IDENTIFYING EXO-PLANETS USING MACHINE LEARNING

Included files:

final-dubey.py <- Full Python Code
final-dubey.html <- Print from Jupyter Notebook for easier legibility/Readme
Plots <- Folder with PNG of all Plots
final-dubey-project.pdf <- Project write-up
final-dubey-readme.pdf <- PDF of HTML

In [2]:
```
import os
import warnings
import math
warnings.filterwarnings('ignore')
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
plt.style.use('fivethirtyeight')
from pylab import rcParams
rcParams['figure.figsize'] = 10, 6
from sklearn.metrics import mean_squared_error, mean_absolute_error
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import train_test_split
from sklearn import linear_model
from sklearn.model_selection import cross_val_score
from sklearn.metrics import precision_score, recall_score,roc_curve,auc, f1_sc
ore, roc_auc_score,confusion_matrix, accuracy_score, classification_report
from sklearn.preprocessing import StandardScaler, normalize
from scipy import ndimage
import seaborn as sns
```

In [3]:
```
test_data = pd.read_csv('data\exoTest.csv').fillna(0)
train_data = pd.read_csv('data\exoTrain.csv').fillna(0)
train_data.head()
```

Out[3]:

	LABEL	FLUX.1	FLUX.2	FLUX.3	FLUX.4	FLUX.5	FLUX.6	FLUX.7	FLUX.8	FLUX.9	.
0	2	93.85	83.81	20.10	-26.98	-39.56	-124.71	-135.18	-96.27	-79.89	.
1	2	-38.88	-33.83	-58.54	-40.09	-79.31	-72.81	-86.55	-85.33	-83.97	.
2	2	532.64	535.92	513.73	496.92	456.45	466.00	464.50	486.39	436.56	.
3	2	326.52	347.39	302.35	298.13	317.74	312.70	322.33	311.31	312.42	.
4	2	-1107.21	-1112.59	-1118.95	-1095.10	-1057.55	-1034.48	-998.34	-1022.71	-989.57	.

5 rows × 3198 columns

Data Pre-Processing

In [3]:
```
# Changing 2 to 1 to represent YES exoplanet
# Changing 1 to 0 to represent NO exoplanet
categ = {2: 1,1: 0}
train_data.LABEL = [categ[item] for item in train_data.LABEL]
test_data.LABEL = [categ[item] for item in test_data.LABEL]
train_data.head()
```

Out[3]:

	LABEL	FLUX.1	FLUX.2	FLUX.3	FLUX.4	FLUX.5	FLUX.6	FLUX.7	FLUX.8	FLUX.9	.
0	1	93.85	83.81	20.10	-26.98	-39.56	-124.71	-135.18	-96.27	-79.89	.
1	1	-38.88	-33.83	-58.54	-40.09	-79.31	-72.81	-86.55	-85.33	-83.97	.
2	1	532.64	535.92	513.73	496.92	456.45	466.00	464.50	486.39	436.56	.
3	1	326.52	347.39	302.35	298.13	317.74	312.70	322.33	311.31	312.42	.
4	1	-1107.21	-1112.59	-1118.95	-1095.10	-1057.55	-1034.48	-998.34	-1022.71	-989.57	.

5 rows × 3198 columns

Reduce memory

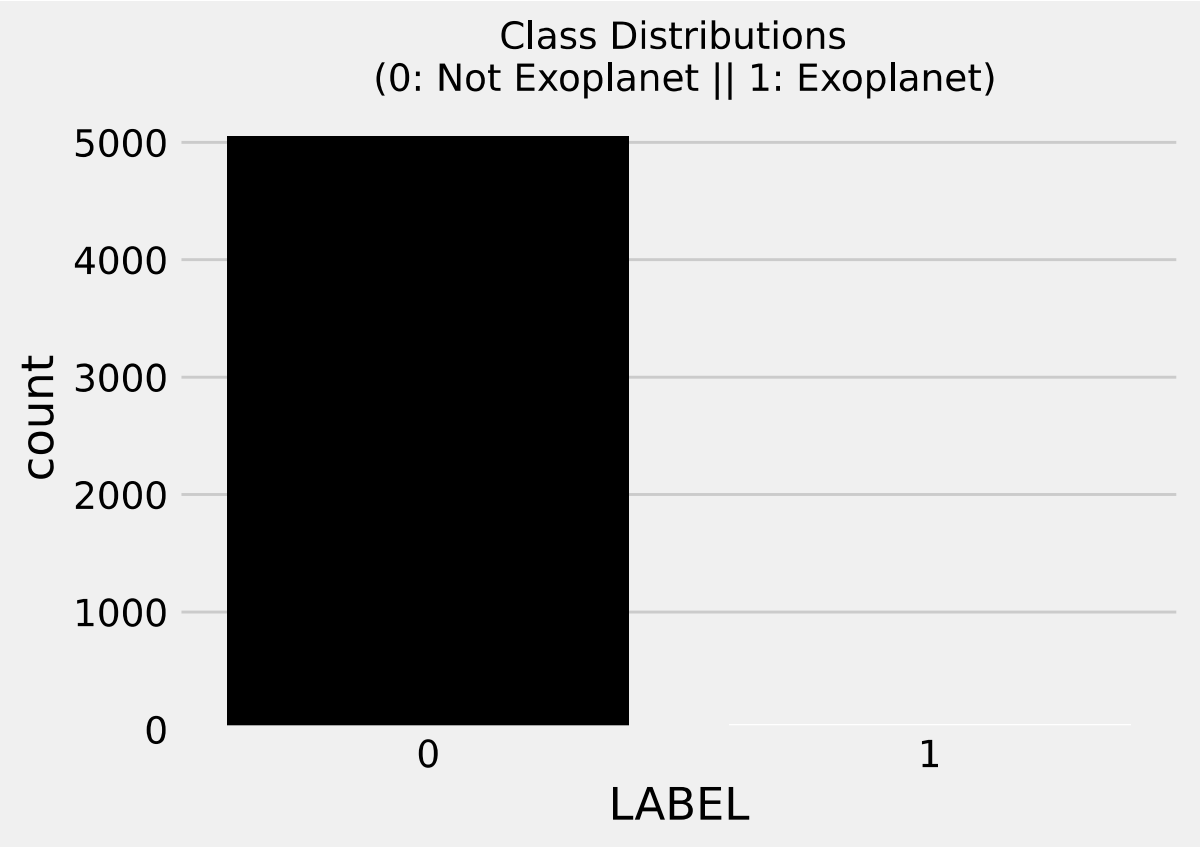
In [4]:
```
#Reduce memory
def reduce_memory(df):
    """ iterate through all the columns of a dataframe and modify the data typ
e
        to reduce memory usage.
    """
    start_mem = df.memory_usage().sum() / 1024**2
    print('Memory usage of dataframe is {:.2f} MB'.format(start_mem))

    for col in df.columns:
        col_type = df[col].dtype

        if col_type != object:
            c_min = df[col].min()
            c_max = df[col].max()
            if str(col_type)[:3] == 'int':
                if c_min > np.iinfo(np.int8).min and c_max < np.iinfo(np.int8)
.max:
                    df[col] = df[col].astype(np.int8)
                elif c_min > np.iinfo(np.int16).min and c_max < np.iinfo(np.in
t16).max:
                    df[col] = df[col].astype(np.int16)
                elif c_min > np.iinfo(np.int32).min and c_max < np.iinfo(np.in
t32).max:
                    df[col] = df[col].astype(np.int32)
                elif c_min > np.iinfo(np.int64).min and c_max < np.iinfo(np.in
t64).max:
                    df[col] = df[col].astype(np.int64)
            else:
                if c_min > np.finfo(np.float16).min and c_max < np.finfo(np.fl
oat16).max:
                    df[col] = df[col].astype(np.float16)
                elif c_min > np.finfo(np.float32).min and c_max < np.finfo(np.
float32).max:
                    df[col] = df[col].astype(np.float32)
                else:
                    df[col] = df[col].astype(np.float64)
        else:
            df[col] = df[col].astype('category')

    end_mem = df.memory_usage().sum() / 1024**2
    print('Memory usage after optimization is: {:.2f} MB'.format(end_mem))
    print('Decreased by {:.1f}%'.format(100 * (start_mem - end_mem) / start_me
m))
    # dftest_data = reduce_memory(test_data)
    return df
```

In [5]:
```
re_train_data = reduce_memory(train_data)
re_test_data = reduce_memory(test_data)
```

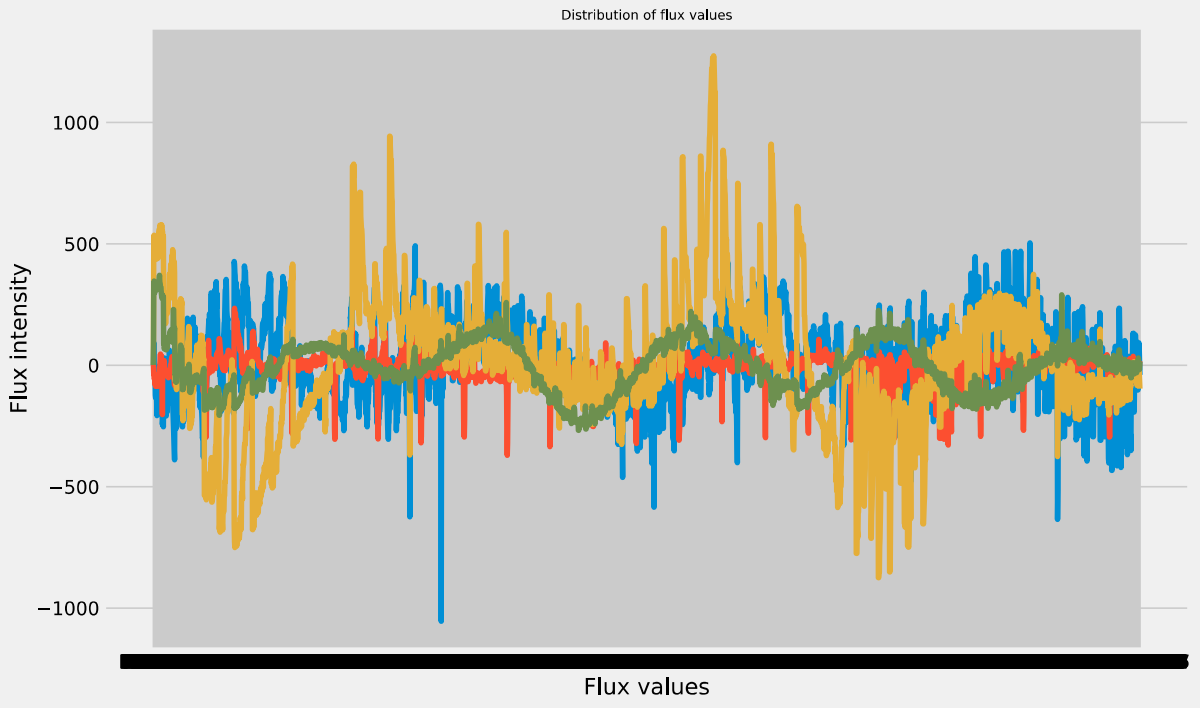
```
Memory usage of dataframe is 124.12 MB
Memory usage after optimization is: 62.04 MB
Decreased by 50.0%
Memory usage of dataframe is 13.91 MB
Memory usage after optimization is: 6.25 MB
Decreased by 55.1%
```

In [6]:
```
plt.figure(figsize=(6,4))
colors = ["0", "1"]
sns.countplot('LABEL', data=train_data, palette=colors)
plt.title('Class Distributions \n (0: Not Exoplanet || 1: Exoplanet)', fontsiz
e=14)
```

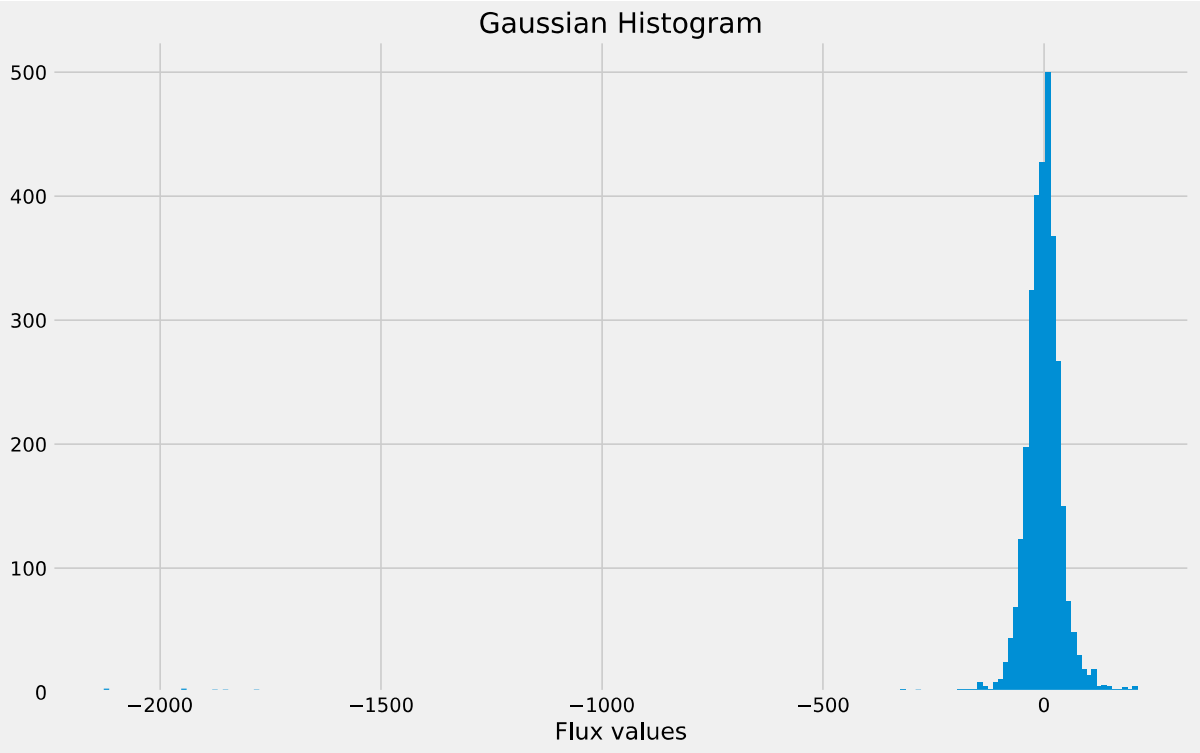
Out[6]: Text(0.5, 1.0, 'Class Distributions \n (0: Not Exoplanet || 1: Exoplanet)')

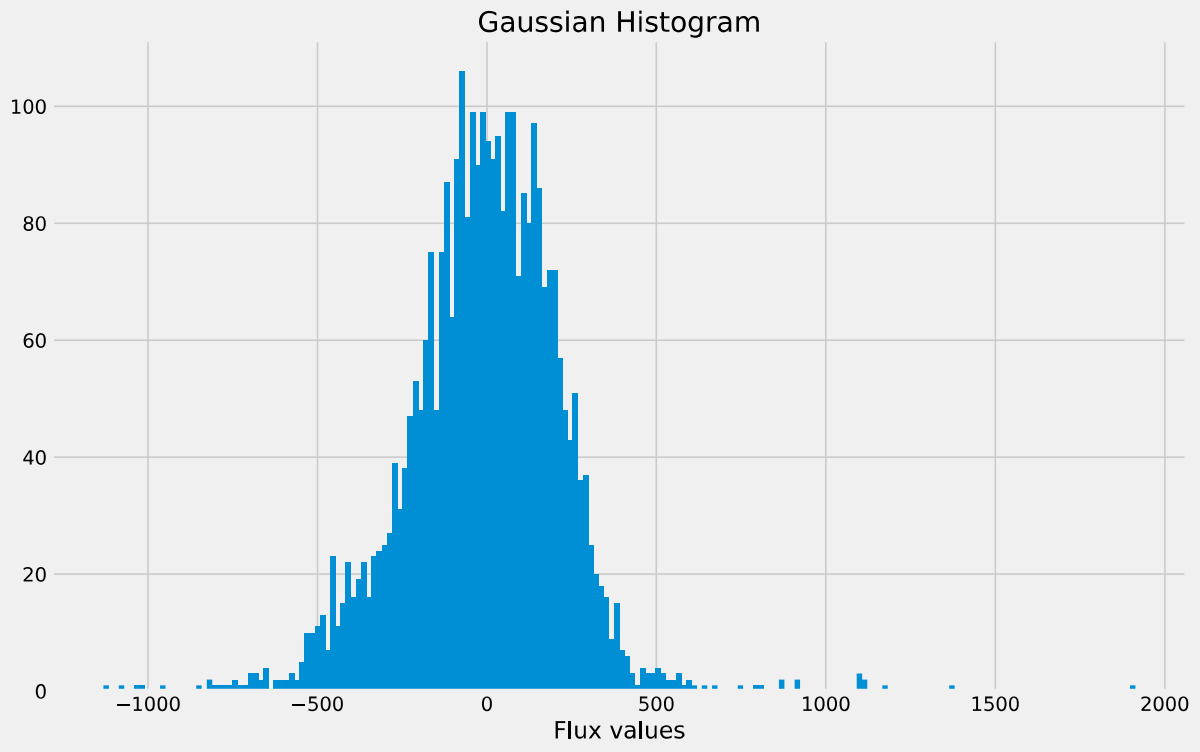


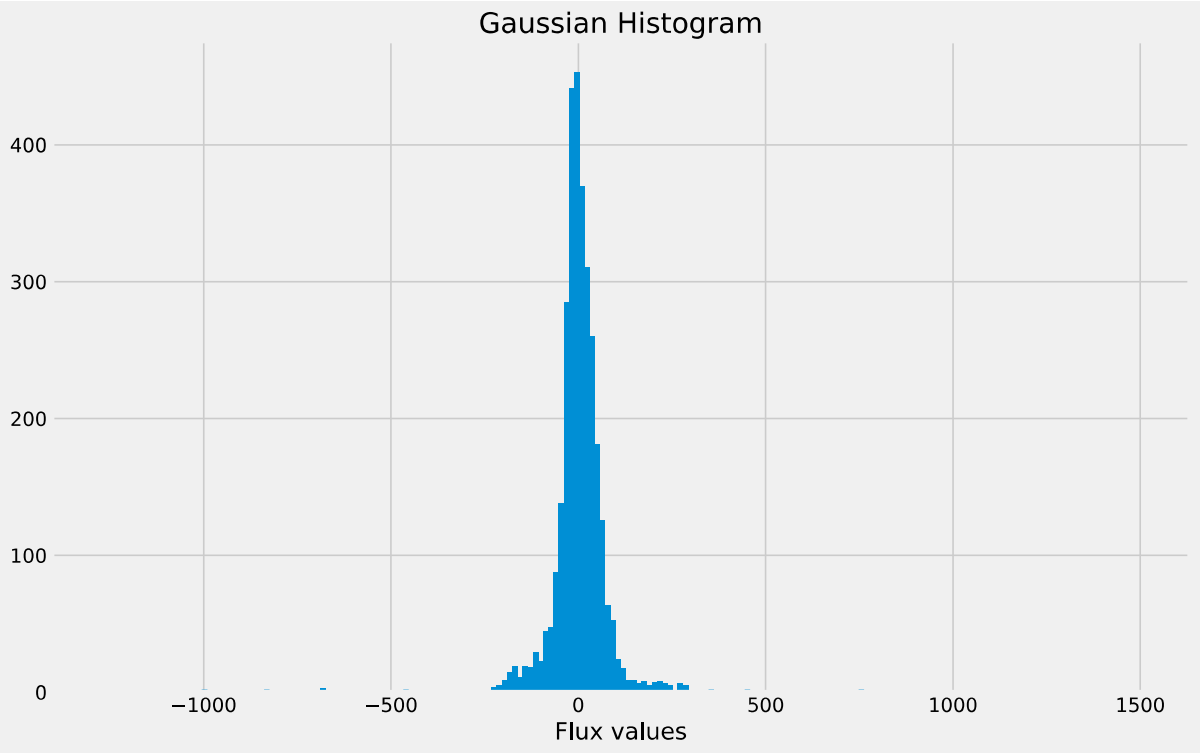
In [7]:
```
from pylab import rcParams
rcParams['figure.figsize'] = 13, 8
plt.title('Distribution of flux values', fontsize=10)
plt.xlabel('Flux values')
plt.ylabel('Flux intensity')
plt.plot(train_data.iloc[0,])
plt.plot(train_data.iloc[1,])
plt.plot(train_data.iloc[2,])
plt.plot(train_data.iloc[3,])
plt.show()
```



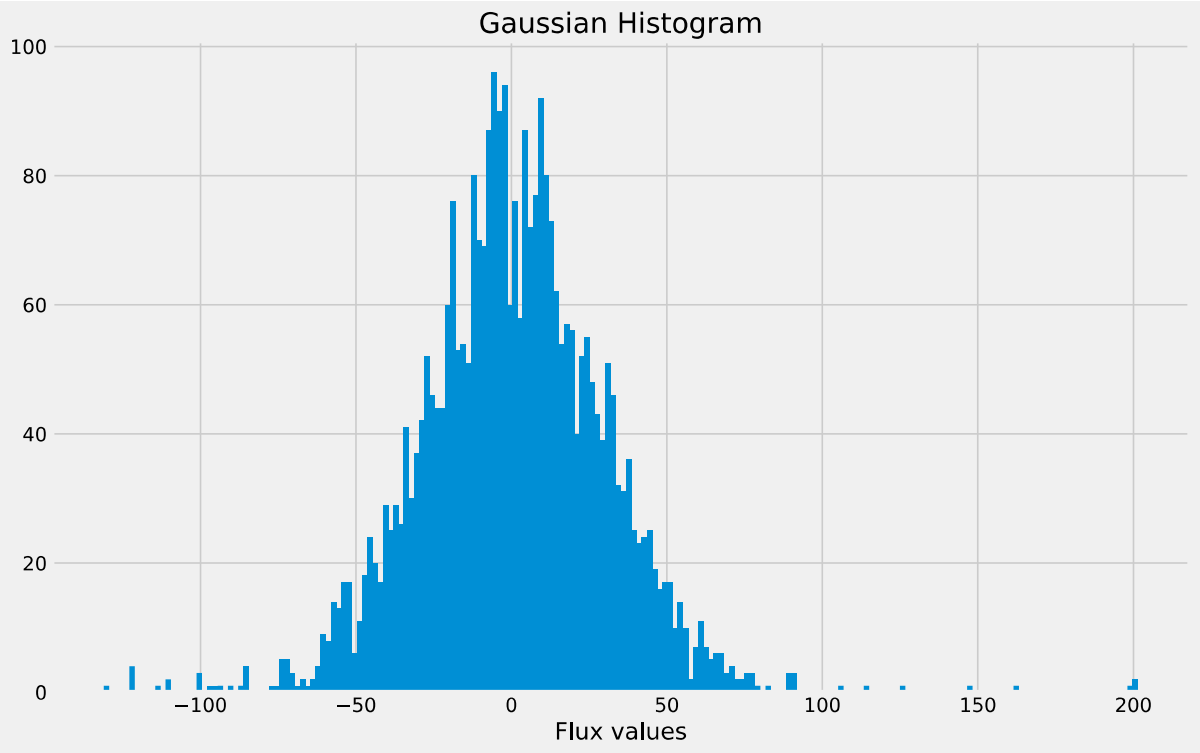
In [8]:
```
labels_1=[100,200,300]

for i in labels_1:
    plt.hist(train_data.iloc[i,:], bins=200)
    plt.title("Gaussian Histogram")
    plt.xlabel("Flux values")
    plt.show()
```

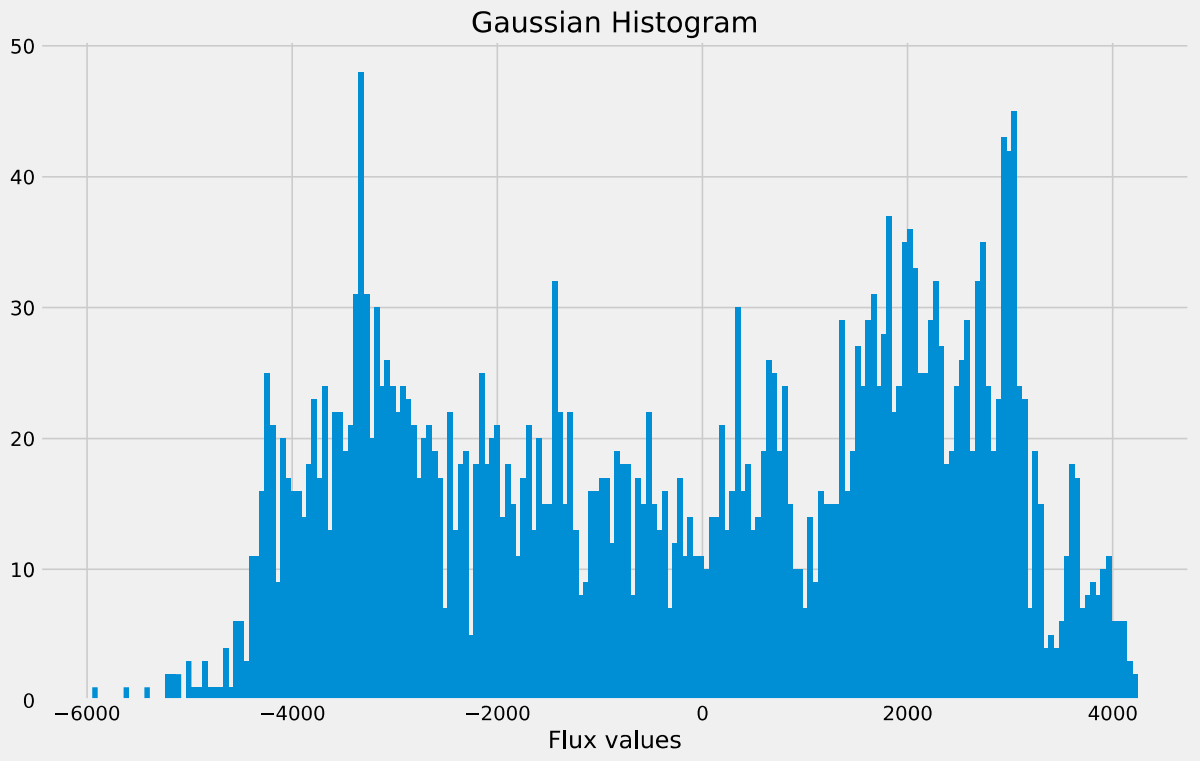

Gaussian Histogram
500
400
300
200
100
0
−2000
−1500
−1000
−500
0
Flux values

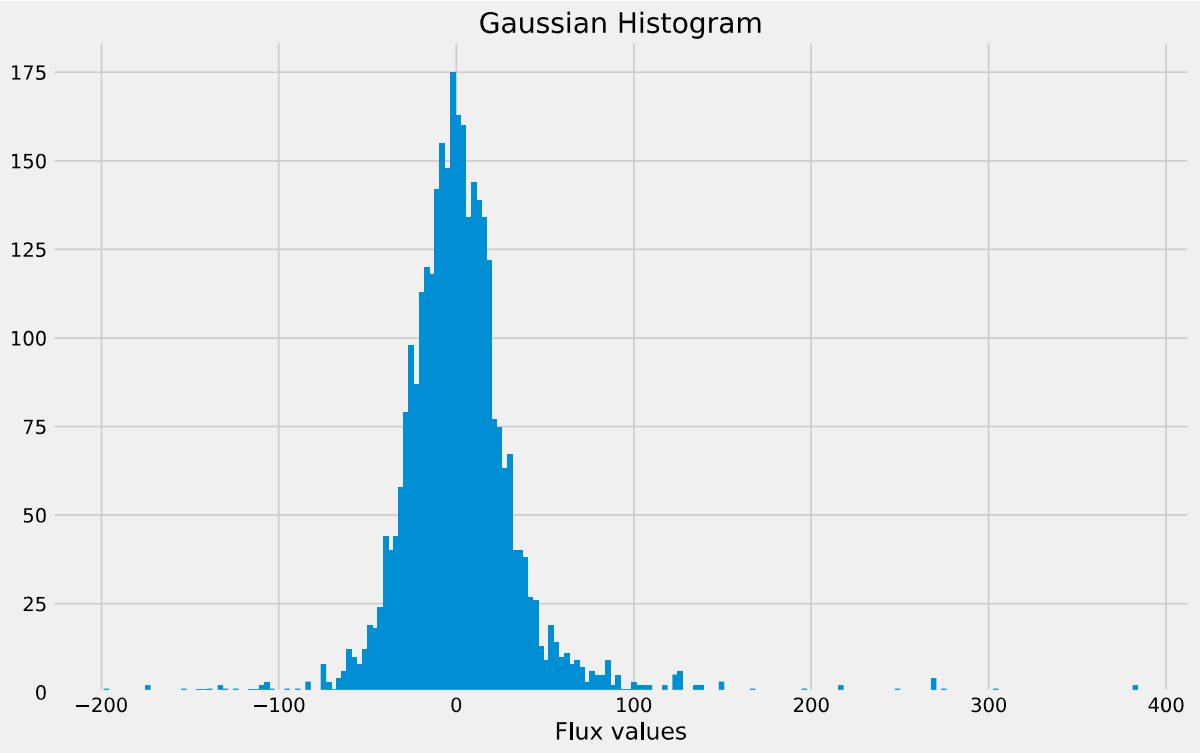

Gaussian Histogram
100
80
60
40
20
0
−1000
−500
0
500
1000
1500
2000
Flux values


Gaussian Histogram
400
300
200
100
0
−1000
−500
0
500
1000
1500
Flux values

In [9]:
```
labels_1=[16,21,25]
for i in labels_1:
    plt.hist(train_data.iloc[i,:], bins=200)
    plt.title("Gaussian Histogram")
    plt.xlabel("Flux values")
    plt.show()
```


Gaussian Histogram
100
80
60
40
20
0
−100
−50
0
50
100
150
200
Flux values


Gaussian Histogram
50
40
30
20
10
0
−6000
−4000
−2000
0
2000
4000
Flux values



In [10]:
```
x_train = re_train_data.drop(["LABEL"],axis=1)
y_train = re_train_data["LABEL"]
x_test = re_test_data.drop(["LABEL"],axis=1)
y_test = re_test_data["LABEL"]
```

In [11]:
```
x_train = normalized = normalize(x_train)
x_test = normalize(x_test)
```

In [12]:
```
x_train = filtered = ndimage.filters.gaussian_filter(x_train, sigma=10)
x_test = ndimage.filters.gaussian_filter(x_test, sigma=10)
```

Feature scaling

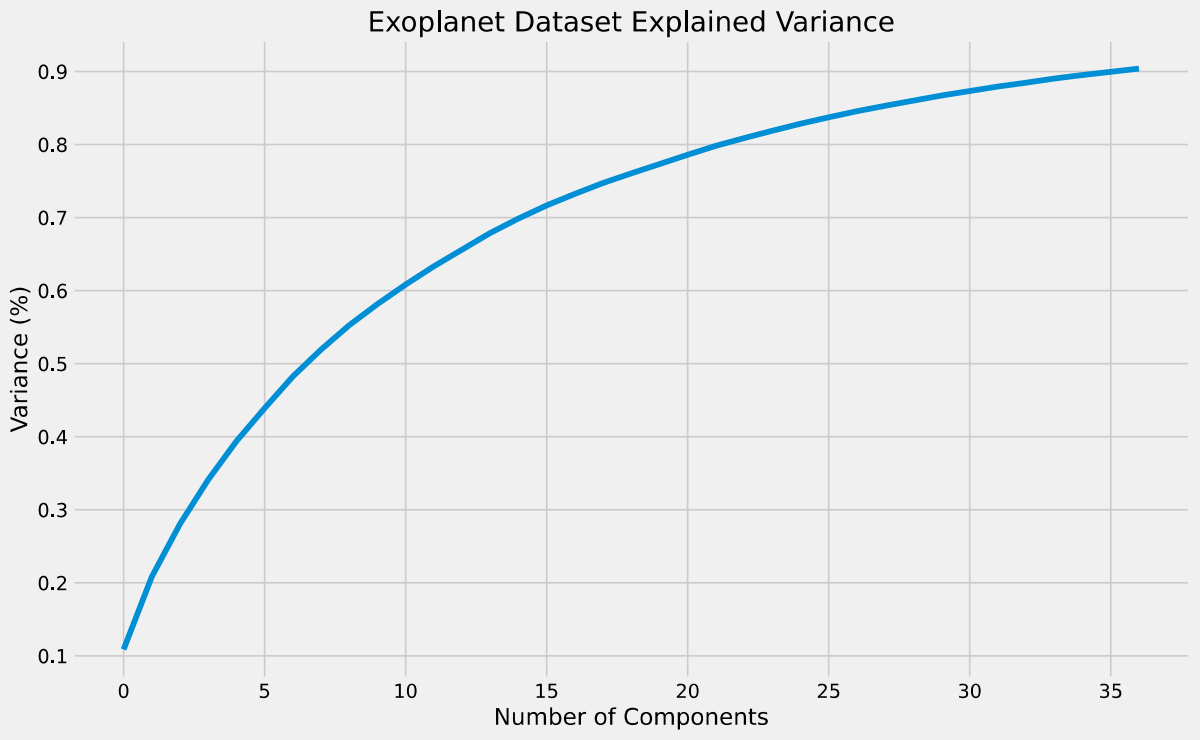
In [13]:
```
#Feature scaling
std_scaler = StandardScaler()
x_train = scaled = std_scaler.fit_transform(x_train)
x_test = std_scaler.fit_transform(x_test)
```

Dimentioanlity reduction

In [14]:
```
#Dimentioanlity reduction
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 < 0.90:
    current_variance += pca.explained_variance_[k]
    k=k+1
```

Apply PCA with 37 componenets

In [15]:
```
#Apply PCA with n_componenets
pca = PCA(n_components=37)
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()
```



Resampling

In [16]:
```
#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
)))
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
)))
```

```
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
```

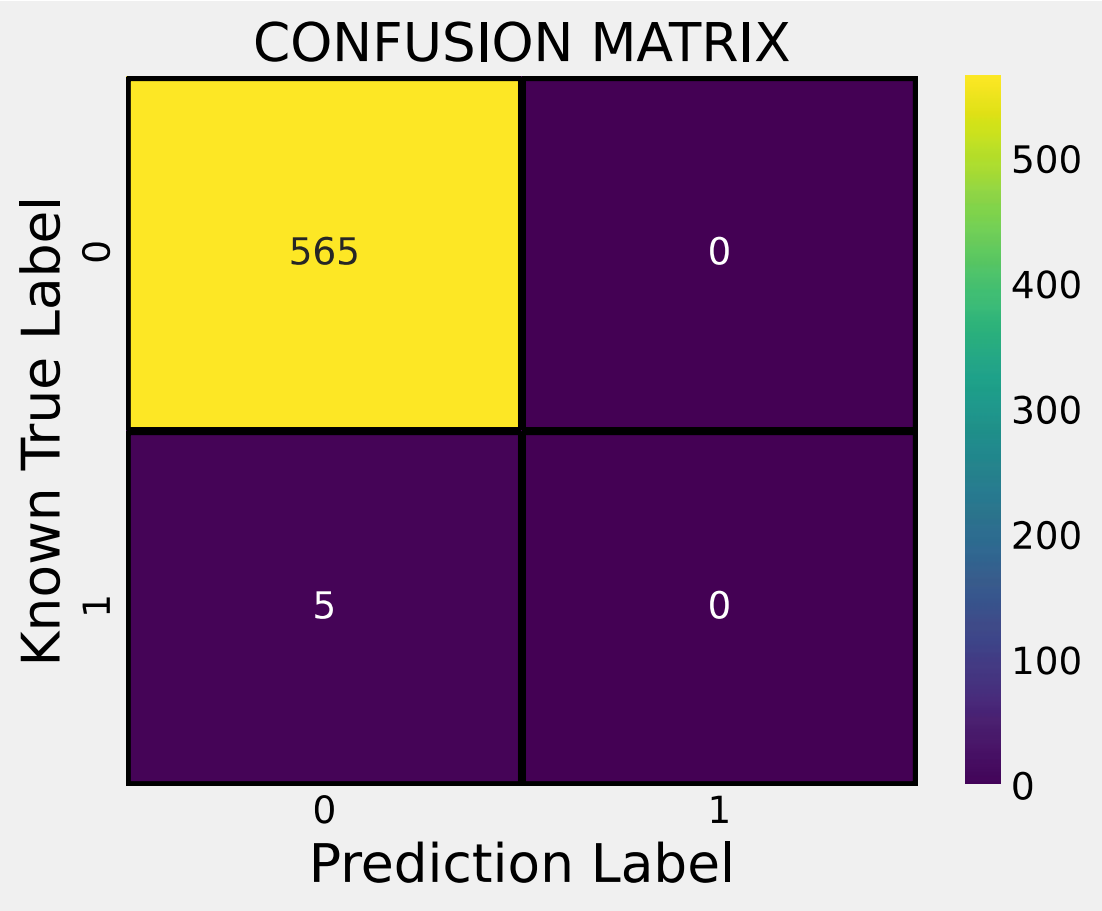
Setting up a function to implement models

In [17]:
```
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,predic
tions)))

    #Confusion matrix
    plt.figure(figsize=(13,10))
    plt.subplot(221)
    sns.heatmap(confusion_matrix(dtest_y,predictions),annot=True,cmap="viridi
s",fmt = "d",linecolor="k",linewidths=3)
    plt.title("CONFUSION MATRIX",fontsize=20)
    plt.ylabel("Known True Label",fontsize=20)
    plt.xlabel("Prediction Label",fontsize=20)
```

Support Vector Machine Algorithm

In [18]:
```
from sklearn.svm import SVC
SVM_model = SVC()
model(SVM_model,x_train_res,y_train_res,x_test,y_test)
```

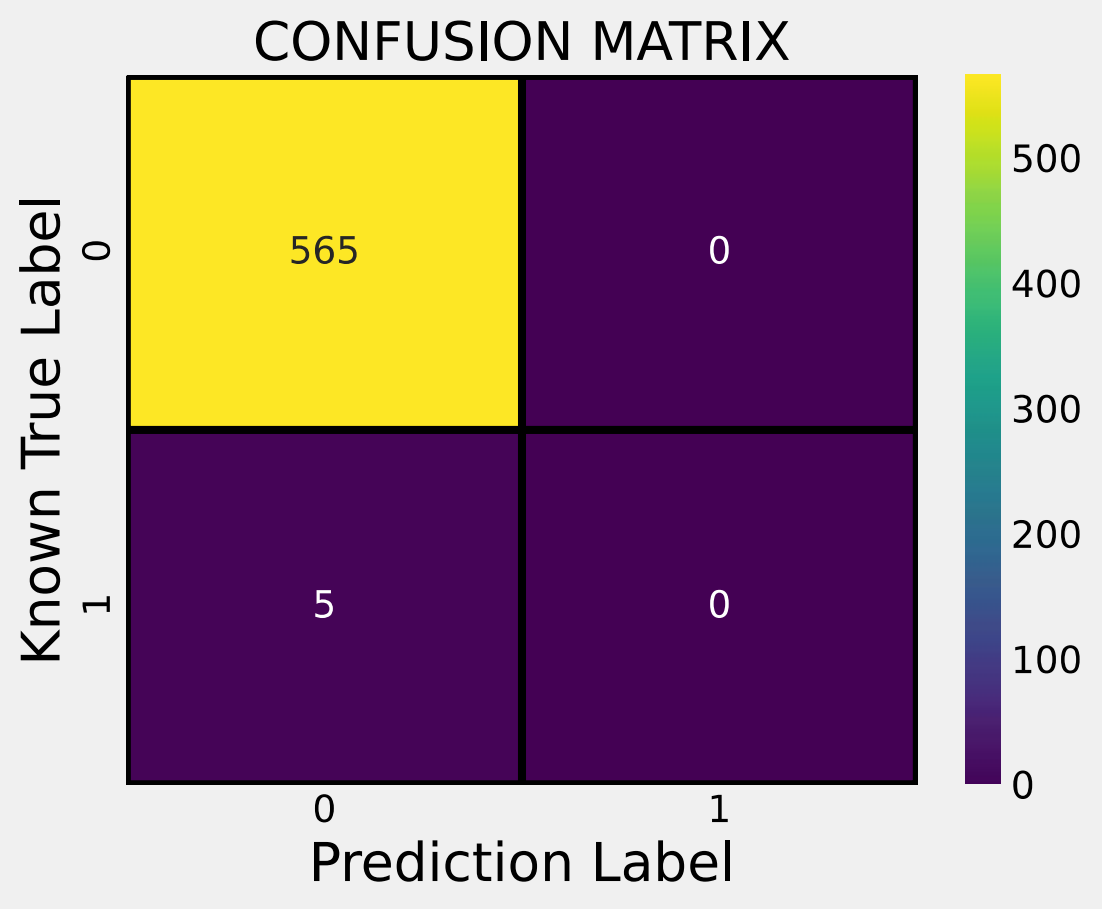
```
Accuracy mean: 0.9991089108910891
Accuracy variance: 0.0017821782178217838

accuracy_score : 0.9912280701754386

classification report :
               precision    recall  f1-score   support

           0       0.99      1.00      1.00       565
           1       0.00      0.00      0.00         5

    accuracy                           0.99       570
   macro avg       0.50      0.50      0.50       570
weighted avg       0.98      0.99      0.99       570
```



Random Forest Classifier

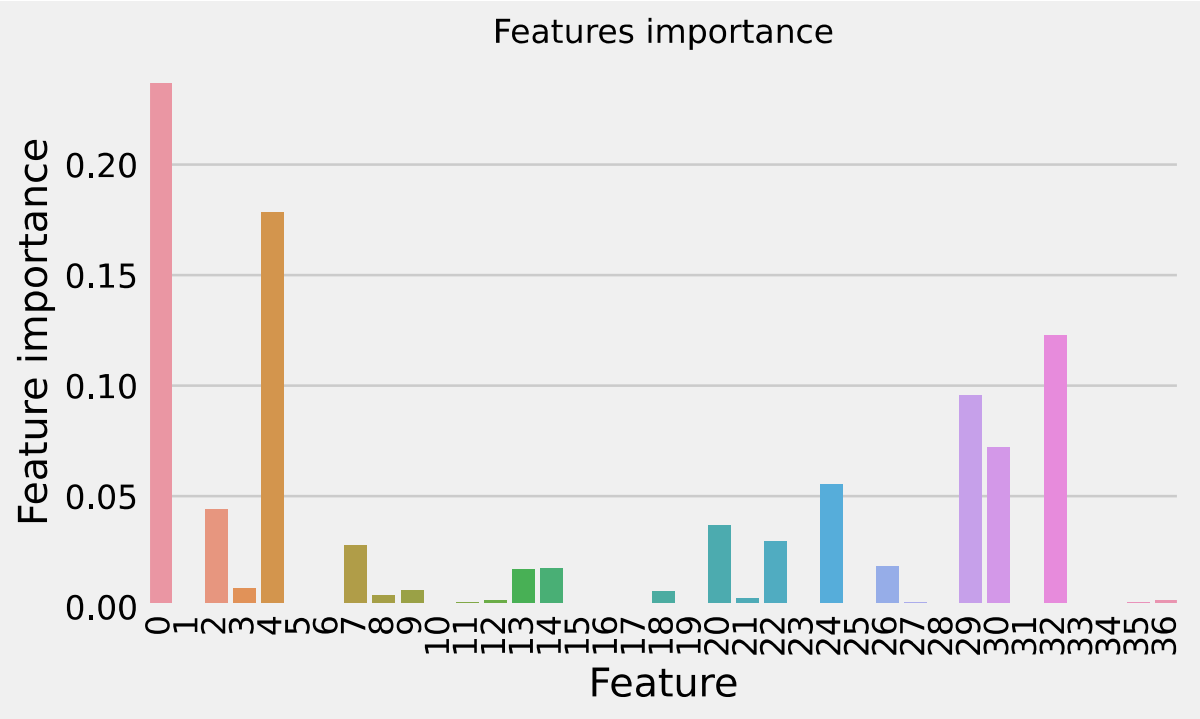
In [19]:
```
from sklearn.ensemble import RandomForestClassifier
rf_classifier = RandomForestClassifier()
model(rf_classifier,x_train_res,y_train_res,x_test,y_test)
```

```
Accuracy mean: 1.0
Accuracy variance: 0.0

accuracy_score : 0.9912280701754386

classification report :
               precision    recall  f1-score   support

           0       0.99      1.00      1.00       565
           1       0.00      0.00      0.00         5

    accuracy                           0.99       570
   macro avg       0.50      0.50      0.50       570
weighted avg       0.98      0.99      0.99       570
```



Random Forest Classifier feature importance

In [26]:
```
#Display feature importance
df1 = pd.DataFrame.from_records(x_train)
tmp = pd.DataFrame({'Feature': df1.columns, 'Feature importance': rf_classifie
r.feature_importances_})
tmp = tmp.sort_values(by='Feature importance',ascending=False)
plt.figure(figsize = (7,4))
plt.title('Features importance',fontsize=14)
s = sns.barplot(x='Feature',y='Feature importance',data=tmp)
s.set_xticklabels(s.get_xticklabels(),rotation=90)
plt.show()
```



Artificial Neural Network

In [27]:
```
from tensorflow import random
random.set_seed(101)
from sklearn.model_selection import cross_val_score
from keras.wrappers.scikit_learn import KerasClassifier
from keras.models import Sequential # initialize neural network library
from keras.layers import Dense # build our layers library
def build_classifier():
    classifier = Sequential() # initialize neural network
    classifier.add(Dense(units = 4, kernel_initializer = 'uniform', activation
= 'relu', input_dim = x_train_res.shape[1]))
    classifier.add(Dense(units = 4, kernel_initializer = 'uniform', activation
= 'relu'))
    classifier.add(Dense(units = 1, kernel_initializer = 'uniform', activation
= 'sigmoid'))
    classifier.compile(optimizer = 'adam', loss = 'binary_crossentropy', metri
cs = ['accuracy'])
    return classifier

classifier = KerasClassifier(build_fn = build_classifier, epochs = 80)
accuracies = cross_val_score(estimator = classifier, X = x_train_res, y = y_tr
ain_res, cv = 5, n_jobs = -1)
mean = accuracies.mean()
variance = accuracies.std()
print("Accuracy mean: "+ str(mean))
print("Accuracy variance: "+ str(variance))
#Accuracy mean: 0.9186138613861387
#Accuracy variance: 0.07308084375906461
```

```
Accuracy mean: 0.8503960371017456
Accuracy variance: 0.19590864221135496
```

Fitting with ANN

In [28]:
```
classifier.fit(x_train_res, y_train_res)
```

```
Epoch 1/80
316/316 [==============================] - 1s 2ms/step - loss: 0.5018 - accur
acy: 0.9501
Epoch 2/80
316/316 [==============================] - 0s 1ms/step - loss: 0.0319 - accur
acy: 0.9971
Epoch 3/80
316/316 [==============================] - 0s 1ms/step - loss: 0.0067 - accur
acy: 0.9998
Epoch 4/80
316/316 [==============================] - 0s 1ms/step - loss: 0.0034 - accur
acy: 0.9999
Epoch 5/80
316/316 [==============================] - 0s 1ms/step - loss: 0.0025 - accur
acy: 0.9997
Epoch 6/80
316/316 [==============================] - 0s 1ms/step - loss: 0.0010 - accur
acy: 0.9999
Epoch 7/80
316/316 [==============================] - 0s 1ms/step - loss: 0.0013 - accur
acy: 0.9995
Epoch 8/80
316/316 [==============================] - 0s 1ms/step - loss: 9.1600e-04 - a
ccuracy: 0.9999
Epoch 9/80
316/316 [==============================] - 0s 1ms/step - loss: 5.0807e-04 - a
ccuracy: 0.9999
Epoch 10/80
316/316 [==============================] - 0s 721us/step - loss: 2.5303e-04 -
accuracy: 1.0000
Epoch 11/80
316/316 [==============================] - 0s 626us/step - loss: 4.4165e-04 -
accuracy: 0.9999
Epoch 12/80
316/316 [==============================] - 0s 712us/step - loss: 3.4880e-04 -
accuracy: 0.9998
Epoch 13/80
316/316 [==============================] - 0s 624us/step - loss: 0.0023 - acc
uracy: 0.9995
Epoch 14/80
316/316 [==============================] - 0s 678us/step - loss: 1.7431e-04 -
accuracy: 1.0000
Epoch 15/80
316/316 [==============================] - 0s 639us/step - loss: 3.4977e-04 -
accuracy: 0.9999
Epoch 16/80
316/316 [==============================] - 0s 706us/step - loss: 2.2357e-04 -
accuracy: 0.9999
Epoch 17/80
316/316 [==============================] - 0s 689us/step - loss: 8.8317e-05 -
accuracy: 1.0000
Epoch 18/80
316/316 [==============================] - 0s 953us/step - loss: 1.0921e-04 -
accuracy: 1.0000
Epoch 19/80
316/316 [==============================] - 0s 625us/step - loss: 1.4045e-04 -
accuracy: 1.0000
```

```
Epoch 20/80
316/316 [==============================] - 0s 643us/step - loss: 9.9160e-05 -
accuracy: 1.0000
Epoch 21/80
316/316 [==============================] - 0s 674us/step - loss: 7.1807e-05 -
accuracy: 1.0000
Epoch 22/80
316/316 [==============================] - 0s 706us/step - loss: 1.1636e-04 -
accuracy: 1.0000
Epoch 23/80
316/316 [==============================] - 0s 804us/step - loss: 0.0019 - acc
uracy: 0.9997
Epoch 24/80
316/316 [==============================] - 0s 656us/step - loss: 3.0199e-04 -
accuracy: 0.9999
Epoch 25/80
316/316 [==============================] - 0s 626us/step - loss: 1.1479e-04 -
accuracy: 1.0000
Epoch 26/80
316/316 [==============================] - 0s 730us/step - loss: 3.3195e-04 -
accuracy: 0.9999
Epoch 27/80
316/316 [==============================] - 0s 652us/step - loss: 8.6235e-04 -
accuracy: 0.9998
Epoch 28/80
316/316 [==============================] - 0s 784us/step - loss: 3.0368e-05 -
accuracy: 1.0000
Epoch 29/80
316/316 [==============================] - 0s 836us/step - loss: 5.2994e-05 -
accuracy: 1.0000
Epoch 30/80
316/316 [==============================] - 0s 684us/step - loss: 2.6094e-05 -
accuracy: 1.0000
Epoch 31/80
316/316 [==============================] - 0s 625us/step - loss: 2.6183e-05 -
accuracy: 1.0000
Epoch 32/80
316/316 [==============================] - 0s 640us/step - loss: 1.6364e-05 -
accuracy: 1.0000
Epoch 33/80
316/316 [==============================] - 0s 879us/step - loss: 3.0445e-05 -
accuracy: 1.0000
Epoch 34/80
316/316 [==============================] - 0s 689us/step - loss: 1.5093e-05 -
accuracy: 1.0000
Epoch 35/80
316/316 [==============================] - 0s 712us/step - loss: 2.8225e-04 -
accuracy: 0.9999
Epoch 36/80
316/316 [==============================] - 0s 619us/step - loss: 5.6513e-04 -
accuracy: 0.9998
Epoch 37/80
316/316 [==============================] - 0s 625us/step - loss: 2.3812e-04 -
accuracy: 0.9999
Epoch 38/80
316/316 [==============================] - 0s 664us/step - loss: 3.5686e-05 -
accuracy: 1.0000
```

```
Epoch 39/80
316/316 [==============================] - 0s 741us/step - loss: 1.7889e-05 -
accuracy: 1.0000
Epoch 40/80
316/316 [==============================] - 0s 788us/step - loss: 1.2250e-05 -
accuracy: 1.0000
Epoch 41/80
316/316 [==============================] - 0s 742us/step - loss: 1.7061e-05 -
accuracy: 1.0000
Epoch 42/80
316/316 [==============================] - 0s 672us/step - loss: 8.7042e-06 -
accuracy: 1.0000
Epoch 43/80
316/316 [==============================] - 0s 639us/step - loss: 1.2682e-05 -
accuracy: 1.0000
Epoch 44/80
316/316 [==============================] - 0s 743us/step - loss: 8.8582e-06 -
accuracy: 1.0000
Epoch 45/80
316/316 [==============================] - 0s 729us/step - loss: 1.3620e-05 -
accuracy: 1.0000
Epoch 46/80
316/316 [==============================] - 0s 836us/step - loss: 1.2321e-05 -
accuracy: 1.0000
Epoch 47/80
316/316 [==============================] - 0s 655us/step - loss: 8.6976e-06 -
accuracy: 1.0000
Epoch 48/80
316/316 [==============================] - 0s 690us/step - loss: 7.1898e-05 -
accuracy: 1.0000
Epoch 49/80
316/316 [==============================] - 0s 623us/step - loss: 3.6283e-06 -
accuracy: 1.0000
Epoch 50/80
316/316 [==============================] - 0s 931us/step - loss: 1.0788e-05 -
accuracy: 1.0000
Epoch 51/80
316/316 [==============================] - 0s 773us/step - loss: 4.5727e-06 -
accuracy: 1.0000
Epoch 52/80
316/316 [==============================] - 0s 703us/step - loss: 3.1292e-06 -
accuracy: 1.0000
Epoch 53/80
316/316 [==============================] - 0s 706us/step - loss: 6.1206e-06 -
accuracy: 1.0000
Epoch 54/80
316/316 [==============================] - 0s 686us/step - loss: 7.2567e-06 -
accuracy: 1.0000
Epoch 55/80
316/316 [==============================] - 0s 702us/step - loss: 6.2655e-06 -
accuracy: 1.0000
Epoch 56/80
316/316 [==============================] - 0s 858us/step - loss: 0.0014 - acc
uracy: 0.9997
Epoch 57/80
316/316 [==============================] - 0s 719us/step - loss: 1.9423e-05 -
accuracy: 1.0000
```

```
Epoch 58/80
316/316 [==============================] - 0s 674us/step - loss: 2.6555e-06 -
accuracy: 1.0000
Epoch 59/80
316/316 [==============================] - 0s 711us/step - loss: 3.6588e-06 -
accuracy: 1.0000
Epoch 60/80
316/316 [==============================] - 0s 945us/step - loss: 2.7716e-06 -
accuracy: 1.0000
Epoch 61/80
316/316 [==============================] - 0s 750us/step - loss: 5.8305e-06 -
accuracy: 1.0000
Epoch 62/80
316/316 [==============================] - 0s 683us/step - loss: 2.0050e-06 -
accuracy: 1.0000
Epoch 63/80
316/316 [==============================] - 0s 621us/step - loss: 3.4462e-06 -
accuracy: 1.0000
Epoch 64/80
316/316 [==============================] - 0s 728us/step - loss: 2.0067e-06 -
accuracy: 1.0000
Epoch 65/80
316/316 [==============================] - 0s 674us/step - loss: 4.5086e-06 -
accuracy: 1.0000
Epoch 66/80
316/316 [==============================] - 0s 729us/step - loss: 1.3803e-06 -
accuracy: 1.0000
Epoch 67/80
316/316 [==============================] - 0s 683us/step - loss: 2.5751e-06 -
accuracy: 1.0000
Epoch 68/80
316/316 [==============================] - 0s 639us/step - loss: 2.5569e-06 -
accuracy: 1.0000
Epoch 69/80
316/316 [==============================] - 0s 831us/step - loss: 1.5361e-06 -
accuracy: 1.0000
Epoch 70/80
316/316 [==============================] - 0s 605us/step - loss: 2.3553e-06 -
accuracy: 1.0000
Epoch 71/80
316/316 [==============================] - 0s 638us/step - loss: 1.8753e-06 -
accuracy: 1.0000
Epoch 72/80
316/316 [==============================] - 0s 773us/step - loss: 1.4478e-06 -
accuracy: 1.0000
Epoch 73/80
316/316 [==============================] - 0s 712us/step - loss: 3.0919e-06 -
accuracy: 1.0000
Epoch 74/80
316/316 [==============================] - 0s 611us/step - loss: 1.8484e-06 -
accuracy: 1.0000
Epoch 75/80
316/316 [==============================] - 0s 660us/step - loss: 2.4278e-06 -
accuracy: 1.0000
Epoch 76/80
316/316 [==============================] - 0s 633us/step - loss: 8.3386e-07 -
accuracy: 1.0000
```

```
Epoch 77/80
316/316 [==============================] - 0s 940us/step - loss: 0.0015 - acc
uracy: 0.9999
Epoch 78/80
316/316 [==============================] - 0s 743us/step - loss: 1.4905e-06 -
accuracy: 1.0000
Epoch 79/80
316/316 [==============================] - 0s 636us/step - loss: 1.1769e-06 -
accuracy: 1.0000
Epoch 80/80
316/316 [==============================] - 0s 703us/step - loss: 1.5811e-06 -
accuracy: 1.0000
```

Out[28]: <tensorflow.python.keras.callbacks.History at 0x12a8e70de08>

Confusion Matrix for ANN

In [29]:
```
# Confusion Matrix for ANN
plt.figure(figsize=(13,10))
plt.subplot(221)
sns.heatmap(confusion_matrix(y_test,classifier.predict(x_test)),annot=True,cma
p="viridis",fmt = "d",linecolor="k",linewidths=3)
plt.title("ANN CONFUSION MATRIX",fontsize=20)
plt.ylabel("Known True Label",fontsize=20)
plt.xlabel("Prediction Label",fontsize=20)
```

Out[29]: Text(0.5, 347.0090909090909, 'Prediction Label')