# Assignment 1: Pandas Exercise 1

1. Importing library: pandas as pd

```
import pandas as pd
```

2. Importing mpg dataset as df

```
df=pd.read_csv('https://github.com/YBI-Foundation/Dataset/raw/main/MPG.csv')
```

3. Inspecting first 5 rows of mpg dataset

```
df.head()
```

| | mpg | cylinders | displacement | horsepower | weight | acceleration | model_year | origin | name |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 18.0 | 8 | 307.0 | 130.0 | 3504 | 12.0 | 70 | usa | chevrolet chevelle malibu |
| **1** | 15.0 | 8 | 350.0 | 165.0 | 3693 | 11.5 | 70 | usa | buick skylark 320 |
| **2** | 18.0 | 8 | 318.0 | 150.0 | 3436 | 11.0 | 70 | usa | plymouth satellite |
| **3** | 16.0 | 8 | 304.0 | 150.0 | 3433 | 12.0 | 70 | usa | amc rebel sst |
| **4** | 17.0 | 8 | 302.0 | 140.0 | 3449 | 10.5 | 70 | usa | ford torino |

4. Inspecting first 10 rows of mpg dataset

Saved successfully!

| | mpg | cylinders | displacement | horsepower | weight | acceleration | model_year | origin | name |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 18.0 | 8 | 307.0 | 130.0 | 3504 | 12.0 | 70 | usa | chevrolet chevelle malibu |
| **1** | 15.0 | 8 | 350.0 | 165.0 | 3693 | 11.5 | 70 | usa | buick skylark 320 |
| **2** | 18.0 | 8 | 318.0 | 150.0 | 3436 | 11.0 | 70 | usa | plymouth satellite |
| **3** | 16.0 | 8 | 304.0 | 150.0 | 3433 | 12.0 | 70 | usa | amc rebel sst |
| **4** | 17.0 | 8 | 302.0 | 140.0 | 3449 | 10.5 | 70 | usa | ford torino |
| **5** | 15.0 | 8 | 429.0 | 198.0 | 4341 | 10.0 | 70 | usa | ford galaxie 500 |
| **6** | 14.0 | 8 | 454.0 | 220.0 | 4354 | 9.0 | 70 | usa | chevrolet impala |
| **7** | 14.0 | 8 | 440.0 | 215.0 | 4312 | 8.5 | 70 | usa | plymouth fury iii |
| **8** | 14.0 | 8 | 455.0 | 225.0 | 4425 | 10.0 | 70 | usa | pontiac catalina |

5. Inspecting last 8 rows of mpg dataset

```
df.tail(8)
```

| | | | ment | horsepower | weight | acceleration | model_year | origin | name |
|---|---|---|---|---|---|---|---|---|---|
| **390** | 32.0 | 4 | 144.0 | 96.0 | 2665 | 13.9 | 82 | japan | toyota celica gt |
| **391** | 36.0 | 4 | 135.0 | 84.0 | 2370 | 13.0 | 82 | usa | dodge charger 2.2 |

6. Displaying all the rows in the mpg dataset

ford mustang

```
pd.options.display.max_rows=None
df
```

Saved successfully!

| | | | ment | horsepower | weight | acceleration | model_year | origin | name |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 18.0 | 8 | 307.0 | 130.0 | 3504 | 12.0 | 70 | usa | chevrolet chevelle malibu |
| **1** | 15.0 | 8 | 350.0 | 165.0 | 3693 | 11.5 | 70 | usa | buick skylark 320 |
| **2** | 18.0 | 8 | 318.0 | 150.0 | 3436 | 11.0 | 70 | usa | plymouth satellite |
| **3** | 16.0 | 8 | 304.0 | 150.0 | 3433 | 12.0 | 70 | usa | amc rebel sst |
| **4** | 17.0 | 8 | 302.0 | 140.0 | 3449 | 10.5 | 70 | usa | ford torino |
| **5** | 15.0 | 8 | 429.0 | 198.0 | 4341 | 10.0 | 70 | usa | ford galaxie 500 |
| **6** | 14.0 | 8 | 454.0 | 220.0 | 4354 | 9.0 | 70 | usa | chevrolet impala |
| **7** | 14.0 | 8 | 440.0 | 215.0 | 4312 | 8.5 | 70 | usa | plymouth fury iii |
| **8** | 14.0 | 8 | 455.0 | 225.0 | 4425 | 10.0 | 70 | usa | pontiac catalina |
| **9** | 15.0 | 8 | 390.0 | 190.0 | 3850 | 8.5 | 70 | usa | amc ambassador dpl |

7. Displaying all the required information about our 'df' dataframe

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 398 entries, 0 to 397
```

Saved successfully!

```
                                  Non-Null Count  Dtype
---  ------                       --------------  -----
 0   mpg                          398 non-null    float64
 1   cylinders                    398 non-null    int64
 2   displacement                 398 non-null    float64
 3   horsepower                   392 non-null    float64
 4   weight                       398 non-null    int64
 5   acceleration                 398 non-null    float64
 6   model_year                   398 non-null    int64
 7   origin                       398 non-null    object
 8   name                         398 non-null    object
dtypes: float64(4), int64(3), object(2)
memory usage: 28.1+ KB
```

18 27.0 4 97.0 88.0 2130 14.5 70 japan datsun

8. Displaying statistical summary of our 'df' dataframe

19 26.0 4 97.0 46.0 1835 20.5 70 europe 1131 deluxe

```
df.describe()
```

| | mpg | cylinders | displacement | horsepower | weight | acceleration | model_year |
|---|---|---|---|---|---|---|---|
| **count** | 398.000000 | 398.000000 | 398.000000 | 392.000000 | 398.000000 | 398.000000 | 398.000000 |
| **mean** | 23.514573 | 5.454774 | 193.425879 | 104.469388 | 2970.424623 | 15.568090 | 76.010050 |
| **std** | 7.815984 | 1.701004 | 104.269838 | 38.491160 | 846.841774 | 2.757689 | 3.697627 |
| **min** | 9.000000 | 3.000000 | 68.000000 | 46.000000 | 1613.000000 | 8.000000 | 70.000000 |
| **25%** | 17.500000 | 4.000000 | 104.250000 | 75.000000 | 2223.750000 | 13.825000 | 73.000000 |
| **50%** | 23.000000 | 4.000000 | 148.500000 | 93.500000 | 2803.500000 | 15.500000 | 76.000000 |
| **75%** | 29.000000 | 8.000000 | 262.000000 | 126.000000 | 3608.000000 | 17.175000 | 79.000000 |
| **max** | 46.600000 | 8.000000 | 455.000000 | 230.000000 | 5140.000000 | 24.800000 | 82.000000 |

ur 'df' dataframe including all(text) columns

Saved successfully!

```
df.describe(include='all')
```

| | mpg | cylinders | displacement | horsepower | weight | acceleration | model_year | origin |
|---|---|---|---|---|---|---|---|---|
| **count** | 398.000000 | 398.000000 | 398.000000 | 392.000000 | 398.000000 | 398.000000 | 398.000000 | 398 |
| **unique** | NaN | NaN | NaN | NaN | NaN | NaN | NaN | 3 |
| **top** | NaN | NaN | NaN | NaN | NaN | NaN | NaN | usa |
| **freq** | NaN | NaN | NaN | NaN | NaN | NaN | NaN | 249 |
| **mean** | 23.514573 | 5.454774 | 193.425879 | 104.469388 | 2970.424623 | 15.568090 | 76.010050 | NaN |
| **std** | 7.815984 | 1.701004 | 104.269838 | 38.491160 | 846.841774 | 2.757689 | 3.697627 | NaN |
| **min** | 9.000000 | 3.000000 | 68.000000 | 46.000000 | 1613.000000 | 8.000000 | 70.000000 | NaN |
| **25%** | 17.500000 | 4.000000 | 104.250000 | 75.000000 | 2223.750000 | 13.825000 | 73.000000 | NaN |
| **50%** | 23.000000 | 4.000000 | 148.500000 | 93.500000 | 2803.500000 | 15.500000 | 76.000000 | NaN |
| **75%** | 29.000000 | 8.000000 | 262.000000 | 126.000000 | 3608.000000 | 17.175000 | 79.000000 | NaN |
| **max** | 46.600000 | 8.000000 | 455.000000 | 230.000000 | 5140.000000 | 24.800000 | 82.000000 | NaN |

10. Displaying Correlation Matrix of our 'df' dataframe

```
df.corr()
```

| | ers | displacement | horsepower | weight | acceleration | model_year |
|---|---|---|---|---|---|---|
| **mpg** | 1.000000 | -0.775396 | -0.804203 | -0.778427 | -0.831741 | 0.420289 | 0.579267 |
| **cylinders** | -0.775396 | 1.000000 | 0.950721 | 0.842983 | 0.896017 | -0.505419 | -0.348746 |
| **displacement** | -0.804203 | 0.950721 | 1.000000 | 0.897257 | 0.932824 | -0.543684 | -0.370164 |
| **horsepower** | 0.778427 | 0.842983 | 0.897257 | 1.000000 | 0.864538 | 0.689196 | 0.416361 |
| **acceleration** | 0.420289 | -0.505419 | -0.543684 | -0.689196 | -0.417457 | 1.000000 | 0.288137 |

11. Displaying all the names of our columns in 'mpg' dataset

```
df.columns
```

```
Index(['mpg', 'cylinders', 'displacement', 'horsepower', 'weight',
       'acceleration', 'model_year', 'origin', 'name'],
      dtype='object')
```

12. Displaying the total number of rows and columns in our 'mpg dataset using shape built-in function

```
df.shape
```

```
(398, 9)
```

13. Displaying the number of unique values in the 'mpg' dataset

```
df.nunique()
```

```
mpg             129
cylinders         5
displacement     82
horsepower       93
weight          351
acceleration     95
```

Saved successfully!

```
name            305
dtype: int64
```

14. Displaying the frequency categories in the Categorical feature 'origin'

```
df['origin'].value_counts()
```

```
usa       249
japan      79
europe     70
Name: origin, dtype: int64
```

15. Displaying the count of missing values in our 'mpg' dataset

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

```
mpg             0
cylinders       0
displacement    0
horsepower      6
weight          0
acceleration    0
model_year      0
origin          0
name            0
dtype: int64
```

16. Displaying a sample of 5 rows randomly

```
df.sample(5)
```

| | | | ment | horsepower | weight | acceleration | model_year | origin | name |
|---|---|---|---|---|---|---|---|---|---|
| **62** | 13.0 | 8 | 350.0 | 165.0 | 4274 | 12.0 | 72 | usa | chevrolet impala |
| **104** | 12.0 | 8 | 400.0 | 167.0 | 4906 | 12.5 | 73 | usa | ford country |
| **339** | 26.6 | 4 | 151.0 | 84.0 | 2635 | 16.4 | 81 | usa | buick skylark |
| **278** | 31.5 | 4 | 89.0 | 71.0 | 1990 | 14.9 | 78 | europe | volkswagen scirocco |

17. Displaying the value in a particular column in a particular row

Example: 4th column 274th row

```
df.horsepower[274]
```

```
103.0
```

18. Displaying first 15 rows from our dataset under only 3 columns using '.loc' function

```
df.loc[0:14,['mpg','origin','name']]
```

| | | | name |
|---|---|---|---|
| **0** | 18.0 | usa | chevrolet chevelle malibu |
| **1** | 15.0 | usa | buick skylark 320 |
| **2** | 18.0 | usa | plymouth satellite |
| **3** | 16.0 | usa | amc rebel sst |
| **4** | 17.0 | usa | ford torino |
| **5** | 15.0 | usa | ford galaxie 500 |
| **6** | 14.0 | usa | chevrolet impala |
| **7** | 14.0 | usa | plymouth fury iii |
| **8** | 14.0 | usa | pontiac catalina |
| **9** | 15.0 | usa | amc ambassador dpl |

19. Displaying last 15 rows from our dataset under same 3 columns using '.iloc' function

**11** 14.0 usa plymouth cuda 340

```
df.iloc[-16:-1,[0,-2,-1]]
```

| | | | name |
|---|---|---|---|
| **382** | 34.0 | japan | toyota corolla |
| **383** | 38.0 | japan | honda civic |
| **384** | 32.0 | japan | honda civic (auto) |
| **385** | 38.0 | japan | datsun 310 gx |
| **386** | 25.0 | usa | buick century limited |
| **387** | 38.0 | usa | oldsmobile cutlass ciera (diesel) |
| **388** | 26.0 | usa | chrysler lebaron medallion |

20. Create and Display a sub sample consisting all rows but constraint columns

```
sub_sample=df.iloc[:,[3,5,7]]
sub_sample
```

Saved successfully!

| | | | origin |
|---|---|---|---|
| **0** | 130.0 | 12.0 | usa |
| **1** | 165.0 | 11.5 | usa |
| **2** | 150.0 | 11.0 | usa |
| **3** | 150.0 | 12.0 | usa |
| **4** | 140.0 | 10.5 | usa |
| **5** | 198.0 | 10.0 | usa |
| **6** | 220.0 | 9.0 | usa |
| **7** | 215.0 | 8.5 | usa |
| **8** | 225.0 | 10.0 | usa |
| **9** | 190.0 | 8.5 | usa |
| **10** | 170.0 | 10.0 | usa |
| **11** | 160.0 | 8.0 | usa |
| **12** | 150.0 | 9.5 | usa |
| **13** | 225.0 | 10.0 | usa |
| **14** | 95.0 | 15.0 | japan |
| **15** | 95.0 | 15.5 | usa |
| **16** | 97.0 | 15.5 | usa |
| **17** | 85.0 | 16.0 | usa |
| **18** | 88.0 | 14.5 | japan |
| **19** | 46.0 | 20.5 | europe |
| **20** | 87.0 | 17.5 | europe |

Saved successfully!

europe

```
df
```

Saved successfully!

| | | | ment | horsepower | weight | acceleration | model_year | origin | name |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 18.0 | 8 | 307.0 | 130.0 | 3504 | 12.0 | 70 | usa | chevrolet chevelle malibu |
| **1** | 15.0 | 8 | 350.0 | 165.0 | 3693 | 11.5 | 70 | usa | buick skylark 320 |
| **2** | 18.0 | 8 | 318.0 | 150.0 | 3436 | 11.0 | 70 | usa | plymouth satellite |
| **3** | 16.0 | 8 | 304.0 | 150.0 | 3433 | 12.0 | 70 | usa | amc rebel sst |
| **4** | 17.0 | 8 | 302.0 | 140.0 | 3449 | 10.5 | 70 | usa | ford torino |
| **5** | 15.0 | 8 | 429.0 | 198.0 | 4341 | 10.0 | 70 | usa | ford galaxie 500 |
| **6** | 14.0 | 8 | 454.0 | 220.0 | 4354 | 9.0 | 70 | usa | chevrolet impala |

22. Set the display default to a particular number of rows(Example: 15)

```
pd.options.display.max_rows=15
```

```
df
```

Saved successfully!

| | | | ment | horsepower | weight | acceleration | model_year | origin | name |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 18.0 | 8 | 307.0 | 130.0 | 3504 | 12.0 | 70 | usa | chevrolet chevelle malibu |
| **1** | 15.0 | 8 | 350.0 | 165.0 | 3693 | 11.5 | 70 | usa | buick skylark 320 |
| **2** | 18.0 | 8 | 318.0 | 150.0 | 3436 | 11.0 | 70 | usa | plymouth satellite |
| **3** | 16.0 | 8 | 304.0 | 150.0 | 3433 | 12.0 | 70 | usa | amc rebel sst |

# Assignment 2: Label Encoding

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| **393** | 27.0 | 4 | 140.0 | 86.0 | 2790 | 15.6 | 82 | usa | [illegible] |

1. Import Library

dodge

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

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| **23** | 26.0 | 4 | 121.0 | 113.0 | 2234 | 12.5 | 70 | europe | bmw 2002 |

2. Import Dataset

```
df=pd.read_csv('https://github.com/YBI-Foundation/Dataset/raw/main/Customer%20Purchase.csv')
```

```
df.head()
```

| | | | | ducation | Review | Purchased |
|---|---|---|---|---|---|---|
| **0** | 1021 | 30 | Female | School | Average | No |
| **1** | 1022 | 68 | Female | UG | Poor | No |
| **2** | 1023 | 70 | Female | PG | Good | No |

```
df.shape
```

```
(50, 6)
```

```
df.columns
```

```
Index(['Customer ID', 'Age', 'Gender', 'Education', 'Review', 'Purchased'], dtype='object')
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50 entries, 0 to 49
Data columns (total 6 columns):
 #   Column       Non-Null Count  Dtype 
---  ------       --------------  ----- 
 0   Customer ID  50 non-null     int64 
 1   Age          50 non-null     int64 
 2   Gender       50 non-null     object
 3   Education    50 non-null     object
 4   Review       50 non-null     object
 5   Purchased    50 non-null     object
dtypes: int64(2), object(4)
memory usage: 2.5+ KB
```

### 3. Displaying y(Categorical) separately

```
y=df['Purchased']
y
```

Saved successfully!

```
1       No
2       No
3       No
4       No
      ...
45     Yes
46      No
47     Yes
48     Yes
49      No
Name: Purchased, Length: 50, dtype: object
```

### 4. Encode y as an Integer Array

```
from sklearn.preprocessing import LabelEncoder
```

```
l=LabelEncoder()
```

```
y=l.fit_transform(y)
```

```
y
```

```
array([0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0,
       1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0,
       0, 1, 0, 1, 1, 0])
```

### 5. Displaying various Encoded Categories

```
l.classes_
```

Saved successfully!

```
ct)
```

6. Reverse Encode

```
l.inverse_transform([0,0,1,1])
```

```
array(['No', 'No', 'Yes', 'Yes'], dtype=object)
```

# Assignment 3 and 6: Ordinal Encoder

1. Import Library

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

2. Import Dataset

```
df=pd.read_csv('https://github.com/YBI-Foundation/Dataset/raw/main/Customer%20Purchase.csv')
```

```
df.head()
```

| | | | | ducation | Review | Purchased |
|---|---|---|---|---|---|---|
| 0 | 1021 | 30 | Female | School | Average | No |

```
df.shape
```

```
(50, 6)
```

```
df.columns
```

```
Index(['Customer ID', 'Age', 'Gender', 'Education', 'Review', 'Purchased'], dtype='object')
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50 entries, 0 to 49
Data columns (total 6 columns):
 #   Column       Non-Null Count  Dtype 
---  ------       --------------  ----- 
 0   Customer ID  50 non-null     int64 
 1   Age          50 non-null     int64 
 2   Gender       50 non-null     object
 3   Education    50 non-null     object
 4   Review       50 non-null     object
 5   Purchased    50 non-null     object
dtypes: int64(2), object(4)
memory usage: 2.5+ KB
```

### 3. Display features(X) separately

```
X=df[['Gender', 'Education', 'Review']]
X
```

Saved successfully!

| | | | |
|---|---|---|---|
| **0** | Female | School | Average |
| **1** | Female | UG | Poor |
| **2** | Female | PG | Good |
| **3** | Female | PG | Good |
| **4** | Female | UG | Average |
| **...** | ... | ... | ... |
| **45** | Male | PG | Poor |
| **46** | Female | PG | Poor |
| **47** | Female | PG | Good |
| **48** | Female | UG | Good |
| **49** | Female | UG | Good |

4. Encode x as an integer array

```
from sklearn.preprocessing import OrdinalEncoder

o=OrdinalEncoder()

X=o.fit_transform(X)

X
```

```
array([[0., 1., 0.],
       [0., 2., 2.],
       [0., 0., 1.],
       [0., 0., 1.],
```

Saved successfully!

```
[1., 1., 1.],
[0., 1., 2.],
[0., 2., 0.],
[1., 2., 1.],
[0., 2., 1.],
[1., 2., 1.],
[1., 1., 2.],
[0., 1., 0.],
[1., 0., 2.],
[1., 2., 2.],
[1., 2., 2.],
[0., 2., 2.],
[1., 1., 1.],
[1., 0., 2.],
[0., 1., 0.],
[1., 0., 0.],
[0., 0., 2.],
[0., 1., 1.],
[0., 0., 0.],
[0., 1., 1.],
[0., 0., 2.],
[0., 0., 2.],
[1., 1., 2.],
[0., 2., 0.],
[1., 2., 0.],
[0., 1., 2.],
[1., 2., 0.],
[0., 0., 1.],
[1., 1., 0.],
[1., 1., 2.],
[0., 2., 1.],
[1., 0., 0.],
[0., 1., 1.],
[1., 0., 2.],
[1., 1., 1.],
[1., 0., 1.],
[0., 0., 1.],
[1., 0., 2.],
```

Saved successfully!

```
       [0., 0., 1.],
       [0., 2., 1.],
       [0., 2., 1.]])
```

### 5. Displaying various Encoding Categories

```
o.categories_
```

```
[array(['Female', 'Male'], dtype=object),
 array(['PG', 'School', 'UG'], dtype=object),
 array(['Average', 'Good', 'Poor'], dtype=object)]
```

### 6. Reverse Encode

```
o.inverse_transform([[1,2,0]])
```

```
array([['Male', 'UG', 'Average']], dtype=object)
```

```
o.inverse_transform([[0,0,0]])
```

```
array([['Female', 'PG', 'Average']], dtype=object)
```

```
o.inverse_transform([[1,1,1]])
```

```
array([['Male', 'School', 'Good']], dtype=object)
```

```
o.inverse_transform([[0,2,2]])
```

```
array([['Female', 'UG', 'Poor']], dtype=object)
```

Saved successfully!

an integer array of your choice

```
X=df[['Gender', 'Education', 'Review']]
```

```
o=OrdinalEncoder(categories=[['Male','Female'],['School','UG','PG'],['Poor','Average','Good']])
```

```
X=o.fit_transform(X)
```

```
o.categories_
```

```
[array(['Male', 'Female'], dtype=object),
 array(['School', 'UG', 'PG'], dtype=object),
 array(['Poor', 'Average', 'Good'], dtype=object)]
```

# Assignment 4: Standard Scaling

1. Import Library

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

2. Import Dataset

```
df=pd.read_csv('https://github.com/YBI-Foundation/Dataset/raw/main/Boston.csv')
```

| | CRIM | ZN | INDUS | CHAS | NX | 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 | 296.0 | 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 | 242.0 | 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 | 242.0 | 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 | 222.0 | 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 | 222.0 | 18.7 | 396.90 | 5.33 | 36.2 |

```
df.shape
```

```
(506, 14)
```

```
df.columns
```

```
Index(['CRIM', 'ZN', 'INDUS', 'CHAS', 'NX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX',
       'PTRATIO', 'B', 'LSTAT', 'MEDV'],
      dtype='object')
```

```
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   NX       506 non-null    float64
 5   RM       506 non-null    float64
 6   AGE      506 non-null    float64
 7   DIS      506 non-null    float64
```

```
                                  nt64
                                  loat64
 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(12), int64(2)
memory usage: 55.5 KB
```

```
df.describe()
```

| | CRIM | ZN | INDUS | CHAS | NX | RM | AGE | DIS | |
|---|---|---|---|---|---|---|---|---|---|
| **count** | 506.000000 | 506.000000 | 506.000000 | 506.000000 | 506.000000 | 506.000000 | 506.000000 | 506.000000 | 506 |
| **mean** | 3.613524 | 11.363636 | 11.136779 | 0.069170 | 0.554695 | 6.284634 | 68.574901 | 3.795043 | 9 |
| **std** | 8.601545 | 23.322453 | 6.860353 | 0.253994 | 0.115878 | 0.702617 | 28.148861 | 2.105710 | 8 |
| **min** | 0.006320 | 0.000000 | 0.460000 | 0.000000 | 0.385000 | 3.561000 | 2.900000 | 1.129600 | 1 |
| **25%** | 0.082045 | 0.000000 | 5.190000 | 0.000000 | 0.449000 | 5.885500 | 45.025000 | 2.100175 | 4 |
| **50%** | 0.256510 | 0.000000 | 9.690000 | 0.000000 | 0.538000 | 6.208500 | 77.500000 | 3.207450 | 5 |
| **75%** | 3.677083 | 12.500000 | 18.100000 | 0.000000 | 0.624000 | 6.623500 | 94.075000 | 5.188425 | 24 |
| **max** | 88.976200 | 100.000000 | 27.740000 | 1.000000 | 0.871000 | 8.780000 | 100.000000 | 12.126500 | 24 |

## 3. Separating y and X

```
y=df['MEDV']
```

```
y.shape
```

```
(506,)
```

```
'NX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX',
       'PTRATIO', 'B', 'LSTAT']]
```

```
X.shape
```

```
(506, 13)
```

## 4. Splitting the data

```
from sklearn.model_selection import train_test_split
```

```
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.3,random_state=2529)
```

```
X_train.shape,X_test.shape,y_train.shape,y_test.shape
```

```
((354, 13), (152, 13), (354,), (152,))
```

## 5. Scaling of X

```
from sklearn.preprocessing import StandardScaler
```

```
s=StandardScaler()
```

```
X_train_s=s.fit_transform(X_train)
```

```
X_test_s=s.fit_transform(X_test)
```

Saved successfully!

y by their own caluculated mean and sd

```
X_train_s
```

```
array([[-0.14113619, -0.48175769, -0.19860022, ...,  0.00438903,
        -0.05084503, -0.01555641],
       [-0.42121529,  3.02166196, -1.33410259, ..., -1.68641979,
         0.42969249, -1.33650784],
       [-0.41266839, -0.48175769,  0.22414717, ...,  0.14148164,
         0.19739169, -0.10842497],
       ...,
       [-0.38944304, -0.48175769, -0.19860022, ...,  0.00438903,
         0.37963873,  0.77313338],
       [-0.41404001,  0.41002186, -0.81324318, ..., -0.72677154,
         0.43161763,  0.09671754],
       [-0.41578561,  2.06618387, -1.3831586 , ..., -0.04130851,
         0.39707198, -0.68781395]])
```

```
X_test_s
```

```
array([[-0.36714008, -0.50235603, -0.6925381 , ..., -0.57641511,
         0.2366856 , -1.24860568],
       [-0.40880876, -0.50235603, -0.58591169, ..., -0.33768188,
         0.43031542, -0.31886558],
       [-0.41291768, -0.50235603, -0.12035979, ..., -0.38542852,
         0.36717526,  0.17122998],
       ...,
       [-0.34428827, -0.50235603,  1.66375525, ...,  1.23795746,
         0.30005961, -0.18769294],
       [-0.05769974, -0.50235603,  1.31684399, ..., -1.86557456,
        -0.3514533 , -0.15886379],
       [-0.42293258,  1.25907688, -0.66100071, ..., -0.48092181,
         0.43031542, -0.75418575]])
```

7. Mean of train and test sample is zero

```
array([ 7.52693576e-18,  2.50897859e-17,  5.01795717e-17,  1.12904036e-17,
       -4.74196953e-16, -1.03369918e-15, -1.85664415e-16,  7.27603790e-17,
       -3.51257002e-17,  1.15413015e-16,  7.32621747e-16, -4.01436574e-17,
       -3.51257002e-17])
```

```
X_test_s.mean(axis=0)
```

```
array([-1.75298372e-17,  3.35988547e-17, -1.16865582e-17,  1.75298372e-17,
       -1.27091320e-16, -4.26559373e-16,  2.62947558e-17,  2.07436407e-16,
       -5.25895117e-17, -1.16865582e-17,  3.15537070e-16, -7.24566606e-16,
       -1.31473779e-16])
```

8. Standard Deviation when train and test sample is one

```
X_train_s.std(axis=0)
```

```
array([1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.])
```

```
X_test_s.std(axis=0)
```

```
array([1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.])
```

# Assignment 5: Feature Scaling

1. Import Library

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

Saved successfully!

## 2. Import Dataset

```
df=pd.read_csv('https://github.com/YBI-Foundation/Dataset/raw/main/Online%20Purchase.csv')
```

```
df.head()
```

| | Customer_ID | Gender | Age | Salary | Purchased |
|---|---|---|---|---|---|
| **0** | 1 | Male | 35 | 500 | 0 |
| **1** | 2 | Female | 25 | 300000 | 1 |
| **2** | 3 | Female | 100 | 200000 | 0 |
| **3** | 15566689 | Female | 35 | 57000 | 0 |
| **4** | 15569641 | Female | 58 | 95000 | 1 |

```
df.shape
```

```
(403, 5)
```

```
df.columns
```

```
Index(['Customer_ID', 'Gender', 'Age', 'Salary', 'Purchased'], dtype='object')
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 403 entries, 0 to 402
Data columns (total 5 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
```

```
 2   Age          403 non-null    int64
 3   Salary       403 non-null    int64
 4   Purchased    403 non-null    int64
dtypes: int64(4), object(1)
memory usage: 15.9+ KB
```

```
df.describe()
```

| | Customer_ID | Age | Salary | Purchased |
|---|---|---|---|---|
| **count** | 4.030000e+02 | 403.000000 | 403.000000 | 403.000000 |
| **mean** | 1.557473e+07 | 37.771712 | 70465.260546 | 0.357320 |
| **std** | 1.352373e+06 | 10.915209 | 36598.127268 | 0.479806 |
| **min** | 1.000000e+00 | 18.000000 | 500.000000 | 0.000000 |
| **25%** | 1.562463e+07 | 29.500000 | 43000.000000 | 0.000000 |
| **50%** | 1.569326e+07 | 37.000000 | 70000.000000 | 0.000000 |
| **75%** | 1.575020e+07 | 46.000000 | 88000.000000 | 1.000000 |
| **max** | 1.581524e+07 | 100.000000 | 300000.000000 | 1.000000 |

## 3. Seperating y and X

```
y=df['Purchased']
```

```
y.shape
```

```
(403,)
```

Saved successfully!

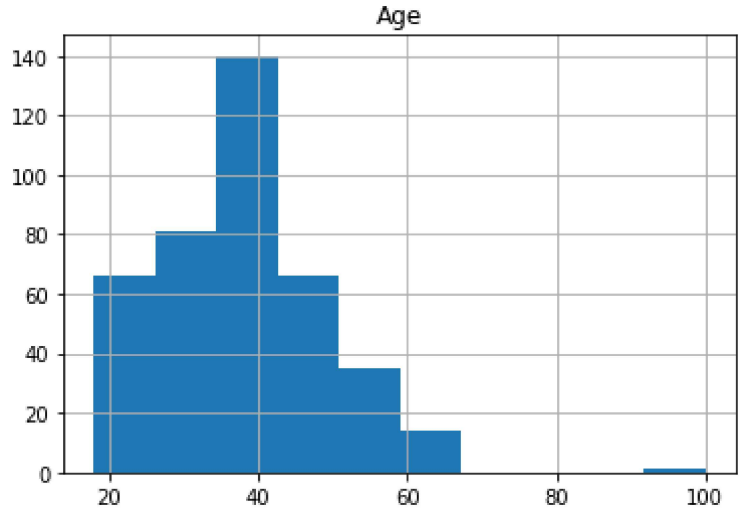
```
X.shape
```

```
(403, 2)
```

## 4. Visualisation of data

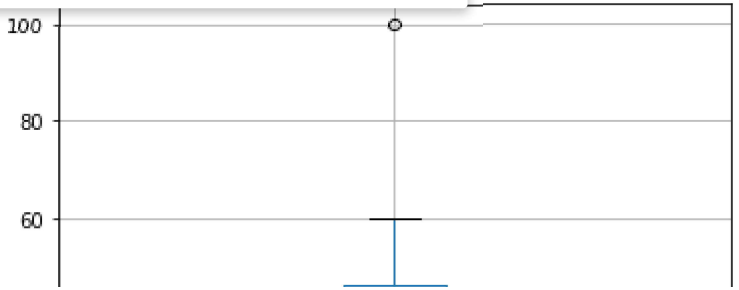
```
df[['Age']].hist()
```

```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7fd9149d34d0>]],
      dtype=object)
```




```
df[['Age']].boxplot()
```

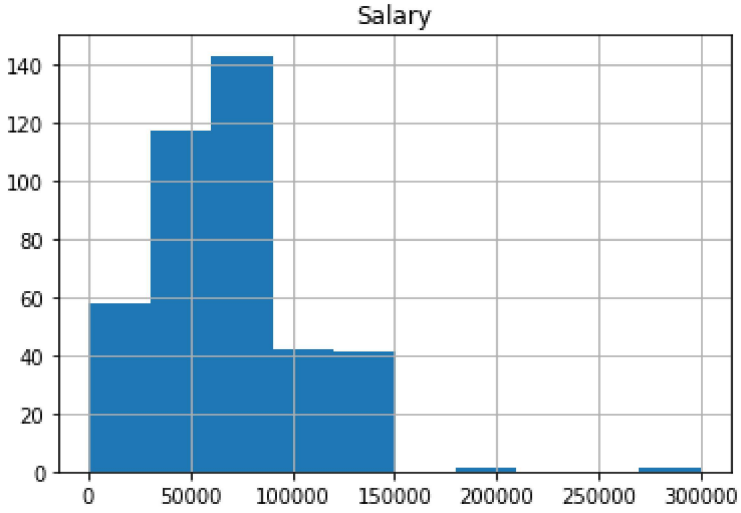
```
Subplot at 0x7fd9144ce890>
```




5. Another outlier

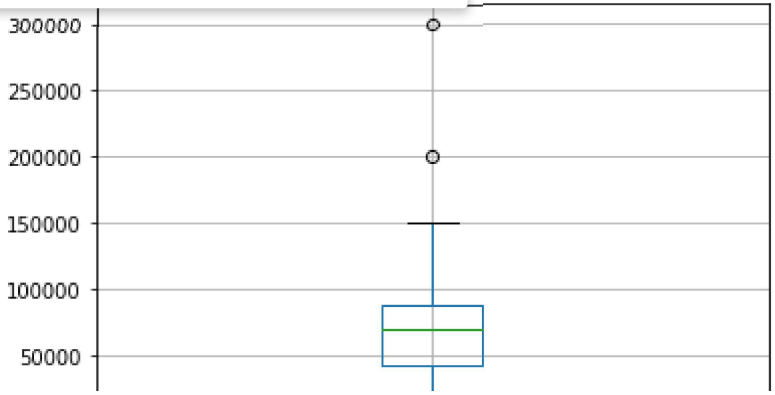

```
df[['Salary']].hist()
```

```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7fd9144a6150>]],
      dtype=object)
```




```
df[['Salary']].boxplot()
```

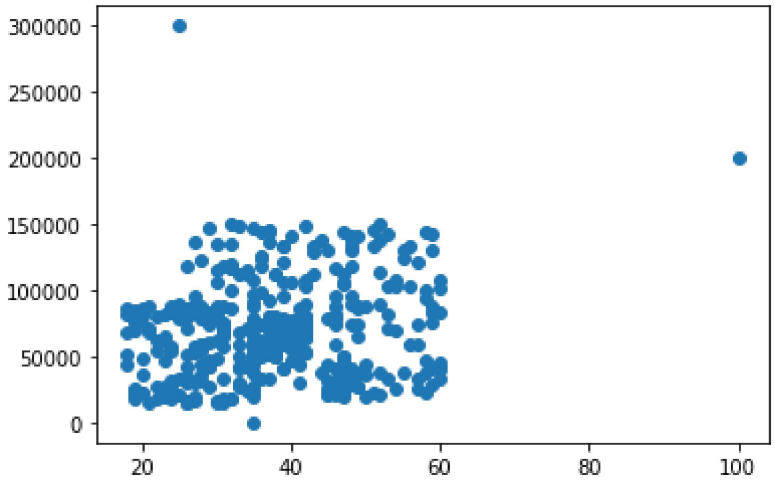
```
<Subplot at 0x7fd914430d90>
```




6. Salary has a outlier

```
plt.scatter(df['Age'],df['Salary'])
```

```
<matplotlib.collections.PathCollection at 0x7fd9143a9910>
```




7. Splitting Data

```
train_test_split

X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.3,stratify=y,random_state=2529)

X_train.shape,X_test.shape,y_train.shape,y_test.shape
```

```
((282, 2), (121, 2), (282,), (121,))
```

## 8. Scaling of X

```
from sklearn.preprocessing import StandardScaler

s=StandardScaler()

X_train_s=s.fit_transform(X_train)

X_test_s=s.fit_transform(X_test)
```

## 9. Visualisation and impact of Scaling

```
X_train_s=pd.DataFrame(X_train_s,columns=X_train.columns)
X_test_s=pd.DataFrame(X_test_s,columns=X_test.columns)
```

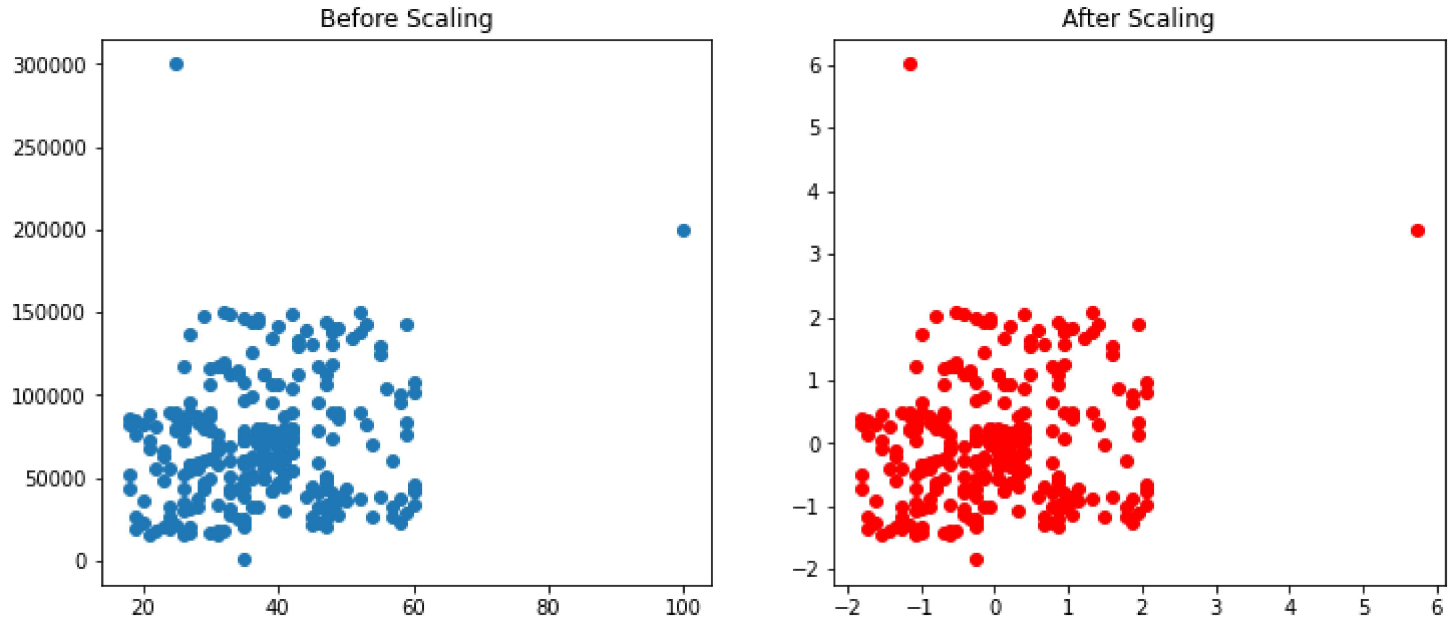
## 10. Reduced impact of outlier

```
fig, (ax1, ax2) = plt.subplots(ncols=2, figsize=(12, 5))

ax1.scatter(X_train['Age'], X_train['Salary'])
```

Saved successfully!

```
ax2.scatter(X_train_s['Age'],X_train_s['Salary'],color="red")
ax2.set_title("After Scaling")
plt.show()
```




## 11. Uniform Scale and Distribution

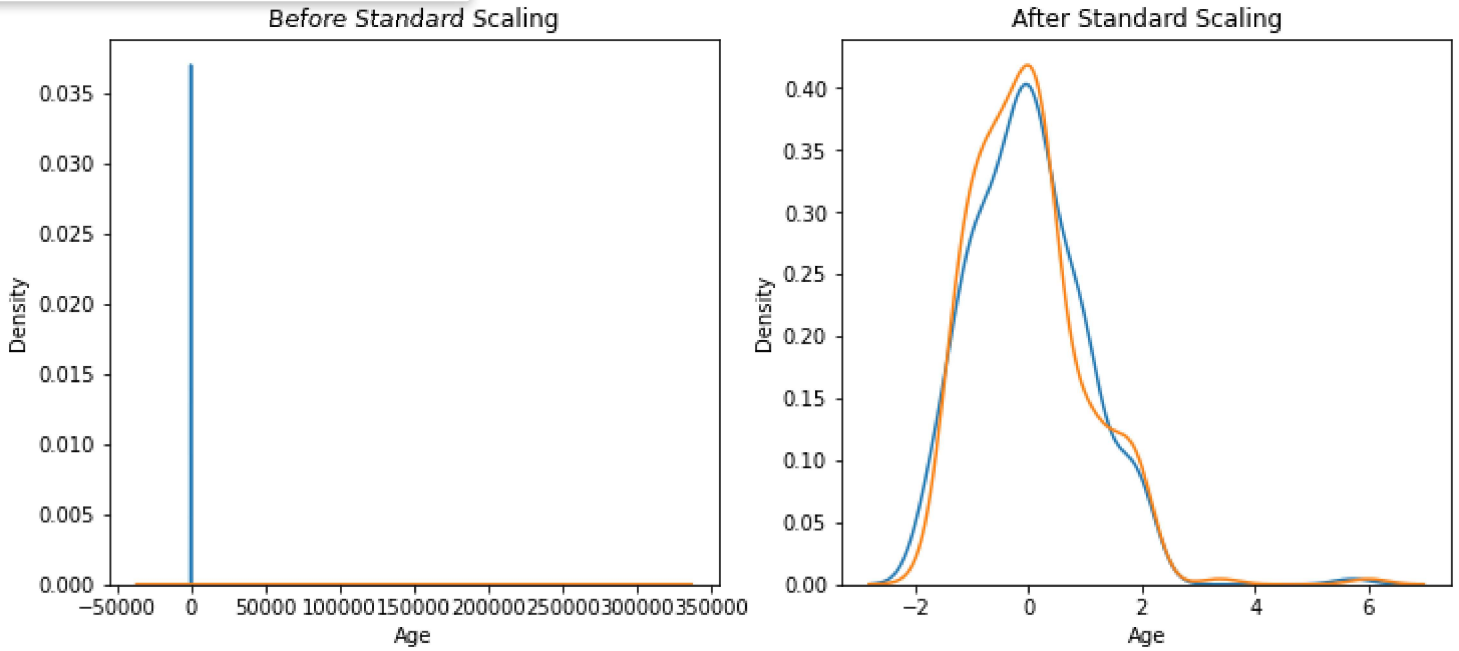
```
fig, (ax1, ax2)= plt.subplots(ncols=2, figsize=(12, 5))

#Before Scaling
ax1.set_title('Before Standard Scaling')
sns.kdeplot(X_train['Age'],ax=ax1)
sns.kdeplot(X_train['Salary'],ax=ax1)

#After Scaling
ax2.set_title('After Standard Scaling')
sns.kdeplot(X_train_s['Age'],ax=ax2)
sns.kdeplot(X_train_s['Salary'],ax=ax2)
```

Saved successfully!

```
<Subplot at 0x7fd914e27550>
```



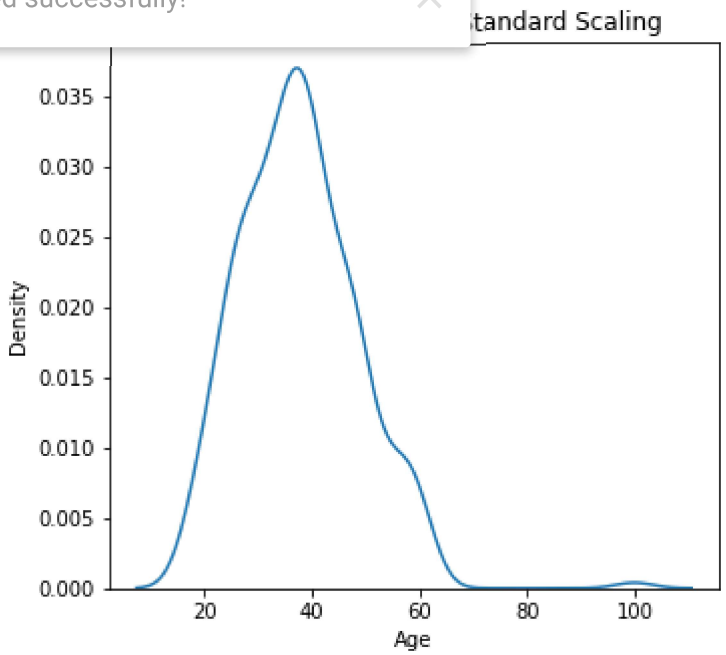

## 12. Comparing Distribution

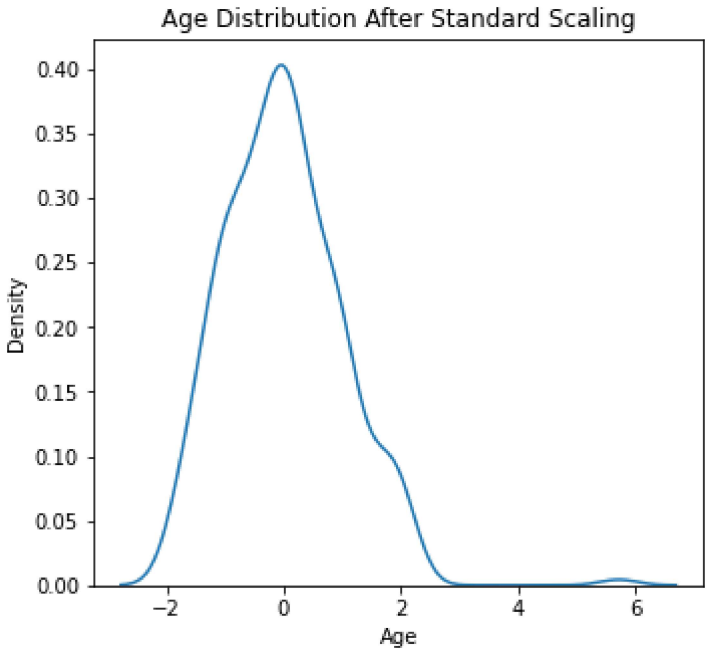
### Age

```
fig, (ax1, ax2)= plt.subplots(ncols=2, figsize=(12, 5))

#Before Standard Scaling
ax1.set_title('Age Distribution Before Standard Scaling')
sns.kdeplot(X_train['Age'],ax=ax1)

#After Standard Scaling
ax2.set_title('Age Distribution After Standard Scaling')
sns.kdeplot(X_train_s['Age'],ax=ax2)
plt.show()
```



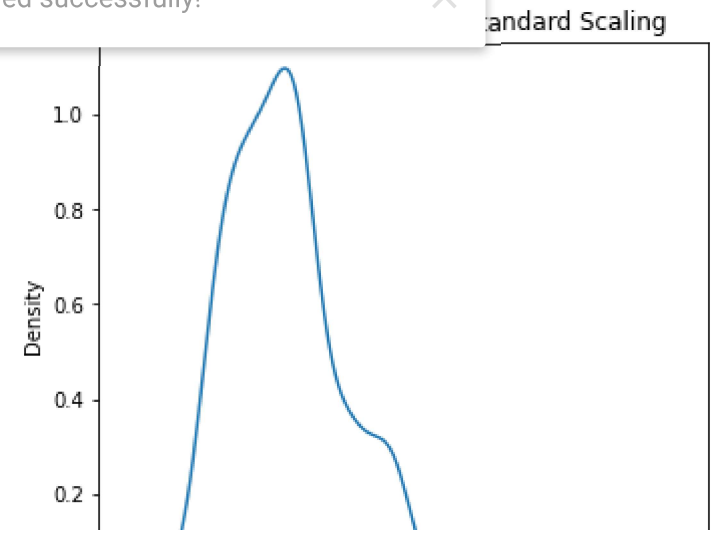



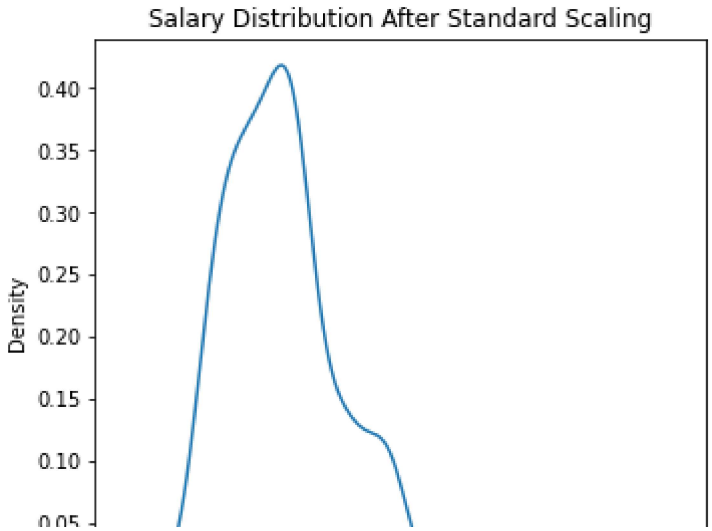

## Salary

```
fig, (ax1, ax2)= plt.subplots(ncols=2, figsize=(12, 5))

#Before Standard Scaling
ax1.set_title('Salary Distribution Before Standard Scaling')
sns.kdeplot(X_train['Salary'],ax=ax1)

#After Standard Scaling
ax2.set_title('Salary Distribution After Standard Scaling')
sns.kdeplot(X_train_s['Salary'],ax=ax2)
plt.show()
```




13. Reason for Scaling

Salary Salary

```
from sklearn.linear_model import LogisticRegression

l=LogisticRegression()

l.fit(X_train,y_train)
```

```
     LogisticRegression()
```

```
y_pred=l.predict(X_test)

from sklearn.metrics import accuracy_score
```

14. Accuracy Score with raw X

```
0.6446280991735537
```

15. Accuracy Score with Scaled X

```
l.fit(X_train_s,y_train)
```

```
LogisticRegression()
```

```
y_pred=l.predict(X_test_s)
```

```
accuracy_score(y_test,y_pred)
```

```
0.8099173553719008
```

*A Significant change can be observed!*

# Assignment 7: One Hot Encoding

1. Import Library

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

2. Import Dataset

```
'BI-Foundation/Dataset/raw/main/Customer%20Purchase.csv')
```

```
df.head()
```

| | Customer ID | Age | Gender | Education | Review | Purchased |
|---|---|---|---|---|---|---|
| **0** | 1021 | 30 | Female | School | Average | No |
| **1** | 1022 | 68 | Female | UG | Poor | No |
| **2** | 1023 | 70 | Female | PG | Good | No |
| **3** | 1024 | 72 | Female | PG | Good | No |
| **4** | 1025 | 16 | Female | UG | Average | No |

```
df.shape
```

```
(50, 6)
```

```
df.columns
```

```
Index(['Customer ID', 'Age', 'Gender', 'Education', 'Review', 'Purchased'], dtype='object')
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50 entries, 0 to 49
Data columns (total 6 columns):
 #   Column       Non-Null Count  Dtype 
---  ------       --------------  ----- 
 0   Customer ID  50 non-null     int64 
 1   Age          50 non-null     int64 
 2   Gender       50 non-null     object
 3   Education    50 non-null     object
 4   Review       50 non-null     object
 5   Purchased    50 non-null     object
```

Saved successfully!

### 3. Displaying X(Features) separately

```
X=df[['Gender', 'Education', 'Review']]
X
```

| | Gender | Education | Review |
|---|---|---|---|
| **0** | Female | School | Average |
| **1** | Female | UG | Poor |
| **2** | Female | PG | Good |
| **3** | Female | PG | Good |
| **4** | Female | UG | Average |
| **...** | ... | ... | ... |
| **45** | Male | PG | Poor |
| **46** | Female | PG | Poor |
| **47** | Female | PG | Good |
| **48** | Female | UG | Good |
| **49** | Female | UG | Good |

50 rows × 3 columns

### 4. Encoding Categorical Features as an integer array

Saved successfully!

eHotEncoder

5. Using One Hot Encoder on one of the Individual column(Example: 'Education')

```
X_Education=ohe.fit_transform(X[['Education']])
```

```
ohe.categories_
```

```
[array(['PG', 'School', 'UG'], dtype=object)]
```

```
X_Education.toarray().shape
```

```
(50, 3)
```

```
X_Education.toarray()
```

```
array([[0., 1., 0.],
       [0., 0., 1.],
       [1., 0., 0.],
       [1., 0., 0.],
       [0., 0., 1.],
       [0., 1., 0.],
       [0., 1., 0.],
       [0., 1., 0.],
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 1., 0.],
       [0., 1., 0.],
       [1., 0., 0.],
       [0., 0., 1.],
       [0., 0., 1.],
```

Saved successfully!

```
[1., 0., 0.],
[0., 1., 0.],
[1., 0., 0.],
[1., 0., 0.],
[0., 1., 0.],
[1., 0., 0.],
[0., 1., 0.],
[1., 0., 0.],
[1., 0., 0.],
[0., 1., 0.],
[0., 0., 1.],
[0., 0., 1.],
[0., 1., 0.],
[0., 0., 1.],
[1., 0., 0.],
[0., 1., 0.],
[0., 1., 0.],
[0., 0., 1.],
[1., 0., 0.],
[0., 1., 0.],
[1., 0., 0.],
[0., 1., 0.],
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
[0., 0., 1.],
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
[0., 0., 1.],
[0., 0., 1.]])
```

## 6. Using One Hot Encoder on 'Gender'

```
X_Gender=ohe.fit_transform(X[['Gender']])
```

```
[array(['Female', 'Male'], dtype=object)]
```

```
X_Gender.toarray().shape
```

```
(50, 2)
```

```
X_Gender.toarray()
```

```
array([[1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [0., 1.],
       [1., 0.],
       [1., 0.],
       [0., 1.],
       [1., 0.],
       [0., 1.],
       [0., 1.],
       [1., 0.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [1., 0.],
       [0., 1.],
       [0., 1.],
       [1., 0.],
       [0., 1.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
```

```
       [0., 1.],
       [1., 0.],
       [0., 1.],
       [1., 0.],
       [0., 1.],
       [0., 1.],
       [1., 0.],
       [0., 1.],
       [1., 0.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [1., 0.],
       [0., 1.],
       [1., 0.],
       [0., 1.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.]])
```

## 7. Using One Hot Encoder on 'Review'

```
X_Review=ohe.fit_transform(X[['Review']])
```

```
ohe.categories_
```

```
[array(['Average', 'Good', 'Poor'], dtype=object)]
```

```
X_Review.toarray().shape
```

```
(50, 3)
```

Saved successfully!

```
array([[1., 0., 0.],
       [0., 0., 1.],
       [0., 1., 0.],
       [0., 1., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.],
       [1., 0., 0.],
       [0., 1., 0.],
       [0., 1., 0.],
       [0., 1., 0.],
       [0., 0., 1.],
       [1., 0., 0.],
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 1., 0.],
       [0., 0., 1.],
       [1., 0., 0.],
       [1., 0., 0.],
       [0., 0., 1.],
       [0., 1., 0.],
       [1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.],
       [1., 0., 0.],
       [1., 0., 0.],
       [0., 0., 1.],
       [1., 0., 0.],
       [0., 1., 0.],
       [1., 0., 0.],
       [0., 0., 1.],
       [0., 1., 0.],
       [1., 0., 0.],
```

```
       [0., 1., 0.],
       [0., 1., 0.],
       [0., 0., 1.],
       [1., 0., 0.],
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 1., 0.],
       [0., 1., 0.],
       [0., 1., 0.]])
```

8. Using One Hot Encoder on all the features

```
X=df[['Gender', 'Education', 'Review']]
```

```
X=ohe.fit_transform(X)
```

```
X.toarray().shape
```

```
(50, 8)
```

```
X.toarray()
```

```
array([[1., 0., 0., 1., 0., 1., 0., 0.],
       [1., 0., 0., 0., 1., 0., 0., 1.],
       [1., 0., 1., 0., 0., 0., 1., 0.],
       [1., 0., 1., 0., 0., 0., 1., 0.],
       [1., 0., 0., 0., 1., 1., 0., 0.],
       [1., 0., 0., 1., 0., 1., 0., 0.],
       [0., 1., 0., 1., 0., 0., 1., 0.],
       [1., 0., 0., 1., 0., 0., 0., 1.],
       [1., 0., 0., 0., 1., 1., 0., 0.],
       [0., 1., 0., 0., 1., 0., 1., 0.],
       [1., 0., 0., 0., 1., 0., 1., 0.],
```

Saved successfully!

```
       1., 0.],
       0., 1.],
[1., 0., 0., 1., 0., 1., 0., 0.],
[0., 1., 1., 0., 0., 0., 0., 1.],
[0., 1., 0., 0., 1., 0., 0., 1.],
[0., 1., 0., 0., 1., 0., 0., 1.],
[1., 0., 0., 0., 1., 0., 0., 1.],
[0., 1., 0., 1., 0., 0., 1., 0.],
[0., 1., 1., 0., 0., 0., 0., 1.],
[1., 0., 0., 1., 0., 1., 0., 0.],
[0., 1., 1., 0., 0., 1., 0., 0.],
[1., 0., 1., 0., 0., 0., 0., 1.],
[1., 0., 0., 1., 0., 0., 1., 0.],
[1., 0., 1., 0., 0., 1., 0., 0.],
[1., 0., 0., 1., 0., 0., 1., 0.],
[1., 0., 1., 0., 0., 0., 0., 1.],
[1., 0., 1., 0., 0., 0., 0., 1.],
[0., 1., 0., 1., 0., 0., 0., 1.],
[1., 0., 0., 0., 1., 1., 0., 0.],
[0., 1., 0., 0., 1., 1., 0., 0.],
[1., 0., 0., 1., 0., 0., 0., 1.],
[0., 1., 0., 0., 1., 1., 0., 0.],
[1., 0., 1., 0., 0., 0., 1., 0.],
[0., 1., 0., 1., 0., 1., 0., 0.],
[0., 1., 0., 1., 0., 0., 0., 1.],
[1., 0., 0., 0., 1., 0., 1., 0.],
[0., 1., 1., 0., 0., 1., 0., 0.],
[1., 0., 0., 1., 0., 0., 1., 0.],
[0., 1., 1., 0., 0., 0., 0., 1.],
[0., 1., 0., 1., 0., 0., 1., 0.],
[0., 1., 1., 0., 0., 0., 1., 0.],
[1., 0., 1., 0., 0., 0., 1., 0.],
[0., 1., 1., 0., 0., 0., 0., 1.],
[1., 0., 0., 0., 1., 1., 0., 0.],
[0., 1., 1., 0., 0., 0., 0., 1.],
[1., 0., 1., 0., 0., 0., 0., 1.],
[1., 0., 1., 0., 0., 0., 1., 0.],
[1., 0., 0., 0., 1., 0., 1., 0.],
[1., 0., 0., 0., 1., 0., 1., 0.]])
```

features as per choice

```
X=df[['Gender','Education','Review']]
```

```
ohe=OneHotEncoder(categories=[['Male','Female'],['School','UG','PG'],['Poor','Average','Good']])
```

```
X=ohe.fit_transform(X)
```

```
ohe.categories_
```

```
[array(['Male', 'Female'], dtype=object),
 array(['School', 'UG', 'PG'], dtype=object),
 array(['Poor', 'Average', 'Good'], dtype=object)]
```

```
X.toarray().shape
```

```
(50, 8)
```

```
X.toarray()
```

```
array([[0., 1., 1., 0., 0., 0., 1., 0.],
       [0., 1., 0., 1., 0., 1., 0., 0.],
       [0., 1., 0., 0., 1., 0., 0., 1.],
       [0., 1., 0., 0., 1., 0., 0., 1.],
       [0., 1., 0., 1., 0., 0., 1., 0.],
       [0., 1., 1., 0., 0., 0., 1., 0.],
       [1., 0., 1., 0., 0., 0., 0., 1.],
       [0., 1., 1., 0., 0., 1., 0., 0.],
       [0., 1., 0., 1., 0., 0., 1., 0.],
       [1., 0., 0., 1., 0., 0., 0., 1.],
       [0., 1., 0., 1., 0., 0., 0., 1.],
       [1., 0., 0., 1., 0., 0., 0., 1.],
       [1., 0., 1., 0., 0., 1., 0., 0.],
       [0., 1., 1., 0., 0., 0., 1., 0.],
       [1., 0., 0., 0., 1., 1., 0., 0.],
```

```
       0., 0.],
       0., 0.],
[0., 1., 0., 1., 0., 1., 0., 0.],
[1., 0., 1., 0., 0., 0., 0., 1.],
[1., 0., 0., 0., 1., 1., 0., 0.],
[0., 1., 1., 0., 0., 0., 1., 0.],
[1., 0., 0., 0., 1., 0., 1., 0.],
[0., 1., 0., 0., 1., 1., 0., 0.],
[0., 1., 1., 0., 0., 0., 0., 1.],
[0., 1., 0., 0., 1., 0., 1., 0.],
[0., 1., 1., 0., 0., 0., 0., 1.],
[0., 1., 0., 0., 1., 1., 0., 0.],
[0., 1., 0., 0., 1., 1., 0., 0.],
[1., 0., 1., 0., 0., 1., 0., 0.],
[0., 1., 0., 1., 0., 0., 1., 0.],
[1., 0., 0., 1., 0., 0., 1., 0.],
[0., 1., 1., 0., 0., 1., 0., 0.],
[1., 0., 0., 1., 0., 0., 1., 0.],
[0., 1., 0., 0., 1., 0., 0., 1.],
[1., 0., 1., 0., 0., 0., 1., 0.],
[1., 0., 1., 0., 0., 1., 0., 0.],
[0., 1., 0., 1., 0., 0., 0., 1.],
[1., 0., 0., 0., 1., 0., 1., 0.],
[0., 1., 1., 0., 0., 0., 0., 1.],
[1., 0., 0., 0., 1., 1., 0., 0.],
[1., 0., 1., 0., 0., 0., 0., 1.],
[1., 0., 0., 0., 1., 0., 0., 1.],
[0., 1., 0., 0., 1., 0., 0., 1.],
[1., 0., 0., 0., 1., 1., 0., 0.],
[0., 1., 0., 1., 0., 0., 1., 0.],
[1., 0., 0., 0., 1., 1., 0., 0.],
[0., 1., 0., 0., 1., 1., 0., 0.],
[0., 1., 0., 0., 1., 0., 0., 1.],
[0., 1., 0., 1., 0., 0., 0., 1.],
[0., 1., 0., 1., 0., 0., 0., 1.]])
```

## 10. Reverse Encode Integer Array

```
0.,0.,1.,0.]])
```

```
array([['Female', 'School', 'Average']], dtype=object)
```

## 11. Drop First Dummy Variable from Each Categorical Feature(X) to avoid Multicolinearity

```
X=df[['Gender','Education','Review']]
```

```
ohe=OneHotEncoder(drop='first')
```

```
X=ohe.fit_transform(X)
```

```
ohe.categories_
```

```
[array(['Female', 'Male'], dtype=object),
 array(['PG', 'School', 'UG'], dtype=object),
 array(['Average', 'Good', 'Poor'], dtype=object)]
```

```
X.toarray().shape
```

```
(50, 5)
```

```
X.toarray()
```

```
array([[0., 1., 0., 0., 0.],
       [0., 0., 1., 0., 1.],
       [0., 0., 0., 1., 0.],
       [0., 0., 0., 1., 0.],
       [0., 0., 1., 0., 0.],
       [0., 1., 0., 0., 0.],
       [1., 1., 0., 1., 0.],
       [0., 1., 0., 0., 1.],
       [0., 0., 1., 0., 0.],
```

Saved successfully!

```
       [1., 1., 0., 0., 1.],
       [0., 1., 0., 0., 0.],
       [1., 0., 0., 0., 1.],
       [1., 0., 1., 0., 1.],
       [1., 0., 1., 0., 1.],
       [0., 0., 1., 0., 1.],
       [1., 1., 0., 1., 0.],
       [1., 0., 0., 0., 1.],
       [0., 1., 0., 0., 0.],
       [1., 0., 0., 0., 0.],
       [0., 0., 0., 0., 1.],
       [0., 1., 0., 1., 0.],
       [0., 0., 0., 0., 0.],
       [0., 1., 0., 1., 0.],
       [0., 0., 0., 0., 1.],
       [0., 0., 0., 0., 1.],
       [1., 1., 0., 0., 1.],
       [0., 0., 1., 0., 0.],
       [1., 0., 1., 0., 0.],
       [0., 1., 0., 0., 1.],
       [1., 0., 1., 0., 0.],
       [0., 0., 0., 1., 0.],
       [1., 1., 0., 0., 0.],
       [1., 1., 0., 0., 1.],
       [0., 0., 1., 1., 0.],
       [1., 0., 0., 0., 0.],
       [0., 1., 0., 1., 0.],
       [1., 0., 0., 0., 1.],
       [1., 1., 0., 1., 0.],
       [1., 0., 0., 1., 0.],
       [0., 0., 0., 1., 0.],
       [1., 0., 0., 0., 1.],
       [0., 0., 1., 0., 0.],
       [1., 0., 0., 0., 1.],
       [0., 0., 0., 0., 1.],
       [0., 0., 0., 1., 0.],
       [0., 0., 1., 1., 0.],
       [0., 0., 1., 1., 0.]])
```

Saved successfully!

Link of the same:

https://colab.research.google.com/drive/14qWmbLOCD_oLXeTu8Sxpfnvvgh5nvcvv?usp=sharing