|  |  |
| --- | --- |
| Project Title | **Instagram fake spammer genuine accounts** |
| Tools | ML, Python, Tableau Desktop, SQL, Excel |
| Domain | Data Analyst, Finance Analyst, Business Analyst |
| Project Difficulties level | intermediate |

Dataset : Dataset is available in the given link. You can download it at your convenience.

[Click](https://drive.google.com/drive/folders/1tgbfqoj2_rob-ZFAdDHXwl-LlE0Xf4T-?usp=sharing) [here](https://drive.google.com/drive/folders/1tgbfqoj2_rob-ZFAdDHXwl-LlE0Xf4T-?usp=sharing) [to](https://drive.google.com/drive/folders/1tgbfqoj2_rob-ZFAdDHXwl-LlE0Xf4T-?usp=sharing) [download](https://drive.google.com/drive/folders/1tgbfqoj2_rob-ZFAdDHXwl-LlE0Xf4T-?usp=sharing) [data](https://drive.google.com/drive/folders/1tgbfqoj2_rob-ZFAdDHXwl-LlE0Xf4T-?usp=sharing) [set](https://drive.google.com/drive/folders/1tgbfqoj2_rob-ZFAdDHXwl-LlE0Xf4T-?usp=sharing)

# About Dataset Context

Fakes and spammers are a major problem on all social media platforms, including Instagram.

This is the subject of my final-year project in which I set out to find ways of detecting them using machine learning.

In this dataset fake and spammer are interchangeable terms.

# Content

I have personally identified the spammer/fake accounts included in this dataset after carefully examining each instance and as such the dataset has high level of accuracy though there might be a couple of misidentified accounts in the spammers list as well.

The dataset has been collected using a crawler from 15-19, March 2019.

# Inspiration

This dataset could be further improved in quantity and quality measures, but how much accuracy can it achieve?

Possible ways of using the models to tackle the problem?

**Example: You can get the basic idea how you can create a project from here**

|  |
| --- |
| **Project: Instagram Account Classification – Fake vs. Genuine Accounts**  **Step 1: Data Preparation**  **Import Libraries and Load Dataset** python code  # Import libraries import pandas as pd import numpy as np import matplotlib.pyplot as plt import seaborn as sns from sklearn.model\_selection import train\_test\_split from sklearn.ensemble import RandomForestClassifier from sklearn.metrics import classification\_report, confusion\_matrix |

|  |
| --- |
| # Load the dataset data = pd.read\_csv('instagram\_data.csv') # replace with the path to your dataset data.head()  **Columns Overview:**   * **profile\_pic**: Binary (1 if the profile has a picture, 0 otherwise). * **nums/length username**: Number of characters or numbers in the username. * **fullname words**: Number of words in the full name. * **nums/length fullname**: Character length and number count in the full name. * **name == username**: Binary (1 if the username and full name are identical, 0 otherwise). * **description length**: Character length of the bio/description. * **external URL**: Binary (1 if an external URL is present, 0 otherwise). * **private**: Binary (1 if the profile is private, 0 otherwise). * **#posts**: Number of posts. * **#followers**: Number of followers. * **#follows**: Number of accounts followed. * **fake**: Target label (1 if fake, 0 if genuine).   **Step 2: Exploratory Data Analysis (EDA)**  **Initial Data Check** python code  # Check for missing values data.isnull().sum() |

|  |
| --- |
| # Basic statistics of each column data.describe()  **Distribution of Target Variable** python code  # Plotting the distribution of fake and genuine accounts sns.countplot(x='fake', data=data) plt.title("Distribution of Fake vs Genuine Accounts") plt.show()  **Correlation Analysis**  Check how features are correlated with each other and the target label:  python code  # Correlation matrix correlation = data.corr() plt.figure(figsize=(10, 8)) sns.heatmap(correlation, annot=True, cmap='coolwarm') plt.title("Feature Correlation Matrix") plt.show()  **Visualization of Key Features** |

|  |
| --- |
| **Profile Picture (Fake vs. Genuine)**:  python code  sns.barplot(x='fake', y='profile\_pic', data=data) plt.title("Profile Picture Presence in Fake vs Genuine Accounts") plt.show()  ●  **Followers and Following Counts**:  python code  sns.boxplot(x='fake', y='#followers', data=data) plt.title("Followers Count in Fake vs Genuine Accounts") plt.show()  sns.boxplot(x='fake', y='#follows', data=data) plt.title("Following Count in Fake vs Genuine Accounts") plt.show()  ●  **Posts Count**:  python code  sns.boxplot(x='fake', y='#posts', data=data) plt.title("Posts Count in Fake vs Genuine Accounts") plt.show() |

|  |
| --- |
| ●  **Step 3: Data Preprocessing**  **Feature Engineering**   * Convert categorical features into numeric formats, if any. * Scale or normalize features if necessary.   python code  # Example: Feature Scaling (Optional) from sklearn.preprocessing import StandardScaler  scaler = StandardScaler() scaled\_features = scaler.fit\_transform(data.drop('fake', axis=1)) scaled\_data = pd.DataFrame(scaled\_features, columns=data.columns[:-1]) scaled\_data['fake'] = data['fake']  **Step 4: Model Building**  We'll use a Random Forest classifier for this binary classification task due to its effectiveness in handling imbalanced data and feature importance analysis.  python code  # Split data into training and test sets |

|  |
| --- |
| X = scaled\_data.drop('fake', axis=1) y = scaled\_data['fake']  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)  # Build Random Forest Model model = RandomForestClassifier(n\_estimators=100, random\_state=42) model.fit(X\_train, y\_train)  # Feature Importance Plot importances = model.feature\_importances\_ indices = np.argsort(importances)[::-1] plt.figure(figsize=(10, 6)) plt.title("Feature Importances") sns.barplot(y=X.columns[indices], x=importances[indices], palette='viridis') plt.show()  **Step 5: Model Evaluation**  **Predictions and Metrics** python code  # Make predictions y\_pred = model.predict(X\_test) |

|  |
| --- |
| # Evaluate the model print("Classification Report:\n", classification\_report(y\_test, y\_pred))  print("Confusion Matrix:\n", confusion\_matrix(y\_test, y\_pred))  **Visualize Confusion Matrix** python code from sklearn.metrics import ConfusionMatrixDisplay  ConfusionMatrixDisplay.from\_predictions(y\_test, y\_pred, display\_labels=['Genuine', 'Fake'], cmap='Blues') plt.title("Confusion Matrix") plt.show()  **Step 6: Interpretation and Insights**   * **Feature Importance**: Analyze the top features contributing to the classification.   ○ For example, if **profile\_pic** and **#followers** are among the most important features, this can indicate that fake accounts often lack profile pictures and tend to have suspiciously high or low follower counts.   * **Model Performance**: Evaluate the model accuracy, precision, recall, and F1-score from the classification report to understand how well it distinguishes between fake and genuine accounts.   **Step 7: Future Improvements** |
| To enhance the model, consider:   * Using more complex models like Gradient Boosting or XGBoost. * Tuning hyperparameters with GridSearchCV. * Applying techniques to handle imbalanced classes if fake accounts are less frequent. |

**Example: You can get the basic idea how you can create a project from here**

**Sample Project code and output**

|  |
| --- |
| *# This Python 3 environment comes with many helpful analytics libraries installed*  *# It is defined by the kaggle/python Docker image:*  *https://github.com/kaggle/docker-python*  *# For example, here's several helpful packages to load*  import numpy as np *# linear algebra* import pandas as pd *# data processing, CSV file I/O (e.g. pd.read\_csv)*  *# Input data files are available in the read-only "../input/" directory* |

|  |
| --- |
| *# For example, running this (by clicking run or pressing Shift+Enter) will list all files under the input directory*  import os for dirname, \_, filenames **in** os.walk('/kaggle/input'):  for filename **in** filenames:  print(os.path.join(dirname, filename))  *# You can write up to 20GB to the current directory*  *(/kaggle/working/) that gets preserved as output when you create a version using "Save & Run All"*  *# You can also write temporary files to /kaggle/temp/, but they won't be saved outside of the current session*  /kaggle/input/instagram-fake-spammer-genuine-accounts/train.csv  /kaggle/input/instagram-fake-spammer-genuine-accounts/test.csv  In [2]: df\_train = pd.read\_csv('/kaggle/input/instagram-fake-spammer-genuine-accou nts/train.csv') |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| In [3]:  df\_train.head()  Out[3]:   |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | |  | pr  ofil  e  pic | nums/ length usern ame | fulln ame wor ds | nums/ length fullna  me | name==u sername | descr iption lengt h | exte rnal  UR  L | pri vat e | #p  ost s | #follo wers | #foll  ows | fa k e | | 0 | 1 | 0.27 | 0 | 0.0 | 0 | 53 | 0 | 0 | 32 | 1000 | 955 | 0 | | 1 | 1 | 0.00 | 2 | 0.0 | 0 | 44 | 0 | 0 | 28  6 | 2740 | 533 | 0 | | 2 | 1 | 0.10 | 2 | 0.0 | 0 | 0 | 0 | 1 | 13 | 159 | 98 | 0 | | 3 | 1 | 0.00 | 1 | 0.0 | 0 | 82 | 0 | 0 | 67  9 | 414 | 651 | 0 | |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | 4 | 1 | 0.00 | 2 | 0.0 | 0 | 0 | 0 | 1 | 6 | 151 | 126 | 0 |   In [4]:  df\_train.shape  Out[4]:  (576, 12)  In [5]: df\_train.info()  <class 'pandas.core.frame.DataFrame'>  RangeIndex: 576 entries, 0 to 575 Data columns (total 12 columns):  # Column Non-Null Count Dtype  --- ------ -------------- -----   1. profile pic 576 non-null int64 2. nums/length username 576 non-null float64 3. fullname words 576 non-null int64 4. nums/length fullname 576 non-null float64 5. name==username 576 non-null int64 6. description length 576 non-null int64 |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 6 external URL 576 non-null | | | | | | | | | | int64 | |  | |
| 7 private 576 non-null | | | | | | | | | | int64 | |  | |
| 8 #posts 576 non-null | | | | | | | | | | int64 | |  | |
| 9 #followers 576 non-null | | | | | | | | | | int64 | |  | |
| 10 #follows 576 non-null | | | | | | | | | | int64 | |  | |
| 11 fake 576 non-null | | | | | | | | | | int64 | |  | |
| dtypes: float64(2), int64(10) memory usage: 54.1 KB | | | | | | | | | |  | | In [6]: | |
| df\_train.describe() | | | | | | | | | |  | | Out[6]: | |
|  |  | profi le pic | num  s/le ngth user nam e | fulln ame wor ds | num  s/le ngth  fulln ame | name= =usern ame | des  cript ion leng th | exte rnal  UR  L | priv ate | #pos  ts | #follo wers | #foll  ows | fake |
| c o | 576.  000 | 576.  000 | 576.  000 | 576.  000 | 576.00 | 576.  000 | 576.  000 | 576.  000 | 576.  000 | 5.76 0000 | 576.  000 | 576.  000 |
|  | | | | | | | | | | | | |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | | | | | | | | | | | | |
|  | u n  t | 000 | 000 | 000 | 000 | 0000 | 000 | 000 | 000 | 000 | e+02 | 000 | 000 |
| m e a n | 0.70 138  9 | 0.16 383  7 | 1.46 006  9 | 0.03 609  4 | 0.0347  22 | 22.6 232  64 | 0.11 631  9 | 0.38 194  4 | 107.  489 583 | 8.53 0724 e+04 | 508.  381 944 | 0.50 000  0 |
| st  d | 0.45 804  7 | 0.21 409  6 | 1.05 260  1 | 0.12 512  1 | 0.1832  34 | 37.7 029  87 | 0.32 088  6 | 0.48 628  5 | 402.  034 431 | 9.10 1485 e+05 | 917.  981 239 | 0.50 043  5 |
| m  in | 0.00 000  0 | 0.00 000  0 | 0.00 000  0 | 0.00 000  0 | 0.0000  00 | 0.00 000  0 | 0.00 000  0 | 0.00 000  0 | 0.00 000  0 | 0.00 0000 e+00 | 0.00 000  0 | 0.00 000  0 |
| 2 5  % | 0.00  000  0 | 0.00  000  0 | 1.00  000  0 | 0.00  000  0 | 0.0000  00 | 0.00  000  0 | 0.00  000  0 | 0.00  000  0 | 0.00  000  0 | 3.90 0000 e+01 | 57.5  000  00 | 0.00  000  0 |
| 5  0 | 1.00  000 | 0.00  000 | 1.00  000 | 0.00  000 | 0.0000 | 0.00  000 | 0.00  000 | 0.00  000 | 9.00  000 | 1.50 5000 | 229.  500 | 0.50  000 |
|  | | | | | | | | | | | | |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | | | | | | | | | | | | |
|  | % | 0 | 0 | 0 | 0 | 00 | 0 | 0 | 0 | 0 | e+02 | 000 | 0 |
| 7 5  % | 1.00  000  0 | 0.31  000  0 | 2.00  000  0 | 0.00  000  0 | 0.0000  00 | 34.0  000  00 | 0.00  000  0 | 1.00  000  0 | 81.5  000  00 | 7.16 0000 e+02 | 589.  500 000 | 1.00  000  0 |
| m a x | 1.00 000  0 | 0.92 000  0 | 12.0 000  00 | 1.00 000  0 | 1.0000  00 | 150.  000 000 | 1.00 000  0 | 1.00 000  0 | 738  9.00  000  0 | 1.53 3854 e+07 | 750  0.00  000  0 | 1.00 000  0 |
|  | | | | | |  |  |  |  |  | In [7]: | |
| df\_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 | | | | | | |  |  |  |  |  | Out[7]: | |

#posts

0

#followers

0

#follows

0

fake

0

dtype:

int64

In

[8]:

df\_train[

'fake'

]

.

value\_counts()

Out[8]:

fake

0

288

1

288

Name:

count,

dtype:

int64

In

[9]:

df\_train

.

nunique()

Out[9]:

profile

pic

2

nums/length

username

54

fullname

words

9

nums/length

fullname

25

|  |  |
| --- | --- |
| name==username | 2 |
| description length | 104 |
| external URL | 2 |
| private | 2 |
| #posts | 193 |
| #followers | 372 |
| #follows | 400 |
| fake dtype: int64 | 2 |

In [10]:

df\_train.corr()

Out[10]:

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | pro  file pic | num  s/len gth user nam e | full  na me  wor ds | num  s/len gth  fulln ame | name= =usern ame | des crip tion len gth | ext ern al  UR  L | priv ate | #po  sts | #fol low ers | #fol low s | fak e |
| profile | 1.0 000 | -0.3  640 | 0.2 132 | -0.1  317 | -0.124 | 0.3 678 | 0.2 367 | 0.1 147 | 0.1 695 | 0.0 611 | 0.1 948 | -0.  637 |

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| pic | 00 | 87 | 95 | 56 | 903 | 92 | 29 | 32 | 70 | 37 | 33 | 315 |
| nums/l ength userna  me | -0.  364 087 | 1.00  000  0 | -0.  225 472 | 0.40  856  7 | 0.0568  90 | -0.3  211  70 | -0.  237 125 | -0.  063 713 | -0.  157 442 | -0.  062 785 | -0.  172 413 | 0.5  876  87 |
| fullnam e words | 0.2  132  95 | -0.2  254  72 | 1.0  000  00 | -0.0  943  48 | -0.082  969 | 0.2  725  22 | 0.1  965  62 | -0.  089 070 | 0.0  733  50 | 0.0  332  25 | 0.0  948  55 | -0.  298 793 |
| nums/l ength fullnam e | -0.  131 756 | 0.40  856  7 | -0.  094 348 | 1.00  000  0 | 0.2911  49 | -0.1  175  21 | -0.  088 724 | -0.  030 030 | -0.  057 716 | -0.  027 035 | -0.  067 971 | 0.2  467  82 |
| name= =usern ame | -0.  124 903 | 0.05  689  0 | -0.  082 969 | 0.29  114  9 | 1.0000  00 | -0.0  648  14 | -0.  039 232 | 0.0  460  84 | -0.  049 808 | -0.  017 761 | -0.  009 529 | 0.1  706  95 |
| descrip tion | 0.3 678 | -0.3  211 | 0.2 725 | -0.1  175 | -0.064  814 | 1.0 000 | 0.4 823 | -0.  110 | 0.1 448 | 0.0 059 | 0.2 265 | -0.  460 |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | length | 92 | 70 | 22 | 21 |  | 00 | 13 | 329 | 24 | 29 | 61 | 825 | | extern  al URL | 0.2  367  29 | -0.2  371  25 | 0.1  965  62 | -0.0  887  24 | -0.039  232 | 0.4  823  13 | 1.0  000  00 | -0.  162 612 | 0.1  650  08 | 0.0  271  89 | 0.1  425  19 | -0.  362 809 | | private | 0.1  147  32 | -0.0  637  13 | -0.  089 070 | -0.0  300  30 | 0.0460  84 | -0.1  103  29 | -0.  162 612 | 1.0  000  00 | -0.  087 495 | -0.  073 473 | -0.  057 542 | -0.  028 586 | | #posts | 0.1  695  70 | -0.1  574  42 | 0.0  733  50 | -0.0  577  16 | -0.049  808 | 0.1  448  24 | 0.1  650  08 | -0.  087 495 | 1.0  000  00 | 0.3  213  85 | 0.0  982  25 | -0.  245 355 | | #follow ers | 0.0  611  37 | -0.0  627  85 | 0.0  332  25 | -0.0  270  35 | -0.017  761 | 0.0  059  29 | 0.0  271  89 | -0.  073 473 | 0.3  213  85 | 1.0  000  00 | -0.  011  066 | -0.  093 689 | | #follow s | 0.1  948  33 | -0.1  724  13 | 0.0  948  55 | -0.0  679  71 | -0.009  529 | 0.2  265  61 | 0.1  425  19 | -0.  057 542 | 0.0  982  25 | -0.  011  066 | 1.0  000  00 | -0.  224 835 | |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | fake | -0.  637 315 | 0.58 768  7 | -0.  298 793 | 0.24 678  2 | 0.1706  95 | -0.4  608  25 | -0.  362 809 | -0.  028 586 | -0.  245 355 | -0.  093 689 | -0.  224 835 | 1.0  000  00 |   In [11]:  import seaborn as sns import matplotlib.pyplot as plt plt.figure(figsize=(10,10)) sns.heatmap(df\_train.corr(), annot=True, cmap='coolwarm') plt.show() |

In

[12]:

df\_train

.

hist(figsize

=

(

10

,

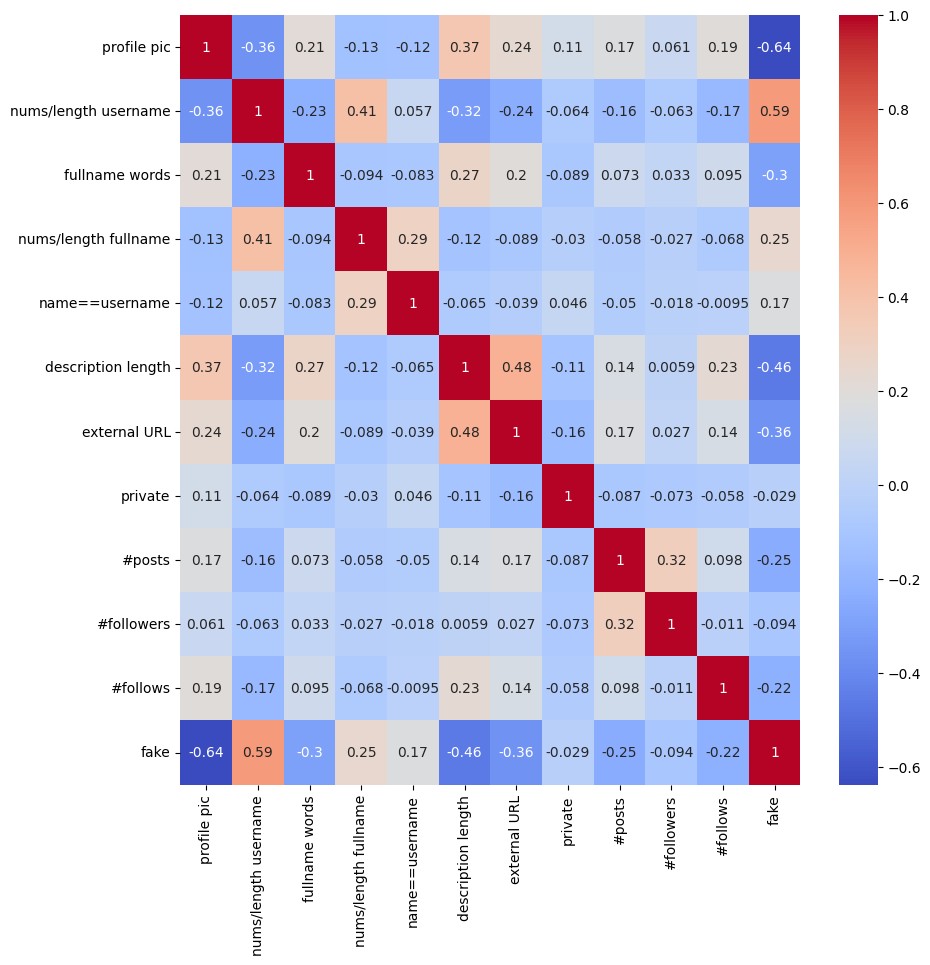
10

))

plt

.

show()



In

[13]:

plt

.

figure(figsize

=

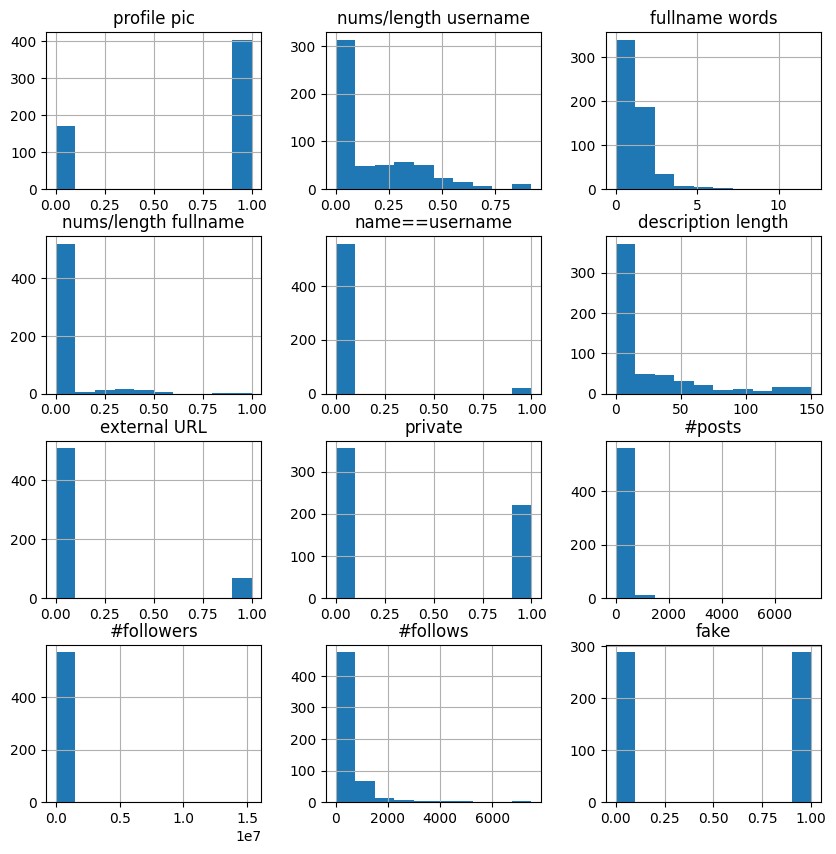
(

20

,

10

))



sns

.

countplot(x

=

'#followers'

,

hue

=

'fake'

,

data

=

df\_train)

plt

.

show()

In

[14]:

*#*

*create*

*bins*

*for*

*#followers*

*column*

bins

=

[

0

,

25

,

50

,

100

,

200

,

300

,

400

,

500

,

1000

,

5000

,

10000

]

*#*

*cut*

*the*

*#followers*

*column*

*into*

*the*

*bins*

df\_train[

'#followers\_bins'

]

=

pd

.

cut(df\_train[

'#followers'

]

,

bins

=

bins)

*#*

*plot*

*#followers*

*with*

*bins*

*and*

*show*

*fake*

*value*

*counts*

plt

.

figure(figsize

=

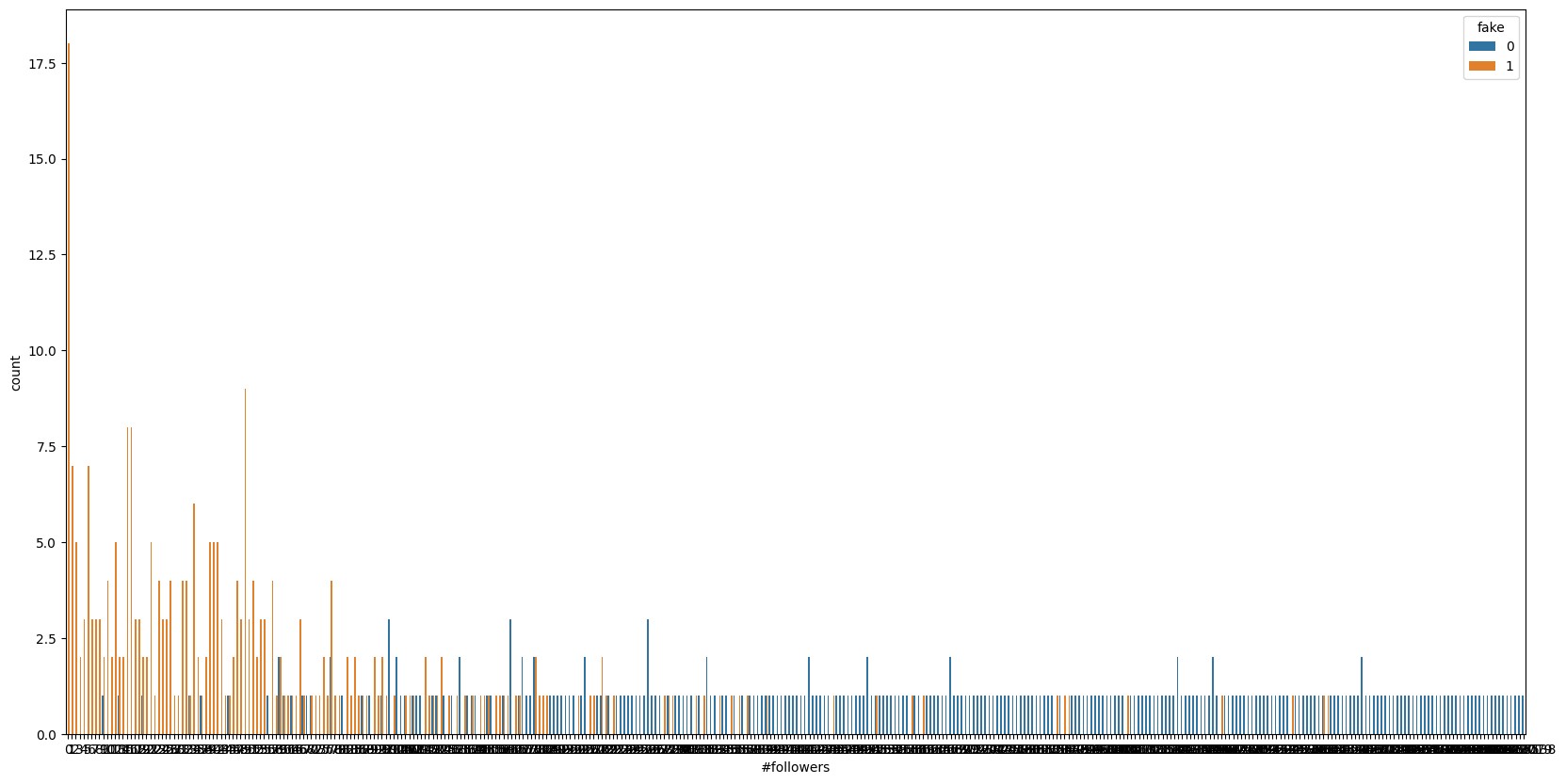
(

20

,

10

))



sns

.

countplot(x

=

'#followers\_bins'

,

hue

=

'fake'

,

data

=

df\_train)

plt

.

show()

In

[15]:

*#*

*plot*

*nums/length*

*username*

*and*

*show*

*fake*

*value*

*counts*

plt

.

figure(figsize

=

(

20

,

10

))

sns

.

countplot(x

=

'nums/length

username'

,

hue

=

'fake'

,

data

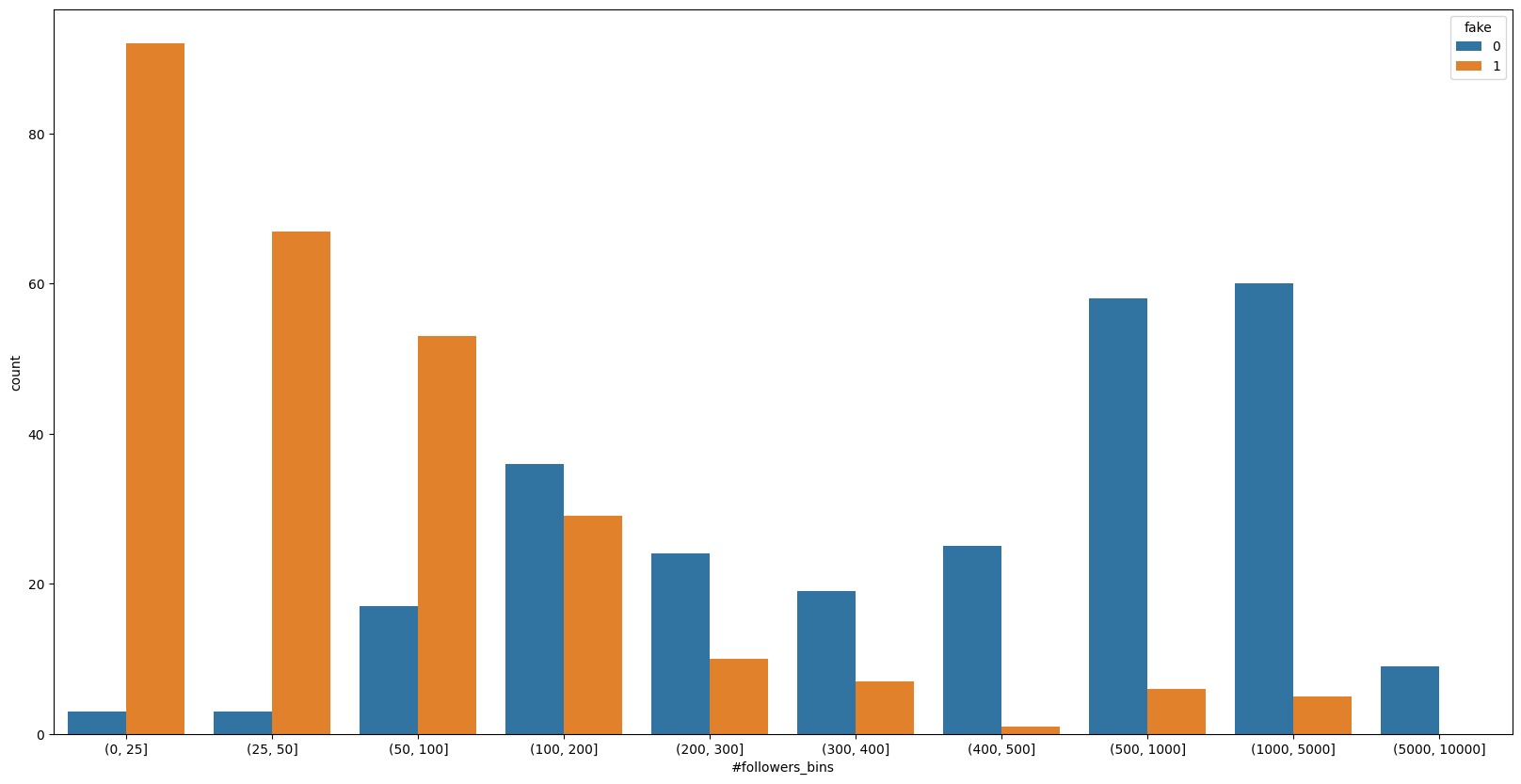
=

df\_train)

plt

.

show()



In

[16]:

*#*

*how*

*is*

*fake*

*distributed*

*amongst*

*private*

*accounts*

plt

.

figure(figsize

=

(

20

,

10

))

sns

.

countplot(x

=

'private'

,

hue

=

'fake'

,

data

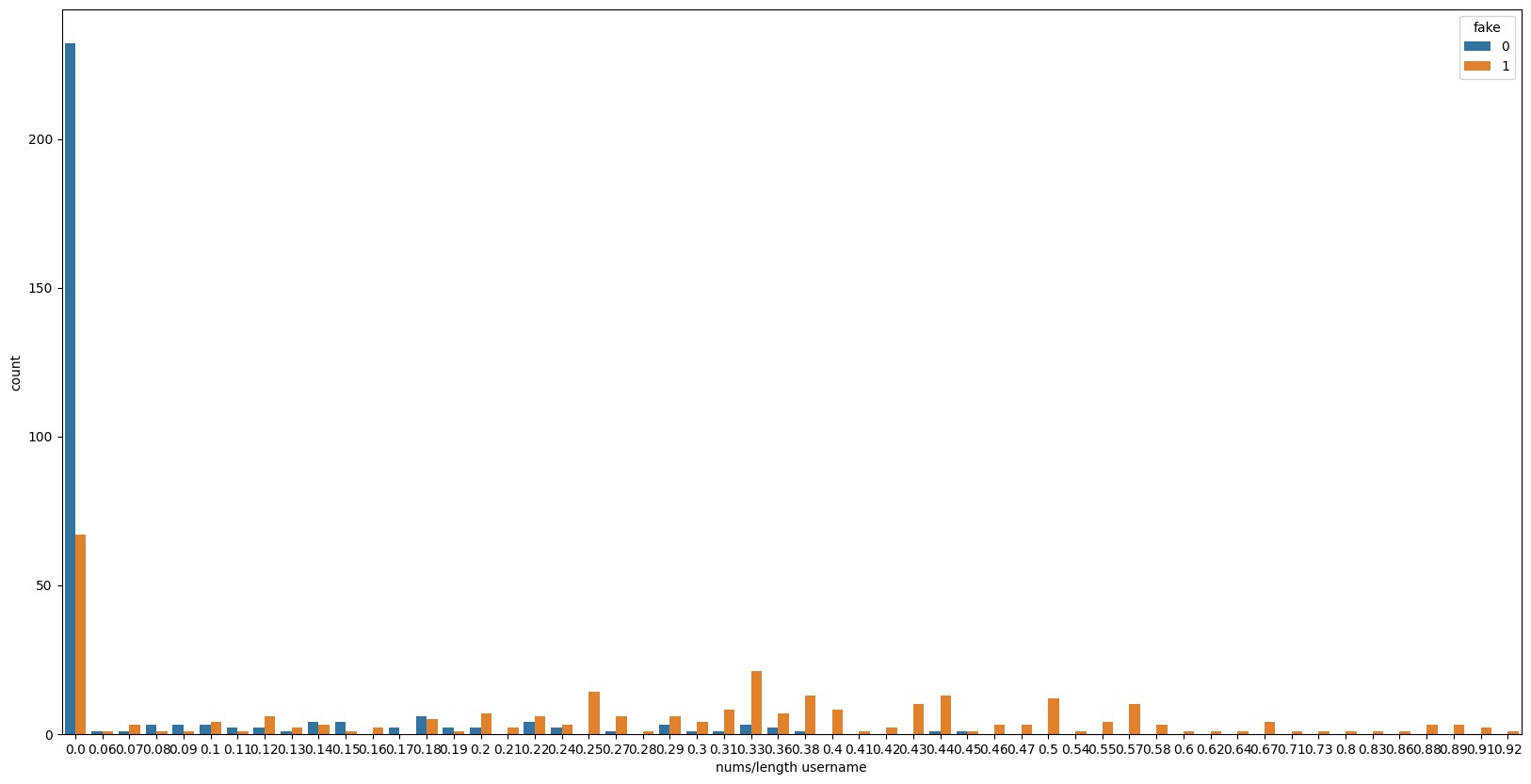
=

df\_train)

plt

.

show()



In

[17]:

from

pandas.plotting

import

scatter\_matrix

scatter\_matrix(df\_train,

figsize

=

(

22

,

22

)

,

diagonal

=

'kde'

,

c

=

df\_train[

'fake'

]

,

cmap

=

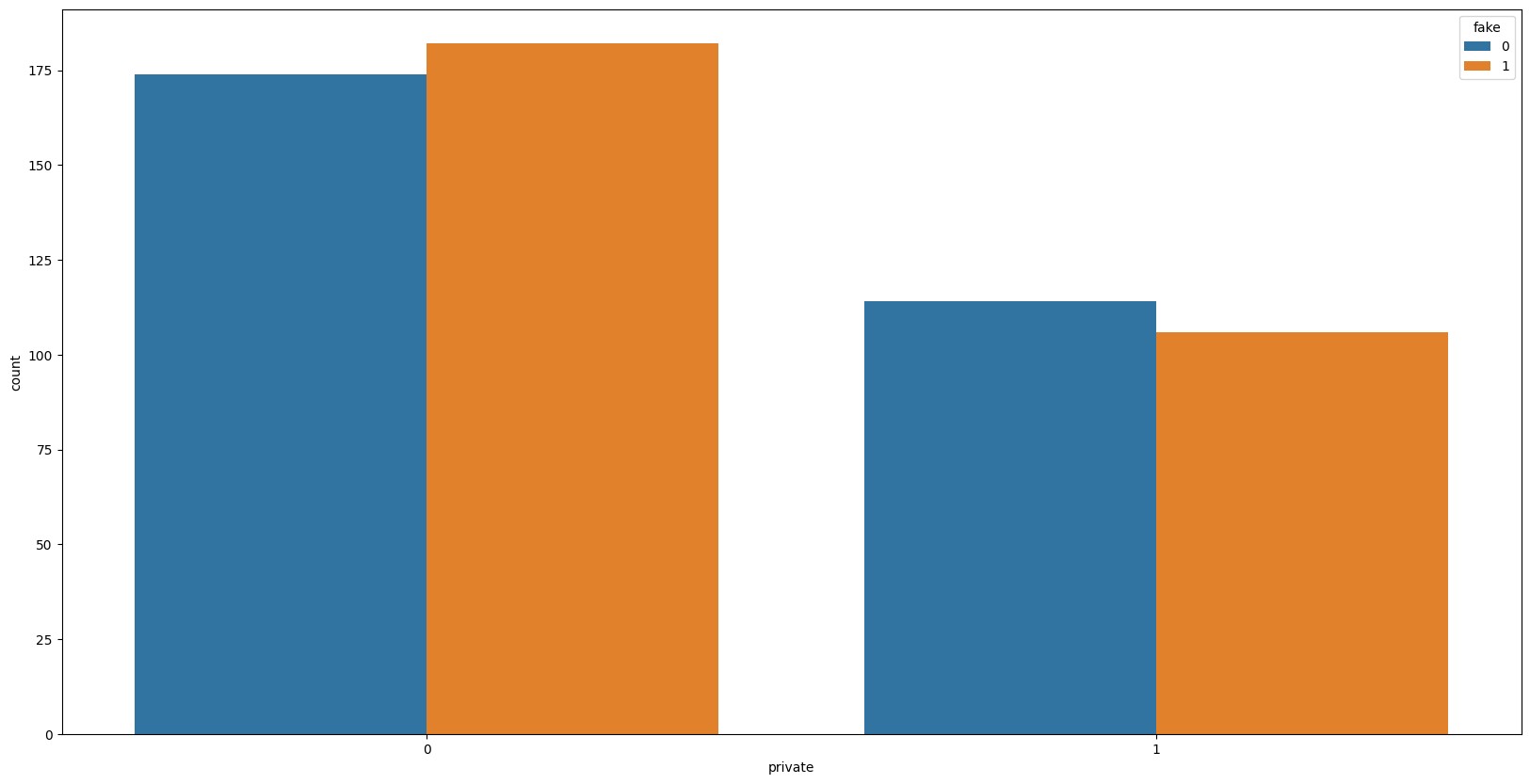
'coolwarm'

)

plt

.

show()



In

[18]:

df\_train

.

plot(kind

=

'box'

,

subplots

=

True

,

layout

=

(

4

,

4

)

,

figsize

=

(

10

,

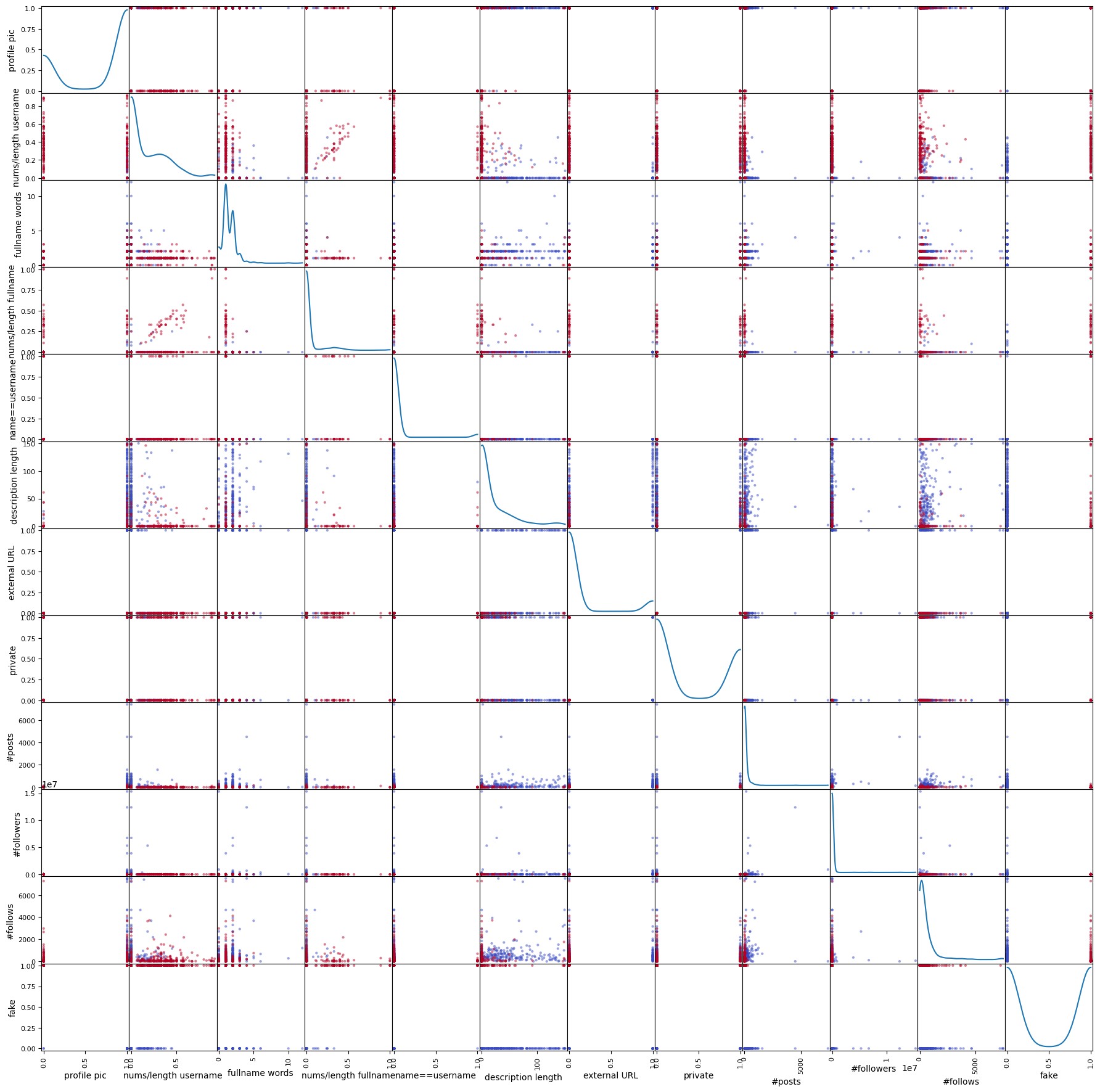
10

))

plt

.

show()



In

[

]:

In

[19]:

from

sklearn.tree

import

DecisionTreeClassifier

from

sklearn.model\_selection

import

train\_test\_split

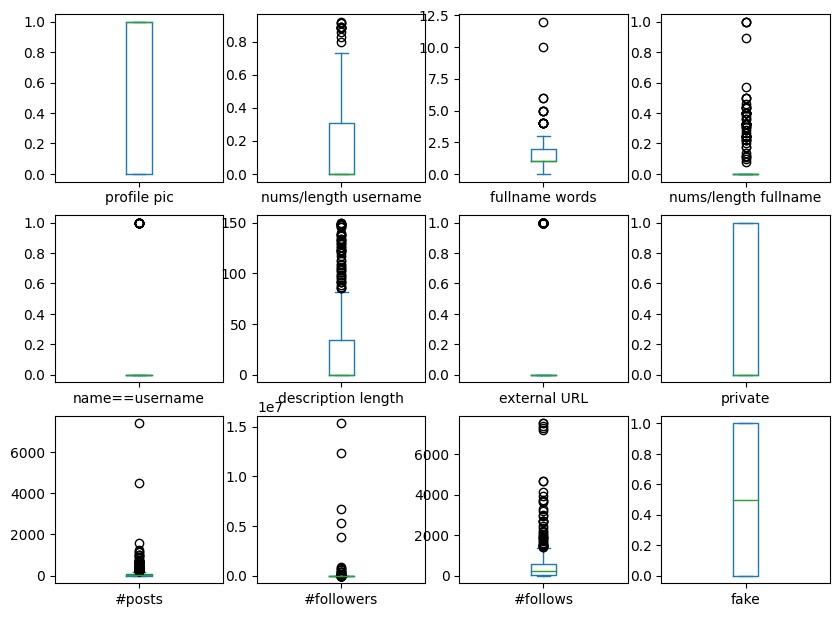
from

sklearn.metrics

import

accuracy\_score,

confusion\_matrix,



classification\_report

In

[20]:

X

=

df\_train

.

drop([

'#followers\_bins'

,

'fake'

]

,

axis

=

1

)

y

=

df\_train[

'fake'

]

X\_train,

X\_test,

y\_train,

y\_test

=

train\_test\_split(X,

y,

test\_size

=

0.2

,

random\_state

=

42

)

In

[21]:

model

=

DecisionTreeClassifier()

In

[22]:

print

X\_train

(

.

dtypes)

print

(

y\_train

.

dtypes)

profile

pic

int64

nums/length

username

float64

fullname

words

int64

nums/length

fullname

float64

name==username

int64

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| description length int64 external URL int64 private int64 #posts int64  #followers int64 #follows int64  dtype: object int64  In [23]:  model.fit(X\_train, y\_train)  Out[23]:   |  |  |  | | --- | --- | --- | | DecisionTreeClassifier | | | | DecisionTreeClassifier() |   In [24]:  y\_pred = model.predict(X\_test)  In [25]: accuracy\_score(y\_test, y\_pred) |

Out[25]:

weighted

avg

0.87

0.87

0.87

116

0.8706896551724138

In [26]:

confusion\_matrix(y\_test, y\_pred)

Out[26]:

array([[57, 6],

[ 9, 44]])

In [27]:

print(classification\_report(y\_test, y\_pred))

precision recall f1-score support

1. 0.86 0.90 0.88 63
2. 0.88 0.83 0.85 53

accuracy 0.87 116 macro avg 0.87 0.87 0.87 116

|  |
| --- |
| In [28]:  *# explain model with tree plot* from sklearn import tree plt.figure(figsize=(20,20)) tree.plot\_tree(model, filled=True, feature\_names=X.columns, class\_names=['real', 'fake'], rounded=True) plt.show() |

In

[29]:

model

.

feature\_importances\_

*#*

*plot*

*feature*

*importance*

plt

.

figure(figsize

=

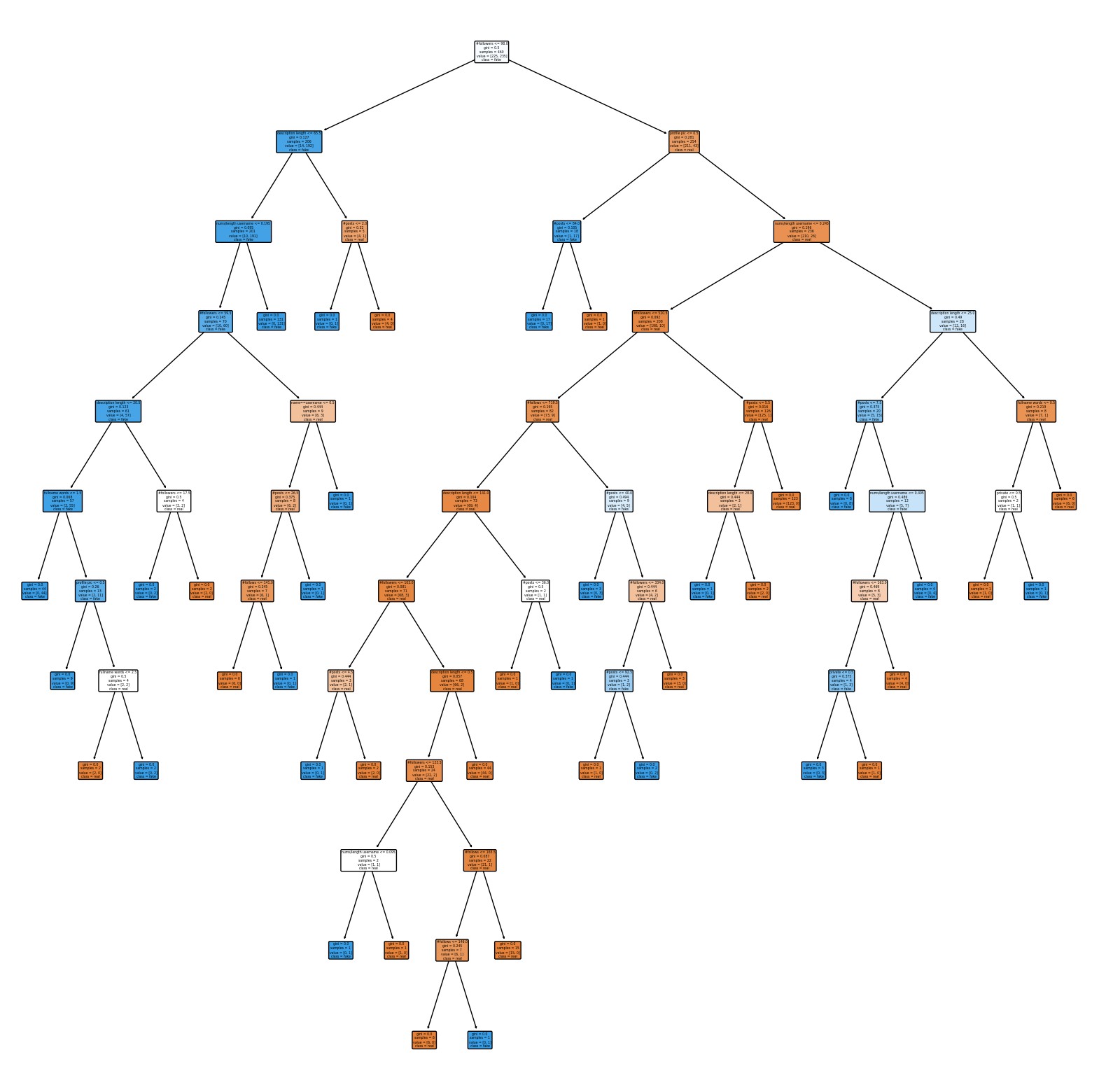
(

10

,

10

))



plt

.

barh(X

.

columns,

model

.

feature\_importances\_)

plt

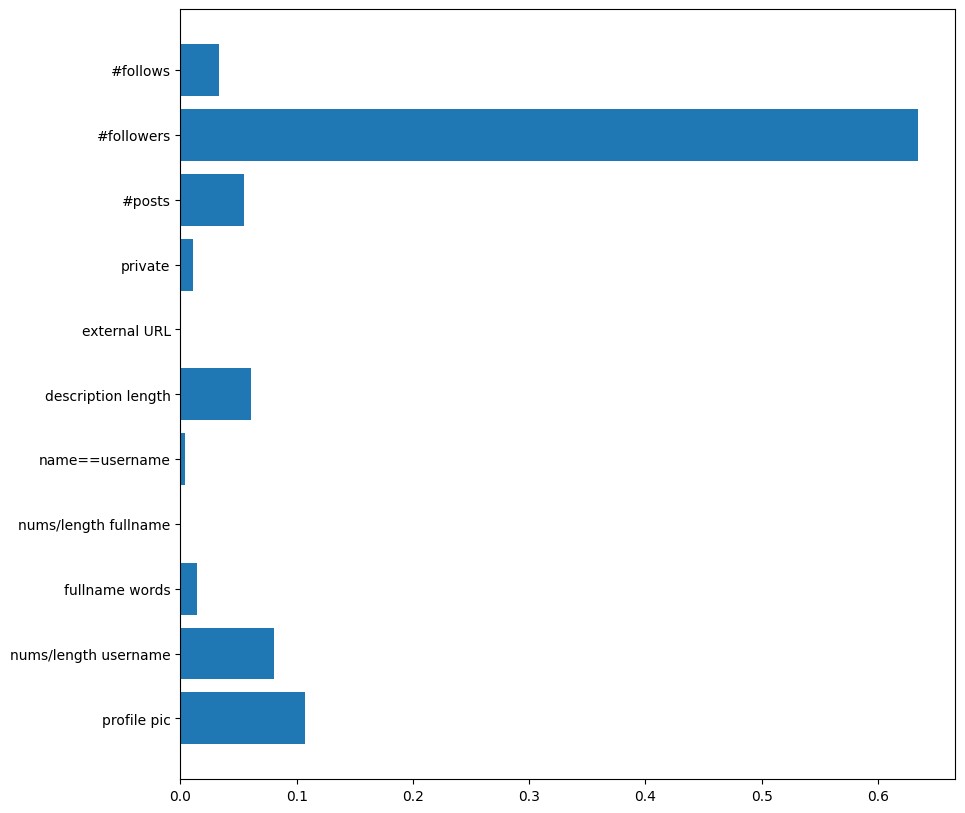
.

show()

In

[

]:



|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| In [30]:  df\_test = pd.read\_csv('/kaggle/input/instagram-fake-spammer-genuine-accou nts/test.csv') df\_test.head()  Out[30]:   |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | |  | pr  ofil  e  pic | nums/ length usern ame | fulln ame wor ds | nums/ length fullna  me | name==u sername | descr iption lengt h | exte rnal  UR  L | pri vat e | #p  ost s | #follo wers | #foll  ows | fa k e | | 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 | 31  9 | 328 | 668 | 0 | |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | 3 | 1 | 0.00 | 1 | 0.00 | 0 | 143 | 0 | 1 | 27  3 | 1489  0 | 736  9 | 0 | | 4 | 1 | 0.50 | 1 | 0.00 | 0 | 76 | 0 | 1 | 6 | 225 | 356 | 0 |   In [31]:  X\_test = df\_test.drop('fake', axis=1)  In [32]:  y\_pred = model.predict(X\_test)  In [33]:  y\_pred  Out[33]:  array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,  0, 1, 0,  0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,  0, 0, 0,  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, |

1, 1, 1,

1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,

1, 0, 1,

1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,

1, 1, 1,

0, 1, 1, 1, 1, 1, 1, 1, 1, 1])

In [34]:

accuracy\_score(df\_test['fake'], y\_pred)

Out[34]:

0.9416666666666667

In [35]:

print(confusion\_matrix(df\_test['fake'], y\_pred))

[[57 3]

[ 4 56]]

In [36]:

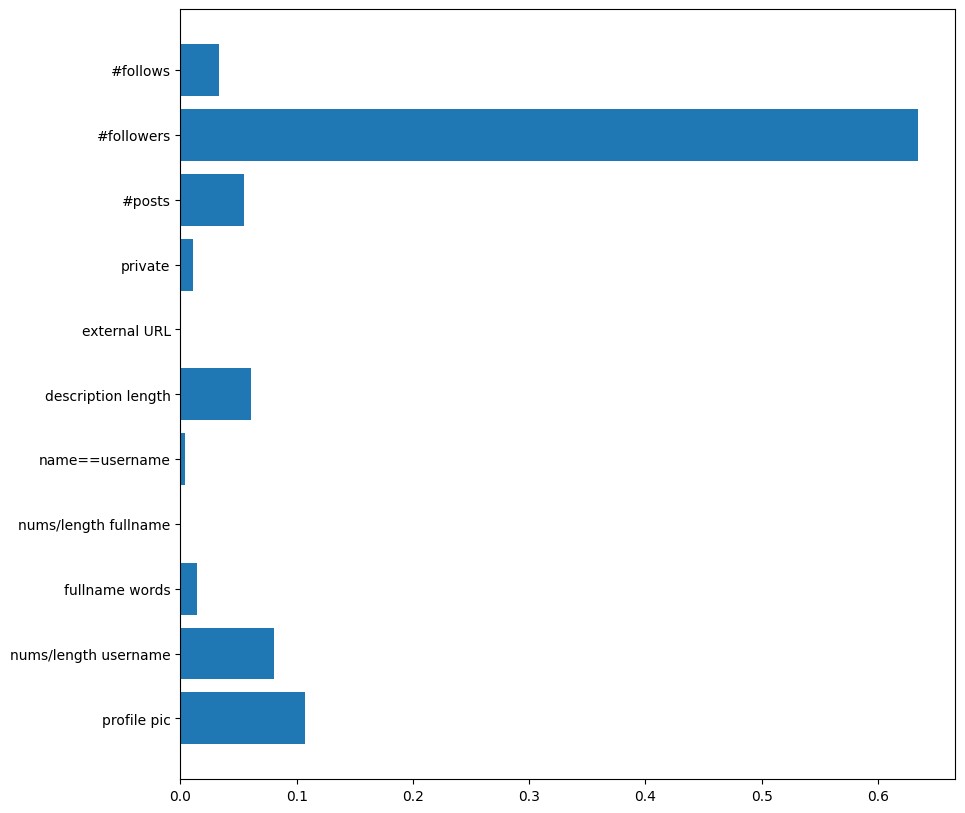
print(classification\_report(df\_test['fake'], y\_pred))

|  |
| --- |
| precision recall f1-score support   1. 0.93 0.95 0.94 60 2. 0.95 0.93 0.94 60   accuracy 0.94 120 macro avg 0.94 0.94 0.94 120 weighted avg 0.94 0.94 0.94 120  In [37]: plt.figure(figsize=(10,10)) plt.barh(X\_test.columns, model.feature\_importances\_) plt.show() |

In

[

]:



[Reference](https://github.com/deepd1534/instagram-fake-account-detection) [link](https://github.com/deepd1534/instagram-fake-account-detection) **Example: You can get the basic idea how you can create a project from here**

|  |
| --- |
| To build an SQL project that identifies Instagram accounts as fake, spam, or genuine based on various indicators, we will create a database schema, write SQL queries for EDA (exploratory data analysis), and build visualizations. This project can be structured to help a business analyst or data scientist with several years of experience to manage, query, and interpret insights from a large dataset.  **Project: Instagram Account Classification (SQL-Based)**  **Objective**  The main goal is to analyze Instagram account data and classify accounts as fake or genuine based on characteristics like username patterns, profile picture presence, follower/following counts, and other user activity metrics.  **Step 1: Database Schema Design**  We’ll create a table called instagram\_accounts with the following columns:  sql  code  CREATE TABLE instagram\_accounts ( account\_id INT PRIMARY KEY,  profile\_pic BOOLEAN, -- 1 if a profile pic exists, 0 otherwise  username\_length INT, -- Number of characters in the username  username\_nums INT, -- Number of digits in the username  fullname\_words INT, -- Number of words in |

|  |
| --- |
| the full name  fullname\_length INT, -- Total character length of the full name name\_equals\_username BOOLEAN, -- 1 if name matches  username, 0 otherwise  description\_length INT, -- Character count of bio description  external\_url BOOLEAN, -- 1 if an external URL exists, 0 otherwise  private BOOLEAN, -- 1 if the account is private, 0 if public  post\_count INT, -- Number of posts by the account  follower\_count INT, -- Number of followers following\_count INT, -- Number of accounts followed by the user  fake BOOLEAN -- 1 if account is fake,  0 otherwise );  **Step 2: Data Ingestion**  Assuming you have a dataset in CSV format, you can load it using SQL tools like PostgreSQL's COPY command or with SQL scripts that import data from structured  files. |

|  |
| --- |
| sql  code  COPY instagram\_accounts FROM '/path/to/instagram\_data.csv'  DELIMITER ',' CSV HEADER;  **Step 3: EDA with SQL Queries**   1. **Basic Data Overview**   sql  code  -- Check basic statistics  SELECT  COUNT(\*) AS total\_accounts,  SUM(CASE WHEN fake = 1 THEN 1 ELSE 0 END) AS  total\_fake\_accounts,  SUM(CASE WHEN fake = 0 THEN 1 ELSE 0 END) AS  total\_genuine\_accounts FROM instagram\_accounts;   1. **Analyzing Profile Picture Distribution**   sql  code  -- Percentage of fake vs. genuine accounts with a profile picture  SELECT profile\_pic, |

|  |
| --- |
| fake,  COUNT(\*) AS count,  ROUND((COUNT(\*) \* 100.0 / (SELECT COUNT(\*) FROM instagram\_accounts)), 2) AS percentage  FROM instagram\_accounts GROUP BY profile\_pic, fake;  **3. Followers and Following Analysis**  Accounts with very high follower-to-following ratios are often indicative of spam accounts. We can analyze this by calculating ratios and plotting distributions.  sql  code  -- Calculate follower-following ratio  SELECT account\_id, follower\_count, following\_count,  ROUND(CAST(follower\_count AS FLOAT) /  NULLIF(following\_count, 0), 2) AS follower\_following\_ratio, fake  FROM instagram\_accounts;  **4. Username and Full Name Patterns**  Analyzing usernames and full names, especially focusing on accounts where names |

|  |
| --- |
| match usernames or where usernames have unusual lengths, can help identify suspicious accounts.  sql  code  -- Average username length and presence of numbers in usernames for fake vs. genuine accounts  SELECT fake,  AVG(username\_length) AS avg\_username\_length,  AVG(username\_nums) AS avg\_username\_nums  FROM instagram\_accounts GROUP BY fake;  **5. Private vs. Public Accounts**  This query identifies how many private accounts are labeled as fake or genuine.  sql  code  SELECT private, fake,  COUNT(\*) AS count  FROM instagram\_accounts  GROUP BY private, fake; |

|  |
| --- |
| **Step 4: Visualization with SQL and BI Tools**  You can export the summarized data from SQL and visualize it using BI tools such as Tableau or Power BI. Here’s what each visualization could represent:   * **Profile Picture Analysis**: Use a bar chart to show the proportion of accounts with and without profile pictures across fake and genuine accounts. * **Follower/Following Ratio**: Visualize the follower-to-following ratio as a box plot, with separate plots for fake and genuine accounts to observe any significant differences. * **Username and Full Name Patterns**: Display the average username length and the average number of numbers in usernames for fake and genuine accounts using a grouped bar chart. * **Private vs Public Analysis**: Create a stacked bar chart to show the count of private and public accounts classified as fake or genuine.   **Step 5: Classification Insights**  Based on the SQL analysis, we can infer key indicators of fake accounts, such as:   * Fake accounts may frequently lack profile pictures or have very short usernames with unusual character patterns. * They often have low follower-to-following ratios. * Fake accounts might have a high likelihood of private profiles.   **Step 6: Next Steps for Model Building**  If further machine learning modeling is required:   1. **Data Export**: Export the processed SQL data to a CSV file for machine learning modeling. 2. **Feature Engineering**: Use the derived statistics (e.g., follower-following ratios) |

|  |
| --- |
| as additional features in a machine learning model.  3. **Modeling**: Build a classification model using Python and libraries like scikit-learn to predict fake accounts.  **Sample Code for Exporting Data**  sql  code  COPY (  SELECT account\_id, profile\_pic, username\_length,  username\_nums, fullname\_words, fullname\_length, name\_equals\_username,  description\_length, external\_url, private, post\_count, follower\_count, following\_count,  fake  FROM instagram\_accounts  ) TO '/path/to/output.csv' DELIMITER ',' CSV HEADER; |