Working on the training data

In [70]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import ConfusionMatrixDisplay

df=pd.read_csv('instagram_user.csv')
df=df.iloc[:,:12]
df.sample(5)
```

Out[70]:

	profile pic	nums/length username	fullname words	nums/length fullname	name==username	description length	externa URI
42	1	0.00	2	0.0	0	23	(
313	0	0.00	1	0.0	0	0	(
212	1	0.00	2	0.0	0	62	'
494	0	0.20	1	0.0	0	0	(
195	1	0.14	2	0.0	0	0	(

In [71]:
```
# check for the missing values
df.shape
```

Out[71]: (576, 12)

In [72]:
```
df.describe()
```

Out[72]:

	profile pic	nums/length username	fullname words	nums/length fullname	name==username	description length
count	576.000000	576.000000	576.000000	576.000000	576.000000	576.000000
mean	0.701389	0.163837	1.460069	0.036094	0.034722	22.623264
std	0.458047	0.214096	1.052601	0.125121	0.183234	37.702987
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.000000	0.000000	1.000000	0.000000	0.000000	0.000000
50%	1.000000	0.000000	1.000000	0.000000	0.000000	0.000000
75%	1.000000	0.310000	2.000000	0.000000	0.000000	34.000000
max	1.000000	0.920000	12.000000	1.000000	1.000000	150.000000

In [73]:
```
df.info()
```

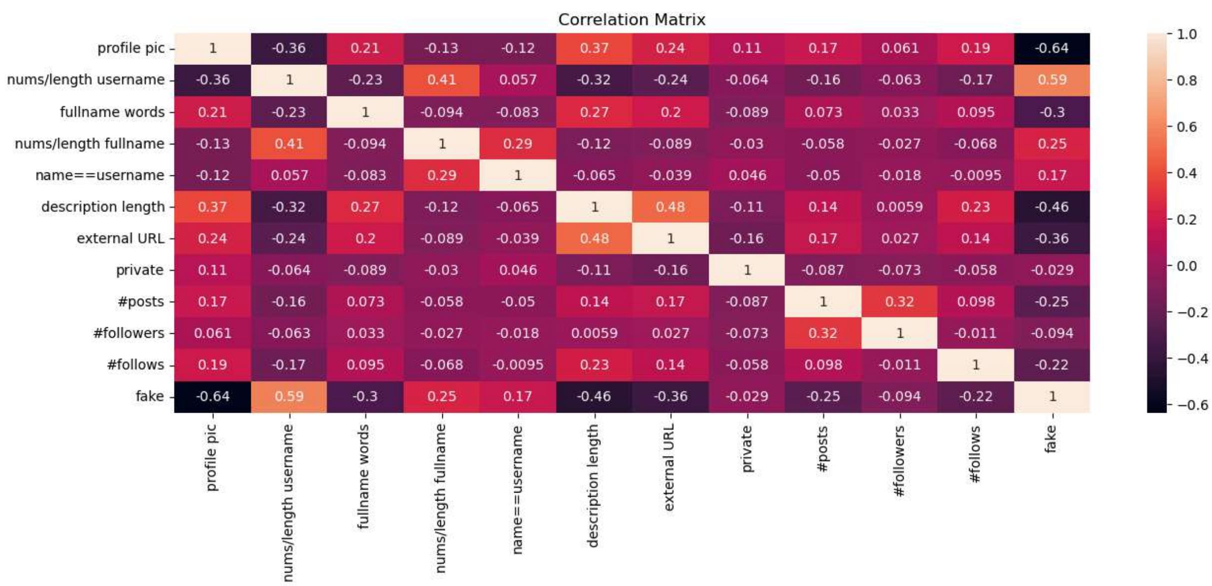
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 576 entries, 0 to 575
Data columns (total 12 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   profile pic           576 non-null    int64  
 1   nums/length username  576 non-null    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
```

In [74]:
```
# number of unique values in dataframe
df.nunique()
```

```
Out[74]: 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                     2
         dtype: int64
```

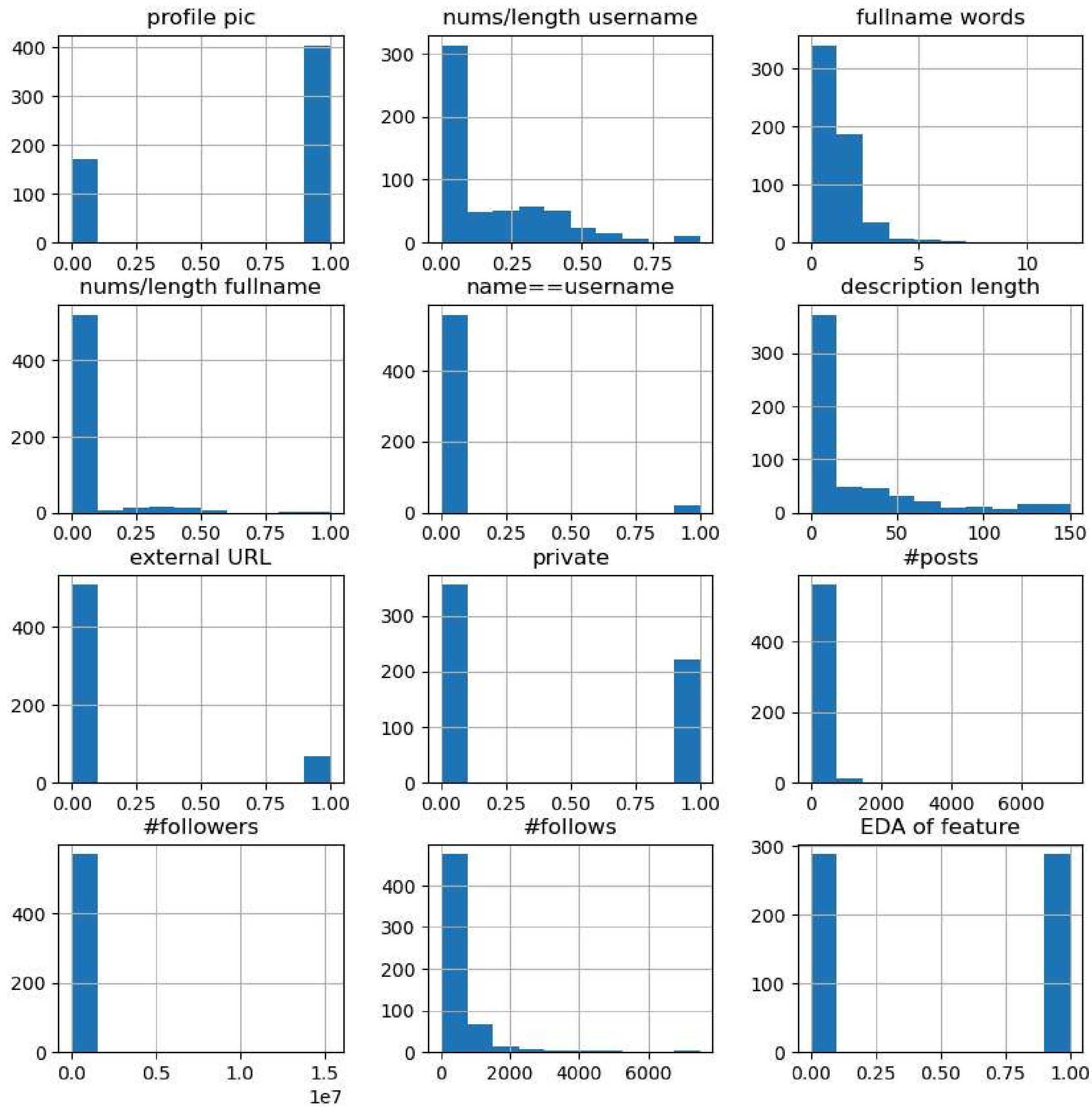
- CORRELATION MATRIX BETWEEN EACH FEATURE

```
In [76]: correlation=df.corr()
         plt.subplots(figsize=(15,5))
         sns.heatmap(correlation,annot=True)
         plt.title('Correlation Matrix')
         plt.show()
```



EDA

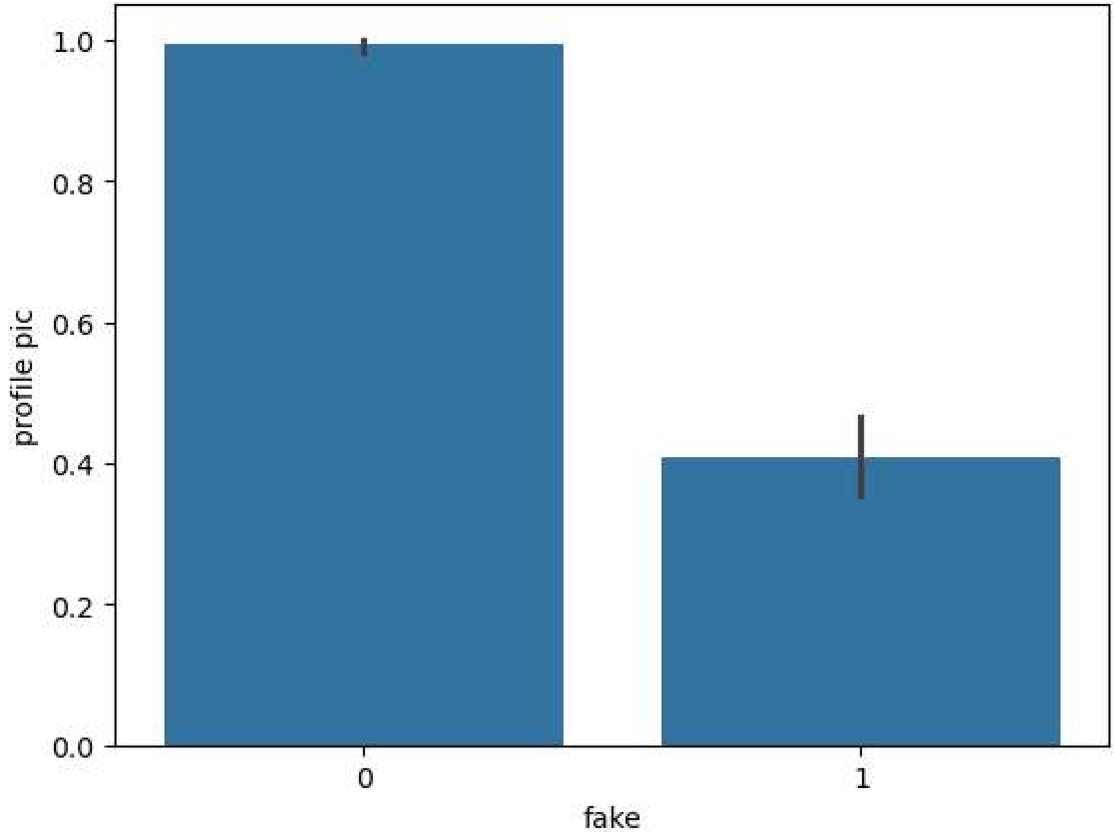
```
In [78]: df.hist(figsize=(10,10))
         plt.title('EDA of feature')
         plt.show()
```



- Profile picture (Fake vs genuine)

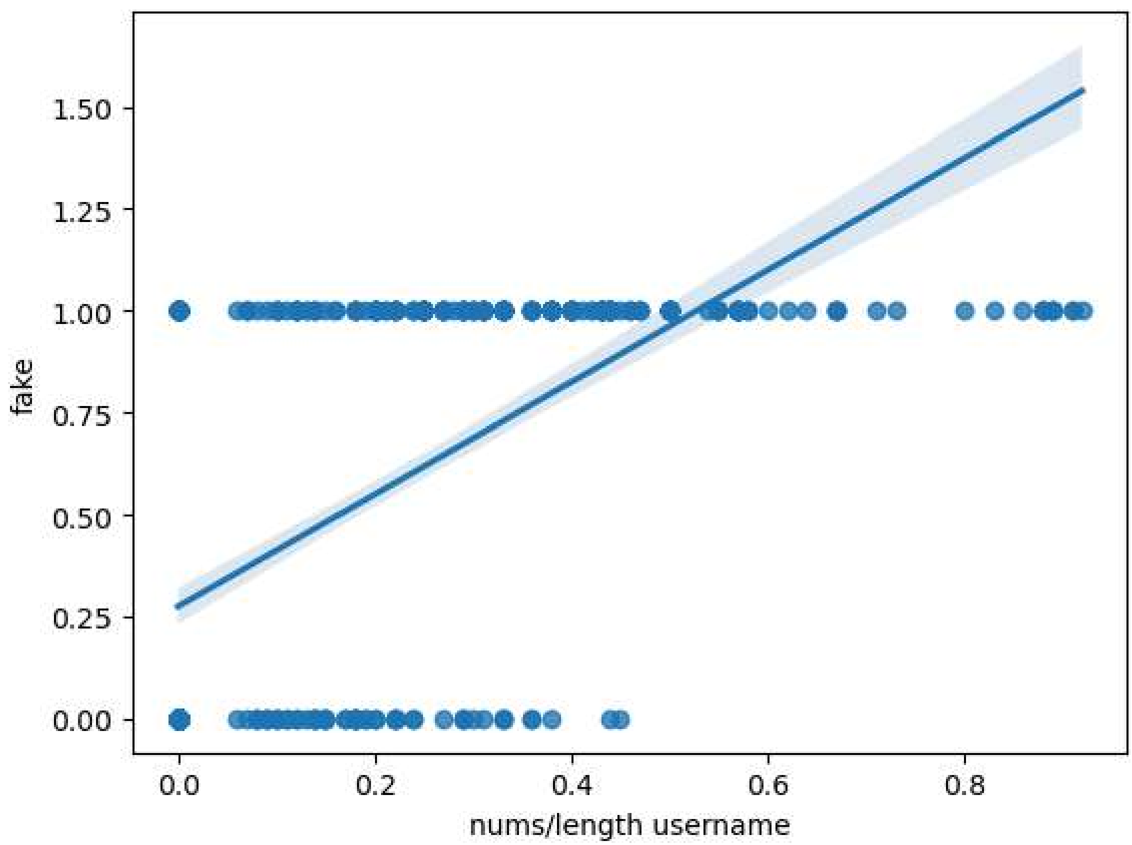
In [80]:
```python
print(df['profile pic'].value_counts())
sns.barplot(x='fake', y='profile pic', data=df)
plt.show()
```

```
profile pic
1    404
0    172
Name: count, dtype: int64
```



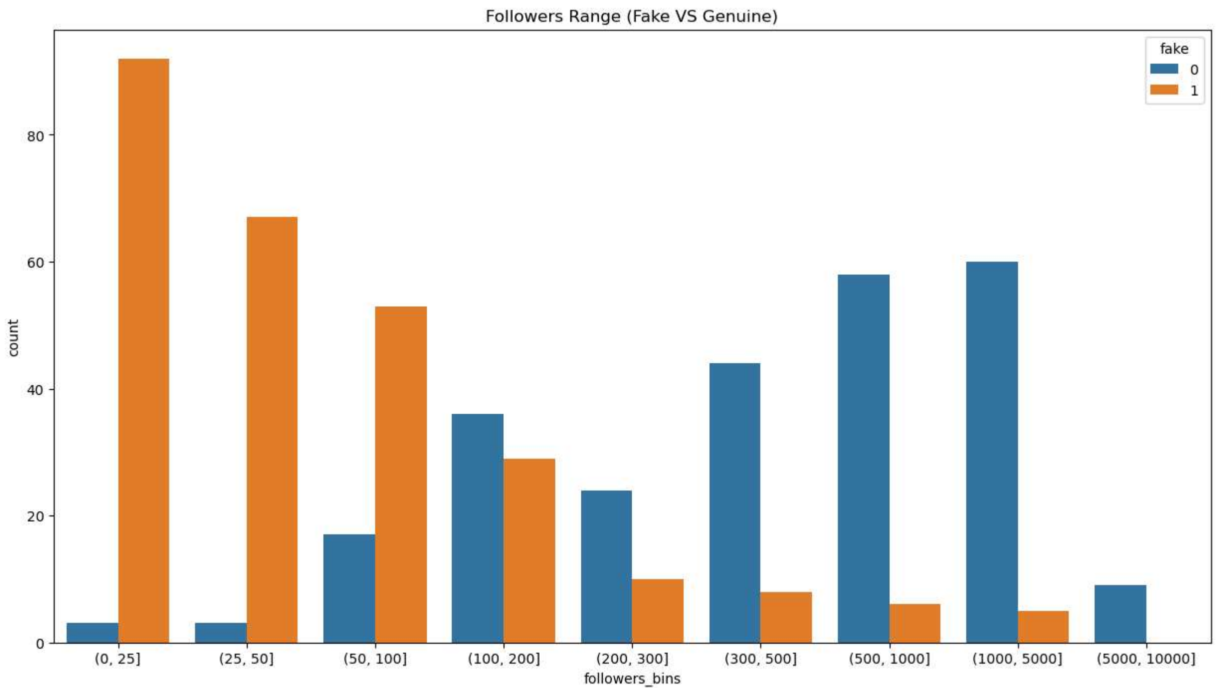
In [81]:
```
sns.regplot(data=df,x='nums/length username',y='fake')
```

Out[81]: <Axes: xlabel='nums/length username', ylabel='fake'>



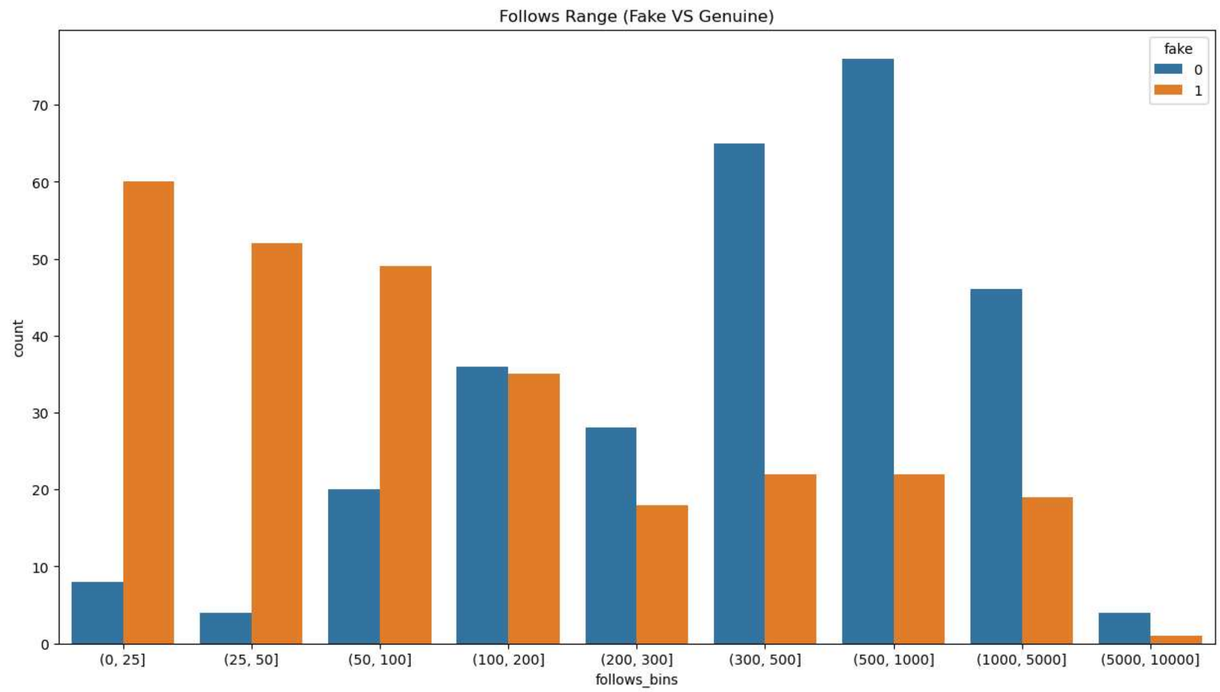
- Followers VS fake

In [83]:
```
# Followers column
bins=[0,25,50,100,200,300,500,1000,5000,10000]
df['followers_bins']=pd.cut(df['#followers'],bins=bins)
plt.subplots(figsize=(15,8))
plt.title('Followers Range (Fake VS Genuine)')
sns.countplot(data=df,x='followers_bins',hue='fake')
plt.show()
```



- Follows which are fake

In [85]:
```
# Follows column which are fake
bins=[0,25,50,100,200,300,500,1000,5000,10000]
df['follows_bins']=pd.cut(df['#follows'],bins=bins)
plt.subplots(figsize=(15,8))
plt.title('Follows Range (Fake VS Genuine)')
sns.countplot(data=df,x='follows_bins',hue='fake')
plt.show()
```



In [86]:

```
# Now removing the follows_bin and followers_bin for further process
df=df.drop(columns=['follows_bins','followers_bins'])
df.head()
```

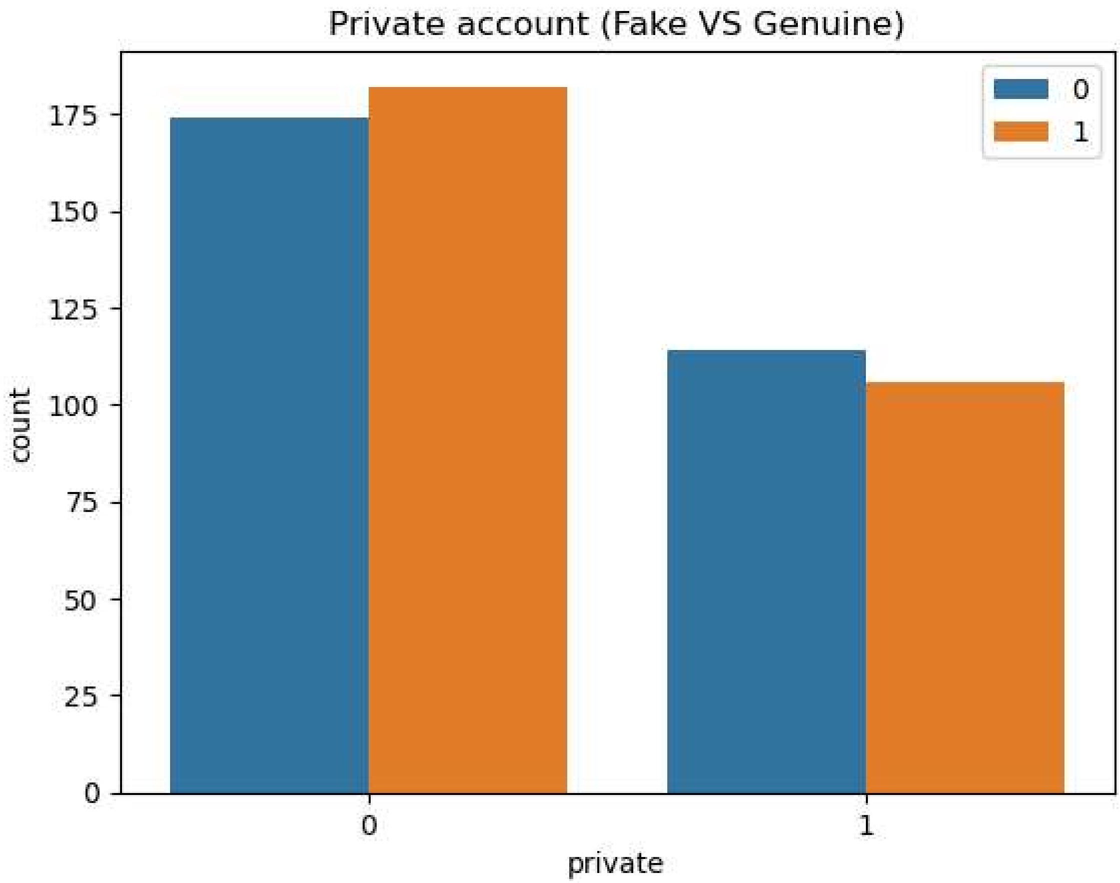
Out[86]:

	profile pic	nums/length username	fullname words	nums/length fullname	name==username	description length	external URL
0	1	0.27	0	0.0	0	53	0
1	1	0.00	2	0.0	0	44	0
2	1	0.10	2	0.0	0	0	0
3	1	0.00	1	0.0	0	82	0
4	1	0.00	2	0.0	0	0	0

- Private account (Fake vs Genuine)

In [88]:

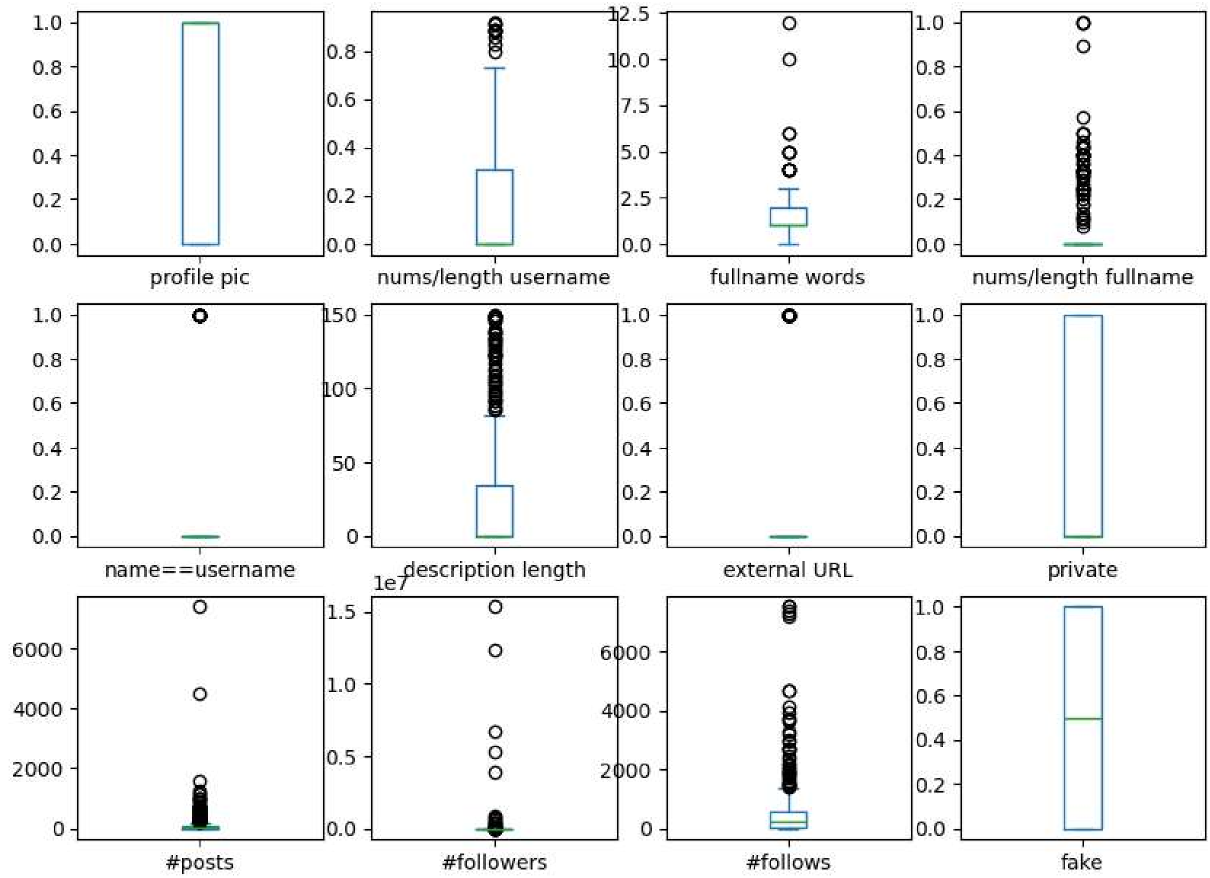
```
sns.countplot(data=df,x='private',hue='fake')
plt.title('Private account (Fake VS Genuine)')
plt.legend()
plt.show()
```



- outlier detection

In [90]:
```
df.plot(kind='box',subplots=True,layout=(4,4),figsize=(10,10))
```

Out[90]:
```
profile pic                 Axes(0.125,0.712609;0.168478x0.167391)
nums/length username     Axes(0.327174,0.712609;0.168478x0.167391)
fullname words           Axes(0.529348,0.712609;0.168478x0.167391)
nums/length fullname     Axes(0.731522,0.712609;0.168478x0.167391)
name==username              Axes(0.125,0.511739;0.168478x0.167391)
description length       Axes(0.327174,0.511739;0.168478x0.167391)
external URL             Axes(0.529348,0.511739;0.168478x0.167391)
private                  Axes(0.731522,0.511739;0.168478x0.167391)
#posts                       Axes(0.125,0.31087;0.168478x0.167391)
#followers                Axes(0.327174,0.31087;0.168478x0.167391)
#follows                  Axes(0.529348,0.31087;0.168478x0.167391)
fake                      Axes(0.731522,0.31087;0.168478x0.167391)
dtype: object
```



calculating accuracy score

In [92]:
```
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier, BaggingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split, GridSearchCV, cross_val_score
from sklearn.metrics import accuracy_score
import numpy as np

# Splitting features and target
x = df.drop(columns=['fake'])
y = df['fake']

# Train-test split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_sta

# Scaling the training and testing data
scaler = StandardScaler()
x_train = scaler.fit_transform(x_train)
x_test = scaler.transform(x_test)

# Initial Random Forest model
clf = RandomForestClassifier(n_estimators=100, random_state=42)
clf.fit(x_train, y_train)
y_pred = clf.predict(x_test)
```

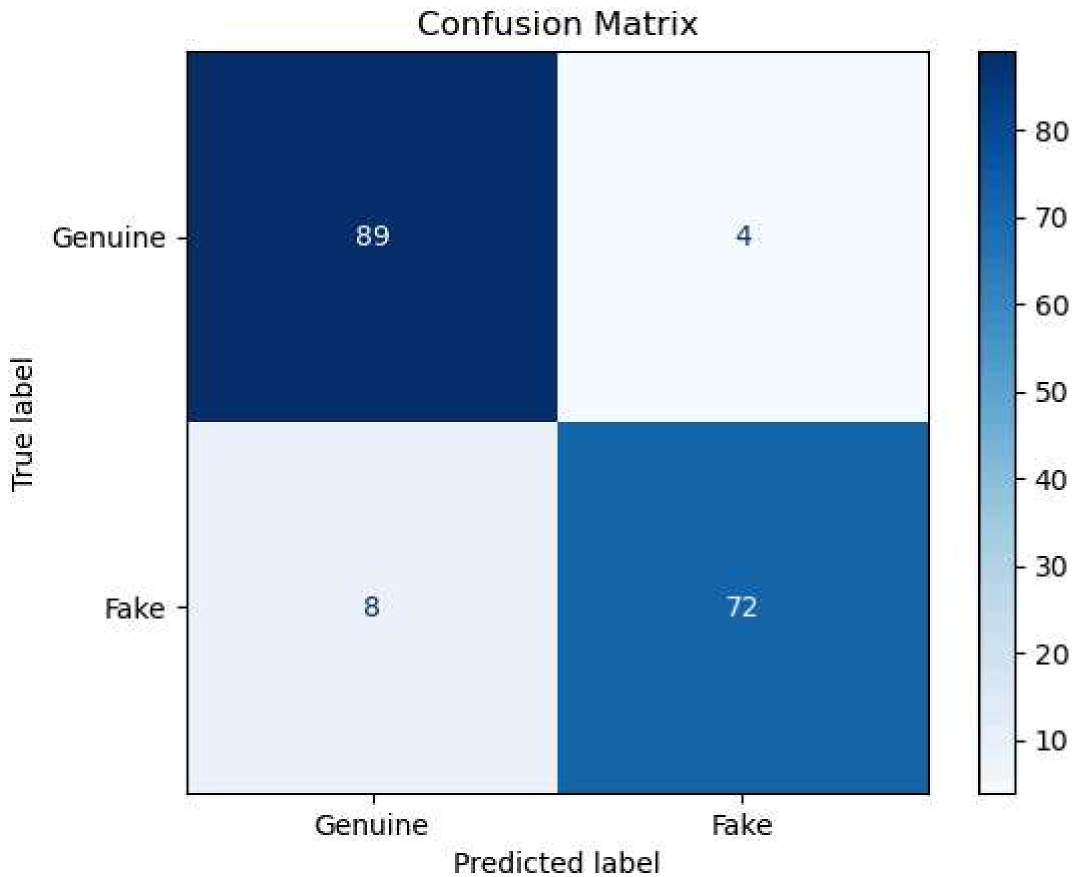
```
# Accuracy before GridSearchCV
accuracy = accuracy_score(y_test, y_pred) * 100
print('Accuracy before GridSearchCV:', accuracy)

# GridSearchCV
param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5]
}
grid_search = GridSearchCV(estimator=clf, param_grid=param_grid, cv=10, scoring='ac
grid_search.fit(x_train, y_train)

# Best model from grid search
best_model = grid_search.best_estimator_
y_pred_best = best_model.predict(x_test)
best_accuracy = accuracy_score(y_test, y_pred_best) * 100
print("Accuracy after GridSearchCV:", best_accuracy)

# Bagging with Decision Tree
bag = BaggingClassifier(
    estimator=DecisionTreeClassifier(),
    n_estimators=100,
    random_state=42
)
bag.fit(x_train, y_train)  # Corrected this line
y_pred1 = bag.predict(x_test)
print('Accuracy after Bagging technique:', accuracy_score(y_test, y_pred1) * 100)

# Cross-validation score using DecisionTree
clf_dt = DecisionTreeClassifier(random_state=42)
scores = cross_val_score(clf_dt, x, y, cv=25, scoring='accuracy')
print("Accuracy with DecisionTree (25-fold CV):", np.mean(scores) * 100)
```

```
Accuracy before GridSearchCV: 93.0635838150289
Accuracy after GridSearchCV: 93.0635838150289
Accuracy after Bagging technique: 93.0635838150289
Accuracy with DecisionTree (25-fold CV): 88.35507246376812
```

visualize confusion matrix

In [161…

```
from sklearn.metrics import confusion_matrix,classification_report
ConfusionMatrixDisplay.from_predictions(y_test, y_pred,
display_labels=['Genuine', 'Fake'], cmap='Blues')
plt.title("Confusion Matrix")
plt.show()

print('Confusion Matrix:\n',confusion_matrix(y_pred,y_test))
print('Classification Report:\n',classification_report(y_pred,y_test))
```




```
Confusion Matrix:
 [[89  8]
 [ 4 72]]
Classification Report:
               precision    recall  f1-score   support

           0       0.96      0.92      0.94        97
           1       0.90      0.95      0.92        76

    accuracy                           0.93       173
   macro avg       0.93      0.93      0.93       173
weighted avg       0.93      0.93      0.93       173
```

plotting feature importance

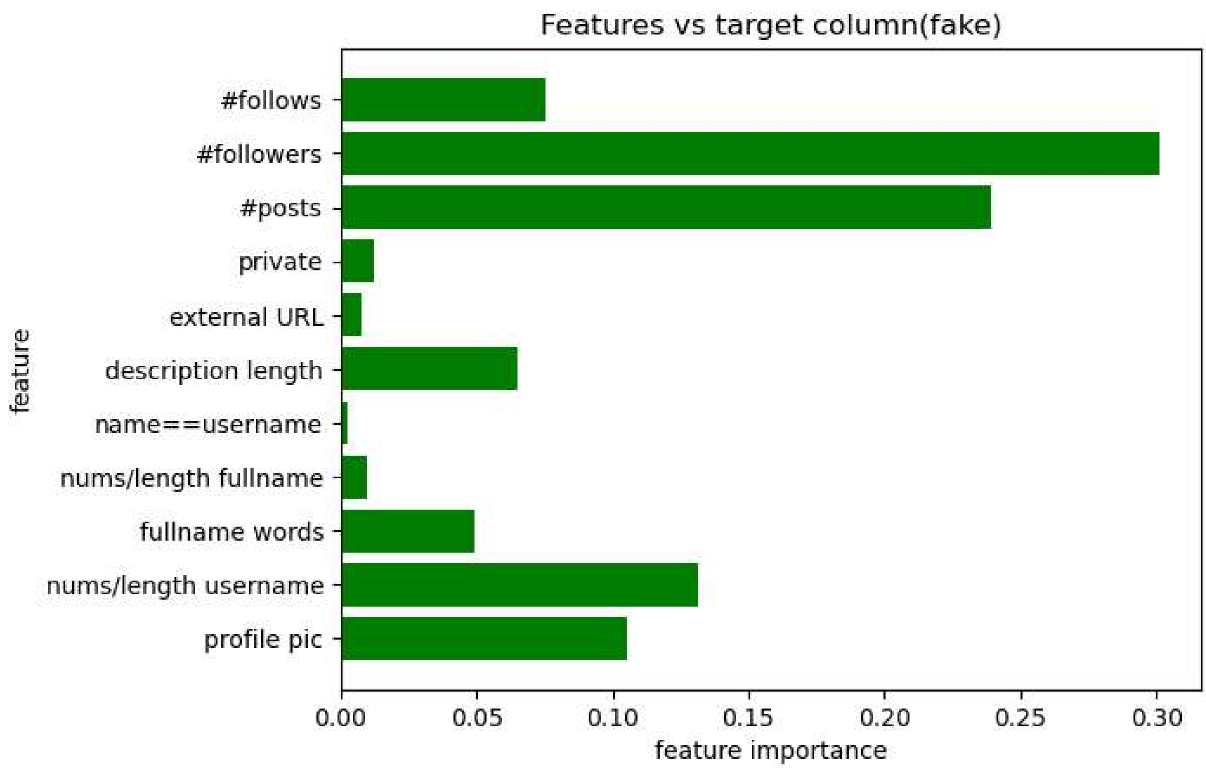
In [157…

```
from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier(random_state=42)
rf.fit(x_train, y_train)
importances = rf.feature_importances_
feature_importance_series = pd.Series(importances, index=x.columns)
print("Feature Importances:\n", feature_importance_series.sort_values(ascending=Fal

plt.barh(x.columns,clf.feature_importances_,color='green')
plt.title('Features vs target column(fake)')
plt.xlabel('feature importance')
```

```
plt.ylabel('feature')
# It shows that '#followers' column is most proportional to the detection of fake
```

```
Feature Importances:
 #followers              0.301345
#posts                  0.239232
nums/length username    0.131646
profile pic             0.105637
#follows                0.075417
description length      0.065220
fullname words          0.049328
private                 0.012381
nums/length fullname    0.009872
external URL            0.007455
name==username          0.002467
dtype: float64
```

Out[157… Text(0, 0.5, 'feature')



In [153…
```
# Final dataset would look as :
df.sample(5)
```

Out[153…

	profile pic	nums/length username	fullname words	nums/length fullname	name==username	description length	externa URl
165	1	0.00	6	0.00	0	117	1
529	0	0.44	1	0.00	0	0	0
103	1	0.00	1	0.00	0	28	0
67	1	0.00	1	0.00	0	42	0
294	0	0.22	1	0.22	0	43	0

In [ ]: