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


# Dataset Definition
# x-axis: Availabitliy of Doctors per 100,000 residents
# y-axis: Availability of Hospitals per 100,000 residents

# Function to get X,Y axis from an excel sheet for given columns
def getData(path, columns):
    x = []
    y = []
    # Using pandas library to extract columns from given excel file
    df = pd.read_excel(path, parse_cols=columns)
    # Appending the values to the array
    for i in df['X2']:
        x.append(i)

    for j in df['X3']:
        y.append(j)
    return x, y


# Function to get regression model
def LinearRegressionModel(a, b):

    # Converting to np array and reshaping
    x=np.array(a).reshape(-1,1)
    y=np.array(b).reshape(-1,1)

    # Using sklearn linear regression package
    lmodel = LinearRegression()

    # Fitting the data to the model
    lmodel.fit(x, y)

    # Adding points in the graph
    plt.scatter(x, y, color='g')

    # Adding prediction line int the graph
    plt.plot(x, lmodel.predict(x), color='b')
    plt.xlabel("Availabitliy of Doctors per 100,000 residents")
```

```
    plt.ylabel("Availability of Hospitals per 100,000 residents")

    # Show the graph
    plt.show()


def main():
    path = "/Volumes/Sibi-CLG/PythonFall17/Python-FL17/Lab-3/Source/data/data.xls"
    columns = "B,C"
    x, y = getData(path, columns)
    LinearRegressionModel(x, y)


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

```
import matplotlib.pyplot as plt
from collections import OrderedDict
import csv
from sklearn.cluster import KMeans


# Function to get data from filepath
def getData(file):
    x = []
    y = []
    csvd = open(file, 'r')
    data = csv.reader(csvd)
    next(data)
    for er in data:
        x.append(int(er[3]))
        y.append(int(er[4]))
    ndata = list(zip(x, y))
    return ndata


# Function to manipulate the data with kmeans clusters
def Kmeans(custdata):

    # Kmeans with 5 clusters
    kmeans = KMeans(n_clusters=5)

    # Fitting the data with the model
    kmeans.fit(custdata)

    # Finding the centroids and the labels
    kcentroids = kmeans.cluster_centers_
    klabels = kmeans.labels_

    # Colors and labels
    kcolors = ["g", "y", "r", "b", "c"]
    klabel = ["C-1", "C-2", "C-3", "C-4", "C-5"]

    # Potters for the graph
    for i in range(len(custdata)):
        plt.scatter(custdata[i][0], custdata[i][1], c=kcolors[klabels[i]], label=klabel[klabels[i]])

    plt.scatter(kcentroids[:, 0], kcentroids[:, 1], label="Centroids", marker=".", s=100,
linewidths=15, zorder=20)
    plt.title('Cluster of Customers')
```

```
    plt.xlabel('Annual Income(k$)')
    plt.ylabel('Spending Score(1-100)')

    # To display the graph with legends
    legendh, legendl = plt.gca().get_legend_handles_labels()
    dolabel = OrderedDict(zip(legendl, legendh))
    plt.legend(dolabel.values(), dolabel.keys())
    plt.show()


def main():
    path = "/Volumes/Sibi-CLG/PythonFall17/Python-FL17/Lab-3/Source/data/Customers.csv"
    data = getData(path)
    Kmeans(data)


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

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


# Function to split test and train data
def SplitData(bcancer):
   x = bcancer.data[:, :2]
   y = bcancer.target
   xtr, xte, ytr, yte = train_test_split(x, y, test_size=0.2)

   return xtr, xte, ytr, yte

# Function to find accuracy of data using linear kernel
def LinearKernel(xtr, xte, ytr, yte):
   svcmodel = svm.SVC()

   linearPrediction = svcmodel.set_params(kernel='linear').fit(xtr, ytr).predict(xte)

   linearAccuracy = metrics.accuracy_score(yte, linearPrediction)

   print "Linear Kernel Accuracy:", linearAccuracy

# Function to find accuracy of data using rbf kernel
def RBFKernel(xtr, xte, ytr, yte):
   svcmodel = svm.SVC()

   rbfPrediction = svcmodel.set_params(kernel='rbf').fit(xtr, ytr).predict(xte)

   rbfAccuracy = metrics.accuracy_score(yte, rbfPrediction)

   print "RBF Kernel Accuracy:", rbfAccuracy

def main():
   bcancer = datasets.load_breast_cancer()
   xtr, xte, ytr, yte = SplitData(bcancer)
   LinearKernel(xtr, xte, ytr, yte)
   RBFKernel(xtr, xte, ytr, yte)


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

```
from nltk import word_tokenize
from nltk.corpus import stopwords
from nltk.stem.wordnet import WordNetLemmatizer
from nltk.tag import pos_tag
import nltk


# Function to Tokenize the input text
def Tokenize(text):
    tokens = word_tokenize(text)
    print '******** Tokenize *********'
    print tokens, '\n'
    return tokens


# Function to remove stop words
def StopWordsRemoval(tokens):
    stopWords = stopwords.words('english')
    filteredWords = [wo for wo in tokens if wo not in stopWords]
    wordsWithoutStops = [wo for wo in filteredWords if len(wo) > 2]
    print '******** Filtered Words *********'
    print wordsWithoutStops, '\n'
    return wordsWithoutStops


# Function to Lemmatize
def Lemmatizer(nonstop):
    resultList = list()
    for j in nonstop:
        resultList.append(WordNetLemmatizer().lemmatize(j))
    print '******** Lemmatized Words *********'
    print resultList, '\n'
    return resultList


# Function to remove verbs
def RemoveVerbs(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 WordFrequenxy(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 GetWords(topfive):
    topFiveWords = topfive.keys()
    print '******** Top Five Words *********'
    print topFiveWords, '\n'
    return topFiveWords


# Function to find sentences with top five words
def FindSentences(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
def Summarizer(sentences):
    print '******** Summary *********'
    print '\n'.join(sentences)


def main():
    text = open('/Volumes/Sibi-CLG/PythonFall17/Python-FL17/Lab-3/Source/data/input.txt',
"r").read()
```

```
    tokens = Tokenize(text)
    nonstop = StopWordsRemoval(tokens)
    lemwords = Lemmatizer(nonstop)
    verbless = RemoveVerbs(lemwords)
    topfive = WordFrequenxy(verbless)
    topfivewords = GetWords(topfive)
    sentences = FindSentences(text, topfivewords)
    Summarizer(sentences)


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