Saint Louis University

SCHOOL OF ACCOUNTANCY, MANAGEMENT, COMPUTING AND INFORMATION STUDIES

Department of Computer Science

CS 322 9345 Data Science

Prelim Lab Activity : Descriptive Statistics

Submitted To

Beverly Estephany Ferrer

Submitted by

BERNABE, Rabelais
LLENA, Anthony

January 2026

Table of Contents

General Instructions

Objectives

- To determine the different types of measures from a data set in order to provide basic statistical information
- To apply basic descriptive statistics
- To make the appropriate data representation in tabular graphical form

Tool

- Dataset with sufficient quantitative data
- Dataset with sufficient quantitative data
- Google Sheets or Microsoft Excel
- Python

Requirements

1. Find a dataset that contains different types of data (numeric, text, etc). The dataset must have at least 100 records.
2. Use Excel or Google sheets feature to get the following basic information:
    a. Different types of measures such as mean, median, mode, variance, and standard deviation
    b. Appropriate data visualization; ensure that these are labelled properly; provide annotations as needed.
3. Use Python to accomplish the same task above.
    a. Use the attached 322 Supplementary Exercise.docx to aid you in using Python.

Introduction

Descriptive Statistics present and interpret data to provide a comprehensive understanding of past events, patterns, and trends. It has been used by analysts to characterize the populations and samples of a certain demography. It is the collection of measurements of location and variability with the use of the central value of a variable and the spread of the data from the center value. It involves summarizing and organization of data with the central tendency or variation as it is also supported by graphs.

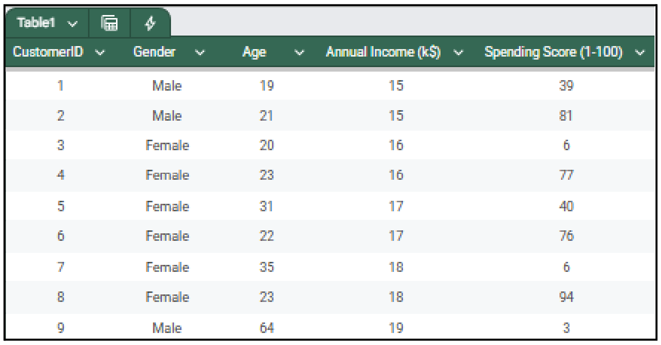
The measures of Central Tendency fall under the descriptive statistics with numerical values that tend to locate the middle of a set of data when arranged in increasing or decreasing order. Such terminologies exist and are often associated with the measures of mean, median, mode, midrange, variance, and standard deviation. In this preliminary activity, the focus stress on the measure of central tendency to provide an analysis of the dataset of the *Mall_Customers.csv*

Methodology

Data Acquisition and Preparation

Dataset: Mall Customers Segmentation

The Mall_Customers.csv dataset consists of 5 columns and 201 is verified to ensure it meets the minimum requirement of at least 100 records as statrows ed in the given instruction. For this specific study, 201 rows consisting of the header and 200 customer entries, providing a robust sample for statistical significance. The five columns are CustomerID, Gender Age, Annual Income(k$), and Spending Score (1-100) to represent a mix of categorical and numerical data thus giving way for a multifaceted evaluation of the customer behaviour.



CustomerID	Gender	Age	Annual Income (k$)	Spending Score (1-100)
1	Male	19	15	39
2	Male	21	15	81
3	Female	20	16	6
4	Female	23	16	77
5	Female	31	17	40
6	Female	22	17	76
7	Female	35	18	6
8	Female	23	18	94
9	Male	64	19	3

Figure 1: Dataset of Mall_Customers.csv

Supplementary Material for Coding in Python

Python Script	Output
Python #import statistical libraries import math import statistics import numpy as np import pandas as pd	

Python
#Compute for the mean/average of the given data
statistics.mean([4,3,7,6,5,2])
	4.5

Python
#Compute for the mean; get the sum of data divided divided by the length of the data
data = ([4,3,7,6,5,2])
mean_=sum(data)/len(data)
mean_
	4.5

Python
from decimal import Decimal as D
statistics.mean([D("0.5"), D("0.75"), D("0.625"), D("0.375")])
	Decimal('0.5625')

Python
#Convert data to floats and compute the arithmetic mean.
statistics.fmean([1.5, 2.0, 2.8, 0.9, 2.25])
	1.89

Python
#median high - When is even, the larger of the two middle values is returned.
data=([7, 3, 6, 5, 2, 1])
statistics.median_high(data)
	5

Python statistics.mode(["apple", "orange", "apple", "grapes", "grapes", "apple"])	**‘apple’**
Python statistics.mode([20, 15 , 12, 17, 18, 15, 20, 20, 15])	**20**
Python # A dataset may contain nan dataset. # Nan means a 'not-a-number value' # In data science, Nan is used to replace missing values in actual datasets	**[1, 3, 2, 4.5, 8.0]**

# In Python, you can use any of the folling: float('nan'), math.nan, np.nan 	
 	
x= [1, 3, 2, 4.5, 8.0]
x_with_nan = [1.0, 3, 2, math.nan, 4.5, 8.0]
x	

Python
x = [1.0, 3, 2, 4.5, 8.0]
statistics.mean(x)
	3.7

Python
x_with_nan = [1.0, 3, 2, math.nan, 4.5, 8.0]
statistics.mean(x_with_nan)
	nan

Python
`#Use NumPy to create a one-dimensional array`
`#Load library import nump as np`
 	
`#Create a vector as a row`
`vector_row=np.array([1, 2, 3])`
 	
`#create a vector as a column`
`vector_column=np.array([[1], [2], [3]])`
	array([[1],
	[2],
	[3]])

Python
`#using NumPy`
`a = np.array([1, 2, 3, 4,])`
`print('Our Array is: ')`
`print (a)`
	Our Array is:
	[1 2 3 4]

Python
`a = np.array([1,2,3,4])`
`np.mean(a)`
	np.float64(2.5)

Python
a = np.array([[1,2],[3,4]])
np.mean(a)
	np.float64(2.5)

Python np.mean(a, axis=0)	**array([2., 3.])**
Python np.mean(a, axis=1)	**array([1.5, 3.5])**

Python
a = np.array([1,2,3,4,5])
np.median(a)
	np.float64(3.0)

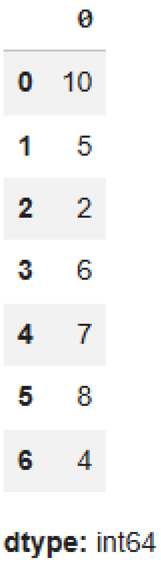
Python
a = np.array([1,2,3,4])
np.median(a)
	np.float64(2.5)

Python
from scipy import stats
 	
a=([20, 15, 12, 17, 18, 15, 20, 20])
stats.mode(a)
	ModeResult(mode=np.int64(20), count=np.int64(3))

Python
#Computing the weighted mean
a=[5, 2, 7, 2]
b=[1, 2, 3, 4]
c=[5, 9, 3, 7]
a,b,c = np.array(a), np.array(b),
np.array(c)
wmean = np.average(a, weights=b)
wmean
	np.float64(3.8)

Python
a=[5, 2, 7, 2]
b=[1, 2, 3, 4]
c=[5, 9, 3, 7]
a,b,c=np.array(a), np.array(b), np.array(c)
wmean = np.average(c, weights=b)
	np.float64(6.0)

```
Python
#Using pandas
import pandas as pd

data = pd.Series([10, 5, 2, 6, 7, 8, 4])
data
```



```
   0
0 10
1  5
2  2
3  6
4  7
5  8
6  4

dtype: int64
```

```
Python
#displays the elements of array data
data.values
```

```
array([10,  5,  2,  6,  7,  8,  4])
```

```
Python
#element at index
data[1]
```

```
np.int64(5)
```

```
Python
#element at index 4
```

```
np.int64(7)
```

data[4]	

Python
#using another index
data = pd.Series([9.25, 10.5, 8.75, 1.3],
index=['w', 'x', 'y', 'z'])
data
	0
	w 9.25
	x 10.50
	y 8.75
	z 1.30
	dtype: float64

Python
#Customizing the index
pd.Series(5, index=[100, 200, 300])
	0
	100 5
	200 5
	300 5
	dtype: int64

Python
#Default index
m = pd.Series([1, 2, 3, 4])
statistics.mean(m)
	2.5

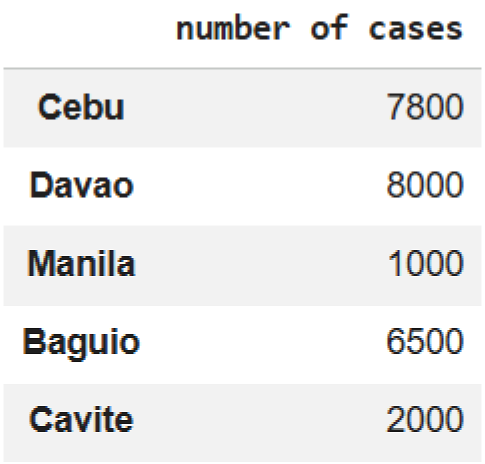
```
Python
#Customizing the index
pd.Series({2:'a', 1:'b', 3:'c'})
```

```
0

2    a

1    b

3    c

dtype: object
```

```
Python
covid_dict={'Cebu':7800, 'Davao':8000,
'Manila':1000, 'Baguio':6500, "Cavite":2000}
covid = pd.Series(covid_dict)
covid
```

```
0

Cebu      7800

Davao     8000

Manila    1000

Baguio    6500

Cavite    2000

dtype: int64
```

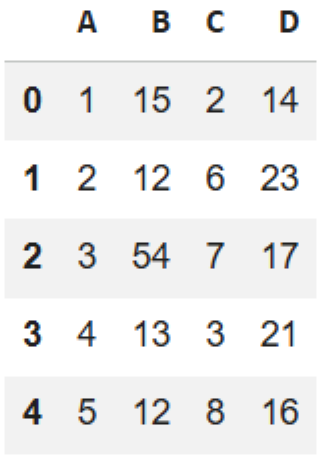
```
Python
#use of dataframe
covid_dict=pd.DataFrame({'number of
cases': covid})
covid_dict
```



	number of cases
Cebu	7800
Davao	8000
Manila	1000
Baguio	6500
Cavite	2000

Python

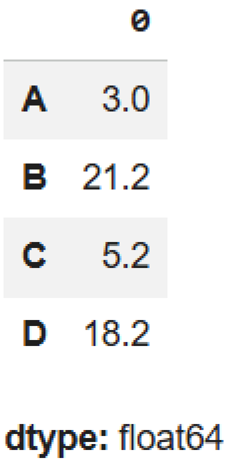
```python
 #Creating the dataframe
df=pd.DataFrame({"A":[1, 2, 3, 4, 5], "B":[15,
12, 54, 13, 12], "C":[2, 6, 7, 3, 8], "D":[14, 23,
17, 21, 16]})
df
```



```
   A   B  C   D
0  1  15  2  14
1  2  12  6  23
2  3  54  7  17
3  4  13  3  21
4  5  12  8  16
```

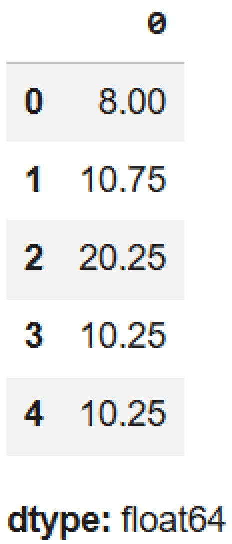
Python

```python
#Compute the mean along rows
df.mean()
```



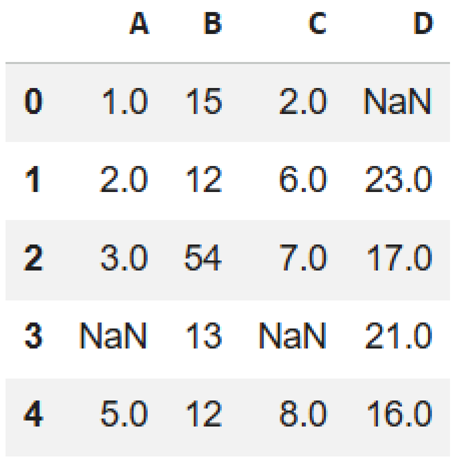
```
      0
A   3.0
B  21.2
C   5.2
D  18.2

dtype: float64
```

```
Python
#Computes the mean along rows
df.mean(axis=0)

#Computes the mean along columns
df.mean(axis=1)
```



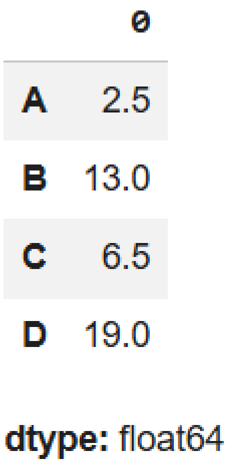
```
       0
0   8.00
1  10.75
2  20.25
3  10.25
4  10.25

dtype: float64
```

```
Python
#Creating the dataframe
df=pd.DataFrame({"A":[1, 2, 3, None, 5],
"B":[15, 12, 54, 13, 12], "C":[2, 6, 7, None, 8],
"D":[None, 23, 17, 21, 16]})
df
```



```
     A   B    C     D
0  1.0  15  2.0   NaN
1  2.0  12  6.0  23.0
2  3.0  54  7.0  17.0
3  NaN  13  NaN  21.0
4  5.0  12  8.0  16.0
```

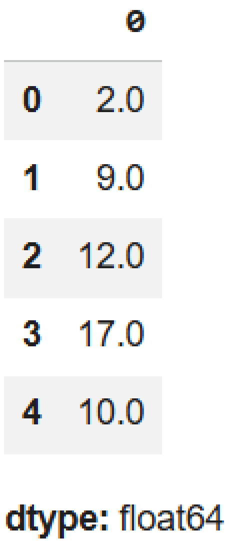
```
Python
#Computing the median using pandas
#Median along rows
df.median(axis=0)

df.median()
```



```
      0
A   2.5
B  13.0
C   6.5
D  19.0

dtype: float64
```

Python

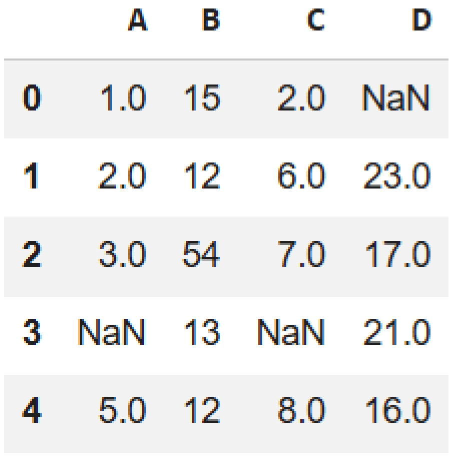
```python
#Median along columns
df.median(axis=1)
```



```
      0
0   2.0
1   9.0
2  12.0
3  17.0
4  10.0

dtype: float64
```

Python

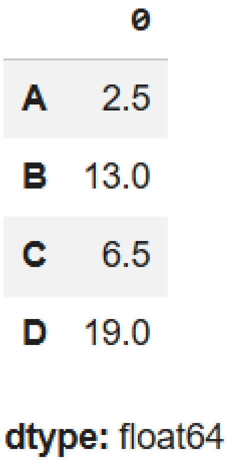
```python
#Creating the dataframe
df=pd.DataFrame({"A":[1, 2, 3, None, 5],
"B":[15, 12, 54, 13, 12], "C":[2, 6, 7, None, 8],
"D":[None, 23, 17, 21, 16,]})

df
```



```
     A   B    C     D
0  1.0  15  2.0   NaN
1  2.0  12  6.0  23.0
2  3.0  54  7.0  17.0
3  NaN  13  NaN  21.0
4  5.0  12  8.0  16.0
```

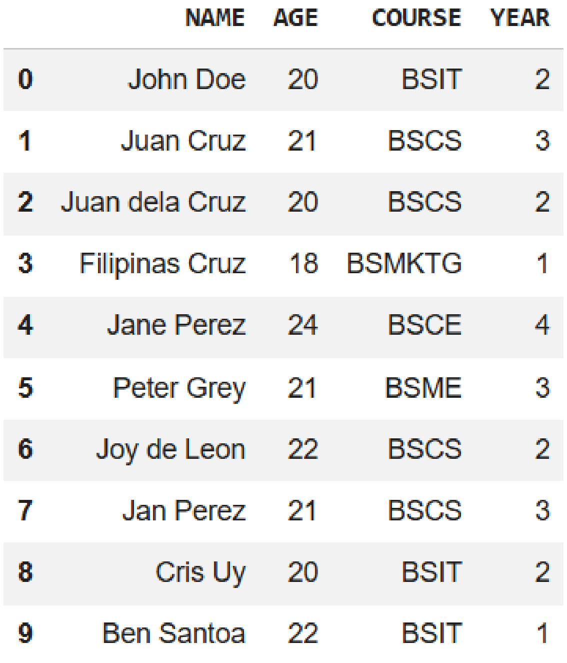
Python

```python
#finding the mean, skip the Nan values
df.median(axis=0, skipna=True)
```



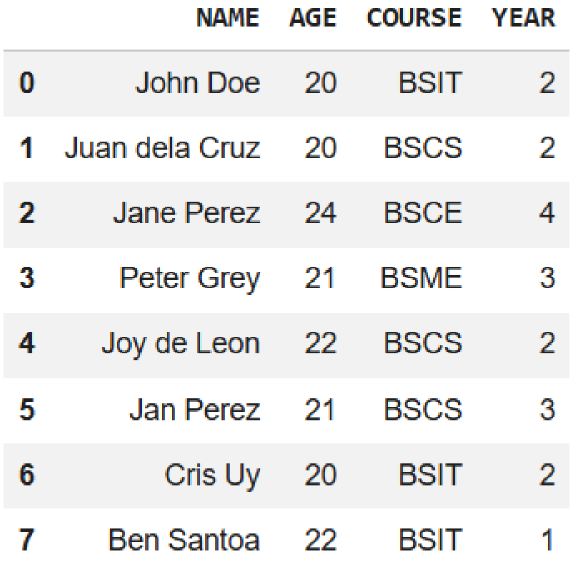
```
      0
A   2.5
B  13.0
C   6.5
D  19.0

dtype: float64
```

Python
```python
#reading csv file
pd.read_csv("SAMPLE.csv")
```

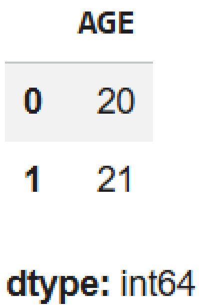


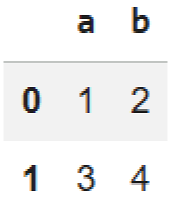
```
              NAME  AGE  COURSE  YEAR
0         John Doe   20    BSIT     2
1        Juan Cruz   21    BSCS     3
2   Juan dela Cruz   20    BSCS     2
3   Filipinas Cruz   18  BSMKTG     1
4       Jane Perez   24    BSCE     4
5       Peter Grey   21    BSME     3
6      Joy de Leon   22    BSCS     2
7        Jan Perez   21    BSCS     3
8          Cris Uy   20    BSIT     2
9       Ben Santoa   22    BSIT     1
```

Python
```python
#skips the passed rows in a new series
pd.read_csv("SAMPLE.csv", skiprows=[2,4])
```



```
              NAME  AGE  COURSE  YEAR
0         John Doe   20    BSIT     2
1   Juan dela Cruz   20    BSCS     2
2       Jane Perez   24    BSCE     4
3       Peter Grey   21    BSME     3
4      Joy de Leon   22    BSCS     2
5        Jan Perez   21    BSCS     3
6          Cris Uy   20    BSIT     2
7       Ben Santoa   22    BSIT     1
```

Python
#reading a csv file
df_sample = pd.read_csv("SAMPLE.csv")
 	
df_sample['AGE'].mode()
	
	AGE
	0 20
	1 21
	dtype: int64

Python
df = pd.DataFrame([[1, 2], [3, 4]],
columns=["a", "b"])
df
	
	a b
	0 1 2
	1 3 4

Python
#find mean of column "a" and "b"
df_mean = df[["a", "b"]].mean()
print(df_mean)
	a 2.0
	b 3.0
	dtype: float64

Python
#Arbitrary Data
v = [10, 14, 16, 11, 12 ,18]
var_x = statistics.variance(x)
var_x
	7.45

Python
#If data includes nan values, then statistics variance() will return nan\
v_nan=[10, 14, 16, 11, math.nan, 12, 18]
v_nan
	[10, 14, 16, 11, nan, 12, 18]

Python
var_nan = statistics.variance(v_nan)
print("Variance with Nan:")
var_nan
	Variance with Nan:
	nan

	Variance of Sample:

Python #Calculating the sample variance with NumPy #Use the function np.var() or the corresponing method .var() #Note: ddof=1 - allows calculation of s**2, with (n-1) in the denominator instaed of n #Since we are calbulating the variance of a sample, not a population var_sample = [3, 5, 7, 2] var_s = np.var(var_sample, ddof=1) print("Variance of Sample: ") var_s	**np.float64(4.916666666666667)**

Python
#Nan values in the dataset -> use np.var() or .var()
#ddof parameter in Python stands for Delta Degrees of Freedom
#this is used when you are working with a sample of a larger population.
#Delta Degree of Freedom refers to how many values are allowed to change freely
np.var(v_nan, ddof=1)
	np.float64(nan)

	np.float64(9.5)

Python
#Skip nan values, use np.nanvar()
np.nanvar(v_nan, ddof=1)	

Python
#population variance
#Use statistics.pvariance(); ddof=0, not ddof=1
x=[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
var_x = np.var(x, ddof=0)
print("Variance of population:")
var_x
	Variance of population:
	np.float64(8.25)

Python
#standard deviation
sd = statistics.stdev(x)
print("Standard Deviation:")
sd
	Standard Deviation:
	3.0276503540974917

Python
#using NumPy
y=[2,7,4,5,8]
np.std(y, ddof=1)
	np.float64(2.3874672772626644)

Python
#population standard deviation
#Use statistics.pstdev()
x=[5, 7, 9, 3, 7, 2, 8, 1, 9, 11, 13, 11]
sd=statistics.pstdev(x)
print("Standard deviation:")
sd
	Standard deviation:
	3.624760528488591

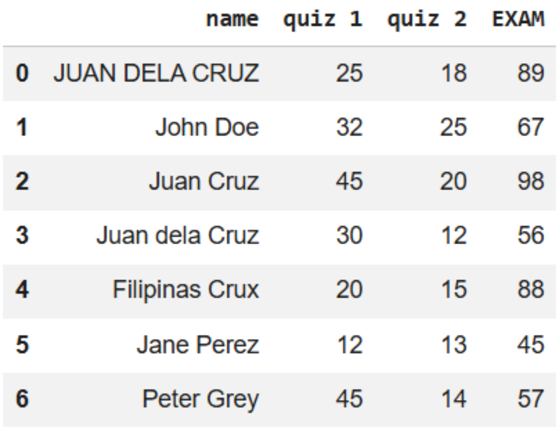
Python
#Range
m=np.ptp(x)
print(m)
	12

Python
a = [102, 128, 131, 98, 140, 93, 115, 109, 89, 119, 97]
np.percentile(a, [25, 50, 75])
	array([97.5, 109. , 123.5])

Python
#interquartile range
quartiles = np.quantile(y, [0.25, 0.75])
quartiles[1] - quartiles[0]
	np.float64(3.0)

Python
#interquartile range
quartiles = np.quantile(y,[0.25, 0.75])
quartiles[1] - quartiles[0]
	np.float64(3.0)

```python
r=pd.read_csv("CLASS.csv")
r
```



	name	quiz 1	quiz 2	EXAM
0	JUAN DELA CRUZ	25	18	89
1	John Doe	32	25	67
2	Juan Cruz	45	20	98
3	Juan dela Cruz	30	12	56
4	Filipinas Crux	20	15	88
5	Jane Perez	12	13	45
6	Peter Grey	45	14	57

```python
r["quiz 1"].var()
```

```
150.47619047619048
```

```python
r["quiz 2"].var()
```

```
21.238095238095234
```

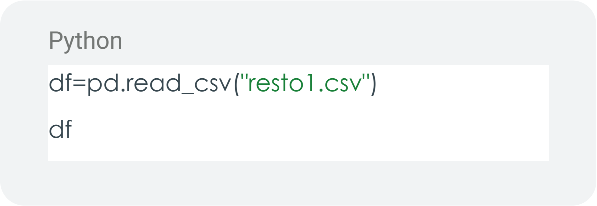
```python
r["EXAM"].var()
```

```
408.95238095238096
```

```python
r["quiz 1"].std()
```

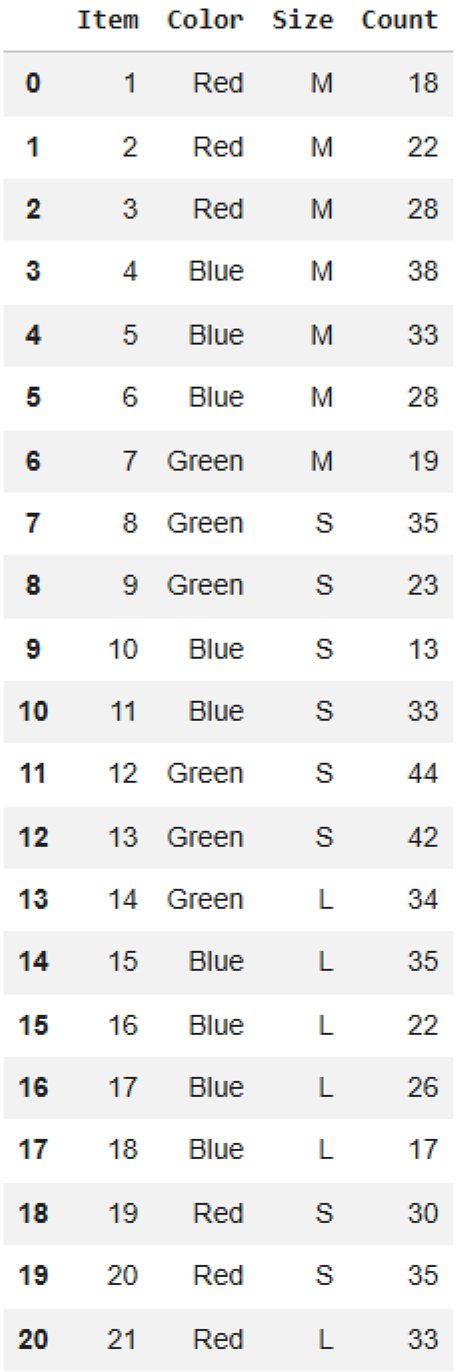
```
12.266873704256945
```

Python `r["quiz 2"].std()`	**4.6084807950229365**
Python `r["EXAM"].std()`	**20.222571076704885**



Python

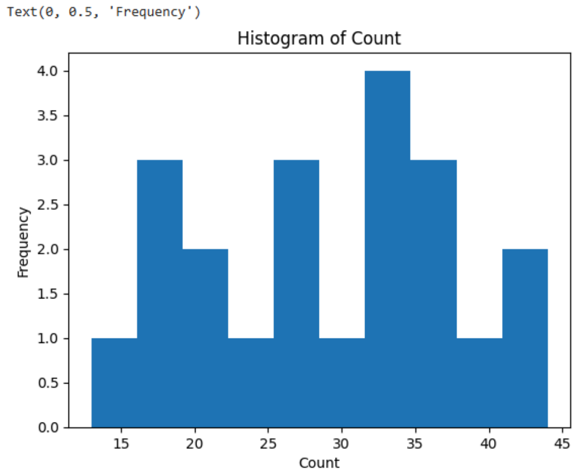
```python
df=pd.read_csv("resto1.csv")
df
```



```
    Item  Color  Size  Count
 0     1    Red     M     18
 1     2    Red     M     22
 2     3    Red     M     28
 3     4   Blue     M     38
 4     5   Blue     M     33
 5     6   Blue     M     28
 6     7  Green     M     19
 7     8  Green     S     35
 8     9  Green     S     23
 9    10   Blue     S     13
10    11   Blue     S     33
11    12  Green     S     44
12    13  Green     S     42
13    14  Green     L     34
14    15   Blue     L     35
15    16   Blue     L     22
16    17   Blue     L     26
17    18   Blue     L     17
18    19    Red     S     30
19    20    Red     S     35
20    21    Red     L     33
```

Python

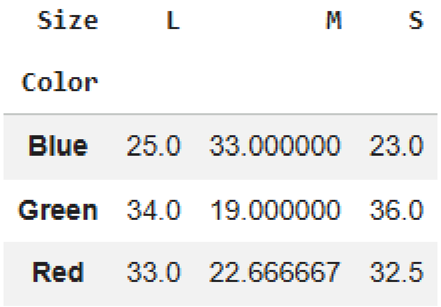
```python
#Generate histogram for Color
plot_histogram=df['Count'].plot.hist(title="Hi
stogram of Count")
plot_histogram.set_xlabel("Count")
plot_histogram.set_ylabel("Frequency")
```



```
Text(0, 0.5, 'Frequency')
```

Python

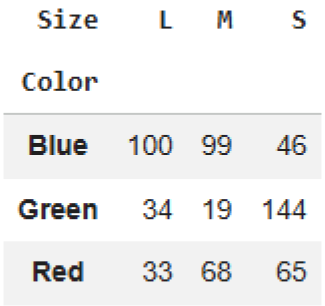
```python
table = pd.pivot_table(
    df, values='Count',
    index=['Color'],  #rows
    columns=['Size'], #columns
    aggfunc=np.mean)  #defines how to
combine values when multiple rows match
table
```



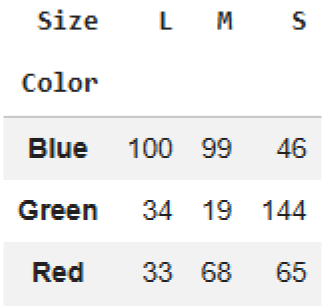
```
 Size      L          M     S
Color
 Blue   25.0  33.000000  23.0
Green   34.0  19.000000  36.0
  Red   33.0  22.666667  32.5
```

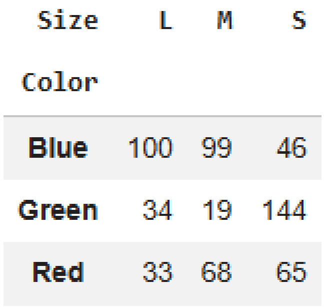
Python

```python
table=pd.pivot_table(
    df, values='Count',
    index=['Color'],
    columns=['Size'],
    aggfunc=np.sum
)
table
```



```
 Size    L   M    S
Color
 Blue  100  99   46
Green   34  19  144
  Red   33  68   65
```

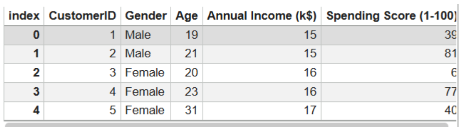
Python
table = pd.pivot_table(
    df,
    values='Count',
    index='Color',
    columns='Size',
    aggfunc="sum"
)
	
	Size L M S
	Color
	Blue 100 99 46
	Green 34 19 144
	Red 33 68 65

Python
table=pd.pivot_table(
    df,
    values='Count',
    index=['Color'],
    columns=['Size'],
    aggfunc="sum"
)
table
	
	Size L M S
	Color
	Blue 100 99 46
	Green 34 19 144
	Red 33 68 65

End of Supplementary Exercise	

Python Code for Descriptive Analytics

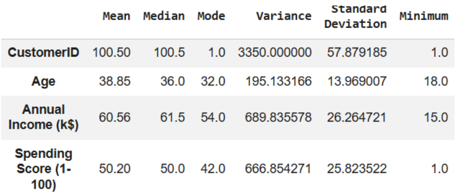
Python Code	Results	Interpretation
Python # @title		Import essential Python Libraries needed for analysis. **Pandas** is used for data loading, cleaning, and computation. **Numpy** supports numerical operations and mathematical calculations. **Matplotlib** is used to create

<table>
<tr>
<td>import pandas as pd
import numpy as np
import matplotlib.pyplot as plt</td>
<td></td>
<td>graphical representations such as histograms and plots.</td>
</tr>
<tr>
<td>Python
df = pd.read_csv("Mall_Customers.csv")
df.head() #.head() is a function that returns the first five rows in a dataset; function and df shows all the complete rows</td>
<td>
<table>
<tr><th>index</th><th>CustomerID</th><th>Gender</th><th>Age</th><th>Annual Income (k$)</th><th>Spending Score (1-100)</th></tr>
<tr><td>0</td><td>1</td><td>Male</td><td>19</td><td>15</td><td>39</td></tr>
<tr><td>1</td><td>2</td><td>Male</td><td>21</td><td>15</td><td>81</td></tr>
<tr><td>2</td><td>3</td><td>Female</td><td>20</td><td>16</td><td>6</td></tr>
<tr><td>3</td><td>4</td><td>Female</td><td>23</td><td>16</td><td>77</td></tr>
<tr><td>4</td><td>5</td><td>Female</td><td>31</td><td>17</td><td>40</td></tr>
</table></td>
<td>Loads the Dataset or <i>Mall_Customer.csv</i> file into a DataFrame named df. The head() function displays the first five rows of the dataset, allowing initial inspection of the data structure, column names, and sample values.</td>
</tr>
<tr>
<td>Python
numeric_cols = df.select_dtypes(include=[np.number]).columns.tolist()</td>
<td>(['CustomerID', 'Age', 'Annual Income (k$)', 'Spending Score (1-100)'],
 ['Gender'])</td>
<td>Separates the variable into the dataset into numeric and categorical columns based on their datatypes to further determine which variable should be mathematically processed. Numeric columns are suitable for calculations while categorical columns represent qualitative data and labels.</td>
</tr>
</table>

```
categorical_cols = df.select_dtypes(exclude=[np.number]).columns.tolist()

numeric_cols, categorical_cols
```

Python

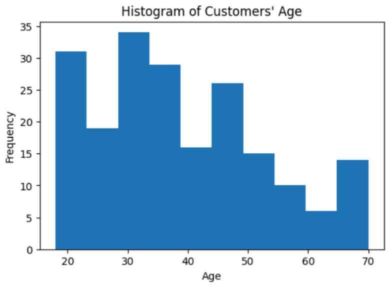
```
# Numeric dataframe
num_df = df[numeric_cols]

# Main descriptive stats
stats_table = num_df.agg(['mean', 'median', 'var', 'std', 'min', 'max']).T

# Extra measures
stats_table['range'] = stats_table['max'] - stats_table['min']
stats_table['mode'] = num_df.mode().iloc[0] # first mode per column
```

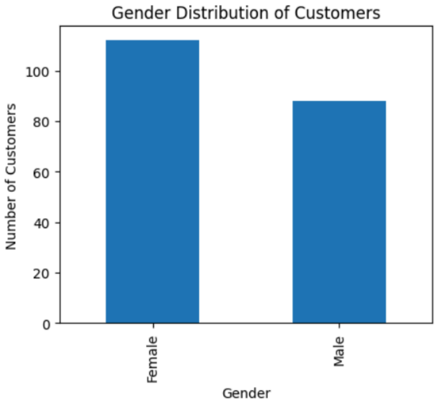


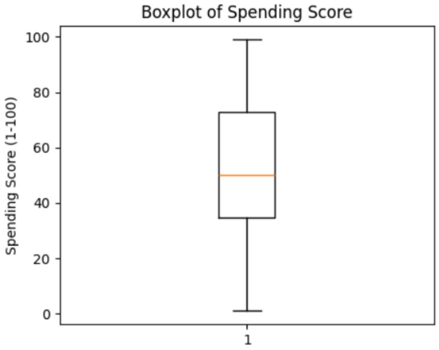
	Mean	Median	Mode	Variance	Standard Deviation	Minimum
CustomerID	100.50	100.5	1.0	3350.000000	57.879185	1.0
Age	38.85	36.0	32.0	195.133166	13.969007	18.0
Annual Income (k$)	60.56	61.5	54.0	689.835578	26.264721	15.0
Spending Score (1-100)	50.20	50.0	42.0	666.854271	25.823522	1.0

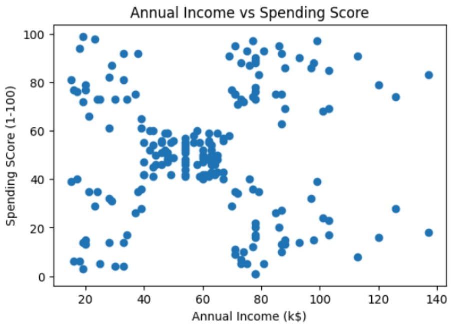
Generates a comprehensive table of descriptive statistics for all numeric variables in the dataset. This calculates the measures of central tendency such as the mean, median, and mode; with measures of dispersion (variance, standard deviation, and range), and measures of position (minimum, maximum, and midrange).

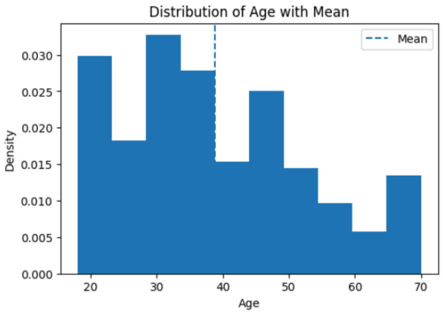
```
stats_table['midrange'] = (stats_table['min']+stats_table['max'])/2

# Arrange columns
stats_table = stats_table[['mean', 'median', 'mode', 'var', 'std', 'min', 'max', 'range', 'midrange']]

# Renmame for neat output
stats_table = stats_table.rename(columns={
    'mean': 'Mean',
    'median': 'Median',
    'mode': 'Mode',
    'var': 'Variance',
    'std': 'Standard Deviation',
    'min': 'Minimum',
    'max': 'Maximum',
    'range': 'Range',
    'midrange': 'Midrange'
})

stats_table
```

Python
plt.figure(figsize=(6, 4)) # Creates a new graph window; 6 = width of the graph in inches, 4 = height of the graph in inches; bigger numbers = bigger graphs
plt.hist(df["Age"], bins=10) # tells Python to draw a histogram; the data being plotted is the age column; splits the values into 10 groups(intervals)
plt.xlabel("Age") # Labels the x-axis or the ABSCISSA
plt.ylabel("Frequency") # Labels the y-axis or the ORDINATE
plt.title("Histogram of Customers' Age") # Gives the graph a title]
plt.show() # Displays the graph
		Creates a histogram to visualize the distribution of customer ages. The abscissa represents age groups, while the ordinate shows the frequency or number of customers in each group. The histogram indicates that most of the customers belongs in the young to middle aged categories, with fewer ages belonging into a younger and older ages range.

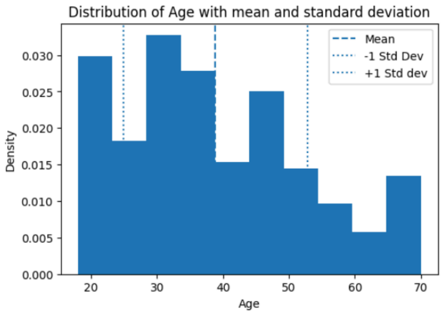
Python
`plt.figure(figsize=(5, 4)) # Creates a new graph window`
`df["Gender"].value_counts().plot(kind="bar") # Counts how many times each gender appears`
`plt.xlabel("Gender")`
`plt.ylabel("Number of Customers")`
`plt.title("Gender Distribution of Customers")`
`plt.show`
		Generates a bar chart to show the distribution of customers by gender. Each bar represents the number of Female and Male that can be seen in the abscissa. The height of the bar which can be seen in the ordinate allows for easy comparison between genders.

Python
`plt.figure(figsize=(5, 4))`
`plt.boxplot(df["Spending Score (1-100)"])`
`plt.ylabel("Spending Score (1-100)")`
`plt.title("Boxplot of Spending Score")`
		Summarizes the distribution of customers' spending scores using quartiles. It represents the middle 50% of the data, while the line inside the box indicates the median spending score. The whiskers show the overall spread of the data, and any points beyond them represent potential outliers.

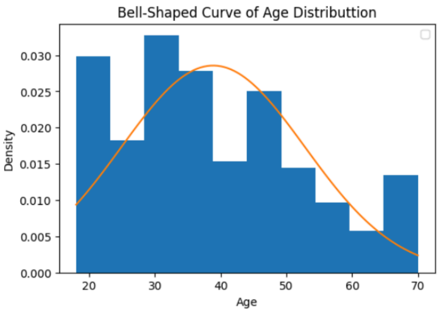
Python
`plt.figure(figsize=(6, 4)) # Creates a new graph window`
 	
`plt.scatter(`
`    df["Annual Income (k$)"],`
`    df["Spending Score (1-100)"]`
`)`
 	
`plt.xlabel("Annual Income (k$)") # Labels the ABSCISSA`
`plt.ylabel("Spending SCore (1-100)") # Labels the ORDINATE`
`plt.title("Annual Income vs Spending Score")`
`plt.show()`
		The plot illustrates the relationship between annual income and spending score. Each point represents an individual customer. The spread of points suggest that customers with similar income levels can have very different spending behaviours. In this case, there is no strong linear relationship between income and spending, indicating that high spending is not solely dependent on high income.

Python `plt.figure(figsize=(6, 4)) # Creates a new graph`		Visualizes the age distribution of customers using a normalized histogram, where the y-axis represents density instead of raw frequency. The dashed vertical line marks the mean age, making it easier to see how the average age compares to the overall distribution.

plt.hist(df["Age"], bins=10, density=True) # density=True scales the histogram so it matches a probability curve
 	
mean_age=df["Age"].mean()
 	
plt.axvline(mean_age, linestyle="--", label="Mean") # Vertical dashed line showing the mean
 	
plt.xlabel("Age")
plt.ylabel("Density")
plt.title("Distribution of Age with Mean")
plt.legend()
 	
plt.show		

Python plt.figure(figsize=(6, 4)) # Creates a new graph		This block displays a normalized histogram of customer ages with reference lines for the mean and one standard deviation above and below the mean. The region between the + standard deviation lines represent the range where most customer ages are concentrated.

plt.hist(df["Age"], bins=10, density=True)
 	
mean_age = df["Age"].mean()
std_age = df["Age"].std()
 	
plt.axvline(mean_age, linestyle="--", label="Mean")
plt.axvline(mean_age-std_age, linestyle=":", label="-1 Std Dev")
plt.axvline(mean_age+std_age, linestyle=":", label="+1 Std dev")
 	
plt.xlabel("Age")
plt.ylabel("Density")
plt.title("Distribution of Age with mean and standard deviation")
plt.legend()
plt.show		

Python
plt.figure(figsize=(6, 4))
 	
data = df["Age"]
mean = data.mean()
std = data.std()
 	
x = np.linspace(data.min(), data.max(), 100)
y = norm.pdf(x, mean, std)
 	
plt.hist(data, bins=10, density=True)
plt.plot(x,y)
 	
plt.xlabel("Age")
plt.ylabel("Density")
plt.title("Bell-Shaped Curve of Age Distributtion")
plt.legend()
plt.show
		Compares the actual distribution of customer ages with a theoretical normal distribution. The histogram shows that the observed age data has a representation of a smooth curve that is expected to have a distribution based on the calculated mean and standard deviation. The close alignment of the histogram and the bell curve suggest that customer age is approximately normally distributed, indicating that most customers' ages are around average with viewer individual at the extremes.

End of Descriptive Analysis Programming

Appendices

Github:

- https://github.com/Llena-Anthony/Data-Science.git

Google Colab Link:

- BERNABE LLENA.ipynb

Google Sheet Link:

- BERNABE LLENA