```
#Importing necessary libraries


import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.preprocessing import StandardScaler

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LinearRegression, LogisticRegression

from sklearn.model_selection import GridSearchCV

from sklearn.metrics import root_mean_squared_error, r2_score, accuracy_score, precision_score,
recall_score, f1_score, mean_absolute_error


# Importing the dataset


df = pd.read_excel('real_estate_valuation.xlsx', index_col=0)

df


#Analyzing the dataset


df.shape

df.describe()

df.info()


#Finding missing values


df.isna().sum()


#Extracting transaction year and dropping transaction date


df['transaction year'] = df['X1 transaction date'].apply(lambda x: int(x))
```

```
df = df.drop(columns = 'X1 transaction date')
df.info()

#Histogram plot to visualize distribution of data

plt.figure(figsize=(15, 10))

for i, j in enumerate(df.columns, 1):
    plt.subplot(4, 2, i)
    sns.histplot(df[j], kde=True, bins=30)
    plt.title(f'Distribution of {j}')
    plt.xlabel(j)
    plt.ylabel('Frequency')

plt.tight_layout()
plt.show()

#Boxplot for better visualization of outliers in features and their spread

plt.figure(figsize=(15, 10))

for i, j in enumerate(df.columns):
    plt.subplot(3, 3, i+1)
    sns.boxplot(x=df[j])
    plt.title(f'Boxplot for {j}')

plt.tight_layout()
plt.show()

#Logarithmic transformation of ‘distance to nearest MRT station’ to counteract skewness and handle outliers
```

```
df['X3 distance to the nearest MRT station'] = np.log1p(df['X3 distance to the nearest MRT station'])
df['X3 distance to the nearest MRT station'].hist(bins=50)
sns.boxplot(x=df['X3 distance to the nearest MRT station'])

#Forming a correlation matrix and generating a heatmap

corr_matrix = df.corr()
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm')
plt.show()

#splitting data into 75-25% training and test set

X = df.drop(columns = 'Y house price of unit area')
Y = df['Y house price of unit area']

X_train, X_test, Y_train, Y_test = train_test_split(X,Y,
                                                    test_size = 0.25,
                                                    random_state = 42)

#Normalization of the features using StrandardScaler

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

#Using GridSearch to find the optimal parameters for linear regression

linear_regression_params = {
    'fit_intercept': [True, False],
    'copy_X': [True, False]
}


linear_regression_grid = GridSearchCV(estimator=LinearRegression(),
param_grid=linear_regression_params,
                              scoring='r2', cv=5, n_jobs=-1)


linear_regression_grid.fit(X_train_scaled, Y_train)

best_lr_model = linear_regression_grid.best_estimator_

best_lr_params = linear_regression_grid.best_params_


print("Best Linear Regression Parameters:", best_lr_params)


#Linear regression with optimal parameters


lin_reg = LinearRegression() #default hyperparameter settings are found to be the best

lin_reg.fit(X_train_scaled, Y_train)

y_pred_linear = lin_reg.predict(X_test_scaled)


#Evaluation of linear regression model


linear_rmse = root_mean_squared_error(Y_test, y_pred_linear)

linear_r2 = r2_score(Y_test, y_pred_linear)

linear_mae = mean_absolute_error(Y_test, y_pred_linear)


print(f'Root Mean Squared Error : {linear_rmse}\nR2-score : {linear_r2}\nMean Absolute Error :
{linear_mae}')


#Binarization of target value


median_price = Y_train.median()

y_train_binary = (Y_train >= median_price).astype(int)

y_test_binary = (Y_test >= median_price).astype(int)
```

```
#Using GridSearch to find optimal parameters for logistic regression model

logistic_regression_params = {
   'max_iter' : [1000, 5000, 10000]
   }

logistic_regression_grid = GridSearchCV(estimator=LogisticRegression(random_state = 42),
param_grid=logistic_regression_params,
                                   scoring='accuracy', cv=5, n_jobs=-1)

logistic_regression_grid.fit(X_train_scaled, y_train_binary)
best_log_model = logistic_regression_grid.best_estimator_
best_log_params = logistic_regression_grid.best_params_
print("Best Linear Regression Parameters:", best_log_params)

#Logistic regression with optimal parameters

log_reg = LogisticRegression(max_iter=1000, class_weight = 'balanced', random_state=42)
log_reg.fit(X_train_scaled, y_train_binary)
y_pred_logistic = log_reg.predict(X_test_scaled)

#Evaluation of logistic regression model

logistic_accuracy = accuracy_score(y_test_binary, y_pred_logistic)
logistic_precision = precision_score(y_test_binary, y_pred_logistic)
logistic_recall = recall_score(y_test_binary, y_pred_logistic)
logistic_f1 = f1_score(y_test_binary, y_pred_logistic)

print(f'Accuracy : {logistic_accuracy}\nPrecision : {logistic_precision}\nRecall : {logistic_recall}\nF1-
score : {logistic_f1}')
```

```
#Printing results of both models in table format

performance_results = {
    'Metric': [
        'Root Mean Squared Error',
        'Mean Absolute Error',
        'R^2 Score',
        'Accuracy',
        'Precision',
        'Recall',
        'F1 Score'
    ],
    'Linear Regression': [
        linear_rmse,
        linear_mae,
        linear_r2,
        None,  # No value for linear regression metrics
        None,
        None,
        None
    ],
    'Logistic Regression': [
        None,  # No value for logistic regression metrics
        None,
        None,
        logistic_accuracy,
        logistic_precision,
        logistic_recall,
        logistic_f1
    ]
}
```

```
results_df = pd.DataFrame(performance_results)

results_df
```