PRACTICE #03 – SCIKIT-LEARN WITH PCA & LDA

Multivariate Statistical Analysis – 19TGMT

Teacher:

- PhD. Lý Quốc Ngọc
- MS. Nguyễn Mạnh Hùng
- MS. Phạm Thanh Tùng

Student:

- Chung Kim Khánh (19127644)

Table of content

I. INFORMATION

STUDENT ID	FULLNAME	MAIL
19127644	Chung Kim Khánh	ckkhanh19@clc.fitus.edu.vn

II. DESCRIBE THE DATA INFORMATION

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).

1. Read data from the given sample CSV file

```python
# 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()
```

The output:

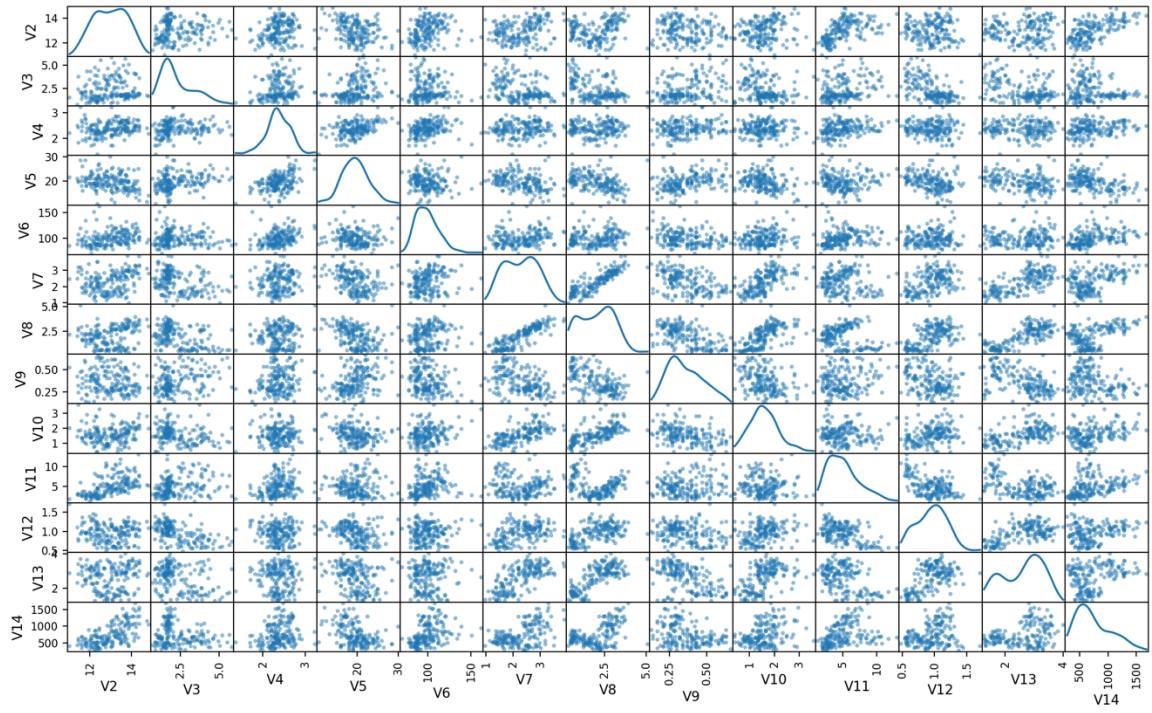
```
source /Users/chungkimkhanh/opt/anaconda3/bin/activate
conda activate base
(base) chungkimkhanh@Chungs-MacBook-Pro source % source /Users/chungkimkhanh/opt/anaconda3/bin/activate
(base) chungkimkhanh@Chungs-MacBook-Pro source % conda activate base
(base) chungkimkhanh@Chungs-MacBook-Pro source %  /usr/bin/env /Users/chungkimkhanh/opt/anaconda3/bin/python /Users/chungkimkhanh/.vscode/extensions/ms-python.python-2022.4.1/pythonFiles/lib/python/debugpy/launcher 54427 -- /Users/chungkimkhanh/Documents/PTDLNB/19127644_lab03/19127644_lab03/source/main.py
## Data:
     V1     V2    V3    V4    V5   V6    V7    V8    V9   V10    V11   V12   V13   V14
0     1  13.20  1.78  2.14  11.2  100  2.65  2.76  0.26  1.28   4.38  1.05  3.40  1050
1     1  13.16  2.36  2.67  18.6  101  2.80  3.24  0.30  2.81   5.68  1.03  3.17  1185
2     1  14.37  1.95  2.50  16.8  113  3.85  3.49  0.24  2.18   7.80  0.86  3.45  1480
3     1  13.24  2.59  2.87  21.0  118  2.80  2.69  0.39  1.82   4.32  1.04  2.93   735
4     1  14.20  1.76  2.45  15.2  112  3.27  3.39  0.34  1.97   6.75  1.05  2.85  1450
..   ..    ...   ...   ...   ...  ...   ...   ...   ...   ...    ...   ...   ...   ...
172   3  13.71  5.65  2.45  20.5   95  1.68  0.61  0.52  1.06   7.70  0.64  1.74   740
173   3  13.40  3.91  2.48  23.0  102  1.80  0.75  0.43  1.41   7.30  0.70  1.56   750
174   3  13.27  4.28  2.26  20.0  120  1.59  0.69  0.43  1.35  10.20  0.59  1.56   835
175   3  13.17  2.59  2.37  20.0  120  1.65  0.68  0.53  1.46   9.30  0.60  1.62   840
176   3  14.13  4.10  2.74  24.5   96  2.05  0.76  0.56  1.35   9.20  0.61  1.60   560

[177 rows x 14 columns]
## Head:
   V1     V2    V3    V4    V5   V6    V7    V8    V9   V10   V11   V12   V13   V14
0   1  13.20  1.78  2.14  11.2  100  2.65  2.76  0.26  1.28  4.38  1.05  3.40  1050
1   1  13.16  2.36  2.67  18.6  101  2.80  3.24  0.30  2.81  5.68  1.03  3.17  1185
2   1  14.37  1.95  2.50  16.8  113  3.85  3.49  0.24  2.18  7.80  0.86  3.45  1480
3   1  13.24  2.59  2.87  21.0  118  2.80  2.69  0.39  1.82  4.32  1.04  2.93   735
4   1  14.20  1.76  2.45  15.2  112  3.27  3.39  0.34  1.97  6.75  1.05  2.85  1450
## Tail:
     V1     V2    V3    V4    V5   V6    V7    V8    V9   V10   V11   V12   V13  V14
172   3  13.71  5.65  2.45  20.5   95  1.68  0.61  0.52  1.06   7.7  0.64  1.74  740
173   3  13.40  3.91  2.48  23.0  102  1.80  0.75  0.43  1.41   7.3  0.70  1.56  750
174   3  13.27  4.28  2.26  20.0  120  1.59  0.69  0.43  1.35  10.2  0.59  1.56  835
175   3  13.17  2.59  2.37  20.0  120  1.65  0.68  0.53  1.46   9.3  0.60  1.62  840
176   3  14.13  4.10  2.74  24.5   96  2.05  0.76  0.56  1.35   9.2  0.61  1.60  560
```

```
## 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)
```

2. Describe the data information

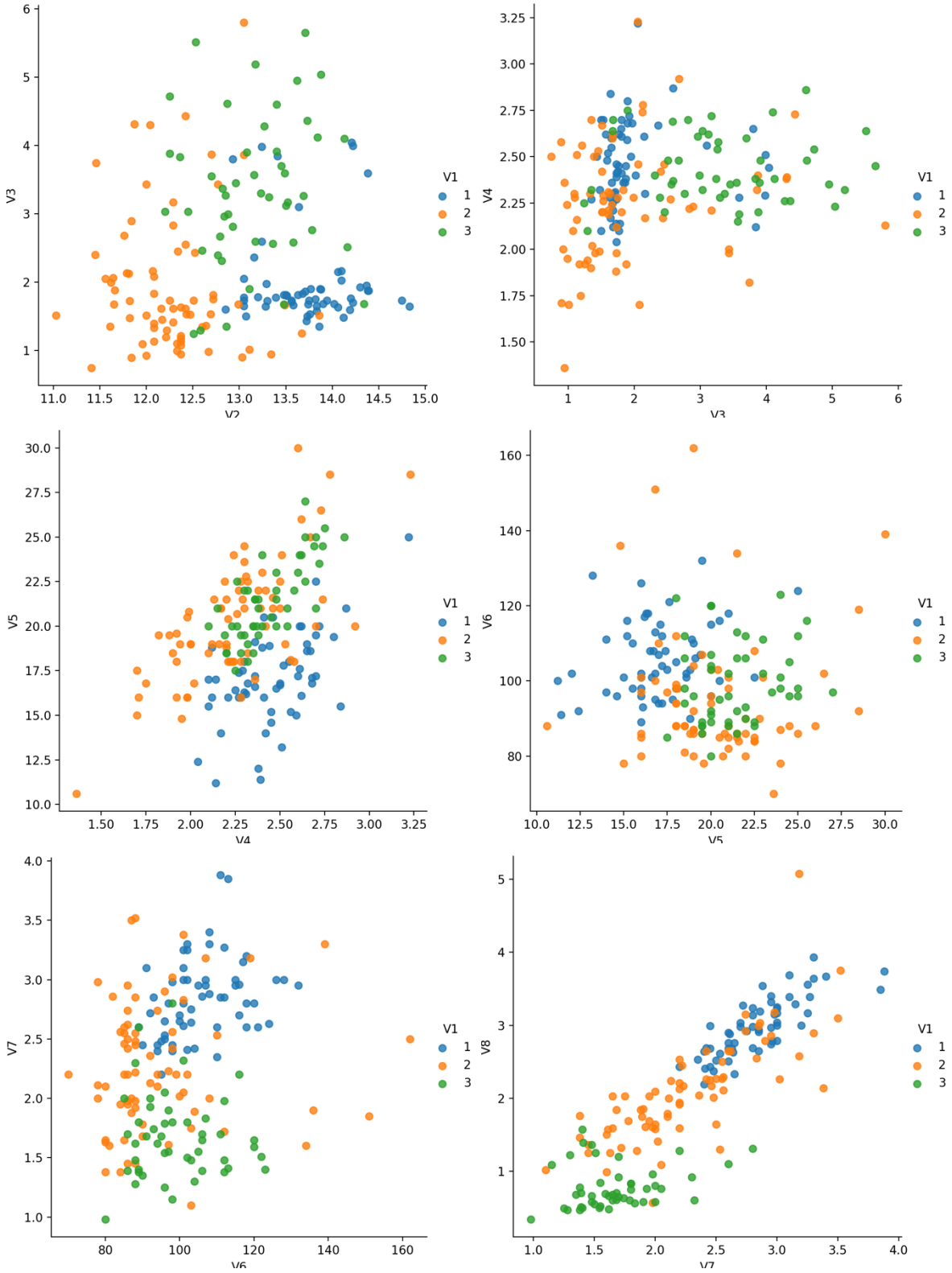
Plot data for visualization

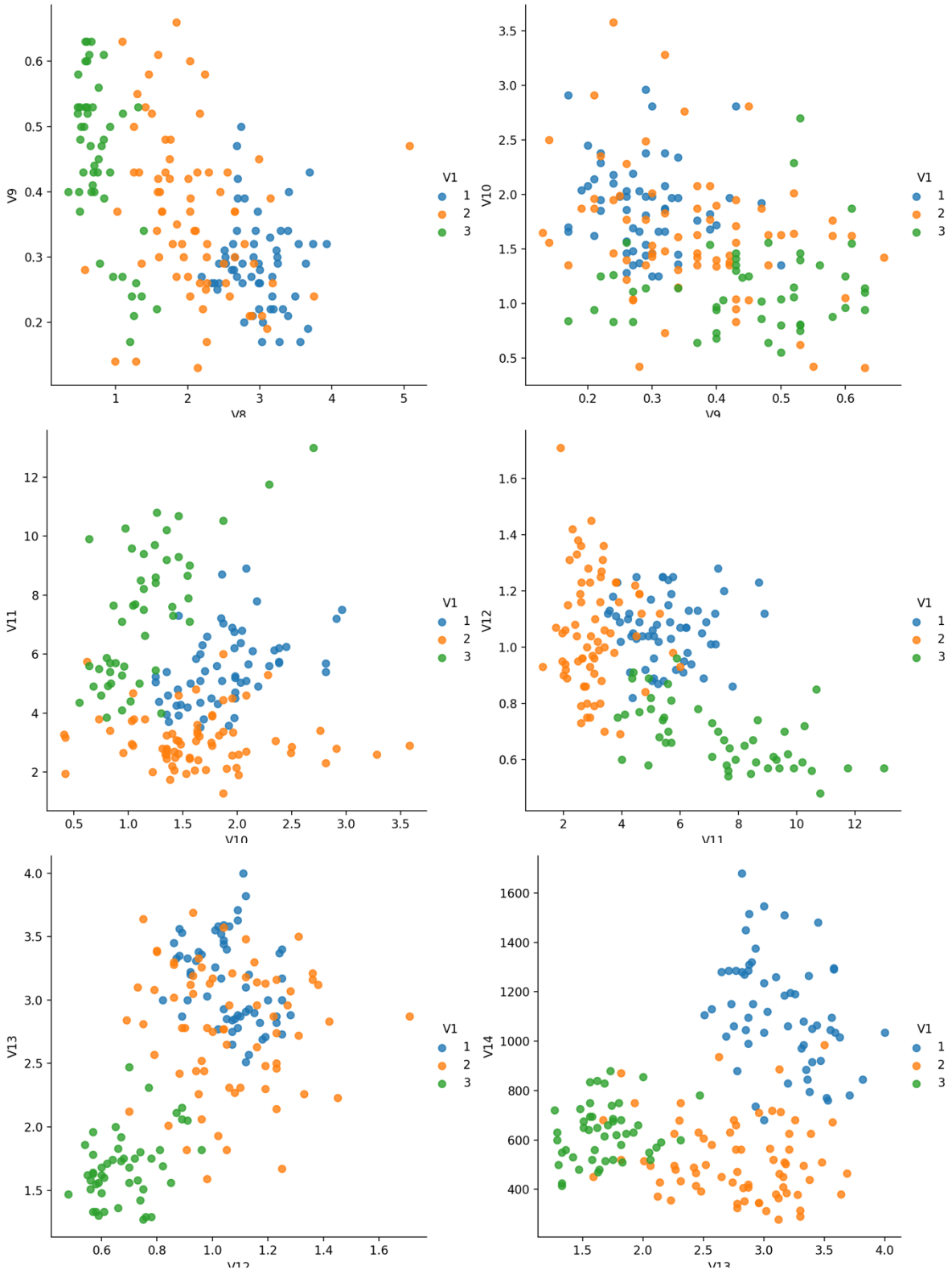
```
38    # plotting multivariate data
39    pd.plotting.scatter_matrix(data.loc[:, "V2":"V14"], diagonal="kde", figsize=(20, 15))
40    plt.show()
```



Scatterplot with the data points labelled by their Group

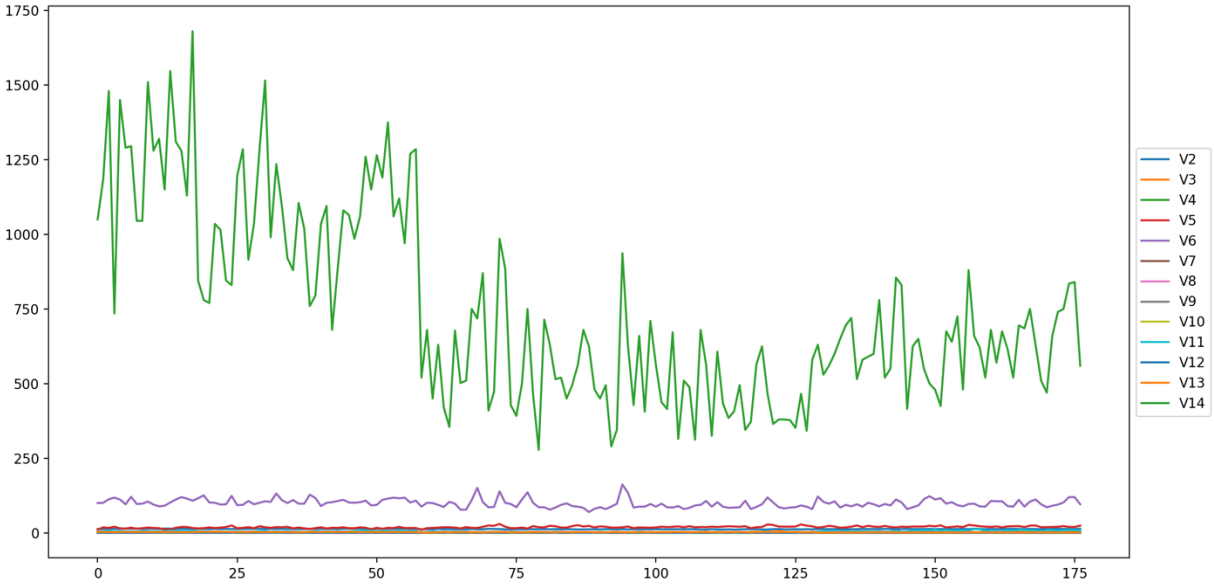
```
41  for i in range(2, 14):
42      sns.lmplot(x="V" + str(i), y="V" + str(i + 1), data=data, hue="V1", fit_reg=False)
```

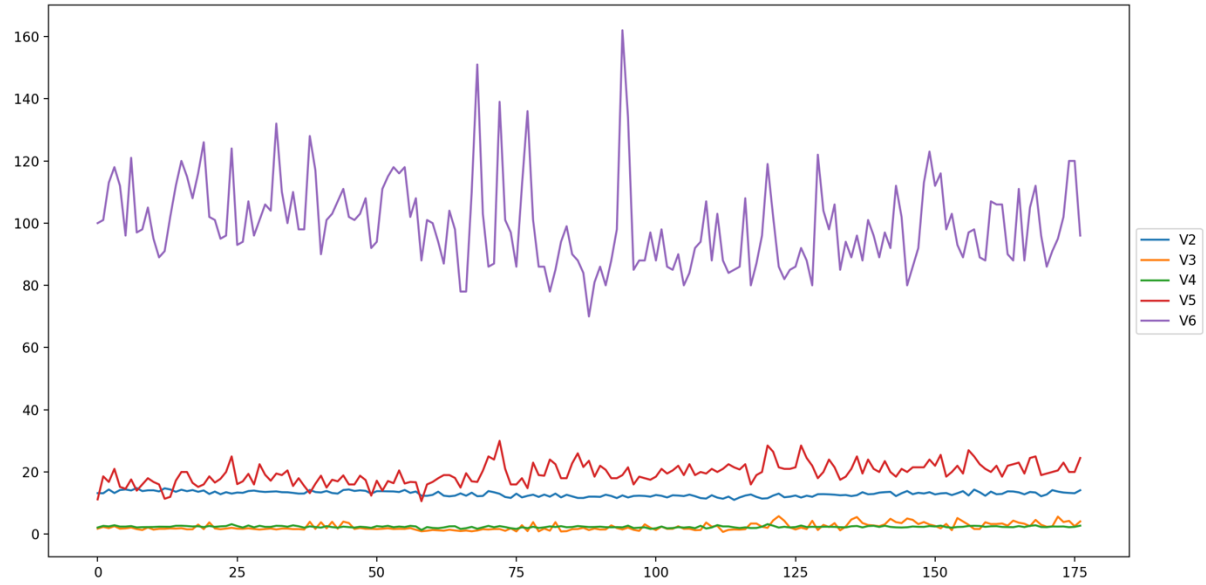


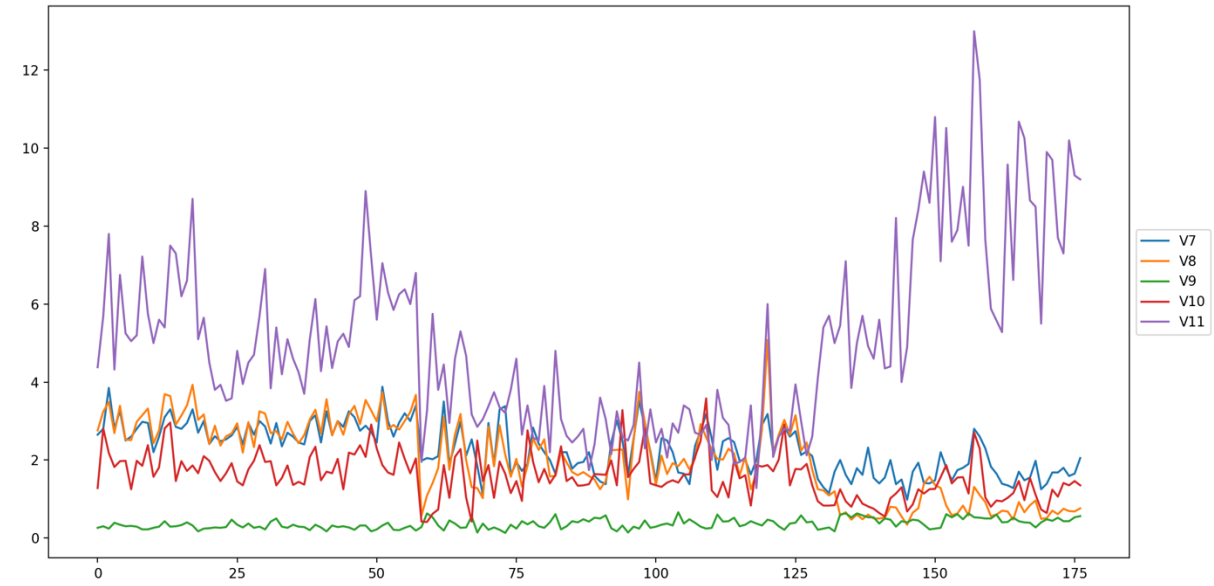


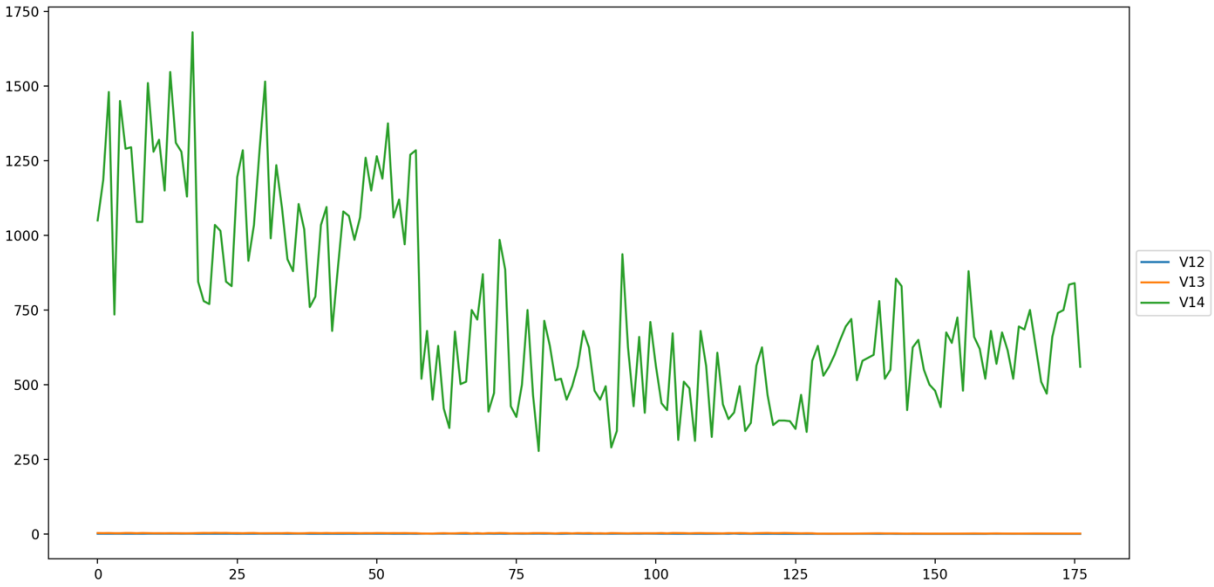
Profile plot, used to shows the variation in each of the variables, by plotting the value of each of the variables for each of the samples

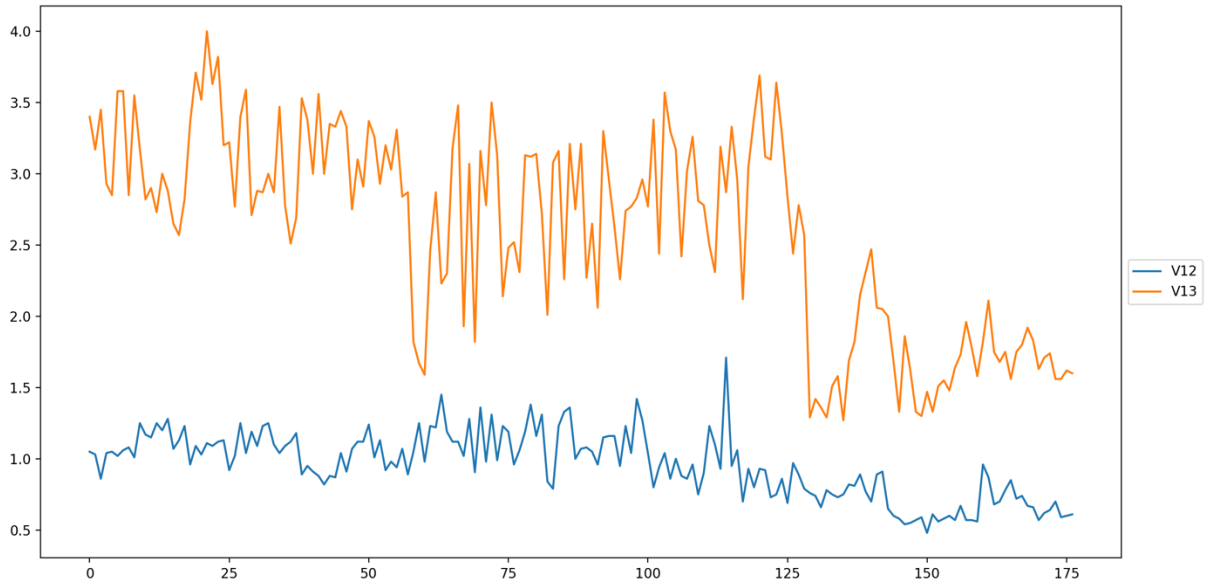
```
45    ax = data[["V2", "V3", "V4", "V5", "V6", "V7", "V8", "V9", "V10", "V11", "V12", "V13", "V14"]].plot(figsize=(20, 15))
46    ax.legend(loc='center left', bbox_to_anchor=(1, 0.5))
```


1750
1500
1250
1000
750
500
250
0
0
25
50
75
100
125
150
175
V2
V3
V4
V5
V6
V7
V8
V9
V10
V11
V12
V13
V14


160
140
120
100
80
60
40
20
0
0
25
50
75
100
125
150
175
V2
V3
V4
V5
V6


V7
V8
V9
V10
V11


V12
V13
V14



III. SOME BASIC MULTIVARIATE ANALYSIS WITH VISUALIZATION

Calculating summary statistics for multivariate data

```
61  # calculating summary statistics for multivariate data
62  print(X.apply(np.mean))
63  print(X.apply(np.std))
64  print(X.apply(np.max))
65  print(X.apply(np.min))
```

The 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
```

- Min

```
V2       11.03
V3        0.74
V4        1.36
V5       10.60
V6       70.00
V7        0.98
V8        0.34
V9        0.13
V10       0.41
V11       1.28
V12       0.48
V13       1.27
V14     278.00
```

Means and variances per group

```
68  # means and variances per group
69  def print_mean_and_sd_by_group(variables, group_variable):
70      data_group_by = variables.groupby(group_variable)
71
72      print("## Means:")
73      print(data_group_by.apply(np.mean))
74
75      print("\n## Standard deviations:")
76      print(data_group_by.apply(np.std))
77
78      print("\n## Sample sizes:")
79      print(pd.DataFrame(data_group_by.apply(len)))
80
81
82  print_mean_and_sd_by_group(X, y)
83
```

The output:

```
## Means:
          V2        V3        V4         V5  ...       V11       V12       V13          V14
V1                                           ...
1   13.736379  2.015862  2.456034  17.062069  ...  5.526379  1.062414  3.144655  1116.586207
2   12.278732  1.932676  2.244789  20.238028  ...  3.086620  1.056282  2.785352   519.507042
3   13.153750  3.333750  2.437083  21.416667  ...  7.396250  0.682708  1.683542   629.895833

[3 rows x 13 columns]

## Standard deviations:
          V2        V3        V4        V5         V6  ...       V10       V11       V12       V13          V14
V1                                                     ...
1   0.457635  0.687396  0.227141  2.539198  10.136128  ...  0.408849  1.238484  0.116446  0.342512  221.418938
2   0.534162  1.008391  0.313238  3.326097  16.635097  ...  0.597813  0.918393  0.201503  0.493064  156.100173
3   0.524689  1.076514  0.182756  2.234515  10.776433  ...  0.404555  2.286743  0.113243  0.269262  113.891805

[3 rows x 13 columns]

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

IV. PCA & LDA

Standardising variables

```
264   # standardising variables
265   standardisedX = scale(X)
266   standardisedX = pd.DataFrame(standardisedX, index=X.index, columns=X.columns)
267
268   print(standardisedX.apply(np.mean))
269   print(standardisedX.apply(np.std))
```

The output:

```
V2    -2.609338e-16
V3     4.252719e-16
V4    -4.378168e-16
V5    -6.410440e-16
V6    -1.028681e-16
V7    -1.279579e-16
V8     1.505387e-16
V9    -5.595022e-16
V10    5.645202e-17
V11    1.568112e-16
V12    6.310081e-16
V13    6.335171e-16
V14    2.389802e-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
```

1. PCA

This method is called Principal Component Analysis (PCA). This method is based on the observation that the data are not normally distributed randomly in space but are often distributed near certain special lines/surfaces. PCA considers a special case where such special faces have linear form as subspaces. PCA is an unsupervised learning method, i.e. it only uses vectors describing the data and not the labels, if any, of the data.

Apply PCA using sklearn

```
272    # principal component analysis
273    pca = PCA().fit(standardisedX)
```

Check the summary of PCA results

```
276  def pca_summary(pca, standardised_data, out=True):
277      names = ["PC"+str(i) for i in range(1, len(pca.explained_variance_ratio_)+1)]
278      a = list(np.std(pca.transform(standardised_data), axis=0))
279      b = list(pca.explained_variance_ratio_)
280      c = [np.sum(pca.explained_variance_ratio_[:i]) for i in range(1, len(pca.explained_variance_ratio_)+1)]
281      columns = pd.MultiIndex.from_tuples([("sdev", "Standard deviation"), ("varprop", "Proportion of Variance"),
282      ("cumprop", "Cumulative Proportion")])
283      summary = pd.DataFrame(list(zip(a, b, c)), index=names, columns=columns)
284      if out:
285          print("Importance of components:")
286          print(summary)
287      return summary
288
289
290  summary = pca_summary(pca, standardisedX)
291  print(summary.sdev)
292  print(np.sum(summary.sdev**2))
```

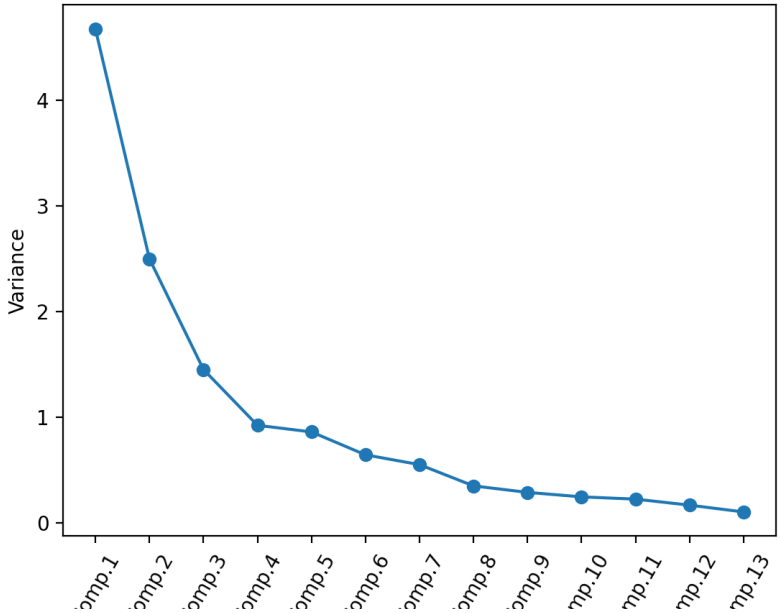
The output:

```
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
dtype: float64
```

Check how many principal components to retain

```
294   # how many principal components to retain
295   def scree_plot(pca, standardised_values):
296       y = np.std(pca.transform(standardised_values), axis=0)**2
297       x = np.arange(len(y)) + 1
298       plt.plot(x, y, "o-")
299       plt.xticks(x, ["Comp."+str(i) for i in x], rotation=60)
300       plt.ylabel("Variance")
301       plt.show()
302
303
304   plt.figure()
305   scree_plot(pca, standardisedX)
```

The output:



Calculate the values of the first principal component

```
# loadings for the principal components
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])

```

The 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]
```

Obtain the loadings for the second principal component

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

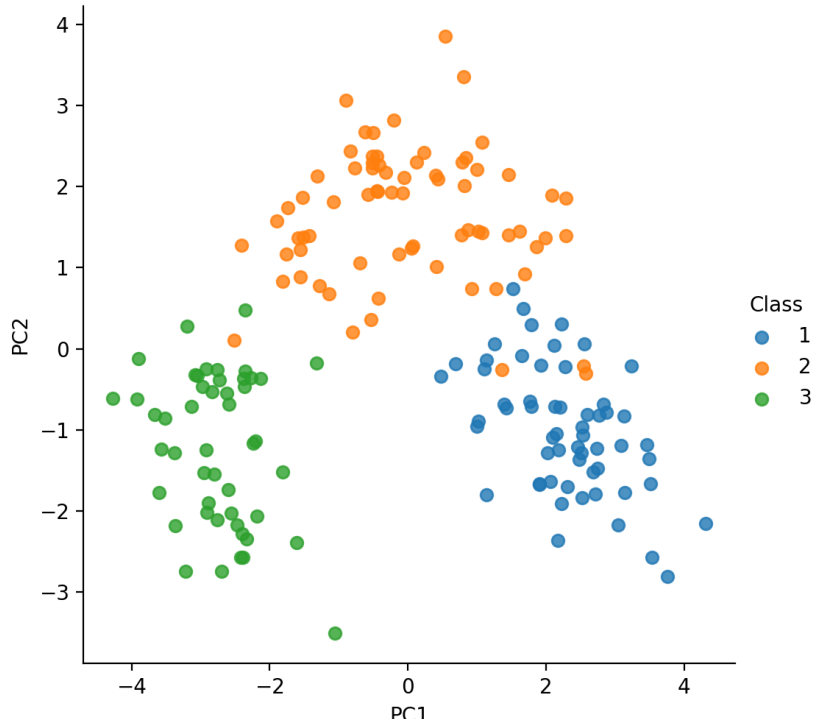
The output:

```
[-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 ]
1.0000000000000002
```

Visualize scatterplots of the principal components

```
337  # scatter plots of the principal components
338  def pca_scatter(pca, standardised_values, classifs):
339      foo = pca.transform(standardised_values)
340      bar = pd.DataFrame(list(zip(foo[:, 0], foo[:, 1], classifs)), columns=["PC1", "PC2", "Class"])
341      sns.lmplot(x="PC1", y="PC2", data=bar, hue="Class", fit_reg=False)
342
343
344  pca_scatter(pca, standardisedX, y)
```

The output:



Get mean, standard deviations, and sample sizes

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

The output:

```
## Means:
          V2        V3        V4        V5        V6  ...       V10       V11       V12       V13       V14
V1                                                    ...
1   0.920878 -0.290306  0.327654 -0.737947  0.452471  ...  0.536273  0.203453  0.461430  0.768532  1.183115
2  -0.886450 -0.364836 -0.442466  0.216759 -0.356467  ...  0.076032 -0.849134  0.434592  0.257511 -0.718452
3   0.198479  0.890440  0.258566  0.571063 -0.019462  ... -0.760460  1.010172 -1.200396 -1.309545 -0.366888

[3 rows x 13 columns]

## Standard deviations:
          V2        V3        V4        V5        V6  ...       V10       V11       V12       V13       V14
V1                                                    ...
1   0.567419  0.615865  0.828067  0.763294  0.717149  ...  0.717369  0.534320  0.50964  0.487140  0.705171
2   0.662305  0.903457  1.141944  0.999839  1.176963  ...  1.048927  0.396222  0.88190  0.701263  0.497145
3   0.650559  0.964490  0.666258  0.671704  0.762452  ...  0.709835  0.986571  0.49562  0.382960  0.362721

[3 rows x 13 columns]

## Sample sizes:
     0
V1
1   58
2   71
3   48
(base) chungkimkhanh@Chungs-MacBook-Pro source %
```

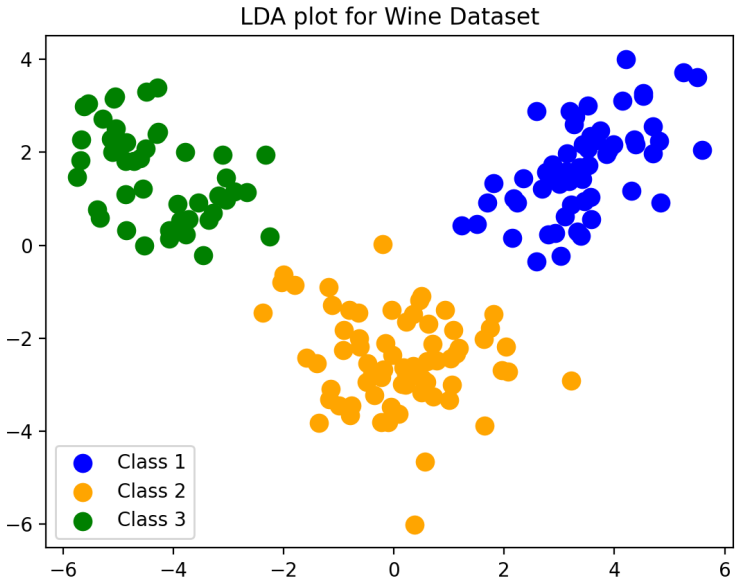
2. LDA

LDA is a data dimensionality reduction method for classification problems. LDA can be thought of as a method of dimensionality reduction, or it can be thought of as a classification method, or it can be applied simultaneously to both (i.e., reducing the data dimensionality so that the most efficient classification).

Apply LDA using sklearn (“main_LDA_2.py”)

```python
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
from sklearn import datasets
import matplotlib.pyplot as plt


# Loading Wine Dataset
wine = datasets.load_wine()

X = wine.data
y = wine.target
target_names = wine.target_names

# fitting the LDA model
lda = LDA(n_components=2)
lda_X = lda.fit(X,y).transform(X)

plt.scatter(lda_X[y == 0, 0], lda_X[y == 0, 1], s =80, c = 'blue', label = 'Class 1')
plt.scatter(lda_X[y == 1, 0], lda_X[y == 1, 1], s =80,  c = 'orange', label = 'Class 2')
plt.scatter(lda_X[y == 2, 0], lda_X[y == 2, 1], s =80,  c = 'green', label = 'Class 3')
plt.title('LDA plot for Wine Dataset')
plt.legend()
plt.show()
```

The output:



Evaluation with Confusion matrix “main_LDA_1.py”

The output:

```
Original & LDA
[[14  0  0]
 [ 0 15  0]
 [ 0  1  6]]

actual / without LDA
[[14  0  0]
 [ 0 15  0]
 [ 0  0  7]]

actual / with LDA
[[14  0  0]
 [ 0 15  0]
 [ 0  1  6]]
```

V. REFERENCE

[1] https://machinelearningcoban.com/2017/06/15/pca/
[2] https://machinelearningcoban.com/2017/06/30/lda/
[3] https://github.com/Ayantika22/Linear-discriminant-Analysis-LDA-for-Wine-Dataset
[4] https://scikit-learn.org/stable/auto_examples/decomposition/plot_pca_vs_lda.html?fbclid=IwAR2YF0CQwa3DKx0GtolPwulVjoZwmcnxyRBuvTIR2dNNa7kFN6o5CtprhFg