Installing Dependencies

In [40]:
```
#pip install -U scikit-learn
```

In [41]:
```
#pip install -U wordcloud
```

In [42]:
```
#pip install clean-text
```

Importing Dependencies

In [4]:
```
#Importing libraries

#import 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
import nltk
from nltk.stem import WordNetLemmatizer
from nltk.corpus import stopwords

#sklearn
from sklearn.svm import LinearSVC
from sklearn.linear_model import LogisticRegression as LR

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

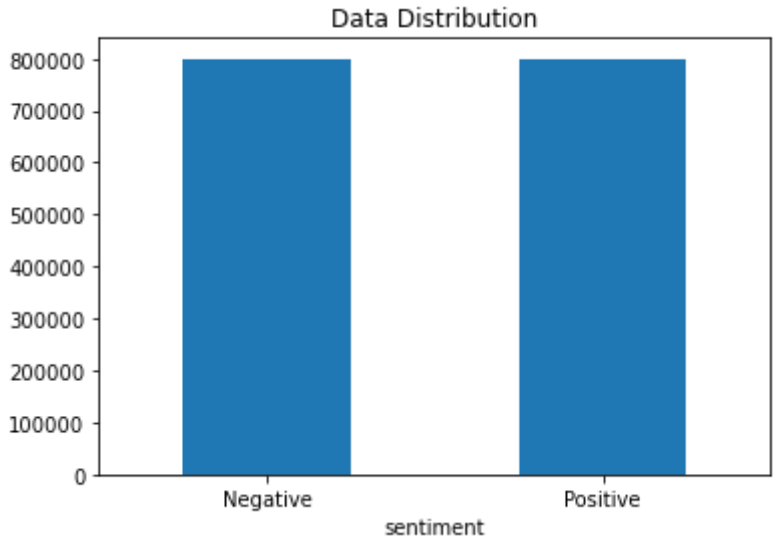
In [5]:
```
from cleantext import clean
```

```
Since the GPL-licensed package `unidecode` is not installed, using Python's `uni
codedata` package which yields worse results.
```

Data Preprocessing

In [6]:
```
#importing dataset


DATASET_COLUMNS = ["sentiment","ids","date","flag","user","text"]
DATASET_ENCODING ="ISO-8859-1"
dataset=pd.read_csv('tweets_data.csv',encoding=DATASET_ENCODING,names=DATASET_CO


#Removing the unnecessary columns
dataset=dataset[['sentiment','text']]
```

```
#Removing 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='Data Distributi

ax.set_xticklabels(['Negative','Positive'],rotation=0)

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



In [7]:
```
dataset.head(5)
```

Out[7]:

	sentiment	text
0	0	@switchfoot http://twitpic.com/2y1zl - Awww, t...
1	0	is upset that he can't update his Facebook by ...
2	0	@Kenichan I dived many times for the ball. Man...
3	0	my whole body feels itchy and like its on fire
4	0	@nationwideclass no, it's not behaving at all....

In [8]:
```
dataset["sentiment"].value_counts()
```

Out[8]:
```
1    800000
0    800000
Name: sentiment, dtype: int64
```

In [9]:
```
dataset['text'][0]
```

Out[9]:
```
"@switchfoot http://twitpic.com/2y1zl - Awww, that's a bummer.  You shoulda got
David Carr of Third Day to do it. ;D"
```

In [10]:
```
#Defining dictionary containing all emojis with their meanings.

emojis={":-)":"smiley",":-]":"smiley",":-3":"smiley",":->":"smiley",
        "8-)":"smiley",
        ":-}":"smiley",
        ":)":"smiley",
        ":]":"smiley",
```

```
        ":3":"smiley",
        ":>":"smiley",
        "8)":"smiley",
        ":}":"smiley",
        ":o)":"smiley",
        ":c)":"smiley",
        ":^)":"smiley",
        "=]":"smiley",
        "=)":"smiley",
        ":-))":"smiley",
        ":-D":"smiley",
        "8-D":"smiley",
        "x-D":"smiley",
        "X-D":"smiley",
        ":D":"smiley",
        "8D":"smiley",
        "xD":"smiley",
        "XD":"smiley",
        ":-(":"sad",
        ":-c":"sad",
        ":-<":"sad",
        ":-[":"sad",
        ":(":"sad",
        ":c":"sad",
        ":<":"sad",
        ":[":"sad",
        ":-||":"sad",
        ">:[":"sad",
        ":{":"sad",
        ":@":"sad",
        ">:(":"sad",
        ":'-(":"sad",
        ":'(":"sad",
        ":-P":"playful",
        "X-P":"playful",
        "x-p":"playful",
        ":-p":"playful",
        ":-Þ":"playful",
        ":-þ":"playful",
        ":-b":"playful",
        ":P":"playful",
        "XP":"playful",
        "xp":"playful",
        ":p":"playful",
        ":Þ":"playful",
        ":þ":"playful",
        ":b":"playful",
```

In [11]:
```
#defining all words containg stopwords

stopwords=stopwords.words('english');
```

In [12]:
```
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=re.sub(userPattern,' USER',tweet)
        #Replace @USERNAME to 'USER'
        tweet=re.sub(alphaPattern," ",tweet)
        #Replace 3 or more consecutive letter by 2 letter
        tweet=re.sub(sequencePattern, SeqReplacePattern, tweet)

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

In [13]:
```
import time
t=time.time()
processedtext=preprocess(text)
print(f'text Preprocessing complete.')
print(f'time taken {round(time.time()-t)} seconds')
```

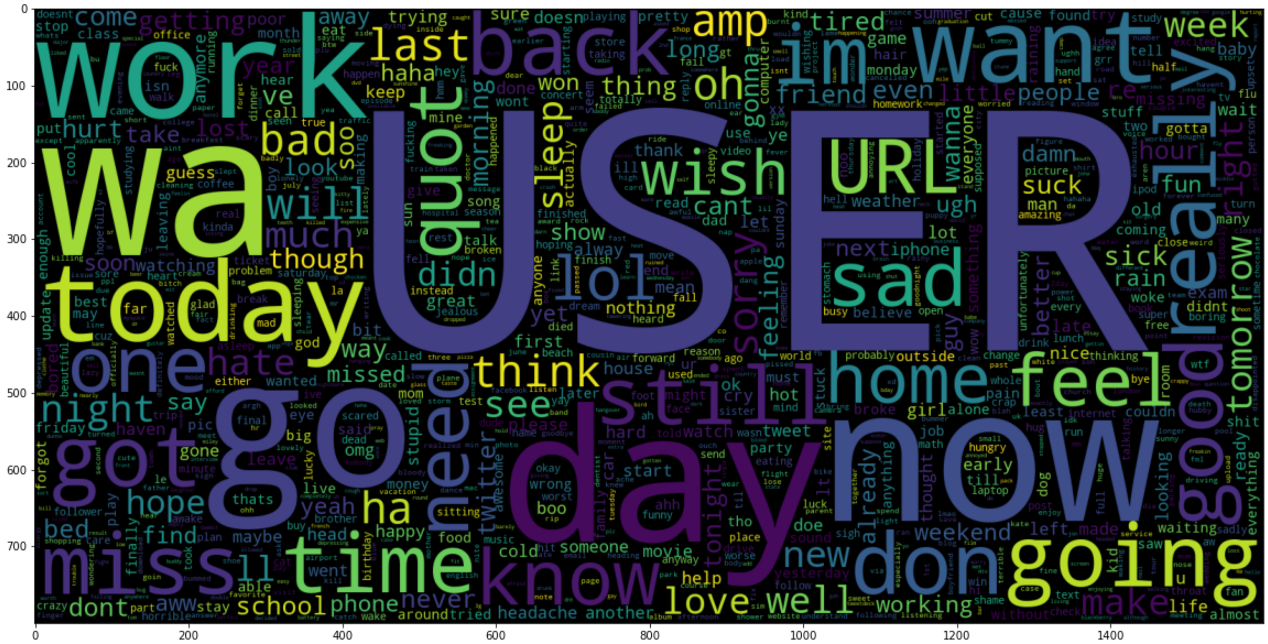
```
text Preprocessing complete.
time taken 151 seconds
```

Analysing the data

Word-Cloud for Negative tweets

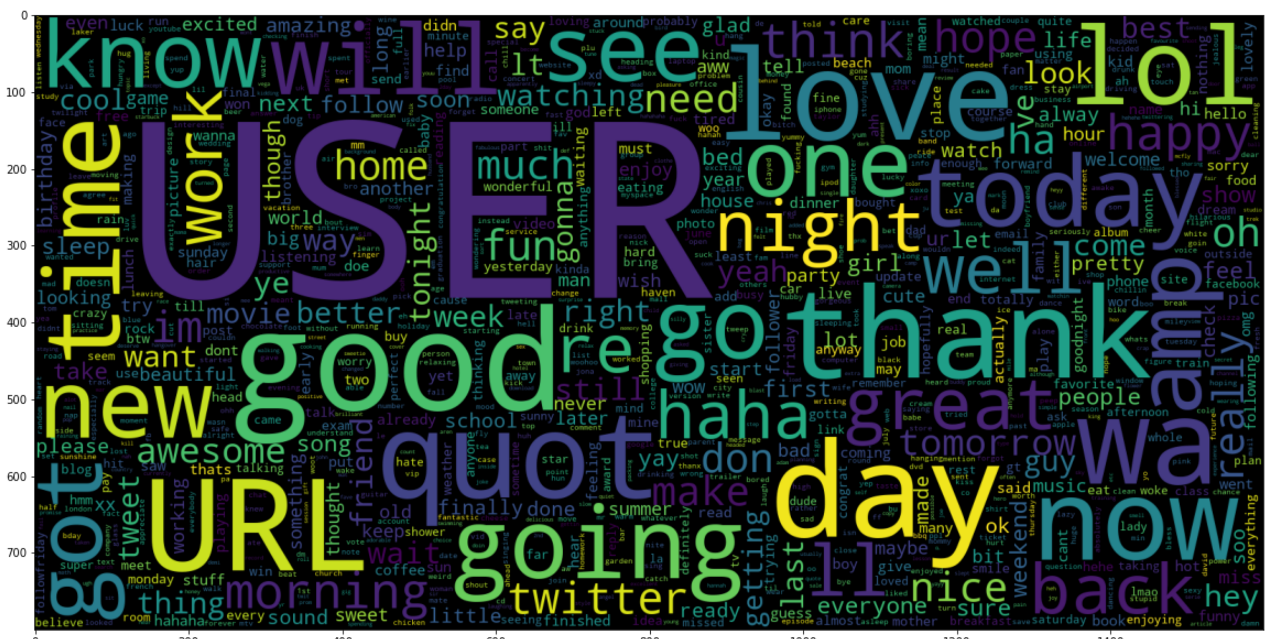
In [14]:
```
data_neg=processedtext[:800000]
plt.figure(figsize = (20,20))
wc= WordCloud(max_words = 1000, width = 1600, height = 800, collocations=False).
plt.imshow(wc)
```

Out[14]:
```
<matplotlib.image.AxesImage at 0x7fe18e2771c0>
```



In [15]:
```
data_pos=processedtext[800000:]

wc= WordCloud(max_words = 1000, width = 1600, height = 800, collocations=False).
plt.figure(figsize = (20,20))
plt.imshow(wc)
```

Out[15]: <matplotlib.image.AxesImage at 0x7fe18e1b2370>



In [16]:
```
len(processedtext)/2
```

Out[16]: 800000.0

Splitting the Data

In [17]:
```
X_train, X_test, y_train, y_test = train_test_split(processedtext, sentiment, te
```

TF-IDF Vectoriser

In [18]:
```
vectorizer= TfidfVectorizer(ngram_range=(1,2), max_features=500000)
#TfidfVectorizer
vectorizer.fit(X_train)
print(f'vectorizer fitted.')
print('No. of feature_words: ',len(vectorizer.get_feature_names_out()))
```

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

Transforming the dataset

In [19]:
```
X_train = vectorizer.transform(X_train)
X_test = vectorizer.transform(X_test)
print("Data Transformed")
```

```
Data Transformed
```

Creating and Evaluating Models

We are creating two different type of models for our analysis problem

1. Linear Support VecoterClassification(LinearSVC)
2. Logistic Regression(LR)

since our dataset is not skewed. i.e. it has equal number of Positive and Negative predictions. We are choosing Accuracy as our evalution metric. Furthermore, we plotiing the confusion matrix to get an understanding of how our model is performing on both classification types

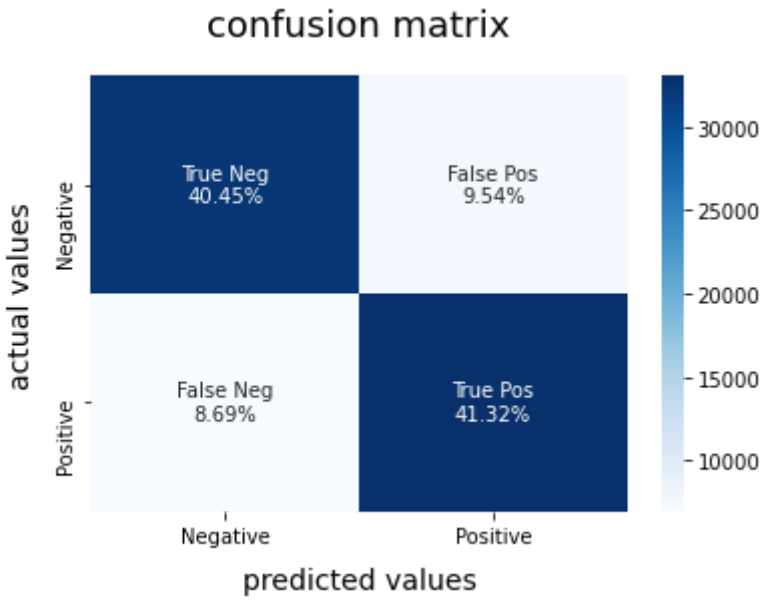
Evaluation Model Function

In [28]:
```
def model_Evaluate(model):
    #Predict value for test dataset
    y_pred = model.predict(X_test)

    #print evaluation matrix 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()

    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=categor

    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)
```

LinearSVC model

In [29]:
```
#SVCmodel = LinearSVC()
#SVCmodel.fit(X_train, y_train)
model_Evaluate(SVCmodel)
```

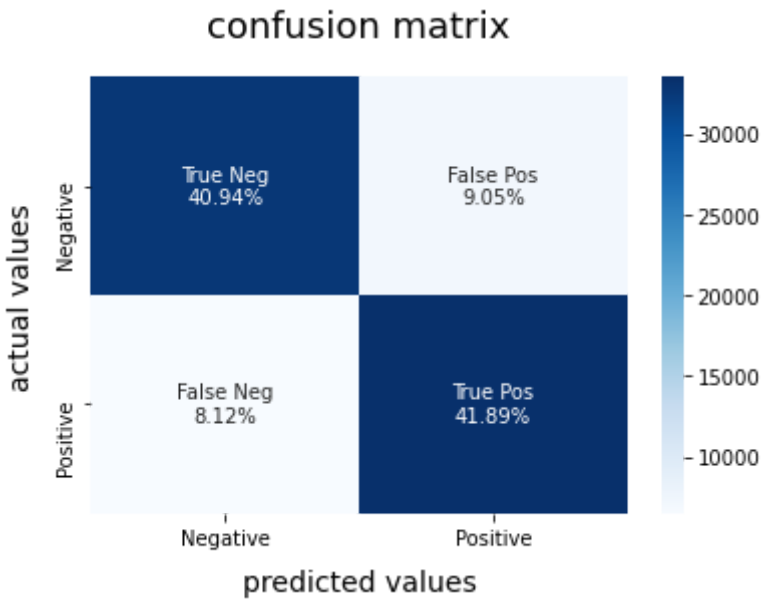
	precision	recall	f1-score	support
0	0.82	0.81	0.82	39989
1	0.81	0.83	0.82	40011
accuracy			0.82	80000
macro avg	0.82	0.82	0.82	80000
weighted avg	0.82	0.82	0.82	80000



Logistic Regression Model

In [31]:
```
LRmodel = LR(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



Saving the Models

In [32]:
```
file=open('vectorizer-ngram-(1,2).pickle','wb')
pickle.dump(vectorizer, file)
file.close()

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

Using the Model

In [39]:
```
def load_models():

    #load the vectorizer.
    file=open('vectorizer-ngram-(1,2).pickle','rb')
    vectorizer =pickle.load(file)
    file.close()

    #Load LR model
    file=open('Sentiment-LR.pickle','rb')
    LRmodel =pickle.load(file)
    file.close()

    return vectorizer, LRmodel

def predict(vectorizer, model, text):
    #predict the sentiment
    textdata=vectorizer.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
    vectorizer, LRmodel = load_models()

    #text to classify should be in a list
    text=["I fell like shit",
          "I am not feeling good today",
          "Mr.Stark, lets celebrate today"]

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

In [ ]: