BostonHousing Description:

Boston housing typically consists of a mix of architectural styles, ranging from historic brownstones and Victorian homes to modern condominiums and apartments. The city's neighborhoods offer diverse housing options to accommodate various preferences and budgets.

In terms of demographics, Boston is a vibrant and culturally rich city with a diverse population. Its housing market reflects this diversity, with neighborhoods catering to students, professionals, families, and retirees alike.

The cost of housing in Boston can vary significantly depending on the neighborhood, proximity to downtown, and the type of housing. Generally, areas closer to the city center tend to have higher housing prices, while suburbs and outlying neighborhoods may offer more affordable options.

Boston's housing market is known for its competitiveness, with high demand often leading to bidding wars and quick sales. This can make finding a suitable home a challenging task, especially for first-time buyers or those on a tight budget.

Despite the challenges, living in Boston offers many benefits, including access to world-class universities, cultural institutions, diverse dining options, and a robust public transportation system. These factors contribute to the city's appeal and help to maintain its status as a desirable place to live.

Project Goals:

Exploratory Data Analysis (EDA) of Boston housing data typically involves examining various aspects of the dataset to gain insights into the characteristics and trends of the housing market in Boston.

Dataset Overview: Begin by loading the dataset and understanding its structure. The Boston housing dataset often includes features such as median home value, crime rate, pupil-teacher ratio, etc.

Summary Statistics: Calculate summary statistics for numerical features like mean, median, standard deviation, minimum, and maximum values. This gives a basic understanding of the distribution of the data.

Data Visualization: Utilize visualizations such as histograms, box plots, and scatter plots to explore the distributions and relationships between variables. For example:

Histograms: Show the distribution of individual features, like housing prices or crime rates. Box plots: Visualize the distribution of a variable across different categories, such as median home value by neighborhood. Scatter plots: Explore relationships between pairs of variables, like median home value versus crime rate or median home value versus distance to employment centers.

Data Preprocessing: Prepare the data for further analysis or modeling by handling missing values, encoding categorical variables, and scaling numerical features.

Problem Statements :

1.Add a new column to the DataFrame that categorizes the ‘medv’ (median value of owner-occupied homes) column into ‘Low’, ‘Medium’, and ‘High’.

2.Replace the ‘chas’ column (Charles River dummy variable) with ‘Yes’ if chas = 1 and ‘No’ if chas = 0.

3.Rename the column ‘rm’ to ‘rooms’.

4.Find out the average number of rooms (‘rooms’ column) per dwelling for each category of ‘medv’ (Low, Medium, High).

5.Find out the percentage of houses that bound the Charles River.

6.Convert the ‘age’ column (proportion of owner-occupied units built prior to 1940) from the DataFrame to a NumPy array and perform the following operations:

A. Compute the mean and standard deviation.

B. Normalize the array.

7.min-max normalization.

8.z-score normalization.

** Importing the necessary libraries**

```
import numpy as np
import pandas as pd
```

Data loading

+ Code + Text

```
df=pd.read_csv('/content/BostonHousing.csv')
```

have a quick look on this data

```
df.head(5)
```

	crim	zn	indus	chas	nox	rm	age	dis	rad	tax	ptratio	b	lstat	medv
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	5.33	36.2

Data Exploration

Dimentiion of data

```
df.shape
```

```
(506, 14)
```

name of the columns

```
df.columns
```

```
Index(['crim', 'zn', 'indus', 'chas', 'nox', 'rm', 'age', 'dis', 'rad', 'tax',
       'ptratio', 'b', 'lstat', 'medv'],
      dtype='object')
```

Data Types Of The Columns

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
Data columns (total 14 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   crim     506 non-null    float64
 1   zn       506 non-null    float64
 2   indus    506 non-null    float64
 3   chas     506 non-null    int64  
 4   nox      506 non-null    float64
 5   rm       506 non-null    float64
 6   age      506 non-null    float64
 7   dis      506 non-null    float64
 8   rad      506 non-null    int64  
 9   tax      506 non-null    int64  
 10  ptratio  506 non-null    float64
 11  b        506 non-null    float64
 12  lstat    506 non-null    float64
 13  medv     506 non-null    float64
dtypes: float64(11), int64(3)
memory usage: 55.5 KB
```

Is There Any Null Value

Double-click (or enter) to edit

```
df.isnull().sum()
```

```
crim       0
zn         0
indus      0
chas       0
nox        0
rm         0
age        0
dis        0
rad        0
tax        0
ptratio    0
b          0
lstat      0
medv       0
dtype: int64
```

Is There Any Duplicate Value

```
df.duplicated().sum()
```

```
0
```

Mathematical OverView

```
df.describe()
```

	crim	zn	indus	chas	nox	rm	ag
count	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.0000(
mean	3.613524	11.363636	11.136779	0.069170	0.554695	6.284634	68.5749(
std	8.601545	23.322453	6.860353	0.253994	0.115878	0.702617	28.1488(
min	0.006320	0.000000	0.460000	0.000000	0.385000	3.561000	2.9000(
25%	0.082045	0.000000	5.190000	0.000000	0.449000	5.885500	45.0250(
50%	0.256510	0.000000	9.690000	0.000000	0.538000	6.208500	77.5000(
75%	3.677083	12.500000	18.100000	0.000000	0.624000	6.623500	94.0750(
max	88.976200	100.000000	27.740000	1.000000	0.871000	8.780000	100.0000(

Data Manipulation

Add a new column to the DataFrame that categorizes the 'medv' (median value of owner-occupied homes) column into 'Low', 'Medium', and 'High'.

```
df["medv Status"]=pd.cut(df["medv"],bins=[0, 20, 35, 50],labels=["Low","Medium","High"])
df
```

	crim	zn	indus	chas	nox	rm	age	dis	rad	tax	ptratio	b	lstat	medv	medv Status
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	4.98	24.0	Medium
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9.14	21.6	Medium
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03	34.7	Medium
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94	33.4	Medium
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	5.33	36.2	High
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
501	0.06263	0.0	11.93	0	0.573	6.593	69.1	2.4786	1	273	21.0	391.99	9.67	22.4	Medium
502	0.04527	0.0	11.93	0	0.573	6.120	76.7	2.2875	1	273	21.0	396.90	9.08	20.6	Medium
503	0.06076	0.0	11.93	0	0.573	6.976	91.0	2.1675	1	273	21.0	396.90	5.64	23.9	Medium
504	0.10959	0.0	11.93	0	0.573	6.794	89.3	2.3889	1	273	21.0	393.45	6.48	22.0	Medium
505	0.04741	0.0	11.93	0	0.573	6.030	80.8	2.5050	1	273	21.0	396.90	7.88	11.9	Low

506 rows × 15 columns

Replace the 'chas' column (Charles River dummy variable) with 'Yes' if chas = 1 and 'No' if chas = 0.

```
df["chas"]=df["chas"].map({1:"Yes",0:"No"})
df
```

	crim	zn	indus	chas	nox	rm	age	dis	rad	tax	ptratio	b	lstat	medv	medv Status
0	0.00632	18.0	2.31	No	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	4.98	24.0	Medium
1	0.02731	0.0	7.07	No	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9.14	21.6	Medium
2	0.02729	0.0	7.07	No	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03	34.7	Medium
3	0.03237	0.0	2.18	No	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94	33.4	Medium
4	0.06905	0.0	2.18	No	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	5.33	36.2	High
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
501	0.06263	0.0	11.93	No	0.573	6.593	69.1	2.4786	1	273	21.0	391.99	9.67	22.4	Medium
502	0.04527	0.0	11.93	No	0.573	6.120	76.7	2.2875	1	273	21.0	396.90	9.08	20.6	Medium
503	0.06076	0.0	11.93	No	0.573	6.976	91.0	2.1675	1	273	21.0	396.90	5.64	23.9	Medium
504	0.10959	0.0	11.93	No	0.573	6.794	89.3	2.3889	1	273	21.0	393.45	6.48	22.0	Medium
505	0.04741	0.0	11.93	No	0.573	6.030	80.8	2.5050	1	273	21.0	396.90	7.88	11.9	Low

506 rows × 15 columns

Double-click (or enter) to edit

Rename the column 'rm' to 'rooms'.

```
df=df.rename(columns={"rm":"rooms"})
df.head()
```

	crim	zn	indus	chas	nox	rooms	age	dis	rad	tax	ptratio	b	ls
0	0.00632	18.0	2.31	No	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	4
1	0.02731	0.0	7.07	No	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9
2	0.02729	0.0	7.07	No	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4
3	0.03237	0.0	2.18	No	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2

Data Analysis

Find out the average number of rooms ('rooms' column) per dwelling for each category of 'medv' (Low, Medium, High).

```
category=df.groupby(['medv Status'])
category=category[['rooms']].mean()
category
```

medv Status	rooms
Low	5.914721
Medium	6.371259
High	7.503000

Find out the percentage of houses that bound the Charles River.

```
c=df["chas"].value_counts(normalize=True)*100
c
```

```
No     93.083004
Yes     6.916996
Name: chas, dtype: float64
```

NumPy Operations

Convert the 'age' column (proportion of owner-occupied units built prior to 1940) from the DataFrame to a NumPy array and perform the following operations:

o Compute the mean and standard deviation.

o Normalize the array.

```
v=df['age']
np.array(v)
```

```
array([ 65.2,  78.9,  61.1,  45.8,  54.2,  58.7,  66.6,  96.1, 100. ,
        85.9,  94.3,  82.9,  39. ,  61.8,  84.5,  56.5,  29.3,  81.7,
        36.6,  69.5,  98.1,  89.2,  91.7, 100. ,  94.1,  85.7,  90.3,
        88.8,  94.4,  87.3,  94.1, 100. ,  82. ,  95. ,  96.9,  68.2,
        61.4,  41.5,  30.2,  21.8,  15.8,   2.9,   6.6,   6.5,  40. ,
        33.8,  33.3,  85.5,  95.3,  62. ,  45.7,  63. ,  21.1,  21.4,
        47.6,  21.9,  35.7,  40.5,  29.2,  47.2,  66.2,  93.4,  67.8,
        43.4,  59.5,  17.8,  31.1,  21.4,  36.8,  33. ,   6.6,  17.5,
         7.8,   6.2,   6. ,  45. ,  74.5,  45.8,  53.7,  36.6,  33.5,
        70.4,  32.2,  46.7,  48. ,  56.1,  45.1,  56.8,  86.3,  63.1,
        66.1,  73.9,  53.6,  28.9,  77.3,  57.8,  69.6,  76. ,  36.9,
        62.5,  79.9,  71.3,  85.4,  87.4,  90. ,  96.7,  91.9,  85.2,
        97.1,  91.2,  54.4,  81.6,  92.9,  95.4,  84.2,  88.2,  72.5,
        82.6,  73.1,  65.2,  69.7,  84.1,  92.9,  97. ,  95.8,  88.4,
        95.6,  96. ,  98.8,  94.7,  98.9,  97.7,  97.9,  95.4,  98.4,
        98.2,  93.5,  98.4,  98.2,  97.9,  93.6, 100. , 100. , 100. ,
        97.8, 100. , 100. ,  95.7,  93.8,  94.9,  97.3, 100. ,  88. ,
        98.5,  96. ,  82.6,  94. ,  97.4, 100. , 100. ,  92.6,  90.8,
        98.2,  93.9,  91.8,  93. ,  96.2,  79.2,  96.1,  95.2,  94.6,
        97.3,  88.5,  84.1,  68.7,  33.1,  47.2,  73.4,  74.4,  58.4,
        83.3,  62.2,  92.2,  95.6,  89.8,  68.8,  53.6,  41.1,  29.1,
        38.9,  21.5,  30.8,  26.3,   9.9,  18.8,  32. ,  34.1,  36.6,
        38.3,  15.3,  13.9,  38.4,  15.7,  33.2,  31.9,  22.3,  52.5,
        72.7,  59.1, 100. ,  92.1,  88.6,  53.8,  32.3,   9.8,  42.4,
        56. ,  85.1,  93.8,  92.4,  88.5,  91.3,  77.7,  80.8,  78.3,
        83. ,  86.5,  79.9,  17. ,  21.4,  68.1,  76.9,  73.3,  70.4,
        66.5,  61.5,  76.5,  71.6,  18.5,  42.2,  54.3,  65.1,  52.9,
         7.8,  76.5,  70.2,  34.9,  79.2,  49.1,  17.5,  13. ,   8.9,
         6.8,   8.4,  32. ,  19.1,  34.2,  86.9, 100. , 100. ,  81.8,
        89.4,  91.5,  94.5,  91.6,  62.8,  84.6,  67. ,  52.6,  61.5,
        42.1,  16.3,  58.7,  51.8,  32.9,  42.8,  49. ,  27.6,  32.1,
        32.2,  64.5,  37.2,  49.7,  24.8,  20.8,  31.9,  31.5,  31.3,
        45.6,  22.9,  27.9,  27.7,  23.4,  18.4,  42.3,  31.1,  51. ,
        58. ,  20.1,  10. ,  47.4,  40.4,  18.4,  17.7,  41.1,  58.1,
        71.9,  70.3,  82.5,  76.7,  37.8,  52.8,  90.4,  82.8,  87.3,
        77.7,  83.2,  71.7,  67.2,  58.8,  52.3,  54.3,  49.9,  74.3,
        40.1,  14.7,  28.9,  43.7,  25.8,  17.2,  32.2,  28.4,  23.3,
        38.1,  38.5,  34.5,  46.3,  59.6,  37.3,  45.4,  58.5,  49.3,
        59.7,  56.4,  28.1,  48.5,  52.3,  27.7,  29.7,  34.5,  44.4,
        35.9,  18.5,  36.1,  21.9,  19.5,  97.4,  91. ,  83.4,  81.3,
        88. ,  91.1,  96.2,  89. ,  82.9,  87.9,  91.4, 100. , 100. ,
        96.8,  97.5, 100. ,  89.6, 100. , 100. ,  97.9,  93.3,  98.8,
```

```
 96.2, 100. ,  91.9,  99.1, 100. , 100. ,  91.2,  98.1, 100. ,
 89.5, 100. ,  98.9,  97. ,  82.5,  97. ,  92.6,  94.7,  98.8,
 96. ,  98.9, 100. ,  77.8, 100. , 100. , 100. ,  96. ,  85.4,
100. , 100. , 100. ,  97.9, 100. , 100. , 100. , 100. , 100. ,
100. , 100. ,  90.8,  89.1, 100. ,  76.5, 100. ,  95.3,  87.6,
 85.1,  70.6,  95.4,  59.7,  78.7,  78.1,  95.6,  86.1,  94.3,
 74.8,  87.9,  95. ,  94.6,  93.3, 100. ,  87.9,  93.9,  92.4,
 97.2, 100. , 100. ,  96.6,  94.8,  96.4,  96.6,  98.7,  98.3,
 92.6,  98.2,  91.8,  99.3,  94.1,  86.5,  87.9,  80.3,  83.7,
 84.4,  90. ,  88.4,  83. ,  89.9,  65.4,  48.2,  84.7,  94.5,
 71. ,  56.7,  84. ,  90.7,  75. ,  67.6,  95.4,  97.4,  93.6,
 97.3,  96.7,  88. ,  64.7,  74.9,  77. ,  40.3,  41.9,  51.9,
 79.8,  53.2,  92.7,  98.3,  98. ,  98.8,  83.5,  54. ,  42.6,
 28.8,  72.9,  70.6,  65.3,  73.5,  79.7,  69.1,  76.7,  91. ,
```

Mathematical Operations

```
np.mean(v)
```

```
68.57490118577076
```

```
np.std(v)
```

```
28.121032570236867
```

Double-click (or enter) to edit

min-max normalization

```
normalized_array = (v - v.min(axis=0)) / (v.max(axis=0) - v.min(axis=0))
print(normalized_array)
```

```
0      0.641607
1      0.782698
2      0.599382
3      0.441813
4      0.528321
         ...
501    0.681771
502    0.760041
503    0.907312
504    0.889804
505    0.802266
Name: age, Length: 506, dtype: float64
```

z-score normalization

```
normalized_array1 = (v - v.mean(axis=0)) / (v.std(axis=0))
print(normalized_array1)
```

```
0     -0.119895
1      0.366803
2     -0.265549
3     -0.809088
4     -0.510674
         ...
501    0.018654
502    0.288648
503    0.796661
504    0.736268
505    0.434302
Name: age, Length: 506, dtype: float64
```