APPLE TWITTER SENTIMENT ANALYSIS

INTRODUCTION

In todays world,we are living in digital times.Social media platforms have been used as a source of public opinions and sentiments. Apple and Google companies monitor opinions and sentiments from customer and analyse the customer satisfactionso as to identify areas of improvement.These sentiments can offer good insights for business to improve their products and manage the brand image. This project aims to develop a Natural Language Processing(NLP) model to analyse Twitter sentiments about Apple and goodgle products.

Using dataset from CrowdFlower,i will train the model to predict the sentiment of any given tweet based on the content.This will help the Stakeholder to make data driven decisions.

BACKGROUND

Sentiment analysis helps in identifying and extracting subjective information from a text.Companies use Sentiment analysis to analyse customer feedback and make data driven decision. This project uses tweets about Apple and Google products rated by humans for sentiments to train and evaluate the NLP model.

DATA OVERVIEW

The dataset was sourced from CrowdFlower via data.world,the link for Kaggle is https://www.kaggle.com/datasets/slythe/apple-twitter-sentiment-crowdflower (https://www.kaggle.com/datasets/slythe/apple-twitter-sentiment-crowdflower)
There are 3886 Records and 12 Features.
The key Features are :
1. sentiment - sentiment labels
2. sentiment:confidence - confidence score for sentiment
3. date- when tweet was posted
4. query- search query used
5. text- actual tweet

The sentiments are already labelled and classified into classes which include:

1. '1' representing 'Negative'
2. '3' representing 'Neutral'
3. '5' representing 'Positive'
4. 'not_relevant' representing 'not relevant'

BUSINESS PROBLEM

It is crucial for companies to understand customer sentiments and enhance their satisfaction.Apple and Google company is interested in understanding the customer sentiments so as to improve their products and enhance customer satisfaction.By building an NLP model that accurately predicts sentiments of the tweets can help the company to make informed decision which can improve their performance hence contributing to business growth.

OBJECTIVE

Build a model that can rate sentiment of a tweet based on the content.

SUMMARY

1. Business and Data Understanding

The project aims in sentiment analysis.It analyses Apple and Google products and classify them into Positive(5),Negative(1),Neutral(3),Non relevant(non_relevant).The data consist of labeled tweets with sentiments making it well suited for classification task.Descriptive analysis was used such as distribution of sentiment classes.The data sourced from Twitter enables the company to gauge the customer satisfaction and get an insight of areas to improve to enhance the company performance.

2. Data Preparation

- The process included Data understanding,Data Cleaning and Text processing.
- Data understanding helped to know the size of dataset i was working on,the data structures i.e shape and identify the data types and whether its a categorical or a numerical feature.
- The data contained 3886 Records and 12 Features.The datatypes were bool(1), float64(2), int64(2), object(7).
- The dataset had 2 columns which had missing data and also the columns were irrelevant according to my objective so i dropped the columns plus other columns which were relevant and remained with two columns which were sentiment and text.
- Text preprocessing to remove URLs,Hashtags,mentions,special characters which helped to reduce noise and focus on the actual content of the tweet and converting text to lowercase to reduce vocabulary size and improve word frequency analysis.
- Stopwords were removed and Lemmatization together with stemming applied to reduce word to their base form.
- The library used for prepare the data were NLTK for Tokenization,stopwords removal,Lemmatization,Stemming amd TfidVectorizer from Scikit learn to convert text to Numeric.

3. Modeling

I used two modeling packages which included Scikit-learn and Natural Language Toolkit(NLTK). Sklearn models included:

- Logistic Regression which is simple and interpretable and effective for text classification task,I usedIT in Binary classification which then expanded to multiclass classification.
- Decision tree and RandomForest was performed and ensemble methods like Bagging to improve the prediction accuracy though it did not help improve the accuracy.it helped reduce overfitting.
- XGBoost was used to capture complex patterns but did not improve the model over Logistic Regression.
- NLTK was used for tokenization,Lemmatization,stemming and Stopwords removal.
- Hyperparameter Tuning was performed using GridsearchCV to find optimal parameters.Performed cross validation with 5 folds to ensure model generelized well on unseen data.
- Feature engineering with Tfidvectorizer with different n_gram range.
- Removed stop words to reduce noise and improve performance.
- Logistic Regression with class_weight ='balanced' was used to handle the class imbalance and improve performance.

4. Evaluation

- The models were evaluated with Accuracy,Precision,Recall and F1 score.
- The best model was Logistic Regression with Gridsearch,it had Accuracy of 72% ,precision of 68% and Bagged RandomForestClassifier with Accuracy 71% and precision 70%.
- In the Validation approach,I used a Train_test split to validate the model performance,ensuring the results are well generelised with new data.The dataset was split into 70% training data and 30% test data.

Data Understanding

Data Preparation

This process involve preparing the data which include Loading the dataset,checking the data types,checking the shape,calculating the summary statistics.

In [1]:

```
# Install XGboost
from xgboost import XGBClassifier
```

Data Description

In [2]:

```
# Import relevant columns
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder,OneHotEncoder,StandardScaler
from sklearn.metrics import accuracy_score,recall_score,precision_score,f1_
from sklearn.model_selection import train_test_split,GridSearchCV
from sklearn.ensemble import RandomForestClassifier,BaggingClassifier
from sklearn.pipeline import Pipeline
import re
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import RegexpTokenizer
from nltk.stem import WordNetLemmatizer,PorterStemmer
from sklearn.feature_extraction.text import TfidfVectorizer,CountVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import MultinomialNB
nltk.download('stopwords')
nltk.download('wordnet')
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\user\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\user\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

Out[2]: True

In [3]:
```python
# Loading the dataset
apple_df = pd.read_csv("Apple-Twitter-Sentiment-DFE.csv",encoding='latin1')
apple_df.head() # checking the first 5 rows
```

Out[3]:

	_unit_id	_golden	_unit_state	_trusted_judgments	_last_judgment_at	sentiment	senti
0	623495513	True	golden	10	NaN	3	
1	623495514	True	golden	12	NaN	3	
2	623495515	True	golden	10	NaN	3	
3	623495516	True	golden	17	NaN	3	
4	623495517	False	finalized	3	12/12/14 12:14	3	

In [4]:
```python
# Checking the columns
apple_df.columns
```

Out[4]:
```
Index(['_unit_id', '_golden', '_unit_state', '_trusted_judgments',
       '_last_judgment_at', 'sentiment', 'sentiment:confidence', 'date',
'id',
       'query', 'sentiment_gold', 'text'],
      dtype='object')
```

In [5]:
```
# Checking information
apple_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3886 entries, 0 to 3885
Data columns (total 12 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   _unit_id             3886 non-null   int64  
 1   _golden              3886 non-null   bool   
 2   _unit_state          3886 non-null   object 
 3   _trusted_judgments   3886 non-null   int64  
 4   _last_judgment_at    3783 non-null   object 
 5   sentiment            3886 non-null   object 
 6   sentiment:confidence 3886 non-null   float64
 7   date                 3886 non-null   object 
 8   id                   3886 non-null   float64
 9   query                3886 non-null   object 
 10  sentiment_gold       103 non-null    object 
 11  text                 3886 non-null   object 
dtypes: bool(1), float64(2), int64(2), object(7)
memory usage: 337.9+ KB
```

In [6]:
```
# Checking the shape
apple_df.shape
```

Out[6]: (3886, 12)

- The dataset has 3886 rows and 12 columns.
- The data types are 1 boolen,2 float64,2 int64 and 7 onject.
- The dataset has columns that has missing values like _last_judgement_at,sentiment_gold
- There is also unecessary columns for our use case.

In [7]:
```
# Calculating the summary statistics
apple_df.describe()
```

Out[7]:

	_unit_id	_trusted_judgments	sentiment:confidence	id
count	3.886000e+03	3886.000000	3886.000000	3.886000e+03
mean	6.234975e+08	3.687082	0.829526	5.410039e+17
std	1.171906e+03	2.004595	0.175864	7.942752e+14
min	6.234955e+08	3.000000	0.332700	5.400000e+17
25%	6.234965e+08	3.000000	0.674475	5.400000e+17
50%	6.234975e+08	3.000000	0.811250	5.410000e+17
75%	6.234984e+08	3.000000	1.000000	5.420000e+17
max	6.235173e+08	27.000000	1.000000	5.420000e+17

In [8]:
```python
# Checking the sentiment classes
print(apple_df['sentiment'].unique())
```

```
['3' '5' '1' 'not_relevant']
```

The sentiment classes include:

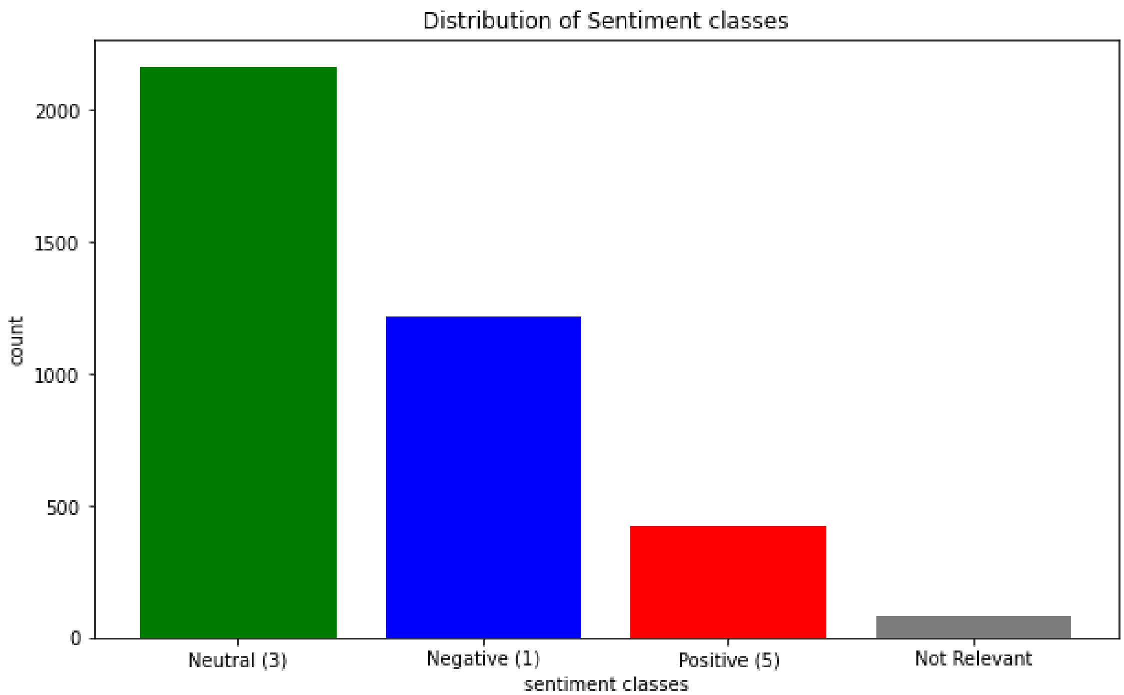
- '1' representing 'Negative'
- '3' representing 'Neutral'
- '5' representing 'Positive'
- 'not_relevant' representing 'not relevant'

In [9]:
```python
#Checking the distribution of the classes
counts = apple_df['sentiment'].value_counts()
counts
```

Out[9]:
```
sentiment
3               2162
1               1219
5                423
not_relevant      82
Name: count, dtype: int64
```

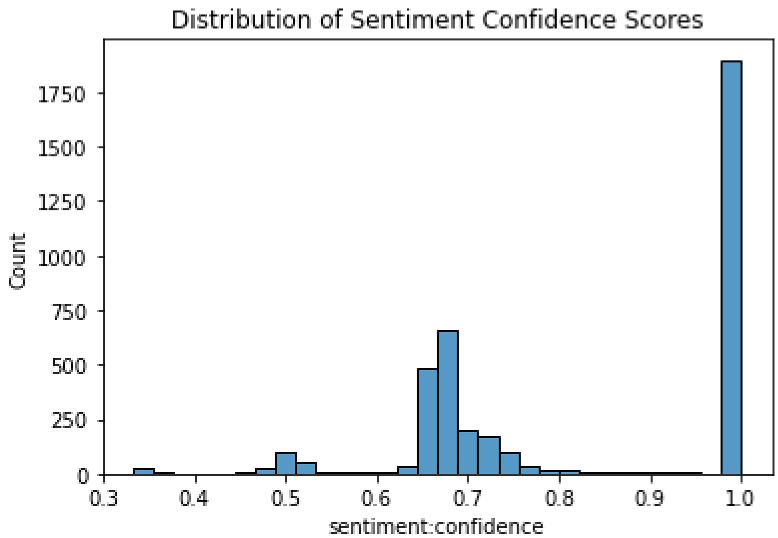
In [10]:

```
# Visualise the sentiment classes
labels = counts.index
values = counts.values

labels = ['Neutral (3)','Negative (1)','Positive (5)','Not Relevant']

plt.figure(figsize=(10,6))
plt.bar(labels,counts.values,color = ['g','b','r','gray'])
plt.title('Distribution of Sentiment classes')
plt.xlabel('sentiment classes')
plt.ylabel('count')
plt.show()
```



- Neutral(3) had the highest sentiment,followed by Negative sentiment(1),then Positive(5) and the non_relevant which had the smallest.

In [11]:
```
# View the distribution of sentiment confidence
sns.histplot(apple_df['sentiment:confidence'], bins=30)
plt.title('Distribution of Sentiment Confidence Scores')
plt.show()
```



- The sentiment confidence of 1 has the highest frequency scores indicating there is many instances where the model is highly confident about the sentiment prediction.
- There is also an indication of moderate level of confidence between 0.5 to 0.8 and also smaller peaks around 0.3 to 0.4 indicating low to moderate confidence.

Data Cleaning

Missing values

In [12]:
```
# Checking the missing values
apple_df.isna().sum()
```

Out[12]:
```
_unit_id                0
_golden                 0
_unit_state             0
_trusted_judgments      0
_last_judgment_at     103
sentiment               0
sentiment:confidence    0
date                    0
id                      0
query                   0
sentiment_gold       3783
text                    0
dtype: int64
```

In [13]:
```
# Drop the missing values and irrelevant columns
apple_df= apple_df.drop(columns=['sentiment_gold','_last_judgment_at'],axis
```

In [14]:
```
# Drop irrelevant columns
apple_cleaned_df = apple_df.drop(columns=['_unit_id', '_golden', '_unit_sta
apple_cleaned_df.dropna().head() # Drop missing rows and display the first
```

Out[14]:

	sentiment	text
0	3	#AAPL:The 10 best Steve Jobs emails ever...htt...
1	3	RT @JPDesloges: Why AAPL Stock Had a Mini-Flas...
2	3	My cat only chews @apple cords. Such an #Apple...
3	3	I agree with @jimcramer that the #IndividualIn...
4	3	Nobody expects the Spanish Inquisition #AAPL

Duplicated columns

In [15]:
```
# Drop the duplicates
apple_cleaned_df= apple_cleaned_df.drop_duplicates()
```

Text Processing

In [16]:
```
# Feature Engineering
# Tokenize
tokenizer = RegexpTokenizer(r'\w+')
# Create a list of stopwords in English
stopwords_list = stopwords.words('English')
# Initialize Lemmatizer
lemmatizer = WordNetLemmatizer()
```

In [17]:

```
def preprocess_text(text, tokenizer, stopwords_list, lemmatizer):
    # Remove URLs
    text = re.sub(r"http\S+|www\S+", "", text)
    # Remove hashtags
    text = re.sub(r"#\S+", " ", text)
    # Remove mentions
    text = re.sub(r"@\S+", " ", text)
    # Remove special characters and numbers
    text = re.sub(r"[^a-zA-Z\s]", " ", text)
    # Convert to lowercase
    text = text.lower()
    # Tokenize
    tokens = tokenizer.tokenize(text)
    # Remove stop words
    tokens = [word for word in tokens if word not in stopwords_list]
    # Lemmatize
    tokens = [lemmatizer.lemmatize(word) for word in tokens]
    # Join tokens back to string
    text = " ".join(tokens)
    return text
```

In [18]:

```
# Apply the preprocessing function to the DataFrame
apple_cleaned_df['prepro_text'] = apple_cleaned_df['text'].apply(lambda x:
apple_cleaned_df.head()
```

Out[18]:

	sentiment	text	prepro_text
0	3	#AAPL:The 10 best Steve Jobs emails ever...htt...	best steve job email ever
1	3	RT @JPDesloges: Why AAPL Stock Had a Mini-Flas...	rt aapl stock mini flash crash today aapl
2	3	My cat only chews @apple cords. Such an #Apple...	cat chew cord
3	3	I agree with @jimcramer that the #IndividualIn...	agree trade extended today pullback good see
4	3	Nobody expects the Spanish Inquisition #AAPL	nobody expects spanish inquisition

Visualize a sample text

In [19]:
```
for i in range (10):
    print(f"original text: {apple_cleaned_df['text'].iloc[i]}")
    print(f"cleaned text: {apple_cleaned_df['prepro_text'].iloc[i]}")
    print(" ")
```

```
original text: #AAPL:The 10 best Steve Jobs emails ever...http://t.co/82G
1kL94tx
cleaned text: best steve job email ever

original text: RT @JPDesloges: Why AAPL Stock Had a Mini-Flash Crash Toda
y $AAPL #aapl
http://t.co/hGFcjYa0E9 (http://t.co/hGFcjYa0E9)
cleaned text: rt aapl stock mini flash crash today aapl

original text: My cat only chews @apple cords. Such an #AppleSnob.
cleaned text: cat chew cord

original text: I agree with @jimcramer that the #IndividualInvestor shoul
d own not trade #Apple #AAPL, it's extended so today's pullback is good t
o see
cleaned text: agree trade extended today pullback good see

original text: Nobody expects the Spanish Inquisition #AAPL
cleaned text: nobody expects spanish inquisition

original text: #AAPL:5 Rocket Stocks to Buy for December Gains: Apple and
More...http://t.co/eG5XhXdLLS
cleaned text: rocket stock buy december gain apple

original text: Top 3 all @Apple #tablets. Damn right! http://t.co/RJiGn2J
UuB (http://t.co/RJiGn2JUuB)
cleaned text: top damn right

original text: CNBCTV: #Apple's margins better than expected? #aapl htt
p://t.co/7geVrtOGLK (http://t.co/7geVrtOGLK)
cleaned text: cnbctv margin better expected

original text: Apple Inc. Flash Crash: What You Need to Know http://t.co/
YJIgtifdAj (http://t.co/YJIgtifdAj) #AAPL
cleaned text: apple inc flash crash need know

original text: #AAPL:This Presentation Shows What Makes The World's Bigge
st Tech Companies ...http://t.co/qlH9PqSoSd
cleaned text: presentation show make world biggest tech company
```

Building a model

I will first limit my analysis to Postive and Negative tweets only building a binary classifier and then add the Neutral tweets to build a Multiclass classifier.

In [20]:
```
print(apple_cleaned_df['sentiment'].unique())
```

```
['3' '5' '1' 'not_relevant']
```

Binary class classifier

In [21]:
```
# Filter to include positive(5) and Negative(1) sentiments only
binary_df = apple_cleaned_df[apple_cleaned_df['sentiment'].isin(['5','1'])]
binary_df.head()
```

Out[21]:

	sentiment	text	prepro_text
6	5	Top 3 all @Apple #tablets. Damn right! http://...	top damn right
7	5	CNBCTV: #Apple's margins better than expected?...	cnbctv margin better expected
10	1	WTF MY BATTERY WAS 31% ONE SECOND AGO AND NOW ...	wtf battery one second ago wtf
13	5	RT @peterpham: Bought my @AugustSmartLock at t...	rt bought store pretty good logo match wait in...
14	1	@apple Contact sync between Yosemite and iOS8 ...	contact sync yosemite io seriously screwed use...

Define X and y

In [22]:
```
X_bin =binary_df['prepro_text']
X_bin.head()
```

Out[22]:
```
6                                      top damn right
7                       cnbctv margin better expected
10                     wtf battery one second ago wtf
13    rt bought store pretty good logo match wait in...
14    contact sync yosemite io seriously screwed use...
Name: prepro_text, dtype: object
```

In [23]:
```
y_bin = binary_df['sentiment']
```

Pipeline

- Create a pipeline for vectorizing and also Hyper tuning
- used class_weight='balanced' to balance the classes

In [24]:
```
# Define pipeline
pipe = Pipeline([
    ('vectorizer',TfidfVectorizer(ngram_range=(1,2))),
    ('model', LogisticRegression(random_state=42,class_weight='balanced')),
])
# Define parameter grid
param_grid = [
    {
        'model':[LogisticRegression(max_iter=200,solver='liblinear',class_w
        'model__C': [0.001, 0.01, 0.1, 1, 10, 100, 1000]
    },

    {
        'model' : [RandomForestClassifier(class_weight='balanced')],
        'model__n_estimators': [50,100],
        'model__criterion':['gini','entropy'],
        'model__max_depth': [None,10,20]
     },
     {
         'model': [DecisionTreeClassifier(random_state=42)],
         'model__criterion':['gini','entropy'],
         'model__max_depth': [None,10,20]
      }

]
```

Train-Test split

In [25]:
```
# Split the data into training and test data
X_train_bin,X_test_bin,y_train_bin,y_test_bin =train_test_split(X_bin,y_bir
```

Model Training and fitting(binary)

In [26]:
```

pipe.fit(X_train_bin,y_train_bin)

```

Out[26]:
```
Pipeline(steps=[('vectorizer', TfidfVectorizer(ngram_range=(1, 2))),
                ('model',
                 LogisticRegression(class_weight='balanced', random_state
=42))])
```

In [27]:
```
print(X_train_bin)
```

```
1131      hour hold customer service trying figure messup
1355    lol fuck going pay left arm amp leg product ma...
2657    ok look yall still reversed back io idk hold i...
2073    leave mac plugged leave house need check power...
1122                               phone die plugged wtf
                              ...
2859    freaking computer went psycho ruined life than...
3241    day without access passwd apple working new ap...
2343                                                duck
3816    rt phone went please tell mathematically possible
2847                       company official acct company
Name: prepro_text, Length: 1036, dtype: object
```

In [28]:
```
# Predict
y_pred_binary = pipe.predict(X_test_bin)
```

Model Evaluation

In [29]:
```
# Convert target labels to integers
y_test_bin = y_test_bin.astype(int)
y_pred_binary = y_pred_binary.astype(int)
```

In [30]:
```
accuracy_bin = accuracy_score(y_test_bin,y_pred_binary)
precisin_bin = precision_score(y_test_bin,y_pred_binary)
recall_bin = recall_score(y_test_bin,y_pred_binary)
f1_bin = f1_score(y_test_bin,y_pred_binary)
roc_auc_bin = roc_auc_score(y_test_bin,y_pred_binary)

print(f"Accuracy : {accuracy_bin}")
print(f"Precision : {precisin_bin}")
print(f"Recall : {recall_bin}")
print(f"F1 : {f1_bin}")
print(f"ROC_AUC : {roc_auc_bin}")
```

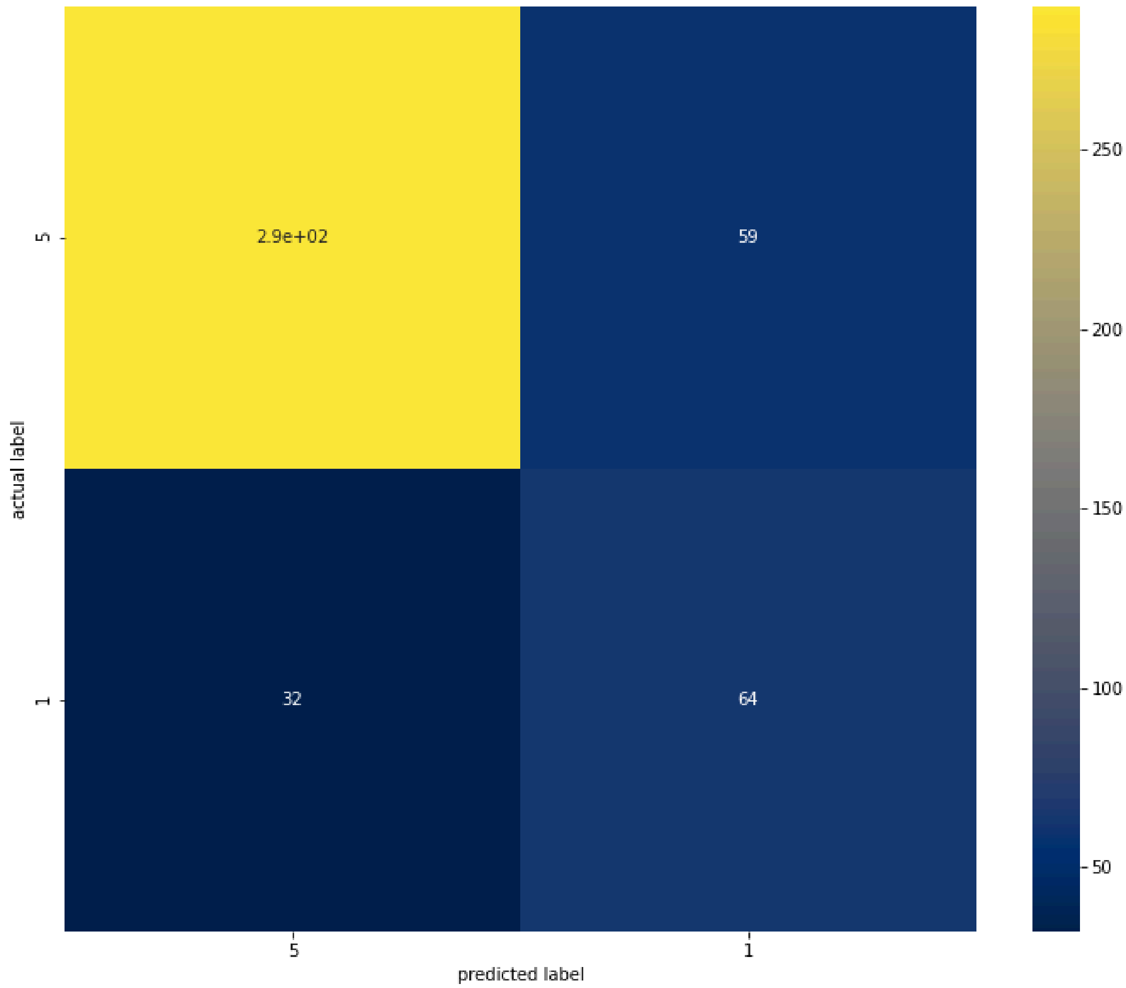
```
Accuracy : 0.7955056179775281
Precision : 0.9006211180124224
Recall : 0.830945558739255
F1 : 0.8643815201192251
ROC_AUC : 0.7488061127029607
```

In [31]:

```
classif_report = classification_report(y_test_bin,y_pred_binary)
conf_matrix = confusion_matrix(y_test_bin,y_pred_binary)

print(f"classification_Report:,{classif_report}")
print(f"confusion_matrix:",conf_matrix)
```

```
classification_Report:,              precision    recall  f1-score   supp
ort

           1       0.90      0.83      0.86       349
           5       0.52      0.67      0.58        96

    accuracy                           0.80       445
   macro avg       0.71      0.75      0.72       445
weighted avg       0.82      0.80      0.80       445

confusion_matrix: [[290  59]
 [ 32  64]]
```

In [32]:
```
# Plot confusion matrix
plt.figure(figsize=(12,10))
sns.heatmap(conf_matrix,annot=True,cmap="cividis",xticklabels=['5', '1'],yt
plt.xlabel('predicted label')
plt.ylabel('actual label')
plt.show()
```



Interpretation:

- The score had accuracy of 80%
- The model performs well for the negative class (1) with high precision, recall,accuracy, and F1 score.
- Performance for the positive class (5) is lower, indicating some challenges in correctly identifying positive tweets.

Join to form a Multi class classifier

In [33]:
```
# Create a dataframe containing the four classes of sentiment
multi_df = apple_cleaned_df[apple_cleaned_df['sentiment'].isin(['3' ,'5',
multi_df.head()
```

Out[33]:

	sentiment	text	prepro_text
0	3	#AAPL:The 10 best Steve Jobs emails ever...htt...	best steve job email ever
1	3	RT @JPDesloges: Why AAPL Stock Had a Mini-Flas...	rt aapl stock mini flash crash today aapl
2	3	My cat only chews @apple cords. Such an #Apple...	cat chew cord
3	3	I agree with @jimcramer that the #IndividualIn...	agree trade extended today pullback good see
4	3	Nobody expects the Spanish Inquisition #AAPL	nobody expects spanish inquisition

In [34]:
```
# Filter out the non relevant class
multi_df = multi_df[multi_df['sentiment'] != 'not_relevant']
```

- Filtered out the non_relevant class as it was not relevant to the Apple and Google products so it was bringing noise to the data hence reducing the modeling performance

In [35]:
```
print(multi_df['sentiment'].unique())
```

```
['3' '5' '1']
```

In [36]:
```
X_multi = multi_df['prepro_text']
y_multi = multi_df['sentiment']
```

Train_Test Split

In [37]:
```
X_train_mult,X_test_mult,y_train_mult,y_test_mult = train_test_split(X_mult
X_train_mult
```

Out[37]:
```
380      ikr u dont need bc hopefully buy one christmas...
1288     star analyst alex gauna jmp security rated aap...
2990     take convert shopper buyer amp create total se...
815                               new patent broken screen
1303                                            domt think
                               ...
3807                              pick number one app year
1190                                       happy thnx fag
1225      apple poised record breaking holiday season aapl
1400     io fucked battery life iphone really starting ...
941                                                 really
Name: prepro_text, Length: 2213, dtype: object
```

Model training and fitting(multi) With Gridsearch

In [38]:

```
Pipe2 = Pipeline([
     ('vectorizer',TfidfVectorizer(ngram_range=(1,2))),
    ('model', LogisticRegression(random_state=42,class_weight='balanced'))

])
# Define parameter grid
param_grid = [
    {
        'model':[LogisticRegression(max_iter=200,solver='liblinear',multi_c
        'model__C': [1,10,100]
    },

    {
        'model' : [RandomForestClassifier(class_weight='balanced')],
        'model__n_estimators': [50,100],
        'model__criterion':['gini','entropy'],
        'model__max_depth': [None,10,20]
     },

     {
         'model': [MultinomialNB()]

      }

]
# Perform Gridsearch
grid_search = GridSearchCV(Pipe2,param_grid,cv=5,verbose=1,n_jobs=-1,scorir
grid_search.fit(X_train_mult,y_train_mult)
```

```
Fitting 5 folds for each of 16 candidates, totalling 80 fits
```

Out[38]:
```
GridSearchCV(cv=5,
             estimator=Pipeline(steps=[('vectorizer',
                                        TfidfVectorizer(ngram_range=(1,
2))),
                                       ('model',
                                        LogisticRegression(class_weight
='balanced',
                                                           random_state=4
2))]),
             n_jobs=-1,
             param_grid=[{'model': [LogisticRegression(C=1,
                                                       class_weight='bala
nced',
                                                       max_iter=200,
                                                       multi_class='ovr',
                                                       solver='liblinea
r')],
                          'model__C': [1, 10, 100]},
                         {'model': [RandomForestClassifier(class_weight
='balanced')],
                          'model__criterion': ['gini', 'entropy'],
                          'model__max_depth': [None, 10, 20],
                          'model__n_estimators': [50, 100]},
                         {'model': [MultinomialNB()]}],
             scoring='accuracy', verbose=1)
```

Best Model Evaluation

In [39]:
```
# Get the best model
best_model = grid_search.best_estimator_
best_model
```

Out[39]:
```
Pipeline(steps=[('vectorizer', TfidfVectorizer(ngram_range=(1, 2))),
                ('model',
                 LogisticRegression(C=1, class_weight='balanced', max_ite
r=200,
                                    multi_class='ovr', solver='liblinea
r'))])
```

In [40]:
```
# Refit the best pipe to train
best_model.fit(X_train_mult,y_train_mult)
# predict the model
y_pred_multi= best_model.predict(X_test_mult)
accuracy_multi = accuracy_score(y_test_mult,y_pred_multi)
```

In [41]:

```
accuracy_multi = accuracy_score(y_test_mult,y_pred_multi)
precision_multi = precision_score(y_test_mult,y_pred_multi,average='macro')
recall_multi = recall_score(y_test_mult,y_pred_multi,average='macro')
f1_multi = f1_score(y_test_mult,y_pred_multi,average='macro')

print(f"Accuracy : {accuracy_multi}")
print(f"Precision : {precision_multi}")
print(f"Recall : {recall_multi}")
print(f"F1 : {f1_multi}")
```

```
Accuracy : 0.7133825079030558
Precision : 0.6772349574362998
Recall : 0.6052073306600754
F1 : 0.6252947061062623
```

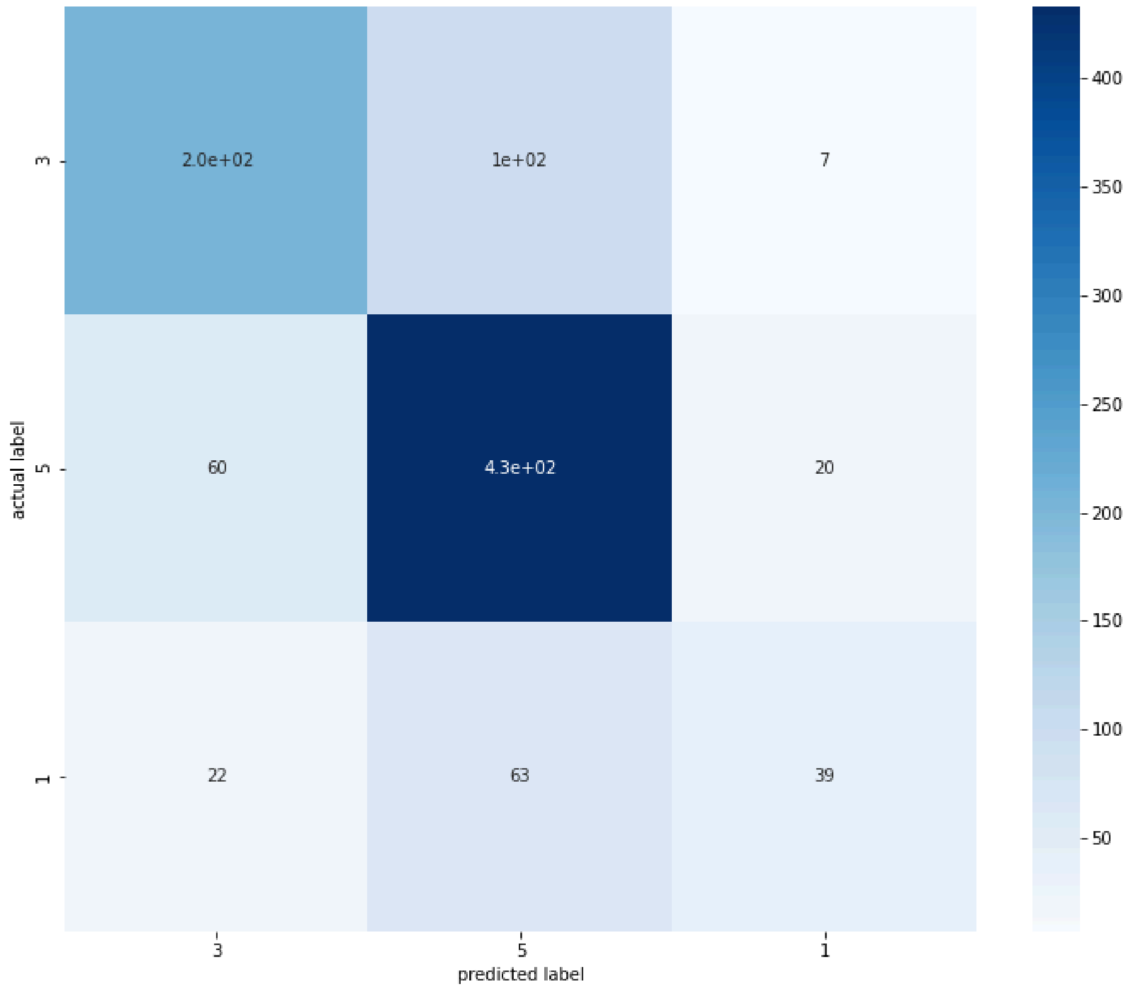
In [42]:

```

classification_report_multi = classification_report(y_test_mult,y_pred_mult
confusion_matrix_multi = confusion_matrix(y_test_mult,y_pred_multi)

print(f"classification Report:,{classification_report_multi}")
print(f"confusion matrix:",confusion_matrix_multi)
```

```
classification Report:,              precision    recall  f1-score   supp
ort

           1       0.71      0.66      0.68       312
           3       0.73      0.84      0.78       513
           5       0.59      0.31      0.41       124

    accuracy                           0.71       949
   macro avg       0.68      0.61      0.63       949
weighted avg       0.70      0.71      0.70       949

confusion matrix: [[205 100   7]
 [ 60 433  20]
 [ 22  63  39]]
```

In [43]:

```
# Plot confusion matrix
plt.figure(figsize=(12,10))
sns.heatmap(confusion_matrix_multi,annot=True,cmap="Blues",xticklabels=['3'
plt.xlabel('predicted label')
plt.ylabel('actual label')
plt.show()
```



Interpretation

- The accuracy of 71% indicates 71% of the predictions made by the model are correct.
- The precision of 68% indicates that, on average, 68% of the predicted positive instances are correct.
- The recall of 61% indicates on average, 63% of the actual positive instances are correctly identified by the model.
- The F1 score of 63% is a balance between precision and recall, providing a single metric that accounts for both false positives and false negatives.

Bagging(Bootstrap aggregation) with Decision Tree Classifier

In [44]:
```
#convert text data(X_train_mult) to TF-IDF features
vectorizer =TfidfVectorizer()
X_train_tdif =vectorizer.fit_transform(X_train_mult)
X_test_tdif =vectorizer.transform(X_test_mult)
```

In [45]:
```
# Define the base model
base_model = DecisionTreeClassifier()

# Define the bagging ensemble model
bagging_model = BaggingClassifier(base_model, n_estimators=50, random_state

# Train the model
bagging_model.fit(X_train_tdif, y_train_mult)
```

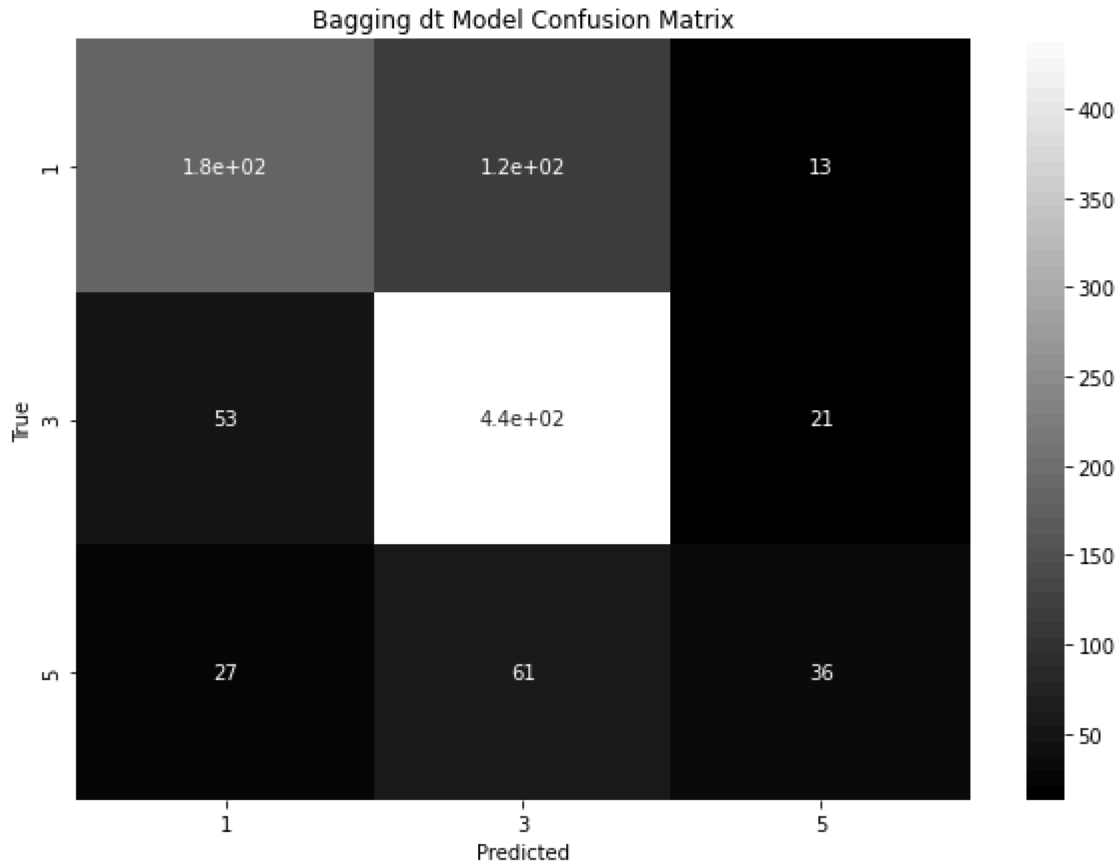
Out[45]:
```
BaggingClassifier(base_estimator=DecisionTreeClassifier(), n_estimators=5
0,
                  random_state=42)
```

In [46]:

```
# Make predictions on the test set
y_pred_dt = bagging_model.predict(X_test_tdif)

# Calculate performance metrics
accuracy_dt = accuracy_score(y_test_mult, y_pred_dt)
precision_dt = precision_score(y_test_mult, y_pred_dt, average='macro')
recall_dt = recall_score(y_test_mult, y_pred_dt, average='macro')
f1_dt = f1_score(y_test_mult, y_pred_dt, average='macro')

print(f"Bagging Model_dt Accuracy: {accuracy_dt}")
print(f"Bagging Model_dt Precision: {precision_dt}")
print(f"Bagging Model_dt Recall: {recall_dt}")
print(f"Bagging Model_dt F1: {f1_dt}")

# Classification report and confusion matrix
classif_report_bagging_dt = classification_report(y_test_mult, y_pred_dt)
conf_matrix_bagging_dt = confusion_matrix(y_test_mult, y_pred_dt)

print(f"Bagging Model Classification Report_dt:{classif_report_bagging_dt}'
print(f"Bagging Model Confusion Matrix_dt:{conf_matrix_bagging_dt}")
```

```
Bagging Model_dt Accuracy: 0.6923076923076923
Bagging Model_dt Precision: 0.6401498320634252
Bagging Model_dt Recall: 0.5764688004359765
Bagging Model_dt F1: 0.5940905041748165
Bagging Model Classification Report_dt:              precision    recall
f1-score   support

           1       0.69      0.58      0.63       312
           3       0.71      0.86      0.78       513
           5       0.51      0.29      0.37       124

    accuracy                           0.69       949
   macro avg       0.64      0.58      0.59       949
weighted avg       0.68      0.69      0.68       949

Bagging Model Confusion Matrix_dt:[[182 117  13]
 [ 53 439  21]
 [ 27  61  36]]
```

In [47]:
```
plt.figure(figsize=(10, 7))
sns.heatmap(conf_matrix_bagging_dt, annot=True, cmap='gray', xticklabels=[
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Bagging dt Model Confusion Matrix')
plt.show()
```



Bagging with Random Forest Classifier

In [48]:
```
rf = RandomForestClassifier(random_state=42)
```

In [49]:
```
# Instantiate the RandomForestClassifier
RandomForestClassifier(n_estimators=200,bootstrap=True, min_samples_leaf=1,
                       max_depth=20, max_features='auto', random_state=42)
```

Out[49]:
```
RandomForestClassifier(class_weight='balanced', max_depth=20, n_estimator
s=200,
                       random_state=42)
```

In [50]:
```
# Fit the classifier
rf.fit(X_train_tdif,y_train_mult)
```

Out[50]:
```
RandomForestClassifier(random_state=42)
```

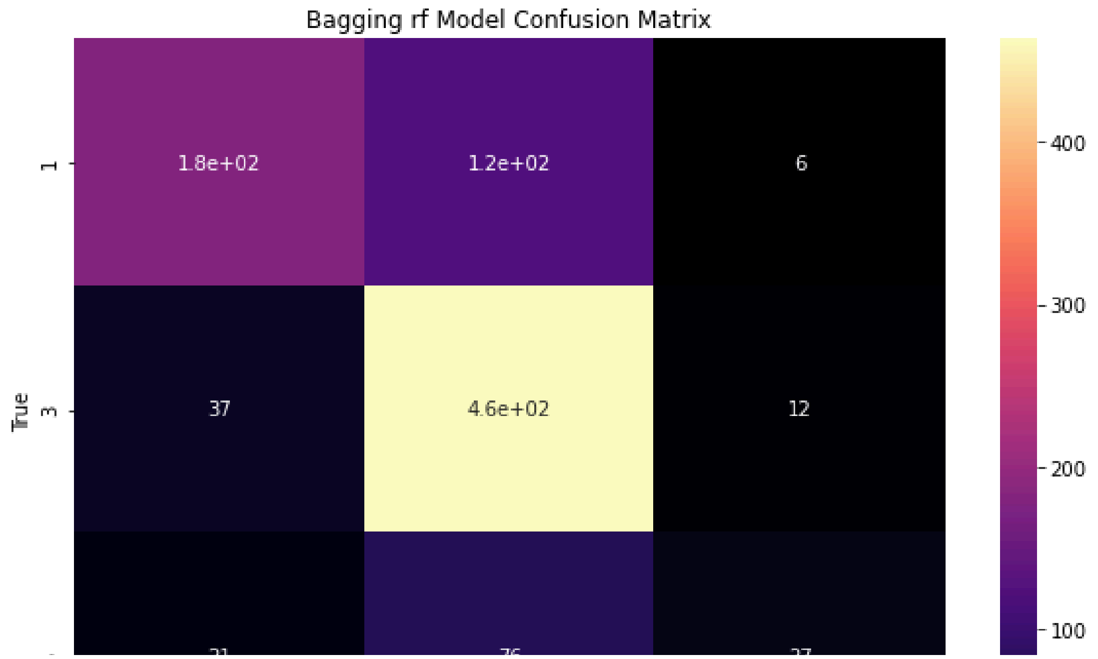
In [51]:
```
y_pred_rf = rf.predict(X_test_tdif)
```

In [52]:
```
accuracy_rf = accuracy_score(y_test_mult, y_pred_rf)
precision_rf = precision_score(y_test_mult, y_pred_rf, average='macro')
recall_rf = recall_score(y_test_mult, y_pred_rf, average='macro')
f1_rf = f1_score(y_test_mult, y_pred_rf, average='macro')

print(f"Bagging Model_rf Accuracy: {accuracy_rf}")
print(f"Bagging Model_rf Precision: {precision_rf}")
print(f"Bagging Model_rf Recall: {recall_rf}")
print(f"Bagging Model_rf F1: {f1_rf}")

# Classification report and confusion matrix
classif_report_rf_bagging = classification_report(y_test_mult, y_pred_rf)
conf_matrix_rf_bagging = confusion_matrix(y_test_mult, y_pred_rf)

print(f"Bagging Model Classification Report_rf:{classif_report_rf_bagging}'
print(f"Bagging Model Confusion Matrix_rf:{conf_matrix_rf_bagging}")
```

```
Bagging Model_rf Accuracy: 0.708113804004215
Bagging Model_rf Precision: 0.6850221788781578
Bagging Model_rf Recall: 0.5674511904704321
Bagging Model_rf F1: 0.5880966047953209
Bagging Model Classification Report_rf:              precision    recall
f1-score   support

           1       0.76      0.58      0.66       312
           3       0.70      0.90      0.79       513
           5       0.60      0.22      0.32       124

    accuracy                           0.71       949
   macro avg       0.69      0.57      0.59       949
weighted avg       0.70      0.71      0.68       949

Bagging Model Confusion Matrix_rf:[[181 125   6]
 [ 37 464  12]
 [ 21  76  27]]
```

In [53]:

```
# Confusion matrix visualization
plt.figure(figsize=(10, 7))
sns.heatmap(conf_matrix_rf_bagging, annot=True, cmap='magma', xticklabels=
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Bagging rf Model Confusion Matrix')
plt.show()
```



XGBoost

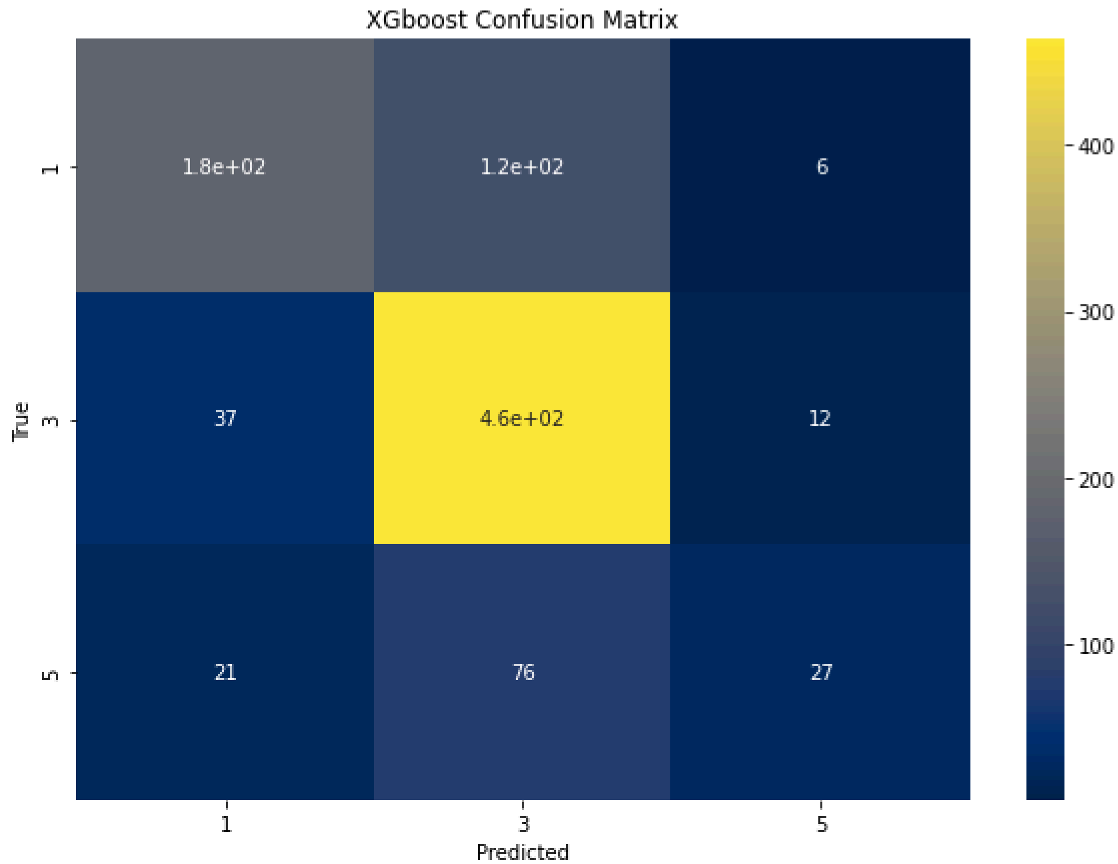
In [54]:

```
# Create a pipeline
pipe3 = Pipeline([
    ('tfidf', TfidfVectorizer()),
    ('xgb', XGBClassifier(random_state =42))
])
# Initialize the Labelencoder
le = LabelEncoder()
# Fit and transform the pipeline
y_train_mult_encoded = le.fit_transform(y_train_mult)
y_test_mult_encoded = le.transform(y_test_mult)
# fit pipeline
pipe3.fit(X_train_mult,y_train_mult_encoded)
# Evaluate
y_pred_encoded = pipe3.predict(X_test_mult)
y_pred = le.inverse_transform(y_pred_encoded)
classif_report_XGboost = classification_report(y_test_mult, y_pred)
conf_matrix_XGboost = (confusion_matrix(y_test_mult, y_pred_rf))

print(f"Classification Report_XG:{classif_report_XGboost}")
print(f"Confusion Matrix_XG:{conf_matrix_XGboost}")
```

```
Classification Report_XG:              precision    recall  f1-score   su
pport

           1       0.72      0.54      0.62       312
           3       0.69      0.88      0.77       513
           5       0.50      0.22      0.30       124

    accuracy                           0.68       949
   macro avg       0.63      0.55      0.56       949
weighted avg       0.67      0.68      0.66       949

Confusion Matrix_XG:[[181 125   6]
 [ 37 464  12]
 [ 21  76  27]]
```

In [55]:

```
plt.figure(figsize=(10, 7))
sns.heatmap(conf_matrix_XGboost, annot=True, cmap='cividis', xticklabels=[
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('XGboost Confusion Matrix')
plt.show()
```



In [56]:

```
# Create a dataframe to compare performance of the model
data = {
    'model': ['LogisticRegression','Bagging Ensemble(rf)','Bagging Ensemble
    'Accuracy': [71,71,69,68],
    'Precision': [68,69,64,63],
    'Recall': [61,57,58,55],
    'F1': [63,59,59,56]
}
df = pd.DataFrame(data)
df
```

Out[56]:

	model	Accuracy	Precision	Recall	F1
0	LogisticRegression	71	68	61	63
1	Bagging Ensemble(rf)	71	69	57	59
2	Bagging Ensemble(DT)	69	64	58	59
3	XGBoost	68	63	55	56

Save and Deploy a model

In [57]:

```
import joblib

# # Save the trained model
model_name = 'Sentiment_model_joblib.pkl'
joblib.dump(best_model,model_name)

# # Save the vectorizer
vec_name = 'TfidfVectorizer_joblib.pkl'
joblib.dump(vectorizer,vec_name)
```

Out[57]:

```
['TfidfVectorizer_joblib.pkl']
```

In [58]:

```
loaded_model = joblib.load('Sentiment_model_joblib.pkl')
loaded_vec = joblib.load('TfidfVectorizer_joblib.pkl')

sample_text = ["My cat only chews @apple cords. Such an #AppleSnob."]

prediction = loaded_model.predict(sample_text)
prediction
```

Out[58]:

```
array(['3'], dtype=object)
```

Achievement of objective

- Insights gained can help business better understand customer sentiment and make Data driven decision to enhance products and service.

Conclusion

- Both Logistic Regression with grid search and Bagged Random Forest performed the best with Accuracy of 71%.
- The Logistic Regression with gridsearch achieved an accuracy of 0.713 and precision of 0.677 while Bagged Random Forest with accuracy of 0.708 and precision of 0.685
- There is still room for improvement, particularly in handling the positive class (5).

Recommendations

By doing the following,the model performance will improve:

- Experimenting with advanced feature engineering,
- Experiment further with different other models
- Implement further ensemble methods

Limitations

- Labeling tweets as "Positive", "Negative" or "No emotion" can be a highly subjective exercise. What I may think is a positive tweet, someone else may interpret as negative.
- The context of these tweets matter. Since we don't know the methodology of how the data was labeled, there could have been human error in labeling where a tweet that was intended to be sarcastic can be labeled incorrectly for example. This would negatively impact the quality of the data.

Next Steps

- Collecting additional data to improve the model.

Type *Markdown* and LaTeX: α^2