analyzingTextualData

July 9, 2022

[6]:
```
# analyzing textual Data

# text normalization
# input text
paragraph="""TO jest test probny. Zamienione zostaną WSZYSTKIE DUZE LItery na␣
 ↪male."""

# converting paragraph in lowercase
print(paragraph.lower())
```

```
to jest test probny. zamienione zostaną wszystkie duze litery na male.
```

[1]:
```
# tokenization
# loading NLTK module
import nltk

# downloading punkt
nltk.download('punkt')

# downloading stopwords
nltk.download('stopwords')

# downloading wordnet
nltk.download('wordnet')

# downloading average_perception_tagger
nltk.download('averaged_perceptron_tagger')
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\Admin\AppData\Roaming\nltk_data…
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\Admin\AppData\Roaming\nltk_data…
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\Admin\AppData\Roaming\nltk_data…
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
```

```
[nltk_data]     C:\Users\Admin\AppData\Roaming\nltk_data…
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
```

[1]:
```
True
```

[7]:
```python
# sentence tokenization
from nltk.tokenize import sent_tokenize

paragraph = """Taj Mahal is one of the beautiful monuments.
It is one of the wonders of the world.
It was build by Shah Jahan is 1631 in memory of his third beloved wife Mumtaj␣
 ↪Mahal."""

tokenized_sentences = sent_tokenize(paragraph)
print(tokenized_sentences)
```

```
['Taj Mahal is one of the beautiful monuments.', 'It is one of the wonders of
the world.', 'It was build by Shah Jahan is 1631 in memory of his third beloved
wife Mumtaj Mahal.']
```

[8]:
```python
# import spacy
import spacy

# loading english language model
nlp = spacy.load("en_core_web_sm")

# build the nlp pipe
sent_pipe = nlp.create_pipe('sentencizer')

# append the sentenzicer pipe to the npl pipeline
paragraph = """Taj Mahal is one of the beautiful monuments.
Is is one of the wonders of the world. Is was built by Shah Jahan."""

# create nlp object to handle lingustic annotatations in a documents
nlp_doc = nlp(paragraph)

# generate list of tokenized sentence
tokenized_sentences = []
for sentence in nlp_doc.sents:
    tokenized_sentences.append(sentence.text)
print(tokenized_sentences)
```

```
['Taj Mahal is one of the beautiful monuments.', '\nIs is one of the wonders of
the world.', 'Is was built by Shah Jahan.']
```

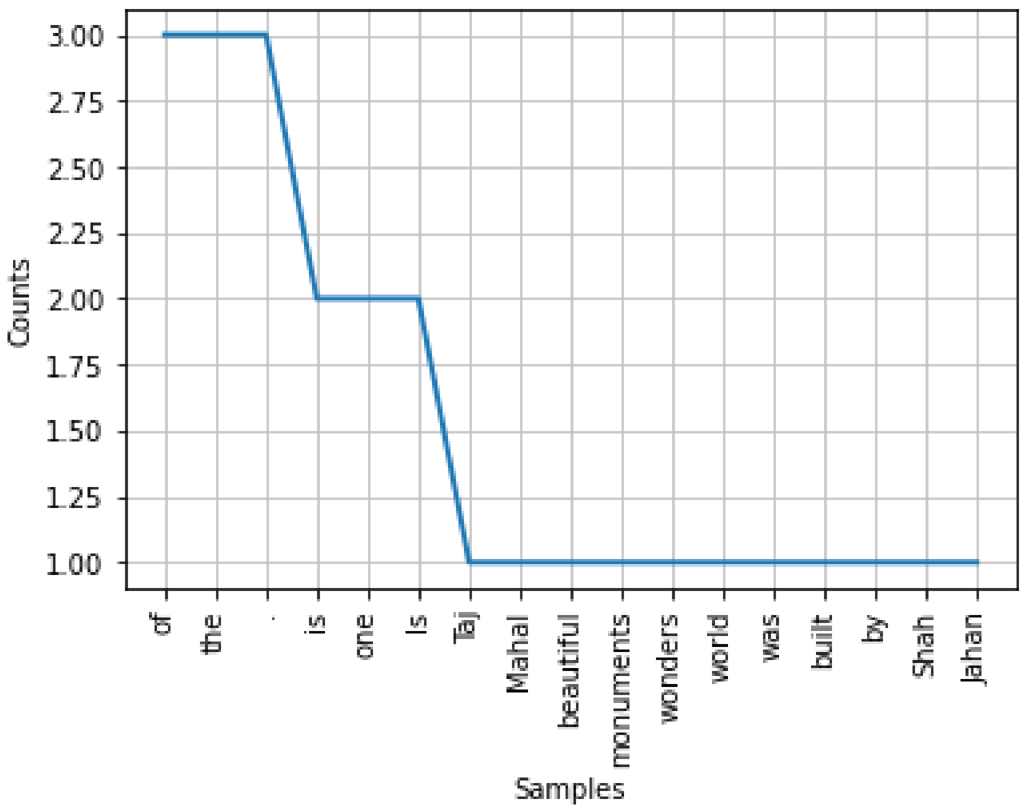
```
[9]: # import nltk word_tokenize method
     from nltk.tokenize import word_tokenize

     # split paragraph into words
     tokenized_words = word_tokenize(paragraph)
     print(tokenized_words)
```

```
['Taj', 'Mahal', 'is', 'one', 'of', 'the', 'beautiful', 'monuments', '.', 'Is',
'is', 'one', 'of', 'the', 'wonders', 'of', 'the', 'world', '.', 'Is', 'was',
'built', 'by', 'Shah', 'Jahan', '.']
```

```
[25]: # import frequence distribution
      from nltk.probability import FreqDist

      # find frequency distribution of paragraph
      fdisk = FreqDist(tokenized_words)

      # check top 5 common words
      fdisk.most_common(5)
```

```
[25]: [('of', 3), ('the', 3), ('.', 3), ('is', 2), ('one', 2)]
```

```
[26]: # import matplotlib
      import matplotlib.pyplot as plt

      # plot frequency distribution
      fdisk.plot(20, cumulative=False)
      plt.show()
```



[11]:
```python
# removing stopwords

# import the nltk stopwords
from nltk.corpus import stopwords

# load english stopwords list
stopwords_set = set(stopwords.words("english"))

# removing stopwords from text
filtered_word_list = []
for word in tokenized_words:
    # filter stopwords
    if word not in stopwords_set:
        filtered_word_list.append(word)
# print tokenized words
print("Tokenized Word List: ", tokenized_words)

# print filtered words
print("Filtered Word List: ", filtered_word_list)
```

```
Tokenized Word List:  ['Taj', 'Mahal', 'is', 'one', 'of', 'the', 'beautiful',
'monuments', '.', 'Is', 'is', 'one', 'of', 'the', 'wonders', 'of', 'the',
'world', '.', 'Is', 'was', 'built', 'by', 'Shah', 'Jahan', '.']
Filtered Word List:  ['Taj', 'Mahal', 'one', 'beautiful', 'monuments', '.',
'Is', 'one', 'wonders', 'world', '.', 'Is', 'built', 'Shah', 'Jahan', '.']
```

[28]:
```python
# stemming and lemmatization

# import Lemmatizer
from nltk.stem.wordnet import WordNetLemmatizer

# import Porter Stemmer
from nltk.stem.porter import PorterStemmer

# create lemmatizer object
lemmatizer = WordNetLemmatizer()

# create stemmer object
stemmer = PorterStemmer()

# take a sample word
sample_word = "crying"
print("Lemmatized Sample Word:", lemmatizer.lemmatize(sample_word, "v"))

print("Stemmed Sample Word:", stemmer.stem(sample_word))
```

```
---------------------------------------------------------------------------
LookupError                               Traceback (most recent call last)
File ~\anaconda3\lib\site-packages\nltk\corpus\util.py:84, in LazyCorpusLoader.
  ↪__load(self)
     83 try:
---> 84     root = nltk.data.find(f"{self.subdir}/{zip_name}")
     85 except LookupError:

File ~\anaconda3\lib\site-packages\nltk\data.py:583, in find(resource_name,␣
  ↪paths)
    582 resource_not_found = f"\n{sep}\n{msg}\n{sep}\n"
--> 583 raise LookupError(resource_not_found)

LookupError:
**********************************************************************
  Resource omw-1.4 not found.
  Please use the NLTK Downloader to obtain the resource:

  >>> import nltk
  >>> nltk.download('omw-1.4')

  For more information see: https://www.nltk.org/data.html
  Attempted to load corpora/omw-1.4.zip/omw-1.4/

  Searched in:
    - 'C:\\Users\\Admin/nltk_data'
    - 'C:\\Users\\Admin\\anaconda3\\nltk_data'
    - 'C:\\Users\\Admin\\anaconda3\\share\\nltk_data'
    - 'C:\\Users\\Admin\\anaconda3\\lib\\nltk_data'
    - 'C:\\Users\\Admin\\AppData\\Roaming\\nltk_data'
    - 'C:\\nltk_data'
    - 'D:\\nltk_data'
    - 'E:\\nltk_data'
**********************************************************************


During handling of the above exception, another exception occurred:

LookupError                               Traceback (most recent call last)
Input In [28], in <cell line: 17>()
     15 # take a sample word
     16 sample_word = "crying"
---> 17 print("Lemmatized Sample Word:", lemmatizer.lemmatize(sample_word, "v")
     19 print("Stemmed Sample Word:", stemmer.stem(sample_word))

File ~\anaconda3\lib\site-packages\nltk\stem\wordnet.py:45, in WordNetLemmatize.
 ↪lemmatize(self, word, pos)
     33 def lemmatize(self, word: str, pos: str = "n") -> str:
     34     """Lemmatize `word` using WordNet's built-in morphy function.
     35     Returns the input word unchanged if it cannot be found in WordNet.
     36 
   (…)
     43     :return: The lemma of `word`, for the given `pos`.
     44     """
---> 45     lemmas = wn._morphy(word, pos)
     46     return min(lemmas, key=len) if lemmas else word

File ~\anaconda3\lib\site-packages\nltk\corpus\util.py:121, in LazyCorpusLoader
 ↪__getattr__(self, attr)
    118 if attr == "__bases__":
    119     raise AttributeError("LazyCorpusLoader object has no attribute␣
 ↪'__bases__'")
--> 121 self.__load()
    122 # This looks circular, but its not, since __load() changes our
    123 # __class__ to something new:
    124 return getattr(self, attr)

File ~\anaconda3\lib\site-packages\nltk\corpus\util.py:89, in LazyCorpusLoader.
 ↪__load(self)
```

```
     86             raise e
     88 # Load the corpus.
---> 89 corpus = self.__reader_cls(root, *self.__args, **self.__kwargs)
     91 # This is where the magic happens!  Transform ourselves into
     92 # the corpus by modifying our own __dict__ and __class__ to
     93 # match that of the corpus.
     95 args, kwargs = self.__args, self.__kwargs

File ~\anaconda3\lib\site-packages\nltk\corpus\reader\wordnet.py:1176, in␣
 ↪WordNetCorpusReader.__init__(self, root, omw_reader)
   1172     warnings.warn(
   1173         "The multilingual functions are not available with this Wordnet␣
 ↪version"
   1174     )
   1175 else:
-> 1176     self.provenances = self.omw_prov()
   1178 # A cache to store the wordnet data of multiple languages
   1179 self._lang_data = defaultdict(list)

File ~\anaconda3\lib\site-packages\nltk\corpus\reader\wordnet.py:1285, in␣
 ↪WordNetCorpusReader.omw_prov(self)
   1283 provdict = {}
   1284 provdict["eng"] = ""
-> 1285 fileids = self._omw_reader.fileids()
   1286 for fileid in fileids:
   1287     prov, langfile = os.path.split(fileid)

File ~\anaconda3\lib\site-packages\nltk\corpus\util.py:121, in LazyCorpusLoader
 ↪__getattr__(self, attr)
    118 if attr == "__bases__":
    119     raise AttributeError("LazyCorpusLoader object has no attribute␣
 ↪'__bases__'")
--> 121 self.__load()
    122 # This looks circular, but its not, since __load() changes our
    123 # __class__ to something new:
    124 return getattr(self, attr)

File ~\anaconda3\lib\site-packages\nltk\corpus\util.py:86, in LazyCorpusLoader.
 ↪__load(self)
     84             root = nltk.data.find(f"{self.subdir}/{zip_name}")
     85         except LookupError:
---> 86             raise e
     88 # Load the corpus.
     89 corpus = self.__reader_cls(root, *self.__args, **self.__kwargs)

File ~\anaconda3\lib\site-packages\nltk\corpus\util.py:81, in LazyCorpusLoader.
 ↪__load(self)
     79 else:
```

```
     80     try:
---> 81         root = nltk.data.find(f"{self.subdir}/{self.__name}")
     82     except LookupError as e:
     83         try:

File ~\anaconda3\lib\site-packages\nltk\data.py:583, in find(resource_name,␣
↪paths)
    581 sep = "*" * 70
    582 resource_not_found = f"\n{sep}\n{msg}\n{sep}\n"
--> 583 raise LookupError(resource_not_found)

LookupError:
**********************************************************************
  Resource omw-1.4 not found.
  Please use the NLTK Downloader to obtain the resource:

  >>> import nltk
  >>> nltk.download('omw-1.4')

  For more information see: https://www.nltk.org/data.html

  Attempted to load corpora/omw-1.4

  Searched in:
    - 'C:\\Users\\Admin/nltk_data'
    - 'C:\\Users\\Admin\\anaconda3\\nltk_data'
    - 'C:\\Users\\Admin\\anaconda3\\share\\nltk_data'
    - 'C:\\Users\\Admin\\anaconda3\\lib\\nltk_data'
    - 'C:\\Users\\Admin\\AppData\\Roaming\\nltk_data'
    - 'C:\\nltk_data'
    - 'D:\\nltk_data'
    - 'E:\\nltk_data'
**********************************************************************
```

[29]:
```
# POS tagging

# import Word Tokenizer and PoS Tagger
from nltk.tokenize import word_tokenize
from nltk import pos_tag

# sample sentence
sentence = "Taj Mahal is one of the beautiful monument."

# tokenize the sentence
send_tokens = word_tokenize(sentence)
```

```
# create PoS tags
sent_pos = pos_tag(send_tokens)

# print tokens with PoS
print(sent_pos)
```

```
[('Taj', 'NNP'), ('Mahal', 'NNP'), ('is', 'VBZ'), ('one', 'CD'), ('of', 'IN'),
('the', 'DT'), ('beautiful', 'JJ'), ('monument', 'NN'), ('.', '.')]
```

[19]:
```
# recognizing entities

# import spacy
import spacy

# load English model for tokenizer, tagger, parser, and NER
nlp = spacy.load("en_core_web_sm")

# sample paragraph
paragraph = """Taj Mahal is one of the beaufiful monuments. It is one of the␣
 ↪wonders of the world. It was built by Shah Jahan in 1631 in memory of his␣
 ↪third beloved wife Mumtaj Mahal."""

# create nlp Object to handle linguistic annotations in documents
docs = nlp(paragraph)
entities=[(i.text, i.label_) for i in docs.ents]
print(entities)
```

```
[('Taj Mahal', 'ORG'), ('Shah Jahan', 'ORG'), ('1631', 'DATE'), ('third',
'ORDINAL'), ('Mumtaj Mahal', 'PERSON')]
```

[20]:
```
# import display for visualization the Entities
from spacy import displacy

# visualize the entities using render function
displacy.render(docs, style="ent", jupyter=True)
```

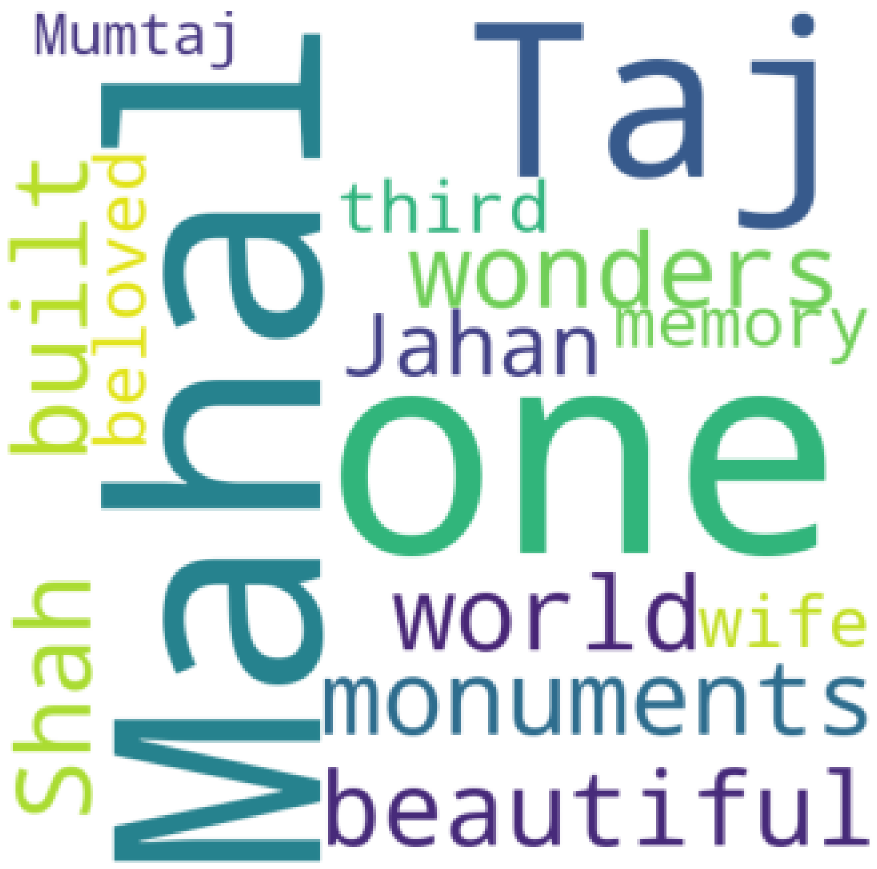
```
<IPython.core.display.HTML object>
```

[21]:
```
# dependency parsing
# import spacy
import spacy

# laod English model for tokenizer, tagger, parser, and NER
nlp = spacy.load("en_core_web_sm")

# create nlp Object to handle linguistic annotations in a documents
```

```
docs = nlp(sentence)

# visualize the using render function
displacy.render(docs, style="dep", jupyter=True, options={'distance':150})
```

```
<IPython.core.display.HTML object>
```

[22]:
```
# creating a word cloud

# importing all necessary modules
from wordcloud import WordCloud
from wordcloud import STOPWORDS
import matplotlib.pyplot as plt

stopword_list = set(STOPWORDS)

paragraph = """Taj Mahal is one of the beautiful monuments. It is one of the␣
  ↪wonders of the world. It was built by Shah Jahan in 1631 in memory of his␣
  ↪third beloved wife Mumtaj Mahal."""

word_cloud = WordCloud(width=550, height=550, background_color='white',
                      stopwords=stopword_list,
                      min_font_size=10).generate(paragraph)
# visualize the WordCloud Plot
# set wordcloud figure size
plt.figure(figsize=(8,6))

# show image
plt.imshow(word_cloud)

# remove Axis
plt.axis("off")

# show plot
plt.show()
```

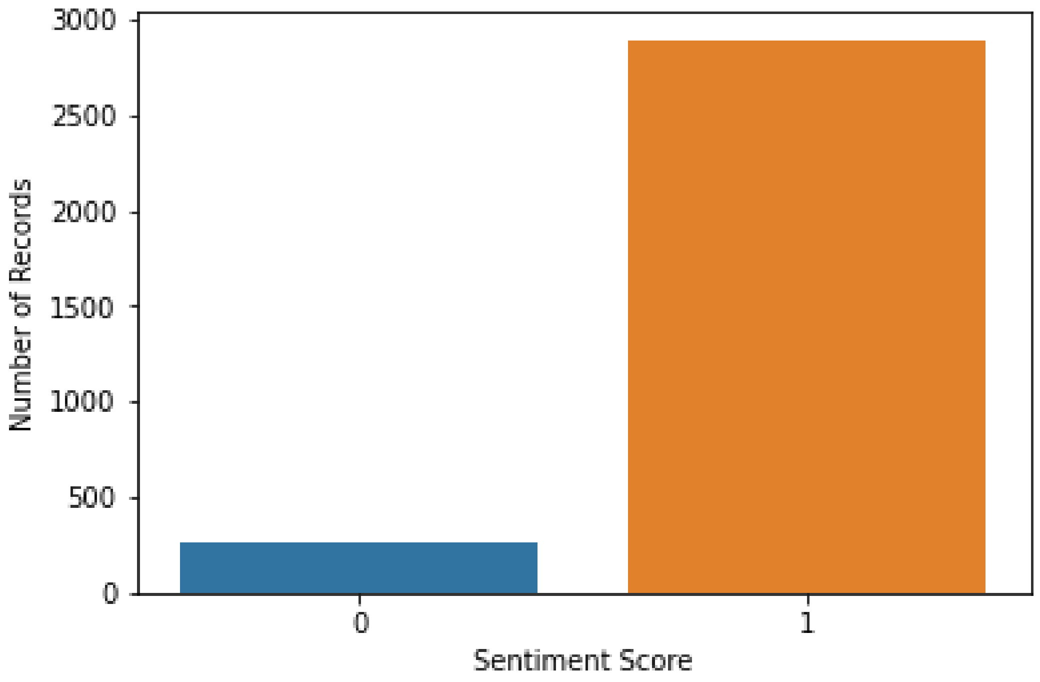


```
[1]: # import libraries
     import pandas as pd

     # read the dataset
     df = pd.read_csv('amazon_alexa.tsv', sep='\t')

     # show top 5-records
     df.head()
```

```
[1]:    rating       date         variation  \
     0       5  31-Jul-18  Charcoal Fabric
     1       5  31-Jul-18  Charcoal Fabric
     2       4  31-Jul-18    Walnut Finish
     3       5  31-Jul-18  Charcoal Fabric
     4       5  31-Jul-18  Charcoal Fabric

                                         verified_reviews  feedback
     0                                      Love my Echo!         1
     1                                          Loved it!         1
```

```
2  Sometimes while playing a game, you can answer…         1
3  I have had a lot of fun with this thing. My 4 …         1
4                                              Music         1
```

[2]:
```python
# import seaborn
import seaborn as sns
import matplotlib.pyplot as plt

# count plot
sns.countplot(x='feedback', data=df)

# set X-axis and Y-axis labels
plt.xlabel('Sentiment Score')
plt.ylabel('Number of Records')

# show the plot using show() function
plt.show()
```



[4]:
```python
# import CountVectorizer and RegexTokenizer
from nltk.tokenize import RegexpTokenizer
from sklearn.feature_extraction.text import CountVectorizer

# create Regex tokenizer for removing special symbols and
# numeric values
```

```
regex_tokenizer = RegexpTokenizer(r'[a-zA-Z]+')

# initilize CountVectorizer object
count_vectorizer = CountVectorizer(lowercase=True,
                                   stop_words='english',
                                   ngram_range=(1,1),
                                   tokenizer=regex_tokenizer.tokenize)
# fit and transform the dataset
count_vectors = count_vectorizer.fit_transform(df['verified_reviews'])
```

[11]:
```
# import train_text_split
from sklearn.model_selection import train_test_split

# partition data into trainig and testing set
feature_train, feature_test, target_train, target_test =␣
 ↪train_test_split(count_vectors, df['feedback'], test_size=0.3,
                random_state=1)
```

[12]:
```
# import logistic regression scikit-learn model
from sklearn.linear_model import LogisticRegression

# create logistic regression model object
logreg = LogisticRegression(solver='lbfgs')

# fit the model with data
logreg.fit(feature_train, target_train)

# forecast the target variable for given test dataset
predictions = logreg.predict(feature_test)

# import metrics module from perfomance evaluation
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score

# assess model performance using accuracy measure
print("Logistic Regression Model Accuracy: ", accuracy_score(target_test,␣
 ↪predictions))

# calculate model prediction
print("Logistic Regression Model Precision: ", precision_score(target_test,␣
 ↪predictions))

# calculate model recall
print("Logistic Regression Model Recall: ", recall_score(target_test,␣
 ↪predictions))
```

```
# calculate model f1 score
print("Logistic Regression Model F1-Score: ", f1_score(target_test,␣
 ↪predictions))
```

```
Logistic Regression Model Accuracy:  0.9428571428571428
Logistic Regression Model Precision:  0.952433628318584
Logistic Regression Model Recall:  0.9873853211009175
Logistic Regression Model F1-Score:  0.9695945945945945
```

[13]:
```
## Classification using TF-IDF

# import libraries
import pandas as pd

# read the dataset
df = pd.read_csv('amazon_alexa.tsv', sep='\t')

# show top 5-records
df.head()
```

[13]:
```
   rating       date         variation  \
0       5  31-Jul-18  Charcoal Fabric
1       5  31-Jul-18  Charcoal Fabric
2       4  31-Jul-18    Walnut Finish
3       5  31-Jul-18  Charcoal Fabric
4       5  31-Jul-18  Charcoal Fabric

                                    verified_reviews  feedback
0                                      Love my Echo!         1
1                                          Loved it!         1
2  Sometimes while playing a game, you can answer…         1
3  I have had a lot of fun with this thing. My 4 …         1
4                                              Music         1
```

[20]:
```
# rest in the book
```

[ ]:
```
```