week-6

```
import numpy as np
X = np.array(([2,9],[1,5],[3,6]))
y=np.array(([92],[86],[89]))

y=y/100

#Sigmoid Function
def sigmoid(x):
  return 1/(1+ np.exp(-x))

#Derivatives of Sigmoid function
def derivatives_sigmoid(x):
  return x*(1-x)
#Variable initialization
epoch=10000
lr=0.1
inputlayer_neurons = 2
hiddenlayers_neurons = 3
output_neurons = 1

#weight and bias initialization
wh=np.random.uniform(size=(inputlayer_neurons,hiddenlayers_neurons))
bias_hidden=np.random.uniform(size=(1,hiddenlayers_neurons))
weight_hidden=np.random.uniform(size=(hiddenlayers_neurons,output_neurons))
bias_output=np.random.uniform(size=(1,output_neurons))

for i in range(epoch):
  hinp1=np.dot(X,wh)
```

```
  hinp=hinp1+ bias_hidden
  hlayer_activation = sigmoid(hinp)

  outinp1=np.dot(hlayer_activation,weight_hidden)
  outinp = outinp1+bias_output
  output = sigmoid(outinp)

EO = y-output
outgrad=derivatives_sigmoid(output)
d_output = EO * outgrad
EH = d_output.dot(weight_hidden.T)
hiddengrad=derivatives_sigmoid(hlayer_activation)
d_hiddenlayer = EH * hiddengrad

weight_hidden += hlayer_activation.T.dot(d_output) * lr
bias_hidden += np.sum(d_hiddenlayer, axis=0,keepdims=True) * lr
wh += X.T.dot(d_hiddenlayer) * lr
bias_output += np.sum(d_output,axis=0,keepdims=True) *lr

print("Input: \n"+str(X))
print("Actual Output: \n"+str(y))
print("Predicted Output: \n",output)
```

OUTPUT

```
Input:
[[2 9]
 [1 5]
 [3 6]]
Actual Output:
```

[[0.92]
 [0.86]
 [0.89]]
Predicted Output:
 [[0.89093595]
 [0.88074638]
 [0.89006502]]

WEEK-8

IMPORT PACKAGES

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

from google.colab import files
uploaded=files.upload()

#  Create New Variable and stores the dataset values as Data Frame
# READING DATA
train=pd.read_csv("train.csv")
train

test=pd.read_csv("test.csv")
test
```

a) Show few rows from the first five and last five record from the dataset

- train # We have 614 rows and 13 columns in the train dataset.

```
train # We have 614 rows and 13 columns in the train dataset.
```

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property_Area	Loan_Status
0	LP001002	Male	No	0	Graduate	No	5849	0.0	NaN	360.0	1.0	Urban	Y
1	LP001003	Male	Yes	1	Graduate	No	4583	1508.0	128.0	360.0	1.0	Rural	N
2	LP001005	Male	Yes	0	Graduate	Yes	3000	0.0	66.0	360.0	1.0	Urban	Y
3	LP001006	Male	Yes	0	Not Graduate	No	2583	2358.0	120.0	360.0	1.0	Urban	Y
4	LP001008	Male	No	0	Graduate	No	6000	0.0	141.0	360.0	1.0	Urban	Y
...	...	...	...	...	...	...	...	...	...	...	...	...	...
609	LP002978	Female	No	0	Graduate	No	2900	0.0	71.0	360.0	1.0	Rural	Y
610	LP002979	Male	Yes	3+	Graduate	No	4106	0.0	40.0	180.0	1.0	Rural	Y
611	LP002983	Male	Yes	1	Graduate	No	8072	240.0	253.0	360.0	1.0	Urban	Y
612	LP002984	Male	Yes	2	Graduate	No	7583	0.0	187.0	360.0	1.0	Urban	Y
613	LP002990	Female	No	0	Graduate	Yes	4583	0.0	133.0	360.0	0.0	Semiurban	N

614 rows × 13 columns

- test #We have 367 rows and 12 columns in test dataset.

```
test  #We have 367 rows and 12 columns in test dataset.
```

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property_Area
0	LP001015	Male	Yes	0	Graduate	No	5720	0	110.0	360.0	1.0	Urban
1	LP001022	Male	Yes	1	Graduate	No	3076	1500	126.0	360.0	1.0	Urban
2	LP001031	Male	Yes	2	Graduate	No	5000	1800	208.0	360.0	1.0	Urban
3	LP001035	Male	Yes	2	Graduate	No	2340	2546	100.0	360.0	NaN	Urban
4	LP001051	Male	No	0	Not Graduate	No	3276	0	78.0	360.0	1.0	Urban
...	...	...	...	...	...	...	...	...	...	...	...	...
362	LP002971	Male	Yes	3+	Not Graduate	Yes	4009	1777	113.0	360.0	1.0	Urban
363	LP002975	Male	Yes	0	Graduate	No	4158	709	115.0	360.0	1.0	Urban
364	LP002980	Male	No	0	Graduate	No	3250	1993	126.0	360.0	NaN	Semiurban
365	LP002986	Male	Yes	0	Graduate	No	5000	2393	158.0	360.0	1.0	Rural
366	LP002989	Male	No	0	Graduate	Yes	9200	0	98.0	180.0	1.0	Rural

367 rows × 12 columns

b) Shows the important information from the dataset.

- train.describe()

```
train.describe()
```

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History
count	614.000000	614.000000	592.000000	600.00000	564.000000
mean	5403.459283	1621.245798	146.412162	342.00000	0.842199
std	6109.041673	2926.248369	85.587325	65.12041	0.364878
min	150.000000	0.000000	9.000000	12.00000	0.000000
25%	2877.500000	0.000000	100.000000	360.00000	1.000000
50%	3812.500000	1188.500000	128.000000	360.00000	1.000000
75%	5795.000000	2297.250000	168.000000	360.00000	1.000000
max	81000.000000	41667.000000	700.000000	480.00000	1.000000

c) Show us the information about the dataset, Like What's the type of culumn

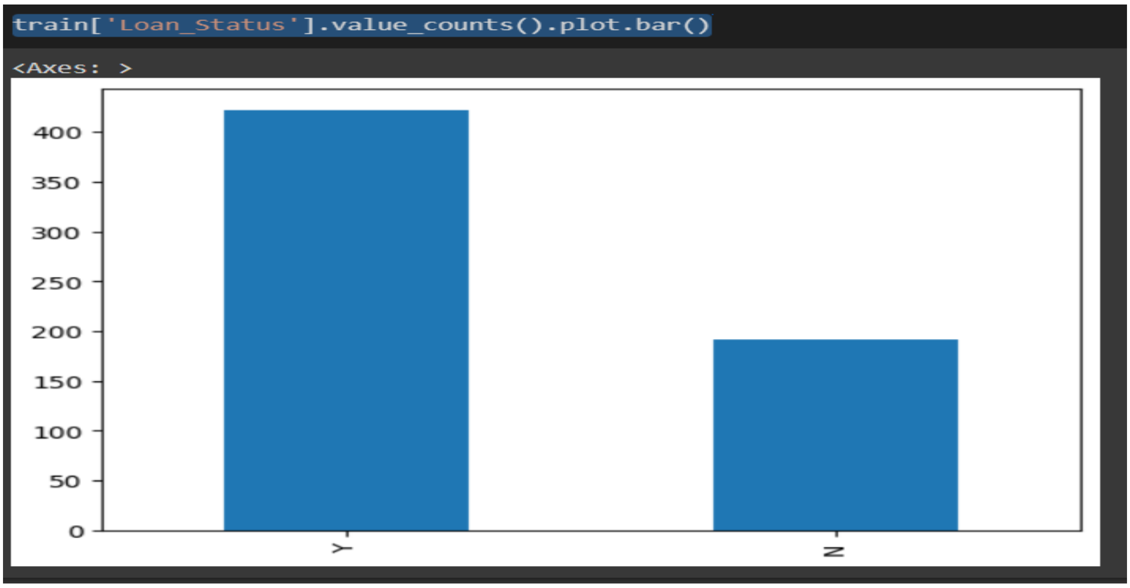
have? How many rows available in the dataset? What are the features are there? How many null values available in the dataset?

- train.info()

```
train.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 614 entries, 0 to 613
Data columns (total 13 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Loan_ID            614 non-null    object
 1   Gender             601 non-null    object
 2   Married            611 non-null    object
 3   Dependents         599 non-null    object
 4   Education          614 non-null    object
 5   Self_Employed      582 non-null    object
 6   ApplicantIncome    614 non-null    int64
 7   CoapplicantIncome  614 non-null    float64
 8   LoanAmount         592 non-null    float64
 9   Loan_Amount_Term   600 non-null    float64
 10  Credit_History     564 non-null    float64
 11  Property_Area      614 non-null    object
 12  Loan_Status        614 non-null    object
dtypes: float64(4), int64(1), object(8)
memory usage: 62.5+ KB
```

- train['Loan_Status'].value_counts().plot.bar()



- Categorical features: These features have categories (Gender, Married ,Self_Employed, Credit_History, Loan_Status)
- Ordinal features: Variables in categorical features having some order involved (Dependents, Education, Property_Area)
- Numerical features: These features have numerical values (ApplicantIncome, CoapplicantIncome, LoanAmount, Loan_Amount_Term)

Independent Variable (Categorical)

train['Gender'].value_counts(normalize=True).plot.bar(title='Gender')

plt.show()

train['Married'].value_counts(normalize=True).plot.bar(title='Married')

plt.show()

train['Self_Employed'].value_counts(normalize=True).plot.bar(title='Self_employee')

plt.show()

train['Credit_History'].value_counts(normalize=True).plot.bar(title='Credit_History')

plt.show()

Independent Variable (Ordinal)

```
train['Dependents'].value_counts(normalize=True).plot.bar(title='Dependents')
plt.show()
train['Education'].value_counts(normalize=True).plot.bar(title='Education')
plt.show()
train['Property_Area'].value_counts(normalize=True).plot.bar(title='Property_Area')
plt.show()
```

Independent Variable (Numerical)

Till now we have seen the categorical and ordinal variables and now let's visualize the numerical variables. Let's look at the distribution of Applicant income first.

```
train['ApplicantIncome'].value_counts(normalize=True).plot.bar(title='ApplicantIncome')
plt.show()
train['CoapplicantIncome'].plot.box()
plt.show()
train['LoanAmount'].plot.box()
plt.show()
```

d) Independent Variable (Categorical)

Independent Variable (Categorical)

```
[15] train['Gender'].value_counts(normalize=True).plot.bar(title='Gender')
     plt.show()
     train['Married'].value_counts(normalize=True).plot.bar(title='Married')
     plt.show()
     train['Self_Employed'].value_counts(normalize=True).plot.bar(title='Self_employee')
     plt.show()
     train['Credit_History'].value_counts(normalize=True).plot.bar(title='Credit_History')
     plt.show()
```

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

```
Loan_ID               0
Gender               13
Married               3
Dependents           15
Education             0
Self_Employed        32
ApplicantIncome       0
CoapplicantIncome     0
LoanAmount           22
Loan_Amount_Term     14
Credit_History       50
Property_Area         0
Loan_Status           0
Income_bin            0
dtype: int64
```

e) We can get the shape of the dataset using shape attribute.

we can get the shape of the dataset using shape attribute

We can see there are three formats of data types:

object: Object format means variables are categorical. Categorical variables in our dataset are Loan_ID, Gender, Married, Dependents, Education, Self_Employed, Property_Area, Loan_Status. **int64**: It represents the integer variables. ApplicantIncome is of this format. **float64**: It represents the variable that has some decimal values involved. They are also numerical

```
print(train.shape, test.shape)
```

```
(614, 13) (367, 12)
```

f) Choose ML Model, Traing the ML Model, Predict Model.

4. Choose ML Model.

In this step, We have a lots of Machine Learning Model from sklearn package, and we need to decide which model is give us the better performance. then we use that model in final stage and send to the production level.

(** Model Building Process** - After creating new features, we can continue the model building process. So we will start with the logistic regression model and then move over to more complex models like RandomForest and XGBoost. We will build the following models in this section.)

```
# import ml model from sklearn pacakge

from sklearn.linear_model import LogisticRegression

from sklearn.metrics import accuracy_score
```

First of all, we are use LogisticRegression from sklearn.linear_model package. Here is the little information about LogisticRegression.

Logistic Regression is a classification algorithm. It is used to predict a binary outcome (1 / 0, Yes / No, and True / False) given a set of independent variables. To represent binary / categorical outcome, we use dummy variables. You can also think of logistic regression as a special case of linear regression when the outcome variable is categorical, where we are using log of odds as the dependent variable.

```
[53] # Let's build the model
     logistic_model = LogisticRegression()
```

5. Traing the ML Model

Before fitting the model, We need to decide how many feature are available for testing and training, then after complete this step. fitt the model Currently, we are using Credit_History', 'Education', 'Gender features for training so let's create train and test variables

```
[54] train_features = ['Credit_History', 'Education', 'Gender']

     x_train = train[train_features].values
     y_train = train['Loan_Status'].values

     x_test = test[train_features].values
     logistic_model.fit(x_train, y_train)
```

```
▾ LogisticRegression
LogisticRegression()
```

6. Predict Model

```
[55] # Predict the model for testin data
     predicted = logistic_model.predict(x_test)

     # check the coefficeints of the trained model
     print('Coefficient of model :', logistic_model.coef_)

     # check the intercept of the model
     print('Intercept of model',logistic_model.intercept_)

     # Accuray Score on train dataset
     # accuracy_train = accuracy_score(x_test, predicted)
     score = logistic_model.score(x_train, y_train)
     print('accuracy_score overall :', score)
     print('accuracy_score percent :', round(score*100,2))
```

```
Coefficient of model : [[ 3.316164   -0.3059193   0.09398266]]
Intercept of model [-1.98307795]
accuracy_score overall : 0.8094462540716613
accuracy_score percent : 80.94
```

```
[56] # predict the target on the test dataset
     predict_test = logistic_model.predict(x_test)
     print('Target on test data',predict_test)
```

```
Target on test data [1 1 1 1 1 1 1 0 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 0 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 0 1 1 1 1 0 1 1 0 0 1 0 1 1 1 1
 1 1 1 1 1 1 0 1 0 1 0 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 0 1 1 1 1 0 1 1 1 1
 1 1 1 1 1 1 0 0 0 1 1 1 0 0 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 0 1 1 1 1 0
 1 1 1 1 1 0 1 1 1 1 1 1 1 0 1 1 1 0 0 1 0 1 1 1 1 0 0 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 0 0 1 1 0 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1
 1 1 0 1 1 1 1 0 1 1 1 1 1 0 0 1 1 1 1 0 1 0 1 0 1 1 1 1 0 1 1 1 1 0 1 1 1
 1 1 1 1 1 1 1 0 1 0 1 1 1 1 0 0 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1
 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1
 1 1 1 1 1 1 0 1 1 1 1 1 1 0 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1]
```

```
[57] x=len(predict_test)
     print(x)
```

```
367
```

WEEK-9

Write a program to predict the quality of water.

```
import numpy as np

import pandas as pd


from google.colab import files

uploaded=files.upload()
```

a) Show few rows from the first five and last five record from the dataset.

```
data=pd.read_csv('water_dataX.csv',encoding="ISO-8859-1")

data.replace('NAN',0)
```

```
data=pd.read_csv('water_dataX.csv',encoding="ISO-8859-1")
data.replace('NAN',0)
```

	STATION CODE	LOCATIONS	STATE	Temp	D.O. (mg/l)	PH	CONDUCTIVITY (µmhos/cm)	B.O.D. (mg/l)	NITRATENAN N+ NITRITENANN (mg/l)	FECAL COLIFORM (MPN/100ml)	TOTAL COLIFORM (MPN/100ml)Mean	year
0	1393	DAMANGANGA AT D/S OF MADHUBAN, DAMAN	DAMAN & DIU	30.6	6.7	7.5	203	0	0.1	11	27	2014
1	1399	ZUARI AT D/S OF PT. WHERE KUMBARJRIA CANAL JOI...	GOA	29.8	5.7	7.2	189	2	0.2	4953	8391	2014
2	1475	ZUARI AT PANCHAWADI	GOA	29.5	6.3	6.9	179	1.7	0.1	3243	5330	2014
3	3181	RIVER ZUARI AT BORIM BRIDGE	GOA	29.7	5.8	6.9	64	3.8	0.5	5382	8443	2014
4	3182	RIVER ZUARI AT MARCAIM JETTY	GOA	29.5	5.8	7.3	83	1.9	0.4	3428	5500	2014
...	...	...	...	...	...	...	...	...	...	...	...	...
1986	1330	TAMBIRAPARANI AT ARUMUGANERI, TAMILNADU	0	0	7.9	738	7.2	2.7	0.518	0.518	202	2003
1987	1450	PALAR AT VANIYAMBADI WATER SUPPLY HEAD WORK, T...	0	29	7.5	585	6.3	2.6	0.155	0.155	315	2003
1988	1403	GUMTI AT U/S SOUTH TRIPURA,TRIPURA	0	28	7.6	98	6.2	1.2	0	0	570	2003
1989	1404	GUMTI AT D/S SOUTH TRIPURA, TRIPURA	0	28	7.7	91	6.5	1.3	0	0	562	2003
1990	1726	CHANDRAPUR, AGARTALA D/S OF HAORA RIVER, TRIPURA	0	29	7.6	110	5.7	1.1	0	0	546	2003

1991 rows × 12 columns

b.Shows the Data type of Each and Every Column in the data set.

```
data.dtypes
```

```
data.dtypes

STATION CODE                          object
LOCATIONS                             object
STATE                                 object
Temp                                 float64
D.O. (mg/l)                          float64
PH                                   float64
CONDUCTIVITY (µmhos/cm)              float64
B.O.D. (mg/l)                        float64
NITRATENAN N+ NITRITENANN (mg/l)     float64
FECAL COLIFORM (MPN/100ml)            object
TOTAL COLIFORM (MPN/100ml)Mean       float64
year                                   int64
dtype: object
```

c) data['wph']=data.npH * 0.165

data['wdo']=data.ndo * 0.281

data['wbdo']=data.nbdo * 0.234

data['wec']=data.nec* 0.009

data['wna']=data.nna * 0.028

data['wco']=data.nco * 0.281

Find the Quality of Water and Display data.

- data['wqi']=data.wph+data.wdo+data.wbdo+data.wec+data.wna+data.wco
  data

```
data['wqi']=data.wph+data.wdo+data.wbdo+data.wec+data.wna+data.wco
data
```

```
data['wqi']=data.wph+data.wdo+data.wbdo+data.wec+data.wna+data.wco
data
```

	station	location	state	do	ph	co	bod	na	tc	year	...	nbdo	nec	nna	wph	wdo	wbdo	wec	wna	wco	wqi
2	1475	ZUARI AT PANCHAWADI	GOA	6.300	6.900	179.0	1.7	0.1	5330.0	2014	...	100	60	100	13.2	28.10	23.40	0.54	2.8	11.24	79.28
3	3181	RIVER ZUARI AT BORIM BRIDGE	GOA	5.800	6.900	64.0	3.8	0.5	8443.0	2014	...	80	100	100	13.2	22.48	18.72	0.90	2.8	11.24	69.34
4	3182	RIVER ZUARI AT MARCAIM JETTY	GOA	5.800	7.300	83.0	1.9	0.4	5500.0	2014	...	100	80	100	16.5	22.48	23.40	0.72	2.8	11.24	77.14
5	1400	MANDOVI AT NEGHBOURHOOD OF PANAJI, GOA	GOA	5.500	7.400	81.0	1.5	0.1	4049.0	2014	...	100	80	100	16.5	22.48	23.40	0.72	2.8	11.24	77.14
6	1476	MANDOVI AT TONCA, MARCELA, GOA	GOA	6.100	6.700	308.0	1.4	0.3	5672.0	2014	...	100	0	100	9.9	28.10	23.40	0.00	2.8	11.24	75.44
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1774	1428	KHARKHLA NEAR SUTNGA KHLIERIAT,JAINTIA HILLS D...	NAN	4.600	3.000	350.0	6.2	2.2	49.0	2006	...	60	0	100	0.0	16.86	14.04	0.00	2.8	22.48	56.18
1775	1631	MYNTDU RIVER JOWAI, MEGHALAYA	NAN	8.800	7.000	172.0	1.6	5.0	2800.0	2006	...	100	60	100	16.5	28.10	23.40	0.54	2.8	11.24	82.58
1776	1632	GANOL RIVER TURA, MEGHALAYA	NAN	10.000	7.100	150.0	1.0	4.0	350.0	2006	...	100	80	100	16.5	28.10	23.40	0.72	2.8	16.86	88.38
1777	1633	SIMSANG RIVER WILLIAMNAGAR, MEGHALAYA	NAN	9.000	7.300	158.0	1.8	7.2	280.0	2006	...	100	60	100	16.5	28.10	23.40	0.54	2.8	16.86	88.20
1778	2050	TLAWNG UPSTREAM AIZAWL	NAN	7.767	7.543	NaN	0.5	NaN	NaN	2006	...	100	0	0	16.5	28.10	23.40	0.00	0.0	0.00	68.00

1777 rows × 23 columns

```
[20] #calculation overall wqi for each year
     ag=data.groupby('year')['wqi'].mean()
```

```
ag.head()
```

```
year
2006    71.308824
2007    72.549000
2008    72.570943
2009    74.085193
2010    74.648723
Name: wqi, dtype: float64
```

```
data=ag.reset_index(level=0,inplace=False)
data
```

	year	wqi
0	2006	71.308824
1	2007	72.549000
2	2008	72.570943
3	2009	74.085193
4	2010	74.648723
5	2011	75.647013
6	2012	78.969041
7	2013	75.009425
8	2014	76.879588

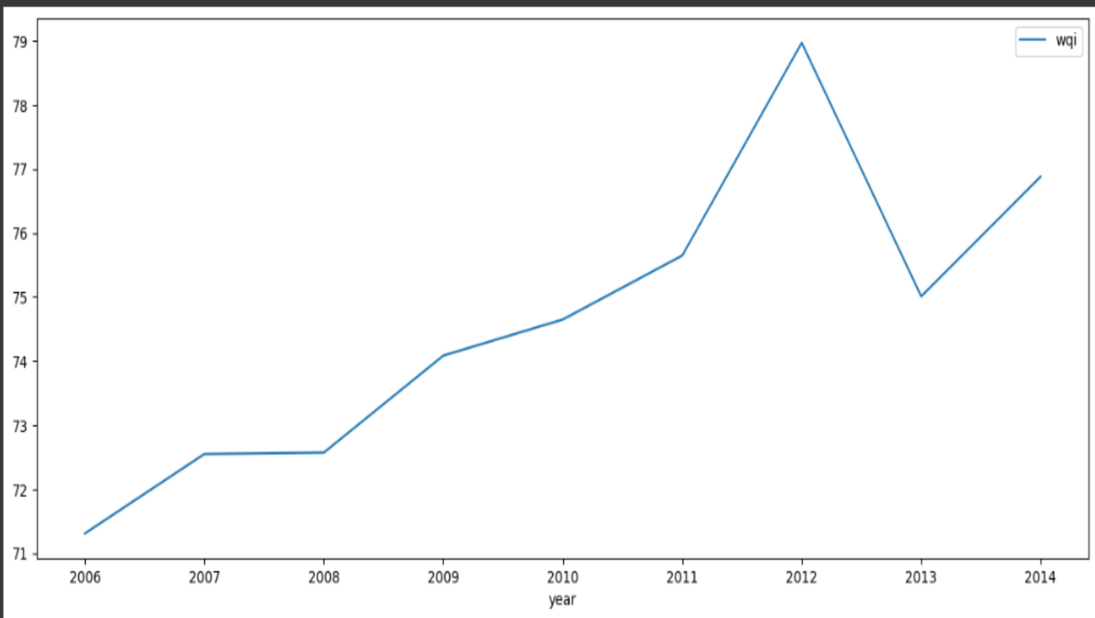
```
data = data[np.isfinite(data['wqi'])]
data.head()
```

	year	wqi
0	2006	71.308824
1	2007	72.549000
2	2008	72.570943
3	2009	74.085193
4	2010	74.648723

```
#scatter plot of data points
cols =['year']
y = data['wqi']
x=data[cols]

plt.scatter(x,y)
plt.show()
```


```
import matplotlib.pyplot as plt
data=data.set_index('year')
data.plot(figsize=(15,6))
plt.show()
```



```python
from sklearn import neighbors,datasets
data=data.reset_index(level=0,inplace=False)
data
```

	year	wqi
0	2006	71.308824
1	2007	72.549000
2	2008	72.570943
3	2009	74.085193
4	2010	74.648723
5	2011	75.647013
6	2012	78.969041
7	2013	75.009425
8	2014	76.879588

d) Using linear regression to predict.

```
#using linear regression to predict
from sklearn import linear_model
from sklearn.model_selection import train_test_split
cols =['year']
y = data['wqi']
x=data[cols]
reg=linear_model.LinearRegression()
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=4)
reg.fit(x_train,y_train)
```

```
▾ LinearRegression
LinearRegression()
```

```
[ ] a=reg.predict(x_test)
    a
```

```
array([73.86206246, 74.59948715])
```

```
[ ] y_test
```

```
3    74.085193
4    74.648723
Name: wqi, dtype: float64
```

```
[ ] from sklearn.metrics import mean_squared_error
    print('mse:%.2f'%mean_squared_error(y_test,a))
```

```
mse:0.03
```

```
dt = pd.DataFrame({'Actual': y_test, 'Predicted': a})
#using gradient descent to optimize it further
x = (x - x.mean()) / x.std()
x = np.c_[np.ones(x.shape[0]), x]
x
```

```
array([[ 1.        , -1.46059349],
       [ 1.        , -1.09544512],
       [ 1.        , -0.73029674],
       [ 1.        , -0.36514837],
       [ 1.        ,  0.        ],
       [ 1.        ,  0.36514837],
       [ 1.        ,  0.73029674],
       [ 1.        ,  1.09544512],
       [ 1.        ,  1.46059349]])
```

```python
alpha = 0.1 #Step size
iterations = 3000 #No. of iterations
m = y.size #No. of data points
np.random.seed(4) #Setting the seed
theta = np.random.rand(2) #Picking some random values to start with

def gradient_descent(x, y, theta, iterations, alpha):
    past_costs = []
    past_thetas = [theta]
    for i in range(iterations):
        prediction = np.dot(x, theta)
        error = prediction - y
        cost = 1/(2*m) * np.dot(error.T, error)
        past_costs.append(cost)
        theta = theta - (alpha * (1/m) * np.dot(x.T, error))
        past_thetas.append(theta)

    return past_thetas, past_costs

past_thetas, past_costs = gradient_descent(x, y, theta, iterations, alpha)
theta = past_thetas[-1]

#Print the results...
print("Gradient Descent: {:.2f}, {:.2f}".format(theta[0], theta[1]))
```

```
Gradient Descent: 74.63, 2.01
```

```python
plt.title('Cost Function J')
plt.xlabel('No. of iterations')
plt.ylabel('Cost')
plt.plot(past_costs)
plt.show()
```

```
#prediction of january(2013-2015) across india
import numpy as np
newB=[74.76, 2.13]

def rmse(y,y_pred):
    rmse= np.sqrt(sum(y-y_pred))
    return rmse


y_pred=x.dot(newB)

dt = pd.DataFrame({'Actual': y, 'Predicted': y_pred})
dt=pd.concat([data, dt], axis=1)
dt
```

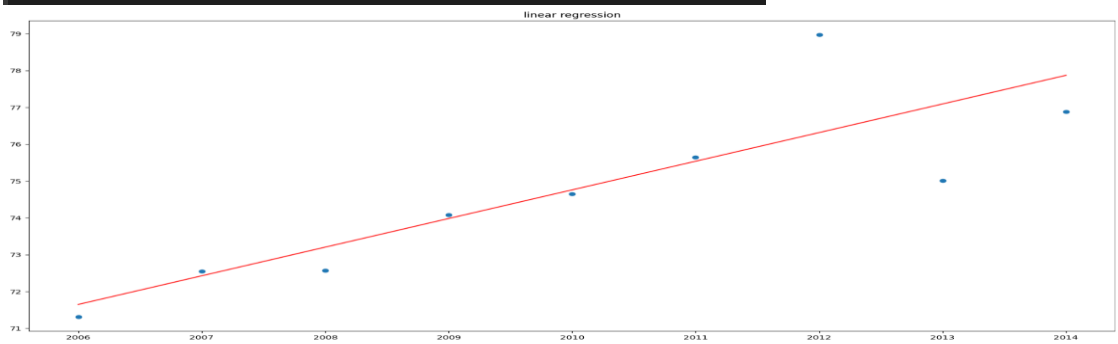
	year	wqi	Actual	Predicted
0	2006	71.308824	71.308824	71.648936
1	2007	72.549000	72.549000	72.426702
2	2008	72.570943	72.570943	73.204468
3	2009	74.085193	74.085193	73.982234
4	2010	74.648723	74.648723	74.760000
5	2011	75.647013	75.647013	75.537766
6	2012	78.969041	78.969041	76.315532
7	2013	75.009425	75.009425	77.093298
8	2014	76.879588	76.879588	77.871064

```
#testing the accuracy of the model

from sklearn import metrics
print(np.sqrt(metrics.mean_squared_error(y,y_pred)))

#1.1987755149740886
#plotting the actual and predicted results
x_axis=dt.year
y_axis=dt.Actual
y1_axis=dt.Predicted
plt.scatter(x_axis,y_axis)
plt.plot(x_axis,y1_axis,color='r')
plt.title("linear regression")

plt.show()
```



WEEK-10

Write a program to predict the winning team in IPL matches.

#IMPORT THE DATASET

from google.colab import files

uploaded=files.upload()

IMPORTING ALL REQUIRED LIBRARIES

import pandas as pd

import numpy as np

import matplotlib

import matplotlib.pyplot as plt

import seaborn as sns

#importing the csv file

import pandas as pd

data=pd.read_csv("matches.csv")

data.head()

a) Shows the important information from the dataset

```
# Let's get a brief summary of the IPL dataset.

data.describe()
```

	id	season	dl_applied	win_by_runs	win_by_wickets
count	756.000000	756.000000	756.000000	756.000000	756.000000
mean	1792.178571	2013.444444	0.025132	13.283069	3.350529
std	3464.478148	3.366895	0.156630	23.471144	3.387963
min	1.000000	2008.000000	0.000000	0.000000	0.000000
25%	189.750000	2011.000000	0.000000	0.000000	0.000000
50%	378.500000	2013.000000	0.000000	0.000000	4.000000
75%	567.250000	2016.000000	0.000000	19.000000	6.000000
max	11415.000000	2019.000000	1.000000	146.000000	10.000000

b) Checking whether there are any null values present in the dataset.

```
# Checking whether there are any null values present in the dataset.

data.isnull().sum()
```

```
id                   0
season               0
city                 7
date                 0
team1                0
team2                0
toss_winner          0
toss_decision        0
result               0
dl_applied           0
winner               4
win_by_runs          0
win_by_wickets       0
player_of_match      4
venue                0
umpire1              2
umpire2              2
umpire3            637
dtype: int64
```

c)information about IPL Matches

```
# information about IPL Matches
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 756 entries, 0 to 755
Data columns (total 18 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   id               756 non-null    int64
 1   season           756 non-null    int64
 2   city             749 non-null    object
 3   date             756 non-null    object
 4   team1            756 non-null    object
 5   team2            756 non-null    object
 6   toss_winner      756 non-null    object
 7   toss_decision    756 non-null    object
 8   result           756 non-null    object
 9   dl_applied       756 non-null    int64
 10  winner           752 non-null    object
 11  win_by_runs      756 non-null    int64
 12  win_by_wickets   756 non-null    int64
 13  player_of_match  752 non-null    object
 14  venue            756 non-null    object
 15  umpire1          754 non-null    object
 16  umpire2          754 non-null    object
 17  umpire3          119 non-null    object
dtypes: int64(5), object(13)
memory usage: 106.4+ KB
```

d) Delete umpire Column from Data set.

```
# As you can see, the values of umpire3 are null in almost all rows, so we are dropping the column umpire3.
# And also dropped some rows containing the null values after removing the umpire3 column.
data = data.iloc[:,:-1]
data.dropna(inplace=True)

# Now let's look into the total teams listed in this dataset.
data["team1"].unique()
```

```
array(['Sunrisers Hyderabad', 'Mumbai Indians', 'Gujarat Lions',
       'Rising Pune Supergiant', 'Kolkata Knight Riders',
       'Royal Challengers Bangalore', 'Delhi Daredevils',
       'Kings XI Punjab', 'Chennai Super Kings', 'Rajasthan Royals',
       'Deccan Chargers', 'Kochi Tuskers Kerala', 'Pune Warriors',
       'Rising Pune Supergiants', 'Delhi Capitals'], dtype=object)
```

e) Visualize the Number of IPL matches won by each team.

Visualizations

Number of IPL matches won by each team.

plt.figure(figsize = (10,6))

sns.countplot(y = 'winner',data = data,order= data['winner'].value_counts().index)

plt.xlabel('Wins')

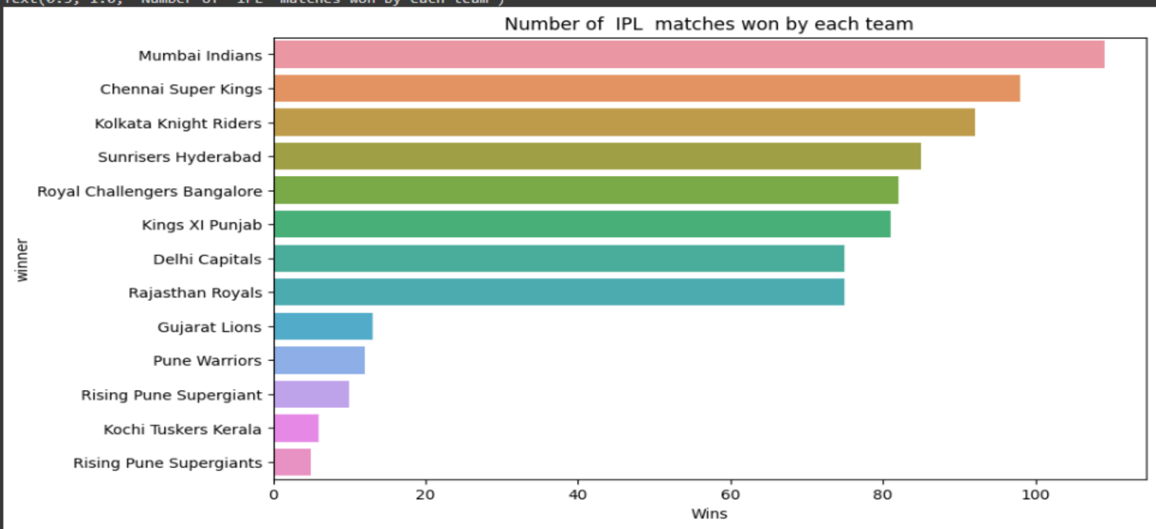
plt.title('Number of IPL matches won by each team')

```
# Visualizations
# Number of IPL matches won by each team.

plt.figure(figsize = (10,6))
sns.countplot(y = 'winner',data = data,order= data['winner'].value_counts().index)
plt.xlabel('Wins')

plt.title('Number of  IPL  matches won by each team')
```

```
Text(0.5, 1.0, 'Number of  IPL  matches won by each team')
```



f) Choose Suitable Model to predict the winning team in IPL matches

The output data is also a categorical value, so we are converting it into numerical using LabelEncoder of sklearn.

from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()

y = le.fit_transform(x)

Now let's convert our data into a training set in order to create the model and test set for evaluating the created model.

from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(x, y, train_size = 0.8)

```
teams_per_season = data.groupby('season')['winner'].value_counts()
teams_per_season
```

```
season  winner
2008    Rajasthan Royals               13
        Kings XI Punjab                10
        Chennai Super Kings             9
        Delhi Capitals                  7
        Mumbai Indians                  7
                                       ..
2019    Kings XI Punjab                 6
        Kolkata Knight Riders           6
        Sunrisers Hyderabad             6
        Rajasthan Royals                5
        Royal Challengers Bangalore     5
Name: winner, Length: 100, dtype: int64
```

```
[23] year = 2008
     winteamseason = pd.DataFrame(columns=['year', 'team', 'wins'])
     winteamseason
```

year	team	wins

```python
for items in teams_per_season.iteritems():
 if items[0][0]==year:
   print(items)
   win_series = pd.DataFrame({
       'year': [items[0][0]],
        'team': [items[0][1]],
        'wins': [items[1]]
        })
   winteamseason = winteamseason.append(win_series)
   year += 1
print(winteamseason)
```

```
((2008, 'Rajasthan Royals'), 13)
((2009, 'Delhi Capitals'), 10)
((2010, 'Mumbai Indians'), 11)
((2011, 'Chennai Super Kings'), 11)
((2012, 'Kolkata Knight Riders'), 12)
((2013, 'Mumbai Indians'), 13)
((2014, 'Kings XI Punjab'), 11)
((2015, 'Chennai Super Kings'), 10)
((2016, 'Sunrisers Hyderabad'), 11)
((2017, 'Mumbai Indians'), 12)
((2018, 'Chennai Super Kings'), 11)
((2019, 'Mumbai Indians'), 11)
   year                   team wins
0  2008       Rajasthan Royals   13
0  2009         Delhi Capitals   10
0  2010         Mumbai Indians   11
0  2011    Chennai Super Kings   11
0  2012  Kolkata Knight Riders   12
0  2013         Mumbai Indians   13
0  2014        Kings XI Punjab   11
0  2015    Chennai Super Kings   10
0  2016    Sunrisers Hyderabad   11
0  2017         Mumbai Indians   12
0  2018    Chennai Super Kings   11
0  2019         Mumbai Indians   11
<ipython-input-24-d7e8bacee7de>:1: FutureWarning: iteritems is
  for items in teams_per_season.iteritems():
```