PRACTICE #03

Scikit-learn with PCA & LDA

(*Keyword:* scikit-learn, PCA, LDA*)*

I. Goals

- Students can use **Python** with **Scikit-learn** library with Multivariate Analysis.

II. Introduction

- **Scikit-learn** (formerly *scikits.learn* and also known as *sklearn*) is a free software machine learning library for the Python programming language.
- **Scikit-learn** features various classification, regression, and clustering algorithms including support vector machines, random forests, gradient boosting, k-means, and DBSCAN, and is designed to interoperate with the Python numerical and scientific libraries *NumPy* and *SciPy*.

III. Content

1. Prepare the necessary programming environment
   - **Python** programming language, minimum recommend version **3.6**
   - IDE / Text Editor: recommend JetBrains PyCharm Community (**PyCharm**) or Microsoft Visual Studio Code (**VS Code**)
   - Recommend use a **virtual environment** for developing:
   https://www.jetbrains.com/help/pycharm/creating-virtual-environment.html
   https://code.visualstudio.com/docs/python/environments
   - Recommend install library by using ***pip:***
   https://www.w3schools.com/python/python_pip.asp

2. Features
   Implement a program with the following features by using Python:
   - find or create an data CSV file, with the class identifier column and other properties (check the sample CSV file), describe the data info in the report.
   - use Scikit-learn (check the sample source code), implement some basic multivariate analysis with visualization, list the corresponding comments in the report.

- apply PCA & LDA by using Scikit-learn with the chose data, list the corresponding comments in the report.

IV. Requirements

1. The directory structure of the compressed submission
   - *doc*: report files include MSSV_report_p03.doc and MSSV_report_p03.pdf
   - *source*: contains entire source code, removed temporary files, intermediate compiled files if exists...
   - *data*: contains the data files
   - *bonus*: use other analysis methods than PCA & LDA, other libraries and compare with Scikit-learn
2. Other requirements
   - The report should be presented clearly and intuitively: list the functions/features included in the program with proof images, summary the usage and implementation (for example: through pseudo-code, description of methods, or how to do it, *do not copy the source code into the report*).
   - The source code needs to be commented on the corresponding lines.
   - It is strictly forbidden to copy other students' work, if detected, both the copyist and the person being copied will be considered with corresponding sanctioning decisions.

V. Additional guidance

1. Install dependencies with a virtual environment by using pip

```
pip install virtualenv
virtualenv your-env
source your-env/bin/activate
(your-env) pip install -r requirements.txt
```

   Import libraries

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import scipy.stats as stats
import seaborn as sns
from sklearn.preprocessing import scale
from sklearn.decomposition import PCA
```

2. Check the sample CSV data file description

   - Download: https://archive.ics.uci.edu/ml/datasets/wine

   - The data is the results of a chemical analysis of wines grown in the same region in Italy but derived from three different cultivars. The analysis determined the quantities of 13 constituents found in each of the three types of wines.

The attributes are:

1) Alcohol

2) Malic acid

3) Ash

4) Alcalinity of ash

5) Magnesium

6) Total phenols

7) Flavanoids

8) Nonflavanoid phenols

9) Proanthocyanins

10)Color intensity

11)Hue

12)OD280/OD315 of diluted wines

13)Proline

- All attributes are continuous. The 1st attribute is the class identifier (1-3).

3. Some basic visualizations with the given sample CSV file

- Read data from the given sample CSV file

```
# read CSV data with Pandas
data = pd.read_csv("data/wine.csv")

# setup data column
data.columns = ["V" + str(i) for i in range(1,
len(data.columns) + 1)]
# independent variables data
X = data.loc[:, "V2":]
# dependent variable data
y = data.V1

print("## Data:")
print(data)

print("## Head:")
print(data.head())

print("## Tail:")
print(data.tail())

print("## Info:")
data.info()
```

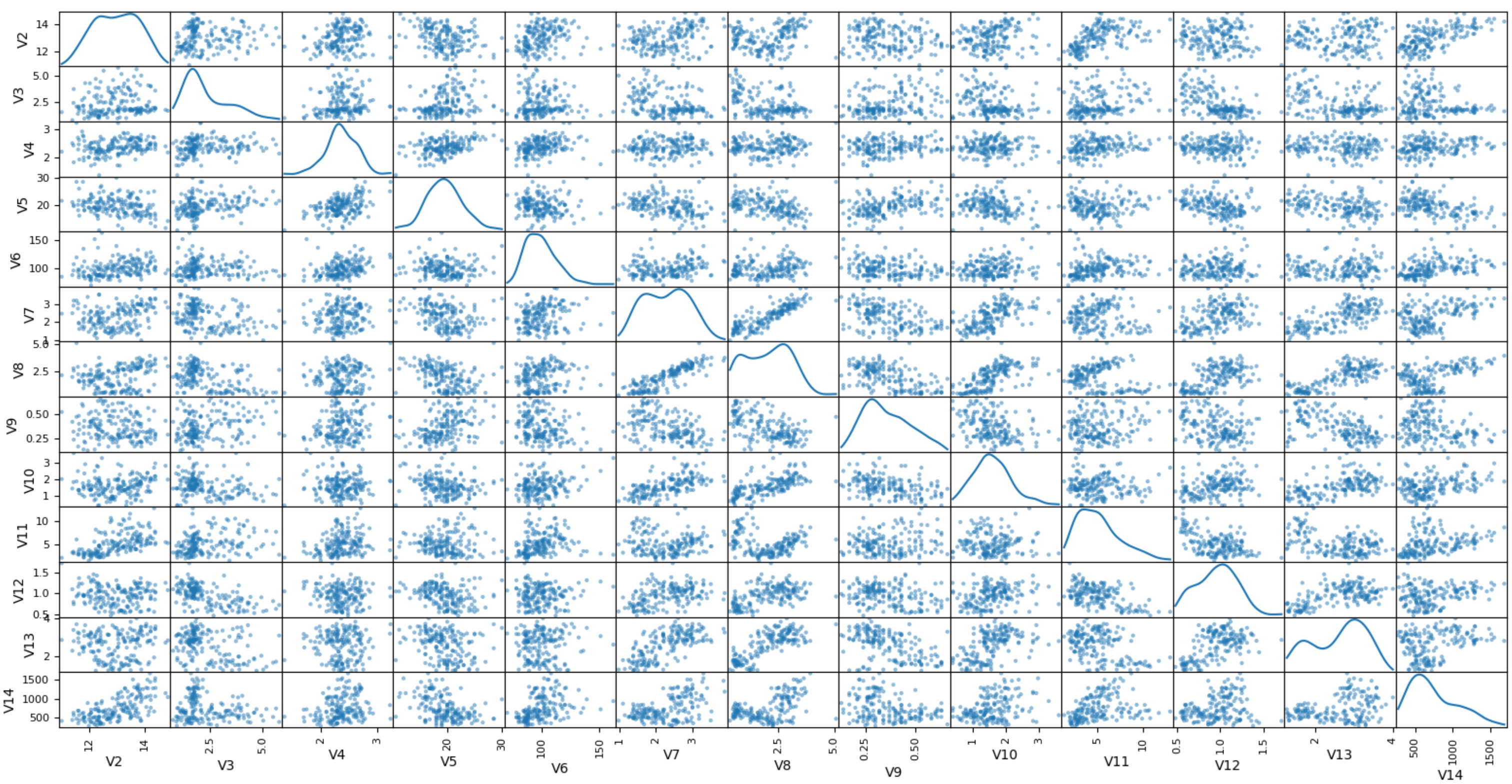
Output:

```
## Data:
     V1     V2    V3    V4    V5   V6  ...    V9   V10
V11   V12   V13   V14
0     1  13.20  1.78  2.14  11.2  100  ...  0.26  1.28
4.38  1.05  3.40  1050
1     1  13.16  2.36  2.67  18.6  101  ...  0.30  2.81
5.68  1.03  3.17  1185
2     1  14.37  1.95  2.50  16.8  113  ...  0.24  2.18
7.80  0.86  3.45  1480
3     1  13.24  2.59  2.87  21.0  118  ...  0.39  1.82
4.32  1.04  2.93   735
4     1  14.20  1.76  2.45  15.2  112  ...  0.34  1.97
6.75  1.05  2.85  1450
..   ..    ...   ...   ...   ...  ...  ...   ...   ...
...   ...   ...   ...
172   3  13.71  5.65  2.45  20.5   95  ...  0.52  1.06
7.70  0.64  1.74   740
173   3  13.40  3.91  2.48  23.0  102  ...  0.43  1.41
7.30  0.70  1.56   750
174   3  13.27  4.28  2.26  20.0  120  ...  0.43  1.35
10.20  0.59  1.56   835
175   3  13.17  2.59  2.37  20.0  120  ...  0.53  1.46
9.30  0.60  1.62   840
176   3  14.13  4.10  2.74  24.5   96  ...  0.56  1.35
9.20  0.61  1.60   560
[177 rows x 14 columns]
## Head:
   V1     V2    V3    V4    V5   V6  ...    V9   V10   V11
V12   V13   V14
0   1  13.20  1.78  2.14  11.2  100  ...  0.26  1.28  4.38
1.05  3.40  1050
1   1  13.16  2.36  2.67  18.6  101  ...  0.30  2.81  5.68
1.03  3.17  1185
2   1  14.37  1.95  2.50  16.8  113  ...  0.24  2.18  7.80
0.86  3.45  1480
3   1  13.24  2.59  2.87  21.0  118  ...  0.39  1.82  4.32
1.04  2.93   735
4   1  14.20  1.76  2.45  15.2  112  ...  0.34  1.97  6.75
1.05  2.85  1450
[5 rows x 14 columns]
## Tail:
     V1     V2    V3    V4    V5   V6  ...    V9   V10   V11
V12   V13  V14
172   3  13.71  5.65  2.45  20.5   95  ...  0.52  1.06   7.7
0.64  1.74  740
173   3  13.40  3.91  2.48  23.0  102  ...  0.43  1.41   7.3
0.70  1.56  750
174   3  13.27  4.28  2.26  20.0  120  ...  0.43  1.35  10.2
0.59  1.56  835
175   3  13.17  2.59  2.37  20.0  120  ...  0.53  1.46   9.3
0.60  1.62  840
176   3  14.13  4.10  2.74  24.5   96  ...  0.56  1.35   9.2
0.61  1.60  560
[5 rows x 14 columns]
## Info:
<class 'pandas.core.frame.DataFrame'>
```

```
RangeIndex: 177 entries, 0 to 176
Data columns (total 14 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   V1      177 non-null    int64
 1   V2      177 non-null    float64
 2   V3      177 non-null    float64
 3   V4      177 non-null    float64
 4   V5      177 non-null    float64
 5   V6      177 non-null    int64
 6   V7      177 non-null    float64
 7   V8      177 non-null    float64
 8   V9      177 non-null    float64
 9   V10     177 non-null    float64
 10  V11     177 non-null    float64
 11  V12     177 non-null    float64
 12  V13     177 non-null    float64
 13  V14     177 non-null    int64
dtypes: float64(11), int64(3)
```

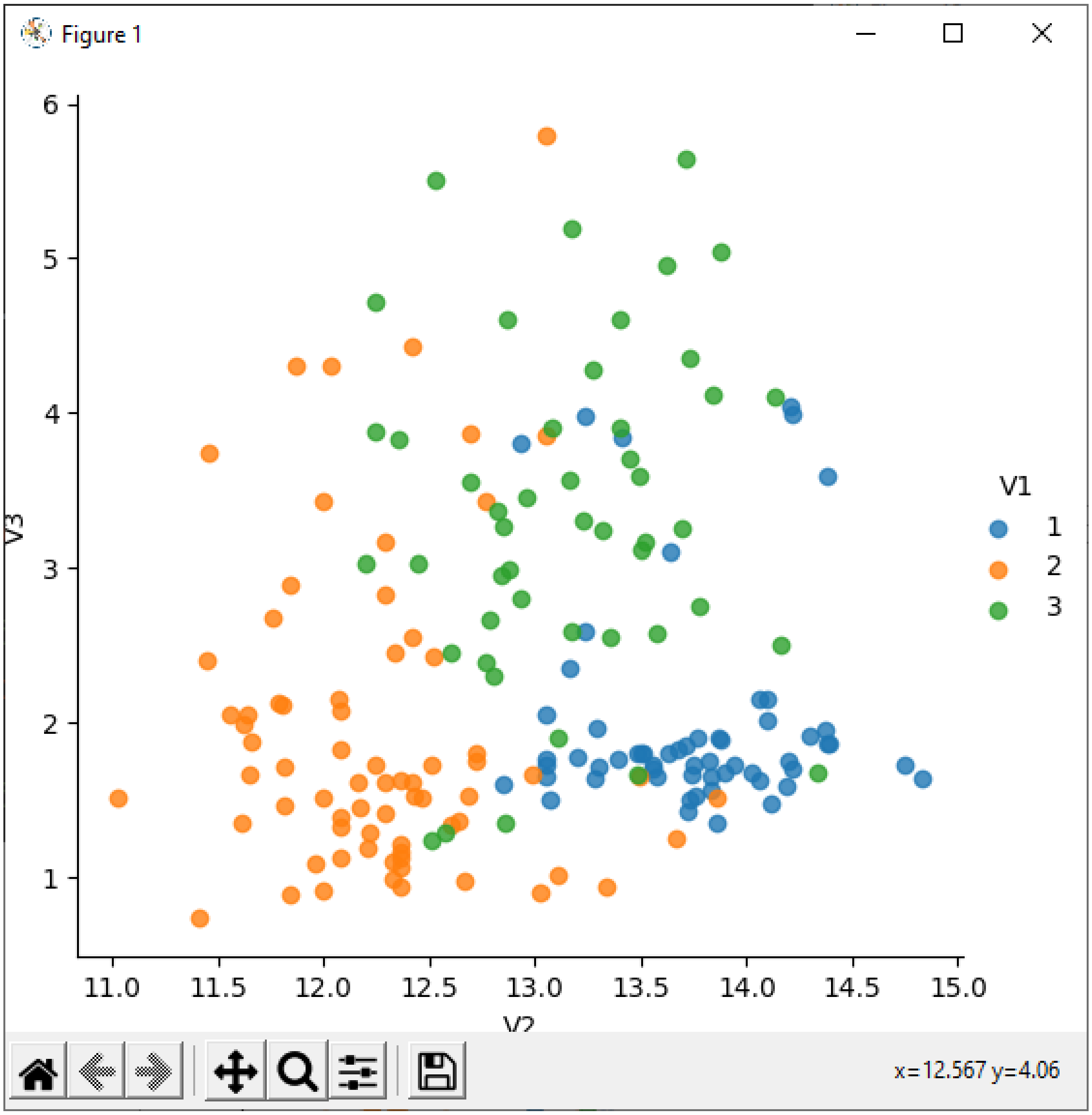
- Plot data for visualization

```
pd.plotting.scatter_matrix(data.loc[:, "V2":"V14"],
diagonal="kde", figsize=(20, 15))
plt.show()
```



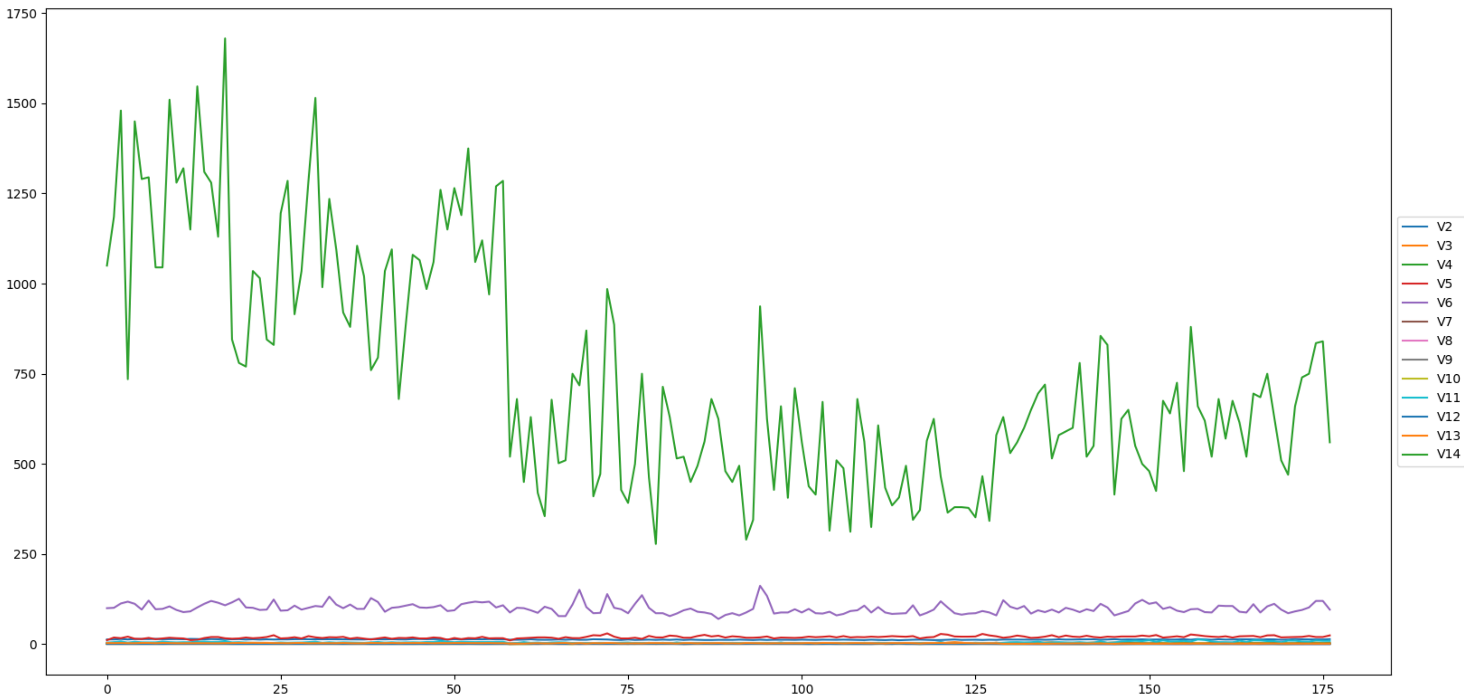
- Scatterplot with the data points labelled by their Group

```
for i in range(2, 14):
    sns.lmplot(x="V" + str(i), y="V" + str(i + 1), data=data,
hue="V1", fit_reg=False)
```



- Profile plot, used to shows the variation in each of the variables, by plotting the value of each of the variables for each of the samples

```
ax = data[["V2", "V3", "V4", "V5", "V6", "V7", "V8", "V9", "V10", "V11", "V12", "V13", "V14"]].plot(figsize=(20, 15))
ax.legend(loc='center left', bbox_to_anchor=(1, 0.5))
```



- Calculating summary statistics for multivariate data

```
print(X.apply(np.mean))
print(X.apply(np.std))
print(X.apply(np.max))
print(X.apply(np.min))
```

Output:

→ Mean

```
V2      12.993672
V3       2.339887
V4       2.366158
V5      19.516949
V6      99.587571
V7       2.292260
V8       2.023446
V9       0.362316
V10      1.586949
V11      5.054802
V12      0.956983
V13      2.604294
V14    745.096045
```

→ Standard deviation

```
V2       0.806520
V3       1.116148
V4       0.274302
V5       3.326634
V6      14.133922
V7       0.624693
V8       0.995833
V9       0.124300
V10      0.569928
V11      2.317871
V12      0.228487
V13      0.703108
V14    313.993283
```

→ Max

```
V2       14.83
V3        5.80
V4        3.23
V5       30.00
V6      162.00
V7        3.88
V8        5.08
V9        0.66
V10       3.58
V11      13.00
V12       1.71
V13       4.00
V14    1680.00
```

- Means and variances per group

```
def print_mean_and_sd_by_group(variables, group_variable):
    data_group_by = variables.groupby(group_variable)

    print("## Means:")
    print(data_group_by.apply(np.mean))

    print("\n## Standard deviations:")
    print(data_group_by.apply(np.std))

    print("\n## Sample sizes:")
    print(pd.DataFrame(data_group_by.apply(len)))


print_mean_and_sd_by_group(X, y)
```

Output:

```
## Means:
           V2        V3        V4  ...       V12       V13
V14
V1                                 ...
1   13.736379  2.015862  2.456034  ...  1.062414  3.144655
1116.586207
2   12.278732  1.932676  2.244789  ...  1.056282  2.785352
519.507042
3   13.153750  3.333750  2.437083  ...  0.682708  1.683542
629.895833

[3 rows x 13 columns]

## Standard deviations:
          V2        V3        V4  ...       V12       V13
V14
V1                                ...
1   0.457635  0.687396  0.227141  ...  0.116446  0.342512
221.418938
2   0.534162  1.008391  0.313238  ...  0.201503  0.493064
156.100173
3   0.524689  1.076514  0.182756  ...  0.113243  0.269262
113.891805

[3 rows x 13 columns]

## Sample sizes:
     0
V1
1   58
2   71
3   48
```

- Check more functions which implemented in the sample source code

- Standardising variables

```
standardisedX = scale(X)
standardisedX = pd.DataFrame(standardisedX, index=X.index,
columns=X.columns)

print(standardisedX.apply(np.mean))
print(standardisedX.apply(np.std))
```

```
V2    -5.218675e-16
V3     2.810056e-16
V4    -3.813647e-16
V5    -2.408619e-16
V6    -8.028731e-17
V7    -2.810056e-16
V8     1.605746e-16
V9    -6.021549e-16
V10   -4.014366e-17
V11    1.806465e-16
V12    6.021549e-16
V13    7.225858e-16
V14    1.605746e-16
dtype: float64

V2     1.0
V3     1.0
V4     1.0
V5     1.0
V6     1.0
V7     1.0
V8     1.0
V9     1.0
V10    1.0
V11    1.0
V12    1.0
V13    1.0
V14    1.0
dtype: float64
```

- Apply PCA using sklearn

```
pca = PCA().fit(standardisedX)
```

Check the summary of PCA results

```
def pca_summary(pca, standardised_data, out=True):
    names = ["PC"+str(i) for i in range(1,
len(pca.explained_variance_ratio_)+1)]
    a = list(np.std(pca.transform(standardised_data),
axis=0))
    b = list(pca.explained_variance_ratio_)
    c = [np.sum(pca.explained_variance_ratio_[:i]) for i in
range(1, len(pca.explained_variance_ratio_)+1)]
    columns = pd.MultiIndex.from_tuples([("sdev", "Standard
deviation"), ("varprop", "Proportion of Variance"),
("cumprop", "Cumulative Proportion")])
    summary = pd.DataFrame(list(zip(a, b, c)), index=names,
columns=columns)
    if out:
        print("Importance of components:")
        print(summary)
    return summary


summary = pca_summary(pca, standardisedX)
print(summary.sdev)
print(np.sum(summary.sdev**2))
```

Output:

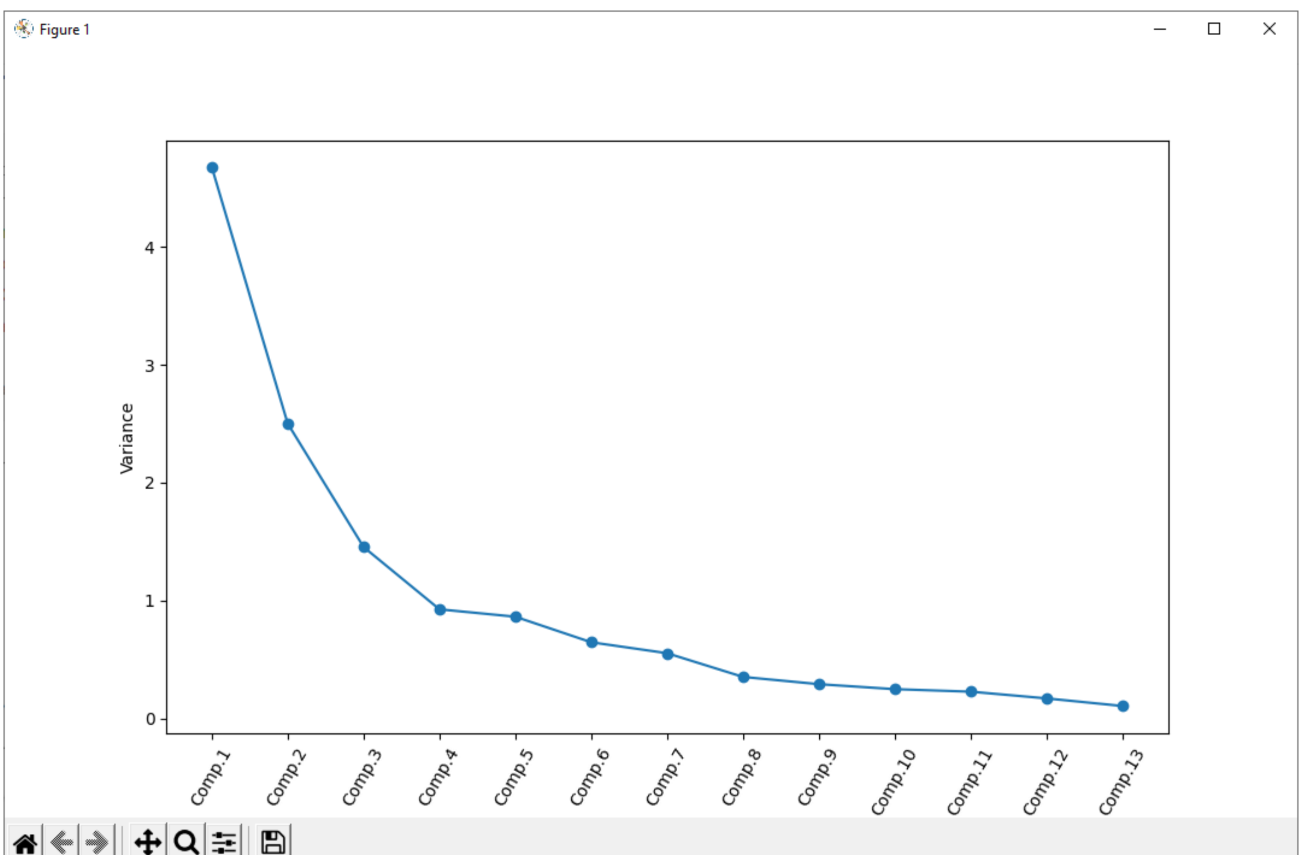
```
Importance of components:
                   sdev                varprop
cumprop
     Standard deviation Proportion of Variance Cumulative
Proportion
PC1            2.162822               0.359831
0.359831
PC2            1.581571               0.192413
0.552244
PC3            1.205541               0.111795
0.664038
PC4            0.961480               0.071111
0.735149
PC5            0.928298               0.066287
0.801437
PC6            0.803024               0.049604
0.851040
PC7            0.742955               0.042460
0.893500
PC8            0.592232               0.026980
0.920480
PC9            0.537755               0.022245
0.942725
PC10           0.496798               0.018985
0.961710
PC11           0.474805               0.017342
0.979052
PC12           0.410337               0.012952
0.992004
```

```
PC13              0.322412                 0.007996
1.000000
      Standard deviation
PC1                2.162822
PC2                1.581571
PC3                1.205541
PC4                0.961480
PC5                0.928298
PC6                0.803024
PC7                0.742955
PC8                0.592232
PC9                0.537755
PC10               0.496798
PC11               0.474805
PC12               0.410337
PC13               0.322412
Standard deviation     13.0
```

- Check how many principal components to retain

```
def scree_plot(pca, standardised_values):
    y = np.std(pca.transform(standardised_values),
axis=0)**2
    x = np.arange(len(y)) + 1
    plt.plot(x, y, "o-")
    plt.xticks(x, ["Comp."+str(i) for i in x], rotation=60)
    plt.ylabel("Variance")
    plt.show()


scree_plot(pca, standardisedX)
```



- Calculate the values of the first principal component

```
print(pca.components_[0])
print(np.sum(pca.components_[0]**2))


def calc_pc(variables, loadings):
    # find the number of samples in the data set and the
number of variables
    num_samples, num_variables = variables.shape
    # make a vector to store the component
    pc = np.zeros(num_samples)
    # calculate the value of the component for each sample
    for i in range(num_samples):
        value_i = 0
        for j in range(num_variables):
            value_ij = variables.iloc[i, j]
            loading_j = loadings[j]
            value_i = value_i + (value_ij * loading_j)
        pc[i] = value_i
    return pc


print(calc_pc(standardisedX, pca.components_[0]))
print(pca.transform(standardisedX)[:, 0])
```

Output:

```
[ 0.13788809 -0.24638109 -0.0043183  -0.23737955  0.1350017
0.39586939
  0.42439422 -0.29913568  0.31280321 -0.09328558
0.29956536  0.37720252
  0.28428101]

1.0000000000000002

[ 2.23024297  2.53192196  3.75467731  1.0201307
3.04919938  2.45822831
  2.06160512  2.51844454  2.76797089  3.48916135
1.76638133  2.12870494
  3.46649467  4.31363172  2.30845048  2.16745547
1.90220844  3.54012997
  2.09274066  3.1319081   1.10804505  2.55760384
1.67255267  1.78792909
  1.00222687  1.79246479  1.25516785  2.20313645
2.27398683  2.51223477
  2.68519586  1.64501308  1.90224111  1.42130848
1.92515877  1.39514757
  1.14008674  1.52596172  2.52872196  2.59899338
0.69107085  3.08322733
  0.48029669  2.12234776  1.14107039  2.73816159
2.83615695  2.01759308
  2.7116427   3.23402865  2.87439604  3.51199543  2.2259525
2.15305567
  2.47635157  2.74480414  2.18479692  3.14461992 -
0.89347977 -1.51961494
```

```
 -1.81823296  0.05512564  2.09192254 -0.5789596
0.92558177  2.28800128
  0.22876166 -0.80125027  1.99108531 -1.5523202
1.69124851 -0.69605139
  2.58036214  1.86768348 -0.83266108  0.40359327 -
1.43101297  1.27464161
  0.4101286   0.81320276  1.07017169 -0.44937791 -
2.51731021  0.86981672
  0.82262339 -0.76802261 -0.51110168 -1.07453318 -
0.50399535 -1.31124431
 -1.52773342 -1.90072666  0.79056143  0.9989761
2.54738234 -0.53118357
  1.07397322  2.29024085  1.45409066  0.8375519  -
0.51305733 -0.12723061
 -0.61536912  0.43497615 -1.73444494 -0.32427811 -1.5888709
0.123098
  1.61791755  1.45726543 -0.23890288 -1.27665922 -
0.41499111 -0.45065081
  0.54139419 -0.20498599 -0.0729984  -2.40448838 -
0.50815969  0.77987484
  1.36652399 -1.14506231 -0.42827684  1.0247241  -
0.05143996  0.07538255
 -1.55993845 -0.43841321 -1.755462   -1.32428406 -
2.37655439 -2.92579835
 -2.13716639 -2.35121439 -3.0531239  -3.89923965 -
3.92146674 -3.08235716
 -2.35895565 -2.764957   -2.27351064 -2.97458054 -
2.36193359 -2.20681631
 -2.61743092 -4.27191366 -3.57327927 -2.80423095 -
2.90607992 -2.33247599
 -2.55578663 -1.81375585 -2.76500454 -2.73013413 -
3.60242822 -2.89083274
 -3.38308624 -1.06059682 -1.61171861 -3.12911042 -
2.23928358 -2.83779448
 -2.59226184 -2.94879996 -3.52184398 -2.41581922 -
2.92351427 -2.18547501
 -2.38683089 -3.19522322 -3.67033271 -2.47138831 -
3.37288802 -2.60215457
 -2.69214577 -2.39839363 -3.21585159]

[ 2.23024297  2.53192196  3.75467731  1.0201307
3.04919938  2.45822831
  2.06160512  2.51844454  2.76797089  3.48916135
1.76638133  2.12870494
  3.46649467  4.31363172  2.30845048  2.16745547
1.90220844  3.54012997
  2.09274066  3.1319081   1.10804505  2.55760384
1.67255267  1.78792909
  1.00222687  1.79246479  1.25516785  2.20313645
2.27398683  2.51223477
  2.68519586  1.64501308  1.90224111  1.42130848
1.92515877  1.39514757
  1.14008674  1.52596172  2.52872196  2.59899338
0.69107085  3.08322733
  0.48029669  2.12234776  1.14107039  2.73816159
2.83615695  2.01759308
  2.7116427   3.23402865  2.87439604  3.51199543  2.2259525
2.15305567
```

```
  2.47635157  2.74480414  2.18479692  3.14461992 -
0.89347977 -1.51961494
 -1.81823296  0.05512564  2.09192254 -0.5789596
0.92558177  2.28800128
  0.22876166 -0.80125027  1.99108531 -1.5523202
1.69124851 -0.69605139
  2.58036214  1.86768348 -0.83266108  0.40359327 -
1.43101297  1.27464161
  0.4101286   0.81320276  1.07017169 -0.44937791 -
2.51731021  0.86981672
  0.82262339 -0.76802261 -0.51110168 -1.07453318 -
0.50399535 -1.31124431
 -1.52773342 -1.90072666  0.79056143  0.9989761
2.54738234 -0.53118357
  1.07397322  2.29024085  1.45409066  0.8375519  -
0.51305733 -0.12723061
 -0.61536912  0.43497615 -1.73444494 -0.32427811 -1.5888709
0.123098
  1.61791755  1.45726543 -0.23890288 -1.27665922 -
0.41499111 -0.45065081
  0.54139419 -0.20498599 -0.0729984  -2.40448838 -
0.50815969  0.77987484
  1.36652399 -1.14506231 -0.42827684  1.0247241  -
0.05143996  0.07538255
 -1.55993845 -0.43841321 -1.755462   -1.32428406 -
2.37655439 -2.92579835
 -2.13716639 -2.35121439 -3.0531239  -3.89923965 -
3.92146674 -3.08235716
 -2.35895565 -2.764957   -2.27351064 -2.97458054 -
2.36193359 -2.20681631
 -2.61743092 -4.27191366 -3.57327927 -2.80423095 -
2.90607992 -2.33247599
 -2.55578663 -1.81375585 -2.76500454 -2.73013413 -
3.60242822 -2.89083274
 -3.38308624 -1.06059682 -1.61171861 -3.12911042 -
2.23928358 -2.83779448
 -2.59226184 -2.94879996 -3.52184398 -2.41581922 -
2.92351427 -2.18547501
 -2.38683089 -3.19522322 -3.67033271 -2.47138831 -
3.37288802 -2.60215457
 -2.69214577 -2.39839363 -3.21585159]
[-0.48583464 -0.22157478 -0.31528188  0.01214349 -
0.30028828 -0.07054905
 -0.00173207 -0.02466918 -0.04144561 -0.52801878
0.27405069  0.16544914
 -0.3695384 ]

1.000000000000001
```

- Obtain the loadings for the second principal component

```
print(pca.components_[1])
print(np.sum(pca.components_[1]**2))
```

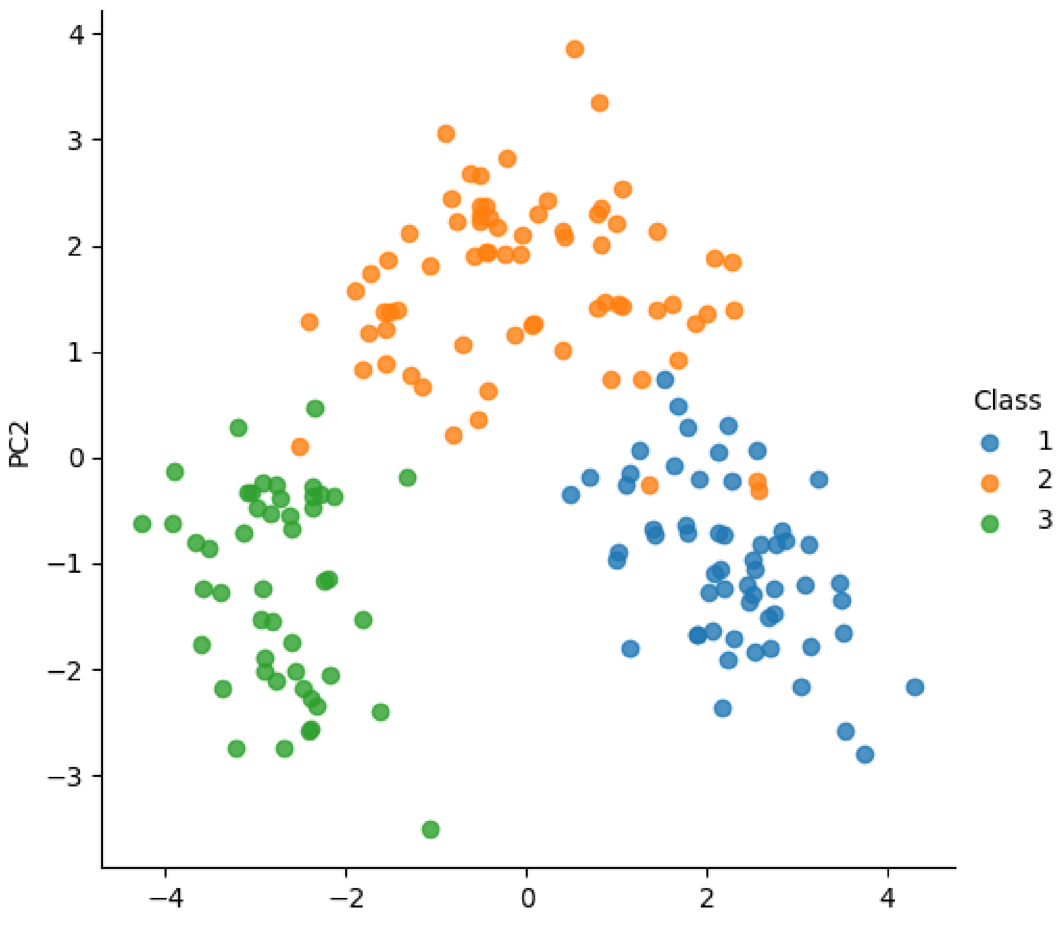
Output:

```
[-0.48583464 -0.22157478 -0.31528188  0.01214349 -
0.30028828 -0.07054905
 -0.00173207 -0.02466918 -0.04144561 -0.52801878
0.27405069  0.16544914
 -0.3695384 ]
1.0000000000000001
```

- Visualize scatterplots of the principal components

```
def pca_scatter(pca, standardised_values, classifs):
    foo = pca.transform(standardised_values)
    bar = pd.DataFrame(list(zip(foo[:, 0], foo[:, 1],
classifs)), columns=["PC1", "PC2", "Class"])
    sns.lmplot("PC1", "PC2", bar, hue="Class",
fit_reg=False)


pca_scatter(pca, standardisedX, y)
```



- Get mean, standard deviations, and sample sizes

```
print_mean_and_sd_by_group(standardisedX, y)
```

Output:

```
## Means:
          V2        V3        V4  ...       V12       V13
V14
V1                                ...
1   0.920878 -0.290306  0.327654  ...  0.461430  0.768532
1.183115
2  -0.886450 -0.364836 -0.442466  ...  0.434592  0.257511 -
0.718452
3   0.198479  0.890440  0.258566  ... -1.200396 -1.309545 -
0.366888

[3 rows x 13 columns]

## Standard deviations:
          V2        V3        V4  ...      V12       V13
V14
V1                                ...
1   0.567419  0.615865  0.828067  ...  0.50964  0.487140
0.705171
2   0.662305  0.903457  1.141944  ...  0.88190  0.701263
0.497145
3   0.650559  0.964490  0.666258  ...  0.49562  0.382960
0.362721

[3 rows x 13 columns]

## Sample sizes:
     0
V1
1   58
2   71
3   48
```