Importing libraries

In [1]:

```
!pip install pyldavis
!pip install --ignore-installed spark-nlp==2.3.1
!pip install bokeh
!pip install nltk
!pip install pandas
import nltk
nltk.download('stopwords')

nltk.download('wordnet')

nltk.download('averaged_perceptron_tagger')

nltk.download('vader_lexicon')
!pip install geopy



from pyspark.sql.functions import lit, when, col, regexp_extract,lower
from geopy.geocoders import Nominatim
import geopy.geocoders
import pandas as pd

from pyspark import SparkConf, SparkContext, SQLContext
from pyspark.sql import SparkSession, DataFrame
from pyspark.sql.functions import *
from pyspark.sql.types import *
import os
# from functools import reduce
import json
import time

from pyspark.sql import SQLContext, Row
from pyspark.sql import SparkSession
from pyspark.sql import Row
from pyspark.sql import HiveContext, Row
from pyspark.sql import SQLContext
import pyspark.sql.functions as F
from pyspark.sql.functions import col,regexp_replace

import re;
from nltk.corpus import stopwords
import matplotlib.pyplot as plt
```

```python
import pandas as pd
import string
import pyspark.sql.functions as f
from nltk.stem import WordNetLemmatizer
from pyspark.ml.feature import CountVectorizer,StringIndexer, RegexTokenizer,StopWordsRemover
from nltk.sentiment.vader import SentimentIntensityAnalyzer

import warnings
warnings.filterwarnings("ignore", category=DeprecationWarning)
warnings.filterwarnings("ignore", category=FutureWarning)
warnings.filterwarnings("ignore", category=RuntimeWarning)
##LDA specific imports and installs


from pyspark.ml.feature import CountVectorizer , IDF
from pyspark.ml.clustering import LDA
from pyspark.sql.functions import explode,size

import pyLDAvis
import numpy as np
```

```
Requirement already satisfied: pyldavis in /Library/anaconda3/lib/python3.7/site-packages (2.1.2)
Requirement already satisfied: joblib>=0.8.4 in /Library/anaconda3/lib/python3.7/site-packages (from pyl
Requirement already satisfied: numexpr in /Library/anaconda3/lib/python3.7/site-packages (from pyldavis)
Requirement already satisfied: numpy>=1.9.2 in /Library/anaconda3/lib/python3.7/site-packages (from pyld
Requirement already satisfied: pandas>=0.17.0 in /Library/anaconda3/lib/python3.7/site-packages (from py
Requirement already satisfied: funcy in /Library/anaconda3/lib/python3.7/site-packages (from pyldavis) (
Requirement already satisfied: scipy>=0.18.0 in /Library/anaconda3/lib/python3.7/site-packages (from pyl
Requirement already satisfied: jinja2>=2.7.2 in /Library/anaconda3/lib/python3.7/site-packages (from pyl
Requirement already satisfied: wheel>=0.23.0 in /Library/anaconda3/lib/python3.7/site-packages (from pyl
Requirement already satisfied: pytest in /Library/anaconda3/lib/python3.7/site-packages (from pyldavis)
Requirement already satisfied: future in /Library/anaconda3/lib/python3.7/site-packages (from pyldavis)
Requirement already satisfied: pytz>=2017.2 in /Library/anaconda3/lib/python3.7/site-packages (from pand
Requirement already satisfied: python-dateutil>=2.6.1 in /Library/anaconda3/lib/python3.7/site-packages
Requirement already satisfied: MarkupSafe>=0.23 in /Library/anaconda3/lib/python3.7/site-packages (from
Requirement already satisfied: py>=1.5.0 in /Library/anaconda3/lib/python3.7/site-packages (from pytest-
Requirement already satisfied: packaging in /Library/anaconda3/lib/python3.7/site-packages (from pytest-
Requirement already satisfied: attrs>=17.4.0 in /Library/anaconda3/lib/python3.7/site-packages (from pyt
Requirement already satisfied: more-itertools>=4.0.0 in /Library/anaconda3/lib/python3.7/site-packages (
Requirement already satisfied: atomicwrites>=1.0 in /Library/anaconda3/lib/python3.7/site-packages (from
Requirement already satisfied: pluggy<1.0,>=0.12 in /Library/anaconda3/lib/python3.7/site-packages (from
Requirement already satisfied: wcwidth in /Library/anaconda3/lib/python3.7/site-packages (from pytest->p
Requirement already satisfied: importlib-metadata>=0.12 in /Library/anaconda3/lib/python3.7/site-package
Requirement already satisfied: six>=1.5 in /Library/anaconda3/lib/python3.7/site-packages (from python-da
Requirement already satisfied: pyparsing>=2.0.2 in /Library/anaconda3/lib/python3.7/site-packages (from p
Requirement already satisfied: zipp>=0.5 in /Library/anaconda3/lib/python3.7/site-packages (from importli
Collecting spark-nlp==2.3.1
  Using cached https://files.pythonhosted.org/packages/c2/e0/c036825e5e5f272b51835a1cf1ed9c871dd520009ca
 (https://files.pythonhosted.org/packages/c2/e0/c036825e5e5f272b51835a1cf1ed9c871dd520009ca64039c3a401c1
Installing collected packages: spark-nlp
Successfully installed spark-nlp-2.3.1
Requirement already satisfied: bokeh in /Library/anaconda3/lib/python3.7/site-packages (1.3.4)
Requirement already satisfied: six>=1.5.2 in /Library/anaconda3/lib/python3.7/site-packages (from bokeh)
Requirement already satisfied: packaging>=16.8 in /Library/anaconda3/lib/python3.7/site-packages (from bo
Requirement already satisfied: PyYAML>=3.10 in /Library/anaconda3/lib/python3.7/site-packages (from bokeh
Requirement already satisfied: numpy>=1.7.1 in /Library/anaconda3/lib/python3.7/site-packages (from bokeh
Requirement already satisfied: pillow>=4.0 in /Library/anaconda3/lib/python3.7/site-packages (from bokeh
Requirement already satisfied: tornado>=4.3 in /Library/anaconda3/lib/python3.7/site-packages (from bokeh
Requirement already satisfied: Jinja2>=2.7 in /Library/anaconda3/lib/python3.7/site-packages (from bokeh
Requirement already satisfied: python-dateutil>=2.1 in /Library/anaconda3/lib/python3.7/site-packages (fi
Requirement already satisfied: pyparsing>=2.0.2 in /Library/anaconda3/lib/python3.7/site-packages (from p
Requirement already satisfied: MarkupSafe>=0.23 in /Library/anaconda3/lib/python3.7/site-packages (from
Requirement already satisfied: nltk in /Library/anaconda3/lib/python3.7/site-packages (3.4.5)
Requirement already satisfied: six in /Library/anaconda3/lib/python3.7/site-packages (from nltk) (1.13.0
Requirement already satisfied: pandas in /Library/anaconda3/lib/python3.7/site-packages (0.25.2)
Requirement already satisfied: pytz>=2017.2 in /Library/anaconda3/lib/python3.7/site-packages (from panda
Requirement already satisfied: numpy>=1.13.3 in /Library/anaconda3/lib/python3.7/site-packages (from pand
Requirement already satisfied: python-dateutil>=2.6.1 in /Library/anaconda3/lib/python3.7/site-packages
Requirement already satisfied: six>=1.5 in /Library/anaconda3/lib/python3.7/site-packages (from python-da

[nltk_data] Downloading package stopwords to
[nltk_data]     /Users/umarajpotla/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to
[nltk_data]     /Users/umarajpotla/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /Users/umarajpotla/nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
[nltk_data] Downloading package vader_lexicon to
[nltk_data]     /Users/umarajpotla/nltk_data...
[nltk_data]   Package vader_lexicon is already up-to-date!

Requirement already satisfied: geopy in /Library/anaconda3/lib/python3.7/site-packages (1.20.0)
Requirement already satisfied: geographiclib<2,>=1.49 in /Library/anaconda3/lib/python3.7/site-packages
```

```
/Library/anaconda3/lib/python3.7/site-packages/past/builtins/misc.py:4: DeprecationWarning: Using or imp
lections.abc' is deprecated, and in 3.8 it will stop working
  from collections import Mapping
```

Spark Session

In [2]:

```
spark = SparkSession.builder.master("local[*]").appName("BDP_Proj").getOrCreate()
sc = spark.sparkContext
sqlContext = SQLContext(sc)

dataf1 = spark.read.option("header",True).option("escape","\"").csv("SparkAllRetweets.csv")
dataf2 = spark.read.option("header",True).option("escape","\"").csv("SparkAllTweets.csv")

dataf=(dataf1.drop('retweeted_id')).union(dataf2)
dataf
```

Out[2]:

```
DataFrame[_c0: string, created_at: string, id: string, text: string, source: string, user: string, geo:
nt: string, favorite_count: string, entities: string, lang: string]
```

Function Definitions

In [3]:

```python
import nltk;
import string

def null_value_count(df):
  null_columns_counts = []
  numRows = df.count()
  for k in df.columns:
    nullRows = df.where(col(k).isNull()).count()
    if(nullRows > 0):
      temp = k,nullRows
      null_columns_counts.append(temp)
  return(null_columns_counts)


def remove_stopwords(x):
    from nltk.corpus import stopwords
    stop_words=set(stopwords.words('english'))
    stop_words.add("rt")
    filtered_sentence = [w for w in x if not w in stop_words]
    return filtered_sentence



def remove_punctuations(x):
    list_punct=list(string.punctuation)
    filtered = [''.join(c for c in s if c not in list_punct) for s in x]
    filtered_space = [s for s in filtered if s] #remove empty space
    return filtered_space

def lemmatization(x):
    lemmatizer_model = WordNetLemmatizer()
    final_Lem = [lemmatizer_model.lemmatize(s) for s in x]
    return final_Lem

def join_tokens(x):
    joinedTokens_list = []
    x = " ".join(x)
    return x


def sentiment_words(x):

    #making a model
    sentiment_analyzer = SentimentIntensityAnalyzer()

    #Analysing the polarity scores
    sentiment_list_temp = []
    for i in x[:-1]:
        temp_list = ''.join(i)
        polarity_score = sentiment_analyzer.polarity_scores(temp_list)
        sentiment_list_temp.append((temp_list, polarity_score))
        sentiment_list_temp = [w for w in sentiment_list_temp if w]

    #Assignment of the polarity score value
    sentiment_list  = []
    for i in sentiment_list_temp:
        text = i[0]
        second = i[1]
        total_neg=[]
        total_pos=[]
        for (norm_Score, v) in second.items():
            if norm_Score == 'compound':
                if v < 0.0:
                    sentiment_list.append((text,"Negative",v*100,x[-1:]))
                    total_neg.append(v*100)

                elif v == 0.0:
                    sentiment_list.append((text,"Neutral",v*100,x[-1:]))

                else:
                    sentiment_list.append((text,"Positive",v*100,x[-1:]))
                    total_pos.append(v*100)


    return sentiment_list


#Extraction of phrases for performing Sentiment Analysis
def extract_phrases(x):
    stop_words=set(stopwords.words('english'))
    stop_words.add("rt")

    #making tokens from words
    sentence_regex = r'(?:(?:[A-Z])(?:.[A-Z])+.?)|(?:\w+(?:-\w+)*)|(?:\$?\d+(?:.\d+)?%?)|(?:...|)(?:
    tokens = nltk.regexp_tokenize(x[0],sentence_regex)
```

```python

    #Tagging parts of speech to every token generated
    pos_tokens = nltk.tag.pos_tag(tokens)

    #performing the chunks using nouns and adjectives
    grammar = r"""
    NP_GRAMMAR:
        {<NN.*|JJ>*<NN.*>}
        {<NN.*|JJ>*<NN.*><IN><NN.*|JJ>*<NN.*>}
    """
    chunker = nltk.RegexpParser(grammar)

    #Making a chunk tree from Parts of Speech tokens
    chunk_tree = chunker.parse(pos_tokens)

    #Finding Noun Phrases (GRAMMAR) from leaf nodes of a chunk tree
    def tree_leaves(tree):
        for subtree in tree.subtrees(filter = lambda t: t.label()=='NP_GRAMMAR'):
            yield subtree.leaves()

    #getting a leaf from the chunk tree
    def get_terms(tree):
        for leaf in tree_leaves(tree):
            term = [w for w,t in leaf if not w in stop_words]
            yield term

    terms = get_terms(chunk_tree)


    #making pharses out of the terms
    temp_phrases = []
    for term in terms:
        if len(term):
            temp_phrases.append(' '.join(term))
    temp_phrases.append(x[1])

    #remove empty lines
    final_Phrase = [w for w in temp_phrases if w]

    return final_Phrase

#to remove null which were coming after the extraction
def blank_as_null(x):
    return when(col(x) != "", col(x)).otherwise(None)


```

Why PokemonGo ? Comaprison with other applications

Why we choose Pokemon Go and how it is one of the major events in 2016.

We have to analyze the popularity of pokemon GO. First, we will be doing data preprocessing using data mining techniques. Then by using tl
find the relations of the game with some other events that we had in 2016. Data segregation can be done on the basis of hashtags present ir
about events in 2016 on Twitter.

In [4]:

```
#removing the null and then transforming Spark data frame to rdd
textOnlydf=dataf.select("text").dropna()
textOnlyRdd = textOnlydf.rdd.map(str)

#filter every thing which is in english
filteredtextdf=textOnlydf.withColumn('FilteredText', F.regexp_replace('text',r'([^A-Za-z \t])|(\w+:\
textOnlyRdd = filteredtextdf.select("FilteredText").rdd.map(str)

#to split and count get the word count
counts = textOnlyRdd.flatMap(lambda line: line.lower().split())\
    .map(lambda word: (word, 1))\
    .reduceByKey(lambda x, y: x + y)


#Converting the word count back to Spark data fram
distinctwordsdf=counts.toDF()
distinctwordsdf=distinctwordsdf.withColumnRenamed('_1', 'word')
distinctwordsdf=distinctwordsdf.withColumnRenamed('_2', 'count')

#adding famous apps from wikipedia and filterign the dataframe

AppList =['pokémongo','pokemon','pokecoins','poké',
          'poke','pokmon','pokémonsunandmoon','pokemonsunandmoon','twitter','facebook','tinder','sna

finalPlotdf=distinctwordsdf.where(col("word").\
isin(AppList)).withColumn('word', regexp_replace(col('word') , r'pok[A-Za-z]*', 'PokemonGO' ))\
    .groupBy('word').sum().sort('sum(count)', ascending=False)



GameList =['pokémongo','pokemon','pokecoins','poké',
          'poke','pokmon','pokémonsunandmoon','pokemonsunandmoon','clash','clashland','reigns','rene

finalPlotdf1=distinctwordsdf.where(col("word").\
isin(GameList)).withColumn('word', regexp_replace(col('word') , r'pok[A-Za-z]*', 'PokemonGO' ))\
    .groupBy('word').sum().sort('sum(count)', ascending=False)

finalPlotdf2=finalPlotdf1.withColumn('word', regexp_replace(col('word') , r'clash[A-Za-z]*', 'ClashR
    .groupBy('word').sum().sort('sum(sum(count))', ascending=False)

#renaming the last column
finalPlotdf1=finalPlotdf1.withColumnRenamed('sum(count)', 'count')
```

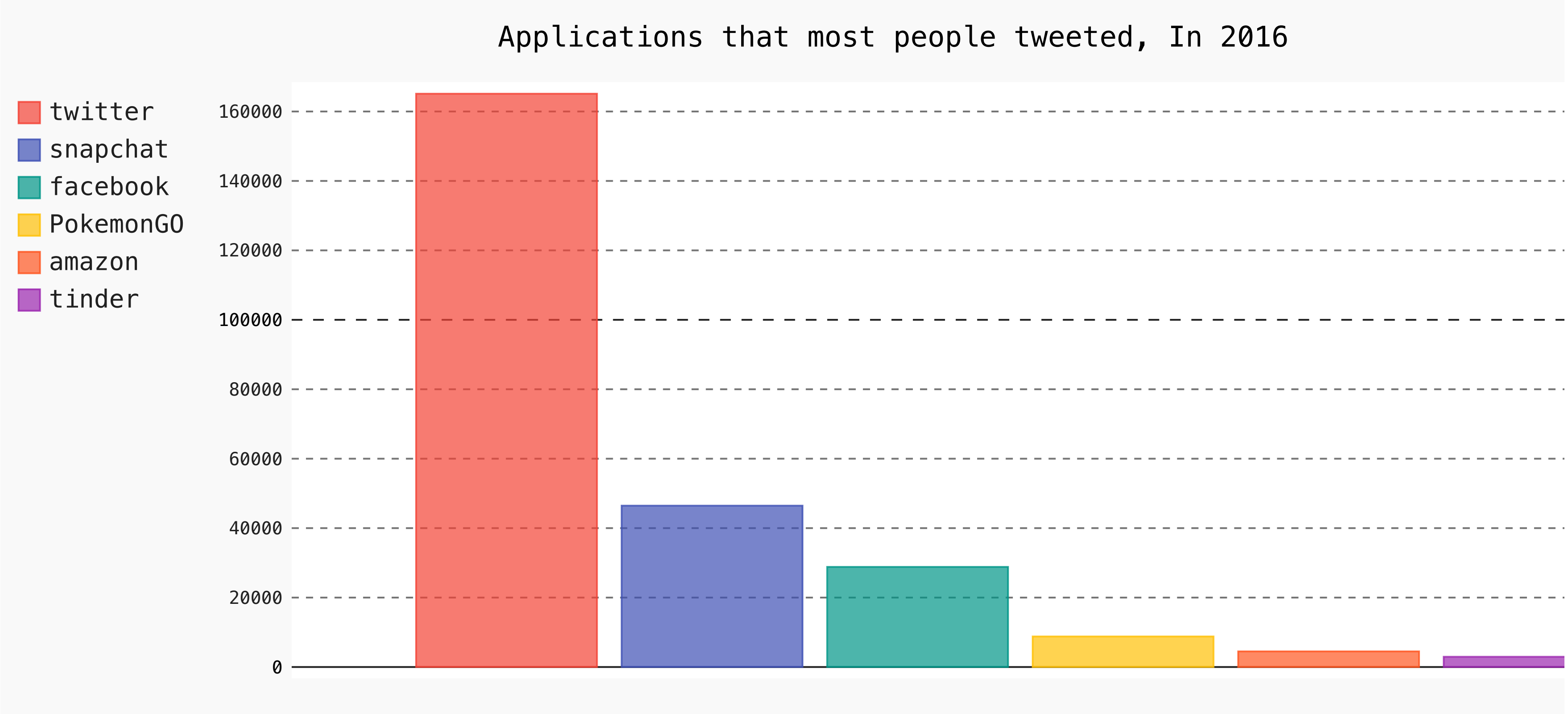
```
finalPlotdf2=finalPlotdf2.withColumnRenamed('sum(sum(count))','count')

finalPlotdf2.toPandas().to_csv("mobile_games.csv",header=True)
finalPlotdf.toPandas().to_csv("All_Apps.csv",header=True)
```

We are using pygal an external library to plot the data.
It brings it intuitive nature using predefined JS

In [5]:

```
finalPlotPandf=finalPlotdf.toPandas()
import pygal
from pygal.style import Style
import pygal
from ipywidgets import HTML
from IPython.display import HTML
import base64

custom_style = Style(
  olors=('#E853A0', '#E8537A', '#E95355', '#E87653', '#E89B53'))

b_chart = pygal.Bar(style=custom_style,width=1000, height=400, explicit_size=True)
b_chart.title = "Applications that most people tweeted, In 2016"

for bar in range(len(finalPlotPandf)):
    b_chart.add(finalPlotPandf['word'].iloc[bar],finalPlotPandf['sum(count)'].iloc[bar])

%matplotlib inline
from IPython.display import SVG, HTML

html_pygal = u"""
    <!DOCTYPE html>
    <html>
        <head>
             <script type="text/javascript" src="http://kozea.github.com/pygal.js/javascripts/svg.jq
              <script type="text/javascript" src="https://kozea.github.io/pygal.js/2.0.x/pygal-toolt
        </head>
        <body><figure>{pygal_render}</figure></body>
    </html>
"""
HTML(html_pygal.format(pygal_render=b_chart.render(is_unicode=True)))
```

Out[5]:

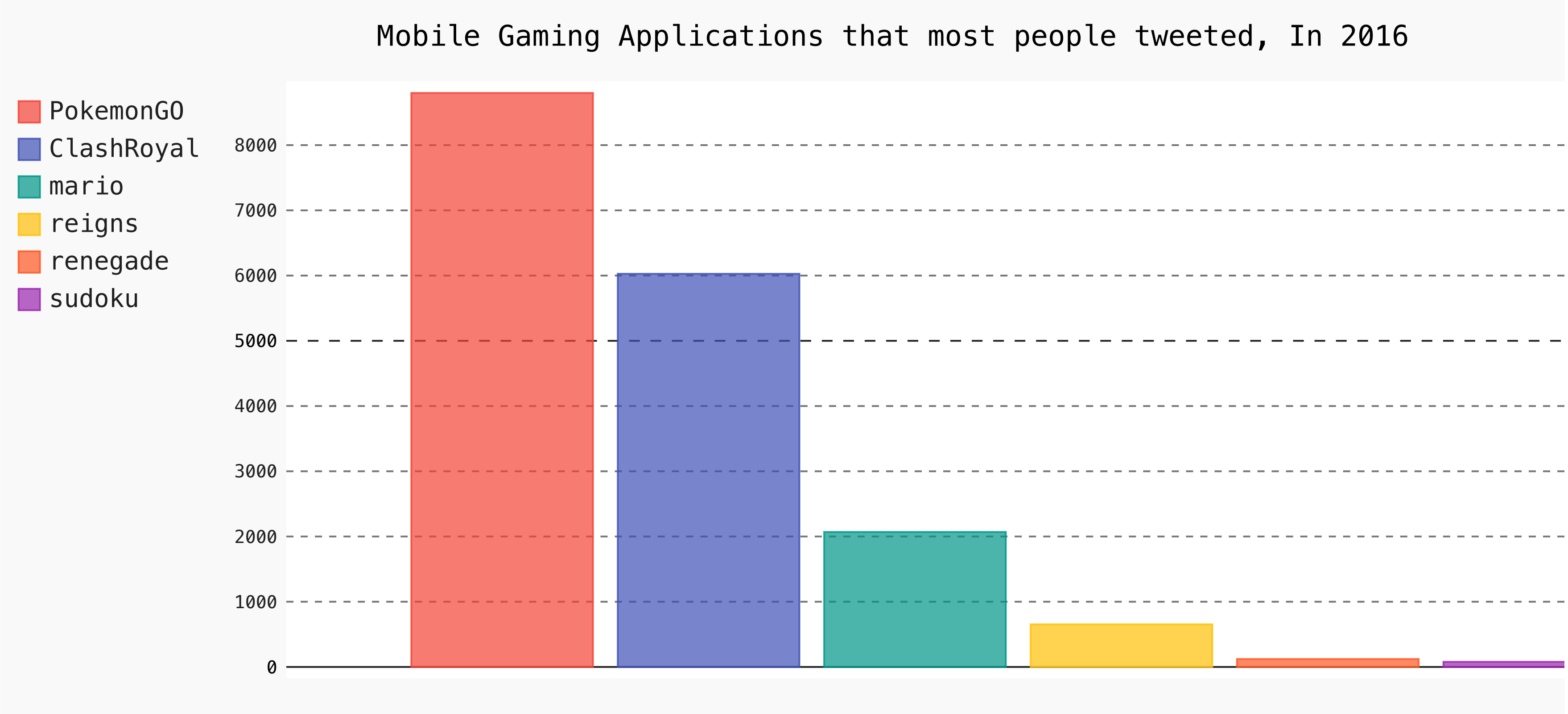


We are using pygal an external library to plot the data.
It brings it intuitive nature using predefined JS

In [6]:

```
finalPlotdf2Pandf=finalPlotdf2.toPandas()
import pygal
from pygal.style import Style
import pygal
from ipywidgets import HTML
from IPython.display import HTML
import base64

custom_style = Style(
  olors=('#E853A0', '#E8537A', '#E95355', '#E87653', '#E89B53'))

b_chart = pygal.Bar(style=custom_style,width=1000, height=400, explicit_size=True)
b_chart.title = "Mobile Gaming Applications that most people tweeted, In 2016"

for bar in range(len(finalPlotdf2Pandf)):
    b_chart.add(finalPlotdf2Pandf['word'].iloc[bar],finalPlotdf2Pandf['count'].iloc[bar])

%matplotlib inline
from IPython.display import SVG, HTML


html_pygal = u"""
    <!DOCTYPE html>
    <html>
        <head>
             <script type="text/javascript" src="http://kozea.github.com/pygal.js/javascripts/svg.jq
              <script type="text/javascript" src="https://kozea.github.io/pygal.js/2.0.x/pygal-toolt
        </head>
        <body><figure>{pygal_render}</figure></body>
    </html>
"""
HTML(html_pygal.format(pygal_render=b_chart.render(is_unicode=True)))
```

Out[6]:



Analyisng popularity of Pokemon GO across the world

The popularity of PokemonGo has been global; it is used by plenty of people spread across the world. Using this fact, we can visualize the tr data provides location information. From the location information, we can derive the various insights. For example, we can determine the pop use various graphical methods for representation of the same.

In [7]:

```

#to remove null which were coming after the extraction
def blank_as_null(x):
    return when(col(x) != "", col(x)).otherwise(None)

#get coordinates from
# dataf1 = spark.read.option("header",True).option("escape","\"").csv("translated_pokemon_tweets.csv
# dataf2 = spark.read.option("header",True).option("escape","\"").csv("translated_pokemon_retweets.c

# dataf=(dataf1.drop('retweeted_id')).union(dataf2)
coor= dataf.select("coordinates")

locationCoord=coor.filter("coordinates not like '0' and coordinates like 'Row(coordinates=%' and coo

#filtering coordinates and splitting latituded and longitudes and then storing them in finalGeoLocat
expr = r'\[(.*?)\]+'
# filter on the basis of the coordinate formates
geo_locations = locationCoord.filter(locationCoord["coordinates"].rlike(expr))

# keeping only the required data ---- we need to check this
new_df = geo_locations.select(regexp_extract('coordinates', r'\[(.*?)\]+', 1).alias('extracted'))

#drop null
dfWithEmptyReplaced = new_df.withColumn("extracted_coordinates", blank_as_null("extracted"))

#remove  na
finalGeoLocation=dfWithEmptyReplaced.na.drop()

finalGeoLocation=finalGeoLocation.toPandas()
finalGeoLocation[['long','lat']]=finalGeoLocation.extracted_coordinates.str.split(",",expand=True)

#Converting all the coordinates in to exact format of latitude and longitude

finalGeoLocation['long'] = finalGeoLocation['long'].apply(lambda x: x.replace("[", ""))
finalGeoLocation['long'] = finalGeoLocation['long'].apply(lambda x: float(x))
finalGeoLocation['lat'] = finalGeoLocation['lat'].apply(lambda x: float(x))
finalGeoLocation['long']=finalGeoLocation['long'].round(decimals=6)
finalGeoLocation['lat']=finalGeoLocation['lat'].round(decimals=6)

#Calling geopy and fetching countries from the coordinated
geopy.geocoders.options.default_timeout = 1000000
geolocator = Nominatim(user_agent="AnitGeoCode")
```

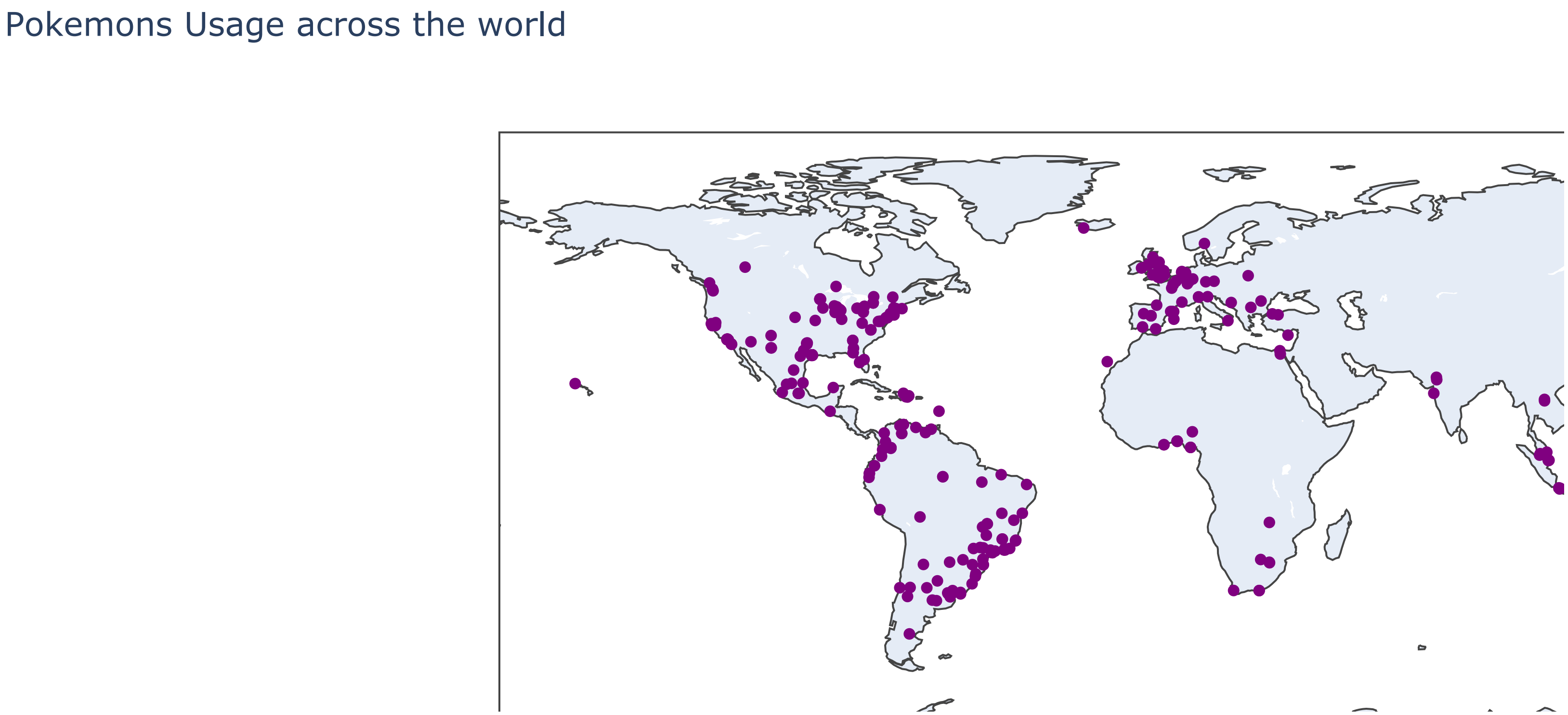
```python
coutrydata=[]
for i in range(len(finalGeoLocation)):
    cordinates=str(finalGeoLocation['lat'].iloc[i])+","+str(finalGeoLocation['long'].iloc[i])
    location = geolocator.reverse(cordinates)
    lat=finalGeoLocation['lat'].iloc[i]
    long=finalGeoLocation['long'].iloc[i]
    country=location.raw['address']['country']
    #making a dictionary to and append in the list in order to push them to a data frame
    coutrydata.append({"coordinates":cordinates,"country":country,"latitude":lat,"longitude":long})



countryDatadf=pd.DataFrame(coutrydata)


countryDatadf.to_csv('coordinates_final.csv')
countryDatadf.head()


```

Out[7]:

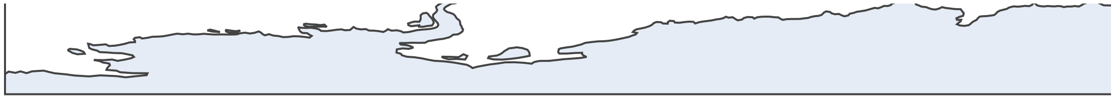
	coordinates	country	latitude	longitude
0	7.081874,125.505375	Philippines	7.081874	125.505375
1	23.7867,73.5231	India	23.786700	73.523100
2	37.782112,-122.400613	United States of America	37.782112	-122.400613
3	40.671743,-73.953282	United States of America	40.671743	-73.953282
4	40.926841,29.123982	Türkiye	40.926841	29.123982

In [8]:

```python
import plotly.graph_objects as go
import seaborn as sns
import plotly#.plotly as py
import plotly.graph_objs as go
plotly.offline.init_notebook_mode(connected=False)

fig = go.Figure(data=go.Scattergeo(lon = countryDatadf['longitude'],
                                   lat = countryDatadf['latitude'],
                                   text = countryDatadf['country'],
                                   mode = 'markers',
                                   marker_color ="purple"))

fig.update_layout(title = 'Pokemons Usage across the world<br>',
                  geo_scope='world')


#fig.show()
plotly.offline.iplot(fig)
```



Pokemons Usage across the world



Sentiment Analysis On tweets "text " fields and finding polarities

Pokemon Go game received both positive and negative feedback related to the implementation using AR in this field, its vivid use, and techr related to PokemonGO with their opinions towards the game. As part of sentiment analysis, we will be analyzing the user's take over the gam characters, people express their opinions precisely through hashtags. So based on those valuable hashtags, we would like to categorize the computing the overall polarity of the game, we will be considering the level of positivity and negativity of the tweets.

In [9]:

```
import re;
from pyspark.sql import functions as F

sentimentAnalysisText = dataf.select("text","created_at","user")
null_columns_count_list_sa = null_value_count(sentimentAnalysisText)
spark.createDataFrame(null_columns_count_list_sa, ['Column_With_Null_Value', 'Null_Values_Count'])#.


sentimentAnalysisText = sentimentAnalysisText.dropna()
sentimentAnalysisText = sentimentAnalysisText.withColumn("only_str",regexp_replace(col('text'), '\d+
sentimentAnalysisText = sentimentAnalysisText.withColumn('value', F.regexp_replace('text', '([^0-9A-
sentimentAnalysisText =  sentimentAnalysisText[sentimentAnalysisText['text'].contains("pok")]
dataf_sa = sentimentAnalysisText.select("value","created_at")
dataf_sa.show(2)
```

```
<>:10: DeprecationWarning:

invalid escape sequence \d

<>:11: DeprecationWarning:

invalid escape sequence \w

<>:10: DeprecationWarning:

invalid escape sequence \d

<>:11: DeprecationWarning:

invalid escape sequence \w

<>:10: DeprecationWarning:

invalid escape sequence \d

<>:11: DeprecationWarning:

invalid escape sequence \w

<ipython-input-9-792a8cf85cd8>:10: DeprecationWarning:

invalid escape sequence \d
```

```
<ipython-input-9-792a8cf85cd8>:11: DeprecationWarning:

invalid escape sequence \w


+--------------------+--------------------+
|               value|          created_at|
+--------------------+--------------------+
|RT OdinYT FolagoR...|Fri Oct 14 15:00:...|
|RT FeelzHurter Wh...|Fri Oct 14 15:00:...|
+--------------------+--------------------+
only showing top 2 rows
```

In [11]:

```
!pip install nltk
import nltk
from pyspark.ml.feature import RegexTokenizer
from pyspark.ml.feature import CountVectorizer,StringIndexer, RegexTokenizer,StopWordsRemover


regex_tokenizer = RegexTokenizer(inputCol="value", outputCol="words", pattern="\\W")
raw_words_pokemon = regex_tokenizer.transform(dataf_sa)

remover = StopWordsRemover(inputCol="words", outputCol="filtered")
pokemon_words_df = remover.transform(raw_words_pokemon)


pokemon_words = pokemon_words_df.select("filtered").rdd.flatMap(lambda x: x)
stopwordRDD_pokemon = pokemon_words.map(remove_stopwords)

rmvPunctRDD_pokemon = stopwordRDD_pokemon.map(remove_punctuations)

lem_wordsRDD_pokemon = rmvPunctRDD_pokemon.map(lemmatization)

joinedTokens_pokemon = lem_wordsRDD_pokemon.map(join_tokens)



#convert joinedTokens to DataFrame add date column and text column and convert back to rdd
df_pokemon = joinedTokens_pokemon.map(lambda x: (x, )).toDF()
df_pokemon = df_pokemon.withColumn('row_index', f.monotonically_increasing_id())
sentimentAnalysisText = sentimentAnalysisText.withColumn('row_index', f.monotonically_increasing_id(
jointokendf_pokemon = df_pokemon.join(sentimentAnalysisText, on=["row_index"]).sort("row_index").dro
jointokendf_pokemon = jointokendf_pokemon.selectExpr("_1 as text", "created_at as Date")
jointokenrdd = jointokendf_pokemon.rdd.map(list)


newrdd=jointokenrdd.map(extract_phrases)
#applying Sentiment Analysis
sentimentRDD= newrdd.map(sentiment_words)

#Removing empty list
sentimentRDD1=sentimentRDD.map(lambda x : None if (x==[]) else x)

#removing Nones
sentimentRDD2=sentimentRDD1.filter(lambda x: x is not None)
```

```

#Mapping Text,Sentiment,Parity Value and the Date
sentimentRDD3 = sentimentRDD2.map(lambda x: (x[0][0],x[0][1],x[0][2],x[0][3][0]))

print(sentimentRDD3.take(5))

#write to an CSV file for us to leverage Tableau
Fileout = open("FinalSentimentScore.csv","w")
Fileout.write("Text_phrase,Sentiment,Parity,DateTime\n")
for line in sentimentRDD3.collect() :
    Fileout.write(','.join(str(var) for var in line))
    Fileout.write('\n')

print('\nThe data is successfully exported to the file :',Fileout.name)
Fileout.close()
```

```
Requirement already satisfied: nltk in /Library/anaconda3/lib/python3.7/site-packages (3.4.5)
Requirement already satisfied: six in /Library/anaconda3/lib/python3.7/site-packages (from nltk) (1.13.0
[('odinyt folagor portaventuraes subes tan alto que va al cielo donde estn tus', 'Neutral', 0.0, 'Fri Oc
t', 'Neutral', 0.0, 'Fri Oct 14 15:00:31 +0000 2016'), ('bignarstie fuck shud pokemon halloween', 'Negat
016'), ('morvantcheryl leak clinton spokesperson', 'Negative', -34.0, 'Thu Oct 13 15:04:27 +0000 2016'),
14:15:08 +0000 2016')]

The data is successfully exported to the file : FinalSentimentScore.csv
```

LDA - Topic Modeling, An unsupervised classification

In what context is Pokemon GO being used in and how are ´ the app users speaking of this game? Using Latent Dirichlet Allocation (LDA) alg
understand the context in which Pokemon GO ´ is being tweeted about. Topic modeling is a method for unsupervised classification of docum
even when the characteristics of each topic is unknown. The main approach for this is to select k number of topics for the algorithm, apply th
pokemongo, and analyze the features of the topics such as most frequent words used in each topic to find a real-life meaning for each topic.
tweets within this topic to understand which topics are the most popular. This type of analysis could be useful to improve the game or help u

In [12]:

```
##uses data frame from SA, filters on 'pok' to get pokemon tweets
# and additional text cleaning and tokenization
#some cleaning may be redone
datalda=dataf.filter(dataf.lang=='en')
tweets=datalda.rdd.map(lambda x : x['text']).filter(lambda x: x is not None)
sw = stopwords.words("english")
tokens = tweets                                                   \
    .map( lambda tweet: " ".join(re.findall('[A-Z][^A-Z]*',tweet)))              \
    .filter( lambda tweet: 'pokemon' in tweet.lower())               \
    .map( lambda tweet: tweet.strip().lower())               \
    .map( lambda tweet: re.sub('pokemon', ' ', tweet))          \
    .map( lambda tweet: re.sub('pokmon', ' ', tweet))          \
    .map( lambda tweet: re.sub('https', ' ', tweet))          \
    .map( lambda tweet: re.sub('\W+', ' ', tweet))          \
    .map( lambda tweet: re.split(" ", tweet))          \
    .map( lambda token_list: [x for x in token_list if x.isalpha()])           \
    .map( lambda token_list: [x for x in token_list if len(x) > 2] )           \
    .map( lambda token_list: [x for x in token_list if x not in sw])    \
    .filter(lambda x: x !=[])        \
    .zipWithIndex()
```

```
<>:14: DeprecationWarning:

invalid escape sequence \W

<>:14: DeprecationWarning:

invalid escape sequence \W

<>:14: DeprecationWarning:

invalid escape sequence \W

<ipython-input-12-d93f0439c441>:14: DeprecationWarning:

invalid escape sequence \W
```

In [13]:

```
##tf-idf vectors
df_tweets = sqlContext.createDataFrame(tokens, ["tokens",'index'])
df = df_tweets.filter(df_tweets.tokens. isNotNull())
# TF
cv = CountVectorizer(inputCol="tokens", outputCol="count_vector", vocabSize=10000)#, minDF=10.0)
cv_model = cv.fit(df_tweets)
tf= cv_model.transform(df_tweets)
# IDF
tf_idf = IDF(inputCol="count_vector", outputCol="features")
tfidfModel = tf_idf.fit(tf)
tfidf= tfidfModel.transform(tf)
```

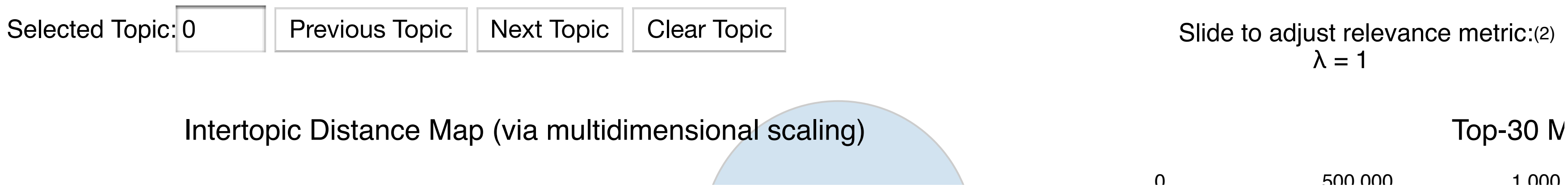
In [14]:

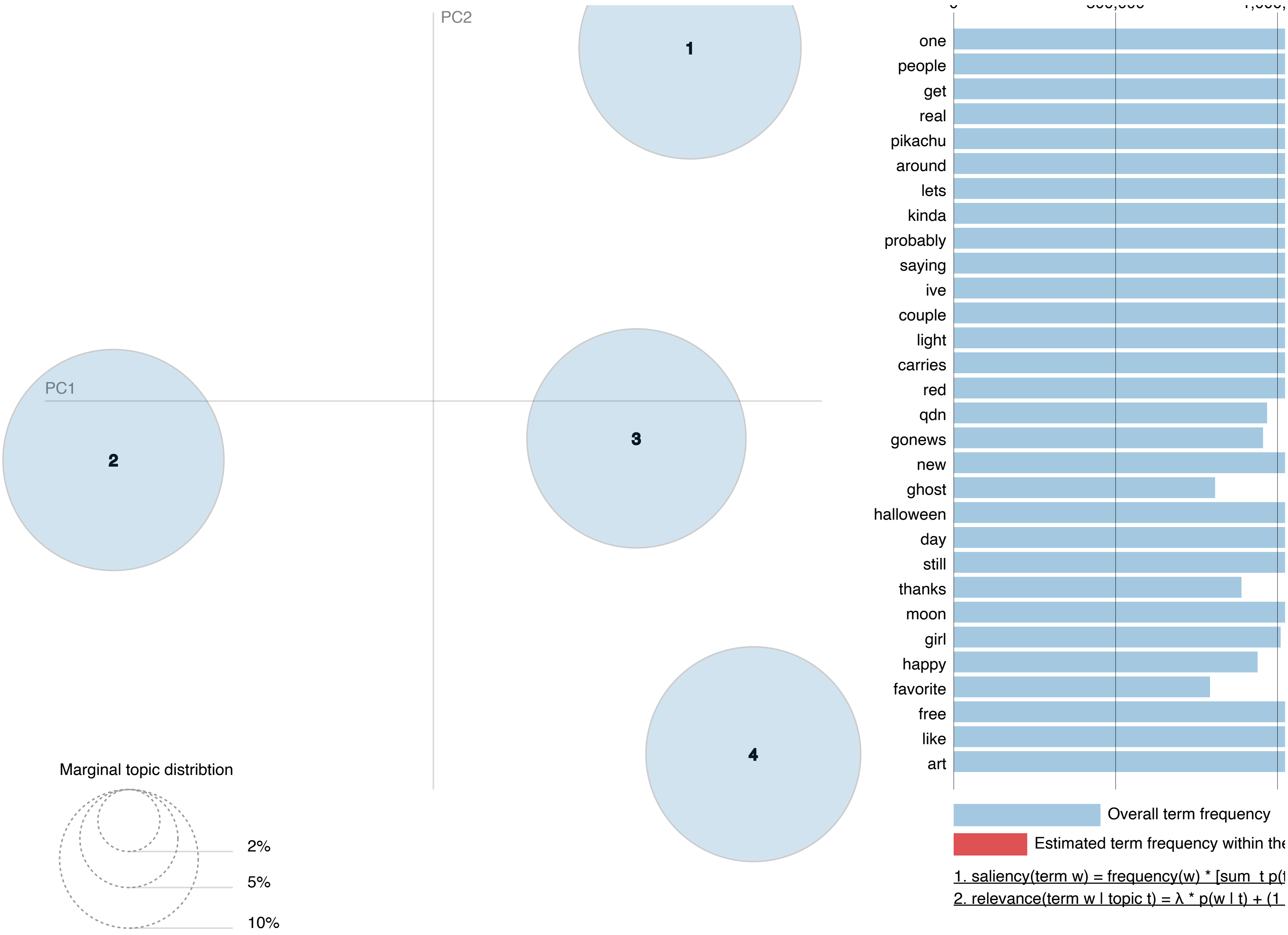
```
##Fitting Model
lda = LDA(k=4, optimizer="em")
lda_model = lda.fit(tfidf[['index','features']])
final_results = lda_model.transform(tfidf)
```

In [15]:

```python
###creates data in correct format to plot using pyLDAvis bubble chart from fitted LDA model
def format_data(dataframe, count_vectorizer, results, lda_model):
    x = dataframe.select((explode(dataframe.tokens)).alias("words")).groupby("words").count()
    word_counts = {r['words']:r['count'] for r in x.collect()}
    word_counts = [word_counts[w] for w in count_vectorizer.vocabulary]


    data = {'topic_term_dists': np.array(lda_model.topicsMatrix().toArray()).T,
            'doc_topic_dists': np.array([x.toArray() for x in results.select(["topicDistribution"]).
            'doc_lengths': [r[0] for r in dataframe.select(size(dataframe.tokens)).collect()],
            'vocab': count_vectorizer.vocabulary,
            'term_frequency': word_counts}

    return data

###filters out tweets where topics
def filter_tweets(data):
    new_doc_topic_dists = []
    new_doc_lengths = []

    for x,y in zip(data['doc_topic_dists'], data['doc_lengths']):
        if np.sum(x)==0 or np.sum(x) != 1 or np.isnan(x).any():
            pass
        else:
            new_doc_topic_dists.append(x)
            new_doc_lengths.append(y)

    data['doc_topic_dists'] = new_doc_topic_dists
    data['doc_lengths'] = new_doc_lengths

data = format_data(df, cv_model, final_results, lda_model)
filter_tweets(data)

pyLDAvis.enable_notebook()
lda_data = pyLDAvis.prepare(**data)
pyLDAvis.display(lda_data)
```

Out[15]:


Selected Topic: 0
Previous Topic
Next Topic
Clear Topic
Slide to adjust relevance metric:(2)
λ = 1
Intertopic Distance Map (via multidimensional scaling)
Top-30 M
0
500,000
1,000


PC2
PC1
1
2
3
4
Marginal topic distribtion
2%
5%
10%
0
one
people
get
real
pikachu
around
lets
kinda
probably
saying
ive
couple
light
carries
red
qdn
gonews
new
ghost
halloween
day
still
thanks
moon
girl
happy
favorite
free
like
art
Overall term frequency
Estimated term frequency within the
1. saliency(term w) = frequency(w) * [sum_t p(t
2. relevance(term w | topic t) = λ * p(w | t) + (1 -

Pokemon Facts - Rarity of Pokemons & Future Prediction(Time Series

Pokemon character they caught and the scores along with screenshot images of the Pokemon. In this study, one of our goals is to analyze th rarest Pokemon caught and tweeted about. We would be analyzing this task using visualizations and graphs either by the python libraries or

In [16]:

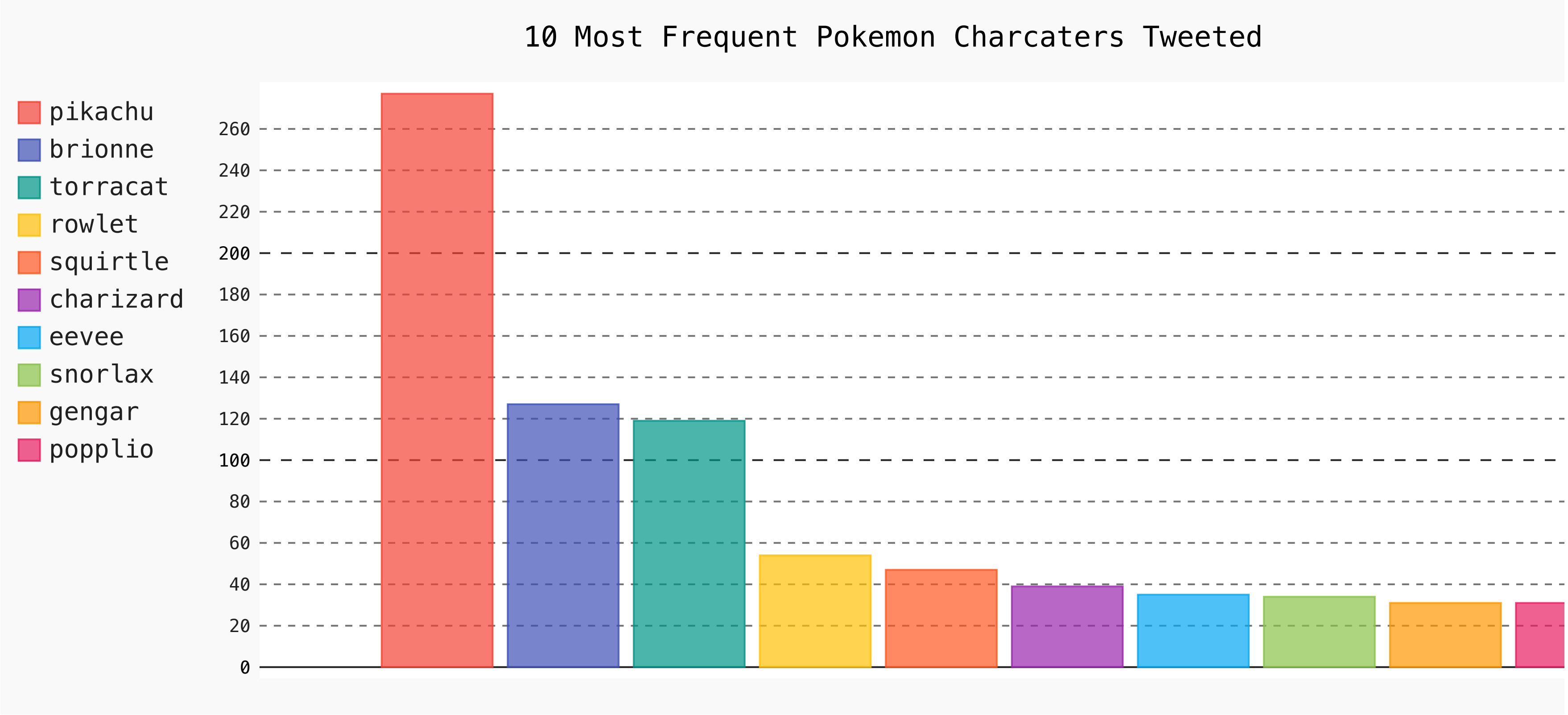
```
# Pokemon characters data
pokemonCharsdata = spark.read.csv('PokemonCharacters.csv',inferSchema=True, header=True)

pokemonChars_rdd = pokemonCharsdata.select("Pokemons").rdd.flatMap(lambda x: x)
#convert all strings to lower case and to list
pokemonList = pokemonChars_rdd.map(lambda x: x.lower()).collect()

commonCharsRDD = stopwordRDD_pokemon.map(lambda word: [x for x in word if x in pokemonList]).filter(
commonCharsRDD1 = commonCharsRDD.flatMap(lambda x: x) # flat map for getting each string
commonCharsRDD2 = commonCharsRDD1.map(lambda x: (x,1)) # assign 1 count for each string
commonCharsCount = commonCharsRDD2.reduceByKey(lambda a, b: a + b).sortBy(lambda x: x[1], ascending

commonCharsCount.collect()

pokemonCharsCountDf = commonCharsCount.toDF()

pokemonCharsCountDfPandas = pokemonCharsCountDf.toPandas()

pokemonCharsCountDf.write.mode('overwrite').csv("pokemonCharsCount")
```

In [17]:

```
!pip install pygal

import pygal
from pygal.style import Style
import pygal
from ipywidgets import HTML
from IPython.display import HTML
import base64

custom_style = Style(
  olors=('#E853A0', '#E8537A', '#E95355', '#E87653', '#E89B53'))

b_chart_pokemonFreq = pygal.Bar(style=custom_style,width=1000, height=400, explicit_size=True)
b_chart_pokemonFreq.title = "10 Most Frequent Pokemon Charcaters Tweeted"

for bar in range(len(pokemonCharsCountDfPandas.head(10))):
    b_chart_pokemonFreq.add(pokemonCharsCountDfPandas['_1'].iloc[bar], pokemonCharsCountDfPandas['_2


from IPython.display import SVG, HTML

"""b64 = base64.b64encode(b_chart.render())
src = 'data:image/svg+xml;charset=utf-8;base64,'+(b64)
HTML('<embed src={}></embed>'.format(src))"""
html_pygal = u"""
    <!DOCTYPE html>
    <html>
        <head>
             <script type="text/javascript" src="http://kozea.github.com/pygal.js/javascripts/svg.jq
              <script type="text/javascript" src="https://kozea.github.io/pygal.js/2.0.x/pygal-toolt
        </head>
        <body><figure>{pygal_render}</figure></body>
    </html>
"""
HTML(html_pygal.format(pygal_render = b_chart_pokemonFreq.render(is_unicode=True)))
```

```
Requirement already satisfied: pygal in /Library/anaconda3/lib/python3.7/site-packages (2.4.0)
```

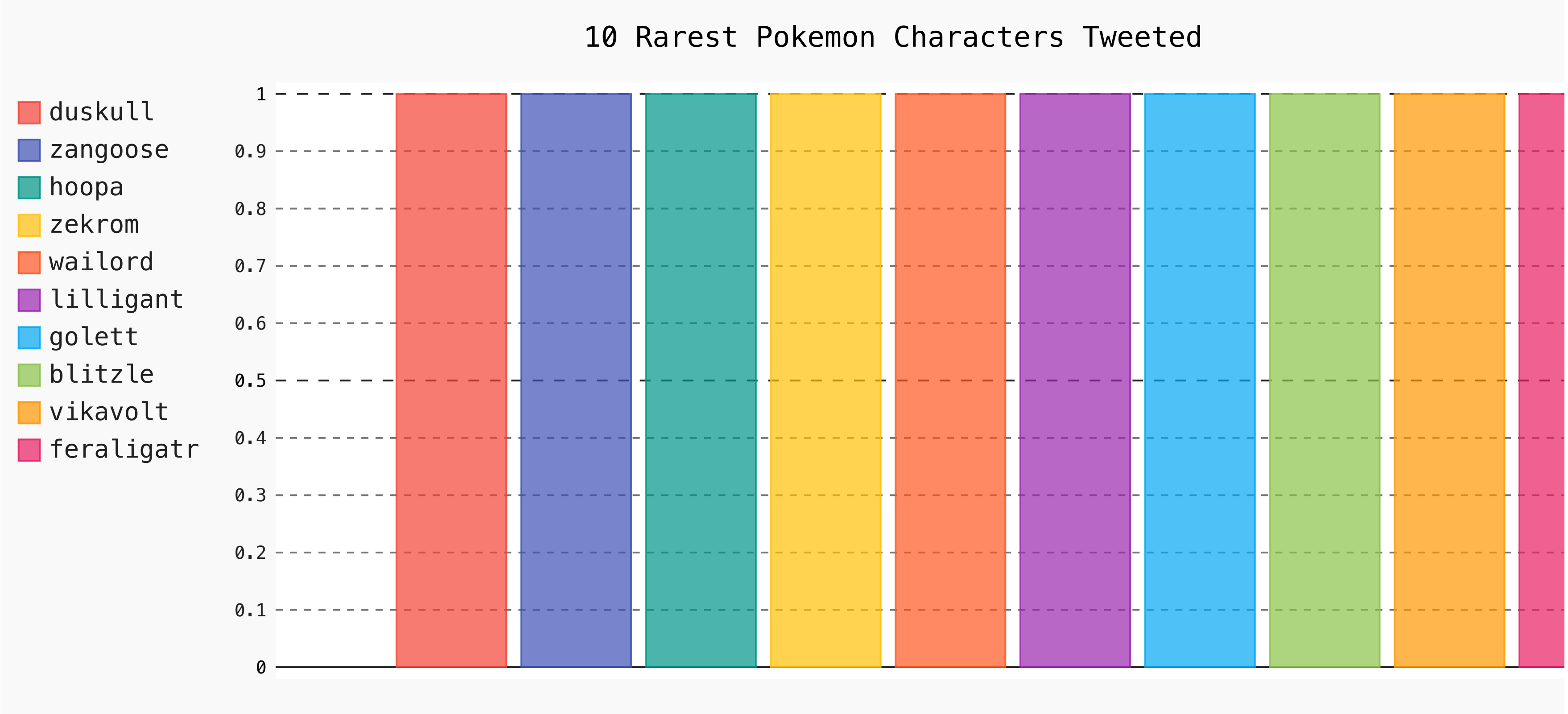
Out[17]:


10 Most Frequent Pokemon Charcaters Tweeted
pikachu
brionne
torracat
rowlet
squirtle
charizard
eevee
snorlax
gengar
popplio
260
240
220
200
180
160
140
120
100
80
60
40
20
0

In [18]:

```
b_chart_pokemonFreq1 = pygal.Bar(style=custom_style,width=1000, height=400, explicit_size=True)
b_chart_pokemonFreq1.title = "10 Rarest Pokemon Characters Tweeted"
rarestPokemonsDf1 = pokemonCharsCountDfPandas.iloc[-10:]
for bar in range(len(rarestPokemonsDf1)):
    b_chart_pokemonFreq1.add(rarestPokemonsDf1['_1'].iloc[bar], rarestPokemonsDf1['_2'].iloc[bar])

HTML(html_pygal.format(pygal_render = b_chart_pokemonFreq1.render(is_unicode=True)))
```

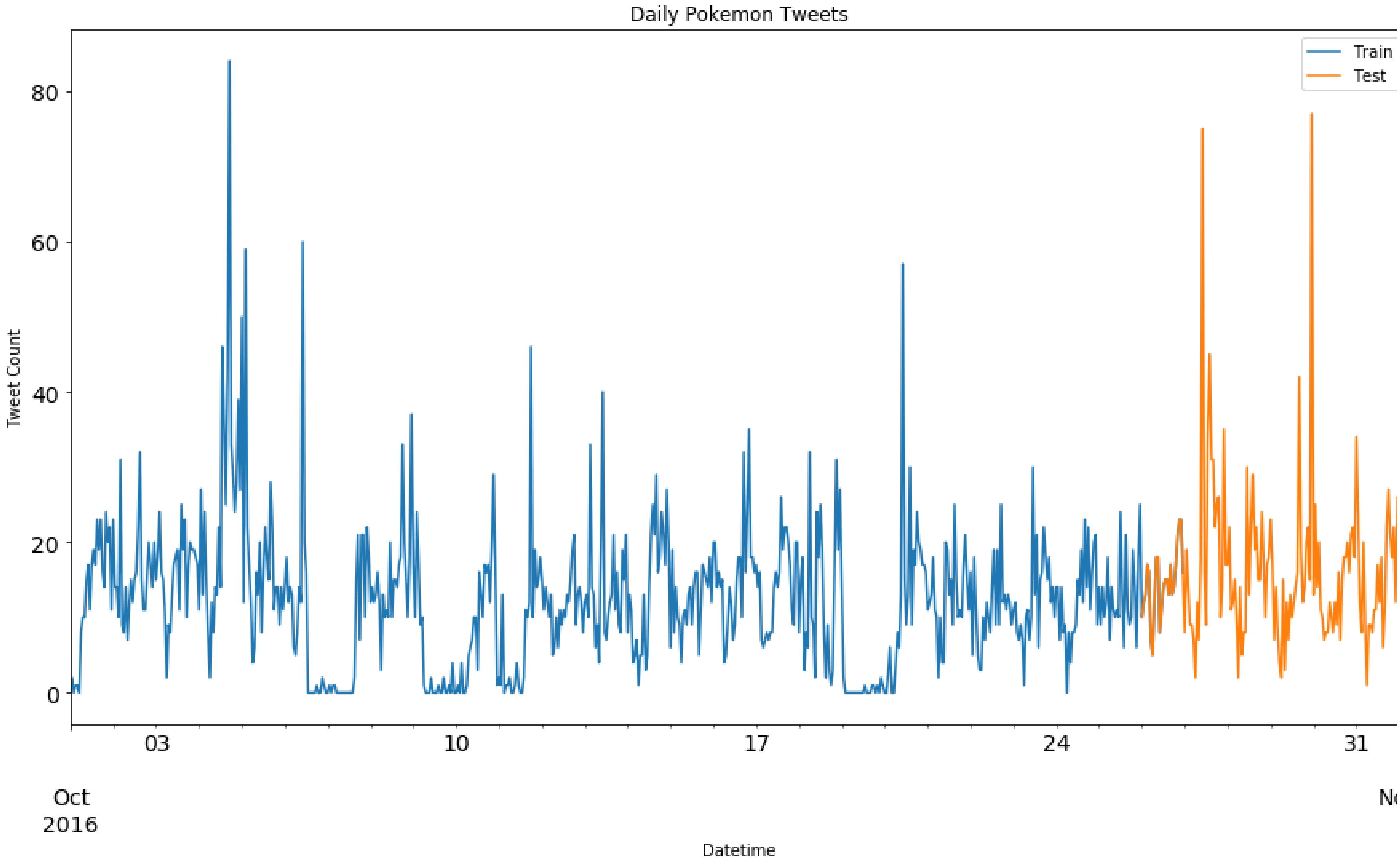
Out[18]:



TIME SERIES ANALYSIS

In [19]:

```
# Get Created At timestamp data from Poekon Tweets and drop nulls
createdAtDf = sentimentAnalysisText.select("created_at")
createdAtDf = createdAtDf.dropna()
from datetime import datetime

createdAtPandasDf = createdAtDf.toPandas()
createdAtPandasDf = createdAtPandasDf.iloc[:-1]

# Remove unnecessary row
createdAtPandasDf['created_at'] = createdAtPandasDf[createdAtPandasDf['created_at']!=' poketesecuest
# Format Datetime from created_at
createdAtPandasDf['datetime'] = pd.to_datetime(createdAtPandasDf.created_at, format = '%a %b %d %H:%
# Format date time to DD-MM-YYYY HH:mm format for Time Series Analysis
createdAtPandasDf['datetime_new'] = createdAtPandasDf['datetime'].dt.strftime('%d-%m-%Y %H:%M')
# get Datetime data to Timestamp format
train=createdAtPandasDf['datetime_new']
train.Timestamp = pd.to_datetime(createdAtPandasDf.datetime_new, format='%d-%m-%Y %H:%M')
train.index = train.Timestamp

#Get Pokemon Tweets counts on Hourly basis
train = train.resample('H').count()
# Get Training and Testing data
fig, ax = plt.subplots(figsize=(13,7))
Train = train.loc['2016-10-01':'2016-10-26']
test = train.loc['2016-10-26':'2016-11-01']
Train.plot(figsize = (15,8), title = 'Daily Pokemon Tweets', fontsize = 14, label = 'Train')
test.plot(figsize = (15,8), title = 'Daily Pokemon Tweets', fontsize =14, label = 'Test')
plt.xlabel('Datetime')
plt.ylabel('Tweet Count')
plt.legend(loc = 'best')
plt.show()


# Taking log values of Train & Test data
Train_log = np.log(Train)
test_log = np.log(test)

# Dropping inf and na values
Train_log = Train_log[~np.isinf(Train_log)]
```

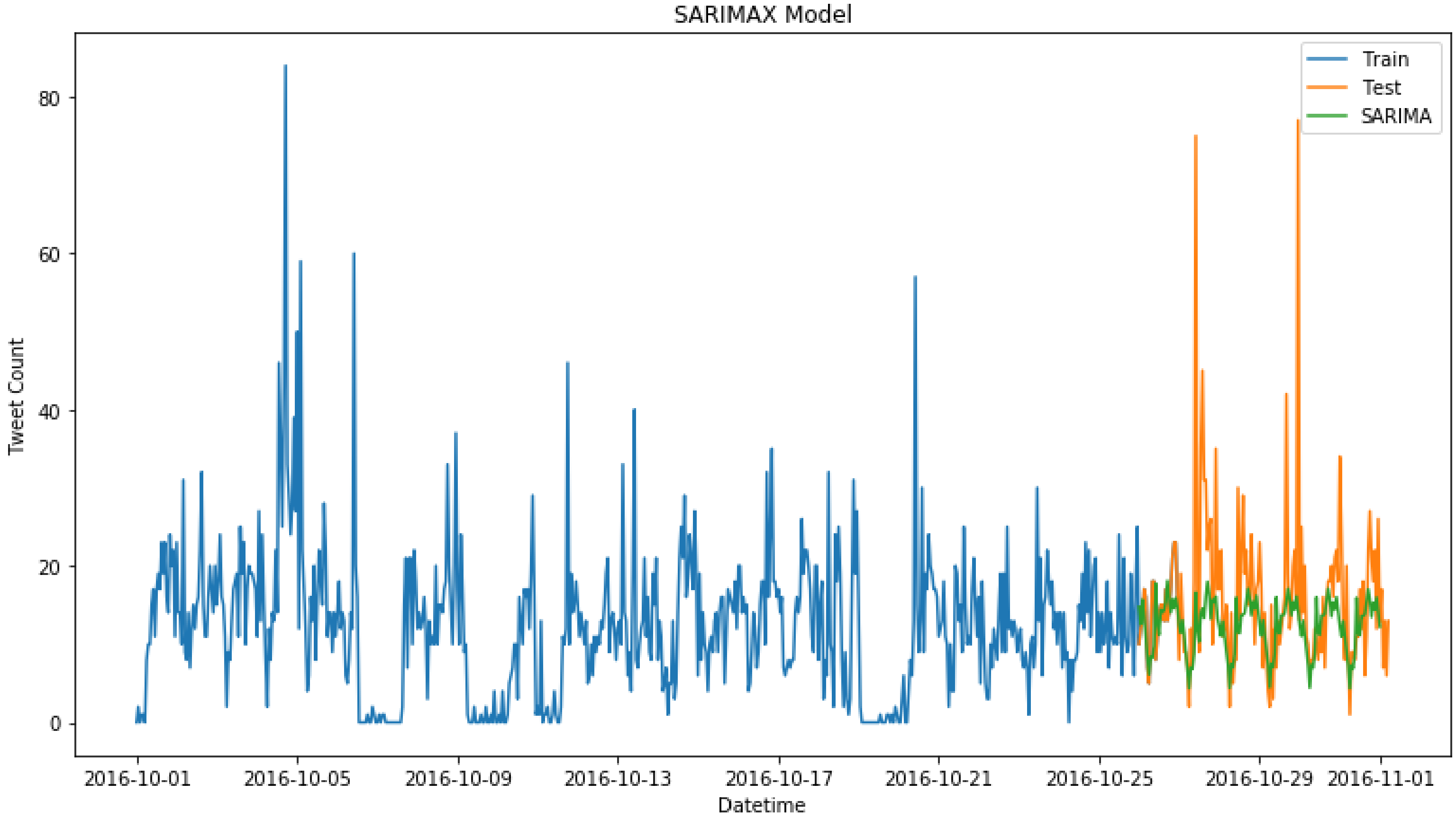


SARIMAX Model

In [20]:

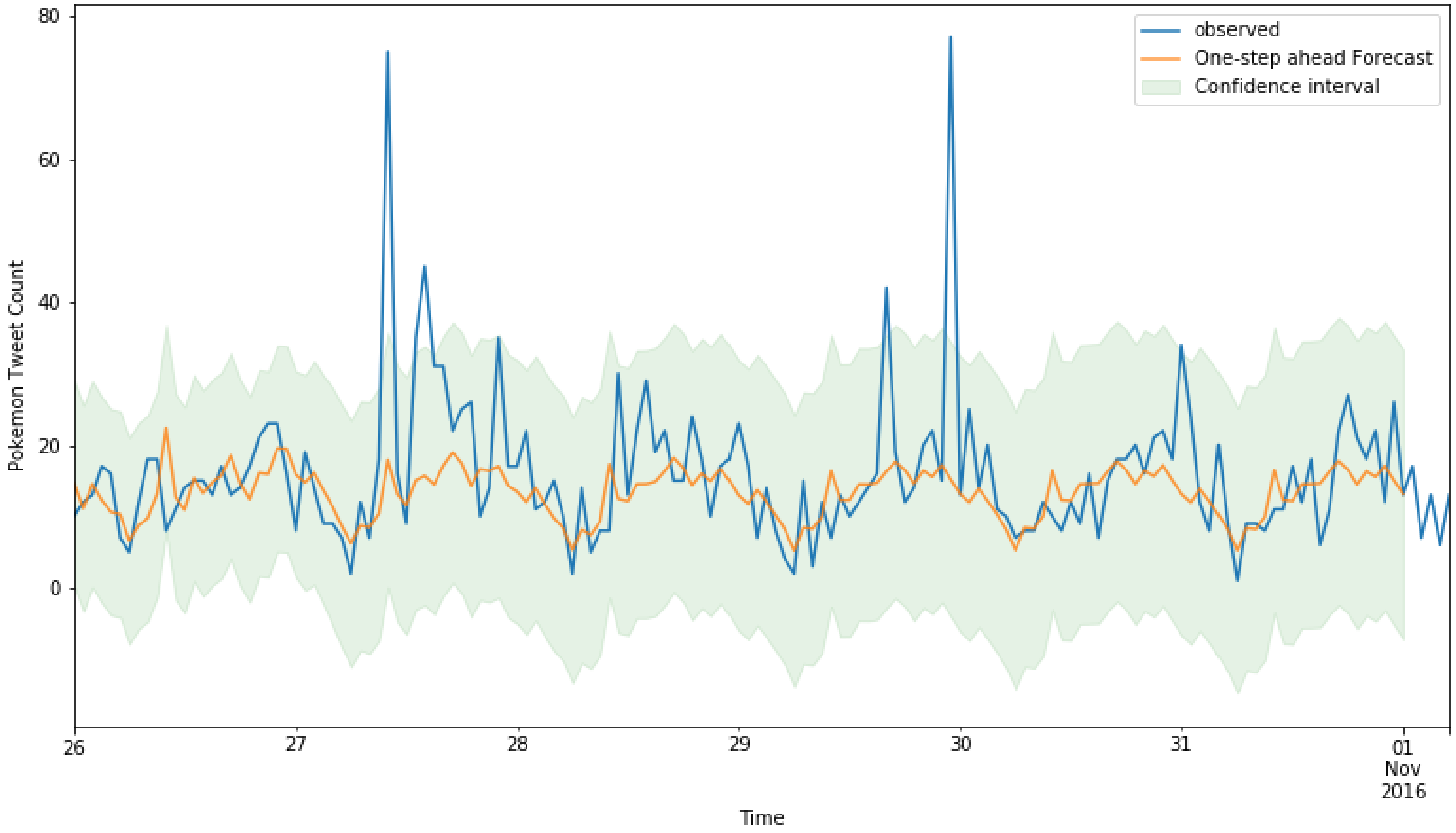
```python
import statsmodels.api as sm
y_hat_avg = test.copy()

# fit SARIMAX model with seasonal order of 24 hrs
fit1 = sm.tsa.statespace.SARIMAX(Train, order = (4,1,1), seasonal_order =(2,1,1,24)).fit()
y_hat_avg['SARIMA'] = fit1.predict(start="2016-10-26", end="2016-11-01", dynamic=True)
print(fit1.summary())

fig, ax = plt.subplots(figsize=(13,7))
plt.plot(Train, label = "Train")
plt.plot(test, label = "Test")
plt.plot(y_hat_avg['SARIMA'], label ="SARIMA")
plt.legend(loc = "best")
plt.title("SARIMAX Model")
plt.xlabel('Datetime')
plt.ylabel('Tweet Count')
plt.show()
```

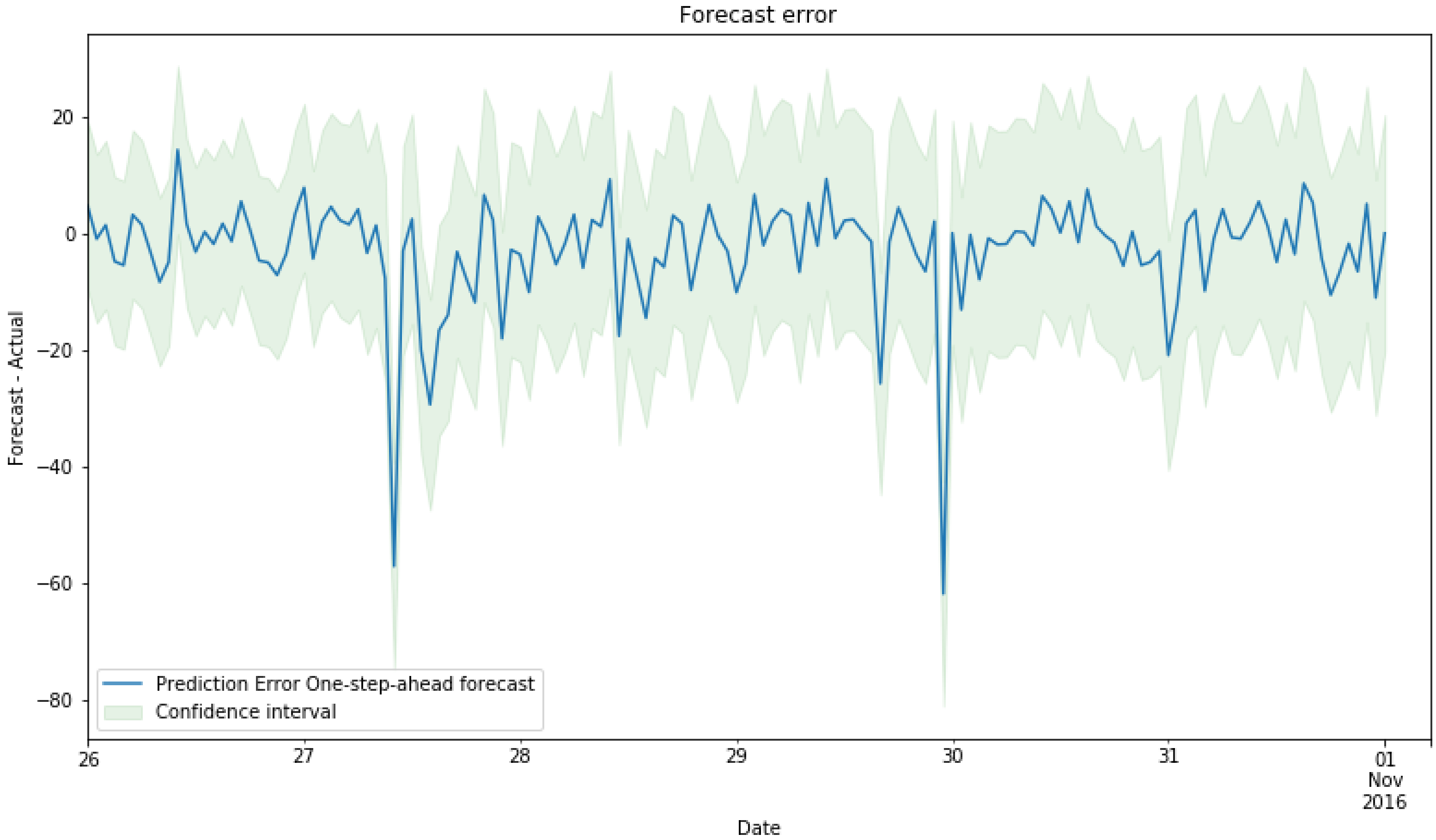
```
/Library/anaconda3/lib/python3.7/site-packages/statsmodels/tsa/base/tsa_model.py:165: ValueWarning:

No frequency information was provided, so inferred frequency H will be used.

/Library/anaconda3/lib/python3.7/site-packages/statsmodels/base/model.py:512: ConvergenceWarning:

Maximum Likelihood optimization failed to converge. Check mle_retvals


                                 Statespace Model Results
==========================================================================================
Dep. Variable:                       datetime_new   No. Observations:                  624
Model:             SARIMAX(4, 1, 1)x(2, 1, 1, 24)   Log Likelihood               -2077.756
Date:                            Mon, 09 Dec 2019   AIC                           4173.511
Time:                                    21:50:07   BIC                           4213.069
Sample:                                10-01-2016   HQIC                          4188.911
                                     - 10-26-2016
Covariance Type:                              opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          0.2854      0.034      8.367      0.000       0.219       0.352
ar.L2          0.1958      0.034      5.822      0.000       0.130       0.262
ar.L3          0.0768      0.033      2.294      0.022       0.011       0.142
ar.L4          0.1467      0.032      4.626      0.000       0.085       0.209
ma.L1         -0.9834      0.015    -63.976      0.000      -1.013      -0.953
ar.S.L24      -0.0328      0.049     -0.670      0.503      -0.129       0.063
ar.S.L48      -0.0898      0.048     -1.855      0.064      -0.185       0.005
ma.S.L24      -0.9962      0.975     -1.021      0.307      -2.908       0.916
sigma2        52.2633     49.668      1.052      0.293     -45.085     149.612
===================================================================================
Ljung-Box (Q):                       26.95   Jarque-Bera (JB):              1692.40
Prob(Q):                              0.94   Prob(JB):                         0.00
Heteroskedasticity (H):               0.55   Skew:                             1.57
Prob(H) (two-sided):                  0.00   Kurtosis:                        10.61
===================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
```



In [21]:

```
# Get Predictions
fig, ax = plt.subplots(figsize=(13,7))
pred = fit1.get_prediction(start="2016-10-26", end="2016-11-01", dynamic=False) # start and end date
pred_ci = pred.conf_int() # confidence intervals of predicted model

ax = test.plot(label='observed')
pred.predicted_mean.plot(ax=ax, label='One-step ahead Forecast', alpha=.8)


ax.fill_between(pred_ci.index, pred_ci.iloc[:,0], pred_ci.iloc[:,1], color='g', alpha=0.1, label='Co

legend = ax.legend(loc='lower right')

ax.set_xlabel('Time')
ax.set_ylabel('Pokemon Tweet Count')
plt.legend()

plt.show()


# Prediction error

# Graph
fig, ax = plt.subplots(figsize=(13,7))
ax.set(title='Forecast error', xlabel='Date', ylabel='Forecast - Actual')

predict_error = pred.predicted_mean - test
predict_error.plot(ax=ax, label='Prediction Error One-step-ahead forecast')
ci = pred_ci.copy()
ci.iloc[:,0] -= test
ci.iloc[:,1] -= test
ax.fill_between(ci.index, ci.iloc[:,0], ci.iloc[:,1], alpha=0.1, color='g', label='Confidence interv

legend = ax.legend(loc='lower left');
legend.get_frame().set_facecolor('w')
```


observed
One-step ahead Forecast
Confidence interval
Pokemon Tweet Count
0
20
40
60
80
26
27
28
29
30
31
01
Nov
2016
Time


Forecast error
20
0
−20
−40
−60
−80
Forecast - Actual
Prediction Error One-step-ahead forecast
Confidence interval
26
27
28
29
30
31
01
Nov
2016
Date

In [22]:

```
# Accuracy metrics
def forecast_accuracy(predictedValue, actualValue):
    meanAbsPercErr = np.mean(np.abs(predictedValue - actualValue)/np.abs(actualValue))
    meanErr = np.mean(predictedValue - actualValue)
    meanAbsErr = np.mean(np.abs(predictedValue - actualValue))
    meanPercErr = np.mean((predictedValue - actualValue)/actualValue)
    rootMeanSqErr = np.mean((predictedValue - actualValue)**2)**.5
    corrVal = np.corrcoef(predictedValue, actualValue)[0,1]
    minimum = np.amin(np.hstack([predictedValue[:,None],
                              actualValue[:,None]]), axis=1)
    maximum = np.amax(np.hstack([predictedValue[:,None],
                              actualValue[:,None]]), axis=1)
    minmaxErr = 1 - np.mean(minimum/maximum)

    return({'Mean Absolute Percentage Error (MAPE)':meanAbsPercErr,
            'Mean Error (ME)':meanErr,
            'Mean Absolute Error (MAE)': meanAbsErr,
            'Mean Percentage Error (MPE)': meanPercErr,
            'Root Mean Squared Error (RMSE)':rootMeanSqErr,
            'Correlation between the Actual and the Forecast (corr)':corrVal,
            'Min-Max Error (minmax)':minmaxErr})

forecast_accuracy(pred.predicted_mean, test[:145].values)
```

Out[22]:

```
{'Mean Absolute Percentage Error (MAPE)': 0.39347628883985236,
 'Mean Error (ME)': -2.748887281490209,
 'Mean Absolute Error (MAE)': 5.706124549674806,
 'Mean Percentage Error (MPE)': 0.07207168796080622,
 'Root Mean Squared Error (RMSE)': 9.866293818645284,
 'Correlation between the Actual and the Forecast (corr)': 0.4134771290686682,
 'Min-Max Error (minmax)': 0.2771540645967224}
```

References:

1. https://www.statsmodels.org/dev/examples/notebooks/generated/statespace_sarimax_stata.html (https://www.statsmodels.org/dev/examples/notebooks/generated/statespace_sarimax_stata.html)
2. https://www.machinelearningplus.com/time-series/arima-model-time-series-forecasting-python/ (https://www.machinelearningplus.com python/)
3. https://medium.com/@rrfd/sarima-modelling-for-car-sharing-basic-data-pipelines-applications-with-python-pt-1-75de4677c0cd (https:/ basic-data-pipelines-applications-with-python-pt-1-75de4677c0cd)
4. https://ashwin-ks.github.io/2018-05-02-Time-Series-Modelling-using-Python/ (https://ashwin-ks.github.io/2018-05-02-Time-Series-Moc

5. Geopy . pypi. Accessed: 2019-11-8.
6. Google translate api. https://pypi.org/project/googletrans/ (https://pypi.org/project/googletrans/). Accessed:2019-12-10.
7. Lda visualization. https://stackoverflow.com/questions/41819761/pyldavis-visualization-of-pyspark-generated-lda-model (https://stacko of-pyspark-generated-lda-model). Accessed: 2019-11-10.
8. Pygal svg graphs with flask tutorial. Accessed: 2019-11-8.
9. Sparkclustering. https://spark.apache.org/docs/latest/ml-clustering.htmllatent-dirichlet-allocation-lda (https://spark.apache.org/docs/lat Accessed: 2019-11-2.
10. Threading interface. https://docs.python.org/2/library/multiprocessing.html.Accessed (https://docs.python.org/2/library/multiprocessing.
11. Nagesh Singh Chauhan. Natural language processing in apache sparkusing nltk (part 1/2), February 2017. Accessed: 2019-11-8.
12. Nagesh Singh Chauhan. Natural language processing in apache sparkusing nltk (part 2/2), February 2017. Accessed: 2019-11-8.
13. Robert R.F. DeFilippi. Sarima modelling for car sharing — basic datapipelines — applications with python pt. 1, May 2018. Accessed: 20
14. the free encyclopedia From Wikipedia. Pok´emon go, November 2019.Accessed: 2019-11-10.
15. Soumya Ghosh. Topic modelling with latent dirichlet allocation (lda)in pyspark. https://medium.com/@connectwithghosh/topic-modelling 2cb3ebd5678e,March2018 (https://medium.com/@connectwithghosh/topic-modelling-with-latent-dirichlet-allocation-lda-in-pyspark-2cl
16. Guru99. Pos (part-of-speech) tagging chunking with nltk, February2017. Accessed: 2019-11-8.[13] K. M. Badran H. Elzayady and G. I. S spark framework, November 2018. Accessed:2019-11-8.
17. PhD James Lani. Time series analysis, December 2019. Accessed:2019-12-06.
18. Ricky Kim. Sentiment analysis with pyspark, 2018.
19. Nikolaos Nodarakis. Large scale sentiment analysis on twitter withspark, November 2016. Accessed: 2019-11-8.
20. C.J. Hutto Eric Gilbert Georgia Institute of Technology.Vader: Aparsimonious rule-based model for sentiment analysis of social mediatex JuliaSilgeandDavidRobinson.Textminingwithr.https://www.tidytextmining.com/topicmodeling.html (https://www.tidytextmining.com/topic
21. P. Sanguanbhoki T. Raicharoen, C. Lursinsap. Application of criticalsupport vector machine to time series prediction.InCircuits andSyste

In [ ]:

```

```