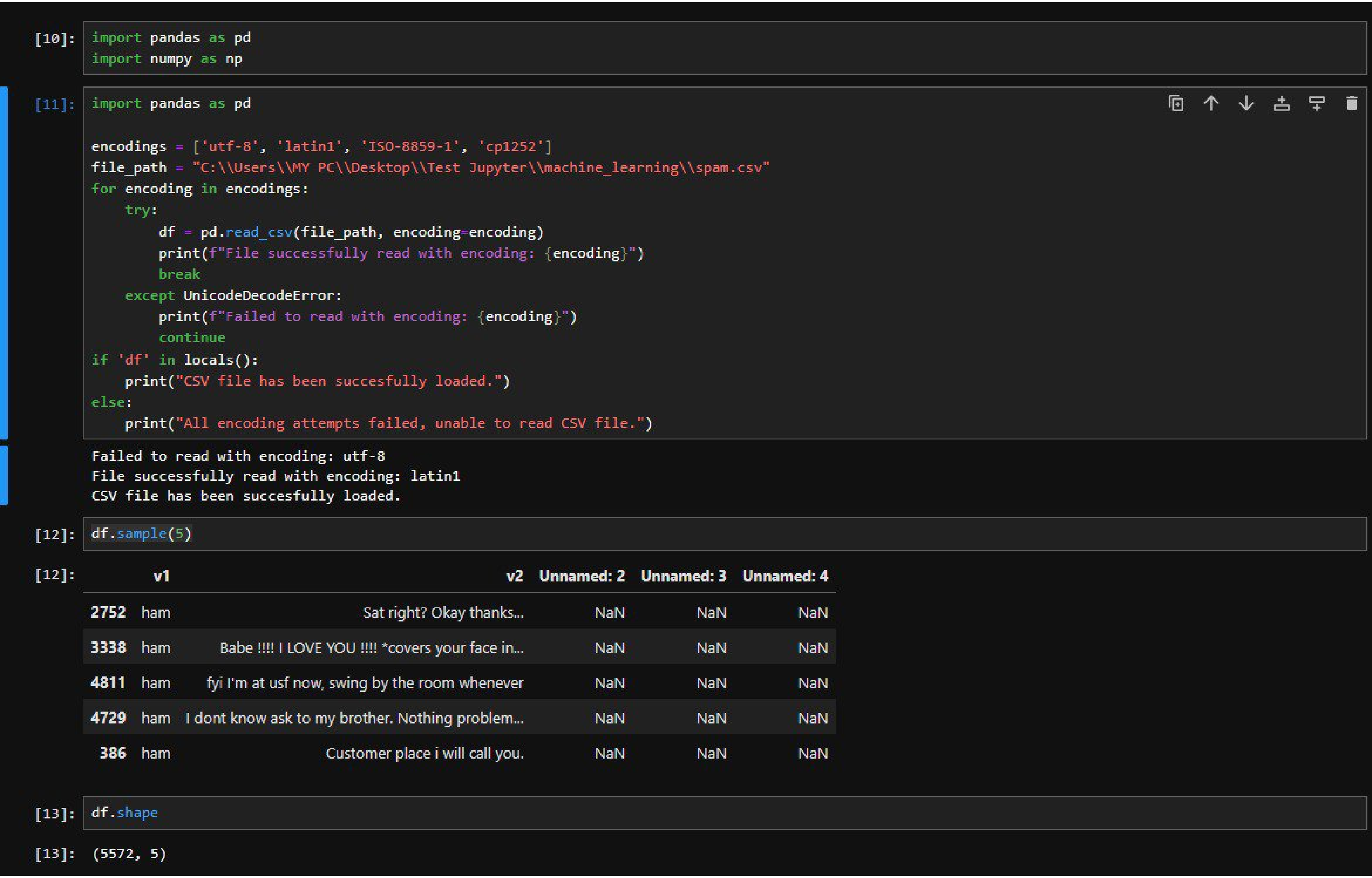


[10]:
```python
import pandas as pd
import numpy as np
```

[11]:
```python
import pandas as pd

encodings = ['utf-8', 'latin1', 'ISO-8859-1', 'cp1252']
file_path = "C:\\Users\\MY PC\\Desktop\\Test Jupyter\\machine_learning\\spam.csv"
for encoding in encodings:
    try:
        df = pd.read_csv(file_path, encoding=encoding)
        print(f"File successfully read with encoding: {encoding}")
        break
    except UnicodeDecodeError:
        print(f"Failed to read with encoding: {encoding}")
        continue
if 'df' in locals():
    print("CSV file has been succesfully loaded.")
else:
    print("All encoding attempts failed, unable to read CSV file.")
```

```
Failed to read with encoding: utf-8
File successfully read with encoding: latin1
CSV file has been succesfully loaded.
```

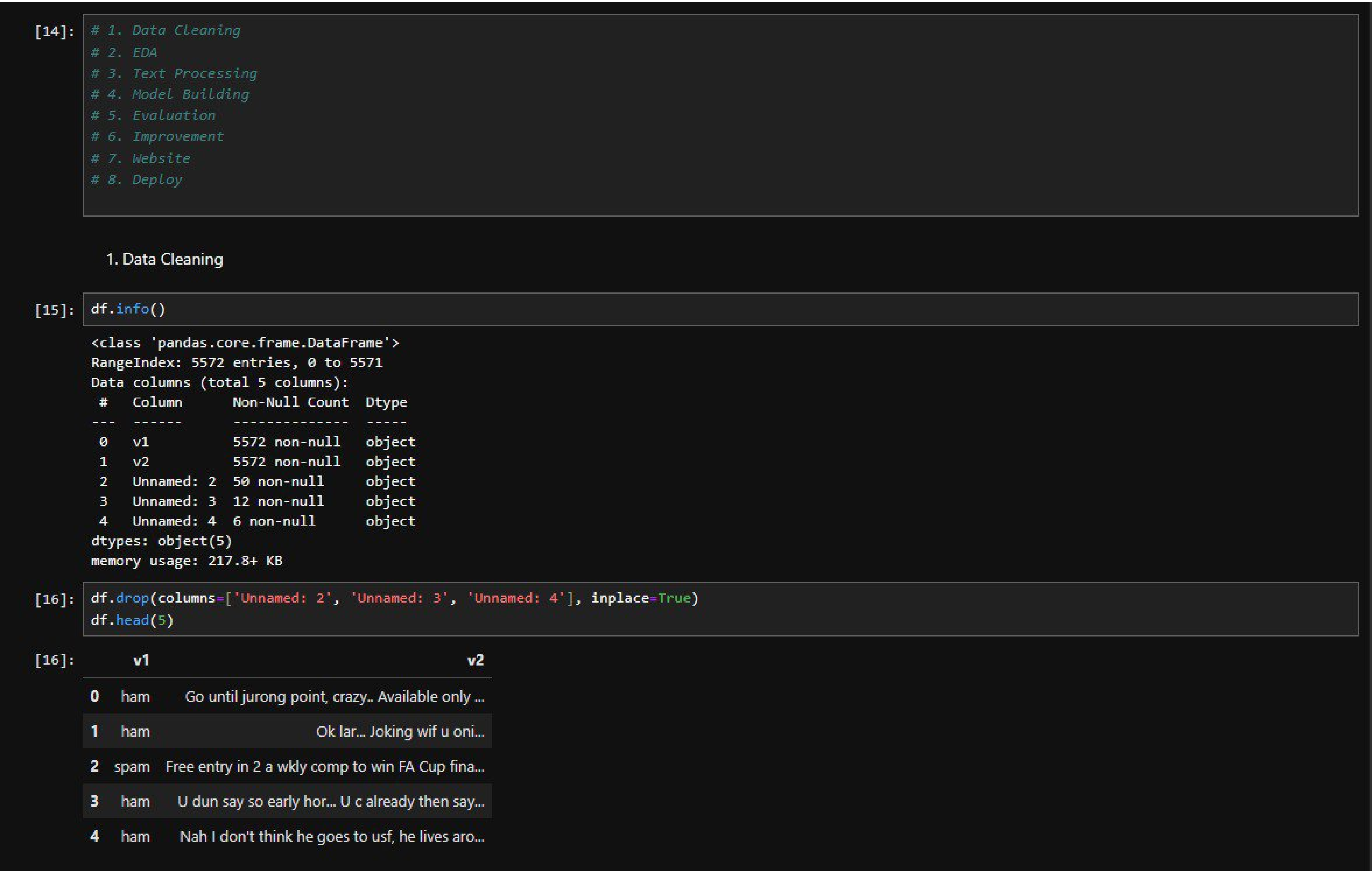
[12]:
```python
df.sample(5)
```

[12]:

	v1	v2	Unnamed: 2	Unnamed: 3	Unnamed: 4
2752	ham	Sat right? Okay thanks...	NaN	NaN	NaN
3338	ham	Babe !!!! I LOVE YOU !!!! *covers your face in...	NaN	NaN	NaN
4811	ham	fyi I'm at usf now, swing by the room whenever	NaN	NaN	NaN
4729	ham	I dont know ask to my brother. Nothing problem...	NaN	NaN	NaN
386	ham	Customer place i will call you.	NaN	NaN	NaN

[13]:
```python
df.shape
```

[13]:
```
(5572, 5)
```



[14]:
```python
# 1. Data Cleaning
# 2. EDA
# 3. Text Processing
# 4. Model Building
# 5. Evaluation
# 6. Improvement
# 7. Website
# 8. Deploy
```

1. Data Cleaning

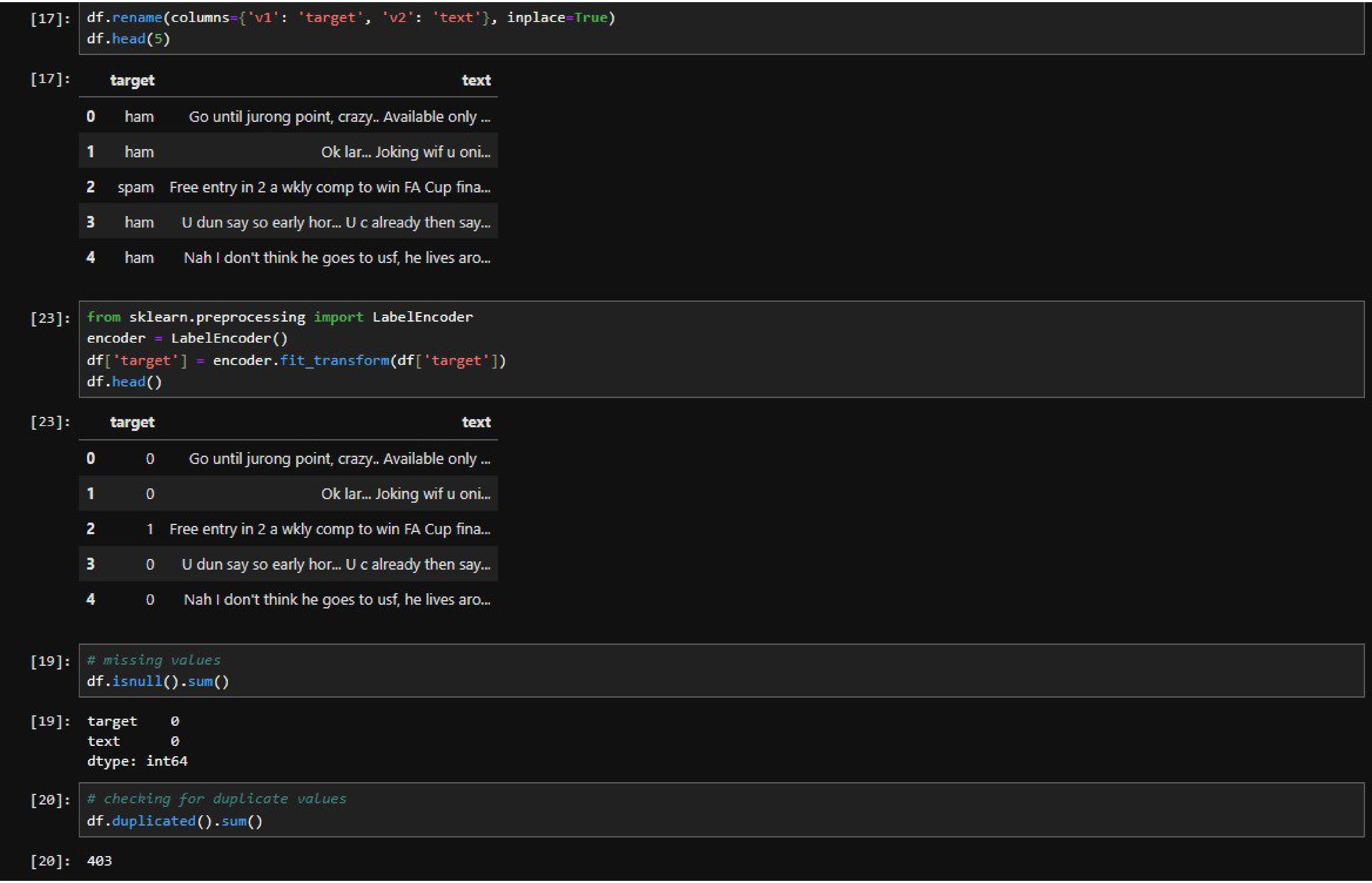
[15]:
```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5572 entries, 0 to 5571
Data columns (total 5 columns):
 #   Column      Non-Null Count  Dtype 
---  ------      --------------  ----- 
 0   v1          5572 non-null   object
 1   v2          5572 non-null   object
 2   Unnamed: 2  50 non-null     object
 3   Unnamed: 3  12 non-null     object
 4   Unnamed: 4  6 non-null      object
dtypes: object(5)
memory usage: 217.8+ KB
```

[16]:
```python
df.drop(columns=['Unnamed: 2', 'Unnamed: 3', 'Unnamed: 4'], inplace=True)
df.head(5)
```

[16]:

	v1	v2
0	ham	Go until jurong point, crazy.. Available only ...
1	ham	Ok lar... Joking wif u oni...
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...
3	ham	U dun say so early hor... U c already then say...
4	ham	Nah I don't think he goes to usf, he lives aro...



```
[17]: df.rename(columns={'v1': 'target', 'v2': 'text'}, inplace=True)
      df.head(5)
```

```
[17]:    target                                              text
      0     ham     Go until jurong point, crazy.. Available only ...
      1     ham                         Ok lar... Joking wif u oni...
      2    spam   Free entry in 2 a wkly comp to win FA Cup fina...
      3     ham   U dun say so early hor... U c already then say...
      4     ham   Nah I don't think he goes to usf, he lives aro...
```

```
[23]: from sklearn.preprocessing import LabelEncoder
      encoder = LabelEncoder()
      df['target'] = encoder.fit_transform(df['target'])
      df.head()
```

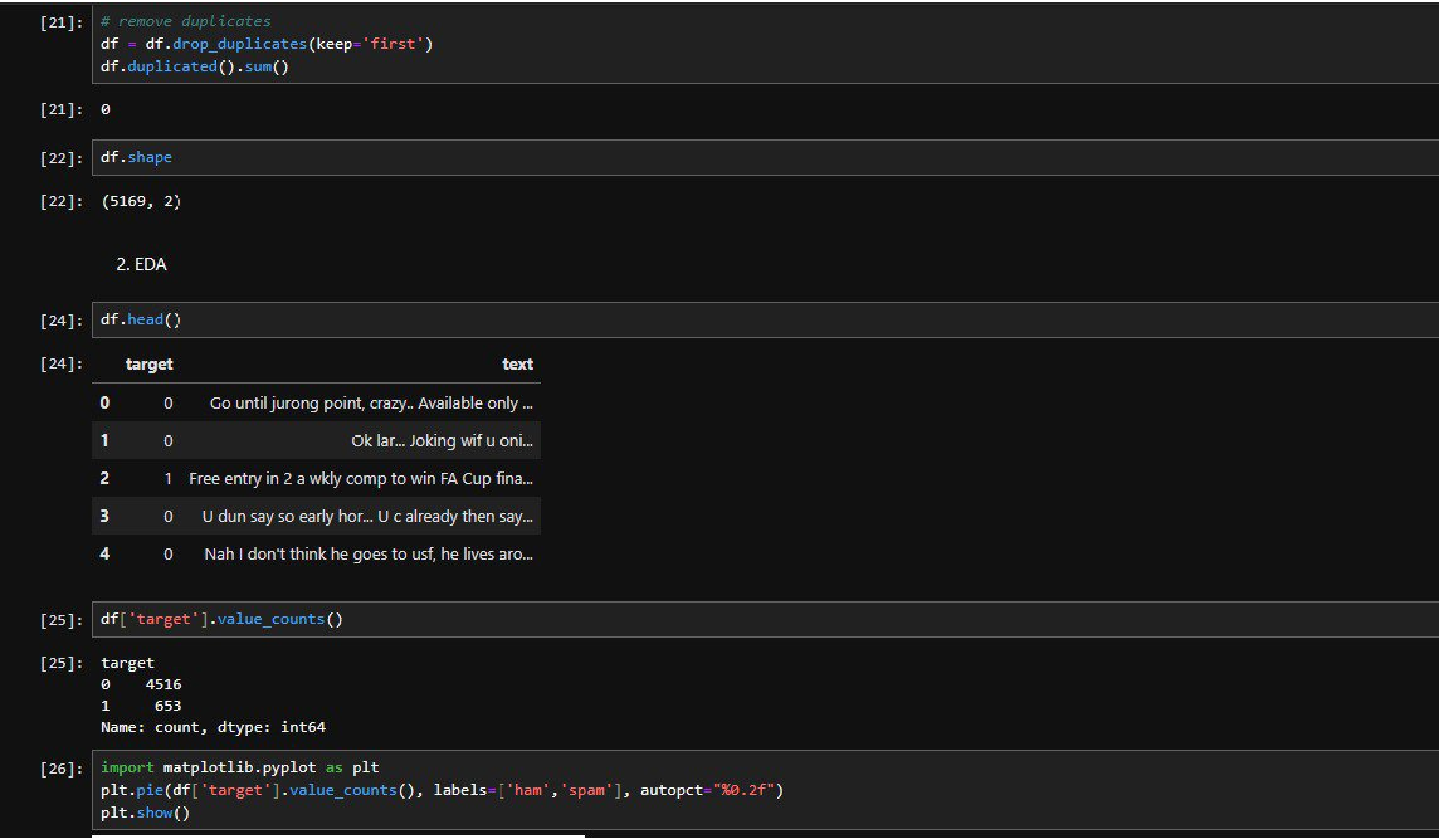
```
[23]:    target                                              text
      0       0     Go until jurong point, crazy.. Available only ...
      1       0                         Ok lar... Joking wif u oni...
      2       1   Free entry in 2 a wkly comp to win FA Cup fina...
      3       0   U dun say so early hor... U c already then say...
      4       0   Nah I don't think he goes to usf, he lives aro...
```

```
[19]: # missing values
      df.isnull().sum()
```

```
[19]: target    0
      text      0
      dtype: int64
```

```
[20]: # checking for duplicate values
      df.duplicated().sum()
```

```
[20]: 403
```



[21]:
```
# remove duplicates
df = df.drop_duplicates(keep='first')
df.duplicated().sum()
```

[21]:
```
0
```

[22]:
```
df.shape
```

[22]:
```
(5169, 2)
```

2. EDA

[24]:
```
df.head()
```

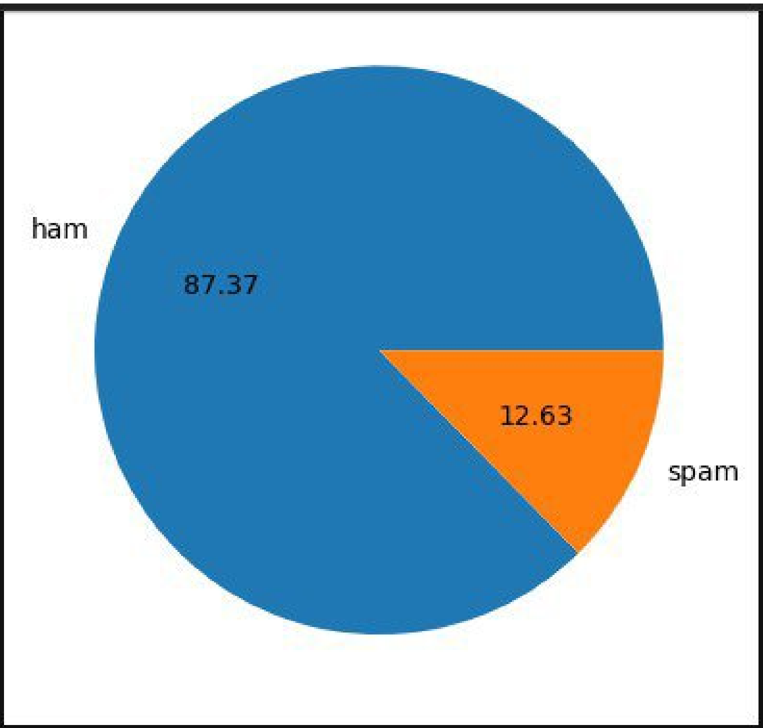
[24]:

	target	text
0	0	Go until jurong point, crazy.. Available only ...
1	0	Ok lar... Joking wif u oni...
2	1	Free entry in 2 a wkly comp to win FA Cup fina...
3	0	U dun say so early hor... U c already then say...
4	0	Nah I don't think he goes to usf, he lives aro...

[25]:
```
df['target'].value_counts()
```

[25]:
```
target
0    4516
1     653
Name: count, dtype: int64
```

[26]:
```
import matplotlib.pyplot as plt
plt.pie(df['target'].value_counts(), labels=['ham','spam'], autopct="%0.2f")
plt.show()
```



```
[29]:  import nltk
       nltk.download('punkt')
```

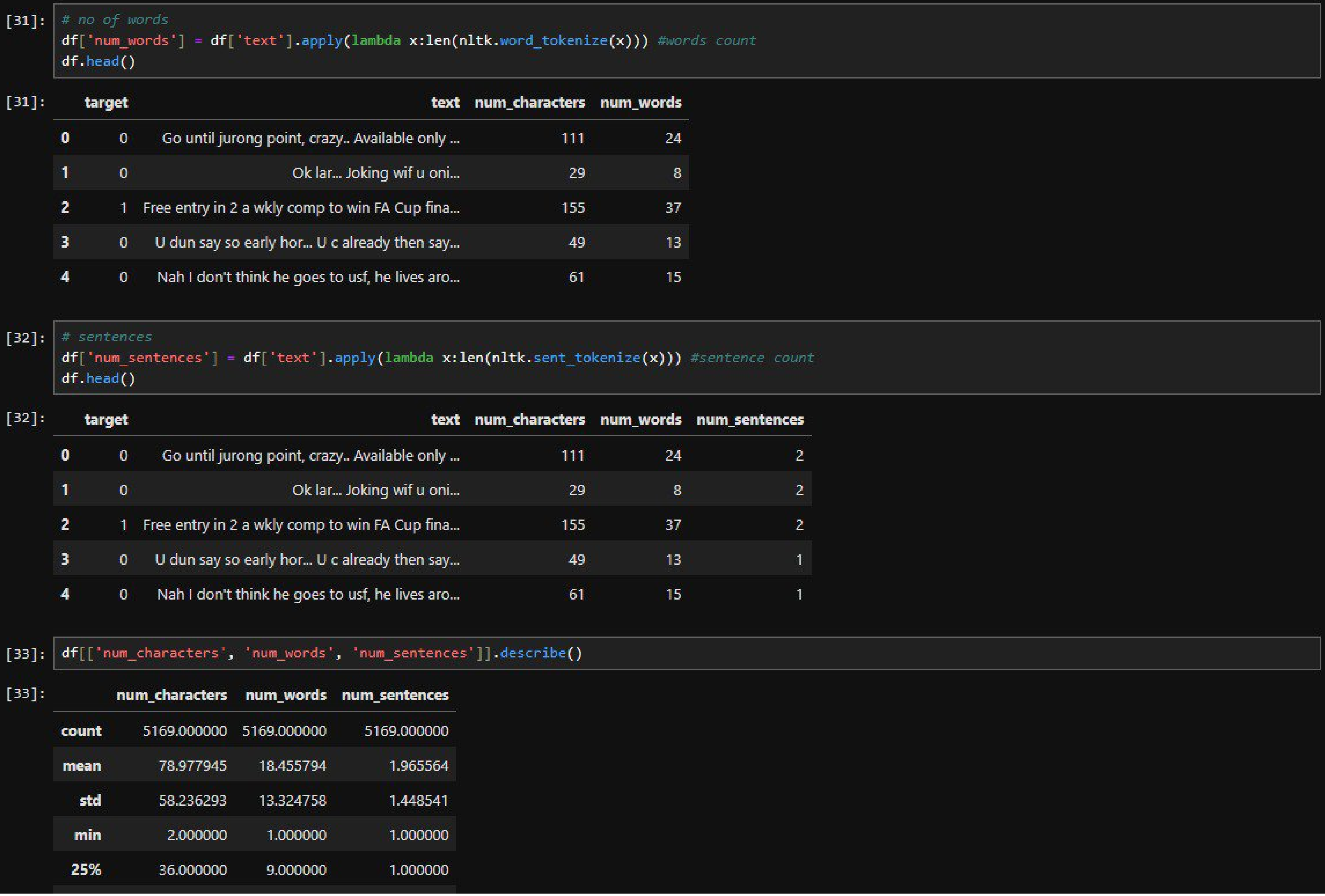
```
[nltk_data] Downloading package punkt to C:\Users\MY
[nltk_data]     PC\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

```
[29]:  True
```

```
[30]:  df['num_characters'] = df['text'].apply(len)  #no of characters
       df.head()
```

[30]:

	target	text	num_characters
0	0	Go until jurong point, crazy.. Available only ...	111
1	0	Ok lar... Joking wif u oni...	29
2	1	Free entry in 2 a wkly comp to win FA Cup fina...	155
3	0	U dun say so early hor... U c already then say...	49
4	0	Nah I don't think he goes to usf, he lives aro...	61



[31]:
```python
# no of words
df['num_words'] = df['text'].apply(lambda x:len(nltk.word_tokenize(x))) #words count
df.head()
```

[31]:

	target	text	num_characters	num_words
0	0	Go until jurong point, crazy.. Available only ...	111	24
1	0	Ok lar... Joking wif u oni...	29	8
2	1	Free entry in 2 a wkly comp to win FA Cup fina...	155	37
3	0	U dun say so early hor... U c already then say...	49	13
4	0	Nah I don't think he goes to usf, he lives aro...	61	15

[32]:
```python
# sentences
df['num_sentences'] = df['text'].apply(lambda x:len(nltk.sent_tokenize(x))) #sentence count
df.head()
```

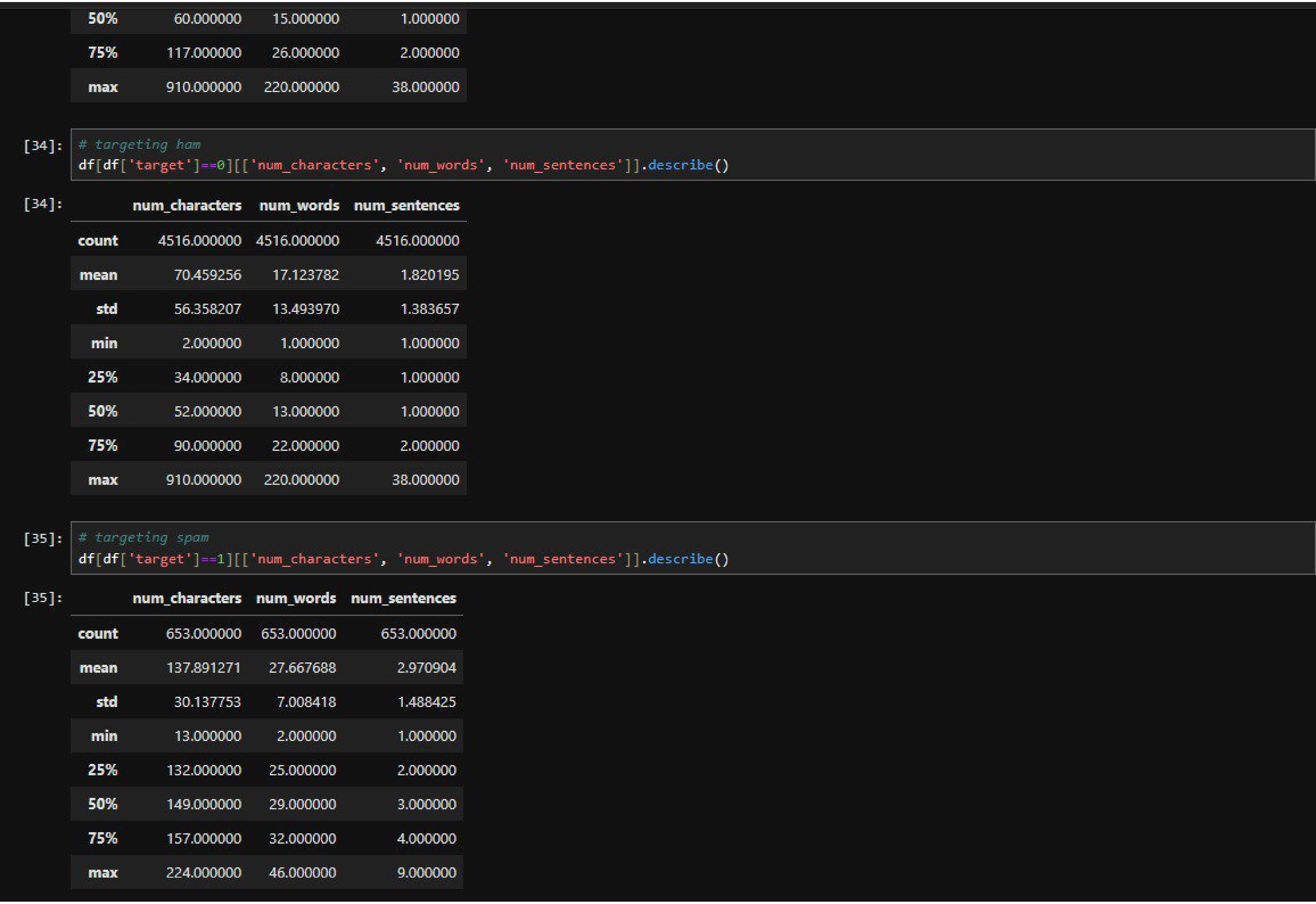
[32]:

	target	text	num_characters	num_words	num_sentences
0	0	Go until jurong point, crazy.. Available only ...	111	24	2
1	0	Ok lar... Joking wif u oni...	29	8	2
2	1	Free entry in 2 a wkly comp to win FA Cup fina...	155	37	2
3	0	U dun say so early hor... U c already then say...	49	13	1
4	0	Nah I don't think he goes to usf, he lives aro...	61	15	1

[33]:
```python
df[['num_characters', 'num_words', 'num_sentences']].describe()
```

[33]:

	num_characters	num_words	num_sentences
count	5169.000000	5169.000000	5169.000000
mean	78.977945	18.455794	1.965564
std	58.236293	13.324758	1.448541
min	2.000000	1.000000	1.000000
25%	36.000000	9.000000	1.000000



50%	60.000000	15.000000	1.000000
75%	117.000000	26.000000	2.000000
max	910.000000	220.000000	38.000000

[34]:
```python
# targeting ham
df[df['target']==0][['num_characters', 'num_words', 'num_sentences']].describe()
```

[34]:

	num_characters	num_words	num_sentences
count	4516.000000	4516.000000	4516.000000
mean	70.459256	17.123782	1.820195
std	56.358207	13.493970	1.383657
min	2.000000	1.000000	1.000000
25%	34.000000	8.000000	1.000000
50%	52.000000	13.000000	1.000000
75%	90.000000	22.000000	2.000000
max	910.000000	220.000000	38.000000

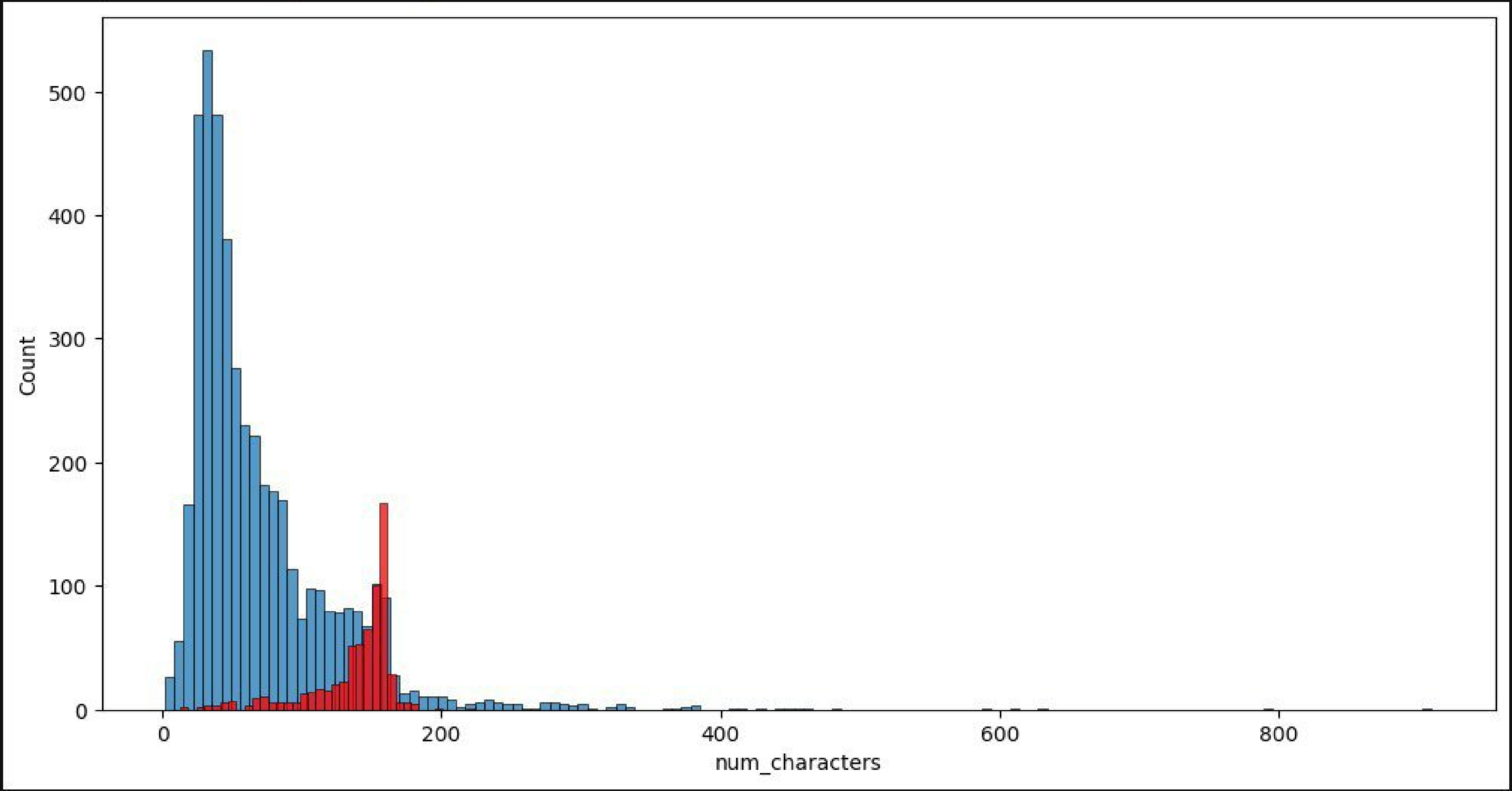
[35]:
```python
# targeting spam
df[df['target']==1][['num_characters', 'num_words', 'num_sentences']].describe()
```

[35]:

	num_characters	num_words	num_sentences
count	653.000000	653.000000	653.000000
mean	137.891271	27.667688	2.970904
std	30.137753	7.008418	1.488425
min	13.000000	2.000000	1.000000
25%	132.000000	25.000000	2.000000
50%	149.000000	29.000000	3.000000
75%	157.000000	32.000000	4.000000
max	224.000000	46.000000	9.000000

```
[36]: import seaborn as sns
      plt.figure(figsize=(12,6))
      sns.histplot(df[df['target']==0]['num_characters'])
      sns.histplot(df[df['target']==1]['num_characters'], color='red')
```

```
[36]: <AxesSubplot: xlabel='num_characters', ylabel='Count'>
```

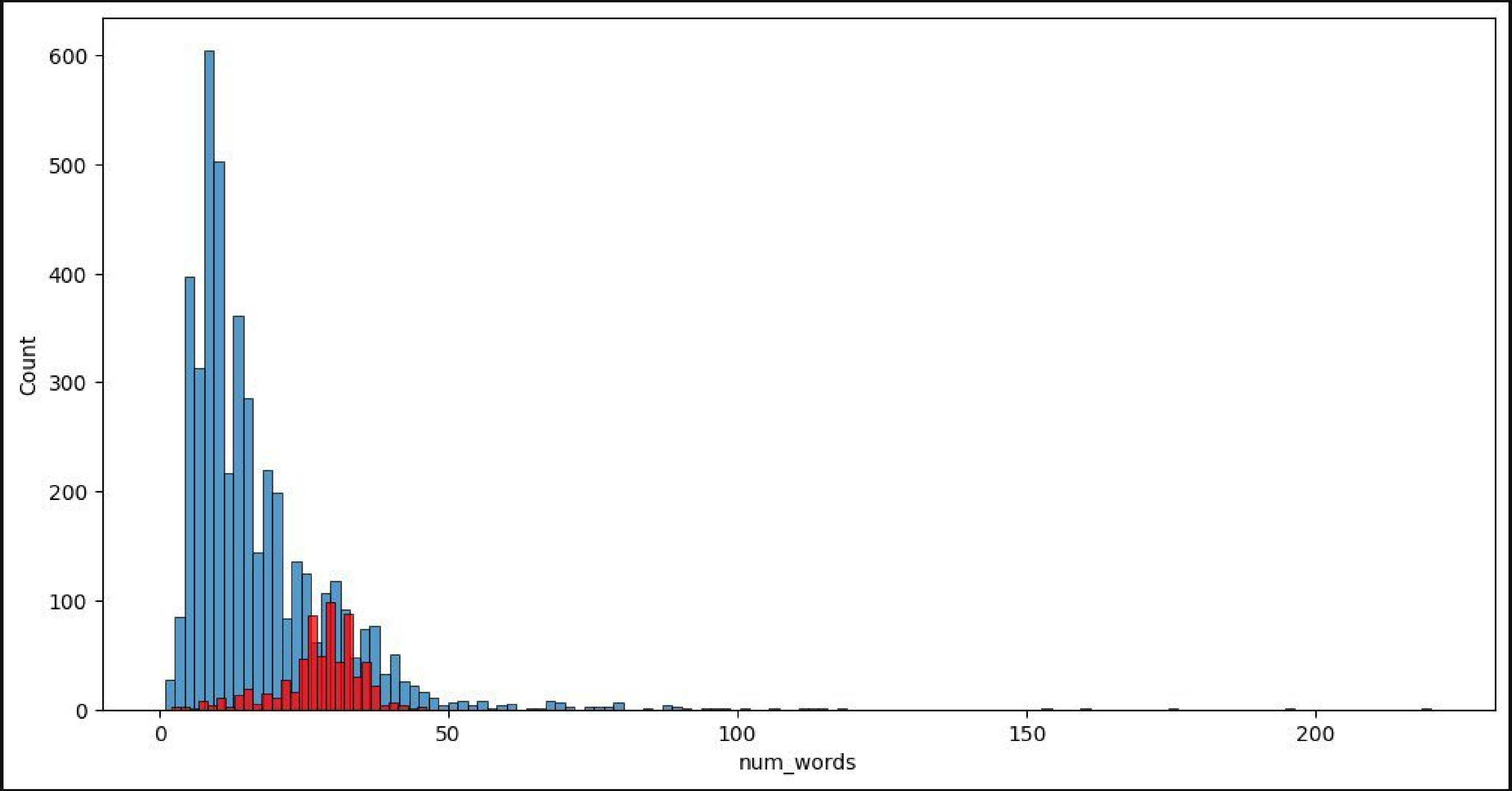


```
[37]: plt.figure(figsize=(12,6))
      sns.histplot(df[df['target']==0]['num_words'])
      sns.histplot(df[df['target']==1]['num_words'], color='red')
```

```
[37]: <AxesSubplot: xlabel='num_words', ylabel='Count'>
```

```
[37]: plt.figure(figsize=(12,6))
      sns.histplot(df[df['target']==0]['num_words'])
      sns.histplot(df[df['target']==1]['num_words'], color='red')
```

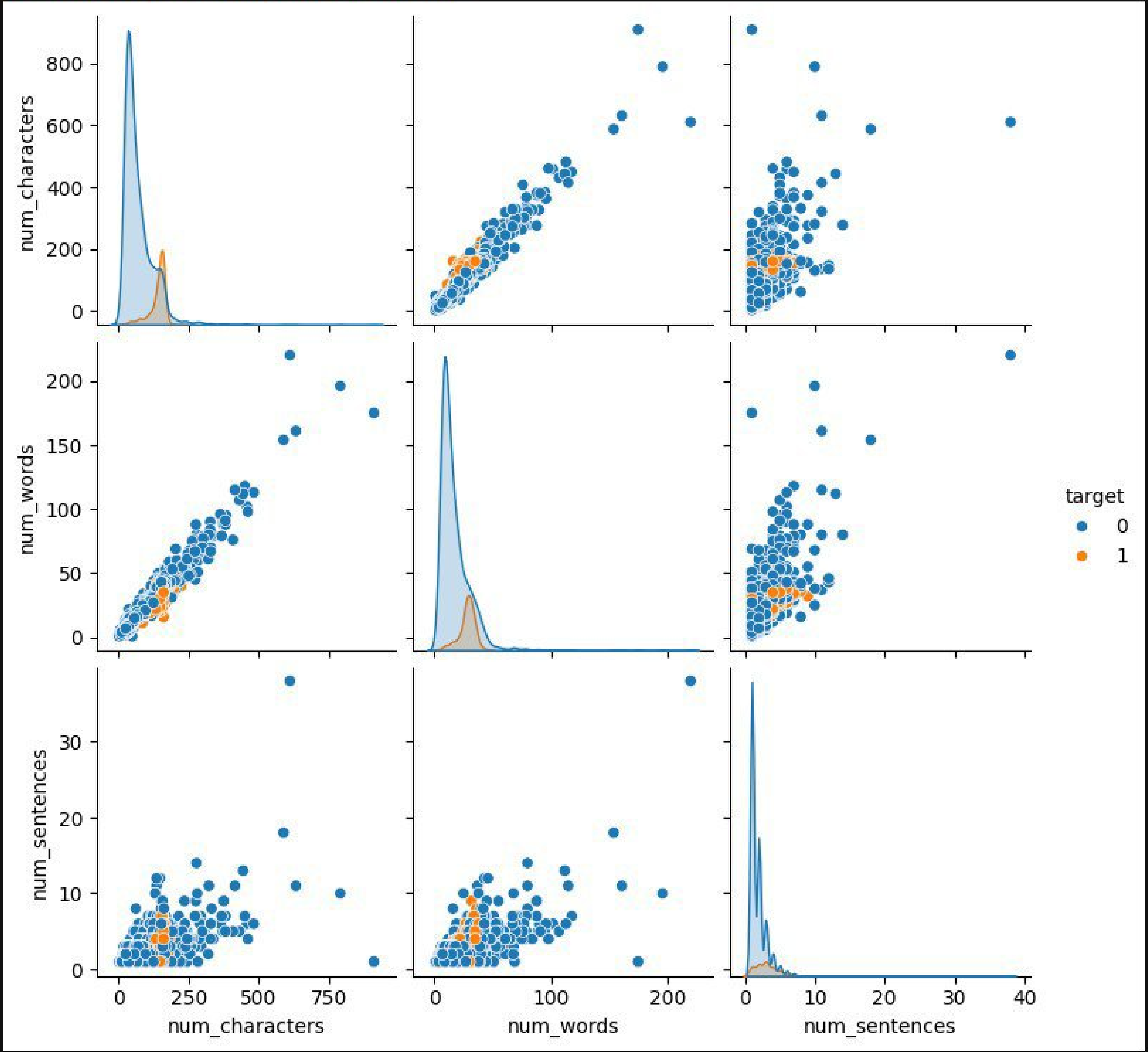
```
[37]: <AxesSubplot: xlabel='num_words', ylabel='Count'>
```



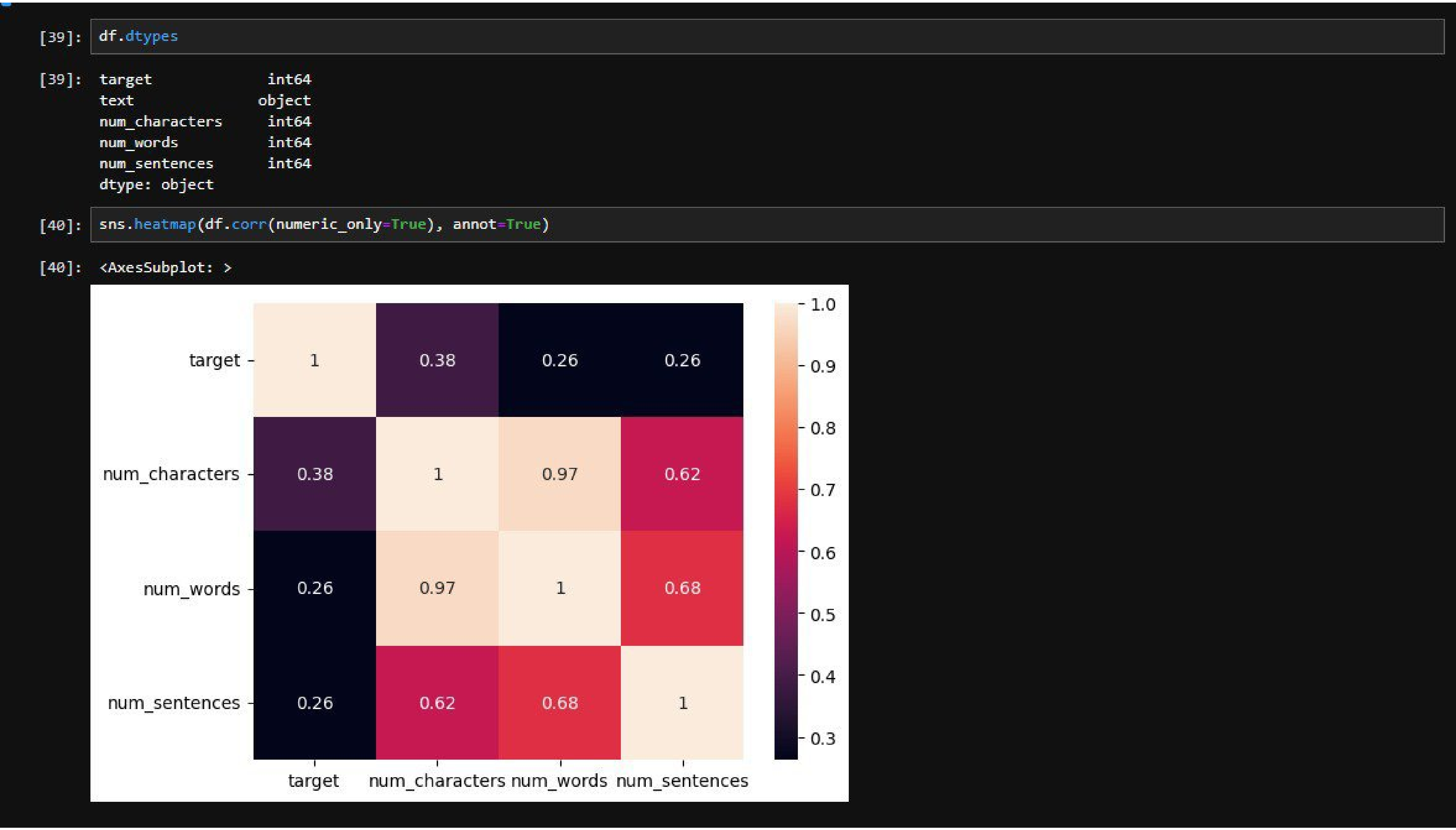
```
[38]: sns.pairplot(df, hue='target')
```

```
[38]: <seaborn.axisgrid.PairGrid at 0x16bb004dae0>
```

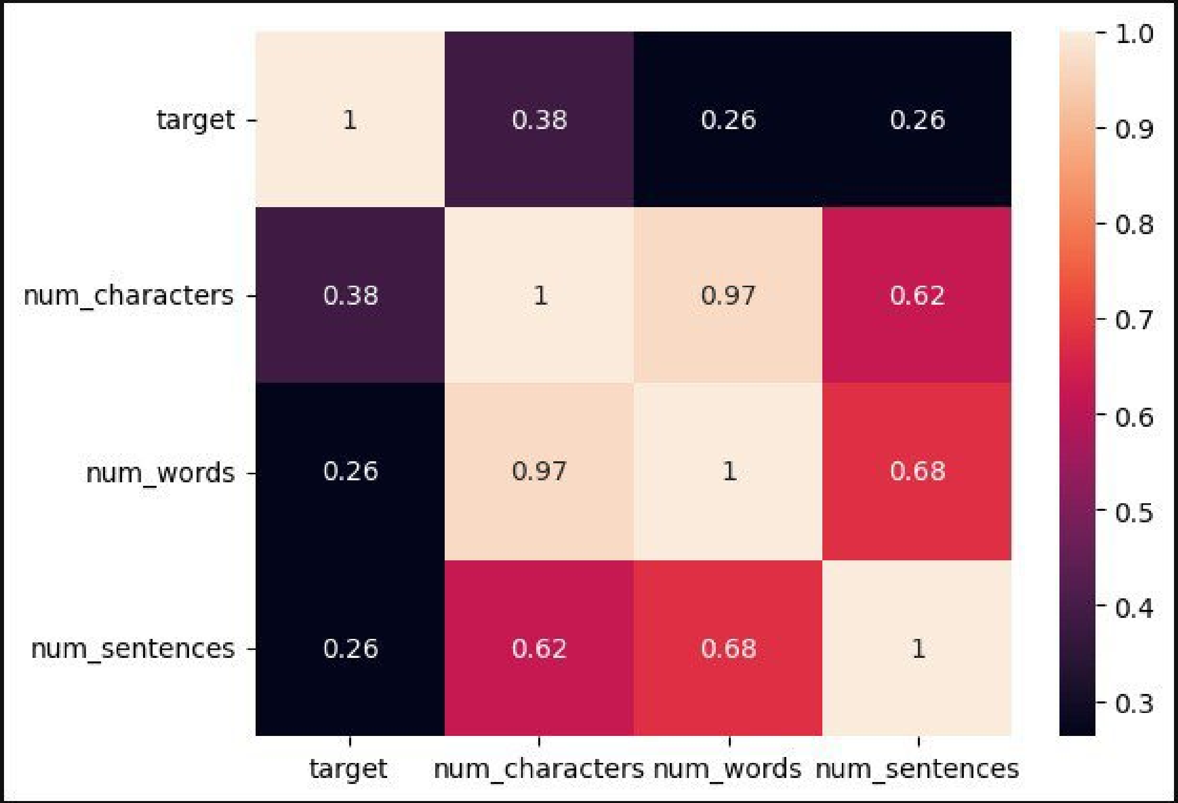
```
[38]:  <seaborn.axisgrid.PairGrid at 0x16bb004dae0>
```



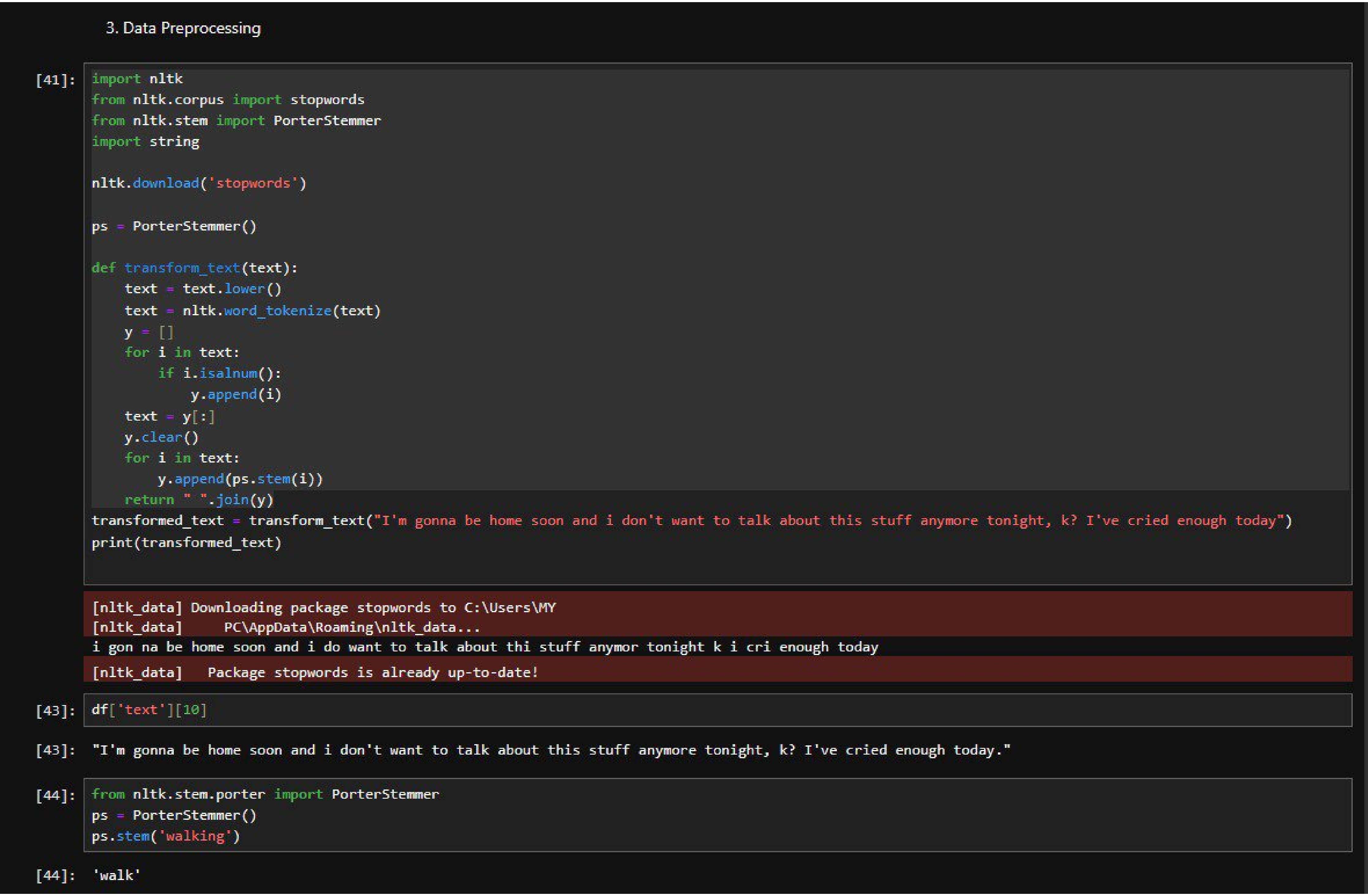
num_characters
num_words
num_sentences
target
0
1
0
250
500
750
0
100
200
0
10
20
30
40
800
600
400
200
0
200
150
100
50
0
30
20
10
0



```
[39]: df.dtypes

[39]: target               int64
      text                object
      num_characters       int64
      num_words            int64
      num_sentences        int64
      dtype: object

[40]: sns.heatmap(df.corr(numeric_only=True), annot=True)

[40]: <AxesSubplot: >
```



	target	num_characters	num_words	num_sentences
target	1	0.38	0.26	0.26
num_characters	0.38	1	0.97	0.62
num_words	0.26	0.97	1	0.68
num_sentences	0.26	0.62	0.68	1



3. Data Preprocessing

[41]:
```python
import nltk
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
import string

nltk.download('stopwords')

ps = PorterStemmer()

def transform_text(text):
    text = text.lower()
    text = nltk.word_tokenize(text)
    y = []
    for i in text:
        if i.isalnum():
            y.append(i)
    text = y[:]
    y.clear()
    for i in text:
        y.append(ps.stem(i))
    return " ".join(y)
transformed_text = transform_text("I'm gonna be home soon and i don't want to talk about this stuff anymore tonight, k? I've cried enough today")
print(transformed_text)
```

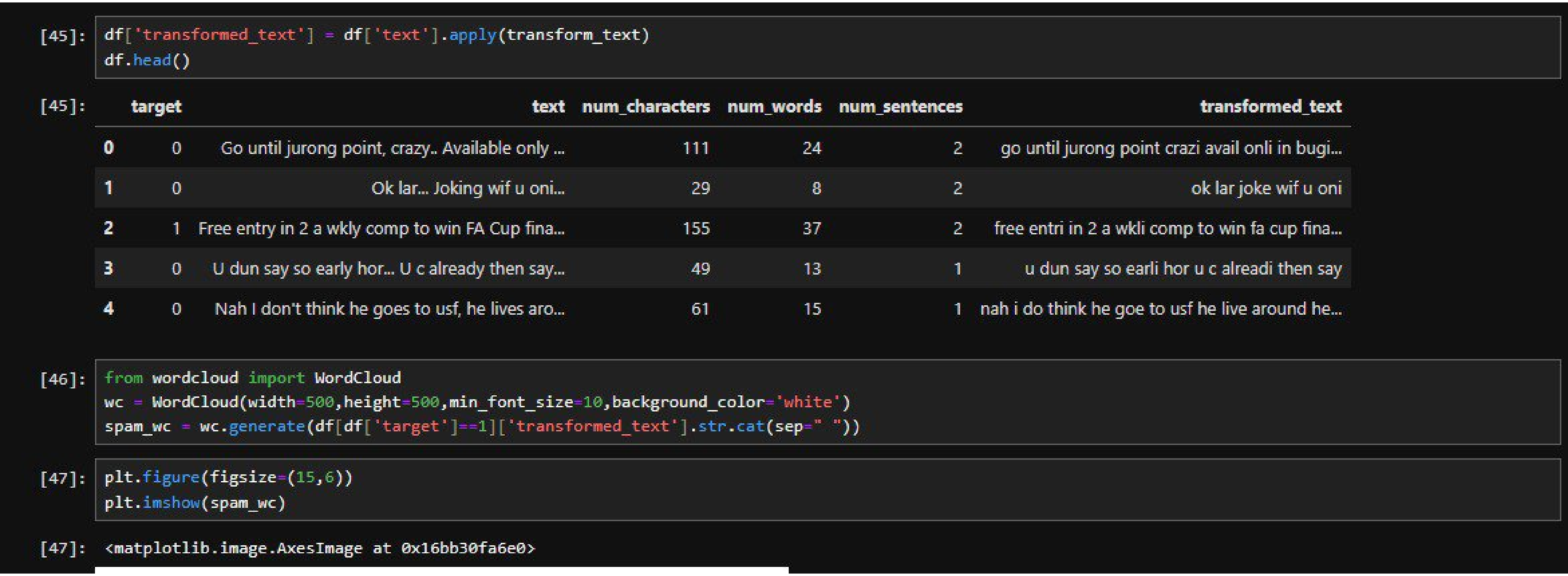
```
[nltk_data] Downloading package stopwords to C:\Users\MY
[nltk_data]     PC\AppData\Roaming\nltk_data...
i gon na be home soon and i do want to talk about thi stuff anymor tonight k i cri enough today
[nltk_data]   Package stopwords is already up-to-date!
```

[43]:
```python
df['text'][10]
```

[43]:
```
"I'm gonna be home soon and i don't want to talk about this stuff anymore tonight, k? I've cried enough today."
```

[44]:
```python
from nltk.stem.porter import PorterStemmer
ps = PorterStemmer()
ps.stem('walking')
```

[44]:
```
'walk'
```



```
[45]: df['transformed_text'] = df['text'].apply(transform_text)
      df.head()
```

[45]:

	target	text	num_characters	num_words	num_sentences	transformed_text
0	0	Go until jurong point, crazy.. Available only ...	111	24	2	go until jurong point crazi avail onli in bugi...
1	0	Ok lar... Joking wif u oni...	29	8	2	ok lar joke wif u oni
2	1	Free entry in 2 a wkly comp to win FA Cup fina...	155	37	2	free entri in 2 a wkli comp to win fa cup fina...
3	0	U dun say so early hor... U c already then say...	49	13	1	u dun say so earli hor u c alreadi then say
4	0	Nah I don't think he goes to usf, he lives aro...	61	15	1	nah i do think he goe to usf he live around he...

```
[46]: from wordcloud import WordCloud
      wc = WordCloud(width=500,height=500,min_font_size=10,background_color='white')
      spam_wc = wc.generate(df[df['target']==1]['transformed_text'].str.cat(sep=" "))
```

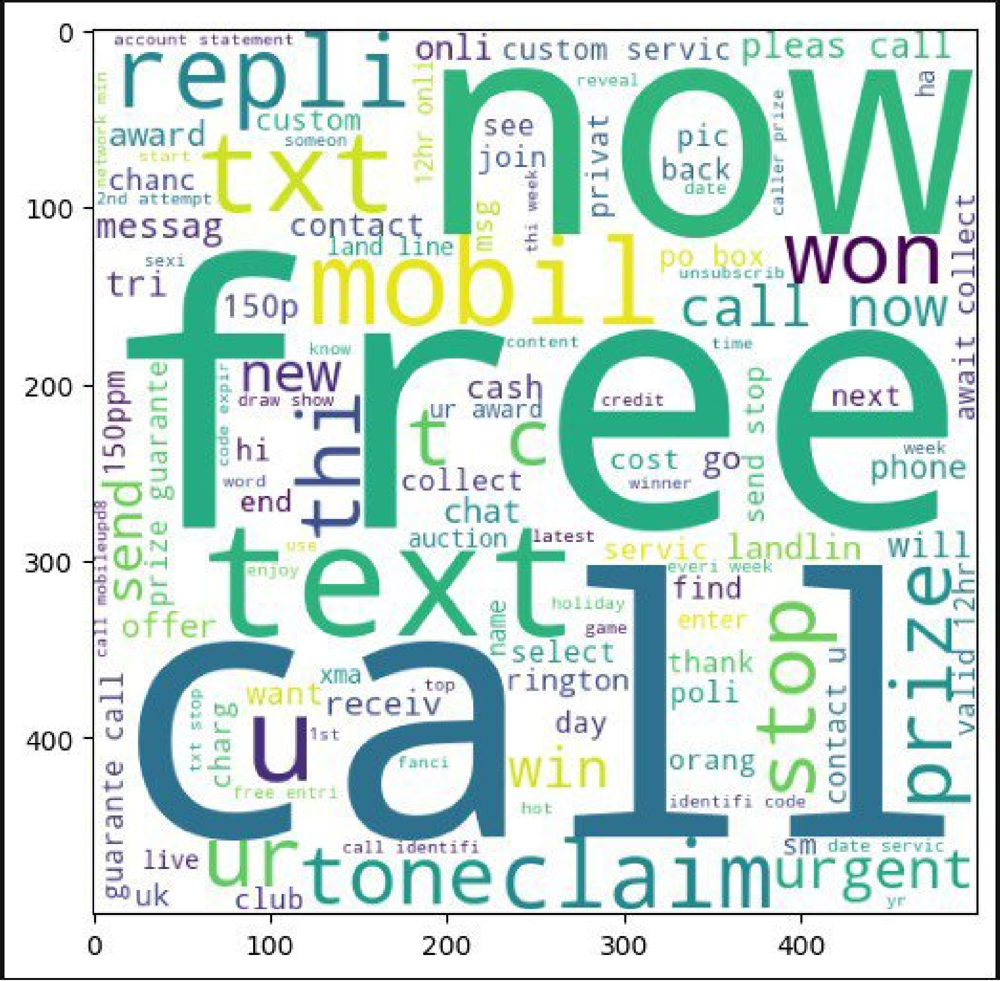
```
[47]: plt.figure(figsize=(15,6))
      plt.imshow(spam_wc)
```

```
[47]: <matplotlib.image.AxesImage at 0x16bb30fa6e0>
```

```
[46]: from wordcloud import WordCloud
      wc = WordCloud(width=500,height=500,min_font_size=10,background_color='white')
      spam_wc = wc.generate(df[df['target']==1]['transformed_text'].str.cat(sep=" "))
```

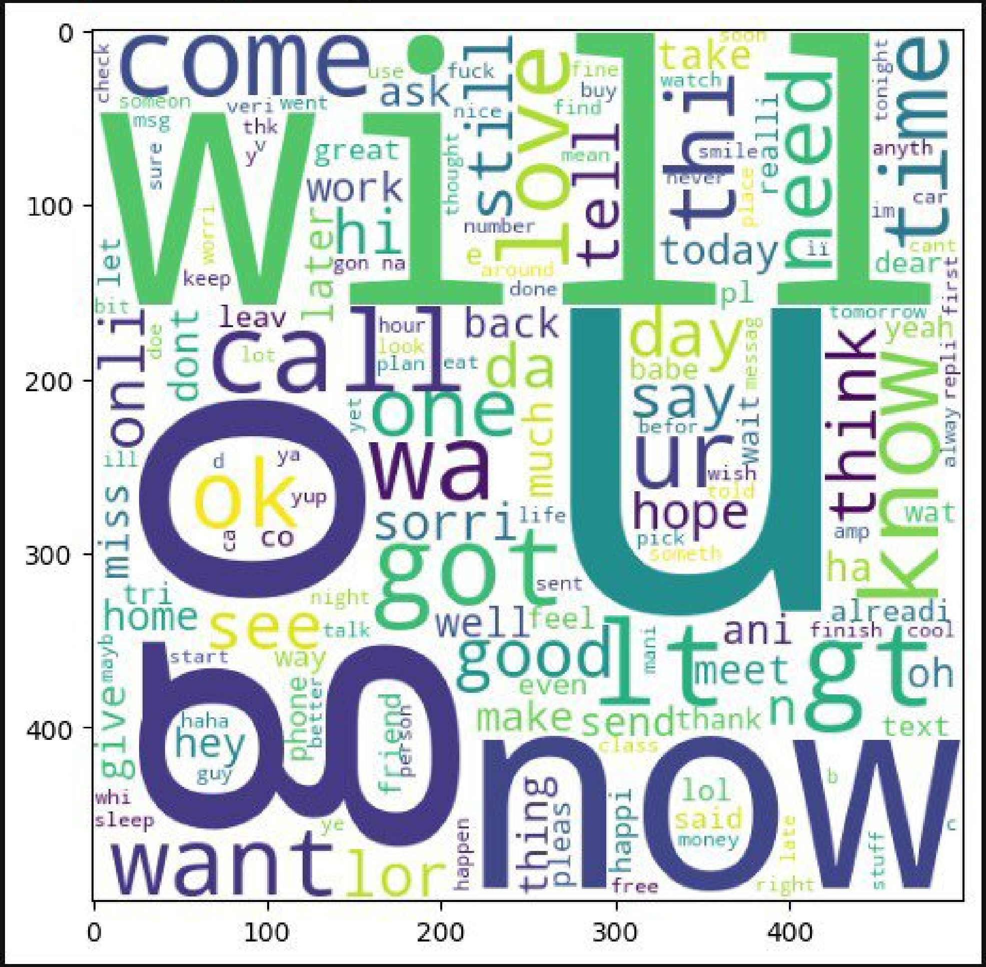
```
[47]: plt.figure(figsize=(15,6))
      plt.imshow(spam_wc)
```

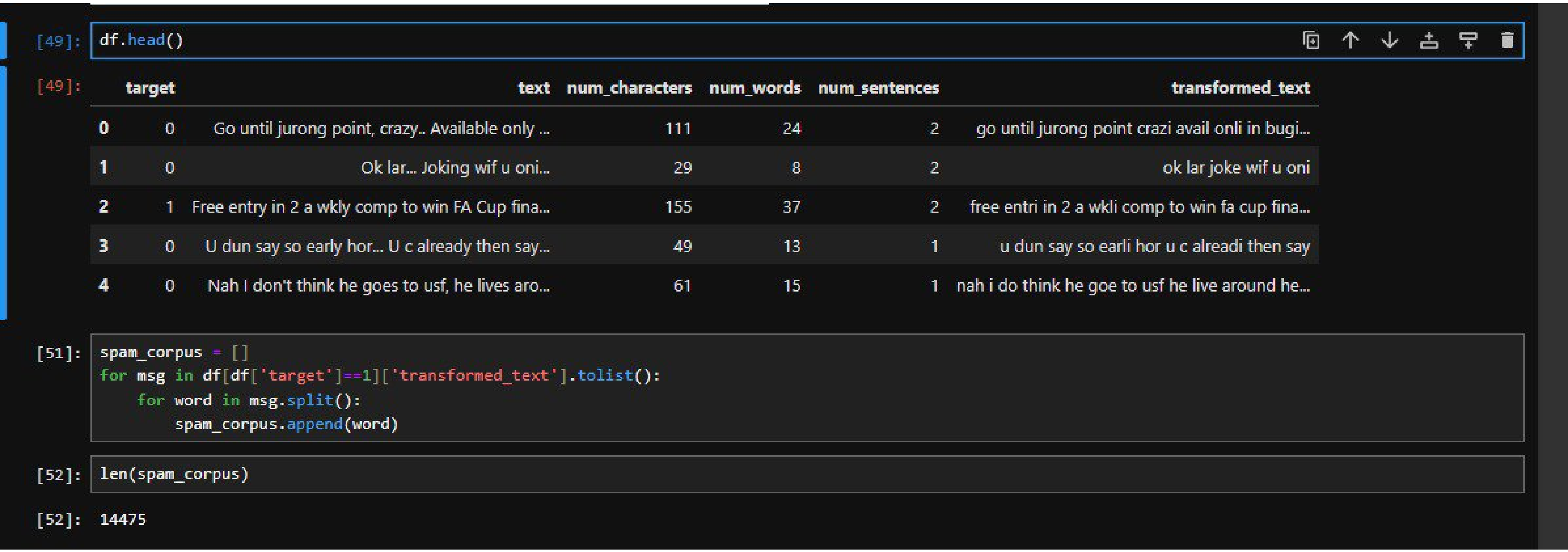
```
[47]: <matplotlib.image.AxesImage at 0x16bb30fa6e0>
```



```
[48]: ham_wc = wc.generate(df[df['target']==0]['transformed_text'].str.cat(sep=" "))
      plt.figure(figsize=(15,6))
      plt.imshow(ham_wc)
```

```
[48]: <matplotlib.image.AxesImage at 0x16bb30a65f0>
```





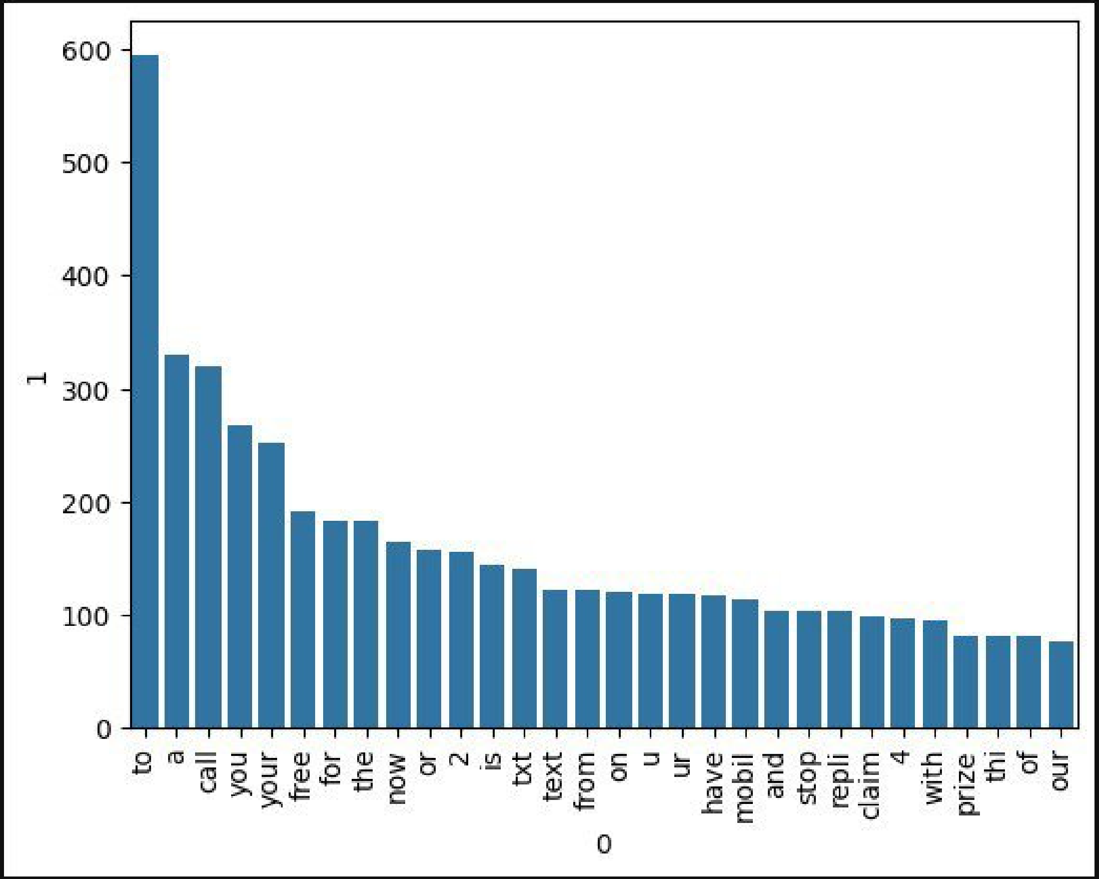
```
[49]: df.head()
```

[49]:

	target	text	num_characters	num_words	num_sentences	transformed_text
0	0	Go until jurong point, crazy.. Available only ...	111	24	2	go until jurong point crazi avail onli in bugi...
1	0	Ok lar... Joking wif u oni...	29	8	2	ok lar joke wif u oni
2	1	Free entry in 2 a wkly comp to win FA Cup fina...	155	37	2	free entri in 2 a wkli comp to win fa cup fina...
3	0	U dun say so early hor... U c already then say...	49	13	1	u dun say so earli hor u c alreadi then say
4	0	Nah I don't think he goes to usf, he lives aro...	61	15	1	nah i do think he goe to usf he live around he...

```
[51]: spam_corpus = []
      for msg in df[df['target']==1]['transformed_text'].tolist():
          for word in msg.split():
              spam_corpus.append(word)

[52]: len(spam_corpus)

[52]: 14475
```

```
[53]: from collections import Counter
      sns.barplot(x=pd.DataFrame(Counter(spam_corpus).most_common(30))[0],y=pd.DataFrame(Counter(spam_corpus).most_common(30))[1])
      plt.xticks(rotation='vertical')
      plt.show()
```

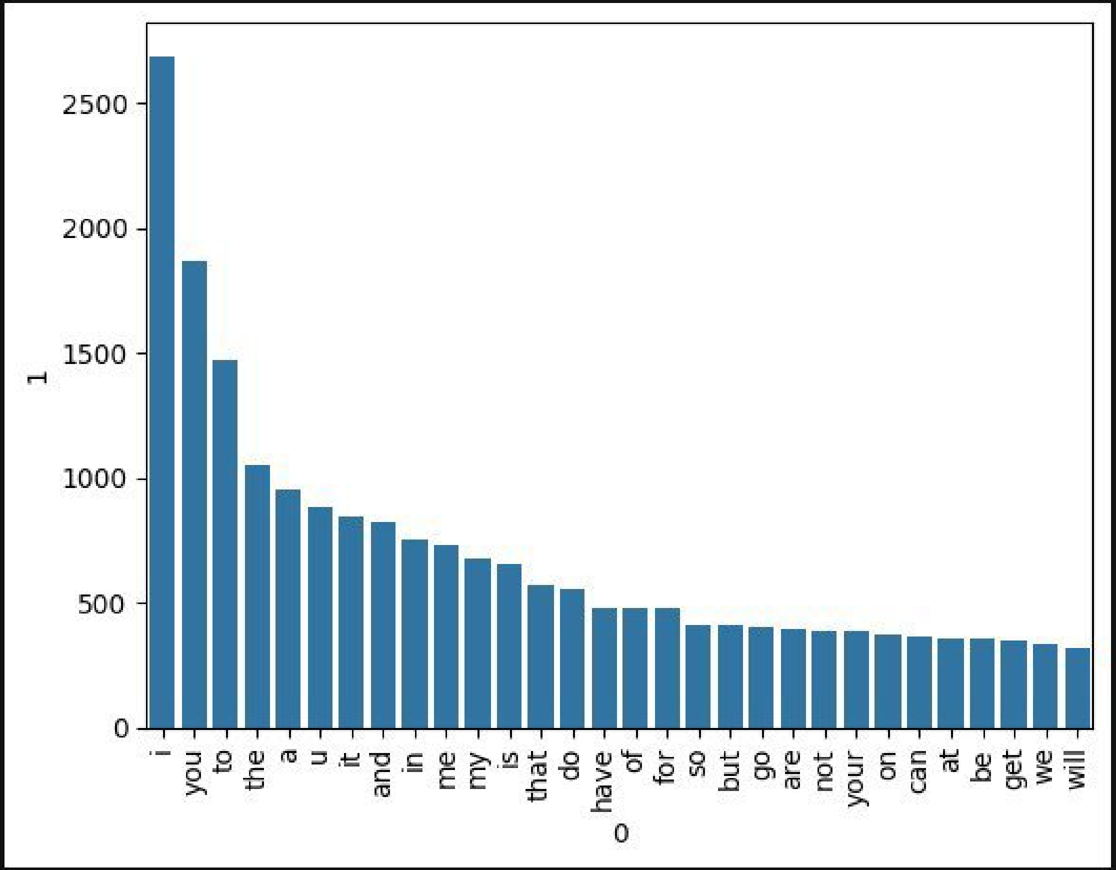


```
[55]: ham_corpus = []
      for msg in df[df['target']==0]['transformed_text'].tolist():
          for word in msg.split():
              ham_corpus.append(word)
```

```
[56]: len(ham_corpus)
```

```
[56]: 62812
```

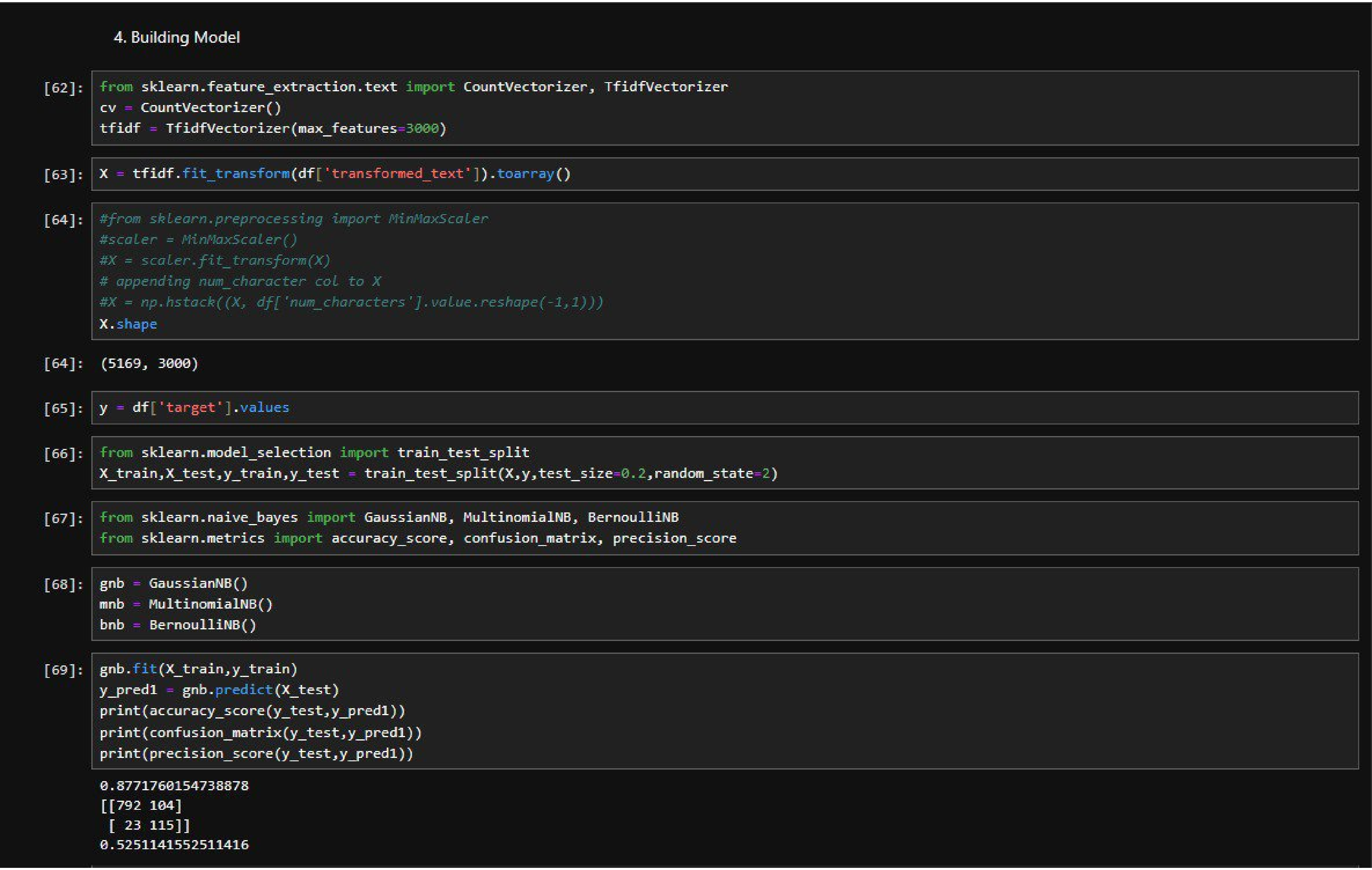
```
[57]: from collections import Counter
      sns.barplot(x=pd.DataFrame(Counter(ham_corpus).most_common(30))[0],y=pd.DataFrame(Counter(ham_corpus).most_common(30))[1])
      plt.xticks(rotation='vertical')
      plt.show()
```



```
[58]: # text vectorization
      df.head()
```

[58]:

	target	text	num_characters	num_words	num_sentences	transformed_text
0	0	Go until jurong point, crazy.. Available only ...	111	24	2	go until jurong point crazi avail onli in bugi...
1	0	Ok lar... Joking wif u oni...	29	8	2	ok lar joke wif u oni
2	1	Free entry in 2 a wkly comp to win FA Cup fina...	155	37	2	free entri in 2 a wkli comp to win fa cup fina...
3	0	U dun say so early hor... U c already then say...	49	13	1	u dun say so earli hor u c alreadi then say
4	0	Nah I don't think he goes to usf, he lives aro...	61	15	1	nah i do think he goe to usf he live around he...



4. Building Model

```
[62]: from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
      cv = CountVectorizer()
      tfidf = TfidfVectorizer(max_features=3000)
```

```
[63]: X = tfidf.fit_transform(df['transformed_text']).toarray()
```

```
[64]: #from sklearn.preprocessing import MinMaxScaler
      #scaler = MinMaxScaler()
      #X = scaler.fit_transform(X)
      # appending num_character col to X
      #X = np.hstack((X, df['num_characters'].value.reshape(-1,1)))
      X.shape
```

```
[64]: (5169, 3000)
```

```
[65]: y = df['target'].values
```

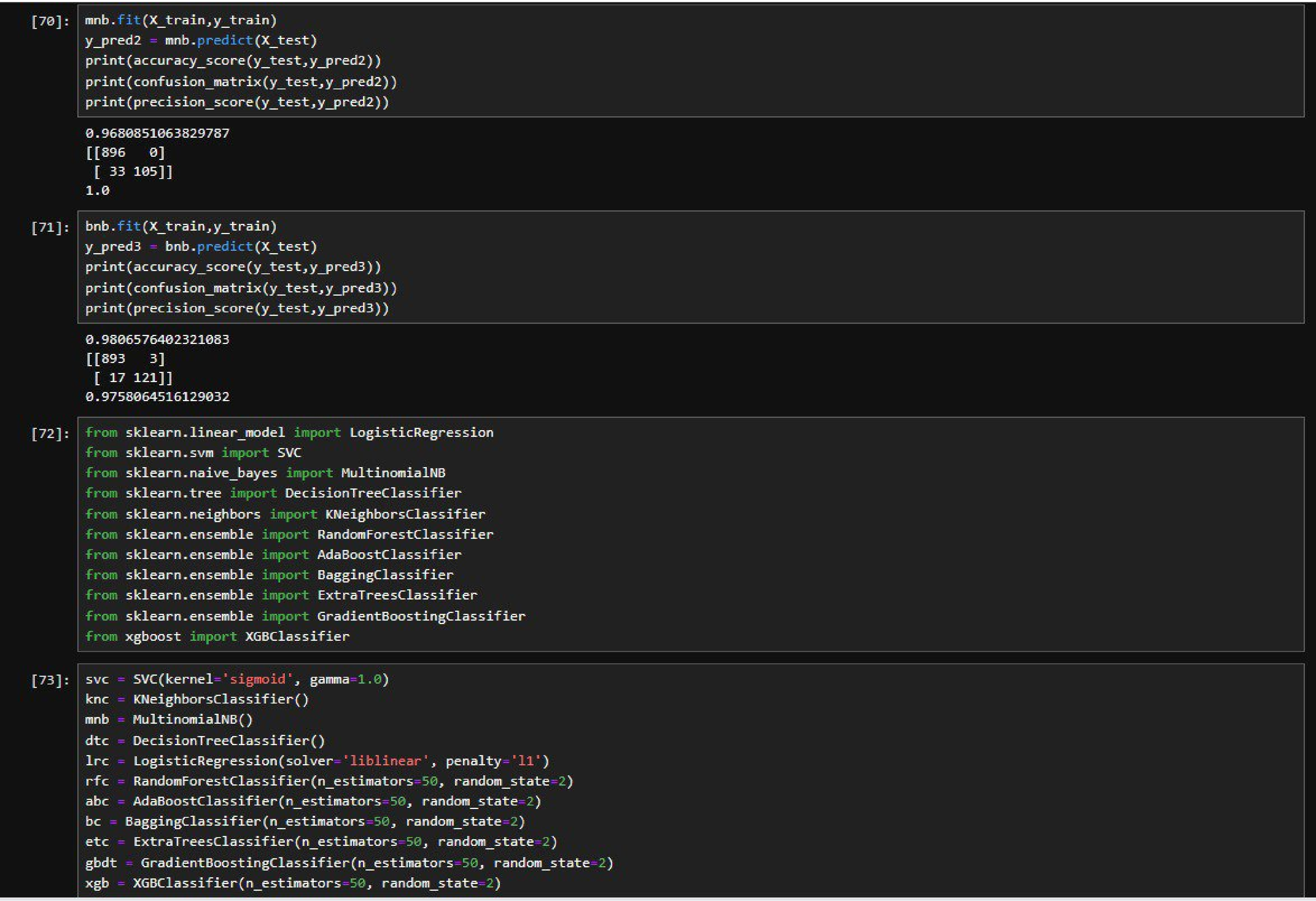
```
[66]: from sklearn.model_selection import train_test_split
      X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2,random_state=2)
```

```
[67]: from sklearn.naive_bayes import GaussianNB, MultinomialNB, BernoulliNB
      from sklearn.metrics import accuracy_score, confusion_matrix, precision_score
```

```
[68]: gnb = GaussianNB()
      mnb = MultinomialNB()
      bnb = BernoulliNB()
```

```
[69]: gnb.fit(X_train,y_train)
      y_pred1 = gnb.predict(X_test)
      print(accuracy_score(y_test,y_pred1))
      print(confusion_matrix(y_test,y_pred1))
      print(precision_score(y_test,y_pred1))
```

```
0.8771760154738878
[[792 104]
 [ 23 115]]
0.5251141552511416
```



```
[70]: mnb.fit(X_train,y_train)
      y_pred2 = mnb.predict(X_test)
      print(accuracy_score(y_test,y_pred2))
      print(confusion_matrix(y_test,y_pred2))
      print(precision_score(y_test,y_pred2))
```

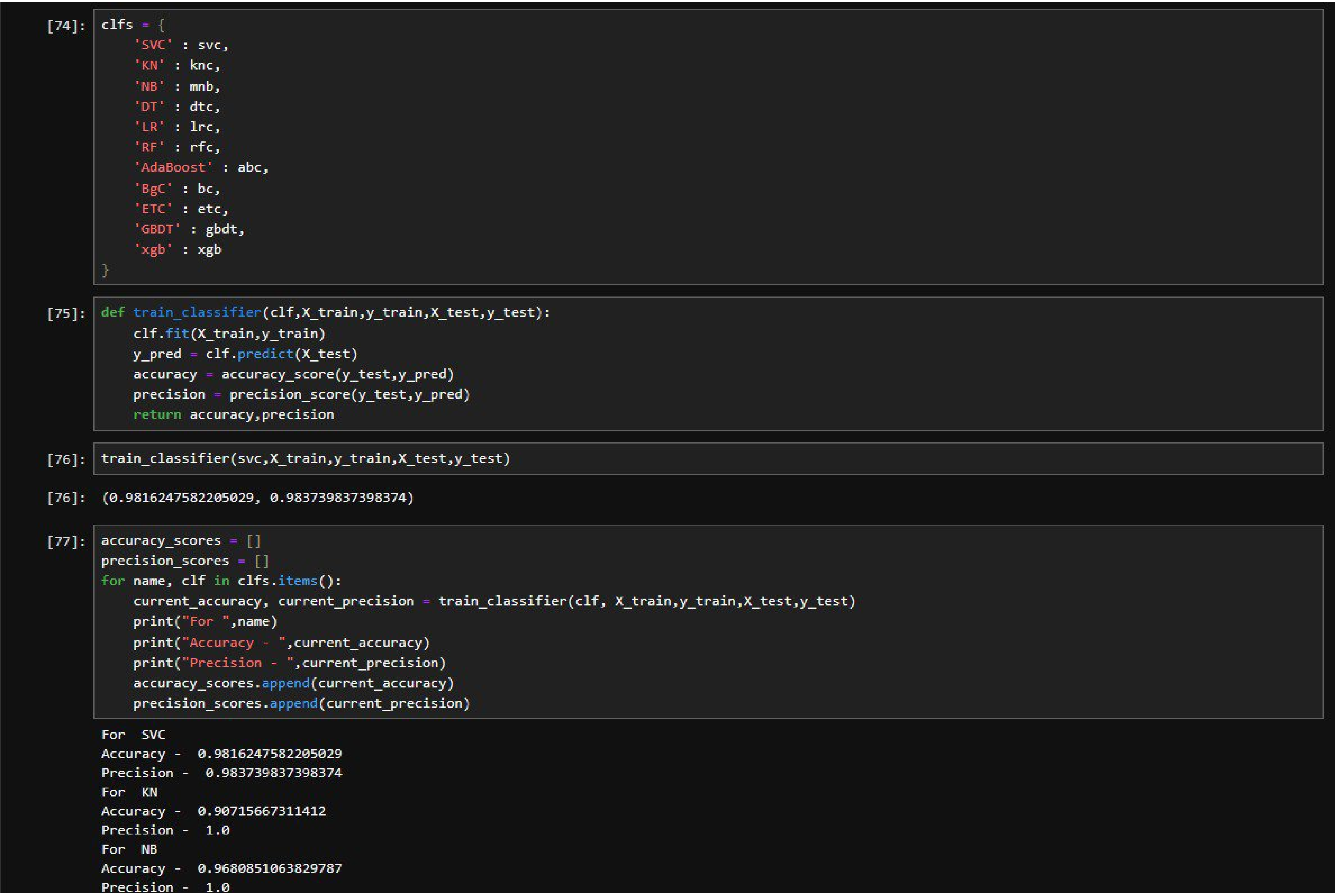
```
0.9680851063829787
[[896   0]
 [ 33 105]]
1.0
```

```
[71]: bnb.fit(X_train,y_train)
      y_pred3 = bnb.predict(X_test)
      print(accuracy_score(y_test,y_pred3))
      print(confusion_matrix(y_test,y_pred3))
      print(precision_score(y_test,y_pred3))
```

```
0.9806576402321083
[[893   3]
 [ 17 121]]
0.9758064516129032
```

```
[72]: from sklearn.linear_model import LogisticRegression
      from sklearn.svm import SVC
      from sklearn.naive_bayes import MultinomialNB
      from sklearn.tree import DecisionTreeClassifier
      from sklearn.neighbors import KNeighborsClassifier
      from sklearn.ensemble import RandomForestClassifier
      from sklearn.ensemble import AdaBoostClassifier
      from sklearn.ensemble import BaggingClassifier
      from sklearn.ensemble import ExtraTreesClassifier
      from sklearn.ensemble import GradientBoostingClassifier
      from xgboost import XGBClassifier
```

```
[73]: svc = SVC(kernel='sigmoid', gamma=1.0)
      knc = KNeighborsClassifier()
      mnb = MultinomialNB()
      dtc = DecisionTreeClassifier()
      lrc = LogisticRegression(solver='liblinear', penalty='l1')
      rfc = RandomForestClassifier(n_estimators=50, random_state=2)
      abc = AdaBoostClassifier(n_estimators=50, random_state=2)
      bc = BaggingClassifier(n_estimators=50, random_state=2)
      etc = ExtraTreesClassifier(n_estimators=50, random_state=2)
      gbdt = GradientBoostingClassifier(n_estimators=50, random_state=2)
      xgb = XGBClassifier(n_estimators=50, random_state=2)
```



```
[74]: clfs = {
          'SVC' : svc,
          'KN' : knc,
          'NB' : mnb,
          'DT' : dtc,
          'LR' : lrc,
          'RF' : rfc,
          'AdaBoost' : abc,
          'BgC' : bc,
          'ETC' : etc,
          'GBDT' : gbdt,
          'xgb' : xgb
      }
```

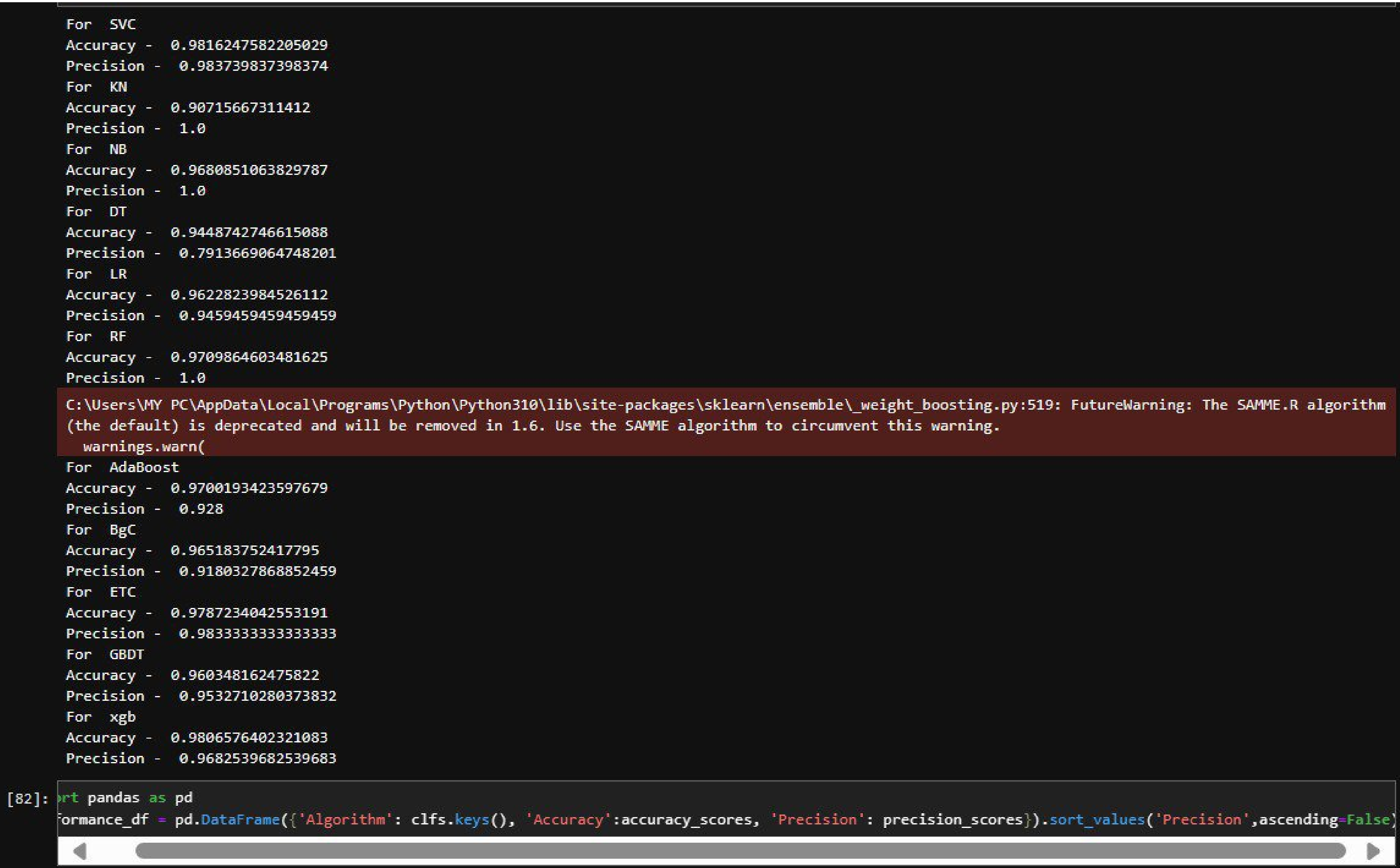
```
[75]: def train_classifier(clf,X_train,y_train,X_test,y_test):
          clf.fit(X_train,y_train)
          y_pred = clf.predict(X_test)
          accuracy = accuracy_score(y_test,y_pred)
          precision = precision_score(y_test,y_pred)
          return accuracy,precision
```

```
[76]: train_classifier(svc,X_train,y_train,X_test,y_test)
```

```
[76]: (0.9816247582205029, 0.983739837398374)
```

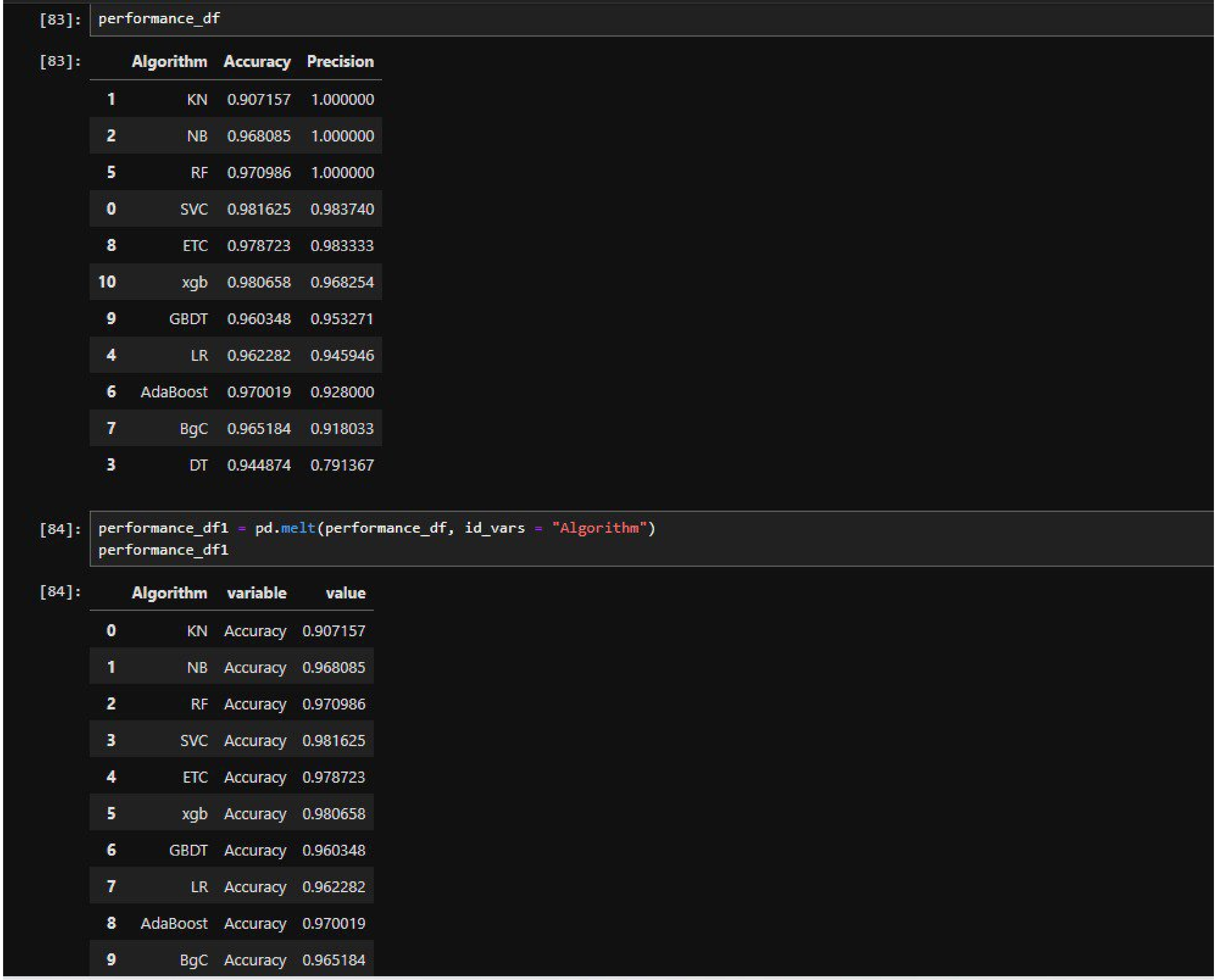
```
[77]: accuracy_scores = []
      precision_scores = []
      for name, clf in clfs.items():
          current_accuracy, current_precision = train_classifier(clf, X_train,y_train,X_test,y_test)
          print("For ",name)
          print("Accuracy - ",current_accuracy)
          print("Precision - ",current_precision)
          accuracy_scores.append(current_accuracy)
          precision_scores.append(current_precision)
```

```
For  SVC
Accuracy -  0.9816247582205029
Precision -  0.983739837398374
For  KN
Accuracy -  0.90715667311412
Precision -  1.0
For  NB
Accuracy -  0.9680851063829787
Precision -  1.0
```



```
For  SVC
Accuracy -  0.9816247582205029
Precision -  0.983739837398374
For  KN
Accuracy -  0.90715667311412
Precision -  1.0
For  NB
Accuracy -  0.9680851063829787
Precision -  1.0
For  DT
Accuracy -  0.9448742746615088
Precision -  0.7913669064748201
For  LR
Accuracy -  0.9622823984526112
Precision -  0.9459459459459459
For  RF
Accuracy -  0.9709864603481625
Precision -  1.0
C:\Users\MY PC\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\ensemble\_weight_boosting.py:519: FutureWarning: The SAMME.R algorithm
(the default) is deprecated and will be removed in 1.6. Use the SAMME algorithm to circumvent this warning.
  warnings.warn(
For  AdaBoost
Accuracy -  0.9700193423597679
Precision -  0.928
For  BgC
Accuracy -  0.965183752417795
Precision -  0.9180327868852459
For  ETC
Accuracy -  0.9787234042553191
Precision -  0.9833333333333333
For  GBDT
Accuracy -  0.960348162475822
Precision -  0.9532710280373832
For  xgb
Accuracy -  0.9806576402321083
Precision -  0.9682539682539683
```

```
[82]: ›rt pandas as pd
formance_df = pd.DataFrame({'Algorithm': clfs.keys(), 'Accuracy':accuracy_scores, 'Precision': precision_scores}).sort_values('Precision',ascending=False)
```



```
[83]: performance_df
```

[83]:

	Algorithm	Accuracy	Precision
1	KN	0.907157	1.000000
2	NB	0.968085	1.000000
5	RF	0.970986	1.000000
0	SVC	0.981625	0.983740
8	ETC	0.978723	0.983333
10	xgb	0.980658	0.968254
9	GBDT	0.960348	0.953271
4	LR	0.962282	0.945946
6	AdaBoost	0.970019	0.928000
7	BgC	0.965184	0.918033
3	DT	0.944874	0.791367

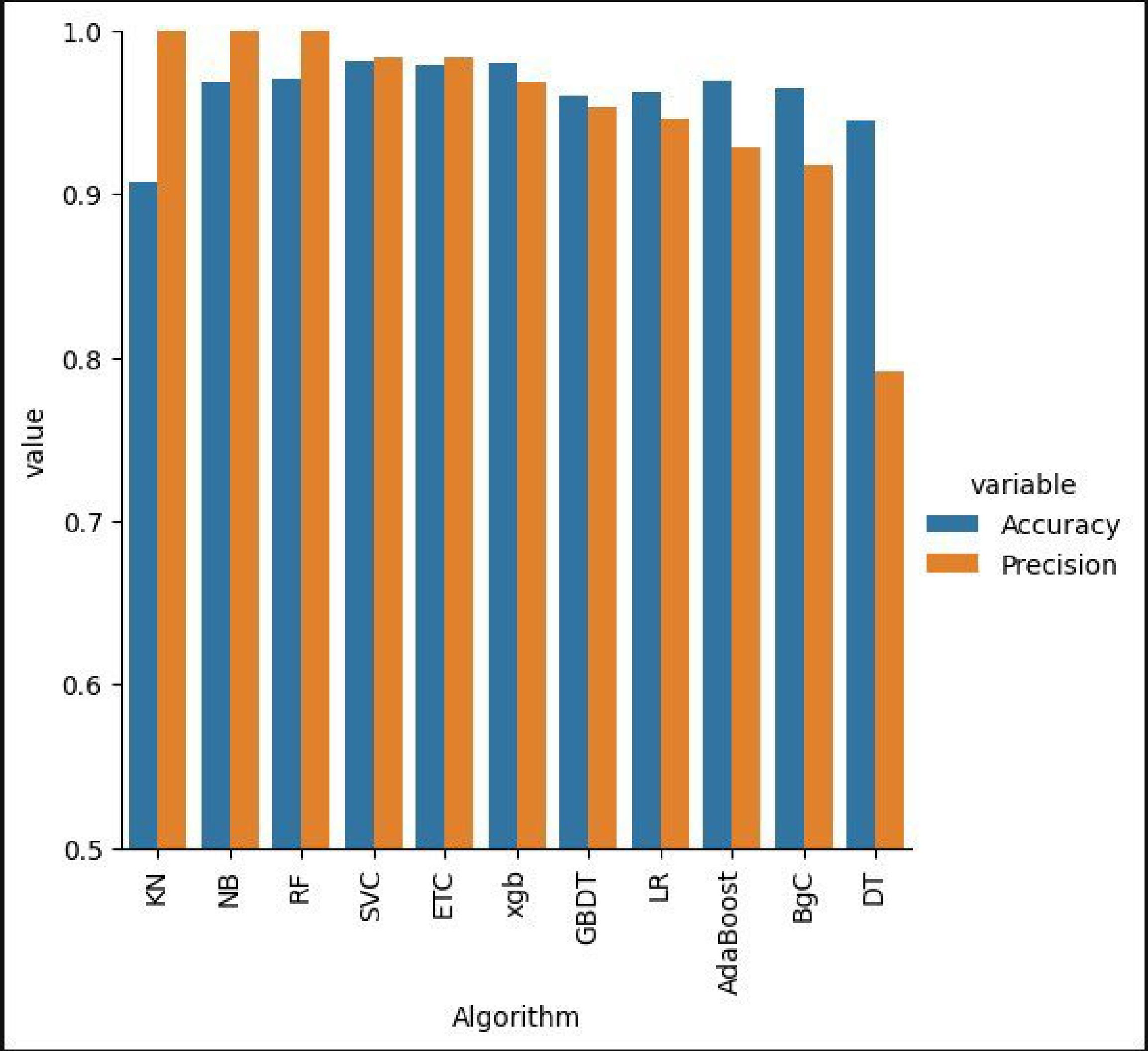
```
[84]: performance_df1 = pd.melt(performance_df, id_vars = "Algorithm")
      performance_df1
```

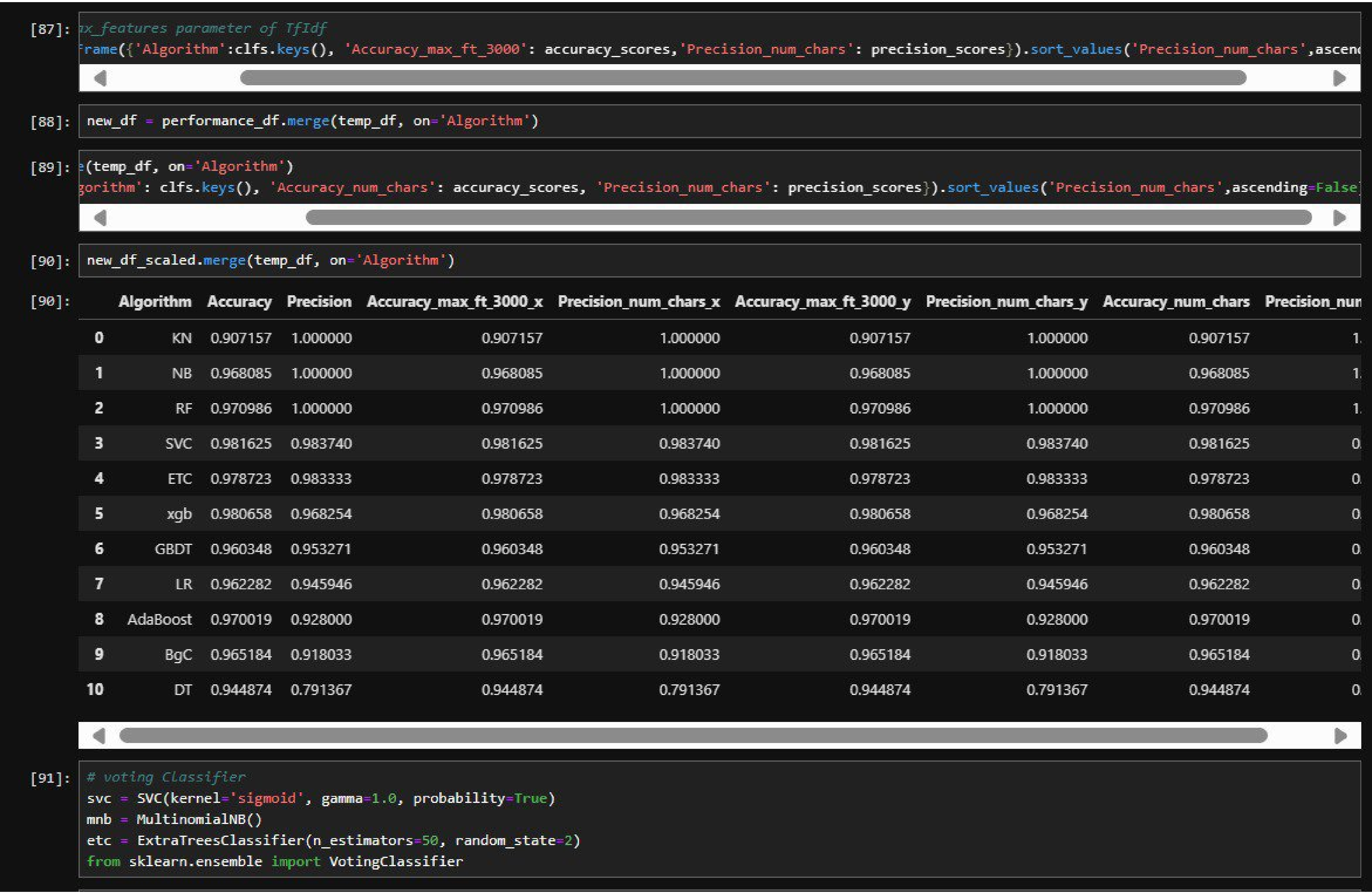
[84]:

	Algorithm	variable	value
0	KN	Accuracy	0.907157
1	NB	Accuracy	0.968085
2	RF	Accuracy	0.970986
3	SVC	Accuracy	0.981625
4	ETC	Accuracy	0.978723
5	xgb	Accuracy	0.980658
6	GBDT	Accuracy	0.960348
7	LR	Accuracy	0.962282
8	AdaBoost	Accuracy	0.970019
9	BgC	Accuracy	0.965184

18	LR	Precision	0.945946
19	AdaBoost	Precision	0.928000
20	BgC	Precision	0.918033
21	DT	Precision	0.791367

```
[85]: sns.catplot(x="Algorithm", y="value", hue="variable", data=performance_df1, kind='bar', height=5)
      plt.ylim(0.5,1.0)
      plt.xticks(rotation='vertical')
      plt.show()
```





```
[87]: ıx_features parameter of TfIdf
      rame({'Algorithm':clfs.keys(), 'Accuracy_max_ft_3000': accuracy_scores,'Precision_num_chars': precision_scores}).sort_values('Precision_num_chars',ascenc
```

```
[88]: new_df = performance_df.merge(temp_df, on='Algorithm')
```

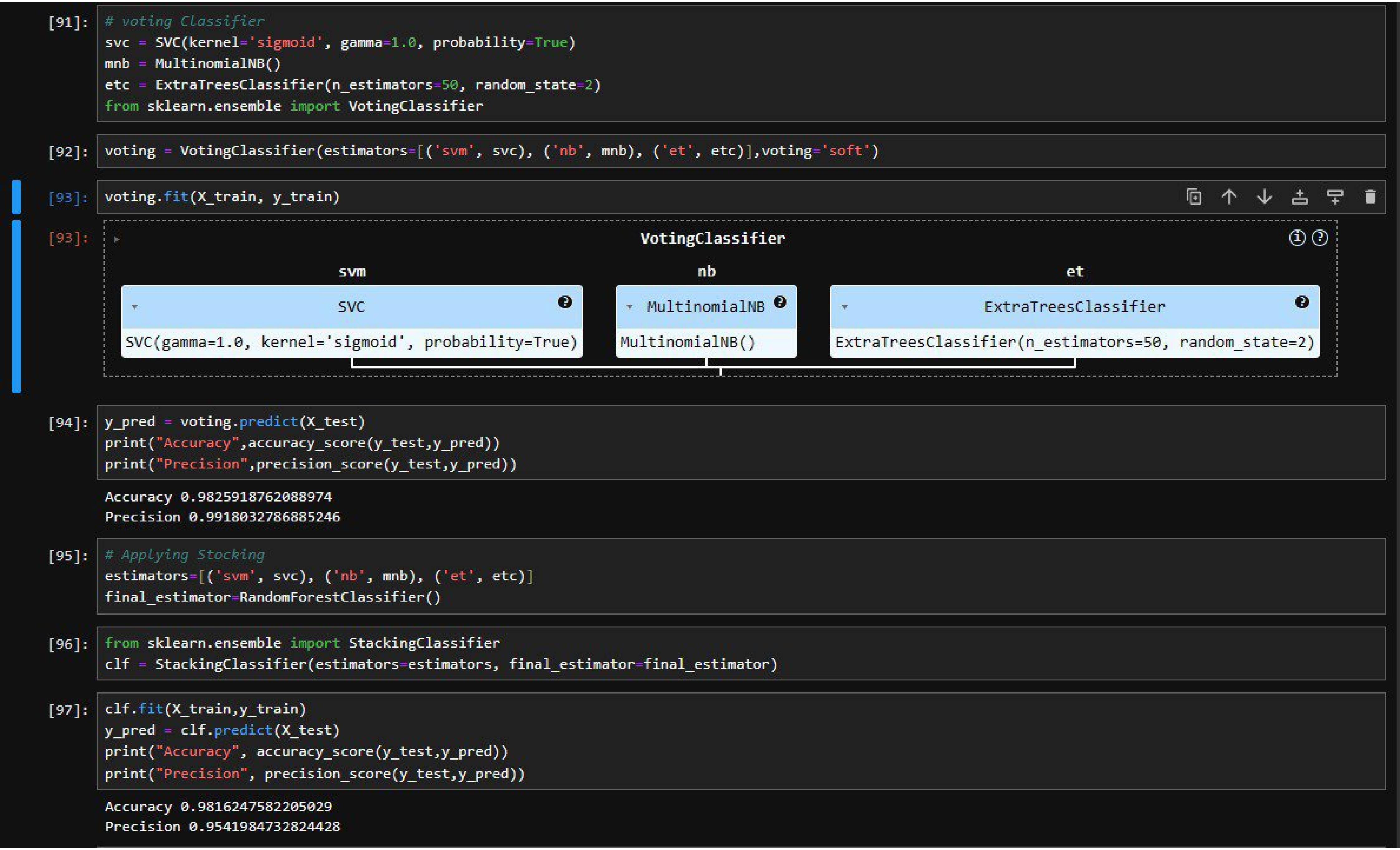
```
[89]: e(temp_df, on='Algorithm')
      gorithm': clfs.keys(), 'Accuracy_num_chars': accuracy_scores, 'Precision_num_chars': precision_scores}).sort_values('Precision_num_chars',ascending=False
```

```
[90]: new_df_scaled.merge(temp_df, on='Algorithm')
```

[90]:

	Algorithm	Accuracy	Precision	Accuracy_max_ft_3000_x	Precision_num_chars_x	Accuracy_max_ft_3000_y	Precision_num_chars_y	Accuracy_num_chars	Precision_nun
0	KN	0.907157	1.000000	0.907157	1.000000	0.907157	1.000000	0.907157	1.
1	NB	0.968085	1.000000	0.968085	1.000000	0.968085	1.000000	0.968085	1.
2	RF	0.970986	1.000000	0.970986	1.000000	0.970986	1.000000	0.970986	1.
3	SVC	0.981625	0.983740	0.981625	0.983740	0.981625	0.983740	0.981625	0.
4	ETC	0.978723	0.983333	0.978723	0.983333	0.978723	0.983333	0.978723	0.
5	xgb	0.980658	0.968254	0.980658	0.968254	0.980658	0.968254	0.980658	0.
6	GBDT	0.960348	0.953271	0.960348	0.953271	0.960348	0.953271	0.960348	0.
7	LR	0.962282	0.945946	0.962282	0.945946	0.962282	0.945946	0.962282	0.
8	AdaBoost	0.970019	0.928000	0.970019	0.928000	0.970019	0.928000	0.970019	0.
9	BgC	0.965184	0.918033	0.965184	0.918033	0.965184	0.918033	0.965184	0.
10	DT	0.944874	0.791367	0.944874	0.791367	0.944874	0.791367	0.944874	0.

```
[91]: # voting Classifier
      svc = SVC(kernel='sigmoid', gamma=1.0, probability=True)
      mnb = MultinomialNB()
      etc = ExtraTreesClassifier(n_estimators=50, random_state=2)
      from sklearn.ensemble import VotingClassifier
```



```
[91]: # voting Classifier
      svc = SVC(kernel='sigmoid', gamma=1.0, probability=True)
      mnb = MultinomialNB()
      etc = ExtraTreesClassifier(n_estimators=50, random_state=2)
      from sklearn.ensemble import VotingClassifier
```

```
[92]: voting = VotingClassifier(estimators=[('svm', svc), ('nb', mnb), ('et', etc)],voting='soft')
```

```
[93]: voting.fit(X_train, y_train)
```

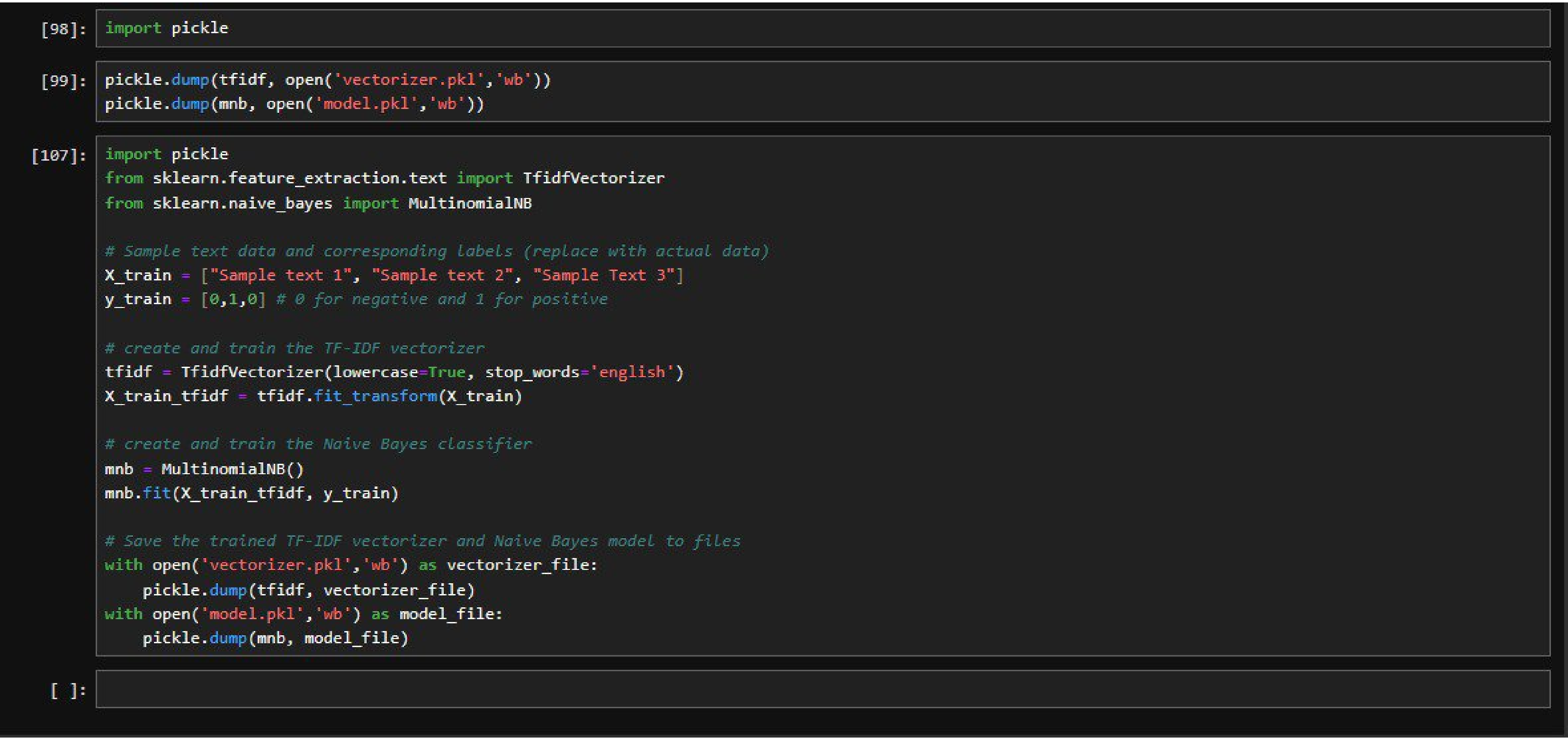
```
[93]: VotingClassifier
        svm: SVC
             SVC(gamma=1.0, kernel='sigmoid', probability=True)
        nb:  MultinomialNB
             MultinomialNB()
        et:  ExtraTreesClassifier
             ExtraTreesClassifier(n_estimators=50, random_state=2)
```

```
[94]: y_pred = voting.predict(X_test)
      print("Accuracy",accuracy_score(y_test,y_pred))
      print("Precision",precision_score(y_test,y_pred))

      Accuracy 0.9825918762088974
      Precision 0.9918032786885246
```

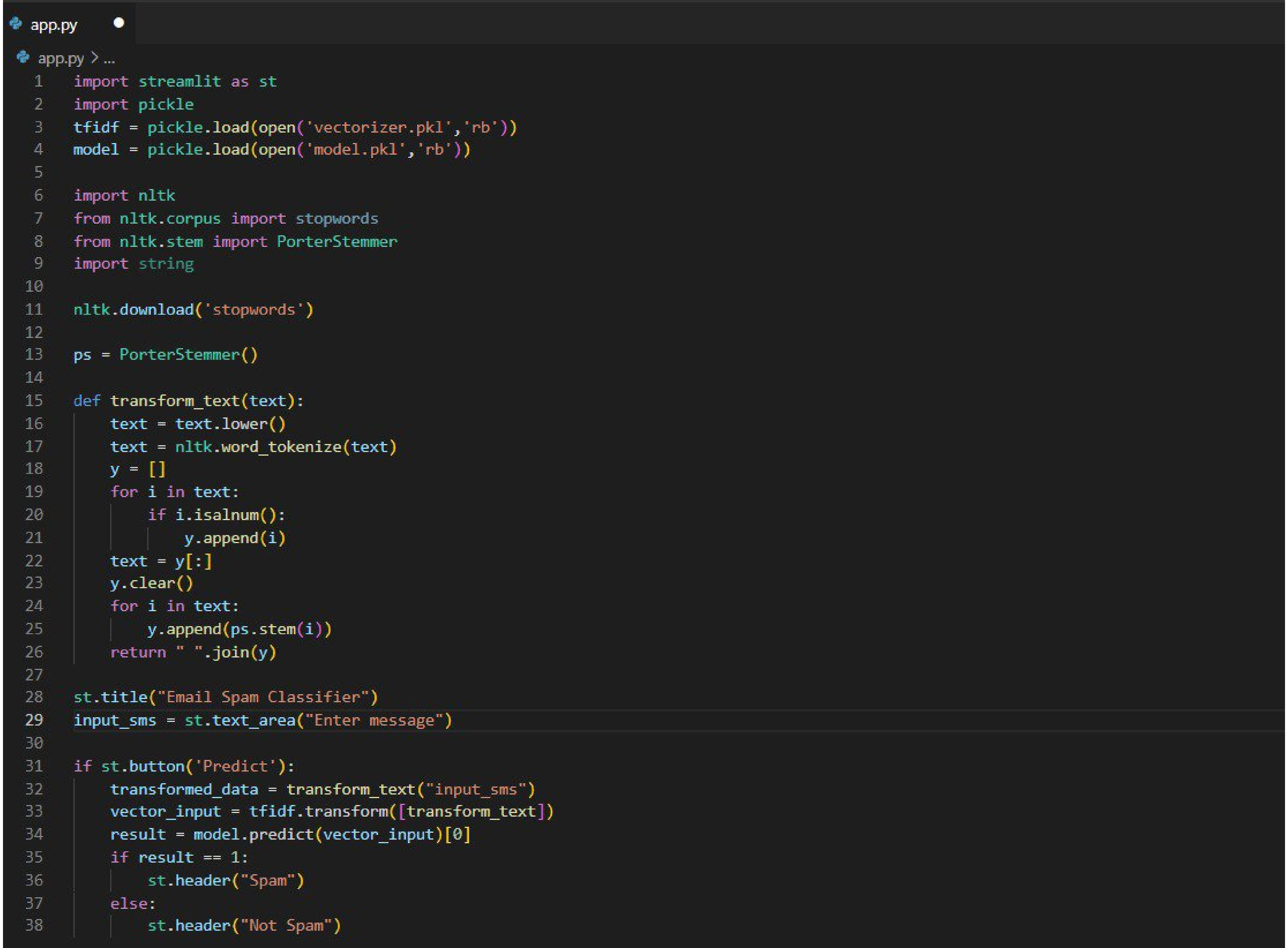
```
[95]: # Applying Stocking
      estimators=[('svm', svc), ('nb', mnb), ('et', etc)]
      final_estimator=RandomForestClassifier()
```

```
[96]: from sklearn.ensemble import StackingClassifier
      clf = StackingClassifier(estimators=estimators, final_estimator=final_estimator)
```

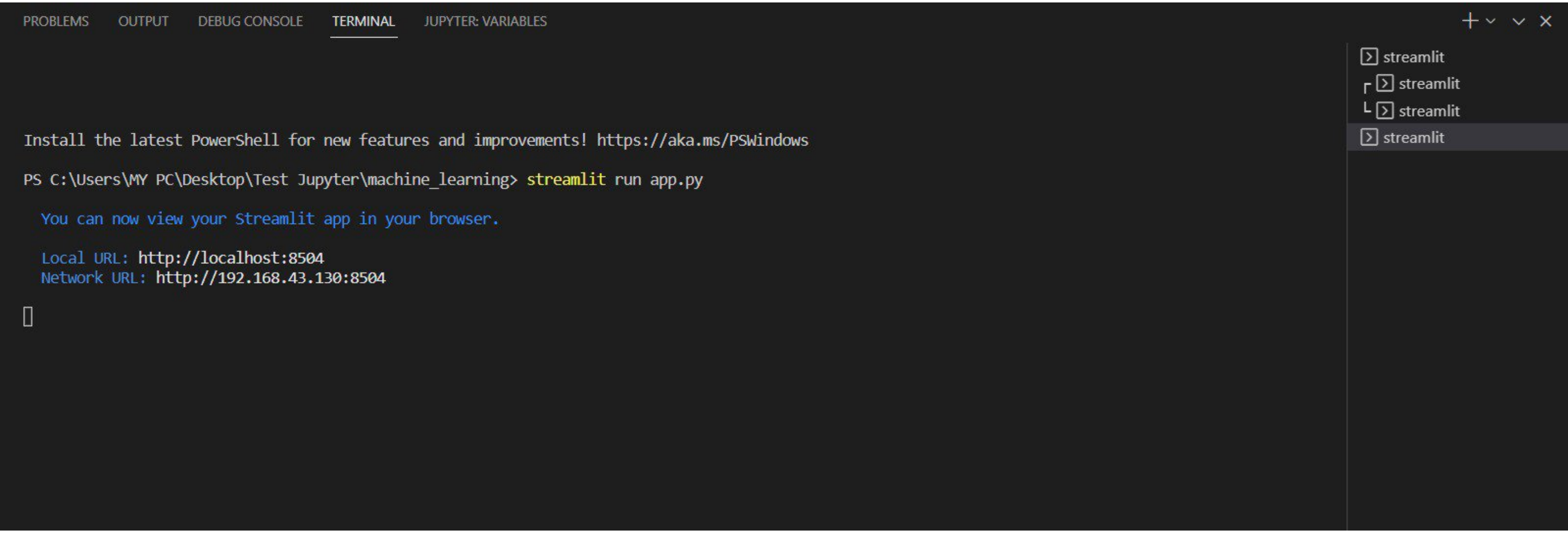
```
[97]: clf.fit(X_train,y_train)
      y_pred = clf.predict(X_test)
      print("Accuracy", accuracy_score(y_test,y_pred))
      print("Precision", precision_score(y_test,y_pred))

      Accuracy 0.9816247582205029
      Precision 0.9541984732824428
```



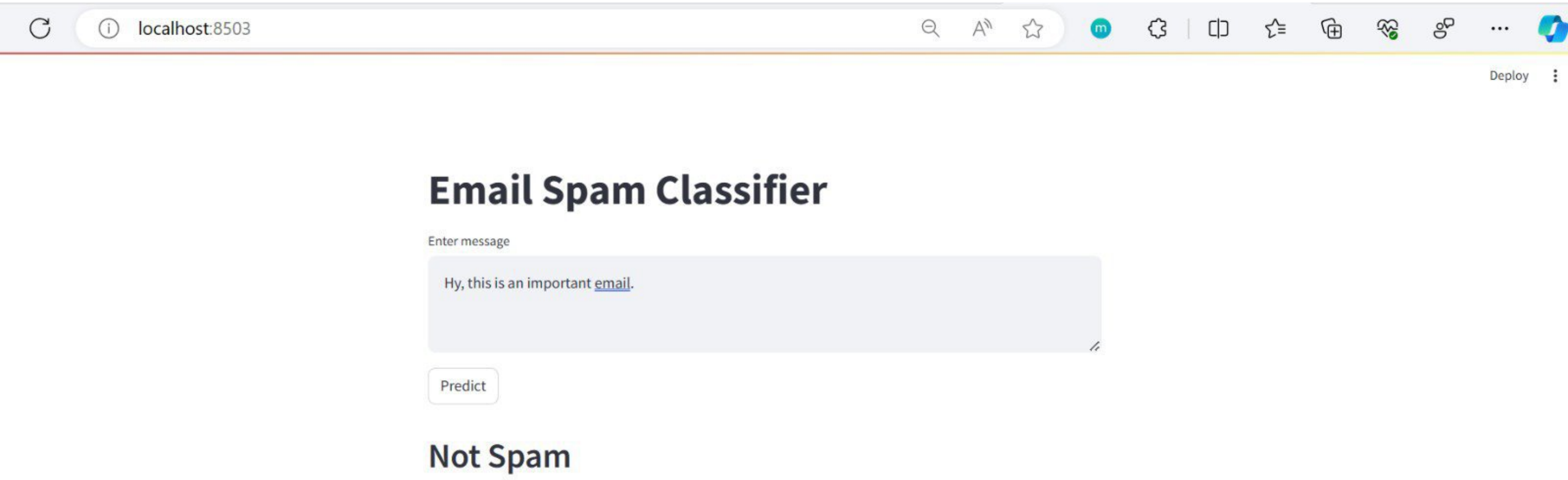
```
[98]:  import pickle

[99]:  pickle.dump(tfidf, open('vectorizer.pkl','wb'))
       pickle.dump(mnb, open('model.pkl','wb'))

[107]: import pickle
       from sklearn.feature_extraction.text import TfidfVectorizer
       from sklearn.naive_bayes import MultinomialNB

       # Sample text data and corresponding labels (replace with actual data)
       X_train = ["Sample text 1", "Sample text 2", "Sample Text 3"]
       y_train = [0,1,0] # 0 for negative and 1 for positive

       # create and train the TF-IDF vectorizer
       tfidf = TfidfVectorizer(lowercase=True, stop_words='english')
       X_train_tfidf = tfidf.fit_transform(X_train)

       # create and train the Naive Bayes classifier
       mnb = MultinomialNB()
       mnb.fit(X_train_tfidf, y_train)

       # Save the trained TF-IDF vectorizer and Naive Bayes model to files
       with open('vectorizer.pkl','wb') as vectorizer_file:
           pickle.dump(tfidf, vectorizer_file)
       with open('model.pkl','wb') as model_file:
           pickle.dump(mnb, model_file)

[ ]:
```



```python
import streamlit as st
import pickle
tfidf = pickle.load(open('vectorizer.pkl','rb'))
model = pickle.load(open('model.pkl','rb'))

import nltk
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
import string

nltk.download('stopwords')

ps = PorterStemmer()

def transform_text(text):
    text = text.lower()
    text = nltk.word_tokenize(text)
    y = []
    for i in text:
        if i.isalnum():
            y.append(i)
    text = y[:]
    y.clear()
    for i in text:
        y.append(ps.stem(i))
    return " ".join(y)

st.title("Email Spam Classifier")
input_sms = st.text_area("Enter message")

if st.button('Predict'):
    transformed_data = transform_text("input_sms")
    vector_input = tfidf.transform([transform_text])
    result = model.predict(vector_input)[0]
    if result == 1:
        st.header("Spam")
    else:
        st.header("Not Spam")
```



```
Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\MY PC\Desktop\Test Jupyter\machine_learning> streamlit run app.py

  You can now view your Streamlit app in your browser.

  Local URL: http://localhost:8504
  Network URL: http://192.168.43.130:8504

```



localhost:8503

Deploy

Email Spam Classifier

Enter message

Hy, this is an important email.

Predict

Not Spam