1.

Code:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets, linear_model, metrics

# load the boston dataset
data_boston= datasets.load_boston(return_X_y=False)

# defining feature matrix(X) and response vector(y)
X = data_boston.data
y = data_boston.target

# split X and y into training and testing sets
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2,

random_state=1)
```

```
# creating a linear regression object
reg = linear_model.LinearRegression()

# training the model using the training sets
reg.fit(X_train, y_train)

# linear regression coefficients
print('Coefficients are: \n', reg.coef_)

# variance score: 1 means perfect prediction
print('Variance score is: {}'.format(reg.score(X_test,
y_test)))


#Scatter Plot

from sklearn.linear_model import LinearRegression


y_pred = lm.predict(X_test)

plt.scatter(y_test,y_pred )
```

```
plt.xlabel("Prices: ")
plt.ylabel("Predicted prices:")
plt.title("Prices vs Predicted prices")
plt.show()   #Ideally, the scatter plot should create a
linear line. Since the model does not fit 100%, the
scatter plot is not creating a linear line.


# plot for residual error

## setting plot style
plt.style.use('fivethirtyeight')

## plotting residual errors in training data
plt.scatter(reg.predict(X_train), reg.predict(X_train) -
y_train,
            color="red", s=10, label='Train data')

## plotting residual errors in test data
plt.scatter(reg.predict(X_test), reg.predict(X_test) -
y_test,
            color="blue", s=10, label='Test data')
```

```
## plotting line for zero residual error
plt.hlines(y=0, xmin=0, xmax=50, linewidth=2)

## plotting legend
plt.legend(loc='upper right')

## plot title
plt.title("Residual errors")

## function to show plot
plt.show()
```

2.

Code:

```
 import pandas
import matplotlib.pyplot as pl
```

```
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA

variables=pandas.read_csv('Customers.csv')
Y=variables[['Annual Income']]
X=variables[['Spending Score']]

Nc=range(1,20)
kmeans=[KMeans(n_clusters=i) for i in Nc]
score=[kmeans[i].fit(Y).score(Y)for i in
range(len(kmeans))]
pl.plot(Nc,score)
pl.xlabel('Number of Clusters')
pl.ylabel('Score')
pl.show()

pca=PCA(n_components=1).fit(Y)
pca_d=pca.transform(Y)
pca_c=pca.transform(X)


kmeans=KMeans(n_clusters=5)
```

```
kmeansoutput=kmeans.fit(Y)


pl.figure('5 Cluster K-Means')

pl.scatter(pca_c[:, 0], pca_d[:, 0],
c=kmeansoutput.labels_)

pl.xlabel('Annual Income')

pl.ylabel('Spending Score')

pl.title('5 Cluster K-Means')

pl.show()
```

3.

Code: Using Linear Kernel

```
import sklearn
from sklearn import svm
from sklearn import datasets, metrics
from sklearn.cross_validation import train_test_split
import matplotlib.pyplot as plt
from matplotlib import style
style.use("ggplot")

#Loading the dataset
breastcancer_data=datasets.load_breast_cancer()

#getting the data and response of the dataset
x=breastcancer_data.data
y=breastcancer_data.target


clf=svm.SVC(kernel='linear',C=1.0)
```

```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_si
ze=0.2)
model= clf.fit(x_train,y_train)

print(model.score(x_test,y_test))
```

Code: Using rbf Kernel

```
import sklearn
from sklearn import svm
from sklearn import datasets, metrics
from sklearn.cross_validation import train_test_split
import matplotlib.pyplot as plt
from matplotlib import style
style.use("ggplot")

#Loading the dataset
breastcancer_data=datasets.load_breast_cancer()
```

```
#getting the data and response of the dataset
x=breastcancer_data.data
y=breastcancer_data.target


clf=svm.SVC(kernel='rbf',C=1.0)

x_train,x_test,y_train,y_test=train_test_split(x,y,test_si
ze=0.2)
model= clf.fit(x_train,y_train)

print(model.score(x_test,y_test))
```

4.

```
import nltk
from nltk.corpus import wordnet as wn
import re, collections
from nltk.book import FreqDist
```

```
file=open("got.txt")
t=file.read()

#lemmetization
from nltk.stem import WordNetLemmatizer
lemmetizer=WordNetLemmatizer()
print(lemmetizer.lemmatize(t))

#POS tagging
from nltk.tokenize import word_tokenize
from nltk.tag import pos_tag
s=nltk.pos_tag(nltk.word_tokenize(t))
print(s)

#removing verbs from input file
file_without_verbs = [word for word,tag in s if tag !=
'VBG' and tag != 'VBZ' and tag!='VBN']
z=' '.join(file_without_verbs)          # z is the file
without verbs
print(z)
s1=nltk.pos_tag(nltk.word_tokenize(z))
```

```
print(s1)                                   # you can see in
the output that all the verbs are removed




fdist=FreqDist(z)
print(fdist)
q=fdist.most_common(5)
print(q)

#word frequency of remaining words
def tokens(text):
    """
    Get all words from the corpus
    """
    return re.findall('[a-z]+', text.lower())
WORD_COUNTS = collections.Counter(tokens(z))
print (WORD_COUNTS)
print (WORD_COUNTS.most_common(5))
```

```
##go through the original file

file=open("got.txt")
t=file.read()
```