HAMLET CHALLENGE

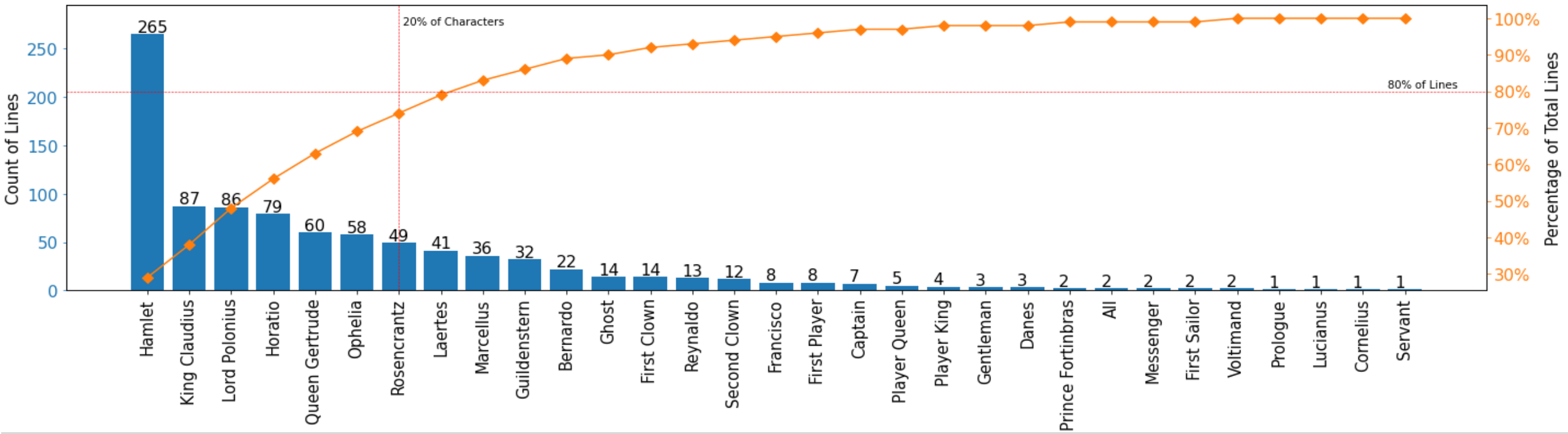
Prepared by: Cenan Altunay
Date: 25/06/2021

1. Objective / Background

Quick analysis of Hamlet, a play written in early 1600s by Shakespeare who is infamous for the vocabulary he used.

2. Lines per Character

* A line is an uninterrupted speech by a single character.

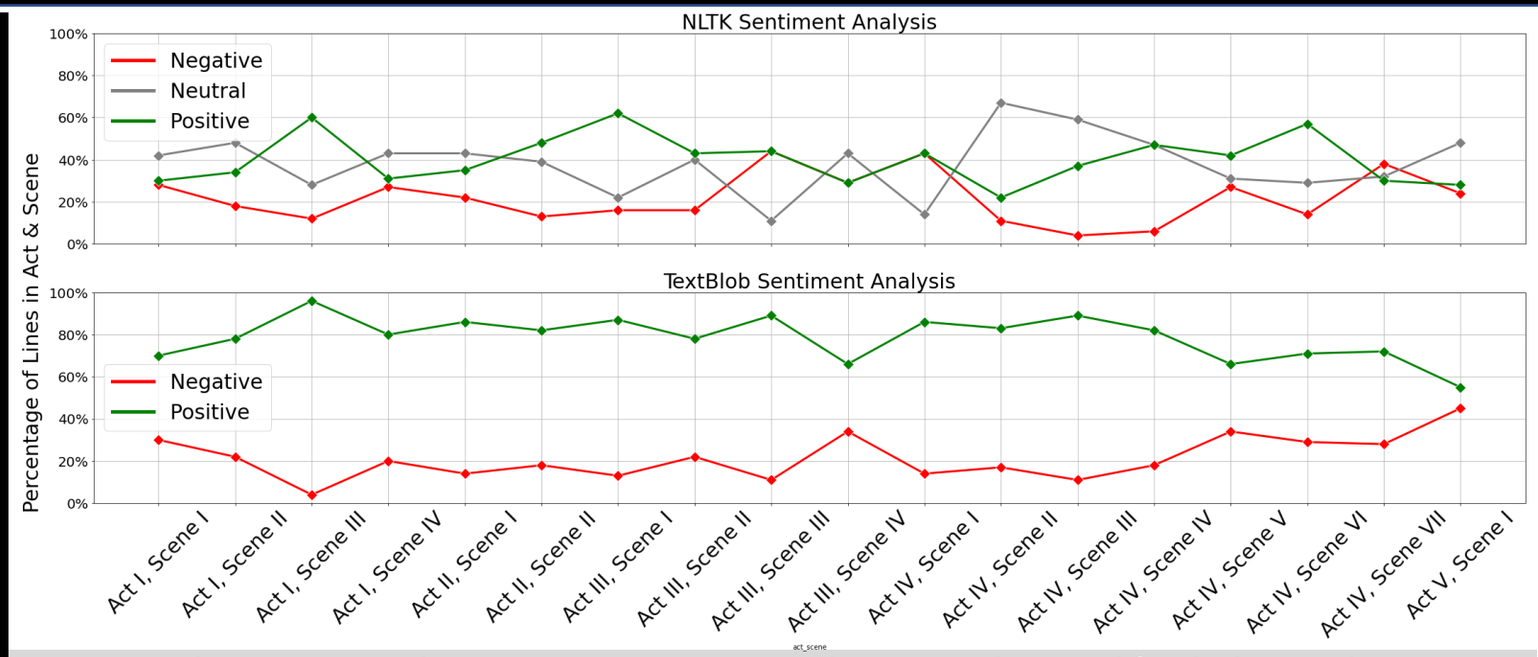


Key point: 80% of the lines are between just under 25% of characters, -*where Pareto Rule roughly applies.*

3. Unusual Words - WordCloud

* Unusual words are that are not found in a range of books written between 1800 to date.

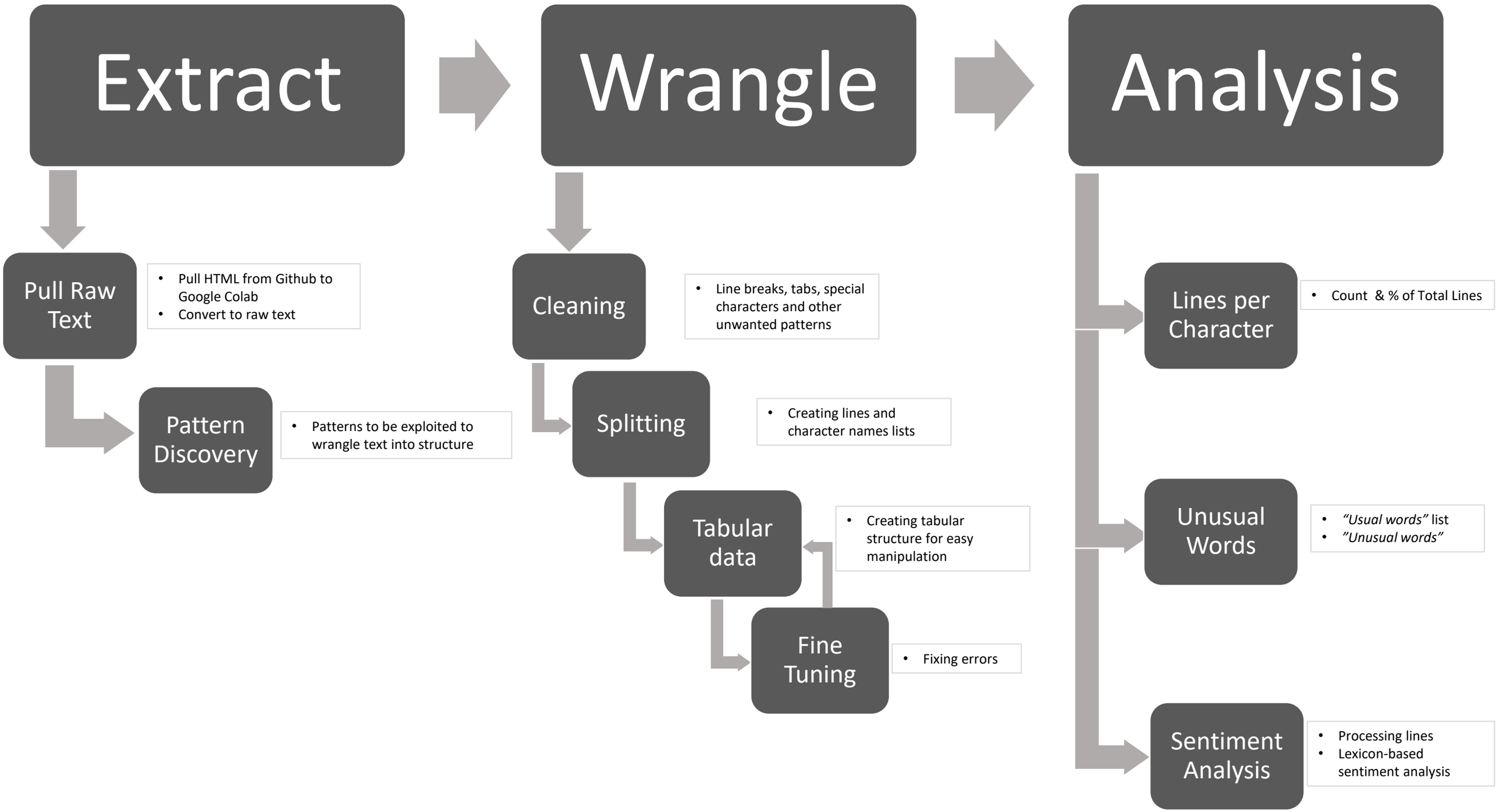
4. Sentiment Analysis per Act



Key point: Positive and negative sentiments start close, spread out and finish close, indicating climactic parts and storytelling style.

Behind the “Stage”

1. Workflow


Extract
Wrangle
Analysis
Pull Raw Text
Pull HTML from Github to Google Colab
Convert to raw text
Pattern Discovery
Patterns to be exploited to wrangle text into structure
Cleaning
Line breaks, tabs, special characters and other unwanted patterns
Splitting
Creating lines and character names lists
Tabular data
Creating tabular structure for easy manipulation
Fine Tuning
Fixing errors
Lines per Character
Count & % of Total Lines
Unusual Words
"Usual words" list
"Unusual words"
Sentiment Analysis
Processing lines
Lexicon-based sentiment analysis

Workflow Breakdown

Extract

Pull Raw Text

```
[23]  # The report is intended to describe your analysis and
      # present your results for a mixed technical audience.

      # Show the number of lines said per character throughout the play.

      # Make a word cloud of unusual words in the script

      # Compute the sentiment score for each scene.

      hamlet_url = 'https://raw.githubusercontent.com/eliiza/challenge-hamlet/master/hamlet.txt'
```

Text Cleaning & Wrangling

```
[24]  import pandas as pd
      import numpy as np
      import requests
      import re
      import matplotlib.pyplot as plt
      import seaborn as sns
```

```
[25]  hamlet = requests.get(hamlet_url).text
      hamlet[:500]
```

```
'\tHAMLET\n\n\n\tDRAMATIS PERSONAE\n\n\nCLAUDIUS\tking of Denmark. (KING CLAUDIUS:)\n\nHAMLET\tson to the late, and nephew to the present kin
g.\n\nPOLONIUS\tlord chamberlain. (LORD POLONIUS:)\n\nHORATIO\tfriend to Hamlet.\n\nLAERTES\tson to Polonius.\n\nLUCIANUS\tnephew to the kin
g.\n\n\nVOLTIMAND\t|\n\t|\nCORNELIUS\t|\n\t|\nROSENCRANTZ\t|  courtiers.\n\t|\nGUILDENSTERN\t|\n\t|\nOSRIC\t|\n\n\n\tA Gentleman, (Gentlemen:)
\n\n\tA Priest. (First Priest:)\n\n\nMARCELLUS\t|\n\t|  officers.\nBERNARDO\t|\n\n\nFRANCISCO\ta soldier.\n\nREYNALDO\tservant to Polonius.\n
\tPla'
```

Pattern Discovery

```
[ ]   print(hamlet[:500])
```

```
        HAMLET


        DRAMATIS PERSONAE


CLAUDIUS        king of Denmark. (KING CLAUDIUS:)

HAMLET  son to the late, and nephew to the present king.

POLONIUS        lord chamberlain. (LORD POLONIUS:)

HORATIO friend to Hamlet.

LAERTES son to Polonius.

LUCIANUS        nephew to the king.
```

- Tabs *("\t")* between characters and lines
- Double line breaks *("\n\n")* between line and next line's character – *as called out in yellow in screenshot*
- Tab and Pipe *("\t|")* for simultaneous lines
- Scenes and Acts starts with a pattern of a line breaks, tabs and words combinations
- Brackets for scene prompts

Workflow Breakdown

Wrangle

Cleaning

- Cleaning the text to leave only character name and line, exploiting identified patterns before

```
1 pattern = '\n\t\[[^\]]*\]\n*\tHAMLET\n*ACT.*\n*SCENE.*\n*\t\[[^\]]*\]\n' # matches scene and act starting points -except act 1
2 pattern_brackets = '\t\[[^\]]*\]|\[[^\]]*\]'# matches prompts in brackets
3
4 # split intro and book
5 book = hamlet[1004:]
6 intro = hamlet[:1005]
7
8 clean = re.sub(pattern, '', book) # removal of scene entries
9 clean = re.sub(pattern_brackets, '', clean[70:]) # removal of prompts and choosing starting point at the first line in book
```

```
[6] 1 clean = re.sub('\n{5}|\n{4}|\n{3}', '\n\n', clean) # swap 3 or more line breaks with 2 only, to be able to split by character lines
    2 clean = re.sub('\t\|', '', clean) # clean lines where multiple characters have say simultaneously
    3 clean_list = clean.split('\n\n') # split for most lines by double line breaks
```

Splitting

- Leveraging tab as a delimiter between character and line
- Creating a list of character names and a list of lines with the same sequence from the book

```
1 # fix simultaneous lines by creating 1 line per each of those characters
2 # create character and line list
3
4 character = []
5 lines = []
6
7 for line_ in clean_list:
8     line = re.split('\t', line_)
9
10    # identify problematic simultaneous lines
11    # split them by line break between
12    # add line to both character names
13    if len(line) == 1 and len(re.findall('\w.*\w', line[0])) != 0 and (line[0].split('\n')[0].isupper() and line[0].split('\n')[-1].isupper()) and len(line[0].split('\n')) > 1:
14        problem_list = line[0].split('\n')
15        character.append(problem_list[0])
16        lines.append(problem_list[1])
17        character.append(problem_list[-1])
18        lines.append(problem_list[1])
19    # if not a simultaneous line, add tab delimeter output to character and lines lists
20    else:
21        character.append(line[0])
22        lines.append(''.join(line[1:]))
23
24 print(f'Char list length:\t {len(character)} \nLines list length:\t {len(lines)}')
25 print(f'\n\nCharacter and Lines length match?: {len(character) == len(lines)}')
```

```
Char list length:       988
Lines list length:      988


Character and Lines length match?: True
```

Workflow Breakdown

Wrangle –*cont'd*

Tabular data

- Using lists created, forming a tabular data structure of characters and lines

- There are apparent errors where lines is mixed in “chars” field as highlighted

```
1 # create data frame from lists
2 data = {'chars':character, 'line':lines}
3 df_ = pd.DataFrame(data)
4
5 df_['chars_len'] = df_['chars'].apply(len) # add character name length, to identify and fix where tab delimiter was not enough
6 df_.sort_values(by = 'chars_len', ascending = True).sample(15)
```

	chars	line	chars_len
656		O, the recorders! let me see one. To withdraw ...	0
761	ROSENCRANTZ:	Hamlet! Lord Hamlet!	12
85	HORATIO	Hail to your lordship!	7
901	LAERTES	Do you see this, O God?	7
180	HAMLET	Ay, marry, is't:\nBut to my mind, though I am ...	6
893	LAERTES	A document in madness, thoughts and remembranc...	7
402	HAMLET	Then are our beggars bodies, and our monarchs ...	6
912	HORATIO 'Horatio, when thou shalt have overlo...	this, give these fellows some means to the kin...	51
[illegible]		I do not know from what part of the world\nI s...	0
867	LAERTES	I thank you: keep the door. O thou vile king,\...	7
612	HAMLET	So you must take your husbands. Begin, murdere...	6
436	GUILDENSTERN	In what, my dear lord?	12
637	HAMLET	Your wisdom should show itself more richer to\...	6
480		My good friends, I'll leave you till night: yo...	0
936	LAERTES	Upon my life, Lamond.	7

Workflow Breakdown

Wrangle –*cont'd*

Fine Tuning

- Using characters names, extracting lines and appending them to the respective row in “line” field while leaving only name in ”chars” field

	chars	line	chars_len
364	LORD POLONIUS How say you by that? Still harp...	daughter: yet he knew me not at first; he said...	56
690	KING CLAUDIUS My words fly up, my thoughts re...	Words without thoughts never to heaven go.	58
370	LORD POLONIUS Though this be madness, yet the...	in 't. Will you walk out of the air, my lord?	59

```
# fix remaining issues in char names
# split char column instances where line is included
pattern = 'HAMLET|HORATIO|LORD POLONIUS|ROSENCRANTZ|KING CLAUDIUS|Danes'

def split_char_line(df_):

    for ind, val in df_.iterrows(): # loop over dataframe

        if val['chars'].find('\n') != -1: # identify and remove line breaks in char column
            df_.loc[ind, 'chars'] = re.sub('\n', '', val['chars'])

        if val['chars_len']>20: # when line is included to char name, length is over 20
            char = re.findall(pattern, val['chars']) # find character name
            line = ''.join(re.split(pattern, val['chars'])[1:]) # find line by character
            line = line + ' ' + val['line'] # if there's a line existing in line column, append them

            # fix char and line columns
            df_.loc[ind, 'chars'] = char
            df_.loc[ind, 'line'] = line

        # fix ghost lines
        if val['chars'] == 'Ghost  Swear.':
            df_.loc[ind, 'chars'] = 'Ghost'
            df_.loc[ind, 'line'] = 'Swear.'

        # replace blanks rows in char column with np.nan
        if val['chars'] == '':
            df_.loc[ind, 'chars'] = np.nan

    # if line is blank, swap with null
    for ind, val in df_.iterrows():
        if val['line'] == '':
            df_.loc[ind, 'line'] = np.nan

    return df_

df_ = split_char_line(df_)

df_.sort_values(by = 'chars_len', ascending = True).tail(3)
```

	chars	line	chars_len
364	LORD POLONIUS	How say you by that? Still harping on my\n d...	56
690	KING CLAUDIUS	My words fly up, my thoughts remain below:\n...	58
370	LORD POLONIUS	Though this be madness, yet there is method\...	59

Workflow Breakdown

Wrangle *–cont'd*

Fine Tuning

- Another issue is caused by the delimiter (tab). Some lines include a tab, and they ended up splitting a characters line into pieces *–as per below*

```
1 df_.loc[884:885]
```

	chars	line	chars_len
884	LAERTES	How now! what noise is that?	7
885		O heat, dry up my brains! tears seven times sa...	0

Fine Tuning

- To make use easier, structure should be as 1 observation per row, in this case one character and line per row, sharing the sequence from the book.
- Merging each partial line from a character to their preceding partial line *-sample highlighted on the side.*

```python
# fix above issue by appending all lines with null chars field to its owner
def merge_lines(df):
    '''takes dataframe, starting iterating from bottom row,
    appends line to above line where, chars field is null
    when chars field is not null, append below cells' to that row and line field'''

    df['line'] = df['line'].replace(np.nan, '') # if line is null, substitute with blank field for appending

    df['chars'] = df['chars'].replace('\s{2}|\n|\:', '', regex = True) # clean chars field for multiple whitespaces
    df['chars'] = df['chars'].str.title() # standardise chars field to lowercase

    for i in np.arange(len(df)-1, -1, -1): # loop from end

        if pd.isna(df.loc[i, 'chars']) or df.loc[i,'chars'] == '': # identify null or blanks chars field
            current_line = df.loc[i, 'line'] # take line field of row
            prev_line = df.loc[i - 1, 'line'] # take line field of rov above

            replacing_line = prev_line + ' ' + current_line # append them in correct order

            df.loc[i - 1, 'line'] = replacing_line # assign appended line to above cell


    return df
```

```python
[47] df = df_.copy() # keep the original dataframe here efficiency

df.drop(index = 0, inplace = True) # drop first null row manually
df.drop(axis = 1, labels = 'chars_len', inplace = True) # no need for chars_len field anymore
df.reset_index(drop = True, inplace =True) #

df = merge_lines(df)

print('Previous version\n')
print(df_.loc[884,'line'])
print('=======\n=======')
print('\nNew version\n')
print(df.loc[883,'line'])
```

```
Previous version

How now! what noise is that?
=======
=======

New version

How now! what noise is that? O heat, dry up my brains! tears seven times salt,
Burn out the sense and virtue of mine eye!
By heaven, thy madness shall be paid by weight,
Till our scale turn the beam. O rose of May!
Dear maid, kind sister, sweet Ophelia!
O heavens! is't possible, a young maid's wits
Should be as moral as an old man's life?
Nature is fine in love, and where 'tis fine,
It sends some precious instance of itself
After the thing it loves.
```

Workflow Breakdown

Wrangle –*cont'd*

Tabular Data

- Assigning act and scenes for each line
- Searching for first line of each scene, and assigning the relative act to that row.
- Since data is in book's order, filling between the assigned scenes by 'extrapolating' the last assigned act

```python
def act_scene_assign(df):
    """takes in dataframe, iterates over rows,
    if identifes first sentence of any act/scene,
    marks it by act/scene number in new column 'act_scene'.

    after iteretion, forward fills acts, since our dataframe is in same sequence with the book
    and since lines with null chars field are  appended to it's owner, drops rows, where chars field is null or nan

    resets index to avoid future confusion
    """

    for ind, val  in df.iterrows():
        for scene_line, scene_tag in zip(scenes, scene_tags):
            if len(re.findall(r'^'+scene_line, val['line'])) != 0:
                df.loc[ind, 'act_scene'] =  scene_tag


    df['act_scene'] = df['act_scene'].ffill() # forward fill acts
    df['chars'] = df['chars'].replace('', np.nan) # replace chars with nan to drop rows
    df.dropna(how = 'any', axis = 0, inplace = True) # drop rows where  chars field is null
    df.reset_index(drop = True, inplace = True) # reset index

    return df

df = act_scene_assign(df)
df
```

	chars	line	act_scene
0	Bernardo	Who's there?	Act I, Scene I
1	Francisco	Nay, answer me: stand, and unfold yourself.	Act I, Scene I
2	Bernardo	Long live the king!	Act I, Scene I
3	Francisco	Bernardo?	Act I, Scene I
4	Bernardo	He.	Act I, Scene I
...	...	...	...
915	First Clown	Cudgel thy brains no more about it, for your d...	Act V, Scene I
916	Hamlet	Has this fellow no feeling of his business, th...	Act V, Scene I
917	Horatio	Custom hath made it in him a property of easin...	Act V, Scene I
918	Hamlet	'Tis e'en so: the hand of little employment ha...	Act V, Scene I
919	First Clown	But age, with his stealing steps,\nHath claw'...	Act V, Scene I

Workflow Breakdown

Analysis

Lines per Character

- Tabular data design is modelled for easy manipulation.
- ‘Pivot’ing by character names and counting how many lines they have gives the line per character *(“size” field in screenshot)*
- Using a running total of percentage, we can also see if most lines are between a small portion of the characters *(“pct” field in screenshot)*

Line Count per Character

- Counting lines will be easy now

- Simply grouping by character and taking count

```
# group by character and count instances

line_count = df.groupby(by = 'chars', as_index = False).size().sort_values(by = 'size', ascending = False)

# will also create a pareto chart to see if most of the lines are piled up between a small number of characters
line_count['pct'] = round(line_count['size'].cumsum() / line_count['size'].sum() * 100)
line_count = line_count.set_index('chars') # set index to chars, for easier plotting

# sanity check
line_count
```

chars	size	pct
Hamlet	260	28.0
King Claudius	85	38.0
Lord Polonius	80	46.0
Horatio	77	55.0
Queen Gertrude	60	61.0
Ophelia	58	67.0
Rosencrantz	47	72.0
Laertes	41	77.0
Marcellus	36	81.0
Guildenstern	32	84.0
Bernardo	22	87.0
First Clown	14	88.0
Reynaldo	13	90.0
Second Clown	12	91.0
Ghost	10	92.0
First Player	8	93.0
Francisco	8	94.0
Captain	7	95.0
Player Queen	5	95.0
Ghostswear.	4	96.0
Player King	4	96.0

Workflow Breakdown

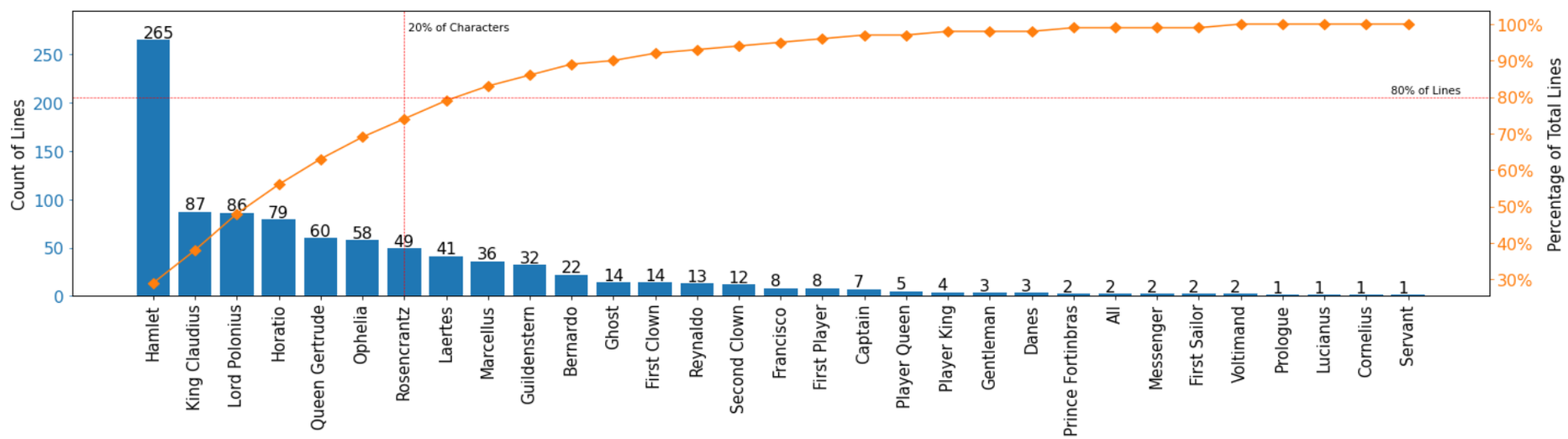
Analysis –*cont'd*

Lines per Character

- Using line count, producing a chart to make it easier to digest the information.

```python
from matplotlib.ticker import PercentFormatter

fig, ax = plt.subplots(figsize=(20,6)) # start object oriented figure

ax.bar(line_count.index, line_count["size"], color="C0") # on ax, create our bar chart with raw count

ax2 = ax.twinx() # start secondary axis

ax2.plot(line_count.index, line_count["pct"], color="C1", marker="D", ms=7) # create line chart, with markers
ax2.yaxis.set_major_formatter(PercentFormatter()) # prettify percentage ticks on axis

# prettify axes
ax.tick_params(axis="y", colors="C0", labelsize = 15)
ax.tick_params(axis = 'x', labelrotation = 90, labelsize = 15)
ax2.tick_params(axis="y", colors="C1", labelsize = 15)
ax.set_ylim(0, 295)

# add pareto lines
# add vertical line at 20% of characters
ax.axvline(x = int(len(line_count)*.2), color = 'r', linestyle = '--', linewidth = .6)
ax.annotate('20% of Characters', xy = (6.1, 275), fontsize = 10)

# add horizontal line at 80% of all lines
ax2.axhline(y = 80, color = 'r', linestyle = '--', linewidth = .6)
ax.annotate('80% of Lines', xy = (29.6, 210), fontsize = 10)

# add data labels for bar chart
for p in ax.patches:
    # print(p)
    ax.annotate(int(p.get_height()),
                xy = (p.get_x()+0.15, p.get_y()+ p.get_height()+2.5),
                fontsize = 15)

# add chart labels
ax.set_ylabel('Count of Lines', fontsize = 15)
ax2.set_ylabel('Percentage of Total Lines', fontsize = 15)
plt.tight_layout()
plt.show()

```

- 80% of the lines are between roughly 25% of characters, and 75% of the characters 'barely' talk



Workflow Breakdown

Analysis –*cont'd*

Unusual Words

- Processing the lines before identifying *"unusual words"*
- Standardising case size, dumping unnecessary whitespaces and punctuation, splitting into words, getting words into their dictionary forms *(lemmatisation)* and removing "stopwords" since they are not unusual *(such as "I", "for", "can", "be", etc.)*

- Creating *"usual words"* vocabulary; to be able to identify *"unusual words"* by looking at the difference of two.

- Using a range of books written by authors other than Shakespeare –*so we do <u>not</u> compare Shakespearean words against themselves*- and creating a *"usual words"* vocabulary using each word

```
[19]
def clean_text(text):
    """takes in text, removes leading, trailing, multiple whitespaces
    standardises letter case, tokenizes words, removes non alpha-numeric elements and lemmatizes words
    removes stopwords, merges into a string again
    """
    text_ = ' '.join(text.split()) # clean leading, traling, multiple whitespaces
    text_ = text_.lower() # standardise cases
    text_ = [token for token in word_tokenize(text_) if token.isalnum()] # tokenize only alpha-numeric and exclude punctuation & special characters
    text_ = [WordNetLemmatizer().lemmatize(token) for token in text_] # lemmatize each word
    text_ = [token for token in text_ if token not in stopwords.words('english')] # remove stopwords
    text_ = ' '.join(text_)

    return text_
```

```
[20]
df['clean_line'] = df['line'].apply(clean_text)
```

```
[21]
def create_usual_vocabulary():
    """creates a set of vocabulary, combining books in gutenberg books vocabulary
    and nltk words and removing stopwords and punctuation
    """
    nltk_words = nltk.corpus.words.words()
    usual_words_vocab = set(nltk_words)

    for ind in nltk.corpus.gutenberg.fileids(): # loop over author-book name index
        if ind.find('shakespeare') == -1: # if index doesn't contain shakespeare, get words of it
            words = nltk.corpus.gutenberg.words(ind) # assing in list

            words = [word.lower() for word in words if word.isalnum()] # drop punctuation and special characters
            words = [word for word in words if word not in stopwords.words('english')] # drop stopwords

            usual_words_vocab.update(words) # add to usual words vocabulary set
            print(f'Added words from book: {ind}') # some sanity check

    return usual_words_vocab

usual_words_vocab = create_usual_vocabulary()
```

```
Added words from book: austen-emma.txt
Added words from book: austen-persuasion.txt
Added words from book: austen-sense.txt
Added words from book: bible-kjv.txt
Added words from book: blake-poems.txt
Added words from book: bryant-stories.txt
Added words from book: burgess-busterbrown.txt
Added words from book: carroll-alice.txt
Added words from book: chesterton-ball.txt
Added words from book: chesterton-brown.txt
Added words from book: chesterton-thursday.txt
Added words from book: edgeworth-parents.txt
Added words from book: melville-moby_dick.txt
Added words from book: milton-paradise.txt
Added words from book: whitman-leaves.txt
```

Workflow Breakdown

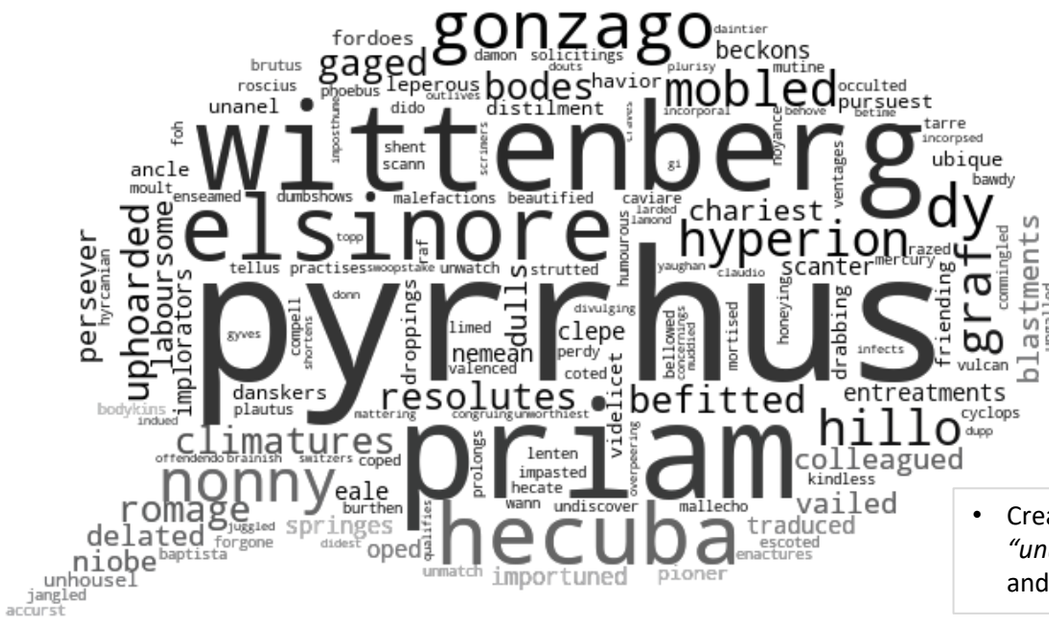
Analysis –*cont'd*

Unusual Words

- Creating *"unusual words"* vocabulary by getting the words that exist among lines of the play but not in *"usual words"* vocabulary previously created
- Checking the usage frequency of each word that exist in *"unusual words"* to determine the font size of those words in the wordcloud.

```python
# create single string from cleaned lines
# remove char names from string, to avoid mentions of characters
# get difference of usual words vocab, to string creating ultimate unusual words used
# create a counter dictionary for how many times each unusual words are used

def unusual_word_counter():
    """ from cleaned lines, removes character names then words that exist also in usual words vocabulary
    starts a dictionary: counter
    loops through clean lines and counts frequency of each word and returns dictionary
    """

    clean_lines = ' '.join(' '.join(df['clean_line']).split())
    char_names_list = ' '.join(df['chars'].str.lower().unique().tolist()).split()
    clean_lines_set = set(clean_lines.split()).difference(set(char_names_list))
    unusual_words = list(clean_lines_set.difference(usual_words_vocab))

    counter = dict()

    for word in clean_lines.split():
        if word in unusual_words:
            if word in counter.keys():
                counter[word] += 1
            else:
                counter[word] = 1

    return counter

counter = unusual_word_counter()
counter
```

```
'donn': 1,
'douts': 1,
'drabbing': 1,
'droppings': 1,
'dulls': 1,
'dumbshows': 1,
'dupp': 1,
'dy': 3,
'eale': 1,
'elsinore': 4,
'enactures': 1,
'enseamed': 1,
```



Unusual Words

- Creating wordcloud, using *"unusual words"* vocabulary and frequencies.

```python
### for running it all at once purposes, commented out mask picture, -speech bubble
###################

mask = np.array(Image.open('/content/bubble3.JPG')) # use default microsoft office speech bubble for wordcloud background
image_colors = ImageColorGenerator(mask) # mask to use coloring from speech bubble

wcloud2 = wordcloud.WordCloud(mask = mask,
                              width = 800, height = 400,
                              mode = "RGBA",
                              background_color = None,
                              ).generate_from_frequencies(counter) # create wordcloud using frequencies of usage
wcloud2.recolor(color_func = image_colors)

plt.figure(figsize=(20,5))
plt.imshow(wcloud2, interpolation = 'bilinear')
plt.axis('off')
plt.tight_layout(pad=0)
plt.show()

wcloud2.to_file('wc.png');
```

Workflow Breakdown

Analysis –*cont'd*

Sentiment Analysis

- Using lexicon based *pretrained models* to identify sentiment of each line. *-since there are no labelled lines with sentiment to train and use a complex, custom classification model*

```python
def act_sentiments(df = df, col = 'clean_line'):
    """takes in dataframe, gets sentiment for each line using textblob and nltk
    assigns sentiment string for line
    outputs 3 dataframes,
    dataframe with sentiments,
    dataframe with count of textblob sentiments for each act,
    dataframe with count of nltk sentiments for each act
    """

    tb = Blobber(analyzer=NaiveBayesAnalyzer())
    sid = SentimentIntensityAnalyzer()

    for ind, val in df.iterrows():

        # textblob sentiment
        blob = tb(val[col])
        textblob_sent = blob.sentiment.classification

        if textblob_sent == 'pos':
            textblob_sent = 'Positive'
        elif textblob_sent == 'neg':
            textblob_sent = 'Negative'
        else:
            textblob_sent = 'Neutral'
        df.loc[ind, 'TextBlob_Sentiments'] = textblob_sent

        # nltk sentiment
        score = sid.polarity_scores(val[col])['compound']

        if score >= 0.05:
            sentiment = 'Positive'
        elif score <= -0.05:
            sentiment = 'Negative'
        else:
            sentiment = 'Neutral'

        df.loc[ind, 'NLTK_Sentiments'] = sentiment


    textblob_sents_by_scene = df.groupby(by = ['act_scene','TextBlob_Sentiments'], as_index = False).size()
    textblob_sents_by_scene['pct_in_act'] = round(textblob_sents_by_scene.groupby(by = 'act_scene', as_index = False)['size'].apply(lambda x: 100*x / x.sum()).reset_index(drop = True))

    nltk_sents_by_scene = df.groupby(by = ['act_scene','NLTK_Sentiments'], as_index = False).size()
    nltk_sents_by_scene['pct_in_act'] = round(nltk_sents_by_scene.groupby(by = 'act_scene', as_index = False)['size'].apply(lambda x: 100*x / x.sum()).reset_index(drop = True))


    return df, textblob_sents_by_scene, nltk_sents_by_scene

df_sentiments, textblob_by_scenes, nltk_by_scenes = act_sentiments(df)
```

- Using 2 of the widely used pretrained models *(TextBlob and NLTK),* to be able to cross-check the integrity of our sentiment labels *-to an extent. -sentiments highlighted in red-*

	chars	line	act_scene	TextBlob_Sentiments	NLTK_Sentiments
28	Marcellus	Peace, break thee off; look, where it comes ag...	Act I, Scene I	Positive	Positive
166	Hamlet	What hour now?	Act I, Scene IV	Negative	Neutral
214	Marcellus	Lord Hamlet,–	Act I, Scene IV	Positive	Neutral
1	Francisco	Nay, answer me: stand, and unfold yourself.	Act I, Scene I	Positive	Neutral
221	Hamlet	O, wonderful!	Act I, Scene IV	Positive	Positive
455	Hamlet	We'll ha't to-morrow night. You could, for a n...	Act II, Scene II	Positive	Neutral
87	Hamlet	I would not hear your enemy say so,\nNor shall...	Act I, Scene II	Positive	Negative
725	Hamlet	Do not believe it.	Act IV, Scene II	Positive	Neutral
446	Lord Polonius	That's good; 'mobled queen' is good.	Act II, Scene II	Positive	Positive
64	Laertes	My dread lord,\nYour leave and favour to retur...	Act I, Scene II	Positive	Positive

Workflow Breakdown

Analysis –*cont'd*

Sentiment Analysis

- Using sentiments for each line in each scene, producing 2 plots from using percentage of lines belonging to each sentiment in each scene.
- Using percentage will normalize the overall sentiment weight in each scene and will make comparison between acts possible.

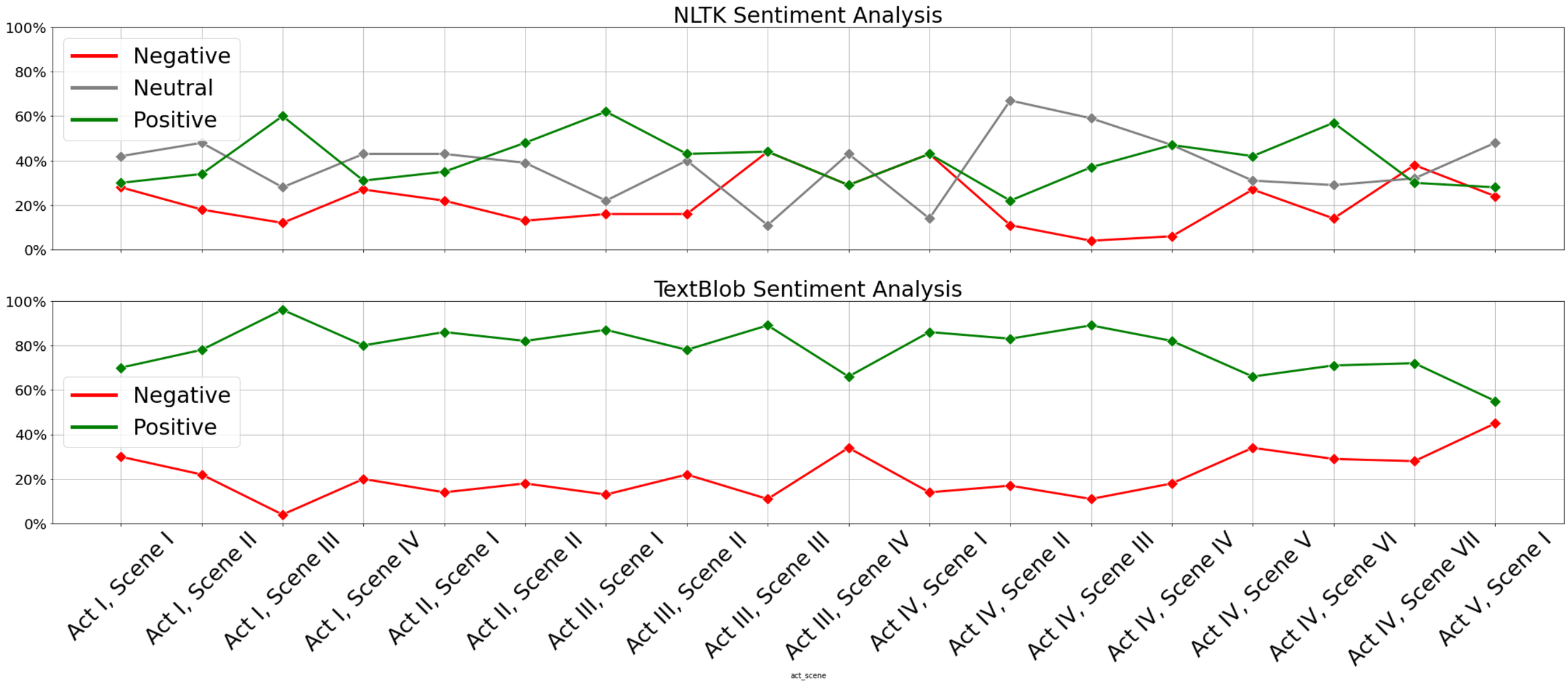
```python
fig, ax = plt.subplots(2, figsize=(31, 14), sharex=True) # create 2 ax plot area

palette ={"Positive": "g", "Neutral": "Grey", "Negative": "r"} # common palette to assign meaningful colours

# nltk plot
g = sns.lineplot(x = 'act', y = 'pct_in_act', data = nltk_by_acts, hue = 'NLTK_Sentiments', marker = 'D', linewidth = 3, markersize = 12, palette = palette, ax = ax[0])

ax[0].set_xticklabels( () ) # remove x ticks from chart on top
ax[0].tick_params(axis = 'y', labelsize = 20) # prettify axis
ax[0].yaxis.set_major_formatter(PercentFormatter(decimals = 0)) # prettify y axis labels

ax[0].set_ylim(0, 100) # standardise y axis with a range of 0 - 100%
ax[0].yaxis.grid(True, which='major') # add gridlines
ax[0].xaxis.grid(True, which='major') # add gridlines

ax[0].set_title('NLTK Sentiment Analysis', {'fontsize': 30}) # add titles
ax[0].set_ylabel(None) # remove y labels to use 1 shared title for both charts

# place legend to avoid overlapping chart
leg = ax[0].legend(loc='upper left',
           fontsize = 30,
           markerfirst = True)

# format legend markers for visibility
for legobj in leg.legendHandles:
    legobj.set_linewidth(5.0)

# textblob plot
g1 = sns.lineplot(x = 'act', y = 'pct_in_act', data = textblob_by_acts, hue = 'TextBlob_Sentiments', marker = 'D', linewidth = 3, markersize = 12, palette = palette, ax = ax[1])

ax[1].tick_params(axis = 'x', labelsize = 30) # prettify axis
ax[1].tick_params(axis = 'y', labelsize = 20) # prettify axis
ax[1].yaxis.set_major_formatter(PercentFormatter(decimals = 0)) # prettify y axis labels


ax[1].set_ylim(0, 100) # standardise y axis with a range of 0 - 100%
ax[1].yaxis.grid(True, which='major') # add gridlines
ax[1].xaxis.grid(True, which='major') # add gridlines

ax[1].set_title('TextBlob Sentiment Analysis', {'fontsize': 30}) # add titles
ax[1].set_ylabel(None)  # remove y labels to use 1 shared title for both charts

# place legend to avoid overlapping chart
leg = ax[1].legend(loc='center left',
           fontsize = 30,
           markerfirst = True)

# format legend markers for visibility
for legobj in leg.legendHandles:
    legobj.set_linewidth(5.0)

# add shared y axis
fig.text(0.001, 0.5, 'Percentage of Lines in Act', va='center', rotation='vertical', fontsize = 30)

# add padding to avoid overlapping
plt.tight_layout(pad=4, w_pad=55, h_pad=3)
plt.show()
```

Workflow Breakdown

Analysis –*cont'd*

Sentiment Analysis

- Both pretrained models indicate that positive and negative sentiments start close, then spreads out through the book and finally finishes close again; indicating climatic moves through the book.
- This shows storytelling style.



Thank You