Exo -Planet Hunting using Machine Learning

Name: Bangaru Mohnish

Reg.No: RA1711003011397

School: SRM IST

Abstract

Are We Alone?

It has been humanity's most appealing question since the beginning of space exploration. We all know how rare life is in the universe despite having studied several systems we have failed to find any signs of life. Since the universe is very vast studying all of is nearly impossible, therefore the scientists at NASA decided to skim the universe for earth like planets which have a potential for harboring life, these are known an exoplanets and a special telescope called Kepler scans the universe and sends the data to NASA, In this project we are using the Time Series data from Kepler to analyze the luminescence of these planets and categorize them as exo planets or non-exoplanets.

Libraries Used:

1.Keras

2.Numpy

3.Pandas

4.MinMaxScaler form sklearn

5.rcParams for pylab

6.ndimage from scipy

7.Seaborn

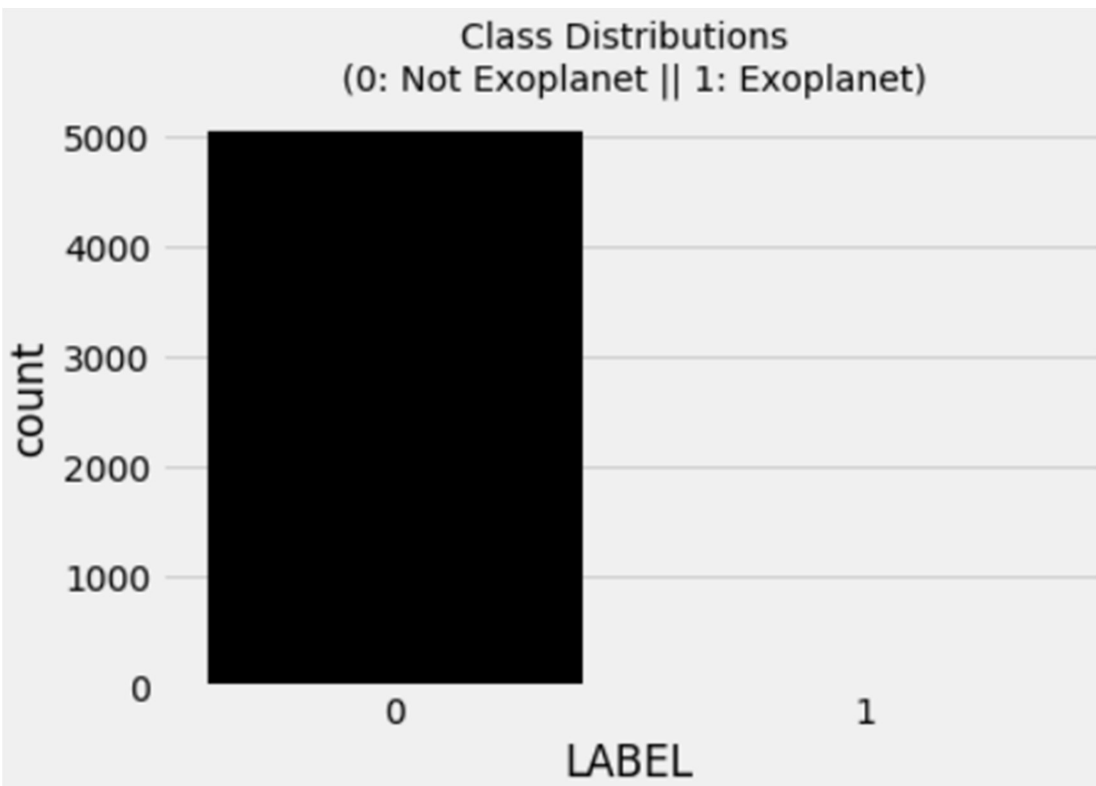
8.Pyplot from Matplotlib

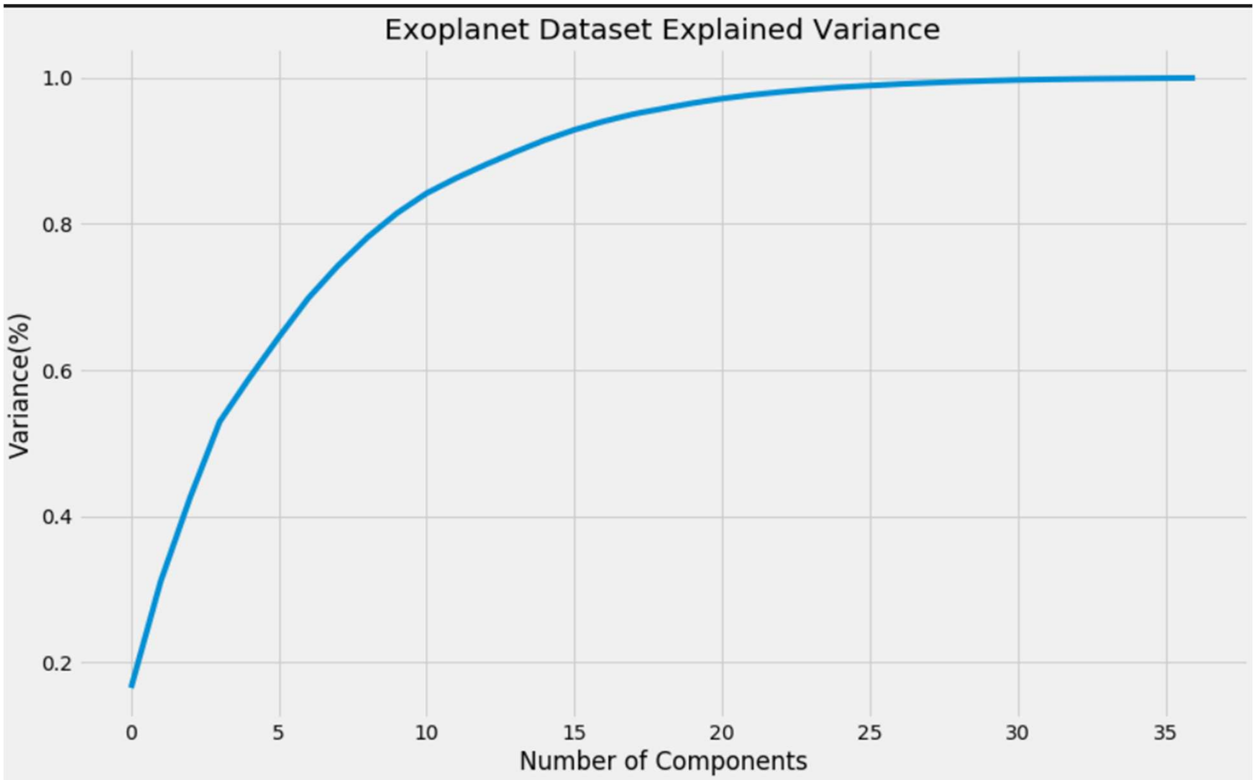
Data Used:

1.Kepler Time series data provided by NASA

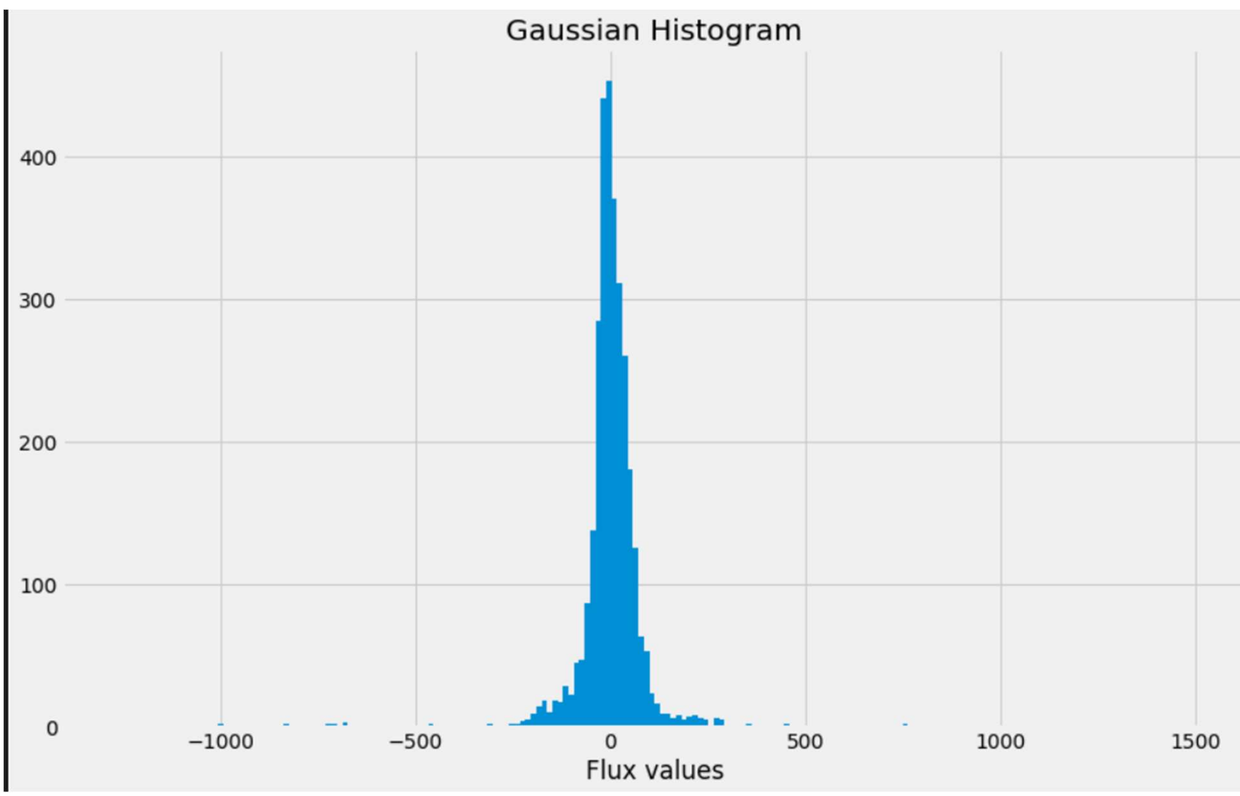
Outputs:

```
Memory usage of dataframe is 6.25 MB
Memory usage after optimization is: 6.25 MB
Decreased by 0.0%
Memory usage of dataframe is 62.04 MB
Memory usage after optimization is: 62.04 MB
Decreased by 0.0%
```

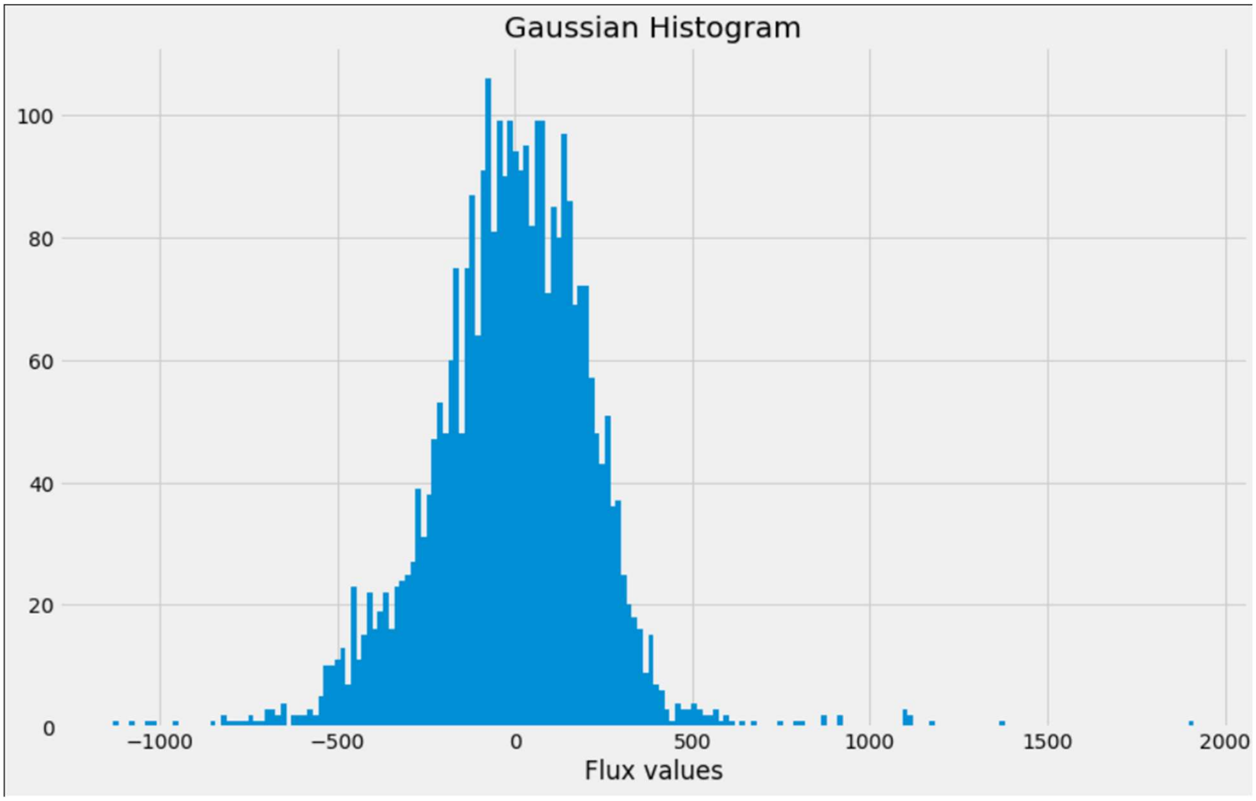


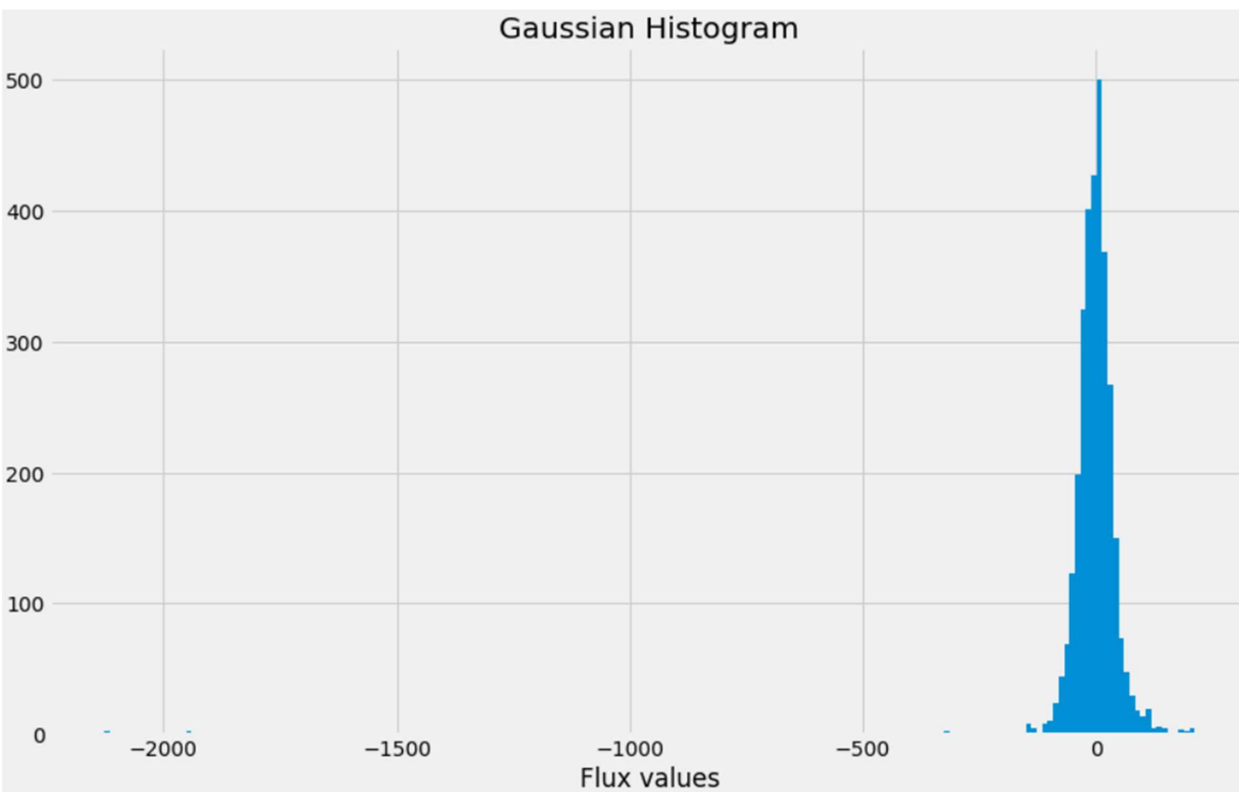


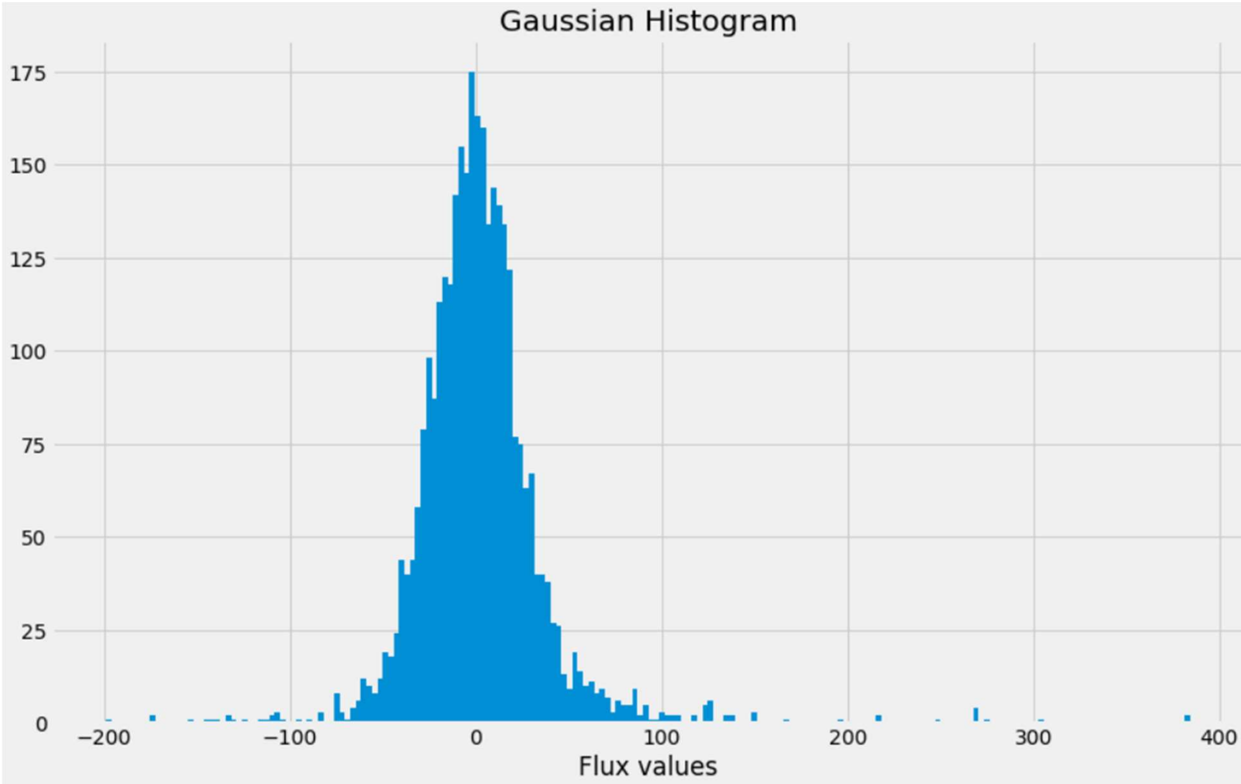
Exoplanet Dataset Explained Variance
Variance(%)
1.0
0.8
0.6
0.4
0.2
0
5
10
15
20
25
30
35
Number of Components

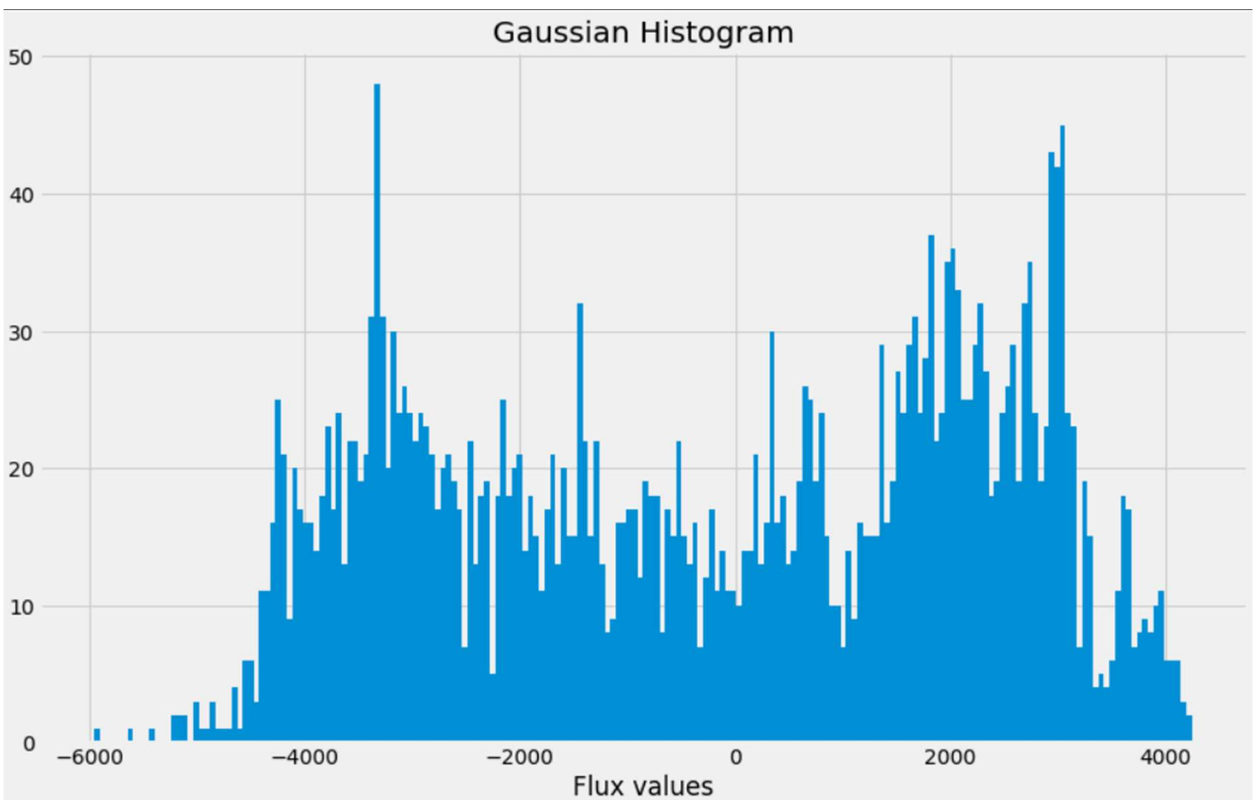


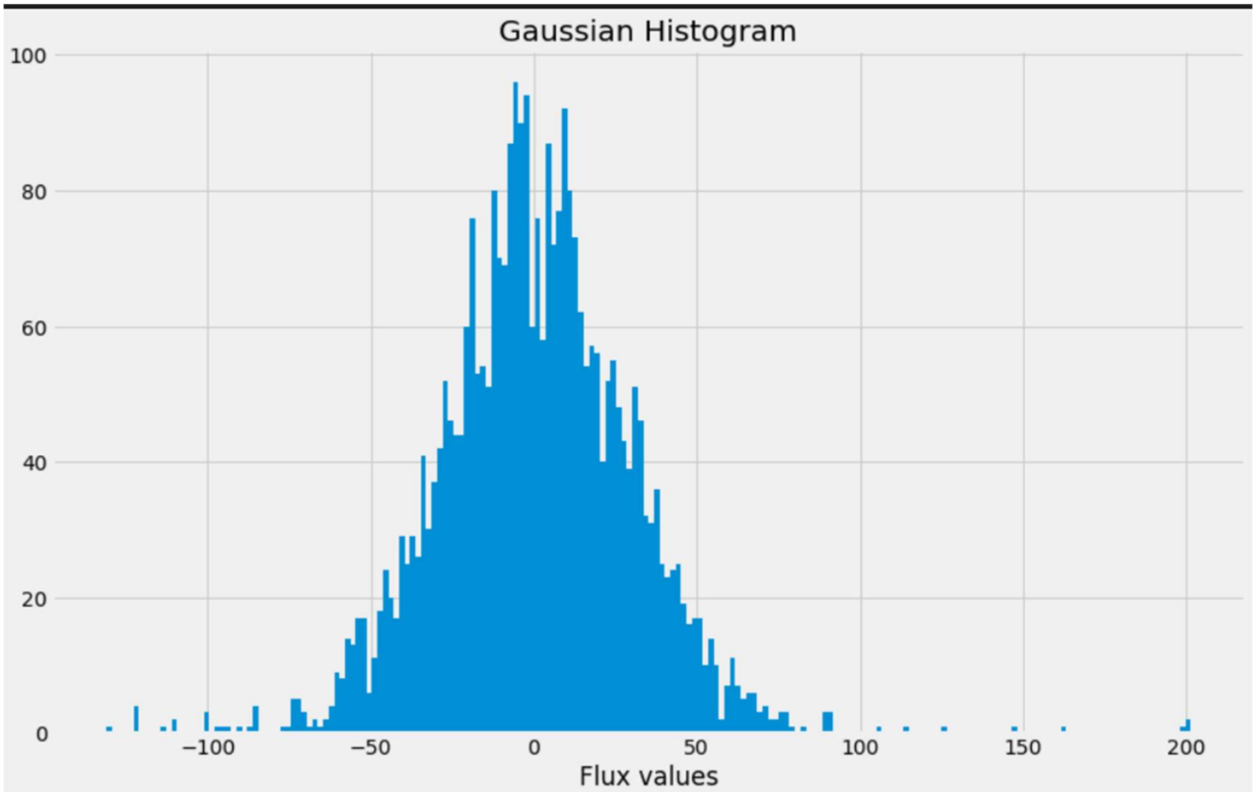
Gaussian Histogram
400
300
200
100
0
−1000
−500
0
500
1000
1500
Flux values

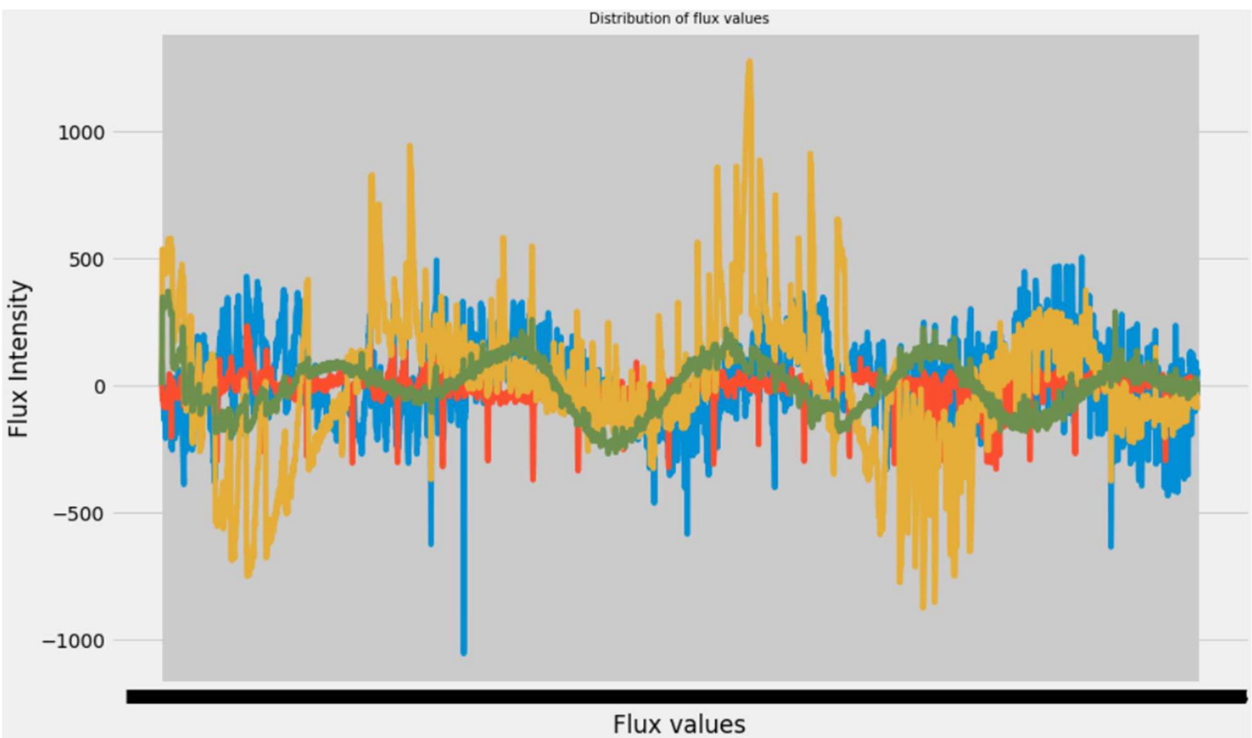

Gaussian Histogram
100
80
60
40
20
0
−1000
−500
0
500
1000
1500
2000
Flux values

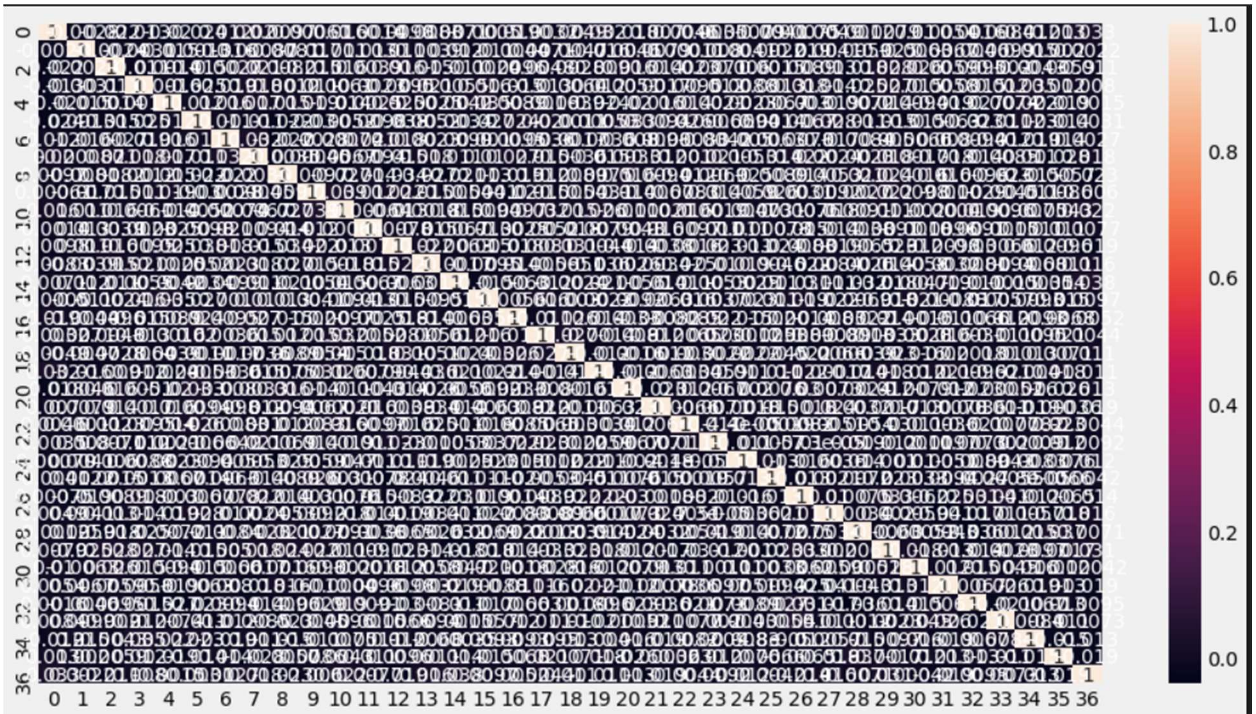

Gaussian Histogram
500
400
300
200
100
0
−2000
−1500
−1000
−500
0
Flux values


Gaussian Histogram
175
150
125
100
75
50
25
0
−200
−100
0
100
200
300
400
Flux values


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


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


Distribution of flux values
1000
500
0
−500
−1000
Flux Intensity
Flux values



Conclusion:

This Project has helped me gain a better insight of the stock market functioning and an approach to solve the problems faced by investors using my technical skills

Appendix:

```
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.metrics import recall_score, classification_report, precision_score,
 confusion_matrix, accuracy_score
from sklearn.preprocessing import StandardScaler, normalize
from scipy import ndimage
import seaborn as sns

test_data = pd.read_csv('D:\Downloads\kepler-labelled-time-series-
data\exoTest.csv')

train_data = pd.read_csv('D:\Downloads\kepler-labelled-time-series-
data\exoTrain.csv')

category = {2: 1, 1:0}
train_data.LABEL = [category[item] for item in train_data.LABEL]
test_data.LABEL = [category[item] for item in test_data.LABEL]

def reduce_mem(df):
    """ iterate through all the columns of a dataframe and modify the data type
        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).ma
x:
```

```
                    df[col] = df[col].astype(np.int8)
                elif c_min > np.iinfo(np.int16).min and c_max < np.iinfo(np.int16
).max:
                    df[col] = df[col].astype(np.int16)
                elif c_min > np.iinfo(np.int32).min and c_max < np.iinfo(np.int32
).max:
                    df[col] = df[col].astype(np.int32)
                elif c_min > np.iinfo(np.int64).min and c_max < np.iinfo(np.int64
).max:
                    df[col] = df[col].astype(np.int64)
            else:
                if c_min > np.finfo(np.float16).min and c_max < np.finfo(np.float
16).max:
                    df[col] = df[col].astype(np.float16)
                elif c_min > np.finfo(np.float32).min and c_max < np.finfo(np.flo
at32).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_mem))
    return df

test_data = reduce_mem(test_data)
train_data = reduce_mem(train_data)

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)', fontsize=1
4)


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,])
```

```python
plt.plot(train_data.iloc[3,])
plt.show()

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()

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()


x_train = train_data.drop(["LABEL"],axis=1)
y_train = train_data["LABEL"]
x_test = test_data.drop(["LABEL"],axis=1)
y_test = test_data["LABEL"]

x_train = normalized = normalize(x_train)
x_test = normalize(x_test)

x_train = filtered = ndimage.filters.gaussian_filter(x_train, sigma =10)
x_test = ndimage.filters.gaussian_filter(x_test, sigma = 10)

std_scaler = StandardScaler()
x_train = scaled = std_scaler.fit_transform(x_train)
x_test = std_scaler.fit_transform(x_test)

from sklearn.decomposition import PCA
pca = PCA()
x_train = pca.fit_transform(x_train)
x_test = pca.fit_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

pca = PCA(n_components=37)
```

```
x_train = pca.fit_transform(x_train)
x_test = pca.fit_transform(x_test)
plt.figure()
plt.plot(np.cumsum(pca.explained_variance_ratio_))
plt.xlabel('Number of Components')
plt.ylabel('Variance(%)')
plt.title('Exoplanet Dataset Explained Variance')
plt.show()

df = pd.DataFrame.from_records(x_train)
corr = df.corr(method = "kendall")
plt.figure(figsize = (15,8))
sns.heatmap(corr, annot=True)
df.columns

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(sampling_strategy = 'not minority',random_state=10)
x_train_res, y_train_res = sm.fit_sample(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)))


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', metrics
= ['accuracy'])
    return classifier

classifier = KerasClassifier(build_fn = build_classifier, epochs = 40)
accuracies = cross_val_score(estimator = classifier, X = x_train_res, y = y_train
_res, cv = 5, n_jobs = -1)
```

```python
mean = accuracies.mean()
variance = accuracies.std()
print("Accuracy mean: "+ str(mean))
print("Accuracy variance: "+ str(variance))
```