PROBLEM STATEMENT

- The SMS Spam Collection is a set of SMS tagged messages that have been collected for SMS Spam research. It contains one set of SMS messages in English of 5,574 messages, tagged acording being ham (legitimate) or spam.
- The files contain one message per line. Each line is composed by two columns: v1 contains the label (ham or spam) and v2 contains the raw text.

STEP #0: LIBRARIES IMPORT

In [1]:

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

STEP #1: IMPORT DATASET

In [2]:

```
spam_df = pd.read_csv("emails.csv")
```

In [3]:

```
spam_df.head(10)
```

Out[3]:

	text	spam
0	Subject: naturally irresistible your corporate...	1
1	Subject: the stock trading gunslinger fanny i...	1
2	Subject: unbelievable new homes made easy im ...	1
3	Subject: 4 color printing special request add...	1
4	Subject: do not have money , get software cds ...	1
5	Subject: great nnews hello , welcome to medzo...	1
6	Subject: here ' s a hot play in motion homela...	1
7	Subject: save your money buy getting this thin...	1
8	Subject: undeliverable : home based business f...	1
9	Subject: save your money buy getting this thin...	1

In [4]:

```
spam_df.tail()
```

Out[4]:

	text	spam
5723	Subject: re : research and development charges...	0
5724	Subject: re : receipts from visit jim , than...	0
5725	Subject: re : enron case study update wow ! a...	0
5726	Subject: re : interest david , please , call...	0
5727	Subject: news : aurora 5 . 2 update aurora ve...	0

In [5]:

```
spam_df.describe()
```

Out[5]:

	spam
count	5728.000000
mean	0.238827
std	0.426404
min	0.000000
25%	0.000000
50%	0.000000
75%	0.000000
max	1.000000

In [6]:

```
spam_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5728 entries, 0 to 5727
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   text    5728 non-null   object
 1   spam    5728 non-null   int64 
dtypes: int64(1), object(1)
memory usage: 89.6+ KB
```

STEP #2: VISUALIZE DATASET

In [7]:

```
# Let's see which message is the most popular ham/spam message
spam_df.groupby('spam').describe()
```

Out[7]:

	text			
	count	**unique**	**top**	**freq**
spam				
0	4360	4327	Subject: re : term project : brian , no prob...	2
1	1368	1368	Subject: considered unsolicited bulk email fro...	1

In [8]:

```
# Let's get the length of the messages
spam_df['length'] = spam_df['text'].apply(len)
spam_df.head()
```

Out[8]:

	text	spam	length
0	Subject: naturally irresistible your corporate...	1	1484
1	Subject: the stock trading gunslinger fanny i...	1	598
2	Subject: unbelievable new homes made easy im ...	1	448
3	Subject: 4 color printing special request add...	1	500
4	Subject: do not have money , get software cds ...	1	235

In [9]:

```
spam_df
```

Out[9]:

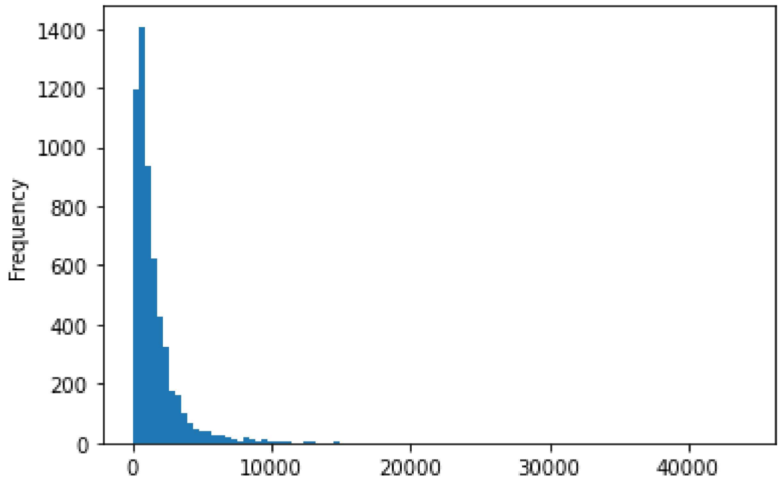
	text	spam	length
0	Subject: naturally irresistible your corporate...	1	1484
1	Subject: the stock trading gunslinger fanny i...	1	598
2	Subject: unbelievable new homes made easy im ...	1	448
3	Subject: 4 color printing special request add...	1	500
4	Subject: do not have money , get software cds ...	1	235
...	...	...	...
5723	Subject: re : research and development charges...	0	1189
5724	Subject: re : receipts from visit jim , than...	0	1167
5725	Subject: re : enron case study update wow ! a...	0	2131
5726	Subject: re : interest david , please , call...	0	1060

In [10]:

```
spam_df['length'].plot(bins=100, kind='hist')
```

Out[10]:

<AxesSubplot:ylabel='Frequency'>



In [11]:

```
spam_df.length.describe()
```

Out[11]:

```
count     5728.000000
mean      1556.768680
std       2042.649812
min         13.000000
25%        508.750000
50%        979.000000
75%       1894.250000
max      43952.000000
Name: length, dtype: float64
```

In [12]:

```
# Let's see the longest message 43952
spam_df[spam_df['length'] == 43952]['text'].iloc[0]
```

Out[12]:

```
'Subject: from the enron india newsdesk - april 27 th newsclips  fyi news
articles from indian press .  - - - - - - - - - - - - - - - - - - - - - - - - -
forwarded by sandeep kohli / enron _ development on 04 / 27 / 2001 08 : 2
4 am - - - - - - - - - - - - - - - - - - - - - - - - - - -  nikita varma
04 / 27 / 2001 07 : 51 am  to : nikita varma / enron _ development @ enro
n _ development  cc : ( bcc : sandeep kohli / enron _ development )  subj
ect : from the enron india newsdesk - april 27 th newsclips  friday apr 2
7 2001 , http : / / www . economictimes . com / today / cmo 3 . htm  dpc
board empowers md to cancel mseb contract  friday apr 27 2001 , http : /
/ www . economictimes . com / today / 27 compl 1 . htm  mseb pays rs 134
cr under \' protest \' to dpc  friday , april 27 , 001 , http : / / www .
businessstandard . com / today / economy 4 . asp ? menu = 3  enron india
md authorised to terminate ppa  friday , april 27 , 2001 , http : / / www
. financialexpress . com / fe 20010427 / topl . html  foreign lenders sla
m brakes on disbursements to dpc , sanjay jog & raghu mohan  global banks
comfortable with enron pull - out  friday , april 27 , 2001 , http : / /
www . indian - express . com / ie 20010427 / nat 23 . html  enron : dabho
l chief gets powers to end deal with the mseb  friday , april 27 , 2001 ,
```

In [13]:

```
# Let's divide the messages into spam and ham
```

In [14]:

```
ham = spam_df[spam_df['spam']==0]
```

In [15]:

```
spam = spam_df[spam_df['spam']==1]
```

In [16]:

```
ham
```

Out[16]:

	text	spam	length
1368	Subject: hello guys , i ' m " bugging you " f...	0	1188
1369	Subject: sacramento weather station fyi - - ...	0	1997
1370	Subject: from the enron india newsdesk - jan 1...	0	7902
1371	Subject: re : powerisk 2001 - your invitation ...	0	3644
1372	Subject: re : resco database and customer capt...	0	5535
...	...	...	...
5723	Subject: re : research and development charges...	0	1189
5724	Subject: re : receipts from visit jim , than...	0	1167
5725	Subject: re : enron case study update wow ! a...	0	2131
5726	Subject: re : interest david , please , call...	0	1060
5727	Subject: news : aurora 5 . 2 update aurora ve...	0	2331

4360 rows × 3 columns

In [17]:

```
spam
```

Out[17]:

	text	spam	length
0	Subject: naturally irresistible your corporate...	1	1484
1	Subject: the stock trading gunslinger fanny i...	1	598
2	Subject: unbelievable new homes made easy im ...	1	448
3	Subject: 4 color printing special request add...	1	500
4	Subject: do not have money , get software cds ...	1	235
...	...	...	...
1363	Subject: are you ready to get it ? hello ! v...	1	347
1364	Subject: would you like a $ 250 gas card ? do...	1	188
1365	Subject: immediate reply needed dear sir , i...	1	3164
1366	Subject: wanna see me get fisted ? fist bang...	1	734
1367	Subject: hot stock info : drgv announces anoth...	1	9342

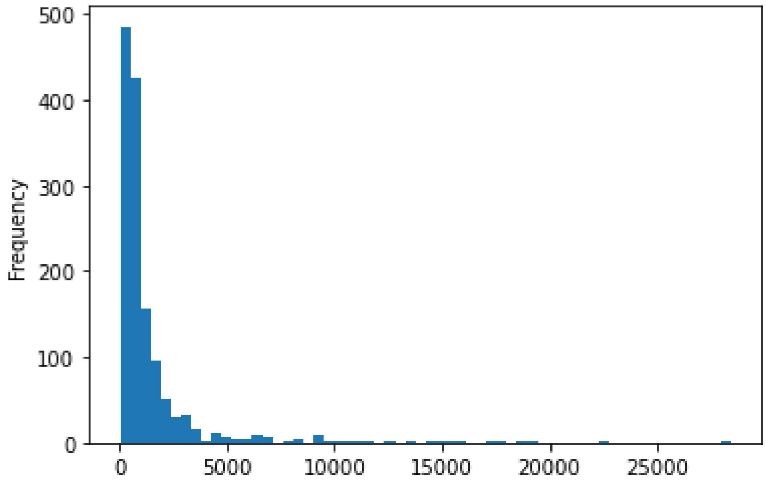
1368 rows × 3 columns

In [18]:

```
spam['length'].plot(bins=60, kind='hist')
```

Out[18]:

```
<AxesSubplot:ylabel='Frequency'>
```

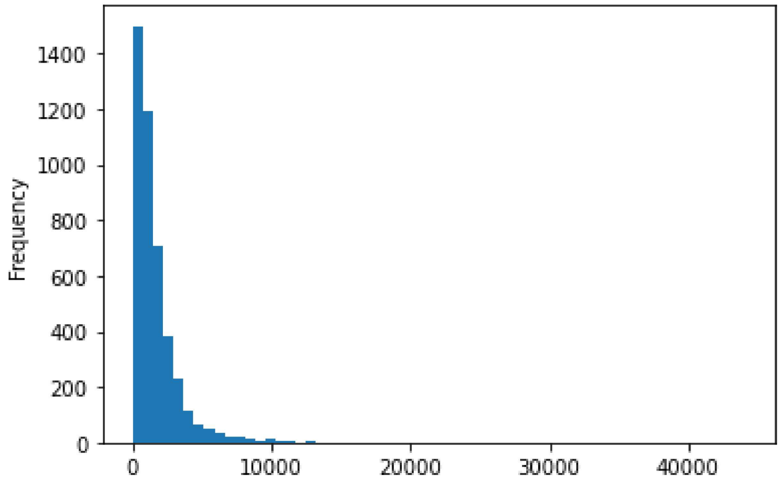


In [19]:

```
ham['length'].plot(bins=60, kind='hist')
```

Out[19]:

```
<AxesSubplot:ylabel='Frequency'>
```



In [20]:

```
print( 'Spam percentage =', (len(spam) / len(spam_df) )*100,"%")
```

```
Spam percentage = 23.88268156424581 %
```

In [21]:

```
print( 'Ham percentage =', (len(ham) / len(spam_df) )*100,"%")
```

```
Ham percentage = 76.11731843575419 %
```

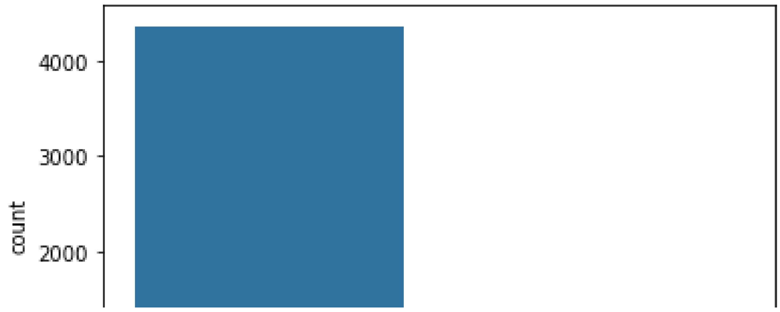
In [22]:

```
sns.countplot(spam_df['spam'], label = "Count")
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\_decorators.py:36: Fut
ureWarning: Pass the following variable as a keyword arg: x. From version
0.12, the only valid positional argument will be `data`, and passing othe
r arguments without an explicit keyword will result in an error or misint
erpretation.
  warnings.warn(
```

Out[22]:

```
<AxesSubplot:xlabel='spam', ylabel='count'>
```



STEP #3: CREATE TESTING AND TRAINING DATASET/DATA CLEANING

STEP 3.1 EXERCISE: REMOVE PUNCTUATION

In [23]:

```
import string
string.punctuation
```

Out[23]:

```
'!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~'
```

In [24]:

```
Test = 'Hello Mr. Future, I am so happy to be learning AI now!!'
```

In [25]:

```
Test_punc_removed = [char for char in Test if char not in string.punctuation]
Test_punc_removed
```

Out[25]:

```
['H',
 'e',
 'l',
 'l',
 'o',
 ' ',
 'M',
 'r',
 ' ',
 'F',
 'u',
 't',
 'u',
 'r',
 'e',
 ' ',
 'I',
 ' ',
```

In [26]:

```
# Join the characters again to form the string.
Test_punc_removed_join = ''.join(Test_punc_removed)
Test_punc_removed_join
```

Out[26]:

```
'Hello Mr Future I am so happy to be learning AI now'
```

STEP 3.2 EXERCISE: REMOVE STOPWORDS

In [28]:

```
import nltk
nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\saile\AppData\Roaming\nltk_data...
[nltk_data]   Unzipping corpora\stopwords.zip.
```

Out[28]:

```
True
```

In [29]:

```
# You have to download stopwords Package to execute this command
from nltk.corpus import stopwords
stopwords.words('english')
```

Out[29]:

```
['i',
 'me',
 'my',
 'myself',
 'we',
 'our',
 'ours',
 'ourselves',
 'you',
 "you're",
 "you've",
 "you'll",
 "you'd",
 'your',
 'yours',
 'yourself',
 'yourselves',
 'he',
```

In [30]:

```
Test_punc_removed_join
```

Out[30]:

```
'Hello Mr Future I am so happy to be learning AI now'
```

In [31]:

```
Test_punc_removed_join_clean = [word for word in Test_punc_removed_join.split() if word.low
```

In [32]:

```
Test_punc_removed_join_clean # Only important (no so common) words are left
```

Out[32]:

```
['Hello', 'Mr', 'Future', 'happy', 'learning', 'AI']
```

STEP 3.3 EXERCISE: COUNT VECTORIZER EXAMPLE

In [33]:

```
from sklearn.feature_extraction.text import CountVectorizer
sample_data = ['This is the first document.','This document is the second document.','And t

vectorizer = CountVectorizer()
X = vectorizer.fit_transform(sample_data)
```

In [34]:

```
print(vectorizer.get_feature_names())
```

```
['and', 'document', 'first', 'is', 'one', 'second', 'the', 'third', 'this']
```

In [35]:

```
print(X.toarray())
```

```
[[0 1 1 1 0 0 1 0 1]
 [0 2 0 1 0 1 1 0 1]
 [1 0 0 1 1 0 1 1 1]
 [0 1 1 1 0 0 1 0 1]]
```

LET'S APPLY THE PREVIOUS THREE PROCESSES TO OUR SPAM/HAM EXAMPLE

In [36]:

```
# Let's define a pipeline to clean up all the messages
# The pipeline performs the following: (1) remove punctuation, (2) remove stopwords

def message_cleaning(message):
    Test_punc_removed = [char for char in message if char not in string.punctuation]
    Test_punc_removed_join = ''.join(Test_punc_removed)
    Test_punc_removed_join_clean = [word for word in Test_punc_removed_join.split() if word
    return Test_punc_removed_join_clean
```

In [ ]:

```
# Let's test the newly added function
spam_df_clean = spam_df['text'].apply(message_cleaning)
```

In [ ]:

```
print(spam_df_clean[0])
```

In [39]:

```
print(spam_df['text'][0])
```

```
Subject: naturally irresistible your corporate identity  lt is really hard t
o recollect a company : the  market is full of suqgestions and the informati
on isoverwhelminq ; but a good  catchy logo , stylish statlonery and outstan
ding website  will make the task much easier .  we do not promise that havin
q ordered a iogo your  company will automaticaily become a world ieader : it
isguite ciear that  without good products , effective business organization
and practicable aim it  will be hotat nowadays market ; but we do promise th
at your marketing efforts  will become much more effective . here is the lis
t of clear  benefits : creativeness : hand - made , original logos , special
ly done  to reflect your distinctive company image . convenience : logo and
stationery  are provided in all formats ; easy - to - use content management
system letsyou  change your website content and even its structure . promptn
ess : you  will see logo drafts within three business days . affordability :
your  marketing break - through shouldn ' t make gaps in your budget . 100 %
satisfaction  guaranteed : we provide unlimited amount of changes with no ex
tra fees for you to  be surethat you will love the result of this collaborat
ion . have a look at our  portfolio _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ not interest
ed . . . _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
```

LET'S APPLY COUNT VECTORIZER TO OUR MESSAGES LIST

In [40]:

```
from sklearn.feature_extraction.text import CountVectorizer
# Define the cleaning pipeline we defined earlier
vectorizer = CountVectorizer(analyzer = message_cleaning)
spamham_countvectorizer = vectorizer.fit_transform(spam_df['text'])
```

In [41]:

```
print(vectorizer.get_feature_names())
```

```
['\x01', '\x02', '\x03', '\x05', '\x06', '\x07', '\x08', '\x0f', '\x10',
'\x12', '\x14', '\x15', '\x19', '0', '00', '000', '0000', '000000', '0000
0000', '0000000000', '000000000003619', '000000000003991', '0000000000039
97', '000000000005168', '000000000005409', '000000000005411', '0000000000
05412', '000000000005413', '000000000005820', '000000000006238', '0000000
00006452', '000000000007494', '000000000007498', '000000000007876', '0000
00000010552', '000000000011185', '000000000012677', '000000000012734', '0
00000000012735', '000000000012736', '000000000012738', '000000000012741',
'000000000012987', '000000000013085', '000000000013287', '00000000001538
4', '000000000015793', '000000000023619', '000000000024099', '00000000002
5307', '000000000025312', '000010220', '0000102317', '0000102374', '00001
02789', '0000104281', '0000104282', '0000104486', '0000104631', '00001047
30', '0000104776', '0000104778', '0000107043', '0000108729', '000066', '0
001', '000166', '0002', '000202', '0003', '0004', '0005', '0006', '0007
6', '0009249480', '0009249481', '0009249504', '0009249505', '0009249506',
'001', '0011', '0015', '00193', '002', '00225', '00235424', '002813', '00
29', '003', '0031', '003399', '00343938', '004', '0044', '00453', '005',
'0052', '0054', '0057', '006', '0061', '00623', '007', '0080', '00971',
'01', '010', '0100', '01019', '0102', '0107', '01075', '0109', '011', '01
```

In [42]:

```
print(spamham_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 [43]:

```
spamham_countvectorizer.shape
```

Out[43]:

```
(5728, 37229)
```

STEP#4: TRAINING THE MODEL WITH ALL DATASET

In [44]:

```
from sklearn.naive_bayes import MultinomialNB

NB_classifier = MultinomialNB()
label = spam_df['spam'].values
NB_classifier.fit(spamham_countvectorizer, label)
```

Out[44]:

```
MultinomialNB()
```

In [45]:

```
testing_sample = ['Free money!!!', "Hi Kim, Please let me know if you need any further info
testing_sample_countvectorizer = vectorizer.transform(testing_sample)
```

In [46]:

```
test_predict = NB_classifier.predict(testing_sample_countvectorizer)
test_predict
```

Out[46]:

```
array([1, 0], dtype=int64)
```

DIVIDE THE DATA INTO TRAINING AND TESTING PRIOR TO TRAINING

In [49]:

```
X = spamham_countvectorizer
y = label
```

In [50]:

```
X.shape
```

Out[50]:

```
(5728, 37229)
```

In [51]:

```
y.shape
```

Out[51]:

```
(5728,)
```

In [52]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

In [53]:

```
from sklearn.naive_bayes import MultinomialNB

NB_classifier = MultinomialNB()
NB_classifier.fit(X_train, y_train)
```

Out[53]:

```
MultinomialNB()
```

In [54]:

```
# from sklearn.naive_bayes import GaussianNB
# NB_classifier = GaussianNB()
# NB_classifier.fit(X_train, y_train)
```

STEP#5: EVALUATING THE MODEL

In [55]:

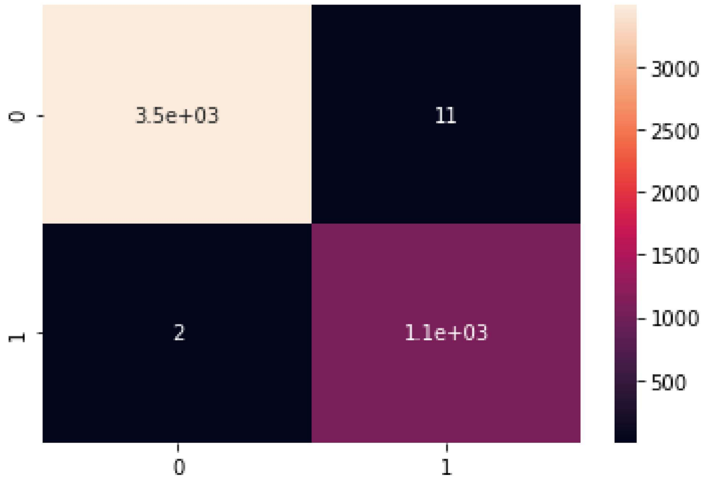
```
from sklearn.metrics import classification_report, confusion_matrix
```

In [56]:

```
y_predict_train = NB_classifier.predict(X_train)
y_predict_train
cm = confusion_matrix(y_train, y_predict_train)
sns.heatmap(cm, annot=True)
```

Out[56]:

```
<AxesSubplot:>
```

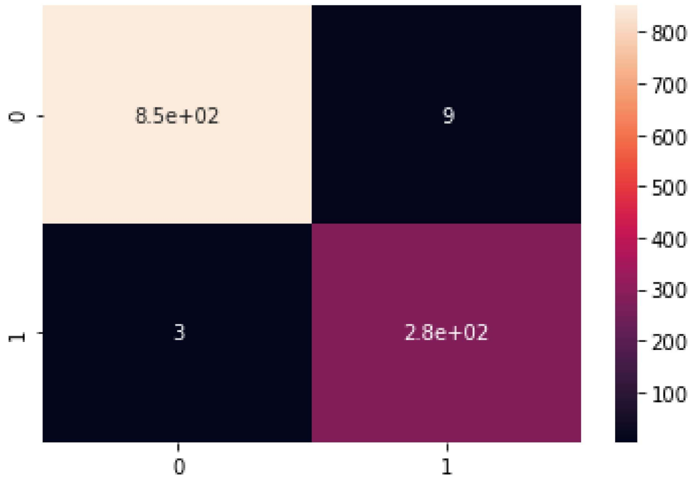


In [57]:

```
# Predicting the Test set results
y_predict_test = NB_classifier.predict(X_test)
cm = confusion_matrix(y_test, y_predict_test)
sns.heatmap(cm, annot=True)
```

Out[57]:

```
<AxesSubplot:>
```



In [58]:

```
print(classification_report(y_test, y_predict_test))
```

```
              precision    recall  f1-score   support

           0       1.00      0.99      0.99       860
           1       0.97      0.99      0.98       286

    accuracy                           0.99      1146
   macro avg       0.98      0.99      0.99      1146
weighted avg       0.99      0.99      0.99      1146
```

STEP #6: LET'S ADD ADDITIONAL FEATURE TF-IDF

- Tf–idf stands for "Term Frequency–Inverse Document Frequency" is a numerical statistic used to reflect how important a word is to a document in a collection or corpus of documents.
- TFIDF is used as a weighting factor during text search processes and text mining.
- The intuition behing the TFIDF is as follows: if a word appears several times in a given document, this word might be meaningful (more important) than other words that appeared fewer times in the same document. However, if a given word appeared several times in a given document but also appeared many times in

other documents, there is a probability that this word might be common frequent word such as 'I' 'am'..etc. (not really important or meaningful!).

- TF: Term Frequency is used to measure the frequency of term occurrence in a document:
  - TF(word) = Number of times the 'word' appears in a document / Total number of terms in the document
- IDF: Inverse Document Frequency is used to measure how important a term is:
  - IDF(word) = log_e(Total number of documents / Number of documents with the term 'word' in it).
- Example: Let's assume we have a document that contains 1000 words and the term “John” appeared 20 times, the Term-Frequency for the word 'John' can be calculated as follows:
  - TF|john = 20/1000 = 0.02
- Let's calculate the IDF (inverse document frequency) of the word 'john' assuming that it appears 50,000 times in a 1,000,000 million documents (corpus).
  - IDF|john = log (1,000,000/50,000) = 1.3
- Therefore the overall weight of the word 'john' is as follows
  - TF-IDF|john = 0.02 * 1.3 = 0.026

In [59]:

```
spamham_countvectorizer
```

Out[59]:

```
<5728x37229 sparse matrix of type '<class 'numpy.int64'>'
	with 565908 stored elements in Compressed Sparse Row format>
```

In [60]:

```
from sklearn.feature_extraction.text import TfidfTransformer

emails_tfidf = TfidfTransformer().fit_transform(spamham_countvectorizer)
print(emails_tfidf.shape)
```

```
(5728, 37229)
```

In [61]:

```
print(emails_tfidf[:,:])
# Sparse matrix with all the values of IF-IDF
```

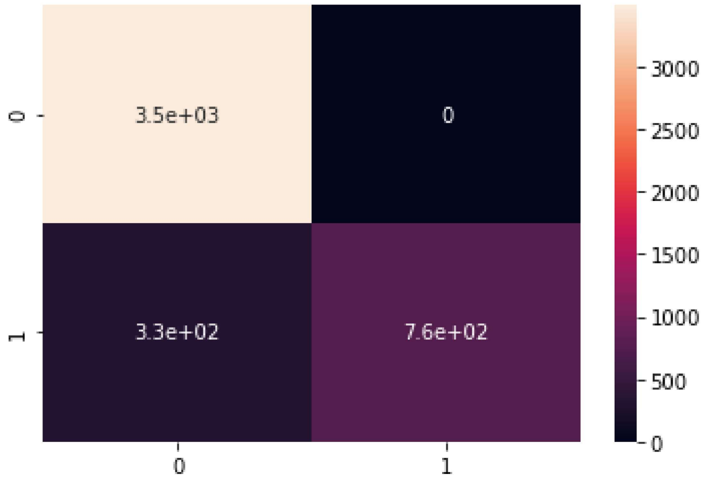
```
  (0, 36565)	0.06908944889543289
  (0, 36432)	0.06757047739651872
  (0, 36430)	0.059679365326344706
  (0, 36025)	0.1319392730989776
  (0, 35034)	0.05233428188145157
  (0, 34800)	0.09384305652743173
  (0, 33562)	0.06921203533637368
  (0, 33037)	0.09490328795519132
  (0, 32843)	0.06073679014431701
  (0, 32617)	0.11152518721878715
  (0, 32602)	0.11962021118089677
  (0, 32319)	0.11962021118089677
  (0, 32263)	0.0789584619498058
  (0, 31968)	0.11850864343422601
  (0, 31959)	0.08499360588016656
  (0, 31547)	0.10454173100334828
  (0, 30218)	0.04607380847274443
  (0, 29858)	0.09333645170409068
  (0, 28879)	0.07691781511072393
```

In [62]:

```
X = emails_tfidf
y = label

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

from sklearn.naive_bayes import MultinomialNB
NB_classifier = MultinomialNB()
NB_classifier.fit(X_train, y_train)

from sklearn.metrics import classification_report, confusion_matrix
y_predict_train = NB_classifier.predict(X_train)
y_predict_train
cm = confusion_matrix(y_train, y_predict_train)
sns.heatmap(cm, annot=True)
```

Out[62]:

```
<AxesSubplot:>
```



In [63]:

```
print(classification_report(y_test, y_predict_test))
```

```
              precision    recall  f1-score   support

           0       0.77      0.76      0.76       867
           1       0.27      0.29      0.28       279

    accuracy                           0.64      1146
   macro avg       0.52      0.52      0.52      1146
weighted avg       0.65      0.64      0.64      1146
```