Assignment1-Pranay_Dhareshwar -230435164

November 10, 2023

1 Data Mnining Assignment 1 - Pranay Dhareshwar 230435164

1.1 Q1) The function "load_wine" from "sklearn.datasets" can be used to load the wine dataset into a "DataFrame" by using the below commands:

```
data = load_wine()
df = pd.DataFrame(data.data, columns=data.feature_names)
df['target'] = pd.Series(data.target)
```

1.1.1 (a) Load the wine dataset. Which feature is categorical and why? Compute the frequency (not the occurrence) of each value of the categorical feature. Include the code in your report.

[1]:
```
# Import necessary libraries
import pandas as pd
from sklearn.datasets import load_wine  # Import function to load the wine␣
↪dataset

# Load the wine dataset
data = load_wine()

# Create a pandas data frame using the dataset's data and feature names
wine = pd.DataFrame(data.data, columns=data.feature_names)

# Add the target column to the data frame to represent the class labels
wine['target'] = pd.Series(data.target)
```

[2]:
```
# Print the data types of the data frame
print(wine.dtypes)
```

```
alcohol                         float64
malic_acid                      float64
ash                             float64
alcalinity_of_ash               float64
magnesium                       float64
total_phenols                   float64
flavanoids                      float64
```

```
nonflavanoid_phenols            float64
proanthocyanins                 float64
color_intensity                 float64
hue                             float64
od280/od315_of_diluted_wines    float64
proline                         float64
target                            int64
dtype: object
```

[3]:
```
# Compute the frequency (not the occurrence) of each value of the categorical␣
↪feature
frequency = wine['target'].value_counts()

# Print out the frequency
print(f'The frequency of each value of the categorical feature:\n{frequency}')
```

```
The frequency of each value of the categorical feature:
target
1    71
0    59
2    48
Name: count, dtype: int64
```

To determine which feature is categorical in the dataset, we utilise the `.dtypes` function provided by the pandas library. This function lets us display the data types associated with each feature or column in the data frame. By examining these data types, we can identify the nature of each feature, assisting in the recognition of categorical features within the dataset.

From the output, it is evident that each feature or column in the data frame holds the `float64` data type, except for the `target` column. The 'target' column is an `int64` data type. We need to carry out more investigation to determine whether the odd one out is a categorical feature or not.

We utilise the `.value_counts()` function from the pandas library. This function allows us to showcase the frequency, not just the occurrence, of each value present within the 'target' feature.

There are three different labels in this target column: labelled 0, 1, and 2.

- Class 0 has 59 occurrences.
- Class 1 has 71 occurrences.
- Class 2 has 48 occurrences.

This information supports the understanding that the 'target' column represents categorical classes or labels since it contains discrete categories (0, 1, and 2). Therefore, based on this analysis, the 'target' column is the categorical feature in our dataset, containing the classes or labels used for classification in this dataset.

1.1.2 (b) Compute two different univariate and two different multivariate summaries for all numerical features. Include the code in your report.

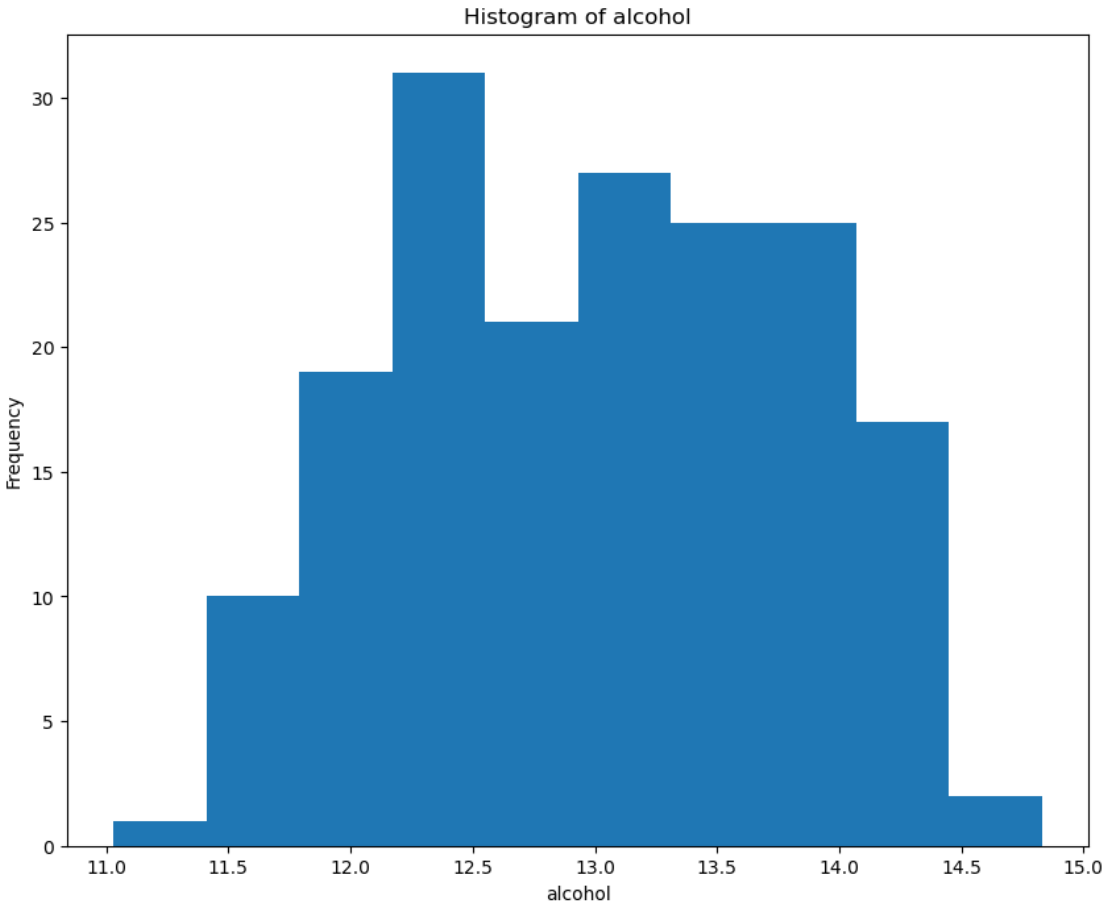
[4]:
```
# Creating a copy of the original data frame without the target column
wine_copy = wine.drop(columns=['target']).copy()
```

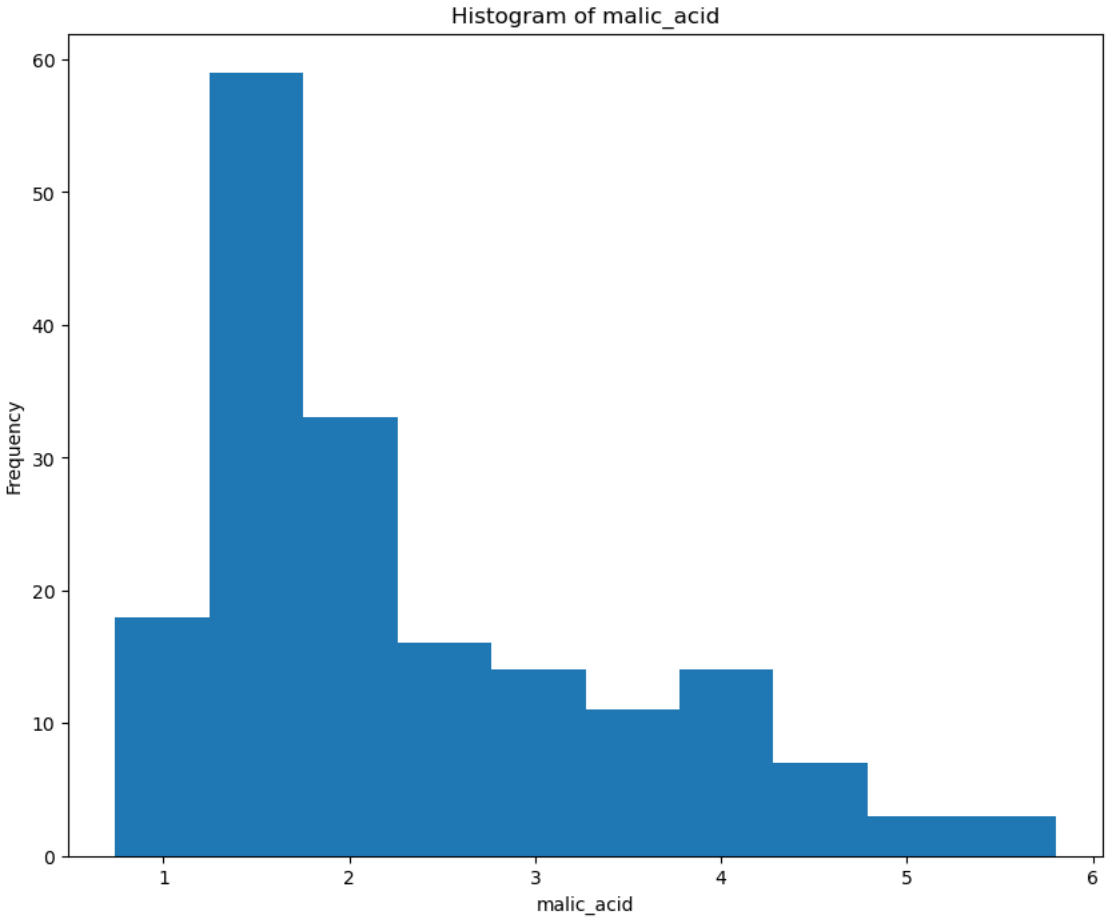
[5]:
```
# First univariate summary for the numerical features
print(wine_copy.describe())
```

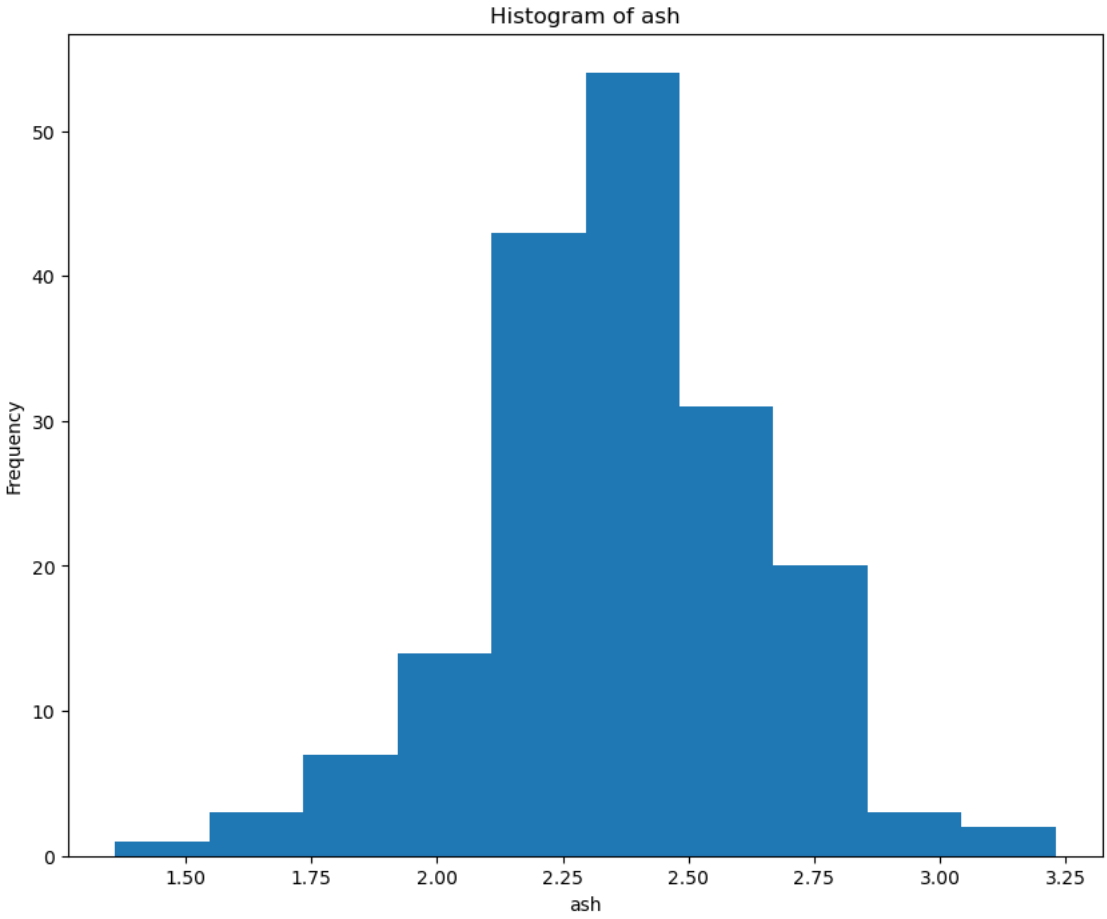
```
          alcohol  malic_acid         ash  alcalinity_of_ash   magnesium  \
count  178.000000  178.000000  178.000000         178.000000  178.000000
mean    13.000618    2.336348    2.366517          19.494944   99.741573
std      0.811827    1.117146    0.274344           3.339564   14.282484
min     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
```

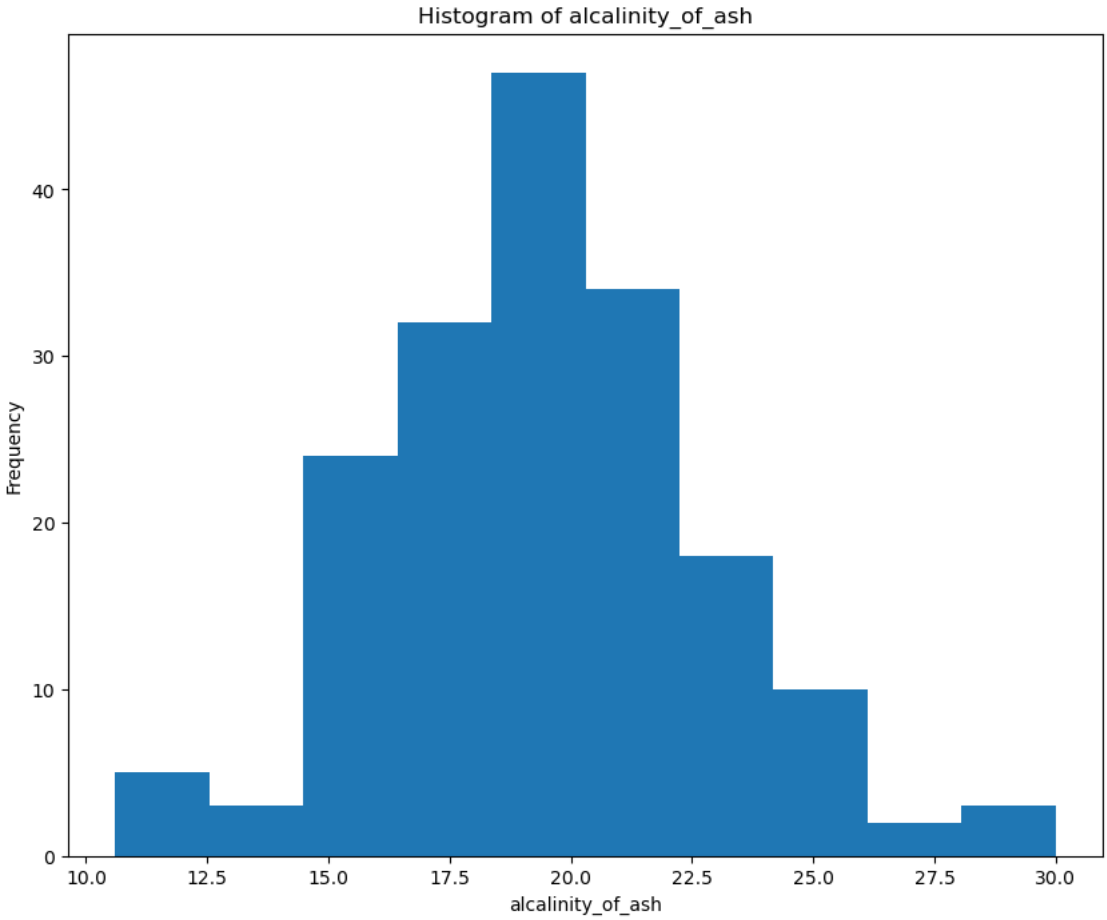
[37]:
```
# Second univariate analysis of the numerical features
import matplotlib.pyplot as plt  # Importing the correct package and module

# Plotting histograms for each column in wine_copy
for column in wine_copy:
    wine_copy[column].plot.hist(figsize=(10, 8))
    plt.title(f'Histogram of {column}')
    plt.xlabel(column)
```

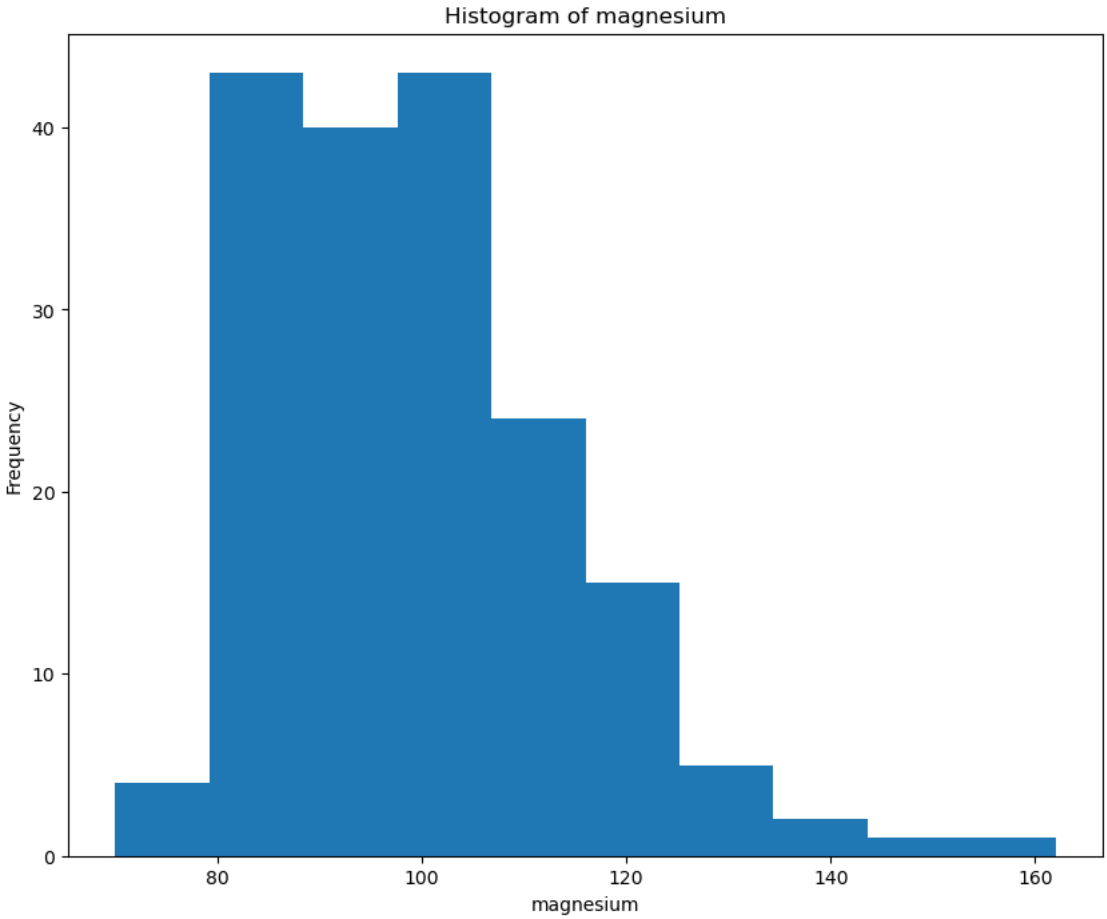
```
    plt.ylabel('Frequency')
    plt.show()  # Display each histogram individually
```

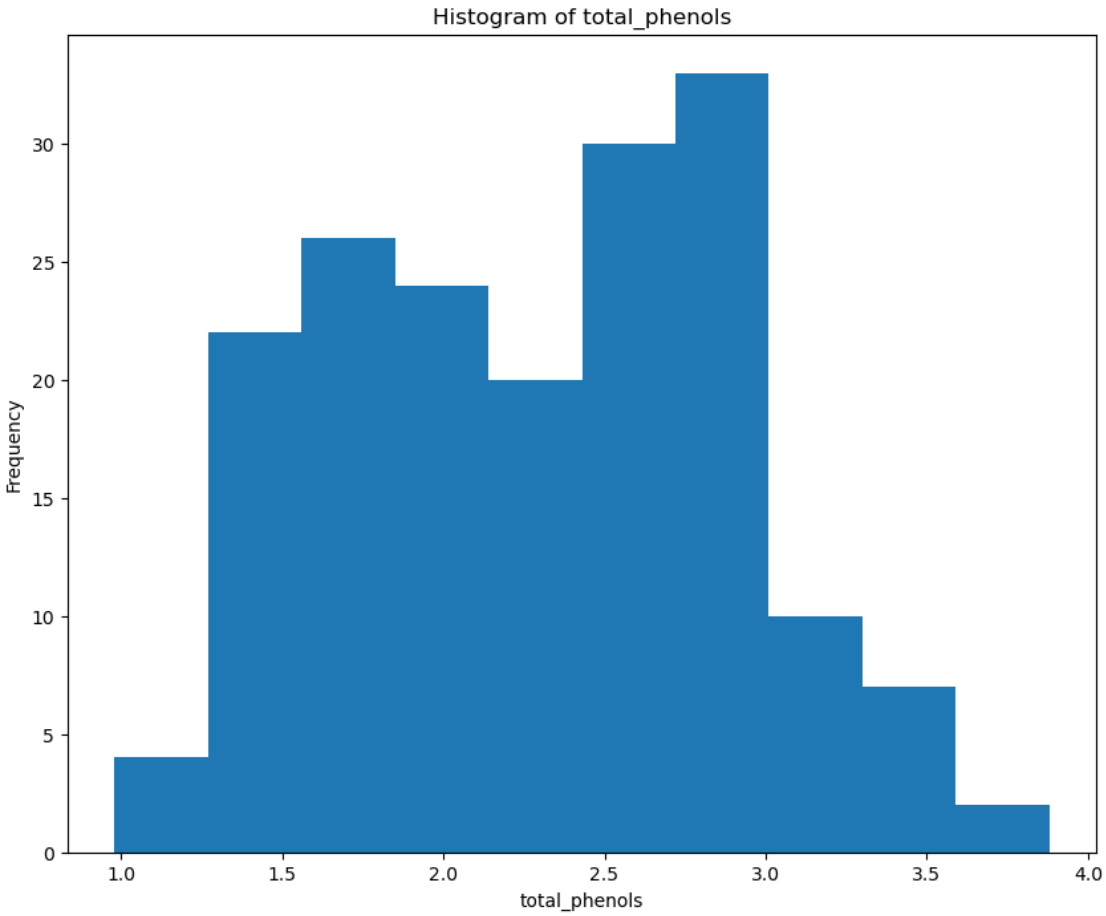


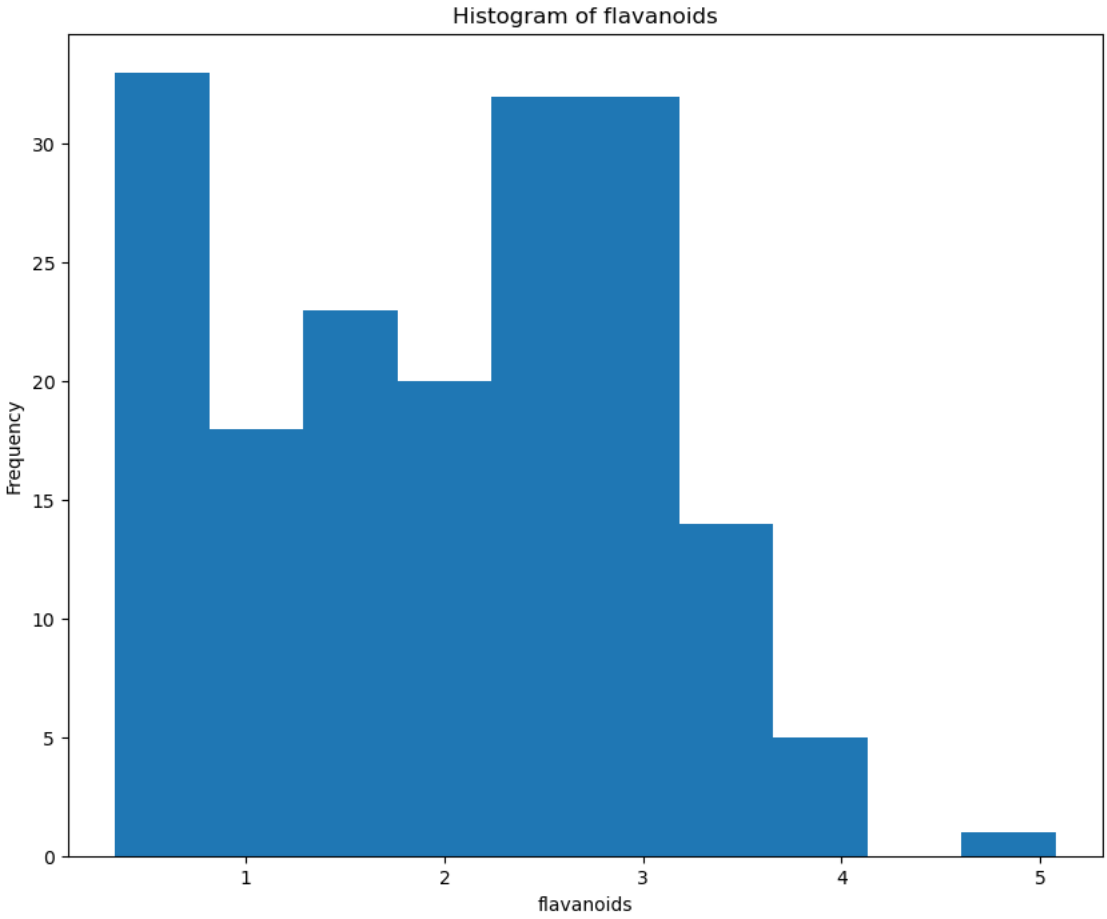

Histogram of malic_acid
Frequency
malic_acid
0
10
20
30
40
50
60
1
2
3
4
5
6

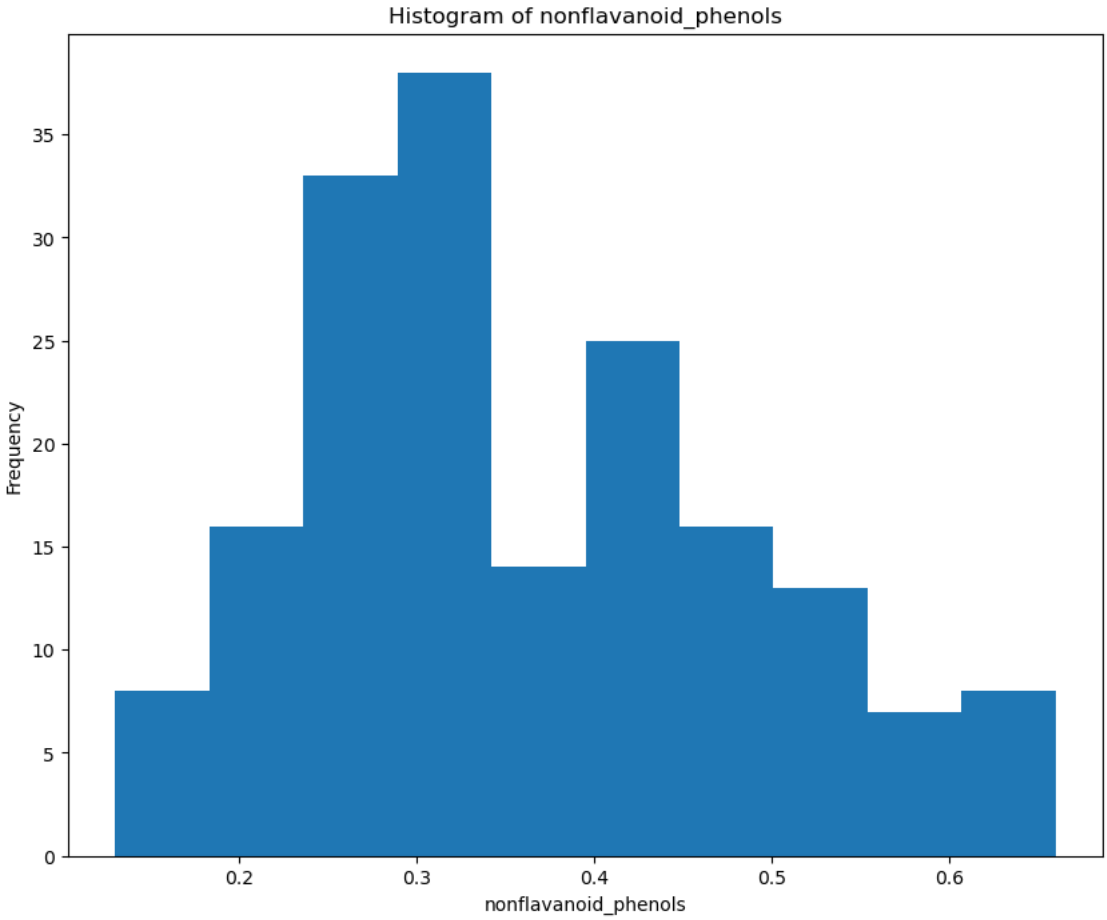

Histogram of ash
Frequency
0
10
20
30
40
50
1.50
1.75
2.00
2.25
2.50
2.75
3.00
3.25
ash

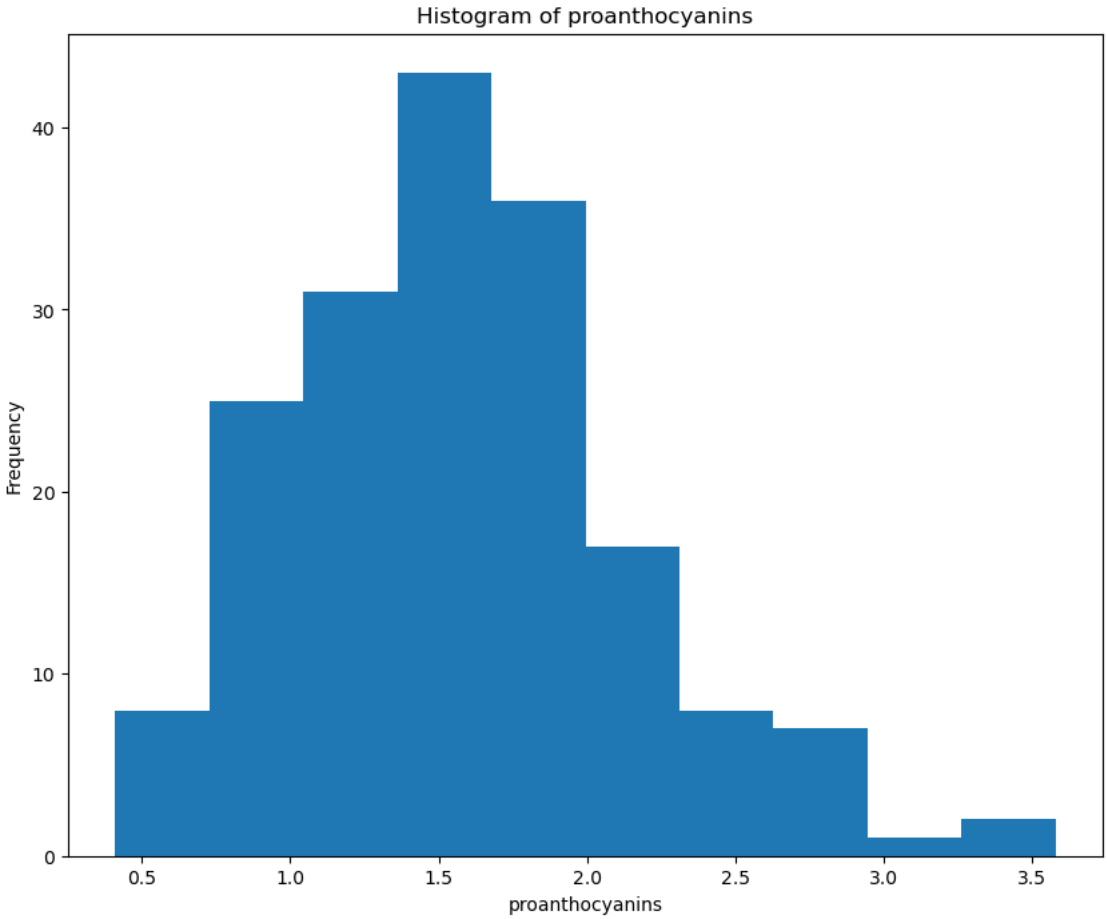

Histogram of alcalinity_of_ash
Frequency
0
10
20
30
40
10.0
12.5
15.0
17.5
20.0
22.5
25.0
27.5
30.0
alcalinity_of_ash

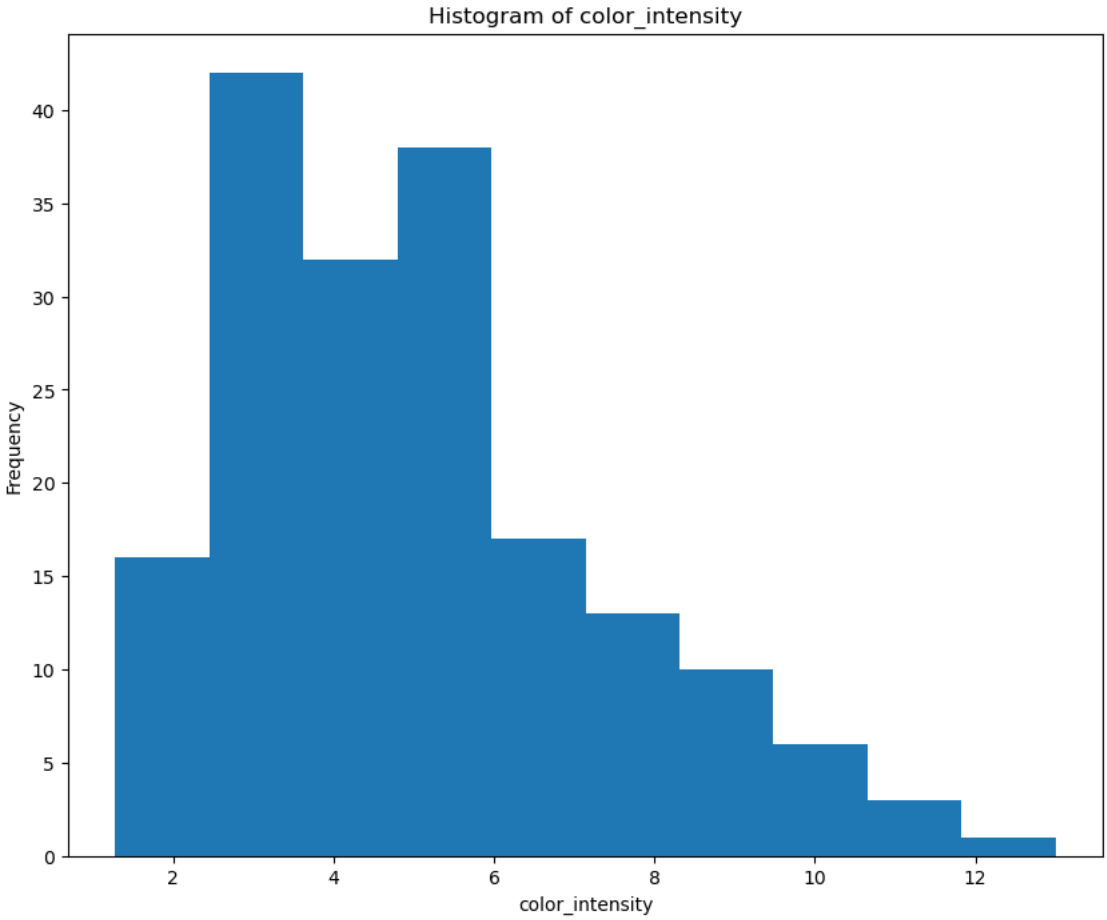

Histogram of magnesium
Frequency
0
10
20
30
40
80
100
120
140
160
magnesium

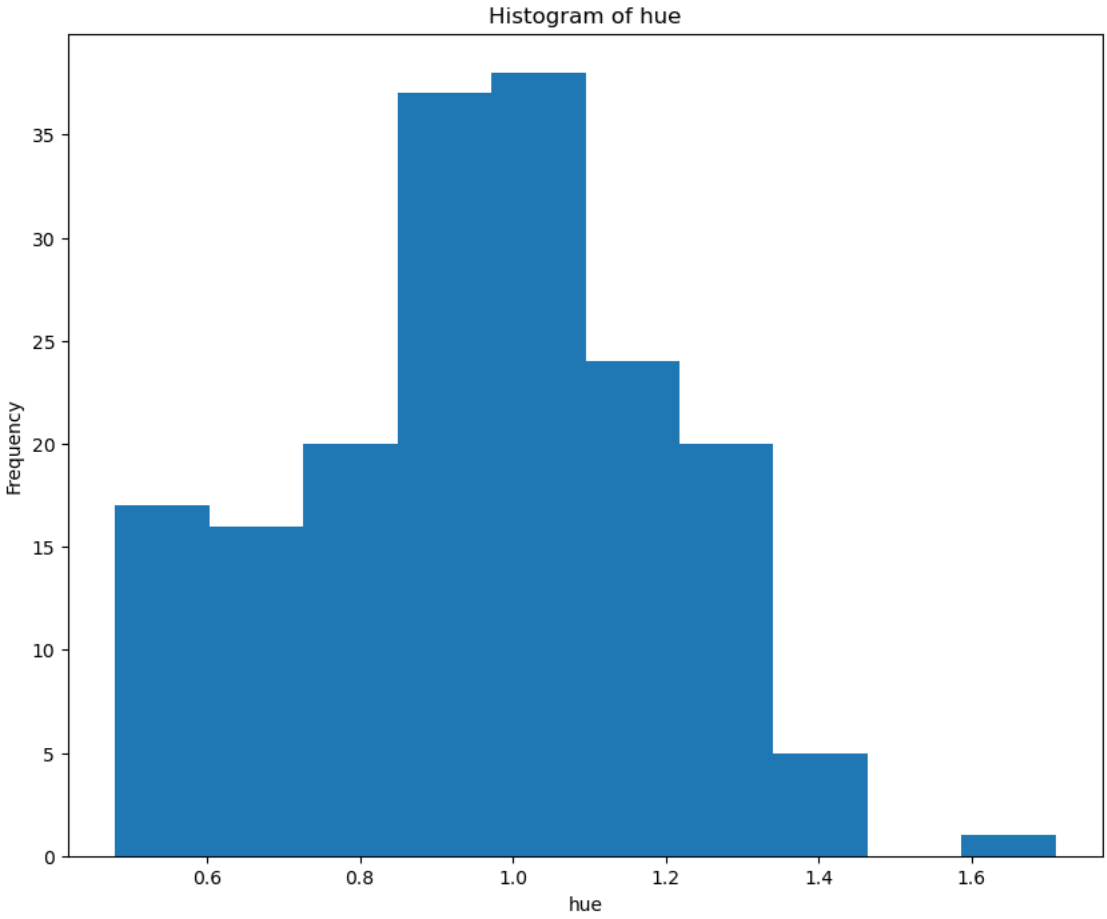

Histogram of total_phenols
Frequency
0
5
10
15
20
25
30
1.0
1.5
2.0
2.5
3.0
3.5
4.0
total_phenols

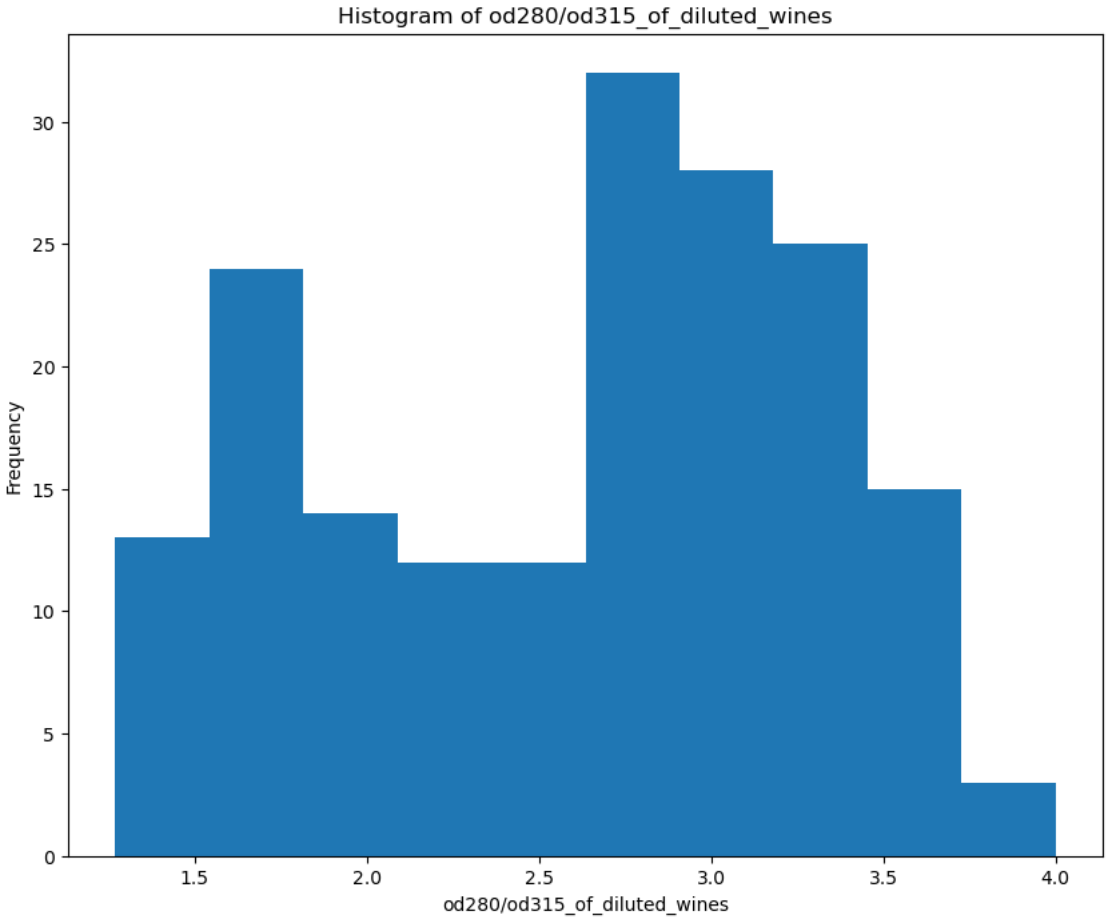

Histogram of flavanoids
Frequency
flavanoids
0
5
10
15
20
25
30
1
2
3
4
5

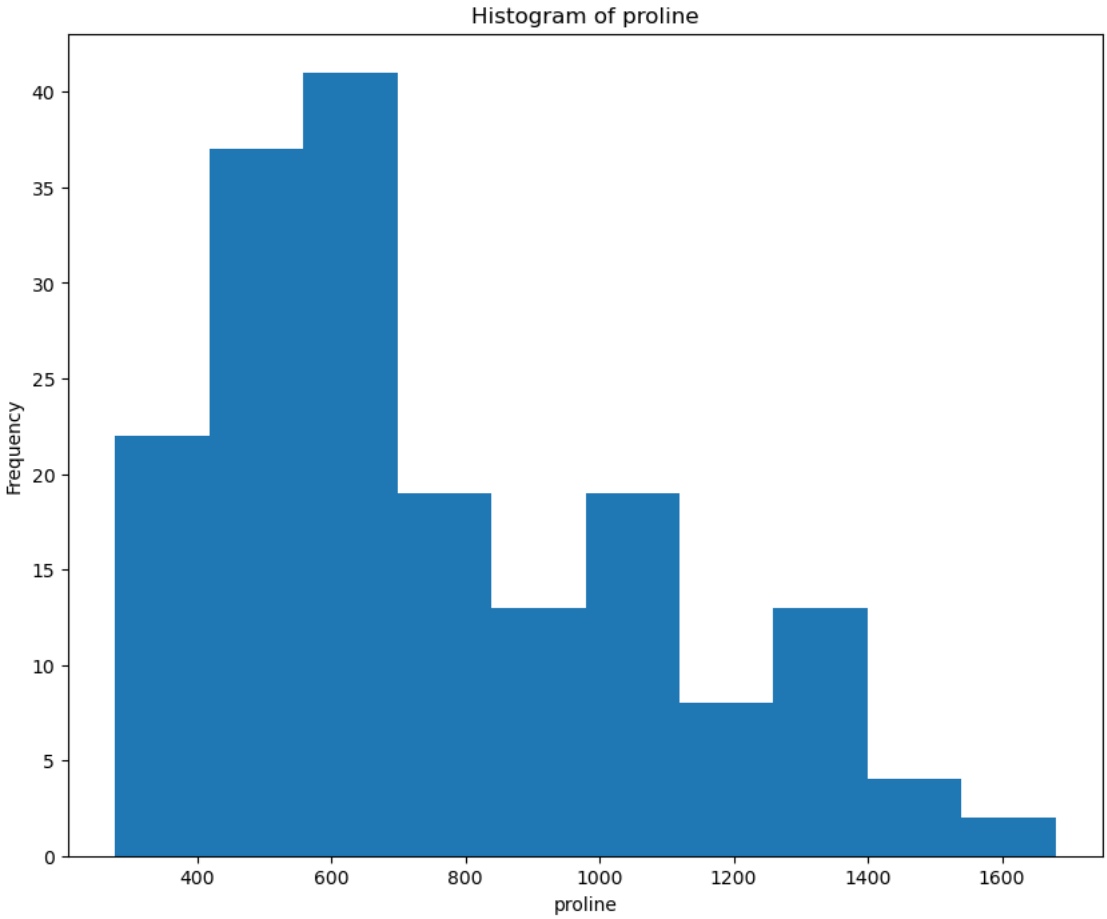

Histogram of nonflavanoid_phenols
Frequency
nonflavanoid_phenols
0
5
10
15
20
25
30
35
0.2
0.3
0.4
0.5
0.6


Histogram of proanthocyanins
Frequency
0
10
20
30
40
0.5
1.0
1.5
2.0
2.5
3.0
3.5
proanthocyanins


Histogram of color_intensity
Frequency
color_intensity
0
5
10
15
20
25
30
35
40
2
4
6
8
10
12


Histogram of hue
Frequency
hue
0
5
10
15
20
25
30
35
0.6
0.8
1.0
1.2
1.4
1.6


Histogram of od280/od315_of_diluted_wines
Frequency
0
5
10
15
20
25
30
1.5
2.0
2.5
3.0
3.5
4.0
od280/od315_of_diluted_wines



[7]:
```
# First multivariate analysis for numerical features
correlation_matrix = wine_copy.corr()

# print out the correlation matrix
print(correlation_matrix)
```

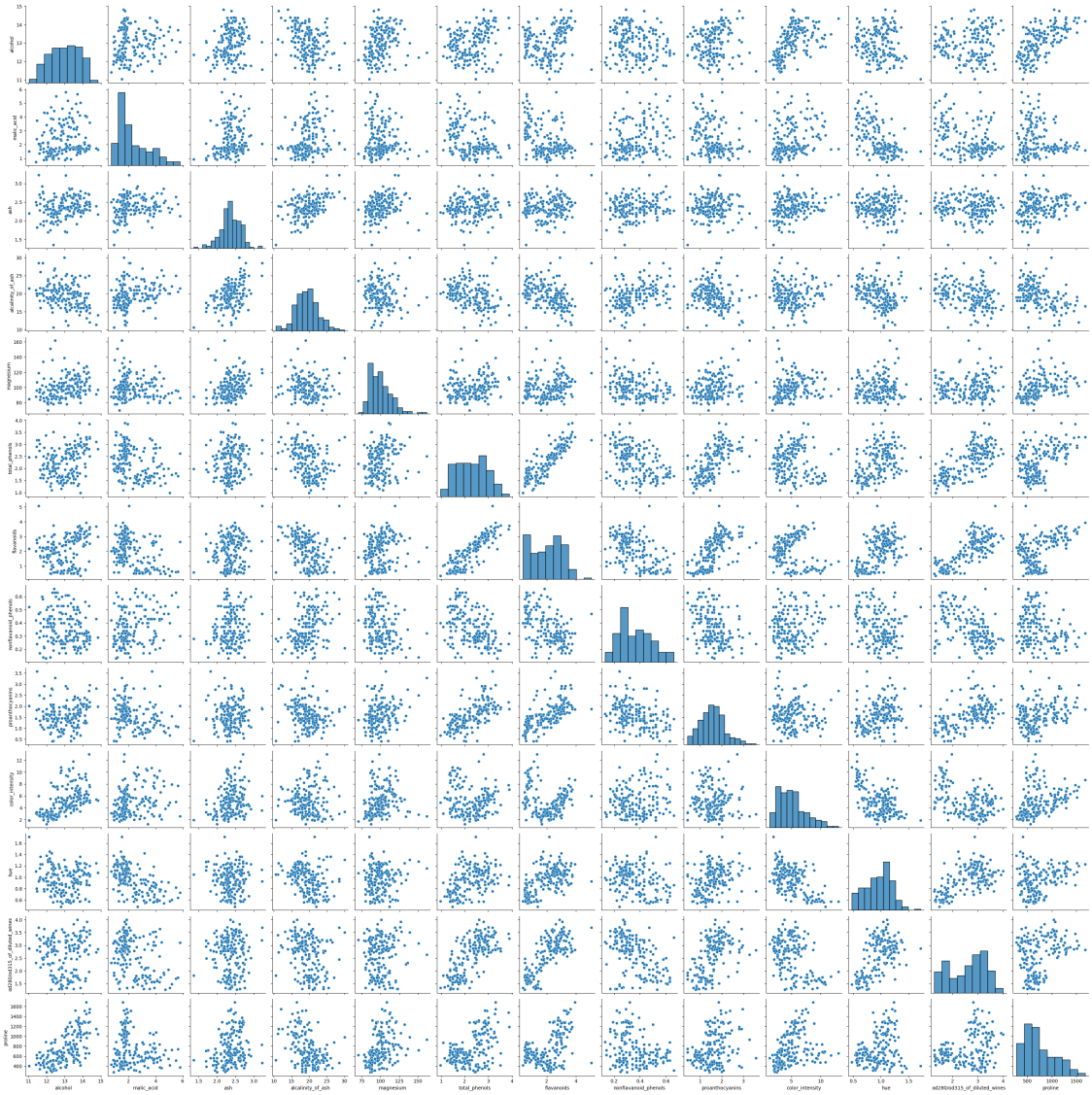
```
                              alcohol  malic_acid       ash  \
alcohol                      1.000000    0.094397  0.211545
malic_acid                   0.094397    1.000000  0.164045
ash                          0.211545    0.164045  1.000000
alcalinity_of_ash           -0.310235    0.288500  0.443367
magnesium                    0.270798   -0.054575  0.286587
total_phenols                0.289101   -0.335167  0.128980
flavanoids                   0.236815   -0.411007  0.115077
nonflavanoid_phenols        -0.155929    0.292977  0.186230
proanthocyanins              0.136698   -0.220746  0.009652
color_intensity              0.546364    0.248985  0.258887
hue                         -0.071747   -0.561296 -0.074667
```

```
od280/od315_of_diluted_wines  0.072343   -0.368710  0.003911
proline                       0.643720   -0.192011  0.223626

                              alcalinity_of_ash  magnesium  total_phenols  \
alcohol                             -0.310235   0.270798       0.289101
malic_acid                           0.288500  -0.054575      -0.335167
ash                                  0.443367   0.286587       0.128980
alcalinity_of_ash                    1.000000  -0.083333      -0.321113
magnesium                           -0.083333   1.000000       0.214401
total_phenols                       -0.321113   0.214401       1.000000
flavanoids                          -0.351370   0.195784       0.864564
nonflavanoid_phenols                 0.361922  -0.256294      -0.449935
proanthocyanins                     -0.197327   0.236441       0.612413
color_intensity                      0.018732   0.199950      -0.055136
hue                                 -0.273955   0.055398       0.433681
od280/od315_of_diluted_wines        -0.276769   0.066004       0.699949
proline                             -0.440597   0.393351       0.498115

                              flavanoids  nonflavanoid_phenols  \
alcohol                         0.236815             -0.155929
malic_acid                     -0.411007              0.292977
ash                             0.115077              0.186230
alcalinity_of_ash              -0.351370              0.361922
magnesium                       0.195784             -0.256294
total_phenols                   0.864564             -0.449935
flavanoids                      1.000000             -0.537900
nonflavanoid_phenols           -0.537900              1.000000
proanthocyanins                 0.652692             -0.365845
color_intensity                -0.172379              0.139057
hue                             0.543479             -0.262640
od280/od315_of_diluted_wines    0.787194             -0.503270
proline                         0.494193             -0.311385

                              proanthocyanins  color_intensity       hue  \
alcohol                              0.136698         0.546364 -0.071747
malic_acid                          -0.220746         0.248985 -0.561296
ash                                  0.009652         0.258887 -0.074667
alcalinity_of_ash                   -0.197327         0.018732 -0.273955
magnesium                            0.236441         0.199950  0.055398
total_phenols                        0.612413        -0.055136  0.433681
flavanoids                           0.652692        -0.172379  0.543479
nonflavanoid_phenols                -0.365845         0.139057 -0.262640
proanthocyanins                      1.000000        -0.025250  0.295544
color_intensity                     -0.025250         1.000000 -0.521813
hue                                  0.295544        -0.521813  1.000000
od280/od315_of_diluted_wines         0.519067        -0.428815  0.565468
proline                              0.330417         0.316100  0.236183
```

```
                              od280/od315_of_diluted_wines   proline
alcohol                                           0.072343  0.643720
malic_acid                                       -0.368710 -0.192011
ash                                               0.003911  0.223626
alcalinity_of_ash                                -0.276769 -0.440597
magnesium                                         0.066004  0.393351
total_phenols                                     0.699949  0.498115
flavanoids                                        0.787194  0.494193
nonflavanoid_phenols                             -0.503270 -0.311385
proanthocyanins                                   0.519067  0.330417
color_intensity                                  -0.428815  0.316100
hue                                               0.565468  0.236183
od280/od315_of_diluted_wines                      1.000000  0.312761
proline                                           0.312761  1.000000
```

[8]:
```
# Second multivariate analysis for numerical features
import seaborn as sn # Import seaborn library to plot a pairplot

# Create a pairplot and display it
sn.pairplot(wine_copy, kind='scatter', diag_kind='auto', markers='o', height=2.
 ↪5, aspect=1)
```

[8]:
```
<seaborn.axisgrid.PairGrid at 0x7fa1015d4f40>
```



1.1.3 (c) Group observations by the categorical feature & compute the corresponding median for each remaining numerical feature. Include the code in your report.

[9]:
```
# Grouping by the target
group_by_target = wine.groupby('target')

# Calculating the median for each numerical feature
median_by_target = group_by_target.median()

# Displaying the median
print(median_by_target)
```

```
        alcohol  malic_acid   ash  alcalinity_of_ash  magnesium  \
```

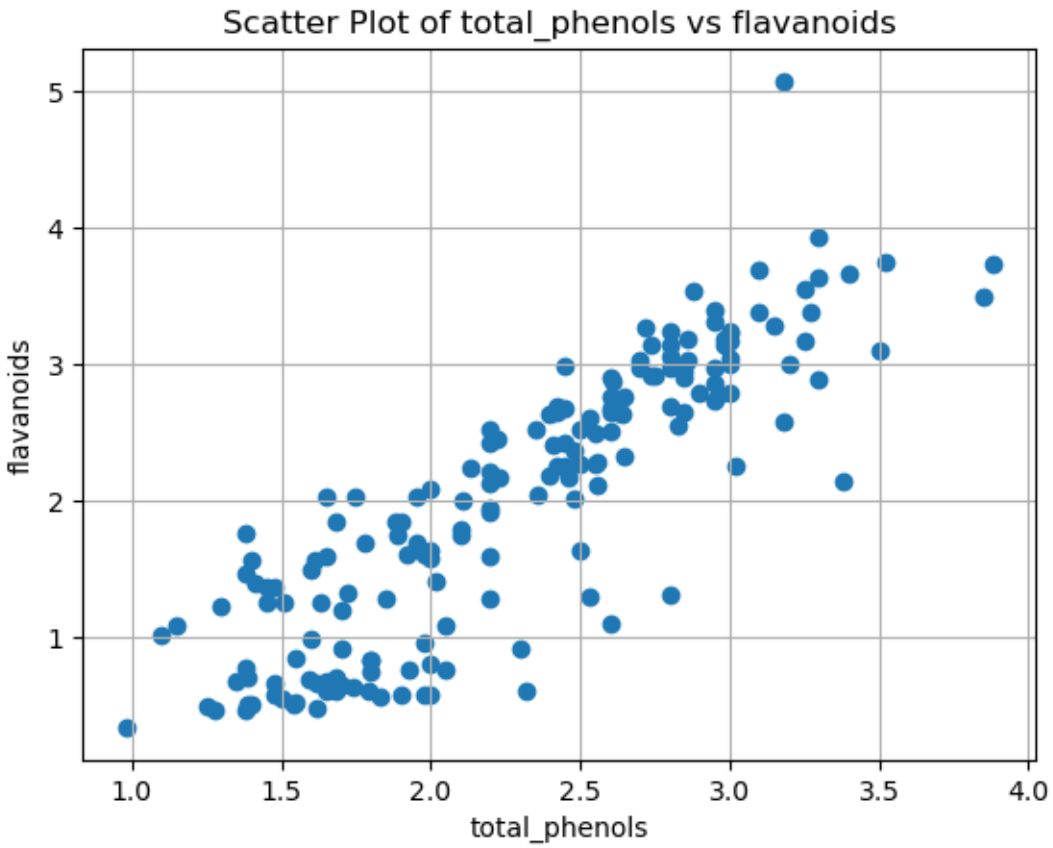
```
target
0        13.750       1.770  2.44                16.8      104.0
1        12.290       1.610  2.24                20.0       88.0
2        13.165       3.265  2.38                21.0       97.0

        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
```

1.1.4 (d) Create a scatter plot for the pair of distinct numerical features with the highest correlation. Include the code in your report.

[10]:
```
# Find the pair of distinct features with the highest correlation
max_corr = correlation_matrix.abs().unstack().sort_values(ascending=False)

# Selecting the highest correlation (excluding self-correlation)
max_corr = max_corr[max_corr != 1].idxmax()

# Extract the names of the features with the highest correlation
feature1, feature2 = max_corr

# Display the names of the features with the highest correlation
print(f"The pair of distinct numerical features with the highest correlation:␣
↪{feature1} and {feature2}")
```

```
The pair of distinct numerical features with the highest correlation: flavanoids
and total_phenols
```

[11]:
```
# Plotting a scatter plot
plt.scatter(wine_copy['total_phenols'], wine_copy['flavanoids'])
plt.xlabel('total_phenols')
plt.ylabel('flavanoids')
plt.title('Scatter Plot of total_phenols vs flavanoids')
plt.grid(True)
plt.show() # Display the scatter plot
```



2 Q2) Consider the following sales data: [5, 20, 1, 6, 13, 8, 9, 11, 17, 7, 2, 12]

2.0.1 Apply the Equal-Frequency Binning to the Sales Data:

Equal frequency binning involves dividing the given sales data into bins, ensuring an `equal` number of data points within each bin. The goal is to achieve a `balanced distribution` across these bins.

To follow good practice we will sort the data points into `ascending` order.

Sales data sorted = [1,2,5,6,7,8,9,11,12,13,17,20]

We have been asked to split the data points into 3 bins; therefore, each bin will contain exactly `four` distinct data points:

- **Bin 1:** [1,2,5,6]
- **Bin 2:** [7,8,9,11]
- **Bin 3:** [12,13,17,20]

2.0.2 Apply the Smoothing by Bin Boundaries to the Sales Data:

The technique of smoothing by bin boundaries involves determining the `maximum` and `minimum` values within each bin, using these values as the boundaries, and then replacing each value in the bin with the closest boundary value.

As good practice, the sales data was sorted in ascending order before binning:

Sales data sorted in ascending order: [1, 2, 5, 6, 7, 8, 9, 11, 12, 13, 17, 20]

Following the sorted data, three distinct bins were formed (as asked), each containing four data points:

- **Bin 1:** [1, 2, 5, 6]
- **Bin 2:** [7, 8, 9, 11]
- **Bin 3:** [12, 13, 17, 20]

For Bin 1, the upper boundary is 6, and the lower boundary is 1. After applying the smoothing by bin boundaries, Bin 1 appears as:

- **Bin 1 (Smoothed):** [1, 1, 6, 6]

Values `[1, 2]` are closer to the lower boundary 1. Values `[5,6]` are closer to the upper boundary 6.

We follow the same concept as above for the other bins:

- **Bin 2 (Smoothed):** [7, 7, 11, 11]
  - As 9 is equidistant from 7 and 11. We have a 'tie-breaking' senario and we can chose either 7 or 11. We have selcted 11 in this case. It is acceptable for all such values to be consistently tied to the same boundary.
- **Bin 3 (Smoothed):** [12, 12, 20, 20]

3 Q3) Load the file country-income.csv which includes numerical and categorical features.

[12]:
```
# Import the necessary libraries
from sklearn.impute import SimpleImputer

# Read the dataset into a DataFrame 'CI'
CI = pd.read_csv('country-income.csv', header=0)

# Display the data frame to see its contents
print(CI)
```

```
   Region   Age   Income Online Shopper
0   India  49.0  86400.0             No
1  Brazil  32.0  57600.0            Yes
2     USA  35.0  64800.0             No
3  Brazil  43.0  73200.0             No
4     USA  45.0      NaN            Yes
5   India  40.0  69600.0            Yes
```

```
6  Brazil   NaN  62400.0            No
7   India  53.0  94800.0           Yes
8     USA  55.0  99600.0            No
9   India  42.0  80400.0           Yes
```

3.1 Perform data cleaning to replace any NaN values with the mean value of that particular feature.

[13]:
```
# Separate numerical and categorical columns
CI_num = CI.select_dtypes('float64').copy()
CI_cat = CI.select_dtypes(object).copy()

# Impute missing values in numerical columns with their means
num_imputer = SimpleImputer(strategy='mean')
num_imputer = num_imputer.fit_transform(CI_num)
CI_num_imputed = pd.DataFrame(num_imputer, columns=CI_num.columns, index=CI_num.
 ↪index)
```

3.2 Then replace any categorical features with numerical features. Display the resulting dataset.

[14]:
```
from sklearn.preprocessing import LabelEncoder

# Encode categorical columns
label_encoder = LabelEncoder()
CI_cat_encoded = CI_cat.apply(lambda x: label_encoder.fit_transform(x))

# Combine the processed numerical and categorical data.
country_income = pd.concat([round(CI_num_imputed), CI_cat_encoded], axis=1) #␣
 ↪Round function used here as we want the age to be represented as a integer␣
 ↪and not a floating point number.

# Display the resulting dataset
print(country_income)
```

```
    Age   Income  Region  Online Shopper
0  49.0  86400.0       1               0
1  32.0  57600.0       0               1
2  35.0  64800.0       2               0
3  43.0  73200.0       0               0
4  45.0  76533.0       2               1
5  40.0  69600.0       1               1
6  44.0  62400.0       0               0
7  53.0  94800.0       1               1
8  55.0  99600.0       2               0
9  42.0  80400.0       1               1
```

4 Q4) Load the file shoesize.csv, which includes measurements of shoe size and height (in inches) for 408 subjects, both female and male.

[15]:
```
# Read the CSV file
ss = pd.read_csv('shoesize.csv', header=0)

# Display the content of the CSV file
print(ss)
```

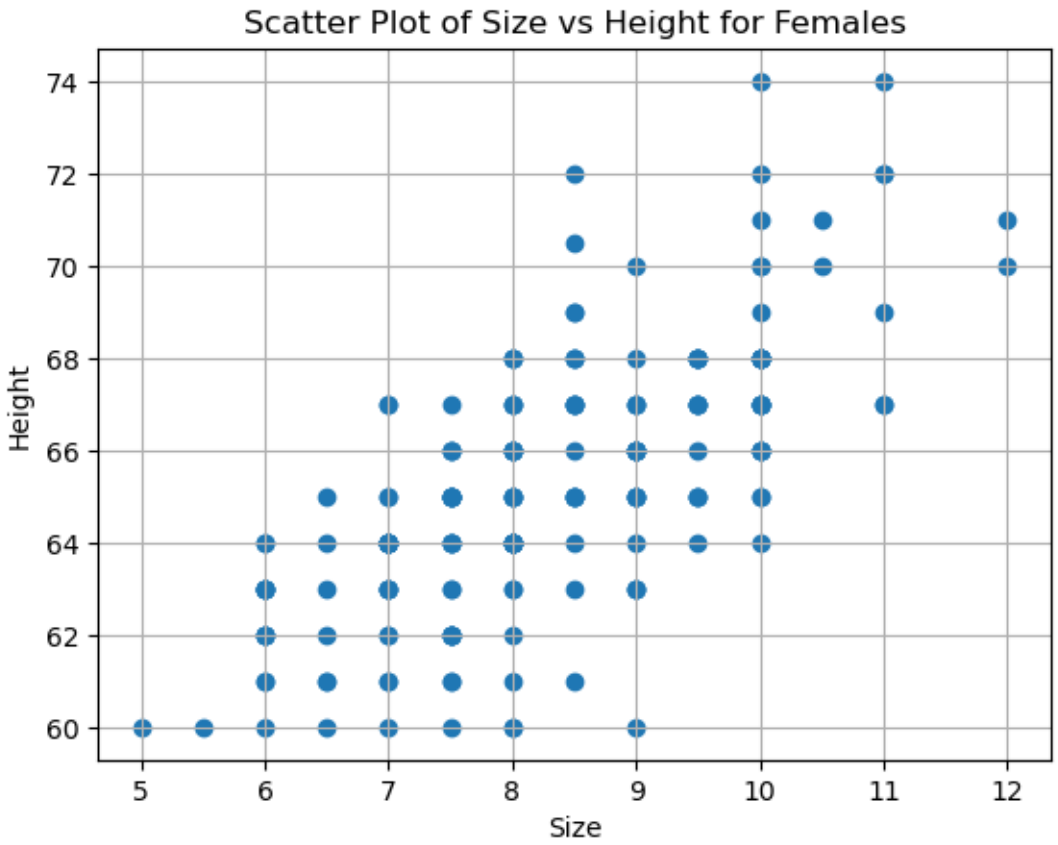
```
     Index Gender  Size  Height
0        1      F   5.5    60.0
1        2      F   6.0    60.0
2        3      F   7.0    60.0
3        4      F   8.0    60.0
4        5      F   8.0    60.0
..     …    …   …     …
403    404      M  13.0    78.0
404    405      M  13.0    78.0
405    406      M  14.0    78.0
406    407      M  15.0    80.0
407    408      M  15.0    81.0

[408 rows x 4 columns]
```

[16]:
```
# Set the 'Index' column as the index of the data frame
ss = pd.read_csv('shoesize.csv', header=0, index_col='Index')
```

4.1 Plot the scatterplots of shoe size versus height for female and male subjects separately.

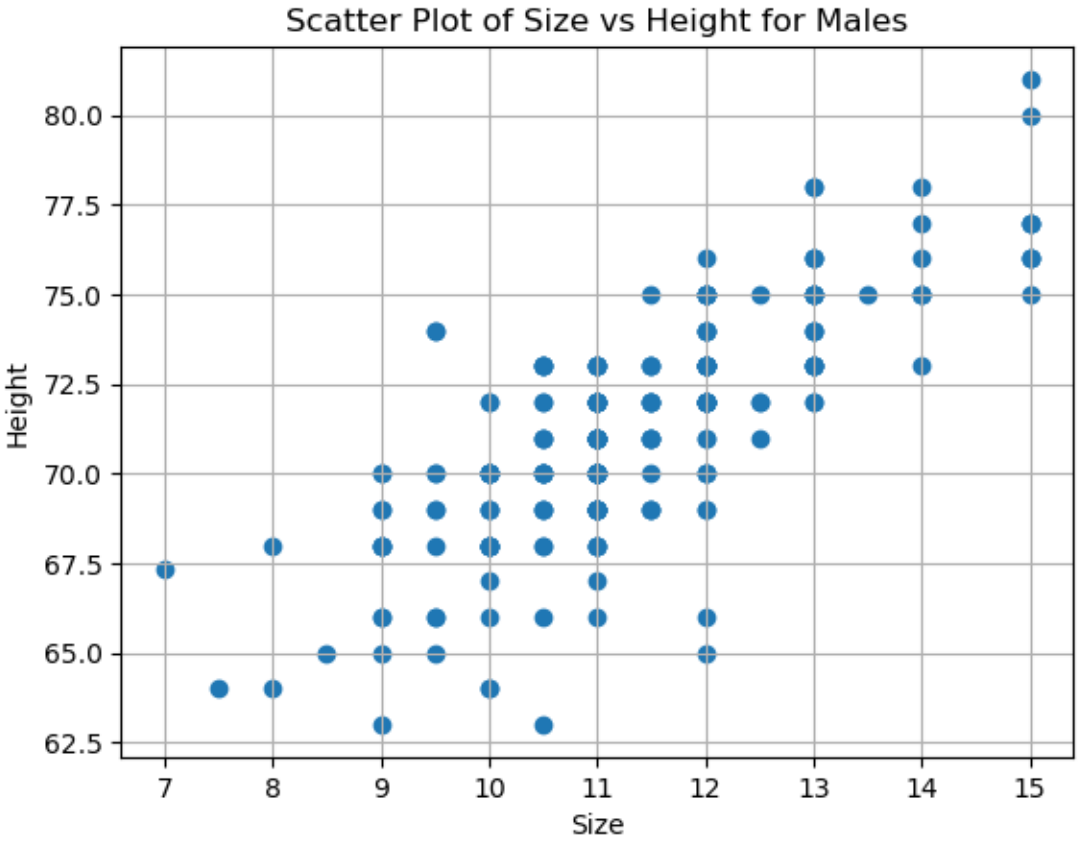
[17]:
```
# Filter the data frame to get rows where gender is female
ss_female = ss[ss['Gender'] == 'F']

# Create a scatter plot of size against height for female data
plt.scatter(ss_female['Size'], ss_female['Height'])

# Label the x and y axes of the plot
plt.xlabel('Size')
plt.ylabel('Height')

# Set the title of the plot
plt.title('Scatter Plot of Size vs Height for Females')

# Display a grid on the plot
plt.grid(True)
```

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



[18]:
```
# Filter the data frame to get rows where gender is male
ss_male = ss[ss['Gender'] == 'M']

# Create a scatter plot of size against height for male data
plt.scatter(ss_male['Size'], ss_male['Height'])

# Label the x and y axes of the plot
plt.xlabel('Size')
plt.ylabel('Height')

# Set the title of the plot
plt.title('Scatter Plot of Size vs Height for Males')

# Display a grid on the plot
plt.grid(True)
```

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



4.2 Compute the Pearson's correlation coefficient of shoe size versus height for female and male subjects separately.

[19]:
```
# Calculate the Pearson correlation between size and height for females
female_corr = ss_female[['Size', 'Height']].corr(method='pearson')

# Display the Pearson correlation coefficient for females
print(female_corr)
```

```
            Size    Height
Size    1.000000  0.707812
Height  0.707812  1.000000
```

[20]:
```
# Calculate the Pearson correlation between size and height for males
male_corr = ss_male[['Size', 'Height']].corr(method='pearson')

# Display the Pearson correlation coefficient for males
```

```
print(male_corr)
```

```
            Size    Height
Size    1.000000  0.767709
Height  0.767709  1.000000
```

4.3 What can be inferred by the scatterplots and computed correlation coefficients?

4.3.1 Scatter Plots Insight:

The scatter plots illustrate a discernible `positive correlation` between shoe size and height for both males and females. Visually, this is depicted by a general trend where an increase in shoe size corresponds to an increase in height for the data points.

4.3.2 Pearson's Correlation:

The calculated Pearson's correlation coefficients indicate a `positive relationship` between shoe size and height for both genders. For females, the correlation coefficient value is `0.707812`, and for males, it is `0.767709`. These values affirm the observations made from the scatter plots, reinforcing the presence of a positive correlation between shoe size and height for both genders.

In summary, the correlation coefficients and scatter plots prove a positive relationship between shoe size and height for both males and females. From our findings, we can infer that the larger the shoe size, the more likely the individual will be taller. These findings are consistent across genders, indicating a general trend applicable to the entire dataset.

5 Q5) Using the wine dataset from question 1, perform Principal Component Analysis (PCA) with 2 components.

[21]:
```
from sklearn.decomposition import PCA

# Define the number of principal components
numComponents = 2

# Initialise a PCA model with the specified number of components
pca = PCA(n_components=numComponents)

# Fit the PCA model to the 'wine_copy' dataset (without standardisation)
pca.fit(wine_copy)

# Project the original data onto the new PCA space
projected = pca.transform(wine_copy)

# Create a data frame to hold the transformed data with columns 'pc1' and 'pc2'
projected = pd.DataFrame(projected, columns=['pc1', 'pc2'])

# Add the original target column back to the projected data frame
```

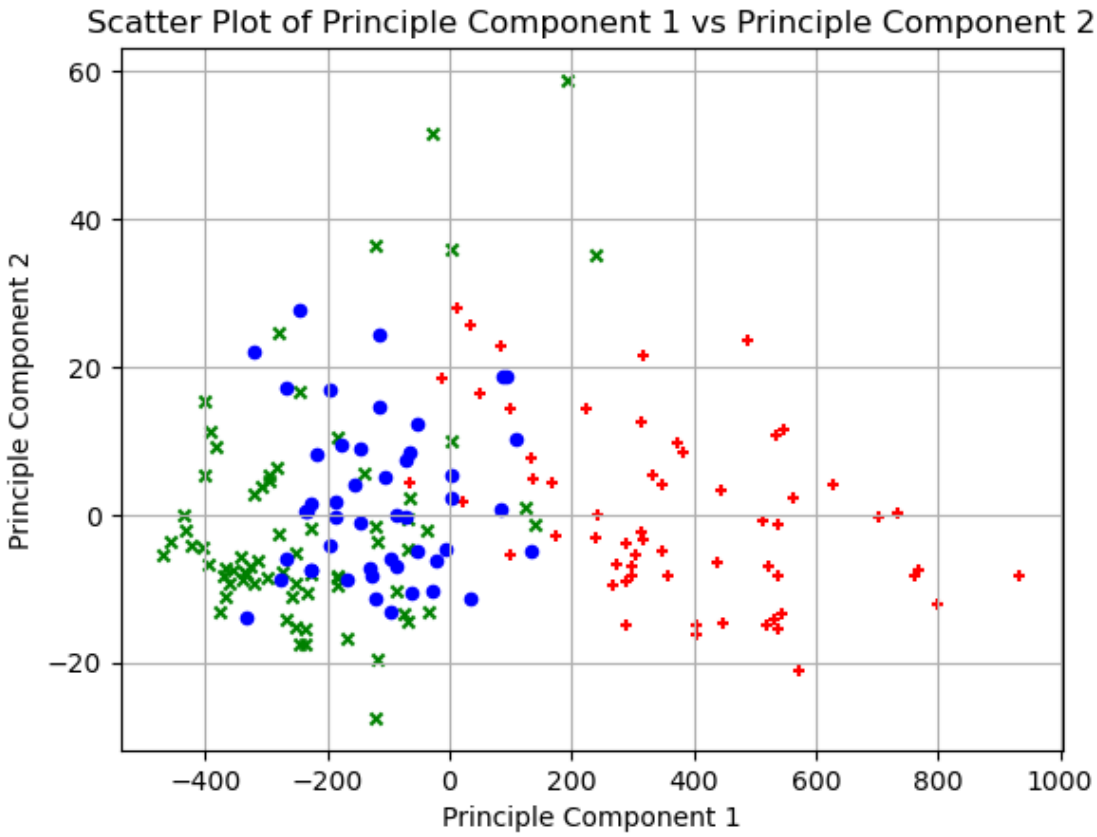
```
projected['target'] = wine['target']

# Display the transformed dataset in the new PCA space
print(projected)
```

```
            pc1        pc2  target
0    318.562979  21.492131       0
1    303.097420  -5.364718       0
2    438.061133  -6.537309       0
3    733.240139   0.192729       0
4    -11.571428  18.489995       0
..          …        …     …
173   -6.980211  -4.541137       2
174    3.131605   2.335191       2
175   88.458074  18.776285       2
176   93.456242  18.670819       2
177 -186.943190  -0.213331       2

[178 rows x 3 columns]
```

5.1 Plot the scatterplot of all samples along the two principal components, color-coded according to the "target" column (this column is the class and should not be used in the PCA analysis).

[22]:
```
import matplotlib.pyplot as plt

# Define colors and marker types for each target class
colors = {0: 'r', 1: 'g', 2: 'b'}
markerTypes = {0: '+', 1: 'x', 2: 'o'}

# Plotting the scatter plot for each class in the PCA-transformed data
for num in markerTypes:
    # Extract data points corresponding to the specific target class
    d = projected[projected['target'] == num]
    # Scatter plot for the target class with specific color, size and marker␣
 ↪type
    plt.scatter(d['pc1'], d['pc2'], c=colors[num],s=20, marker=markerTypes[num])

# Label the x and y axes of the plot
plt.xlabel('Principle Component 1')
plt.ylabel('Principle Component 2')

# Set the title of the plot
plt.title('Scatter Plot of Principle Component 1 vs Principle Component 2')

# Display a grid on the plot
plt.grid(True)
```

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



5.2 What insights can you obtain by viewing the scatterplot of the principal components?

5.2.1 Observations from the PCA Scatter Plot Analysis:

The prevalence of data points clustering towards the lower end of the graph indicates a critical inference regarding the principal components derived from the PCA analysis.

Capturing Data Variance: The concentration of points towards this lower spectrum strongly suggests that the initial principal components, notably PC1 and PC2, encompass the substantial majority of variance within the dataset. This signifies a dense representation of critical information within these dimensions.

Discriminatory Power and Classification Challenges: The discernible clustering towards this lower space reflects that these principal components may lack robust discriminatory power among the various classes in the data. As a consequence, the task of classifying or distinctly differentiating data points based solely on these components could be challenging.

Class Separability Challenges: The absence of clear separability among the classes within the reduced two-dimensional space (PC1 vs. PC2) signifies a lack of distinct boundaries or discernible differences among the diverse groups present in the dataset. This difficulty persists despite employing different colours to seprate classes, as the points exhibit significant overlap (we cannot easily distinguish that samples that belong to one class from samples that belong to another class).

The implications drawn from the lower-end clustering in the PCA scatter plot point towards the dominance of variance within the initial principal components, potential limitations in discriminatory power, and the resultant challenge in effectively distinguishing classes solely based on these reduced dimensions.

5.3 Why is this happening?

5.3.1 Challenges in Distinguishing Classes in PCA Scatter Plot Analysis:

The difficulty in distinctly differentiating samples belonging to various classes in a Principal Component Analysis (PCA) scatter plot arises from PCA's inherent nature and objectives.

Reasons for the Lack of Distinct Separation:

1. **Variance Maximisations in PCA:** PCA, primarily aimed at capturing the maximum variance within the dataset, projects data onto a new set of dimensions (principal components). This focus on maximising overall data structure prioritises variance capture rather than preserving specific class boundaries.
2. **Emphasis on Global Data Structure:** The fundamental objective of PCA lies in uncovering patterns that explain the most variance across the entire dataset. However, it does not explicitly consider optimising class separability, potentially leading to overlap among different classes in the reduced-dimensional space.
3. **Diminished Discriminatory Features in Lower Dimensions:** The transformation of data into a lower-dimensional space, such as from higher dimensions to a two-dimensional scatter plot, might result in the loss of finer details and nuanced features that distinguish different classes in the original high-dimensional space.
4. **Inherent Class Overlaps and Complex Boundaries:** Real-world datasets often overlap classes due to shared characteristics or intricate relationships. This intricacy makes it challenging to separate classes using only a few principal components effectively.
5. **Impact of No Data Preprocessing:** The absence of essential data preprocessing steps before PCA could significantly contribute to the challenges in class separation. Normalisation, handling missing values, dealing with outliers, feature selection, and categorical variable encoding play vital roles in enhancing the interpretability and effectiveness of PCA. Without preprocessing, biases, scaling differences, or irrelevant features impact PCA's efficacy, contributing to class distinction challenges.

5.4 What can be done to the data prior to performing PCA in order to alleviate this issue?

Data Preprocessing: Standardisation

Standardisation (z-score normalisation) rescales features with a mean of 0 and a standard deviation of 1. This process transforms the data to a common scale, ensuring that all features contribute equally to the analysis. In the context of PCA and dealing with overlapping classes, standardisation can be beneficial for the following reasons:

1. **Scaling Features Equally:** PCA involves the computation of feature variances, and features with larger scales might dominate the analysis, leading to incorrect assumptions about the importance of different features. Standardisation brings all features to a common scale, preventing one feature from dominating due to its larger magnitude.
2. **Addressing Varied Ranges:** If one feature has a range of values between 0 and 100, while another ranges from 0 to 1,000, the scale difference can impact the performance of PCA. Standardising the features would ensure that both contribute equally to the principal components' calculation.
3. **Improving Interpretability:** Standardisation aids in making the data more interpretable and more accessible to compare. It doesn't change the shape of the data distribution. Still, it repositions it to have a mean of 0 and a standard deviation of 1, simplifying the interpretation of coefficients in the resulting analysis.

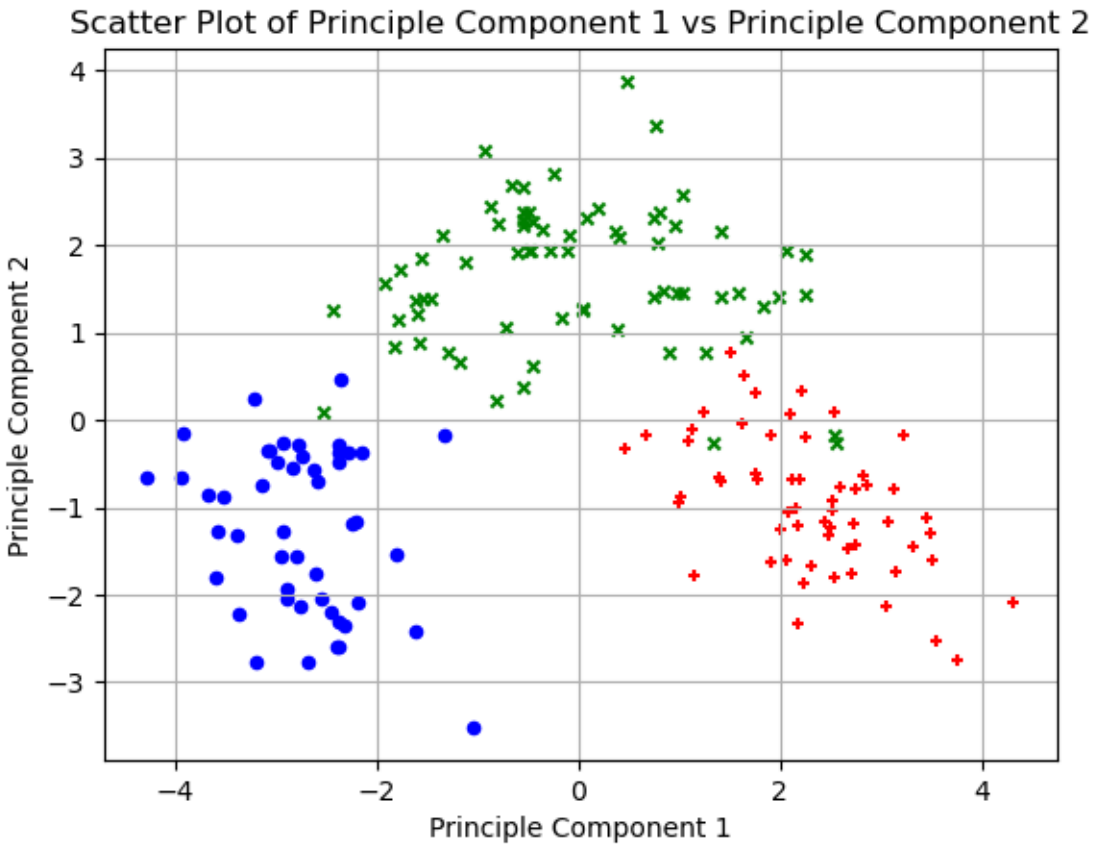
Implementing standardisation as a preprocessing step before PCA is crucial to ensure that the principal components are calculated on a uniform scale and effectively capture the variability in the data, thereby helping to address the issue of overlapping classes.

5.5 Perform and plot PCA again

[23]:
```
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

# Load the wine dataset
data = load_wine()

# Create a pandas data frame using the dataset's data and feature names
wine = pd.DataFrame(data.data, columns=data.feature_names)

# Add the 'target' column to the data frame to represent the class labels
wine['target'] = pd.Series(data.target)

# Features list
features = ['alcohol', 'malic_acid', 'ash', 'alcalinity_of_ash', 'magnesium',␣
 ↪'total_phenols', 'flavanoids', 'nonflavanoid_phenols', 'proanthocyanins',␣
 ↪'color_intensity', 'hue', 'od280/od315_of_diluted_wines', 'proline']

# Separating out the features from the 'wine' data frame
x = wine.loc[:, features].values

# Separating out the target
y = wine.loc[:, 'target'].values
```

```
# Standardising the features
x = StandardScaler().fit_transform(x)

# PCA with 2 principal components
pca = PCA(n_components=2)

# Perform PCA on the standardised features to derive principal components
principalDf = pd.DataFrame(data=pca.fit_transform(x), columns=['pc1', 'pc2'])

# Combine the principal components with the 'target' column in a final data␣
 ↪frame
finalDf = pd.concat([principalDf, wine[['target']]], axis=1)
```

[24]:
```
colors = {0: 'r', 1: 'g', 2: 'b'}  # Assigning colors for different target␣
 ↪classes
markerTypes = {0: '+', 1: 'x', 2: 'o'}  # Assigning marker types for different␣
 ↪target classes

# Iterate through marker types and plot a scatter plot for each target class
for num in markerTypes:
    # Filter data points for each target class
    d = finalDf[finalDf['target'] == num]
    # Plot a scatter plot for the target class with specific color, size and␣
 ↪marker type
    plt.scatter(d['pc1'], d['pc2'], c=colors[num],s=20, marker=markerTypes[num])

# Label the x and y axes of the plot
plt.xlabel('Principle Component 1')
plt.ylabel('Principle Component 2')

# Set the title of the plot
plt.title('Scatter Plot of Principle Component 1 vs Principle Component 2')

# Display a grid on the plot
plt.grid(True)

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



Now are the different classes (quite) distinctive one from the other?

The analysis of the new scatter plot demonstrates a considerable distinctiveness among the different classes, signifying a robust relationship between the two principal components.

The clear separation observed in the scatter plot indicates a strong relationship between the two principal components. This delineation implies that data points within each class exhibit higher similarity to each other than they do to data points in other classes. The evident distinctiveness highlights the effectiveness of these principal components in representing the inherent structures and relationships present in the data.

The first principal component (PC1) appears to be more crucial in distinguishing between the classes than the second principal component (PC2). The spread of data points along the PC1 axis is more pronounced than along the PC2 axis.

There might be underlying data structure not captured by the first two principal components, as the data points are not perfectly separated along either axis.

6 Q6) In Lab session 3 you had created and visualised a heatmap for the distance matrix for the graduation_rate.csv. You may have noticed that the distance matrix visualisation is not very informative. However, it is still possible to infer that the average distance between students whose parents only have some high school education and students whose parents have a master's degree is larger than the average distance between students whose parents only have some high school education. Explain how this inference is possible from the visualisation.

Heatmaps utilise colours to convey values, whereas darker hues typically represent higher values or distances within a distance matrix. In this context, larger values suggest increased dissimilarity or separation between pairs of data points, reflecting greater distances in terms of students' graduation rates.

The visual interpretation relies on recognising that darker colours in the heatmap signify more considerable distances or dissimilarities between students. The inference drawn is that the average distance between students with significantly different parental educational backgrounds is more significant than between students with similar parental educational levels. This inference is based on the consistent pattern of darker colours in the heatmap between these groups, denoting more significant dissimilarity in graduation rates and, consequently, more considerable distances within the dataset.

7 Question 7 Use the file country-income.csv and perform the following:

7.1 (a) Load the CSV file using Cubes, create a JSON file for the data cube model, and create a data cube for the data. Use as dimensions the region, age, and online shopper fields. Use as measure the income. Define aggregate functions in the data cube model for the total, average, minimum, and maximum income.

[27]:
```
from collections.abc import MutableMapping

class CustomMutableMapping(MutableMapping):
    pass

import sys

sys.modules['collections'].MutableMapping = CustomMutableMapping
```

[28]:
```
# Install the 'cubes' library and a specific version of sqlalchemy
!pip install cubes
!pip install sqlalchemy==1.3.20
```

```
from sqlalchemy import create_engine
from cubes.tutorial.sql import create_table_from_csv

# Create an engine connecting to an sqlite database
engine = create_engine('sqlite:///data.sqlite')

# Create a table named 'income_cube' in the sqlite database
create_table_from_csv(engine,
                      "country-income.csv",
                      table_name="income_cube",
                      fields=[
                          ("Region", "string"),
                          ("Age", "integer"),
                          ("Income", "integer"),
                          ("Online Shopper", "string")],
                      create_id=True
                     )
```

```
Requirement already satisfied: cubes in /opt/conda/lib/python3.10/site-packages
(1.1)
Requirement already satisfied: python-dateutil in
/opt/conda/lib/python3.10/site-packages (from cubes) (2.8.2)
Requirement already satisfied: jsonschema in /opt/conda/lib/python3.10/site-
packages (from cubes) (4.19.2)
Requirement already satisfied: expressions>=0.2.3 in
/opt/conda/lib/python3.10/site-packages (from cubes) (0.2.3)
Requirement already satisfied: grako>=3.9.3 in /opt/conda/lib/python3.10/site-
packages (from expressions>=0.2.3->cubes) (3.99.9)
Requirement already satisfied: attrs>=22.2.0 in /opt/conda/lib/python3.10/site-
packages (from jsonschema->cubes) (23.1.0)
Requirement already satisfied: jsonschema-specifications>=2023.03.6 in
/opt/conda/lib/python3.10/site-packages (from jsonschema->cubes) (2023.7.1)
Requirement already satisfied: referencing>=0.28.4 in
/opt/conda/lib/python3.10/site-packages (from jsonschema->cubes) (0.30.2)
Requirement already satisfied: rpds-py>=0.7.1 in /opt/conda/lib/python3.10/site-
packages (from jsonschema->cubes) (0.10.6)
Requirement already satisfied: six>=1.5 in /opt/conda/lib/python3.10/site-
packages (from python-dateutil->cubes) (1.16.0)
Requirement already satisfied: sqlalchemy==1.3.20 in
/opt/conda/lib/python3.10/site-packages (1.3.20)
```

[29]:
```
from cubes import Workspace

# Initialise a Workspace
workspace = Workspace()

# Register the default store for the workspace, specifying it as an SQL store
```

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

# Import the model into the workspace from the 'income_cube.json' file
workspace.import_model("income_cube.json")

# Retrieve a cube named 'income_cube' from the workspace
cube = workspace.cube("income_cube")
```

7.2 (b) Using the created data cube and data cube model, produce aggregate results for:

7.2.1 i) the whole data cube;

[30]:
```
# Create a browser to interact with the 'income_cube' cube in the workspace
browser = workspace.browser(cube)

# Perform an aggregation operation
result = browser.aggregate()

# Retrieve a summary of the aggregated data
result = result.summary

# Display the summary result, which contains aggregated data from the cube
print(result)
```

```
{'income_sum': 688800.0, 'income_max': '', 'income_min': 57600, 'income_avg':
68880.0, 'record_count': 10}
```

7.2.2 ii) results per region;

[31]:
```
# Perform an aggregation operation with a drilldown on the 'Region' dimension
result = browser.aggregate(drilldown=["Region"])

# Iterate through the results of the drilldown operation
for record in result:
    print(record)
```

```
{'Region': 'Brazil', 'income_sum': 193200, 'income_max': 73200, 'income_min':
57600, 'income_avg': 64400.0, 'record_count': 3}
{'Region': 'India', 'income_sum': 331200, 'income_max': 94800, 'income_min':
69600, 'income_avg': 82800.0, 'record_count': 4}
{'Region': 'USA', 'income_sum': 164400.0, 'income_max': '', 'income_min': 64800,
'income_avg': 54800.0, 'record_count': 3}
```

7.2.3 iii) results per online shopping activity;

[32]:
```
# Perform an aggregation operation with a drilldown on the 'Online_Shopper'␣
↪dimension
result = browser.aggregate(drilldown=["Online_Shopper"])

# Iterate through the results of the drilldown operation
for record in result:
    print(record)
```

```
{'Online_Shopper': 'No', 'income_sum': 386400, 'income_max': 99600,
'income_min': 62400, 'income_avg': 77280.0, 'record_count': 5}
{'Online_Shopper': 'Yes', 'income_sum': 302400.0, 'income_max': '',
'income_min': 57600, 'income_avg': 60480.0, 'record_count': 5}
```

7.2.4 iv) results for all people aged between 40 and 50.

[33]:
```
import cubes as cubes

# Define a range for the 'Age' dimension from 40 to 50
cuts = [cubes.RangeCut("Age", ["40"], ["50"])]

# Create a cell for the cube, specifying the cut on the 'Age' dimension
cell = cubes.Cell(cube, cuts)

# Perform an aggregation operation with the specified cell
result = browser.aggregate(cell)

# Retrieve a summary of the aggregated data from the specific cell
print(result.summary)
```

```
{'income_sum': 309600.0, 'income_max': '', 'income_min': 69600, 'income_avg':
61920.0, 'record_count': 5}
```

8 Q8) Consider a dataset that contains only two observations x1 = (1,2) and x2 = (−1,0). Suppose that the class of the first observation is y1 = 1 and that the class of the second observation is y2 = 0.

8.1 How would a 1-nearest neighbour classifier based on the Euclidean distance classify the observation x3 = (3,2) and why?

In evaluating the classification of the observation (x3 = (3,2)) based on the 1-NN (nearest neighbour) algorithm, we computed the Euclidean distance between (x3) and the existing observations.

The distance between (x3) and (x1 = (1,2)) was found to be 2, while the distance between (x3) and (x2 = (-1,0)) was approximately 4.47.

Consequently, the nearest neighbor to (x3) is (x1) with a distance of 2. Therefore, (x3 = (3,2)) would be classified as (y1 = 1) under the 1-NN classification, as the nearest neighbor (x1) bears the class label (y1 = 1).

8.2 How would the same classifier classify the observation x4 = (0,1) and why?

In the context of the 1-NN (nearest neighbour) classification for the observation (x4 = (0,1)), the Euclidean distances were computed between (x4) and the existing observations.

The distance between (x4) and (x1 = (1,2)) was approximately 1.41, and the distance between (x4) and (x2 = (-1,0)) was also approximately 1.41.

Since both (x1) and (x2) were equidistant from (x4) with a distance of approximately 1.41, the tie was assumed to be broken by selecting the closest neighbour in the dataset, which is (x1).

Consequently, (x4 = (0,1)) would be classified as (y1 = 1) under the 1-NN classification, as the nearest neighbor (x1) bears the class label (y1 = 1).

In summary, both (x3 = (3,2)) and (x4 = (0,1)) would be classified as (y1 = 1) using the 1-NN classifier based on Euclidean distance.