CAPSTONE PROJECT
Final Report

Analyzing User Behaviour to investigate Age group Impact on Video Engagement Using Machine Learning Techniques

Batch Details	PGPDSE-FT Chennai Oct 2022
Team Name	Seventh Sense
Team Members	1. Gabrilla Sabadini H 2. Hamdaan P 3. Monisha Lenin 4. Rohith G.V 5. Sanjay Gowtham C 6. Shyam Sridhar
Domain of the Project	Social Media
Proposed project title	Analyzing User Behaviour to Investigate Age group Impact on Video Engagement
Group Number	2
Team Leader	Shyam Sridhar
Mentor	Mrs.Pranita Mahajan

Date: 05-04-2023

Signature of the Mentor

Signature of the team leader

Table of Contents

1.Introduction:

Social media refers to a collection of online communication channels that facilitate interaction, content-sharing, and collaboration among individuals and groups. These platforms are primarily web and mobile-based, and they allow users to create and share user-generated content, connect with others, and engage in real-time conversations.

Social media has revolutionized the way we communicate and share information, enabling us to connect with people across the globe instantaneously. From Facebook and Twitter to Instagram and Snapchat, social media platforms have become a central part of our daily lives, providing us with a platform to connect with friends, family, and colleagues, share our experiences, and keep up with the latest news and trends.

However, social media has also been subject to criticism for its potential negative effects, including addiction, cyberbullying, misinformation, and the spread of hate speech. Despite these concerns, social media continues to play an essential role in shaping our digital landscape and shaping the way we communicate and interact with each other.

Current state of social media:

Describe the current state of the social media industry. Discuss the major players in the market, their market share, and the trends that are driving the industry. You can also discuss the different types of social media platforms, such as social networking sites, messaging apps, and photo and video sharing sites.

Key challenges:

Discuss the challenges that the social media industry is facing. This can include issues such as data privacy, fake news, and cyberbullying. Discuss how these challenges are being addressed and what impact they could have on the industry in the future.

Future outlook:

Finally, discuss the future outlook for the social media industry. What new trends are emerging? What new technologies are being developed? What impact will these changes have on the industry and its users?

2.INDUSTRY REVIEW

2.1 ABSTRACT

Age groups can be a useful predictor of user video engagement in social media sites because different age groups tend to have distinct preferences and behaviors when it comes to using social media and consuming video content. In this capstone project, we aim to develop a machine learning model to predict the age group of the social media user based on the user's involvement using historical data of video consumption and all relevant features of the dataframe. Our model will be trained on a large dataset of "Trell" social media and evaluated using various performance metrics. The outcome of this project will be a predictive model that can be used for user engagement,targeted advertising,content personalization and product development.

2.2 CURRENT PRACTICES:

The current practice for age group prediction through machine learning involves using various supervised learning algorithms such as Random Forest, Support Vector Machines,Neural Networks,Cluster Analysis and Ensemble Learning. These algorithms are trained on large datasets of social media sites and their features.Feature engineering techniques are also employed to select the most relevant features and transform them into a suitable format for the machine learning algorithms.The below given algorithms are very much in use in today's times.

1. **Random Forest** : Random Forest is a decision tree-based algorithm that is commonly used for classification problems. It can be used to predict the age group of social media users based on various features, such as user behavior, interests, demographics, and other relevant data.
2. **Support Vector Machines (SVMs)** : SVMs are a popular algorithm for classification problems. They can be used to predict age groups based on features such as the user's activity level, the type of content they interact with, and their social network.
3. **Neural Networks** : Neural Networks are deep learning models that can be used for classification problems. They can learn complex relationships between features and make accurate predictions. Neural Networks have been used to predict age groups based on various features such as user activity, location, and the types of content they engage with.

4. Cluster Analysis : Cluster analysis is a statistical technique that can be used to group users into different age groups based on their behavior and interests on social media platforms. This technique can help identify patterns and similarities among users that can be used to predict age group.

5. Ensemble Learning : Ensemble learning is a machine learning technique that combines multiple models to improve prediction accuracy. This technique has been used to predict age groups by combining multiple models such as Random Forest, SVM, and Neural Networks.

2.3 PROBLEM STATEMENT:

Analyzing User Behaviour to investigate Age group Impact on Video Engagement

1. Business Problem Understanding :

The business problem arises as enormous quantities of videos are available in the social media and these videos should be segregated to people of varying age based on their preferences.A predictive model can help in determining the targeted audience and also personalized content creation for people of varying age groups.

2. Business Objective :

To understand the user behavior and engagement on Trell, a social media platform that focuses on short-form video content, and develop a machine learning model that can predict user engagement and preferences based on their activity on the platform.

3. Approach :

The approach for predicting age groups using machine learning typically involves collecting data on social media sites, cleaning and preprocessing the data, selecting an appropriate algorithm, training and validating the model, and testing the model on new data to evaluate its accuracy and performance.

4. Conclusions :

Overall, machine learning models can be useful for age prediction in social media apps, providing insights into user demographics and behavior that can be used for personalized marketing, content recommendations, and user experience

improvement. However, careful consideration should be given to data privacy and ethical implications of such models.

2.4 IMPACT ON BUSINESS:

1. As we are predicting the age group of the user, class 1 contains the users with age of less than equal to 18. The Government Education Departments and The Education tutoring institutes can concentrate those users and can provide the videos, advertisements and other motivating videos.Social media influencers can help to promote educational content to their followers, who are often interested in learning and personal growth. By working with influencers who align with the brand's values and mission, the app can reach a wider audience and build trust with users.

2. Engage with users : Engaging with users by responding to comments, hosting Q&A sessions, or creating forums or groups can help to build a community around educational content. This can create a sense of belonging and increase the likelihood of users sharing and recommending the app to others.

3. Improved Customer Experience : Predicting the resale value of a car can help businesses to offer more personalized pricing to their customers, resulting in a better customer experience. By tailoring the price based on the car's condition, mileage, and other factors, businesses can provide more accurate pricing, resulting in a more satisfied customer.

4. Identify the target audience : The first step is to identify the target audience for the election campaign. This could involve analyzing user data to determine which groups are most likely to engage with the campaign or using polling data to target specific demographics.

5. Develop messaging : Develop messaging that resonates with the target audience and aligns with the campaign's values and goals. This could involve creating slogans, catchphrases, or memorable images that will stick in the minds of users.

6. Advertise through social media : Social media platforms are often used to advertise election campaigns because they allow for precise targeting and have

high user engagement. The app could use targeted ads on social media platforms to reach potential voters and build awareness for the campaign.
7. Social media marketing : Social media apps can be used for marketing e-commerce products to a wide audience. By creating a social media presence for the e-commerce store, businesses can advertise their products, build brand awareness, and engage with customers.

8. Social selling : Social media apps can also be used for social selling, which involves using social media platforms to sell products directly to customers. This could involve posting product information, reviews, or special offers on social media platforms to attract customers.

9. Influencer marketing : Influencer marketing involves partnering with social media influencers to promote e-commerce products to their followers. This can help to increase brand visibility and build trust with potential customers.

3.Data Exploration :

3.1 Dataset :

Dataset Name	Trell dataset
Reference	Cascade Cup Data Science Hackathon
Link to web page	https://aiplanet.com/challenges/46/cascade-cup-data-science-hackathon/overview/about

3.2 Data Description :

userId : Unique number given to each user

tier : Tier of the city in which the user is residing

gender : Categorical feature representing the gender of the user. 1 represents male and 2 represents female

following_rate : Number of accounts followed by the user(feature is normalized)

followers_avg_age : Average of age groups of all the followers of the user

following_avg_age : Average of age groups of all the accounts followed by the user

max_repetitive_punc : Maximum repetitive punctuations found in the bio and comments of the user

num_of_hashtags_per_action : Average number of hashtags used by the user per comment

emoji_count_per_action : Average number of emojis used by the user per comment

punctuations_per_action : Average number of punctuations used by the user per comment

number_of_words_per_action : Average number of words used by the user per comment

avgCompletion : Average watch time completion rate of the videos

avgTimeSpent : Average time spent by the user on a video in seconds

avgDuration : Average duration of the videos that the user has watched till date

avgComments : Average number of comments per video watched

creations : Average number of videos uploaded by the user

content_views : Average number of videos watched

num_of_comments : Total number of comments made by the user(normalized)

weekends_trails_watched_per_day : Number of videos watched on weekends per day

weekdays_trails_watched_per_day : Number of videos watched on weekdays per day

slot1_trails_watched_per_day : The day is divided into 4 slots. This feature represents the average number of videos watched in this particular time slot

slot2_trails_watched_per_day : The day is divided into 4 slots. This feature represents the average number of videos watched in this particular time slot

slot3_trails_watched_per_day : The day is divided into 4 slots. This feature represents the average number of videos watched in this particular time slot

slot4_trails_watched_per_day : The day is divided into 4 slots. This feature represents the average number of videos watched in this particular time slot

avgt2 : Average number of followers of all the accounts followed by the user

age_group : This is a categorical feature denoting the age of the user. Age of users is divided into 4 groups,
1: <18y; 2: 18-24y; 3: 24-30y; 4: >30y

3.3 Data Understanding :

Importing the data :

Reading the dataset

```
In [2]:
1 df = pd.read_csv('train_age_dataset.csv')
2 df.head(5)
```

Out[2]:

	Unnamed: 0	userId	tier	gender	following_rate	followers_avg_age	following_avg_age	max_repetitive_punc	num_of_hashtags_per_action	emoji_count_per_
0	265153	48958844	2	1	0.0	0.0	0.0	0	0.0	
1	405231	51100441	2	2	0.0	0.0	0.0	0	0.0	
2	57867	6887426	2	1	0.0	0.0	0.0	0	0.0	
3	272618	50742404	2	1	0.0	0.0	0.0	0	0.0	
4	251123	45589200	2	2	0.0	0.0	0.0	0	0.0	

5 rows × 27 columns

- **Shape of the data :**

Understanding number of entries and attributes

```
In [4]:
1 df.shape
```

Out[4]: (488877, 27)

There are 488877 rows and 27 columns present in the dataset.

- **Dropping redundant columns :**

The columns ['Unnamed: 0','userId'], are insignificant so we are dropping the columns respectively.

Dropping unneccassary columns

In [6]:
```
df.drop(['Unnamed: 0','userId'],axis=1,inplace = True)
df.columns
```

Out[6]:
```
Index(['tier', 'gender', 'following_rate', 'followers_avg_age',
       'following_avg_age', 'max_repetitive_punc',
       'num_of_hashtags_per_action', 'emoji_count_per_action',
       'punctuations_per_action', 'number_of_words_per_action',
       'avgCompletion', 'avgTimeSpent', 'avgDuration', 'avgComments',
       'creations', 'content_views', 'num_of_comments',
       'weekends_trails_watched_per_day', 'weekdays_trails_watched_per_day',
       'slot1_trails_watched_per_day', 'slot2_trails_watched_per_day',
       'slot3_trails_watched_per_day', 'slot4_trails_watched_per_day', 'avgt2',
       'age_group'],
      dtype='object')
```

- **Data Types of variables :**

Understanding the DataType of each columns

In [7]:
```
1 df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 488877 entries, 0 to 488876
Data columns (total 25 columns):
 #   Column                          Non-Null Count   Dtype
---  ------                          --------------   -----
 0   tier                            488877 non-null  int64
 1   gender                          488877 non-null  int64
 2   following_rate                  488877 non-null  float64
 3   followers_avg_age               488877 non-null  float64
 4   following_avg_age               488877 non-null  float64
 5   max_repetitive_punc             488877 non-null  int64
 6   num_of_hashtags_per_action      488877 non-null  float64
 7   emoji_count_per_action          488877 non-null  float64
 8   punctuations_per_action         488877 non-null  float64
 9   number_of_words_per_action      488877 non-null  float64
 10  avgCompletion                   488877 non-null  float64
 11  avgTimeSpent                    488877 non-null  float64
 12  avgDuration                     488877 non-null  float64
 13  avgComments                     488877 non-null  int64
```

Understanding the DataType of each columns

```
In [7]: df.info()
```

```
 8   punctuations_per_action          488877 non-null  float64
 9   number_of_words_per_action       488877 non-null  float64
 10  avgCompletion                    488877 non-null  float64
 11  avgTimeSpent                     488877 non-null  float64
 12  avgDuration                      488877 non-null  float64
 13  avgComments                      488877 non-null  int64
 14  creations                        488877 non-null  float64
 15  content_views                    488877 non-null  float64
 16  num_of_comments                  488877 non-null  float64
 17  weekends_trails_watched_per_day  488877 non-null  float64
 18  weekdays_trails_watched_per_day  488877 non-null  float64
 19  slot1_trails_watched_per_day     488877 non-null  float64
 20  slot2_trails_watched_per_day     488877 non-null  float64
 21  slot3_trails_watched_per_day     488877 non-null  float64
 22  slot4_trails_watched_per_day     488877 non-null  float64
 23  avgt2                            488877 non-null  float64
 24  age_group                        488877 non-null  int64
dtypes: float64(20), int64(5)
memory usage: 93.2 MB
```

The variables gender, tier are the categorical columns which are already encoded and hence they are in int64 data type.

Also the columns ‘followers_avg_age’ and ‘following_avg_age’ which are actually categorical columns with continuous variables which will be encoded later.

The target variable ‘age_group’ is already encoded and in int64 data type.

All the other columns are in proper data type as per the data description.

3.4 Feature Engineering :

The feature engineering pipeline is the preprocessing steps that transform raw data into features that can be used in machine learning algorithms, such as predictive models. Predictive models consist of an outcome variable and predictor variables, and it is during the feature engineering process that the most useful predictor variables are created and selected for the predictive model.

Feature engineering in ML consists of four main steps:

- Feature Creation
- Transformations
- Feature Extraction
- Feature Selection

Feature engineering consists of creation, transformation, extraction, and selection of features, also known as variables, that are most conducive to creating an accurate ML algorithm.

In our dataset, we are going to convert the columns ‘followers_avg_age’ and

'following_avg_age' into categorical columns by binning the data as shown below :

- Class - 0 - No followers,No age group - binning range = (0)
- Class - 1 - < 18 years - binning range = (>0 to 1.5)
- Class - 2 - 18-24 years - binning range = (1.5 to 2.5)
- Class - 3 - 24-30 years - binning range = (2.5 to 3.5)
- Class - 4 - >30 years - binning range = (>3.5)

```
In [10]:
df.loc[(df['followers_avg_age']>0)  & (df['followers_avg_age']<1.5),'followers_avg_age']=1
df.loc[(df['followers_avg_age']>=1.5) & (df['followers_avg_age']<2.5),'followers_avg_age']=2
df.loc[(df['followers_avg_age']>=2.5) & (df['followers_avg_age']<3.5),'followers_avg_age']=3
df.loc[(df['followers_avg_age']>=3.5),'followers_avg_age']=4
```

```
In [11]:
df.loc[(df['followers_avg_age']>0)  & (df['following_avg_age']<1.5),'followers_avg_age']=1
df.loc[(df['followers_avg_age']>=1.5) & (df['following_avg_age']<2.5),'followers_avg_age']=2
df.loc[(df['followers_avg_age']>=2.5) & (df['following_avg_age']<3.5),'followers_avg_age']=3
df.loc[(df['followers_avg_age']>=3.5),'following_avg_age']=4
```

```
In [19]:
df['followers_avg_age'].value_counts()
```

```
Out[19]:
0.0    406543
2.0     56556
1.0     13870
3.0     10977
4.0       931
Name: followers_avg_age, dtype: int64
```

```
In [20]:
df['following_avg_age'].value_counts()
```

```
Out[20]:
0.0    406543
2.0     56556
1.0     13870
3.0     10977
4.0       931
Name: following_avg_age, dtype: int64
```

3.5 Summary Statistics for categorical values and numerical values :

- **Categorical variables :**

As the categorical columns 'gender','tier','followers_avg_age', 'following_avg_age' and 'age_group' are in numerical data type, we are converting them into object data type for performing the statistical analysis.

a) Categorical - summary statistics

```
In [13]:
df['gender']= df['gender'].astype(object)
df['tier']= df['tier'].astype(object)
df['followers_avg_age']= df['followers_avg_age'].astype(object)
df['following_avg_age']= df['followers_avg_age'].astype(object)
df['age_group']= df['age_group'].astype(object)
```

```
In [14]:
df.describe(exclude=np.number)
```

Out[14]:

	tier	gender	followers_avg_age	following_avg_age	age_group
count	488877	488877	488877.0	488877.0	488877
unique	3	2	5.0	5.0	4
top	2	1	0.0	0.0	1
freq	397890	384728	406543.0	406543.0	308315

Here the attributes 'followers_avg_age' and 'following_avg_age' have 5 different

classes, the attribute ‘gender’ have 2 different classes, the attribute ‘tier’ has 3 different classes and the target variable ‘age_group’ have 4 four different classes.

- **Numerical variables :**

a) Numerical - summary statistics

```
In [15]: 1 df.describe()
```

Out[15]:

	following_rate	max_repetitive_punc	num_of_hashtags_per_action	emoji_count_per_action	punctuations_per_action	number_of_words_per_action	avgCon
count	488877.000000	488877.000000	488877.000000	488877.000000	488877.000000	488877.000000	488877
mean	0.082233	0.739748	0.000277	0.000981	0.012805	0.179148	0
std	2.467781	3.075954	0.012221	0.020381	0.159553	0.647588	0
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0
25%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0
50%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0
75%	0.008621	0.000000	0.000000	0.000000	0.000000	0.150183	0
max	895.304000	624.000000	2.333333	3.000000	27.333333	262.666667	1

3.6 Null values :

Null values, also known as missing values, are a common occurrence in data frames. Null values occur when there is no data available for a particular cell in the dataframe. Null values indicate missing data, which can lead to incomplete analysis and inaccurate results. It can also lead to errors in data processing and modeling. In order to handle null values, data imputation techniques such as mean/mode imputation, forward/backward filling or knn imputation can be used. However, these techniques can introduce bias in the data and affect the accuracy of the analysis.

Here in our dataset, there are no null values found in the columns.

```
In [14]: 1 df.isnull().sum()
```

```
Out[14]: tier                          0
         gender                        0
         following_rate                0
         followers_avg_age             0
         following_avg_age             0
         max_repetitive_punc           0
         num_of_hashtags_per_action    0
         emoji_count_per_action        0
         punctuations_per_action       0
         number_of_words_per_action    0
         avgCompletion                 0
         avgTimeSpent                  0
         avgDuration                   0
         avgComments                   0
         creations                     0
         content_views                 0
         num_of_comments               0
         weekends_trails_watched_per_day  0
         weekdays_trails_watched_per_day  0
```

```
In [14]:  1 df.isnull().sum()
```

```
emoji_count_per_action            0
punctuations_per_action           0
number_of_words_per_action        0
avgCompletion                     0
avgTimeSpent                      0
avgDuration                       0
avgComments                       0
creations                         0
content_views                     0
num_of_comments                   0
weekends_trails_watched_per_day   0
weekdays_trails_watched_per_day   0
slot1_trails_watched_per_day      0
slot2_trails_watched_per_day      0
slot3_trails_watched_per_day      0
slot4_trails_watched_per_day      0
avgt2                             0
age_group                         0
dtype: int64
```

- **Checking for negative values:**

There are no columns with negative values.

b) Negative value detection

Checking if any of the attributes has any negative values which might be missing values since all the attributes we have are count of some unit

```
In [15]: negative_cols = df.columns[(df < 0).any()]

         print("Columns with negative values:", list(negative_cols))
```

```
Columns with negative values: []
```

There are no columns with negative values.

3.6 Outlier detection in Target column:

An outlier is an observation or data point that lies an abnormal distance away from other values in a dataset. In other words, it is a data point that is significantly different from the other observations in the same sample.

Outliers can have a significant impact on statistical analyses because they can skew results, affecting the accuracy of the findings. Outliers can also affect the performance of machine learning algorithms, leading to biased models and incorrect predictions.

In some cases, outliers may be the result of measurement errors or data entry mistakes, and they should be removed from the dataset. However, in other cases, outliers may be genuine and meaningful observations that should not be

ignored.

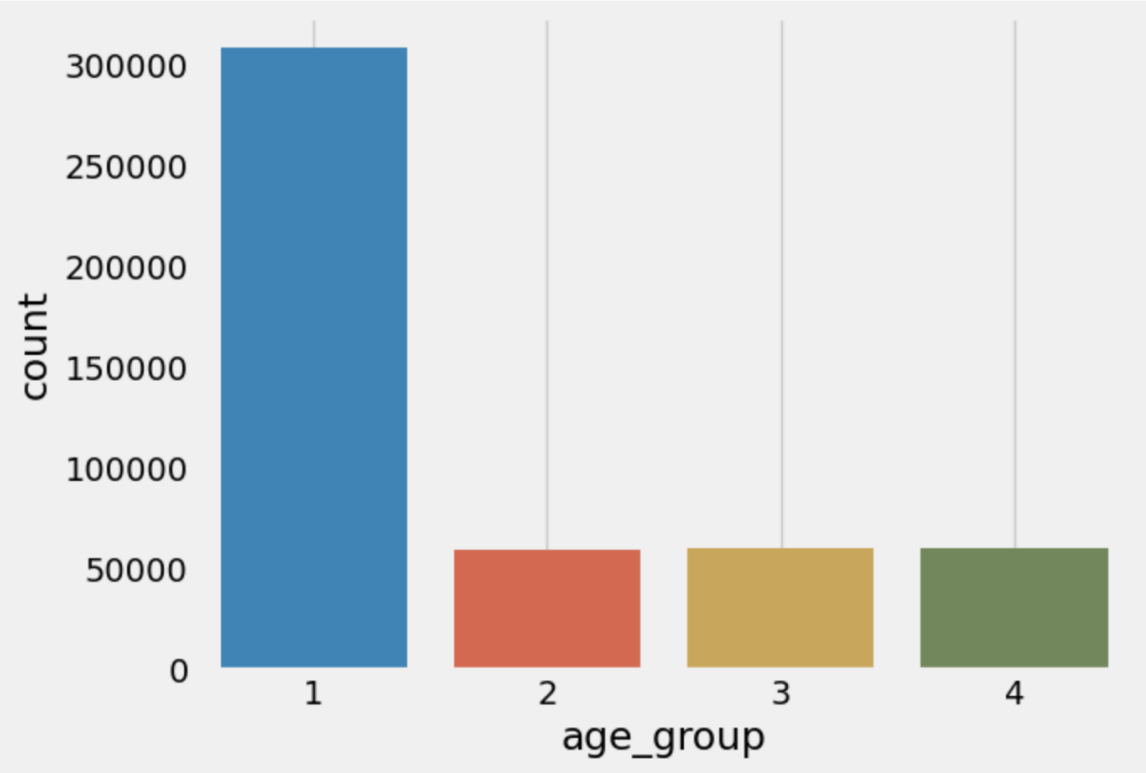
Therefore, it is important to carefully analyze outliers and determine the reason for their occurrence before deciding whether to remove them from the dataset or not

Here we check for outliers only for the target column.

Since the target column is categorical we check if any of the classes is represented very less by checking the balance of the data.

```
print('Total rows:', df['age_group'].value_counts().sum())
print(df['age_group'].value_counts())
sns.countplot(df['age_group'])
plt.grid()
plt.show()
```

```
Total rows: 488877
1    308315
4     60803
3     60404
2     59355
Name: age_group, dtype: int64
```



4. Statistical Analysis :

The process of hypothesis testing is to draw inferences or some conclusion about the overall population or data by conducting some statistical tests on a sample.

For drawing some inferences, we must make some assumptions that lead to two terms that are used in the hypothesis testing.

• Null hypothesis: It is regarding the assumption that there is no anomaly pattern or believing according to the assumption made.

• Alternate hypothesis: Contrary to the null hypothesis, it shows that observation is the result of real effect.

P-value :

In statistical analysis, the p-value is a measure of the strength of evidence against a null hypothesis. It is a probability value that represents the likelihood of obtaining the observed results (or more extreme results) if the null hypothesis is true.

Generally, we select the level of significance by 5 % and **i**f you have a strong prior knowledge about your data functionality, you can decide the level of significance.

If the p-value is less than the significant level, then we reject the null hypothesis and vice versa.

Two sample t-test (one tail test) :

Null Hypothesis (Ho): Average number of followers for female >= Average number of followers for male

Alternate Hypothesis (Ha): Average number of followers for female < Average number of followers for male

The significant level is 0.05.

In [17]:
```
import pandas as pd
from scipy.stats import ttest_ind

# Create separate dataframes for males and females
male_df = df[df['gender'] == 1]
female_df = df[df['gender'] == 2]

# Calculate the mean number of followers for each data frame
male_mean = male_df['following_rate'].mean()
female_mean = female_df['following_rate'].mean()

# Perform the one-tailed t-test
t_stat, p_value = ttest_ind(male_df['following_rate'], female_df['following_rate'], alternative='greater')

# Print the results
print('Male mean:', male_mean)
print('Female mean:', female_mean)
print('t-statistic:', t_stat)
print('p-value:', p_value)
```

```
Male mean: 0.09087721339931046
Female mean: 0.05030148136620538
t-statistic: 4.707319950416057
p-value: 1.2553171309784035e-06
```

The p-value is almost equal to 0.

That is , the p-value is less than 0.05 which implies that Null hypothesis (H0) is rejected.

Hence, the average number of followers for females are less than the average number of followers for males.

5.Data Visualization and Transformation:

5.1 Univariate:

A univariate plot is a type of graphical representation that displays the distribution of a single variable. It is a visualization tool used to understand the distribution of a dataset by showing the frequency of each observation or the values it takes.

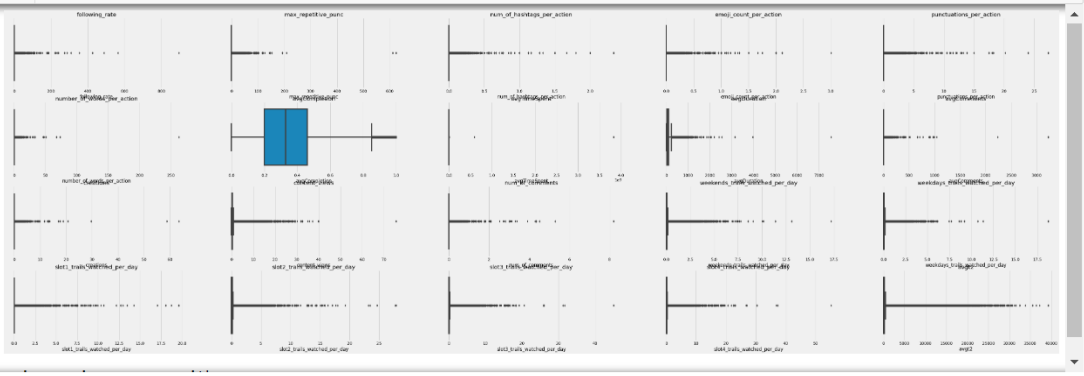
Univariate plots are commonly used to explore the central tendency, variability, and shape of a dataset, which helps in identifying patterns, outliers, and potential issues with data quality. Some common types of univariate plots include histograms, box plots, density plots, and bar charts.

By examining the distribution of a single variable through univariate plots, researchers can draw insights about the underlying population or phenomenon being studied, and make informed decisions based on the characteristics of the

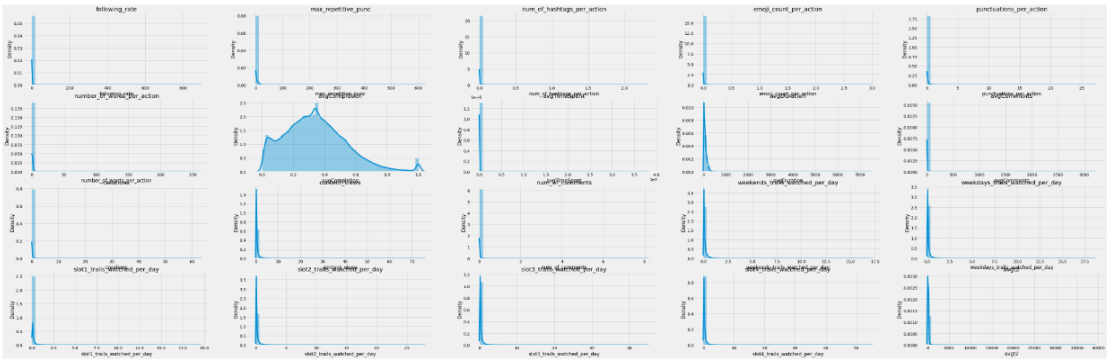
data.

For Numerical columns:

```
In [93]:
numerical_columns = df.select_dtypes(include='number').columns.tolist()
r = 6
c = 5
it = 1
plt.figure(figsize=(60,30))
for i in numerical_columns:
    plt.subplot(r,c,it)
    sns.boxplot(df[i])
    plt.title(i)
    it += 1
plt.show()
plt.tight_layout()
```



Based on the above box plot we can see a lot of values are distributed in and around 0. We had a doubt if these 0's are missing values or they are of 0 units. Based on the observation we found out that these 0's are not missing values rather they are actually 0 units for the respective attribute.

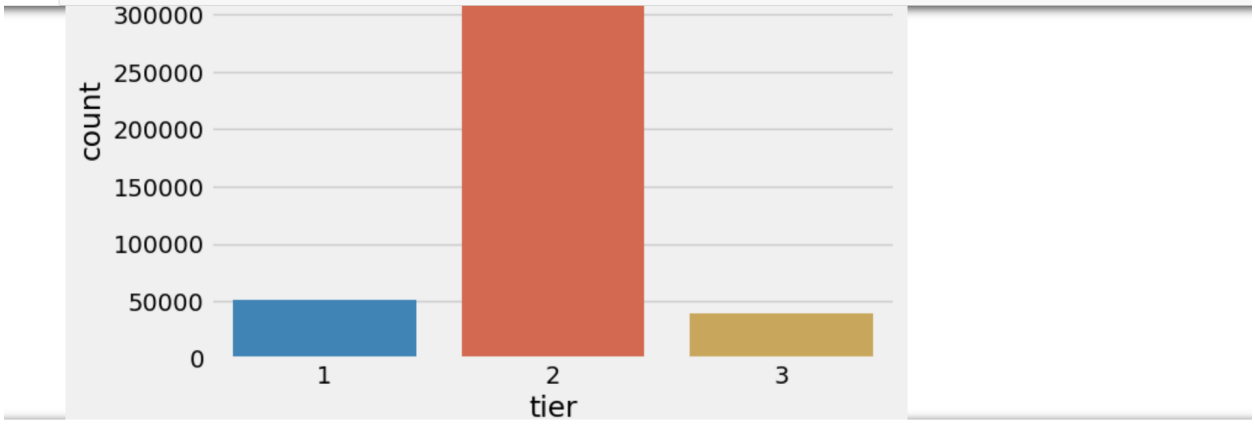


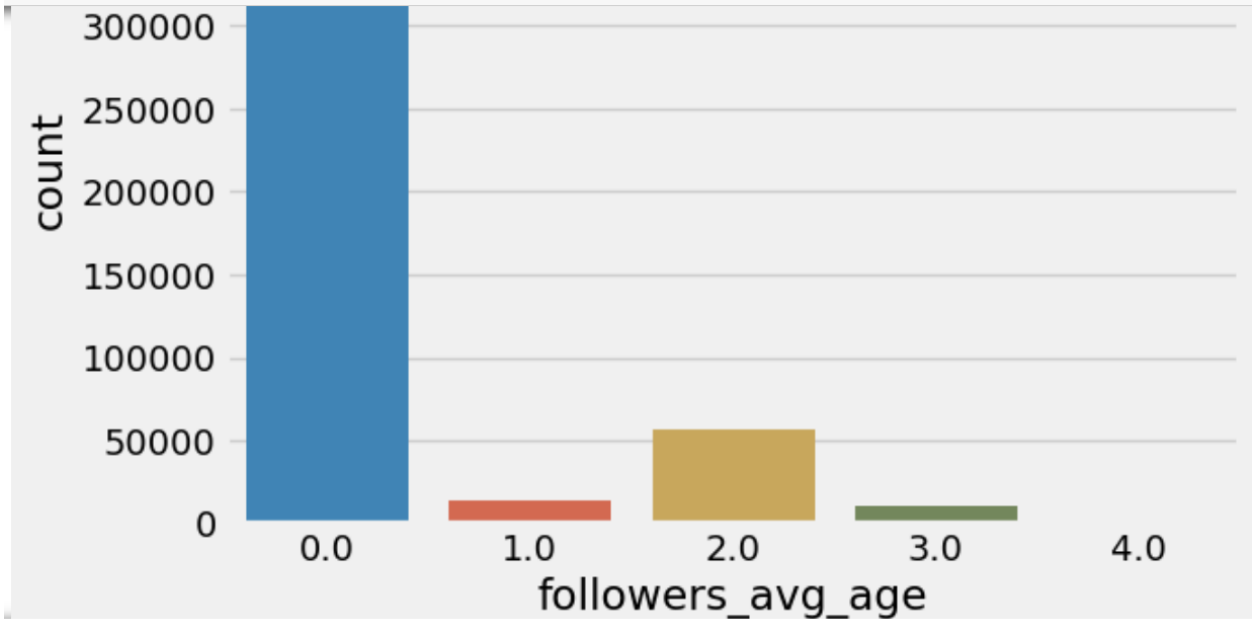
As we can see there is a lot of skew in most of the attributes so we have to perform transformation for sure to reduce the impact of outliers.

For Categorical columns:

ii) Categorical columns

```
[20]: categorical_columns = df.select_dtypes(include='object').columns.tolist()
      for i in categorical_columns:
          sns.countplot(df[i])
          plt.show()
```





As we can see here for the target column the data is defined well for both the classes.

For the columns followers_avg_age, following_avg_age class 1 is represented very less which means users having followers of age_group< 18 is of very less number.

Similarly since there are lot of users do not have any followers the average age

of people following them or average age of people followed by the user is 0 represented by Class-0

We can infer a lot of people from tier 2 are using Trell.

Based on gender we can infer most of the users are males and very less female users for the given data.

5.2 Bivariate:

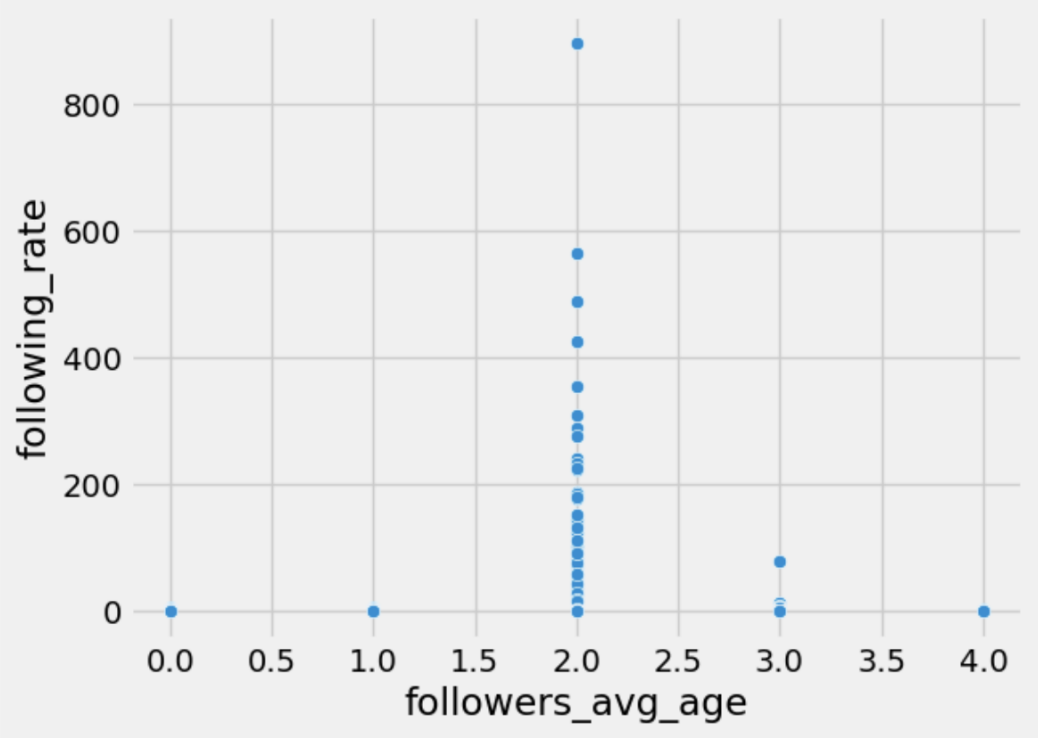
A bivariate plot is a type of graphical representation that displays the relationship between two variables. It is a visualization tool used to explore the association or correlation between two variables, which helps in identifying patterns, trends, and potential relationships between them.

Bivariate plots are commonly used to identify the strength and direction of the relationship between two variables, and to examine how they vary together. Some common types of bivariate plots include scatterplots, bubble charts, and heat maps.

By examining the relationship between two variables through bivariate plots, researchers can draw insights about how changes in one variable affect the other, and vice versa. Bivariate plots are often used in fields such as economics, social sciences, and engineering to understand the relationships between different variables and to make informed decisions based on their characteristics.

Followers_avg_age vs Follwing_rate

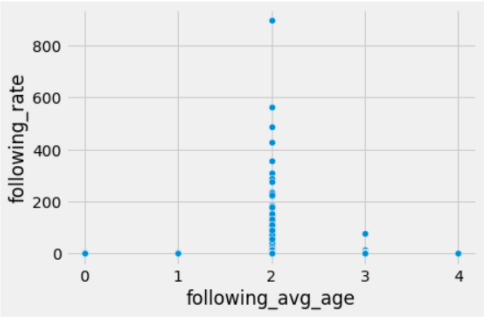
```
sns.scatterplot(y=df['following_rate'],x=df['followers_avg_age'])
plt.show()
```



Based on the above two graphs we have a doubt. When the number of followers = 0 the average followers_age and following_age should be on Class 0 but we have data points on Class 1(<18) ,Class 2(>=18) which is illogical.

Following_avg_age vs Following_rate

In [26]:
```
sns.scatterplot(y=df['following_rate'],x=df['following_avg_age'])
plt.show()
```



In [25]:
```
df[((df['followers_avg_age']==1)|(df['followers_avg_age']==2)|(df['followers_avg_age']==3)|(df['followers_avg_age']==4)) & (df['f
```

Out[25]:

tier	gender	following_rate	followers_avg_age	following_avg_age	max_repetitive_punc	num_of_hashtags_per_action	emoji_count_per_action	punctuations_pe

0 rows × 25 columns

In [26]:
```
df[((df['following_avg_age']==1)|(df['following_avg_age']==2)|(df['following_avg_age']==3)|(df['following_avg_age']==4)) & (df['f
```

Out[26]:

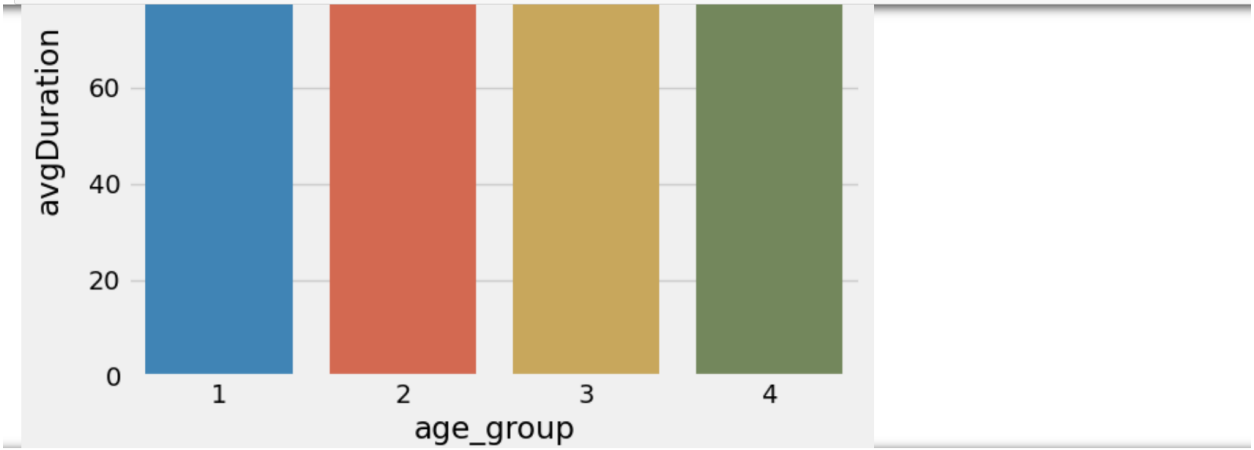
_avg_age	max_repetitive_punc	num_of_hashtags_per_action	emoji_count_per_action	punctuations_per_action	number_of_words_per_action	...	content_views	nu

Based on the above analysis we understand that there are no users who have 0 followers but have average following_age or followers_age in Class 1 or Class 2. They might be values very close to 0.

ii)Categorical vs Numerical

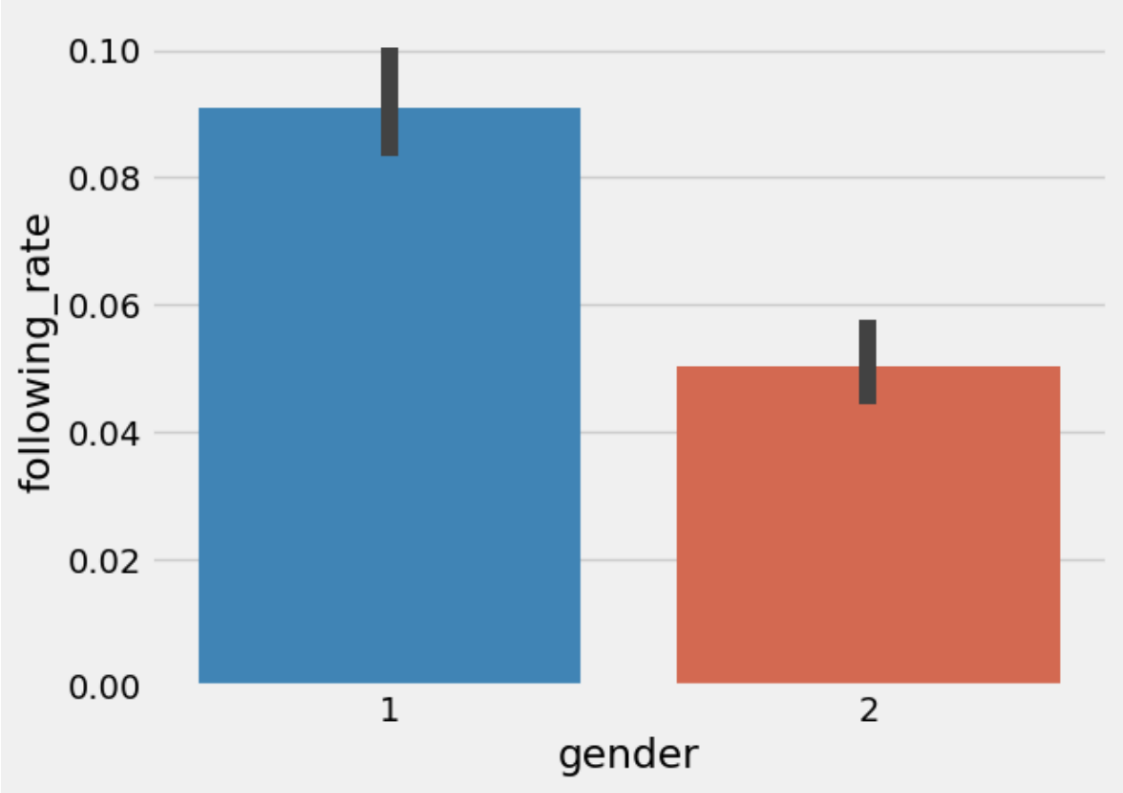
Target vs Numerical

```
for i in numerical_columns:
    sns.barplot(y=df[i],x=df['age_group'])
    plt.show()
```



Gender vs Following_rate(Number of followers)

```
sns.barplot(x=df['gender'],y=df['following_rate'])
plt.show()
```

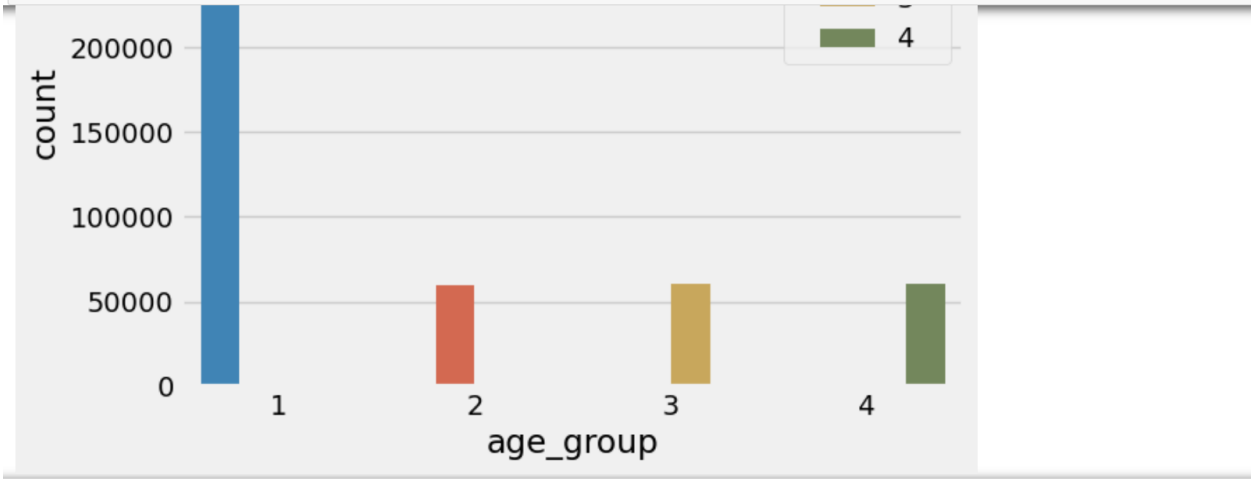


As we can see males have more followers than females.

iii) Categorical vs Categorical

Target vs Categorical

```
for i in categorical_columns:
    sns.countplot(df[i],hue=df['age_group'])
    plt.show()
```



1) Tier vs Age_group - In general we have more users who are of age less than 18 but in tier 1 we can see there are a lot of users who are above 18 years than below 18 years.

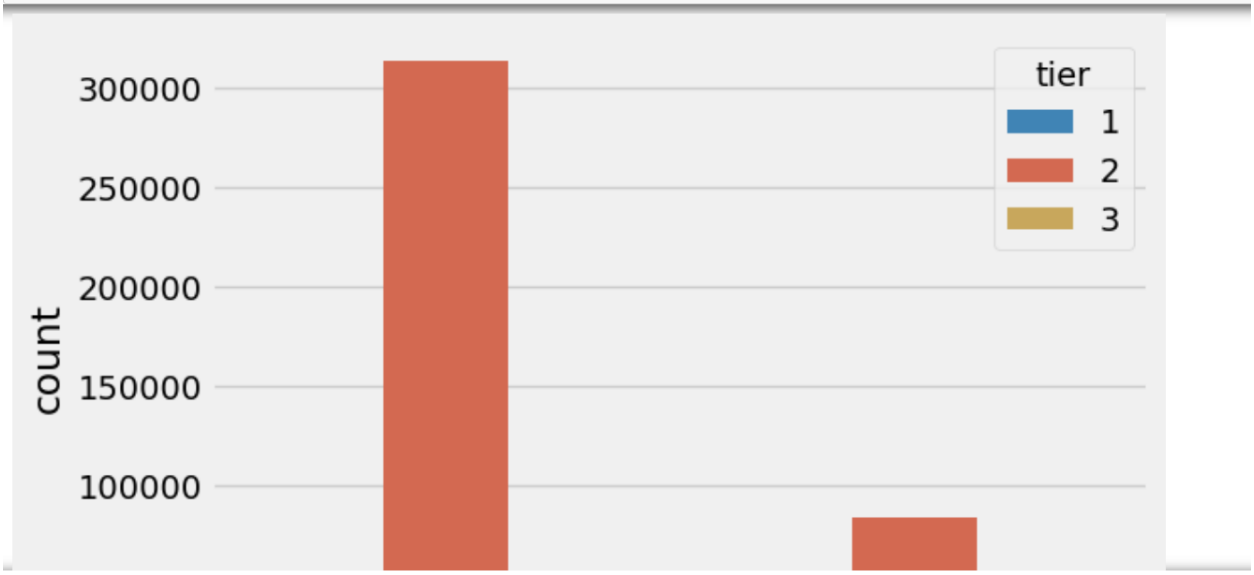
2)Gender vs Age_group - we might see there's same amount of variation in both the age_groups in terms of gender i.e. Proportion of Age_group >18:<18 is equal in both genders

3) As we can see here there might be a lot of new users since they don't have any followers because of which the average_age is also 0.
Also we can see there are people who are < 18 have more followers > 18 and users who are > 18 also have followers who are > 18.
This tells us that people who are >18 follow people irrespective of age.

4)Similarly we can see there are people who are < 18 are following more users > 18 and users who are > 18 are also following more users who are > 18. It might give us an insight that users > 18 are probably more interested in socializing than users < 18.

Gender vs Tier

```
sns.countplot(df['gender'],hue=df['tier'])
plt.show()
```



We can infer that there are almost always male, female users from Tier 1 and Tier 3.

5.3 Multivariate:

A multivariate plot is a type of graphical representation that displays the relationship between three or more variables. It is a visualization tool used to explore the complex relationships between multiple variables, which helps in identifying patterns, trends, and potential relationships between them.

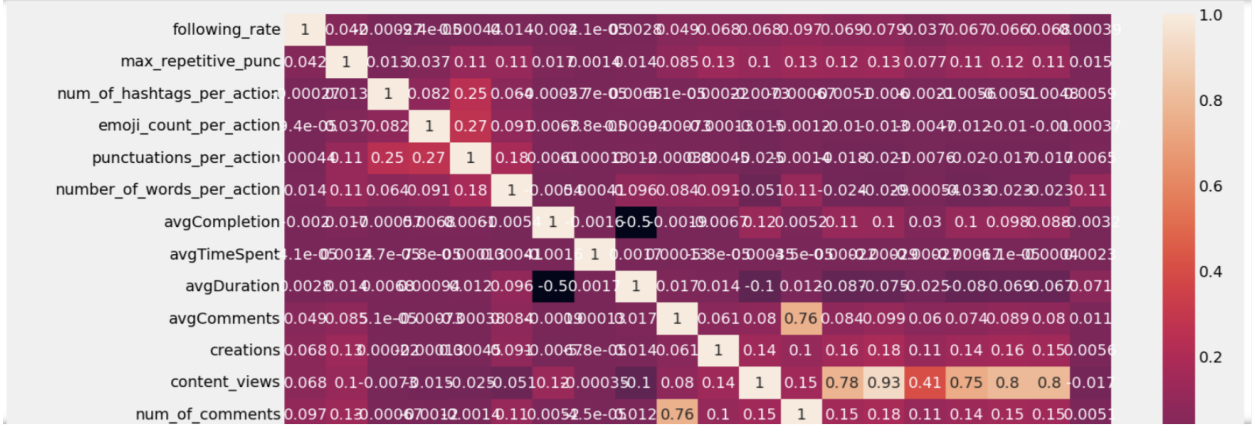
Multivariate plots are commonly used in fields such as data science, finance, and social sciences to examine the relationships between multiple variables simultaneously. Some common types of multivariate plots include scatterplot matrices, 3D scatterplots, parallel coordinate plots, and heat maps.

By examining the relationship between multiple variables through multivariate plots, researchers can draw insights about how changes in one variable affect the others and how the variables interact with each other. This can help in identifying hidden patterns or trends that may not be apparent from univariate or bivariate plots.

```
df.corr()
```

	following_rate	max_repetitive_punc	num_of_hashtags_per_action	emoji_count_per_action	punctuations_per_action	number_of_wo
following_rate	1.000000	0.042046	-0.000267	-0.000094	0.000437	
max_repetitive_punc	0.042046	1.000000	0.013202	0.036628	0.107544	
num_of_hashtags_per_action	-0.000267	0.013202	1.000000	0.082216	0.246677	
emoji_count_per_action	-0.000094	0.036628	0.082216	1.000000	0.268548	
punctuations_per_action	0.000437	0.107544	0.246677	0.268548	1.000000	
number_of_words_per_action	0.013535	0.113900	0.063917	0.091448	0.178658	
avgCompletion	-0.001991	0.017007	-0.000569	0.006767	0.006133	
avgTimeSpent	-0.000041	0.001450	-0.000027	-0.000078	-0.000129	
avgDuration	0.002808	0.014316	0.006848	0.000940	0.012055	
avgComments	0.048721	0.085199	0.000051	-0.000731	-0.000375	
creations	0.068423	0.128265	0.000220	-0.000131	0.000455	
content_views	0.068105	0.104363	-0.007262	-0.014948	-0.025064	
num_of_comments	0.097322	0.134459	-0.000665	-0.001184	-0.001434	
ekends_trails_watched_per_day	0.069426	0.124927	-0.005060	-0.010307	-0.017546	
ekdays_trails_watched_per_day	0.078755	0.132210	-0.006013	-0.012633	-0.020994	
slot1_trails_watched_per_day	0.037387	0.076631	-0.002130	-0.004726	-0.007591	
slot2_trails_watched_per_day	0.067126	0.109411	-0.005646	-0.011646	-0.019514	
slot3_trails_watched_per_day	0.066273	0.117538	-0.005094	-0.010229	-0.017457	
slot4_trails_watched_per_day	0.067535	0.112034	-0.004785	-0.010284	-0.017003	
avgt2	0.000386	0.015238	0.005928	0.000373	0.006542	

```
plt.figure(figsize=(15,10))
sns.heatmap(df.corr(),annot=True)
plt.show()
```



Based on the observation we can see most of the columns have very less correlation with each other. So we can expect less multicollinearity in the dataset.

Here we can see avg_duration of video is negatively correlated to avg_completion. This tells us users are interested in completing shorter videos rather than watching long videos. This might be the reason Instagram reels, Youtube shorts and such similar features are famous

The features content_views and weekdays_trials_watched_per_day are highly correlated.

5.4 Transformation:

In data analysis and statistics, transformation refers to the process of applying a mathematical function to a variable or set of variables to change their distribution or scale. The purpose of transformation is to make the data more suitable for analysis by improving normality, reducing skewness, or reducing the influence of outliers.

There are several types of transformations that are commonly used in data analysis, including:

- Logarithmic transformation: This involves taking the logarithm of a variable. It is commonly used to reduce skewness in data.

- Square root transformation: This involves taking the square root of a variable. It is also used to reduce skewness and improve normality in data.

- Box-Cox transformation: This is a family of transformations that includes both logarithmic and power transformations. The purpose of the Box-Cox transformation is to identify the most appropriate transformation for the data based on the likelihood function.

- Z-score transformation: This involves subtracting the mean of a variable and dividing by its standard deviation. The purpose of this transformation is to standardize the data and make it easier to compare across variables.

- Min-max scaling: This involves rescaling the data to a specific range, usually between 0 and 1. The purpose of this transformation is to make the data more interpretable and to reduce the influence of outliers.

- The Johnson transformation is often used in statistical analysis to make the data more suitable for parametric tests that assume normality, such as t-tests and ANOVA. It can also be used in machine learning applications where normality is a requirement for certain models.

Transformations can be applied to individual variables or to multiple variables simultaneously. The choice of transformation depends on the characteristics of the data and the goals of the analysis. It is important to carefully consider the implications of any transformation and to interpret the results of the analysis accordingly.

Separating the numeric and categorical columns:

```
X = df.drop('age_group',axis=1)
y = df[['age_group']]
```

```
X_num = X[['following_rate','max_repetitive_punc',
                        'num_of_hashtags_per_action','emoji_count_per_action','punctuations_per_action',
                        'number_of_words_per_action','avgCompletion','avgTimeSpent','avgDuration','avgComments',
                        'creations','content_views','num_of_comments','weekends_trails_watched_per_day',
                        'weekdays_trails_watched_per_day','slot1_trails_watched_per_day','slot2_trails_watched_per_day',
                        'slot3_trails_watched_per_day','slot4_trails_watched_per_day','avgt2']]
X_cat = X[['tier','gender','followers_avg_age','following_avg_age']]
```

Checking skewness before transformation:

```
X_num.skew()
```

following_rate	182.220682
max_repetitive_punc	41.830434
num_of_hashtags_per_action	98.282348
emoji_count_per_action	49.421971
punctuations_per_action	56.131696
number_of_words_per_action	149.173174
avgCompletion	0.623219
avgTimeSpent	677.458265
avgDuration	5.300170
avgComments	239.802805
creations	189.998644
content_views	9.849459
num_of_comments	109.570695
weekends_trails_watched_per_day	10.635757
weekdays_trails_watched_per_day	12.721963
slot1_trails_watched_per_day	26.337319
slot2_trails_watched_per_day	11.380347
slot3_trails_watched_per_day	13.531891
slot4_trails_watched_per_day	16.179045

As we can see we have a high value of skew for most of the columns. So we apply yeo-johnson transformation technique to make sure the data is normalized as much as possible. The reason we apply yeo-johnson is that after scaling the data will have both positive and negative values. So yeo-johnson will be the best method of transformation in such a scenario.

Yeo - Johnson Transformation:

```
from sklearn.preprocessing import PowerTransformer
pt = PowerTransformer(method='yeo-johnson')
df_transformed = pd.DataFrame(pt.fit_transform(X_num),columns=X_num.columns)
df_transformed.head()
```

	following_rate	max_repetitive_punc	num_of_hashtags_per_action	emoji_count_per_action	punctuations_per_action	number_of_words_per_action	avgCompleti
0	-0.56388	-0.40533	-0.047542	-0.09579	-0.236055	-0.776454	0.7063
1	-0.56388	-0.40533	-0.047542	-0.09579	4.452601	0.832302	0.5497
2	-0.56388	-0.40533	-0.047542	-0.09579	-0.236055	-0.776454	0.1146
3	-0.56388	-0.40533	-0.047542	-0.09579	-0.236055	-0.776454	-2.0810
4	-0.56388	-0.40533	-0.047542	-0.09579	-0.236055	-0.776454	0.6754

Checking skewness after transformation:

```
df_transformed.skew()
```

```
following_rate                    1.733886
max_repetitive_punc               2.061848
num_of_hashtags_per_action       20.990729
emoji_count_per_action           10.349309
punctuations_per_action           4.068353
number_of_words_per_action        0.882614
avgCompletion                     0.010438
avgTimeSpent                     -0.077770
avgDuration                      -0.010749
avgComments                       2.854754
creations                         1.087440
content_views                     0.553165
num_of_comments                   2.928127
weekends_trails_watched_per_day   0.989292
weekdays_trails_watched_per_day   0.839751
slot1_trails_watched_per_day      2.126087
slot2_trails_watched_per_day      0.890366
slot3_trails_watched_per_day      0.901285
slot4_trails_watched_per_day      0.896502
avgt2                             0.123972
dtype: float64
```

The skew is high for 'num_of_hashtags_per_action' even after transformation which indicates that many people are not using hashtags

5.5 Encoding

Many statistical and machine learning algorithms require numerical inputs, and categorical variables need to be transformed into numerical values before they can be used in these algorithms. Encoding allows us to convert the categorical variables into a form that can be processed by these algorithms.

In our dataset, all the categorical columns are already encoded so there is no need to perform encoding. We converted them into object data type for analysis
purpose. Now, we are converting them back into numerical data type as all the variables should be numeric for model building.

Encoding the data

As we can see our categorical columns are already encoded we did not have to perform any encoding. Only thing we have to do here is to convert these category clolumns from object datatype to integer datatype

In [38]:
```
X_cat['gender']= X_cat['gender'].astype(np.number)
X_cat['tier']= X_cat['tier'].astype(np.number)
X_cat['followers_avg_age']= X_cat['followers_avg_age'].astype(np.number)
X_cat['following_avg_age']= X_cat['following_avg_age'].astype(np.number)
y['age_group']= y['age_group'].astype(int)
```

6. Data Preparation before Model Building

Before model building, we concat the transformed numerical columns and encoded categorical columns into a new dataframe.

We have separated the target variable ‘age_group’ and stored it in the variable ‘y’.

Combining Numerical and Categorical as X

In [39]:
```
df_feature = pd.concat([df_transformed,X_cat],axis=1)
```

In [40]:
```
df_feature.head(1)
```

Out[40]:

	following_rate	max_repetitive_punc	num_of_hashtags_per_action	emoji_count_per_action	punctuations_per_action	number_of_words_per_action	avgCompleti
0	-0.56388	-0.40533	-0.047542	-0.09579	-0.236055	-0.776454	0.7063

1 rows × 24 columns

In [41]:
```
y.head(1)
```

Out[41]:

	age_group
0	1

6.1 Train test split

In machine learning, train-test split refers to the process of splitting a dataset into two parts: one for training a machine learning model, and one for testing the performance of the model.

During training, the machine learning algorithm learns patterns and relationships between input features and output targets in the training set. The testing set is used to evaluate the performance of the model on unseen data. The accuracy of the model on the testing set is an indicator of how well the model is likely to perform on new, unseen data.

We have splitted the dataset into the following ratio :

Training data - 80%
Testing data - 20%

Train Test Split

In [42]:
```
xtrain,xtest,ytrain,ytest = train_test_split(df_feature,y,test_size=0.2,random_state=100)
```

In [43]:
```
print(xtrain.shape)
print(xtest.shape)
print(ytrain.shape)
print(ytest.shape)
```

```
(391101, 24)
(97776, 24)
(391101, 1)
(97776, 1)
```

6.2 Stratified Shuffle Split

Stratified Shuffle Split is a cross-validation technique used in machine learning to split the dataset into training and testing sets. This technique is particularly useful when the target variable is imbalanced or when we want to maintain the proportion of different classes in both the training and testing sets.

By using the Stratified Shuffle Split technique, we can ensure that the model is trained and tested on a representative sample of the data, which helps to avoid overfitting and underfitting. This technique is commonly used in classification problems where the target variable is categorical and unbalanced.

The Stratified Shuffle Split technique can be implemented using the scikit-learn library in Python, which provides a cross-validation module with several functions to split the data.

We are using this because our target column ‘age_group’ is imbalanced.

Separating the train data based on the age group

In [46]:
```
train_data1 = train_data[train_data['age_group']==1]
train_data2 = train_data[train_data['age_group']==2]
train_data3 = train_data[train_data['age_group']==3]
train_data4 = train_data[train_data['age_group']==4]
```

The count of age_group 1 is more compared to other age_groups. Hence we are using the stratified shuffle split to split the age_group 1 into three equal proportions.

In [47]:
```
# create samples of class 1 empty

group1_sample1 = None
group1_sample2 = None
group1_sample3 = None

sss=StratifiedShuffleSplit(n_splits=1,test_size=0.33,random_state=10)
for train_index , test_index in sss.split(train_data1,train_data1['age_group']):
    group1_sample1 = train_data1.iloc[test_index]
    group1_sample1_remaining = train_data1.iloc[train_index]
```

In [48]:
```
group1_sample1.shape
```

Out[48]: (81411, 25)

In [49]:
```
group1_sample1_remaining.shape
```

Out[49]: (165289, 25)

In [50]:
```
sss=StratifiedShuffleSplit(n_splits=1,test_size=0.5,random_state=10)
for train_index , test_index in sss.split(group1_sample1_remaining,group1_sample1_remaining['age_group']):
    group1_sample2 = group1_sample1_remaining.iloc[test_index]
    group1_sample3 = group1_sample1_remaining.iloc[train_index]
```

In [51]:
```
group1_sample2.shape
```

Out[51]: (82645, 25)

In [52]:
```
group1_sample3.shape
```

Out[52]: (82644, 25)

Concating the splitted age_group 1 dataframe with the other three age_group dataframes(train_data2,train_data3 and train_data4]) and creating three new data frames as df1_final, df2_final and df3_final.

In [53]:
```
df_1 = pd.concat([group1_sample1,train_data2,train_data3,train_data4])
```

In [54]:
```
df1_final = df_1.sample(frac=1).reset_index(drop=True)
df1_final['age_group'].value_counts()
```

Out[54]:
```
1    81411
4    48584
3    48341
2    47476
Name: age_group, dtype: int64
```

In [55]:
```
df_2 = pd.concat([group1_sample2,train_data2,train_data3,train_data4])
```

In [56]:
```
df2_final = df_2.sample(frac=1).reset_index(drop=True)
df2_final['age_group'].value_counts()
```

Out[56]:
```
1    82645
4    48584
3    48341
2    47476
Name: age_group, dtype: int64
```

In [57]:
```
df_3 = pd.concat([group1_sample3,train_data2,train_data3,train_data4])
```

In [58]:
```
df3_final = df_3.sample(frac=1).reset_index(drop=True)
df3_final['age_group'].value_counts()
```

Out[58]:
```
1    82644
4    48584
3    48341
2    47476
Name: age_group, dtype: int64
```

Splitting the new data frames df1_final, df2_final and df3_final as (x1, y1), (x2,y2) and (x3,y3) for model building.

```
In [59]:
1 x1 = df1_final.drop('age_group',axis=1)
2 y1 = df1_final['age_group']

In [60]:
1 x2 = df2_final.drop('age_group',axis=1)
2 y2 = df2_final['age_group']

In [68]:
1 x3 = df3_final.drop('age_group',axis=1)
2 y3 = df3_final['age_group']
```

6.3 Visualization for each sample

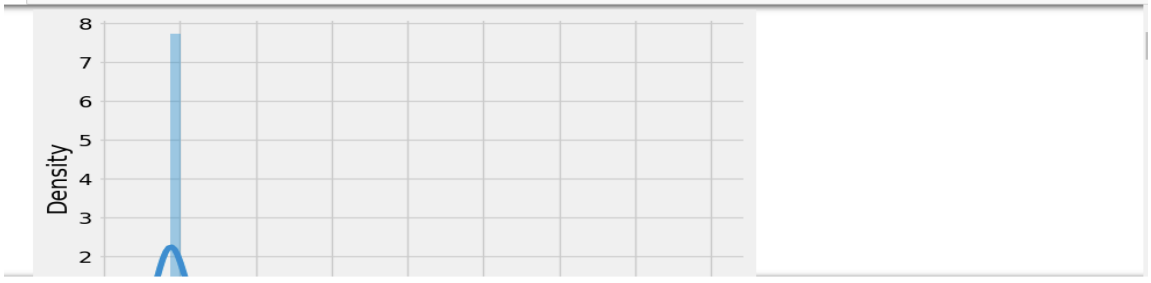
As we split the data into 3 samples, Now we want to do the data visualization over all the three samples.

Data Visualization of Sample 1:

Numerical columns

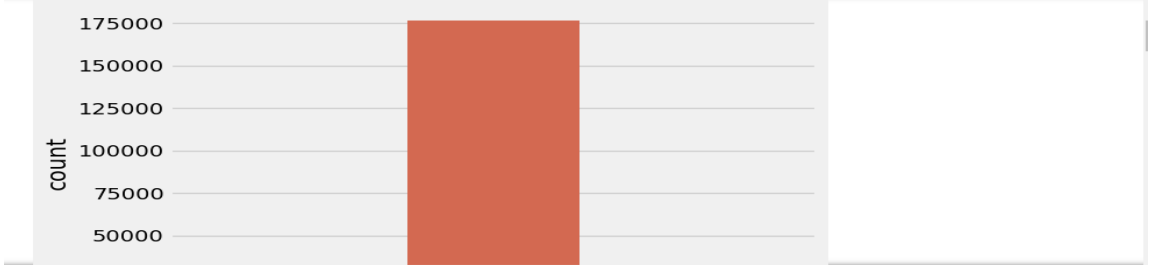
1st sample

```
]: for i in df1_final:
    if i not in categorical_columns:
        sns.distplot(df1_final[i])
        plt.show()
```



Categorical columns

```
]: for i in df1_final:
    if i in categorical_columns:
        sns.countplot(df1_final[i])
        plt.show()
```



Bivariate analysis

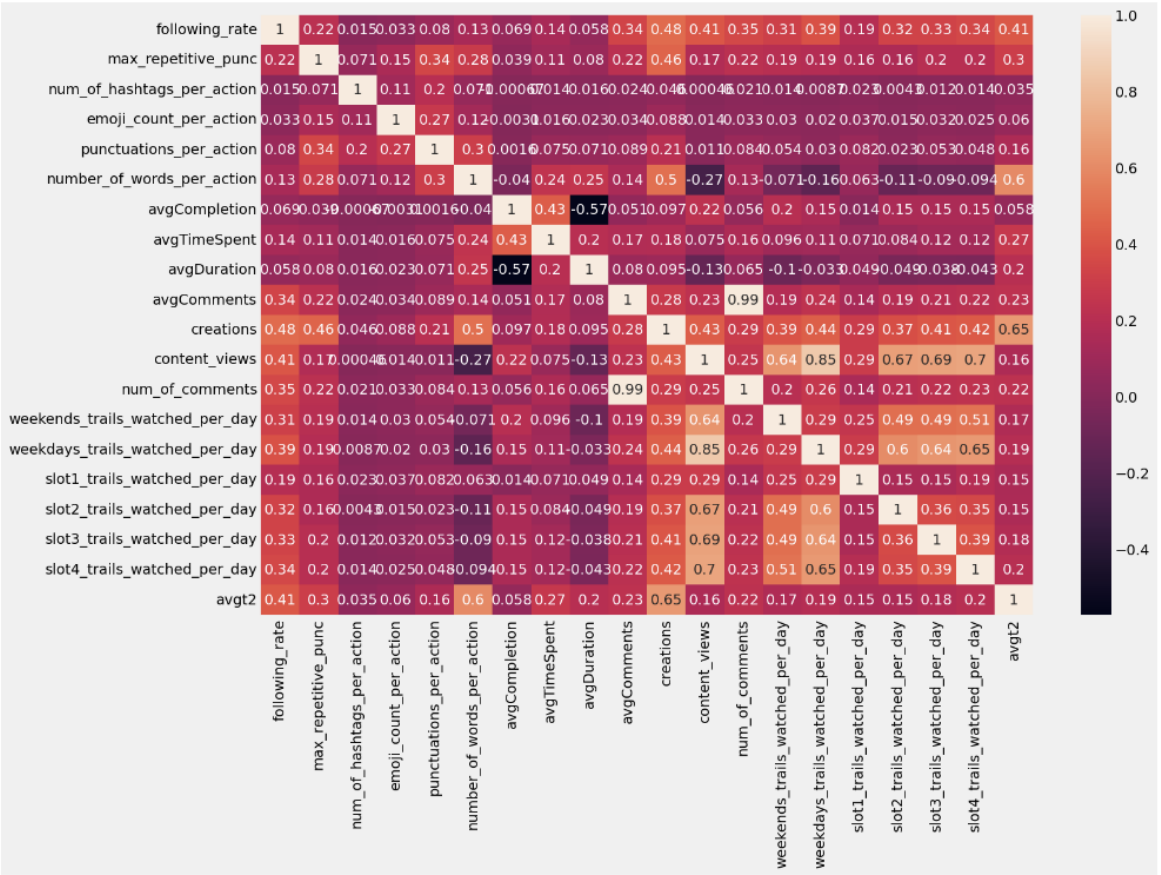
Numerical vs Numerical

In [59]:
```
df1_final[numerical_columns].corr()
```

Out[59]:

	following_rate	max_repetitive_punc	num_of_hashtags_per_action	emoji_count_per_action	punctuations_per_action	num
following_rate	1.000000	0.217162	0.014975	0.032571	0.079678	
max_repetitive_punc	0.217162	1.000000	0.071413	0.147762	0.339677	
num_of_hashtags_per_action	0.014975	0.071413	1.000000	0.111910	0.202793	
emoji_count_per_action	0.032571	0.147762	0.111910	1.000000	0.270050	
punctuations_per_action	0.079678	0.339677	0.202793	0.270050	1.000000	
number_of_words_per_action	0.133176	0.284639	0.070758	0.119968	0.304459	
avgCompletion	0.069486	0.039158	-0.000671	-0.003138	0.001567	
avgTimeSpent	0.137446	0.109069	0.014391	0.016366	0.074532	
avgDuration	0.058389	0.080189	0.015862	0.022765	0.070562	
avgComments	0.335114	0.218070	0.023523	0.034299	0.088715	

In [60]:
```
plt.figure(figsize=(15,10))
sns.heatmap(df1_final[numerical_columns].corr(),annot=True)
plt.show()
```



Based on the observation we can see most of the columns have very less

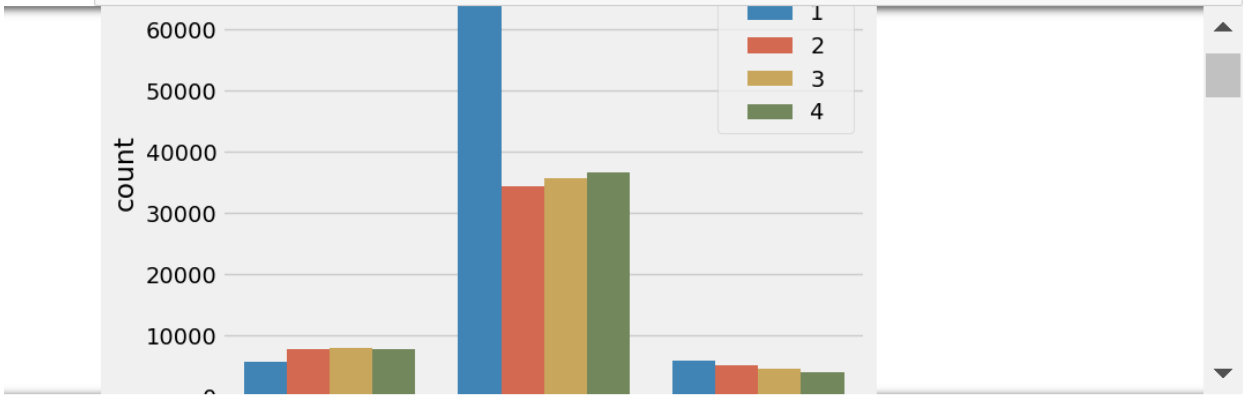
correlation with each other. So we can expect less multicollinearity in the dataset.

Here we can see avg_duration of video is negatively correlated to avg_completion. This tells us users are interested in completing shorter videos rather than watching long videos. This might be the reason Instagram reels, Youtube shorts and such similar features are famous.

The features content_views and weekdays_trials_watched_per_day are highly correlated.

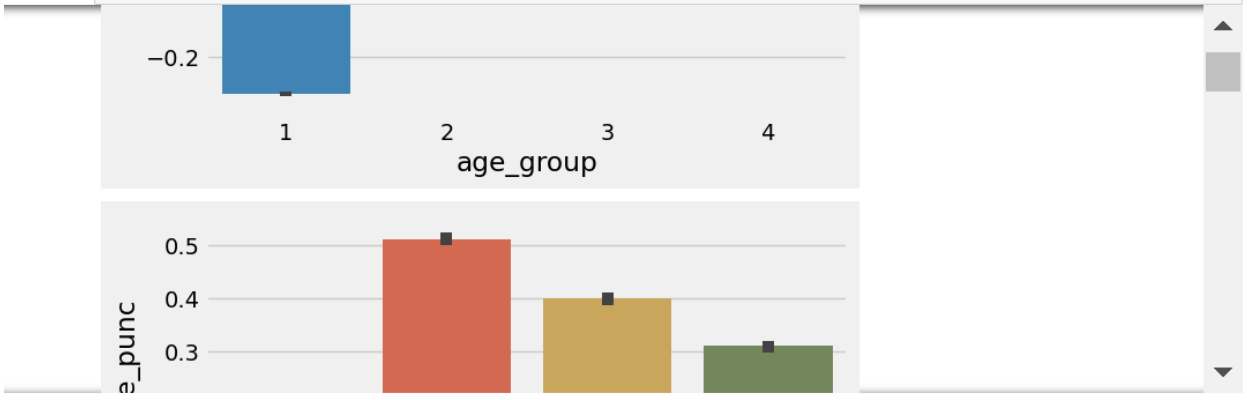
Target vs categorical

```
In [61]: for i in df1_final.iloc[:24]:
             if i in categorical_columns:
                 sns.countplot(hue=df1_final["age_group"],x=df1_final[i])
                 plt.show()
```



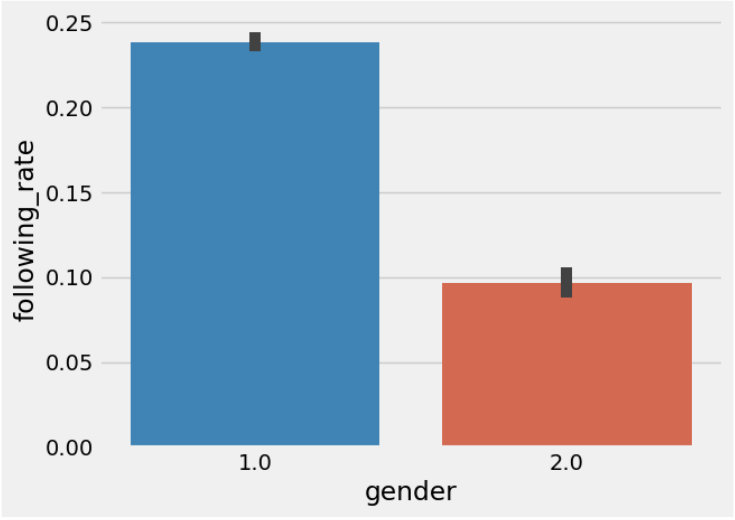
Target vs Numerical

```
In [62]: for i in df1_final:
             if i in numerical_columns:
                 sns.barplot(x=df1_final["age_group"],y=df1_final[i])
                 plt.show()
```



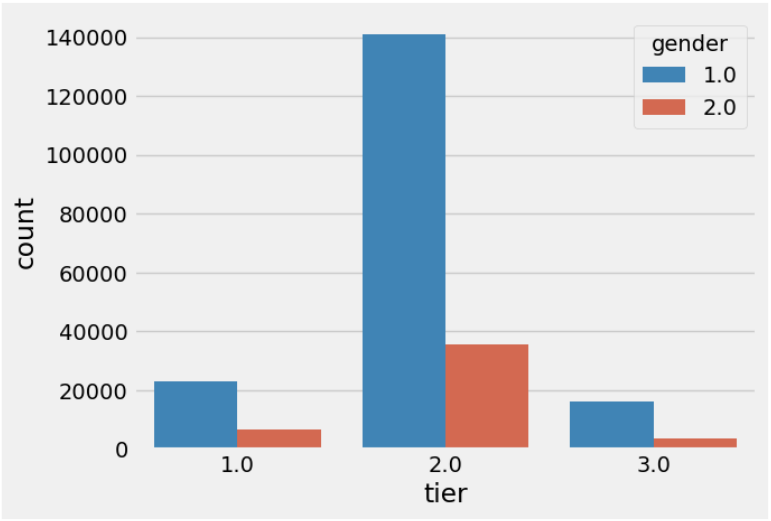
Gender vs following_rate

In [63]:
```
sns.barplot(x=df1_final['gender'],y=df1_final['following_rate'])
plt.show()
```



Gender vs Tier

In [64]:
```
sns.countplot(df1_final["tier"],hue=df1_final["gender"])
plt.show()
```



Similarly, we have performed the visualization for the remaining 2 samples.

8. Model Building:

8.1 Decision Tree before undersampling

A decision tree is a predictive model in machine learning that uses a tree-like structure to model decisions and their possible consequences. It is a type of supervised learning algorithm that can be used for both classification and regression tasks.

In a decision tree, each internal node represents a decision or a test on a feature of the data, and each branch represents the outcome of the test. The leaves of the tree represent the final outcome or prediction.

In [87]:
```
# Decision tree without undersampling

from sklearn.metrics import f1_score,classification_report
```

In [88]:
```
dt = DecisionTreeClassifier(criterion='gini',random_state = 10)
dt.fit(xtrain,ytrain)
y_pred_base = dt.predict(xtest)
y_pred_base_train = dt.predict(xtrain)
```

In [89]:
```
f1_score(ytest,y_pred_base,average= 'weighted')
```

Out[89]:
```
0.6928659313469843
```

In [91]:
```
print(classification_report(ytrain,y_pred_base_train))
print(classification_report(ytest,y_pred_base))
```

```
              precision    recall  f1-score   support

           1       1.00      1.00      1.00    246700
           2       1.00      1.00      1.00     47476
           3       1.00      1.00      1.00     48341
           4       1.00      1.00      1.00     48584

    accuracy                           1.00    391101
   macro avg       1.00      1.00      1.00    391101
weighted avg       1.00      1.00      1.00    391101

              precision    recall  f1-score   support

           1       0.88      0.88      0.88     61615
           2       0.42      0.42      0.42     11879
           3       0.35      0.35      0.35     12063
           4       0.35      0.35      0.35     12219

    accuracy                           0.69     97776
   macro avg       0.50      0.50      0.50     97776
weighted avg       0.69      0.69      0.69     97776
```

As we can see the model is overfitting and most importantly Class 1 is predicted well. So we proceed with ensemble methods to generalize the model and predict equally for all the classes.

8.2 Voting Classification:

Voting classification is a type of ensemble learning method in machine learning where multiple models are used to make a prediction by taking a majority vote. In other words, each model predicts the target variable independently, and the final prediction is the one that receives the most votes. There are two types of voting classification: hard voting and soft voting.

In hard voting, each model in the ensemble predicts a class label, and the class with the most votes is selected as the final prediction. This method works well when the models in the ensemble have similar performance and there is little disagreement among them.

In soft voting, each model predicts a probability distribution over the classes, and the probabilities are averaged across all models. The class with the highest average probability is selected as the final prediction. Soft voting works better than hard voting when the models in the ensemble have different strengths or when the data is noisy.

Voting classification can be used with any type of machine learning algorithm, including decision trees, logistic regression, support vector machines, and neural networks. It is often used in practice to improve the accuracy and robustness of the predictions by combining the strengths of multiple models.

Here we are using hard voting and checking for f1 scores.

Checking using Voting Classifier

```
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import VotingClassifier
from sklearn.naive_bayes import GaussianNB
```

```
model1 = DecisionTreeClassifier()
model2 = LogisticRegression()
model3 = GaussianNB()

# Create an ensemble of models using a voting classifier
voting_ensemble = VotingClassifier(estimators=[('dt', model1), ('lr', model2), ('nb', model3)], voting='hard')
```

```
voting_ensemble.fit(x1,y1)
# Make predictions on the test set using the ensemble model
y_pred_voting = voting_ensemble.predict(xtest)
```

```
f1_score(ytest,y_pred_voting,average='weighted')
```

```
0.6744871841207234
```

8.3 Logistic Regression

When we want to understand the relationship between one or more predictor variables and a continuous response variable, we often use linear regression. However, when the response variable is categorical we can instead use logistic regression.

Logistic regression is a type of classification algorithm because it attempts to "classify" observations from a dataset into distinct categories.

Here are a few examples of when we might use logistic regression:

- We want to use credit score and bank balance to predict whether or not a given customer will default on a loan. (Response variable = "Default" or "No default")
- We want to use average rebounds per game and average points per game to predict whether or not a given basketball player will get drafted into the NBA (Response variable = "Drafted" or "Not Drafted")
- We want to use square footage and the number of bathrooms to predict whether or not a house in a certain city will be listed at a selling price of $200k or more. (Response variable = "Yes" or "No")

Notice that the response variable in each of these examples can only take on one of two values. Contrast this with linear regression in which the response variable takes on some continuous value.

The Logistic Regression Equation

Logistic regression uses a method known as maximum likelihood estimation (details will not be covered here) to find an equation of the following form:

$$\log[p(X) / (1-p(X))] = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_p X_p$$

where:

X_j: The jth predictor variable
β_j: The coefficient estimate for the jth predictor variable
The formula on the right side of the equation predicts the log odds of the response variable taking on a value of 1.

Thus, when we fit a logistic regression model we can use the following equation to calculate the probability that a given observation takes on a value of 1:

$$p(X) = e\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_p X_p / (1 + e\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_p X_p)$$

We then use some probability threshold to classify the observation as either 1 or 0.

For example, we might say that observations with a probability greater than or equal to 0.5 will be classified as “1” and all other observations will be classified as “0.”

Assumptions of Logistic Regression

Logistic regression uses the following assumptions:

1. The response variable is binary. It is assumed that the response variable can only take on two possible outcomes.

2. The observations are independent. It is assumed that the observations in the dataset are independent of each other. That is, the observations should not come from repeated measurements of the same individual or be related to each other in any way.

3. There is no severe multicollinearity among predictor variables. It is assumed that none of the predictor variables are highly correlated with each other.

4. There are no extreme outliers. It is assumed that there are no extreme outliers or influential observations in the dataset.

5. There is a linear relationship between the predictor variables and the logit of the response variable. This assumption can be tested using a Box-Tidwell test.

6. The sample size is sufficiently large. As a rule of thumb, you should have a minimum of 10 cases with the least frequent outcome for each explanatory variable. For example, if you have 3 explanatory variables and the expected probability of the least frequent outcome is 0.20, then you should have a sample size of at least (10*3) / 0.20 = 150.

Below are the python snippets of models built with 3 different samples

Logistic Regression

```
In [92]: from sklearn.linear_model import LogisticRegression
```

1st sample

```
In [93]: lr = LogisticRegression(random_state = 10)
         lr.fit(x1,y1)
         y_pred1 = lr.predict(xtest)
         y_pred_train = lr.predict(x1)

In [97]: f1_score(ytest,y_pred1,average= 'weighted')

Out[97]: 0.6960263355569551
```

2nd sample

```
In [98]: lr = LogisticRegression(random_state = 10)
         lr.fit(x2,y2)
         y_pred2 = lr.predict(xtest)
         y_pred_train = lr.predict(x2)

In [99]: f1_score(ytest,y_pred2,average= 'weighted')

Out[99]: 0.6959819176581605
```

3rd sample

```
In [100]: lr = LogisticRegression(random_state = 10)
          lr.fit(x3,y3)
          y_pred3 = lr.predict(xtest)
          y_pred_train = lr.predict(x3)

In [101]: f1_score(ytest,y_pred3,average= 'weighted')

Out[101]: 0.6959144358986286
```

Interpretation of Logistic Regression Models with different samples

The F1 scores calculated for different samples indicate the performance of the logistic regression model trained on each sample. The F1 score is a measure of the model's accuracy that takes into account both precision and recall.

A high F1 score indicates that the model is performing well on the given sample and has good precision and recall. Conversely, a low F1 score indicates that the model's performance is poor and it may not be accurately predicting the target variable.

Comparing the F1 scores for different samples can help identify which sample is providing the best results and therefore can be used to make the most accurate predictions on new data.

In the above models, F1 scores obtained from different samples have been more or less similar.

In the F1 score function, the parameter ‘average’ determines how to calculate the overall F1 score when there are multiple classes. The **average='weighted'** option means that the F1 score is calculated as the weighted average of the F1 scores for each class, where the weights are determined by the number of samples in each class.This means that the F1 score for each class is weighted by the number of samples in that class, giving more importance to classes with more samples.

The average='weighted' option is a good choice when you have imbalanced classes, as it ensures that the F1 score takes into account the contribution of each class to the overall performance of the model.

8.4 Decision Tree Classification

Decision Tree is a Supervised learning technique that can be used for both classification and Regression problems, but ***mostly it is preferred for solving Classification problems.*** It is a tree-structured classifier, where internal nodes represent the features of a dataset, branches represent the decision rules and each leaf node represents the outcome.In a Decision tree, there are two nodes, which are the Decision Node and Leaf Node. Decision nodes are used to make any decision and have multiple branches, whereas Leaf nodes are the output of those decisions and do not contain any further branches.The decisions or the test are performed on the basis of features of the given dataset.It is a graphical representation for getting all the possible solutions to a problem/decision based on given conditions.It is called a decision tree because, similar to a tree, it starts with the root node, which expands on further branches and constructs a tree-like structure.

Below are the python snippets of models built with different samples

Decision Tree

For 1st sample

```
from sklearn.tree import DecisionTreeClassifier
dt = DecisionTreeClassifier(criterion='gini',random_state = 10)
dt.fit(x1,y1)
y_pred1 = dt.predict(xtest)
y_pred_train = dt.predict(x1)
```

```
from sklearn.metrics import classification_report
print(classification_report(y1,y_pred_train))
print(classification_report(ytest,y_pred1))
```

```
              precision    recall  f1-score   support

           1       1.00      1.00      1.00     81411
           2       1.00      1.00      1.00     47476
           3       1.00      1.00      1.00     48341
           4       1.00      1.00      1.00     48584

    accuracy                           1.00    225812
   macro avg       1.00      1.00      1.00    225812
weighted avg       1.00      1.00      1.00    225812

              precision    recall  f1-score   support

           1       0.94      0.84      0.89     61615
           2       0.40      0.47      0.43     11879
           3       0.34      0.40      0.36     12063
           4       0.34      0.40      0.37     12219

    accuracy                           0.69     97776
   macro avg       0.50      0.53      0.51     97776
weighted avg       0.73      0.69      0.70     97776
```

```
from sklearn.metrics import f1_score
f1_score(ytest,y_pred1,average= 'weighted')
```

```
0.7041084480964619
```

For 2nd sample

```
from sklearn.tree import DecisionTreeClassifier
dt = DecisionTreeClassifier(criterion='gini',random_state = 10)
dt.fit(x2,y2)
y_pred2 = dt.predict(xtest)
y_pred_train = dt.predict(x2)
```

```
from sklearn.metrics import classification_report
print(classification_report(y2,y_pred_train))
print(classification_report(ytest,y_pred2))
```

```
              precision    recall  f1-score   support

           1       1.00      1.00      1.00     82645
           2       1.00      1.00      1.00     47476
           3       1.00      1.00      1.00     48341
           4       1.00      1.00      1.00     48584

    accuracy                           1.00    227046
   macro avg       1.00      1.00      1.00    227046
weighted avg       1.00      1.00      1.00    227046

              precision    recall  f1-score   support

           1       0.94      0.85      0.89     61615
           2       0.39      0.46      0.42     11879
           3       0.34      0.39      0.37     12063
           4       0.34      0.40      0.36     12219

    accuracy                           0.69     97776
   macro avg       0.50      0.52      0.51     97776
weighted avg       0.72      0.69      0.70     97776
```

```
from sklearn.metrics import f1_score
f1_score(ytest,y_pred2,average= 'weighted')
```

```
0.7027161172826492
```

For 3rd sample

```
from sklearn.tree import DecisionTreeClassifier
dt = DecisionTreeClassifier(criterion='gini',random_state = 10)
dt.fit(x3,y3)
y_pred3 = dt.predict(xtest)
y_pred_train = dt.predict(x3)
```

```
from sklearn.metrics import classification_report
print(classification_report(y3,y_pred_train))
print(classification_report(ytest,y_pred3))
```

```
              precision    recall  f1-score   support

           1       1.00      1.00      1.00     82644
           2       1.00      1.00      1.00     47476
           3       1.00      1.00      1.00     48341
           4       1.00      1.00      1.00     48584

    accuracy                           1.00    227045
   macro avg       1.00      1.00      1.00    227045
weighted avg       1.00      1.00      1.00    227045

              precision    recall  f1-score   support

           1       0.94      0.85      0.89     61615
           2       0.40      0.46      0.43     11879
           3       0.34      0.39      0.36     12063
           4       0.34      0.40      0.37     12219

    accuracy                           0.69     97776
   macro avg       0.50      0.52      0.51     97776
weighted avg       0.72      0.69      0.70     97776
```

```
from sklearn.metrics import f1_score
f1_score(ytest,y_pred3,average= 'weighted')
```

```
0.7030984573891249
```

Interpretation of above built models

Decision Tree models are built with the criterion parameter set to 'gini'. The 'gini' criterion is used to measure the quality of the split. The random_state parameter is set to 10 to ensure reproducibility of the results. In all of the three models built, class 1 is predicted well on the test data whereas the other classes namely 2,3,4 are not predicted well. F1 score combines precision and recall into a single score

that reflects the trade-off between the two.The F1 score is defined as the harmonic mean of precision and recall.The F1 score is a useful metric for imbalanced datasets where one class is much rarer than the other, as it takes into account both precision and recall for each class.

It has been used in conjunction with the classification report so that metric accuracy could be compared and a more comprehensive view of the classifier model could be understood.

9. Advance Model Building:

9.1 Algorithm for Voting:

A data frame ‘ypred_random’ has been created using the ypred’s obtained from the 3 models.

On comparing the three ypred’s, we got the final ypred by using the below algorithm :

- If all three model predictions are the same then set the final predicted label for that row to be the label predicted by the three models.
- Else, if two out of the three model predictions are the same then set the final predicted label for that row to be the label predicted by the two models.
- Else, if two models have the same prediction but not the third model then set the final predicted label

9.2 Random Forest Classification:

Random Forest Classifier is a supervised learning algorithm in machine learning that is based on the ensemble learning technique. It is a type of decision tree-based ensemble algorithm that combines multiple decision trees to improve the performance and reduce the risk of overfitting.

Random Forest Classifier works by creating a forest of decision trees, where each tree is trained on a subset of the training data using a random sample of the features. This process is called bootstrap aggregating or bagging. The final prediction is made by aggregating the predictions of all the trees in the forest using either a simple majority vote or weighted average.

Random Forest Classifier has several advantages over other machine

learning models. It can handle both categorical and continuous features, can handle missing values, and is relatively insensitive to noisy data. It is also easy to interpret and can provide information about the importance of each feature.

Below are the python snippets of random forest models built with three different samples

RandomForest

1st sample

```
In [93]: from sklearn.ensemble import RandomForestClassifier
         rf = RandomForestClassifier(n_estimators=50,criterion='entropy',random_state = 10)
         rf.fit(x1,y1)
         y_pred1 = rf.predict(xtest)
         y_pred_train = rf.predict(x1)

In [94]: f1_score(ytest,y_pred1,average= 'weighted')
Out[94]: 0.7243831372448677
```

2nd sample

```
In [95]: from sklearn.ensemble import RandomForestClassifier
         rf = RandomForestClassifier(n_estimators=50,criterion='entropy',random_state = 10)
         rf.fit(x2,y2)
         y_pred2 = rf.predict(xtest)
         y_pred_train2 = rf.predict(x2)

In [96]: from sklearn.metrics import f1_score
         f1_score(ytest,y_pred2,average= 'weighted')
Out[96]: 0.72497273575893
```

3rd sample

```
In [97]: from sklearn.ensemble import RandomForestClassifier
         rf = RandomForestClassifier(n_estimators=50,criterion='entropy',random_state = 10)
         rf.fit(x3,y3)
         y_pred3 = rf.predict(xtest)
         y_pred_train3 = rf.predict(x3)

In [98]: from sklearn.metrics import f1_score
         f1_score(ytest,y_pred3,average= 'weighted')
Out[98]: 0.724163800879828
```

Interpretation of above models:

In multi-class classification problems, the classes are typically not equally represented in the dataset, meaning some classes may have significantly fewer samples than others. This class imbalance can have a significant impact on the performance metrics used to evaluate the model.

The weighted F1 score is a commonly used performance metric for multi-class classification problems that takes into account the class imbalance by weighting the F1 score of each class by its proportion in the dataset. This means

that the F1 score of each class is weighted by the number of samples belonging to that class, which gives more importance to the performance of the minority classes.

- F1 score of first sample - 0.724
- F1 score of second sample - 0.725
- F1 score of third sample - 0.724

The f1 score got slightly improved in the Random Forest.

Voting:

Voting

```
ypred_random = pd.concat([pd.Series(y_pred1),pd.Series(y_pred2),pd.Series(y_pred3)],axis=1)
```

```
ypred_random['final']=0
ypred_random.head(1)
```

	0	1	2	final
0	1	1	1	0

```
for i in range (ypred_random.shape[0]):
    if (ypred_random.iloc[i,0]==ypred_random.iloc[i,1]==ypred_random.iloc[i,2]):
        ypred_random.loc[i,'final'] = ypred_random.iloc[i,0]
    elif ((ypred_random.iloc[i,0]==ypred_random.iloc[i,1])|(ypred_random.iloc[i,2]==ypred_random.iloc[i,1])):
        ypred_random.loc[i,'final'] = ypred_random.iloc[i,1]
    elif ((ypred_random.iloc[i,0]==ypred_random.iloc[i,2])):
        ypred_random.loc[i,'final'] = ypred_random.iloc[i,0]
    else :
        ypred_random.loc[i,'final'] = ypred_random.iloc[i,2]
```

```
f1_score(ytest,ypred_random['final'],average='weighted')
```

```
0.7278597813750364
```

```
# Combine the 3 sets of predictions into a single 2D array
ypred = np.array([y_pred1, y_pred2, y_pred3])

# Get the majority vote for each sample
majority_vote = np.apply_along_axis(lambda x: np.bincount(x).argmax(), axis=0, arr=ypred)

print("Majority vote for the 3 samples:", majority_vote)
```

```
Majority vote for the 3 samples: [1 2 1 ... 4 1 3]
```

```
f1_score(ytest,majority_vote,average='weighted')
```

```
0.7277730382468421
```

The final ypred which we got by comparing the ypred's obtained from the three samples is then used to calculate the f1 score.

The final f1 score is slightly improved.

9.3 Gradient Boosting for all samples:

Gradient Boosting is a popular machine learning algorithm that builds an ensemble of decision trees iteratively to improve model performance. The algorithm is based on the idea of gradient descent, where a model is trained by minimizing the loss function (i.e., the difference between the predicted and actual values) in the direction of the negative gradient.

Gradient Boosting is a powerful algorithm that is particularly effective for complex datasets with a large number of features. Some popular implementations of Gradient Boosting include XGBoost, LightGBM, and CatBoost, which offer various optimizations and improvements to the basic algorithm.

One important thing to note is that Gradient Boosting is prone to overfitting, particularly if the model is too complex or the dataset is too small. Therefore, it is important to tune the hyperparameters carefully and use techniques like cross-validation and regularization to prevent overfitting.

Gradient Boosting ¶

1st sample

```
In [107]: gbc = GradientBoostingClassifier(n_estimators=50,random_state = 10)
          gbc.fit(x1,y1)
          y_pred1 = gbc.predict(xtest)
          y_pred_train = gbc.predict(x1)

In [108]: f1_score(ytest,y_pred1,average= 'weighted')
Out[108]: 0.7302053928979699
```

2nd sample

```
In [109]: gbc = GradientBoostingClassifier(n_estimators=50,random_state = 10)
          gbc.fit(x2,y2)
          y_pred2 = gbc.predict(xtest)
          y_pred_train = gbc.predict(x2)

In [110]: f1_score(ytest,y_pred2,average= 'weighted')
Out[110]: 0.7296223028712837
```

3rd sample

```
In [111]: gbc = GradientBoostingClassifier(n_estimators=50,random_state = 10)
          gbc.fit(x3,y3)
          y_pred3 = gbc.predict(xtest)
          y_pred_train = gbc.predict(x3)

In [112]: f1_score(ytest,y_pred3,average = 'weighted')
Out[112]: 0.7305046090299866
```

Interpretation of above models:

In multi-class classification problems, the classes are typically not equally represented in the dataset, meaning some classes may have significantly fewer samples than others. This class imbalance can have a significant impact on the performance metrics used to evaluate the model.

The weighted F1 score is a commonly used performance metric for multi-class classification problems that takes into account the class imbalance by weighting the F1 score of each class by its proportion in the dataset. This means that the F1 score of each class is weighted by the number of samples belonging to that class, which gives more importance to the performance of the minority classes.

- F1 score of first sample - 0.73
- F1 score of second sample - 0.729
- F1 score of third sample - 0.73

The f1 score got slightly improved by using the Gradient Boosting.

Voting:

A data frame ‘ypred_random’ has been created using the ypred’s obtained from the 3 models.

On comparing the three ypred’s, we got the final ypred by using the below algorithm :

- If all three model predictions are the same then set the final predicted label for that row to be the label predicted by the three models.
- Else, if two out of the three model predictions are the same then set the final predicted label for that row to be the label predicted by the two models.
- Else, if two models have the same prediction but not the third model then set the final predicted label

Voting

```
ypred_gradient = pd.concat([pd.Series(y_pred1),pd.Series(y_pred2),pd.Series(y_pred3)],axis=1)
```

```
ypred_gradient['final']=0
ypred_gradient.head(1)
```

	0	1	2	final
0	1	1	1	0

```
for i in range (ypred_gradient.shape[0]):
    if (ypred_gradient.iloc[i,0]==ypred_gradient.iloc[i,1]==ypred_gradient.iloc[i,2]):
        ypred_gradient.loc[i,'final'] = ypred_gradient.iloc[i,0]
    elif ((ypred_gradient.iloc[i,0]==ypred_gradient.iloc[i,1])|(ypred_gradient.iloc[i,2]==ypred_gradient.iloc[i,1])):
        ypred_gradient.loc[i,'final'] = ypred_gradient.iloc[i,1]
    elif ((ypred_gradient.iloc[i,0]==ypred_gradient.iloc[i,2])):
        ypred_gradient.loc[i,'final'] = ypred_gradient.iloc[i,0]
    else :
        ypred_gradient.loc[i,'final'] = ypred_gradient.iloc[i,2]
```

```
f1_score(ytest,ypred_gradient['final'],average='weighted')
```

```
0.730007389033497
```

The final ypred which we got by comparing the ypred's obtained from the three samples is then used to calculate the f1 score.

9.4 Stacking Classification:

Stacking classification is a type of ensemble learning method in machine learning where multiple models are combined to make a prediction by using a meta-model that learns to combine the outputs of the base models. It is a more sophisticated technique than simple voting classification, as it allows for more flexibility in the combination of the models and can result in higher accuracy.

The stacking classification method works as follows:

- The dataset is divided into a training set and a validation set.
- The base models are trained on the training set and used to make predictions on the validation set.
- The predictions from the base models are combined to create a new feature matrix, which is used to train the meta-model.
- The meta-model learns to combine the predictions from the base models by using the new feature matrix as input.
- The final prediction is made by passing the test data through the base models to obtain their predictions, which are then combined using the

meta-model.

Stacking classification can be used with any type of machine learning algorithm, including decision trees, logistic regression, support vector machines, and neural networks.

Here we are using Random Forest, Gradient Boost and XGBoost as the base model for stacking classifiers.

Stacking Classifier

In [142]:
```
from sklearn.ensemble import StackingClassifier,GradientBoostingClassifier
from xgboost import XGBClassifier
```

In [145]:
```
model1 = RandomForestClassifier(n_estimators=50,random_state=10)
model2 = GradientBoostingClassifier(n_estimators=50,random_state = 10)
model3 = XGBClassifier(n_estimators=50,random_state=10)

# Create an ensemble of models using a voting classifier
stacking_ensemble = StackingClassifier(estimators=[('rf', model1), ('gb', model2), ('xg', model3)],final_estimator=model2)
```

In [146]:
```
stacking_ensemble.fit(x1,y1)
```

Out[146]:
```
StackingClassifier(estimators=[('rf',
                                RandomForestClassifier(n_estimators=50,
                                                       random_state=10)),
                               ('gb',
                                GradientBoostingClassifier(n_estimators=50,
                                                           random_state=10)),
                               ('xg',
                                XGBClassifier(base_score=None, booster=None,
                                              callbacks=None,
                                              colsample_bylevel=None,
                                              colsample_bynode=None,
                                              colsample_bytree=None,
                                              early_stopping_rounds=None,
                                              enable_categorical=False,
                                              eval_m...
```

Stacking classifier model is created for three samples and the f1 score is calculated for each sample.

In [147]:
```
y_pred1 = stacking_ensemble.predict(xtest)
```

In [148]:
```
f1_score(ytest,y_pred1,average='weighted')
```

Out[148]:
```
0.7544838960855587
```

In [147]:
```
y_pred1 = stacking_ensemble.predict(xtest)
```

In [148]:
```
f1_score(ytest,y_pred1,average='weighted')
```

Out[148]:
```
0.7544838960855587
```

In [155]:
```
model1 = RandomForestClassifier(n_estimators=50,random_state=10)
model2 = GradientBoostingClassifier(n_estimators=50,random_state = 10)
model3 = XGBClassifier(n_estimators=50,random_state=10)

# Create an ensemble of models using a voting classifier
stacking_ensemble = StackingClassifier(estimators=[('rf', model1), ('gb', model2), ('xg', model3)],final_estimator=model2)


stacking_ensemble.fit(x2,y2)
y_pred2 = stacking_ensemble.predict(xtest)
```

In [156]:
```
f1_score(ytest,y_pred2,average='weighted')
```

Out[156]:
```
0.7514346240993183
```

```
In [ ]:
model1 = RandomForestClassifier(n_estimators=50,random_state=10)
model2 = GradientBoostingClassifier(n_estimators=50,random_state = 10)
model3 = XGBClassifier(n_estimators=50,random_state=10)

# Create an ensemble of models using a voting classifier
stacking_ensemble = StackingClassifier(estimators=[('rf', model1), ('gb', model2), ('xg', model3)],final_estimator=model2)


stacking_ensemble.fit(x3,y3)
y_pred3 = stacking_ensemble.predict(xtest)
```

```
In [148]:
f1_score(ytest,y_pred3,average='weighted')

Out[148]:
0.7544838960855587
```

Compared to the above models, the f1 score has been improved in Stacking classifier.

Voting:

Voting

```
In [ ]:
ypred_stacking = pd.concat([pd.Series(y_pred1),pd.Series(y_pred2),pd.Series(y_pred3)],axis=1)
```

```
In [ ]:
ypred_stacking['final']=0
ypred_stacking.head(1)
```

```
In [ ]:
for i in range (ypred_stacking.shape[0]):
    if (ypred_stacking.iloc[i,0]==ypred_stacking.iloc[i,1]==ypred_stacking.iloc[i,2]):
        ypred_stacking.loc[i,'final'] = ypred_stacking.iloc[i,0]
    elif ((ypred_stacking.iloc[i,0]==ypred_stacking.iloc[i,1])|(ypred_stacking.iloc[i,2]==ypred_stacking.iloc[i,1])):
        ypred_stacking.loc[i,'final'] = ypred_stacking.iloc[i,1]
    elif ((ypred_stacking.iloc[i,0]==ypred_stacking.iloc[i,2])):
        ypred_stacking.loc[i,'final'] = ypred_stacking.iloc[i,0]
    else :
        ypred_stacking.loc[i,'final'] = ypred_stacking.iloc[i,2]
```

```
In [ ]:
f1_score(ytest,ypred_stacking['final'],average='weighted')
```

The F1 score is about 0.753. Thus the F1 score of the Stacking classifier model is the best of all other models.

9.5 Improvements:

Grid search is a hyperparameter tuning technique used to find the optimal combination of hyperparameters for a given machine learning model. Random forest is a popular ensemble learning algorithm that is used for classification and regression tasks. In scikit-learn, we can perform grid search on random forests using the GridSearchCV function.

To improve the F1 score we can use Grid Search on Stacking Classifier. For the time being we are not using it.

9. Business Recommendations and insights:

Target Marketing:

Target marketing is a marketing strategy in which a company divides its overall market into smaller segments of consumers who have similar needs and characteristics, and then tailors its marketing efforts and messaging to reach and appeal to these specific groups.

The benefits of target marketing include:

- **Increased efficiency**: By focusing on a specific audience, target marketing allows companies to be more efficient with their marketing resources. Instead of spending money and effort trying to appeal to everyone, they can focus on the audience that is most likely to respond to their message.

- **Improved customer satisfaction**: Target marketing allows companies to

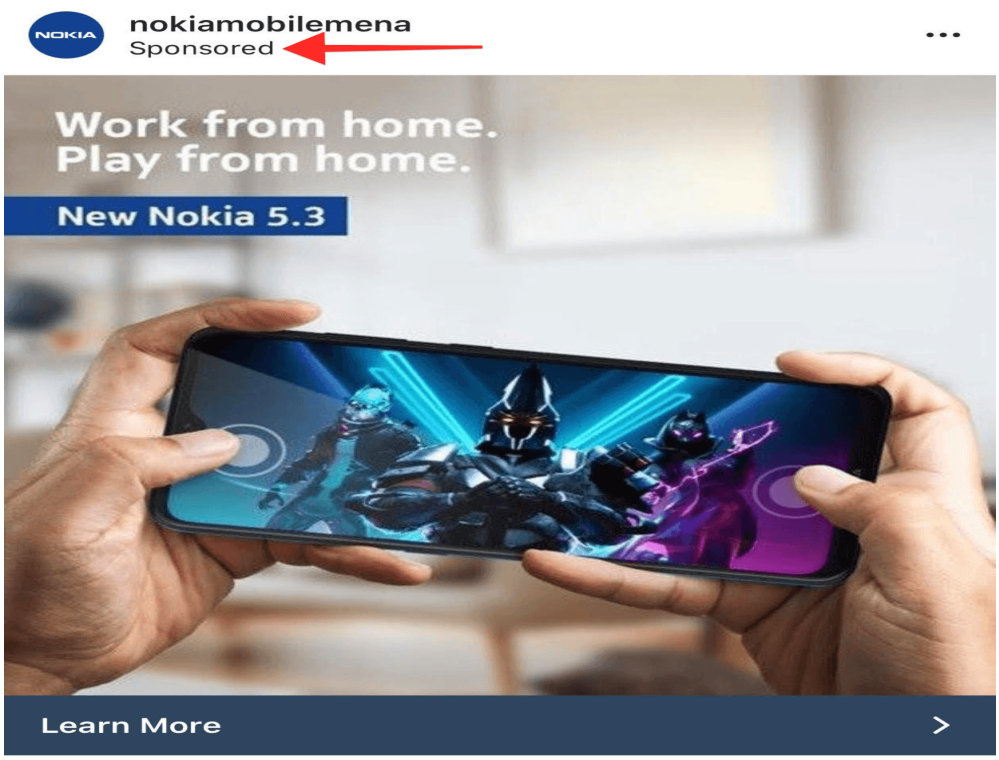
create products, services, and messaging that are specifically tailored to the needs and wants of their target audience, leading to higher levels of customer satisfaction.

- **Greater return on investment**: By reaching a specific audience with targeted messaging, companies can increase the likelihood that their marketing efforts will lead to conversions and sales, resulting in a greater return on investment.

- **Enhanced brand loyalty**: By consistently targeting and serving a specific audience, companies can build stronger relationships with their customers and enhance brand loyalty.

- **Competitive advantage**: Target marketing can help companies differentiate themselves from their competitors by offering products, services, and messaging that are tailored to the specific needs and preferences of their target audience, making it more difficult for competitors to replicate their approach.

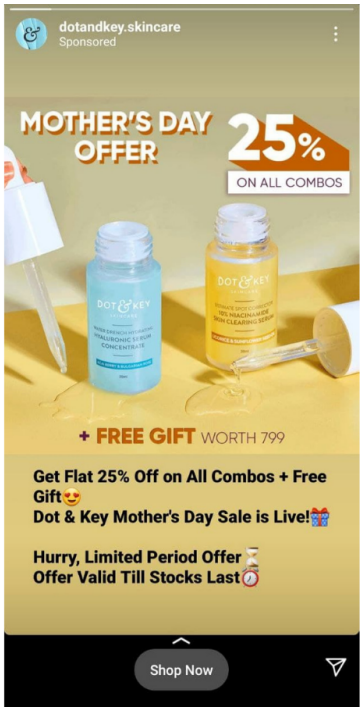
Social Media Analysis for target marketing:



4,242 likes

nokiamobilemena Enjoy your break from work and social media and actually disconnect from the world with Nokia 5.3, a true gaming experience. #Nokia5dot3

View all 86 comments



Social media analysis can be a powerful tool for target marketing. Here are some ways in which social media analysis can be used in target marketing:

- **Audience segmentation**: Social media analytics tools can help companies identify and segment their target audience based on factors such as demographics, interests, behaviors, and attitudes. By understanding the characteristics and preferences of their target audience, companies can create targeted messaging and content that is more likely to resonate with them.

- **Influencer identification**: Social media analytics tools can help companies identify influencers and thought leaders who have a strong following within their target audience. By partnering with these influencers, companies can amplify their message and reach a larger audience.

- **Campaign tracking**: Social media analytics tools can help companies track the performance of their marketing campaigns on social media. By monitoring metrics such as engagement, reach, and conversions, companies can evaluate the effectiveness of their campaigns and make adjustments to improve their targeting and messaging.

- **Content optimization**: Social media analytics tools can provide insights into the types of content that resonate most with a company's target audience. By analyzing metrics such as likes, comments, and shares, companies can optimize their content to better appeal to their target audience.

Why Is Social Media Marketing Important For Brands?

For business purposes, sites like Facebook, Twitter, Instagram and Trell in our case present an opportunity to engage with a massive audience.

Last year, there were more than 4.7 billion people worldwide using social media platforms, which means a whole lot of potential customers.

Social media allows you to tell your story and humanize your brand.

Without a large budget allocation, it lets you build an audience and stay top of mind with your targets.

You can connect and interact with customers, deal with feedback (both positive

and negative), and build authenticity just by being active on the right sites.

Here are some key stats about social media marketing:

- 55% of people learn about brands from social media.
- The average internet user spends 397 minutes per day online, with much of that on social media sites.
- 79.7% of people make purchases based on online or social media advertisements.

From paid display ads targeting a highly specific demographic to organic posts that go viral, social media presents an incredible opportunity to evangelize your brand, increase your visibility, and find new customers.

Case study:

Apple: The Shot on iPhone Challenge

When: 2015

Campaign Outline:

The world's most popular smartphone manufacturer, Apple, takes great pride in the quality of images that can be captured on its devices.

To highlight the great photos that it can take, it launched a competition that asked iPhone users to "capture the little things in a big way."

Photographers were then invited to share their images on Instagram and other social media sites using the hashtag #ShotOniPhone.

A panel of judges then selected 10 winners from tens of thousands of entries, which were then featured on Apple's website, the company's Instagram, and on 10,000+ billboards in 25 countries.

The Numbers:

The first round of the campaign had more than 6.5 billion impressions.
It was mentioned by 24,000 influencers, with a 95% positive comment rating.

Impact:

- User-generated content (UGC) is a low-investment way for companies to promote their brand on social media, but this isn't the reason for this campaign's success.

- Instead, Shot on iPhone encouraged people to discuss the campaign, which closely aligned with Apple's reputation for creativity, lifestyle, and innovation.

In our case, we perform audience segmentation based on age group. By doing this we can focus on performing e-commerce recommendations.

For example

- For people who are in the age group 1 (1 to 18) we can promote more toys, e-books, eatables like chocolates, ice-creams etc,..
- Similarly for people in the age group 2 (18 to 24) we can promote more sports accessories, electronics , clothes and other accessories.
- Next for group 3 (24 to 30) again we can focus on promoting clothes, accessories, movies, travel, beauty products etc,..
- Finally for group 4 (above 30) we can promote home appliances,pharmacy, Furniture, Groceries, Baby products, Professional things, Automotive, etc,..

We can also promote election campaigns using this classification of age group. Instead of promoting voting campaigns for all the users we can promote only for people who are of age group 2,3 and 4.

Next comes the important part of content optimization based on age group.

- For youngsters we can optimize our application to suggest more educational content which can be very fruitful.

11. Conclusion:

- From the above models we created, we found GradientBoostingClassifier and StackingClassifier with GradientBoostingClassifier as final estimator are reliable and consistent. So, we can use these models on a new set of data and be used for real time prediction. Now the question is which model should be used among these two.

- The Stacking Classifier is more reliable than the Gradient boosting Classifier because the weighted f1_score of test data is significantly high.

- In the future a dataset can be collected and this model can be used to predict the weighted f1_score.

12. Reference :

1. https://www.ibm.com/in-en/topics/social-media-analytics
2. https://www.researchgate.net/publication/259148570_The_Power_of_Social_Media_Analytics
3. https://www.targetinternet.com/resources/how-different-age-groups-are-using-social-media
4. https://www.simplilearn.com/intersection-of-data-science-and-social-media-article#:~:text=Social%20media%20analytics%20and%20bigcontent%20based%20on%20customer%20sentiment.
5. Data Science for Humanity
6. http://jenpan.com/jen_pan/50c.pdf