Linear Regression

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as pl
from sklearn.linear_model import LinearRegression
import csv

# Function to get data from the input file
def get_data(file_name):
#Reading from CSV file using Pandas library
    data = pd.read_csv(file_name)
    x_parameter =[]
    y_parameter =[]
#Appending age and annual income data into x and y parameters
    for age, annualincome in zip(data['Age'],data['Annual Income']):
        x_parameter.append([float(age)])
        y_parameter.append([float(annualincome)])
    return age,annualincome


file_name ='Customers.csv'
data = pd.read_csv(file_name)
x_parameter = []
y_parameter = []
#Iterating through the files to get columns data
for age, annualincome in zip(data['Age'], data['Annual Income']):
    x_parameter.append([float(age)])
    y_parameter.append([float(annualincome)])

#Using numpy arrays to extract data for x and y
x = np.array(x_parameter).reshape(-1, 1)
y = np.array(y_parameter).reshape(-1, 1)

#Using sklearn Linear Regression model
linearModel = LinearRegression()

#Fiiting the data into the model
linearModel.fit(x, y)
pl.scatter(x, y, color='y')

#Predicting the Annual incomes for a given age values
pl.plot(x, linearModel.predict(x), color='r')
#Labelling x and y axis
pl.xlabel("Ages")
pl.ylabel("Annual Income")
#Displaying the graph
pl.show()
```

KMeans Clustering

```
import csv
import numpy as np
import matplotlib.pyplot as pl
from sklearn.cluster import KMeans

#Extracting the data from Customers.csv file

def getData(fileName):
    x=[]
    y=[]
    csvdata = open (fileName, 'r')
    fileData = csv.reader(csvdata)
    next(fileData)
```

```
    for columns in fileData:
        x.append(int(columns[2]))
        y.append(int(columns[3]))
    data = list(zip(x,y))
    return data

fileName = "Customers.csv"
data = getData(fileName)
#Appending the Kmeans algorithm for the data with 5 clusters
kmeans = KMeans(n_clusters = 5)
#Fitting the data into the model
kmeans.fit(data)
#Determining the centroids and respective labels
centroids = kmeans.cluster_centers_
labels = kmeans.labels_
#Appending colors and labels
colors = ["r","g","b","y","c"]
clusterLabel = ["Cluster-1","Cluster-2","Cluster-3","Cluster-4","Cluster-5"]

#Performing Clustering

for i in range(len(data)):
    pl.scatter(data[i][0],data[i][1],c=colors[labels[i]],
label=clusterLabel[labels[i]])

#Plotting the graph with description
pl.scatter(centroids[:, 0], centroids[:, 1], label="Centroids", marker=".", s=100,
linewidths=20, zorder=25)
pl.title('Customer Clusters')
pl.xlabel('Customer Age')
pl.ylabel('Annual Income(k$)')
pl.show()
```

NLP

```
# -*- coding: utf-8 -*-
from nltk import word_tokenize
from nltk.corpus import stopwords
from nltk.tag import pos_tag
from nltk.stem.wordnet import WordNetLemmatizer
import nltk


# Function that tokenizes the input text
def tokenization(text):
    tokens = word_tokenize(text)
    print ('Tokenize')
    print (tokens, '\n')
    return tokens


# Function that removes stop words using English language
def stopWordsdeletion(tokens):
    stopWords = stopwords.words('english')
    filteredWords = [word for word in tokens if word not in stopWords]
    wordsWithoutStops = [word for word in filteredWords if len(word) > 2]
    print ('Filtered Words')
    print (wordsWithoutStops, '\n')
    return wordsWithoutStops


# Function that Lemmatizes the input paragraph
def lemmatization(nonstop):
    resultList = list()
    for j in nonstop:
```

```
        resultList.append(WordNetLemmatizer().lemmatize(j))
    print ('Lemmatized Words')
    print (resultList, '\n')
    return resultList


# Function to remove verbs from the paragraph
def verbsRemoval(lemwords):
    resultList = list()
    for j in pos_tag(lemwords):
        if j[1][:2] == 'VB':
            continue
        else:
            resultList.append(j[0])
    print ('Verb Removal')
    print (resultList, '\n')
    return resultList


# Function to calculate word frequency
def wordsFrequency(verbless):
    wordsFreq = nltk.FreqDist(verbless)
    topFive = dict()
    for w, f in wordsFreq.most_common(5):
        topFive[w] = f
    print ('Top Five Keys and Values')
    print (topFive, '\n')
    return topFive


# Function to get just top five words
def mostRepeatingwords(topfive):
    topFiveWords = topfive.keys()
    print ('Top Five Words')
    print (topFiveWords, '\n')
    return topFiveWords


# Function to find sentences with top five words
def sentenceFinder(text, topfive):
    result = list()
    for l in text.split('\n'):
        for w in topfive:
            if w in l.lower():
                result.append(l)
                break
    return result


# Function to summarize the sentence
def Summarizer(sentences):
    print ('Summarizing')
    print ('\n'.join(sentences))


def main():
    fileName = 'input.txt'
    text = open(fileName, "r").read()
    tokens = tokenization(text)
    nonstop = stopWordsdeletion(tokens)
    lemwords = lemmatization(nonstop)
    verbless = verbsRemoval(lemwords)
    topfive = wordsFrequency(verbless)
    topfivewords = mostRepeatingwords(topfive)
    sentences = sentenceFinder(text, topfivewords)
    Summarizer(sentences)

#Execution starts here
```

```
if __name__=="__main__":
    main()
```

SVM Classification

```
import numpy as np
from sklearn import datasets
from sklearn import svm
from sklearn.model_selection import train_test_split
from sklearn import metrics


# Splitting the test and training data
digits = datasets.load_digits()
x = digits.data[:, :2]
y = digits.target
xtrain, xtest, ytrain, ytest = train_test_split(x, y, test_size=0.2)



#  Code that finds accuracy of the data using linear kernel
linearKernel = svm.SVC()

linearPrediction = linearKernel.set_params(kernel='linear').fit(xtrain,
ytrain).predict(xtest)

linearAccuracy = metrics.accuracy_score(ytest, linearPrediction)

print ("Linear Kernel Accuracy:", linearAccuracy)


# Code that finds accuracy of data using rbf kernel

rbfKernel = svm.SVC()

rbfPrediction = rbfKernel.set_params(kernel='rbf').fit(xtrain,
ytrain).predict(xtest)

rbfAccuracy = metrics.accuracy_score(ytest, rbfPrediction)

print ("RBF Kernel Accuracy:", rbfAccuracy)
```