```
# IMPORTANT: RUN THIS CELL IN ORDER TO IMPORT YOUR KA
# TO THE CORRECT LOCATION (/kaggle/input) IN YOUR NOT
# THEN FEEL FREE TO DELETE THIS CELL.
# NOTE: THIS NOTEBOOK ENVIRONMENT DIFFERS FROM KAGGLE
# ENVIRONMENT SO THERE MAY BE MISSING LIBRARIES USED
# NOTEBOOK.

import os
import sys
from tempfile import NamedTemporaryFile
from urllib.request import urlopen
from urllib.parse import unquote, urlparse
from urllib.error import HTTPError
from zipfile import ZipFile
import tarfile
import shutil

CHUNK_SIZE = 40960
DATA_SOURCE_MAPPING = 'boston-house-price-prediction:

KAGGLE_INPUT_PATH='/kaggle/input'
KAGGLE_WORKING_PATH='/kaggle/working'
KAGGLE_SYMLINK='kaggle'

!umount /kaggle/input/ 2> /dev/null
shutil.rmtree('/kaggle/input', ignore_errors=True)
os.makedirs(KAGGLE_INPUT_PATH, 0o777, exist_ok=True)
os.makedirs(KAGGLE_WORKING_PATH, 0o777, exist_ok=True

try:
  os.symlink(KAGGLE_INPUT_PATH, os.path.join("..", 'i
except FileExistsError:
  pass
try:
  os.symlink(KAGGLE_WORKING_PATH, os.path.join("..",
except FileExistsError:
  pass
```

```
    pass

for data_source_mapping in DATA_SOURCE_MAPPING.split(
    directory, download_url_encoded = data_source_map
    download_url = unquote(download_url_encoded)
    filename = urlparse(download_url).path
    destination_path = os.path.join(KAGGLE_INPUT_PATH
    try:
        with urlopen(download_url) as fileres, NamedT
            total_length = fileres.headers['content-l
            print(f'Downloading {directory}, {total_l
            dl = 0
            data = fileres.read(CHUNK_SIZE)
            while len(data) > 0:
                dl += len(data)
                tfile.write(data)
                done = int(50 * dl / int(total_length
                sys.stdout.write(f"\r[{'=' * done}{'
                sys.stdout.flush()
                data = fileres.read(CHUNK_SIZE)
            if filename.endswith('.zip'):
              with ZipFile(tfile) as zfile:
                zfile.extractall(destination_path)
            else:
              with tarfile.open(tfile.name) as tarfil
                tarfile.extractall(destination_path)
            print(f'\nDownloaded and uncompressed: {d
    except HTTPError as e:
        print(f'Failed to load (likely expired) {down
        continue
    except OSError as e:
        print(f'Failed to load {download_url} to path
        continue

print('Data source import complete.')
```

```
Downloading boston-house-price-prediction, 11892 bytes compressed
[==================================================] 11892 bytes downloaded
Downloaded and uncompressed: boston-house-price-prediction
Data source import complete.
```

```
# Importing necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Load the dataset
data = pd.read_csv('/kaggle/input/boston-house-price-

#Displaying the first few rows of the dataframe
print("First 5 rows of the dataset:")
print(data.head())
```

```
First 5 rows of the dataset:
      crim    zn  indus  chas    nox     rm   age     dis  rad  tax  ptratio  \
0  0.00632  18.0   2.31     0  0.538  6.575  65.2  4.0900    1  296     15.3
1  0.02731   0.0   7.07     0  0.469  6.421  78.9  4.9671    2  242     17.8
2  0.02729   0.0   7.07     0  0.469  7.185  61.1  4.9671    2  242     17.8
3  0.03237   0.0   2.18     0  0.458  6.998  45.8  6.0622    3  222     18.7
4  0.06905   0.0   2.18     0  0.458  7.147  54.2  6.0622    3  222     18.7

        b  lstat  medv
0  396.90   4.98  24.0
1  396.90   9.14  21.6
2  392.83   4.03  34.7
3  394.63   2.94  33.4
4  396.90   5.33  36.2
```

```
# Checking for any missing values in the dataset
print("\nMissing values in the dataset:")
print(data.isnull().sum())
```

```
Missing values in the dataset:
crim       0
zn         0
indus      0
chas       0
nox        0
rm         5
age        0
dis        0
rad        0
tax        0
ptratio    0
b          0
lstat      0
medv       0
dtype: int64
```

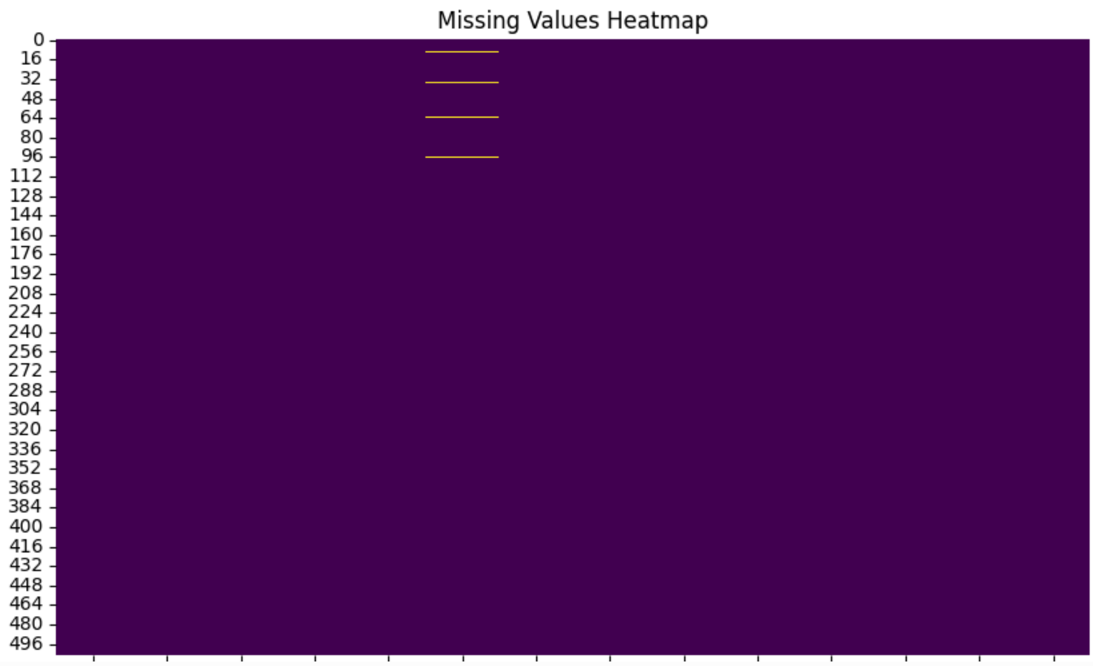
```
# Displaying the summary statistics of the dataset
print("\nSummary Statistics:")
print(data.describe())
```

```
Summary Statistics:
             crim          zn       indus        chas         nox          rm  \
count  506.000000  506.000000  506.000000  506.000000  506.000000  501.000000
mean     3.613524   11.363636   11.136779    0.069170    0.554695    6.284341
std      8.601545   23.322453    6.860353    0.253994    0.115878    0.705587
min      0.006320    0.000000    0.460000    0.000000    0.385000    3.561000
25%      0.082045    0.000000    5.190000    0.000000    0.449000    5.884000
50%      0.256510    0.000000    9.690000    0.000000    0.538000    6.208000
75%      3.677083   12.500000   18.100000    0.000000    0.624000    6.625000
max     88.976200  100.000000   27.740000    1.000000    0.871000    8.780000
```

```
              age         dis         rad         tax     ptratio           b  \
count  506.000000  506.000000  506.000000  506.000000  506.000000  506.000000
mean    68.574901    3.795043    9.549407  408.237154   18.455534  356.674032
std     28.148861    2.105710    8.707259  168.537116    2.164946   91.294864
min      2.900000    1.129600    1.000000  187.000000   12.600000    0.320000
25%     45.025000    2.100175    4.000000  279.000000   17.400000  375.377500
50%     77.500000    3.207450    5.000000  330.000000   19.050000  391.440000
75%     94.075000    5.188425   24.000000  666.000000   20.200000  396.225000
max    100.000000   12.126500   24.000000  711.000000   22.000000  396.900000

            lstat        medv
count  506.000000  506.000000
mean    12.653063   22.532806
std      7.141062    9.197104
min      1.730000    5.000000
25%      6.950000   17.025000
50%     11.360000   21.200000
75%     16.955000   25.000000
max     37.970000   50.000000
```

```
# Visualizing missing data (optional, for better unde
plt.figure(figsize=(10, 6))
sns.heatmap(data.isnull(), cbar=False, cmap="viridis"
plt.title('Missing Values Heatmap')
plt.show()
```



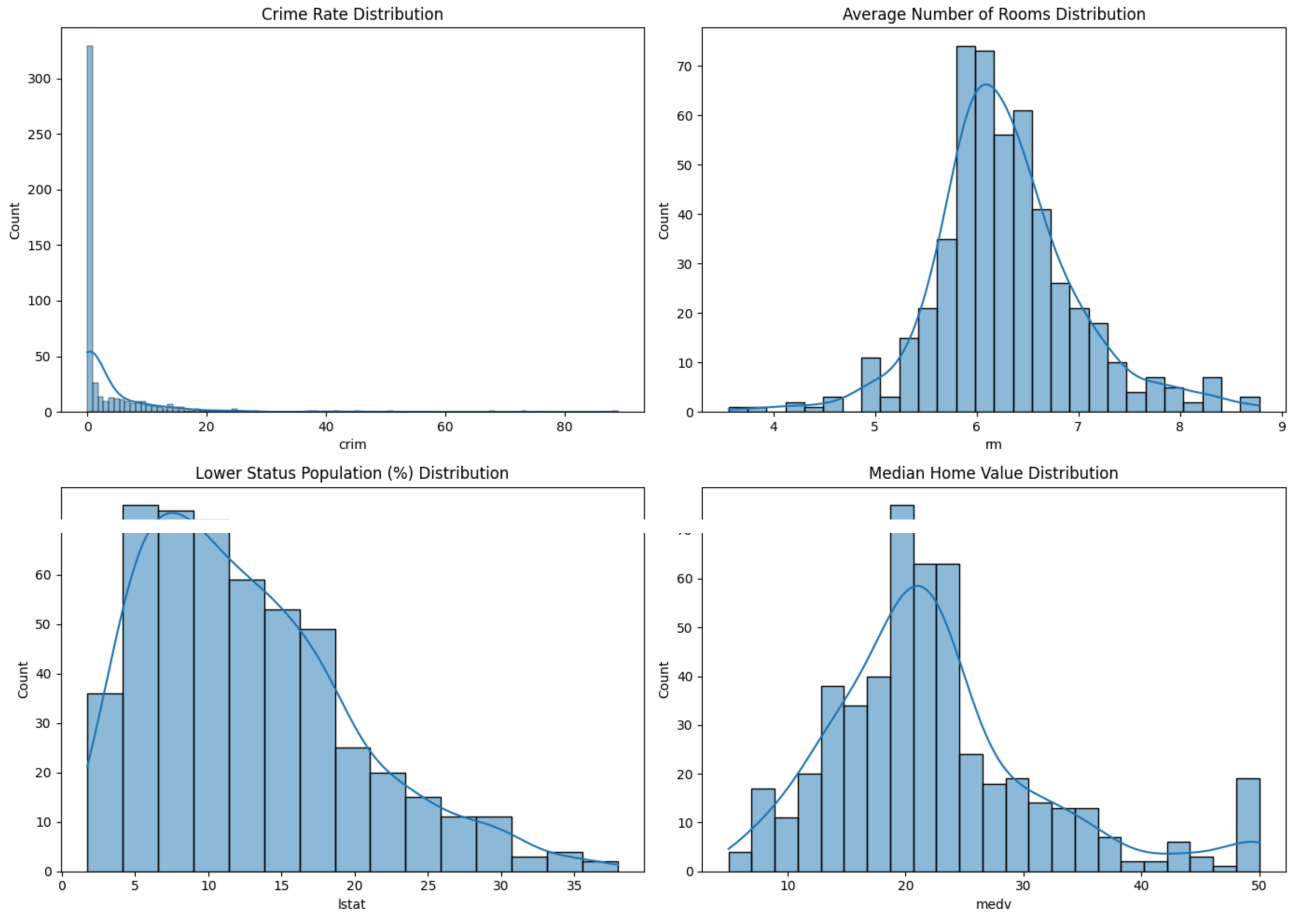

```
# Visualizing the distribution of a few important fea
plt.figure(figsize=(14, 10))
plt.subplot(2, 2, 1)
sns.histplot(data['crim'], kde=True)
plt.title('Crime Rate Distribution')

plt.subplot(2, 2, 2)
sns.histplot(data['rm'], kde=True)
plt.title('Average Number of Rooms Distribution')
```

```
plt.subplot(2, 2, 3)
sns.histplot(data['lstat'], kde=True)
plt.title('Lower Status Population (%) Distribution')

plt.subplot(2, 2, 4)
sns.histplot(data['medv'], kde=True)
plt.title('Median Home Value Distribution')

plt.tight_layout()
plt.show()
```



Question No.02

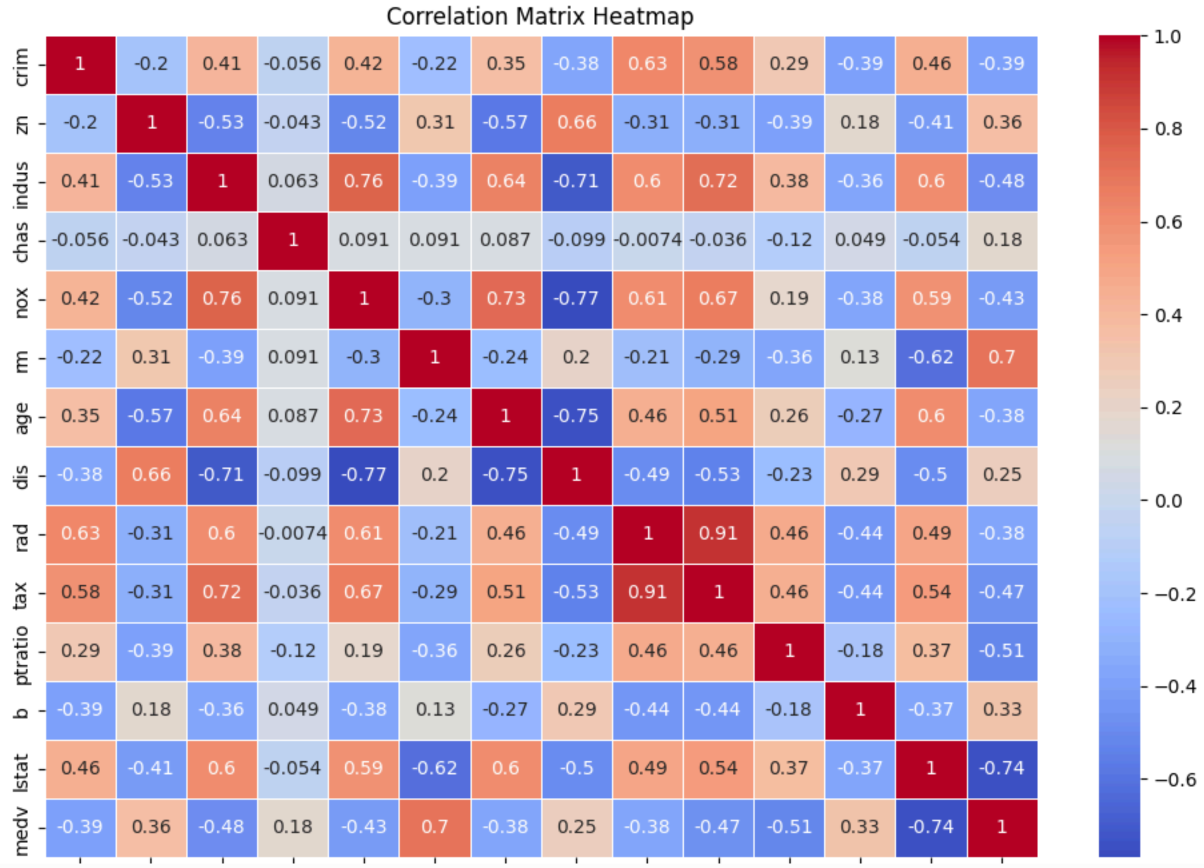
```python
# Calculate the correlation matrix
correlation_matrix = data.corr()

# Displaying the correlation matrix
print("Correlation Matrix:")
print(correlation_matrix)
```

```
Correlation Matrix:
             crim        zn     indus      chas       nox        rm       age  \
crim     1.000000 -0.200469  0.406583 -0.055892  0.420972 -0.219433  0.352734
zn      -0.200469  1.000000 -0.533828 -0.042697 -0.516604  0.311173 -0.569537
indus    0.406583 -0.533828  1.000000  0.062938  0.763651 -0.394193  0.644779
chas    -0.055892 -0.042697  0.062938  1.000000  0.091203  0.091468  0.086518
nox      0.420972 -0.516604  0.763651  0.091203  1.000000 -0.302751  0.731470
rm      -0.219433  0.311173 -0.394193  0.091468 -0.302751  1.000000 -0.240286
age      0.352734 -0.569537  0.644779  0.086518  0.731470 -0.240286  1.000000
dis     -0.379670  0.664408 -0.708027 -0.099176 -0.769230  0.203507 -0.747881
rad      0.625505 -0.311948  0.595129 -0.007368  0.611441 -0.210718  0.456022
tax      0.582764 -0.314563  0.720760 -0.035587  0.668023 -0.292794  0.506456
ptratio  0.289946 -0.391679  0.383248 -0.121515  0.188933 -0.357612  0.261515
b       -0.385064  0.175520 -0.356977  0.048788 -0.380051  0.128107 -0.273534
lstat    0.455621 -0.412995  0.603800 -0.053929  0.590879 -0.615721  0.602339
medv    -0.388305  0.360445 -0.483725  0.175260 -0.427321  0.696169 -0.376955

              dis       rad       tax   ptratio         b     lstat      medv
crim    -0.379670  0.625505  0.582764  0.289946 -0.385064  0.455621 -0.388305
zn       0.664408 -0.311948 -0.314563 -0.391679  0.175520 -0.412995  0.360445
indus   -0.708027  0.595129  0.720760  0.383248 -0.356977  0.603800 -0.483725
chas    -0.099176 -0.007368 -0.035587 -0.121515  0.048788 -0.053929  0.175260
nox     -0.769230  0.611441  0.668023  0.188933 -0.380051  0.590879 -0.427321
rm       0.203507 -0.210718 -0.292794 -0.357612  0.128107 -0.615721  0.696169
age     -0.747881  0.456022  0.506456  0.261515 -0.273534  0.602339 -0.376955
dis      1.000000 -0.494588 -0.534432 -0.232471  0.291512 -0.496996  0.249929
rad     -0.494588  1.000000  0.910228  0.464741 -0.444413  0.488676 -0.381626
tax     -0.534432  0.910228  1.000000  0.460853 -0.441808  0.543993 -0.468536
ptratio -0.232471  0.464741  0.460853  1.000000 -0.177383  0.374044 -0.507787
b        0.291512 -0.444413 -0.441808 -0.177383  1.000000 -0.366087  0.333461
lstat   -0.496996  0.488676  0.543993  0.374044 -0.366087  1.000000 -0.737663
medv     0.249929 -0.381626 -0.468536 -0.507787  0.333461 -0.737663  1.000000
```

```python
# Visualize the correlation matrix using a heatmap
plt.figure(figsize=(12, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coc
plt.title('Correlation Matrix Heatmap')
plt.show()
```




```
# Identify the features with the highest positive anc
# Assume 'medv' is the target variable (median home v
target_variable = 'medv'
correlation_with_target = correlation_matrix[target_v

# Display the features with highest positive and nega
print("\nFeatures with highest positive correlation w
print(correlation_with_target[correlation_with_target

print("\nFeatures with highest negative correlation w
print(correlation_with_target[correlation_with_target
```

```
Features with highest positive correlation with house prices:
medv     1.000000
rm       0.696169
zn       0.360445
b        0.333461
dis      0.249929
Name: medv, dtype: float64

Features with highest negative correlation with house prices:
age     -0.376955
rad     -0.381626
crim    -0.388305
nox     -0.427321
tax     -0.468536
Name: medv, dtype: float64
```

Question no.03

```
# Import necessary libraries
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Select the features and the target variable
X = data.drop(columns=['medv'])  # Features (all colu
y = data['medv']  # Target variable (house prices)

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X

# Standardize the feature variables using StandardSca
scaler = StandardScaler()

# Fit the scaler on the training data and transform b
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Print shapes to verify the splits
print("Training data shape:", X_train_scaled.shape)
print("Testing data shape:", X_test_scaled.shape)
```

```
Training data shape: (404, 13)
Testing data shape: (102, 13)
```

Question No.04

```
from sklearn.linear_model import LinearRegression
# Impute missing values using the mean for each colum
data.fillna(data.mean(), inplace=True)

# Re-select the features and target variable after im
X = data.drop(columns=['medv'])  # Features (all colu
y = data['medv']  # Target variable (house prices)

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X

# Standardize the feature variables using StandardSca
scaler = StandardScaler()
```

```
# Fit the scaler on the training data and transform b
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Train the Linear Regression model
model = LinearRegression()
model.fit(X_train_scaled, y_train)
```

```
▾ LinearRegression
```

```
# Display the model's coefficients and intercept
print("Model Coefficients:", model.coef_)
print("Model Intercept:", model.intercept_)
```

```
Model Coefficients: [-1.00208747  0.69855082  0.28733122  0.71955092 -2.02070833  3.13708935
 -0.17081271 -3.06972351  2.25417948 -1.76697719 -2.04359481  1.12936985
 -3.61451369]
Model Intercept: 22.796534653465343
```

Question No.05

```
# Import necessary libraries
from sklearn.metrics import mean_absolute_error, mear
import numpy as np

# Predict the house prices using the testing data
y_pred = model.predict(X_test_scaled)

# Calculate and display performance metrics
# Mean Absolute Error (MAE)
mae = mean_absolute_error(y_test, y_pred)
# Mean Squared Error (MSE)
mse = mean_squared_error(y_test, y_pred)
# Root Mean Squared Error (RMSE)
rmse = np.sqrt(mse)

print(f"Mean Absolute Error (MAE): {mae}")
print(f"Mean Squared Error (MSE): {mse}")
print(f"Root Mean Squared Error (RMSE): {rmse}")
```

```
Mean Absolute Error (MAE): 3.2064039639003856
Mean Squared Error (MSE): 24.40482518814648
Root Mean Squared Error (RMSE): 4.940124005341008
```

```
# Plot the predicted vs actual house prices
plt.figure(figsize=(8, 6))
```

```
plt.figure(figsize=(8, 6))
plt.scatter(y_test, y_pred, color='blue', edgecolor='
plt.plot([min(y_test), max(y_test)], [min(y_test), ma
plt.title('Predicted vs Actual House Prices')
plt.xlabel('Actual House Prices')
plt.ylabel('Predicted House Prices')
plt.grid(True)
plt.show()
```