In [1]:
```python
#Importing Necessary Libraries
import pandas as pd
import numpy as np

import matplotlib.pyplot as plt
%matplotlib inline

import seaborn as sns
sns.set()

import warnings
warnings.filterwarnings('ignore')

from pylab import rcParams
```

In [2]:
```python
#Loading our Dataset
#Use the link to download Dataset https://www.kaggle.com/keplersmachines/kepler-labelled-time-series-data/download
df_train=pd.read_csv("exoTrain.csv")
df_test=pd.read_csv("exoTest.csv")
```

In [3]:
```python
#Printing Top 5 Rows
df_train.head()
```

Out[3]:

	LABEL	FLUX.1	FLUX.2	FLUX.3	FLUX.4	FLUX.5	FLUX.6	FLUX.7	FLUX.8	FLUX.9	...	FLUX.3188	FLUX.3189	FLUX.3190	FLUX.3191	FLUX.3192	FLUX.3193	FLUX.3194	FLUX.3195	FLUX.3196	FLUX.3197
0	2	93.85	83.81	20.10	-26.98	-39.56	-124.71	-135.18	-96.27	-79.89	...	-78.07	-102.15	-102.15	25.13	48.57	92.54	39.32	61.42	5.08	-39.54
1	2	-38.88	-33.83	-58.54	-40.09	-79.31	-72.81	-86.55	-85.33	-83.97	...	-3.28	-32.21	-32.21	-24.89	-4.86	0.76	-11.70	6.46	16.00	19.93
2	2	532.64	535.92	513.73	496.92	456.45	466.00	464.50	486.39	436.56	...	-71.69	13.31	13.31	-29.89	-20.88	5.06	-11.80	-28.91	-70.02	-96.67
3	2	326.52	347.39	302.35	298.13	317.74	312.70	322.33	311.31	312.42	...	5.71	-3.73	-3.73	30.05	20.03	-12.67	-8.77	-17.31	-17.35	13.98
4	2	-1107.21	-1112.59	-1118.95	-1095.10	-1057.55	-1034.48	-998.34	-1022.71	-989.57	...	-594.37	-401.66	-401.66	-357.24	-443.76	-438.54	-399.71	-384.65	-411.79	-510.54

5 rows × 3198 columns

In [4]:
```python
#Changing Labels of Data
Categories = {2:1,1:0}
df_train.LABEL = [Categories[item] for item in df_train.LABEL]
df_test.LABEL = [Categories[item] for item in df_test.LABEL]
```

In [5]:
```python
print("Shape of Training Data : ",df_train.shape)
print("Shape of Test Data     : ",df_test.shape)
```

```
Shape of Training Data :  (5087, 3198)
Shape of Test Data     :  (570, 3198)
```

In [6]:
```python
df_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5087 entries, 0 to 5086
Columns: 3198 entries, LABEL to FLUX.3197
dtypes: float64(3197), int64(1)
memory usage: 124.1 MB
```

In [7]:
```python
#Checking Target Variable
df_train['LABEL'].value_counts()
```

Out[7]:
```
0    5050
1      37
Name: LABEL, dtype: int64
```

In [8]:
```python
df_test['LABEL'].value_counts()
```

Out[8]:
```
0    565
1      5
Name: LABEL, dtype: int64
```

This shows we have highly unbalanced dataset

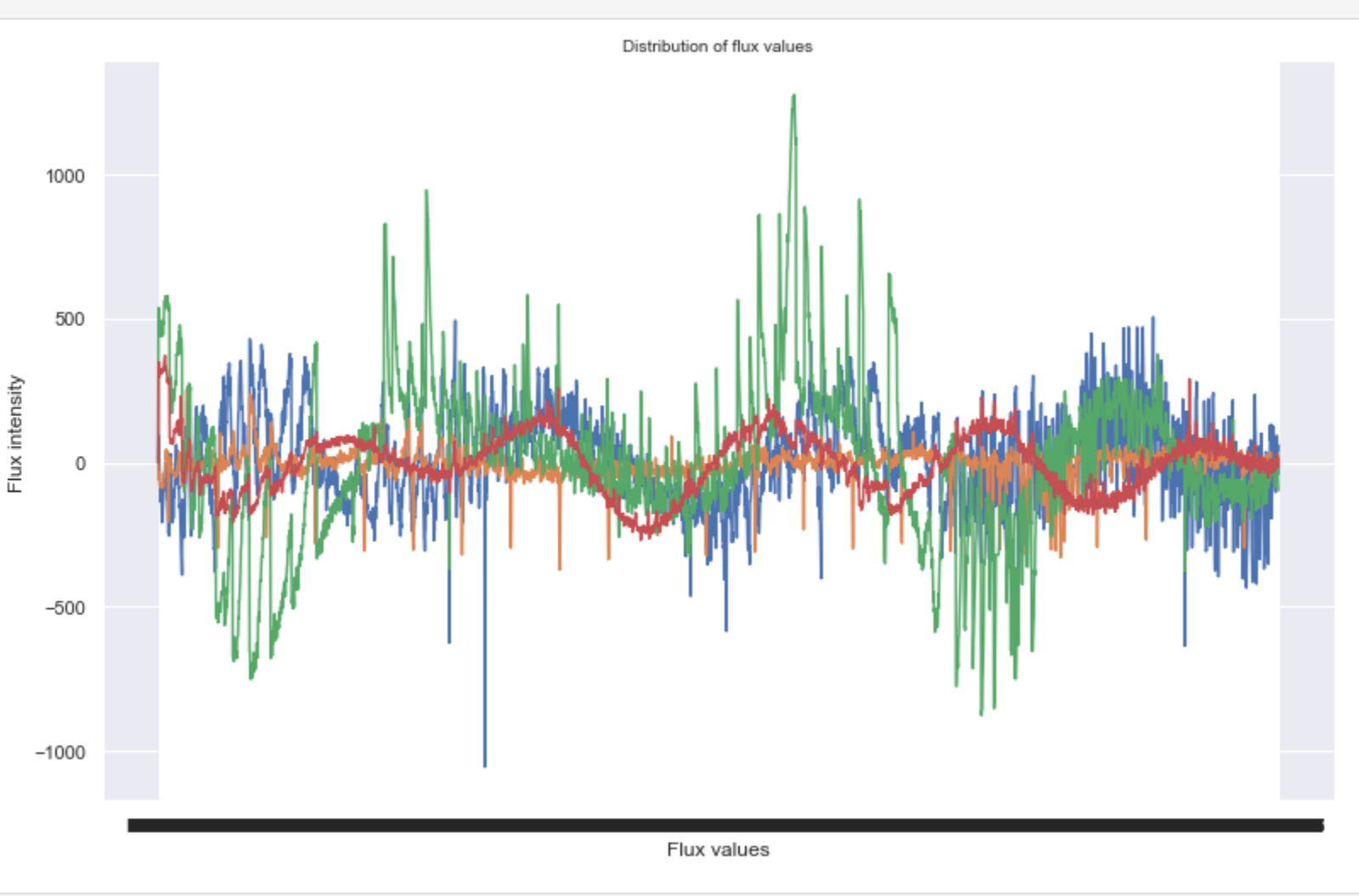
Flux

Flux (or radiant flux), F, is the total amount of energy that crosses a unit area per unit time. Flux is measured in joules per square metre per second (joules/m2/s), or watts per square metre (watts/m2). Reference By : https://astronomy.swin.edu.au/cosmos/F/Flux

F=L/4πr2

- F = flux measured at distance r
- L = luminosity of the source,
- r = distance to the source

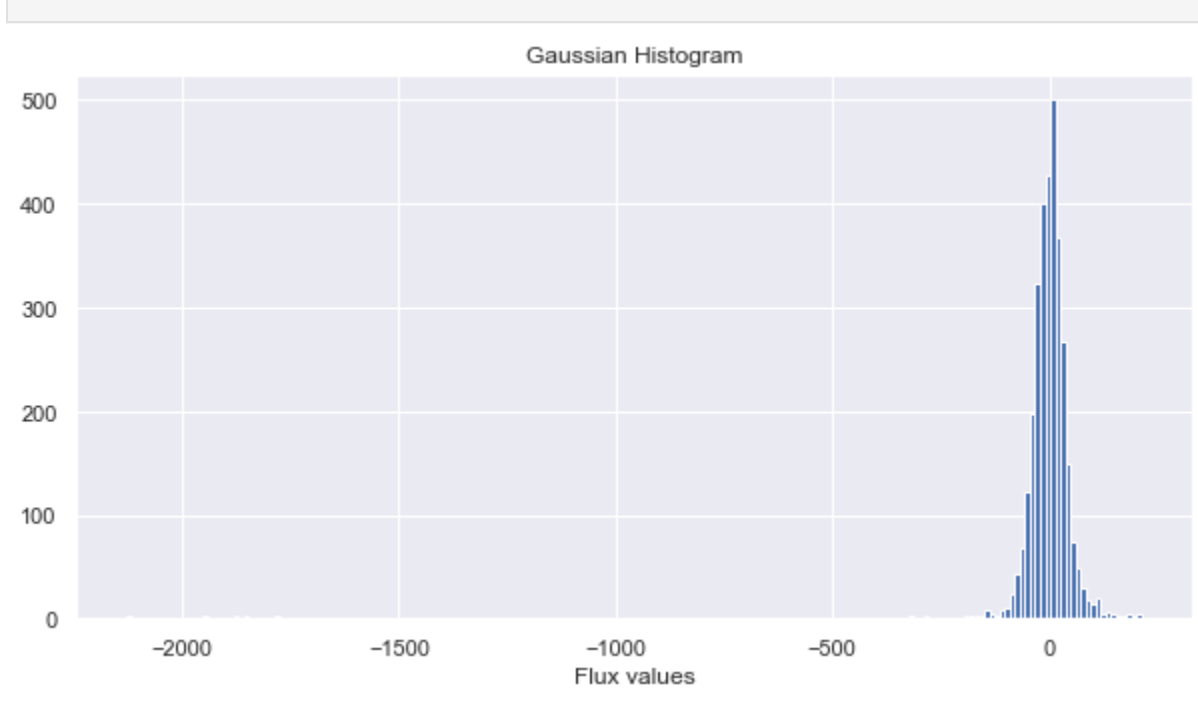
In [9]:
```python
#Checking Intensity of first 4 Rows

rcParams['figure.figsize'] = 13, 8
plt.title('Distribution of flux values', fontsize=10)
plt.xlabel('Flux values')
plt.ylabel('Flux intensity')
plt.plot(df_train.iloc[0,])
plt.plot(df_train.iloc[1,])
plt.plot(df_train.iloc[2,])
plt.plot(df_train.iloc[3,])
plt.show()
```

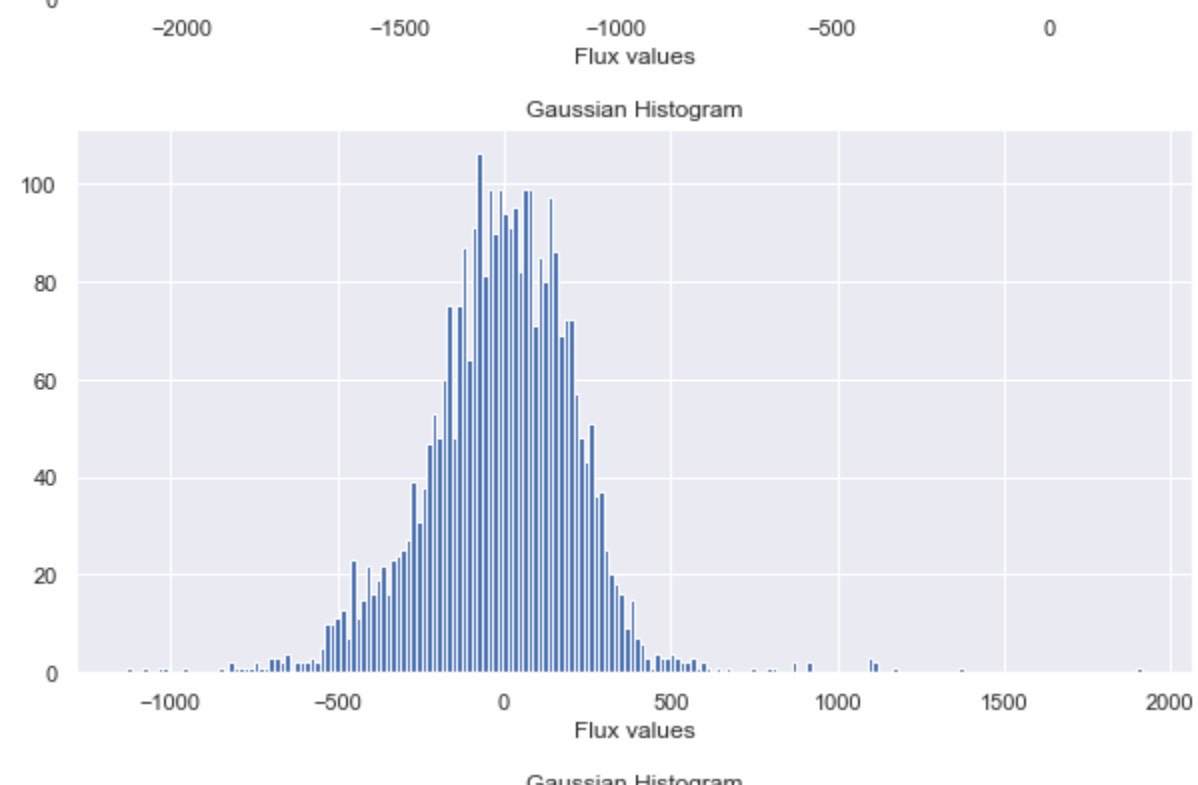


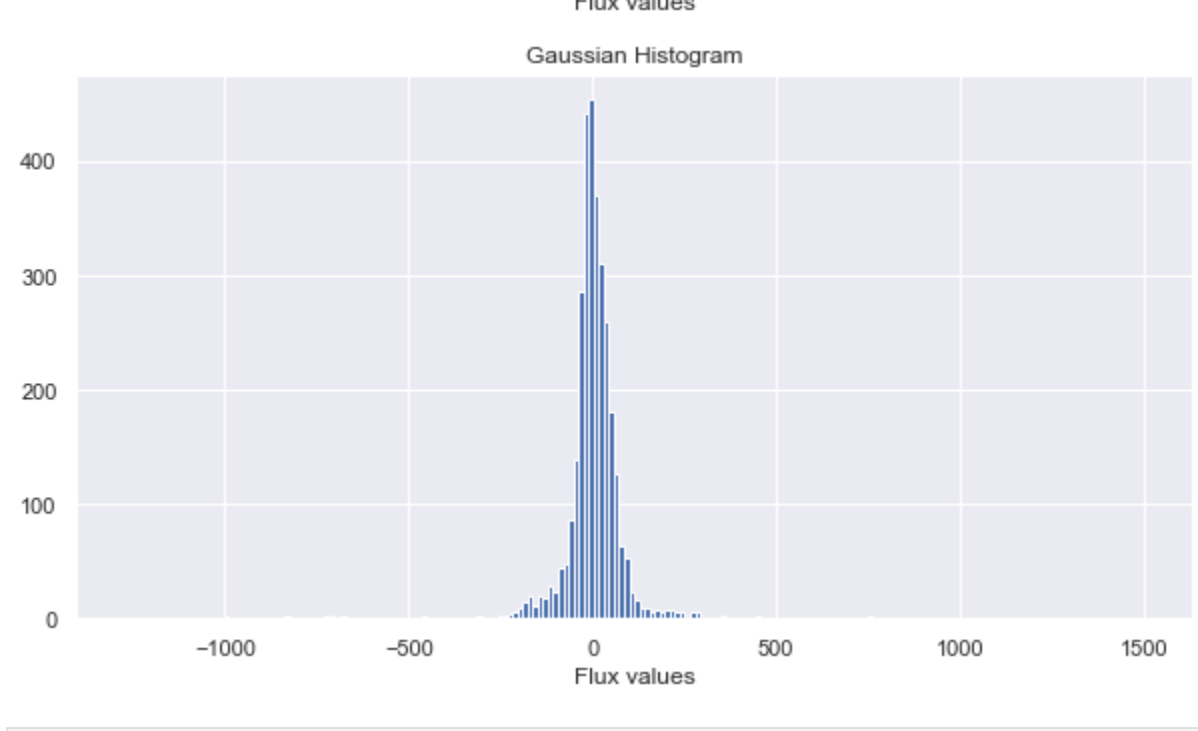
In [10]:
```python
#pip install plotly

train_exo_y=df_train[df_train['LABEL'] >0 ]
train_exo_n=df_train[df_train['LABEL'] <1]
train_t_n=train_exo_n.iloc[:,1:].T
train_t_y=train_exo_y.iloc[:,1:].T

import plotly.express as px
import plotly.graph_objects as go
from plotly.subplots import make_subplots
fig = make_subplots(rows=2, cols=2,subplot_titles=("Flux variation of star 37", "Flux variation of star 5086",
                                                   "Flux variation of star 3000", "Flux variation of star 3001"))
fig.add_trace(
    go.Scatter(y=train_t_n[37], x=train_t_n.index),
    row=1, col=1
)
fig.add_trace(
    go.Scatter(y=train_t_n[5086], x=train_t_n.index),
    row=1, col=2
)
fig.add_trace(
    go.Scatter(y=train_t_n[3000], x=train_t_n.index),
    row=2, col=1
)
fig.add_trace(
    go.Scatter(y=train_t_n[3001], x=train_t_n.index),
    row=2, col=2
)
fig.update_layout(height=600, width=800, title_text="Non Exoplanets Star examples",showlegend=False)
fig.show()
```

In [66]:
```python
fig = make_subplots(rows=2, cols=2,subplot_titles=("Flux variation of star 0", "Flux variation of star 1",
                                                   "Flux variation of star 35", "Flux variation of star 36"))
fig.add_trace(
    go.Scatter(y=train_t_y[0], x=train_t_y.index),
    row=1, col=1
)
fig.add_trace(
    go.Scatter(y=train_t_y[1], x=train_t_y.index),
    row=1, col=2
)
fig.add_trace(
    go.Scatter(y=train_t_y[35], x=train_t_y.index),
    row=2, col=1
)
fig.add_trace(
    go.Scatter(y=train_t_y[36], x=train_t_y.index),
    row=2, col=2
)
fig.update_layout(height=600, width=800, title_text="Exoplanets Stars examples",showlegend=False)
```

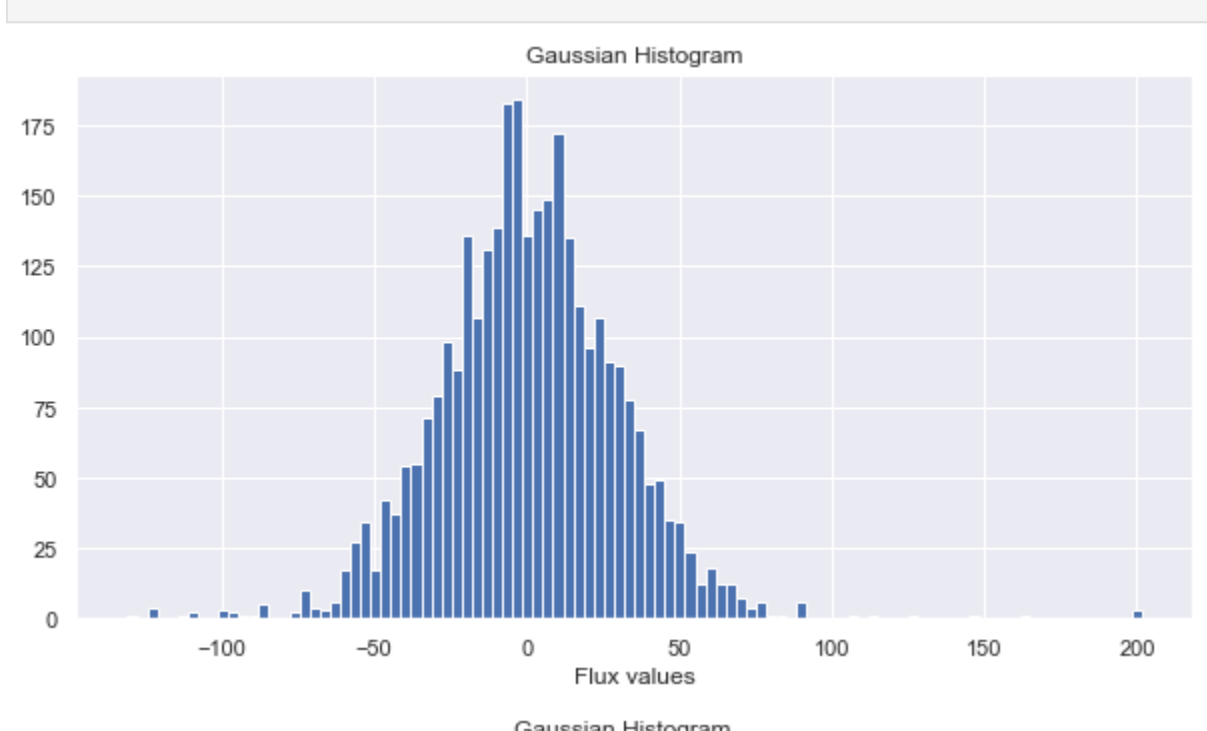
In [11]:
```python
#Plotting Gaussian Hisogram of non-exoplanets data

labels_1=[100,200,300]
for i in labels_1:
    plt.figure(figsize=(10,5))
    plt.hist(df_train.iloc[i,:], bins=200)
    plt.title("Gaussian Histogram")
    plt.xlabel("Flux values")
    plt.show()
```

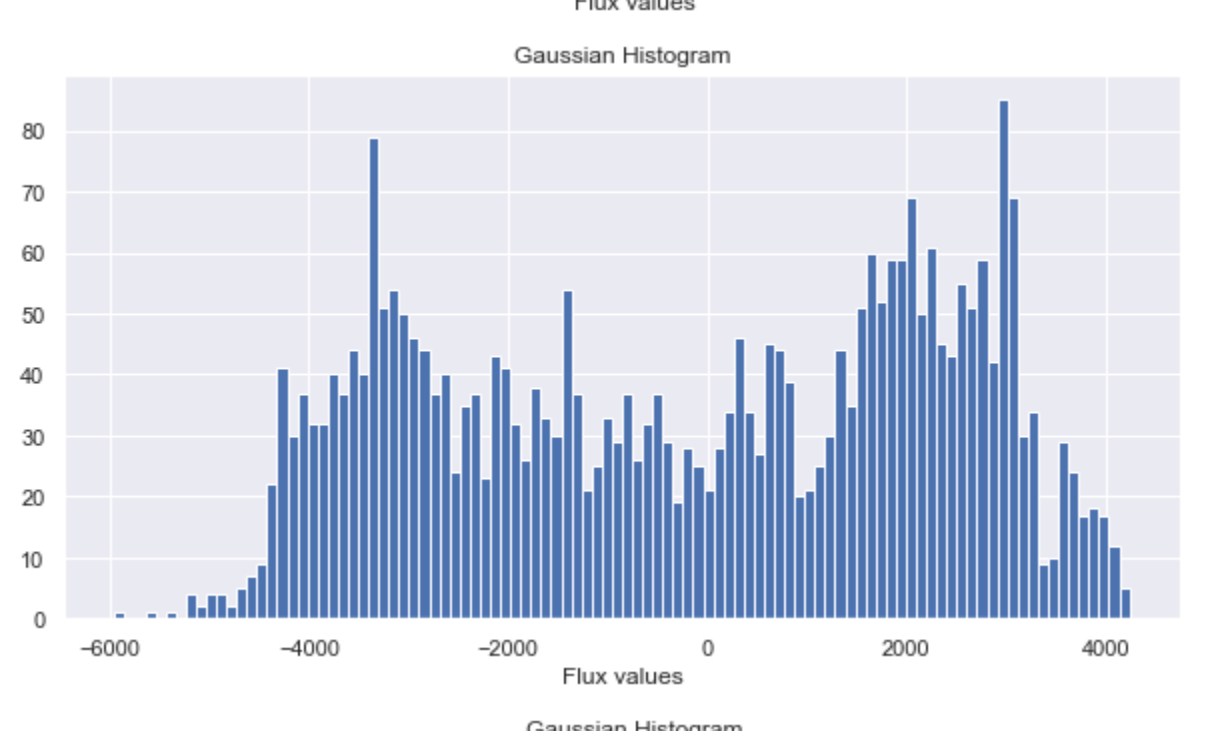


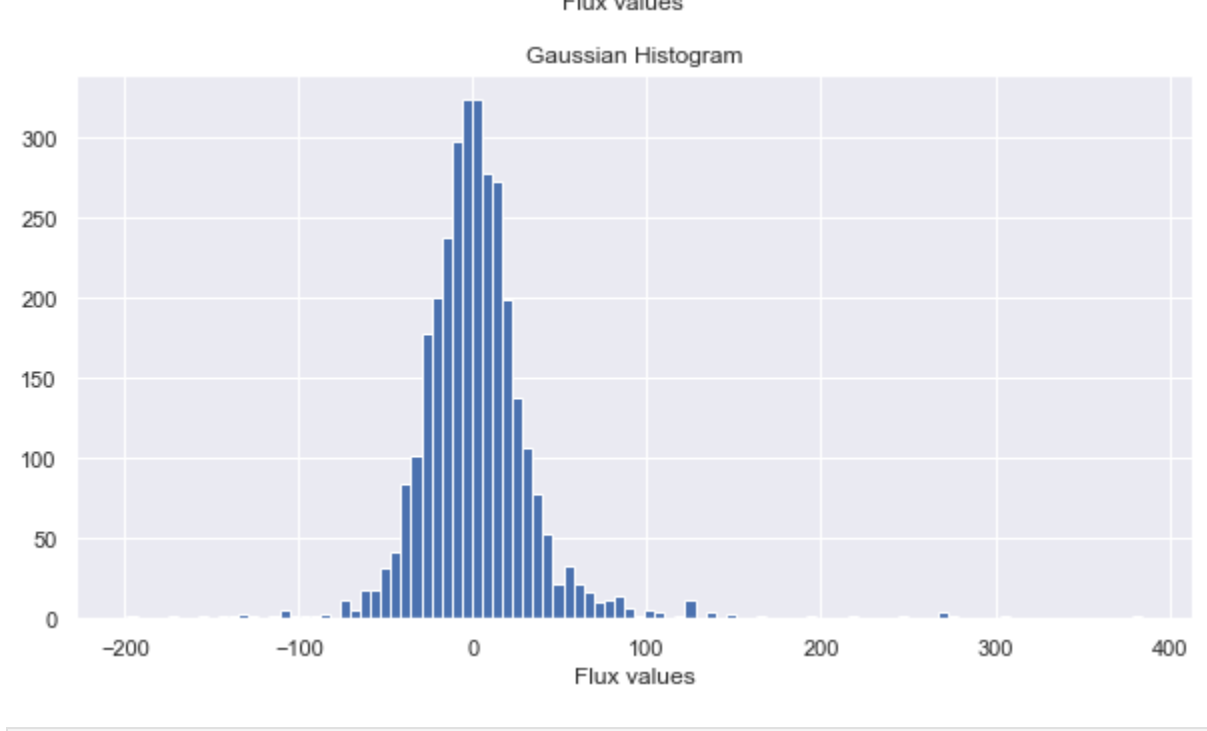




In [12]:
```python
#Plotting Gaussian Hisogram of Exoplanets data

labels_1=[16,21,25]
for i in labels_1:
    plt.figure(figsize=(10,5))
    plt.hist(df_train.iloc[i,:], bins=100)
    plt.title("Gaussian Histogram")
    plt.xlabel("Flux values")
    plt.show()
```







In [23]:
```python
#Splitting our Dataset to Train and Test

X_train = df_train.drop(["LABEL"],axis=1)
y_train = df_train["LABEL"]
X_test = df_test.drop(["LABEL"],axis=1)
y_test = df_test["LABEL"]
y_train
```

Out[23]:
```
0       1
1       1
2       1
3       1
4       1
       ..
5082    0
5083    0
5084    0
5085    0
5086    0
Name: LABEL, Length: 5087, dtype: int64
```

In [24]:
```python
from sklearn.preprocessing import StandardScaler, normalize

X_train = normalised = normalize(X_train)
X_test = normalize(X_test)
X_train
```

Out[24]:
```
array([[ 0.01030876,  0.00920594,  0.00220784, ...,  0.00674656,
         0.000558  , -0.00434319],
       [-0.01137337, -0.00989612, -0.01712441, ...,  0.00188971,
         0.0046804 ,  0.00583002],
       [ 0.03341091,  0.03361665,  0.03222474, ..., -0.00181344,
        -0.00439214, -0.00606382],
       ...,
       [ 0.07870498,  0.08003213,  0.07534824, ...,  0.02545482,
         0.0227631 ,  0.02286673],
       [ 0.00614169,  0.00336024, -0.00528957, ..., -0.02339309,
        -0.01030582, -0.00409982],
       [ 0.09246656,  0.087627  ,  0.08385145, ..., -0.00478236,
        -0.00403011,  0.00795725]])
```

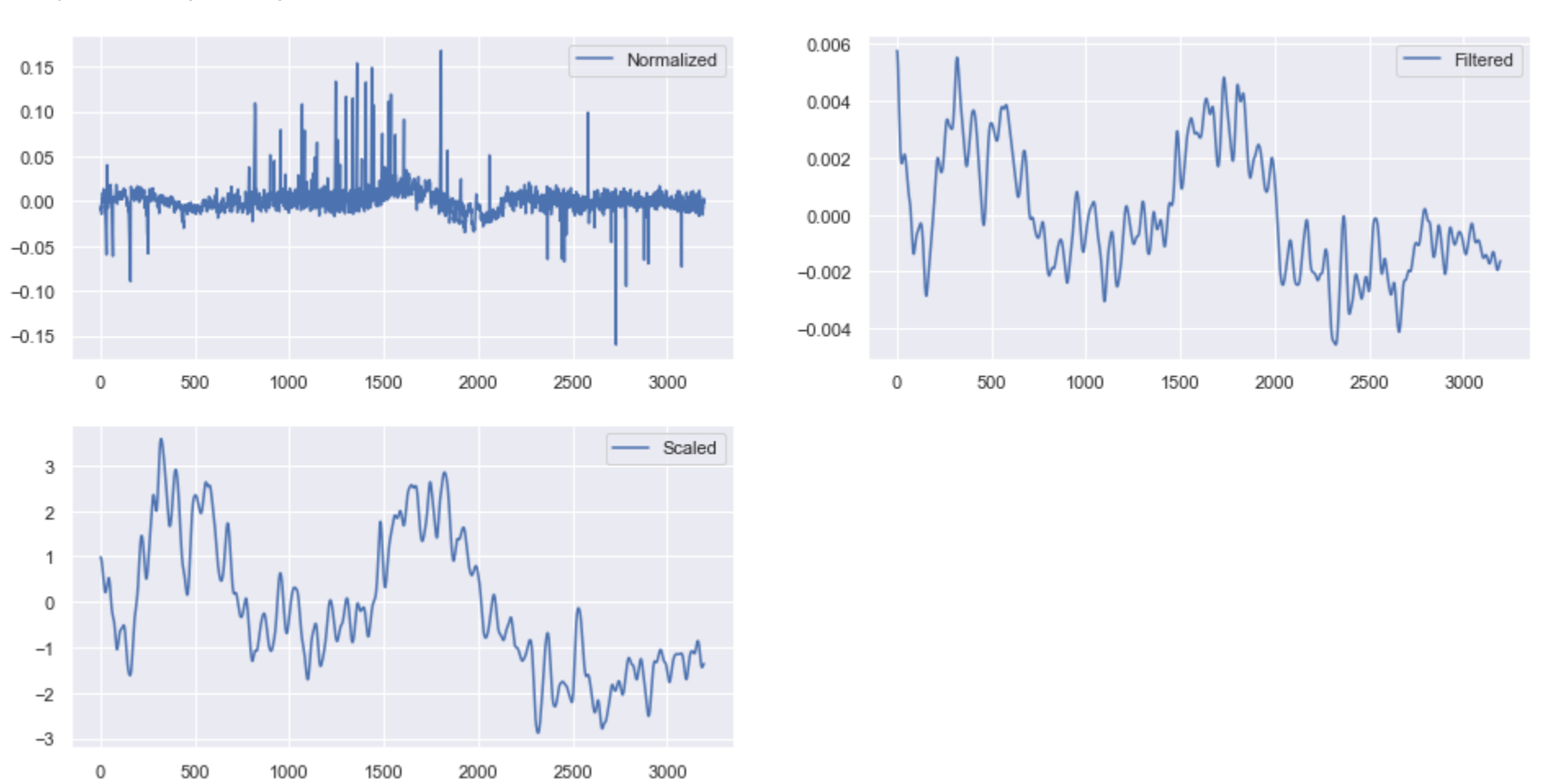
In [25]:
```python
#Using Gaussian Filters to remove random noise from Data
from scipy import ndimage
X_train = filtered = ndimage.filters.gaussian_filter(X_train, sigma=10)
X_test = ndimage.filters.gaussian_filter(X_test, sigma=10)
```

In [26]:
```python
#Feature scaling
std_scaler = StandardScaler()
X_train = scaled = std_scaler.fit_transform(X_train)
X_test = std_scaler.fit_transform(X_test)
X_train.shape
```

Out[26]:
```
(5087, 3197)
```

In [27]:
```python
series_number=38
plt.figure(figsize=(16,8))
plt.subplot(221)
# plt.plot(absolute[series_number], label='Original')
# plt.legend(loc='best')
# plt.subplot(222)
plt.plot(normalised[series_number], label='Normalized')
plt.legend(loc='best')
plt.subplot(222)
plt.plot(filtered[series_number],label='Filtered')
plt.legend(loc='best')
plt.subplot(223)
plt.plot(scaled[series_number],label='Scaled')
plt.legend(loc='best')
```

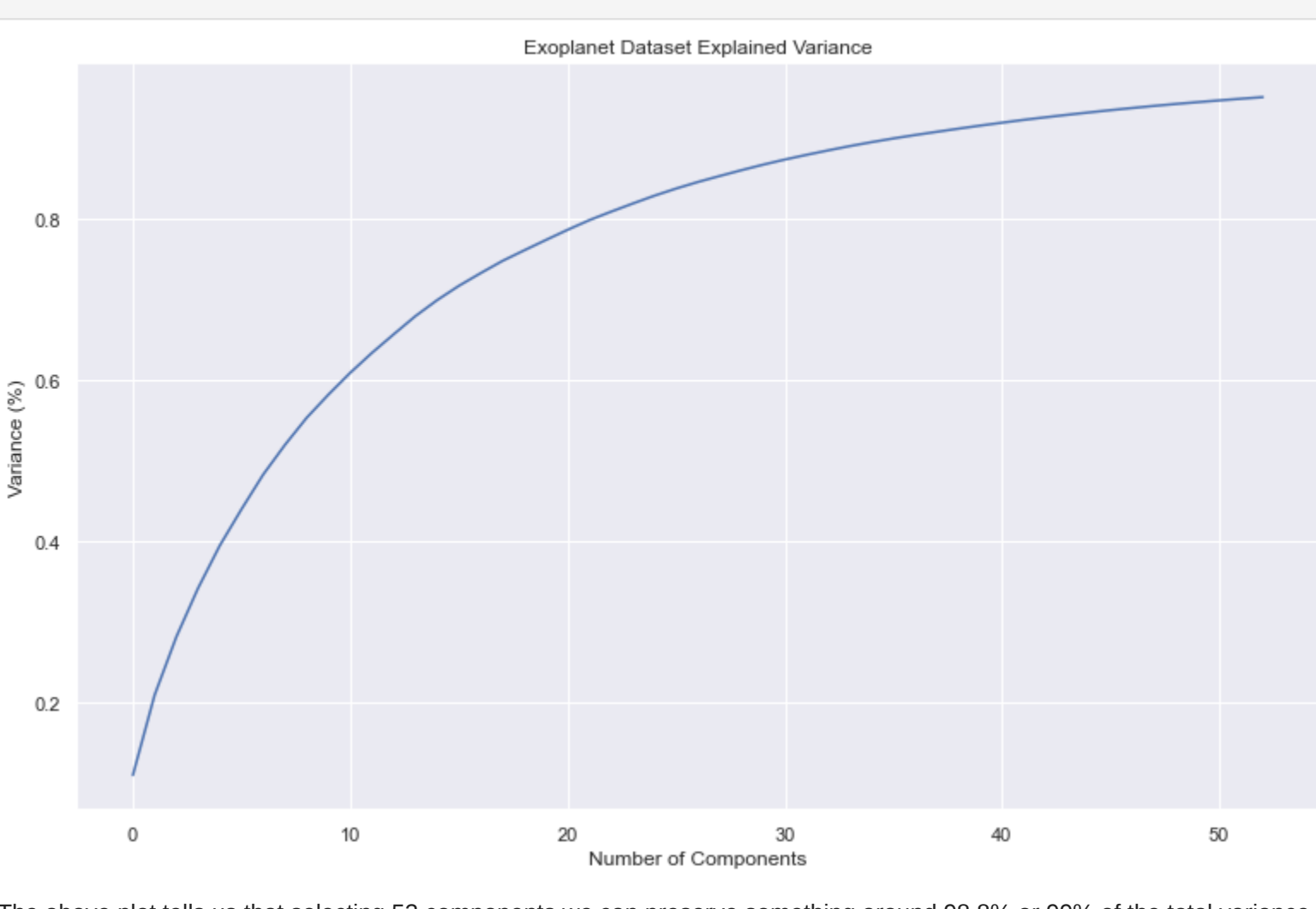
Out[27]:
```
<matplotlib.legend.Legend at 0x1fcdf8afdf0>
```



In [28]:
```python
#For Reducing the dimensions of our dataset, We will be using Principal Component Analysis

from sklearn.decomposition import PCA
pca = PCA()
x_train = pca.fit_transform(X_train)
x_test = pca.transform(X_test)
total=sum(pca.explained_variance_)

k=0
current_variance=0
while current_variance/total < 0.95:
    current_variance += pca.explained_variance_[k]
    k=k+1

print("Value of K : ",k)
```

```
Value of K :  53
```

In [29]:
```python
pca = PCA(n_components=53)
X_train = pca.fit_transform(X_train)
X_test = pca.transform(X_test)
plt.figure()
plt.plot(np.cumsum(pca.explained_variance_ratio_))
plt.xlabel('Number of Components')
plt.ylabel('Variance (%)') #for each component
plt.title('Exoplanet Dataset Explained Variance')
plt.show()
```



The above plot tells us that selecting 53 components we can preserve something around 98.8% or 99% of the total variance of the data. It makes sense, we'll not use 100% of our variance, because it denotes all components, and we want only the principal ones.

As we are dealing with Unbalanced Dataset, We will be using SMOTE(Synthetic Minority Over-sampling Technique)

It is an over-sampling method. What it does is, it creates synthetic (not duplicate) samples of the minority class. Hence making the minority class equal to the majority class. SMOTE does this by selecting similar records and altering that record one column at a time by a random amount within the difference to the neighboring records.

In [30]:
```python
#Resampling
print("Before OverSampling, counts of label '1': {}".format(sum(y_train==1)))
print("Before OverSampling, counts of label '0': {} \n".format(sum(y_train==0)))
#pip install imblearn
from imblearn.over_sampling import SMOTE
sm = SMOTE(random_state=27)
x_train_res, y_train_res = sm.fit_resample(X_train, y_train.ravel())
print("After OverSampling, counts of label '1': {}".format(sum(y_train_res==1)))
print("After OverSampling, counts of label '0': {}".format(sum(y_train_res==0)))

x_train_res.shape
```

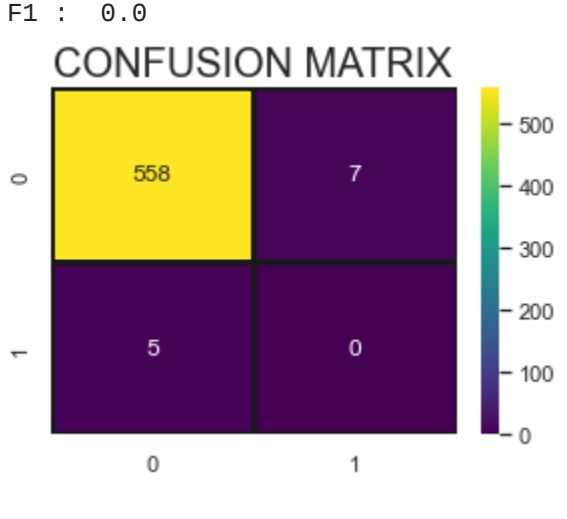
```
Before OverSampling, counts of label '1': 37
Before OverSampling, counts of label '0': 5050

After OverSampling, counts of label '1': 5050
After OverSampling, counts of label '0': 5050
```

Out[30]:
```
(10100, 53)
```

In [31]:
```python
from sklearn.model_selection import cross_val_score
from sklearn.metrics import precision_score, recall_score,roc_curve,auc, f1_score,roc_auc_score,confusion_matrix, accuracy_score, classification_report
from sklearn.metrics import plot_precision_recall_curve,plot_confusion_matrix
from sklearn.svm import LinearSVC
def model(classifier,dtrain_x,dtrain_y,dtest_x,dtest_y):
    #fit the model
    classifier.fit(dtrain_x,dtrain_y)
    predictions = classifier.predict(dtest_x)

    #Cross validation
    accuracies = cross_val_score(estimator = classifier, X = x_train_res, y = y_train_res, cv = 5, n_jobs = -1)
    mean = accuracies.mean()
    variance = accuracies.std()
    print("Accuracy mean: "+ str(mean))
    print("Accuracy variance: "+ str(variance))

    #Accuracy
    print ("\nAccuracy_score :",accuracy_score(dtest_y,predictions))

    #Classification report
    print ("\nclassification report :\n",(classification_report(dtest_y,predictions)))

    #Confusion matrix
    plt.figure(figsize=(10,7))
    plt.subplot(221)
    sns.heatmap(confusion_matrix(dtest_y,predictions),annot=True,cmap="viridis",fmt = "d",linecolor="k",linewidths=3)
    plt.title("CONFUSION MATRIX",fontsize=20)

    print("Precision : ",precision_score(dtest_y,predictions))
    print("Recall : ",recall_score(dtest_y,predictions))
    print("F1 : ",f1_score(dtest_y,predictions))
```

In [32]:
```python
LinearSVC_model=LinearSVC()
model(LinearSVC_model,x_train_res,y_train_res,X_test,y_test)
```

```
Accuracy mean: 0.9428712871287128
Accuracy variance: 0.020193165792200634

Accuracy_score : 0.9789473684210527

classification report :
               precision    recall  f1-score   support

           0       0.99      0.99      0.99       565
           1       0.00      0.00      0.00         5

    accuracy                           0.98       570
   macro avg       0.50      0.49      0.49       570
weighted avg       0.98      0.98      0.98       570

Precision :  0.0
Recall :  0.0
F1 :  0.0
```



In Linear SVC, We are unable to predict Exo-planets. Lets try with Sigmoid Kernel

In [33]:
```python
from sklearn.svm import SVC
SVM_model=SVC(gamma='auto', kernel='sigmoid')
model(SVM_model,x_train_res,y_train_res,X_test,y_test)
```

```
Accuracy mean: 0.554059405940594
Accuracy variance: 0.03531393232553576

Accuracy_score : 0.6070175438596491

classification report :
               precision    recall  f1-score   support

           0       1.00      0.60      0.75       565
           1       0.02      1.00      0.04         5

    accuracy                           0.61       570
   macro avg       0.51      0.80      0.40       570
weighted avg       0.99      0.61      0.75       570

Precision :  0.021834061135371178
Recall :  1.0
F1 :  0.042735042735042736
```