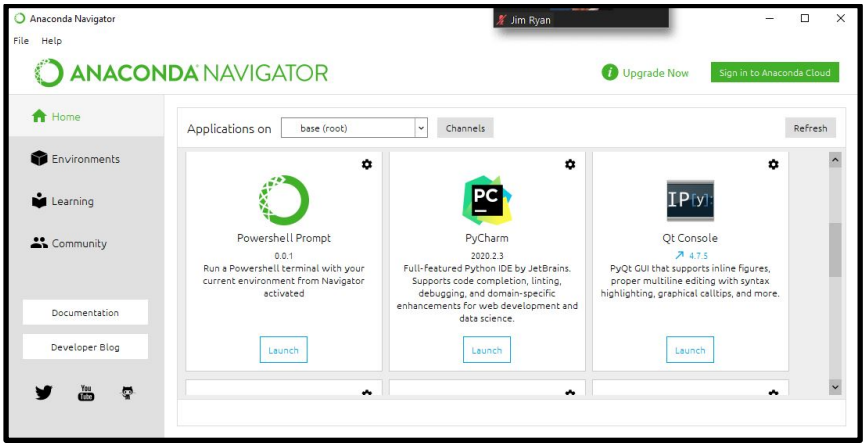
Natural Language Processing (NLP)

Text mining continued via NLP

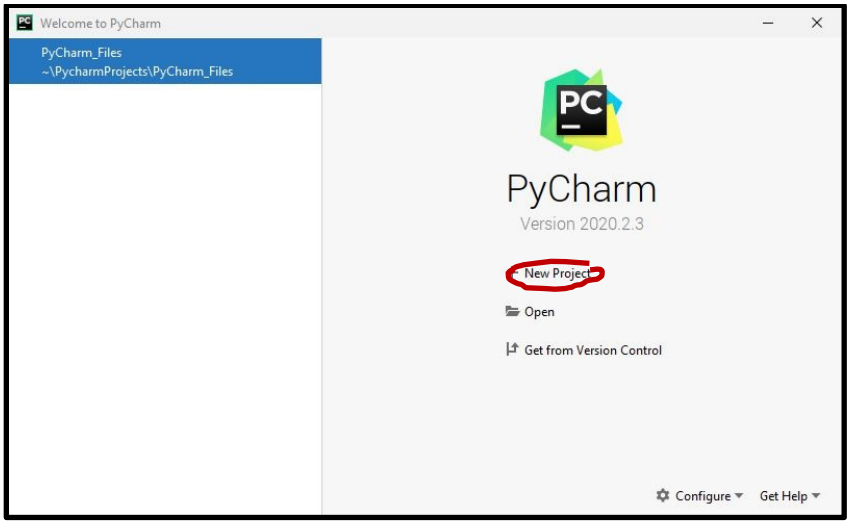
As we saw in Tutorial Eight-B, text mining is a subset of data mining and it implies analyzing data patterns in large batches using one or more software applications. With respect to this tutorial, we will continue our focus on Python data mining processes for text mining via Natural Language Processing (NLP). Text Mining analyzes the text itself, while NLP deals with the underlying/latent metadata to answer questions like - What is the sentiment? - What are the keywords? (using POS tagging & parsers) - What category of content it falls under? - Which are the entities in the sentence? In Tutorial Ten, we will introduce NLP across five sections of Python code in PyCharm: (1) Data Cleaning, (2) Count Vectorizer | TFIDF, (3) Reading a Data Frame using Pandas, and (4) NLP example with Spam data. For more information on NLP outside of this tutorial, please refer to the following URL: https://medium.com/towards-artificial-intelligence/natural-language-processing-nlp-with-python-tutorial-for-beginners-1f54e610a1a0

Create the NLP Project in PyCharm

0) To ensure your various Python package libraries are available to import, be sure to use Anaconda Navigator to launch PyCharm.

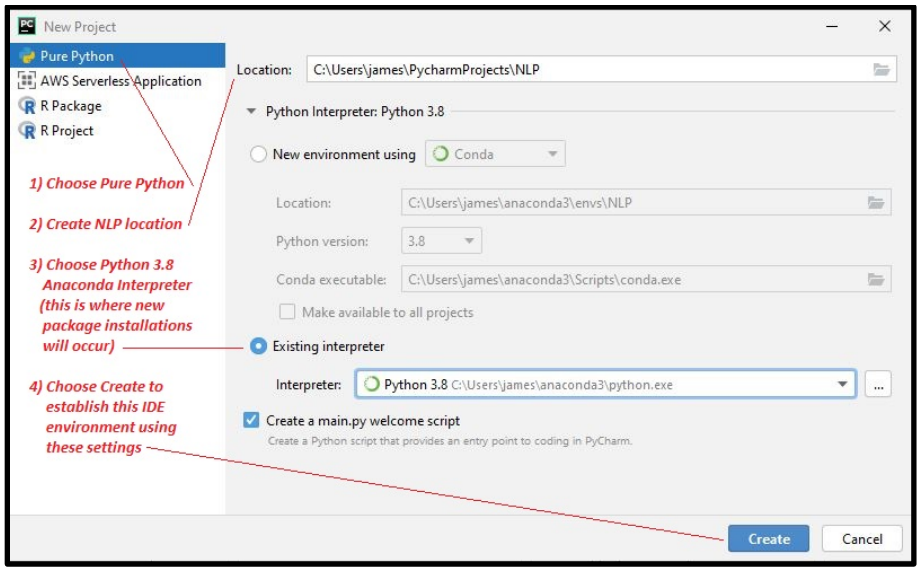


1) Launch PyCharm and choose a New Project.



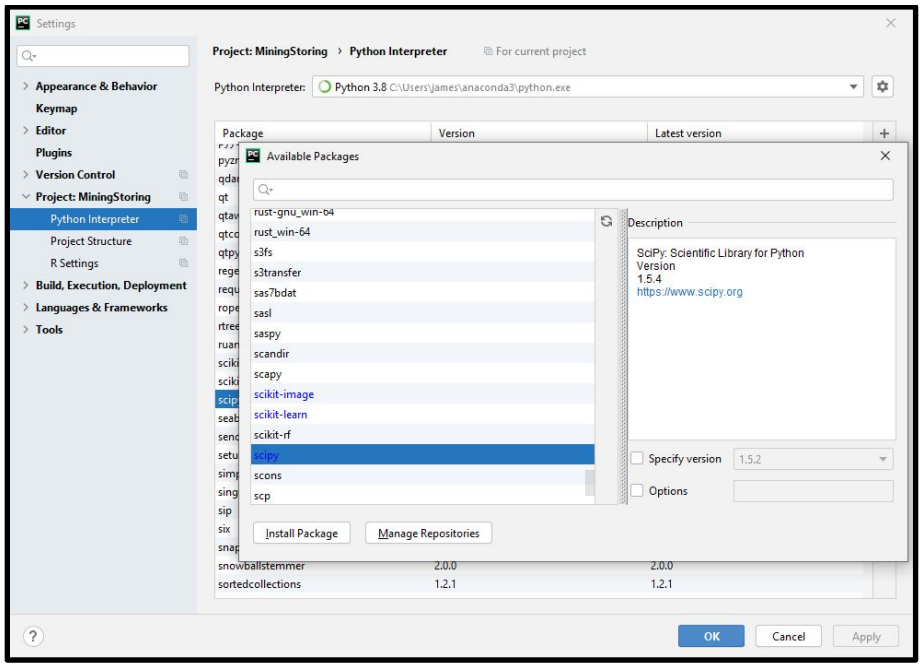
1) Choose a Pure Python project,

2) Change your project location to create a NLP folder
3) Be sure to choose the Python interpreter as Python 3.8 associated with Anaconda.
4) Choose Create to open PyCharm with these settings.



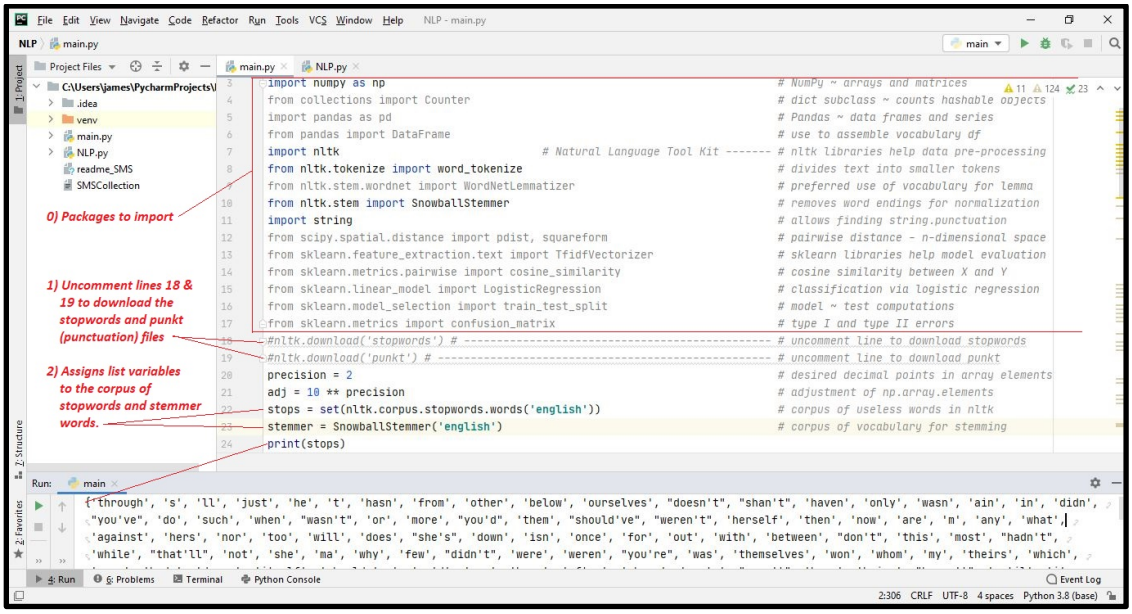
5) *Download* the SMSCollection and readme_SMS from the Week 12 Canvas Module to the NLP directory created in the last step. The readme_SMS file explains the data in the SMSCollection file.

6) *When importing a new package library, check the Python Interpreter to ensure the package is installed via Files->Settings->Python Interpreter or left-click Python3.8 in the lower right screen corner choosing Interpreter Settings...* Once in the Python Interpreter, pan down to find the particular Python package library and double click on package version. In the resulting window, all installed packages are highlighted in blue.



Data Cleaning

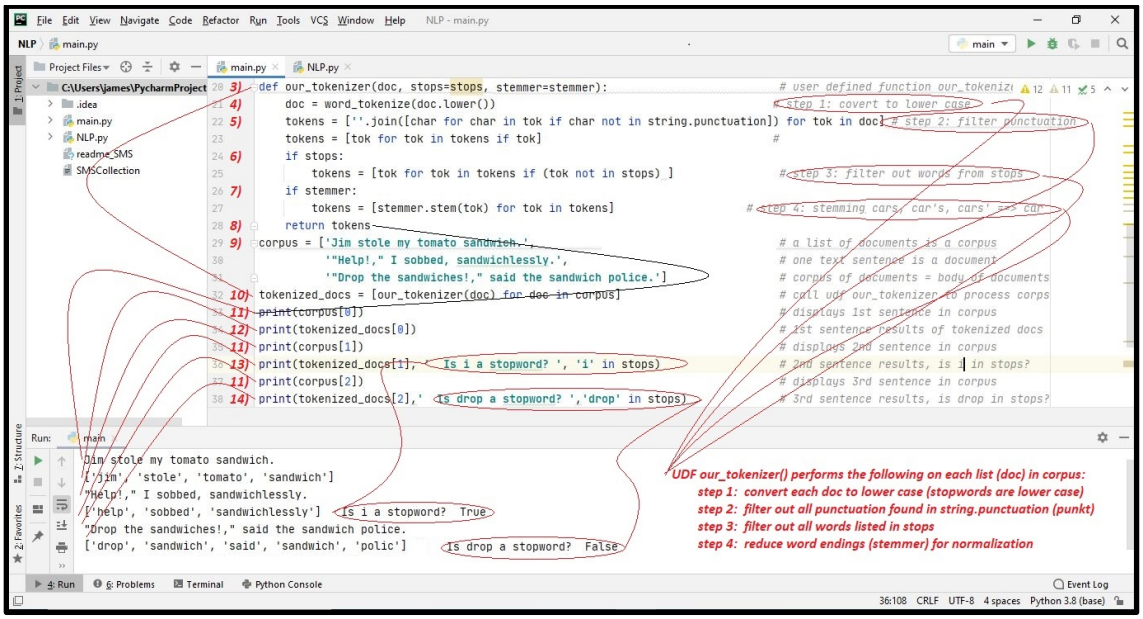
In this section of Python code, we will use the Natural Language Took Kit (NLTK) as well as the string packages. The PyCharm screenshot below lists Python code for: 0) all the packages used in Tutorial Ten, 1) downloading stopwords and punkt files from NLTK, and 2) assigns the stops list variable to a corpus of English stopwords from NLTK found in the downloaded stopwords file. The last line of code on the screenshot displays all the stops listed in the corpus of stopwords file from NLTK. The Python download code for stopwords and punkt is commented out and the '#" symbol should be removed once to download the files to the Python code directory. Running the download code on lines 17 and 18 more than once will result in unwanted verification notices the files are up-to-date. The punkt file lists punctuation marks to filter out using strings.punctuation in the UDF our_tokenizer(). We introduce stemming and lemmatization in the data cleaning package imports below. For more information on these two topics, please refer to the following URL:
https://www.datacamp.com/community/tutorials/stemming-lemmatization-python



All visualizations may be viewed at 400% magnification for clarity.

The next set of Python code in the last data cleaning PyCharm screenshot establishes an user defined function (UDF) named our_tokenizer() that takes each list as a document from the corpus list variable we assign to three sentences and tokenizes each word (doc) in each sentence (corpus[N]) while filtering out punctuation and words listed in stops list of stopwords. These actions clean the corpus data lists and prepares the vocabulary words for vectorization. Specifically, the Python code (3) defines the UDF our_tokenizer() that has parameters list variable, stops, and stemmer; (4) step 1 in the UDF is to convert each doc (word) to lower case; (5) filter out punctuation in each doc; (6) filter out words found in the stops list variable; (7) use stemmer.stem() to reduce a word's conjugation toward normalization; (8) return the tokenized list of words; (9) assigns a list variable corpus to three sentences; (10) call the UDF our_tokenizer for each sentence in corpus; (11) display a corpus sentence; (12) display the

tokenized words; (13) display the tokenized words and check if 'I' is among the stopwords; and (14) display the tokenized words and check if 'drop' is among the stopwords.



Exercise deliverable: Review the Python code in the prior two PyCharm screenshots. *All the Python code listed in these PyCharm screenshots may be downloaded in the **Week 12 file NLP.py**.* Replicate the code exactly as listed, especially the imports and UDF, but create your own sentences to use in the list variable corpus. Capture a PyCharm screenshot showing each sentence of your corpus, followed by the tokenized words.

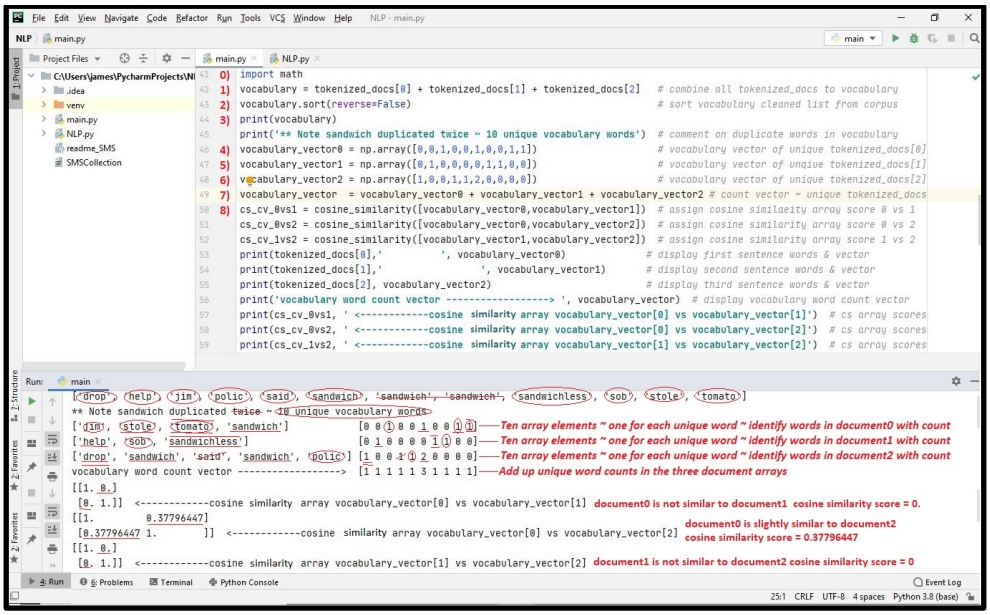
Count Vectorizer | TFIDF

In this section of Python code, we use the tokenized words (tokenized_docs) to create a sorted vocabulary list of ten unique words. The ten unique words in the sorted vocabulary list are:

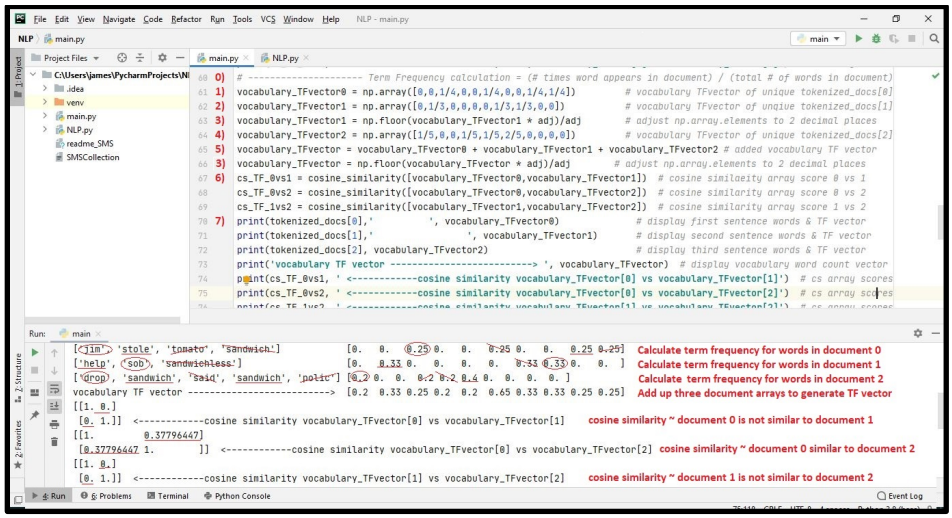
['drop', 'help', 'jim', 'polic', 'said', 'sandwich', '~~sandwich~~', '~~sandwich~~', 'sandwichless', 'sob', 'stole', 'tomato']

The vocabulary list is the basis for creating NumPy one-dimensional arrays (count vector) that reflect the occurrence of tokenized words in the three documents, where a document is a sentence in the corpus. The Python code adds up the three document arrays to form a count vector. Next, the Python code formulates a term frequency (TF) vector, inverse document frequency (IDF) vector, and their product TFIDF vector. The Python code yields NumPy one-dimensional arrays (vectors) of scores that we can load into many different machine learning algorithms. The different vectors reflect uniqueness of words in the documents and the cosine similarity function from SciKit-Learn Metrics Pairwise package library measures the Euclidean distance between any two of these NumPy array vectors. The following PyCharm screen shot shows Python code for: (0) importing the math libraries for logarithms; (1) creating a vocabulary list by adding the document0, document1, and document2 tokenized_docs lists together; (2) sorting the new vocabulary list; (3) printing the lists and/or arrays; (4) creating a ten element NumPy array for unique words in document0; (5) creating a ten element NumPy array for unique words in document1; (6) creating a ten element NumPy array for unique words in document2; (7) creating a ten element NumPy array that sums the prior three unique word count arrays; and (8) calculating the cosine similarity scores for the three unique word

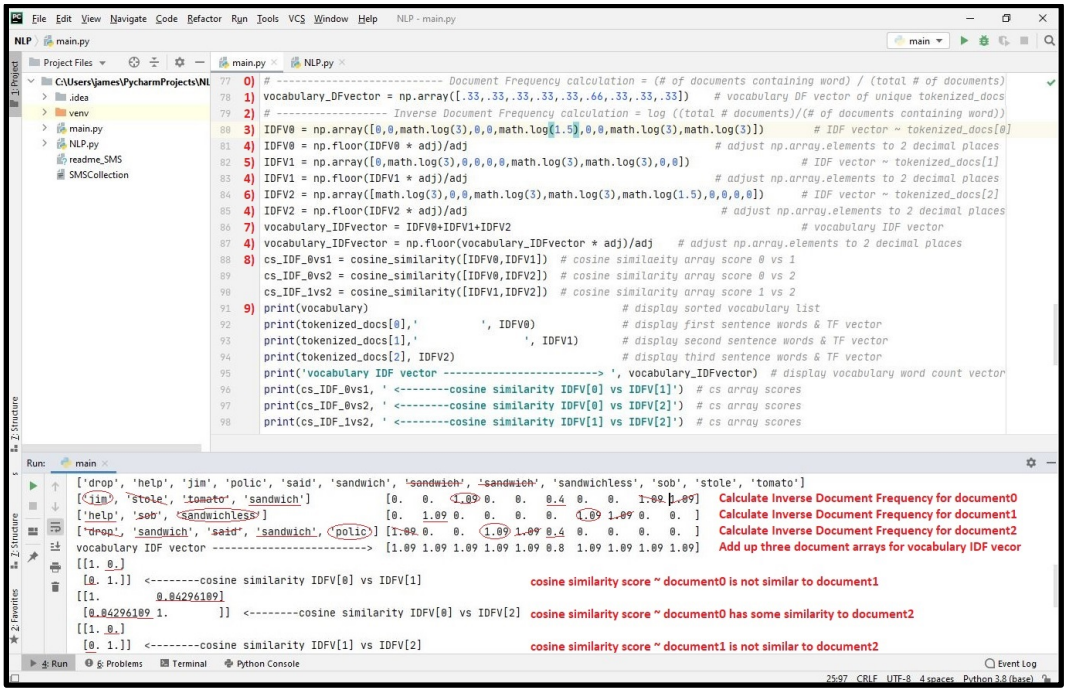
count arrays (vectors) from the three documents (e.g., 0 vs 1, 0 vs 2, and 1 vs 2). Note document 0 and document 2 have similarity, while the other document combos do not.



Upon calculating the vocabulary word count vector, our next steps are to calculate the term frequency (TF) vectors, the inverse document frequency (IDF) vectors, and use these two vector arrays to calculate the TFIDF vectors. The PyCharm screenshot below contains the Python code to create the TF vectors. The Python code specifically executes: (0) comment statement for the formula to calculate TF vectors for each unique word in each document, (1) assign TF vector array elements for tokenized_docs[0], (2)assign TF vector array elements for tokenized_docs[1], (3) adjust the np.array.elements to two decimal places, (4) assign TF vector array elements for tokenized_docs[2], (5) sum the three document TF arrays to create the vocabulary_TFvector array, (6) calculate the cosine similarity scores comparing the three document TF vector arrays (e.g., 0 vs 1, 0 vs 2, and 1 vs 2), and (7) print out TF results. Note document 0 and document 2 continue the similarity scores while the other document combinations do not.

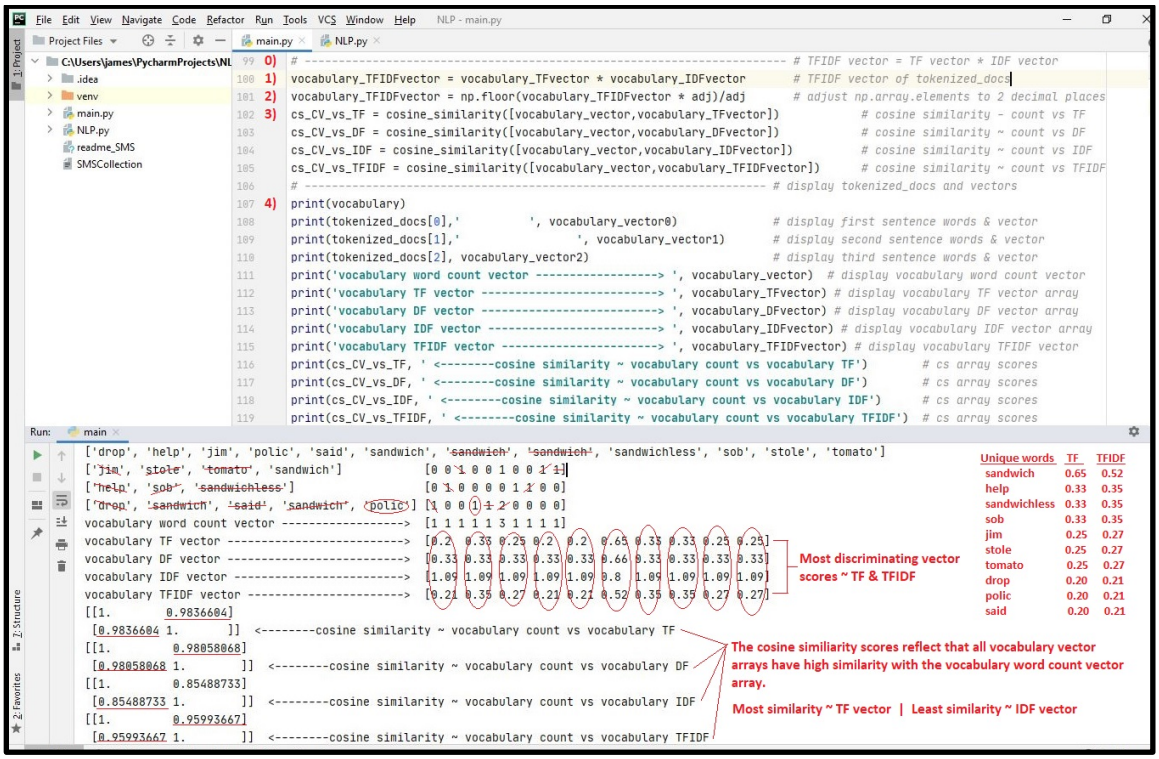


The following PyCharm screenshot contains the Python code to calculate the document frequency (DF) vector and inverse document frequency (IDF) vectors of the tokenized_docs. Specifically, the Python code executes: (0) a comment statement on the formula to calculate the DF vector array, (1) assigns a NumPy array to vocabulary_DFvector, (2) a comment statement on the formula to calculate the IDF vector arrays, (3) assign IDF vector array elements for tokenized_docs[0], (4) adjust the np.array.elements to two decimal places, (5) assign IDF vector array elements for tokenized_docs[1], (6) assign IDF vector array elements for tokenized_docs[2], (7) sum the three document IDF arrays to create the vocabulary_IDFvector array, (6) calculate the cosine similarity scores comparing the three document IDF vector arrays (e.g., 0 vs 1, 0 vs 2, and 1 vs 2), and (7) print out IDF results. Note document 0 and document 2 continue to have some similarity scores while other document combinations do not.



The last PyCharm screenshot for count vectorization identifies Python code to calculate the vocabulary vector array for TFIDF, calculate cosine similarity scores between combinations of the initial vocabulary word count vector and each derived vocabulary vector array (e.g., word count vs TF, word count vs DF, word count vs IDF, and word count vs TFIDF), and then display the results in a summary of count vectorization outcomes and associated cosine similarity. Specifically, the Python code executes: (0) a comment statement on the formula to calculate the TFIDF vector array, (1) calculates the vocabulary TFIDF vector array by multiplying the vocabulary TF vector by the vocabulary IDF vector, (2) adjust the np.array.elements to two decimal places, (3) calculate the cosine similarity scores comparing the vocabulary word count vector array in combination with the derived vocabulary vector arrays (e.g., TF, DF, IDF, and TFIDF), and (4) print out count vectorization and similarity summary results. The annotation on the PyCharm screenshot identifies the uniqueness associated with each tokenized_doc word, based on vocabulary vector arrays for TF and TFIDF. TF and TFIDF vocabulary vector arrays had the most discriminating array elements, where higher is better. Sandwich was the most unique word and said was the least unique word. All four derived vocabulary vector arrays (e.g., TF,

DF, IDF, and TFIDF) show strong similarity scores with TF having the highest and IDF having the lowest.



Exercise deliverable: Review the Python code in the prior four PyCharm screenshots. All the Python code listed in these PyCharm screenshots may be downloaded in the Week 12 file NLP.py. Replicate the code exactly as listed, especially the imports, but use the tokenized_docs from your prior exercise deliverable to build your vocabulary list. Capture a PyCharm screenshot similar to the one above where you summarize your findings.

Reading a text file into a Pandas DataFrame:

We learned about Pandas in prior Tutorials. Again in this tutorial we will look at Data Frames and we will use the SMSCollections.csv file from the text mining example in Tutorial Eight-B. For more specific details about Pandas not covered in this tutorial, please refer to the following URL:

https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.read_table.html

1) Anaconda will load the libraries in PyCharm. All the Python code listed in the following PyCharm screenshots may be found in the downloaded Week 12 file NLP_Spam.py. Copy the Python code for the library imports, which are:

```
import numpy as np
import pandas as pd
import nltk
import string
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
from sklearn.linear_model import LogisticRegression
```

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
#nltk.download('stopwords') # uncomment to download stopwords
#nltk.download('punkt') # uncomment to download punkt
```

2) The following Python code formats the run window to handle the size of the csv data file we will be using in Tutorial Ten. You will see this code in the file NLP_Spam.py

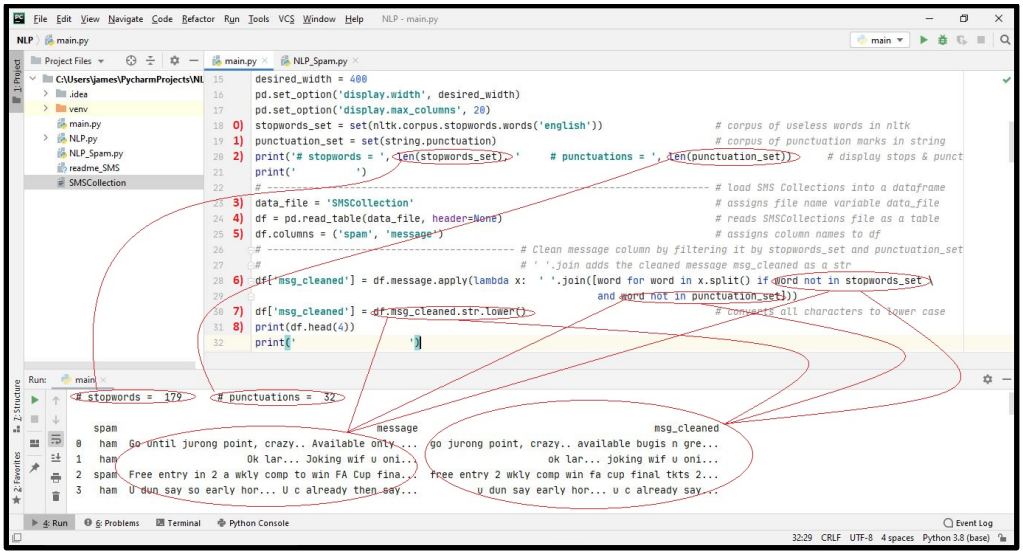
```
desired_width = 400
pd.set_option('display.width', desired_width)
pd.set_option('display.max_columns', 20)
```

3) The following Python code reads the data file SMSCollection into a pandas dataframe as a table. The table does not have a header row, so columns labels should be assigned. The Python code is available as file NLP_spam.py with Tutorial Ten.

```
data_file = 'SMSCollection'                    # assigns var data_file
df = pd.read_table(data_file, header=None)     # reads file as a table
df.columns = ('spam', 'message')
```

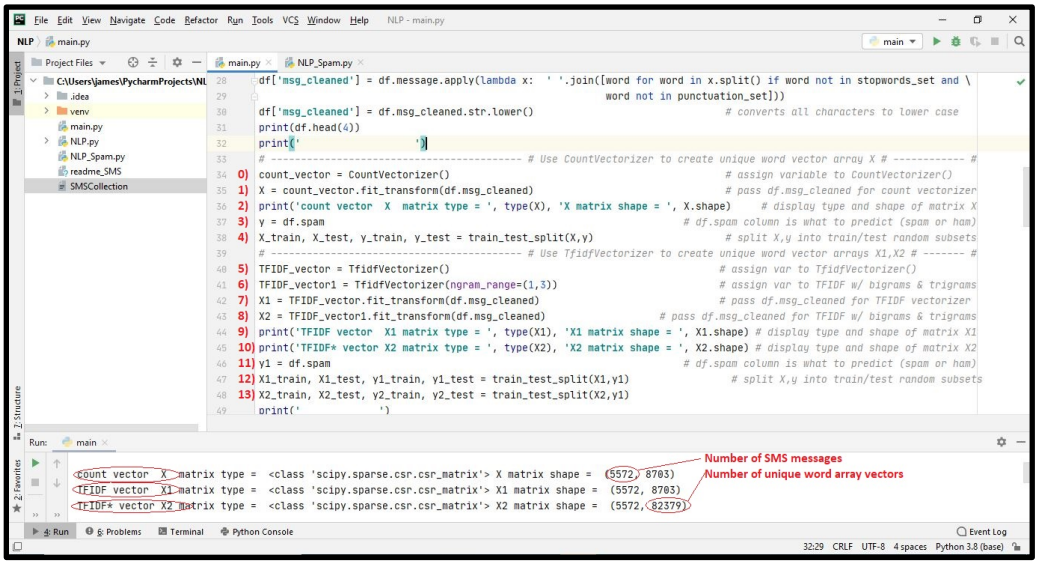
NLP Example with Spam:

We will continue working with the SMSCollection file we used for the text mining example in Tutorial Eight-B. However, in this Tutorial Ten example we will clean the SMS messages by filtering out the stopwords and punctuation in each SMS message to then use vectorization libraries from sklearn to generate vector arrays (X) to classify the SMS message as spam or ham (y). In this example, we use logistic regression for classification as well as introducing two new classifier functions in RandomForestClassifier() as well as well as GradientBoostingClassifier(). We split the vector arrays (X) using train_test_split from sklearn and use a confusion matrix to review the type I and type II errors. In this NLP example, we tweak NLP classification models by looking at three different vectorization methods combined with three different classification tools. The PyCharm screenshots depict examples of Python code for data cleaning, vectorization, random selection of X,y data points for model training and testing, as well as classification R2 scores and associated type I/type II errors. All Python code in the PyCharm screenshots is available in file NLP_Spam.py.



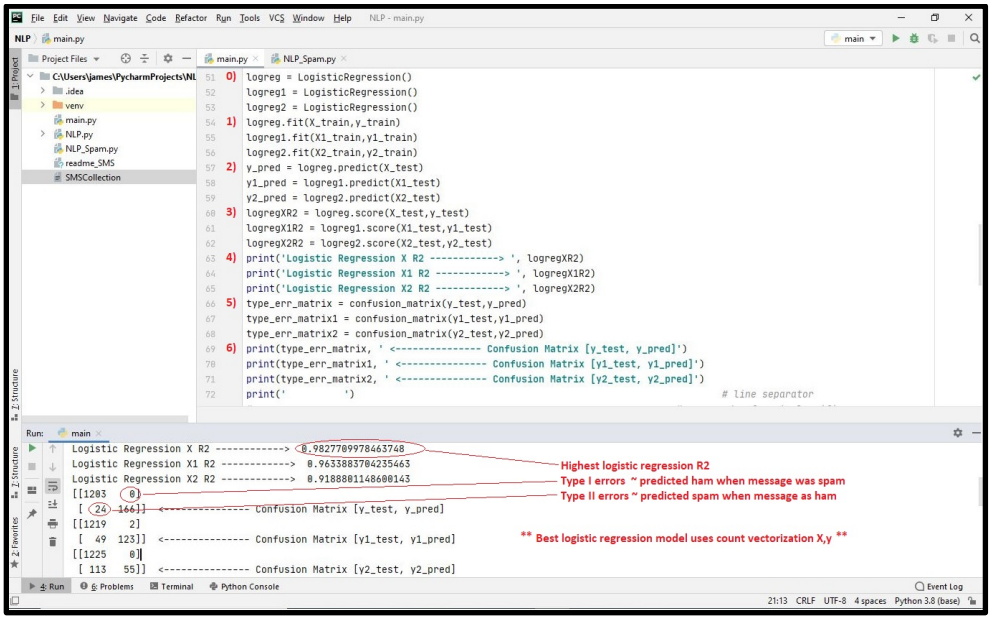
1. ***Data cleaning***: In the preceding PyCharm screenshot, the Python code execution (0) assigns a variable to the set of stopwords from nltk, (1) assigns a variable to the set of punctuation marks from string, (2) displays the number of stopwords in the set and the number of punctuation marks, (3) assigns a file name variable to SMSCollection, (4) reads the file datafile (i.e., SMSCollection) as a table into a Pandas dataframe df, (5) assigns column names of spam and message to the dataframe table df, (6) uses the lambda function to create a new df column msg_cleaned that uses the df column message as the basis and then filters out the stopwords and punctuation, (7) converts each row of the df column msg_cleaned to lower case, and (8) displays the first four rows of the Pandas dataframe df.

2. ***Vectorization***: The PyCharm screenshot below shows Python code that executes (0) assigning a variable to the CountVectorizer() function; (1) transforms the df column msg_cleaned into unique word count vectorization and stores it in a NumPy matrix X; (2) displays the type and shape of X; (3) assigns the df column spam as y; (4) randomly splits X and y into model X/y training and test subsets; (5) assigning a variable to the TFIDFVectorizer() function; (6) assigning a variable to the TFIDFVectorizer() function that uses tokenized words, double words, and triple words; (7) transforms the df column msg_cleaned into unique word TFIDF vectorization and stores it in a NumPy matrix X1; (8) transforms the df column msg_cleaned into unique word TFIDF vectorization using bigrams/trigrams and stores it in a NumPy matrix X2; (9) displays the type and shape of X1; (10) displays the type and shape of X2; (11) assign the df column spam rows as y1; (12) randomly splits X1 and y1 into model X1/y1 training and test subsets; and (13) randomly splits X2 and y1 into model X2/y2 training and test subsets For more information on bigrams and trigrams, refer to the following URL: https://www.geeksforgeeks.org/tf-idf-for-bigrams-trigrams/

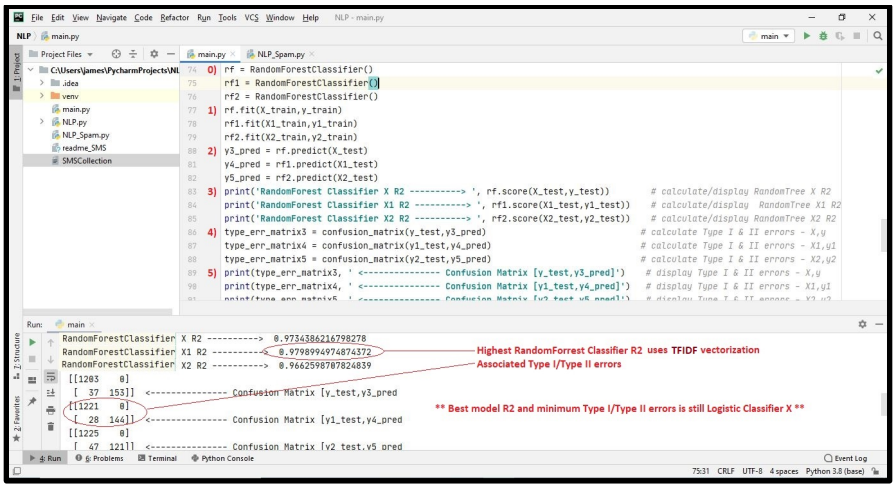


3. ***Logistic Regression Classification:*** In the following PyCharm screenshot, the Python code execution (0) assigns three unique variables the LogisiticRegression() function; (1) fits the unique word count vectors (X,y), unique word TFIDF vectors

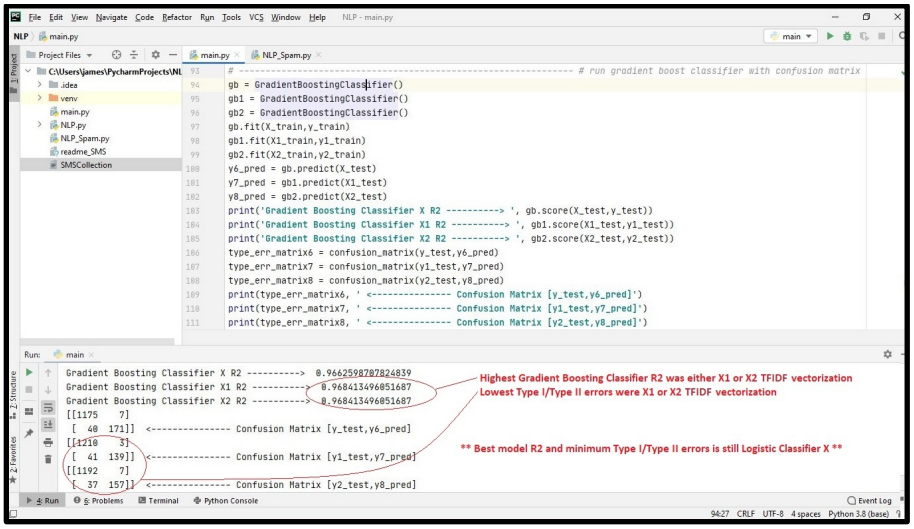
(X1,y1), and unique word TFIDF vectors using bi/trigrams (X2,y2) using the training data sets; (2) predicts the ham/spam classifications using the three array vectors test subsets; (3) calculates the R2 score for each array vector test subset; (4) displays each models R2 score; (5) calculates the confusion matrix for each array vector predictions; and (6) displays the confusion matrix of type I and type II errors for the array vector predictions. Note: All three of the array vector models yielded high R2 logistic regression scores. However, the highest logistic regression classification R2 score with the minimum type I and type II errors is matrix X that uses the unique word count vectorization.



4. ***Random Forest Classification***: In the following PyCharm screenshot, the Python code execution (0) assigns three unique variables the RandomForestClassifier() function; (1) fits the unique word count vectors (X,y), unique word TFIDF vectors (X1,y1), and unique word TFIDF vectors using bi/trigrams (X2,y2) using the training data sets; (2) predicts the ham/spam classifications using the three array vectors test subsets; (3) calculates the R2 score for each array vector test subset; (4) displays each models R2 score; (5) calculates the confusion matrix for each array vector predictions; and (6) displays the confusion matrix of type I and type II errors for the array vector predictions. Note: All three of the array vector models yielded high R2 random forest classification scores. However, the highest random forest classification R2 score with the minimum type I and type II errors is matrix X2 that uses the unique word TFIDF vectorization. The best model with the highest R2 score and lowest Type I/Type II errors is still the unique word count vectorization using logistic regression classification.



5. ***Gradient Boosting Classification***: In the following PyCharm screenshot, the Python code execution (0) assigns three unique variables the gradient boosting classification() function; (1) fits the unique word count vectors (X,y), unique word TFIDF vectors (X1,y1), and unique word TFIDF vectors using bi/trigrams (X2,y2) using the training data sets; (2) predicts the ham/spam classifications using the three array vectors test subsets; (3) calculates the R2 score for each array vector test subset; (4) displays each models R2 score; (5) calculates the confusion matrix for each array vector predictions; and (6) displays the confusion matrix of type I and type II errors for the array vector predictions. Note: All three of the array vector models yielded high R2 gradient boosting classification scores. However, the highest gradient boosting classification R2 score with the minimum type I and type II errors is matrix X1 or X2 that uses the unique word TFIDF vectorization or unique word TFIDF using bi/trigrams. The best model with the highest R2 score and lowest Type I/Type II errors is still the unique word count vectorization using logistic regression classification.



6. **Exercise deliverable:** Review the vectorization tools of count, TFIDF, and TFIDF with bi/trigrams, as well as classification tools of logistic regression, random forest classification, and gradient boosting classification. Choose one vectorization and one classification tool on application columns of your project data or on the SMSCollection file. Capture the Python code and results for the data cleaning, vectorization, and classification R2 score/confusion matrix on three separate PyCharm screenshots. Save the PyCharm screenshots for use later in the Tutorial.

Deliverables from Exercises Above:

Take the noted deliverables (i.e., PyCharm screenshots) from each exercise above (i.e., total of 8 screenshots), insert the screenshots into a MS Word document as you label each screenshot deliverable from the corresponding exercise, save the document as one PDF, and submit the PDF to the Tutorial Eight-B Canvas Assignment uplink having the following qualities:

a. Your screenshots for each of the four exercise deliverables above is included.
b. Each screenshot is labeled appropriately for each exercise.
    i. Data Cleaning …….. 1 screenshot
    ii. Vectorizer | TFIDF ……. 1 screenshot
    iii. NLP Example with Spam ……. 3 screenshots
    Total PyCharm screenshots to submit …… 5 screenshots
c. All screenshots are legible output for each PyCharm exercise deliverable.