In [363…

```
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 [364…

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

Out[364]:

	column_1	column_2	column_3	column_4	column_5	column_6	column_7	column_8	column_9	column
0	1	0	0.99539	-0.05889	0.85243	0.02306	0.83398	-0.37708	1.00000	0.03
1	1	0	1.00000	-0.18829	0.93035	-0.36156	-0.10868	-0.93597	1.00000	-0.04
2	1	0	1.00000	-0.03365	1.00000	0.00485	1.00000	-0.12062	0.88965	0.01
3	1	0	1.00000	-0.45161	1.00000	1.00000	0.71216	-1.00000	0.00000	0.00
4	1	0	1.00000	-0.02401	0.94140	0.06531	0.92106	-0.23255	0.77152	-0.16

5 rows × 35 columns

In [365…

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

Out[365]:

```
(351, 35)
```

In [366…

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

Out[366]:

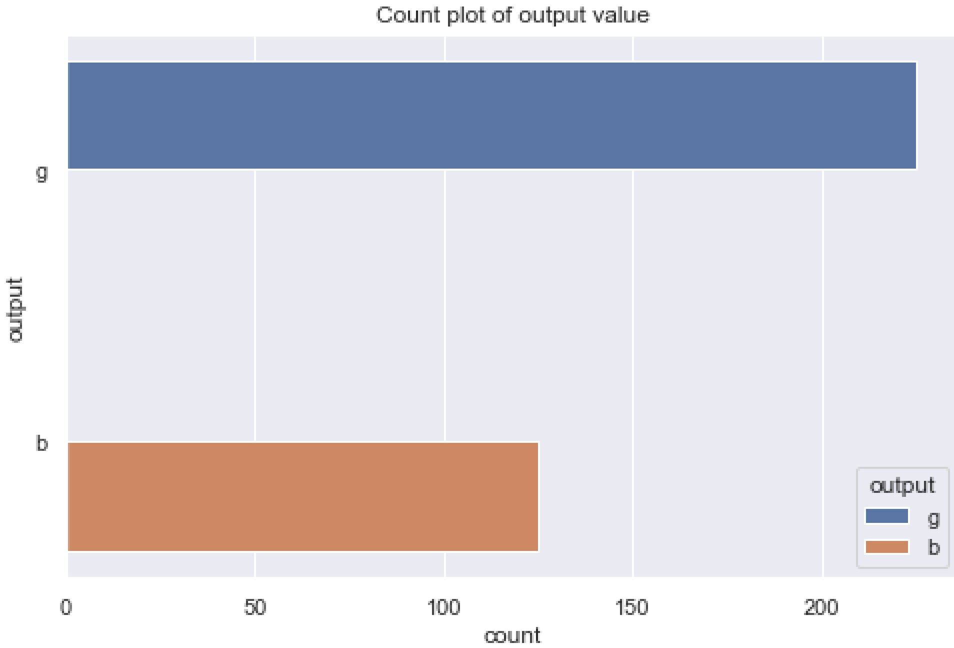
```
Int64Index([248], dtype='int64')
```

In [367…

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

In [368…

```
plt.figure(figsize=(8,5))
sn.countplot(y=ionospher_df["output"],hue = ionospher_df["output"])
plt.title('Count plot of output value')
plt.show()
```



In [369…

```
ionospher_df.dtypes
```

Loading [MathJax]/extensions/Safe.js

Out[369]:
```
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 [370…
```
ionospher_df.isna().sum()
```

Out[370]:
```
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 [371…
```
#find and replace
ionospher_df["output"].value_counts()
set_nums = {"output":  {"g": 1, "b": 0}}
ionospher_df = ionospher_df.replace(set_nums)
```

In [372…
```
ionospher_df["output"].value_counts()
```

Out[372]:
```
1    225
0    125
Name: output, dtype: int64
```

In [373…
```
ionospher_df.corr()
```

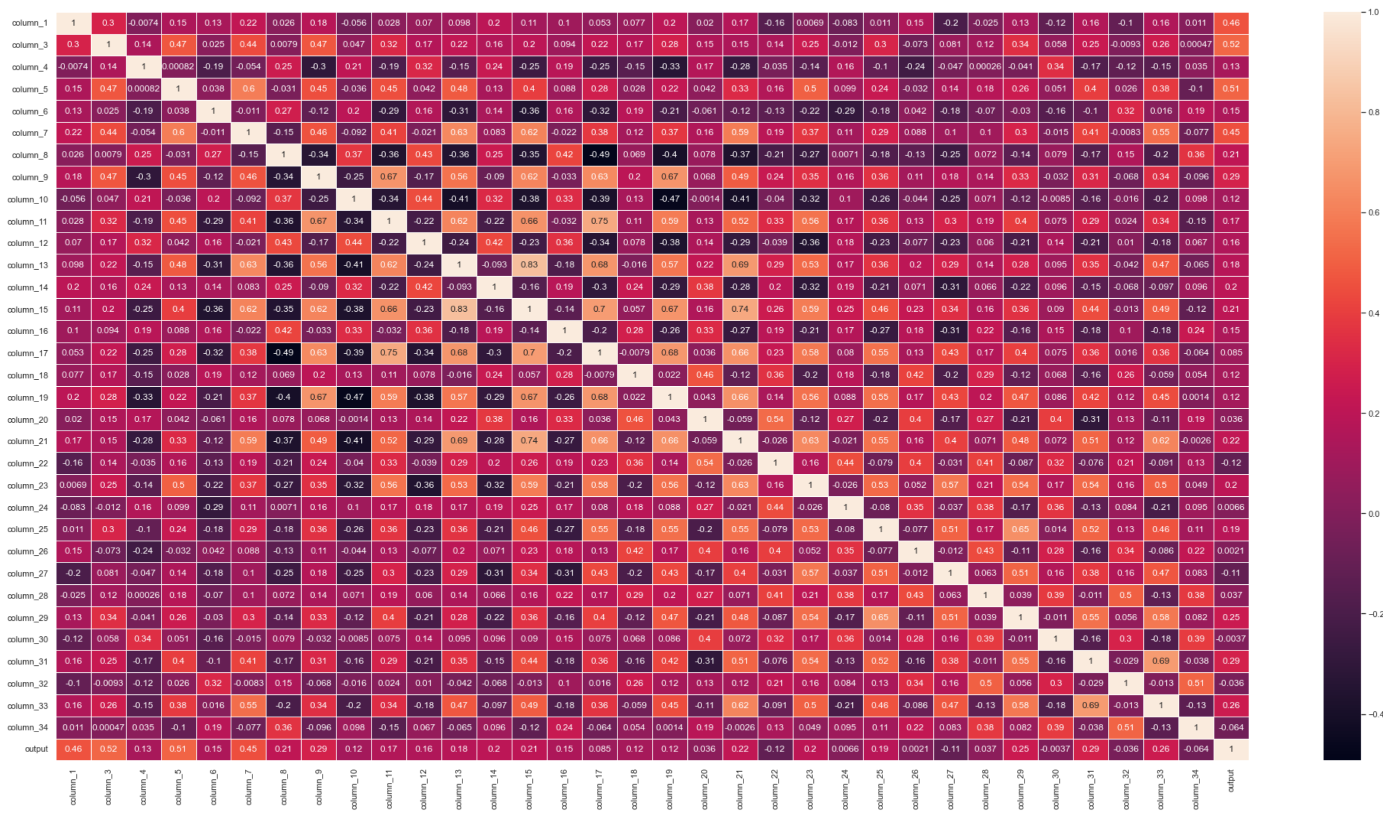
Out[373]:

	column_1	column_2	column_3	column_4	column_5	column_6	column_7	column_8	column_9
column_1	1.000000	NaN	0.295648	-0.007442	0.148700	0.127056	0.215631	0.025500	0.183388
column_2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
column_3	0.295648	NaN	1.000000	0.143337	0.474355	0.024901	0.437956	0.007890	0.469690
column_4	-0.007442	NaN	0.143337	1.000000	0.000820	-0.190400	-0.054449	0.254960	-0.303055
column_5	0.148700	NaN	0.474355	0.000820	1.000000	0.037565	0.595580	-0.030614	0.448624
column_6	0.127056	NaN	0.024901	-0.190400	0.037565	1.000000	-0.011052	0.274628	-0.121628
column_7	0.215631	NaN	0.437956	-0.054449	0.595580	-0.011052	1.000000	-0.151439	0.460153
column_8	0.025500	NaN	0.007890	0.254960	-0.030614	0.274628	-0.151439	1.000000	-0.337194
column_9	0.183388	NaN	0.469690	-0.303055	0.448624	-0.121628	0.460153	-0.337194	1.000000
column_10	-0.055630	NaN	0.046653	0.207634	-0.035553	0.199868	-0.091650	0.373424	-0.253455
column_11	0.027553	NaN	0.322995	-0.190531	0.448347	-0.292382	0.411328	-0.364959	0.670032
column_12	0.070487	NaN	0.169251	0.315836	0.041944	0.163745	-0.021439	0.429032	-0.168882
column_13	0.098492	NaN	0.215860	-0.149493	0.481192	-0.307872	0.630501	-0.356537	0.561362
column_14	0.200058	NaN	0.164253	0.236566	0.126841	0.135089	0.083206	0.253648	-0.089669
column_15	0.110646	NaN	0.196907	-0.253406	0.398053	-0.359898	0.615064	-0.352729	0.618085
column_16	0.100392	NaN	0.093955	0.185837	0.087649	0.157648	-0.022029	0.419617	-0.033187
column_17	0.053368	NaN	0.219807	-0.251462	0.276566	-0.317353	0.378643	-0.492575	0.633058
column_18	0.076990	NaN	0.172439	-0.147451	0.027665	0.188095	0.116158	0.068727	0.201101
column_19	0.197961	NaN	0.283971	-0.332540	0.220157	-0.209102	0.371575	-0.401119	0.673131
column_20	0.019845	NaN	0.151332	0.167260	0.042193	-0.061234	0.159350	0.077660	0.067547
column_21	0.171397	NaN	0.147752	-0.281370	0.325159	-0.115425	0.586165	-0.371026	0.491747
column_22	-0.155879	NaN	0.138335	-0.035406	0.163924	-0.132446	0.191095	-0.212034	0.237623
column_23	0.006928	NaN	0.249338	-0.143968	0.502111	-0.216341	0.372123	-0.271179	0.351176
column_24	-0.082672	NaN	-0.012197	0.164233	0.098826	-0.286494	0.113270	0.007117	0.161812
column_25	0.011234	NaN	0.303295	-0.104901	0.241419	-0.178205	0.285260	-0.180512	0.355341
column_26	0.152751	NaN	-0.072861	-0.236957	-0.031853	0.041893	0.088342	-0.132945	0.108044
column_27	-0.198378	NaN	0.081475	-0.046707	0.144280	-0.175007	0.100700	-0.253853	0.175232
column_28	-0.025014	NaN	0.117863	0.000257	0.179995	-0.070289	0.104595	0.071562	0.142718
column_29	0.129852	NaN	0.343061	-0.041306	0.256118	-0.029887	0.299249	-0.140254	0.328596
column_30	-0.122413	NaN	0.058232	0.342323	0.051348	-0.158065	-0.015009	0.078627	-0.031870
column_31	0.163996	NaN	0.245092	-0.172550	0.398778	-0.100748	0.414209	-0.167191	0.314868
column_32	-0.102062	NaN	-0.009327	-0.122788	0.025754	0.316836	-0.008314	0.152397	-0.067576
column_33	0.159461	NaN	0.261666	-0.154258	0.382230	0.016429	0.545065	-0.201443	0.343602
column_34	0.010661	NaN	0.000471	0.034600	-0.099772	0.185210	-0.076696	0.360617	-0.095826
output	0.461280	NaN	0.516765	0.125823	0.514353	0.148530	0.448103	0.207213	0.292165

35 rows × 35 columns

In [374…
```
ionospher_df.drop("column_2", axis=1, inplace=True)
```

In [375…
```
#plotting heatmap to see the correlation and checking for the dependence of columns
sn.set(rc = {'figure.figsize':(35,18)})
hm = sn.heatmap(data=ionospher_df.corr(),linewidths=.75,annot=True)
plt.show()
```



In [376…
```
print("column_3 :",np.corrcoef(ionospher_df['output'],ionospher_df['column_3']))
print("column_5 :",np.corrcoef(ionospher_df['output'],ionospher_df['column_5']))
```

```
column_3 : [[1.         0.51676545]
 [0.51676545 1.        ]]
column_5 : [[1.         0.51435326]
 [0.51435326 1.        ]]
```

In [377…
```
# for i,col in enumerate(ionospher_df,1):
#     ionospher_df[col]=(ionospher_df[col]-ionospher_df[col].mean())/(ionospher_df[col].
```

In [378…
```
ionospher_df
```

Out[378]:

	column_1	column_3	column_4	column_5	column_6	column_7	column_8	column_9	column_10	colu
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	(

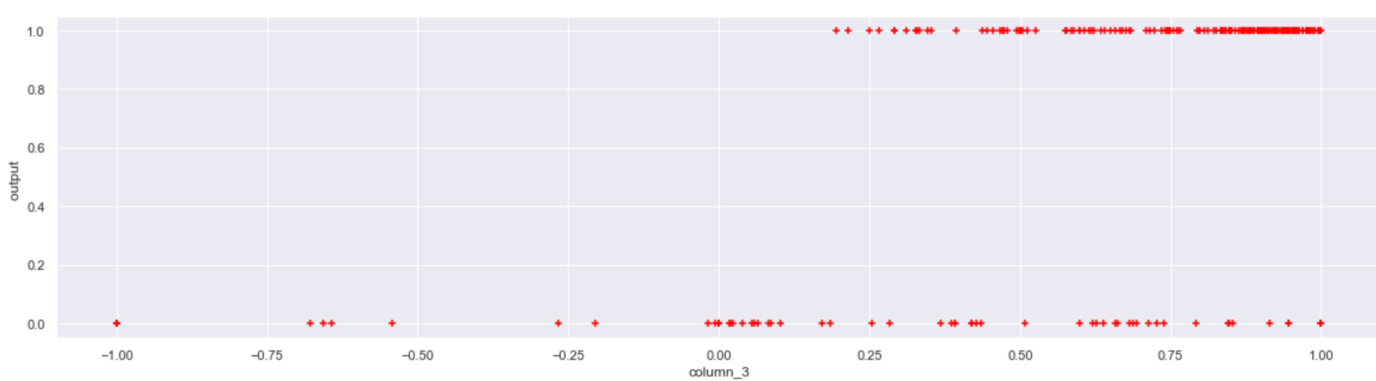
350 rows × 34 columns

In [379…

```
%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[379]:

```
<matplotlib.collections.PathCollection at 0x240ee51f520>
```

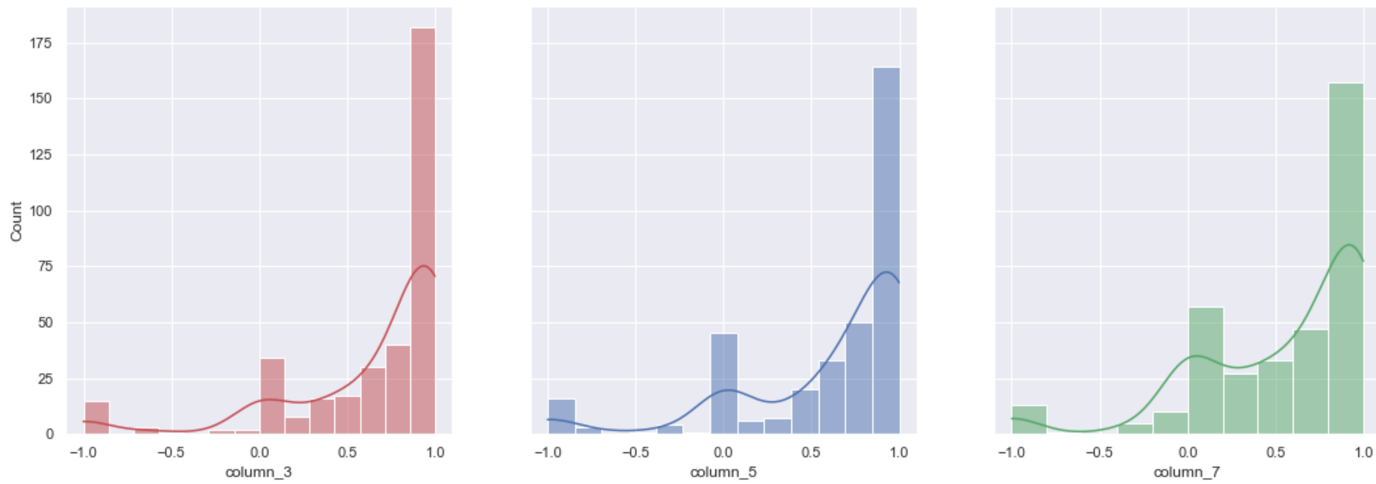


In [380…

```
fig, axes = plt.subplots(1, 3, figsize=(18, 6), sharey=True)
sn.histplot(ionospher_df, ax=axes[0], x="column_3", kde=True, color='r')
sn.histplot(ionospher_df, ax=axes[1], x="column_5", kde=True, color='b')
sn.histplot(ionospher_df, ax=axes[2], x="column_7", kde=True,color='g')
```

Out[380]:

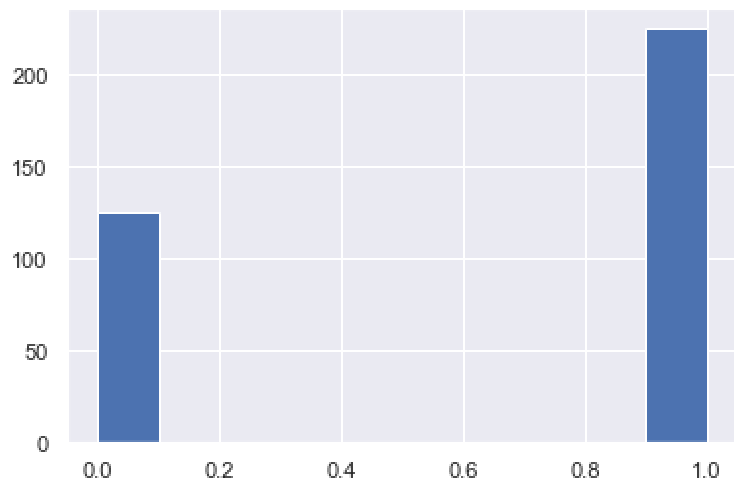
```
<AxesSubplot:xlabel='column_7', ylabel='Count'>
```



In [381…

```
ionospher_df['output'].hist()
```

Out[381]: <AxesSubplot:>



Naive Bayes model using inbuilt model for comaprison

In [ ]:

```
#Naive Bayes model using inbuilt model for univariate
```

In [382…

```
from sklearn.naive_bayes import GaussianNB
gaussian = GaussianNB()
```

In [383…

```
X = ionospher_df[['column_3']]
Y = ionospher_df['output']
data =ionospher_df.sample(frac=1,random_state=13)

# 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
```

Loading [MathJax]/extensions/Safe.js

```
train_size = int(0.8 * len(X))

train_set = data[:train_size]
test_set = data[train_size:]

# 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:]

Y_pred = gaussian.fit(X_train, Y_train).predict(X_test)
```

In [384…

```
print( "Accuracy on test set by sklearn model   : {0}".format((Y_pred == Y_test).sum().a
```

```
Accuracy on test set by sklearn model   : 82.85714285714286
```

In [ ]:

```
#Naive Bayes model using inbuilt model for multivariate
```

In [395…

```

X = ionospher_df.drop('output',axis="columns")
Y = ionospher_df['output']
data =ionospher_df.sample(frac=1,random_state=13)

# 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))

train_set = data[:train_size]
test_set = data[train_size:]

# 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:]

Y_pred = gaussian.fit(X_train, Y_train).predict(X_test)
```

In [396…

```
print( "Accuracy on test set by sklearn model   : {0}".format((Y_pred == Y_test).sum().a
```

```
Accuracy on test set by sklearn model   : 84.28571428571429
```

Naive Bayes model

In [385…

```
def calculate_prior(df, Y):
    classes = sorted(list(df[Y].unique()))
    prior = []
    for i in classes:
```

```
        prior.append(len(df[df[Y]==i])/len(df))
    return prior
```

In [386…

```
def calculate_likelihood(df, feat_name, feat_val, Y, label):
    feat = list(df.columns)
    df = df[df[Y]==label]
    mean, std = df[feat_name].mean(), df[feat_name].std()
    p_x_given_y = (1 / (np.sqrt(2 * np.pi) * std)) *  np.exp(-((feat_val-mean)**2 / (2 *
    return p_x_given_y
```

In [387…

```
def naive_bayes_gaussian(df, X, Y):
    # get feature names
    features = list(df.columns)[:-1]

    # calculate prior
    prior = calculate_prior(df, Y)

    Y_pred = []
    # loop over every data sample
    for x in X:
        # calculate likelihood
        labels = sorted(list(df[Y].unique()))
        likelihood = [1]*len(labels)
        for j in range(len(labels)):
            for i in range(len(features)):
                likelihood[j] *= calculate_likelihood(df, features[i], x[i], Y, labels[j

        # calculate posterior probability (numerator only)
        post_prob = [1]*len(labels)
        for j in range(len(labels)):
            post_prob[j] = likelihood[j] * prior[j]

        Y_pred.append(np.argmax(post_prob))

    return np.array(Y_pred)
```

Naive bayes model for all the features

In [388…

```
import warnings
warnings.filterwarnings( "ignore" )
data =ionospher_df.sample(frac=1,random_state=42)

train_size = int(0.8* len(X))

train_set = data[:train_size]
test_set = data[train_size:]
X_train = train_set.iloc[:,:-1].values
Y_train = train_set.iloc[:,-1].values

X_test = test_set.iloc[:,:-1].values
Y_test = test_set.iloc[:,-1].values


Y_pred=naive_bayes_gaussian(train_set,X_test,'output')
```

In [389…
```
Y_pred
```
Out[389]:
```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1], dtype=int64)
```
In [390…
```
acc=((Y_pred == Y_test).sum().astype(float) / len(Y_pred)*100)
print ('Accuracy on test set by our model   : {0} %'.format(acc))
```
```
Accuracy on test set by our model   : 68.57142857142857 %
```
Naive bayes model for some the features

In [391…
```
train_set = data[:train_size]
test_set = data[train_size:]

X_train = train_set.iloc[:,:-1].values
Y_train = train_set.iloc[:,-1].values

train_set=train_set[['column_1','column_3','column_5','column_7','column_8','column_9','
test_set=test_set[['column_1','column_3','column_5','column_7','column_8','column_9','co

X_test = test_set.iloc[:,:-1].values
Y_test = test_set.iloc[:,-1].values
train_set=train_set[['column_1','column_3','column_5','column_7','column_8','column_9','

Y_pred=naive_bayes_gaussian(train_set,X_test,'output')
```
In [392…
```
acc=((Y_pred == Y_test).sum().astype(float) / len(Y_pred)*100)
print ('Accuracy on test set by our model   : {0} %'.format(acc))
```
```
Accuracy on test set by our model   : 68.57142857142857 %
```
Univariate Naive bayes model for one the features

In [393…
```
train_set = data[:train_size]
test_set = data[train_size:]

X_train = train_set.iloc[:,:-1].values
Y_train = train_set.iloc[:,-1].values

test_set=test_set[['column_3','output']]
X_test = test_set.iloc[:,:-1].values
Y_test = test_set.iloc[:,-1].values
train_set=train_set[['column_3','output']]

Y_pred=naive_bayes_gaussian(train_set,X_test,'output')
```
In [394…
```
acc=((Y_pred == Y_test).sum().astype(float) / len(Y_pred)*100)
print ('Accuracy on test set by our model   : {0} %'.format(acc))
```
```
Accuracy on test set by our model   : 88.57142857142857 %
```

In [ ]:

Loading [MathJax]/extensions/Safe.js