2_0_Lexical_Complexity_Binary_Classification_Prediction_Data_Preparat

April 6, 2025

[61]:
```
#@title Install Packages
```

[62]:
```
!pip install -q transformers
!pip install -q torchinfo
!pip install -q datasets
!pip install -q evaluate
!pip install -q nltk
!pip install -q contractions
```

[63]:
```
!sudo apt-get update
! sudo apt-get install tree
```

```
Get:1 https://cloud.r-project.org/bin/linux/ubuntu jammy-cran40/ InRelease
[3,632 B]
Hit:2 https://developer.download.nvidia.com/compute/cuda/repos/ubuntu2204/x86_64
InRelease
Hit:3 http://archive.ubuntu.com/ubuntu jammy InRelease
Get:4 http://security.ubuntu.com/ubuntu jammy-security InRelease [129 kB]
Hit:5 https://r2u.stat.illinois.edu/ubuntu jammy InRelease
Get:6 http://archive.ubuntu.com/ubuntu jammy-updates InRelease [128 kB]
Hit:7 https://ppa.launchpadcontent.net/deadsnakes/ppa/ubuntu jammy InRelease
Hit:8 https://ppa.launchpadcontent.net/graphics-drivers/ppa/ubuntu jammy
InRelease
Hit:9 http://archive.ubuntu.com/ubuntu jammy-backports InRelease
Hit:10 https://ppa.launchpadcontent.net/ubuntugis/ppa/ubuntu jammy InRelease
Fetched 261 kB in 2s (105 kB/s)
Reading package lists… Done
W: Skipping acquire of configured file 'main/source/Sources' as repository
'https://r2u.stat.illinois.edu/ubuntu jammy InRelease' does not seem to provide
it (sources.list entry misspelt?)
Reading package lists… Done
Building dependency tree… Done
Reading state information… Done
tree is already the newest version (2.0.2-1).
0 upgraded, 0 newly installed, 0 to remove and 44 not upgraded.
```

[64]:
```
#@title Imports
import nltk
from nltk.tokenize import RegexpTokenizer

import evaluate
import transformers

import contractions

from torchinfo import summary
from datasets import load_dataset

from transformers import AutoTokenizer, AutoModel,␣
 ↪AutoModelForSequenceClassification
from transformers import TrainingArguments, Trainer

import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

import sklearn
```

[65]:
```
# @title Mount Google Drive
```

[66]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
```

[67]:
```
dir_root = '/content/drive/MyDrive/266-final/'
# dir_data = '/content/drive/MyDrive/266-final/data/'
# dir_data = '/content/drive/MyDrive/266-final/data/se21-t1-comp-lex-master/'
dir_data = '/content/drive/MyDrive/266-final/data/266-comp-lex-master'
dir_models = '/content/drive/MyDrive/266-final/models/'
dir_results = '/content/drive/MyDrive/266-final/results/'
```

[68]:
```
!tree /content/drive/MyDrive/266-final/data/266-comp-lex-master/
```

```
/content/drive/MyDrive/266-final/data/266-comp-lex-master/
  fe-test-labels
  fe-train
  fe-trial-val
  test-labels
      lcp_multi_test.tsv
      lcp_single_test.tsv
```

```
train
    lcp_multi_train.tsv
    lcp_single_train.tsv
trial
    lcp_multi_trial.tsv
    lcp_single_trial.tsv

6 directories, 6 files
```

[69]:
```
!ls -R /content/drive/MyDrive/266-final/data/266-comp-lex-master/
```

```
/content/drive/MyDrive/266-final/data/266-comp-lex-master/:
fe-test-labels  fe-train  fe-trial-val  test-labels  train  trial

/content/drive/MyDrive/266-final/data/266-comp-lex-master/fe-test-labels:

/content/drive/MyDrive/266-final/data/266-comp-lex-master/fe-train:

/content/drive/MyDrive/266-final/data/266-comp-lex-master/fe-trial-val:

/content/drive/MyDrive/266-final/data/266-comp-lex-master/test-labels:
lcp_multi_test.tsv  lcp_single_test.tsv

/content/drive/MyDrive/266-final/data/266-comp-lex-master/train:
lcp_multi_train.tsv  lcp_single_train.tsv

/content/drive/MyDrive/266-final/data/266-comp-lex-master/trial:
lcp_multi_trial.tsv  lcp_single_trial.tsv
```

[70]:
```
#@title Import Data
```

[71]:
```
# train_single_df = pd.read_csv(os.path.join(dir_data, "train",␣
 ↪"lcp_single_train.tsv"), sep="\t")
# train_multi_df = pd.read_csv(os.path.join(dir_data, "train", "lcp_multi_train.
 ↪tsv"), sep="\t")

# trail_val_single_df = pd.read_csv(os.path.join(dir_data, "trial",␣
 ↪"lcp_single_trial.tsv"), sep="\t")
# trail_val_multi_df = pd.read_csv(os.path.join(dir_data, "trial",␣
 ↪"lcp_multi_trial.tsv"), sep="\t")

# test_single_df = pd.read_csv(os.path.join(dir_data, "test-labels",␣
 ↪"lcp_single_test.tsv"), sep="\t")
# test_multi_df = pd.read_csv(os.path.join(dir_data, "test-labels",␣
 ↪"lcp_multi_test.tsv"), sep="\t")
```

[72]:
```
# # Try to load the files containing unterminated strings
# try:
#     # Approach 1: Try with the C engine but with error handling
#     multi_test_df = pd.read_csv(
#         os.path.join(dir_data, "test", "lcp_multi_test.tsv"),
#         sep="\t",
#         on_bad_lines='skip'  # Skip bad lines
#     )
#     print("Loaded with skipping bad lines")
# except Exception as e:
#     print(f"First approach failed: {e}")
#     try:
#         # Approach 2: Try with the Python engine which might be more forgiving
#         multi_test_df = pd.read_csv(
#             os.path.join(dir_data, "test", "lcp_multi_test.tsv"),
#             sep="\t",
#             engine="python",
#             quoting=3  # QUOTE_NONE
#         )
#         print("Loaded with Python engine")
```

[73]:
```
# Load train data into train_*_df
train_single_df = pd.read_csv(
    os.path.join(dir_data, "train", "lcp_single_train.tsv"),
    sep = "\t",
    engine = "python",
    quoting = 3
)
train_multi_df = pd.read_csv(
    os.path.join(dir_data, "train", "lcp_multi_train.tsv"),
    sep = "\t",
    engine = "python",
    quoting = 3
)

# Load trial data into trial_val_*_df
trial_val_single_df = pd.read_csv(
    os.path.join(dir_data, "trial", "lcp_single_trial.tsv"),
    sep = "\t",
    engine = "python",
    quoting = 3
)
trial_val_multi_df = pd.read_csv(
    os.path.join(dir_data, "trial", "lcp_multi_trial.tsv"),
    sep = "\t",
    engine = "python",
    quoting = 3
```

```
)

# Load test data (with labels) into test_*_df
test_single_df = pd.read_csv(
    os.path.join(dir_data, "test-labels", "lcp_single_test.tsv"),
    sep = "\t",
    engine = "python",
    quoting = 3
)
test_multi_df = pd.read_csv(
    os.path.join(dir_data, "test-labels", "lcp_multi_test.tsv"),
    sep = "\t",
    engine = "python",
    quoting = 3
)

print("Data successfully loaded into train, trial-val, and test variables")
```

```
Data successfully loaded into train, trial-val, and test variables
```

[74]:
```
#@title EDA
```

[75]:
```
def print_dataframe_summary(df_name, df):
    # Print section header
    print(f"========== {df_name} ==========")

    # Shape and Columns
    print(f"Shape: {df.shape}")
    print(f"Columns: {list(df.columns)}\n")

    # Data Types
    print("Data Types:")
    print(df.dtypes)
    print()

    # Missing Values
    print("Missing Values (by column):")
    print(df.isna().sum())
    print()

    # 'complexity' column stats
    desc = df['complexity'].describe()  # count, mean, std, min, 25%, 50%, 75%,␣
 ↪max
    print("'complexity' Column Stats (incl. quartiles and median):")
    print(desc)

    # Calculate frequency counts for each quartile range
```

```
    q1 = desc['25%']
    q2 = desc['50%']  # This is the median
    q3 = desc['75%']
    q_max = desc['max']

    # Note: We'll define the ranges as:
    #   <= Q1
    #   > Q1 and <= Q2
    #   > Q2 and <= Q3
    #   > Q3

    freq_q1 = np.sum(df['complexity'] <= q1)
    freq_q2 = np.sum((df['complexity'] > q1) & (df['complexity'] <= q2))
    freq_q3 = np.sum((df['complexity'] > q2) & (df['complexity'] <= q3))
    freq_q4 = np.sum(df['complexity'] > q3)

    print()
    print("Quartile Frequency Counts (tab-separated next to each quartile):")
    print(f"25%: {q1}\tCount (<= Q1): {freq_q1}")
    print(f"50% (Median): {q2}\tCount (Q1 < x <= Q2): {freq_q2}")
    print(f"75%: {q3}\tCount (Q2 < x <= Q3): {freq_q3}")
    print(f"100% (Max): {q_max}\tCount (Q3 < x <= Max): {freq_q4}")

    print("=====================================\n")

# Now we call this for each of our dataframes
print_dataframe_summary("train_single_df", train_single_df)
print_dataframe_summary("train_multi_df", train_multi_df)
print_dataframe_summary("trial_val_single_df", trial_val_single_df)
print_dataframe_summary("trial_val_multi_df", trial_val_multi_df)
print_dataframe_summary("test_single_df", test_single_df)
print_dataframe_summary("test_multi_df", test_multi_df)
```

```
========== train_single_df ==========
Shape: (7662, 5)
Columns: ['id', 'corpus', 'sentence', 'token', 'complexity']

Data Types:
id             object
corpus         object
sentence       object
token          object
complexity    float64
dtype: object

Missing Values (by column):
id            0
corpus        0
```

```
sentence      0
token         7
complexity    0
dtype: int64

'complexity' Column Stats (incl. quartiles and median):
count    7662.000000
mean        0.302288
std         0.132977
min         0.000000
25%         0.211538
50%         0.279412
75%         0.375000
max         0.861111
Name: complexity, dtype: float64

Quartile Frequency Counts (tab-separated next to each quartile):
25%: 0.2115384615384615 Count (<= Q1): 1928
50% (Median): 0.2794117647058823        Count (Q1 < x <= Q2): 1937
75%: 0.375      Count (Q2 < x <= Q3): 1984
100% (Max): 0.8611111111111112  Count (Q3 < x <= Max): 1813
====================================

========== train_multi_df ==========
Shape: (1517, 5)
Columns: ['id', 'corpus', 'sentence', 'token', 'complexity']

Data Types:
id             object
corpus         object
sentence       object
token          object
complexity    float64
dtype: object

Missing Values (by column):
id            0
corpus        0
sentence      0
token         0
complexity    0
dtype: int64

'complexity' Column Stats (incl. quartiles and median):
count    1517.000000
mean        0.418362
std         0.155536
min         0.027778
```

```
25%         0.302632
50%         0.409091
75%         0.529412
max         0.975000
Name: complexity, dtype: float64

Quartile Frequency Counts (tab-separated next to each quartile):
25%: 0.3026315789473685 Count (<= Q1): 382
50% (Median): 0.409090909090909 Count (Q1 < x <= Q2): 377
75%: 0.5294117647058824 Count (Q2 < x <= Q3): 380
100% (Max): 0.975       Count (Q3 < x <= Max): 378
====================================

========== trial_val_single_df ==========
Shape: (421, 5)
Columns: ['id', 'subcorpus', 'sentence', 'token', 'complexity']

Data Types:
id             object
subcorpus      object
sentence       object
token          object
complexity    float64
dtype: object

Missing Values (by column):
id            0
subcorpus     0
sentence      0
token         0
complexity    0
dtype: int64

'complexity' Column Stats (incl. quartiles and median):
count    421.000000
mean       0.298631
std        0.137619
min        0.000000
25%        0.214286
50%        0.266667
75%        0.359375
max        0.875000
Name: complexity, dtype: float64

Quartile Frequency Counts (tab-separated next to each quartile):
25%: 0.2142857142857143 Count (<= Q1): 106
50% (Median): 0.2666666666666667        Count (Q1 < x <= Q2): 107
75%: 0.359375   Count (Q2 < x <= Q3): 103
```

```
100% (Max): 0.875       Count (Q3 < x <= Max): 105
======================================

========== trial_val_multi_df ==========
Shape: (99, 5)
Columns: ['id', 'subcorpus', 'sentence', 'token', 'complexity']

Data Types:
id             object
subcorpus      object
sentence       object
token          object
complexity    float64
dtype: object

Missing Values (by column):
id            0
subcorpus     0
sentence      0
token         0
complexity    0
dtype: int64

'complexity' Column Stats (incl. quartiles and median):
count    99.000000
mean      0.417961
std       0.153752
min       0.000000
25%       0.309028
50%       0.421875
75%       0.513932
max       0.825000
Name: complexity, dtype: float64

Quartile Frequency Counts (tab-separated next to each quartile):
25%: 0.3090277777777778 Count (<= Q1): 25
50% (Median): 0.421875  Count (Q1 < x <= Q2): 25
75%: 0.5139318885448916 Count (Q2 < x <= Q3): 24
100% (Max): 0.825       Count (Q3 < x <= Max): 25
======================================

========== test_single_df ==========
Shape: (917, 5)
Columns: ['id', 'corpus', 'sentence', 'token', 'complexity']

Data Types:
id             object
corpus         object
```

```
sentence       object
token          object
complexity    float64
dtype: object

Missing Values (by column):
id            0
corpus        0
sentence      0
token         0
complexity    0
dtype: int64

'complexity' Column Stats (incl. quartiles and median):
count    917.000000
mean       0.296362
std        0.127290
min        0.000000
25%        0.214286
50%        0.276316
75%        0.357143
max        0.777778
Name: complexity, dtype: float64

Quartile Frequency Counts (tab-separated next to each quartile):
25%: 0.2142857142857143 Count (<= Q1): 237
50% (Median): 0.2763157894736842        Count (Q1 < x <= Q2): 224
75%: 0.3571428571428571 Count (Q2 < x <= Q3): 229
100% (Max): 0.7777777777777777  Count (Q3 < x <= Max): 227
======================================

========== test_multi_df ==========
Shape: (184, 5)
Columns: ['id', 'corpus', 'sentence', 'token', 'complexity']

Data Types:
id             object
corpus         object
sentence       object
token          object
complexity    float64
dtype: object

Missing Values (by column):
id            0
corpus        0
sentence      0
token         0
```

```
complexity    0
dtype: int64

'complexity' Column Stats (incl. quartiles and median):
count    184.000000
mean       0.422312
std        0.155785
min        0.000000
25%        0.316667
50%        0.428571
75%        0.527778
max        0.800000
Name: complexity, dtype: float64

Quartile Frequency Counts (tab-separated next to each quartile):
25%: 0.3166666666666666 Count (<= Q1): 47
50% (Median): 0.4285714285714286        Count (Q1 < x <= Q2): 46
75%: 0.5277777777777778 Count (Q2 < x <= Q3): 46
100% (Max): 0.8 Count (Q3 < x <= Max): 45
====================================
```

[76]:
```
print(train_single_df.head())
```

```
                              id corpus
sentence     token  complexity
0  3ZLW647WALVGE8EBR5OEGUBPU4P32A  bible  Behold, there came up out of the river
seven c…     river    0.000000
1  34ROBODSP1ZBN3DVY8J8XSIY551E5C  bible  I am a fellow bondservant with you and
with yo…  brothers    0.000000
2  3S1WOPCJFGTJU2SGNAN2Y213N6WJE3  bible  The man, the lord of the land, said to
us, 'By…  brothers    0.050000
3  3BFNCI9LYKQNO9BHXHH9CLSX5KP738  bible  Shimei had sixteen sons and six
daughters; but…  brothers    0.150000
4  3G5RUKN2EC3YIWSKUXZ8ZVH95R49N2  bible               "He has put my brothers
far from me.  brothers    0.263889
```

[77]:
```
print(train_multi_df.head())
```

```
                              id corpus
sentence            token  complexity
0  3S37Y8CWI8ON8KVM53U4E6JKCDC4WE  bible  but the seventh day is a Sabbath to
Yahweh you…      seventh day    0.027778
1  3WGCNLZJKF877FYC1Q6COKNWTDWD11  bible  But let each man test his own work,
and then h…          own work    0.050000
2  3UOMW19E6D6WQ5TH2HDD74IVKTP5CB  bible  To him who by understanding made the
heavens; …  loving kindness    0.050000
3  36JW4WBRO6KF9AXMUL4N476OMF8FHD  bible  Remember to me, my God, this also, and
spare m…  loving kindness    0.050000
```

```
4  3HRWUH63QU2FH9Q8R7MRNFC7JX2N5A  bible  Because your loving kindness is better
than li…  loving kindness    0.075000
```

[78]:
```
#@title Data Engineering
```

[79]:
```
# Assuming you have already loaded the DataFrames:
# train_single_df, train_multi_df, trial_val_single_df, trial_val_multi_df,␣
 ↪test_single_df, test_multi_df

def print_distinct_values(df, column_name):
    """Prints the distinct values of a specified column in a DataFrame."""
    distinct_values = df[column_name].unique()
    print(f"Distinct values in '{column_name}' column:")
    for value in distinct_values:
        print(value)
    print("-" * 30)  # Separator

# Print distinct values for each DataFrame
print_distinct_values(train_single_df, "corpus")
print_distinct_values(train_multi_df, "corpus")
print_distinct_values(trial_val_single_df, "subcorpus")
print_distinct_values(trial_val_multi_df, "subcorpus")
print_distinct_values(test_single_df, "corpus")
print_distinct_values(test_multi_df, "corpus")
```

```
Distinct values in 'corpus' column:
bible
biomed
europarl
------------------------------
Distinct values in 'corpus' column:
bible
biomed
europarl
------------------------------
Distinct values in 'subcorpus' column:
bible
biomed
europarl
------------------------------
Distinct values in 'subcorpus' column:
bible
biomed
europarl
------------------------------
Distinct values in 'corpus' column:
bible
biomed
```

```
europarl
------------------------------
Distinct values in 'corpus' column:
bible
biomed
europarl
------------------------------
```

0.1 standardize column headers: convert trial_val header from 'subcorpus' to 'corpus'

[80]:
```
# Rename the 'subcorpus' column to 'corpus'
trial_val_single_df = trial_val_single_df.rename(columns={'subcorpus':␣
 ↪'corpus'})
trial_val_multi_df = trial_val_multi_df.rename(columns={'subcorpus': 'corpus'})

# Verify the change (optional)
print(trial_val_single_df.columns)
print(trial_val_multi_df.columns)
```

```
Index(['id', 'corpus', 'sentence', 'token', 'complexity'], dtype='object')
Index(['id', 'corpus', 'sentence', 'token', 'complexity'], dtype='object')
```

[81]:
```
dataframes = [train_single_df, train_multi_df, trial_val_single_df,␣
 ↪trial_val_multi_df, test_single_df, test_multi_df]

# Get the headers (column names) of the first DataFrame as a reference
reference_headers = list(dataframes[0].columns)

# Loop through the remaining DataFrames and compare headers
all_headers_match = True
for df in dataframes[1:]:
    if list(df.columns) != reference_headers:
        all_headers_match = False
        print(f"Headers do not match for DataFrame: {df.head(0)}")  # Print␣
 ↪which DataFrame has different headers
        break  # Exit the loop if a mismatch is found

# Print the result
if all_headers_match:
    print("All DataFrames have matching headers.")
else:
    print("Headers do not match for all DataFrames.")
```

```
All DataFrames have matching headers.
```

0.2 Interrogate Span Length by Corpus Value by Data Split

[82]:
```
# Analyzing sentence spans by complexity quartile and corpus

tokenizer = RegexpTokenizer(r'\w+') # setup tokenizer

def analyze_sentence_spans_by_corpus_and_quartile(dfs_dict):
    """
    Analyze sentence spans (length metrics) grouped by corpus and complexity␣
 ↪quartile
    for multiple dataframes.
    """
    results = []

    for df_name, df in dfs_dict.items():
        print(f"Processing {df_name}...")

        # Calculate complexity quartiles for this dataframe
        q1 = df['complexity'].quantile(0.25)
        q2 = df['complexity'].quantile(0.50)
        q3 = df['complexity'].quantile(0.75)

        # Define quartile ranges for labeling
        def get_quartile(x):
            if x <= q1:
                return 'Q1'
            elif x <= q2:
                return 'Q2'
            elif x <= q3:
                return 'Q3'
            else:
                return 'Q4'

        # Add quartile column
        df = df.copy()
        df['quartile'] = df['complexity'].apply(get_quartile)

        # Compute sentence metrics using RegexpTokenizer instead of␣
 ↪word_tokenize
        def compute_span_metrics(sentence):
            if pd.isna(sentence):
                return pd.Series({'word_count': 0, 'char_count': 0,␣
 ↪'avg_word_len': 0})

            # Use our tokenizer that doesn't require punkt_tab
            words = tokenizer.tokenize(sentence)
            word_count = len(words)
```

```
            char_count = len(sentence)
            avg_word_len = np.mean([len(word) for word in words]) if word_count␣
 ↪> 0 else 0
            return pd.Series({'word_count': word_count, 'char_count':␣
 ↪char_count, 'avg_word_len': avg_word_len})

        # Apply the function to each sentence
        span_metrics = df['sentence'].apply(compute_span_metrics)
        df = pd.concat([df, span_metrics], axis=1)

        # Get corpus column name (could be 'corpus' or 'subcorpus')
        corpus_col = 'corpus' if 'corpus' in df.columns else 'subcorpus'

        # Group by corpus and quartile
        for corpus_name, corpus_df in df.groupby(corpus_col):
            for quartile, quartile_df in corpus_df.groupby('quartile'):
                # Calculate statistics
                complexity_range = f"{quartile_df['complexity'].min():.
 ↪3f}-{quartile_df['complexity'].max():.3f}"
                stats = {
                    'Dataframe': df_name,
                    'Corpus': corpus_name,
                    'Quartile': quartile,
                    'Complexity Range': complexity_range,
                    'Count': len(quartile_df),
                    'Avg Words': quartile_df['word_count'].mean(),
                    'Median Words': quartile_df['word_count'].median(),
                    'Min Words': quartile_df['word_count'].min(),
                    'Max Words': quartile_df['word_count'].max(),
                    'Std Words': quartile_df['word_count'].std(),
                    'Avg Chars': quartile_df['char_count'].mean(),
                    'Avg Word Len': quartile_df['avg_word_len'].mean()
                }
                results.append(stats)

    # Convert to DataFrame and sort
    results_df = pd.DataFrame(results)
    results_df = results_df.sort_values(['Dataframe', 'Corpus', 'Quartile'])

    return results_df

# Create dictionary of dataframes
dfs = {
    'train_single_df': train_single_df,
    'train_multi_df': train_multi_df,
    'trial_val_single_df': trial_val_single_df,
    'trial_val_multi_df': trial_val_multi_df,
```

```
    'test_single_df': test_single_df,
    'test_multi_df': test_multi_df
}

# Run analysis
span_analysis = analyze_sentence_spans_by_corpus_and_quartile(dfs)

# Display results
pd.set_option('display.max_rows', None)
pd.set_option('display.max_columns', None)
pd.set_option('display.width', 1000)
display(span_analysis)

# Save the analysis results
results_path = os.path.join(dir_results, 'sentence_span_analysis.csv')
span_analysis.to_csv(results_path, index=False)
print(f"Analysis saved to: {results_path}")
```

```
Processing train_single_df…
Processing train_multi_df…
Processing trial_val_single_df…
Processing trial_val_multi_df…
Processing test_single_df…
Processing test_multi_df…
```

	Dataframe	Corpus	Quartile	Complexity Range	Count	Avg Words	Median Words	Min Words	Max Words	Std Words	Avg Chars	Avg Word Len
60	test_multi_df	bible	Q1	0.025-0.317	26	23.076923	22.0	4.0	48.0	11.831900	118.653846	4.128898
61	test_multi_df	bible	Q2	0.325-0.417	11	20.545455	17.0	7.0	47.0	12.917923	109.545455	4.209752
62	test_multi_df	bible	Q3	0.432-0.528	18	21.111111	21.5	4.0	43.0	10.889222	112.777778	4.474206
63	test_multi_df	bible	Q4	0.542-0.694	11	22.363636	20.0	7.0	51.0	11.935432	126.181818	4.605062
64	test_multi_df	biomed	Q1	0.000-0.312	11	29.818182	29.0	17.0	47.0	8.388304	195.727273	5.491145
65	test_multi_df	biomed	Q2	0.324-0.417	11	27.090909	24.0	9.0	47.0	11.449494	171.818182	5.436237
66	test_multi_df	biomed	Q3	0.456-0.528	10	26.900000	26.5	10.0	49.0	10.712921	177.500000	5.497409
67	test_multi_df	biomed	Q4	0.562-0.800	21	32.285714	34.0	14.0	56.0	13.598319	209.285714	5.460101
68	test_multi_df	europarl	Q1	0.214-0.303	10	24.700000	24.5	7.0	56.0	14.189589	146.900000	5.049688
69	test_multi_df	europarl	Q2	0.321-0.429	24	27.833333	27.0	9.0	73.0	15.352855	172.291667	5.269610

```
70        test_multi_df  europarl       Q3      0.432-0.516     18  32.944444 ␣
 ↪      32.0         6.0       68.0  19.129504  209.888889      5.512245
71        test_multi_df  europarl       Q4      0.531-0.562     13  39.000000 ␣
 ↪      36.0         6.0       95.0  29.631065  237.076923      5.100616
48       test_single_df     bible       Q1      0.000-0.214     79  22.835443 ␣
 ↪      22.0         7.0       49.0  10.602891  116.797468      4.031532
49       test_single_df     bible       Q2      0.217-0.276     68  24.176471 ␣
 ↪      21.0         2.0       77.0  14.393138  125.955882      4.167352
50       test_single_df     bible       Q3      0.278-0.353     67  22.388060 ␣
 ↪      20.0         4.0       63.0  11.306950  119.731343      4.254090
51       test_single_df     bible       Q4      0.359-0.732     69  20.579710 ␣
 ↪      19.0         1.0       55.0  11.264736  110.550725      4.337010
52       test_single_df    biomed       Q1      0.000-0.214     75  27.080000 ␣
 ↪      25.0        10.0       84.0  12.025603  172.893333      5.271985
53       test_single_df    biomed       Q2      0.217-0.275     58  30.275862 ␣
 ↪      26.0        10.0       83.0  15.856587  197.775862      5.434573
54       test_single_df    biomed       Q3      0.278-0.357     66  29.833333 ␣
 ↪      29.0        13.0       85.0  11.754650  191.863636      5.334048
55       test_single_df    biomed       Q4      0.359-0.778     90  31.144444 ␣
 ↪      30.0        14.0       83.0  12.089146  203.055556      5.393138
56       test_single_df  europarl       Q1      0.000-0.214     83  25.337349 ␣
 ↪      21.0         3.0       82.0  16.032191  151.891566      5.044222
57       test_single_df  europarl       Q2      0.217-0.276     98  32.326531 ␣
 ↪      30.0         1.0       97.0  18.707061  195.653061      5.062296
58       test_single_df  europarl       Q3      0.278-0.357     96  33.000000 ␣
 ↪      30.0         3.0      141.0  21.404377  201.760417      5.124551
59       test_single_df  europarl       Q4      0.361-0.583     68  33.235294 ␣
 ↪      29.0         1.0      130.0  20.440023  206.514706      5.164123
12        train_multi_df     bible       Q1      0.028-0.300    163  23.588957 ␣
 ↪      22.0         3.0       67.0  12.429421  124.834356      4.232989
13        train_multi_df     bible       Q2      0.304-0.409    132  24.053030 ␣
 ↪      22.0         6.0       65.0  11.738444  129.575758      4.302615
14        train_multi_df     bible       Q3      0.411-0.529    131  23.770992 ␣
 ↪      23.0         4.0       50.0  11.158691  127.389313      4.324088
15        train_multi_df     bible       Q4      0.533-0.778     79  25.481013 ␣
 ↪      24.0         3.0       81.0  13.490605  139.240506      4.486716
16        train_multi_df    biomed       Q1      0.028-0.303     87  29.091954 ␣
 ↪      28.0         9.0       77.0  11.882792  185.954023      5.276290
17        train_multi_df    biomed       Q2      0.304-0.408     74  30.716216 ␣
 ↪      28.0        11.0       85.0  13.521693  195.864865      5.370313
18        train_multi_df    biomed       Q3      0.411-0.529    111  29.783784 ␣
 ↪      29.0         8.0       61.0  10.912383  193.855856      5.430133
19        train_multi_df    biomed       Q4      0.531-0.975    242  29.595041 ␣
 ↪      28.0        10.0       75.0  12.040443  194.995868      5.534629
20        train_multi_df  europarl       Q1      0.118-0.303    132  29.363636 ␣
 ↪      27.0         3.0      101.0  17.874146  176.553030      5.002618
```

```
21       train_multi_df  europarl       Q2      0.304-0.409    171  31.654971 ␣
 ↪      28.0        3.0      108.0  19.099221  195.152047      5.176834
22       train_multi_df  europarl       Q3      0.411-0.529    138  33.398551 ␣
 ↪      30.0        7.0      101.0  18.992715  208.304348      5.286607
23       train_multi_df  europarl       Q4      0.533-0.750     57  34.596491 ␣
 ↪      31.0        6.0       96.0  20.318763  218.350877      5.345891
0       train_single_df     bible       Q1      0.000-0.212    701  23.275321 ␣
 ↪      22.0        4.0       61.0  11.760701  121.607703      4.126789
1       train_single_df     bible       Q2      0.212-0.279    640  23.753125 ␣
 ↪      22.0        3.0       60.0  11.577932  124.576562      4.148961
2       train_single_df     bible       Q3      0.281-0.375    624  23.823718 ␣
 ↪      22.0        3.0       70.0  11.958906  126.230769      4.208102
3       train_single_df     bible       Q4      0.380-0.861    609  23.577997 ␣
 ↪      21.0        3.0       69.0  12.461688  126.518883      4.295608
4       train_single_df    biomed       Q1      0.000-0.212    586  28.534130 ␣
 ↪      27.0        2.0       85.0  12.115387  182.011945      5.319754
5       train_single_df    biomed       Q2      0.212-0.279    583  30.435678 ␣
 ↪      29.0        7.0       92.0  11.872558  193.789022      5.285758
6       train_single_df    biomed       Q3      0.281-0.375    659  29.860395 ␣
 ↪      28.0        4.0       77.0  11.591263  191.050076      5.328161
7       train_single_df    biomed       Q4      0.381-0.861    748  29.176471 ␣
 ↪      28.0        3.0       85.0  12.246613  186.909091      5.298112
8       train_single_df  europarl       Q1      0.025-0.212    641  26.761310 ␣
 ↪      24.0        2.0      107.0  15.230853  159.180967      4.942557
9       train_single_df  europarl       Q2      0.212-0.279    714  30.420168 ␣
 ↪      27.0        1.0      129.0  18.383783  183.093838      4.995672
10      train_single_df  europarl       Q3      0.281-0.375    701  30.523538 ␣
 ↪      28.0        1.0      122.0  18.163026  185.840228      5.114587
11      train_single_df  europarl       Q4      0.381-0.775    456  33.528509 ␣
 ↪      31.0        2.0      235.0  21.704693  203.592105      5.054701
36   trial_val_multi_df     bible       Q1      0.000-0.292     11  26.272727 ␣
 ↪      21.0       13.0       64.0  13.950562  141.363636      4.282457
37   trial_val_multi_df     bible       Q2      0.333-0.400      7  20.571429 ␣
 ↪      23.0        5.0       28.0   7.412987  110.857143      4.279406
38   trial_val_multi_df     bible       Q3      0.425-0.500      5  19.600000 ␣
 ↪      19.0        9.0       32.0   8.905055  109.200000      4.431391
39   trial_val_multi_df     bible       Q4      0.525-0.661      6  22.333333 ␣
 ↪      20.5        9.0       44.0  12.242004  117.833333      4.178525
40   trial_val_multi_df    biomed       Q1      0.083-0.303      6  26.833333 ␣
 ↪      25.0       15.0       49.0  11.771434  159.166667      4.899969
41   trial_val_multi_df    biomed       Q2      0.317-0.422      7  25.428571 ␣
 ↪      21.0       15.0       48.0  11.588171  156.000000      5.194383
42   trial_val_multi_df    biomed       Q3      0.438-0.513      6  37.833333 ␣
 ↪      39.5       26.0       44.0   6.675827  247.500000      5.438593
43   trial_val_multi_df    biomed       Q4      0.537-0.825     14  30.642857 ␣
 ↪      29.5       17.0       43.0   9.849695  211.428571      5.730623
```

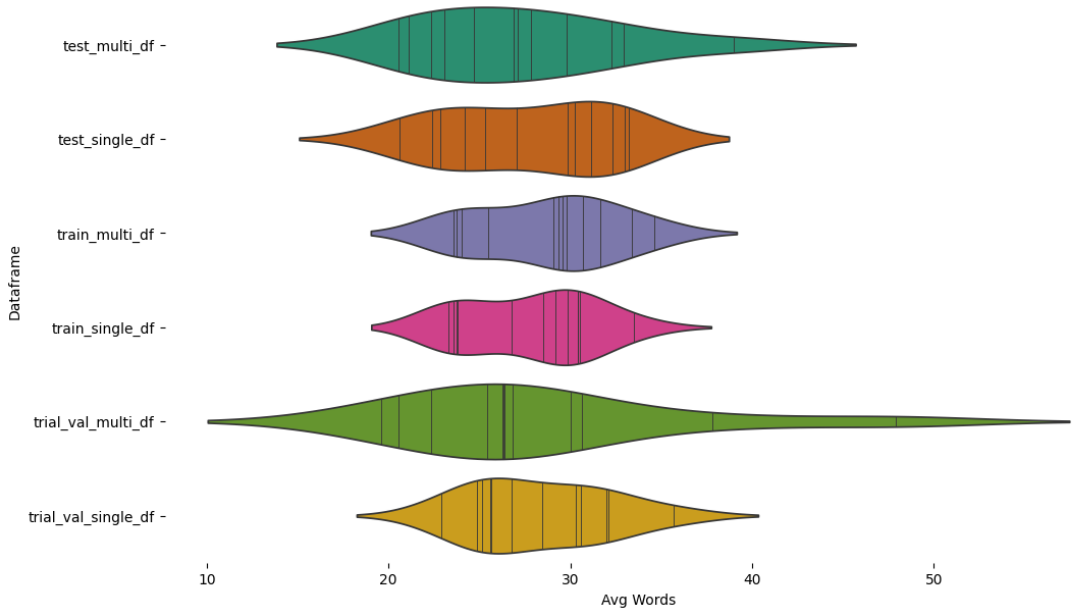
```
44   trial_val_multi_df  europarl       Q1      0.176-0.306      8  30.000000
 ↪      25.5        4.0       64.0  20.361027  186.750000      5.306837
45   trial_val_multi_df  europarl       Q2      0.312-0.412     11  47.909091
 ↪      46.0       24.0       78.0  18.651834  296.909091      5.058375
46   trial_val_multi_df  europarl       Q3      0.432-0.500     13  26.307692
 ↪      26.0        5.0       66.0  18.167666  166.153846      5.263847
47   trial_val_multi_df  europarl       Q4      0.515-0.714      5  26.400000
 ↪      15.0        6.0       66.0  24.316661  164.600000      4.998182
24  trial_val_single_df     bible       Q1      0.000-0.214     52  26.750000
 ↪      26.0        5.0       73.0  15.530962  137.230769      4.071006
25  trial_val_single_df     bible       Q2      0.217-0.266     38  24.868421
 ↪      23.0        7.0       50.0  10.768249  131.236842      4.195550
26  trial_val_single_df     bible       Q3      0.268-0.355     26  22.884615
 ↪      20.5        5.0       44.0   9.961233  121.269231      4.312026
27  trial_val_single_df     bible       Q4      0.361-0.633     27  25.666667
 ↪      23.0        6.0       49.0  12.554497  137.555556      4.212685
28  trial_val_single_df    biomed       Q1      0.028-0.214     21  25.571429
 ↪      21.0       13.0       65.0  11.543706  163.904762      5.305404
29  trial_val_single_df    biomed       Q2      0.217-0.267     28  30.571429
 ↪      27.5       11.0       57.0  12.099674  198.142857      5.315287
30  trial_val_single_df    biomed       Q3      0.268-0.359     38  32.105263
 ↪      29.0       11.0       61.0  12.710476  206.947368      5.364934
31  trial_val_single_df    biomed       Q4      0.364-0.875     48  25.145833
 ↪      25.5        6.0       56.0  11.721937  163.979167      5.439709
32  trial_val_single_df  europarl       Q1      0.050-0.214     33  31.969697
 ↪      28.0        5.0       81.0  20.356947  185.969697      4.799024
33  trial_val_single_df  europarl       Q2      0.217-0.267     41  28.463415
 ↪      28.0        4.0       71.0  15.386841  172.780488      4.997706
34  trial_val_single_df  europarl       Q3      0.268-0.359     39  30.282051
 ↪      28.0        3.0       99.0  20.040681  184.358974      5.086945
35  trial_val_single_df  europarl       Q4      0.367-0.605     30  35.700000
 ↪      30.5        5.0       77.0  20.142852  215.400000      4.910759

Analysis saved to:
/content/drive/MyDrive/266-final/results/sentence_span_analysis.csv
```

[82]:

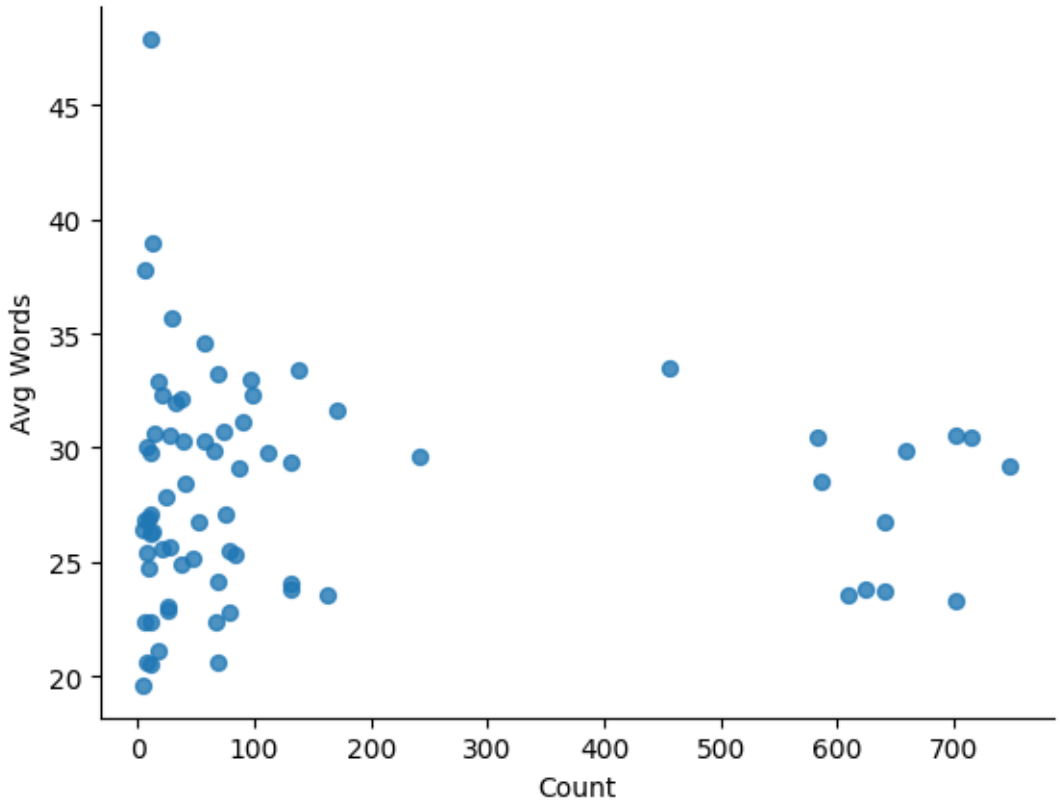
[82]:

[83]:
```
from matplotlib import pyplot as plt
import seaborn as sns
figsize = (12, 1.2 * len(span_analysis['Dataframe'].unique()))
plt.figure(figsize=figsize)
sns.violinplot(span_analysis, x='Avg Words', y='Dataframe', inner='stick',␣
 ↪palette='Dark2')
sns.despine(top=True, right=True, bottom=True, left=True)
```

```
<ipython-input-83-00a8ad5642c1>:5: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.violinplot(span_analysis, x='Avg Words', y='Dataframe', inner='stick',
palette='Dark2')
```

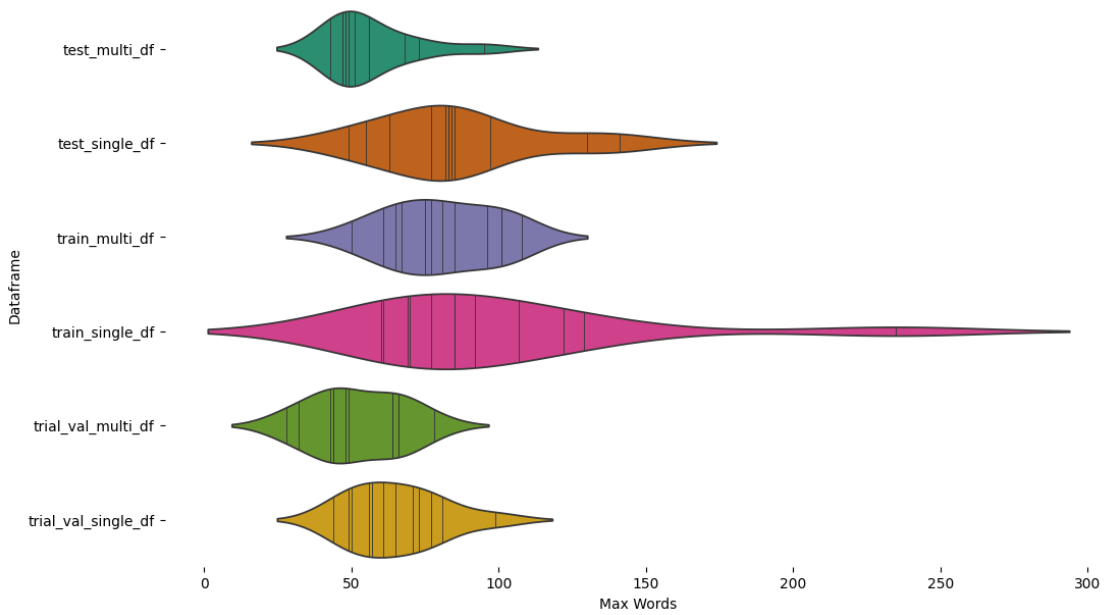


[84]:
```
from matplotlib import pyplot as plt
span_analysis.plot(kind='scatter', x='Count', y='Avg Words', s=32, alpha=.8)
plt.gca().spines[['top', 'right',]].set_visible(False)
```



[85]:
```
from matplotlib import pyplot as plt
import seaborn as sns
figsize = (12, 1.2 * len(span_analysis['Dataframe'].unique()))
plt.figure(figsize=figsize)
sns.violinplot(span_analysis, x='Max Words', y='Dataframe', inner='stick',␣
↪palette='Dark2')
sns.despine(top=True, right=True, bottom=True, left=True)
```

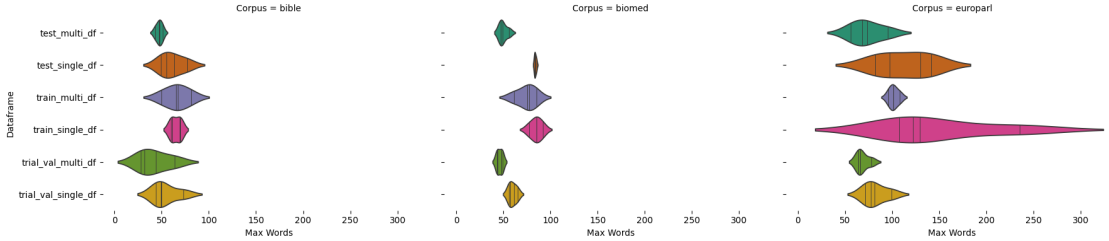
```
<ipython-input-85-01bf0c89d620>:5: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.violinplot(span_analysis, x='Max Words', y='Dataframe', inner='stick',
palette='Dark2')
```



[86]:
```
# Create a FacetGrid
g = sns.FacetGrid(span_analysis, col="Corpus", col_wrap=3, height=4, aspect=1.
↪5)  # Adjust col_wrap and height as needed

# Map the violinplot to the FacetGrid
g.map(sns.violinplot, "Max Words", "Dataframe", inner='stick', palette='Dark2')

# Remove spines for cleaner look
g.despine(top=True, right=True, bottom=True, left=True)

# Adjust layout and display the plot
plt.tight_layout()
plt.show()
```

```
/usr/local/lib/python3.11/dist-packages/seaborn/axisgrid.py:718: UserWarning:
Using the violinplot function without specifying `order` is likely to produce an
incorrect plot.
  warnings.warn(warning)
/usr/local/lib/python3.11/dist-packages/seaborn/axisgrid.py:854: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  func(*plot_args, **plot_kwargs)
/usr/local/lib/python3.11/dist-packages/seaborn/axisgrid.py:854: FutureWarning:
```

```
Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  func(*plot_args, **plot_kwargs)
/usr/local/lib/python3.11/dist-packages/seaborn/axisgrid.py:854: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  func(*plot_args, **plot_kwargs)
```



[86]:

[86]: