Capstone 2 Project Milestone 2 Report:

After we did initial exploratory data analysis on our **twitter feeds scraped** from March 30th to April 30th, 2020 we saw some interesting things including firstly that our dataset is huge, there were 4 gig of data with 1.5 million rows. We will start with taking a fraction of the dataset to begin **Natural Learning Process** on the tweets. Our goal is to discern sentiments and trends through unsupervised **topic clustering**.

Prepping our Data:

```
truncated_df_eng = combined_eng[['user_id', 'created_at', 'screen_name', 'text', 'is_quote',
                                 'is_retweet', 'favourites_count', 'retweet_count', 'followers_count',
                                 'friends_count', 'account_created_at']]
truncated_df_eng.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7315060 entries, 0 to 7315059
Data columns (total 11 columns):
 #   Column              Dtype 
---  ------              ----- 
 0   user_id             int64 
 1   created_at          object
 2   screen_name         object
 3   text                object
 4   is_quote            bool  
 5   is_retweet          bool  
 6   favourites_count    int64 
 7   retweet_count       int64 
 8   followers_count     int64 
 9   friends_count       int64 
 10  account_created_at  object
dtypes: bool(2), int64(5), object(4)
memory usage: 516.2+ MB
```

We have about 7.16 million rows of data. We reduce our data to about 100,000 rows by picking a random sample in order to work with our available resources.

```
# Selecting a random fraction sample of the full dataframe
trunc_df = truncated_df_eng.sample(frac=0.015)
trunc_df = trunc_df.drop_duplicates(subset = ["text"], keep='last')
```

```
trunc_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 109257 entries, 2852980 to 7274277
Data columns (total 12 columns):
 #   Column              Non-Null Count   Dtype              
---  ------              --------------   -----              
 0   user_id             109257 non-null  int64              
 1   created_at          109257 non-null  datetime64[ns, UTC]
 2   screen_name         109257 non-null  object             
 3   text                109257 non-null  object             
 4   is_quote            109257 non-null  bool               
 5   is_retweet          109257 non-null  bool               
 6   favourites_count    109257 non-null  int64              
 7   retweet_count       109257 non-null  int64              
 8   followers_count     109257 non-null  int64              
 9   friends_count       109257 non-null  int64              
 10  account_created_at  109257 non-null  datetime64[ns, UTC]
 11  account_age         109257 non-null  int64              
dtypes: bool(2), datetime64[ns, UTC](2), int64(6), object(2)
memory usage: 9.4+ MB
```

Clean, lemmatize, tokenize our text:

Our text data is contained in the “text” column. Cleaning up the text data is necessary to highlight attributes that we want our machine learning system to pick up on. We will use spaCy to process our texts in the following sequence:

1. Break our texts into tokens
2. Remove capitals and lemmatize
3. Remove stopwords and punctuations

```
# Importing spaCy natural learning processing library
import spacy
import en_core_web_sm
```

```
# Tokenizing our text to start
from spacy.lang.en.stop_words import STOP_WORDS
punctuations = string.punctuation
stopwords = list(STOP_WORDS)

parser = en_core_web_sm.load(disable=["tagger", "ner"])
parser.max_length = 500000

def spacy_tokenizer(sentence):
    mytokens = parser(sentence)
    mytokens = [ word.lemma_.lower().strip() if word.lemma_ != "-PRON-" else word.lower_ for word in mytokens ]
    mytokens = [ word for word in mytokens if word not in stopwords and word not in punctuations ]
    mytokens = " ".join([i for i in mytokens])
    return mytokens
```

```
trunc_df["processed_text"] = trunc_df["text"].progress_apply(spacy_tokenizer)
```

Vectorize our processed text

Once our data has been cleaned and tokenized we will scikit-learn to vectorize our data for machine learning. We us TfidfVectorizer to get the Tf-idf or the *term frequency inverse document frequency* weight for each token represented by:

$$w_{i,j} = tf_{i,j} \times \log\left(\frac{N}{df_i}\right)$$

This weight is a statistical importance of a word to a document in our data.

```
# Importing scikit-learn modules
from sklearn.feature_extraction.text import TfidfVectorizer
```

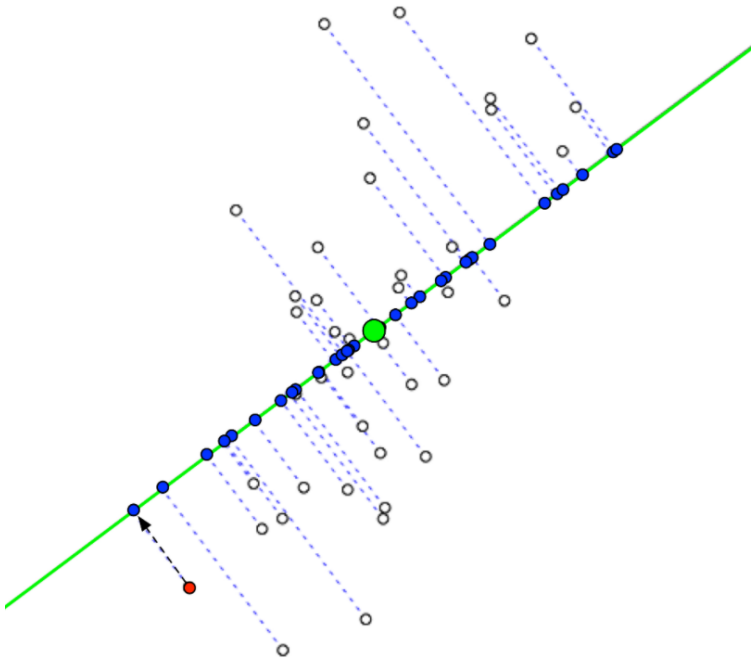
```
# Vectorizing text with scikit-learn TfidfVectorizer
def vectorize(text, features):

    vectorizer = TfidfVectorizer(max_features=features)
    X = vectorizer.fit_transform(text)
    return X
```

```
text = trunc_df['processed_text'].values
X = vectorize(text, 2 ** 11)
X.shape
```

```
(109257, 2048)
```

Once vectorized, our data takes the dimensions of 109,257 x 2,048. We next use scikit-learn to run **PCA** or principal component analysis **to reduce the dimensions of our data**. PCA calculates the most important variables and drops the least important variables by finding the direction of the line that maximizes the variance (the mean of squared distances from the projected points to the origin).



PCA reduces the dimensions down to 109,257 x 1,735.

```
from sklearn.decomposition import PCA

pca = PCA(n_components=0.95, random_state=42)
X_reduced= pca.fit_transform(X.toarray())
X_reduced.shape
```

```
(109257, 1735)
```

Use KMeans Clustering to discover topical clusters

Since our data isn't labeled we use kmeans as an unsupervised learning method to discover clusters to discern topics. Kmeans uses an iterative algorithm to locate centroids among k number clusters.

```
from sklearn.cluster import KMeans
```

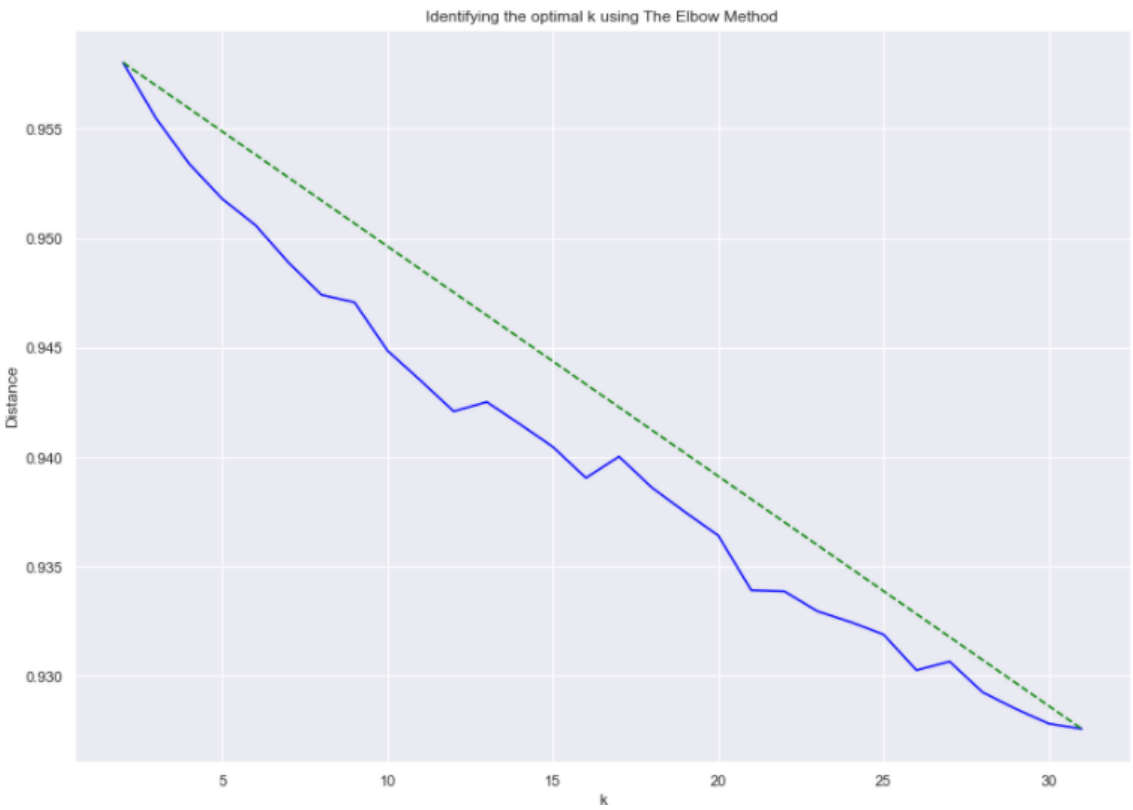
One way to find out the optimal number for k clusters is to use the elbow method. We iterate through different k clusters the distance between the sum of squares to its centroid for each k. The distance decreases at the bend of the elbow - this highlights the optimal k value.

```
from sklearn import metrics
from scipy.spatial.distance import cdist

# run kmeans with many different k
distance = []
K = range(2, 32)
for k in K:
    k_means = KMeans(n_clusters=k, random_state=42)
    k_means.fit(X_reduced)
    distance.append(sum(np.min(cdist(X_reduced, k_means.cluster_centers_, 'euclidean'), axis=1)) / X.shape[0])
```

```
X_line = [K[0], K[-1]]
Y_line = [distance[0], distance[-1]]

plt.plot(K, distance, color='blue')
plt.plot(X_line, Y_line, color='green', linestyle='dashed')
plt.xlabel('k')
plt.ylabel('Distance')
plt.title('Identifying the optimal k using The Elbow Method')
plt.show()
```



We see from the plot above the bend of the elbow starts to diminish at between k = 12. This is a good place to start.

```
k = 12
kmeans = KMeans(n_clusters=k, random_state=42, n_jobs=-1)
y_pred = kmeans.fit_predict(X_reduced)
trunc_df['y'] = y_pred
```

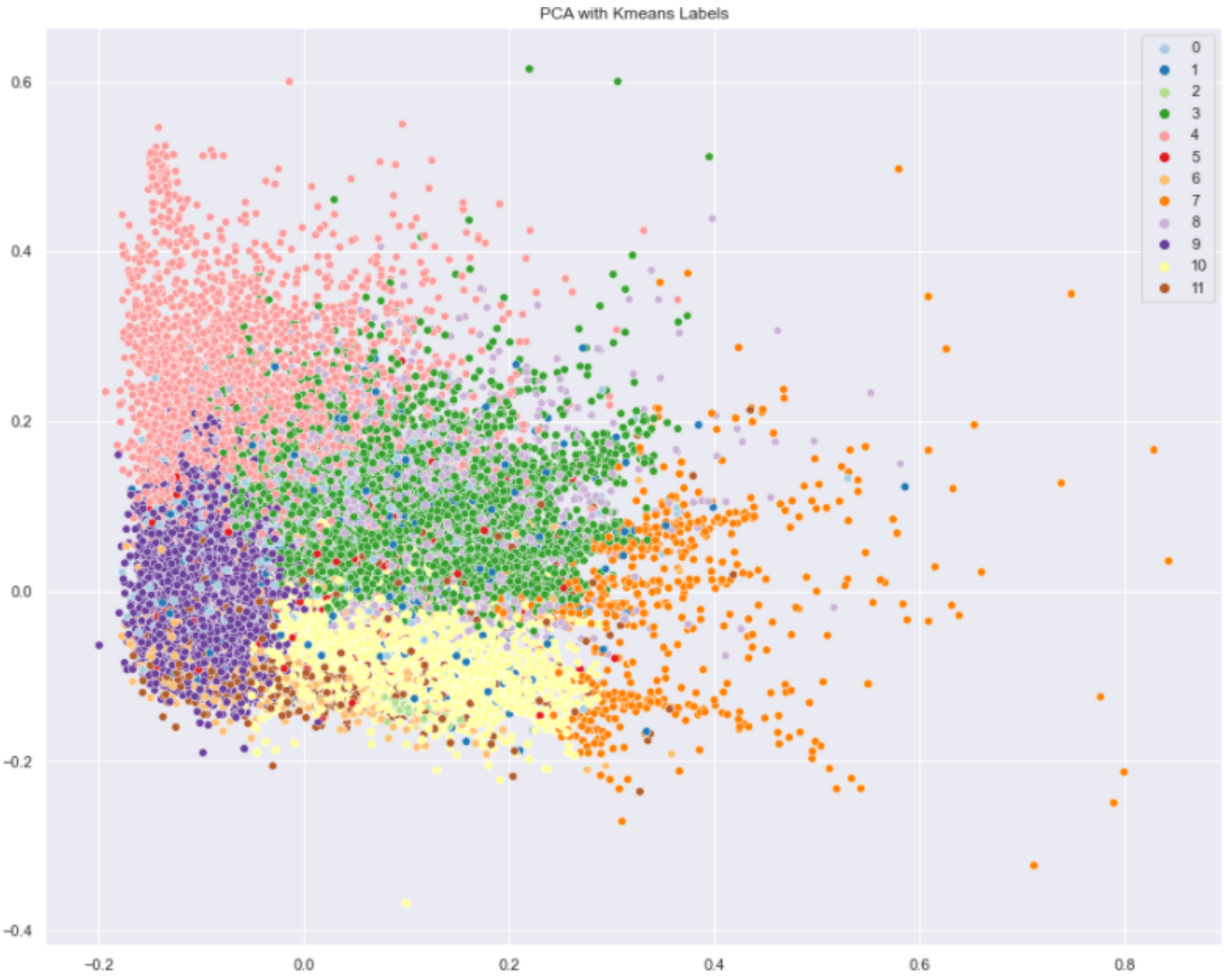
In the following pages we will plot our topic clusters to see how they turn out.

Plotting our Results:

1) PCA

```
# sns settings
sns.set(rc={'figure.figsize':(15, 12)})

# plot
sns.scatterplot(X_reduced[:,0], X_reduced[:,1], hue=y_pred, legend='full', palette='Paired')
sns.color_palette()
plt.title('PCA with Kmeans Labels')
plt.show()
```



We see there is some grouping of the clusters. The clusters do overlap alot but we do see some discernible grouping. Our legend shows 12 clusters. PCA may not be the best plot to show our clusters because it's a linear dimension reduction technique.

Evaluating our model

Next we want to evaluate how well the model classified the clusters. Remember we used kmeans to create 12 unsupervised clusters in y_pred?

```
k = 12
kmeans = KMeans(n_clusters=k, random_state=42, n_jobs=-1)
y_pred = kmeans.fit_predict(X_reduced)
trunc_df['y'] = y_pred
```

We can now use these 'labels' to see how well it can fit unseen data. We split the data into 80/20 train/test sets of our data.

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import precision_score, recall_score, mean_squared_error
import math
```

```
from sklearn.model_selection import train_test_split

# test set size of 20% of the data
X_train, X_test, y_train, y_test = train_test_split(X.toarray(),y_pred, test_size=0.2, random_state=42)
```

```
# Fit a basic Random Forest model
rf = RandomForestClassifier()
rf_model = rf.fit(X_train, y_train)
```

```
# Make predictions on the test set using the fit model
y_pred = rf_model.predict(X_test)
```

We get below precision and recall of 95% and accuracy of 98%.

```
# Evalute model predictions using precision and recall
precision = precision_score(y_test, y_pred, average='micro')
recall = recall_score(y_test, y_pred, average='micro')
mse = mean_squared_error(y_test, y_pred)
accuracy = math.sqrt(mse)
print('Precision: {}'.format(round(precision, 3)))
print('Recall: {}'.format(round(recall, 3)))
print('Accuracy: {}'.format(round(max(0,accuracy), 3)))
```

```
Precision: 0.95
Recall: 0.95
Accuracy: 0.987
```

Conclusion

We were able to use **unsupervised learning techniques** to perform **natural language processing** on a smaller portion of the entire set (there were originally 7.3 million rows of data, of which we randomly chose about 110,000 rows from). After we cleaned, tokenized and vectorized our text data, we decided on a few topic clustering methods to find out what important topics emerged. Using KMeans, the texts cluster into 12 optimal topics, we use PCA to reduce dimensionality. The plots show much better group clusters with PCA. The model evaluates to precision and recall of 95% and accuracy of 98%. Which is great.

In the next and final report we will extract out the keywords for each of the topic clusters using **Latent Dirichlet Allocation**.