```
import numpy as np  # --> Importing NumPy for numerical operations

import pandas as pd  # --> Importing Pandas for data manipulation

import seaborn as sns  # --> Importing Seaborn for data visualization

import matplotlib.pyplot as plt  # --> Importing Matplotlib for plotting graphs

from sklearn.model_selection import train_test_split  # --> Importing function to split data

from sklearn.preprocessing import StandardScaler  # --> Importing scaler for normalization

from sklearn.linear_model import LinearRegression  # --> Importing Linear Regression model

from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score  # --> Importing evaluation metrics

import keras  # --> Importing Keras for building neural networks

from keras.models import Sequential  # --> Importing Sequential model from Keras

from keras.layers import Dense  # --> Importing Dense layer for neural network

import plotly.graph_objects as go  # --> Importing Plotly for interactive plots


url = "https://lib.stat.cmu.edu/datasets/boston"  # --> Boston housing dataset URL

raw_df = pd.read_csv(url, sep="\s+", skiprows=22, header=None)  # --> Reading dataset from URL

data = np.hstack([raw_df.values[::2, :], raw_df.values[1::2, :2]])  # --> Combining feature rows

target = raw_df.values[1::2, 2]  # --> Extracting target variable (PRICE)


feature_names = ['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS',  # --> Defining column names

                 'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT']

data = pd.DataFrame(data, columns=feature_names)  # --> Creating DataFrame with features

data['PRICE'] = target  # --> Adding target column to DataFrame


data.head()  # --> Displaying first few rows of dataset


print("First look at the data:")  # --> Printing title

print(data.head())  # --> Printing first few rows

print("\nData shape:", data.shape)  # --> Printing shape of dataset

print("\nNull values check:")  # --> Checking for null values
```

```
print(data.isnull().sum())  # --> Summing null values in each column
print("\nData statistics:")  # --> Descriptive statistics of dataset
print(data.describe())  # --> Showing summary statistics

sns.set(style="whitegrid")  # --> Setting Seaborn style

plt.figure(figsize=(12, 6))  # --> Setting figure size
sns.distplot(data['PRICE'])  # --> Plotting distribution of house prices
plt.title('Distribution of House Prices')  # --> Title of the plot
plt.show()  # --> Displaying the plot

data.head(n=10)  # --> Viewing first 10 rows of the dataset

print(data.shape)  # --> Printing shape of dataset

data.isnull().sum()  # --> Checking for null values

data.describe()  # --> Showing dataset statistics

import seaborn as sns  # --> Re-importing Seaborn (already imported above)
sns.distplot(data.PRICE)  # --> Plotting price distribution again

sns.boxplot(data.PRICE)  # --> Drawing boxplot for target variable

correlation = data.corr()  # --> Calculating correlation matrix
correlation.loc['PRICE']  # --> Viewing correlation with PRICE

import matplotlib.pyplot as plt  # --> Re-importing matplotlib (already imported above)
fig, axes = plt.subplots(figsize=(15, 12))  # --> Setting up heatmap plot size
sns.heatmap(correlation, square=True, annot=True)  # --> Plotting heatmap with annotations
```

```
plt.figure(figsize=(20, 5))  # --> Plotting figure for scatter plots

features = ['LSTAT', 'RM', 'PTRATIO']  # --> Features for scatter plot

for i, col in enumerate(features):  # --> Looping through selected features

   plt.subplot(1, len(features), i+1)  # --> Creating subplots

   x = data[col]  # --> Selecting x values

   y = data.PRICE  # --> Setting y as target variable

   plt.scatter(x, y, marker='o')  # --> Plotting scatter points

   plt.title("Variation in House prices")  # --> Plot title

   plt.xlabel(col)  # --> X-axis label

   plt.ylabel('House prices in $1000')  # --> Y-axis label


X = data.iloc[:, :-1]  # --> Selecting all columns except target as features

y = data.PRICE  # --> Selecting target variable


from sklearn.model_selection import train_test_split  # --> Re-importing train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)  # -->
Splitting dataset


mean = X_train.mean(axis=0)  # --> Calculating feature means

std = X_train.std(axis=0)  # --> Calculating feature standard deviations

X_train = (X_train - mean) / std  # --> Normalizing training features

X_test = (X_test - mean) / std  # --> Normalizing test features


from sklearn.linear_model import LinearRegression  # --> Re-importing Linear Regression

regressor = LinearRegression()  # --> Creating Linear Regression model


regressor.fit(X_train, y_train)  # --> Training the Linear Regression model


y_pred = regressor.predict(X_test)  # --> Making predictions on test set


from sklearn.metrics import mean_squared_error  # --> Re-importing MSE
```

```
rmse = (np.sqrt(mean_squared_error(y_test, y_pred)))  # --> Calculating RMSE
print(rmse)  # --> Printing RMSE

from sklearn.metrics import r2_score  # --> Re-importing R2 Score
r2 = r2_score(y_test, y_pred)  # --> Calculating R2 score
print(r2)  # --> Printing R2 score

from sklearn.preprocessing import StandardScaler  # --> Re-importing StandardScaler
sc = StandardScaler()  # --> Creating StandardScaler object
X_train = sc.fit_transform(X_train)  # --> Scaling training data
X_test = sc.transform(X_test)  # --> Scaling test data

import keras  # --> Re-importing Keras
from keras.layers import Dense, Activation, Dropout  # --> Importing Keras layers
from keras.models import Sequential  # --> Re-importing Sequential model

model = Sequential()  # --> Initializing Sequential model
model.add(Dense(128, activation='relu', input_dim=13))  # --> Adding input layer with 128 neurons
model.add(Dense(64, activation='relu'))  # --> Adding hidden layer with 64 neurons
model.add(Dense(32, activation='relu'))  # --> Adding hidden layer with 32 neurons
model.add(Dense(16, activation='relu'))  # --> Adding hidden layer with 16 neurons
model.add(Dense(1))  # --> Adding output layer with 1 neuron

model.compile(optimizer='adam', loss='mean_squared_error', metrics=['mae'])  # --> Compiling model
history = model.fit(X_train, y_train, epochs=100, validation_split=0.05)  # --> Training model with validation

from plotly.subplots import make_subplots  # --> Importing subplots from Plotly
import plotly.graph_objects as go  # --> Re-importing Plotly
```

```
fig = go.Figure()  # --> Initializing figure

fig.add_trace(go.Scattergl(y=history.history['loss'], name='Train'))  # --> Adding training loss plot

fig.add_trace(go.Scattergl(y=history.history['val_loss'], name='Valid'))  # --> Adding validation loss plot

fig.update_layout(height=500, width=700, xaxis_title='Epoch', yaxis_title='Loss')  # --> Updating plot layout

fig.show()  # --> Showing the plot


fig = go.Figure()  # --> Creating new figure

fig.add_trace(go.Scattergl(y=history.history['mae'], name='Train'))  # --> Adding training MAE plot

fig.add_trace(go.Scattergl(y=history.history['val_mae'], name='Valid'))  # --> Adding validation MAE plot

fig.update_layout(height=500, width=700, xaxis_title='Epoch', yaxis_title='Mean Absolute Error')  # --> Layout update

fig.show()  # --> Showing the plot


y_pred = model.predict(X_test)  # --> Predicting test data with neural network

mse_nn, mae_nn = model.evaluate(X_test, y_test)  # --> Evaluating model performance

print('Mean squared error on test data: ', mse_nn)  # --> Printing neural network MSE

print('Mean absolute error on test data: ', mae_nn)  # --> Printing neural network MAE


from sklearn.metrics import mean_absolute_error  # --> Re-importing MAE


lr_model = LinearRegression()  # --> Creating Linear Regression model again

lr_model.fit(X_train, y_train)  # --> Training the model

y_pred_lr = lr_model.predict(X_test)  # --> Predicting test set

mse_lr = mean_squared_error(y_test, y_pred_lr)  # --> Calculating MSE

mae_lr = mean_absolute_error(y_test, y_pred_lr)  # --> Calculating MAE


print('Mean squared error on test data: ', mse_lr)  # --> Printing MSE

print('Mean absolute error on test data: ', mae_lr)  # --> Printing MAE
```

```
from sklearn.metrics import r2_score  # --> Re-importing R2 Score again

r2 = r2_score(y_test, y_pred)  # --> Recalculating R2 score

print(r2)  # --> Printing R2


from sklearn.metrics import mean_squared_error  # --> Re-importing MSE again

rmse = (np.sqrt(mean_squared_error(y_test, y_pred)))  # --> Recalculating RMSE

print(rmse)  # --> Printing RMSE


import sklearn  # --> Importing sklearn for preprocessing

new_data = sklearn.preprocessing.StandardScaler().fit_transform(([[0.1, 10.0, 5.0, 0, 0.4, 6.0,
50, 6.0, 1, 400, 20, 300, 10]]))  # --> Scaling new input data

prediction = model.predict(new_data)  # --> Predicting price for new data

print("Predicted house price:", prediction)  # --> Printing predicted price
```