More About Statistics

1. Basic Staticial Operations

In [ ]:

```
# import required libraries
import pandas as pd
import numpy as np
# load iris dataset
IR = pd.read_csv("Iris.csv")
IR
```

Out[ ]:

	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	1.3	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
...	...	...	...	...	...	...
145	146	6.7	3.0	5.2	2.3	Iris-virginica
146	147	6.3	2.5	5.0	1.9	Iris-virginica
147	148	6.5	3.0	5.2	2.0	Iris-virginica
148	149	6.2	3.4	5.4	2.3	Iris-virginica
149	150	5.9	3.0	5.1	1.8	Iris-virginica

150 rows × 6 columns

In [ ]:

```
# display the details of numeric columns in dataset
IR.describe()
```

Out[ ]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
count	150.000000	150.000000	150.000000	150.000000	150.000000
mean	75.500000	5.843333	3.054000	3.758667	1.198667
std	43.445368	0.828066	0.433594	1.764420	0.763161
min	1.000000	4.300000	2.000000	1.000000	0.100000
25%	38.250000	5.100000	2.800000	1.600000	0.300000
50%	75.500000	5.800000	3.000000	4.350000	1.300000
75%	112.750000	6.400000	3.300000	5.100000	1.800000
max	150.000000	7.900000	4.400000	6.900000	2.500000

In [ ]:

```
#calculate mean of sepal-length column
IR["SepalLengthCm"].mean()
```

Out[ ]:

```
5.843333333333334
```

In [ ]:

```
#calculate median of sepal-width column
IR["SepalWidthCm"].median()
```

Out[ ]:

```
3.0
```

In [ ]:

```
#calculate mode of petal-length column
IR["PetalLengthCm"].mode()
```

Out[ ]:

```
0    1.5
Name: PetalLengthCm, dtype: float64
```

In [ ]:

```
#calculate mean of all individual-columns having numeric data
IR.median()
```

```
C:\Users\My Net\AppData\Local\Temp\ipykernel_6864\1186170656.py:2: FutureWarning: Dropping of nuisance columns in DataFrame redu
ctions (with 'numeric_only=None') is deprecated; in a future version this will raise TypeError.  Select only valid columns befor
e calling the reduction.
  IR.median()
```

Out[ ]:

```
Id               75.50
SepalLengthCm     5.80
SepalWidthCm      3.00
PetalLengthCm     4.35
PetalWidthCm      1.30
dtype: float64
```

Testing

> ## 1. Shapiro Wilk Test
>
> Shapiro Wilk tests is used to identify whether the data or sub-samples of data has Gaussian Distribution or not.

- Each sample from observation are independent and identically distributed.
    - **H0**: if the data sample has gasussian distribution(normally ditributed)
        - **H1**: if the data sample has gasussian distribution(normally ditributed).

In [ ]:

```
# import library
from scipy.stats import shapiro
# applying shapiro's test on sepal-length column taken from iris dataset
stat,p = shapiro(IR['SepalLengthCm'])
print('stats = ', stat)
print('p value = ', p)
if p > 0.05:
    print('Normally Distibuted')
else:
    print('Not Normally Distibuted')
```

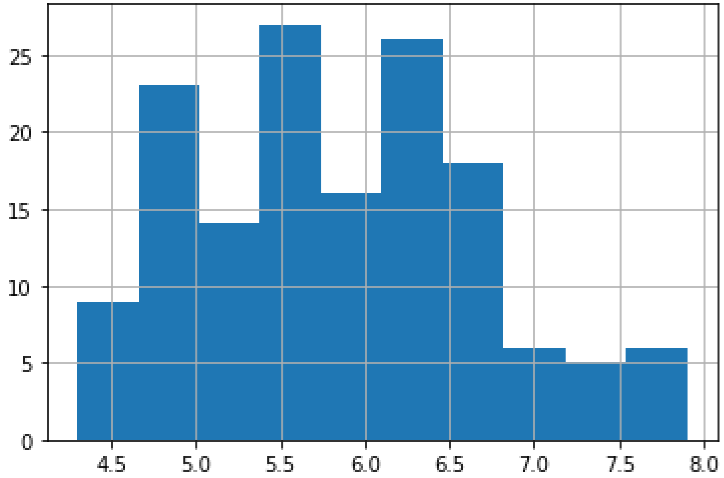
```
stats =  0.9760897755622864
p value =  0.010179840028285 98
Not Normally Distibuted
```

In [ ]:

```
# visualize data to see if it is normally ditributed or not
import matplotlib.pyplot as plt
```

```
IR['SepalLengthCm'].hist()
# histogram of SepalLengthCm col also show non normality of data
```

Out[ ]: <AxesSubplot:>

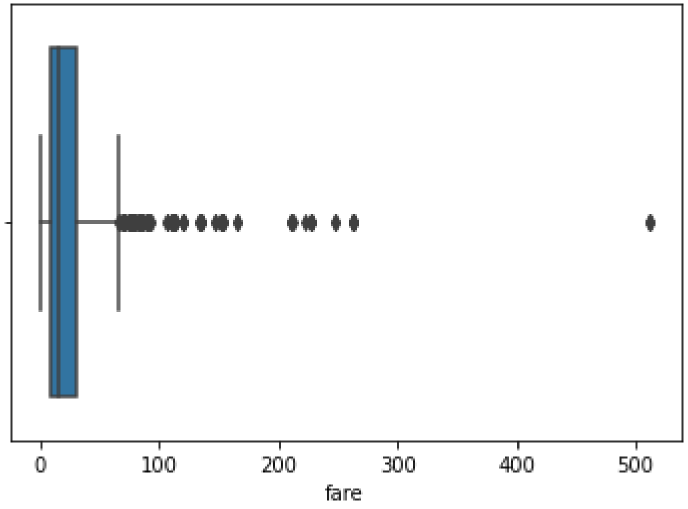


In [ ]:

```
# we can visualize it using boxplot
import seaborn as sns
# load titanic dataset
k= sns.load_dataset('titanic')
#boxplot of fare column from titanic dataset
sns.boxplot(k['fare'])
# shows non normality of fare column
```

```
C:\Users\My Net\AppData\Local\Programs\Python\Python310\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the fol
lowing variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arg
uments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

Out[ ]: <AxesSubplot:xlabel='fare'>



In [ ]:

```
#boxplot of fare column from titanic dataset
stat,p = shapiro(k['fare'])
print('stats = ', stat)
print('p value = ', p)
if p > 0.05:
    print('Normally Distibuted')
else:
    print('Not Normally Distibuted')
```

```
stats =  0.5218914747238159
p value =  1.0789998175301091e-43
Not Normally Distibuted
```

> ## 2. Correlation Test
>
> 1. **Pearson's Correlation** Test is used to check if the two data samples have a linear relationship or not.
>   - Checks observtions in each sample are independent and identically distributed.
>   - Each sample from observation are normally distributed.
>   - Each sample from observation have same variance.

Interpretation:

- **H0**: the two samples are independent

- **H1**: there is a dependency b/w data samples

In [ ]:

```
# Example of the Pearson's Correlation test
from scipy.stats import pearsonr
data1 = [0.873, 2.817, 0.121, -0.945, -0.055, -1.436, 0.360, -1.478, -1.637, -1.869]
data2 = [0.353, 3.517, 0.125, -7.545, -0.555, -1.536, 3.350, -1.578, -3.537, -1.579]
stat, p = pearsonr(data1, data2)
print('stat=%.3f, p=%.3f' % (stat, p))
if p > 0.05:
  print('Probably independent')
else:
  print('Probably dependent')
```

```
stat=0.688, p=0.028
Probably dependent
```

1. **Spearman's Rank Correlation** Test is used to check if the two data samples have a monotonic relationship.
   - observtions in each sample can be ranked.
   - Each sample from observation are independent.

Interpretation:

- **H0**: the two samples are independent.
- **H1**: there is a dependency b/w data samples.

In [ ]:

```
# Example of the Spearman's Rank Correlation test
from scipy.stats import spearmanr
data1 = [0.873, 2.817, 0.121, -0.945, -0.055, -1.436, 0.360, -1.478, -1.637, -1.869]
data2 = [0.353, 3.517, 0.125, -7.545, -0.555, -1.536, 3.350, -1.578, -3.537, -1.579]
stat, p = spearmanr(data1, data2)
print('stat=%.3f, p=%.3f' % (stat, p))
if p > 0.05:
  print('Probably independent(No correlation)')
else:
  print('Probably dependent(correlation exists)')
```

```
stat=0.855, p=0.002
Probably dependent(correlation exists)
```

3. Chi-Squared Test

Chi-Squared Test is used to check if the two categorical variables are related or independent.

Interpretation:

- **H0**: the two samples are independent
- **H1**: there is a dependency b/w data samples.

In [ ]:
```
from scipy.stats import chi2_contingency
table=[12,13,15],[6,9,10]
stat, p,dof,expected = chi2_contingency(table)
print('stat=%.3f, p=%.3f' % (stat, p))
if p > 0.05:
  print('Probably independent')
else:
  print('Probably dependent')
```

```
stat=0.281, p=0.869
Probably independent
```

Parametric Statitical Hypothesis Test

1. Student's t-test

In [ ]:
```
# Example of the Student's t-test
from scipy.stats import ttest_ind
data1 = [0.873, 2.817, 0.121, -0.945, -0.055, -1.436, 0.360, -1.478, -1.637, -1.869]
data2 = [1.142, -0.432, -0.938, -0.729, -0.846, -0.157, 0.500, 1.183, -1.075, -0.169]
stat, p = ttest_ind(data1, data2)
print('stat=%.3f, p=%.3f' % (stat, p))
if p > 0.05:
  print('Probably the same dataset')
else:
  print('Probably different data set')
```

```
stat=-0.326, p=0.748
Probably the same dataset
```

2. Paired Student's t-test

In [ ]:

```
# Example of the Paired Student's t-test
from scipy.stats import ttest_rel
data1 = [0.873, 2.817, 0.121, -0.945, -0.055, -1.436, 0.360, -1.478, -1.637, -1.869]
data2 = [1.142, -0.432, -0.938, -0.729, -0.846, -0.157, 0.500, 1.183, -1.075, -0.169]
stat, p = ttest_rel(data1, data2)
print('stat=%.3f, p=%.3f' % (stat, p))
if p > 0.05:
  print('Probably the same distribution')
else:
  print('Probably different distributions')
```

```
stat=-0.334, p=0.746
Probably the same distribution
```