*#importing libraries*

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

*#reading dataset*

ds = pd.read\_csv('../input/customer-segmentation-tutorial-in-python/Mall\_Customers.csv')

ds.head()

*#getting dimensions*

ds.shape

ds.isnull().any()

plt.figure(figsize=(14,6))

plt.subplot(1, 2, 1)

sns.set(style = 'whitegrid')

sns.distplot(ds['Annual Income (k$)'])

plt.title('Distribution of Annual Income', fontsize = 20)

plt.xlabel('Range of Annual Income')

plt.ylabel('Count')

plt.subplot(1, 2, 2)

sns.set(style = 'whitegrid')

sns.distplot(ds['Age'], color = 'red')

plt.title('Distribution of Age', fontsize = 20)

plt.xlabel('Range of Age')

plt.ylabel('Count')

plt.show()

plt.figure(figsize=(8,5))

sns.countplot(x = 'Gender',data = ds)

plt.figure(figsize=(20,8))

sns.countplot(ds['Spending Score (1-100)'])

*#Pairplot for the Data*

sns.pairplot(ds,hue = 'Gender')

plt.subplots\_adjust(hspace = 0.8)

plt.figure(figsize=(10,8))

sns.heatmap(ds.corr(), cmap = 'Wistia', annot = True)

plt.title('Heatmap for the Data', fontsize = 20)

plt.show()

sns.jointplot(x = 'Age',y = 'Spending Score (1-100)',data = ds, hue = 'Gender')

plt.figure(figsize=(10,6))

sns.stripplot(x = 'Gender',y = 'Spending Score (1-100)',data = ds,)

*#defining dependent variable*

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

In [13]:

linkcode

*#ELBOW METHOD TO FIND OPTIMAL NUMBER OF CLUSTERS*

from sklearn.cluster import KMeans

wcss = []

for i **in** range(1,11):

kmeans = KMeans(n\_clusters = i, init = 'k-means++')

kmeans.fit(x)

wcss.append(kmeans.inertia\_)

plt.plot(range(1,11),wcss)

*#training model*

kmeans = KMeans(n\_clusters = 5, init = 'k-means++')

y\_pred = kmeans.fit\_predict(x)

In [15]:

linkcode

*#visualizing clusters*

plt.figure(figsize=(12,8))

plt.scatter(x[y\_pred == 0,0],x[y\_pred == 0,1],label = 'Cluster-1', s = 100)

plt.scatter(x[y\_pred == 1,0],x[y\_pred == 1,1],label = 'Cluster-2', s = 100)

plt.scatter(x[y\_pred == 2,0],x[y\_pred == 2,1],label = 'Cluster-3', s = 100)

plt.scatter(x[y\_pred == 3,0],x[y\_pred == 3,1],label = 'Cluster-4', s = 100)

plt.scatter(x[y\_pred == 4,0],x[y\_pred == 4,1],label = 'Cluster-5', s = 100)

plt.scatter(kmeans.cluster\_centers\_[:,0], kmeans.cluster\_centers\_[:, 1], s = 50, c = 'black' , label = 'centeroid')

plt.legend()

plt.xlabel('Annual Income')

plt.ylabel('Spending Score')

*#defining dependent variable*

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

In [17]:

linkcode

*#getting optimal number of clusters using dendogram*

from scipy.cluster.hierarchy import dendrogram, linkage

plt.figure(figsize = (12,6))

dendo = dendrogram(linkage(x,method = 'ward'))

plt.title('Dendrogam', fontsize = 20)

plt.xlabel('Customers')

plt.ylabel('Ecuclidean Distance')

plt.show()

from sklearn.cluster import AgglomerativeClustering

ac = AgglomerativeClustering(n\_clusters = 5, affinity = 'euclidean', linkage = 'ward')

y\_pred = ac.fit\_predict(x)

In [19]:

linkcode

*#visualizing clusters*

plt.figure(figsize=(12,8))

plt.scatter(x[y\_pred == 0,0],x[y\_pred == 0,1],label = 'Cluster-1', s = 100)

plt.scatter(x[y\_pred == 1,0],x[y\_pred == 1,1],label = 'Cluster-2', s = 100)

plt.scatter(x[y\_pred == 2,0],x[y\_pred == 2,1],label = 'Cluster-3', s = 100)

plt.scatter(x[y\_pred == 3,0],x[y\_pred == 3,1],label = 'Cluster-4', s = 100)

plt.scatter(x[y\_pred == 4,0],x[y\_pred == 4,1],label = 'Cluster-5', s = 100)

plt.legend()

plt.xlabel('Annual Income')

plt.ylabel('Spending Score')

import numpy as np

import matplotlib.pyplot as plt

%matplotlib inline

x = np.arange(-5, 5, 0.01)

def plot(func, yaxis=(-1.4, 1.4)):

    plt.ylim(yaxis)

    plt.locator\_params(nbins=5)

    plt.xticks(fontsize = 14)

    plt.yticks(fontsize = 14)

    plt.axhline(lw=1, c='black')

    plt.axvline(lw=1, c='black')

    plt.grid(alpha=0.4, ls='-.')

    plt.box(on=None)

    plt.plot(x, func(x), c='r', lw=3)

binary\_step = np.vectorize(lambda x: 1 if x > 0 else 0, otypes=[np.float])

plot(binary\_step, yaxis=(-0.4, 1.4))

piecewise\_linear = np.vectorize(lambda x: 1 if x > 3 else 0 if x < -3 else 1/6\*x+1/2, otypes=[np.float])

plot(piecewise\_linear, yaxis=(-0.4, 1.4))

def sigmoid(x):

    return 1 / (1 + np.exp(-x))

plot(sigmoid, yaxis=(-0.4, 1.4))

def tanh(x):

    return 2 / (1 + np.exp(-2 \* x)) -1

plot(tanh)