1. **Implement Simple Linear Regression using Head Size as the independent variable and Brain Weight as dependent variable from headbrain.csv file. Also predict the brain weight for a new head size.**

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

plt.rcParams['figure.figsize']=(20.0,10.0)
df=pd.read_csv('headbrain.csv')

df.head()
X=df['Head Size(cm^3)'].values
Y = df['Brain Weight(grams)'].values
mean_x=np.mean(X)
mean_y=np.mean(Y)
n=len(X)
numer=0
denom=0
for i in range(n):
    numer+=(X[i]-mean_x)*(Y[i]-mean_y)
    denom+=(X[i]-mean_x)**2
m=numer/denom
c=mean_y-(m*mean_x)
print(m,c)

%matplotlib inline

plt.rcParams['figure.figsize'] = (10.0, 5.0)
# max_x = np.max(X) + 100
# min_x = np.min(X) - 100

y = m * X + c
print(y)

X_i = int(input("Enter the head size\n"))
Y_n = m*X_i+c
print(Y_n)
# Ploting Line
plt.plot(X, y, color='blue', label='Regression Line')
# Ploting Scatter Points
plt.scatter(X, Y, c='green', label='Scatter data')
plt.xlabel('Head Size in cm3')
plt.ylabel('Brain Weight in grams')
plt.legend()
plt.show()

rmse = 0
for i in range(n):
    y_pred = c + m * X[i]
    rmse += (Y[i] - y_pred) ** 2

rmse = np.sqrt(rmse/n)
print("Root Mean Square Error is",rmse)

ss_tot = 0
ss_res = 0
for i in range(n):
    y_pred = c + m * X[i]
    ss_tot += (Y[i] - mean_y) ** 2
    ss_res += (Y[i] - y_pred) ** 2
r2 = 1 - (ss_res/ss_tot)
print("R2 Score",r2)
```

2. **Implement Simple Linear regression using price column as the dependent variable and the column total_sqft_int as the independent variable using the file hprice.csv. Find the root mean square error and R squared value. Predict the price for one new price 1425 and then for 3 new prices.**

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

df=pd.read_csv('hprice.csv')

df.head()
print(df)

X = df['total_sqft_int'].values
Y = df['price'].values

mean_x=np.mean(X)
mean_y=np.mean(Y)

n=len(X)
```

```
numer=0
denom=0
for i in range(n):
    numer+=(X[i]-mean_x)*(Y[i]-mean_y)
    denom+=(X[i]-mean_x)**2
m=numer/denom
c=mean_y-(m*mean_x)
print(m,c)

y = m * X + c
print(y)

rmse = 0
for i in range(n):
    y_pred = c + m * X[i]
    rmse += (Y[i] - y_pred) ** 2
rmse = np.sqrt(rmse/n)
print("Root Mean Square Error is",rmse)

ss_tot = 0
ss_res = 0
for i in range(n):
    y_pred = c + m * X[i]
    ss_tot += (Y[i] - mean_y) ** 2
    ss_res += (Y[i] - y_pred) ** 2
r2 = 1 - (ss_res/ss_tot)
print("R2 Score",r2)

X_i = int(input("Enter the new Price"))
Y_n = m*X_i+c
print(Y_n)

%matplotlib inline

plt.plot(X, y, color='blue', label='Regression Line')
plt.scatter(X_i, Y_n, c='black', label='predicted data')
# Ploting Scatter Points
plt.scatter(X, Y, c='green', label='Scatter data')

plt.xlabel('Total sqft')
plt.ylabel('Price')
plt.legend()
plt.show()
```

3. **Implement Multilinear Regression using the data in the file “MyData.csv”. Display the coefficients. (that is $\beta_1, \beta_2, \beta_3$) and display the intercept(that is β_0) also. Find Y for the new data given below.**

Y	**X1**	**X2**	**X3**
?	**50**	**70**	**80**
?	**30**	**40**	**50**

```
import pandas as pd
df=pd.read_csv('MyData.csv')
df.head()

X=df.drop('Y',axis=1)
print(X)

X.insert(0,'B0',[1,1,1,1,1,1])
print(X)

Y=df['Y']
X=X.values
print(X)
Y=Y.values
print(Y)
XT=X.T
print(XT)

XTX=XT.dot(X)
import numpy as np
XTXINV=np.linalg.inv(XTX)
print(XTXINV)

XTY=XT.dot(Y)
print(XTY)
BHAT=XTXINV.dot(XTY)
print("Bhat:",BHAT)
print(BHAT[0]+BHAT[1]*50+BHAT[2]*70+BHAT[3]*80)
print(BHAT[0]+BHAT[0]*30+BHAT[0]*40+BHAT[0]*50)
```

4. **Implement Multiple Linear Regression to predict the price given the data set below. Do data preprocessing to fill the null value. (Hint fill the null value with the median). Display the coefficients. (that is $\beta_1, \beta_2, \beta_3$) and display the intercept(that is β_0) also.**

Area	**Bedrooms**	**Age**	**Price**
Predict the price			
3000	**3**	**40**	**?**
2500	**4**	**5**	**?**

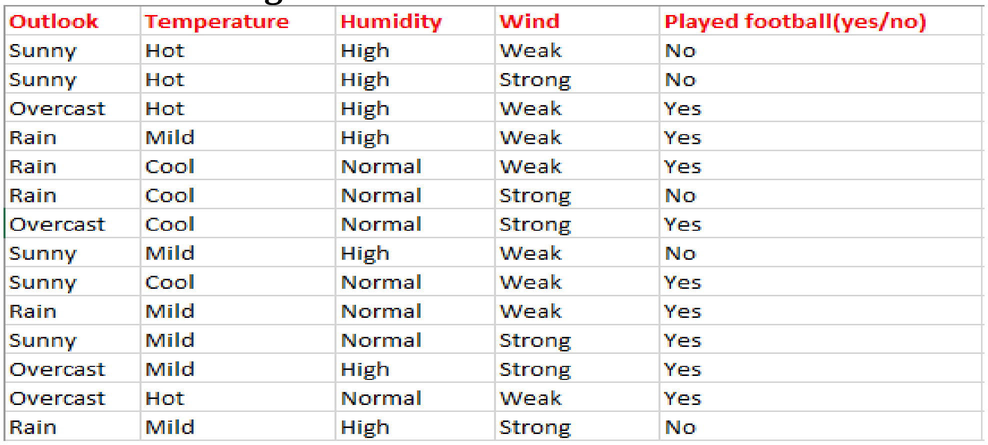
```
import pandas as pd
import numpy as np
df=pd.read_csv("HPriceData.csv")
print(df)

data=df.fillna(df.median())
print(data)

X=data[['Area','Bedrooms','Age']]
Y=data['Price']
print("X",X)
print("Y",Y)

X.insert(0,'X0',len(X)*[1])
print(X)

XT=np.transpose(X)
print("XT",XT)
XTX=np.dot(XT,X)
print("XTX",XTX)
XTXI=np.linalg.inv(XTX)
print("XTXI",XTXI)
XTY=np.dot(XT,Y)
print("XTY",XTY)
BHAT=np.dot(XTXI,XTY)
print("BHAT",BHAT)

y_pred=BHAT[0]+(BHAT[1]*3000)+(BHAT[2]*3)+(BHAT[3]*40)
print("predicted value1",y_pred)
y_pred=BHAT[0]+(BHAT[1]*2500)+(BHAT[2]*4)+(BHAT[3]*5)
print("predicted value2",y_pred)
```

5. **Generate random dataset using the following code.**
   Display the values of x and y.
   Predict the value of y given x as 2.30965656
   Generate a best fit line for the data using polynomial regression

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

np.random.seed(0)
x = 2 - 3 * np.random.normal(0, 1, 20)
y = x - 2 * (x ** 2) + 0.5 * (x ** 3) + np.random.normal(-3, 3, 20)
y_pred=[]

df = pd.DataFrame({'Y':y,'X':x})
df.insert(0,'B0',1)
df.insert(3,'X2',x*x)
print(df)

X=df.drop('Y',axis=1)
X=X.values
y=df['Y']
y=y.values

xt = X.T
xtx = np.dot(xt,X)
xtxinv = np.linalg.inv(xtx)
xty = np.dot(xt,y)
bhat = np.dot(xtxinv,xty)
print(bhat)

n = len(x)

plt.scatter(df['X'],y, s=10)
plt.scatter(df['X2'],y, s=10)
plt.show()

rmse = 0
for i in range(n):
    y_pred = bhat[0]+bhat[1]*df['X']+bhat[2]*df['X2']
    rmse += (y[i] - y_pred) ** 2

rmse = np.sqrt(rmse/n)
```

```
print("Root Mean Square Error is\n",rmse)

mean_y=np.mean(y)
ss_tot = 0
ss_res = 0
for i in range(n):
    y_pred = bhat[0]+bhat[1]*df['X']+bhat[2]*df['X2']
    ss_tot += (y[i] - mean_y) ** 2
    ss_res += (y[i] - y_pred) ** 2
r2 = 1 - (ss_res/ss_tot)
print("R2 Score\n",r2)

x_new=float(input("Enter the value of X:"))
print(bhat[0]+bhat[1]*x_new+bhat[2]*(x_new**2))
```

6. Refer the below given dataset.



Outlook	Temperature	Humidity	Wind	Played football(yes/no)
Sunny	Hot	High	Weak	No
Sunny	Hot	High	Strong	No
Overcast	Hot	High	Weak	Yes
Rain	Mild	High	Weak	Yes
Rain	Cool	Normal	Weak	Yes
Rain	Cool	Normal	Strong	No
Overcast	Cool	Normal	Strong	Yes
Sunny	Mild	High	Weak	No
Sunny	Cool	Normal	Weak	Yes
Rain	Mild	Normal	Weak	Yes
Sunny	Mild	Normal	Strong	Yes
Overcast	Mild	High	Strong	Yes
Overcast	Hot	Normal	Weak	Yes
Rain	Mild	High	Strong	No

Create a decision tree from scratch using the above dataset using ID3 algorithm.
Given a new set of features, predict whether game will be played.

Outlook	**Temperature**	**Humidity**	**Wind**	**Play**
Rain	**Hot**	**Normal**	**Weak**	**?**

```
import numpy as np
import math
import pandas as pd
df=pd.read_csv('PlayTennis.csv')
print(df)

outlook=df['Outlook'].tolist()
temp=df['Temperature'].tolist()
```

```
humidity=df['Humidity'].tolist()
wind=df['Wind'].tolist()
play=df['Play'].tolist()

totoutlookraws=len(outlook)

totyes = totno = 0

for i in play:
    if i == 'Yes':
        totyes += 1
    elif i == 'No':
        totno += 1
print("\nyes is:\n",totyes)
print("\nNo is:\n",totno)

totyesno=totyes+totno
print("\nTotal yes and no count is:\n",totyesno)

entropy=-((totyes/totyesno)*math.log2(totyes/totyesno))-
((totno/totyesno)*math.log2(totno/totyesno))
print("\nEntropy:\n",entropy)

sunnyyes = sunnyno = overyes = overno = rainyes = rainno = 0

for i,j in zip(outlook,play):
     if i=="Sunny" and j=="Yes":
         sunnyyes += 1
     elif i=="Sunny" and j=="No":
         sunnyno += 1
     elif i=="Overcast" and j=="Yes":
         overyes += 1
     elif i=="Overcast" and j=="No":
         overno += 1
     elif i=="Rain" and j=="Yes":
         rainyes += 1
     elif i=="Rain" and j=="No":
         rainno += 1
print("\nsunny yes:\t",sunnyyes,"\nsunny no:\t",sunnyno,"\novercast
yes:\t",overyes,"\novercast no:\t",overno,"\nrain yes:\t",rainyes,"\nrain no:\t",rainno)

totsunny = sunnyyes + sunnyno
totovercast = overyes + overno
```

```
totrain = rainyes + rainno
print("\nTotal sunny:\t",totsunny,"\nTotal overcast:\t",totovercast,"\nTotal rain:\t",totrain)

if totsunny == sunnyyes or totsunny == sunnyno:
    sunnyentropy = 0
else:
    sunnyentropy = -((sunnyyes/totsunny)*math.log2(sunnyyes/totsunny))-
((sunnyno/totsunny)*math.log2(sunnyno/totsunny))

print("\nSunny Entropy:\n",sunnyentropy)

if totovercast == overyes or totovercast == overno:
    overcastentropy = 0
else:
    overcastentropy = -((overyes/totovercast)*math.log2(overyes/totovercast))-
((overno/totovercast)*math.log2(overno/totovercast))
print("\nOvercast Entropy:\n",overcastentropy)

if totrain == rainyes or totrain == rainno:
    rainentropy = 0
else:
    rainentropy = -((rainyes/totrain)*math.log2(rainyes/totrain))-
((rainno/totrain)*math.log2(rainno/totrain))
print("\nRain Entropy:\n",rainentropy)

gainoutlook=entropy-(totsunny/totyesno)*sunnyentropy-
(totovercast/totyesno)*overcastentropy-(totrain/totyesno)*rainentropy
print("\nOutlook Gain is:\t",gainoutlook)

print("\n---------------------Temperature-----------------------\n")
#temperature
tottemprows=len(temp)

hotyes = hotno = mildyes = mildno = coolyes = coolno = 0

for i,j in zip(temp,play):
    if i=="Hot" and j=="Yes":
        hotyes += 1
    elif i=="Hot" and j=="No":
        hotno += 1
    elif i=="Mild" and j=="Yes":
        mildyes += 1
    elif i=="Mild" and j=="No":
         mildno += 1
     elif i=="Cool" and j=="Yes":
         coolyes += 1
     elif i=="Cool" and j=="No":
         coolno += 1
print("\nHot yes:\t",hotyes,"\nHot no:\t",hotno,"\nMild yes:\t",mildyes,"\nMild
no:\t",mildno,"\nCool yes:\t",coolyes,"\nCool no:\t",coolno)

tothot = hotyes + hotno
totcool = coolyes + coolno
totmild = mildyes + mildno
print("\nTotal hot:\t",tothot,"\nTotal cool:\t",totcool,"\nTotal mild:\t",totmild)

if tothot == hotyes or tothot == hotno:
    hotentropy = 0
else:
    hotentropy = -((hotyes/tothot)*math.log2(hotyes/tothot))-
((hotno/tothot)*math.log2(hotno/tothot))

print("\nHot Entropy:\n",hotentropy)

if totcool == coolyes or totcool == coolno:
    coolentropy = 0
else:
    coolentropy = -((coolyes/totcool)*math.log2(coolyes/totcool))-
((coolno/totcool)*math.log2(coolno/totcool))
print("\nCool Entropy:\n",coolentropy)

if totmild == mildyes or totmild == mildno:
    mildentropy = 0
else:
    mildentropy = -((mildyes/totmild)*math.log2(mildyes/totmild))-
((mildno/totmild)*math.log2(mildno/totmild))
print("\nMild Entropy:\n",mildentropy)

gaintemp=entropy-(tothot/totyesno)*hotentropy-(totcool/totyesno)*coolentropy-
(totmild/totyesno)*mildentropy
print("\nTemperature Gain is:\t",gaintemp)

print("\n---------------------Humidity-----------------------\n")
#humidity
highyes = highno = normalyes = normalno = 0
```

```
for i,j in zip(humidity,play):
    if i=="High" and j=="Yes":
        highyes += 1
    elif i=="High" and j=="No":
        highno += 1
    elif i=="Normal" and j=="Yes":
        normalyes += 1
    elif i=="Normal" and j=="No":
        normalno += 1
print("\nHigh yes:\t",highyes,"\nHigh no:\t",highno,"\nNormal
yes:\t",normalyes,"\nNormal no:\t",normalno)

tothigh = highyes + highno
totnormal = normalyes + normalno
print("\nTotal high:\t",tothigh,"\nTotal normal:\t",totnormal)

if tothigh == highyes or tothigh == highno:
   highentropy = 0
else:
   highentropy = -((highyes/tothigh)*math.log2(highyes/tothigh))-
((highno/tothigh)*math.log2(highno/tothigh))

print("\nHot Entropy:\n",highentropy)

if totnormal == normalyes or totnormal == normalno:
   normalentropy = 0
else:
   normalentropy = -((normalyes/totnormal)*math.log2(normalyes/totnormal))-
((normalno/totnormal)*math.log2(normalno/totnormal))
print("\nCool Entropy:\n",normalentropy)

gainhumid=entropy-(tothigh/totyesno)*highentropy-(totnormal/totyesno)*normalentropy
print("\nHumidity Gain is:\t",gainhumid)

print("\n---------------------Wind-----------------------\n")
#humidity
weakyes = weakno = strongyes = strongno = 0

for i,j in zip(wind,play):
    if i=="Weak" and j=="Yes":
        weakyes += 1
    elif i=="Weak" and j=="No":
        weakno += 1
    elif i=="Strong" and j=="Yes":
        strongyes += 1
    elif i=="Strong" and j=="No":
        strongno += 1
print("\nStrong yes:\t",strongyes,"\nStrong no:\t",strongno,"\nWeak
yes:\t",weakyes,"\nWeak no:\t",weakno)

tothigh = highyes + highno
totnormal = normalyes + normalno
print("\nTotal high:\t",tothigh,"\nTotal normal:\t",totnormal)

if tothigh == highyes or tothigh == highno:
    highentropy = 0
else:
    highentropy = -((highyes/tothigh)*math.log2(highyes/tothigh))-
((highno/tothigh)*math.log2(highno/tothigh))

print("\nHigh Entropy:\n",highentropy)

if totnormal == normalyes or totnormal == normalno:
    normalentropy = 0
else:
    normalentropy = -((normalyes/totnormal)*math.log2(normalyes/totnormal))-
((normalno/totnormal)*math.log2(normalno/totnormal))
print("\nNormal Entropy:\n",normalentropy)

gainhumid=entropy-(tothigh/totyesno)*highentropy-(totnormal/totyesno)*normalentropy
print("\nHumidity Gain is:\t",gainhumid)
```

7. **Given the csv file “Social_Network_Ads” with independent variables “Age” and “EstimatedSalary” and dependent variable “Purchased”, predict if a vehicle will be purchased by a person who is 36 year old with estimated salary 76000.**
   Develop the Logistic Regression code from scratch.

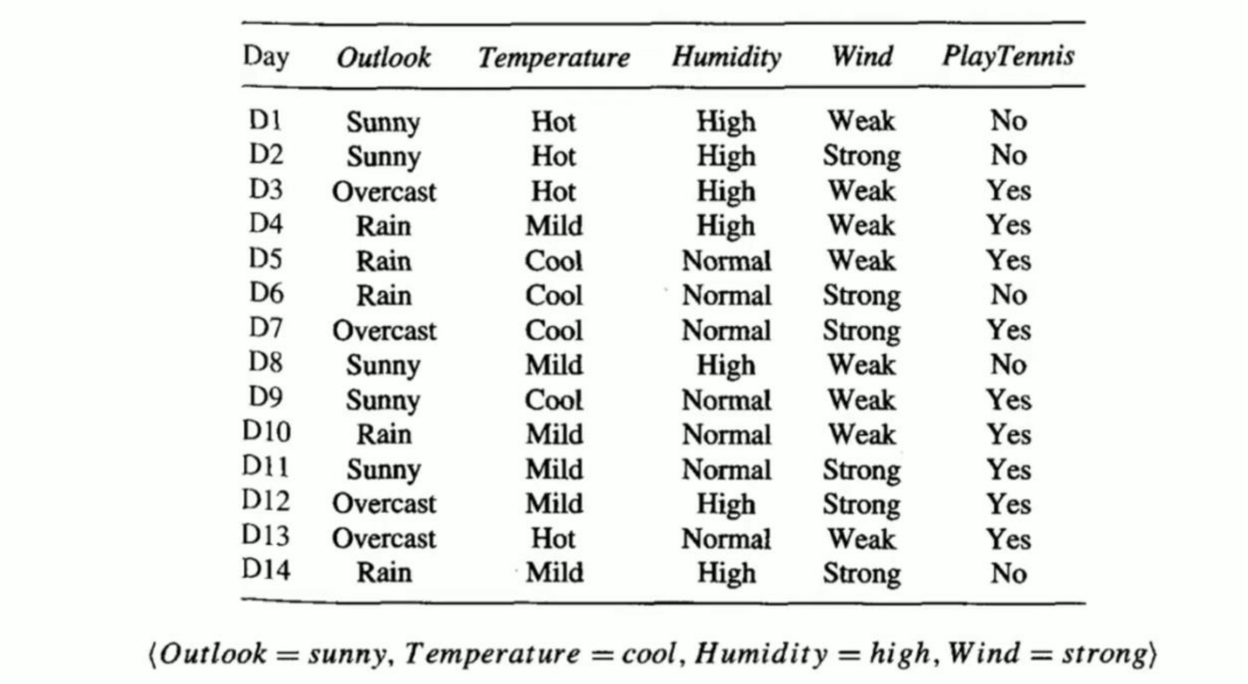
```
import numpy as np
import pandas as pd

df=pd.read_csv('Social_Network_Ads.csv')
print(df)
gen={'Male':1,'Female':0}
df.Gender=[gen[item]for item in df.Gender]
print(df)

X=df[['Age','EstimatedSalary']]
Y=df['Purchased']
print(X,”\n”,Y)

X.insert(0,'B0',1)
xt=X.T
print(“X”,X,"\nXT",xt)

XTX=np.dot(xt,X)
print("XTX",XTX)

XTXI=np.linalg.inv(XTX)
print("XTXI",XTXI)
XTY=np.dot(xt,Y)
print("XTY",XTY)
BHAT=np.dot(XTXI,XTY)
print("BHAT",BHAT)

age = int(input("Enter the age : "))
salary = float(input("Enter the salary : "))
y_pred=BHAT[0]+BHAT[1]*age+BHAT[2]*salary
print("Y_PRED",y_pred)
prediction=1/(1+(2.718)**-y_pred)
print("Value of sigmoid function : ",prediction)
if prediction>0.5:
   print("PREDICTION : YES")
else:
   print("PREDICTION : NO")
```

8. Given the support vectors
S1=(1,0) -> negatively labeled data point
S2=(3,1) -> positively labeled data point
S3=(3,-1) -> positively labeled data point
Find the weights and the intercept.
Classify the new data point (4,2) as either negative labeled or positive labelled

```
import numpy as np
import pandas as pd

s1 = [1,0]
s2 = [3,1]
s3 = [3,-1]

s1.append(1)
s2.append(1)
s3.append(1)

s1t=np.transpose(s1)
s2t=np.transpose(s2)
s3t=np.transpose(s3)
print("Transpose")
print(s1t)
print(s2t)
print(s3t)

s1ts1=np.dot(s1t,s1)
s1ts2=np.dot(s1t,s2)
s1ts3=np.dot(s1t,s3)
s2ts1=np.dot(s2t,s1)
s2ts2=np.dot(s2t,s2)
s2ts3=np.dot(s2t,s3)
s3ts1=np.dot(s3t,s1)
s3ts2=np.dot(s3t,s2)
s3ts3=np.dot(s3t,s3)

x=np.array([[s1ts1,s1ts2,s1ts3],[s2ts1,s2ts2,s2ts3],[s3ts1,s3ts2,s3ts3]])

xt=np.transpose(x)
xtx=np.dot(xt,x)
xtxi=np.linalg.inv(xtx)

y=np.array([-1,1,1])
```

```
xty=np.dot(xt,y)

bhat=np.dot(xtxi,xty)

w=np.array([])
w=bhat[0] *s1t+bhat[1]*s2t+bhat[2]*s3t

print("\nw",w)

b=np.array(round(w[2]))
w=np.array([w[0],w[1]])

x=int(input("Enter x "))
y=int(input("Enter y "))
p=np.array([x,y])
res=round(np.dot(w,p))
print("\nRes",res)

if(res>b):
    print("\nPositively Classified")
elif (res<b):
    print("\nNegatively Classified")
```

9. **Given the data**



Day	*Outlook*	*Temperature*	*Humidity*	*Wind*	*PlayTennis*
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

$\langle Outlook = sunny, Temperature = cool, Humidity = high, Wind = strong\rangle$

For New Instance given above, find if the game will be played. Use Naïve Bayes

```
import numpy as np
import pandas as pd
import math

df = pd.read_csv("PlayTennis.csv")
print(df)

outlook = df['Outlook'].tolist()
Temp = df['Temperature'].tolist()
humidity = df['Humidity'].tolist()
wind = df['Wind'].tolist()
play = df['Play'].tolist()

ycount = ncount = 0
for i in play:
    if i == "Yes":
        ycount += 1
    elif i == "No":
        ncount += 1
totcount = ycount + ncount

play_yes = ycount / totcount
```

```
play_no = ncount / totcount

suscount = suncount = oscount = oncount = rscount = rncount = 0
for i, j in zip(outlook, play):
    if i == "Sunny" and j == "Yes":
        suscount += 1
    elif i == "Sunny" and j == "No":
        suncount += 1
    elif i == "Overcast" and j == "Yes":
        oscount += 1
    elif i == "Overcast" and j == "No":
        oncount += 1
    elif i == "Rain" and j == "Yes":
        rscount += 1
    elif i == "Rain" and j == "No":
        rncount += 1

hscount=hncount=cscount=cncount=mscount=mncount=0
for i, j in zip(Temp, play):
    if i == "hot" and j == "yes":
        hscount += 1
    elif i == "hot" and j == "no":
        hncount += 1
    elif i == "Cool" and j == "Yes":
        cscount += 1
    elif i == "Cool" and j == "No":
        cncount += 1
    elif i == "Mild" and j == "Yes":
        mscount += 1
    elif i == "Mild" and j == "No":
        mncount += 1

hiscount=hincount=nscount=nncount=0
for i, j in zip(humidity, play):
    if i == "High" and j == "Yes":
        hiscount += 1
    elif i == "High" and j == "No":
        hincount += 1
    elif i == "Normal" and j == "Yes":
        nscount += 1
    elif i == "Normal" and j == "No":
        nncount += 1
```

```
wscount = wncount = stscount = stncount = 0
for i, j in zip(wind, play):
    if i == "Weak" and j == "Yes":
        wscount += 1
    elif i == "Weak" and j == "No":
        wncount += 1
    elif i == "Strong" and j == "Yes":
        stscount += 1
    elif i == "Strong" and j == "No":
        stncount += 1

vnb_yes = play_yes * (suscount / ycount) * (cscount / ycount) * (hiscount / ycount) * (stscount / ycount)
vnb_no = play_no * (suncount / ncount) * (cncount / ncount) * (hiscount / ncount) * (stncount / ncount)

print("\nYes",vnb_yes,"\nNo",vnb_no)

vnb_yes1 = vnb_yes / (vnb_yes + vnb_no)
print("\nprob_yes",vnb_yes1)

vnb_no1 = vnb_yes / (vnb_yes + vnb_no)
print("prob_no",vnb_no1)
if (vnb_yes1 > vnb_no1):
    print("\nYes")
else:
    print("\nNo")
```

10. Given the data

	Height	**Weight**
1	**185**	**72**
2	**170**	**56**
3	**168**	**60**
4	**179**	**68**
5	**182**	**72**
6	**188**	**77**
7	**180**	**71**
8	**180**	**70**
9	**183**	**84**
10	**180**	**88**
11	**180**	**67**
12	**177**	**76**

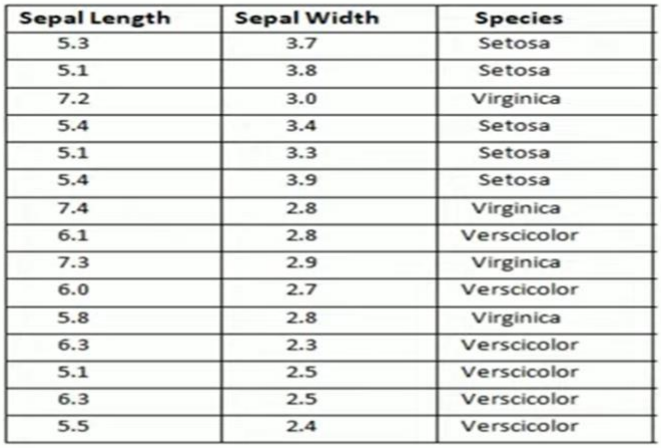
Divide the above given data points into 2 clusters using k-Means clustering.

```
import pandas as pd
import numpy as np
import math
df=pd.read_csv("kmeans.csv")

x = df['Height'].values
y = df['Weight'].values
inc1 = [185,72]
inc2 = [170,56]

def calkmean(c1,c2,ii):
        resc1 = []
        resc2 = []
        for i,j in zip(x,y):
             res =  math.sqrt((c1[0]-i) **2 + (c1[1]-j) **2)
             resc1.append(res)
```

```
            res =  math.sqrt((c2[0]-i) **2 + (c2[1]-j) **2)
            resc2.append(res)
        clnumber = []
        for i,j in zip(resc1,resc2):
            if (i<j):
                clnumber.append("c1")
            elif (j<i):
                clnumber.append("c2")
        nc1 = []
        nc2 = []
        c1resx = []
        c2resx = []
        for i,j in zip(x,clnumber):
            if j == "c1":
                c1resx.append(i)
            elif j == "c2":
                c2resx.append(i)
        nc1.append(round(np.mean(c1resx),2))
        nc2.append(round(np.mean(c2resx),2))
        c1resy = []
        c2resy = []
        for i,j in zip(y,clnumber):
            if j == "c1":
                c1resy.append(i)
            elif j == "c2":
                c2resy.append(i)
        fc1 = []
        fc1= [c1resx[1],c1resy[1]]
        nc1.append(round(np.mean(c1resy),2))
        nc2.append(round(np.mean(c2resy),2))
        return nc1,nc2,c1resx,c1resy,c2resx,c2resy
nc1 = []
nc2 = []
i = 0
cmpc1 = inc1
cmpc2 = inc2
nc1,nc2,c1resx,c1resy,c2resx,c2resy, = calkmean(inc1,inc2,i)
for i in range(10):
    if (cmpc1 != nc1) or (cmpc2 != nc2):
        cmpc1 = nc1
        cmpc2 = nc2
        i+= 1
        nc1,nc2,c1resx,c1resy,c2resx,c2resy = calkmean(nc1,nc2,i)
```

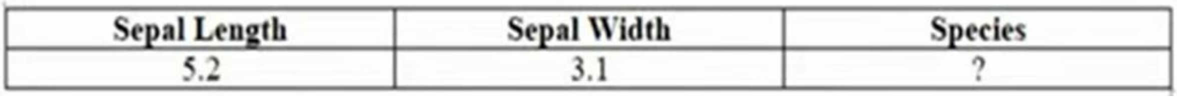
```
 else:
    print("Final Cluster")
    print("Cluster 1 ",nc1," Cluster 2 ",nc2)
    print("c1 cluster")
    for i in range(len(c1resx)):
       print(c1resx[i],c1resy[i])
    print("c2 cluster")
    for i in range(len(c2resx)):
       print(c2resx[i],c2resy[i])
    break
```

11. Given the data



Sepal Length	Sepal Width	Species
5.3	3.7	Setosa
5.1	3.8	Setosa
7.2	3.0	Virginica
5.4	3.4	Setosa
5.1	3.3	Setosa
5.4	3.9	Setosa
7.4	2.8	Virginica
6.1	2.8	Verscicolor
7.3	2.9	Virginica
6.0	2.7	Verscicolor
5.8	2.8	Virginica
6.3	2.3	Verscicolor
5.1	2.5	Verscicolor
6.3	2.5	Verscicolor
5.5	2.4	Verscicolor

Classify the new instance given below using K Nearest Neighbour..

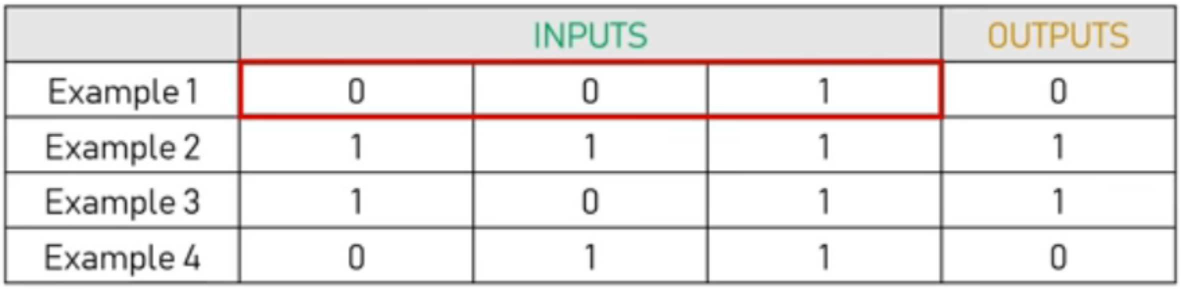


Sepal Length	Sepal Width	Species
5.2	3.1	?

```
import pandas as pd
import numpy as np
import math

df=pd.read_csv("knn.csv")

x=df['Sepal Length'].values
y=df['Sepal Width'].values
z=df['Species'].values

k=math.sqrt(len(x))
round(k+1)
sl=float(input("Enter sepal Length"))
sw=float(input("Enter sepal Width"))
n=len(x)
```

```
ls=[]
for i in range(15):
    v1=((x[i]-sl)**2)
    v2=((y[i]-sw)**2)
    d=math.sqrt(v1+v2)
    ls.append(d)

for i in range(n):
    ls[i]==0.2236
pair=zip(z,ls)
from operator import itemgetter
from heapq import nsmallest
result = nsmallest(5, pair, key=itemgetter(1))

def Extract(lst):
    return list(list(zip(*lst))[0])

k=[]
k=Extract(result)

s=0
vir=0
ver=0
ll=len(k)
for i in range(ll):
    if k[i]=='Setosa':
        s+=1
    elif k[i]=="Virginica":
        vir+=1
    elif k[i]=="Verscicolor":
        ver+=1
p=max(s,vir,ver)

if s>ver:
    if s>vir:
        print("Belongs to Setosa")
    else:
        print("Belongs to Virginica")
else:
    if vir>ver:
        print("Belongs to Virginica")
    else:
        print("Belongs to Verscicolor")
```

12. Implement Artficial Neural Network for the first output using the below given records and calculate the total loss.



	INPUTS			OUTPUTS
Example 1	0	0	1	0
Example 2	1	1	1	1
Example 3	1	0	1	1
Example 4	0	1	1	0

Initialize the weights as w1=0.15, w2=0.20 and w3=0.25
Use sigmoid activation function.

```
import numpy as np
def sigmoid(x):
    return (1/(1+np.exp(-x)))
trainX=np.array([[0,0,1],
                [1,1,1],
                [1,0,1],
                [0,1,1]])
trainY=np.array([[0,1,1,0]]).T
weights=np.array([0.15,0.20,0.25])
print(weights)
for i in range(1):
    input_layer=trainX
    output=sigmoid(np.dot(input_layer,weights))
print("Outputs after training")
print(output)
rmse=0
for i in range(len(trainY)):
     y_pred=sigmoid(np.dot(input_layer[0],weights))
     rmse+=(trainX[0]-y_pred)**2
rmse=np.sqrt(rmse/len(trainY))
print(rmse)

e1=((trainY[0]-output[0])**2)/2
e2=((trainY[1]-output[1])**2)/2
e3=((trainY[2]-output[2])**2)/2
e4=((trainY[3]-output[3])**2)/2
etotal=e1+e2+e3+e4
print("total loss=",etotal)
```