Naïve Bayes & NLP

▼ Manual Vectorization

Feature Extraction

```
1 with open("One.txt") as mytext:
2     a = mytext.read()
3 a
```

✓ 0.4s

```
'This is a story about dogs\nour canine pets\nDogs are furry animals\n'
```

```
1 a.lower().split()
```

✓ 0.7s

```
['this',
 'is',
 'a',
 'story',
 'about',
 'dogs',
 'our',
 'canine',
 'pets',
 'dogs',
 'are',
 'furry',
 'animals']
```

```
1 with open("One.txt") as mytext:
2     a = mytext.readlines()
3 a
```

✓ 0.1s

```
['This is a story about dogs\n',
 'our canine pets\n',
 'Dogs are furry animals\n']
```

```
1  with open("One.txt") as mytext:
2      words_one = mytext.read().lower().split()
3      uni_words_one = set(words_one)
4  uni_words_one
```

✓ 0.9s

```
{'a',
 'about',
 'animals',
 'are',
 'canine',
 'dogs',
 'furry',
 'is',
 'our',
 'pets',
 'story',
 'this'}
```

```
1  with open("Two.txt") as mytext:
2      words_two = mytext.read().lower().split()
3      uni_words_two = set(words_two)
4  uni_words_two
```

✓ 0.1s

```
{'a',
 'about',
 'catching',
 'fun',
 'is',
 'popular',
 'sport',
 'story',
 'surfing',
 'this',
 'water',
 'waves'}
```

```python
all_uni_words = set()
all_uni_words.update(uni_words_one)
all_uni_words.update(uni_words_two)
all_uni_words
# boş bir set oluşturuldu .update ile içine uni_words
# .. setleri atıldı ve Unique kelimeleri birleştirdik.
```

✓ 0.2s

```
{'a',
 'about',
 'animals',
 'are',
 'canine',
 'catching',
 'dogs',
 'fun',
 'furry',
 'is',
 'our',
 'pets',
 'popular',
 'sport',
 'story',
 'surfing',
 'this',
 'water',
 'waves'}
```

```python
full_vocab = dict()
i=0

for word in all_uni_words:
    full_vocab[word] = i
    i = i+1

full_vocab
```

✓ 0.1s

```
{'catching': 0,
 'dogs': 1,
 'our': 2,
 'a': 3,
 'waves': 4,
 'are': 5,
 'canine': 6,
 'water': 7,
 'furry': 8,
 'surfing': 9,
 'about': 10,
 'popular': 11,
 'pets': 12,
 'this': 13,
 'is': 14,
 'fun': 15,
 'sport': 16,
 'story': 17,
 'animals': 18}
```

```
one_freq = [0]*len(full_vocab)
two_freq = [0]*len(full_vocab)
all_words = [""]*len(full_vocab)
```

✓ 0.5s

```
with open("One.txt") as f:
    one_text = f.read().lower().split()
```

✓ 0.9s

```
for word in one_text:
    word_ind = full_vocab[word]
    one_freq[word_ind] += 1

one_freq
```

✓ 0.9s

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

```
with open("Two.txt") as f:
    two_text = f.read().lower().split()
```

✓ 0.1s

```
for word in two_text:
    word_ind = full_vocab[word]
    two_freq[word_ind] += 1

two_freq
```

✓ 0.1s

```
[1, 0, 0, 1, 1, 0, 0, 1, 0, 2, 1, 1, 0, 1, 3, 1, 1, 1, 0]
```

```
for word in full_vocab:
    word_ind = full_vocab[word]
    all_words[word_ind] = word

all_words
```

✓ 0.1s

```
['catching',
 'dogs',
 'our',
 'a',
 'waves',
 'are',
 'canine',
 'water',
 'furry',
 'surfing',
 'about',
 'popular',
 'pets',
 'this',
 'is',
 'fun',
 'sport',
 'story',
 'animals']
```

```
import pandas as pd
```

✓ 0.8s Python

```
bow = pd.DataFrame(data=[one_freq,two_freq], columns=all_words)
bow
```

✓ 0.1s Python

	catching	dogs	our	a	waves	are	canine	water	furry	surfing	about	popular	pets	this	is	fun	sport	story
0	0	2	1	1	0	1	1	0	1	0	1	0	1	1	1	0	0	1
1	1	0	0	1	1	0	0	1	0	2	1	1	0	1	3	1	1	1

▼ Scikit-Learn vectorization

```
text = [
    "This is a line",
    "This is another line",
    "Compeletely another line"
]
```

```
from sklearn.feature_extraction.text import CountVectorizer, TfidfTransformer
```

```
cv = CountVectorizer()
# cv = CountVectorizer(stop_words="english") <
#  a, this, or, is gibi kelimeleri kaldırır
```

```
sparse_matrix = cv.fit_transform(text)
```

```
sparse_matrix.todense()
```

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

```
1 cv.vocabulary_
```

```
{'this': 4, 'is': 2, 'line': 3, 'another': 0, 'compeletely': 1}
```

```
1 tfidf = TfidfTransformer()
```

```
1 results = tfidf.fit_transform(sparse_matrix)
```

```
1 results.todense()
```

```
matrix([[0.        , 0.        , 0.61980538, 0.48133417, 0.61980538],
        [0.52682017, 0.        , 0.52682017, 0.40912286, 0.52682017],
        [0.54783215, 0.72033345, 0.        , 0.42544054, 0.        ]])
```

```
1 from sklearn.feature_extraction.text import TfidfVectorizer
```

```
1 tv = TfidfVectorizer()
```

```
1 tv_results = tv.fit_transform(text)
```

```
1 tv_results.todense()
```

```
matrix([[0.        , 0.        , 0.61980538, 0.48133417, 0.61980538],
        [0.52682017, 0.        , 0.52682017, 0.40912286, 0.52682017],
        [0.54783215, 0.72033345, 0.        , 0.42544054, 0.        ]])
```

▼ Airline Tweets

Imports, Data and EDA

```
1 import pandas as pd
2 import seaborn as sns
3 import matplotlib.pyplot as plt
```

✓ 5.2s

```
1 df = pd.read_csv("airline_tweets.csv")
```

✓ 0.1s

```
1 df.head()
```

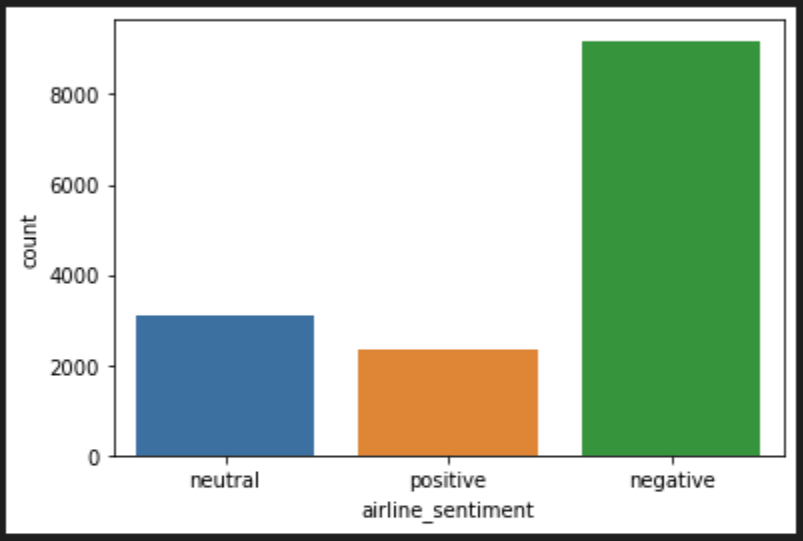
✓ 0.6s

	tweet_id	airline_sentiment	airline_sentiment_confidence	negativereason	ne
0	570306133677760513	neutral	1.0000	NaN	
1	570301130888122368	positive	0.3486	NaN	
2	570301083672813571	neutral	0.6837	NaN	

```
1  sns.countplot(data=df, x="airline_sentiment")
```

✓ 0.3s

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



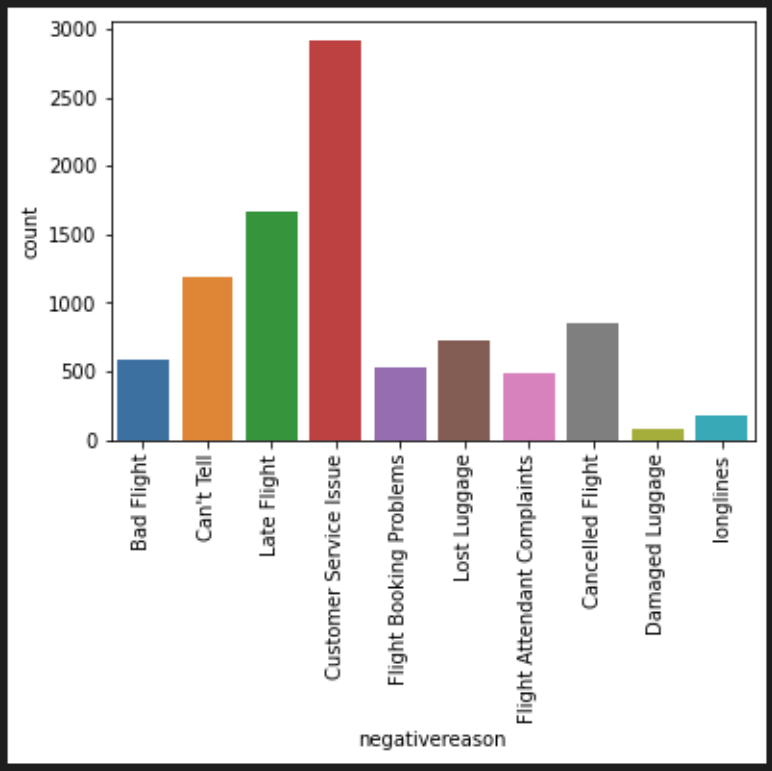

```
1  df['airline_sentiment'].value_counts()
```

✓ 0.5s

```
negative    9178
neutral     3099
positive    2363
Name: airline_sentiment, dtype: int64
```

```
1  sns.countplot(data=df, x="negativereason")
2  plt.xticks(rotation=90);
```

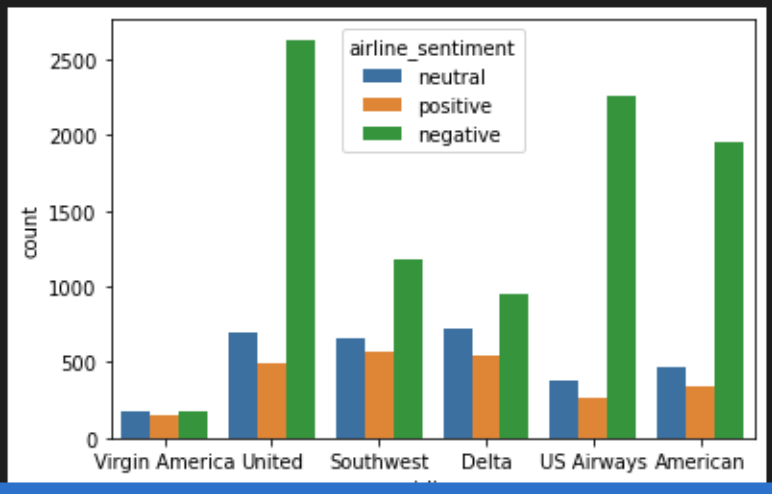
✓ 0.3s




```
1  sns.countplot(data=df, x="airline", hue="airline_sentiment")
```

✓ 0.2s

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



ML Model

```
1  data = df[["airline_sentiment","text"]]
2  data
```

✓ 0.1s

	airline_sentiment	text
0	neutral	@VirginAmerica What @dhepburn said.
1	positive	@VirginAmerica plus you've added commercials t...
2	neutral	@VirginAmerica I didn't today... Must mean I n...
3	negative	@VirginAmerica it's really aggressive to blast...
4	negative	@VirginAmerica and it's a really big bad thing...

```
1  X = df['text']
2  y = df['airline_sentiment']
```

✓ 0.6s

```
1  from sklearn.model_selection import train_test_split
```

✓ 0.9s

```
1  X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=101)
```

✓ 0.1s

```
1  from sklearn.feature_extraction.text import TfidfVectorizer
```
✓ 0.2s

```
1  tfidf = TfidfVectorizer(stop_words="english")
```
✓ 0.5s

```
1  tfidf.fit(X_train)
```
✓ 0.3s

```
TfidfVectorizer(stop_words='english')
```

```
1  X_train_tfidf = tfidf.transform(X_train)
2  X_test_tfidf = tfidf.transform(X_test)
```
✓ 0.3s

Model Comparison

```
1  # NAIVE BAYES
2  from sklearn.naive_bayes import MultinomialNB
```
✓ 0.3s

```
1  nb = MultinomialNB()
```
✓ 0.4s

```
1  nb.fit(X_train_tfidf, y_train)
```
✓ 0.7s

```
MultinomialNB()
```

```
# LOGISTIC REGR
from sklearn.linear_model import LogisticRegression
```
✓ 0.4s

```
log_model = LogisticRegression(max_iter= 1000)
```
✓ 0.5s

```
log_model.fit(X_train_tfidf, y_train)
```
✓ 2.3s

```
LogisticRegression(max_iter=1000)
```

```
# SUPPORT VECTOR MACHINES
from sklearn.svm import SVC, LinearSVC
```
✓ 0.4s

```
rbf_svc = SVC()
```
✓ 0.8s

```
rbf_svc.fit(X_train_tfidf, y_train)
```
✓ 17.1s

```
SVC()
```

```
linear_svc = LinearSVC()
```
✓ 0.4s

```
linear_svc.fit(X_train_tfidf, y_train)
```
✓ 0.1s

```
LinearSVC()
```

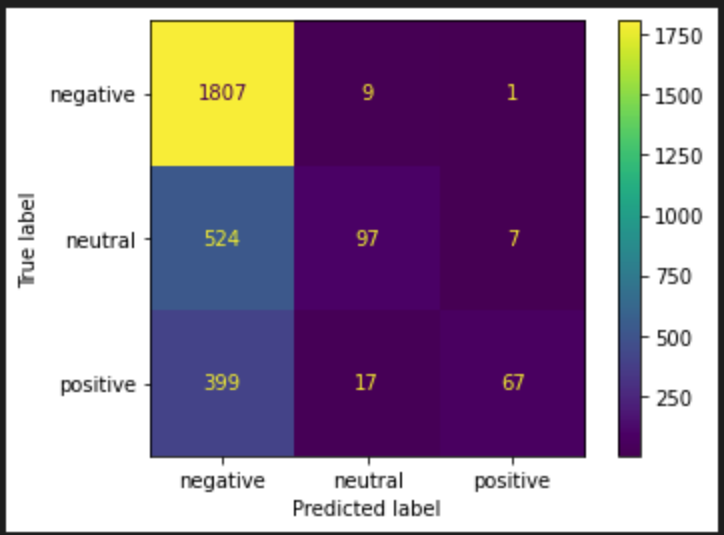
```
1 from sklearn.metrics import plot_confusion_matrix, classification_report
```
✓ 0.4s

```
1 def report(model):
2     preds = model.predict(X_test_tfidf)
3     print(classification_report(y_test, preds))
4     plot_confusion_matrix(model, X_test_tfidf, y_test)
```
✓ 0.4s

and Debug (Ctrl+Shift+D)

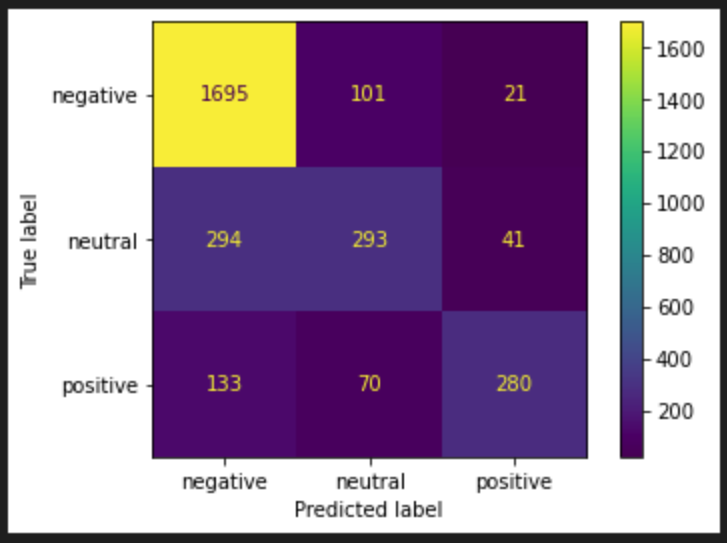
```
1 report(nb)
```
✓ 0.3s

	precision	recall	f1-score	support
negative	0.66	0.99	0.79	1817
neutral	0.79	0.15	0.26	628
positive	0.89	0.14	0.24	483
accuracy			0.67	2928
macro avg	0.78	0.43	0.43	2928
weighted avg	0.73	0.67	0.59	2928



```
1 report(log_model)
✓ 0.4s
```

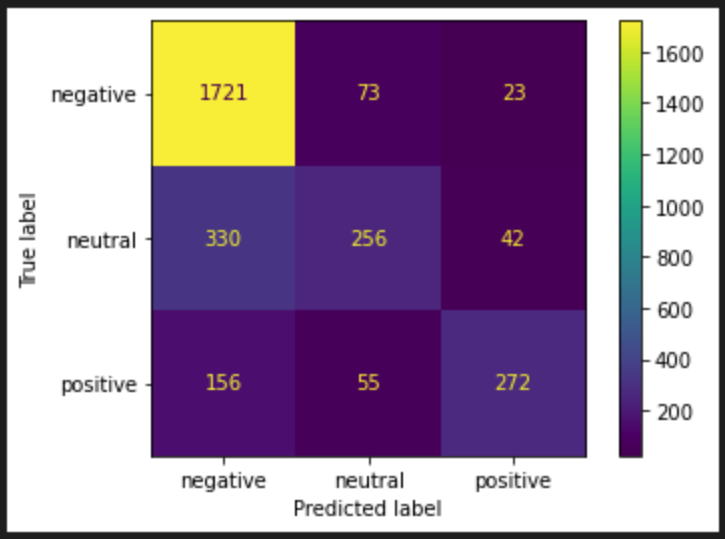
	precision	recall	f1-score	support
negative	0.80	0.93	0.86	1817
neutral	0.63	0.47	0.54	628
positive	0.82	0.58	0.68	483
accuracy			0.77	2928
macro avg	0.75	0.66	0.69	2928
weighted avg	0.77	0.77	0.76	2928



```
1  report(rbf_svc)
```

✓ 4.8s

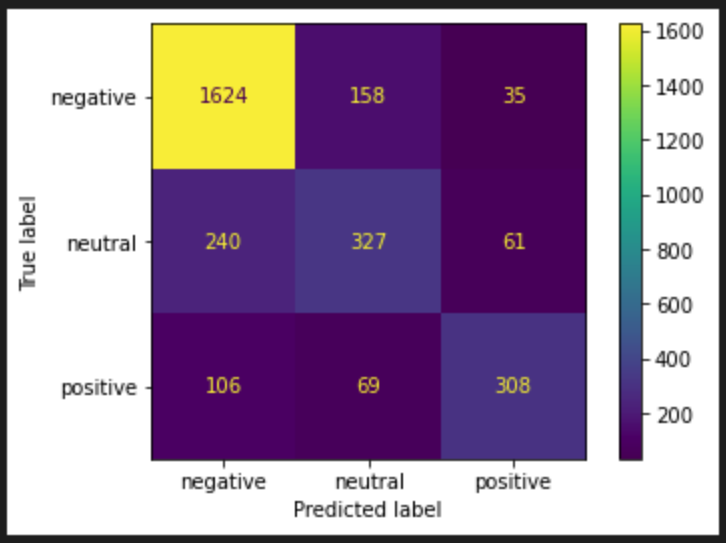
	precision	recall	f1-score	support
negative	0.78	0.95	0.86	1817
neutral	0.67	0.41	0.51	628
positive	0.81	0.56	0.66	483
accuracy			0.77	2928
macro avg	0.75	0.64	0.67	2928
weighted avg	0.76	0.77	0.75	2928



```
1  report(linear_svc)
```

✓ 0.4s

	precision	recall	f1-score	support
negative	0.82	0.89	0.86	1817
neutral	0.59	0.52	0.55	628
positive	0.76	0.64	0.69	483
accuracy			0.77	2928
macro avg	0.73	0.68	0.70	2928
weighted avg	0.76	0.77	0.77	2928



```
1 from sklearn.pipeline import Pipeline
```
✓ 0.4s

```
1 pipe = Pipeline([
2     ("tfidf",TfidfVectorizer()),
3     ("svc",LinearSVC())
4     ])
```
✓ 0.4s

```
1 pipe.fit(X,y)
```
✓ 0.6s

```
Pipeline(steps=[('tfidf', TfidfVectorizer()), ('svc', LinearSVC())])
```

```
1 pipe.predict(["good flight"])
```
✓ 0.1s

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

```
1 pipe.predict(["badd flight"])
```
✓ 0.1s

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

```
1 pipe.predict(["meh flight"])
```
✓ 0.1s

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

▼ Movie review

Data Cleaning

```
1 import numpy as np
2 import pandas as pd
```

✓ 0.1s

```
1 df = pd.read_csv('moviereviews.csv')
```

✓ 0.1s

```
1 df.head()
```

✓ 0.1s

	label	review
0	neg	how do films like mouse hunt get into theatres...
1	neg	some talented actresses are blessed with a dem...
2	pos	this has been an extraordinary year for austra...
3	pos	according to hollywood movies made in last few...
4	neg	my first press screening of 1998 and already i...

```
1 df.isnull().sum()
```

✓ 0.1s

```
label      0
review    35
dtype: int64
```

```
1 df.isna().sum()
```

✓ 0.9s

```
label      0
review    35
dtype: int64
```

```
1 df = df.dropna()
```
✓ 0.1s

```
1 df.isnull().sum()
```
✓ 0.1s

```
label     0
review    0
dtype: int64
```

```
1 df["review"].str.isspace().sum()
```
✓ 0.7s

```
27
```

```
1 df = df[~df["review"].str.isspace()]
2 # ~ "is not" olarak kullanılır.
3 # review kolonu boşluk olmayanları getirdik
4 df
```
✓ 0.1s

	label	review
0	neg	how do films like mouse hunt get into theatres...
1	neg	some talented actresses are blessed with a dem...
2	pos	this has been an extraordinary year for austra...
3	pos	according to hollywood movies made in last few...
4	neg	my first press screening of 1998 and already i...
...	...	...

```
1  df[df["review"].apply(lambda review: review=="")]
2  # doğru yaptık demektir.
```

✓ 0.1s

label	review

```
1  df.info()
```

✓ 0.7s

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1938 entries, 0 to 1999
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   label   1938 non-null   object
 1   review  1938 non-null   object
dtypes: object(2)
memory usage: 45.4+ KB
```

```
1  df["label"].value_counts()
```

✓ 0.6s

```
neg    969
pos    969
Name: label, dtype: int64
```

EDA

```
1 from sklearn.feature_extraction.text import CountVectorizer
```

✓ 0.4s Python

```
1 cv = CountVectorizer(stop_words='english')
```

✓ 0.4s Python

Kelimeleri saymak için kullandık

```
1 matrix = cv.fit_transform(df[df['label']=='neg']['review'])
2 freqs = zip(cv.get_feature_names(), matrix.sum(axis=0).tolist()[0])
3 # sort from largest to smallest
4 print("Top 20 words used for Negative reviews.")
5 print(sorted(freqs, key=lambda x: -x[1])[:20])
```

✓ 0.7s Python

```
Top 20 words used for Negative reviews.
[('film', 4063), ('movie', 3131), ('like', 1808), ('just', 1480), ('time',
1127), ('good', 1117), ('bad', 997), ('character', 926), ('story', 908),
('plot', 888), ('characters', 838), ('make', 813), ('really', 743),
('way', 734), ('little', 696), ('don', 683), ('does', 666), ('doesn',
648), ('action', 635), ('scene', 634)]
```

```
1 matrix = cv.fit_transform(df[df['label']=='pos']['review'])
2 freqs = zip(cv.get_feature_names(), matrix.sum(axis=0).tolist()[0])
3 # sort from largest to smallest
4 print("Top 20 words used for Positive reviews.")
5 print(sorted(freqs, key=lambda x: -x[1])[:20])
```

✓ 0.4s Pytho

```
Top 20 words used for Positive reviews.
[('film', 5002), ('movie', 2389), ('like', 1721), ('just', 1273), ('story', 1199), ('good', 1193),
('time', 1175), ('character', 1037), ('life', 1032), ('characters', 957), ('way', 864), ('films',
851), ('does', 828), ('best', 788), ('people', 769), ('make', 764), ('little', 751), ('really',
731), ('man', 728), ('new', 702)]
```

burada word map de kullanılabilir

ML Model

```
1 from sklearn.model_selection import train_test_split
```

✓ 0.1s Pyt

```
1 X = df['review']
2 y = df['label']
```

✓ 0.8s Pyt

```
1 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=101)
```

✓ 0.8s Pyt

```
1 from sklearn.pipeline import Pipeline
2 from sklearn.feature_extraction.text import TfidfVectorizer
3 from sklearn.svm import LinearSVC
```

✓ 0.8s

```
1 pipe = Pipeline([
2     ('tfidf', TfidfVectorizer()),
3     ('svc', LinearSVC())
4 ])
```

✓ 0.8s

```
1 # Feed the training data through the pipeline
2 pipe.fit(X_train, y_train)
```

✓ 1.3s

```
Pipeline(steps=[('tfidf', TfidfVectorizer()), ('svc', LinearSVC())])
```

```
1 from sklearn.metrics import classification_report,plot_confusion_matrix
```
✓ 0.7s

```
1 preds = pipe.predict(X_test)
```
✓ 0.5s

```
1 print(classification_report(y_test,preds))
```
✓ 0.9s

```
              precision    recall  f1-score   support

         neg       0.81      0.86      0.83       191
         pos       0.85      0.81      0.83       197

    accuracy                           0.83       388
   macro avg       0.83      0.83      0.83       388
weighted avg       0.83      0.83      0.83       388
```

```
1 plot_confusion_matrix(pipe,X_test,y_test)
```
✓ 1.3s

```
<sklearn.metrics._plot.confusion_matrix.Confusio
```