PRACTICAL FILE

Machine Learning And Information retrieval

(ITMDAE18)

Name – Kurakula James Paul
Roll No – 2020PMD4211
Course – M. Tech (Machine Intelligence and Data Analytics)

under the guidance of

MR. VIKAS MAHESHKAR

Department of Information Technology

Index

S.no	Practical	Date	Page no
1	Character recognition system using python	31/05/2021	1
2	Bayes Classifier using Single Feature	07/06/2021	4
3	Bayes classifiers with multiple class	07/06/2021	8
4	character recognition system using Neural Network.	14/06/2021	10
5	Text Processing using Bayes Classifier.	21/06/2021	14
6	K-fold Cross Validation(knn)	28/06/2021	19
7	Linear Regression with multiple features	28/06/2021	23
8	Logistic regression using Python.	05/07/2021	28
9	Regularization using Python.	05/07/2021	32
10	Weighted KNN	12/07/2021	36
11	K-means using Python.	26/07/2021	40
12	Implement PCA	19/07/2021	44
13	Assignment PCA	19/07/2021	47

Practical-1

Objective:

Design a character recognition system using basic python

Description:

Given a set of 10 characters, identify the features which will distinguish each character in the character set most accurately.

Character set:

{ B , D , E , F , H , K , L , M , N , P }

Features:

1. Number of vertical lines
2. Number of horizontal lines
3. Number of diagonals
4. Number of curves

Alpha	V Lines	H Lines	Diagonal	Curve
B	1	0	0	2
D	1	0	0	1
E	1	3	0	0
F	1	2	0	0
H	2	1	0	0
K	1	0	2	0
L	1	1	0	0
M	2	0	2	0
N	2	0	1	0
P	1	0	0	1

Algorithm:

1. Read the dataset which is hardcoded in csv file using pandas.
2. Define a function which will calculate Euclidean distance for unknown/new character by using below formula .

dist(new,compare)= square root(∑(new[i]-compare)^2).

3. For new character calculate the Euclidean distance for each character present in dataset and output the one which has least distance.

Modified dataset after preprocessing:

Alpha	V Lines	H Lines	Curve	closed region
B	1	0	2	2
D	1	0	1	1
E	1	3	0	0
F	1	2	0	0
H	2	1	0	0
K	1	0	0	0
L	1	1	0	0
M	2	0	0	0
N	2	0	0	0
P	1	0	1	1

Implementation:

```
# 1 import dataset

import pandas as pd
features = pd.read_csv('features.csv')


# 2 calculate Euclidean distance

import math
def predict(new_char,features):
    min_dist = 99
    best_char = []
    count = features.shape[0]
    for i in range(count):
```

```
        s = [(features.loc[i][j+1]-new_char[j])**2 for j in range(features.shape[1]-1)]
        eDist = math.sqrt(sum(s))
        #print(eDist)
        if( eDist <= min_dist ):
            if(eDist == min_dist):
                best_char.append(i)
            else:
                best_char.clear()
                best_char.append(i)
            min_dist = eDist
    return [features.loc[i][0] for i in best_char]

# 3 input new char

input_data = list()
for i in features_new.columns[1:]:
    input_data.append(int(input(str(i)+' : ')))

# 4 output

print(predict(input_data,features_new))
```

Output:

Actual Char	Before post-processing		After post-processing	
	Input(V,H,D,C)	Output	Input(V,H,D,C)	Output
F	[1,2,0,1]	F	[1,2,0,0]	F
L	[1,1,0,1]	D,L,P	[1,1,0,0]	L
M	[2,0,0,2]	B	[2,0,0,0]	M,N

Practical-2

Objective:

implement Bayes Classifier using Single Feature.

Description:

1. Bayes' theorem is also known as Bayes' Rule or Bayes' law, which is used to determine the probability of a hypothesis with prior knowledge. It depends on the conditional probability.

2. The formula for Bayes' theorem is given as:

$$P(A|B)=P(B|A) * P(A) / P(B)$$

3. P(A|B) is Posterior probability: Probability of hypothesis A on the observed event B.

4. P(A) is Prior Probability: Probability of hypothesis before observing the evidence.

5. P(B) is Marginal Probability: Probability of Evidence. 6. Suppose we have a dataset of Status of Subject Class and corresponding target variable "Held Status". So using this dataset we need to decide that whether that class happened or not on a particular day. So to solve this problem, we need to follow the below steps: -

Generate Likelihood table by finding the probabilities of given features.
Now, use Bayes theorem to calculate the posterior probability.

Dataset:

1. Have a dataset of Subject Names and corresponding target variable "Yes/No/Canceled" based on class held Status.

2. Have defined three Subject name i.e. Natural Language Processing, Mobile Computing, and Network and Web Security .

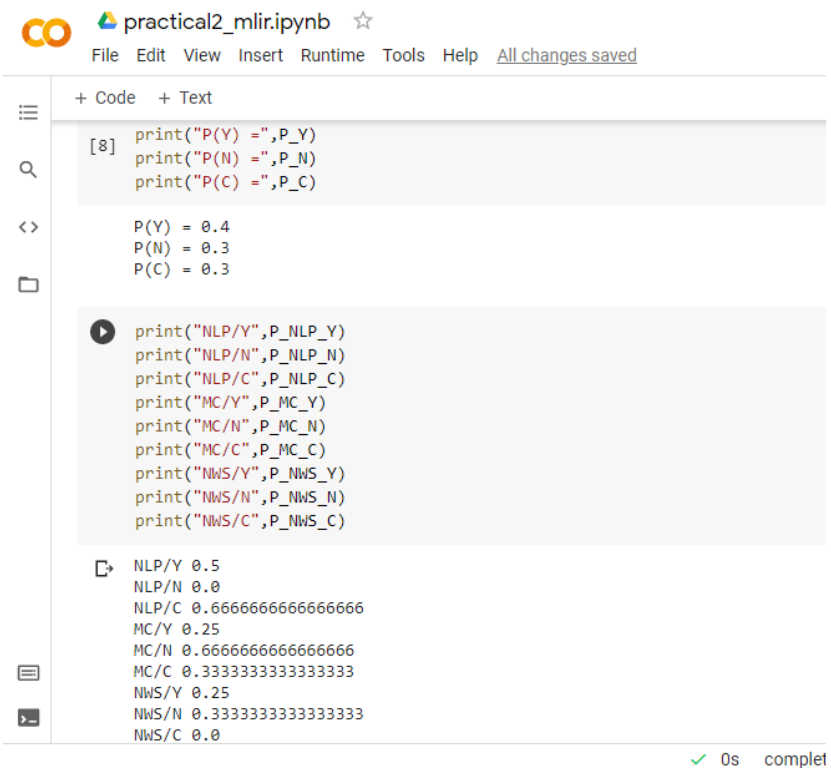
3. Have three target values that describes the Status of Class Held : Yes/No/Canceled.

Implementation:

```
#Import libraries
import array
import sys
import math

#Create Feature & Class List
list1 = ["NLP","MC","MC","NLP","NWS","NWS","NLP","MC","MC","NLP"] list2 =
[“Y”,"N","C","Y","Y","N","C","Y","N","C"]

#Calculating Distinct class count
countY=list2.count('Y')
countN=list2.count('N')
countC=list2.count(‘C')

#Calculating Class Probabilities
X=countY+countN+countC
P_Y=countY/X
P_N=countN/X
P_C=countC/X
count_NLP_Y=0
count_NLP_N=0
count_NLP_C=0
count_MC_Y=0
count_MC_C=0
count_MC_N=0
count_NWS_Y=0
count_NWS_C=0
count_NWS_N=0

for i in range(len(list1)):
        if(list1[i] == "NLP" and list2[i] == "Y"):
                count_NLP_Y=count_NLP_Y+1
        elif(list1[i] == "NLP" and list2[i] == "N"):
                count_NLP_N=count_NLP_N+1
        elif(list1[i] == "NLP" and list2[i] == "C"):
                count_NLP_C=count_NLP_C+1
        elif(list1[i] == "MC" and list2[i] == "N"):
                count_MC_N=count_MC_N+1
        elif(list1[i] == "MC" and list2[i] == "Y"):
                count_MC_Y=count_MC_Y+1
        elif(list1[i] == "MC" and list2[i] == "C"):
                count_MC_C=count_MC_C+1
        elif(list1[i] == "NWS" and list2[i] == "N"):
                count_NWS_N=count_NWS_N+1
        elif(list1[i] == "NWS" and list2[i] == "Y"):
```

```
                count_NWS_Y=count_NWS_Y+1
    elif(list1[i] == "NWS" and list2[i] == "C"):
            count_NWS_C=count_NWS_C+1


P_NLP_Y=count_NLP_Y/countY
P_NLP_N=count_NLP_N/countN
P_NLP_C=count_NLP_C/countC
P_MC_Y=count_MC_Y/countY
P_MC_N=count_MC_N/countN
P_MC_C=count_MC_C/countC
P_NWS_Y=count_NWS_Y/countY
P_NWS_N=count_NWS_N/countN
P_NWS_C=count_NWS_C/countC

#Taking Test Value for Prediction
val=input("Enter Value MC/NLP/NWS : “)
if val == "NLP":
        a=P_NLP_Y*P_Y
        b=P_NLP_N*P_N
        c=P_NLP_C*P_C
elif val == "NWS":
        a=P_NWS_Y*P_Y
        b=P_NWS_N*P_N
        c=P_NWS_C*P_C
elif val == "MC":
        a=P_MC_Y*P_Y
        b=P_MC_N*P_N
        c=P_MC_C*P_C


if (a > b) and (a > c):
        predictedOutput="Y"
elif (b > a) and (b > c):
        predictedOutput="N"
elif (c > a) and (c > b):
        predictedOutput=“C"

print("Predicted Output : ",predictedOutput)
```

Output:


practical2_mlir.ipynb
File Edit View Insert Runtime Tools Help All changes saved
+ Code + Text
[8] print("P(Y) =",P_Y)
print("P(N) =",P_N)
print("P(C) =",P_C)
P(Y) = 0.4
P(N) = 0.3
P(C) = 0.3
print("NLP/Y",P_NLP_Y)
print("NLP/N",P_NLP_N)
print("NLP/C",P_NLP_C)
print("MC/Y",P_MC_Y)
print("MC/N",P_MC_N)
print("MC/C",P_MC_C)
print("NWS/Y",P_NWS_Y)
print("NWS/N",P_NWS_N)
print("NWS/C",P_NWS_C)
NLP/Y 0.5
NLP/N 0.0
NLP/C 0.6666666666666666
MC/Y 0.25
MC/N 0.6666666666666666
MC/C 0.3333333333333333
NWS/Y 0.25
NWS/N 0.3333333333333333
NWS/C 0.0
0s complet


practical2_mlir.ipynb
File Edit View Insert Runtime Tools Help All changes saved
+ Code + Text
[10] #Taking Test Value for Prediction
val=input("Enter Value MC/NLP/NWS : ")
Enter Value MC/NLP/NWS : NWS
[12] if (a > b) and (a > c):
predictedOutput="Y"
elif (b > a) and (b > c):
predictedOutput="N"
elif (c > a) and (c > b):
predictedOutput="C"
print("Predicted Output : ",predictedOutput)
Predicted Output : Y

Practical-3

Objective: To implement Bayes classifier using multiple class and multiple features.

Classes: { yes, no }

Features: chills, runny nose, headache, fever.

Training:

Chills	Runny nose	Headache	Fever	Flu
Y	N	Mild	Y	N
Y	Y	No	N	Y
Y	N	Strong	Y	Y
N	Y	Mild	Y	Y
N	N	No	N	N
N	Y	Strong	Y	Y
N	Y	Strong	N	N
Y	Y	Mild	Y	Y

Testing: Using *Bayes Theorem* to find the class belongingness of any object.

A priori Probabilities : $P(W_i)$ where i = 1, 2, 3 (for all the 3 classes).

Conditional Probabilities : $P(x_j/W_i)$ for feature x which can take values x_j.

Posteriori Probabilities : that an object has a feature x and also belongs to the class W_i.

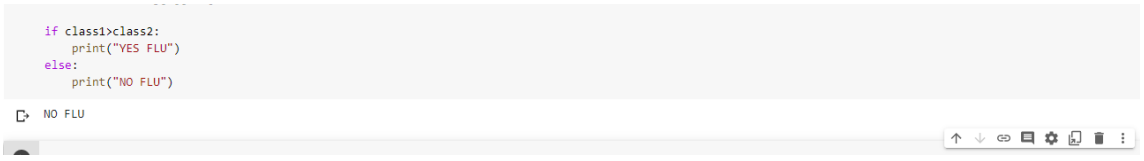
$P(W_i/x) = P(x/W_i) * P(W_i) / P(x)$

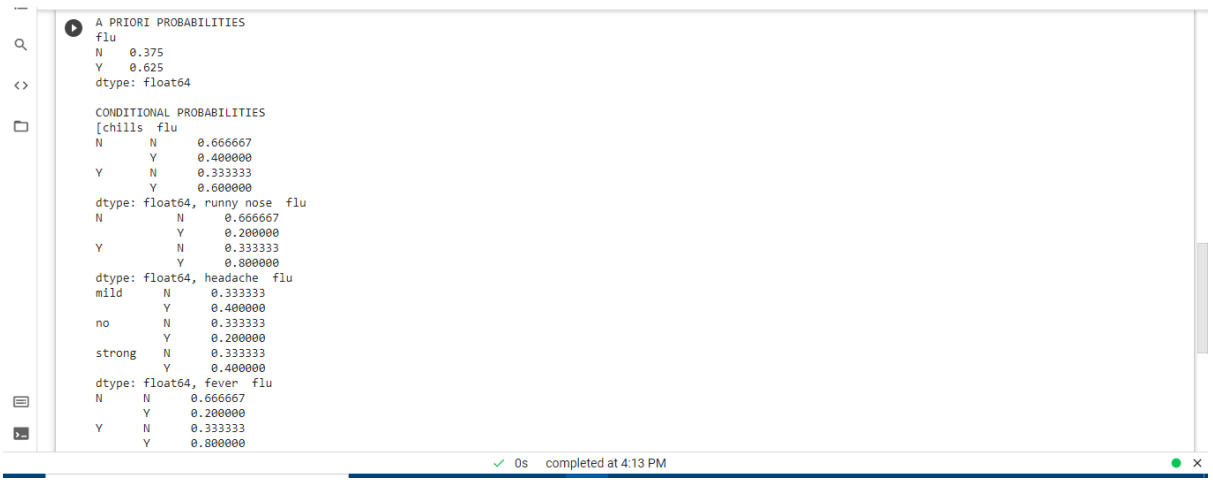
CODE :

```
import pandas as pd

test = input().split()

df = pd.read_csv("data.csv")
g1 = df.groupby(['flu'])
g2 = df.groupby(['chills', 'flu'])
g3 = df.groupby(['runny nose', 'flu'])
g4 = df.groupby(['headache', 'flu'])
g5 = df.groupby(['fever', 'flu'])

apriori_prob = g1.size().div(len(df))
conditional_prob = [g2.size().div(len(df)).div(apriori_prob, axis=0, level='flu'),
                    g3.size().div(len(df)).div(apriori_prob, axis=0, level='flu'),
```

```
                g4.size().div(len(df)).div(apriori_prob, axis=0, level='flu'),
                g5.size().div(len(df)).div(apriori_prob, axis=0, level='flu')]

class1, class2 = apriori_prob[1], apriori_prob[0]

for i, j in zip(test, conditional_prob):
    class1 *= j[i]['Y']
    class2 *= j[i]['N']

if class1 > class2:
    print("YES FLU")
else:
    print("NO FLU")

print()
print("A PRIORI PROBABILITIES")
print(apriori_prob)
print()
print("CONDITIONAL PROBABILITIES")
print(conditional_prob.first())
```

OUTPUT :





Practical-4

Objective :

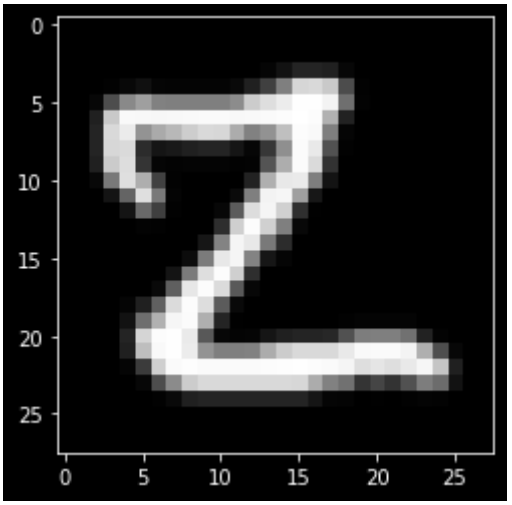
Design a character recognition system using neural network.

Description:

Design and build a convolutional neural network for recognizing characters which are trained using handwritten character set published in EMNIST dataset.

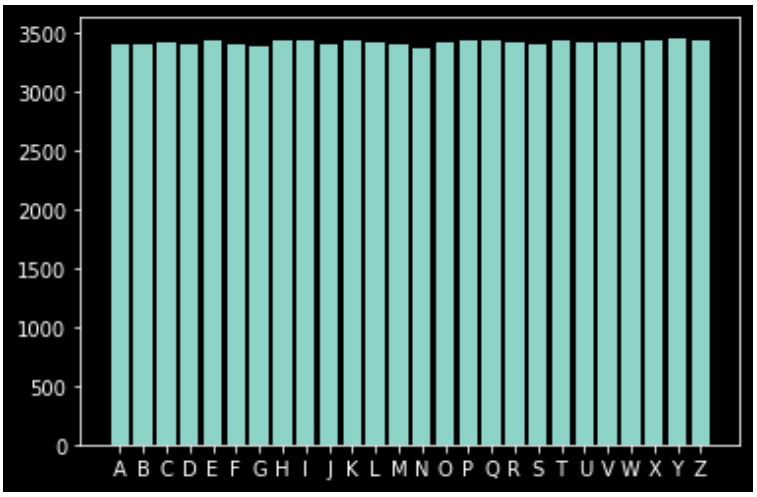
Dataset:

1. Name: EMNIST Letters dataset.
2. Source: https://www.kaggle.com/crawford/emnist
3. Info:
    - The EMNIST Letters dataset merges a balanced set of the uppercase and lowercase letters into a single 26-class.
    - This samples in dataset were collected form high-school students and the census employees.
    - Data is collected from 145,600 samples and is balanced into 26 classes.
    - This dataset contains 88800 train and test images each of size 28x28.
4. A sample form data set

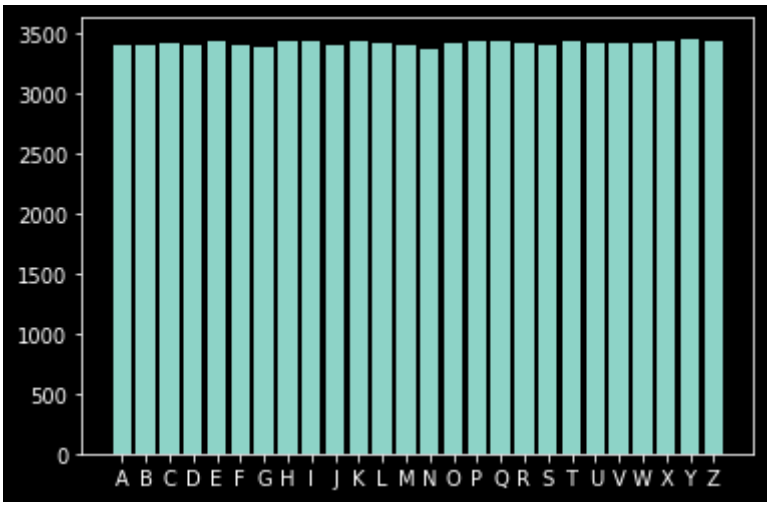


5. Training data and testing data are disjoint.

6. Data distribution in training data.



7. Data distribution in testing data.



Procedure:

1. Read testing and training data.
2. Since data(x) and class are in same file, slice them into x and y
3. Load labels meta and make dictionary to map it with actual label
4. Preprocessing
    - Apply black and white filter
    - Since the images in dataset are in -90deg orientation, rotate it by 90 d
    - Normalize image by using (0,1) technique
    - Reshape image from 28x28 to 28x28x1 to explicitly mention dimension
    - Convert y to categorical
5. Build CNN
    - The CNN Architecture is as following
        - ★ Convolution layer (64 filter of 3x3 size, relu).
        - ★ Convolution layer (32 filter of 3x3 size, relu).
        - ★ Maxpooling.

        - ★ Convolution layer (32 filter of 3x3 size, relu).
        - ★ Convolution layer (16 filter of 3x3 size, relu) .
        - ★ Flatten generated output for dense layer.
        - ★ Dense layer (128 neurons ,relu).
        - ★ Dense layer for classification (26, softmax) .
    - Optimizer: adam
    - Loss: categorical_crossentropy
    - Batch size: 128
    - Epochs: 10
    - Validation split: 20%

6. Use testing data to calculate accuracy.

Implementation:

```
# 1 loading data

train_data = pd.read_csv('data/emnist-letters-train.csv',header=None)
test_data = pd.read_csv('data/emnist-letters-test.csv',header=None)

# 2 split

x = np.array(train_data.iloc[:,1:].values)
y = np.array(train_data.iloc[:,0].values)
y = y-1

# 3 load labels

label_data = pd.read_csv("data/emnist-balancedmapping.txt",delimiter = ' ',header=None)
labels_dict = dict()
for i in range(label_data.shape[0]):
    labels_dict[i] = label_data.loc[i][1]
labels_dict

# 4 preprocessing

 def rotate(image):
      image = image.reshape(28, 28)
     image = np.fliplr(image)
    image = np.rot90(image)
   return image

 x = np.apply_along_axis(rotate, 1, x)
 x = x/255.0
 x = x.reshape(-1,28,28,1)
 y = to_categorical(y, 26)
```

```
# 5 cnn

model=Sequential()
model.add(Conv2D(64,(3,3),input_shape=(28,28,1),activation='relu'))
model.add(Conv2D(32,(3,3),activation='relu')) model.add(MaxPool2D())
model.add(Conv2D(32,(3,3),activation='relu')) model.add(Conv2D(16,(3,3),activation='relu'))
model.add(Flatten())
model.add(Dense(128,activation='relu'))
model.add(Dense(26,activation='softmax'))
model.summary()
model.compile(optimizer='adam',metrics=['accuracy'],loss='categorical_crossentrop y')
model_cnn = model.fit(x,y,batch_size=128,epochs=10,validation_split=0.2)

# 6 Testing

x_test = np.array(train_data.iloc[:,1:].values)
y_test = np.array(train_data.iloc[:,0].values)
y_test = y_test-1
x_test = np.apply_along_axis(rotate, 1, x_test)
x_test = x_test/255.0
x_test = x_test.reshape(-1,28,28,1)
preditctions = cnn.predict(x_test)
result = np.argmax(preditctions,axis=1)
```

Output:

Training metrices:

- loss:0.0944
- accuracy: 0.9620
- validation loss: 0.2707
- validation accuracy: 0.9278

Testing accuracy: 0.9584

Practical-5

Objective: Implement Text classification using Bayesian rule.

Dataset:

1.Dataset contains information about the news articles classified into four classes i.e., World, Sports, Sci/Tech, Business.

2.It contains 200 articles with each of the category having 50 articles on an average

3. Below are the 10 random sets from this training set:

```
3 ---- Rescuing an Old Saver
2 ---- USC Begins Season Where It Ended, at No. 1 (AP)
1 ---- Najaf battle a crucial test for Allawi
4 ---- Southeast Coast Sees Fewer Turtle Nests (AP)
4 ---- Apple Ships Motion
3 ---- Chad seeks refugee aid from IMF
1 ---- Eye on Athens, China stresses a 'frugal' 2008 Olympics
4 ---- FCC mobile spam rule doesn't cover some SMS
4 ---- Google, Yahoo Settle Patent and Share Disputes
2 ---- USC starts at the top
```

4.Testing query:

[India vs Pakistan ODI match,India wins **;**Google launches new phone with jio]

Procedure:

1.Load the dataset train.csv and convert it into a list containing list of two elements, class name and article heading.

2.Do the pre-processing of the text present in article heading in following manner:

- Remove all the numeric, alpha-numeric and special characters from the text
- Converts the strings into tokens
- Remove all the stop words
- Apply stemming for getting the root word for all the text.

3.Combine all the article text and find unique words from it.

4.Create different space for each given class and count occurrence of each character in each class.

5.Calculate prior probabilities for each word in each space that is created. Formula for prior probability is:

P(wk/classi) = (nk + 1)/(n + |vocabulary|)

Where wk is the current word, nk is the number of times the word appears for class i, n is the total number of words in class i and |vocabulary| is the number of unique words in text corpus.

6.For given test data, multiply all the probabilities of all the words present in all elements of test data. Do this for each class

7.Find the maximum probability and the class that gives it. This is the required prediction

Code:

```
import nltk
nltk.download('all')

from nltk import word_tokenize

from csv import reader
import numpy as np
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
import re

tokens = []
class_article = []
stop_words = set(stopwords.words('english'))
ps = PorterStemmer()


with open('/content/train.csv', 'r') as read_obj:
 csv_reader = reader(read_obj)
 for row in csv_reader:
 article_description = row[1]
 class_article.append([row[0],article_description])
 cleaned_text = re.sub('[^A-Za-z0-9]+|[0-9]', " ", article_description)
 token = [ps.stem(w) for w in word_tokenize(cleaned_text)]
```

```
 tokens = np.append(tokens, token)
unique_tokens = np.unique(tokens)


filtered_sentence = [w.lower() for w in unique_tokens if not w.lower() in stop_words]


import random
random.shuffle(class_article)
for ca in class_article[:10]:
 print(ca[0], "----", ca[1])

print(class_article[0])
print(filtered_sentence)
print(len(filtered_sentence))


freq_dict_1 = {}
freq_dict_2 = {}
freq_dict_3 = {}
freq_dict_4 = {}


for w in filtered_sentence:
 freq_dict_1[w] = 0 if w not in freq_dict_1 else freq_dict_1[w] + 1
 freq_dict_2[w] = 0 if w not in freq_dict_2 else freq_dict_2[w] + 1
 freq_dict_3[w] = 0 if w not in freq_dict_3 else freq_dict_3[w] + 1
 freq_dict_4[w] = 0 if w not in freq_dict_4 else freq_dict_4[w] + 1

for w in filtered_sentence:
 for item in class_article:
 if item[0] == '1' and w.lower() in item[1].lower():
 freq_dict_1[w] = freq_dict_1[w] + 1
 if item[0] == '2' and w.lower() in item[1].lower():
 freq_dict_2[w] = freq_dict_2[w] + 1
 if item[0] == '3' and w.lower() in item[1].lower():
 freq_dict_3[w] = freq_dict_3[w] + 1
 if item[0] == '4' and w.lower() in item[1].lower():
 freq_dict_4[w] = freq_dict_4[w] + 1


print(freq_dict_1)
print(freq_dict_2)
print(freq_dict_3)
print(freq_dict_4)


n_1 = sum(freq_dict_1.values())
```

```
n_2 = sum(freq_dict_2.values())
n_3 = sum(freq_dict_3.values())
n_4 = sum(freq_dict_4.values())
vocab = len(filtered_sentence)


for w in filtered_sentence:
 freq_dict_1[w] = (freq_dict_1[w] + 1)/(n_1 + vocab)
 freq_dict_2[w] = (freq_dict_2[w] + 1)/(n_2 + vocab)
 freq_dict_3[w] = (freq_dict_3[w] + 1)/(n_3 + vocab)
 freq_dict_4[w] = (freq_dict_4[w] + 1)/(n_4 + vocab)


test_data = []
reply = 'y'
while (reply.lower() == 'y'):
 test_string = input("Input article heading: ")
 test_data.append(test_string)
 reply = input("Wish to enter more? (y/n) ")
print(test_data)


def evaluate(test_data):
 result = []
 for test in test_data:
 probabilities = [1,1,1,1]
 test = test.lower()
 if test_word in freq_dict_1:
 probabilities[0] = probabilities[0] * freq_dict_1[test_word]
 if test_word in freq_dict_2:
 probabilities[1] = probabilities[1] * freq_dict_2[test_word]
 if test_word in freq_dict_3:
 probabilities[2] = probabilities[2] * freq_dict_3[test_word]
 if test_word in freq_dict_4:
 probabilities[3] = probabilities[3] * freq_dict_4[test_word]
 max_prob = max(probabilities)
 result.append([test, probabilities.index(max_prob)+1])
 return result


def predict(values):
 for value in values:
 if value[1] == 1:
 print(value[0], " belongs to class World")
 elif value[1] == 2:
 print(value[0], " belongs to class Sports")
 elif value[1] == 3:
 print(value[0], " belongs to class Business")
```

```
elif value[1] == 4:
print(value[0], " belongs to class Sci/Tech")


predict(evaluate(test_data))
```

Output:

India vs Pakistan odi match, India wins **belongs to class Sports**
google launches new phone with jio **belongs to class Sci/Tech**

Practical-6

Objective: Design a classifier with k-fold cross validation technique.

Dataset: some toy dataset with x and y coordinates and with classes 0,1.
sample data points are as follows;

	x	y	class
0	0.700335	-0.247068	0.0
1	-3.950019	2.740080	1.0
2	0.150222	-2.157638	1.0
3	-1.672050	-0.941519	1.0
4	2.560483	-1.846577	1.0

Procedure:

1. Load the dataset.
2. Split the dataset into train and test with as 70% ,30%.
3. Now again split the train data set into two;
   one for training(70% of 70)
   Another for cross validation (30% of 70)

Simple cross Validation:

4. For each odd k from 1 to 30 train the k-nearest neighbbour classifier with train dataset and predict on cross validation dataset.
5. We will get the best classifier amoung all those by seeing the metric accuracy.
6. Now we will test the classfier on test data set by selecting the k with got more accuracy.

K Fold cross validation:

7. Create a list of odd numbers from 1 to 50.
8. For each k in the list apply the k nearest neighbour classifier with 10 folds as cross validation.
9. Calculate the mean cross validation score for each k and store it in a list.
10. Obtain the misclassification error from cross validation score.
11. Pick up the best k with least misclassification error.
12. Test the classifier with that best k on test data set.
13. Plot a graph between the k and the and the misclassification error to analyse.

Code:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
from sklearn.model_selection import cross_val_score
from collections import Counter
from sklearn.metrics import accuracy_score
from sklearn import model_selection



# Part I
# ============================== data preprocessing
==========================================

# define column names
names = ['x', 'y', 'class']

# loading training data
df = pd.read_csv('3.concertriccir2.csv', header=None, names=names)
print(df.head())

# create design matrix X and target vector y
X = np.array(df.iloc[:, 0:2]) # end index is exclusive
y = np.array(df['class']) # showing you two ways of indexing a pandas df




X_1, X_test, y_1, y_test = model_selection.train_test_split(X, y, test_size=0.3, random_state=0)

# split the train data set into cross validation train and cross validation test
X_tr, X_cv, y_tr, y_cv = model_selection.train_test_split(X_1, y_1, test_size=0.3)

for i in range(1,30,2):
    # instantiate learning model (k = 30)
    knn = KNeighborsClassifier(n_neighbors=i)
```

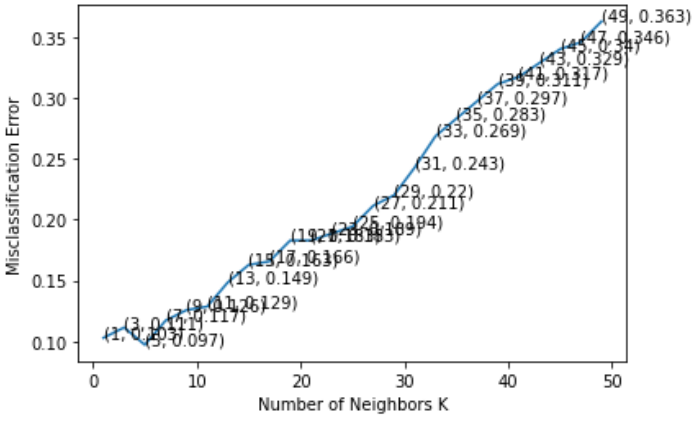
```
    # fitting the model on crossvalidation train
    knn.fit(X_tr, y_tr)

    # predict the response on the crossvalidation train
    pred = knn.predict(X_cv)

    # evaluate CV accuracy
    acc = accuracy_score(y_cv, pred, normalize=True) * float(100)
    print('\nCV accuracy for k = %d is %d%%' % (i, acc))

knn = KNeighborsClassifier(1)
knn.fit(X_tr,y_tr)
pred = knn.predict(X_test)
acc = accuracy_score(y_test, pred, normalize=True) * float(100)
print('\n****Test accuracy for k = 1 is %d%%' % (acc))




# creating odd list of K for KNN
myList = list(range(0,50))
neighbors = list(filter(lambda x: x % 2 != 0, myList))

# empty list that will hold cv scores
cv_scores = []

# perform 10-fold cross validation
for k in neighbors:
    knn = KNeighborsClassifier(n_neighbors=k)
    scores = cross_val_score(knn, X_1, y_1, cv=10, scoring='accuracy')
    cv_scores.append(scores.mean())

# changing to misclassification error
MSE = [1 - x for x in cv_scores]

# determining best k
optimal_k = neighbors[MSE.index(min(MSE))]
print('\nThe optimal number of neighbors is %d.' % optimal_k)

# plot misclassification error vs k
plt.plot(neighbors, MSE)
```

```
for xy in zip(neighbors, np.round(MSE,3)):
    plt.annotate('(%s, %s)' % xy, xy=xy, textcoords='data')

plt.xlabel('Number of Neighbors K')
plt.ylabel('Misclassification Error')
plt.show()

print("the misclassification error for each k value is : ", np.round(MSE,3))
```

Output:

```
CV accuracy for k = 3 is 85%
CV accuracy for k = 5 is 82%
CV accuracy for k = 7 is 80%
CV accuracy for k = 9 is 80%
CV accuracy for k = 11 is 80%
CV accuracy for k = 13 is 79%
CV accuracy for k = 15 is 80%
CV accuracy for k = 17 is 78%
CV accuracy for k = 19 is 79%
CV accuracy for k = 21 is 77%
CV accuracy for k = 23 is 74%
CV accuracy for k = 25 is 74%
CV accuracy for k = 27 is 72%
CV accuracy for k = 29 is 68%
****Test accuracy for k = 1 is 90%
```

```
The optimal number of neighbors is 5.
```



Practical-7

Objective: Design a multivariate linear regression classifier with gradient descent.

The implementation is done by creating 3 modules each used for performing different operations in the Training Process.

Linear regression:

It is the principal function that takes the features matrix (X), Target Variable Vector (y), learning rate (alpha) and number of iterations (num_iters) as input and outputs the final optimized theta i.e., the values of [theta_0, theta_1, theta_2, theta_3,….,theta_n] for which the cost function almost achieves minima following Batch Gradient Descent, and cost which stores the value of cost for every iteration.

Hypothesis:

It is the function that calculates and outputs the hypothesis value of the Target Variable, given theta (theta_0, theta_1, theta_2, theta_3, …., theta_n), Features in a matrix, X of dimension [m X (n+1)] where m is the number of samples and n is the number of features.

Batch Gradient descent:

It is the function that performs the Batch Gradient Descent Algorithm taking current value of theta (theta_0, theta_1,…, theta_n), learning rate (alpha), number of iterations (num_iters), list of hypothesis values of all samples (h), feature set (X), Target Variable set (y) and Number of Features (n) as input and outputs the optimized theta (theta_0, theta_1, theta_2, theta_3, …, theta_n) and the cost history or cost which contains the value of the cost function over all the iterations.

Problem Statement:

“Given the size of the house and number of bedrooms, analyze and predict the possible price of the house”.

Dataset:

Dataset consists of size of the house , number of bedrooms, price of the house.

Procedure:

1.Load the dataset.

2.Calculate the mean and standard deviation of each attribute of the data set.

3.Normalize each data point as follows,

x=(x-u)/Σ.

4.obtain theta and cost from linear regression function with some learning rate.

5.The cost has been reduced in the course of Batch Gradient Descent function iteration-by-iteration. The reduction in the cost is shown with the help of Line Curve.

6. Plot Side-by-Side Visualization of Features and Target Variable Actual and Prediction using 3-D Scatter Plots

7.That's all about the Implementation of Multi-Variate Linear Regression in Python using Gradient Descent from scratch.

Code:

```
def hypothesis(theta, X, n):
    h = np.ones((X.shape[0],1))
    theta = theta.reshape(1,n+1)
    for i in range(0,X.shape[0]):
        h[i] = float(np.matmul(theta, X[i]))
    h = h.reshape(X.shape[0])
    return h

def BGD(theta, alpha, num_iters, h, X, y, n):
    cost = np.ones(num_iters)
    for i in range(0,num_iters):
        theta[0] = theta[0] - (alpha/X.shape[0]) * sum(h - y)
        for j in range(1,n+1):
```

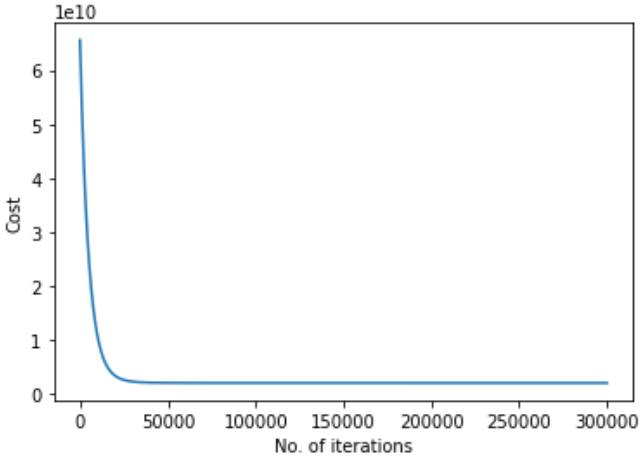
```
            theta[j] = theta[j] - (alpha/X.shape[0]) * sum((h-y) * X.transpose()[j])
        h = hypothesis(theta, X, n)
        cost[i] = (1/X.shape[0]) * 0.5 * sum(np.square(h - y))
    theta = theta.reshape(1,n+1)
    return theta, cost

def linear_regression(X, y, alpha, num_iters):
    n = X.shape[1]
    one_column = np.ones((X.shape[0],1))
    X = np.concatenate((one_column, X), axis = 1)
    # initializing the parameter vector...
    theta = np.zeros(n+1)
    # hypothesis calculation....
    h = hypothesis(theta, X, n)
    # returning the optimized parameters by Gradient Descent...
    theta, cost = BGD(theta,alpha,num_iters,h,X,y,n)
    return theta, cost

data = np.loadtxt('data2.txt', delimiter=',')
X_train = data[:,[0,1]] #feature set
y_train = data[:,2] #label set

mean = np.ones(X_train.shape[1])
std = np.ones(X_train.shape[1])
for i in range(0, X_train.shape[1]):
    mean[i] = np.mean(X_train.transpose()[i])
    std[i] = np.std(X_train.transpose()[i])
    for j in range(0, X_train.shape[0]):
        X_train[j][i] = (X_train[j][i] - mean[i])/std[i]

# calling the principal function with learning_rate = 0.0001 and
```

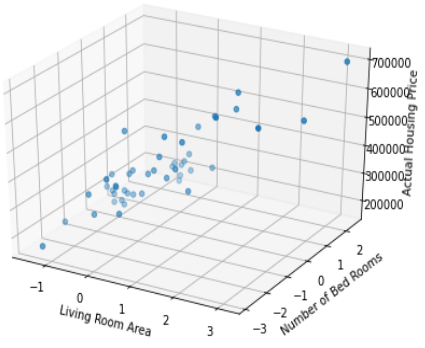
```
# num_iters = 300000
theta, cost = linear_regression(X_train, y_train,0.0001, 300000)

import matplotlib.pyplot as plt
cost = list(cost)
n_iterations = [x for x in range(1,300001)]
plt.plot(n_iterations, cost)
plt.xlabel('No. of iterations')
plt.ylabel('Cost')

from matplotlib import pyplot
from mpl_toolkits.mplot3d import Axes3D
sequence_containing_x_vals = list(X_train.transpose()[0])
sequence_containing_y_vals = list(X_train.transpose()[1])
sequence_containing_z_vals = list(y_train)
fig = pyplot.figure()
ax = Axes3D(fig)
ax.scatter(sequence_containing_x_vals, sequence_containing_y_vals,
 sequence_containing_z_vals)
ax.set_xlabel('Living Room Area', fontsize=10)
ax.set_ylabel('Number of Bed Rooms', fontsize=10)
ax.set_zlabel('Actual Housing Price', fontsize=10)

X_train = np.concatenate((np.ones((X_train.shape[0],1)), X_train),axis = 1)
predictions = hypothesis(theta, X_train, X_train.shape[1] - 1)
from matplotlib import pyplot
from mpl_toolkits.mplot3d import Axes3D
sequence_containing_x_vals = list(X_train.transpose()[1])
sequence_containing_y_vals = list(X_train.transpose()[2])
sequence_containing_z_vals = list(predictions)
fig = pyplot.figure()
```

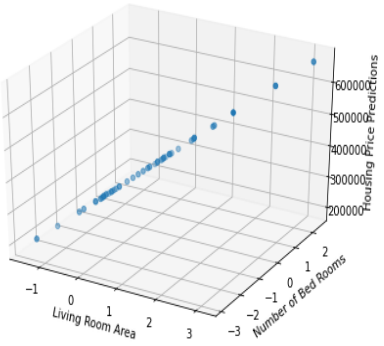
```
ax = Axes3D(fig)
ax.scatter(sequence_containing_x_vals, sequence_containing_y_vals,
           sequence_containing_z_vals)
ax.set_xlabel('Living Room Area', fontsize=10)
ax.set_ylabel('Number of Bed Rooms', fontsize=10)
ax.set_zlabel('Housing Price Predictions', fontsize=10)
```

output:

```
Text(0, 0.5, 'Cost')
```







Practical-8

Objective:

Design a logistic regression classifier with gradient descent.

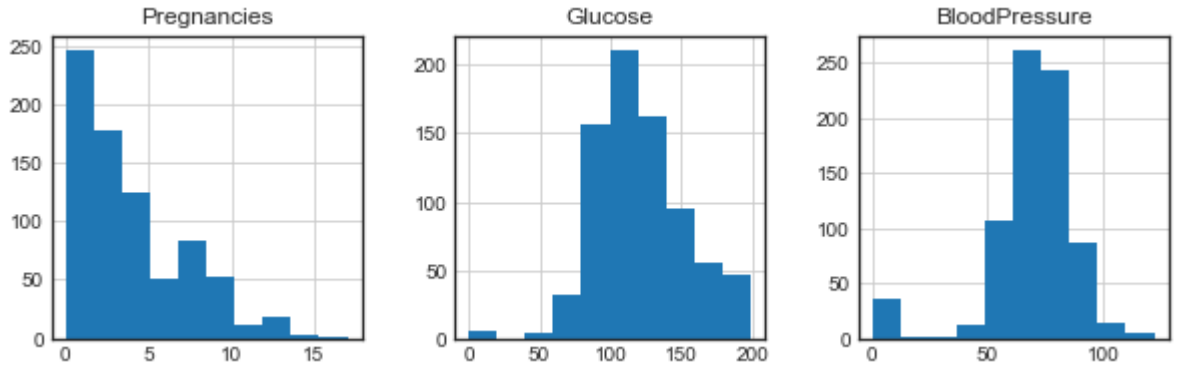
Description:

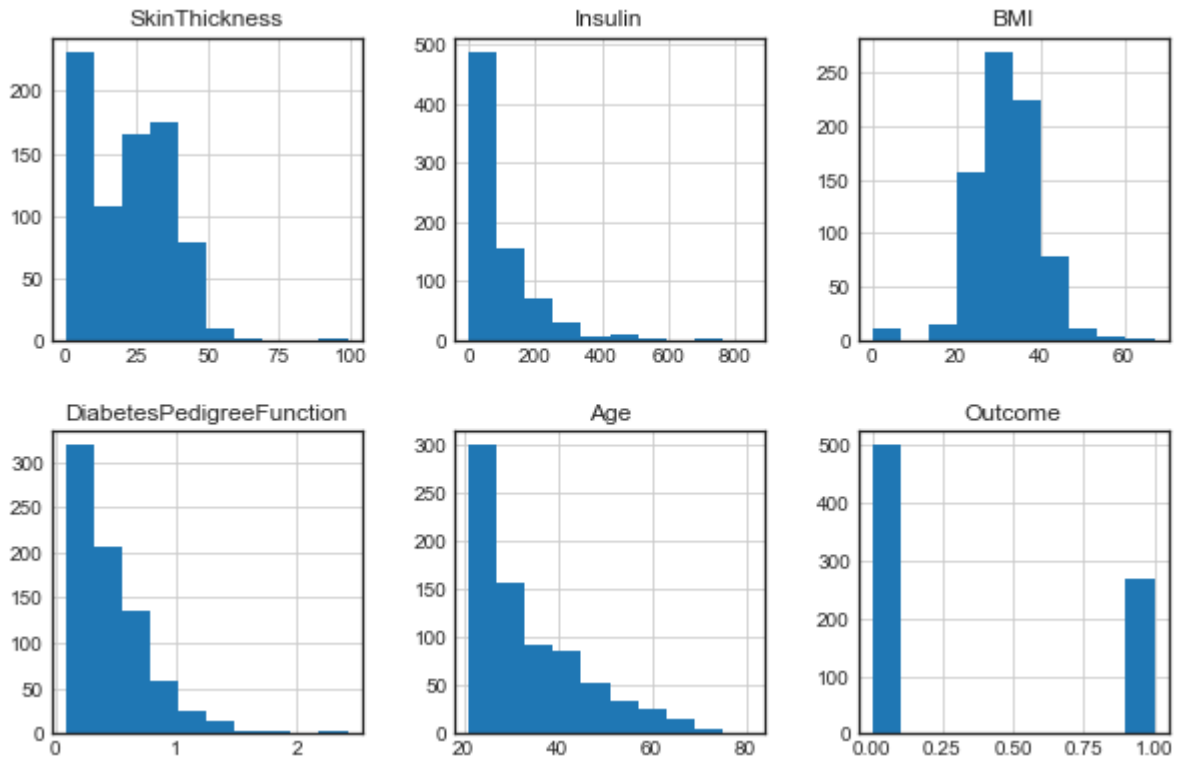
Given a data set with multiple features (can have multiple values) and a class(discrete values), Design a logistic regression classifier with gradient descent which will give out target value x for unknown examples.

Dataset:

1. Name: Diabetics prediction
2. Source: https://www.kaggle.com/kandij/diabetes-dataset
3. The data was collected and made available by “National Institute of Diabetes and Digestive and Kidney Diseases” as part of the Pima Indians Diabetes Database
4. There are 8 features and 1 class(Outcome).
5. There are 768 samples
6. Correlation with Outcome :

Pregnancies	0.221898
Glucose	0.466581
BloodPressure	0.065068
SkinThickness	0.074752
Insulin	0.130548
BMI	0.292695
DiabetesPedigreeFunction	0.173844
Age	0.238356
Outcome	1.000000





Algorithm:

1. Read data from csv using pandas
2. Covert dataset into list of x(data) and y(target)
3. Normalize x by using (0,1) normalization
4. Split the data into training and test data
5. Logistic Regression:
    - Initialize theta vector(size = number of features + 1) to zero
    - Calculate y_cap(ie: predictions) by performing dot product of x and theta and applying sigmoid function for the obtained value.
    - Calculate the cost function by using below formula:

      cost= - 1 /m * [y log(y pred) + (1-y) log(1-y pred)

      m = number of samples
      since we are doing multivariate here we will sum up error vector to get resultant error.

    - Update the value of theta by using following formula:

      θ = θ-(η /m) * (xt.error)

      error= (y - ypred)

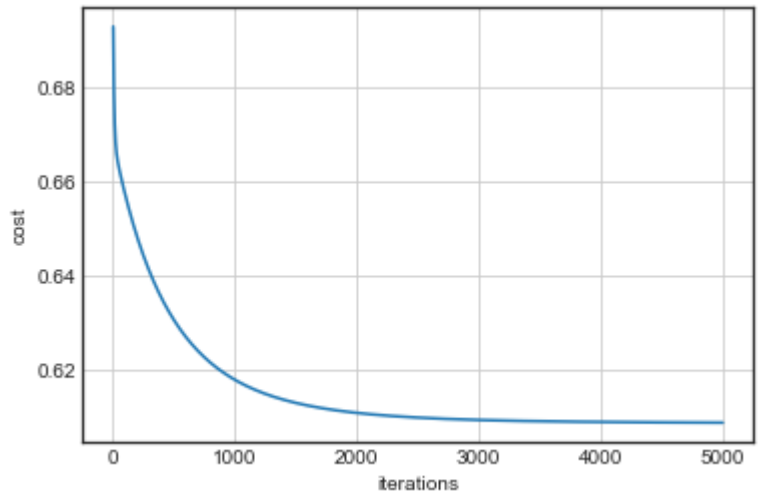
- Repeat (c-f) for specified amount of iterations

Implementation:

```
# 1 read data

data = pd.read_csv('./data/ diabetes2.csv')

# 2 split data into x and y

x = np.array(data.iloc[:,:-1].values)
y = np.array(data['Outcome'])

# 3 Normalize the data

def normalize(x):
 return x/np.max(x, axis=0),np.max(x, axis=0)
x,temp_max = normalize(x)

# 4 split data into test and training

from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.30)
n=len(x_train[0])-1
m=len(x_train)

# 5 Logistic regression

def sigmoid(x):
 return 1/(1+np.e**(-x))

def logistic_regression(x, y, theta, lr, itr):
 cost_plot = np.zeros(itr)
 for i in range(itr):
 y_cap = sigmoid(np.dot(x, theta))
 cost = y * np.log(y_cap) + (1 - y)*np.log(1 - y_cap)
 cost = -1*(np.sum(cost) / m)
 cost_plot[i] = cost
 updated_theta = lr * np.dot(x.T,(y_cap - y)) / m
 theta = theta - updated_theta
 plt.plot(range(1, itr + 1), cost_plot)
 plt.grid()
 plt.xlabel("iterations")
 plt.ylabel("cost")
 return theta
```

```
theta = np.zeros(n+1)
itr = 5000
theta = logistic_regression(x, y, theta, 0.2 ,itr)
y_pred = sigmoid(np.dot(x, theta))

count=0

for i in range(len(y)):
 if(y_pred[i]>0.5 and y[i] == 1):
 count+=1
 continue
 if(y_pred[i]<=0.5 and y[i] == 0):
 count+=1
print(count/len(y))
```

Output:



Accuracy : 69.140625%

Theta values:

array([2.07826746, 2.43421778, -3.62649312, -0.02572106, 0.7349827 ,
-0.33453605, 0.7574637 , -1.07804988])

Practical-9

Objective:

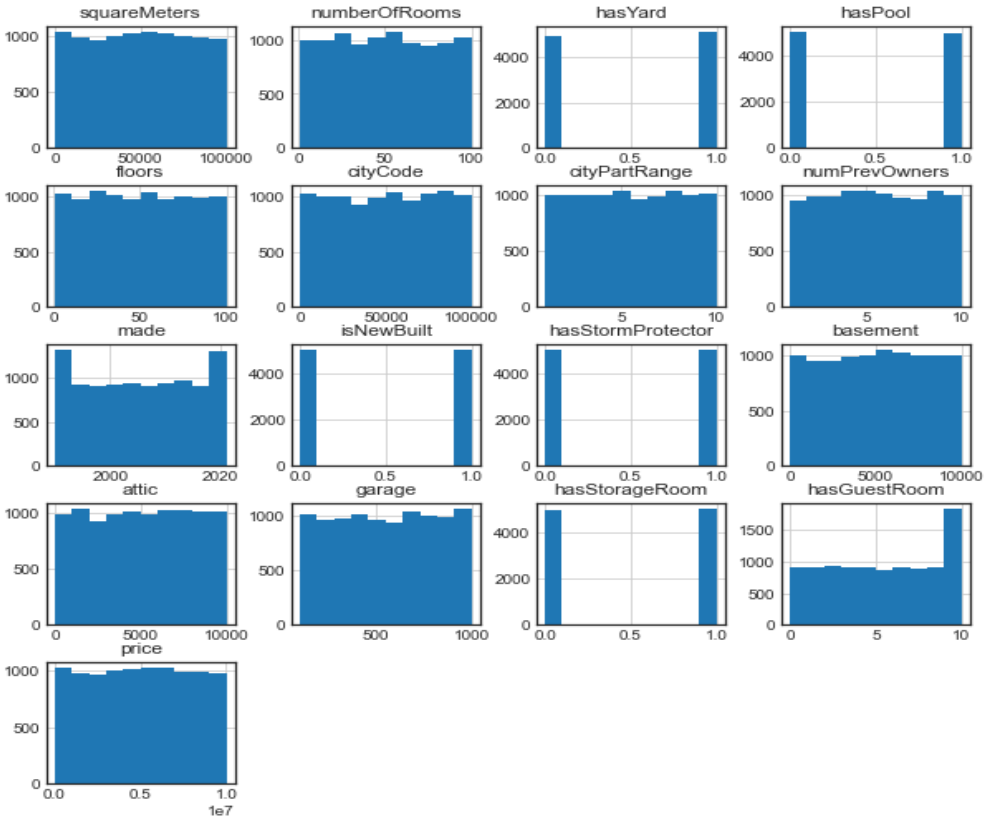
_____Design a classifier with regularization technique used in it.

Description:

Given a data set with multiple features (can have multiple values) and a class(continuous values), Design a linear regression classifier with gradient descent along with regularization technique used in it which will give out target value x for unknown examples.

Dataset:

1. Name: Paris Housing Price Prediction
2. Source: https://www.kaggle.com/mssmartypants/paris-housing-priceprediction
3. This is a set of data created from imaginary data of house prices in an urban environment – Paris
4. There are 16 variables(house features) and 1 class(price).
5. There are 10000 samples.



6. Correlation with price:

squareMeters	0.999999
numberOfRooms	0.009591
hasYard	-0.006119
hasPool	-0.005070
floors	0.001654
cityCode	-0.001539
cityPartRange	0.008813
numPrevOwners	0.016619
made	-0.007210
isNewBuilt	-0.010643
hasStormProtector	0.007496
Basement	-0.003967
attic	-0.000600
Garage	-0.017229
hasStorageRoom	-0.003485
hasGuestRoom	-0.000644
price	1.000000

Algorithm:

1. Read data from csv using pandas
2. Covert dataset into list of x(data) and y(target)
3. Normalize x by using (0,1) normalization
4. Split the data into training and test data
5. Linear Regression with regularization:
    - Insert 1 as first data to all the data since the value of x0 is 1.(theta0 is intercept)
    - Initialize theta vector(size = number of features + 1) to zero
    - Calculate y_cap(ie: predictions) by performing dot product of x and theta.
    - Calculate error by subtracting y_cap from y
    - Calculate the cost function by using below formula:
      cost= 1/2m *(error)^2
      m= no of samples
    - Update the value of theta by using following formula:
      θ = θ-(η/m) * (xt . error)
      θ = θ -(η/m) * (xt . error) + λ/m * θ

- Repeat (c-f) for specified amount of iterations.

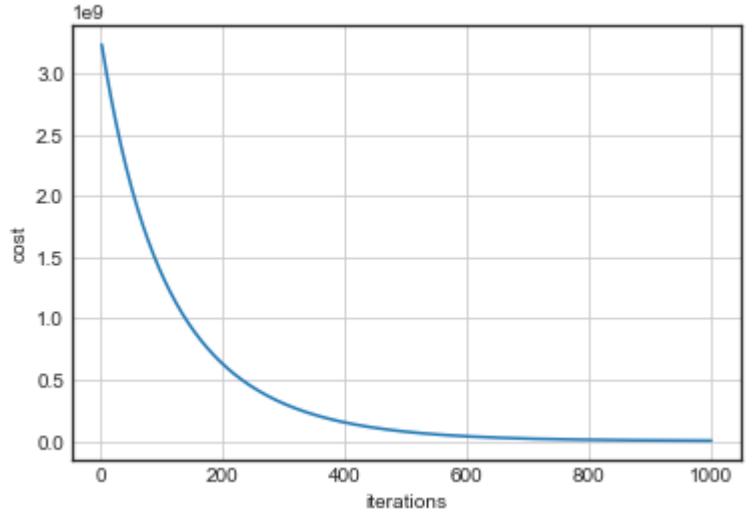
Implementation:

```
# 1 read data
data = pd.read_csv('./data/ParisHousing.csv')

# 2 split data into x and y
x = np.array(data.iloc[:,:-1].values)

# 5a append 1 for x0 value
x = np.hstack((np.ones((data.shape[0],1)), x))
y = np.array(data['price'])

# 3 Normalize the data
def normalize(x):
 return x/np.max(x, axis=0),np.max(x, axis=0)
x,temp_max = normalize(x)

# 4 split data into test and training
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.30)
n=len(x_train[0])-1
m=len(x_train)

# 5 Linear Regression with regularization

def gd_r(x, y, theta, lr, itr , lambda_value):
 cost_plot = np.zeros(itr)
 theta_0 = theta[0]
 theta_1_n = theta[1:]
 for i in range(itr):
 y_cap = x.dot(theta)
 e = y_cap - y
 cost = 1 / (2 * m) * np.sum(e*e)
 cost_plot[i] = cost
 updated_theta_0 = (lr / m) * (x[:,0].dot(e))
 updated_theta_1_n = (lr / m) * ( (((x[:,1:]).transpose()).dot(e)) +
((lambda_value/m) * theta_1_n) )
 theta = theta - np.hstack((updated_theta_0, updated_theta_1_n))
 plt.plot(range(1, itr + 1), cost_plot)
 plt.grid()
 plt.xlabel("iterations")
 plt.ylabel("cost")
 return theta

theta = np.zeros(n+1)
itr = 500;
```

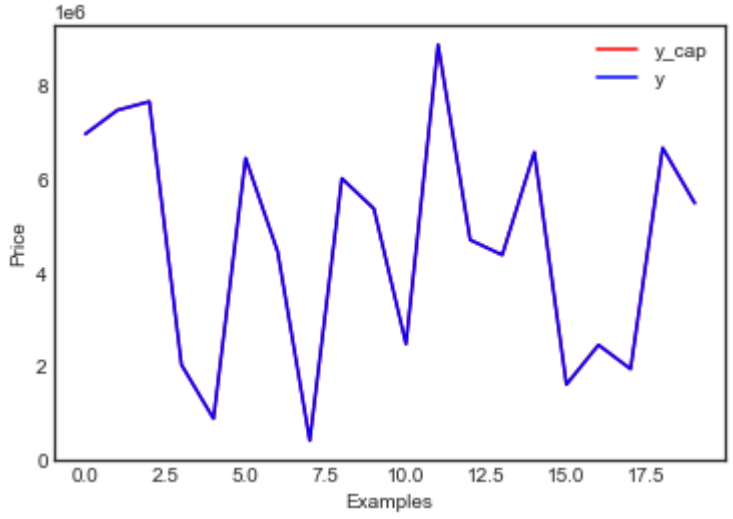
```
theta = gd_r(x_train, y_train, theta, 0.1, 1000,0.2)
y_pred = list()
for i in range(len(x_test)):
 y_pred.append(theta.dot(x_test[i]))
```

Output:

GD graph:



Prediction vs actual graph:



Theta values:

array([6.89661881e+03, 9.99746216e+06, -2.04197309e+03, 2.50151252e+03,
2.50199203e+03, 3.25549527e+03, -2.27395285e+03, -2.05252177e+03,
-2.31281380e+03, 6.16421515e+03, -3.49527348e+02, -3.52515333e+02,
-2.61626756e+03, -3.39951180e+03, -3.12073992e+02, -2.03773057e+03])

Practical-10

Objective

Implement weighted K- Nearest neighbors model.

Dataset -

- For this program an Iris flower dataset is used.
- This data set is a multivariate data set introduced by the British statistician and biologist Ronald Fisher in his 1936 paper The use of multiple measurements in taxonomic problems.
- The dataset contains a set of 150 records under 5 attributes - Petal Length, Petal Width, Sepal Length, Sepal width and Class(Species).
- The dataset contains 5 columns and the target field is *'Species'.*
- Income column is divided into three classes: *'Iris Setosa'* , *'Iris virginica'* and *'Iris versicolor '.*

Species	No of values
Iris Setosa	50
Iris virginica	50
Iris versicolor	50

Model structure -

- Weighted K-nearest neighbors (KNN) algorithm is a KNN algorithm in which the weights of each of the nearest neighbours is made proportional to the distance from x, the closer the neighbour the greater its impact.
- A weight is assigned to each neighbour with the help of some mathematical concepts.
- The one used here for this program is given below -

$$Wi = \begin{cases} (d_k - d_i)/(d_k - d_1) & d_k \neq d_1 \\ 1 & d_k = d_1 \end{cases}$$

Where -
d1, d2 , d3…….dk are distances from x so that d1 is smallest and dk is greatest distance.

Implementation & Output:

```
# Mounting google colab and project path
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
# import libraries
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from collections import Counter
```

```
# read and store data
data = pd.read_csv(r"/content/drive/MyDrive/Dataset/Iris.csv")
data
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa
...	...	...	...	...	...	...
145	146	6.7	3.0	5.2	2.3	Iris-virginica
146	147	6.3	2.5	5.0	1.9	Iris-virginica
147	148	6.5	3.0	5.2	2.0	Iris-virginica
148	149	6.2	3.4	5.4	2.3	Iris-virginica
149	150	5.9	3.0	5.1	1.8	Iris-virginica

150 rows × 6 columns

```
#convert string column of dataset to int column
classes = ['Iris-setosa', 'Iris-versicolor', 'Iris-virginica']
for i in range(len(classes)):
  data.loc[(data.Species == classes[i]), 'Species'] = i
data.head()
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	0
1	2	4.9	3.0	1.4	0.2	0
2	3	4.7	3.2	1.3	0.2	0
3	4	4.6	3.1	1.5	0.2	0
4	5	5.0	3.6	1.4	0.2	0

```
[ ] #split the data into X and Y
    X = data.drop('Species', axis=1)
    Y = data['Species']
    Y = Y.to_numpy()
```

```
[ ] X = X.to_numpy(dtype='float')
```

```
[ ] # split data into training and testing
    X_train, X_test, Y_tarin, Y_test = train_test_split(X, Y, test_size = 0.3)
```

```
    # function to return class index based on maximum weights
    def getClassIdx(a, b, c):
      if a>b :
        if a>c:
          return 0
        else:
          return 2
      else:
        if b > c:
          return 1
        else :
          return 2
```

```
[ ] # KNN model class
    class KNN:
      def __init__(self, k =3):
        self.k = k

      def fit(self, x, y):
        self.X_train = x
        self.Y_train = y

      def predict(self, X):
        predicted_values = [self._predict(x) for x in X]
        return np.array(predicted_values)

      def _predict(self, x):
        # distances
        distnaces = []
        for i in range(len(X_train)):
          dist = np.sqrt(sum(np.square(X_train[i] - x)))
          distnaces.append([i, dist])
        # get k nearest data
        k_indices = sorted(distnaces, key= lambda a : a[1])[: self.k]
        # get weights for K nearest data
        d1 = k_indices[0][1]
        dk = k_indices[self.k - 1][1]
        weights = []
```

```
[ ]     for i in range(len(k_indices)):
          wt = (dk - k_indices[i][1] )/ (dk - d1)
          weights.append([k_indices[i][0], wt])
        k_nearest_labels = [self.Y_train[i[0]] for i in weights]
        c1 , c2, c3 = 0 , 0 , 0
        #calculate majority vote and common class label
        for i in range(len(k_nearest_labels)):
          if k_nearest_labels[i] == 0:
            c1+=weights[i][1]
          if k_nearest_labels[i] == 1:
            c2+=weights[i][1]
          else:
            c3+=weights[i][1]
        return getClassIdx(c1, c2, c3)
```

```
[ ] # fit the model
    classifier = KNN(k= 5)
    classifier.fit(X_train, Y_tarin)
    predictions = classifier.predict(X_test)
    accuracy = np.sum(predictions == Y_test) / len(Y_test)
    accuracy = round((accuracy * 100), 2)
    print('Classifier Accuracy : ',accuracy , '%')
```

```
Classifier Accuracy :  77.78 %
```

```
[ ] # new data defined by user
    x = [[0.22, 0.56,0.3,0.25,0.8]]
    y = classifier.predict(x)
    print('Predicted Class : ', classes[y[0]])
```

```
Predicted Class :  Iris-virginica
```

Practical-11

Objective:

Implement k Means for Machine Learning Application.

Description:

- We are given a data set of items, with certain features, and values for these features (like a vector). The task is to categorise those items into groups. To achieve this, we will use the kMeans algorithm, an unsupervised learning algorithm
- The algorithm works as follows:
    1. First we initialise k points, called means, randomly.
    2. We categorise each item to its closest mean and we update the mean's coordinates, which are the averages of the items categorised in that mean so far
    3. We repeat the process for a given number of iterations and at the end, we have our clusters
    4. K means is one of the most popular Unsupervised Machine Learning Algorithms Used for Solving Classification Problems. K Means segregates the unlabelled data into various groups, called clusters, based on having similar features, common patterns.

Steps:

The working of the K-Means algorithm is explained in the below steps:

1. Select the value of K, to decide the number of clusters to be formed.
2. Select random K points which will act as centroids.
3. Assign each data point, based on their distance from the randomly selected points (Centroid), to the nearest/closest centroid which will form the predefined clusters.
4. Place a new centroid of each cluster. - Repeat step no.3, which reassign each
5. If any reassignment occurs, then go to step-4 else go to Step 7.
6. FINISH

Implementation & Output:

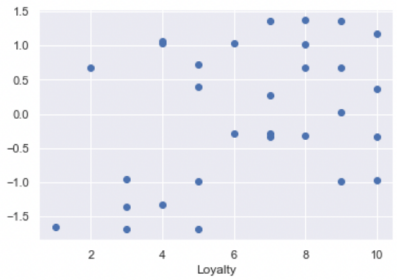
In [1]:
```
import numpy as np
import pandas as pd
import statsmodels.api as sm
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()
from sklearn.cluster import KMeans
```

In [2]:
```
data = pd.read_csv('KMeansDataset.csv')
data.head()
```

Out[2]:

	Satisfaction	Loyalty
0	4	-1.33
1	6	-0.28
2	5	-0.99
3	7	-0.29
4	4	1.06

In [3]:
```
plt.scatter(data['Satisfaction'], data['Loyalty'])
plt.xlabel('Satisfaction')
plt.xlabel('Loyalty')
plt.show()
```



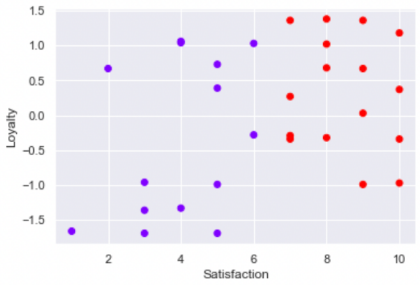
In [4]:
```
#Selecting Features
x=data.copy()
```

In [5]:
```
#Clustering
kmeans=KMeans(2)
kmeans.fit(x)
```

Out[5]: KMeans(n_clusters=2)

In [6]:
```
#Clustering Result
clusters=x.copy()
clusters['cluster_pred']=kmeans.fit_predict(x)
```

In [7]:
```
#Plot
plt.scatter(clusters['Satisfaction'], clusters['Loyalty'],c=clusters['cluster_pred'],cmap='rainbow')
plt.xlabel('Satisfaction')
plt.ylabel('Loyalty')
plt.show()
```



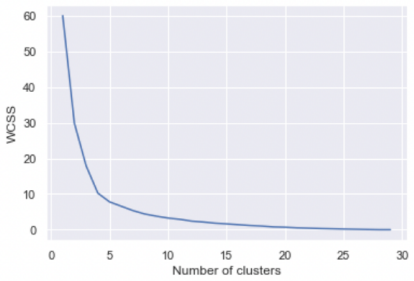
In [8]:
```
#Standardizing the variable
from sklearn import preprocessing
x_scaled=preprocessing.scale(x)
x_scaled
```

Out[8]:
```
array([[-0.93138063, -1.3318111 ],
       [-0.15523011, -0.28117124],
       [-0.54330537, -0.99160391],
       [ 0.23284516, -0.29117733],
       [-0.93138063,  1.05964534],
       [-2.09560642, -1.6620122 ],
       [ 1.39707095, -0.97159172],
       [ 0.62092042, -0.32119561],
       [ 0.62092042,  1.01962097],
       [ 0.62092042,  0.67941378],
       [ 1.39707095, -0.3412078 ],
       [-0.54330537,  0.38923705],
       [-0.54330537, -1.69203048],
       [-1.70753116,  0.66940768],
       [ 0.23284516,  0.26916393],
       [ 1.00899568,  1.35982816],
       [ 0.62092042,  1.37984035],
       [ 0.23284516,  1.35982816],
       [ 0.23284516, -0.3412078 ],
       [ 1.00899568,  0.66940768],
       [ 1.39707095,  1.17971847],
       [-1.31945589, -1.69203048],
       [-0.93138063,  1.03963316],
       [-1.31945589, -0.96158562],
       [-0.15523011,  1.02962706],
       [ 1.00899568, -0.99160391],
       [ 1.39707095,  0.36922486],
       [ 1.00899568,  0.02901767],
       [-1.31945589, -1.36182938],
       [-0.54330537,  0.72944425]])
```

In [9]:
```
#Elbow Method
wcss=[]
for i in range(1,30):
    kmeans=KMeans(i)
    kmeans.fit(x_scaled)
    wcss.append(kmeans.inertia_)

wcss
```

Out[9]:
```
[59.999999999999986,
 29.818973034723143,
 17.913349527387968,
 10.247181805928422,
 7.792695153937187,
 6.569489487091783,
 5.348079410290981,
 4.395247193896115,
 3.7799886162052667,
 3.2503144612222012,
 2.9080369240790245,
 2.3969501211705038,
 2.145058238505448,
 1.8156574192323445,
 1.6198133783661601,
 1.395897265180343,
 1.1645532493533495,
 1.0151995950549708,
 0.7689799439090226,
 0.6764410827034856,
 0.5146512302075544,
 0.42313027513905704,
 0.32271198172750115,
 0.2472105330779867,
 0.17170908442847233,
 0.11383861748989679,
 0.0559681505513213,
 0.0014517677692203244,
 0.00020024383023728806]
```

In [10]:
```
#Visualizing the Elbow Method
plt.plot(range(1,30),wcss)
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()
```

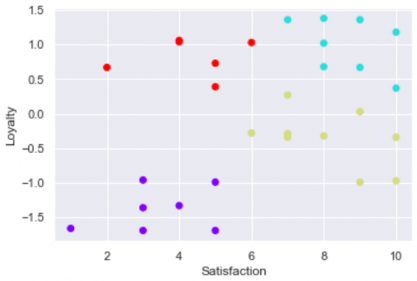


In [14]:
```
#Strong clutering
kmeans_new=KMeans(4)
kmeans.fit(x_scaled)
cluster_new=x.copy()
cluster_new['cluster_pred']=kmeans_new.fit_predict(x_scaled)
cluster_new
```

Out[14]:

	Satisfaction	Loyalty	cluster_pred
0	4	-1.33	2
1	6	-0.28	0
2	5	-0.99	2
3	7	-0.29	0
4	4	1.06	3
5	1	-1.66	2
6	10	-0.97	0
7	8	-0.32	0
8	8	1.02	1
9	8	0.68	1
10	10	-0.34	0
11	5	0.39	3
12	5	-1.69	2
13	2	0.67	3
14	7	0.27	0

In [12]:
```
#Plotting the newly cluster
plt.scatter(cluster_new['Satisfaction'],cluster_new['Loyalty'],c=cluster_new['cluster_pred'],cmap='rainbow')
plt.xlabel('Satisfaction')
plt.ylabel('Loyalty')
plt.show()
```



Practical-12

Objective:

By using PCA perform dimensionality reduction .

Description

Given a 2D data set perform dimensionality reduction by using PCA technique and convert it to 1D data points and test the difference.

Dataset:

1. Name: Social_Network_Ads.
2. Source: https://www.kaggle.com/dragonheir/logistic-regression
3. This is a basic dataset designed for learning logistic regression.
4. It shows whether a person(Male or Female) with salary(s) has bought a product or not 5.
5. It has 400 samples.

Procedure:

1. Read data from csv using pandas
2. Consider only female records
3. Drop ID and gender
4. Covert dataset into list of x(data) and y(target)
5. Split the data into training and testing data
6. Perform logistic regression and note down accuracy
7. PCA
    - Compute the mean along feature
    - Compute center matrix by subtracting x by it's feature mean respectively
    - Compute co-variance matrix on center
    - Compute Eigen values and vectors and normalize it(consider largest value for calculating Eigen vector)
    - Perform dot product of transpose(considered Eigen vector) and data samples to get reduced 1D data
8. Perform PCA on x and reshape it
9. Perform logistic regression and note down accuracy

Implementation:

```
# 1 read data
import pandas as pd
data = pd.read_csv('./Social_Network_Ads.csv')
```

```
# 2 Consider only female records
data = data[data['Gender']=='Female']

# 3 Drop gender and ID
data = data.drop(columns=['Gender','User ID'])

# 4 Covert dataset into list of x(data) and y(target)
x = data.iloc[:,:-1].values
y = data.iloc[:,-1].values

# 5 split data into test and training
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.30)

# 6 Logistic reg
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
model.fit(x_train, y_train)
y_pred = model.predict(x_test)
from sklearn.metrics import accuracy_score
print("accuracy before : ", accuracy_score(y_test, y_pred))

# 7 PCA
import numpy as np
def PCA(data):
 mean = np.mean(data.T, axis=1)
 print("MEAN : ")
 print(mean)
 center = data - mean
 co_var = np.cov(center.T)
 print("Co-variance : ")
 print(co_var)
 e_values, e_vectors = np.linalg.eig(co_var)
 print("Eigen values : ")
 print(e_values)
 print("Eigen vectors : ")
 print(e_vectors)
 considered_e_values = np.array(e_vectors[0])
 print("Considered Eigen vector : ")
 print(considered_e_values)
 reduced = list()
 for i in center:
 reduced.append(considered_e_values.T.dot(i))
 return reduced

# 8 Preform PCA on X and reshape it
x = data.iloc[:,:-1].values
x = PCA(x)
```

```
x = np.array(x).reshape(-1, 1)
y = data.iloc[:,-1].values

# 9 Perform logistic regression and note down accuracy
x_train, x_test, y_train, y_test = train_test_split(x,y, test_size = 0.3)
model = LogisticRegression()
model.fit(x_train, y_train)
y_pred = model.predict(x_test)
print("accuracy after : ", accuracy_score(y_test, y_pred))
```

Output:

accuracy before : 0.6129032258064516
accuracy after : 0.8225806451612904

MEAN : [3.84117647e+01 7.17598039e+04]

Co-variance : [[1.17918285e+02 6.78530861e+04] [6.78530861e+04 1.26702084e+09]]

Eigen values : [1.14284531e+02 1.26702084e+09]

Eigen vectors : [[-9.99999999e-01 -5.35532566e-05] [5.35532566e-05 -9.99999999e-01]]

Considered Eigen vector : [-9.99999999e-01 -5.35532566e-05]

Principal component Analysis Assignment

Step 1:

Dataset

feature	x	y
1	7	10
2	10	12
3	13	15
4	5	7
5	11	13.

Step 2: Computation of mean

$$\bar{x} = \frac{7 + 10 + 13 + 5 + 11}{5}$$

$$= 9.2$$

$$\bar{y} = \frac{10 + 12 + 15 + 7 + 13}{5}$$

$$= 11.4$$

Step 3: Computation of Co-variance.

$$Cov(x,x) = \frac{1}{4}\left[(7-9.2)^2 + (10-9.2)^2 + (13-9.2)^2 + (5-9.2)^2 + (11-9.2)^2\right]$$

$$= 10.2$$

$$Cov(y,y) = \frac{1}{4}\left[(10-11.4)^2 + (12-11.4)^2 + (15-11.4)^2 + (7-11.4)^2 + (13-11.4)^2\right]$$

$$= 9.34$$

$$Cov(x,y) = \frac{1}{4}\left[(7-9.2)(10-11.4) + (10-9.2)(12-11.4) + (13-9.2)(15-11.4) + (5-9.2)(7-11.4) + (11-9.2)(13-11.4)\right]$$

$$= 9.65$$

$$Cov(y,x) = Cov(x,y)$$

$$S = \begin{bmatrix} 10.2 & 9.65 \\ 9.63 & 9.34 \end{bmatrix}$$

Step 4: Eigen value

$$|S - \lambda I| = 0$$

$$\left| \begin{bmatrix} 10.2 & 9.65 \\ 9.65 & 9.3 \end{bmatrix} - \begin{bmatrix} \lambda & 0 \\ 0 & \lambda \end{bmatrix} \right| = 0$$

$$\left| \begin{bmatrix} 10.2 - \lambda & 9.65 \\ 9.65 & 9.3 - \lambda \end{bmatrix} \right| = 0$$

$$(10.2 - \lambda)(9.3 - \lambda) - (9.65)^2 = 0$$

$$\lambda = 19.4104 \text{ or } 0.0895$$

Eigen vector

$$(S - \lambda I) u_1 = 0$$

$$\begin{bmatrix} 10.2 - \lambda & 9.65 \\ 9.65 & 9.3 - \lambda \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} = 0$$

$$\begin{bmatrix} (10.2 - \lambda)u_1 + 9.65 u_2 \\ 9.65 u_1 + (9.3 - \lambda)u_2 \end{bmatrix} = 0$$

$(10.2-\lambda)u_1 + 9.65\,u_2 = 0$

$9.65\,u_1 + (9.3-\lambda)u_2 = 0$

$$\frac{u_1}{9.65} = \frac{u_2}{10.2-\lambda} = t$$

When $t=1$,

Eigen vector $= \begin{bmatrix} 9.65 \\ 10.2 - 19.4104 \end{bmatrix}$

$$= \begin{bmatrix} 9.65 \\ -9.2104 \end{bmatrix}$$

Normalized $= \begin{bmatrix} 9.65 / \sqrt{(9.65)^2 + (-9.21)^2} \\ -9.2104 / \sqrt{(9.65)^2 + (9.21)^2} \end{bmatrix}$

$$= \begin{bmatrix} 0.7233 \\ -0.69044 \end{bmatrix}$$

Step 5: Derive new vector

$$eM_1 = e_1^T \times [e_1 - M]$$

$$= [0.7233 \quad -0.69044] \begin{bmatrix} -2.2 \\ -1.4 \end{bmatrix}$$

$$= -0.62$$

$$ex_9 = e_1^T * \begin{bmatrix} 0.8 \\ 0.6 \end{bmatrix} = 0.16$$

$$ex_3 = e_1^T * \begin{bmatrix} 3.8 \\ 3.6 \end{bmatrix} = 0.26$$

$$ex_4 = e_1^T * \begin{bmatrix} -.42 \\ -0.4 \end{bmatrix} = -0.00028$$

$$ex_5 = e_1^T * \begin{bmatrix} 1.9 \\ 1.6 \end{bmatrix} = 0.19739$$