1 Project 4 – NLP Model

- Student name: Greg Osborne
- Student pace: self paced / part time
- Scheduled project review date/time: 3/23/23
- Instructor name: Morgan Jones
- Blog post URL: https://medium.com/@gregosborne (https://medium.com/@gregosborne)

1.1 Business Understanding

1.1.1 Stakeholder:

Google

1.1.2 Problem:

Some people view Google more negatively than their chief competitor, Apple. Google needs a tool to help measure the publics' sentiment toward both Apple and Google to strategize for a more positive public sentiment. This tool will offer Google insight to guide their business decisions.

1.1.3 Using Natural Language Processing (NLP) to build a tool

Google's departments of both Public Relations and product development could benefit from a snapshot of the publics' perception of both Google and their competition, Apple. The input of this model is tweets that discuss either Google or Apple. The output will how many of the tweets view both companies, positively and negatively, as well as a mentioning them in a neutral way. The model will also classify whether each tweet addresses the company brand, or the products it makes. So each tweet is classified as:

1. Company: Google or Apple
2. Aspect of Company: Brand or Product
3. Emotion: Positive, Negative, or Neutral

This works out to each tweet being classified as one of twelve different classifications:

1. Negative - Apple - Brand
2. Neutral - Apple - Brand
3. Positive - Apple - Brand
4. Negative - Apple - Products
5. Neutral - Apple - Products
6. Positive - Apple - Products
7. Negative - Google - Brand
8. Neutral - Google - Brand
9. Positive - Google - Brand
10. Negative - Google - Products

11. Neutral - Google - Products
12. Positive - Google - Products

The second insight the Natural Language Processing provides comes in associated words. When you think of McDonald's, you probably think of some kind of food, you may also think of Happy Meals, Burgers, French Fries, or a sugary iced frappuccino. What do people think when they think of when they think about Google? Or Google's competition Apple?

Through Word2Vec vectorization methods, we can assign a multidimensional vector to each word found in a set of tweets. Then, we can measure which words align the best. By doing so, we can see what words people identify with each company, and the companies products. And, we can perform this analysis for people speaking both negatively and positively.

Greg Osborne accepted the challenge to build such a tool.

2 Table of Contents

Project 4 – NLP Model

3 Python Libraries

In [1]:
```
# DataFrames and computation
import pandas as pd
import numpy as np

# Graphing
import matplotlib.pyplot as plt
import seaborn as sns
from plotly import graph_objs as go
from collections import Counter
%matplotlib inline
import plotly.express as px
from palettable.colorbrewer.qualitative import Pastel1_7

# Import the Tokenize library
from nltk.tokenize import RegexpTokenizer

# For Frequency Distributions
from nltk import FreqDist
from matplotlib.ticker import MaxNLocator

# For Wordclouds
from wordcloud import WordCloud
from wordcloud import ImageColorGenerator

# For stopwords
import nltk
from nltk.corpus import stopwords

# For stemming words
from nltk.stem.snowball import SnowballStemmer

# The Train/Test Split
from sklearn.model_selection import train_test_split

# Import the TfidfVectorizer class
from sklearn.feature_extraction.text import TfidfVectorizer

# Bag of Words
from sklearn.feature_extraction.text import CountVectorizer

# Word2Vec
import gensim

# For Min/Max Scaling
from sklearn.preprocessing import MinMaxScaler

# Scores
from sklearn.model_selection import cross_val_score
from sklearn.metrics import precision_score, recall_score, accuracy_score
from sklearn.metrics import f1_score
from sklearn.metrics import classification_report

# Multinomial Naive Bayes
from sklearn.naive_bayes import MultinomialNB

# Logistic Regression Model Classifier
from sklearn.linear_model import LogisticRegression
```

```
# Support Vector Machine
from sklearn import svm

# Random Forest Classifier
from sklearn.ensemble import RandomForestClassifier

# For XGBoost Classifier
from xgboost import XGBClassifier

# The Confusion Matrix
from sklearn.metrics import confusion_matrix
from sklearn.metrics import ConfusionMatrixDisplay

# For Hyperparameter Tuning
from sklearn.model_selection import GridSearchCV

# Setting DataFrame Display settings
pd.set_option("display.max_rows", 600)

# Formatting decimals to show three numbers after the point.
pd.options.display.float_format = '{:,.3f}'.format
```

Note, after data cleaning, I will run the entire dataset through the data cleaning process a second time, but without making the same deletions. Therefore, I will program most actions in the form of functions so they can be repeated after data cleaning.

4 Data Cleaning

4.1 Loading the Data and Previewing It

The set of tweets gleaned for this project were all written during the 2011 South By South West festival (Abbreviated as SXSW). Both Google and Apple came to the festival with showcases of their products and presentations about the future direction of the tech industry. Apple constructed a "Pop-Up" store during the event, and people waited in line for hours to get a new iPad2. Google did not launch any new products, but there were rumors and excitement for the imminent launch of their social media platform, codenamed Google Circles. Google held a big party and had one of their top executives, Marissa Mayer, speak.

No, I'll begin to analyze the collected tweets. The first step of any Data Science analysis is, of course, loading and previewing the data.

In [2]:
```
df_raw = pd.read_csv("judge-1377884607_tweet_product_company.csv",
                     encoding='ANSI')
df_raw.head(7)
```

Out[2]:

	tweet_text	emotion_in_tweet_is_directed_at	is_there_an_emotion_directed_at_a_brand_or
0	.@wesley83 I have a 3G iPhone. After 3 hrs twe...	iPhone	Negativ
1	@jessedee Know about @fludapp ? Awesome iPad/i...	iPad or iPhone App	Positiv
2	@swonderlin Can not wait for #iPad 2 also. The...	iPad	Positiv
3	@sxsw I hope this year's festival isn't as cra...	iPad or iPhone App	Negativ
4	@sxtxstate great stuff on Fri #SXSW: Marissa M...	Google	Positiv
5	@teachntech00 New iPad Apps For #SpeechTherapy...	NaN	No emotion toward brand
6	NaN	NaN	No emotion toward brand

In [3]:
```
print('The DataFrame includes',len(df_raw), 'rows of data.')
print()
df_raw.info()
```

```
The DataFrame includes 9093 rows of data.

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9093 entries, 0 to 9092
Data columns (total 3 columns):
 #   Column                                              Non-Null Count  Dtyp
e
---  ------                                              --------------  ----
-
 0   tweet_text                                          9092 non-null   obje
ct
 1   emotion_in_tweet_is_directed_at                     3291 non-null   obje
ct
 2   is_there_an_emotion_directed_at_a_brand_or_product  9093 non-null   obje
ct
dtypes: object(3)
memory usage: 213.2+ KB
```

From this, I see that there are a total of 9093 rows. The tweet_text column is missing one cell of data. However, the emotion_in_tweet_is_direct_at column dwarfs that number at 5801 missing cells. I'll have to account for both of these columns during data cleaning.

4.1.1 Value counts of categorized columns

The people who created this dataset hand classified several of the tweets as one of the following categories. Unfortunately, this classification is incomplete, as nearly 6000 tweets are unclassified in this manner.

In [4]:
```
df_raw['emotion_in_tweet_is_directed_at'].value_counts()
```

Out[4]:
```
iPad                               946
Apple                              661
iPad or iPhone App                 470
Google                             430
iPhone                             297
Other Google product or service    293
Android App                         81
Android                             78
Other Apple product or service      35
Name: emotion_in_tweet_is_directed_at, dtype: int64
```

Likewise, the people who created this dataset classified each tweet as either positive, negative, neutral) or indecipherable. These labels are given to each tweet, though the tweets labeled "I can't tell" were not used in the model or analysis. However, that leaves nearly 9000 classified tweet with which to do our analysis.

In [5]:
```
df_raw['is_there_an_emotion_directed_at_a_brand_or_product'].value_counts()
```

Out[5]:
```
No emotion toward brand or product    5389
Positive emotion                      2978
Negative emotion                       570
I can't tell                           156
Name: is_there_an_emotion_directed_at_a_brand_or_product, dtype: int64
```

Analysis and explanation of next steps.

4.1.2 Rename Columns

The columns need more succinct names.

In [6]:
```
def rename_columns(dforg):
    name = {
        'tweet_text': 'Tweet',
        'emotion_in_tweet_is_directed_at': 'Product',
        'is_there_an_emotion_directed_at_a_brand_or_product': 'Emotion'
        }
    dfone = dforg.copy()
    dfone = dfone.rename(columns=name)
    return dfone.copy()
df = rename_columns(df_raw)
df.head()
```

Out[6]:

	Tweet	Product	Emotion
0	.@wesley83 I have a 3G iPhone. After 3 hrs twe...	iPhone	Negative emotion
1	@jessedee Know about @fludapp ? Awesome iPad/i...	iPad or iPhone App	Positive emotion
2	@swonderlin Can not wait for #iPad 2 also. The...	iPad	Positive emotion
3	@sxsw I hope this year's festival isn't as cra...	iPad or iPhone App	Negative emotion
4	@sxtxstate great stuff on Fri #SXSW: Marissa M...	Google	Positive emotion

In [7]:
```
# The labels for the emotions need to change to short labels for graphing
# purposes.
def abbreviate_emotion(dfone):
    for i in range(len(dfone)):
        if dfone.loc[i, 'Emotion'] == 'No emotion toward brand or product':
            dfone.loc[i, 'Emotion'] = 'Neutral'
        elif dfone.loc[i, 'Emotion'] == 'Negative emotion':
            dfone.loc[i, 'Emotion'] = 'Negative'
        elif dfone.loc[i, 'Emotion'] == 'Positive emotion':
            dfone.loc[i, 'Emotion'] = 'Positive'
        elif dfone.loc[i, 'Emotion'] == "I can't tell":
            dfone.loc[i, 'Emotion'] = 'Indecipherable'
    return dfone.copy()

df = abbreviate_emotion(df)
```

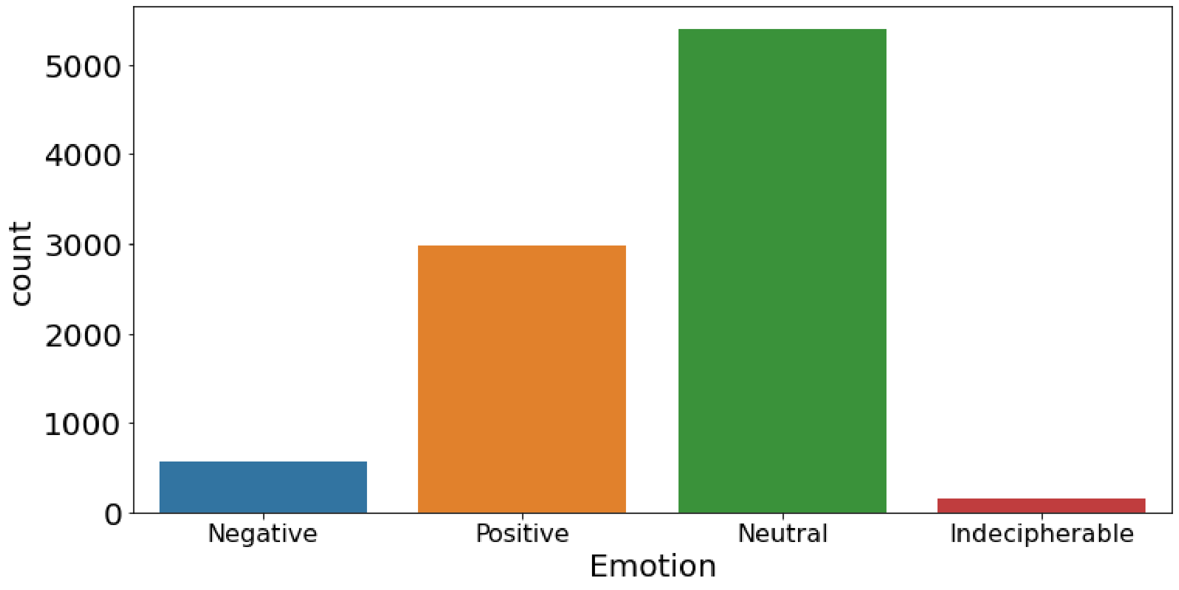
4.1.3 Visualize the quantities of the data.

In [8]:
```
temp = df.groupby('Emotion').count()[
    'Tweet'].reset_index().sort_values(by='Tweet',ascending=False)
temp.style.background_gradient(cmap='Reds')
```

Out[8]:

	Emotion	Tweet
2	Neutral	5388
3	Positive	2978
1	Negative	570
0	Indecipherable	156

In [9]:
```
font = {'family' : 'DejaVu Sans',
            'weight' : 'normal',
            'size'   : 16}
plt.rc('font', **font)
plt.figure(figsize=(12,6))
plt.rc('axes', labelsize=20)
plt.rc('ytick', labelsize=20)
sns.countplot(x='Emotion', data=df);
```



Observations:

1. This graph shows the large disparity between two of the emotional tweet states and the other two.

I will eventually delete the Indecipherable Emotion tweets, but not the negative ones. The negative predictions will suffer from so few tweets to train on.

As is clear, there are more positive and neutral tweets in the set than either negative or indecipherable, contrasting people's general opinion of Twitter. This will likely mean that the model will not be as accurate in predicting a negative emotion in a tweet. I'll see when I get there.

4.2 Preparing Data For Analysis

4.2.1 Standardizing Case

NLP will recognize words by the characters used in succession that make them up. So "This" will be distinct from "this". To solve this problem, I'll make every alphabet character in the tweets lowercase.

In [10]:
```
df['Tweet'] = df['Tweet'].str.lower()
df.head()
```

Out[10]:

	Tweet	Product	Emotion
0	.@wesley83 i have a 3g iphone. after 3 hrs twe...	iPhone	Negative
1	@jessedee know about @fludapp ? awesome ipad/i...	iPad or iPhone App	Positive
2	@swonderlin can not wait for #ipad 2 also. the...	iPad	Positive
3	@sxsw i hope this year's festival isn't as cra...	iPad or iPhone App	Negative
4	@sxtxstate great stuff on fri #sxsw: marissa m...	Google	Positive

4.2.2 Testing for bad data

There are 9093 rows of data and 9092 filled cells in the Tweet column. Therefore, one of the Tweet cells contains no value, so these next two cells detect which row is missing tweet data and deletes it.

In [11]:
```
# Testing for any Tweets that are not strings
bad_tweets = []
for i in range(len(df)):
    typ = type(df.loc[i, 'Tweet'])
    if typ != str:
        print('Index number {} is missing a value in the'.format(i),
              'Tweet column.')
        bad_tweets.append(i)
```

```
Index number 6 is missing a value in the Tweet column.
```

In [12]:
```
# Dropping bad data and reseting the index
df.drop(index=bad_tweets, inplace=True)
df.reset_index(drop=True, inplace=True)
df.head(8)
```

Out[12]:

	Tweet	Product	Emotion
0	.@wesley83 i have a 3g iphone. after 3 hrs twe...	iPhone	Negative
1	@jessedee know about @fludapp ? awesome ipad/i...	iPad or iPhone App	Positive
2	@swonderlin can not wait for #ipad 2 also. the...	iPad	Positive
3	@sxsw i hope this year's festival isn't as cra...	iPad or iPhone App	Negative
4	@sxtxstate great stuff on fri #sxsw: marissa m...	Google	Positive
5	@teachntech00 new ipad apps for #speechtherapy...	NaN	Neutral
6	#sxsw is just starting, #ctia is around the co...	Android	Positive
7	beautifully smart and simple idea rt @madebyma...	iPad or iPhone App	Positive

4.2.3 Combining compound nouns

A compound noun is two or more words that make a noun that are not referring to the same thing separately. For instance, *Apple* can mean the brand, *Store* can mean any store, but the compound noun *Apple Store* refers to something neither word refers to alone.

I'll get better results in this NLP analysis if I standardize the corpus's compound nouns using underscores for spaces. I'll do this before tokenization.

In [13]:
```
# This function searches the tweets for keys in a given dictionary and replace
# them with the associated value.
# df = The dataframe to search
# begin_replace = There are some tweets that begin with someone's last name,
#                 this requires separate coding to insert their first name
#                 before their last, so this dictionary will specify those
#                 compound nouns that need this treatment.
# replace_dict  = The dictionary with keys and values that turns more than one
#                 word into compound nouns with underscores.

def string_replace(dfone, begin_replace, replace_dict, to_print = True):
    counts = pd.DataFrame(columns = ['Replaced'])
    for n in range(len(dfone)):
        tweet = dfone.loc[n,'Tweet']
        for fnd, replce in zip(begin_replace.keys(), begin_replace.values()):
            if fnd in tweet[0:len(fnd)]:
                tweet = replce + tweet[len(fnd):]
                replced = fnd + ' / ' + replce
                counts.loc[len(counts),'Replaced'] = replced
                dfone.loc[n,'Tweet'] = ''+tweet
        for find, replace in zip(replace_dict.keys(), replace_dict.values()):
            if find in tweet:
                change = False
                first = tweet[0:len(find)]
                last = tweet[len(tweet) - len(find):]
                if (find in first):
                    change = True
                    tweet = replace + tweet[len(find):]
                    replaced = find + ' / ' + replace
                    counts.loc[len(counts),'Replaced'] = replaced
                if (find in last):
                    change = True
                    tweet = tweet[:len(tweet)-len(find)] + replace
                    replaced = find + ' / ' + replace
                    counts.loc[len(counts),'Replaced'] = replaced
                complete = 0
                while complete < len(tweet) - len(find):
                    for i in range(0, len(tweet) - len(find)):
                        if find in tweet[i:i+len(find)]:
                            change = True
                            first = tweet[:i]
                            last = tweet[i+len(find):]
                            tweet = first + replace + last
                            complete = 0
                            replaced = find + ' / ' + replace
                            counts.loc[len(counts),'Replaced'] = replaced
                            break
                        else:
                            complete += 1
                if change == True:
                    dfone.loc[n,'Tweet'] = ''+tweet
    if to_print == True:
        if counts.empty:
            print('No replacements made.')
        else:
            print(counts.value_counts())
    return dfone.copy()
```

I created this list by reading which words came up most often in the dataset I started with the most numerous words and went down the list in descending order. As I read certain words, I'd then investigate. Seeing the word Tyson led me to search for the word Tyson in the tweets, which led me to learn that Mike Tyson was at SXSW 2011. So his compound noun is mike_tyson.

```
In [14]: first_replace = {'mayer'                      : 'marissa_mayer',
                          'vinh'                       : 'khoi_vinh'}

         replacements =  {'apple store'                : 'apple_store',
                          'applestore'                 : 'apple_store',
                          'apple_stores'               : 'apple_store',
                          'app store'                  : 'app_store',
                          'appstore'                   : 'app_store',
                          'marissa mayer'              : 'marissa_mayer',
                          'marissa meyer'              : 'marissa_mayer',
                          'marissameyer'               : 'marissa_mayer',
                          'marissamayer'               : 'marissa_mayer',
                          'marisa meyer'               : 'marissa_mayer',
                          'marissa@mention'            : 'marissa_mayer',
                          'melissa mayer'              : 'marissa_mayer',
                          'merissa mayer'              : 'marissa_mayer',
                          'm mayer'                    : 'marissa_mayer',
                          'm.mayer'                    : 'marissa_mayer',
                          '-mayer'                     : 'marissa_mayer',
                          '#mayer'                     : '#marissa_mayer',
                          ' mayer'                     : 'marissa_mayer',
                          'marissa_mayers'             : 'marissa_mayer',
                          "marissa_mayer's"            : 'marissa_mayer',
                          'mentionmarissa_mayer'       : 'mention marissa_mayer',
                          '#mayer'                     : '#marissa_mayer',
                          "tim o'reilly"               : "tim_o_reilly",
                          'tim oreilly'                : "tim_o_reilly",
                          "tim_o'reilly's"             : "tim_o_reilly",
                          "tim_o'reillys"              : "tim_o_reilly",
                          "'re"                        : ' are',
                          'tim o areily'               : "tim_o_reilly",
                          'tim soo'                    : 'tim_soo',
                          'tim ferris'                 : 'tim_ferris',
                          'tim_ferriss'                : 'tim_ferris',
                          'tim wu'                     : 'tim_wu',
                          'matt mullenweg'             : 'matt_mullenweg',
                          'matt_mullenwegs'            : 'matt_mullenweg',
                          "matt_mullenweg's"           : 'matt_mullenweg',
                          'jonathan dahl'              : 'jonathan_dahl',
                          'mark belinsky'              : 'mark_belinsky',
                          'maggie mae'                 : 'maggie_mae',
                          'maggie may'                 : 'maggie_mae',
                          "maggie_mae's"               : 'maggie_mae',
                          "maggie_maes"                : 'maggie_mae',
                          "mike tyson"                 : 'mike_tyson',
                          "mike_tyson's"               : 'mike_tyson',
                          "mike_tysons"                : 'mike_tyson',
                          "tyson's"                    : 'mike_tyson',
                          'matt damon'                 : 'matt_damon',
                          'barry diller'               : 'barry_diller',
                          'nyt'                        : 'ny_times',
                          'new york times'             : 'ny_times',
                          'ny times'                   : 'ny_times',
                          "can't"                      : 'can not',
                          "won't"                      : 'would not',
                          "n't"                        : ' not',
                          "'ve"                        : ' have',
                          "'ll"                   : ' will',
                          'steve jobs'            : 'steve_jobs',
                          'pop up'                : 'pop_up',
                          'popup'                 : 'pop_up',
                          'pop-up'                : 'pop_up',
                          'pop shop'              : 'pop_up shop',
                          'pop-uitp'              : 'pop_up',
                          'pop- up'               : 'pop_up',
                          'pop-u‰û_'              : 'pop_up',
                          'pop-store'             : 'pop_up store',
                          'pop store'             : 'pop_up store',
                          'wi-fi'                 : 'wifi',
                          'wi fi'                 : 'wifi',
                          '.com'                  : 'com',
                          'dennis crowley'        : 'dennis_crowley',
                          ' crowley'              : 'dennis_crowley',
                          'i-phone'               : 'iphone',
                          'i-pad'                 : 'ipad',
                          'ipads'                 : 'ipad',
                          'iphones'               : 'iphone',
                          'ipad 2'                : 'ipad2',
                          'ipad_2'                : 'ipad2',
                          'ipad2s'                : 'ipad2',
                          'ipad 1'                : 'ipad1',
                          'ipad1s'                : 'ipad1',
                          'iphone app'            : 'iphone_app',
                          'ipad app'              : 'ipad_app',
                          'andoid'                : 'android',
                          'droid app'             : 'droid_app',
                          'iphone_app_store'      : 'iphone app_store',
                          'ipad_app_store'        : 'ipad app_store',
                          'droid_app_store'       : 'droid app_store',
                          'iphone_apps'           : 'iphone_app',
                          'ipad_apps'             : 'ipad_app',
                          'iphone_application'    : 'iphone application',
                          'droid_apps'            : 'droid_app',
                          ' droid_app'            : ' android_app',
                          'ûªll'                  : ' will',
                          "‰ûªs"                  : "'s",
                          "‰ûªre"                 : ' are',
                          "ûªt"                   : ' not',
                          'khoi vinh'             : 'khoi_vinh',
                          'jonathan ive'          : 'jonathan_ive',
                          'winsåêsxsw'            : 'wins sxsw',
                          'matt carlson'          : 'matt_carlson',
                          'matthew davis'         : 'matthew_davis',
                          'william patry'         : 'william_patry',
                          'josh williams'         : 'josh_williams',
                          'david foster'          : 'david_foster',
                          'david carr'            : 'david_carr',
                          'û_please'              : 'please',
                          'adam beckley'          : 'adam_beckley',
                          'paul adams'            : 'paul_adams',
                          'googled'               : 'google_verb_d',
                          'google_verb_d'         : 'googled_verb'}

         df = string_replace(df, first_replace, replacements)
```

```
Replaced
ipad 2 / ipad2                                    988
apple store / apple_store                         597
pop-up / pop_up                                   423
n't /  not                                        377
're /  are                                        269
iphone app / iphone_app                           251
popup / pop_up                                    222
marissa mayer / marissa_mayer                     178
ipad app / ipad_app                               157
pop up / pop_up                                   157
'll /  will                                       139
've /  have                                       117
.com / com                                         97
app store / app_store                              90
ipads / ipad                                       89
can't / can not                                    78
droid app / droid_app                              73
ipad2s / ipad2                                     47
ipad_apps / ipad_app                               43
nyt / ny_times                                     38
‰ûªs / 's                                          35
iphone_apps / iphone_app                           30
ipad 1 / ipad1                                     27
won't / would not                                  27
barry diller / barry_diller                        23
maggie mae / maggie_mae                            22
mark belinsky / mark_belinsky                      21
google_verb_d / googled_verb                       21
googled / google_verb_d                            21
maggie_mae's / maggie_mae                          16
mike tyson / mike_tyson                            16
 mayer / marissa_mayer                             16
mayer / marissa_mayer                              15
‰ûªre /  are                                       14
ûªll /  will                                       13
marissa_mayer's / marissa_mayer                    12
ûªt /  not                                         11
andoid / android                                   11
marissa meyer / marissa_mayer                      10
apple_stores / apple_store                         10
tim o'reilly / tim_o_reilly                         9
ipad_2 / ipad2                                      9
iphones / iphone                                    8
steve jobs / steve_jobs                             8
maggie_maes / maggie_mae                            8
khoi vinh / khoi_vinh                               7
new york times / ny_times                           7
droid_apps / droid_app                              7
iphone_app_store / iphone app_store                 6
wi-fi / wifi                                        6
mentionmarissa_mayer / mention marissa_mayer        6
applestore / apple_store                            6
dennis crowley / dennis_crowley                     6
 crowley / dennis_crowley                           5
 droid_app /  android_app                           4
pop- up / pop_up                                    4
```

```
marissameyer / marissa_mayer                     4
melissa mayer / marissa_mayer                    4
appstore / app_store                             3
winsåêsxsw / wins sxsw                           3
josh williams / josh_williams                    3
iphone_application / iphone application         3
william patry / william_patry                    3
matthew davis / matthew_davis                    3
tim ferris / tim_ferris                          2
tim soo / tim_soo                                2
pop-store / pop_up store                         2
-mayer / marissa_mayer                           2
paul adams / paul_adams                          2
ny times / ny_times                              2
maggie may / maggie_mae                          2
tim o areily / tim_o_reilly                      2
matt mullenweg / matt_mullenweg                  2
matt damon / matt_damon                          2
matt carlson / matt_carlson                      2
marissamayer / marissa_mayer                     2
jonathan dahl / jonathan_dahl                    2
jonathan ive / jonathan_ive                      2
marissa_mayers / marissa_mayer                   2
û_please / please                                2
vinh / khoi_vinh                                 1
tyson's / mike_tyson                             1
adam beckley / adam_beckley                      1
tim_ferriss / tim_ferris                         1
tim wu / tim_wu                                  1
tim oreilly / tim_o_reilly                       1
#mayer / #marissa_mayer                          1
pop store / pop_up store                         1
pop-u‰û_ / pop_up                                1
david carr / david_carr                          1
david foster / david_foster                      1
pop-uitp / pop_up                                1
marissa@mention / marissa_mayer                  1
pop shop / pop_up shop                           1
i-pad / ipad                                     1
i-phone / iphone                                 1
mike_tyson's / mike_tyson                        1
ipad1s / ipad1                                   1
merissa mayer / marissa_mayer                    1
m mayer / marissa_mayer                          1
m.mayer / marissa_mayer                          1
marisa meyer / marissa_mayer                     1
dtype: int64
```

4.2.4 Tokenize

The process of tokenizing the data is to turn each tweet into a list, with each word becomes a single item on the list. This will allow NLP methods to work their analytical magic.

In [15]:
```
# Creating the tokenizer
basic_token_pattern = r"(?u)\b\w\w+\b"
words_and_hashtags_token_pattern = r"(?u)\#?\b\w\w+\b"
hashtags_token_pattern = r"(?u)\#\b\w\w+\b"

tokenizer = RegexpTokenizer(basic_token_pattern)
hashtag_tokenizer = RegexpTokenizer(hashtags_token_pattern)
words_and_hashtags_tokenizer = RegexpTokenizer(words_and_hashtags_token_patter
```

In [16]:
```
# Creating a new column for the tokenized tweet
def tokenize_columns(dfone):
    dfone['Tokens'] = None
    dfone['Tokens With Hashtags'] = None
    dfone['Hashtags'] = None

    # Creating the token lists. One list will include tokens of each words, th
    # other will only include the words used as Hashtags.
    for i in range(len(dfone)):
        dfone['Tokens'][i] = tokenizer.tokenize(dfone['Tweet'][i])
        dfone['Tokens With Hashtags'
             ][i] = words_and_hashtags_tokenizer.tokenize(dfone['Tweet'][i])
        dfone['Hashtags'][i] = hashtag_tokenizer.tokenize(dfone['Tweet'][i])
    return dfone.copy()

df = tokenize_columns(df)
```

In [17]:
```
for i in df.index:
    if 'googled_verb' in df['Tokens'][i]:
        print(i)
        print(df['Tweet'][i])
```

```
594
ironic? i googled_verb the directions to @mention party and ended up walked 6
blocks in the wrong direction. time for bed i think. #sxsw
6163
rt @mention ironic? i googled_verb the directions to @mention party and ended
up walked 6 blocks in the wrong direction. time for bed i think. #sxsw
7293
#sxsw - downtown austin apple_store: congress &amp; west 6th street. (answere
d my own question the old fashion way... i googled_verb it).
```

4.3 Purging Retweets Identical to Included Originals

Multiple tweets included in the dataset are identical to each other besides one of them beginning with the words "rt" and "mention". We see so many begin with "rt" because that is Twitter shorthand for "retweet," which is an exact copy of someone else's tweet with the option to add a few words at the beginning. The @mention word is common because it follows the format of Twitter usernames, also known as Twitter handles. The dataset censors anyone's twitter handle, ex: @TwitterHandle, by replacing the handle with @mention. The reason "mention" is so often the second word is that "rt @mention" is the default beginning for retweeting a tweet, with "mention" being the original author's twitter handle.

In [18]:
```
# Top five first  and second words
temp_first_words = df.copy()
temp_first_words['First'] = temp_first_words['Tokens'].apply(lambda x:x[0])
temp_first_words['Second'] = temp_first_words['Tokens'].apply(lambda x:x[1])

temp_count_first_words = Counter(temp_first_words['First'])
temp_counts_first = pd.DataFrame(temp_count_first_words.most_common(5))
temp_counts_first.columns = ['Top Five First Words','First Words Count']

temp_count_first_words = Counter(temp_first_words['Second'])
temp_counts_second = pd.DataFrame(temp_count_first_words.most_common(5))
temp_counts_second.columns = ['Top Five Second Words','Second Words Count']
temp = pd.concat([temp_counts_first, temp_counts_second], axis=1)
temp.index = np.arange(1, len(temp) + 1)
temp.style.background_gradient(cmap='Blues')
```

Out[18]:

	Top Five First Words	First Words Count	Top Five Second Words	Second Words Count
1	rt	1990	mention	2276
2	mention	680	to	352
3	google	411	sxsw	211
4	sxsw	289	the	209
5	apple	250	is	181

1990 retweets and then 2276 that have mention as the second word. That's a lot.

If two tweets are identical in every way except for the first or second words, then they are not worth analyzing twice. I'll get rid of all retweets that have the original tweet in the dataset. These retweets do not add value to the model.

The process of checking for retweets can be done multiple times with slightly different methodology. In the cells below, I use three methodologies:

The cells below check for matches after...:

1. Deleting the initial "rt" and then every subsequent repeated instance of "mention" from the retweet.
2. Deleting every initial repeated instance of both "rt" and "mention" from the retweet.
3. Deleting the first one through six words from the retweet.

To account for these slight differences, I created four functions to make this process easily repeatable. The parameters allow me to change which methodology I use.

The four functions below perform the following four actions:

1. Create a Dataframe with nothing but the retweets, including the tokenized versions.
2. Trim the retweets by specified parameters.
3. Check to see if there is an exact match for each trimmed retweet in the original dataset.
4. Purge the retweets.

The final step, deleting the retweet from the dataset, is wrapped into another function that performs all of these steps at once.

4.3.1 The Retweet Purge Functions

Now, for the four functions described above and an additional small function that just tallies everything in a list.

In [19]:
```
# Creating a df of nothing but the retweets
def retweet_trim_one(dfone):
    for i in dfone.index:
        twt = dfone.loc[i, 'Tokens']
        # Twitter parlance: rt = retweet
        # Dropping all tweets that don't start in rt
        if 'rt' != twt[0]:
            dfone.drop(index=i, inplace=True)
    return dfone
```

In [20]:
```
# Creates a dictionary of specified keys with a count as the values, and then
# sorts the list from greatest to least in dictionary form.
def tally(lst):
    tally_dict = {}
    for item in lst:
        if item not in tally_dict:
            tally_dict[item] = 1
        else:
            tally_dict[item] += 1
    # Sorting the dictionaries
    count = sorted(tally_dict.items(), key=lambda x: x[1], reverse=True)
    return count
```

In [21]:
```
# Trimming the terms rt and mention from the beginning of the tweet, counting
# the first and second words of the remaining tweets, and determining which
# tweets to drop.

def retweet_trim_two(dftwo, drop_first_words=['skip'], drop_num=1):
    # Creating a column to put the trimmed retweet
    dftwo['trimmed_rt'] = 'Blank'
    dftwo['Match'] = False

    # I want to test the data to see if there's anything that stands out as
    # repetitive in the first or second words of the newly trimmed tweets.
    # Creating a list of the first and second words of each trimmed tweet.
    firstword_lst = []
    secondword_lst = []

    # Starting a list to drop any retweets.
    retweets_to_drop = []

    for i in dftwo.index:
        # Making the trimmed variable, the tokens of the next tweet to edit
        trimmed = dftwo.loc[i, 'Tokens']
        # Removing the set number of dropped tokens.
        # Default is just the first token, "rt."
        trimmed = trimmed[drop_num:]
        if len(trimmed) == 0:
            if drop_num == 1:
                retweets_to_drop.append(i)
                dftwo.loc[i,'Match'] = True
            continue
        # Several retweets include 'mention' after the rt, so I'll trim that.
        # This while loop quits dropping words with the first token that isn't
        # in the drop list specified by parameter.
        if drop_first_words != ['skip']:
            while trimmed[0] in drop_first_words:
                trimmed = trimmed[1:]
                # Some tweets contained only "rt" and "mention," and are now
                # empty."
                # Adding any empty token lists to the retweets_to_drop list.
                if len(trimmed) == 0:
                    if drop_num == 1:
                        retweets_to_drop.append(i)
                        dftwo.loc[i,'Match'] = True
                    break
        if len(trimmed) == 0:
            continue

        # Adding first and second words of the trimmed tweet to the lists.
        firstword_lst.append(trimmed[0])
        secondword_lst.append(trimmed[1])

        # Replacing the trimmed tokens to a new column, "trimmed_rt".
        dftwo.at[i,'trimmed_rt'] = trimmed

    # I want to test the data to see if there's anything that stands out as
    # repetitive in the first word of the newly trimmed tweets.
    # Creating a dictionary to count the first word in each trimmed tweet.
    firstword_count = tally(firstword_lst)
    secondword_count = tally(secondword_lst)

    return dftwo, firstword_count, secondword_count, retweets_to_drop
```

In [22]:
```
# Checking to see if any of theses newly trimmed retweets are exact matches of
# Original tweets in the dataset.


def retweet_trim_three(dfthree, dforg, retweets_to_drop):
    original_tweets = []

    for i in dforg.index:
        original_tweets.append(dforg.loc[i, 'Tokens'])

    for i in dfthree.index:
        retweet = dfthree.loc[i, 'trimmed_rt']
        if retweet in original_tweets:
            dfthree.loc[i, 'Match'] = True
            retweets_to_drop.append(i)

    return dfthree, retweets_to_drop
```

In [23]:
```
# Creating a single function that runs all of the previous functions but allow
# a list parameter to choose what words are filtered out of the first words.
# Also allows for deleting the duplicate tweets in one function.
def retweet_trim_all(retweets, word_list=['skip'], delete=True, drop=1):
    retweets = retweet_trim_one(retweets)
    retweets, fw, sw, rtd = retweet_trim_two(retweets, word_list, drop)
    retweets, rtd = retweet_trim_three(retweets, df, rtd)
    # Dropping the tweets.
    if delete == False:
        print(retweets['Match'].value_counts())
        return retweets.loc[retweets['Match'] == True]
    df.drop(index=rtd, inplace=True)
    # df.reset_index(drop=True, inplace=True)
    return retweets['Match'].value_counts()
```

4.3.2 Purge Retweets: Deleting Initial Words "rt" & "mention"

As discussed above, these cells delete any retweet that matches an original tweet once the initial "rt" and every subsequent repeated "mention" are removed from the beginning of the retweet.

In [24]:
```
# Creating a copy of the dataframe for editing.
retweets = df[['Tweet', 'Tokens']].copy()

# Purging original tweets from the new copy, so that all that is left
# are retweets.
retweets = retweet_trim_one(retweets)

# Creating an initial count of all retweets
retweet_count = len(retweets)

# Previewing the retweets.
retweets.head()
```

Out[24]:

	Tweet	Tokens
24	rt @laurieshook: i'm looking forward to the #s...	[rt, laurieshook, looking, forward, to, the, s...
25	rt haha, awesomely rad ipad_app by @madebymany...	[rt, haha, awesomely, rad, ipad_app, by, madeb...
199	rt ' it's 4 p.m. and the #ipad2 line at the ap...	[rt, it, and, the, ipad2, line, at, the, apple...
867	rt hiring marketers, designers, creatives, soc...	[rt, hiring, marketers, designers, creatives, ...
1054	l.a.m.e. rt @mention &quot;...by the law of a...	[rt, mention, quot, by, the, law, of, averages...

In [25]:
```
# Trimming the retweets of the first "rt" and the subsequent word "mention."
retweets, fw, sw, rtd = retweet_trim_two(retweets, ['mention'])
```

In [26]:
```
# Checking to see if any of the trimmed retweets match any original tweets in
# the dataset.
retweets, rtd = retweet_trim_three(retweets, df, rtd)

# Variable that tells me how many matches I've found.
initial_purge = sum(retweets['Match'])

# Variable that tells me how many retweets di not find a match.
initial_remain = sum(~retweets['Match'])

# If a the Match column value is True, then the retweet matches an original
# tweet and will be deleted in the next cell
retweets['Match'].value_counts()
```

Out[26]:
```
True     1463
False     527
Name: Match, dtype: int64
```

I will now remove all of these tweets from the dataset.

In [27]:
```
# Dropping the duplicate retweets from the original dataset.
df.drop(index=rtd, inplace=True)
```

4.3.3 Purge Retweets: Removing all "rt" and "mention" Instances

For reasons I don't understand, some of the tweets were retweets of retweets. So the first two words are not "rt" and "mention", but "rt" and "rt". So I'll check if there were even more retweets of originals in the dataset.

I checked for matches after I removed all initial subsequent instances of both "rt" and "mention" from the tweets to see if any other matches surfaced. Then I reran the method for just initial repeated "rt". Together, they found four matches and I deleted all four.

In [28]:
```python
# Before each run through these functions, I need to make a new copy of the
# retweets DafaFrame.
retweets = df[['Tweet', 'Tokens']].copy()

# Checking for multiple subsequent copies of both "rt" and "mention."
roundtwo = retweet_trim_all(retweets, ['rt', 'mention'])
deleted_two = roundtwo.loc[True]
roundtwo
```

Out[28]:
```
False    526
True       1
Name: Match, dtype: int64
```

In [29]:
```python
# Checking for multiple subsequent copies of just "rt."
retweets = df[['Tweet', 'Tokens']].copy()
roundtwo = retweet_trim_all(retweets, ['rt'])
deleted_two += roundtwo.loc[True]
roundtwo
```

Out[29]:
```
False    523
True       3
Name: Match, dtype: int64
```

4.3.4 Purge Retweets: Removing A Set Number of Words

That takes care of all the tweets that are duplicates save for "rt" and "mention" at the beginning. However, the whole dataset does not follow the same convention of censoring the Twitter handle with @mention. Several tweets do contain the @ symbol followed by a twitter handle. Therefore several tweets begin "rt @username" .

This means that I should try to remove the first two words from every tweet that begins with "rt" and check those for matches with the original data. When I added a parameter to my functions to do just that, it dawned on me that there might be other reasons to try comparing the retweets after cutting the more than the first two words. There could be a reason where deletion of the first five words will result in a match with an original tweet. I decided to preview what matches show up when removing the first words sequentially from one to six words.

Below is that preview.

In [30]:
```
pd.options.display.max_colwidth = 1000
def retweet_preview(num):
    retweets = df.loc[:,['Tweet', 'Tokens']].copy()
    preview = retweet_trim_all(retweets, delete = False, drop = num)
    if preview.empty:
        print('No matches found.')
        return None
    return preview.head()

retweet_preview(1)
```

```
False    523
Name: Match, dtype: int64
No matches found.
```

In [31]:
```
retweet_preview(2)
```

```
False    457
True      66
Name: Match, dtype: int64
```

Out[31]:

	Tweet	Tokens	trimmed_rt	Match
24	rt @laurieshook: i'm looking forward to the #smcdallas pre #sxsw party wed., and hoping i will win an #ipad resulting from my shameless promotion. #chevysmc	[rt, laurieshook, looking, forward, to, the, smcdallas, pre, sxsw, party, wed, and, hoping, will, win, an, ipad, resulting, from, my, shameless, promotion, chevysmc]	[looking, forward, to, the, smcdallas, pre, sxsw, party, wed, and, hoping, will, win, an, ipad, resulting, from, my, shameless, promotion, chevysmc]	True
2112	rt stevecase: google to launch major new social network called circles {link} #sxsw	[rt, stevecase, google, to, launch, major, new, social, network, called, circles, link, sxsw]	[google, to, launch, major, new, social, network, called, circles, link, sxsw]	True
2932	;-) rt @mention @mention the geeks need somewhere downtown to line up to get the ipad2. i will be dropping by. #sxsw	[rt, mention, mention, the, geeks, need, somewhere, downtown, to, line, up, to, get, the, ipad2, will, be, dropping, by, sxsw]	[mention, the, geeks, need, somewhere, downtown, to, line, up, to, get, the, ipad2, will, be, dropping, by, sxsw]	True
5043	rt @mention . @mention double fisting at the keynote #sxsw #apple {link}	[rt, mention, mention, double, fisting, at, the, keynote, sxsw, apple, link]	[mention, double, fisting, at, the, keynote, sxsw, apple, link]	True
5046	rt @mention .@mention &quot;google launched checkins a month ago.&quot; check ins are ok, but check outs are the future. #sxsw #bizzy	[rt, mention, mention, quot, google, launched, checkins, month, ago, quot, check, ins, are, ok, but, check, outs, are, the, future, sxsw, bizzy]	[mention, quot, google, launched, checkins, month, ago, quot, check, ins, are, ok, but, check, outs, are, the, future, sxsw, bizzy]	True

In [32]:
```
retweet_preview(3)
```

```
False    519
True       4
Name: Match, dtype: int64
```

Out[32]:

	Tweet	Tokens	trimmed_rt	Match
3776	rt %‰ûï@mention hoot!! *new blog post:* #hootsuite mobile for #sxsw ~ updates for iphone, #blackberry &amp; #android - {link}	[rt, ûï, mention, hoot, new, blog, post, hootsuite, mobile, for, sxsw, updates, for, iphone, blackberry, amp, android, link]	[hoot, new, blog, post, hootsuite, mobile, for, sxsw, updates, for, iphone, blackberry, amp, android, link]	True
4313	rt &gt; @mention guy gets tattoo at sxsw so he could win a free ipad2 {link} #sxsw #tattoo #ipad #internet	[rt, gt, mention, guy, gets, tattoo, at, sxsw, so, he, could, win, free, ipad2, link, sxsw, tattoo, ipad, internet]	[guy, gets, tattoo, at, sxsw, so, he, could, win, free, ipad2, link, sxsw, tattoo, ipad, internet]	True
5097	rt @mention @mention @mention at #sxsw: &quot;apple comes up with cool technology no one's ever heard of because they do not go to conferences&quot;	[rt, mention, mention, mention, at, sxsw, quot, apple, comes, up, with, cool, technology, no, one, ever, heard, of, because, they, do, not, go, to, conferences, quot]	[mention, at, sxsw, quot, apple, comes, up, with, cool, technology, no, one, ever, heard, of, because, they, do, not, go, to, conferences, quot]	True
5748	rt%‰ûï@mention fret not, iphone owners. apple to open temporary store at #sxsw. {link} via @mention	[rt, ûï, mention, fret, not, iphone, owners, apple, to, open, temporary, store, at, sxsw, link, via, mention]	[fret, not, iphone, owners, apple, to, open, temporary, store, at, sxsw, link, via, mention]	True

In [33]:
```
retweet_preview(4)
```

```
False    521
True       2
Name: Match, dtype: int64
```

Out[33]:

	Tweet	Tokens	trimmed_rt	Match
4451	rt@mention t's not a rumor: apple is opening up a temporary store in downtown austin for #sxsw and the ipad2 launch {link}	[rt, mention, not, rumor, apple, is, opening, up, temporary, store, in, downtown, austin, for, sxsw, and, the, ipad2, launch, link]	[apple, is, opening, up, temporary, store, in, downtown, austin, for, sxsw, and, the, ipad2, launch, link]	True
6847	rt @mention via @mention still a big line outside of apple's pop_up shop, 3 days after ipad's debut. #sxsw	[rt, mention, via, mention, still, big, line, outside, of, apple, pop_up, shop, days, after, ipad, debut, sxsw]	[still, big, line, outside, of, apple, pop_up, shop, days, after, ipad, debut, sxsw]	True

In [34]:
```
retweet_preview(5)
```

```
False    521
True       2
Name: Match, dtype: int64
```

Out[34]:

	Tweet	Tokens	trimmed_rt	Match
5108	rt @mention @mention 2 at #sxsw? apple is opening a pop_up store in austin for sxsw {link}	[rt, mention, mention, at, sxsw, apple, is, opening, pop_up, store, in, austin, for, sxsw, link]	[apple, is, opening, pop_up, store, in, austin, for, sxsw, link]	True
6780	rt @mention they are everywhere: it's just crazy to look around sxsw and realize last year no one had an ipad. #sxsw	[rt, mention, they, are, everywhere, it, just, crazy, to, look, around, sxsw, and, realize, last, year, no, one, had, an, ipad, sxsw]	[it, just, crazy, to, look, around, sxsw, and, realize, last, year, no, one, had, an, ipad, sxsw]	True

In [35]:
```
retweet_preview(6)
```

```
False    523
Name: Match, dtype: int64
No matches found.
```

Removing all the retweets shown above would not delete any significant words that would alter modeling. No matches are found by removing one and six words, so I'll delete any match found after removing the first two through five words.

In [36]:
```
# Removing matches that occur when removing the first two through the first
# five words

# Variable to count the number of words deleted
deleted_three = 0

# Removing the second through fifth words, checking for matches, and deleting
# the matched retweets from the dataset.
# also, I'm keeping track of how many I delete this round.
for i in range(2, 6):
    retweets = df[['Tweet', 'Tokens']].copy()
    retweet_trim_all(retweets, drop=i)
    deleted_three += len(retweets.loc[retweets['Match'] == True])
```

4.3.5 Purge Retweets: Summary

In [37]:
```python
tot_dataset = (len(df_raw)-1)
tot_del = initial_purge + deleted_two + deleted_three
per_det = round(tot_del / tot_dataset * 10000) / 100
new_tot = tot_dataset - tot_del
retweets_remain = retweet_count - tot_del

# Putting the process above into words.
print('The initial purge deleted', initial_purge,
      'retweets that already existed in the dataset')
print('once all instances of both "rt" and "mention" were deleted from the')
print('beginning of each retweet.')
print()

print('Then, removing all instances of "rt" and "mention" found',
     deleted_two, 'additional matches.')
print('These', deleted_two, 'were deleted for a total of',
      initial_purge + deleted_two, 'deleted tweets')
print()

print('After this inital purge, there were', initial_remain - deleted_two,
      'retweets remaining.')
print('The above cells found, and deleted, an additional', deleted_three,
      'retweets that matched')
print('original tweets in the data set.')
print()
print('In total, I deleted', tot_del, 'out of', retweet_count, 'retweets.',
     'Only', retweets_remain,'retweets remain.')
print()
print('Of the', tot_dataset, 'tweets, the', tot_del, 'deleted tweets',
      'represent {}% of the entire'.format(per_det))
print('dataset.')
print()
print('There are now', new_tot, 'tweets remaining in the dataset.')
```

```
The initial purge deleted 1463 retweets that already existed in the dataset
once all instances of both "rt" and "mention" were deleted from the
beginning of each retweet.

Then, removing all instances of "rt" and "mention" found 4 additional matche
s.
These 4 were deleted for a total of 1467 deleted tweets

After this inital purge, there were 523 retweets remaining.
The above cells found, and deleted, an additional 74 retweets that matched
original tweets in the data set.

In total, I deleted 1541 out of 1990 retweets. Only 449 retweets remain.

Of the 9092 tweets, the 1541 deleted tweets represent 16.95% of the entire
dataset.

There are now 7551 tweets remaining in the dataset.
```

A good rule of thumb for pruning datasets is not to delete more than 10% of the data, but I've already deleted more than that. However, I would argue that I have not. I have deleted duplicate data with a few additional words that provide no additional context. These tweets do not provide a benefit for the purposes of modeling. I will see how close to the 10% rule I get from this point forward. I will try to avoid deleting an additional 10% of the data from what I've already deleted.

The retweet purge is now complete. I'll now move on to categorizing the data.

4.4 Identifying Company

Now that so many duplicates of data are gone, I can begin identifying each tweet as one of my twelve different categories. Again, those categories are:

1. Company: Google or Apple
2. Aspect of Company: Brand or Product
3. Emotion: Positive, Negative, or Neutral

First I'll start with categorizing these by Company.

4.4.1 Gleaning Keywords From The Tweets

As I observed earlier, the products labels are not reliable because they are not included in 63.8% of the data. I'm going to have to scrub the tweets for keywords that will tell me what company they refer to, and then compare it to the original product column.

For starters, here is a summary of the given product information:

In [38]:
```
# Counting the number of rows that have products labeled:
products_labeled = len(df.loc[df['Product'].notna()])
print('Of the', len(df), 'remaining tweets',
      len(df) - products_labeled,
      'do not have a label in the Products column.')

df['Product'].value_counts()
```

```
Of the 7551 remaining tweets 4756 do not have a label in the Products column.
```

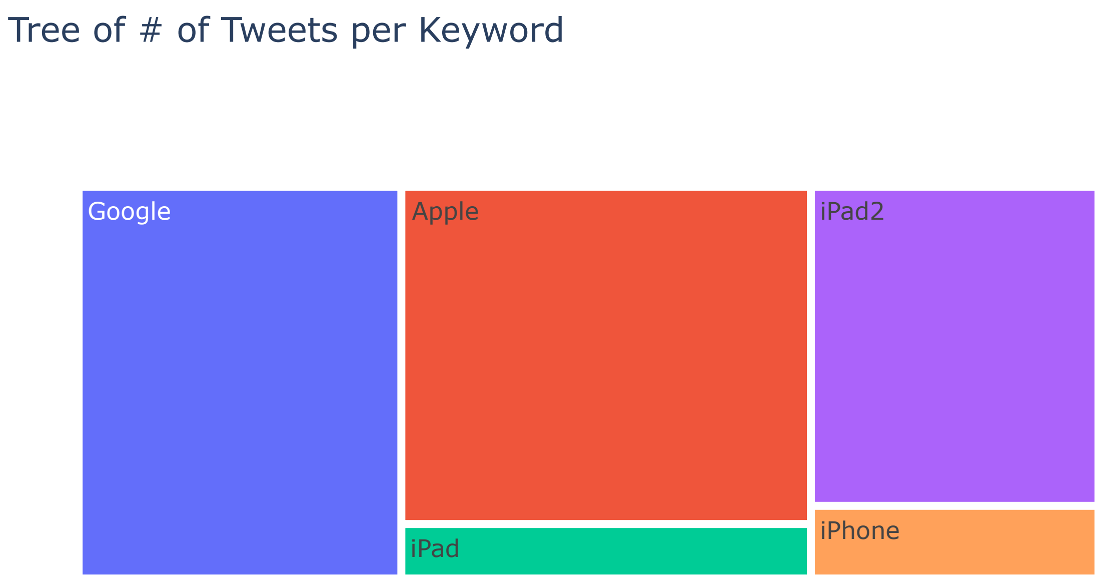
Out[38]:
```
iPad                               821
Apple                              561
iPad or iPhone App                 384
Google                             351
iPhone                             266
Other Google product or service    235
Android App                         75
Android                             71
Other Apple product or service      31
Name: Product, dtype: int64
```

To fill in the rest, I'm going to do a word search for different products these two companies make, Apple and Google. I'll use this keyword search to identify which company the tweets discuss, Apple or Google. Here are the keywords that I'll use for each company:

In [39]:
```
keywords_apple = ['Apple', 'iPad', 'iPad1', 'iPad2', 'iPhone', 'iTunes',
                  'iPhone5', 'iTune', 'Laptop', 'Apple_Store', 'App_Store',
                  'iOS', 'Macbook', 'iPad_App', 'iPhone_App', 'Pop_Up', 'iMac'

# Google did not make laptops in March 2011, so I'll assume all the laptops
# are Apple.

keywords_google = ['Google', 'Marissa_Mayer', 'MarissaGoogle', 'Googlecom',
                  'Android', 'Droid', 'Circles', 'Blogger', 'Chrome',
                  'Samsung', 'Galaxy', 'Maps', 'GoogleDoodle', 'ChromeOS',
                  'Googlecircles', 'Googletv', 'Android_App']
```

In [40]:
```
# Keywords that I'll use to identify what product the tweet discusses.

def keyword_labels(dfone, kw_apple, kw_google):
    kw_all = kw_apple + kw_google

    # A list of index numbers for tweets that don't include any of the keyword
    no_kw = []

    # Create a column for each keyword, to be purged later.
    for word in kw_all:
        dfone[word] = False

    # Labeling each tweet with what keywords they include.
    # Checks for the same word and the word with a hashtag.
    # The column will be labeled 'True' whether or not there's a hashtag.
    for i in dfone.index:
        for word in kw_all:
            if (word.lower() in dfone['Tokens'][i]) | (
                '#' + word.lower() in dfone['Tokens'][i]):
                dfone.loc[i, word] = True

        # appending no_kw with the index number of all rows
        # without any keywords.
        if sum(dfone.loc[i, kw_all].values) == 0:
            no_kw.append(i)

    return dfone.copy(), no_kw

df, nokeywords = keyword_labels(df, keywords_apple, keywords_google)
```

In [41]:
```
keywords_all = keywords_apple + keywords_google
kw_count = df[keywords_all].copy()
kw_count
kw_list = []
for i in kw_count.index:
    for col in kw_count:
        if kw_count[col][i] == True:
            kw_list.append(col)

temp_count = Counter(kw_list)
temp_kw = pd.DataFrame(temp_count.most_common(len(keywords_all)))
temp_kw.columns = ['Keyword','Count']
# print('Number of Tweets with included Keyword:')
# temp_kw.style.background_gradient(cmap='Blues')

fig = px.treemap(temp_kw, path=['Keyword'],
                 values='Count',
                 title='Tree of # of Tweets per Keyword')
fig.show()
```



Observations:

1. The most popular of all keywords is Google, followed by Apple.

2. After the top two words, the keywords that are most numerous are things the customers can purchase, or where they can purchase. "Circles" is the one to this, as it is about a future unknown product rather than currently available products.

In [42]:
```
keyword_tally = []
for col in keywords_all:
    for i in df.index:
        if df.loc[i,col]:
            keyword_tally.append(col)
keyword_count = tally(keyword_tally)
#    print(col + ':', df[col].sum())
print("There are {} tweets that don't have any of the above words".format(
    len(nokeywords)))
```

```
There are 558 tweets that don't have any of the above words
```

4.4.2 Identifying The Company By the Keywords

Now to identify these tweets as discussing either Apple or Google. I'll label the rows as either "Apple", "Google" or "Both" based on the key words I already gleaned from the tweets.

In [43]:
```
# This cell labels each row as "Apple", "Google", "Both" or "Unknown"
def company_organization(dfone, key_apple, key_google):
    dfone['Company'] = 'Blank'
    for i in dfone.index:
        # Checking for "Apple"
        for col in key_apple:
            if dfone[col][i]:
                dfone.loc[i, 'Company'] = 'Apple'
                # Checking for both Apple and Google
                for goog in key_google:
                    if dfone[goog][i]:
                        dfone.loc[i, 'Company'] = 'Both'
                        break
                break
        # Checking for "Google"
        if dfone.loc[i, 'Company'] not in ['Apple', 'Both']:
            for goog in key_google:
                if dfone[goog][i]:
                    dfone.loc[i, 'Company'] = 'Google'
                    break
        # Checking for Unknowns
        if dfone.loc[i, 'Company'] not in ['Apple', 'Both', 'Google']:
            dfone.loc[i, 'Company'] = 'Unknown'
    return dfone.copy()

df = company_organization(df, keywords_apple, keywords_google)

print(df['Company'].value_counts())
```

```
Apple      4469
Google     2267
Unknown     558
Both        257
Name: Company, dtype: int64
```

In [44]:
```
num_both = len(df.loc[df['Company'] == 'Both'])
print(
    'Out of', len(df), 'tweets,', num_both,
    'are labeled as "Both", {}% of the'.format(
        round((num_both / len(df)) * 10000) / 100), 'remaining data.')
```

```
Out of 7551 tweets, 257 are labeled as "Both", 3.4% of the remaining data.
```

There are so few that are labeled as "Both", I think I'll just drop them all.

In [45]:
```
df.drop(index=df.loc[df['Company'] == 'Both'].index, inplace=True)
df['Company'].value_counts()
```

Out[45]:
```
Apple      4469
Google     2267
Unknown     558
Name: Company, dtype: int64
```

4.4.3 Comparing the Product Column To The Company Column

I think it's possible that some of the tweets that don't have any of the keywords do have a value in the original product column. If so, I can identify which company they address via the provided product column rather than the keywords.

First, I'm going to check if the product labels match my keyword search for company. To do this, I'll create a new column, "Pd-Cp Match" which will include the company name gleaned solely from the Product column.

In [46]:
```
# Product column values that are Apple products.
Apple = [
    'iPad', 'Apple', 'iPad or iPhone App', 'iPhone',
    'Other Apple product or service'
]
# Product column values that are Google products.
Google = [
    'Google', 'Other Google product or service', 'Android App', 'Android'
]
```

In [47]:
```
# Creating the "Company by Product" column and populating it based on the
# Product column
def company_by_product(dfone):
    dfone['Company by Product'] = dfone['Product'].map(lambda x: 'Apple'
                                                       if x in Apple else np.nan)
    possgoog = dfone['Company by Product'] != 'Apple'

    dfone.loc[possgoog,
              'Company by Product'] = dfone.loc[possgoog, 'Product'].map(
        lambda x: 'Google' if x in Google else np.nan)
    return dfone.copy()

df = company_by_product(df)

df.loc[0:5, ['Tweet', 'Product', 'Company by Product']]
```

Out[47]:

	Tweet	Product	Company by Product
0	.@wesley83 i have a 3g iphone. after 3 hrs tweeting at #rise_austin, it was dead! i need to upgrade. plugin stations at #sxsw.	iPhone	Apple
1	@jessedee know about @fludapp ? awesome ipad/iphone_app that you will likely appreciate for its design. also, they are giving free ts at #sxsw	iPad or iPhone App	Apple
2	@swonderlin can not wait for #ipad2 also. they should sale them down at #sxsw.	iPad	Apple
3	@sxsw i hope this year's festival is not as crashy as this year's iphone_app. #sxsw	iPad or iPhone App	Apple
4	@sxtxstate great stuff on fri #sxsw: marissa_mayer (google), tim_o_reilly (tech books/conferences) &amp; matt_mullenweg (wordpress)	Google	Google
5	@teachntech00 new ipad_app for #speechtherapy and communication are showcased at the #sxsw conference http://ht.ly/49n4m #iear #edchat #asd	NaN	NaN

In [48]:
```
# Checking which rows don't have matching values in columns "Company" and
# "Company by Product"
def no_match(dfone):
    dfone['Pd-Cp Match'] = "Match"
    for i in dfone.index:
        by_keyword = dfone.loc[i, 'Company']
        by_product = dfone.loc[i, 'Company by Product']
        if by_keyword != by_product:
            dfone.loc[i, 'Pd-Cp Match'] = str([by_keyword,
                                                by_product])[1:-1].replace("'", "")
    return dfone.copy()

df = no_match(df)

df['Pd-Cp Match'].value_counts()
```

Out[48]:
```
Match              2703
Apple, nan         2470
Google, nan        1561
Unknown, nan        548
Unknown, Apple       10
Google, Apple         2
Name: Pd-Cp Match, dtype: int64
```

Most of the cells defining the company between *Company* and *Pd-Cp Match* columns either were equal or didn't have data in the second column to match. None of those situations concerns me.

There are two tweets that the keyword analysis labeled as Google and given Product column labeled as "Apple." I'll review these.

In [49]:
```
df.loc[(df['Pd-Cp Match'] == 'Google, Apple') |
       (df['Pd-Cp Match'] == 'Apple, Google') ,
       ['Company', 'Tweet', 'Pd-Cp Match']]
```

Out[49]:

	Company	Tweet	Pd-Cp Match
222	Google	just read about #groupme at #sxsw...sounds like an incredible app. is it available for android phones yet?	Google, Apple
2798	Google	several years too late? i think the trend of social apps is over... @mention #sxsw #google #circles #conversation @mention	Google, Apple

The first tweet in the set above, 222, clearly refers to both Apple and Google. 2798 is exclusively Google, and is labeled as such under the Company column. Therefore, I'll leave it as is.

I'll delete 222 and leave 2795 in place, but change *Pd-Cp Match* to 'Match'.

In [50]:
```
def mismatch(dfone):
    dfone.loc[2798,'Pd-Cp Match'] = 'Match'
    dfone.loc[2798, 'Company by Product'] = 'Google'
    return dfone.copy()

df = mismatch(df)

# Checking if it deleted the right one
print(df['Pd-Cp Match'].value_counts())

mismatch_products = df.loc[(
    df['Pd-Cp Match'] == 'Google, Apple')].index
df.drop(index = mismatch_products, inplace=True)

df['Pd-Cp Match'].value_counts()
```

```
Match              2704
Apple, nan         2470
Google, nan        1561
Unknown, nan        548
Unknown, Apple       10
Google, Apple         1
Name: Pd-Cp Match, dtype: int64
```

Out[50]:
```
Match              2704
Apple, nan         2470
Google, nan        1561
Unknown, nan        548
Unknown, Apple       10
Name: Pd-Cp Match, dtype: int64
```

4.4.4 Tweets Without Keywords, with a *Product* Column Value

There are a few tweets that have no keywords, but were given labels in the given *Products* column. I need to review these to see if they are useful.

In [51]:
```
# Creating a DataFrame of just the tweets that don't have any of the key words
def no_keywords(dfone, nkw):
    cut = dfone.copy()
    for i in dfone.index:
        if i not in nkw:
            cut.drop(index=i, inplace=True)
    return cut.copy(), nkw

cut_tweets, nokeywords = no_keywords(df, nokeywords)

# Checking if any of these tweets without the key words have a product labeled
cut_tweets['Product'].value_counts()
```

Out[51]:
```
iPad or iPhone App    10
Name: Product, dtype: int64
```

I need to review these tweets that have a value in the *Product* column, but don't have any

In [52]:
```
# Printing the tweets that don't have any of the keywords, but do have a given
# label in the Product column. Also, saving their index numbers in a list titl
# apple_tweets

def tweets_to_edit(nkw, cut, will_print = True):
    tweets = []
    for i in nkw:
        if cut.loc[i, 'Product'] in ['iPad or iPhone App', 'iPhone', 'Apple']:
            tweets.append(i)
    if will_print == True:
        to_print = cut.loc[tweets,['Tweet','Product']].copy()
        return tweets, to_print
    return tweets
apple_tweets, printing = tweets_to_edit(nokeywords, cut_tweets)
printing
```

Out[52]:

	Tweet	Product
405	if you are coming to #sxsw be sure to download the sxsw go app, it has all the schedules and locations of the events. {link}	iPad or iPhone App
1166	check out the new @mention app {link} - this is gonna be huge next week at #sxsw and beyond.	iPad or iPhone App
1683	@mention #hollergram app is killing it at #sxsw {link}	iPad or iPhone App
1917	want to make #sxsw film more fun? download #filmaster app {link} check-in to screenings &amp; get private recommendations!	iPad or iPhone App
2344	@mention check it. rt @mention #sxsw free app festival explorer: find the bands you want to see from your music tastes {link}	iPad or iPhone App
4213	just downloaded the asddieu app in preparation for #sxsw. pumped. you need to check it out {link} #app #events #networking	iPad or iPhone App
4552	this is one of the three best apps we have seen at #sxsw {link}	iPad or iPhone App
7007	@mention one of the must have apps to survive #sxsw {link} join our #sxswbarcrawl during #sxswi	iPad or iPhone App
7619	‰ûïfoursquare for bands&quot; just in time for #sxsw {link}	iPad or iPhone App
8188	woohoo! rt @mention @mention #hollergram app is killing it at #sxsw {link}	iPad or iPhone App

Seeing as all of these are talking about apps, it is reasonable to say their labeling in the *Product* column is correct.

I'll identify all of these tweets as Apple, matching between product/company, and finally mark them as True in the *iPhone_Apps* column.

In [53]:
```
# This function hand labels a few tweets that have no keywords, but are clearl
# tweets about Apple apps.

def apple_tweet_edit(dfone, nkw, tweets):
    for i in tweets:
        nkw.remove(i)
        dfone.loc[i, 'Company'] = 'Apple'
        dfone.loc[i, 'Pd-Cp Match'] = 'Match'
        dfone.loc[i, 'iPhone_App'] = True
    return dfone.copy(), nkw

df, nokeywords = apple_tweet_edit(df, nokeywords, apple_tweets)

df.loc[apple_tweets, ['Tweet', 'iPhone_App', 'Company', 'Pd-Cp Match']]
```

Out[53]:

	Tweet	iPhone_App	Company	Pd-Cp Match
405	if you are coming to #sxsw be sure to download the sxsw go app, it has all the schedules and locations of the events. {link}	True	Apple	Match
1166	check out the new @mention app {link} - this is gonna be huge next week at #sxsw and beyond.	True	Apple	Match
1683	@mention #hollergram app is killing it at #sxsw {link}	True	Apple	Match
1917	want to make #sxsw film more fun? download #filmaster app {link} check-in to screenings &amp; get private recommendations!	True	Apple	Match
2344	@mention check it. rt @mention #sxsw free app festival explorer: find the bands you want to see from your music tastes {link}	True	Apple	Match
4213	just downloaded the asddieu app in preparation for #sxsw. pumped. you need to check it out {link} #app #events #networking	True	Apple	Match
4552	this is one of the three best apps we have seen at #sxsw {link}	True	Apple	Match
7007	@mention one of the must have apps to survive #sxsw {link} join our #sxswbarcrawl during #sxswi	True	Apple	Match
7619	‰ûïfoursquare for bands&quot; just in time for #sxsw {link}	True	Apple	Match
8188	woohoo! rt @mention @mention #hollergram app is killing it at #sxsw {link}	True	Apple	Match

4.4.5 Reviewing the Uncategorized Tweets

I'm unsure what the criteria was for assembling this list of tweets. The most common theme seems to be South by Southwest 2011, an annual technology and art expo in Austin. Also, the deletion of the twitter handles can make the tweet indecipherable. The way twitter works, the @TwitterHandles are critical to understand the context of the entire tweet.

For instance, "Happy Birthday @danauerbach" tells us the author is celebrating the birthday of Dan Auerbach of the Black Keys fame. But "Happy Birthday @mention" doesn't tell us anything.

This being said, There are still 548 tweets that contain no keywords, and no clues in the *Product* column. The only alternative is to do further study on these tweets. I'll start with seeing what the most popular words on in these tweets.

In [54]:
```
temp_one = df.loc[df['Company'] == 'Unknown',:].copy()
temp_one['Temp'] = temp_one['Tweet'].apply(lambda x:str(x).split())

temp_count = Counter([item for sublist in temp_one['Temp'] for item in sublist
temp_counts = pd.DataFrame(temp_count.most_common(20))
temp_counts.columns = ['Common Words','Count']
temp_counts.style.background_gradient(cmap='Blues')
```

Out[54]:

	Common Words	Count
0	@mention	587
1	{link}	492
2	#sxsw	471
3	the	243
4	to	218
5	at	174
6	for	156
7	rt	106
8	are	99
9	of	98
10	a	91
11	in	84
12	you	82
13	on	75
14	and	72
15	free	68
16	&	65
17	not	60
18	is	55
19	from	48

In [55]:
```
df.loc[df['Company'] == 'Unknown', 'Company'].value_counts()
```

Out[55]:
```
Unknown    548
Name: Company, dtype: int64
```

Observations:

1. The tweets that don't specify a company are still overwhelmingly about SXSW. The number of times the term "SXSW" appears in this subset is higher than the number of tweets without a company.
2. Unfortunately, I cannot find any other words that clarify which company these tweets address. I spared you, the reader, the longer list, but I've read the top thousand of the most common words found in these unknown tweets, but I can't pick out any more keywords that tell me what company they are addressing.

If I'm going to justify deleting these tweets, I'm going to have to read through each one. I'll

In [56]:
```
unknowns = [670, 1860, 3859, 4925, 6901]
df.loc[unknowns, ['Tweet','Company']]
```

Out[56]:

	Tweet	Company
670	this just in: @mention to release new #socialmedia tool at #sxsw - {link}	Unknown
1860	learning about the future of search from @mention &amp; @mention creepily social and awesome. #sxsw	Unknown
3859	{link} this is insane... even more of a reason to go to sxsw #sxsw #appletogo	Unknown
4925	how many of you are in austin? follow @mention to find out what our googlers are up to at #sxsw {link}	Unknown
6901	rt @mention we are not launching any products at #sxsw but we are doing plenty else. {link} #googlecircle	Unknown

A few key words make it obvious what these are talking about:

670: "new #socialmedia tool": A reference to rumored Google Circles.

1860: "future of search": Google is who is famous for search.

3859: "#appletogo": Apple.

4925: "googlers": Google.

6901: "#googlecircle": Google Circle.

The next cell labels these accordingly.

In [57]:
```
# The lists and dictionaries below are hand made after reading through all the
# tweets with unknown companies. This function properly labels them.
def corrections(dfone, nkw):
    unknowns = [670, 1860, 3859, 4925, 6901]

    for i in unknowns:
        nkw.remove(i)

    new_company_labels = {
        670: 'Google',
        1860: 'Google',
        3859: 'Apple',
        4925: 'Google',
        6901: 'Google'
        }

    # Labeling the companies
    new_product_labels = {
        670:  'Circles',
        1860: 'Circles',
        6901: 'Circles'
        }
    for i in new_company_labels.keys():
        dfone.loc[int(i), 'Company'] = new_company_labels[i]
        dfone.loc[int(i), new_company_labels[i]] = True

    # Labeling the products for df
    for num, label in zip(new_product_labels.keys(), new_product_labels.values
        dfone.loc[int(num), label] = True

    dfone.loc[unknowns, ['Apple', 'Google', 'Android', 'Circles', 'Company']]

    return dfone.copy(), nkw

df, nokeywords = corrections(df, nokeywords)
```

In [58]:
```
still_unknown = len(df.loc[df['Company'] == 'Unknown', ['Tweet', 'Company']])
df['Company'].value_counts()
```

Out[58]:
```
Apple      4480
Google     2270
Unknown     543
Name: Company, dtype: int64
```

In [59]:
```
new_tot = len(df)
new_del = num_both + len(mismatch_products) + still_unknown
perc_del = round(new_del / new_tot * 10000) / 100

print('I deleted', num_both, 'tweets because they discussed both',
      'Google and Apple.')
print('I deleted', len(mismatch_products),
      'tweet because the keyword analysis mismatched the given Product')
print('column.')
print('I will now delete', still_unknown, 'tweets that do not specify a',
      'company or product.')
print('After deleting duplicates, I had', new_tot, 'tweets remaining.')
print('With the additional', new_del,
      'tweets described above deleted, I deleted a total of')
print('{}% of the original data that was original.'.format(perc_del))
```

```
I deleted 257 tweets because they discussed both Google and Apple.
I deleted 1 tweet because the keyword analysis mismatched the given Product
column.
I will now delete 543 tweets that do not specify a company or product.
After deleting duplicates, I had 7293 tweets remaining.
With the additional 801 tweets described above deleted, I deleted a total of
10.98% of the original data that was original.
```

I didn't want to go higher than 10%, but 1% higher is not something to be upset about. I will now drop those rows.

In [60]:
```
# Dropping rows with no company known.
df.drop(index=df.loc[df['Company'] == 'Unknown'].index, inplace=True)
```

Now, all the remaining rows in the DataFrame all have companies assigned. Now to specify each tweet as addressing Brand or Product.

4.5 Identifying Brand or Product

Now that I've pruned the data, and I know what company each tweet addresses, I will now begin the task identifying each tweet as addressing the company's products or brand. I'll start by creating two columns that list every keyword used for both Apple and Google.

In [61]:
```
# Create a column for both Apple Keywords and Google Keywords.

def categorize(dfone, kw_apple, kw_google):
    for i in dfone.index:
        # This variable will put each word in list format. I'll be able to
        # easily alphabetize the words in this form.
        keyword_lst = []
        # I want this cell to appear as a string, so I'll store the string her
        keyword_string = ''

        # This function picks the Apple or Google keywords and destination.
        if dfone.loc[i, 'Company'] == 'Apple':
            columns = kw_apple
            place = 'Apple Keyword'
        elif dfone.loc[i, 'Company'] == 'Google':
            columns = kw_google
            place = 'Google Keyword'
        else:
            continue

        # This for loop puts the keywords in the correct column and format.
        for col in columns:
            if dfone.loc[i, col] == True:
                keyword_lst.append(col)

        # Alphabetize the list
        # keyword_lst.sort()

        # Place the words in a string.
        for word in keyword_lst:
            keyword_string += word + ' '
        dfone.loc[i, place] = keyword_string[0:-1]
    return dfone

df = categorize(df, keywords_apple, keywords_google)
```

```
In [62]: apple_key_count = df.groupby('Apple Keyword').count()[
             'Tweet'].reset_index().sort_values(by='Tweet',ascending=False)
         apple_key_count.style.background_gradient(cmap='Greens')
```

Out[62]:

	Apple Keyword	Tweet
46	iPad	813
83	iPhone	788
73	iPad2	455
1	Apple	439
8	Apple Pop_Up	269
23	Apple iPad2	240
94	iPhone_App	182
41	Apple_Store	181
28	Apple iPad2 Pop_Up	150
81	iPad_App	105
74	iPad2 Apple_Store	102
43	Apple_Store Pop_Up	68
60	iPad iPhone	65
84	iPhone App_Store	61
96	iTunes	50
75	iPad2 Apple_Store Pop_Up	46
14	Apple iPad Pop_Up	45
48	iPad Apple_Store	42
10	Apple iPad	40
56	iPad iPad2	37
78	iPad2 Pop_Up	29
50	iPad Laptop	24
3	Apple Apple_Store	14
33	Apple iPhone	13
4	Apple Apple_Store Pop_Up	12
24	Apple iPad2 Apple_Store	11
25	Apple iPad2 Apple_Store Pop_Up	10
68	iPad1	10
86	iPhone Laptop	10
16	Apple iPad iPad2	10
52	iPad Pop_Up	9
51	iPad Macbook	8
70	iPad1 iPad2	7
91	iPhone iPad_App	6
62	iPad iPhone Laptop	6

	Apple Keyword	Tweet
49	iPad Apple_Store Pop_Up	5
79	iPad2 iPhone	5
40	Apple iTunes	5
45	Pop_Up	4
11	Apple iPad Apple_Store	4
65	iPad iPhone_App	4
90	iPhone iOS	4
2	Apple App_Store	4
66	iPad iTunes	3
64	iPad iPhone iOS	3
63	iPad iPhone Macbook	3
57	iPad iPad2 Apple_Store	3
6	Apple Laptop	3
67	iPad iTunes App_Store	3
13	Apple iPad Macbook	3
9	Apple iOS	3
7	Apple Macbook	3
85	iPhone Apple_Store	3
93	iPhone iTunes	3
61	iPad iPhone App_Store	3
76	iPad2 Laptop	2
77	iPad2 Macbook	2
89	iPhone Pop_Up	2
92	iPhone iPhone_App	2
18	Apple iPad iPad2 Pop_Up	2
42	Apple_Store Macbook	2
58	iPad iPad2 Pop_Up	2
20	Apple iPad iPhone	2
12	Apple iPad Apple_Store Pop_Up	2
27	Apple iPad2 Macbook	2
87	iPhone Macbook	1
82	iPad_App iPhone_App	1
32	Apple iPad2 iTunes Pop_Up	1
31	Apple iPad2 iPhone Macbook	1
30	Apple iPad2 iPhone Laptop	1
29	Apple iPad2 iPhone	1

	Apple Keyword	Tweet
59	iPad iPad_App	1
88	iPhone Macbook iMac	1
80	iPad2 iTunes	1
26	Apple iPad2 Laptop	1
17	Apple iPad iPad2 Laptop	1
22	Apple iPad1 iPad2 Apple_Store	1
21	Apple iPad1	1
95	iTune	1
5	Apple Apple_Store iOS	1
35	Apple iPhone Laptop	1
34	Apple iPhone App_Store	1
36	Apple iPhone Pop_Up	1
37	Apple iPhone iOS	1
38	Apple iPhone iTunes	1
39	Apple iPhone_App	1
15	Apple iPad iMac	1
72	iPad1 iPhone	1
71	iPad1 iPad2 Macbook	1
69	iPad1 Apple_Store	1
44	Laptop	1
47	iPad App_Store	1
53	iPad iOS	1
54	iPad iPad1	1
55	iPad iPad1 Laptop	1
19	Apple iPad iPad_App	1
0	App_Store	1

In [63]:
```
google_key_count = df.groupby('Google Keyword').count()[
    'Tweet'].reset_index().sort_values(by='Tweet',ascending=False)
google_key_count.style.background_gradient(cmap='Spectral')
```

Out[63]:

	Google Keyword	Tweet
11	Google	1138
17	Google Circles	476
0	Android	214
24	Google Marissa_Mayer	126
21	Google Maps	107
25	Google Marissa_Mayer Maps	42
7	Android_App	40
22	Google MarissaGoogle	13
9	Circles	12
32	Marissa_Mayer	11
14	Google Blogger	11
1	Android Android_App	10
26	Google Marissa_Mayer MarissaGoogle	8
19	Google Googlecircles	6
12	Google Android	6
2	Android Chrome	5
15	Google Chrome	5
30	Googlecom	5
18	Google Circles Googlecircles	5
13	Google Android ChromeOS	4
23	Google MarissaGoogle Maps	4
6	Android Samsung Galaxy	3
5	Android Samsung	3
31	Maps	2
3	Android Droid	2
4	Android Galaxy	2
8	Blogger	2
20	Google Googlecom	1
27	Google Marissa_Mayer MarissaGoogle Maps	1
28	Google Samsung	1
29	Google Samsung Galaxy	1
16	Google Chrome Maps	1
10	Droid	1
33	Marissa_Mayer Maps	1
34	Samsung Galaxy	1

Defining which refer to the brand and which refer to products needs to be done in a broad way. I'll identify the brand for Apple as any tweet that exclusively uses the keywords "Apple", "Apple_Store", and "Pop_Up". Any other apple terms will used in the tweet will result in the tweet categorized as a Apple Products tweet.

I will classify the tweets about the Apple Store and the Pop Up as part of the Apple Brand, since a store is not a product itself. A store is the physical space used to sell products, and the final bit of marketing to get people to buy once they have arrived with that purpose in mind. So since a store and a brand are both part of marketing, I'll label tweets about stores as part of the brand.

Likewise, Google Brand tweets will exclusively use the keywords "Google", "Marissa_Mayer", "MarissaGoogle", "Googlecom", "Circles", and "Googlecircles". And likewise, any other Google Keyword used will result in the tweet categorized as a Google Products tweet.

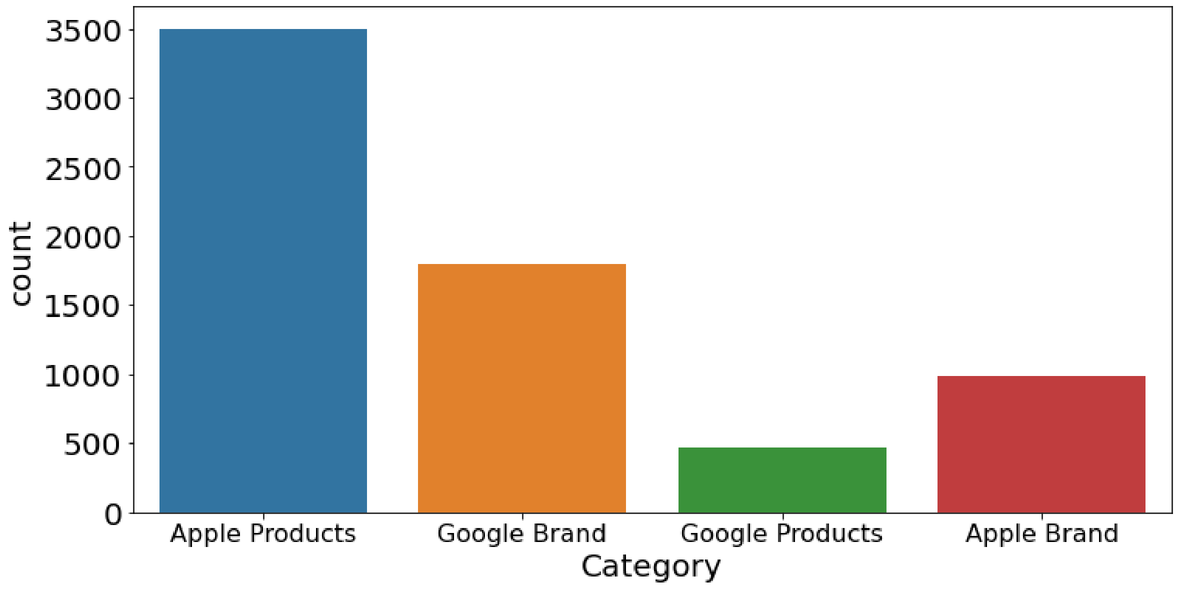
I am classifying all Google Circles tweets as part of Google Brand since, in March 2011, Google Circles was mere speculation for it's upcoming social network, that eventually was confirmed to be Google Plus. I decided to leave this category as Google Brand because people's speculation about the product reflects what they think of the possibilities and quality they associate with the Google Brand.

The following cell creates a single column for Keywords, and then identifies each tweet as one of now four categories:

1. Apple Brand
2. Apple Products
3. Google Brand
4. Google Products

In [64]:
```
# This labels each tweet as either "Apple Brand", "Apple Products",
#  "Google Brand" or "Google Products".
def keyword_combos(dfone, kw_apple, kw_google):
    # Create lists of words that, if used exclusively, will categorize the
    # tweet as referring to a brand.
    apple_brand = ['Apple', 'Apple_Store', 'Pop_Up']
    google_brand = ['Google', 'Marissa_Mayer', 'MarissaGoogle', 'Googlecom',
                    'Circles', 'Googlecircles']

    # Create a list of keywords that, if used in the tweet, will supercede the
    # brand categorization, and categorize the tweet as a product.
    apple_products = kw_apple.copy()
    google_products = kw_google.copy()

    for prod in apple_products.copy():
        if prod in apple_brand:
            apple_products.remove(prod)

    for prod in google_products.copy():
        if prod in google_brand:
            google_products.remove(prod)

    # Categorizing the tweets
    for i in dfone.index:
        company = dfone.loc[i, 'Company']
        goog_prod_sum = sum(dfone.loc[i, google_products])
        apple_prod_sum = sum(dfone.loc[i, apple_products])
        is_it_a_product = goog_prod_sum + apple_prod_sum
        if (company == 'Apple') | (company == 'Google'):
            if is_it_a_product > 0:
                string = company + ' ' + 'Products'
            else:
                string = company + ' ' + 'Brand'
            dfone.loc[i, 'Keywords'] = dfone.loc[i, company + ' Keyword']
            dfone.loc[i, 'Category'] = string

    return dfone.copy()

df = keyword_combos(df, keywords_apple, keywords_google)
```

In [65]:
```
font = {'family' : 'DejaVu Sans',
            'weight' : 'normal',
            'size'   : 16}
plt.rc('font', **font)
plt.figure(figsize=(12,6))
plt.rc('axes', labelsize=20)
plt.rc('ytick', labelsize=20)
sns.countplot(x = 'Category', data = df);
```



Observations:

Another graph that makes it clear Google is talked about far less than Apple. This makes it clear people talk about Apple Products more than the Google brand and Google products combined.

4.6 Cutting the DataFrame Columns

Before moving on, I don't need most of the columns in the DataFrame now. Many of the columns have been created exclusively to categorize the tweets. Now that the categorization is complete, I no longer need these columns. Also, I no longer need the given Products column. So now I'll trim the DataFrame to only the columns I need for NLP Analysis.

In [66]:
```
df.columns
```

Out[66]:
```
Index(['Tweet', 'Product', 'Emotion', 'Tokens', 'Tokens With Hashtags',
       'Hashtags', 'Apple', 'iPad', 'iPad1', 'iPad2', 'iPhone', 'iTunes',
       'iPhone5', 'iTune', 'Laptop', 'Apple_Store', 'App_Store', 'iOS',
       'Macbook', 'iPad_App', 'iPhone_App', 'Pop_Up', 'iMac', 'Google',
       'Marissa_Mayer', 'MarissaGoogle', 'Googlecom', 'Android', 'Droid',
       'Circles', 'Blogger', 'Chrome', 'Samsung', 'Galaxy', 'Maps',
       'GoogleDoodle', 'ChromeOS', 'Googlecircles', 'Googletv', 'Android_Ap
p',
       'Company', 'Company by Product', 'Pd-Cp Match', 'Apple Keyword',
       'Google Keyword', 'Keywords', 'Category'],
      dtype='object')
```

In [67]:
```
df = df.loc[:, ['Tweet', 'Emotion', 'Tokens', 'Tokens With Hashtags',
                'Hashtags', 'Keywords', 'Company', 'Category']].copy()

df[['Tweet', 'Category']].head()
```

Out[67]:

	Tweet	Category
0	.@wesley83 i have a 3g iphone. after 3 hrs tweeting at #rise_austin, it was dead! i need to upgrade. plugin stations at #sxsw.	Apple Products
1	@jessedee know about @fludapp ? awesome ipad/iphone_app that you will likely appreciate for its design. also, they are giving free ts at #sxsw	Apple Products
2	@swonderlin can not wait for #ipad2 also. they should sale them down at #sxsw.	Apple Products
3	@sxsw i hope this year's festival is not as crashy as this year's iphone_app. #sxsw	Apple Products
4	@sxtxstate great stuff on fri #sxsw: marissa_mayer (google), tim_o_reilly (tech books/conferences) &amp; matt_mullenweg (wordpress)	Google Brand

4.7 Creating Targets

Now that I have the data I want, I need to set some targets for each. As stated earlier, There will be twelve targets:

1. Company: Google or Apple
2. Aspect of Company: Brand or Product
3. Emotion: Positive, Negative, or Neutral

The emotion classification doesn't need any work done on it because it came ready to go. However, I had forgotten that "Indecipherable" is still an emotion label. After all the deletions, there are still 130 tweets with the unusable emotion label. I'll delete those.

In [68]:
```
print('Emotion labeled "' + "Indecipherable" + '" count: ',
      len(df.loc[df['Emotion'] == "Indecipherable"]))

# Dropping rows with no emotion known.
df.drop(index=df.loc[df['Emotion'] == "Indecipherable"].index, inplace=True)

# Resetting the dataset's index. (last deletion and reset)
df.reset_index(drop=True, inplace=True)

print('Purged this emotion label from the dataset.\nChecking:')
print('Emotion labeled "' + "Indecipherable" + '" count: ',
      len(df.loc[df['Emotion'] == "Indecipherable"]))
```

```
Emotion labeled "Indecipherable" count:  130
Purged this emotion label from the dataset.
Checking:
Emotion labeled "Indecipherable" count:  0
```

Now it's finally time to assign each tweet a target and target number.

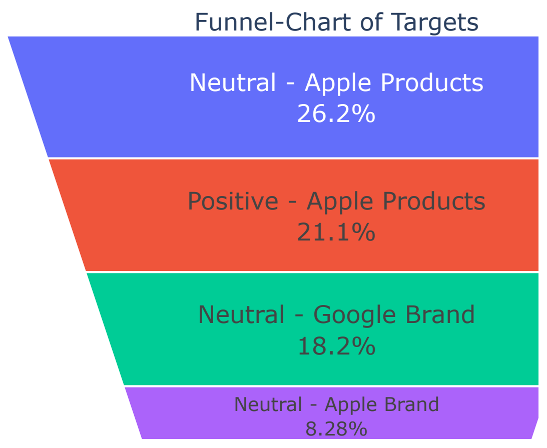
Natural Language Processing is a statistical process that converts the corpus into numerical data, and then tries to correlate this numerical data to the target number. I point this out now to clarify that it must be a number, and not text. Therefore, each of the tweve targets will be assigned the numbers 0-11, as the models require. Since I have each tweet labeled with an emotion and a company-product/brand category, I'll use simple arithmatic to set up the numerical labels.

```
In [69]: emotion_num = {'Negative': 0, 'Neutral': 1, 'Positive': 2}

         category_num = {
             'Apple Brand': 0,
             'Apple Products': 3,
             'Google Brand': 6,
             'Google Products': 9
             }

         def targets(dfone):
             # This for loop will label three columns.
             dfone['Target Text'] = ''
             dfone['Target'] = 0
             for i in range(len(dfone)):
                 if (dfone.loc[i, 'Company'] != 'Unknown') & (
                 dfone.loc[i, 'Company'] != 'Both') & (
                 dfone.loc[i, 'Emotion'] != "Indecipherable"):
                     dfone.loc[i,'Target Text'] = dfone.loc[
                               i, 'Emotion'] + ' - ' + dfone.loc[i, 'Category']
                     dfone.loc[i, 'Target'] = (emotion_num[dfone.loc[i, 'Emotion']] +
                                              category_num[dfone.loc[i, 'Category']])
             return dfone.copy()

         df = targets(df)

         target_count = df.groupby('Target Text').count()[
             'Tweet'].reset_index().sort_values(by='Tweet',ascending=False)
         target_count.rename(columns={'Tweet' : 'Count'},inplace=True)
         target_count.style.background_gradient(cmap='Reds')
```

Out[69]:

	Target Text	Count
5	Neutral - Apple Products	1733
9	Positive - Apple Products	1400
6	Neutral - Google Brand	1207
4	Neutral - Apple Brand	548
10	Positive - Google Brand	457
8	Positive - Apple Brand	361
1	Negative - Apple Products	292
7	Neutral - Google Products	235
11	Positive - Google Products	205
2	Negative - Google Brand	98
0	Negative - Apple Brand	60
3	Negative - Google Products	24

```
In [70]: fig = go.Figure(go.Funnelarea(
             text =target_count['Target Text'],
             values = target_count['Count'],
             title = {"position": "top center", "text":
                       "Funnel-Chart of Targets"}
             ))
         fig.layout.update(showlegend=False)
         fig.show()
```



Observations:

1. Of the six lowest represented targets, four are the four negative targets. This illustrates a point already seen in above graphs, that negative tweets are underrepresented in this data set. This will lead to more errors when the model predicts negative targets.
2. Of the top seven highest represented targets, five of them are Apple. It's clear that people are tweeting about Apple far more often than Google.

4.8 Removing Stopwords

Now that I've got the official targets and numbers, I will remove stopwords and see what the word distribution is like for the remaining tokens.

The reason removing stopwords is critical to the NLP process is an excess of simple pronouns and to-be verbs clutters the NLP output with definition-free words that don't direct the model well. The standard procedure is to remove them.

During the search for compound nouns, I found several odd tokens that needed deletion entirely. I made this list.

The first five words make sense, the rest is some kind of nonsense from the scraping process of twitter. For the first five, here is why they need to be deleted:

Mention: Deleted because it represents any of thousands of twitter handles.

rt: A sign that it is not the author's tweet, not part of the author's sentence.

quot: Represents punctuation, not a word.

link: Signifies the tweet had a links. These represent anything.

amp Signifies an ampersand. "And" is stopword regardless.

In [71]:
```
# Downloading the standard stopword list
nltk.download('stopwords', quiet=True)

stopwords_list = stopwords.words('english')
```

In [72]:
```
# Additional stopwords
stopwords_list.extend(['mention', 'rt', 'quot', 'link', 'amp',
                       'ûó', 'ûò', 'ûï', 'û_', 'ã_', 'ûª', 'åç', 'åè', 'ä_',
                       'â_', 'dì'])
```

In [73]:
```
# Returns a token list without the list of specified stopwords.

def remove_stopwords(token_list, stop_list):
    """
    Given a list of tokens, return a list where the tokens
    that are also present in stopwords_list have been
    removed
    """
    stopwords_removed = [
        token for token in token_list if token not in stop_list
    ]
    return stopwords_removed
```

In [74]:
```
def trim_stopwords(dfone):
    dfone.insert(2, 'Tidy Tweet', '')
    dfone.insert(5, 'Tokens Without Stopwords', '')
    dfone["Tokens Without Stopwords"] = dfone[
        "Tokens"].map(lambda x : remove_stopwords(x, stopwords_list))
    for ind in dfone.index:
        dfone.loc[ind, 'Tidy Tweet'] = " ".join(i
                                   for i in dfone.loc[ind, 'Tokens Without Stopwords'
    return dfone.copy()

df = trim_stopwords(df)

df[['Tweet', 'Tidy Tweet']].head()
```

Out[74]:

	Tweet	Tidy Tweet
0	.@wesley83 i have a 3g iphone. after 3 hrs tweeting at #rise_austin, it was dead! i need to upgrade. plugin stations at #sxsw.	wesley83 3g iphone hrs tweeting rise_austin dead need upgrade plugin stations sxsw
1	@jessedee know about @fludapp ? awesome ipad/iphone_app that you will likely appreciate for its design. also, they are giving free ts at #sxsw	jessedee know fludapp awesome ipad iphone_app likely appreciate design also giving free ts sxsw
2	@swonderlin can not wait for #ipad2 also. they should sale them down at #sxsw.	swonderlin wait ipad2 also sale sxsw
3	@sxsw i hope this year's festival is not as crashy as this year's iphone_app. #sxsw	sxsw hope year festival crashy year iphone_app sxsw
4	@sxtxstate great stuff on fri #sxsw: marissa_mayer (google), tim_o_reilly (tech books/conferences) &amp; matt_mullenweg (wordpress)	sxtxstate great stuff fri sxsw marissa_mayer google tim_o_reilly tech books conferences matt_mullenweg wordpress

4.9 Stemmer Tokens

Stemming the tokens involves eliminating the normal suffixes of the English language. For instance, instead of the NLP model considering the words "snowballs" and "snowball" as separate words, the stemmer trims the "s" off "snowballs", thus they are computed as the same word.

For stemming, I will use the standard snowball stemmer. I'll use the stemmed tokens for the modeling and association portions of the project below.

In [75]:
```
stemmer = SnowballStemmer(language="english")

def stem_and_tokenize(document):
    tokens = tokenizer.tokenize(document)
    return [stemmer.stem(token) for token in tokens]
```

In [76]:
```
# The stopwords themselves needed to be stemmed.
stemmed_stopwords = [stemmer.stem(word) for word in stopwords_list]
```

In [77]:
```
def stem_tokens(dfone):
    dfone.insert(3, 'Stemmed Tweet', '')
    dfone.insert(7, 'Stemmed Tokens', '')
    dfone["Stemmed Tokens"] = dfone['Tokens Without Stopwords'].apply(
        lambda x: [stemmer.stem(i) for i in x])
    dfone["Stemmed Tokens"] = dfone[
        "Stemmed Tokens"].map(lambda x : remove_stopwords(x,stemmed_stopwords)
    # The stemmer got overzealous with some of the keywords. This fixes that.
    fix_list = {'appl' : 'apple', 'googl' : 'google', 'iphon': 'iphone',
                'apple_stor' : 'apple_store', 'circl': 'circles',
                'marissa_may' : 'marissa_mayer', 'itun' : 'itunes',
                'app_stor' : 'app_store', 'io' : 'ios',
                'marissagoogl' : 'marissagoogle', 'map' : 'maps',
                'googledoodl' : 'googledoodle', 'chromeo' : 'chromeos',
                'googlecircl' : 'googlecircles'}
    for i in dfone["Stemmed Tokens"].index:
        for j in range(len(dfone.loc[i, "Stemmed Tokens"])):
            for fix in fix_list:
                if dfone.loc[i, "Stemmed Tokens"][j] == fix:
                    dfone.loc[i, "Stemmed Tokens"][j] = fix_list[fix]

    for ind in dfone.index:
        dfone.loc[ind, 'Stemmed Tweet'] = " ".join(i
                                   for i in dfone.loc[ind, 'Stemmed Tokens'])
    return dfone.copy()

df = stem_tokens(df)

df.loc[0:4,['Tokens Without Stopwords', 'Stemmed Tokens']]
```

Out[77]:

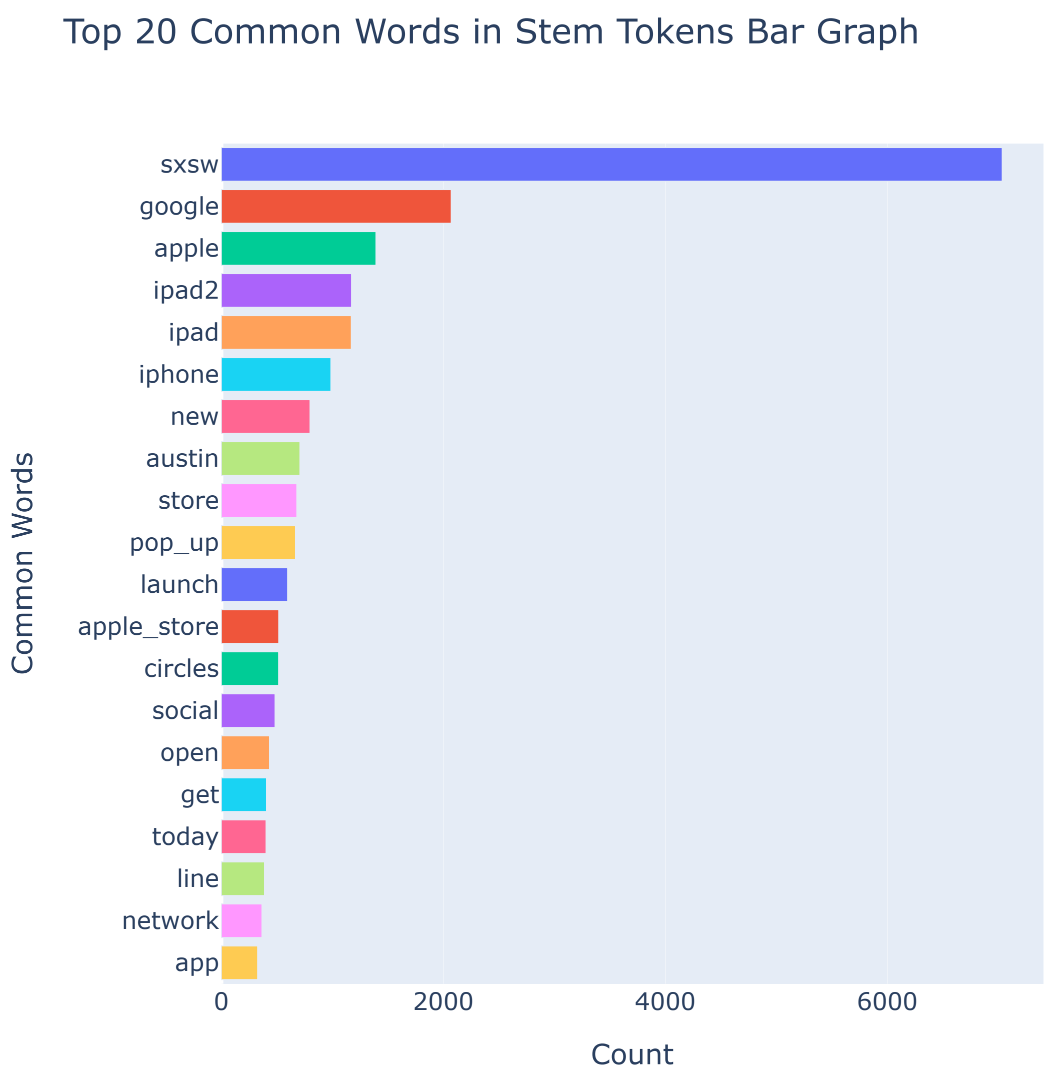
	Tokens Without Stopwords	Stemmed Tokens
0	[wesley83, 3g, iphone, hrs, tweeting, rise_austin, dead, need, upgrade, plugin, stations, sxsw]	[wesley83, 3g, iphone, hrs, tweet, rise_austin, dead, need, upgrad, plugin, station, sxsw]
1	[jessedee, know, fludapp, awesome, ipad, iphone_app, likely, appreciate, design, also, giving, free, ts, sxsw]	[jessede, know, fludapp, awesom, ipad, iphone_app, like, appreci, design, also, give, free, ts, sxsw]
2	[swonderlin, wait, ipad2, also, sale, sxsw]	[swonderlin, wait, ipad2, also, sale, sxsw]
3	[sxsw, hope, year, festival, crashy, year, iphone_app, sxsw]	[sxsw, hope, year, festiv, crashi, year, iphone_app, sxsw]
4	[sxtxstate, great, stuff, fri, sxsw, marissa_mayer, google, tim_o_reilly, tech, books, conferences, matt_mullenweg, wordpress]	[sxtxstate, great, stuff, fri, sxsw, marissa_mayer, google, tim_o_reilli, tech, book, confer, matt_mullenweg, wordpress]

4.10 Frequency Distributions

Now I'll display the word frequencies for the dataset after the the tweets have been stemmed. First, I'll make a graph of all the words.

4.10.1 Top 20 Common Words Bar Graph

In [78]:
```
# This cell creates a bar graph showing the frequency of the top twenty
# most used words.
temp_sw = df[['Tweet', 'Stemmed Tweet']].copy()
temp_sw['Temp'] = temp_sw['Stemmed Tweet'].apply(lambda x:str(x).split())

temp_count_sw = Counter([
    item for sublist in temp_sw['Temp'] for item in sublist])
temp_counts_sw = pd.DataFrame(temp_count_sw.most_common(20))
temp_counts_sw.columns = ['Common Words','Count']

fig = px.bar(temp_counts_sw, x="Count", y="Common Words",
             title='Top 20 Common Words in Stem Tokens Bar Graph',
             orientation='h',
             width=600, height=600, color='Common Words')
fig.layout.update(showlegend=False)
fig.show()
```



Observations:

1. The most obvious takeaway from this graph is that "SXSW" is by far the most used words in this tweet set. "SXSW" is used more than three times as often as the #2 used word, "Google", and five times more than the #3 used word "Apple".
2. If I split the words into words into three categories, Company-Free, Google, and Apple it works out to 12 company-free words, 2 Google words, and six Apple words. These top six apple words were used 5909 times, and the top two Google words 2587 times.

4.10.2 Top 10 most used words by Target

These next three functions allow easy creation of a top ten visual showing the frequency of the words in the dataset.

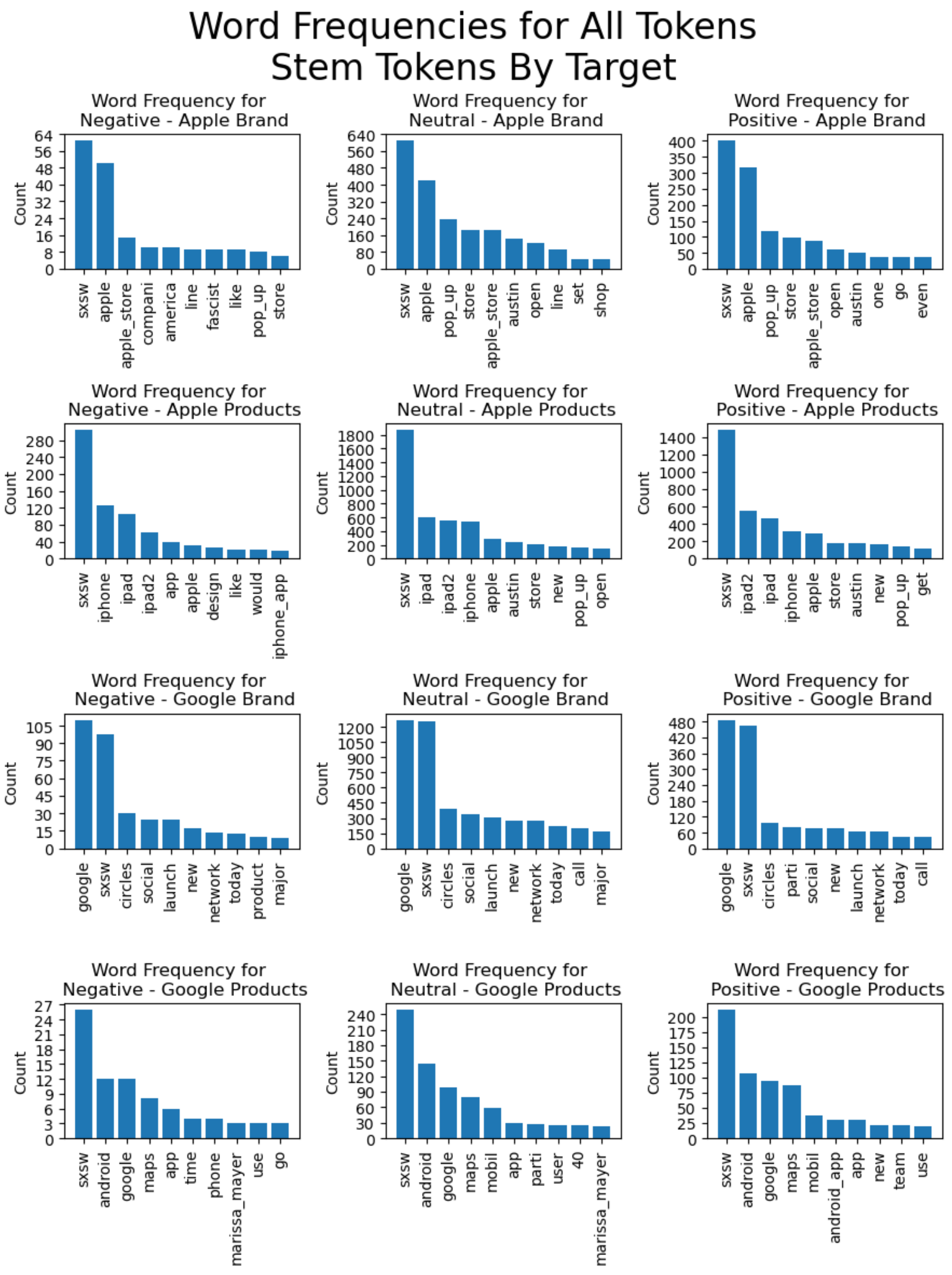
In [79]:
```
# Returns a visualization of the distribution of the top ten words in the
# specified data.
def visualize_top_10(freq_dist, title):

    # Extract data for plotting
    top_10 = list(zip(*freq_dist.most_common(10)))
    tokens = top_10[0]
    counts = top_10[1]

    # Set up plot and plot data
    fig, ax = plt.subplots()
    ax.bar(tokens, counts)

    # Customize plot appearance
    ax.set_title(title)
    ax.set_ylabel("Count")
    ax.yaxis.set_major_locator(MaxNLocator(integer=True))
    ax.tick_params(axis="x", rotation=90)
```

In [80]:
```
# Sets up twelve subplots. This is used with other functions.

def setup_twelve_subplots():
    fig = plt.figure(figsize=(9, 12))
    fig.set_tight_layout(True)
    gs = fig.add_gridspec(4, 3)
    ax1 = fig.add_subplot(gs[0, 0])
    ax2 = fig.add_subplot(gs[0, 1])
    ax3 = fig.add_subplot(gs[0, 2])
    ax4 = fig.add_subplot(gs[1, 0])
    ax5 = fig.add_subplot(gs[1, 1])
    ax6 = fig.add_subplot(gs[1, 2])
    ax7 = fig.add_subplot(gs[2, 0])
    ax8 = fig.add_subplot(gs[2, 1])
    ax9 = fig.add_subplot(gs[2, 2])
    ax10 = fig.add_subplot(gs[3, 0])
    ax11 = fig.add_subplot(gs[3, 1])
    ax12 = fig.add_subplot(gs[3, 2])
    return fig, [ax1, ax2, ax3, ax4, ax5, ax6, ax7, ax8, ax9, ax10, ax11, ax12
```

In [81]:
```
# Sets up twelve word frequency distributions for all target categories.
def plots(dataf, targ, column, axes, title="Word Frequency for"):
    ax_num = 0

    # Creating a DataFrame of just the target text with their target number as
    # the index.
    df_target = pd.DataFrame()

    for index, category in zip(dataf['Target'].unique(),
                               dataf['Target Text'].unique()):
        df_target.loc[index, 'Target Text'] = category

    df_target = df_target.sort_index()

    for index, category in zip(df_target.index, df_target.values):
        # Calculate frequency distribution for this subset
        all_words = dataf[targ == index][column].explode()
        freq_dist = FreqDist(all_words)
        top_10 = list(zip(*freq_dist.most_common(10)))
        tokens = top_10[0]
        counts = top_10[1]

        # Set up plot
        ax = axes[ax_num]
        ax.bar(tokens, counts)

        # Customize plot appearance
        ax.set_title(f"{title} \n {str(category)[2:-2]}")
        ax.set_ylabel("Count")
        ax.yaxis.set_major_locator(MaxNLocator(integer=True))
        ax.tick_params(axis="x", rotation=90)
        ax_num += 1
```

Now, I'll split the word distributions up into twelve distribution plots, one per target, showing the top ten for each.

In [82]:
```
# Returning font sizes to their default.
plt.rcParams.update(plt.rcParamsDefault)

# Printing a grid of the word frequencies.
fig, axes = setup_twelve_subplots()
plots(df, df['Target'], "Stemmed Tokens", axes)
fig.suptitle("Word Frequencies for All Tokens\nStem Tokens By Target",
             fontsize=24);
```



Observations:

1. Each top ten category had "SXSW" as the the number one or two used term. This almost certainly was used in the scraping criteria.
2. In most target graphs, with the exception of "SXSW" the highest frequency words all are specific to a company. There's a noticeable drop off after these words that occurs in most targets. For example, above you see that the top word frequencies for "Positive - Google Products" are "SXSW", "Android", "Google", and "Maps". Directly after those words, the frequency is cut considerably, with "Mobil" occurring less than half as often as "Maps".
3. In the grid of twelve different graphs above, you'll notice that each column represents its own very few words have an obvious emotional charge based on the word above. I see the following words with an obvious emotional charge:
   A. Negative: Fascist
   B. Positive: New, Launch, Go

4.10.3 Word Cloud For Full Dataset

These next two functions print word clouds, and also twelve word clouds, one for each target.

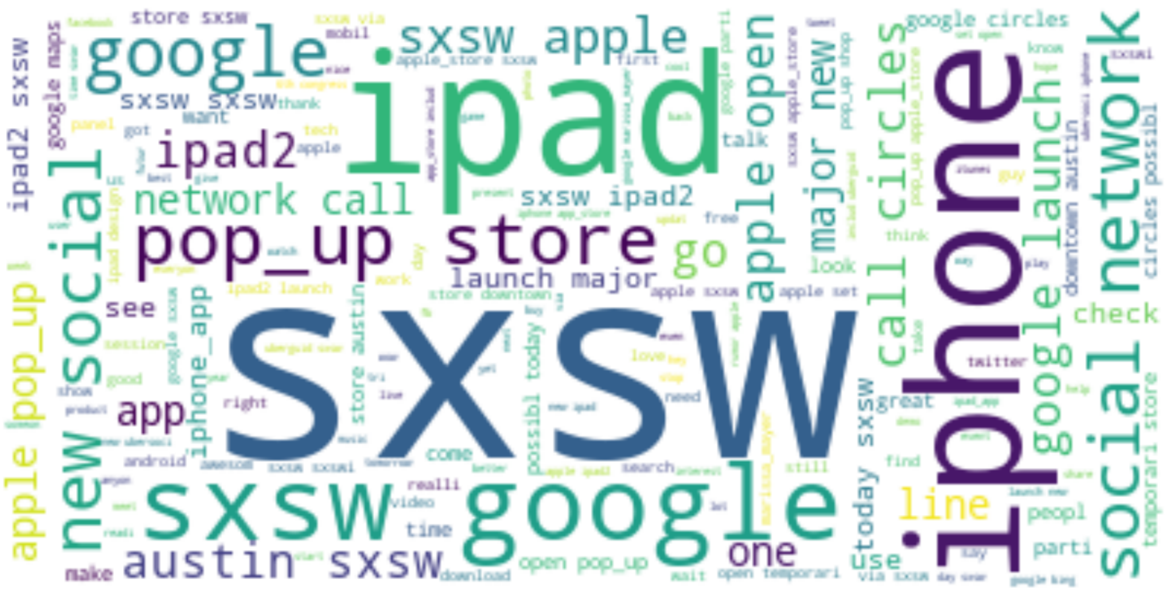
In [83]:
```python
# Plots a single word cloud for the specified data.

def wc(tokens, title='', re = r"(?:\w[\w']+)"):
    if isinstance(tokens, pd.Series):
        text = " ".join(i for i in tokens.explode())
    else:
        text = " ".join(i for i in tokens)
    wordcloud = WordCloud(background_color="white", regexp = re).generate(text
    fig = plt.figure(figsize=(15, 10))
    plt.imshow(wordcloud, interpolation='bilinear')
    plt.axis("off")
    fig.set_tight_layout(True)

    if title != '':
        fig.suptitle(title, fontsize=48)
    plt.show()
```

In [84]:
```
# Plots twelve word clouds for each specified target.

def word_cloud_grid(dataf, targ, column, title=''):
    tgt_lst = []
    wc_lst = []
    # Creating a DataFrame of just the target text with their target number as
    # the index.
    df_target = pd.DataFrame()
    for index, category in zip(dataf['Target'].unique(),
                               dataf['Target Text'].unique()):
        df_target.loc[index, 'Target Text'] = category
    df_target = df_target.sort_index()

    tgt_num = list(df_target.index)

    for num in tgt_num:
        tgt_lst.append(" ".join(
            i for i in dataf[column].loc[targ == num].explode()))
    for tgt in tgt_lst:
        wc_lst.append(WordCloud(background_color="white").generate(tgt))
    fig = plt.figure(figsize=(9, 9))
    fig.set_tight_layout(True)
    gs = fig.add_gridspec(4, 3)
    ax1 = fig.add_subplot(gs[0, 0])
    ax2 = fig.add_subplot(gs[0, 1])
    ax3 = fig.add_subplot(gs[0, 2])
    ax4 = fig.add_subplot(gs[1, 0])
    ax5 = fig.add_subplot(gs[1, 1])
    ax6 = fig.add_subplot(gs[1, 2])
    ax7 = fig.add_subplot(gs[2, 0])
    ax8 = fig.add_subplot(gs[2, 1])
    ax9 = fig.add_subplot(gs[2, 2])
    ax10 = fig.add_subplot(gs[3, 0])
    ax11 = fig.add_subplot(gs[3, 1])
    ax12 = fig.add_subplot(gs[3, 2])

    i = 0
    for ax in [ax1, ax2, ax3, ax4, ax5, ax6, ax7, ax8, ax9, ax10, ax11, ax12]:
        ax.axis("off")
        ax.imshow(wc_lst[i], interpolation='bilinear')
        ref = tgt_num[i]
        ax.set_title(f"Word Cloud \n {df_target.loc[ref,'Target Text']}")
        i += 1
    if title != '':
        fig.suptitle(title, fontsize=36)
    plt.show()
```

Since word clouds are so neat, here's a word cloud for all the stem tokens.

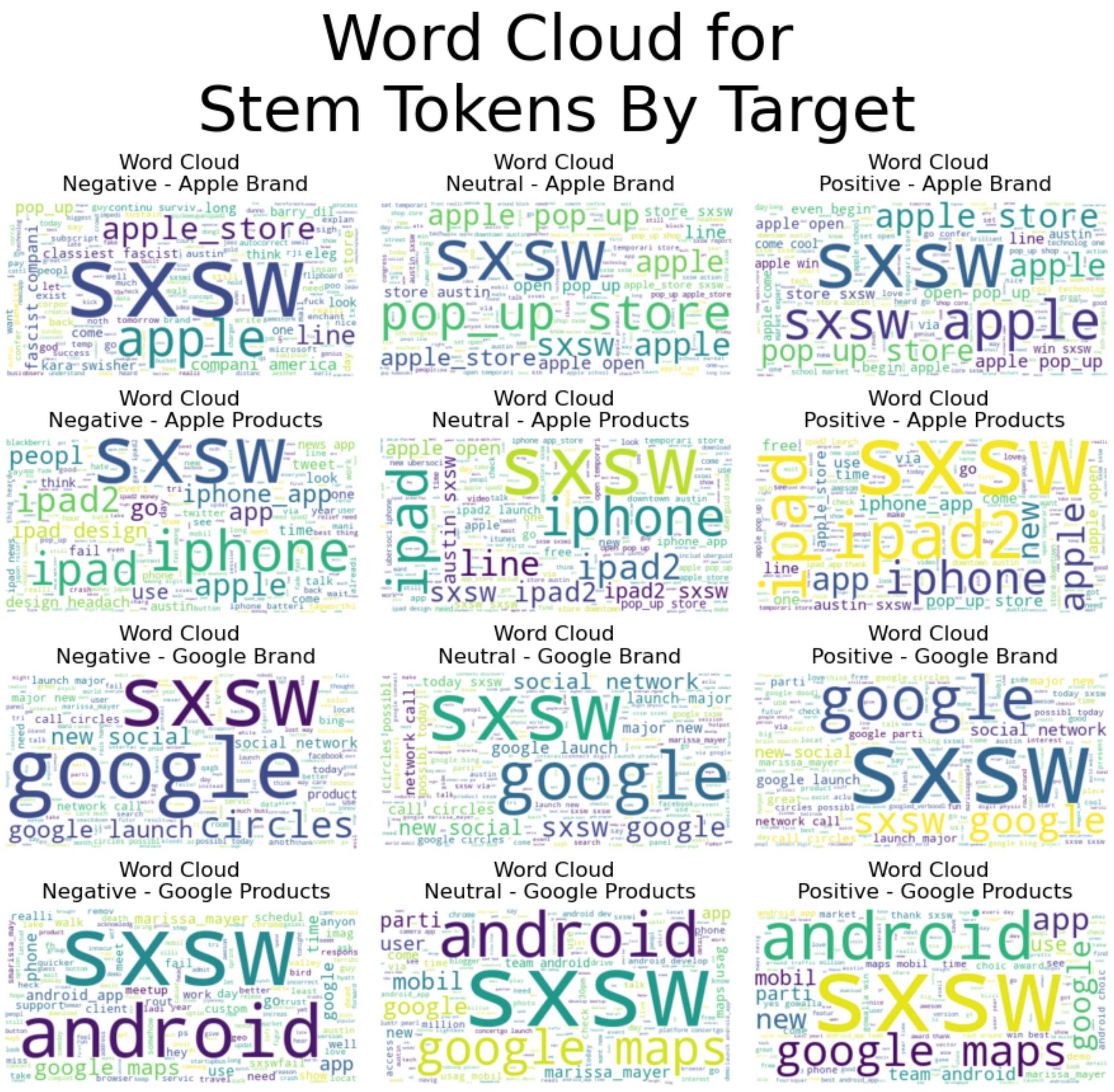
In [85]:
```
wc(df['Stemmed Tokens'],
   title="Word Cloud for Entire\nStem Tokens DataFrame")
```

Word Cloud for Entire
Stem Tokens DataFrame



Finally, here is a word cloud for each of the twelve targets.

```
In [86]: word_cloud_grid(df, df['Target'], 'Stemmed Tokens',
                         title = "Word Cloud for\nStem Tokens By Target")
```



These graphs, though built on word frequencies, are not meant for serious research observations. Most of what you can see in them are the top ten words, which you can already see in the word frequency bar charts, which is where I made observations on word frequency.

4.11 Splitting the data into Train and Test sets

This is the last stop before I begin vectorization and modeling.

The modeling process uses statistical analysis to predict future outcomes. It observes multiple instances of data and the data's proper classification. This is called the training process. The multiple observations then allow the model to predict a classification for a new set of Data. Since the more data used to train yields better results, people prefer to give a model as much data as they can to train with. However, the ending of the training process is dictated by how much data is available. Also, before using the model to classify new data, It's best policy to verify that the model's classifications are correct. To accomplish this, a portion of the already classified data needs to be saved to verify if the model is classifying the data correctly. Again, how much data you save for testing purposes is determined by the amount of data you have.

It's standard practice to split whatever data is available between 80% for training and 20% for

In [87]:
```
# Creating the Test/Train split, 80% / 20%
xtrain, xtest, ytrain, ytest = train_test_split(df.iloc[:, 0:-1],
                                                df.iloc[:, -1],
                                                train_size=.8,
                                                random_state=42)
```

I'm paranoid that the function will somehow pick too few of one of the less common targets for training purposes and it then won't predict that target in the test. So I will test to see how close the train/test split is to the specified 80% train size for each target.

In [88]:
```
for targ in range(len(ytrain.unique())):
    train_size = ytrain.value_counts()[targ]
    test_size = ytest.value_counts()[targ]
    train_perc = round(train_size / (train_size + test_size), 3)
    print(targ, '-', train_perc)
```

```
0 - 0.767
1 - 0.774
2 - 0.759
3 - 0.788
4 - 0.798
5 - 0.8
6 - 0.867
7 - 0.821
8 - 0.794
9 - 0.75
10 - 0.826
11 - 0.82
```

I'm impressed. This is nigh perfect. I'll trust this function more in the future.

5 Vectorizing The Data

For this NLP analysis, I will test three vectorization methods against five different models. Each vectorization technique and model will be explained further in the cells below. Here is a list of the methods used below:

Vectorization Techniques:

1. TF-IDF
2. Bag-Of-Words
3. Word2Vec

The models:

1. Multinomial Naive Bayes
2. Logistic Regression
3. Support Vector Machine
4. RandomForest
5. XGBoost

5.1 Build TF-IDF Vectorizer and Evaluate it with a Baseline Model, Multinomial Naive Bayes

The goal of this next section is twofold:

1. Convert the extracted stemmed tokens to quantifiable data, or numbers. This process is known as vectorization. The technique used in this section is called **Term Frequency–Inverse Document Frequency (TF-IDF)**.
2. Test the vectorization and fine tune the method for best results by using the data from part one to train the **Multinomial Naive Bayes model**, then evaluating the new model's results using the **K-Fold Cross Validation method** and the **weighted f1 score**.

5.1.1 Definition of terms:

Term Frequency–Inverse Document Frequency (TF-IDF): This is two different values multiplied together. The Term Frequency, and the Inverse Document Frequency. For starters, this method analyzes each word, or token. The **Term Frequency** takes the total number of times a token occurs in an individual tweet, and then divides that number by the total number of tokens in the tweet. The **Inverse Document Frequency** takes the total number of tweets and divides it by the total number of tweets with this token in it. Then, these two numbers are multiplied together.

Multinomial Naive Bayes model: A multi-algorithmic model that is often used in NLP. It uses multiple algorithms that calculate the probability of each classification and returns the highest classification probability as the prediction.

K-Fold Cross Validation method: This acts as a miniature test of the whole process. The function splits the training data into five even sets, called folds. Then it further splits each fold into five additional sets, with four acting as training data and one as test data for each fold. The process is repeated five times, but the "K" in K-Fold is a stand in for the number of folds. So in the scenario I just described, K is five, but K can be other numbers as well. Also, the K-Fold Cross Validation method can return whatever score I choose. For this project, the score that makes the most sense is the weighted f1-score. Before I explain this, I have to explain a few other terms.

Context for scores: The remaining terms I'm defining are all within the context of a classification model's output. So remember, you give a model input, it classifies the input, and then returns it to you classified. Trouble is, sometimes it's wrong, and sometimes it's right. These next few terms help us describe how the model performed.

True-Positive (TP): The number of **correct** predictions the model gave that a specific classification **is** that classification. This is calculated per class, so in this case, there's a TP value for 'Neutral - Apple Brand', and another for 'Neutral - Apple Products'. If the model said there were 100 'Neutral - Google Products' targets, and only 80 were correct, the TP for 'Neutral - Google Products' is 80.

True-Negative (TN): The number of **correct** predictions the model gave that a specific classification **is not** that classification. This is calculated per class, so in this case, there's a TN value for 'Neutral - Apple Brand', and another for 'Neutral - Apple Products'. If the model said there were 100 'Neutral - Google Products' targets, and only 80 were correct, but there were a total of 500 predictions for all categories and 180 true targets in this category, the TN for 'Neutral - Google Products' is 300.

False Positive (FP): The number of **incorrect** predictions the model gave that a specific classification **is** that classification. This is calculated per class, so in this case, there's a FP value for 'Neutral - Apple Brand', and another for 'Neutral - Apple Products'. If the model said there were 100 'Neutral - Google Products' targets, and only 80 were correct, the FP for 'Neutral - Google Products' is 20.

False Negative (FN): The number of **incorrect** predictions the model gave that a specific classification **is not** that classification. This is calculated per class, so in this case, there's a FN value for 'Neutral - Apple Brand', and another for 'Neutral - Apple Products'.If the model said there were 100 'Neutral - Google Products' targets, and only 80 were correct, but there were a total of 500 predictions for all categories and 180 true targets in this category, the FN for 'Neutral - Google Products' is 100.

Precision: $\frac{TP}{TP+FP}$ A measurement of the number of True-Positives divided by every positive prediction made for that category. "When you said the animal was a rabbit, you were right $\frac{3}{4}$ of the time." = Precision of 0.75. Precision is prioritized when you want to avoid False-Positives.

If you were sending people of different nationalities back to their home country, and sending someone to Fascist-Land incorrectly meant they would be executed on sight, you'd prioritize **Precision** for residents of Fascist-Land.

Recall: $\frac{TP}{TP+FN}$ A measurement of True-Positives divided by the sum of all true cases of that particular classification. "When you said the animal was a rabbit, you were right $\frac{3}{4}$ times, but you only identified $\frac{3}{5}$ of the rabbits as rabbits," is a Recall of 0.6. Recall is prioritized when you want to avoid False-Negatives.

If you were in Fascist-Land and you were helping anyone who wanted to escape on a boat, and anyone who wasn't a resident of Fascist-Land who didn't get on boat would be executed, you'd prioritize **Recall** for citizens of other countries.

Accuracy: $\frac{Correct Predictions}{Total Predictions}$ or $\frac{TP+TN}{TP+TN+FP+FN}$ A measurement of True-Positives divided by the sum of all true cases of that particular classification. "We had 15 animals, five rabbits and ten chinchillas, you identified 3 rabbits as rabbits, and nine chinchillas as chinchillas, so you were right 80% of the time" is an accuracy of 0.8. Accuracy is rarely prioritized because it does not provide any context for the consequences of False-Negatives and False-Positives. If you were developing a test for a medical condition that only one in a million people had, you could have a highly accurate model by doing nothing at all other than predicting somebody didn't have the medical condition. Unlike the others, Accuracy is the same value for all classifications in a multiclass model.

If you were non-Fascist-Land citizens back to their home country, and if you sent them to the wrong country then that country would send them to Fascist-Land, you'd prioritize **Accuracy**.

f1-score: $2 \times \frac{Precision \times Recall}{Precision + Recall}$ or $\frac{TP}{TP + \frac{1}{2}(FP + FN)}$ The F1-Score is a way to balance both Precision and Recall. It's a way to rate the entire model with a single number rather than considering different options between multiple numbers. F1-Score is what you'd prioritize when you want the model to be as useful as possible, but there are no significant difference in consequences between False-Negatives and False-Positives.

Weighted average: $\sum_{AllClasses} Score_{Class} \times \frac{N_{Class}}{N_{Total Dataset}}$ Because Recall, Precision and F1-Scores are all specific to a single classification within a multiclass model, and because the model's performance on a single class's score is highly dependent on the amount of training data available for that class, a way is needed to score the entire multiclass model that takes data imbalances into account. If I show you a thousand pictures of rabbits, and only a single picture of a chinchilla and a pica, you'd likely guess rabbits very well, but maybe get confused

5.1.2 Basic TF-IDF vectorizer

Now, I'm going to construct the first vectorizer using the TfidfVectorizer technique, and test different options using the model MultinomialNB and K-fold Cross Validation with f1 as the scorer. The goal in this section is to find the best options for TfidfVectorizer. I will vectorize the corpus with three different options:

1. No special options.
2. Removing stopwords
3. Removing stems

This first vectorization doesn't remove any stopwords or stem the tweet. It just takes the words as they are.

In [89]:
```python
# For breaking up the the selected Tweet string into tokens.
def to_tokens(document):
    tokens = tokenizer.tokenize(document)
    return [token for token in tokens]
```

```
In [90]: # Instantiate a vectorizer with max_features=10
         # Default token pattern
         tfidf = TfidfVectorizer(max_features=10,
                                 tokenizer = to_tokens,
                                 token_pattern = None)

         # Fit the vectorizer on xtrain["Tweet"] and numerically transform it
         # into TF-IDF.
         xtrain_tfidf = tfidf.fit_transform(xtrain['Tweet'])

         pd.set_option('display.precision', 5)
```

```
In [91]: # Testing to see if the data is in the correct format
         assert xtrain_tfidf.shape[0] == xtrain.shape[0]
         assert xtrain_tfidf.shape[1] == 10
```

```
In [92]: # Instantiate a MultinomialNB classifier
         baseline_model = MultinomialNB()

         # Evaluate the classifier on xtrain_tfidf and ytrain
         baseline_cv = cross_val_score(baseline_model, xtrain_tfidf, ytrain,
                                       scoring = 'f1_weighted')
         baseline_cv
```

```
Out[92]: array([0.28654993, 0.29164937, 0.28790502, 0.29275851, 0.28812374])
```

```
In [93]: print('K-Fold Cross Validation Weighted F1-Scores for:')
         print("Baseline: {0:.3f}".format(baseline_cv.mean()))
```

```
K-Fold Cross Validation Weighted F1-Scores for:
Baseline: 0.289
```

An f1 score of less than .3 is bad. I can do better.

5.1.3 TF-IDF vectorizer with Stopwords Removed

I removed the stopwords earlier in the Jupyter Notebook. Now I'll see what a difference it makes to the F1-score.

In [94]:
```
# Fit the vectorizer on xtrain["text"] and transform it
xtrain_tfidf = tfidf.fit_transform(xtrain["Tidy Tweet"])

# Visually inspect the vectorized data
pd.DataFrame.sparse.from_spmatrix(xtrain_tfidf,
                                  columns=tfidf.get_feature_names_out()).head(
```

Out[94]:

	apple	austin	google	ipad	ipad2	iphone	new	pop_up	store	sxsw
0	0.000	0.741	0.000	0.632	0.000	0.000	0.000	0.000	0.000	0.227
1	0.000	0.000	0.912	0.000	0.000	0.000	0.000	0.000	0.000	0.411
2	0.000	0.000	0.000	0.000	0.941	0.000	0.000	0.000	0.000	0.339
3	0.000	0.000	0.912	0.000	0.000	0.000	0.000	0.000	0.000	0.411
4	0.000	0.000	0.000	0.000	0.941	0.000	0.000	0.000	0.000	0.339

In [95]:
```
# Evaluate the classifier on xtrain_tfidf and ytrain
stopwords_removed_cv = cross_val_score(baseline_model, xtrain_tfidf,
                                       ytrain, scoring = 'f1_weighted')
stopwords_removed_cv
```

Out[95]:
```
array([0.38415608, 0.40871075, 0.39261732, 0.39315338, 0.4085535 ])
```

In [96]:
```
# Now I'll see how the two models compare.
print('K-Fold Cross Validation Weighted F1-Scores for:')
print("Baseline:          {0:.3f}".format(baseline_cv.mean()))
print("Stopwords removed: {0:.3f}".format(stopwords_removed_cv.mean()))
```

```
K-Fold Cross Validation Weighted F1-Scores for:
Baseline:          0.289
Stopwords removed: 0.397
```

Removing a standard list of stop words actually caused an increase in the model's accuracy. Now I'll see how stemming affects the model.

5.1.4 TF-IDF vectorizer with Stemmed Stopwords Removed

Again, I already stemmed the data already in the notebook. Now I'm going to pass the stemmed data into the model.

In [97]:
```
# Fit the vectorizer on xtrain["text"] and transform it
xtrain_tfidf = tfidf.fit_transform(xtrain["Stemmed Tweet"])
xtest_tfidf = tfidf.fit_transform(xtest["Stemmed Tweet"])
```

In [98]:
```
stemmed_cv = cross_val_score(baseline_model, xtrain_tfidf, ytrain,
                              scoring = 'f1_weighted')
stemmed_cv
```

Out[98]:
```
array([0.38415608, 0.40792695, 0.39261732, 0.39297933, 0.40882299])
```

In [99]:
```
print('K-Fold Cross Validation Weighted F1-Scores for:')
print("Baseline:          {0:.3f}".format(baseline_cv.mean()))
print("Stopwords removed: {0:.3f}".format(stopwords_removed_cv.mean()))
print("Stemmed:           {0:.3f}".format(stemmed_cv.mean()))
mnb_tfidf_xtrain_kfold = stemmed_cv
```

```
K-Fold Cross Validation Weighted F1-Scores for:
Baseline:          0.289
Stopwords removed: 0.397
Stemmed:           0.397
```

Not much better than removing stopwords. I'll have to see how this vectorization method does under Train and Test conditions with the various models below. I will, however, use the stemmed version exclusively from here forward though.

5.2 Build Bag-Of-Words Vectorizer

Bag-of-Words Vectorizor: A large bag of words with a count for how often each word appears in each tweet. It's a simple model, but sometimes it still yields a good result.

Unlike the section above, I will only used the stemmed tokens for this vectorization technique.

In [100]:
```
# To avoid meticulous editting, I will use this cell to define the text and
# tokens to be used in the vectorization and modeling below. Changing this cel
# will change the results below.
modstr = 'Stemmed Tweet'  # String to model
modtok = 'Stemmed Tokens' # Tokens to model
```

In [101]:
```
bow_vectorizer = CountVectorizer(max_df=0.90, min_df=2,
                                 max_features=10000)
df_bow = bow_vectorizer.fit_transform(df[modstr])
xtrain_bow = df_bow[ytrain.index]
xtest_bow  = df_bow[ytest.index]
print(xtrain_bow.shape, xtest_bow.shape)
```

```
(5296, 3432) (1324, 3432)
```

In [102]:
```
mnb_bow_xtrain_kfold = cross_val_score(baseline_model, xtrain_bow, ytrain,
                              scoring = 'f1_weighted')
mnb_bow_xtrain_kfold
```

Out[102]:
```
array([0.57274761, 0.58396229, 0.58662038, 0.56794822, 0.58197005])
```

In [103]:
```
print('K-Fold Cross Validation Weighted F1-Scores for:')
print("TF-IDF:       {0:.3f}".format(mnb_tfidf_xtrain_kfold.mean()))
print('Bag Of Words: {0:.3f}'.format(mnb_bow_xtrain_kfold.mean()))
```

```
K-Fold Cross Validation Weighted F1-Scores for:
TF-IDF:       0.397
Bag Of Words: 0.579
```

The K-Fold Cross Validation F1-score takes a significant jump from its TF-IDF counterpart. One of the other vectorization techniques will have to score high to beat Bag Of Words.

5.3 Build Word2Vec Vectorizer

Word2Vec Vectorizor: This vectorizer is probably the most interesting. It takes each word, or token, and turns it into a multi-dimensional vector. Then, in theory, simple arithmetic computations can yield known words. The classic example is: $King - Man + Woman = Queen$.

Whether this method leads to a better model or not isn't super important. This vectorizer will allow us to find correlations between words. I'll explain more about that in later sections.

In [104]:
```
vsize = 692
def word2vec_model(tokens):
    model = gensim.models.Word2Vec(
        tokens, # Panda series of lists with word tokens in it.
        vector_size = vsize,  # desired no. of features/independent variables
        window=5,  # context window size
        min_count=1,  # Ignores all words with total frequency lower than 2.
        sg=1,  # 1 for skip-gram model
        hs=0,
        negative=10,  # for negative sampling
        workers=32,  # no.of cores
        seed=34)
    model.train(tokens, total_examples=len(tokens), epochs=20)
    return model

model_w2v_df = word2vec_model(df[modtok])
```

In [105]:
```
def word_vector(model, tokens, size):
    vec = np.zeros(size).reshape((1, size))
    count = 0
    for word in tokens:
        try:
            vec += model.wv[word].reshape(1, size)
            count += 1
        except KeyError:  # handling the case where the token is not in
            continue      # vocabulary
    if count != 0:
        vec /= count
    return vec
```

In [106]:
```
def word2Vec(tokens, vector_size):
    wordvec_arrays = np.zeros((len(tokens), vector_size))
    array_index = 0
    for i in tokens.index:
        wordvec_arrays[array_index, :] = word_vector(model_w2v_df,
                                             tokens[i], vector_size)
        array_index += 1
    wordvec_df = pd.DataFrame(wordvec_arrays)
    print('Word2Vec Shape:', wordvec_df.shape)

    # PredictorScaler=StandardScaler()
    PredictorScaler = MinMaxScaler()

    # Storing the fit object for later reference
    PredictorScalerFit = PredictorScaler.fit(wordvec_df)

    # Generating the standardized values of X
    wordvec_norm = PredictorScalerFit.transform(wordvec_df)

    return wordvec_df, wordvec_norm
```

In [107]:
```
print('X-Train Word2Vec:')
xtrain_w2v, xtrain_w2v_norm = word2Vec(xtrain[modtok],
                                        vsize)

print('X-Test Word2Vec:')
xtest_w2v, xtest_w2v_norm = word2Vec(xtest[modtok],
                                      vsize)
```

```
X-Train Word2Vec:
Word2Vec Shape: (5296, 692)
X-Test Word2Vec:
Word2Vec Shape: (1324, 692)
```

In [108]:
```
mnb_w2v_xtrain_kfold = cross_val_score(baseline_model,
                                       xtrain_w2v_norm, ytrain,
                                       scoring = 'f1_weighted')
mnb_w2v_xtrain_kfold
```

Out[108]:
```
array([0.45827504, 0.44040136, 0.44579303, 0.4578815 , 0.45980801])
```

In [109]:
```
print('K-Fold Cross Validation Weighted F1-Scores for:')
print("TF-IDF:       {0:.3f}".format(mnb_tfidf_xtrain_kfold.mean()))
print('Bag Of Words: {0:.3f}'.format(mnb_bow_xtrain_kfold.mean()))
print('Word2Vec:     {0:.3f}'.format(mnb_w2v_xtrain_kfold.mean()))
```

```
K-Fold Cross Validation Weighted F1-Scores for:
TF-IDF:       0.397
Bag Of Words: 0.579
Word2Vec:     0.452
```

6 Positive and Negative Word Associations

As written above, the Word2Vec vectorization method converts each token to it's own multidimensional vector. With that completed, it doesn't take much to compare the different vectors to see which are pointed in the same direction. Likewise, we can see which vectors are pointed in the opposite direction. That's all there is to word association with this method.

However, the slicing and interpretation of the many ways you can run the association methods can get exhausting. So I set out to build a tool that could easily specify which target the method was pulling the word associations with, and easily switch the word. The construction of this tool began with creating multiple Word2Vec models and dataframes containing its vectors that can be easily referenced.

In the interest of keeping this notebook as brief as possible, I cut most of the DataFrame and Word2Vec models from this document. For each DataFrame and Word2Vec model I created, see the Jupyter Notebook student-Word2Vec-Workspace.ipynb (http://localhost:8888/notebooks/student-Word2Vec-Workspace.ipynb).

6.1 Creating Targeted Models and DataFrames

Below is a behemoth of a DataFrame. It slices up the dataset into subsets representing each of the twelve categories, both the DataFrames and Word2Vec models. The titles of each model represent shorthand to identify what it i. Each of the letters between the underscores is separated as such:

df/model = DataFrame or Word2Vec model

w2v = Word2Vec

neg/neu/pos = Negative, Neutral or Positive

ab/ap/gb/gp/a/g = Apple Brand, Apple Products, Google Brand, Google Products, Apple, Google

The different Word2Vec models and DataFrames are used for quick access to create visualizations.

In [110]:
```
# Google DataFrames and Word2Vec models for analysis

# For titles in visualizations
titles = {}

# All negative Google Tweets
df_w2v_neg_g = df.loc[(df['Company'] == 'Google') &
                      (df['Emotion'] == 'Negative')]
model_w2v_neg_g = word2vec_model(df_w2v_neg_g[modtok])

# All neutral Google Tweets
df_w2v_neu_g = df.loc[(df['Company'] == 'Google') &
                      (df['Emotion'] == 'Neutral')]
model_w2v_neu_g = word2vec_model(df_w2v_neu_g[modtok])

# All positive Google Tweets
df_w2v_pos_g = df.loc[(df['Company'] == 'Google') &
                      (df['Emotion'] == 'Positive')]
model_w2v_pos_g = word2vec_model(df_w2v_pos_g[modtok])

# For naming the DataFrames and Word2Vec models to create refernces for the
# visualizations
df_name  = ['df_w2v_neg_g',  'df_w2v_neu_g',    'df_w2v_pos_g']
mdl_name = ['model_w2v_neg_g', 'model_w2v_neu_g', 'model_w2v_pos_g']
titles_name = ['Negative', 'Neutral', 'Positive']

# Naming the DataFrames and Word2Vec models
for dfone, mdl, df_nm, mdl_nm, ttl_nm in zip(
                    [df_w2v_neg_g,  df_w2v_neu_g,    df_w2v_pos_g],
                    [model_w2v_neg_g, model_w2v_neu_g, model_w2v_pos_g],
                     df_name, mdl_name, titles_name):
    dfone.name = df_nm
    mdl.name = mdl_nm
    titles[dfone.name] = ttl_nm + ' - Google (Products & Brand)'
    titles[mdl.name] = ttl_nm + ' - Google (Products & Brand)'

# Creating a list of all of the above.
google_df = [df_w2v_neg_g, df_w2v_neu_g, df_w2v_pos_g]
```

And now Apple.

In [111]:
```
# Apple DataFrames and Word2Vec models for analysis

# All negative Apple Tweets
df_w2v_neg_a = df.loc[(df['Company'] == 'Apple') &
                      (df['Emotion'] == 'Negative')]
model_w2v_neg_a = word2vec_model(df_w2v_neg_a[modtok])

# All neutral Apple Tweets
df_w2v_neu_a = df.loc[(df['Company'] == 'Apple') &
                      (df['Emotion'] == 'Neutral')]
model_w2v_neu_a = word2vec_model(df_w2v_neu_a[modtok])

# All positive Apple Tweets
df_w2v_pos_a = df.loc[(df['Company'] == 'Apple') &
                      (df['Emotion'] == 'Positive')]
model_w2v_pos_a = word2vec_model(df_w2v_pos_a[modtok])

# For naming the DataFrames and Word2Vec models to create refernces for the
# visualizations
df_name  = ['df_w2v_neg_a',  'df_w2v_neu_a',    'df_w2v_pos_a']
mdl_name = ['model_w2v_neg_a', 'model_w2v_neu_a', 'model_w2v_pos_a']

# Naming the DataFrames and Word2Vec models
for dfone, mdl, df_nm, mdl_nm, ttl_nm in zip(
                    [df_w2v_neg_a,  df_w2v_neu_a,    df_w2v_pos_a],
                    [model_w2v_neg_a, model_w2v_neu_a, model_w2v_pos_a],
                     df_name, mdl_name, titles_name):
    dfone.name = df_nm
    mdl.name = mdl_nm
    titles[dfone.name] = ttl_nm + ' - Apple (Products & Brand)'
    titles[mdl.name] = ttl_nm + ' - Apple (Products & Brand)'

# Creating a list of all of the above.
apple_df  = [df_w2v_neg_a, df_w2v_neu_a, df_w2v_pos_a]
```

6.2 Word Association Functions

These next three functions create word clouds for a given word and slice of the dataset. The word clouds also list the word association correlation values.

In [112]:
```
# Checks to see if the word is in the give model, and returns the pair of
# association tuple lists.

def word_association(mdl, wrd):
    if wrd not in mdl.wv.key_to_index:
        string = 'The word "' + wrd.capitalize() + '" is not in the model '
        string += titles[mdl.name]
        return  string, ''
    pos = mdl.wv.most_similar(positive=wrd)
    neg = mdl.wv.most_similar(negative=wrd)
    return pos, neg
```

In [113]:
```
# Creates a pair of clouds that that show the top ten positively and negativel
# associated word in the given association list of association tuples, the wor
# and the title of the graph.

def word_assoc_cloud(pos_assoc, neg_assoc, word, title = '', subs = []):
    word_cap = word.capitalize()
    pos_lst = []
    neg_lst = []
    for i in range(len(pos_assoc)):
        pos_lst.append(pos_assoc[i][0])
        neg_lst.append(neg_assoc[i][0])
    pos_text = " ".join(i for i in pos_lst)
    neg_text = " ".join(i for i in neg_lst)
    wc_lst = []

    wc_lst.append(WordCloud(background_color="white").generate(pos_text))
    wc_lst.append(WordCloud(background_color="white").generate(neg_text))

    fig = plt.figure(figsize=(9, 4))
    fig.set_tight_layout(True)
    gs = fig.add_gridspec(1, 2)
    ax1 = fig.add_subplot(gs[0, 0])
    ax2 = fig.add_subplot(gs[0, 1])

    if subs == []:
        subtitles = ['Associated With '     + word_cap,
                     'Not Associated With ' + word_cap]
    else:
        subtitles = subs
    i = 0
    for ax in [ax1, ax2]:
        ax.spines['bottom'].set_color('0.5')
        ax.spines['top'].set_color('0.5')
        ax.spines['right'].set_color('0.5')
        ax.spines['left'].set_color('0.5')
        ax.patch.set_facecolor('0.1')
        ax.tick_params(axis='x', which='both', bottom = False,
                       labelbottom = False)
        ax.tick_params(axis='y', which='both', left = False,
                       labelleft = False)
        ax.yaxis.label.set_color('0.9')
        ax.xaxis.label.set_color('0.9')
        ax.margins(0.5)
        ax.imshow(wc_lst[i], interpolation='bilinear')
        ax.set_title(subtitles[i], fontsize=18)
        i += 1
    if title != '':
        fig.suptitle(title, fontsize=24)
    plt.show()
```

In [114]:
```
# Creates a word cloud graphic and lists underneath the words and their
# association similarity for the given word and model.

def word_association_visual(model_lst, word):
    if type(model_lst) is not list:
        temp = []
        temp.append(model_lst)
        model_lst = temp
    for model in model_lst:
        pos_tup_lst, neg_tup_lst = word_association(model, word)
        if type(pos_tup_lst) is str:
            print(pos_tup_lst)
            continue

        first = 'Word Clouds for\n'
        mid = titles[model.name] + ' Tweets\n'
        last  =  'Associated With The Word "{}"'.format(word.capitalize())
        model_title = first + mid + last
        word_assoc_cloud(pos_tup_lst, neg_tup_lst, word,
            title = model_title)

        max_len = 0
        for i in range(len(pos_tup_lst)):
            if len(pos_tup_lst[i][0]) > max_len:
                max_len = len(pos_tup_lst[i][0])
            if len(neg_tup_lst[i][0]) > max_len:
                max_len = len(neg_tup_lst[i][0])
        for i in range(len(pos_tup_lst)):
            pos_txt = pos_tup_lst[i][0]
            neg_txt = neg_tup_lst[i][0]
            spaces_pos = ''
            spaces_neg = ''
            between_space = ''
            pos_num = str('%.3f' % round(pos_tup_lst[i][1],3))
            neg_num = str('%.3f' % round(neg_tup_lst[i][1],3))
            num_spaces_pos = max_len - len(pos_txt) + 4
            num_spaces_neg = max_len - len(neg_txt) + 4
            if float(pos_num) < 0:
                num_spaces_pos -= 1
            if float(neg_num) < 0:
                num_spaces_neg -= 1
            for i in range(num_spaces_pos):
                spaces_pos += ' '
            for i in range(num_spaces_neg):
                spaces_neg += ' '
            txt =  '(' + pos_txt + spaces_pos + pos_num + ')'
            between = 40 - len(txt)
            for i in range(between):
                between_space += ' '
            txt += between_space + '(' + neg_txt + spaces_neg + neg_num + ')'
            print(txt)
        print()
    return
```

The function below lists the top ten hashtags used for the given model slice and word. Similar to the function above, it gives a word cloud of the hashtags and a tally below that shows how many times that hastag was used in the given model.

```
In [115]: def hashtag_search(model_lst, word):
              if type(model_lst) is not list:
                  temp = []
                  temp.append(model_lst)
                  model_lst = temp
              for dfone in model_lst:
                  hashtagbag = []
                  num_tweets = 0
                  for i in dfone.index:
                      if (word in dfone.loc[i, 'Tokens']):
                          num_tweets += 1
                          for hashtag in dfone.loc[i, 'Hashtags']:
                              hashtagbag.append(hashtag)
                  count = tally(hashtagbag)
                  t = ('Word Cloud For Included Hashtags In\n' +
                        titles[dfone.name] + ' Tweets\n' +
                        'That Include The Word "' + word.capitalize() + '"\n')
                  tot = str(len(dfone))
                  if len(hashtagbag) !=0:
                      wc(hashtagbag, title = t, re = r"#(?:\w[\w']+)")
                      print('For the', titles[dfone.name], 'set of', tot,'tweets:')
                      print('The word "' + word.capitalize() +'" is included in',
                            str(num_tweets) + '/' + tot, 'tweets.')
                      print()
                      print('Top ten Hashtags included with the word "' +
                            word.capitalize() +'" in this dataset:')
                      for pair in count[:10]:
                          print(pair)
                  else:
                      string = 'The word "' + word.capitalize() + '" is not in the '
                      string += 'DataFrame ' + titles[dfone.name]
                      print(string)

              return count
```

6.3 Word Association Tool – Workspace

Here is the fruit of my labor. With only two terms, I can now provide word clouds and a list of ten positive association and negative association words. I can do this for multiple slices of the dataset at once, and I can do it for all the data in the set.

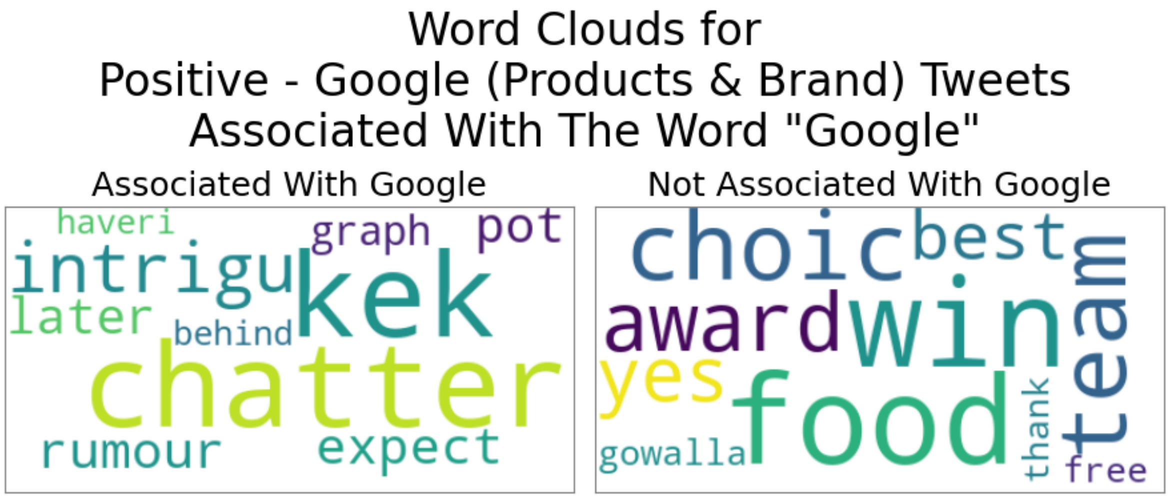
I performed multiple searches through multiple words (http://localhost:8888/notebooks/student-Word2Vec-Workspace.ipynb) and it's clear the search could continue forever. To limit the scope to what is appropriate for the time I'm given, I performed analysis on a single keyword for four words: The #1 keyword in each of the following categories:

1. Google Brand : Google
2. Google Products : Android

3. Apple Brand : Apple
4. Apple Products : iPad

Below I list the word associations for each word for all the available slices of the DataFrame they exist in.

```
In [116]: word_association_visual(model_w2v_pos_g, 'google');
```



```
(kek        0.979)                  (win        -0.368)
(chatter    0.979)                  (food       -0.387)
(intrigu    0.974)                  (choic      -0.388)
(expect     0.974)                  (team       -0.397)
(rumour     0.971)                  (award      -0.415)
(later      0.965)                  (yes        -0.430)
(pot        0.965)                  (best       -0.439)
(graph      0.964)                  (gowalla    -0.443)
(behind     0.963)                  (free       -0.445)
(haveri     0.963)                  (thank      -0.451)
```

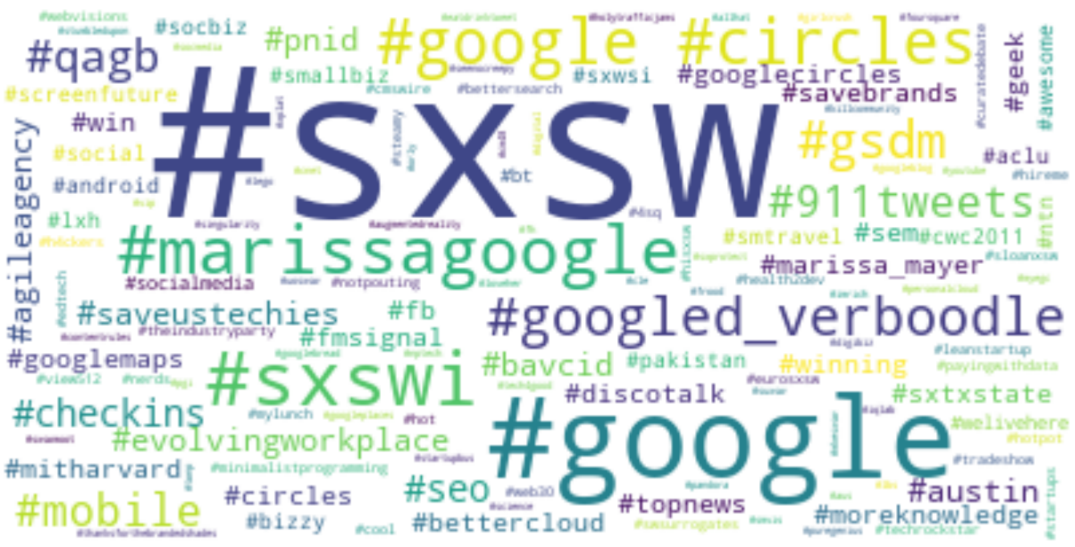
6.4 Hashtag Counter Tool – Workspace

Word associations throughout a dataset are great, but what about intentional associations to link up with other like minded people on Twitter? Enter the hashtag.

As described above, this function lists all the top ten hashtags used in the given word for each data slice. Below I list the hashtags for each word I'm performing this analysis on for all the available slices of the DataFrame they exist in.

```
In [117]: hash_count = hashtag_search(df_w2v_pos_g, 'google')
```

Word Cloud For Included Hashtags In Positive - Google (Products & Brand) Tweets That Include The Word "Google"



```
For the Positive - Google (Products & Brand) set of 662 tweets:
The word "Google" is included in 541/662 tweets.

Top ten Hashtags included with the word "Google" in this dataset:
('#sxsw', 541)
('#google', 55)
('#sxswi', 15)
('#circles', 12)
('#marissagoogle', 12)
('#googled_verboodle', 8)
('#gsdm', 8)
('#qagb', 6)
('#911tweets', 5)
('#mobile', 5)
```

6.5 Word Associations Conclusions.

I read each of the lists [(found here) (http://localhost:8888/notebooks/student-Word2Vec-Workspace.ipynb)](http://localhost:8888/notebooks/student-Word2Vec-Workspace.ipynb) and grabbed words and hashtags that were most practically applicable to the word triggering the lists. Here is a list of all of the associated words and hashtags.

In [178]:
```
# Google
key_assoc_negative_google  = ['trend', 'buzz', 'app', 'android',
                              'imag', 'maps', 'marissa_mayer']
key_assoc_neutral_google   = ['marissa_mayer', 'maps']
key_assoc_positive_google  = ['maps', 'navi', 'expect', 'evolv',
                              'fast', 'cool', 'video', 'circles']

hashtags_negative_google   = ['#circles', '#social', '#bing']
hashtags_neutral_google    = ['#circles', '#facebook', '#socialmedia',
                              '#marissagoogle']
hashtags_positive_google   = ['#circles', '#marissagoogle']

# Android
key_assoc_negative_android = ['google', 'app', 'phon',
                               'updat', 'vers', 'opt']
key_assoc_neutral_android  = ['team', 'award', 'samsung', 'chrome', 'detail']
key_assoc_positive_android = ['congrat', 'job', 'huzzah', 'awesom', 'thank',
                               'android_app', 'choic']

hashtags_negative_android  = ['#fail']
hashtags_neutral_android   = ['#tech', '#app']
hashtags_positive_android  = ['#teamandroid']

# Apple
key_assoc_negative_apple   = ['apple_store', 'store', 'macbook', 'facist',
                               'stylish', 'flip']
key_assoc_neutral_apple    = ['geekfest', 'technolog', 'rumor', 'store',
                               'realest', 'bait']
key_assoc_positive_apple   = ['sxsw', 'excus', 'model', 'genius', 'demand',
                               'retail', 'scene', 'debut', 'brilliant']

hashtags_negative_apple    = ['#sxsw', '#apple', '#flipboard', '#ipad2']
hashtags_neutral_apple     = ['#sxsw', '#apple', '#ipad2', '#tech',
                               '#apple_store']
hashtags_positive_apple    = ['#sxsw', '#apple', '#tech', '#technology']

# iPad
key_assoc_negative_ipad    = ['media', 'app', ]
key_assoc_neutral_ipad     = ['bliss', 'structur', 'book', 'ultim', 'present',
                               'scheme']
key_assoc_positive_ipad    = ['futur', 'mom', 'envi', 'media']

hashtags_negative_ipad     = ['#sxsw', '#ipad', '#tapworthy']
hashtags_neutral_ipad      = ['#sxsw', '#ipad', '#apple', '#tapworthy',
                               '#touchingstories', '#art']
hashtags_positive_ipad     = ['#sxsw', '#ipad', '#apple', '#tapworthy',
                               '#poursite', '#newsapps']

# Combinations for function below.
google_associations  = [key_assoc_negative_google, key_assoc_neutral_google,
                        key_assoc_positive_google]
android_associations = [key_assoc_negative_android, key_assoc_neutral_android,
                        key_assoc_positive_android]

apple_associations   = [key_assoc_negative_apple, key_assoc_neutral_apple,
                        key_assoc_positive_apple]
```

```
ipad_associations    = [key_assoc_negative_ipad, key_assoc_neutral_ipad,
                        key_assoc_positive_ipad]
```

The function below displays lists of how often each word in a list appears in tweets that include the associated word. It splits the lists into the three emotions, negative, neutral and positive. For example, these lists will tell you how many times the word "Trend" appears in negative tweets that include the word "Google."

In [119]:
```
def association_count(list_of_lists, model_list, word):
    emo = ['Negative', 'Neutral', 'Positive']
    colors = ['Reds', 'Greens', 'Blues']
    for e, lst, mdl, c in zip(emo, list_of_lists, model_list, colors):
        tweet_count = 0
        count = []
        for i in mdl.index:
            for stm in mdl.loc[i,'Stemmed Tokens']:
                if word == stm:
                    tweet_count +=1
                    for wrd in lst:
                        if wrd in mdl.loc[i,'Stemmed Tweet']:
                            count.append(wrd)
        if tweet_count == 0:
            print('There are no occurences of "' + word.capitalize() +
                  '" in the ' + e +' emotion tweets.')
        else:
            string = 'Associated word occurrences found in the '
            string += str(tweet_count) + ' ' + e +' emotion tweets'
            string_two = 'that include the word "' + word.capitalize() + '":'
            print(string)
            print(string_two)
            column_title = e + ' Tweets that include "'+ word.capitalize() + '
            count_total = pd.DataFrame(count)
            count_total.rename(columns = {0:'Word'}, inplace = True)
            count_total = count_total.groupby('Word')['Word'].count()
            count_total = pd.DataFrame(count_total)
            count_total.rename(columns = {'Word':column_title}, inplace = True
            count_total = count_total.sort_values(column_title,
                                                  ascending = False)
            count_total.reset_index(drop=False, inplace=True)
            display(count_total.style.background_gradient(cmap=c))
    return None
```

6.5.1 "Google" Word Association Analysis

In [179]:
```python
association_count(google_associations, google_df, 'google')
```

```
Associated word occurrences found in the 122 Negative emotion tweets
that include the word "Google":
```

	Word	Negative Tweets that include "Google"
0	maps	10
1	app	8
2	marissa_mayer	8
3	buzz	6
4	android	1
5	imag	1
6	trend	1

```
Associated word occurrences found in the 1368 Neutral emotion tweets
that include the word "Google":
```

	Word	Neutral Tweets that include "Google"
0	marissa_mayer	125
1	maps	97

```
Associated word occurrences found in the 581 Positive emotion tweets
that include the word "Google":
```

	Word	Positive Tweets that include "Google"
0	circles	96
1	maps	93
2	cool	26
3	fast	11
4	navi	7
5	video	5
6	evolv	4
7	expect	4

"Google" Word Association Analysis Takeaways:

1. People like to tweet about Google Maps. Their opinions of the product are mostly positive.
2. Tweets regarding Marissa Meyer are more neutral than anything else.
3. People think Google is "cool" and associate it with being "fast".

6.5.2 "Android" Word Association Analysis

In [121]:
```
association_count(android_associations, google_df, 'android')
```

```
Associated word occurrences found in the 12 Negative emotion tweets
that include the word "Android":
```

	Word	Negative Tweets that include "Android"
0	app	5
1	phon	3
2	opt	2
3	google	1
4	updat	1
5	vers	1

```
Associated word occurrences found in the 144 Neutral emotion tweets
that include the word "Android":
```

	Word	Neutral Tweets that include "Android"
0	team	20
1	samsung	9
2	chrome	8
3	detail	7
4	award	6

```
Associated word occurrences found in the 107 Positive emotion tweets
that include the word "Android":
```

	Word	Positive Tweets that include "Android"
0	thank	12
1	choic	11
2	android_app	10
3	congrat	5
4	awesom	3
5	job	3
6	huzzah	1

"Android" Word Association Analysis Takeaways:

1. People tweet complaints about apps on android, sometimes.
2. People who have android products seem to be happy with their choice.

6.5.3 "Apple" Word Association Analysis

```
In [122]: association_count(apple_associations, apple_df, 'apple')
```

```
Associated word occurrences found in the 81 Negative emotion tweets
that include the word "Apple":
```

	Word	Negative Tweets that include "Apple"
0	store	20
1	flip	5
2	apple_store	2
3	macbook	2
4	facist	1
5	stylish	1

```
Associated word occurrences found in the 712 Neutral emotion tweets
that include the word "Apple":
```

	Word	Neutral Tweets that include "Apple"
0	store	402
1	rumor	47
2	technolog	25
3	geekfest	16
4	realest	2
5	bait	1

```
Associated word occurrences found in the 600 Positive emotion tweets
that include the word "Apple":
```

	Word	Positive Tweets that include "Apple"
0	sxsw	600
1	genius	16
2	brilliant	14
3	retail	11
4	demand	3
5	model	3
6	scene	2
7	debut	1
8	excus	1

"Apple" Word Association Analysis Takeaways:

1. People are enjoying the experience of being at SXSW and associate it with the Apple store.
2. People associate a lot of positive words with Apple, like genius and brilliant.

6.5.4 "iPad" Word Association Analysis

In [123]:
```
association_count(ipad_associations, apple_df, 'ipad')
```

```
Associated word occurrences found in the 106 Negative emotion tweets
that include the word "Ipad":
```

	Word	Negative Tweets that include "Ipad"
0	app	33
1	media	3

```
Associated word occurrences found in the 603 Neutral emotion tweets
that include the word "Ipad":
```

	Word	Neutral Tweets that include "Ipad"
0	book	19
1	present	5
2	scheme	2
3	ultim	2
4	bliss	1
5	structur	1

```
Associated word occurrences found in the 462 Positive emotion tweets
that include the word "Ipad":
```

	Word	Positive Tweets that include "Ipad"
0	envi	11
1	mom	7
2	futur	6
3	media	3

"iPad" Word Association Analysis Takeaways:

1. People are complaining about apps on iPad, likely they want apps that are not available yet.
2. People plan to read books on their iPad.
3. People want to give iPads as presents, possibly to their mom.
4. People associate the future with iPad.

So here are the word analyses of the top keyword in each of the company / brand or product categories. Rather than digging into the set any further, I think it makes a lot more sense to empower the client, Google, to perform these searches on their own. I propose that Google hires our company to turn this analysis into a tool of its own. It will empower not just data scientists to search for word associations, but brand managers, marketers, public relations specialists, even HR managers looking for qualities in who to hire.

I recommend we move forward to make this tool accessible to multiple, non-python coders, at Google.

7 Building Models

It's finally time to build the classification models. I will fit each vectorization technique with the following models:

1. Multinomial Naive Bayes
2. Logistic Regression
3. Support Vector Machine
4. RandomForest
5. XGBoost

I'll explain each of the models below in the same section in which they are created. Before I start, here's one last preview of the DataFrame I am about to model.

In [124]:
```
df[[modstr, modtok, 'Hashtags',
            'Keywords', 'Target Text', 'Target']].head()
```

Out[124]:

	Stemmed Tweet	Stemmed Tokens	Hashtags	Keywords	Target Text	Target
0	wesley83 3g iphone hrs tweet rise_austin dead need upgrad plugin station sxsw	[wesley83, 3g, iphone, hrs, tweet, rise_austin, dead, need, upgrad, plugin, station, sxsw]	[#rise_austin, #sxsw]	iPhone	Negative - Apple Products	3
1	jessede know fludapp awesom ipad iphone_app like appreci design also give free ts sxsw	[jessede, know, fludapp, awesom, ipad, iphone_app, like, appreci, design, also, give, free, ts, sxsw]	[#sxsw]	iPad iPhone_App	Positive - Apple Products	5
2	swonderlin wait ipad2 also sale sxsw	[swonderlin, wait, ipad2, also, sale, sxsw]	[#ipad2, #sxsw]	iPad2	Positive - Apple Products	5
3	sxsw hope year festiv crashi year iphone_app sxsw	[sxsw, hope, year, festiv, crashi, year, iphone_app, sxsw]	[#sxsw]	iPhone_App	Negative - Apple Products	3
4	sxtxstate great stuff fri sxsw marissa_mayer google tim_o_reilli tech book confer matt_mullenweg wordpress	[sxtxstate, great, stuff, fri, sxsw, marissa_mayer, google, tim_o_reilli, tech, book, confer, matt_mullenweg, wordpress]	[#sxsw]	Google Marissa_Mayer	Positive - Google Brand	8

Here I create a new DataFrame that I will add all of the pertinent model stats to for an easy comparison.

In [125]:
```
# Creating a DataFrame to keep all of this information
scores = pd.DataFrame(columns = ['Model', 'Vectorization',
        'K-Fold - F1 Weighted', 'Precision', 'Recall',
        'Accuracy', 'F1 Weighted Average'])
```

7.1 Multinomial Naive Bayes

I wrote about how this model work above. I'll repeat it here:

Multinomial Naive Bayes model: A multi-algorithmic model that is often used in NLP. It uses multiple algorithms that calculate the probability of each classification and returns the highest classification probability as the prediction.

In [126]:
```
# Multinomial Naive Bayes, TF-IDF

# Fit the model.
baseline_model.fit(xtrain_tfidf, ytrain)

# Peform prediction on ytest.
prediction = baseline_model.predict(xtest_tfidf)

# Calculate Statistics.
df_row = ['Multinomial Naive Bayes', 'TF-IDF', mnb_tfidf_xtrain_kfold.mean()]
df_row.append(precision_score(ytest, prediction, average = 'weighted',
                              zero_division = 0))
df_row.append(recall_score(ytest, prediction, average = 'weighted'))
df_row.append(accuracy_score(ytest, prediction))
df_row.append(f1_score(ytest, prediction, average = 'weighted'))

# Add to scores DataFrame.
scores.loc[len(scores)] = df_row
```

In [127]:
```
# Multinomial Naive Bayes, Bag of Words

# Fit the model.
baseline_model.fit(xtrain_bow, ytrain)

# Peform prediction on ytest.
prediction = baseline_model.predict(xtest_bow)

# Calculate Statistics.
df_row = ['Multinomial Naive Bayes',
          'Bag-Of-Words', mnb_bow_xtrain_kfold.mean()]
df_row.append(precision_score(ytest, prediction, average = 'weighted',
                              zero_division = 0))
df_row.append(recall_score(ytest, prediction, average = 'weighted'))
df_row.append(accuracy_score(ytest, prediction))
df_row.append(f1_score(ytest, prediction, average = 'weighted'))

# Add to scores DataFrame.
scores.loc[len(scores)] = df_row
```

In [128]:
```
# Multinomial Naive Bayes, Word2Vec

# Fit the model.
baseline_model.fit(xtrain_w2v_norm, ytrain)

# Peform prediction on ytest.
prediction = baseline_model.predict(xtest_w2v_norm)

# Calculate Statistics.
df_row = ['Multinomial Naive Bayes', 'Word2Vec', mnb_w2v_xtrain_kfold.mean()]
df_row.append(precision_score(ytest, prediction, average = 'weighted',
                              zero_division = 0))
df_row.append(recall_score(ytest, prediction, average = 'weighted'))
df_row.append(accuracy_score(ytest, prediction))
df_row.append(f1_score(ytest, prediction, average = 'weighted'))

# Add to scores DataFrame.
scores.loc[len(scores)] = df_row
```

In [129]:
```
scores
```

Out[129]:

	Model	Vectorization	K-Fold - F1 Weighted	Precision	Recall	Accuracy	F1 Weighted Average
0	Multinomial Naive Bayes	TF-IDF	0.397	0.359	0.494	0.494	0.394
1	Multinomial Naive Bayes	Bag-Of-Words	0.579	0.571	0.576	0.576	0.560
2	Multinomial Naive Bayes	Word2Vec	0.452	0.461	0.446	0.446	0.438

To analyze the first models created, it's worth a moment to restate the business problem.

Client: Google **Problem**: Some people view Google more negatively than their chief competitor, Apple. Google needs a tool to help measure the publics' sentiment toward both Apple and Google to strategize for a more positive public sentiment. This tool will offer Google insight to guide their business decisions.

So for this tool to be effective, recall is the most important metric. Google will want to identify as many True-Positive cases as possible. However, it is reasonable to balance this with precision as well, because I do not want the model to over predict any single category. Therefore, F1-score is the most reasonable metric to use to evaluate the model by a single score.

In general, any of the scores below 50% is more or less useless, even in a multiclass model. So the Word2Vec and TF-IDF vectorizations did not yield useable results. A recall of less that .5 means that the model predicted less than half of the tweets in each category correct. So the only vectoriztion technique to consider in this first batch is Bag-Of-Words.

The Bag-Of-Words is slightly better than .5. There is room for improvement. Hopefully the next few models will show improvement.

7.2 Logistic Regression

Logistic Regression: This model creates an "S" curve that predicts the likelihood of any one class. It fits the Sigmoid function over each class: $f(x) = \frac{1}{1+e^{-x}}$. To fit, it uses maximum likelihood estimation, instead of ordinary least squares.

```
In [130]: # Setting up the model.
          lreg = LogisticRegression(solver='lbfgs', max_iter=8000)
```

```
In [131]: # Calculating Cross Val Score, TF-IDF
          log_tfidf_train = cross_val_score(lreg, xtrain_tfidf, ytrain,
                                              scoring = 'f1_weighted')
          log_tfidf_train.mean()

Out[131]: 0.4431310526981268
```

```
In [132]: # Calculating Cross Val Score, Bag of Words
          log_bow_xtrain_kfold = cross_val_score(lreg, xtrain_bow, ytrain,
                                              scoring = 'f1_weighted')
          log_bow_xtrain_kfold.mean()

Out[132]: 0.6187898378331784
```

```
In [133]: # Calculating Cross Val Score, Word2Vec
          log_w2v_xtrain_kfold = cross_val_score(lreg, xtrain_w2v_norm, ytrain,
                                              scoring = 'f1_weighted')
          log_w2v_xtrain_kfold.mean()

Out[133]: 0.5944384437555141
```

In [134]:
```
# Logistic Regression, TF-IDF

# Fit the model.
lreg.fit(xtrain_tfidf, ytrain)

# Peform prediction on ytest.
prediction = lreg.predict(xtest_tfidf)

# Calculate Statistics.
df_row = ['Logistic Regression', 'TF-IDF', log_tfidf_train.mean()]
df_row.append(precision_score(ytest, prediction, average = 'weighted',
                              zero_division = 0))
df_row.append(recall_score(ytest, prediction, average = 'weighted'))
df_row.append(accuracy_score(ytest, prediction))
df_row.append(f1_score(ytest, prediction, average = 'weighted'))


# Add to scores DataFrame.
scores.loc[len(scores)] = df_row
```

In [135]:
```
# Logistic Regression, Bag of Words

# Fit the model.
lreg.fit(xtrain_bow, ytrain)

# Peform prediction on ytest.
prediction = lreg.predict(xtest_bow)

# Calculate Statistics.
df_row = ['Logistic Regression', 'Bag-Of-Words', log_bow_xtrain_kfold.mean()]
df_row.append(precision_score(ytest, prediction, average = 'weighted',
                              zero_division = 0))
df_row.append(recall_score(ytest, prediction, average = 'weighted'))
df_row.append(accuracy_score(ytest, prediction))
df_row.append(f1_score(ytest, prediction, average = 'weighted'))


# Add to scores DataFrame.
scores.loc[len(scores)] = df_row
```

In [136]:
```
# Logistic Regression, Word2Vec

# Fit the model.
lreg.fit(xtrain_w2v, ytrain)

# Peform prediction on ytest.
prediction = lreg.predict(xtest_w2v)

# Calculate Statistics.
df_row = ['Logistic Regression', 'Word2Vec', log_w2v_xtrain_kfold.mean()]
df_row.append(precision_score(ytest, prediction, average = 'weighted',
                              zero_division = 0))
df_row.append(recall_score(ytest, prediction, average = 'weighted'))
df_row.append(accuracy_score(ytest, prediction))
df_row.append(f1_score(ytest, prediction, average = 'weighted'))


# Add to scores DataFrame.
scores.loc[len(scores)] = df_row
```

In [137]:
```
scores
```

Out[137]:

	Model	Vectorization	K-Fold - F1 Weighted	Precision	Recall	Accuracy	F1 Weighted Average
0	Multinomial Naive Bayes	TF-IDF	0.397	0.359	0.494	0.494	0.394
1	Multinomial Naive Bayes	Bag-Of-Words	0.579	0.571	0.576	0.576	0.560
2	Multinomial Naive Bayes	Word2Vec	0.452	0.461	0.446	0.446	0.438
3	Logistic Regression	TF-IDF	0.443	0.430	0.505	0.505	0.447
4	Logistic Regression	Bag-Of-Words	0.619	0.628	0.639	0.639	0.625
5	Logistic Regression	Word2Vec	0.594	0.585	0.595	0.595	0.562

These Logistic Regression models are an improvement over the Multinomial Naive Bayes for each vectorization technique. TF-IDF still didn't get above the halfway mark for the F1-score, but both Bag-Of-Words and Word2Vec did, and the Bag-Of-Words model's improvement put the values into usable range. The recall score for this model shows that 64% of the categories were identified correctly, 36% were not. This is a good contender for the final model, but I'll see if I can get the scores even higher.

7.3 Support Vector Machine

Support Vector Machine: Support vector machine finds a hyperplane that is fit to all the points in the multiclass. That's it, a 3D plane between all the points.

```
In [138]: # Setting up the model.
          svc = svm.SVC(kernel='linear', C=1, probability=True)
```

```
In [139]: # Calculating Cross Val Score, TF-IDF
          svc_tfidf_xtrain_kfold = cross_val_score(svc, xtrain_tfidf, ytrain,
                                                   scoring = 'f1_weighted')
          svc_tfidf_xtrain_kfold.mean()

Out[139]: 0.4196044379366891
```

```
In [140]: # Calculating Cross Val Score, Bag of Words
          svc_bow_xtrain_kfold = cross_val_score(svc, xtrain_bow, ytrain,
                                                 scoring = 'f1_weighted')
          svc_bow_xtrain_kfold.mean()

Out[140]: 0.6105835905438862
```

```
In [141]: # Calculating Cross Val Score, Word2Vec
          svc_w2v_xtrain_kfold = cross_val_score(svc, xtrain_w2v, ytrain,
                                                 scoring = 'f1_weighted')
          svc_w2v_xtrain_kfold.mean()

Out[141]: 0.5821919006371523
```

```
In [142]: # Support Vector Machine, TF-IDF

          # Fit the model.
          svc.fit(xtrain_tfidf, ytrain)

          # Peform prediction on ytest.
          prediction = svc.predict(xtest_tfidf)

          # Calculate Statistics.
          df_row = ['Support Vector Machine', 'TF-IDF', svc_tfidf_xtrain_kfold.mean()]
          df_row.append(precision_score(ytest, prediction, average = 'weighted',
                                        zero_division = 0))
          df_row.append(recall_score(ytest, prediction, average = 'weighted'))
          df_row.append(accuracy_score(ytest, prediction))
          df_row.append(f1_score(ytest, prediction, average = 'weighted'))


          # Add to scores DataFrame.
          scores.loc[len(scores)] = df_row
```

In [143]:
```
# Support Vector Machine, Bag of Words

# Fit the model.
svc.fit(xtrain_bow, ytrain)

# Peform prediction on ytest.
prediction = svc.predict(xtest_bow)

# Calculate Statistics.
df_row = ['Support Vector Machine', 'Bag-Of-Words', svc_bow_xtrain_kfold.mean(
df_row.append(precision_score(ytest, prediction, average = 'weighted',
                              zero_division = 0))
df_row.append(recall_score(ytest, prediction, average = 'weighted'))
df_row.append(accuracy_score(ytest, prediction))
df_row.append(f1_score(ytest, prediction, average = 'weighted'))


# Add to scores DataFrame.
scores.loc[len(scores)] = df_row
```

In [144]:
```
# Support Vector Machine, Word2Vec

# Fit the model.
svc.fit(xtrain_w2v, ytrain)

# Peform prediction on ytest.
prediction = svc.predict(xtest_w2v)

# Calculate Statistics.
df_row = ['Support Vector Machine', 'Word2Vec', svc_w2v_xtrain_kfold.mean()]
df_row.append(precision_score(ytest, prediction, average = 'weighted',
                              zero_division = 0))
df_row.append(recall_score(ytest, prediction, average = 'weighted'))
df_row.append(accuracy_score(ytest, prediction))
df_row.append(f1_score(ytest, prediction, average = 'weighted'))


# Add to scores DataFrame.
scores.loc[len(scores)] = df_row
```

In [145]:
```
scores
```

Out[145]:

	Model	Vectorization	K-Fold - F1 Weighted	Precision	Recall	Accuracy	F1 Weighted Average
0	Multinomial Naive Bayes	TF-IDF	0.397	0.359	0.494	0.494	0.394
1	Multinomial Naive Bayes	Bag-Of-Words	0.579	0.571	0.576	0.576	0.560
2	Multinomial Naive Bayes	Word2Vec	0.452	0.461	0.446	0.446	0.438
3	Logistic Regression	TF-IDF	0.443	0.430	0.505	0.505	0.447
4	Logistic Regression	Bag-Of-Words	0.619	0.628	0.639	0.639	0.625
5	Logistic Regression	Word2Vec	0.594	0.585	0.595	0.595	0.562
6	Support Vector Machine	TF-IDF	0.420	0.396	0.502	0.502	0.426
7	Support Vector Machine	Bag-Of-Words	0.611	0.613	0.625	0.625	0.616
8	Support Vector Machine	Word2Vec	0.582	0.588	0.603	0.603	0.552

The support Vector Machine models showed a dip in score for each vectorization technique, though not a big dip. I will have to see if hyperparameter tuning will make either of these models clearly better than the other. For now, Logistic Regression Bag-Of-Words is still the best model, but there are still two model techniques left to try.

7.4 Random Forest

Random Forest: Random forests creates multiple decision trees that work independently of each other, then the model sees what most of the decision trees decide and takes that result. It's not unlike a bunch of human voting on something.

In [146]:
```
# Setting up the model.
rf = RandomForestClassifier(n_estimators=400, random_state=11)
```

In [147]:
```
# Calculating Cross Val Score, TF-IDF
rf_tfidf_xtrain_kfold = cross_val_score(rf, xtrain_tfidf, ytrain,
                                        scoring = 'f1_weighted')
rf_tfidf_xtrain_kfold.mean()
```

Out[147]:
```
0.4453602093876981
```

In [148]:
```
# Calculating Cross Val Score, Bag of Words
rf_bow_xtrain_kfold = cross_val_score(rf, xtrain_bow, ytrain,
                                       scoring = 'f1_weighted')
rf_bow_xtrain_kfold.mean()
```

Out[148]:
```
0.5952550322661374
```

In [149]:
```
# Calculating Cross Val Score, Word2Vec
rf_w2v_xtrain_kfold = cross_val_score(rf, xtrain_w2v, ytrain,
                                       scoring = 'f1_weighted')
rf_w2v_xtrain_kfold.mean()
```

Out[149]:
```
0.5419064652676904
```

In [150]:
```
# Random Forest, TF-IDF

# Fit the model.
rf.fit(xtrain_tfidf, ytrain)

# Peform prediction on ytest.
prediction = rf.predict(xtest_tfidf)

# Calculate Statistics.
df_row = ['Random Forest', 'TF-IDF', rf_tfidf_xtrain_kfold.mean()]
df_row.append(precision_score(ytest, prediction, average = 'weighted',
                              zero_division = 0))
df_row.append(recall_score(ytest, prediction, average = 'weighted'))
df_row.append(accuracy_score(ytest, prediction))
df_row.append(f1_score(ytest, prediction, average = 'weighted'))


# Add to scores DataFrame.
scores.loc[len(scores)] = df_row
```

In [151]:
```
# Random Forest, Bag of Words

# Fit the model.
rf.fit(xtrain_bow, ytrain)

# Peform prediction on ytest.
prediction = rf.predict(xtest_bow)

# Calculate Statistics.
df_row = ['Random Forest', 'Bag-Of-Words', rf_bow_xtrain_kfold.mean()]
df_row.append(precision_score(ytest, prediction, average = 'weighted',
                              zero_division = 0))
df_row.append(recall_score(ytest, prediction, average = 'weighted'))
df_row.append(accuracy_score(ytest, prediction))
df_row.append(f1_score(ytest, prediction, average = 'weighted'))


# Add to scores DataFrame.
scores.loc[len(scores)] = df_row
```

In [152]:
```
# Random Forest, Word2Vec

# Fit the model.
rf.fit(xtrain_w2v, ytrain)

# Peform prediction on ytest.
prediction = rf.predict(xtest_w2v)

# Calculate Statistics.
df_row = ['Random Forest', 'Word2Vec', rf_w2v_xtrain_kfold.mean()]
df_row.append(precision_score(ytest, prediction, average = 'weighted',
                              zero_division = 0))
df_row.append(recall_score(ytest, prediction, average = 'weighted'))
df_row.append(accuracy_score(ytest, prediction))
df_row.append(f1_score(ytest, prediction, average = 'weighted'))


# Add to scores DataFrame.
scores.loc[11] = df_row
```

In [153]:
```
scores
```

Out[153]:

	Model	Vectorization	K-Fold - F1 Weighted	Precision	Recall	Accuracy	F1 Weighted Average
0	Multinomial Naive Bayes	TF-IDF	0.397	0.359	0.494	0.494	0.394
1	Multinomial Naive Bayes	Bag-Of-Words	0.579	0.571	0.576	0.576	0.560
2	Multinomial Naive Bayes	Word2Vec	0.452	0.461	0.446	0.446	0.438
3	Logistic Regression	TF-IDF	0.443	0.430	0.505	0.505	0.447
4	Logistic Regression	Bag-Of-Words	0.619	0.628	0.639	0.639	0.625
5	Logistic Regression	Word2Vec	0.594	0.585	0.595	0.595	0.562
6	Support Vector Machine	TF-IDF	0.420	0.396	0.502	0.502	0.426
7	Support Vector Machine	Bag-Of-Words	0.611	0.613	0.625	0.625	0.616
8	Support Vector Machine	Word2Vec	0.582	0.588	0.603	0.603	0.552
9	Random Forest	TF-IDF	0.445	0.439	0.497	0.497	0.435
10	Random Forest	Bag-Of-Words	0.595	0.597	0.604	0.604	0.572
11	Random Forest	Word2Vec	0.542	0.553	0.551	0.551	0.509

The Random Forest modeling technique showed a significant decline in both Bag-Of-Words and Word2Vec vectorization techniques. Oddly, the model did show a slight improvement with TF-IDF vectorization and Support Vector Machine modeling, but the improvement is still not good enough to knock Logistic Regression Bag-Of-Words off it's pedestal as the best model. One modeling technique left.

7.5 XGBoost

XGBoost: XGBoost is a popular Python library provides gradient boosting. It often outperforms other models. It is a composit model of multiple models, or weak learners, are used to perform the same task.

In [154]:
```
# Setting up the model.
xgb = XGBClassifier(max_depth=6, n_estimators=1000, num_class = 12,
                    use_label_encoder=False, eval_metric = 'mlogloss')
```

In [155]:
```
# Calculating Cross Val Score, TF-IDF
xgb_tfidf_xtrain_kfold = cross_val_score(xgb, xtrain_tfidf, ytrain,
                                       scoring = 'f1_weighted')
xgb_tfidf_xtrain_kfold.mean()
```

Out[155]:
```
0.44767715432793265
```

In [156]:
```
# Calculating Cross Val Score, Bag of Words
xgb_bow_xtrain_kfold = cross_val_score(xgb, xtrain_bow, ytrain,
                                       scoring = 'f1_weighted')
xgb_bow_xtrain_kfold.mean()
```

Out[156]:
```
0.5971242839054627
```

In [157]:
```
# Calculating Cross Val Score, Word2Vec
xgb_w2v_xtrain_kfold = cross_val_score(xgb, xtrain_w2v, ytrain,
                                       scoring = 'f1_weighted')
xgb_w2v_xtrain_kfold.mean()
```

Out[157]:
```
0.5779513237660625
```

In [158]:
```
# XG Boost, TF-IDF

# Fit the model.
xgb.fit(xtrain_tfidf, ytrain)

# Peform prediction on ytest.
prediction = xgb.predict(xtest_tfidf)

# Calculate Statistics.
df_row = ['XGBoost', 'TF-IDF', xgb_tfidf_xtrain_kfold.mean()]
df_row.append(precision_score(ytest, prediction, average = 'weighted',
                              zero_division = 0))
df_row.append(recall_score(ytest, prediction, average = 'weighted'))
df_row.append(accuracy_score(ytest, prediction))
df_row.append(f1_score(ytest, prediction, average = 'weighted'))


# Add to scores DataFrame.
scores.loc[len(scores)] = df_row
```

In [159]:
```
# XG Boost, Bag of Words

# Fit the model.
xgb.fit(xtrain_bow, ytrain)

# Peform prediction on ytest.
prediction = xgb.predict(xtest_bow)

# Calculate Statistics.
df_row = ['XGBoost', 'Bag-Of-Words', xgb_bow_xtrain_kfold.mean()]
df_row.append(precision_score(ytest, prediction, average = 'weighted',
                              zero_division = 0))
df_row.append(recall_score(ytest, prediction, average = 'weighted'))
df_row.append(accuracy_score(ytest, prediction))
df_row.append(f1_score(ytest, prediction, average = 'weighted'))


# Add to scores DataFrame.
scores.loc[len(scores)] = df_row
```

In [160]:
```
# XG Boost, Word2Vec

# Fit the model.
xgb.fit(xtrain_w2v, ytrain)

# Peform prediction on ytest.
prediction = xgb.predict(xtest_w2v)

# Calculate Statistics.
df_row = ['XGBoost', 'Word2Vec', xgb_w2v_xtrain_kfold.mean()]
df_row.append(precision_score(ytest, prediction, average = 'weighted',
                              zero_division = 0))
df_row.append(recall_score(ytest, prediction, average = 'weighted'))
df_row.append(accuracy_score(ytest, prediction))
df_row.append(f1_score(ytest, prediction, average = 'weighted'))


# Add to scores DataFrame.
scores.loc[14] = df_row
```

In [161]:
```
scores
```

Out[161]:

	Model	Vectorization	K-Fold - F1 Weighted	Precision	Recall	Accuracy	F1 Weighted Average
0	Multinomial Naive Bayes	TF-IDF	0.397	0.359	0.494	0.494	0.394
1	Multinomial Naive Bayes	Bag-Of-Words	0.579	0.571	0.576	0.576	0.560
2	Multinomial Naive Bayes	Word2Vec	0.452	0.461	0.446	0.446	0.438
3	Logistic Regression	TF-IDF	0.443	0.430	0.505	0.505	0.447
4	Logistic Regression	Bag-Of-Words	0.619	0.628	0.639	0.639	0.625
5	Logistic Regression	Word2Vec	0.594	0.585	0.595	0.595	0.562
6	Support Vector Machine	TF-IDF	0.420	0.396	0.502	0.502	0.426
7	Support Vector Machine	Bag-Of-Words	0.611	0.613	0.625	0.625	0.616
8	Support Vector Machine	Word2Vec	0.582	0.588	0.603	0.603	0.552
9	Random Forest	TF-IDF	0.445	0.439	0.497	0.497	0.435
10	Random Forest	Bag-Of-Words	0.595	0.597	0.604	0.604	0.572
11	Random Forest	Word2Vec	0.542	0.553	0.551	0.551	0.509
12	XGBoost	TF-IDF	0.448	0.421	0.461	0.461	0.424
13	XGBoost	Bag-Of-Words	0.597	0.599	0.611	0.611	0.603
14	XGBoost	Word2Vec	0.578	0.587	0.597	0.597	0.578

XGBoost has a reputation for delivering the best models, but it did not perform better than Logistic Regression, but it did show a slight improvement for Word2Vec techniques.

Now, I'll look at the top five models' F1-Scores.

In [162]:
```
# Printing a dataframe of the top five performing models, organized by
# F1-Score weighted average.
rankings = scores.sort_values('F1 Weighted Average', ascending = False).copy()
rankings.reset_index(drop=True, inplace=True)

print('Top Five Performing Models')
rankings[0:5]
```

```
Top Five Performing Models
```

Out[162]:

	Model	Vectorization	K-Fold - F1 Weighted	Precision	Recall	Accuracy	F1 Weighted Average
0	Logistic Regression	Bag-Of-Words	0.619	0.628	0.639	0.639	0.625
1	Support Vector Machine	Bag-Of-Words	0.611	0.613	0.625	0.625	0.616
2	XGBoost	Bag-Of-Words	0.597	0.599	0.611	0.611	0.603
3	XGBoost	Word2Vec	0.578	0.587	0.597	0.597	0.578
4	Random Forest	Bag-Of-Words	0.595	0.597	0.604	0.604	0.572

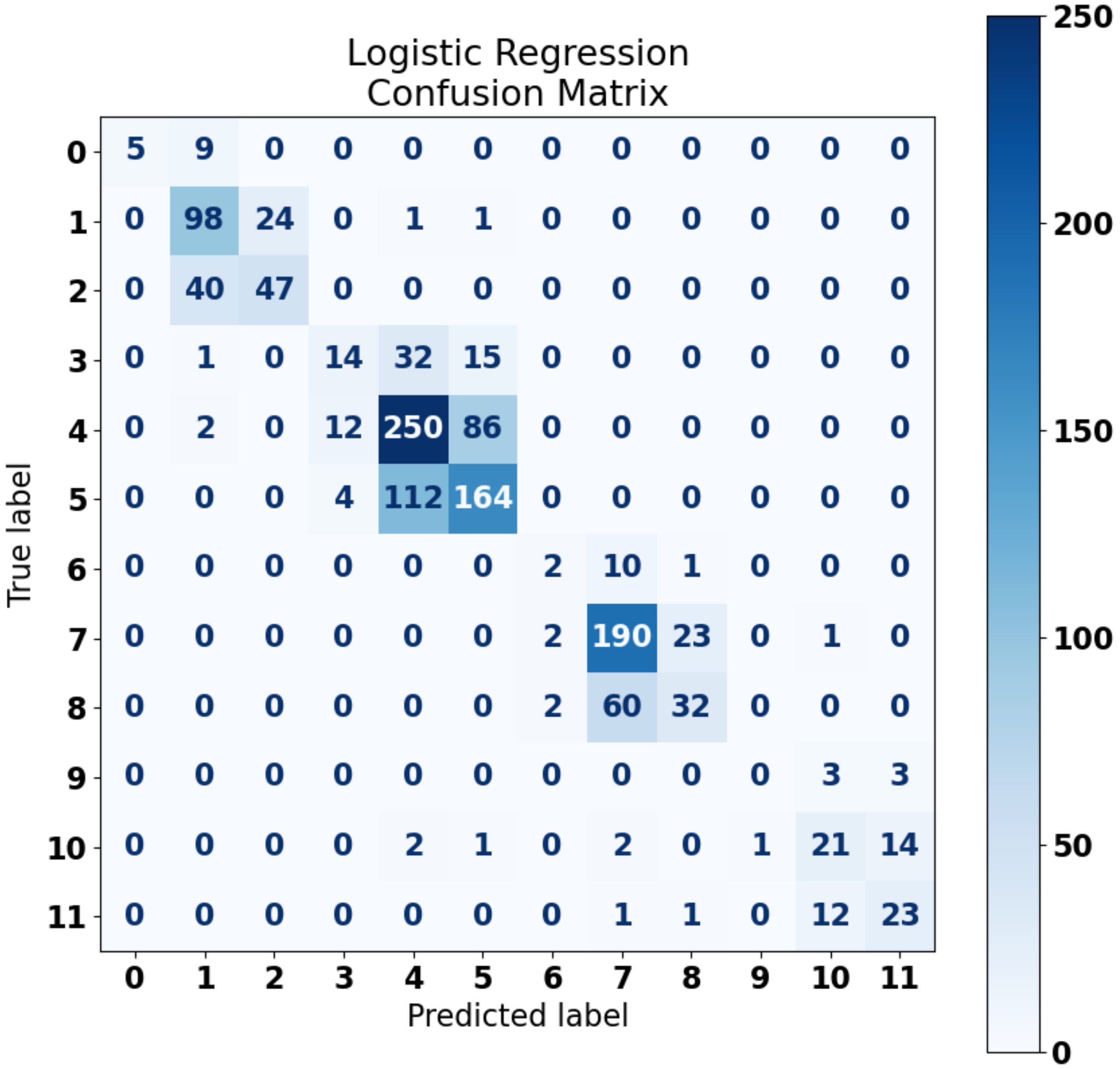
Clearly Bag-Of-Words is the highest performing vectorization technique. I'm going to plot the confusion matrix and classification reports for the top two models, since their scores are within .009 of each other.

7.6 Final Model Selection

In [163]:
```
# Function to plot a confusion matrix for the model.
def plot_matrix(model, xt, yt, the_title = '', color = plt.cm.Blues):

    # Fixing the font
    font = {'family' : 'DejaVu Sans',
            'weight' : 'bold',
            'size'   : 16}
    plt.rc('font', **font)

    # Printing the matrix.
    fig, ax = plt.subplots(figsize=(10,10))
    ConfusionMatrixDisplay.from_estimator(model, xt, yt,
                                          cmap=color, ax=ax)

    # The title is a parameter given by the user.
    plt.title(the_title)
    plt.show()
```

7.6.1 Logistic Regression Confusion Matrix

In [164]:
```
# The Confusion Matrix

lreg.fit(xtrain_bow, ytrain)
prediction = lreg.predict(xtest_bow)

# Plot Confusion Matrix
plot_matrix(lreg, xtest_bow, ytest,
            'Logistic Regression\nConfusion Matrix', plt.cm.Blues)

# Classification Report
print('Logisic Regression Classification Report')
print(classification_report(ytest, prediction, zero_division = 0, digits=4));
```



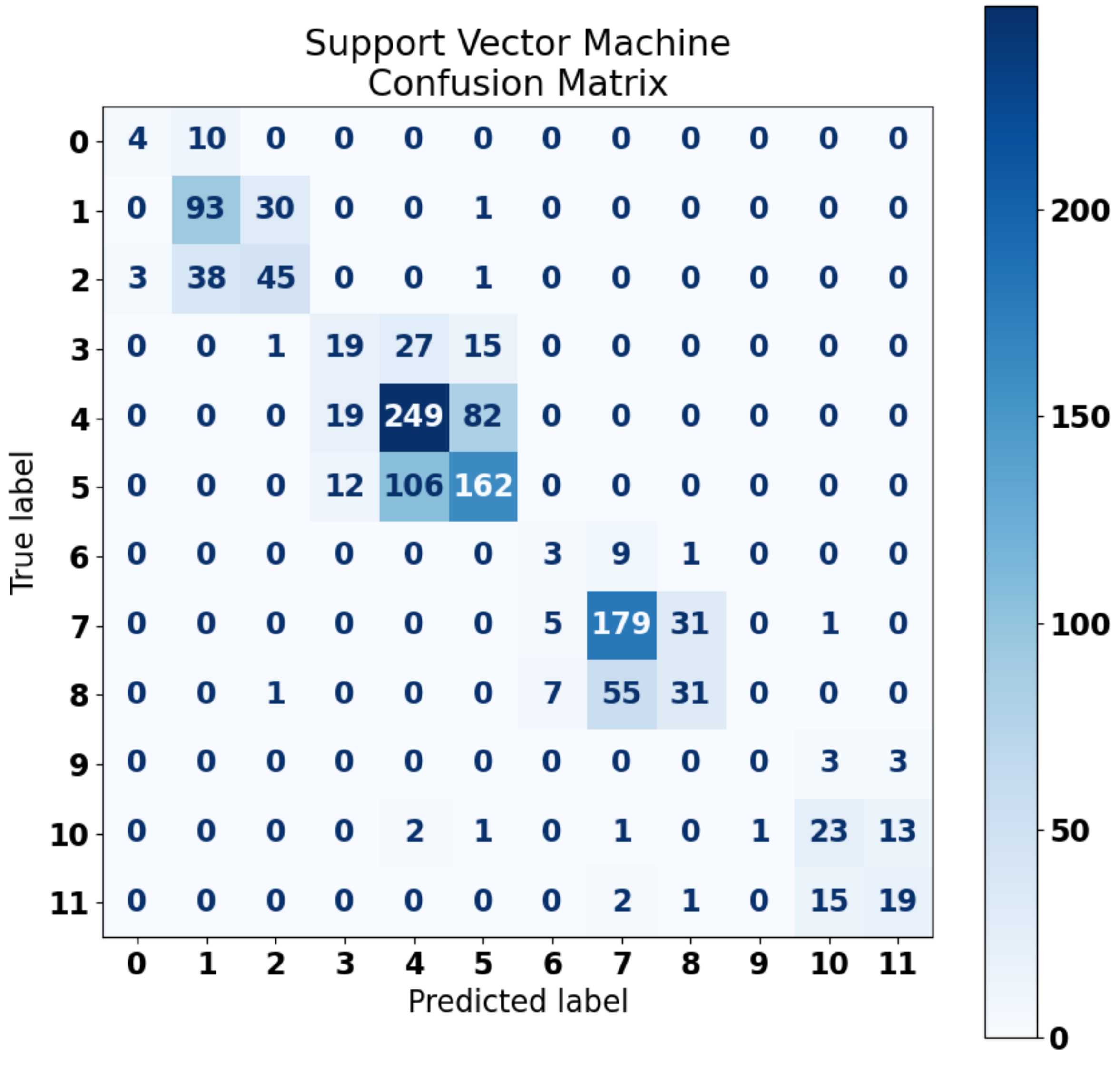
```
Logisic Regression Classification Report
              precision    recall  f1-score   support

           0     1.0000    0.3571    0.5263        14
           1     0.6533    0.7903    0.7153       124
           2     0.6620    0.5402    0.5949        87
           3     0.4667    0.2258    0.3043        62
           4     0.6297    0.7143    0.6693       350
           5     0.6142    0.5857    0.5996       280
           6     0.3333    0.1538    0.2105        13
           7     0.7224    0.8796    0.7933       216
           8     0.5614    0.3404    0.4238        94
           9     0.0000    0.0000    0.0000         6
          10     0.5676    0.5122    0.5385        41
          11     0.5750    0.6216    0.5974        37

    accuracy                         0.6390      1324
   macro avg     0.5655    0.4768    0.4978      1324
weighted avg     0.6281    0.6390    0.6246      1324
```

See analysis at the end of the section below.

7.6.2 Support Vector Machine Confusion Matrix

```
In [165]: # Plot Confusion Matrix
          svc.fit(xtrain_bow, ytrain)
          prediction = svc.predict(xtest_bow)

          # Plot Confusion Matrix
          plt.figure(figsize=(20, 20))
          plot_matrix(svc, xtest_bow, ytest,
                      'Support Vector Machine\nConfusion Matrix')

          # Classification Report
          print('Support Vector Machine Classification Report')
          print(classification_report(ytest, prediction, zero_division = 0, digits=4))
```

```
<Figure size 2000x2000 with 0 Axes>
```



	0	1	2	3	4	5	6	7	8	9	10	11
0	4	10	0	0	0	0	0	0	0	0	0	0
1	0	93	30	0	0	1	0	0	0	0	0	0
2	3	38	45	0	0	1	0	0	0	0	0	0
3	0	0	1	19	27	15	0	0	0	0	0	0
4	0	0	0	19	249	82	0	0	0	0	0	0
5	0	0	0	12	106	162	0	0	0	0	0	0
6	0	0	0	0	0	0	3	9	1	0	0	0
7	0	0	0	0	0	0	5	179	31	0	1	0
8	0	0	1	0	0	0	7	55	31	0	0	0
9	0	0	0	0	0	0	0	0	0	0	3	3
10	0	0	0	0	2	1	0	1	0	1	23	13
11	0	0	0	0	0	0	0	2	1	0	15	19

```
Support Vector Machine Classification Report
              precision    recall  f1-score   support

           0     0.5714    0.2857    0.3810        14
           1     0.6596    0.7500    0.7019       124
           2     0.5844    0.5172    0.5488        87
           3     0.3800    0.3065    0.3393        62
           4     0.6484    0.7114    0.6785       350
           5     0.6183    0.5786    0.5978       280
           6     0.2000    0.2308    0.2143        13
           7     0.7276    0.8287    0.7749       216
           8     0.4844    0.3298    0.3924        94
           9     0.0000    0.0000    0.0000         6
          10     0.5476    0.5610    0.5542        41
          11     0.5429    0.5135    0.5278        37

    accuracy                         0.6246      1324
   macro avg     0.4971    0.4678    0.4759      1324
weighted avg     0.6134    0.6246    0.6158      1324
```

Confusion Matrix Analysis:

The confusion matrix shows is an effective way to view the entire output of the model predictions vs. the true targets. The y-axis shows the true label, and the x-axis shows the predicted label. Each number printed in the graph represents the number of predictions made for that specific category that were actually the true category found in the y-axis. The 1 in point (8, 11) means the model predicted 8 when it was actually 11 only one time.

If the model predicted correctly, then the prediction and true values are the same, and thus, the number increased is along the diagonal from upper left to the bottom right. If the model were to predict everything perfectly, the only numbers above zero on the confusion matrix would be along this same diagonal.

Also, the targets are organized in groups of three representing the positive, negative, and neutral emotions of that combo of company and brand/product. It's easy to imagine—and I'll draw this in the presentation—four 3x3 boxes along the diagonal that represent all the values for that company and brand/product target. You'll then notice that nearly all of the predictions occur within these boxes, for both models. In fact, only 13 predictions fall outside of these boxes for the Logistic Regression model, and 12 predictions for the Support Vector Machine model.

This means that both models do an excellent job at identifying the correct company, and identifying whether the tweet is talking about that company's brand or it's product. Most of it's errors occur when the model tries to identify what emotion the tweet is projecting.

Classification Report Analysis:

The classification report is gives the individual scores for each classification as well as the number of true targets for that classification, listed under "support." Since I already demonstrated that the Train/Test split function cut a nigh even 20% of existing data off for the creation of the test set, it's unsurprising that the targets with few true values performed the lowest. As I pointed out earlier, negative tweets are underrepresented in this dataset, so the model is not as good at identifying these tweets.

The lack of the models ability to identify the negative tweets impacts its ability to identify neutral and positive tweets. Neither model predicted any values for target 9, "Negative - Google

8 Tuning Models

The f1 scores of both Logistic Regression and Support Vector Machine using Bag-Of-Words vectorization are the highest. I'll try to tune both of them. First, Logistic Regression.

I'll use two different methods: GridSearchCV and RandomizedSearchCV. GridSearchCV runs a model for each possible combination of hyperparameter variables you give it, and thus it takes a very long time. RandomizedSearchCV, however, runs a sample of the models available, and ceases to use variables that obviously lower the model's effectiveness. I can specify which score to use, so of course I'll prioritie the f1-score.

In [166]:
```
# The Hyperparameters I'll fine tune and their options.
param_grid_lreg = {
    'solver'   : ['liblinear', 'sag'],
    'penalty'  : ['l1', 'l2', ],
    'C'        : [1.0, 0.1],
    'max_iter' : [1000, 2000]
}

# I previously ran this cell with the following paramenters.
# I deleted all but the best parameters and one other option to save time
# when I run the whole notebook.

#    'solver'   : ['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga'],
#    'penalty'  : ['l1', 'l2', 'elasticnet'],
#    'C'        : [100, 10, 1.0, 0.1, 0.01],
#    'max_iter' : [1000, 5000, 6000, 7000, 8000, 9000, 10000]

lreg_cv = LogisticRegression()
grid_log = GridSearchCV(lreg_cv, param_grid_lreg,
                        scoring='f1_weighted', cv=None)
grid_log.fit(xtrain_bow, ytrain)

print("Tuned Logistic Regression Parameters: {}".format(grid_log.best_params_)
print("Best score is {}".format(grid_log.best_score_))
print("The scorer used is {}".format(grid_log.scorer_))
```

```
C:\Users\g_osb\anaconda3\envs\learn-env\lib\site-packages\sklearn\model_se
lection\_validation.py:378: FitFailedWarning:


20 fits failed out of a total of 80.
The score on these train-test partitions for these parameters will be set
to nan.
If these failures are not expected, you can try to debug them by setting e
rror_score='raise'.

Below are more details about the failures:
--------------------------------------------------------------------------
------
20 fits failed with the following error:
Traceback (most recent call last):
  File "C:\Users\g_osb\anaconda3\envs\learn-env\lib\site-packages\sklearn
\model_selection\_validation.py", line 686, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "C:\Users\g_osb\anaconda3\envs\learn-env\lib\site-packages\sklearn
```

In [167]:
```
from sklearn.model_selection import RandomizedSearchCV

# The Hyperparameters I'll fine tune and their options.
param_grid_lreg = {
    'solver'   : ['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga'],
    'penalty'  : [None, 'l1', 'l2', 'elasticnet'],
    'C'        : [100, 10, 1.0, 0.1, 0.01],
    'max_iter' : [1000, 5000, 6000, 7000, 8000, 9000, 10000]
}

lreg_cv = LogisticRegression()
rand_log = RandomizedSearchCV(lreg_cv, param_grid_lreg, n_iter=500,
                              scoring='f1_weighted', n_jobs=-1, cv=None,
                              random_state=1)
rand_log.fit(xtrain_bow, ytrain)

print("Tuned Logistic Regression Parameters: {}".format(rand_log.best_params_)
print("Best score is {}".format(rand_log.best_score_))
print("The scorer used is {}".format(rand_log.scorer_))
```

```
C:\Users\g_osb\anaconda3\envs\learn-env\lib\site-packages\sklearn\model_se
lection\_validation.py:378: FitFailedWarning:


1140 fits failed out of a total of 2500.
The score on these train-test partitions for these parameters will be set
to nan.
If these failures are not expected, you can try to debug them by setting e
rror_score='raise'.

Below are more details about the failures:
--------------------------------------------------------------------------
------
125 fits failed with the following error:
Traceback (most recent call last):
  File "C:\Users\g_osb\anaconda3\envs\learn-env\lib\site-packages\sklearn
\model_selection\_validation.py", line 686, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "C:\Users\g_osb\anaconda3\envs\learn-env\lib\site-packages\sklearn
```

Now for Support Vector Machine:

In [168]:
```
# The Hyperparameters I'll fine tune and their options.
param_grid_svm = {
    'C': [1, 10],
    'gamma': [1, 0.1],
    'kernel': ['rbf', 'poly'],
    'probability' : [False, True]
}

# I previously ran this cell with the following paramenters.
# I deleted all but the best parameters and one other option to save time
# when I run the whole notebook.
#    'C': [0.1, 1, 10, 100],
#    'gamma': [1, 0.1, 0.01, 0.001, 'scale', 'auto'],
#    'kernel': ['rbf', 'poly', 'sigmoid', 'linear'],

svm_cv = svm.SVC()
grid_svm = GridSearchCV(svm_cv, param_grid_svm,
                        scoring='f1_weighted', cv=None)
grid_svm.fit(xtrain_bow, ytrain)

print("Tuned Support Vector Machine Parameters: {}".format(
    grid_svm.best_params_))
print("Best score is {}".format(grid_svm.best_score_))
print("The scorer used is {}".format(grid_svm.scorer_))
```

```
Tuned Support Vector Machine Parameters: {'C': 10, 'gamma': 0.1, 'kernel': 'r
bf', 'probability': False}
Best score is 0.6035840364138886
The scorer used is make_scorer(f1_score, pos_label=None, average=weighted)
```

In [169]:
```
# The Hyperparameters I'll fine tune and their options.
param_grid_svm = {
    'C': [0.1, 1, 10, 100],
    'gamma': [1, 0.1, 0.01, 0.001, 'scale', 'auto'],
    'kernel': ['rbf', 'poly', 'sigmoid', 'linear'],
    'probability' : [False, True]
}

svm_cv = svm.SVC()
rand_svm = RandomizedSearchCV(svm_cv, param_grid_svm,n_iter=500,
                              scoring='f1_weighted', n_jobs=-1, cv=None,
                              random_state=1)
rand_svm.fit(xtrain_bow, ytrain)

print("Tuned Support Vector Machine Parameters: {}".format(
    rand_svm.best_params_))
print("Best score is {}".format(rand_svm.best_score_))
print("The scorer used is {}".format(rand_svm.scorer_))
```

```
C:\Users\g_osb\anaconda3\envs\learn-env\lib\site-packages\sklearn\model_selec
tion\_search.py:305: UserWarning:

The total space of parameters 192 is smaller than n_iter=500. Running 192 ite
rations. For exhaustive searches, use GridSearchCV.


Tuned Support Vector Machine Parameters: {'probability': False, 'kernel': 'rb
f', 'gamma': 0.01, 'C': 10}
Best score is 0.6127867744213832
The scorer used is make_scorer(f1_score, pos_label=None, average=weighted)
```

These scores actually do worse than the defaults, which is odd. This is because both functions split the available data given to them into its own train and test split to evaluate its results, so the results may be slightly different than if I run these same parameters on the entire test data, which I'll do for the final model.

Logistic Regression Bag-Of-Words remains the best model. I'll adopt the best parameters for Logistic Regression listed by the runing functions above since that will yield the highest f1-score.

9 Final Model And Analysis

Now that I have selected the model, vectorizer, and best parameters, I need to examine how its predictions on the test set compare to the true targets. These next few cells set up the numbers needed for the graphs below.

In [170]:
```
# For the the final model visualizations below, I need a DataFrame, dictionary
# and string sorted 0-11 with the target text for each corresponding number.
df_target = pd.DataFrame()
for index, category in zip(df['Target'].unique(),
                            df['Target Text'].unique()):
    df_target.loc[index, 'Target Text'] = category
df_target = df_target.sort_index()

df_target_dict = {}
target_strings = []
for i in df_target.index:
    string = df_target['Target Text'][i]
    df_target_dict[i] = string
    target_strings.append(string)
```

In [171]:
```
# Final Model - Logistic Regression Utilizing Bag of Words Vectorization.

# Set model final model parameters.
lreg_final = LogisticRegression(C = 1, max_iter = 1000, penalty = 'l2',
                                solver = 'sag')

# Calculating Final Model Cross Val Score.
lreg_final_kfold = cross_val_score(lreg_final, xtrain_bow, ytrain,
                                   scoring = 'f1_weighted')

# Fit the final model.
lreg_final.fit(xtrain_bow, ytrain)

# Perform prediction on ytest.
prediction = lreg_final.predict(xtest_bow)

# Calculate Statistics.
df_row = ['Final Model - Logistic Regression', 'Bag-Of-Words',
          lreg_final_kfold.mean()]
df_row.append(precision_score(ytest, prediction, average = 'weighted',
                              zero_division = 0))
df_row.append(recall_score(ytest, prediction, average = 'weighted'))
df_row.append(accuracy_score(ytest, prediction))
df_row.append(f1_score(ytest, prediction, average = 'weighted'))

# Add to scores DataFrame.
scores.loc[len(scores)] = df_row
```

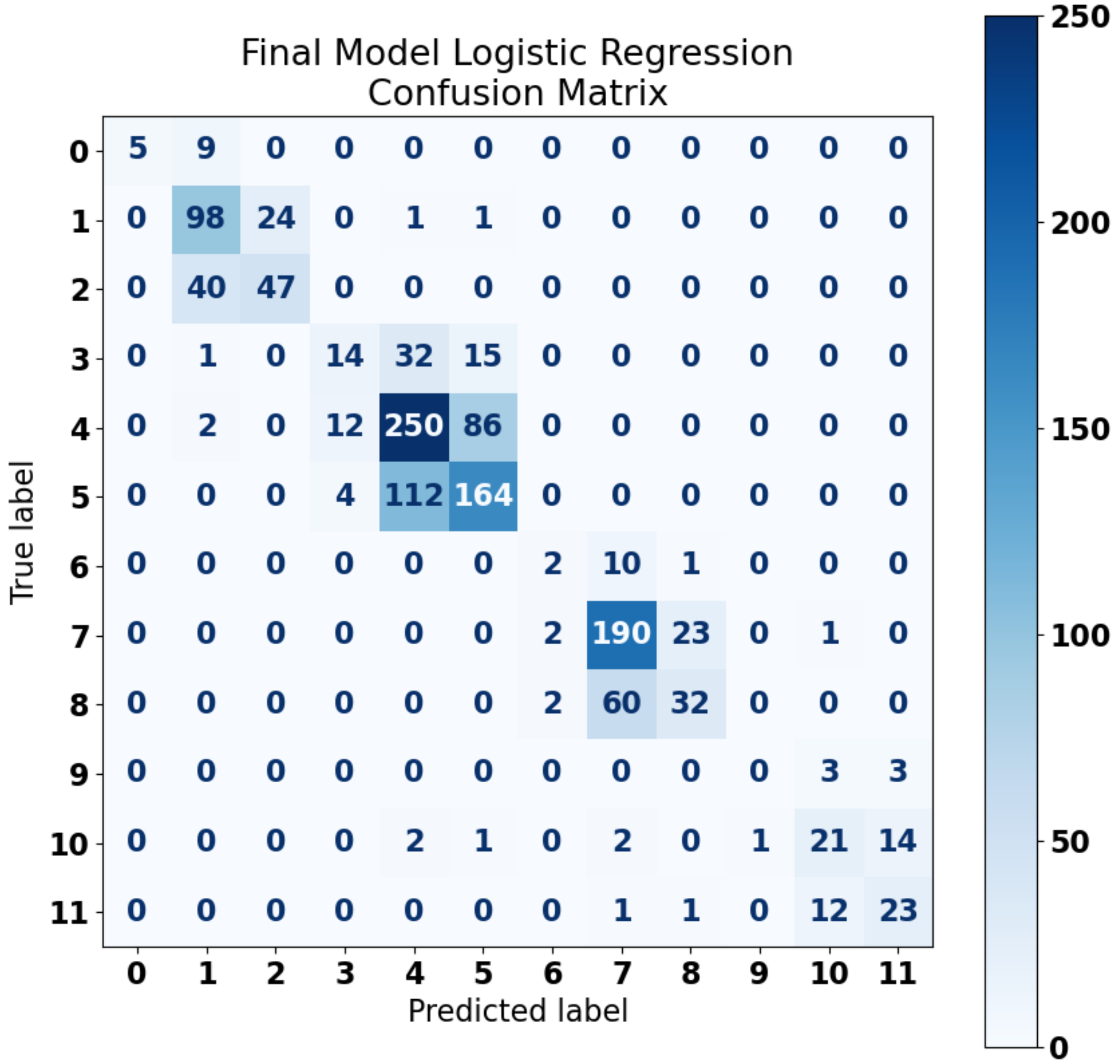
In [172]:
```python
# This cell allows me to create a DataFrame that I can use to calculate the
# total occurrences of TP, FP, TN, FN for each target.

# Creating the DataFrame
ytest_results_columns = ['Target', 'Target Text',
                         'Prediction', 'Prediction Text']
columns_tp = list(df_target['Target Text'] +' TP')
columns_fp = list(df_target['Target Text'] +' FP')
columns_tn = list(df_target['Target Text'] +' TN')
columns_fn = list(df_target['Target Text'] +' FN')
results_col = columns_tp + columns_fp + columns_tn + columns_fn
ytest_results_columns.extend(results_col)
ytest_results = pd.DataFrame(columns = ytest_results_columns)

# Setting the values for the DataFrame
ytest_results['Target'] = ytest.copy()
ytest_results[results_col] = False
ytest_results['Target Text'] = xtest['Target Text']
ytest_results['Prediction'] = prediction

# Labeling each row as TP, FP, TN, or FN.
for i,j in zip(ytest_results.index, range(len(prediction))):
    pt = df_target_dict[prediction[j]]
    ytest_results.loc[i, 'Prediction Text'] = pt
    t = ytest_results.loc[i,'Target']
    tt = ytest_results.loc[i,'Target Text']
    p = ytest_results.loc[i,'Prediction']
    for col in df_target_dict.values():
        if col == tt:
            if t == p:
                ytest_results.loc[i, tt + ' TP'] = True
            elif t != p:
                ytest_results.loc[i, tt + ' FN'] = True
        elif col == pt:
            ytest_results.loc[i, pt + ' FP'] = True
        elif col != pt:
            ytest_results.loc[i, col + ' TN'] = True
```

In [173]:
```
# Creating a DataFrame that counts each classification's TP, FN, and FP.
tp_df = pd.DataFrame(ytest_results[columns_tp].sum())
fn_df = pd.DataFrame(ytest_results[columns_fn].sum())
fp_df = pd.DataFrame(ytest_results[columns_fp].sum())

# Creating lists of the same data above for graphing.
tp_count = []
fn_count = []
fp_count = []
for i, j, k in zip(tp_df.index, fn_df.index, fp_df.index):
    tp_count.append(int(tp_df.loc[i,:]))
    fn_count.append(int(fn_df.loc[j,:]))
    fp_count.append(int(fp_df.loc[k,:]))

# Creating a count for each classification's target and prediction.
targ_count = []
pred_count = []
for i in range(len(tp_count)):
    targ_count.append(tp_count[i] + fn_count[i])
    pred_count.append(tp_count[i] + fp_count[i])
```

After all that code, I can now easily create the graphs I want. I will print the same confusion matrix that I used earlier and two graphs. First, the confusion matrix.

In [174]:
```
# The scores in the DataFrame format I used earlier.
display(scores.loc[len(scores)-1:len(scores)-1])

# Plot Confusion Matrix
plot_matrix(lreg, xtest_bow, ytest,
            'Final Model Logistic Regression\nConfusion Matrix', plt.cm.Blues)

# Classification Report
print('Final Model Logisic Regression Classification Report')
print(classification_report(ytest, prediction, zero_division = 0, digits=4));
```

	Model	Vectorization	K-Fold - F1 Weighted	Precision	Recall	Accuracy	F1 Weighted Average
15	Final Model - Logistic Regression	Bag-Of-Words	0.618	0.629	0.640	0.640	0.625



$$\begin{array}{c|cccccccccccc}
 & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 \\ \hline
0 & 5 & 9 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 98 & 24 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
2 & 0 & 40 & 47 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
3 & 0 & 1 & 0 & 14 & 32 & 15 & 0 & 0 & 0 & 0 & 0 & 0 \\
4 & 0 & 2 & 0 & 12 & 250 & 86 & 0 & 0 & 0 & 0 & 0 & 0 \\
5 & 0 & 0 & 0 & 4 & 112 & 164 & 0 & 0 & 0 & 0 & 0 & 0 \\
6 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 10 & 1 & 0 & 0 & 0 \\
7 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 190 & 23 & 0 & 1 & 0 \\
8 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 60 & 32 & 0 & 0 & 0 \\
9 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 3 & 3 \\
10 & 0 & 0 & 0 & 0 & 2 & 1 & 0 & 2 & 0 & 1 & 21 & 14 \\
11 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 12 & 23
\end{array}$$

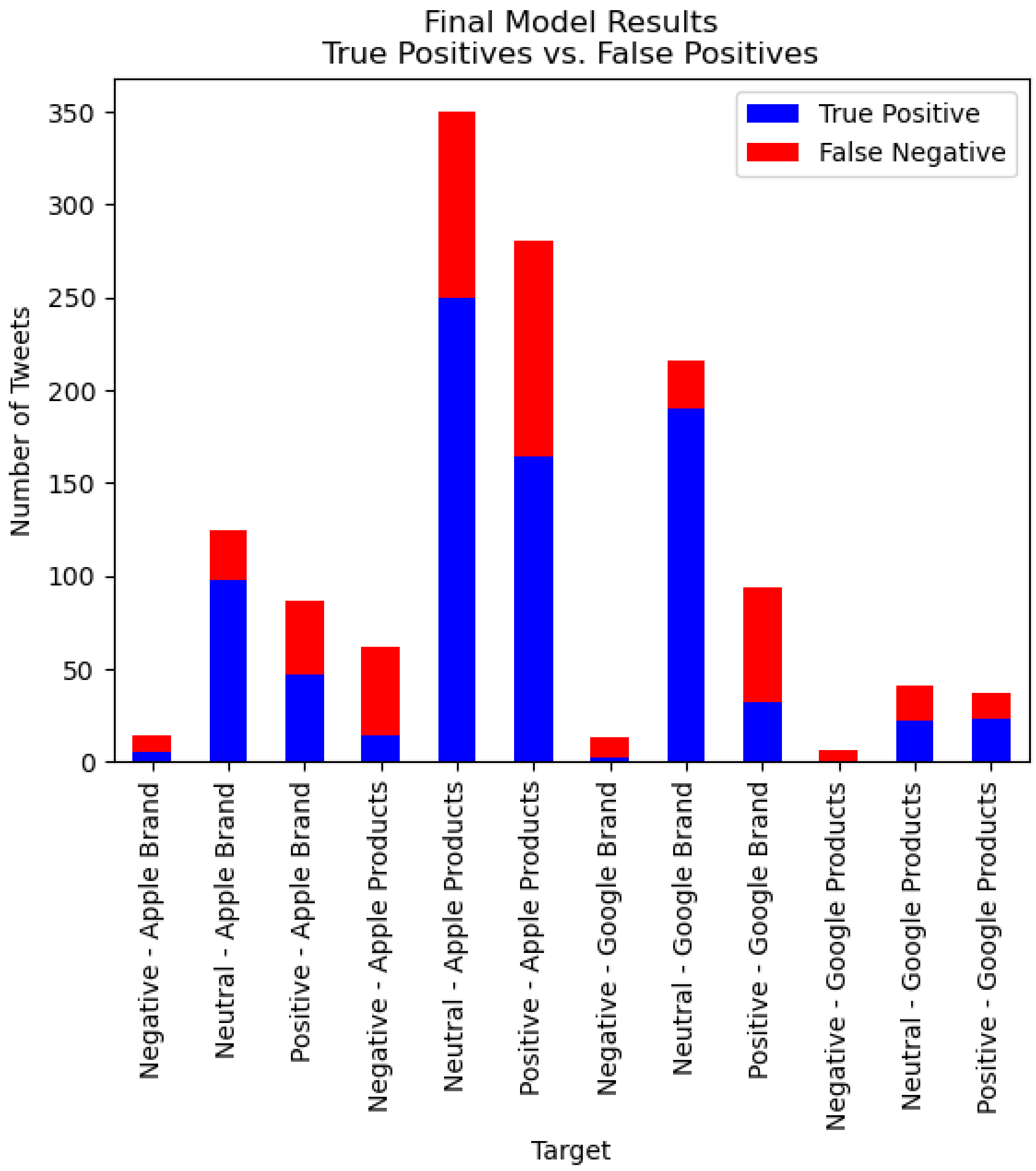
```
Final Model Logisic Regression Classification Report
              precision    recall  f1-score   support

           0     1.0000    0.3571    0.5263        14
           1     0.6533    0.7903    0.7153       124
           2     0.6620    0.5402    0.5949        87
           3     0.4667    0.2258    0.3043        62
           4     0.6297    0.7143    0.6693       350
           5     0.6165    0.5857    0.6007       280
           6     0.3333    0.1538    0.2105        13
           7     0.7224    0.8796    0.7933       216
           8     0.5614    0.3404    0.4238        94
           9     0.0000    0.0000    0.0000         6
          10     0.5789    0.5366    0.5570        41
          11     0.5750    0.6216    0.5974        37

    accuracy                         0.6397      1324
   macro avg     0.5666    0.4788    0.4994      1324
weighted avg     0.6290    0.6397    0.6254      1324
```

This model is identical to the earlier Logistic Regression / Bag-Of-Words model. Again, the boxes for each company - brand/products combination, you'll notice that nearly all of the predictions occur within these boxes, in fact, only 13 predictions fall outside of these boxes. Again, the model does poorest with the negative tweets. Again, it has a 64% accuracy overall and a weighted F1 score of 0.63

This connects to the business problem because it will allow Google to quantify, with a 64% accuracy, how many tweets in a set are positive about them, and likewise positive about their competitor Apple. This information alone can only tell Google how they are doing during this time period. It does not tell Google what they need to focus on improving and what people like about the competition. For that, Google will need to combine the classified tweets with the Word Association techniques above.

Next, a stacked bar graph:

In [175]:
```python
# Reseting text parameters
plt.rcParams.update(plt.rcParamsDefault)

# Bar Plot
bar_plot = pd.DataFrame({'True Positive' : tp_count,
                         'False Negative' : fn_count},
                         index = target_strings)
bar_plot.plot(kind = 'bar', stacked = True,
              color = ['Blue', 'Red'])
plt.xlabel('Target')
plt.ylabel('Number of Tweets')
plt.title('Final Model Results\nTrue Positives vs. False Positives')
plt.show()
```



Before I move forward, it is a good idea to look back. Here is the problem statement from the beginning of the document:

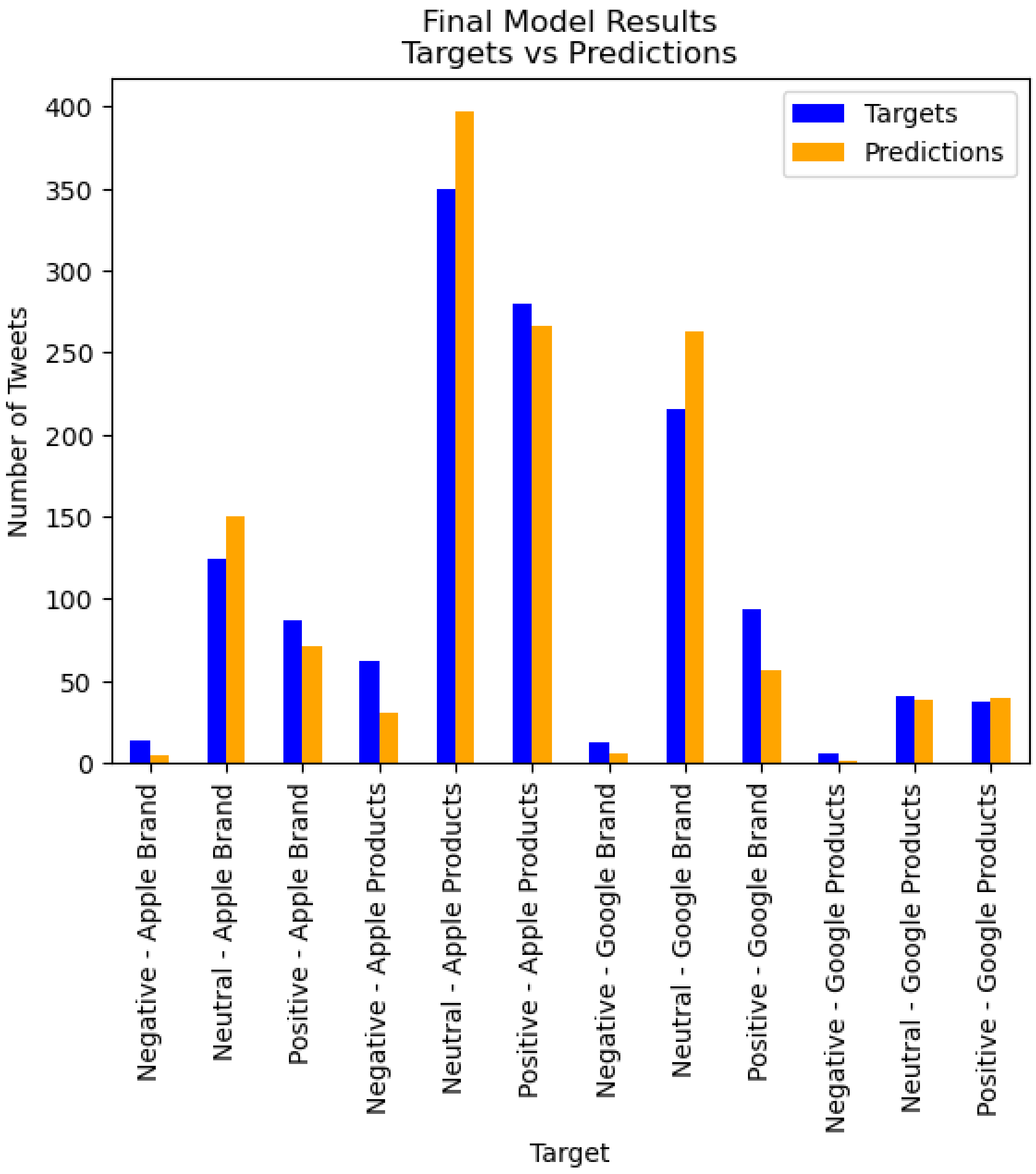
Problem: Some people view Google more negatively than their chief competitor, Apple. Google needs a tool to help measure the publics' sentiment toward both Apple and Google to strategize for a more positive public sentiment. This tool will offer Google insight to guide their business decisions.

In that sense, the model itself is the tool that solves the problem, not the analysis of this set of tweets. When Google uses this tool in the future, they will use it on a completely new set of tweets. They will not have a confusion matrix unless they take the time to self-label any of the tweets they classify with the model. Likewise the graph above will not be available for a new set of tweets.

Two pieces of information are on display in the graph above. First, the total count of all true targets, and second, the TP/FP divide. This graph displays the False Negatives for each category, and the proportion between TP/FP.

Most of what we glean from this graph is visualizing the accuracy, which is 64%. Now, I'll compare the true values with the predicted values.

In [177]:
```
# Second Bar Plot
bar_plot = pd.DataFrame({'Targets' : targ_count,
                         'Predictions' : pred_count},
  index = target_strings)
bar_plot.plot(kind = 'bar', stacked = False,
              color = ['Blue', 'Orange'])
plt.xlabel('Target')
plt.ylabel('Number of Tweets')
plt.title('Final Model Results\nTargets vs Predictions')
plt.show()
```



The graph above is more pertinent to the problem statement than either the confusion matrix or the TP/FP graph because it is similar to the classification data that Google will receive from the model when they use it on a future sets of tweets.

From the graph above, we can make the following observations.

During SXSW 2011:

1. People tweeted about Apple more often than Google.
2. People did not tweet many negative things about either Google or Apple.
3. People were more likely to tweet about Apple products rather than the Apple brand. But with Google, the opposite is true. People were more likely to tweet about the Google brand than Google products.
4. People were most likely to tweet with neutral sentiment when tweeting about either company.
5. People were most likely to talk positively about Apple products
6. People spoke positively about the Apple brand about as often as the Google brand, but the model showed a disparity between these two counts, with Apple receiving more positive brand tweet predictions. Improving this disparity can be a future research opportunity.

Finally, we can also see that none of the counts the model produced were too far off the mark. Most counts were close to their true counterparts. Probably the worst of models mistakes is the lack of properly predicted negative tweets for all categories. Each negative category drastically underperformed, but this is unsurprising as there was not a lot of negative data to train the model on. This can also be a future research opportunity.

10 Conclusion

After learning the perspectives of multiple customers at South By Southwest 2011, and utilizing the tools of NLP, here are my recommendations:

10.1 Limitations of Scope / Provided Data

Every Data Science project is limited by the acquired data and time. In this case, I analyzed 6620 tweets across one week of time 12 years ago. Both Apple and Google are vastly different companies today, and both have more than 6600 people tweeting about them. More tweets, and more recent tweets, to study would provide a much more accurate model, which will lead to a better snapshot tool.

A particular limit in this case was negative tweets. There were very few negative tweets found in the set compared to neutral and positive emotion tweets. The consequences of this limitation were particularly obvious when looking at the model results. The model did not predict negative emotions well in any of the categories.

Another limitation is that this data only included a single SXSW for a single year. We don't have the data to see if Google improved its image over time, or if Apple's image deteriorated. But this leads me to future research opportunities.

10.2 Future Research Opportunities

10.2.1 More Negative Tweet Data

As mentioned above, the primary thing this model needs is more negative tweet examples. SXSW primarily attracts fans of these companies, and fans who made the trip tend to view the companies they came to see through a optimistic lens, but they also are known to insult the companies they don't like, and sometimes they can turn bitter if they are disappointed by their favorite companies. We need to see more examples of these tweets, both scorned fans and naysayers who don't insult the companies they don't like. That is the only way we'll build a better model.

10.2.2 Scrape Annual SXSW tweets Every Time Google Presents

Another research opportunity is getting a sample of tweets for every year of SXSW that Google ever presented at. This can lead to changes over time and rely on a similar sample audience of twitter users at SXSW. It provides a useful bin that can show changes over time.

10.3 Recommendations

The analysis of the tweets available make it clear that Google is quite behind Apple in terms of both discussion and positive sentiment. To turn this around, I recommend:

10.3.1 Further Develop the Word Association Tool

No one person's, or one consulting firm's, assessment of the data found in 6000 tweets is complete. Google can give the word association tools to multiple departments. The maps team can look for word associations relevant to their application, and the search team can do the same for their purposes. This tool needs to be so accessible that non-technical people can use it, and perform searches to satisfy their own curiosity. This will lead to improvement areas and initiatives associated with their own products, and inspire creative innovations from Google's highly qualified employees.

10.3.2 Track Public Perception at SXSW With The Classification Model

The results are in. People tweeted about Apple more often, and more positively, than they did with Google. Google needs to improve their image. I'll address some specific ideas on how they can do that below. Regardless of what strategy Google chooses to improve its image, Google needs to keep track of how they are doing compared to its rival, Apple.

The simplest way to do this is to gather a sample of tweets every year at SXSW. This will provide a standardized audience and time of year that will provide meaningful data, allowing Google to track perception changes. Once Google have tweet sets from each of the past 12 SXSW festivals, Google will see how their image has changed over time, and they can then make better assessments regarding their future.

10.3.3 Create Google Branded Products

In 2011, Chromebooks were not available yet. Google brand phones did exist, but customers associated them with the companies that made them rather than Google, like HTC or Samsung. Google didn't sell anything physical to customers to get them excited, and this was the most glaring disparity between how people tweeted about Google and Apple.

People were excited to get to get their recently released iPad 2. People were excited to go inside the "Pop-Up" Apple Store to buy products. Google, by contrast, had a party that people could go to and meet Matt Damon. What Google was offering couldn't compete with the excitement of getting a new tech toy.

Google needs to create top of the line chromebooks and phones if it is going to compete with the excitement people have about Apple. It cannot rely on software alone.

10.3.4 Monetize Google Maps

Google Maps was what tweeters enjoyed most about Google at SXSW 2011. Google needs to ensure its ease of use and deep catalog of locations, including robust information about each one, is as popular in the future as it was at SXSW 2021. An upset in the digital maps business is certainly possible. Multiple other mapping software and apps exist.

I doubt people would continue to use Google Maps if they had to purchase the app, but you could sell advertising space to companies so they will appear more prominent when people search for their type of business. Google could also charge companies who want to pay for trips planned to their destinations. Whatever Google does though, they need to ensure people still enjoy using Maps.

10.3.5 Try Social Media Again

As a technology user, it was interesting to go back in time to 2011, when Google Plus was not a product in people's hands, but an idea in their heads. At that time, Facebook and Twitter had established their dominance in the Social Media space. The Arab Spring was in the midst of happening. People had a wonderfully positive view of the potential of Social Media. We had already seen some social media companies come and go. Myspace had already been closed for three years, and people seemed to think it was possible for another media giant to come in and rock the boat. Though Instagram was a few months old at the time, it's rise in popularity and the rise of TikTok are the only two players competing with Twitter and Facebook. Facebook purchased Instagram, of course, so TikTok is the only company to challenge their dominance in a decade.

It's possible the social media space is due for a major upset. All of these companies have dismal customer opinion. People have come to view them less as platforms to keep up with family and old friends, and more as hellscapes of trolling and the exploitation of people's anger for these company's profit.

Google has the financial resources to give social media another try, and specifically avoid the pitfalls of the other platforms. In 2023, the public is aware of the myriad of abuses of the current social media platforms. Google can use their negative sentiment to develop a product that avoids the same problems. It may be challenging, but it's possible.

10.3.6 Build on Google's Strengths Counter Apple's Strengths

Two of the positive words that came up during the word association analysis with Google were "Cool" and "Fast". This is what Google's customers associated with the brand in 2011. Contrary to that, people associated Apple with words like "Genius" and "Brilliant". They also associated the iPad with the "Future".

Make sure Google products continue to inspire people thinking they are "cool" and make sure they work seamlessly, very fast. Get with a marketing team to brainstorm ideas about how to combat the words that Apple is associated with.

10.3.7 Bring Apps to Android Faster

A repeated complaint for Android users was simply that apps that were already available on Apple were not available yet on Google. There are multiple reasons why developers might choose to launch on Apple first. Launch an investigation. Incentivize third-party developers to launch their apps on Android at least simultaneously with Apple, if not before the competition.

That's a wrap! Thanks for hiring Greg Osborne and I hope to work with you more in the future.