**Lab 3: Introduction to Python Libraries for Machine Learning**

**Numpy Exercises**

**1.Basic Array Creation & Manipulation**

• Create a 1D array of numbers from 1 to 20.

• Create a 3×4 matrix of ones and reshape it to 4×3.

• Create a 5×5 identity matrix.

• Generate 15 equally spaced numbers between 5 and 50.

• Generate a 4×4 matrix of random integers between 1 and 100.

|  |
| --- |
| In [1]:  import numpy as np |

|  |
| --- |
| In [2]:  #Create a 1D array of numbers from 1 to 20.  arr = np.arange(1,21)  print(arr) |

[ 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20]

|  |
| --- |
| In [3]:  #Create a 3×4 matrix of ones and reshape it to 4×3.  arr = np.ones((3,4))  print(arr)  arr = arr.reshape(4,3)  print(arr) |

[[1. 1. 1. 1.]

[1. 1. 1. 1.]

[1. 1. 1. 1.]]

[[1. 1. 1.]

[1. 1. 1.]

[1. 1. 1.]

[1. 1. 1.]]

|  |
| --- |
| In [4]:  # Create a 5×5 identity matrix.  arr = np.eye(5)  print(arr) |

[[1. 0. 0. 0. 0.]

[0. 1. 0. 0. 0.]

[0. 0. 1. 0. 0.]

[0. 0. 0. 1. 0.]

[0. 0. 0. 0. 1.]]

|  |
| --- |
| In [5]:  #Generate 15 equally spaced numbers between 5 and 50.  arr = np.linspace(5,50,15,)  print(arr) |

[ 5. 8.21428571 11.42857143 14.64285714 17.85714286 21.07142857

24.28571429 27.5 30.71428571 33.92857143 37.14285714 40.35714286

43.57142857 46.78571429 50. ]

|  |
| --- |
| In [6]:  #Generate a 4×4 matrix of random integers between 1 and 100  arr = np.random.randint(1,101,(4,4))  print(arr) |

[[10 19 13 67]

[68 40 60 30]

[69 58 10 49]

[35 53 70 33]]

**2.Indexing, Slicing, and Broadcasting**

• Create a 3×3 matrix of random integers between 1 and 100.

• Extract: First row, Second column, Center element.

• Replace all values greater than 50 in a matrix with 999.

• Multiply a 1D array of size 5 (random integers between 1 to 10) by 10 using

broadcasting

|  |
| --- |
| In [7]:  #Create a 3×3 matrix of random integers between 1 and 100.  arr = np.random.randint(1,101,(3,3))  print(arr) |

[[63 80 27]

[75 35 69]

[80 63 26]]

|  |
| --- |
| In [8]:  #Extract: First row, Second column, Center element.  print(arr[0])  print(arr[:,1:2])  print(arr[1,1]) |

[63 80 27]

[[80]

[35]

[63]]

35

|  |
| --- |
| In [9]:  #Replace all values greater than 50 in a matrix with 999.  arr[arr > 50] = 999  print(arr) |

[[999 999 27]

[999 35 999]

[999 999 26]]

|  |
| --- |
| In [10]:  #Multiply a 1D array of size 5 (random integers between 1 to 10) by 10 using broadcasting  arr1 = np.random.randint(1,11,5)  print(arr1)  arr2 = 10  print(arr1\*arr2) |

[5 7 1 8 4]

[50 70 10 80 40]

**3.Mathematical and Statistical Operations**

• Create a 3×3 matrix of random integers between 1 and 100

• Compute sum, mean, median, std, var, min, and max of the above array.

• Normalize a 1D array of size 5 (random integers between 1 to 10) to scale values

between 0 and 1

|  |
| --- |
| In [11]:  #Create a 3×3 matrix of random integers between 1 and 100.  arr = np.random.randint(1,101,(3,3))  print(arr) |

[[84 65 69]

[39 43 99]

[26 19 38]]

|  |
| --- |
| In [12]:  # Compute sum, mean, median, std, var, min, and max of the above array.  print(np.sum(arr))  print(np.mean(arr))  print(np.median(arr))  print(np.std(arr))  print(np.var(arr))  print(np.min(arr))  print(np.max(arr)) |

482

53.55555555555556

43.0

25.56086901283597

653.358024691358

19

99

|  |
| --- |
| In [13]:  #Normalize a 1D array of size 5 (random integers between 1 to 10) to scale values between 0 and 1  arr2 = np.random.randint(1,11,5)  print(arr2)  arr2 = (arr2 - np.min(arr2)) / (np.max(arr2) - np.min(arr2))  print(arr2) |

[10 4 4 6 3]

[1. 0.14285714 0.14285714 0.42857143 0. ]

**4.NumPy Matrix Operations and Linear Algebra**

• Generate Following two NumPy matrix import numpy as np

A = ([4, 2], [1, 3])

B = ([2, 0], [1, 5])

• Find matrix multiplication of A and B.

• Find dot product of A and B.

• Find element wise addition/ subtraction/ multiplications/ division of A and B

• Transpose matrix A.

• Compute determinant of A.

• Compute inverse of A (if possible).

• Find eigenvalues and eigenvectors.

• Solve the system of equations:

2x + y = 8

3x + 4y = 18

|  |
| --- |
| In [14]:  #Generate Following two NumPy matrix import numpy as np  A = np.array([[4, 2], [1, 3]])  B = np.array([[2, 1], [1, 5]])  # print(A)  # print(B)  #Find matrix multiplication of A and B.  print("Matrix Multiplication:")  print(A\*B)  # Find dot product of A and B.  print("\nDot Product:")  print(np.dot(A, B)) |

Matrix Multiplication:

[[ 8 2]

[ 1 15]]

Dot Product:

[[10 14]

[ 5 16]]

|  |
| --- |
| In [15]:  #Find element wise addition/ subtraction/ multiplications/ division of A and B  print("\nElement Wise Addition:")  print(A+B)  print("\nElement Wise Subtraction:")  print(A-B)  print("\nElement Wise Multiplication:")  print(A\*B)  print("\nElement Wise Division:")  print(A/B) |

Element Wise Addition:

[[6 3]

[2 8]]

Element Wise Subtraction:

[[ 2 1]

[ 0 -2]]

Element Wise Multiplication:

[[ 8 2]

[ 1 15]]

Element Wise Division:

[[2. 2. ]

[1. 0.6]]

|  |
| --- |
| In [16]:  #Transpose matrix A.  print(A.T) |

[[4 1]

[2 3]]

|  |
| --- |
| In [17]:  #Compute determinant of A  print(np.linalg.det(A)) |

10.000000000000002

|  |
| --- |
| In [18]:  #Compute inverse of A (if possible)  print(np.linalg.inv(A)) |

[[ 0.3 -0.2]

[-0.1 0.4]]

|  |
| --- |
| In [19]:  #Find eigenvalues and eigenvectors.  eigenvalues, eigenvectors = np.linalg.eig(A)  print("Eigenvalues:")  print(eigenvalues)  print("\nEigenvectors:")  print(eigenvectors) |

Eigenvalues:

[5. 2.]

Eigenvectors:

[[ 0.89442719 -0.70710678]

[ 0.4472136 0.70710678]]

|  |
| --- |
| In [20]:  # Solve the system of equations: 2x + y = 8, 3x + 4y = 18  A = np.array([[2, 1], [3, 4]])  B = np.array([8, 18])  solution = np.linalg.solve(A, B)  print("Solution (x, y):", solution) |

Solution (x, y): [2.8 2.4]

**Pandas Exercises**

**1.Series & DataFrame Basics**

• Given the following list of marks: [78, 85, 92, 70, 66]

• Create a Pandas Series and assign the following student names as indices: ['Amit', 'Bhavna', 'Chetan', 'Divya', 'Esha']

• Display the Series.

|  |
| --- |
| In [21]:  import pandas as pd |

|  |
| --- |
| In [22]:  marks = [78, 85, 92, 70, 66]  names = ['Amit', 'Bhavna', 'Chetan', 'Divya', 'Esha'] |

|  |
| --- |
| In [23]:  arr = pd.Series(marks, index = names)  print(arr) |

Amit 78

Bhavna 85

Chetan 92

Divya 70

Esha 66

dtype: int64

Using the following dictionary:

data = {

'Name': ['Amit', 'Bhavna', 'Chetan', 'Divya', 'Esha'],

'Gender': ['Male', 'Female', 'Male', 'Female', 'Female'],

'Math': [78, 85, 92, 70, 66],

'Science': [88, 79, 95, 72, 60]

}

Create a Pandas DataFrame and display:

• The full DataFrame

• The column names

• The shape of the DataFrame

|  |
| --- |
| In [24]:  data = {  'Name': ['Amit', 'Bhavna', 'Chetan', 'Divya', 'Esha'],  'Gender': ['Male', 'Female', 'Male', 'Female', 'Female'],  'Math': [78, 85, 92, 70, 66],  'Science': [88, 79, 95, 72, 60]  } |

|  |
| --- |
| In [25]:  df = pd.DataFrame(data)  print(df) |

Name Gender Math Science

0 Amit Male 78 88

1 Bhavna Female 85 79

2 Chetan Male 92 95

3 Divya Female 70 72

4 Esha Female 66 60

|  |
| --- |
| In [26]:  print(df.columns)  print(df.shape) |

Index(['Name', 'Gender', 'Math', 'Science'], dtype='object')

(5, 4)

**2.Data Exploration**

• Load the dataset from this url "https://archive.ics.uci.edu/ml/machine-learning-databases/autos/imports-85.data"

• Assign names of columns:

["symboling","normalized-losses","make","fuel-type","aspiration", "num-of-doors","body-style", "drive-wheels","engine-location","wheel-base","length","width","height","curb-weight","engine-type","num-of-cylinders", "engine-size","fuel-system","bore","stroke","compression-ratio","horsepower",

"peak-rpm","city-mpg","highway-mpg","price"]

• Display .shape, .columns, .info(), and .describe().

• Display only ","width","height","curb-weight","engine-type" columns.

• Display car details which have num-of-doors = four

|  |
| --- |
| In [27]:  data\_url = "https://archive.ics.uci.edu/ml/machine-learning-databases/autos/imports-85.data"  column\_names = ["symboling","normalized-losses","make","fuel-type","aspiration", "num-of-doors","body-style", "drive-wheels","engine-location","wheel-base","length","width","height","curb-weight","engine-type","num-of-cylinders", "engine-size","fuel-system","bore","stroke","compression-ratio","horsepower",  "peak-rpm","city-mpg","highway-mpg","price"] |

|  |
| --- |
| In [28]:  df\_data = pd.read\_csv(data\_url, names = column\_names, delimiter= ",", na\_values= "?") |

|  |
| --- |
| In [29]:  print(df\_data.head(5)) |

symboling normalized-losses make fuel-type aspiration \

0 3 NaN alfa-romero gas std

1 3 NaN alfa-romero gas std

2 1 NaN alfa-romero gas std

3 2 164.0 audi gas std

4 2 164.0 audi gas std

num-of-doors body-style drive-wheels engine-location wheel-base ... \

0 two convertible rwd front 88.6 ...

1 two convertible rwd front 88.6 ...

2 two hatchback rwd front 94.5 ...

3 four sedan fwd front 99.8 ...

4 four sedan 4wd front 99.4 ...

engine-size fuel-system bore stroke compression-ratio horsepower \

0 130 mpfi 3.47 2.68 9.0 111.0

1 130 mpfi 3.47 2.68 9.0 111.0

2 152 mpfi 2.68 3.47 9.0 154.0

3 109 mpfi 3.19 3.40 10.0 102.0

4 136 mpfi 3.19 3.40 8.0 115.0

peak-rpm city-mpg highway-mpg price

0 5000.0 21 27 13495.0

1 5000.0 21 27 16500.0

2 5000.0 19 26 16500.0

3 5500.0 24 30 13950.0

4 5500.0 18 22 17450.0

[5 rows x 26 columns]

|  |
| --- |
| In [30]:  print(df\_data.shape)  print(df\_data.columns)  print(df\_data.info())  print(df\_data.describe()) |

(205, 26)

Index(['symboling', 'normalized-losses', 'make', 'fuel-type', 'aspiration',

'num-of-doors', 'body-style', 'drive-wheels', 'engine-location',

'wheel-base', 'length', 'width', 'height', 'curb-weight', 'engine-type',

'num-of-cylinders', 'engine-size', 'fuel-system', 'bore', 'stroke',

'compression-ratio', 'horsepower', 'peak-rpm', 'city-mpg',

'highway-mpg', 'price'],

dtype='object')

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 205 entries, 0 to 204

Data columns (total 26 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 symboling 205 non-null int64

1 normalized-losses 164 non-null float64

2 make 205 non-null object

3 fuel-type 205 non-null object

4 aspiration 205 non-null object

5 num-of-doors 203 non-null object

6 body-style 205 non-null object

7 drive-wheels 205 non-null object

8 engine-location 205 non-null object

9 wheel-base 205 non-null float64

10 length 205 non-null float64

11 width 205 non-null float64

12 height 205 non-null float64

13 curb-weight 205 non-null int64

14 engine-type 205 non-null object

15 num-of-cylinders 205 non-null object

16 engine-size 205 non-null int64

17 fuel-system 205 non-null object

18 bore 201 non-null float64

19 stroke 201 non-null float64

20 compression-ratio 205 non-null float64

21 horsepower 203 non-null float64

22 peak-rpm 203 non-null float64

23 city-mpg 205 non-null int64

24 highway-mpg 205 non-null int64

25 price 201 non-null float64

dtypes: float64(11), int64(5), object(10)

memory usage: 41.8+ KB

None

symboling normalized-losses wheel-base length width \

count 205.000000 164.000000 205.000000 205.000000 205.000000

mean 0.834146 122.000000 98.756585 174.049268 65.907805

std 1.245307 35.442168 6.021776 12.337289 2.145204

min -2.000000 65.000000 86.600000 141.100000 60.300000

25% 0.000000 94.000000 94.500000 166.300000 64.100000

50% 1.000000 115.000000 97.000000 173.200000 65.500000

75% 2.000000 150.000000 102.400000 183.100000 66.900000

max 3.000000 256.000000 120.900000 208.100000 72.300000

height curb-weight engine-size bore stroke \

count 205.000000 205.000000 205.000000 201.000000 201.000000

mean 53.724878 2555.565854 126.907317 3.329751 3.255423

std 2.443522 520.680204 41.642693 0.273539 0.316717

min 47.800000 1488.000000 61.000000 2.540000 2.070000

25% 52.000000 2145.000000 97.000000 3.150000 3.110000

50% 54.100000 2414.000000 120.000000 3.310000 3.290000

75% 55.500000 2935.000000 141.000000 3.590000 3.410000

max 59.800000 4066.000000 326.000000 3.940000 4.170000

compression-ratio horsepower peak-rpm city-mpg highway-mpg \

count 205.000000 203.000000 203.000000 205.000000 205.000000

mean 10.142537 104.256158 5125.369458 25.219512 30.751220

std 3.972040 39.714369 479.334560 6.542142 6.886443

min 7.000000 48.000000 4150.000000 13.000000 16.000000

25% 8.600000 70.000000 4800.000000 19.000000 25.000000

50% 9.000000 95.000000 5200.000000 24.000000 30.000000

75% 9.400000 116.000000 5500.000000 30.000000 34.000000

max 23.000000 288.000000 6600.000000 49.000000 54.000000

price

count 201.000000

mean 13207.129353

std 7947.066342

min 5118.000000

25% 7775.000000

50% 10295.000000

75% 16500.000000

max 45400.000000

|  |
| --- |
| In [31]:  print(df\_data.head(5)[["width","height","curb-weight","engine-type"]]) |

width height curb-weight engine-type

0 64.1 48.8 2548 dohc

1 64.1 48.8 2548 dohc

2 65.5 52.4 2823 ohcv

3 66.2 54.3 2337 ohc

4 66.4 54.3 2824 ohc

|  |
| --- |
| In [32]:  print(df\_data[df\_data["num-of-doors"] == "four"].head(5)) |

symboling normalized-losses make fuel-type aspiration num-of-doors \

3 2 164.0 audi gas std four

4 2 164.0 audi gas std four

6 1 158.0 audi gas std four

7 1 NaN audi gas std four

8 1 158.0 audi gas turbo four

body-style drive-wheels engine-location wheel-base ... engine-size \

3 sedan fwd front 99.8 ... 109

4 sedan 4wd front 99.4 ... 136

6 sedan fwd front 105.8 ... 136

7 wagon fwd front 105.8 ... 136

8 sedan fwd front 105.8 ... 131

fuel-system bore stroke compression-ratio horsepower peak-rpm city-mpg \

3 mpfi 3.19 3.4 10.0 102.0 5500.0 24

4 mpfi 3.19 3.4 8.0 115.0 5500.0 18

6 mpfi 3.19 3.4 8.5 110.0 5500.0 19

7 mpfi 3.19 3.4 8.5 110.0 5500.0 19

8 mpfi 3.13 3.4 8.3 140.0 5500.0 17

highway-mpg price

3 30 13950.0

4 22 17450.0

6 25 17710.0

7 25 18920.0

8 20 23875.0

[5 rows x 26 columns]

**3.Missing values handling**

• Missing value is represented by ‘?’ in this dataset. Replace it with NULL.

• Check how many missing values are there in each attribute.

• Replace missing values of "normalized-losses", "stroke", "bore", "horsepower" with

mean.

• Drop all the rows which has missing value in attribute “price”.\*\*bold text\*\*

• Replace missing values of “num-of-doors” with mode.

• Replace all other missing values with median.

|  |
| --- |
| In [33]:  #Missing value is represented by ‘?’ in this dataset. Replace it with NULL.  print(df\_data.head(7).fillna(value = "NULL")) |

symboling normalized-losses make fuel-type aspiration num-of-doors \

0 3 NULL alfa-romero gas std two

1 3 NULL alfa-romero gas std two

2 1 NULL alfa-romero gas std two

3 2 164.0 audi gas std four

4 2 164.0 audi gas std four

5 2 NULL audi gas std two

6 1 158.0 audi gas std four

body-style drive-wheels engine-location wheel-base ... engine-size \

0 convertible rwd front 88.6 ... 130

1 convertible rwd front 88.6 ... 130

2 hatchback rwd front 94.5 ... 152

3 sedan fwd front 99.8 ... 109

4 sedan 4wd front 99.4 ... 136

5 sedan fwd front 99.8 ... 136

6 sedan fwd front 105.8 ... 136

fuel-system bore stroke compression-ratio horsepower peak-rpm city-mpg \

0 mpfi 3.47 2.68 9.0 111.0 5000.0 21

1 mpfi 3.47 2.68 9.0 111.0 5000.0 21

2 mpfi 2.68 3.47 9.0 154.0 5000.0 19

3 mpfi 3.19 3.40 10.0 102.0 5500.0 24

4 mpfi 3.19 3.40 8.0 115.0 5500.0 18

5 mpfi 3.19 3.40 8.5 110.0 5500.0 19

6 mpfi 3.19 3.40 8.5 110.0 5500.0 19

highway-mpg price

0 27 13495.0

1 27 16500.0

2 26 16500.0

3 30 13950.0

4 22 17450.0

5 25 15250.0

6 25 17710.0

[7 rows x 26 columns]

|  |
| --- |
| In [34]:  #Check how many missing values are there in each attribute.  print(df\_data.isnull().sum()) |

symboling 0

normalized-losses 41

make 0

fuel-type 0

aspiration 0

num-of-doors 2

body-style 0

drive-wheels 0

engine-location 0

wheel-base 0

length 0

width 0

height 0

curb-weight 0

engine-type 0

num-of-cylinders 0

engine-size 0

fuel-system 0

bore 4

stroke 4

compression-ratio 0

horsepower 2

peak-rpm 2

city-mpg 0

highway-mpg 0

price 4

dtype: int64

|  |
| --- |
| In [35]:  #Replace missing values of "normalized-losses", "stroke", "bore", "horsepower" with mean.  df\_data["normalized-losses"] = df\_data["normalized-losses"].fillna(df\_data["normalized-losses"].mean())  df\_data["stroke"] = df\_data["stroke"].fillna(df\_data["stroke"].mean())  df\_data["bore"] = df\_data["bore"].fillna(df\_data["bore"].mean())  df\_data["horsepower"] = df\_data["horsepower"].fillna(df\_data["horsepower"].mean())  print(df\_data.head(5)) |

symboling normalized-losses make fuel-type aspiration \

0 3 122.0 alfa-romero gas std

1 3 122.0 alfa-romero gas std

2 1 122.0 alfa-romero gas std

3 2 164.0 audi gas std

4 2 164.0 audi gas std

num-of-doors body-style drive-wheels engine-location wheel-base ... \

0 two convertible rwd front 88.6 ...

1 two convertible rwd front 88.6 ...

2 two hatchback rwd front 94.5 ...

3 four sedan fwd front 99.8 ...

4 four sedan 4wd front 99.4 ...

engine-size fuel-system bore stroke compression-ratio horsepower \

0 130 mpfi 3.47 2.68 9.0 111.0

1 130 mpfi 3.47 2.68 9.0 111.0

2 152 mpfi 2.68 3.47 9.0 154.0

3 109 mpfi 3.19 3.40 10.0 102.0

4 136 mpfi 3.19 3.40 8.0 115.0

peak-rpm city-mpg highway-mpg price

0 5000.0 21 27 13495.0

1 5000.0 21 27 16500.0

2 5000.0 19 26 16500.0

3 5500.0 24 30 13950.0

4 5500.0 18 22 17450.0

[5 rows x 26 columns]

|  |
| --- |
| In [36]:  #Drop all the rows which has missing value in attribute “price”  print(df\_data[df\_data["price"].isnull()])  df\_data = df\_data.dropna(subset=['price']) |

symboling normalized-losses make fuel-type aspiration num-of-doors \

9 0 122.0 audi gas turbo two

44 1 122.0 isuzu gas std two

45 0 122.0 isuzu gas std four

129 1 122.0 porsche gas std two

body-style drive-wheels engine-location wheel-base ... engine-size \

9 hatchback 4wd front 99.5 ... 131

44 sedan fwd front 94.5 ... 90

45 sedan fwd front 94.5 ... 90

129 hatchback rwd front 98.4 ... 203

fuel-system bore stroke compression-ratio horsepower peak-rpm \

9 mpfi 3.13 3.40 7.0 160.0 5500.0

44 2bbl 3.03 3.11 9.6 70.0 5400.0

45 2bbl 3.03 3.11 9.6 70.0 5400.0

129 mpfi 3.94 3.11 10.0 288.0 5750.0

city-mpg highway-mpg price

9 16 22 NaN

44 38 43 NaN

45 38 43 NaN

129 17 28 NaN

[4 rows x 26 columns]

|  |
| --- |
| In [37]:  # Replace missing values of “num-of-doors” with mode.  df\_data["num-of-doors"] = df\_data["num-of-doors"].fillna(df\_data["num-of-doors"].mode()[0])  print(df\_data.head(5)) |

symboling normalized-losses make fuel-type aspiration \

0 3 122.0 alfa-romero gas std

1 3 122.0 alfa-romero gas std

2 1 122.0 alfa-romero gas std

3 2 164.0 audi gas std

4 2 164.0 audi gas std

num-of-doors body-style drive-wheels engine-location wheel-base ... \

0 two convertible rwd front 88.6 ...

1 two convertible rwd front 88.6 ...

2 two hatchback rwd front 94.5 ...

3 four sedan fwd front 99.8 ...

4 four sedan 4wd front 99.4 ...

engine-size fuel-system bore stroke compression-ratio horsepower \

0 130 mpfi 3.47 2.68 9.0 111.0

1 130 mpfi 3.47 2.68 9.0 111.0

2 152 mpfi 2.68 3.47 9.0 154.0

3 109 mpfi 3.19 3.40 10.0 102.0

4 136 mpfi 3.19 3.40 8.0 115.0

peak-rpm city-mpg highway-mpg price

0 5000.0 21 27 13495.0

1 5000.0 21 27 16500.0

2 5000.0 19 26 16500.0

3 5500.0 24 30 13950.0

4 5500.0 18 22 17450.0

[5 rows x 26 columns]

/tmp/ipython-input-61-1088669806.py:2: SettingWithCopyWarning:

A value is trying to be set on a copy of a slice from a DataFrame.

Try using .loc[row\_indexer,col\_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy

df\_data["num-of-doors"] = df\_data["num-of-doors"].fillna(df\_data["num-of-doors"].mode()[0])

|  |
| --- |
| In [38]:  # Replace all other missing values with median.  # peak-rpm has 2 missing values  df\_data["peak-rpm"] = df\_data["peak-rpm"].fillna(df\_data["peak-rpm"].median())  #no missing values  print(df\_data.isnull().sum()) |

symboling 0

normalized-losses 0

make 0

fuel-type 0

aspiration 0

num-of-doors 0

body-style 0

drive-wheels 0

engine-location 0

wheel-base 0

length 0

width 0

height 0

curb-weight 0

engine-type 0

num-of-cylinders 0

engine-size 0

fuel-system 0

bore 0

stroke 0

compression-ratio 0

horsepower 0

peak-rpm 0

city-mpg 0

highway-mpg 0

price 0

dtype: int64

/tmp/ipython-input-62-1935293583.py:3: SettingWithCopyWarning:

A value is trying to be set on a copy of a slice from a DataFrame.

Try using .loc[row\_indexer,col\_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy

df\_data["peak-rpm"] = df\_data["peak-rpm"].fillna(df\_data["peak-rpm"].median())

**4.Grouping, Sorting, and Aggregation**

• Group by “Fuel-type” and compute average price.

• In our dataset, the fuel consumption columns "city-mpg" and "highway-mpg" are

represented by mpg (miles per gallon) unit.

Assume we are developing an application in

a country that accept the fuel consumption with L/100km standard

We will need to apply data transformation to transform mpg into L/100km?

The formula for unit conversion is

L/100km = 235 / mpg

-Sort the DataFrame based on “price” in descending order.

|  |
| --- |
| In [39]:  # Group by “Fuel-type” and compute average price.  fuel\_price = df\_data.groupby('fuel-type')['price'].mean()  print(fuel\_price) |

fuel-type

diesel 15838.15000

gas 12916.40884

Name: price, dtype: float64

|  |
| --- |
| In [40]:  #convert mpg (miles per gallon) to L/100km standard for 2 rows "city-mpg" and "highway-mpg".  # conversion formula: L/100km = 235 / mpg |

|  |
| --- |
| In [41]:  # Convert mpg to L/100km  df\_data['city-L/100km'] = 235 / df\_data['city-mpg']  df\_data['highway-L/100km'] = 235 / df\_data['highway-mpg']  # Display the first 5 rows with the new columns  print(df\_data[['city-mpg', 'city-L/100km', 'highway-mpg', 'highway-L/100km']].head()) |

city-mpg city-L/100km highway-mpg highway-L/100km

0 21 11.190476 27 8.703704

1 21 11.190476 27 8.703704

2 19 12.368421 26 9.038462

3 24 9.791667 30 7.833333

4 18 13.055556 22 10.681818

|  |
| --- |
| In [42]:  #Sort the DataFrame based on “price” in descending order. |

|  |
| --- |
| In [43]:  # Sort the DataFrame based on “price” in descending order.  df\_data\_sorted = df\_data.sort\_values(by='price', ascending=False)  print(df\_data\_sorted["price"].head(7)) |

74 45400.0

16 41315.0

73 40960.0

128 37028.0

17 36880.0

49 36000.0

48 35550.0

Name: price, dtype: float64