



***Progressive Education Society's***  
***MODERN COLLEGE OF ARTS, SCIENCE AND COMMERCE***  
***GANESHKHIND, PUNE – 411016***

**A PROJECT ON:**  
***AI-Based Fake Account Identifier***

**SUBMITTED TO:**  
***Savitribai Phule Pune University***

**BY:**  
***Sanskriti Joshi (Roll no.: 233331028, Seat no.: 2018)***  
***Priyanka Rout (Roll no.: 233331064, Seat no.: 2014)***

**T.Y. BSc (Computer Science) [2023-24]**



***Progressive Education Society's***  
***MODERN COLLEGE OF ARTS, SCIENCE AND COMMERCE***  
***GANESHKHIND, PUNE – 411016***

## **CERTIFICATE**

This is to certify that **Sanskruti Joshi** and **Priyanka Rout** of **T.Y. BSC (Computer Science)** completed the project work titled “***AI-Based Fake Account Identifier***” for the curriculum of Savitribai Phule Pune University during the academic year **2023-2024**.

**Project Guide**

Asst. Prof. Swapnali Waghmare

**Head of Department**

Dr. Shubhangi Bhatambrekar

**Internal Examiner**

**External Examiner**

# **INDEX**

| <b>S.No</b> | <b>Title</b>                | <b>Page No</b> |
|-------------|-----------------------------|----------------|
| 1.          | Abstract                    | 4              |
| 2.          | Literature Survey           | 5              |
| 3.          | Introduction                | 6              |
| 4.          | System Model (UML Diagrams) | 8              |
| 5.          | Data Discovery              | 12             |
| 6.          | Data Understanding          | 14             |
| 7.          | Data Preparation            | 17             |
| 8.          | Exploratory Data Analysis   | 21             |
| 9.          | Data Modeling               | 32             |
| 10.         | Model Evaluation            | 41             |
| 11.         | Model Comparison            | 52             |
| 12.         | Website Specifications      | 58             |
| 13.         | Conclusion                  | 62             |
| 14.         | Scope for Improvement       | 64             |
| 15.         | Bibliography                | 65             |

# **I- ABSTRACT**

## **AI-Based Fake Account Identifier**

Fake Account Identifier aims to build and train a network model to detect fake or spam accounts in social media sites like Facebook, Twitter(X), Instagram etc. This will enhance the security and integrity of these platforms by preventing the proliferation of fraudulent activities often associated with fake accounts.

## **Existing System**

The existing systems use very few factors to decide whether an account is fake or not. The factors largely affect the way decision making occurs. When the number of factors is low, the accuracy of the decision making is reduced significantly.

## **Proposed System**

Unlike the existing system, Fake Account Identifier will employ multiple machine learning algorithms to classify accounts as spam and genuine. As we are using Artificial Intelligence, the predictions will be unbiased and the data is protected. The entire lifecycle of the project will be displayed via a website.

## **Software Requirements**

1. *Operating System* : CentOS7 (Linux)
2. *IDE* : Jupyter Notebook / Google Collab
3. *Web browser* : Firefox / Google Chrome

## **Hardware Requirements**

1. *Hard disk* : 564 GB & more.
2. *Processor* : 11th Gen Intel(R) Core(TM) i3-1115G4 @ 3.00GHz
3. *RAM* : 8.00 GB & more.

## **Technologies Used**

1. *Front End* : HTML, CSS, Bootstrap, Javascript
2. *Back End* : Python, PHP, MySql

## **II- LITERATURE SURVEY**

Nowadays the usage of digital technology has been increasing exponentially. At the same time the rate of malicious users has been increasing. Online social sites like Facebook and Twitter attract millions of people globally. This interest in online networking has opened to various issues including the risk of exposing false data by creating fake accounts resulting in the spread of malicious content. Fake accounts are a popular way to forward spam, commit fraud, and abuse through online social networks. These problems need to be tackled in order to give the user a reliable online social network. In this paper, we are using different ML algorithms like Support Vector Machine (SVM), Decision Tree (DT), Random Forest (RF), K-Nearest Neighbours (KNN) and Extreme Gradient Boosting (XGBoost). Along with these algorithms we have used one normalization technique that is Standard Scaler , to improve accuracy. We have implemented it to detect fake accounts and bots. Our approach achieved high accuracy and true positive rate for Random Forest and XGBoost.

**Keywords:** Data mining, Classification, Decision Tree, Support Vector Machine, K-Nearest Neighbours, Random forest, XGBoost, Standard Scaling

# **III- INTRODUCTION**

## **Problem Statement**

Social Networking plays an integral part of information in the modern world. It structures users' views on religious, political, social-political, interests, hobbies etc.

Therefore, cybersecurity of personal information in such social networking sites , social media application plays a crucial role and requires attention.

One of the main problems in such networking is social bots or harmful accounts. They are a tool for reducing trust in social networks, mass theft of personal data, boosting non-relevant content etc.

At the moment, fake accounts create a lot of nuisance both for ordinary users as well as for those who use social networks to conduct a marketing campaign or for some research. They use incorrect information or statistics about some real-world person to create a fake account.

## **Problem Understanding**

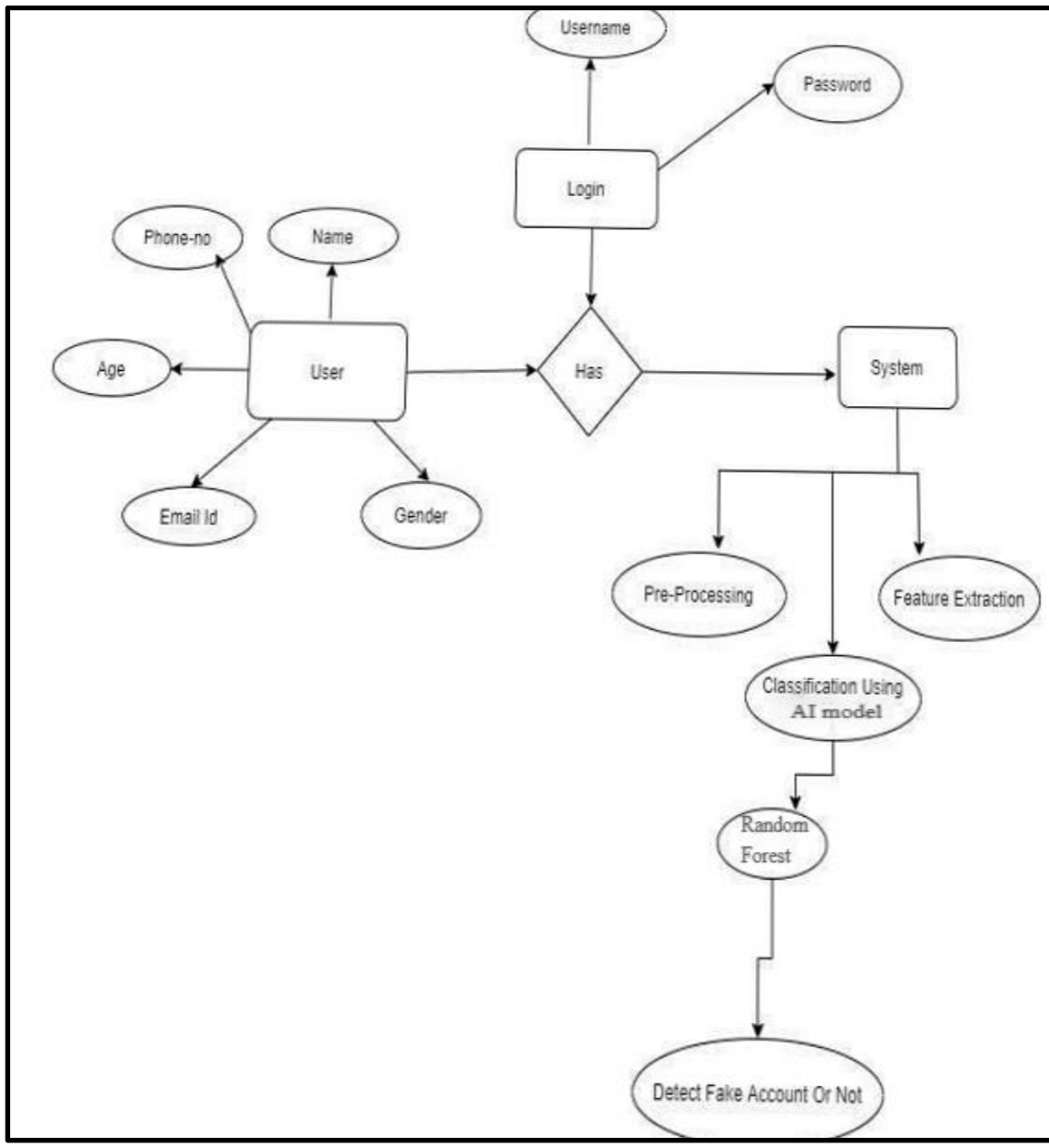
- ❖ This Project's aim is to build and train a network model to detect fake or spam accounts in social media sites like Facebook, Twitter(X), Instagram etc.
- ❖ To restrict fake accounts that are created to sell fake services and products which leads to misleading for ordinary users.



- ❖ It will detect spam accounts made to boost any random services illegally which degrades the real products value in the networking market.
- ❖ To protect an individual or group's data as they also use it to impersonate others to harm or criticize their reputation.

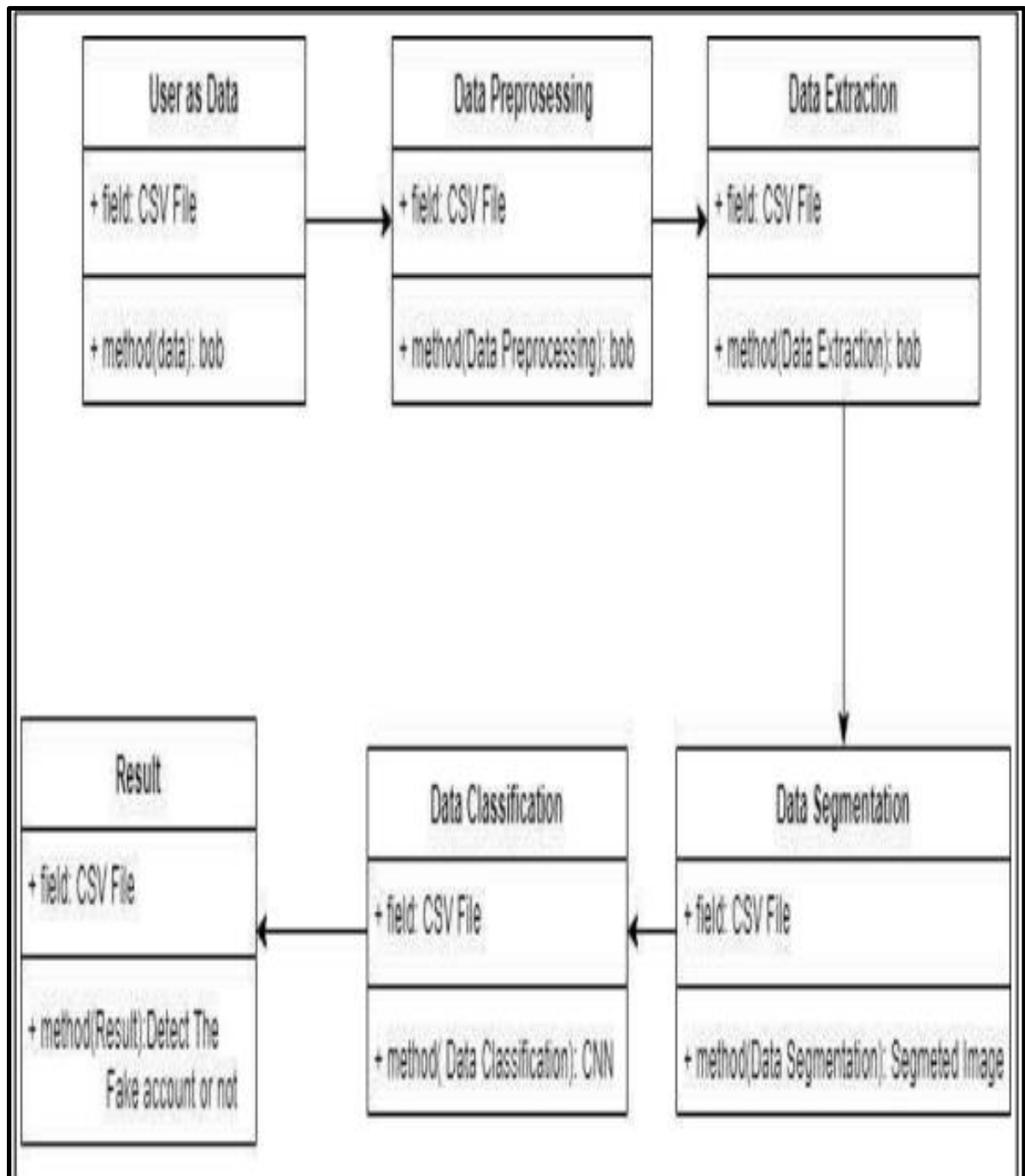
## IV- System Model (UML Diagrams)

### *1. ERD (Entity Relationship Diagram):*

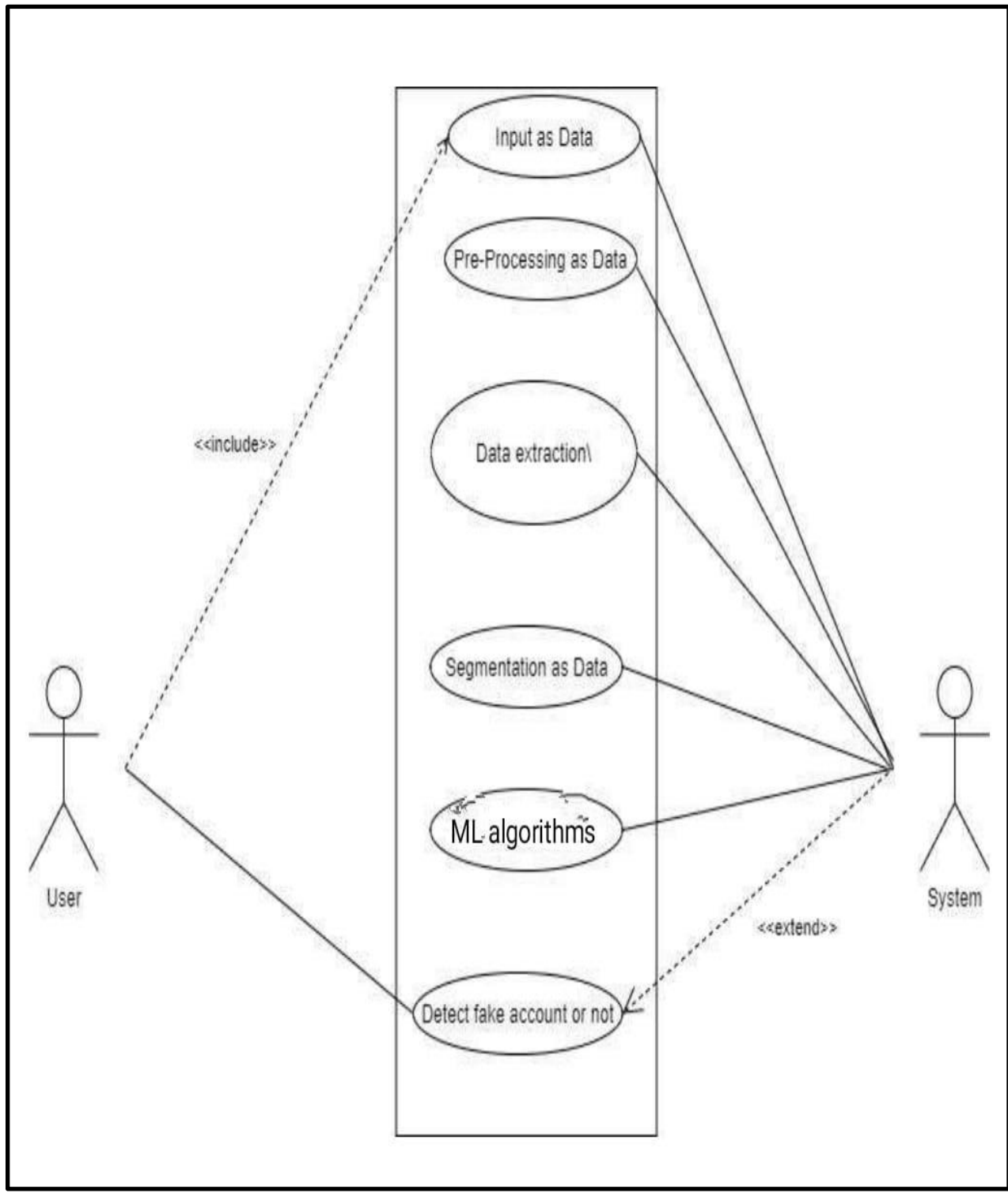




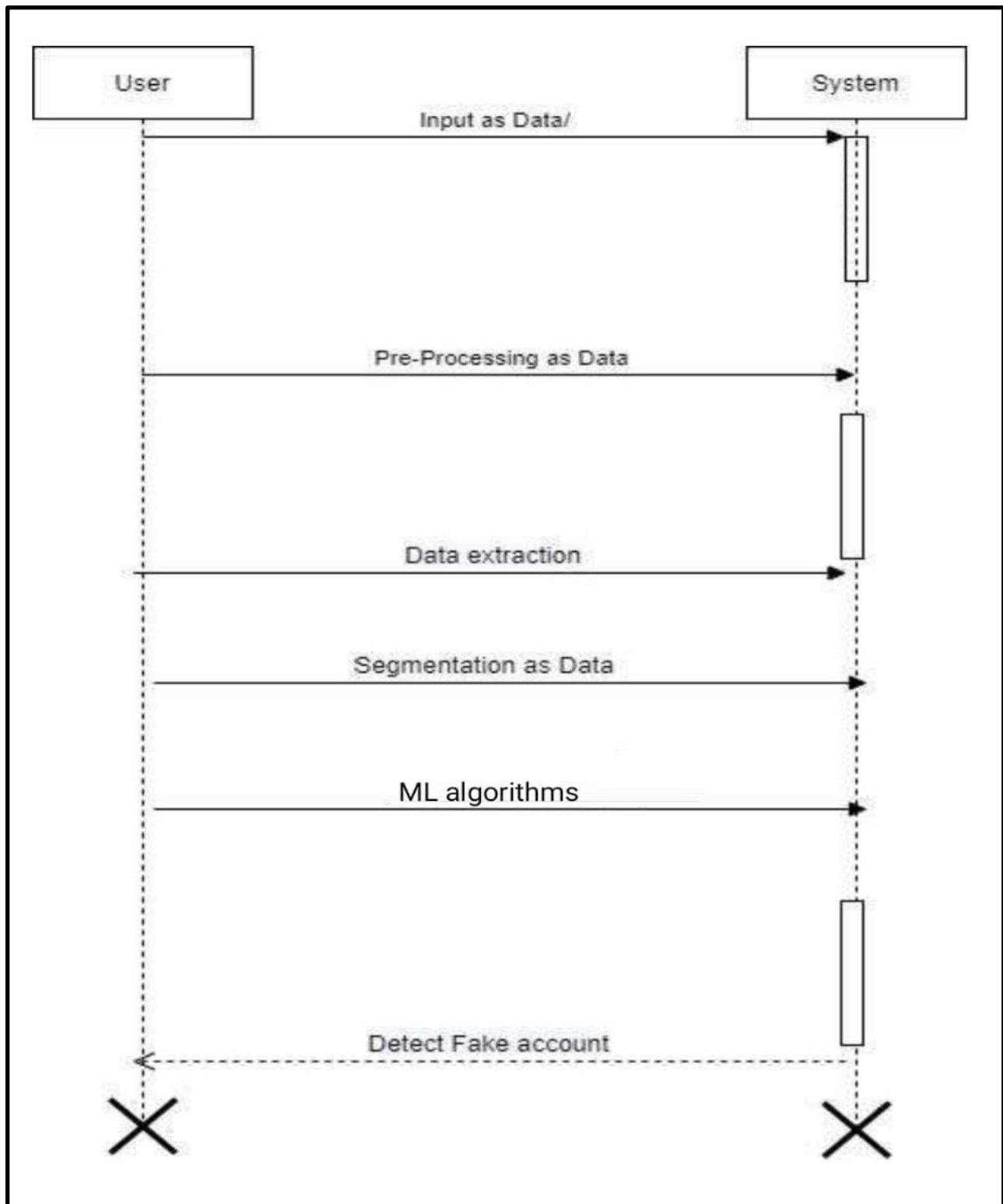
## 2. Class Diagram :



### 3. Use Case Diagram :



#### 4. Sequence Diagram :



## **V- DATA DISCOVERY**

The dataset is available on *Kaggle* website. There are two datasets i.e ‘*train.csv*’ which will be used to train the machine learning model over a subset of the original dataset and ‘*test.csv*’ to test the quality & evaluate whether it can generalize well to the new or unseen data of our model.

- **Training Dataset :**

The training data lets the ML algorithms learn how to make predictions for the given task. It is highly responsible for the model's accuracy and prediction ability. It means that the better the quality of the training data, the better will be the performance of the model. Training data is approximately more than or equal to 60% of the total data for an ML project.

- **Testing Dataset :**

Once we train the model with the training dataset, it's time to test the model with the test dataset. This dataset evaluates the performance of the model and ensures that the model can generalize well with the new or unseen dataset. Test data is a well-organized dataset that contains data for each type of scenario for a given problem that the model would be facing when used in the real world. Usually, the test dataset is approximately 20-25% of the total original data for an ML project.

## Importing Training Dataset

```
train = pd.read_csv(r'train.csv')
train
```

|     | profile pic | nums/length | username | fullname | words | nums/length | fullname | name==username | description | length | external URL | private | #posts | #followers | #follows | fake |
|-----|-------------|-------------|----------|----------|-------|-------------|----------|----------------|-------------|--------|--------------|---------|--------|------------|----------|------|
| 0   | 1           |             | 0.27     |          | 0     |             | 0.00     | 0              |             | 53     | 0            | 0       | 32     | 1000       | 955      | 0    |
| 1   | 1           |             | 0.00     |          | 2     |             | 0.00     | 0              |             | 44     | 0            | 0       | 286    | 2740       | 533      | 0    |
| 2   | 1           |             | 0.10     |          | 2     |             | 0.00     | 0              |             | 0      | 0            | 1       | 13     | 159        | 98       | 0    |
| 3   | 1           |             | 0.00     |          | 1     |             | 0.00     | 0              |             | 82     | 0            | 0       | 679    | 414        | 651      | 0    |
| 4   | 1           |             | 0.00     |          | 2     |             | 0.00     | 0              |             | 0      | 0            | 1       | 6      | 151        | 126      | 0    |
| ... | ...         |             | ...      |          | ...   |             | ...      | ...            |             | ...    | ...          | ...     | ...    | ...        | ...      | ...  |
| 571 | 1           |             | 0.55     |          | 1     |             | 0.44     | 0              |             | 0      | 0            | 0       | 33     | 166        | 596      | 1    |
| 572 | 1           |             | 0.38     |          | 1     |             | 0.33     | 0              |             | 21     | 0            | 0       | 44     | 66         | 75       | 1    |
| 573 | 1           |             | 0.57     |          | 2     |             | 0.00     | 0              |             | 0      | 0            | 0       | 4      | 96         | 339      | 1    |
| 574 | 1           |             | 0.57     |          | 1     |             | 0.00     | 0              |             | 11     | 0            | 0       | 0      | 57         | 73       | 1    |
| 575 | 1           |             | 0.27     |          | 1     |             | 0.00     | 0              |             | 0      | 0            | 0       | 2      | 150        | 487      | 1    |

576 rows × 12 columns

## Importing Testing Dataset

```
test = pd.read_csv(r'test.csv')
test
```

|     | profile pic | nums/length | username | fullname | words | nums/length | fullname | name==username | description | length | external URL | private | #posts | #followers | #follows | fake |
|-----|-------------|-------------|----------|----------|-------|-------------|----------|----------------|-------------|--------|--------------|---------|--------|------------|----------|------|
| 0   | 1           |             | 0.33     |          | 1     |             | 0.33     | 1              |             | 30     | 0            | 1       | 35     | 488        | 604      | 0    |
| 1   | 1           |             | 0.00     |          | 5     |             | 0.00     | 0              |             | 64     | 0            | 1       | 3      | 35         | 6        | 0    |
| 2   | 1           |             | 0.00     |          | 2     |             | 0.00     | 0              |             | 82     | 0            | 1       | 319    | 328        | 668      | 0    |
| 3   | 1           |             | 0.00     |          | 1     |             | 0.00     | 0              |             | 143    | 0            | 1       | 273    | 14890      | 7369     | 0    |
| 4   | 1           |             | 0.50     |          | 1     |             | 0.00     | 0              |             | 76     | 0            | 1       | 6      | 225        | 356      | 0    |
| ... | ...         |             | ...      |          | ...   |             | ...      | ...            |             | ...    | ...          | ...     | ...    | ...        | ...      | ...  |
| 115 | 1           |             | 0.29     |          | 1     |             | 0.00     | 0              |             | 0      | 0            | 0       | 13     | 114        | 811      | 1    |
| 116 | 1           |             | 0.40     |          | 1     |             | 0.00     | 0              |             | 0      | 0            | 0       | 4      | 150        | 164      | 1    |
| 117 | 1           |             | 0.00     |          | 2     |             | 0.00     | 0              |             | 0      | 0            | 0       | 3      | 833        | 3572     | 1    |
| 118 | 0           |             | 0.17     |          | 1     |             | 0.00     | 0              |             | 0      | 0            | 0       | 1      | 219        | 1695     | 1    |
| 119 | 1           |             | 0.44     |          | 1     |             | 0.00     | 0              |             | 0      | 0            | 0       | 3      | 39         | 68       | 1    |

120 rows × 12 columns

## VI- DATA UNDERSTANDING

```
tr_smp.info()
```

```
<class 'pandas.core.frame.DataFrame'>
```

```
Int64Index: 10 entries, 137 to 2
```

```
Data columns (total 12 columns):
```

| #  | Column               | Non-Null Count | Dtype   |
|----|----------------------|----------------|---------|
| 0  | profile pic          | 10 non-null    | int64   |
| 1  | nums/length username | 10 non-null    | float64 |
| 2  | fullname words       | 10 non-null    | int64   |
| 3  | nums/length fullname | 10 non-null    | float64 |
| 4  | name==username       | 10 non-null    | int64   |
| 5  | description length   | 10 non-null    | int64   |
| 6  | external URL         | 10 non-null    | int64   |
| 7  | private              | 10 non-null    | int64   |
| 8  | #posts               | 10 non-null    | int64   |
| 9  | #followers           | 10 non-null    | int64   |
| 10 | #follows             | 10 non-null    | int64   |
| 11 | fake                 | 10 non-null    | int64   |

```
dtypes: float64(2), int64(10)
```

```
memory usage: 1.0 KB
```

```
train.value_counts().sum()
```

```
576
```

```
test.value_counts().sum()
```

```
120
```

Both datasets contain the following attributes:

**1. profile pic:**

This column shows whether the account has a profile picture or not (0 for “no profile pic”, 1 for “profile pic”).

**2. nums/length username:**

This column shows the ratio of the number of numeric characters in the username to its length.

**3. fullname words:**

This depicts the number of words in the full name of the account.

**4. nums/length fullname:**

This column shows the ratio of the number of numeric characters in the fullname to its length.

**5. name==username:**

This shows whether the username is the same as the full name registered by an account (0 for “not same” , 1 for “same”).

**6. description length:**

This column shows the number of characters written in the bio section of an account.

**7. external URL:**

It shows whether the account has any external URL in bio (0 for “no external URL”, 1 for “has external URL”).

**8. private:**

Shows whether the account is private or not (0 for false, 1 for true).

**9. #posts:**

It depicts the number of publications like images, videos etc posted by the account.

**10. #followers:**

It shows the number of profiles following the account.

**11. #follows:**

It shows the number of profiles the account follows.

**12. fake:**

This is our target variable which shows whether the account is fake or not (0 for “not fake”, 1 for “fake”).



## VII- DATA PREPARATION

Data Preparation involves preprocessing of the datasets.

Data Preprocessing is a crucial step in preparing the dataset for analysis, especially when extracting dependent and independent variables.

We include following preprocessing steps for the datasets:

### ❖ Handling Null Values:

```
# Checking for null values
train.isnull().sum()

profile pic          0
nums/length username 0
fullname words       0
nums/length fullname 0
name==username       0
description length   0
external URL         0
private              0
#posts              0
#followers           0
#follows             0
fake                 0
dtype: int64
```

Identifies and addresses null values in the dataset, employing techniques such as imputation or removal based on the context.

### ❖ Duplicacy Check :

```
# checkng and removing duplicate records
train=train.drop_duplicates()

train.duplicated().sum()

0
```

Ensured the dataset contains unique accounts, eliminating duplicate entries that could skew model training.

## ❖ Correlation Check :

```
# correlation between target variable, i.e, "fake" and other variables for train dataset
print("For train dataset\n")
train.corr()[["fake"]]

For train dataset
```

|                      | fake      |
|----------------------|-----------|
| profile pic          | -0.638899 |
| nums/length username | 0.587863  |
| fullname words       | -0.297777 |
| nums/length fullname | 0.247248  |
| name==username       | 0.171003  |
| description length   | -0.459736 |
| external URL         | -0.363524 |
| private              | -0.028667 |
| #posts               | -0.244854 |
| #followers           | -0.093843 |
| #follows             | -0.224098 |
| fake                 | 1.000000  |

Correlation process measures the strength and direction of a relationship between a target variable i.e “fake” with other variables for the datasets.

## ❖ Feature selection and extraction :

```
#Extracting independent & dependent variable from train.csv
feature_tr=train.drop(["fake"] , axis=1)
target_tr=train["fake"]
```

```
print("Target variable:\n",target_tr)

Target variable:
0      0
1      0
2      0
3      0
4      0
..
571    1
572    1
573    1
574    1
575    1
Name: fake, Length: 574, dtype: int64
```

Identifying variables is crucial for further research. The next step is to separate our target variable and features into different data sets, both for training and testing, in order to start training our Machine Learning model.

```
print("\nDependent variables:\n",feature_tr)
```

Dependent variables:

|     | profile pic | nums/length | username | fullname | words | nums/length | fullname |
|-----|-------------|-------------|----------|----------|-------|-------------|----------|
| 0   | 1           |             | 0.27     |          | 0     |             | 0.00     |
| 1   | 1           |             | 0.00     |          | 2     |             | 0.00     |
| 2   | 1           |             | 0.10     |          | 2     |             | 0.00     |
| 3   | 1           |             | 0.00     |          | 1     |             | 0.00     |
| 4   | 1           |             | 0.00     |          | 2     |             | 0.00     |
| ..  | ...         |             | ...      |          | ...   |             | ...      |
| 571 | 1           |             | 0.55     |          | 1     |             | 0.44     |
| 572 | 1           |             | 0.38     |          | 1     |             | 0.33     |
| 573 | 1           |             | 0.57     |          | 2     |             | 0.00     |
| 574 | 1           |             | 0.57     |          | 1     |             | 0.00     |
| 575 | 1           |             | 0.27     |          | 1     |             | 0.00     |

|     | name==username | description length | external URL | private | #posts | \ |
|-----|----------------|--------------------|--------------|---------|--------|---|
| 0   | 0              | 53                 | 0            | 0       | 32     |   |
| 1   | 0              | 44                 | 0            | 0       | 286    |   |
| 2   | 0              | 0                  | 0            | 1       | 13     |   |
| 3   | 0              | 82                 | 0            | 0       | 679    |   |
| 4   | 0              | 0                  | 0            | 1       | 6      |   |
| ..  | ...            | ...                | ...          | ...     | ...    |   |
| 571 | 0              | 0                  | 0            | 0       | 33     |   |
| 572 | 0              | 21                 | 0            | 0       | 44     |   |
| 573 | 0              | 0                  | 0            | 0       | 4      |   |
| 574 | 0              | 11                 | 0            | 0       | 0      |   |
| 575 | 0              | 0                  | 0            | 0       | 2      |   |

|     | #followers | #follows |
|-----|------------|----------|
| 0   | 1000       | 955      |
| 1   | 2740       | 533      |
| 2   | 159        | 98       |
| 3   | 414        | 651      |
| 4   | 151        | 126      |
| ..  | ...        | ...      |
| 571 | 166        | 596      |
| 572 | 66         | 75       |
| 573 | 96         | 339      |
| 574 | 57         | 73       |
| 575 | 150        | 487      |

[574 rows x 11 columns]

## ❖ Numerical Feature Scaling:

```
# Scaling the datasets
tr=StandardScaler().fit_transform(train)
print(tr)

[[ 0.65411061  0.50580845 -1.38631564 ... -0.09286701  0.48558343
   -1.          ]
 [ 0.65411061 -0.76841447  0.51283753 ... -0.09095687  0.02615876
   -1.          ]
 [ 0.65411061 -0.29648005  0.51283753 ... -0.09379024 -0.4474188
   -1.          ]
 ...
 [ 0.65411061  1.9216117   0.51283753 ... -0.0938594  -0.18504594
    1.          ]
 [ 0.65411061  1.9216117  -0.43673906 ... -0.09390222 -0.4746359
    1.          ]
 [ 0.65411061  0.50580845 -0.43673906 ... -0.09380012 -0.0239207
    1.          ]]
```

Standardized numerical features to a consistent scale using Standard Scaler, which helps to get standardized distribution, with a zero mean and standard deviation of one (unit variance). It standardizes features by subtracting the mean value from the feature and then dividing the result by feature standard deviation.

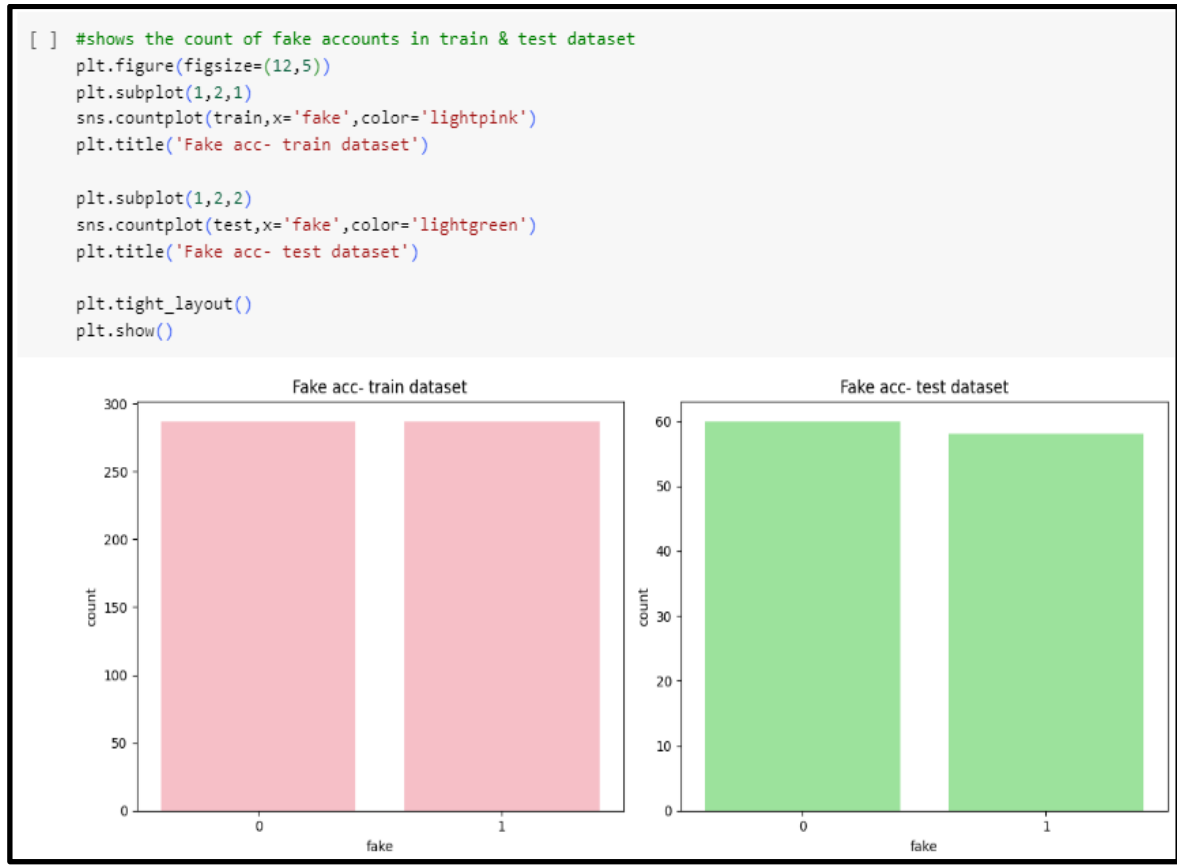
# **VIII- EXPLORATORY DATA ANALYSIS**

We have divided EDA into following parts :

- **Univariate Analysis:** Examining a single variable to understand its distribution and characteristics.
- **Bivariate Analysis:** Exploring the relationship between two variables. This includes studying how changes in one variable relate to changes in another.
- **Trivariate Analysis:** Considering the relationships and interactions between three variables at a time. Examines how three variables are dependent on each other.
- **Multivariate Analysis:** This includes examining relationships between more than three variables simultaneously. It includes complex statistical tools as it contains more variables.

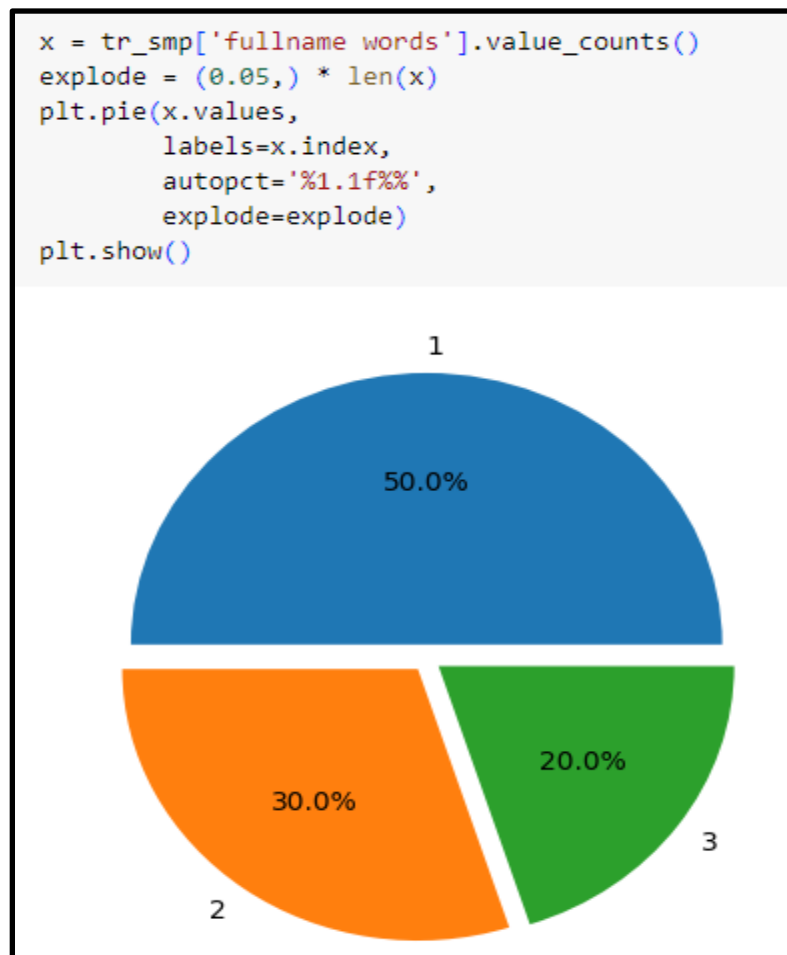
# Univariate Analysis

## ➤ Bar Graph:



The two graphs represent the count of fake accounts, the left side (light pink) graph indicates fake account count for train dataset. Hence, in the training dataset, the number of fake accounts and genuine accounts are the same. On the right side (light green) graph shows fake account count for test dataset. Here, the number of real accounts is slightly lesser than that of fake accounts.

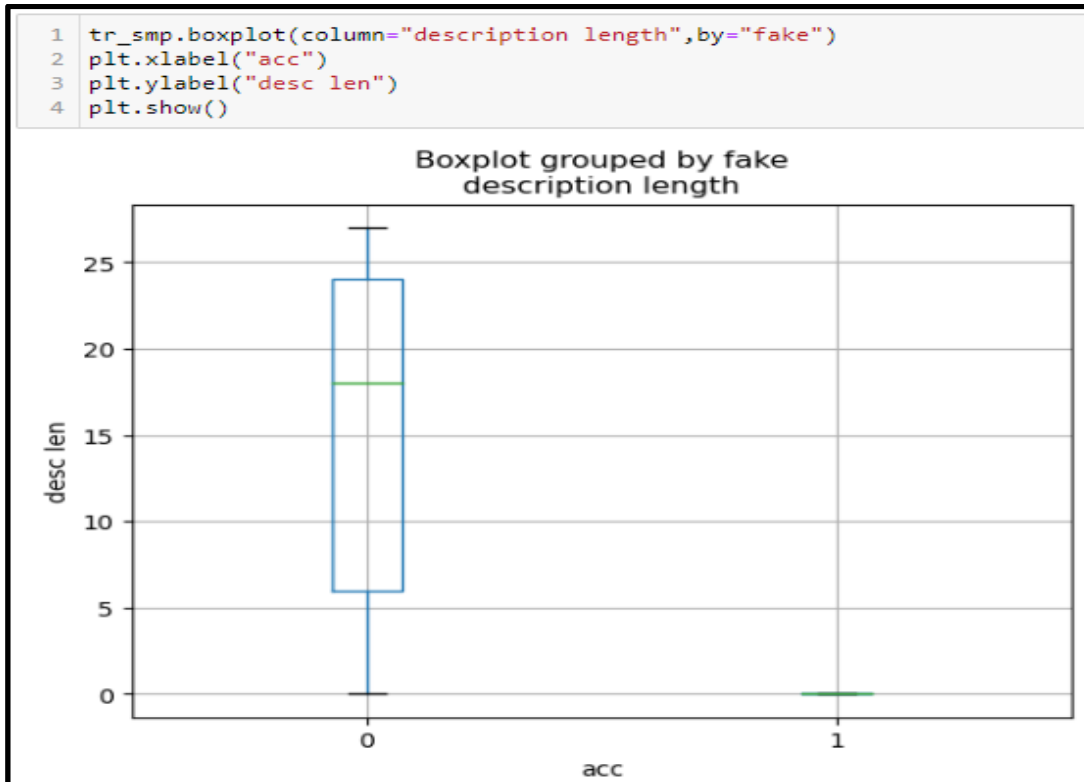
## ➤ Pie Chart:



Above diagram shows the distribution of word lengths in the fullname words column of the train dataframe. The pie chart has three slices, corresponding to the possible word lengths: 1, 2, and 3. The percentage annotation shows the proportion of each word length in the column. For example, the slice for word length 1 has 50%, which means that half of the total accounts in the sample have only one word in their full name.

## Bivariate Analysis

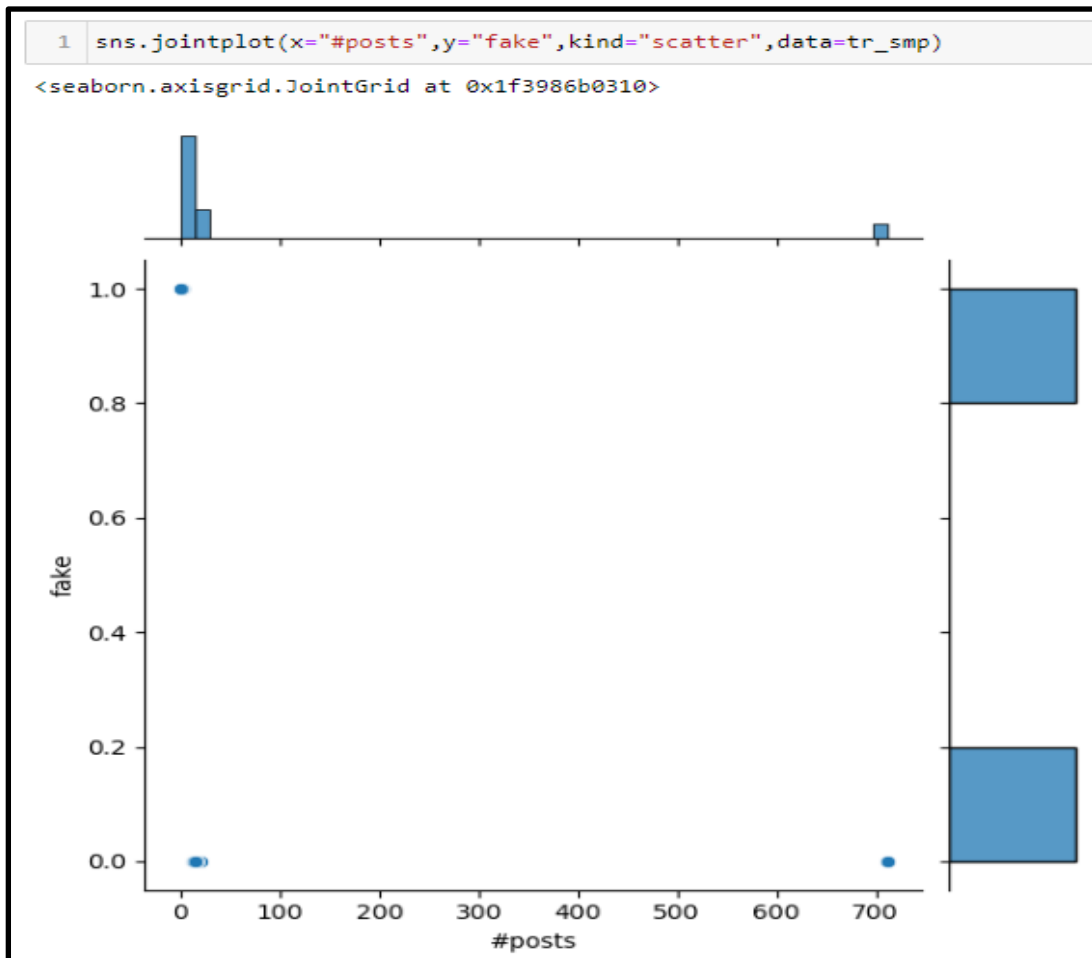
### ➤ Box Plot:



The above box plot shows the distribution in description length of an account against the type of account, namely, real (0) or fake (1). For real accounts, the values occur roughly in the range 0-27, with the median being approximately 18. On the other hand, fake accounts in the given sample do not have a description at all, making its range and median 0. There are also no outliers in the given sample, for both datasets.



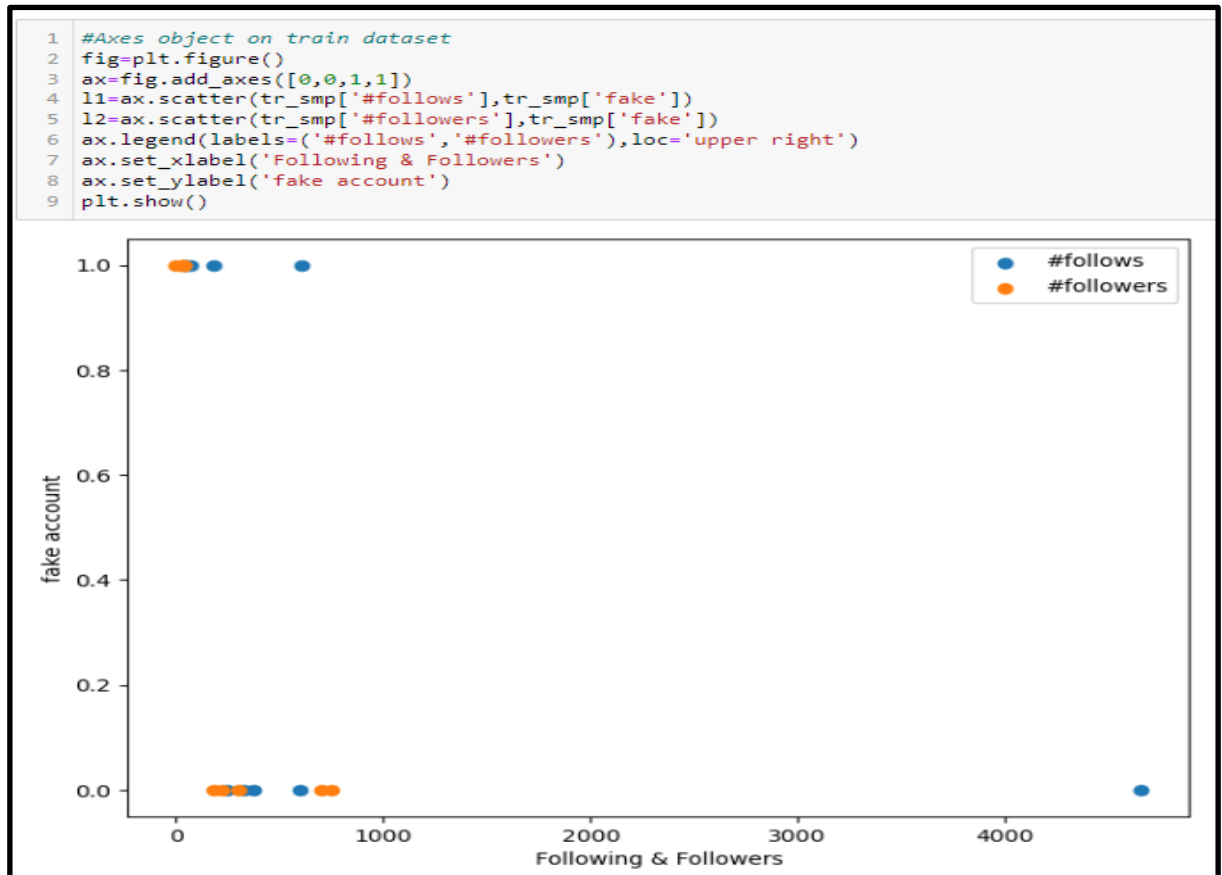
## ➤ Joint Plot:



Fake accounts have less or no posts, they just have the fake profile pic posted. They don't post anything; they track the information and posts from other users. Fake accounts post advertisements like lottery, money debited in your bank account, investment etc. On the other hand, genuine accounts users don't post anything related to such advertisements.

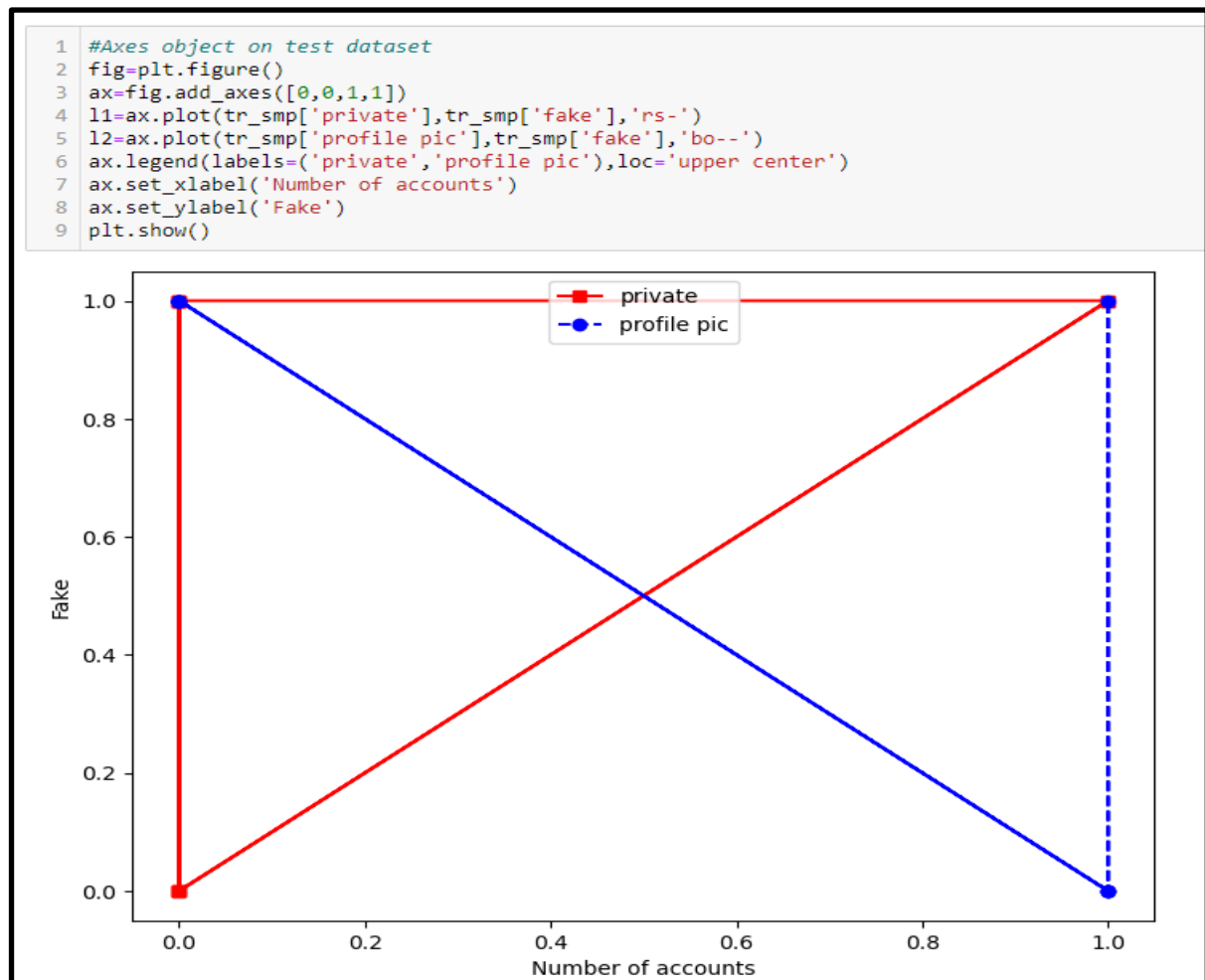
# Trivariate Analysis

## ➤ Scatter Plot:



Fake accounts tend to have a low number of followers and follow very few accounts. On the other hand, real accounts have a relatively larger number of followers and followings. Fake accounts usually have less than 50 followers and approximately 200 followings, with one outlier having higher than 500 follows. In contrast, genuine accounts followings are between 0 and 1000, with an outlier whose followings are more than 4000, implying an influencer account.

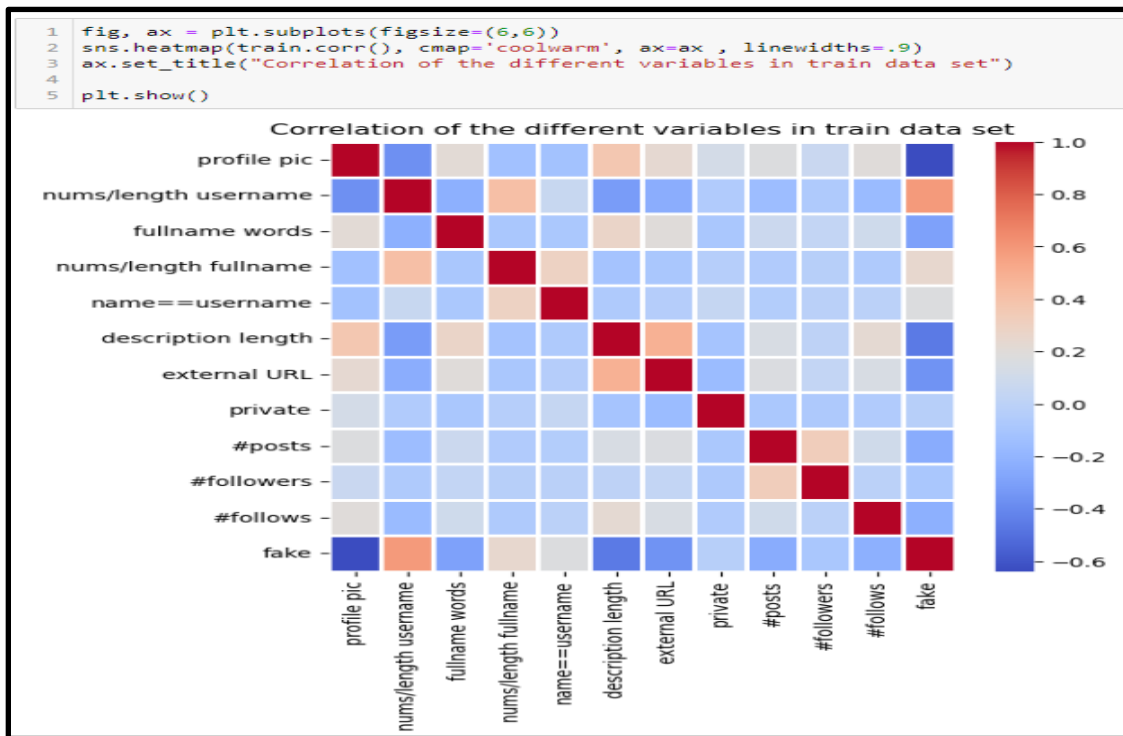
## ➤ Axes Plot:



From the graph, it can be seen that the statistics of an account being private and having a profile picture is the same. Both the parameters appear more in real accounts instead of fake. This suggests that spam accounts are generally public, without a profile picture. Hence, the data hidden by the user plays a factor into determining whether the account is legitimate.

# Multivariate Analysis

## ➤ Heat Map:



This shows the correlation of different variables by various colors. The **light red color** shows a slight positive correlation between the attributes in the corresponding row and column. Similarly, **dark red** shows a high correlation while **light blue** and **dark blue** colors show a negative slight and high correlation respectively. Hence, fake has the highest correlation with itself. It has the highest negative correlation with profile pic attribute and highest positive one with length of username.

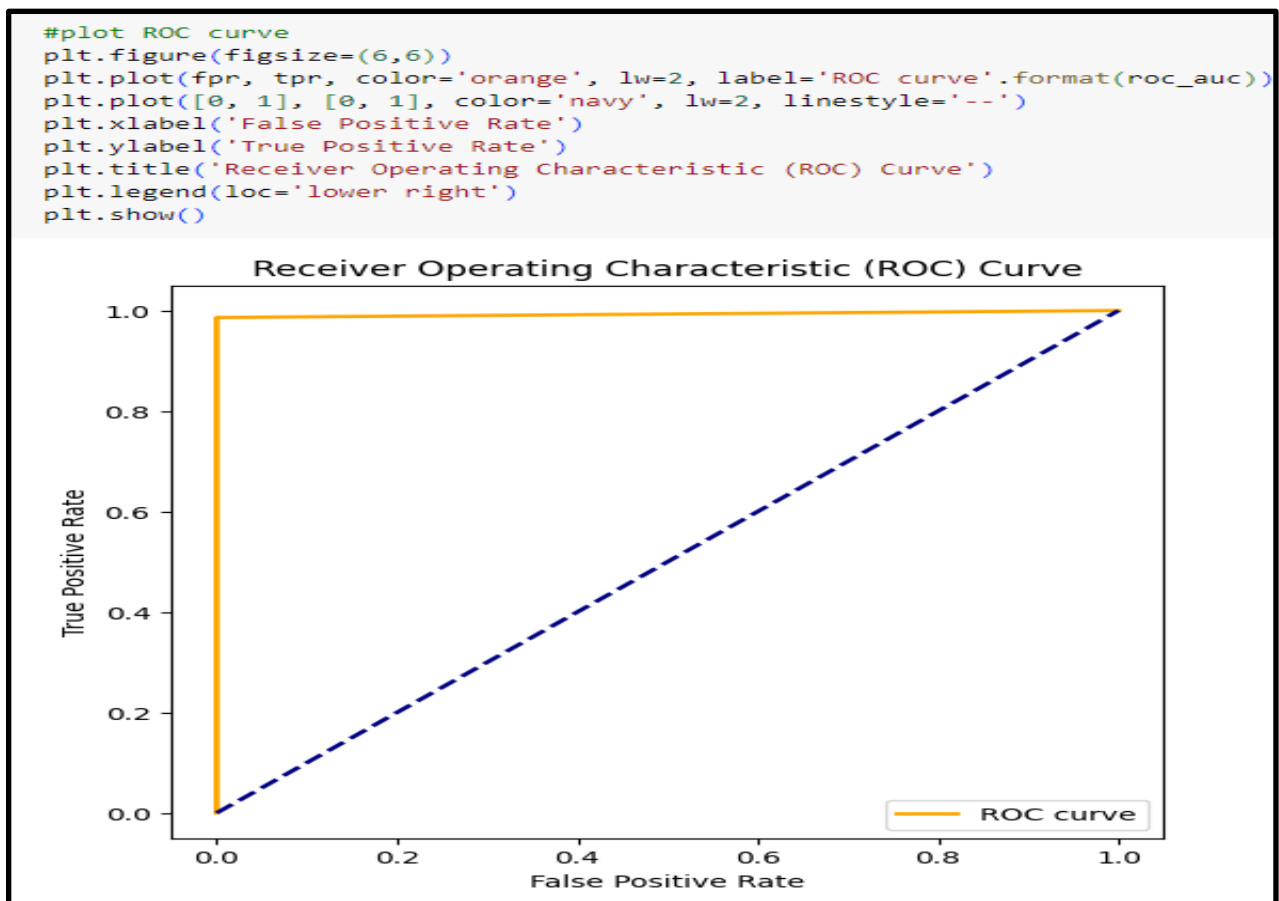
## ➤ PairPlot



The default pairplot shows scatter plots between variables on the upper and lower triangle and histograms along the diagonal. It plots multiple pairwise bivariate distributions in the training dataset.

## ROC Curve

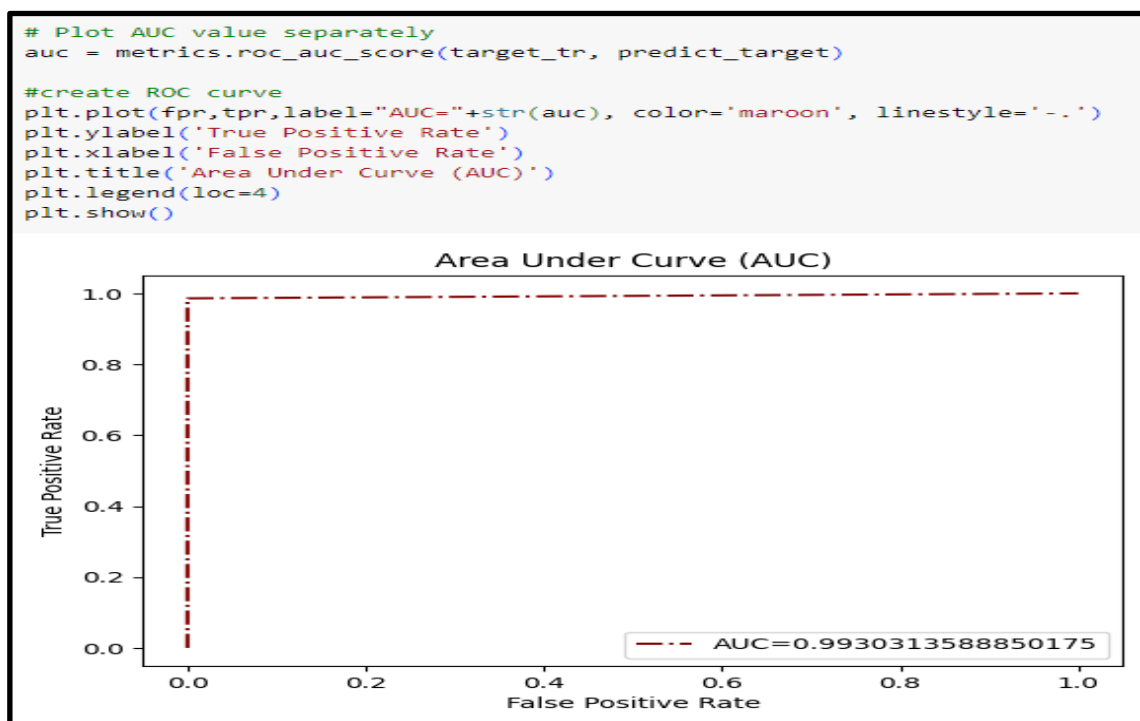
ROC stands for Receiver Operating Characteristics, and the ROC curve is the graphical representation of the effectiveness of the binary classification model. It plots the true positive rate (TPR) vs the false positive rate (FPR) at different classification thresholds.



There are some areas where using AUC might not be ideal. In cases where the dataset is highly imbalanced, *the ROC curve can give an overly optimistic assessment of the model's performance.*

## AUC Curve

The AUC curve or Area Under Curve, is a graphical representation of the performance of a binary classification model at various classification thresholds. It measures the overall performance of the binary classification model.



As both TPR and FPR range between 0 to 1, So, the area will always lie between 0 and 1. Our AUC is 0.993 which means that when we took two data points belonging to separate classes then there is a 99.3% chance the model would be able to segregate them or rank them correctly i.e positive point has a higher prediction probability than the negative class.

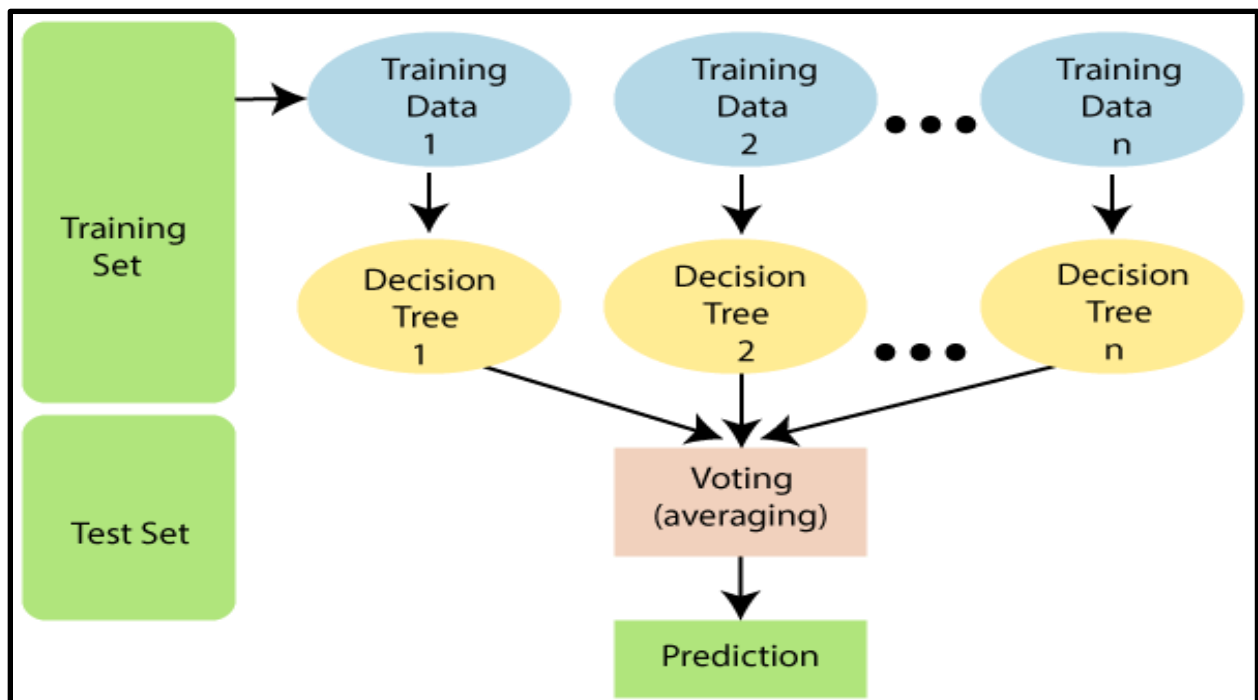
# IX- DATA MODELING

## 1. Random Forest Classifier

To detect spam profiles, a binary classifier is needed to sort the dataset into “fake” and “not fake” categories.

*Random Forest Classifier* Algorithm is a binary classifier that contains a number of decision trees on various subsets of the given dataset and takes the average to improve the predictive accuracy of that dataset.

Instead of relying on one decision tree, the random forest takes the prediction from each tree and based on the majority votes of predictions, it predicts the final output.





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

Parameters that can be used for the algorithm are:

**n\_estimators** : (int, default = 100)

The number of trees in the forest.

**max\_depth** : (int, default = None)

The maximum depth of the tree.

**random\_state** : (int, RandomState instance or None, default = None)

Controls both the randomness of the bootstrapping of the samples used when building trees.

**max\_samples** : (int or float, default = None)

If bootstrap is True, the number of samples to draw from X to train each base estimator.

Model creation:

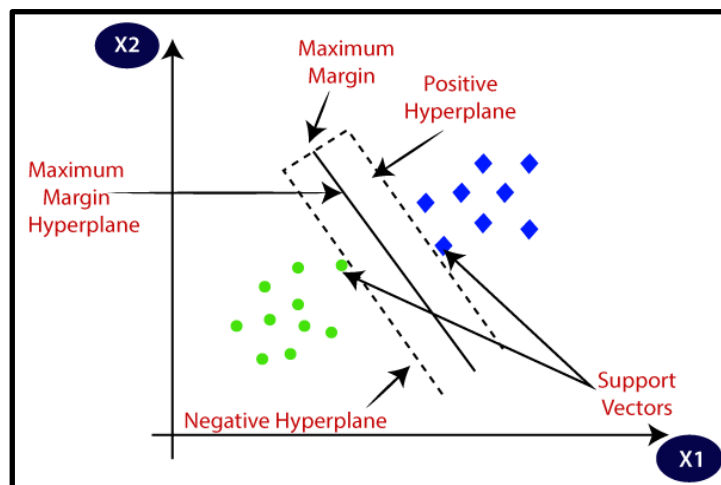
```
# Fitting the Random Forest algorithm to the training set

classifier = RandomForestClassifier(n_estimators=115 , random_state=28 , max_depth = 100 , max_samples = 350 )
classifier.fit(feature_tr,target_tr)
```

```
▼ RandomForestClassifier
RandomForestClassifier(max_depth=100, max_samples=350, n_estimators=115,
                      random_state=28)
```

## 2. Linear SVC

Linear SVMs use a linear decision boundary to separate the data points of different classes. When the data can be precisely linearly separated, linear SVMs are very suitable. This means that a single straight line (in 2D) or a hyperplane (in higher dimensions) can entirely divide the data points into their respective classes. A hyperplane that maximizes the margin between the classes is the decision boundary.



The objective of a Linear SVC (**Support Vector Classifier**) is to fit to the data you provide, returning a "best fit" hyperplane that divides, or categorizes, your data. From there, after getting the hyperplane, you can then feed some features to your classifier to see what the "predicted" class is.

```
# Fitting the Linear SVC algorithm to the training set
classifier2 = LinearSVC(random_state=28, max_iter=1000)
classifier2.fit(feature_tr, target_tr)

/usr/local/lib/python3.10/dist-packages/sklearn/svm/_base.py:1244:
  warnings.warn(
    LinearSVC
  LinearSVC(random_state=28)
```

Parameters that are used for this algorithm are:

**random\_state** (*int, default=None*)

Controls the pseudo random number generation for shuffling the data for the dual coordinate descent (if **dual=True**). When **dual=False** the underlying implementation of LinearSVC is not random and **random\_state** has no effect on the results.

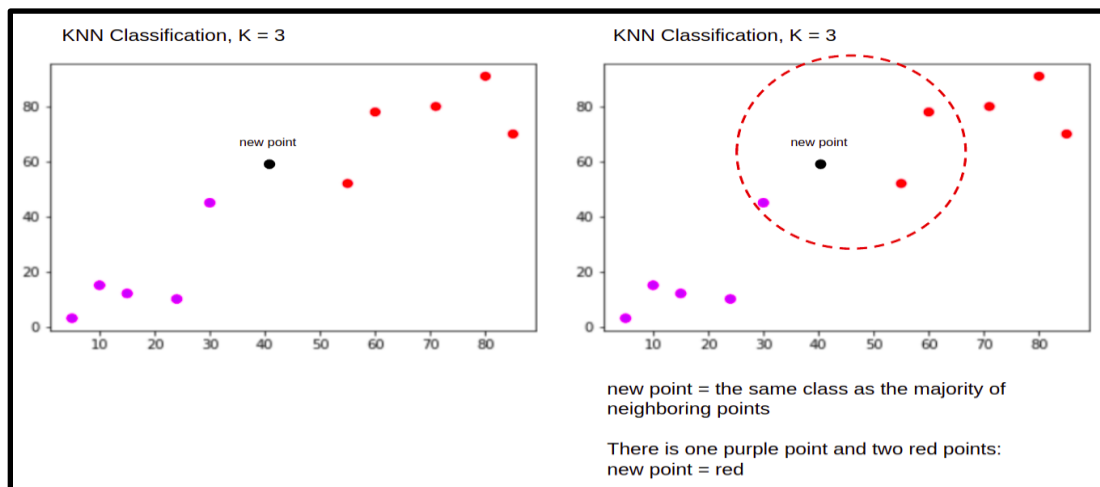
**max\_iter** (*int, default=1000*)

The maximum number of iterations to be run.

### 3. K-Neighbours

The K-nearest neighbor or K-NN algorithm basically creates an imaginary boundary to classify the data. When new data points come in, the algorithm will try to predict that to the nearest of the boundary line. Therefore, a larger k value means smoother curves of separation resulting in less complex models. Whereas, smaller k values tend to overfit the data and result in complex models.

The kNN algorithm can be considered a voting system, where the majority class label determines the class label of a new data point among its nearest 'k' (where k is an integer) neighbors in the feature space. When training a kNN classifier, it's essential to normalize the features. This is because kNN measures the distance between points. The default is to use the Euclidean Distance, which is the square root of the sum of the squared differences between two points.



```
# Fitting the K-Neighbours algorithm to the training set
classifier3 = KNeighborsClassifier(n_neighbors=10 , weights="distance")
classifier3.fit(feature_tr,target_tr)
```

```
▼ KNeighborsClassifier
KNeighborsClassifier(n_neighbors=10, weights='distance')
```

The parameters used for this algorithm are:

**N\_neighbors** (*int, default=5*)

Number of neighbors to use by default for k-neighbors queries.

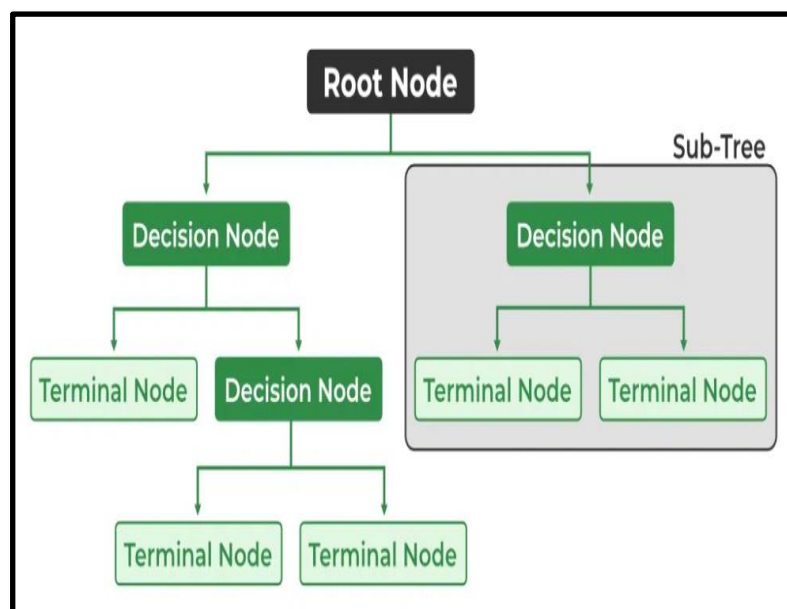
**Weights** (*{'uniform', 'distance'}, callable or None, default='uniform'*)

Weight function used in prediction. **weights='distance'** implies weight points by the inverse of their distance. In this case, closer neighbors of a query point will have a greater influence than neighbors which are further away.

## 4. Decision Tree

A decision tree is a flowchart-like tree structure where each internal node denotes the feature, branches denote the rules and the leaf nodes denote the result of the algorithm. It is a versatile supervised machine-learning algorithm, which is used for both classification and regression problems.

The decision tree operates by analyzing the data set to predict its classification. It commences from the tree's root node, where the algorithm views the value of the root attribute compared to the attribute of the record in the actual data set. Based on the comparison, it proceeds to follow the branch and move to the next node. The algorithm repeats this action for every subsequent node by comparing its attribute values with those of the sub-nodes and continuing the process further. It repeats until it reaches the leaf node of the tree.



```
# Fitting the Decision Tree algorithm to the training set
classifier4 = DecisionTreeClassifier(min_impurity_decrease=0.001)
classifier4.fit(feature_tr,target_tr)
```

```
▼ DecisionTreeClassifier
DecisionTreeClassifier(min_impurity_decrease=0.001)
```

Parameter of this algorithm is:

**Min\_impurity\_decrease** (*float, default=0.0*)

A node will be split if this split induces a decrease of the impurity greater than or equal to this value.

The weighted impurity decrease equation is the following:

$$N_t / N * (\text{impurity} - N_{t\_R} / N_t * \text{right\_impurity} \\ - N_{t\_L} / N_t * \text{left\_impurity})$$

where **N** is the total number of samples, **N<sub>t</sub>** is the number of samples at the current node, **N<sub>t\_L</sub>** is the number of samples in the left child, and **N<sub>t\_R</sub>** is the number of samples in the right child.

## 5. XGBoost

XGBoost stands for “Extreme Gradient Boosting” and is an optimized distributed gradient boosting library designed for efficient and scalable training of machine learning models. It is an ensemble learning method that combines the predictions of multiple weak models to produce a stronger prediction.

In this algorithm, decision trees are created in sequential form. Weights are assigned to all the independent variables which are then fed into the decision tree which predicts results. The weight of variables predicted wrong by the tree is increased and these variables are then fed to the second decision tree. These individual classifiers/predictors then ensemble to give a strong and more precise model. It can work on regression, classification, ranking, and user-defined prediction problems.





# X- MODEL EVALUATION

In **machine learning**, classification is the process of categorizing a given set of data into different categories. After training the model appropriately, the model is tested on new data values to check its performance. Predicted values are those values, which are predicted by the model, and actual values are the true values for the given observations.

```
# Predicting the Test Set result  
predict_target = classifier.predict(feature_te)
```

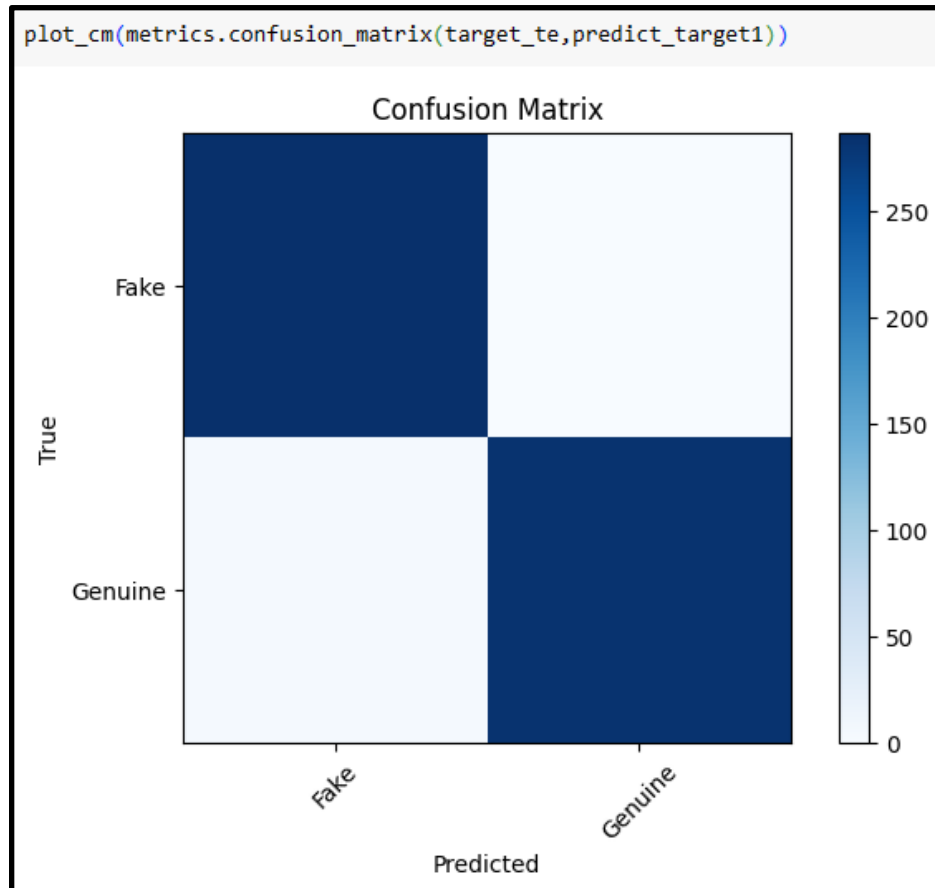
A **confusion matrix** is a matrix that summarizes the performance of a machine learning model on a set of test data.

The matrix is divided into two dimensions, that are **predicted values** and **actual values** along with the total number of predictions. Following are the specifications of classification report:

- **True Positives (TP):** When the actual value is Positive and predicted is also Positive.
- **True negatives (TN):** When the actual value is Negative and prediction is also Negative.
- **False positives (FP):** When the actual is negative but prediction is positive. Also called the **Type 1 error**.
- **False negatives (FN):** When the actual is Positive but prediction is Negative. Also called the **Type 2 error**.

# I. Random Forest Algorithm :

## Confusion Matrix



- **True Positives (TP)**: The model has 283 TP predictions.
- **True negatives (TN)**: The model has 287 TN predictions.
- **False positives (FP)**: There are no FP predictions made by the model. Hence, the algorithm does not have Type 1 error.
- **False negatives (FN)**: Model made 4 FN predictions and thus has a slight Type 2 error.

## Classification Report

```
print("\nClassification report:\n")
print(metrics.classification_report(target_te,predict_target))
```

Classification report:

|              | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| 0            | 0.99      | 1.00   | 0.99     | 287     |
| 1            | 1.00      | 0.99   | 0.99     | 287     |
| accuracy     |           |        | 0.99     | 574     |
| macro avg    | 0.99      | 0.99   | 0.99     | 574     |
| weighted avg | 0.99      | 0.99   | 0.99     | 574     |

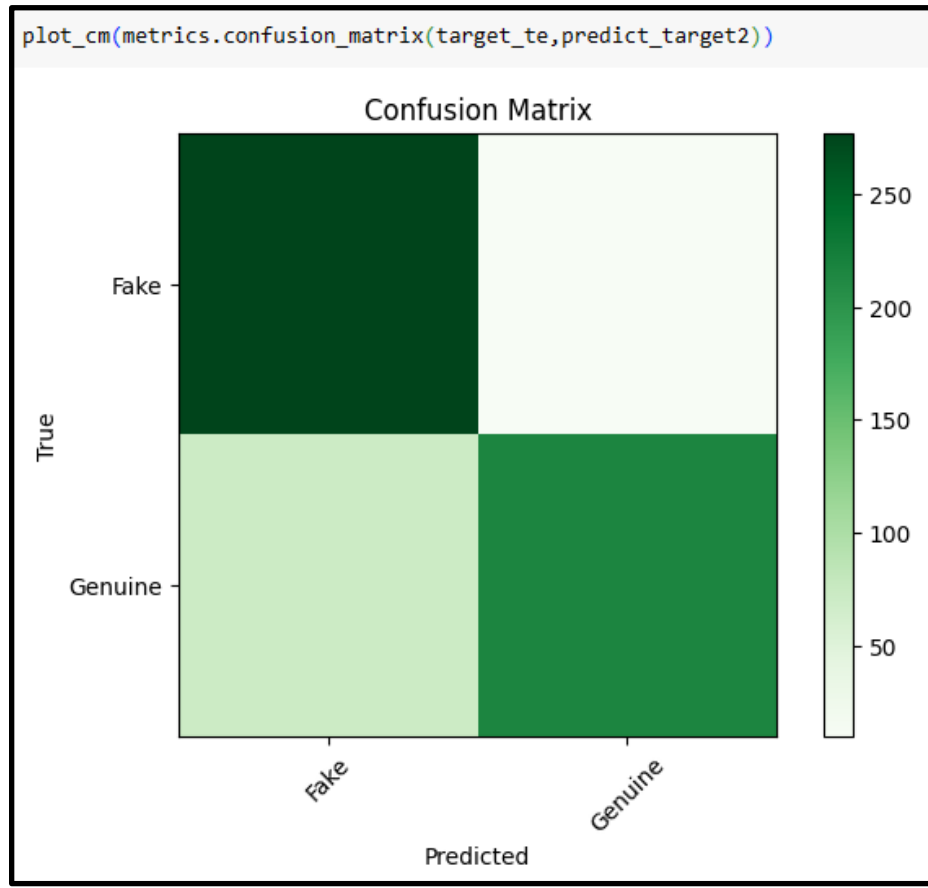
```
print("\nAverage accuracy score of model is:\n")
print(metrics.accuracy_score(target_te,predict_target))
```

Average accuracy score of model is:

0.9930313588850174

## II. Linear SVC :

### Confusion Matrix



- **True Positives (TP)**: The model has 216 TP predictions.
- **True negatives (TN)**: The model has 277 TN predictions.
- **False positives (FP)**: The model has 10 FP predictions. Hence has a bit of Type 1 error.
- **False negatives (FN)**: Model made 71 FN predictions and thus has a much of Type 2 error.

## Classification Report

```
print("\nClassification report:\n")
print(metrics.classification_report(target_te,predict_target2))
```

Classification report:

|              | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| 0            | 0.80      | 0.97   | 0.87     | 287     |
| 1            | 0.96      | 0.75   | 0.84     | 287     |
| accuracy     |           |        | 0.86     | 574     |
| macro avg    | 0.88      | 0.86   | 0.86     | 574     |
| weighted avg | 0.88      | 0.86   | 0.86     | 574     |

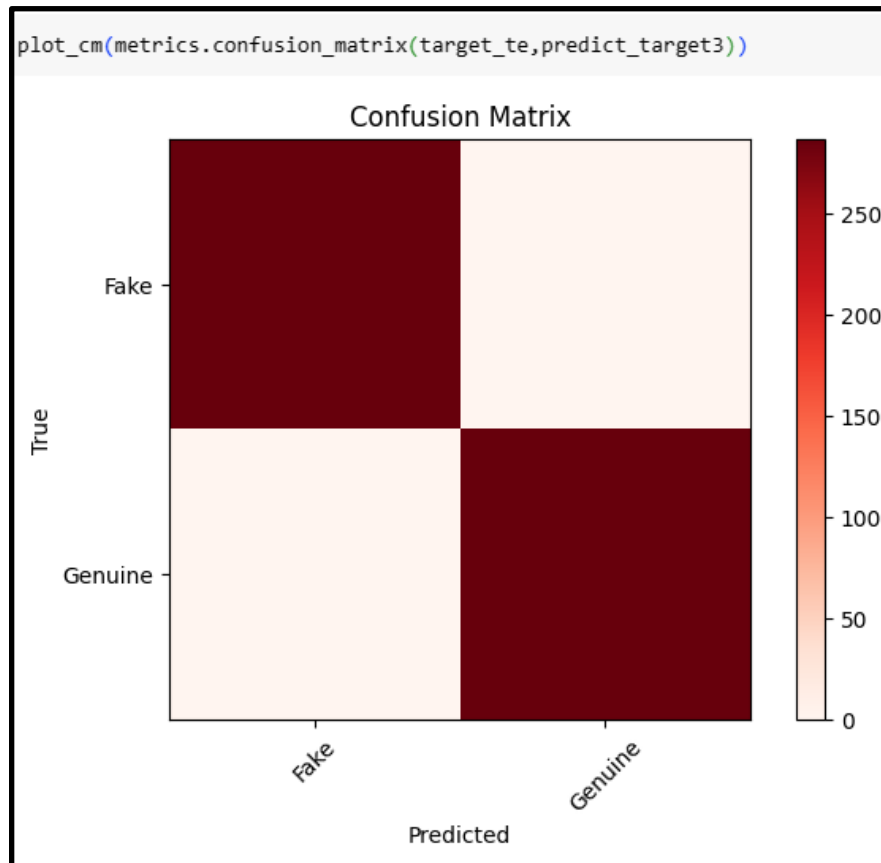
```
print("\nAverage accuracy score of Linear SVC model is:\n")
print(metrics.accuracy_score(target_te,predict_target2))
```

Average accuracy score of Linear SVC model is:

0.8588850174216028

### III. K-Neighbors :

#### Confusion Matrix



- **True Positives (TP)**: The model has 287 TP predictions i.e all positive values are correctly predicted.
- **True negatives (TN)**: The model has 287 TN predictions i.e all negative values are correctly predicted.
- **False positives (FP)**: The model has 0 Type 1 error.
- **False negatives (FN)**: Model made 0 FN predictions and thus has no Type 2 error.

## Classification Report

```
print("\nClassification report:\n")
print(metrics.classification_report(target_te,predict_target3))
```

Classification report:

|              | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| 0            | 1.00      | 1.00   | 1.00     | 287     |
| 1            | 1.00      | 1.00   | 1.00     | 287     |
| accuracy     |           |        | 1.00     | 574     |
| macro avg    | 1.00      | 1.00   | 1.00     | 574     |
| weighted avg | 1.00      | 1.00   | 1.00     | 574     |

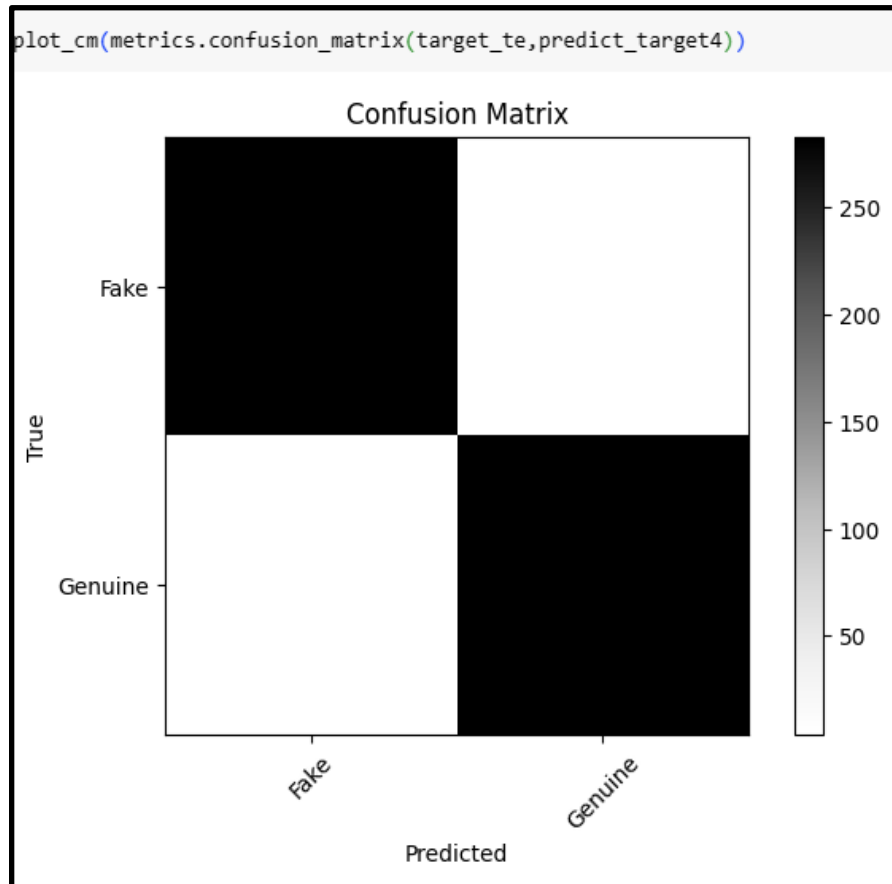
```
print("\nAverage accuracy score of K-Neighbour model is:\n")
print(metrics.accuracy_score(target_te,predict_target3))
```

Average accuracy score of K-Neighbour model is:

1.0

## IV. Decision Trees :

### Confusion Matrix



- **True Positives (TP):** The model has 283 TP predictions .
- **True negatives (TN):** The model has 283 TN predictions .
- **False positives (FP):** The model has 4 Type 1 errors. Thus have a slight Type 1 error.
- **False negatives (FN):** Model made 4 FN predictions and thus has a slight of Type 2 error.



## Classification Report

```
print("\nClassification report:\n")
print(metrics.classification_report(target_te,predict_target4))
```

Classification report:

|              | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| 0            | 0.99      | 0.99   | 0.99     | 287     |
| 1            | 0.99      | 0.99   | 0.99     | 287     |
| accuracy     |           |        | 0.99     | 574     |
| macro avg    | 0.99      | 0.99   | 0.99     | 574     |
| weighted avg | 0.99      | 0.99   | 0.99     | 574     |

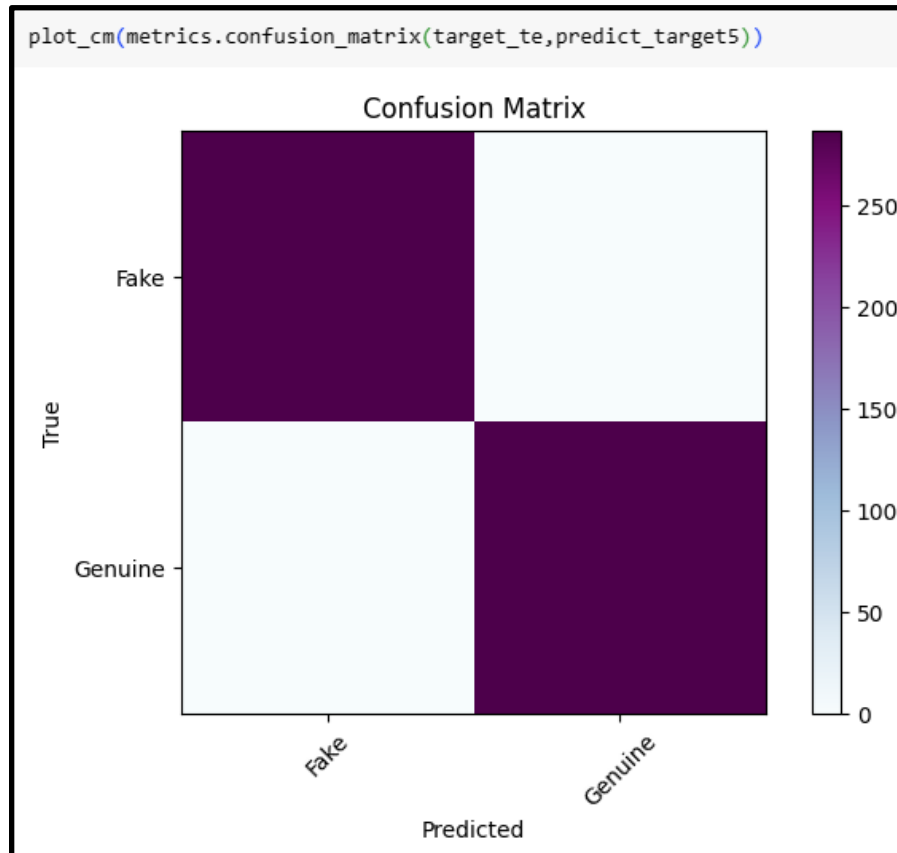
```
print("\nAverage accuracy score of decision tree model is:\n")
print(metrics.accuracy_score(target_te,predict_target4))
```

Average accuracy score of decision tree model is:

0.9860627177700348

## V. XGBoost :

### Confusion Matrix



- **True Positives (TP)**: The model has 287 TP predictions i.e all positive values are correctly predicted.
- **True negatives (TN)**: The model has 287 TN predictions i.e all negative values are correctly predicted.
- **False positives (FP)**: The model has 0 Type 1 error.
- **False negatives (FN)**: Model made 0 FN predictions and thus has no Type 2 error.

## Classification Report

```
print("\nClassification report:\n")
print(metrics.classification_report(target_te,predict_target5))
```

Classification report:

|              | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| 0            | 1.00      | 1.00   | 1.00     | 287     |
| 1            | 1.00      | 1.00   | 1.00     | 287     |
| accuracy     |           |        | 1.00     | 574     |
| macro avg    | 1.00      | 1.00   | 1.00     | 574     |
| weighted avg | 1.00      | 1.00   | 1.00     | 574     |

```
print("\nAverage accuracy score of XGBoost model is:\n")
print(metrics.accuracy_score(target_te,predict_target5))
```

Average accuracy score of XGBoost model is:

1.0

# XI- MODEL COMPARISON

Comparing Machine Learning Algorithms (MLAs) are important to come out with the best-suited algorithm for a particular problem. The MLAs we have taken up for comparison are **Random Forest classifier**, **K-nearest neighbor classifier**, **Decision Tree classifier**, **eXtreme Gradient Boosting (XGBoost)** and **Linear Support Vector Machine (SVM)**.

- Firstly, we initialized a list of Machine Learning Algorithms we selected.
- Preprocessing, Exploratory Analysis, Evaluation of the data was previously done so we stored the results in a pandas Dataframe (MLA\_compare).
- It sorts the results based on Accuracy in descending order and displays the comparison table (MLA\_compare).

```
[197] MLA = [#random forest          #Linear SVC      #K-Neighbours      #Decision Tree      #XGBoost
           ensemble.RandomForestClassifier(), svm.LinearSVC(), neighbors.KNeighborsClassifier(), tree.DecisionTreeClassifier(), xgb.XGBClassifier()]
MLA_columns = []
MLA_compare = pd.DataFrame(columns = MLA_columns)
```

```
[198] row_index = 0
      for alg in MLA:
          predicted = alg.fit(feature_tr, target_tr).predict(feature_te)
          fp, tp, th = roc_curve(target_te, predicted)
          MLA_name = alg.__class__.__name__
          MLA_compare.loc[row_index, 'Algorithms'] = MLA_name
          MLA_compare.loc[row_index, 'Accuracy'] = metrics.accuracy_score(target_te, predicted)
          MLA_compare.loc[row_index, 'Precision'] = metrics.precision_score(target_te, predicted)
          MLA_compare.loc[row_index, 'Recall'] = metrics.recall_score(target_te, predicted)
          MLA_compare.loc[row_index, 'F1-score'] = metrics.f1_score(target_te, predicted)

          row_index += 1

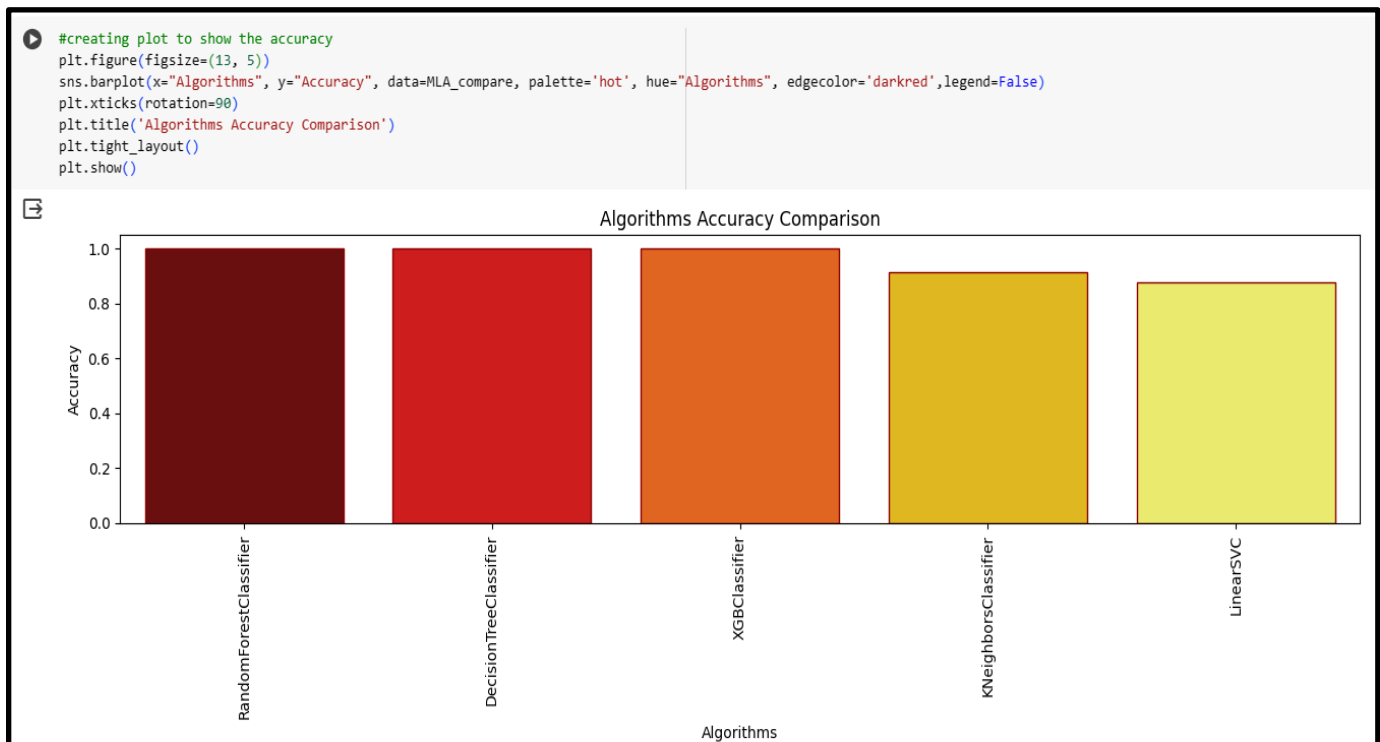
MLA_compare.sort_values(by='Accuracy', ascending=False, inplace=True)
MLA_compare
```

/usr/local/lib/python3.10/dist-packages/sklearn/svm/\_base.py:1244: ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.  
warnings.warn(

|   | Algorithms             | Accuracy | Precision | Recall   | F1-score |
|---|------------------------|----------|-----------|----------|----------|
| 0 | RandomForestClassifier | 1.000000 | 1.000000  | 1.000000 | 1.000000 |
| 3 | DecisionTreeClassifier | 1.000000 | 1.000000  | 1.000000 | 1.000000 |
| 4 | XGBClassifier          | 1.000000 | 1.000000  | 1.000000 | 1.000000 |
| 2 | KNeighborsClassifier   | 0.916376 | 0.910653  | 0.923345 | 0.916955 |
| 1 | LinearSVC              | 0.879791 | 0.957983  | 0.794425 | 0.868571 |

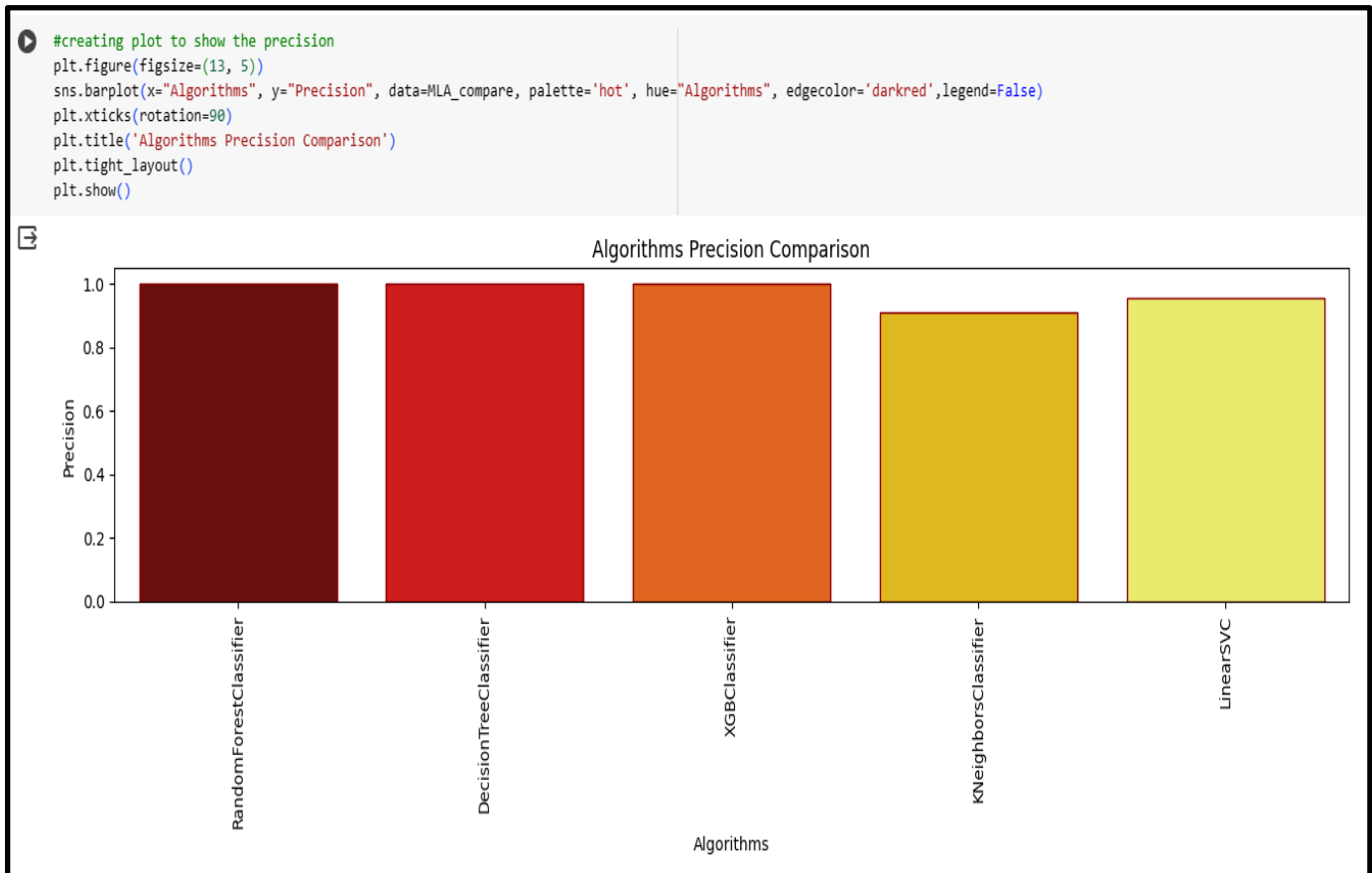
The following bar plots visualizes the attributes of classification report of different machine learning models. Each bar represents an algorithm labeled along the x-axis. The height of each bar indicates the report achieved by the models.

### Comparison of Machine Learning Algorithms on the basis of Accuracy :



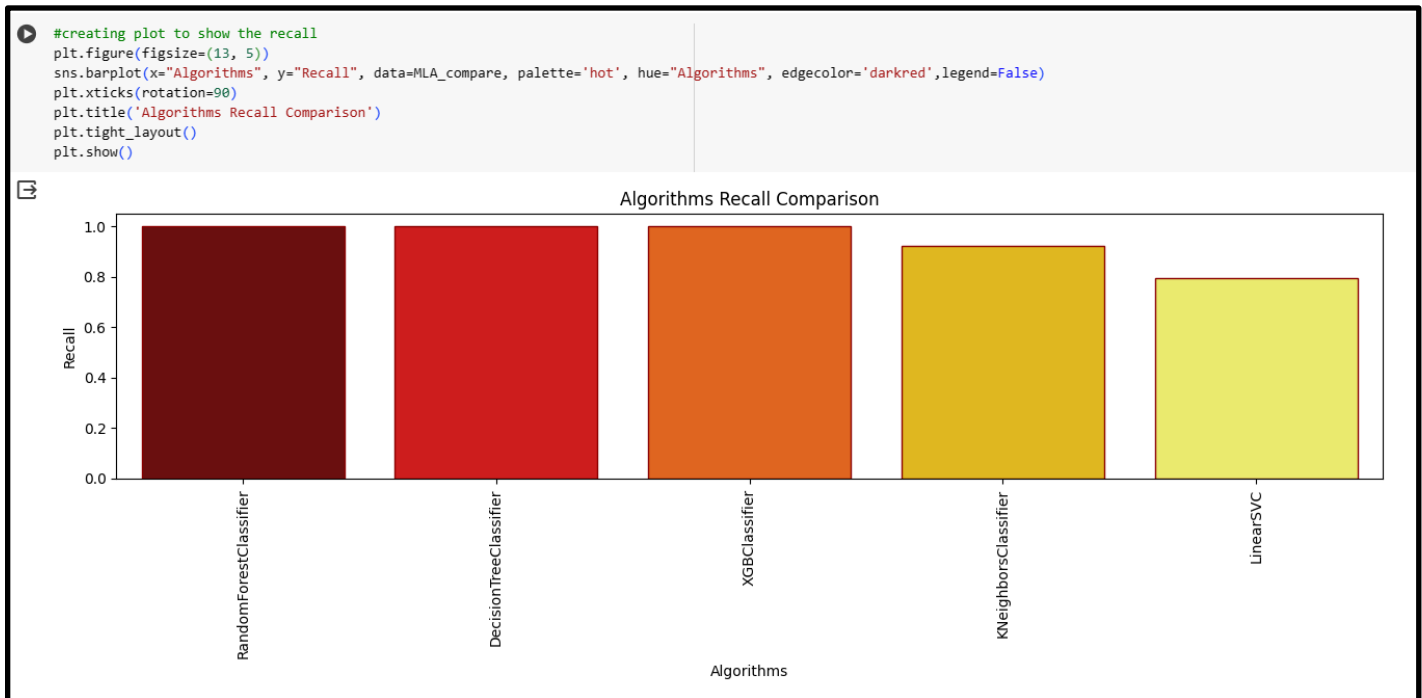
According to the above plot, we can interpret that the **Random Forest Classifier** gives better accuracy as compared to other algorithms. Random Forest accuracy is 99.3%, Decision Tree accuracy is 98.6%, XGBoost accuracy is 97.4%, K-Neighbor accuracy is 91.6%, Linear SVC accuracy is 87.9%.

## Comparison of Machine Learning Algorithms on the basis of Precision :



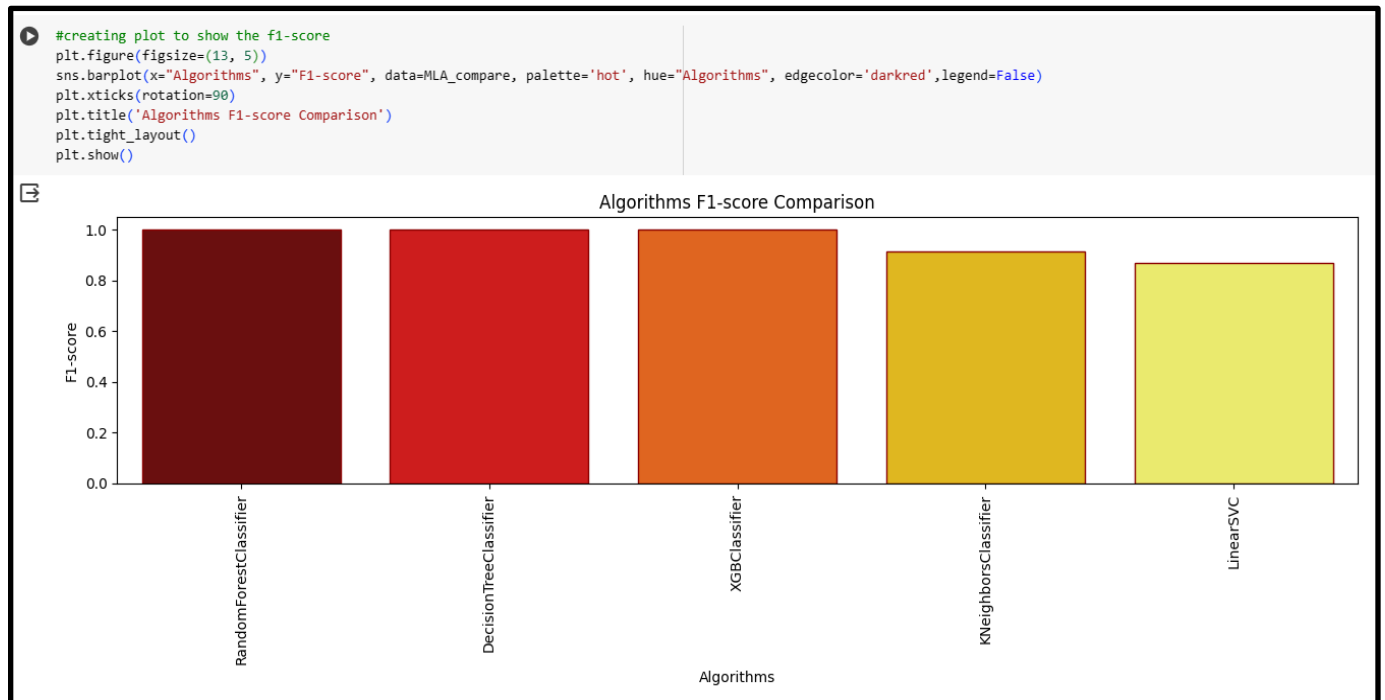
From the above plot, we can interpret that the **Random Forest Classifier** gives better Precision as compared to other algorithms. Random Forest Precision is 99%, Decision Tree Precision is 98%, XGBoost Precision is 97%, K-Neighbor Precision is 91.06%, Linear SVC Precision is 95.79%.

## Comparison of Machine Learning Algorithms on the basis of Recall:



From the above plot, we can interpret that the **Random Forest Classifier** & **XGBoost** gives better Recall as compared to other algorithms. Random Forest Recall is 100%, Decision Tree Recall is 99%, XGBoost Recall is 100%, K-Neighbor Recall is 92.3%, Linear SVC Recall is 79.45%.

## Comparison of Machine Learning Algorithms on the basis of F1-Score:



Above plot shows that the **XGBoost** gives better F1-score as compared to other algorithms. Random Forest F1-score is 99%, Decision Tree F1-score is 99%, XGBoost F1-score is 100%, K-Neighbor F1-score is 91.6%, Linear SVC F1-score is 86.54%.

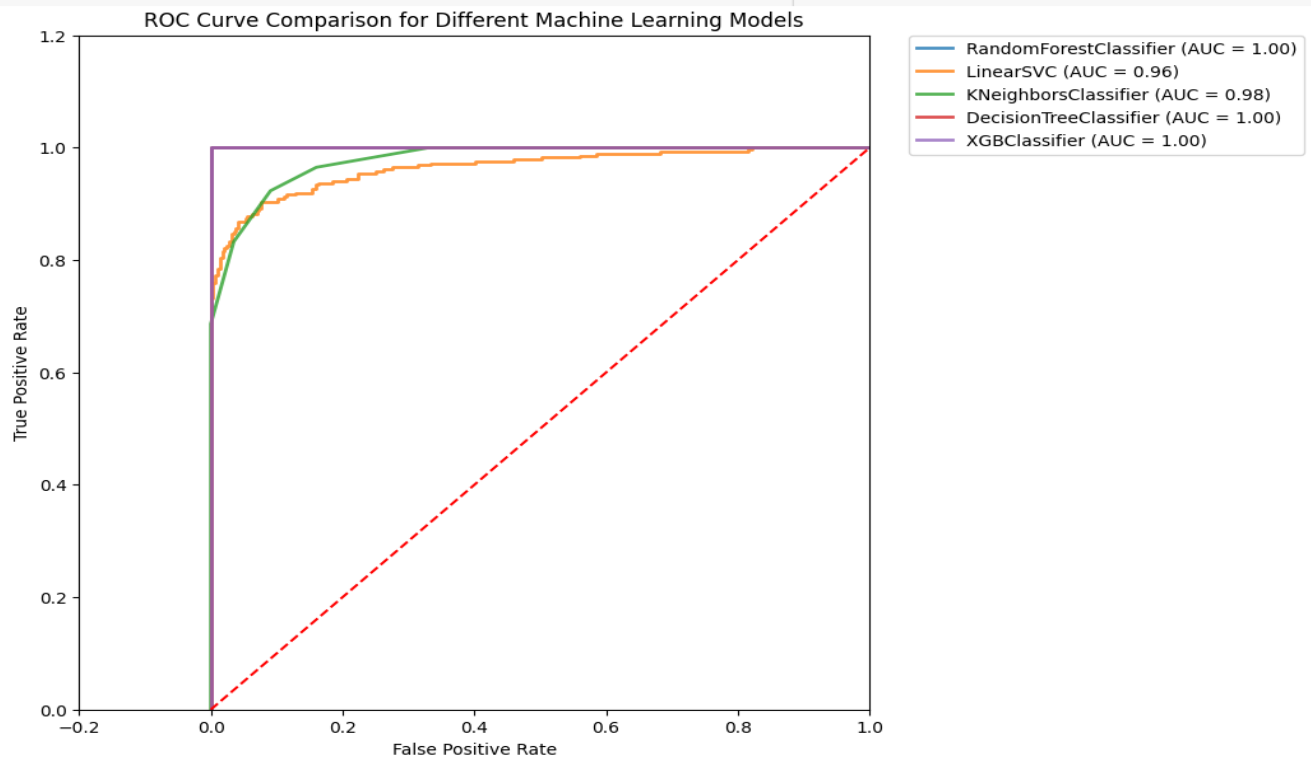


## Comparison of Machine Learning Algorithms on the basis of ROC Curve :

The area under the ROC curve is often called AUC and it is also a good measure of the predictability of the machine learning algorithms. A higher AUC is an indication of more accurate prediction.

```
#creating plot to show ROC for different Algorithm
plt.figure(figsize=(8, 8))
for alg in MLA:
    if hasattr(alg, "predict_proba"): # Check if the algorithm supports probability estimates
        predicted = alg.fit(feature_tr, target_tr).predict_proba(feature_te)[: , 1]
    elif hasattr(alg, "decision_function"): # For LinearSVC which doesn't have predict_proba
        calibrated_clf = CalibratedClassifierCV(alg) # Calibrate probabilities using CalibratedClassifierCV
        calibrated_clf.fit(feature_tr, target_tr)
        predicted = calibrated_clf.predict_proba(feature_te)[: , 1]
    else:
        predicted = alg.fit(feature_tr, target_tr).decision_function(feature_te) # For algorithms like SVM
    fp, tp, _ = roc_curve(target_te, predicted)
    roc_auc_mla = metrics.auc(fp, tp)
    MLA_name = alg.__class__.__name__
    plt.plot(fp, tp, lw=2, alpha=0.8, label='%s (AUC = %0.2f)' % (MLA_name, roc_auc_mla))

plt.title('ROC Curve Comparison for Different Machine Learning Models')
plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)
plt.plot([0, 1], [0, 1], 'r--')
plt.xlim([-0.2, 1])
plt.ylim([0, 1.2])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```



## **XII- WEBSITE SPECIFICATIONS**

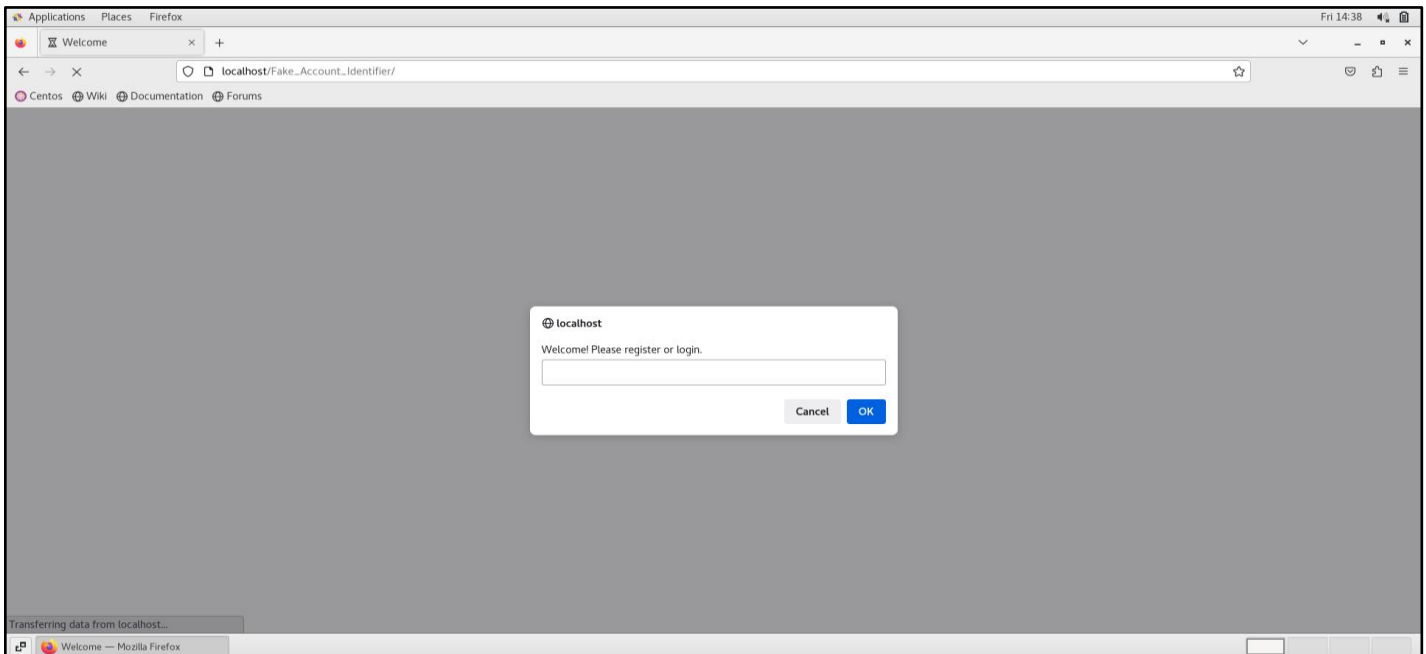
To demonstrate our project, we made a User Authenticated based website in which users need to Login or Register with their desired credential to access the website.

The AI model checks the user's credentials and categorizes it as Fake or Genuine. The user is allowed to view the main content of the website only if their credentials are Genuine.

### **Steps involved are:**

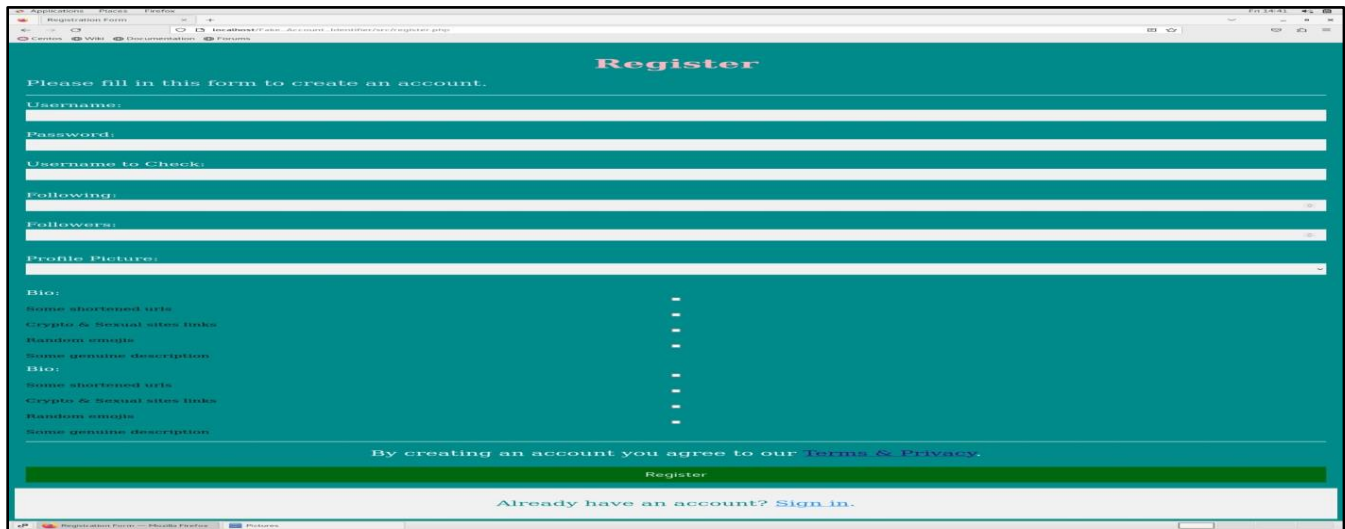
- **Prompt Box** - When the user clicks on the project folder a prompt box would appear asking the user whether he/she wants to “Login” or “Register”. On typing the desired option, it will redirect to the asked option.

(a). Prompt box



- **Registration Page** - This page contains some required inputs from the user by which he/she can login for further process. The user is then classified as “Real” or “Fake” and the inputs are saved in the database.

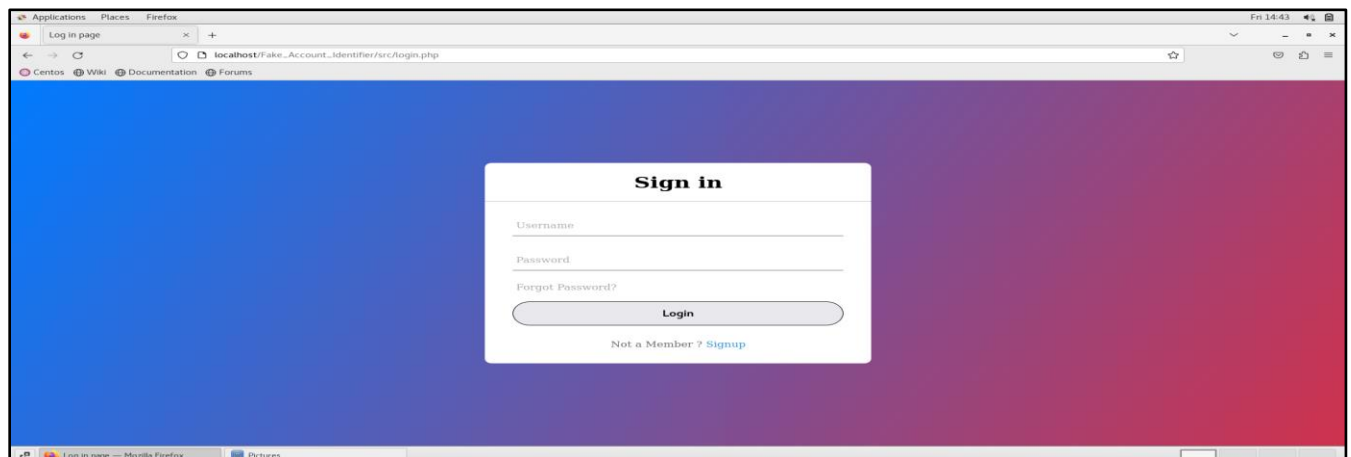
### (b). Registration Page



The screenshot shows a web browser window with a registration form. The form is titled "Register" and has a teal background. It contains the following fields: Username, Password, Username to Check, Following, Followers, Profile Picture, Bio, and a list of social media links (Some shortened url, Crypto & Social sites links, Random image, Some genuine description). Below the form, there is a green bar with the text "By creating an account you agree to our Terms & Privacy." and a "Register" button. At the bottom, there is a link "Already have an account? Sign in."

- **Login/Sign in Page** - Users should remember the credentials from the registration page. If the username & password matches with the already registered data & if it’s “Real” then it would redirect the websites to “Home Page” otherwise it would show “Fake account Logged off!!”.

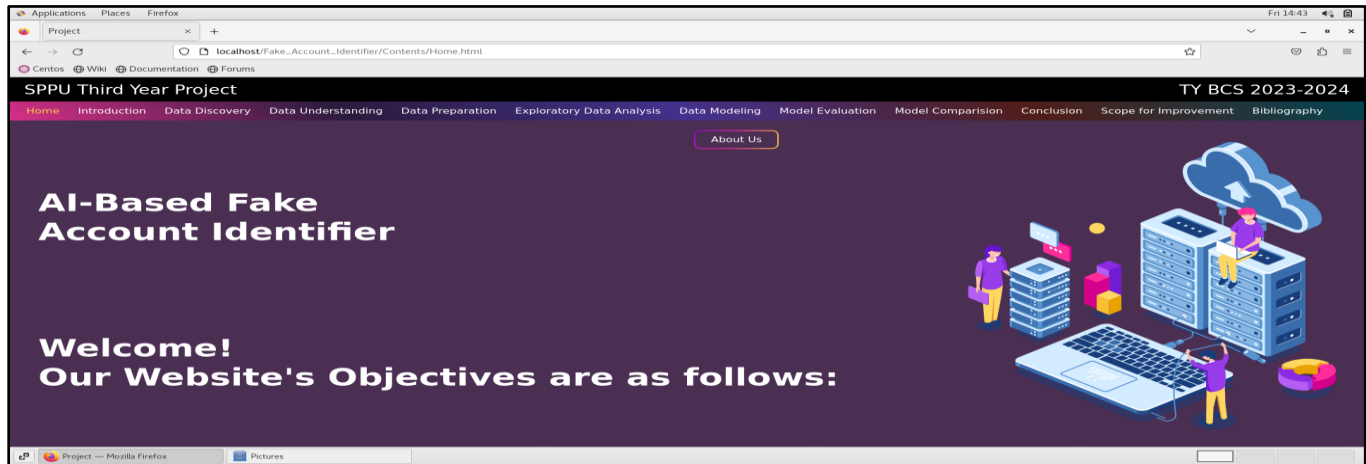
### (c) . Login/Sign in Page



The screenshot shows a web browser window with a login/sign in form. The form is titled "Sign in" and has a white background. It contains the following fields: Username, Password, and a "Forgot Password?" link. Below the form, there is a "Login" button and a link "Not a Member ? Signup". The background of the page is a gradient of blue and purple.

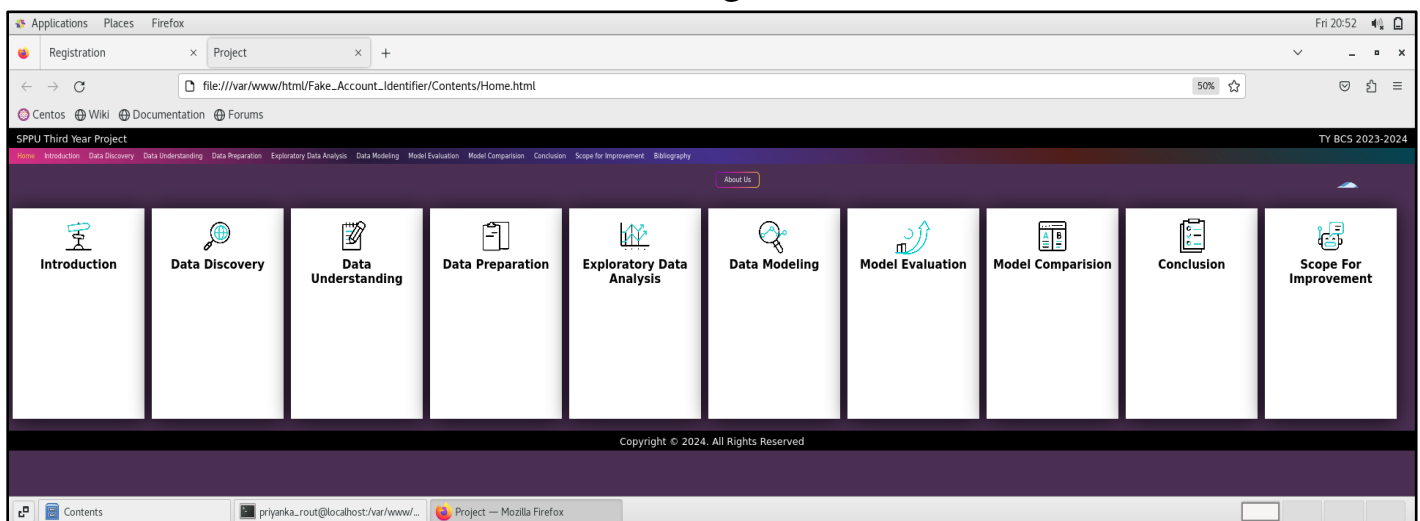
- **Home Page** - Authorized user is redirected to this page, which describes the aim and contents of the website in brief. The user can navigate to the desired section by using the navigation bar.

(d). Home Page



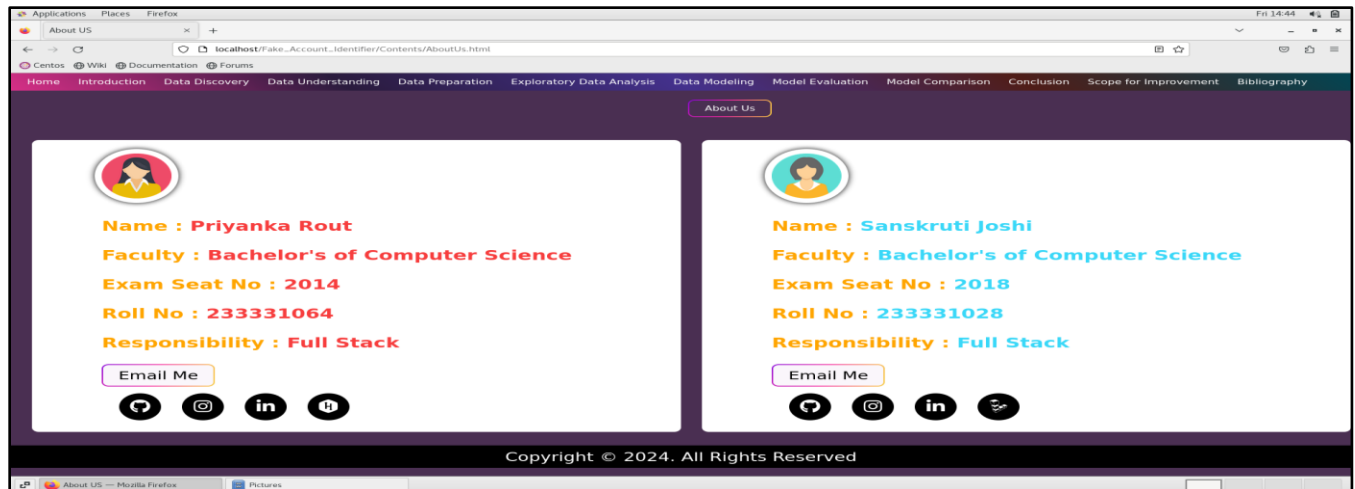
- **Web pages of Model stages** - The project is structured according to the basic life cycle of a Data Analytics Project. Thus, each step of this process, namely, Data Discovery, Data Understanding, Data Preparation, EDA, Data Modeling and Comparison, and finally, the end result of the project have been displayed separately in individual web pages, which can be accessed by using the navigation bar.

(e). Model Stages



- **About Us page** - This is the final web page which gives the basic details about the AI model creators and the website's developers. Their contact information can be accessed by clicking on the desired platform's icon.

#### (f). About Page



- **Database description** - MySQL database has been used to store the credentials of the users, along with their profile results, ie, Fake or Genuine. The website accesses this table to authorize and authenticate the visitor.

#### (g). MYSQL Table

Terminal window showing MySQL database queries and results:

```

MariaDB [Fake]> select * from register;
+----+-----+-----+-----+-----+-----+-----+-----+-----+
| id | username | password | username_to_check | followers | following | profile_pic_option | bio_options | result |
+----+-----+-----+-----+-----+-----+-----+-----+-----+
1 row in set (0.003 sec)

MariaDB [Fake]> select * from register;
+----+-----+-----+-----+-----+-----+-----+-----+-----+
| id | username | password | username_to_check | followers | following | profile_pic_option | bio_options | result |
+----+-----+-----+-----+-----+-----+-----+-----+-----+
2 rows in set (0.000 sec)

MariaDB [Fake]> select * from register;
+----+-----+-----+-----+-----+-----+-----+-----+-----+
| id | username | password | username_to_check | followers | following | profile_pic_option | bio_options | result |
+----+-----+-----+-----+-----+-----+-----+-----+-----+
3 rows in set (0.005 sec)

MariaDB [Fake]> select * from register;
+----+-----+-----+-----+-----+-----+-----+-----+-----+
| id | username | password | username_to_check | followers | following | profile_pic_option | bio_options | result |
+----+-----+-----+-----+-----+-----+-----+-----+-----+
4 rows in set (0.000 sec)

```

| id | username | password | username_to_check | followers | following | profile_pic_option    | bio_options                                | result |
|----|----------|----------|-------------------|-----------|-----------|-----------------------|--|--------|
| 1  | roy      | abc      | roy_23            | 260       | 250       | GenuineHighResolution | Genuine description                        | Real   |
| 2  | priya    | singh2   | pgfdnrtihmhf1234  | 607       | 242       | NoProfilePic          | Shortened URLs Random emojis               | Fake   |
| 3  | saarvi   | saqah23  | ncgsuddkpsondx    | 200       | 345       | LowResolutionPic      | Crypto & Sexual sites links                | Real   |
| 4  | harsh    | harsh23  | ncgsuddkpsondx    | 61        | 1800      | LowResolutionPic      | Shortened URLs Crypto & Sexual sites links | Fake   |

## **XIII- CONCLUSION**

Based on the information that is easily accessible, we employed various machine learning algorithms to identify fake accounts.

From the five algorithms chosen, it can be seen that Random Forest Classifier has the best performance, considering accuracy, recall, precision, etc. It is closely followed by Decision Tree Classifier and XGBoost Classifier while K-Neighbours and Linear SVC's performances were comparatively subpar.

➤ **This model has several uses:**

1. It helps to prevent cyberbullying and harassment by reducing spamming and phishing attacks.
2. Cybercrimes such as identity theft and fraud can be monitored and controlled.
3. It will reduce the spread of fake news and misinformation.

➤ **However, this project also has some limitations, namely:**

1. The model only utilizes visible data and lacks real-time applications.
2. It can be difficult to detect sophisticated fake profiles that use advanced techniques such as machine learning.
3. Fake profiles that are created by humans instead of bots may not be classified correctly by the model.

Fake profile detection is a useful technique for various application domains, such as:

- **Online security and privacy:**

Fake profiles can pose serious threats to the security and privacy of online users, as they can be used for phishing, identity theft, fraud, cyberbullying, and harassment. Detecting and removing fake profiles can help protect the personal information and reputation of online users.

- **Online marketing and advertising:**

Fake profiles can manipulate the online market and influence consumer behavior, as they can be used for spamming, fake reviews, click fraud, and social bots. Detecting and removing fake profiles can help improve the quality and credibility of online marketing and advertising campaigns.

- **Online social and political movements:**

Fake profiles can affect the online social and political movements, as they can be used for spreading fake news, misinformation, propaganda, and polarization. Detecting and removing fake profiles can help promote the authenticity and diversity of online social and political discourse.

## **XIV- SCOPE FOR IMPROVEMENT**

- ❖ Tweaking the number of iterations in Linear SVC and k values in K-Neighbors algorithms can significantly improve their performance.
- ❖ Further jobs may be completed by running a neural network on the numerical, categorical, and profile photo data.
- ❖ Better outcomes could also occur from the addition of new parameters, the fusion of different models, and the creation of a real-time model.
- ❖ With some modifications, the model will be ready for more social media platforms like LinkedIn, Snapchat, WeChat, QQ, etc.
- ❖ Incorporating more features and indicators that can help distinguish between genuine and fake profiles, such as profanity, gender, sentiment, behavior, and network structure.
- ❖ Addressing the ethical and privacy issues involved in fake profile detection, such as respecting the user's consent, avoiding discrimination and bias, and ensuring transparency and accountability.



## **XV- BIBLIOGRAPHY**

- *GeeksForGeeks* : [geeksforgeeks.org](http://geeksforgeeks.org)
- *GitHub* : [randomforestclassifier](https://github.com/randomforestclassifier)
- *Javatpoint* : [javatpoint.com](http://javatpoint.com)
- *Google Collab* : [TYProject.ipynb](https://colab.research.google.com/github/TYProject/ipyb)
- *Kaggle* : [Fake-Account](#)
- *Machine Algorithm Comparison* : [Comparison MAC](#)
- *Sci-kit learn* : [scikit-learn: machine learning in Python — scikit-learn 1.4.1 documentation](http://scikit-learn.org/stable/index.html)