```
# IMPORTANT: RUN THIS CELL IN ORDER TO IMPORT YOUR KAGGLE DATA SOURCES,
# THEN FEEL FREE TO DELETE THIS CELL.
# NOTE: THIS NOTEBOOK ENVIRONMENT DIFFERS FROM KAGGLE'S PYTHON
# ENVIRONMENT SO THERE MAY BE MISSING LIBRARIES USED BY YOUR
# NOTEBOOK.
import kagglehub
free4ever1_instagram_fake_spammer_genuine_accounts_path = kagglehub.dataset_download('free4ever1/instagram-fake-spammer-genuine-accounts'

print('Data source import complete.')
```

```
Downloading from https://www.kaggle.com/api/v1/datasets/download/free4ever1/instagram-fake-spammer-genuine-accounts?dataset_version
100%|██████████| 6.81k/6.81k [00:00<00:00, 7.77MB/s]Extracting files...
Data source import complete.
```

Introduction and Project Scope

It is difficult to determine the authenticity of an Instagram account with complete certainty. However, it is possible to build a model that can predict the likelihood that an Instagram account is fake, based on certain characteristics of the account and its activity.

Some possible characteristics that could be used as input features for a fake Instagram account detection model include:

1. The number of followers the account has
2. The ratio of followers to following
3. The age of the account
4. The amount of activity on the account (e.g. number of posts, comments, likes)
5. The type of content that is posted
6. The use of hashtags
7. The presence of a profile picture and biography
8. The use of third-party apps to boost the account's activity

Using these and other relevant features, it is possible to train a machine learning model to predict the likelihood that an Instagram account is fake.However, it is important to note that building an accurate fake Instagram account detection model would likely require a large and diverse dataset of real and fake accounts, as well as careful feature engineering and model selection. It would also be important to continuously update the model as fake accounts evolve and change over time.

Importing Libraries and Datasets

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

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.layers import Dense, Activation, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.metrics import Accuracy

from sklearn import metrics
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import classification_report,accuracy_score,roc_curve,confusion_matrix

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))


import warnings

warnings.filterwarnings("ignore")


def fxn():
    warnings.warn("deprecated", DeprecationWarning)

with warnings.catch_warnings():
```

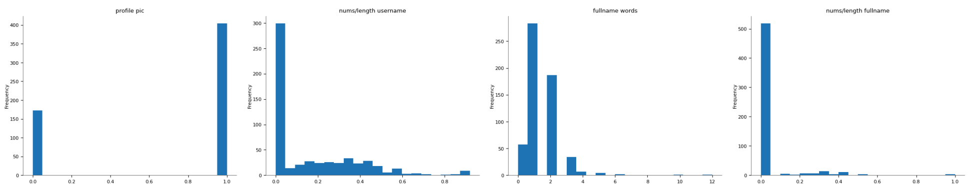
```
    warnings.filterwarnings("ignore", category=DeprecationWarning)
    fxn()
```

```
# Load the training dataset
import pandas as pd
instagram_df_train=pd.read_csv('/content/archive (2).zip')
instagram_df_train
```

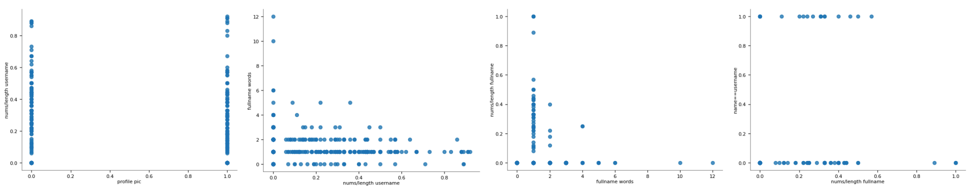
	profile pic	nums/length username	fullname words	nums/length fullname	name==username	description length	external URL	private	#posts	#followers	#follows	fa
0	1	0.27	0	0.00	0	53	0	0	32	1000	955	
1	1	0.00	2	0.00	0	44	0	0	286	2740	533	
2	1	0.10	2	0.00	0	0	0	1	13	159	98	
3	1	0.00	1	0.00	0	82	0	0	679	414	651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

576 rows × 12 columns

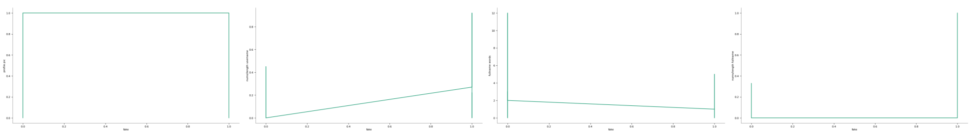
Distributions



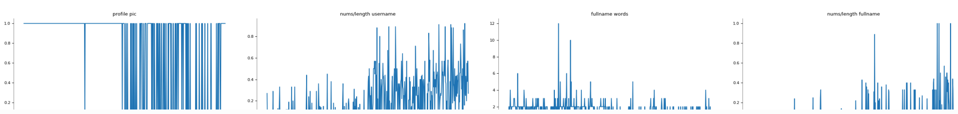
2-d distributions



Time series



Values




```
# Load the testing data
instagram_df_test=pd.read_csv('/content/test.csv')
instagram_df_test
```

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

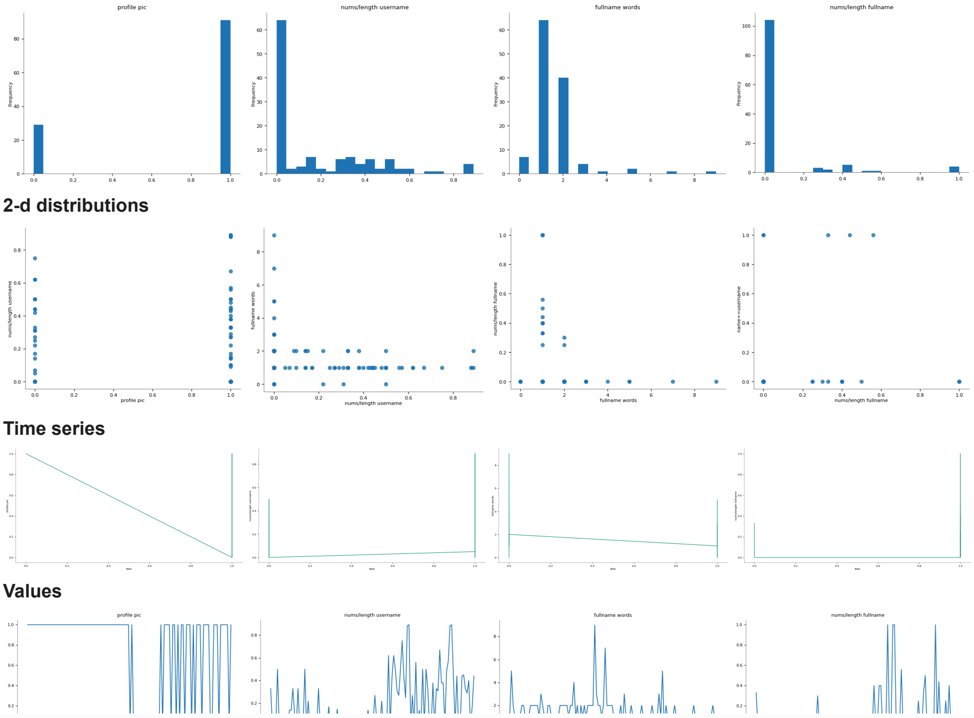
120 rows × 12 columns

Distributions

2-d distributions

Time series

Values



Start coding or generate with AI.

Statistical Analysis

```
instagram_df_train.head()
```

	profile pic	nums/length username	fullname words	nums/length fullname	name==username	description length	external URL	private	#posts	#followers	#follows	fake
0	1	0.27	0	0.0	0	53	0	0	32	1000	955	0
1	1	0.00	2	0.0	0	44	0	0	286	2740	533	0
2	1	0.10	2	0.0	0	0	0	1	13	159	98	0
3	1	0.00	1	0.0	0	82	0	0	679	414	651	0
4	1	0.00	2	0.0	0	0	0	1	6	151	126	0

```
instagram_df_train.tail()
```

	profile pic	nums/length username	fullname words	nums/length fullname	name==username	description length	external URL	private	#posts	#followers	#follows	fake
571	1	0.55	1	0.44	0	0	0	0	33	166	596	1
572	1	0.38	1	0.33	0	21	0	0	44	66	75	1
573	1	0.57	2	0.00	0	0	0	0	4	96	339	1
574	1	0.57	1	0.00	0	11	0	0	0	57	73	1

```
# Getting dataframe info
instagram_df_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
```

```
# Get the statistical summary of the dataframe
instagram_df_train.describe()
```

	profile pic	nums/length username	fullname words	nums/length fullname	name==username	description length	external URL	private	#posts	#followers
count	576.000000	576.000000	576.000000	576.000000	576.000000	576.000000	576.000000	576.000000	576.000000	5.760000e+02
mean	0.701389	0.163837	1.460069	0.036094	0.034722	22.623264	0.116319	0.381944	107.489583	8.530724e+04
std	0.458047	0.214096	1.052601	0.125121	0.183234	37.702987	0.320886	0.486285	402.034431	9.101485e+05
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000e+00
25%	0.000000	0.000000	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	3.900000e+01
50%	1.000000	0.000000	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	9.000000	1.505000e+02
75%	1.000000	0.310000	2.000000	0.000000	0.000000	34.000000	0.000000	1.000000	81.500000	7.160000e+02
max	1.000000	0.920000	12.000000	1.000000	1.000000	150.000000	1.000000	1.000000	7389.000000	1.533854e+07

```
# Checking if null values exist
instagram_df_train.isnull().sum()
```

	0
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

```
# Get the number of unique values in the "profile pic" feature
instagram_df_train['profile pic'].value_counts()
```

profile pic	count
1	404
0	172

dtype: int64

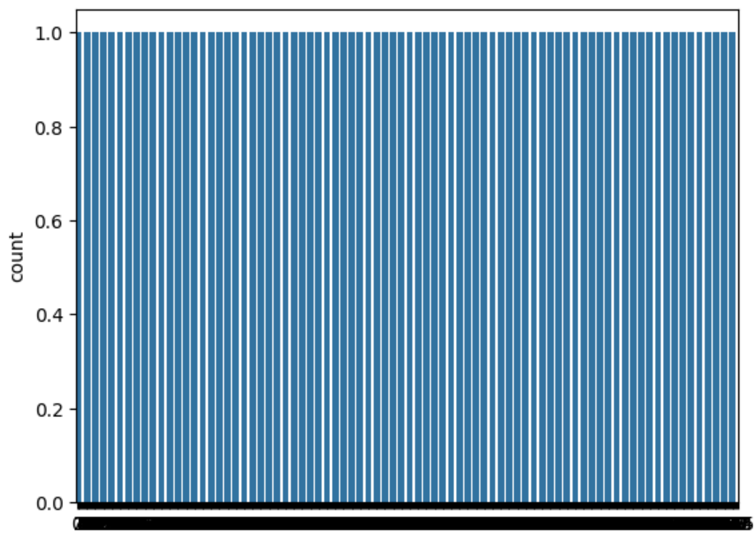
```
# Get the number of unique values in "fake" (Target column)
instagram_df_train['fake'].value_counts()
```

fake	count
0	288
1	288

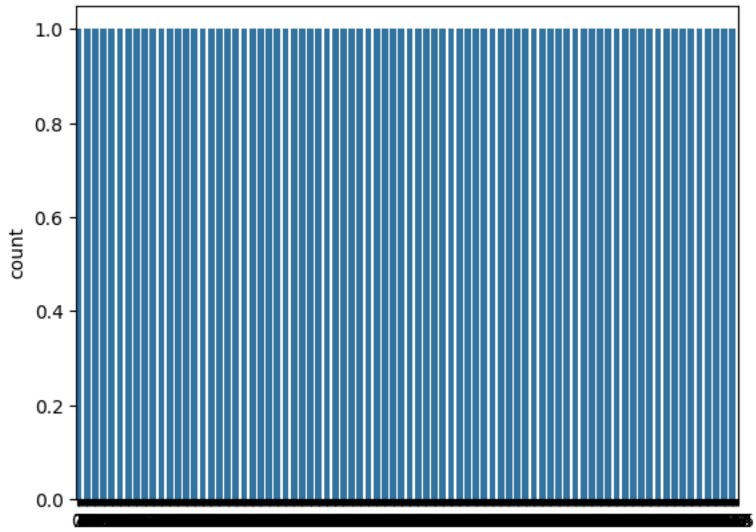
dtype: int64

Data Visualization

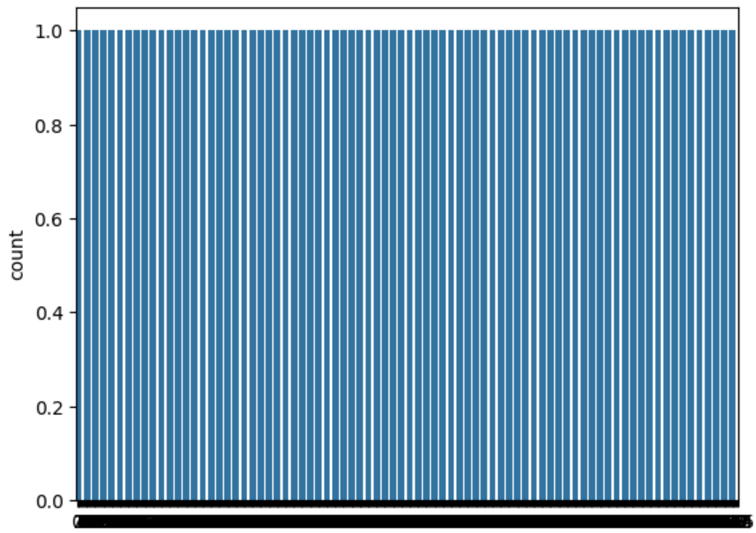
```
# Visualize the data
sns.countplot(instagram_df_train['fake'])
plt.show()
```



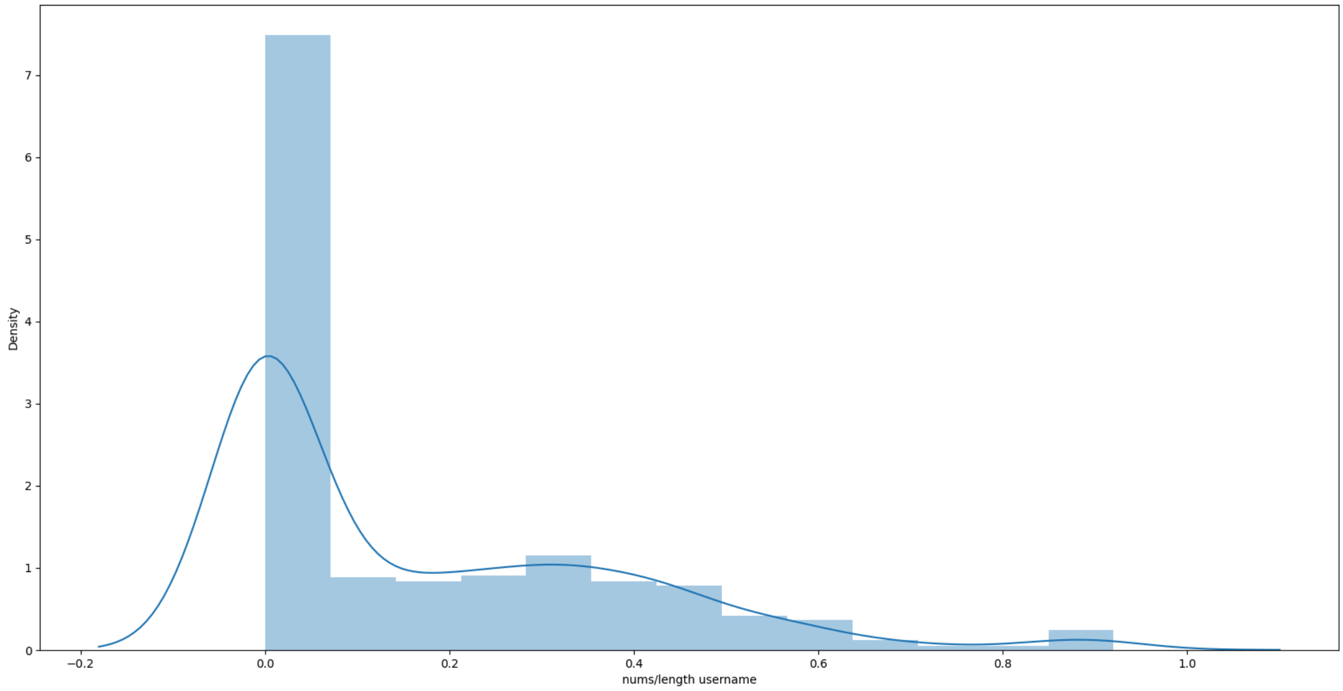

```
# Visualize the private column data
sns.countplot(instagram_df_train['private'])
plt.show()
```



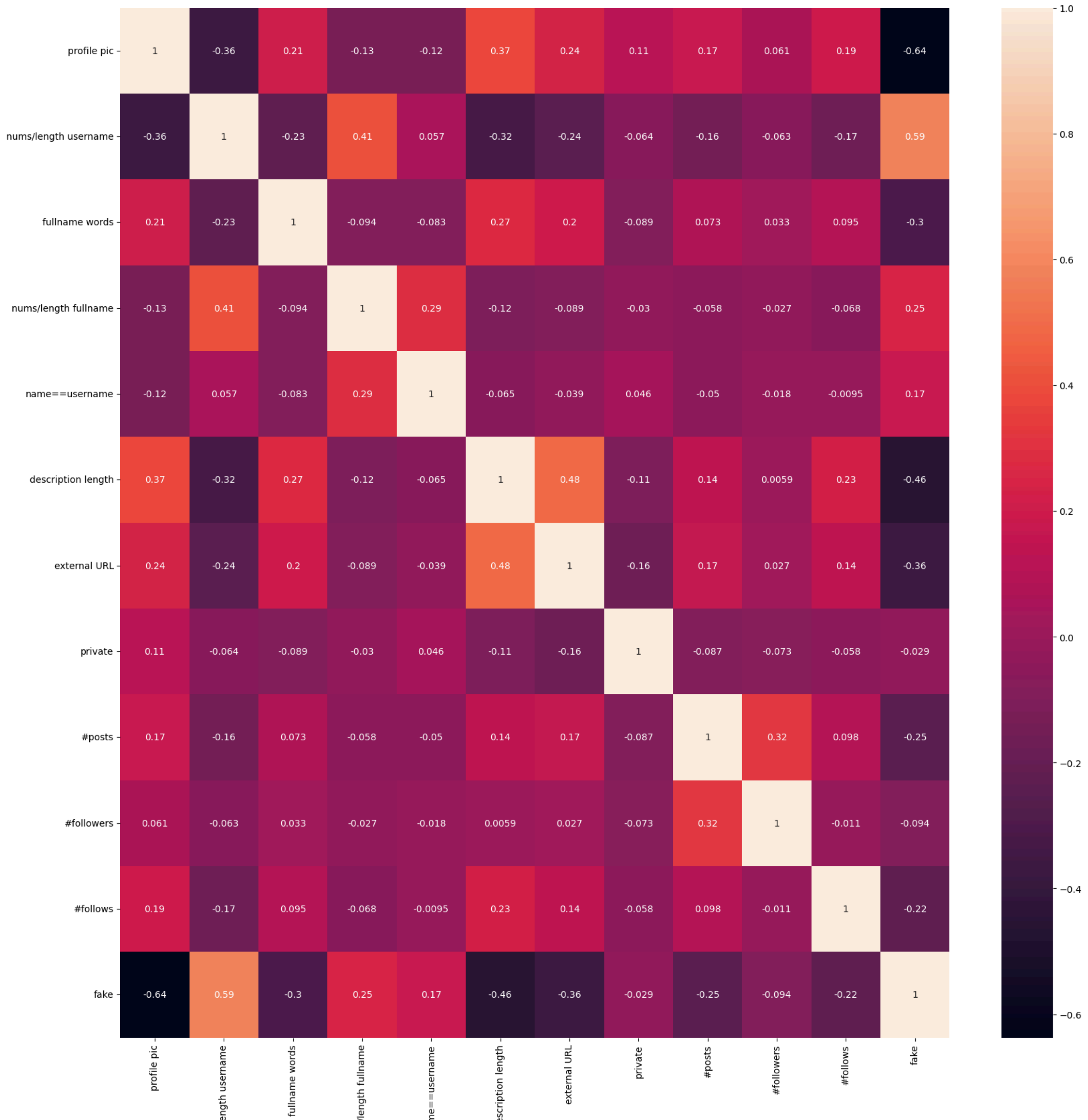

```
# Visualize the "profile pic" column data
sns.countplot(instagram_df_train['profile pic'])
plt.show()
```




```
# Visualize the data
plt.figure(figsize = (20, 10))
sns.distplot(instagram_df_train['nums/length username'])
plt.show()
```




```
# Correlation plot
plt.figure(figsize=(20, 20))
cm = instagram_df_train.corr()
ax = plt.subplot()
sns.heatmap(cm, annot = True, ax = ax)
plt.show()
```



Data Modelling

```
# Training and testing dataset (inputs)
X_train = instagram_df_train.drop(columns = ['fake'])
X_test = instagram_df_test.drop(columns = ['fake'])
X_train
```

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

```
# Training and testing dataset (Outputs)
y_train = instagram_df_train['fake']
y_test = instagram_df_test['fake']
y_train
```

	fake
0	0
1	0
2	0
3	0
4	0
...	...
571	1
572	1
573	1
574	1
575	1

576 rows × 1 columns

dtype: int64

```
# Scale the data before training the model
from sklearn.preprocessing import StandardScaler, MinMaxScaler

scaler_x = StandardScaler()
X_train = scaler_x.fit_transform(X_train)
X_test = scaler_x.transform(X_test)


y_train = tf.keras.utils.to_categorical(y_train, num_classes = 2)
y_test = tf.keras.utils.to_categorical(y_test, num_classes = 2)


y_train
```

```
array([[1., 0.],
       [1., 0.],
       [1., 0.],
       ...,
       [0., 1.],
       [0., 1.],
       [0., 1.]])
```

```
import tensorflow.keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout

model = Sequential()
model.add(Dense(50, input_dim=11, activation='relu'))
model.add(Dense(150, activation='relu'))
```

```
model.add(Dropout(0.3))
model.add(Dense(150, activation='relu'))
model.add(Dropout(0.3))
model.add(Dense(25, activation='relu'))
model.add(Dropout(0.3))
model.add(Dense(2,activation='softmax'))

model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 50)	600
dense_1 (Dense)	(None, 150)	7,650
dropout (Dropout)	(None, 150)	0
dense_2 (Dense)	(None, 150)	22,650
dropout_1 (Dropout)	(None, 150)	0
dense_3 (Dense)	(None, 25)	3,775
dropout_2 (Dropout)	(None, 25)	0
dense_4 (Dense)	(None, 2)	52

Total params: 34,727 (135.65 KB)
Trainable params: 34,727 (135.65 KB)

```
model.compile(optimizer = 'adam', loss = 'categorical_crossentropy', metrics = ['accuracy'])
```

```
epochs_hist = model.fit(X_train, y_train, epochs = 50,  verbose = 1, validation_split = 0.1)
```

```
Epoch 1/50
17/17 ━━━━━━━━━━━━━━━━━━━━ 2s 21ms/step - accuracy: 0.6434 - loss: 0.6525 - val_accuracy: 0.8103 - val_loss: 0.4312
Epoch 2/50
17/17 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.8601 - loss: 0.4304 - val_accuracy: 0.8793 - val_loss: 0.2481
Epoch 3/50
17/17 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.8893 - loss: 0.3226 - val_accuracy: 0.8966 - val_loss: 0.2166
Epoch 4/50
17/17 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.9149 - loss: 0.2464 - val_accuracy: 0.8966 - val_loss: 0.1923
Epoch 5/50
17/17 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.9120 - loss: 0.2891 - val_accuracy: 0.8793 - val_loss: 0.2195
Epoch 6/50
17/17 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.8991 - loss: 0.2383 - val_accuracy: 0.8966 - val_loss: 0.1724
Epoch 7/50
17/17 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.9198 - loss: 0.2458 - val_accuracy: 0.8793 - val_loss: 0.2005
Epoch 8/50
17/17 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.9298 - loss: 0.2249 - val_accuracy: 0.9138 - val_loss: 0.1664
Epoch 9/50
17/17 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.9214 - loss: 0.2076 - val_accuracy: 0.8793 - val_loss: 0.1852
Epoch 10/50
17/17 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.9265 - loss: 0.2060 - val_accuracy: 0.8966 - val_loss: 0.1776
Epoch 11/50
17/17 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.9070 - loss: 0.2212 - val_accuracy: 0.8793 - val_loss: 0.2204
Epoch 12/50
17/17 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.9069 - loss: 0.2581 - val_accuracy: 0.8793 - val_loss: 0.1957
Epoch 13/50
17/17 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.9208 - loss: 0.1904 - val_accuracy: 0.9138 - val_loss: 0.1750
Epoch 14/50
17/17 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9286 - loss: 0.1870 - val_accuracy: 0.8966 - val_loss: 0.1907
Epoch 15/50
17/17 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9361 - loss: 0.1923 - val_accuracy: 0.8621 - val_loss: 0.2925
Epoch 16/50
17/17 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.9240 - loss: 0.2228 - val_accuracy: 0.8966 - val_loss: 0.1899
Epoch 17/50
17/17 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.9242 - loss: 0.1786 - val_accuracy: 0.8793 - val_loss: 0.2206
Epoch 18/50
17/17 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.9230 - loss: 0.1728 - val_accuracy: 0.8966 - val_loss: 0.1731
Epoch 19/50
17/17 ━━━━━━━━━━━━━━━━━━━━ 1s 17ms/step - accuracy: 0.9223 - loss: 0.2028 - val_accuracy: 0.9138 - val_loss: 0.1900
Epoch 20/50
17/17 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step - accuracy: 0.9222 - loss: 0.1849 - val_accuracy: 0.9138 - val_loss: 0.1838
Epoch 21/50
17/17 ━━━━━━━━━━━━━━━━━━━━ 1s 19ms/step - accuracy: 0.9268 - loss: 0.1742 - val_accuracy: 0.8793 - val_loss: 0.2542
Epoch 22/50
17/17 ━━━━━━━━━━━━━━━━━━━━ 1s 14ms/step - accuracy: 0.9324 - loss: 0.1667 - val_accuracy: 0.9310 - val_loss: 0.1653
Epoch 23/50
17/17 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - accuracy: 0.9449 - loss: 0.1565 - val_accuracy: 0.9310 - val_loss: 0.1909
Epoch 24/50
17/17 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - accuracy: 0.9468 - loss: 0.1367 - val_accuracy: 0.9310 - val_loss: 0.1665
Epoch 25/50
17/17 ━━━━━━━━━━━━━━━━━━━━ 1s 21ms/step - accuracy: 0.9366 - loss: 0.1654 - val_accuracy: 0.8966 - val_loss: 0.2365
Epoch 26/50
17/17 ━━━━━━━━━━━━━━━━━━━━ 1s 19ms/step - accuracy: 0.9205 - loss: 0.1784 - val_accuracy: 0.8966 - val_loss: 0.2128
```

```
Epoch 27/50
17/17 ━━━━━━━━━━━━━━━━━━━━ 1s 22ms/step - accuracy: 0.9117 - loss: 0.1671 - val_accuracy: 0.9138 - val_loss: 0.2175
Epoch 28/50
17/17 ━━━━━━━━━━━━━━━━━━━━ 1s 30ms/step - accuracy: 0.9226 - loss: 0.1539 - val_accuracy: 0.9310 - val_loss: 0.1761
Epoch 29/50
17/17 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - accuracy: 0.9254 - loss: 0.1644 - val_accuracy: 0.9138 - val_loss: 0.1948
```

Model Validation and Results

```
print(epochs_hist.history.keys())
```

```
dict_keys(['accuracy', 'loss', 'val_accuracy', 'val_loss'])
```

```
plt.plot(epochs_hist.history['loss'])
plt.plot(epochs_hist.history['val_loss'])
```