```
# Importing dependencies

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from xgboost import XGBRegressor

from sklearn.metrics import mean_squared_error as mse

# Importing dataset
housing = pd.read_csv('housing.csv')
```

Cleaning and Exploratory Data Analysis

```
housing.head()
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	hou
0	-122.23	37.88	41.0	880.0	129.0	322.0	
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0	
2	-122.24	37.85	52.0	1467.0	190.0	496.0	
3	-122.25	37.85	52.0	1274.0	235.0	558.0	
4	-122.25	37.85	52.0	1627.0	280.0	565.0	

Next steps: Generate code with housing | View recommended plots | New interactive sheet

```
housing.shape
```

```
(20640, 10)
```

```
housing.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 10 columns):
 #   Column              Non-Null Count  Dtype
```

```
---  ------              --------------  -----
 0   longitude           20640 non-null  float64
 1   latitude            20640 non-null  float64
 2   housing_median_age  20640 non-null  float64
 3   total_rooms         20640 non-null  float64
 4   total_bedrooms      20433 non-null  float64
 5   population          20640 non-null  float64
 6   households          20640 non-null  float64
 7   median_income       20640 non-null  float64
 8   median_house_value  20640 non-null  float64
 9   ocean_proximity     20640 non-null  object
dtypes: float64(9), object(1)
memory usage: 1.6+ MB
```

```
# Checking for null values

housing.isnull().sum()
```

	0
longitude	0
latitude	0
housing_median_age	0
total_rooms	0
total_bedrooms	207
population	0
households	0
median_income	0
median_house_value	0
ocean_proximity	0

dtype: int64

```
# Checking for duplicate values

housing.duplicated().sum()
```

```
0
```

We can observe that:

- The dataset contains 20,640 samples and 8 features;
- All features are numerical features encoded as floating number except **ocean_proximity**.

- There are missing values in the **total_bedrooms** feature.
- There are no duplicate rows.

Data Analysis and Visualisations

```
# Analysing the values of the ocean_proximity feature

print(housing['ocean_proximity'].value_counts())
```

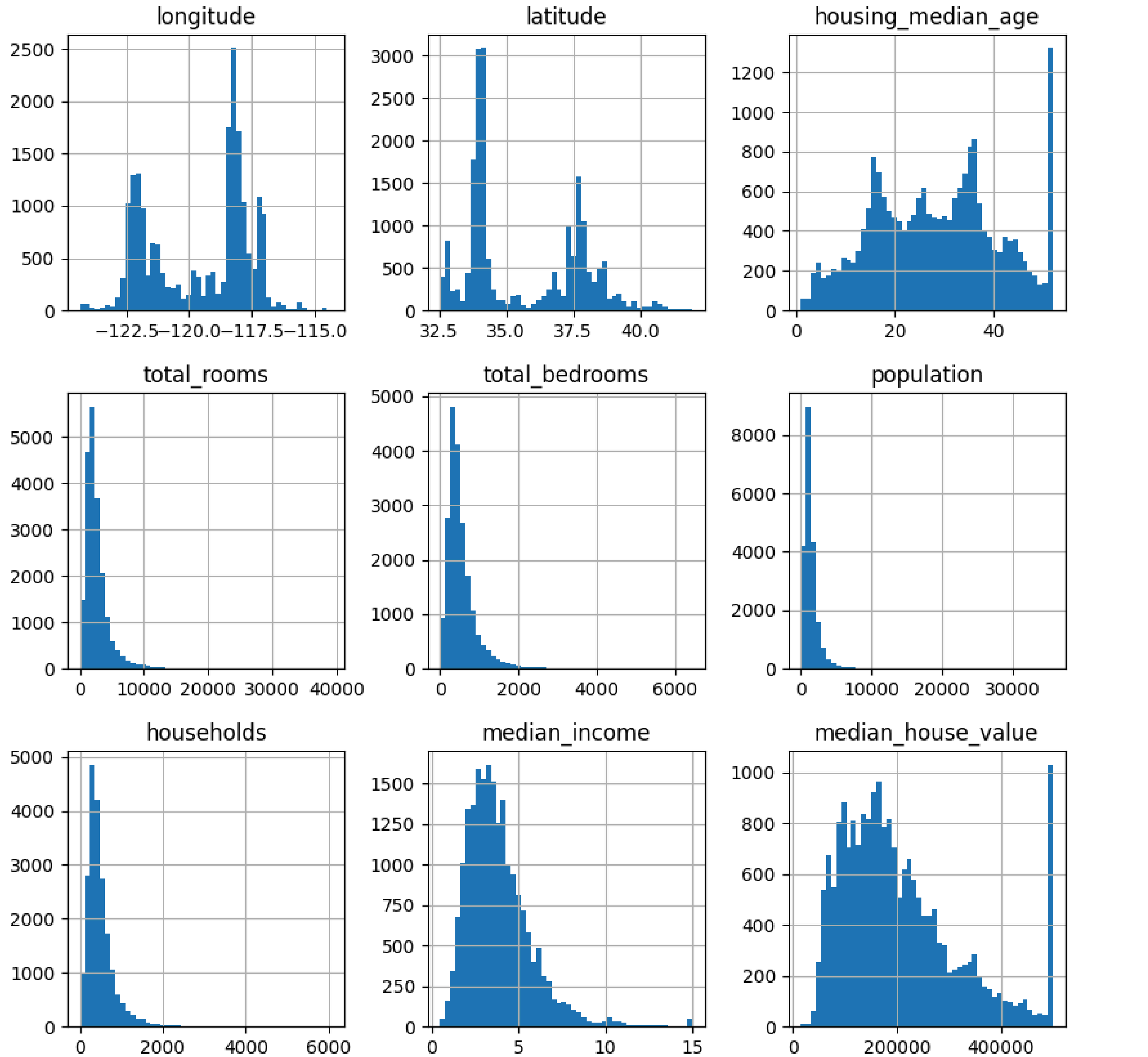
```
ocean_proximity
<1H OCEAN     9136
INLAND        6551
NEAR OCEAN    2658
NEAR BAY      2290
ISLAND           5
Name: count, dtype: int64
```

```
housing.describe()
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	po
count	20640.000000	20640.000000	20640.000000	20640.000000	20433.000000	2064
mean	-119.569704	35.631861	28.639486	2635.763081	537.870553	142
std	2.003532	2.135952	12.585558	2181.615252	421.385070	113
min	-124.350000	32.540000	1.000000	2.000000	1.000000	
25%	-121.800000	33.930000	18.000000	1447.750000	296.000000	78
50%	-118.490000	34.260000	29.000000	2127.000000	435.000000	116
75%	-118.010000	37.710000	37.000000	3148.000000	647.000000	172
max	-114.310000	41.950000	52.000000	39320.000000	6445.000000	3568

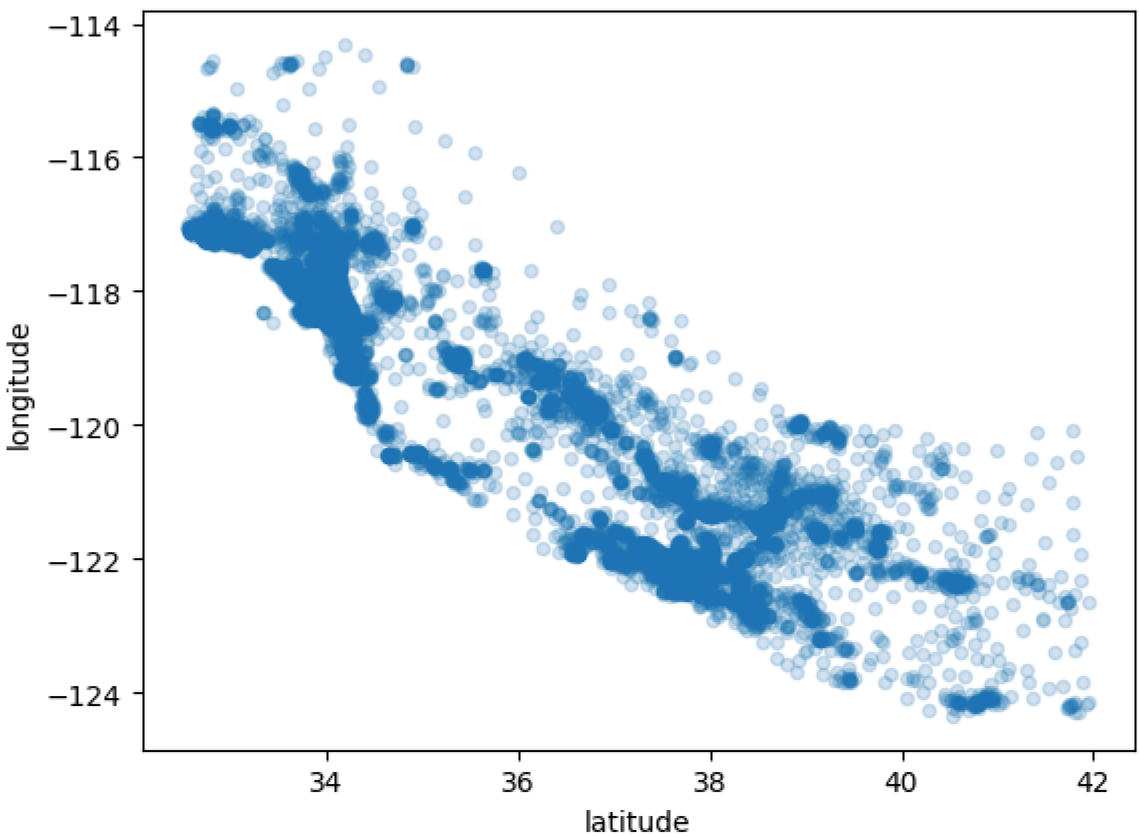
```
# Creating a histogram to understand the distribution

housing.hist(bins = 50, figsize = (10, 10)) # Bins - Number of chucks we split the dat
```

```
array([[<Axes: title={'center': 'longitude'}>,
        <Axes: title={'center': 'latitude'}>,
        <Axes: title={'center': 'housing_median_age'}>],
       [<Axes: title={'center': 'total_rooms'}>,
        <Axes: title={'center': 'total_bedrooms'}>,
        <Axes: title={'center': 'population'}>],
       [<Axes: title={'center': 'households'}>,
        <Axes: title={'center': 'median_income'}>,
        <Axes: title={'center': 'median_house_value'}>]], dtype=object)
```



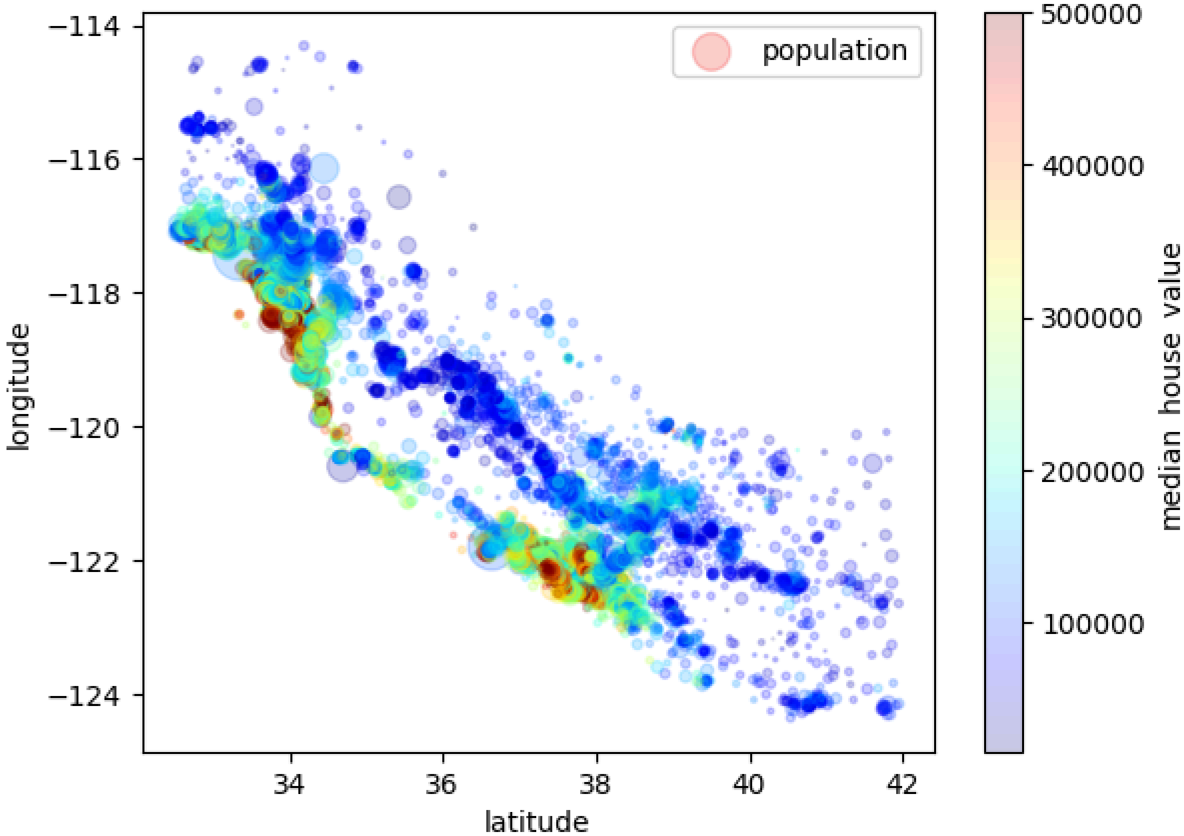
```
housing.plot(kind = 'scatter', x = 'latitude', y = 'longitude', alpha = 0.2)
# alpha adjusts the transparency of the dots such they are obervable

'''
Insights:
We can observe that the graph looks like the map of california.
The dense areas are where we have more observations (points are overlaid on eachother)
'''
```

```
'\nInsights: \nWe can observe that the graph looks like the map of california.\nThe den
se areas are where we have more observations (points are overlaid on eachother).\n'
```




```
# Plotting a scatter plot the location with the population and median house price
housing.plot(kind = 'scatter',
             x = 'latitude', y = 'longitude',
             alpha = 0.2,
             s = housing['population'] / 100, # Size of the points is proportional to
             label = 'population',
             c = 'median_house_value', # Color of each point based on the price
             cmap = plt.get_cmap('jet'), # Color map used to plot the points
             colorbar = True) # Scale to interpret the colours

# Observation:
# As we go inland the median value of the houses decreases as compared to the houses r
```

```
<Axes: xlabel='latitude', ylabel='longitude'>
```




```
# Detecting relationship between different features using a scatter plot

# Correlation matrix - shows how closely realated two variables are.
numerical_features = housing.select_dtypes(include = np.number)
correlation_matrix = numerical_features.corr()


# Observing only the price with all the other features
correlation_matrix['median_house_value'].sort_values(ascending = False)
```

	median_house_value
median_house_value	1.000000
median_income	0.688075
total_rooms	0.134153
housing_median_age	0.105623
households	0.065843
total_bedrooms	0.049686
population	-0.024650
longitude	-0.045967
latitude	-0.144160

dtype: float64

- The median_income is positively correlated with the median price, meaning, higher the income, the more expensive the house.
- The latitude is negatively correlated with the median_price, meaning, the houses up north are cheaper compared to the ones in southern california.

```
# Dealing with the missing values in the bedrooms

# Dropping the rows with missing values
df = housing.dropna(subset = ['total_bedrooms'])
df.shape
```

```
(20433, 10)
```

```
# Handing the categorical variable using get_dummies

dummies = pd.get_dummies(df['ocean_proximity']).astype(int)
dummies
```

	<1H OCEAN	INLAND	ISLAND	NEAR BAY	NEAR OCEAN
0	0	0	0	1	0
1	0	0	0	1	0
2	0	0	0	1	0
3	0	0	0	1	0
4	0	0	0	1	0
...	...	...	...	...	...
20635	0	1	0	0	0
20636	0	1	0	0	0
20637	0	1	0	0	0
20638	0	1	0	0	0
20639	0	1	0	0	0

20433 rows × 5 columns

Next steps: Generate code with dummies | View recommended plots | New interactive sheet

```
df_processed = pd.concat([df, dummies], axis = 'columns')
df_processed.head()
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	hou
0	-122.23	37.88	41.0	880.0	129.0	322.0	
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0	
2	-122.24	37.85	52.0	1467.0	190.0	496.0	
3	-122.25	37.85	52.0	1274.0	235.0	558.0	
4	-122.25	37.85	52.0	1627.0	280.0	565.0	

Next steps: Generate code with df_processed | View recommended plots | New interactive sheet

```
# Dropping the ocean_proximity and one of the variables in dummies (to avoid multicoll

# Dropping island beacause we have only 5 samples
```

```
housing_data = df_processed.drop(['ocean_proximity', 'ISLAND'], axis = 1)
housing_data.head()
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	hou
0	-122.23	37.88	41.0	880.0	129.0	322.0	
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0	
2	-122.24	37.85	52.0	1467.0	190.0	496.0	
3	-122.25	37.85	52.0	1274.0	235.0	558.0	

Next steps: Generate code with housing_data | View recommended plots | New interactive sheet

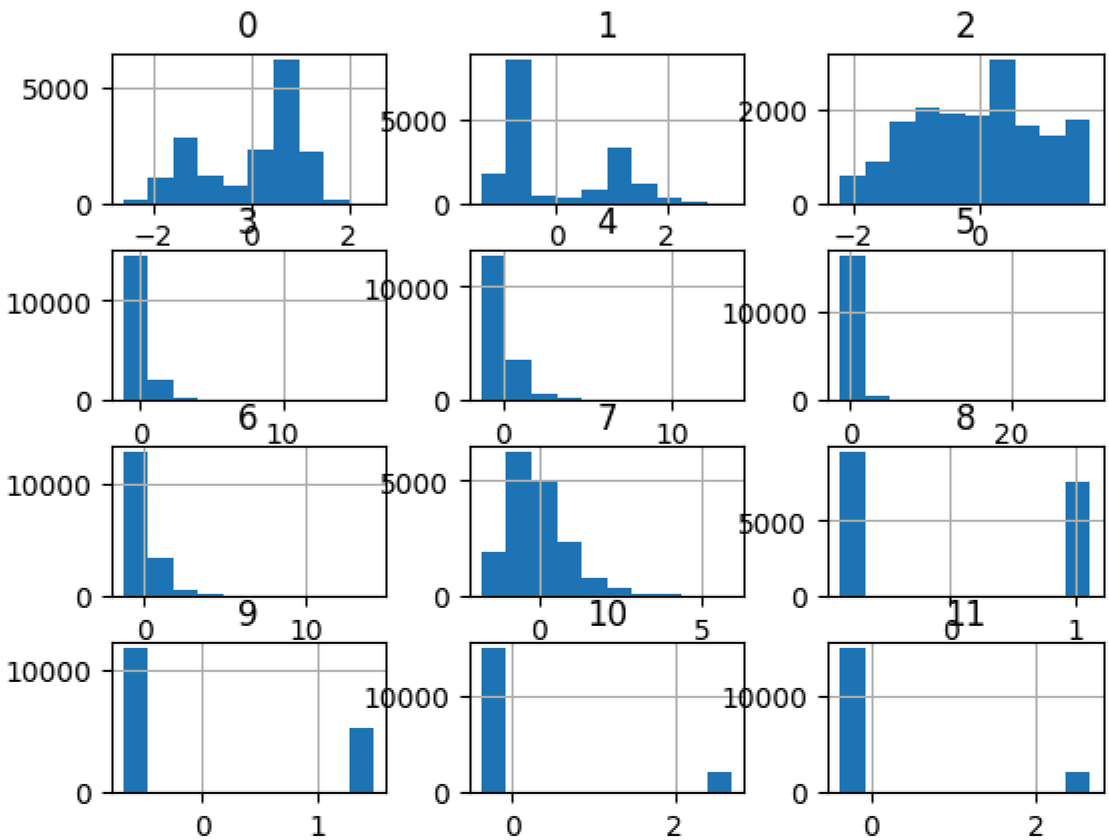
Preparing the data to be fed into the model

```
housing_data.shape
```

```
(20433, 13)
```

```
# Splitting the data into Training, Validation and testing sets
train_set, val_set, test_set = housing_data[1:17000], housing_data[17000:19000], housi

# Splitting data and labels
X_train, y_train = train_set.drop('median_house_value', axis = 1), train_set['median_h
X_val, y_val = val_set.drop('median_house_value', axis = 1), val_set['median_house_val
X_test, y_test = test_set.drop('median_house_value', axis = 1), test_set['median_house


# Standardising the data

standardize = StandardScaler()
X_train_std = standardize.fit_transform(X_train)
X_val_std = standardize.transform(X_val)
X_test_std = standardize.transform(X_test)


# Generating a historgram to observe the difference

pd.DataFrame(X_train_std).hist()
```

```
array([[<Axes: title={'center': '0'}>, <Axes: title={'center': '1'}>,
        <Axes: title={'center': '2'}>],
       [<Axes: title={'center': '3'}>, <Axes: title={'center': '4'}>,
        <Axes: title={'center': '5'}>],
       [<Axes: title={'center': '6'}>, <Axes: title={'center': '7'}>,
        <Axes: title={'center': '8'}>],
       [<Axes: title={'center': '9'}>, <Axes: title={'center': '10'}>,
        <Axes: title={'center': '11'}>]], dtype=object)
```



Ordinary Least Squares Regression Model

```
# Training the model

ols = LinearRegression()
ols.fit(X_train_std, y_train)
```

```
▾ LinearRegression
LinearRegression()
```

```
# Display the intercept, coeffiecients and R-squared values of the model

print("The intercept is:", ols.intercept_)
print("\nThe coefficients are:\n", ols.coef_)
print("\nThe R-squared value for: ", ols.score(X_train_std, y_train))
```

```
The intercept is: 205461.62027178076

The coefficients are:
 [-55991.92039902 -55717.79776612  14244.19712472 -15715.83928676
  45883.68061283 -43867.43145999  19405.60609718  74928.30490767
 -74905.48923332 -88084.75491088 -52016.06585172 -48566.79303563]

The R-squared value for:  0.6330493362487711
```

```
# Performing prediction
y_pred = ols.predict(X_test_std)

# making a dataframe with the true value and the predicted value
performance = pd.DataFrame({'True_val': y_test, 'Pred_val': y_pred})
performance.head()
```

	True_val	Pred_val
19185	384600.0	299506.008633
19186	221100.0	188912.500449
19187	293500.0	246482.476514
19188	242600.0	204967.661805
19189	172200.0	223225.533149

Next steps: Generate code with performance | View recommended plots | New interactive sheet

```
# Calculating the difference between the True value and prediction
performance['error'] = performance['True_val'] - performance['Pred_val']
performance.head()
```

	True_val	Pred_val	error
19185	384600.0	299506.008633	85093.991367
19186	221100.0	188912.500449	32187.499551
19187	293500.0	246482.476514	47017.523486
19188	242600.0	204967.661805	37632.338195
19189	172200.0	223225.533149	-51025.533149

Next steps: Generate code with performance | View recommended plots | New interactive sheet

```
# Plotting a graph to visualise the error

# Resetting the index and adding it as a column
performance.reset_index(drop = True, inplace = True)
performance.reset_index(inplace = True)

performance.head()
```

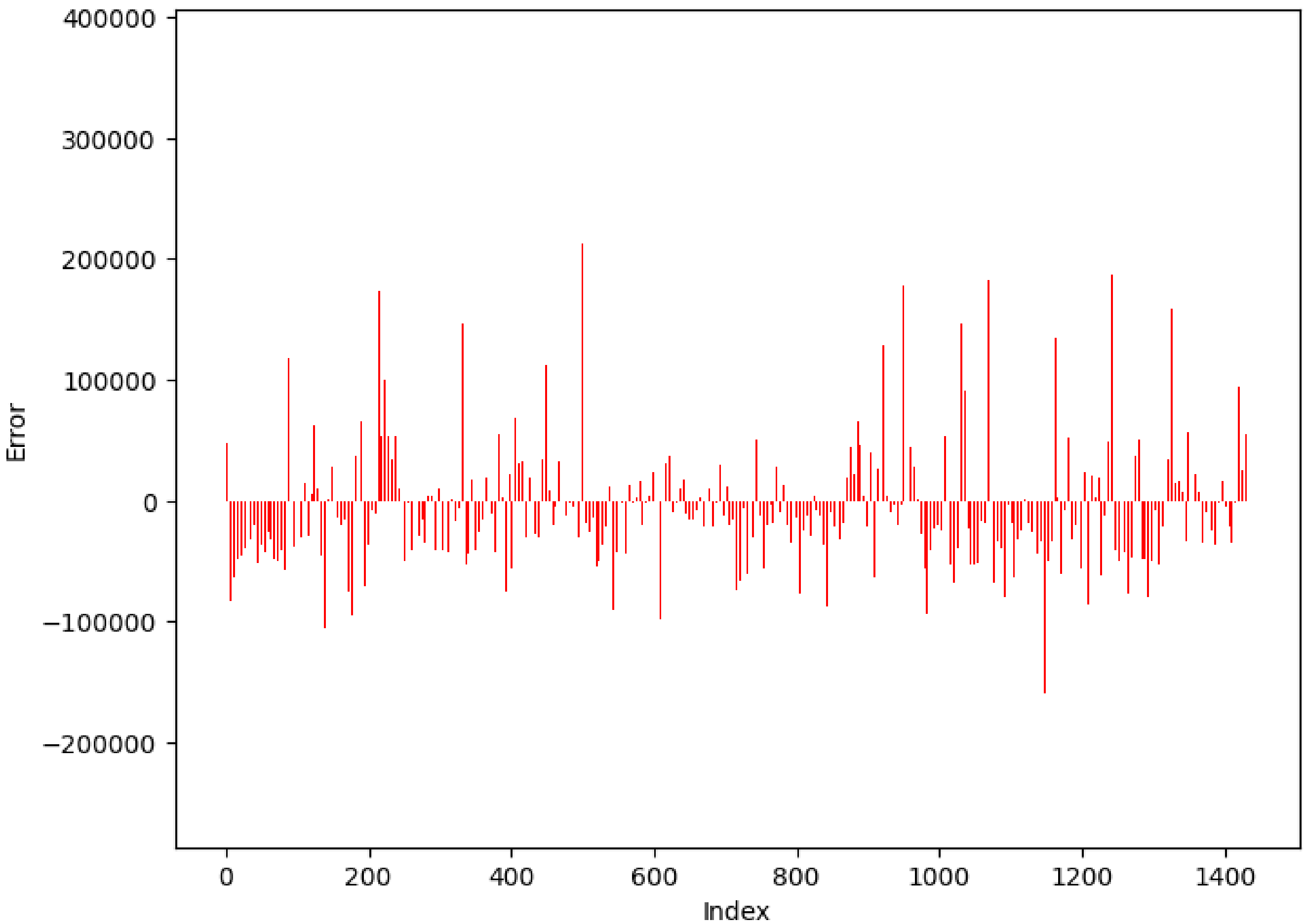
	index	True_val	Pred_val	error
0	0	384600.0	299506.008633	85093.991367
1	1	221100.0	188912.500449	32187.499551
2	2	293500.0	246482.476514	47017.523486
3	3	242600.0	204967.661805	37632.338195
4	4	172200.0	223225.533149	-51025.533149

Next steps: Generate code with performance | View recommended plots | New interactive sheet

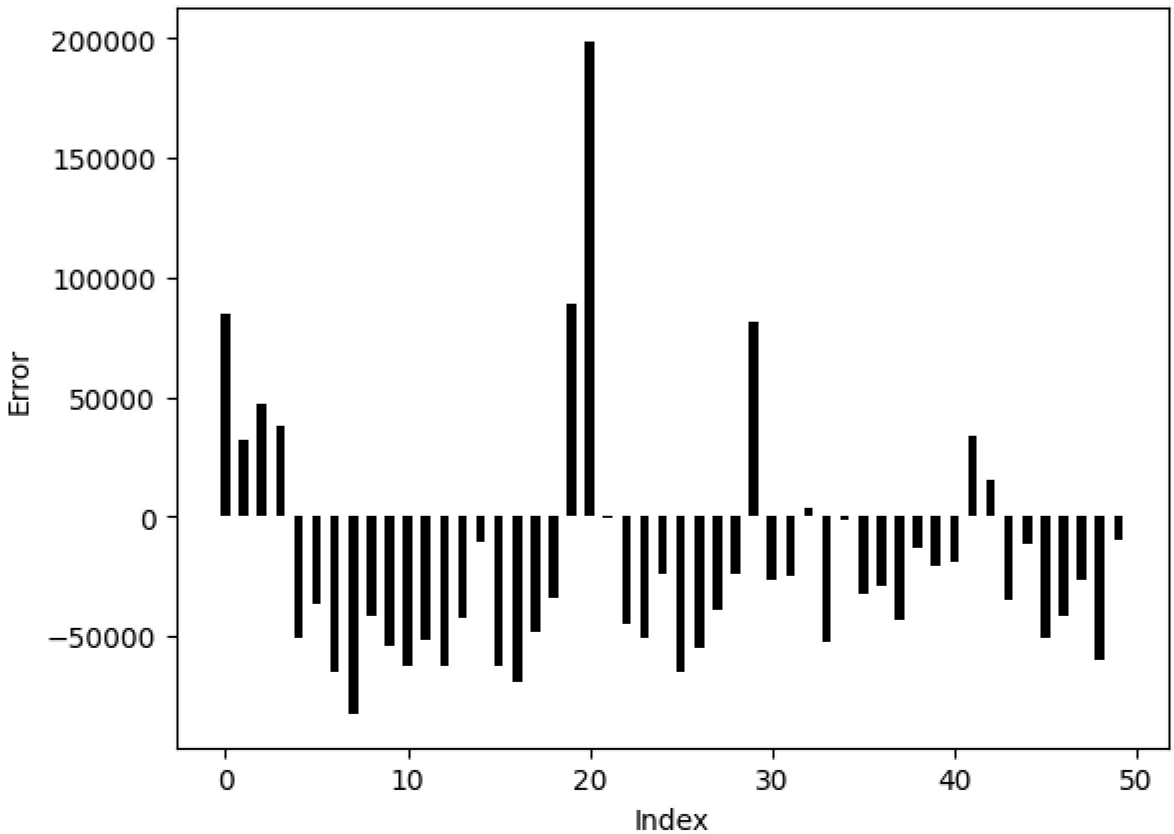
```
performance.shape
```

```
(1433, 4)
```

```
# Plotting the bar chart
fig = plt.figure(figsize = (8, 6))

plt.bar(performance['index'], performance['error'], data = performance, color = 'red',
plt.xlabel('Index')
plt.ylabel('Error')
plt.show()
```




```
# Observing just a few observations - zooming into on the first 50 observation

plt.bar('index', 'error', data = performance[:50], color = 'black', width = 0.5)
plt.xlabel('Index')
plt.ylabel('Error')
plt.show()
```




```
print("The Mean error for the training set:", np.sqrt(mse(ols.predict(X_train_std), y_
print("The Mean error for the Validation set:", np.sqrt(mse(ols.predict(X_val_std), y_
```

```
The Mean error for the training set: 69627.58905924245
The Mean error for the Validation set: 69916.03027565347
```

This graph shows how the model underestimates or overestimates the price.

- If the residual value is positive, the model has underestimated the price.
- If the residual value is negative, the model has overestimated the price of the house.

Random Forest Model

```
rfr = RandomForestRegressor(max_depth = 5).fit(X_train_std, y_train)

print("The mean error for the training data:", np.sqrt(mse(rfr.predict(X_train_std), y
print("The mean error for the validation data:", np.sqrt(mse(rfr.predict(X_val_std), y
```

```
The mean error for the training data: 67461.31075288086
The mean error for the validation data: 71496.65676748301
```

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import *
from tensorflow.keras.callbacks import ModelCheckpoint
from tensorflow.keras.metrics import RootMeanSquaredError
from tensorflow.keras.optimizers import Adam

# Simple Neural Net
simple_nn = Sequential()
simple_nn.add(InputLayer(input_shape = (X_train_std.shape[1],)))
simple_nn.add(Dense(32, 'relu'))
simple_nn.add(Dense(16, 'relu'))
simple_nn.add(Dense(1,'linear'))

opt = Adam(learning_rate = 0.5)
cp = ModelCheckpoint('models/simple_nn.keras', save_best_only = True)

simple_nn.compile(optimizer = opt, loss = 'mse', metrics = [RootMeanSquaredError()])
simple_nn.fit(x = X_train_std, y = y_train, validation_data = (X_val_std, y_val), call
```

```
Epoch 73/100
532/532 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 2734520064.0000 - root_mean_squared_
Epoch 74/100
532/532 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 2730743808.0000 - root_mean_squared_
Epoch 75/100
532/532 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - loss: 2712798976.0000 - root_mean_squared_
Epoch 76/100
532/532 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - loss: 2757194496.0000 - root_mean_squared_
Epoch 77/100
532/532 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 2638529280.0000 - root_mean_squared_
Epoch 78/100
532/532 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 2790941696.0000 - root_mean_squared_
Epoch 79/100
532/532 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 2694113280.0000 - root_mean_squared_
Epoch 80/100
532/532 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 2681300736.0000 - root_mean_squared_
```