Assignment1_PavanKumar_GondabalRamakrishna_230049961

November 10, 2023

#Q1 A

[ ]:
```
from sklearn.datasets import load_wine
import pandas as pd
data = load_wine()
df = pd.DataFrame(data.data, columns=data.feature_names)
df['target'] = pd.Series(data.target)
```

[ ]:
```
df['target'].value_counts()

# 1 - 71
# 0 - 59
# 2- 48
```

[ ]:
```
1    71
0    59
2    48
Name: target, dtype: int64
```

[ ]:
```
#calculate the frequency for each catagories
print(59/len(df['target'])) #0.331
print(71/len(df['target'])) #0.398
print(48/len(df['target'])) #0.269
```

```
0.33146067415730335
0.398876404494382
0.2696629213483146
```

1 1a) df['target'] is the only catagorical column conisting of [0,1,2] as it has 3 catagories where the classification occurs based on different indipendent features

2 The frequency of 0 is 0.33, 1 is 0.398, 2 is 0.269 respectively

3 Q1 b)

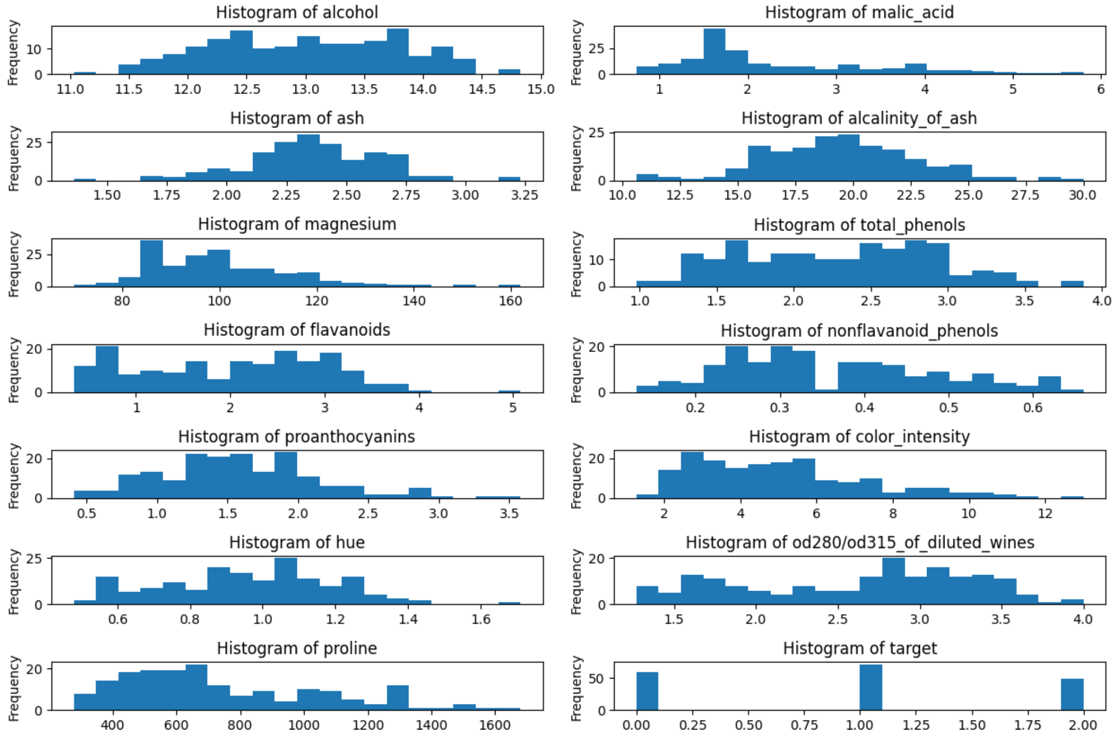
Univariative Summary

[ ]:
```python
univariate_summary = df.describe()
print(univariate_summary)
```

```
          alcohol  malic_acid         ash  alcalinity_of_ash    magnesium  \
count  178.000000  178.000000  178.000000         178.000000  178.000000
mean    13.000618    2.336348    2.366517          19.494944   99.741573
std      0.811827    1.117146    0.274344           3.339564   14.282484
min     11.030000    0.740000    1.360000          10.600000   70.000000
25%     12.362500    1.602500    2.210000          17.200000   88.000000
50%     13.050000    1.865000    2.360000          19.500000   98.000000
75%     13.677500    3.082500    2.557500          21.500000  107.000000
max     14.830000    5.800000    3.230000          30.000000  162.000000

       total_phenols  flavanoids  nonflavanoid_phenols  proanthocyanins  \
count     178.000000  178.000000            178.000000       178.000000
mean        2.295112    2.029270              0.361854         1.590899
std         0.625851    0.998859              0.124453         0.572359
min         0.980000    0.340000              0.130000         0.410000
25%         1.742500    1.205000              0.270000         1.250000
50%         2.355000    2.135000              0.340000         1.555000
75%         2.800000    2.875000              0.437500         1.950000
max         3.880000    5.080000              0.660000         3.580000

       color_intensity         hue  od280/od315_of_diluted_wines      proline  \
count       178.000000  178.000000                    178.000000   178.000000
mean          5.058090    0.957449                      2.611685   746.893258
std           2.318286    0.228572                      0.709990   314.907474
min           1.280000    0.480000                      1.270000   278.000000
25%           3.220000    0.782500                      1.937500   500.500000
50%           4.690000    0.965000                      2.780000   673.500000
75%           6.200000    1.120000                      3.170000   985.000000
max          13.000000    1.710000                      4.000000  1680.000000

           target
count  178.000000
mean     0.938202
std      0.775035
min      0.000000
25%      0.000000
50%      1.000000
75%      2.000000
max      2.000000
```

[ ]:
```python
import matplotlib.pyplot as plt

# Get numerical features
numerical_features = df.select_dtypes(include=['number']).columns
```

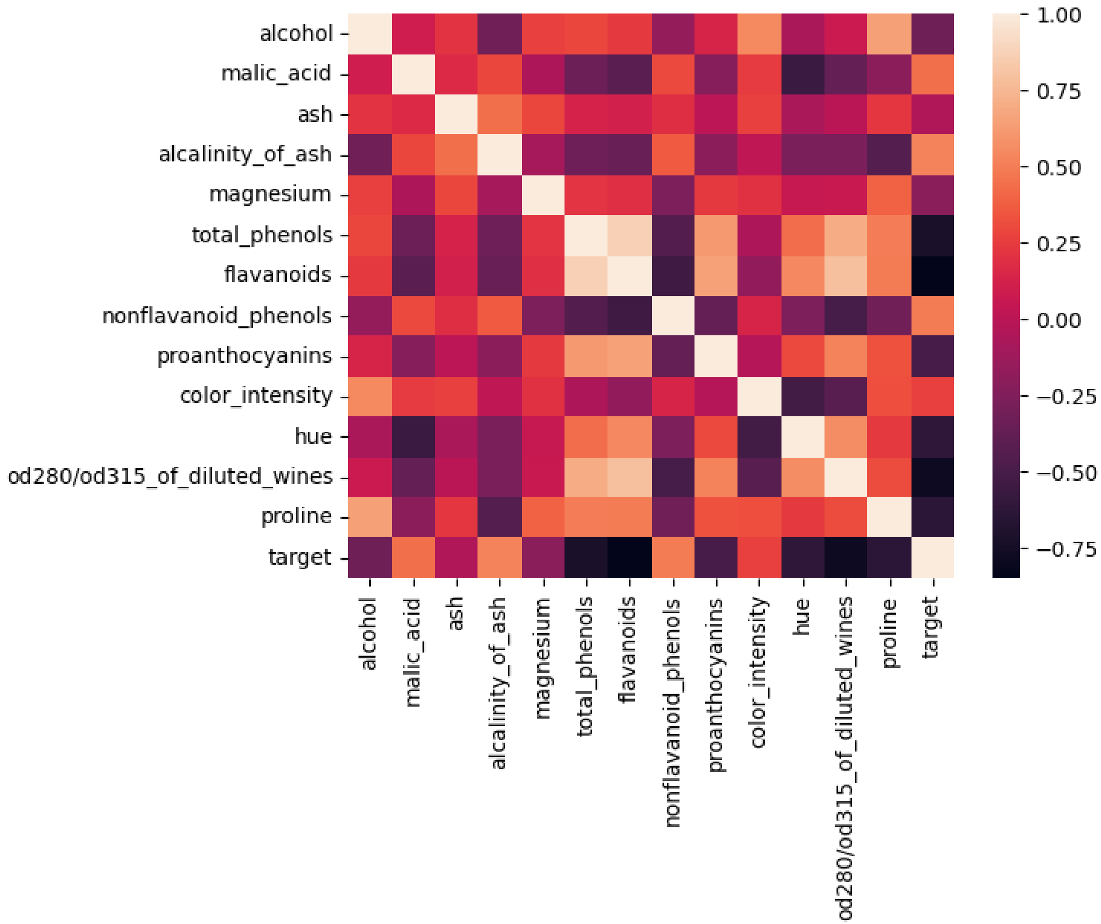
```python
# Determine the number of rows and columns in the grid
num_cols = 2  # Number of columns in the grid
num_rows = (len(numerical_features) + num_cols - 1) // num_cols

# Create a grid of subplots
fig, axes = plt.subplots(num_rows, num_cols, figsize=(12, 8))

# Plot histograms for each numerical feature
for i, feature in enumerate(numerical_features):
    row = i // num_cols
    col = i % num_cols
    ax = axes[row, col]
    df[feature].plot(kind='hist', bins=20, ax=ax)
    ax.set_title(f'Histogram of {feature}')
    ax.set_xlabel('')
    ax.set_ylabel('Frequency')

# Remove any empty subplots
for i in range(len(numerical_features), num_rows * num_cols):
    fig.delaxes(axes[i // num_cols, i % num_cols])

# Adjust layout and spacing
plt.tight_layout()

# Show the plot
plt.show()
```



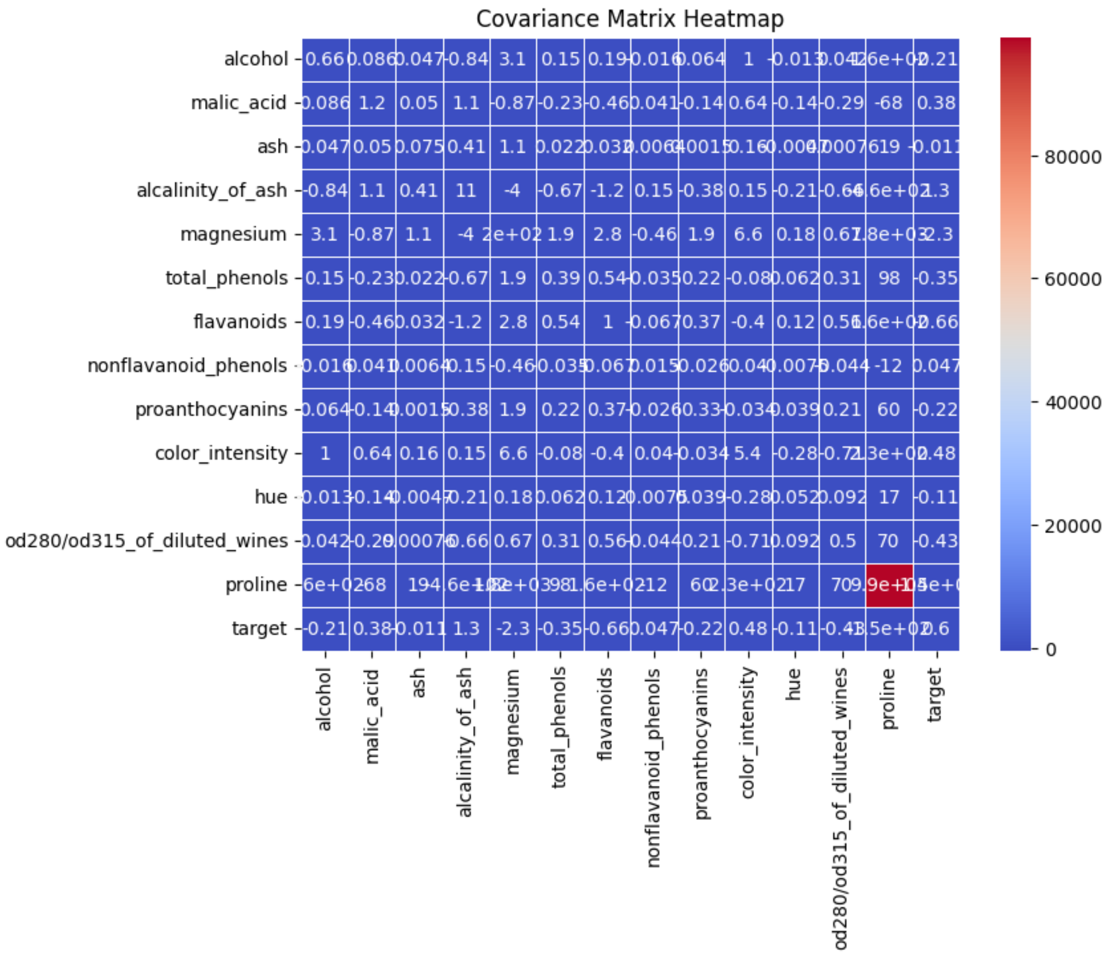
Multivariative Summary

[ ]:
```
import seaborn as sns
%matplotlib inline
correlation_matrix = df.corr()
sns.heatmap(correlation_matrix)
```

[ ]: <Axes: >




```
[ ]: covariance_matrix = df.cov()
# Set the size of the heatmap
plt.figure(figsize=(8, 6))

# Create the heatmap
sns.heatmap(covariance_matrix, annot=True, cmap='coolwarm', linewidths=0.5)

# Add a title
plt.title('Covariance Matrix Heatmap')

# Show the plot
plt.show()
```



4 Q1 c)

```
[ ]: median_by_category = df.groupby('target').median()

     # Display the median values
     print(median_by_category)
```

```
        alcohol  malic_acid   ash  alcalinity_of_ash  magnesium  \
target
0        13.750       1.770  2.44               16.8      104.0
1        12.290       1.610  2.24               20.0       88.0
2        13.165       3.265  2.38               21.0       97.0

        total_phenols  flavanoids  nonflavanoid_phenols  proanthocyanins  \
target
0               2.800       2.980                  0.29            1.870
```

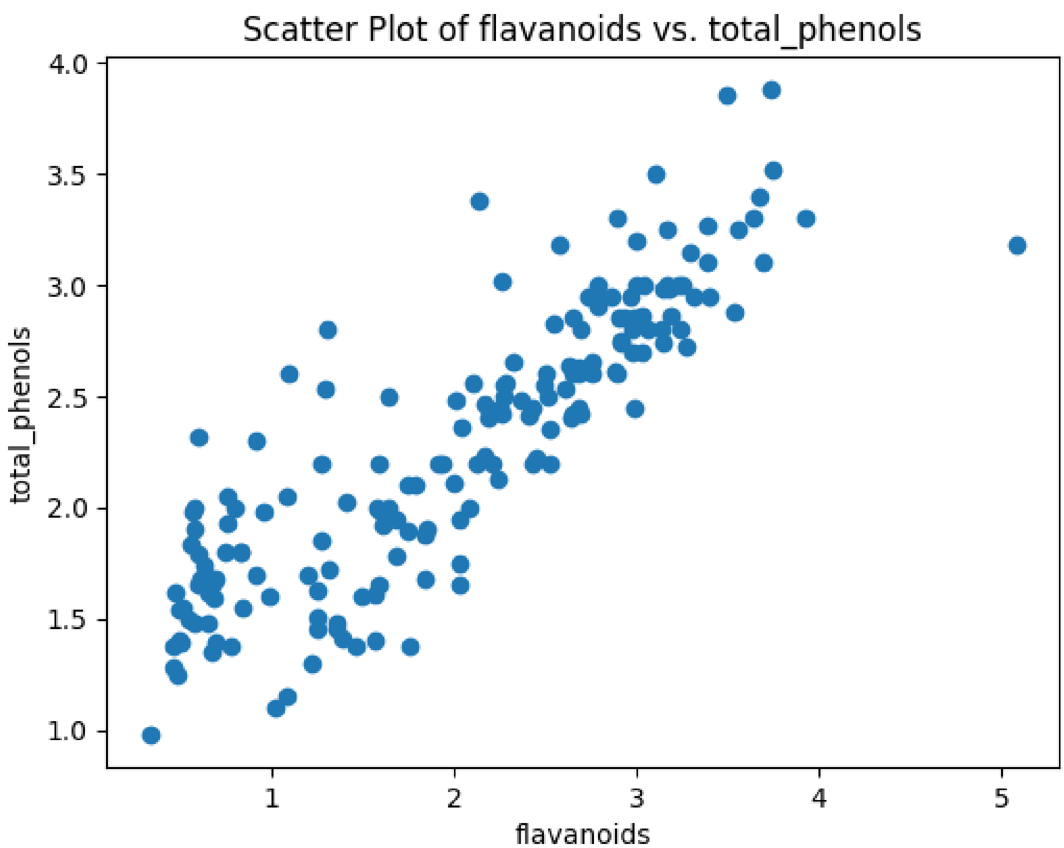
```
1                2.200       2.030                0.37           1.610
2                1.635       0.685                0.47           1.105

        color_intensity    hue  od280/od315_of_diluted_wines  proline
target
0                  5.40  1.070                          3.17   1095.0
1                  2.90  1.040                          2.83    495.0
2                  7.55  0.665                          1.66    627.5
```

5 Q1 d)

```
[ ]: highest_corr_columns = correlation_matrix.unstack().sort_values(ascending=False)
     highest_corr_columns = highest_corr_columns[highest_corr_columns < 1].idxmax()

     print("The two column names with the highest correlation are:",␣
      ↪highest_corr_columns)
```

```
The two column names with the highest correlation are: ('flavanoids',
'total_phenols')
```

```
[ ]: # Scatter plot
     plt.scatter(df['flavanoids'], df['total_phenols'])
     plt.xlabel('flavanoids')
     plt.ylabel('total_phenols')
     plt.title('Scatter Plot of flavanoids vs. total_phenols')
     plt.show()
```



#Q2

Equal-Frequency Binning (3 Bins with 4 Numbers in Each):

Bin 1: [1, 2, 5, 6]

Bin 2: [7, 8, 9, 11]

Bin 3: [12, 13, 17, 20]

Binning by Boundry:

Bin 1: [1, 1, 6, 6]

Bin 2: [7, 7, 11, 11]

Bin 3: [12, 12, 20, 20]

#Q3

[ ]:
```
import pandas as pd

income = pd.read_csv('/content/country-income.csv')
```

[ ]:
```
income['Income'].fillna(income['Income'].mean(),inplace=True)
```

[ ]:
```python
income['Online Shopper'] = income['Online Shopper'].map({'Yes': 1, 'No': 0})
```

[ ]:
```python
income
```

[ ]:
```
   Region   Age        Income  Online Shopper
0   India  49.0  86400.000000               0
1  Brazil  32.0  57600.000000               1
2     USA  35.0  64800.000000               0
3  Brazil  43.0  73200.000000               0
4     USA  45.0  76533.333333               1
5   India  40.0  69600.000000               1
6  Brazil   NaN  62400.000000               0
7   India  53.0  94800.000000               1
8     USA  55.0  99600.000000               0
9   India  42.0  80400.000000               1
```

#Q4

[ ]:
```python
df1 = pd.read_csv('/content/shoesize.csv')
```

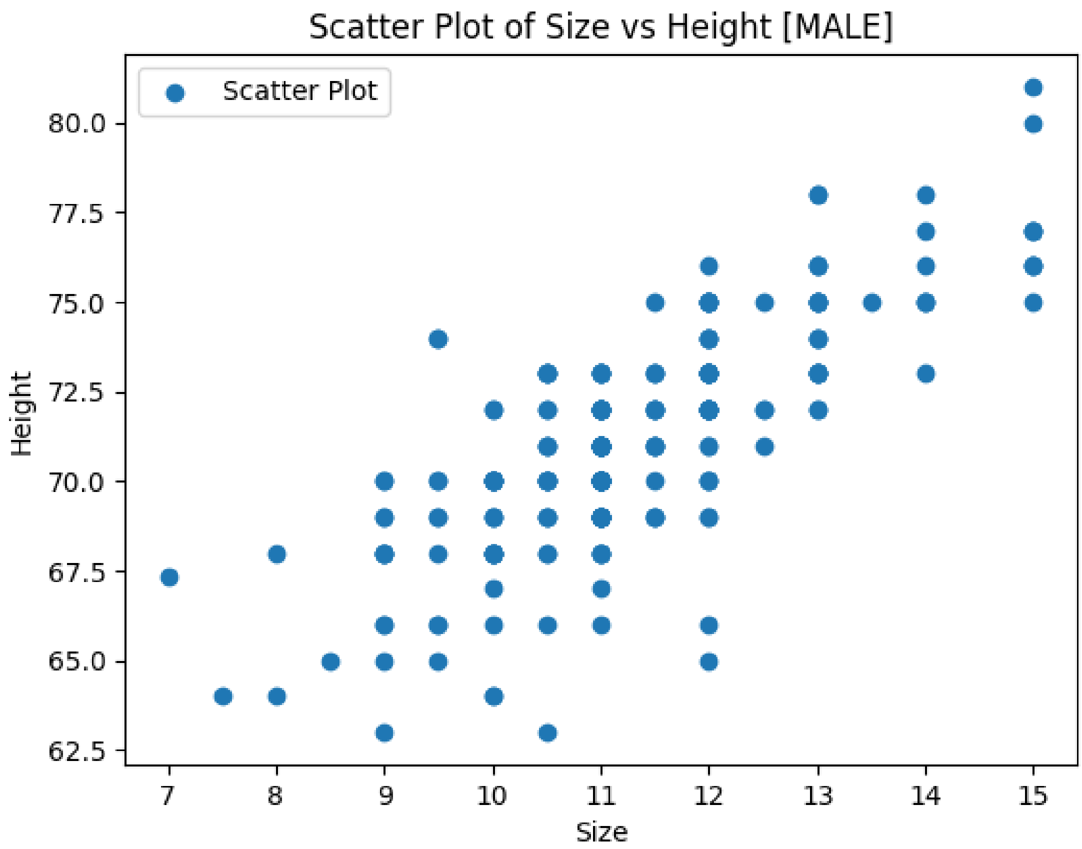
[ ]:
```python
df_male = df1[df1['Gender'] == 'M']
df_female = df1[df1['Gender'] == 'F']
```

[ ]:
```python
df_male
```

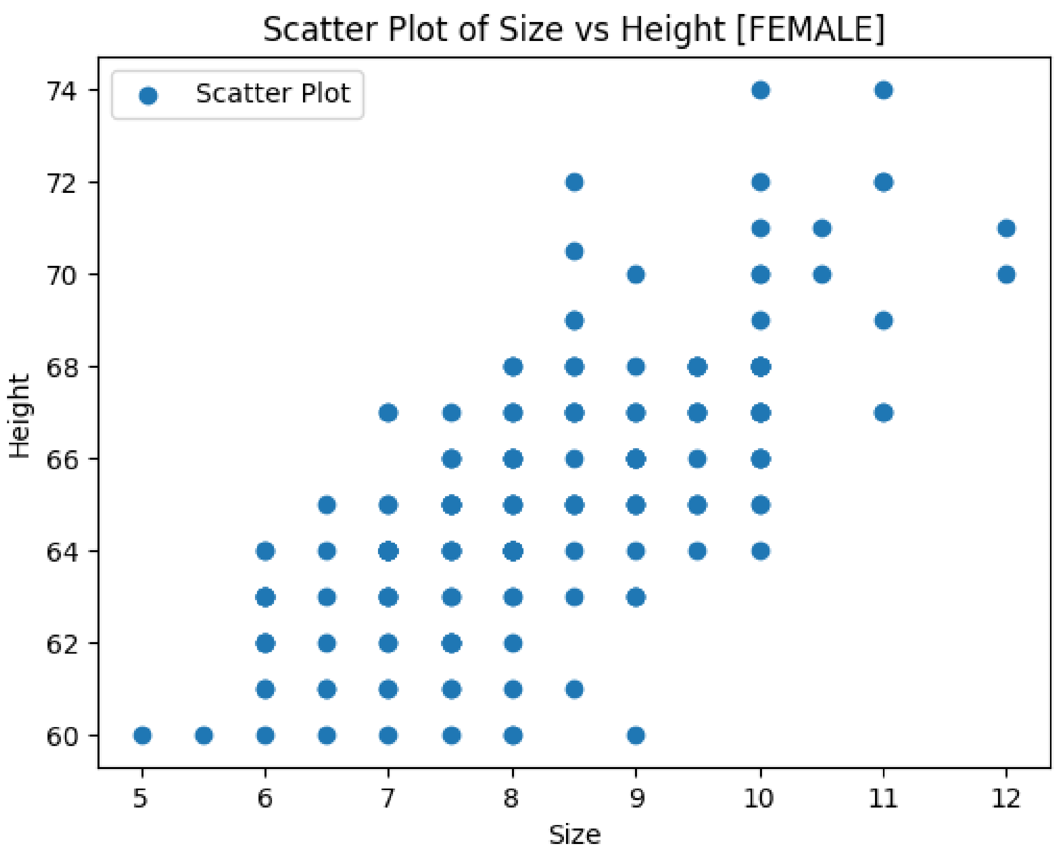
[ ]:
```
     Index Gender  Size  Height
187    188      M  10.5    63.0
188    189      M   9.0    63.0
189    190      M   7.5    64.0
190    191      M   8.0    64.0
191    192      M  10.0    64.0
..     …      …     …       …
403    404      M  13.0    78.0
404    405      M  13.0    78.0
405    406      M  14.0    78.0
406    407      M  15.0    80.0
407    408      M  15.0    81.0

[221 rows x 4 columns]
```

[ ]:
```python
import matplotlib.pyplot as plt
plt.scatter(df_male['Size'], df_male['Height'], label='Scatter Plot')

# Add labels and title
plt.xlabel('Size')
plt.ylabel('Height')
plt.title('Scatter Plot of Size vs Height [MALE]')
```

```
# Add a legend (if needed)
plt.legend()

# Show the plot
plt.show()
```



[ ]:
```
import matplotlib.pyplot as plt
plt.scatter(df_female['Size'], df_female['Height'], label='Scatter Plot')

# Add labels and title
plt.xlabel('Size')
plt.ylabel('Height')
plt.title('Scatter Plot of Size vs Height [FEMALE]')

# Add a legend (if needed)
plt.legend()

# Show the plot
plt.show()
```



[ ]:
```
correlation = df_male['Size'].corr(df_male['Height'])

# Print the Pearson correlation coefficient
print(f"Pearson Correlation between Size and Height for Male: {correlation}")
```

```
Pearson Correlation between Size and Height for Male: 0.7677093547300968
```

[ ]:
```
correlation = df_female['Size'].corr(df_female['Height'])

# Print the Pearson correlation coefficient
print(f"Pearson Correlation between Size and Height for Female: {correlation}")
```

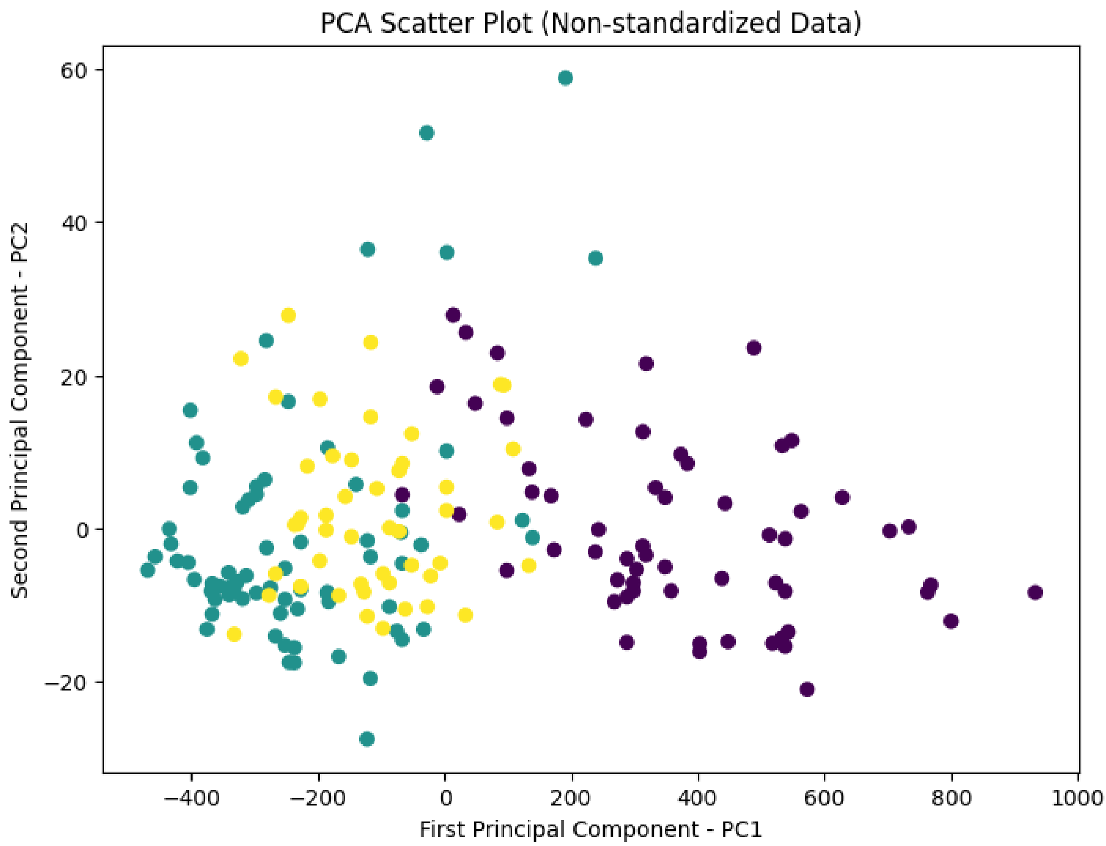
```
Pearson Correlation between Size and Height for Female: 0.7078119417143995
```

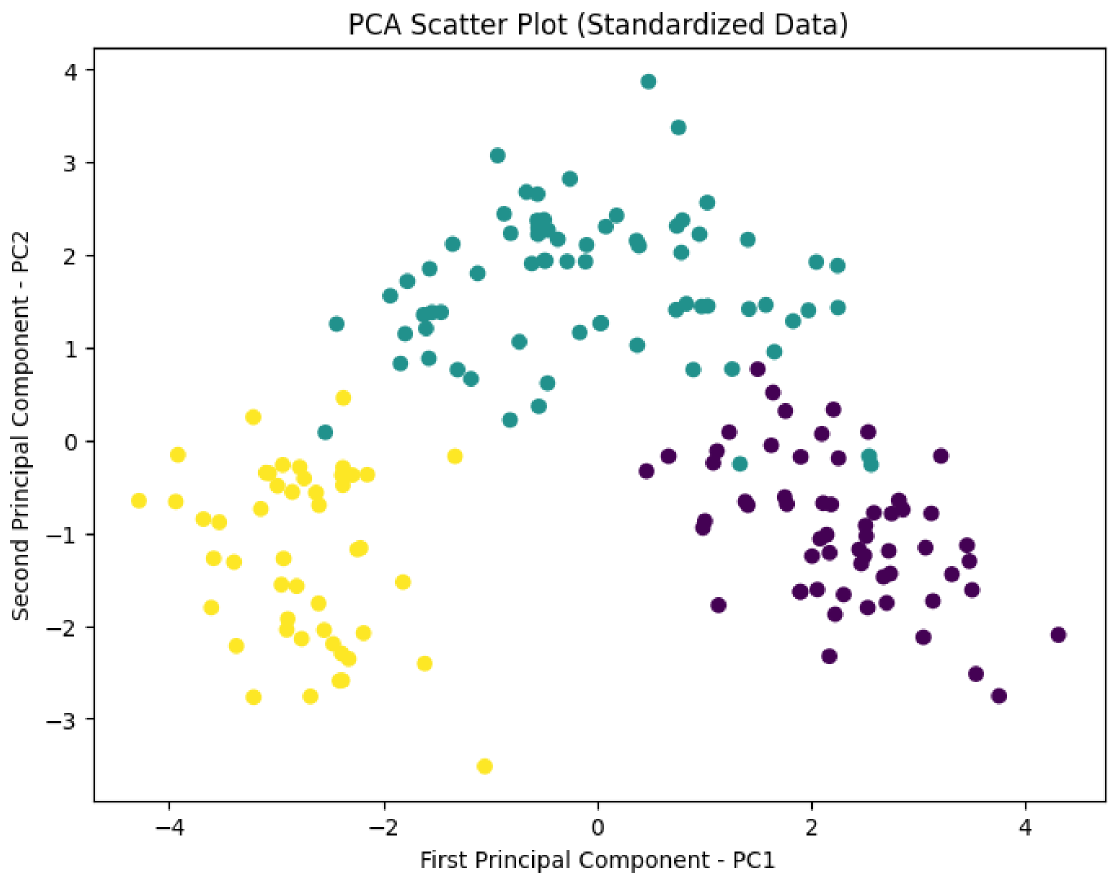
6 From the graphs and pearson corelation for Height VS Size for both Male and Female we can conclude that there exists a positive linear corelation and its 0.77 for Male and 0.70 for female.

#Q5

[ ]:

```python
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import load_wine

# Load the wine dataset
data = load_wine()
df = pd.DataFrame(data.data, columns=data.feature_names)
target = data.target

# Perform PCA with 2 components on non-standardized data
pca = PCA(n_components=2)
principalComponents = pca.fit_transform(df)

# Create a DataFrame with the principal components
df_pca = pd.DataFrame(data=principalComponents, columns=['PC1', 'PC2'])
df_pca['Target'] = target

# Plot the scatterplot of all samples along the two principal components
plt.figure(figsize=(8,6))
plt.scatter(df_pca['PC1'], df_pca['PC2'], c=df_pca['Target'])
plt.xlabel('First Principal Component - PC1')
plt.ylabel('Second Principal Component - PC2')
plt.title('PCA Scatter Plot (Non-standardized Data)')
plt.show()

# Standardize the features
scaler = StandardScaler()
df_scaled = scaler.fit_transform(df)

# Perform PCA with 2 components on standardized data
pca = PCA(n_components=2)
principalComponents = pca.fit_transform(df_scaled)

# Create a DataFrame with the principal components
df_pca = pd.DataFrame(data=principalComponents, columns=['PC1', 'PC2'])
df_pca['Target'] = target

# Plot the scatterplot of all samples along the two principal components
plt.figure(figsize=(8,6))
plt.scatter(df_pca['PC1'], df_pca['PC2'], c=df_pca['Target'])
plt.xlabel('First Principal Component - PC1')
plt.ylabel('Second Principal Component - PC2')
plt.title('PCA Scatter Plot (Standardized Data)')
plt.show()
```


PCA Scatter Plot (Non-standardized Data)
Second Principal Component - PC2
First Principal Component - PC1
60
40
20
0
−20
−400
−200
0
200
400
600
800
1000



7 As seen from the Wine data set we have 13 features and it is crucial to perform PCA which reduces the dimentinality of the features into 2 dimenions and 3 seperate classes. PCA will increase the varience between the classes and the data points does not have distinct boundires as they have been projected into different planes with PCA as we can see in the Graph 1 there is overlap between the classes, hence we standardize the dataset which reduces the mean of the data points which help us to separate the classes more distinctively hence we can see the clear boundries in the graph 2 after standardisation.

#Q6

Even while the distance matrix display in Lab 3, especially in subsection 1.9, seems to be limited in its informativeness, it provides insightful information. Interestingly, it shows that Students whose parents completed some high school have a higher average distance than students whose parents completed a master's degree. This observation mostly stems from the way the data is organized in the matrix.

The distance heatmap matrix presents a neatly organized student grouping according to parental educational attainment. This category is cleverly depicted by darker blue lines focused in the top-right and bottom-left corners. Students with parents who are less educated (bottom-left) and those who are more educated (top-right) are essentially divided by these lines, which signify higher parental education levels.

A distinct pattern can be seen when comparing the average distances between students in these two clusters: those with a master's degree and those who have only completed high school are significantly farther apart on average. This gap indicates notable differences in the characteristics of the students that are related to the educational attainment of their parents. It basically means that there is less of a similarity between pupils from these different parental education groups, which could be caused by variations in socioeconomic status, educational possibilities, or other relevant factors.

In conclusion, a careful analysis of the heatmap matrix and its data arrangement allows for the drawing of significant conclusions about the relationships between student clusters depending on the educational backgrounds of their parents, even when the distance matrix display may not provide exact numeric data.

#Q7

[15]:
```
!pip install cubes
!pip install sqlalchemy==1.3.20
from sqlalchemy import create_engine
```

```
Requirement already satisfied: cubes in /usr/local/lib/python3.10/dist-packages
(1.1)
Requirement already satisfied: python-dateutil in
/usr/local/lib/python3.10/dist-packages (from cubes) (2.8.2)
Requirement already satisfied: jsonschema in /usr/local/lib/python3.10/dist-
packages (from cubes) (4.19.2)
Requirement already satisfied: expressions>=0.2.3 in
/usr/local/lib/python3.10/dist-packages (from cubes) (0.2.3)
Requirement already satisfied: grako>=3.9.3 in /usr/local/lib/python3.10/dist-
packages (from expressions>=0.2.3->cubes) (3.99.9)
Requirement already satisfied: attrs>=22.2.0 in /usr/local/lib/python3.10/dist-
packages (from jsonschema->cubes) (23.1.0)
Requirement already satisfied: jsonschema-specifications>=2023.03.6 in
/usr/local/lib/python3.10/dist-packages (from jsonschema->cubes) (2023.7.1)
Requirement already satisfied: referencing>=0.28.4 in
/usr/local/lib/python3.10/dist-packages (from jsonschema->cubes) (0.30.2)
Requirement already satisfied: rpds-py>=0.7.1 in /usr/local/lib/python3.10/dist-
packages (from jsonschema->cubes) (0.12.0)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-
packages (from python-dateutil->cubes) (1.16.0)
Requirement already satisfied: sqlalchemy==1.3.20 in
/usr/local/lib/python3.10/dist-packages (1.3.20)
```

#7a

[21]:
```
from cubes.tutorial.sql import create_table_from_csv


# Create SQLite database engine
engine = create_engine("sqlite:///mydatabase.db")

create_table_from_csv(engine,
                      "/content/country-income.csv",
                      table_name="c_income",
                      fields=[
                          ("Region", "string"),
                          ("Age", "string"),
                          ("Online_Shopper", "string"),
                          ("Income", "string")],
                      create_id=True
                     )
```

[27]:
```
#json file

{
    "dimensions": [

          {
            "name":"Region",
            "attributes": ["Region"]

          },
          {
            "name":"Age",
            "attributes": ["Age"]

          },
          {
            "name":"Online_Shopper",
            "attributes": ["Online_Shopper"]
          },
          {
            "name" : "item",
            "levels" : [
                        {
                          "name":"Region",
                          "attributes": ["Region"]

                        },
                        {
```

```
                    "name":"Age",
                    "attributes": ["Age"]

                  },
                  {
                    "name":"Online_Shopper",
                    "attributes": ["Online_Shopper"]
                  }
          ]
        }
    ],
    "cubes": [
        {
            "name": "c_income",
            "dimensions": ["Region", "Age", "Online_Shopper", "item"],
            "measures": [{"name":"Income", "label":"Income"}],
            "aggregates": [
                    {
                        "name": "income_sum",
                        "function": "sum",
                        "measure": "Income"
                    },
                    {
                        "name": "record_count",
                        "function": "count"
                    },
                    {
                        "name": "average",
                        "function": "avg",
                        "measure": "Income"
                    },
                    {
                        "name": "min",
                        "function": "min",
                        "measure": "Income"
                    },
                    {
                        "name": "max",
                        "function": "max",
                        "measure": "Income"
                    }
                ],

            "mappings": {
                    "item.Region": "Region",
                    "item.Age": "Age",
                    "item.Online_Shopper": "Online_Shopper"
```

```
                         }


        }
    ]
}
```

[26]:
```
from cubes import Workspace
import collections
from collections import abc
collections.MutableMapping = abc.MutableMapping

workspace = Workspace()
workspace.register_default_store("sql", url="sqlite:///data.sqlite")



workspace.import_model("tutorial_model.json")
```

[23]:
```
cube = workspace.cube("c_income")
browser = workspace.browser(cube)
result = browser.aggregate()
```

#7B

[ ]:
```
#Total aggregates
result.summary["record_count"]
#output = 10
result.summary["income_sum"]
#output = 688800.0

result.summary["average"]
#76533.33
result.summary["min"]

#57600
result.summary["max"]
#99600
```

[ ]:
```
#results per region;

result = browser.aggregate(drilldown=["Region"])
for record in result:
    print(record)
```

[ ]:
```
{'Region': 'Brazil', 'income_sum': 193200, 'record_count': 3, 'average': 64400.
↪0, 'min': '57600', 'max': '73200'}
```

```
{'Region': 'India', 'income_sum': 331200, 'record_count': 4, 'average': 82800.
↪0, 'min': '69600', 'max': '94800'}
{'Region': 'USA', 'income_sum': 164400.0, 'record_count': 3, 'average': 54800.
↪0, 'min': '', 'max': '99600'}
```

[ ]:
```
# results per online shopping activity

result = browser.aggregate(drilldown=["Online_Shopper"])
for record in result:
    print(record)
```

[ ]:
```
{'Online_Shopper': 'No', 'income_sum': 386400, 'record_count': 5, 'average':␣
↪77280.0, 'min': '62400', 'max': '99600'}
{'Online_Shopper': 'Yes', 'income_sum': 302400.0, 'record_count': 5, 'average':␣
↪60480.0, 'min': '', 'max': '94800'}
```

[ ]:
```
# results for all people aged between 40 and 50

from sqlalchemy import create_engine, MetaData, Table
from cubes import Workspace, PointCut

# Create SQLAlchemy engine to connect to the database

# Define the lower and upper bounds of the age range
lower_bound = "40"
upper_bound = "50"

# Use PointCut to filter "Age" between the specified range
cut = PointCut("Age", [str(age) for age in range(int(lower_bound),␣
↪int(upper_bound) + 1)])



# Apply the cut to filter the data
result = browser.aggregate(drilldown=["Age"], cut=cut)

# Print aggregated data
for record in result:
    print(record)
```

[ ]:
```
#results for all people aged between 40 and 50


{'Age': '40', 'income_sum': 69600, 'record_count': 1, 'average': 69600.0, 'min':
↪ '69600', 'max': '69600'}
{'Age': '42', 'income_sum': 80400, 'record_count': 1, 'average': 80400.0, 'min':
↪ '80400', 'max': '80400'}
```

```
{'Age': '43', 'income_sum': 73200, 'record_count': 1, 'average': 73200.0, 'min':
 ↪ '73200', 'max': '73200'}
{'Age': '45', 'income_sum': 0.0, 'record_count': 1, 'average': 0.0, 'min': '',␣
 ↪'max': ''}
{'Age': '49', 'income_sum': 86400, 'record_count': 1, 'average': 86400.0, 'min':
 ↪ '86400', 'max': '86400'}
```

Q8>

In 1-nearest neighbor (1-NN) classifier, a new observation is classified based on the class of it's nearest neighbor in the training dataset. The distance metric used for finding the nearest neighbor is the Euclidean distance.

-> Given dataset: $x_1 = (1, 2), \; y_1 = 1$
$x_2 = (-1, 0), \; y_2 = 0$

new observations: $x_3 = (3, 2)$
$x_4 = (0, 1)$

1. Distance to x_3:

$$d(x_1, x_3) = \sqrt{(1-3)^2 + (2-2)^2} = 2$$

$$d(x_2, x_3) = \sqrt{(-1-3)^2 + (0-2)^2} = 4.47$$

-> The euclidean distance of $d(x_1, x_3) = 2$ which is less than $d(x_2, x_3) = 4.47$. hence it can be classified as $(y_1 = 1)$. Since it's nearest neighbour is x_1.

2. Distance to x_4:

$$d(x_1, x_4) = \sqrt{(1-0)^2 + (2-1)^2} = 1.41$$

$$d(x_2, x_4) = \sqrt{(-1-0)^2 + (0-1)^2} = 1.41$$

-> Since x_4 has same euclidean distance from both x_1 and x_2 i.e, 1.41. It can be classified into either of classes $y_1 = 1$ or $y_2 = 0$,
But the classifier would classify the x_4 as x_1 (1, 2), with class $y_1 = 1$ based on tie-breaking rule as x_1 is the first encounter