Principal Component Analysis Technique

PCA is used for reducing the dimensionality of such datasets, increasing interpretability but at the same time minimizing information loss.

In [1]:

```
#Importing the libraries "pandas, numpy, matplotlib.pyplot and seaborn" to my python scri
pt
#with the standard short name as "pd, np, plt and sns".

#Uploading the file on google colab and choosing the selected dataset by clicking "choose
files".

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from google.colab import files
uploaded = files.upload()
```

Choose File **No file selected**

Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving New_Data6.csv to New_Data6.csv
```

Loading Data

In [2]:

```
#In the first step, importing the dataset in the project by the "read_csv" function
#and that reads the data into a pandas dataframe object.

dframe = pd.read_csv('New_Data6.csv')
```

In [3]:

```
#head() method returns a particular rows from the top and where did not mention
#the number below hence returns first 5 rows

dframe.head()
```

Out[3]:

	Alcohol	Malic_Acid	Ash	Ash_Alcanity	Magnesium	Total_Phenols	Flavanoids	Nonflavanoid_Phenols	Proanthocyanins	Co
0	14.23	1.71	2.43	15.6	127	2.80	3.06	0.28	2.29	
1	13.20	1.78	2.14	11.2	100	2.65	2.76	0.26	1.28	
2	13.16	2.36	2.67	18.6	101	2.80	3.24	0.30	2.81	
3	14.37	1.95	2.50	16.8	113	3.85	3.49	0.24	2.18	
4	13.24	2.59	2.87	21.0	118	2.80	2.69	0.39	1.82	

In [4]:

```
#The info() method gives the information about the dataframe where the information contai
ns;
#number of columns
#column labels
#column data types
```

```
#memory usage
#range index
#number of cells in each column

dframe.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 178 entries, 0 to 177
Data columns (total 14 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   Alcohol               178 non-null    float64
 1   Malic_Acid            178 non-null    float64
 2   Ash                   178 non-null    float64
 3   Ash_Alcanity          178 non-null    float64
 4   Magnesium             178 non-null    int64  
 5   Total_Phenols         178 non-null    float64
 6   Flavanoids            178 non-null    float64
 7   Nonflavanoid_Phenols  178 non-null    float64
 8   Proanthocyanins       178 non-null    float64
 9   Color_Intensity       178 non-null    float64
 10  Hue                   178 non-null    float64
 11  OD280                 178 non-null    float64
 12  Proline               178 non-null    int64  
 13  Customer_Segment      178 non-null    int64  
dtypes: float64(11), int64(3)
memory usage: 19.6 KB
```

Checking Null values

In [5]:

```
#.isnull().sum() returns the number of missing values in the dataset.

dframe.isnull().sum()
```

Out[5]:

```
Alcohol                 0
Malic_Acid              0
Ash                     0
Ash_Alcanity            0
Magnesium               0
Total_Phenols           0
Flavanoids              0
Nonflavanoid_Phenols    0
Proanthocyanins         0
Color_Intensity         0
Hue                     0
OD280                   0
Proline                 0
Customer_Segment        0
dtype: int64
```

In [6]:

```
#The describe() method gives description of the data in the dataframe
#The details are included in the description for each column if the dataframe includes nu
merical data.

dframe.describe()
```

Out[6]:

	Alcohol	Malic_Acid	Ash	Ash_Alcanity	Magnesium	Total_Phenols	Flavanoids	Nonflavanoid_Phenols	Proar
count	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	
mean	13.000618	2.336348	2.366517	19.494944	99.741573	2.295112	2.029270	0.361854	
std	0.811827	1.117146	0.274344	3.339564	14.282484	0.625851	0.998859	0.124453	
min	11.030000	0.740000	1.360000	10.600000	70.000000	0.980000	0.340000	0.130000	

	Alcohol	Malic_Acid	Ash	Ash_Alcanity	Magnesium	Total_Phenols	Flavanoids	Nonflavanoid_Phenols	Proar
25%	12.362500	1.602500	2.210000	17.200000	88.000000	1.742500	1.205000	0.270000	
50%	13.050000	1.865000	2.360000	19.500000	98.000000	2.355000	2.135000	0.340000	
75%	13.677500	3.082500	2.557500	21.500000	107.000000	2.800000	2.875000	0.437500	
max	14.830000	5.800000	3.230000	30.000000	162.000000	3.880000	5.080000	0.660000	

In [7]:

```
#The .drop() function is used to drop and dropping the customer_segment.

#head() method returns a particular rows from the top and where did not mention
#the number below hence returns first 5 rows after droping the customer_segment.

df = dframe.drop(["Customer_Segment"], axis=1)
df.head()
```

Out[7]:

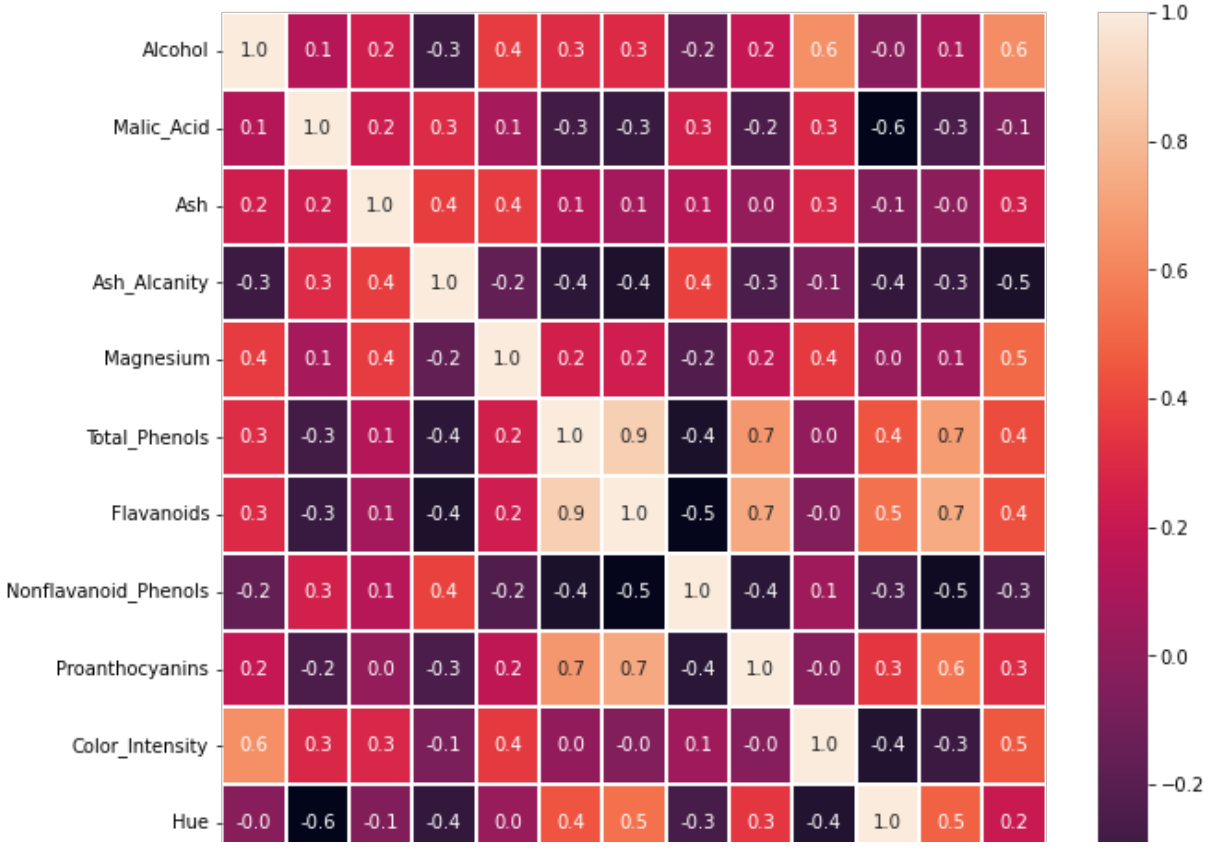
	Alcohol	Malic_Acid	Ash	Ash_Alcanity	Magnesium	Total_Phenols	Flavanoids	Nonflavanoid_Phenols	Proanthocyanins	Co
0	14.23	1.71	2.43	15.6	127	2.80	3.06	0.28	2.29	
1	13.20	1.78	2.14	11.2	100	2.65	2.76	0.26	1.28	
2	13.16	2.36	2.67	18.6	101	2.80	3.24	0.30	2.81	
3	14.37	1.95	2.50	16.8	113	3.85	3.49	0.24	2.18	
4	13.24	2.59	2.87	21.0	118	2.80	2.69	0.39	1.82	

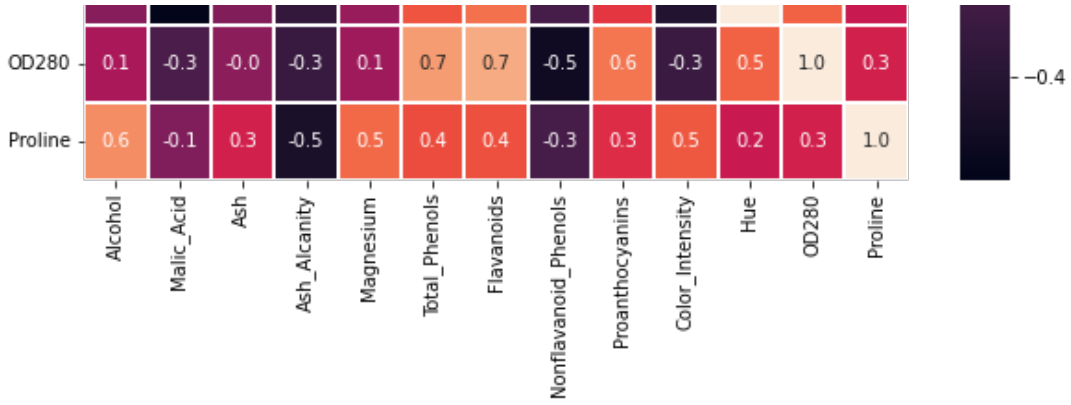
Correlation matrix

In [8]:

```
#The .heatma() function is a great way to visualize data, because it can show
#the relation between variabels including time.

f,ax = plt.subplots(figsize=(10,10))
sns.heatmap(df.corr(method='spearman'),annot=True,fmt=".1f",linewidths=1,ax=ax)
plt.show()
```





In [9]:

```
#The .corr() function is used to find the pairwise correlation of all columns
#in the Dataframe.

corr = df.corr(method='spearman')
th = 0.6
corr[corr > th]
```

Out[9]:

	Alcohol	Malic_Acid	Ash	Ash_Alcanity	Magnesium	Total_Phenols	Flavanoids	Nonflavanoid_Phenols
Alcohol	1.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Malic_Acid	NaN	1.0	NaN	NaN	NaN	NaN	NaN	NaN
Ash	NaN	NaN	1.0	NaN	NaN	NaN	NaN	NaN
Ash_Alcanity	NaN	NaN	NaN	1.0	NaN	NaN	NaN	NaN
Magnesium	NaN	NaN	NaN	NaN	1.0	NaN	NaN	NaN
Total_Phenols	NaN	NaN	NaN	NaN	NaN	1.000000	0.879404	NaN
Flavanoids	NaN	NaN	NaN	NaN	NaN	0.879404	1.000000	NaN
Nonflavanoid_Phenols	NaN	NaN	NaN	NaN	NaN	NaN	NaN	1.0
Proanthocyanins	NaN	NaN	NaN	NaN	NaN	0.666689	0.730322	NaN
Color_Intensity	0.635425	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Hue	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
OD280	NaN	NaN	NaN	NaN	NaN	0.687207	0.741533	NaN
Proline	0.633580	NaN	NaN	NaN	NaN	NaN	NaN	NaN

After this preliminary research, we note the following:

A strong linear relationship exists between Total_Phenols, Flavanoids, Proanthocyanins and OD280. There is a high linear realtion between Alcohol, Color_Intensity and Proline. Now we can exclude some of those factors to prevent redundant data from tainting the outcomes of our study.

In [10]:

```
#Here dropping the columns Flavanoids, Proanthocyanins, Color_Intensity, OD280 and Prolin
e using .drop()

df = dframe.drop(["Flavanoids","Proanthocyanins","Color_Intensity","OD280","Proline"], a
xis=1)
df.head()
```

Out[10]:

	Alcohol	Malic_Acid	Ash	Ash_Alcanity	Magnesium	Total_Phenols	Nonflavanoid_Phenols	Hue	Customer_Segment
0	14.23	1.71	2.43	15.6	127	2.80	0.28	1.04	1

	Alcohol	Malic_Acid	Ash	Ash_Alcanity	Magnesium	Total_Phenols	Nonflavanoid_Phenols	Hue	Customer_Segment
1	13.20	1.78	2.14	11.2	100	2.65	0.26	1.05	1
2	13.16	2.36	2.67	18.6	101	2.80	0.30	1.03	1
3	14.37	1.95	2.50	16.8	113	3.85	0.24	0.86	1
4	13.24	2.59	2.87	21.0	118	2.80	0.39	1.04	1

PCA Technique

In [11]:

```
#Here the same dataset with different name by the "read_csv" function
#and that reads the data into a pandas dataframe object.

df2 = pd.read_csv('New_Data6.csv')
df2.head()
```

Out[11]:

	Alcohol	Malic_Acid	Ash	Ash_Alcanity	Magnesium	Total_Phenols	Flavanoids	Nonflavanoid_Phenols	Proanthocyanins	Co
0	14.23	1.71	2.43	15.6	127	2.80	3.06	0.28	2.29	
1	13.20	1.78	2.14	11.2	100	2.65	2.76	0.26	1.28	
2	13.16	2.36	2.67	18.6	101	2.80	3.24	0.30	2.81	
3	14.37	1.95	2.50	16.8	113	3.85	3.49	0.24	2.18	
4	13.24	2.59	2.87	21.0	118	2.80	2.69	0.39	1.82	

In [12]:

```
#Here dropping the customer_segment with .drop() function.

df2 = df2.drop(["Customer_Segment"], axis=1)
df2.describe()
```

Out[12]:

	Alcohol	Malic_Acid	Ash	Ash_Alcanity	Magnesium	Total_Phenols	Flavanoids	Nonflavanoid_Phenols	Proar
count	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	
mean	13.000618	2.336348	2.366517	19.494944	99.741573	2.295112	2.029270	0.361854	
std	0.811827	1.117146	0.274344	3.339564	14.282484	0.625851	0.998859	0.124453	
min	11.030000	0.740000	1.360000	10.600000	70.000000	0.980000	0.340000	0.130000	
25%	12.362500	1.602500	2.210000	17.200000	88.000000	1.742500	1.205000	0.270000	
50%	13.050000	1.865000	2.360000	19.500000	98.000000	2.355000	2.135000	0.340000	
75%	13.677500	3.082500	2.557500	21.500000	107.000000	2.800000	2.875000	0.437500	
max	14.830000	5.800000	3.230000	30.000000	162.000000	3.880000	5.080000	0.660000	

In [13]:

```
#Rows and columns can be choosen by position or index using .iloc.
#It displays an index error if the position or index is invalid.

X = df2.iloc[:, 0:len(df2.columns)-1].values
y = df2.iloc[:, len(df2.columns)-1].values
```

Train/Test Split

In [14]:

```
#Importing train_test_split function to split arrays or matrices into
```

```
#random subsets for train and test data.

#Splitting the data using train_test_split.

#After that can check the validation of the model by fitting the training data and
#predicting using the test or validation by applying the technique.

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25,random_state
= 0)
```

Feature Scaling

In [15]:

```
#Importing standardscaler from sklearn.preprocessing.

#It removes the mean and scales each feature or variable to unit variance.

#The .fit_transform is used on the training data so that we can scale the
#training data and also learn the scaling parameters of that data.

from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

PCA

In [16]:

```
#Importing PCA from sklearn.decomposition for the model.

from sklearn.decomposition import PCA
pca = PCA(n_components = None)
```

In [17]:

```
#To train the data and also learn the scaling parameters of that data.

X_train_pca = pca.fit_transform(X_train)
X_test_pca = pca.transform(X_test)
```

In [18]:

```
#The .explained_variance_ratio_ is used to get the ration of variance.

explained_variance = pca.explained_variance_ratio_
print("Variance Explained for each PC")
print(explained_variance)
var_exp = np.round(np.sum(explained_variance[0:5]),4)
print("With 5 PC it would explain the {var}% of the variance".format(var=var_exp*100))
```

```
Variance Explained for each PC
[0.37568924 0.17871865 0.11472285 0.08232531 0.06449993 0.05301091
 0.04437137 0.02667586 0.02154598 0.01718258 0.0138095  0.00744781]
With 5 PC it would explain the 81.6% of the variance
```