# ARYAMAN MISHRA

# 19BCE1027

**Lab Exercise 6**

Take any text corpora, apply necessary preprocessing and

perform the k-means clustering on the corpora.

**Proposed Algorithm/Pseudocode:**

**Kmeans** algorithm is an iterative algorithm that tries to partition the dataset into *K*pre-defined distinct non-overlapping subgroups (clusters) where each data point belongs to **only one group**. It tries to make the intra-cluster data points as similar as possible while also keeping the clusters as different (far) as possible. It assigns data points to a cluster such that the sum of the squared distance between the data points and the cluster's centroid (arithmetic mean of all the data points that belong to that cluster) is at the minimum. The less variation we have within clusters, the more homogeneous (similar) the data points are within the same cluster.

The way kmeans algorithm works is as follows:

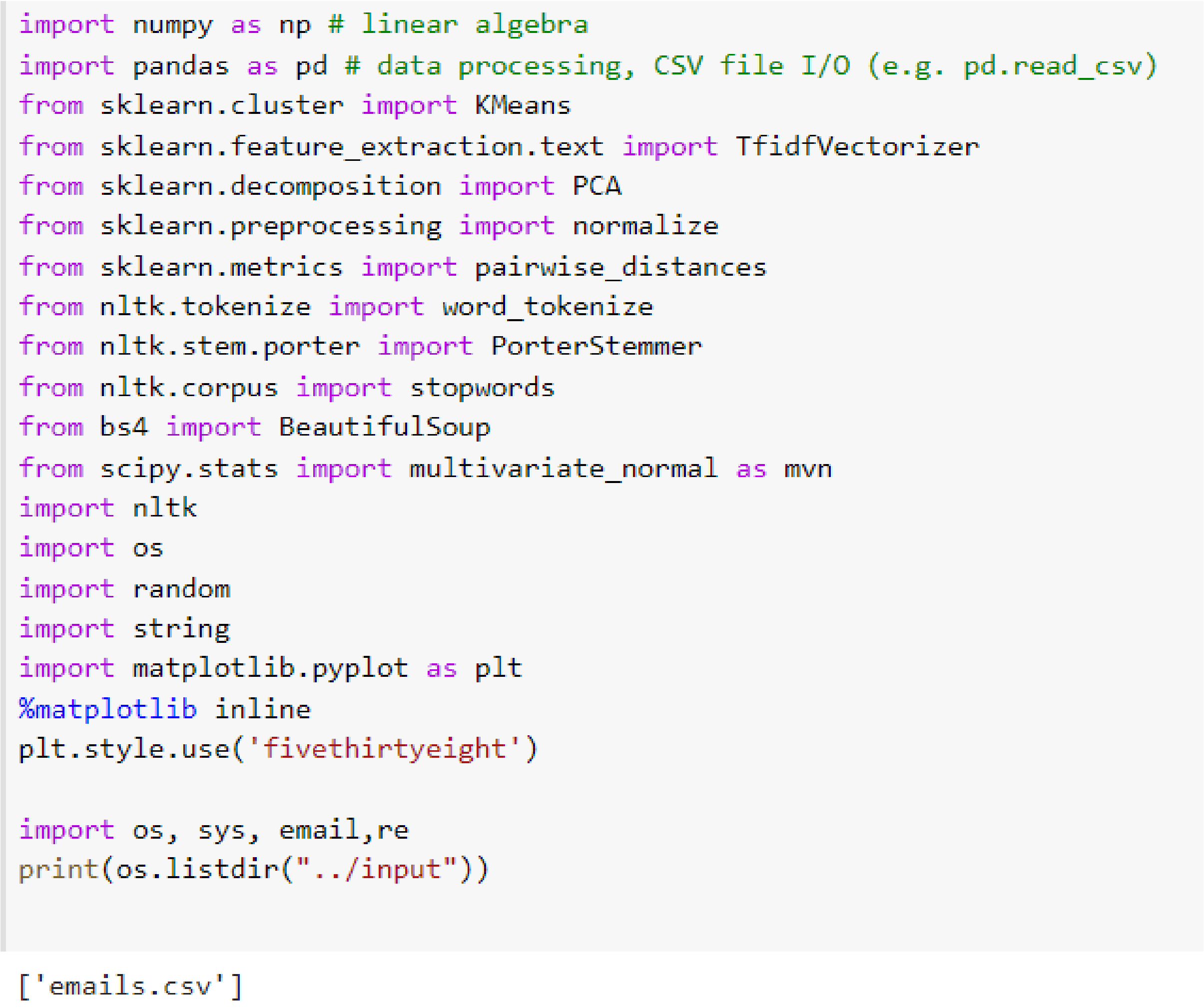
1. Specify number of clusters *K*.
2. Initialize centroids by first shuffling the dataset and then randomly selecting *K* data points for the centroids without replacement.
3. Keep iterating until there is no change to the centroids. i.e assignment of data points to clusters isn't changing.

- Compute the sum of the squared distance between data points and all centroids.
- Assign each data point to the closest cluster (centroid).

- Compute the centroids for the clusters by taking the average of the all data points that belong to each cluster.

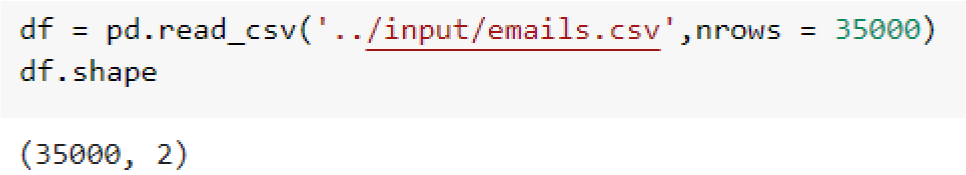
**Data Structure Proposed: 2D-Arrays, Dictionaries.**

**IMPLEMENTATION CODE AND RESULTS:**



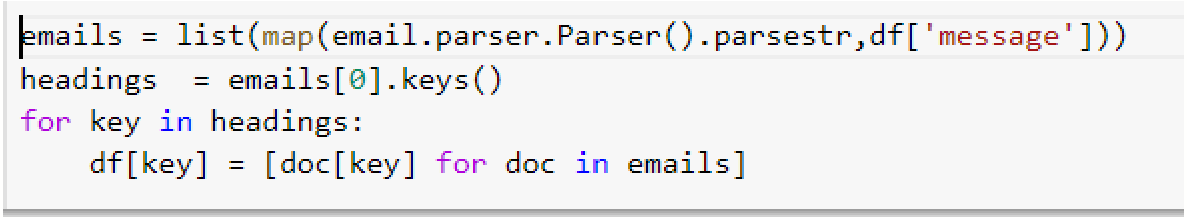
```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
from sklearn.cluster import KMeans
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.decomposition import PCA
from sklearn.preprocessing import normalize
from sklearn.metrics import pairwise_distances
from nltk.tokenize import word_tokenize
from nltk.stem.porter import PorterStemmer
from nltk.corpus import stopwords
from bs4 import BeautifulSoup
from scipy.stats import multivariate_normal as mvn
import nltk
import os
import random
import string
import matplotlib.pyplot as plt
%matplotlib inline
plt.style.use('fivethirtyeight')

import os, sys, email,re
print(os.listdir("../input"))
```

```
['emails.csv']
```

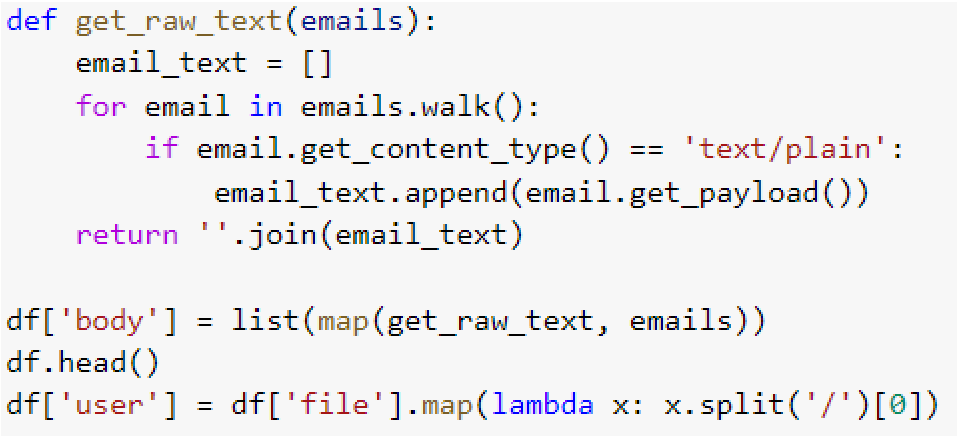


```
df = pd.read_csv('../input/emails.csv',nrows = 35000)
df.shape
```

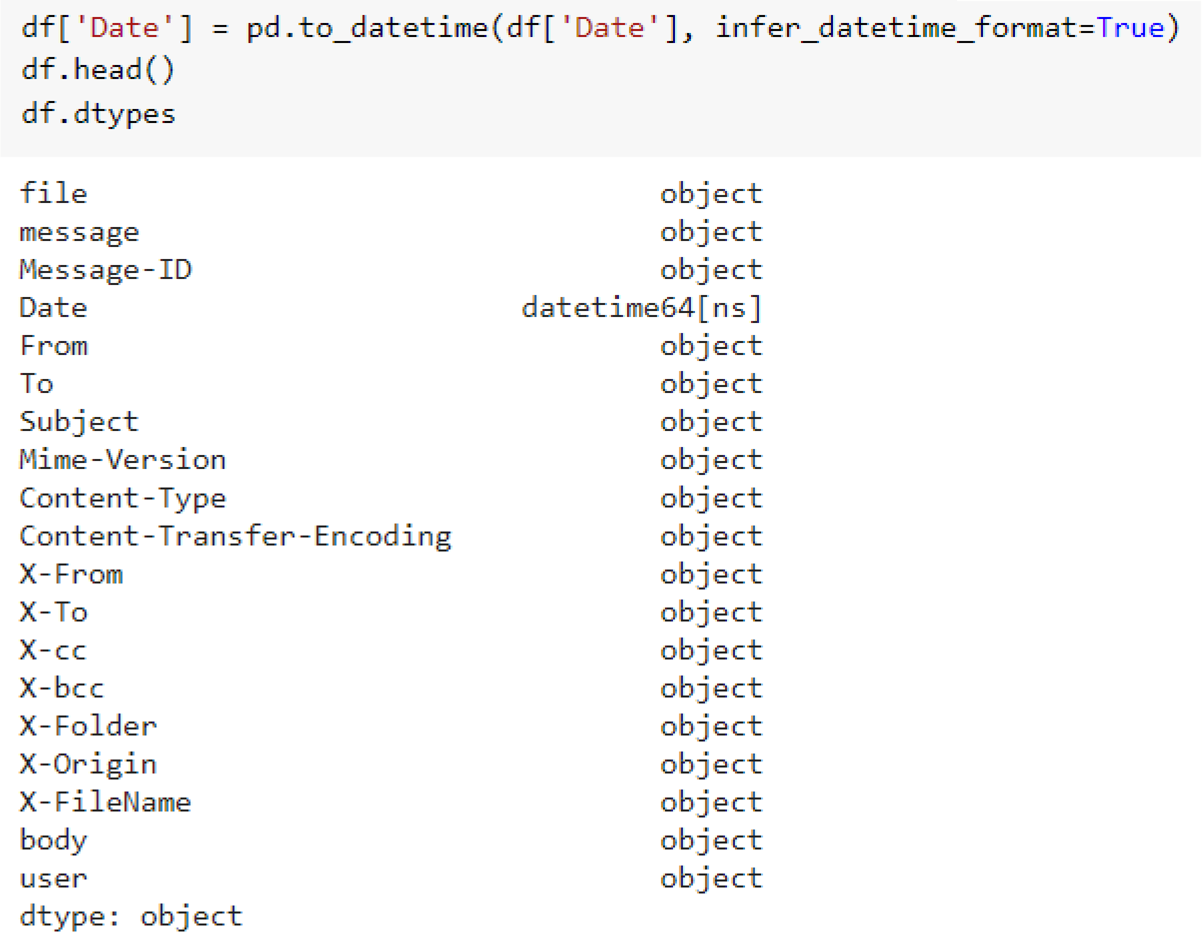
```
(35000, 2)
```



```
emails = list(map(email.parser.Parser().parsestr,df['message']))
headings  = emails[0].keys()
for key in headings:
    df[key] = [doc[key] for doc in emails]
```

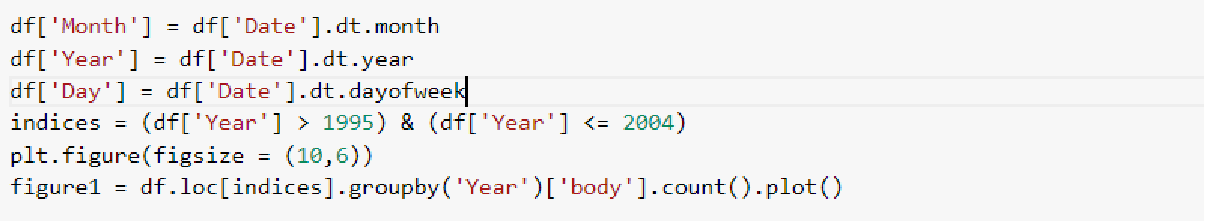


```python
def get_raw_text(emails):
    email_text = []
    for email in emails.walk():
        if email.get_content_type() == 'text/plain':
            email_text.append(email.get_payload())
    return ''.join(email_text)

df['body'] = list(map(get_raw_text, emails))
df.head()
df['user'] = df['file'].map(lambda x: x.split('/')[0])
```

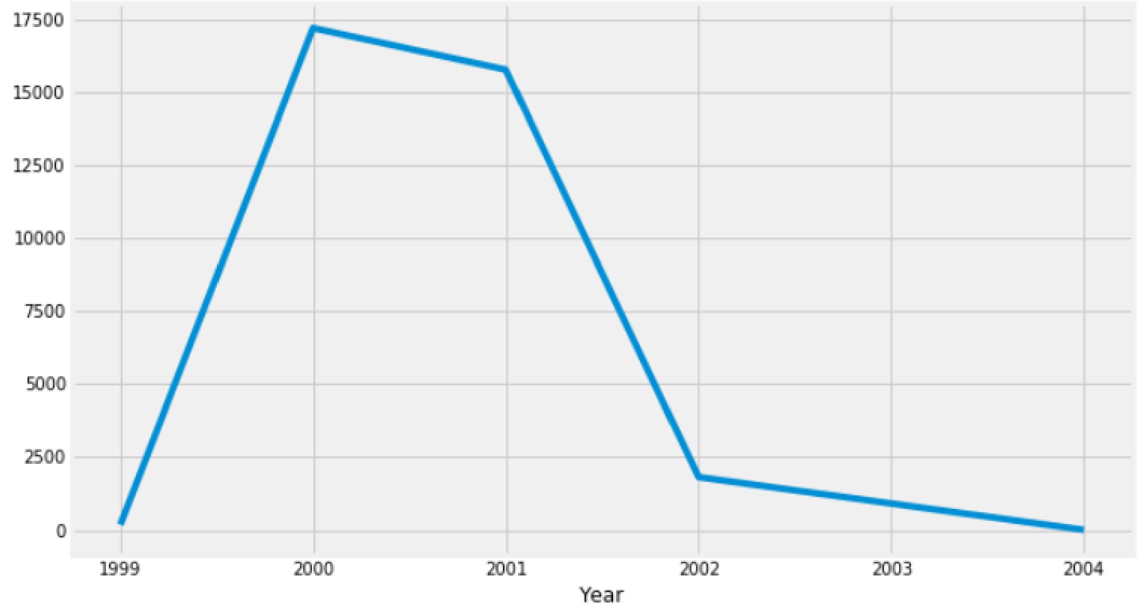


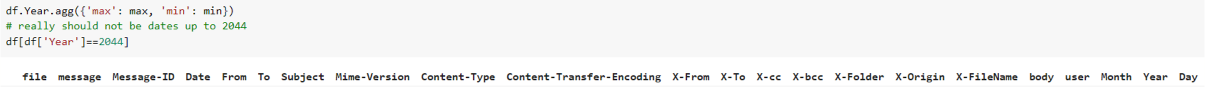
```python
df['Date'] = pd.to_datetime(df['Date'], infer_datetime_format=True)
df.head()
df.dtypes
```

```
file                                 object
message                              object
Message-ID                           object
Date                         datetime64[ns]
From                                 object
To                                   object
Subject                              object
Mime-Version                         object
Content-Type                         object
Content-Transfer-Encoding            object
X-From                               object
X-To                                 object
X-cc                                 object
X-bcc                                object
X-Folder                             object
X-Origin                             object
X-FileName                           object
body                                 object
user                                 object
dtype: object
```



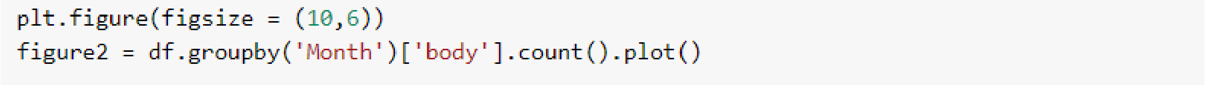
```python
df['Month'] = df['Date'].dt.month
df['Year'] = df['Date'].dt.year
df['Day'] = df['Date'].dt.dayofweek
indices = (df['Year'] > 1995) & (df['Year'] <= 2004)
plt.figure(figsize = (10,6))
figure1 = df.loc[indices].groupby('Year')['body'].count().plot()
```



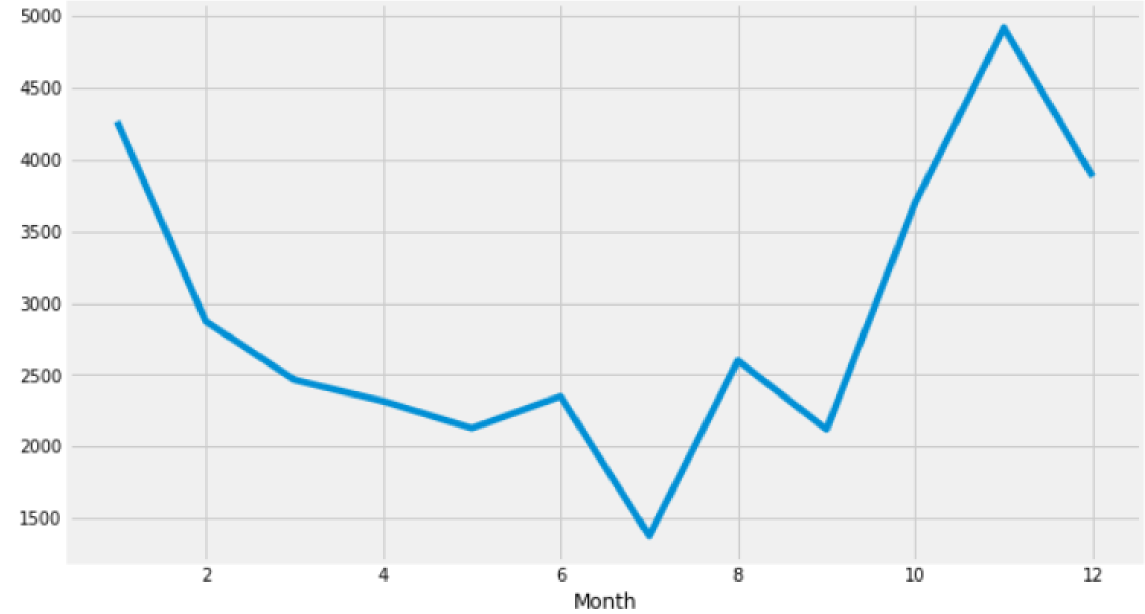


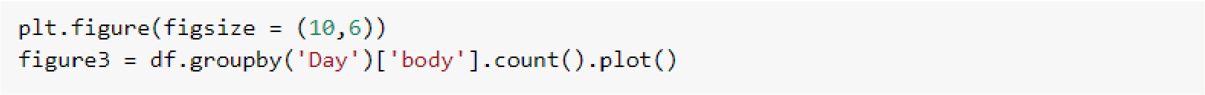
```python
df.Year.agg({'max': max, 'min': min})
# really should not be dates up to 2044
df[df['Year']==2044]
```

```
  file  message  Message-ID  Date  From  To  Subject  Mime-Version  Content-Type  Content-Transfer-Encoding  X-From  X-To  X-cc  X-bcc  X-Folder  X-Origin  X-FileName  body  user  Month  Year  Day
```

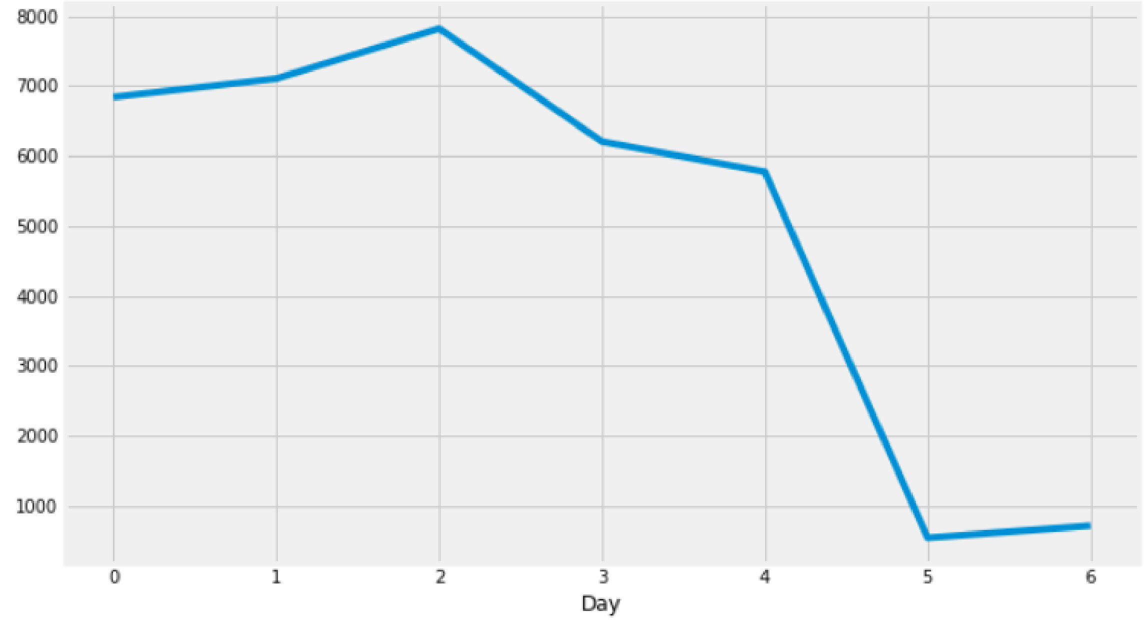


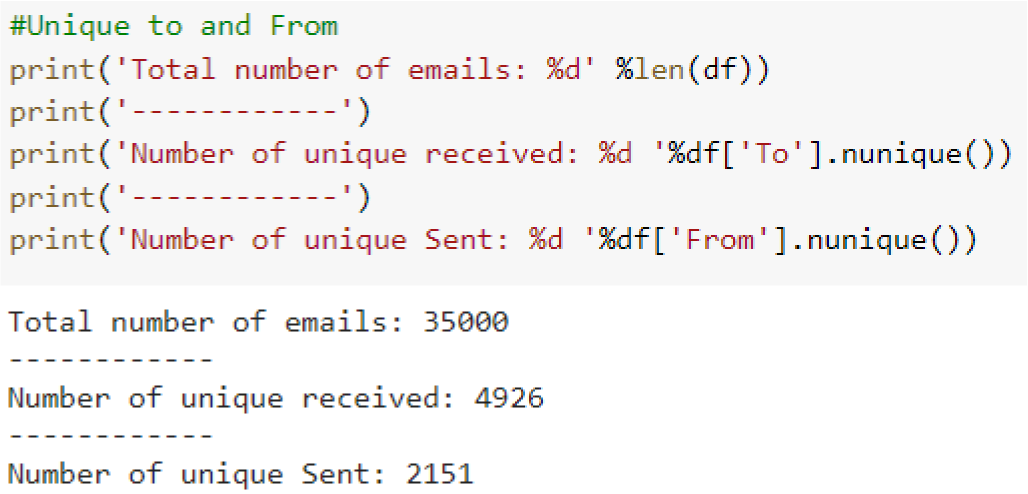
```python
plt.figure(figsize = (10,6))
figure2 = df.groupby('Month')['body'].count().plot()
```





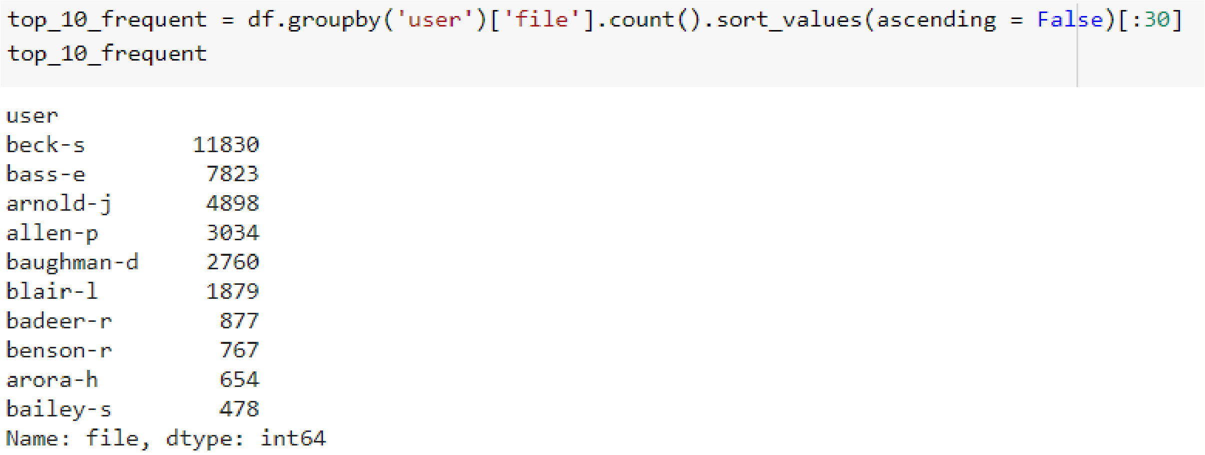
```
plt.figure(figsize = (10,6))
figure3 = df.groupby('Day')['body'].count().plot()
```





```
#Unique to and From
print('Total number of emails: %d' %len(df))
print('------------')
print('Number of unique received: %d '%df['To'].nunique())
print('------------')
print('Number of unique Sent: %d '%df['From'].nunique())
```

```
Total number of emails: 35000
------------
Number of unique received: 4926
------------
Number of unique Sent: 2151
```

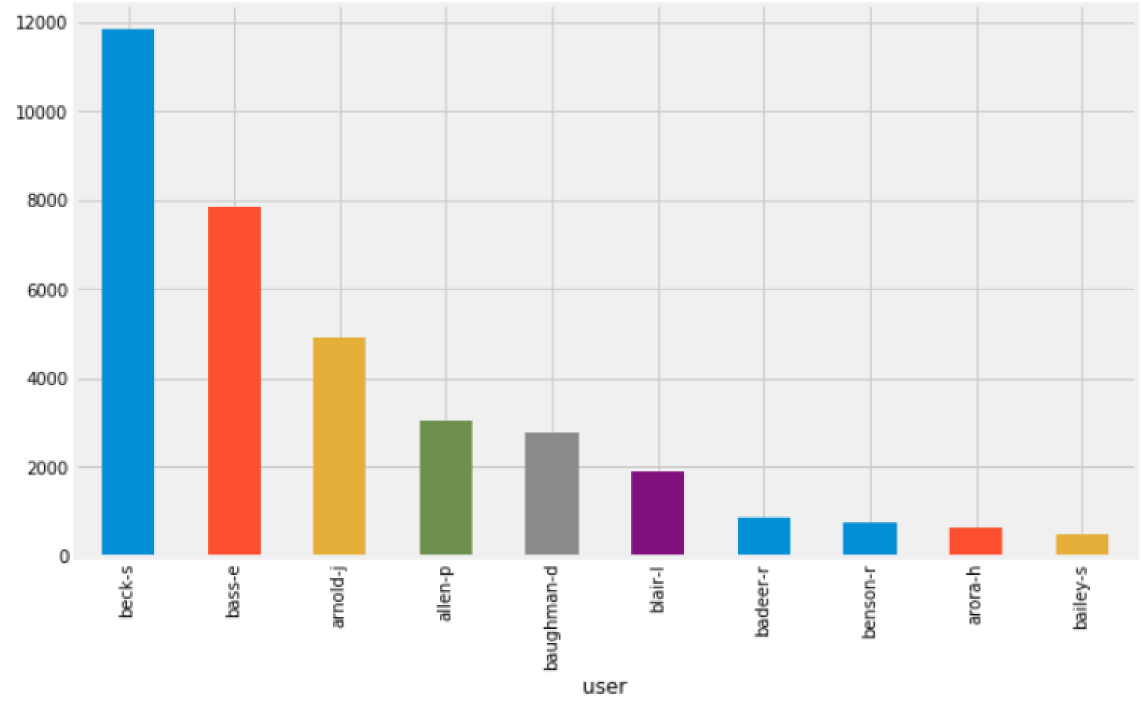


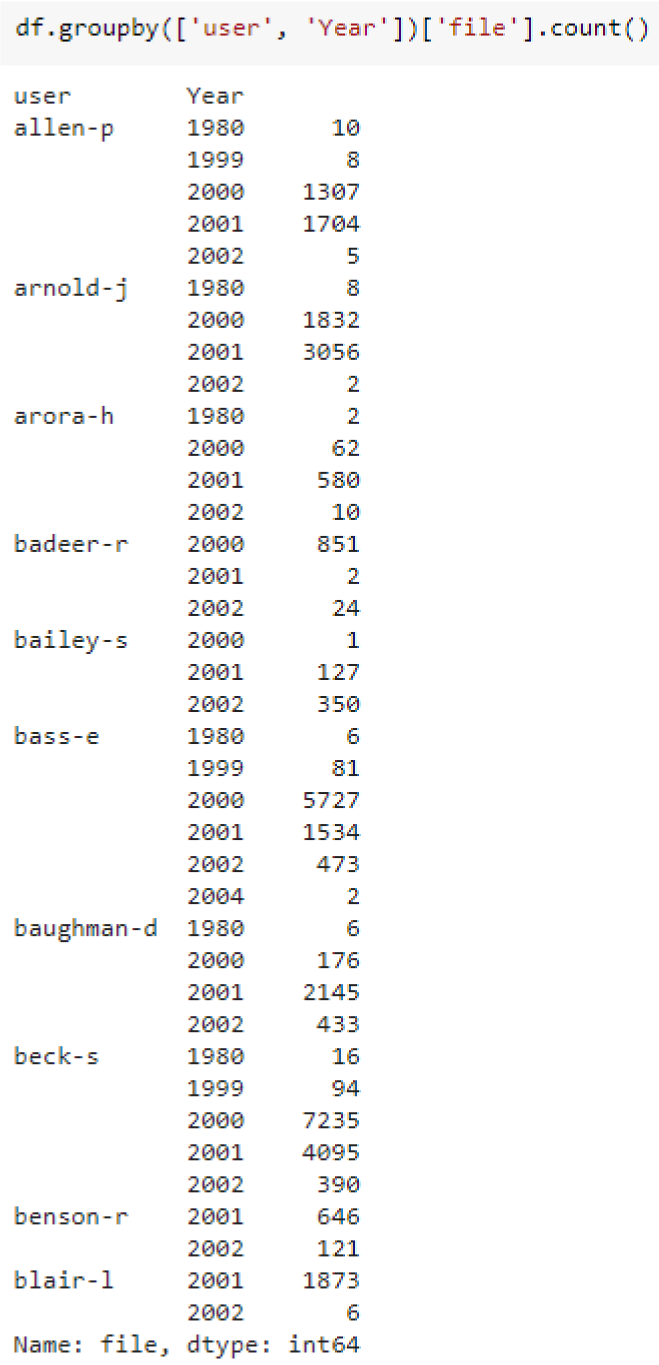
```
top_10_frequent = df.groupby('user')['file'].count().sort_values(ascending = False)[:30]
top_10_frequent
```

```
user
beck-s        11830
bass-e         7823
arnold-j       4898
allen-p        3034
baughman-d     2760
blair-l        1879
badeer-r        877
benson-r        767
arora-h         654
bailey-s        478
Name: file, dtype: int64
```

```
plt.figure(figsize = (10,6))
top_10_frequent.plot(kind = 'bar')
```

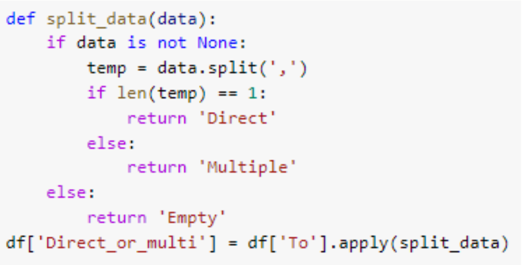
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f246e15f208>
```



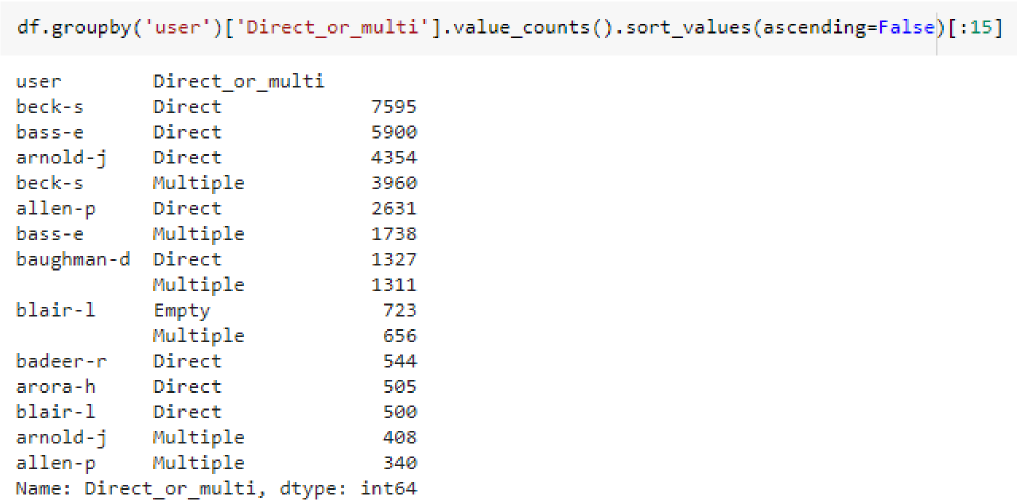


```
df.groupby(['user', 'Year'])['file'].count()
```

```
user        Year
allen-p     1980      10
            1999       8
            2000    1307
            2001    1704
            2002       5
arnold-j    1980       8
            2000    1832
            2001    3056
            2002       2
arora-h     1980       2
            2000      62
            2001     580
            2002      10
badeer-r    2000     851
            2001       2
            2002      24
bailey-s    2000       1
            2001     127
            2002     350
bass-e      1980       6
            1999      81
            2000    5727
            2001    1534
            2002     473
            2004       2
baughman-d  1980       6
            2000     176
            2001    2145
            2002     433
beck-s      1980      16
            1999      94
            2000    7235
            2001    4095
            2002     390
benson-r    2001     646
            2002     121
blair-l     2001    1873
            2002       6
Name: file, dtype: int64
```

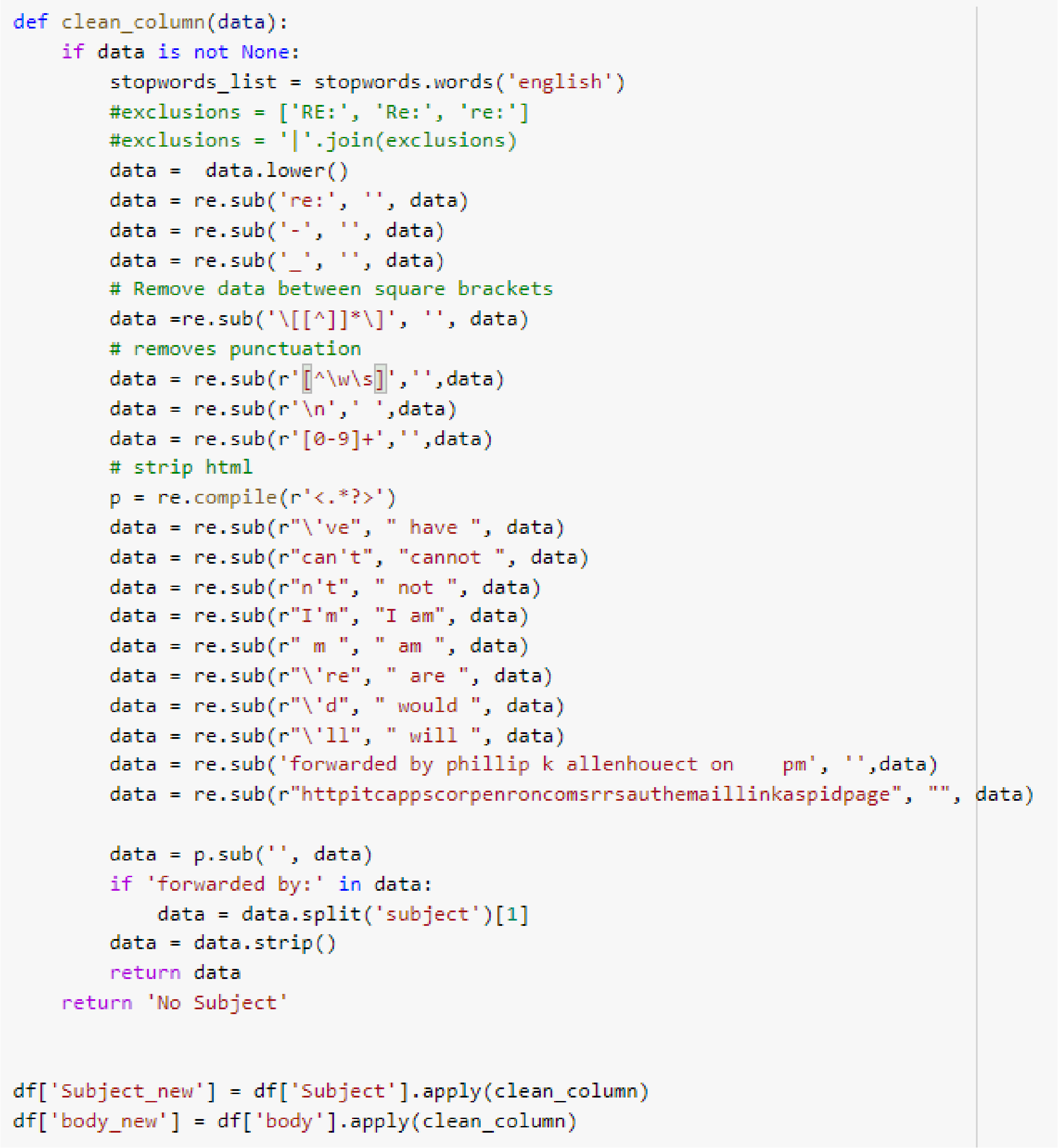


```
def split_data(data):
    if data is not None:
        temp = data.split(',')
        if len(temp) == 1:
            return 'Direct'
        else:
            return 'Multiple'
    else:
        return 'Empty'
df['Direct_or_multi'] = df['To'].apply(split_data)
```



```
df.groupby('user')['Direct_or_multi'].value_counts().sort_values(ascending=False)[:15]
```

```
user        Direct_or_multi
beck-s      Direct             7595
bass-e      Direct             5900
arnold-j    Direct             4354
beck-s      Multiple           3960
allen-p     Direct             2631
bass-e      Multiple           1738
baughman-d  Direct             1327
            Multiple           1311
blair-l     Empty               723
            Multiple            656
badeer-r    Direct              544
arora-h     Direct              505
blair-l     Direct              500
arnold-j    Multiple            408
allen-p     Multiple            340
Name: Direct_or_multi, dtype: int64
```



```
def clean_column(data):
    if data is not None:
        stopwords_list = stopwords.words('english')
        #exclusions = ['RE:', 'Re:', 're:']
        #exclusions = '|'.join(exclusions)
        data =  data.lower()
        data = re.sub('re:', '', data)
        data = re.sub('-', '', data)
        data = re.sub('_', '', data)
        # Remove data between square brackets
        data =re.sub('\[[^]]*\]', '', data)
        # removes punctuation
        data = re.sub(r'[^\w\s]','',data)
        data = re.sub(r'\n',' ',data)
        data = re.sub(r'[0-9]+','',data)
        # strip html
        p = re.compile(r'<.*?>')
        data = re.sub(r"\'ve", " have ", data)
        data = re.sub(r"can't", "cannot ", data)
        data = re.sub(r"n't", " not ", data)
        data = re.sub(r"I'm", "I am", data)
        data = re.sub(r" m ", " am ", data)
        data = re.sub(r"\'re", " are ", data)
        data = re.sub(r"\'d", " would ", data)
        data = re.sub(r"\'ll", " will ", data)
        data = re.sub('forwarded by phillip k allenhouect on    pm', '',data)
        data = re.sub(r"httpitcappscorpenroncomsrrsauthemaillinkaspidpage", "", data)

        data = p.sub('', data)
        if 'forwarded by:' in data:
            data = data.split('subject')[1]
        data = data.strip()
        return data
    return 'No Subject'


df['Subject_new'] = df['Subject'].apply(clean_column)
df['body_new'] = df['body'].apply(clean_column)
```

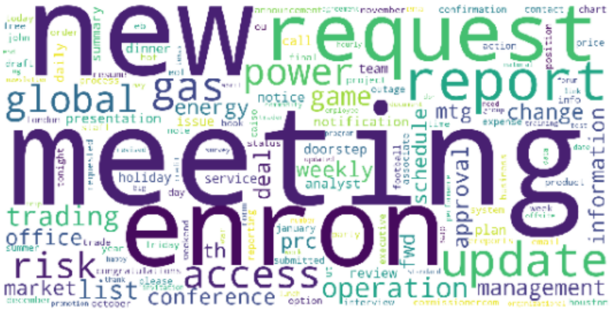
```
df['body_new'].head(5)
```

```
0                                  here is our forecast
1     traveling to have a business meeting takes the...
2                            test successful  way to go
3     randy   can you send me a schedule of the sala...
4                             lets shoot for tuesday at
Name: body_new, dtype: object
```

```
from wordcloud import WordCloud, STOPWORDS
stopwords = set(STOPWORDS)
to_add = ['FW', 'ga', 'httpitcappscorpenroncomsrrsauthemaillinkaspidpage', 'cc', 'aa', 'aaa', 'aaaa',
          'hou', 'cc', 'etc', 'subject', 'pm']

for i in to_add:
    stopwords.add(i)
```

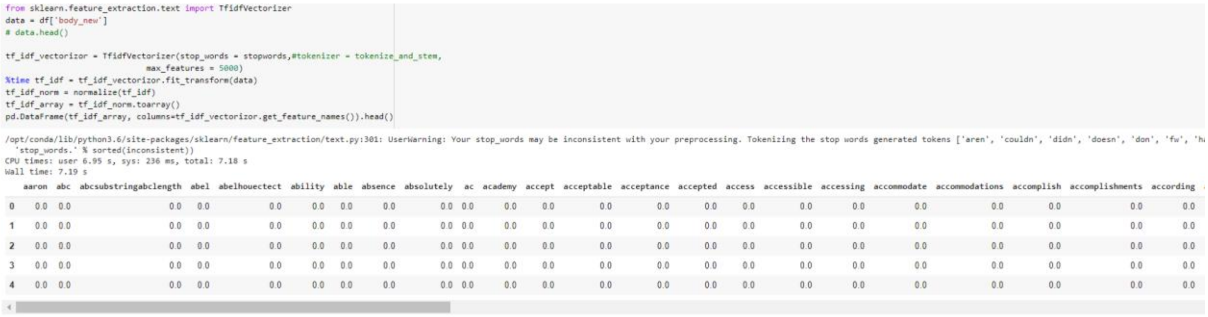
```
wordcloud = WordCloud(
                          collocations = False,
                          width=1600, height=800,
                          background_color='white',
                          stopwords=stopwords,
                          max_words=150,
                          #max_font_size=40,
                          random_state=42
                         ).generate(' '.join(df['Subject_new'])) # can't pass a series, needs to be strings and function computes frequencies
print(wordcloud)
plt.figure(figsize=(9,8))
fig = plt.figure(1)
plt.imshow(wordcloud)
plt.axis('off')
plt.show()
```

```
<wordcloud.wordcloud.WordCloud object at 0x7f246da0e7f0>
```



```
stemmer = PorterStemmer()
def stemming_tokenizer(str_input):
    words = re.sub(r"[^A-Za-z0-9\-]", " ", str_input).lower().split()
    words = [porter_stemmer.stem(word) for word in words]
    return words
```

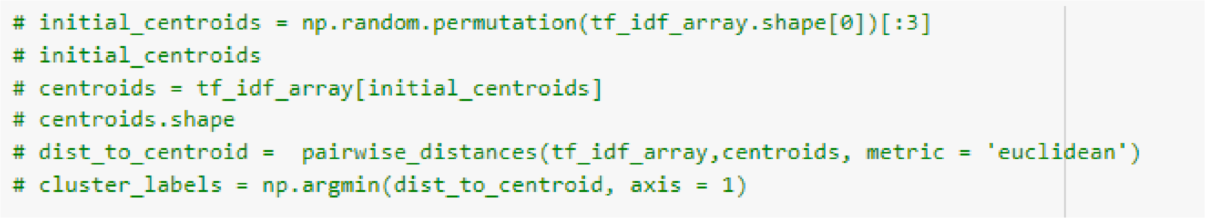
```
def tokenize_and_stem(text):
    # first tokenize by sentence, then by word to ensure that punctuation is caught as it's own token
    tokens = [word for sent in nltk.sent_tokenize(text) for word in nltk.word_tokenize(sent)]
    filtered_tokens = []
    # filter out any tokens not containing letters (e.g., numeric tokens, raw punctuation)
    for token in tokens:
        if re.search('[a-zA-Z]', token):
            filtered_tokens.append(token)
    stems = [stemmer.stem(t) for t in filtered_tokens]
    return stems
```



```
from sklearn.feature_extraction.text import TfidfVectorizer
data = df['body_new']
# data.head()

tf_idf_vectorizor = TfidfVectorizer(stop_words = stopwords,#tokenizer = tokenize_and_stem,
                             max_features = 5000)
%time tf_idf = tf_idf_vectorizor.fit_transform(data)
tf_idf_norm = normalize(tf_idf)
tf_idf_array = tf_idf_norm.toarray()
pd.DataFrame(tf_idf_array, columns=tf_idf_vectorizor.get_feature_names()).head()
```

```
/opt/conda/lib/python3.6/site-packages/sklearn/feature_extraction/text.py:301: UserWarning: Your stop_words may be inconsistent with your preprocessing. Tokenizing the stop words generated tokens ['aren', 'couldn', 'didn', 'doesn', 'don', 'fw', 'h
  'stop_words.' % sorted(inconsistent))
CPU times: user 6.95 s, sys: 236 ms, total: 7.18 s
Wall time: 7.19 s
```

| | aaron | abc | abcsubstringabclength | abel | abelhouectect | ability | able | absence | absolutely | ac | academy | accept | acceptable | acceptance | accepted | access | accessible | accessing | accommodate | accommodations | accomplish | accomplishments | according |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 1 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 2 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 3 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 4 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |

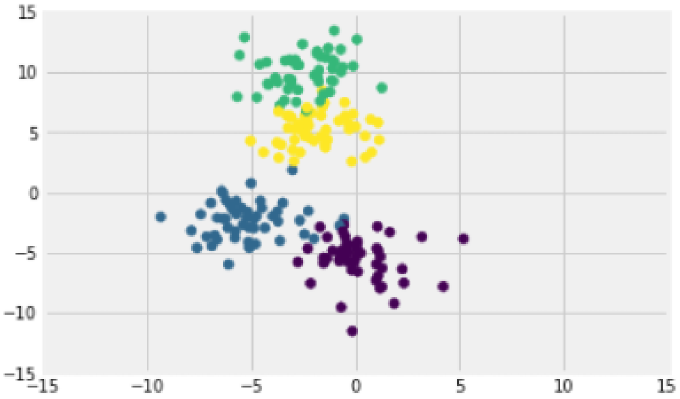


```
# initial_centroids = np.random.permutation(tf_idf_array.shape[0])[:3]
# initial_centroids
# centroids = tf_idf_array[initial_centroids]
# centroids.shape
# dist_to_centroid =  pairwise_distances(tf_idf_array,centroids, metric = 'euclidean')
# cluster_labels = np.argmin(dist_to_centroid, axis = 1)
```

```
class Kmeans:

    def __init__(self, k, seed = None, max_iter = 200):
        self.k = k
        self.seed = seed
        if self.seed is not None:
            np.random.seed(self.seed)
        self.max_iter = max_iter



    def initialise_centroids(self, data):


        initial_centroids = np.random.permutation(data.shape[0])[:self.
k]

        self.centroids = data[initial_centroids]

        return self.centroids


    def assign_clusters(self, data):


        if data.ndim == 1:
            data = data.reshape(-1, 1)

        dist_to_centroid =  pairwise_distances(data, self.centroids, me
tric = 'euclidean')
        self.cluster_labels = np.argmin(dist_to_centroid, axis = 1)
        return  self.cluster_labels


    def update_centroids(self, data):

        self.centroids = np.array([data[self.cluster_labels == i].mean(
axis = 0) for i in range(self.k)])

        return self.centroids


    def convergence_calculation(self):
        pass

    def predict(self, data):


        return self.assign_clusters(data)

    def fit_kmeans(self, data):
        """
        This function contains the main loop to fit the algorithm
        Implements initialise centroids and update_centroids
        according to max_iter

        self.centroids = self.initialise_centroids(data)

        # Main kmeans loop
        for iter in range(self.max_iter):

            self.cluster_labels = self.assign_clusters(data)
            self.centroids = self.update_centroids(data)
            if iter % 100 == 0:
                print("Running Model Iteration %d " %iter)
        print("Model finished running")
        return self
```

```
from sklearn.datasets import make_blobs
data = make_blobs(n_samples=200, n_features=2, centers=4, cluster_std=1.6, random_state=50)
points = data[0]
plt.scatter(data[0][:,0], data[0][:,1], c=data[1], cmap='viridis')
plt.xlim(-15,15)
plt.ylim(-15,15)

X = data[0]
X[2]
```

```
array([-2.30970265,  5.8496944 ])
```

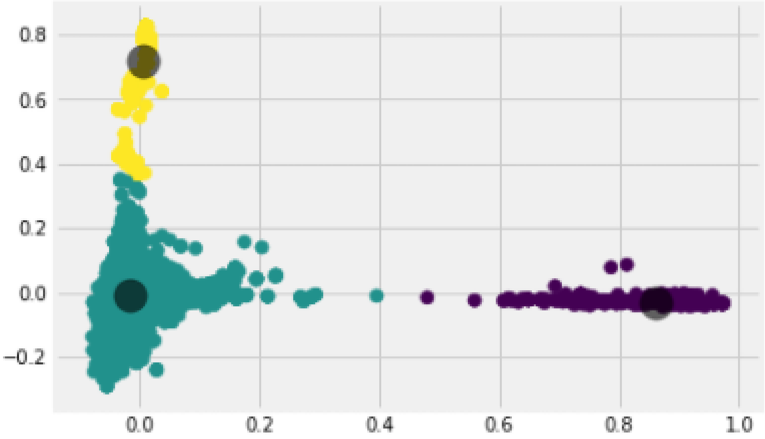


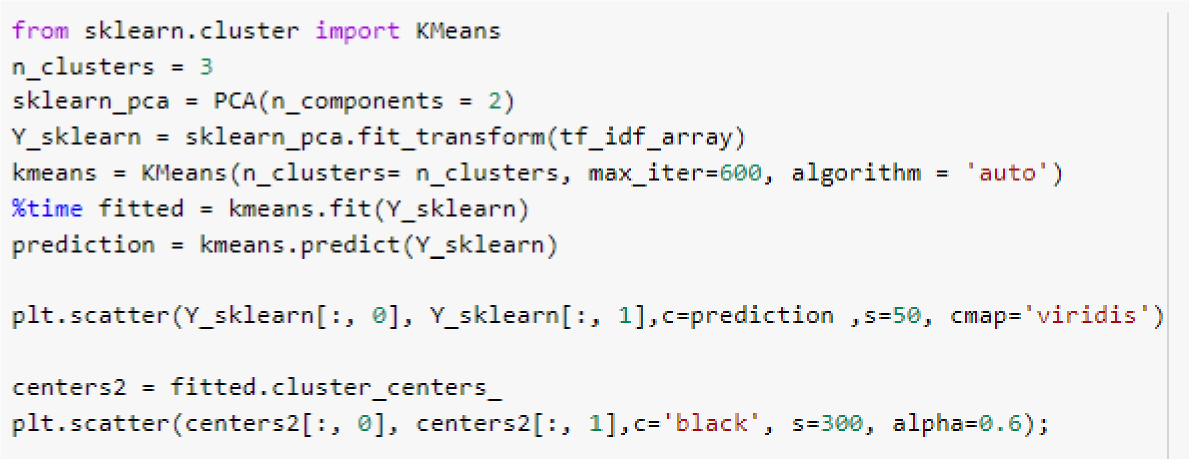
```
temp_k  = Kmeans(4, 1, 600)
temp_fitted  = temp_k.fit_kmeans(X)
new_data = np.array([[1.066, -8.66],
                     [1.87876, -6.516],
                     [-1.59728965,  8.45369045],
                     [1.87876, -6.516]])
temp_fitted.predict(new_data)
```

```
Running Model Iteration 0
Running Model Iteration 100
Running Model Iteration 200
Running Model Iteration 300
Running Model Iteration 400
Running Model Iteration 500
Model finished running
array([2, 2, 1, 2])
```

```
sklearn_pca = PCA(n_components = 2)
Y_sklearn = sklearn_pca.fit_transform(tf_idf_array)
test_e = Kmeans(3, 1, 600)
%time fitted = test_e.fit_kmeans(Y_sklearn)
predicted_values = test_e.predict(Y_sklearn)

plt.scatter(Y_sklearn[:, 0], Y_sklearn[:, 1], c=predicted_values, s=50, cmap='viridis')

centers = fitted.centroids
plt.scatter(centers[:, 0], centers[:, 1],c='black', s=300, alpha=0.6);
```

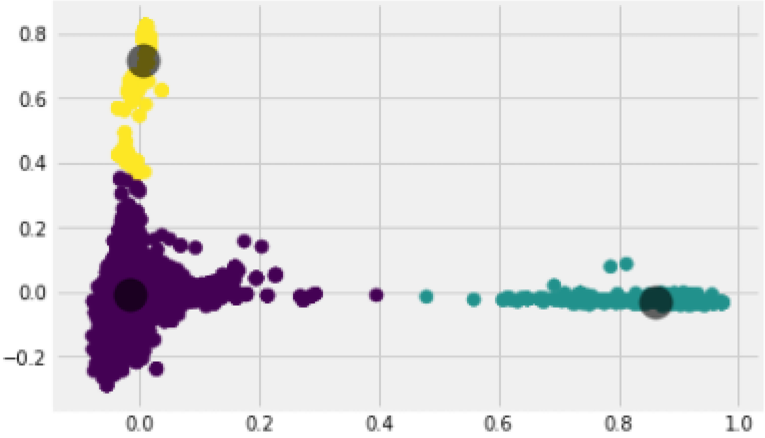
```
Running Model Iteration 0
Running Model Iteration 100
Running Model Iteration 200
Running Model Iteration 300
Running Model Iteration 400
Running Model Iteration 500
Model finished running
CPU times: user 2.41 s, sys: 4 ms, total: 2.42 s
Wall time: 2.41 s
```

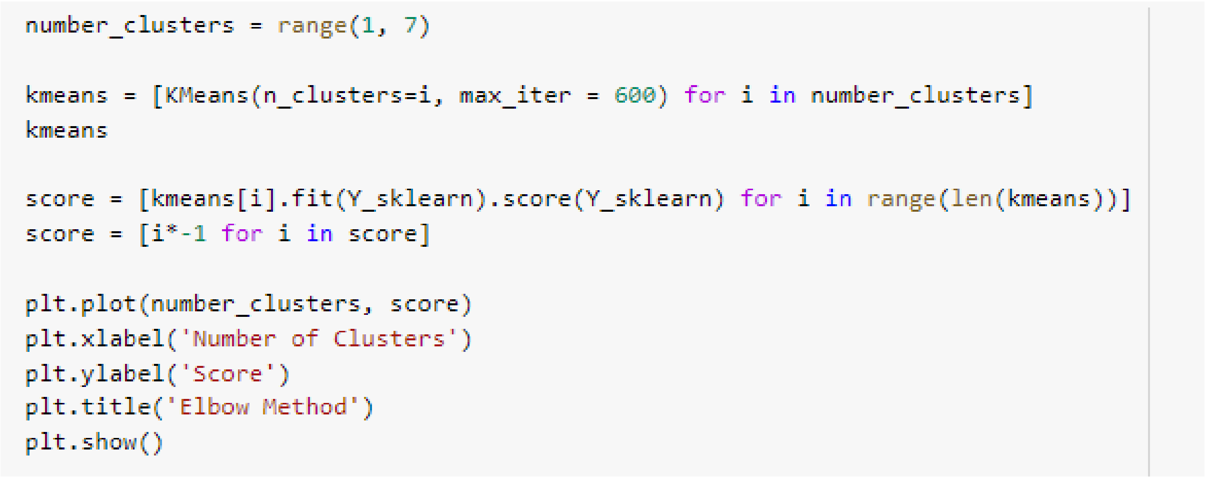




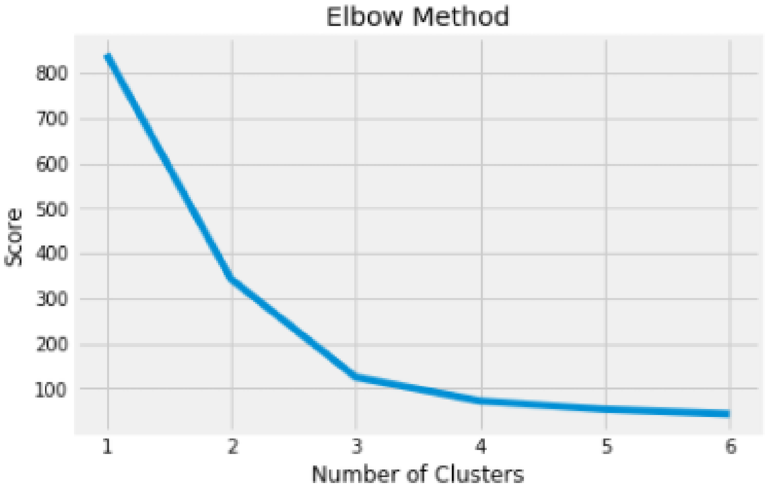
```
from sklearn.cluster import KMeans
n_clusters = 3
sklearn_pca = PCA(n_components = 2)
Y_sklearn = sklearn_pca.fit_transform(tf_idf_array)
kmeans = KMeans(n_clusters= n_clusters, max_iter=600, algorithm = 'auto')
%time fitted = kmeans.fit(Y_sklearn)
prediction = kmeans.predict(Y_sklearn)

plt.scatter(Y_sklearn[:, 0], Y_sklearn[:, 1],c=prediction ,s=50, cmap='viridis')

centers2 = fitted.cluster_centers_
plt.scatter(centers2[:, 0], centers2[:, 1],c='black', s=300, alpha=0.6);
```

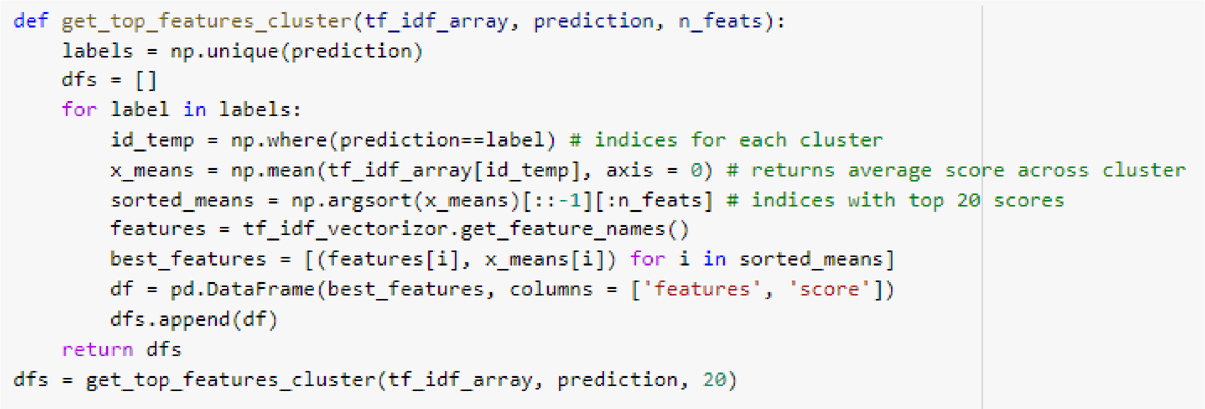
```
CPU times: user 192 ms, sys: 320 ms, total: 512 ms
Wall time: 132 ms
```



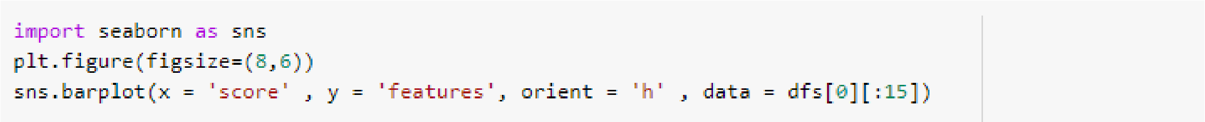


```
number_clusters = range(1, 7)

kmeans = [KMeans(n_clusters=i, max_iter = 600) for i in number_clusters]
kmeans

score = [kmeans[i].fit(Y_sklearn).score(Y_sklearn) for i in range(len(kmeans))]
score = [i*-1 for i in score]

plt.plot(number_clusters, score)
plt.xlabel('Number of Clusters')
plt.ylabel('Score')
plt.title('Elbow Method')
plt.show()
```



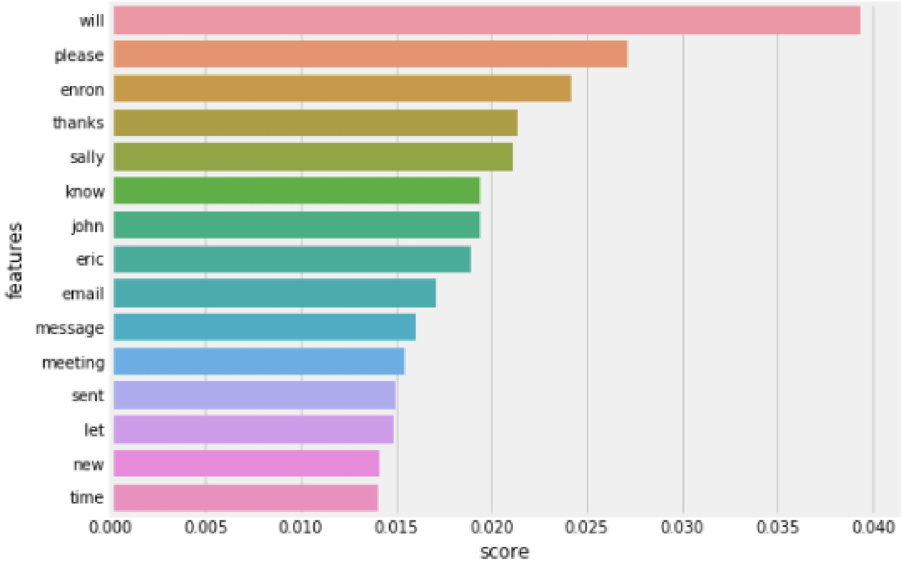


```
def get_top_features_cluster(tf_idf_array, prediction, n_feats):
    labels = np.unique(prediction)
    dfs = []
    for label in labels:
        id_temp = np.where(prediction==label) # indices for each cluster
        x_means = np.mean(tf_idf_array[id_temp], axis = 0) # returns average score across cluster
        sorted_means = np.argsort(x_means)[::-1][:n_feats] # indices with top 20 scores
        features = tf_idf_vectorizor.get_feature_names()
        best_features = [(features[i], x_means[i]) for i in sorted_means]
        df = pd.DataFrame(best_features, columns = ['features', 'score'])
        dfs.append(df)
    return dfs
dfs = get_top_features_cluster(tf_idf_array, prediction, 20)
```



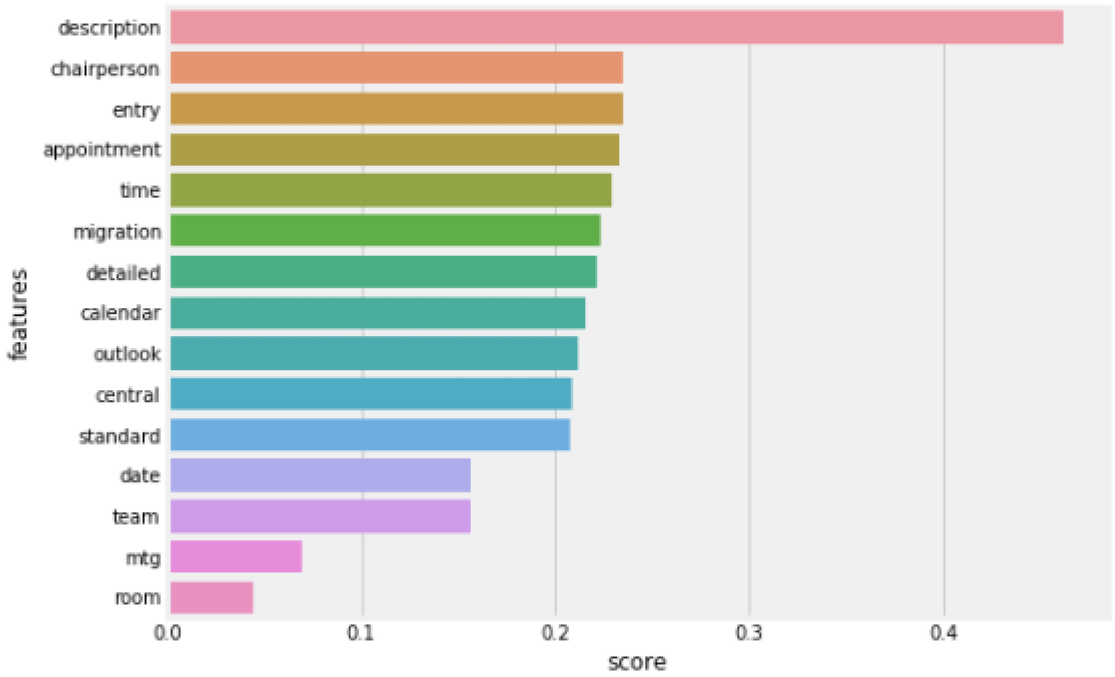
```
import seaborn as sns
plt.figure(figsize=(8,6))
sns.barplot(x = 'score' , y = 'features', orient = 'h' , data = dfs[0][:15])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f2462f19a20>
```



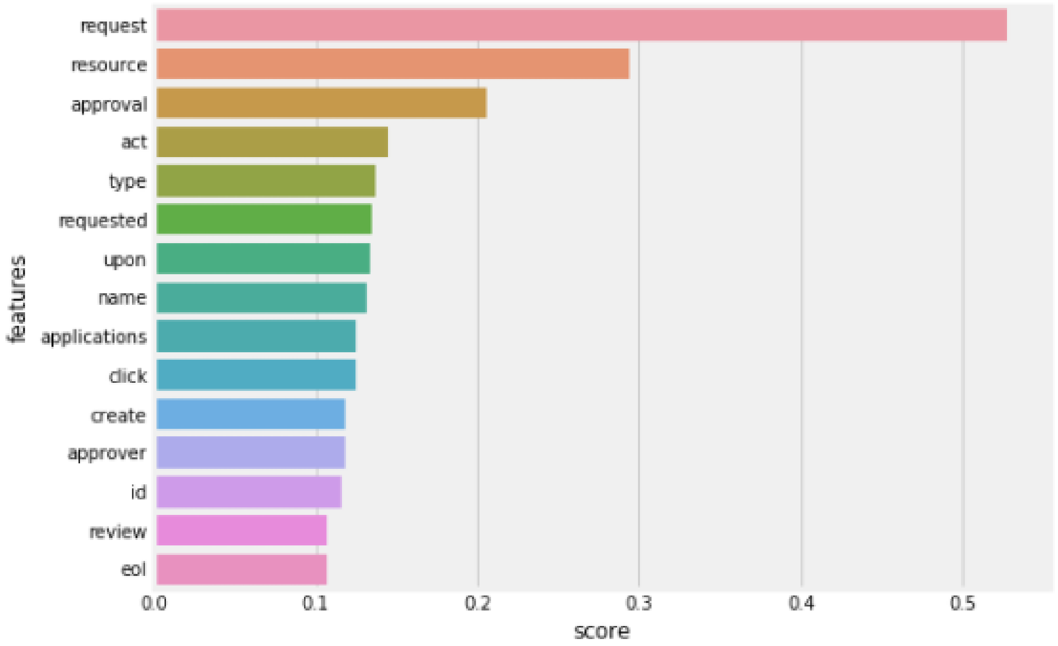
```
plt.figure(figsize=(8,6))
sns.barplot(x = 'score' , y = 'features', orient = 'h' , data = dfs[1][:15])
```

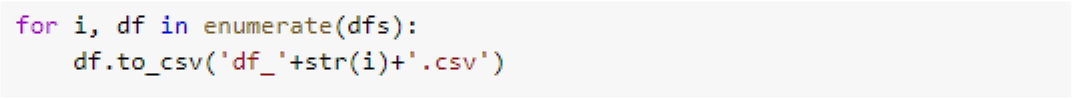
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f2462f49eb8>
```



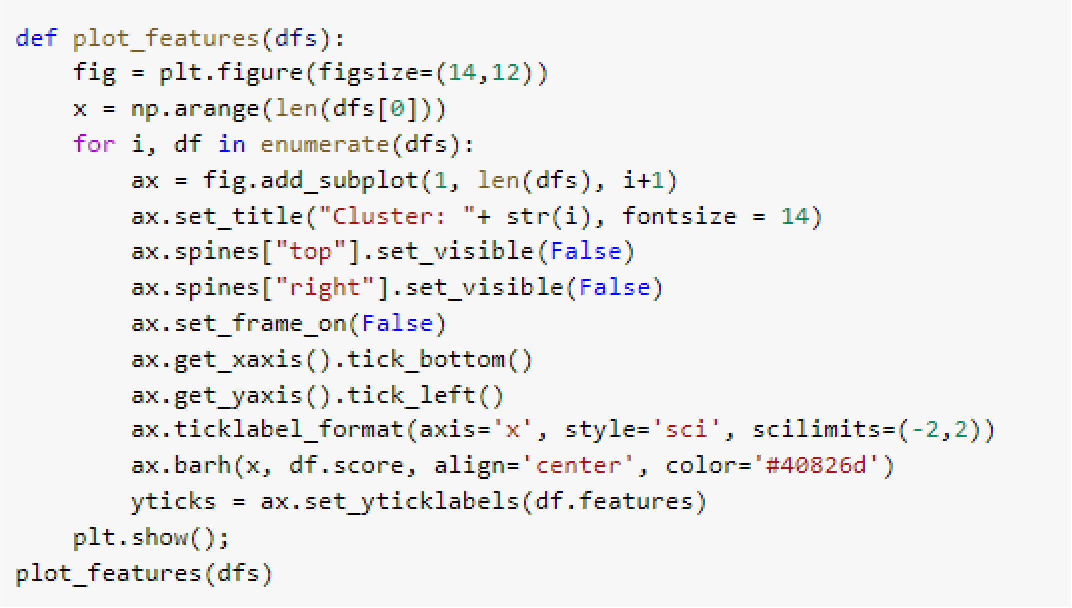
```
plt.figure(figsize=(8,6))
sns.barplot(x = 'score' , y = 'features', orient = 'h' , data = dfs[2][:15])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f2468074dd8>
```

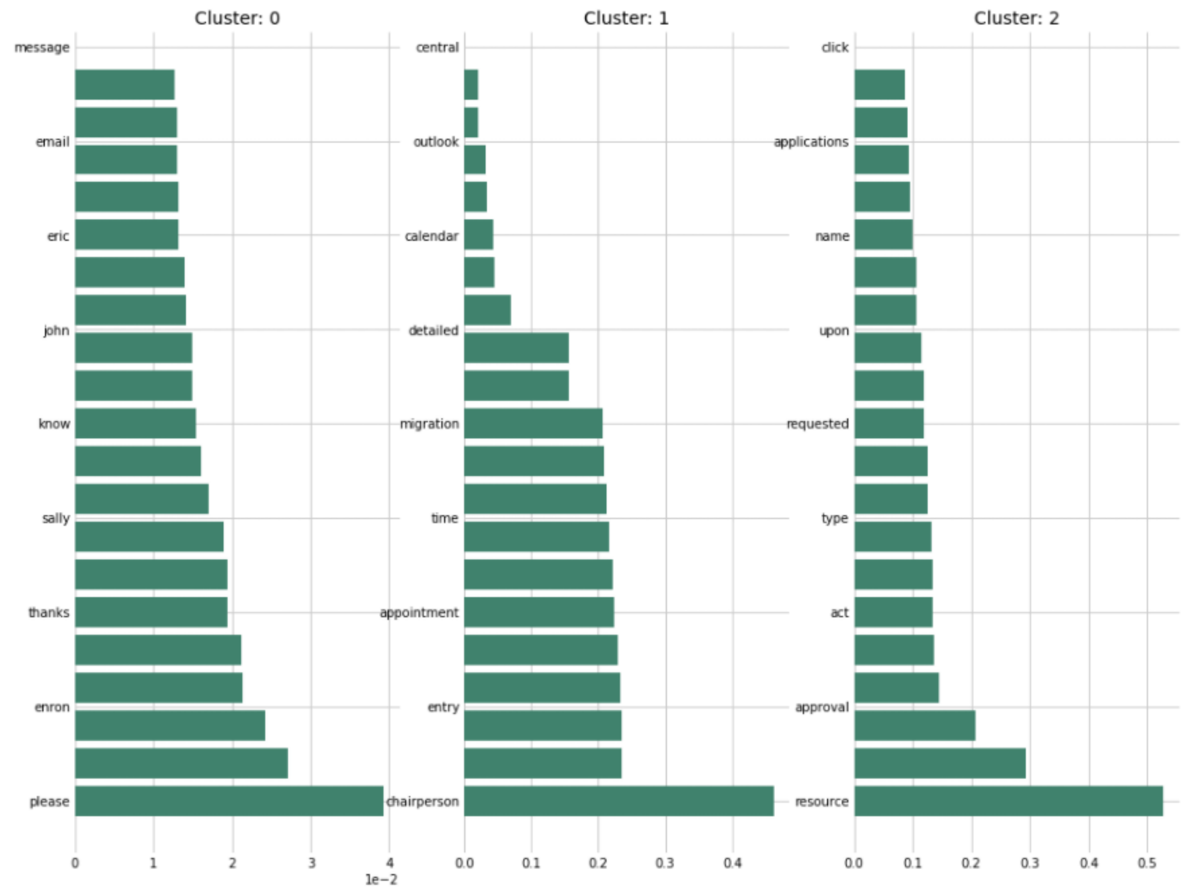




```python
for i, df in enumerate(dfs):
    df.to_csv('df_'+str(i)+'.csv')
```



```python
def plot_features(dfs):
    fig = plt.figure(figsize=(14,12))
    x = np.arange(len(dfs[0]))
    for i, df in enumerate(dfs):
        ax = fig.add_subplot(1, len(dfs), i+1)
        ax.set_title("Cluster: "+ str(i), fontsize = 14)
        ax.spines["top"].set_visible(False)
        ax.spines["right"].set_visible(False)
        ax.set_frame_on(False)
        ax.get_xaxis().tick_bottom()
        ax.get_yaxis().tick_left()
        ax.ticklabel_format(axis='x', style='sci', scilimits=(-2,2))
        ax.barh(x, df.score, align='center', color='#40826d')
        yticks = ax.set_yticklabels(df.features)
    plt.show();
plot_features(dfs)
```



**CONCLUSION:**

**K-Means Clustering for email txt corpora has been successfully implemented and executed.**