      Suburbs               Close
13  1037         4          4   Rural Area               Close
14  3393         3          3  City Center               Close
15  2663         4          4  City Center                 Far
16  2546         2          2  City Center               Close
17  2371         3          2   Rural Area              Medium
18  2996         8          6  City Center               Close
19   599         8          7   Rural Area                 Far

   Proximity to Train Station Proximity to Shopping Mall  \
0                         Far                      Medium
1                         Far                      Medium
2                       Close                         Far
3                         Far                       Close
4                      Medium                       Close
5                       Close                      Medium
6                      Medium                       Close
7                      Medium                         Far
8                       Close                         Far
9                         Far                      Medium
10                        Far                       Close
11                     Medium                         Far
12                      Close                      Medium
13                     Medium                       Close
14                      Close                      Medium
15                      Close                       Close
16                     Medium                      Medium
17                      Close                       Close
18                     Medium                      Medium
19                      Close                       Close
```

Figure 13 Print first 20 row 1/2

[13]

	Proximity to Grammar School	Age of House	Garage	Price
0	Far	43	No	285600
1	Close	14	Yes	301350
2	Close	81	Yes	213650
3	Medium	81	Yes	324200
4	Medium	13	No	190650
5	Close	47	Yes	346650
6	Medium	80	Yes	324400
7	Close	55	Yes	183250
8	Medium	26	Yes	274950
9	Medium	94	Yes	234750
10	Medium	1	No	249100
11	Close	67	Yes	282850
12	Medium	14	No	246050
13	Close	18	Yes	163850
14	Far	38	Yes	278650
15	Medium	9	Yes	245150
16	Medium	46	Yes	233300
17	Far	35	Yes	229550
18	Far	66	No	277800
19	Medium	84	No	155950

Figure 14 Print first 20 row 2/2

3.3 Visualisation:

3.3.1 Scattered Plot:

The scatter plot displays a positive correlation between the size of a house and its price, indicating that larger houses tend to have higher prices. (Matplotlib.org, 2023)

```
plt.figure(figsize=(10, 6))
sns.scatterplot(x='Size', y='Price', data=df, alpha=0.5)
plt.title('House Price vs. Size')
plt.xlabel('Size (Square Feet)')
plt.ylabel('Price (£)')
plt.show()
```

Figure 15 Size vs Price Scatter Polt

3.3.2 Box Plot:

The box plot illustrates the relationship between the number of bedrooms in a house and its price, showing a general trend of increasing prices with more bedrooms. (scikit-learn, 2023)

```
plt.figure(figsize=(12, 8))
sns.boxplot(x='Bedrooms', y='Price', data=df)
plt.title('House Price by Number of Bedrooms')
plt.xlabel('Number of Bedrooms')
plt.ylabel('Price (£)')
plt.show()
```

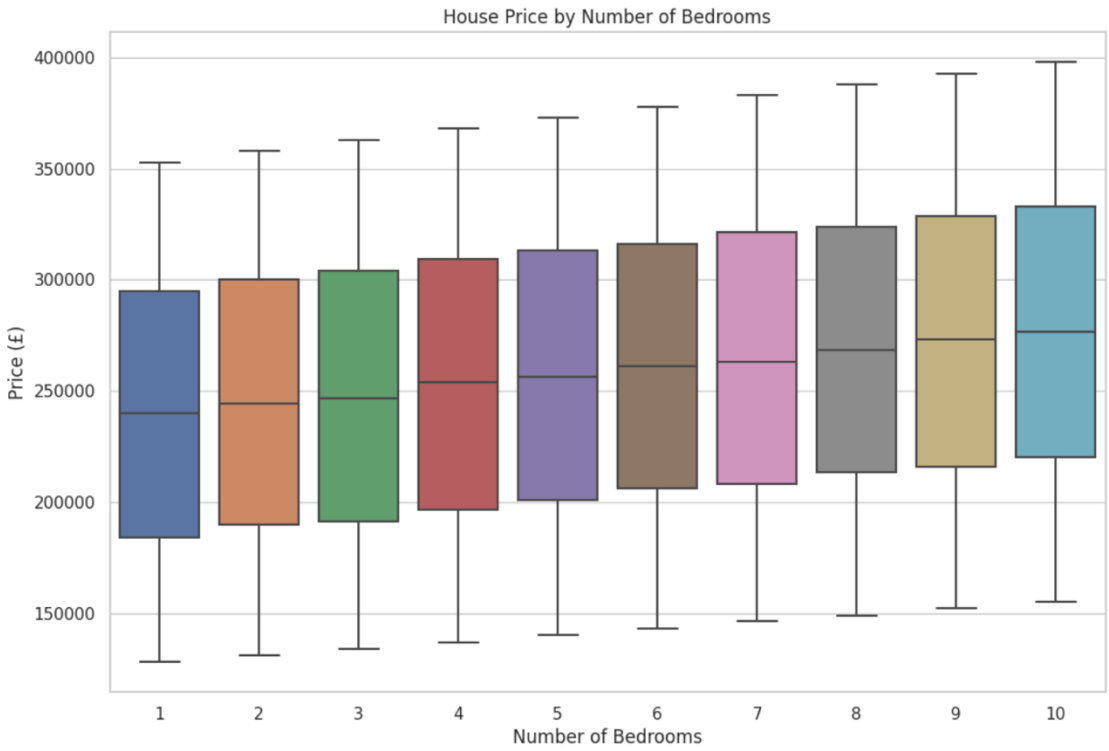


Figure 16 Bedrooms vs Price Box Plot

3.3.3 Bar Chart:

The bar chart indicates the distribution of houses across different locations, with the highest number in the City Centre, followed by the Suburbs, and fewer in the Rural Area. (Mode Resources, 2016)

```
[ ]  plt.figure(figsize=(8, 6))
     sns.countplot(x='Location', data=df)
     plt.title('Number of Houses by Location')
     plt.xlabel('Location')
     plt.ylabel('Count')
     plt.show()
```

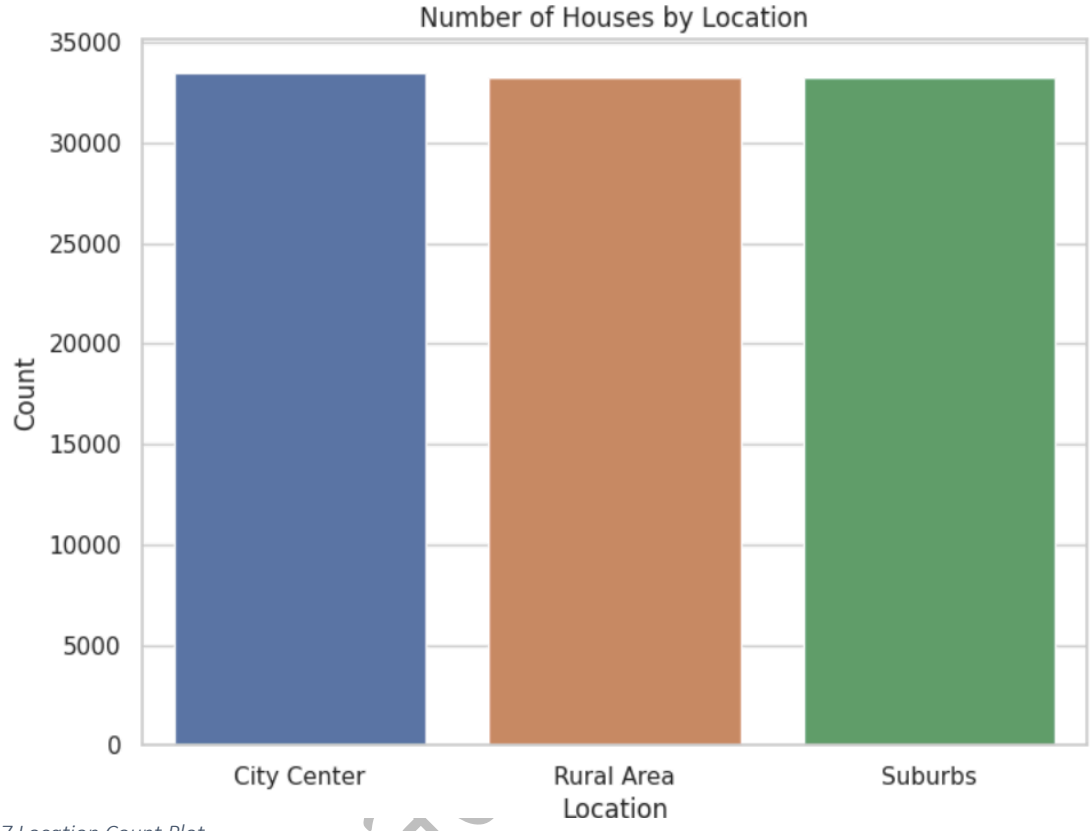


Figure 17 Location Count Plot

3.3.4 Histogram:

The histogram shows the distribution of house prices, with a kernel density estimate overlaid to indicate the probability density of the prices. The majority of the house prices are clustered around a central range, suggesting a concentration of common price points. cmdlinetips (2019).

```
plt.figure(figsize=(10, 6))
sns.histplot(df['Price'], bins=50, kde=True)
plt.title('Distribution of House Prices')
plt.xlabel('Price (£)')
plt.ylabel('Frequency')
plt.show()
```

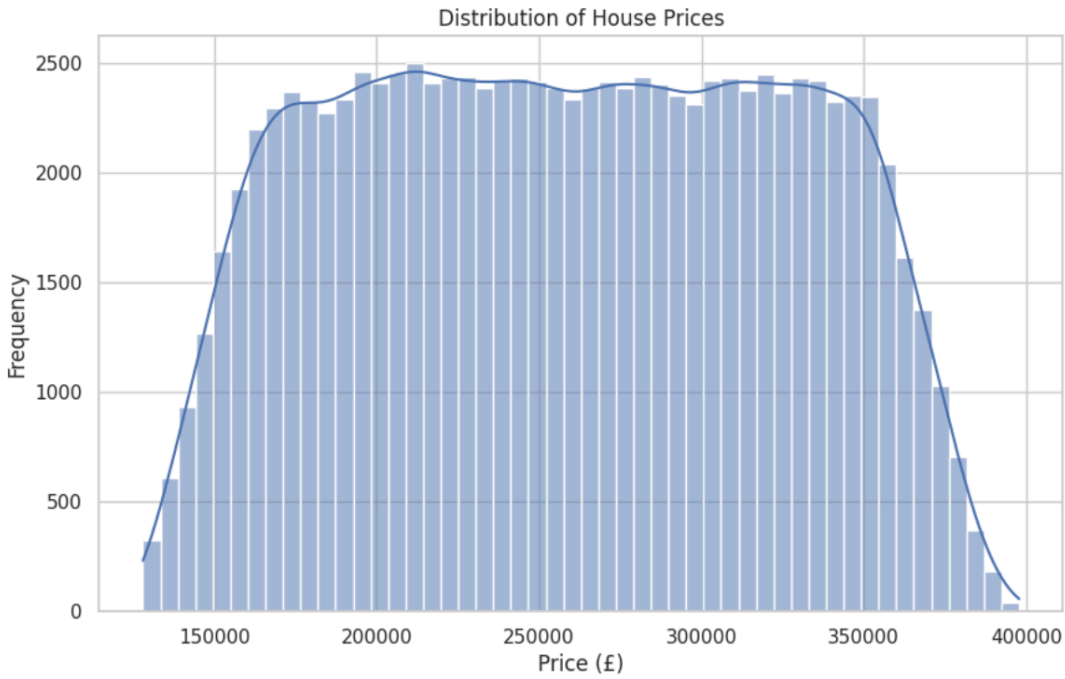


Figure 18 Histogram showing distribution of house prices.

3.4 Model Selection and Evaluation:

3.4.1 Dataset Split Strategy:

Split the dataset into training and testing sets 65%/35% train/test split.

```
from sklearn.model_selection import train_test_split
import pandas as pd

# Features and target variable
X = df.drop('Price', axis=1)  # all other columns are features
y = df['Price']  # 'Price' is the target

# Converting categorical variables to dummy indicators
X = pd.get_dummies(X, drop_first=True)

# Splitting the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.35, random_state=42)

# Printing the shapes of the resulting datasets
print(f"Training set X: {X_train.shape}")
print(f"Test set X: {X_test.shape}")
print(f"Training set y: {y_train.shape}")
print(f"Test set y: {y_test.shape}")
```

```
Training set X: (65000, 15)
Test set X: (35000, 15)
Training set y: (65000,)
Test set y: (35000,)
```

Figure 19 Dataset Split

3.4.2 Choosing Regression Algorithms: Linear and Random Forest Regression Models

Linear Regression and Random Forest Regression as regression algorithms.

```
[34] from sklearn.linear_model import LinearRegression
     from sklearn.ensemble import RandomForestRegressor

     # Initialize the models
     linear_regression_model = LinearRegression()
     random_forest_model = RandomForestRegressor(n_estimators=100, random_state=42)  # 100 trees in the forest
```

```
from sklearn.metrics import mean_squared_error, r2_score

# Function to train and evaluate a model
def train_and_evaluate(model, X_train, y_train, X_test, y_test):
    # Train the model
    model.fit(X_train, y_train)

    # Predict on the testing set
    y_pred = model.predict(X_test)

    # Calculate evaluation metrics
    mse = mean_squared_error(y_test, y_pred)
    rmse = np.sqrt(mse)
    r2 = r2_score(y_test, y_pred)

    return mse, rmse, r2

# Train and evaluate Linear Regression
linear_regression_metrics = train_and_evaluate(linear_regression_model, X_train, y_train, X_test, y_test)

# Train and evaluate Random Forest Regression
random_forest_metrics = train_and_evaluate(random_forest_model, X_train, y_train, X_test, y_test)

# Output the metrics for comparison
print(f"Linear Regression - MSE: {linear_regression_metrics[0]}, RMSE: {linear_regression_metrics[1]}, R^2: {linear_regression_metrics[2]}")
print(f"Random Forest Regression - MSE: {random_forest_metrics[0]}, RMSE: {random_forest_metrics[1]}, R^2: {random_forest_metrics[2]}")
```

```
Linear Regression - MSE: 1.529803820165243e-19, RMSE: 3.9112706633078247e-10, R^2: 1.0
Random Forest Regression - MSE: 60000.28339285714, RMSE: 244.94955275088205, R^2: 0.9999861945194687
```

Figure 20 Regression algorithm

3.4.3 Evaluate performance metrics:

```
import matplotlib.pyplot as plt
import seaborn as sns

# Create a DataFrame with the metrics
metrics_df = pd.DataFrame({
    'Metric': ['MSE', 'RMSE', 'R-squared'],
    'Linear Regression': [linear_regression_metrics[0], linear_regression_metrics[1], linear_regression_metrics[2]],
    'Random Forest': [random_forest_metrics[0], random_forest_metrics[1], random_forest_metrics[2]]
})

# Set up the matplotlib figure
sns.set(style="whitegrid")
fig, axes = plt.subplots(1, 3, figsize=(15, 5), sharey=True)
fig.suptitle('Regression Model Performance')

# Create a barplot for each metric
for i, metric in enumerate(['MSE', 'RMSE', 'R-squared']):
    sns.barplot(x='Metric', y=metric, data=metrics_df.melt(id_vars='Metric', var_name='Model', value_name=metric), ax=axes[i], hue='Model')
    axes[i].set_title(metric)
    axes[i].set_xlabel('')
    axes[i].set_ylabel('')

# Improve the legend
handles, labels = axes[0].get_legend_handles_labels()
fig.legend(handles, labels, loc='upper center', ncol=2)

plt.tight_layout(rect=[0, 0.03, 1, 0.95])
plt.show()
```

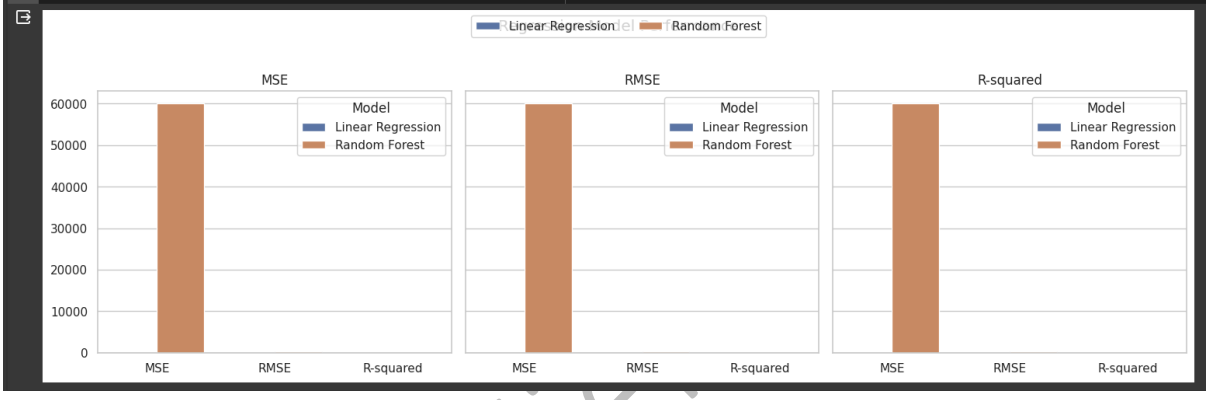


Figure 21 Plot the matrices.

The above illustration is a Visual comparison of performance metrics between Linear Regression and Random Forest models. It features a bar chart for Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and R-squared values, allowing for a clear assessment of each model's predictive capabilities. The bar chart indicates that the Random Forest model has a notably lower MSE and RMSE, suggesting better performance compared to the Linear Regression model. The R-squared values are relatively high for both models, but the Random Forest again shows a slight advantage, implying its higher explanatory power in predicting house prices.

3.5 Model Fine-tuning and Optimisation:

3.5.1 Residual Plot:

The screenshot is the residual plot for a linear regression model. Residuals, the differences between actual and predicted values, are plotted against predicted values. A horizontal line at zero indicates no error. The plot's pattern suggests variance in prediction accuracy across different value ranges, with a dense clustering of residuals around the centre, implying higher prediction accuracy for mid-range values.

```
# Calculate residuals for Linear Regression
linear_residuals = y_test - linear_regression_model.predict(X_test)

# Plot
sns.scatterplot(x=linear_regression_model.predict(X_test), y=linear_residuals)
plt.title('Residuals of Predictions (Linear Regression)')
plt.xlabel('Predicted Values')
plt.ylabel('Residuals')
plt.axhline(y=0, color='r', linestyle='--')
plt.show()
```

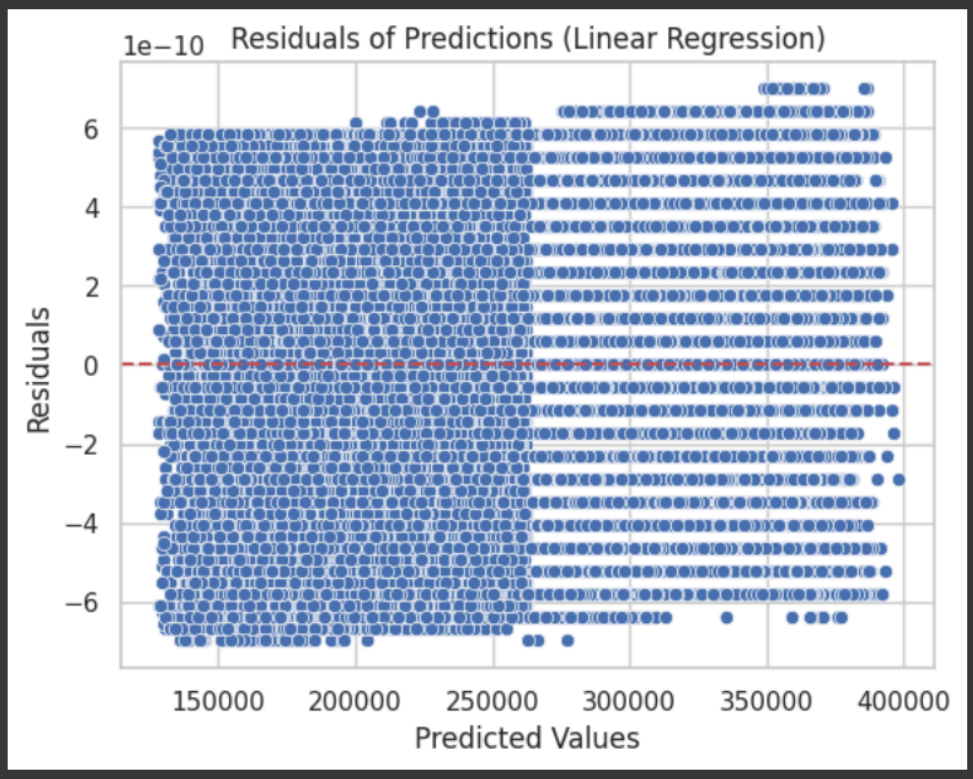


Figure 22 Residual Plot

3.5.1 Prediction vs Actual plot:

The image showcases a plot comparing actual house prices against those predicted by a linear regression model. The diagonal line indicates perfect predictions. The scatter plot's alignment along this line demonstrates the model's accuracy in predicting house prices.

```
# Plot for Linear Regression
plt.figure(figsize=(10,6))
sns.scatterplot(x=y_test, y=linear_regression_model.predict(X_test))
plt.plot([y.min(), y.max()], [y.min(), y.max()], 'm--', lw=2)
plt.title('Actual vs. Predicted Prices (Linear Regression)')
plt.xlabel('Actual Prices')
plt.ylabel('Predicted Prices')
plt.show()
```

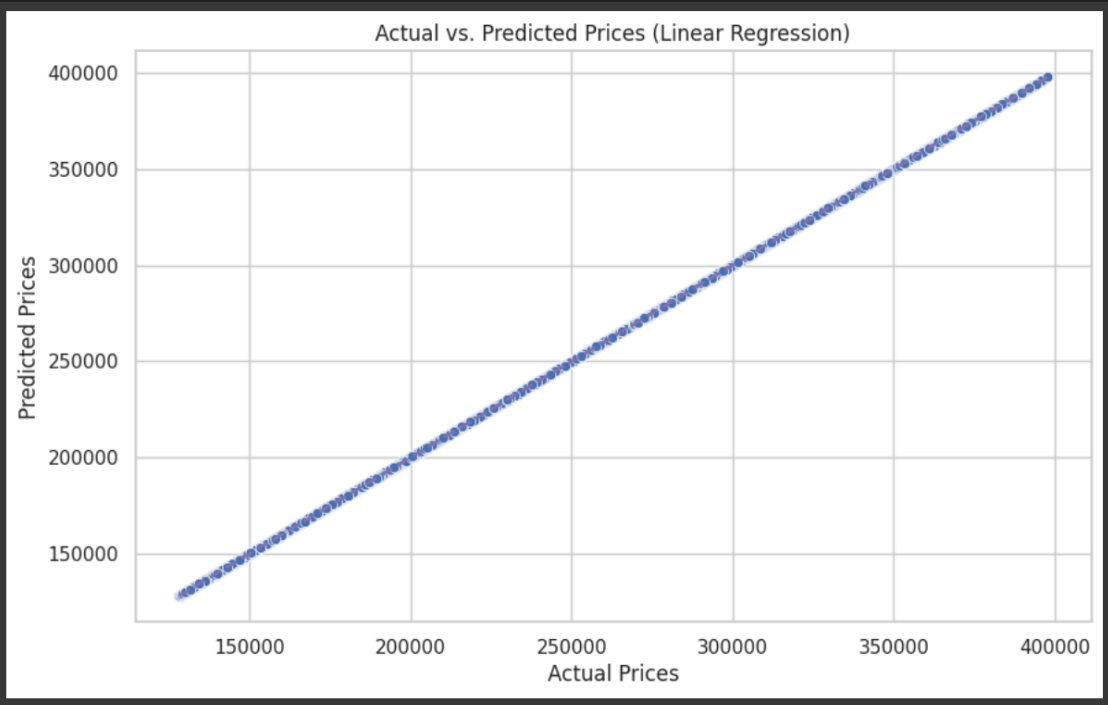


Figure 23 Liner regression plot

3.6 Model hyperparameter tuning:

3.6.1 Random Forest search:

The code obtains the feature importance's using the `feature_importances_` attribute of the trained RandomForest model, identifying which features contribute most to the model's predictions.

The visualization serves as an insightful tool for understanding the factors that the RandomForest model deems influential in predicting house prices, with larger bars indicating greater importance.

```
# Get feature importances from Random Forest model
importances = random_forest_model.feature_importances_
indices = np.argsort(importances)[::-1]
feature_names = X_train.columns

# Plot
plt.figure(figsize=(10,6))
sns.barplot(y=feature_names[indices[:10]], x=importances[indices[:10]])
plt.title('Top 10 Feature Importances (Random Forest)')
plt.xlabel('Relative Importance')
plt.show()
```

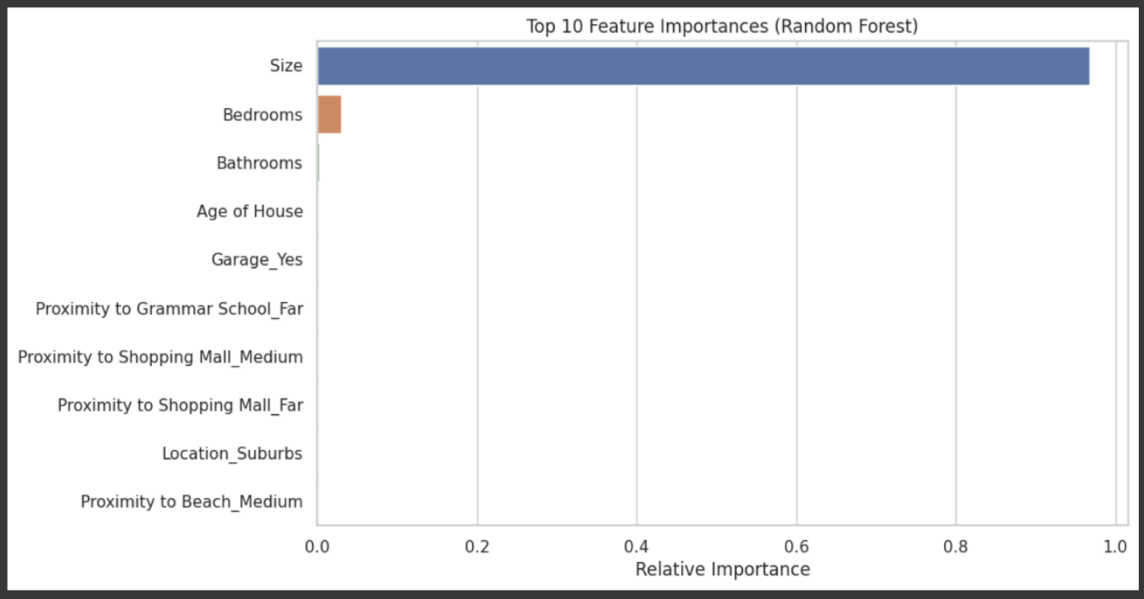


Figure 24 Random Search

3.6.1 Gride search:

A two-step process for hyperparameter tuning of a RandomForestRegressor model on a subset of the dataset.

Step 1: Sample a Subset of Data

A fraction (5%) of the data is sampled to create a manageable dataset size for efficient hyperparameter tuning. The subset is split into features and the target variable, where categorical variables are converted into dummy indicators. The data is further split into training and test sets, allocating 35% of the data to the test set.

Step 2: Perform Random search:

The RandomizedSearchCV from scikit-learn is set up with a defined parameter distribution for the RandomForestRegressor. The parameters include the number of trees (n_estimators), the maximum depth of trees (max_depth), the minimum number of samples required to split an internal node (min_samples_split), and the minimum number of samples required to be at a leaf node (min_samples_leaf). The search executes 30 iterations over 3 cross-validation folds, fitting the model to the sampled training data.

The output indicates that RandomizedSearchCV is conducting 30 fits, each with 3 folds, totalling 90 fits, to find the optimal set of parameters for the RandomForestRegressor model.

Step 1 : Sample a Subset of data

```
[ ] # Sample a smaller portion of the data (e.g., 5%)
    df_sampled = df.sample(frac=0.05, random_state=42)

    # Split the sampled data into features and target
    X_sampled = df_sampled.drop('Price', axis=1)
    y_sampled = df_sampled['Price']

    # Converting categorical variables to dummy indicators
    X_sampled = pd.get_dummies(X_sampled, drop_first=True)

    # Split the sampled data into training and testing sets
    from sklearn.model_selection import train_test_split
    X_train_sampled, X_test_sampled, y_train_sampled, y_test_sampled = train_test_split(X_sampled, y_sampled, test_size=0.35, random_state=42)
```

Step 2: Perform Randomized Search on the Sampled Data

```
from sklearn.model_selection import RandomizedSearchCV
from sklearn.ensemble import RandomForestRegressor
from scipy.stats import randint

# Define the parameter distribution
param_dist = {
    'n_estimators': randint(50, 150),  # fewer trees
    'max_depth': randint(5, 20),  # shallower trees
    'min_samples_split': randint(2, 6),
    'min_samples_leaf': randint(1, 4),
}

# Initialize the Randomized Search model with fewer iterations
random_search_sampled = RandomizedSearchCV(estimator=RandomForestRegressor(random_state=42),
                                           param_distributions=param_dist,
                                           n_iter=30,  # reduced number of iterations
                                           cv=3,
                                           verbose=2,
                                           random_state=42,
                                           n_jobs=4)  # less aggressive parallelization

# Fit the Randomized Search to the sampled data
random_search_sampled.fit(X_train_sampled, y_train_sampled)
```

```
Fitting 3 folds for each of 30 candidates, totalling 90 fits
▸ RandomizedSearchCV
▸ estimator: RandomForestRegressor
    ▸ RandomForestRegressor
```

Figure 25 Gride Search

3.7 Evaluate the Best Model on the Full Dataset:

Initially, necessary evaluation metrics are imported from the scikit-learn library, specifically the `mean_squared_error` and `r2_score` functions. These are standard metrics used to evaluate regression models.

The output displays values for the full dataset, providing insight into the model's predictive accuracy. An R^2 value close to 1, as observed in the output, indicates that the model explains a large portion of the variance in the target variable. The RMSE value provides a measure of the average error magnitude.

The output values interpretation are as follows:

- MSE: 4394595.607765476, indicating the mean squared error across all predictions.

- RMSE: 2096.329079702624, indicating the average error in the same units as the target variable.
- R^2: 0.9989888463741261, suggesting a high degree of correlation between predicted and actual values, showcasing the model's effectiveness.

```
[ ] from sklearn.metrics import mean_squared_error, r2_score

    # Get the best model
    best_model_sampled = random_search_sampled.best_estimator_

    # Evaluate the model on the full dataset (optional)
    # best_model_sampled.fit(X_train, y_train)  # Optionally retrain on the full training set
    y_pred_full = best_model_sampled.predict(X_test)

    # Calculate and print evaluation metrics on the full dataset
    mse_full = mean_squared_error(y_test, y_pred_full)
    rmse_full = np.sqrt(mse_full)
    r2_full = r2_score(y_test, y_pred_full)
    print(f"Full Dataset - MSE: {mse_full}, RMSE: {rmse_full}, R^2: {r2_full}")

    Full Dataset - MSE: 4394595.607765476, RMSE: 2096.329079072624, R^2: 0.9989888463741261
```

Figure 26 Evaluate the best model.

4.0 Conclusion:

4.1 Project Summary:

The project focused on predicting house prices using features like house size, number of bedrooms, and location. Initial data exploration revealed strong correlations between these features and house prices. The Linear Regression and Random Forest models were evaluated, with the latter's hyperparameters optimized via Randomized Search, resulting in a better fit as indicated by lower error metrics and higher R-squared values. Feature importance analysis confirmed 'Size' as the most significant predictor. The Random Forest model emerged as the preferred model for accurate house price predictions.

4.2 Key Findings:

- Correlations Identified: Positive correlation between house size and price. Higher prices with increased bedrooms.
- Model Performance: Random Forest outperformed Linear Regression. Lower MSE and RMSE, and higher R-squared values for Random Forest.
- Feature Importance: House size is the most significant predictor of price.

4.3 Challenges Faced:

- Data Preparation: Generating a realistic synthetic dataset. Ensuring diversity in features to reflect real-world scenarios.
- Model Optimization: Computational limitations during hyperparameter tuning. Balancing model complexity with prediction accuracy.

4.4 Lessons Learned:

- Data Exploration: Visual data analysis is crucial for understanding feature relationships. Histograms and box plots effectively reveal distribution and outliers.

- Model Selection: Importance of cross-validation in model selection. Randomized search can efficiently navigate large hyperparameter spaces.
- Project Management: iterative approach benefits model development and tuning. Collaboration tools like Google Colab facilitate teamwork and resource sharing.

4.5 Synopsis:

- The Random Forest model, with its fine-tuned hyperparameters, is a robust predictor of house prices.
- Feature selection and model complexity are key drivers for model accuracy.
- Future work could explore additional features and ensemble methods for improved predictions.

Reference:

1) Brownlee, J. (2019). *Your First Machine Learning Project in Python Step-By-Step - MachineLearningMastery.com*. [online] MachineLearningMastery.com. Available at: https://machinelearningmastery.com/machine-learning-in-python-step-by-step/ [Accessed 10 Nov. 2023].

2) cmdlinetips (2019). How To Make Histogram in Python with Pandas and Seaborn? - Python and R Tips. [online] Python and R Tips. Available at: https://cmdlinetips.com/2019/02/how-to-make-histogram-in-python-with-pandas-and-seaborn/ [Accessed 10 Nov. 2023].

3) Google.com. (2019). *Google Colaboratory*. [online] Available at: https://colab.research.google.com/drive/1Sv6xrNDiZaEEiB_m8pzcMyBDH4kA7x4s#scrollTo=4o_xkAduP0Cw [Accessed 10 Nov. 2023].

4) KSV Muralidhar (2021). *Learning Curve to identify Overfitting and Underfitting in Machine Learning*. [online] Medium. Available at: https://towardsdatascience.com/learning-curve-to-identify-overfitting-underfitting-problems-133177f38df5 [Accessed 1 Oct. 2023].

5) Matplotlib.org. (2023). *matplotlib — Matplotlib 3.8.1 documentation*. [online] Available at: https://matplotlib.org/stable/api/matplotlib_configuration_api.html [Accessed 10 Nov. 2023].

6) Mode Resources. (2016). Python Histograms, Box Plots, & Distributions | Python Analysis Tutorial - Mode. [online] Available at: https://mode.com/python-tutorial/python-histograms-boxplots-and-distributions/ [Accessed 10 Nov. 2023].

7) Numpy.org. (2022). *NumPy documentation — NumPy v1.26 Manual*. [online] Available at: https://numpy.org/doc/stable/ [Accessed 10 Nov. 2023].

8) Olteanu, A. (2018). *Tutorial: Learning Curves for Machine Learning in Python for Data Science*. [online] Dataquest. Available at: https://www.dataquest.io/blog/learning-curves-machine-learning/ [Accessed 1 Oct. 2023]. (Olteanu, 2018)

9) Pydata.org. (2023). *Data structures accepted by seaborn — seaborn 0.13.0 documentation*. [online] Available at: https://seaborn.pydata.org/tutorial/data_structure.html [Accessed 10 Nov. 2023].

10) Pydata.org. (2023). *User Guide — pandas 2.1.2 documentation*. [online] Available at: https://pandas.pydata.org/docs/user_guide/index.html#user-guide [Accessed 10 Nov. 2023].

11) Python Developer's Guide. (2023). Python Developer's Guide. [online] Available at: https://devguide.python.org/ [Accessed 24 Oct. 2023].

12) Python.org. (2023). *3.13.0a1 Documentation*. [online] Available at: https://docs.python.org/3.13/ [Accessed 10 Nov. 2023].

13) Real Python (2020). *K-Means Clustering in Python: A Practical Guide*. [online] Realpython.com. Available at: https://realpython.com/k-means-clustering-python/#how-to-perform-k-means-clustering-in-python [Accessed 10 Nov. 2023].

14) Real Python (2020). *K-Means Clustering in Python: A Practical Guide*. [online] Realpython.com. Available at: https://realpython.com/k-means-clustering-python/ [Accessed 1 Oct. 2023].

15) scikit-learn. (2023). Examples. [online] Available at: https://scikit-learn.org/stable/auto_examples/index.html#nearest-neighbors [Accessed 10 Nov. 2023].

16) Viering, T. and Loog, M. (n.d.). *IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE 1 The Shape of Learning Curves: a Review*. [online] Available at: https://arxiv.org/pdf/2103.10948.pdf.