```
import pandas as pd
from google.colab import files
```

```
uploaded = files.upload()
```

Choose Files No file chosen

```
uploaded
```

```
{}
```

```
import pandas as pd
import numpy as np

df = pd.read_csv('IRIS.csv',header= 0,
                 encoding= 'unicode_escape')
df.head()
```

	sepal_length	sepal_width	petal_length	petal_width	outcome
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

```
df.shape
```

```
(150, 5)
```

```
df ['outcome'].unique()
```

```
array(['Iris-setosa', 'Iris-versicolor', 'Iris-virginica'], dtype=object)
```

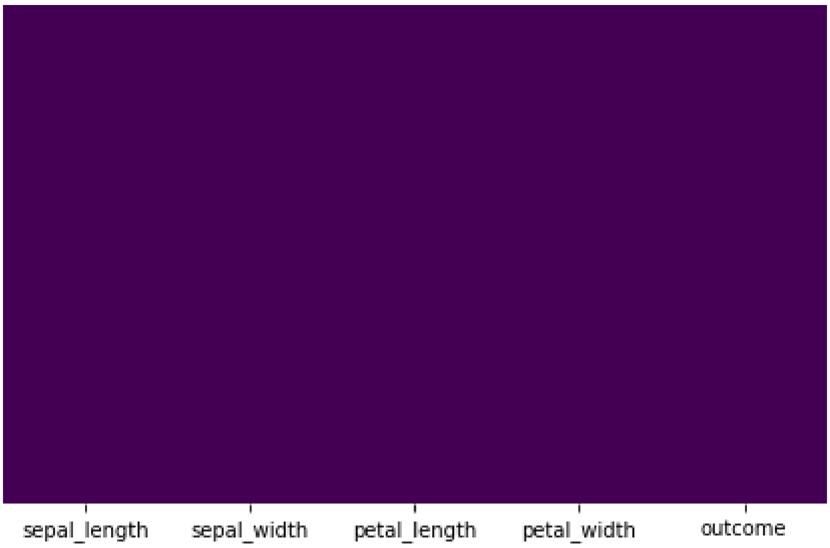
```
df.dtypes
```

```
sepal_length    float64
sepal_width     float64
petal_length    float64
petal_width     float64
outcome          object
dtype: object
```

```
df.tail()
```

	sepal_length	sepal_width	petal_length	petal_width	outcome
145	6.7	3.0	5.2	2.3	Iris-virginica
146	6.3	2.5	5.0	1.9	Iris-virginica
147	6.5	3.0	5.2	2.0	Iris-virginica
148	6.2	3.4	5.4	2.3	Iris-virginica
149	5.9	3.0	5.1	1.8	Iris-virginica

```
import matplotlib.pyplot as plt
import seaborn as sns
def get_heatmap(df):
#this function gives heatmap of all nan values
 plt.figure(figsize=(6,4))
 sns.heatmap(df.isnull(), yticklabels=False, cbar=False, cmap='viridis')
 plt.tight_layout()
 return plt.show()
get_heatmap(df)
```




```
#data preprocessing
from sklearn import preprocessing
#label encoding
LE= preprocessing.LabelEncoder()

df.outcome= LE.fit_transform(df.outcome)

df.head()
```

	sepal_length	sepal_width	petal_length	petal_width	outcome

```
#data preprocessing
from sklearn import preprocessing
#label encoding
LE= preprocessing.LabelEncoder()

df.petal_width  = LE.fit_transform(df.petal_width)

df.head()
```

	sepal_length	sepal_width	petal_length	petal_width	outcome
0	5.1	3.5	1.4	1	0
1	4.9	3.0	1.4	1	0
2	4.7	3.2	1.3	1	0
3	4.6	3.1	1.5	1	0
4	5.0	3.6	1.4	1	0

```
df.columns
```

```
Index(['sepal_length', 'sepal_width', 'petal_length', 'petal_width',
       'outcome'],
      dtype='object')
```

```
#data preprocessing
from sklearn import preprocessing
#label encoding
LE= preprocessing.LabelEncoder()

df['sepal_length']= LE.fit_transform(df['sepal_length'])
df['sepal_length'].unique()
df.head()
```

	sepal_length	sepal_width	petal_length	petal_width	outcome
0	8	3.5	1.4	1	0
1	6	3.0	1.4	1	0
2	4	3.2	1.3	1	0
3	3	3.1	1.5	1	0
4	7	3.6	1.4	1	0

```
#data preprocessing
from sklearn import preprocessing
#label encoding
LE= preprocessing.LabelEncoder()
```

```
df['petal_length']= LE.fit_transform(df['petal_length'])
df['petal_length'].unique()
df.head()
```

	sepal_length	sepal_width	petal_length	petal_width	outcome
0	8	3.5	4	1	0
1	6	3.0	4	1	0
2	4	3.2	3	1	0
3	3	3.1	5	1	0
4	7	3.6	4	1	0

```
#data preprocessing
from sklearn import preprocessing
#label encoding
LE= preprocessing.LabelEncoder()

df['sepal_width']= LE.fit_transform(df['sepal_width'])
df['sepal_width'].unique()
df.head()
```

	sepal_length	sepal_width	petal_length	petal_width	outcome
0	8	14	4	1	0
1	6	9	4	1	0
2	4	11	3	1	0
3	3	10	5	1	0
4	7	15	4	1	0

```
# Importing StandardScaler from scikit-learn
from sklearn.preprocessing import StandardScaler
sst = StandardScaler()

# Standardizing the data apart from the Class column
data_scaled=df.iloc[:,:-1].values

data_scaled=sst.fit_transform(data_scaled)
data_scaled=pd.DataFrame(data_scaled)


data_scaled.head()
```

	0	1	2	3
0	-0.906512	1.040637	-1.223108	-1.250977
1	-1.151958	-0.125996	-1.223108	-1.250977

```
data_scaled.columns=['sepal_length','sepal_width','petal_length','petal_width']
data_scaled.head()
```

	sepal_length	sepal_width	petal_length	petal_width
0	-0.906512	1.040637	-1.223108	-1.250977
1	-1.151958	-0.125996	-1.223108	-1.250977
2	-1.397404	0.340657	-1.309243	-1.250977
3	-1.520126	0.107330	-1.136974	-1.250977
4	-1.029235	1.273963	-1.223108	-1.250977

```
data_scaled['Class']= df.outcome
```

```
data_scaled = data_scaled[data_scaled["Class"].notna()]
```

```
data_scaled
```

	sepal_length	sepal_width	petal_length	petal_width	Class
0	-0.906512	1.040637	-1.223108	-1.250977	0
1	-1.151958	-0.125996	-1.223108	-1.250977	0
2	-1.397404	0.340657	-1.309243	-1.250977	0
3	-1.520126	0.107330	-1.136974	-1.250977	0
4	-1.029235	1.273963	-1.223108	-1.250977	0
...	...	...	...	...	...
145	1.057052	-0.125996	0.844117	1.568421	2
146	0.566161	-1.292630	0.671848	0.941888	2
147	0.811607	-0.125996	0.844117	1.098521	2
148	0.443438	0.807310	1.016386	1.568421	2
149	0.075270	-0.125996	0.757983	0.785255	2

150 rows × 5 columns

```
#Loading the data
X = data_scaled.iloc[:,0:4]
Y = data_scaled.iloc[:,4:5]
```

```
#Splitting the dataset
#Splitting the dataset into Train & Test Dataset
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,Y,test_size=0.1,random_state=3)
```

```
X_train
```

	sepal_length	sepal_width	petal_length	petal_width
40	-1.029235	1.040637	-1.309243	-1.094344
72	0.566161	-1.292630	0.585714	0.315355
135	2.161558	-0.125996	1.619326	1.568421
113	-0.170176	-1.292630	0.671848	1.098521
42	-1.765572	0.340657	-1.309243	-1.250977
...	...	...	...	...
107	1.793389	-0.359323	1.705461	0.785255
21	-0.906512	1.507290	-1.136974	-0.937711
0	-0.906512	1.040637	-1.223108	-1.250977
131	2.284280	1.740617	1.791595	1.098521
106	-1.151958	-1.292630	0.241176	0.628621

135 rows × 4 columns

```
y_train
```

	Class
40	0
72	1
135	2
113	2
42	0
...	...
107	2
21	0
0	0
131	2
106	2

135 rows × 1 columns

```
from sklearn.linear_model import LogisticRegression
clf=LogisticRegression()
clf.fit(X_train,y_train)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:993: DataConversi
  y = column_or_1d(y, warn=True)
LogisticRegression()
```

```
y_pred=clf.predict(X_test)
```

```
y_pred
```

```
array([0, 0, 0, 0, 0, 2, 1, 0, 2, 1, 1, 0, 1, 1, 2])
```

```
y_test
```

	Class
47	0
3	0
31	0
25	0
15	0
118	2
89	1
6	0
103	2
65	1
88	1
38	0
92	1
53	1
140	2

```
y_train_pred=clf.predict(X_train)
```

```
y_train_pred
```

```
array([0, 1, 2, 2, 0, 2, 2, 2, 1, 0, 2, 2, 1, 1, 1, 0, 0, 2, 1, 0, 0, 2,
       0, 2, 1, 2, 1, 0, 0, 2, 1, 0, 2, 2, 1, 0, 0, 2, 1, 1, 0, 2, 0, 2,
       1, 0, 0, 2, 2, 0, 0, 1, 2, 2, 0, 2, 1, 0, 0, 2, 2, 2, 1, 1, 1, 0,
       0, 2, 2, 1, 2, 1, 2, 0, 1, 0, 1, 1, 2, 2, 0, 1, 0, 1, 1, 1, 0, 2,
       0, 2, 1, 2, 1, 2, 1, 0, 2, 1, 2, 1, 0, 1, 2, 0, 1, 0, 0, 0, 1, 2,
       0, 0, 2, 0, 1, 2, 1, 2, 2, 1, 1, 2, 1, 0, 1, 1, 0, 1, 2, 2, 2, 0,
       0, 2, 2])
```

```
y_actual=np.array(y_train)
y_actual.flatten()
```

```
array([0, 1, 2, 2, 0, 2, 2, 2, 1, 0, 2, 2, 1, 1, 1, 0, 0, 2, 1, 0, 0, 1,
       0, 2, 1, 2, 1, 0, 0, 2, 1, 0, 1, 2, 1, 0, 0, 2, 1, 1, 0, 2, 0, 2,
       1, 0, 0, 2, 1, 0, 0, 1, 2, 2, 0, 2, 1, 0, 0, 2, 2, 2, 1, 1, 1, 0,
       0, 2, 2, 1, 2, 1, 2, 0, 2, 0, 1, 1, 2, 2, 0, 1, 0, 1, 1, 1, 0, 2,
       0, 2, 1, 2, 1, 2, 1, 0, 2, 1, 2, 1, 0, 1, 2, 0, 1, 0, 0, 0, 1, 2,
       0, 0, 2, 0, 1, 2, 1, 2, 2, 1, 1, 2, 1, 0, 1, 1, 0, 1, 2, 2, 2, 0,
       0, 2, 2])
```

```
#Evaluating the Model
#Train set results
data = {'y_pred': y_train_pred, 'y_actual': y_actual.flatten()}
```

```
data=pd.DataFrame(data)
data
```

	y_pred	y_actual
0	0	0
1	1	1
2	2	2
3	2	2
4	0	0
...	...	...
130	2	2
131	0	0
132	0	0
133	2	2
134	2	2

135 rows × 2 columns

```
df1=data_scaled
```

```
df1
```

	sepal_length	sepal_width	petal_length	petal_width	Class
0	-0.906512	1.040637	-1.223108	-1.250977	0
1	-1.151958	-0.125996	-1.223108	-1.250977	0
2	-1.397404	0.340657	-1.309243	-1.250977	0
3	-1.520126	0.107330	-1.136974	-1.250977	0
4	-1.029235	1.273963	-1.223108	-1.250977	0
...	...	...	...	...	...
145	1.057052	-0.125996	0.844117	1.568421	2
146	0.566161	-1.292630	0.671848	0.941888	2
147	0.811607	-0.125996	0.844117	1.098521	2
148	0.443438	0.807310	1.016386	1.568421	2
149	0.075270	-0.125996	0.757983	0.785255	2

150 rows × 5 columns

```
import math
# Initializing all the weights as 0
W0_new = 0
W1_new = 0
W2_new = 0
W3_new = 0
W4_new = 0
W5_new = 0


# Alpha - learning rate
a = 0.03
import numpy as np
# MSE
MSE = np.array([])

#sigmoid function
def sigmoid(output):
    z = 1/(1+math.exp(-output))
    return z
for epoch in range(5):

    p_preds = np.array([])
    p_pred_exps = np.array([])
    error = np.array([])
    error_x1 = np.array([])
    error_x2 = np.array([])
    error_x3 = np.array([])
    error_x4 = np.array([])
    error_x5 = np.array([])
```

```
    p_class = np.array([])

    # Assigning all the weights their new values after an epoch:
    W0 = W0_new
    W1 = W1_new
    W2 = W2_new
    W3 = W3_new
    W4 = W4_new




    # Iterating through the Df and calculating all parameters:
    for row in df1.itertuples():

        #The predicted value:
        p_pred = W0 + W1*row[1]+ W2*row[2] + W3*row[3] + W4*row[4]
        p_preds = np.append(p_preds, p_pred)

        # Predicted value after applying the sigmoid function
        p_pred_exp = sigmoid(p_pred)
        p_pred_exps = np.append(p_pred_exps, p_pred_exp)

        # Bifurcating the predicted class as per its probability to be the default class

        if p_pred_exp > 0.5:
            p_class = np.append(p_class,1.0)
        else:
            p_class = np.append(p_class,0.0)


    # The error in prediction
    error = p_pred_exps - df1.Class

    # Pre-calculating the error*x values for all the weights:
    error_x1 = error*df1['sepal_length']
    error_x2 = error*df1['sepal_width']
    error_x3 = error*df1['petal_length']
    error_x4 = error*df1['petal_width']




    # Calculating MSE
    MSE_val = (error).mean()
    MSE = np.append(MSE,MSE_val)

    # Updating the weights
    W0_new = W0 - a*np.sum(error)
    W1_new = W1 - a*np.sum(error_x1)
    W2_new = W2 - a*np.sum(error_x2)
    W3_new = W3 - a*np.sum(error_x3)
    W4_new = W4 - a*np.sum(error_x4)
```

```
    # Adding the predicted class as a separate column to check for performance:
df1['pred_class']=p_class

# Check if any class has been mis classified
```

```
df.dtypes
```

```
sepal_length    int64
sepal_width     int64
petal_length    int64
petal_width     int64
outcome         int64
dtype: object
```

```
df.columns
```

```
Index(['sepal_length', 'sepal_width', 'petal_length', 'petal_width',
       'outcome'],
      dtype='object')
```

```
# True Positives: - Model Correctly predicts the positive class
print('TP: ',df1.Class[(df1.Class==1) & (df1.pred_class==1)].count())
# False Positives: - positive outcomes that the model predicted incorrectly
print('FP: ',df1.Class[(df1.Class==0) & (df1.pred_class==1)].count())
#True Negatives: - Model Correctly predicts the Negative class
print('TN: ',df1.Class[(df1.Class==0) & (df1.pred_class==0)].count())
#False Negatives: - negative outcomes that the model predicted incorrectly
print('FN: ',df1.Class[(df1.Class==1) & (df1.pred_class==0)].count())
```

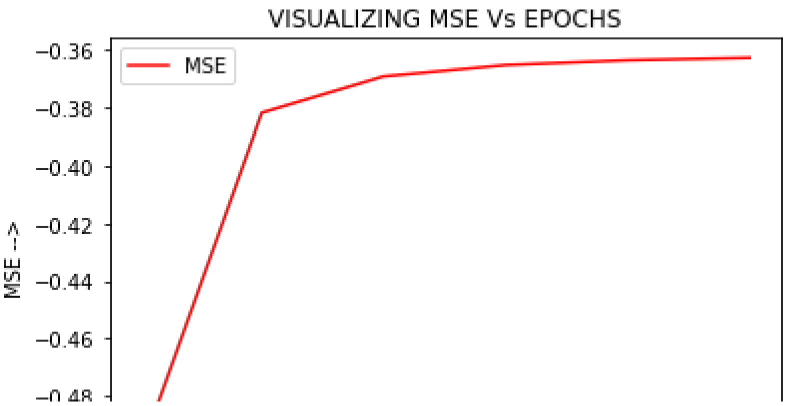
```
TP:  46
FP:  0
TN:  50
FN:  4
```

```
import matplotlib.pyplot as plt

plt.plot(MSE,label='MSE',color='red')

# Add labels and title
plt.title("VISUALIZING MSE Vs EPOCHS")
plt.xlabel("EPOCHS -->")
plt.ylabel("MSE -->")

plt.legend()
plt.show()
```



Double-click (or enter) to edit

✓ 0s completed at 11:31 PM