ML MAJOR PROJECT (OCT)

By
AISHWARYA B

INDEX

S.No.			Contents	Page No.
1			Introduction	1
2			Steps for classifying the tweets based on gender	1
	2.1		Understanding and cleaning the dataset	1
	2.2		EDA and visualizing the dataset	12
	2.3		QnA on the dataset	17
	2.4		Feature Selection	18
	2.5		Classification	20
		2.5.1	SVM model	20
		2.5.2	Removing outliers	22
		2.5.3	KNN model	22
		2.5.4	Random forest classification model	24
3			Conclusion	26

1.INTRODUCTION:

Given a dataset has details of tweets posted on twitter. Our aim is to classify the tweets with respect to gender, using ensemble learning. In ensemble learning we make use of 3-4 classification algorithms and make use of each and every accuracy after evaluating each model.

2.STEPS FOR CLASSIFYING THE TWEETS BASED ON GENDER:

The following steps were followed to complete our major project:

2.1. UNDERSTANDING AND CLEANING THE DATASET

The provided dataset 'Information.csv' was imported using pandas into a jupyter notebook. The dataset has 20050 rows and 26 columns although Fig-2.1 here displays only the first three rows.

Importing the dataset

In [1]:
```
#Importing the library
import pandas as pd
```

In [2]:
```
#Reading the CSV file
df=pd.read_csv('Information.csv',encoding='latin-1')
```

In [3]:
```
#First 3 rows of the data set
df.head(3)
```

Out[3]:

	_unit_id	_golden	_unit_state	_trusted_judgments	_last_judgment_at	gender	gender:confidence	profile_yn	profile_yn:confidence	created	...	
0	815719226	False	finalized	3	10/26/15 23:24	male	1.0000	yes	1.0	12/5/13 1:48	...	https:/
1	815719227	False	finalized	3	10/26/15 23:30	male	1.0000	yes	1.0	10/1/12 13:51	...	https:/
2	815719228	False	finalized	3	10/26/15 23:33	male	0.6625	yes	1.0	11/28/14 11:30	...	https:/

Fig-2.1.1:Importing dataset into jupyter notebook

Firstly, using info () function, it was ensured that all the columns had non-null values.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20050 entries, 0 to 20049
Data columns (total 26 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   _unit_id               20050 non-null  int64
 1   _golden                20050 non-null  bool
 2   _unit_state            20050 non-null  object
 3   _trusted_judgments     20050 non-null  int64
 4   _last_judgment_at      20000 non-null  object
 5   gender                 19953 non-null  object
 6   gender:confidence      20024 non-null  float64
 7   profile_yn             20050 non-null  object
 8   profile_yn:confidence  20050 non-null  float64
 9   created                20050 non-null  object
 10  description            16306 non-null  object
 11  fav_number             20050 non-null  int64
 12  gender_gold            50 non-null     object
 13  link_color             20050 non-null  object
 14  name                   20050 non-null  object
 15  profile_yn_gold        50 non-null     object
 16  profileimage           20050 non-null  object
 17  retweet_count          20050 non-null  int64
 18  sidebar_color          20050 non-null  object
 19  text                   20050 non-null  object
 20  tweet_coord            159 non-null    object
 21  tweet_count            20050 non-null  int64
 22  tweet_created          20050 non-null  object
 23  tweet_id               20050 non-null  float64
 24  tweet_location         12566 non-null  object
 25  user_timezone          12252 non-null  object
```

Fig-2.1.2: df.info () to learn about the dataset in a nutshell

However, the columns _unit_state and _golden were similar. If _unit_state=”golden” then the corresponding value in _golden=True otherwise it was false.Hence,it’s definitely sufficient to retain one of these columns and drop the other.Thus,the _golden column was removed using drop() and the dataset had only 25 columns then.

Further, the column named tweet_coord had only NaN values in all rows. As it was useless to have a column filled with only missing values, it was also removed using drop (). The dataset then had only 24 columns.

The column named tweet_location had a considerable number of NaN values.7484 out of 20050 values were missing which turned out to be 37% of the data (Fig-2.3). Hence, they had to be fixed. As they were to be filled with the name of a place, it was decided to replace them with one such place which wasn’t already there in the dataset. Unfortunately, when Bangalore was tried first, it was observed that there were already 5 entries with location as Bangalore! (Fig-2.4)

When it was again randomly tried with 'Mysore', it was found that Mysore did not exist (Fig-2.5) before in the dataset. Hence, it was decided to replace all NaN values of the column with Mysore considering it to be more like a default location.

```
In [16]: df[df['tweet_location'].isna()].count()

         #7484 out of 20050 entries have missing(NaN) values which is 37% of data.So,we must fill them!
         #lets fill it as Bangalore.But for this shall we check if Bangalore is already there in this col??

Out[16]: _unit_id                 7484
         _unit_state              7484
         _trusted_judgments       7484
         _last_judgment_at        7466
         gender                   7442
         gender:confidence        7471
         profile_yn               7484
         profile_yn:confidence    7484
         created                  7484
         description              4441
         fav_number               7484
         gender_gold                18
         link_color               7484
         name                     7484
         profile_yn_gold            18
         profileimage             7484
         retweet_count            7484
         sidebar_color            7484
         text                     7484
         tweet_count              7484
         tweet_created            7484
         tweet_id                 7484
         tweet_location              0
         user_timezone            3048
         dtype: int64
```

Fig-2.1.3: Missing values in tweet_location

```
In [18]: #Get the count of all columns where tweet_location is 'Bangalore'
         df[df['tweet_location'] == "Bangalore"].count()

Out[18]: _unit_id                 5
         _unit_state              5
         _trusted_judgments       5
         _last_judgment_at        5
         gender                   5
         gender:confidence        5
         profile_yn               5
         profile_yn:confidence    5
         created                  5
         description              5
         fav_number               5
         gender_gold              0
         link_color               5
         name                     5
         profile_yn_gold          0
         profileimage             5
         retweet_count            5
         sidebar_color            5
         text                     5
         tweet_count              5
         tweet_created            5
         tweet_id                 5
         tweet_location           5
         user_timezone            5
         dtype: int64
```

Fig-2.1.4: Bangalore is already existing!

```
In [19]: df[df['tweet_location'] == "Mysore"].count()  #as there are entries with Bangalore,shall we try with a diff place,s
         #and it worked!So,lets fill all NaN values as Mysore
```

```
Out[19]: _unit_id                 0
         _unit_state              0
         _trusted_judgments       0
         _last_judgment_at        0
         gender                   0
         gender:confidence        0
         profile_yn               0
         profile_yn:confidence    0
         created                  0
         description              0
         fav_number               0
         gender_gold              0
         link_color               0
         name                     0
         profile_yn_gold          0
         profileimage             0
         retweet_count            0
         sidebar_color            0
         text                     0
         tweet_count              0
         tweet_created            0
         tweet_id                 0
         tweet_location           0
         user_timezone            0
         dtype: int64
```

Fig-2.1.15: No entries of Mysore

Moreover, the column named 'user_timezone' had 38% of missing values (Fig-2.6) in it which too had to be fixed.'UTC + 9' is a time-zone in Japan, Korea etc. Hence, it was planned to replace NaN with 'UTC+9'. Also, this time-zone did not exist before-hand in the dataset. (Fig- 2.7).

```
In [22]: df[df['user_timezone'].isna()].count()
         #38% NaN values=>fix/replace them by another timezone which is not already available in the col
```

```
Out[22]: _unit_id                 7798
         _unit_state              7798
         _trusted_judgments       7798
         _last_judgment_at        7774
         gender                   7768
         gender:confidence        7790
         profile_yn               7798
         profile_yn:confidence    7798
         created                  7798
         description              5097
         fav_number               7798
         gender_gold                24
         link_color               7798
         name                     7798
         profile_yn_gold            24
         profileimage             7798
         retweet_count            7798
         sidebar_color            7798
         text                     7798
         tweet_count              7798
         tweet_created            7798
         tweet_id                 7798
         tweet_location           7798
         user_timezone               0
         dtype: int64
```

Fig-2.1.6: Missing values in user_timezone

```
In [23]: df[df['user_timezone'] == "UTC+9"]['user_timezone'].count()

         #Randomly fill with UTC+9 which is the time-zone in Japan,Korea etc.,

Out[23]: 0
```

```
In [24]: #Fill null values of user_timezone with the value 'UTC+9'
         df['user_timezone'] = df['user_timezone'].fillna("UTC+9")
```

```
In [25]: #View the user_timezone column
         df['user_timezone']

Out[25]: 0                             Chennai
         1         Eastern Time (US & Canada)
         2                            Belgrade
         3         Pacific Time (US & Canada)
         4                               UTC+9
                             ...
         20045                           UTC+9
         20046                           UTC+9
         20047                           UTC+9
         20048                           UTC+9
         20049                           UTC+9
         Name: user_timezone, Length: 20050, dtype: object
```

```
In [26]: #To view the dataset
         df
```

Fig-2.1.7: Fixing NaN values in user_timezone column

The next column named _last_judgment_at had 50 NaN values out of 20050 rows, which was just 0.249% of data (Fig-2.8). So, all these rows were just removed. The dataset then had 20000 rows and the number of columns being the same 24.

```
In [27]: df[df['_last_judgment_at'].isna()].count()

         #50 out of 200050 are missing values=>0.249% missing values < 10%=>so,remove these 50 rows

Out[27]: _unit_id                 50
         _unit_state              50
         _trusted_judgments       50
         _last_judgment_at         0
         gender                   50
         gender:confidence        50
         profile_yn               50
         profile_yn:confidence    50
         created                  50
         description              44
         fav_number               50
         gender_gold              50
         link_color               50
         name                     50
         profile_yn_gold          50
         profileimage             50
         retweet_count            50
         sidebar_color            50
         text                     50
         tweet_count              50
         tweet_created            50
         tweet_id                 50
         tweet_location           50
         user_timezone            50
         dtype: int64
```

Fig-2.1.8: Missing values in _last_judgment_at

Besides, the independent column, our target variable itself had a very few missing values-97 out of 20000 were missing (Fig-2.9) which too were simply removed like the _last_judgment_at column. The dataset then had 19903 rows and 24 columns.

```
In [30]: #Detect the missing values of gender column & count it
         df[df['gender'].isna()].count()

Out[30]: _unit_id                 97
         _unit_state              97
         _trusted_judgments       97
         _last_judgment_at        97
         gender                    0
         gender:confidence        71
         profile_yn               97
         profile_yn:confidence    97
         created                  97
         description              82
         fav_number               97
         gender_gold               0
         link_color               97
         name                     97
         profile_yn_gold           0
         profileimage             97
         retweet_count            97
         sidebar_color            97
         text                     97
         tweet_count              97
         tweet_created            97
         tweet_id                 97
         tweet_location           97
         user_timezone            97
         dtype: int64
```

Fig-2.1.9: Missing values in gender column

The ‘description’ column had 3723 NaN values out of 19903 values. Hence, it had to be fixed and thus the missing values were replaced with ‘Personal’.

The column named profile_yn_gold had only missing values (Fig-2.10) like tweet_coord. Hence, it was also dropped using drop () after which the dataset had only 23 columns and the rows were still 19903.

```
In [43]: #To detect the null values in the column if it holds to be true
         df[df['profile_yn_gold'].isna() == True].count()

Out[43]: _unit_id                 19903
         _unit_state              19903
         _trusted_judgments       19903
         _last_judgment_at        19903
         gender                   19903
         gender:confidence        19903
         profile_yn               19903
         profile_yn:confidence    19903
         created                  19903
         description              19903
         fav_number               19903
         gender_gold                  0
         link_color               19903
         name                     19903
         profile_yn_gold              0
         profileimage             19903
         retweet_count            19903
         sidebar_color            19903
         text                     19903
         tweet_count              19903
         tweet_created            19903
         tweet_id                 19903
         tweet_location           19903
         user_timezone            19903
         dtype: int64
```

Fig-2.1.10: Missing values in profile_yn

Finally, it was cross-checked and confirmed that the entire remaining dataset did not have any missing value at all. Hence, all such NaN values were removed successfully and the entire dataset was cleaned successfully.

2.2. EDA AND VISUALIZING THE DATASET:

Firstly, non-categorical columns were found using describe () (Fig-2.2.1)

DRAW BOX PLOTS AND VISUALIZE ALL THE COLUMNS

In [76]:
```python
#Importing seaborn & matplotlib libraries
import seaborn as sb
import matplotlib.pyplot as plt
```

In [77]:
```python
#To generate descriptive statistics of the dataset
df.describe()
```

Out[77]:

	_unit_id	_trusted_judgments	gender:confidence	profile_yn:confidence	fav_number	retweet_count	tweet_count	tweet_id
count	1.990300e+04	19903.0	19903.000000	19903.000000	19903.000000	19903.000000	1.990300e+04	1.990300e+04
mean	8.157294e+08	3.0	0.885820	0.994363	4371.563634	0.079285	3.898116e+04	6.587350e+17
std	5.874119e+03	0.0	0.184444	0.043119	12361.967831	2.658359	1.170531e+05	5.000124e+12
min	8.157192e+08	3.0	0.314000	0.627200	0.000000	0.000000	1.000000e+00	6.587300e+17
25%	8.157243e+08	3.0	0.678250	1.000000	10.000000	0.000000	2.398000e+03	6.587300e+17
50%	8.157294e+08	3.0	1.000000	1.000000	456.000000	0.000000	1.149200e+04	6.587400e+17
75%	8.157345e+08	3.0	1.000000	1.000000	3319.000000	0.000000	4.008100e+04	6.587400e+17
max	8.157396e+08	3.0	1.000000	1.000000	341621.000000	330.000000	2.680199e+06	6.587400e+17

In [78]:
```python
#To draw a box plot to show distributions with respect to categories
sb.boxplot(data=df,y="_unit_id",x="gender")
plt.show()
```

Fig-2.2.1: Finding columns with numerical values

Secondly, box plots were drawn for each of these columns taking gender along X-axis as the gender column is our target variable or independent column or our categorical column of interest.

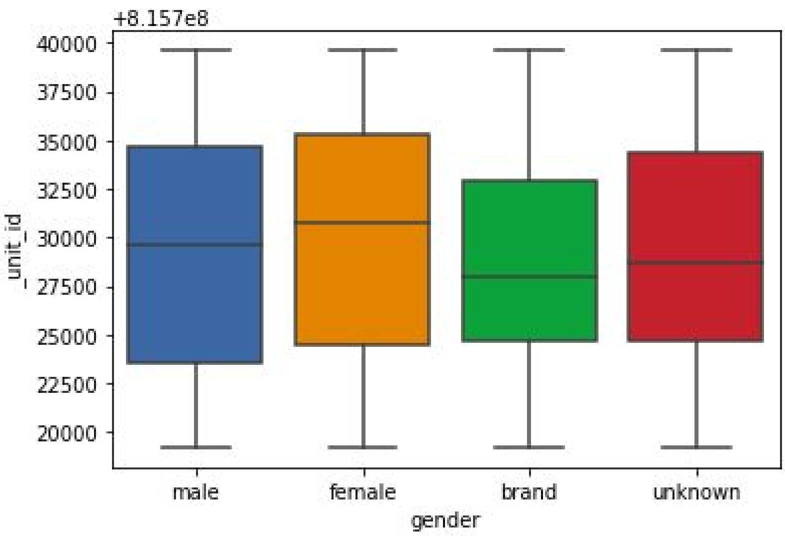

Fig-2.2.2: Box plot of _unit_id vs gender

```
In [79]: df[['gender']].values
         #To draw a box plot to show distributions with respect to categories
         sb.boxplot(data=df,y="_trusted_judgments", x="gender")
         plt.show()
```

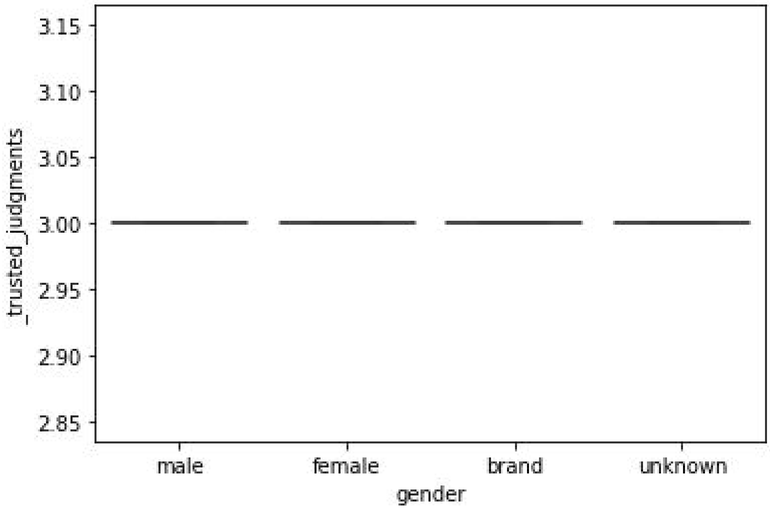

Fig-2.2.3: Box plot of _trusted_judgments vs gender

```
In [80]: #To draw a box plot to show distributions with respect to categories
         sb.boxplot(data=df,y="gender:confidence",x="gender")
         plt.show()
```

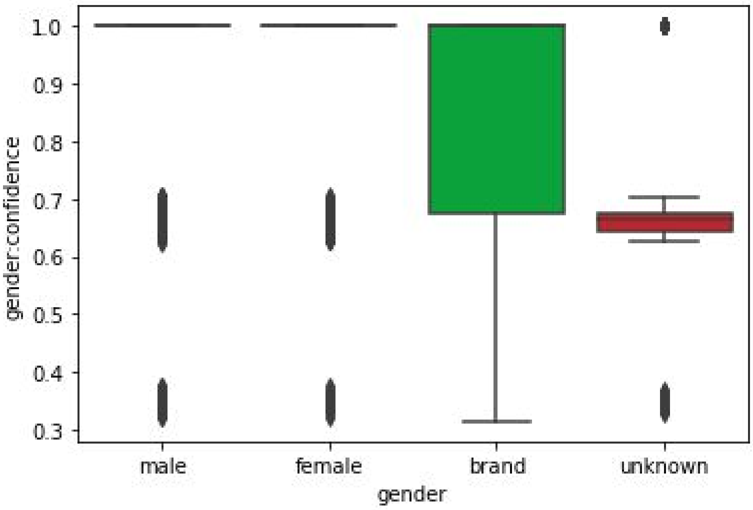

Fig-2.2.4: Box plot of gender:confidence vs gender

In [81]:
```
#To draw a box plot to show distributions with respect to categories
sb.boxplot(data=df,y="profile_yn:confidence",x="gender")
plt.show()
```

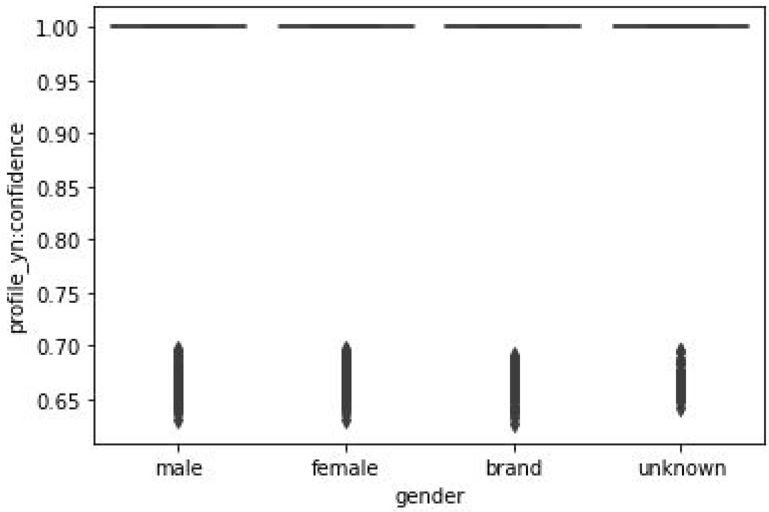

Fig-2.2.5: Box plot of profile_yn_confidence vs gender

In [82]:
```
#To draw a box plot to show distributions with respect to categories
sb.boxplot(data=df,y="fav_number",x="gender")
plt.show()
```

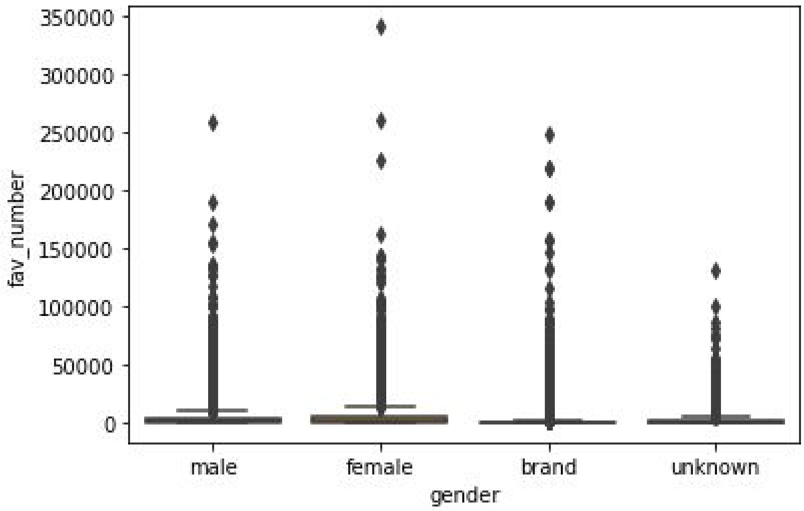

Fig-2.2.6: Box plot of fav_number vs gender

In [83]:
```
#To draw a box plot to show distributions with respect to categories
sb.boxplot(data=df,y="retweet_count",x="gender")
plt.show()
```

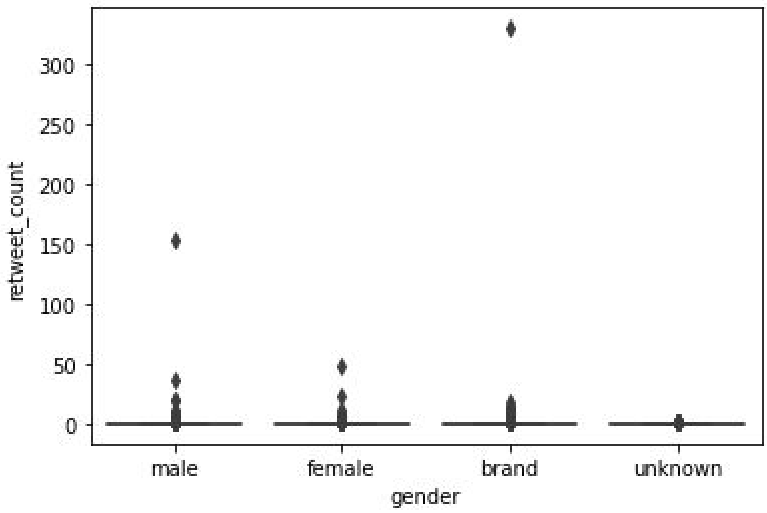

Fig-2.2.7: Box plot of retweet_count vs gender

In [84]:
```
#To draw a box plot to show distributions with respect to categories
sb.boxplot(data=df,y="tweet_count",x="gender")
plt.show()
```

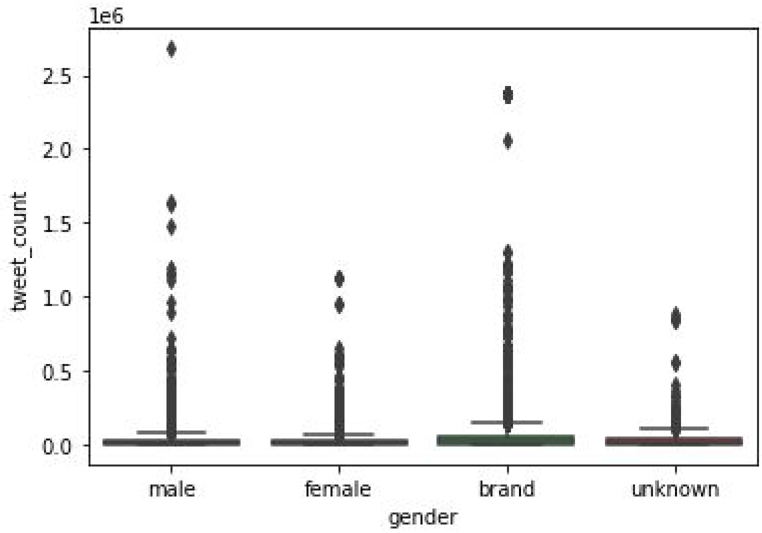

Fig-2.2.8: Box plot of tweet_count vs gender

In [85]:
```
#To draw a box plot to show distributions with respect to categories
sb.boxplot(data=df,y="tweet_id",x="gender")
plt.show()
```

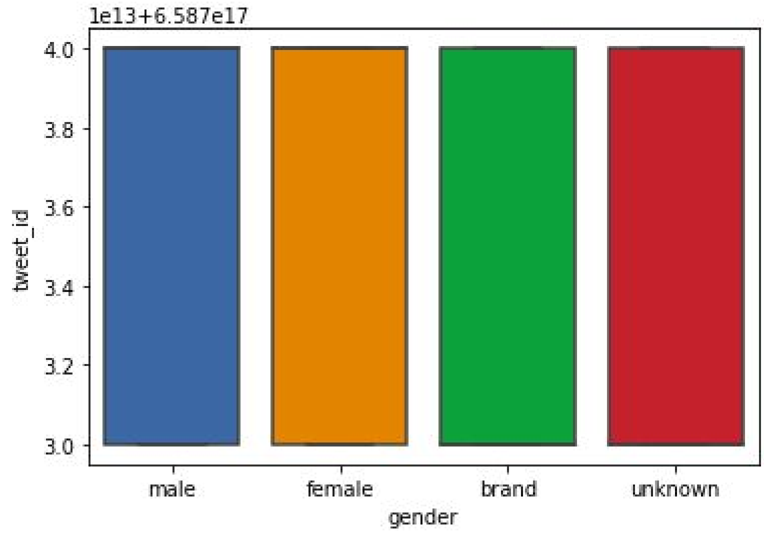

Fig-2.2.9: Box plot of tweet_id vs gender

2.3. QnA ON THE DATASET:

The two questions on the gender dataset include:

Q1: Which gender makes more typos in their tweets?

Q2: What is the maximum number of tweets tweeted by women?

Answers:

1. Considering retweet_count as the number of times a person tweets again and posts, brand is the gender with maximum typos (Fig-2.3.1) in their tweets. A person will retweet only if there was a typo in the tweet.

EDA QUESTIONS:

Q1:Which gender makes more typos in their tweets?

Q2:What is the maximum number of tweets tweeted by women?

```
In [86]: #df[df["_unit_state"] == "finalized"]['gender'].count()

         #all are finalized only(after cleaning the data)

In [87]: #To group DataFrame of the gender columns using retweet_count and find it's maximum
         df.groupby("gender")["retweet_count"].max()

Out[87]: gender
         brand      330
         female      49
         male       153
         unknown      2
         Name: retweet_count, dtype: int64

In [88]: #To find maximum from the group DataFrame of the gender columns using retweet_count and find it's maximum
         maxTypos = max(df.groupby("gender")["retweet_count"].max())
         #Get the dataset if retweet_count is equal to the above variable
         df[df["retweet_count"] == maxTypos]["gender"]
         #Q1

Out[88]: 1209    brand
         Name: gender, dtype: object
```

Fig-2.3.1: Answer for Q1. has maximum retweet count for each of the four genders. The next cell shows only the gender making the maximum number of typos.

2. 1125963 is the maximum number of tweets tweeted by women (Fig-2.3.2)

```
In [89]: #Get maximum tweet_count from the dataset were gender is equal to 'Female'
         df[df["gender"] == "female"]["tweet_count"].max()
         #Q2

Out[89]: 1125963
```

Fig-2.3.2: Answer for Q2

2.4. FEATURE SELECTION:

Firstly, label encoding was performed on all the categorical columns (Fig-2.4.1) including gender. This step was essential only to draw the heat map and learn the dependencies between the columns(features).

PREPROCESSING

```
In [90]: #Importing Label Encoding
         from sklearn.preprocessing import LabelEncoder
```

```
In [91]: #Encode target labels with value between 0 and n_classes-1
         le = LabelEncoder()
         #Fit label encoder and return encoded labels in a variable
         df['gender'] = le.fit_transform(df['gender'])
         #Fit label encoder and return encoded labels in a variable
         df['profile_yn'] = le.fit_transform(df['profile_yn'])
         #Fit label encoder and return encoded labels in a variable
         df['name'] = le.fit_transform(df['name'])
         #Fit label encoder and return encoded labels in a variable
         df['link_color'] = le.fit_transform(df['link_color'])
         #Fit label encoder and return encoded labels in a variable
         df['_unit_state'] = le.fit_transform(df['_unit_state'])
```

Fig-2.4.1: Label Encoding

Hence, it was convenient enough to draw the heat map (Fig-2.4.2) and thus gender was observed depend only on 12 columns out of 23 columns. These 12 columns were thus considered as the features:

'user_timezone','tweet_id','tweet_count','sidebar_color','profileimage', 'link_color','fav_number','created','profile_yn','_last_judgment_at' and '_unit_state' which can be found in the direction of Y-axis(Fig-2.4.2).Also, all of these columns are the independent variables and gender is the dependent variable as it depends on all these features.

FEATURE SELECTION

In [93]:
```
#Plot rectangular data as a color-encoded matrix
sb.heatmap(df.corr())
```

Out[93]: <matplotlib.axes._subplots.AxesSubplot at 0x7f29f94c0ac0>

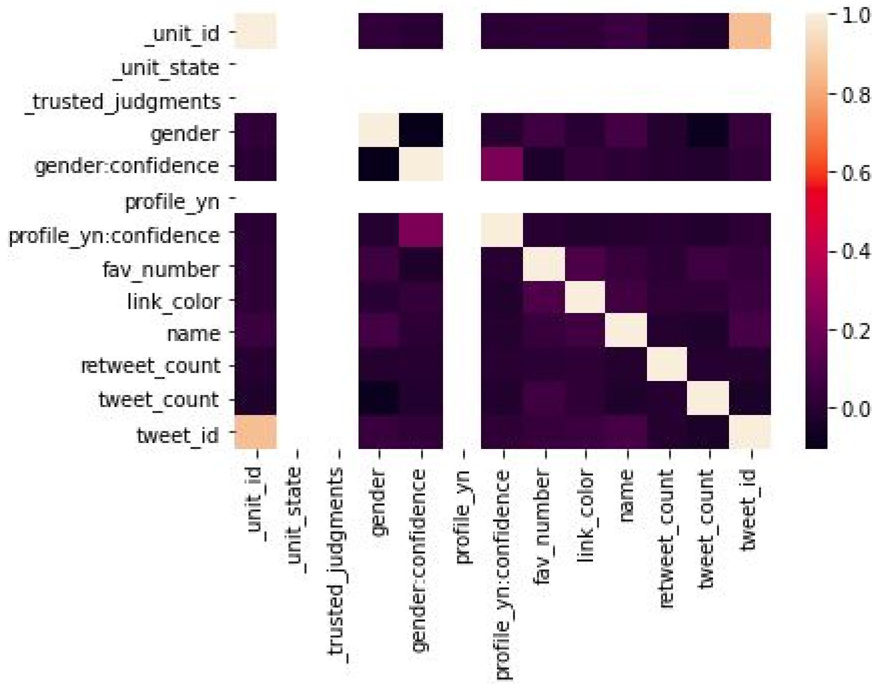

Fig-2.4.2: Heat map

2.5. CLASSIFICATION:

The aim was to classify the gender as male, female, unknown or brand which after label encoding changed to 0, 1 , 2 and 3, given all the above dozen features. The three algorithms used include

Target variable/ Dependent variable:

Here, the target variable is 'gender', which is dependent on other features in our dataset.

Response variable/ Independent variable:

Here, the response variables are 'gender:confidence', 'profile_yn:confidence', 'name', 'link_color', 'retweet_count', 'tweet_count' and '_unit_id', which are independent variables in our dataset.

```
In [100]: #Target variable/ dependent variable
          Y = df[['gender']].values
          #Independent variables/ Response variables
          X = df[['_unit_id', 'gender:confidence', 'profile_yn:confidence', 'name', 'link_color', 'retweet_count', 'tweet_cou
```

Fig-2.5: Target & Response variables

Classification algorithms used:

- SVC (Support Vector Classification)
- KNN (K Nearest Neighbours)
- Random Forest Classification.

2.5. 1. SVM MODEL:

As the SVM algorithm works irrespective of outliers, the step of removing outliers was performed after training this model. Its predictions are 34.41% accurate (Fig-2.5.1.2). The SVM model does not suit this data at all as it is the least accurate of all the three models.

SVM

```
In [106]: #Importing SVC
          from sklearn.svm import SVC
          #C-Support Vector Classification
          svc = SVC(kernel='rbf')
          # training Linear Regression model on training data
          svc.fit(X_train, Y_train)# The coefficients

/home/aishwarya/anaconda3/lib/python3.8/site-packages/sklearn/utils/validation.py:73: DataConversionWarning: A col
umn-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example u
sing ravel().
  return f(**kwargs)

Out[106]: SVC()
```

Fig-2.5.1.1: SVM model initialization

```
In [107]: #Perform classification on samples in X
          y_pred = svc.predict(X_test)
          #Importing classification_report & confusion_matrix
          from sklearn.metrics import classification_report, confusion_matrix
          #Build a text report showing the main classification metrics
          print (classification_report(Y_test, y_pred))
          #Accuracy classification score
          print("Test set Accuracy: ", metrics.accuracy_score(Y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.00      0.00      0.00      1454
           1       0.34      1.00      0.51      1712
           2       0.00      0.00      0.00      1551
           3       0.00      0.00      0.00       259

    accuracy                           0.34      4976
   macro avg       0.09      0.25      0.13      4976
weighted avg       0.12      0.34      0.18      4976

Test set Accuracy:  0.3440514469453376
```

```
/home/aishwarya/anaconda3/lib/python3.8/site-packages/sklearn/metrics/_classification.py:1221: UndefinedMetricWarn
ing: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_div
ision` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

Fig-2.5.1.2: SVM model’s accuracy

2.5. 2. REMOVING OUTLIERS:

Outliers could not be removed before training the model because the box plots for the dataset were quite absurd! Moreover, the values in Y-axis weren't exactly the values of the data points thereby leaving us in an ambiguous state to decide the barrier in-order to remove the outliers. Hence, data was given without removing any outlier.

Outliers detection

```
In [99]: #To determine the figure width & height
         plt.rcParams['figure.figsize'] = 10,5
         df.boxplot(column=['gender:confidence','_trusted_judgments','fav_number','retweet_count','tweet_count'])

Out[99]: <matplotlib.axes._subplots.AxesSubplot at 0x7f6ac2d54cd0>
```

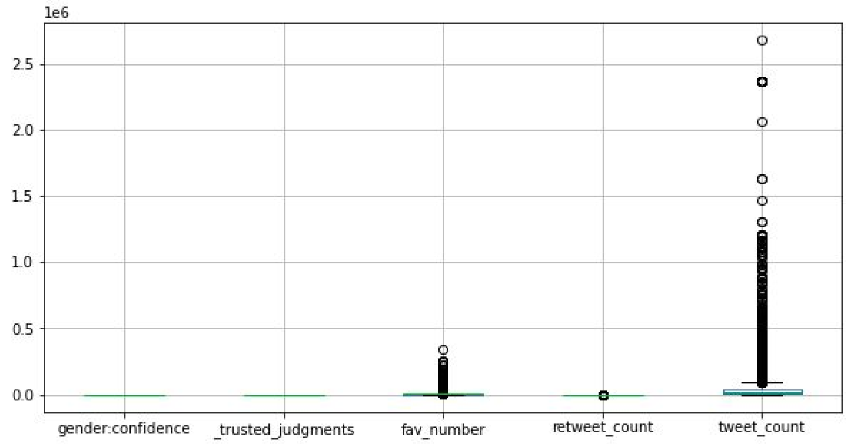

Fig-2.5.3.1: Outliers Detection

2.5. 3. KNN MODEL:

The model was the most accurate when K=number of neighbours was chosen as 17 although it turned out to be only 42.75% accurate (Fig-2.5.3.3).The KNN model is more accurate than SVM but less accurate than random forest.Moreover,only half of the entire dataset was given to KNN as it is a lazy algorithm and works better with less data or smaller datasets. The number 17 was chosen for K by trial and error method.

KNN Classifier

```
In [101]: # Knn Classifier
          from sklearn.model_selection import train_test_split
          #Split arrays or matrices into random train and test subsets
          X_train, X_test, Y_train, Y_test = train_test_split(X,Y)

In [102]: #Returns shape of each train & test data
          X_train.shape, X_test.shape, Y_train.shape, Y_test.shape

Out[102]: ((14927, 7), (4976, 7), (14927, 1), (4976, 1))

In [103]: #Importing KNeighborsClassifier & accuracy_score
          from sklearn.neighbors import KNeighborsClassifier
          from sklearn.metrics import accuracy_score
```

Fig-2.5.3.2: KNN Model Importing & Initialization

```
In [105]: #Importing metrics
          from sklearn import metrics
          knn = KNeighborsClassifier(n_neighbors=17)
          #Fit the model using X as training data and y as target values
          knn.fit(X_train, Y_train)
          #Predict the class labels for the provided data
          y_pred = knn.predict(X_test)
          #Accuracy classification score
          print("Test set Accuracy: ", metrics.accuracy_score(Y_test, y_pred))

          <ipython-input-105-a8192ebdfe37>:5: DataConversionWarning: A column-vector y was passed when a 1d array was expect
          ed. Please change the shape of y to (n_samples, ), for example using ravel().
            knn.fit(X_train, Y_train)

          Test set Accuracy:  0.427451768488746
```

Fig-2.5.3.3: KNN Model's accuracy

2.5. 4. RANDOM FOREST CLASSIFICATION MODEL:

The model is **best suited** for the given gender dataset because it shows the highest accuracy of all (Fig-2.5.4.2). It is around 48.10% accurate which is far better than the previous two models which hardly touched 50% accuracy.

Random Forest

```
In [108]: #Importing RandomForestClassifier
          from sklearn.ensemble import RandomForestClassifier
          #Intializing a random forest classifier
          rfc = RandomForestClassifier()
          # training Linear Regression model on training data
          rfc.fit(X_train, Y_train)

          <ipython-input-108-fa6e3ffe1b15>:6: DataConversionWarning: A column-vector y was passed when a 1d array was expect
          ed. Please change the shape of y to (n_samples,), for example using ravel().
            rfc.fit(X_train, Y_train)

Out[108]: RandomForestClassifier()
```

Fig-2.5.4.1: Import & Initialize Random Forest Classifier model

In [109]:
```
#Predict class for X
y_pred = rfc.predict(X_test)
#Build a text report showing the main classification metrics
print (classification_report(Y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.52      0.52      0.52      1454
           1       0.52      0.55      0.53      1712
           2       0.41      0.42      0.42      1551
           3       0.36      0.16      0.22       259

    accuracy                           0.48      4976
   macro avg       0.45      0.41      0.42      4976
weighted avg       0.48      0.48      0.48      4976
```

In [110]:
```
#Accuracy classification score
print("Test set Accuracy: ", metrics.accuracy_score(Y_test, y_pred))
```

```
Test set Accuracy:  0.4809083601286174
```

Fig-2.5.4.2: Accuracy of Random Forest Classifier model

CONCLUSION:

All the algorithms used here were from built-in libraries. Also, the same set of features were given to train all the three models and then their accuracies were compared using accuracy_score.

SVM does not suit for the twitter dataset as its huge and thus SVM model consumes more time to train. It has a poor accuracy as poor as 34.41%.

KNN model shows average performance on the dataset lying in between the SVM model and random forest classifier with the accuracy of 42.75%.

However, a random forest classifier is the only model which successfully fits this huge dataset of 19903 rows and 23 columns even after removing redundant columns and missing values. Hence, using random forest classification, we could achieve nearly 48% accuracy.