In [131…

```python
#importing the libraries
import pandas as pd
import numpy as np
from sklearn import preprocessing
from sklearn.preprocessing import LabelEncoder
import seaborn as sn
import matplotlib.pyplot as plt
#add headers to the dataset
headers=[]
for i in range(1,35):
    column="column_"+str(i)
    headers.append(column)
headers.append("output")
#print(headers)
ionospher_df=pd.read_csv('./ionosphere.data',names=headers)
```

In [131…

```python
#see top 5 data rows
ionospher_df.head()
```

Out[1317]:

	column_1	column_2	column_3	column_4	column_5	column_6	column_7	column_8	column_9	colum
0	1	0	0.99539	-0.05889	0.85243	0.02306	0.83398	-0.37708	1.00000	0.0
1	1	0	1.00000	-0.18829	0.93035	-0.36156	-0.10868	-0.93597	1.00000	-0.0
2	1	0	1.00000	-0.03365	1.00000	0.00485	1.00000	-0.12062	0.88965	0.0
3	1	0	1.00000	-0.45161	1.00000	1.00000	0.71216	-1.00000	0.00000	0.0
4	1	0	1.00000	-0.02401	0.94140	0.06531	0.92106	-0.23255	0.77152	-0.1

5 rows × 35 columns

In [131…

```python
#to see the size of the data
ionospher_df.shape[0],ionospher_df.shape[1]
```

Out[1318]:

```
(351, 35)
```

In [131…

```python
ionospher_df.index[ionospher_df.duplicated()]
```

Out[1319]:

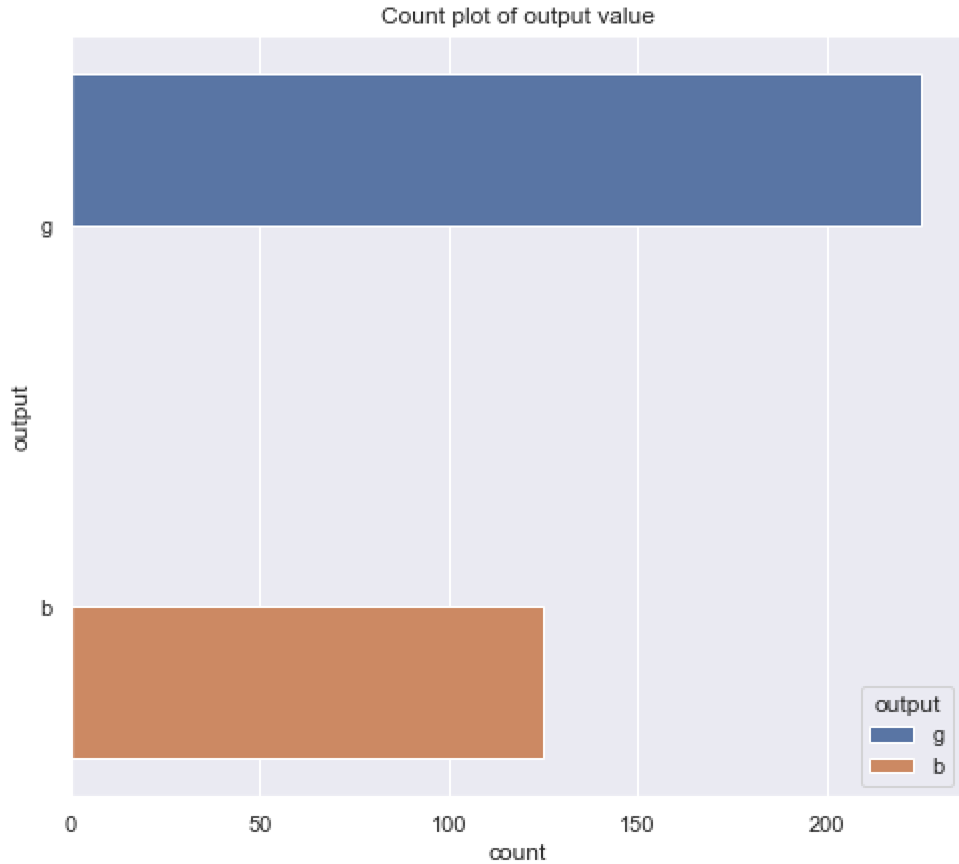
```
Int64Index([248], dtype='int64')
```

In [132…

```python
ionospher_df= ionospher_df.drop_duplicates()
```

In [132…

```python
plt.figure(figsize=(8,7))
sn.countplot(y=ionospher_df["output"],hue = ionospher_df["output"])
plt.title('Count plot of output value')
plt.show()
```



In [132…

```
ionospher_df.shape[0],ionospher_df.shape[1]
```

Out[1322]:

```
(350, 35)
```

In [132…

```
ionospher_df.dtypes
```

Out[1323]:
```
column_1       int64
column_2       int64
column_3     float64
column_4     float64
column_5     float64
column_6     float64
column_7     float64
column_8     float64
column_9     float64
column_10    float64
column_11    float64
column_12    float64
column_13    float64
column_14    float64
column_15    float64
column_16    float64
column_17    float64
column_18    float64
column_19    float64
column_20    float64
column_21    float64
column_22    float64
column_23    float64
column_24    float64
column_25    float64
column_26    float64
column_27    float64
column_28    float64
column_29    float64
column_30    float64
column_31    float64
column_32    float64
column_33    float64
column_34    float64
output        object
dtype: object
```

In [132…
```
ionospher_df.isna().sum()
```

Out[1324]:
```
column_1     0
column_2     0
column_3     0
column_4     0
column_5     0
column_6     0
column_7     0
column_8     0
column_9     0
column_10    0
column_11    0
column_12    0
column_13    0
column_14    0
column_15    0
column_16    0
column_17    0
column_18    0
column_19    0
column_20    0
column_21    0
column_22    0
column_23    0
column_24    0
column_25    0
column_26    0
column_27    0
column_28    0
column_29    0
column_30    0
column_31    0
column_32    0
column_33    0
column_34    0
output       0
dtype: int64
```

In [132…
```
#find and replace
ionospher_df["output"].value_counts()
set_nums = {"output":  {"g": 1, "b": 0}}
ionospher_df = ionospher_df.replace(set_nums)
```

In [132…
```
ionospher_df["output"].value_counts()
```

Out[1326]:
```
1    225
0    125
Name: output, dtype: int64
```

In [132…
```
# sort based on the data output column so that to choose training data and testing data
ionospher_df.sort_values('output')
```

Out[1327]:

	column_1	column_2	column_3	column_4	column_5	column_6	column_7	column_8	column_9	colu
174	1	0	0.62121	-0.63636	0.00000	0.00000	0.00000	0.00000	0.34470	
128	1	0	0.00000	0.00000	-0.33672	0.85388	0.00000	0.00000	0.68869	-
130	1	0	0.00000	0.00000	0.98919	-0.22703	0.18919	-0.05405	0.00000	
132	1	0	1.00000	1.00000	1.00000	-1.00000	0.00000	0.00000	0.00000	
134	1	0	0.00000	0.00000	1.00000	-1.00000	0.00000	0.00000	0.00000	
...	...	...	...	...	...	...	...	...	...	
165	1	0	1.00000	0.54902	0.62745	1.00000	0.01961	1.00000	-0.49020	
167	1	0	0.44444	0.44444	0.53695	0.90763	-0.22222	1.00000	-0.33333	
169	1	0	1.00000	0.00000	1.00000	0.00000	0.50000	0.50000	0.75000	
145	1	0	0.25000	0.16667	0.46667	0.26667	0.19036	0.23966	0.07766	
350	1	0	0.84710	0.13533	0.73638	-0.06151	0.87873	0.08260	0.88928	-

350 rows × 35 columns

In [132…

```
ionospher_df["column_2"].value_counts()
```

Out[1328]:

```
0    350
Name: column_2, dtype: int64
```

In [132…

```
ionospher_df.corr()
```

Loading [MathJax]/extensions/Safe.js

Out[1329]:

	column_1	column_2	column_3	column_4	column_5	column_6	column_7	column_8	column_
column_1	1.000000	NaN	0.295648	-0.007442	0.148700	0.127056	0.215631	0.025500	0.18338
column_2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
column_3	0.295648	NaN	1.000000	0.143337	0.474355	0.024901	0.437956	0.007890	0.46969
column_4	-0.007442	NaN	0.143337	1.000000	0.000820	-0.190400	-0.054449	0.254960	-0.30305
column_5	0.148700	NaN	0.474355	0.000820	1.000000	0.037565	0.595580	-0.030614	0.44862
column_6	0.127056	NaN	0.024901	-0.190400	0.037565	1.000000	-0.011052	0.274628	-0.12162
column_7	0.215631	NaN	0.437956	-0.054449	0.595580	-0.011052	1.000000	-0.151439	0.46015
column_8	0.025500	NaN	0.007890	0.254960	-0.030614	0.274628	-0.151439	1.000000	-0.33719
column_9	0.183388	NaN	0.469690	-0.303055	0.448624	-0.121628	0.460153	-0.337194	1.00000
column_10	-0.055630	NaN	0.046653	0.207634	-0.035553	0.199868	-0.091650	0.373424	-0.25345
column_11	0.027553	NaN	0.322995	-0.190531	0.448347	-0.292382	0.411328	-0.364959	0.67003
column_12	0.070487	NaN	0.169251	0.315836	0.041944	0.163745	-0.021439	0.429032	-0.16888
column_13	0.098492	NaN	0.215860	-0.149493	0.481192	-0.307872	0.630501	-0.356537	0.56136
column_14	0.200058	NaN	0.164253	0.236566	0.126841	0.135089	0.083206	0.253648	-0.08966
column_15	0.110646	NaN	0.196907	-0.253406	0.398053	-0.359898	0.615064	-0.352729	0.61808
column_16	0.100392	NaN	0.093955	0.185837	0.087649	0.157648	-0.022029	0.419617	-0.03318
column_17	0.053368	NaN	0.219807	-0.251462	0.276566	-0.317353	0.378643	-0.492575	0.63305
column_18	0.076990	NaN	0.172439	-0.147451	0.027665	0.188095	0.116158	0.068727	0.20110
column_19	0.197961	NaN	0.283971	-0.332540	0.220157	-0.209102	0.371575	-0.401119	0.67313
column_20	0.019845	NaN	0.151332	0.167260	0.042193	-0.061234	0.159350	0.077660	0.06754
column_21	0.171397	NaN	0.147752	-0.281370	0.325159	-0.115425	0.586165	-0.371026	0.49174
column_22	-0.155879	NaN	0.138335	-0.035406	0.163924	-0.132446	0.191095	-0.212034	0.23762
column_23	0.006928	NaN	0.249338	-0.143968	0.502111	-0.216341	0.372123	-0.271179	0.35117
column_24	-0.082672	NaN	-0.012197	0.164233	0.098826	-0.286494	0.113270	0.007117	0.16181
column_25	0.011234	NaN	0.303295	-0.104901	0.241419	-0.178205	0.285260	-0.180512	0.35534
column_26	0.152751	NaN	-0.072861	-0.236957	-0.031853	0.041893	0.088342	-0.132945	0.10804
column_27	-0.198378	NaN	0.081475	-0.046707	0.144280	-0.175007	0.100700	-0.253853	0.17523
column_28	-0.025014	NaN	0.117863	0.000257	0.179995	-0.070289	0.104595	0.071562	0.14271
column_29	0.129852	NaN	0.343061	-0.041306	0.256118	-0.029887	0.299249	-0.140254	0.32859
column_30	-0.122413	NaN	0.058232	0.342323	0.051348	-0.158065	-0.015009	0.078627	-0.03187
column_31	0.163996	NaN	0.245092	-0.172550	0.398778	-0.100748	0.414209	-0.167191	0.31486
column_32	-0.102062	NaN	-0.009327	-0.122788	0.025754	0.316836	-0.008314	0.152397	-0.06757
column_33	0.159461	NaN	0.261666	-0.154258	0.382230	0.016429	0.545065	-0.201443	0.34360
column_34	0.010661	NaN	0.000471	0.034600	-0.099772	0.185210	-0.076696	0.360617	-0.09582
output	0.461280	NaN	0.516765	0.125823	0.514353	0.148530	0.448103	0.207213	0.29216

35 rows × 35 columns

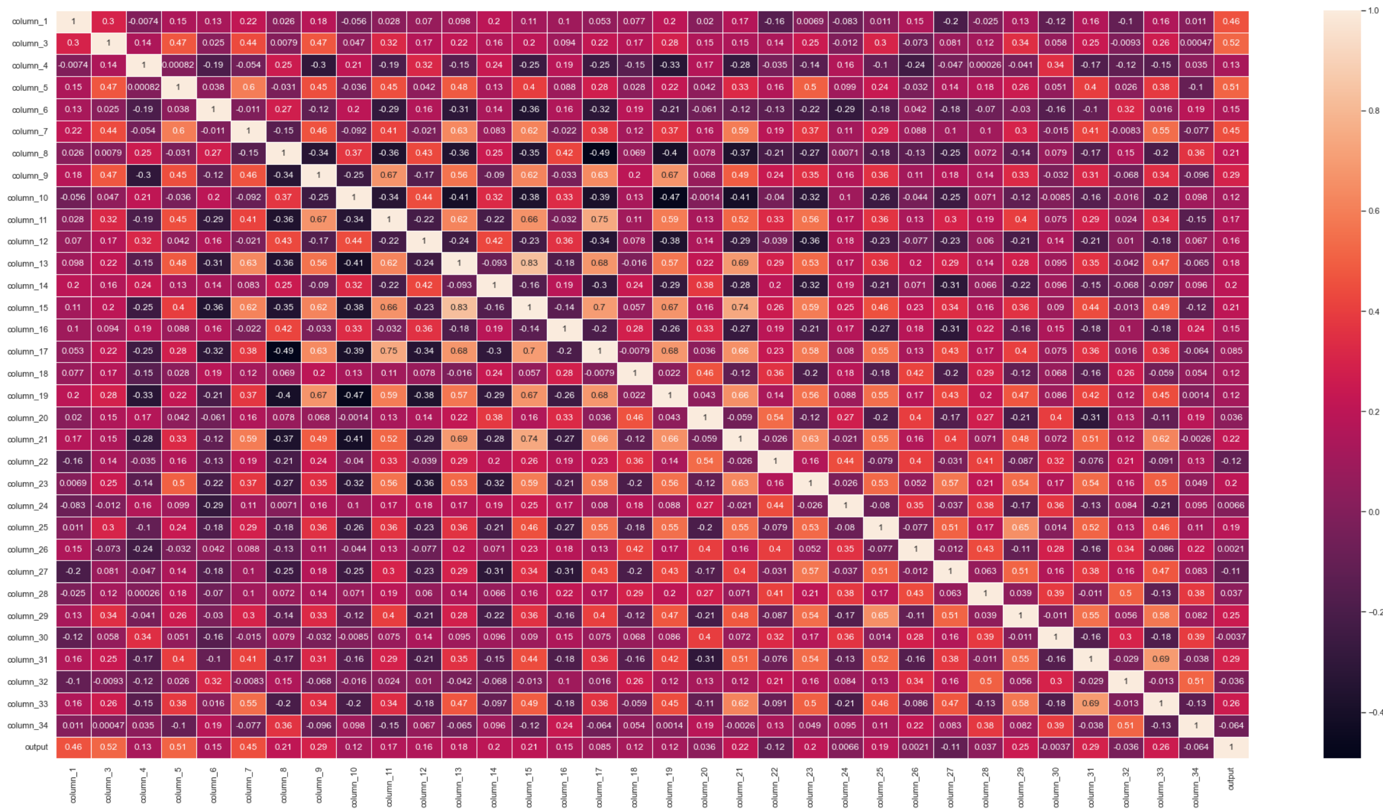
Loading [MathJax]/extensions/Safe.js

In [133…
```
ionospher_df.drop("column_2", axis=1, inplace=True)
```

In [ ]:
```

```

In [133…
```
#plotting heatmap to see the correlation
sn.set(rc = {'figure.figsize':(35,18)})
hm = sn.heatmap(data=ionospher_df.corr(),linewidths=.75,annot=True)
plt.show()
```



In [133…
```
np.corrcoef(ionospher_df['output'],ionospher_df['column_3'])
```

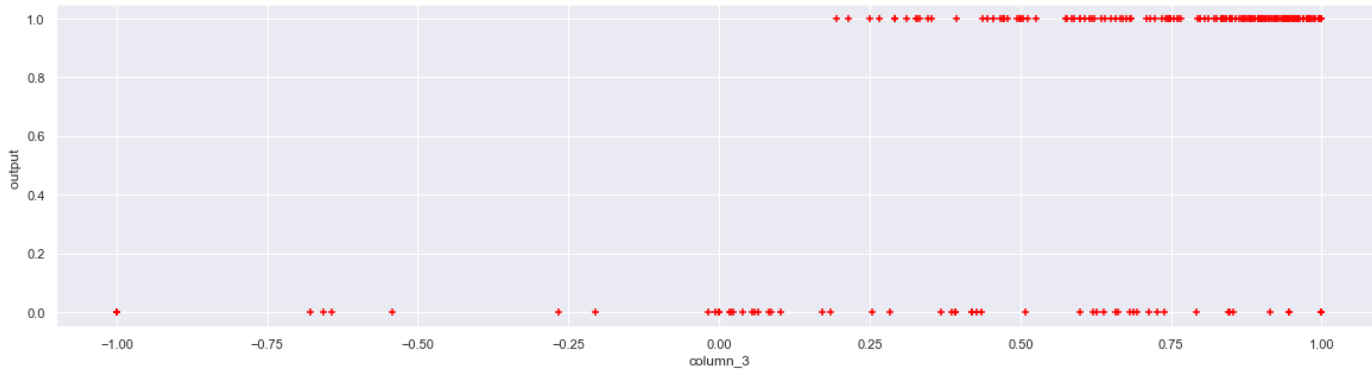
Out[1332]:
```
array([[1.        , 0.51676545],
       [0.51676545, 1.        ]])
```

In [133…
```
np.corrcoef(ionospher_df['output'],ionospher_df['column_5'])
```

Out[1333]:
```
array([[1.        , 0.51435326],
       [0.51435326, 1.        ]])
```

In [133…
```
%matplotlib inline
plt.figure(figsize=(20,5))
plt.xlabel("column_3")
plt.ylabel("output ")
plt.scatter(ionospher_df["column_3"],ionospher_df["output"],color='red',marker='+')
```

Out[1334]:
```
<matplotlib.collections.PathCollection at 0x20305286970>
```



In [133…

```
#min max normalization is used when features are of different scales.

# for i,col in enumerate(ionospher_df,1):
#     ionospher_df[col]=(ionospher_df[col]-ionospher_df[col].min())/(ionospher_df[col].m
```

In [133…

```
ionospher_df
```

Out[1336]:

	column_1	column_3	column_4	column_5	column_6	column_7	column_8	column_9	column_10	co
0	1	0.99539	-0.05889	0.85243	0.02306	0.83398	-0.37708	1.00000	0.03760	
1	1	1.00000	-0.18829	0.93035	-0.36156	-0.10868	-0.93597	1.00000	-0.04549	
2	1	1.00000	-0.03365	1.00000	0.00485	1.00000	-0.12062	0.88965	0.01198	
3	1	1.00000	-0.45161	1.00000	1.00000	0.71216	-1.00000	0.00000	0.00000	
4	1	1.00000	-0.02401	0.94140	0.06531	0.92106	-0.23255	0.77152	-0.16399	
...	...	...	...	...	...	...	...	...	...	
346	1	0.83508	0.08298	0.73739	-0.14706	0.84349	-0.05567	0.90441	-0.04622	
347	1	0.95113	0.00419	0.95183	-0.02723	0.93438	-0.01920	0.94590	0.01606	
348	1	0.94701	-0.00034	0.93207	-0.03227	0.95177	-0.03431	0.95584	0.02446	
349	1	0.90608	-0.01657	0.98122	-0.01989	0.95691	-0.03646	0.85746	0.00110	
350	1	0.84710	0.13533	0.73638	-0.06151	0.87873	0.08260	0.88928	-0.09139	

350 rows × 34 columns

Univariate Logistic regression using built in function

In [133…

```
from sklearn.linear_model import LogisticRegression
```

In [133…

```
ionospher_df_data = ionospher_df.sample(frac=1,random_state=42)

train_size = int(0.8 * len(X))


# Split your dataset
train_set = ionospher_df_data[:train_size]
test_set = ionospher_df_data[train_size:]
```

Loading [MathJax]/extensions/Safe.js

```
= LogisticRegression()
```

```
logisticRegr.fit(np.array(train_set["column_3"].values).reshape(-1,1), train_set.output.
logisticRegr.predict(np.array(train_set["column_3"].values).reshape(-1,1))
```

Out[1339]:
```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0,
       0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0,
       0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1,
       1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0,
       1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1,
       1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0,
       1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1,
       0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0,
       1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1,
       1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0,
       1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1], dtype=int64)
```

In [134…
```
score = logisticRegr.score(np.array(test_set["column_3"].values).reshape(-1,1), test_set
print("Univariate Logistic Regression; Accuracy on test set by sklearn model :",score*10
```

```
Univariate Logistic Regression; Accuracy on test set by sklearn model : 88.5714285714285
7 %
```

Multivariate Logistic regression using built in function

In [134…
```
X = ionospher_df.iloc[:,:-1]
Y = ionospher_df.iloc[:,-1]


X_sample = X.sample(frac=1,random_state=13)
Y_sample = Y.sample(frac=1,random_state=13)

# Define a size for your train set
train_size = int(0.8 * len(X))

# Split your dataset
X_train = X_sample[:train_size]
Y_train = Y_sample[:train_size]

X_test = X_sample[train_size:]
Y_test = Y_sample[train_size:]

model1 = LogisticRegression()
model1.fit( X_train, Y_train)

Y_pred1 =model1.predict( X_test )
```

In [134…
```
#finding accuracy
count = 0
correctly_classified = 0
for count in range( np.size( Y_pred1 ) ) :
    if Y_test.values[count] == Y_pred1[count] :
        correctly_classified = correctly_classified + 1

    count = count + 1
print( "Multivariate Logistic Regression -Accuracy on test set by sklearn model   :  ",
```

```
Multivariate Logistic Regression -Accuracy on test set by sklearn model   :   85.7142857
1428571 %
```

Logistic regression using gradient descent Model

In [134…

```
a=0.004
iterations=8000
cost=0
error_cost=[]
class LR_Model :

    # gradient function
    def gradient_fun( self, X, Y ) :
        # no_of_training_examples, no_of_features
        self.m, self.n = X.shape
        # parameter initialization
        self.W = np.zeros( self.n )
        self.b = 0
        self.X = X
        self.Y = Y
        error_cost.clear()
        # gradient descent updating

        for i in range(iterations ) :
            self.update()
        return self


    def update( self ) :
        #h=p(x)=1/1+e^-(WT.X+b)
        P = 1 / ( 1 + np.exp( - ( self.X.dot( self.W ) + self.b ) ) )
        cost = -(1/m)*np.sum( self.Y*np.log(P) + (1-self.Y)*np.log(1-P))
        error_cost.append(cost)
        # calculate gradients
        tmp = ( P- self.Y.T )
        #tmp = np.reshape( tmp, self.m )
        #dW = (1/m)*(XT(P-Y))
        #dB = (1/m)*sum(P - Y)
        dW = np.dot( self.X.T, tmp ) / self.m
        db = np.sum( tmp ) / self.m

        # parameters
        self.W = self.W - a * dW
        self.b = self.b - a * db
        return self

    def predict( self, X ) :
        #h=p(x)=1/1+e^-(WT.X+b)
        Z = 1 / ( 1 + np.exp( - ( X.dot( self.W ) + self.b ) ) )
        Y = np.where( Z >= 0.5, 1, 0 )
        return Y
```

Multivariate Logistic regression with all features

In [135…

```
#considering all the features
```

Loading [MathJax]/extensions/Safe.js

```python
X = ionospher_df.iloc[:,:-1]
Y = ionospher_df.iloc[:,-1]

# Shuffle the dataset
X_sample = X.sample(frac=1,random_state=13)
Y_sample = Y.sample(frac=1,random_state=13)

# Define a size for your train set
train_size = int(0.8 * len(X))

# Split the dataset
X_train = X_sample[:train_size]
Y_train = Y_sample[:train_size]

X_test = X_sample[train_size:]
Y_test = Y_sample[train_size:]

# Model class
lr = LR_Model()
lr.gradient_fun( X_train, Y_train )


# Prediction on test set for both models
Y_pred = lr.predict( X_test )

# measure performance
correctly_classified = 0
count = 0
for count in range( np.size( Y_pred ) ) :

    if Y_test.values[count] == Y_pred[count] :
        correctly_classified = correctly_classified + 1
    count = count + 1

print( "Accuracy on test set by  Multivariate logistic regression  model with all featur
```

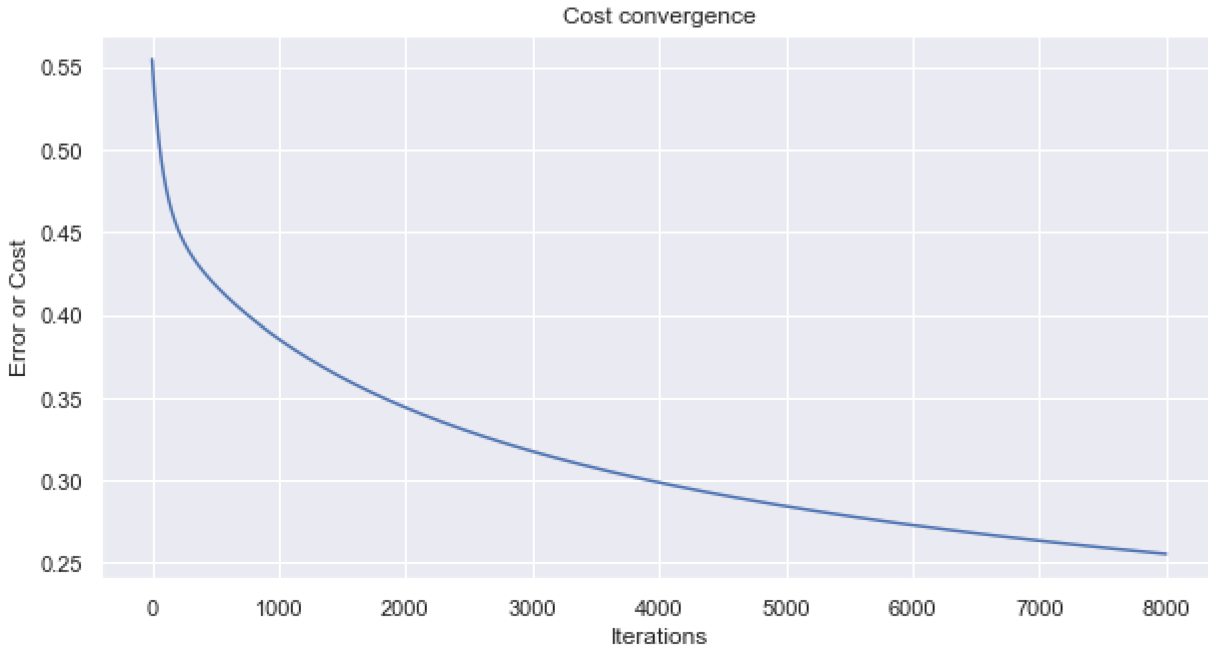
```
Accuracy on test set by  Multivariate logistic regression  model with all features  :
81.42857142857143 %
```

In [134…

```python
plt.figure(figsize=(10,5))
plt.title("Cost convergence")
plt.xlabel("Iterations")
plt.ylabel("Error or Cost")
plt.plot(error_cost)
```

Out[1345]:

```
[<matplotlib.lines.Line2D at 0x2037f12efa0>]
```



Multivariate Logistic regression with some features

In [135…

```
# considering only some of the features
X = ionospher_df.drop(['column_17','column_20','column_22','column_24','column_26','colu

Y = ionospher_df['output']

# Shuffle the dataset
X_sample = X.sample(frac=1,random_state=13)
Y_sample = Y.sample(frac=1,random_state=13)

# Define a size for your train set
train_size = int(0.8 * len(X))

# Split your dataset
X_train = X_sample[:train_size]
Y_train = Y_sample[:train_size]

X_test = X_sample[train_size:]
Y_test = Y_sample[train_size:]

# Model training
lr = LR_Model()
lr.gradient_fun( X_train, Y_train )


# Prediction on test set for both models
Y_pred = lr.predict( X_test )

# measure performance
correctly_classified = 0
count = 0
for count in range( np.size( Y_pred ) ) :

    if Y_test.values[count] == Y_pred[count] :
        ectly_classified = correctly_classified + 1
```

Loading [MathJax]/extensions/Safe.js

```
    count = count + 1

print( "Accuracy on test set by Multivariate logistic regression  model with some featur
```

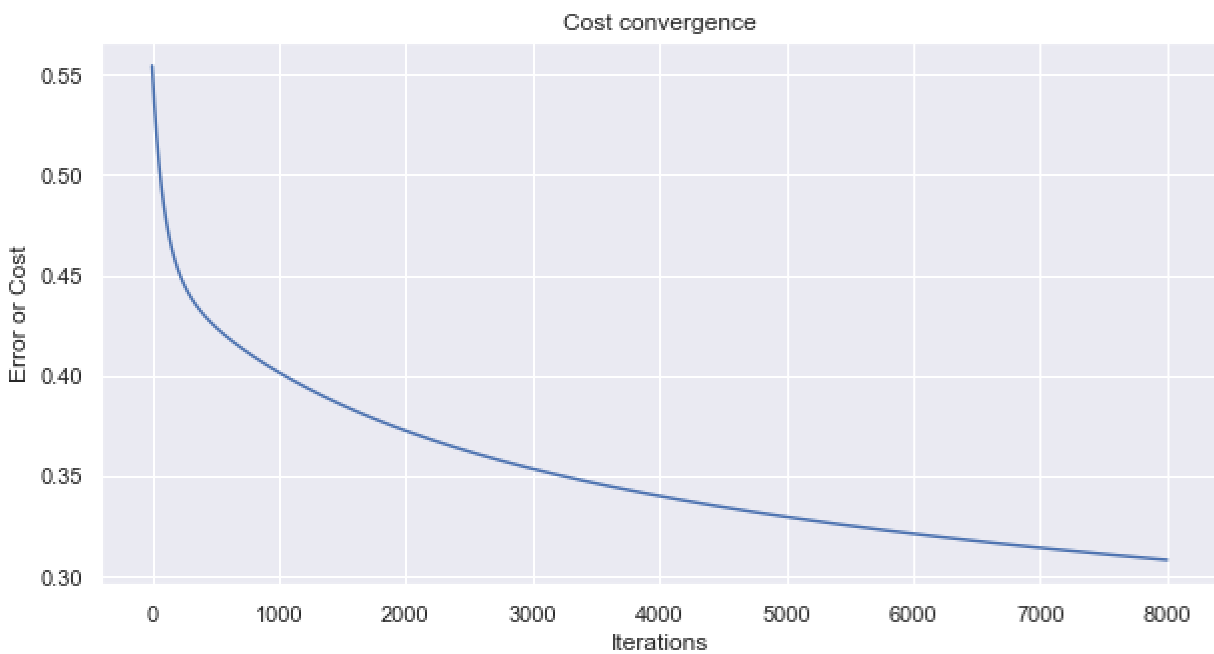
```
Accuracy on test set by Multivariate logistic regression  model with some features
:   82.85714285714286 %
```

In [134…

```
plt.figure(figsize=(10,5))
plt.title("Cost convergence")
plt.xlabel("Iterations")
plt.ylabel("Error or Cost")
plt.plot(error_cost)
```

Out[1347]:

```
[<matplotlib.lines.Line2D at 0x20305ec2d30>]
```



Univariate Logistic regresssion

In [135…

```
#considering one of the features with more correlation
X =  ionospher_df[['column_3']]
Y = ionospher_df['output']


# Shuffle the dataset
X_sample = X.sample(frac=1,random_state=13)
Y_sample = Y.sample(frac=1,random_state=13)

# Define a size for your train set size
train_size = int(0.8 * len(X))

# Split your dataset
X_train = X_sample[:train_size]
Y_train = Y_sample[:train_size]

X_test = X_sample[train_size:]
Y_test = Y_sample[train_size:]
```

Loading [MathJax]/extensions/Safe.js

```
lr = LR_Model()
lr.gradient_fun( X_train, Y_train )


# Prediction on test set for both models
Y_pred = lr.predict( X_test )

# measure performance
correctly_classified = 0
count = 0
for count in range( np.size( Y_pred ) ) :

    if Y_test.values[count] == Y_pred[count] :
        correctly_classified = correctly_classified + 1
    count = count + 1

print( "Accuracy on test set by the univariate logistic regression model       :  ", (co
```

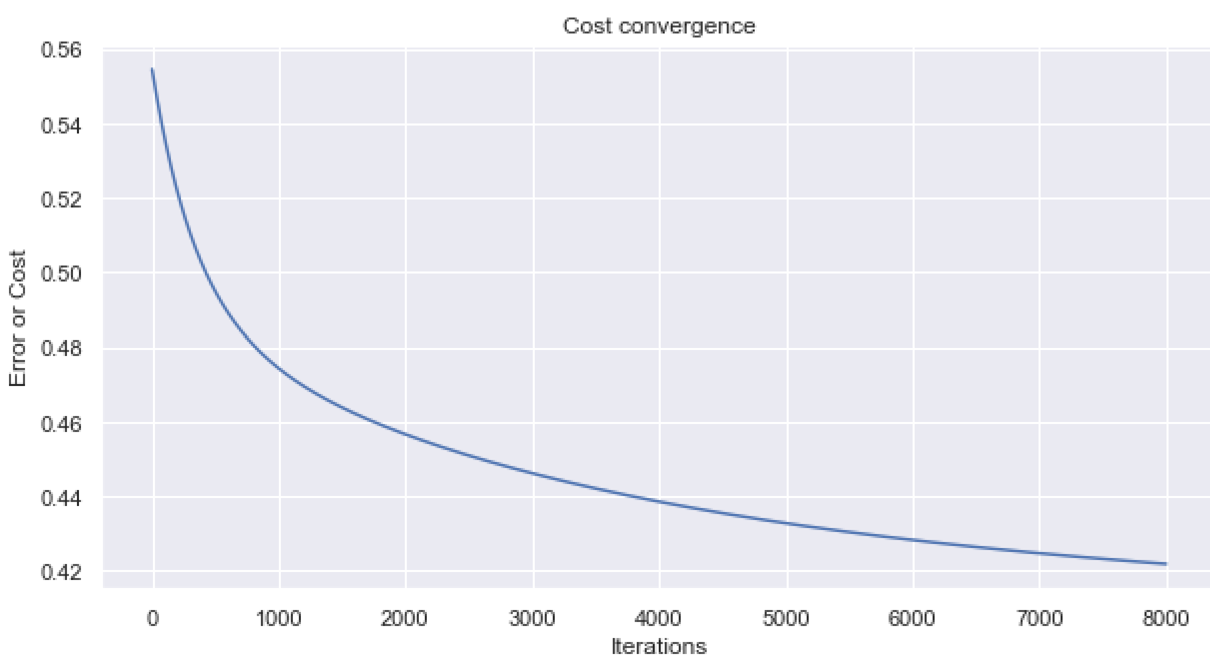
```
Accuracy on test set by the univariate logistic regression model       :   81.4285714285
7143 %
```

In [134…

```
plt.figure(figsize=(10,5))
plt.title("Cost convergence")
plt.xlabel("Iterations")
plt.ylabel("Error or Cost")
plt.plot(error_cost)
```

Out[1349]:

```
[<matplotlib.lines.Line2D at 0x20305f83fa0>]
```



In [ ]: