[IT 341] Machine Learning Lab

Name: Ankita Priya Roll no.: BTECH/60036/18

INDEX

Multiple Linear Regression from Scratch

In [58]:
```
# Importing the necessary libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
```

Loading the dataset

In [47]:
```
# Loading the dataframe
df = pd.read_csv('energy.txt')
df.head()
```

Out[47]:

	T	V	AP	RH	EP
0	8.58	38.38	1021.03	84.37	482.26
1	21.79	58.20	1017.21	66.74	446.94
2	16.64	48.92	1011.55	78.76	452.56
3	31.38	71.32	1009.17	60.42	433.44
4	9.20	40.03	1017.05	92.46	480.38

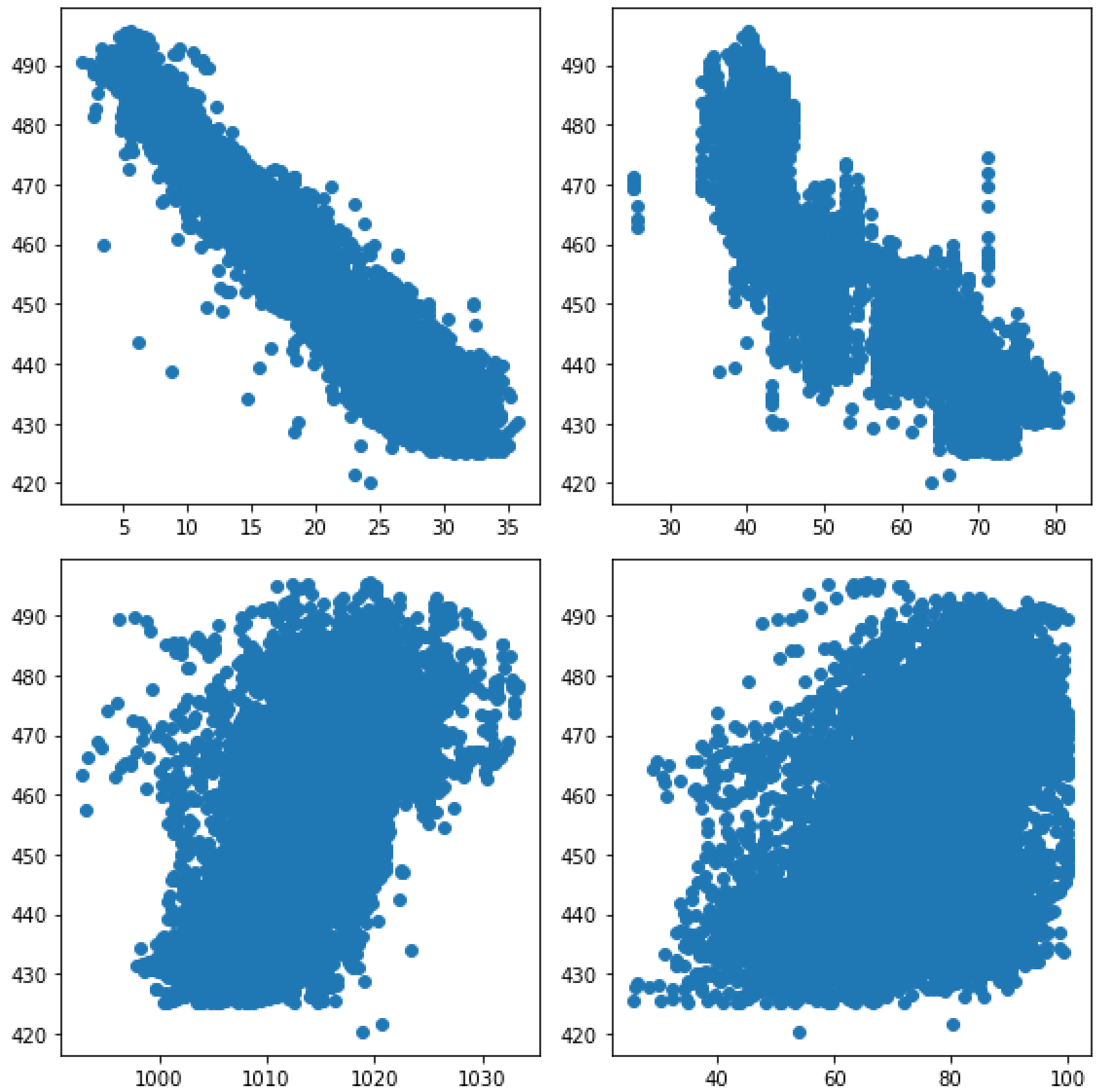
In [25]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7176 entries, 0 to 7175 Data
columns (total 5 columns):
 #    Column   Non-Null Count    Dtype
-----  ----------  ----------------------  -------------
  0    T         7176  non-null     float64
  1    V         7176  non-null     float64
  2    AP        7176  non-null     float64
  3    RH        7176  non-null     float64
  4    EP        7176  non-null     float64
dtypes: float64(5)
memory usage: 280.4 KB
```

Visualizing the dataset

In [53]:

```python
# Here out target variable is EP. So we will plot graphs correspondin g to each feature
# plt.figure(figsize=(20, 20))
fig, ax = plt.subplots(nrows=2, ncols=2, figsize=(8, 8)) fig.figsize
= (10, 10)
ax[0, 0].scatter(df['T'], df['EP'])
ax[0, 1].scatter(df['V'], df['EP'])
ax[1, 0].scatter(df['AP'], df['EP'])
ax[1, 1].scatter(df['RH'], df['EP'])

plt.tight_layout()
```



Preprocessing the dataset

In [144]:

```
# Seperate features and target variable
X = df.iloc[:, :4]
y = df.iloc[:, -1]

# Scaling the dataset to fit the model
from sklearn.preprocessing import StandardScaler sc
= StandardScaler()
X = sc.fit_transform(X)

# Dividing the data into training and testing data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25)
```

In [146]:

```
# Printing the size of datasets
print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
```

```
(5382, 4) (1794, 4) (5382,) (1794,)
```

Linear Regression from Scratch

In [224]:

```
def cost_function(X, y, w, b):
        """
        Parameters:
        X: features
        y: target values w:
        weights
        b: bias

        Returns:
        cost: cost with current weights and bias """
        cost = np.sum((((X.dot(w) + b) - y) ** 2) / (2*len(y)))
        return cost
```

In [270]:

```
def gradient_descent_function(X, y, w, b, alpha=0.01, epochs=1000):
    """
    Parameters:
    X: features
    y: target values w:
    initial weights b:
    initial bias
    alpha: learning rate
    epochs: number of iterations

    Returns:
    costs: cost per epoch w:
    finalised weights b:
    finalised bias
    """
    m = len(y)
    costs = [0] * epochs

    for epoch in range(epochs):
        # Calculate the value -- Forward Propagation
        z = X.dot(w) + b

        # Calculate the losses
        loss = z - y

        # Calculate gradient descent
        weight_gradient = X.T.dot(loss) / m
        bias_gradient = np.sum(loss) / m

        # Update weights and bias
        w = w - alpha*weight_gradient b
        = b - alpha*bias_gradient

        # Store current lost
        cost = cost_function(X, y, w, b)
        costs[epoch] = cost
```

In [277]:

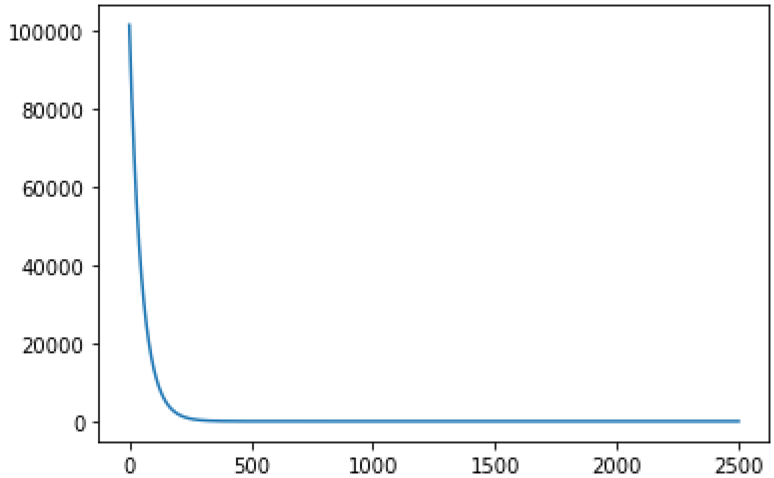
```
w = np.random.randn(X_train.shape[1]) b
= 0
weights, bias, costs = gradient_descent_function(X_train, y_train, w, b,
epochs=2500);
```

In [278]:

```
print(weights)
print(bias)
```

```
[-14.28178324   -3.37300348    0.46981278   -2.13213563]
454.3437214571094
```

```
In [279]: # Plotting the cost function
plt.plot(costs)
plt.show()
```



Calculating the performance of our model

```
In [249]: def predict(X, w, b):
      return X.dot(w) + b
```

```
In [248]: def r2score(y_pred, y):
      """
      Parameters:
      y_pred: predicted values y:
      actual values

      Returns:
      r2: r2 score
      """
      rss = np.sum((y_pred - y) ** 2) tss
      = np.sum((y-y.mean()) ** 2)

      r2 = 1 - (rss / tss)
      return r2
```

```
In [257]: # Predicted values with our model
y_pred = predict(X_test, weights, bias)
```

```
In [258]: r2 = r2score(y_pred, y_test)
print(r2)
```

0.9303029606124354

The r2 value of our model is 0.93 which is impressive.

Linear Regression by using library

In [206]:
```
from sklearn.linear_model import LinearRegression
```

In [261]:
```
model = LinearRegression()
model.fit(X_train, y_train)
print(m.coef_)
print(m.intercept_)
```

```
[-14.95685865   -2.87349112    0.35046583   -2.3514856 ]
454.34381898777883
```

In [262]:
```
model.score(X_test, y_test)
```

Out[262]:
```
0.9303028950981764
```

KNN from scratch

In [1]:
```
# Importing the important libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
```

In [2]:
```
# Loading the dataset
df = pd.read_csv('thyroid.txt')
df.head()
```

Out[2]:

	on_thyroxine	query_on_thyroxine	on_antithyroid_medication	thyroid_surgery	query_hypothy
0	0	0	0	0	
1	0	0	0	0	
2	0	0	0	0	
3	0	0	0	0	
4	0	0	0	0	

5 rows × 24 columns

In [3]:
```
df.describe()
```

Out[3]:

	on_thyroxine	query_on_thyroxine	on_antithyroid_medication	thyroid_surgery	query_hyp
count	3152.000000	3152.000000	3152.000000	3152.000000	315
mean	0.145305	0.017449	0.013325	0.032995	
std	0.352464	0.130959	0.114680	0.178652	
min	0.000000	0.000000	0.000000	0.000000	
25%	0.000000	0.000000	0.000000	0.000000	
50%	0.000000	0.000000	0.000000	0.000000	
75%	0.000000	0.000000	0.000000	0.000000	
max	1.000000	1.000000	1.000000	1.000000	

8 rows × 24 columns

Preprocessing the data

In [4]:

```
# Divide the data into features and target
X = df.iloc[:, :-1]
y = df.iloc[:, -1]
```

In [5]:

```
# Scale the data
sc =  StandardScaler() X =
sc.fit_transform(X)
```

In [6]:

```
# Divide the data into train and test samples
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size= 0.20)
print(X_train.shape, y_train.shape)
```

```
(2521, 23) (2521,)
```

Implementing KNN from scratch

In [7]:

```
# Euclidean Distance
def euclidean_distance(data1, data2):
    """
    Parameters:
    data1: point no. 1
    data2: point no. 2

    Returns:
    euclidiean distance between both the points """
    distance = 0
    for i in range(data2.shape[0]):
        distance += np.square(data1[i] - data2[i])
    return np.sqrt(distance)
```

In [8]:

```
# KNN to give out the result directly
def knn(train_x, train_y, dis_func, sample, k):
    """
    Parameters:
    train_x: training samples train_y:
    corresponding labels dis_func:
    calculates distance sample: one
    test sample
    k: number of nearest neighbors

    Returns:
    cl: class of the sample """

    distances = {}
    for i in range(len(train_x)):
        d = dis_func(sample, train_x[i])
        distances[i] = d
    sorted_dist = sorted(distances.items(), key = lambda x : (x[1], x [0]))

    # take k nearest neighbors
    neighbors = []
    for i in range(k):
        neighbors.append(sorted_dist[i][0])

    #convert indices into classes
    classes = [train_y.iloc[c] for c in neighbors]

    #count each classes in top k
    from collections import Counter
    counts = Counter(classes)

    #take vote of max number of samples of a class
    list_values = list(counts.values())
    list_keys = list(counts.keys())
    cl = list_keys[list_values.index(max(list_values))]

    return cl
```

In [9]:

```
sl = knn(X_train, y_train, euclidean_distance, X_test[10], 5)
```

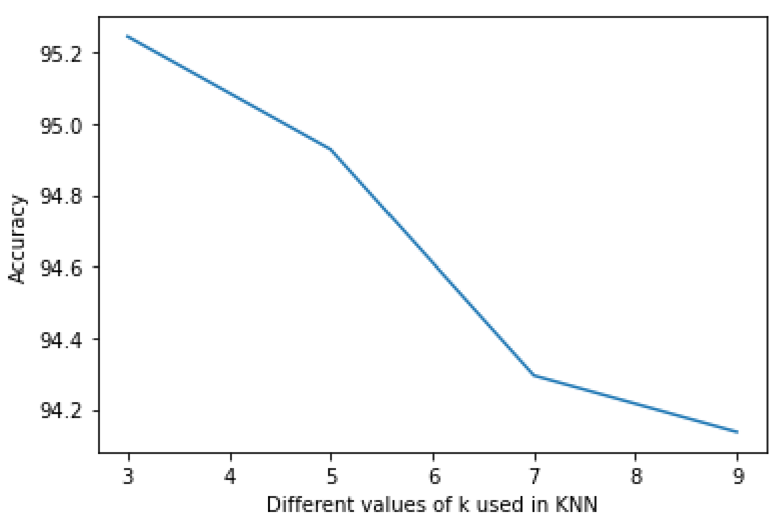
Testing our model with different values of k

In [12]:
```
def get_accuracy(test_x, test_y, train_x, train_y, k): correct =
    0
    for i in range(len(test_x)): sample
        = test_x[i] true_label =
        test_y.iloc[i]
        predicted_label_euclidean = knn(train_x, train_y, euclidean_d istance,
sample, k)
        if predicted_label_euclidean == true_label: correct
            += 1

    accuracy = (correct / len(test_x)) * 100

    print("Model accuracy with k = %d is %.2f" %(k, accuracy))
    return accuracy
```

In [21]:
```
diff_k = [3, 5, 7, 9] diff_acc =
[0] * len(diff_k) for i in
range(len(diff_k)):
    diff_acc[i] = get_accuracy(X_test, y_test, X_train, y_train, diff
_k[i])

print(diff_acc)
```

```
Model accuracy with k = 3 is 95.25
Model accuracy with k = 5 is 94.93
Model accuracy with k = 7 is 94.29
Model accuracy with k = 9 is 94.14
[95.24564183835183, 94.92868462757528, 94.29477020602218, 94.13629160
063391]
```

In [22]:
```
# Plotting the graph of k against accuracy
plt.plot(diff_k, diff_acc) plt.xlabel('Different values of k
used in KNN') plt.ylabel('Accuracy')
plt.show()
```



Using the library functions

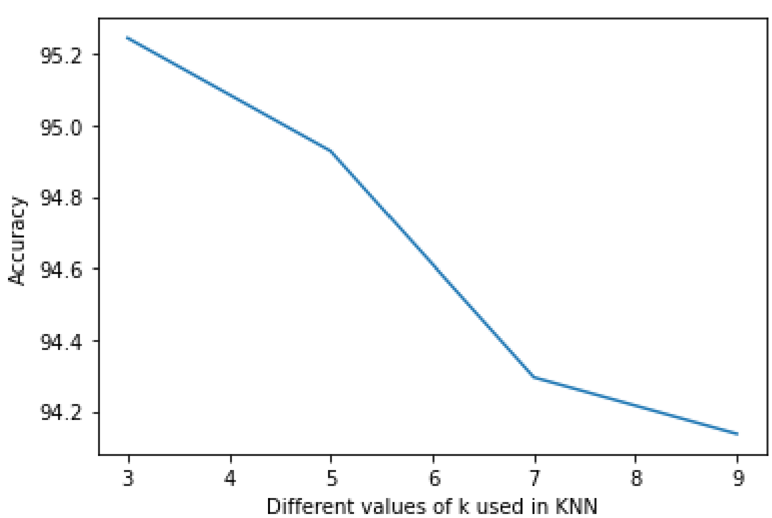
In [23]:
```
# Import the library
from sklearn.neighbors import KNeighborsClassifier
```

In [33]:
```
def get_accuracy_lib(test_x, test_y, train_x, train_y, k):
        # Initializing the model
        model = KNeighborsClassifier(n_neighbors = k)
        model.fit(train_x, train_y)

        # Testing the data
        accuracy = model.score(test_x, test_y)

        print("Model accuracy with k = %d is %.2f" %(k, accuracy))
        return accuracy * 100
```

In [34]:
```
diff_k = [3, 5, 7, 9] diff_acc_lib = [0]
* len(diff_k) for i in
range(len(diff_k)):
        diff_acc_lib[i] = get_accuracy_lib(X_test, y_test, X_train, y_tra in, diff_k[i])

print(diff_acc_lib)
```

```
Model  accuracy  with  k  =  3  is  0.95
Model  accuracy  with  k  =  5  is  0.95
Model  accuracy  with  k  =  7  is  0.94
Model accuracy with k = 9 is 0.94
[95.24564183835183, 94.92868462757528, 94.29477020602218, 94.13629160
063391]
```

In [35]:
```
# Plotting the graph of k against accuracy
plt.plot(diff_k, diff_acc) plt.xlabel('Different values of k
used in KNN') plt.ylabel('Accuracy')
plt.show()
```



In [36]:
```
diff_acc
```

Out[36]:
```
[95.24564183835183, 94.92868462757528, 94.29477020602218, 94.13629160
063391]
```

In [37]:
```
diff_acc_lib
```

Out[37]:
```
[95.24564183835183, 94.92868462757528, 94.29477020602218, 94.13629160
063391]
```

In [ ]:

In [ ]:

K Means Clustering for Customer Data

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

import plotly as py
import plotly.graph_objs as go

from sklearn.cluster import KMeans

import warnings
warnings.filterwarnings('ignore')
```

Data Exploration

In [2]:
```
df = pd.read_csv('../input/customer-segmentation-tutorial-in-python/M
all_Customers.csv')
df.head()
```

Out[2]:

	CustomerID	Gender	Age	Annual Income (k$)	Spending Score (1-100)
0	1	Male	19	15	39
1	2	Male	21	15	81
2	3	Female	20	16	6
3	4	Female	23	16	77
4	5	Female	31	17	40

In [3]:
```
df.columns
```

Out[3]:
```
Index(['CustomerID', 'Gender', 'Age', 'Annual Income (k$)', 'Spending
       Score (1-100)'],
      dtype='object')
```

In [4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199 Data
columns (total 5 columns):
 #    Column                  Non-Null Count    Dtype
-----  ----------------------  ----------------  -----------
 0    CustomerID              200 non-null      int64
 1    Gender                  200 non-null      object
 2    Age                     200 non-null      int64
 3    Annual Income  (k$)     200 non-null      int64
 4    Spending Score (1-100)  200 non-null      int64
dtypes: int64(4), object(1)
memory usage: 7.9+ KB
```

In [5]:
```
df.describe()
```

Out[5]:

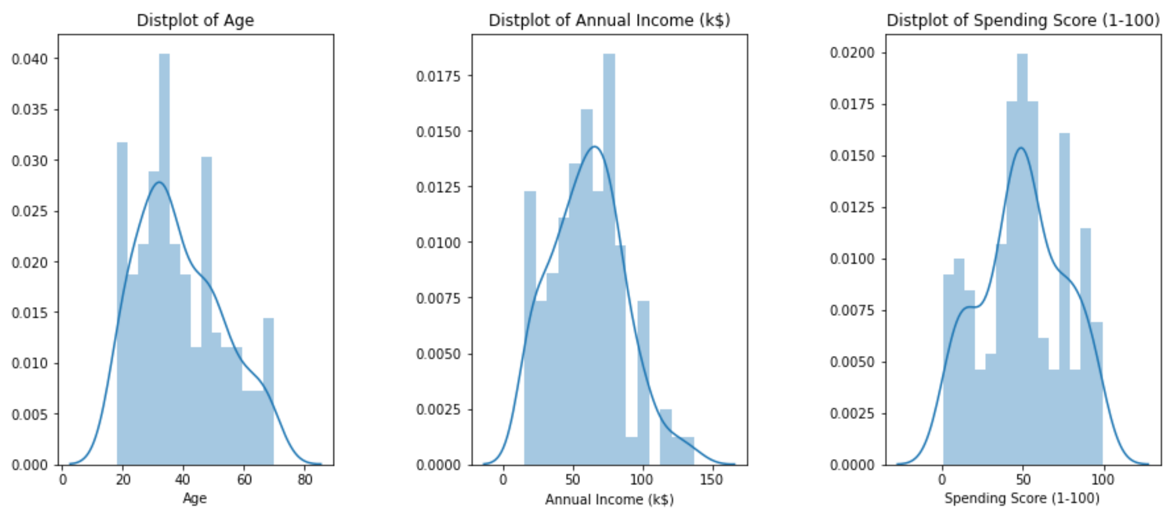
	CustomerID	Age	Annual Income (k$)	Spending Score (1-100)
count	200.000000	200.000000	200.000000	200.000000
mean	100.500000	38.850000	60.560000	50.200000
std	57.879185	13.969007	26.264721	25.823522
min	1.000000	18.000000	15.000000	1.000000
25%	50.750000	28.750000	41.500000	34.750000
50%	100.500000	36.000000	61.500000	50.000000
75%	150.250000	49.000000	78.000000	73.000000
max	200.000000	70.000000	137.000000	99.000000

Checking for null values

In [6]:
```
df.isnull().sum()
```

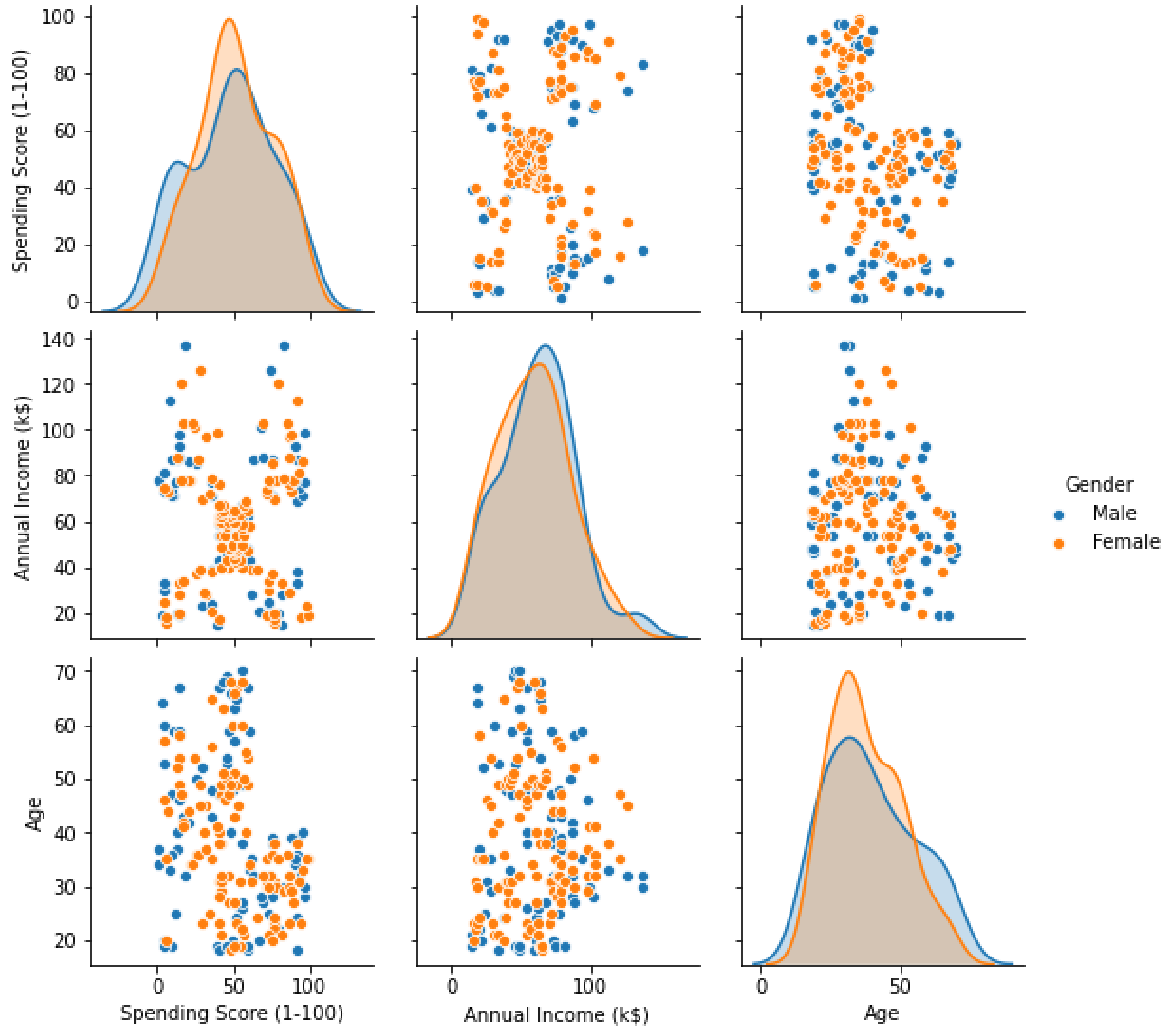
Out[6]:
```
CustomerID                0
Gender                    0
Age                       0
Annual Income  (k$)       0
Spending Score (1-100)    0
dtype: int64
```

In [7]:
```
plt.figure(1 , figsize = (15 , 6)) n = 0
for x in ['Age' , 'Annual Income (k$)' , 'Spending Score (1-100)']: n += 1
      plt.subplot(1 , 3 , n) plt.subplots_adjust(hspace = 0.5 ,
      wspace = 0.5) sns.distplot(df[x] , bins = 15)
      plt.title('Distplot of {}'.format(x))
plt.show()
```



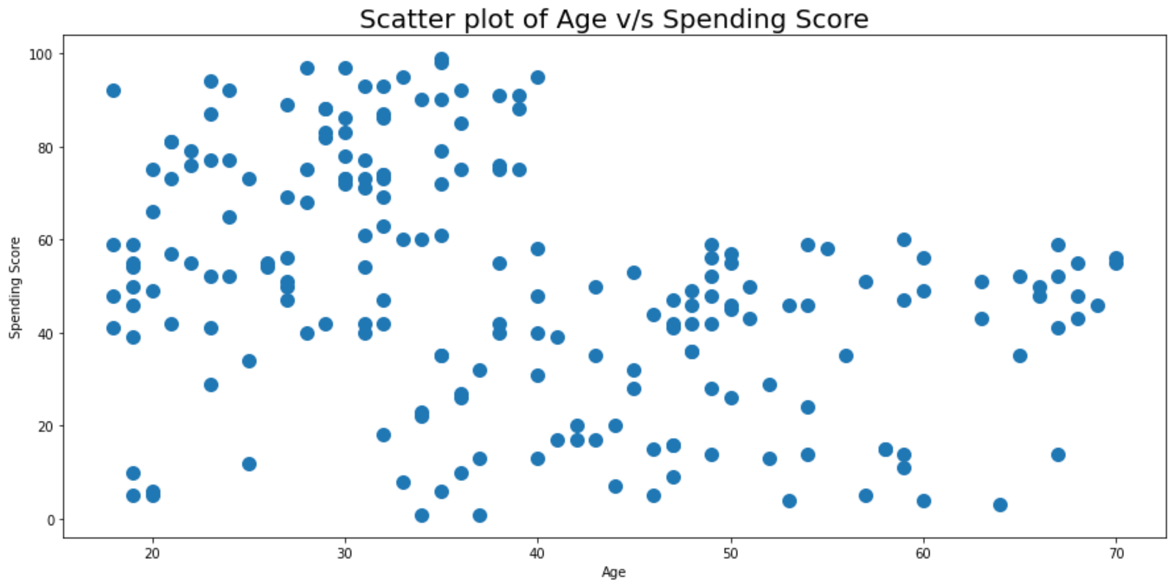
In [8]:
```
sns.pairplot(df, vars = ['Spending Score (1-100)', 'Annual Income (k
$)', 'Age'], hue = "Gender")
```

Out[8]: <seaborn.axisgrid.PairGrid at 0x7f8ac2c79890>



2D Clustering based on Age and Spending Score

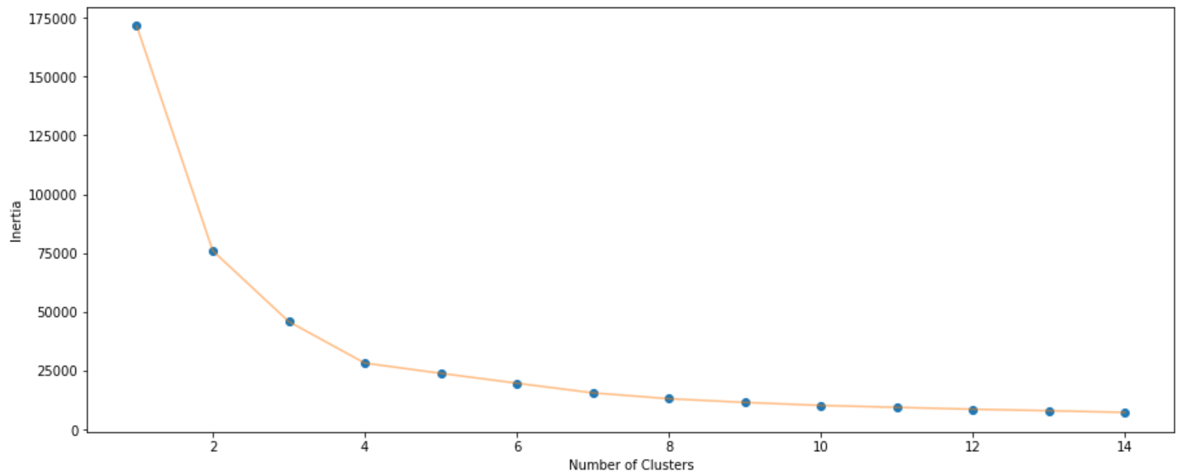
In [9]:
```
plt.figure(1 , figsize = (15 , 7))
plt.title('Scatter plot of Age v/s Spending Score', fontsize = 20)
plt.xlabel('Age')
plt.ylabel('Spending Score')
plt.scatter( x = 'Age', y = 'Spending Score (1-100)', data = df, s = 100)
plt.show()
```



Deciding K value

In [10]:
```
X1 = df[['Age' , 'Spending Score (1-100)']].iloc[: , :].values inertia = []
for n in range(1 , 15):
    algorithm = (KMeans(n_clusters = n ,init='k-means++', n_init = 10
,max_iter=300,
                        tol=0.0001,   random_state= 111   , algorithm=
'elkan') )
    algorithm.fit(X1)
    inertia.append(algorithm.inertia_)
```

In [11]:
```
plt.figure(1 , figsize = (15 ,6)) plt.plot(np.arange(1
, 15) , inertia , 'o')
plt.plot(np.arange(1 , 15) , inertia , '-' , alpha = 0.5)
plt.xlabel('Number of Clusters') , plt.ylabel('Inertia') plt.show()
```

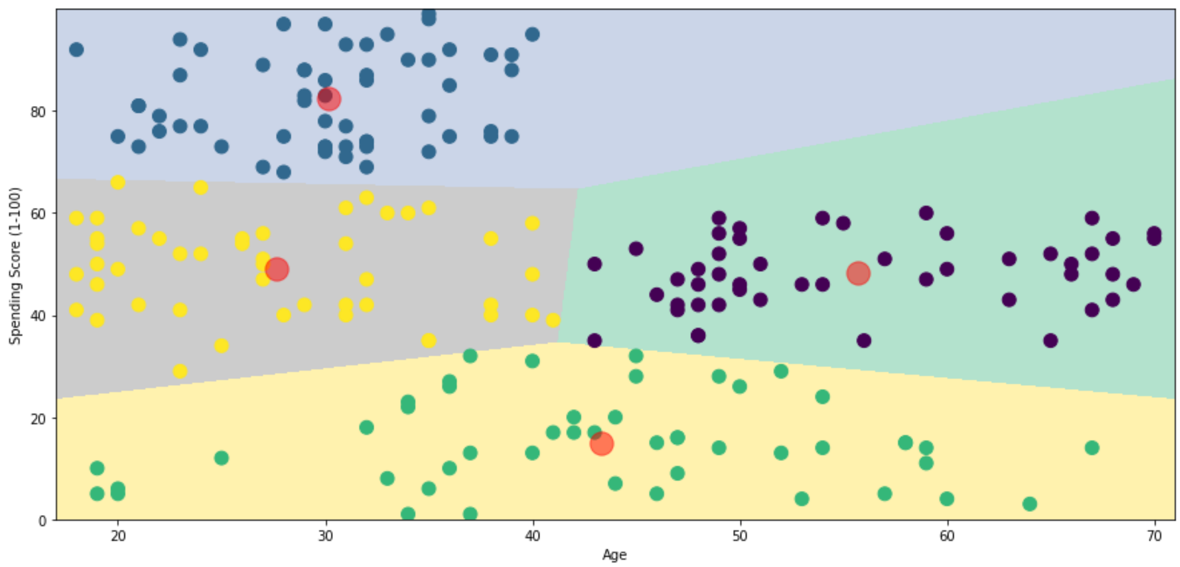


Applying KMeans for k=4

In [12]:
```
algorithm = (KMeans(n_clusters = 4 ,init='k-means++', n_init = 10 ,ma
x_iter=300,
                        tol=0.0001,   random_state= 111   , algorithm=
'elkan') )
algorithm.fit(X1)
labels1 = algorithm.labels_
centroids1 = algorithm.cluster_centers_
```

In [13]:
```
h = 0.02
x_min, x_max = X1[:, 0].min() - 1, X1[:, 0].max() + 1
y_min, y_max = X1[:, 1].min() - 1, X1[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_m ax, h))
Z = algorithm.predict(np.c_[xx.ravel(), yy.ravel()])
```

In [14]:
```
plt.figure(1 , figsize = (15 , 7) ) plt.clf()
Z = Z.reshape(xx.shape)
plt.imshow(Z , interpolation='nearest',
           extent=(xx.min(), xx.max(), yy.min(), yy.max()),
           cmap = plt.cm.Pastel2, aspect = 'auto', origin='lower')

plt.scatter( x = 'Age', y = 'Spending Score (1-100)', data = df, c = labels1, s =
100)
plt.scatter(x = centroids1[: , 0] , y =              centroids1[: , 1] , s = 300
, c = 'red' , alpha = 0.5)
plt.ylabel('Spending Score (1-100)') , plt.xlabel('Age') plt.show()
```

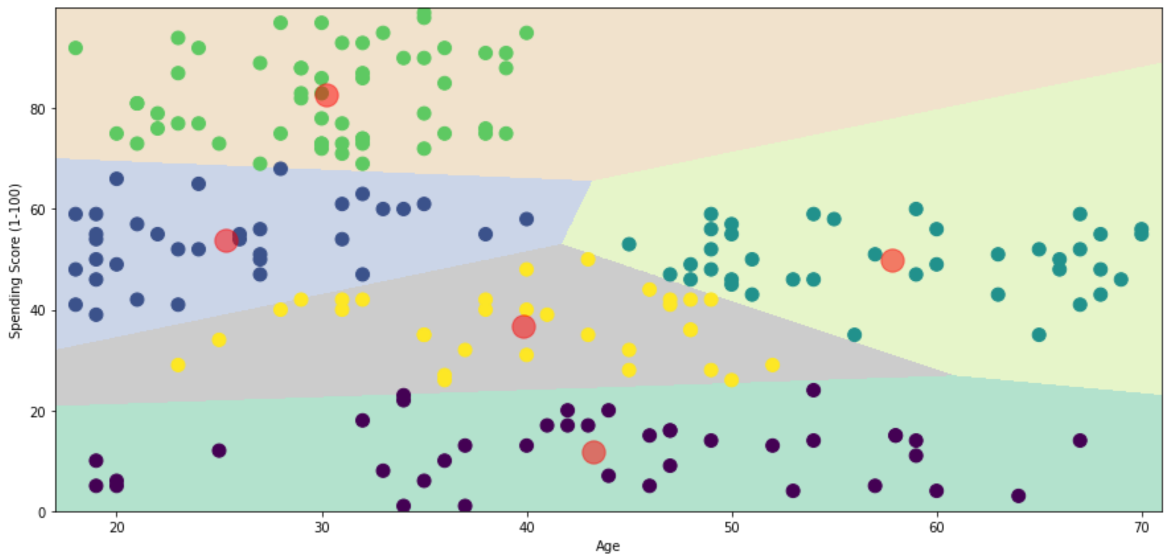


Applying KMeans for k=5

In [15]:
```
algorithm = (KMeans(n_clusters = 5, init='k-means++', n_init = 10, ma
x_iter=300,
                        tol=0.0001, random_state= 111 , algorithm='el
kan'))
algorithm.fit(X1)
labels1 = algorithm.labels_
centroids1 = algorithm.cluster_centers_
```

In [16]:
```
h = 0.02
x_min, x_max = X1[:, 0].min() - 1, X1[:, 0].max() + 1
y_min, y_max = X1[:, 1].min() - 1, X1[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_m ax, h))
Z = algorithm.predict(np.c_[xx.ravel(), yy.ravel()])
```

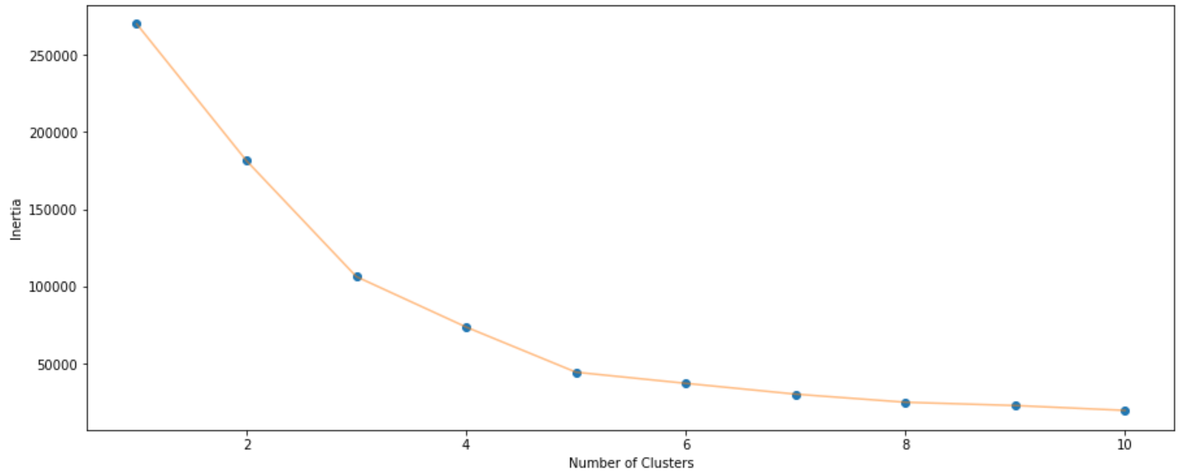
In [17]:
```
plt.figure(1 , figsize = (15 , 7) ) plt.clf()
Z = Z.reshape(xx.shape)
plt.imshow(Z , interpolation='nearest',
                extent=(xx.min(), xx.max(), yy.min(), yy.max()),
                cmap = plt.cm.Pastel2, aspect = 'auto', origin='lower')

plt.scatter( x = 'Age', y = 'Spending Score (1-100)', data = df, c = labels1, s =
100)
plt.scatter(x = centroids1[: , 0] , y =              centroids1[: , 1] , s = 300
, c = 'red' , alpha = 0.5)
plt.ylabel('Spending Score (1-100)') , plt.xlabel('Age') plt.show()
```



2D Clustering based on Annual Income and Spending Score

In [18]:
```
X2 = df[['Annual Income (k$)' , 'Spending Score (1-100)']].iloc[: ,
:].values
inertia = []
for n in range(1 , 11):
    algorithm = (KMeans(n_clusters = n ,init='k-means++', n_init = 10
,max_iter=300,
                        tol=0.0001,    random_state= 111   , algorithm=
'elkan') )
    algorithm.fit(X2)
    inertia.append(algorithm.inertia_)
```

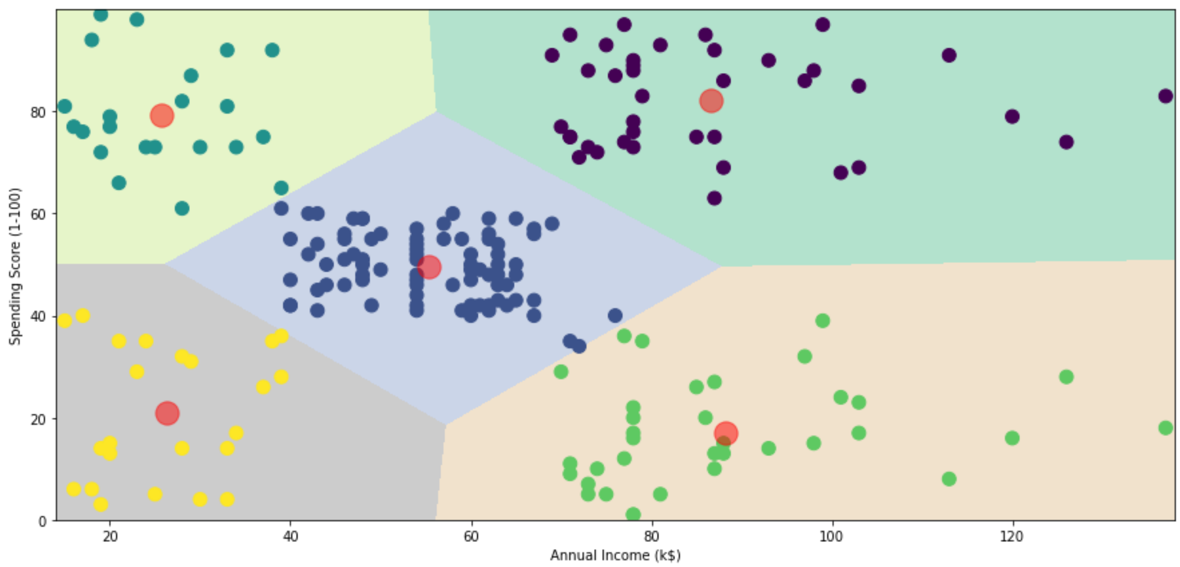
In [19]:
```
plt.figure(1 , figsize = (15 ,6)) plt.plot(np.arange(1
, 11) , inertia , 'o')
plt.plot(np.arange(1 , 11) , inertia , '-' , alpha = 0.5)
plt.xlabel('Number of Clusters') , plt.ylabel('Inertia') plt.show()
```



In [20]:
```
algorithm = (KMeans(n_clusters = 5 ,init='k-means++', n_init = 10 ,ma
x_iter=300,
                        tol=0.0001,   random_state= 111   , algorithm=
'elkan') )
algorithm.fit(X2)
labels2 = algorithm.labels_
centroids2 = algorithm.cluster_centers_
```

In [21]:
```
h = 0.02
x_min, x_max = X2[:, 0].min() - 1, X2[:, 0].max() + 1
y_min, y_max = X2[:, 1].min() - 1, X2[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_m ax, h))
Z2 = algorithm.predict(np.c_[xx.ravel(), yy.ravel()])
```

In [22]:
```
plt.figure(1 , figsize = (15 , 7) ) plt.clf()
Z2 = Z2.reshape(xx.shape)
plt.imshow(Z2 , interpolation='nearest',
                extent=(xx.min(), xx.max(), yy.min(), yy.max()),
                cmap = plt.cm.Pastel2, aspect = 'auto', origin='lower')

plt.scatter( x = 'Annual Income (k$)' ,y = 'Spending Score (1-100)' , data = df , c
= labels2 ,
                 s = 100 )
plt.scatter(x = centroids2[: , 0] , y =               centroids2[: , 1] , s = 300
, c = 'red' , alpha = 0.5)
plt.ylabel('Spending Score (1-100)') , plt.xlabel('Annual Income (k
$)')
plt.show()
```

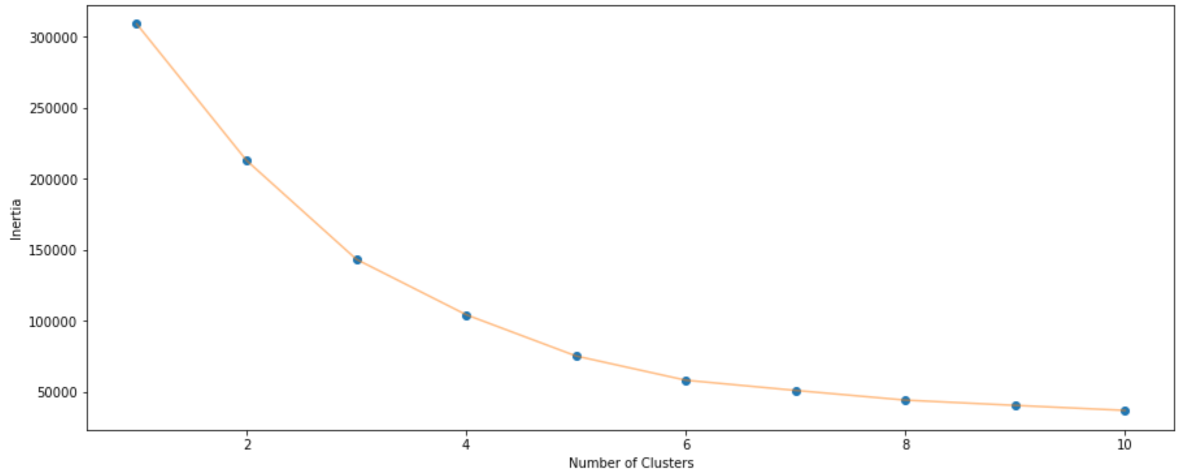


3D Clustering Age , Annual Income and Spending Score

In [23]:
```
X3 = df[['Age' , 'Annual Income (k$)' ,'Spending Score (1-100)']].ilo c[: ,
:].values
inertia = []
for n in range(1 , 11):
      algorithm = (KMeans(n_clusters = n, init='k-means++', n_init = 10
, max_iter=300,
                                    tol=0.0001, random_state= 111, algorithm='elk
an'))
      algorithm.fit(X3)
      inertia.append(algorithm.inertia_)
```

In [24]:

```
plt.figure(1 , figsize = (15 ,6)) plt.plot(np.arange(1
, 11) , inertia , 'o')
plt.plot(np.arange(1 , 11) , inertia , '-' , alpha = 0.5)
plt.xlabel('Number of Clusters') , plt.ylabel('Inertia') plt.show()
```



In [25]:

```
algorithm = (KMeans(n_clusters = 6 ,init='k-means++', n_init = 10 ,ma
x_iter=300,
                        tol=0.0001,   random_state= 111   , algorithm=
'elkan') )
algorithm.fit(X3)
labels3 = algorithm.labels_
centroids3 = algorithm.cluster_centers_

y_kmeans = algorithm.fit_predict(X3)
df['cluster'] = pd.DataFrame(y_kmeans)
df.head()
```

Out[25]:

	CustomerID	Gender	Age	Annual Income (k$)	Spending Score (1-100)	cluster
0	1	Male	19	15	39	4
1	2	Male	21	15	81	3
2	3	Female	20	16	6	4
3	4	Female	23	16	77	3
4	5	Female	31	17	40	4

Final Note

Thus, we have analysed Customer data and performed 2D and 3D clustering using K Means Algorithm. This kind of cluster analysis helps design better customer acquisition strategies and helps in business growth.

Anime recommendation based on user clustering

In [14]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

%matplotlib inline


plt.rcParams['figure.figsize'] = (6, 4)
plt.style.use('ggplot')
%config InlineBackend.figure_formats = {'png', 'retina'}
```

In [15]:
```
anime = pd.read_csv('../input/anime.csv')
```

In [16]:
```
anime.head()
```

Out[16]:

	anime_id	name	genre	type	episodes	rating	members
0	32281	Kimi no Na wa.	Drama, Romance, School, Supernatural	Movie	1	9.37	200630
1	5114	Fullmetal Alchemist: Brotherhood	Action, Adventure, Drama, Fantasy, Magic, Mili...	TV	64	9.26	793665
2	28977	Gintama°	Action, Comedy, Historical, Parody, Samurai, S...	TV	51	9.25	114262
3	9253	Steins;Gate	Sci-Fi, Thriller	TV	24	9.17	673572
4	9969	Gintama'	Action, Comedy, Historical, Parody, Samurai, S...	TV	51	9.16	151266

In [17]:
```
print(anime.shape)
```

(12294, 7)

In [18]:
```
user = pd.read_csv('../input/rating.csv')
```

In [19]:
```
user.head(10)
```

Out[19]:

	user_id	anime_id	rating
0	1	20	-1
1	1	24	-1
2	1	79	-1
3	1	226	-1
4	1	241	-1
5	1	355	-1
6	1	356	-1
7	1	442	-1
8	1	487	-1
9	1	846	-1

In [20]:
```
print(user.shape)
```

(7813737, 3)

In [21]:
```
# User 1 has a negative in rating mean

user[user['user_id']==1].rating.mean()
```

Out[21]: -0.7124183006535948

In [22]:
```
# User 2 has a very low in rating mean

user[user['user_id']==2].rating.mean()
```

Out[22]: 2.6666666666666665

In [23]:
```
# Rating mean of user 5 is very close to 5 which is half of max ratin g

user[user['user_id']==5].rating.mean()
```

Out[23]: 4.263383297644539

Calculate mean rating per user

In [24]:
```
MRPU = user.groupby(['user_id']).mean().reset_index()
MRPU['mean_rating'] = MRPU['rating']

MRPU.drop(['anime_id','rating'],axis=1, inplace=True)
```

In [25]:
```
MRPU.head(10)
```

Out[25]:

	user_id	mean_rating
0	1	-0.712418
1	2	2.666667
2	3	7.382979
3	4	-1.000000
4	5	4.263383
5	6	-1.000000
6	7	7.387755
7	8	8.333333
8	9	8.000000
9	10	2.875000

In [26]:
```
user = pd.merge(user,MRPU,on=['user_id','user_id'])
```

In [27]:
```
user.head(5)
```

Out[27]:

	user_id	anime_id	rating	mean_rating
0	1	20	-1	-0.712418
1	1	24	-1	-0.712418
2	1	79	-1	-0.712418
3	1	226	-1	-0.712418
4	1	241	-1	-0.712418

In [28]:
```
user = user.drop(user[user.rating < user.mean_rating].index)
```

In [29]:
```
# 3 anime were assigned as user 1 favorite anime

user[user['user_id']== 1].head(10)
```

Out[29]:

	user_id	anime_id	rating	mean_rating
47	1	8074	10	-0.712418
81	1	11617	10	-0.712418
83	1	11757	10	-0.712418
101	1	15451	10	-0.712418

In [30]:
```
# user2 favorite only one anime

user[user['user_id']== 2].head(10)
```

Out[30]:

	user_id	anime_id	rating	mean_rating
153	2	11771	10	2.666667

In [31]:
```
user[user['user_id']== 5].head(10)
```

Out[31]:

	user_id	anime_id	rating	mean_rating
302	5	6	8	4.263383
303	5	15	6	4.263383
304	5	17	6	4.263383
305	5	18	6	4.263383
306	5	20	6	4.263383
307	5	22	5	4.263383
310	5	45	7	4.263383
311	5	47	8	4.263383
312	5	57	7	4.263383
314	5	67	6	4.263383

In [32]:
```
print(user.shape)
```

```
(4262566, 4)
```

In [33]:
```
user["user_id"].unique()
```

Out[33]:
```
array([    1,     2,     3, ..., 73514, 73515, 73516])
```

In [34]:
```
user = user.rename({'rating':'userRating'}, axis='columns')
```

Combine two datasets

In this kernel, I decide to reduce size of dataset, because of running ti
me

In [35]:
```
# merge 2 dataset
mergedata = pd.merge(anime,user,on=['anime_id','anime_id'])
mergedata= mergedata[mergedata.user_id <= 20000]
mergedata.head(10)
```

Out[35]:

	anime_id	name	genre	type	episodes	rating	members	user_id	userRating	mea
0	32281	Kimi no Na wa.	Drama, Romance, School, Supernatural	Movie	1	9.37	200630	152	10	7
1	32281	Kimi no Na wa.	Drama, Romance, School, Supernatural	Movie	1	9.37	200630	244	10	8
2	32281	Kimi no Na wa.	Drama, Romance, School, Supernatural	Movie	1	9.37	200630	271	10	7
3	32281	Kimi no Na wa.	Drama, Romance, School, Supernatural	Movie	1	9.37	200630	322	10	8
4	32281	Kimi no Na wa.	Drama, Romance, School, Supernatural	Movie	1	9.37	200630	398	10	-0
5	32281	Kimi no Na wa.	Drama, Romance, School, Supernatural	Movie	1	9.37	200630	462	8	7
6	32281	Kimi no Na wa.	Drama, Romance, School, Supernatural	Movie	1	9.37	200630	490	10	8
7	32281	Kimi no Na wa.	Drama, Romance, School, Supernatural	Movie	1	9.37	200630	548	10	8
8	32281	Kimi no Na wa.	Drama, Romance, School, Supernatural	Movie	1	9.37	200630	570	10	8
9	32281	Kimi no Na wa.	Drama, Romance, School, Supernatural	Movie	1	9.37	200630	598	10	8

In [36]:
```
len(mergedata['anime_id'].unique())
```

Out[36]: 7852

In [37]:
```
len(anime['anime_id'].unique())
```
Out[37]:
```
12294
```

Create Crosstable

Show detail of anime which each user like

In [38]:
```
user_anime = pd.crosstab(mergedata['user_id'], mergedata['name'])
user_anime.head(10)
```
Out[38]:

name / user_id	"Bungaku Shoujo" Kyou no Oyatsu: Hatsukoi	"Bungaku Shoujo" Memoire	"Bungaku Shoujo" Movie	"Eiji"	.hack//G.U. Returner	.hack//G Tril
1	0	0	0	0	0	
2	0	0	0	0	0	
3	0	0	0	0	0	
4	0	0	0	0	0	
5	0	0	0	0	0	
6	0	0	0	0	0	
7	0	0	0	0	0	
8	0	0	0	0	0	
9	0	0	0	0	0	
10	0	0	0	0	0	

In [39]:
```
user_anime.shape
```
Out[39]:
```
(20000, 7852)
```

Principal component analysis

In [40]:
```
from sklearn.decomposition import PCA

pca = PCA(n_components=3)
pca.fit(user_anime)
pca_samples = pca.transform(user_anime)
```

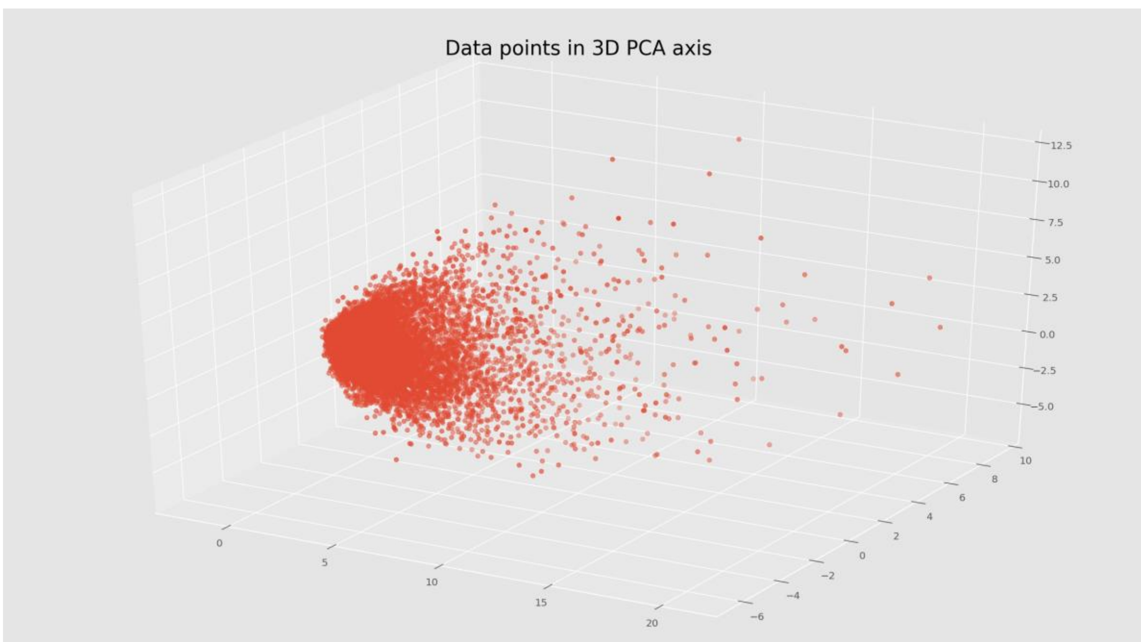
In [41]:
```
ps = pd.DataFrame(pca_samples)
ps.head()
```

Out[41]:

	0	1	2
0	-1.579129	-0.500240	0.415772
1	-1.773553	-0.272593	0.116393
2	0.218814	-1.232282	-0.985795
3	0.199435	-0.291005	0.681026
4	3.532125	-0.184797	-0.743374

In [42]:
```
tocluster = pd.DataFrame(ps[[0,1,2]])
```

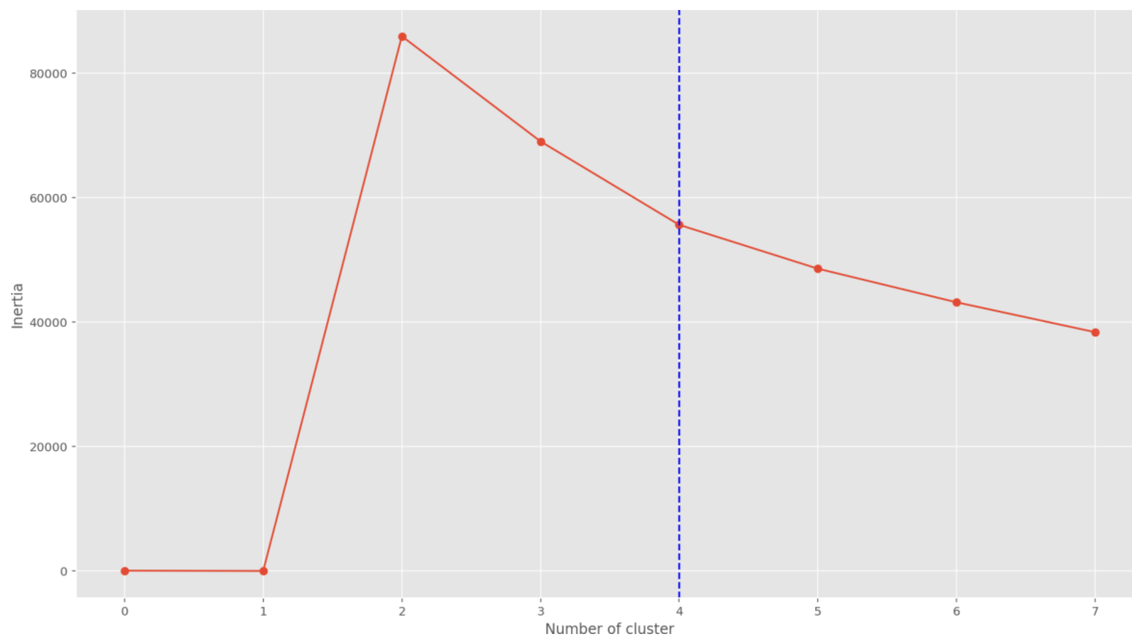
In [43]:
```
plt.rcParams['figure.figsize'] = (16, 9)


fig = plt.figure() ax
= Axes3D(fig)
ax.scatter(tocluster[0], tocluster[2], tocluster[1])

plt.title('Data points in 3D PCA axis', fontsize=20) plt.show()
```



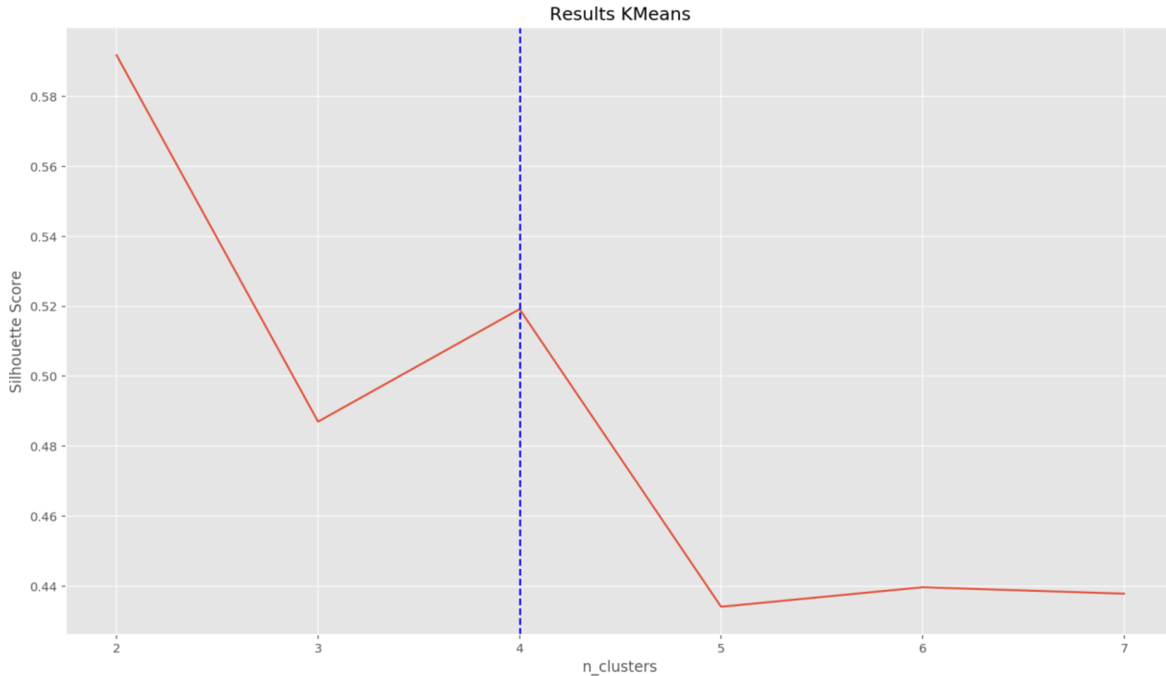
Selecting number of k

In [44]:
```
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score

scores = []
inertia_list = np.empty(8)

for i in range(2,8):
    kmeans = KMeans(n_clusters=i)
    kmeans.fit(tocluster) inertia_list[i] =
    kmeans.inertia_
    scores.append(silhouette_score(tocluster, kmeans.labels_))
```

In [45]:
```
plt.plot(range(0,8),inertia_list,'-o') plt.xlabel('Number
of cluster') plt.axvline(x=4, color='blue', linestyle='--')
plt.ylabel('Inertia')
plt.show()
```



In [46]:
```
plt.plot(range(2,8), scores);
plt.title('Results KMeans')
plt.xlabel('n_clusters');
plt.axvline(x=4, color='blue', linestyle='--')
plt.ylabel('Silhouette Score');
plt.show()
```

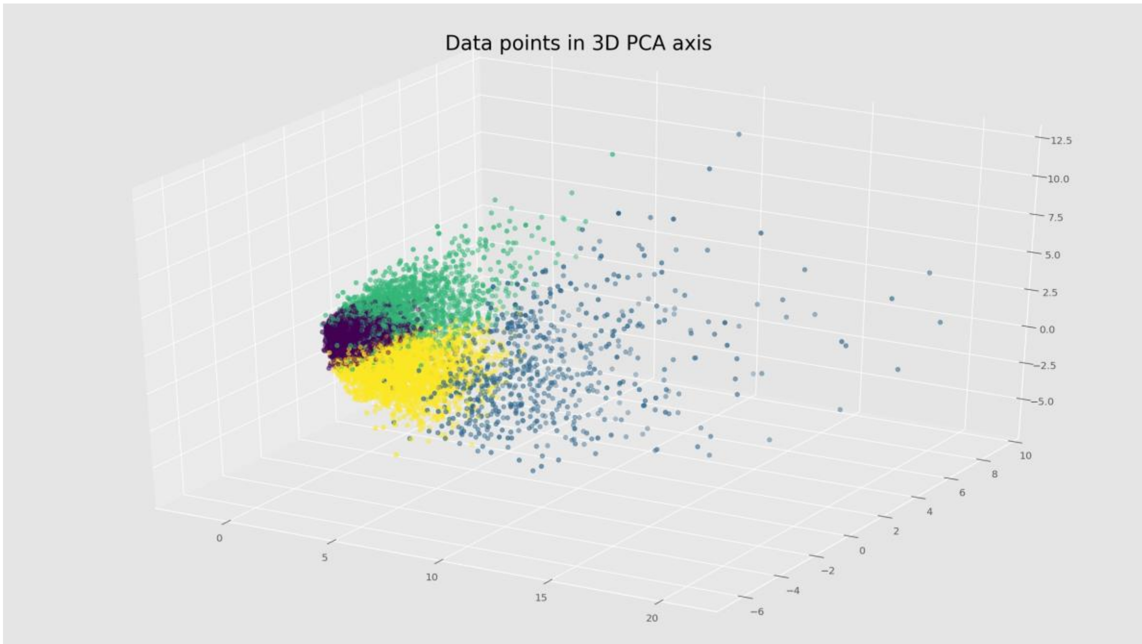


K means clustering

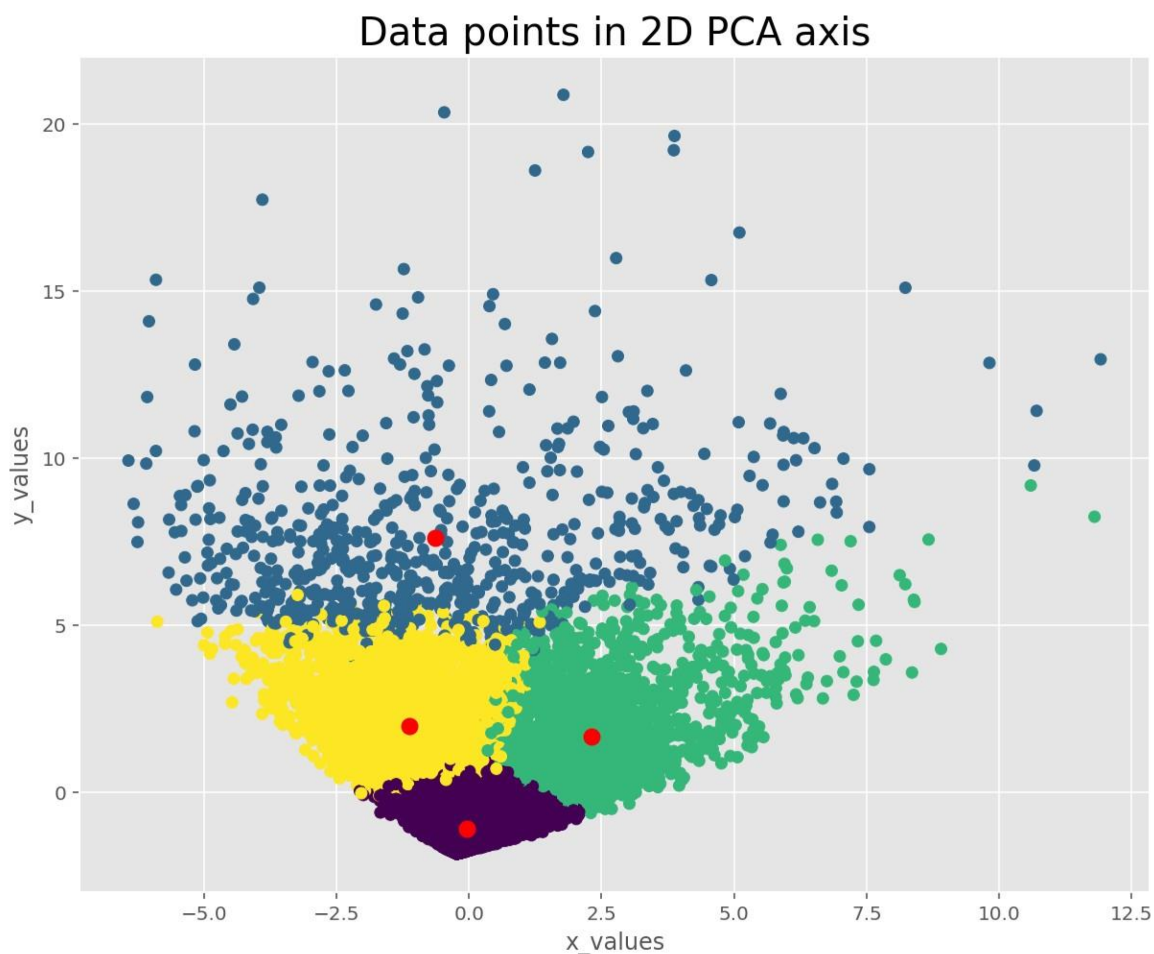
In [47]:
```
from sklearn.cluster import KMeans

clusterer = KMeans(n_clusters=4,random_state=30).fit(tocluster) centers
= clusterer.cluster_centers_
c_preds = clusterer.predict(tocluster)

print(centers)
```

```
[[-1.08874971  -0.04026584   0.06666433]
 [ 7.61700382  -0.64256859   0.83955247]
 [ 1.6784451    2.31533837  -0.02522808]
 [ 1.97875213  -1.12654215  -0.4351448 ]]
```

In [48]:
```
fig = plt.figure() ax
= Axes3D(fig)
ax.scatter(tocluster[0], tocluster[2], tocluster[1], c = c_preds) plt.title('Data
points in 3D PCA axis', fontsize=20)

plt.show()
```



In [49]:
```
fig = plt.figure(figsize=(10,8))
plt.scatter(tocluster[1],tocluster[0],c = c_preds) for ci,c in
enumerate(centers):
    plt.plot(c[1], c[0], 'o', markersize=8, color='red', alpha=1)

plt.xlabel('x_values')
plt.ylabel('y_values')

plt.title('Data points in 2D PCA axis', fontsize=20) plt.show()
```



In [50]:

```python
user_anime['cluster'] = c_preds


user_anime.head(10)
```

Out[50]:

name	"Bungaku Shoujo" Kyou no Oyatsu: Hatsukoi	"Bungaku Shoujo" Memoire	"Bungaku Shoujo" Movie	"Eiji"	.hack//G.U. Returner	.hack//G Tril
user_id						
1	0	0	0	0	0	
2	0	0	0	0	0	
3	0	0	0	0	0	
4	0	0	0	0	0	
5	0	0	0	0	0	
6	0	0	0	0	0	
7	0	0	0	0	0	
8	0	0	0	0	0	
9	0	0	0	0	0	
10	0	0	0	0	0	

SVM with sklearn

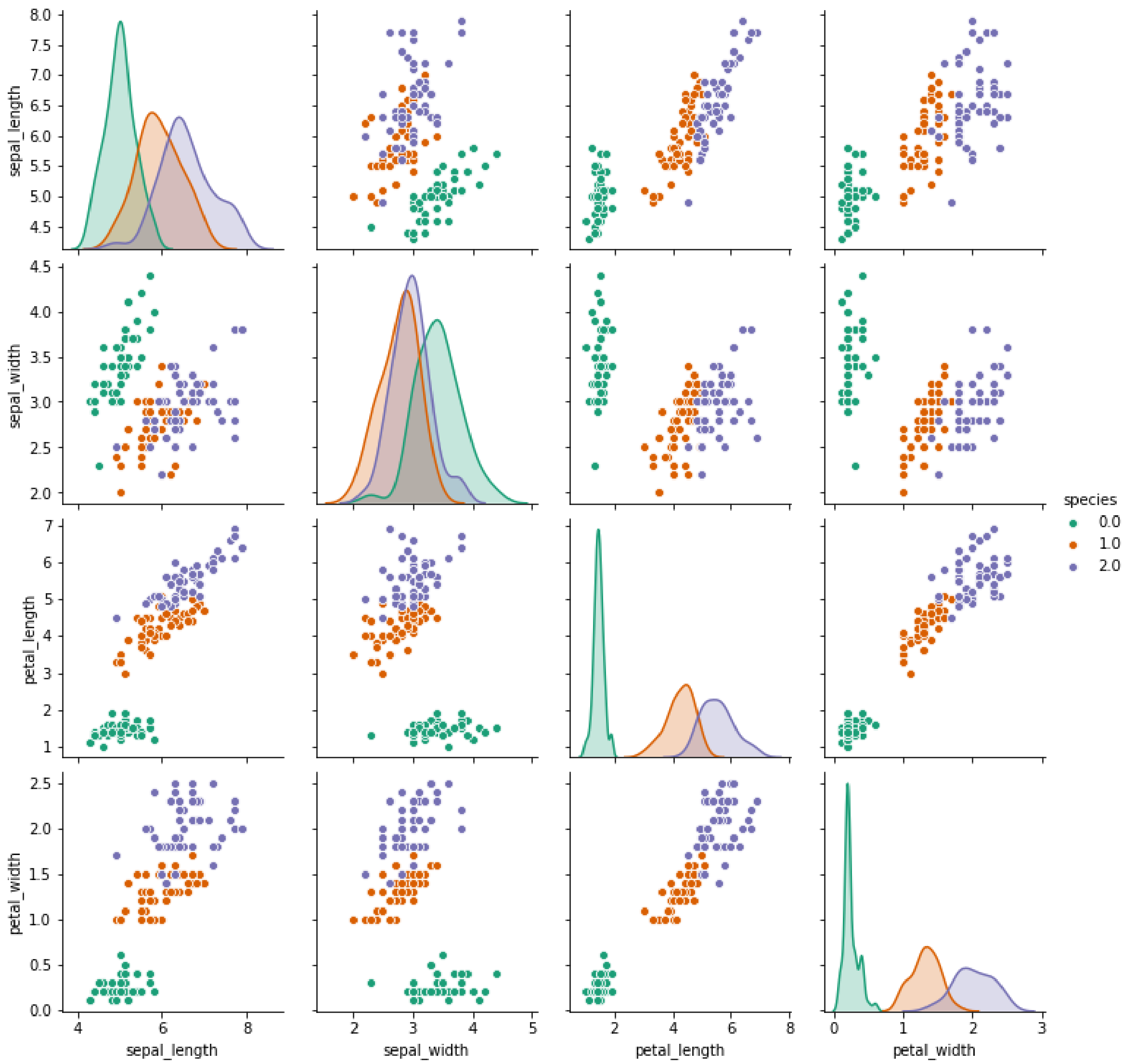
In [15]:

```
# Import the necessary libraries
import pandas as pd
import numpy as np
from sklearn.datasets import load_iris

# Import data visualisation libraries
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

# Import model libraries
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import classification_report, confusion_matrix
```

In [3]:

```
# Load the dataset
iris = load_iris()

# Seperate the features and target variables
X = iris.data
y = iris.target data =
np.c_[X, y]

# Make a header list
cols = ['sepal_length', 'sepal_width', 'petal_length', 'petal_width'] header = cols +
['species']

iris = pd.DataFrame(data=data, columns=header)
```

In [5]:
```
sns.pairplot(iris,hue='species',palette='Dark2');
```



In [8]:
```
X = iris.drop('species',axis=1) y =
iris['species']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=
0.2,random_state=101)
```

Train the model

In [10]:
```
model = SVC()
model.fit(X_train, y_train)
```

Out[10]: SVC()

In [11]:
```
preds = model.predict(X_test)
```

In [13]:
```
print(confusion_matrix(y_test,preds))
```

```
[[10  0  0]
 [ 0 12  0]
 [ 0  1  7]]
```

In [14]:
```
print(classification_report(y_test,preds))
```

```
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00        10
         1.0       0.92      1.00      0.96        12
         2.0       1.00      0.88      0.93         8

    accuracy                           0.97        30
   macro avg       0.97      0.96      0.96        30
weighted avg       0.97      0.97      0.97        30
```

Here we get an accuracy of 0.97 with precision at 1.

In [ ]:

SVM From Scratch

In [10]:
```
import numpy as np
import pandas as pd
import statsmodels.api as sm
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split as tts
from sklearn.metrics import accuracy_score, recall_score, precision_s core
from sklearn.utils import shuffle
```

In [11]:
```
def remove_correlated_features(X):
    corr_threshold = 0.9
    corr = X.corr()
    drop_columns = np.full(corr.shape[0], False, dtype=bool)
    for i in range(corr.shape[0]):
        for j in range(i + 1, corr.shape[0]):
            if corr.iloc[i, j] >= corr_threshold:
                drop_columns[j] = True
    columns_dropped = X.columns[drop_columns]
    X.drop(columns_dropped, axis=1, inplace=True)
    return columns_dropped
```

In [12]:
```
def remove_less_significant_features(X, Y): sl =
    0.05
    regression_ols = None
    columns_dropped = np.array([])
    for itr in range(0, len(X.columns)): regression_ols =
        sm.OLS(Y, X).fit() max_col =
        regression_ols.pvalues.idxmax() max_val =
        regression_ols.pvalues.max()
        if max_val > sl:
            X.drop(max_col, axis='columns', inplace=True)
            columns_dropped = np.append(columns_dropped, [max_col])
        else:
            break
    regression_ols.summary()
    return columns_dropped
```

In [13]:
```
def compute_cost(W, X, Y): #
    calculate hinge loss N =
    X.shape[0]
    distances = 1 - Y * (np.dot(X, W))
    distances[distances < 0] = 0        # equivalent to max(0, distance)
    hinge_loss = regularization_strength * (np.sum(distances) / N)

    # calculate cost
    cost = 1 / 2 * np.dot(W, W) + hinge_loss
    return cost
```

In [14]:
```
def calculate_cost_gradient(W, X_batch, Y_batch):
    # if only one example is passed (eg. in case of SGD)
    if type(Y_batch) == np.float64:
        Y_batch = np.array([Y_batch])
        X_batch = np.array([X_batch])     # gives multidimensional array

    distance = 1 - (Y_batch * np.dot(X_batch, W)) dw =
    np.zeros(len(W))

    for ind, d in enumerate(distance):
        if max(0, d) == 0: di
            = W
        else:
            di = W - (regularization_strength * Y_batch[ind] * X_batc
h[ind])
        dw += di

    dw = dw/len(Y_batch)     # average
    return dw
```

In [15]:
```
def sgd(features, outputs):
    max_epochs = 5000
    weights = np.zeros(features.shape[1]) nth
    = 0
    prev_cost = float("inf") cost_threshold =
    0.01                    # in percent
    # stochastic gradient descent
    for epoch in range(1, max_epochs):
        # shuffle to prevent repeating update cycles
        X, Y = shuffle(features, outputs)
        for ind, x in enumerate(X):
            ascent = calculate_cost_gradient(weights, x, Y[ind]) weights
            = weights - (learning_rate * ascent)

        # convergence check on 2^nth epoch
        if epoch == 2 ** nth or epoch == max_epochs - 1: cost =
            compute_cost(weights, features, outputs)
            print("Epoch is: {} and Cost is: {}".format(epoch, cost))
            # stoppage criterion
            if abs(prev_cost - cost) < cost_threshold * prev_cost:
                return weights
            prev_cost = cost nth
            += 1
    return weights
```

In [18]:
```
def init():
    print("reading dataset...")
    # read data in pandas (pd) data frame
    data = pd.read_csv('./data.csv')

    # drop last column (extra column added by pd) #
    and unnecessary first column (id)
    data.drop(data.columns[[-1, 0]], axis=1, inplace=True)

    print("applying feature engineering...") #
    convert categorical labels to numbers
    diag_map = {'M': 1.0, 'B': -1.0}
    data['diagnosis'] = data['diagnosis'].map(diag_map)

    # put features & outputs in different data frames
    Y = data.loc[:, 'diagnosis'] X =
    data.iloc[:, 1:]

    # filter features
    remove_correlated_features(X)
    remove_less_significant_features(X, Y)

    # normalize data for better convergence and to prevent overflow
    X_normalized = MinMaxScaler().fit_transform(X.values) X =
    pd.DataFrame(X_normalized)

    # insert 1 in every row for intercept b
    X.insert(loc=len(X.columns), column='intercept', value=1)

    # split data into train and test set
    print("splitting dataset into train and test sets...")
    X_train, X_test, y_train, y_test = tts(X, Y, test_size=0.2, rando
m_state=42)

    # train the model
    print("training started...")
    W = sgd(X_train.to_numpy(), y_train.to_numpy())
    print("training finished.")
    print("weights are: {}".format(W))

    # testing the model print("testing the
    model...") y_train_predicted =
    np.array([]) for i in
    range(X_train.shape[0]):
        yp = np.sign(np.dot(X_train.to_numpy()[i], W))
        y_train_predicted = np.append(y_train_predicted, yp)

    y_test_predicted = np.array([])
    for i in range(X_test.shape[0]):
        yp = np.sign(np.dot(X_test.to_numpy()[i], W))
        y_test_predicted = np.append(y_test_predicted, yp)

    print("accuracy on test dataset: {}".format(accuracy_score(y_test
, y_test_predicted)))
```

```
    print(
"recall on
test
dataset:
{}".format
(recall_sco
re(y_test,
y_
test_predic
ted)))
```

```
    print("precision on test dataset: {}".format(recall_score(y_test,
y_test_predicted)))
```

In [19]:

```
# set hyper-parameters and call init
regularization_strength = 10000
learning_rate = 0.000001
init()
```

```
reading dataset...
applying feature engineering...
splitting dataset into train and test sets... training
started...
Epoch is: 1 and Cost is: 7248.9355732240265 Epoch
is: 2 and Cost is: 6614.984942398622 Epoch is: 4 and
Cost is: 5434.982795519678 Epoch is: 8 and Cost is:
3824.280833205177 Epoch is: 16 and Cost is:
2669.877066596218 Epoch is: 32 and Cost is:
1958.1489098233037 Epoch is: 64 and Cost is:
1588.7039461302384 Epoch is: 128 and Cost is:
1330.659617566685 Epoch is: 256 and Cost is:
1159.9686398419585 Epoch is: 512 and Cost is:
1074.9691665529758 Epoch is: 1024 and Cost is:
1048.1230628482883 Epoch is: 2048 and Cost is:
1044.6875887383574 training finished.
weights are: [ 3.53571049 11.0564073   -2.27828017 -7.90383862 10.1560
1742 -1.29543824
  -6.43649506   2.26580158 -3.87138135   3.24581543   4.94961304   4.83535
89
  -4.70176986]
testing the model...
accuracy on test dataset: 0.9912280701754386
recall on test dataset: 0.9767441860465116
precision on test dataset: 0.9767441860465116
```

Here we can see that we get an accuracy of 0.99 along with 0.97 recall and 0.97 precision.

In [ ]:

Decision Tree on Iris dataset using sklearn

In [1]:
```
#importing respective libraries and setting up the enviornment

'''data working libraries'''
import pandas as pd
import numpy as np
from sklearn.datasets import load_iris

'''data visualisation libraries'''
import matplotlib.pyplot as plt
import seaborn as sns
import plotly as py
py.offline.init_notebook_mode(connected=True)
%matplotlib inline
```

In [2]:
```
#loading the data
iris = load_iris()
```

In [3]:
```
#setting up our x and y variables correspondingly
x=iris.data
y=iris.target
```

In [4]:
```
#concatinating the x and y np arrays into a single np array so that c an be
#converted to a dataframe later on
data=np.c_[x,y]
```

In [5]:
```
#making a header list for corresponding column indices in DF
cols=['sepal_length','sepal_width','petal_length','petal_width']
header=cols+['species']
#converting into a dataframe for visualisation purposes
iris_df=pd.DataFrame(data=data,columns=header)
```

In [6]:
```
#updating values 0,1,2 in species column with real names
iris_df.species.replace(0.0,'iris-setosa',inplace=True)
iris_df.species.replace(1.0,'iris-versicolor',inplace=True)
iris_df.species.replace(2.0,'iris-virginica',inplace=True)
```

Analysing The Data

In [7]:
```
'''our dataset has 150 datapoints(entries) and 4 featues'''
iris_df.shape
```

Out[7]: (150, 5)

Writing and Visualizing Our Own Decision Tress

In [8]:
```
class Question:
    #initialise column and value variables-> #eg->if
    ques is ->is sepal_length>=1cm then
    #sepal_length==col and 1cm=value
    def__init__(self,column,value):
        self.column=column
        self.value=value
    #it matches wheter the given data is in accordace with the value set or not
    #returns true and false accordingly
    def match(self,data):
        value=data[self.column]
        return value>=self.value
    # This is just a helper method to print # the
    question in a readable format.
    def__repr__(self):
        condition = ">="
        return "Is %s %s %s?" % (
            header[self.column], condition, str(self.value))
```

In [9]:
```
'demo of question class'
#forming a question
Question(0,5)
## it takes column as 0 and value as 5
q=Question(0,5)
#now it checks wheter the values on 0th column of the 4th datapoint i s >= 5 or
not
#and returns true or false accordingly
q.match(x[3])
```

Out[9]: False

In [10]:
```
#count the unique values of labels and store them in a dictionary
def count_values(rows):
    #will return a dictionary with species values as key and frequenc y as values
    count={}
    #takes whole dataset in as argument
    for row in    rows:
        #traverse on each datapoint
        label=row[-1]
        #labels are in the last column
        #if label is not even once come initialise it
        if label not in count:
            count[label]=0
        #increase the count of present label by 1
        count[label]+=1
    return count
```

In [11]:
```
'''demo count function'''
count_values(data) #hinglish
comment
#haar row main jayega -> last element ko label se initialise karega->
```

Out[11]: {0.0: 50, 1.0: 50, 2.0: 50}

In [12]:
```
#spliting the data based on the respective ques.
def partition(rows,question):
    #intialise two seprate lists
    true_row,false_row=[],[]
    for row in rows:
        #traverse on each datapoint
        #match the given datapoint with the respective question
        if question.match(row):
            #if question.match returns true aka value is satisfied #append
            the given row in true row list true_row.append(row)
        else:
            false_row.append(row)
    return true_row,false_row
```

In [13]:
```
#demo of partition function
#our question is ->
print(Question(0,5))
#t_r represents true_rows and f_r false_rows
t_r,f_r=partition(data,Question(0,5))
#thus t_r will only contain sepal legnth values > 5cm
t_r
```

Is sepal_length >= 5?

Out[13]:

```
[array([5.1, 3.5,  1.4,  0.2, 0.  ]),
 array([5. , 3.6,  1.4,  0.2, 0.  ]),
 array([5.4, 3.9,  1.7,  0.4, 0.  ]),
 array([5. , 3.4,  1.5,  0.2, 0.  ]),
 array([5.4, 3.7,  1.5,  0.2, 0.  ]),
 array([5.8, 4. ,  1.2,  0.2, 0.  ]),
 array([5.7, 4.4,  1.5,  0.4, 0.  ]),
 array([5.4, 3.9,  1.3,  0.4, 0.  ]),
 array([5.1, 3.5,  1.4,  0.3, 0.  ]),
 array([5.7, 3.8,  1.7,  0.3, 0.  ]),
 array([5.1, 3.8,  1.5,  0.3, 0.  ]),
 array([5.4, 3.4,  1.7,  0.2, 0.  ]),
 array([5.1, 3.7,  1.5,  0.4, 0.  ]),
 array([5.1, 3.3,  1.7,  0.5, 0.  ]),
 array([5. , 3. ,  1.6,  0.2, 0.  ]),
 array([5. , 3.4,  1.6,  0.4, 0.  ]),
 array([5.2, 3.5,  1.5,  0.2, 0.  ]),
 array([5.2, 3.4,  1.4,  0.2, 0.  ]),
 array([5.4, 3.4,  1.5,  0.4, 0.  ]),
 array([5.2, 4.1,  1.5,  0.1, 0.  ]),
 array([5.5, 4.2,  1.4,  0.2, 0.  ]),
 array([5. , 3.2,  1.2,  0.2, 0.  ]),
 array([5.5, 3.5,  1.3,  0.2, 0.  ]),
 array([5.1, 3.4,  1.5,  0.2, 0.  ]),
 array([5. , 3.5,  1.3,  0.3, 0.  ]),
 array([5. , 3.5,  1.6,  0.6, 0.  ]),
 array([5.1, 3.8,  1.9,  0.4, 0.  ]),
 array([5.1, 3.8,  1.6,  0.2, 0.  ]),
 array([5.3, 3.7,  1.5,  0.2, 0.  ]),
 array([5. , 3.3,  1.4,  0.2, 0.  ]),
 array([7. , 3.2,  4.7,  1.4, 1.  ]),
 array([6.4, 3.2,  4.5,  1.5, 1.  ]),
 array([6.9, 3.1,  4.9,  1.5, 1.  ]),
 array([5.5, 2.3,  4. ,  1.3, 1.  ]),
 array([6.5, 2.8,  4.6,  1.5, 1.  ]),
 array([5.7, 2.8,  4.5,  1.3, 1.  ]),
 array([6.3, 3.3,  4.7,  1.6, 1.  ]),
 array([6.6, 2.9,  4.6,  1.3, 1.  ]),
 array([5.2, 2.7,  3.9,  1.4, 1.  ]),
 array([5. , 2. ,  3.5,  1. , 1.  ]),
 array([5.9, 3. ,  4.2,  1.5, 1.  ]),
 array([6. , 2.2,  4. ,  1. , 1.  ]),
 array([6.1, 2.9,  4.7,  1.4, 1.  ]),
 array([5.6, 2.9,  3.6,  1.3, 1.  ]),
 array([6.7, 3.1,  4.4,  1.4, 1.  ]),
 array([5.6, 3. ,  4.5,  1.5, 1.  ]),
 array([5.8, 2.7,  4.1,  1. , 1.  ]),
 array([6.2, 2.2,  4.5,  1.5, 1.  ]),
 array([5.6, 2.5,  3.9,  1.1, 1.  ]),
 array([5.9, 3.2,  4.8,  1.8, 1.  ]),
 array([6.1, 2.8,  4. ,  1.3, 1.  ]),
 array([6.3, 2.5,  4.9,  1.5, 1.  ]),
 array([6.1, 2.8,  4.7,  1.2, 1.  ]),
 array([6.4, 2.9,  4.3,  1.3, 1.  ]),
 array([6.6, 3. ,  4.4,  1.4, 1.  ]),
 array([6.8, 2.8,  4.8,  1.4, 1.  ]),
 array([6.7, 3. ,  5. ,  1.7, 1.  ]),
```

```
 array([6. ,  2.9,  4.5,  1.5,  1.  ]),
 array([5.7,  2.6,  3.5,  1. ,  1.  ]),
 array([5.5,  2.4,  3.8,  1.1,  1.  ]),
 array([5.5,  2.4,  3.7,  1. ,  1.  ]),
 array([5.8,  2.7,  3.9,  1.2,  1.  ]),
 array([6. ,  2.7,  5.1,  1.6,  1.  ]),
 array([5.4,  3. ,  4.5,  1.5,  1.  ]),
 array([6. ,  3.4,  4.5,  1.6,  1.  ]),
 array([6.7,  3.1,  4.7,  1.5,  1.  ]),
 array([6.3,  2.3,  4.4,  1.3,  1.  ]),
 array([5.6,  3. ,  4.1,  1.3,  1.  ]),
 array([5.5,  2.5,  4. ,  1.3,  1.  ]),
 array([5.5,  2.6,  4.4,  1.2,  1.  ]),
 array([6.1,  3. ,  4.6,  1.4,  1.  ]),
 array([5.8,  2.6,  4. ,  1.2,  1.  ]),
 array([5. ,  2.3,  3.3,  1. ,  1.  ]),
 array([5.6,  2.7,  4.2,  1.3,  1.  ]),
 array([5.7,  3. ,  4.2,  1.2,  1.  ]),
 array([5.7,  2.9,  4.2,  1.3,  1.  ]),
 array([6.2,  2.9,  4.3,  1.3,  1.  ]),
 array([5.1,  2.5,  3. ,  1.1,  1.  ]),
 array([5.7,  2.8,  4.1,  1.3,  1.  ]),
 array([6.3,  3.3,  6. ,  2.5,  2.  ]),
 array([5.8,  2.7,  5.1,  1.9,  2.  ]),
 array([7.1,  3. ,  5.9,  2.1,  2.  ]),
 array([6.3,  2.9,  5.6,  1.8,  2.  ]),
 array([6.5,  3. ,  5.8,  2.2,  2.  ]),
 array([7.6,  3. ,  6.6,  2.1,  2.  ]),
 array([7.3,  2.9,  6.3,  1.8,  2.  ]),
 array([6.7,  2.5,  5.8,  1.8,  2.  ]),
 array([7.2,  3.6,  6.1,  2.5,  2.  ]),
 array([6.5,  3.2,  5.1,  2. ,  2.  ]),
 array([6.4,  2.7,  5.3,  1.9,  2.  ]),
 array([6.8,  3. ,  5.5,  2.1,  2.  ]),
 array([5.7,  2.5,  5. ,  2. ,  2.  ]),
 array([5.8,  2.8,  5.1,  2.4,  2.  ]),
 array([6.4,  3.2,  5.3,  2.3,  2.  ]),
 array([6.5,  3. ,  5.5,  1.8,  2.  ]),
 array([7.7,  3.8,  6.7,  2.2,  2.  ]),
 array([7.7,  2.6,  6.9,  2.3,  2.  ]),
 array([6. ,  2.2,  5. ,  1.5,  2.  ]),
 array([6.9,  3.2,  5.7,  2.3,  2.  ]),
 array([5.6,  2.8,  4.9,  2. ,  2.  ]),
 array([7.7,  2.8,  6.7,  2. ,  2.  ]),
 array([6.3,  2.7,  4.9,  1.8,  2.  ]),
 array([6.7,  3.3,  5.7,  2.1,  2.  ]),
 array([7.2,  3.2,  6. ,  1.8,  2.  ]),
 array([6.2,  2.8,  4.8,  1.8,  2.  ]),
 array([6.1,  3. ,  4.9,  1.8,  2.  ]),
 array([6.4,  2.8,  5.6,  2.1,  2.  ]),
 array([7.2,  3. ,  5.8,  1.6,  2.  ]),
 array([7.4,  2.8,  6.1,  1.9,  2.  ]),
 array([7.9,  3.8,  6.4,  2. ,  2.  ]),
 array([6.4,  2.8,  5.6,  2.2,  2.  ]),
 array([6.3,  2.8,  5.1,  1.5,  2.  ]),
 array([6.1,  2.6,  5.6,  1.4,  2.  ]),
 array([7.7,  3. ,  6.1,  2.3,  2.  ]),
```

```
 array([6.3,  3.4,  5.6,  2.4,  2.  ]),
 array([6.4,  3.1,  5.5,  1.8,  2.  ]),
 array([6. ,  3. ,  4.8,  1.8,  2.  ]),
 array([6.9,  3.1,  5.4,  2.1,  2.  ]),
 array([6.7,  3.1,  5.6,  2.4,  2.  ]),
 array([6.9,  3.1,  5.1,  2.3,  2.  ]),
 array([5.8,  2.7,  5.1,  1.9,  2.  ]),
 array([6.8,  3.2,  5.9,  2.3,  2.  ]),
 array([6.7,  3.3,  5.7,  2.5,  2.  ]),
 array([6.7,  3. ,  5.2,  2.3,  2.  ]),
 array([6.3,  2.5,  5. ,  1.9,  2.  ]),
 array([6.5,  3. ,  5.2,  2. ,  2.  ]),
 array([6.2,  3.4,  5.4,  2.3,  2.  ]),
 array([5.9,  3. ,  5.1,  1.8,  2.  ])]
```

In [14]:

```
#now we need some method by which we can quantify this right question #we
are talking about.For this we use various methods like->
```

In [15]:

```
#entropy is basically a measure of chaos-randomness
def entropy(rows):
    #initialise entropy
    entropy=0
    from math import log
    #calculating log(x) in base 2
    log2=lambda x:log(x)/log(2)
    count=count_values(rows)
    #storing and traversing the dictionary
    for label in count:
        #probablity of each unique label
        p=count[label]/float(len(rows))
        #calculating entropy
        entropy-=p*log2(p)
    return entropy
```

In [16]:

```
'demo entropy'
entropy(data)
```

Out[16]: 1.584962500721156

In [17]:

```
#info gain is basically the method in which we quantify
#by spliting upon this feature how much information have we gained
```

In [18]:

```
#weighted info gain
def  info_gain_entropy(current,left,right): p
    =float(len(left))/len(left)+len(right)
    return current-p*entropy(left)-(1-p)*entropy(right)
```

Best Split

In [19]:
```
#this is one of the most important function             as it lets
#us decide given the current data what is the best feature and featur e value to
split upon
#i.e it decides both wheter to split on petal length and what should be the petal
 length value (6.9cm) that we should split upon
```

In [20]:
```
def best_split(rows):
        #initialise best gain and best question
        best_gain=0
        best_question=None
        #calculate the current_gain
        current=entropy(rows) #total
        number of features
        features=len(rows[0])-1
        for col in range(features):
                #collects all unique classes for a feature
                values=set([row[col] for row in rows])
                for val in values:
                        #traverse each unique classs #ask
                        the corresponding question
                        question=Question(col,val)
                        #devide the data based on that ques
                        true_rows,false_rows=partition(rows,question) if
                        len(true_rows)==0 or len(false_rows) ==0:
                                #no use go to next iteration
                                continue
                        #calculate corresponding gain
                        gain=info_gain_entropy(current,true_rows,false_rows) #if
                        gain is > than the best replace
                        if gain>=best_gain:
                                best_gain,best_question=gain,question
                        #iterate through each unique class of each feature and re turn the
best gain and best question
        return best_gain,best_question
```

In [21]:
```
'demo best split'
a,b=best_split(data)
'best question initially and info gain by the respective ques' print(b)
print(a)
```

```
Is petal_length >= 6.9?
237.73467071046556
```

In [22]:
```
#we are done with our utility functions and classes now we will move on to
 major
#classes to actually build and print the tree
```

In [23]:
```
#this class represents all nodes in the tree
class DecisionNode:
    def___init___(self,question,true_branch,false_branch):
        #question object stores col and val variables regarding the q uestion of
that node
        self.question = question
        #this stores the branch that is true
        self.true_branch = true_branch #this
        stores the false branch
        self.false_branch = false_branch
```

In [24]:
```
#Leaf class is the one whichstores leaf of trees
#these are special Leaf Nodes -> on reaching them either
#100% purity is achieved or no features are left to split upon
class Leaf:
    def___init___(self,rows):
        #stores unique labels and their values in predictio
        self.predictions=count_values(rows)
```

In [25]:
```
#build tree function recurssively builds the tree
```

In [26]:
```
def build_tree(rows):
    #takes the whole dataset as argument
    #gets the best gain and best question
    gain,question=best_split(rows)

    #if gian=0 i.e. leaf conditions are satisfied
    if gain==0:
        #make a leaf object and return
        return Leaf(rows)
    # If we reach here, we have found a useful feature / value # to
    partition on.
    true_rows, false_rows = partition(rows, question)

    # Recursively build the true branch.
    true_branch = build_tree(true_rows)

    # Recursively build the false branch.
    false_branch = build_tree(false_rows)

    #returns the root question node storing branches as well as the q uesiton
    return DecisionNode(question, true_branch, false_branch)
```

In [27]:
```
#building the tree
tree=build_tree(data)
```

In [28]:

```
def print_tree(node,indentation=""):
    '''printing function'''
    #base case means we have reached the leaf #if
    the node object is of leaf type
    if isinstance(node,Leaf):
        print(indentation+"PREDICTION",node.predictions)
        return
    #print the question at node
    print(indentation + str(node.question))

    #call the function on true branch print(indentation+
    "True Branch")
    print_tree(node.true_branch,indentation + " ")

    #on flase branch
    print(indentation+ "False Branch")
    print_tree(node.false_branch,indentation + " ")
```

In [29]:

```
print_tree(tree)
```

```
Is petal_length >= 6.9?
True Branch
 PREDICTION {2.0: 1}
False Branch
 Is sepal_width >= 4.4?
 True Branch
  PREDICTION {0.0: 1}
 False Branch
  Is sepal_width >= 4.2? True
  Branch
   PREDICTION {0.0: 1}
  False Branch
   Is sepal_length >= 7.9? True
   Branch
    PREDICTION {2.0: 1}
   False Branch
    Is sepal_width >= 4.1?
    True Branch
     PREDICTION {0.0: 1}
    False Branch
     Is sepal_width >= 4.0?
     True Branch
      PREDICTION {0.0: 1}
     False Branch
      Is petal_length >= 6.7? True
      Branch
       PREDICTION {2.0: 2}
      False Branch
       Is petal_length >= 6.6? True
       Branch
        PREDICTION {2.0: 1}
       False Branch
        Is petal_length >= 6.3? True
        Branch
         PREDICTION {2.0: 1}
        False Branch
         Is sepal_length >= 7.7?
         True Branch
          PREDICTION {2.0: 1}
         False Branch
          Is sepal_length >= 7.4? True
          Branch
           PREDICTION {2.0: 1}
          False Branch
           Is petal_length >= 6.1? True
           Branch
            PREDICTION {2.0: 1}
           False Branch
            Is sepal_width >= 3.9? True
            Branch
             PREDICTION {0.0: 2}
            False Branch
             Is petal_width >= 2.5? True
             Branch
```

decision-tree-from-scratch-not-

PREDICTION {2.0: 2}

False Branch

Is petal_length >= 6.0?

```
True Branch
 PREDICTION {2.0: 1}
False Branch
 Is sepal_length >= 7.2? True
 Branch
  PREDICTION {2.0: 1}
 False Branch
  Is sepal_length >= 7.1? True
  Branch
   PREDICTION {2.0: 1}
  False Branch
   Is sepal_length >= 7.0? True
   Branch
    PREDICTION {1.0: 1}
   False Branch
    Is petal_length >= 5.9?
    True Branch
     PREDICTION {2.0: 1}
    False Branch
     Is petal_length >= 5.8? True
     Branch
      PREDICTION {2.0: 2}
     False Branch
      Is petal_length >= 5.7? True
      Branch
       PREDICTION {2.0: 2}
      False Branch
       Is sepal_width >= 3.8? True
       Branch
        PREDICTION {0.0: 4}
       False Branch
        Is sepal_width >= 3.7?
        True Branch
         PREDICTION {0.0: 3}
        False Branch
         Is sepal_width >= 3.6? True
         Branch
          PREDICTION {0.0: 3}
         False Branch
          Is petal_width >= 2.4? True
          Branch
           PREDICTION {2.0: 3}
          False Branch
           Is petal_width >= 2.3? True
           Branch
            PREDICTION {2.0: 4}
           False Branch
            Is petal_width >= 2.2?
            True Branch
             PREDICTION {2.0: 1}
            False Branch
             Is petal_width >= 2.1? True
             Branch
              PREDICTION {2.0: 3}
             False Branch
```

decision-tree-from-scratch-not-

Is sepal_length >= 6.9? True Branch

```
    PREDICTION {1.0: 1}
   False Branch
    Is sepal_length >= 6.8?
    True Branch
     PREDICTION {1.0: 1}
    False Branch
     Is sepal_length >= 6.7? True
     Branch
      PREDICTION {1.0: 3}
     False Branch
      Is sepal_length >= 6.6? True
      Branch
       PREDICTION {1.0: 2}
      False Branch
       Is petal_length >= 5.6? True
       Branch
        PREDICTION {2.0: 2}
       False Branch
        Is petal_length >= 5.5?
        True Branch
         PREDICTION {2.0: 2}
        False Branch
         Is petal_length >= 5.3? True
         Branch
          PREDICTION {2.0: 1}
         False Branch
          Is petal_length >= 5.2? True
          Branch
           PREDICTION {2.0: 1}
          False Branch
           Is petal_width >= 2.0? True
           Branch
            PREDICTION {2.0: 3}
           False Branch
            Is sepal_length >= 6.5?
            True Branch
             PREDICTION {1.0: 1}
            False Branch
             Is sepal_length >= 6.4? True
             Branch
              PREDICTION {1.0: 2}
             False Branch
              Is sepal_width >= 3.5? True
              Branch
               PREDICTION {0.0: 6}
              False Branch
               Is petal_width >= 1.9? True
               Branch
                PREDICTION {2.0: 3}
               False Branch
                Is sepal_width >= 3.4? True
                Branch
                 Is petal_width >= 1.6?
                 True Branch
                  PREDICTION {1.0: 1}
```

decision-tree-from-scratch-not-

False Branch

PREDICTION {0.0: 9}

```
False Branch
  Is petal_width >= 1.8? True
  Branch
    Is petal_length >= 5.1?
    True Branch
      PREDICTION {2.0: 1}
    False Branch
      Is sepal_length >= 6.3?
      True Branch
        PREDICTION {2.0: 1}
      False Branch
        Is petal_length >= 4.9?
        True Branch
          PREDICTION {2.0: 1}
        False Branch
          Is sepal_length >= 6.2? True
          Branch
            PREDICTION {2.0: 1}
          False Branch
            Is sepal_width >= 3.2? True
            Branch
              PREDICTION {1.0: 1}
            False Branch
              PREDICTION {2.0: 1}
  False Branch
    Is petal_width >= 1.7?
    True Branch
      PREDICTION {2.0: 1}
    False Branch
      Is petal_width >= 1.6?
      True Branch
        PREDICTION {1.0: 2}
      False Branch
        Is sepal_width >= 3.3?
        True Branch
          PREDICTION {0.0: 2}
        False Branch
          Is petal_length >= 5.1? True
          Branch
            PREDICTION {2.0: 1}
          False Branch
            Is sepal_length >= 6.
3?

            True Branch
              PREDICTION {1.0: 2}
            False Branch
              Is sepal_length >= 6.
2?

              True Branch
                PREDICTION {1.0: 2}
              False Branch
                Is sepal_length >=
6.1?

                True Branch
                  PREDICTION {1.0: 4}
```

False Branch Is petal_length >=

ength >= 5.8?

5.0?

1}

6.0?

2}

= 5.9?

1}

= 1.5?

0: 2}

>= 1.4?

0: 1}

h >= 4.5?

{1.0: 1}

th >= 4.4?

{1.0: 1}

gth >= 4.2?

{1.0: 3}

ngth >= 4.1?

{1.0: 3}

h

True
Branch
PREDICT
ION
{2.0:

False Branch
Is
sepal_length
>=

True
Branch
PREDICT
ION
{1.0:

False Branch
Is
sepal_length
>

True Branch
PREDICTIO
N {1.0:

False Branch
Is
petal_width
>

True
Branc
h
PREDI
CTION
{1.

False
Branch
Is
petal_widt
h

True
Branc
h
PREDI
CTION
{1.

False
Branch
Is

petal_lengt

True Branch
PREDICTIO
N

False Branch Is
petal_leng

True Branch
PREDICTIO
N

False Branch Is
petal_len

True Branch
PREDICTIO
N

False Branch
Is petal_le

True Branch
PREDICTIO
N

False Branc

Is sepal_l

True Branc

h
PREDICTIO
N {1.0: 2}
False Bran
ch
Is sepal_
length >= 5.7? ch
True Bran
ON {1.0: 1}
PREDICTI
nch
False Bra
_length >= 4.0?
Is petal
nch
True Bra
ION {1.0: 2}
PREDICT
anch
False Br
l_width >= 1.3?
Is peta
anch
True Br
TION {1.0: 1}
PREDIC
ranch
False B
al_length >= 3.9?
Is pet
ranch
True B
CTION {1.0: 1}
PREDI
Branch
False
tal_length >= 3.8?
Is pe
Branch
True
ICTION {1.0: 1}
PRED
Branch
False
epal_width >= 3.2?
Is s
Branch
True
DICTION {0.0: 5}
PRE
e Branch petal_width
Fals
>= 1.1? e Branch
Is

decision-tree-from-scratch-not-

Tru

PR

EDICTION {1.0: 1}

Fal

se Branch sepal_width

Is

>= 3.1? ue Branch

Tr

REDICTION {0.0: 4}

P

lse Branch

Fa

s petal_length >= 3.7?

I

rue Branch

T

PREDICTION {1.0: 1}

alse Branch

F

Is petal_length >= 3.5?

True Branch

PREDICTION {1.0: 1}

False Branch

Is sepal_width >= 3.0?

True Branch

PREDICTION {0.0: 6}

False Branch

Is sepal_width >= 2.9?

True Branch

PREDICTION {0.0: 1}

False Branch

Is sepal_width >= 2.4?

True Branch

PREDICTION {1.0: 1}

False Branch

Is petal_width >= 1.0?

True Branch

PREDICTION {1.0: 1}

False Branch

PREDICTION {0.0: 1}

Output Corresponding To Project Requirements

In [30]:

```
def   split_info(left,right):
      num=float(len(left))
      den=len(left)+len(right)
      p=num/den
      from math import log
      log10=lambda x:log(x)/log(10)
      return -(log10(p)*p+(1-p)*log10(1-p))
```

In [31]:

```
def print_tree_output(data,level):
      gain,question=best_split(data)
      #base case means we have reached the leaf #if
      the node object is of leaf type
      if gain==0:
            print("Level ",level)
            count=count_values(data)
            for value in count:
                  print("Count of ",value," = ",count[value])
            print("Current entropy is = ",entropy(data))
            print("Reached Leaf Node")
            return
      print('Level ',level)
      count=count_values(data)
      for value in count:
            print("Count of ",value," = ",count[value])
      feature=question.column
      true_rows, false_rows = partition(data, question)
      split=split_info(true_rows,false_rows)
      gain1=info_gain_entropy(entropy(data),true_rows,false_rows)
      print("Current entropy is = ",entropy(data))
      print("Spliting on feature ",header[feature]," with gain ratio ", gain1/split)

      print_tree_output(true_rows,level+1)
      print_tree_output(false_rows,level+1)
```

In [32]:
```
print_tree_output(data,0)
```

Level 0
Count of 0.0 = 50
Count of 1.0 = 50
Count of 2.0 = 50
Current entropy is = 1.58496250072115
6
Spliting on feature petal_length with gain ratio 13668.4901896228
1
8
Level 1
Count of 2.0 = 1
Current entropy is = 0.0
Reached Leaf Node
Level 1
Count of 0.0 = 50
Count of 1.0 = 50
Count of 2.0 = 49
Current entropy is = 1.5848973705351974
Spliting on feature sepal_width with gain ratio
13501.92708309278 Level 2
Count of 0.0 = 1
Current entropy is = 0.0
Reached Leaf Node
Level 2
Count of 0.0 = 49
Count of 1.0 = 50
Count of 2.0 = 49
Current entropy is = 1.584896783115256
3
Spliting on feature sepal_width with gain ratio 13335.19347939470
Level 3
4
Count of 0.0 = 1
Current entropy is = 0.0
Reached Leaf Node
Level 3
Count of 0.0 = 48
Count of 1.0 = 50
Count of 2.0 = 49
Current entropy is = 1.584762195958834
5
Spliting on feature sepal_length with gain ratio 13169.9662250811
5
4
Level 4
Count of 2.0 = 1
Current entropy is = 0.0
Reached Leaf Node
Level 4
Count of 0.0 = 48
Count of 1.0 = 50
Count of 2.0 = 48
Current entropy is = 1.58469298656517
3

Spliting on feature sepal_width with gain ratio 13004.56208724807
Level 5 2
Count of 0.0 = 1
Current entropy is = 0.0
Reached Leaf Node
Level 5
Count of 0.0 = 47
Count of 1.0 = 50
Count of 2.0 = 48
Current entropy is = 1.5844836724135505

Spliting on feature sepal_width with gain ratio 12838.95779740254
Level 6 1
Count of 0.0 = 1
Current entropy is = 0.0
Reached Leaf Node
Level 6
Count of 0.0 = 46
Count of 1.0 = 50
Count of 2.0 = 48
Current entropy is =
1.58412736601193
5
Spliting on feature petal_length with gain ratio 7125.37644420821
Level 7 8
Count of 2.0 = 2
Current entropy is = 0.0
Reached Leaf Node
Level 7
Count of 0.0 = 46
Count of 1.0 = 50
Count of 2.0 = 46
Current entropy is =
1.583828038899179
2
Spliting on feature petal_length with gain ratio 12351.3176175303
0

2
Level 8
Count of 2.0 = 1
Current entropy is = 0.0
Reached Leaf Node
Level 8
Count of 0.0 = 46
Count of 1.0 = 50
Count of 2.0 = 45
Current entropy is =
1.583451715503799
5
Spliting on feature petal_length with gain ratio 12188.3647164049
5

4
Level 9
Count of 2.0 = 1
Current entropy is = 0.0
Reached Leaf Node
Level 9
Count of 0.0 = 46
Count of 1.0 = 50
Count of 2.0 = 44
Current entropy is =
1.582914083463538
5
Spliting on feature sepal_length with gain ratio 12025.1823624129
6

3
Level 10
Count of 2.0 = 1
Current entropy is = 0.0
Reached Leaf Node

Level 10
Count of 0.0 = 46
Count of 1.0 = 50
Count of 2.0 = 43
Current entropy is = 1.5822069438058886
Spliting on feature sepal_length with gain ratio 11861.75577870337 7
Level 11
Count of 2.0 = 1

Current entropy is = 0.0
Reached Leaf Node
Level 11
Count of 0.0 = 46
Count of 1.0 = 50
Count of 2.0 = 42
Current entropy is = 1.5813216218211636
Spliting on feature petal_length with gain ratio
11698.06922176539 8
Level 12
Count of 2.0 = 1
Current entropy is = 0.0
Reached Leaf Node
Level 12
Count of 0.0 = 46
Count of 1.0 = 50
Count of 2.0 = 41
Current entropy is = 1.5802489321816928
Spliting on feature sepal_width with gain ratio
6493.507808938599 Level 13
Count of 0.0 = 2
Current entropy is = 0.0
Reached Leaf Node
Level 13
Count of 0.0 = 44
Count of 1.0 = 50
Count of 2.0 = 41
Current entropy is = 1.5800197978055068
Spliting on feature petal_width with gain ratio
6312.741487405622 Level 14
Count of 2.0 = 2
Current entropy is = 0.0
Reached Leaf Node
Level 14
Count of 0.0 = 44
Count of 1.0 = 50
Count of 2.0 = 39
Current entropy is = 1.577549448701152
Spliting on feature petal_length with gain ratio 10904.6870151332
7
1
Level 15
Count of 2.0 = 1
Current entropy is = 0.0
Reached Leaf Node
Level 15
Count of 0.0 = 44
Count of 1.0 = 50
Count of 2.0 = 38
Current entropy is = 1.5759922540581992
Spliting on feature sepal_length with gain ratio 10742.9795745220
6
7
Level 16
Count of 2.0 = 1
Current entropy is = 0.0

Reached Leaf Node
Level 16
Count of 0.0 = 44

Count of 1.0 = 50
Count of 2.0 = 37
Current entropy is = 1.5742048699569278
Spliting on feature sepal_length with gain ratio
10580.93058566067 5
Level 17
Count of 2.0 = 1
Current entropy is = 0.0
Reached Leaf Node
Level 17
Count of 0.0 = 44
Count of 1.0 = 50
Count of 2.0 = 36
Current entropy is = 1.5721747302538347
Spliting on feature sepal_length with gain ratio
10443.00197060713 6
Level 18
Count of 1.0 = 1
Current entropy is = 0.0
Reached Leaf Node
Level 18
Count of 0.0 = 44
Count of 1.0 = 49
Count of 2.0 = 36
Current entropy is = 1.5736065295813195
Spliting on feature petal_length with gain ratio
10282.32285163621 6
Level 19
Count of 2.0 = 1
Current entropy is = 0.0
Reached Leaf Node
Level 19
Count of 0.0 = 44
Count of 1.0 = 49
Count of 2.0 = 35
Current entropy is = 1.5714009439471868
Spliting on feature petal_length with gain ratio
5690.542009977429 Level 20
Count of 2.0 = 2
Current entropy is = 0.0
Reached Leaf Node
Level 20
Count of 0.0 = 44
Count of 1.0 = 49
Count of 2.0 = 33
Current entropy is = 1.5661626257180497
Spliting on feature petal_length with gain ratio
5507.719782168353 Level 21
Count of 2.0 = 2
Current entropy is = 0.0
Reached Leaf Node
Level 21
Count of 0.0 = 44
Count of 1.0 = 49
Count of 2.0 = 31

decision-tree-from-scratch-not-

Current entropy is = 1.5597135748733315

Spliting on feature sepal_width with gain ratio 3050.7365082147076

Level 22
Count of 0.0 = 4
Current entropy is = 0.0
Reached Leaf Node
Level 22
Count of 0.0 = 40
Count of 1.0 = 49
Count of 2.0 = 31
Current entropy is = 1.5604073307824116
Spliting on feature sepal_width with gain ratio 3622.911031758813 Level 23
Count of 0.0 = 3
Current entropy is = 0.0
Reached Leaf Node
Level 23
Count of 0.0 = 37
Count of 1.0 = 49
Count of 2.0 = 31
Current entropy is = 1.558820766490161
Spliting on feature sepal_width with gain ratio 3453.168757138821 Level 24
Count of 0.0 = 3
Current entropy is = 0.0
Reached Leaf Node
Level 24
Count of 0.0 = 34
Count of 1.0 = 49
Count of 2.0 = 31
Current entropy is = 1.5550426055143043
Spliting on feature petal_width with gain ratio 3274.0517913842446 Level 25
Count of 2.0 = 3
Current entropy is = 0.0
Reached Leaf Node
Level 25
Count of 0.0 = 34
Count of 1.0 = 49
Count of 2.0 = 28
Current entropy is = 1.5448622016494786
Spliting on feature petal_width with gain ratio 2445.2681378573125 Level 26
Count of 2.0 = 4
Current entropy is = 0.0
Reached Leaf Node
Level 26
Count of 0.0 = 34
Count of 1.0 = 49
Count of 2.0 = 24
Current entropy is = 1.5252649398454041
Spliting on feature petal_width with gain ratio 7065.4766814482855 Level 27
Count of 2.0 = 1
Current entropy is = 0.0
Reached Leaf Node
Level 27

decision-tree-from-scratch-not-

```
Count of  0.0  =  34
Count of  1.0  =  49
```

Count of 2.0 = 23
Current entropy is = 1.5190860776436896
Spliting on feature petal_width with gain ratio 2783.643069783842 7
Level 28
Count of 2.0 = 3
Current entropy is = 0.0
Reached Leaf Node
Level 28
Count of 0.0 = 34
Count of 1.0 = 49
Count of 2.0 = 20
Current entropy is = 1.4968589654171605
Spliting on feature sepal_length with gain ratio 6512.493863978548 Level 29
Count of 1.0 = 1
Current entropy is = 0.0
Reached Leaf Node
Level 29
Count of 0.0 = 34
Count of 1.0 = 48
Count of 2.0 = 20
Current entropy is = 1.5009498661393947
Spliting on feature sepal_length with gain ratio 6414.906047696535 Level 30
Count of 1.0 = 1
Current entropy is = 0.0
Reached Leaf Node
Level 30
Count of 0.0 = 34
Count of 1.0 = 47
Count of 2.0 = 20
Current entropy is = 1.5049642101863716
Spliting on feature sepal_length with gain ratio 2585.200235996691 Level 31
Count of 1.0 = 3
Current entropy is = 0.0
Reached Leaf Node
Level 31
Count of 0.0 = 34
Count of 1.0 = 44
Count of 2.0 = 20
Current entropy is = 1.516472193908067
Spliting on feature sepal_length with gain ratio 3415.69878179385 Level 32
Count of 1.0 = 2
Current entropy is = 0.0
Reached Leaf Node
Level 32
Count of 0.0 = 34
Count of 1.0 = 42
Count of 2.0 = 20
Current entropy is = 1.5236121855444196
Spliting on feature petal_length with gain ratio 3254.872888388312 3

Level 33
Count of 2.0 = 2
Current entropy is = 0.0

Reached Leaf Node
Level 33
Count of 0.0 = 34
Count of 1.0 = 42
Count of 2.0 = 18
Current entropy is = 1.506613124175711
Spliting on feature petal_length with gain ratio
3091.00811237361 Level 34
Count of 2.0 = 2
Current entropy is = 0.0
Reached Leaf Node
Level 34
Count of 0.0 = 34
Count of 1.0 = 42
Count of 2.0 = 16
Current entropy is = 1.4860503434568013
Spliting on feature petal_length with gain ratio
5208.872635788621 Level 35
Count of 2.0 = 1
Current entropy is = 0.0
Reached Leaf Node
Level 35
Count of 0.0 = 34
Count of 1.0 = 42
Count of 2.0 = 15
Current entropy is = 1.4742295051069683
Spliting on feature petal_length with gain ratio
5061.528119363404 Level 36
Count of 2.0 = 1
Current entropy is = 0.0
Reached Leaf Node
Level 36
Count of 0.0 = 34
Count of 1.0 = 42
Count of 2.0 = 14
Current entropy is = 1.4612526822404976
Spliting on feature petal_width with gain ratio 1961.4432272297972
Level 37
Count of 2.0 = 3
Current entropy is = 0.0
Reached Leaf Node
Level 37
Count of 0.0 = 34
Count of 1.0 = 42
Count of 2.0 = 11
Current entropy is = 1.414152505455283
Spliting on feature sepal_length with gain ratio 4526.8536374067
Level 38
Count of 1.0 = 1
Current entropy is = 0.0
Reached Leaf Node
Level 38
Count of 0.0 = 34
Count of 1.0 = 41
Count of 2.0 = 11

decision-tree-from-scratch-not-

Current entropy is = 1.4182750268315956

Spliting on feature sepal_length with gain ratio 2526.931807930457

6
Level 39
Count of 1.0 = 2
Current entropy is = 0.0
Reached Leaf Node
Level 39
Count of 0.0 = 34
Count of 1.0 = 39
Count of 2.0 = 11
Current entropy is = 1.4261487210745998
Spliting on feature sepal_width with gain ratio
1010.2466468793115 Level 40
Count of 0.0 = 6
Current entropy is = 0.0
Reached Leaf Node
Level 40
Count of 0.0 = 28
Count of 1.0 = 39
Count of 2.0 = 11
Current entropy is = 1.4291153963205705
Spliting on feature petal_width with gain ratio
1466.8574501872913 Level 41
Count of 2.0 = 3
Current entropy is = 0.0
Reached Leaf Node
Level 41
Count of 0.0 = 28
Count of 1.0 = 39
Count of 2.0 = 8
Current entropy is = 1.3656636991193396
Spliting on feature sepal_width with gain ratio
338.54361634126457 Level 42
Count of 0.0 = 9
Count of 1.0 = 1
Current entropy is = 0.4689955935892812
Spliting on feature petal_width with gain ratio
3.321928094887363 Level 43
Count of 1.0 = 1
Current entropy is = 0.0
Reached Leaf Node
Level 43
Count of 0.0 = 9
Current entropy is = 0.0
Reached Leaf Node
Level 42
Count of 0.0 = 19
Count of 1.0 = 38
Count of 2.0 = 8
Current entropy is = 1.3434159935471564
Spliting on feature petal_width with gain ratio 233.38692402898286
Level 43
Count of 1.0 = 1
Count of 2.0 = 5
Current entropy is =
Spliting on feature

decision-tree-from-scratch-not-
0.6500224216483541
petal_length with gain ratio 21.76893192910433
7

Count of 2.0 = 1
Current entropy is = 0.0
Reached Leaf Node
Level 44
Count of 1.0 = 1
Count of 2.0 = 4
Current entropy is = 0.7219280948873623
Spliting on feature sepal_length with gain ratio 18.25420522051331 3
Level 45
Count of 2.0 = 1
Current entropy is = 0.0
Reached Leaf Node
Level 45
Count of 1.0 = 1
Count of 2.0 = 3
Current entropy is = 0.8112781244591328
Spliting on feature petal_length with gain ratio 14.60232370248752 3
Level 46
Count of 2.0 = 1
Current entropy is = 0.0
Reached Leaf Node
Level 46
Count of 1.0 = 1
Count of 2.0 = 2
Current entropy is = 0.9182958340544896
Spliting on feature sepal_length with gain ratio 10.55691266455545 8
Level 47
Count of 2.0 = 1
Current entropy is = 0.0
Reached Leaf Node
Level 47
Count of 1.0 = 1
Count of 2.0 = 1
Current entropy is = 1.0
Spliting on feature sepal_width with gain ratio 3.321928094887363 Level 48
Count of 1.0 = 1
Current entropy is = 0.0
Reached Leaf Node
Level 48
Count of 2.0 = 1
Current entropy is = 0.0
Reached Leaf Node
Level 43
Count of 0.0 = 19
Count of 1.0 = 37
Count of 2.0 = 3
Current entropy is = 1.1671297479065408
Spliting on feature petal_width with gain ratio 1754.623928839013 6
Level 44

Count of 2.0 = 1
Count of 2.0 = 1
Current entropy is = 0.0
Reached Leaf Node

Count of 0.0 = 19
Count of 1.0 = 37
Count of 2.0 = 2
Current entropy is = 1.108663800581033
Spliting on feature petal_width with gain ratio
983.774568246391 Level 45
Count of 1.0 = 2
Current entropy is = 0.0
Reached Leaf Node
Level 45
Count of 0.0 = 19
Count of 1.0 = 35
Count of 2.0 = 2
Current entropy is = 1.1245776301200694
Spliting on feature sepal_width with gain ratio
909.7702046357939 Level 46
Count of 0.0 = 2
Current entropy is = 0.0
Reached Leaf Node
Level 46
Count of 0.0 = 17
Count of 1.0 = 35
Count of 2.0 = 2
Current entropy is = 1.1065213189351428
Spliting on feature petal_length with gain ratio
1390.162950260009 2
Level 47
Count of 2.0 = 1
Current entropy is = 0.0
Reached Leaf Node
Level 47
Count of 0.0 = 17
Count of 1.0 = 35
Count of 2.0 = 1
Current entropy is = 1.0295850980664145
Spliting on feature sepal_length with gain ratio
779.1532536959066 Level 48
Count of 1.0 = 2
Current entropy is = 0.0
Reached Leaf Node
Level 48
Count of 0.0 = 17
Count of 1.0 = 33
Count of 2.0 = 1
Current entropy is = 1.0459180039306184
Spliting on feature sepal_length with gain ratio
739.0238207738696 Level 49
Count of 1.0 = 2
Current entropy is = 0.0
Reached Leaf Node
Level 49
Count of 0.0 = 17
Count of 1.0 = 31
Count of 2.0 = 1
Current entropy is = 1.0623230293550265

decision-tree-from-scratch-not-

Spliting on feature sepal_length with gain ratio 409.8549414278636 Level 50

Count of 1.0 = 4
Current entropy is = 0.0
Reached Leaf Node
Level 50
Count of 0.0 = 17
Count of 1.0 = 27
Count of 2.0 = 1
Current entropy is = 1.0947679661147989
Spliting on feature petal_length with gain ratio 938.6340168141011 Level 51
Count of 2.0 = 1
Current entropy is = 0.0
Reached Leaf Node
Level 51
Count of 0.0 = 17
Count of 1.0 = 27
Current entropy is = 0.9624127354629923
Spliting on feature sepal_length with gain ratio 521.2235293553695 Level 52
Count of 1.0 = 2
Current entropy is = 0.0
Reached Leaf Node
Level 52
Count of 0.0 = 17
Count of 1.0 = 25
Current entropy is = 0.9736680645496201
Spliting on feature sepal_length with gain ratio 841.2420024562715 Level 53
Count of 1.0 = 1
Current entropy is = 0.0
Reached Leaf Node
Level 53
Count of 0.0 = 17
Count of 1.0 = 24
Current entropy is = 0.9788698505067785
Spliting on feature petal_width with gain ratio 466.81907539463356 Level 54
Count of 1.0 = 2
Current entropy is = 0.0
Reached Leaf Node
Level 54
Count of 0.0 = 17
Count of 1.0 = 22
Current entropy is = 0.9881108365218301
Spliting on feature petal_width with gain ratio 746.9609067118135 Level 55
Count of 1.0 = 1
Current entropy is = 0.0
Reached Leaf Node
Level 55
Count of 0.0 = 17
Count of 1.0 = 21
Current entropy is = 0.9919924034538556
Spliting on feature petal_length with gain ratio 715.536313822048

Level 56
Count of 1.0 = 1

Reached Leaf Node
Level 56
Count of 0.0 = 17
Count of 1.0 = 20
Current entropy is = 0.9952525494396791
Spliting on feature petal_length with gain ratio
684.1005839919533 Level 57
Count of 1.0 = 1
Current entropy is = 0.0
Reached Leaf Node
Level 57
Count of 0.0 = 17
Count of 1.0 = 19
Current entropy is = 0.9977724720899821
Spliting on feature petal_length with gain ratio
272.7427127721933 Level 58
Count of 1.0 = 3
Current entropy is = 0.0
Reached Leaf Node
Level 58
Count of 0.0 = 17
Count of 1.0 = 16
Current entropy is = 0.9993375041688847
Spliting on feature petal_length with gain ratio
231.3912303018377 4
Level 59
Count of 1.0 = 3
Current entropy is = 0.0
Reached Leaf Node
Level 59
Count of 0.0 = 17
Count of 1.0 = 13
Current entropy is = 0.9871377743721863
Spliting on feature sepal_length with gain ratio
263.7209323779444 Level 60
Count of 1.0 = 2
Current entropy is = 0.0
Reached Leaf Node
Level 60
Count of 0.0 = 17
Count of 1.0 = 11
Current entropy is = 0.9666186325481028
Spliting on feature sepal_length with gain ratio
398.1569183437486 6
Level 61
Count of 1.0 = 1
Current entropy is = 0.0
Reached Leaf Node
Level 61
Count of 0.0 = 17
Count of 1.0 = 10
Current entropy is = 0.9509560484549725
Spliting on feature petal_length with gain ratio

decision-tree-from-scratch-not-
205.4520630292470 3

Level 62
Count of 1.0 = 2

Reached Leaf Node
Level 62
Count of 0.0 = 17
Count of 1.0 = 8
Current entropy is = 0.904381457724494
Spliting on feature petal_width with gain ratio 298.957510680968
6
Level 63
Count of 1.0 = 1
Current entropy is = 0.0
Reached Leaf Node
Level 63
Count of 0.0 = 17
Count of 1.0 = 7
Current entropy is = 0.8708644692353646
Spliting on feature petal_length with gain ratio
264.7646805913552 Level 64
Count of 1.0 = 1
Current entropy is = 0.0
Reached Leaf Node
Level 64
Count of 0.0 = 17
Count of 1.0 = 6
Current entropy is = 0.828055725379504
Spliting on feature petal_length with gain ratio 229.673368100222
8
6
Level 65
Count of 1.0 = 1
Current entropy is = 0.0
Reached Leaf Node
Level 65
Count of 0.0 = 17
Count of 1.0 = 5
Current entropy is = 0.7732266742876346
Spliting on feature sepal_width with gain ratio
67.1533232215734 Level 66
Count of 0.0 = 5
Current entropy is = 0.0
Reached Leaf Node
Level 66
Count of 0.0 = 12
Count of 1.0 = 5
Current entropy is = 0.8739810481273578
Spliting on feature petal_width with gain ratio
142.5946751572193 Level 67
Count of 1.0 = 1
Current entropy is = 0.0
Reached Leaf Node
Level 67
Count of 0.0 = 12
Count of 1.0 = 4
Current entropy is = 0.8112781244591328
Spliting on feature sepal_width with gain ratio
48.443510525288 Level 68
Count of 0.0 = 4

Current entropy is = 0.0
Reached Leaf Node
Level 68

Count of 0.0 = 8
Count of 1.0 = 4
Current entropy is = 0.9182958340544896
Spliting on feature petal_length with gain ratio 82.01854998441854 Level 69
Count of 1.0 = 1
Current entropy is = 0.0
Reached Leaf Node
Level 69
Count of 0.0 = 8
Count of 1.0 = 3
Current entropy is = 0.8453509366224365
Spliting on feature petal_length with gain ratio 60.95633879769478 Level 70
Count of 1.0 = 1
Current entropy is = 0.0
Reached Leaf Node
Level 70
Count of 0.0 = 8
Count of 1.0 = 2
Current entropy is = 0.7219280948873623
Spliting on feature sepal_width with gain ratio 16.155204693246237 Level 71
Count of 0.0 = 6
Current entropy is = 0.0
Reached Leaf Node
Level 71
Count of 0.0 = 2
Count of 1.0 = 2
Current entropy is = 1.0
Spliting on feature sepal_width with gain ratio 15.375080272125093 Level 72
Count of 0.0 = 1
Current entropy is = 0.0
Reached Leaf Node
Level 72
Count of 0.0 = 1
Count of 1.0 = 2
Current entropy is = 0.9182958340544896
Spliting on feature sepal_width with gain ratio 10.556912664555458 Level 73
Count of 1.0 = 1
Current entropy is = 0.0
Reached Leaf Node
Level 73
Count of 0.0 = 1
Count of 1.0 = 1
Current entropy is = 1.0
Spliting on feature petal_width with gain ratio 3.321928094887363 Level 74
Count of 1.0 = 1
Current entropy is = 0.0
Reached Leaf Node
Level 74
Count of 0.0 = 1

decision-tree-from-scratch-not-

Current entropy is = 0.0
Reached Leaf Node

In [33]:

```
#as we havent any max_depth this tree is overfitting to a very large extent
#and will probably perform very poorly on any new data its performs o n
```

Linear Neural Network using Gradient Descent

In [2]:
```
# Import necessary libraries
import sys
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import load_iris
```

In [3]:
```
iris= load_iris()
```

In [45]:

```
class NeuralNetwork(object):
    def__init__(self, input_nodes, hidden_nodes, output_nodes, learn
ing_rate):
        # Set number of nodes in input, hidden and output layers.
        self.input_nodes = input_nodes
        self.hidden_nodes = hidden_nodes
        self.output_nodes = output_nodes

        # Initialize weights
        self.weights_input_to_hidden = np.random.normal(0.0, self.inp
ut_nodes**-0.5,
                                        (self.input_nodes, self.hidden
_nodes))

        self.weights_hidden_to_output = np.random.normal(0.0, self.hi
dden_nodes**-0.5,
                                        (self.hidden_nodes, self.outpu
t_nodes))
        self.lr = learning_rate

        self.activation_function = lambda x : 1./(1.+np.exp(-x))


    def train(self, features, targets):
        ''' Train the network on batch of features and targets.

            Arguments


            features: 2D array, each row is one data record, each col umn is a
feature
            targets: 1D array of target values

        '''
        n_records = features.shape[0]
        delta_weights_i_h      =      np.zeros(self.weights_input_to_hidden.sha
pe)
        delta_weights_h_o = np.zeros(self.weights_hidden_to_output.sh for X,
ape)
        y in zip(features, targets):

            final_outputs, hidden_outputs = self.forward_pass_train(X
)
            delta_weights_i_h, delta_weights_h_o = self.backpropagati
on(final_outputs, hidden_outputs, X, y,

delta_weights_i_h, delta_weights_h_o) self.update_weights(delta_weights_i_h,
        delta_weights_h_o, n_r
ecords)


    def forward_pass_train(self, X):
        ''' Implement forward pass here
            ----------
```

Arguments

```
            X: features batch
        '''
        ### Forward pass ###
        hidden_inputs = np.dot(X, self.weights_input_to_hidden) # sig nals into
hidden layer
        hidden_outputs = self.activation_function(hidden_inputs) # si gnals
from hidden layer

        final_inputs = np.dot(hidden_outputs, self.weights_hidden_to_ output)
# signals into final output layer
        final_outputs = final_inputs # signals from final output laye
r
        return final_outputs, hidden_outputs

    def backpropagation(self, final_outputs, hidden_outputs, X, y, de
lta_weights_i_h, delta_weights_h_o):
        ''' Implement backpropagation

            Arguments
            ----------
            final_outputs: output from forward pass y:
            target (i.e. label) batch
            delta_weights_i_h: change in weights from input to hidden
layers
            delta_weights_h_o: change in weights from hidden to outpu
t  layers
        '''
        ### Backward pass ###
        error = y - final_outputs # Output layer error is the differe nce between
desired target and actual output.


        output_error_term = error
        hidden_error = np.dot(self.weights_hidden_to_output, output_e
rror_term)

        hidden_error_term = hidden_error * hidden_outputs * (1 - hidd
en_outputs)

        # Weight step (input to hidden)
        delta_weights_i_h += hidden_error_term * X[:, np.newaxis]
        # Weight step (hidden to output)
        delta_weights_h_o += output_error_term * hidden_outputs[:, np
.newaxis]
        return delta_weights_i_h, delta_weights_h_o

    def update_weights(self, delta_weights_i_h, delta_weights_h_o, n_
records):
        ''' Update weights on gradient descent step

            Arguments
            ----------
                        t layers
layers
```

delta_weights_i_h:

change in weights

from input to

hidden

delta_weights_h_o

: change in

weights from

hidden to outpu

n_records: number of records

```
            '''
            self.weights_hidden_to_output += self.lr * delta_weights_h_o
/ n_records # update hidden-to-output weights with gradient descent s tep
            self.weights_input_to_hidden += self.lr * delta_weights_i_h / n_records
# update input-to-hidden weights with gradient descent step

      def run(self, features):
            ''' Run a forward pass through the network with input feature
s

                  Arguments

                  ----------
                  features: 1D array of feature values
            '''
            hidden_inputs = np.dot(features, self.weights_input_to_hidden
) # signals into hidden layer
            hidden_outputs = self.activation_function(hidden_inputs) # si gnals
from hidden layer

            final_inputs = np.dot(hidden_outputs, self.weights_hidden_to_ output)
# signals into final output layer
            final_outputs = final_inputs # signals from final output laye
r
            return final_outputs
```

In [50]:

```
def MSE(y, Y):
      return np.mean((y-Y)**2)
```

In [49]:

```
X = iris.data
y = iris.target
```

In [46]:

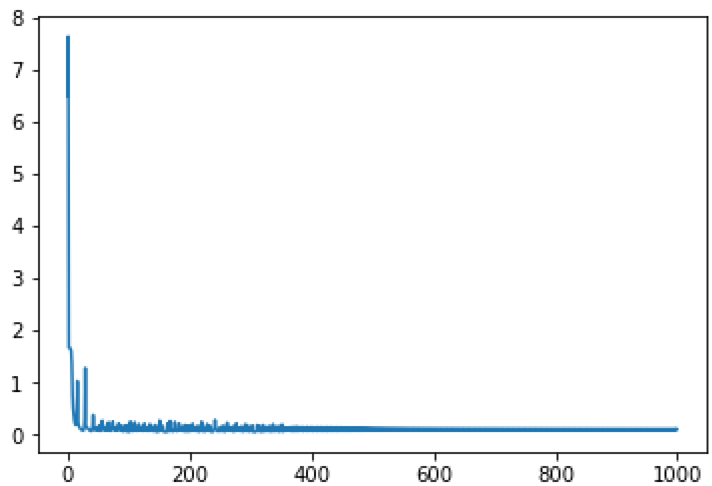
```
######## Set up hyperparameters ########

# Input nodes
N_i = iris.data.shape[1] #
Hidden Nodes hidden_nodes
= 10
# Output Node
output_nodes = 1

# Number of iterations
epochs = 1000
# Learning Rate
learning_rate = 0.7
```

In [53]:

```
network = NeuralNetwork(N_i, hidden_nodes, output_nodes, learning_rat e)
losses = {'train':[], 'validation':[]}

for ii in range(epochs):
      network.train(X, y)

      train_loss = MSE(network.run(X).T, y)

      losses['train'].append(train_loss)
```

In [56]:

```
# Plot the train loss
plt.plot(losses['train'], label='Training Loss') plt.show()
```



In [59]:

```
# To get the predictions we can easily call the 'run' method. #
network.run(X_test)
```

In [ ]:

SVD

In [1]:
```
# Import necessary libraries
import pandas as pd
import numpy as np
from sklearn.datasets import load_digits
from matplotlib import pyplot as plt
from sklearn.decomposition import TruncatedSVD
```

In [2]:
```
# Load the dataset
X, y = load_digits(return_X_y=True)
```

In [3]:
```
X.shape
```

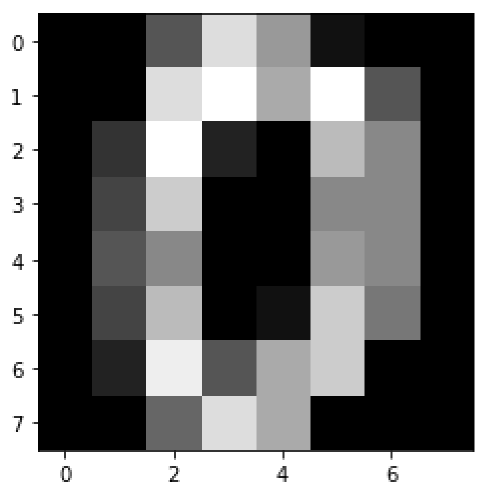
Out[3]: (1797, 64)

In [4]:
```
y.shape
```

Out[4]: (1797,)

In [9]:
```
image = X[0].reshape((8, 8))
plt.imshow(image, cmap='gray')
```

Out[9]: <matplotlib.image.AxesImage at 0x7f6af33c6bb0>



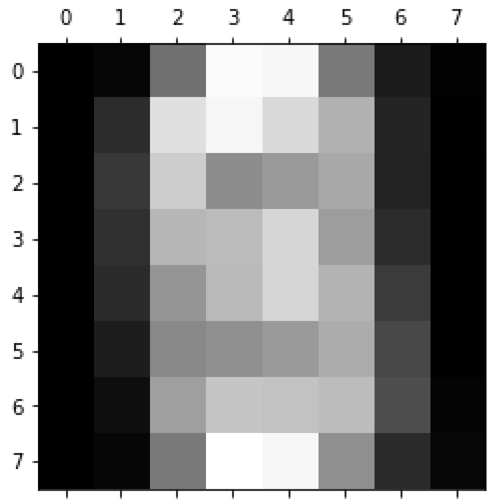
In [18]:
```
# Apply SVD on the dataset
svd = TruncatedSVD(n_components=2)
X_reduced = svd.fit_transform(X)
svd.explained_variance_ratio_.sum()
```

Out[18]: 0.17760900859732742

Now all the digits contains only 2 components.

In [15]:
```
image_reduced = svd.inverse_transform(X_reduced[0].reshape(1,-1))
image_reduced = image_reduced.reshape((8,8))
plt.matshow(image_reduced, cmap = 'gray')
```

Out[15]: <matplotlib.image.AxesImage at 0x7f6af337f970>



We can see that SVD pixalated the image a lot and it is no longer recognizable. It only contains 17% features of the original data.

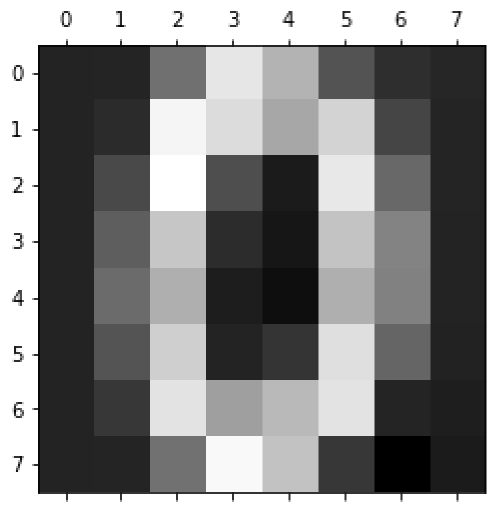
In [16]:
```
svd = TruncatedSVD(n_components=16)
X_reduced = svd.fit_transform(X)
svd.explained_variance_ratio_.sum()
```

Out[16]: 0.8479575375450391

Here we can see that with 16 components we have retained almost 85% of the image.

In [17]:
```
image_reduced = svd.inverse_transform(X_reduced[0].reshape(1,-1))
image_reduced = image_reduced.reshape((8,8))
plt.matshow(image_reduced, cmap = 'gray')
```

Out[17]: <matplotlib.image.AxesImage at 0x7f6af329e400>



In [ ]:

Principal Component Analysis

In [1]:
```
import numpy as np
from matplotlib.image import imread
import matplotlib.pyplot as plt

image_raw = imread("/input/commonwanderer/TheCommonWanderer_-2.jpg")
print(image_raw.shape)

# Displaying the image
plt.figure(figsize=[12,8])
plt.imshow(image_raw);
```

(681, 1000, 3)

Out[1]: <matplotlib.image.AxesImage at 0x7f98eefcddd8>

In [3]:

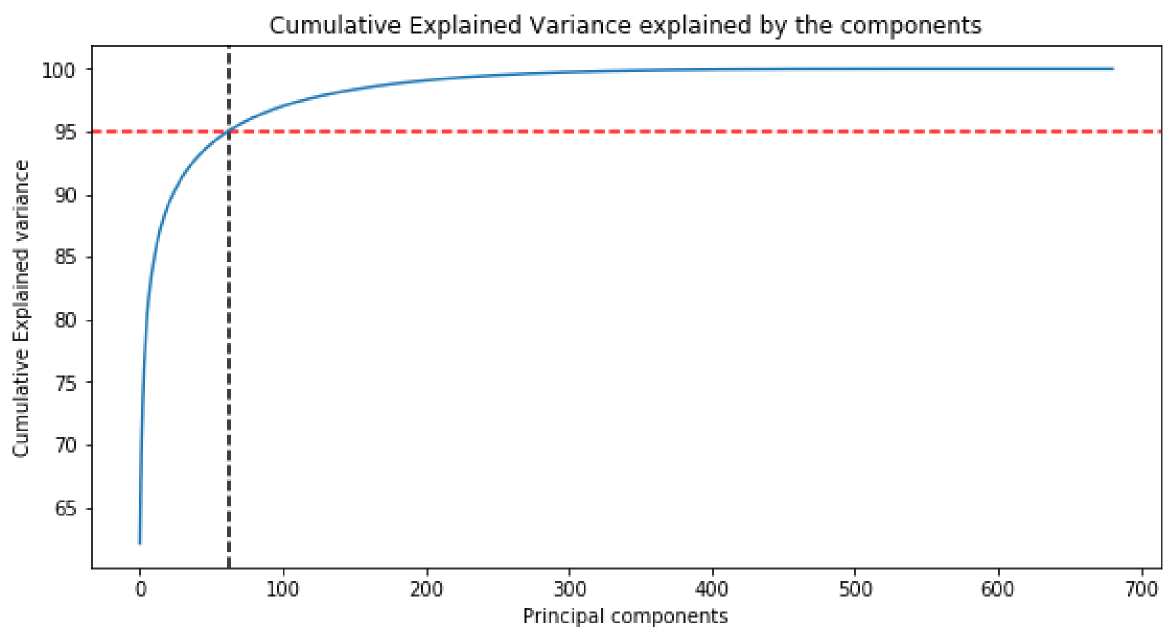
```
# Converting the image to grayscale to apply PCA
image_sum = image_raw.sum(axis=2)
print(image_sum.shape)

image_bw =
image_sum/image_sum.max()
print(image_bw.max())

plt.figure(figsize=[12,8])
```

```
(681, 1000)
1.0
```

In [4]:

```
from sklearn.decomposition import PCA, IncrementalPCA
pca = PCA()
pca.fit(image_bw)

# Getting the cumulative variance

var_cumu = np.cumsum(pca.explained_variance_ratio_)*100

# How many PCs explain 95% of the variance?
k = np.argmax(var_cumu>95)
print("Number of components explaining 95% variance: "+ str(k))

plt.figure(figsize=[10,5])
plt.title('Cumulative Explained Variance explained by the components'
)
plt.ylabel('Cumulative Explained variance')
plt.xlabel('Principal components') plt.axvline(x=k,
color="k", linestyle="--") plt.axhline(y=95,
color="r", linestyle="--") ax = plt.plot(var_cumu)
```

Number of components explaining 95% variance: 62



62 of the components explain about 95% of the image. We will only those components and remake the image.

In [ ]:

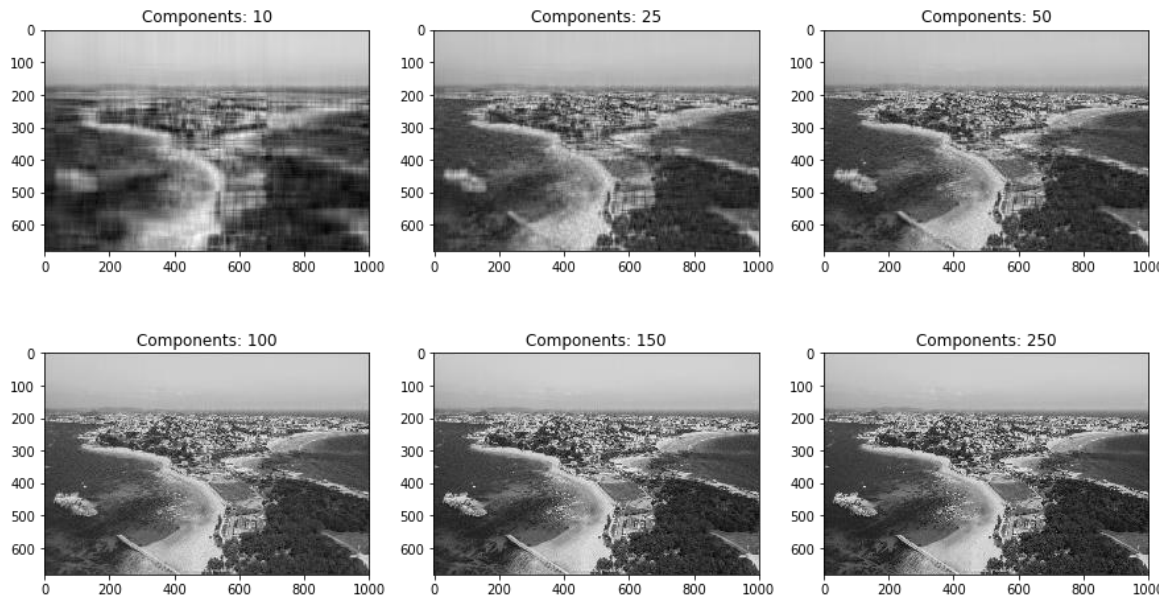
```
# Function to reconstruct and plot image for a given number of compon ents

def plot_at_k(k):
    ipca = IncrementalPCA(n_components=k)
    image_recon = ipca.inverse_transform(ipca.fit_transform(image_bw
))
    plt.imshow(image_recon,cmap = plt.cm.gray)


k = 150
plt.figure(figsize=[12,8])
plot_at_k(100)
```

In [6]:

```
ipca = IncrementalPCA(n_components=k)
image_recon = ipca.inverse_transform(ipca.fit_transform(image_bw))

# Plotting the reconstructed image
plt.figure(figsize=[12,8])
plt.imshow(image_recon,cmap = plt.cm.gray);
```

In [9]:

```
ks = [10, 25, 50, 100, 150, 250]

plt.figure(figsize=[15,9])

for i in range(6):
    plt.subplot(2,3,i+1)
    plot_at_k(ks[i])
    plt.title("Components: "+str(ks[i]))

plt.subplots_adjust(wspace=0.2, hspace=0.0)
plt.show()
```



In [ ]: