Practical No. 3

Name: Prathamesh Pawar | Roll No: B-23

In [1]:

```
import numpy as np
import pandas as pd
import statistics as st
```

Descriptive Statistics - Measures of Central Tendency and variability ¶

- Perform the following operations on any open source dataset (e.g., data.csv)
- 1. Provide summary statistics (mean, median, minimum, maximum, standard deviation) for a dataset (age, income etc.) with numeric variables grouped by one of the qualitative (categorical) variable. For example, if your categorical variable is age groups and quantitative variable is income, then provide summary statistics of income grouped by the age groups. Create a list that contains a numeric value for each response to the categorical variable.
- 2. Write a Python program to display some basic statistical details like percentile, mean, standard deviation etc. of the species of 'Iris-setosa', 'Iris-versicolor' and 'Iris- versicolor' of iris.csv dataset.
- Provide the codes with outputs and explain everything that you do in this step.

1. Summary statistics
2. Types of Variables
3. Summary statistics of income grouped by the age groups
4. Display basic statistical details on the iris dataset.

In [2]:

```
df = pd.read_csv("Mall_Customers.csv")
```

In [3]:

```
df
```

Out[3]:

	CustomerID	Genre	Age	Annual Income (k$)	Spending Score (1-100)
0	1	Male	19	15	39
1	2	Male	21	15	81
2	3	Female	20	16	6
3	4	Female	23	16	77
4	5	Female	31	17	40
...	...	...	...	...	...
195	196	Female	35	120	79
196	197	Female	45	126	28
197	198	Male	32	126	74
198	199	Male	32	137	18
199	200	Male	30	137	83

200 rows × 5 columns

1. Summary statistics

1. Mean()

In [5]:

```
df.mean()  # mean of all columns
```

```
C:\Users\COMPHOD\AppData\Local\Temp\ipykernel_12048\734542734.py:1: Future
Warning: The default value of numeric_only in DataFrame.mean is deprecate
d. In a future version, it will default to False. In addition, specifying
'numeric_only=None' is deprecated. Select only valid columns or specify th
e value of numeric_only to silence this warning.
  df.mean()  # mean of all columns
```

Out[5]:

```
CustomerID                100.50
Age                        38.85
Annual Income (k$)         60.56
Spending Score (1-100)     50.20
dtype: float64
```

In [6]:

```
df.loc[:,'Age'].mean()   # mean of specific column
```

Out[6]:

```
38.85
```

In [10]:

```
df.mean(axis=1)[0:4] # mean row wise
```

```
C:\Users\COMPHOD\AppData\Local\Temp\ipykernel_12048\1227886012.py:1: Futur
eWarning: Dropping of nuisance columns in DataFrame reductions (with 'nume
ric_only=None') is deprecated; in a future version this will raise TypeErr
or.  Select only valid columns before calling the reduction.
  df.mean(axis=1)[0:4] # mean row wise
```

Out[10]:

```
0    18.50
1    29.75
2    11.25
3    30.00
dtype: float64
```

2. Median

In [11]:

```
df.median() # median of all columns
```

```
C:\Users\COMPHOD\AppData\Local\Temp\ipykernel_12048\3838006088.py:1: Futur
eWarning: The default value of numeric_only in DataFrame.median is depreca
ted. In a future version, it will default to False. In addition, specifyin
g 'numeric_only=None' is deprecated. Select only valid columns or specify
the value of numeric_only to silence this warning.
  df.median() # median of all columns
```

Out[11]:

```
CustomerID               100.5
Age                       36.0
Annual Income (k$)        61.5
Spending Score (1-100)    50.0
dtype: float64
```

In [12]:

```
df.loc[:,'Age'].median() # median of specific column
```

Out[12]:

```
36.0
```

In [13]:

```
df.median(axis=1)[0:4] #median row wise
```

```
C:\Users\COMPHOD\AppData\Local\Temp\ipykernel_12048\2735118674.py:1: Futur
eWarning: Dropping of nuisance columns in DataFrame reductions (with 'nume
ric_only=None') is deprecated; in a future version this will raise TypeErr
or.  Select only valid columns before calling the reduction.
  df.median(axis=1)[0:4] #median row wise
```

Out[13]:

```
0    17.0
1    18.0
2    11.0
3    19.5
dtype: float64
```

3. Mode

In [14]:

```
df.mode() # mode of all columns
```

Out[14]:

	CustomerID	Genre	Age	Annual Income (k$)	Spending Score (1-100)
0	1	Female	32.0	54.0	42.0
1	2	NaN	NaN	78.0	NaN
2	3	NaN	NaN	NaN	NaN
3	4	NaN	NaN	NaN	NaN
4	5	NaN	NaN	NaN	NaN
...	...	...	...	...	...
195	196	NaN	NaN	NaN	NaN
196	197	NaN	NaN	NaN	NaN
197	198	NaN	NaN	NaN	NaN
198	199	NaN	NaN	NaN	NaN
199	200	NaN	NaN	NaN	NaN

200 rows × 5 columns

In [15]:

```
df.loc[:,'Age'].mode() # mode of a specific column.
```

Out[15]:

```
0    32
Name: Age, dtype: int64
```

4. Minimum

In [16]:

```
df.min() # minimum of all columns
```

Out[16]:

```
CustomerID                    1
Genre                    Female
Age                          18
Annual Income (k$)           15
Spending Score (1-100)        1
dtype: object
```

In [17]:

```
df.loc[:,'Age'].min(skipna = False)  # minimum of Specific column
```

Out[17]:

```
18
```

5.Maximum

In [18]:

```
df.max() # Maximum of all columns
```

Out[18]:

```
CustomerID                  200
Genre                      Male
Age                          70
Annual Income (k$)          137
Spending Score (1-100)       99
dtype: object
```

In [19]:

```
df.loc[:,'Age'].max(skipna = False) # Maximum of Specific column
```

Out[19]:

```
70
```

6. Standard Deviation

In [20]:

```
df.std() # Standard Deviation of all columns
```

```
C:\Users\COMPHOD\AppData\Local\Temp\ipykernel_12048\301237031.py:1: Future
Warning: The default value of numeric_only in DataFrame.std is deprecated.
In a future version, it will default to False. In addition, specifying 'nu
meric_only=None' is deprecated. Select only valid columns or specify the v
alue of numeric_only to silence this warning.
  df.std() # Standard Deviation of all columns
```

Out[20]:

```
CustomerID                57.879185
Age                       13.969007
Annual Income (k$)        26.264721
Spending Score (1-100)    25.823522
dtype: float64
```

In [21]:

```
df.loc[:,'Age'].std() # Standard Deviation of specific column
```

Out[21]:

```
13.96900733155888
```

In [22]:

```
df.std(axis=1)[0:4] # Standard Deviation row wise
```

```
C:\Users\COMPHOD\AppData\Local\Temp\ipykernel_12048\1639279273.py:1: Futur
eWarning: Dropping of nuisance columns in DataFrame reductions (with 'nume
ric_only=None') is deprecated; in a future version this will raise TypeErr
or.  Select only valid columns before calling the reduction.
  df.std(axis=1)[0:4] # Standard Deviation row wise
```

Out[22]:

```
0    15.695010
1    35.074920
2     8.057088
3    32.300671
dtype: float64
```

2. Types of Variables:

A variable is a characteristic that can be measured and that can assume different values. Height, age, income, province or country of birth, grades obtained at school and type of housing are all examples of variables. Variables may be classified into two main categories:

1. Categorical and
2. Numeric.

Each category is then classified in two subcategories: nominal or ordinal for categorical variables, discrete or continuous for numeric variables.

3. Summary statistics of income grouped by the age groups

Problem Statement:

For example, if your categorical variable is age groups and quantitative variable is income, then provide summary statistics of income grouped by the age groups. Create a list that contains a numeric value for each response to the categorical variable.

1. Categorical Variable: Genre
2. Quantitative Variable : Age

In [24]:

```
df.groupby(['Genre'])['Age'].mean()
```

Out[24]:

```
Genre
Female    38.098214
Male      39.806818
Name: Age, dtype: float64
```

1. Categorical Variable: Genre
2. Quantitative Variable : Income

In [43]:

```
df_u=df.rename(columns= {'Annual Income (k$)':'Income'}, inplace= False)
```

In [46]:

```
df_u
```

Out[46]:

	CustomerID	Genre	Age	Income	Spending Score (1-100)
0	1	Male	19	15	39
1	2	Male	21	15	81
2	3	Female	20	16	6
3	4	Female	23	16	77
4	5	Female	31	17	40
...	...	...	...	...	...
195	196	Female	35	120	79
196	197	Female	45	126	28
197	198	Male	32	126	74
198	199	Male	32	137	18
199	200	Male	30	137	83

200 rows × 5 columns

In [45]:

```
df_u.groupby(['Genre']).Income.mean()
```

Out[45]:

```
Genre
Female    59.250000
Male      62.227273
Name: Income, dtype: float64
```

To create a list that contains a numeric value for each response to the categorical variable

- One hot encoding is a technique used to represent categorical variables as numerical values in a machine learning model.
- In this technique, the categorical parameters will prepare separate columns for both Male and Female labels. So, wherever there is Male, the value will be 1 in Male column and 0 in Female column, and vice-versa.

In [47]:

```
from sklearn import preprocessing
enc = preprocessing.OneHotEncoder()
enc_df = pd.DataFrame(enc.fit_transform(df[['Genre']]).toarray())
enc_df
```

Out[47]:

	0	1
0	0.0	1.0
1	0.0	1.0
2	1.0	0.0
3	1.0	0.0
4	1.0	0.0
...	...	...
195	1.0	0.0
196	1.0	0.0
197	0.0	1.0
198	0.0	1.0
199	0.0	1.0

200 rows × 2 columns

To concat numerical list to dataframe

In [48]:

```
df_encode =df_u.join(enc_df)
df_encode
```

Out[48]:

	CustomerID	Genre	Age	Income	Spending Score (1-100)	0	1
0	1	Male	19	15	39	0.0	1.0
1	2	Male	21	15	81	0.0	1.0
2	3	Female	20	16	6	1.0	0.0
3	4	Female	23	16	77	1.0	0.0
4	5	Female	31	17	40	1.0	0.0
...	...	...	...	...	...	...	...
195	196	Female	35	120	79	1.0	0.0
196	197	Female	45	126	28	1.0	0.0
197	198	Male	32	126	74	0.0	1.0
198	199	Male	32	137	18	0.0	1.0
199	200	Male	30	137	83	0.0	1.0

200 rows × 7 columns

4. Display basic statistical details on the iris dataset.

In [49]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

In [64]:

```
df_iris = pd.read_csv("Iris.csv")
df_iris.head()
```

Out[64]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	NaN	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa

Basic statistical details of Iris dataset

In [69]:

```
print('Iris-setosa')
setosa = df_iris['Species'] == 'Iris-setosa'
print(df_iris[setosa].describe())
print('\nIris-versicolor')
versicolor = df_iris['Species'] == 'Iris-versicolor'
print(df_iris[versicolor].describe())
print('\nIris-virginica')
virginica = df_iris['Species'] == 'Iris-virginica'
print(df_iris[virginica].describe())
```

```
Iris-setosa
             Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm
count  50.00000       50.00000     50.000000      49.000000      50.00000
mean   25.50000        5.00600      3.418000       1.467347       0.24400
std    14.57738        0.35249      0.381024       0.173671       0.10721
min     1.00000        4.30000      2.300000       1.000000       0.10000
25%    13.25000        4.80000      3.125000       1.400000       0.20000
50%    25.50000        5.00000      3.400000       1.500000       0.20000
75%    37.75000        5.20000      3.675000       1.600000       0.30000
max    50.00000        5.80000      4.400000       1.900000       0.60000

Iris-versicolor
              Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm
count   50.00000      50.000000     50.000000      50.000000     50.000000
mean    75.50000       5.936000      2.770000       4.260000      1.326000
std     14.57738       0.516171      0.313798       0.469911      0.197753
min     51.00000       4.900000      2.000000       3.000000      1.000000
25%     63.25000       5.600000      2.525000       4.000000      1.200000
50%     75.50000       5.900000      2.800000       4.350000      1.300000
75%     87.75000       6.300000      3.000000       4.600000      1.500000
max    100.00000       7.000000      3.400000       5.100000      1.800000

Iris-virginica
              Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm
count   50.00000       50.00000     50.000000      50.000000      50.00000
mean   125.50000        6.58800      2.974000       5.552000       2.02600
std     14.57738        0.63588      0.322497       0.551895       0.27465
min    101.00000        4.90000      2.200000       4.500000       1.40000
25%    113.25000        6.22500      2.800000       5.100000       1.80000
50%    125.50000        6.50000      3.000000       5.550000       2.00000
75%    137.75000        6.90000      3.175000       5.875000       2.30000
max    150.00000        7.90000      3.800000       6.900000       2.50000
```

In [70]:

```
df_iris.dtypes.value_counts()
```

Out[70]:

```
float64    4
int64      1
object     1
dtype: int64
```