Understanding of the code

Part 1:

```
# utilities
import re
import pickle
import numpy as np
import pandas as pd

# plotting
import seaborn as sns
from wordcloud import WordCloud
import matplotlib.pyplot as plt

# nltk
from nltk.stem import WordNetLemmatizer

# sklearn
from sklearn.svm import LinearSVC
from sklearn.naive_bayes import BernoulliNB
from sklearn.linear_model import LogisticRegression

from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics import confusion_matrix, classification_report
```

Explanation:
This code imports necessary libraries and modules for text preprocessing, visualization, and machine learning. It sets up the environment for natural language processing (NLP) tasks, including data cleaning, visualization using word clouds, lemmatization, and classification using various machine learning algorithms such as LinearSVC, BernoulliNB, and LogisticRegression.

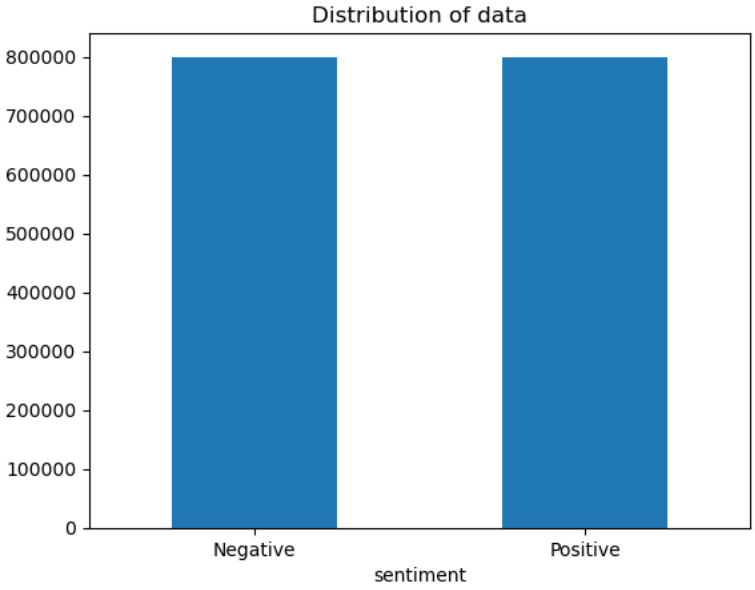
Part 2:

```
# Importing the dataset
DATASET_COLUMNS  = ["sentiment", "ids", "date", "flag", "user", "text"]
DATASET_ENCODING = "ISO-8859-1"
dataset = pd.read_csv('/Users/taha/Downloads/training.1600000.processed.noemoticon.csv',
                      encoding=DATASET_ENCODING , names=DATASET_COLUMNS)

# Removing the unnecessary columns.
dataset = dataset[['sentiment','text']]
# Replacing the values to ease understanding.
dataset['sentiment'] = dataset['sentiment'].replace(4,1)

# Plotting the distribution for dataset.
ax = dataset.groupby('sentiment').count().plot(kind='bar', title='Distribution of data',
                                               legend=False)
ax.set_xticklabels(['Negative','Positive'], rotation=0)

# Storing data in lists.
text, sentiment = list(dataset['text']), list(dataset['sentiment'])
```



Explanation:

This segment imports a dataset from a CSV file, sets its encoding, and renames its columns. It then extracts and processes the relevant columns (`sentiment` and `text`). Afterward, it visualizes the distribution of sentiment labels and stores the text and sentiment label data into separate lists for further processing.

Part 3:

```
# Defining dictionary containing all emojis with their meanings.
emojis = {':)': 'smile', ':-)': 'smile', ';d': 'wink', ':-E': 'vampire', ':(': 'sad',
          ':-(': 'sad', ':-<': 'sad', ':P': 'raspberry', ':O': 'surprised',
          ':-@': 'shocked', ':@': 'shocked',':-$': 'confused', ':\\': 'annoyed',
          ':#': 'mute', ':X': 'mute', ':^)': 'smile', ':-&': 'confused', '$_$': 'greedy',
          '@@': 'eyeroll', ':-!': 'confused', ':-D': 'smile', ':-0': 'yell', 'O.o': 'confused',
          '<(_-)>': 'robot', 'd[-_-]b': 'dj', ":'-)": 'sadsmile', ';)': 'wink',
          ';-)': 'wink', 'O:-)': 'angel','O*-)': 'angel','(:-D': 'gossip', '=^.^=': 'cat'}

## Defining set containing all stopwords in english.
stopwordlist = ['a', 'about', 'above', 'after', 'again', 'ain', 'all', 'am', 'an',
             'and','any','are', 'as', 'at', 'be', 'because', 'been', 'before',
             'being', 'below', 'between','both', 'by', 'can', 'd', 'did', 'do',
             'does', 'doing', 'down', 'during', 'each','few', 'for', 'from',
             'further', 'had', 'has', 'have', 'having', 'he', 'her', 'here',
             'hers', 'herself', 'him', 'himself', 'his', 'how', 'i', 'if', 'in',
             'into','is', 'it', 'its', 'itself', 'just', 'll', 'm', 'ma',
             'me', 'more', 'most','my', 'myself', 'now', 'o', 'of', 'on', 'once',
             'only', 'or', 'other', 'our', 'ours','ourselves', 'out', 'own', 're',
             's', 'same', 'she', "shes", 'should', "shouldve",'so', 'some', 'such',
             't', 'than', 'that', "thatll", 'the', 'their', 'theirs', 'them',
             'themselves', 'then', 'there', 'these', 'they', 'this', 'those',
             'through', 'to', 'too','under', 'until', 'up', 've', 'very', 'was',
             'we', 'were', 'what', 'when', 'where','which','while', 'who', 'whom',
             'why', 'will', 'with', 'won', 'y', 'you', "youd","youll", "youre",
            "youve", 'your', 'yours', 'yourself', 'yourselves']
```

Explanation:
This section defines two dictionaries: one containing emojis with their meanings and another containing a list of English stopwords. The emoji dictionary maps various emoticons to their corresponding meanings, while the stopword list contains common words typically removed from text data during natural language processing tasks like sentiment analysis or text classification.

Part 4:

```
def preprocess(textdata):
    processedText = []

    # Create Lemmatizer and Stemmer.
    wordLemm = WordNetLemmatizer()

    # Defining regex patterns.
    urlPattern        = r"((http://)[^ ]*|(https://)[^ ]*|( www\.)[^ ]*)"
    userPattern       = '@[^\s]+'
    alphaPattern      = "[^a-zA-Z0-9]"
    sequencePattern   = r"(.)\1\1+"
    seqReplacePattern = r"\1\1"

    for tweet in textdata:
        tweet = tweet.lower()

        # Replace all URls with 'URL'
        tweet = re.sub(urlPattern,' URL',tweet)
        # Replace all emojis.
        for emoji in emojis.keys():
            tweet = tweet.replace(emoji, "EMOJI" + emojis[emoji])
        # Replace @USERNAME to 'USER'.
        tweet = re.sub(userPattern,' USER', tweet)
        # Replace all non alphabets.
        tweet = re.sub(alphaPattern, " ", tweet)
        # Replace 3 or more consecutive letters by 2 letter.
        tweet = re.sub(sequencePattern, seqReplacePattern, tweet)

        tweetwords = ''
        for word in tweet.split():
            # Checking if the word is a stopword.
            #if word not in stopwordlist:
            if len(word)>1:
                # Lemmatizing the word.
                word = wordLemm.lemmatize(word)
                tweetwords += (word+' ')

        processedText.append(tweetwords)

    return processedText
```

Explanation:
This `preprocess` function takes a list of text data as input and performs several text preprocessing steps on each element of the list. These steps include converting text to lowercase, replacing URLs with 'URL', replacing emojis with their corresponding meanings, replacing user mentions with 'USER', removing non-alphabetic characters, replacing sequences of three or more identical characters with two occurrences of the character, and lemmatizing words. Finally, it returns the preprocessed text data as a list.

Part 5:

```
import time
t = time.time()
processedtext = preprocess(text)
print(f'Text Preprocessing complete.')
print(f'Time Taken: {round(time.time()-t)} seconds')
```

```
Text Preprocessing complete.
Time Taken: 73 seconds
```

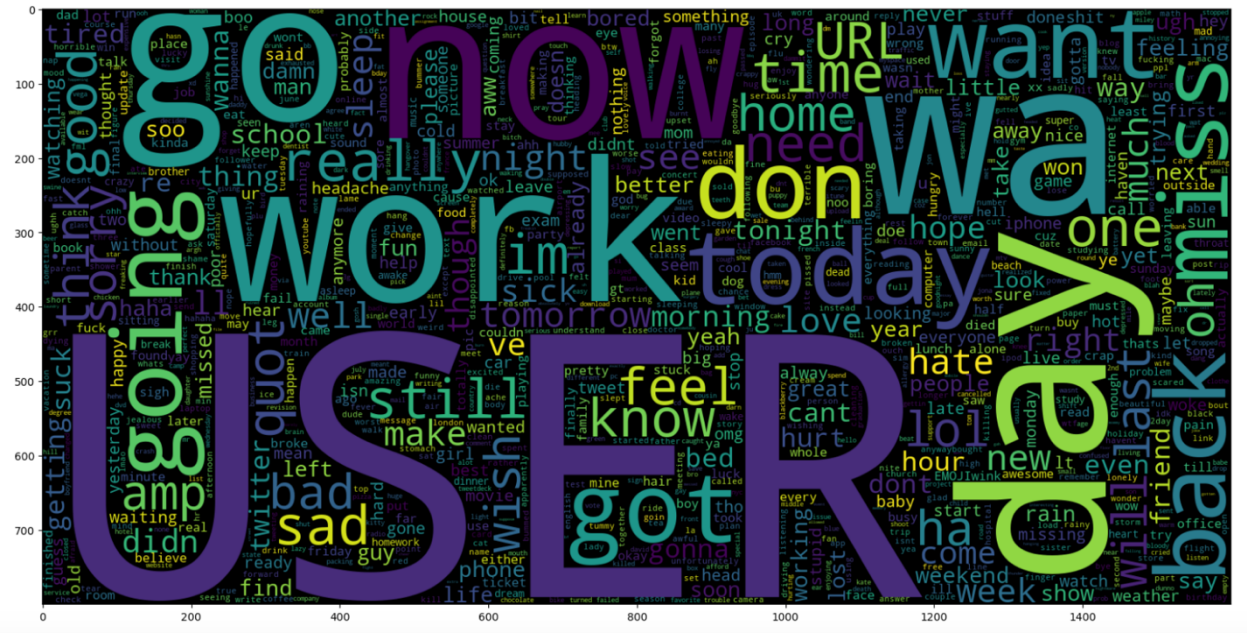
Explanation:

This code snippet measures the time taken to preprocess the text data using the `preprocess` function. It records the current time before preprocessing using `time.time()`, then calls the `preprocess` function to preprocess the text data stored in the variable `text`, and records the time again after preprocessing. Finally, it prints a message indicating that the text preprocessing is complete along with the time taken for the preprocessing operation.

Part 6:

```
data_neg = processedtext[:800000]
plt.figure(figsize = (20,20))
wc = WordCloud(max_words = 1000 , width = 1600 , height = 800,
               collocations=False).generate(" ".join(data_neg))
plt.imshow(wc)
```

```
<matplotlib.image.AxesImage at 0x7faa57480bb0>
```



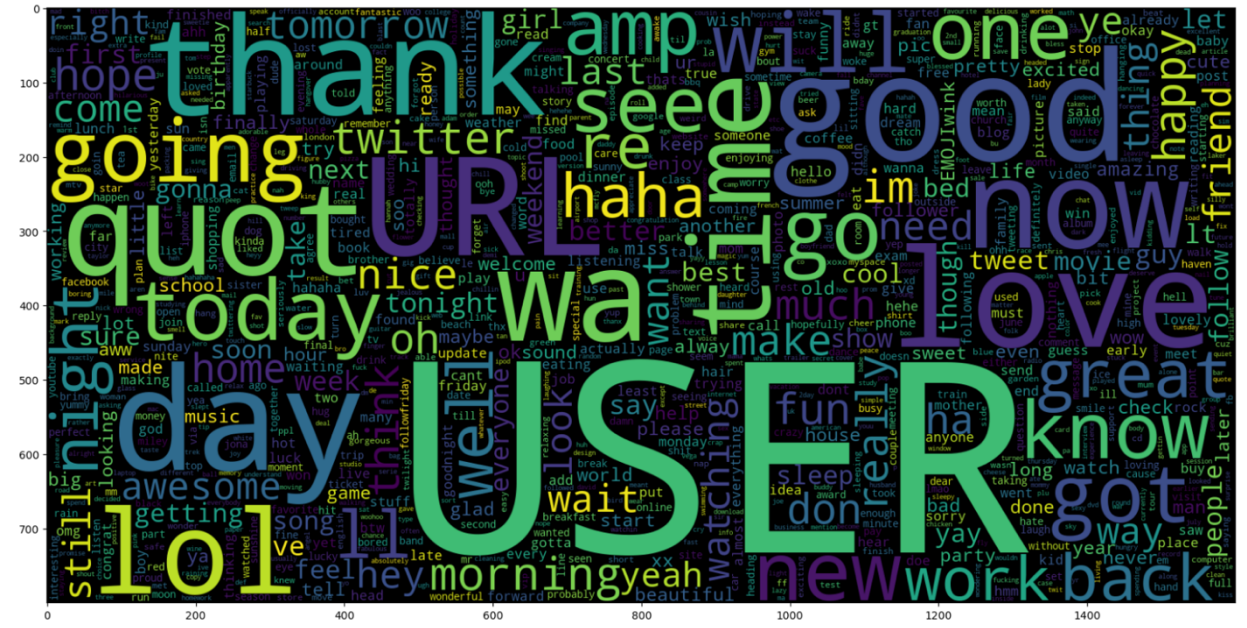
Explanation:

This code generates a word cloud visualization for the processed text data. It first selects the first 800,000 elements from the `processedtext` list and assigns them to the variable `data_neg`. Then, it creates a word cloud object `wc` using the WordCloud module, specifying parameters such as `max_words`, `width`, `height`, and `collocations`. Finally, it generates the word cloud using the processed text data in `data_neg` and displays it using `plt.imshow(wc)` within a figure with a specified size of 20x20 inches.

Part 7:

```
data_pos = processedtext[800000:]
wc = WordCloud(max_words = 1000 , width = 1600 , height = 800,
               collocations=False).generate(" ".join(data_pos))
plt.figure(figsize = (20,20))
plt.imshow(wc)
```

```
<matplotlib.image.AxesImage at 0x7faa55495a00>
```



Explanation:

This segment of the code generates another word cloud visualization, but this time for the positive sentiment portion of the processed text data. It selects the elements from index 800,000 to the end of the `processedtext` list and assigns them to the variable `data_pos`. Then, it creates a word cloud object `wc` with the same parameters as before. Finally, it generates the word cloud using the processed text data in `data_pos` and displays it within a new figure with the same specified size of 20x20 inches.

Part 8:

```
X_train, X_test, y_train, y_test = train_test_split(processedtext, sentiment,
                                                    test_size = 0.05, random_state = 0)
print(f'Data Split done.')
```

```
Data Split done.
```

Explanation:

This code splits the preprocessed text data (`processedtext`) and corresponding sentiment labels (`sentiment`) into training and testing sets using the `train_test_split` function from scikit-learn. It allocates 95% of the data for training (`X_train` and `y_train`) and 5% for testing (`X_test` and `y_test`). The `random_state` parameter ensures reproducibility of the split. Finally, it prints a message indicating that the data split is done.

Part 9:

```
vectoriser = TfidfVectorizer(ngram_range=(1,2), max_features=500000)
vectoriser.fit(X_train)
print(f'Vectoriser fitted.')
print('No. of feature_words: ', len(vectoriser.get_feature_names()))
```

```
Vectoriser fitted.
No. of feature_words:  500000
```

Explanation:

This code creates a TF-IDF vectorizer (`vectoriser`) using scikit-learn's `TfidfVectorizer` class. It is configured to consider unigrams and bigrams (`ngram_range=(1,2)`) and to limit the number of features to 500,000 (`max_features=500000`). Then, it fits the vectorizer to the training data (`X_train`) using the `fit` method. Finally, it prints a message confirming that the vectorizer is fitted and displays the number of feature words extracted from the training data.

Part 10:

```
X_train = vectoriser.transform(X_train)
X_test  = vectoriser.transform(X_test)
print(f'Data Transformed.')
```

```
Data Transformed.
```

Explanation:

This code transforms the text data in both the training set (`X_train`) and the test set (`X_test`) into TF-IDF representations using the previously fitted `vectoriser`. The `transform` method is used to convert the raw text data into TF-IDF feature vectors. Finally, it prints a message confirming that the data transformation is complete.

Part 11:

```
def model_Evaluate(model):

    # Predict values for Test dataset
    y_pred = model.predict(X_test)

    # Print the evaluation metrics for the dataset.
    print(classification_report(y_test, y_pred))

    # Compute and plot the Confusion matrix
    cf_matrix = confusion_matrix(y_test, y_pred)

    categories  = ['Negative','Positive']
    group_names = ['True Neg','False Pos', 'False Neg','True Pos']
    group_percentages = ['{0:.2%}'.format(value) for value in cf_matrix.flatten() / np.sum(cf_matrix)]

    labels = [f'{v1}\n{v2}' for v1, v2 in zip(group_names,group_percentages)]
    labels = np.asarray(labels).reshape(2,2)

    sns.heatmap(cf_matrix, annot = labels, cmap = 'Blues',fmt = '',
                xticklabels = categories, yticklabels = categories)

    plt.xlabel("Predicted values", fontdict = {'size':14}, labelpad = 10)
    plt.ylabel("Actual values"   , fontdict = {'size':14}, labelpad = 10)
    plt.title ("Confusion Matrix", fontdict = {'size':18}, pad = 20)
```

Explanation:
This function evaluates a machine learning model using the test dataset. It predicts the values for the test dataset using the provided model, prints the classification report containing evaluation metrics such as precision, recall, and F1-score, and plots the confusion matrix to visualize the model's performance. The confusion matrix shows the number of true positives, true negatives, false positives, and false negatives.

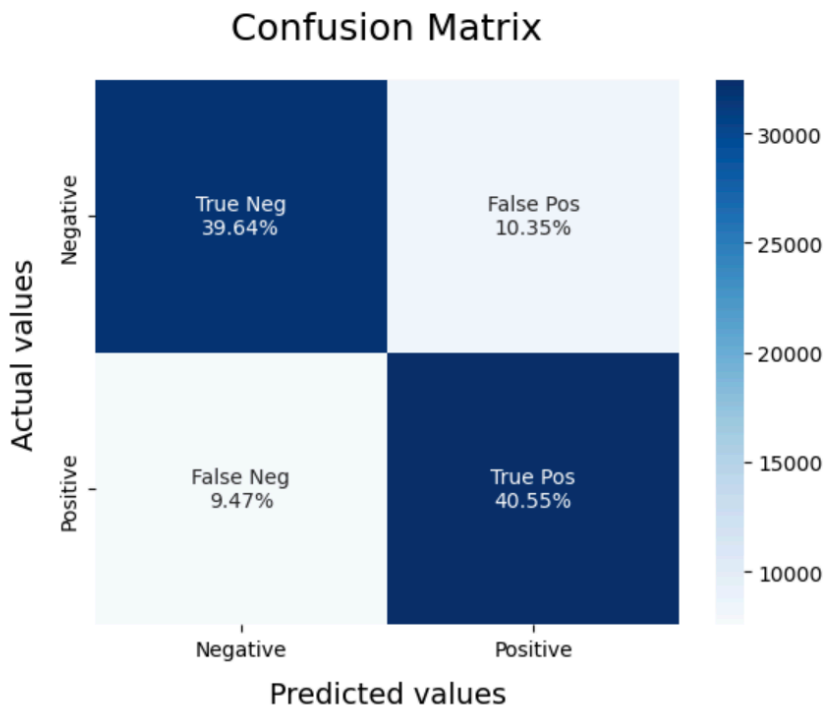
Part 12:

```
[12]: BNBmodel = BernoulliNB(alpha = 2)
      BNBmodel.fit(X_train, y_train)
      model_Evaluate(BNBmodel)
```

```
              precision    recall  f1-score   support

           0       0.81      0.79      0.80     39989
           1       0.80      0.81      0.80     40011

    accuracy                           0.80     80000
   macro avg       0.80      0.80      0.80     80000
weighted avg       0.80      0.80      0.80     80000
```



Explanation:
This code initializes a Bernoulli Naive Bayes model with a smoothing parameter (`alpha`) set to 2, fits the model to the training data (`X_train` and `y_train`), and then evaluates the

model using the `model_Evaluate` function. The function prints the classification report and visualizes the confusion matrix for the model's predictions on the test dataset.

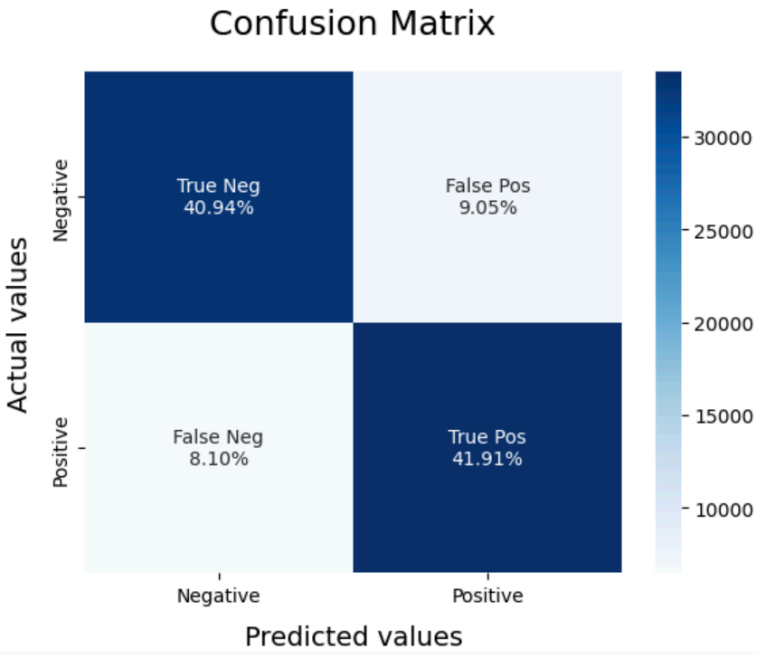
Part 13:

```
[13]: LRmodel = LogisticRegression(C = 2, max_iter = 1000, n_jobs=-1)
      LRmodel.fit(X_train, y_train)
      model_Evaluate(LRmodel)
```

```
              precision    recall  f1-score   support

           0       0.83      0.82      0.83     39989
           1       0.82      0.84      0.83     40011

    accuracy                           0.83     80000
   macro avg       0.83      0.83      0.83     80000
weighted avg       0.83      0.83      0.83     80000
```



Explanation:

This snippet creates a Logistic Regression model with regularization strength parameter `C` set to 2, maximum number of iterations `max_iter` set to 1000, and uses all available processors for parallel computing (`n_jobs=-1`). Then it trains the model on the training data (`X_train` and `y_train`), and evaluates the model using the `model_Evaluate` function, which prints the classification report and visualizes the confusion matrix for the model's predictions on the test dataset.

Part 14:

```
file = open('vectoriser-ngram-(1,2).pickle','wb')
pickle.dump(vectoriser, file)
file.close()

file = open('Sentiment-LR.pickle','wb')
pickle.dump(LRmodel, file)
file.close()

file = open('Sentiment-BNB.pickle','wb')
pickle.dump(BNBmodel, file)
file.close()
```

Explanation:
These lines save the trained vectorizer (`vectoriser`) and the trained models (`LRmodel` and `BNBmodel`) using pickle serialization. They are saved as binary files with the extensions ".pickle". This allows you to later load these objects and use them without retraining.

Part 15:

```
def load_models():
    '''
    Replace '..path/' by the path of the saved models.
    '''

    # Load the vectoriser.
    file = open('..path/vectoriser-ngram-(1,2).pickle', 'rb')
    vectoriser = pickle.load(file)
    file.close()
    # Load the LR Model.
    file = open('..path/Sentiment-LRv1.pickle', 'rb')
    LRmodel = pickle.load(file)
    file.close()

    return vectoriser, LRmodel

def predict(vectoriser, model, text):
    # Predict the sentiment
    textdata = vectoriser.transform(preprocess(text))
    sentiment = model.predict(textdata)

    # Make a list of text with sentiment.
    data = []
    for text, pred in zip(text, sentiment):
        data.append((text,pred))

    # Convert the list into a Pandas DataFrame.
    df = pd.DataFrame(data, columns = ['text','sentiment'])
    df = df.replace([0,1], ["Negative","Positive"])
    return df

if __name__=="__main__":
    # Loading the models.
    #vectoriser, LRmodel = load_models()

    # Text to classify should be in a list.
    text = ["I hate twitter",
            "May the Force be with you.",
            "Mr. Stark, I don't feel so good"]

    df = predict(vectoriser, LRmodel, text)
    print(df.head())
```

```
                              text sentiment
0                  I hate twitter  Negative
1      May the Force be with you.  Positive
2  Mr. Stark, I don't feel so good  Negative
```

Explanation:
This code defines two functions: `load_models()` and `predict()`.

- `load_models()` loads the saved vectorizer and logistic regression model from the specified file paths using pickle deserialization.

- `predict()` takes the loaded vectorizer and model along with a list of text as input, preprocesses the text, predicts the sentiment using the model, and returns a DataFrame containing the text and predicted sentiment.

In the `if __name__ == "__main__":` block, it demonstrates how to use these functions by loading the models, predicting the sentiment for a list of sample texts, and printing the DataFrame containing the results. However, the `vectoriser` and `LRmodel` variables need to be uncommented for this code to run properly.