# SENTIMENTAL ANALYSIS FOR MARKETING

## PHASE 4 PROJECT SUBMISSION

## INTRODUCTION:

Building a successful project involves a series of crucial activities, each playing a vital role in achieving your goals. In this context, we'll discuss three key stages: feature engineering, model training, and evaluation. Feature engineering is the art of selecting, transforming, and creating meaningful input variables to enhance the predictive power of your models. Model training is the process of teaching your machine learning algorithms to make accurate predictions based on your data. Lastly, evaluation is where you measure how well your model performs and make informed decisions about its effectiveness. Let's delve into each of these stages to understand their importance in project development.

## ABSTRACT:

The project aims to develop a machine learning model for a specific task. It involves several key activities, including feature engineering, model training, and evaluation. These activities are organized into separate modules to ensure a structured and efficient workflow.

## MODULE:

Module 1: Feature Engineering

- Description: This module focuses on data preprocessing and feature engineering. It involves tasks like data cleaning, feature selection, transformation, and creation to prepare the data for model training.

-Key Tasks:

- Data preprocessing and cleaning
- Feature selection
- Feature transformation
- Feature generation

-Output: Processed dataset with engineered features.

Module 2: Model Training

- *Description:* This module is responsible for training machine learning models using the processed data from the feature engineering module.

- Key Tasks:
  - Model selection
  - Hyperparameter tuning
  - Model training
- Output: Trained machine learning model.

Module 3: Evaluation

- Description: This module evaluates the performance of the trained model to assess its effectiveness.
- Key Tasks:
  - Performance metrics calculation
  - Cross-validation
  - Model comparison
- Output: Model evaluation results, including metrics and visualizations.

By breaking down the project into these modules, you can create a structured and organized workflow for your project, making it easier to manage and collaborate with others. Each module can be further developed with specific tools, libraries, and code, depending on your project's requirements.

Building a sentiment analysis project for marketing typically involves several key activities:

1. Data Collection: Gather data from various sources, such as social media, customer reviews, or surveys. This data will be used to train and test your sentiment analysis model.

2. Data Preprocessing: Clean and preprocess the collected data. This includes tasks like text cleaning, tokenization, and removing stop words.

3. Feature Engineering: Extract relevant features from the text data. This might involve techniques like TF-IDF (Term Frequency-Inverse Document Frequency) or word embeddings like Word2Vec or GloVe.

4. Labeling: Annotate your data with sentiment labels (e.g., positive, negative, neutral) to create a labeled dataset for training and evaluation.

5. Model Selection: Choose a machine learning or deep learning model for sentiment analysis. Common choices include Naïve Bayes, Support Vector Machines, or Recurrent Neural Networks (RNNs) and Transformers like BERT.

6. Model Training: Train your selected model on the labeled dataset. This involves feeding the model with your feature-engineered data and adjusting model parameters to improve performance.

7. Evaluation: Assess the model's performance using metrics like accuracy, precision, recall, and F1-score. Cross-validation and fine-tuning may be necessary to optimize performance.

8. Testing: Use the trained model to predict sentiment on new, unseen data to ensure its generalization capability.

9. Deployment: If you plan to use this sentiment analysis in marketing campaigns, deploy the model in your marketing infrastructure. This could involve creating APIs or integrating it with your marketing tools.

10. Monitoring and Maintenance: Regularly monitor the model's performance and retrain it if necessary, as sentiment patterns can change over time.

Remember that the success of your sentiment analysis project depends on the quality of your data, feature engineering, and the choice of the appropriate model for your specific marketing context.

## PROGRAM:

Classification task with and without doing feature engineering

| Split | Real | Fake | Total |
|---|---|---|---|
| Train | 3360 | 3060 | 6420 |
| Validation | 1120 | 1020 | 2140 |

Test 1120 1020 2140

List of features

I will be listing out a total of 15 features that we can use for the above dataset, number of features totally depends upon the type of dataset you are using.

1. Number of Characters

Count the number of characters present in a tweet.

```
Def count_chars(text):
  Return len(text)
```

2. Number of words

Count the number of words present in a tweet.

```
Def count_words(text):
  Return len(text.split())
```

3. Number of capital characters

Count the number of capital characters present in a tweet.

Python Code:

4. Number of capital words

Count the number of capital words present in a tweet.

```
Def count_capital_words(text):
  Return sum(map(str.isupper,text.split()))
```

5. Count the number of punctuations

In this function, we return a dictionary of 32 punctuation with the counts, which can be used as separate features, which I will discuss in the next section.

```
Def count_punctuations(text):
    Punctuations='!"#$%&'()*+,-./:;⇔?@[\]^_`{|}~'
    D=dict()
    For I in punctuations:
        D[str(i)+' count']=text.count(i)
    Return d
```

6. Number of words in quotes

The number of words in the single quotation and double quotation.

```
Def count_words_in_quotes(text):
    X = re.findall("'.'|"."", text)
    Count=0
    If x is None:
        Return 0
    Else:
        For I in x:
            T=i[1:-1]
            Count+=count_words(t)
        Return count
```

7. Number of sentences

Count the number of sentences in a tweet.

```
Def count_sent(text):
   Return len(nltk.sent_tokenize(text))
```

8. Count the number of unique words

Count the number of unique words in a tweet.

```
Def count_unique_words(text):
   Return len(set(text.split()))
```

9. Count of hashtags

Since we are using the Twitter dataset we can count the number of times users used the hashtag.

```
Def count_htags(text):
   X = re.findall(r'(#w[A-Za-z0-9]*)', text)
   Return len(x)
```

10. Count of mentions

On Twitter, most of the time people reply or mention someone in their tweet, counting the number of mentions can also be treated as a feature.

```
Def count_mentions(text):
   X = re.findall(r'(@w[A-Za-z0-9]*)', text)
   Return len(x)
```

11. Count of stopwords

Here we will count the number of stopwords used in a tweet.

```
Def count_stopwords(text):
```

```
Stop_words = set(stopwords.words('english'))
Word_tokens = word_tokenize(text)
Stopwords_x = [w for w in word_tokens if w in stop_words]
Return len(stopwords_x)
```

12. Calculating average word length

This can be calculated by dividing the counts of characters by counts of words.

```
Df['avg_wordlength'] = df['char_count']/df['word_count']
```

13. Calculating average sentence length

This can be calculated by dividing the counts of words by the counts of sentences.

```
Df['avg_sentlength'] = df['word_count']/df['sent_count']
```

14. Unique words vs word count feature

This feature is basically the ratio of unique words to a total number of words.

```
Df['unique_vs_words'] = df['unique_word_count']/df['word_count']
```

15. Stopwords count vs words counts feature

This feature is also the ratio of counts of stopwords to the total number of words.

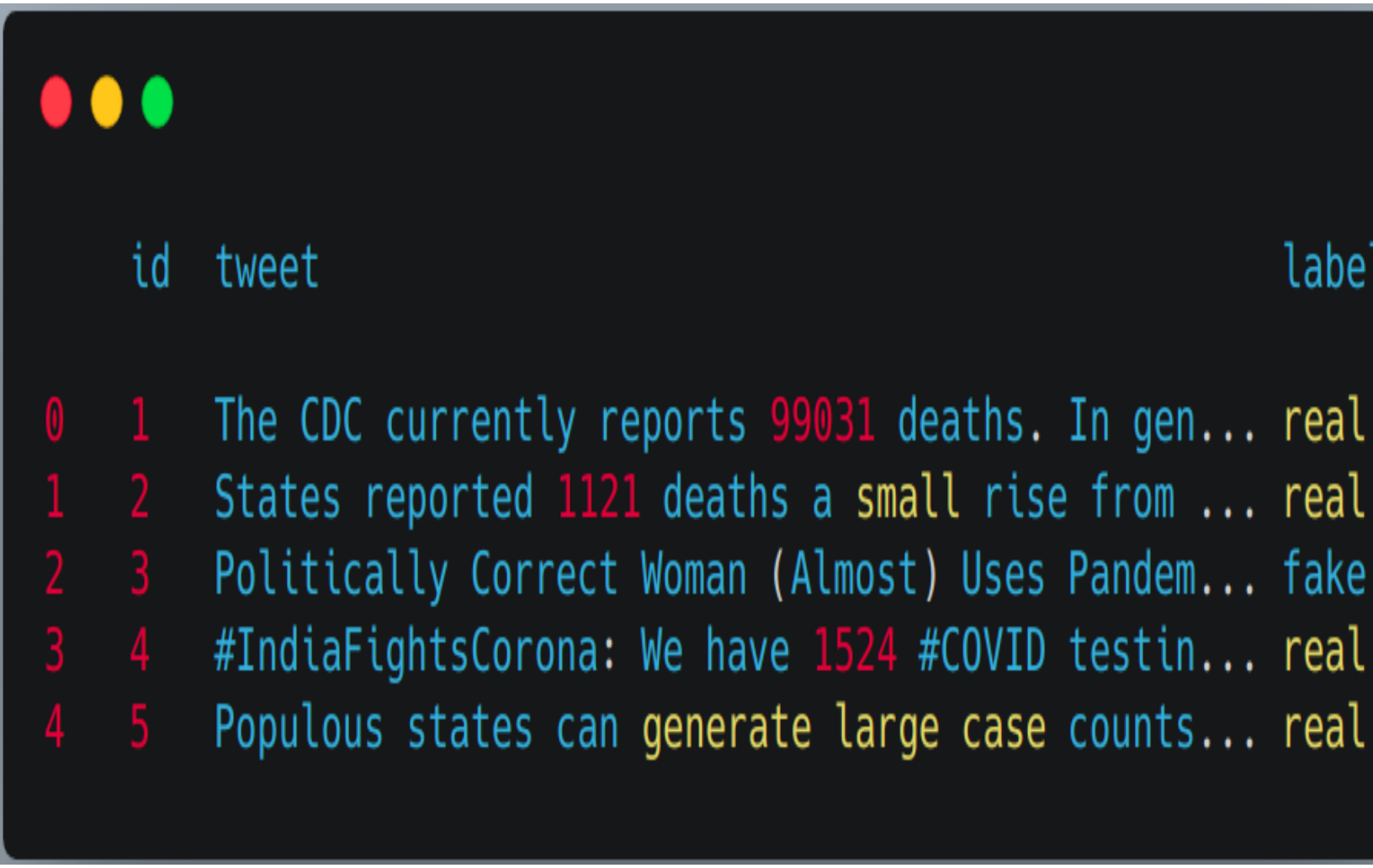
```
Df['stopwords_vs_words'] = df['stopword_count']/df['word_count']
```

Implementation

You can download the dataset from here. After downloading we can start implementing all features we defined above. We will focus more on feature engineering, for this we will keep the approach simple, by using TF-IDF and simple pre

Reading train, validation, and test set using pandas.

```
Train = pd.read_csv("train.csv")
```



| | id | tweet | label |
|---|---|---|---|
| 0 | 1 | The CDC currently reports 99031 deaths. In gen... | real |
| 1 | 2 | States reported 1121 deaths a small rise from ... | real |
| 2 | 3 | Politically Correct Woman (Almost) Uses Pandem... | fake |
| 3 | 4 | #IndiaFightsCorona: We have 1524 #COVID testin... | real |
| 4 | 5 | Populous states can generate large case counts... | real |

Val = pd.read_csv("validation.csv")

Test = pd.read_csv(testWithLabel.csv")

# For this task we will combine the train and validation dataset and then use

# simple train test split from sklern.

Df = pd.concat([train, val])

Df.head()

Feature engineering in NLP head

First 5 entries

Applying the above-defined feature extraction on train and test set.

```
Df['char_count'] = df["tweet"].apply(lambda x:count_chars(x))

Df['word_count'] = df["tweet"].apply(lambda x:count_words(x))

Df['sent_count'] = df["tweet"].apply(lambda x:count_sent(x))

Df['capital_char_count'] = df["tweet"].apply(lambda x:count_capital_chars(x))

Df['capital_word_count'] = df["tweet"].apply(lambda x:count_capital_words(x))

Df['quoted_word_count'] = df["tweet"].apply(lambda x:count_words_in_quotes(x))

Df['stopword_count'] = df["tweet"].apply(lambda x:count_stopwords(x))

Df['unique_word_count'] = df["tweet"].apply(lambda x:count_unique_words(x))

Df['htag_count'] = df["tweet"].apply(lambda x:count_htags(x))

Df['mention_count'] = df["tweet"].apply(lambda x:count_mentions(x))

Df['punct_count'] = df["tweet"].apply(lambda x:count_punctuations(x))

Df['avg_wordlength'] = df['char_count']/df['word_count']

Df['avg_sentlength'] = df['word_count']/df['sent_count']

Df['unique_vs_words'] = df['unique_word_count']/df['word_count']

Df['stopwords_vs_words'] = df['stopword_count']/df['word_count']

# SIMILARLY YOU CAN APPLY THEM ON TEST SET
```

Dding some extra features using punctuation count

We will create a DataFrame from the dictionary returned by the "punct_count" function and then merge it with the main dataset.

```
Df_punct = pd.DataFrame(list(df.punct_count))

Test_punct = pd.DataFrame(list(test.punct_count))


# Merging pnctuation DataFrame with main DataFrame

Df = pd.merge(df, df_punct, left_index=True, right_index=True)

Test = pd.merge(test, test_punct,left_index=True, right_index=True)

# We can drop "punct_count" column from both df and test DataFrame

Df.drop(columns=['punct_count'],inplace=True)
```

Test.drop(columns=[‘punct_count’],inplace=True)

Df.columns

A Guide to Feature Engineering in NLP column

Final columns list

```
Index(['id', 'tweet', 'label', 'char_count', 'word_count', 'sent_count',
       'capital_char_count', 'capital_word_count', 'quoted_word_count',
       'stopword_count', 'unique_word_count', 'htag_count', 'mention_count',
       'avg_wordlength', 'avg_sentlength', 'unique_vs_words',
       'stopwords_vs_words', '! count', '" count', '# count', '$ count',
       '% count', '& count', ' count', '( count', ') count', '* count',
       '+ count', ', count', '- count', '. count', '/ count', ': count',
       '; count', '< count', '= count', '> count', '? count', '@ count',
       '[ count', '\ count', '] count', '^ count', '_ count', '` count',
       '{ count', '| count', '} count', '~ count'],
      dtype='object')
```

Re-processing

We performed a simple pre-processing step, like removing links, removing user name, numbers, double space, punctuation, lower casing, etc.

Def remove_links(tweet):

‘’’Takes a string and removes web links from it’’’

Tweet = re.sub(r’httpS+’, ‘’, tweet) # remove http links

Tweet = re.sub(r’bit.ly/S+’, ‘’, tweet) # rempve bitly links

Tweet = tweet.strip(‘[link]’) # remove [links]

```
    Return tweet
Def remove_users(tweet):
    '''Takes a string and removes retweet and @user information'''
    Tweet = re.sub('(RTs@[A-Za-z]+[A-Za-z0-9-_]+)', '', tweet) # remove retweet
    Tweet = re.sub('(@[A-Za-z]+[A-Za-z0-9-_]+)', '', tweet) # remove tweeted at
    Return tweet
My_punctuation = '!"$%&'()*+,-./:;⇔?[\]^_`{|}~•@'
Def preprocess(sent):
    Sent = remove_users(sent)
    Sent = remove_links(sent)
    Sent = sent.lower() # lower case
    Sent = re.sub('['+my_punctuation + ']+', ' ', sent) # strip punctuation
    Sent = re.sub('s+', ' ', sent) #remove double spacing
    Sent = re.sub('([0-9]+)', '', sent) # remove numbers
    Sent_token_list = [word for word in sent.split(' ')]
    Sent = ' '.join(sent_token_list)
    Return sent
Df['tweet']   = df['tweet'].apply(lambda x: preprocess(x))
Test['tweet'] = test['tweet'].apply(lambda x: preprocess(x))
```

Encoding text

We will encode our text data using TF-IDF. We first fit transform on our train and test set's tweet column and then merge it with all features columns.

```
Vectorizer          =  TfidfVectorizer()
Train_tf_idf_features =  vectorizer.fit_transform(df['tweet']).toarray()
Test_tf_idf_features  =  vectorizer.transform(test['tweet']).toarray()

# Converting above list to DataFrame
Train_tf_idf        = pd.DataFrame(train_tf_idf_features)
```

```
Test_tf_idf          = pd.DataFrame(test_tf_idf_features)

# Saparating train and test labels from all features
Train_Y              = df['label']
Test_Y               = test['label']

#Listing all features
Features = ['char_count', 'word_count', 'sent_count',
      'capital_char_count', 'capital_word_count', 'quoted_word_count',
      'stopword_count', 'unique_word_count', 'htag_count', 'mention_count',
      'avg_wordlength', 'avg_sentlength', 'unique_vs_words',
      'stopwords_vs_words', '! Count', '" count', '# count', '$ count',
      '% count', '& count', ' count', '( count', ') count', '* count',
      '+ count', ', count', '- count', '. Count', '/ count', ': count',
      '; count', '< count', '= count', '> count', '? Count', '@ count',
      '[ count', ' count', '] count', '^ count', '_ count', ` count',
      '{ count', '| count', '} count', '~ count']

# Finally merging all features with above TF-IDF.
Train = pd.merge(train_tf_idf,df[features],left_index=True, right_index=True)
Test  = pd.merge(test_tf_idf,test[features],left_index=True, right_index=True)
```

Training

For training, we will be using the Random forest algorithm from the sci-kit learn library.

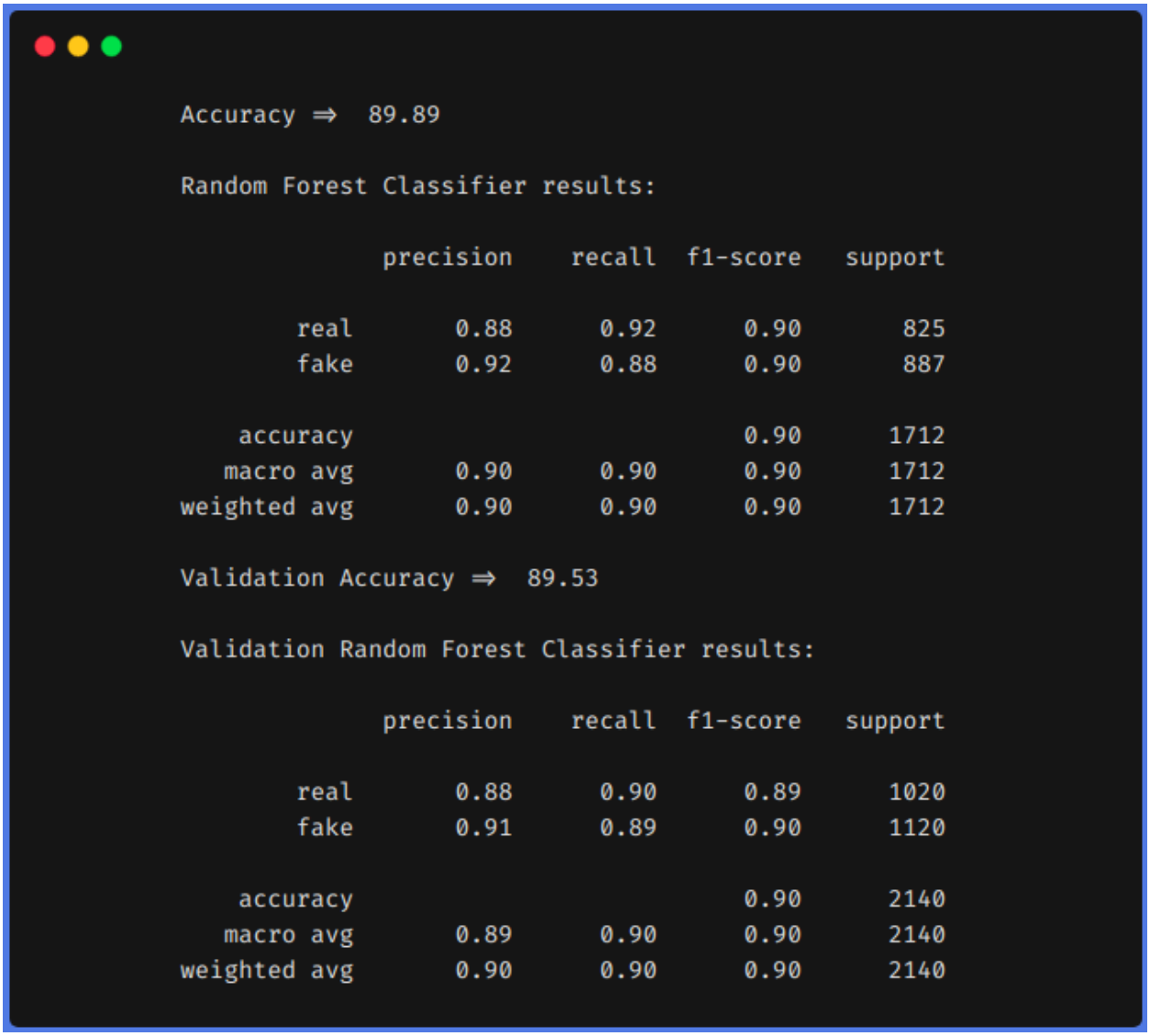
```
X_train, X_test, y_train, y_test = train_test_split(train, train_Y, test_size=0.2, random_state = 42)
# Random Forest Classifier
Clf_model = RandomForestClassifier(n_estimators = 1000, min_samples_split = 15, random_state = 42)
Clf_model.fit(X_train, y_train)
_RandomForestClassifier_prediction = clf_model.predict(X_test)
```

Val_RandomForestClassifier_prediction = clf_model.predict(test)

Result comparison

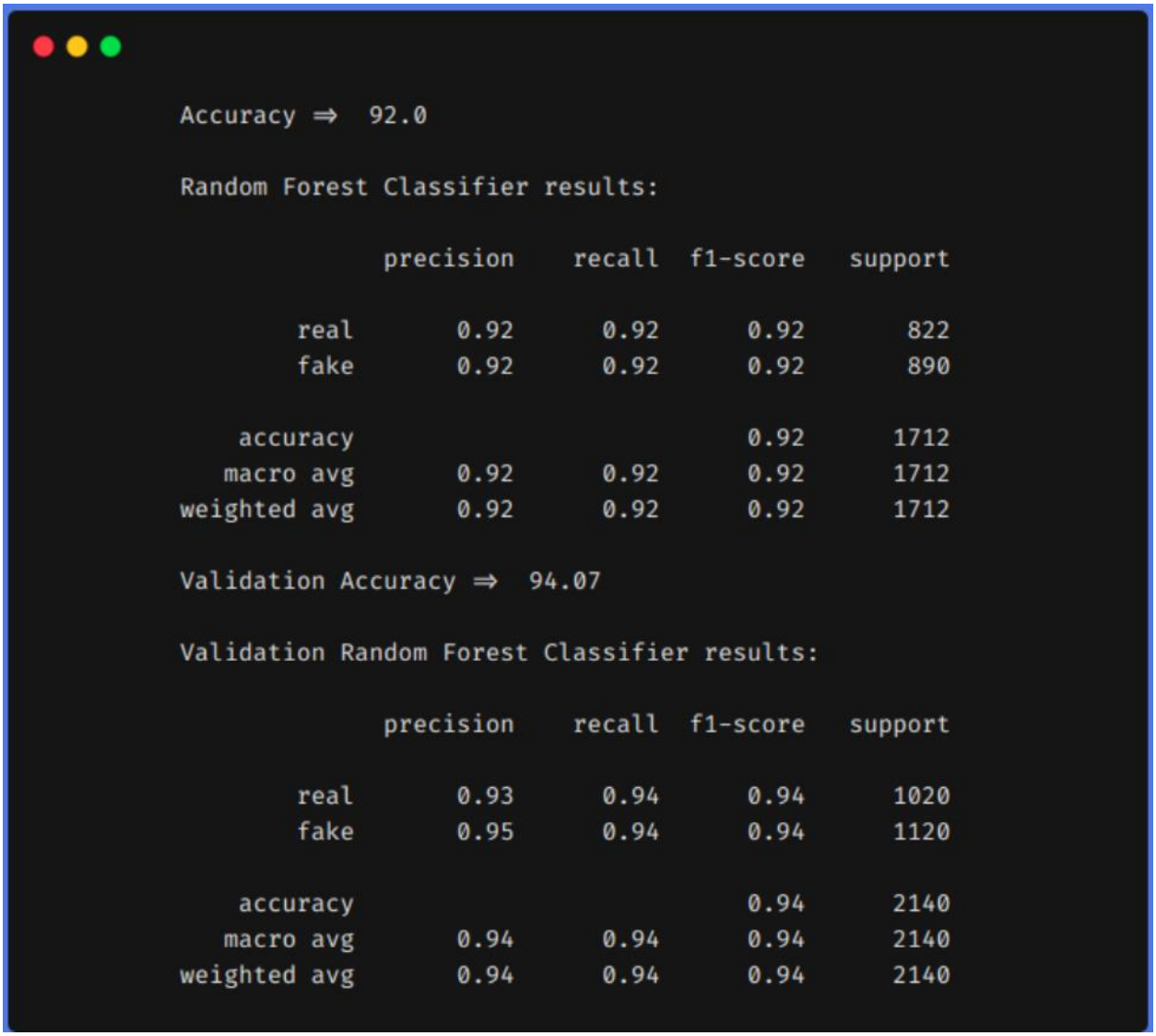
For comparison, we first trained our model on the above dataset by using features engineering techniques and then without using feature engineering techniques. In both approaches, we pre-processed the dataset using the same method as described above and TF-IDF was used in both approaches for encoding the text data. You can use whatever encoding techniques you want to use like word2vec, glove, etc.

1. Without using Feature Engineering techniques



```
Accuracy ⇒  89.89

Random Forest Classifier results:

              precision    recall  f1-score   support

        real       0.88      0.92      0.90       825
        fake       0.92      0.88      0.90       887

    accuracy                           0.90      1712
   macro avg       0.90      0.90      0.90      1712
weighted avg       0.90      0.90      0.90      1712

Validation Accuracy ⇒  89.53

Validation Random Forest Classifier results:

              precision    recall  f1-score   support

        real       0.88      0.90      0.89      1020
        fake       0.91      0.89      0.90      1120

    accuracy                           0.90      2140
   macro avg       0.89      0.90      0.90      2140
weighted avg       0.90      0.90      0.90      2140
```

2. Without using Feature Engineering



```
Accuracy ⇒  92.0

Random Forest Classifier results:

              precision    recall  f1-score   support

        real       0.92      0.92      0.92       822
        fake       0.92      0.92      0.92       890

    accuracy                           0.92      1712
   macro avg       0.92      0.92      0.92      1712
weighted avg       0.92      0.92      0.92      1712

Validation Accuracy ⇒  94.07

Validation Random Forest Classifier results:

              precision    recall  f1-score   support

        real       0.93      0.94      0.94      1020
        fake       0.95      0.94      0.94      1120

    accuracy                           0.94      2140
   macro avg       0.94      0.94      0.94      2140
weighted avg       0.94      0.94      0.94      2140
```

Here validation accuracy is test accuracy.

From the above results, we can see that feature engineering techniques helped us to increase our f1 from 0.90 to 0.92 in the train set and from 0.90 to 0.94 in the test set

Loading the Data from a Data Set

```
>>> import pandas as pd
>>> df = pd.read_csv('/datasets/sentiment/amazon_cells_labelled.txt', names=['review', 'sentiment'], sep='\t')
>>> df.head()
```

0 So there is no way for me to plug it in here i… 0

1 Good case, Excellent value. 1

2 Great for the jawbone. 1

3 Tied to charger for conversations lasting more… 0

4 The mic is great. 1

Splitting the Data Set into a Training Set and a Test Set

```
>>> from sklearn.model_selection import train_test_split
>>> reviews = df['review'].values
>>> labels = df['sentiment'].values
>>> reviews_train, reviews_test, y_train, y_test = train_test_split(reviews, labels, test_size=0.2, random_state=1000)
```

Loading the Data from a Data Set

In the article example, we'll take advantage of the Sentiment Labelled Sentences Data Set available from the UCI Machine Learning Repository. This data set contains customer product reviews labeled with

positive or negative sentiment. The reviews were taken from three different websites, including imdb.com, amazon.com, and yelp.com. So, the data comes in three different files. The total number of instances in the data set is 3000. In the article example, we'll use the instances (labeled sentences) that were taken from amazon.com only. These are 1000 sentences, where 500 ones are positive and the other 500 are negative.

In the code snippet below, you load the data into a pandas dataframe to simplify further calculations:

```
>>> import pandas as pd

>>> df = pd.read_csv('/datasets/sentiment/amazon_cells_labelled.txt', names=['review', 'sentiment'], sep='\t')

>>> df.head()

0 So there is no way for me to plug it in here i… 0

1 Good case, Excellent value. 1

2 Great for the jawbone. 1

3 Tied to charger for conversations lasting more… 0

4 The mic is great. 1
```

Splitting the Data Set into a Training Set and a Test Set

If you recall, our goal is to train a model to predict the sentiment of a review. The first step is to split the data we have into training and test sets. With the sklearn library, this can be accomplished with a few lines of code:

```
>>> from sklearn.model_selection import train_test_split

>>> reviews = df['review'].values

>>> labels = df['sentiment'].values

>>> reviews_train, reviews_test, y_train, y_test = train_test_split(reviews, labels, test_size=0.2, random_state=1000)
```

In this particular example, we divided the data set with the 80/20 pattern, meaning that 80% of instances chosen randomly is used for training a model and the other 20% for its further testing. Anyway, what we have at the moment are still two sets with labeled text data. However, to train an ML model and then test it, we need a way to represent the text data numerically.

```
>>> from spacy.lang.en.stop_words import STOP_WORDS
>>> from spacy.lang.en import English
>>> import string
>>> punctuations = string.punctuation
>>> parser = English()
>>> stopwords = list(STOP_WORDS)
>>> def spacy_tokenizer(utterance):
    Tokens = parser(utterance)
```

Transforming Text into Numerical Feature Vectors

As mentioned, transforming text to feature vectors can be done with sklearn's CountVectorizer() function. In the example below, we use the spaCy's custom tokenizer created in the previous section. Alternatively, you might use the default option, passing no parameters to CountVectorizer().

```
>>> from sklearn.feature_extraction.text import CountVectorizer
>>> vectorizer = CountVectorizer(tokenizer = spacy_tokenizer, ngram_range=(1,1))
>>> #By default, the vectorizer might be created as follows:
>>> #vectorizer = CountVectorizer()
>>> vectorizer.fit(reviews_train)
```

Below you transform the text into numerical feature vectors:

```
>>> X_train = vectorizer.transform(reviews_train)
>>> X_test = vectorizer.transform(reviews_test)
```

## CONCLUSION:

In conclusion, sentiment analysis is a valuable tool in the field of marketing. Building a project for sentiment analysis involves several key steps, including feature engineering, model training, and evaluation. Feature engineering is crucial for extracting meaningful insights from textual data, while model training enables the creation of accurate sentiment classifiers. Evaluation helps in assessing the model's performance.

This project's successful execution can provide businesses with valuable insights into customer opinions and preferences, allowing them to make data-driven decisions for product development, advertising, and customer engagement strategies. Sentiment analysis can be a powerful asset for marketing, enabling companies to understand and respond to consumer sentiment effectively.