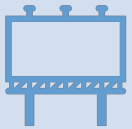
TWITTER MARKETING CLASSIFICATION & FORECASTING

ALEX GAMBOA, GERARD TIENG, VITO PEREZ

PROJECT BACKGROUND

We are working with a Los Angeles based marketing and networking expert with more than 10,000 followers on Twitter.



We were able to access two years of native Twitter analytics.

We are giving our client a better understanding of her tweets and social media influence.

GOAL OF TWITTER ANALYTICS

- This project is to create models that can benefit a social media influencer to create better branding and add value to future performance.
- There is a lot of data illustration on past data but we wanted to use machine learning to give brands awareness and guidance.
- Part 1: Tweet Classifications
- Part 2: Tweets Forecasting

UNDERSTANDING THE TWITTER ANALYTICS

Tweet permalink	Tweet text	Type	impressions	engagements	engagement rate	retweets	replies	likes	user profile clicks	url clicks
/status/11768922...	So excited that @Forbes shared my article on @...	3.0	608789.0	38345.0	0.062986	71.0	13.0	907.0	2077.0	753.0
/status/11839781...	My next few 💚 talks & livestreams:\n✨ #Vid...	2.0	283615.0	85.0	0.000300	11.0	1.0	22.0	10.0	0.0
/status/11527092...	On a magical journey with #Funko Mira of @dark...	4.0	221055.0	6372.0	0.028825	42.0	3.0	339.0	172.0	80.0
/status/11926178...	We're on the red carpet for @OriginalFunko #Ho...	1.0	218574.0	3854.0	0.017632	16.0	11.0	216.0	387.0	17.0
/status/11526773...	Always an absolute joy to collaborate with one...	2.0	209145.0	5912.0	0.028267	13.0	3.0	205.0	276.0	67.0

Brand Message Types:

After understanding the brand, we manually sorted each tweet in the set and labeled each as one of their 4 core message types.

1. Business & Promotions (brand message, weblinks, hashtags)
2. Spotlight & Shout Out (using @Tags to give support to other users)
3. Positivity & Wellness (directed towards the brands audience, hashtags)
4. Replies & Miscellaneous (replies and other tweets)

Machine Learning Multi-Classifier

In the absence of a person to view each tweet, we are looking to find if supervised machine learning plus **Natural Language Processing** libraries can correctly classify the text of any given tweet from their account into one of the four types of message category.

Data Cleansing

- We have to drop any null values before working with the set
- Create a new column for tweet types and classification
- There are a lot of tweets with non-words and symbols (like emojis, hashtags, and links). We created simple words to describe the characters

```
df= df.dropna()
df = df[["Tweet text", "Type"]]
df = df.rename(columns={"Tweet text":"text", "Type":"class"})
df = df.reset_index(drop=True)
df.head()
```

	text	class
0	HAPPY NEW YEAR 🎉\nMay all your dreams come tru...	2.0
1	“I've learned that you shouldn't go through li...	3.0
2	This 👇 https://t.co/4niBsvseUQ	1.0
3	Where should I go next? #travel https://t.co/D...	3.0
4	DONE with first day of #work back. Current fee...	2.0

```
#compiles list of non-emoji symbols
non_emoji = [letter for letter in "\\\'\"1234567890abcdefg
VWXYZ`~!@#$%^&*()-_=+[{]}|,<.>/?"]


import re
for i in range(df.shape[0]):

    dirty = df.iloc[i,0]
    dirty = re.sub("https\S+", "url", dirty)
    dirty = re.sub("@\S+", "friend", dirty)
    clean = re.sub("#\S+", "hashtag", dirty)

    sentence = clean.split()
    for words in sentence:
        for letters in words:
            if letters not in non_emoji:
                clean = clean.replace(letters, " emoji ")

    df.iloc[i,0] = clean

df
```

	text	class
0	HAPPY NEW YEAR emoji \nMay all your dreams co...	2.0
1	emoji I've learned that you shouldn't go thro...	3.0
2	This emoji url	1.0
3	Where should I go next? hashtag url	3.0
4	DONE with first day of hashtag back. Current f...	2.0
...	...	...
10178	friend friend friend Aww. I think I still don'...	4.0
10179	Excited to have my first day to sleep in on Sa...	3.0
10180	friend friend friend friend friend url	4.0
10181	Destiny is not a matter of chance emoji it is...	3.0
10182	friend old school foam rollers to sleep in for...	4.0

MACHINE LEARNING PRE PROCESS

The Porter Stemmer() function will be used to consolidate related words into root strings (e.g., happy, happiness = “happi”).

Stop words is a common filter for non-essential words (i.e. “the” and “a”)

The following code will split apart each tweet into individual words and stem each word before assembling the full tweet back together as a ”cleaned” string.

```
#Natural Language Toolkit for NLP processing
import nltk
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
import re

stemmer = PorterStemmer()
words = stopwords.words("english")

def machine_ready(tweet):
    tweet = tweet.split()
    ready = []
    for elements in tweet:
        if elements not in words:
            elements = re.sub("[^a-zA-Z]", "", elements) #remove punctuation
            elements = stemmer.stem(elements) #stem words
            elements = elements.lower() #lowercase words
            ready.append(elements)
    return " ".join(ready)

df["cleaned"] = df["text"].apply(machine_ready)
df.head()
```

```
#standard data libraries
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

#scikit-learn for Machine Learning algorithms

from sklearn.pipeline import Pipeline
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.feature_selection import SelectKBest, chi2
from sklearn.ensemble import RandomForestClassifier

#pickle for algorithm export
import pickle
```

ML - VECTORIZATION

We'll use sklearn's Vectorizer to convert the contents of each tweet as an array notating the appearance of each word for machine learning.

The dataset will need to be vectorized as an array in order for computers to process language.

Each unique word (+8,500) will be represented in its own position in the vector and a combination of the vector positions will form a tweet.

```
from sklearn.feature_extraction.text import TfidfVectorizer

vectorizer = TfidfVectorizer(stop_words="english")
features = vectorizer.fit_transform(df['cleaned']).toarray()
features.shape
```

```
(10183, 8596)
```

```
from sklearn.model_selection import train_test_split

X = features
y = df["class"]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25)
```

USING 4 MODEL ALGORITHMS

We are comparing the use of 4 different model algorithms to see which is best used to predict our clients 4 core message types.

Using **Pipeline** will help us keep track of the models used along with the **Kbest** priority feature selector

Random Forest Classifier appears to be the most accurate of all the models, although the classifier is only right about half the time in any scenario.

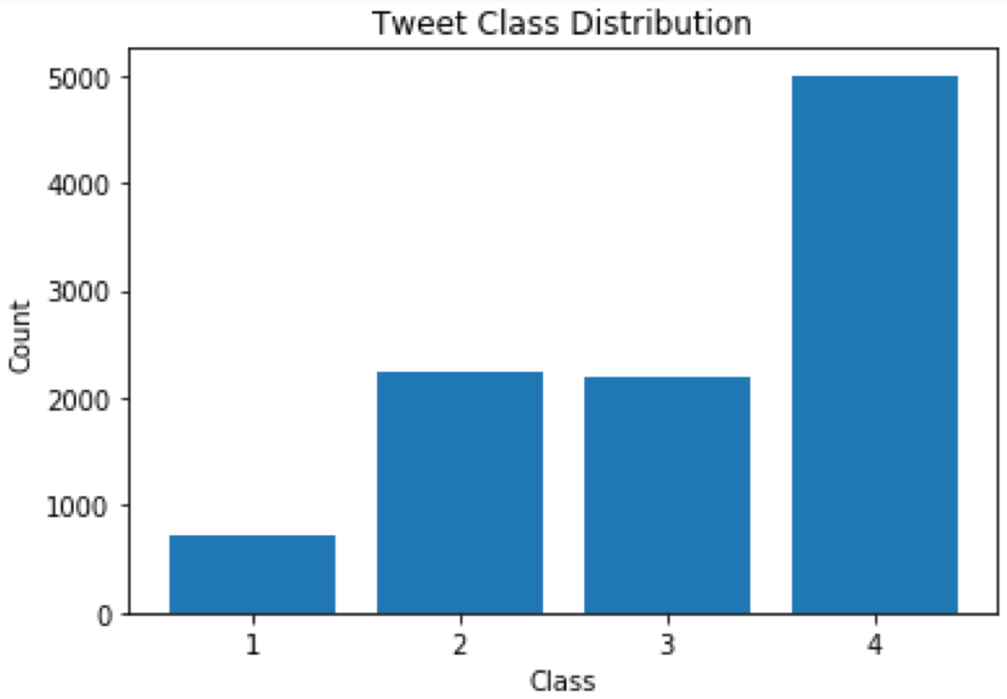
```
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import MultinomialNB
from sklearn.svm import LinearSVC


lr = Pipeline([('chi',  SelectKBest(chi2, k=20)),
                      ('clf', LogisticRegression())])
rfc = Pipeline([('chi',  SelectKBest(chi2, k=20)),
                      ('clf', RandomForestClassifier())])
mnb = Pipeline([('chi',  SelectKBest(chi2, k=20)),
                      ('clf', MultinomialNB())])
lsvc = Pipeline([('chi',  SelectKBest(chi2, k=20)),
                      ('clf', LinearSVC())])


models = [lr, rfc, mnb, lsvc]
model_name = ["lr", "rfc", "mnb", "lsvc"]

model_train_score= {}
model_test_score= {}


for i in range(len(models)):
    model_fit = models[i].fit(X_train, y_train)

    train_score = model_fit.score(X_train, y_train)
    test_score = model_fit.score(X_test, y_test)

    model_train_score[model_name[i]] = round(train_score, 2)
    model_test_score[model_name[i]] = round(test_score, 2)

print("train scores")
print(model_train_score)
print("test scores")
print(model_test_score)
```

```
train scores
{'lr': 0.51, 'rfc': 0.62, 'mnb': 0.49, 'lsvc': 0.52}
test scores
{'lr': 0.53, 'rfc': 0.52, 'mnb': 0.51, 'lsvc': 0.53}
```

TWEETS ANALYSIS

Interestingly, when looking at the distribution of the values, about half of the tweets in the dataset belong in the Class 4 category.

```
plt.bar([1,2,3,4], df["class"].value_counts().sort_index())
plt.title("Tweet Class Distribution")
plt.xlabel("Class")
plt.xticks([1,2,3,4])
plt.ylabel("Count")
```



Using the Pickle Library we have exported the RFC algorithm to a file.

We'll program a custom classify function to accept the tweet string, filter it through the classifier, and return the result.

```
pickle_load = open('rfc.pickle', 'rb')
clf = pickle.load(pickle_load)

def classify(string):
    non_emoji = [letter for letter in "\\\'\"1234567890abcdefghijklmnopqrstuvwxyz
RSTUVWXYZ`~!@#$%^&*()-_=+[{}]|,<.>/?"]
    for i in string:
        dirty = str(string)
        dirty = re.sub("https\S+", "", dirty) #links
        dirty = re.sub("@\S+", "", dirty) #@mentions
        dirty = re.sub("&amp", "", dirty) #ampersands
        dirty = re.sub("\n", "", dirty) #newline
        clean = re.sub("#", "", dirty) #hashtags

        sentence = clean.split()
        for words in sentence:
            for letters in words:
                if letters not in non_emoji:
                    clean = clean.replace(letters, "") #emojis

        tweet = clean.split()
        ready = []

        for elements in tweet:
            if elements not in words:
                elements = re.sub("[^a-zA-Z]", "", elements) #remove punctuation
                elements = stemmer.stem(elements) #stem words
                elements = elements.lower() #lowercase words
                ready.append(elements)
        tweet = " ".join(ready)

        pred = clf.predict(vectorizer.transform([tweet]))[0]
        return pred
```

TESTING OUR ALGORITHM

Exploring the Algorithm

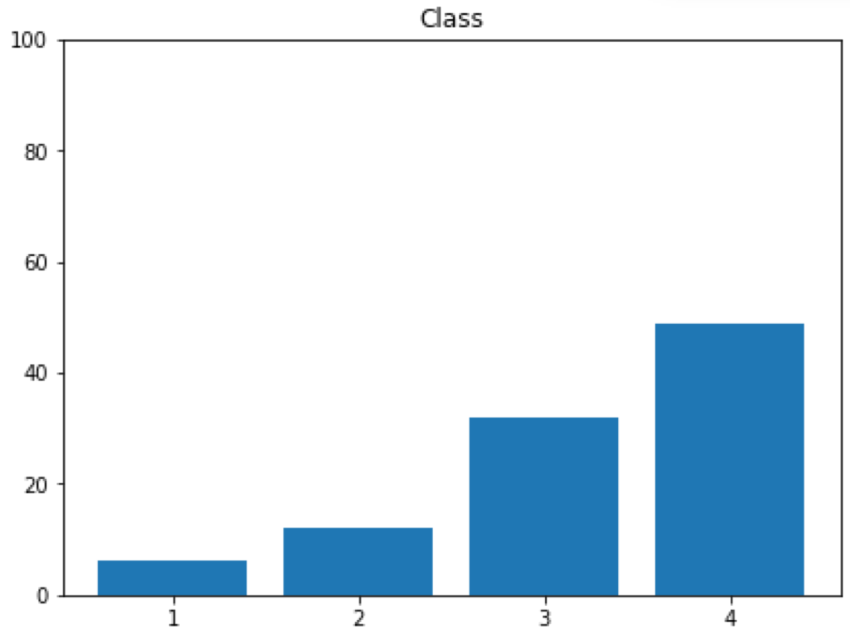
We used 100 tweets from March 2020 which has never been used in testing or training to validate the data.

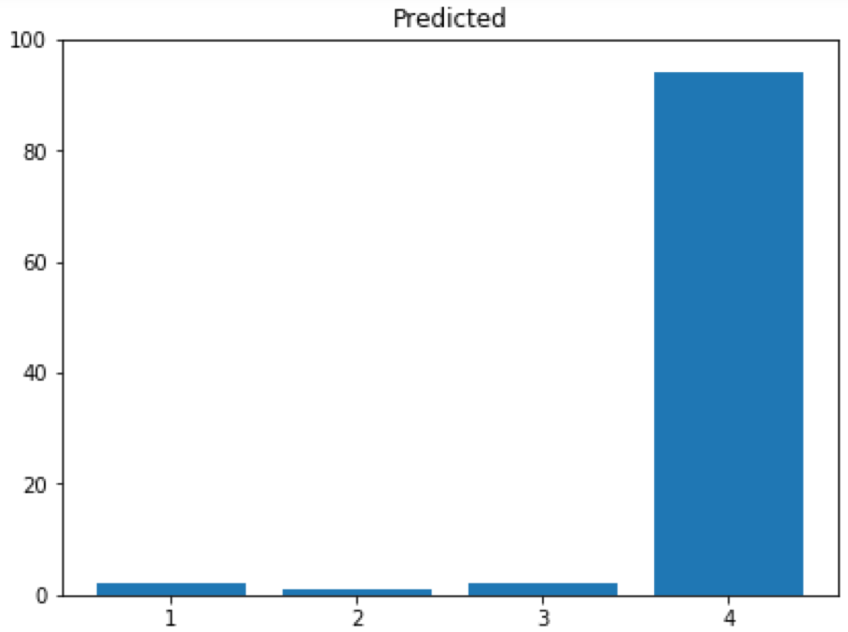
According to the bar plots below, the classifier isn't near the 50% projected accuracy of our algorithm for this set.

This can be because of a shift in tweets based upon COVID-19 changing normal behaviors and events.

```
#actual for March Data
plt.figure(figsize=(15,5))
plt.subplot(1,2,1)
plt.bar([1,2,3,4], march["class"].value_counts().sort_index())
plt.title("Class")
plt.xticks([1,2,3,4])
plt.ylim(0,100)

#predicted for March data
plt.subplot(1,2,2)
plt.bar([1,2,3,4], march["predictions"].value_counts().sort_index())
plt.title("Predicted")
plt.xticks([1,2,3,4])
plt.ylim(0,100)
```

```
(0, 100)
```

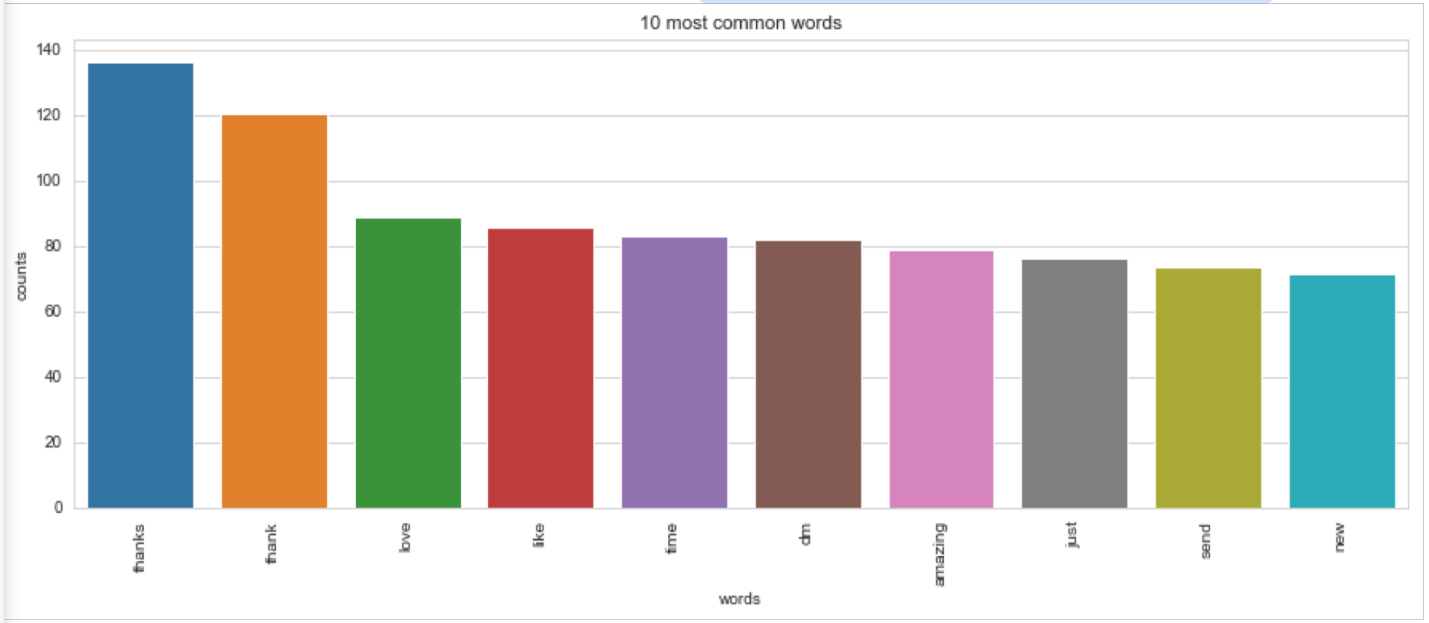




Considering Other Models

In further research and experimentation, **Latent Dirichlet Allocation (LDA)** had been identified as a potential solution for the classification our data via unsupervised topic modeling.
The model will display information on the most common terms and potential relationships of such terms to other form of patterns.

```
import seaborn as sns
sns.set_style('whitegrid')
# Helper function
def plot_10_most_common_words(count_data, count_vectorizer):
    import matplotlib.pyplot as plt
    words = count_vectorizer.get_feature_names()
    total_counts = np.zeros(len(words))
    for t in count_data:
        total_counts+=t.toarray()[0]

    count_dict = (zip(words, total_counts))
    count_dict = sorted(count_dict, key=lambda x:x[1], reverse=True)[0:10]
    words = [w[0] for w in count_dict]
    counts = [w[1] for w in count_dict]
    x_pos = np.arange(len(words))

    plt.figure(2, figsize=(15, 5))
    plt.subplot(title='10 most common words')
    sns.set_context("notebook", font_scale=1.25, rc={"lines.linewidth": 2})
    sns.barplot(x_pos, counts)
    plt.xticks(x_pos, words, rotation=90)
    plt.xlabel('words')
    plt.ylabel('counts')
    plt.show()

# Fit and transform the processed titles
count_data = vectorizer.fit_transform(df['text'])
# Visualise the 10 most common words
plot_10_most_common_words(count_data, vectorizer)
```



Latent Dirichlet Allocation (LDA)

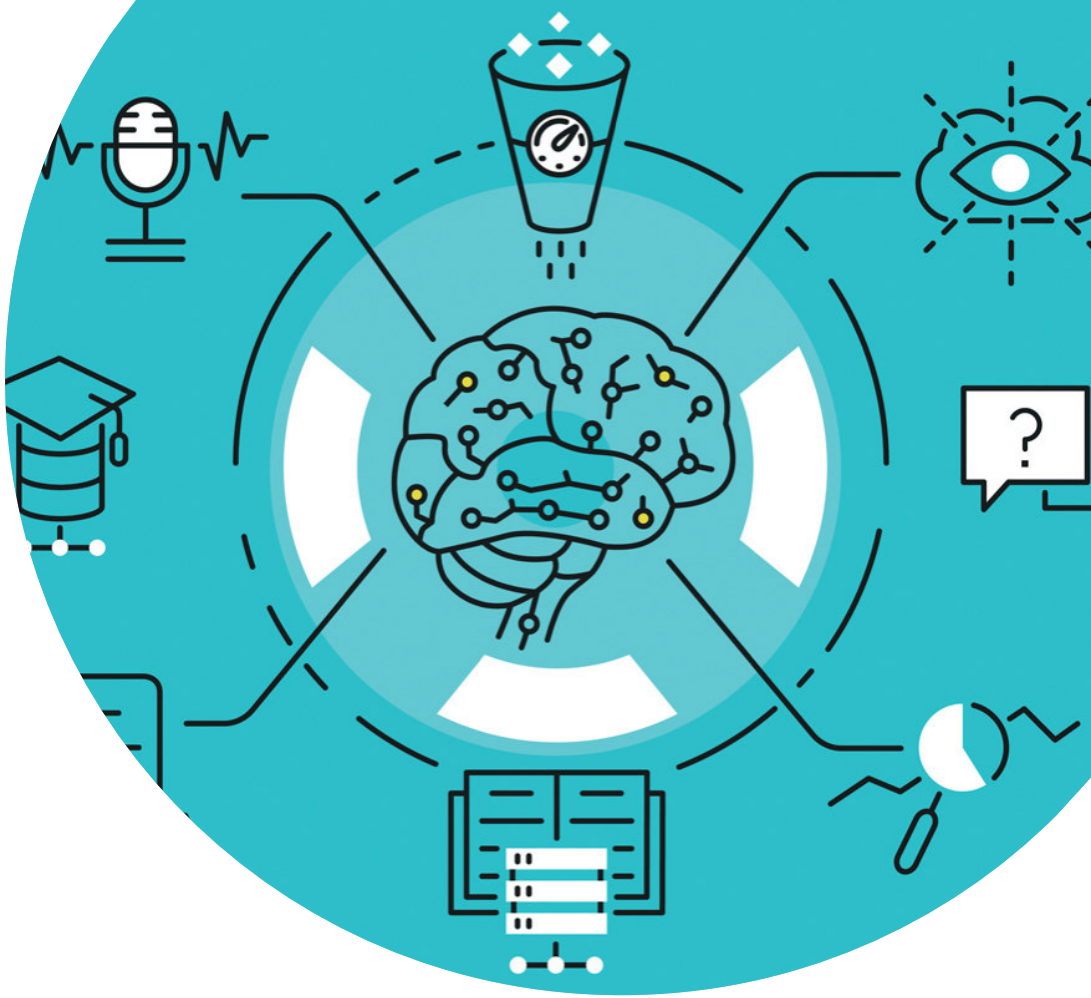
Unsupervised training has naturally segmented a few popular topics from the dataset including themes like LinkedIn, DnD & Adobe (Class 1), gratitude content (Class 2), but no easily visible tweets from Class 3 (positivity) & 4 (replies).

```
Topics found via LDA:

Topic #0:
dm send pls congrats adobeinsiders life shoot video awnewyork heart

Topic #1:
millennialtalk got winniesun today love just awesome year think linkedin

Topic #2:
adobesummit yep right woo happy ooo brand bday people john

Topic #3:
thank amazing omg hahaha cute love let lol sis people

Topic #4:
hi time wait say love friends ok thing feel new

Topic #5:
good did real just going ll love learn make like

Topic #6:
yes followed true sweet worth kerry time value based listen

Topic #7:
fun adobemax like nasasocial hello lets new sounds hahahaha just

Topic #8:
lovely wow youre content know soon oh thank hope think

Topic #9:
thanks sharing aww yay thank proud mention yup article new
```

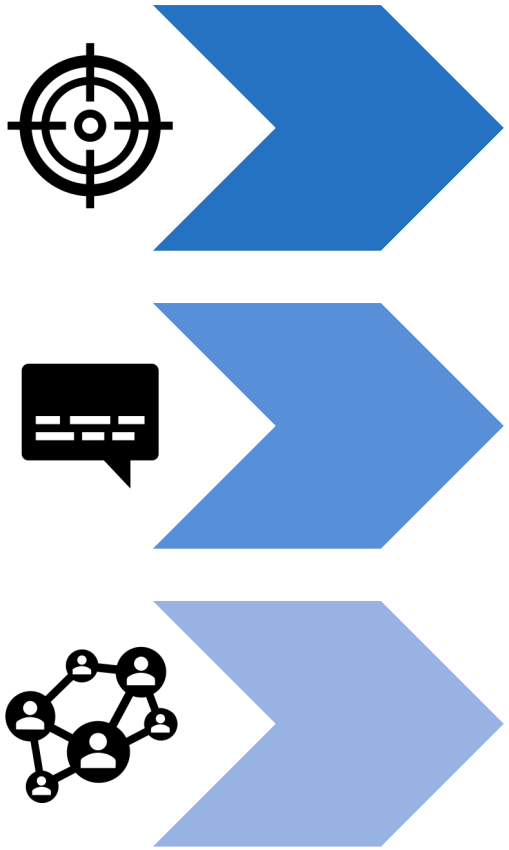
```python
# Load the LDA model from sk-learn
from sklearn.decomposition import LatentDirichletAllocation as LDA

# Helper function
def print_topics(model, vectorizer, n_top_words):
    words = vectorizer.get_feature_names()
    for topic_idx, topic in enumerate(model.components_):
        print("\nTopic #%d:" % topic_idx)
        print(" ".join([words[i]
                        for i in topic.argsort()[:-n_top_words - 1:-1]]))

# Tweak the two parameters below
number_topics = 10
number_words = 10
# Create and fit the LDA model
lda = LDA(n_components=number_topics, n_jobs=-1)
lda.fit(count_data)
# Print the topics found by the LDA model
print("Topics found via LDA:")
print_topics(lda, vectorizer, number_words)
```

MACHINE LEARNING MULTI-CLASSIFIER CONCLUSION

- A conversation on the twitter platform can be very unique and varied (e.g. the contextual use of images/gifs/emoji, replies, and changes in topics) in comparison to a traditional string expression from an email or product review.
- We anticipate a more accurate model could be created with more training data and more disciplined style of writing under the identified classes.



Performance Forecasting



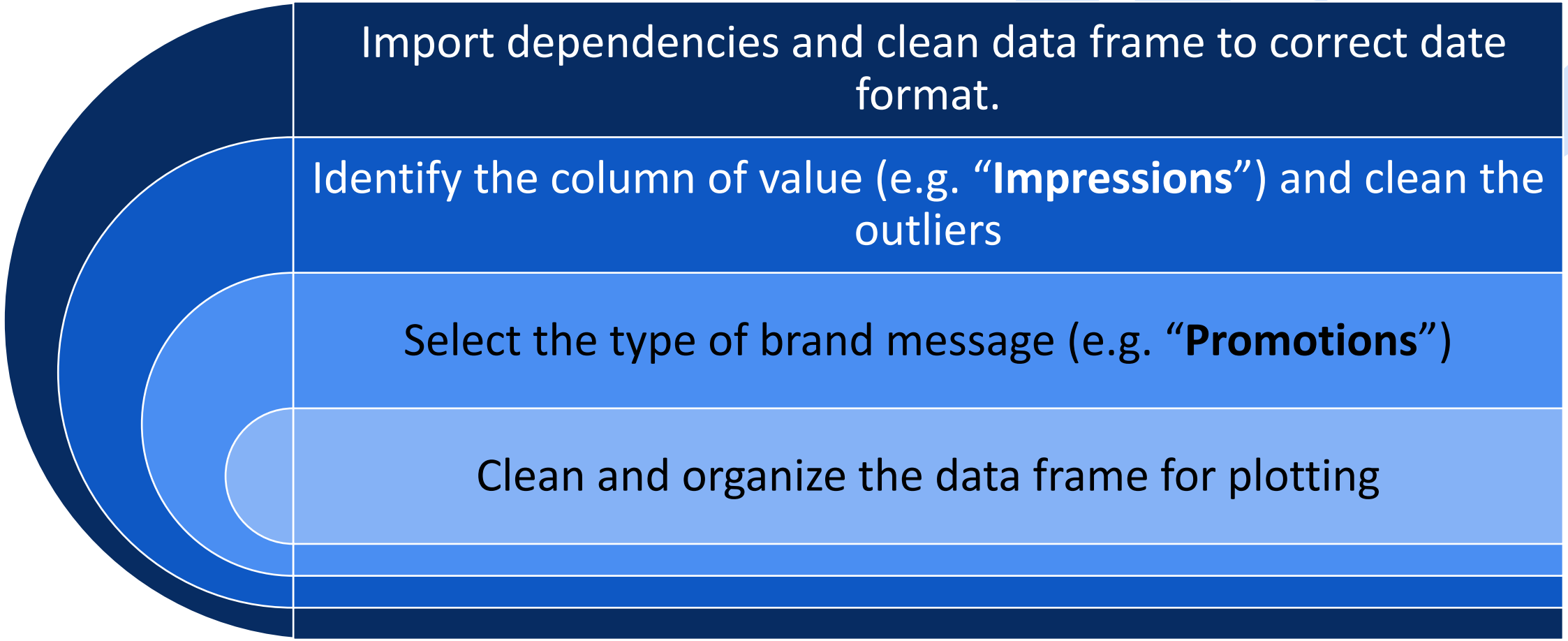
- We are using FB prophet to forecast future performance for brand goal setting.
- We will use prediction modeling for Promotions, Spotlight, and Positivity message types.
- Prediction modeling will be based on 5 criteria: Impressions, URL Clicks, Likes, Retweets, and Replies.

FB Prophet to Forecast Performance

Facebook Prophet provides a model that can be trained on time series data in order to predict future outcomes.

The Prophet object, requires a data frame with a date column and a y (output) column. After plotting a forecast, it will detect and mark points of change in the plot.

Steps to Forecast:



Train Model on Historical Data and Create the Prediction Forecast for Impressions

```
#Instantiate Prophet Class for Promotions-Impressions
m_promo_impr = Prophet(yearly_seasonality=True, changepoint_prior_scale=2.08)
```

```
#Train model on Impression Data for Promotional Posts
m_promo_impr.fit(promo_impr_df)
```

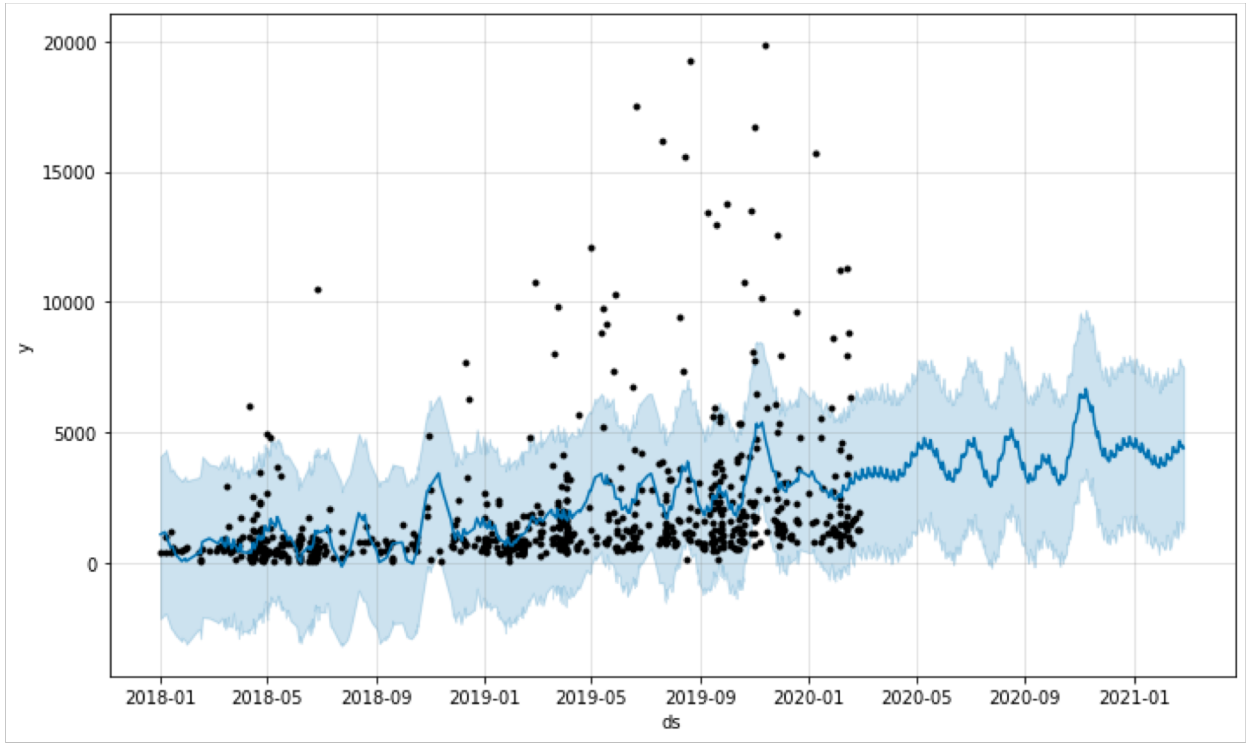
```
#Generate 365 days into future for x-axis
future_promo_impr = m_promo_impr.make_future_dataframe(periods=365)
future_promo_impr.tail()
```

	ds
1081	2021-02-22
1082	2021-02-23
1083	2021-02-24
1084	2021-02-25
1085	2021-02-26

```
#Predict Impressions of Promotional Posts for 1 Year
forecast_promo_impr = m_promo_impr.predict(future_promo_impr)
forecast_promo_impr[['ds', 'yhat', 'yhat_lower', 'yhat_upper']].tail()
```

	ds	yhat	yhat_lower	yhat_upper
1081	2021-02-22	4653.580291	1568.925530	7793.299133
1082	2021-02-23	4487.777414	1513.566027	7619.700657
1083	2021-02-24	4415.920280	1166.896254	7548.214343
1084	2021-02-25	4483.279939	1513.269340	7552.485850
1085	2021-02-26	4378.159428	1310.760416	7467.076730

```
#Plot Forecast of Impressions for Promotional Posts
forecast_promo_impr_plot = m_promo_impr.plot(forecast_promo_impr)
```



Calibrating your Prediction Model

The objective of this analysis is to determine the optimal changepoint value parameter when instantiating the Prophet model object. This is done by plotting a range of changepoint values vs the average margin error.

Loop through range of changepoint values

We test the model by tuning the changepoint parameter to each value in the range above.

Produce a prediction data frame, then isolate y-actual and y-theoretical values, and calculate margin error per entry.

Take the mean of all the margin errors and append to list; this list will be the y values for the plot below.

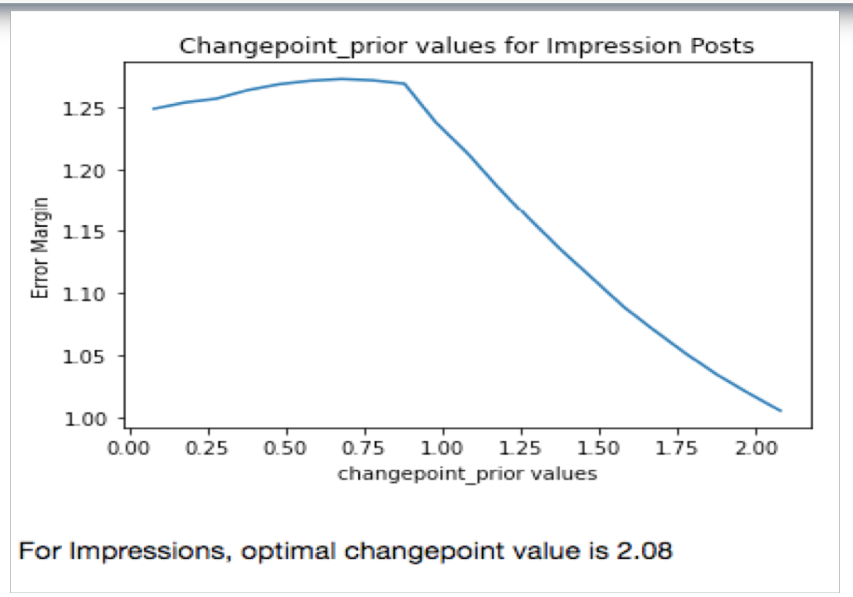
The changepoint value with the lowest average margin error will be the optimal parameter for the Prophet model. (2.08)

Create a range of viable changepoint values for Impressions

```
x_impr = np.arange(0.08, 2.1, 0.1)
x_impr
```

```
array([0.08, 0.18, 0.28, 0.38, 0.48, 0.58, 0.68, 0.78, 0.88, 0.98, 1.08,
       1.18, 1.28, 1.38, 1.48, 1.58, 1.68, 1.78, 1.88, 1.98, 2.08])
```

```
y = []
avg = []
for val in x_impr:
    m_promo_impr = Prophet(yearly_seasonality=True, changepoint_prior_scale=val)
    m_promo_impr.fit(promo_impr_df)
    future_promo_impr = m_promo_impr.make_future_dataframe(periods=365)
    forecast_promo_impr = m_promo_impr.predict(future_promo_impr)
    df_cv = cross_validation(m_promo_impr, initial='366 days', period='180 days',
                             horizon = '365 days')
    y_zipped = zip(df_cv["y"], df_cv["yhat"])
    for a,b in y_zipped:
        x = (a-b)/b
        y.append(abs(x))

    avg.append(mean(y))
```



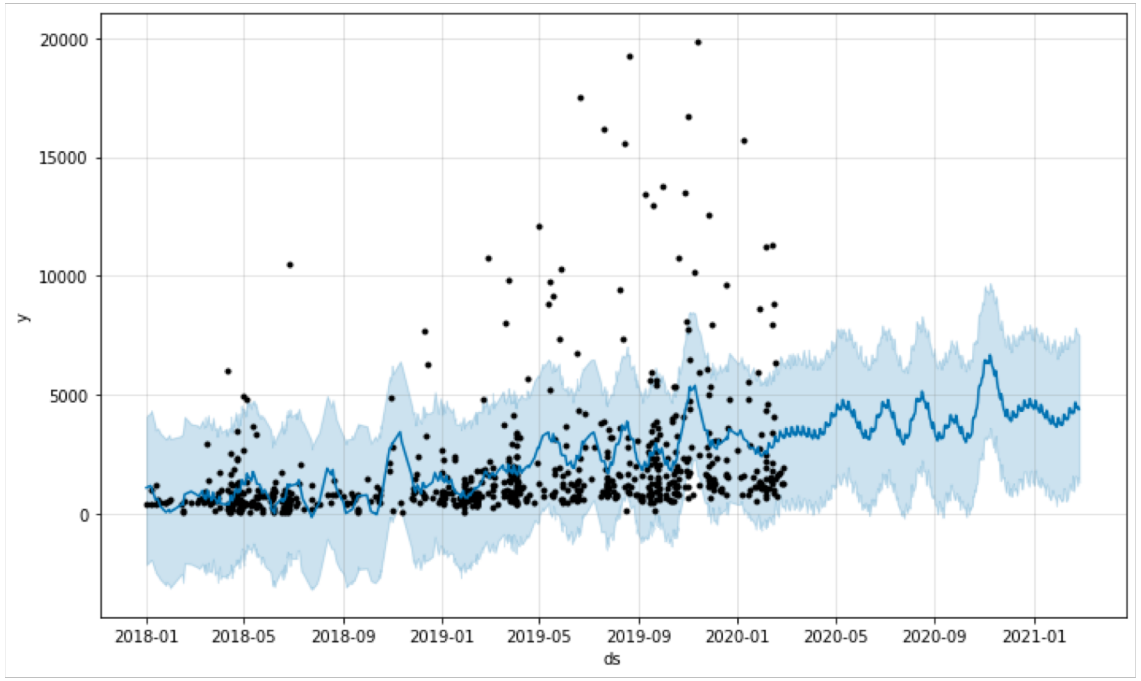
For Impressions, optimal changepoint value is 2.08

Comparison of Changepoint:

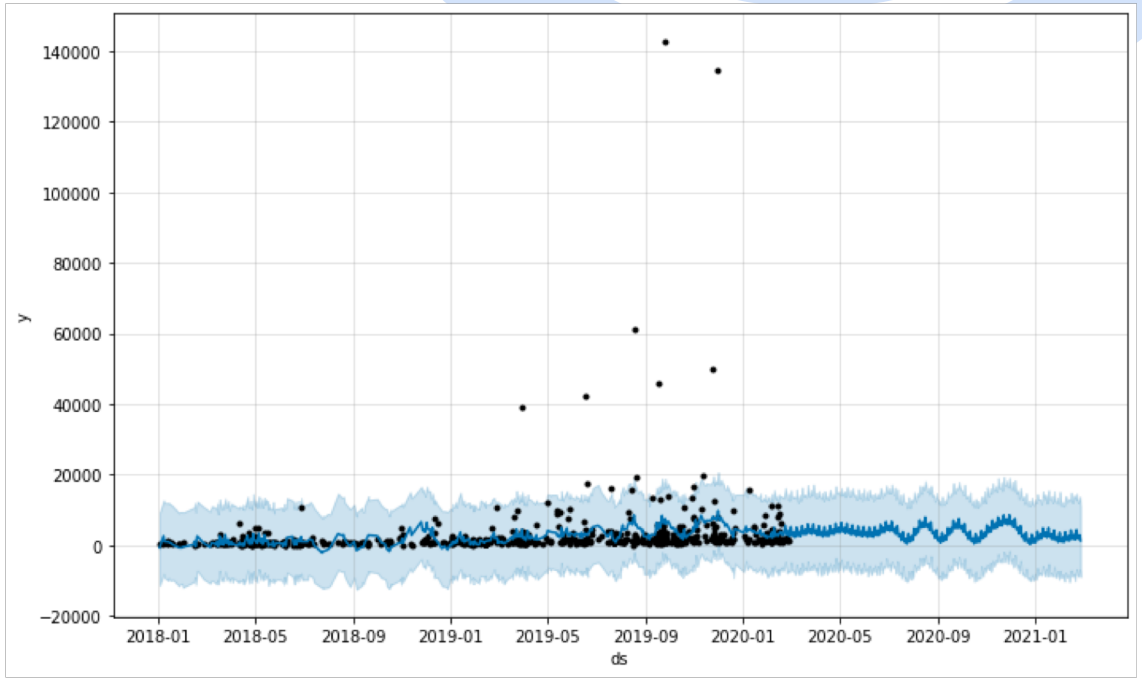
Finding the right Changepoint can change the prediction model by a lot.

Here is a comparison of Forecast of Impressions for Promotional Posts

Yearly Forecast Trend



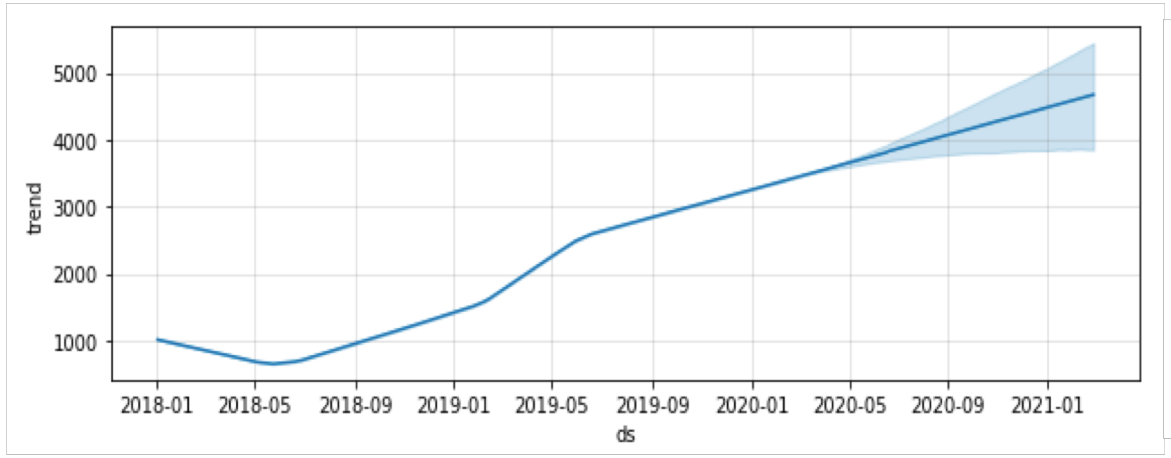
Changepoint scale of 2.08

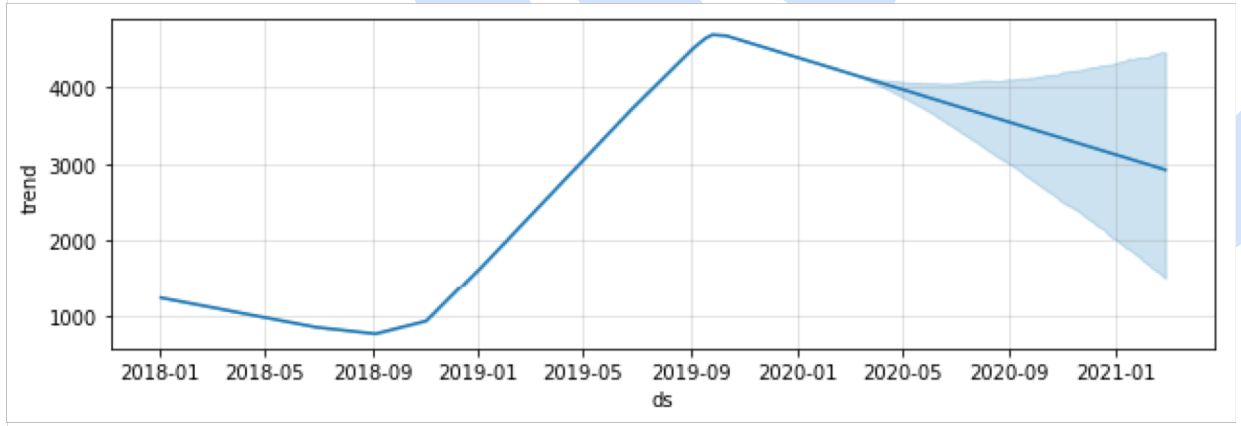


Changepoint scale of 0.5

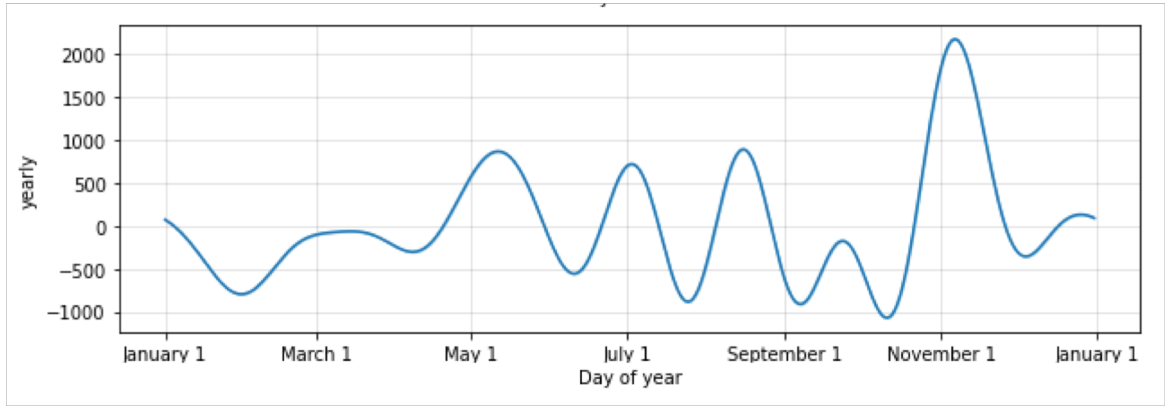
Comparison of Changepoint:

Yearly Forecast Trend

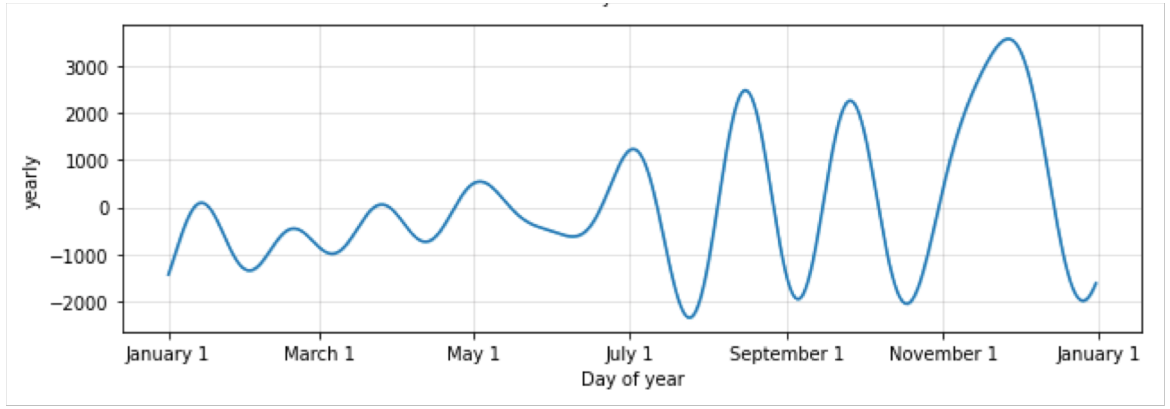




Seasonal Trend



Changepoint scale of 2.08

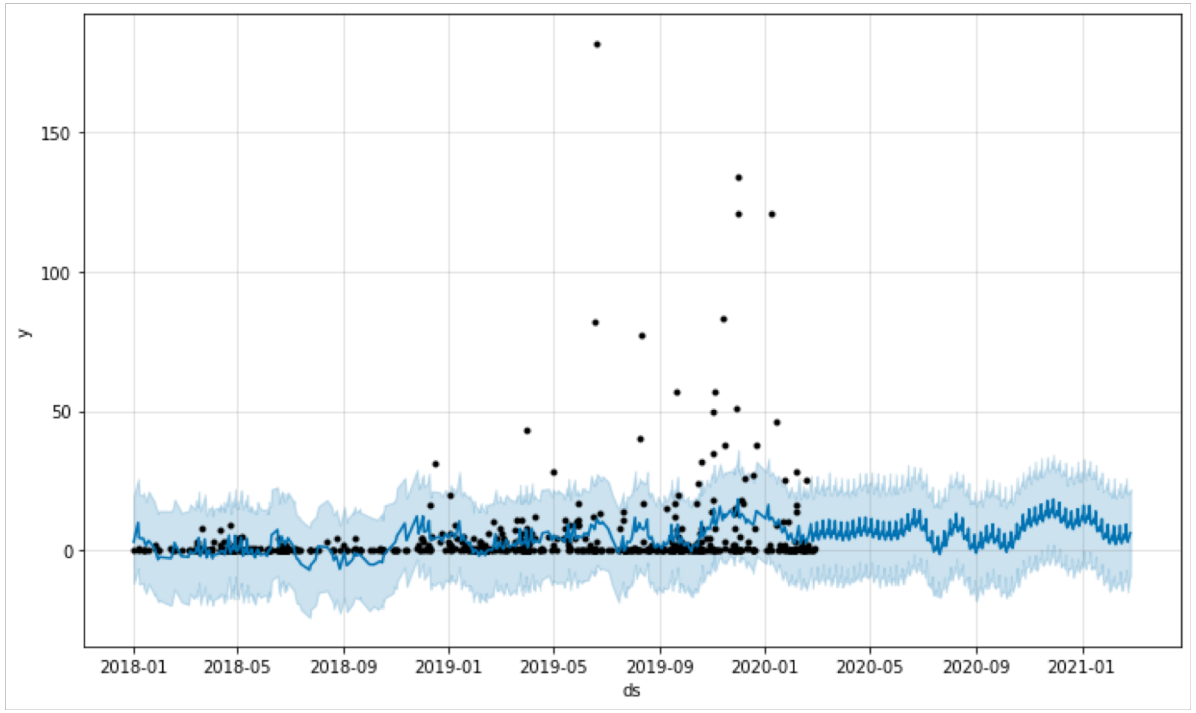


Changepoint scale of 0.5

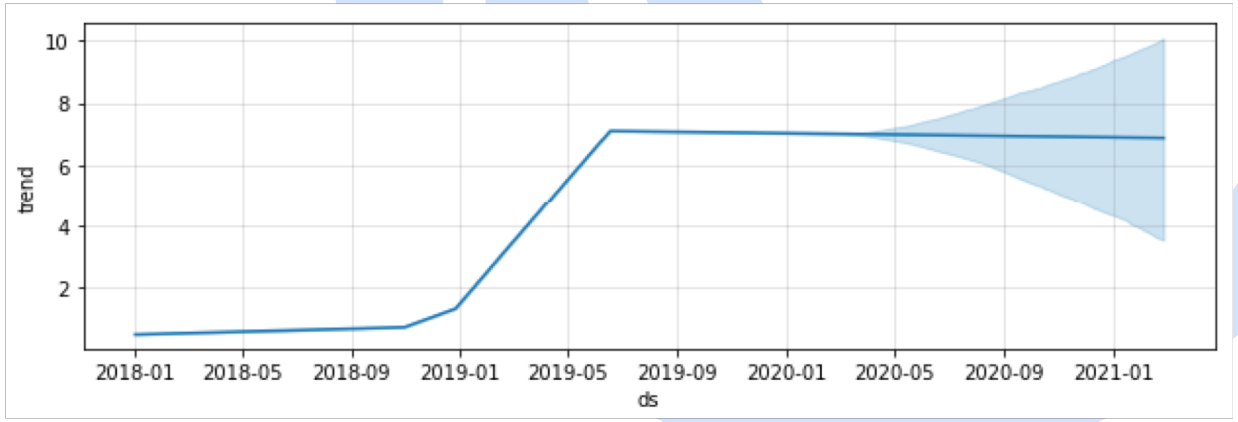
1. Promotional Forecast:

Promotional tweets are closely related to the brands business and usually involve a URL link and a call to action. We would put emphasis of viewing forecast of **clicks** for **promotional post**.

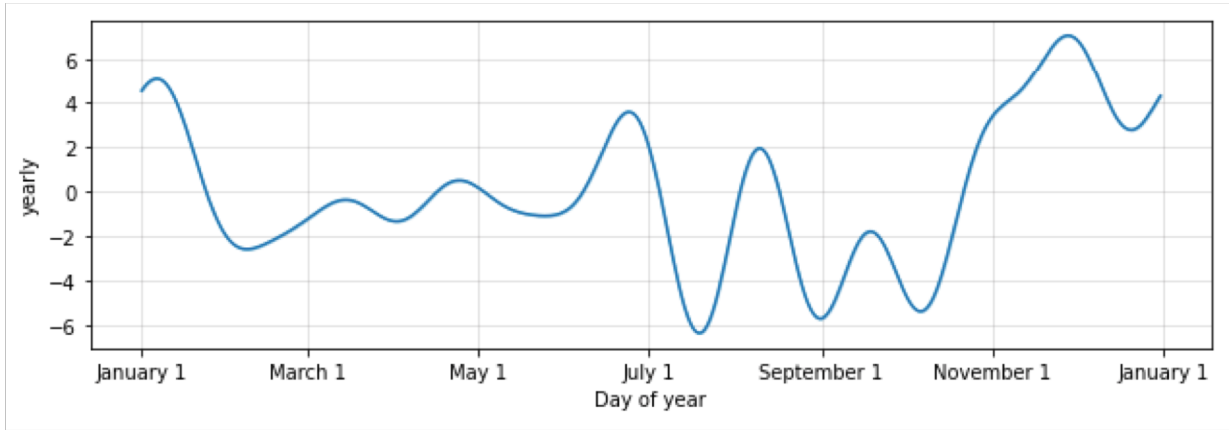
Yearly Forecast Trend



Daily Trend



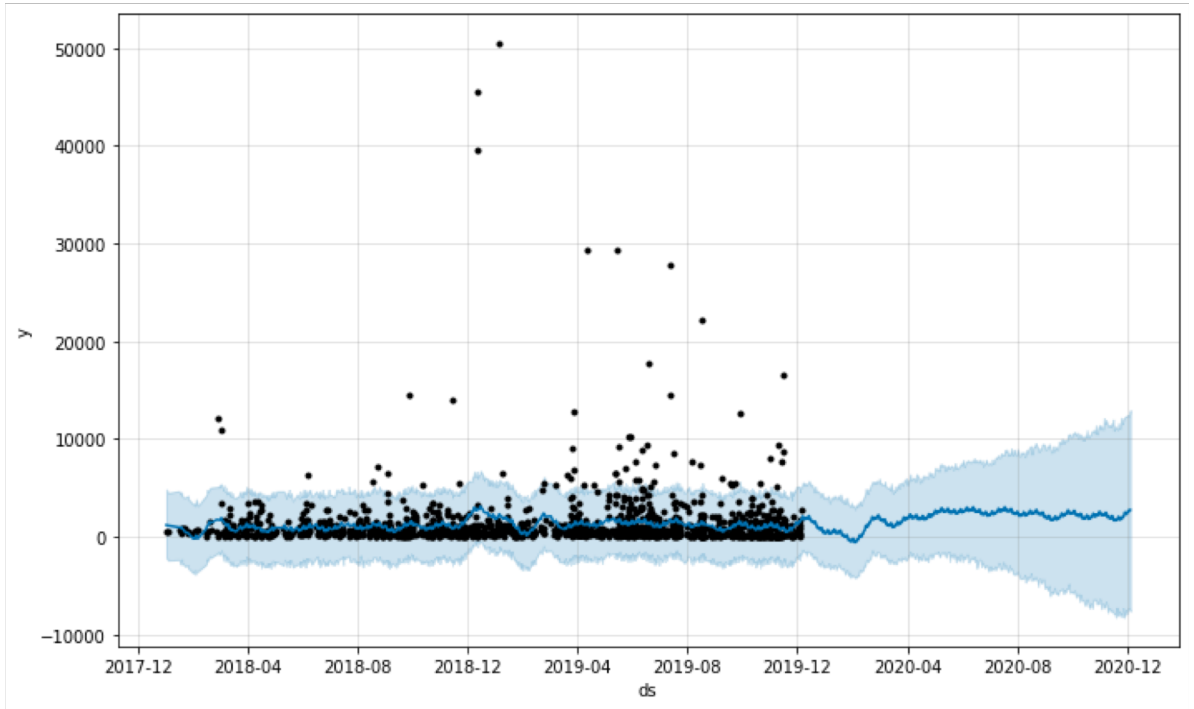
Seasonal Trend



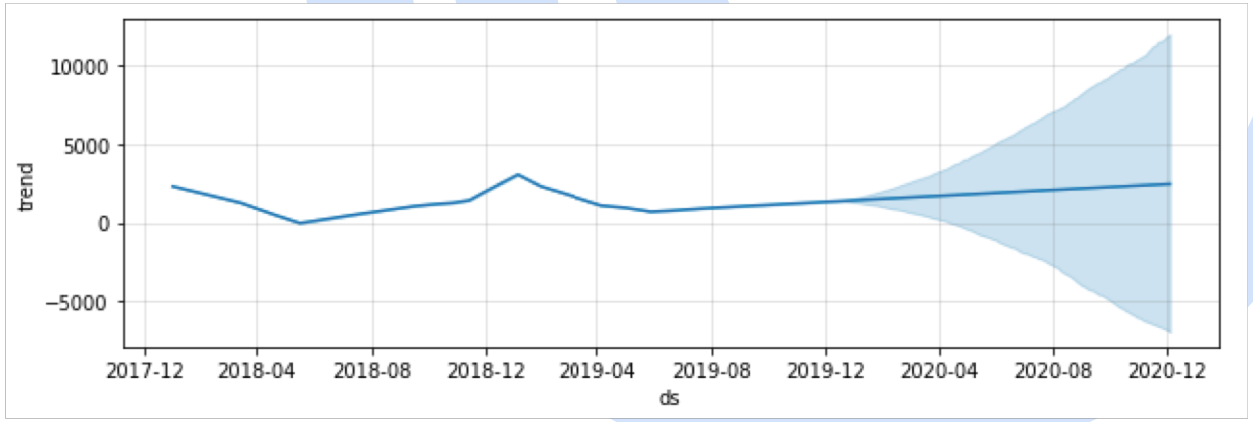
2. Spotlight Forecast:

Spotlight tweets are closely related to the brands recognition and twitter community involvement. We would put emphasis of viewing forecast of **Impressions** for **spotlight post**.

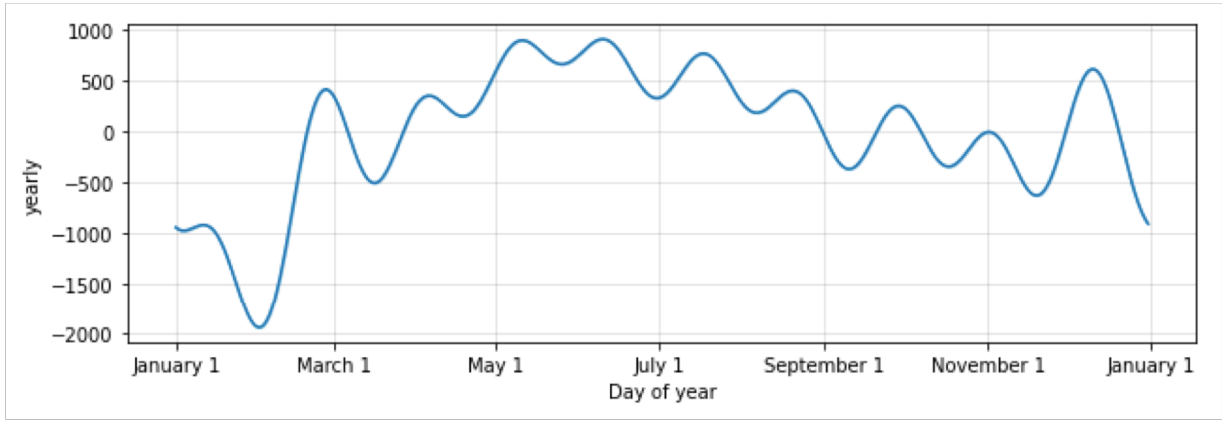
Yearly Forecast Trend



Daily Trend



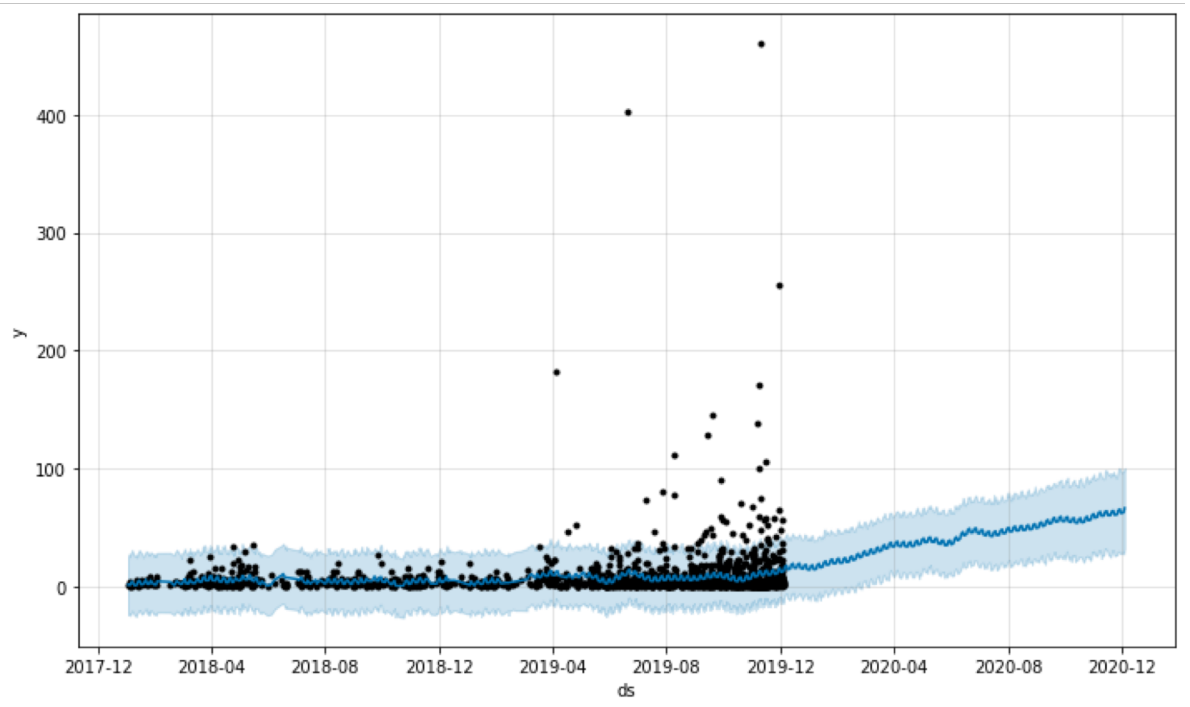
Seasonal Trend



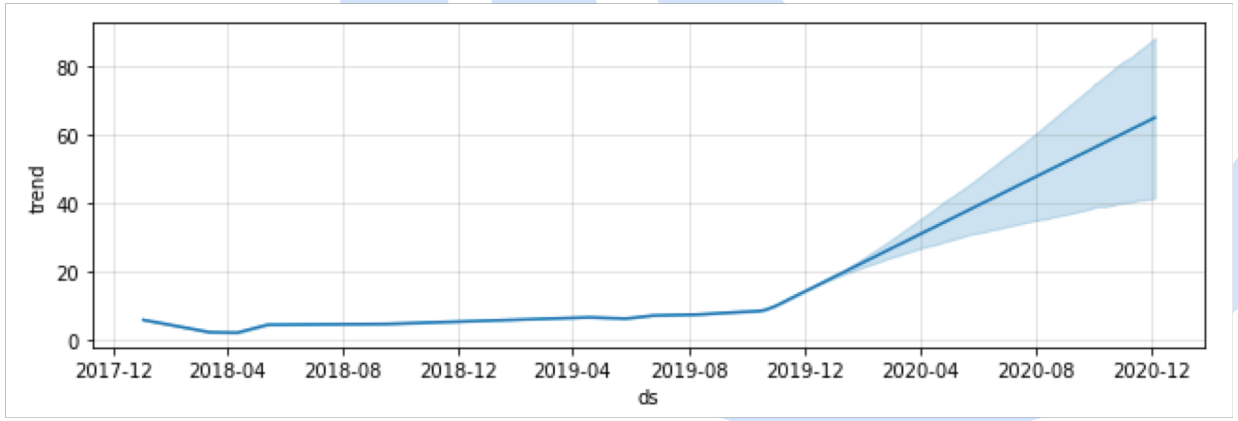
3. Positivity Forecast:

Positivity tweets are closely related to the brands perspective and values towards its audience. We would put emphasis of viewing forecast of **likes** to see if the audience is still engaging in **positivity post**.

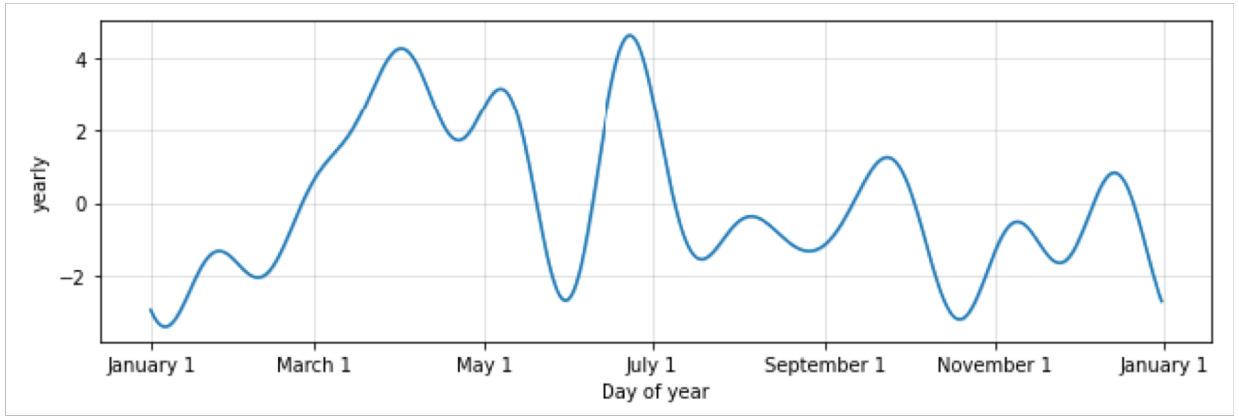
Yearly Forecast Trend



Daily Trend



Seasonal Trend



FB PROPHET – FORECAST CONCLUSION

- Forecast prediction can allow the client to understand where future performance can go with continuous performance on twitter.
- The model can help set a range of brand goals by predicting low or high margins on future tweets.
- Critical note: the dataset indicates that the social media branding/marketing industry is very volatile. Relative error is a better measure of accuracy rather than a standard scale.

NEXT STEPS:

Create an engaging dashboard to give clients a better understanding of brand impressions, engagement, and forecasting.

The client can use the dashboard to create brand goals for engagement and take more educated approaches to grow their brand.

Contoso

THANK YOU!