# Practical-1

**Create Comma Separate Files (CSV), Load CSV files into internal Data Structure.**

**What is a CSV?**

CSV (Comma Separated Values) is a simple file format used to store tabular data, such as a spreadsheet or database. A CSV file stores tabular data (numbers and text) in plain text. Each line of the file is a data record. Each record consists of one or more fields, separated by commas. The use of the comma as a field separator is the source of the name for this file format. For working CSV files in python, there is an inbuilt module called csv.

**How to create a CSV file?**

Step 1: Open the Anaconda Navigator.

Step 2: Launch the Jupyter Notebook.

Step 3: Clink on ‘New’ button and select ‘Text File’.

Step 4: Input the data and separating the values by using comma. For example

Name, Class, Dorm, Room, GPA

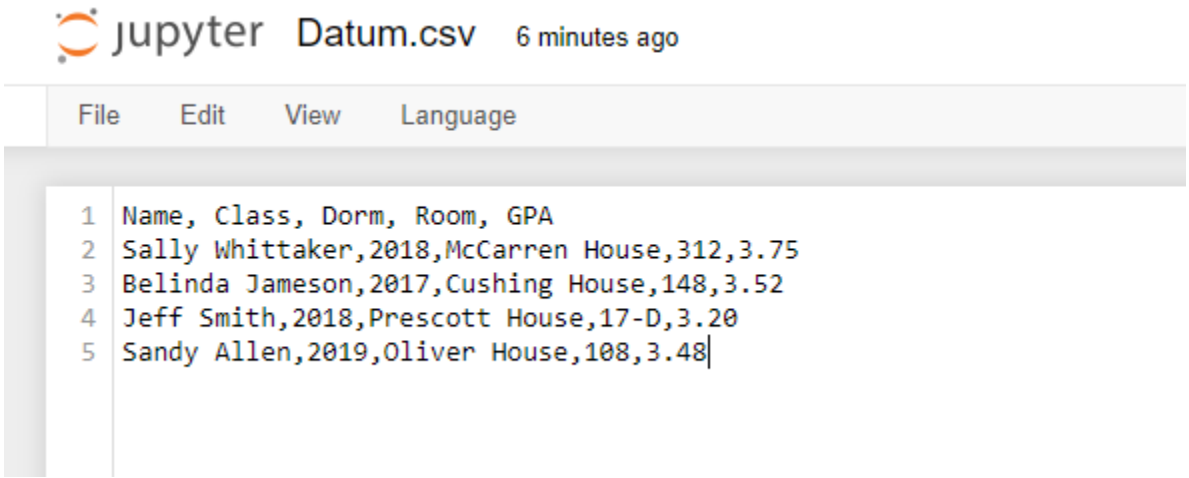
Sally Whittaker,2018,McCarren House,312,3.75

Belinda Jameson,2017,Cushing House,148,3.52

Jeff Smith,2018,Prescott House,17-D,3.20

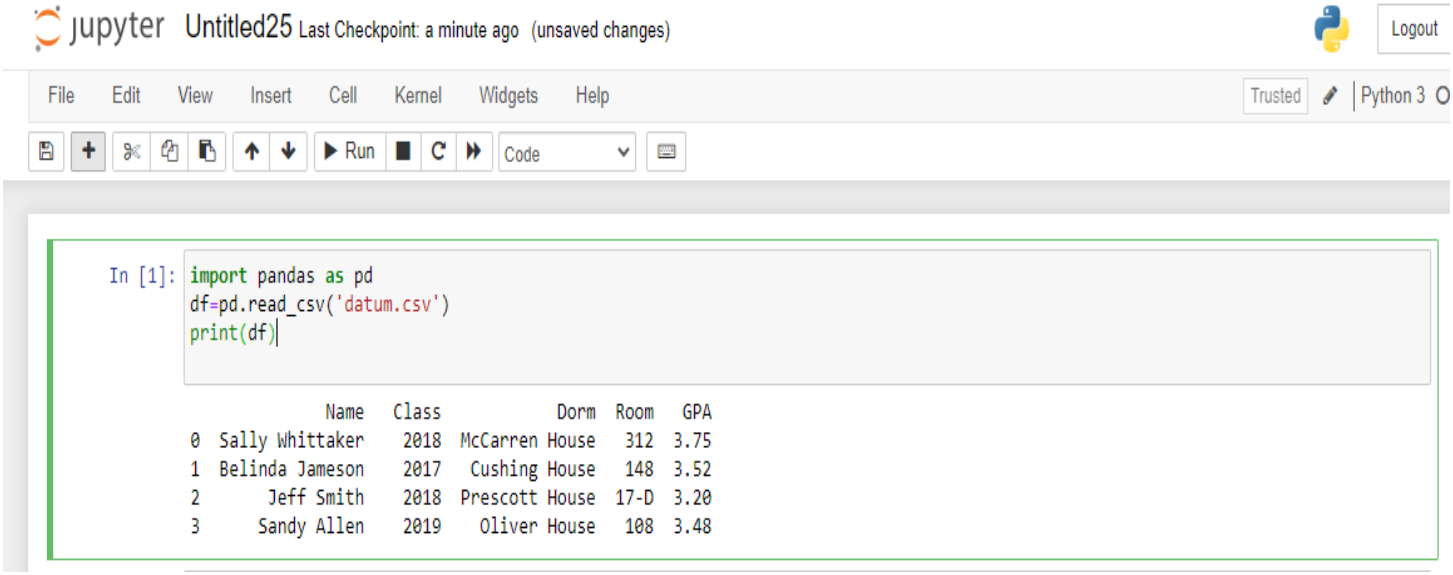
Sandy Allen,2019,Oliver House,108,3.48

Step 5: Save the file using extension as ‘csv’ eg: ‘data.csv’



**Import CSV files into internal Data Structure**

```
import pandas as pd
df=pd.read_csv('datum.csv')
print(df)
```

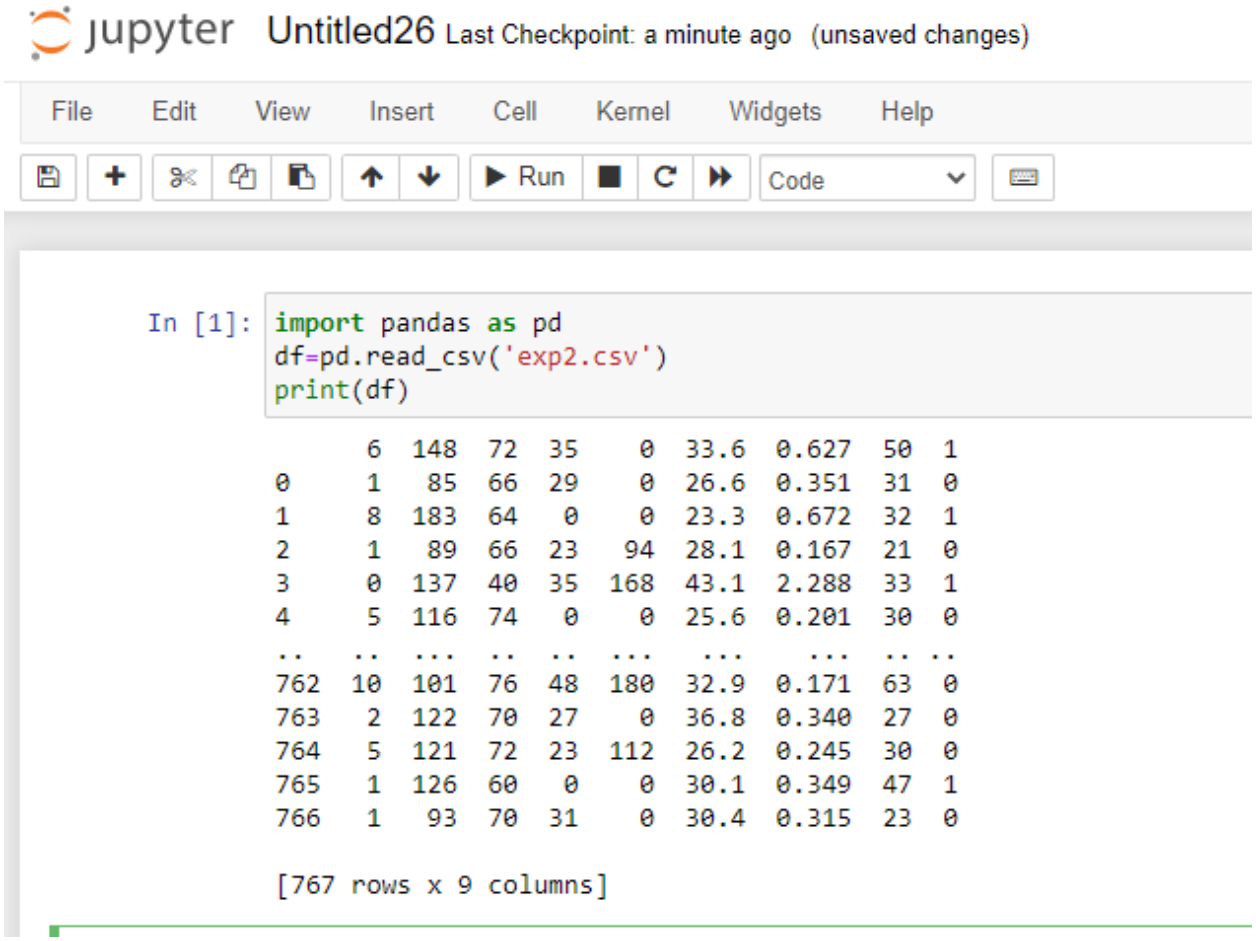


# Practical-2

**Python Programming for Loading the Data.**

**Load CSV with Python Standard Library**

The Python API provides the module CSV and the function reader() that can be used to load CSV files. Once loaded, you convert the CSV data to a NumPy array and use it for machine learning.




All fields are numeric and there is no header line. Running the recipe below will load the CSV file and convert it to a NumPy array.

**# Load CSV (using python)**

```
import csv
import numpy
filename = 'exp2.csv'
raw_data = open(filename, 'rt')
reader = csv.reader(raw_data, delimiter=',', quoting=csv.QUOTE_NONE)
```

```
x = list(reader)
data = numpy.array(x).astype('float')
print(data.shape)
```

```
In [3]: # Load CSV (using python)
import csv
import numpy
filename = 'exp2.csv'
raw_data = open(filename, 'rt')
reader = csv.reader(raw_data, delimiter=',', quoting=csv.QUOTE_NONE)
x = list(reader)
data = numpy.array(x).astype('float')
print(data.shape)

(768, 9)
```

The above example loads an object that can iterate over each row of the data and can easily be converted into a NumPy array. Running the example prints the shape of the array.

**Load CSV File With Pandas**

You can load your CSV data using Pandas and the pandas.read_csv() function.

This function is very flexible and is perhaps my recommended approach for loading your machine learning data. The function returns a pandas. DataFrame that you can immediately start summarizing and plotting.

The example below assumes that the 'exp2.csv' file is in the current working directory.

```
# Load CSV using Pandas
import pandas
filename = 'exp2.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
data = pandas.read_csv(filename, names=names)
print(data.shape)
```

```
In [4]:
# Load CSV using Pandas
import pandas
filename = 'exp2.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
data = pandas.read_csv(filename, names=names)
print(data.shape)

(768, 9)
```

Note that in this example we explicitly specify the names of each attribute to the DataFrame. Running the example displays the shape of the data

# Practical-3

**Python Programming for Cleaning the Data.**

**How to perform Data Cleaning in Python?**

For understanding, let's take the example of a survey in which a company's HR staff has to locate all of its area employees, and make sure they are safe at home. Before that, let's understand that there are no Standard Data Cleaning Techniques. It is not possible to comment on which one is best. The only aspect which needs to be considered on cleaning methods depends on the nature of Data. This helps us choose which technique should be used.

We will get back to the example. To keep it simpler, we are looking at below fields.

| Sr.no | Name | Address | Mobile no. | Project | email |
|---|---|---|---|---|---|
| 1 | Princy Odom | Street 1 New Lanes | 7894561230 | Client | princy@text.com |
| 2 | Appy Alexa | Ac Street | 5689237412 | Client | appy@text.com |
| 3 | Luna Knox | P.O. Box 12-Lorem Park | | | luna@text.com |
| 4 | Clark Andrews | P.O. Box 156-Settle Road | 1240589635 | Client | clark@text.com |
| 5 | Riley Drew | P.O. Box 16-Thames Street | 7894758698 | Internal | riley@text.com |
| 6 | August Bork | P.O. Box 17-Walter Street | 1254789635 | Internal | aug@text.com |
| 7 | Riley Drew | P.O. Box 16-Thames Street | 7894758698 | Internal | riley@text.com |
| 8 | XYS | Street 2 New way | 000000000 | | |
| 9 | Erica Flint | Street 7 New lane road | 7895471238 | internal | erica@text.com |

Look at the table carefully. You'll notice that certain fields are either blank or have irrelevant values. If we process such data, then our prediction will be in trouble. Thus, we will carry out the below steps for Data Cleaning.

- Remove Repeated Row
- Missing value treatment
- Removal of Irrelevant Data
- Manual error while typing
- Renaming Columns

As to process data, our first step would be to read data in Python.

```
#Step1 Import Package
import pandas as pd
import numpy as np
employee=pd.read_csv('exp3.csv')
print(employee)
```

```
In [7]: #Step1 Import Package
        import pandas as pd
        import numpy as np
        employee=pd.read_csv('exp3.csv')
        print(employee)
```

```
   Sr.no          Name                   Address    Mobile no.   Project  \
0      1    Princy Odom        Street 1 New Lanes  7.894561e+08    Client
1      2     Appy Alexa                AC Street  5.689237e+08    Client
2      3      Luna Knox     P.O.Box 12-Lorem Park           NaN       NaN
3      4  Clark Andrews  P.O. Box 156-Settle Road  1.240590e+08    Client
4      5     Riley Drew   P.O.Box 16-Thams Street  7.894759e+09  Internal
5      6    August Bork  P.O.Box 17-Walter Street  1.254790e+08  Internal
6      7     Riley Drew   P.O.Box 16-Thams Street  7.894759e+09  Internal
7      8            XYS          Street 2 New way  0.000000e+00       NaN
8      9    Erica Flint    Street 7 New lane road  7.895471e+09  internal

             email
0  princy@text.com
1    appy@text.com
2    luna@text.com
3   clark@text.com
4   riley@text.com
5     aug@text.com
6   riley@text.com
7              NaN
8   erica@text.com
```

Observe the output table carefully; it's the same table which we had in the first place. The two important packages which we imported are **Panda and Numpy**. These are needed for a Python code to run. The next important thing is acronyms we adopted as a good practice. employee variable used to store data read from **exp3.csv** file saved at the mentioned location. The command used to read data is read_csv and displayed on the screen by using print. For the fields having missing value, the system has filled it with NaN(Not a number).

**Remove Repeated Values**

We know that there are duplicates in the dataset and that need to be removed. Row 5 and 7 have the same employee data.

```
4      5     Riley Drew   P.O.Box 16-Thams Street  7.894759e+09  Internal
5      6    August Bork  P.O.Box 17-Walter Street  1.254790e+08  Internal
6      7     Riley Drew   P.O.Box 16-Thams Street  7.894759e+09  Internal
```

We can delete the last row, and keep the first row

Function drop_duplicates returns output with repeated rows removed. Below are the parameters used in a command.

subset: We have assigned column name to subset parameters to check repeated values. By default, it takes all columns.

keep: for keeping the first row(5)

inplace: Boolean case, default false. By assigning true we asked the command to drop last value.

Data with no repeated values.

***#Duplicate rows on keeping the first row***
employee.drop_duplicates(subset="Name", keep="first", inplace=True)
print(employee).

```
In [10]: #Duplicate rows on keeping the first row
         employee.drop_duplicates(subset="Name", keep="first", inplace=True)
         print(employee)
```

```
   Sr.no          Name                   Address    Mobile no.   Project  \
0      1    Princy Odom        Street 1 New Lanes  7.894561e+08    Client
1      2     Appy Alexa                AC Street  5.689237e+08    Client
2      3      Luna Knox     P.O.Box 12-Lorem Park           NaN       NaN
3      4  Clark Andrews  P.O. Box 156-Settle Road  1.240590e+08    Client
4      5     Riley Drew   P.O.Box 16-Thams Street  7.894759e+09  Internal
5      6    August Bork  P.O.Box 17-Walter Street  1.254790e+08  Internal
7      8            XYS          Street 2 New way  0.000000e+00       NaN
8      9    Erica Flint    Street 7 New lane road  7.895471e+09  internal

             email
0  princy@text.com
1    appy@text.com
2    luna@text.com
3   clark@text.com
4   riley@text.com
5     aug@text.com
7              NaN
8   erica@text.com
```

**Missing Value Treatment**

In panda, missing data is represented in two ways. None or NaN. In our dataset, missing values are recognized as NaN. For checking missing values, we have used the function is.null().

***#check missing value***
missing=employee.isnull()
print(missing)

```
In [11]: #check missing value
         missing=employee.isnull()
         print(missing)
```

```
   Sr.no   Name  Address  Mobile no.  Project  email
0  False  False    False       False    False  False
1  False  False    False       False    False  False
2  False  False    False        True     True  False
3  False  False    False       False    False  False
4  False  False    False       False    False  False
5  False  False    False       False    False  False
7  False  False    False       False     True   True
8  False  False    False       False    False  False
```

The output of the command is Boolean which is True for NaN values.

As shown in the output image, column Mobile no., Project and email have missing values. Missing values can either be filled or dropped.

In our example, we are working on employee data so filling it with any value will be inappropriate. Hence, we have dropped the missing values.

***#missing value***
employee=employee.dropna(axis=0)
print(employee)

```
In [12]: #missing value
         employee=employee.dropna(axis=0)
         print(employee)
```

```
   Sr.no           Name                   Address    Mobile no.   Project  \
0      1     Princy Odom        Street 1 New Lanes  7.894561e+08    Client
1      2      Appy Alexa                AC Street  5.689237e+08    Client
3      4   Clark Andrews  P.O. Box 156-Settle Road  1.240590e+08    Client
4      5      Riley Drew   P.O.Box 16-Thams Street  7.894759e+09  Internal
5      6     August Bork  P.O.Box 17-Walter Street  1.254790e+08  Internal
8      9     Erica Flint    Street 7 New lane road  7.895471e+09  internal

             email
0  princy@text.com
1    appy@text.com
3   clark@text.com
4   riley@text.com
5     aug@text.com
8   erica@text.com
```

The command dropna drops rows/columns having at least null values in CSV file.
As shown in the output table, missing values are removed from the dataset.

**Removal of irrelevant data**

Sometimes, certain categories/columns in a dataset are not useful. In our case, column name, serial no are not important. Retaining it will take unnecessary space and consume time.
Panda provides an easy command del to remove the unwanted column.

***#removal of irrelevant data***

del employee['Sr.no']

```
In [13]: #removal of irrelevant data
         del employee['Sr.no']
```

```
In [14]: print(employee)
```

```
            Name                    Address    Mobile no.   Project  \
0    Princy Odom        Street 1 New Lanes  7.894561e+08    Client
1     Appy Alexa                 AC Street  5.689237e+08    Client
3  Clark Andrews  P.O. Box 156-Settle Road  1.240590e+08    Client
4     Riley Drew   P.O.Box 16-Thams Street  7.894759e+09  Internal
5    August Bork  P.O.Box 17-Walter Street  1.254790e+08  Internal
8    Erica Flint    Street 7 New lane road  7.895471e+09  internal

             email
0  princy@text.com
1    appy@text.com
3   clark@text.com
4   riley@text.com
5     aug@text.com
8   erica@text.com
```

**Manual Error While Typing**

In this step, we need to make sure the data has correct values whenever there are categories mentioned. In our case, the column Project has two possible values Client or Internal. But close observation of the output table can help us to point row 8 where the value is not following the possible case. (i.e, internal should be corrected to Internal).
In python, we can replace the column with the corrected value.

***#typing mistake***

employee['Project']=employee['Project'].str.replace('internal','Internal')

```
In [15]: #typing mistake
         employee['Project']=employee['Project'].str.replace('internal','Internal')
```

```
In [16]: print(employee)
```

```
            Name                    Address    Mobile no.   Project  \
0    Princy Odom        Street 1 New Lanes  7.894561e+08    Client
1     Appy Alexa                AC Street  5.689237e+08    Client
3  Clark Andrews  P.O. Box 156-Settle Road  1.240590e+08    Client
4     Riley Drew   P.O.Box 16-Thams Street  7.894759e+09  Internal
5    August Bork  P.O.Box 17-Walter Street  1.254790e+08  Internal
8    Erica Flint    Street 7 New lane road  7.895471e+09  Internal
```

**Renaming Columns**

The dataset we are working on has a column name that does not speak about the importance of the data it holds. Thus, we will add some sensible labels by renaming the columns.
The output demonstrates the change in column name.

***#Renaming Columns***

employee.columns=['EmployeeName','Address','ContactNumber','ProjectType','EmailID']
print(employee)

```
In [17]: #Renaming Columns
         employee.columns=['EmployeeName','Address','ContactNumber','ProjectType','EmailID']
         print(employee)
```

```
    EmployeeName                    Address  ContactNumber ProjectType  \
0    Princy Odom        Street 1 New Lanes   7.894561e+08      Client
1     Appy Alexa                AC Street   5.689237e+08      Client
3  Clark Andrews  P.O. Box 156-Settle Road   1.240590e+08      Client
4     Riley Drew   P.O.Box 16-Thams Street   7.894759e+09    Internal
5    August Bork  P.O.Box 17-Walter Street   1.254790e+08    Internal
8    Erica Flint    Street 7 New lane road   7.895471e+09    Internal

          EmailID
0  princy@text.com
1    appy@text.com
3   clark@text.com
4   riley@text.com
5     aug@text.com
8   erica@text.com
```

# Practical-4

**WAP to implement the support vector classifier using Python libray Scikit. Visualizing and analyzing the model.**

```
#Data Pre-processing Step

# importing libraries

import numpy as nm

import matplotlib.pyplot as mtp

import pandas as pd


#importing datasets

data_set= pd.read_csv('user_data.csv')


#Extracting Independent and dependent Variable

x= data_set.iloc[:, [2,3]].values

y= data_set.iloc[:, 4].values


# Splitting the dataset into training and test set.

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

from sklearn.preprocessing import StandardScaler

st_x= StandardScaler()

x_train= st_x.fit_transform(x_train)

x_test= st_x.transform(x_test)

#Fitting the SVM classifier to the training set:

#Now the training set will be fitted to the SVM classifier. To create the SVM classifier, we will import SVC class from Sklearn.svm library. Below is the code for it:

from sklearn.svm import SVC # "Support vector classifier"
```

```
classifier = SVC(kernel='linear', random_state=0)

classifier.fit(x_train, y_train)
```

#Predicting the test set result:

#Now, we will predict the output for test set. For this, we will create a new vector y_pred. Below is the code for it:

```
#Predicting the test set result

y_pred= classifier.predict(x_test)

#Creating the Confusion matrix

from sklearn.metrics import confusion_matrix

cm= confusion_matrix(y_test, y_pred)
```

#Visualizing the training set result:

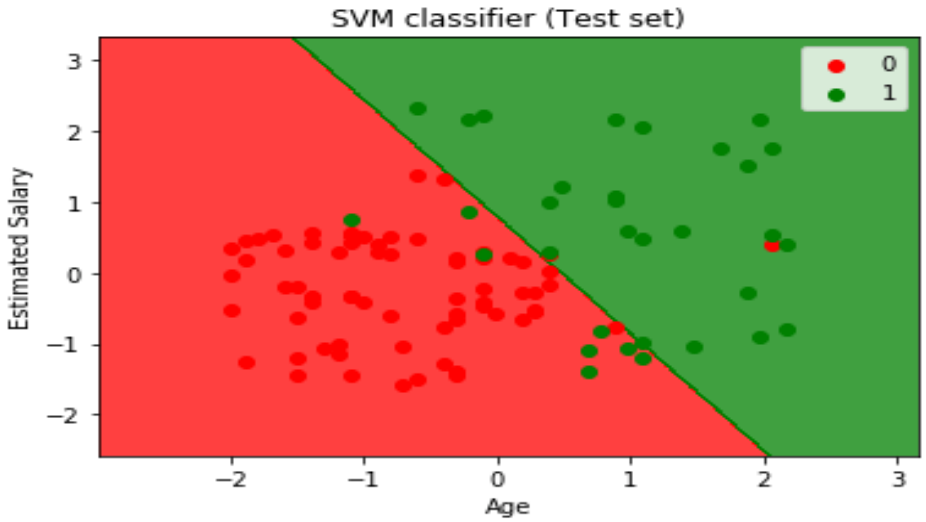
#Now we will visualize the training set result, below is the code for it:

```
from matplotlib.colors import ListedColormap

x_set, y_set = x_train, y_train

x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop = x_set[:, 0].max() + 1, step  =0.01),

nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0.01))

mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(), x2.ravel()]).T).reshape(x1.shape),

alpha = 0.75, cmap = ListedColormap(('red', 'green')))

mtp.xlim(x1.min(), x1.max())

mtp.ylim(x2.min(), x2.max())

for i, j in enumerate(nm.unique(y_set)):

    mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],

        c = ListedColormap(('red', 'green'))(i), label = j)

mtp.title('SVM classifier (Training set)')

mtp.xlabel('Age')

mtp.ylabel('Estimated Salary')

mtp.legend()

mtp.show()
```

```
#Visulaizing the test set result
from matplotlib.colors import ListedColormap
x_set, y_set = x_test, y_test
x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop = x_set[:, 0].max() + 1, step  =0.01),
nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0.01))
mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(), x2.ravel()]).T).reshape(x1.shape),
alpha = 0.75, cmap = ListedColormap(('red','green' )))
mtp.xlim(x1.min(), x1.max())
mtp.ylim(x2.min(), x2.max())
for i, j in enumerate(nm.unique(y_set)):
    mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
        c = ListedColormap(('red', 'green'))(i), label = j)
mtp.title('SVM classifier (Test set)')
mtp.xlabel('Age')
mtp.ylabel('Estimated Salary')
mtp.legend()
mtp.show()
```

**Output:**

By executing the above code, we will get the output as:



# Practical-5

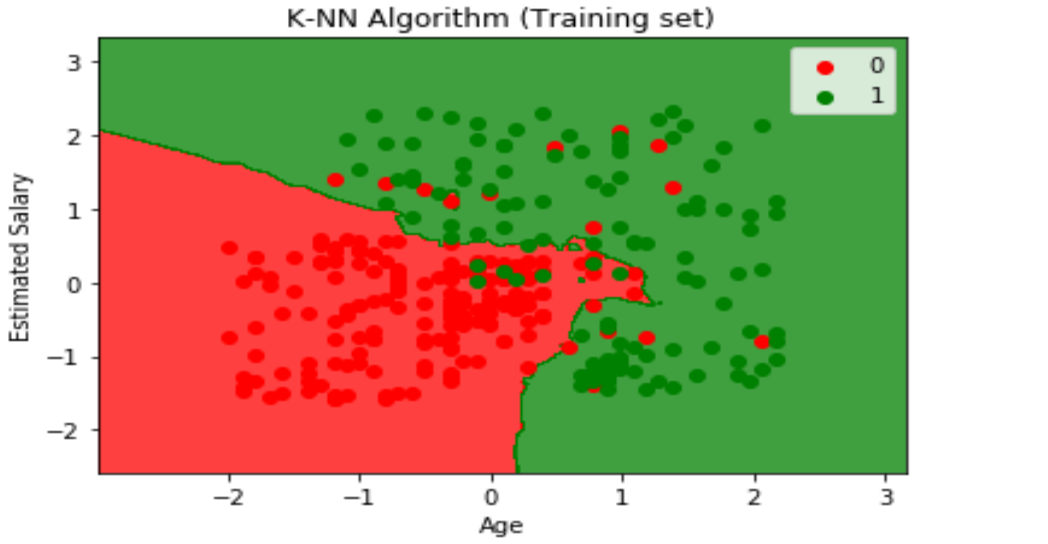
**WAP to implement the k-nearest neighbour algorithm using Python library. Claculate the distance, Find K nearest point, predict the class, check the accuracy**

```
# importing libraries

import numpy as nm

import matplotlib.pyplot as mtp

import pandas as pd


#importing datasets

data_set= pd.read_csv('user_data.csv')


#Extracting Independent and dependent Variable

x= data_set.iloc[:, [2,3]].values

y= data_set.iloc[:, 4].values


# Splitting the dataset into training and test set.

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

from sklearn.preprocessing import StandardScaler

st_x= StandardScaler()

x_train= st_x.fit_transform(x_train)

x_test= st_x.transform(x_test)

#Fitting K-NN classifier to the training set

from sklearn.neighbors import KNeighborsClassifier

classifier= KNeighborsClassifier(n_neighbors=5, metric='minkowski', p=2 )
```

```
classifier.fit(x_train, y_train)
#Predicting the test set result
y_pred= classifier.predict(x_test)
Creating the Confusion matrix
    from sklearn.metrics import confusion_matrix
    cm= confusion_matrix(y_test, y_pred)
#Visulaizing the trianing set result
from matplotlib.colors import ListedColormap
x_set, y_set = x_train, y_train
x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop = x_set[:, 0].max() + 1, step  =0.01),
nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0.01))
mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(), x2.ravel()]).T).reshape(x1.shape),
alpha = 0.75, cmap = ListedColormap(('red','green' )))
mtp.xlim(x1.min(), x1.max())
mtp.ylim(x2.min(), x2.max())
for i, j in enumerate(nm.unique(y_set)):
    mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
        c = ListedColormap(('red', 'green'))(i), label = j)
mtp.title('K-NN Algorithm (Training set)')
mtp.xlabel('Age')
mtp.ylabel('Estimated Salary')
mtp.legend()
mtp.show()
```

**Output:**

**By executing the above code, we will get the below graph:**




#Visualizing the test set result

from matplotlib.colors import ListedColormap

x_set, y_set = x_test, y_test

x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop = x_set[:, 0].max() + 1, step =0.01),

nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0.01))

mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(), x2.ravel()]).T).reshape(x1.shape),

alpha = 0.75, cmap = ListedColormap(('red','green' )))

mtp.xlim(x1.min(), x1.max())

mtp.ylim(x2.min(), x2.max())

for i, j in enumerate(nm.unique(y_set)):

    mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],

        c = ListedColormap(('red', 'green'))(i), label = j)

mtp.title('K-NN algorithm(Test set)')

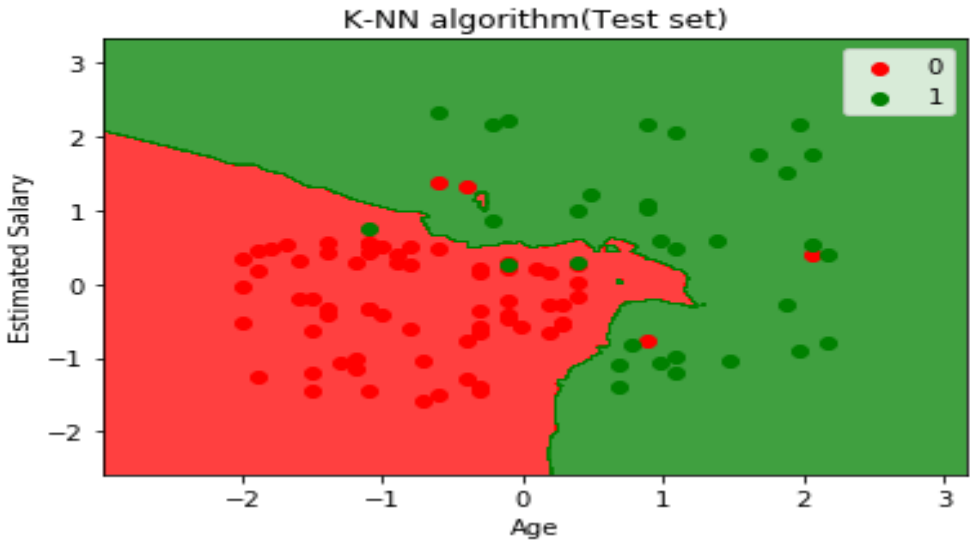
mtp.xlabel('Age')

mtp.ylabel('Estimated Salary')

mtp.legend()

mtp.show()

**Output:**



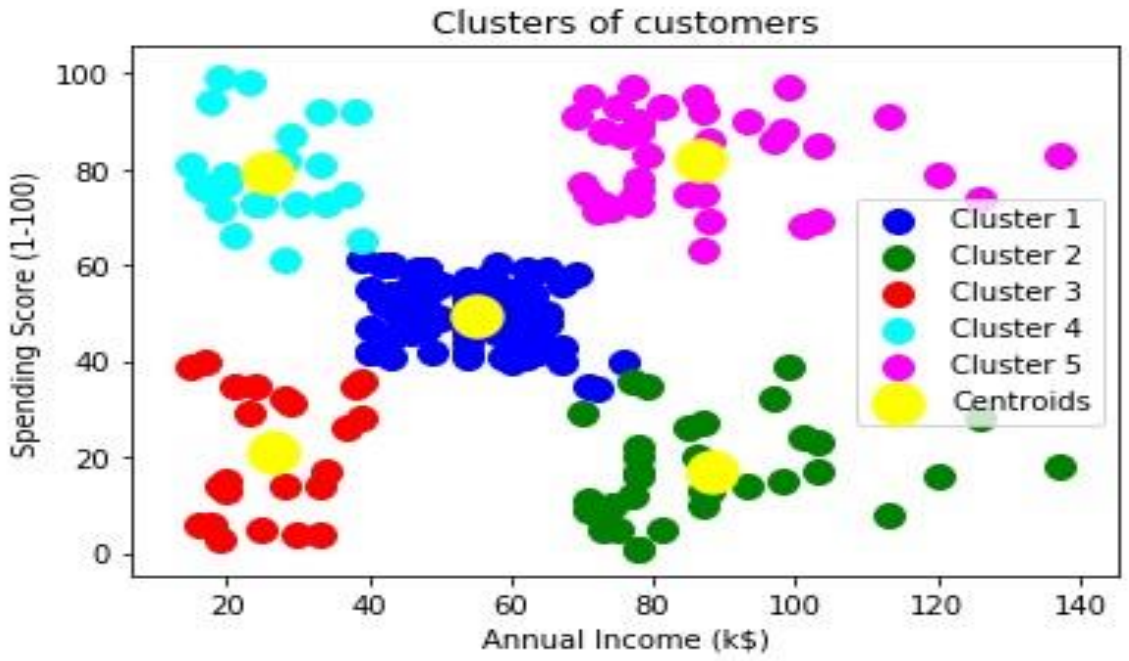
# Practical-6

**WAP to implement the k-means clustering using Python library Scikit-learn. Visualizing and analyzing the model.**

```
# importing libraries

import numpy as nm

import matplotlib.pyplot as mtp

import pandas as pd


# Importing the dataset

dataset = pd.read_csv('Mall_Customers_data.csv')

#Extracting Independent Variables

x = dataset.iloc[:, [3, 4]].values


#finding optimal number of clusters using the elbow method

from sklearn.cluster import KMeans

wcss_list= []  #Initializing the list for the values of WCSS


#Using for loop for iterations from 1 to 10.

for i in range(1, 11):

   kmeans = KMeans(n_clusters=i, init='k-means++', random_state= 42)

   kmeans.fit(x)

   wcss_list.append(kmeans.inertia_)

mtp.plot(range(1, 11), wcss_list)

mtp.title('The Elobw Method Graph')

mtp.xlabel('Number of clusters(k)')

mtp.ylabel('wcss_list')

mtp.show()
```

```
#training the K-means model on a dataset

kmeans = KMeans(n_clusters=5, init='k-means++', random_state= 42)

y_predict= kmeans.fit_predict(x)


#visulaizing the clusters

mtp.scatter(x[y_predict == 0, 0], x[y_predict == 0, 1], s = 100, c = 'blue', label = 'Cluster 1') #for first cluster

mtp.scatter(x[y_predict == 1, 0], x[y_predict == 1, 1], s = 100, c = 'green', label = 'Cluster 2') #for second cluster

mtp.scatter(x[y_predict== 2, 0], x[y_predict == 2, 1], s = 100, c = 'red', label = 'Cluster 3') #for third cluster

mtp.scatter(x[y_predict == 3, 0], x[y_predict == 3, 1], s = 100, c = 'cyan', label = 'Cluster 4') #for fourth cluster

mtp.scatter(x[y_predict == 4, 0], x[y_predict == 4, 1], s = 100, c = 'magenta', label = 'Cluster 5') #for fifth cluster

mtp.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s = 300, c = 'yellow', label = 'Centroid')

mtp.title('Clusters of customers')

mtp.xlabel('Annual Income (k$)')

mtp.ylabel('Spending Score (1-100)')

mtp.legend()

mtp.show()
```

**Output:**


Clusters of customers
Spending Score (1-100)
Annual Income (k$)
100
80
60
40
20
0
20
40
60
80
100
120
140
Cluster 1
Cluster 2
Cluster 3
Cluster 4
Cluster 5
Centroids

# Practical-7

**WAP to implement K-nearest neighbour algorithm to classify the isris data set. Print both correct and wrong predictions.**

```
import pandas as pd

import numpy as np

import math

import operator


#loading the data set and analysis

col=['sepal_length','sepal_width','petal_length','petal_width','type']

iris=pd.read_csv("iris.xlsx",names=col)

#data set link:https://github.com/saireddyavs/applied-ai/blob/master/iris.xlsx

print("First five rows")

print(iris.head())

print("**********")

print("columns",iris.columns)

print("**********")

print("shape:",iris.shape)

print("**********")

print("Size:",iris.size)

print("**********")

print("no of samples available for each type")

print(iris['type'].value_counts())

print("**********")

print(iris.describe())


#code for finding euclidian diastance and knn:
```

```
def euclidianDistance(data1, data2, length):
    distance = 0
    for x in range(length):
        distance += np.square(data1[x] - data2[x])

    return np.sqrt(distance)
def knn(trainingSet, testInstance, k):

    distances = {}
    sort = {}
length = testInstance.shape[1]
    print(length)
    # Calculating euclidean distance between each row of training data and test data
    for x in range(len(trainingSet)):
        dist = euclidianDistance(testInstance, trainingSet.iloc[x], length)
distances[x] = dist[0]
    # Sorting them on the basis of distance
    sorted_d = sorted(distances.items(), key=operator.itemgetter(1)) #by using it we store indices also
    sorted_d1 = sorted(distances.items())
    print(sorted_d[:5])
    print(sorted_d1[:5])
    neighbors = []
    # Extracting top k neighbors
    for x in range(k):
        neighbors.append(sorted_d[x][0])
counts = {"Iris-setosa":0,"Iris-versicolor":0,"Iris-virginica":0}
    # Calculating the most freq class in the neighbors
```

```
    for x in range(len(neighbors)):
        response = trainingSet.iloc[neighbors[x]][-1]
        if response in counts:
            counts[response] += 1
        else:
            counts[response] = 1
    print(counts)
    sortedVotes = sorted(counts.items(), key=operator.itemgetter(1), reverse=True)
    print(sortedVotes)
    return(sortedVotes[0][0], neighbors)
#now apply some test point for the above method to predict:
testSet = [[1.4, 3.6, 3.4, 1.2]]
test = pd.DataFrame(testSet)
result,neigh = knn(iris, test, 4)#here we gave k=4
print("And the flower is:",result)
print("the neighbors are:",neigh)
```

**output:**

**And the flower is: Iris-setosa**

**the neighbors are: [57, 8, 42, 93]**

# Practical-8

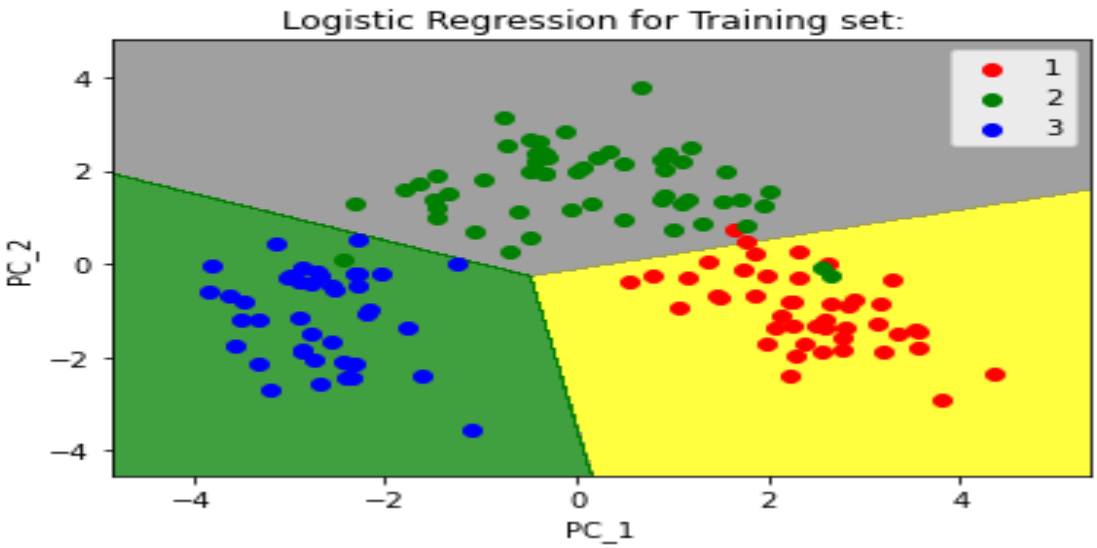
**WAP to implement the PCA using python library Scikit-learn. Visualizing and analyzing the model.**

```
import numpy as nmp

import matplotlib.pyplot as mpltl

import pandas as pnd

DS = pnd.read_csv('Wine.csv')


# Now, we will distribute the dataset into two components "X" and "Y"

X = DS.iloc[: , 0:13].values

Y = DS.iloc[: , 13].values

from sklearn.model_selection import train_test_split as tts

X_train, X_test, Y_train, Y_test = tts(X, Y, test_size = 0.2, random_state = 0)

rom sklearn.preprocessing import StandardScaler as SS

SC = SS()

X_train = SC.fit_transform(X_train)

X_test = SC.transform(X_test)

from sklearn.decomposition import PCA

PCa = PCA (n_components = 1)

X_train = PCa.fit_transform(X_train)

X_test = PCa.transform(X_test)

explained_variance = PCa.explained_variance_ratio_

from sklearn.linear_model import LogisticRegression as LR


classifier_1 = LR (random_state = 0)
```

```
classifier_1.fit(X_train, Y_train)
Y_pred = classifier_1.predict(X_test)
#We will create the confusion matrix.
from sklearn.metrics import confusion_matrix as CM
c_m = CM (Y_test, Y_pred)
#Then, predict the result of the training set.
from matplotlib.colors import ListedColormap as LCM
X_set, Y_set = X_train, Y_train
X_1, X_2 = nmp.meshgrid(nmp.arange(start = X_set[:, 0].min() - 1,
                stop = X_set[: , 0].max() + 1, step = 0.01),
                  nmp.arange(start = X_set[: , 1].min() - 1,
                  stop = X_set[: , 1].max() + 1, step = 0.01))

mpltl.contourf(X_1, X_2, classifier_1.predict(nmp.array([X_1.ravel(),
            X_2.ravel()]).T).reshape(X_1.shape), alpha = 0.75,
            cmap = LCM (('yellow', 'grey', 'green')))

mpltl.xlim (X_1.min(), X_1.max())
mpltl.ylim (X_2.min(), X_2.max())

for s, t in enumerate(nmp.unique(Y_set)):
    mpltl.scatter(X_set[Y_set == t, 0], X_set[Y_set == t, 1],
                c = LCM (('red', 'green', 'blue'))(s), label = t)

mpltl.title('Logistic Regression for Training set: ')
mpltl.xlabel ('PC_1') # for X_label
mpltl.ylabel ('PC_2') # for Y_label
```

```
mpltl.legend() # for showing legend


# show scatter plot

mpltl.show()
```

**Output:**




```
#At last, we will visualize the result of the testing set.

from matplotlib.colors import ListedColormap as LCM

X_set, Y_set = X_test, Y_test

X_1, X_2 = nmp.meshgrid(nmp.arange(start = X_set[: , 0].min() - 1,

                    stop = X_set[: , 0].max() + 1, step = 0.01),

                    nmp.arange(start = X_set[: , 1].min() - 1,

                    stop = X_set[: , 1].max() + 1, step = 0.01))

mpltl.contourf(X_1, X_2, classifier_1.predict(nmp.array([X_1.ravel(),

            X_2.ravel()]).T).reshape(X_1.shape), alpha = 0.75,

            cmap = LCM(('pink', 'grey', 'aquamarine')))
```

```
mpltl.xlim(X_1.min(), X_1.max())

mpltl.ylim(X_2.min(), X_2.max())


for s, t in enumerate(nmp.unique(Y_set)):

    mpltl.scatter(X_set[Y_set == t, 0], X_set[Y_set == t, 1],

                c = LCM(('red', 'green', 'blue'))(s), label = t)


# title for scatter plot

mpltl.title('Logistic Regression for Testing set')

mpltl.xlabel ('PC_1') # for X_label

mpltl.ylabel ('PC_2') # for Y_label

mpltl.legend()


# show scatter plot

mpltl.show()
```

**Output:**