Group - 4

Twitter Bot Detection

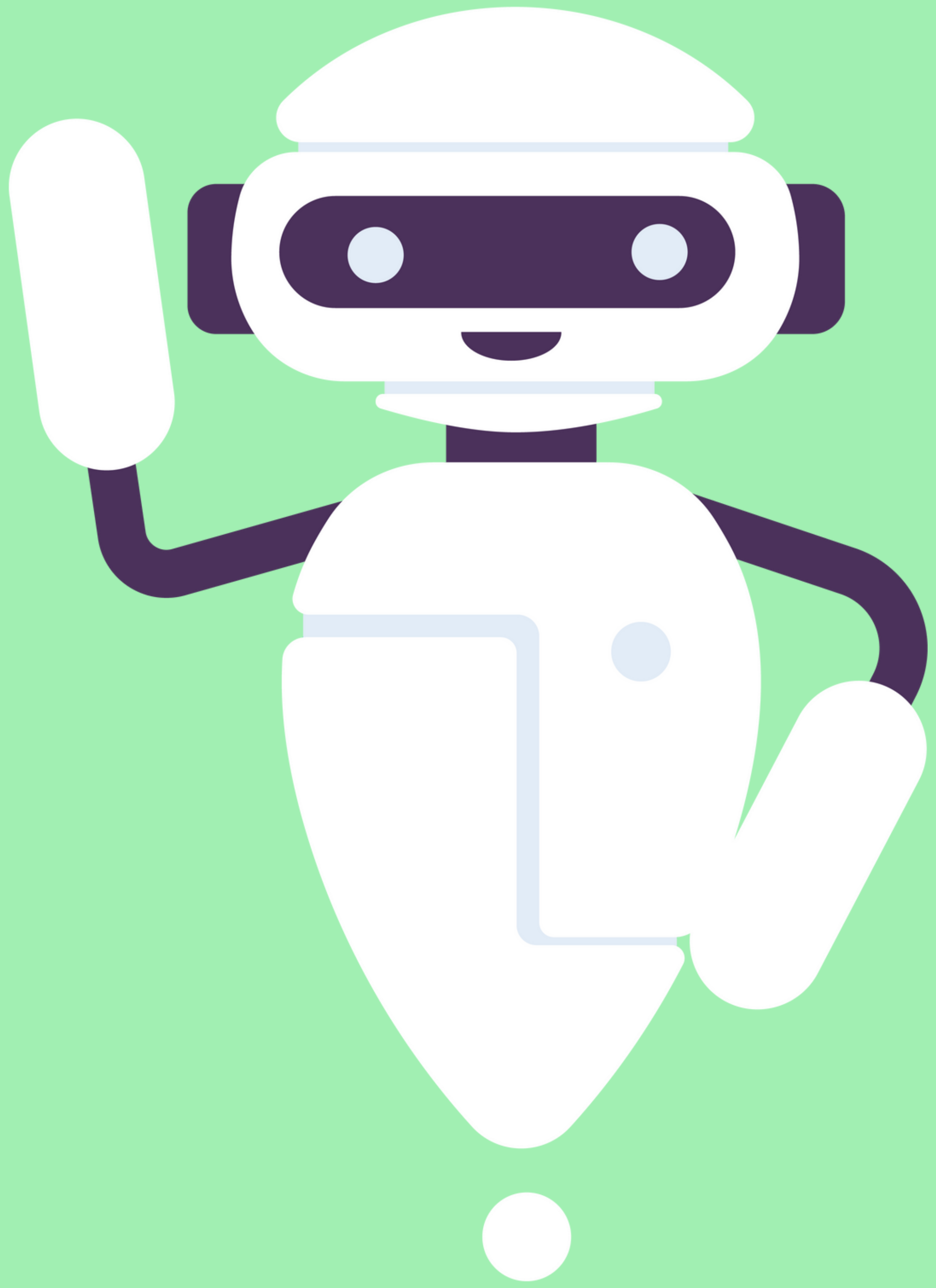
A Machine Learning project for CyberSecurity

INTRODUCTION

In the Internet and social media world, about **3.8 billion active social media users** and **4.5 billion people access the internet daily**. Every year there is a **9%** growth in the number of users and half of the internet traffic consists of mostly bots.

What is a Bot?

A bot -- short for robot and also called an internet bot -- is a **computer program** that operates as an agent for a user or other program or to simulate a human activity. Bots are normally used to automate certain tasks, meaning they can run without specific instructions from humans. Bots are mainly categorized into two categories: **good and bad bots**; good bots consist of web crawlers and chatbots whereas bad bots consist of malicious bots which make up **20%** of the traffic.



What do Bad Bots do?

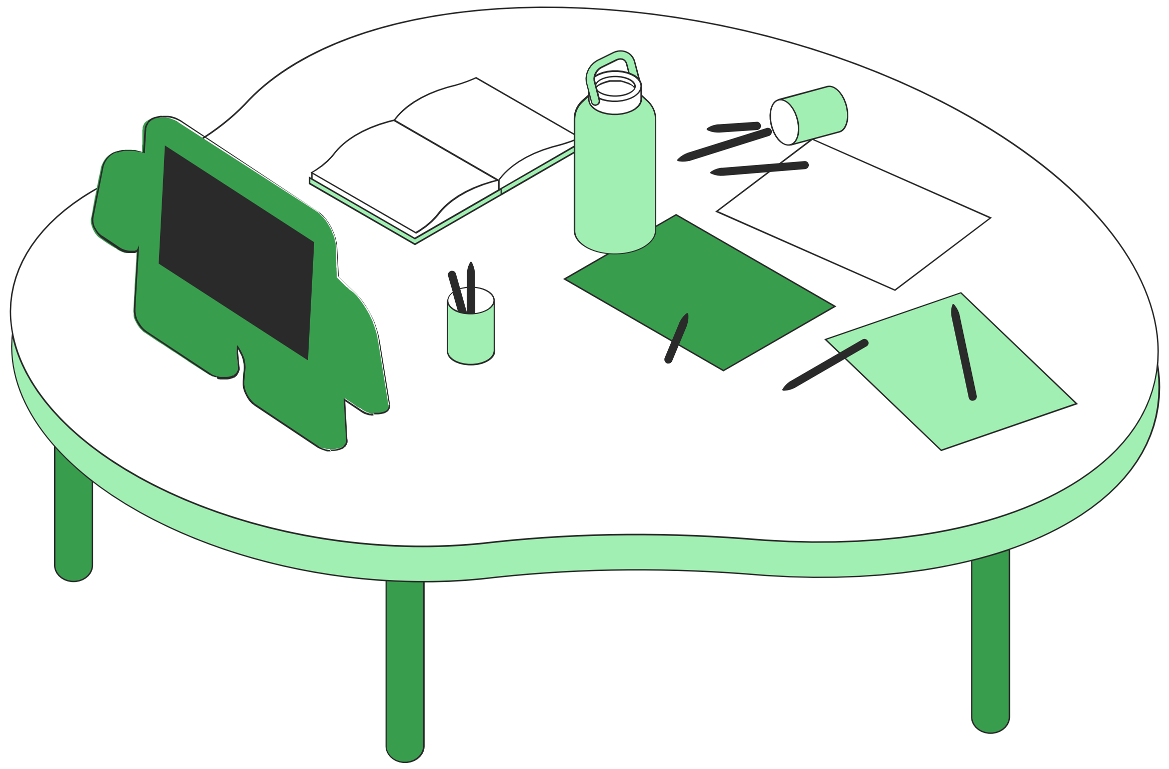
The reason Bad Bots are not good is that they are used for nefarious purposes, they can mimic human behavior, they can impersonate legal traffic, attack IoT devices and exploit their performance. Among all these concerns, the primary concern is for social media users as they represent a large group of active users on the internet, they are more vulnerable to a breach of data, and changes in opinion based on data.

The Need to Act

Detection of such bots is crucial to prevent further mishaps such as cyber-attacks and leaks. We are using three supervised Machine learning techniques in our project namely- Decision Tree, Naïve Bayes, and Random Forest to calculate their accuracies and compare them.

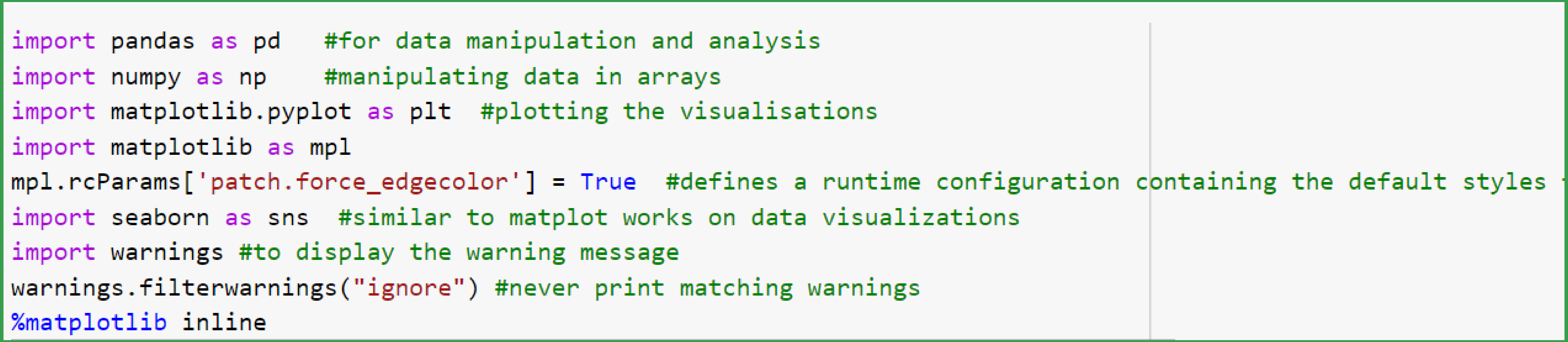
OUR PROJECT

Complete process of how we are detecting bots in Twitter

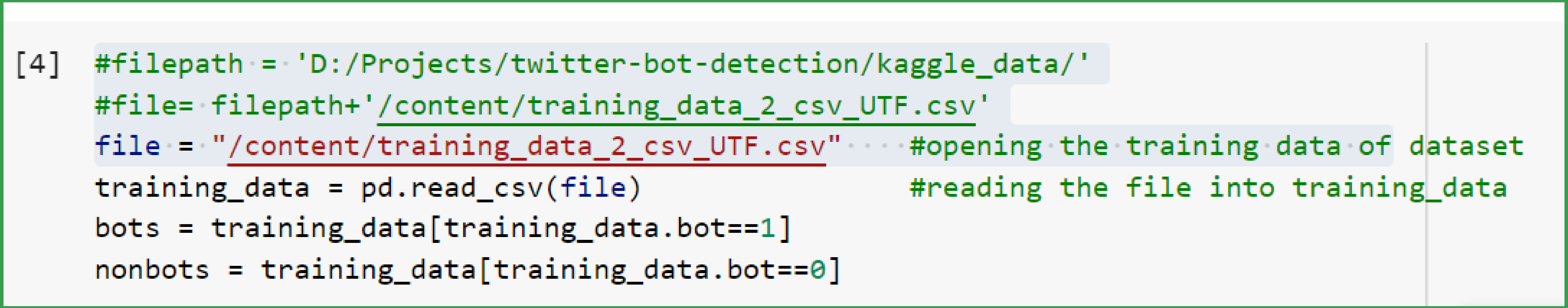


Importing Libraries and Loading dataset
Data Analysis
Identifying Imbalance in Data
Feature Independence and Spearman Correlation
Feature Engineering and Extraction
Decision Tree Model
Random Forest Model
Naive Bayes Model
Final Accuracies and Conclusion

IMPORTING AND LOADING DATASET



```
import pandas as pd   #for data manipulation and analysis
import numpy as np    #manipulating data in arrays
import matplotlib.pyplot as plt  #plotting the visualisations
import matplotlib as mpl
mpl.rcParams['patch.force_edgecolor'] = True  #defines a runtime configuration containing the default styles
import seaborn as sns  #similar to matplot works on data visualizations
import warnings #to display the warning message
warnings.filterwarnings("ignore") #never print matching warnings
%matplotlib inline
```

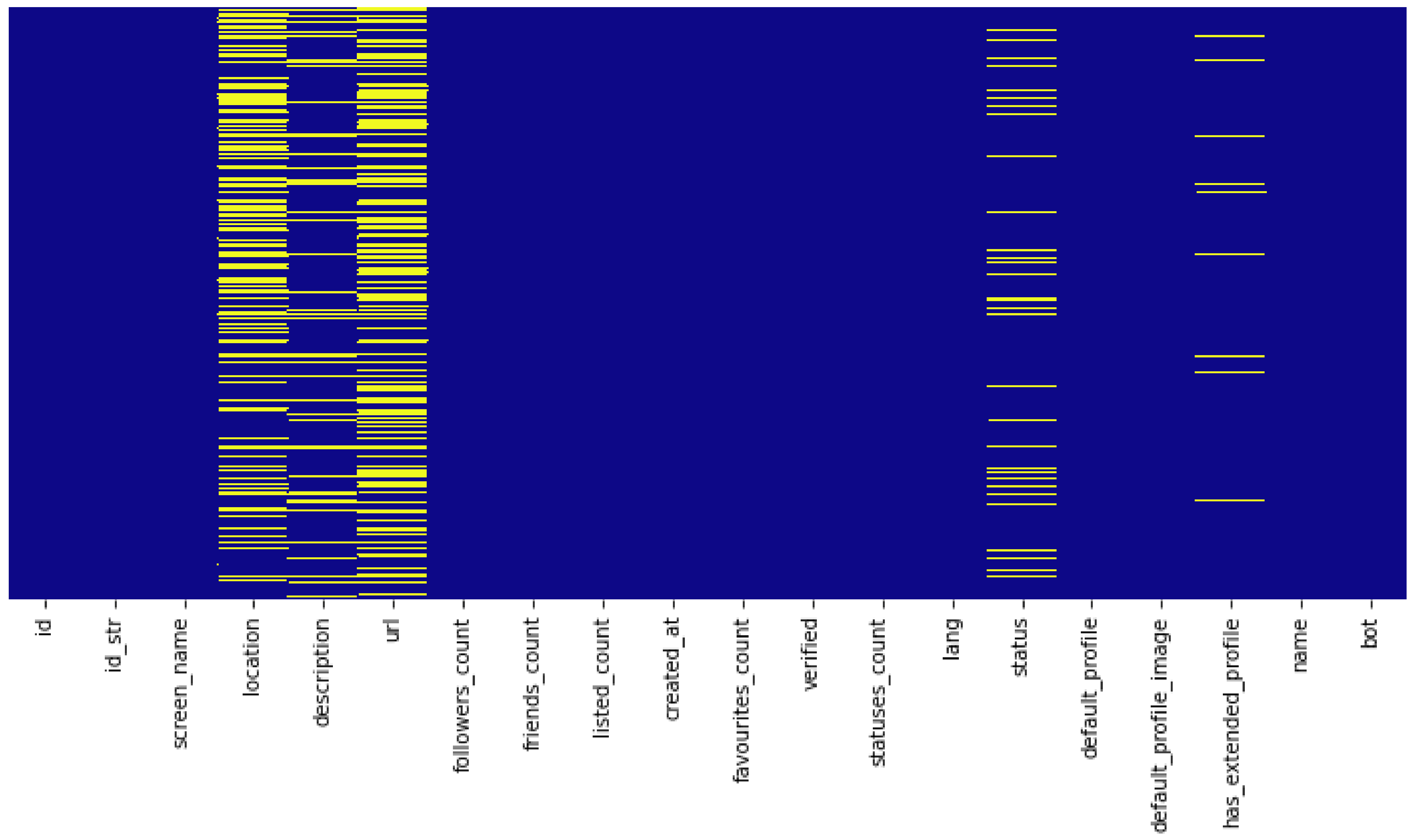


```
[4]  #filepath = 'D:/Projects/twitter-bot-detection/kaggle_data/'
     #file= filepath+'/content/training_data_2_csv_UTF.csv'
     file = "/content/training_data_2_csv_UTF.csv"    #opening the training data of dataset
     training_data = pd.read_csv(file)                #reading the file into training_data
     bots = training_data[training_data.bot==1]
     nonbots = training_data[training_data.bot==0]
```

Identifying Missing data

```
[ ] def get_heatmap(df):
        #This function gives heatmap of all NaN values
        plt.figure(figsize=(10,6))
        sns.heatmap(df.isnull(), yticklabels=False, cbar=False, cmap='plasma')
        plt.tight_layout()
        return plt.show()

    get_heatmap(training_data)
```

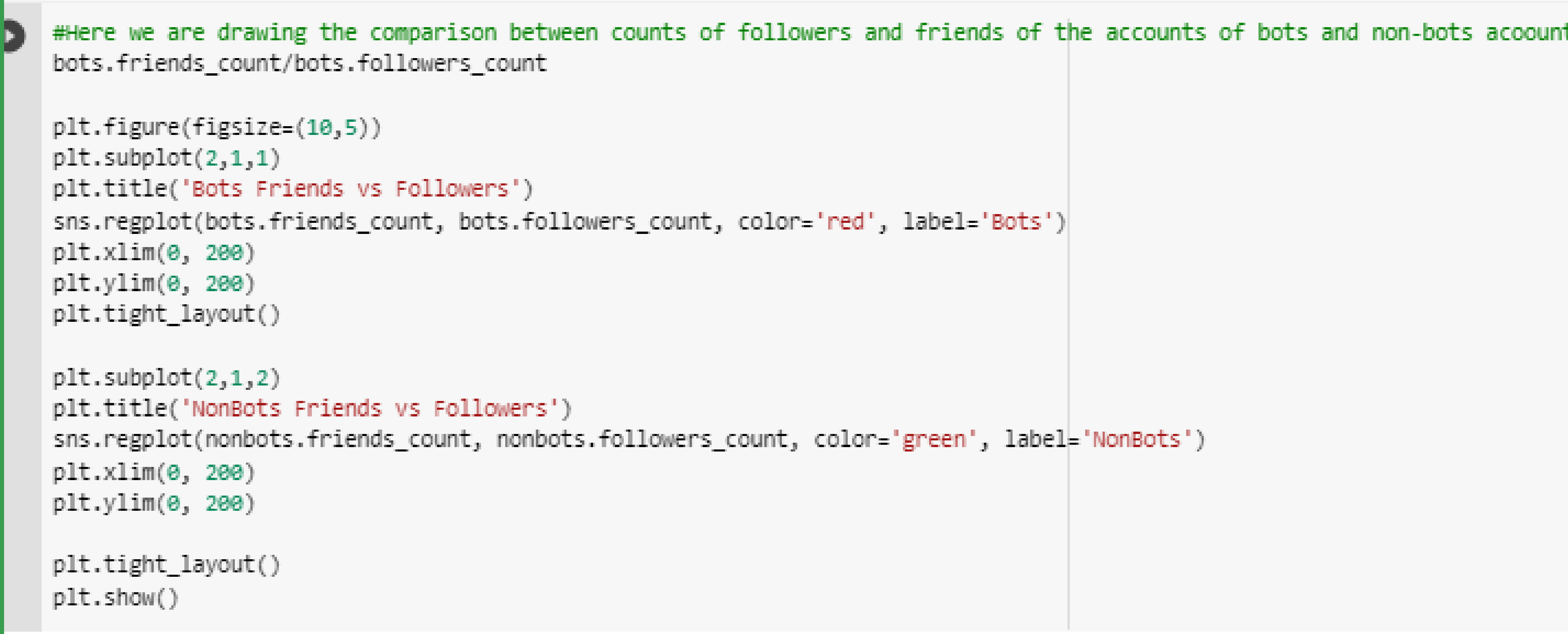


DATA ANALYSIS

IDENTIFYING MISSING VALUES

DATA ANALYSIS

FRIENDS VS FOLLOWERS

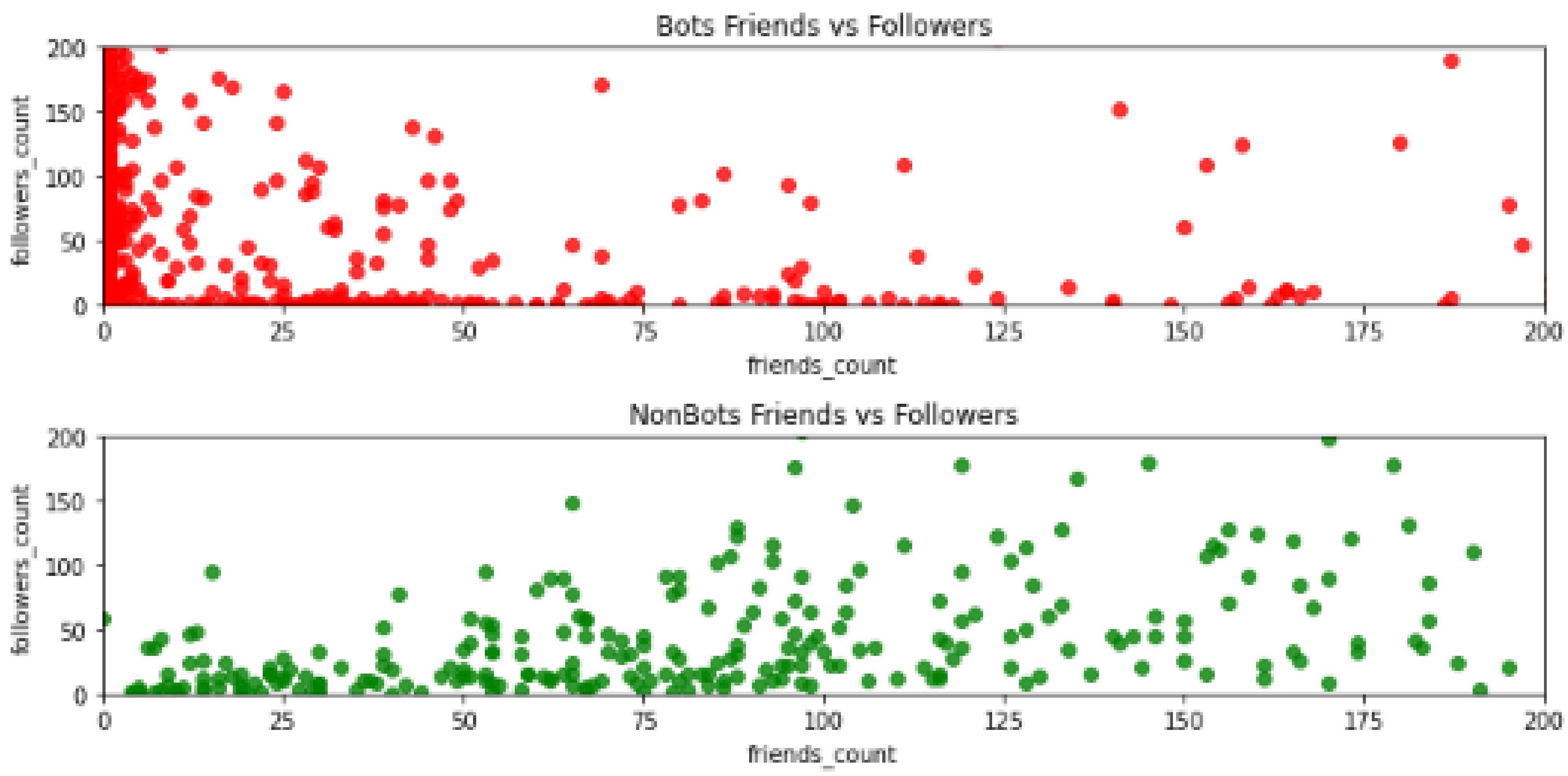


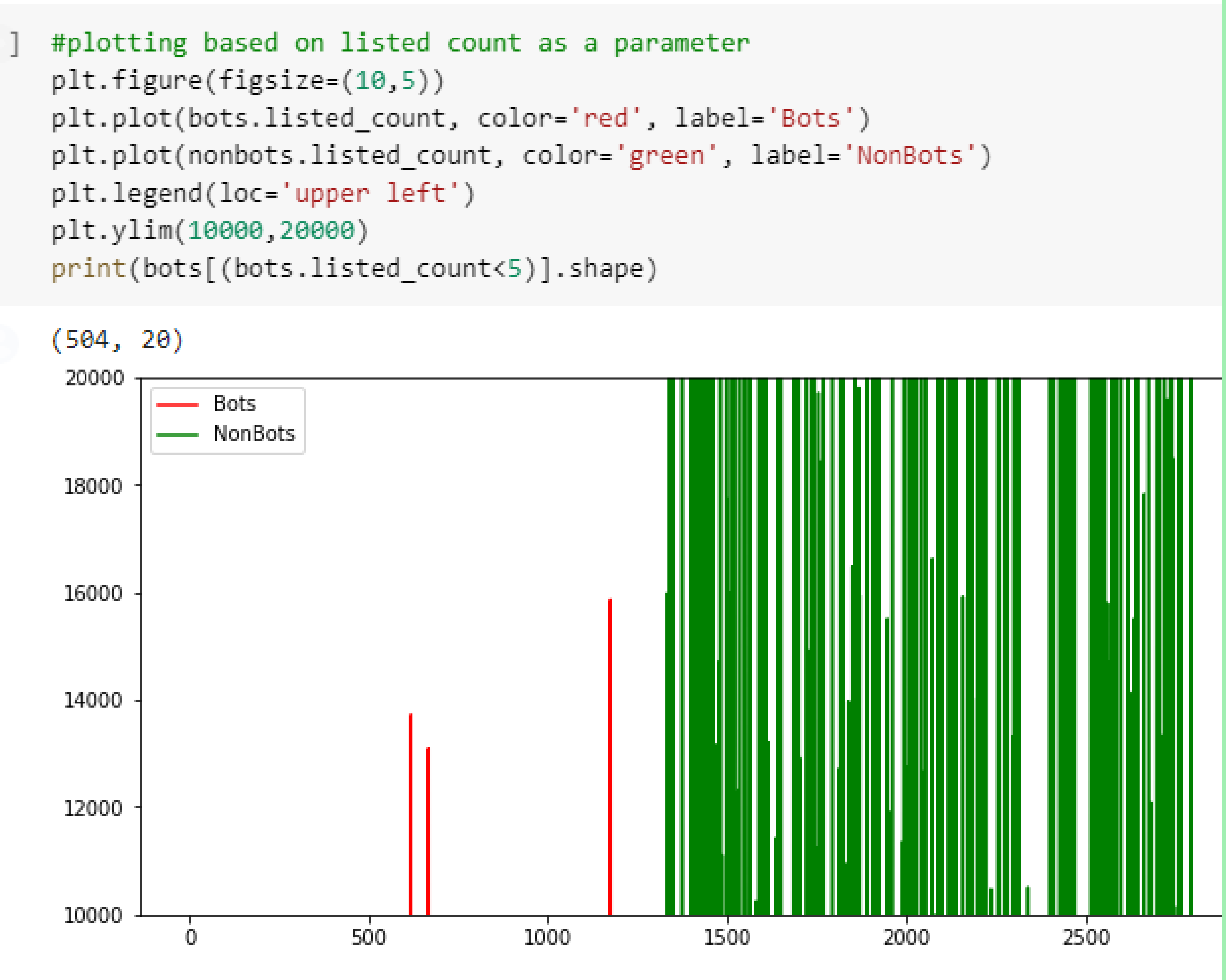
```
#Here we are drawing the comparison between counts of followers and friends of the accounts of bots and non-bots acoount
bots.friends_count/bots.followers_count

plt.figure(figsize=(10,5))
plt.subplot(2,1,1)
plt.title('Bots Friends vs Followers')
sns.regplot(bots.friends_count, bots.followers_count, color='red', label='Bots')
plt.xlim(0, 200)
plt.ylim(0, 200)
plt.tight_layout()

plt.subplot(2,1,2)
plt.title('NonBots Friends vs Followers')
sns.regplot(nonbots.friends_count, nonbots.followers_count, color='green', label='NonBots')
plt.xlim(0, 200)
plt.ylim(0, 200)

plt.tight_layout()
plt.show()
```

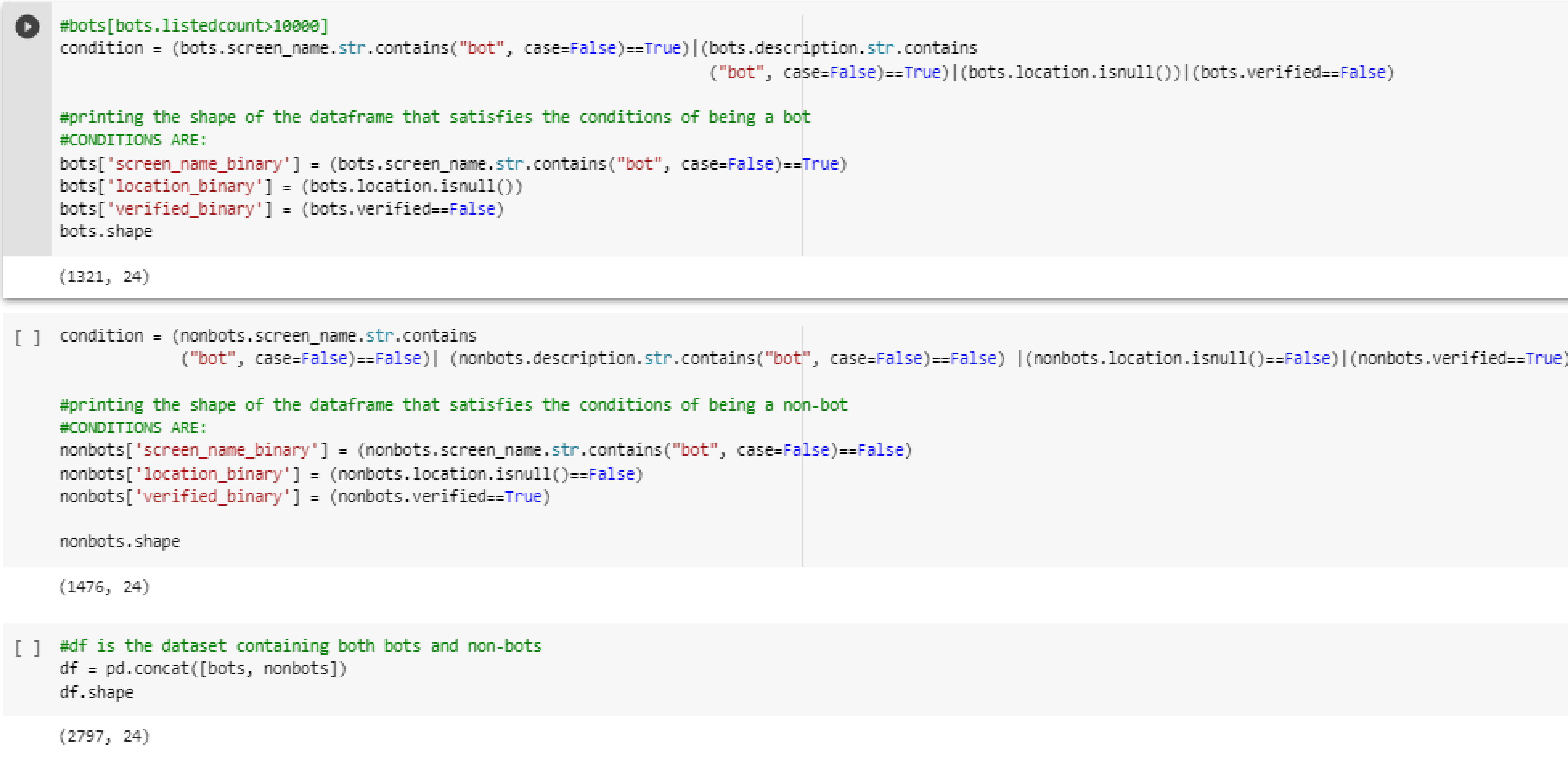




```
]  #plotting based on listed count as a parameter
   plt.figure(figsize=(10,5))
   plt.plot(bots.listed_count, color='red', label='Bots')
   plt.plot(nonbots.listed_count, color='green', label='NonBots')
   plt.legend(loc='upper left')
   plt.ylim(10000,20000)
   print(bots[(bots.listed_count<5)].shape)
```

```
(504, 20)
```

IMBALANCE IN DATA



```
#bots[bots.listedcount>10000]
condition = (bots.screen_name.str.contains("bot", case=False)==True)|(bots.description.str.contains
                                                                      ("bot", case=False)==True)|(bots.location.isnull())|(bots.verified==False)

#printing the shape of the dataframe that satisfies the conditions of being a bot
#CONDITIONS ARE:
bots['screen_name_binary'] = (bots.screen_name.str.contains("bot", case=False)==True)
bots['location_binary'] = (bots.location.isnull())
bots['verified_binary'] = (bots.verified==False)
bots.shape
```

```
(1321, 24)
```

```
[ ]
condition = (nonbots.screen_name.str.contains
             ("bot", case=False)==False)| (nonbots.description.str.contains("bot", case=False)==False) |(nonbots.location.isnull()==False)|(nonbots.verified==True

#printing the shape of the dataframe that satisfies the conditions of being a non-bot
#CONDITIONS ARE:
nonbots['screen_name_binary'] = (nonbots.screen_name.str.contains("bot", case=False)==False)
nonbots['location_binary'] = (nonbots.location.isnull()==False)
nonbots['verified_binary'] = (nonbots.verified==True)

nonbots.shape
```

```
(1476, 24)
```

```
[ ]
#df is the dataset containing both bots and non-bots
df = pd.concat([bots, nonbots])
df.shape
```

```
(2797, 24)
```

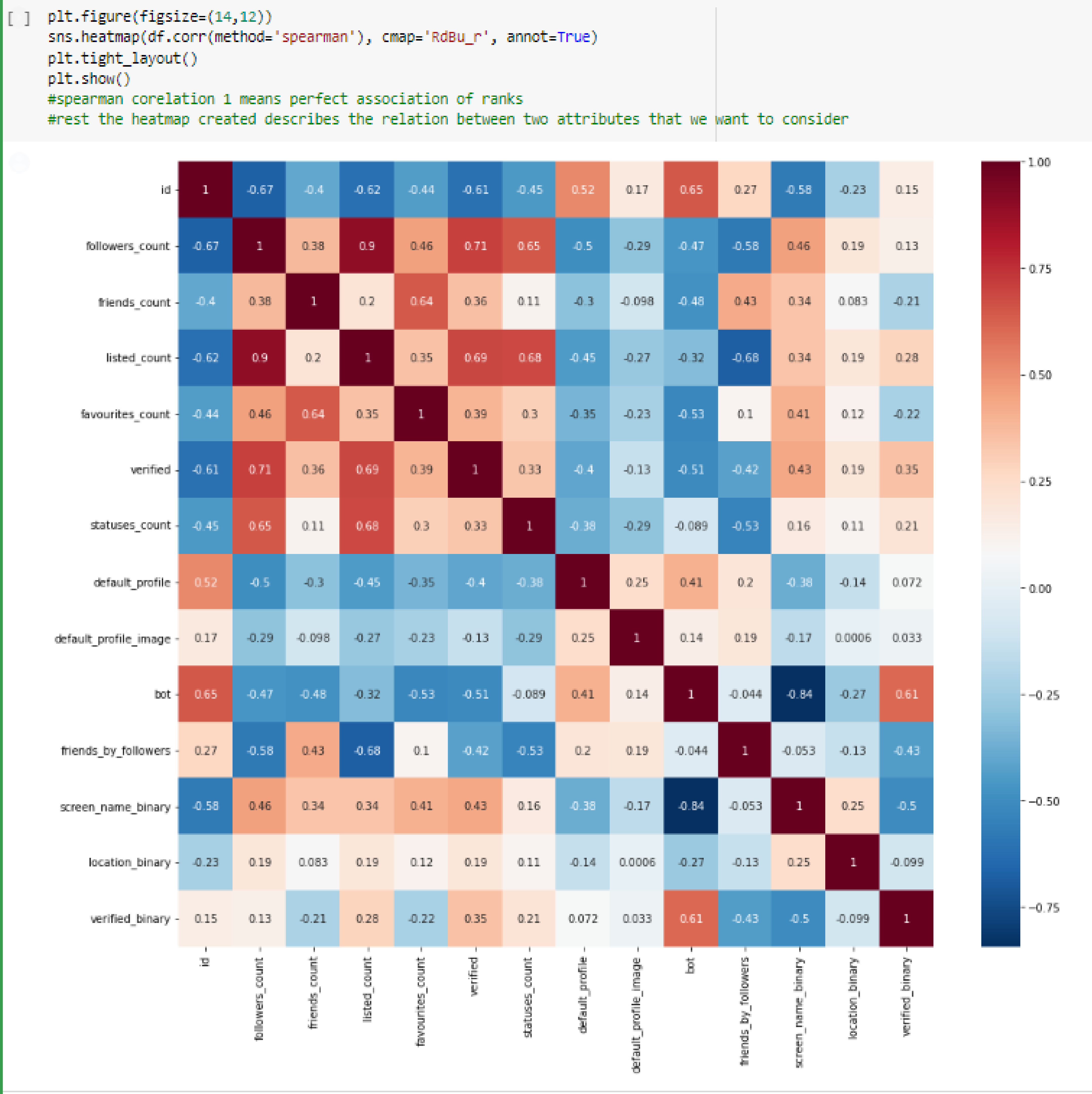
Feature Independence and Spearman Correlation

RESULT

There is no correlation between id, statuses_count, default_profile, default_profile_image and target variable.

There is strong correlation between verified, listed_count, friends_count, followers_count and target variable.

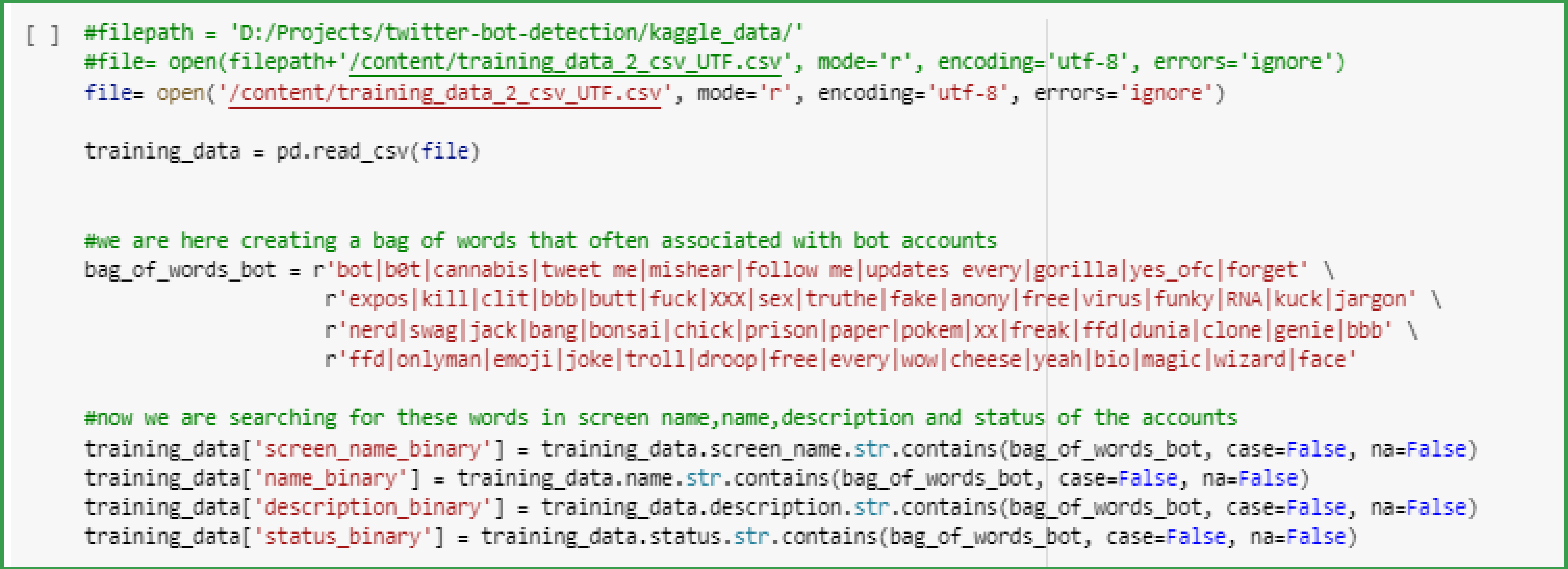
We cannot perform correlation for categorical attributes. So we will take screen_name, name, description, status into feature engineering. While use verified, listed_count for feature extraction.



```
[ ] plt.figure(figsize=(14,12))
    sns.heatmap(df.corr(method='spearman'), cmap='RdBu_r', annot=True)
    plt.tight_layout()
    plt.show()
    #spearman corelation 1 means perfect association of ranks
    #rest the heatmap created describes the relation between two attributes that we want to consider
```

	id	followers_count	friends_count	listed_count	favourites_count	verified	statuses_count	default_profile	default_profile_image	bot	friends_by_followers	screen_name_binary	location_binary	verified_binary
id	1	-0.67	-0.4	-0.62	-0.44	-0.61	-0.45	0.52	0.17	0.65	0.27	-0.58	-0.23	0.15
followers_count	-0.67	1	0.38	0.9	0.46	0.71	0.65	-0.5	-0.29	-0.47	-0.58	0.46	0.19	0.13
friends_count	-0.4	0.38	1	0.2	0.64	0.36	0.11	-0.3	-0.098	-0.48	0.43	0.34	0.083	-0.21
listed_count	-0.62	0.9	0.2	1	0.35	0.69	0.68	-0.45	-0.27	-0.32	-0.68	0.34	0.19	0.28
favourites_count	-0.44	0.46	0.64	0.35	1	0.39	0.3	-0.35	-0.23	-0.53	0.1	0.41	0.12	-0.22
verified	-0.61	0.71	0.36	0.69	0.39	1	0.33	-0.4	-0.13	-0.51	-0.42	0.43	0.19	0.35
statuses_count	-0.45	0.65	0.11	0.68	0.3	0.33	1	-0.38	-0.29	-0.089	-0.53	0.16	0.11	0.21
default_profile	0.52	-0.5	-0.3	-0.45	-0.35	-0.4	-0.38	1	0.25	0.41	0.2	-0.38	-0.14	0.072
default_profile_image	0.17	-0.29	-0.098	-0.27	-0.23	-0.13	-0.29	0.25	1	0.14	0.19	-0.17	0.0006	0.033
bot	0.65	-0.47	-0.48	-0.32	-0.53	-0.51	-0.089	0.41	0.14	1	-0.044	-0.84	-0.27	0.61
friends_by_followers	0.27	-0.58	0.43	-0.68	0.1	-0.42	-0.53	0.2	0.19	-0.044	1	-0.053	-0.13	-0.43
screen_name_binary	-0.58	0.46	0.34	0.34	0.41	0.43	0.16	-0.38	-0.17	-0.84	-0.053	1	0.25	-0.5
location_binary	-0.23	0.19	0.083	0.19	0.12	0.19	0.11	-0.14	0.0006	-0.27	-0.13	0.25	1	-0.099
verified_binary	0.15	0.13	-0.21	0.28	-0.22	0.35	0.21	0.072	0.033	0.61	-0.43	-0.5	-0.099	1

FEATURE ENGINEERING AND EXTRACTION

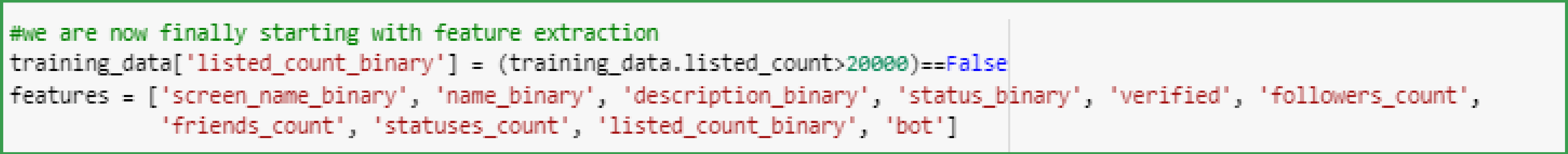


```
[ ]  #filepath = 'D:/Projects/twitter-bot-detection/kaggle_data/'
     #file= open(filepath+'/content/training_data_2_csv_UTF.csv', mode='r', encoding='utf-8', errors='ignore')
     file= open('/content/training_data_2_csv_UTF.csv', mode='r', encoding='utf-8', errors='ignore')

     training_data = pd.read_csv(file)


     #we are here creating a bag of words that often associated with bot accounts
     bag_of_words_bot = r'bot|b0t|cannabis|tweet me|mishear|follow me|updates every|gorilla|yes_ofc|forget' \
                        r'expos|kill|clit|bbb|butt|fuck|XXX|sex|truthe|fake|anony|free|virus|funky|RNA|kuck|jargon' \
                        r'nerd|swag|jack|bang|bonsai|chick|prison|paper|pokem|xx|freak|ffd|dunia|clone|genie|bbb' \
                        r'ffd|onlyman|emoji|joke|troll|droop|free|every|wow|cheese|yeah|bio|magic|wizard|face'

     #now we are searching for these words in screen name,name,description and status of the accounts
     training_data['screen_name_binary'] = training_data.screen_name.str.contains(bag_of_words_bot, case=False, na=False)
     training_data['name_binary'] = training_data.name.str.contains(bag_of_words_bot, case=False, na=False)
     training_data['description_binary'] = training_data.description.str.contains(bag_of_words_bot, case=False, na=False)
     training_data['status_binary'] = training_data.status.str.contains(bag_of_words_bot, case=False, na=False)
```



```
#we are now finally starting with feature extraction
training_data['listed_count_binary'] = (training_data.listed_count>20000)==False
features = ['screen_name_binary', 'name_binary', 'description_binary', 'status_binary', 'verified', 'followers_count',
            'friends_count', 'statuses_count', 'listed_count_binary', 'bot']
```

Multinomial Naive Bayes Classifier

Naive Bayes classifiers are a collection of classification algorithms based on Bayes' Theorem.

It is not a single algorithm but a family of algorithms where all of them share a common principle, i.e. every pair of features being classified is independent of each other.

The algorithm is a supervised learning algorithm. It is mainly used in text classification that includes a high-dimensional training dataset.

The fundamental Naive Bayes assumption is that each feature makes an: independent and equal contribution to the outcome.

It is a probabilistic classifier, which means it predicts on the basis of the probability of an object.

Naïve Bayes is one of the fast and easy ML algorithms to predict a class of datasets.

It can be used for Binary as well as Multi-class Classifications.

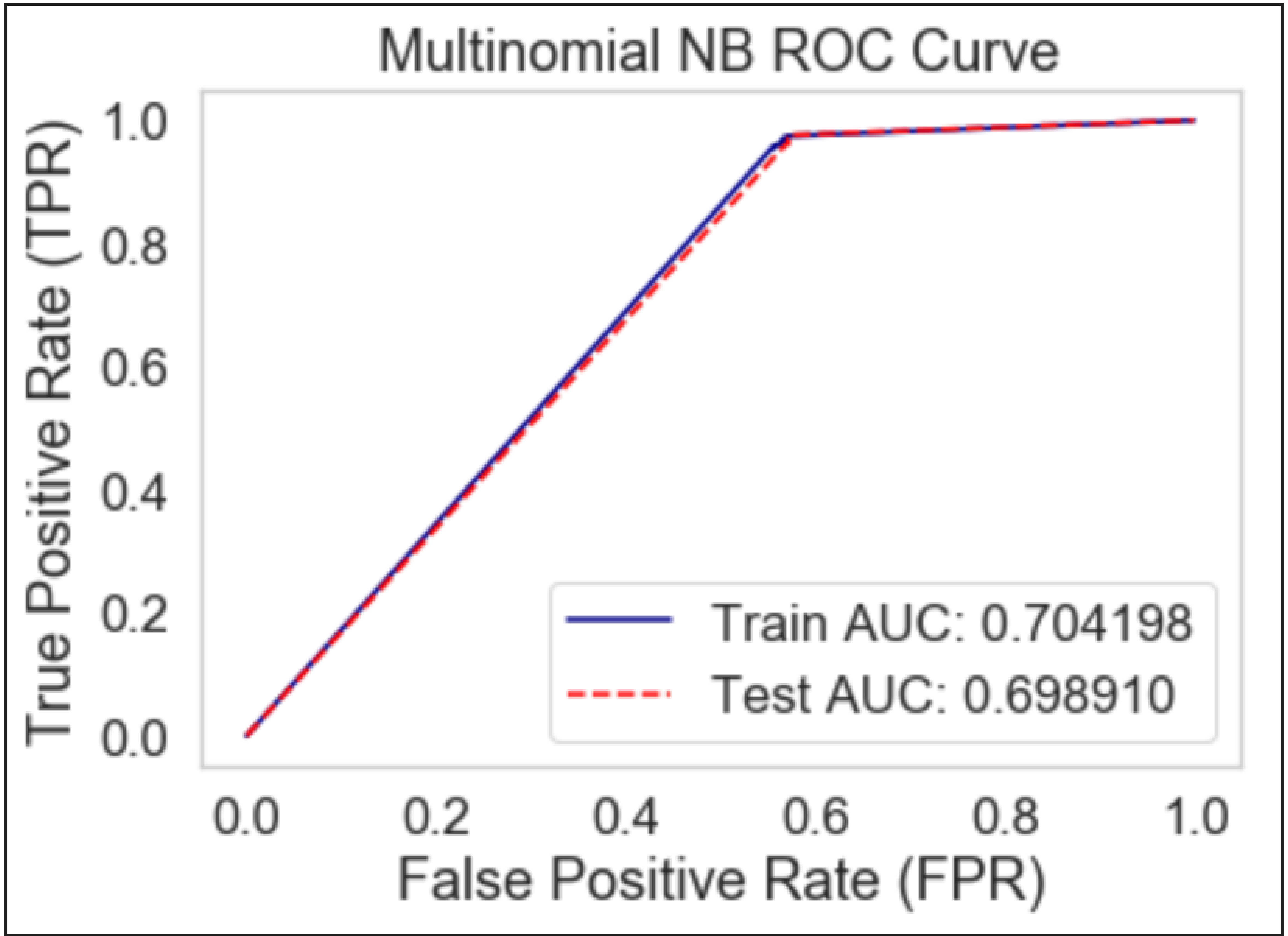
It performs well in Multi-class predictions as compared to the other Algorithms.

It is the most popular choice for text classification problems.

Naive Bayes assumes that all features are independent or unrelated, so it cannot learn the relationship between features.

This algorithm faces the 'zero-frequency problem' where it assigns zero probability to a categorical variable whose category in the test data set wasn't available in the training dataset.

Training Accuracy: 0.67961 Test Accuracy: 0.69762



Decision Tree Classifier

Decision Tree is a Supervised learning technique that can be used for both classification and Regression problems.

It is a tree-structured classifier where internal nodes represent the features of a dataset, branches represent the decision rules and each leaf node represents the outcome.

A decision tree can contain categorical data (YES/NO) as well as numeric data.

In a decision tree, for predicting the class of the given dataset, the algorithm starts from the root node of the tree.

This algorithm compares the values of root attribute with the record (real dataset) attribute and, based on the comparison, follows the branch and jumps to the next node.

It is simple to understand as it follows the same process which a human follow while making any decision in real-life.

It can be very useful for solving decision-related problems.
It helps to think about all the possible outcomes for a problem.

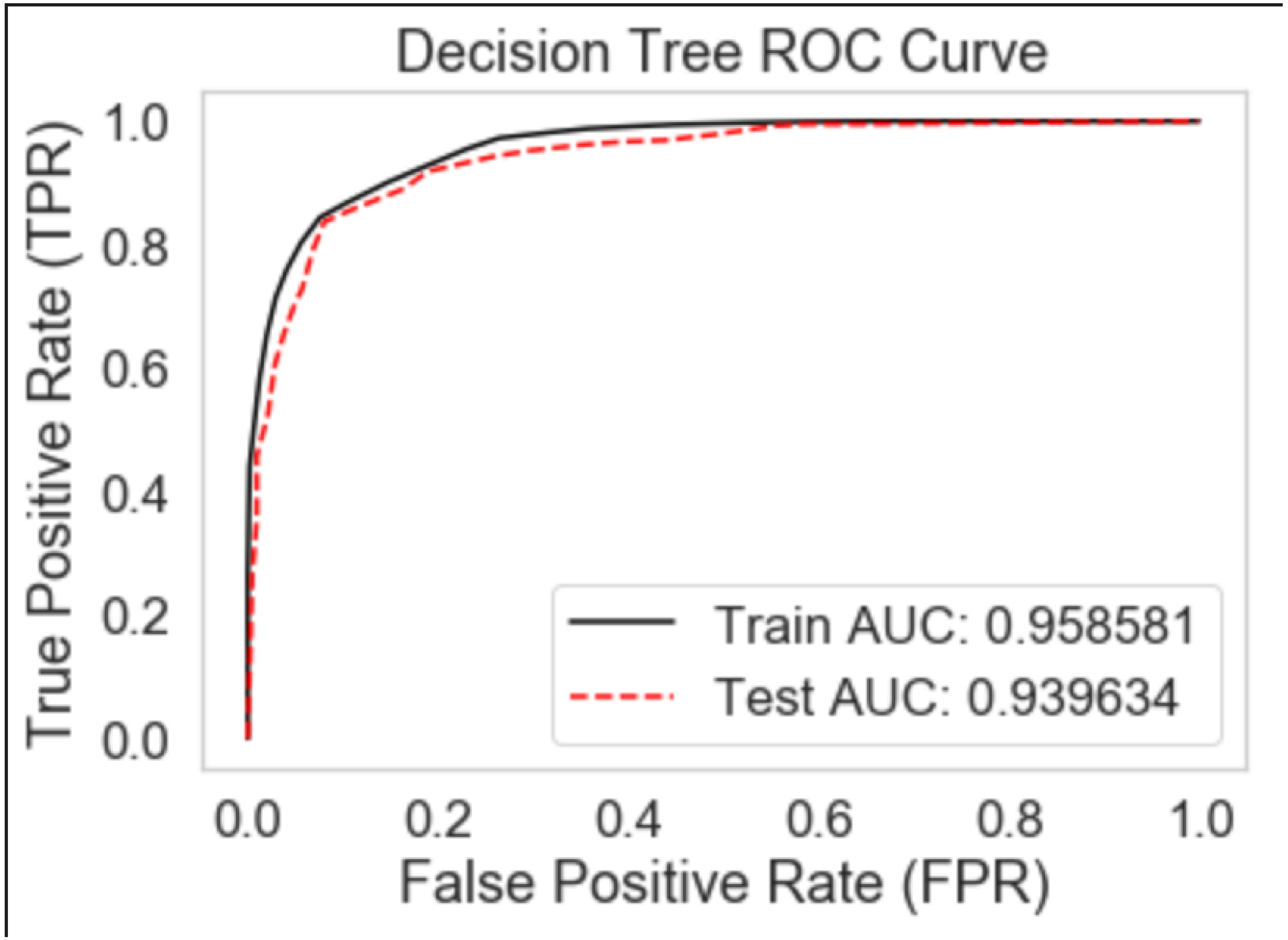
There is less requirement of data cleaning compared to other algorithms.

The decision tree contains lots of layers, which makes it complex.

It may have an overfitting issue, which can be resolved using the Random Forest algorithm.

For more class labels, the computational complexity of the decision tree may increase.

Test Accuracy: 0.87857 and Training Accuracy: 0.88707



Random Forest Classifier

Random Forest is a popular machine learning algorithm that belongs to the supervised learning technique.

It is based on the concept of ensemble learning, which is a process of combining multiple classifiers to solve a complex problem and to improve the performance of the model.

The greater number of trees in the forest leads to higher accuracy and prevents the problem of overfitting.

Select random K data points from the training set.

Build the decision trees associated with the selected data points (Subsets).

Choose the number N for decision trees that you want to build. Repeat Step 1 & 2

For new data points, find the predictions of each decision tree, and assign the new data points to the category that wins the majority votes.

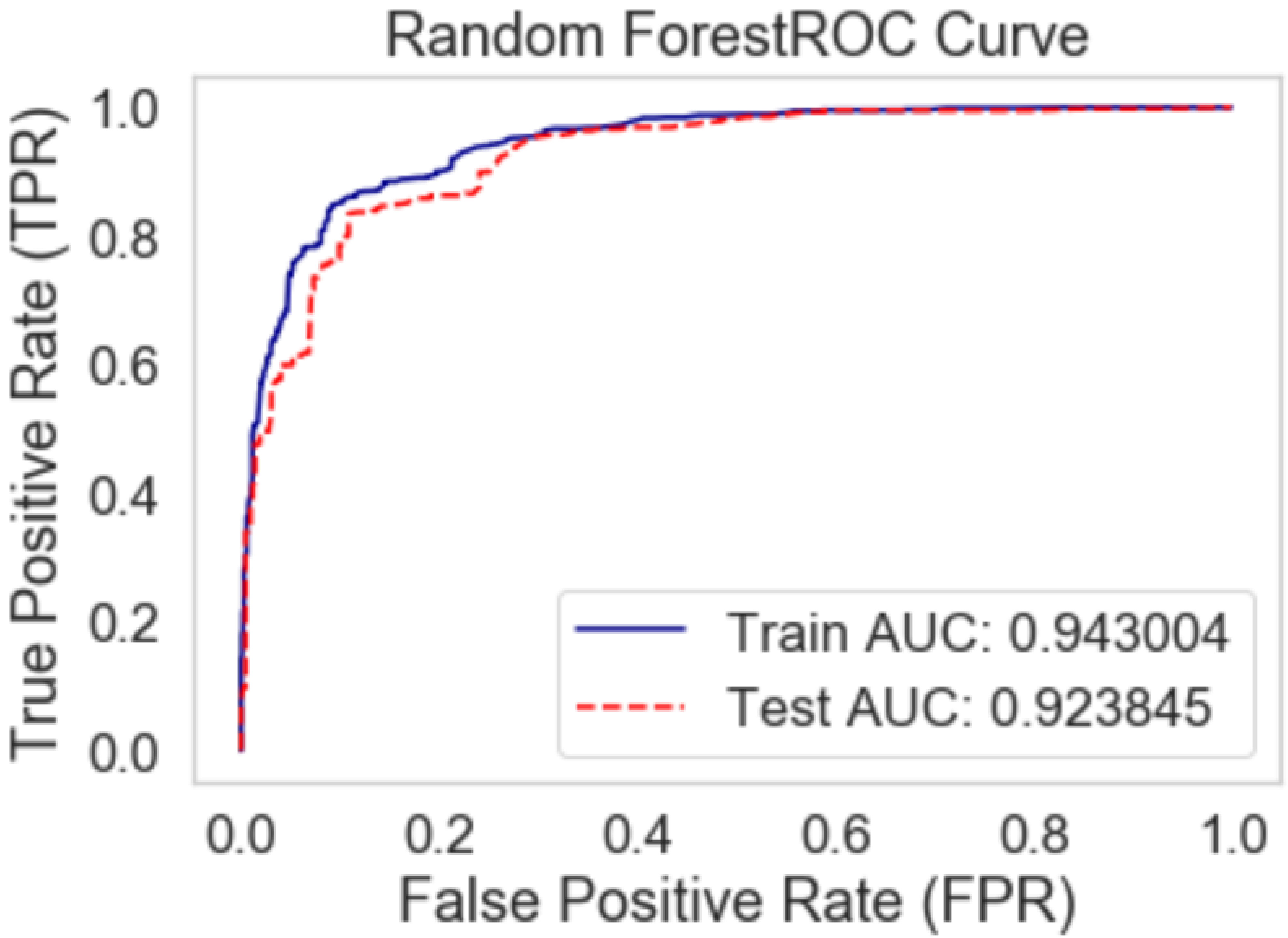
Random Forest is capable of performing both Classification and Regression tasks.

It is capable of handling large datasets with high dimensionality.

It enhances the accuracy of the model and prevents the overfitting issue.

Although random forest can be used for both classification and regression tasks, it is not more suitable for Regression tasks.

Train accuracy of 0.87839 and Test accuracy of 0.85238.



CONCLUSION

The Decision Tree Algorithm was found to be the best learning model with a training accuracy of 0.88707 and test accuracy of 0.87857