STATISTICAL CALCULATOR

- SIMI SUDHAKARAN & SANJEEV THAPAR

In [1]:

```
#Importing libraries
import pandas as pd
import numpy as np
import math
import statistics as stats
import plotly.express as px
import seaborn as sns
from matplotlib import pyplot as plt
import plotly.graph_objects as go
```

Taking input for X variables as either manual input or read from an excel

by SIMI SUDHAKARAN

In [47]:

```
class input_collection:
    def __init__(self,list1):
        self.list1 = list1
        #self.list2 = list2

class Statistical_Calculator:

    def input_variable(self):

        x = input("Enter values for x separated by comma(,): ")

        list1 = x.split(",")
        print(list1)
        l1 = []
        for i in list1:
            l1.append(int(i))

#         q1 = input("Do you want to enter y values in order to perform other sta
tistical functions? yes/no: ")

#         l2 = []
#         if q1 == "yes".casefold():
#             y = input("Enter values for y separated by comma(,): ")
#             list2 = y.split(",")
#             print(list2)
#             for i in list2:
#                 l2.append(int(i))

        response = input_collection(l1)
        return response

    def read_excel():
        print("Read excel function")
        excel_file = input("Enter the http url to read the excel file: ")
        #sample excel:  /Users/ss/Saint Peters MSDS/DS542/PROJECT 542/Sample tes
t file.xlsx
        df = pd.read_excel(excel_file)
        df = df.rename(columns={df.columns.values[0]:"X",df.columns.values[1]:
"Y"})
        df.head()
        return df

Stats_Calc = Statistical_Calculator()

a = input("Enter input variables(enter var) or read from excel(enter excel): ")

if a.lower() == "var":
    print("Enter input variables:")
    input_variable = Stats_Calc.input_variable()
    print("List1 - ",input_variable.list1)
    #print("List2 - ",input_variable.list2)
    data_Tuples = list(zip(input_variable.list1))
    df = pd.DataFrame(data_Tuples,columns=['X'])
    print(df)

elif a.lower() == "excel":
    print("Read EXCEL file:")
    df = Statistical_Calculator.read_excel()
    print(df)
```

```
else:
    print("Invalid input")
```

```
Enter input variables(enter var) or read from excel(enter excel): ex
cel
Read EXCEL file:
Read excel function
Enter the http url to read the excel file: /Users/ss/Saint Peters MS
DS/DS542/PROJECT 542/Sample test file.xlsx
      X   Y
0   257  12
1   350  16
2   268  12
3   315  10
4   100   9
5   256  11
6   108   3
7   235   8
8   243  11
9   400  16
10  249  13
11  149  10
12   53   3
13   48   2
14  108   3
15  289  15
16  170  11
17  420  18
18  170  11
19  199  13
20   85   5
21  170  11
22  108   3
23   85   5
24  315  10
```

Calculate the no of observations in the Dataframe

by SIMI SUDHAKARAN

In [34]:

```
nob = df.index.stop
print("Number of observations: ",nob)
```

```
Number of observations:  26
```

Class for generating the Arthmetic Mean

by SIMI SUDHAKARAN

In [35]:

```
class Calculate_ArthmeticMean:

    arth_mean = sum(df['X'])/nob
    print("Manual Arthmetic Mean for X values : ",arth_mean)

#Invoke Arthmetic Mean class
Calculate_ArthmeticMean = Calculate_ArthmeticMean()
```

```
Manual Arthmetic Mean for X values :  144.69230769230768
```

Function for generating the Median

by SIMI SUDHAKARAN

In [36]:

```
df_X = df['X'].sort_values()
df_X = df_X.reset_index(drop=True)
#print(df_X)

class Calculate_Median:
    def Calculate(self,df_set):
        if (len(df_set)/2)%2 == 0:
            #print("Even number of observations")
            mid_1 = math.floor((len(df_set)-1)/2)
            mid_2 = mid_1+1
            Median = (df_set[mid_1] + df_set[mid_2])/2
            return Median
        else:
            #print("Odd number of observation ")
            Median = df_set[round(len(df_set)/2)]
            return Median


Calculate_Median =Calculate_Median()
Q2 = Calculate_Median.Calculate(df_X)
print("Q2/Median : ", Q2)
```

```
Q2/Median :  80
```

Calculate Quartiles(Q1 & Q3)

by SIMI SUDHAKARAN

In [37]:

```
   #Code for testing the even nob
    ##########################################
#     df_X = df_X.drop([24])
#     print(df_X)
#     nob = len(df_X)
#     print("Number of observations: ",nob)
    ##########################################
df_X_1 = df_X.iloc[:round((len(df_X)/2))]
df_X_2 = df_X.iloc[round((len(df_X)/2)):len(df_X)]

if len(df_X)%2 != 0:
    #print("odd")
    df_Q2 = pd.Series([Q2])
    df_X_1 = df_X_1.append(df_Q2,ignore_index=True)

df_X_2 = df_X_2.reset_index(drop=True)

#print(df_X_1)
#print("************")
#print(df_X_2)

Q1 = Calculate_Median.Calculate(df_X_1)
#print("$$$$$$$$$$$$$")
Q3 = Calculate_Median.Calculate(df_X_2)
print("Q1/Quartile 1 : ",Q1)
print("Q2/Quartile 2 : ",Q3)
```

```
Q1/Quartile 1 :  43
Q2/Quartile 2 :  304
```

Class for generating the Mode(and check for unimodal & bimodal outputs

by SIMI SUDHAKARAN

In [38]:

```
class Calculate_Mode:
    count = df['X'].value_counts()
    freq = count.to_dict()
    #print("Frequency: ",freq)
    Mode = []
    maxi = 0
    for key,value in freq.items():
        #print(key ,":", value)
        if value >= maxi:
            #print("value is greater than max",maxi,value)
            maxi = value
            #print(key,value)
            Mode.append(key)
    #print("Mode/s: ",Mode)

    if len(Mode)== 1:
        print("Unimodal : ",Mode)
    elif len(Mode)==2:
        print("Bimodal : ",Mode)
    else:
        print("No Mode present")
#     elif len(Mode) >= 3:
#         print("Multimodal : ",Mode)


Calculate_Mode =Calculate_Mode()
```

```
Unimodal :  [78]
```

Class for determining the smallest value

by SANJEEV THAPAR

In [39]:

```
smallest = None
class small:

        #print('Before:', smallest)

        for itervar in df['X']:

            if smallest is None or itervar < smallest:

                smallest = itervar

                #print('Loop:', itervar, smallest)

        print('Smallest:', smallest)



small=small()
```

```
Smallest: 12
```

Class for determining the largest value

by SANJEEV THAPAR

In [40]:

```
largest = None
class large:
        #print('Before:', largest)

        for itervar in df['X']:

            if largest is None or itervar > largest :

                largest = itervar

                #print('Loop:', itervar, largest)

        print('Largest:', largest)

large=large()
```

```
Largest: 405
```

Class for determining the Standard Deviation & Variance

by SANJEEV THAPAR

In [42]:

```
# Python3 code to demonstrate working of

# Standard deviation of list

# Using sum() + list comprehension
variance = None
res = None
class standard:

# initializing list

    test_list = input_variable.list1
    # printing list
    #print("The original list : " + str(test_list))
    # Standard deviation of list
    # Using sum() + list comprehension
    #mean = sum(test_list) / len(test_list)
    variance = sum([((x - Calculate_ArthmeticMean.arth_mean) ** 2) for x in test
_list]) / len(test_list)
    res = variance ** 0.5
    # Printing result
    print("Variance of sample is : ", variance)
    print("Standard deviation of sample is : " + str(res))


standard=standard()
```

```
Variance of sample is :  20768.751479289942
Standard deviation of sample is : 144.11367554569532
```

5 point summary using Libraries:

by SIMI SUDHAKARAN & SANJEEV THAPAR

In [43]:

```
Lib_Mean = np.mean(df['X'])
print("Lib_Mean - ",Lib_Mean)
Lib_Median = np.median(df['X'])
print("Lib_Median - ",Lib_Median)
Lib_Mode = stats.multimode(df['X'])
print("Lib_Mode - ",Lib_Mode)
Lib_Min = np.min(df['X'])
print("Lib_Min - ",Lib_Min)
Lib_Max = np.max(df['X'])
print("Lib_Max - ",Lib_Max)
Lib_Q1 = np.quantile(df['X'], .25)
print("Lib_Q1 - ",Lib_Q1)
Lib_Q3 = np.quantile(df['X'], .75)
print("Lib_Q3 - ",Lib_Q3)
Lib_Variance = stats.variance(df['X'])
print("Lib_Variance - ",Lib_Variance)
Lib_StdDev = stats.stdev(df['X'])
print("Lib_StdDev - ",Lib_StdDev)
```

```
Lib_Mean -  144.69230769230768
Lib_Median -  79.0
Lib_Mode -  [78]
Lib_Min -  12
Lib_Max -  405
Lib_Q1 -  46.25
Lib_Q3 -  253.5
Lib_Variance -  21599.50153846154
Lib_StdDev -  146.96768875661596
```

Statistical Calculator summary table:

by SIMI SUDHAKARAN

In [44]:

```
fig = go.Figure(data=[go.Table(
    header=dict(values=['<b>STATISTICAL FUNCTION</b>','<b>MANUAL</b>', '<b>LIBRA
RY</b>'],
                line_color='darkslategray',
                fill_color='paleturquoise',
                align='center'),
    cells=dict(values=[["Number of Observations", "Arthmetic Mean", "Quartile_1
(Q1)","Median(Q2)",
                        "Quartile_3(Q3)","Mode","Min Value", "Max Value","Varian
ce","Standard Deviation"], #1st column
                       [nob,Calculate_ArthmeticMean.arth_mean,Q1,Q2,Q3,Calculate
_Mode.Mode,small.smallest,
                        large.largest,standard.variance,standard.res], #2nd Colu
mn
                      [nob,Lib_Mean,Lib_Q1,Lib_Median,Lib_Q3,Lib_Mode,Lib_Min,
                       Lib_Max,Lib_Variance,Lib_StdDev]], # 3rd column
               line_color='darkslategray',
               fill_color='lavender',
               align='center'))
])

fig.update_layout(width=800, height=600)
fig.show()
```

Function for displaying the 5-point summary on a distribution plot

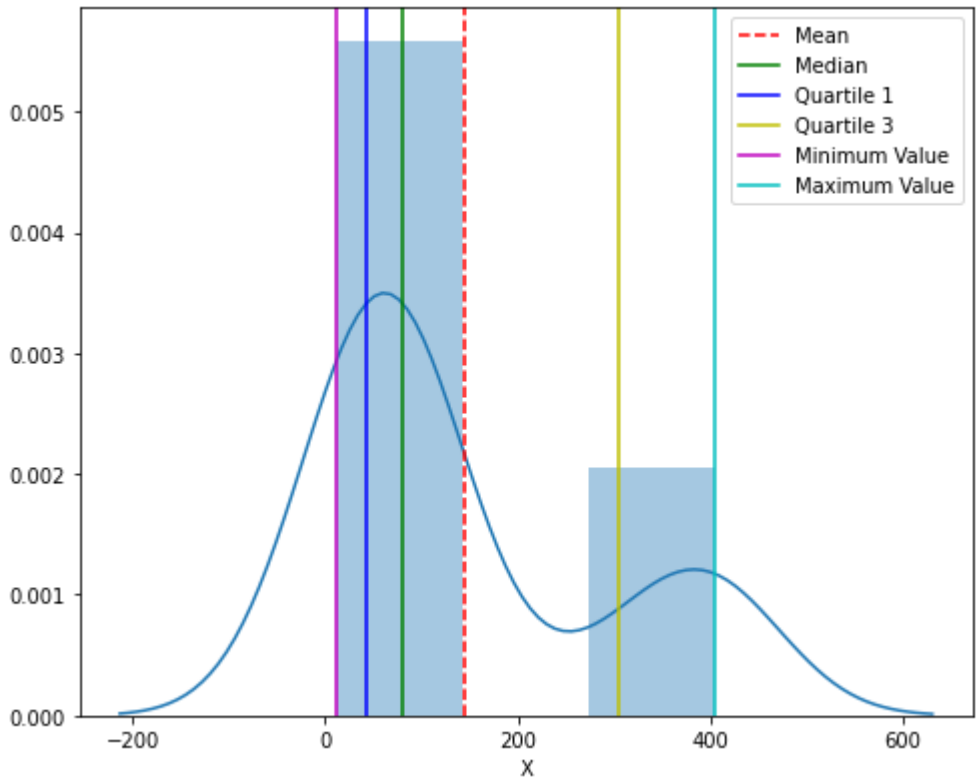
by SIMI SUDHAKARAN

In [45]:

```
class Generate_Graph:
    plt.figure(figsize = [8.0,6.5])
    sns.distplot(df['X'])
    plt.axvline(Calculate_ArthmeticMean.arth_mean, color='r', linestyle='--')
    plt.axvline(Q2, color='g', linestyle='-')
    plt.axvline(Q1, color='b', linestyle='-')
    plt.axvline(Q3, color='y', linestyle='-')
    plt.axvline(small.smallest, color='m', linestyle='-')
    plt.axvline(large.largest, color='c', linestyle='-')

    plt.legend({'Mean':Calculate_ArthmeticMean.arth_mean,'Median':Q2,'Quartile
 1':Q1,'Quartile 3':Q3,
                'Minimum Value':small.smallest,'Maximum Value':large.largest})
    plt.show()

Generate_Graph =Generate_Graph()
```



Violin plot with an in shown boxplot to display all the datapoints and 5 point summary

by SIMI SUDHAKARAN

In [46]:

```
fig = px.violin(df['X'], y="X", box=True, points='all')
fig.show()
```

In [ ]:

In [ ]:

In [ ]: