Introduction :

The purpose of this report is to provide a comprehensive analysis and evaluation of the code available at the following link: https://www.kaggle.com/code/monawankhede/final-data-modeling-on-sentiment-analysis. The code focuses on sentiment analysis, a vital task in natural language processing (NLP), and aims to build an effective sentiment analysis model using a dataset from Kaggle.

Overview:

The code presents a step-by-step approach to building a sentiment analysis model using Python and several popular libraries such as pandas, scikit-learn, and NLTK. The dataset used in the code contains labeled tweets, classified as positive or negative sentiment, which serves as the foundation for training and evaluating the model.

Code Review:

a . Preprocessing:

1. The code begins with the necessary import statements to bring in the required libraries.
2. It proceeds with data pre-processing steps, including data loading, cleaning, and exploration.
3. The text data undergoes various cleaning operations, such as removing special characters, URLs, and stopwords.
4. Tokenization and stemming techniques are employed to transform text into a suitable format for analysis.
5. Exploratory data analysis (EDA) is performed to gain insights into the dataset's characteristics.

Dataset Context

This is the sentiment140 dataset. It contains around 1,600,000 tweets extracted using the twitter API. The tweets have been annotated (0 = negative, 4 = positive) and they can be used to detect sentiment.

Dataset Content

It contains the following 6 fields:

target : The polarity of the tweet (0 = negative, 2 = neutral, 4 = positive)

ids : The id of the tweet (2087)

date : The date of the tweet (Sat May 16 23:58:44 UTC 2009)

flag : The query (lyx). If there is no query, then this value is NO_QUERY.

user : The user that tweeted (robotickilldozr)

text : The text of the tweet (Lyx is cool)

Step 1 : Import Libraries

```
# import libraries
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import nltk
from wordcloud import WordCloud
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer # for creating our Bag of words
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import BernoulliNB
from sklearn.svm import LinearSVC
from sklearn.metrics import accuracy_score, classification_report
from sklearn.metrics import  confusion_matrix,ConfusionMatrixDisplay
from matplotlib import style
style.use('ggplot')
import warnings
warnings.filterwarnings('ignore')

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

Step 2 : Read Data and set header names

```
# Read data
data = pd.read_csv('/kaggle/input/sentiment140/training.1600000.processed.noemoticon.csv',
                     encoding='latin-1')
data.head()
```

Output :

Out[2]:

	0	1467810369	Mon Apr 06 22:19:45 PDT 2009	NO_QUERY	_TheSpecialOne_	@switchfoot http://twitpic.com/2y1zl - Awww, that's a bummer. You shoulda got David Carr of Third Day to do it. ;D
0	0	1467810672	Mon Apr 06 22:19:49 PDT 2009	NO_QUERY	scotthamilton	is upset that he can't update his Facebook by ...
1	0	1467810917	Mon Apr 06 22:19:53 PDT 2009	NO_QUERY	mattycus	@Kenichan I dived many times for the ball. Man...
2	0	1467811184	Mon Apr 06 22:19:57 PDT 2009	NO_QUERY	ElleCTF	my whole body feels itchy and like its on fire
3	0	1467811193	Mon Apr 06 22:19:57 PDT 2009	NO_QUERY	Karoli	@nationwideclass no, it's not behaving at all....
4	0	1467811372	Mon Apr 06 22:20:00 PDT 2009	NO_QUERY	joy_wolf	@Kwesidei not the whole crew

```
# set columns header
data.columns = ['Target','ids','Date','Flag','User','Text']
data
```

Out[3]:

	Target	ids	Date	Flag	User	Text
0	0	1467810672	Mon Apr 06 22:19:49 PDT 2009	NO_QUERY	scotthamilton	is upset that he can't update his Facebook by ...
1	0	1467810917	Mon Apr 06 22:19:53 PDT 2009	NO_QUERY	mattycus	@Kenichan I dived many times for the ball. Man...
2	0	1467811184	Mon Apr 06 22:19:57 PDT 2009	NO_QUERY	ElleCTF	my whole body feels itchy and like its on fire
3	0	1467811193	Mon Apr 06 22:19:57 PDT 2009	NO_QUERY	Karoli	@nationwideclass no, it's not behaving at all....
4	0	1467811372	Mon Apr 06 22:20:00 PDT 2009	NO_QUERY	joy_wolf	@Kwesidei not the whole crew

Step 3 : Exploratory Data Analysis

n [4]:

```
print("The shape of Data is : ",data.shape)
```

```
The shape of Data is :  (1599999, 6)
```

In [5]:

```
# count of data
data.count()
```

Out[5]:

```
Target    1599999
ids       1599999
Date      1599999
Flag      1599999
User      1599999
Text      1599999
dtype: int64
```

In [6]:

```
# Check data having missing values or not
data.isnull().sum()
```

Out[6]:

```
Target    0
ids       0
Date      0
Flag      0
User      0
Text      0
dtype: int64
```

There is no missing values in Data so I can perform Analysis

In [7]:
```
# check info of data
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1599999 entries, 0 to 1599998
Data columns (total 6 columns):
 #   Column  Non-Null Count    Dtype
---  ------  --------------    -----
 0   Target  1599999 non-null  int64
 1   ids     1599999 non-null  int64
 2   Date    1599999 non-null  object
 3   Flag    1599999 non-null  object
 4   User    1599999 non-null  object
 5   Text    1599999 non-null  object
dtypes: int64(2), object(4)
memory usage: 73.2+ MB
```

From this I get to know there are no missing values as well as 2 integer and 4 object datatype but date should be in datetime format so I will use pd.datetime()

In [8]:
```
data['Date']=  pd.to_datetime(data['Date'], infer_datetime_format=True)
```

In [9]:
```
# again check info to see changes are done or not
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1599999 entries, 0 to 1599998
Data columns (total 6 columns):
 #   Column  Non-Null Count    Dtype
---  ------  --------------    -----
 0   Target  1599999 non-null  int64
 1   ids     1599999 non-null  int64
 2   Date    1599999 non-null  datetime64[ns]
 3   Flag    1599999 non-null  object
 4   User    1599999 non-null  object
 5   Text    1599999 non-null  object
dtypes: datetime64[ns](1), int64(2), object(3)
memory usage: 73.2+ MB
```

In [10]:
```
# lets set Target 4 as positive tweets and 0 as negative
data["Target"] = data["Target"].replace(4,"Positive")
data["Target"] = data["Target"].replace(0,"Negative")
```

In [11]:
```
# see final data after changes
data.head()
```

Out[11]:

	Target	ids	Date	Flag	User	Text
0	Negative	1467810672	2009-04-06 22:19:49	NO_QUERY	scotthamilton	is upset that he can't update his Facebook by ...
1	Negative	1467810917	2009-04-06 22:19:53	NO_QUERY	mattycus	@Kenichan I dived many times for the ball. Man...
2	Negative	1467811184	2009-04-06 22:19:57	NO_QUERY	ElleCTF	my whole body feels itchy and like its on fire
3	Negative	1467811193	2009-04-06 22:19:57	NO_QUERY	Karoli	@nationwideclass no, it's not behaving at all....
4	Negative	1467811372	2009-04-06 22:20:00	NO_QUERY	joy_wolf	@Kwesidei not the whole crew

Here I can see date object is converted into datetime format

Step 4 : Data Pre-processing

In [12]:
```
# 4.1: Selecting the text and Target column for our further analysis
df  = data[["Text","Target"]]
df.head()
```

Out[12]:

	Text	Target
0	is upset that he can't update his Facebook by ...	Negative
1	@Kenichan I dived many times for the ball. Man...	Negative
2	my whole body feels itchy and like its on fire	Negative
3	@nationwideclass no, it's not behaving at all....	Negative
4	@Kwesidei not the whole crew	Negative

In [13]:
```
# lets set Target 4 as positive tweets and 0 as negative
data["Target"] = data["Target"].replace("Positive",4)
data["Target"] = data["Target"].replace("Negative",0)
```

In [14]:
```
# 4.2: Separating positive and negative tweets

data_pos = data[data['Target'] == 4]
data_neg = data[data['Target'] == 0]
```

In [16]:
```
# 4.3 : Taking one-fourth of the data so we can run it on our machine easily

data_pos = data_pos.iloc[:int(200000)]
data_neg = data_neg.iloc[:int(200000)]
```

In [17]:
```
# 4.4 : Combining positive and negative tweets

dataset = pd.concat([data_pos, data_neg])
dataset.head()
```

Output

Out[17]:

	Target	ids	Date	Flag	User	Text
799999	4	1467822272	2009-04-06 22:22:45	NO_QUERY	ersle	I LOVE @Health4UandPets u guys r the best!!
800000	4	1467822273	2009-04-06 22:22:45	NO_QUERY	becca210	im meeting up with one of my besties tonight! ...
800001	4	1467822283	2009-04-06 22:22:46	NO_QUERY	Wingman29	@DaRealSunisaKim Thanks for the Twitter add, S...
800002	4	1467822287	2009-04-06 22:22:46	NO_QUERY	katarinka	Being sick can be really cheap when it hurts t...
800003	4	1467822293	2009-04-06 22:22:46	NO_QUERY	_EmilyYoung	@LovesBrooklyn2 he has that effect on everyone

Step 6 : Text Pre-processing steps :

- Step 1 -> Converting everything into lower or an upper cases.
- Step 2 -> Remove all the special characters (such as @, #, !, numbers).

- Step 3 -> Remove the stop words.
- Step 4 -> Remove URL's
- Step 5 -> Remove

In [18]:
```
# Step 6.1 - text pre-processing
dataset['Text'] = dataset['Text'].str.lower().str.replace('[^a-z\']', ' ')
```

In [19]:
```
# Step 6.2 : Downloading stopwords
import nltk

nltk.download('stopwords')

from nltk.corpus import stopwords

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

Define a fuction to remove stopwords and further cleaning of text

```
def sw(x):
    x = [word for word in x.split() if word not in stop]
    # Remove Stopwords
    return ' '.join (x)
    return re.sub('((www.[^s]+)|(https?://[^s]+))',' ',data)
    # Cleaning and removing URLs
    dataset['Text'] = dataset['Text'].apply(lambda x: sw(x))
    return re.sub('[0-9]+', '', data)
    # Cleaning and removing numeric numbers
    dataset['Text'] = dataset['Text'].apply(lambda x: sw(x))
```

In [21]:
```
# Step 6.4  Applying our user defined function on text column and then storin
g result
# in same column
dataset['Text'] = dataset['Text'].apply(sw)
```

In [22]:
```
dataset['Text'] = dataset['Text'].apply(lambda x:x.split())
```

In [23]:
```
# Lets check clean text
dataset["Text"]
```

Out[23]:
```
799999          [love, health, uandpets, u, guys, r, best]
800000    [im, meeting, one, besties, tonight, cant, wai...
800001    [darealsunisakim, thanks, twitter, add, sunisa...
800002    [sick, really, cheap, hurts, much, eat, real, ...
```

Applying Stemming :

Stemming is a technique used in natural language processing (NLP) to reduce words to their base or root form, known as a stem. It aims to simplify text analysis by grouping together different forms of the same word.

In the process of stemming, suffixes are removed from words to obtain their base form. For example, the word "running" would be stemmed to "run," and "cats" would be stemmed to "cat." This helps in reducing the dimensionality of text data and allows algorithms to focus on the core meaning of words rather than their specific variations.

In [24]:
```
# 6.5 applying stemming

import nltk
st = nltk.PorterStemmer()
def stemming_on_text(data):
    text = [st.stem(word) for word in data]
    return data
```

In [25]:
```
dataset['Text']= dataset['Text'].apply(lambda x: stemming_on_text(x))
```

In [26]:
```
dataset["Text"][20]
```

Out[26]:
```
['one',
 'friend',
```

In [28]:

```
#6.7 import worldcloud to see maximum words in tweet

from wordcloud import WordCloud

data_neg = df['Text'][400000:]
plt.figure(figsize = (15,8))
wc_neg = WordCloud(max_words = 1000 , width = 1600 , height = 800,
               collocations=False).generate(" ".join(data_neg))
plt.imshow(wc_neg)
```

Out[28]:

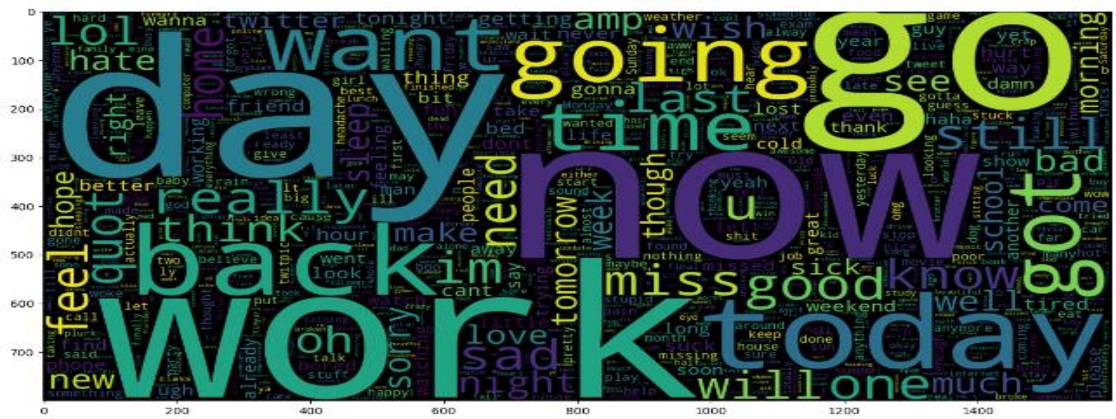
```
<matplotlib.image.AxesImage at 0x7adfa7b81570>
```

In [29]:

```
# 6.8: Plot a cloud of words for positive tweets

data_pos = df['Text'][:400000]
wc_pos = WordCloud(max_words = 1000 , width = 1600 , height = 800,
               collocations=False).generate(" ".join(data_pos))
plt.figure(figsize = (15,8))
plt.imshow(wc_pos)
```

```
<matplotlib.image.AxesImage at 0x7adf904b60b0>
```



Step-7: Splitting Our Data Into Train and Test Subsets

In [30]:

```
from sklearn.model_selection import train_test_split
# Split data into features and labels
X = data['Text']
y = data['Target']

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, ra
ndom_state=42)
```

b. Feature Extraction:

1. The code utilizes the TF-IDF (Term Frequency-Inverse Document Frequency) vectorization technique to convert the preprocessed text into numerical features.
2. The TF-IDF approach calculates the importance of words in a document by considering their frequency and rarity across the entire dataset.

Step-8 : Transforming the Dataset Using TF-IDF Vectorizer

[31]:

```
# using sklearn import  TfidfVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer # for creating o
ur Bag of words

# Create TF-IDF vectorizer
vectorizer = TfidfVectorizer()
X_train_tfidf = vectorizer.fit_transform(X_train)
X_test_tfidf = vectorizer.transform(X_test)
```

c. Model Training and Evaluation:

1. The code employs several machine learning algorithms, including Logistic Regression, Naive Bayes, and Support Vector Machines (SVM), to train and evaluate the sentiment analysis model.
2. The dataset is split into training and testing sets, and the models are trained using the training set.
3. The accuracy, precision, recall, and F1-score metrics are used to evaluate the performance of each model.
4. The code concludes by selecting the best-performing model based on evaluation results.

Step-9: Model Building

Model 1 :logisticRegresion

In [33]:
```
# Build model 1 :logisticRegresion

from sklearn.linear_model import LogisticRegression

from sklearn.metrics import accuracy_score, classification_report
from sklearn.metrics import  confusion_matrix,ConfusionMatrixDisplay
```

In [34]:
```
#create model object
logreg = LogisticRegression()

#fit model object
logreg.fit(X_train_tfidf, y_train)

#Predict model object
logreg_pred = logreg.predict(X_test_tfidf)

#check accuracy
logreg_acc = accuracy_score(logreg_pred, y_test)

#Print accuracy
print("Test accuracy: {:.2f}%".format(logreg_acc*100))

print("confusion matrix : ")
confusion_matrix(logreg_pred, y_test)
```

```
Test accuracy: 80.22%
confusion matrix :
```

Out[34]:
```
array([[126436,  30243],
       [ 33058, 130263]])
```

In [35]:
```
print(classification_report(logreg_pred, y_test))
```

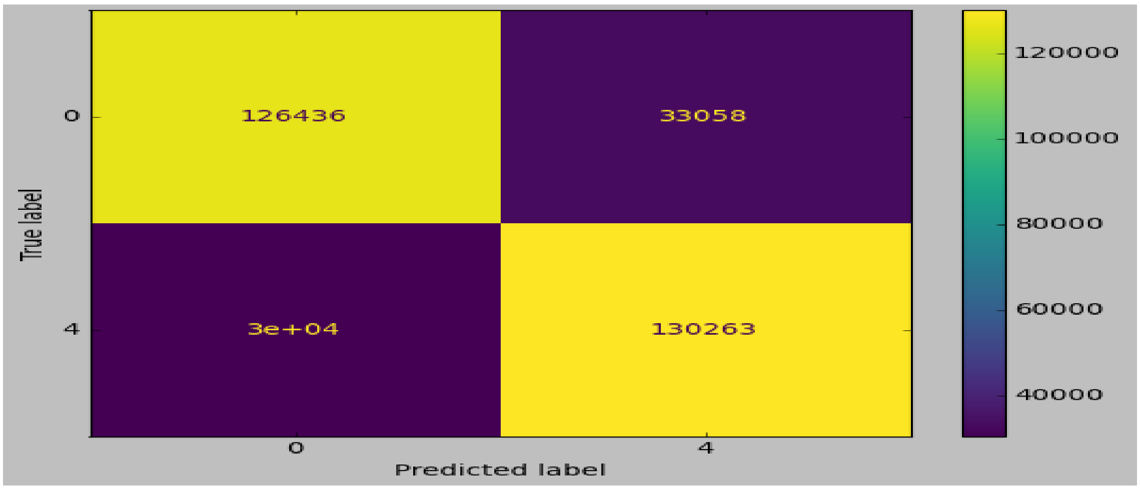
```
              precision    recall  f1-score   support

           0       0.79      0.81      0.80    156679
           4       0.81      0.80      0.80    163321

    accuracy                           0.80    320000
   macro avg       0.80      0.80      0.80    320000
weighted avg       0.80      0.80      0.80    320000
```

In [36]:
```
from matplotlib import style
style.use('ggplot')
```

In [37]:
```
# Graphical representation for confusion matric
style.use('classic')
cm = confusion_matrix(y_test, logreg_pred, labels=logreg.classes_)
disp = ConfusionMatrixDisplay(confusion_matrix = cm, display_labels=logreg.
classes_)
disp.plot()
```



Model 2 : Bernolli

```
#create model object
BNBmodel = BernoulliNB()

# fit model
BNBmodel.fit(X_train_tfidf, y_train)

#predict model
BNBmodel_pred = BNBmodel.predict(X_test_tfidf)

# check accuracy
BNBmodel_acc = accuracy_score(BNBmodel_pred, y_test)

#Print  test accuracy
print("Test accuracy: {:.2f}%".format(BNBmodel_acc*100))


print("confusion matrix : ")
confusion_matrix(BNBmodel_pred, y_test)
```

```
Test accuracy: 78.26%
confusion matrix :
```

Out[38]:

```
array([[128517,  38591],
       [ 30977, 121915]])
```

In [39]:

```
print(classification_report(BNBmodel_pred, y_test))
```

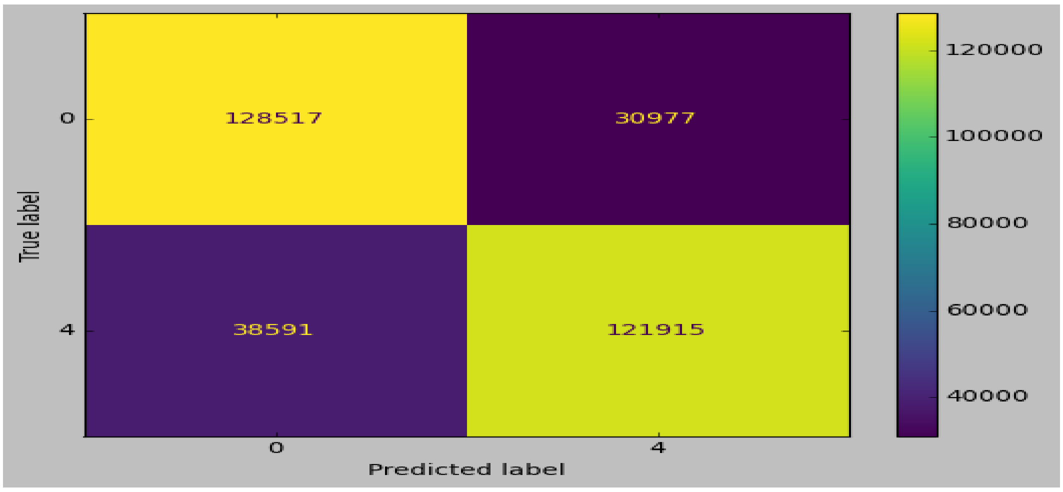
```
              precision    recall  f1-score   support

           0       0.81      0.77      0.79    167108
           4       0.76      0.80      0.78    152892

    accuracy                           0.78    320000
   macro avg       0.78      0.78      0.78    320000
weighted avg       0.78      0.78      0.78    320000
```

In [40]:

```
# Graphical representation for confusion matric
style.use('classic')
cm_1 = confusion_matrix(y_test,BNBmodel_pred , labels=BNBmodel.classes_)
disp_1 = ConfusionMatrixDisplay(confusion_matrix = cm_1, display_labels=BNBm
odel.classes_)
disp_1.plot()
```



Model 3 : Linear Regresion

```
## Create our model object

from sklearn.svm import LinearSVC

#create model object
SVCmodel = LinearSVC()

# fit model
SVCmodel.fit(X_train_tfidf, y_train)

#predict model
SVCmodel_pred = SVCmodel.predict(X_test_tfidf)

# check accuracy
SVC_acc  = accuracy_score(SVCmodel_pred, y_test)

#Print  test accuracy
print("Test accuracy: {:.2f}%".format(SVC_acc*100))

print("confusion matrix : " ,confusion_matrix(SVCmodel_pred, y_test))
```

```
Test accuracy: 79.61%
confusion matrix :  [[126193  31935]
 [ 33301 128571]]
```

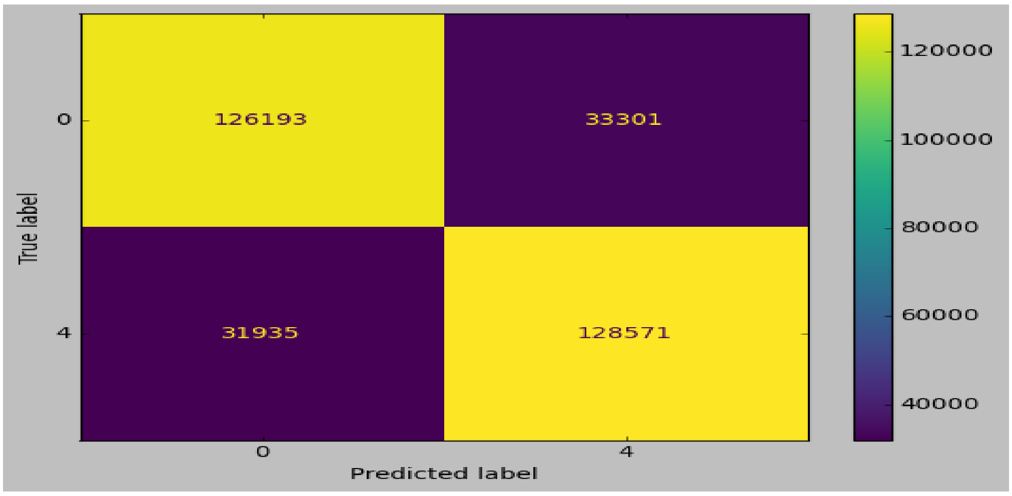
```
42]:  print(classification_report(SVCmodel_pred, y_test))
```

```
              precision    recall  f1-score   support

           0       0.79      0.80      0.79    158128
           4       0.80      0.79      0.80    161872

    accuracy                           0.80    320000
   macro avg       0.80      0.80      0.80    320000
weighted avg       0.80      0.80      0.80    320000
```

```
In [43]:
# Graphical representation for confusion matric
style.use('classic')
cm_1 = confusion_matrix(y_test,SVCmodel_pred , labels=SVCmodel.classes_)
disp_1 = ConfusionMatrixDisplay(confusion_matrix = cm_1, display_labels=SVCm
odel.classes_)
disp_1.plot()
```



Step-10: Model Evaluation

Upon evaluating all the models, we can conclude the following details i.e.

Accuracy : As far as the accuracy of the model is concerned, Logistic Regression performs better than SVM, which in turn performs better than Bernoulli Naive Bayes.

F1-score : The F1 Scores for class 0 and class 1 are :

(a) For class 0: Bernoulli Naive Bayes(accuracy = 0.79) < SVM (accuracy =0.79) < Logistic Regression (accuracy = 0.80)

(b) For class 1: Bernoulli Naive Bayes (accuracy = 0.78) < SVM (accuracy = 0.80) < Logistic Regression (accuracy = 0.80)

Strengths:

1. The code provides a clear and well-structured approach to sentiment analysis, making it easy to follow and understand.
2. Preprocessing techniques such as cleaning, tokenization, and stemming contribute to improving the quality of the input data.
3. The use of TF-IDF vectorization allows the model to capture the significance of words within the dataset effectively.
4. The code employs a variety of popular machine learning algorithms, enabling a comprehensive comparison of their performance.

Limitations and Suggestions for Improvement:

- The code could benefit from incorporating techniques for handling imbalanced datasets, as sentiment analysis datasets often have class imbalances.
- Further explanation of the model selection process and justification for choosing specific algorithms would enhance the code's transparency.
- The addition of cross-validation and hyperparameter tuning techniques could provide a more robust evaluation of the models.
- It would be valuable to explore the use of deep learning models, such as recurrent neural networks (RNNs) or transformers, for sentiment analysis to compare their performance against traditional ML algorithms.

Conclusion:

The code presented in the provided link offers a valuable demonstration of sentiment analysis using a well-defined methodology and a variety of machine learning algorithms. By following the step-by-step instructions, one can effectively preprocess text data, extract relevant features, and train sentiment analysis models. However, certain enhancements, such as handling class imbalances, further justifying algorithm selection, and exploring advanced models, could elevate the code's quality and provide a more comprehensive analysis.