Python Lab Assignment-2

Authors

Lab ID-1

1.Nagababu Chilukuri, Class Id-26

2.Nikita Goyal, Class Id-7

3.Ronnie Dean Antwiler II, Class Id-1

Introduction

This lab Assignment includes the Data Analysis, Data Exploration ,Cleaning of the data , Handling the missing values and predicting some results using various Machine Learning Algorithms. The Natural language process has also been used to understand various text analysis concepts.

The Dataset we used in completing the Lab Assignment was:

1. Titantic
2. Wine Quality
3. Startups Sales and Profit

Objective

The main Objective is to understand and learn the various Machine learning Algorithm and NLP concepts using scikit Learn:

1. KNN, SVM , Naive bayes Classifier
2. K-mean Clustering
3. Tokenization, Lemmatization,Trigrams
4. Linear and Multiple Regression

Workflow

Question-1

---

1. Pick any dataset from the dataset sheet in the class sheet or online which includes both numeric and non-numeric features

- Perform exploratory data analysis on the data set (like Handling null values, removing the features not correlated to the target class, encoding the categorical features, …)
- Apply the three classification algorithms Naïve Baye’s, SVM and KNN on the chosen data set and report which classifier gives better result.

Solution:

Loading the Data

```
# Imports

# pandas
import pandas as pd
from pandas import Series,DataFrame

# numpy, matplotlib, seaborn
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_style('whitegrid')
%matplotlib inline

# machine learning
from sklearn.svm import SVC, LinearSVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import GridSearchCV, cross_val_score, StratifiedKFold, learning_curve
```

```
# Load data
# Load train and Test set

data_train = pd.read_csv("train.csv")
data_test = pd.read_csv("test.csv")
```

Analyzing the Data through Visualization

```
## Join train and test datasets in order to obtain the same number of features during categorical conversion
train_len = len(data_train)
train =  pd.concat(objs=[data_train, data_test], axis=0).reset_index(drop=True)
```

```
E:\anaconda\lib\site-packages\ipykernel_launcher.py:3: FutureWarning: Sorting because non-concatenation axis is not aligned.
A future version
of pandas will change to not sort by default.

To accept the future behavior, pass 'sort=True'.

To retain the current behavior and silence the warning, pass sort=False

  This is separate from the ipykernel package so we can avoid doing imports until
```

```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1309 entries, 0 to 1308
Data columns (total 12 columns):
Age            1046 non-null float64
Cabin          295 non-null object
Embarked       1307 non-null object
Fare           1308 non-null float64
Name           1309 non-null object
Parch          1309 non-null int64
PassengerId    1309 non-null int64
Pclass         1309 non-null int64
Sex            1309 non-null object
SibSp          1309 non-null int64
Survived       891 non-null float64
Ticket         1309 non-null object
dtypes: float64(3), int64(4), object(5)
memory usage: 122.8+ KB
```

Checking for Missing Values and

Replacing it

```
# Fill empty and NaNs values with NaN
train = train.fillna(np.nan)

# Check for Null values
train.isnull().sum()
```

```
Age            263
Cabin         1014
Embarked         2
Fare             1
Name             0
Parch            0
PassengerId      0
Pclass           0
Sex              0
SibSp            0
Survived       418
Ticket           0
dtype: int64
```

```
train.head(10)
```

	Age	Cabin	Embarked	Fare	Name	Parch	PassengerId	Pclass	Sex	SibSp	Survived	Ticket
0	22.0	NaN	S	7.2500	Braund, Mr. Owen Harris	0	1	3	male	1	0.0	A/5 21171
1	38.0	C85	C	71.2833	Cumings, Mrs. John Bradley (Florence Briggs Th...	0	2	1	female	1	1.0	PC 17599
2	26.0	NaN	S	7.9250	Heikkinen, Miss. Laina	0	3	3	female	0	1.0	STON/O2. 3101282
3	35.0	C123	S	53.1000	Futrelle, Mrs. Jacques Heath (Lily May Peel)	0	4	1	female	1	1.0	113803
4	35.0	NaN	S	8.0500	Allen, Mr. William Henry	0	5	3	male	0	0.0	373450
5	NaN	NaN	Q	8.4583	Moran, Mr. James	0	6	3	male	0	0.0	330877
6	54.0	E46	S	51.8625	McCarthy, Mr. Timothy J	0	7	1	male	0	0.0	17463
7	2.0	NaN	S	21.0750	Palsson, Master. Gosta Leonard	1	8	3	male	3	0.0	349909
8	27.0	NaN	S	11.1333	Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)	2	9	3	female	0	1.0	347742
9	14.0	NaN	C	30.0708	Nasser, Mrs. Nicholas (Adele Achem)	0	10	2	female	1	1.0	237736

```
train.info()
print("----------------------------")
train.dtypes
```

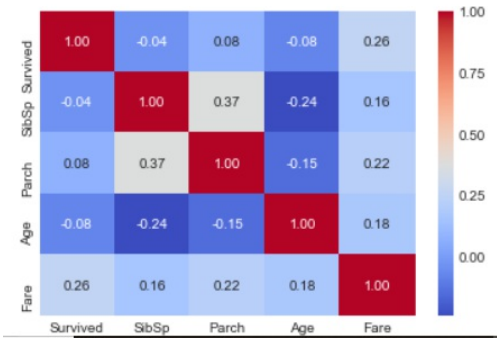
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1309 entries, 0 to 1308
Data columns (total 12 columns):
Age            1046 non-null float64
Cabin          295 non-null object
Embarked       1307 non-null object
Fare           1308 non-null float64
Name           1309 non-null object
Parch          1309 non-null int64
PassengerId    1309 non-null int64
Pclass         1309 non-null int64
Sex            1309 non-null object
SibSp          1309 non-null int64
Survived       891 non-null float64
Ticket         1309 non-null object
dtypes: float64(3), int64(4), object(5)
memory usage: 122.8+ KB
----------------------------

Age            float64
Cabin           object
Embarked        object
Fare           float64
Name            object
Parch            int64
PassengerId      int64
Pclass           int64
Sex             object
SibSp            int64
Survived       float64
Ticket          object
dtype: object
```

Exploring the Data for analyzing it

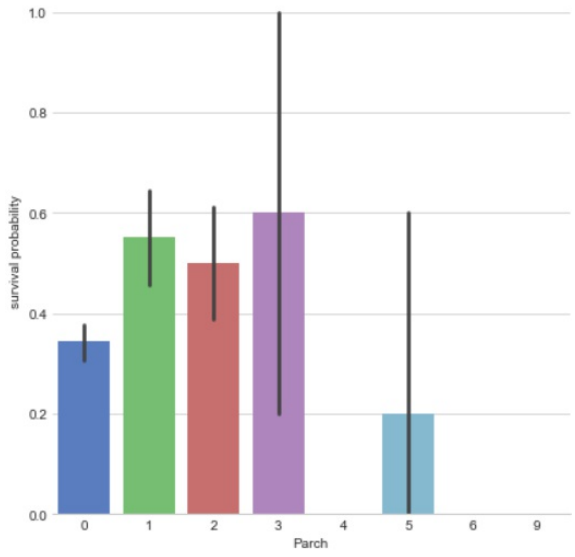
```
# Summarize data
train.describe()
```

	Age	Fare	Parch	PassengerId	Pclass	SibSp	Survived
count	1046.000000	1308.000000	1309.000000	1309.000000	1309.000000	1309.000000	891.000000
mean	29.881138	33.295479	0.385027	655.000000	2.294882	0.498854	0.383838
std	14.413493	51.758668	0.865560	378.020061	0.837836	1.041658	0.486592
min	0.170000	0.000000	0.000000	1.000000	1.000000	0.000000	0.000000
25%	21.000000	7.895800	0.000000	328.000000	2.000000	0.000000	0.000000
50%	28.000000	14.454200	0.000000	655.000000	3.000000	0.000000	0.000000
75%	39.000000	31.275000	0.000000	982.000000	3.000000	1.000000	1.000000
max	80.000000	512.329200	9.000000	1309.000000	3.000000	8.000000	1.000000

```
# Correlation matrix between numerical values (SibSp Parch Age and Fare values) and Survived

g = sns.heatmap(train[["Survived","SibSp","Parch","Age","Fare"]].corr(),annot=True, fmt = ".2f", cmap = "coolwarm")
```



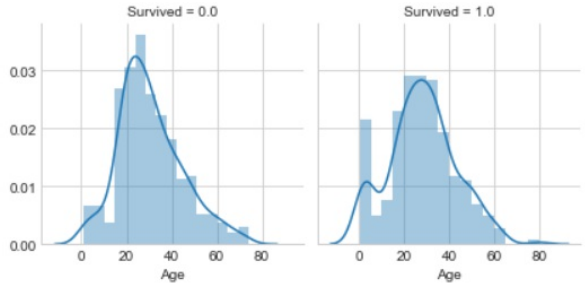

```
# Explore Parch feature vs Survived
g  = sns.factorplot(x="Parch",y="Survived",data=train,kind="bar", size = 6 ,
palette = "muted")
g.despine(left=True)
g = g.set_ylabels("survival probability")
```



Small families have more chance to survive, more than single (Parch 0), medium (Parch 3,4) and large families (Parch 5,6).

```
# Explore Age vs Survived
g = sns.FacetGrid(train, col='Survived')
g = g.map(sns.distplot, "Age")
```

```
E:\anaconda\lib\site-packages\matplotlib\axes\_axes.py:6462: UserWarning: The 'normed' kwarg is deprecated, and has been rep
laced by the 'density' kwarg.
  warnings.warn("The 'normed' kwarg is deprecated, and has been "
E:\anaconda\lib\site-packages\matplotlib\axes\_axes.py:6462: UserWarning: The 'normed' kwarg is deprecated, and has been rep
laced by the 'density' kwarg.
  warnings.warn("The 'normed' kwarg is deprecated, and has been "
```



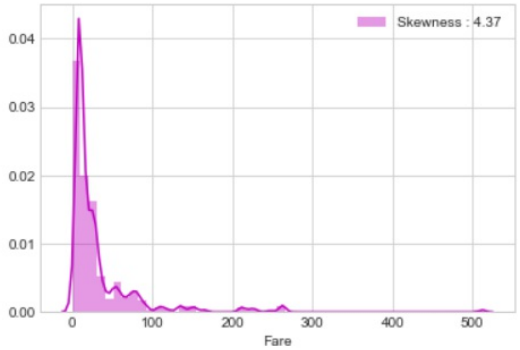

```
train.isnull().sum()
```

```
Age            263
Cabin         1014
Embarked         2
Fare             1
Name             0
Parch            0
PassengerId      0
Pclass           0
Sex              0
SibSp            0
Survived       418
Ticket           0
dtype: int64
```

```
#Fill Fare missing values with the median value
train["Fare"] = train["Fare"].fillna(train["Fare"].median())
```

```
# Explore Fare distribution
g = sns.distplot(train["Fare"], color="m", label="Skewness : %.2f"%(train["Fare"].skew()))
g = g.legend(loc="best")
```

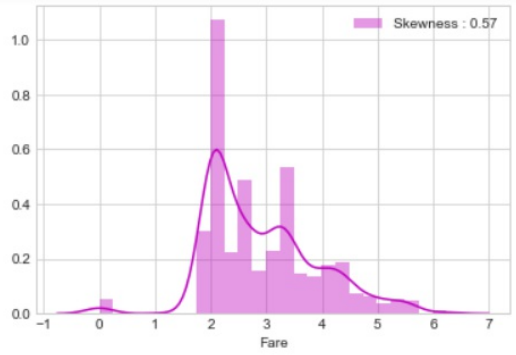
```
E:\anaconda\lib\site-packages\matplotlib\axes\_axes.py:6462: UserWarning: The 'normed' kwarg is deprecated, and has been rep
laced by the 'density' kwarg.
  warnings.warn("The 'normed' kwarg is deprecated, and has been "
```



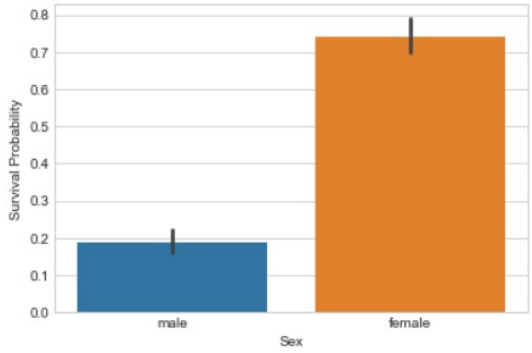

```
# Apply log to Fare to reduce skewness distribution
train["Fare"] = train["Fare"].map(lambda i: np.log(i) if i > 0 else 0)
```

```
# Explore Fare distribution
g = sns.distplot(train["Fare"], color="m", label="Skewness : %.2f"%(train["Fare"].skew()))
g = g.legend(loc="best")
```

```
E:\anaconda\lib\site-packages\matplotlib\axes\_axes.py:6462: UserWarning: The 'normed' kwarg is deprecated, and has been rep
laced by the 'density' kwarg.
  warnings.warn("The 'normed' kwarg is deprecated, and has been "
```



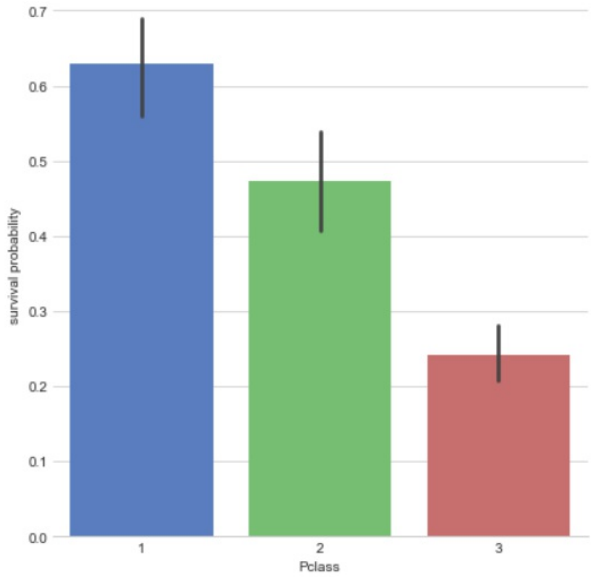

```
g = sns.barplot(x="Sex",y="Survived",data=train)
g = g.set_ylabel("Survival Probability")
```



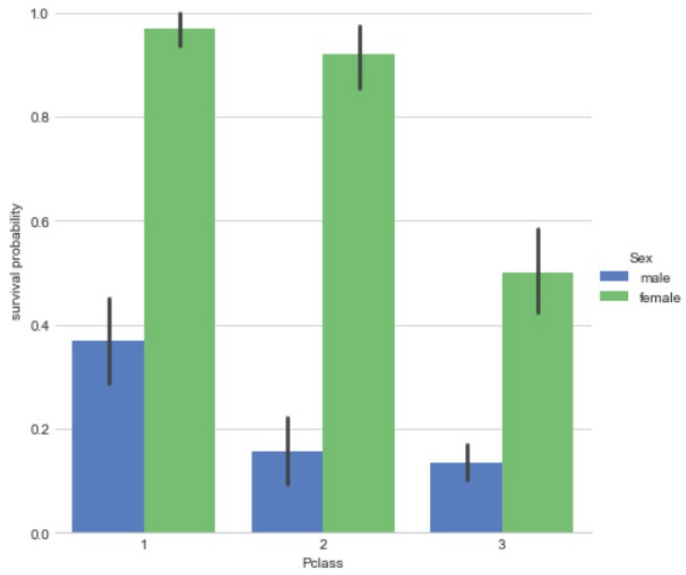

```
train[["Sex","Survived"]].groupby('Sex').mean()
```

Sex	Survived
female	0.742038
male	0.188908

```
# Explore Pclass vs Survived
g = sns.factorplot(x="Pclass",y="Survived",data=train,kind="bar", size = 6 ,
palette = "muted")
g.despine(left=True)
g = g.set_ylabels("survival probability")
```



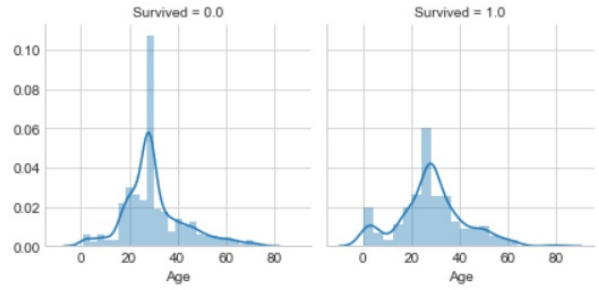

```
# Explore Pclass vs Survived by Sex
g = sns.factorplot(x="Pclass", y="Survived", hue="Sex", data=train,
                   size=6, kind="bar", palette="muted")
g.despine(left=True)
g = g.set_ylabels("survival probability")
```

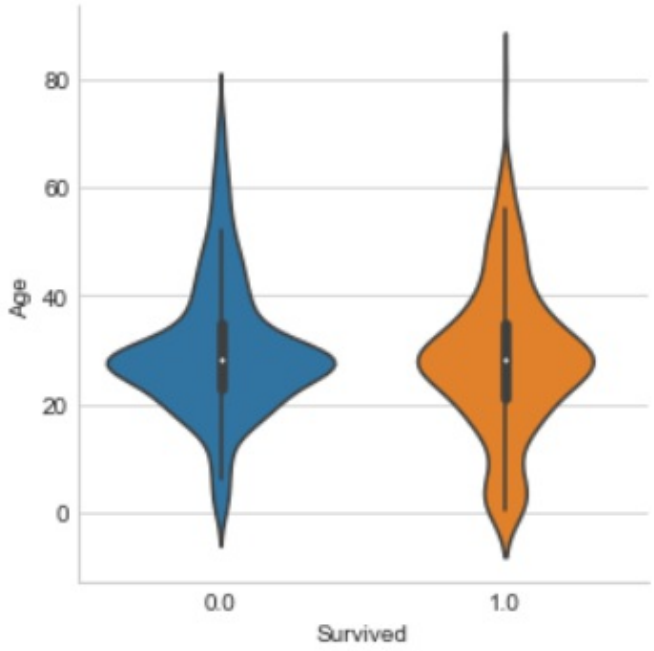



```
age_median=train['Age'].median()
count=train['PassengerId'].count()
for i in range(0,count):
    if np.isnan(train['Age'].iloc[i]):
        train["Age"].iloc[i]=age_median
```

```
# Explore Age vs Survived
g = sns.FacetGrid(train, col='Survived')
g = g.map(sns.distplot, "Age")
g = sns.factorplot(x="Survived", y = "Age",data = train, kind="violin")
```

```
E:\anaconda\lib\site-packages\matplotlib\axes\_axes.py:6462: UserWarning: The 'normed' kwarg is deprecated, and has been rep
laced by the 'density' kwarg.
  warnings.warn("The 'normed' kwarg is deprecated, and has been "
E:\anaconda\lib\site-packages\matplotlib\axes\_axes.py:6462: UserWarning: The 'normed' kwarg is deprecated, and has been rep
laced by the 'density' kwarg.
  warnings.warn("The 'normed' kwarg is deprecated, and has been "
```






```
# convert Sex into categorical value 0 for male and 1 for female
train["Sex"] = train["Sex"].map({"male": 0, "female":1})
```

```
# Create categorical values for Pclass
train["Pclass"] = train["Pclass"].astype("category")
train = pd.get_dummies(train, columns = ["Pclass"],prefix="Pc")
```

```
#get dummies for embarked
train = pd.get_dummies(train, columns = ["Embarked"], prefix="Em")
```

```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1309 entries, 0 to 1308
Data columns (total 16 columns):
Age            1309 non-null float64
Cabin          295 non-null object
Fare           1309 non-null float64
Name           1309 non-null object
Parch          1309 non-null int64
PassengerId    1309 non-null int64
Sex            1309 non-null int64
SibSp          1309 non-null int64
Survived       891 non-null float64
Ticket         1309 non-null object
Pc_1           1309 non-null uint8
Pc_2           1309 non-null uint8
Pc_3           1309 non-null uint8
Em_C           1309 non-null uint8
Em_Q           1309 non-null uint8
Em_S           1309 non-null uint8
dtypes: float64(3), int64(4), object(3), uint8(6)
memory usage: 110.0+ KB
```

Split The Data for Validation

```
# Drop useless variables
train.drop(labels = ["PassengerId","Name","Ticket","Cabin"], axis = 1, inplace = True)
```

```
## Separate train dataset and test dataset

train1 = train[:train_len]
test = train[train_len:]
test.drop(labels=["Survived"],axis = 1,inplace=True)
```

```
E:\anaconda\lib\site-packages\pandas\core\frame.py:3694: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy
  errors=errors)
```

```
## Separate train features and label

train1["Survived"] = train1["Survived"].astype(int)

Y_train = train1["Survived"]

X_train = train1.drop(labels = ["Survived"],axis = 1)
```

```
E:\anaconda\lib\site-packages\ipykernel_launcher.py:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy
  This is separate from the ipykernel package so we can avoid doing imports until
```

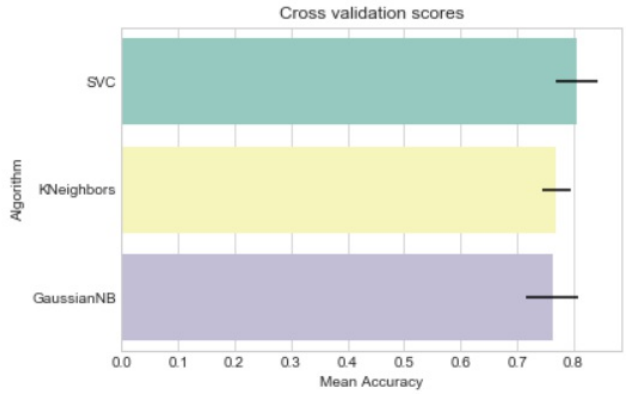
```
# Cross validate model with Kfold stratified cross val
kfold = StratifiedKFold(n_splits=10)
```

Evaluating the model using three Classifier

We use three Classifiers that is SVM, Naive Bayes and KNN Classifier to evaluate the model

```
# Modeling step Test differents algorithms
random_state = 2
classifiers = []
classifiers.append(SVC(random_state=random_state))
classifiers.append(KNeighborsClassifier())
classifiers.append(GaussianNB())

cv_results = []
for classifier in classifiers :
    cv_results.append(cross_val_score(classifier, X_train, y = Y_train, scoring = "accuracy", cv = kfold, n_jobs=4))

cv_means = []
cv_std = []
for cv_result in cv_results:
    cv_means.append(cv_result.mean())
    cv_std.append(cv_result.std())
cv_res = pd.DataFrame({"CrossValMeans":cv_means,"CrossValerrors": cv_std,"Algorithm":["SVC","KNeighbors","GaussianNB"]})
g = sns.barplot("CrossValMeans","Algorithm",data = cv_res, palette="Set3",orient = "h",**{'xerr':cv_std})
g.set_xlabel("Mean Accuracy")
g = g.set_title("Cross validation scores")
```

Results



The SVM Classifier gives the highest accuracy score than other two Classifier.

QUESTION-2

Choose any dataset of your choice. Apply K-means on the dataset and visualize the clusters using matplotlib or seaborn.

- Report which K is the best using the elbow method.
- Evaluate with silhouette score or other scores relevant for unsupervised approaches (before applying clustering clean the data set with the EDA learned in the class)

Solution:

Loading the Data

```
#import packages required for clustering
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn.decomposition import pca
from sklearn.model_selection import KFold, cross_val_score, train_test_split
from sklearn.metrics import *
from IPython.display import display
import random
import warnings
warnings.filterwarnings("ignore")
from sklearn.metrics import silhouette_score
%matplotlib inline
```

```
#read the data
data=pd.read_csv('Wine.csv')
```

```
data.sample(5)
```

	Alcohol	Malic_Acid	Ash	Ash_Alcanity	Magnesium	Total_Phenols	Flavanoids	Nonflavanoid_Phenols	Proanthocyanins	Color_Intensity	Hue	OD28
105	12.42	2.55	2.27	22.0	90	1.68	1.84	0.66	1.42	2.70	0.86	3.3
90	12.08	1.83	2.32	18.5	81	1.60	1.50	0.52	1.64	2.40	1.08	2.2
153	13.23	3.30	2.28	18.5	98	1.80	0.83	0.61	1.87	10.52	0.56	1.5
3	14.37	1.95	2.50	16.8	113	3.85	3.49	0.24	2.18	7.80	0.86	3.4
159	13.48	1.67	2.64	22.5	89	2.60	1.10	0.52	2.29	11.75	0.57	1.7

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 178 entries, 0 to 177
Data columns (total 14 columns):
Alcohol                 178 non-null float64
Malic_Acid              178 non-null float64
Ash                     178 non-null float64
Ash_Alcanity            178 non-null float64
Magnesium               178 non-null int64
Total_Phenols           178 non-null float64
Flavanoids              178 non-null float64
Nonflavanoid_Phenols    178 non-null float64
Proanthocyanins         178 non-null float64
Color_Intensity         178 non-null float64
Hue                     178 non-null float64
OD280                   178 non-null float64
Proline                 178 non-null int64
Customer_Segment        178 non-null int64
dtypes: float64(11), int64(3)
memory usage: 19.5 KB
```

```
data.describe()
```

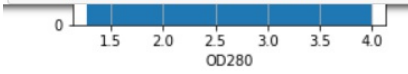
	Alcohol	Malic_Acid	Ash	Ash_Alcanity	Magnesium	Total_Phenols	Flavanoids	Nonflavanoid_Phenols	Proanthocyanins	Color_Intensity
count	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000
mean	13.000618	2.336348	2.366517	19.494944	99.741573	2.295112	2.029270	0.361854	1.590899	5.058090
std	0.811827	1.117146	0.274344	3.339564	14.282484	0.625851	0.998859	0.124453	0.572359	2.318286
min	11.030000	0.740000	1.360000	10.600000	70.000000	0.980000	0.340000	0.130000	0.410000	1.280000
25%	12.362500	1.602500	2.210000	17.200000	88.000000	1.742500	1.205000	0.270000	1.250000	3.220000
50%	13.050000	1.865000	2.360000	19.500000	98.000000	2.355000	2.135000	0.340000	1.555000	4.690000
75%	13.677500	3.082500	2.557500	21.500000	107.000000	2.800000	2.875000	0.437500	1.950000	6.200000
max	14.830000	5.800000	3.230000	30.000000	162.000000	3.880000	5.080000	0.660000	3.580000	13.000000

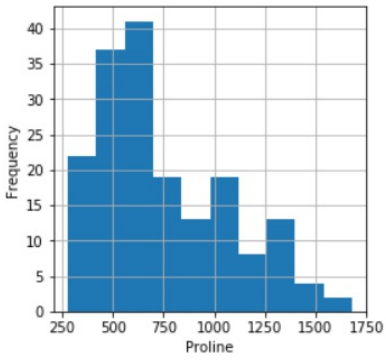
```
data.isnull().any()
```

```
Alcohol                False
Malic_Acid             False
Ash                    False
Ash_Alcanity           False
Magnesium              False
Total_Phenols          False
Flavanoids             False
Nonflavanoid_Phenols   False
Proanthocyanins        False
Color_Intensity        False
Hue                    False
OD280                  False
Proline                False
Customer_Segment       False
dtype: bool
```

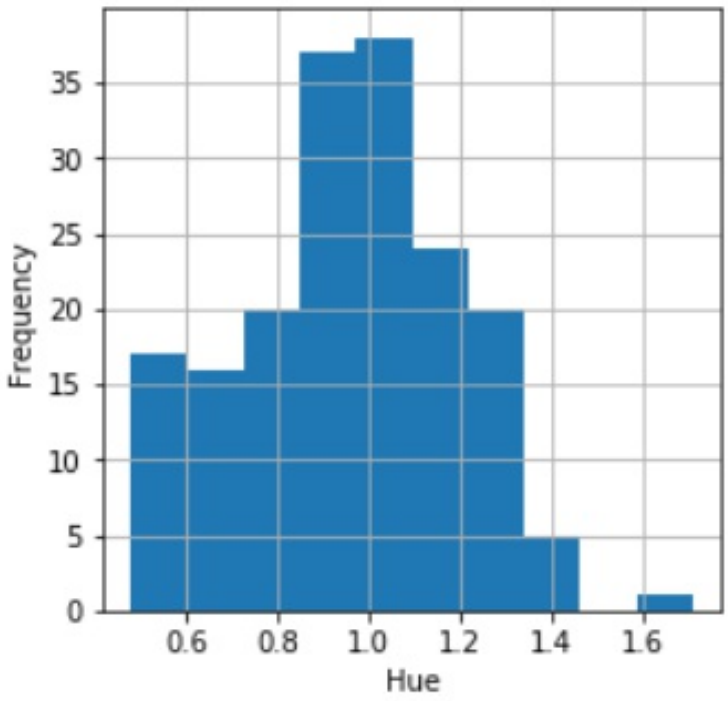
Data Exploration

```
for i in data.columns:
    plt.figure(figsize=(4,4))
    data[i].hist()
    plt.xlabel(str(i))
    plt.ylabel("Frequency")
```

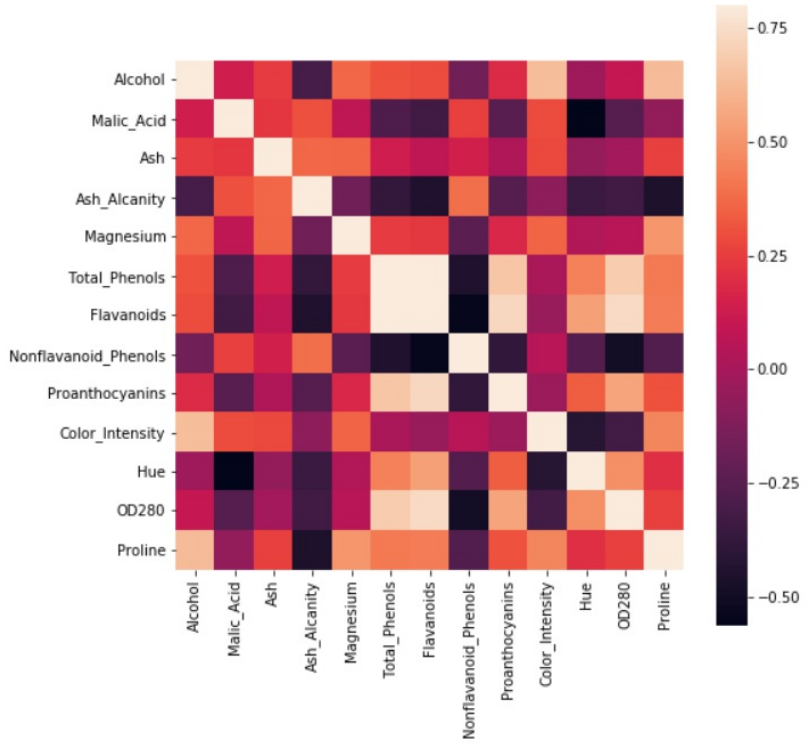





```
#drop the column which is not useful
data.drop('Customer_Segment',axis=1,inplace=True)
```




```
corrmat = data.corr(method='spearman')
f, ax = plt.subplots(figsize=(8, 8))

# Draw the heatmap using seaborn
sns.heatmap(corrmat, vmax=.8, square=True)
plt.show()
```



Applying K-mean Clustering model

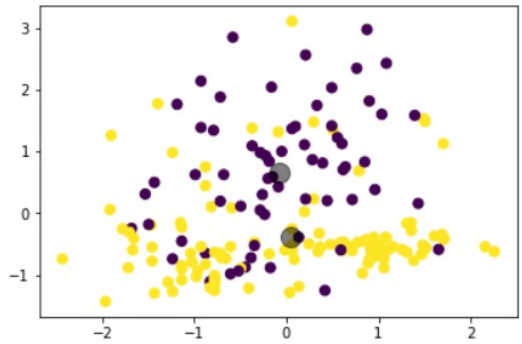
```
#preprocess the data and apply mean normalizaion for every column
from sklearn import preprocessing

scaler = preprocessing.StandardScaler()

scaler.fit(data)
X_scaled_array = scaler.transform(data)
X_scaled = pd.DataFrame(X_scaled_array, columns = data.columns)
```

```
#model
#apply kmeans clustering algorithm
kmeans=KMeans(n_clusters=2)
kmeans.fit(X_scaled)
#cluster centers
print(kmeans.cluster_centers_)
```

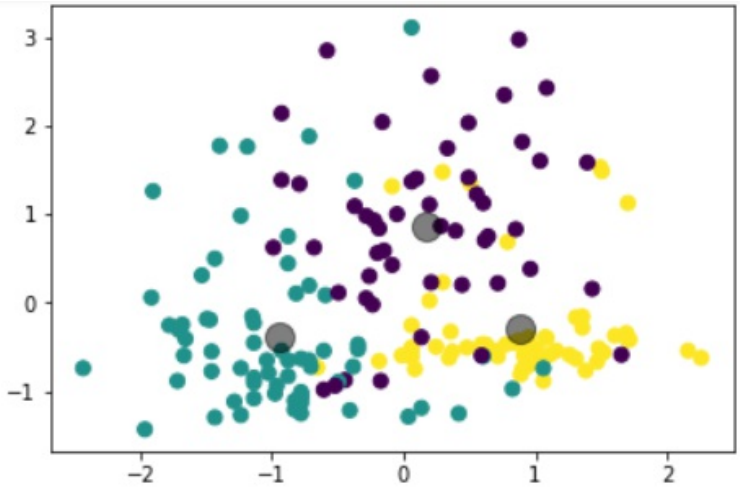
```
[[-0.07297886  0.66451047  0.18987553  0.51662255 -0.15468782 -0.94370681
  -1.04663574  0.83794915 -0.71615568  0.54346866 -0.88207198 -1.06931809
  -0.45318098]
 [ 0.04197899 -0.38224053 -0.10922044 -0.29717226  0.08897972  0.5428402
   0.60204711 -0.48200615  0.41194796 -0.31261472  0.50738654  0.61509448
   0.26067933]]
```

Making Clusters of different Values and Calculating Silhoutte Score

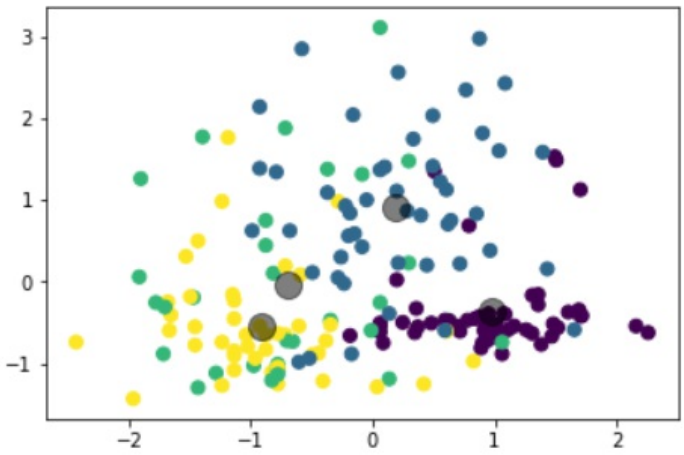
```
#find the no of clusters
wcss = []
##elbow method to know the number of clusters
for i in range(2,11):
    kmeans = KMeans(n_clusters=i,init='k-means++',max_iter=300,n_init=10,random_state=0)
    kmeans.fit(X_scaled)
    cluster_an=kmeans.predict(X_scaled)
    wcss.append(kmeans.inertia_)
    plt.scatter(X_scaled_array[:,0],X_scaled_array[:,1],c=cluster_an,s=50)
    centers=kmeans.cluster_centers_
    plt.scatter(centers[:,0],centers[:,1],c='black',s=200,alpha=0.5)
    plt.show()
    score = silhouette_score (X_scaled, cluster_an, metric='euclidean')
    print ("For n_clusters = {}, silhouette score is {})".format(i, score))
```



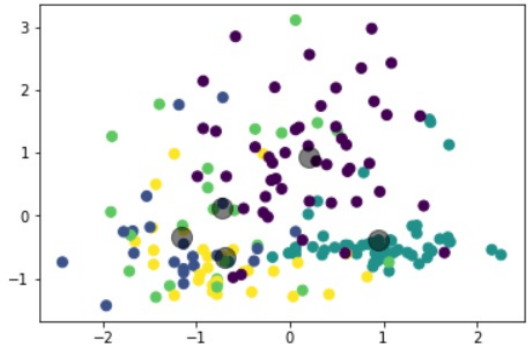

```
For n_clusters = 2, silhouette score is 0.2683134097105213)
```



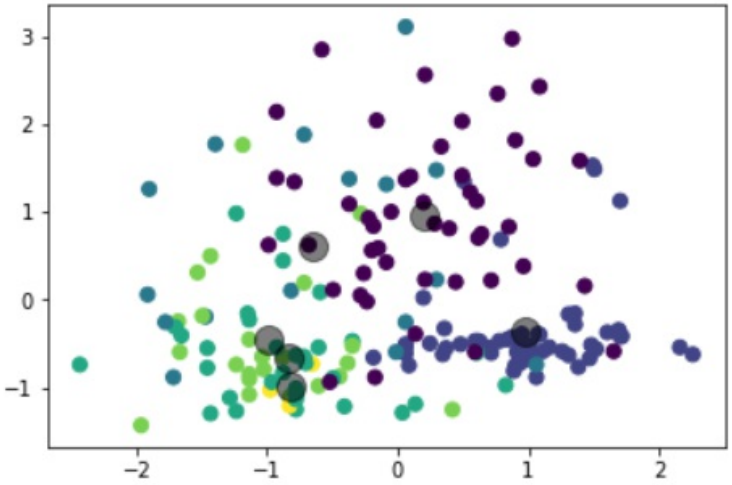

```
For n_clusters = 3, silhouette score is 0.28594199657074876)
```



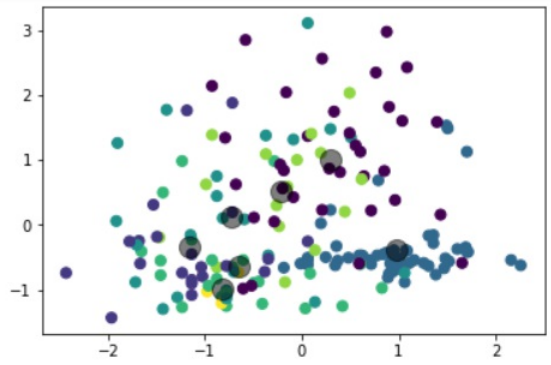
For n_clusters = 4, silhouette score is 0.251733430116964 75)



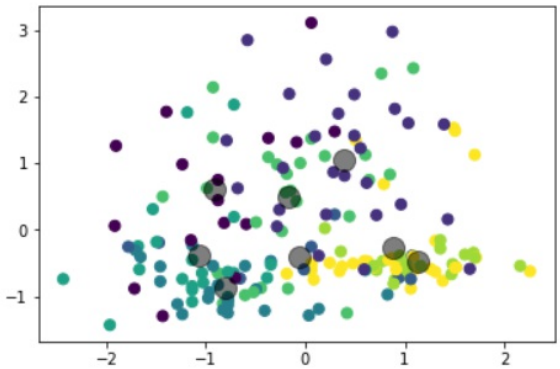
For n_clusters = 5, silhouette score is 0.23187479572412723)



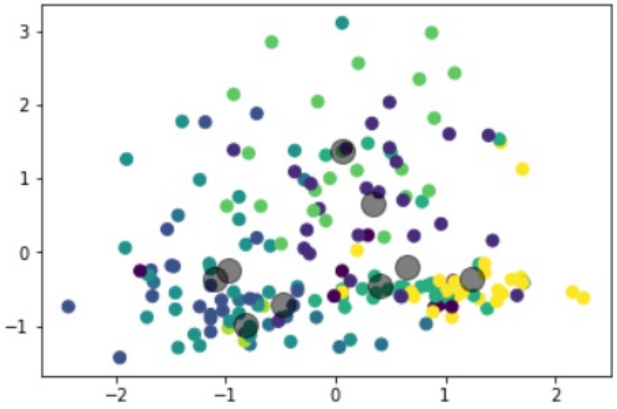
For n_clusters = 6, silhouette score is 0.23964277899912415)



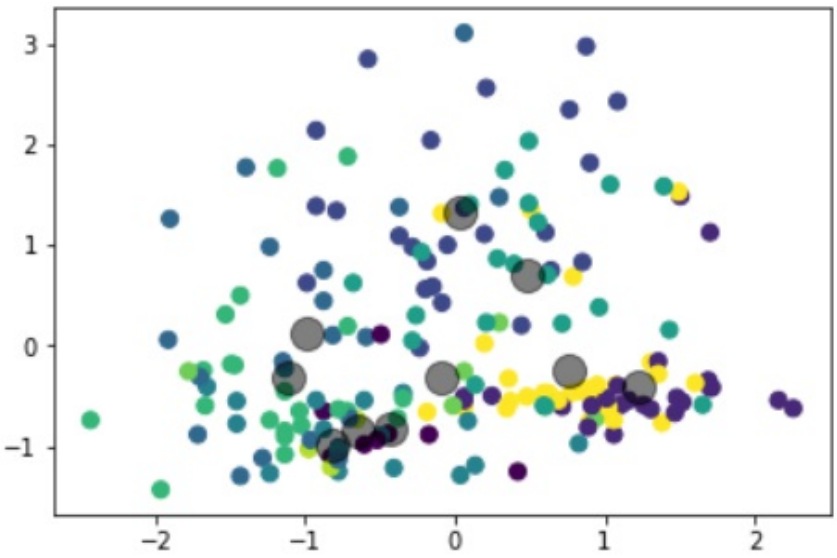
For n_clusters = 7, silhouette score is 0.1977124515910614)



For n_clusters = 8, silhouette score is 0.133114891253478)



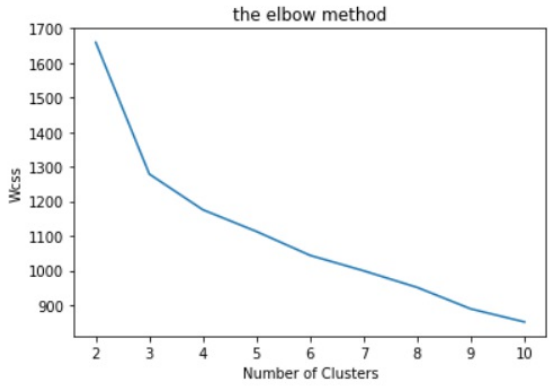
For n_clusters = 9, silhouette score is 0.14035373736325835)



```
For n_clusters = 10, silhouette score is 0.14398677296462006)
```

Elbow Method Analysis

```
#plot the data for elbow method
plt.plot(range(2,11),wcss)
plt.title('the elbow method')
plt.xlabel('Number of Clusters')
plt.ylabel('Wcss')
plt.show()
```



QUESTION-3

Write a program in which take an Input file, use the simple approach below to summarize a text file:Link to input file: https://umkc.box.com/s/7by0f4540cdbdp3pm60h5fxxffefsvrwa.

- Read the data from a file
- Tokenize the text into words and apply lemmatization technique on each word.
- Find all the trigrams for the words.
- Extract the top 10 of the most repeated trigrams based on their count.
- Go through the text in the file
- Find all the sentences with the most repeated tri-gramsg. Extract those sentences and concatenateh. Print the concatenated result

> ***Solution***

Read the data from File

```
# 3. Write a program in which take an Input file, use the simple approach below to summarize a text file:
# Link to input file: https://umkc.box.com/s/7by0f4540cdbdp3pm60h5fxxffefsvrw
# a. Read the data from a file
# b. Tokenize the text into words and apply lemmatization technique on each word.
# c. Find all the trigrams for the words.
# d. Extract the top 10 of the most repeated trigrams based on their count.
# e. Go through the text in the file
# f. Find all the sentences with the most repeated tri-grams
# g. Extract those sentences and concatenate
# h. Print the concatenated result

import nltk
from nltk.stem import WordNetLemmatizer
from nltk.util import ngrams

# Part a. Read the data from a file
with open('nlp_input.txt') as data:
    text = data.read().strip()

# For proper creation of sentence tokens, some text needs to be replaced. In particular ?? and (?) are used
# which causes the sentence tokenizer to break up the sentence incorrectly.
text = text.replace('??', 'qq')
text = text.replace('(?)', '(q)')

# Periods were added after titles to break them apart in sentence tokens. Also fixed some typos.
text = text.replace('Setosa.io)', 'Setosa.io).')
text = text.replace('Equation for least ordinary squares', 'Equation for least ordinary squares.')
text = text.replace('These are known as L1 regularization(Lasso regression) and L2 regularization(ridge regression).',
                    'These are known as L1 regularization(Lasso regression) and L2 regularization(ridge regression). ')
text = text.replace('L2 regularization penalty term', 'L2 regularization penalty term.')
text = text.replace('n(', 'n (')
text = text.replace('L1 regularization penalty term', 'L1 regularization penalty term.')
text = text.replace('significantly higher weight than the rest', 'significantly higher weight than the rest.')
text = text.replace('Performing Lasso regression', 'Performing Lasso regression.')
text = text.replace('Performing Elastic Net regression', 'Performing Elastic Net regression.')
text = text.replace('1 denotes lasso)', '1 denotes lasso).')
```

Tokenizing into words and applying lemmatization

```
# Part b. Tokenize the text into words and apply lemmatization technique on each word.
# Tokenize the text into words
from string import punctuation
stoplist = set(list(punctuation))

# Punctuation was not wanted to be part of the tokens created
wtokens = [token for token in nltk.word_tokenize(text) if token.lower() not in stoplist]

for i in wtokens:
    print(i)

print()
print()
print()
```

```
Regression
analysis
is
a
statistical
technique
that
models
and
approximates
the
relationship
between
a
dependent
and
one
or
more
independent
```

In [7]:

```
#Lemmatize each word
lemmat = WordNetLemmatizer()

for l in wtokens:
    print(lemmat.lemmatize(l))

print()
print()
print()
```

Finding Trigrams

```
more
independent
variable
This
article
will
quickly
introduce
three
commonly
used
regression
model
using
R
and
the
Boston
housing
```

In [8]:
```
# Part c. Find all the trigrams for the words.
trigrams = ngrams(wtokens,3)
for i in trigrams:
    print(i)
print()
print()
print()
```

```
('Regression', 'analysis', 'is')
('analysis', 'is', 'a')
('is', 'a', 'statistical')
('a', 'statistical', 'technique')
('statistical', 'technique', 'that')
('technique', 'that', 'models')
('that', 'models', 'and')
('models', 'and', 'approximates')
('and', 'approximates', 'the')
('approximates', 'the', 'relationship')
('the', 'relationship', 'between')
('relationship', 'between', 'a')
('between', 'a', 'dependent')
('a', 'dependent', 'and')
('dependent', 'and', 'one')
('and', 'one', 'or')
('one', 'or', 'more')
('or', 'more', 'independent')
('more', 'independent', 'variables')
('independent', 'variables', 'This')
```

```
# Part d. Extract the top 10 of the most repeated trigrams based on their count.
# find all trigrams and their frequency using functions from the nltk
temp = nltk.collocations.TrigramCollocationFinder.from_words(wtokens)
# Sort trigrams in the frequency they appear
trigrams2 = sorted(temp.ngram_fd.items(), key=lambda t: (-t[1], t[0]))

# Extract the first top 10 trigrams
runs = 0
most_trigrams = ()
for t in trigrams2:
    most_trigrams = most_trigrams + (t,)
    runs += 1
    if runs >= 10:
        break

for g in most_trigrams:
    print(g)
print()
print()
print()
```

```
(('we', 'need', 'to'), 3)
(('Performing', 'Elastic', 'Net'), 2)
(('We', 'can', 'see'), 2)
(('a', 'lambda', 'value'), 2)
(('a', 'number', 'of'), 2)
(('alpha', 'and', 'lambda'), 2)
(('and', 'L2', 'regularization'), 2)
(('equal', 'to', 'the'), 2)
(('find', 'the', 'optimal'), 2)
(('is', 'equal', 'to'), 2)
```

```
# Part e. Go through the text in the file
# Part f. Find all the sentences with the most repeated tri-grams
# Part g. Extract those sentences and concatenate
# Tokenize the sentences
stokens = nltk.sent_tokenize(text)

# Pulled out part of the tuple trigrams to use for searching for sentences with certain trigrams
most_trigrams2 = ()
for i,u in most_trigrams:
    most_trigrams2 = most_trigrams2 + (i,)

# The separated part of the tuples were converted to a list then joined them together to form a string.
# The trigram strings were then put in a list
tristrings = []
temporary = list(most_trigrams2)
for d in range(0,10):
    tempstring = ' '.join(temporary[d])
    tristrings.append(tempstring)

# The tuple stokes was converted into a list.
Sentlist = list(stokens)

# Sentences with the top 10 trigrams were found and concatenated together.
Sentences = []
temps = []

for k in range(len(tristrings)):
    # Reset temporary strings and lists. These need to be reset whenever the k value moves to the next trigram string
    temps2 = ''
    temps = []
    for i in range(len(Sentlist)):
        # Check to see if trigram string is in the Sentence string
        if tristrings[k] in Sentlist[i]:
            # Append Sentences to a list
            ''
            temps.append(Sentlist[i])
    # After all Sentences have been found with a selected trigram string. The sentences in the temporary list
    # are joined together to form 1 string. That string is then appended to the final list storage.
    temps2 = ' '.join(temps)
    Sentences.append(temps2)

# Part h. Print the concatenated result
q = 1
for i in Sentences:
    print(q)
    print(i)
    print()
    q += 1
```

5
The gradient descent algorithm is used to find the optimal cost function by going over a number of iterations. We will tune the model by iterating over a number of alpha and lambda pairs and we can see which pair has the lowest associated error.

6
Performing Elastic Net requires us to tune parameters to identify the best alpha and lambda values and for this we need to u se the caret package. We will tune the model by iterating over a number of alpha and lambda pairs and we can see which pair has the lowest associated error.

7
These are known as L1 regularization (Lasso regression) and L2 regularization (ridge regression). A third commonly used mode l of regression is the Elastic Net which incorporates penalties from both L1 and L2 regularization:
In addition to setting and choosing a lambda value elastic net also allows us to tune the alpha parameter where qq = 0 corre sponds to ridge and qq = 1 to lasso.

8
The L2 term is equal to the square of the magnitude of the coefficients. The penalty applied for L2 is equal to the absolute value of the magnitude of the coefficients:
L1 regularization penalty term.

9
The equation for this model is referred to as the cost function and is a way to find the optimal error by minimizing and mea suring it. The gradient descent algorithm is used to find the optimal cost function by going over a number of iterations.

10
The L2 term is equal to the square of the magnitude of the coefficients. The penalty applied for L2 is equal to the absolute value of the magnitude of the coefficients:
L1 regularization penalty term.

QUESTION-4

Create Multiple Regression by choosing a dataset of your choice (again before evaluating, clean the data set with the EDA learned in the class). Evaluate the model using RMSE and R2 and also report if you saw any improvement before and after the EDA.

Solution:

Loading the Data

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
```

```
data = pd.read_csv('data50.csv',index_col=0)
```

```
data.sample(5)
```

	R&D Spend	Administration	Marketing Spend	State	Profit
37	44069.95	51283.14	197029.42	California	89949.14
29	65605.48	153032.06	107138.38	New York	101004.64
42	23640.93	96189.63	148001.11	California	71498.49
18	91749.16	114175.79	294919.57	Florida	124266.90
11	100671.96	91790.61	249744.55	California	144259.40

```
# check for missing values
data.isnull().sum().sort_values(ascending=False)
```

```
Profit            0
State             0
Marketing Spend   0
Administration    0
R&D Spend         0
dtype: int64
```

Exploring the Data

```
# Table of Missing values
missing= data.isnull().sum().sort_values(ascending=False)
percent= (data.isnull().sum()/data.isnull().count())
total= pd.concat([missing, percent],axis=1, keys=["Total", "Percent"])
print(total)
```

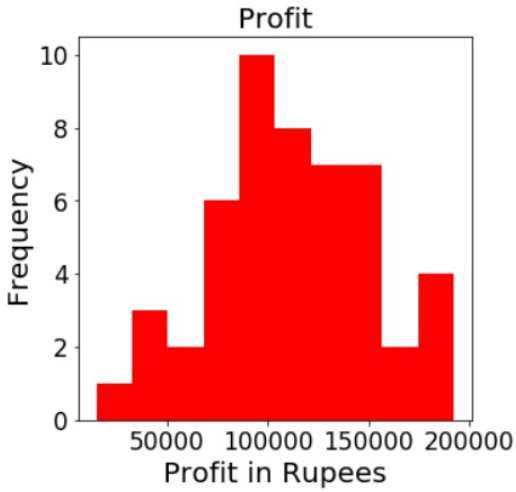
```
                Total  Percent
Administration      0      0.0
Marketing Spend     0      0.0
Profit              0      0.0
R&D Spend           0      0.0
State               0      0.0
```

```
E:\anaconda\lib\site-packages\ipykernel_launcher.py:4: FutureWarning: Sorting because non-concatenation axis is not aligned.
A future version
of pandas will change to not sort by default.

To accept the future behavior, pass 'sort=True'.

To retain the current behavior and silence the warning, pass sort=False

  after removing the cwd from sys.path.
```

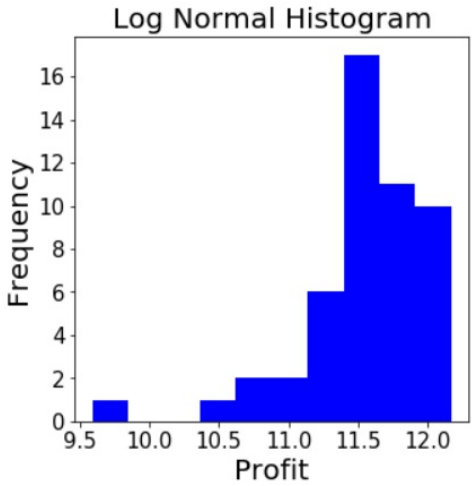
```
# Get all the categorical variables
categorical= data.select_dtypes(include=[np.object])
categorical.sample(5)
```

	State
44	California
12	Florida
21	New York
27	New York
3	New York

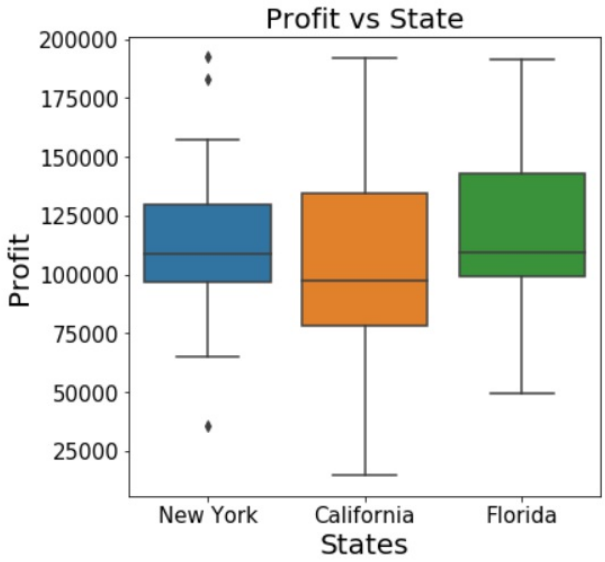
```
# Histogram of target variable
plt.figure(figsize=(5,5))
plt.hist(data["Profit"],color="red")
plt.title("Profit", size=20)
plt.ylabel("Frequency", size=20)
plt.xlabel("Profit in Rupees", size=20)
plt.tick_params(labelsize=17)
# plt.tick_params(labelsize=20)
# ax.xaxis.set_tick_params(labelsize=20)
# ax.yaxis.set_tick_params(labelsize=20)
plt.show()
```



```
# Log normal histogram
target= np.log(data["Profit"])
plt.figure(figsize=(5,5))
plt.hist(target, color="blue")
plt.xlabel("Profit", size=20)
plt.ylabel("Frequency", size=20)
plt.title("Log Normal Histogram", size=20)
plt.tick_params(labelsize=15)
plt.show()
```

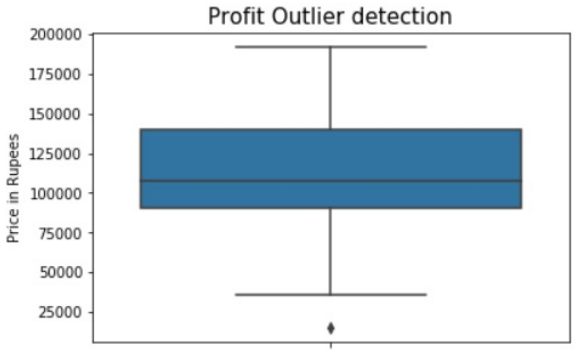


```
# Boxplot of Profit and Marketing spend
plt.figure(figsize=(6,6))
sns.boxplot(y="Profit", x="State", data=data)
plt.xlabel("States", size=20)
plt.ylabel("Profit", size=20)
plt.title("Profit vs State", size=20)
plt.tick_params(labelsize=15)
plt.show()
```



Detecting and Removing Outliers

```
# Outlier detection
# plt.figure(figsize=(12,6))
sns.boxplot(data["Profit"],orient= "v")
plt.title("Profit Outlier detection", size=15)
plt.xlabel("", size=15)
plt.ylabel("Price in Rupees")
plt.show()
```



```
#Remove outlier
#data[data["Profit"] < 25000]
data.drop(data[data["Profit"] < 20000].index)
```

	R&D Spend	Administration	Marketing Spend	State	Profit
0	165349.20	136897.80	471784.10	New York	192261.83
1	162597.70	151377.59	443898.53	California	191792.06
2	153441.51	101145.55	407934.54	Florida	191050.39
3	144372.41	118671.85	383199.62	New York	182901.99
4	142107.34	91391.77	366168.42	Florida	166187.94
5	131876.90	99814.71	362861.36	New York	156991.12
6	134615.46	147198.87	127716.82	California	156122.51
7	130298.13	145530.06	323876.68	Florida	155752.60
8	120542.52	148718.95	311613.29	New York	152211.77
9	123334.88	108679.17	304981.62	California	149759.96
10	101913.08	110594.11	229160.95	Florida	146121.95
11	100671.96	91790.61	249744.55	California	144259.40
12	93863.75	127320.38	249839.44	Florida	141585.52
13	91992.39	135495.07	252664.93	California	134307.35

14	119943.24	156547.42	256512.92	Florida	132602.65
15	114523.61	122616.84	261776.23	New York	129917.04
16	78013.11	121597.55	264346.06	California	126992.93
17	94657.16	145077.58	282574.31	New York	125370.37
18	91749.16	114175.79	294919.57	Florida	124266.90
19	86419.70	153514.11	0.00	New York	122776.86
20	76253.86	113867.30	298664.47	California	118474.03
21	78389.47	153773.43	299737.29	New York	111313.02
22	73994.56	122782.75	303319.26	Florida	110352.25
23	67532.53	105751.03	304768.73	Florida	108733.99
24	77044.01	99281.34	140574.81	New York	108552.04
25	64664.71	139553.16	137962.62	California	107404.34
26	75328.87	144135.98	134050.07	Florida	105733.54
27	72107.60	127864.55	353183.81	New York	105008.31
28	66051.52	182645.56	118148.20	Florida	103282.38
29	65605.48	153032.06	107138.38	New York	101004.64
30	61994.48	115641.28	91131.24	Florida	99937.59
31	61136.38	152701.92	88218.23	New York	97483.56
32	63408.86	129219.61	46085.25	California	97427.84
33	55493.95	103057.49	214634.81	Florida	96778.92
34	46426.07	157693.92	210797.67	California	96712.80
35	46014.02	85047.44	205517.64	New York	96479.51
36	28663.76	127056.21	201126.82	Florida	90708.19
37	44069.95	51283.14	197029.42	California	89949.14

33	55493.95	103057.49	214634.81	Florida	96778.92
34	46426.07	157693.92	210797.67	California	96712.80
35	46014.02	85047.44	205517.64	New York	96479.51
36	28663.76	127056.21	201126.82	Florida	90708.19
37	44069.95	51283.14	197029.42	California	89949.14
38	20229.59	65947.93	185265.10	New York	81229.06
39	38558.51	82982.09	174999.30	California	81005.76
40	28754.33	118546.05	172795.67	California	78239.91
41	27892.92	84710.77	164470.71	Florida	77798.83
42	23640.93	96189.63	148001.11	California	71498.49
43	15505.73	127382.30	35534.17	New York	69758.98
44	22177.74	154806.14	28334.72	California	65200.33
45	1000.23	124153.04	1903.93	New York	64926.08
46	1315.46	115816.21	297114.46	Florida	49490.75
47	0.00	135426.92	0.00	California	42559.73
48	542.05	51743.15	0.00	New York	35673.41

```
# State is a categorical varible which we need to convert to numeric
data["State"].value_counts()
```

```
California    17
New York      17
Florida       16
Name: State, dtype: int64
```

Split the Data for Validation

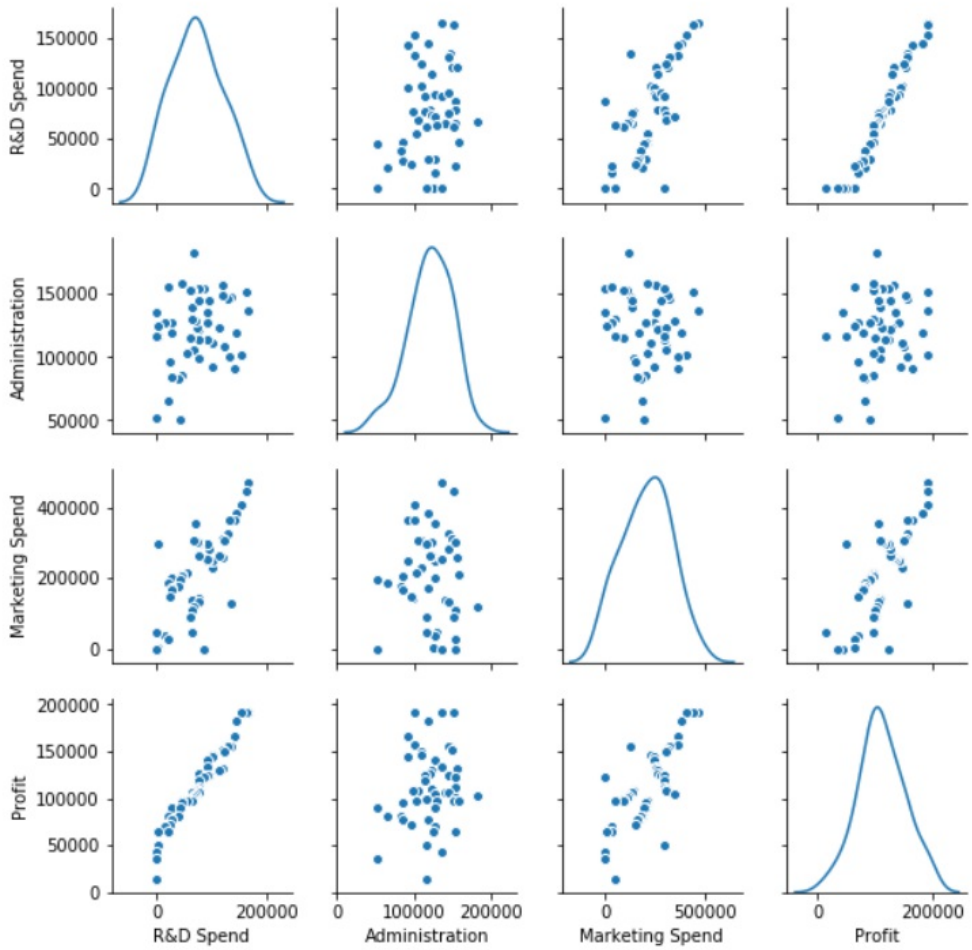
```
# Split the data into training and test set
X = data.iloc[:, :-2].values
y = data.iloc[:, 4].values
```

```
X.shape
```

```
(50, 3)
```

```
X=X[:,1:]
```

```
# Pairplot of numeric variables
columns= ["R&D Spend", "Administration", "Marketing Spend", "Profit"]
sns.pairplot(data[columns],size=2, kind="scatter", diag_kind="kde")
plt.show()
```



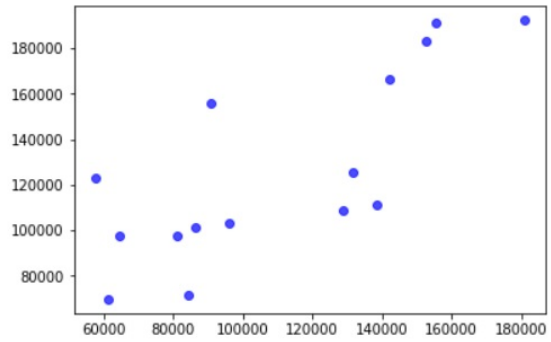

```
# Split data into training and testing set
from sklearn.cross_validation import train_test_split
X_train, X_test, Y_train, Y_test= train_test_split(X,y,test_size=0.3, random_state=5)
```

```
from sklearn.linear_model import LinearRegression
regressor=LinearRegression()
regressor.fit(X_train,Y_train)
y_pred=regressor.predict(X_test)
```

```
# rmse on train set
from sklearn.metrics import mean_squared_error,r2_score
rmse = np.sqrt(mean_squared_error(Y_test, y_pred))
print("Root Mean Squared Error: {}".format(rmse))
r2=r2_score(Y_test,y_pred)
print("R2 score: {}".format(r2))
```

```
Root Mean Squared Error: 31020.071177180747
R2 score: 0.3937345527145165
```

```
plt.scatter(y_pred,Y_test,alpha=0.7,color='b')
plt.show()
```



After Applying OneHotEncoding there is change in the scores

```
X = data.iloc[:, :-1].values
y=data.iloc[:,4]
```

```
# Encoding categorical data
from sklearn.preprocessing import LabelEncoder, OneHotEncoder

labelencoder= LabelEncoder()
X[:,3]= labelencoder.fit_transform(X[:, 3])

onehotencoder= OneHotEncoder(categorical_features=[3])
X= onehotencoder.fit_transform(X).toarray()
```

```
# Split data into training and testing set
from sklearn.cross_validation import train_test_split
X_train, X_test, Y_train, Y_test= train_test_split(X,y,test_size=0.3, random_state=5)
```

```
from sklearn.linear_model import LinearRegression
regressor=LinearRegression()
regressor.fit(X_train,Y_train)
y_pred=regressor.predict(X_test)
```

```
# rmse on train set
from sklearn.metrics import mean_squared_error,r2_score
rmse = np.sqrt(mean_squared_error(Y_test, y_pred))
print("Root Mean Squared Error: {}".format(rmse))
r2=r2_score(Y_test,y_pred)
print("R2 score: {}".format(r2))
```

```
Root Mean Squared Error: 6246.578410351991
R2 score: 0.9754154859052265
```

```
plt.scatter(y_pred,Y_test,alpha=0.7,color='b')
plt.show()
```