Syllabus

1. Introduction to project
2. Dataset description
3. Importing necessary Libraries
4. Exploratory Analysis
5. ML models
6. Conclusion

Introduction to project

This project is the final assignment from *NLP:Twitter Sentiment Analysis* course in Coursera (https://www.coursera.org/learn/twitter-sentiment-analysis/home (https://www.coursera.org/learn/twitter-sentiment-analysis/home)). The project follows the learning techniques from the course. This project is an attempt to gain foundational understanding on Natural Language Processing (NLP). IBM defines Natural language processing (NLP) as the branch of computer science concerned with giving computers the ability to understand text and spoken words in much the same way human beings can. The customer reviews of amazon product *Alexa* is used as the primary dataset of the project. The source of this dataset is : https://www.kaggle.com/datasets/sid321axn/amazon-alexa-reviews (https://www.kaggle.com/datasets/sid321axn/amazon-alexa-reviews) .

Dataset Description

The dataset contains information about the customer reviews of amazon product, Alexa. The dataset has 3150 reviews or observation with 5 attributes. The description of these 5 attributes are:
date : Date of review
variation: physical apperance of the product bought
verified reviews: the review given by the customer
rating: rating given by the customer for the product, 5 being the highest and 1 being the lowest
feedback: 0 is a negative review and 1 is a positive review

Importing the libraries

We will import all the necessary libraries in the very beginning.

In [1]:

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from jupyterthemes import jtplot
!pip install -q wordcloud
from wordcloud import WordCloud
import string
import nltk
import warnings
from nltk.corpus import stopwords
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, confusion_matrix, accuracy_
jtplot.style(theme='monokai' ,context ='notebook', ticks= True, grid=False)
```

Exploratory analysis

In [2]:

```
import pandas as pd
amazon_reviews = pd.read_csv ("amazon_alexa.tsv", sep = '\t')# as the dataset
```

Lets see the first few observations

In [3]:

```
amazon_reviews.head()
```

Out[3]:

	rating	date	variation	verified_reviews	feedback
0	5	31-Jul-18	Charcoal Fabric	Love my Echo!	1
1	5	31-Jul-18	Charcoal Fabric	Loved it!	1
2	4	31-Jul-18	Walnut Finish	Sometimes while playing a game, you can answer...	1
3	5	31-Jul-18	Charcoal Fabric	I have had a lot of fun with this thing. My 4 ...	1
4	5	31-Jul-18	Charcoal Fabric	Music	1

In [4]:
```
amazon_reviews.describe()
```

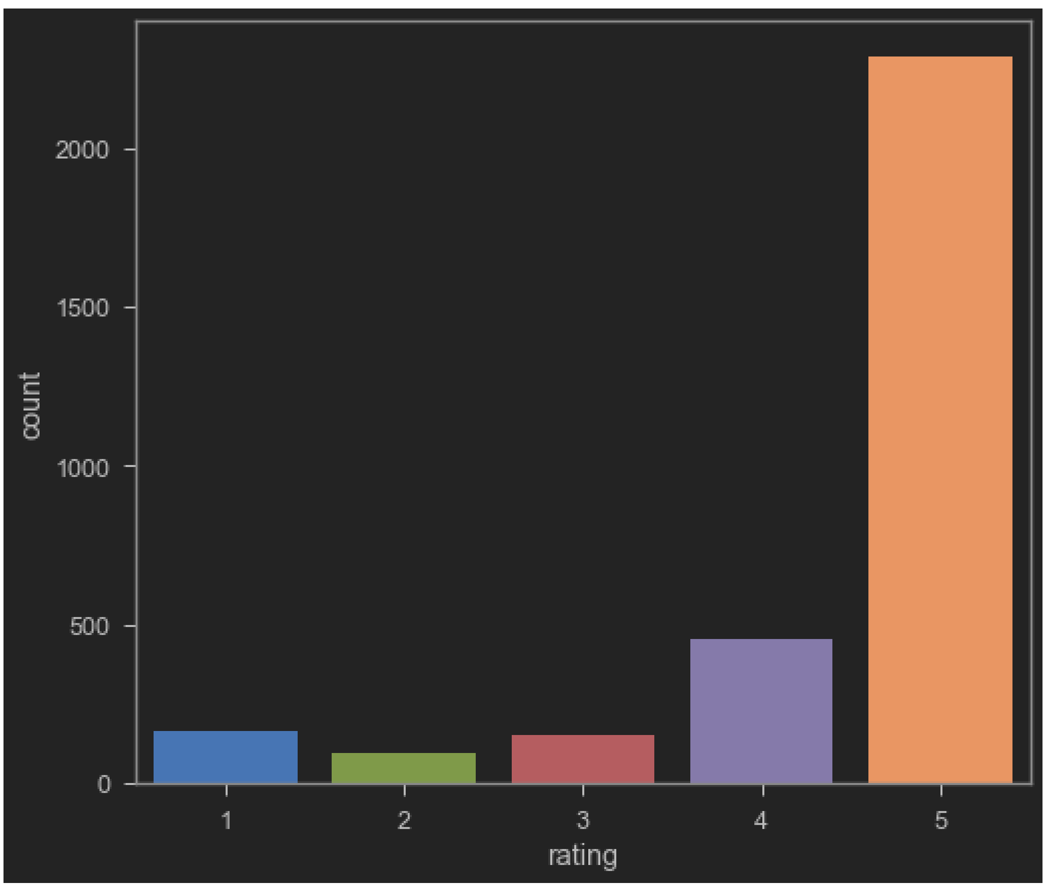
Out[4]:

	rating	feedback
count	3150.000000	3150.000000
mean	4.463175	0.918413
std	1.068506	0.273778
min	1.000000	0.000000
25%	4.000000	1.000000
50%	5.000000	1.000000
75%	5.000000	1.000000
max	5.000000	1.000000

The mean rating is 4.46 which means that the product is popular. We can also visualize this using a countplot.

In [5]:
```
sns.countplot(amazon_reviews['rating'])
```

Out[5]:
```
<AxesSubplot:xlabel='rating', ylabel='count'>
```



The columns are:

In [6]:
```
list(amazon_reviews)
```

Out[6]: ['rating', 'date', 'variation', 'verified_reviews', 'feedback']

The descriptions of these columns are already given in the previous section. Lets see if the dataset contains any null values.

In [7]:
```
sns.heatmap(amazon_reviews.isnull(),yticklabels = False,cbar= False,cmap ="Blu
```

Out[7]: <AxesSubplot:>

As we cab see from the heatmap, it contains no null values. Now we will calculate the characters in each review and input this in a new column named 'length'.

In [8]:
```
amazon_reviews['length'] = amazon_reviews['verified_reviews'].apply(len)
```

The updated dataset with 6 columns in total now looks like this:

In [9]:
```
amazon_reviews.head()
```

Out[9]:

	rating	date	variation	verified_reviews	feedback	length
0	5	31-Jul-18	Charcoal Fabric	Love my Echo!	1	13
1	5	31-Jul-18	Charcoal Fabric	Loved it!	1	9
2	4	31-Jul-18	Walnut Finish	Sometimes while playing a game, you can answer...	1	195
3	5	31-Jul-18	Charcoal Fabric	I have had a lot of fun with this thing. My 4 ...	1	172
4	5	31-Jul-18	Charcoal Fabric	Music	1	5

In [10]:
```
amazon_reviews.describe()
```

Out[10]:

	rating	feedback	length
count	3150.000000	3150.000000	3150.000000
mean	4.463175	0.918413	132.049524
std	1.068506	0.273778	182.099952
min	1.000000	0.000000	1.000000
25%	4.000000	1.000000	30.000000
50%	5.000000	1.000000	74.000000
75%	5.000000	1.000000	165.000000
max	5.000000	1.000000	2851.000000

So, the average length of reviews was 132 characters. Someone left a review with just one character. Let's look at this review.

In [11]:
```
amazon_reviews[ amazon_reviews['length'] == 1 ]['verified_reviews']
```

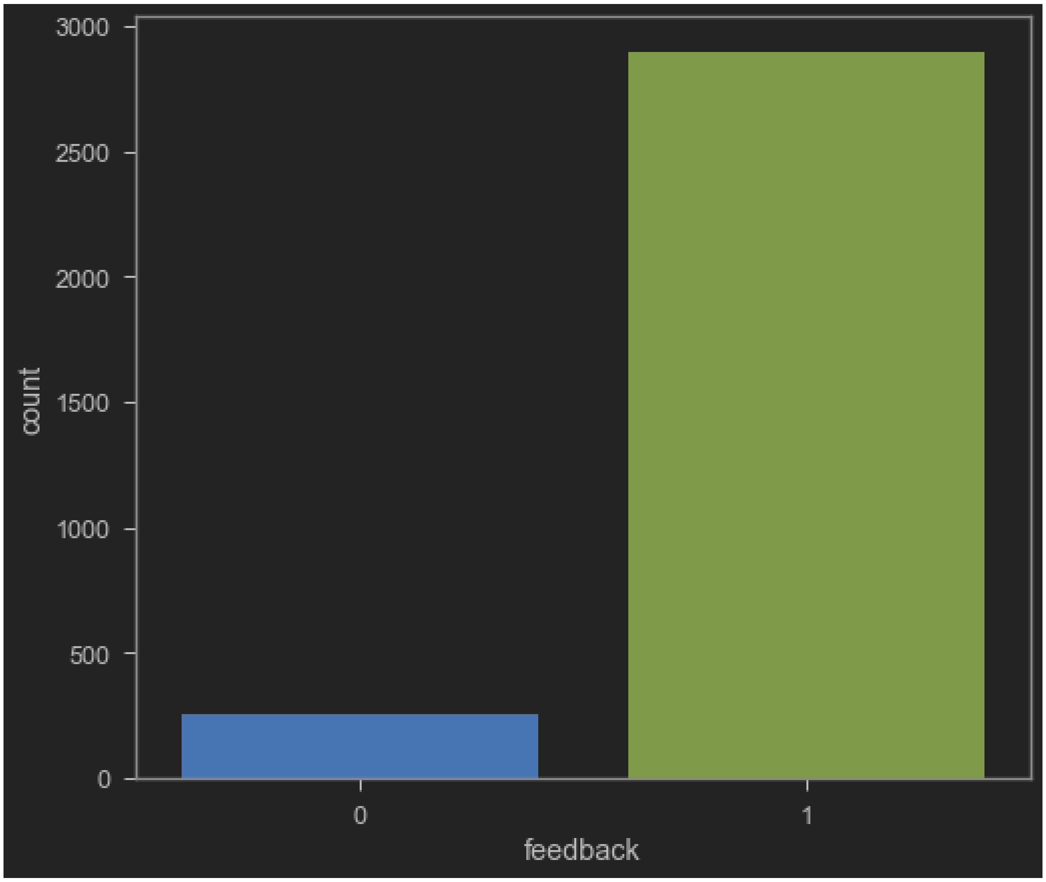
Out[11]:
```
60      😍
85
183
219
374
       ..
3114
3120
3123
3126
3141
Name: verified_reviews, Length: 81, dtype: object
```

So, there were actually 81 reviews with just one character. And, as we can see that they could be emojis.

Now, let us investigate the feedback column.

In [12]:
```
sns.countplot(x = amazon_reviews['feedback'])
```

Out[12]:
```
<AxesSubplot:xlabel='feedback', ylabel='count'>
```



Most of the feedback have the value '1' which means that most of the feedback were positive. This supports our observation from analysing the 'rating' column where we saw that the average rating was 4.46.

Plotting the Wordcloud

We will plot the wordcloud to see the most frequent words used in the reviews. We can gain a lot of insight about the general public sentiment towards the product from the wordcloud.

So, we will create a list that contains all of the reviews and join all the reviews to get a single block of text having the content of all the reviews.

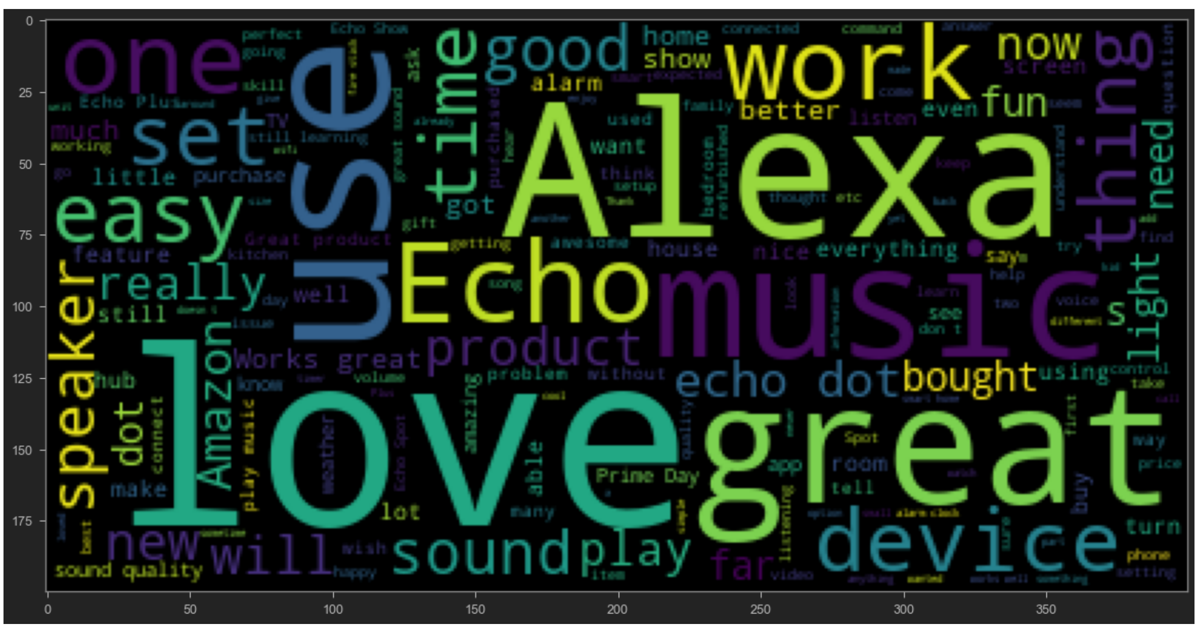
In [13]:
```
sentences = amazon_reviews['verified_reviews'].to_list()
single_block = " ".join(sentences)
single_block # contains all of the reviews in a single block
```

```
things set up love the grocery list app It performs pretty much as expecte
d, but I am very disappointed it will not provide me with the terminology I
need to play various kinds of music without subscribing to the Amazon music
service.  I understand my music choices may be limited, but without knowing
HOW TO ASK, I cannot access the music that is provided free along with my E
cho purchase.  Do you have a chart showing the language I need to use? Easy
to set up. I like the product except that the speakers are not the high qua
lity I expected A great investment. Alexa has helped me out and made me lau
gh. Yet another Exho for our home and love them all!!! I love it, I can pla
n any gender of music from big band to jazz, not easy to find on local radi
o. Sound is good and I can stop it with a voice command if I get a phone ca
ll. Good Value Super easy set up and am loving our new Echo! Whats not to l
ike about this speaker.  Just ask Alexa BEST father\'s day gift. Dad joked
to my mom that Alexa will be the one listening to all of his (repeated) sto
ries going forward. Great addition to our breakfast room kitchen.  Tunes an
d information instantly available.  Slowly learning more features. Sound qu
ality Sad joke. Worthless. Entertainment Very good quality Works great soun
ds great does not miss a beat wish it had a battery for better portability
Loved it till someone stoled it. Can\'t afford to replace it yet but I am g
oing too. Alexa rocks Got this as a gift and love it. I never would have bo
```

Now, we will plot the wordcloud using this single block of text that contains everything.

In [14]:
```
plt.figure(figsize=(20,20))
plt.imshow(WordCloud().generate(single_block))
```

Out[14]: <matplotlib.image.AxesImage at 0x24c229c8100>



As we can see that words like 'love', 'great', 'easy', 'good','light' are the most frequent. This again shows that customers are loving the product. This is great to see as a seller. However, there is always room for improvement in any product. So, this wordcloud is not too useful if you are looking for areas of improvement. Something we can do is to look at the wordcloud for

Lets create a dataset containing negative reviews only.

In [15]:
```
negative = amazon_reviews[amazon_reviews['feedback'] == 0]
negative
```

Out[15]:

	rating	date	variation	verified_reviews	feedback	length
46	2	30-Jul-18	Charcoal Fabric	It's like Siri, in fact, Siri answers more acc...	0	163
111	2	30-Jul-18	Charcoal Fabric	Sound is terrible if u want good music too get...	0	53
141	1	30-Jul-18	Charcoal Fabric	Not much features.	0	18
162	1	30-Jul-18	Sandstone Fabric	Stopped working after 2 weeks ,didn't follow c...	0	87
176	2	30-Jul-18	Heather Gray Fabric	Sad joke. Worthless.	0	20
...	...	...	...	...	...	...
3047	1	30-Jul-18	Black Dot	Echo Dot responds to us when we aren't even ta...	0	120
3048	1	30-Jul-18	White Dot	NOT CONNECTED TO MY PHONE PLAYLIST :(	0	37
3067	2	30-Jul-18	Black Dot	The only negative we have on this product is t...	0	240
3091	1	30-Jul-18	Black Dot	I didn't order it	0	17
3096	1	30-Jul-18	White Dot	The product sounded the same as the emoji spea...	0	210

257 rows × 6 columns

Now, we can follow similar steps as before to plot the wordcloud.

In [16]:
```
neg_sentences = negative['verified_reviews'].tolist()
neg_block =" ".join(neg_sentences)
plt.figure(figsize = (20,20))
plt.imshow(WordCloud().generate(neg_block))
```

Out[16]: <matplotlib.image.AxesImage at 0x24c229791f0>

Here, we can see words that show negative sentiments like 'disappointed', 'problem','doesn't'. As a seller, this can provide us hints at what areas the customers are having problems and where we could make improvements.

The appearance of 'refurbished' could mean that customers are having troubles when they bought refurbished products. The word 'echo' could mean that there is presence of echo, 'work' and 'working' could mean that the product never worked, 'screen' could mean that there are problems with the screen.

Data Cleaning and Count Vectorizer

In the next steps, we will make the data ready for applying models. We will clean the data by removing punctuation and stopwords. Then, we will perform count vectorization in the data (count vectorization is counting the number of occurences of each words in the whole document).

In [17]:
```
def cleaning(text): #this function removes punctuation and stopwords
    removing_punctuation = [char for char in text if char not in string.punctu
    joined = ''.join(removing_punctuation)
    clean_data = [word for word in joined.split() if word.lower() not in stopw
    return clean_data
```

In [18]:
```
# we will apply the function to our reviews
clean_reviews = amazon_reviews['verified_reviews'].apply(cleaning)
```

Lets see how the original reviews compare to the cleaned reviews.

In [19]:
```
print(amazon_reviews['verified_reviews'][1]) #original first review
```

```
Loved it!
```

In [20]:
```
print(clean_reviews[1]) #cleaned first review
```

```
['Loved']
```

As we can see that our function was successful in removing the stopword ('it') and the punctuation ('!').

Now we will perform count vectorization.

In [21]:
```
vectorizer = CountVectorizer(analyzer = cleaning)
reviews_countvectorizer = vectorizer.fit_transform(amazon_reviews['verified_re
```

In [22]:
```
# Let see the unique words in all reviews
print(vectorizer.get_feature_names())
```

```
t', 'aren't', 'argue', 'argument', 'arguments', 'arises', 'arlo', 'arm', 'a
round', 'aroundno', 'array', 'arrive', 'arrived', 'arriving', 'articles',
'artist', 'artistbut', 'artists', 'asap', 'ase', 'ask', 'askMy', 'asked',
'askes', 'asking', 'asleep', 'aspect', 'aspects', 'aspectsa', 'ass', 'assig
ned', 'assist', 'assistance', 'assistant', 'assume', 'assumed', 'assuming',
'assumption', 'atención', 'atmosphere', 'atrás', 'attach', 'attached', 'att
achment', 'attempt', 'attempted', 'attempting', 'attention', 'attractive',
'audible', 'audibleI', 'audibles', 'audio', 'audioApple', 'audiobook', 'aud
iobooks', 'audiophile', 'aunt', 'auto', 'automatic', 'automatically', 'auto
mation', 'aux', 'auxiliary', 'avail', 'availability', 'available', 'avoid',
'awake', 'aware', 'away', 'awesome', 'awesomeunderstands', 'awful', 'awhil
e', 'awkward', 'awsome', 'baby', 'back', 'backThe', 'background', 'backgrou
nds', 'backyard', 'bad', 'baffle', 'baffled', 'ball', 'ban', 'band', 'bandw
agon', 'bandwidth', 'bang', 'bar', 'bare', 'barely', 'bargain', 'bark', 'ba
rn', 'base', 'based', 'basement', 'basic', 'basically', 'bass', 'bathroom',
'bathrooms', 'batman', 'batteries', 'battery', 'bc', 'beam', 'beat', 'beaut
iful', 'beautifully', 'beauty', 'became', 'becausse', 'become', 'becomes',
'becoming', 'bed', 'bedMany', 'bedroom', 'bedrooms', 'bedside', 'bedtime',
'beefy', 'begin', 'beginners', 'beginning', 'begun', 'behaved', 'behind',
'believe', 'believer', 'bells', 'belong', 'benefit', 'benefits', 'beside',
```

In [23]:
```
# the array of our reviews, this will be used as input for our model
print(reviews_countvectorizer.toarray())
```

```
[[0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 ...
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]]
```

In [24]:
```
# let's see the shape of pur array
reviews_countvectorizer.shape
```

Out[24]:
```
(3150, 5211)
```

This means that we have 3150 reviews and 5211 unique words.

ML Models

Now that we have cleaned the reviews and transformed them into array form, we can apply machine learning models to perform classification on the data. Our categorical variable 'feedback' is the outcome variable we are trying to predict based on our predictor variable which is the array from reviews. **Our goal is to create a model that can predict whethere a review is 'positive' or 'negative'.**

The predictor variable or the array is 'X' and the outcome variable or the 'feedback' is y.

In [25]:
```
reviews = pd.DataFrame(reviews_countvectorizer.toarray())
X = reviews
y = amazon_reviews['feedback']
```

As we are performing supervised learning on the reviews, we need to split the data into training and testing data. We will use a 70/30 ratio to split the data.

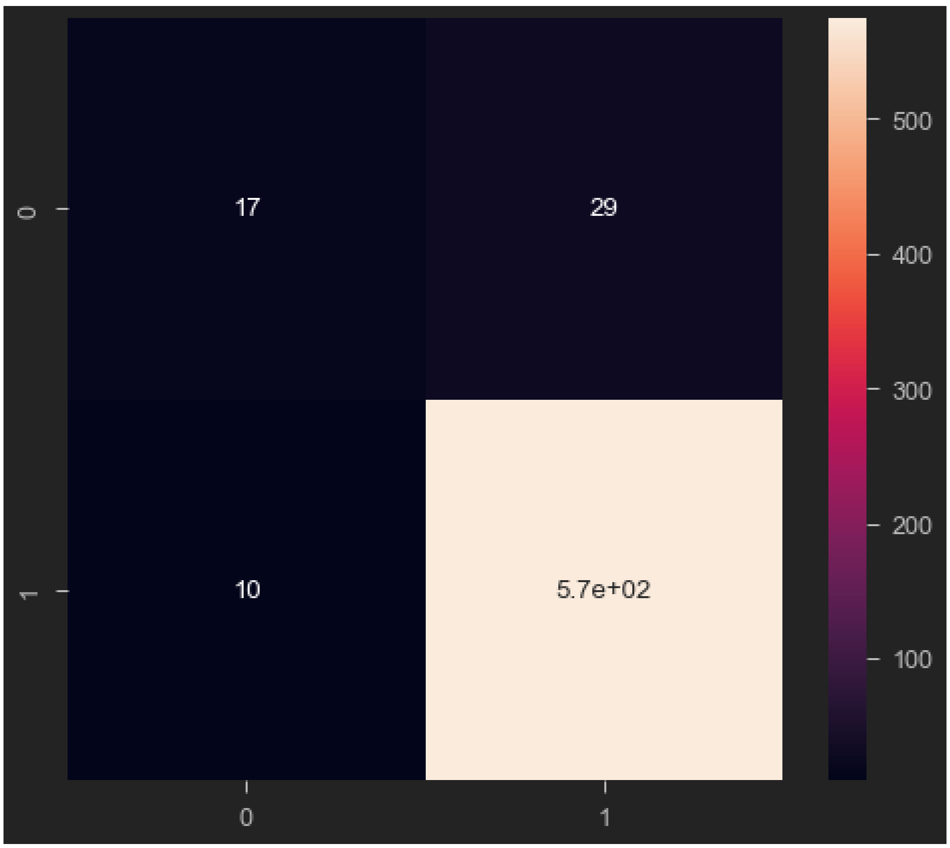
In [26]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

Naive Bayes

In [27]:
```
NB_classifier  =MultinomialNB()
NB_classifier.fit(X_train,y_train)
y_predict = NB_classifier.predict(X_test)
```

In [28]:
```
# Lets see the perfomance of our model
cm = confusion_matrix(y_test, y_predict)
sns.heatmap(cm, annot=True)
print(accuracy_score(y_test,y_predict))
print(classification_report(y_test,y_predict))
```

```
0.9380952380952381
              precision    recall  f1-score   support

           0       0.63      0.37      0.47        46
           1       0.95      0.98      0.97       584

    accuracy                           0.94       630
   macro avg       0.79      0.68      0.72       630
weighted avg       0.93      0.94      0.93       630
```



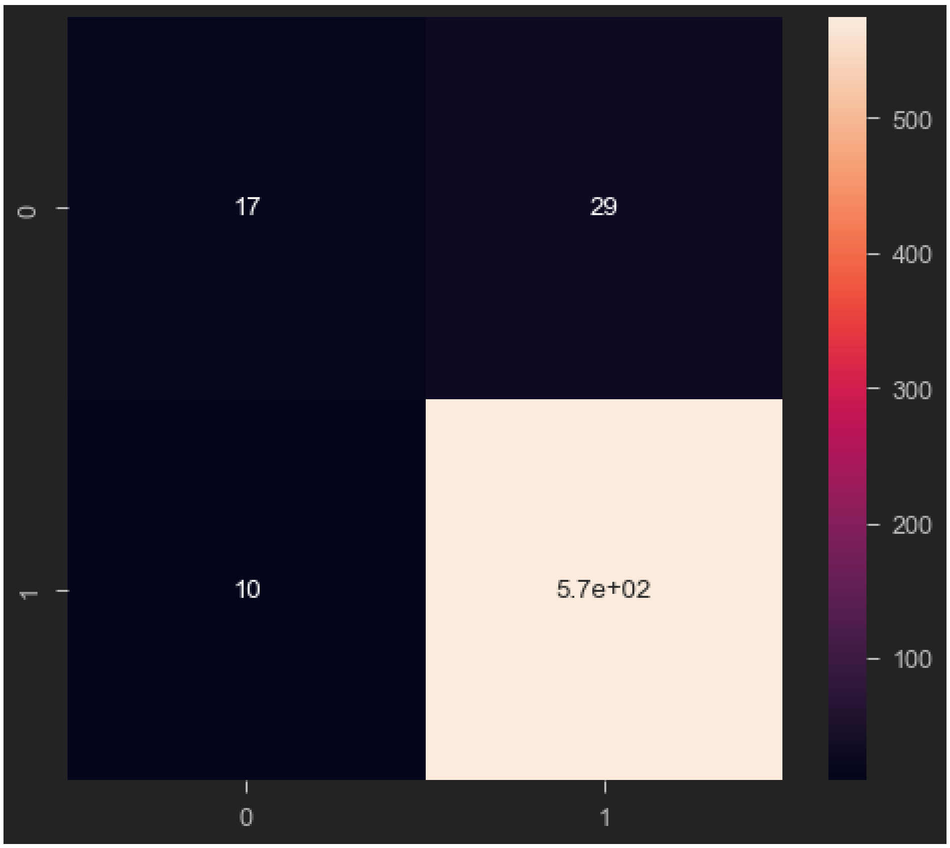
So, our model performs pretty good with an accuracy of 94%.

Logistic Regression

In [29]:
```
logit = LogisticRegression()
logit.fit(X_train, y_train)
y_pred_logit = logit.predict(X_test)
```

In [30]:
```
# Lets see the perfomance of our model
cm_logit = confusion_matrix(y_test, y_predict)
sns.heatmap(cm_logit, annot=True)
print(accuracy_score(y_test,y_pred_logit))
print(classification_report(y_test,y_pred_logit))
```

```
0.9317460317460318
              precision    recall  f1-score   support

           0       0.58      0.24      0.34        46
           1       0.94      0.99      0.96       584

    accuracy                           0.93       630
   macro avg       0.76      0.61      0.65       630
weighted avg       0.92      0.93      0.92       630
```



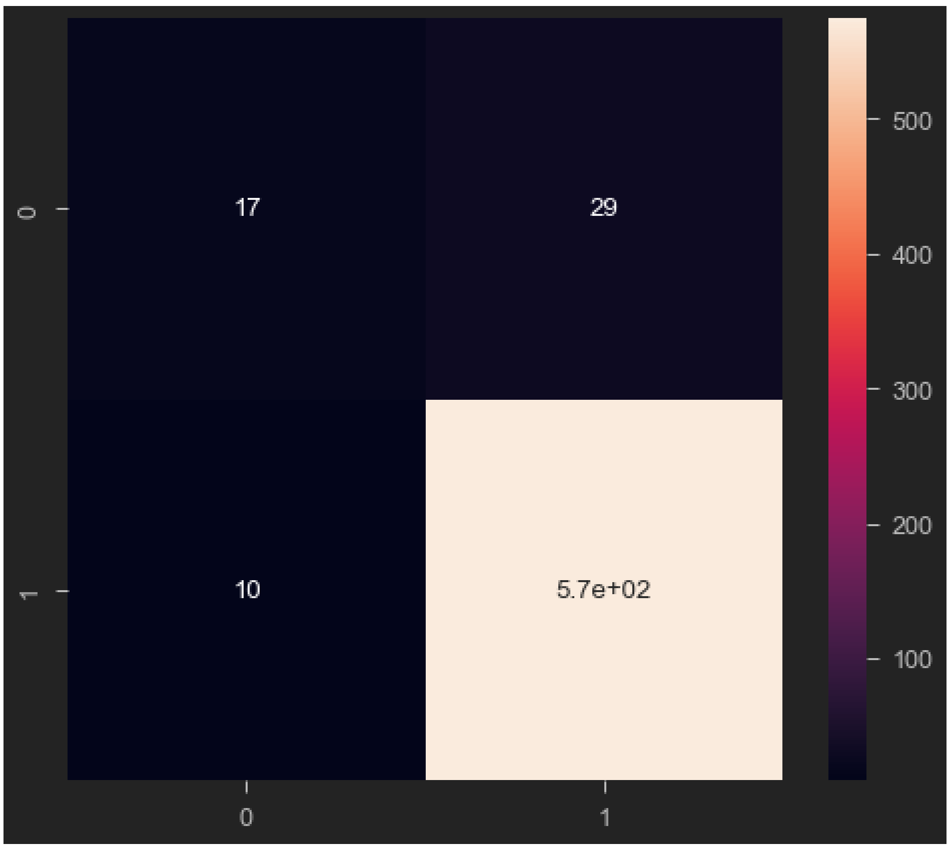
The accuracy of our model is 93.1 %.

Gradient Boost

In [31]:
```
XG = GradientBoostingClassifier()
XG.fit(X_train, y_train)

y_pred_XG = XG.predict(X_test)
```

In [32]:
```
# Lets see the perfomance of our model
cm_XG = confusion_matrix(y_test, y_predict)
sns.heatmap(cm_XG, annot=True)
print(accuracy_score(y_test,y_pred_XG))
print(classification_report(y_test,y_pred_XG))
```

```
0.9317460317460318
              precision    recall  f1-score   support

           0       0.67      0.13      0.22        46
           1       0.94      0.99      0.96       584

    accuracy                           0.93       630
   macro avg       0.80      0.56      0.59       630
weighted avg       0.92      0.93      0.91       630
```



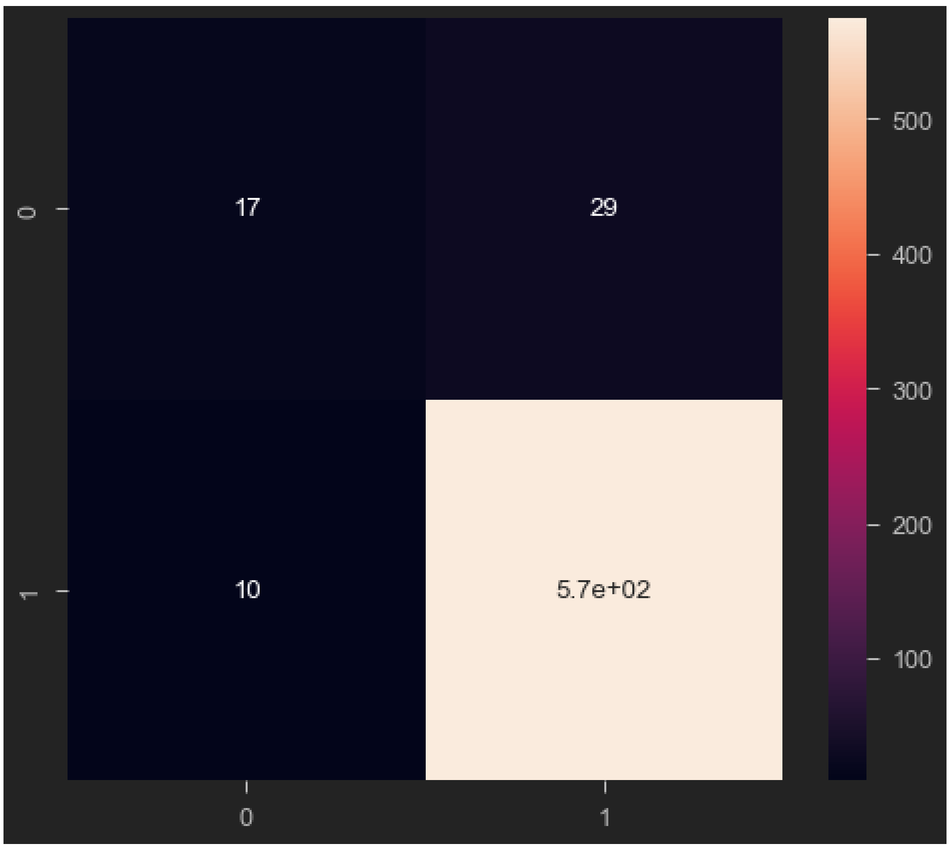
The accuracy of our model is 92.3 %.

Decision Tree

In [33]:
```
DT = DecisionTreeClassifier()
DT.fit(X_train, y_train)
y_pred_DT = DT.predict(X_test)
```

In [34]:
```
# Lets see the perfomance of our model
cm_DT = confusion_matrix(y_test, y_predict)
sns.heatmap(cm_DT, annot=True)
print(accuracy_score(y_test,y_pred_DT))
print(classification_report(y_test,y_pred_DT))
```

```
0.9301587301587302
              precision    recall  f1-score   support

           0       0.53      0.39      0.45        46
           1       0.95      0.97      0.96       584

    accuracy                           0.93       630
   macro avg       0.74      0.68      0.71       630
weighted avg       0.92      0.93      0.93       630
```



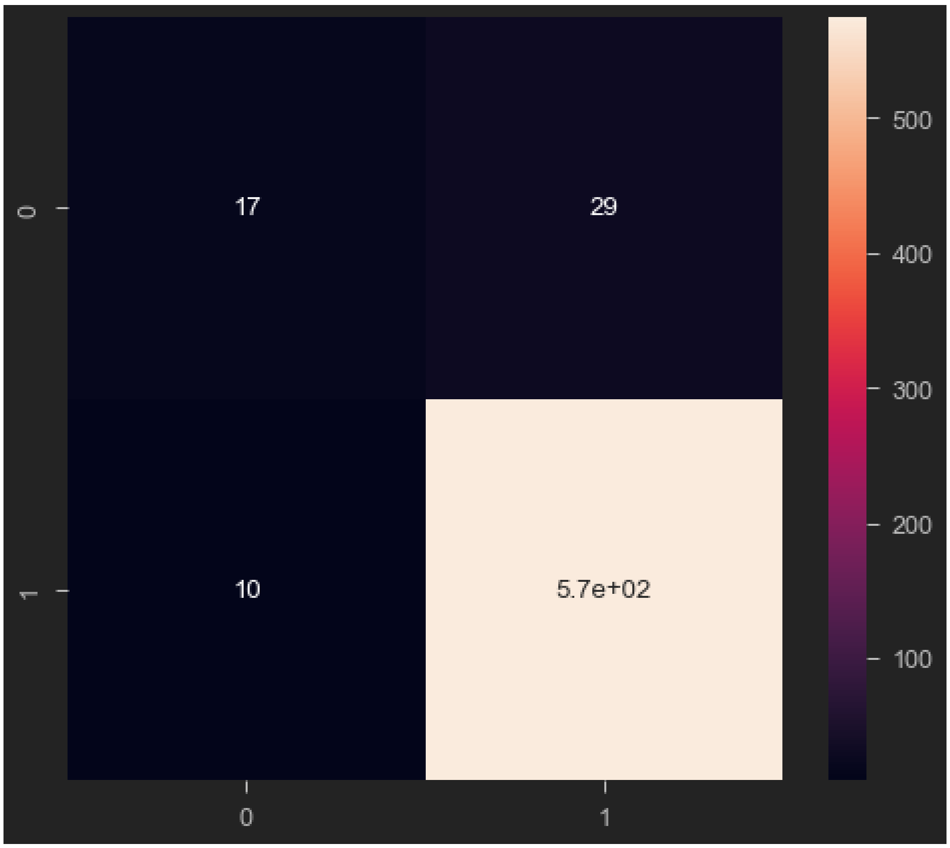
The accuracy of our Decision Tree model is 92.8 %.

Random Forest

In [35]:
```
RF = RandomForestClassifier()
RF.fit(X_train, y_train)
y_pred_RF = RF.predict(X_test)
```

In [36]:
```
# Lets see the perfomance of our model
cm_RF = confusion_matrix(y_test, y_predict)
sns.heatmap(cm_RF, annot=True)
print(accuracy_score(y_test,y_pred_RF))
print(classification_report(y_test,y_pred_RF))
```

```
0.9396825396825397
              precision    recall  f1-score   support

           0       0.90      0.20      0.32        46
           1       0.94      1.00      0.97       584

    accuracy                           0.94       630
   macro avg       0.92      0.60      0.64       630
weighted avg       0.94      0.94      0.92       630
```



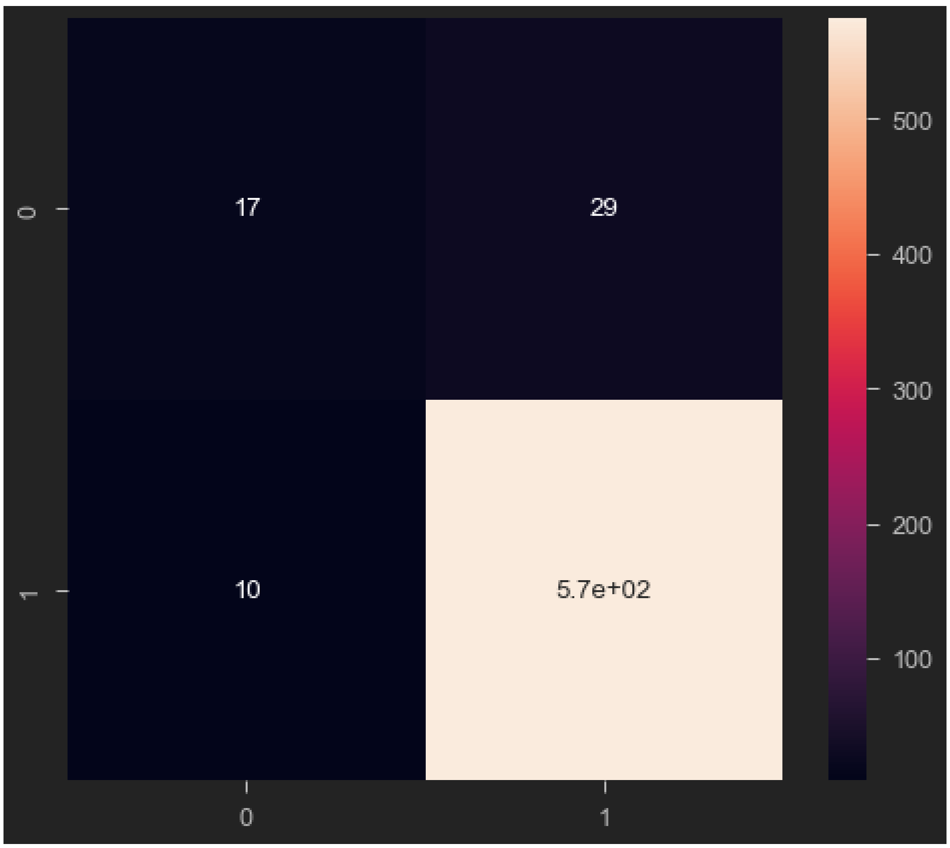
The accuracy of our random forest model is 93.5 %.

Support Vector Machine

In [37]:
```
SVC = SVC()
SVC.fit(X_train, y_train)
y_pred_SVC = SVC.predict(X_test)
```

In [38]:
```
# Lets see the perfomance of our model
cm_SVC = confusion_matrix(y_test, y_predict)
sns.heatmap(cm_SVC, annot=True)
print(accuracy_score(y_test,y_pred_SVC))
print(classification_report(y_test,y_pred_SVC))
```

```
0.9301587301587302
              precision    recall  f1-score   support

           0       1.00      0.04      0.08        46
           1       0.93      1.00      0.96       584

    accuracy                           0.93       630
   macro avg       0.96      0.52      0.52       630
weighted avg       0.94      0.93      0.90       630
```



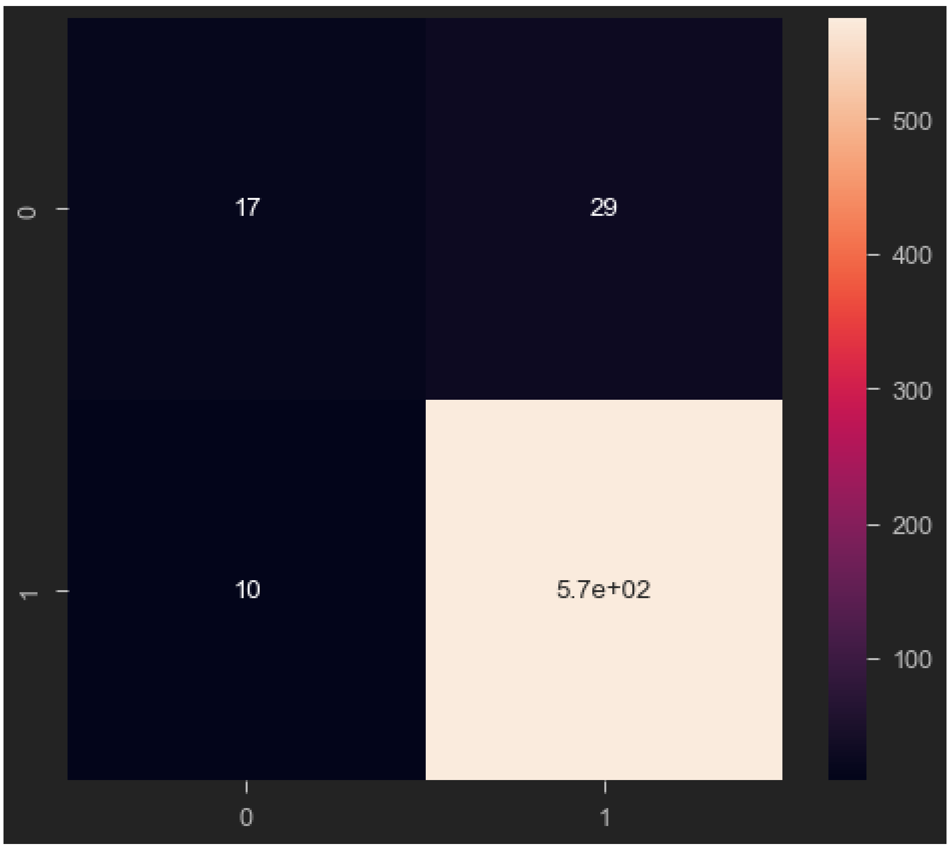
The accuracy of our support vector machine model is 92.06 %.

K-Nearest Neighbour (KNN)

In [39]:
```
KNN = KNeighborsClassifier()
KNN.fit(X_train, y_train)
y_pred_KNN = KNN.predict(X_test)
```

In [40]:
```
# Lets see the perfomance of our model
cm_KNN = confusion_matrix(y_test, y_predict)
sns.heatmap(cm_KNN, annot=True)
print(accuracy_score(y_test,y_pred_KNN))
print(classification_report(y_test,y_pred_KNN))
```

```
0.919047619047619
              precision    recall  f1-score   support

           0       0.31      0.09      0.14        46
           1       0.93      0.98      0.96       584

    accuracy                           0.92       630
   macro avg       0.62      0.54      0.55       630
weighted avg       0.89      0.92      0.90       630
```



The accuracy of our KNN model is 90.6 %.

So, the best performing machine learning model for our data is **Naive Bayes** with an accuracy of 94.1 %.

Conclusion

Thus, we successfully found a machine learning model capable of predicting whether a review is positive or negative with a high accuracy of 94.1 %.