IGAudit: Using Simple Statistical Tools and Machine Learning to Audit Instagram Accounts for Authenticity

Athiya Deviyani

1 Introduction

During the world-wideCoronavirus lockdown, businesses have started increasing the use of social media influencers to market their products while their physical outlets are temporary closed. However, it is sad that there are some that will try and game the system for their own good. But in a world where a single influencer's post is worth as much as an average 9-5 Joe's annual salary, influencer marketing fake followers and fake engagement is a price that brands shouldn't have to pay for.

Inspired by igaudit.io that was taken down by Facebook only recently.

[1]:
```
# Imports

import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix,␣
 ↪classification_report
from sklearn.model_selection import GridSearchCV, cross_val_score,␣
 ↪StratifiedKFold, learning_curve
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier,␣
 ↪GradientBoostingClassifier, ExtraTreesClassifier, VotingClassifier

from instagram_private_api import Client, ClientCompatPatch
import getpass

import random
```

2 Understanding and Splitting the Data

Dataset source: https://www.kaggle.com/eswarchandt/is-your-insta-fake-or-genuine

Import the data

[6]:
```
train = pd.read_csv("train.csv")
test = pd.read_csv("test.csv")
```

Inspect the training data

[70]:
```
train.head()
```

[70]:
```
   profile pic  nums/length username  fullname words  nums/length fullname  \
0            1                  0.27               0                   0.0
1            1                  0.00               2                   0.0
2            1                  0.10               2                   0.0
3            1                  0.00               1                   0.0
4            1                  0.00               2                   0.0

   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

   #followers  #follows  fake
0        1000       955     0
1        2740       533     0
2         159        98     0
3         414       651     0
4         151       126     0
```

The features in the training data are the following: - profile pic: does the user have a profile picture? - nums/length username: ratio of numerical to alphabetical characters in the username - fullname words: how many words are in the user's full name? - nums/length fullname: ratio of numerical to alphabetical characters in the full name - name==username: is the user's full name the same as the username? - description length: how many characters is in the user's Instagram bio? - external URL: does the user have an external URL linked to their profile? - private: is the user private? - #posts: number of posts - #followers: number of people following the user - #follows: number of people the user follows - fake: if the user is fake, fake=1, else fake=0

[4]:
```
train.describe()
```

[4]:
```
       profile pic  nums/length username  fullname words  \
count   576.000000            576.000000      576.000000
mean      0.701389              0.163837        1.460069
std       0.458047              0.214096        1.052601
```

```
min       0.000000              0.000000        0.000000
25%       0.000000              0.000000        1.000000
50%       1.000000              0.000000        1.000000
75%       1.000000              0.310000        2.000000
max       1.000000              0.920000       12.000000

       nums/length fullname  name==username  description length  external URL  \
count            576.000000      576.000000          576.000000    576.000000
mean               0.036094        0.034722           22.623264      0.116319
std                0.125121        0.183234           37.702987      0.320886
min                0.000000        0.000000            0.000000      0.000000
25%                0.000000        0.000000            0.000000      0.000000
50%                0.000000        0.000000            0.000000      0.000000
75%                0.000000        0.000000           34.000000      0.000000
max                1.000000        1.000000          150.000000      1.000000

          private       #posts    #followers     #follows        fake
count  576.000000   576.000000  5.760000e+02   576.000000  576.000000
mean     0.381944   107.489583  8.530724e+04   508.381944    0.500000
std      0.486285   402.034431  9.101485e+05   917.981239    0.500435
min      0.000000     0.000000  0.000000e+00     0.000000    0.000000
25%      0.000000     0.000000  3.900000e+01    57.500000    0.000000
50%      0.000000     9.000000  1.505000e+02   229.500000    0.500000
75%      1.000000    81.500000  7.160000e+02   589.500000    1.000000
max      1.000000  7389.000000  1.533854e+07  7500.000000    1.000000
```

[5]:
```
train.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
```

[8]:
```
train.shape
```

[8]:
```
(576, 12)
```

Inspect the test data

[9]:
```
test.head()
```

[9]:
```
   profile pic  nums/length username  fullname words  nums/length fullname  \
0            1                  0.33               1                  0.33
1            1                  0.00               5                  0.00
2            1                  0.00               2                  0.00
3            1                  0.00               1                  0.00
4            1                  0.50               1                  0.00

   name==username  description length  external URL  private  #posts  \
0               1                  30             0        1      35
1               0                  64             0        1       3
2               0                  82             0        1     319
3               0                 143             0        1     273
4               0                  76             0        1       6

   #followers  #follows  fake
0         488       604     0
1          35         6     0
2         328       668     0
3       14890      7369     0
4         225       356     0
```

[10]:
```
test.describe()
```

[10]:
```
       profile pic  nums/length username  fullname words  \
count   120.000000            120.000000      120.000000
mean      0.758333              0.179917        1.550000
std       0.429888              0.241492        1.187116
min       0.000000              0.000000        0.000000
25%       1.000000              0.000000        1.000000
50%       1.000000              0.000000        1.000000
75%       1.000000              0.330000        2.000000
max       1.000000              0.890000        9.000000

       nums/length fullname  name==username  description length  external URL  \
count            120.000000      120.000000          120.000000    120.000000
mean               0.071333        0.041667           27.200000      0.100000
std                0.209429        0.200664           42.588632      0.301258
min                0.000000        0.000000            0.000000      0.000000
25%                0.000000        0.000000            0.000000      0.000000
```

```
50%                0.000000        0.000000           0.000000       0.000000
75%                0.000000        0.000000          45.250000       0.000000
max                1.000000        1.000000         149.000000       1.000000

          private       #posts    #followers      #follows         fake
count  120.000000   120.000000  1.200000e+02    120.000000   120.000000
mean     0.308333    82.866667  4.959472e+04    779.266667     0.500000
std      0.463741   230.468136  3.816126e+05   1409.383558     0.502096
min      0.000000     0.000000  0.000000e+00      1.000000     0.000000
25%      0.000000     1.000000  6.725000e+01    119.250000     0.000000
50%      0.000000     8.000000  2.165000e+02    354.500000     0.500000
75%      1.000000    58.250000  5.932500e+02    668.250000     1.000000
max      1.000000  1879.000000  4.021842e+06   7453.000000     1.000000
```

[11]:
```
test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 120 entries, 0 to 119
Data columns (total 12 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   profile pic           120 non-null    int64
 1   nums/length username  120 non-null    float64
 2   fullname words        120 non-null    int64
 3   nums/length fullname  120 non-null    float64
 4   name==username        120 non-null    int64
 5   description length    120 non-null    int64
 6   external URL          120 non-null    int64
 7   private               120 non-null    int64
 8   #posts                120 non-null    int64
 9   #followers            120 non-null    int64
 10  #follows              120 non-null    int64
 11  fake                  120 non-null    int64
dtypes: float64(2), int64(10)
memory usage: 11.4 KB
```

[12]:
```
test.shape
```

[12]:
```
(120, 12)
```

Check for NULL values

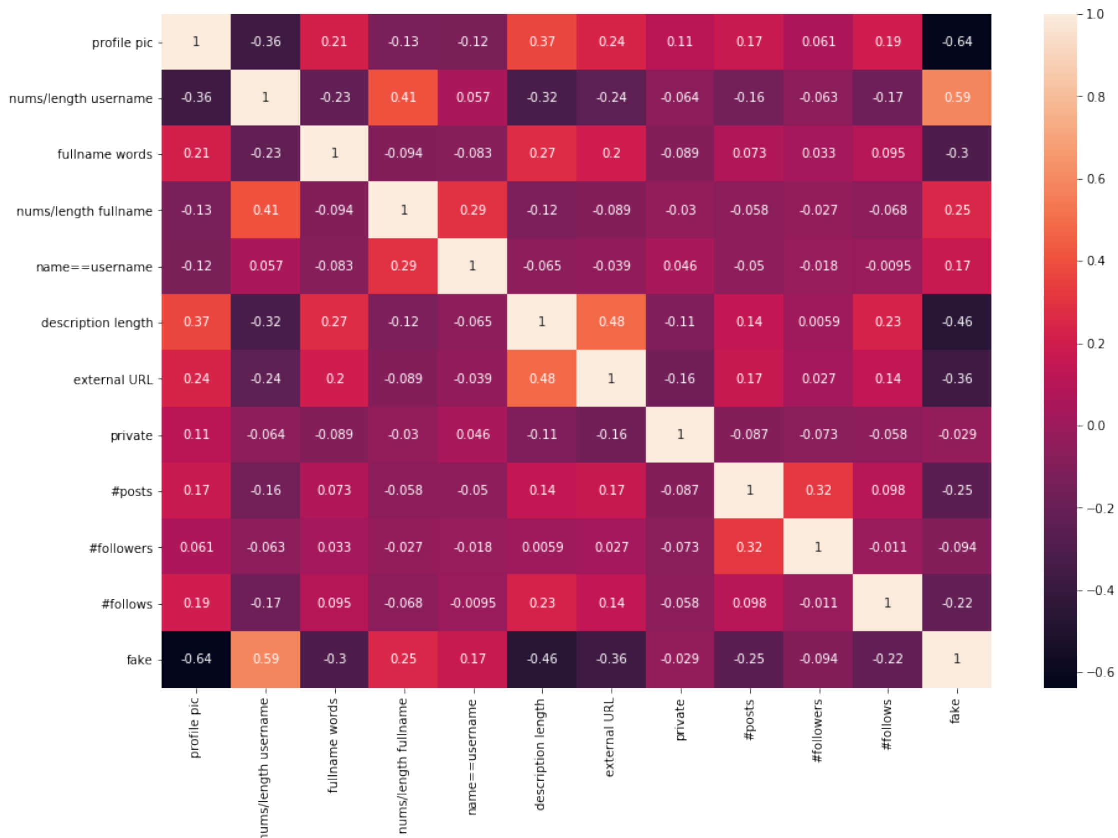
[13]:
```
print(train.isna().values.any().sum())
print(test.isna().values.any().sum())
```

```
0
0
```

Create a correlation matrix for the features in the training data to check for significantly relevant features

[14]:
```
fig, ax = plt.subplots(figsize=(15,10))
corr=train.corr()
sns.heatmap(corr, annot=True)
```

[14]: <matplotlib.axes._subplots.AxesSubplot at 0x10a5f4590>



Split the training set into data and labels

[15]:
```
# Labels
train_Y = train.fake
train_Y = pd.DataFrame(train_Y)

# Data
train_X = train.drop(columns='fake')
train_X.head()
```

[15]:
```
   profile pic  nums/length username  fullname words  nums/length fullname  \
0            1                  0.27               0                   0.0
```

```
1           1                 0.00               2                   0.0
2           1                 0.10               2                   0.0
3           1                 0.00               1                   0.0
4           1                 0.00               2                   0.0

   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

   #followers  #follows
0        1000       955
1        2740       533
2         159        98
3         414       651
4         151       126
```

Split the test set into data and labels

[18]:
```python
# Labels
test_Y = test.fake
test_Y = pd.DataFrame(test_Y)

# Data
test_X = test.drop(columns='fake')
test_X.head()
```

[18]:
```
   profile pic  nums/length username  fullname words  nums/length fullname  \
0            1                  0.33               1                  0.33
1            1                  0.00               5                  0.00
2            1                  0.00               2                  0.00
3            1                  0.00               1                  0.00
4            1                  0.50               1                  0.00

   name==username  description length  external URL  private  #posts  \
0               1                  30             0        1      35
1               0                  64             0        1       3
2               0                  82             0        1     319
3               0                 143             0        1     273
4               0                  76             0        1       6

   #followers  #follows
0         488       604
1          35         6
2         328       668
```

3	14890	7369
4	225	356

3 Comparing Classification Models

Baseline Classifier Classify everything as the majority class.

[22]:
```
# Baseline classifier
fakes = len([i for i in train.fake if i==1])
auth = len([i for i in train.fake if i==0])
fakes, auth

# classify everything as fake
pred = [1 for i in range(len(test_X))]
pred = np.array(pred)
print("Baseline accuracy: " + str(accuracy_score(pred, test_Y)))
```

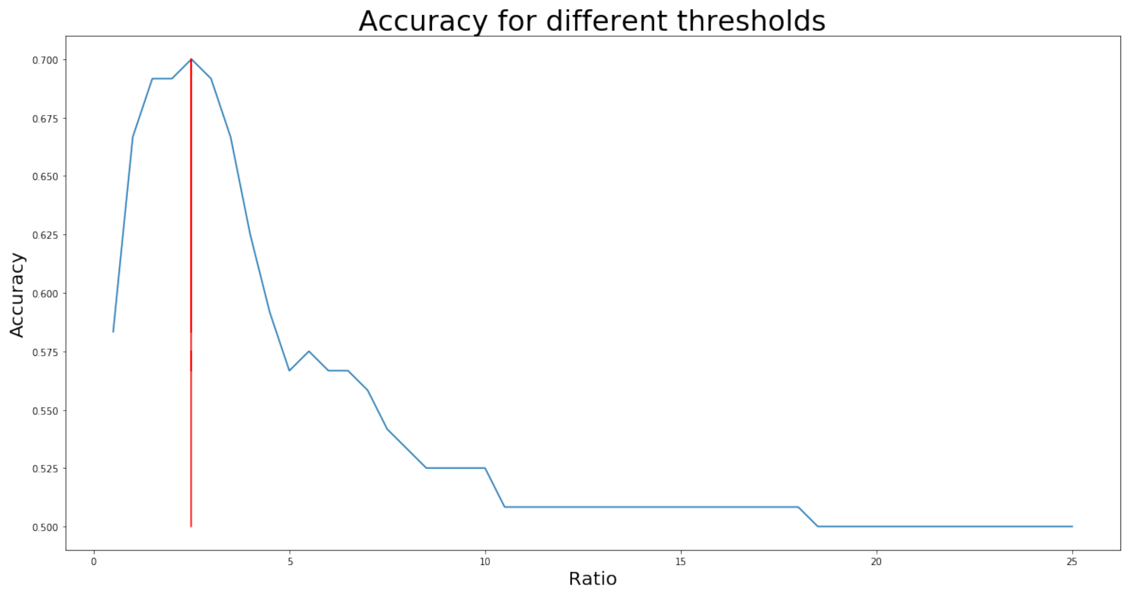
```
Baseline accuracy: 0.5
```

Statistical Method Classify all users with a following to follower ratio above a certain threshold as 'fake'. i.e. a user with 10 follower and 200 followings will be classified as fake if the threshold r=20

[41]:
```
# Statistical method
def stat_predict(test_X, r):
    pred = []
    for row in range(len(test_X)):
        followers = test_X.loc[row]['#followers']
        followings = test_X.loc[row]['#follows']
        if followers == 0:
            followers = 1
        if followings == 0:
            followings == 1

        ratio = followings/followers

        if ratio >= r:
            pred.append(1)
        else:
            pred.append(0)

    return np.array(pred)
accuracies = []
for i in [x / 10.0 for x in range(5, 255, 5)]:
    prediction = stat_predict(test_X, i)
    accuracies.append(accuracy_score(prediction, test_Y))
```

```
f, ax = plt.subplots(figsize=(20,10))
plt.plot([x / 10.0 for x in range(5, 255, 5)], accuracies)
plt.plot([2.5 for i in range(len(accuracies))], accuracies, color='red')
plt.title("Accuracy for different thresholds", size=30)
plt.xlabel('Ratio', fontsize=20)
plt.ylabel('Accuracy', fontsize=20)
print("Maximum Accuracy for the statistical method: " + str(max(accuracies)))
```

```
Maximum Accuracy for the statistical method: 0.7
```



Logistic Regression

[19]:
```
lm = LogisticRegression()

# Train the model
model1 = lm.fit(train_X, train_Y)

# Make a prediction
lm_predict = model1.predict(test_X)
```

```
/Users/athiyadeviyani/miniconda3/lib/python3.7/site-
packages/sklearn/utils/validation.py:73: DataConversionWarning: A column-vector
y was passed when a 1d array was expected. Please change the shape of y to
(n_samples, ), for example using ravel().
  return f(**kwargs)
/Users/athiyadeviyani/miniconda3/lib/python3.7/site-
packages/sklearn/linear_model/_logistic.py:762: ConvergenceWarning: lbfgs failed
to converge (status=1):
```

```
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
```

[23]:
```
# Compute the accuracy of the model
acc = accuracy_score(lm_predict, test_Y)
print("Logistic Regression accuracy: " + str(acc))
```

```
Logistic Regression accuracy: 0.9083333333333333
```

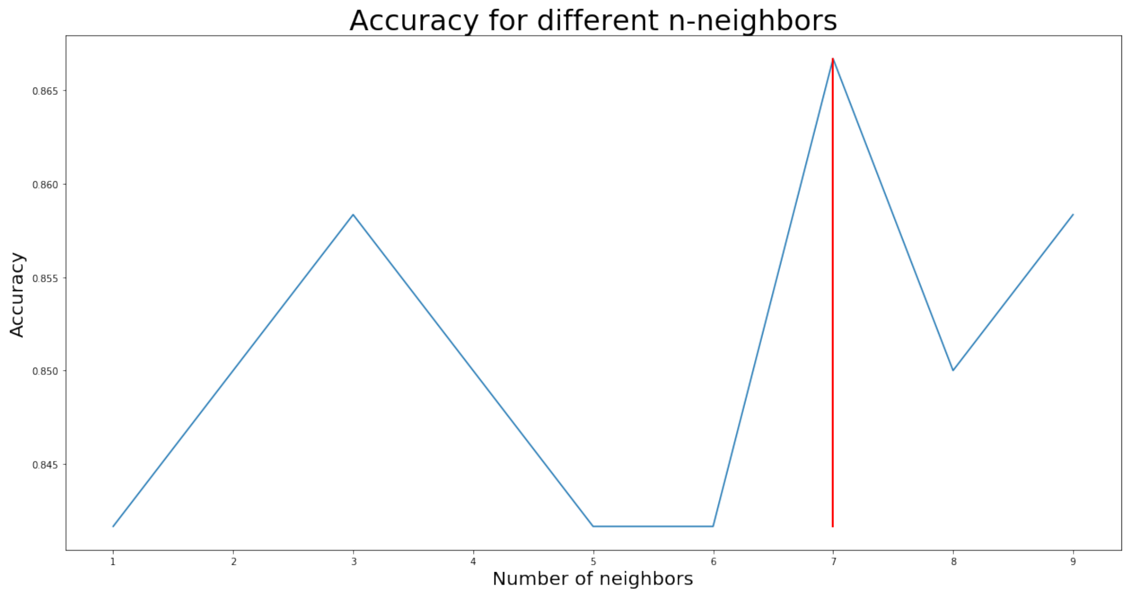
KNN Classifier

[42]:
```
accuracies = []

# Compare the accuracies of using the KNN classifier with different number of␣
 ↪neighbors
for i in range(1,10):
    knn = KNeighborsClassifier(n_neighbors=i)
    model_2 = knn.fit(train_X,train_Y)
    knn_predict = model_2.predict(test_X)
    accuracy = accuracy_score(knn_predict,test_Y)
    accuracies.append(accuracy)

max_acc = (0, 0)
for i in range(1, 10):
    if accuracies[i-1] > max_acc[1]:
        max_acc = (i, accuracies[i-1])

max_acc

f, ax = plt.subplots(figsize=(20,10))
plt.plot([i for i in range(1,10)], accuracies)
plt.plot([7 for i in range(len(accuracies))], accuracies, color='red')
plt.title("Accuracy for different n-neighbors", size=30)
plt.xlabel('Number of neighbors', fontsize=20)
plt.ylabel('Accuracy', fontsize=20)

print("The highest accuracy obtained using KNN is " + str(max_acc[1]) + "␣
 ↪achieved by a value of n=" + str(max_acc[0]))
```

```
/Users/athiyadeviyani/miniconda3/lib/python3.7/site-
packages/ipykernel_launcher.py:6: DataConversionWarning: A column-vector y was
passed when a 1d array was expected. Please change the shape of y to (n_samples,
), for example using ravel().
```

```
/Users/athiyadeviyani/miniconda3/lib/python3.7/site-
packages/ipykernel_launcher.py:6: DataConversionWarning: A column-vector y was
passed when a 1d array was expected. Please change the shape of y to (n_samples,
), for example using ravel().

/Users/athiyadeviyani/miniconda3/lib/python3.7/site-
packages/ipykernel_launcher.py:6: DataConversionWarning: A column-vector y was
passed when a 1d array was expected. Please change the shape of y to (n_samples,
), for example using ravel().

/Users/athiyadeviyani/miniconda3/lib/python3.7/site-
packages/ipykernel_launcher.py:6: DataConversionWarning: A column-vector y was
passed when a 1d array was expected. Please change the shape of y to (n_samples,
), for example using ravel().

/Users/athiyadeviyani/miniconda3/lib/python3.7/site-
packages/ipykernel_launcher.py:6: DataConversionWarning: A column-vector y was
passed when a 1d array was expected. Please change the shape of y to (n_samples,
), for example using ravel().

/Users/athiyadeviyani/miniconda3/lib/python3.7/site-
packages/ipykernel_launcher.py:6: DataConversionWarning: A column-vector y was
passed when a 1d array was expected. Please change the shape of y to (n_samples,
), for example using ravel().

/Users/athiyadeviyani/miniconda3/lib/python3.7/site-
packages/ipykernel_launcher.py:6: DataConversionWarning: A column-vector y was
passed when a 1d array was expected. Please change the shape of y to (n_samples,
), for example using ravel().

/Users/athiyadeviyani/miniconda3/lib/python3.7/site-
packages/ipykernel_launcher.py:6: DataConversionWarning: A column-vector y was
passed when a 1d array was expected. Please change the shape of y to (n_samples,
), for example using ravel().

/Users/athiyadeviyani/miniconda3/lib/python3.7/site-
packages/ipykernel_launcher.py:6: DataConversionWarning: A column-vector y was
passed when a 1d array was expected. Please change the shape of y to (n_samples,
), for example using ravel().
```

```
The highest accuracy obtained using KNN is 0.8666666666666667 achieved by a
value of n=7
```



Decision Tree Classifier

[43]:
```python
DT = DecisionTreeClassifier()

# Train the model
model3 = DT.fit(train_X, train_Y)

# Make a prediction
DT_predict = model3.predict(test_X)
```

[45]:
```python
# Compute the accuracy of the model
acc = accuracy_score(DT_predict, test_Y)
print("Decision Tree accuracy: " + str(acc))
```

```
Decision Tree accuracy: 0.9
```

Random Forest Classifier

[46]:
```python
rfc = RandomForestClassifier()

# Train the model
model_4 = rfc.fit(train_X, train_Y)

# Make a prediction
rfc_predict = model_4.predict(test_X)
```

```
/Users/athiyadeviyani/miniconda3/lib/python3.7/site-
packages/ipykernel_launcher.py:4: DataConversionWarning: A column-vector y was
passed when a 1d array was expected. Please change the shape of y to
```

```
(n_samples,), for example using ravel().
  after removing the cwd from sys.path.
```

[47]:
```
# Compute the accuracy of the model
acc = accuracy_score(rfc_predict, test_Y)
print("Random Forest accuracy: " + str(acc))
```

```
Random Forest accuracy: 0.925
```

4 Obtaining Instagram Data

We are going to use the hassle-free unofficial Instagram API. To install: `$ pip install git+https://git@github.com/ping/instagram_private_api.git@1.6.0`

Log in to your Instagram account (preferably not your personal one! I created one just for this project)

[49]:
```
def login():
    username = input("username: ")
    password = getpass.getpass("password: ")
    api = Client(username, password)
    return api

api = login()
```

```
username: ins.tapolice
password: ········
```

Get the Instagram user ID

[50]:
```
def get_ID(username):
    return api.username_info(username)['user']['pk']
```

[58]:
```
# The user used for the experiment below is anonymised!
# i.e. this cell was run and then changed to protect the user's anonymity
userID = get_ID('<USERNAME HERE>')
```

The API needs some sort of rank to query followers, posts, etc.

[55]:
```
rank = api.generate_uuid()
```

Get the user's list follower usernames (this may take a while, depending on how many followers the user have)

[56]:
```
def get_followers(userID, rank):
    followers = []
    next_max_id = True

    while next_max_id:
```

```
        if next_max_id == True: next_max_id=''
        f = api.user_followers(userID, rank, max_id=next_max_id)
        followers.extend(f.get('users', []))
        next_max_id = f.get('next_max_id', '')

    user_fer = [dic['username'] for dic in followers]

    return user_fer
```

[59]:
```
followers = get_followers(userID, rank)
```

[63]:
```
# You can check the number of followers if you'd like to
# len(followers)
```

5 Preparing the Data

Inspect the data (and what other data can you obtain from it) and compare it with the train and test tables above. Find out what you need to do to obtain the features for a data point in order to make a prediction.

Recall that the features for a data point are the following: - profile pic: does the user have a profile picture? - nums/length username: ratio of numerical to alphabetical characters in the username - fullname words: how many words are in the user's full name? - nums/length fullname: ratio of numerical to alphabetical characters in the full name - name==username: is the user's full name the same as the username? - description length: how many characters is in the user's Instagram bio? - external URL: does the user have an external URL linked to their profile? - private: is the user private? - #posts: number of posts - #followers: number of people following the user - #follows: number of people the user follows - fake: if the user is fake, fake=1, else fake=0

[65]:
```
# This will print the first follower username on the list
# print(followers[0])
```

[67]:
```
# This will get the information on a certain user
info = api.user_info(get_ID(followers[0]))['user']

# Check what information is available for one particular user
info.keys()
```

[67]:
```
dict_keys(['pk', 'username', 'full_name', 'is_private', 'profile_pic_url',
'profile_pic_id', 'is_verified', 'has_anonymous_profile_picture', 'media_count',
'geo_media_count', 'follower_count', 'following_count', 'following_tag_count',
'biography', 'biography_with_entities', 'external_url', 'external_lynx_url',
'total_igtv_videos', 'total_clips_count', 'total_ar_effects', 'usertags_count',
'is_favorite', 'is_favorite_for_stories', 'is_favorite_for_highlights',
'live_subscription_status', 'is_interest_account', 'has_chaining',
'hd_profile_pic_versions', 'hd_profile_pic_url_info', 'mutual_followers_count',
'has_highlight_reels', 'can_be_reported_as_fraud',
```

```
'is_eligible_for_smb_support_flow', 'smb_support_partner',
'smb_delivery_partner', 'smb_donation_partner', 'smb_support_delivery_partner',
'displayed_action_button_type', 'direct_messaging', 'fb_page_call_to_action_id',
'address_street', 'business_contact_method', 'category', 'city_id', 'city_name',
'contact_phone_number', 'is_call_to_action_enabled', 'latitude', 'longitude',
'public_email', 'public_phone_country_code', 'public_phone_number', 'zip',
'instagram_location_id', 'is_business', 'account_type',
'professional_conversion_suggested_account_type', 'can_hide_category',
'can_hide_public_contacts', 'should_show_category',
'should_show_public_contacts', 'personal_account_ads_page_name',
'personal_account_ads_page_id', 'include_direct_blacklist_status',
'is_potential_business', 'show_post_insights_entry_point', 'is_bestie',
'has_unseen_besties_media', 'show_account_transparency_details',
'show_leave_feedback', 'robi_feedback_source', 'auto_expand_chaining',
'highlight_reshare_disabled', 'is_memorialized',
'open_external_url_with_in_app_browser'])
```

You can see that we have pretty much all the features to make a user data point for prediction, but we need to filter and extract them, and perform some very minor calculations. The following function will do just that:

[75]:
```
def get_data(info):

    """Extract the information from the returned JSON.

    This function will return the following array:
        data = [profile pic,
                nums/length username,
                full name words,
                nums/length full name,
                name==username,
                description length,
                external URL,
                private,
                #posts,
                #followers,
                #followings]
    """

    data = []

    # Does the user have a profile photo?
    profile_pic = not info['has_anonymous_profile_picture']
    if profile_pic == True:
        profile_pic = 1
    else:
        profile_pic = 0
```

```python
    data.append(profile_pic)

    # Ratio of number of numerical chars in username to its length
    username = info['username']
    uname_ratio = len([x for x in username if x.isdigit()]) /␣
↪float(len(username))
    data.append(uname_ratio)

    # Full name in word tokens
    full_name = info['full_name']
    fname_tokens = len(full_name.split(' '))
    data.append(fname_tokens)

    # Ratio of number of numerical characters in full name to its length
    if len(full_name) == 0:
        fname_ratio = 0
    else:
        fname_ratio = len([x for x in full_name if x.isdigit()]) /␣
↪float(len(full_name))
    data.append(fname_ratio)

    # Is name == username?
    name_eq_uname = (full_name == username)
    if name_eq_uname == True:
        name_eq_uname = 1
    else:
        name_eq_uname = 0
    data.append(name_eq_uname)

    # Number of characters on user bio
    bio_length = len(info['biography'])
    data.append(bio_length)

    # Does the user have an external URL?
    ext_url = info['external_url'] != ''
    if ext_url == True:
        ext_url = 1
    else:
        ext_url = 0
    data.append(ext_url)

    # Is the user private or no?
    private = info['is_private']
    if private == True:
        private = 1
    else:
        private = 0
```

```
    data.append(private)

    # Number of posts
    posts = info['media_count']
    data.append(posts)

    # Number of followers
    followers = info['follower_count']
    data.append(followers)

    # Number of followings
    followings = info['following_count']
    data.append(followings)


    return data
```

[73]:
```
# Check if the function returns as expected
get_data(info)
```

[73]: `[1, 0.0, 3, 0.0, 0, 118, 1, 0, 589, 22227, 510]`

Unfortunately the Instagram Private API has a very limited number of API calls per hour so we will not be able to analyse *all* of the user's followers.

Fortunately, I took Statistics and learned that **random sampling** is useful to cull a smaller sample size from a larger population and use it to research and make generalizations about the larger group.

This will allow us to make user authenticity approximations despite the API limitations and still have a data that is representative of the user's followers.

[96]:
```
# Get a random sample of 50 followers
random_followers = random.sample(followers, 50)
```

Get user information for each follower

[100]:
```
f_infos = []

for follower in random_followers:
    info = api.user_info(get_ID(follower))['user']
    f_infos.append(info)
```

Extract the relevant features

[102]:
```
f_table = []

for info in f_infos:
    f_table.append(get_data(info))
```

```
f_table
```

[102]:
```
[[1, 0.0, 3, 0.0, 0, 43, 0, 1, 108, 788, 764],
 [1, 0.0, 1, 0, 0, 45, 0, 0, 1, 252, 483],
 [1, 0.0, 3, 0.0, 0, 90, 0, 0, 536, 1818, 7486],
 [1, 0.5, 3, 0.0, 0, 0, 0, 0, 157, 148, 813],
 [1, 0.0, 1, 0.0, 0, 102, 0, 1, 24, 481, 592],
 [1, 0.0, 1, 0.0, 0, 59, 0, 1, 19, 773, 3639],
 [1, 0.0, 1, 0, 0, 8, 0, 1, 0, 3, 3639],
 [1, 0.0, 3, 0.0, 0, 90, 1, 0, 27, 63, 19],
 [1, 0.0, 4, 0.0, 0, 148, 0, 1, 458, 682, 436],
 [1, 0.0, 2, 0.0, 0, 0, 0, 1, 35, 1054, 1046],
 [1, 0.36363636363636365, 1, 0.0, 0, 96, 0, 1, 96, 50, 98],
 [1, 0.0, 1, 0.0, 0, 0, 0, 1, 2, 10, 202],
 [1, 0.0, 2, 0.0, 0, 135, 1, 1, 159, 52, 240],
 [1, 0.0, 1, 0.0, 0, 20, 0, 0, 87, 1864, 692],
 [1, 0.0, 1, 0.0, 0, 0, 0, 1, 35, 275, 2039],
 [1, 0.0625, 3, 0.0, 0, 98, 0, 0, 9, 98, 847],
 [1, 0.0, 3, 0.0, 0, 92, 0, 1, 10, 11, 46],
 [1, 0.0, 2, 0.0, 0, 69, 0, 1, 16, 2686, 6570],
 [1, 0.0, 2, 0.0, 0, 68, 0, 1, 31, 18, 64],
 [1, 0.0, 3, 0.0, 0, 6, 0, 0, 27, 1628, 1037],
 [1, 0.0, 1, 0, 0, 2, 0, 0, 21, 1730, 1298],
 [0, 0.18181818181818182, 2, 0.0, 0, 0, 0, 1, 219, 183, 275],
 [1, 0.0, 2, 0.0, 0, 38, 0, 0, 11, 645, 4452],
 [1, 0.0, 2, 0.0, 0, 30, 1, 0, 42, 1258, 952],
 [1, 0.0, 1, 0.0, 0, 9, 0, 0, 2, 629, 485],
 [1, 0.23529411764705882, 1, 0.0, 0, 62, 0, 1, 12, 1270, 951],
 [1, 0.0, 1, 0.0, 0, 86, 0, 0, 299, 1669, 1133],
 [1, 0.0, 2, 0.0, 0, 14, 0, 0, 11, 753, 853],
 [1, 0.2, 2, 0.0, 0, 9, 0, 0, 0, 213, 700],
 [1, 0.0, 1, 0.0, 0, 133, 0, 1, 11, 28, 169],
 [1, 0.0, 2, 0.0, 0, 0, 0, 1, 3, 1395, 794],
 [1, 0.0, 2, 0.0, 0, 0, 0, 0, 71, 831, 1024],
 [1, 0.0, 3, 0.0, 0, 29, 0, 0, 61, 680, 566],
 [1, 0.0, 2, 0.0, 0, 64, 0, 0, 1729, 6114, 5758],
 [1, 0.0, 2, 0.0, 0, 17, 0, 0, 73, 2104, 7091],
 [1, 0.0, 3, 0.0, 0, 36, 0, 1, 20, 728, 4139],
 [1, 0.0, 2, 0.0, 0, 106, 0, 1, 23, 83, 458],
 [1, 0.0, 2, 0.0, 0, 31, 0, 1, 78, 2035, 1035],
 [1, 0.0, 2, 0.0, 0, 35, 0, 1, 12, 11549, 712],
 [1, 0.0, 3, 0.08333333333333333, 0, 100, 0, 1, 56, 39, 190],
 [1, 0.13333333333333333, 1, 0.0, 0, 103, 0, 1, 109, 1053, 6221],
 [1, 0.0, 1, 0.0, 0, 0, 0, 0, 49, 412, 520],
 [1, 0.0, 1, 0, 0, 7, 0, 0, 110, 317, 334],
 [1, 0.0, 1, 0.0, 0, 31, 1, 0, 141, 2490, 1043],
```

```
[1, 0.18181818181818182, 2, 0.0, 0, 35, 1, 0, 320, 2345, 861],
[1, 0.0, 3, 0.0, 0, 115, 0, 1, 1336, 1018, 1208],
[1, 0.0, 1, 0.0, 0, 0, 0, 1, 39, 37, 611],
[1, 0.0, 1, 0.0, 0, 0, 0, 1, 0, 513, 633],
[1, 0.0, 2, 0.0, 0, 46, 0, 0, 23, 83, 306],
[1, 0.0, 1, 0.0, 0, 0, 0, 0, 30, 126, 372]]
```

Create a pandas dataframe

[103]:
```
test_data = pd.DataFrame(f_table,
                         columns = ['profile pic',
                                    'nums/length username',
                                    'fullname words',
                                    'nums/length fullname',
                                    'name==username',
                                    'description length',
                                    'external URL',
                                    'private',
                                    '#posts',
                                    '#followers',
                                    '#follows'])
test_data
```

[103]:
```
    profile pic  nums/length username  fullname words  nums/length fullname  \
0             1              0.000000               3              0.000000
1             1              0.000000               1              0.000000
2             1              0.000000               3              0.000000
3             1              0.500000               3              0.000000
4             1              0.000000               1              0.000000
5             1              0.000000               1              0.000000
6             1              0.000000               1              0.000000
7             1              0.000000               3              0.000000
8             1              0.000000               4              0.000000
9             1              0.000000               2              0.000000
10            1              0.363636               1              0.000000
11            1              0.000000               1              0.000000
12            1              0.000000               2              0.000000
13            1              0.000000               1              0.000000
14            1              0.000000               1              0.000000
15            1              0.062500               3              0.000000
16            1              0.000000               3              0.000000
17            1              0.000000               2              0.000000
18            1              0.000000               2              0.000000
19            1              0.000000               3              0.000000
20            1              0.000000               1              0.000000
21            0              0.181818               2              0.000000
22            1              0.000000               2              0.000000
```

23	1	0.000000	2	0.000000
24	1	0.000000	1	0.000000
25	1	0.235294	1	0.000000
26	1	0.000000	1	0.000000
27	1	0.000000	2	0.000000
28	1	0.200000	2	0.000000
29	1	0.000000	1	0.000000
30	1	0.000000	2	0.000000
31	1	0.000000	2	0.000000
32	1	0.000000	3	0.000000
33	1	0.000000	2	0.000000
34	1	0.000000	2	0.000000
35	1	0.000000	3	0.000000
36	1	0.000000	2	0.000000
37	1	0.000000	2	0.000000
38	1	0.000000	2	0.000000
39	1	0.000000	3	0.083333
40	1	0.133333	1	0.000000
41	1	0.000000	1	0.000000
42	1	0.000000	1	0.000000
43	1	0.000000	1	0.000000
44	1	0.181818	2	0.000000
45	1	0.000000	3	0.000000
46	1	0.000000	1	0.000000
47	1	0.000000	1	0.000000
48	1	0.000000	2	0.000000
49	1	0.000000	1	0.000000

	name==username	description length	external URL	private	#posts	\
0	0	43	0	1	108	
1	0	45	0	0	1	
2	0	90	0	0	536	
3	0	0	0	0	157	
4	0	102	0	1	24	
5	0	59	0	1	19	
6	0	8	0	1	0	
7	0	90	1	0	27	
8	0	148	0	1	458	
9	0	0	0	1	35	
10	0	96	0	1	96	
11	0	0	0	1	2	
12	0	135	1	1	159	
13	0	20	0	0	87	
14	0	0	0	1	35	
15	0	98	0	0	9	
16	0	92	0	1	10	
17	0	69	0	1	16	

18	0	68	0	1	31
19	0	6	0	0	27
20	0	2	0	0	21
21	0	0	0	1	219
22	0	38	0	0	11
23	0	30	1	0	42
24	0	9	0	0	2
25	0	62	0	1	12
26	0	86	0	0	299
27	0	14	0	0	11
28	0	9	0	0	0
29	0	133	0	1	11
30	0	0	0	1	3
31	0	0	0	0	71
32	0	29	0	0	61
33	0	64	0	0	1729
34	0	17	0	0	73
35	0	36	0	1	20
36	0	106	0	1	23
37	0	31	0	1	78
38	0	35	0	1	12
39	0	100	0	1	56
40	0	103	0	1	109
41	0	0	0	0	49
42	0	7	0	0	110
43	0	31	1	0	141
44	0	35	1	0	320
45	0	115	0	1	1336
46	0	0	0	1	39
47	0	0	0	1	0
48	0	46	0	0	23
49	0	0	0	0	30

	#followers	#follows
0	788	764
1	252	483
2	1818	7486
3	148	813
4	481	592
5	773	3639
6	3	3639
7	63	19
8	682	436
9	1054	1046
10	50	98
11	10	202
12	52	240

13	1864	692
14	275	2039
15	98	847
16	11	46
17	2686	6570
18	18	64
19	1628	1037
20	1730	1298
21	183	275
22	645	4452
23	1258	952
24	629	485
25	1270	951
26	1669	1133
27	753	853
28	213	700
29	28	169
30	1395	794
31	831	1024
32	680	566
33	6114	5758
34	2104	7091
35	728	4139
36	83	458
37	2035	1035
38	11549	712
39	39	190
40	1053	6221
41	412	520
42	317	334
43	2490	1043
44	2345	861
45	1018	1208
46	37	611
47	513	633
48	83	306
49	126	372

6 Making the Prediction

In part 2, we have compared the different classifiers and found that the Random Forest Classifier had the highest accuracy at 92.5%. Therefore, we are going to use this classifier to make the prediction.

[104]:
```
rfc = RandomForestClassifier()
```

```
# Train the model
# We've done this in Part 2 but I'm redoing it here for coherence
rfc_model = rfc.fit(train_X, train_Y)
```

```
/Users/athiyadeviyani/miniconda3/lib/python3.7/site-
packages/ipykernel_launcher.py:5: DataConversionWarning: A column-vector y was
passed when a 1d array was expected. Please change the shape of y to
(n_samples,), for example using ravel().
  """
```

[105]:
```
rfc_labels = rfc_model.predict(test_data)
rfc_labels
```

[105]:
```
array([0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1,
       0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 0, 0, 0])
```

Calculate the number of fake accounts in the random sample of 50 followers

[106]:
```
no_fakes = len([x for x in rfc_labels if x==1])
```

Calculate the Instagram user's authenticity, where authenticity = (#followers - #fakes)*100 / #followers

[110]:
```
authenticity = (len(random_followers) - no_fakes) * 100 / len(random_followers)
print("User X's Instagram Followers is " + str(authenticity) + "% authentic.")
```

```
User X's Instagram Followers is 82.0% authentic.
```

7 Extension - Fake Likes

The method above can also be extended to check fake likes within a post.

Get the user's posts

[120]:
```
def get_user_posts(userID, min_posts_to_be_retrieved):
    # Retrieve all posts from my profile
    my_posts = []
    has_more_posts = True
    max_id = ''

    while has_more_posts:
        feed = api.user_feed(userID, max_id=max_id)
        if feed.get('more_available') is not True:
            has_more_posts = False

        max_id = feed.get('next_max_id', '')
        my_posts.extend(feed.get('items'))
```

```
        # time.sleep(2) to avoid flooding

        if len(my_posts) > min_posts_to_be_retrieved:
            print('Total posts retrieved: ' + str(len(my_posts)))
            return my_posts

        if has_more_posts:
            print(str(len(my_posts)) + ' posts retrieved so far...')

    print('Total posts retrieved: ' + str(len(my_posts)))

    return my_posts
```

[121]:
```
posts = get_user_posts(userID, 10)
```

```
Total posts retrieved: 18
```

Pick one post to analyse (here I'm just going to pick by random)

[122]:
```
random_post = random.sample(posts, 1)
```

Get post likers

[126]:
```
random_post[0].keys()
```

[126]:
```
dict_keys(['taken_at', 'pk', 'id', 'device_timestamp', 'media_type', 'code',
'client_cache_key', 'filter_type', 'carousel_media_count', 'carousel_media',
'can_see_insights_as_brand', 'location', 'lat', 'lng', 'user',
'can_viewer_reshare', 'caption_is_edited', 'comment_likes_enabled',
'comment_threading_enabled', 'has_more_comments', 'next_max_id',
'max_num_visible_preview_comments', 'preview_comments',
'can_view_more_preview_comments', 'comment_count',
'inline_composer_display_condition', 'inline_composer_imp_trigger_time',
'like_count', 'has_liked', 'top_likers', 'photo_of_you', 'usertags', 'caption',
'can_viewer_save', 'organic_tracking_token'])
```

[127]:
```
likers = api.media_likers(random_post[0]['id'])
```

Get a list of usernames

[130]:
```
likers_usernames = [liker['username'] for liker in likers['users']]
```

Get a random sample of 50 users

[132]:
```
random_likers = random.sample(likers_usernames, 50)
```

Retrieve the information for the 50 users

[135]:
```
l_infos = []

for liker in random_likers:
    info = api.user_info(get_ID(liker))['user']
    l_infos.append(info)
```

[137]:
```
l_table = []

for info in l_infos:
    l_table.append(get_data(info))

l_table
```

[137]:
```
[[1, 0.0, 1, 0, 0, 30, 0, 0, 6, 21, 177],
 [1, 0.0, 1, 0.0, 0, 69, 0, 1, 131, 942, 1229],
 [1, 0.0, 2, 0.0, 0, 83, 0, 1, 609, 1558, 2925],
 [1, 0.0, 1, 0.0, 0, 39, 0, 0, 851, 2940, 1255],
 [1, 0.0, 1, 0.0, 0, 36, 1, 0, 106, 1626, 1050],
 [0, 0.0, 1, 0, 0, 0, 0, 1, 7, 371, 350],
 [1, 0.0, 2, 0.0, 0, 96, 1, 0, 405, 1656, 2843],
 [1, 0.0, 2, 0.0, 0, 5, 1, 0, 9, 1363, 854],
 [1, 0.0, 1, 0, 0, 1, 0, 1, 5, 433, 371],
 [1, 0.0, 6, 0.0, 0, 93, 1, 0, 73, 1356, 1081],
 [1, 0.0, 3, 0.0, 0, 80, 1, 1, 188, 966, 966],
 [1, 0.0, 3, 0.0, 0, 0, 0, 1, 156, 1401, 1249],
 [1, 0.0, 2, 0.0, 0, 118, 1, 0, 115, 6557, 2423],
 [1, 0.0, 1, 0.0, 0, 12, 0, 0, 84, 1552, 661],
 [1, 0.0, 1, 0.0, 0, 80, 0, 0, 99, 1413, 2479],
 [1, 0.0, 1, 0.0, 0, 23, 0, 1, 12, 1116, 1031],
 [1, 0.0, 1, 0.0, 0, 20, 0, 0, 87, 1864, 692],
 [1, 0.0, 3, 0.0, 0, 62, 1, 0, 17, 1266, 1107],
 [1, 0.0, 2, 0.0, 0, 20, 0, 1, 15, 636, 579],
 [1, 0.0, 4, 0.0, 0, 17, 0, 1, 127, 546, 536],
 [1, 0.0, 1, 0.0, 0, 18, 0, 0, 5, 918, 678],
 [1, 0.2857142857142857, 1, 0.0, 0, 0, 0, 1, 0, 20, 35],
 [1, 0.0, 2, 0.0, 0, 8, 0, 0, 39, 1490, 1321],
 [1, 0.0, 2, 0.0, 0, 0, 0, 0, 10, 519, 547],
 [1, 0.0, 2, 0.0, 0, 0, 0, 0, 43, 933, 1101],
 [1, 0.0, 2, 0.0, 0, 10, 0, 1, 19, 613, 612],
 [1, 0.25, 3, 0.0, 0, 139, 1, 0, 104, 1738, 999],
 [1, 0.0, 3, 0.0, 0, 42, 1, 0, 17, 2973, 1339],
 [1, 0.0, 1, 0.0, 0, 20, 0, 1, 107, 749, 857],
 [1, 0.0, 4, 0.0, 0, 119, 1, 0, 655, 675, 1904],
 [1, 0.0, 1, 0.0, 0, 103, 1, 0, 48, 10075, 2379],
 [1, 0.0, 1, 0.0, 0, 0, 0, 0, 12, 534, 563],
 [1, 0.0, 1, 0, 0, 0, 0, 1, 58, 2220, 1418],
 [1, 0.0, 1, 0.0, 0, 11, 1, 1, 18, 775, 514],
```

```
 [1, 0.0, 3, 0.0, 0, 30, 0, 0, 10, 1070, 1364],
 [1, 0.0, 1, 0.0, 0, 18, 0, 0, 108, 1148, 832],
 [1, 0.0, 2, 0.0, 0, 133, 0, 1, 52, 394, 432],
 [1, 0.0, 1, 0, 0, 30, 1, 0, 48, 3441, 1293],
 [1, 0.0, 2, 0.0, 0, 40, 1, 0, 1434, 1642, 1684],
 [1, 0.0, 1, 0.0, 0, 64, 1, 0, 33, 17955, 781],
 [1, 0.0, 2, 0.0, 0, 91, 1, 1, 217, 1014, 1409],
 [1, 0.0, 1, 0, 0, 0, 0, 1, 1, 1347, 872],
 [1, 0.3076923076923077, 1, 0.0, 0, 0, 0, 0, 59, 161, 544],
 [1, 0.0, 3, 0.0, 0, 141, 1, 1, 274, 922, 913],
 [1, 0.0, 1, 0.0, 0, 69, 1, 0, 69, 904, 596],
 [1, 0.0, 1, 0.0, 0, 42, 0, 0, 598, 1877, 6379],
 [1, 0.0, 2, 0.0, 0, 4, 0, 1, 11, 660, 643],
 [1, 0.0, 2, 0.0, 0, 24, 0, 0, 6, 345, 358],
 [1, 0.0, 2, 0.0, 0, 29, 0, 0, 23, 293, 538],
 [1, 0.0, 1, 0.0, 0, 10, 1, 1, 3, 690, 549]]
```

[138]:
```
# Generate pandas dataframe
l_test_data = pd.DataFrame(l_table,
                           columns = ['profile pic',
                                      'nums/length username',
                                      'fullname words',
                                      'nums/length fullname',
                                      'name==username',
                                      'description length',
                                      'external URL',
                                      'private',
                                      '#posts',
                                      '#followers',
                                      '#follows'])
l_test_data
```

[138]:

	profile pic	nums/length username	fullname words	nums/length fullname \
0	1	0.000000	1	0.0
1	1	0.000000	1	0.0
2	1	0.000000	2	0.0
3	1	0.000000	1	0.0
4	1	0.000000	1	0.0
5	0	0.000000	1	0.0
6	1	0.000000	2	0.0
7	1	0.000000	2	0.0
8	1	0.000000	1	0.0
9	1	0.000000	6	0.0
10	1	0.000000	3	0.0
11	1	0.000000	3	0.0
12	1	0.000000	2	0.0
13	1	0.000000	1	0.0

14	1	0.000000	1	0.0
15	1	0.000000	1	0.0
16	1	0.000000	1	0.0
17	1	0.000000	3	0.0
18	1	0.000000	2	0.0
19	1	0.000000	4	0.0
20	1	0.000000	1	0.0
21	1	0.285714	1	0.0
22	1	0.000000	2	0.0
23	1	0.000000	2	0.0
24	1	0.000000	2	0.0
25	1	0.000000	2	0.0
26	1	0.250000	3	0.0
27	1	0.000000	3	0.0
28	1	0.000000	1	0.0
29	1	0.000000	4	0.0
30	1	0.000000	1	0.0
31	1	0.000000	1	0.0
32	1	0.000000	1	0.0
33	1	0.000000	1	0.0
34	1	0.000000	3	0.0
35	1	0.000000	1	0.0
36	1	0.000000	2	0.0
37	1	0.000000	1	0.0
38	1	0.000000	2	0.0
39	1	0.000000	1	0.0
40	1	0.000000	2	0.0
41	1	0.000000	1	0.0
42	1	0.307692	1	0.0
43	1	0.000000	3	0.0
44	1	0.000000	1	0.0
45	1	0.000000	1	0.0
46	1	0.000000	2	0.0
47	1	0.000000	2	0.0
48	1	0.000000	2	0.0
49	1	0.000000	1	0.0

	name==username	description length	external URL	private	#posts	\
0	0	30	0	0	6	
1	0	69	0	1	131	
2	0	83	0	1	609	
3	0	39	0	0	851	
4	0	36	1	0	106	
5	0	0	0	1	7	
6	0	96	1	0	405	
7	0	5	1	0	9	
8	0	1	0	1	5	

9	0	93	1	0	73
10	0	80	1	1	188
11	0	0	0	1	156
12	0	118	1	0	115
13	0	12	0	0	84
14	0	80	0	0	99
15	0	23	0	1	12
16	0	20	0	0	87
17	0	62	1	0	17
18	0	20	0	1	15
19	0	17	0	1	127
20	0	18	0	0	5
21	0	0	0	1	0
22	0	8	0	0	39
23	0	0	0	0	10
24	0	0	0	0	43
25	0	10	0	1	19
26	0	139	1	0	104
27	0	42	1	0	17
28	0	20	0	1	107
29	0	119	1	0	655
30	0	103	1	0	48
31	0	0	0	0	12
32	0	0	0	1	58
33	0	11	1	1	18
34	0	30	0	0	10
35	0	18	0	0	108
36	0	133	0	1	52
37	0	30	1	0	48
38	0	40	1	0	1434
39	0	64	1	0	33
40	0	91	1	1	217
41	0	0	0	1	1
42	0	0	0	0	59
43	0	141	1	1	274
44	0	69	1	0	69
45	0	42	0	0	598
46	0	4	0	1	11
47	0	24	0	0	6
48	0	29	0	0	23
49	0	10	1	1	3

	#followers	#follows
0	21	177
1	942	1229
2	1558	2925
3	2940	1255

4	1626	1050
5	371	350
6	1656	2843
7	1363	854
8	433	371
9	1356	1081
10	966	966
11	1401	1249
12	6557	2423
13	1552	661
14	1413	2479
15	1116	1031
16	1864	692
17	1266	1107
18	636	579
19	546	536
20	918	678
21	20	35
22	1490	1321
23	519	547
24	933	1101
25	613	612
26	1738	999
27	2973	1339
28	749	857
29	675	1904
30	10075	2379
31	534	563
32	2220	1418
33	775	514
34	1070	1364
35	1148	832
36	394	432
37	3441	1293
38	1642	1684
39	17955	781
40	1014	1409
41	1347	872
42	161	544
43	922	913
44	904	596
45	1877	6379
46	660	643
47	345	358
48	293	538
49	690	549

Finally, make the prediction!

[139]:
```
rfc = RandomForestClassifier()
rfc_model = rfc.fit(train_X, train_Y)
rfc_labels_likes = rfc_model.predict(l_test_data)
rfc_labels_likes
```

```
/Users/athiyadeviyani/miniconda3/lib/python3.7/site-
packages/ipykernel_launcher.py:2: DataConversionWarning: A column-vector y was
passed when a 1d array was expected. Please change the shape of y to
(n_samples,), for example using ravel().
```

[139]:
```
array([1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 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, 1, 0,
       0, 0, 0, 0, 0, 0])
```

Calculate the fake accounts that liked the user's media

[140]:
```
no_fake_likes = len([x for x in rfc_labels_likes if x==1])
```

Calculate the media likes authenticity

[143]:
```
media_authenticity = (len(random_likers) - no_fake_likes) * 100 /␣
 ↪len(random_likers)
print("The media with the ID:XXXXX has " + str(media_authenticity) + "%␣
 ↪authentic likes.")
```

```
The media with the ID:XXXXX has 92.0% authentic likes.
```

8 Comparison With Another User

I have specifically chosen user X because I trusted their social media 'game' and seemed to have a loyal and engaged following. Let's compare their metrics with a user Y, a user that has a noticable follower growth spike when examined on SocialBlade.

I am going to skip the explanation here because it's just a repetition of the steps performed on user X.

[144]:
```
# Re-login because of API call limits
api = login()
```

```
username: ins.tafakebusters
password: ········
```

[145]:
```
userID_y = get_ID('<USERNAME>')
```

[146]:
```
rank = api.generate_uuid()
```

USER Y FOLLOWERS ANALYSIS

[147]:
```
y_followers = get_followers(userID_y, rank)
```

[162]:
```
y_random_followers = random.sample(y_followers, 50)
```

[164]:
```
y_infos = []

for follower in y_random_followers:
    info = api.user_info(get_ID(follower))['user']
    y_infos.append(info)
```

[165]:
```
y_table = []

for info in y_infos:
    y_table.append(get_data(info))

y_table
```

[165]:
```
[[1, 0.14285714285714285, 1, 0.0, 0, 0, 0, 0, 16, 32, 1549],
 [1, 0.2222222222222222, 1, 0.0, 0, 0, 0, 1, 15, 337, 2058],
 [1, 0.25, 2, 0.0, 0, 0, 0, 0, 5, 310, 6343],
 [1, 0.0, 4, 0.0, 0, 97, 0, 0, 1, 14107, 7514],
 [1, 0.36363636363636365, 2, 0.0, 0, 0, 0, 0, 16, 8, 1050],
 [1, 0.25, 2, 0.0, 0, 13, 0, 0, 15, 87, 6741],
 [1, 0.0, 1, 0, 0, 0, 0, 1, 21, 24, 5862],
 [1, 0.0, 1, 0, 0, 13, 0, 1, 27, 1289, 689],
 [1, 0.0, 1, 0.0, 0, 29, 0, 1, 0, 31, 148],
 [1, 0.0, 1, 0, 0, 119, 0, 0, 32, 636, 1293],
 [1, 0.0, 4, 0.0, 0, 20, 0, 0, 144, 3617, 1346],
 [1, 0.21428571428571427, 2, 0.0, 0, 0, 0, 0, 17, 71, 7495],
 [1, 0.13333333333333333, 2, 0.0, 0, 113, 0, 1, 3, 305, 303],
 [0, 0.4444444444444444, 2, 0.0, 0, 0, 0, 1, 1, 63, 283],
 [1, 0.0, 3, 0.0, 0, 0, 0, 0, 17, 115, 7506],
 [0, 0.0625, 2, 0.0, 0, 0, 0, 1, 272, 1446, 2362],
 [1, 0.15384615384615385, 2, 0.0, 0, 0, 0, 0, 6, 1150, 732],
 [1, 0.0, 2, 0.0, 0, 0, 0, 0, 15, 60, 1631],
 [1, 0.0, 1, 0, 0, 13, 0, 0, 15, 11, 221],
 [1, 0.0, 1, 0, 0, 1, 0, 1, 0, 21, 23],
 [1, 0.23076923076923078, 1, 0, 0, 0, 0, 0, 0, 4, 173],
 [1, 0.25, 1, 0.0, 0, 20, 0, 0, 1, 29, 457],
 [1, 0.5, 1, 0.0, 0, 0, 0, 0, 1, 831, 5424],
 [1, 0.0, 3, 0.0, 0, 150, 1, 0, 158, 7063, 1355],
 [1, 0.0, 1, 0.0, 0, 0, 0, 1, 15, 39, 2045],
 [1, 0.0, 4, 0.05555555555555555, 0, 127, 0, 0, 196, 486, 198],
 [1, 0.0, 1, 0.0, 0, 76, 0, 1, 7, 509, 372],
 [1, 0.0, 2, 0.0, 0, 48, 0, 0, 1, 5079, 879],
 [1, 0.0, 1, 0.0, 0, 19, 0, 1, 9, 1778, 1477],
```

```
 [1, 0.0, 2, 0.0, 0, 0, 0, 0, 15, 29, 543],
 [1, 0.0, 3, 0.0, 0, 77, 0, 1, 784, 526, 1235],
 [1, 0.0, 2, 0.0, 0, 81, 1, 0, 3, 9123, 6144],
 [1, 0.0, 2, 0.0, 0, 33, 0, 0, 15, 134, 416],
 [1, 0.0, 2, 0.0, 0, 79, 0, 1, 38, 506, 804],
 [1, 0.0, 2, 0.0, 0, 0, 0, 0, 20, 27, 2557],
 [1, 0.125, 2, 0.0, 0, 0, 0, 0, 15, 9, 1151],
 [1, 0.42105263157894735, 2, 0.0, 0, 0, 0, 0, 18, 12, 1212],
 [1, 0.0, 1, 0.0, 0, 0, 0, 0, 15, 14, 600],
 [1, 0.0, 5, 0.0, 0, 25, 0, 0, 12, 1224, 774],
 [1, 0.0, 1, 0.0, 0, 0, 0, 0, 15, 23, 2056],
 [1, 0.42857142857142855, 1, 0.0, 0, 0, 0, 0, 18, 27, 395],
 [1, 0.0, 2, 0.0, 0, 0, 0, 1, 10, 444, 1116],
 [1, 0.0, 1, 0.0, 0, 43, 0, 0, 57, 214, 2377],
 [1, 0.047619047619047616, 2, 0.0, 0, 0, 0, 1, 15, 15, 6047],
 [1, 0.05263157894736842, 2, 0.0, 0, 1, 0, 0, 15, 55, 5313],
 [1, 0.18181818181818182, 2, 0.0, 0, 0, 0, 0, 16, 95, 1228],
 [1, 0.15384615384615385, 1, 0.0, 0, 0, 0, 0, 16, 56, 3665],
 [1, 0.0, 1, 0, 0, 0, 0, 0, 15, 5, 1568],
 [0, 0.16666666666666666, 2, 0.0, 0, 0, 0, 1, 3, 8, 28],
 [1, 0.4117647058823529, 2, 0.0, 0, 0, 0, 0, 1, 69, 196]]
```

[166]:
```python
# Generate pandas dataframe
y_test_data = pd.DataFrame(y_table,
                           columns = ['profile pic',
                                      'nums/length username',
                                      'fullname words',
                                      'nums/length fullname',
                                      'name==username',
                                      'description length',
                                      'external URL',
                                      'private',
                                      '#posts',
                                      '#followers',
                                      '#follows'])
y_test_data
```

[166]:
```
    profile pic  nums/length username  fullname words  nums/length fullname  \
0             1              0.142857               1              0.000000
1             1              0.222222               1              0.000000
2             1              0.250000               2              0.000000
3             1              0.000000               4              0.000000
4             1              0.363636               2              0.000000
5             1              0.250000               2              0.000000
6             1              0.000000               1              0.000000
7             1              0.000000               1              0.000000
8             1              0.000000               1              0.000000
```

9	1	0.000000	1	0.000000
10	1	0.000000	4	0.000000
11	1	0.214286	2	0.000000
12	1	0.133333	2	0.000000
13	0	0.444444	2	0.000000
14	1	0.000000	3	0.000000
15	0	0.062500	2	0.000000
16	1	0.153846	2	0.000000
17	1	0.000000	2	0.000000
18	1	0.000000	1	0.000000
19	1	0.000000	1	0.000000
20	1	0.230769	1	0.000000
21	1	0.250000	1	0.000000
22	1	0.500000	1	0.000000
23	1	0.000000	3	0.000000
24	1	0.000000	1	0.000000
25	1	0.000000	4	0.055556
26	1	0.000000	1	0.000000
27	1	0.000000	2	0.000000
28	1	0.000000	1	0.000000
29	1	0.000000	2	0.000000
30	1	0.000000	3	0.000000
31	1	0.000000	2	0.000000
32	1	0.000000	2	0.000000
33	1	0.000000	2	0.000000
34	1	0.000000	2	0.000000
35	1	0.125000	2	0.000000
36	1	0.421053	2	0.000000
37	1	0.000000	1	0.000000
38	1	0.000000	5	0.000000
39	1	0.000000	1	0.000000
40	1	0.428571	1	0.000000
41	1	0.000000	2	0.000000
42	1	0.000000	1	0.000000
43	1	0.047619	2	0.000000
44	1	0.052632	2	0.000000
45	1	0.181818	2	0.000000
46	1	0.153846	1	0.000000
47	1	0.000000	1	0.000000
48	0	0.166667	2	0.000000
49	1	0.411765	2	0.000000

	name==username	description length	external URL	private	#posts	\
0	0	0	0	0	16	
1	0	0	0	1	15	
2	0	0	0	0	5	
3	0	97	0	0	1	

4	0	0	0	0	16
5	0	13	0	0	15
6	0	0	0	1	21
7	0	13	0	1	27
8	0	29	0	1	0
9	0	119	0	0	32
10	0	20	0	0	144
11	0	0	0	0	17
12	0	113	0	1	3
13	0	0	0	1	1
14	0	0	0	0	17
15	0	0	0	1	272
16	0	0	0	0	6
17	0	0	0	0	15
18	0	13	0	0	15
19	0	1	0	1	0
20	0	0	0	0	0
21	0	20	0	0	1
22	0	0	0	0	1
23	0	150	1	0	158
24	0	0	0	1	15
25	0	127	0	0	196
26	0	76	0	1	7
27	0	48	0	0	1
28	0	19	0	1	9
29	0	0	0	0	15
30	0	77	0	1	784
31	0	81	1	0	3
32	0	33	0	0	15
33	0	79	0	1	38
34	0	0	0	0	20
35	0	0	0	0	15
36	0	0	0	0	18
37	0	0	0	0	15
38	0	25	0	0	12
39	0	0	0	0	15
40	0	0	0	0	18
41	0	0	0	1	10
42	0	43	0	0	57
43	0	0	0	1	15
44	0	1	0	0	15
45	0	0	0	0	16
46	0	0	0	0	16
47	0	0	0	0	15
48	0	0	0	1	3
49	0	0	0	0	1

	#followers	#follows
0	32	1549
1	337	2058
2	310	6343
3	14107	7514
4	8	1050
5	87	6741
6	24	5862
7	1289	689
8	31	148
9	636	1293
10	3617	1346
11	71	7495
12	305	303
13	63	283
14	115	7506
15	1446	2362
16	1150	732
17	60	1631
18	11	221
19	21	23
20	4	173
21	29	457
22	831	5424
23	7063	1355
24	39	2045
25	486	198
26	509	372
27	5079	879
28	1778	1477
29	29	543
30	526	1235
31	9123	6144
32	134	416
33	506	804
34	27	2557
35	9	1151
36	12	1212
37	14	600
38	1224	774
39	23	2056
40	27	395
41	444	1116
42	214	2377
43	15	6047
44	55	5313
45	95	1228

```
46             56      3665
47              5      1568
48              8        28
49             69       196
```

[167]:
```
# Predict (no retraining!)
rfc_labels_y = rfc_model.predict(y_test_data)
rfc_labels_y
```

[167]:
```
array([1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1,
       1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1,
       1, 1, 1, 1, 1, 1])
```

[168]:
```
# Calculate the number of fake accounts in the random sample of 50 followers
no_fakes_y = len([x for x in rfc_labels_y if x==1])
```

[216]:
```
# Calculate the authenticity
y_authenticity = (len(y_random_followers) - no_fakes_y) * 100 /␣
 ↪len(y_random_followers)
print("User Y's Instagram Followers is " + str(y_authenticity) + "% authentic.")
```

```
User Y's Instagram Followers is 38.0% authentic.
```

Ahh, the joys of being right!

USER Y LIKES ANALYSIS

[191]:
```
y_posts = get_user_posts(userID_y, 10)
```

```
Total posts retrieved: 18
```

[192]:
```
y_random_post = random.sample(y_posts, 1)
```

[193]:
```
y_likers = api.media_likers(y_random_post[0]['id'])
```

[194]:
```
y_likers_usernames = [liker['username'] for liker in y_likers['users']]
```

[207]:
```
y_random_likers = random.sample(y_likers_usernames, 50)
```

[210]:
```
y_likers_infos = []

for liker in y_random_likers:
    info = api.user_info(get_ID(liker))['user']
    y_likers_infos.append(info)
```

[211]:
```
y_likers_table = []

for info in y_likers_infos:
```

```
    y_likers_table.append(get_data(info))

y_likers_table
```

[211]:
```
[[1, 0.0, 2, 0.0, 0, 0, 0, 0, 2, 897, 830],
 [0, 0.0, 2, 0.0, 0, 0, 0, 1, 0, 129, 132],
 [1, 0.0, 2, 0.0, 0, 8, 0, 1, 72, 1157, 698],
 [1, 0.0, 1, 0, 0, 10, 0, 1, 6, 1410, 619],
 [1, 0.0, 1, 0.0, 0, 0, 0, 0, 0, 1916, 731],
 [1, 0.2222222222222222, 3, 0.0, 0, 72, 0, 1, 13, 950, 649],
 [1, 0.0, 1, 0.0, 0, 19, 0, 1, 17, 1543, 1289],
 [1, 0.2, 5, 0.0, 0, 11, 0, 0, 33, 1076, 606],
 [1, 0.0, 1, 0.0, 0, 104, 0, 1, 6, 202, 485],
 [1, 0.2, 1, 0.0, 0, 15, 0, 0, 7, 1262, 679],
 [1, 0.15384615384615385, 2, 0.0, 0, 0, 0, 0, 6, 1150, 732],
 [1, 0.0, 1, 0.0, 0, 17, 1, 0, 28, 2442, 629],
 [1, 0.0, 2, 0.0, 0, 61, 0, 0, 159, 556, 765],
 [1, 0.0, 2, 0.0, 0, 34, 0, 1, 10, 531, 526],
 [1, 0.0, 3, 0.0, 0, 127, 0, 0, 23, 1137, 909],
 [1, 0.0, 2, 0.0, 0, 66, 0, 1, 25, 583, 805],
 [1, 0.13333333333333333, 2, 0.0, 0, 67, 1, 0, 141, 4615, 1948],
 [1, 0.0, 2, 0.0, 0, 47, 0, 1, 387, 75, 162],
 [1, 0.0, 1, 0.0, 0, 142, 0, 1, 8144, 664, 1527],
 [1, 0.0, 3, 0.0, 0, 4, 0, 1, 1, 466, 325],
 [1, 0.058823529411764705, 1, 0.0, 0, 32, 0, 0, 14, 419, 414],
 [1, 0.0, 3, 0.0, 0, 75, 1, 0, 353, 1399, 764],
 [1, 0.0, 1, 0, 0, 0, 0, 0, 9, 611, 554],
 [1, 0.0, 1, 0.0, 0, 29, 0, 1, 3, 2064, 1077],
 [1, 0.0, 1, 0.0, 0, 26, 0, 1, 37, 628, 714],
 [1, 0.0, 2, 0.0, 0, 89, 1, 1, 243, 2316, 1030],
 [1, 0.0, 2, 0.0, 0, 140, 1, 0, 666, 4460, 492],
 [1, 0.0, 2, 0.0, 0, 20, 0, 0, 71, 4101, 878],
 [1, 0.0, 2, 0.0, 0, 5, 0, 0, 148, 424, 716],
 [1, 0.0, 1, 0, 0, 0, 0, 1, 2, 640, 730],
 [1, 0.0, 2, 0.0, 0, 64, 0, 1, 8, 1141, 891],
 [1, 0.0, 3, 0.0, 0, 29, 0, 1, 10, 1378, 986],
 [1, 0.0, 2, 0.0, 0, 14, 0, 1, 3, 994, 698],
 [1, 0.0, 1, 0.0, 0, 29, 0, 1, 43, 181, 169],
 [1, 0.0, 1, 0.0, 0, 58, 1, 0, 24, 1144, 1091],
 [1, 0.0, 2, 0.0, 0, 25, 0, 1, 36, 687, 574],
 [1, 0.0, 3, 0.0, 0, 8, 0, 1, 33, 1846, 996],
 [1, 0.5714285714285714, 2, 0.0, 0, 18, 0, 1, 202, 1180, 600],
 [1, 0.0, 2, 0.0, 0, 7, 0, 0, 45, 1206, 676],
 [1, 0.0, 2, 0.0, 0, 76, 0, 0, 12, 661, 3004],
 [1, 0.0, 1, 0.0, 0, 9, 0, 1, 5, 759, 706],
 [0, 0.0, 3, 0.0, 0, 61, 0, 1, 9, 439, 612],
 [1, 0.16666666666666666, 1, 0.0, 0, 0, 0, 1, 3, 911, 822],
```

```
 [1, 0.4, 2, 0.0, 0, 82, 0, 0, 99, 556, 733],
 [1, 0.0, 2, 0.0, 0, 80, 0, 1, 21, 478, 385],
 [1, 0.0, 1, 0, 0, 0, 0, 1, 0, 653, 312],
 [1, 0.0, 1, 0.0, 0, 13, 0, 1, 40, 713, 657],
 [1, 0.0, 2, 0.0, 0, 0, 0, 1, 4, 113, 311],
 [1, 0.0, 2, 0.0, 0, 33, 0, 0, 74, 3564, 1051],
 [1, 0.0, 1, 0.0, 0, 121, 0, 0, 958, 904, 479]]
```

[212]:
```
y_likers_data = pd.DataFrame(y_likers_table,
                         columns = ['profile pic',
                                    'nums/length username',
                                    'fullname words',
                                    'nums/length fullname',
                                    'name==username',
                                    'description length',
                                    'external URL',
                                    'private',
                                    '#posts',
                                    '#followers',
                                    '#follows'])
y_likers_data
```

[212]:

	profile pic	nums/length username	fullname words	nums/length fullname \
0	1	0.000000	2	0.0
1	0	0.000000	2	0.0
2	1	0.000000	2	0.0
3	1	0.000000	1	0.0
4	1	0.000000	1	0.0
5	1	0.222222	3	0.0
6	1	0.000000	1	0.0
7	1	0.200000	5	0.0
8	1	0.000000	1	0.0
9	1	0.200000	1	0.0
10	1	0.153846	2	0.0
11	1	0.000000	1	0.0
12	1	0.000000	2	0.0
13	1	0.000000	2	0.0
14	1	0.000000	3	0.0
15	1	0.000000	2	0.0
16	1	0.133333	2	0.0
17	1	0.000000	2	0.0
18	1	0.000000	1	0.0
19	1	0.000000	3	0.0
20	1	0.058824	1	0.0
21	1	0.000000	3	0.0
22	1	0.000000	1	0.0
23	1	0.000000	1	0.0

24	1	0.000000	1	0.0
25	1	0.000000	2	0.0
26	1	0.000000	2	0.0
27	1	0.000000	2	0.0
28	1	0.000000	2	0.0
29	1	0.000000	1	0.0
30	1	0.000000	2	0.0
31	1	0.000000	3	0.0
32	1	0.000000	2	0.0
33	1	0.000000	1	0.0
34	1	0.000000	1	0.0
35	1	0.000000	2	0.0
36	1	0.000000	3	0.0
37	1	0.571429	2	0.0
38	1	0.000000	2	0.0
39	1	0.000000	2	0.0
40	1	0.000000	1	0.0
41	0	0.000000	3	0.0
42	1	0.166667	1	0.0
43	1	0.400000	2	0.0
44	1	0.000000	2	0.0
45	1	0.000000	1	0.0
46	1	0.000000	1	0.0
47	1	0.000000	2	0.0
48	1	0.000000	2	0.0
49	1	0.000000	1	0.0

	name==username	description length	external URL	private	#posts \
0	0	0	0	0	2
1	0	0	0	1	0
2	0	8	0	1	72
3	0	10	0	1	6
4	0	0	0	0	0
5	0	72	0	1	13
6	0	19	0	1	17
7	0	11	0	0	33
8	0	104	0	1	6
9	0	15	0	0	7
10	0	0	0	0	6
11	0	17	1	0	28
12	0	61	0	0	159
13	0	34	0	1	10
14	0	127	0	0	23
15	0	66	0	1	25
16	0	67	1	0	141
17	0	47	0	1	387
18	0	142	0	1	8144

19	0	4	0	1	1
20	0	32	0	0	14
21	0	75	1	0	353
22	0	0	0	0	9
23	0	29	0	1	3
24	0	26	0	1	37
25	0	89	1	1	243
26	0	140	1	0	666
27	0	20	0	0	71
28	0	5	0	0	148
29	0	0	0	1	2
30	0	64	0	1	8
31	0	29	0	1	10
32	0	14	0	1	3
33	0	29	0	1	43
34	0	58	1	0	24
35	0	25	0	1	36
36	0	8	0	1	33
37	0	18	0	1	202
38	0	7	0	0	45
39	0	76	0	0	12
40	0	9	0	1	5
41	0	61	0	1	9
42	0	0	0	1	3
43	0	82	0	0	99
44	0	80	0	1	21
45	0	0	0	1	0
46	0	13	0	1	40
47	0	0	0	1	4
48	0	33	0	0	74
49	0	121	0	0	958

	#followers	#follows
0	897	830
1	129	132
2	1157	698
3	1410	619
4	1916	731
5	950	649
6	1543	1289
7	1076	606
8	202	485
9	1262	679
10	1150	732
11	2442	629
12	556	765
13	531	526

```
14        1137       909
15         583       805
16        4615      1948
17          75       162
18         664      1527
19         466       325
20         419       414
21        1399       764
22         611       554
23        2064      1077
24         628       714
25        2316      1030
26        4460       492
27        4101       878
28         424       716
29         640       730
30        1141       891
31        1378       986
32         994       698
33         181       169
34        1144      1091
35         687       574
36        1846       996
37        1180       600
38        1206       676
39         661      3004
40         759       706
41         439       612
42         911       822
43         556       733
44         478       385
45         653       312
46         713       657
47         113       311
48        3564      1051
49         904       479
```

[213]:
```
# Predict!
y_likers_pred = rfc_model.predict(y_likers_data)
y_likers_pred
```

[213]:
```
array([0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       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])
```

[218]:
```
# Calculate the number of fake likes
no_fakes_yl = len([x for x in y_likers_pred if x==1])
```

```
# Calculate media likes authenticity
y_post_authenticity = (len(y_random_likers) - no_fakes_yl) * 100 /␣
 ↪len(y_random_likers)
print("The media with the ID:YYYYY has " + str(y_post_authenticity) + "%␣
 ↪authentic likes.")
```

```
The media with the ID:YYYYY has 96.0% authentic likes.
```

Very high likes authenticity but very low follower authenticity? How is that possible?

We can use **engagement rates** to explain this phenomena further.

Engagement rate = average number of engagements (likes+comments) / number of followers)

[220]:
```
y_posts[0].keys()
```

[220]:
```
dict_keys(['taken_at', 'pk', 'id', 'device_timestamp', 'media_type', 'code',
'client_cache_key', 'filter_type', 'carousel_media_count', 'carousel_media',
'can_see_insights_as_brand', 'location', 'lat', 'lng', 'user',
'can_viewer_reshare', 'caption_is_edited', 'comment_likes_enabled',
'comment_threading_enabled', 'has_more_comments',
'max_num_visible_preview_comments', 'preview_comments',
'can_view_more_preview_comments', 'comment_count',
'inline_composer_display_condition', 'inline_composer_imp_trigger_time',
'like_count', 'has_liked', 'top_likers', 'photo_of_you', 'caption',
'can_viewer_save', 'organic_tracking_token'])
```

[226]:
```
count = 0

for post in y_posts:
    count += post['comment_count']
    count += post['like_count']

average_engagements = count / len(y_posts)
engagement_rate = average_engagements*100 / len(y_followers)

engagement_rate
```

[226]:
```
9.50268408791654
```

This means that only roughly 9.5% of user Y's followers engage with their content.

9 Thoughts

Making sense of the result

So user X received an 82% follower authenticity score and a 92% media likes authenticity on one of their posts. Is that good enough? What about user Y with a 35% follower authenticity score and a 96% media likes authenticity?

Since this entire notebook is an exploratory analysis, there's not really a hard line between a 'good' influencer and a 'bad' influencer. For user X, we can tell that the user has authentic and loyal followers. However for user Y, we can assume that they have a rather low authentic follower score, however their likes consist of real followers. This means that user Y might have invested on buying followers, but not likes! This causes a really low engagement rate.

In fact, with a little bit more research, you can sort of establish a pattern just by observation: - High follower authenticity, high media authenticity, high engagement rate = authentic user - Low follower authenticity, high media authenticity, low engagement rate = buys followers, does not buy likes - Low follower authenticity, high media authenticity, high engagement rate = buys followers AND likes - … and so on!

So is this influencer worth investing or not?

Remember that we used a *random sample* of 50 followers out of thousands. As objective as random sampling could be, it still isn't an *absolutely complete* picture of the user's followers. However, the follower authenticity combined with the media likes authenticity still provides an insight for brands who are planning to invest on the influencer.

Personally, I feel like any number under 50% is rather suspicious, and there are other ways that you can confirm this suspicion: - Low engagement rates (engagement rate = average number of engagements (likes+comments) / number of followers) - Spikes in follower growth (uneven growth chart) - Comments (loyal followers acutally care about the user's content)

But of course, you have to be aware of tech-savvy influencers who cheats the audit system and try to avoid getting caught, such as influencers who buys 'drip-followers' - i.e. you buy followers in bulk but they arrive slowly. This method will make their follower growth seem gradual.

Conclusion

The rapid growth of technology allows anyone with a computer to create bots to follow users and like media on any platform. However, this also means that our ability to detect fake engagements should also improve!

Businesses, small or large, invest on social media influencers to reach a wider audience, especially during times of a global pandemic where everyone is constantly on their phones! Less tech-savvy and less aware ones are prone to this kind of misinformation.

For brands who rely on influencers for marketing, it is highly recommended to check out services such as SocialBlade to check user authenticity and engagement. Some services are more pricey, but is definitely worth the investment!