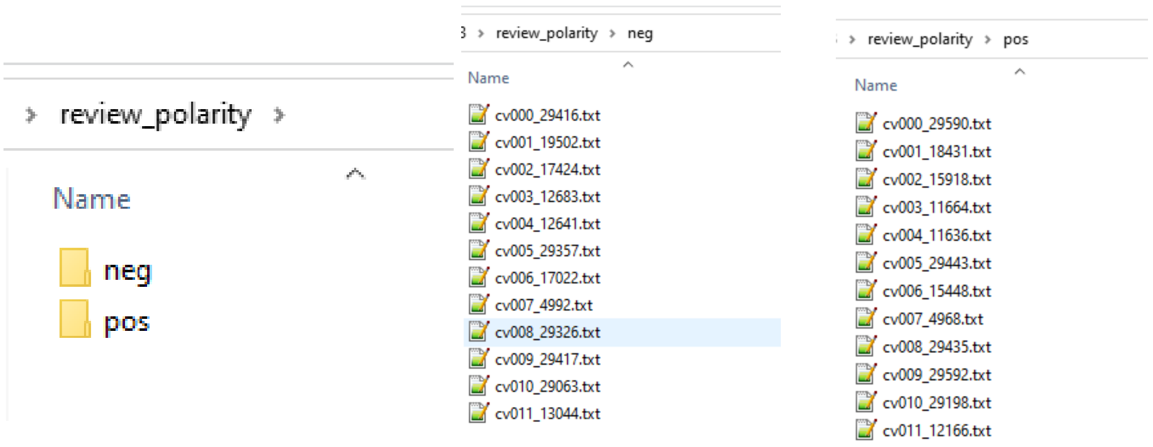
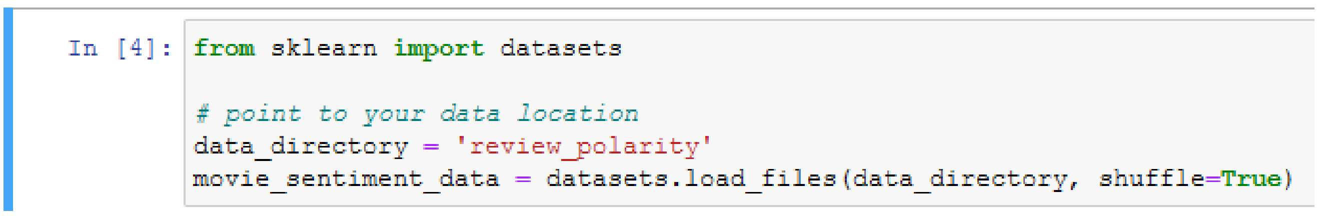
Section 5:

1. Movie Review Data

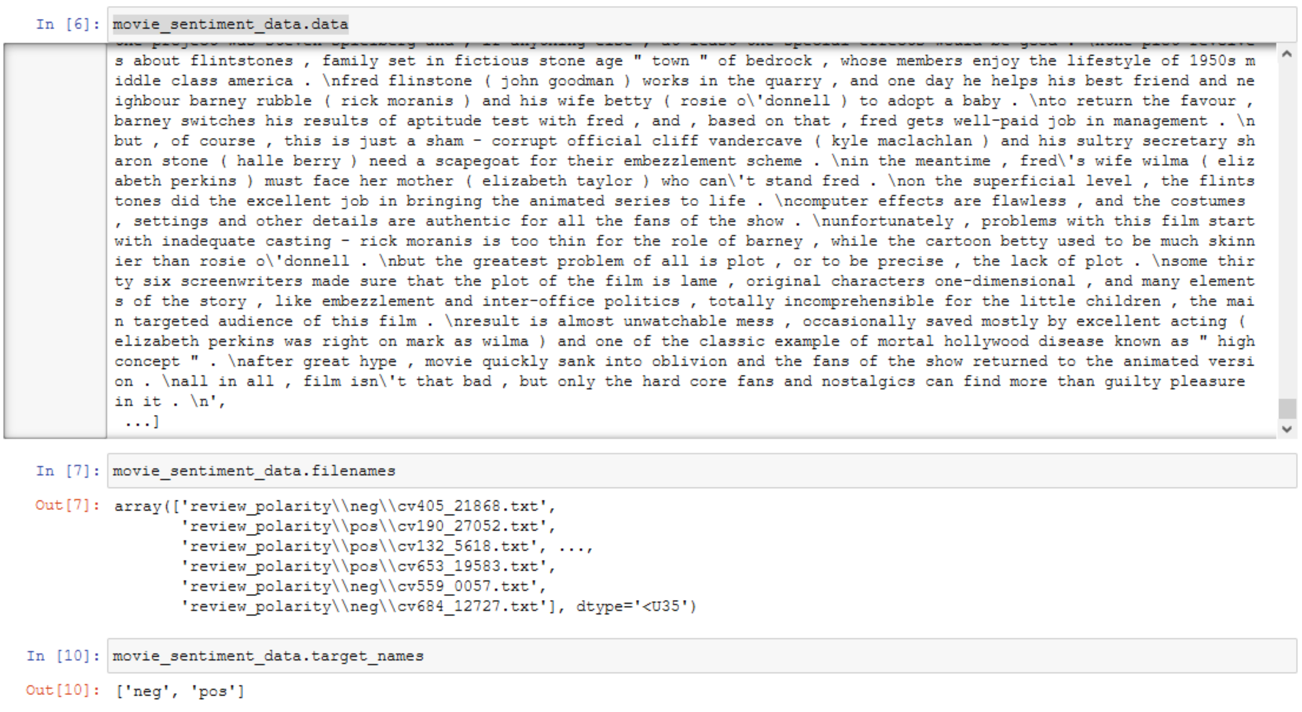
- http://www.cs.cornell.edu/people/pabo/movie-review-data/
  - *Download* polarity dataset v2.0 (3.0Mb)
  - 1000 positive and 1000 negative processed reviews
- *Extract*



2. Loading data

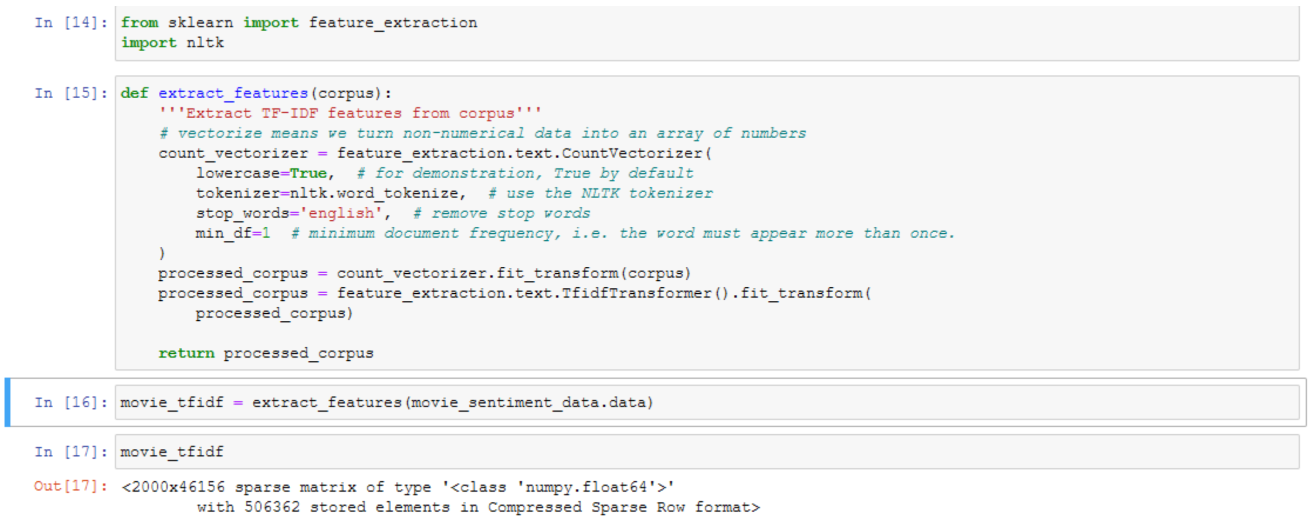


```
In [4]: from sklearn import datasets

        # point to your data location
        data_directory = 'review_polarity'
        movie_sentiment_data = datasets.load_files(data_directory, shuffle=True)
```



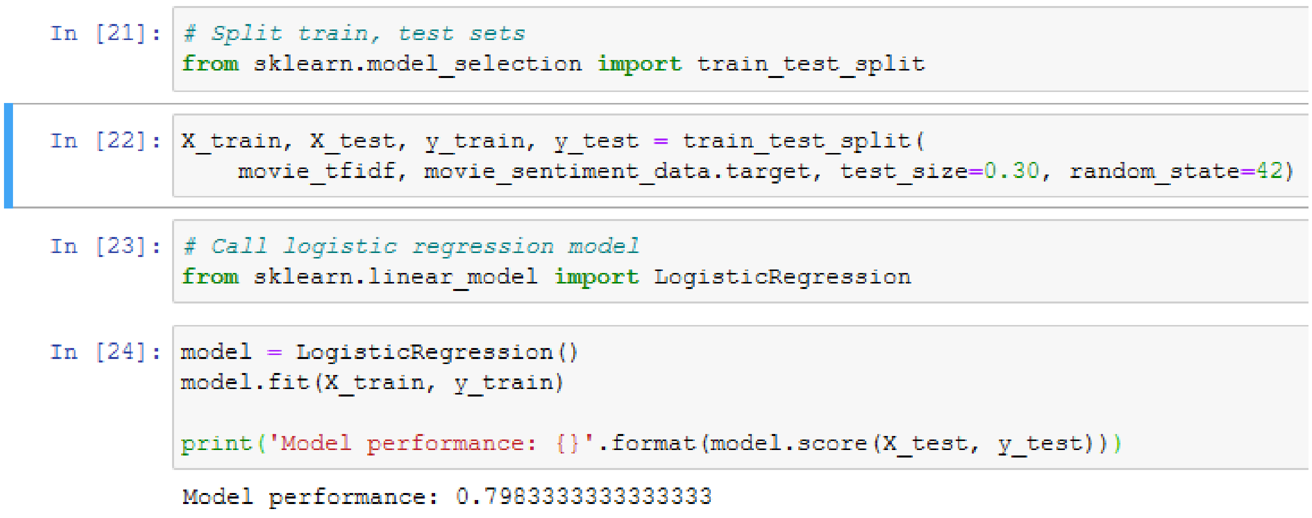
```
In [6]: movie_sentiment_data.data

        s about flintstones , family set in fictious stone age " town " of bedrock , whose members enjoy the lifestyle of 1950s m
        iddle class america . \nfred flinstone ( john goodman ) works in the quarry , and one day he helps his best friend and ne
        ighbour barney rubble ( rick moranis ) and his wife betty ( rosie o\'donnell ) to adopt a baby . \nto return the favour ,
        barney switches his results of aptitude test with fred , and , based on that , fred gets well-paid job in management . \n
        but , of course , this is just a sham - corrupt official cliff vandercave ( kyle maclachlan ) and his sultry secretary sh
        aron stone ( halle berry ) need a scapegoat for their embezzlement scheme . \nin the meantime , fred\'s wife wilma ( eliz
        abeth perkins ) must face her mother ( elizabeth taylor ) who can\'t stand fred . \non the superficial level , the flints
        tones did the excellent job in bringing the animated series to life . \ncomputer effects are flawless , and the costumes
        , settings and other details are authentic for all the fans of the show . \nunfortunately , problems with this film start
        with inadequate casting - rick moranis is too thin for the role of barney , while the cartoon betty used to be much skinn
        ier than rosie o\'donnell . \nbut the greatest problem of all is plot , or to be precise , the lack of plot . \nsome thir
        ty six screenwriters made sure that the plot of the film is lame , original characters one-dimensional , and many element
        s of the story , like embezzlement and inter-office politics , totally incomprehensible for the little children , the mai
        n targeted audience of this film . \nresult is almost unwatchable mess , occasionally saved mostly by excellent acting (
        elizabeth perkins was right on mark as wilma ) and one of the classic example of mortal hollywood disease known as " high
        concept " . \nafter great hype , movie quickly sank into oblivion and the fans of the show returned to the animated versi
        on . \nall in all , film isn\'t that bad , but only the hard core fans and nostalgics can find more than guilty pleasure
        in it . \n',
         ...]

In [7]: movie_sentiment_data.filenames

Out[7]: array(['review_polarity\\neg\\cv405_21868.txt',
               'review_polarity\\pos\\cv190_27052.txt',
               'review_polarity\\pos\\cv132_5618.txt', ...,
               'review_polarity\\pos\\cv653_19583.txt',
               'review_polarity\\neg\\cv559_0057.txt',
               'review_polarity\\neg\\cv684_12727.txt'], dtype='<U35')

In [10]: movie_sentiment_data.target_names

Out[10]: ['neg', 'pos']
```

3. Preprocessing the dataset



```
In [14]: from sklearn import feature_extraction
         import nltk

In [15]: def extract_features(corpus):
             '''Extract TF-IDF features from corpus'''
             # vectorize means we turn non-numerical data into an array of numbers
             count_vectorizer = feature_extraction.text.CountVectorizer(
                 lowercase=True,  # for demonstration, True by default
                 tokenizer=nltk.word_tokenize,  # use the NLTK tokenizer
                 stop_words='english',  # remove stop words
                 min_df=1  # minimum document frequency, i.e. the word must appear more than once.
             )
             processed_corpus = count_vectorizer.fit_transform(corpus)
             processed_corpus = feature_extraction.text.TfidfTransformer().fit_transform(
                 processed_corpus)

             return processed_corpus

In [16]: movie_tfidf = extract_features(movie_sentiment_data.data)

In [17]: movie_tfidf

Out[17]: <2000x46156 sparse matrix of type '<class 'numpy.float64'>'
                 with 506362 stored elements in Compressed Sparse Row format>
```

4. Logistic regression model



```
In [21]: # Split train, test sets
         from sklearn.model_selection import train_test_split

In [22]: X_train, X_test, y_train, y_test = train_test_split(
             movie_tfidf, movie_sentiment_data.target, test_size=0.30, random_state=42)

In [23]: # Call logistic regression model
         from sklearn.linear_model import LogisticRegression

In [24]: model = LogisticRegression()
         model.fit(X_train, y_train)

         print('Model performance: {}'.format(model.score(X_test, y_test)))

         Model performance: 0.7983333333333333
```

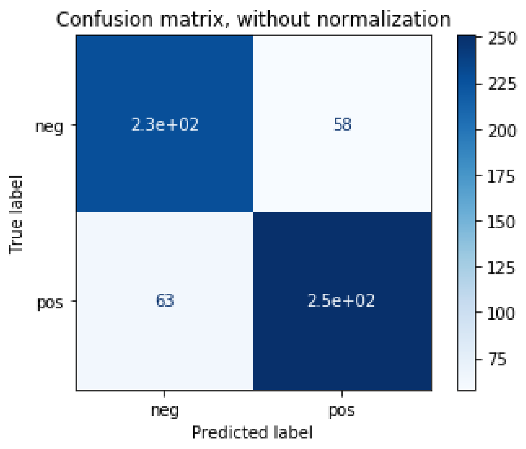
```
In [28]: p_pred = model.predict(X_test)
```

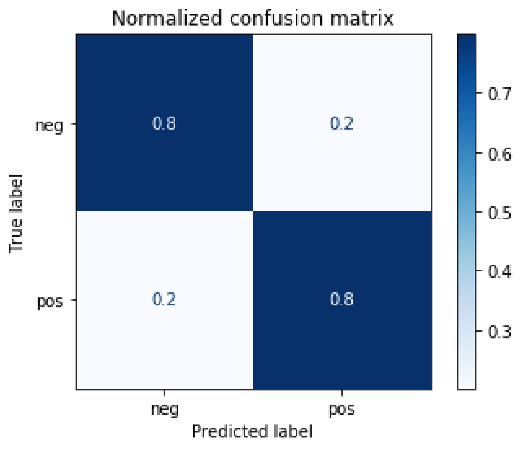
```
In [33]: from sklearn.metrics import plot_confusion_matrix
         import matplotlib.pyplot as plt
```

```
In [34]: classifier = model
         class_names = movie_sentiment_data.target_names

         # Plot non-normalized confusion matrix
         titles_options = [("Confusion matrix, without normalization", None),
                           ("Normalized confusion matrix", 'true')]
         for title, normalize in titles_options:
             disp = plot_confusion_matrix(classifier, X_test, y_test,
                                          display_labels=class_names,
                                          cmap=plt.cm.Blues,
                                          normalize=normalize)
             disp.ax_.set_title(title)

             print(title)
             print(disp.confusion_matrix)

         plt.show()
```

5. Confusion matrix

```
Confusion matrix, without normalization
[[228  58]
 [ 63 251]]
Normalized confusion matrix
[[0.7972028  0.2027972 ]
 [0.20063694 0.79936306]]
```


Confusion matrix, without normalization
True label
neg
pos
2.3e+02
2.5e+02
Predicted label


Normalized confusion matrix
True label
neg
pos
Predicted label