Instagram fake spammer genuine accounts

```
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.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
from sklearn.model_selection import GridSearchCV
plt.style.use('ggplot')
%matplotlib inline
train_df = pd.read_csv('train.csv')
test_df = pd.read_csv('test.csv')
 print("Training dataset shape:", train_df.shape)
print("Test dataset shape:", test_df.shape)
print("\nTraining dataset columns:")
print(train_df.columns.tolist())
print("\nTraining dataset info:")
train_df.info()
```

Output:

```
Training dataset shape: (576, 12)
Test dataset shape: (120, 12)

Training dataset columns:
['profile pic', 'nums/length username', 'fullname words', 'nums/length fullname', 'name==username', 'description length', 'external URL', 'private', '#po
sts', '#followers', '#follows', 'fake']

Training dataset info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 576 entries, 0 to 575
Data columns (total 12 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   profile pic           576 non-null    int64
 1   nums/length username  576 non-null    float64
 2   fullname words        576 non-null    int64
 3   nums/length fullname  576 non-null    float64
 4   name==username        576 non-null    int64
 5   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
```

```
print("Missing values in training set:")
print(train_df.isnull().sum())
print("\nMissing values in test set:")
print(test_df.isnull().sum())
plt.figure(figsize=(8, 6))
sns.countplot(x='fake', data=train_df)
plt.title('Distribution of Target Variable (Fake vs Genuine)')
plt.xlabel('Account Type (0 = Genuine, 1 = Fake)')
plt.ylabel('Count')
plt.show()
fake_percentage = train_df['fake'].value_counts(normalize=True) * 100
print(f"Percentage of genuine accounts: {fake_percentage[0]:.2f}%")
print(f"Percentage of fake accounts: {fake_percentage[1]:.2f}%")
plt.figure(figsize=(12, 10))
correlation_matrix = train_df.corr()
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f')
plt.title('Correlation Matrix')
plt.show()
correlation_with_target = correlation_matrix['fake'].sort_values(ascending=False)
print("Correlation with target variable (fake):")
print(correlation_with_target)
```

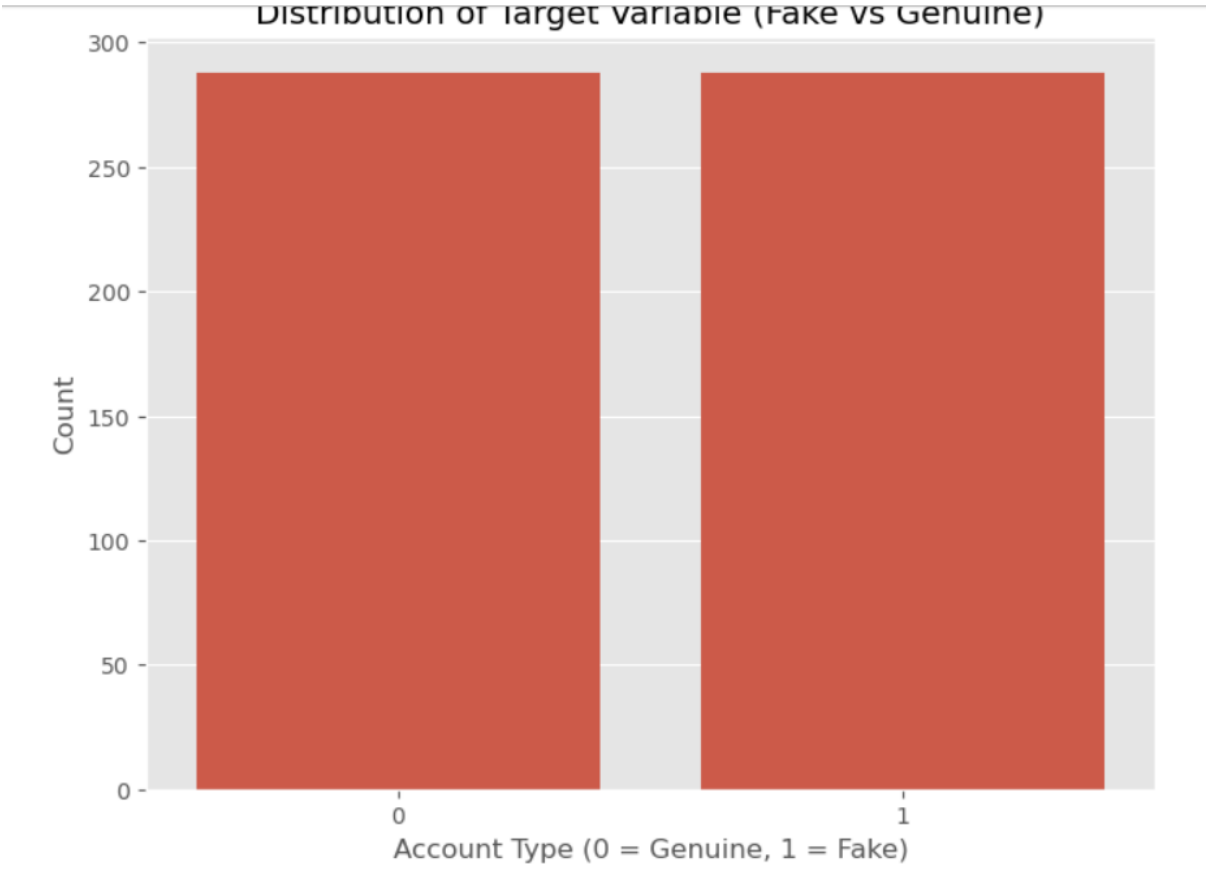
```
fig, axes = plt.subplots(2, 2, figsize=(15, 12))
profile_pic_comparison = train_df.groupby('fake')['profile
pic'].value_counts(normalize=True).unstack()
profile_pic_comparison.plot(kind='bar', ax=axes[0, 0])
axes[0, 0].set_title('Profile Picture Presence by Account Type')
axes[0, 0].set_xlabel('Account Type (0 = Genuine, 1 = Fake)')
axes[0, 0].set_ylabel('Percentage')
axes[0, 0].legend(['No Profile Pic', 'Has Profile Pic'])
genuine_followers = train_df[train_df['fake'] == 0]['#followers']
fake_followers = train_df[train_df['fake'] == 1]['#followers']

axes[0, 1].hist([np.log1p(genuine_followers), np.log1p(fake_followers)],
          bins=30, alpha=0.7, label=['Genuine', 'Fake'])
axes[0, 1].set_title('Distribution of Followers Count (Log Scale)')
axes[0, 1].set_xlabel('Log(Followers + 1)')
axes[0, 1].set_ylabel('Frequency')
axes[0, 1].legend()
genuine_follows = train_df[train_df['fake'] == 0]['#follows']
fake_follows = train_df[train_df['fake'] == 1]['#follows']

axes[1, 0].hist([np.log1p(genuine_follows), np.log1p(fake_follows)],
          bins=30, alpha=0.7, label=['Genuine', 'Fake'])
axes[1, 0].set_title('Distribution of Following Count (Log Scale)')
axes[1, 0].set_xlabel('Log(Following + 1)')
axes[1, 0].set_ylabel('Frequency')
axes[1, 0].legend()
genuine_posts = train_df[train_df['fake'] == 0]['#posts']
fake_posts = train_df[train_df['fake'] == 1]['#posts']

axes[1, 1].hist([np.log1p(genuine_posts), np.log1p(fake_posts)],
          bins=30, alpha=0.7, label=['Genuine', 'Fake'])
```

```
axes[1, 1].set_title('Distribution of Posts Count (Log Scale)')
axes[1, 1].set_xlabel('Log(Posts + 1)')
axes[1, 1].set_ylabel('Frequency')
axes[1, 1].legend()

plt.tight_layout()
plt.show()
fig, axes = plt.subplots(2, 2, figsize=(15, 12))
external_url_comparison = train_df.groupby('fake')['external
URL'].value_counts(normalize=True).unstack()
external_url_comparison.plot(kind='bar', ax=axes[0, 0])
axes[0, 0].set_title('External URL Presence by Account Type')
axes[0, 0].set_xlabel('Account Type (0 = Genuine, 1 = Fake)')
axes[0, 0].set_ylabel('Percentage')
axes[0, 0].legend(['No External URL', 'Has External URL'])
private_comparison =
train_df.groupby('fake')['private'].value_counts(normalize=True).unstack()
private_comparison.plot(kind='bar', ax=axes[0, 1])
axes[0, 1].set_title('Private Account Status by Account Type')
axes[0, 1].set_xlabel('Account Type (0 = Genuine, 1 = Fake)')
axes[0, 1].set_ylabel('Percentage')
axes[0, 1].legend(['Not Private', 'Private'])
genuine_desc = train_df[train_df['fake'] == 0]['description length']
fake_desc = train_df[train_df['fake'] == 1]['description length']

axes[1, 0].hist([genuine_desc, fake_desc], bins=30, alpha=0.7, label=['Genuine', 'Fake'])
axes[1, 0].set_title('Distribution of Description Length')
axes[1, 0].set_xlabel('Description Length')
axes[1, 0].set_ylabel('Frequency')
axes[1, 0].legend()
genuine_username = train_df[train_df['fake'] == 0]['nums/length username']
```

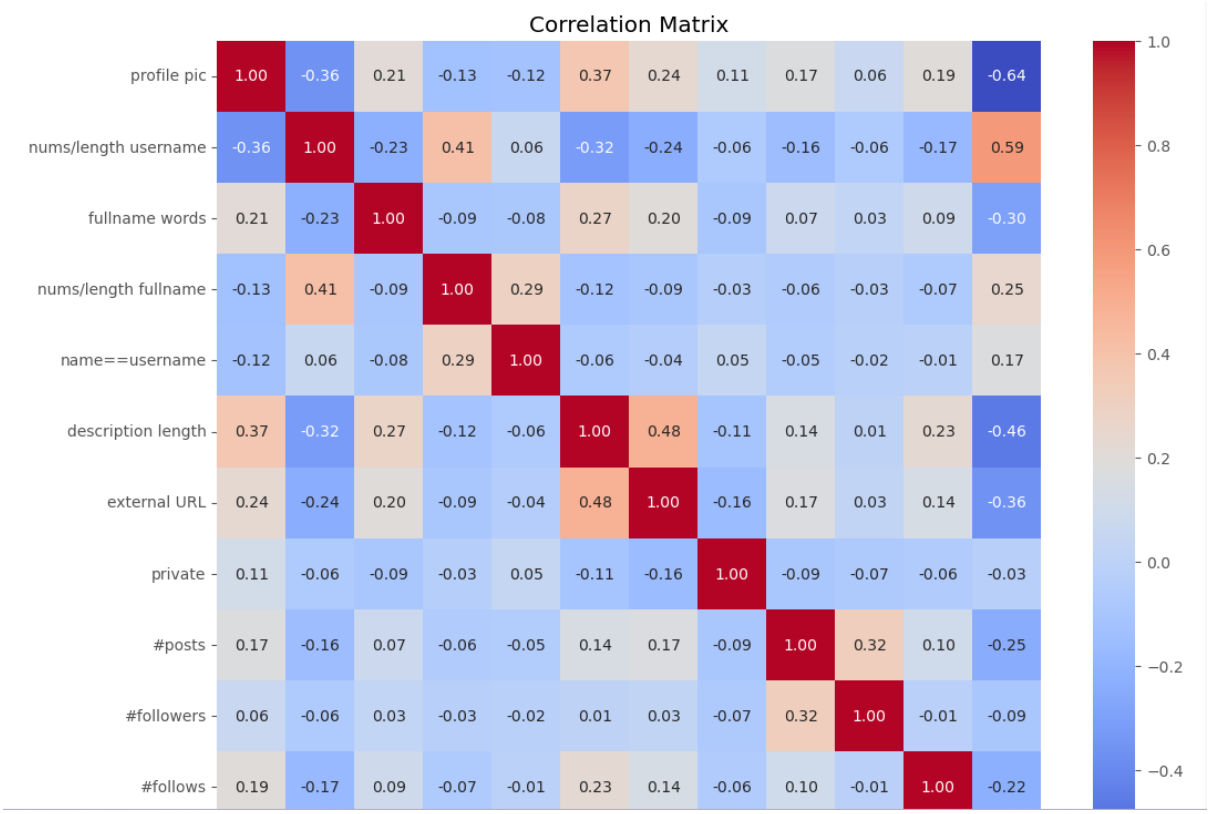
```
fake_username = train_df[train_df['fake'] == 1]['nums/length username']


axes[1, 1].hist([genuine_username, fake_username], bins=30, alpha=0.7, label=['Genuine',
'Fake'])

axes[1, 1].set_title('Distribution of Username Numeric Ratio')

axes[1, 1].set_xlabel('Username Numeric Ratio')

axes[1, 1].set_ylabel('Frequency')

axes[1, 1].legend()


plt.tight_layout()

plt.show()
```

Output:

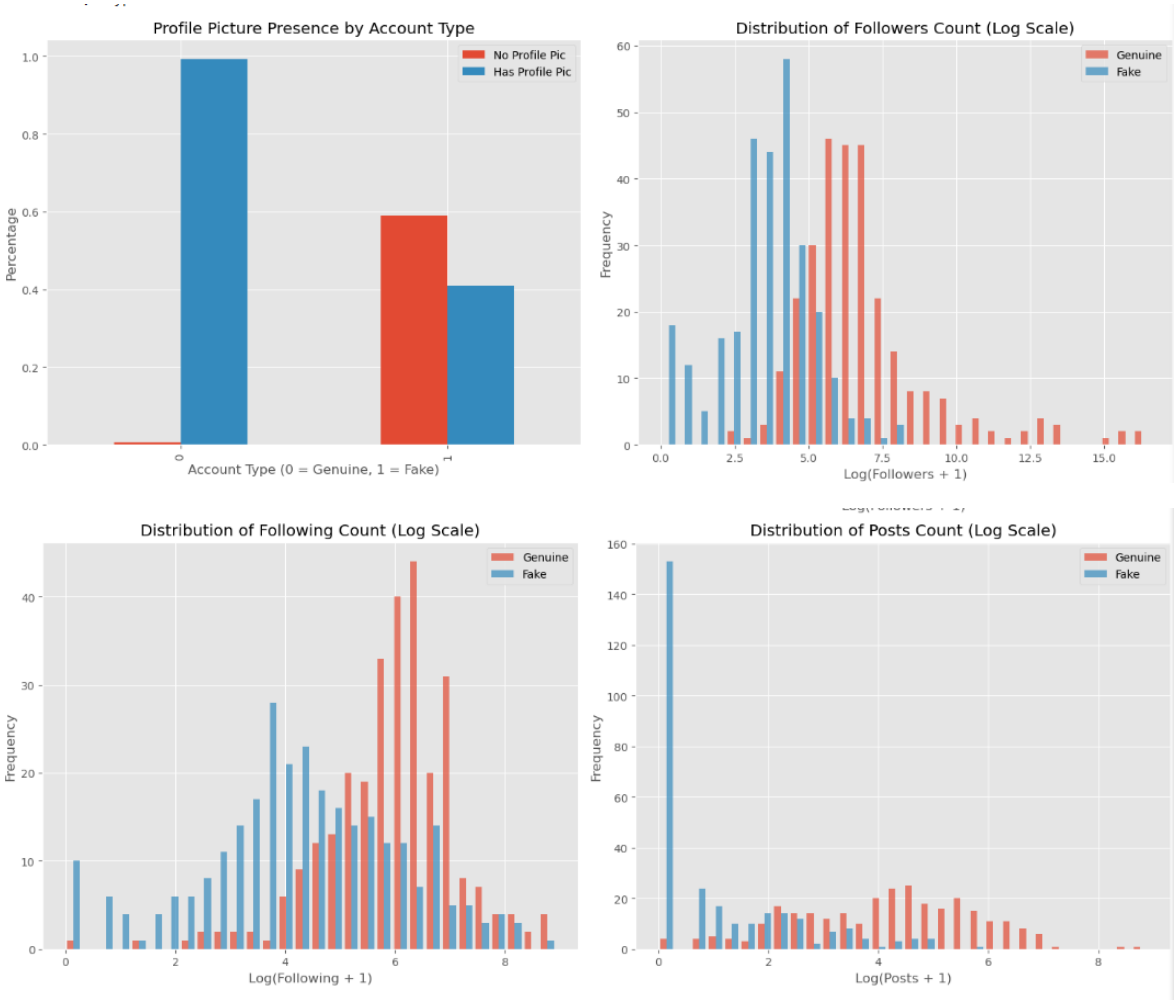
```
Missing values in training set:
profile pic             0
nums/length username    0
fullname words          0
nums/length fullname    0
name==username          0
description length      0
external URL            0
private                 0
#posts                  0
#followers              0
#follows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

Missing values in test set:
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
```

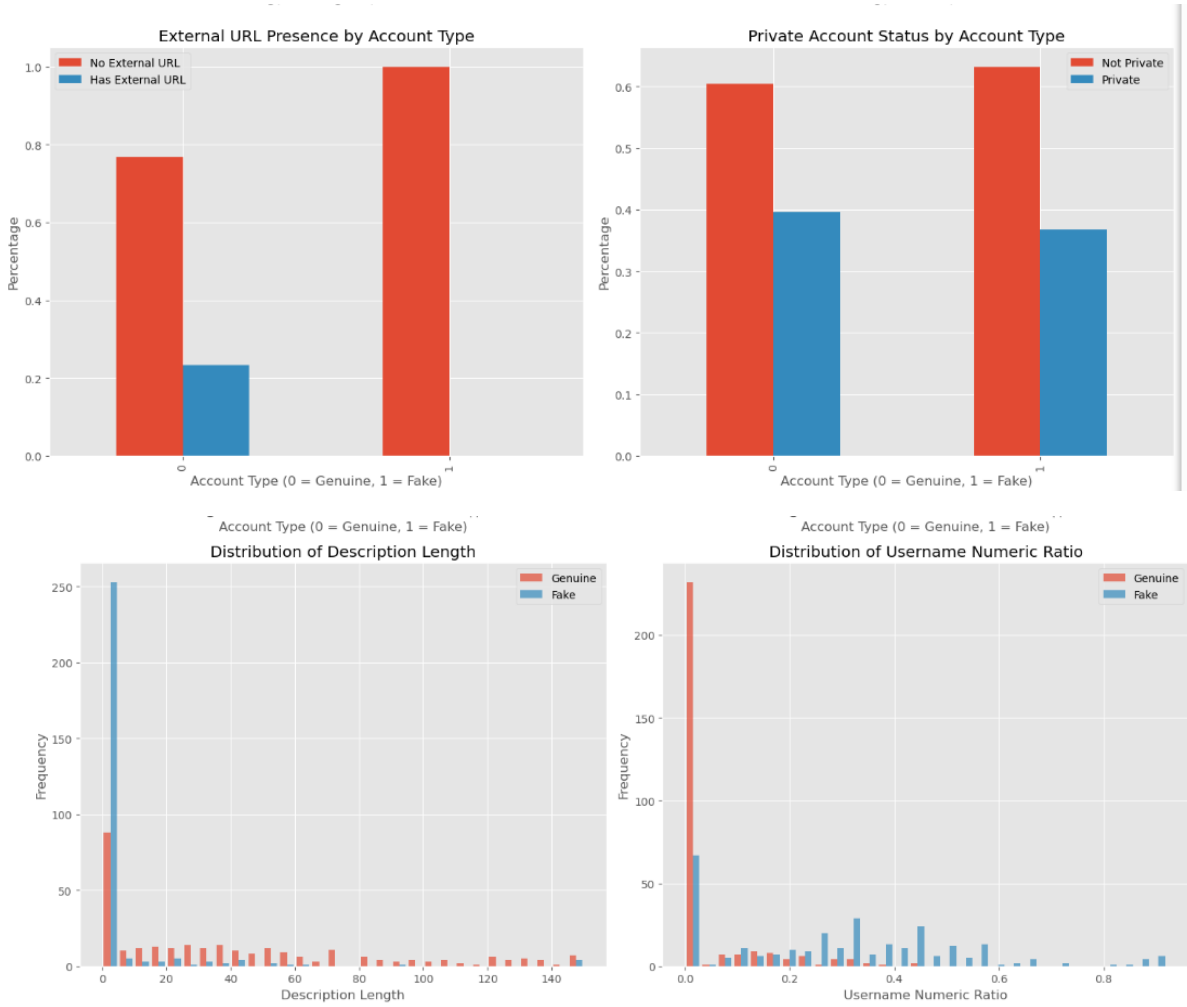



```
Percentage of genuine accounts: 50.00%
Percentage of fake accounts: 50.00%
```



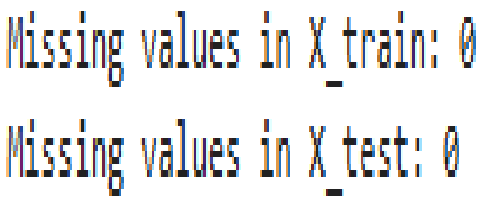
```
Correlation with target variable (fake):
fake                    1.000000
nums/length username    0.587687
nums/length fullname    0.246782
name==username          0.170695
private                -0.028586
#followers             -0.093689
#follows               -0.224835
#posts                 -0.245355
fullname words         -0.298793
external URL           -0.362809
description length     -0.460825
profile pic            -0.637315
Name: fake, dtype: float64
```






```
X_train = train_df.drop('fake', axis=1)
y_train = train_df['fake']
X_test = test_df.drop('fake', axis=1)
y_test = test_df['fake']
print("Missing values in X_train:", X_train.isnull().sum().sum())
print("Missing values in X_test:", X_test.isnull().sum().sum())
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
X_train_scaled = pd.DataFrame(X_train_scaled, columns=X_train.columns)
X_test_scaled = pd.DataFrame(X_test_scaled, columns=X_test.columns)
```

Output:

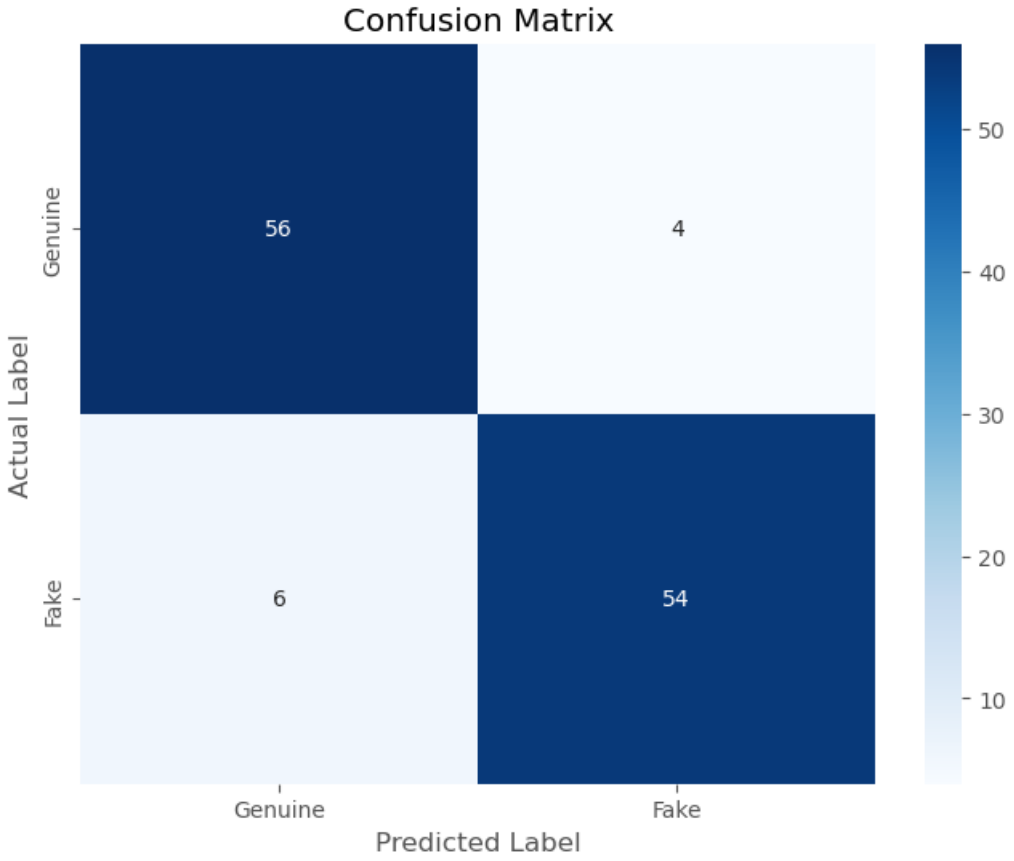


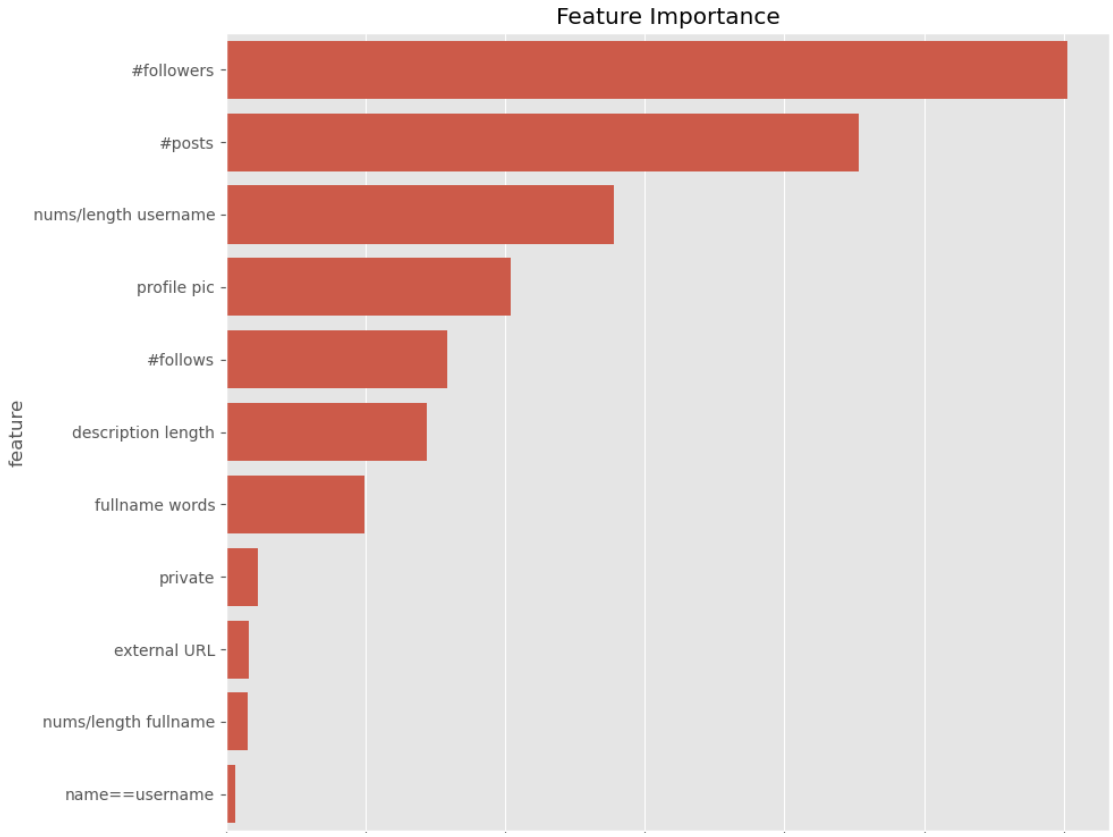

```
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_train_scaled, y_train)
y_pred = rf_model.predict(X_test_scaled)
y_pred_proba = rf_model.predict_proba(X_test_scaled)[:, 1]
```


```
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_train_scaled, y_train)
y_pred = rf_model.predict(X_test_scaled)
y_pred_proba = rf_model.predict_proba(X_test_scaled)[:, 1]
```


Output:

```
Model Accuracy: 0.9167

Classification Report:
              precision    recall  f1-score   support

           0       0.90      0.93      0.92        60
           1       0.93      0.90      0.92        60

    accuracy                           0.92       120
   macro avg       0.92      0.92      0.92       120
weighted avg       0.92      0.92      0.92       120
```


Confusion Matrix
Genuine
Fake
Actual Label
56
4
6
54
Genuine
Fake
Predicted Label
50
40
30
20
10


Feature Importance
feature
#followers
#posts
nums/length username
profile pic
#follows
description length
fullname words
private
external URL
nums/length fullname
name==username

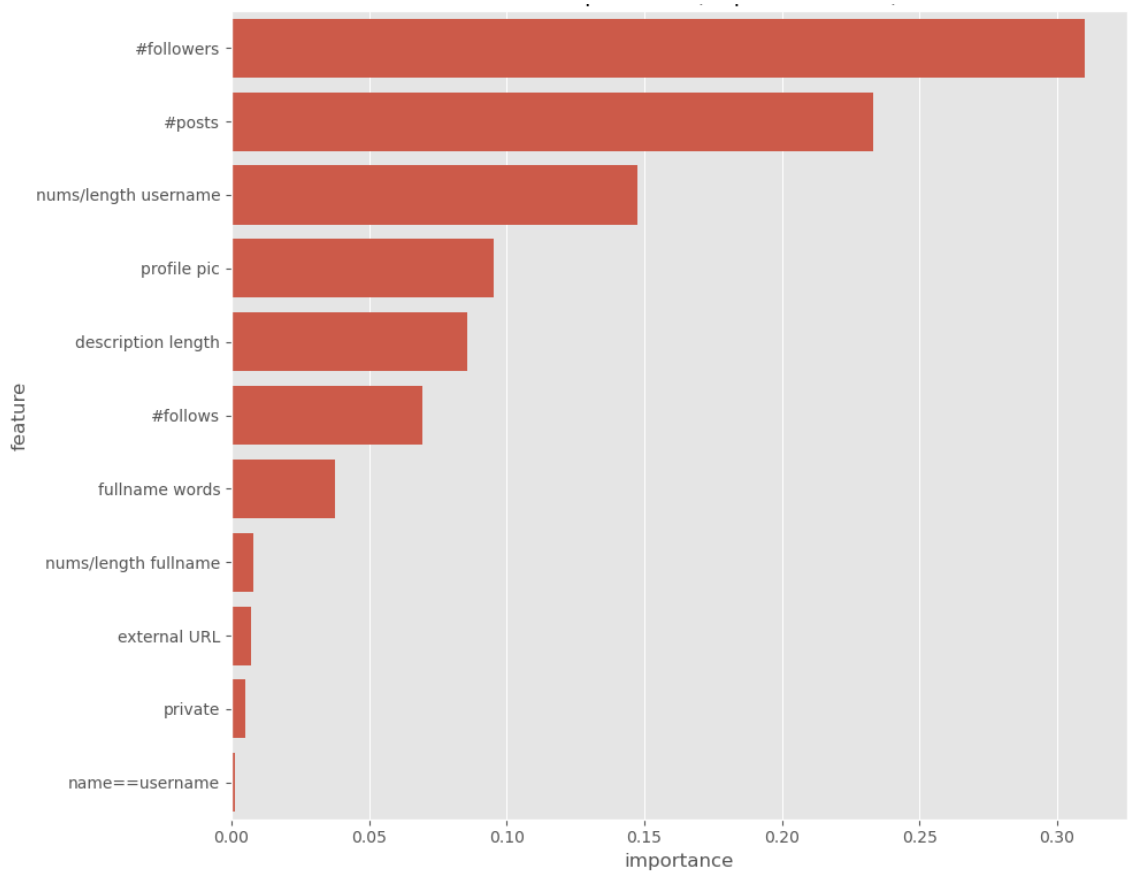
```
Top 10 Most Important Features:
                 feature  importance
9             #followers    0.301272
8                #posts    0.226466
1   nums/length username    0.138969
0            profile pic    0.102057
10             #follows    0.079188
5     description length    0.071859
2         fullname words    0.049695
7                private    0.011263
6           external URL    0.008201
3   nums/length fullname    0.007791
```

```
param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}
if len(X_train_scaled) > 5000:
    X_train_sample, _, y_train_sample, _ = train_test_split(
        X_train_scaled, y_train, train_size=0.5, random_state=42, stratify=y_train)
else:
    X_train_sample, y_train_sample = X_train_scaled, y_train
grid_search = GridSearchCV(
    RandomForestClassifier(random_state=42),
    param_grid,
    cv=3,
    scoring='accuracy',
    n_jobs=-1,
    verbose=1
)

grid_search.fit(X_train_sample, y_train_sample)
print("Best parameters:", grid_search.best_params_)
print("Best cross-validation score:", grid_search.best_score_)
best_rf_model = grid_search.best_estimator_
```

```
best_rf_model.fit(X_train_scaled, y_train)
y_pred_improved = best_rf_model.predict(X_test_scaled)
improved_accuracy = accuracy_score(y_test, y_pred_improved)
print(f"Improved Model Accuracy: {improved_accuracy:.4f}")

print("\nImproved Classification Report:")
print(classification_report(y_test, y_pred_improved))
improved_feature_importance = pd.DataFrame({
    'feature': X_train.columns,
    'importance': best_rf_model.feature_importances_
}).sort_values('importance', ascending=False)

plt.figure(figsize=(10, 8))
sns.barplot(x='importance', y='feature', data=improved_feature_importance)
plt.title('Feature Importance (Improved Model)')
plt.tight_layout()
plt.show()

print("Top 10 Most Important Features (Improved Model):")
print(improved_feature_importance.head(10))
```

Output:

```
Fitting 3 folds for each of 108 candidates, totalling 324 fits
Best parameters: {'max_depth': 10, 'min_samples_leaf': 1, 'min_samples_split': 10, 'n_estimators': 200}
Best cross-validation score: 0.9253472222222222
Improved Model Accuracy: 0.9083

Improved Classification Report:
              precision    recall  f1-score   support

           0       0.92      0.90      0.91        60
           1       0.90      0.92      0.91        60

    accuracy                           0.91       120
   macro avg       0.91      0.91      0.91       120
weighted avg       0.91      0.91      0.91       120
```




```
Top 10 Most Important Features (Improved Model):
                feature  importance
9             #followers    0.309957
8                 #posts    0.233095
1   nums/length username    0.147695
0            profile pic    0.095312
5     description length    0.085665
10              #follows    0.069275
2         fullname words    0.037481
3   nums/length fullname    0.007799
6           external URL    0.007191
7                private    0.005233
```

```
def predict_account_type(account_features):
    if isinstance(account_features, dict):
        account_df = pd.DataFrame([account_features])
    else:
        account_df = account_features.copy()
    account_scaled = scaler.transform(account_df)
    account_scaled = pd.DataFrame(account_scaled, columns=account_df.columns)
    prediction = best_rf_model.predict(account_scaled)[0]
    probability = best_rf_model.predict_proba(account_scaled)[0][1]
```

return prediction, probability