Observing Repetition in Presidential Speeches Over Time Using the Lempel-Ziv-Welch Algorithm

Abhimanyu Dhir
AP Statistics: Period 8
May 23, 2018

Abstract

This study is meant to uncover how repetition in presidential speeches has changed over time in order to see if speeches have been simplified for the modern American public. In order to measure repetition I employed the Lempel-Ziv-Welch (LZW) compression algorithm, and measured unique word count as well. The data concludes that there is a slightly negative correlation between the two variables, meaning that over time, repetitiveness in speeches has gone down.

Contents

1 Introduction

After the unorthodox election of 2016, it seems as though political speeches have gotten more simplified for the American public. Although repetition of key phrases is a rhetorical strategy, it seems as though modern speeches have just consisted of a repetition of buzz words. When thinking of the United States government at its start over 200 years ago, one might picture the eloquent language of the founding documents and the professionalism of the presidential speeches. This seems in stark contrast to the speeches heard in modern times which often contains more informal language and is meant to be more accessible to the public and to voters. In order to see if presidential speeches have indeed increased in repetition over time I decided to measure the percent compression of all major presidential speeches, starting from George Washington, and also measured how many days after January 1st 1789 the speech was delivered. In order to calculate the percent compression of speeches, I utilized teh Lempel-Ziv-Welch (LZW) compression algorithm, which is a well known, although outdated, compression algorithm which was used to zip text files. I will elaborate more on why specifically I used a compression algorithm and not simply unique word count later in Section 2. In addition to measuring compression, I also measure the categorical variable of political party. I felt as though this was a meaningful categorical variable as it would be able to determine whether or not people of a certain party had a habit of repeating themselves in speeches more than people of another party. When researching about this topic I was not able to find an extensive study on the repetition of American presidents throughout history, but I did find a paper on the lexical density of President Obama's and McCain's speeches during the 2008 election (Savoy). I also was able to find research over the repetition in music versus its popularity, and it was from this article which I got the idea to use the LZW compression algorithm (Morris). The specific population being observed in this study is all major presidential speeches throughout history. I feel as though this study has greater implications, as by observing the speech data, it can be seen whether or not speeches have become more "dumbed down" for the American public and it can also be seen how different parties have different levels of repetition. Thus, this study can reflect on the political state of the United States and show how political speech has changed.

2 Data Collection

2.1 Collecting Speeches

At the start of this project, I planned on using all political speeches in general over the past 50 or so years. However, when attempting to collect data for this, I realized that no real database existed which contained a well representative sample of this population. While exploring other options, I found a database called the Miller Center which was hosted by the University of Virginia. This website contained all the major presidential speeches since George Washington (around 1,000 speeches), and was a natural fit for this study ("Miller Center"). Next in order to extract the speeches from the website I used the python library

Figure 1

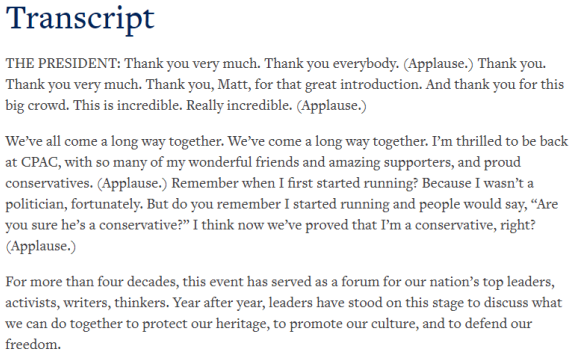


Figure 2

BeautifulSoup to get the HTML from the website and save all the speeches as text files, with other additional information which was needed. As can be seen in figure 1, the main page of the website had links to all the major presidential speeches. Each page which contained a speech had a transcript as depicted in figure 2 and information about the speech as depicted in figure 3.

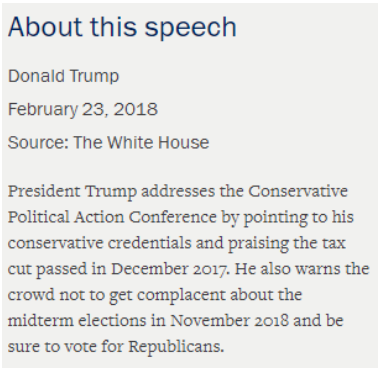


Figure 3

First to get all of the speeches, I first had to generate a list of all the links which lead to speeches. The Python code which I used to do this is contained in the Appendix. A portion of the output of this code can be seen in figure 4. Given all the links to the speeches, the next step was to extract all of the information I needed. The Python code which I used in order to do this is also located in the Appendix. Using the code, I created a text document for each of the speeches on the website as can be seen in figure 5. I titled each of the files with the president's name and a number in ascending order, as to make the titles of speeches by the same president unique. What was stored inside of the files can be seen in figure 6. As can be seen, inside each document was the date it was delivered on the first line, and the rest contained the transcript of the speech.

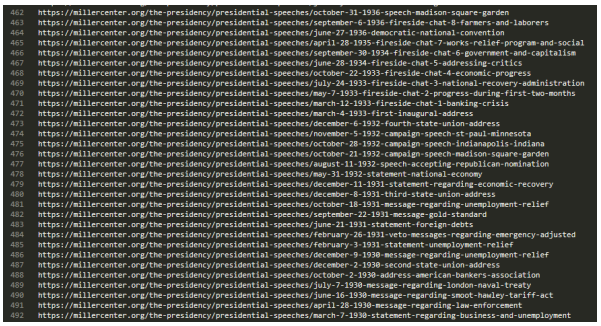


Figure 4

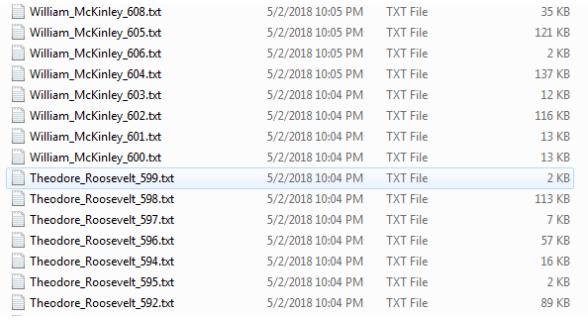


Figure 5

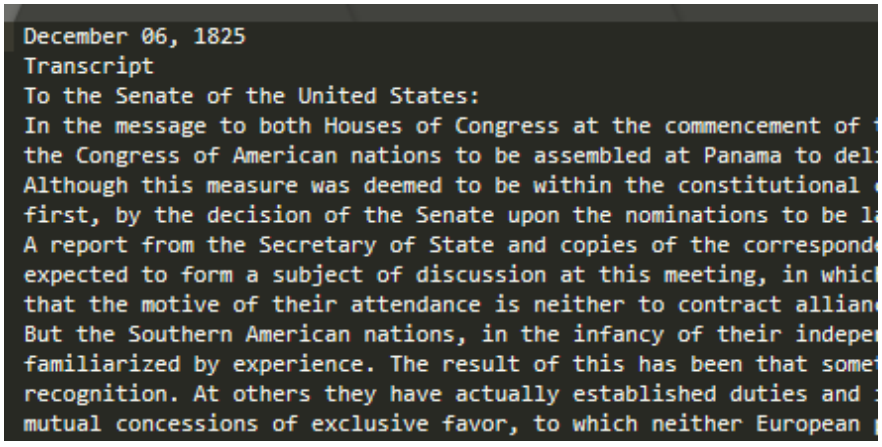


Figure 6

2.2 Running Algorithms on the Speeches

Given all the speeches in the format necessary to perform the analysis I wanted, all that was left to do was write the different algorithms I would use to collect data. This naturally leads to the the question of what algorithm would be the best to measure repetition.

2.2.1 Why not measure unique word count?

One of my first instincts was to simply measure the unique word count percentage of the speeches as the variable to measure repetition. This, however, I realized was flawed. I can demonstrate why through the following example text coming from Donald Trump's "Remarks at the Conservative Political Action Conference" speech delivered on February 23rd, 2018:

> *"Thank you very much. Thank you everybody. Thank you. Thank you very much. Thank you, Matt, for that great introduction. And thank you for this big crowd. This is incredible. Really incredible. We've all come a long way together. We've come a long way together"*

As we can see, this segment seems very repetitive, and the percentage of repeated words is

around 38%. However, when scrambling the words randomly, as shown below, the speech seems less repetitive.

> *"incredible. Thank you very crowd. Matt, for introduction. This way Really this is way incredible. together. very you, all long big We've for thank long together. We've Thank come great you Thank you much. that Thank come everybody. Thank much. you. a And a you"*

This, although nonsensical, has the same unique word count as the unscrambled speech, but seems less repetitive, as it does not contain key phrases such as "Thank you very much" and "We've come a long way together" which are repeated in the unscrambled segment. What this shows is that an algorithm which can differentiate repetition in speeches, even if they use the same words is needed. This lead me to use the LZW compression algorithm which is explained in the next section.

2.2.2 How Lempel-Ziv-Welch works

Given the same example of Donald Trump's speech as shown in the previous section, we can understand how the LZW compression algorithm works. How it works is that the algorithm goes through the text message and adds new characters, words, and phrases to a python dictionary. Then, when going through new words and phrases, if they are already contained inside the dictionary, the redundant text is replaced with a reference or pointer back to the first instance of the text. A very simple example of this would be with the compression of the phrase "I liked dogs. I like cats. I like animals." This would turn into "I liked dogs. *** cat****e animal*" where each of the '*' characters represents a pointer (in reality many of these pointers would be different, but just a '*' is used to represent all of them for simplicity's sake). Moving back to the example used in the previous section:

> *"Thank you very much. Thank you everybody. Thank you. Thank you very much. Thank you, Matt, for that great introduction. And thank you for this big crowd. This is incredible. Really incredible. We've all come a long way together. We've come a long way together"*

This would compress down to:

> *"Thank you very much. *****e**body**********u, Matt*for t*t gre* intr**tion*And**** ***is big c*wd******edible*R*ll****** We'* a**ome* l**wa* toge*******c**a** ****h*"*

This gives a compression percentage of around 25.8%. When running the algorithm on the scrambled words, the compression percentage is only 24.2%. While this isn't a large difference, it is still a significant one as over a larger speech or section of text, the disparity increases. Although I ended up using the built in python library, lzw, I still wrote my own simplistic compression algorithm which I used to prepare these examples. The code for this is found in the Appendix.

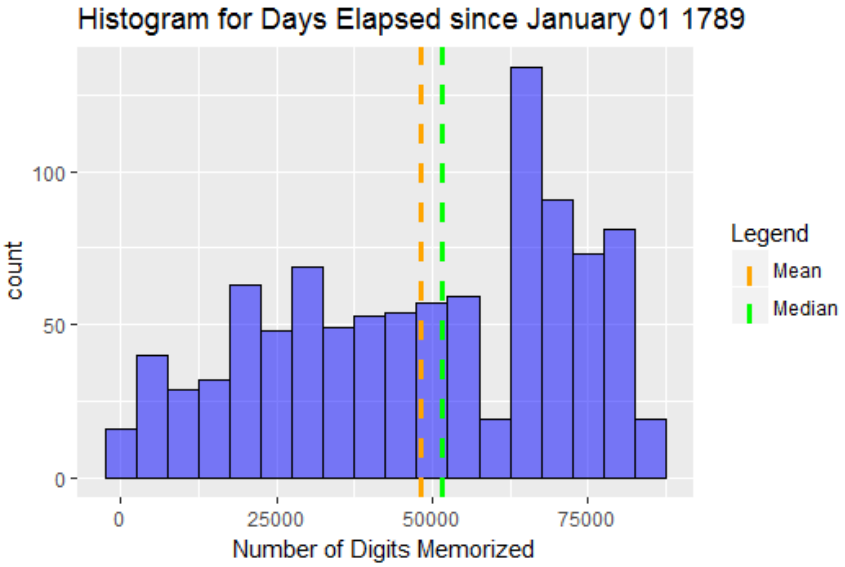
2.2.3 Running LZW in Python

After realizing that my compression algorithm was more simplistic and didn't capture the repetition as well as the built in python library function, I decided to use that as my main source of data. Using the python code as seen in the Appendix, I collected all the information I needed. First, I got the title name of each file, and extracted the President's name from it. Next, using a python dictionary, I was able to get all the presidents' party affiliation as well. Next, using the datetime library, I got the days that have elapsed since January 1st 1789 when the speech was delivered. Next I ran the library compression method, the custom written compression method, and lastly a unique word count percentage. Then I also noted down the length of the speech for good measure to see if there was significant trend in that way, considering that longer speeches would tend to have increased compression as well since it is more likely that a given word would be repeated. After storing all of these values into a python, 2 dimensional array, I output the array as a csv file so that I would be able to analyze it in R. The data collection was not random per se, but the website did seem to contain all the major presidential speeches through time. There is a slight bias in the fact that more modern speeches have been recorded than older ones. The dataset is too large to fit in this report, but a snapshot of it is contained in the Appendix

3 Data Analysis

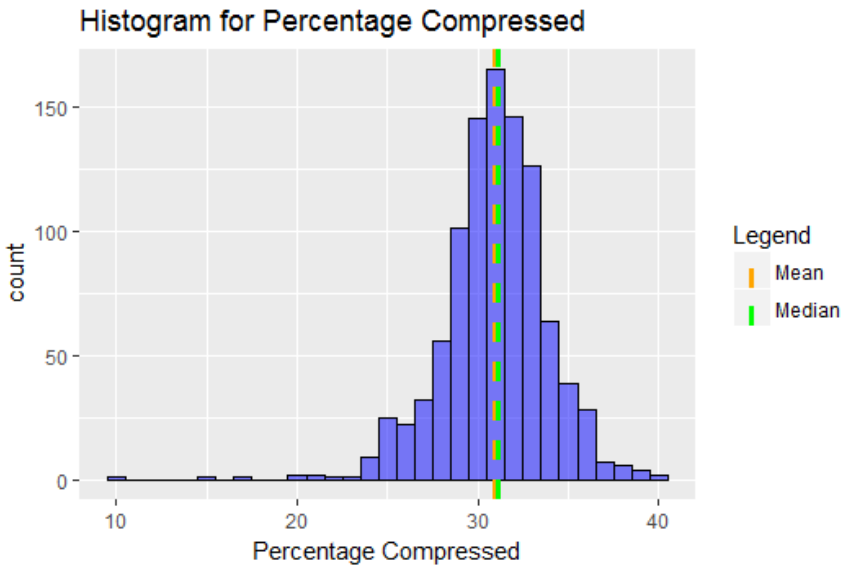
3.1 Summary Statistics

All charts and data analysis was done in RStudio. The code which I used is in the Appendix First I examined the histogram for days elapsed since January 1st 1789.



As we can see from the histogram the shape is skew left, meaning that more speeches have been recorded in more recent years. There do not seem to be any outlier years, meaning that there have been speeches throughout the years since George Washington. The mean of the distribution is 48,217 days after January 1st 1789 which is in 1921. The median of the distribution is 51,462 after January 1st 1789 which is in 1929. Lastly, the spread of the distribution is 83598 days or 229 years, which makes sense as that is the duration of when American presidents have existed.

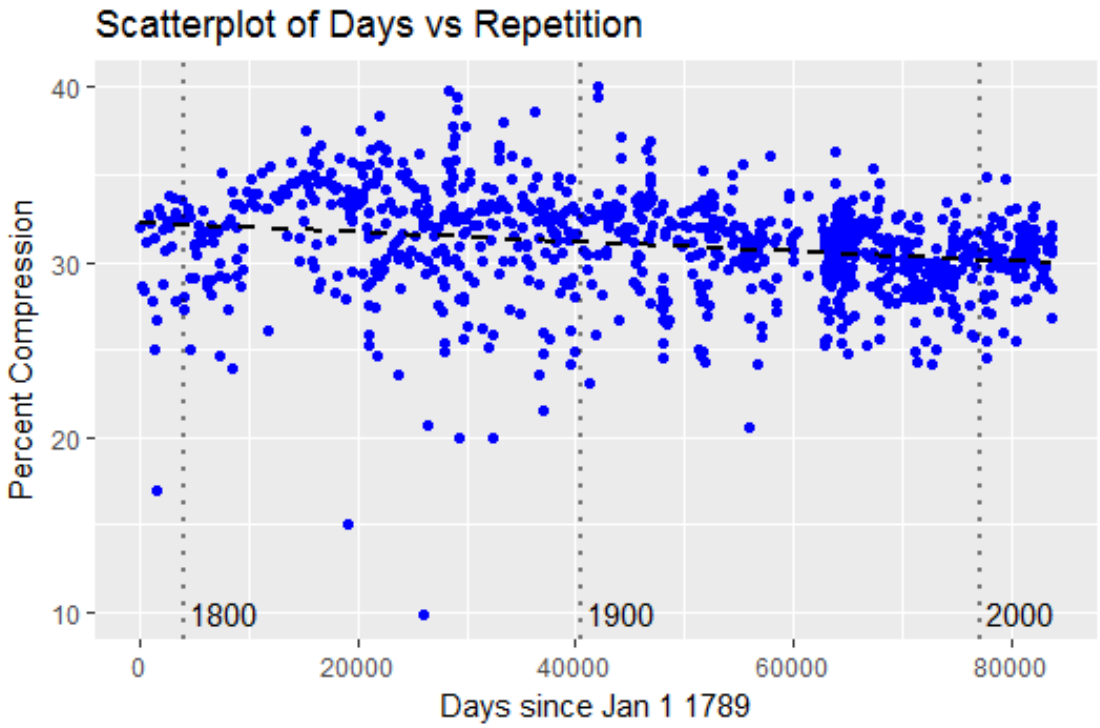
Next I examined the histogram for the percentage that the speeches had been compressed using LZW.



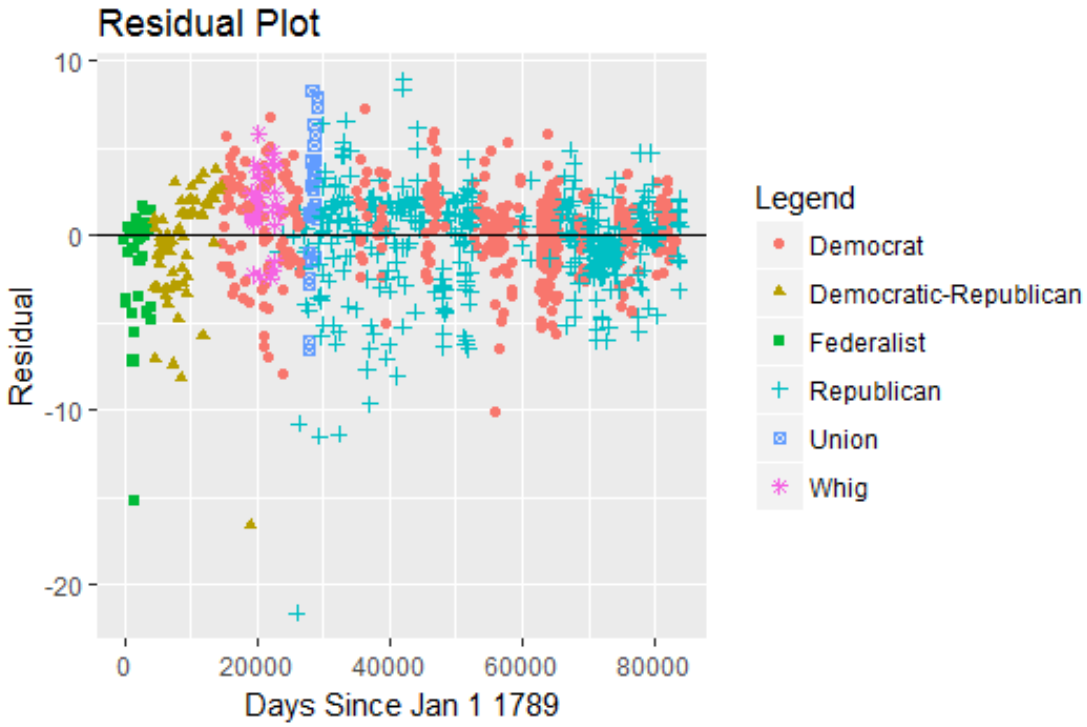
As we can see from the histogram the shape is fairly normal, although there do seem to be values which are far to the left which have been less compressed. There do seem to be some outlier compression values, which resulted from very short speeches delivered and thus getting compressed less. The mean of the distribution is 30.94% compressed and the median of the distribution is 31.1% compressed. The similarity between the two values also support the normality of the plot. Lastly, the spread of the distribution is 30.1%.

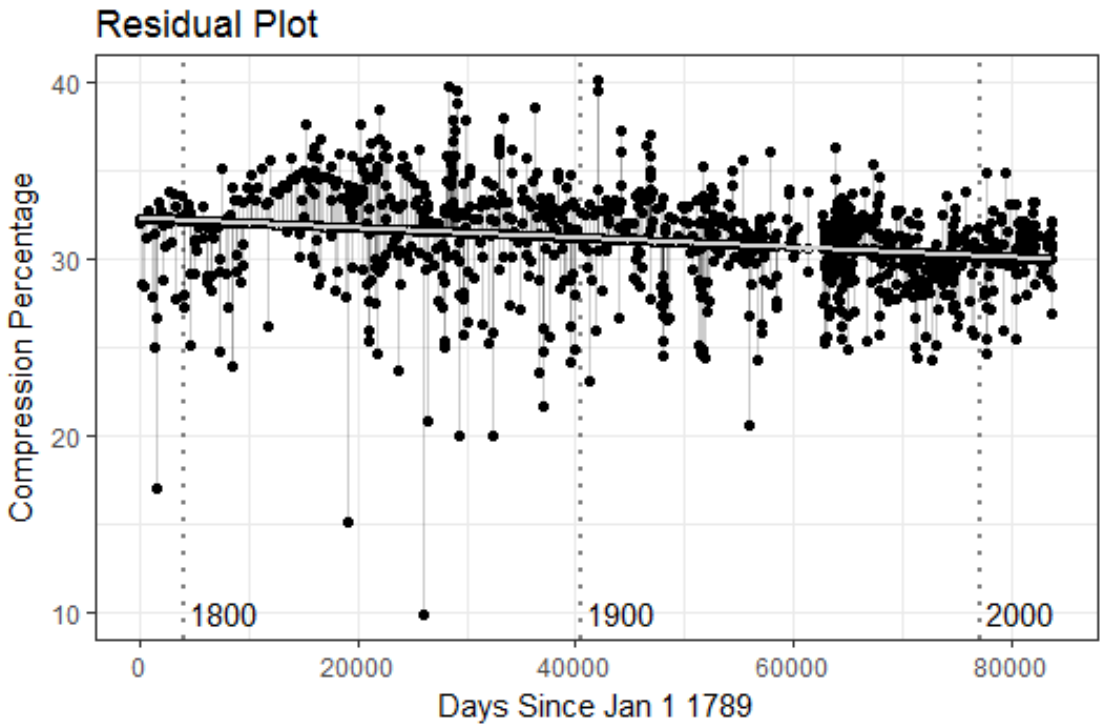
3.2 Scatterplot Analysis

Next, I examined the scatterplot showing the compression percentage of speeches over time.



Just examining it, I could see that there was a lot of variance in the data, but the negative trend seemed apparent. There are a couple of outliers, but due to their location and the quantity of data, I did not think they significantly changed the trend line. Next I plotted residual plots to see if the linear model was a good fit.





As can be seen by the residual plots, there seems to be no clear pattern, thus making the linear model a good fit for the data.

3.3 Hypothesis Test for Linear Regression

In order to conduct a hypothesis test for linear regression on the data, the following conditions must be met:

- **Randomness:** This condition is not fully met as all the speeches were from the same website. However, since the website seems to have a good distribution of all the major presidential speeches throughout time, I decided I could proceed with caution.
- **Independence:** This condition is met as the sample size of around 1000 speeches is less than 10% of all speeches delivered by American presidents.
- **Normality:** There are greater than 30 speeches, and the data seems fairly normal in both the y and x axis.
- **Linearity:** As can be seen by the residual plots and as discussed above, a linear model is appropriate.
- **Equal Variance:** As can be seen on the residual plot, there seems to be fairly even variance for all values of x.

Next the Hypotheses for the test are as follows:

$H_0 : \beta = 0$ (There is no useful linear relationship between compression percentage and days

elapsed)
$H_a : \beta \neq 0$ (There is a useful linear relationship between compression percentage and days elapsed)

In order to perform the Linear Regression hypothesis test, I used RStudio to analyze the trend. The resulting output is what is shown below:

```
Coefficients:
                          Estimate Std. Error t value Pr(>|t|)
(Intercept)              3.226e+01  2.082e-01  154.97  < 2e-16 ***
Days.since.Jan.01.1789 -2.745e-05  3.883e-06   -7.07 2.93e-12 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 2.858 on 984 degrees of freedom
Multiple R-squared:  0.04835,   Adjusted R-squared:  0.04738
F-statistic: 49.99 on 1 and 984 DF,  p-value: 2.927e-12
```

I set my α value to equal .01. As can be seen from the image, the test statistic calculated was -7.07 and the degrees of freedom were 984. The resulting p-value was 2.927×10^{-12}. This p-value is much lower than the α value of .01, and this would allow me to reject the null hypothesis and accept the alternative hypothesis. This concludes that there is a significant linear relationship between compression percentage and day elapsed.

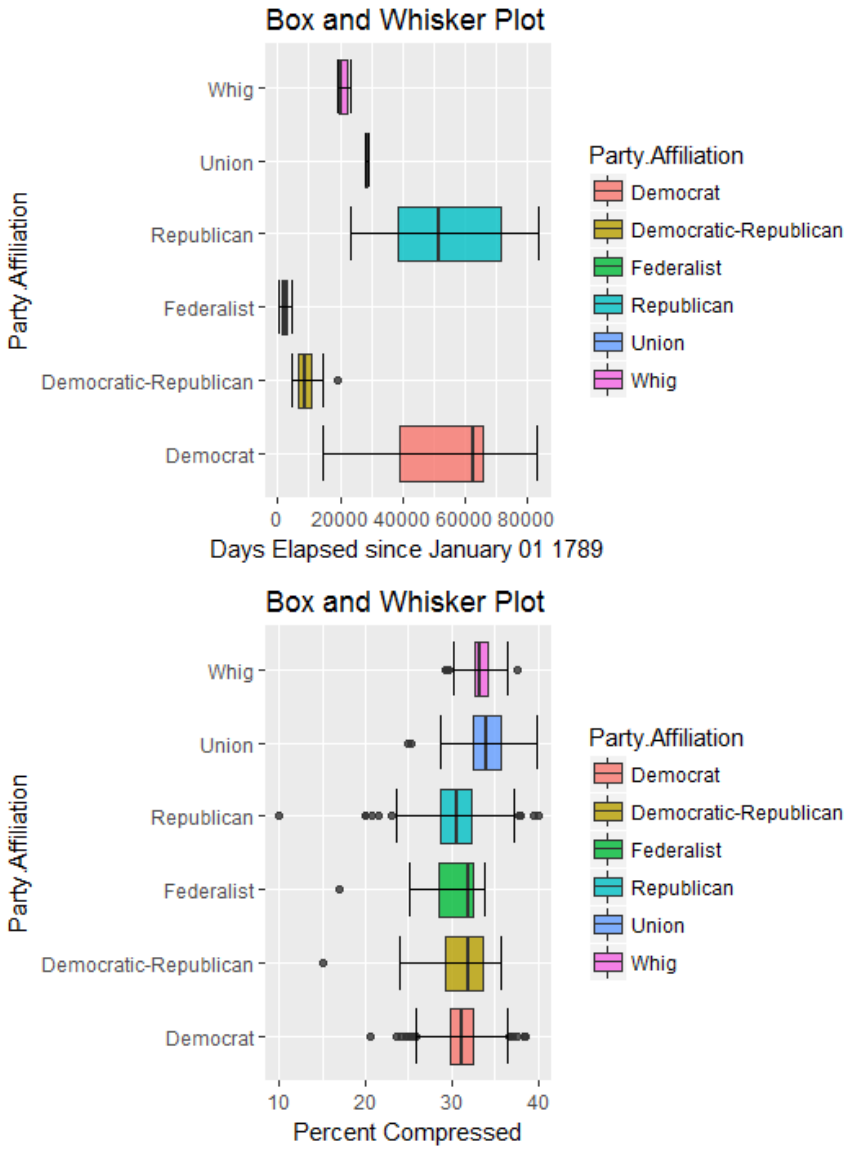
The given equation for the trend line is: Percent Compression = 32.26 + $(-2.745 \times 10^{-5}) \times$ Number of Days since Jan 1st 1789. The intercept of 32.26 means that if a President were to deliver a speech on January 1st 1789, the amount the trend line predicts it could compressed by would be 32.26%. The slope of -2.745×10^{-5} suggests that for each day, the compression of a presidential speech decreases by 2.745×10^{-5}%. Another thing to note about the hypothesis test, is that the R^2 value is very low at .04835. This however could be explained by that large variance and few outliers. In addition since the slope of the trend line is very low, the R^2 value could be brought down by that as well. Next, I created a confidence interval based off of the given slope.

```
> confint(fitM, 'Days.since.Jan.01.1789', level=0.95)
                              2.5 %        97.5 %
Days.since.Jan.01.1789 -3.507343e-05 -1.983391e-05
>
```

The resulting 95% confidence interval confidence interval suggest that the true slope of the data lies between -3.507343×10^{-5} and -1.983391×10^{-5}.

3.4 Seeing how Political Party Ties In

The categorical variable that I introduced to the speech was political party, and in order to see if there was any correlation between them, I first decided to check the box and whisker plots when distributing over political party. This can be seen below:



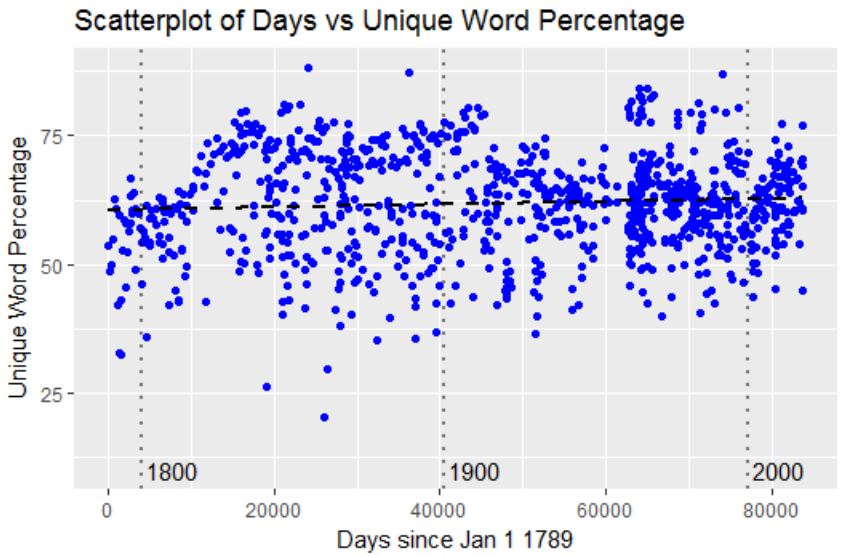
No significant insight was given from the box and whisker plot for days elapsed as different political parties came to power at different periods in history, and for different durations. The plot just confirms this. The box and whisker plot for percentage compressed however seems to suggest that Union presidents are the most repetitive and that Republicans are the least. However, there do seem to be outliers, as Republicans have the largest spread, and thus have very repetitive speeches as well.

Next I looked at the various trendlines of all the political parties over time separately. This required looking at 6 different plots, which are all in the Appendix as well as the different summary statistics of each. It was notable that the parties other than Democrats and Republicans had much less data and thus had different overall trends. Only the Democrats, the Republicans, and the Whigs had negative trends in compression over time, whereas the other parties had positive ones. The most negative trend was the Whigs with a slope of -6.117×10^{-5} and the most positive slope was the Union party with a slope of .005.

When comparing Democrats and Republicans, Democrats had a lower slope of -3.99×10^{-5} compared to $2 - 2.38 \times 10^{-5}$ meaning that they were becoming less repetitive faster. The intercept on Democrats is higher, at 33.3% compared to 31.6%, meaning that Democrats started more repetitive.

3.5 Looking Back at Unique Word Percentage

When starting the data collection process, I decided not to use unique word percentage as my main source of comparison to time. However, in order to preserve thoroughness, I collected unique word percentage regardless and was able to plot what is seen below:



As can be seen above, the unique word count percentage has gone up over time, adding more evidence to the fact that speeches have decreased in repetition over time. A disclaimer to note, is that this method also contains a lot of variance.

4 Conclusion

Based off of the data collection and analysis, I can conclude that out of major presidential speeches, repetition seems to have decreased over time. This means that older presidents tended to repeat ideas and phrases more than modern presidents, which came as a surprise to me. After collecting all the speeches from the UVA Miller center, and running the LZW algorithm on them, I created a linear model of percent compression versus days elapsed since January 1st 1789. This linear model was determined to be a good fit based off of the residual plot and thus was used to run a hypothesis test. The test with an α level of .01 and a p-value of around 3×10^{-12} that there is a significant linear correlation between percent compression and days elapsed. Next, I examined how political parties as a categorical variable tied in to the study. The data was limited in the fact that only a large number of speeches

were found for Democratic and Republican presidents. Nevertheless, the data showed that Republicans seemed to be the least repetitive overall and that Union members tended to be the most repetitive. After analysis of the trends of the various political parties, it was shown that Democrats had a lower slope than Republicans, meaning that they were decreasing in repetitiveness faster. Ultimately this data set is quite large and thus probably fairly accurately captures the true nature of American presidential speeches over time. However, this study could be improved by getting more speeches, creating a more randomized sampling method, and perhaps trying to run various other zipping or compression algorithms. With the data and analysis as is though, I can conclude with some confidence that presidential speeches have in fact gotten less repetitive over time as an overall trend.

5 Appendix

5.1 Python Code to Generate Document of Speech Links

```
# Gets HTML File from Main Page
speeches_html_file = open('Presidential Speeches _ Miller Center.html')
soup = BeautifulSoup(speeches_html_file,'html.parser')
# Creates a New Document for Writing the Links
links_file = open("links.txt",'w')
# For Loop which looks for the specific html tags for the intended links
for i in range(0,len(soup.find_all('span',class_='field-content'))):
    if(i%2==0):
        # Writes the each link found to the document "links.txt"
        links_file.write((list(soup.find_all('span',class_='field-content')
        [i].children)[0].get('href'))+"\n")
```

5.2 Python Code to Generate Text Documents of Speeches

```
# Opens the "links.txt" document
file =  open('links.txt')
# Creates an array with all the links, and goes through them one by one
lines = file.read().splitlines()
for i in range(len(lines)):
    # Gets the HTML from the link
    page = requests.get(lines[i])
    soup = BeautifulSoup(page.content, 'html.parser')
    # Creates a text file with the name of the President and a Number
    file = open("Speeches_Master/" + (soup.find_all('p',class_='president-
    name')[0].get_text()).replace(" ","_") + "_" + str(i+1) + ".txt","w")
    # Adds the date the speech was delivered to the file
    file.write(soup.find_all('p', class_='episode-date')[0].get_text())
```

```
    # Older Speeches were located in a different format
    # Created three cases for how to write the transcript into the document
    # Speeches were made uniform by removing line breaks between paragraphs
    try:
        file.write((soup.find_all('div',class_='transcript-inner')[0]
        .get_text()).encode('utf-8'))
    except IndexError:
        file.write("\n" + list(soup.find_all('div',class_='view-transcript')
        [0].children)[0].get_text().encode('utf-8') + "\n")
        for j in range(1,len(list(soup.find_all('div',class_='view-
        transcript')[0].children))):
            try:
                file.write(list(soup.find_all('div',class_='view-transcript')
                [0].children)[j].get_text().encode('utf-8')
                .replace("\n\n","\n"))
            except AttributeError:
                pass
```

5.3 Python Code for the Custom LZW Algorithm

```
def compress1(uncompressed):
    # creates python dictionary of length 256 to contain the references
    dict_size = 256
    dictionary = dict((chr(i), chr(i)) for i in xrange(dict_size))
    w = ""
    result = []
    # goes through each character and group of characters in message
    for c in uncompressed:
        wc = w + c
        # checks if there is repetition
        if wc in dictionary:
            # adds pointer
            w = wc
        else:
            # if new, adds reference to dictionary
            result.append(dictionary[w])
            dictionary[wc] = dict_size
            dict_size += 1
            w = c

    # outputs the message
    if w:
        result.append(dictionary[w])
    return result
```

5.4 Python Code for Collecting the Compression Data

```
import lzw
import pandas as pd
import glob, os
from datetime import datetime
import csv

def compress1(uncompressed):
    # creates python dictionary of length 256 to contain the references
    dict_size = 256
    dictionary = dict((chr(i), chr(i)) for i in xrange(dict_size))
    w = ""
    result = []
    # goes through each character and group of characters in message
    for c in uncompressed:
        wc = w + c
        # checks if there is repetition
        if wc in dictionary:
            # adds pointer
            w = wc
        else:
            # if new, adds reference to dictionary
            result.append(dictionary[w])
            dictionary[wc] = dict_size
            dict_size += 1
            w = c

    # outputs the message
    if w:
        result.append(dictionary[w])
    return result
# Gives the compression percentage given the custom function
def custom_compressor(data):
    compressed_list = compress1(data)
    return str(float(len(data)-len(list(compress1(data))))/
    float(len(data))*100)
# Gives the compression percentage given the library compression method
def library_compressor(data):
    return str(float(len(data)-len(list(lzw.compress(lzw.ByteEncoder(12).
    encodetobytes(data)))))/float(len(data))*100)
# Returns the unique word percentage of the speech
def lexicalDensity(message):
  list_words = message.split()
```

```
  words = float(len(list_words))
  uWords = float(len(set(list_words)))
  return (words-(uWords))/(words)*100
# Dictionary mapping president name to political party
party_dict = {'Abraham_Lincoln' : 'Republican', 'Andrew_Jackson' : 'Democrat',
'Andrew_Johnson' : 'Union', 'Barack_Obama' : 'Democrat',
'Benjamin_Harrison' : 'Republican', 'Bill_Clinton' : 'Democrat',
'Calvin_Coolidge' : 'Republican', 'Chester_A._Arthur' : 'Republican',
'Donald_Trump' : 'Republican', 'Dwight_D._Eisenhower' : 'Republican',
'Franklin_D._Roosevelt' : 'Democrat', 'Franklin_Pierce' : 'Democrat',
'George_H._W._Bush' : 'Republican', 'George_W._Bush' : 'Republican',
'George_Washington' : 'Federalist', 'Gerald_Ford' : 'Republican',
'Grover_Cleveland' : 'Democrat', 'Harry_S._Truman' : 'Democrat',
'Herbert_Hoover' : 'Republican', 'James_A._Garfield' : 'Republican',
'James_Buchanan' : 'Democrat', 'James_K._Polk' : 'Democrat',
'James_Madison' : 'Democratic-Republican',
'James_Monroe' : 'Democratic-Republican',
'Jimmy_Carter' : 'Democrat', 'John_Adams' : 'Federalist',
'John_F._Kennedy' : 'Democrat', 'John_Quincy_Adams' : 'Democratic-Republican',
'John_Tyler' : 'Whig', 'Lyndon_B._Johnson' : 'Democrat',
'Martin_Van_Buren' : 'Democrat', 'Millard_Fillmore' : 'Whig',
'Richard_Nixon' : 'Republican', 'Ronald_Reagan' : 'Republican',
'Rutherford_B._Hayes' : 'Republican', 'Theodore_Roosevelt' : 'Republican',
'Thomas_Jefferson' : 'Democratic-Republican', 'Ulysses_S._Grant' : 'Republican',
'Warren_G._Harding' : 'Republican', 'William_Harrison' : 'Whig',
'William_McKinley' : 'Republican', 'William_Taft' : 'Republican',
'Woodrow_Wilson' : 'Democrat', 'Zachary_Taylor' : 'Whig'}

# Initializes different arrays to hold data
name_list = []
party_list = []
date_list = []
custom_compress_list = []
library_compress_list = []
lexical_list = []
length_list = []

# Opens all the text files in the folder
os.chdir("Speeches_Master/")
for file in glob.glob("*.txt"):
    # Gets the name of each file
    split_file = (file).split("_")
    name_list.append(file.replace((split_file)[len(split_file)-1],'')[:-1])
    # Gets the political party from the dictionary
    party_list.append(party_dict[file.replace((split_file)[len(split_file)-1]
```

```
    ,'')[:-1]])
    # Reads the file
    f = open(file,'r')
    # Gets the difference in days from the first line of the file
    date_str = datetime.strptime(f.readline().replace('\n',''), '%B %d, %Y')
    - datetime.strptime('January 01 1789', '%B %d %Y')
    date_list.append(int(date_str.days))
    data = f.read().strip().replace('\n', ' ')
    # Runs the compression algorithms
    custom_compress_list.append(custom_compressor(data))
    library_compress_list.append(library_compressor(data))
    lexical_list.append(lexicalDensity(data))
    # Gets the character length of the speech
    length_list.append(len(data))

# Puts all the data sets into a multidimensional array
full_data = []
full_data.append(name_list)
full_data.append(party_list)
full_data.append(date_list)
full_data.append(custom_compress_list)
full_data.append(library_compress_list)
full_data.append(lexical_list)
full_data.append(length_list)

# Outputs the array as a csv, titled "output.csv"
df = pd.DataFrame(full_data).T
df.columns = ['Name', 'Party.Affiliation', 'Days.since.Jan.01.1789',
'Custom.Compression.Percentage', 'Library.Compression.Percentage',
'Unique.Word.Count.Percentage', 'Character.Length']
df.to_csv("output.csv",)
```

5.5 Picture of part of the Dataset

	A	B	C	D	E	F	G
1	Name	Party.Affiliation	Days.since.Jan.01.1789	Custom.Compression.Percentage	Library.Compression.Percentage	Unique.Word.Count.Percentage	Character.Length
2	Abraham_Lincoln	Republican	27820	57.33909947	27.21953701	45.32374101	3931
3	Abraham_Lincoln	Republican	27732	70.58242007	30.82137612	66.7633015	35404
4	Abraham_Lincoln	Republican	27368	71.22613446	31.96495213	67.83828383	36978
5	Abraham_Lincoln	Republican	27349	50.91525424	27.52542373	41.06463878	1475
6	Abraham_Lincoln	Republican	27264	67.94857887	30.55435018	77.59612474	18436
7	Abraham_Lincoln	Republican	27027	58.97793264	29.57026713	57.79036827	4305
8	Abraham_Lincoln	Republican	26996	72.26356434	31.50937719	70.17103217	49855
9	Abraham_Lincoln	Republican	26633	71.80775269	32.06859075	68.52173913	41405
10	Abraham_Lincoln	Republican	26481	71.60643455	33.04869987	70.30469128	36304
11	Abraham_Lincoln	Republican	26359	68.71165644	32.82921957	64.77811889	21027
12	Abraham_Lincoln	Republican	26338	43.24324324	20.76167076	29.60526316	814
13	Abraham_Lincoln	Republican	25988	34.35897436	9.914529915	20.2020202	585
14	Abraham_Lincoln	Republican	25367	67.60869565	31.51630435	62.70967742	18400
15	Abraham_Lincoln	Republican	24028	78.34151606	31.6732823	88.20530044	187183
16	Abraham_Lincoln	Republican	23196	73.6241416	30.63803721	81.15370162	62034
17	Andrew_Jackson	Democrat	17593	73.38186029	33.27267464	73.19751128	48358
18	Andrew_Jackson	Democrat	17520	65.61455626	35.17452655	59.63353692	10772
19	Andrew_Jackson	Democrat	17504	75.463153	34.45529924	77.01543739	73086
20	Andrew_Jackson	Democrat	17140	74.96850063	34.58397499	75.79511834	64287
21	Andrew_Jackson	Democrat	16769	75.77748358	34.58646617	77.44052502	80066
22	Andrew_Jackson	Democrat	16545	55.8575101	28.86522218	52.73522976	2723
23	Andrew_Jackson	Democrat	16539	76.62898225	36.71023092	79.90161073	74832

5.6 Rstudio Code for Data Analysis

```
library(ggplot2)
library(plyr)

#---------------Histogram for Days since January 1 1789
ggplot(data=output, aes(x=output$Days.since.Jan.01.1789),
color="legend") +
  geom_histogram(binwidth=5000, fill=I("blue"),col=I("black"),
  alpha=I(.5)) +
  ggtitle("Histogram␣for␣Days␣Elapsed␣since␣January␣01␣1789") +
  labs(x="Days␣Elapsed␣since␣January␣01␣1789") +
  geom_vline(aes(xintercept = mean(output$Days.since.Jan.01.1789),
  color="Mean"), size=1.2, linetype="dashed") +
  geom_vline(aes(xintercept = median(output$Days.since.Jan.01.1789),
  color="Median"), size=1.2, linetype="dashed") +
  scale_color_manual(name="Legend", values = c(Mean = "orange",
  Median = "green"))

#---------------Histogram for Percentage Compressed
ggplot(data=output, aes(x=output$Library.Compression.Percentage),
color="legend") +
  geom_histogram(binwidth=1, fill=I("blue"),col=I("black"),alpha=I(.5)) +
  ggtitle("Histogram␣for␣Percentage␣Compressed") +
  labs(x="Percentage␣Compressed") +
  geom_vline(aes(xintercept = mean(output$Library.Compression.Percentage),
  color="Mean"), size=1.2, linetype="dashed") +
  geom_vline(aes(xintercept = median(output$Library.Compression.
  Percentage), color="Median"), size=1.2, linetype="dashed") +
  scale_color_manual(name="Legend", values = c(Mean = "orange",
```

```
  Median = "green"))

#---------------Box and Whisker Plot for Days since Jan 1 1789
ggplot(output,aes(x=Party.Affiliation,y=Days.since.Jan.01.1789,
fill=Party.Affiliation))+
  geom_boxplot(alpha=.8)+
  coord_flip()+
  ggtitle("Box and Whisker Plot")+
  labs(y="Days Elapsed since January 01 1789")+
  stat_boxplot(geom ='errorbar')

#---------------Box and Whisker Plot for Percent Compression
ggplot(output,aes(x=Party.Affiliation,y=Library.Compression.Percentage,
fill=Party.Affiliation))+
  geom_boxplot(alpha=.8)+
  coord_flip()+
  ggtitle("Box and Whisker Plot")+
  labs(y="Percent Compressed")+
  stat_boxplot(geom ='errorbar')

#---------------Scatterplot of Days vs Percent Compression
df <- data.frame(x = output$Days.since.Jan.01.1789, y = output$Library.
Compression.Percentage)
p <- ggplot(data = df, aes(x = x, y = y)) +
  geom_point(color="blue") +
  geom_smooth(method = "lm", se=FALSE, color="black", linetype="dashed",
  formula = y ~ x) +
  geom_vline(aes(xintercept = 4017), size=.9, linetype="dotted",
  alpha=.5) +
  geom_vline(aes(xintercept = 40541), size=.9, linetype="dotted",
  alpha=.5) +
  geom_vline(aes(xintercept = 77065), size=.9, linetype="dotted",
  alpha=.5) +
  ggtitle("Scatterplot of Days vs Repetition") +
  labs(x="Days since Jan 1 1789",y="Percent Compression")
p
p <- p + annotate("text",x=c(4017,40541,77065),y=10,label=c("1800",
"1900","2000"),hjust=-0.1)
p

#---------------Scatterplot of Days vs Percent Compression with Parties
ggplot(output,aes(x=output$Days.since.Jan.01.1789, y=output$Library.
Compression.Percentage, shape=output$Party.Affiliation,
color=output$Party.Affiliation)) +
  geom_point() +
  geom_vline(aes(xintercept = 4017), size=.9, linetype="dotted",
  alpha=.5) +
```

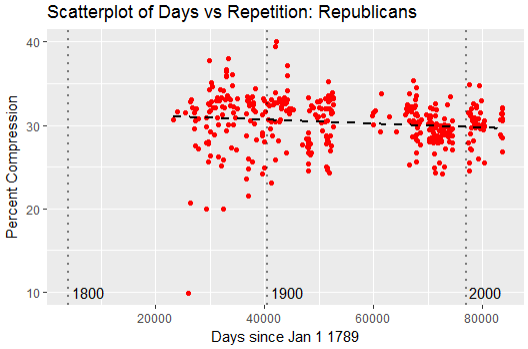
```
  geom_vline(aes(xintercept = 40541), size=.9, linetype="dotted",
  alpha=.5) +
  geom_vline(aes(xintercept = 77065), size=.9, linetype="dotted",
  alpha=.5) +
  annotate("text",x=c(4017,40541,77065),y=10,label=c("1800","1900",
  "2000"),hjust=-0.1) +
  ggtitle("Scatterplot␣of␣Days␣vs␣Repetition") +
  labs(x="Days␣since␣Jan␣1␣1789",y="Percent␣Compression", shape="Legend",
  color="Legend")

#---------------Residual Plot on top of Scatterplot
e <- output
fit <- lm(Library.Compression.Percentage~Days.since.Jan.01.1789, data=e)
e$predicted <- predict(fit)
e$residuals <- residuals(fit)
ggplot(e, aes(x = Days.since.Jan.01.1789, y = Library.Compression.
Percentage)) +
  geom_point() +
  geom_vline(aes(xintercept = 4017), size=.9, linetype="dotted",
  alpha=.5) +
  geom_vline(aes(xintercept = 40541), size=.9, linetype="dotted",
  alpha=.5) +
  geom_vline(aes(xintercept = 77065), size=.9, linetype="dotted",
  alpha=.5) +
  annotate("text",x=c(4017,40541,77065),y=10,label=c("1800","1900",
  "2000"),hjust=-0.1) +
  geom_point(aes(y = predicted), shape = 1) +
  geom_segment(aes(xend = Days.since.Jan.01.1789, yend = predicted),
  alpha=.2) +
  geom_smooth(method = "lm", se = FALSE, color = "lightgrey") +
  theme_bw() +
  ggtitle("Residual␣Plot") +
  geom_text(x = 50, y = 3, label = lm_eqn(df), parse = TRUE) +
  labs(x="Days␣Since␣Jan␣1␣1789", y="Compression␣Percentage")

#---------------Residual Plot
ggplot(e, aes(x=output$Days.since.Jan.01.1789, y=e$residuals,
shape=output$Party.Affiliation, color=output$Party.
Affiliation)) +
  geom_point() +
  ggtitle("Residual␣Plot") +
  labs(x="Days␣Since␣Jan␣1␣1789", y="Residual", shape="Legend",
  color="Legend") +
  geom_hline(yintercept=0)

#---------------Linear Regression Hypothesis Test
fitM <- lm(Library.Compression.Percentage~Days.since.Jan.01.1789,
```

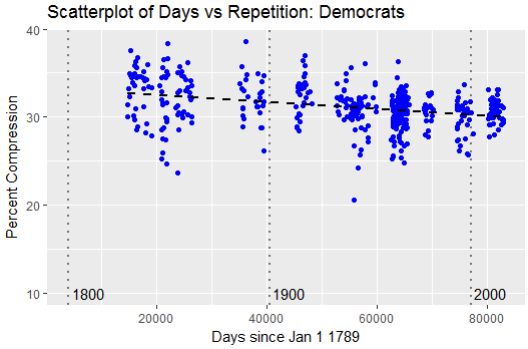
```
data=output)
summary(fitM)
confint(fitM, 'Days.since.Jan.01.1789', level=0.95)

#---------------Scatterplot of Days v Unique Word Percentage
ggplot(output,aes(x=output$Days.since.Jan.01.1789, y=output$Unique.
Word.Count.Percentage)) +
  geom_point(color="blue") +
  geom_vline(aes(xintercept = 4017), size=.9, linetype="dotted",
  alpha=.5) +
  geom_vline(aes(xintercept = 40541), size=.9, linetype="dotted",
  alpha=.5) +
  geom_vline(aes(xintercept = 77065), size=.9, linetype="dotted",
  alpha=.5) +
  annotate("text",x=c(4017,40541,77065),y=10,label=c("1800","1900",
  "2000"),hjust=-0.1) +
  ggtitle("Scatterplot␣of␣Days␣vs␣Unique␣Word␣Percentage") +
  geom_smooth(method = "lm", se=FALSE, color="black", linetype="dashed",
  formula = y ~ x) +
  labs(x="Days␣since␣Jan␣1␣1789",y="Unique␣Word␣Percentage")

#---------------Scatter Plots broken down by Party
Democrat <- output[ which(output$Party.Affiliation=='Democrat'),]
DR <- output[ which(output$Party.Affiliation=='Democratic-Republican'),]
Federalist <- output[ which(output$Party.Affiliation=='Federalist'),]
Republican <- output[ which(output$Party.Affiliation=='Republican'),]
Union <- output[ which(output$Party.Affiliation=='Union'),]
Whig <- output[ which(output$Party.Affiliation=='Whig'),]

fitD <- lm(Library.Compression.Percentage~Days.since.Jan.01.1789,
data=Democrat)
fitDR <- lm(Library.Compression.Percentage~Days.since.Jan.01.1789,
data=DR)
fitF <- lm(Library.Compression.Percentage~Days.since.Jan.01.1789,
data=Federalist)
fitR <- lm(Library.Compression.Percentage~Days.since.Jan.01.1789,
data=Republican)
fitU <- lm(Library.Compression.Percentage~Days.since.Jan.01.1789,
data=Union)
fitW <- lm(Library.Compression.Percentage~Days.since.Jan.01.1789,
data=Whig)

summary(fitD)
summary(fitDR)
summary(fitF)
summary(fitR)
summary(fitU)
```

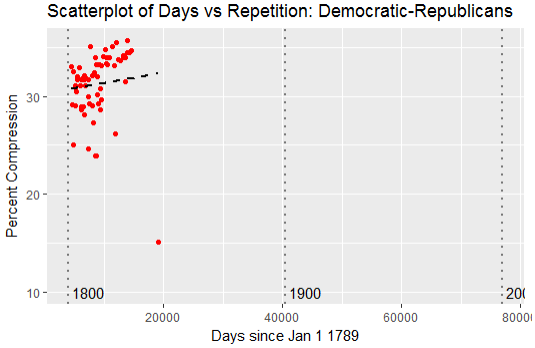
```
summary(fitW)

ggplot(Democrat,aes(x=Days.since.Jan.01.1789, y=Library.
Compression.Percentage)) +
  geom_point(color="blue") +
  geom_vline(aes(xintercept = 4017), size=.9, linetype="dotted",
  alpha=.5) +
  geom_vline(aes(xintercept = 40541), size=.9, linetype="dotted",
  alpha=.5) +
  geom_vline(aes(xintercept = 77065), size=.9, linetype="dotted",
  alpha=.5) +
  geom_smooth(method = "lm", se=FALSE, color="black", linetype="dashed",
  formula = y ~ x) +
  annotate("text",x=c(4017,40541,77065),y=10,label=c("1800","1900",
  "2000"),hjust=-0.1) +
  ggtitle("Scatterplot␣of␣Days␣vs␣Repetition:␣Democrats") +
  labs(x="Days␣since␣Jan␣1␣1789",y="Percent␣Compression")
ggplot(DR,aes(x=Days.since.Jan.01.1789, y=Library.
Compression.Percentage)) +
  geom_point(color="red") +
  geom_vline(aes(xintercept = 4017), size=.9, linetype="dotted",
  alpha=.5) +
  geom_vline(aes(xintercept = 40541), size=.9, linetype="dotted",
  alpha=.5) +
  geom_vline(aes(xintercept = 77065), size=.9, linetype="dotted",
  alpha=.5) +
  geom_smooth(method = "lm", se=FALSE, color="black", linetype="dashed",
  formula = y ~ x) +
  annotate("text",x=c(4017,40541,77065),y=10,label=c("1800","1900",
  "2000"),hjust=-0.1) +
  ggtitle("Scatterplot␣of␣Days␣vs␣Repetition:␣Democratic-Republicans") +
  labs(x="Days␣since␣Jan␣1␣1789",y="Percent␣Compression")
ggplot(Federalist,aes(x=Days.since.Jan.01.1789, y=Library.
Compression.Percentage)) +
  geom_point(color="gray") +
  geom_vline(aes(xintercept = 4017), size=.9, linetype="dotted",
  alpha=.5) +
  geom_vline(aes(xintercept = 40541), size=.9, linetype="dotted",
  alpha=.5) +
  geom_vline(aes(xintercept = 77065), size=.9, linetype="dotted",
  alpha=.5) +
  geom_smooth(method = "lm", se=FALSE, color="black", linetype="dashed",
  formula = y ~ x) +
  annotate("text",x=c(4017,40541,77065),y=10,label=c("1800","1900",
  "2000"),hjust=-0.1) +
  ggtitle("Scatterplot␣of␣Days␣vs␣Repetition:␣Federalists") +
  labs(x="Days␣since␣Jan␣1␣1789",y="Percent␣Compression")
```

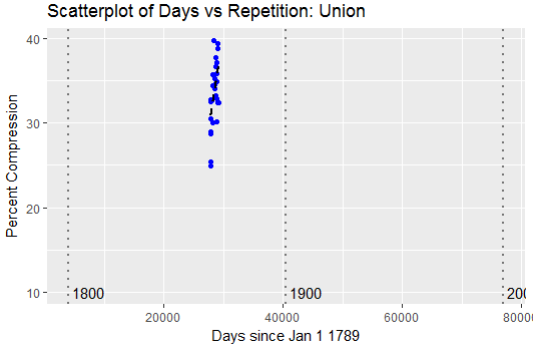
```
ggplot(Republican,aes(x=Days.since.Jan.01.1789, y=Library.
Compression.Percentage)) +
  geom_point(color="red") +
  geom_vline(aes(xintercept = 4017), size=.9, linetype="dotted",
  alpha=.5) +
  geom_vline(aes(xintercept = 40541), size=.9, linetype="dotted",
  alpha=.5) +
  geom_vline(aes(xintercept = 77065), size=.9, linetype="dotted",
  alpha=.5) +
  geom_smooth(method = "lm", se=FALSE, color="black", linetype="dashed",
  formula = y ~ x) +
  annotate("text",x=c(4017,40541,77065),y=10,label=c("1800","1900",
  "2000"),hjust=-0.1) +
  ggtitle("Scatterplot␣of␣Days␣vs␣Repetition:␣Republicans") +
  labs(x="Days␣since␣Jan␣1␣1789",y="Percent␣Compression")
ggplot(Union,aes(x=Days.since.Jan.01.1789, y=Library.
Compression.Percentage)) +
  geom_point(color="blue") +
  geom_vline(aes(xintercept = 4017), size=.9, linetype="dotted",
  alpha=.5) +
  geom_vline(aes(xintercept = 40541), size=.9, linetype="dotted",
  alpha=.5) +
  geom_vline(aes(xintercept = 77065), size=.9, linetype="dotted",
  alpha=.5) +
  geom_smooth(method = "lm", se=FALSE, color="black", linetype="dashed",
  formula = y ~ x) +
  annotate("text",x=c(4017,40541,77065),y=10,label=c("1800","1900",
  "2000"),hjust=-0.1) +
  ggtitle("Scatterplot␣of␣Days␣vs␣Repetition:␣Union") +
  labs(x="Days␣since␣Jan␣1␣1789",y="Percent␣Compression")
ggplot(Whig,aes(x=Days.since.Jan.01.1789, y=Library.
Compression.Percentage)) +
  geom_point(color="blue") +
  geom_vline(aes(xintercept = 4017), size=.9, linetype="dotted",
  alpha=.5) +
  geom_vline(aes(xintercept = 40541), size=.9, linetype="dotted",
  alpha=.5) +
  geom_vline(aes(xintercept = 77065), size=.9, linetype="dotted",
  alpha=.5) +
  geom_smooth(method = "lm", se=FALSE, color="black", linetype="dashed",
  formula = y ~ x) +
  annotate("text",x=c(4017,40541,77065),y=10,label=c("1800","1900",
  "2000"),hjust=-0.1) +
  ggtitle("Scatterplot␣of␣Days␣vs␣Repetition:␣Whigs") +
  labs(x="Days␣since␣Jan␣1␣1789",y="Percent␣Compression")
```

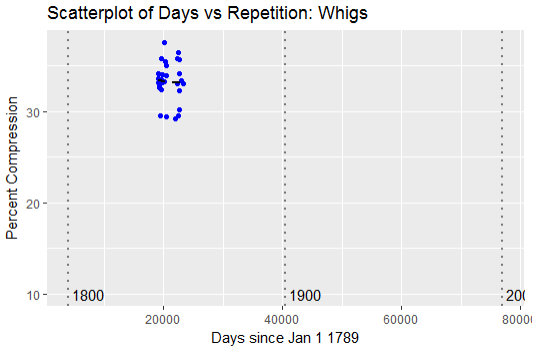
5.7 Images of Various scatter plots and Summaries, Broken Down by Political Party

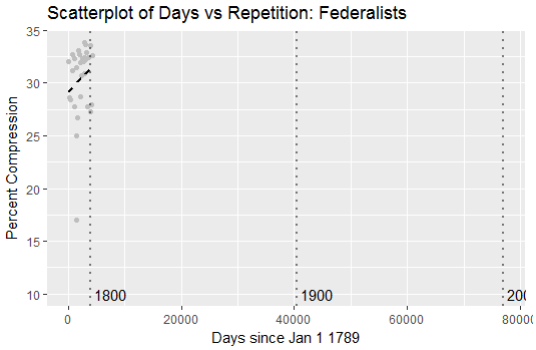













```
> summary(fitR)

Call:
lm(formula = Library.Compression.Percentage ~ Days.since.Jan.01.1789,
    data = Republican)

Residuals:
     Min       1Q   Median       3Q      Max
-21.0801  -1.4461   0.3298   1.8984   9.4241

Coefficients:
                          Estimate Std. Error t value Pr(>|t|)
(Intercept)             3.161e+01  5.039e-01  62.740  < 2e-16 ***
Days.since.Jan.01.1789 -2.375e-05  8.675e-06  -2.737  0.00649 **
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 3.038 on 381 degrees of freedom
Multiple R-squared:  0.01928,    Adjusted R-squared:  0.01671
F-statistic: 7.492 on 1 and 381 DF,  p-value: 0.006488
```

```
> summary(fitD)

Call:
lm(formula = Library.Compression.Percentage ~ Days.since.Jan.01.1789,
    data = Democrat)

Residuals:
     Min       1Q   Median       3Q      Max
-10.4971  -1.0850   0.2425   1.4028   6.7201

Coefficients:
                          Estimate Std. Error t value Pr(>|t|)
(Intercept)             3.330e+01  3.100e-01 107.430  < 2e-16 ***
Days.since.Jan.01.1789 -3.989e-05  5.354e-06  -7.449 4.92e-13 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 2.302 on 445 degrees of freedom
Multiple R-squared:  0.1109,    Adjusted R-squared:  0.1089
F-statistic: 55.49 on 1 and 445 DF,  p-value: 4.92e-13
```

```
> summary(fitDR)

Call:
lm(formula = Library.Compression.Percentage ~ Days.since.Jan.01.1789,
    data = DR)

Residuals:
     Min       1Q   Median       3Q      Max
-17.2162  -1.6587   0.9258   2.1991   4.0287

Coefficients:
                          Estimate Std. Error t value Pr(>|t|)
(Intercept)             3.027e+01  1.323e+00  22.881   <2e-16 ***
Days.since.Jan.01.1789 1.074e-04  1.404e-04   0.765    0.447
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 3.488 on 63 degrees of freedom
Multiple R-squared:  0.0092,    Adjusted R-squared:  -0.006527
F-statistic: 0.585 on 1 and 63 DF,  p-value: 0.4472
```

```
> summary(fitW)

Call:
lm(formula = Library.Compression.Percentage ~ Days.since.Jan.01.1789,
    data = Whig)

Residuals:
    Min      1Q  Median      3Q     Max
-3.9590 -0.7675 -0.0538  0.9636  4.2474

Coefficients:
                          Estimate Std. Error t value Pr(>|t|)
(Intercept)             3.452e+01  5.593e+00   6.171 1.16e-06 ***
Days.since.Jan.01.1789 -6.117e-05  2.690e-04  -0.227    0.822
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 2.146 on 28 degrees of freedom
Multiple R-squared:  0.001844,  Adjusted R-squared:  -0.0338
F-statistic: 0.05173 on 1 and 28 DF,  p-value: 0.8217
```

```
> summary(fitU)

Call:
lm(formula = Library.Compression.Percentage ~ Days.since.Jan.01.1789,
    data = Union)

Residuals:
    Min      1Q  Median      3Q     Max
-5.9445 -1.8792  0.4911  1.9764  6.6624

Coefficients:
                          Estimate Std. Error t value Pr(>|t|)
(Intercept)            -97.268062  38.009615  -2.559  0.01598 *
Days.since.Jan.01.1789   0.004598   0.001336   3.442  0.00177 **
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 3.098 on 29 degrees of freedom
Multiple R-squared:  0.2901,    Adjusted R-squared:  0.2656
F-statistic: 11.85 on 1 and 29 DF,  p-value: 0.001773
```

```
> summary(fitF)

Call:
lm(formula = Library.Compression.Percentage ~ Days.since.Jan.01.1789,
    data = Federalist)

Residuals:
    Min      1Q  Median      3Q     Max
-13.008  -1.435   1.404   2.093   3.175

Coefficients:
                          Estimate Std. Error t value Pr(>|t|)
(Intercept)             2.911e+01  1.351e+00  21.552   <2e-16 ***
Days.since.Jan.01.1789 5.725e-04  5.318e-04   1.076    0.291
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 3.474 on 28 degrees of freedom
Multiple R-squared:  0.03974,   Adjusted R-squared:  0.005447
F-statistic: 1.159 on 1 and 28 DF,  p-value: 0.2909
```

Works Cited

“Miller Center.” *University of Virginia* (2018). Web.

Morris, Colin. “Are Pop Lyrics Getting More Repetitive?” *pudding.cool* (2017). Web.

Savoy, Jacques. “Lexical Analysis of US Political Speeches.” *Journal of Quantitative Linguistics* 21 (2010). Web.