NAME: GANGULA KARTHIK
ADMIN NUMBER: 223715Y
CLASS: AA2202

0. Understanding the Problem

The process of classifying the authors for a given text is called authorship attribution. Each author writes about different topics and has their own style of writing (author fingerprint) which allows for this identification. Applications of this kind of model include plaigarism detection and resolving the disputed authorship of texts.

In the dataset given there are 2 columns: Author (6 authors) and Text This makes it a supervised learning problem since there is data and a assigned label to each text.

In the problem, the cost of false positive and false negatives both carry significant consequences. Therefore, a good model should have a balance of sensitivity and specificity. F1-score would be the ideal metric for the model evaluation

Aim of notebook (part 1 / 2)

This notebook is part 1 of 2 notebooks, it aims to clean the textual data and perform feature engineering on the data. Specifically, text features will be created from the text data which describe the structure of the text. Some of them are number of words, average sentence length, readability of text, etc.

1. Installing and Importing Packages

In [ ]:
```
# %pip install openpyxl --upgrade
# %pip install textstat
```

In [ ]:
```
import re
import pandas as pd
from pprint import pprint
import numpy as np
import seaborn as sns
import string
from nltk.corpus import stopwords
from nltk import bigrams, trigrams, FreqDist
from nltk.stem import WordNetLemmatizer
from nltk.tokenize import MWETokenizer
import matplotlib.pyplot as plt
import contractions
import textstat
from nltk.tokenize import word_tokenize
from collections import Counter
import warnings
from wordcloud import WordCloud, STOPWORDS, ImageColorGenerator
import spacy

nlp = spacy.load("en_core_web_sm") # spacy english model
warnings.filterwarnings("ignore")
sns.set_style("darkgrid")
palette = "cool"
```

2. Data Understanding

In [ ]:
```
df_train = pd.read_excel("Assignment_Data/Data.xlsx")

df_train.head()
```

Out[ ]:

	Text	Author
0	Scoring in PROC DISCRIM is as easy as validati...	AM
1	In the GLM procedure, you may have used LSMEAN...	AM
2	The first problem, accuracy of the data file, ...	AM
3	If the homogeneity of covariance matrices assu...	AM
4	With a CONTRAST statement, you specify L, in t...	AM

In [ ]:
```
df_train.shape
```

Out[ ]:
```
(1922, 2)
```

In [ ]:
```
df_train.info()
# no null values in the dataset
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1922 entries, 0 to 1921
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   Text    1922 non-null   object
 1   Author  1922 non-null   object
dtypes: object(2)
memory usage: 30.2+ KB
```

In [ ]:
```
df_train.isnull().sum() # no null values verified
```

Out[ ]:
```
Text      0
Author    0
dtype: int64
```

In [ ]:
```
# check for duplicates in the text data
duplicate_data = df_train.duplicated(keep="first")

print(duplicate_data.sum(), "duplicate rows are present within the data \n")


display(df_train[duplicate_data].sort_values(by="Text").head(8))
```

```
1106 duplicate rows are present within the data
```

	Text	Author
1367	%distribution(data=&data,out=&report_name,cont...	DM
1419	%distribution(data=&data,out=&report_name,cont...	DM
1414	%generate_grouping(from=work.profile_codes,val...	DM
1243	%generate_grouping(from=work.profile_codes,val...	DM
1362	*\tTemporal infidelity occurs when model input...	DM
1370	*\tTemporal infidelity occurs when model input...	DM
248	*\texamining group differences on predictor va...	AM
44	*\texamining group differences on predictor va...	AM

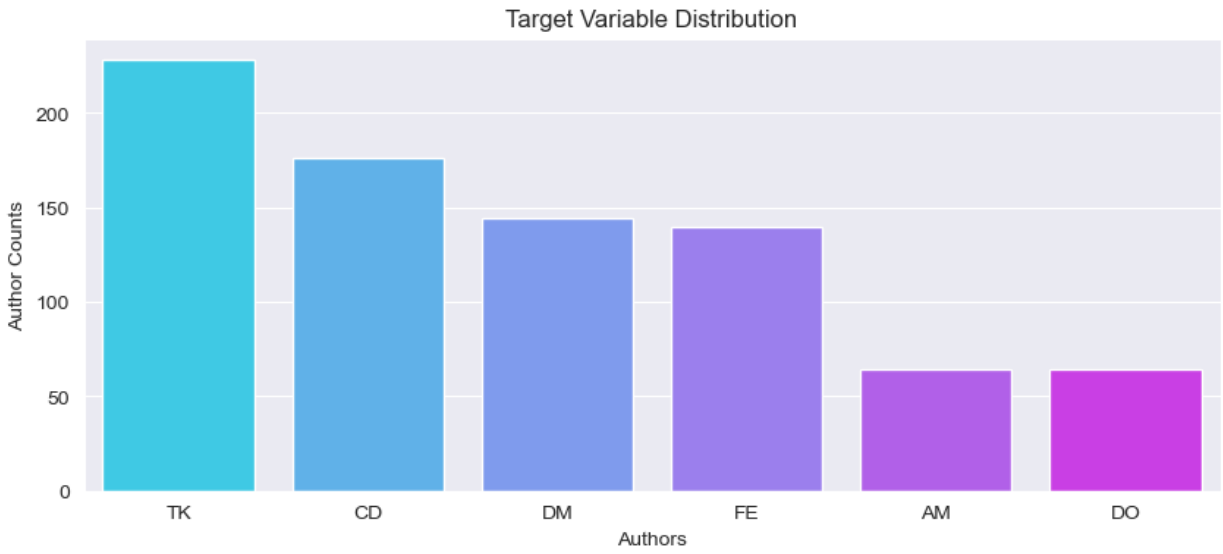
In [ ]:
```
# remove the duplicate rows within the dataset
df_train.drop_duplicates(inplace=True)
display(df_train.shape)
```

```
(816, 2)
```

There's quite a lot of duplicated rows present. These will be unhelpful for training the model and need to be removed.

3. Feature Engineering

In [ ]:
```
plt.figure(figsize=(10, 4))
sns.countplot(data=df_train, x="Author", order=df_train["Author"].value_counts
plt.title("Target Variable Distribution");
plt.ylabel("Author Counts");
plt.xlabel("Authors");
plt.xticks();
```



Data is imbalanced. There are 2 ways that we can handle this:
1. Perform a combination of oversampling and undersampling techniques
2. Make use of a metric that accounts for the class imbalance. In our case we will look at the weighted average f1 score as it accounts for this.
For simplicity, the second option will be considered.

In [ ]:
```
df_train["text_length"] = df_train.Text.apply(len)
df_train.head()
```

Out[ ]:

	Text	Author	text_length
0	Scoring in PROC DISCRIM is as easy as validati...	AM	215
1	In the GLM procedure, you may have used LSMEAN...	AM	782
2	The first problem, accuracy of the data file, ...	AM	990
3	If the homogeneity of covariance matrices assu...	AM	934
4	With a CONTRAST statement, you specify L, in t...	AM	1490

In [ ]:
```
plt.figure(figsize=(10, 6))
ax = sns.violinplot(data=df_train, x="Author", y="text_length", element="poly"
ax.set_title('Distribution of Text Lengths by Author');
ax.set_ylabel('Text Length');
```

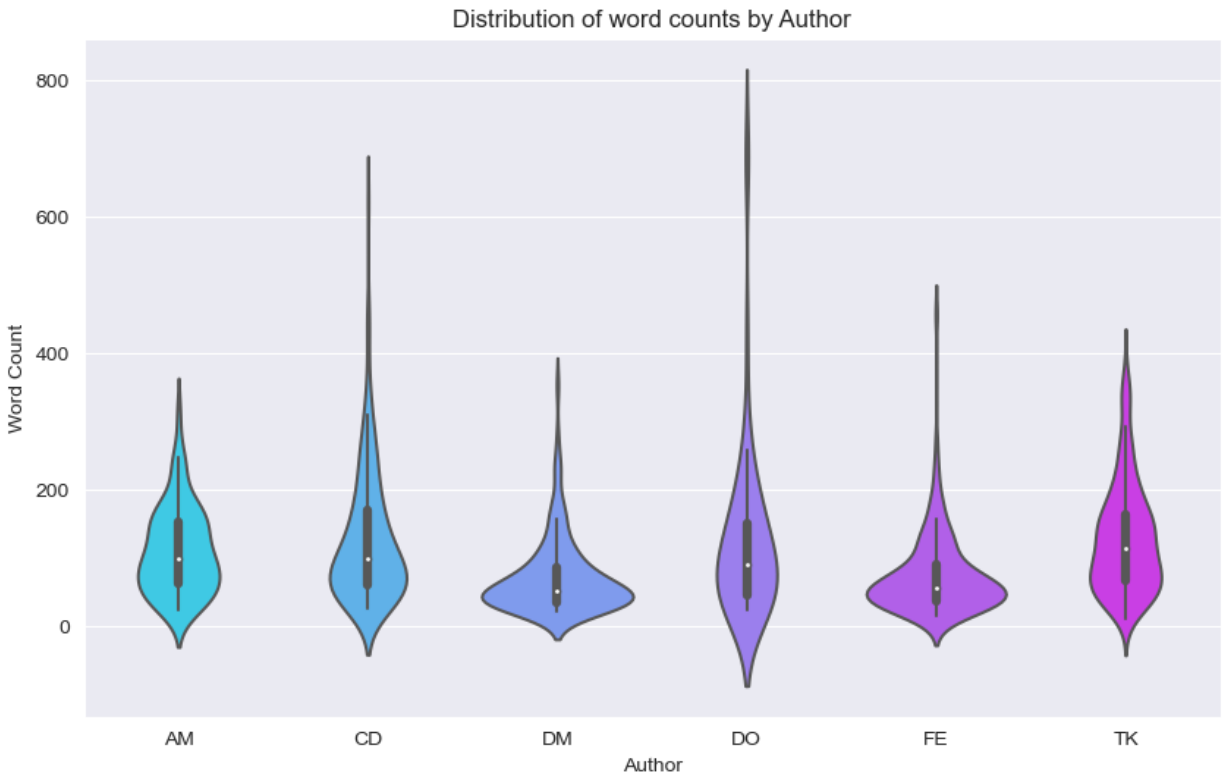
--> Text length is a feature used to indicate the depth of discussion and detail in the text. Longer texts might cover topics more thoroughly, while shorter texts could be more concise.
--> Shows that some authors like AM and DM tend to write shorter texts overall. This could be a distinguishing feature in the model.
--> Looks like majority of the text lengths are around 400 - 500 characters with slight differences.
--> This is not a strong feature in the classification, other features need to be looked at.

In [ ]:
```
# function to count the number of words inside a sentence
df_train["word_count"] = df_train["Text"].apply(lambda sent: len(sent.split(" '

df_train.head()
```

Out[ ]:

	Text	Author	text_length	word_count
0	Scoring in PROC DISCRIM is as easy as validati...	AM	215	37
1	In the GLM procedure, you may have used LSMEAN...	AM	782	129
2	The first problem, accuracy of the data file, ...	AM	990	159
3	If the homogeneity of covariance matrices assu...	AM	934	146
4	With a CONTRAST statement, you specify L, in t...	AM	1490	247

In [ ]:
```
plt.figure(figsize=(10, 6))
ax = sns.violinplot(data=df_train, x="Author", y="word_count", element="poly",
ax.set_title('Distribution of word counts by Author');
ax.set_ylabel('Word Count');
```



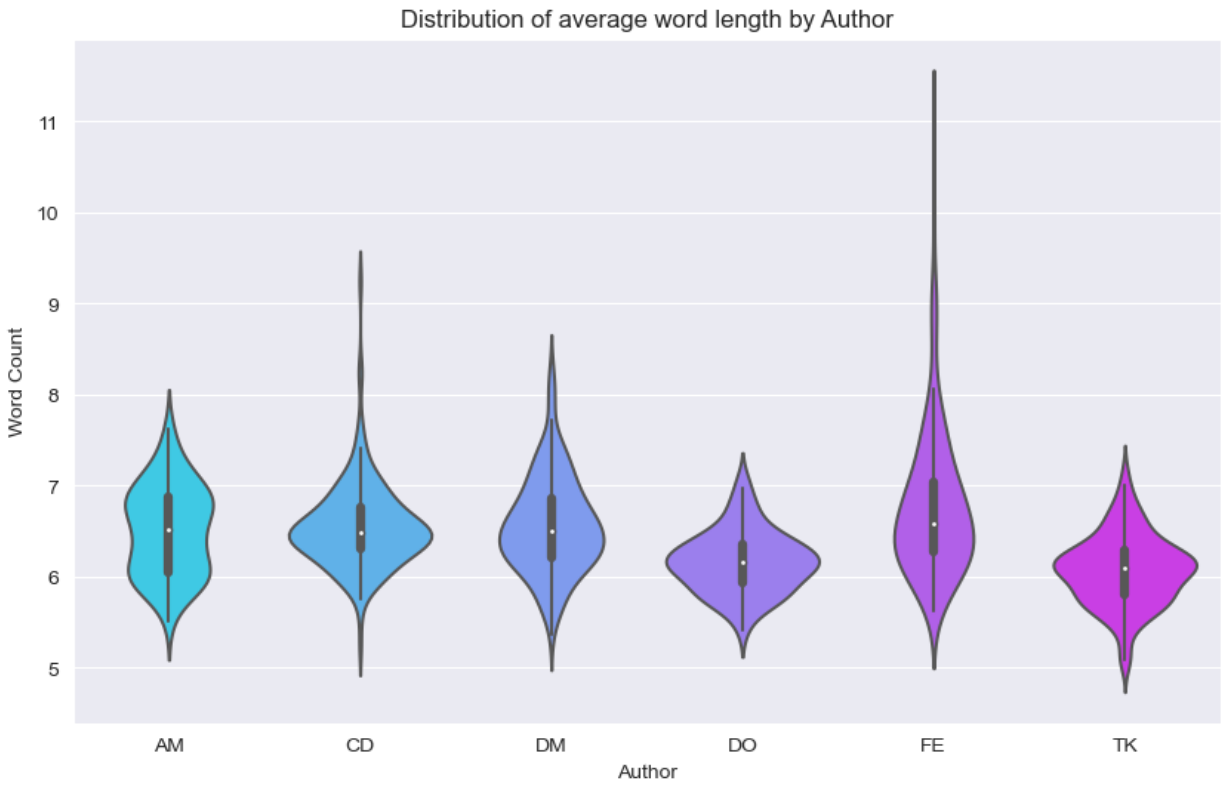
--> Both word count and text length are highly correlated features.

In [ ]:
```
# getting the average word length

df_train["avg_word_length"] = df_train["Text"].apply(lambda sent: sum([len(wrd

df_train.head()
```

Out[ ]:

	Text	Author	text_length	word_count	avg_word_length
0	Scoring in PROC DISCRIM is as easy as validati...	AM	215	37	5.810811
1	In the GLM procedure, you may have used LSMEAN...	AM	782	129	6.062016
2	The first problem, accuracy of the data file, ...	AM	990	159	6.226415
3	If the homogeneity of covariance matrices assu...	AM	934	146	6.397260
4	With a CONTRAST statement, you specify L, in t...	AM	1490	247	6.032389

In [ ]:
```
plt.figure(figsize=(10, 6))
ax = sns.violinplot(data=df_train, x="Author", y="avg_word_length", element="p
ax.set_title('Distribution of average word length by Author');
ax.set_ylabel('Word Count');
```



--> Average word length is an indicator of lexical complexity of the text.
--> This can be a good distinguishing feature since each author has different average word lengths.
--> Presence of outliers in author texts like FE indicates the usage of unique vocabulary.

In [ ]:

```
# adding a type-token ratio feature to check the lexical richness of the text,

# the aim is to look at the lexical richness of each author's texts by groupin

def text_tokenizer_1(sent):
    lemmatizer = WordNetLemmatizer()
    sent = sent.lower() # text to lowercase
    tokens = re.split(r'\W+', sent) # split text on non word characters
    clean_tokens = [lemmatizer.lemmatize(i) for i in tokens if i not in string
    clean_tokens_no_stopwords = [i for i in clean_tokens if i not in stopwords
    return clean_tokens_no_stopwords


def calculate_ttr(text):
    tokens = text_tokenizer_1(text.lower())
    types = set(tokens)
    return len(types) / len(tokens) if tokens else 0

df_train['type_to_token_ratio'] = df_train['Text'].apply(calculate_ttr)

df_train.head()
```

Out[ ]:

	Text	Author	text_length	word_count	avg_word_length	type_to_token_ratio
0	Scoring in PROC DISCRIM is as easy as validati...	AM	215	37	5.810811	0.840000
1	In the GLM procedure, you may have used LSMEAN...	AM	782	129	6.062016	0.661972
2	The first problem, accuracy of the data file, ...	AM	990	159	6.226415	0.712766
3	If the homogeneity of covariance matrices assu...	AM	934	146	6.397260	0.604938
4	With a CONTRAST statement, you specify L, in t...	AM	1490	247	6.032389	0.488372

In [ ]:

```
# looking at the row with the smallest type to token ratio.
small_ttt = df_train["type_to_token_ratio"].argmin()
print(f"Text with the smallest lexical richness by {df_train.iloc[small_ttt].A
print(df_train.iloc[small_ttt].Text)

print("\n\n\n")

large_ttt = df_train["type_to_token_ratio"].argmax()
print(f"Text with the largest lexical richness by {df_train.iloc[large_ttt].Au
print(df_train.iloc[large_ttt].Text)
```

```
Text with the smallest lexical richness by CD:
The LOGISTIC procedure enables you to specify whether model hierarchy is to be
preserved, how model hierarchy is applied, and whether a single effect or mult
iple effects can be moved in a single step. Model hierarchy refers to the requ
irement that for any effect in the model, all effects it contains must also be
in the model. For example, in order for the interaction A*B to enter the mode
l, the main effects A and B must be in the model. Model hierarchy is desirable
because models that are hierarchically well formulated have inferences that ar
e invariant to the coding you choose for your predictor variables (Kleinbaum,
Kupper, and Muller 1988). If the model is not hierarchically well formulated,
then the tests for the lower order terms will depend on the coding (reference
versus effect coding for categorical variables). The HIERARCHY= option specifi
es whether hierarchy is maintained and whether a single effect or multiple eff
ects are allowed to enter or leave the model in one step for SELECTION=FORWAR
D, SELECTION=BACKWARD, and SELECTION=STEPWISE. For HIERARCHY=SINGLE, only one
effect can enter or leave the model at one time, subject to model hierarchy. F
or example, suppose that you specify the main effects A and B and the interact
ion of A*B in the model. For backward elimination, the interaction A*B must fi
rst be removed before the main effects are removed. For forward selection, the
main effects must enter the model before the interaction. For HIERARCHY=MULTIP
LE, more than one effect can enter or leave the model at one time, subject to
model hierarchy. For forward selection, a single main effect can enter the mod
el or an interaction can enter the model together with all the effects that ar
e contained in the interaction. Backward selection can remove an interaction i
tself or the interaction together with all the effects that the interaction co
ntains. If you do not want to have model hierarchy, specify HIERARCHY=NONE. In
that case, any single effect can enter or leave the model at any given step of
the selection process. The default is HIERARCHY=SINGLE.
```

```
Text with the largest lexical richness by CD:
Because the response variable has more than 2 levels, by default PROC LOGISTIC
models cumulative logits. With the DESCENDING option, the probabilities modele
d are cumulated by the jth category and higher. Score Test for the Proportiona
l Odds Assumption
```

--> The type to token ratio measures lexical diversity. A higher ratio indicates a richer vocabulary with less repetition, which shows a more creative writing styles.
--> First text seems to be of more detailed and explanatory nature while the second is of a summary or conclusive type.
--> Both texts are written by the same author which shows the author's adaptability in terms of writing style (writing more varied concise words and in depth, writing of repeated technical terms).

In [ ]:
```
df_train["punctuations_count"] = df_train['Text'].apply(lambda x: len([i for i

df_train.head()

# initially the number of commas and exclamation marks were 2 different column
```

Out[ ]:

	Text	Author	text_length	word_count	avg_word_length	type_to_token_ratio	punctu
0	Scoring in PROC DISCRIM is as easy as validati...	AM	215	37	5.810811	0.840000	
1	In the GLM procedure, you may have used LSMEAN...	AM	782	129	6.062016	0.661972	
2	The first problem, accuracy of the data file, ...	AM	990	159	6.226415	0.712766	
3	If the homogeneity of covariance matrices assu...	AM	934	146	6.397260	0.604938	
4	With a CONTRAST statement, you specify L, in t...	AM	1490	247	6.032389	0.488372	

In [ ]:
```
def avg_sentence_length(txt):
    sents = re.split(r'[.!?]+', txt)
    sents = [i.strip() for i in sents if i.strip()]
    word_counts = [len(i.split()) for i in sents]

    if len(word_counts) > 0:
        return sum(word_counts) / len(word_counts)
    else:
        return 0

df_train['avg_sentence_length'] = df_train['Text'].apply(avg_sentence_length)
```

In [ ]:
```
df_train.head()
```

Out[ ]:

	Text	Author	text_length	word_count	avg_word_length	type_to_token_ratio	punctu
0	Scoring in PROC DISCRIM is as easy as validati...	AM	215	37	5.810811	0.840000	
1	In the GLM procedure, you may have used LSMEAN...	AM	782	129	6.062016	0.661972	
2	The first problem, accuracy of the data file, ...	AM	990	159	6.226415	0.712766	
3	If the homogeneity of covariance matrices assu...	AM	934	146	6.397260	0.604938	
4	With a CONTRAST statement, you specify L, in t...	AM	1490	247	6.032389	0.488372	

In [ ]:

```
def pos_proportions(text, pos_tag):
    doc = nlp(text)
    pos_counts = 0
    total_words = 0
    for token in doc:
        if token.is_alpha:
            total_words += 1
        if token.pos_ == pos_tag:
            pos_counts += 1
    return pos_counts / total_words if total_words > 0 else 0


# counting proportion of words which are nouns
df_train['POS_Nouns'] = df_train['Text'].apply(lambda x: pos_proportions(x, "N
# counting proporiton of words which are everbs
df_train['POS_Verbs'] = df_train['Text'].apply(lambda x: pos_proportions(x, "V


df_train.head()
```

Out[ ]:

	Text	Author	text_length	word_count	avg_word_length	type_to_token_ratio	punctu
0	Scoring in PROC DISCRIM is as easy as validati...	AM	215	37	5.810811	0.840000	
1	In the GLM procedure, you may have used LSMEAN...	AM	782	129	6.062016	0.661972	
2	The first problem, accuracy of the data file, ...	AM	990	159	6.226415	0.712766	
3	If the homogeneity of covariance matrices assu...	AM	934	146	6.397260	0.604938	
4	With a CONTRAST statement, you specify L, in t...	AM	1490	247	6.032389	0.488372	

In [ ]:

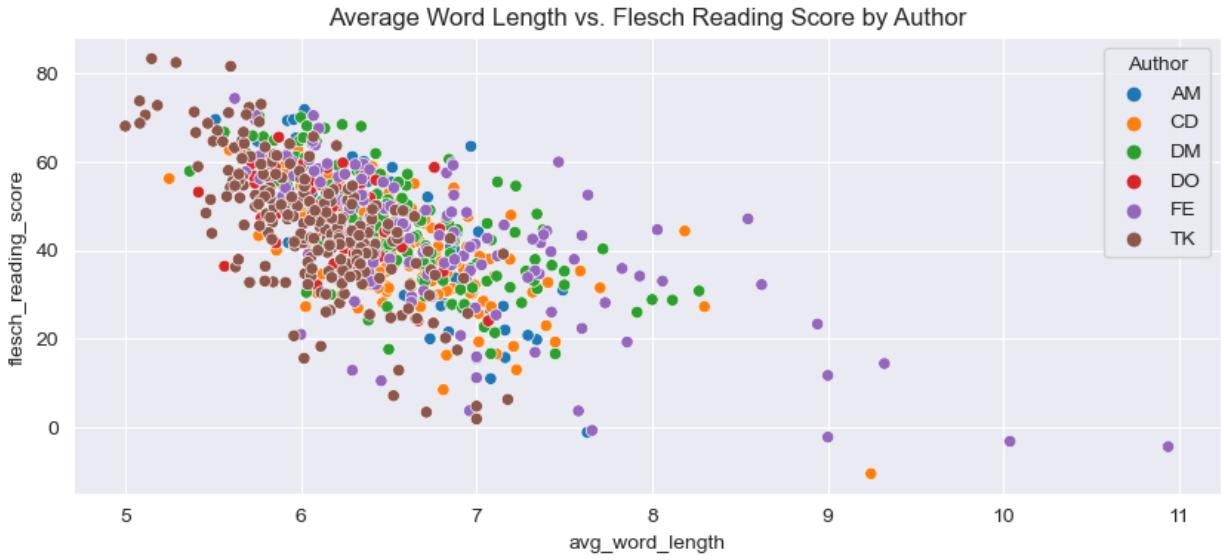
```
df_train['flesch_reading_score'] = df_train['Text'].apply(textstat.flesch_read

df_train.head()
```

Out[ ]:

	Text	Author	text_length	word_count	avg_word_length	type_to_token_ratio	punctu
0	Scoring in PROC DISCRIM is as easy as validati...	AM	215	37	5.810811	0.840000	
1	In the GLM procedure, you may have used LSMEAN...	AM	782	129	6.062016	0.661972	
2	The first problem, accuracy of the data file, ...	AM	990	159	6.226415	0.712766	
3	If the homogeneity of covariance matrices assu...	AM	934	146	6.397260	0.604938	
4	With a CONTRAST statement, you specify L, in t...	AM	1490	247	6.032389	0.488372	

In [ ]:

```
plt.figure(figsize=(10, 4))
sns.scatterplot(x='avg_word_length', y='flesch_reading_score', hue='Author', da
plt.title('Average Word Length vs. Flesch Reading Score by Author')
plt.show()
```



--> The Flesch Reading Score is a readability index which indicates how easy a text is to understand.
--> It's useful for understanding the reader's reading level.
--> more readable texts tend to use shorter and more simpler words

In [ ]:
```
df_train['gunning_fog_index'] = df_train['Text'].apply(textstat.gunning_fog)
df_train.head()
```

Out[ ]:

	Text	Author	text_length	word_count	avg_word_length	type_to_token_ratio	punctu
0	Scoring in PROC DISCRIM is as easy as validati...	AM	215	37	5.810811	0.840000	
1	In the GLM procedure, you may have used LSMEAN...	AM	782	129	6.062016	0.661972	
2	The first problem, accuracy of the data file, ...	AM	990	159	6.226415	0.712766	
3	If the homogeneity of covariance matrices assu...	AM	934	146	6.397260	0.604938	
4	With a CONTRAST statement, you specify L, in t...	AM	1490	247	6.032389	0.488372	

--> The Gunning Fog Index measures the complexity of the text, especially in terms of syllable count and sentence length.
--> It's useful for understanding the academic level of the content.

In [ ]:
```
df_train['unique_word_count'] = df_train['Text'].apply(lambda text: len(set(te
df_train.head()
```

Out[ ]:

	Text	Author	text_length	word_count	avg_word_length	type_to_token_ratio	punctu
0	Scoring in PROC DISCRIM is as easy as validati...	AM	215	37	5.810811	0.840000	
1	In the GLM procedure, you may have used LSMEAN...	AM	782	129	6.062016	0.661972	
2	The first problem, accuracy of the data file, ...	AM	990	159	6.226415	0.712766	
3	If the homogeneity of covariance matrices assu...	AM	934	146	6.397260	0.604938	
4	With a CONTRAST statement, you specify L, in t...	AM	1490	247	6.032389	0.488372	

--> unique word count reflect the range of vocabulary used and can be a sign of more original and in depth writing

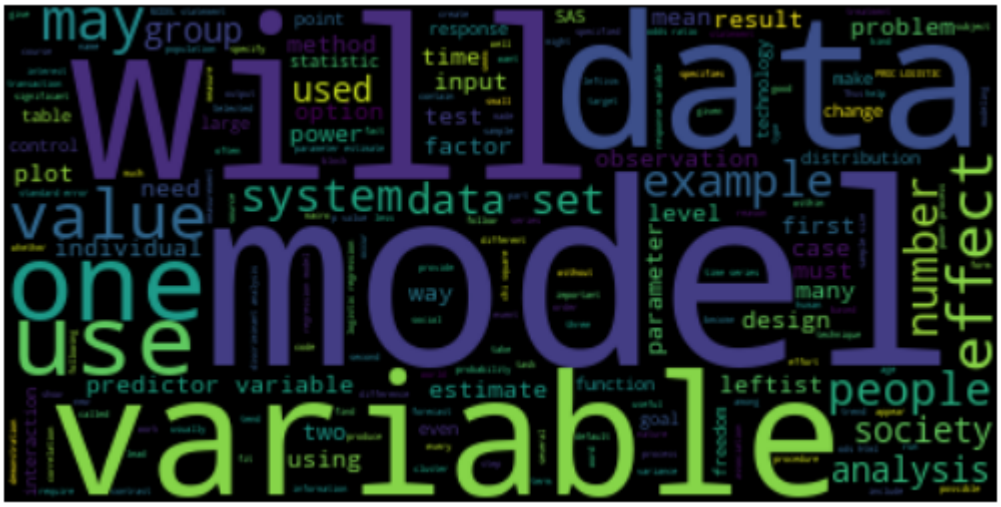
In [ ]:

```
def number_of_stopwords(text):
    return len([w for w in str(text).lower().split() if w in stopwords.words("

df_train["stopwords_number"] = df_train["Text"].apply(number_of_stopwords)
df_train.head()
```

Out[ ]:

	Text	Author	text_length	word_count	avg_word_length	type_to_token_ratio	punctu
0	Scoring in PROC DISCRIM is as easy as validati...	AM	215	37	5.810811	0.840000	
1	In the GLM procedure, you may have used LSMEAN...	AM	782	129	6.062016	0.661972	
2	The first problem, accuracy of the data file, ...	AM	990	159	6.226415	0.712766	
3	If the homogeneity of covariance matrices assu...	AM	934	146	6.397260	0.604938	
4	With a CONTRAST statement, you specify L, in t...	AM	1490	247	6.032389	0.488372	

In [ ]:
```
text = " ".join(df_train.Text)

wordcloud = WordCloud().generate(text)

plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.show()
```



At first glance it can be seen that:

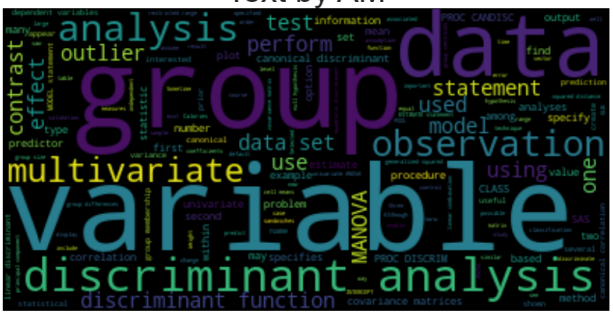
-> Most important words: "Model", "Data", "Analysis", "Variable"

-> Stopwords: "One", "May", "Even", "Must", "Will"

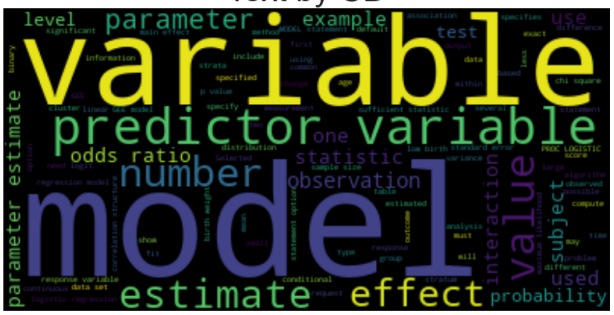
In [ ]:

```
# look at words each author has written

author_texts = {author: " ".join(df_train[df_train['Author'] == author]['Text'

num_authors = len(author_texts)
num_rows = num_authors // 2 + num_authors % 2

plt.figure(figsize=(16, 5 * num_rows)) # to show the wordclouds as subplots fo


for i, (author, text) in enumerate(author_texts.items(), start=1):
    wordcloud = WordCloud().generate(text)
    plt.subplot(num_rows, 2, i)
    plt.imshow(wordcloud, interpolation='bilinear')
    plt.title(f"Text by {author}", fontsize=30)
    plt.axis("off")

plt.tight_layout()
plt.show()
```

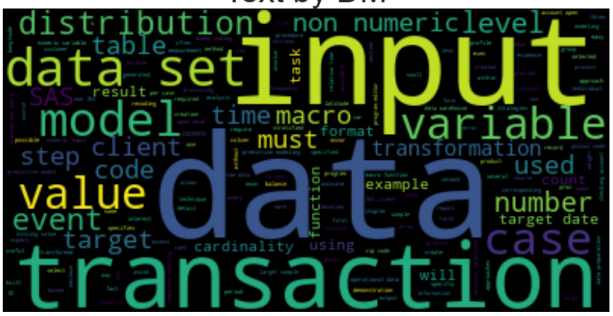
Text by AM



Text by CD

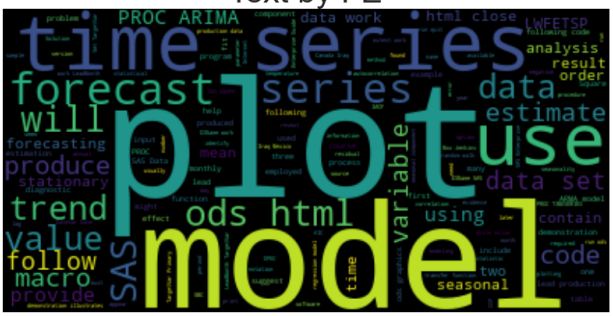


Text by DM



Text by DO

Text by FE



Text by TK

At first glance it can be seen that each author is speaking about different topics:
-> AM: talks about statistical classification techniques, discussing things such as discriminant analysis, group differences, and variable selection
-> CD: talks about machine learning and probability concepts like logistic regression, odds ratios, model estimation, predictor variable
-> DM: talks about data modeling, specifically addressing the management and analysis of transactional data sets
-> DO: talks about experimental design and topics like block design and factorial experiements
-> FE: talks about time series forecasting with topics like PROC, ARIMA and forecast
-> TK: talks about societal impact, with topics like society, psychological, and people.

4. Data Cleaning

In [ ]:

```
file_path_pattern = r"[a-zA-Z]:\\[\\\S|*\S]?.*"
url_pattern = r"http[s]?://(?:[a-zA-Z]|[0-9]|[$-_@.&+]|[!*\\(\\),]|(?:%[0-9a-f
html_tag_pattern = r"<[^>]+>"

# Iterate through the DataFrame
for text in df_train["Text"]:
    if re.search(file_path_pattern, text) or re.search(url_pattern, text) or r
        print(text)
        print("\n\n\n")

# some of the text also has file paths which are not required for the text
# these must be removed from the text as they are considered noise
# File names (xxxx.xx) must also be removed.
# urls must also be removed
# there's no html tags present within the text
```

The Tofu Veg Delite does not appear to fit with the general pattern of the res t of the data. You also notice that the Frozen sandwiches are close to one ano ther near the low end of Calories and Sodium. Principal Components and Outlier s: SAS Code within the SAS/IML Workshop For more than three-dimensional space, the easiest way to spot outliers can be with a histogram of principal componen t scores. The program App_OutliersIML.iml is displayed below: libname amul ' C:\workshop\winsas\amul\data\';

Open the file Managing Data Pathologies 1.sas in the program editor. This gene ral-purpose program creates a convenient statistical report on any SAS data se t and places the report in a PDF file for review and printing. The report incl udes a CONTENTS procedure summary of the data set columns, statistical summari es and a distribution plot for every numeric variable, a cardinality report fo r every character variable, and a frequency table for every character variable with cardinality below a specified threshold. The program begins by loading a utility macro file specialized to the needs of this chapter. *** Include patho logy macros; filename crsmacs catalog "dmdp.course_macros"; %include crsmacs(" pathology_macros.source"); A discussion of the macros used in the report follo ws in an optional section. Next, the name of the data set to analyze and the l ocation of the analysis report are specified. *** Define data set of interest and report destination; %let data=dmdp.ins_modeling_sample; %let report_name=' C:\temp\output.pdf'; The data set DMDP.INS_MODELING_SAMPLE is a merging of all analysis variables created to this point. It also contains input from other da ta sources related to the INS prediction problem, but not specifically analyze d in the previous chapters. In particular, limited transaction-based inputs fr om accounts other than checking are attached. In all, the data set contains 89 columns and 2,328 rows. Also, note that none of the census demographic fields from the end of chapter 4 have been appended.

The United States Department of Transportation through its Bureau of Transport ation Statistics publishes airline transportation statistics on the following Web site, http://www.transtats.bts.gov Monthly data for January 1990, through June 2008, have been extracted for analysis. The data set contains monthly pas senger totals for scheduled flights of domestic U.S. carriers. The data helps illustrate concepts of statistical time series analysis. A visual examination of time series data helps suggest strategies for forecasting future values of the series. Program Ch01_01.sas provides SAS code for producing most of the pl ots that are included in this chapter.

This demonstration illustrates how to produce a time plot. For this and subseq uent demonstrations, the SAS windowing environment will be used. SAS/GRAPH sof tware is easy to use, but default settings produce rather bland results. ODS G raphics have expanded the capabilities of SAS/GRAPH software to produce public ation quality plots with minimal effort. The Ministry of Manpower, Singapore G overnment, publishes employment statistics. Data can be obtained (with English text) from: http://www.mom.gov.sg/publish/momportal/en/home.html Program Demo1 _01Plot.sas contains SAS code to plot the Singapore unemployment time series. Use PROC CONTENTS to obtain information about the SAS data set LWFETSP.SGunemp loyment. title1 "Unemployment in Singapore"; proc contents data=LWFETSP.SGunem ployment; run; ASCII text from the output window follows. Unemployment in Sing apore 2

This demonstration illustrates the modeling of trend and seasonality using the NOAA temperature data for the state of Texas. The code for this demonstration can be found in Dem3_03Temp.sas. The data was obtained from the National Climatic Data Center Web site of the NOAA. http://www.ncdc.noaa.gov/oa/ncdc.html The SAS data set LWFETSP.USA_TX_NOAA contains several climate variables for the state of Texas in the United States. This demonstration focuses on temperature values in Fahrenheit from January 1895, to February 2009. Temperature data sets gathered over long periods can help determine the effects of global warming. The following program creates a working copy of the temperature data set and provides the primary diagnostic plots for model determination. %WorkCopy(LWFETSP.USA_TX_NOAA); ods html; %AllTSPlots(DSName=work.USA_TX_NOAA, TargetVar=Temperature, DateVar=Date); ods html close; The diagnostic plots follow.

SAS Enterprise Guide has several tasks for forecasting. If SAS Enterprise Guide is used for this course, tasks will primarily be performed within the SAS Code task. If you use SAS Enterprise Guide, you will need to assign the library LWFETSP, and for programs that use course macros, you will need to include the following statement at the beginning of your program. %include "S:\workshop\sassrc\LWFETSP.sas";

In [ ]:
```
# for the file paths the regex was created to start with any letter A - Z (capi
# or for the file names any non white space characters followed by a dot and o

# Lambda function to remove file paths (C:\\karthik\nlp_assignment) and files
remove_file_paths = lambda text: re.sub(r'\b[A-Za-z]:\\[^ ]*|\S+\.\S+', '', tex

# Lambda function to remove URLs (http://www.ncdc.noaa.gov/oa/ncdc.html)
remove_urls = lambda text: re.sub(r'http[s]?://(?:[a-zA-Z]|[0-9]|[$-_@.&+]|[!*

# Lambda function to remove punctuation
remove_punctuation = lambda text: re.sub(r'[%s]' % re.escape(string.punctuatio

remove_file_paths, remove_punctuation, remove_urls
```

Out[ ]:
```
(<function __main__.<lambda>(text)>,
 <function __main__.<lambda>(text)>,
 <function __main__.<lambda>(text)>)
```

In [ ]:
```
# preprocessing text and making them into tokens

def preprocessing_text(sent):
    lemmatizer = WordNetLemmatizer()
    sent_expand = contractions.fix(sent) # expand the contractions (I'd => I w
    sent_expand = sent_expand.lower() # text to lowercase
    clean_text = remove_file_paths(sent_expand)
    clean_text = remove_urls(clean_text)
    clean_text = remove_punctuation(clean_text)
    tokens = re.split(r'\W+', clean_text) # split based on non word characters
    no_stopwords = [i for i in tokens if i not in stopwords.words('english')]
    clean_tokens = [
        lemmatizer.lemmatize(i).strip() if i.lower() != "sas" else "sas"
        for i in no_stopwords
        if i not in string.punctuation
        and not any(char.isdigit() for char in i) # check for digits
        and i != ""
    ] # remove number, punctuation, remove words with numbers and lemmatize te
    return ' '.join(clean_tokens)



print(df_train["Text"].iloc[52])
print("=========================================================================
print(preprocessing_text(df_train["Text"].iloc[52])) # looking at the results
```

```
The Tofu Veg Delite does not appear to fit with the general pattern of the res
t of the data. You also notice that the Frozen sandwiches are close to one ano
ther near the low end of Calories and Sodium. Principal Components and Outlier
s: SAS Code within the SAS/IML Workshop For more than three-dimensional space,
the easiest way to spot outliers can be with a histogram of principal componen
t scores. The program App_OutliersIML.iml is displayed below: libname amul '
C:\workshop\winsas\amul\data\';
=========================================================================
========================
tofu veg delite appear fit general pattern rest data also notice frozen sandwi
ch close one another near low end calorie sodium principal component outlier s
as code within sas iml workshop three dimensional space easiest way spot outli
er histogram principal component score program displayed libname amul
```

In [ ]:
```
df_train["cleaned_text"] = df_train["Text"].apply(preprocessing_text)
df_train.head()
```

Out[ ]:

	Text	Author	text_length	word_count	avg_word_length	type_to_token_ratio	punctu
0	Scoring in PROC DISCRIM is as easy as validati...	AM	215	37	5.810811	0.840000	
1	In the GLM procedure, you may have used LSMEAN...	AM	782	129	6.062016	0.661972	
2	The first problem, accuracy of the data file, ...	AM	990	159	6.226415	0.712766	
3	If the homogeneity of covariance matrices assu...	AM	934	146	6.397260	0.604938	
4	With a CONTRAST statement, you specify L, in t...	AM	1490	247	6.032389	0.488372	

In [ ]:

```
# these words with hyphens in the middle will be the compound words

pattern = re.compile(r'\b[A-Za-z]+(?:-[A-Za-z]+)+\b')

def find_words_hyphen(text):
    """
    finding all the words with hyphen and to make them as compound words when
    """
    return pattern.findall(text)

hyphenated_words = df_train['Text'].apply(find_words_hyphen).sum()
hyphenated_words_flat = [''.join(word) for word in hyphenated_words]
hyphenated_word_counts = Counter(hyphenated_words_flat)
hyphenated_word_list = list(hyphenated_word_counts)

pprint(hyphenated_word_counts)
```

```
Counter({'chi-square': 33,
         'non-numeric': 28,
         'p-values': 22,
         'p-value': 16,
         'industrial-technological': 15,
         'small-scale': 13,
         'goodness-of-fit': 12,
         'Box-Jenkins': 12,
         'Breslow-Day': 11,
         'stratum-specific': 11,
         'non-INS': 11,
         'long-term': 10,
         'self-esteem': 10,
         'model-based': 9,
         'R-Square': 9,
         'subject-matter': 8,
         'case-control': 8,
         'SAS-Data-Set': 7,
         'power-hungry': 7,
         'so-called': 6,
         'higher-order': 6,
         'within-group': 5,
         'log-likelihood': 5,
         'Dickey-Fuller': 5,
         'organization-dependent': 5,
         'quasi-complete': 4,
         'two-factor': 4,
         'D-optimal': 4,
         'pre-whitening': 4,
         'Ljung-Box': 4,
         'black-style': 4,
         'long-distance': 4,
         'one-way': 3,
         'three-way': 3,
         'R-squared': 3,
         'Goodness-of-Fit': 3,
         'chi-squares': 3,
         'upper-left': 3,
         'upper-right': 3,
         'population-averaged': 3,
         'time-independent': 3,
         'time-dependent': 3,
         'Mantel-Haenszel': 3,
         'logit-based': 3,
         'Newton-Raphson': 3,
         'follow-up': 3,
         'Self-Study': 3,
         'A-optimality': 3,
         'SAS-data-set': 3,
         'Yule-Walker': 3,
         'cross-correlation': 3,
         'short-term': 3,
         'pre-industrial': 3,
         'middle-class': 3,
         'small-business': 3,
         'non-leftist': 3,
         'three-dimensional': 2,
         'data-set': 2,
         'right-click': 2,
         'one-unit': 2,
```

```
'one-step': 2,
'model-building': 2,
'dummy-coded': 2,
'chi-squared': 2,
'likelihood-ratio': 2,
'quasi-likelihood': 2,
'between-cluster': 2,
'Case-Control': 2,
'variance-covariance': 2,
'Cochran-Mantel-Haenszel': 2,
'two-sided': 2,
'Quasi-complete': 2,
'non-existent': 2,
'threshold-based': 2,
'Non-Numeric': 2,
'Pass-Through': 2,
'three-step': 2,
'non-stationarities': 2,
'I-optimality': 2,
'S-optimal': 2,
'D-optimality': 2,
're-coding': 2,
'A-optimal': 2,
'G-optimal': 2,
'I-optimal': 2,
'B-splines': 2,
'pre-whitened': 2,
'cross-correlations': 2,
'self-hatred': 2,
'clear-cut': 2,
'small-group': 2,
'power-holding': 2,
'over-socialized': 2,
'non-leftists': 2,
'hard-pressed': 2,
'large-scale': 2,
'child-rearing': 2,
'anti-government': 2,
'upper-middle': 2,
'high-tech': 2,
'crypto-leftist': 2,
'power-elite': 2,
'self-confidence': 2,
'stress-producing': 2,
'instructor-based': 1,
'type-I': 1,
'type-II': 1,
'one-observation': 1,
'near-zero': 1,
'Three-way': 1,
'two-and': 1,
'nearest-neighbor': 1,
'within-class': 1,
'between-class': 1,
'Chi-Square': 1,
'One-way': 1,
're-specify': 1,
'y-intercept': 1,
'lower-order': 1,
'Hotelling-Lawley': 1,
```

```
'F-approximation': 1,
'fast-food': 1,
'class-level': 1,
'dimension-reduction': 1,
'well-informed': 1,
'DSM-IV': 1,
'z-scores': 1,
'S-shaped': 1,
'high-resolution': 1,
'dummy-coding': 1,
're-estimating': 1,
'user-defined': 1,
'Hosmer-Lemeshow': 1,
'off-diagonals': 1,
'Newton-Raphson-type': 1,
'cross-classifying': 1,
'parallel-lines': 1,
'lower-right': 1,
'lower-left': 1,
're-estimate': 1,
'subject-specific': 1,
'between-stratum': 1,
'stratum-constant': 1,
'normal-theory': 1,
'Subject-matter': 1,
'all-available': 1,
'non-smokers': 1,
'one-year': 1,
'built-in': 1,
'health-related': 1,
'cross-products': 1,
'modeling-building': 1,
'lower-bounds': 1,
'Quasi-likelihood': 1,
'log-odds': 1,
'non-stratification': 1,
'n-way': 1,
'm-dependent': 1,
'lack-of-fit': 1,
'quasi-Newton': 1,
'model-predicted': 1,
'hypothesis-testing': 1,
'independence-only': 1,
'log-log': 1,
't-tests': 1,
'reference-cell': 1,
'case-specific': 1,
'real-world': 1,
'key-punch': 1,
'five-step': 1,
'grouped-ranks': 1,
'general-purpose': 1,
'transaction-based': 1,
'non-disclosure': 1,
'rank-based': 1,
'Site-specific': 1,
'choice-based': 1,
'y-conditional': 1,
'outcome-dependent': 1,
'input-target': 1,
```

```
'half-year': 1,
'model-appropriate': 1,
'Non-numeric': 1,
'Free-form': 1,
'five-digit': 1,
'two-dimensional': 1,
'target-based': 1,
'gender-specific': 1,
'Time-Dependent': 1,
'non-zero': 1,
'well-designed': 1,
'self-reported': 1,
'Two-level': 1,
'two-level': 1,
'macro-based': 1,
'optimality-criterion': 1,
'factor-response': 1,
'high-order': 1,
'e-mail': 1,
'sub-population': 1,
'counter-intuitive': 1,
'month-to-month': 1,
'n-b': 1,
'non-trivial': 1,
'N-way': 1,
'non-blocked': 1,
'G-efficiency': 1,
'k-level': 1,
'mower-driver': 1,
'block-to-block': 1,
'distance-based': 1,
'custom-made': 1,
're-code': 1,
'G-optimality': 1,
'S-optimality': 1,
'variance-based': 1,
'power-based': 1,
'error-based': 1,
'Cross-correlation': 1,
'chicken-and-egg': 1,
're-using': 1,
'x-axis': 1,
'y-variable': 1,
'x-variable': 1,
'add-in': 1,
'b-spline': 1,
'high-density': 1,
'two-stage': 1,
'n-k': 1,
'Identify-Estimate-Forecast': 1,
'post-intervention': 1,
'lead-lag': 1,
'step-ahead': 1,
'closed-form': 1,
'B-spline': 1,
'seven-day': 1,
'client-server': 1,
'start-up': 1,
'on-demand': 1,
'time-consuming': 1,
```

```
'vendor-supplied': 1,
'non-moral': 1,
'counter-ideal': 1,
'ghetto-dweller': 1,
'franchise-granting': 1,
'high-prestige': 1,
'upper-middle-class': 1,
'mid-life': 1,
'unheard-of': 1,
'well-developed': 1,
'ever-increasing': 1,
'one-millionth': 1,
'self-deceiving': 1,
'upper-level': 1,
'self-promoter': 1,
'self-worth': 1,
'life-expectancy': 1,
'un-leftist': 1,
'non-scientific': 1,
'life-cycles': 1,
'de-industrialization': 1,
'nation-by-nation': 1,
'non-industrial': 1,
'dictator-controlled': 1,
'noise-making': 1,
'Self-hatred': 1,
'pro-choice': 1,
'pro-life': 1,
'left-wing': 1,
'rabble-rousing': 1,
'world-view': 1,
'youth-gang': 1,
'mid-to-late': 1,
'thought-out': 1,
'long-lasting': 1,
'catch-phrases': 1,
'lower-level': 1,
'self-defense': 1,
'nation-wide': 1,
'anti-leftist': 1,
're-engineering': 1,
'life-and-death': 1,
'Small-scale': 1,
'Organization-dependent': 1,
'H-bomb': 1,
'busy-work': 1,
'MAN-MADE': 1,
'self-determination': 1,
'co-religionist': 1,
'Would-be': 1,
'crypto-leftists': 1,
'well-socialized': 1,
'well-sublimated': 1,
'machine-made': 1,
'man-made': 1,
'well-organized': 1,
'self-interest': 1,
'far-reaching': 1,
'off-shoot': 1,
'English-speaking': 1,
```

```
 'soft-hearted': 1,
 'Han-min': 1,
 'Chun-mai': 1,
 'factory-made': 1,
 'post-industrial': 1,
 'high-handed': 1,
 'self-reliance': 1,
 'anti-individualistic': 1,
 'pro-collectivist': 1,
 'thumb-nail': 1,
 'single-minded': 1,
 'double-crossed': 1,
 'process-having': 1,
 'goal-that': 1,
 'vis-a-vis': 1,
 'black-and-white': 1,
 'side-effects': 1,
 'stress-reduction': 1,
 'self-confident': 1,
 'anti-depressant': 1,
 'support-systems': 1,
 'quasi-religious': 1,
 'middle-level': 1,
 'other-directed': 1,
 'clear-sighted': 1,
 'money-maker': 1,
 're-engineered': 1,
 'break-down': 1})
```

In [ ]:
```
df_train.head()
```

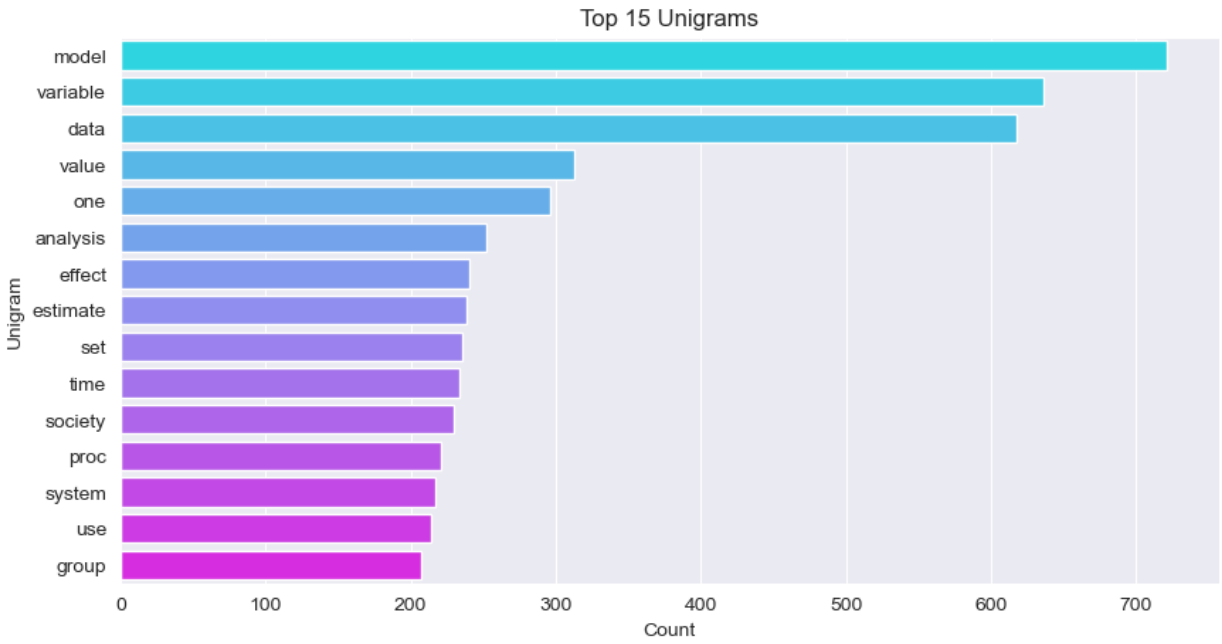
Out[ ]:

	Text	Author	text_length	word_count	avg_word_length	type_to_token_ratio	punctu
0	Scoring in PROC DISCRIM is as easy as validati...	AM	215	37	5.810811	0.840000	
1	In the GLM procedure, you may have used LSMEAN...	AM	782	129	6.062016	0.661972	
2	The first problem, accuracy of the data file, ...	AM	990	159	6.226415	0.712766	
3	If the homogeneity of covariance matrices assu...	AM	934	146	6.397260	0.604938	
4	With a CONTRAST statement, you specify L, in t...	AM	1490	247	6.032389	0.488372	

--> All the words with the '-' look like compound words.

In [ ]:

```
all_tokens = [word for text in df_train['cleaned_text'] for word in word_token
unigram_freq = FreqDist(all_tokens)

unigrams, counts = zip(*unigram_freq.most_common(15))
unigram_df = pd.DataFrame({'Unigram': unigrams, 'Count': counts})

plt.figure(figsize=(10, 5))
sns.barplot(x='Count', y='Unigram', data=unigram_df, order=unigram_df.sort_valu
plt.title('Top 15 Unigrams')
plt.show()
```

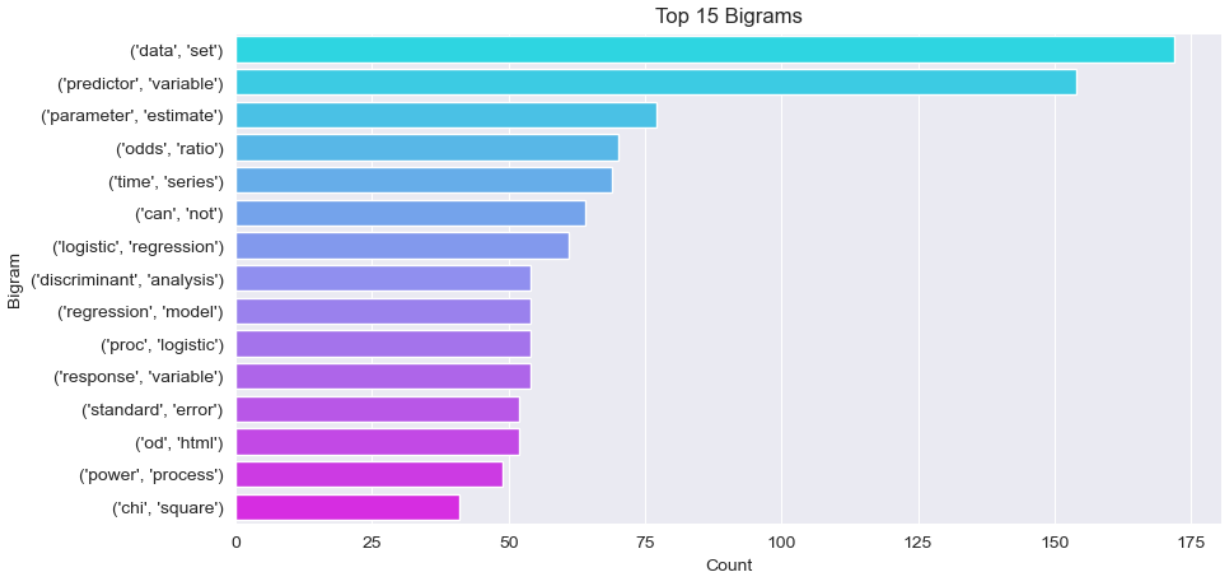


--> Words like model and variable cannot be considered as stopwords since there are some authors who dont use it as often as some of the authors as seen in the wordcloud.

In [ ]:

```
all_bigrams = list(bigrams(all_tokens))
bigram_freq = FreqDist(all_bigrams)
print(bigram_freq.most_common(10))

bigrams, counts = zip(*bigram_freq.most_common(15))
bigram_df = pd.DataFrame({'Bigram': bigrams, 'Count': counts})

plt.figure(figsize=(10, 5))
sns.barplot(x='Count', y='Bigram', data=bigram_df, order=bigram_df.sort_values
plt.title('Top 15 Bigrams')
plt.show()
```

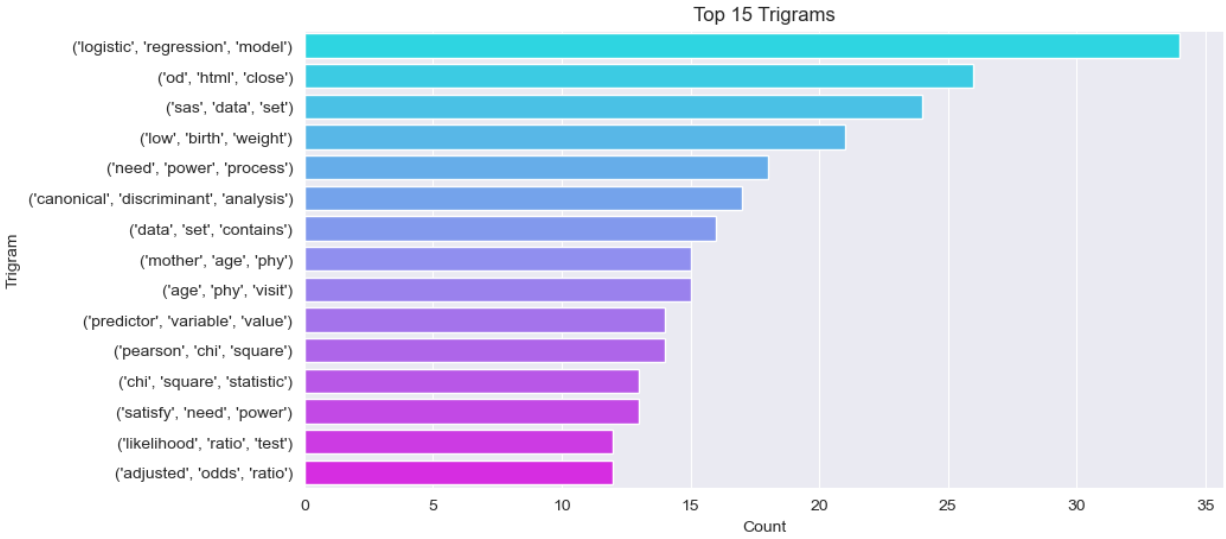
```
[(('data', 'set'), 172), (('predictor', 'variable'), 154), (('parameter', 'est
imate'), 77), (('odds', 'ratio'), 70), (('time', 'series'), 69), (('can', 'not
'), 64), (('logistic', 'regression'), 61), (('discriminant', 'analysis'), 54),
(('regression', 'model'), 54), (('proc', 'logistic'), 54)]
```



--> The top 15 bigrams are useful in determining the compound words, especially looking at the bottom few words like chi square and p value which are compound words.

In [ ]:
```
all_trigrams = list(trigrams(all_tokens))
trigram_freq = FreqDist(all_trigrams)
print(trigram_freq.most_common(10))

trigrams, counts = zip(*trigram_freq.most_common(15))
trigram_df = pd.DataFrame({'Trigram': trigrams, 'Count': counts})

plt.figure(figsize=(10, 5))
sns.barplot(x='Count', y='Trigram', data=trigram_df, order=trigram_df.sort_valu
plt.title('Top 15 Trigrams')
plt.show()
```

```
[(('logistic', 'regression', 'model'), 34), (('od', 'html', 'close'), 26), (('
sas', 'data', 'set'), 24), (('low', 'birth', 'weight'), 21), (('need', 'power
', 'process'), 18), (('canonical', 'discriminant', 'analysis'), 17), (('data',
'set', 'contains'), 16), (('mother', 'age', 'phy'), 15), (('age', 'phy', 'visi
t'), 15), (('predictor', 'variable', 'value'), 14)]
```



--> Trigrams do not look very useful for creating the compound words so it will not be considered.

In [ ]:
```
compound_words = [i[0] for i in bigram_freq.most_common(15)]

hyphenated_word_list = [i.split("-") for i in hyphenated_word_list]
compound_words += hyphenated_word_list
mwe_tokenizer = MWETokenizer(compound_words, separator="_")
mwe_tokenizer
```

Out[ ]:
```
<nltk.tokenize.mwe.MWETokenizer at 0x16dfd7370>
```

In [ ]:
```
# use this to update the tokens with compound words found from the bigram analy
# also remove special characters and numbers


def apply_mwe_to_tokens(row):
    tokens = mwe_tokenizer.tokenize(row.split())
    cleaned_tokens = [token.replace('\t', '') for token in tokens if not token
    return cleaned_tokens

df_train['cleaned_tokens'] = df_train['cleaned_text'].apply(apply_mwe_to_token
df_train.head()
```

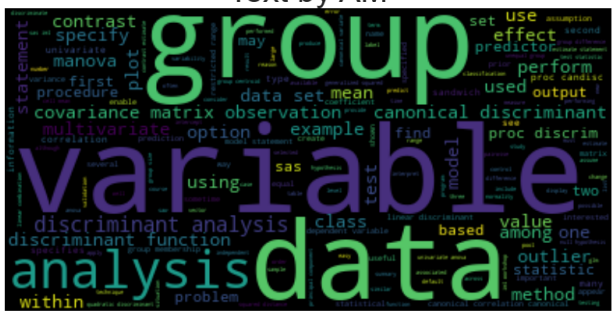
Out[ ]:

	Text	Author	text_length	word_count	avg_word_length	type_to_token_ratio	punctu
0	Scoring in PROC DISCRIM is as easy as validati...	AM	215	37	5.810811	0.840000	
1	In the GLM procedure, you may have used LSMEAN...	AM	782	129	6.062016	0.661972	
2	The first problem, accuracy of the data file, ...	AM	990	159	6.226415	0.712766	
3	If the homogeneity of covariance matrices assu...	AM	934	146	6.397260	0.604938	
4	With a CONTRAST statement, you specify L, in t...	AM	1490	247	6.032389	0.488372	

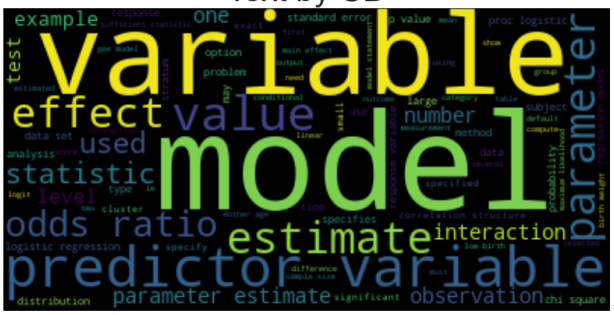
In [ ]:
```
# save the tokenizer so the list of compound words dont have to be defined aga
import pickle

with open('mwe_tokenizer.pkl', 'wb') as f:
    pickle.dump(mwe_tokenizer, f)
```

In [ ]:
```
# look at words each author has written

author_texts = {author: " ".join(df_train[df_train['Author'] == author]['clean

num_authors = len(author_texts)
num_rows = num_authors // 2 + num_authors % 2

plt.figure(figsize=(16, 5 * num_rows)) # to show the wordclouds as subplots fo


for i, (author, text) in enumerate(author_texts.items(), start=1):
    wordcloud = WordCloud().generate(text)
    plt.subplot(num_rows, 2, i)
    plt.imshow(wordcloud, interpolation='bilinear')
    plt.title(f"Text by {author}", fontsize=30)
    plt.axis("off")

plt.tight_layout()
plt.show()
```

Text by AM



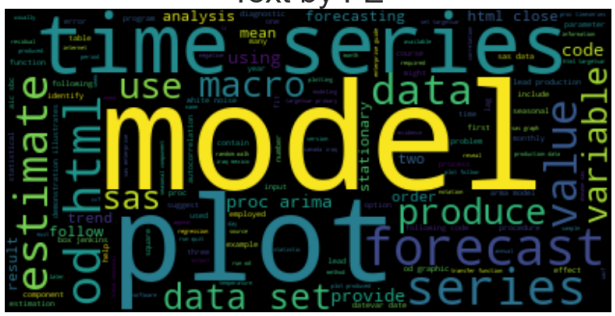
Text by CD



Text by DM

Text by DO

Text by FE



Text by TK

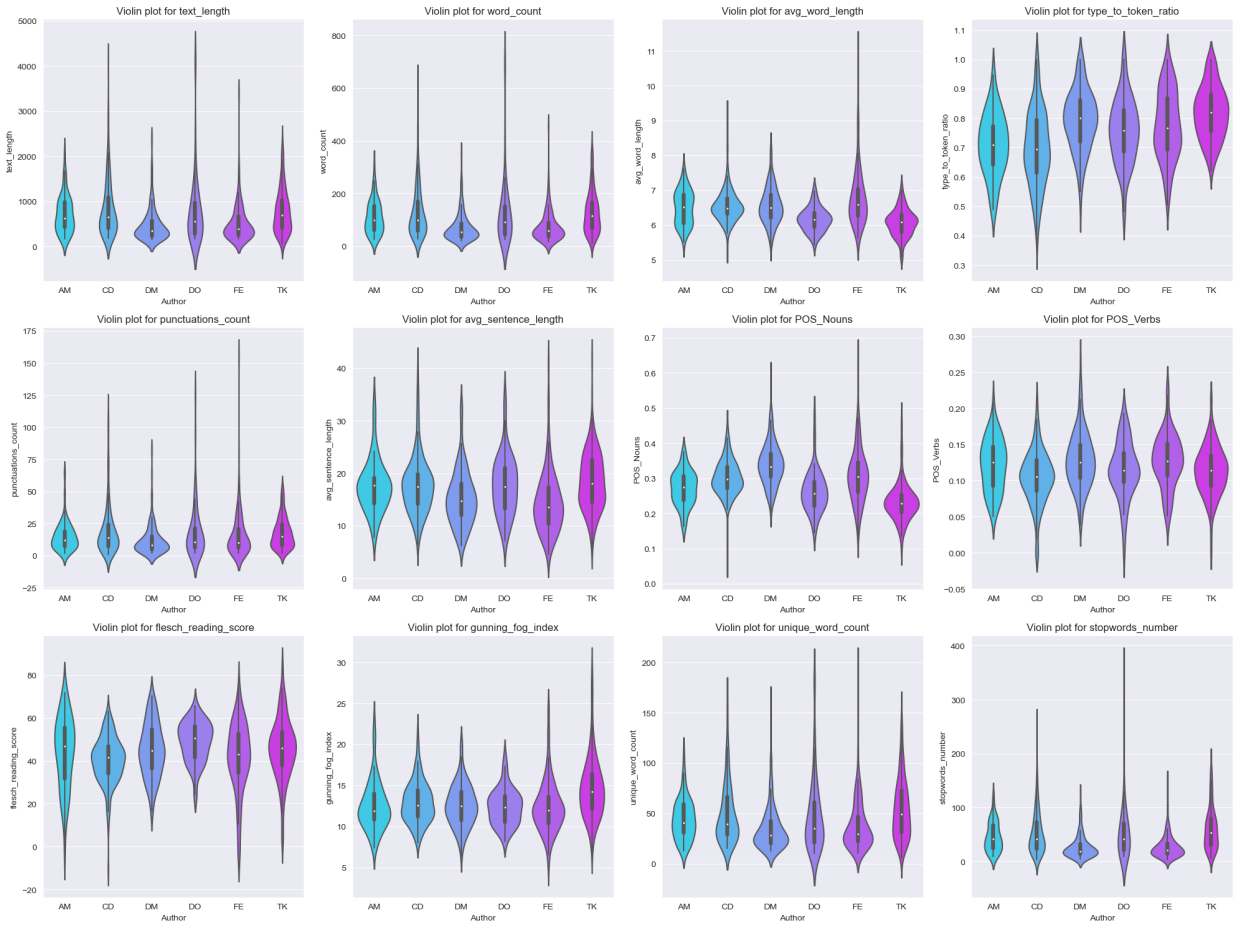
In [ ]:
```
df_train.describe().T # summary statistics of all the features created
# easier to understand through visualization
```

Out[ ]:

	count	mean	std	min	25%	50%	
text_length	816.0	688.286765	532.106926	61.000000	304.000000	526.500000	92
word_count	816.0	108.006127	84.159438	12.000000	47.000000	81.000000	14
avg_word_length	816.0	6.429311	0.597206	5.000000	6.056834	6.349312	
type_to_token_ratio	816.0	0.767340	0.117589	0.376963	0.690234	0.770330	
punctuations_count	816.0	16.705882	15.819899	1.000000	6.000000	12.000000	2
avg_sentence_length	816.0	16.911385	5.650943	4.500000	13.000000	16.333333	2
POS_Nouns	816.0	0.285110	0.071083	0.054054	0.234565	0.278879	
POS_Verbs	816.0	0.117164	0.037620	0.000000	0.093562	0.116586	
flesch_reading_score	816.0	43.933051	13.594932	-10.420000	35.940000	44.240000	5
gunning_fog_index	816.0	13.193885	3.155677	5.150000	11.167500	12.725000	1
unique_word_count	816.0	45.797794	28.505293	6.000000	25.000000	37.000000	6
stopwords_number	816.0	46.257353	38.436849	3.000000	19.000000	34.000000	6

In [ ]:

```
col_names = df_train.describe().columns
fig, axes = plt.subplots(3, 4, figsize=(20, 15))

axes = axes.flatten()

for i, col in enumerate(col_names):
    sns.violinplot(x="Author", y=col, data=df_train, ax=axes[i], palette=palet
    axes[i].set_title(f'Violin plot for {col}')

plt.tight_layout()
plt.show()
```



Each author has a unique way of writing that sets them apart.
--> Some authors, like CD and DO, tend to write longer texts, while others, like AM and TK, have shorter ones.
--> FE uses longer words than the other authors. This could be a clue that a text with bigger words was likely written by FE.
--> TK's writing uses a wide variety of words, making it rich and diverse. CD's writing is more varied compared to the rest as they have a higher lexical richness range
--> Authors use punctuation marks almost similarly however it can be seen that TK uses them a little bit more than the rest.
--> FE writes shorter sentences on average, while TK writes longer ones.
--> TK uses fewer nouns while DM uses more, this can be used to show what each author writes about.
--> All the authors seem to use action words (verbs) about the same amount
--> Flesch reading score shows the readability of the text and it seems that DO and AM having a wider readability distribution of around 60. This means that the text which is more readable is more likely to be of these authors.
--> Gunning fog talks about the complexity of the text, and in this case DO has the highest median complexity in their texts.
--> The analysis shows that DO and FE use a lot of common words which suggests that their writing might be describing things more clearly and comprehensively.

In [ ]:
```
df_train.to_csv("cleaned_data.csv")
```