mie-maarsh-joy-martin-assignment-1

February 3, 2024

0.0.1 Assignment-1: IE 7275 Data Mining Techniques

0.0.2 Question 1:

0.0.3 Part 1.1: Deciding whether to issue a loan to an applicant based on demographic and financial data (with reference to a database of similar data on prior customers)

Solution: Supervised Learning

Explanation: Since this involves in making informed decisions based on pre-trained historical data, which has the tendency to produce either of the 2 labeled outcome, which is approval or denial of the loan.

0.0.4 Part 1.2: In an online bookstore, making recommendations to customers concerning additional items to buy based on the buying patterns in prior transactions

Solution: Unsupervised Learning

Explanation: Since this pertains to recognizing patterns in customer actions without clear and distinct labels, focusing on grouping or linking items based on their transaction history, which may not be the right way.

0.0.5 Part 1.3: Identifying a network data packet as dangerous (virus, hacker attack) based on comparison to other packets whose threat status is known

Solution: Supervised Learning

Explaination: Since this task involves assigning roles to network packets – determining if they're risky or harmless based on past experiences(having the tendency to produce atmost 2 labelled outcomes).

0.0.6 Part 1.4: Identifying segments of similar customers

Solution: Unsupervised Learning

Explaination: Since this includes bringing together customers who share common traits, without having pre-trained labels, which will lead to multiple and untested outcomes.

0.0.7 Part 1.5: Predicting whether a company will go bankrupt based on comparing its financial data to those of similar bankrupt and nonbankrupt firms

Solution: Supervised Learning

Explanation: Since this case involves predicting or getting only either of 2 possible outcomes, whether going bankrupt or not, based on pre-trained historical data.

0.0.8 Part 1.6: Estimating the repair time required for an aircraft based on a trouble ticket

Solution: Supervised Learning

Explanation: Since this case involves that an informed decision can be made by using historical trained data model

0.0.9 Part 1.7: Automated sorting of mail by zip code scanning.

Solution: Supervised Learning

Explanation: Since in this process, the system classifies the e-mails into predetermined categories of zip codes, based on knowledge gained from historical data models.

0.0.10 Part 1.8: Printing of custom discount coupons at the conclusion of a grocery store checkout based on what you just bought and what others have bought previously

Solution: Unsupervised Learning

Explanation: In this process, the system figures out trends in what customers purchase and suggests personalized discounts based on those trends.

0.0.11 Question 2:

From the given problem, we are able to infer that the existing dataset containing numerous records have 7% of the data to be null. In order to remove them we have the following formula to calculate the total number of entries to be removed:

To ensure a dataset is complete, it must not contain any missing values. With a 7% chance of data being missing, the probability of a record having no missing values is 93% for each of the 35 attributes. The overall likelihood of a record being free of missing values across all variables is calculated as 0.93^35, resulting in approximately 7.88%. This implies a 92.11% chance (1 - 0.0788) of having one or more missing observations in a single record.

Out of 2000 records, around 92.11% or 1842 records are expected to have missing observations. If the strategy of deleting records with any missing data is employed, only 1079 records would be retained for analysis.

Therefore, The total number of observations removed would be 2000 - 158 = 1842.

0.0.12 Question 3:

There are 3 methods that the data can be normalised:

1) Min-Max Scaling: Xnormalised = (X - X.min()) / (X.max() - X.min())

This function scales the data to a specific range, majorly between 0 & 1. It is also sensitive to outliers.

2) Z-Score (Standardization): Z = (X - X.mean()) / X.std()

The function scales the data to have a mean of 0 and a standard deviation of 1, and is generally suitable for data having normally distributed features.

3) Robust Scaling Robust Scaling, utilizing the interquartile range (IQR), is less sensitive to outliers than Min-Max Scaling. It's particularly useful when dealing with datasets containing outliers, as it provides a balanced adjustment without compromising overall scaling accuracy.

Important Note: There would be a slight variation in the values obtained from sklearn and pandas. Eventhough they are conceptually the same, it uses a slight different scaling formula and internal calculations

Using Sklearn

[25]:
```
import pandas as pd
from sklearn.preprocessing import MinMaxScaler, StandardScaler

# Original Data
activity = {'Weight': [25, 56, 65, 32, 41, 49],
        'Walking_Steps': [49000, 156000, 99000, 192000, 39000, 57000]}

activity_df = pd.DataFrame(activity)

# Min-Max Scaling
scaler = MinMaxScaler()
activity_scale_df = pd.DataFrame(scaler.fit_transform(activity_df),␣
 ↪columns=activity_df.columns)

# Z-score Standardization
scaler = StandardScaler()
activity_Std_df = pd.DataFrame(scaler.fit_transform(activity_df),␣
 ↪columns=activity_df.columns)

# Decimal Scaling
activity_float_df = activity_df / 10**4
# Assuming 4 as the number of decimals

print("Data under study:")
print(activity_df)
```

```
print("\nMin-Max Scaled function for the data under study:")
print(activity_scale_df)
print("\nZ-score Standardized function for the data under study:")
print(activity_Std_df)
print("\nDecimal Scaled Data:")
print(activity_float_df)
```

```
Data under study:
   Weight  Walking_Steps
0      25          49000
1      56         156000
2      65          99000
3      32         192000
4      41          39000
5      49          57000

Min-Max Scaled function for the data under study:
   Weight  Walking_Steps
0   0.000       0.065359
1   0.775       0.764706
2   1.000       0.392157
3   0.175       1.000000
4   0.400       0.000000
5   0.600       0.117647

Z-score Standardized function for the data under study:
     Weight  Walking_Steps
0 -1.438597      -0.865431
1  0.829022       0.999021
2  1.487363       0.005808
3 -0.926554       1.626314
4 -0.268213      -1.039679
5  0.316979      -0.726033

Decimal Scaled Data:
   Weight  Walking_Steps
0  0.0025            4.9
1  0.0056           15.6
2  0.0065            9.9
3  0.0032           19.2
4  0.0041            3.9
5  0.0049            5.7
```

Using Pandas

[26]:
```
# Method 1: Min-Max Scaling
activity_scale_df_Pd = (activity_df - activity_df.min()) / (activity_df.max() -␣
↪activity_df.min())
```

```
# Method 2: Z-Score Standardization
activity_Std_df_Pd = (activity_df - activity_df.mean()) / activity_df.std()

# Method 3: Robust Scaling
Q1 = activity_df.quantile(0.25)
Q3 = activity_df.quantile(0.75)
IQR = Q3 - Q1
activity_robust_df_Pd = (activity_df - activity_df.median()) / IQR

# Display the results
print("f The data under study:")
print(activity_df)

print("\nMin-Max Scaling function for the data under study:")
print(activity_scale_df_Pd)

print("\nZ-Score Standardization function for the data under study:")
print(activity_Std_df_Pd)

print("\nRobust Scaling function for the data under study:")
print(activity_robust_df_Pd)
```

```
f The data under study:
   Weight  Walking_Steps
0      25          49000
1      56         156000
2      65          99000
3      32         192000
4      41          39000
5      49          57000

Min-Max Scaling function for the data under study:
   Weight  Walking_Steps
0   0.000       0.065359
1   0.775       0.764706
2   1.000       0.392157
3   0.175       1.000000
4   0.400       0.000000
5   0.600       0.117647

Z-Score Standardization function for the data under study:
     Weight  Walking_Steps
0 -1.313253      -0.790027
1  0.756790       0.911977
2  1.357770       0.005302
3 -0.845824       1.484614
```

```
4 -0.244844      -0.949093
5  0.289361      -0.662774

Robust Scaling function for the data under study:
   Weight  Walking_Steps
0   -1.00      -0.319559
1    0.55       0.859504
2    1.00       0.231405
3   -0.65       1.256198
4   -0.20      -0.429752
5    0.20      -0.231405
```

0.0.13 Question 4

[14]:
```
import matplotlib.pyplot as plt
import plotly.express as px
import pandas as pd
import seaborn as sns
import numpy as np
import plotly.graph_objects as go
import plotly.express as px

from plotly.subplots import make_subplots
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer
```

[2]:
```
#Loading of the data
FLIR_df = pd.read_csv('/Users/jaamiemaarshj/Desktop/  DAE Course Materials/Data␣
 ↪Mining/Assignment -1/FLIR_groups1and2.csv', header=[2])
display(FLIR_df.head())
```

```
  SubjectID  Unnamed: 1  T_offset1  Max1R13_1  Max1L13_1  aveAllR13_1  \
0  161117-1         NaN       0.58      34.98      35.36        34.44
1  161117-2         NaN       0.83      34.71      34.51        34.46
2  161117-3         NaN       0.85      35.70      35.44        35.00
3  161117-4         NaN       0.90      35.17      35.50        34.25
4  161117-5         NaN       1.08      35.33      35.55        34.31

   aveAllL13_1  T_RC1  T_RC_Dry1  T_RC_Wet1  …  aveOralM  Gender    Age  \
0        34.85  34.91      34.91      34.60  …     36.59    Male  41-50
1        34.24  34.68      34.68      34.44  …     37.19  Female  31-40
2        34.78  35.67      35.67      35.46  …     37.34  Female  21-30
3        35.00  35.14      35.14      35.08  …     37.09  Female  21-30
4        35.14  35.50      35.30      35.50  …     37.04    Male  18-20

                  Ethnicity  T_atm  Humidity  Distance  Cosmetics      Time  \
0                     White   24.0      28.0       0.8        NaN  12:43:46
1  Black or African-American   24.0      26.0       0.8        NaN  15:22:48
```

```
2                      White   24.0      26.0       0.8        NaN  15:52:56
3  Black or African-American   24.0      27.0       0.8        NaN  16:07:53
4                      White   24.0      27.0       0.8        NaN  16:28:06

       Date
0  16-11-17
1  16-11-17
2  16-11-17
3  16-11-17
4  16-11-17

[5 rows x 125 columns]
```

[4]:
```
# "date" field transformations

FLIR_df['Date'] = pd.to_datetime(FLIR_df['Date'])

# Fetch day, month, and year separately
FLIR_df['Day'] = FLIR_df['Date'].dt.day
FLIR_df['Month'] = FLIR_df['Date'].dt.month
FLIR_df['Year'] = FLIR_df['Date'].dt.year

display(FLIR_df.head())
```

```
  SubjectID  Unnamed: 1  T_offset1  Max1R13_1  Max1L13_1  aveAllR13_1  \
0  161117-1         NaN       0.58      34.98      35.36        34.44
1  161117-2         NaN       0.83      34.71      34.51        34.46
2  161117-3         NaN       0.85      35.70      35.44        35.00
3  161117-4         NaN       0.90      35.17      35.50        34.25
4  161117-5         NaN       1.08      35.33      35.55        34.31

   aveAllL13_1  T_RC1  T_RC_Dry1  T_RC_Wet1  …                  Ethnicity  \
0        34.85  34.91      34.91      34.60  …                      White
1        34.24  34.68      34.68      34.44  …  Black or African-American
2        34.78  35.67      35.67      35.46  …                      White
3        35.00  35.14      35.14      35.08  …  Black or African-American
4        35.14  35.50      35.30      35.50  …                      White

   T_atm  Humidity  Distance  Cosmetics      Time       Date  Day  Month  Year
0   24.0      28.0       0.8        NaN  12:43:46 2017-11-16   16     11  2017
1   24.0      26.0       0.8        NaN  15:22:48 2017-11-16   16     11  2017
2   24.0      26.0       0.8        NaN  15:52:56 2017-11-16   16     11  2017
3   24.0      27.0       0.8        NaN  16:07:53 2017-11-16   16     11  2017
4   24.0      27.0       0.8        NaN  16:28:06 2017-11-16   16     11  2017

[5 rows x 128 columns]
```

[5]:
```
#Checking out for unique values in the below field.
diff_values = FLIR_df['Cosmetics'].unique()
display(diff_values)

FLIR_df['Cosmetics'] = FLIR_df['Cosmetics'].fillna(0)
```

```
array([nan,  0.,  1.])
```

[6]:
```
# Checking the dataset for null values
Nan_values = FLIR_df.isnull().sum()
print (Nan_values)
```

```
SubjectID        0
Unnamed: 1    1020
T_offset1       17
Max1R13_1       16
Max1L13_1       16
              ...
Time             0
Date             0
Day              0
Month            0
Year             0
Length: 128, dtype: int64
```

[7]:
```
#Dropping of the unccessary columns
FLIR_df.drop(columns=['Unnamed: 1', 'Unnamed: 29', 'Unnamed: 57', 'Unnamed:␣
 ↪85', 'Unnamed: 113'], inplace=True)
```

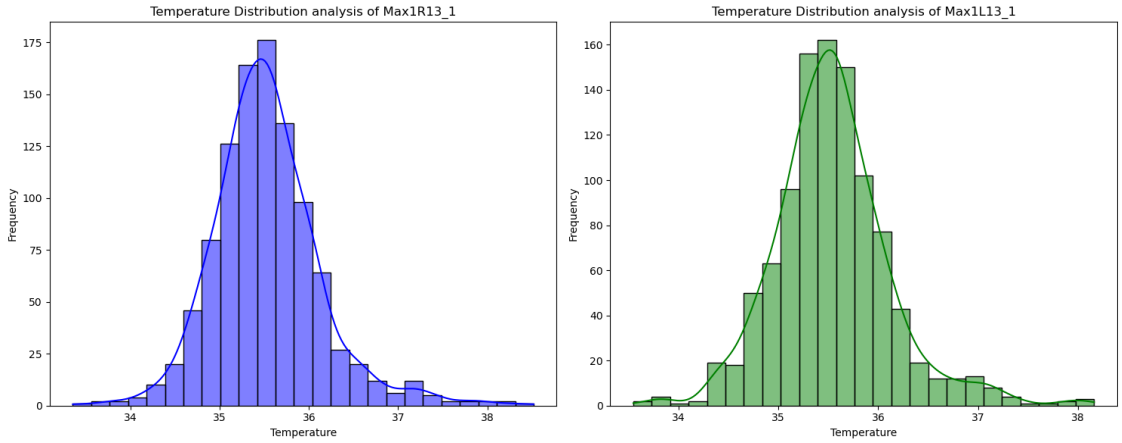
[8]:
```
#Taking only the numeric fields into consideration
column_means = FLIR_df.mean(numeric_only=True)

# Filling in all the NaN values with its corresponding column mean values
FLIR_df_filled = FLIR_df.fillna(column_means)

FLIR_df_no_duplicates = FLIR_df_filled.drop_duplicates()

# Displaying the Dataset
display(FLIR_df_no_duplicates.head())
```

```
  SubjectID  T_offset1  Max1R13_1  Max1L13_1  aveAllR13_1  aveAllL13_1  T_RC1  \
0  161117-1       0.58      34.98      35.36        34.44        34.85  34.91
1  161117-2       0.83      34.71      34.51        34.46        34.24  34.68
2  161117-3       0.85      35.70      35.44        35.00        34.78  35.67
3  161117-4       0.90      35.17      35.50        34.25        35.00  35.14
4  161117-5       1.08      35.33      35.55        34.31        35.14  35.50

   T_RC_Dry1  T_RC_Wet1  T_RC_Max1  ...                Ethnicity  T_atm  \
```

```
0      34.91      34.60      34.98  …                        White   24.0
1      34.68      34.44      34.71  …  Black or African-American   24.0
2      35.67      35.46      35.70  …                        White   24.0
3      35.14      35.08      35.17  …  Black or African-American   24.0
4      35.30      35.50      35.52  …                        White   24.0

   Humidity  Distance  Cosmetics      Time       Date  Day  Month  Year
0      28.0       0.8        0.0  12:43:46 2017-11-16   16     11  2017
1      26.0       0.8        0.0  15:22:48 2017-11-16   16     11  2017
2      26.0       0.8        0.0  15:52:56 2017-11-16   16     11  2017
3      27.0       0.8        0.0  16:07:53 2017-11-16   16     11  2017
4      27.0       0.8        0.0  16:28:06 2017-11-16   16     11  2017

[5 rows x 123 columns]
```

Task 4.1: Temperature Distribution Analysis

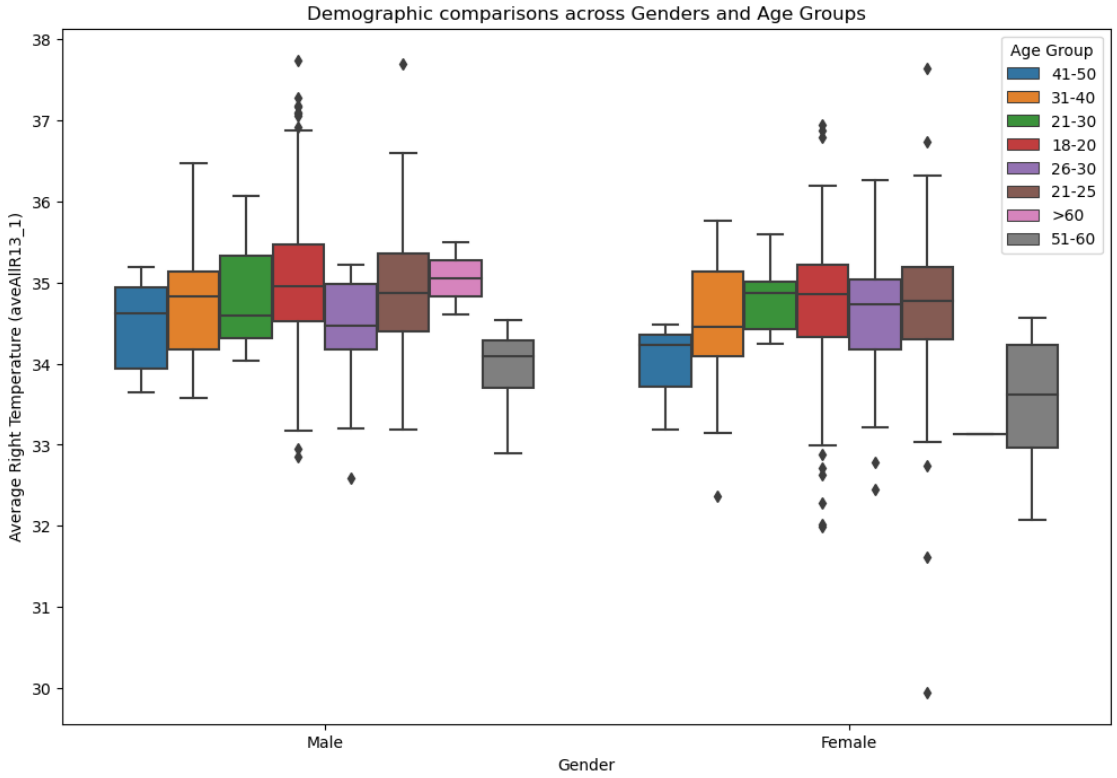
[27]:
```
#Plotting the temperature fields
plt.figure(figsize=(15, 6))

# Plotting the histogram for Max1R13_1
plt.subplot(1, 2, 1)
sns.histplot(FLIR_df_no_duplicates['Max1R13_1'], bins=25, color='blue',␣
 ↪kde=True)
plt.title('Temperature Distribution analysis of Max1R13_1')
plt.xlabel('Temperature')
plt.ylabel('Frequency')

# Plotting the histogram for Max1L13_1
plt.subplot(1, 2, 2)
sns.histplot(FLIR_df_no_duplicates['Max1L13_1'], bins=25, color='green',␣
 ↪kde=True)  # You can set kde=False if you don't want KDE
plt.title('Temperature Distribution analysis of Max1L13_1')
plt.xlabel('Temperature')
plt.ylabel('Frequency')

plt.tight_layout()
plt.show()
```



Insights: It can be found that the data is symmetrical distributed for both the temperature fields. Both of them attain their symmentrical peaks somewhere between 35.5 - 35.7 C and around 4-5% of the data are found to be outliers.

Task 4.2: Demographic Group Comparisions

[28]:
```
plt.figure(figsize=(12, 8))
sns.boxplot(x='Gender', y='aveAllR13_1', hue='Age', data=FLIR_df_no_duplicates)
plt.title('Demographic comparisons across Genders and Age Groups')
plt.xlabel('Gender')
plt.ylabel('Average Right Temperature (aveAllR13_1)')
plt.legend(title='Age Group')
plt.show()
```



Insights: A significant outliers can be found in the age catagory between 18-20 but the only difference being that the male catagory has most of its outliers above 36.6 C and for the female data ranges between 32 to 33.

Even for the 21-30 catagory, the asymetrically negatively skewed whereas it's the opposite skewness for the female data but the plot is the same for the 21-25 catagories for both the genders.

The average median for the male data is around 34.5 whereas to 34.2 to female data.

Task 4.3: Environmental Impact Analysis

[31]:
```
# Creating subplots
fig = make_subplots(rows=1, cols=2, subplot_titles=['Environmental analysis of␣
 ↪Average Left Temperature', 'Environmental analysis of Average Right␣
 ↪Temperature'])

# Adding the scatter plot for Average Left Temperature
scatter_left = px.scatter(FLIR_df_no_duplicates, x='aveAllL13_1', y='T_atm',␣
 ↪color='Humidity',
                          size='Humidity', title='Average Left Temperature vs␣
 ↪Atmospheric Temperature',
                          labels={'aveAllL13_1': 'Average Left Temperature␣
 ↪(°C)', 'T_atm': 'Atmospheric Temperature (°C)',
```

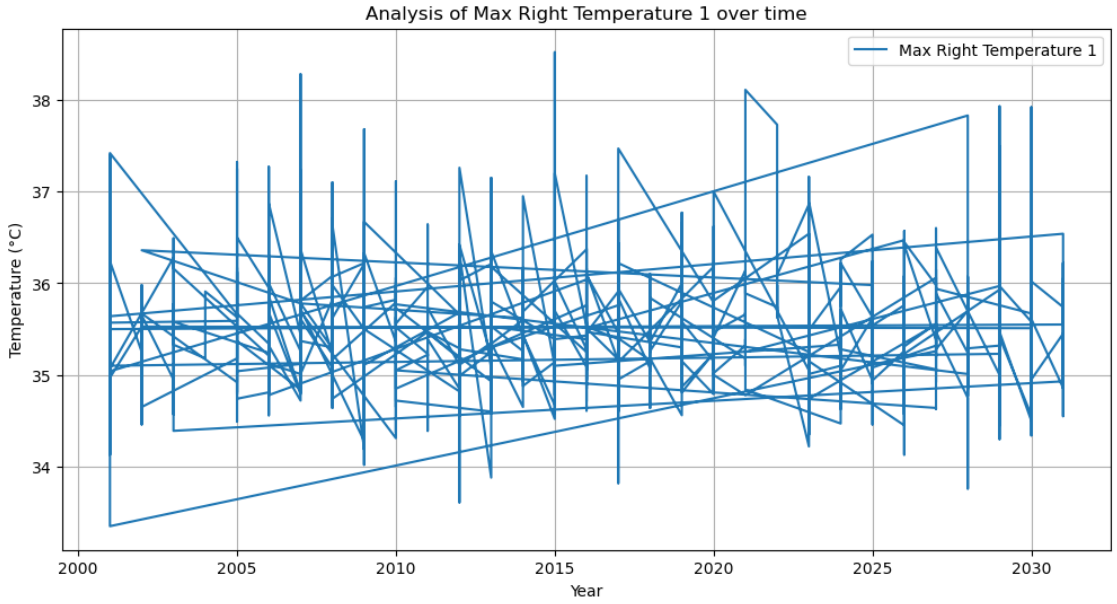
```
                                    'Humidity': 'Humidity (%)'},
                           size_max=20,
                           template='plotly')

# Adding the scatter plot for Average Right Temperature
scatter_right = px.scatter(FLIR_df_no_duplicates, x='aveAllR13_1', y='T_atm',␣
 ↪color='Humidity',
                            size='Humidity', title='Average Right Temperature vs␣
 ↪Atmospheric Temperature',
                            labels={'aveAllR13_1': 'Average Right Temperature␣
 ↪(°C)', 'T_atm': 'Atmospheric Temperature (°C)',
                                    'Humidity': 'Humidity (%)'},
                            size_max=20,
                            template='plotly')

# Updating as subplot interms of rows and columns
fig.add_trace(scatter_left['data'][0], row=1, col=1)
fig.add_trace(scatter_right['data'][0], row=1, col=2)

#setting up the x and y axis labels
fig.update_xaxes(title_text='Average Left Temperature (°C)', row=1, col=1)
fig.update_yaxes(title_text='Atmospheric Temperature (°C)', row=1, col=1)

fig.update_xaxes(title_text='Average Right Temperature (°C)', row=1, col=2)
fig.update_yaxes(title_text='Atmospheric Temperature (°C)', row=1, col=2)


# Update layout
fig.update_layout(width=1000, height=500, showlegend=False)

# Show the interactive plot
fig.show()
```

Insights The most common relation between the 2 sets of graphs is that the humidity is found to be very less for certain atmospheric temperature which averages around the 23.7 - 24.2 C. It is also found that the presence of high humidity makes the atmospheric temperature to drop, which is inversely proportional.

Task 4.4: Time Series Analysis

[32]:
```
# Create a time series plot for 'Max1R13_1'
plt.figure(figsize=(12, 6))
plt.plot(FLIR_df_no_duplicates['Year'], FLIR_df_no_duplicates['Max1R13_1'],␣
 ↪label='Max Right Temperature 1')
plt.title('Analysis of Max Right Temperature 1 over time')
plt.xlabel('Year')
plt.ylabel('Temperature (°C)')
```

```
plt.legend()
plt.grid(True)
plt.show()
```

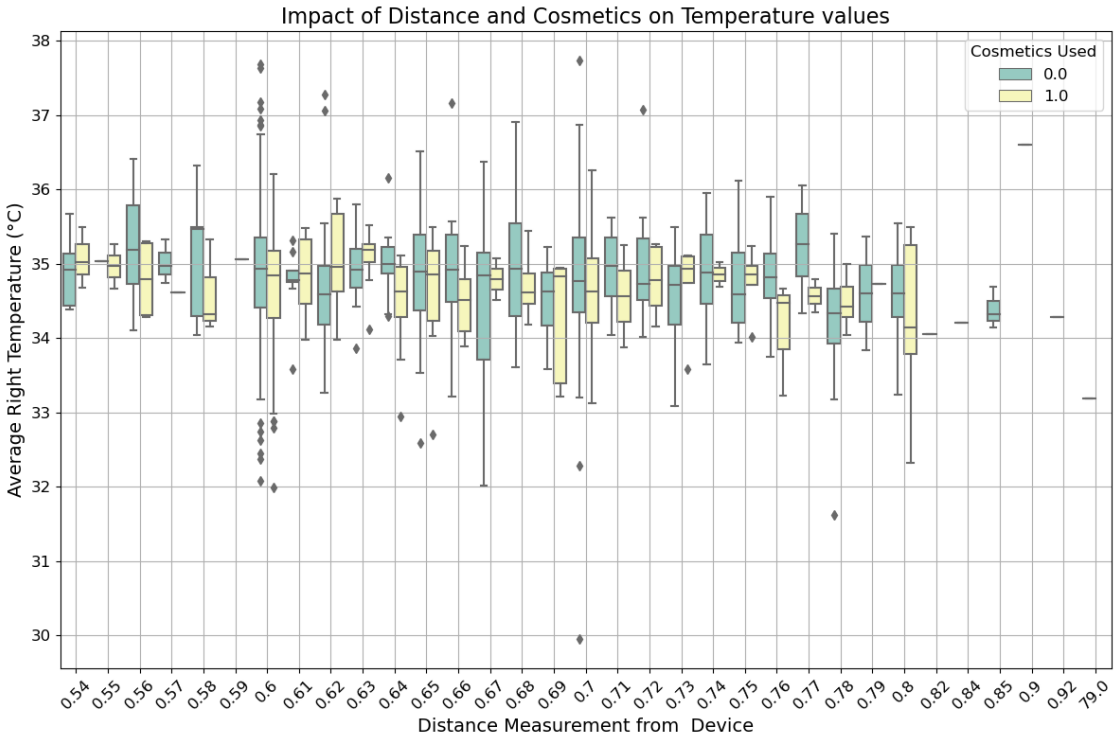


Insights: It can be predicted that the average lowest temperature will be higher in 2030 when compared to 2000 and can also be found out that the it has been and will increase in decimal points each year.

Task 4.5: Impact of Distance and Cosmetics

[33]:
```
FLIR_df_no_duplicates['Distance'] = FLIR_df_no_duplicates['Distance'].round(2)

# Create a box plot using Seaborn
plt.figure(figsize=(12, 8))
sns.boxplot(x='Distance', y='aveAllR13_1', hue='Cosmetics',␣
 ↪data=FLIR_df_no_duplicates,
            palette='Set3')

# Customize the plot
plt.title('Impact of Distance and Cosmetics on Temperature values', fontsize=16)
plt.xlabel('Distance Measurement from  Device', fontsize=14)
plt.ylabel('Average Right Temperature (°C)', fontsize=14)
plt.xticks(fontsize=12, rotation=45)
plt.yticks(fontsize=12)
plt.legend(title='Cosmetics Used', fontsize=12, title_fontsize=12)
plt.grid(True)
```

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

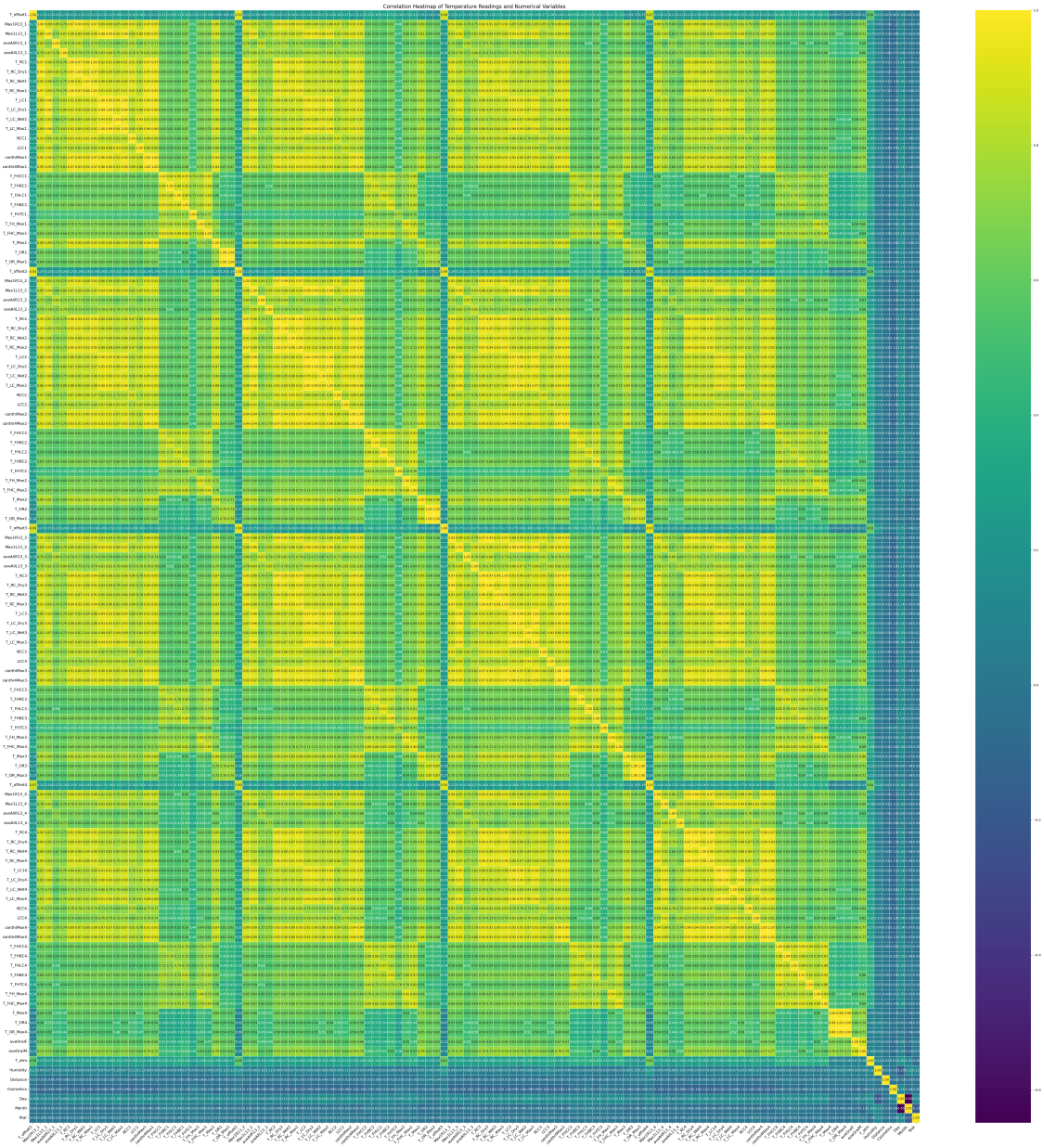


Insights: It can be found that the 0 type cosmetics used tend to have more interquartile range, more range of values, than the other type. Eventhough the median values for the distance measurement of 0.59 and 0.60 tend to be the almost same but the former has a lot more outlier values which needs to be taken into consideration.

Task 4.6: Correlation heatmap

[17]:
```
corr_matrix = FLIR_df_no_duplicates.corr()

# Create correlation heatmap
plt.figure(figsize=(50, 50))
sns.heatmap(corr_matrix, annot=True, cmap='viridis', fmt=".2f", linewidths=0.5,␣
 ↪cbar=True)
plt.title('Correlation Heatmap of Temperature Readings and Numerical␣
 ↪Variables', fontsize=16)
plt.xticks(rotation=45, ha='right', fontsize=12)
plt.yticks(rotation=0, fontsize=12)
plt.tight_layout()
plt.show()
```

```
/var/folders/0c/6l7fjjk134q20fw6s25nlr4h0000gn/T/ipykernel_92401/704306879.py:1:
FutureWarning:

The default value of numeric_only in DataFrame.corr is deprecated. In a future
version, it will default to False. Select only valid columns or specify the
value of numeric_only to silence this warning.
```

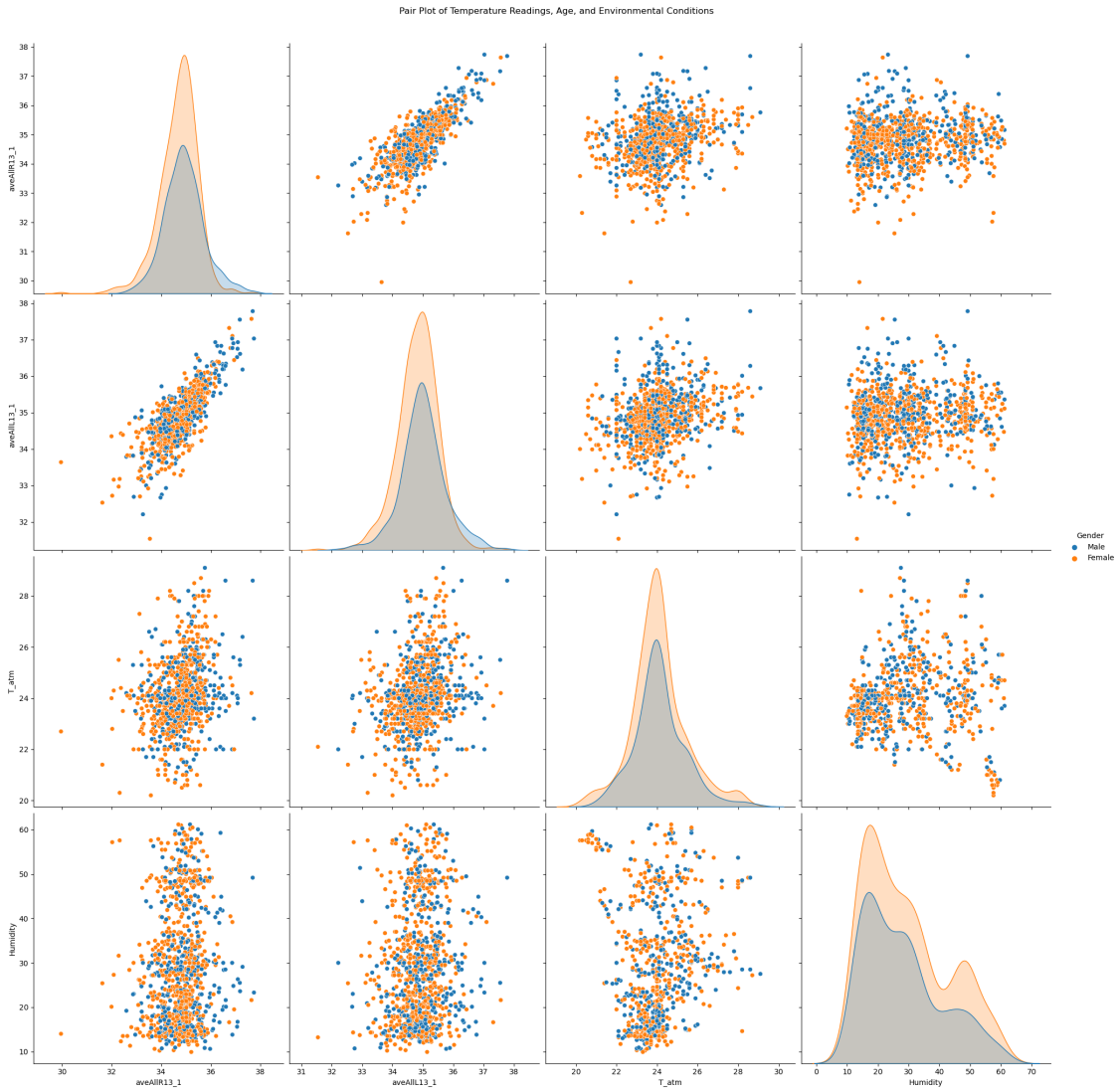


Insights: It can be found that the highest correlation apart from itself is between the fields T_RC_MAX1 & MAX1R13_1, which are positively correlation almost equal to 1. The T_offset_4

field has a constant correlation with almost all the fields in picture.

Part 4.7: Multi-variable Analysis

[20]:
```
# Select columns for pair plot
selected_attributes = ['aveAllR13_1', 'aveAllL13_1', 'Gender', 'Age', 'T_atm',␣
 ↪'Humidity']

# Subset the DataFrame and remove rows with missing values
subset_FLIR_df = FLIR_df_no_duplicates[selected_attributes].dropna()

# Create a pair plot with color-coded points based on 'Gender'
sns.pairplot(subset_FLIR_df, hue='Gender', height=5)
plt.suptitle('Pair Plot of Temperature Readings, Age, and Environmental␣
 ↪Conditions', y=1.02)

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



Insights: Lets look at the findings on a higher level.

Diagonal Plots/Histograms: :

The diagonal of the pair plot usually contains histograms which show the univariate distribution of each variable.

Scatter Plots:

The main area of the plot consists of scatter plots for various combinations of the variables in picture. Most of the plots has very minimal outliers

Part 4.8: Data Preprocessing observations During the preprocessing stage, there were a lot of missing/null values in the dataset which was handled by filling in with the mean of the corresponding columns as there was a concious effort made of not to lose data, as each and every

record could give us a valuable insight.

There was a few columns dropped, 'Unnamed: 1', 'Unnamed: 29' to name a few, and also dropping of the duplicates.

There was also this important step which was to be carried out was the field transformation on the "date" field.

Part 4.9: Final Insights and Conclusions: It can be predicted that the average lowest temperature will be higher in 2030 when compared to 2000 and can also be found out that the it has been and will increase in decimal points each year.

The most common relation between the 2 sets of graphs is that the humidity is found to be very less for certain atmospheric temperature which averages around the 23.7 - 24.2 C. It is also found that the presence of high humidity makes the atmospheric temperature to drop, which is inversely proportional.

A significant outliers can be found in the age catagory between 18-20 but the only difference being that the male catagory has most of its outliers above 36.6 C and for the female data ranges between 32 to 33.

Even for the 21-30 catagory, the asymetrically negatively skewed whereas it's the opposite skewness for the female data but the plot is the same for the 21-25 catagories for both the genders.

The average median for the male data is around 34.5 whereas to 34.2 to female data.

Question 5: PCA on University rankings

[34]:
```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer
```

[35]:
```
file_path = '/Users/jaamiemaarshj/Desktop/  DAE Course Materials/Data Mining/
  ↪Assignment -1/Universities.csv'

university_df = pd.read_csv(file_path)
display(university_df.head())
```

```
                      College Name State  Public (1)/ Private (2)  \
0          Alaska Pacific University    AK                        2
1  University of Alaska at Fairbanks    AK                        1
2     University of Alaska Southeast    AK                        1
3  University of Alaska at Anchorage    AK                        1
4        Alabama Agri. & Mech. Univ.    AL                        1

   # appli. rec'd  # appl. accepted  # new stud. enrolled  \
0           193.0             146.0                  55.0
1          1852.0            1427.0                 928.0
2           146.0             117.0                  89.0
3          2065.0            1598.0                1162.0
```

```
4          2817.0            1920.0                 984.0

   % new stud. from top 10%  % new stud. from top 25%  # FT undergrad  \
0                      16.0                      44.0           249.0
1                       NaN                       NaN          3885.0
2                       4.0                      24.0           492.0
3                       NaN                       NaN          6209.0
4                       NaN                       NaN          3958.0

   # PT undergrad  in-state tuition  out-of-state tuition    room   board  \
0           869.0            7560.0                7560.0  1620.0  2500.0
1          4519.0            1742.0                5226.0  1800.0  1790.0
2          1849.0            1742.0                5226.0  2514.0  2250.0
3         10537.0            1742.0                5226.0  2600.0  2520.0
4           305.0            1700.0                3400.0  1108.0  1442.0

   add. fees  estim. book costs  estim. personal $  % fac. w/PHD  \
0      130.0              800.0             1500.0          76.0
1      155.0              650.0             2304.0          67.0
2       34.0              500.0             1162.0          39.0
3      114.0              580.0             1260.0          48.0
4      155.0              500.0              850.0          53.0

   stud./fac. ratio  Graduation rate
0              11.9             15.0
1              10.0              NaN
2               9.5             39.0
3              13.7              NaN
4              14.3             40.0
```

[36]:
```
university_df_bkp = university_df.copy()
Nan_values = university_df.isnull().sum()
print (Nan_values)
```

```
College Name                 0
State                        0
Public (1)/ Private (2)      0
# appli. rec'd              10
# appl. accepted            11
# new stud. enrolled         5
% new stud. from top 10%    235
% new stud. from top 25%    202
# FT undergrad               3
# PT undergrad              32
in-state tuition            30
out-of-state tuition        20
room                       321
board                      498
```

```
add. fees                   274
estim. book costs            48
estim. personal $           181
% fac. w/PHD                 32
stud./fac. ratio              2
Graduation rate              98
dtype: int64
```

[37]:
```
#dropping all the catagorical columns
columns_dropped = ['College Name', 'State','Public (1)/ Private (2)']
# Use the drop() method to remove the specified columns
university_df.drop(columns=columns_dropped, inplace=True)
display(columns_dropped)
```

```
['College Name', 'State', 'Public (1)/ Private (2)']
```

[38]:
```
updated_university_df = university_df.dropna()
Nan_values_upd = updated_university_df.isnull().sum()
print (Nan_values_upd)
```

```
# appli. rec'd              0
# appl. accepted            0
# new stud. enrolled        0
% new stud. from top 10%    0
% new stud. from top 25%    0
# FT undergrad              0
# PT undergrad              0
in-state tuition            0
out-of-state tuition        0
room                        0
board                       0
add. fees                   0
estim. book costs           0
estim. personal $           0
% fac. w/PHD                0
stud./fac. ratio            0
Graduation rate             0
dtype: int64
```

[39]:
```
column_name_mapping = {
    '# appli. rec\'d': 'Application_recieved_count',
    '# appl. accepted': 'Accepted_application_count',
    '# new stud. enrolled': 'New_enrolled_student_count',
    '% new stud. from top 10%': 'Percent_of_New_Student_top10',
    '% new stud. from top 25%': 'Percent_of_New_Student_top25',
    '# FT undergrad': 'FT_undergrad_count',
    '# PT undergrad': 'PT_undergrad_count',
    'in-state tuition': 'In_state_tuition',
```

```
    'out-of-state tuition': 'Out_state_tuition',
    'room': 'room',
    'board': 'board',
    'add. fees': 'additional_fees',
    'estim. book costs': 'Estimated_book_cost',
    'estim. personal $': 'Estimate_personal_cost',
    '% fac. w/PHD' : 'Perct_PHD_faculty',
    'stud./fac. ratio': 'Student_faculty_ratio',
    'Graduation rate': 'Graduation_rate'

}

# Use the rename method to apply the name changes
updated_university_df.rename(columns=column_name_mapping, inplace=True)

# Display the modified DataFrame
display(updated_university_df.head())
```

```
/var/folders/0c/6l7fjjk134q20fw6s25nlr4h0000gn/T/ipykernel_92401/2981456893.py:2
3: SettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

	Application_recieved_count	Accepted_application_count	\
0	193.0	146.0	
2	146.0	117.0	
9	805.0	588.0	
11	608.0	520.0	
21	4414.0	1500.0	

	New_enrolled_student_count	Percent_of_New_Student_top10	\
0	55.0	16.0	
2	89.0	4.0	
9	287.0	67.0	
11	127.0	26.0	
21	335.0	30.0	

	Percent_of_New_Student_top25	FT_undergrad_count	PT_undergrad_count	\
0	44.0	249.0	869.0	
2	24.0	492.0	1849.0	
9	88.0	1376.0	207.0	
11	47.0	538.0	126.0	
21	60.0	908.0	119.0	

```
    In_state_tuition  Out_state_tuition    room   board  additional_fees  \
0             7560.0             7560.0  1620.0  2500.0            130.0
2             1742.0             5226.0  2514.0  2250.0             34.0
9            11660.0            11660.0  2050.0  2430.0            120.0
11            8080.0             8080.0  1380.0  2540.0            100.0
21            5666.0             5666.0  1424.0  1540.0            418.0

    Estimated_book_cost  Estimate_personal_cost  Perct_PHD_faculty  \
0                 800.0                  1500.0               76.0
2                 500.0                  1162.0               39.0
9                 400.0                   900.0               74.0
11                500.0                  1100.0               63.0
21               1000.0                  1400.0               56.0

    Student_faculty_ratio  Graduation_rate
0                    11.9             15.0
2                     9.5             39.0
9                    14.0             72.0
11                   11.4             44.0
21                   15.5             46.0
```

0.0.14 Normalized PCA process

[40]:
```
column_arrays = {}

# Iterate over columns and convert each to a NumPy array
for column in updated_university_df.columns:
    column_arrays[column] = np.array(updated_university_df[column])

# Now, you can access the NumPy arrays using the dictionary keys
X1 = column_arrays['Application_recieved_count']
X2 = column_arrays['Accepted_application_count']
X3 = column_arrays['New_enrolled_student_count']
X4 = column_arrays['Percent_of_New_Student_top10']
X5 = column_arrays['Percent_of_New_Student_top25']
X6 = column_arrays['FT_undergrad_count']
X7 = column_arrays['PT_undergrad_count']
X8 = column_arrays['In_state_tuition']
X9 = column_arrays['Out_state_tuition']
X10 = column_arrays['room']
X11 = column_arrays['board']
X12 = column_arrays['additional_fees']
X13 = column_arrays['Estimated_book_cost']
X14 = column_arrays['Estimate_personal_cost']
X15 = column_arrays['Perct_PHD_faculty']
X16 = column_arrays['Student_faculty_ratio']
```

```
X17 = column_arrays['Graduation_rate']
```

[41]:
```
#Standardisation of the columns

X1_standardized =(X1 - np.mean(X1))/np.std(X1)
X2_standardized =(X2 - np.mean(X2))/np.std(X2)
X3_standardized =(X3 - np.mean(X3))/np.std(X3)
X4_standardized =(X4 - np.mean(X4))/np.std(X4)
X5_standardized =(X5 - np.mean(X5))/np.std(X5)
X6_standardized =(X6 - np.mean(X6))/np.std(X6)
X7_standardized =(X7 - np.mean(X7))/np.std(X7)
X8_standardized =(X8 - np.mean(X8))/np.std(X8)
X9_standardized =(X9- np.mean(X9))/np.std(X9)
X10_standardized =(X10 - np.mean(X10))/np.std(X10)
X11_standardized =(X11 - np.mean(X11))/np.std(X11)
X12_standardized =(X12 - np.mean(X12))/np.std(X12)
X13_standardized =(X13 - np.mean(X13))/np.std(X13)
X14_standardized =(X14 - np.mean(X9))/np.std(X9)
X15_standardized =(X15 - np.mean(X10))/np.std(X10)
X16_standardized =(X16 - np.mean(X11))/np.std(X11)
X17_standardized =(X17 - np.mean(X12))/np.std(X12)



X_standardized = np.
 ↪stack((X1_standardized,X2_standardized,X3_standardized,X4_standardized,X5_standardized,X6_s
```

[42]:
```
#Step 2: computing Covariance matrix
cov_matrix = np.cov(X_standardized )
#Step 3:
eigenvalues, eigenvectors = np.linalg.eig(cov_matrix)
#Step 4: Sort Eigen Values and Eigen Vectors

idx = np.argsort(eigenvalues)[::-1]
eigenvalues = eigenvalues[idx]
eigenvectors = eigenvectors[: , idx]

#Step 5: Select top k eigenvalues

k = 10
selected_eigenvectors = eigenvectors[:, :k]
#Step 6: transform the original matrix
PCA_result = np.dot(selected_eigenvectors.T, X_standardized)
```

[43]:
```
print("Standardised data: \n", X_standardized)
print("Covariance Matrix: \n", cov_matrix)
print("Eigenvalues:\n", eigenvalues)
print("eigenvectors:\n", eigenvectors)
```

```
print("Selected Eigenvectors (PC):\n", selected_eigenvectors)
print("PCA result:\n", PCA_result )
```

```
Standardised data:
 [[-0.72608508 -0.73763636 -0.57567262 … -0.63047977  0.26511336
  -0.2748477 ]
 [-0.76644694 -0.77804187 -0.58972424 … -0.6257085   0.64093751
  -0.21868652]
 [-0.79341424 -0.75624199 -0.53976829 … -0.66549796  1.3407103
   0.31956669]
 …
 [-3.01097154 -3.06290642 -3.01377883 … -3.06150277 -3.00956789
  -2.98991685]
 [-3.72628201 -3.73052035 -3.72257346 … -3.7269884  -3.71798192
  -3.72063089]
 [-1.02403514 -0.95652027 -0.86368734 … -0.91994972 -0.92557596
  -0.93964156]]
Covariance Matrix:
 [[ 1.00212766e+00  9.37761030e-01  8.21601760e-01  2.99384546e-01
    3.26083679e-01  7.97335831e-01  4.13135697e-01 -1.19164537e-01
    7.59352925e-02  2.32861833e-01  1.26162452e-01  3.00407515e-01
    1.05613306e-01  2.82108664e-02  8.83221657e-03  5.36558942e-04
    9.30386340e-03]
 [ 9.37761030e-01  1.00212766e+00  8.97060020e-01  1.36324294e-01
    2.01644283e-01  8.66587974e-01  4.71371854e-01 -2.02759131e-01
   -5.67969385e-03  1.71097377e-01  5.38830650e-02  2.93178903e-01
    7.11544469e-02  3.25931001e-02  8.04291921e-03  1.13064006e-03
    5.02602573e-03]
 [ 8.21601760e-01  8.97060020e-01  1.00212766e+00  1.38756055e-01
    1.79844068e-01  9.70704414e-01  5.59354184e-01 -3.36346010e-01
   -1.39992015e-01  1.97765681e-02 -3.42954587e-02  2.84207621e-01
    6.15414412e-02  4.68824308e-02  7.49513482e-03  1.55710048e-03
    6.50919998e-04]
 [ 2.99384546e-01  1.36324294e-01  1.38756055e-01  1.00212766e+00
    9.08056815e-01  1.02779847e-01 -1.26636223e-01  5.54322019e-01
    6.27323102e-01  3.44308446e-01  3.76594474e-01  2.64483461e-02
    1.16292877e-01 -2.06743463e-02  1.27765998e-02 -3.17236236e-03
    2.85253573e-02]
 [ 3.26083679e-01  2.01644283e-01  1.79844068e-01  9.08056815e-01
    1.00212766e+00  1.55900187e-01 -9.76808155e-02  5.00073750e-01
    5.77808580e-01  3.24778080e-01  3.60673759e-01  1.59783029e-02
    1.25255033e-01 -1.95991731e-02  1.31943547e-02 -2.55552233e-03
    2.94039761e-02]
 [ 7.97335831e-01  8.66587974e-01  9.70704414e-01  1.02779847e-01
    1.55900187e-01  1.00212766e+00  5.94037215e-01 -3.87669625e-01
   -1.85120346e-01  8.59476253e-03 -5.51601135e-02  2.49860181e-01
    7.05295139e-02  5.34298463e-02  7.33182406e-03  1.76524270e-03
```

```
 -2.15702477e-03]
[ 4.13135697e-01  4.71371854e-01  5.59354184e-01 -1.26636223e-01
 -9.76808155e-02  5.94037215e-01  1.00212766e+00 -3.41954065e-01
 -2.21306183e-01 -2.51541135e-02 -1.56625380e-02  8.96880637e-02
  8.05725260e-02  4.62588293e-02  3.23286654e-03  1.34086412e-03
 -1.20230104e-02]
[-1.19164537e-01 -2.02759131e-01 -3.36346010e-01  5.54322019e-01
  5.00073750e-01 -3.87669625e-01 -3.41954065e-01  1.00212766e+00
  9.39376618e-01  4.91434757e-01  5.83649517e-01 -2.19671487e-01
  5.53353355e-02 -5.53395912e-02  6.75953086e-03 -4.16467750e-03
  2.96236290e-02]
[ 7.59352925e-02 -5.67969385e-03 -1.39992015e-01  6.27323102e-01
  5.77808580e-01 -1.85120346e-01 -2.21306183e-01  9.39376618e-01
  1.00212766e+00  5.58927656e-01  6.07928662e-01 -9.79317432e-02
  6.75474400e-02 -5.07168239e-02  1.07322612e-02 -4.10814394e-03
  3.17523572e-02]
[ 2.32861833e-01  1.71097377e-01  1.97765681e-02  3.44308446e-01
  3.24778080e-01  8.59476253e-03 -2.51541135e-02  4.91434757e-01
  5.58927656e-01  1.00212766e+00  4.83577541e-01  9.21641413e-02
  1.93317657e-01 -2.81819992e-02  7.52956864e-03 -2.38054295e-03
  1.88861025e-02]
[ 1.26162452e-01  5.38830650e-02 -3.42954587e-02  3.76594474e-01
  3.60673759e-01 -5.51601135e-02 -1.56625380e-02  5.83649517e-01
  6.07928662e-01  4.83577541e-01  1.00212766e+00 -1.39253732e-02
  1.39200812e-01 -2.63779047e-02  7.53947410e-03 -2.36718987e-03
  2.10110644e-02]
[ 3.00407515e-01  2.93178903e-01  2.84207621e-01  2.64483461e-02
  1.59783029e-02  2.49860181e-01  8.96880637e-02 -2.19671487e-01
 -9.79317432e-02  9.21641413e-02 -1.39253732e-02  1.00212766e+00
  1.80968329e-02  3.06271624e-03  3.94891549e-03  5.29638374e-04
  2.36010628e-03]
[ 1.05613306e-01  7.11544469e-02  6.15414412e-02  1.16292877e-01
  1.25255033e-01  7.05295139e-02  8.05725260e-02  5.53353355e-02
  6.75474400e-02  1.93317657e-01  1.39200812e-01  1.80968329e-02
  1.00212766e+00  2.41062826e-02  1.47356548e-04 -5.60711563e-04
  2.56945010e-03]
[ 2.82108664e-02  3.25931001e-02  4.68824308e-02 -2.06743463e-02
 -1.95991731e-02  5.34298463e-02  4.62588293e-02 -5.53395912e-02
 -5.07168239e-02 -2.81819992e-02 -2.63779047e-02  3.06271624e-03
  2.41062826e-02  2.50614260e-02 -2.73863517e-04  1.28257476e-04
 -1.93522708e-03]
[ 8.83221657e-03  8.04291921e-03  7.49513482e-03  1.27765998e-02
  1.31943547e-02  7.33182406e-03  3.23286654e-03  6.75953086e-03
  1.07322612e-02  7.52956864e-03  7.53947410e-03  3.94891549e-03
  1.47356548e-04 -2.73863517e-04  5.47216661e-04 -3.42957850e-05
  4.93349770e-04]
[ 5.36558942e-04  1.13064006e-03  1.55710048e-03 -3.17236236e-03
 -2.55552233e-03  1.76524270e-03  1.34086412e-03 -4.16467750e-03
```

```
  -4.10814394e-03 -2.38054295e-03 -2.36718987e-03  5.29638374e-04
  -5.60711563e-04  1.28257476e-04 -3.42957850e-05  4.74071624e-05
  -1.12122363e-04]
 [ 9.30386340e-03  5.02602573e-03  6.50919998e-04  2.85253573e-02
   2.94039761e-02 -2.15702477e-03 -1.20230104e-02  2.96236290e-02
   3.17523572e-02  1.88861025e-02  2.10110644e-02  2.36010628e-03
   2.56945010e-03 -1.93522708e-03  4.93349770e-04 -1.12122363e-04
   2.60604417e-03]]
Eigenvalues:
 [4.30782999e+00 3.91592806e+00 1.09440339e+00 9.65702723e-01
 9.34718337e-01 5.80040574e-01 4.72855854e-01 3.72276314e-01
 1.97546626e-01 9.72900530e-02 4.18921917e-02 3.35267315e-02
 2.19895207e-02 1.83200113e-02 1.30823147e-03 2.66454641e-04
 2.66010835e-05]
eigenvectors:
 [[ 4.31088664e-01  1.30587137e-01 -3.25080434e-02  3.96110240e-02
   3.98382596e-02 -2.46980764e-01 -1.21982656e-01 -7.19993584e-03
   6.13176774e-01 -2.04451866e-01  2.77903484e-01 -3.90199721e-01
  -2.58187105e-01 -5.38465303e-02  8.80140924e-03 -7.45287524e-03
  -1.85211590e-04]
 [ 4.50634087e-01  6.67282612e-02  7.66866244e-03  1.45151830e-02
   1.16061704e-01 -2.83758515e-01 -1.51673952e-01 -1.12487986e-01
   2.48525845e-01  2.32973911e-01 -3.51849294e-01  4.93804957e-01
   4.12735731e-01  9.18311136e-02 -6.03344474e-03  7.45827442e-03
  -2.12402823e-04]
 [ 4.63526530e-01  5.29653632e-03 -7.01668352e-02 -7.10486297e-02
   3.42864842e-02 -5.40264875e-02 -1.02977777e-01 -3.99628558e-02
  -4.69411145e-01 -6.49229303e-02 -3.46901529e-01  5.05716273e-02
  -5.69884019e-01 -2.93618609e-01  1.94725299e-02 -5.38936895e-03
   2.36377976e-04]
 [ 6.04315829e-02  4.13873601e-01 -3.76761544e-01 -4.81187788e-02
  -2.80249669e-01  1.89813323e-01  1.55945199e-01  1.40062093e-01
   2.60865845e-02 -6.56562071e-01 -1.49585433e-01  1.95478737e-01
   1.58651960e-01  6.85487416e-02 -6.70071592e-03  4.86323766e-03
   2.60608571e-03]
 [ 8.55713891e-02  4.00132794e-01 -3.86636684e-01 -7.97969229e-02
  -2.97117384e-01  1.71853934e-01  1.38917933e-01  2.23719366e-01
   6.02970230e-02  6.80994288e-01  5.35049459e-02 -9.46420739e-02
  -8.42057381e-02 -5.33191070e-02  1.96577065e-02  4.99223516e-03
  -1.00737329e-03]
 [ 4.61203573e-01 -1.66852197e-02 -4.94004266e-02 -1.13900592e-01
   2.65930481e-02 -2.75094630e-02 -5.21919180e-02  3.30124021e-02
  -5.12798980e-01 -2.24772237e-02  3.90645243e-01 -2.95429131e-01
   4.05294688e-01  3.13754965e-01 -1.49766324e-02  6.25081281e-03
  -1.38327194e-03]
 [ 3.05924842e-01 -9.25300473e-02  2.73677490e-01 -3.17710352e-01
   1.93022367e-01  6.38315492e-01  4.40334439e-01 -2.00833904e-01
   2.01175580e-01  2.05770539e-02 -4.36784004e-02  1.02994146e-02
```

```
 -5.64221098e-03  8.35273906e-03 -7.72940987e-03  2.36963503e-03
  3.09959877e-04]
[-1.86652893e-01  4.27954279e-01  3.03203905e-02 -6.15131237e-02
  1.69107381e-01 -5.24817669e-02 -6.33634438e-02 -4.64820371e-01
 -5.58224383e-02  2.82236104e-02 -4.61405480e-01 -5.26410039e-01
  1.30235485e-01  1.21041887e-01  6.45468087e-03 -1.78153443e-02
  1.16846044e-03]
[-8.54164250e-02  4.58236532e-01  2.60654799e-02 -1.03061341e-02
  1.83381052e-01 -3.64196803e-02 -2.48700153e-02 -5.03039950e-01
 -1.22934806e-01  1.16825082e-02  5.31588071e-01  3.98474419e-01
 -1.29080655e-01 -1.39382738e-01  1.40466612e-02  2.01234848e-02
  2.48267325e-03]
[ 3.33019461e-02  3.33282249e-01  4.21341845e-01  2.17762210e-01
  1.70512313e-01 -3.54400568e-01  5.99159146e-01  3.65374262e-01
 -1.18112026e-01 -5.33957813e-03 -3.32347776e-02 -1.24646188e-02
 -1.30705219e-02 -1.70119570e-02  7.14155660e-04  7.02077040e-04
  4.69055055e-04]
[-1.64612649e-02  3.52222072e-01  3.42959185e-01  9.54883053e-03
  2.74055416e-01  3.74008458e-01 -5.75509771e-01  4.58691652e-01
  1.64403271e-03 -1.13373049e-02 -7.54381399e-03  3.32590134e-02
  7.27827844e-03 -7.54254651e-04  2.81511355e-03  1.54992164e-03
 -2.27794276e-04]
[ 1.79294610e-01 -1.58513692e-02  4.10725576e-02  8.87452774e-01
 -1.53933982e-01  3.28197248e-01  6.09858455e-04 -2.01785392e-01
 -3.52051684e-02  3.26662719e-02 -1.96337150e-02 -4.91426131e-02
  3.16995900e-02  9.54652456e-03  3.27784292e-03  1.47625265e-03
  2.87180922e-04]
[ 5.23037028e-02  9.46688616e-02  5.71471658e-01 -1.75867583e-01
 -7.67032323e-01 -6.60575689e-02 -1.28033114e-01 -1.43912898e-01
 -1.11402161e-02  6.41611230e-04  2.20009453e-04  1.74360849e-02
 -1.16928442e-02  2.04402398e-02 -6.95447717e-04 -1.91845845e-03
  8.43661509e-05]
[ 2.34460825e-02 -2.02710155e-02  1.06847832e-02 -2.54691039e-02
 -2.32754652e-02  1.13659441e-02 -3.55323241e-03  1.21305615e-02
 -2.42841303e-02 -3.95673799e-02  3.91430530e-02 -1.62493714e-01
  4.53467566e-01 -8.72634260e-01 -1.40126313e-02 -2.36757219e-03
  5.36583554e-03]
[ 3.67635364e-03  6.33830658e-03 -3.34665094e-03  1.11622802e-03
  6.49610831e-04  4.09550012e-03  2.66002362e-03 -2.39120534e-04
 -4.01923389e-03  4.31170449e-03  1.76996837e-02  2.26828174e-02
  3.36676914e-03  1.63104345e-03 -5.25339709e-02 -9.98092707e-01
 -9.17258341e-03]
[ 7.96931455e-04 -1.96468141e-03  3.03758868e-07 -2.18869781e-05
 -7.03505870e-06 -4.00736022e-04 -7.92085865e-04  1.61832325e-03
  3.12226513e-05  2.74800469e-03  2.12648392e-04 -2.77568498e-04
 -2.40254075e-03  5.15716602e-03 -1.30985546e-02 -8.50351901e-03
  9.99854075e-01]
[-3.01987288e-04  1.72346365e-02 -6.57366055e-03  4.77426358e-03
```

```
  -6.54806113e-04 -3.23464444e-03 -5.05046058e-03 -4.91099615e-03
   1.69165482e-04  1.42363486e-02 -2.75319074e-03 -1.05676721e-03
  -3.20270925e-02 -2.12047456e-03 -9.97699958e-01  5.26161426e-02
  -1.26908483e-02]]
Selected Eigenvectors (PC):
 [[ 4.31088664e-01  1.30587137e-01 -3.25080434e-02  3.96110240e-02
   3.98382596e-02 -2.46980764e-01 -1.21982656e-01 -7.19993584e-03
   6.13176774e-01 -2.04451866e-01]
 [ 4.50634087e-01  6.67282612e-02  7.66866244e-03  1.45151830e-02
   1.16061704e-01 -2.83758515e-01 -1.51673952e-01 -1.12487986e-01
   2.48525845e-01  2.32973911e-01]
 [ 4.63526530e-01  5.29653632e-03 -7.01668352e-02 -7.10486297e-02
   3.42864842e-02 -5.40264875e-02 -1.02977777e-01 -3.99628558e-02
  -4.69411145e-01 -6.49229303e-02]
 [ 6.04315829e-02  4.13873601e-01 -3.76761544e-01 -4.81187788e-02
  -2.80249669e-01  1.89813323e-01  1.55945199e-01  1.40062093e-01
   2.60865845e-02 -6.56562071e-01]
 [ 8.55713891e-02  4.00132794e-01 -3.86636684e-01 -7.97969229e-02
  -2.97117384e-01  1.71853934e-01  1.38917933e-01  2.23719366e-01
   6.02970230e-02  6.80994288e-01]
 [ 4.61203573e-01 -1.66852197e-02 -4.94004266e-02 -1.13900592e-01
   2.65930481e-02 -2.75094630e-02 -5.21919180e-02  3.30124021e-02
  -5.12798980e-01 -2.24772237e-02]
 [ 3.05924842e-01 -9.25300473e-02  2.73677490e-01 -3.17710352e-01
   1.93022367e-01  6.38315492e-01  4.40334439e-01 -2.00833904e-01
   2.01175580e-01  2.05770539e-02]
 [-1.86652893e-01  4.27954279e-01  3.03203905e-02 -6.15131237e-02
   1.69107381e-01 -5.24817669e-02 -6.33634438e-02 -4.64820371e-01
  -5.58224383e-02  2.82236104e-02]
 [-8.54164250e-02  4.58236532e-01  2.60654799e-02 -1.03061341e-02
   1.83381052e-01 -3.64196803e-02 -2.48700153e-02 -5.03039950e-01
  -1.22934806e-01  1.16825082e-02]
 [ 3.33019461e-02  3.33282249e-01  4.21341845e-01  2.17762210e-01
   1.70512313e-01 -3.54400568e-01  5.99159146e-01  3.65374262e-01
  -1.18112026e-01 -5.33957813e-03]
 [-1.64612649e-02  3.52222072e-01  3.42959185e-01  9.54883053e-03
   2.74055416e-01  3.74008458e-01 -5.75509771e-01  4.58691652e-01
   1.64403271e-03 -1.13373049e-02]
 [ 1.79294610e-01 -1.58513692e-02  4.10725576e-02  8.87452774e-01
  -1.53933982e-01  3.28197248e-01  6.09858455e-04 -2.01785392e-01
  -3.52051684e-02  3.26662719e-02]
 [ 5.23037028e-02  9.46688616e-02  5.71471658e-01 -1.75867583e-01
  -7.67032323e-01 -6.60575689e-02 -1.28033114e-01 -1.43912898e-01
  -1.11402161e-02  6.41611230e-04]
 [ 2.34460825e-02 -2.02710155e-02  1.06847832e-02 -2.54691039e-02
  -2.32754652e-02  1.13659441e-02 -3.55323241e-03  1.21305615e-02
  -2.42841303e-02 -3.95673799e-02]
 [ 3.67635364e-03  6.33830658e-03 -3.34665094e-03  1.11622802e-03
```

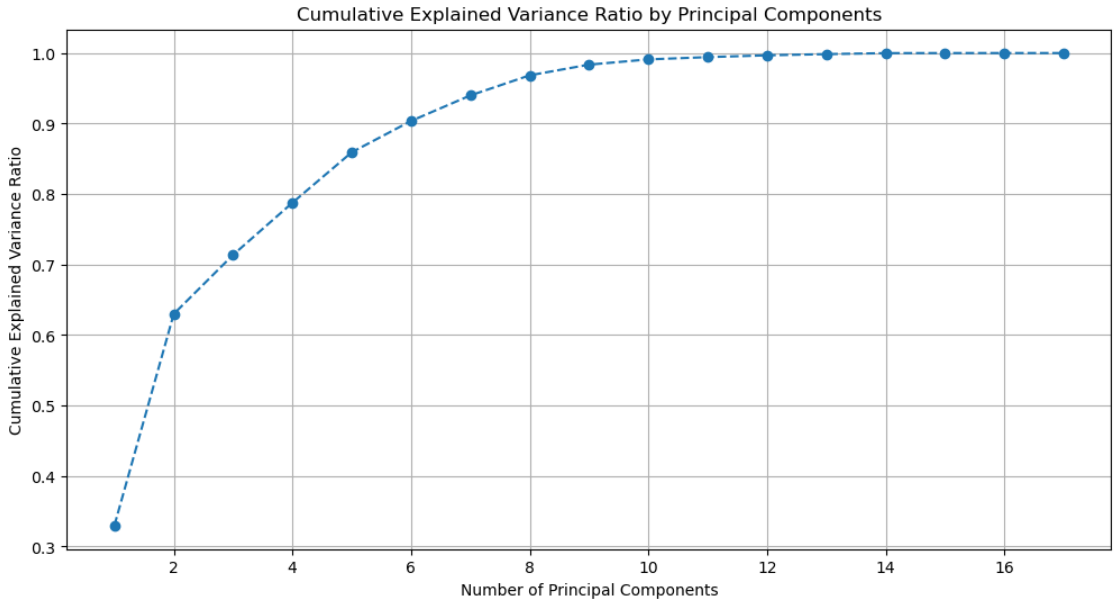
```
   6.49610831e-04  4.09550012e-03  2.66002362e-03 -2.39120534e-04
  -4.01923389e-03  4.31170449e-03]
 [ 7.96931455e-04 -1.96468141e-03  3.03758868e-07 -2.18869781e-05
  -7.03505870e-06 -4.00736022e-04 -7.92085865e-04  1.61832325e-03
   3.12226513e-05  2.74800469e-03]
 [-3.01987288e-04  1.72346365e-02 -6.57366055e-03  4.77426358e-03
  -6.54806113e-04 -3.23464444e-03 -5.05046058e-03 -4.91099615e-03
   1.69165482e-04  1.42363486e-02]]
PCA result:
 [[-1.45298279 -1.20568907 -1.19298748 … -1.57582236  2.03904053
   0.63124523]
 [-0.98150197 -2.31130157  1.78344446 … -0.79895196 -1.50886778
  -1.88503707]
 [ 1.279524    1.34137667 -1.89659081 …  0.11155883  0.57824958
  -0.10068733]
 …
 [ 0.27044947  1.17054401  0.90073122 … -0.63163065  0.59156529
   0.42884017]
 [ 0.33481131  0.34074828  0.14619402 …  0.11692463 -0.24535445
  -0.35586625]
 [ 0.07773511 -0.2011523  -0.24506787 … -0.07687367  0.6451856
  -0.17298874]]
```

[44]:
```
# Calculate the Explained Variance Ratio
explained_variance_ratio = eigenvalues / np.sum(eigenvalues)

# Calculate the Cumulative Explained Variance Ratio
cumulative_variance_ratio = np.cumsum(explained_variance_ratio)

# Print Summary
print("Eigenvalues:")
print(eigenvalues)

print("\nExplained Variance Ratio:")
print(explained_variance_ratio)

print("\nCumulative Explained Variance Ratio:")
print(cumulative_variance_ratio)
```

```
Eigenvalues:
[4.30782999e+00 3.91592806e+00 1.09440339e+00 9.65702723e-01
 9.34718337e-01 5.80040574e-01 4.72855854e-01 3.72276314e-01
 1.97546626e-01 9.72900530e-02 4.18921917e-02 3.35267315e-02
 2.19895207e-02 1.83200113e-02 1.30823147e-03 2.66454641e-04
 2.66010835e-05]

Explained Variance Ratio:
[3.29952194e-01 2.99935015e-01 8.38242919e-02 7.39666450e-02
```

```
 7.15934394e-02 4.44273938e-02 3.62177307e-02 2.85139819e-02
 1.51308066e-02 7.45179509e-03 3.20867364e-03 2.56793295e-03
 1.68425648e-03 1.40319556e-03 1.00202154e-04 2.04087193e-05
 2.03747266e-06]

Cumulative Explained Variance Ratio:
[0.32995219 0.62988721 0.7137115  0.78767815 0.85927158 0.90369898
 0.93991671 0.96843069 0.9835615  0.99101329 0.99422197 0.9967899
 0.99847416 0.99987735 0.99997755 0.99999796 1.         ]
```

[45]:
```
#Plotting of Cumulative variance ratio
plt.figure(figsize=(12, 6))
plt.plot(range(1, len(cumulative_variance_ratio) + 1),␣
 ↪cumulative_variance_ratio, marker='o', linestyle='--')
plt.title('Cumulative Explained Variance Ratio by Principal Components')
plt.xlabel('Number of Principal Components')
plt.ylabel('Cumulative Explained Variance Ratio')
plt.grid(True)
plt.show()
```

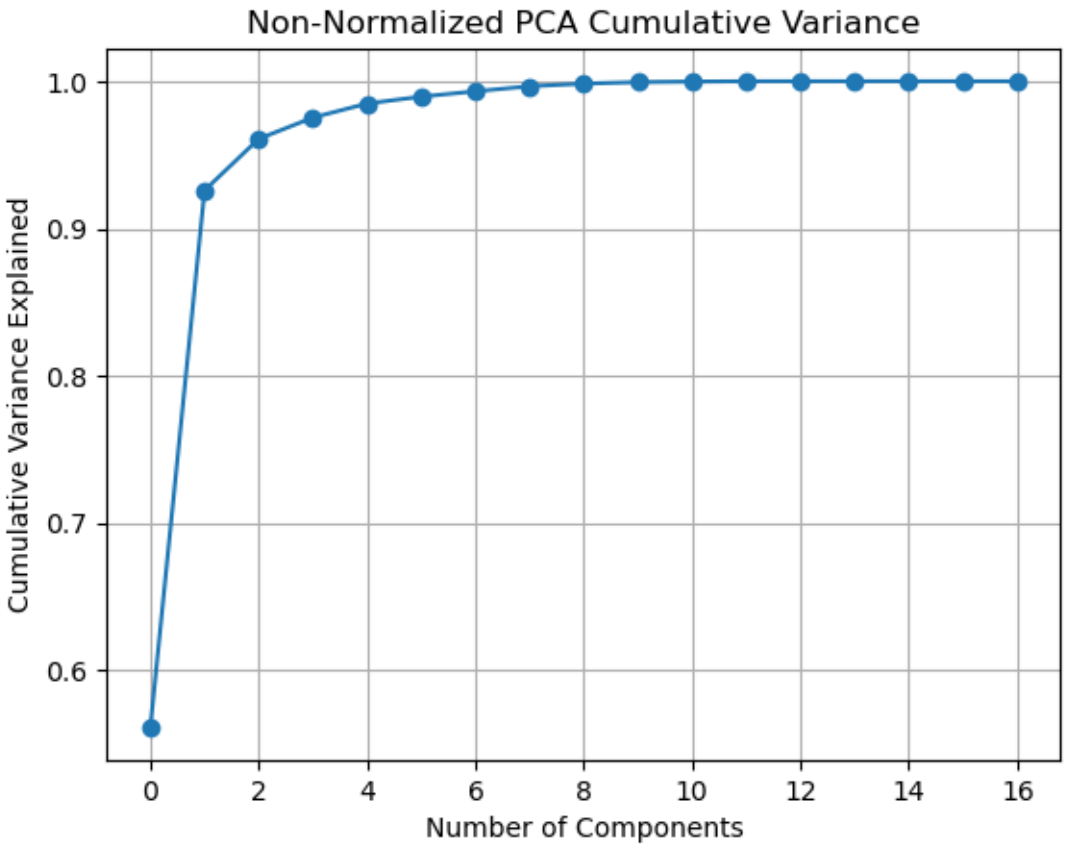


Non-Normalized PCA Process

[46]:
```
from sklearn.decomposition import PCA
import numpy as np
import matplotlib.pyplot as plt

# Assuming df_numerical_no_missing is your cleaned numerical dataset
```

```
#dropping all the catagorical columns
columns_dropped_NN = ['College Name', 'State','Public (1)/ Private (2)']
# Use the drop() method to remove the specified columns
university_df_bkp.drop(columns=columns_dropped_NN, inplace=True)
display(columns_dropped_NN)

university_df_bkp_NN = university_df_bkp.dropna()

# Step 3a: Non-Normalized PCA
pca = PCA(n_components=17)  # Adjust the number of components as needed
principal_components_NN = pca.fit_transform(university_df_bkp_NN)

# Display explained variance for each component
print("\nNon-Normalized PCA Explained Variance:")
print(pca.explained_variance_ratio_)

# Cumulative Variance Plot
cumulative_variance_non_normalized = np.cumsum(pca.explained_variance_ratio_)
plt.plot(cumulative_variance_non_normalized, marker='o')
plt.title('Non-Normalized PCA Cumulative Variance')
plt.xlabel('Number of Components')
plt.ylabel('Cumulative Variance Explained')
plt.grid(True)
plt.show()

# Loadings for each variable on the first principal component
loadings_non_normalized = pca.components_[0]

# Identify columns that influence the model based on loadings
influential_columns_non_normalized = university_df_bkp_NN.columns[np.
  ↪abs(loadings_non_normalized) > 0.1]
print("\nInfluential Columns (Non-Normalized PCA):")
print(influential_columns_non_normalized)

# Calculate Covariance Matrix
cov_matrix_non_normalized = np.cov(university_df_bkp_NN, rowvar=False)
print("\nCovariance Matrix (Non-Normalized PCA):")
print(cov_matrix_non_normalized)
```

```
['College Name', 'State', 'Public (1)/ Private (2)']


Non-Normalized PCA Explained Variance:
[5.61369738e-01 3.64524323e-01 3.49690910e-02 1.44578137e-02
 9.51484565e-03 4.69611378e-03 3.62308432e-03 3.42738841e-03
 1.77302096e-03 1.02888325e-03 3.62643169e-04 2.46160387e-04
 3.68905624e-06 1.59580389e-06 1.23425612e-06 2.88817856e-07
 8.58481964e-08]
```

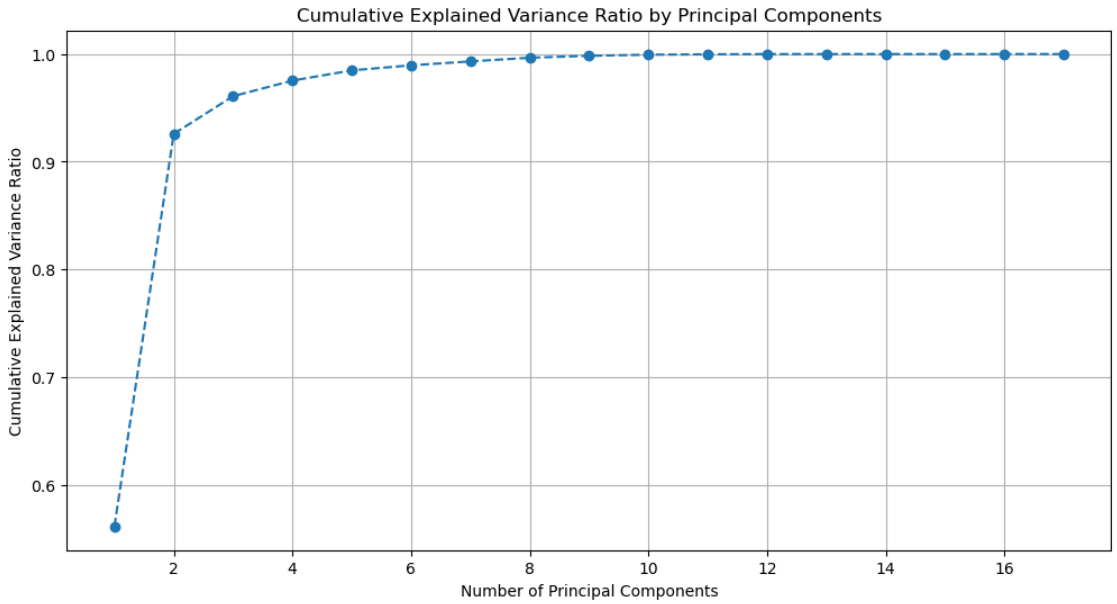



```
Influential Columns (Non-Normalized PCA):
Index(['# appli. rec'd', '# appl. accepted', '# FT undergrad',
       '# PT undergrad', 'in-state tuition', 'out-of-state tuition'],
      dtype='object')

Covariance Matrix (Non-Normalized PCA):
[[ 1.65904543e+07  9.54310519e+06  3.05765886e+06  2.24863280e+04
   2.69371379e+04  1.51318596e+07  2.59567975e+06 -2.67202137e+06
   1.33074695e+06  6.75007339e+05  2.90678437e+05  4.34500483e+05
   7.00497340e+04  4.94388356e+05  2.56023537e+04  1.23623243e+03
   1.34568309e+04]
 [ 9.54310519e+06  6.26877786e+06  2.05216120e+06  6.29397078e+03
   1.02393078e+04  1.01094140e+07  1.82047339e+06 -2.79470274e+06
  -6.11841651e+04  3.04870805e+05  7.63127825e+04  2.60660058e+05
   2.90102988e+04  3.51107238e+05  1.43313199e+04  1.60128769e+03
   4.46854854e+03]
 [ 3.05765886e+06  2.05216120e+06  8.38384340e+05  2.34279100e+03
   3.33972681e+03  4.14124062e+06  7.90019330e+05 -1.69539781e+06
  -5.51502537e+05  1.28870512e+04 -1.77628133e+04  9.24075871e+04
```

```
  9.17588489e+03  1.84694674e+05  4.88407217e+03  8.06476608e+02
  2.11640864e+02]
[ 2.24863280e+04  6.29397078e+03  2.34279100e+03  3.41480688e+02
  3.40321466e+02  8.84938164e+03 -3.60969303e+03  5.63908919e+04
  4.98766213e+04  4.52805606e+03  3.93650289e+03  1.73552920e+02
  3.49941062e+02 -1.64375669e+03  1.68027131e+02 -3.31603740e+01
  1.87182179e+02]
[ 2.69371379e+04  1.02393078e+04  3.33972681e+03  3.40321466e+02
  4.13078502e+02  1.47633466e+04 -3.06234997e+03  5.59518066e+04
  5.05269350e+04  4.69768672e+03  4.14652606e+03  1.15318047e+02
  4.14543583e+02 -1.71386542e+03  1.90847071e+02 -2.93798473e+01
  2.12213344e+02]
[ 1.51318596e+07  1.01094140e+07  4.14124062e+06  8.84938164e+03
  1.47633466e+04  2.18016751e+07  4.27846351e+06 -9.96484309e+06
 -3.71896212e+06  2.85601088e+04 -1.45687953e+05  4.14278423e+05
  5.36258930e+04  1.07337513e+06  2.43634065e+04  4.66232897e+03
 -3.57643550e+03]
[ 2.59567975e+06  1.82047339e+06  7.90019330e+05 -3.60969303e+03
 -3.06234997e+03  4.27846351e+06  2.38948657e+06 -2.90993860e+06
 -1.47186682e+06 -2.76721153e+04 -1.36951878e+04  4.92308435e+04
  2.02814029e+04  3.07658987e+05  3.55648611e+03  1.17243999e+03
 -6.59957320e+03]
[-2.67202137e+06 -2.79470274e+06 -1.69539781e+06  5.63908919e+04
  5.59518066e+04 -9.96484309e+06 -2.90993860e+06  3.04350217e+07
  2.22971433e+07  1.92945086e+06  1.82134617e+06 -4.30339122e+05
  4.97104931e+04 -1.31354642e+06  2.65389910e+04 -1.29963603e+04
  5.80330506e+04]
[ 1.33074695e+06 -6.11841651e+04 -5.51502537e+05  4.98766213e+04
  5.05269350e+04 -3.71896212e+06 -1.47186682e+06  2.22971433e+07
  1.85905159e+07  1.71507203e+06  1.48269525e+06 -1.49940701e+05
  4.74256542e+04 -9.40850110e+05  3.29319919e+04 -1.00194742e+04
  4.86151941e+04]
[ 6.75007339e+05  3.04870805e+05  1.28870512e+04  4.52805606e+03
  4.69768672e+03  2.85601088e+04 -2.76721153e+04  1.92945086e+06
  1.71507203e+06  5.08637280e+05  1.95084953e+05  2.33408360e+04
  2.24509299e+04 -8.64765914e+04  3.82168806e+03 -9.60359137e+02
  4.78296022e+03]
[ 2.90678437e+05  7.63127825e+04 -1.77628133e+04  3.93650289e+03
  4.14652606e+03 -1.45687953e+05 -1.36951878e+04  1.82134617e+06
  1.48269525e+06  1.95084953e+05  3.21331435e+05 -2.80306895e+03
  1.28492297e+04 -6.43338546e+04  3.04157623e+03 -7.59037545e+02
  4.22936331e+03]
[ 4.34500483e+05  2.60660058e+05  9.24075871e+04  1.73552920e+02
  1.15318047e+02  4.14278423e+05  4.92308435e+04 -4.30339122e+05
 -1.49940701e+05  2.33408360e+04 -2.80306895e+03  1.26632961e+05
  1.04865993e+03  4.68924382e+03  1.00007430e+03  1.06612072e+02
  2.98232710e+02]
[ 7.00497340e+04  2.90102988e+04  9.17588489e+03  3.49941062e+02
```

```
  4.14543583e+02  5.36258930e+04  2.02814029e+04  4.97104931e+04
  4.74256542e+04  2.24509299e+04  1.28492297e+04  1.04865993e+03
  2.66295317e+04  1.69252339e+04  1.71132403e+01 -5.17576840e+01
  1.48892316e+02]
[ 4.94388356e+05  3.51107238e+05  1.84694674e+05 -1.64375669e+03
 -1.71386542e+03  1.07337513e+06  3.07658987e+05 -1.31354642e+06
 -9.40850110e+05 -8.64765914e+04 -6.43338546e+04  4.68924382e+03
  1.69252339e+04  4.64915656e+05 -8.40351434e+02  3.12810965e+02
 -2.96297499e+03]
[ 2.56023537e+04  1.43313199e+04  4.88407217e+03  1.68027131e+02
  1.90847071e+02  2.43634065e+04  3.55648611e+03  2.65389910e+04
  3.29319919e+04  3.82168806e+03  3.04157623e+03  1.00007430e+03
  1.71132403e+01 -8.40351434e+02  2.77743850e+02 -1.38356128e+01
  1.24942260e+02]
[ 1.23623243e+03  1.60128769e+03  8.06476608e+02 -3.31603740e+01
 -2.93798473e+01  4.66232897e+03  1.17243999e+03 -1.29963603e+04
 -1.00194742e+04 -9.60359137e+02 -7.59037545e+02  1.06612072e+02
 -5.17576840e+01  3.12810965e+02 -1.38356128e+01  1.52010688e+01
 -2.25693567e+01]
[ 1.34568309e+04  4.46854854e+03  2.11640864e+02  1.87182179e+02
  2.12213344e+02 -3.57643550e+03 -6.59957320e+03  5.80330506e+04
  4.86151941e+04  4.78296022e+03  4.22936331e+03  2.98232710e+02
  1.48892316e+02 -2.96297499e+03  1.24942260e+02 -2.25693567e+01
  3.29310431e+02]]
```

[47]:
```
plt.figure(figsize=(12, 6))
plt.plot(range(1, len(cumulative_variance_non_normalized) + 1),␣
 ↪cumulative_variance_non_normalized, marker='o', linestyle='--')
plt.title('Cumulative Explained Variance Ratio by Principal Components')
plt.xlabel('Number of Principal Components')
plt.ylabel('Cumulative Explained Variance Ratio')
plt.grid(True)
plt.show()
```



Insights Normalizing/standardizing the data in PCA is generally advised, and the decision is contingent on both the variable scale and feature significance. This practice ensures that all variables contribute equally to the analysis, if there is a possibility of weighing more of one over the other.

The "elbow" value in the cumulative explained variance ratio chart indicates a good point to stop and look for the number of principal components but when comparing the 2 methods, there is found to be a slight difference of components (one is 10 and one is 8), which is not advisable and backs the statement of weighing one component more than the other.

For example, When comparing the Higher loadings for PC1, it can be found that it contributes to about 0.91(or 91%) when the data is not normalised to 0.6 (60%) when it is normalised. This backs the above statement and further strengthen the need for normalising the data for PCA.

Question 6: Principal Component Analysis (PCA) with Cumulative and Explained Variance

Theoretical Understanding:

Define PCA and its role in data analysis. Principal Component Analysis (PCA) is a widely used dimensionality reduction technique in data analysis and machine learning. Its main purpose is to transform a dataset, potentially containing correlated features, into a new set of linearly uncorrelated features known as principal components. These components are arranged in a manner where the first one captures the highest variance present in the data, succeeded by subsequent components capturing progressively lower amounts of variance.

The various steps to calculate PCA are as follows:

1) Normalising/standardising the data

2) Covariance matrix
3) Calculating Eigen Values and Eigen Vectors
4) Sorting of Eigen values
5) Projection of data

Explain the mathematical principles behind PCA, focusing on Eigenvalues and Eigenvector Eigenvalues and eigenvectors play a pivotal role in Principal Component Analysis (PCA). The Eigenvectors determine the directions of the new feature space, known as principal components, while eigenvalues quantify the variance captured along each eigenvector. The higher eigenvalue signifies the greater importance for the corresponding eigenvector in elucidating the variability within the data.

Therefore, PCA streamlines intricate datasets by converting them into a novel coordinate system that highlights directions of maximum variance. Eigenvalues and eigenvectors are indispensable in this transformation, helping in the identification of the most influential features in the data.

[165]:
```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import LabelEncoder
from sklearn.decomposition import PCA
```

[166]:
```
file_path = '/Users/jaamiemaarshj/Desktop/  DAE Course Materials/Data Mining/
 ↪Assignment -1/wine.csv'

Wine_df = pd.read_csv(file_path)
display(Wine_df.head())
```

	Wine	Alcohol	Malic.acid	Ash	Acl	Mg	Phenols	Flavanoids	\
0	1	14.23	1.71	2.43	15.6	127	2.80	3.06	
1	1	13.20	1.78	2.14	11.2	100	2.65	2.76	
2	1	13.16	2.36	2.67	18.6	101	2.80	3.24	
3	1	14.37	1.95	2.50	16.8	113	3.85	3.49	
4	1	13.24	2.59	2.87	21.0	118	2.80	2.69	

	Nonflavanoid.phenols	Proanth	Color.int	Hue	OD	Proline
0	0.28	2.29	5.64	1.04	3.92	1065
1	0.26	1.28	4.38	1.05	3.40	1050
2	0.30	2.81	5.68	1.03	3.17	1185
3	0.24	2.18	7.80	0.86	3.45	1480
4	0.39	1.82	4.32	1.04	2.93	735

[167]:
```
Wine_df.describe(include='all')
```

[167]:

	Wine	Alcohol	Malic.acid	Ash	Acl	Mg	\
count	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	
mean	1.938202	13.000618	2.336348	2.366517	19.494944	99.741573	

```
std      0.775035    0.811827    1.117146    0.274344    3.339564   14.282484
min      1.000000   11.030000    0.740000    1.360000   10.600000   70.000000
25%      1.000000   12.362500    1.602500    2.210000   17.200000   88.000000
50%      2.000000   13.050000    1.865000    2.360000   19.500000   98.000000
75%      3.000000   13.677500    3.082500    2.557500   21.500000  107.000000
max      3.000000   14.830000    5.800000    3.230000   30.000000  162.000000

          Phenols  Flavanoids  Nonflavanoid.phenols     Proanth   Color.int  \
count  178.000000  178.000000            178.000000  178.000000  178.000000
mean     2.295112    2.029270              0.361854    1.590899    5.058090
std      0.625851    0.998859              0.124453    0.572359    2.318286
min      0.980000    0.340000              0.130000    0.410000    1.280000
25%      1.742500    1.205000              0.270000    1.250000    3.220000
50%      2.355000    2.135000              0.340000    1.555000    4.690000
75%      2.800000    2.875000              0.437500    1.950000    6.200000
max      3.880000    5.080000              0.660000    3.580000   13.000000

              Hue          OD      Proline
count  178.000000  178.000000   178.000000
mean     0.957449    2.611685   746.893258
std      0.228572    0.709990   314.907474
min      0.480000    1.270000   278.000000
25%      0.782500    1.937500   500.500000
50%      0.965000    2.780000   673.500000
75%      1.120000    3.170000   985.000000
max      1.710000    4.000000  1680.000000
```

[168]:
```
Nan_values = Wine_df.isnull().sum()
print (Nan_values)
```

```
Wine                    0
Alcohol                 0
Malic.acid              0
Ash                     0
Acl                     0
Mg                      0
Phenols                 0
Flavanoids              0
Nonflavanoid.phenols    0
Proanth                 0
Color.int               0
Hue                     0
OD                      0
Proline                 0
dtype: int64
```

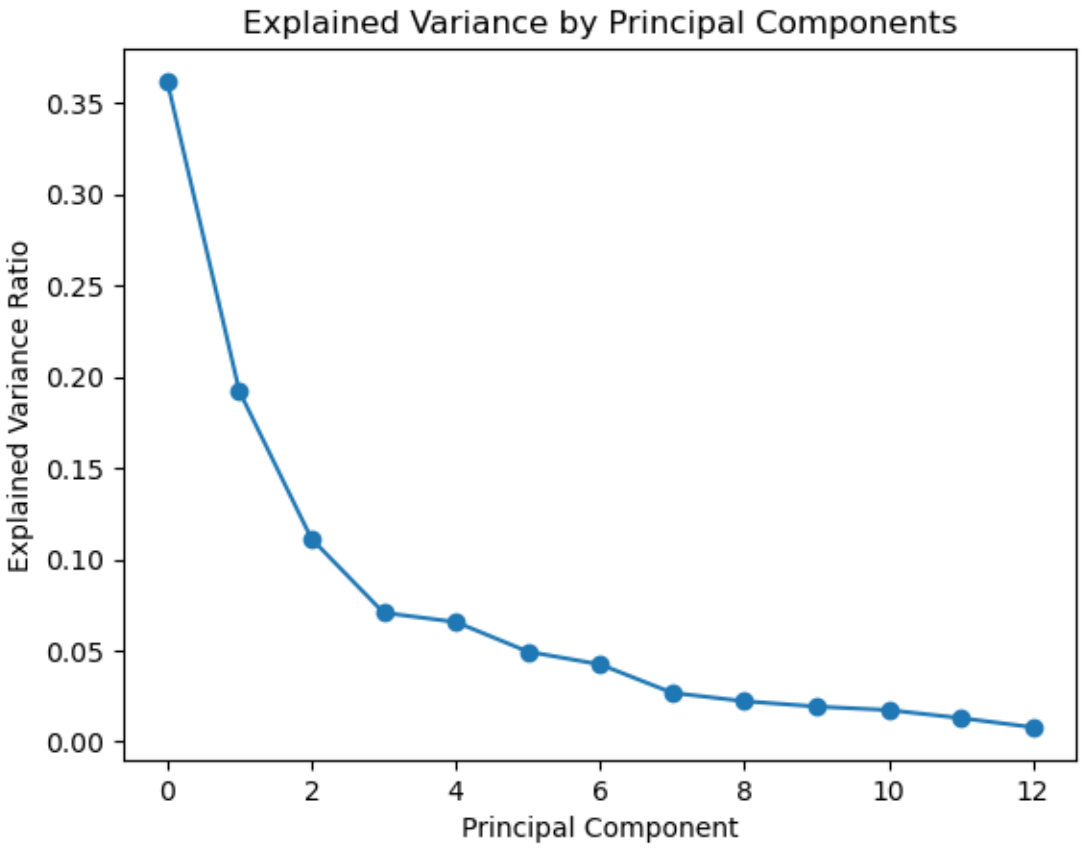
Using PCA library method

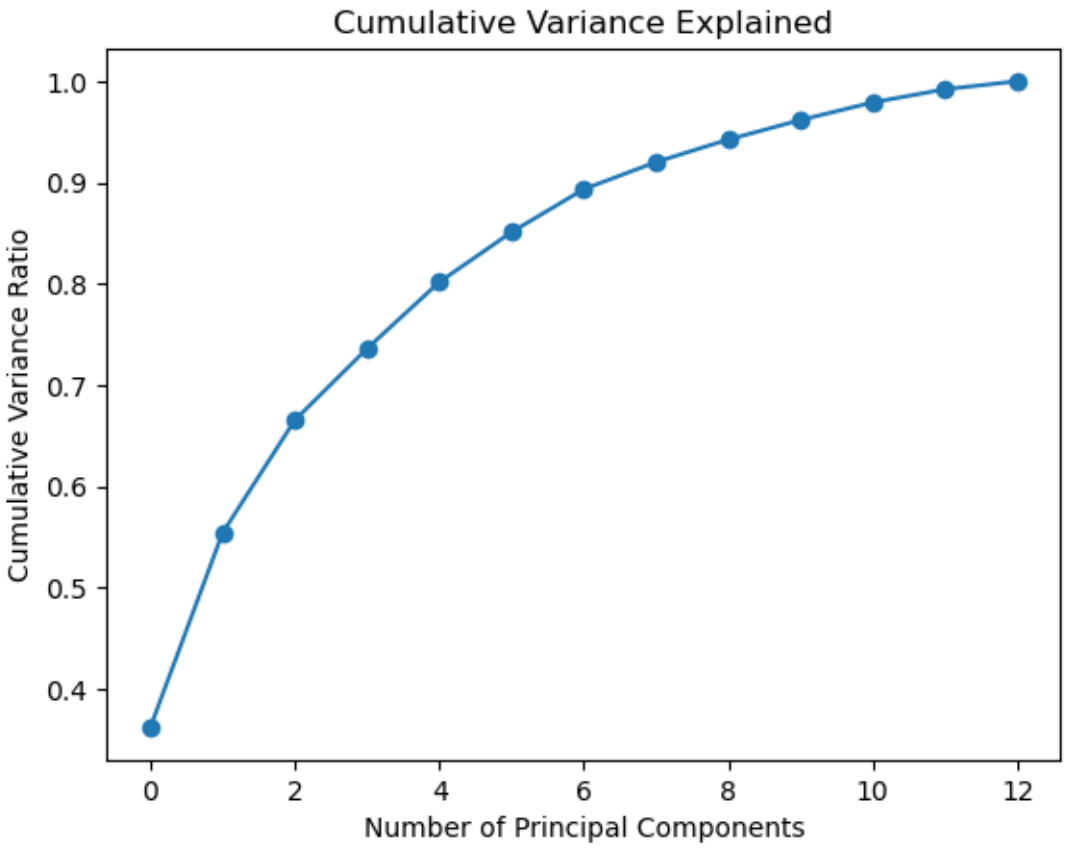
[170]:
```
# Exploratory Data Analysis
print(Wine_df.info())
print(Wine_df.describe())

# Normalize the dataset
scaler = StandardScaler()
Wine_df_normal = scaler.fit_transform(Wine_df.iloc[:, 1:])

# Task 3: PCA Implementation
pca = PCA()
principal_components = pca.fit_transform(Wine_df_normal)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 178 entries, 0 to 177
Data columns (total 14 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   Wine                  178 non-null    int64
 1   Alcohol               178 non-null    float64
 2   Malic.acid            178 non-null    float64
 3   Ash                   178 non-null    float64
 4   Acl                   178 non-null    float64
 5   Mg                    178 non-null    int64
 6   Phenols               178 non-null    float64
 7   Flavanoids            178 non-null    float64
 8   Nonflavanoid.phenols  178 non-null    float64
 9   Proanth               178 non-null    float64
 10  Color.int             178 non-null    float64
 11  Hue                   178 non-null    float64
 12  OD                    178 non-null    float64
 13  Proline               178 non-null    int64
dtypes: float64(11), int64(3)
memory usage: 19.6 KB
None
             Wine     Alcohol  Malic.acid         Ash         Acl          Mg  \
count  178.000000  178.000000  178.000000  178.000000  178.000000  178.000000
mean     1.938202   13.000618    2.336348    2.366517   19.494944   99.741573
std      0.775035    0.811827    1.117146    0.274344    3.339564   14.282484
min      1.000000   11.030000    0.740000    1.360000   10.600000   70.000000
25%      1.000000   12.362500    1.602500    2.210000   17.200000   88.000000
50%      2.000000   13.050000    1.865000    2.360000   19.500000   98.000000
75%      3.000000   13.677500    3.082500    2.557500   21.500000  107.000000
max      3.000000   14.830000    5.800000    3.230000   30.000000  162.000000

          Phenols  Flavanoids  Nonflavanoid.phenols     Proanth   Color.int  \
count  178.000000  178.000000            178.000000  178.000000  178.000000
mean     2.295112    2.029270              0.361854    1.590899    5.058090
```

```
std      0.625851    0.998859              0.124453    0.572359    2.318286
min      0.980000    0.340000              0.130000    0.410000    1.280000
25%      1.742500    1.205000              0.270000    1.250000    3.220000
50%      2.355000    2.135000              0.340000    1.555000    4.690000
75%      2.800000    2.875000              0.437500    1.950000    6.200000
max      3.880000    5.080000              0.660000    3.580000   13.000000

              Hue          OD      Proline
count  178.000000  178.000000   178.000000
mean     0.957449    2.611685   746.893258
std      0.228572    0.709990   314.907474
min      0.480000    1.270000   278.000000
25%      0.782500    1.937500   500.500000
50%      0.965000    2.780000   673.500000
75%      1.120000    3.170000   985.000000
max      1.710000    4.000000  1680.000000
```

[172]:
```
# Task 4: Variance Analysis
explain_variance_ratio_PCA = pca.explained_variance_ratio_
cum_variance_ratio_PCA = explained_variance_ratio.cumsum()

# Plot explained variance by each principal component
plt.plot(explain_variance_ratio_PCA, marker='o')
plt.title('Explained Variance by Principal Components')
plt.xlabel('Principal Component')
plt.ylabel('Explained Variance Ratio')
plt.show()

# Plot cumulative variance explained
plt.plot(cum_variance_ratio_PCA, marker='o')
plt.title('Cumulative Variance Explained')
plt.xlabel('Number of Principal Components')
plt.ylabel('Cumulative Variance Ratio')
plt.show()
```

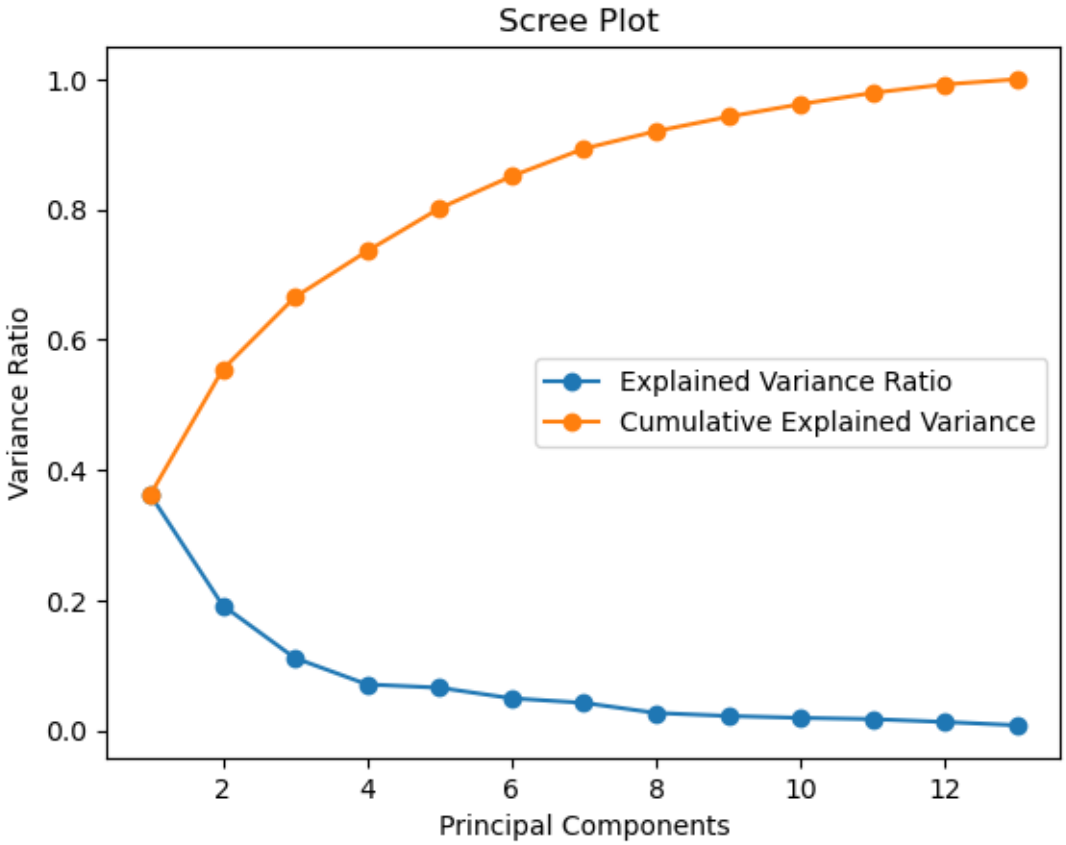

Explained Variance by Principal Components
Explained Variance Ratio
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
0
2
4
6
8
10
12
Principal Component



Insights From the above graphs, A higher cumulative variance indicates that the retained number of principal components, which is may be considered as 8 or 9 (taken as the elbow) which collectively capture a significant portion of the dataset's variability

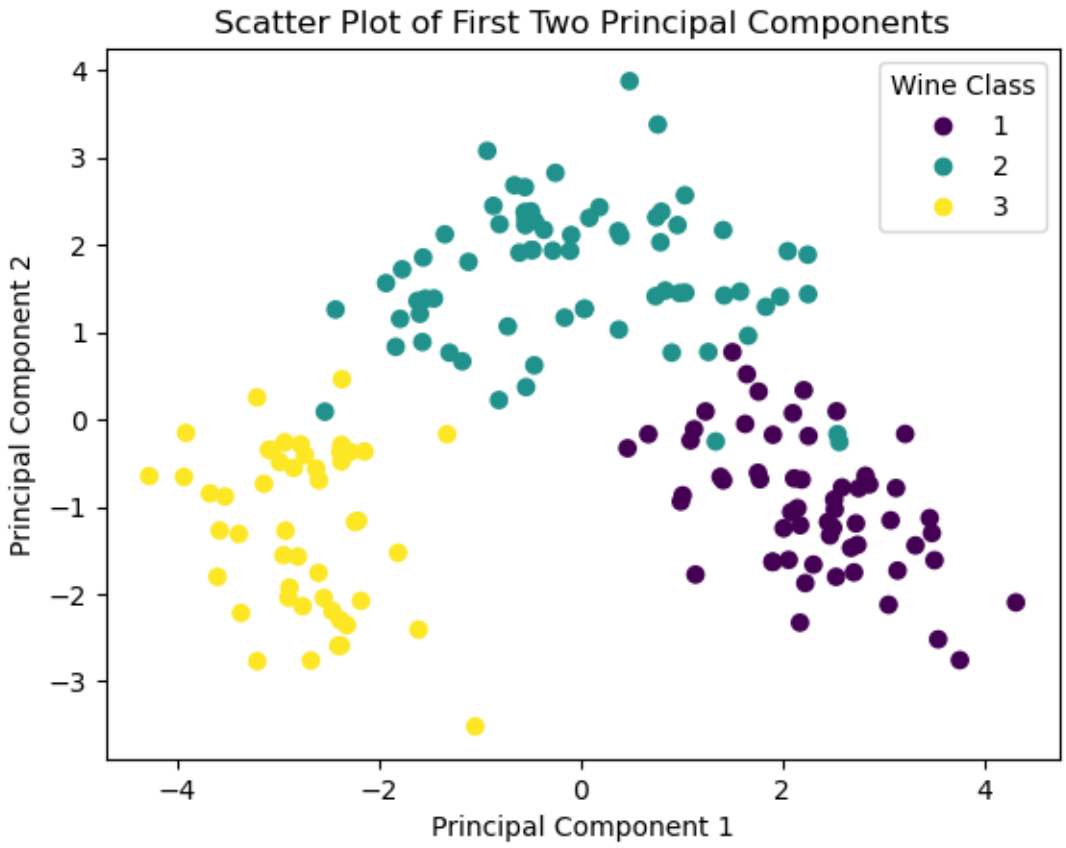
[173]:
```
#Task 5: Plotting the Scree plot

# Plot explained variance ratio
plt.plot(range(1, len(explain_variance_ratio_PCA) + 1),␣
 ↪explain_variance_ratio_PCA, marker='o', linestyle='-', label='Explained␣
 ↪Variance Ratio')

# Plot cumulative explained variance
plt.plot(range(1, len(cum_variance_ratio_PCA) + 1), cum_variance_ratio_PCA,␣
 ↪marker='o', linestyle='-', label='Cumulative Explained Variance')

# Add labels and title
plt.xlabel('Principal Components')
plt.ylabel('Variance Ratio')
plt.title('Scree Plot ')
```

```
plt.legend()

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



[175]:
```
# Task 6: Visualization and Interpretation
scatter = plt.scatter(principal_components[:, 0], principal_components[:, 1],␣
 ↪c=Wine_df["Wine"], cmap='viridis')

# Add legend
legend_labels = Wine_df["Wine"].unique()
legend = plt.legend(*scatter.legend_elements(), title='Wine Class', loc='upper␣
 ↪right')

# Set plot labels
plt.title('Scatter Plot of First Two Principal Components')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')

# Show the plot
```

```
plt.show()
```



Insights: In the content of the dataset, the Wine class 1 has Negative loadings on the principal components which means that the original features have an opposite influence on that component whereas the other 2 class have positive loadings.

It can also be found out that, all the values of relevant class are clustered close together and presence of less outliers for that particular class.