Text Statistical Analysis

In [4]:
```
import pandas as pd
import os
from nltk.tokenize import word_tokenize
from nltk.tokenize import sent_tokenize
from collections import Counter
from nltk.corpus import stopwords
import  matplotlib.pyplot as plt
```

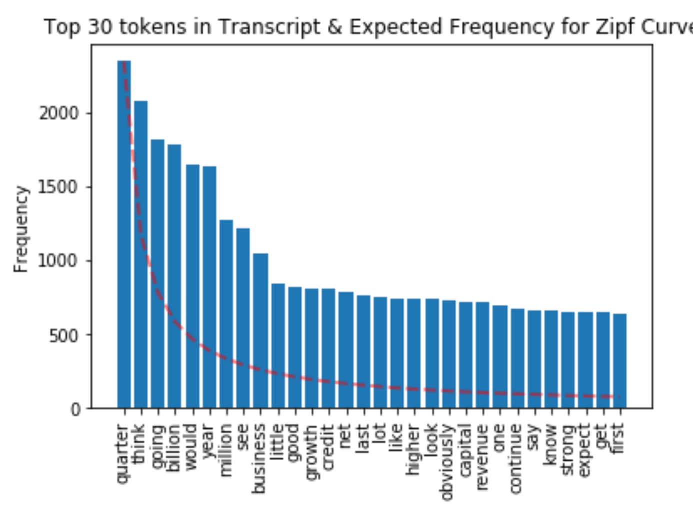
In [2]:
```
os.chdir('/Users/yang-/PycharmProjects/hw/Fintech/pj')
transcript = pd.read_csv('transcripts.csv', index_col=0)
transcript = transcript.iloc[:, 1]
```

In [5]:
```
# sentence tokenize
transcript = transcript.apply(lambda x: sent_tokenize(x.lower()))
# eliminate 'title:'
transcript = transcript.apply(lambda x: ["".join(sentence.split(sep=':', maxsplit=1)[1:]) if ':'in sentence else sente
nce for sentence in x ])
```

In [6]:
```
#frequency analysis
transcript = transcript.apply(lambda x : "".join(x))
str_transcript = "".join(list(transcript))
tokens = [w for w in word_tokenize(str_transcript)if w.isalpha()]
no_stops = [t for t in tokens if t not in stopwords.words('english')]
count = Counter(no_stops).most_common()
```

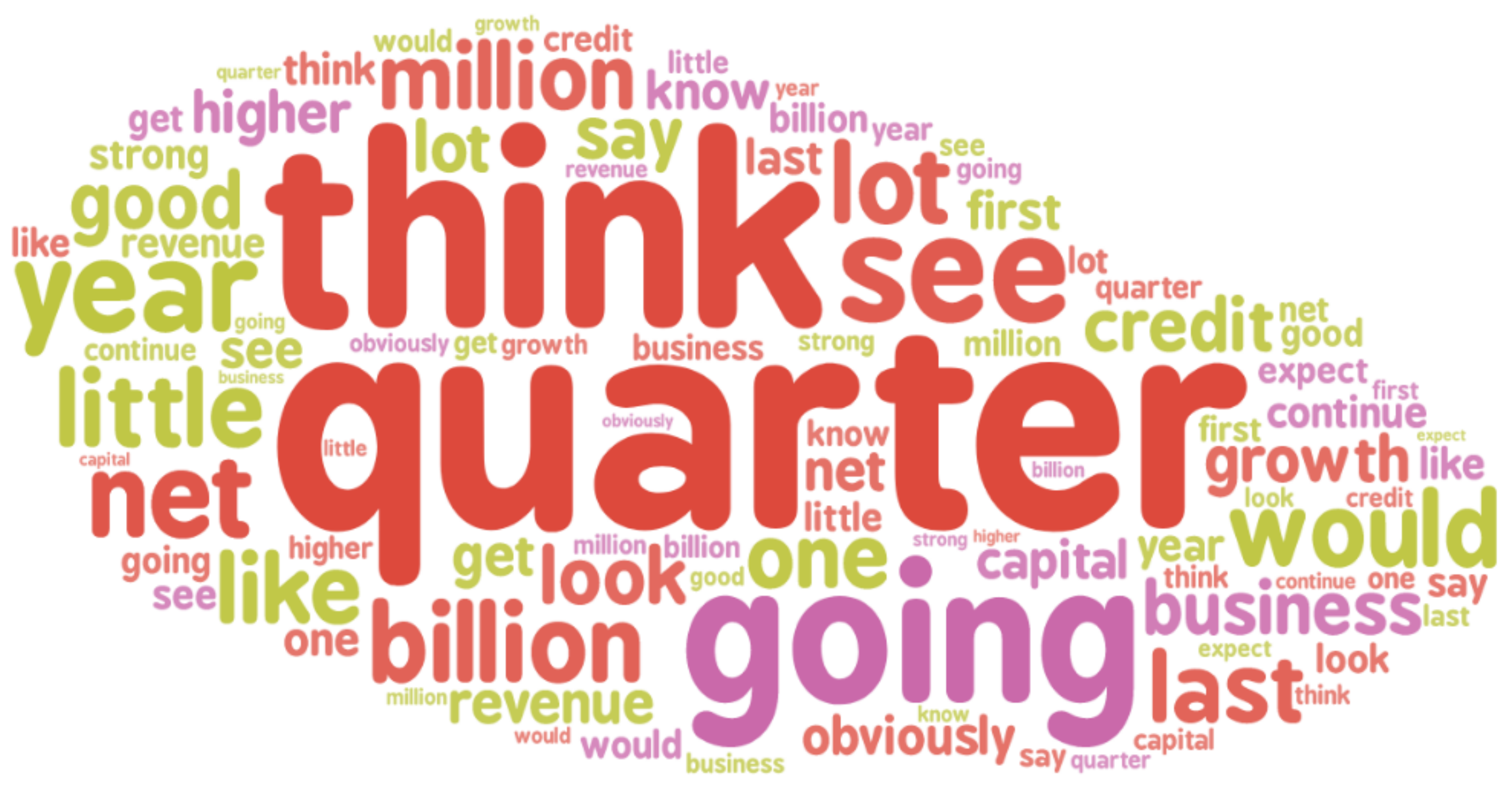
In [11]:
```
#bar plot & zipf plot
rank = pd.DataFrame(map(lambda x: x[0], count), columns=['word'])
rank['count'] = pd.Series(map(lambda x: x[1], count))
rank['expected_zipf'] = [ rank['count'][0]/(i+1) for i in range(rank.shape[0])]
rank_30 = rank.iloc[0:30]
plt.bar(rank_30['word'], rank_30['count'])
plt.plot(rank_30['word'], rank_30['expected_zipf'], color='r', linestyle='--', linewidth=2,alpha=0.5)
plt.xticks(rotation=90)
plt.ylabel('Frequency')
plt.title('Top 30 tokens in Transcript & Expected Frequency for Zipf Curve')
```

Out[11]:
```
Text(0.5, 1.0, 'Top 30 tokens in Transcript & Expected Frequency for Zipf Curve')
```



The actual observations in most cases does not strictly follow Zipf's distribution, but rather follow a trend of "near-Zipfian" distribution.Even though we can see the plot follows the trend of Zipf's Law, but it looks like it has more area above the expected Zipf curve in higher ranked words.

Word Cloud for top 30 tokens in full rank:

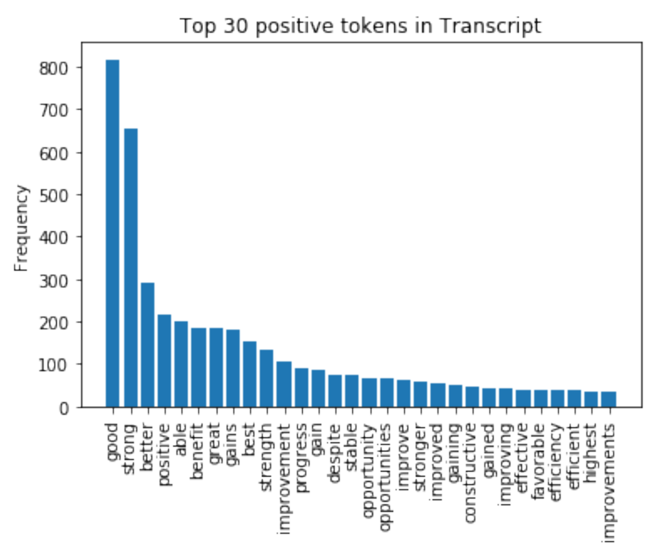


Analysis based on financial dictionary

In [8]:
```
positive_dict = pd.read_csv('positive.csv')
positive_dict = positive_dict.iloc[:,0].apply(lambda x : x.lower())
negative_dict = pd.read_csv('negative.csv')
negative_dict = negative_dict.iloc[:,0].apply(lambda x : x.lower())
positive_dict = positive_dict.rename('word')
negative_dict = negative_dict.rename('word')
rank_positive = pd.merge(rank,positive_dict, how = 'inner')
rank_negative = pd.merge(rank,negative_dict, how = 'inner')
```

In [9]:
```
rank_30 = rank_positive.iloc[0:30]
plt.bar(rank_30['word'], rank_30['count'])
plt.xticks(rotation=90)
plt.ylabel('Frequency')
plt.title('Top 30 positive tokens in Transcript')
```

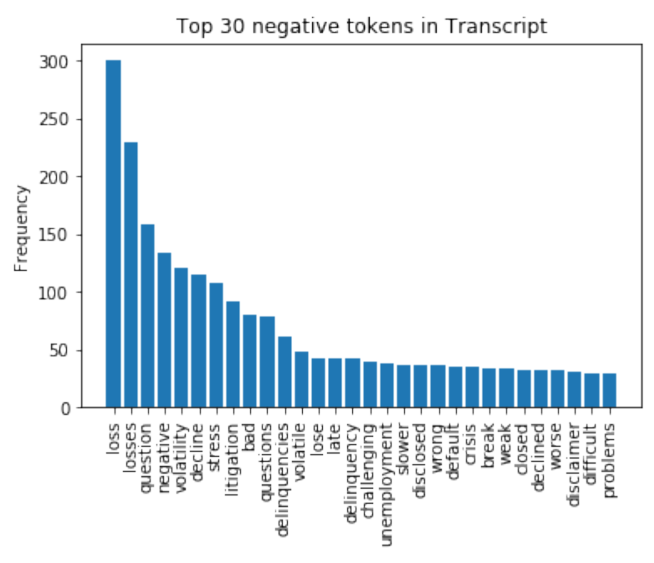
Out[9]:
```
Text(0.5, 1.0, 'Top 30 positive tokens in Transcript')
```



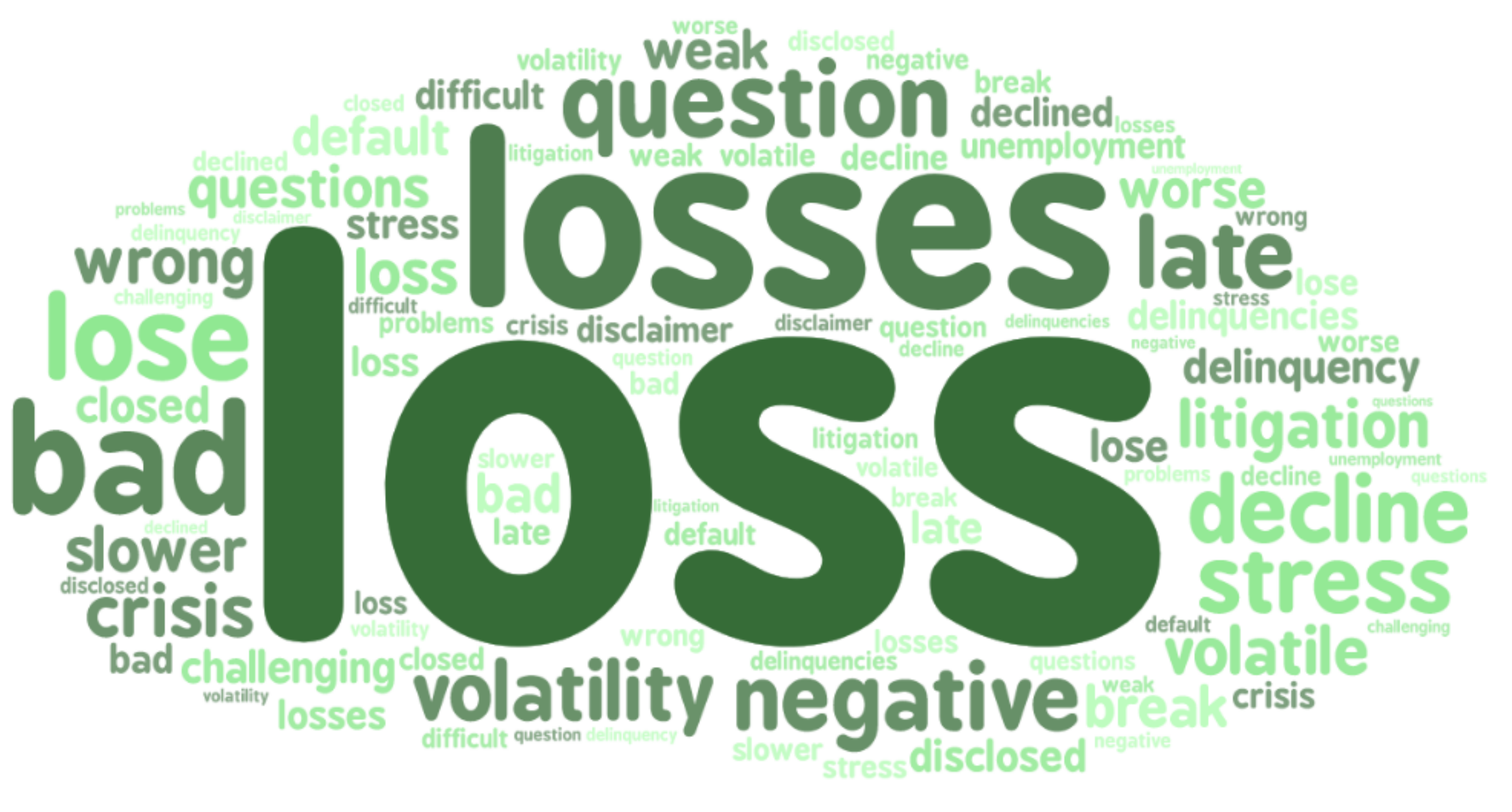
Word Cloud for top 30 positive tokens:

In [10]:
```
rank_30 = rank_negative.iloc[0:30]
plt.bar(rank_30['word'], rank_30['count'])
plt.xticks(rotation=90)
plt.ylabel('Frequency')
plt.title('Top 30 negative tokens in Transcript')
```

Out[10]:
```
Text(0.5, 1.0, 'Top 30 negative tokens in Transcript')
```



Word Cloud for top 30 negative tokens:



Word Cloud for Positive and Negative Words:

In [ ]:

```
from sklearn.linear_model import LinearRegression
from datetime import datetime
import pandas as pd
import numpy as np
import glob
import sys
import os
import re


def parse_args():
    data_path = sys.argv
    return data_path


def get_time(entries):
    ec_date = []
    quarter_year = []

    for file in entries:
        basename = os.path.basename(file)
        quarter_f = basename[0:2]
        year_f = basename[3:7]

        quarter_year.append('{}_{}'.format(quarter_f, year_f))

        with open(file, 'r') as call:
            raw_txt = call.read()
            ec_date.extend(re.findall(r"[\d]{1,2} [ADFJMNOS]\w* [\d]
{4}", raw_txt[:120]))

    ec_date = list(map(lambda x: datetime.strptime(x, '%d %B %Y'),
ec_date))

    return ec_date, quarter_year


def get_trend(df_p, date, days):
    trend = []

    for i in date:
        if np.sign(days) < 0:

            price_range = df_p.iloc[df_p.index.get_loc(i) +
days:df_p.index.get_loc(i), ]
        else:
            price_range =
df_p.iloc[df_p.index.get_loc(i):df_p.index.get_loc(i) + days, ]

        reg = LinearRegression()
        x = np.array(range(1, abs(days) + 1)).reshape(-1, 1)
        y = price_range.values.reshape(-1, 1)

        reg.fit(x, y)
```

```
        trend.append(reg.coef_.item())

    return trend


def main():
    try:
        arg = parse_args()
        earning_dir = arg[1]
        price_dir = arg[2]
        eps_dir = arg[3]
        output_dir = arg[4]
    except:
        print('Usage: python spread.py <earning call path> <price
path> <eps_path> <output path>')
        sys.exit()

    # get a list of all the files in a directory
    entries = []

    for f in glob.glob('{}/*'.format(earning_dir)):
        entries.append(f)

    ec_date, quarter_year = get_time(entries)

    df_eps = pd.read_excel(eps_dir)
    df_eps['EPS_Spread'] = df_eps['Reported_EPS'] -
df_eps['Consensus_Estimate']
    df_eps['Quarter'] = df_eps['Quarter'].str.replace(' ', '_')

    df_price = pd.read_csv(price_dir, index_col='date',
parse_dates=True)['close']
    trend_before = get_trend(df_price, ec_date, -5)
    trend_after = get_trend(df_price, ec_date, 5)

    df_trend = pd.DataFrame({'Quarter_Year': quarter_year,
'Trend_Before': trend_before, 'Trend_After': trend_after})
    df_trend['Trend_Spread'] = df_trend['Trend_After'] -
df_trend['Trend_Before']

    df_spread = pd.merge(df_eps[['Quarter', 'EPS_Spread']],
df_trend[['Quarter_Year', 'Trend_Spread']],
                         left_on="Quarter", right_on='Quarter_Year',
how='left')
    df_spread = df_spread[['Quarter_Year', 'EPS_Spread',
'Trend_Spread']].dropna(axis=0)
    df_spread.to_csv('{}/spread.csv'.format(output_dir))


if __name__ == '__main__':
    main()
```

```
import pandas as pd
import numpy as np
import sys

from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor,
DecisionTreeClassifier
from sklearn.linear_model import LinearRegression,
LogisticRegression

np.set_printoptions(linewidth=100, precision=3)
pd.set_option('display.max_columns', 100)
pd.set_option('display.precision', 3)


def get_data(data, col_name, classification=False):
    if classification == True:
        data_subset = data[[col_name, 'EPS_Spread',
'Trend_Spread_Class']]
    else:
        data_subset = data[[col_name, 'EPS_Spread', 'Trend_Spread']]

    return data_subset


def split_data(data, test_size=0.2):
    x = np.array(data.drop(data.columns[-1], axis=1))
    y = np.array(data[data.columns[-1]])

    x_train, x_test, y_train, y_test = train_test_split(x, y,
test_size=.2)
    return x_train, x_test, y_train, y_test


def main():
    try:
        data_path = sys.argv[1]
        output_dir = sys.argv[2]
    except:
        print('Please provide appropriate data path & output
directory')
        sys.exit()

    data = pd.read_csv(data_path, index_col=0)

    df_prediction = pd.DataFrame({'Model': ['Tree_reg', 'Linear',
'Tree_Class', 'Logistic']})

    for col_name in data.columns[:8]:

        data_pred_r = get_data(data, col_name=col_name,
classification=False).dropna(axis=0)
        data_pred_c = get_data(data, col_name=col_name,
classification=True).dropna(axis=0)
```

```
        iter_cv = 20
        score = np.empty([4, iter_cv])

        for i in range(iter_cv):

            x_train_r, x_test_r, y_train_r, y_test_r =
split_data(data_pred_r, test_size=0.2)
            x_train_c, x_test_c, y_train_c, y_test_c =
split_data(data_pred_c, test_size=0.2)

            tree_r = DecisionTreeRegressor()
            tree_r.fit(x_train_r, y_train_r)
            score[0, i] = tree_r.score(x_test_r, y_test_r)

            lm = LinearRegression(normalize=True)
            lm.fit(x_train_r, y_train_r)
            score[1, i] = lm.score(x_test_r, y_test_r)

            tree_c = DecisionTreeClassifier()
            tree_c.fit(x_train_c, y_train_c)
            score[2, i] = tree_c.score(x_test_c, y_test_c)

            lg = LogisticRegression(penalty='none')
            lg.fit(x_train_c, y_train_c)
            score[3, i] = lg.score(x_test_c, y_test_c)

        xtra = pd.DataFrame({'with {}'.format(col_name):
score.mean(axis=1)})
        df_prediction = pd.concat([df_prediction, xtra], axis=1)

    df_prediction.set_index('Model', inplace=True)
    df_prediction.to_csv('{}/prediction.csv'.format(output_dir))


if __name__ == '__main__':
    main()
```

```
import glob
import os
import sys

import pandas as pd


def parse_args():
    data_path = sys.argv
    return data_path


def process_file(file):
    # throw away top part
    top_part_ending = ["All rights reserved", "FDCH e-Media."]
    for top in top_part_ending:
        if top in file:
            file.split(top)[1]

    f = file.split("\n")
    indexes = []

    f_no_factiva = []
    for line in f:
        if len(line) == 0:
            continue
        if line.startswith("\f"):
            line = line[1:]
        if not line.endswith(" Factiva, Inc. All rights reserved."):
            f_no_factiva.append(line)
    count = 0
    for line in f_no_factiva:
        if ":" in line:
            indexes.append(count)
        count = count + 1

    final_text = []
    tuples = [[x, y] for x, y in zip(indexes, indexes[1:])]
    for tup in tuples:
        line = f_no_factiva[tup[0]:tup[1]]
        line = " ".join(line).replace("\n", "")
        final_text.append(line.strip())
    string_out = ""
    ceos_cfos = ["MARIANNE L", "JAMIE D", "JENNIFER A", "BILL H",
"DINA D", "MIKE C"]
    for speaker in final_text:
        for person in ceos_cfos:
            if speaker.startswith(person):
                string_out = string_out + "\n" + speaker
    return string_out


def read_files(entries):
    transcript = []
```

```
    quarter_year = []
    # for each earning call do the following
    for file in entries:
        # extract basename of file ( i.e. only "name.txt")
        basename = os.path.basename(file)
        # from file name get the quarter and year
        quarter_f = basename[0:2]
        year_f = basename[3:7]
        # read the file into a string
        with open(file, "r") as call:
            unprocessed_f = call.read()

        # clean file and get only the speakers that we want
        file_clean = process_file(unprocessed_f)
        transcript.append(file_clean)
        quarter_year.append('{}_{}'.format(quarter_f, year_f))

    return transcript, quarter_year


def main():
    # parse input arguements 1. where JP morgan calls flder  is 2.
where price  folder is 3. where to write transcript
    try:
        args = parse_args()
        earning_dir = args[1]
        output_dir = args[2]
    except IndexError:
        print('Usage: python preprocess.py <earning call path>
<output path>')
        sys.exit()

    # get a list of all the files in a directory
    entries = []

    for f in glob.glob('{}/*'.format(earning_dir)):
        entries.append(f)

    # read all files and return date , transcript and quarter year
    transcript, quarter_year = read_files(entries)
    df_t = pd.DataFrame(data={'quarter_year': quarter_year,
'transcript': transcript})
    df_t.to_csv("{}/transcripts.csv".format(output_dir))


if __name__ == '__main__':
    main()
```

```
import re
import sys

import pandas as pd
from textblob import TextBlob


def parse_args():
    data_path = sys.argv
    return data_path


def read_lexicon(positive, negative, basecase = False):
    postive_lex = {'big', 'grew', 'growth', 'high', 'increased',
'margin', 'over', 'profit', 'strong', 'up'}
    negative_lex = {'down', 'debt', 'loss', 'not', 'reduce',
'reduced', 'restrict', 'restricted', 'weak'}
    if basecase:
        return postive_lex, negative_lex
    with open(positive, "r") as p:
        pos_word = p.readlines()[1:]
    with open(negative, "r") as n:
        neg_word = n.readlines()[1:]

    postive_lex.update(x.rstrip().lower() for x in pos_word)
    negative_lex.update(x.rstrip().lower() for x in neg_word)
    return postive_lex, negative_lex


def numeric_sentiment_based_scoring(text, positive_lex,
negative_lex):
    """ window_size: integer
        alpha: float in range (0, 1)
    """
    tb = TextBlob(text)
    positive = 0
    negative = 0
    index = 0
    numeric_regex = re.compile('\d+\.*\d*')
    window_size = 5
    while index < len(tb.tokens):
        token = tb.tokens[index]
        if re.match(numeric_regex, token) is not None:
            low_bound = index - window_size
            high_bound = index + window_size
            # Iterate window_size words before and after token
            for l in tb.tokens[low_bound:index] + tb.tokens[index +
1:high_bound + 1]:
                if l in positive_lex:
                    positive += 1
                if l in negative_lex:
                    negative += 1
                # More rules can be added in these lines, for
example, composite rules.
```

```
        index += 1
    score = (positive - negative) / (positive + negative)
    return score


def main():
    try:
        args = parse_args()
        data_path = args[1]
        positive = args[2]
        negative = args[3]
        output_dir = args[4]
    except IndexError:
        print('Usage: python preprocess_data.py <data_path> <lexicon
path> <output dir>')
        sys.exit()

    # read csv
    data = pd.read_csv(data_path, index_col=0)
    positive_lex_basecase, negative_lex_basecase =
read_lexicon(positive, negative, True)

    positive_lex, negative_lex = read_lexicon(positive, negative)
    data['numeric_sentiment_score_basecase'] =
data['transcript'].apply(numeric_sentiment_based_scoring,

args=(positive_lex, negative_lex))
    data['numeric_sentiment_score'] =
data['transcript'].apply(numeric_sentiment_based_scoring,

args=(positive_lex_basecase, negative_lex_basecase))
    data = data.drop(columns=["transcript"])
    data.to_csv("{}/rule_based_scores.csv".format(output_dir))


if __name__ == '__main__':
    main()
```

```
import glob
import sys
import numpy as np
import pandas as pd
from textblob import TextBlob
from nltk.tokenize import sent_tokenize


def parse_args():
    data_path = sys.argv
    return data_path


def text_polarity(r, column):
    text_in_sentences = [r[column].split("\n")[1:]][0]
    #  we want to score without the name of the speaker
    keep_only_text = [x.split(":")[1] for x in text_in_sentences]
    tb = TextBlob("".join(keep_only_text))
    return tb.polarity


def sentence_polarity_avg(r, column):
    polarities = []
    text_in_sentences = [r[column].split("\n")[1:]][0]
    #  we want to score without the name of the speaker
    keep_only_text = [x.split(":")[1] for x in text_in_sentences]
    tb = TextBlob("".join(keep_only_text))
    for sentence in tb.sentences:
        polarities.append(sentence.polarity)
    return np.mean(polarities)


def check_sentence_length(sentence):
    tokens = sentence.split()
    if len(tokens) < 4:
        return True
    else:
        return False


def check_word_in_lexicon(sentence, lexicon):
    # todo: we can refine the string matching here or normalize our
words?
    tokens = sentence.split()
    for tok in tokens:
        if tok in lexicon:
            return False
    return True


def filtering(r, column, method, lexicon=None):
    text = r[column].split("\n")[1:]
    words_filtered = []
    words_thrown = []
```

```
    for speaker in text:
        # sentences we keep
        filtered_sentences = []
        # sentences we throw out
        kicked_out = []
        # split their name and their words
        split = speaker.split(":")
        person = split[0]
        words = speaker.split(":")[1]
        words = sent_tokenize(words)

        for sentence in words:
            # use the sentence length method to calculate score or
lexicon method
            if method == "len":
                if check_sentence_length(sentence):
                    kicked_out.append(sentence)
                    continue
                else:
                    filtered_sentences.append(sentence)
            elif method == "lexicon":
                if check_word_in_lexicon(sentence, lexicon):
                    kicked_out.append(sentence)
                    continue
                else:
                    filtered_sentences.append(sentence)

        # put together again their words without the filtered
sentences
        if len(filtered_sentences) > 0:
            words_filtered.append("\n" + person + ":" +
"".join(filtered_sentences))
        if len(kicked_out) > 0:
            words_thrown.append(kicked_out)
        # put together all the turns of the speakers

    return "".join(words_filtered)


def read_lexicon(entries):
    list_words = []
    for file in entries:
        with open(file, "r") as f:
            list_words.append(f.readlines()[1:])
    keep_words = set()
    for sentiment_list in list_words:
        for word in sentiment_list:
            keep_words.add(word.rstrip().lower())
    return keep_words


def main():
    try:
        args = parse_args()
```

```
        data_path = args[1]
        lexicon_dir = args[2]
        output_dir = args[3]
    except IndexError:
        print('Usage: python preprocess_data.py <data_path> <lexicon
path> <output dir>')
        sys.exit()
    # read csv
    data = pd.read_csv(data_path, index_col=0)

    # get a list of lexicon files
    entries = []
    for f in glob.glob('{}/*.csv'.format(lexicon_dir)):
        entries.append(f)
    lexicon_dictionary = read_lexicon(entries)

    # Base case scoring without filtering with the two methods
    data['txt_polarity_basecase'] = data.apply(text_polarity,
args=("transcript",), axis=1)
    data['sentence_polarity_basecase'] =
data.apply(sentence_polarity_avg, args=("transcript",), axis=1)

    # Do the filtering Method length
    data['transcripts_filtered_length'] = data.apply(filtering,
args=("transcript", "len"), axis=1)
    # Do the filtering Method lexicon
    data['transcripts_filtered_lexicon'] = data.apply(filtering,
args=("transcript", "lexicon", lexicon_dictionary), axis=1)

    # Calculate the new scores for length method
    data['txt_polarity_filtered_length'] = data.apply(text_polarity,
args=("transcripts_filtered_length",), axis=1)
    data['sentence_polarity_filtered_length'] =
data.apply(sentence_polarity_avg,
args=("transcripts_filtered_length",), axis=1)

    # Calcualte the new scores for lexicon method
    data['txt_polarity_filtered_lexicon'] =
data.apply(text_polarity, args=("transcripts_filtered_lexicon",),
axis=1)
    data['sentence_polarity_filtered_lexicon'] =
data.apply(sentence_polarity_avg,
args=("transcripts_filtered_lexicon",),
                                                                axis=1)

    data = data.drop(columns=["transcripts_filtered_length",
"transcript", "transcripts_filtered_lexicon"])
    data.to_csv("{}/NPL_scores.csv".format(output_dir))


if __name__ == '__main__':
    main()
```

R Notebook

```r
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```r
library(ggplot2)
library(tidyverse)
```

```
## -- Attaching packages ----------------------------------------------------- tidyverse 1.3.0 --

## v tibble  2.1.3     v purrr   0.3.3
## v tidyr   1.0.0     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.4.0

## -- Conflicts -------------------------------------------------------- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```r
library(caret)
```

```
## Loading required package: lattice

##
## Attaching package: 'caret'

## The following object is masked from 'package:purrr':
##
##     lift
```

```r
#install.packages("car",dependencies=TRUE)
library(car)
```

```
## Loading required package: carData

##
## Attaching package: 'car'

## The following object is masked from 'package:purrr':
##
##     some

## The following object is masked from 'package:dplyr':
##
##     recode
```

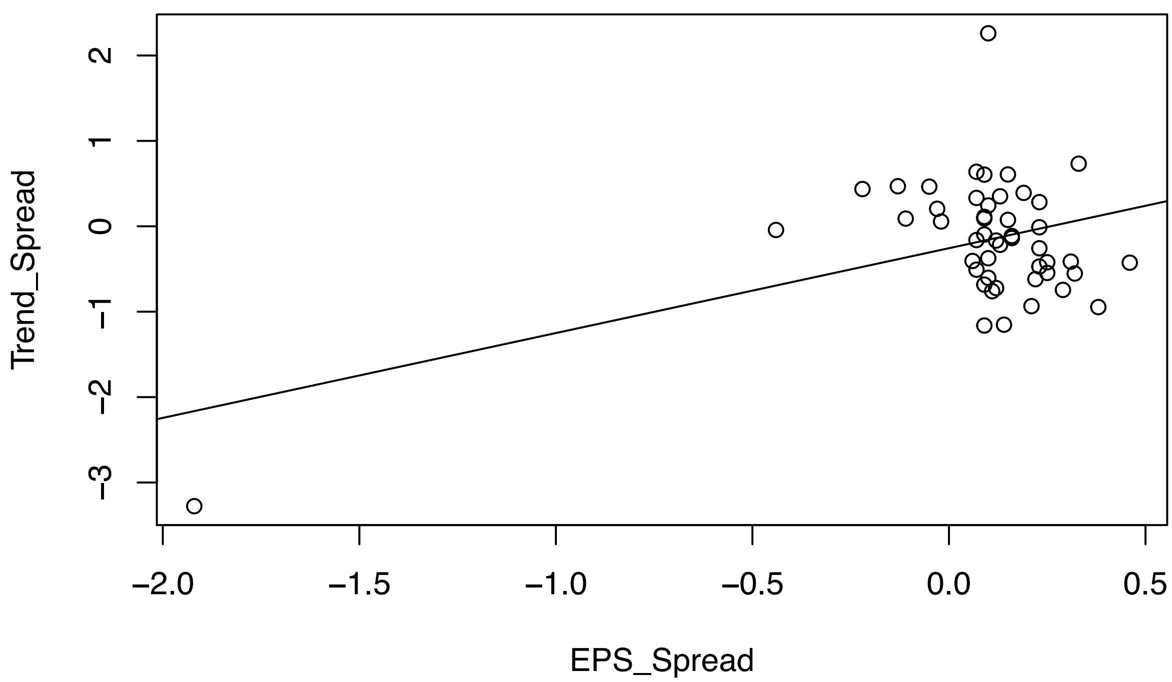
```r
library(gvlma)
data <- read.csv("data_stats.csv")
```

1. Simple linear regression i)EPS vs Trend_Spread

```r
lm_EPS <- lm(Trend_Spread~EPS_Spread, data =  data)
sum_lm_EPS <- summary(lm_EPS)
sum_lm_EPS
```

```
##
## Call:
## lm(formula = Trend_Spread ~ EPS_Spread, data = data)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.11313 -0.52119 -0.02286  0.46386  2.41736
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -0.2567     0.1018  -2.521   0.0153 *
## EPS_Spread    0.9933     0.3015   3.294   0.0019 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.6853 on 46 degrees of freedom
##   (14 observations deleted due to missingness)
## Multiple R-squared:  0.1909, Adjusted R-squared:  0.1733
## F-statistic: 10.85 on 1 and 46 DF,  p-value: 0.001903
```

```r
with(data, plot(EPS_Spread,Trend_Spread))
abline(lm_EPS)
```



ii)txt_polarity_basecase & txt_polarity_filtered_lexicon vs Trend_Spread

```r
colnames(data)
```

```
##  [1] "Quarter_Year"                     "txt_polarity_basecase"
##  [3] "sentence_polarity_basecase"        "txt_polarity_filtered_length"
##  [5] "sentence_polarity_filtered_length" "txt_polarity_filtered_lexicon"
##  [7] "sentence_polarity_filtered_lexicon" "numeric_sentiment_score"
##  [9] "numeric_sentiment_score_basecase"   "EPS_Spread"
## [11] "Trend_Spread"                     "Trend_Spread_Class"
```

```r
head(data)
```

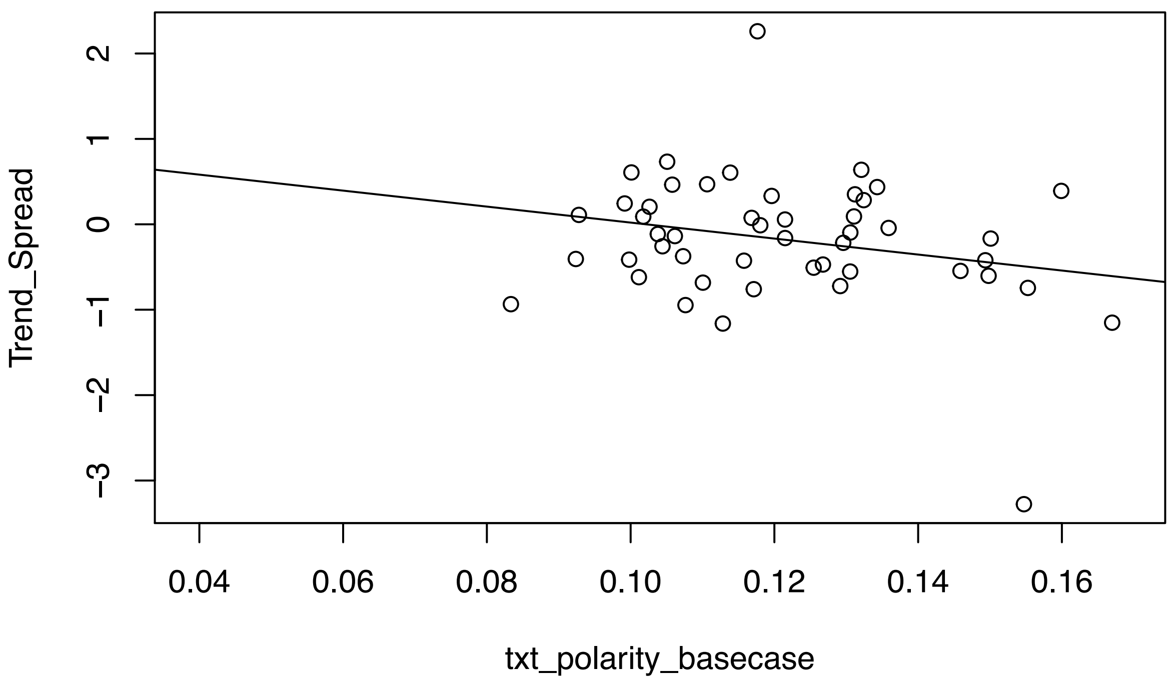
```
##   Quarter_Year txt_polarity_basecase sentence_polarity_basecase
## 1      Q3_2019             0.1267547                 0.11691864
## 2      Q3_2018             0.1176491                 0.09072403
## 3      Q4_2010             0.1295794                 0.09628150
## 4      Q1_2018             0.1100527                 0.09159486
## 5      Q4_2011             0.1214739                 0.07799837
## 6      Q1_2019             0.1050817                 0.07708446
##   txt_polarity_filtered_length sentence_polarity_filtered_length
## 1                    0.1182558                        0.12065628
## 2                    0.1149488                        0.09088038
## 3                    0.1268051                        0.12053531
## 4                    0.1045307                        0.17776528
## 5                    0.1158916                        0.10568206
## 6                    0.1142718                        0.14059921
##   txt_polarity_filtered_lexicon sentence_polarity_filtered_lexicon
## 1                     0.1818258                         0.13965773
## 2                     0.1468105                         0.16538178
## 3                     0.1345811                         0.14835217
## 4                     0.1129000                         0.11120730
## 5                     0.1410116                         0.06225377
## 6                     0.1326509                         0.22606248
##   numeric_sentiment_score numeric_sentiment_score_basecase EPS_Spread
## 1               0.6904762                        0.7368421       0.23
## 2               0.5463918                        0.5402299       0.10
## 3              -0.2500000                       -0.2727273       0.13
## 4               0.4947368                        0.4761905       0.09
## 5              -0.3000000                       -0.1428571      -0.02
## 6               0.5172414                        0.5584416       0.33
##   Trend_Spread Trend_Spread_Class
## 1       -0.471                  0
## 2        2.260                  1
## 3       -0.217                  0
## 4       -0.683                  0
## 5        0.056                  1
## 6        0.733                  1
```

```r
lm_txt_polarity_basecase <- lm(Trend_Spread~txt_polarity_basecase, data = data)
sum_lm_txt_polarity_basecase <- summary(lm_txt_polarity_basecase)
sum_lm_txt_polarity_basecase
```

```
##
## Call:
## lm(formula = Trend_Spread ~ txt_polarity_basecase, data = data)
##
```

```
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.78496 -0.30529  0.02165  0.39593  2.40511
##
## Coefficients:
##                        Estimate Std. Error t value Pr(>|t|)
## (Intercept)              0.9561     0.6783   1.409   0.1654
## txt_polarity_basecase   -9.3598     5.5344  -1.691   0.0976 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.7392 on 46 degrees of freedom
##   (14 observations deleted due to missingness)
## Multiple R-squared:  0.05854,    Adjusted R-squared:  0.03807
## F-statistic:  2.86 on 1 and 46 DF,  p-value: 0.09756
```

```
with(data, plot(txt_polarity_basecase,Trend_Spread))
abline(lm_txt_polarity_basecase)
```

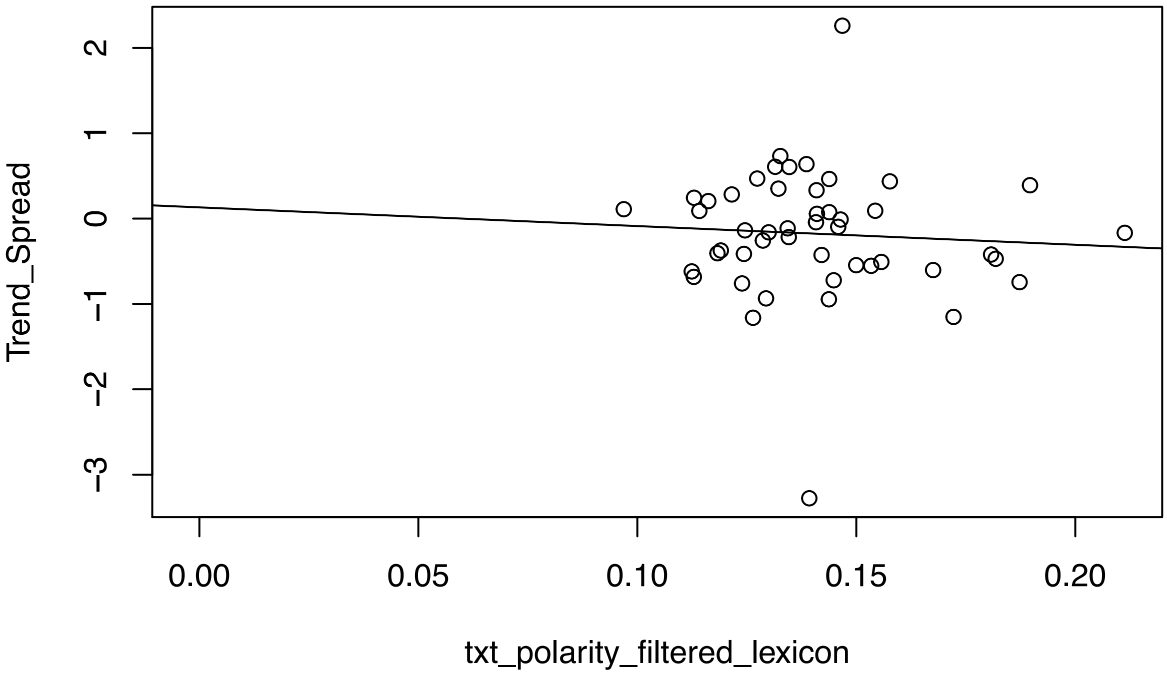



```
lm_txt_polarity_filtered_lexicon <- lm(Trend_Spread~txt_polarity_filtered_lexicon, data = data)
sum_lm_txt_polarity_filtered_lexicon <- summary(lm_txt_polarity_filtered_lexicon)
sum_lm_txt_polarity_filtered_lexicon
```

```
##
## Call:
## lm(formula = Trend_Spread ~ txt_polarity_filtered_lexicon, data = data)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -3.10357 -0.34875  0.02596  0.37429  2.44995
##
## Coefficients:
##                                Estimate Std. Error t value Pr(>|t|)
```

```
## (Intercept)                       0.1314     0.6930    0.19    0.850
## txt_polarity_filtered_lexicon  -2.1891     4.8592   -0.45    0.654
##
## Residual standard error: 0.7602 on 46 degrees of freedom
##   (14 observations deleted due to missingness)
## Multiple R-squared:  0.004393,   Adjusted R-squared:  -0.01725
## F-statistic: 0.203 on 1 and 46 DF,  p-value: 0.6545
```

```
with(data, plot(txt_polarity_filtered_lexicon,Trend_Spread))
abline(lm_txt_polarity_filtered_lexicon)
```




```
txt_polarity_basecase_r2 <- data.frame(cbind(sum_lm_txt_polarity_basecase$r.squared, sum_lm_txt_polarit
colnames(txt_polarity_basecase_r2) <- c("R.squared", "adj.R.squared","P_value for slope")


txt_polarity_filtered_lexicon_r2 <- data.frame(cbind(sum_lm_txt_polarity_filtered_lexicon$r.squared, su
colnames(txt_polarity_filtered_lexicon_r2) <- c("R.squared", "adj.R.squared","P_value for slope")
txt.vs.txt_filtered <- rbind(txt_polarity_basecase_r2,txt_polarity_filtered_lexicon_r2)
rownames(txt.vs.txt_filtered) <- c("txt","txt_filtered")
txt.vs.txt_filtered
```

```
##                R.squared adj.R.squared P_value for slope
## txt          0.058537535    0.03807096        0.09756435
## txt_filtered 0.004392581   -0.01725106        0.65446625
```

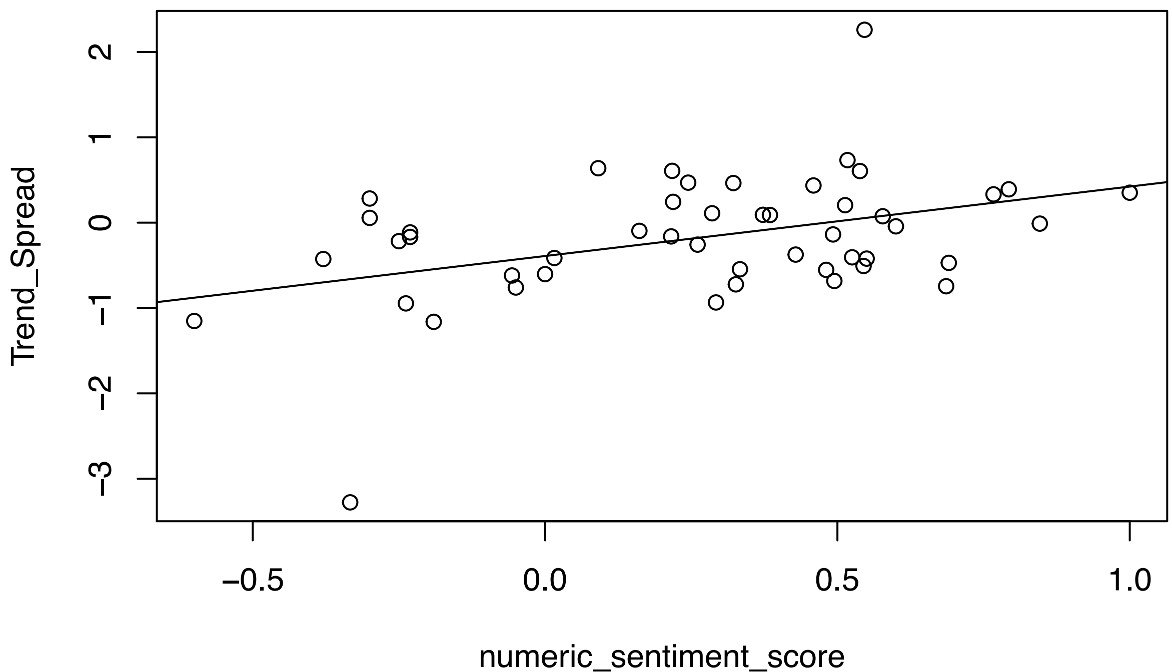
iii)numeric_sentiment_score & numeric_sentiment_score_basecase vs Trend_Spread

```
lm_numeric_sentiment_score <- lm(Trend_Spread~numeric_sentiment_score, data = data)
sum_lm_numeric_sentiment_score <- summary(lm_numeric_sentiment_score)
sum_lm_numeric_sentiment_score
```

```
##
## Call:
## lm(formula = Trend_Spread ~ numeric_sentiment_score, data = data)
##
```

```
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.61469 -0.37747 -0.01992  0.42226  2.20635
##
## Coefficients:
##                         Estimate Std. Error t value Pr(>|t|)
## (Intercept)              -0.3910     0.1238  -3.158  0.00281 **
## numeric_sentiment_score   0.8138     0.2737   2.973  0.00468 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.6978 on 46 degrees of freedom
##   (14 observations deleted due to missingness)
## Multiple R-squared:  0.1612, Adjusted R-squared:  0.143
## F-statistic:  8.84 on 1 and 46 DF,  p-value: 0.004679
```

```
with(data, plot(numeric_sentiment_score,Trend_Spread))
abline(lm_numeric_sentiment_score)
```

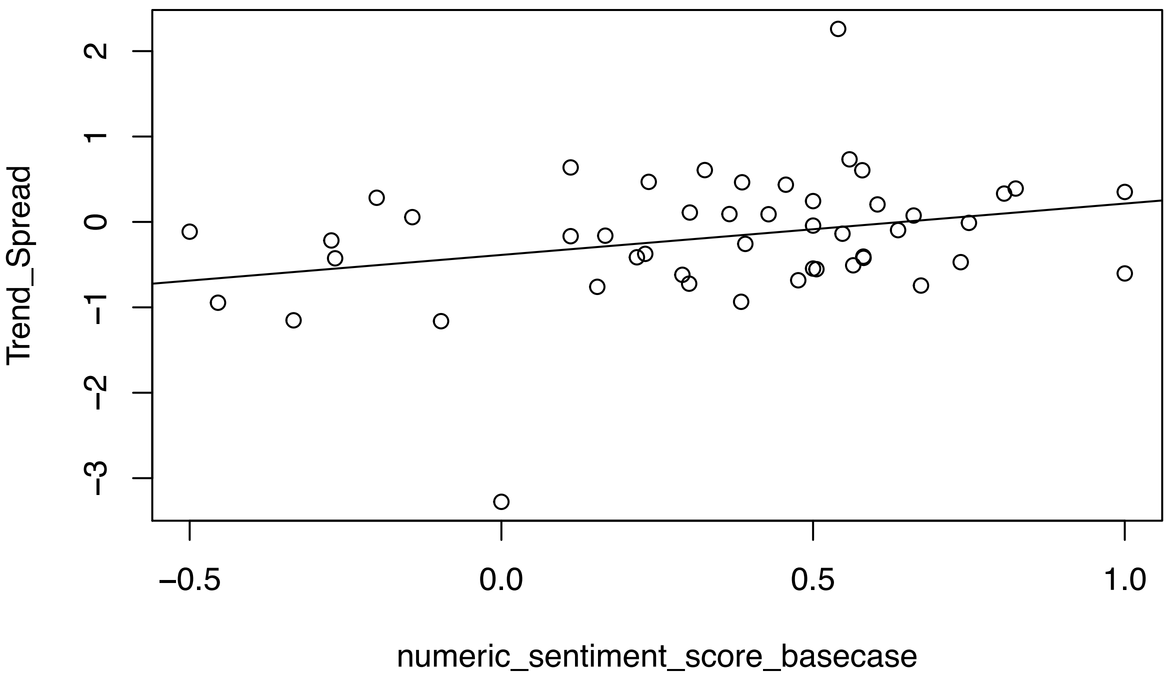



```
lm_numeric_sentiment_score_basecase <- lm(Trend_Spread~numeric_sentiment_score_basecase, data = data)
sum_lm_numeric_sentiment_score_basecase <- summary(lm_numeric_sentiment_score_basecase)
sum_lm_numeric_sentiment_score_basecase
```

```
##
## Call:
## lm(formula = Trend_Spread ~ numeric_sentiment_score_basecase,
##     data = data)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.89072 -0.46061  0.05307  0.33050  2.32137
##
## Coefficients:
```

```
##                                   Estimate Std. Error t value Pr(>|t|)
## (Intercept)                        -0.3863     0.1472  -2.624   0.0118 *
## numeric_sentiment_score_basecase    0.6014     0.2954   2.036   0.0475 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.7297 on 46 degrees of freedom
##   (14 observations deleted due to missingness)
## Multiple R-squared:  0.08265,    Adjusted R-squared:  0.06271
## F-statistic: 4.145 on 1 and 46 DF,  p-value: 0.04755
```

```
with(data, plot(numeric_sentiment_score_basecase,Trend_Spread))
abline(lm_numeric_sentiment_score_basecase)
```




```
numeric_sentiment_score_r2 <- data.frame(cbind(sum_lm_numeric_sentiment_score$r.squared, sum_lm_numeric_
colnames(numeric_sentiment_score_r2) <- c("R.squared", "adj.R.squared","P_value for slope")


numeric_sentiment_score_basecase_r2 <- data.frame(cbind(sum_lm_numeric_sentiment_score_basecase$r.square
colnames(numeric_sentiment_score_basecase_r2) <- c("R.squared", "adj.R.squared","P_value for slope")

score.vs.baseline <- rbind(numeric_sentiment_score_r2,numeric_sentiment_score_basecase_r2)
rownames(score.vs.baseline) <- c("score","basecase")
score.vs.baseline
```

```
##           R.squared adj.R.squared P_value for slope
## score    0.16119258    0.14295764       0.004679046
## basecase 0.08265421    0.06271191       0.047548642
```

2.Multiple Linear Regression

```
mlr <- lm(Trend_Spread~EPS_Spread+numeric_sentiment_score+txt_polarity_basecase, data = data)
sum_mlr <- summary(mlr)
sum_mlr
```

```
##
## Call:
## lm(formula = Trend_Spread ~ EPS_Spread + numeric_sentiment_score +
##     txt_polarity_basecase, data = data)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.07868 -0.43658 -0.00825  0.34290  2.20685
##
## Coefficients:
##                         Estimate Std. Error t value Pr(>|t|)
## (Intercept)               0.1979     0.6144   0.322  0.74892
## EPS_Spread                0.8197     0.2908   2.819  0.00719 **
## numeric_sentiment_score   0.6938     0.2533   2.739  0.00886 **
## txt_polarity_basecase    -5.1493     4.9180  -1.047  0.30080
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.639 on 44 degrees of freedom
##   (14 observations deleted due to missingness)
## Multiple R-squared:  0.3272, Adjusted R-squared:  0.2813
## F-statistic: 7.131 on 3 and 44 DF,  p-value: 0.0005256
```

```r
newdata <- data.frame(cbind(data$EPS_Spread,data$numeric_sentiment_score,data$txt_polarity_basecase))
colnames(newdata) <- c("EPS_Spread", "numeric_sentiment_score", "txt_polarity_basecase")
Trend_Spread_pre <- predict(mlr, newdata = newdata)
pre_table <- data.frame(cbind(Trend_Spread_pre, data$Trend_Spread))
colnames(pre_table) <- c("Trend_Spread_pre","Trend_Spread")
na.omit(pre_table)
```

```
##    Trend_Spread_pre Trend_Spread
## 1      0.2127931909     -0.47100
## 2      0.0531528893      2.26000
## 3     -0.5362377024     -0.21700
## 4      0.0482330224     -0.68300
## 5     -0.6521440871      0.05600
## 6      0.2861686058      0.73300
## 7      0.0296801457     -0.25700
## 9     -0.0079437329      0.11000
## 10    -0.2206466604     -0.16100
## 13    -0.1493831074      0.09000
## 14    -0.0128449280     -0.50800
## 15     0.1120148939     -0.74500
## 18    -0.4462093004     -0.04300
## 19    -0.9635287318     -1.15200
## 20     0.1215632251     -0.55300
## 21     0.1714959088      0.33200
## 27    -0.5036028219      0.28300
## 28    -0.0438113651      0.60700
## 29    -0.4914871558     -0.60300
## 30    -0.2887440457     -0.09600
## 31     0.0589863541      0.60500
## 32     0.0804849272      0.39100
## 33    -0.1643865869      0.46400
## 34     0.0009078796      0.20500
```

```
## 35    -0.0788021007      0.24400
## 36    -0.1447708853      0.09200
## 37    -0.1818829649     -0.61900
## 38    -2.4038418006     -3.27700
## 39     0.3226245947      0.35100
## 40    -0.3085647899      0.46900
## 41     0.1357942122     -0.40600
## 42     0.1436763461     -0.93500
## 43    -0.3654469826     -0.11400
## 44     0.0153971547     -0.42100
## 45    -0.6367194678     -0.16698
## 46    -0.3555122946      0.43600
## 47    -0.3497633139     -0.76000
## 48    -0.2843414672     -0.42600
## 51     0.1198849555      0.07500
## 52    -0.2100444419     -0.94600
## 53    -0.3618436279      0.63800
## 54    -0.4414590591     -1.16200
## 55     0.3657044613     -0.01100
## 56    -0.1424336872     -0.72300
## 57     0.0246681852     -0.37400
## 60     0.1241715019     -0.13800
## 61    -0.1171410275     -0.54600
## 62    -0.0508443173     -0.41400
```

```
##Check mean residuals are 0
mean(sum_mlr$residuals)
```

```
## [1] -7.256249e-18
```

```
##Correlation Test
cor.test(na.omit(data$Trend_Spread), sum_mlr$residuals)
```

```
##
##  Pearson's product-moment correlation
##
## data:  na.omit(data$Trend_Spread) and sum_mlr$residuals
## t = 9.7265, df = 46, p-value = 9.759e-13
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  0.6990624 0.8956559
## sample estimates:
##       cor
## 0.8202689
```

```
##Detecting multicollonearity
vif(mlr)
```

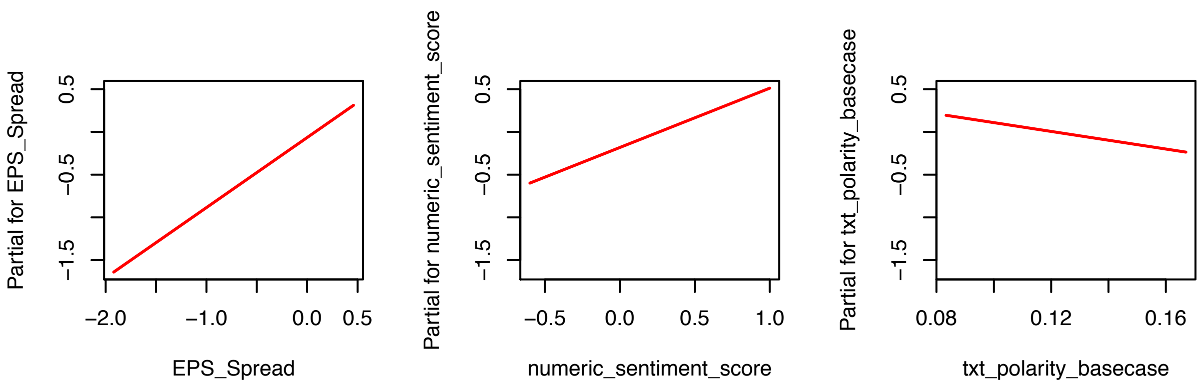
```
##              EPS_Spread numeric_sentiment_score   txt_polarity_basecase
##                1.069715                1.021149                1.056851
```

```
par(mfrow=c(2,2))
mlr <- lm(Trend_Spread~EPS_Spread+numeric_sentiment_score+txt_polarity_basecase, data = data)
gvlma(mlr)
```

```
##
## Call:
```

```
## lm(formula = Trend_Spread ~ EPS_Spread + numeric_sentiment_score +
##     txt_polarity_basecase, data = data)
##
## Coefficients:
##             (Intercept)               EPS_Spread  numeric_sentiment_score
##                  0.1979                   0.8197                   0.6938
##   txt_polarity_basecase
##                 -5.1493
##
##
## ASSESSMENT OF THE LINEAR MODEL ASSUMPTIONS
## USING THE GLOBAL TEST ON 4 DEGREES-OF-FREEDOM:
## Level of Significance =  0.05
##
## Call:
##  gvlma(x = mlr)
##
##                     Value   p-value                   Decision
## Global Stat        24.377 6.711e-05 Assumptions NOT satisfied!
## Skewness            5.744 1.655e-02 Assumptions NOT satisfied!
## Kurtosis            5.108 2.382e-02 Assumptions NOT satisfied!
## Link Function      10.254 1.364e-03 Assumptions NOT satisfied!
## Heteroscedasticity  3.271 7.049e-02    Assumptions acceptable.
```

```r
par(mfrow=c(2,3))
termplot(mlr)
#plot(mlr)
```



Statistical Analysis on Numeric Variables

In [59]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn import metrics
%matplotlib inline
```

In [22]:
```
##read tables
##below are three tables we need to combined and analyze
df1=pd.read_csv('/Users/chenpengguan/Desktop/GR5293/NLP_scores.csv',index_col=0)
df2=pd.read_csv('/Users/chenpengguan/Desktop/GR5293/rule_based_scores.csv',index_col=0)
df3=pd.read_csv('/Users/chenpengguan/Desktop/GR5293/spread.csv',index_col=0 )
```

In [23]:
```
#df1.head()
```

In [24]:
```
#df2.shape
```

In [25]:
```
#df2.head()
```

In [26]:
```
#df3.head()
```

In [27]:
```
#df3.shape
```

In [28]:
```
##join three tables together
df_left = pd.merge(df1, df2, on='Quarter_Year', how='left')
df=pd.merge(df_left,df3, on='Quarter_Year', how='left')
```

In [29]:
```
df.head()
```

Out[29]:

	Quarter_Year	txt_polarity_basecase	sentence_polarity_basecase	txt_polarity_filtered_length	sentence
0	Q3_2019	0.126755	0.116919	0.118256	
1	Q3_2018	0.117649	0.090724	0.114949	
2	Q4_2010	0.129579	0.096281	0.126805	
3	Q1_2018	0.110053	0.091595	0.104531	
4	Q4_2011	0.121474	0.077998	0.115892	

Explanation for variables

```
- txt_polarity is overall score for a transcript
- sentence_polarity is sentence wise scores
- above two each have three methods (basecase,length,lexicon)
- numeric_sentiment_score is a scoring method based on words around numbers in EC
tanscripts
- numeric_sentiment_score has a basecase column and a more accurated, improved col
umn
```

In [35]:
```
df['Trend_Spread_Class']=df['Trend_Spread']
```

In [36]:
```
for i in range(0,len(df['Quarter_Year'])-1):
    if df['Trend_Spread'][i]<0:
       df['Trend_Spread_Class'][i]=0
    else:
       df['Trend_Spread_Class'][i]=1
```

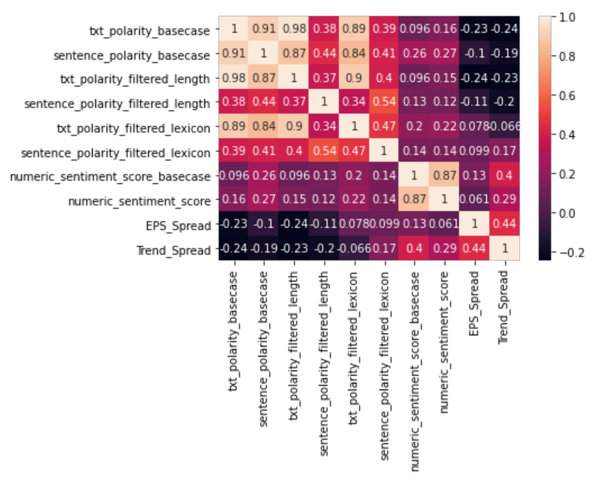
```
/Users/chenpengguan/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:3: Setti
ngWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
guide/indexing.html#returning-a-view-versus-a-copy
  This is separate from the ipykernel package so we can avoid doing imports until
/Users/chenpengguan/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:5: Setti
ngWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
guide/indexing.html#returning-a-view-versus-a-copy
  """
```

In [40]:
```
df.shape
```

Out[40]: (62, 12)

In [48]:
```
##a subset for analysis
df_elr=df.iloc[:,1:11]
```

In [57]:
```
corrMatrix = df_elr.corr()
sns.heatmap(corrMatrix, annot=True)
plt.show()
```



In [66]:
```
df.describe()
```

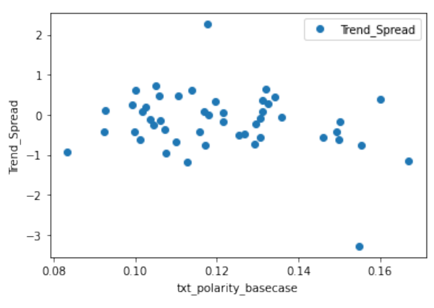
Out[66]:

	txt_polarity_basecase	sentence_polarity_basecase	txt_polarity_filtered_length	sentence_polarity_fi
count	62.000000	62.000000	62.000000	
mean	0.118167	0.095324	0.116403	
std	0.026204	0.022079	0.027338	
min	0.039010	0.030668	0.024897	
25%	0.102912	0.084741	0.102737	
50%	0.117390	0.093049	0.115899	
75%	0.131874	0.107008	0.130370	
max	0.169164	0.144983	0.174152	

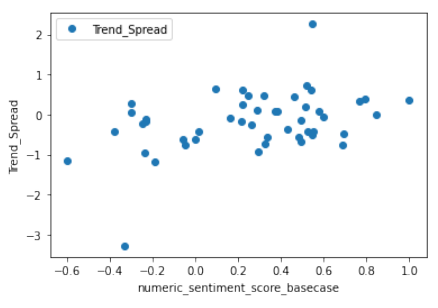
In [67]:
```
##Cor with Trend——Spread
corr=df_elr.corr()
corr['Trend_Spread']
```

Out[67]:
```
txt_polarity_basecase                  -0.241945
sentence_polarity_basecase             -0.193356
txt_polarity_filtered_length           -0.231591
sentence_polarity_filtered_length      -0.202263
txt_polarity_filtered_lexicon          -0.066277
sentence_polarity_filtered_lexicon      0.168417
numeric_sentiment_score_basecase        0.401488
numeric_sentiment_score                 0.287496
EPS_Spread                              0.436928
Trend_Spread                            1.000000
Name: Trend_Spread, dtype: float64
```

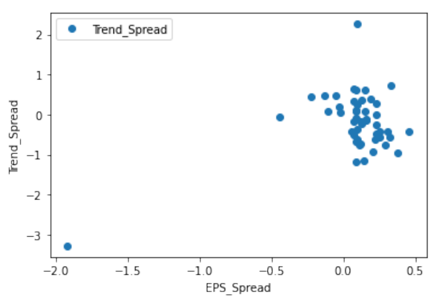
In [60]:
```
df.plot(x='txt_polarity_basecase', y='Trend_Spread', style='o')

plt.xlabel('txt_polarity_basecase')
plt.ylabel('Trend_Spread')
plt.show()
```



In [63]:
```
df.plot(x='numeric_sentiment_score_basecase', y='Trend_Spread', style='o')

plt.xlabel('numeric_sentiment_score_basecase')
plt.ylabel('Trend_Spread')
plt.show()
```



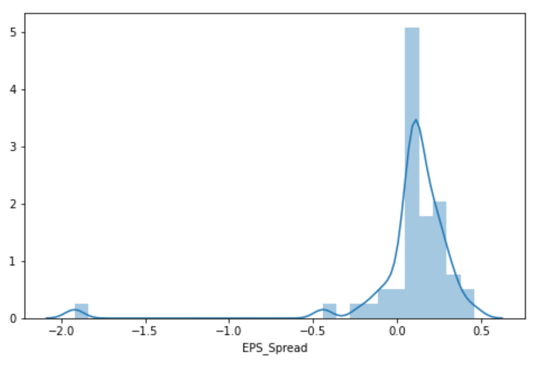
In [65]:
```
df.plot(x='EPS_Spread', y='Trend_Spread', style='o')

plt.xlabel('EPS_Spread')
plt.ylabel('Trend_Spread')
plt.show()
```



Note: we wanna pay attention to the seemingly 'outlier' here. Its affecting the relationship alot, we wanna make sure its not an anomaly data point. Otherwise, its actually a good thing.

In [74]:
```
plt.figure(figsize=(8,5))
plt.tight_layout()
sns.distplot(df['EPS_Spread'])
```

Out[74]: <matplotlib.axes._subplots.AxesSubplot at 0x1a1c0346a0>



Next Step: Need Simple Linear Regression

- EPS vs Trend_Spread
- txt_polarity_basecase vs Trend_Spread
    - sentence_polarity_basecase vs Trend_Spread (to check which one is better)
- numeric_sentiment_score VS Trend_Spread
    - numeric_sentiment_score_basecase vs Trend_Spread (check which one is better)
- CHECK ASSUMPTION PLOTS

Could also try Multiple Linear Regression

- EPS + numeric_sentiment_score + sentence/txt_polarity_basecase ~ Trend_Spread
- then use them to either predict Trend_Spred or binary Trend_Spread_Class
- CHECK ASSUMPTION PLOTS

In [78]:
```
df.to_csv('/Users/chenpengguan/Desktop/GR5293/data_stats.csv', index=False)
```