twitter_videogame

June 15, 2018

1 How Many Videogames Can You Sell?

By Sarah Floris

1.1 The Project

With the advances in technology, it is no surprise that both the number of videogames and tweets have increased. VGChartz contains a database with the characteristics of videogames including genre, year of release, and both weekly and total sales. Combining all of these characteristis, technology companies are continually competing with each other for the best game and prices. Of course, how does the game's characteristics influence the total unit sales?

This project investigates the correlation between publisher continent, number of tweets to sales, and ultimately be able to predict the weekly sales for the following week.

1.2 The Data

I web-scraped names,genres,platforms,sales,dates for global, North American, Asian, and European regions from VGChartz.com. Global includes United States, Europe, and Japan. Europe includes UK,Germany, and France. North American is United States and Asian is just Japan. The countries that represent an entire region such as United States for North America means that the sales for the other countries are minimal in comparison to United States. In order to "standardize" the data among the regions, I used the populations defined by worldometers for 2018.

Additionally, I web-scraped Twitter to get enough tweets to give an accurate representation of the popularity of a game.

1.3 Methodology

I approached the problem from a supervised learning regression view The values that are predicted show the unit sales of videogames per 1000 people in the United States.

pandas for:

```
data loading, wrangling, cleaning, and manipulation
feature selection and engineering
descriptive statistics
dummy variable creation
```

bs4 for:

```
web-scraping, converting json to readable format
```

numpy for:

```
array data structure, the primary input for classifiers
model comparison
matrix manipulation
```

multiprocessing for:

```
task distribution
```

scikit-learn for:

```
classifier models
parameter gridsearch
model evaluation
```

geocoder for:

```
finding videogame publishers' locations
```

matplotlib for:

```
data visualization
```

plotly for:

```
data visualization
```

1.4 Data Wrangling and Cleaning

Since the majority of data is web-scraped, this process takes the longest. However, when it is done running, it is automatically saved to a file, which can be easily accessed next time. Pandas quickly reads the files in using pd.read_csv. The majority of cleaning required the careful removal of NaN's and unnecessary columns. Since the total sales data included genre and other characteristics that the weekly sales had, I had to go through the genres, location, etc and add them individually to the dataframe.

In order for the Twitter to run appropriately, I made a dataframe with all of the names of the games and then the dates that need to be ran. I cut the dates off at 2006 because that is the earliest date that Twitter has tweets.

1.5 Feature Selection and Engineering

The dataset contained a mixture of categorical and numerical data. First, the performance of just the numerical columns was verified. When these models were giving a small rˆ2, dummy variables were created for the categorical data. Ultimately, the dataset contained both the numerical columns and the dummy variable columns.

1.6 Model Fitting

Models were fit using RandomSearchCV or GridSearchCV, which searches through a grid of parameters for each model, returning the model that gives the highest k-fold cross validation score.

Linear Regression Linear regression basically models a linear relationship between dependent variable and the other independent variables.

Elastic Net ElasticNet is introduced as a compromise between these two techniques, and has a penalty which is a mix of L1 and L2 norms, where L1 penalty limits the size of the coefficient vector and L2 penalty imposes sparsity among the coefficients, making the fitted model more interpretable.

RandomForest Regression This ensemble method creates random trees and evaluates them. This is a good way to check many different trees, as opposed to DTC. As it creates and compares 10 different trees every time it's called, RF gets very expensive, so cross-validation was limited to 5-fold.

GradientBoostingRegressor Gradient boosting involves a loss function to be optimized, a weak learner to make predictions, and an additive model to add weak learners to minimize the loss function.

AdaBoostRegressor with RandomForest Regressor as initial estimator AdaBoost (Adaptive Boosting) fits a sequence of weak learners on different weighted training data. It starts by predicting original data set and gives equal weight to each observation. If prediction is incorrect using the first learner, then it gives higher weight to observation which have been predicted incorrectly. Being an iterative process, it continues to add learner(s) until a limit is reached in the number of models or accuracy.

1.7 Model Evaluation

The R squared value in combinations with the mean squared error is a fairly good indication of how well the model is doing. The goal of the model would be to predict videogame unit sales.

I also ran k-fold cross-validation to make sure that the variations in accuracy across different data samples were consistent. Inconsistency of the data suggests overfitting of the training data.

1.8 Results

The majority of the games are produced in North America and Europe. The most common genre is action and shooting. EA Canada and Capcom produced the most games. In the selection of games, the most common genre is action and shooting. The game who sold the most units was Super Mario Bros for the Nintendoplatform while the game with the most tweets was Grand Theft Auto V.

The most unit sales are around the generally around the release date and during the holidays.

The best machine learning method was AdaBoostRegressor with a RandomForestRegressor as a base estimator. Test set accuracy score for best params: 0.375

0.37511167387659317 AdaBoostRegressor(base_estimator=RandomForestRegressor(bootstrap=True, criterion=’mse’, max_depth=None, max_features=’sqrt’, max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, min_samples_leaf=1, min_samples_split=2, min_weight_fraction_leaf=0.0, n_estimators=9, n_jobs=-1, oob_score=False, random_state=None, verbose=0, warm_start=False), learning_rate=1.0, loss=’linear’, n_estimators=50, random_state=None)

Using this method, I was able to predict the unit sales of videogames by name and console for the next four weeks.

1.9 Limitations

Current limitations of this project would be lacking the resources to run web-scraping of the tweets to get a more accurate representation of the popularity. The modeling is only as good as the data that is available and the underlying assumptions made. A variety of assumptions were made: * Publisher’s location is the first location that pops up on OpenMapQuest * 500 tweets is the max tweets available per week for a set of games (does not include all videogames) * Populations in countries are unchanged

1.10 Further Research

For further analysis, I would lookat adding more tweets. I currently only used a maximum of 500 tweets per week for every game. This currently does not create an accurate representation of popularity.

Since I currently assumed that the popularity of the videogame is determined by the number of tweets, it would be informative to add a sentiment analysis on top of just the tweets. A negative tweet could have a certain value such as -1 and a positive tweet would be than 1, determinating the underlining reasoning behind the popularity of a game.

1.11 Client Recommendations

To the publisher company, I would suggest making fighter and action focused games geared towards North Americans and Europeans. For asian-focused publishers, I would recommend designing more games in the puzzles and strategy types of games.

Since the unit sales are the highest during holidays and the release time, I would suggest either to publish their game during the holidays, or to release the videogame separate from the holidays but still during the fall.

2 Code

2.0.1 Import Modules

```
In [1]: from pandas.io.json import json_normalize
        from multiprocessing.pool import ThreadPool
        from bs4 import BeautifulSoup
        import pandas as pd
        import requests
        import scipy.stats as stats
        import matplotlib.pyplot as plt
```

```
         import json
         import seaborn as sns
         import re
         import difflib
         import geocoder
         import datetime
         import numpy as np
         import os
```

First. Let's grab all the data from the videogames sales from the website. The first set of data will be the overview of the majority of videogames. The second set of data will be videogames with top 75 weekly sales and total sales.

3 Web Scraping VGChartz for Total Sales

```
In [2]: pages = 200
        rank = []
        game = []
        platform = []
        year = []
        genre = []
        publisher = []
        sales_na = []
        sales_eu = []
        sales_jp = []
        sales_ot = []
        sales_tot = []
        console = []
        chartz_score = []
        critic_score = []
        user_score = []

        urlhead = 'http://www.vgchartz.com/gamedb/?page='
        urltail = '&results=1000&name=&keyword=&console=&region=All&developer=&publisher=&goty_
        clean_list = []

        for page in range(1,pages):
            surl = urlhead + str(page) + urltail
            r = requests.get(surl)
            soup = BeautifulSoup(r.text,'html.parser')

            #even rows
            charts = soup.find_all("tr", attrs={"style": "background-image:url(../imgs/chartBa

            for row in charts:
                values = []
                for value in row.find_all("td"):
                    if value.text != ' ':
                        values.append(value.text)
                values = list(filter(str.strip, values))
                clean_list.append(values)

            #odd rows
            charts = soup.find_all("tr", attrs={"style": "background-image:url(../imgs/chartBa

            for row in charts:
                values = []
                for value in row.find_all("td"):
                    if value.text != ' ':
                        values.append(value.text)
                values = list(filter(str.strip, values))
                clean_list.append(values)


        for row in clean_list:
            rank.append(row[0])
            game.append(row[1])
            console.append(row[2])
            publisher.append(row[3])
            chartz_score.append(row[4])
            critic_score.append(row[5])
            user_score.append(row[6])
            sales_tot.append(row[7])
            sales_na.append(row[8])
            sales_eu.append(row[9])
            sales_jp.append(row[10])
            sales_ot.append(row[11])
            year.append(row[12])
```

```
In [3]: columns = ['Name','Platform','Publisher','Chartz_Score','User_Score','Critic_Score','G
        sales = pd.DataFrame(np.column_stack([game,console,publisher,chartz_score,critic_score
        sales.to_csv("vgsales.csv",sep=",",encoding='utf-8')
```

```
In [4]: #Cleaning up game's year release in sales dataframe
        cleaner_dates = []
        for date in sales['Year_of_Release']:
            if not date[2].isdigit():
                date = date[0:2] + date[4:len(date)]
            cleaner_dates.append(date)
        sales['Year_of_Release'] = cleaner_dates
```

```
In [5]: #Creates a genre column for the overall list

        urlhead = 'http://www.vgchartz.com/gamedb/?page='
        urltail = '&results=1000&name=&keyword=&console=&region=All&developer=&publisher=&goty
        clean_list = []
        links = []

        pattern = re.compile("http:")
        ##pages is defined above (2 boxes up)
        for page in range(1,pages):
            surl = urlhead + str(page) + urltail
            r = requests.get(surl)
            soup = BeautifulSoup(r.text, 'html.parser')
            #even
            charts = soup.find_all("tr", attrs={"style": "background-image:url(../imgs/chartBa
            for row in charts:
                for value in row.find_all("a"):
                    if pattern.match(value['href']):
                        links.append(value['href'])
            #odd
            charts = soup.find_all("tr", attrs={"style": "background-image:url(../imgs/chartBa
            for row in charts:
                for value in row.find_all("a"):
                    if pattern.match(value['href']):
                        links.append(value['href'])
```

```
In [6]: def linked(link):
            """Given a videogame link, find the name of the game and genre."""
            genre = []
            r = requests.get(link)
            soup = BeautifulSoup(r.text, 'html.parser')
            for title in soup.find_all("h1"):
                if title.find('a'):
                    values = title.text.split(" ")
                    name = ' '.join(values[:-1])
                    genre.append(name)

            for text in soup.find_all("h2"):
                if text.text == "Genre":
                    genre.append(text.next_sibling.text)
            return genre
```

```
In [7]: if not os.path.exists('Data/genre.csv') or os.stat('Data/genre.csv').st_size == 0:
            genre = []
            pool=ThreadPool(10)
            genre = pool.map(linked,links)
            pool.close()
            pool.join()
            with open('genre.csv', 'w') as f:
                f.write(json.dumps(genre))
        else:
            #Now read the file back into a Python list object
```

```
    f = open('genre.csv', 'r')
    l = eval(f.read())
    l = [row for row in l if row != ["Game Search (Add new game)       "]]
    l = list(filter(None, l))
    genre = pd.DataFrame(l, columns=['Name','Genre'])
```

```
In [8]: #Need to initialize the Genre column
        sales['Genre'] = np.nan
        for index, row in genre.iterrows():
            close_matches = difflib.get_close_matches(row['Name'],sales['Name'])
            for close_match in close_matches:
                sales.loc[sales['Name']==close_match,'Genre'] = row['Genre']
```

```
In [9]: names = []
        for name in sales['Name']:
            match = re.search(r'Read the review', str(name))
            if match:
                names.append(re.sub(r'Read the review','',name))
            else:
                names.append(name)
        sales['Name'] = names
```

```
In [10]: try:
             sales = sales.where(sales != 'N/A')
             sales = sales.where(sales != "N/A  ")
             sales['JP_Sales'] = sales['JP_Sales'].str.replace('m', '')
             sales['EU_Sales'] = sales['EU_Sales'].str.replace('m', '')
             sales['NA_Sales'] = sales['NA_Sales'].str.replace('m', '')
             sales['Other_Sales'] = sales['Other_Sales'].str.replace('m', '')
             sales['Global_Sales'] = sales['Global_Sales'].str.replace('m', '')
             sales.to_csv("vgsales.csv",sep=",",encoding='utf-8')
         except:
             print('m is already substituted')
```

```
In [11]: sales.to_csv("vgsales.csv",sep=",",encoding='utf-8')
         sales = pd.read_csv('vgsales.csv')
```

3.1 Webscraping VGChartz for Weekly Sales

Now, we are ready to get the weekly sales information off of the website.

```
In [12]: def get_dates():
             """Find all the dates currently on vgchartz' weekly tables"""
             url = 'https://www.vgchartz.com/weekly/'
             links_weekly = []
             game = []

             def get_listing(url):
                 html = None
```

```
        links = None
        try:
            r = requests.get(url)
            soup = BeautifulSoup(r.text,'html.parser')
            listing_section = soup.find('p')
            listing_section = soup.find_all('a', attrs={'href': re.compile("^http://"
            links = [link['href'].strip() for link in listing_section]
        except Exception as ex:
            print(str(ex))
        finally:
            return links

    links_weekly = get_listing(url)
    #Select Dates
    try:
        index_weekly = links_weekly.index('http://feeds.feedburner.com/VGChartz')
    except:
        print(index_weekly)
    first_link = links_weekly[index_weekly+1]
    r = requests.get(first_link)
    soup = BeautifulSoup(r.text,'html.parser')
    dates_weekly = soup.find('div',class_="chart_date_selector")
    clean_dates_weekly = dates_weekly.get_text(",", strip=True)
    clean_dates_weekly = clean_dates_weekly.split(',')

    #Cleaning up dates
    clean_dates_weekly = clean_dates_weekly[1:]
    clean_dates_weekly = clean_dates_weekly[:-1]
    cleaner_dates = []
    for weekly_date in clean_dates_weekly[2:]:
        if not weekly_date[2].isdigit():
            weekly_date = weekly_date[0:2] + weekly_date[4:len(weekly_date)]
            cleaner_dates.append(weekly_date)
    return cleaner_dates
dates = get_dates()
```

In [13]:

```
url = 'https://www.vgchartz.com/weekly/'
links_weekly = []
game = []

def get_listing(url):
    html = None
    links = None
    try:
        r = requests.get(url)
        soup = BeautifulSoup(r.text,'html.parser')
        listing_section = soup.find('h1')
        new_links = listing_section.find_all_next("a")
```

```
                 links = [link['href'].strip() for link in new_links]
                 index = links.index('http://www.vgchartz.com/methodology.php')
                 links = links[:index+1]
             except Exception as ex:
                 print(str(ex))
             finally:
                 return links

         links_weekly = get_listing(url)

In [14]: #Find me the values for the top 30 each week
         def get_text(values):
             url = values[0]
             date = values[1]
             locations = ['Global','USA','Europe','UK','Germany','France','Japan']
             for location in locations:
                 if location in url:
                     geo_sales = location
                     break
                 else:
                     continue
             game = []
             r = requests.get(url)
             soup = BeautifulSoup(r.text,'html.parser')
             chart_bodies = soup.find_all("div",id="chart_body")
             for chart_body in chart_bodies:
                 text = chart_body.text
                 start_index = text.find("Pos")
                 if chart_body.text != ' ':
                     text = text.replace(',','')
                     text = text.replace('\n',',')
                     text = text.replace("("," ")
                     text = text.replace(")"," ")
                     text = text.replace("'"," ")
                     text = text.replace(":"," ")
                     game.append(text[start_index:])

             def clean_game(game):
                 clean_games= []
                 position = []
                 game_name = []
                 weekly_sales = []
                 total_sales = []
                 week_num = []
                 for row in game:
                     try:
                         new_list = row.split(',')
                         new_list = [e for e in new_list if e]
```

```
    except:
        continue
    composite_list = [new_list[x:x+5] for x in range(0, len(new_list),5)]
    for item in composite_list[1:]:
        if item[0] == '31':
            break
        else:
            try:
                position.append(item[0])
                game_name.append(item[1])
                weekly_sales.append(item[2])
                total_sales.append(item[3])
                week_num.append(item[4])
            except:
                continue

names = []
console = []
developer = []
genre = []
console_list = ['PS4','NS','3DS','XOne','PC','PSV','Wii','PS3','WiiU','X360',
for text in game_name:
    try:
        words = text.split()
    except:
        continue
    index=[]
    name=[]
    values = []
    for word in words:
        if word in console_list:
            index = words.index(word)
            values = words[index:]
            name = words[:index]
            names.append(' '.join(name))
            if len(values) == 2:
                console.append(values[0])
                developer.append(values[1])
                genre.append(" ")
            elif len(values) == 3:
                console.append(values[0])
                developer.append(values[1])
                genre.append(values[2])
            elif len(values) == 4:
                console.append(values[0])
                developer.append(' '.join(values[1:2]))
                genre.append(values[3])
            elif len(values) == 5:
```

```
                                 console.append(values[0])
                                 developer.append(' '.join(values[1:3]))
                                 genre.append(values[4])
                             else:
                                 console.append(' ')
                                 developer.append(' ')
                                 genre.append(' ')
                 combined_data = pd.concat([position,names,console,developer,genre,weekly_sales
                                          keys=['position','name','console','developer','genre
                 try:
                     return combined_data
                 except:
                     return
             clean_games = clean_game(game)
             return clean_games
```

```
In [15]: index = links_weekly.index('http://www.vgchartz.com/weekly/40832/Japan/') #The last v
         links_weekly = links_weekly[:index+1]
```

```
In [16]: links_dates = []
         for date in dates:
             for i in range(0,7):
                 links_dates.append(date)
```

```
In [17]: games_sales = []
         if not os.path.exists('Data/game_sales.csv') or os.stat('Data/game_sales.csv').st_size
             test = pd.DataFrame()
             test['links'] = links_weekly
             test['dates'] = links_dates[:len(links_weekly)]
             pool=ThreadPool(50)
             games_sales = pool.map(get_text,test.values)
             pool.close()
             pool.join()
             sales_weekly = pd.concat(games_sales)
             sales_weekly.to_csv('Data/game_sales.csv')
         else:
             sales_weekly = pd.read_csv('Data/game_sales.csv')
```

```
In [18]: from datetime import datetime,timedelta
         end_date = []
         start_date = []
         for date in sales_weekly['date']:
             try:
                 dt = datetime.strptime(date, "%d %b %Y")
                 start_date.append(dt.strftime("%Y-%m-%d"))
                 dt = dt + timedelta(days=7)
                 end_date.append(dt.strftime("%Y-%m-%d"))
             except:
                 start_date.append(date)
                 end_date.append(date)
         sales_weekly['start_date'] = start_date
         sales_weekly['end_date'] = end_date
```

```
In [19]: sales_weekly['year'] = pd.to_datetime(sales_weekly["start_date"]).dt.year
```

```
In [20]: list_videogames = sales_weekly[['name','start_date','end_date']].values
         list_videogames = pd.DataFrame(list_videogames)
         list_videogames = list_videogames.dropna()
         list_videogames.to_csv('Data/videogames.csv', sep=',')
```

3.2 Tweets Code

!python '/project/Springboard/Capstone_Project/Data/Twitter_Tool/main.py'

```
In [21]: class TweetCriteria:

             def __init__(self):
                 self.maxTweets = 0

             def setUsername(self, username):
                 self.username = username
                 return self

             def setSince(self, since):
                 self.since = since
                 return self

             def setUntil(self, until):
                 self.until = until
                 return self

             def setQuerySearch(self, querySearch):
                 self.querySearch = querySearch
                 return self

             def setMaxTweets(self, maxTweets):
                 self.maxTweets = maxTweets
                 return self

             def setLang(self, Lang):
                 self.lang = Lang
                 return self

             def setTopTweets(self, topTweets):
                 self.topTweets = topTweets
                 return self
```

```
In [22]: class Tweet:
```

```
             def __init__(self):
                 pass

In [23]: try:
             import http.cookiejar as cookielib
         except ImportError:
             import cookielib

         import urllib,json,re,datetime,sys
         from pyquery import PyQuery

         class TweetManager:

             def __init__(self):
                 pass

             @staticmethod
             def getTweets(tweetCriteria, receiveBuffer=None, bufferLength=100, proxy=None):
                 """Description"""
                 refreshCursor = ''

                 results = []
                 resultsAux = []
                 cookieJar = cookielib.CookieJar()

                 if hasattr(tweetCriteria, 'username') and (tweetCriteria.username.startswith(
                     tweetCriteria.username = tweetCriteria.username[1:-1]

                 active = True

                 while active:
                     json = TweetManager.getJsonReponse(tweetCriteria, refreshCursor, cookieJa
                     if len(json['items_html'].strip()) == 0:
                         break

                     refreshCursor = json['min_position']
                     scrapedTweets = PyQuery(json['items_html'])
                     #Remove incomplete tweets withheld by Twitter Guidelines
                     scrapedTweets.remove('div.withheld-tweet')
                     tweets = scrapedTweets('div.js-stream-tweet')

                     if len(tweets) == 0:
                         break

                     for tweetHTML in tweets:
                         tweetPQ = PyQuery(tweetHTML)
                         tweet = Tweet()
```

```
                usernameTweet = tweetPQ("span:first.username.u-dir b").text()
                txt = re.sub(r"\s+", " ", tweetPQ("p.js-tweet-text").text().replace('
                retweets = int(tweetPQ("span.ProfileTweet-action--retweet span.Profil
                favorites = int(tweetPQ("span.ProfileTweet-action--favorite span.Prof
                dateSec = int(tweetPQ("small.time span.js-short-timestamp").attr("data
                id = tweetPQ.attr("data-tweet-id")
                permalink = tweetPQ.attr("data-permalink-path")

                geo = ''
                geoSpan = tweetPQ('span.Tweet-geo')
                if len(geoSpan) > 0:
                    geo = geoSpan.attr('title')

                tweet.id = id
                tweet.permalink = 'https://twitter.com' + permalink
                tweet.username = usernameTweet
                tweet.text = txt
                tweet.date = datetime.datetime.fromtimestamp(dateSec)
                tweet.retweets = retweets
                tweet.favorites = favorites
                tweet.mentions = " ".join(re.compile('(@\\w*)').findall(tweet.text))
                tweet.hashtags = " ".join(re.compile('(#\\w*)').findall(tweet.text))
                tweet.geo = geo

                results.append(tweet)
                resultsAux.append(tweet)

                if receiveBuffer and len(resultsAux) >= bufferLength:
                    receiveBuffer(resultsAux)
                    resultsAux = []

                if tweetCriteria.maxTweets > 0 and len(results) >= tweetCriteria.maxT
                    active = False
                    break


        if receiveBuffer and len(resultsAux) > 0:
            receiveBuffer(resultsAux)

        return results

    @staticmethod
    def getJsonReponse(tweetCriteria, refreshCursor, cookieJar, proxy):
        """Description"""
        url = "https://twitter.com/i/search/timeline?f=tweets&q=%s&src=typd&max_posit

        urlGetData = ''
```

```
                 if hasattr(tweetCriteria, 'username'):
                     urlGetData += ' from:' + tweetCriteria.username

                 if hasattr(tweetCriteria, 'querySearch'):
                     urlGetData += ' ' + tweetCriteria.querySearch

                 if hasattr(tweetCriteria, 'near'):
                     urlGetData += "&near:" + tweetCriteria.near + " within:" + tweetCriteria.

                 if hasattr(tweetCriteria, 'since'):
                     urlGetData += ' since:' + tweetCriteria.since

                 if hasattr(tweetCriteria, 'until'):
                     urlGetData += ' until:' + tweetCriteria.until

                 if hasattr(tweetCriteria, 'topTweets'):
                     if tweetCriteria.topTweets:
                         url = "https://twitter.com/i/search/timeline?q=%s&src=typd&max_positi
                 url = url % (urllib.parse.quote(urlGetData), refreshCursor)

                 headers = [('Host', "twitter.com"),
                     ('User-Agent', "Mozilla/5.0 (Windows NT 6.1; Win64; x64)"),
                     ('Accept', "application/json, text/javascript, */*; q=0.01"),
                     ('Accept-Language', "de,en-US;q=0.7,en;q=0.3"),
                     ('X-Requested-With', "XMLHttpRequest"),
                     ('Referer', url),
                     ('Connection', "keep-alive")]
                 if proxy:
                     opener = urllib.request.build_opener(urllib.request.ProxyHandler({'http':
                 else:
                     opener = urllib.request.build_opener(urllib.request.HTTPCookieProcessor(c
                 opener.addheaders = headers

                 try:
                     response = opener.open(url)
                     jsonResponse = response.read()
                 except:
                     print ("Twitter weird response. Try to see on browser: https://twitter.co
                     sys.exit()
                     return
             dataJson = json.loads(jsonResponse)

             return dataJson
```

```
In [24]: def finding_tweets(videogame):
             """Description"""
             name = []
             start_date = []
             tweet_date = []
             tweet_text = []
             assert isinstance(videogame[0],str)
             assert isinstance(videogame[1],str)
             assert isinstance(videogame[2],str)
             try:
                 tweetCriteria = TweetCriteria().setQuerySearch(videogame[0]).setSince(videogam
                 tweets = TweetManager.getTweets(tweetCriteria)
                 for tweet in tweets:
                     tweet_date.append(tweet.date)
                     tweet_text.append(tweet.text)
             except:
                 print(videogame[0],videogame[1],videogame[2], 'empty')
             df = pd.DataFrame(np.column_stack((tweet_date,tweet_text)))
             df['Name']= videogame[0]
             df['start_date'] = videogame[1]
             df['end_date'] = videogame[2]
             return df

In [25]: from multiprocessing.pool import ThreadPool
         import pickle
         total_list = []
         n=10
         try:
             list_videogames = list_videogames.values
         except:
             print('already array')
         if not os.path.exists('Data/tweets.csv') or os.stat('Data/tweets.csv').st_size == 0:
             chunks = [list_videogames[i:i + n] for i in range(0, len(list_videogames), n)]
             for chunk in chunks:
                 text_results = []
                 pool=ThreadPool(6000)
                 text_results = pool.map(finding_tweets,chunk)
                 pool.close()
                 pool.join()
                 total_list = total_list + text_results
                 with open('objs.pkl', 'wb') as f:
                     pickle.dump([total_list], f)
             new_results = pd.DataFrame(np.vstack(total_list),columns=['date','text','name','s
             new_results.to_csv('/project/Springboard/Capstone_Project/Data/tweets.csv')
         else:
             tweets_df = pd.read_csv('Data/tweets.csv')
             tweets_df = tweets_df.drop(tweets_df.columns[0], axis=1)

In [26]: tweets_df["date"] = pd.to_datetime(tweets_df["date"])
         tweets_df['week'] = tweets_df["date"].dt.week
         tweets_df['year'] = tweets_df["date"].dt.year
         tweets_week = pd.DataFrame({'count' : tweets_df.groupby(['name','week']).size()}).rese
```

```
In [27]: sales_weekly['week'] = pd.to_datetime(sales_weekly["start_date"]).dt.week
         sales_weekly['month'] = pd.to_datetime(sales_weekly["start_date"]).dt.month
         sales_weekly['week_of_day'] = pd.to_datetime(sales_weekly["start_date"]).dt.weekday
         sales_weekly['year'] = pd.to_datetime(sales_weekly["start_date"]).dt.year
```

```
In [28]: columns = ['position','total_sales','week_num','weekly_sales']
         for col in columns:
             sales_weekly[col] = pd.to_numeric(sales_weekly[col],downcast='integer')
         sales_weekly.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21893 entries, 0 to 21892
Data columns (total 25 columns):
Unnamed: 0          21893 non-null int64
Unnamed: 0.1        21893 non-null int64
Unnamed: 0.1.1      21893 non-null int64
Unnamed: 0.1.1.1    21893 non-null int64
console             21893 non-null object
date                21893 non-null object
developer           21893 non-null object
genre               21893 non-null object
location            21893 non-null object
name                21893 non-null object
position            21893 non-null int8
total_sales         21893 non-null int32
week_num            21893 non-null int16
weekly_sales        21893 non-null int32
start_date          21893 non-null object
end_date            21893 non-null object
year                21893 non-null int64
month               21893 non-null int64
week                21893 non-null int64
hour                21893 non-null int64
weekday             21893 non-null int64
total_tweets        21893 non-null float64
year_of_release     21893 non-null object
time_difference     21893 non-null float64
week_of_day         21893 non-null int64
dtypes: float64(2), int16(1), int32(2), int64(10), int8(1), object(9)
memory usage: 3.7+ MB
```

```
In [29]: sales_weekly['total_tweets'] = 0.0
         for index,tweet in tweets_week.iterrows():
             sales_weekly.loc[(sales_weekly.name == tweet[0]) & (sales_weekly.week == tweet[1]
```

```
In [30]: sales_weekly = sales_weekly.dropna()
```

3.2.1 Now, the tweets are found for the full videogame set

```
In [31]: sales['Year_of_Release'] = pd.to_datetime(sales['Year_of_Release'],errors='coerce')
         sales['Year'] = sales['Year_of_Release'].dt.year
         new_sales = sales.dropna(subset=['User_Score'])
         dict_dates = {}
         for index,row in new_sales.iterrows():
             try:
                 if row['Name'] not in dict_dates and int(row['Year']) >= 2006:
                     date_list = pd.date_range(row['Year_of_Release'],pd.datetime.today(),freq
                     dict_dates[row['Name']] = date_list
             except:
                 continue
```

```
In [32]: sales_tweets = []
         for key,values in dict_dates.items():
             for value in values:
                 sales_tweets.append([key.strip(' '),str(value.date()),str((value+datetime.tim
```

```
In [33]: #sales dataframe results
         total_results = []
         n=100
         chunks = [sales_tweets[i:i + n] for i in range(0, len(sales_tweets), n)]
         if not os.path.exists('Data/sales_tweets.csv') or os.stat('Data/sales_tweets.csv').st
             for chunk in chunks:
                 text_results = []
                 pool=ThreadPool(16000)
                 text_results = pool.map(finding_tweets,chunk)
                 pool.close()
                 pool.join()
                 total_results = total_results + text_results
                 with open('objs.pkl', 'wb') as f:
                         pickle.dump([total_results], f)
                 full_results = pd.DataFrame(np.vstack(total_results),columns=['date','text','
                 #full_results.to_csv('/project/Springboard/Capstone_Project/Data/sales_tweets
         else:
             #import pickle
             #full_results = pickle.load(open( "objs.pkl", "rb" ))
             full_results = pd.read_csv('Data/sales_tweets.csv',names=['date','text','name','s
```

```
In [34]: grouped = full_results.groupby('name').size().reset_index()
         for index, row in grouped.iterrows():
             close_matches = difflib.get_close_matches(row['name'],sales['Name'])
             for close_match in close_matches:
                 sales.loc[sales['Name']==close_match,'Total_Tweets'] = row[0]
```

```
In [35]: file = '/project/twitter.xlsx'
         data = list(pd.read_excel(file,header=1))
         openmapquest=str(data[5])
```

```
def country_to_continents(country):
    state_continents = {"AD":"Europe","AE":"Asia","AF":"Asia","AG":"North America",
    "AI":"North America","AL":"Europe","AM":"Asia","AN":"North America","AO":"Africa"
    "AU":"Australia","AW":"North America","AZ":"Asia","BA":"Europe","BB":"North Ameri
    "BS":"North America","BT":"Asia","BW":"Africa","BY":"Europe","BZ":"North America"
    "CO":"South America","CR":"North America","CU":"North America","CV":"Africa","CX"
    "EH":"Africa","ER":"Africa","ES":"Europe","ET":"Africa","FI":"Europe","FJ":"Austr
    "GI":"Europe","GL":"North America","GM":"Africa","GN":"Africa","GP":"North Americ
    "HT":"North America","HU":"Europe","ID":"Asia","IE":"Europe","IL":"Asia","IM":"Eu
    "KI":"Australia","KM":"Africa","KN":"North America","KP":"Asia","KR":"Asia","KW":
    "LY":"Africa","MA":"Africa","MC":"Europe","MD":"Europe","ME":"Europe","MG":"Afric
    "MV":"Asia","MW":"Africa","MX":"North America","MY":"Asia","MZ":"Africa","NA":"Af
    "OM":"Asia","PA":"North America","PE":"South America","PF":"Australia","PG":"Aust
    "RE":"Africa","RO":"Europe","RS":"Europe","RU":"Europe","RW":"Africa","SA":"Asia"
    "SO":"Africa","SR":"South America","ST":"Africa","SV":"North America","SY":"Asia"
    "TR":"Asia","TT":"North America","TV":"Australia","TW":"Asia","TZ":"Africa","UA":
    "UZ":"Asia","VC":"North America","VE":"South America","VG":"North America","VI":"

    for k,v in state_continents.items():
        if country == k:
            return v
    return None

if not os.path.exists('Data/publisher_location.csv') or os.stat('Data/publisher_locat
    def location_lookup(text):
        geo = geocoder.mapquest(text,key=openmapquest)
        if geo != None:
            lat,lng = geo.latlng
            return text, geo.country,country_to_continents(geo.country),lat,lng
        else:
            return text,None,None,None,None

    unique_values = sales.Publisher
    #cut the pieces up into sublists of 100 values
    n=100
    chunks = [unique_values[i:i + n] for i in range(0, len(unique_values), n)]
    for x in chunks:
        text_results = []
        pool=ThreadPool(1000)
        text_results = pool.map(location_lookup,x)
        pool.close()
        pool.join()
        loc_publication = loc_publication + text_results
        with open('objs_loc.pkl', 'wb') as f:
            pickle.dump([loc_publication], f)
    flatten = [item for sublist in loc_publication for item in sublist]
    pd.DataFrame(flatten).to_csv('publisher_location.csv', sep=',')
else:
```

```
             df = pd.read_csv('Data/publisher_location.csv',skiprows=1,names=['name','country'
             degrees = df[['latitude','longitude']]
             sales['Publisher_Country'] = df['country']
             sales['Publisher_Continent'] = df['continent']
```

```
In [36]: sales['Publisher'] = sales['Publisher'].astype(str)
         df['name'] = df['name'].astype(str)
         sales['Publisher'] = [row.replace(';','') for row in sales['Publisher']]
         for index, row in df.iterrows():
             close_matches = difflib.get_close_matches(row['name'],sales['Publisher'])
             sales.loc[sales['Publisher']==close_matches[0],'Publisher_Lat'] = row['latitude']
             sales.loc[sales['Publisher']==close_matches[0],'Publisher_Long'] = row['longitude
```

The unit sales are in millions. As a result, I mulitplied it by 10^6 and divided by the population to adjust for region size. I also calculated the sales in terms of sales per thousand people.

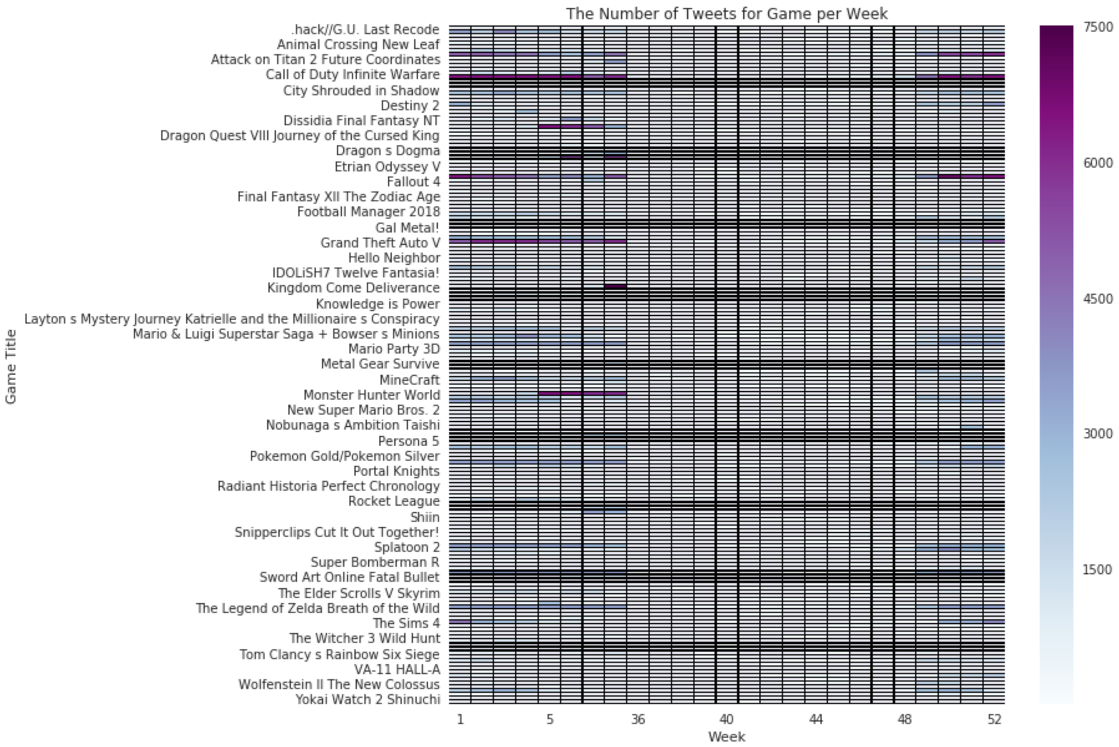
```
In [37]: USA = 326067398
         UK = 66454759
         Germany = 82239223
         France =  65156451
         Japan = 127275698
         Europe = UK + Germany + France
         Global_pop = USA + Europe + Japan

         sales['JP_Sales'] = [float(x)*1000000.0*1000.0/float(Japan) for x in sales['JP_Sales']
         sales['EU_Sales'] = [float(x)*1000000.0*1000.0/float(Europe) for x in sales['EU_Sales
         sales['NA_Sales'] = [float(x)*1000000.0*1000.0/float(USA) for x in sales['NA_Sales']]
         sales['Global_Sales'] = [float(x)*1000000.0*1000.0/float(Global_pop) for x in sales['
         columns = ['Chartz_Score','User_Score','Critic_Score','Other_Sales']
         for col in columns:
             sales[col] = pd.to_numeric(sales[col],downcast='integer')
         sales['Year'] = pd.to_datetime(sales['Year_of_Release'],dayfirst=True,errors='coerce'
         sales.head()
```

```
Out[37]:    Unnamed: 0                      Name   Platform                 Publisher  \
         0           0    Super Mario Bros.       Nintendo              Nintendo EAD
         1           1    Wii Sports Resort       Nintendo              Nintendo EAD
         2           2              Tetris       Nintendo    Bullet Proof Software
         3           3            Wii Play       Nintendo              Nintendo EAD
         4           4           Duck Hunt       Nintendo             Nintendo R&D1

            Chartz_Score  User_Score  Critic_Score  Global_Sales    NA_Sales   EU_Sales  \
         0           NaN        10.0           NaN     60.312336   89.184016  16.740672
         1           8.8         8.0           8.8     49.311030   47.873538  51.344296
         2           NaN         NaN           NaN     45.354157   71.150934  10.568134
         3           5.3         5.9           4.5     43.345744   42.813235  42.927199
         4           NaN         NaN           NaN     42.431467   82.590287   2.945984

             JP_Sales  Other_Sales Year_of_Release     Genre    Year  Total_Tweets  \
         0  53.505894         0.77      1985-10-18  Platform  1985.0           NaN
         1  25.849397         3.02      2009-07-26    Sports  2009.0      115679.0
         2  33.156369         0.58      1989-06-01    Puzzle  1989.0          19.0
         3  23.020891         2.85      2007-02-12       NaN  2007.0       72671.0
         4   2.199949         0.47      1985-10-15   Shooter  1985.0           NaN

           Publisher_Country Publisher_Continent  Publisher_Lat  Publisher_Long
         0                MX       North America       14.88102       -92.27582
         1                GB              Europe       14.88102       -92.27582
         2                DE              Europe       51.47722        -0.06931
         3                AU           Australia       14.88102       -92.27582
         4                RU              Europe       49.00519        11.92915
```

3.3 The trends

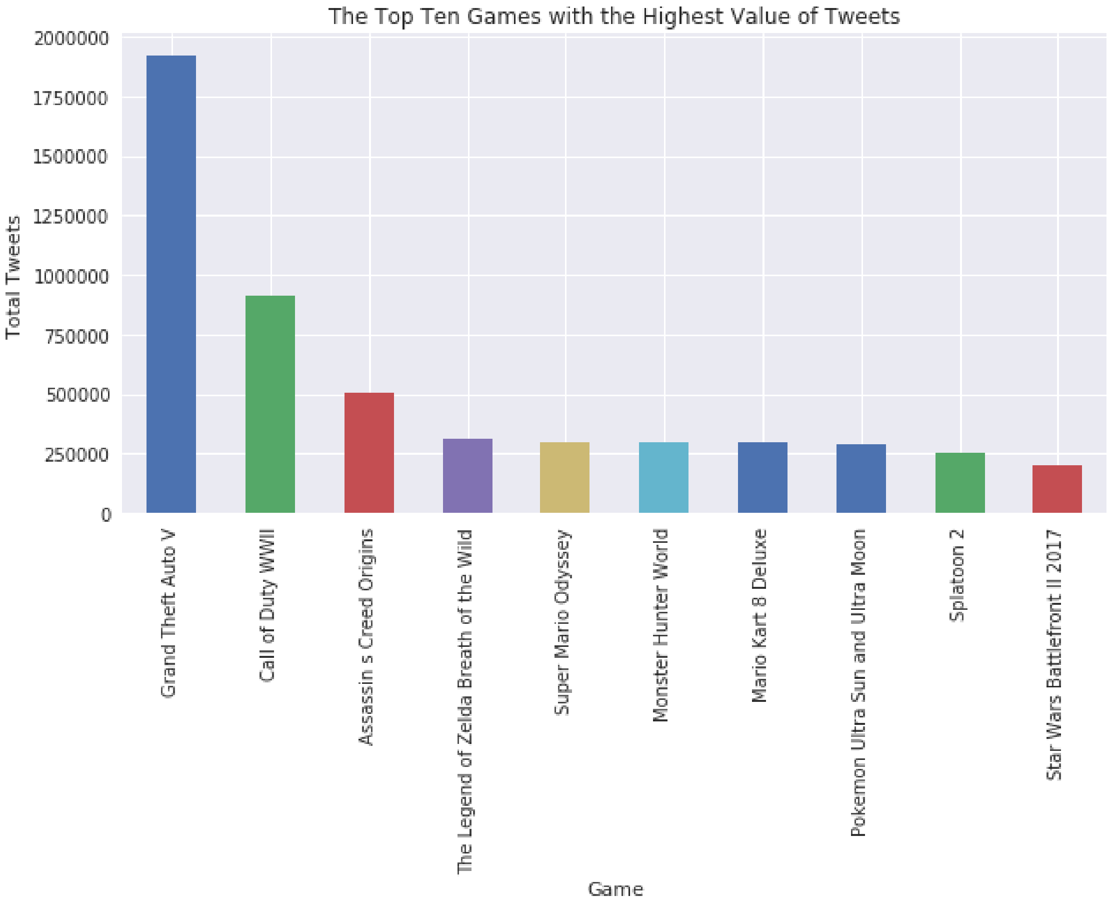
I first started off by looking at the sales_weekly data that was produced from the twitter search.

```
In [38]: #
         tweets_week = pd.DataFrame({'count' : tweets_df.groupby(['name','week']).size()}).rese
         df_heat = tweets_week.pivot("name","week",'count')
         sns.set(font_scale=1.0)
         plt.figure(figsize = (10,10))
         sns.heatmap(df_heat,cmap="BuPu",linewidths=0.2, linecolor='black',xticklabels=4)
         plt.xlabel('Week')
         plt.ylabel('Game Title')
         plt.title('The Number of Tweets for Game per Week')

         plt.show()
```



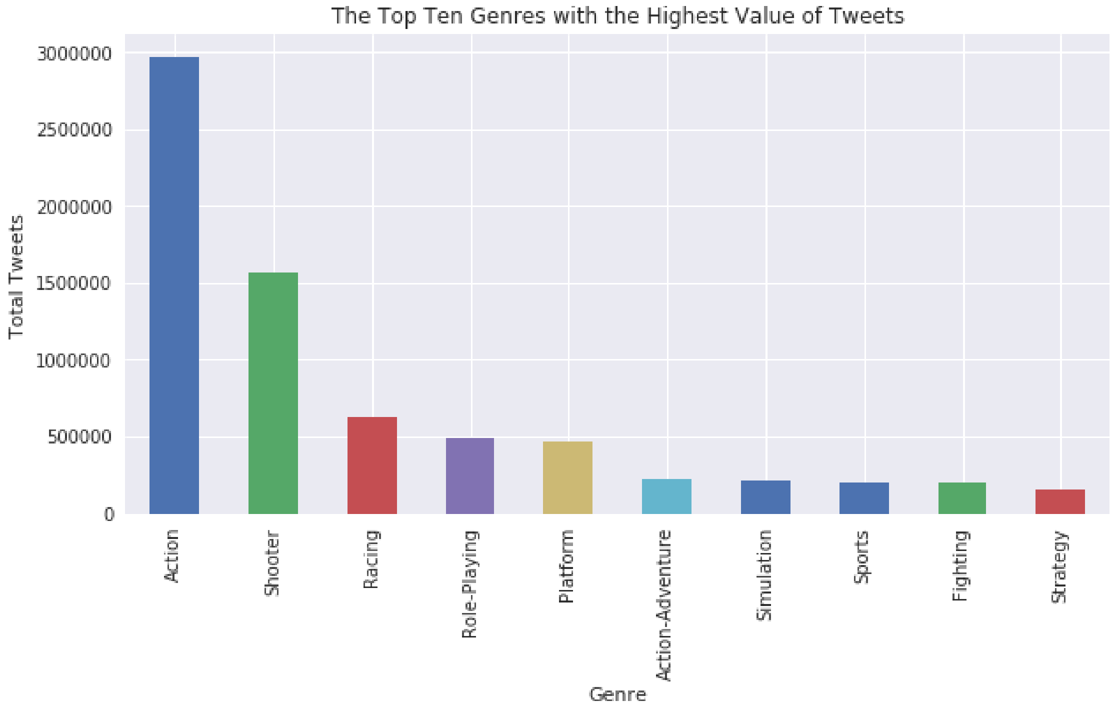
The tweet number from week 48 from the previous year to the following year should not be that surprising. Winter and Spring are during this time where the weather gets cooler and thus more individuals tend to be inside. However, the lack of tweets between weeks 36 and 48 might be contributed to the lack of intensive computing power available at the moment. Out of all the tweets, who had the most tweets?

```
In [39]: sns.set()
         plt.figure(figsize = (10,5))
         sales_weekly.groupby('name').sum()['total_tweets'].nlargest(10).plot(kind='bar')
         plt.xticks(rotation=90)
         plt.xlabel('Game')
         plt.ylabel('Total Tweets')
         plt.title('The Top Ten Games with the Highest Value of Tweets')
         plt.show()
```



Truth to be told. It was in fact Grand Theft Auto V! I have listed its characters below. It is interesting though because Grand Theft Auto V came out in 2013 whereas Call of Duty WWII the second highest number of tweets was published in 2017. This suggests that the number of tweets for Call of Duty happened at a much quicker rate than Grand Theft Auto. Does this mean that the sales are actually better? What does this suggest about its popularity?

```
In [41]: sns.set()
         plt.figure(figsize = (10,5))
         sales_weekly.groupby('genre').sum()['total_tweets'].nlargest(10).plot(kind='bar')
         plt.xticks(rotation=90)
         plt.xlabel('Genre')
         plt.ylabel('Total Tweets')
         plt.title('The Top Ten Genres with the Highest Value of Tweets')
         plt.show()
```



The genre with the most tweets is action and the second highest is shooter. What is the difference between these two genres? Action would be more like God of War while shooter genre includes games like the Call of Duty series. Notice that in the dataset, the majority of games are action or shooter. Therefore, the tweets are skewed towards these two genres.

```
In [42]: import pandas as pd
         import numpy as np
         import matplotlib
         import cufflinks as cf
         import plotly
         import plotly.offline as py
         import plotly.graph_objs as go
         from plotly.graph_objs import *

         cf.go_offline() # required to use plotly offline (no account required).
         py.init_notebook_mode() #
         trace = go.Pie(
             labels=sales_weekly.groupby(["genre"]).count().index,
                                values = sales_weekly.groupby(["genre"]).count().sort_values(a
         data = [trace]
         py.iplot(data, filename = 'basic_table')
```

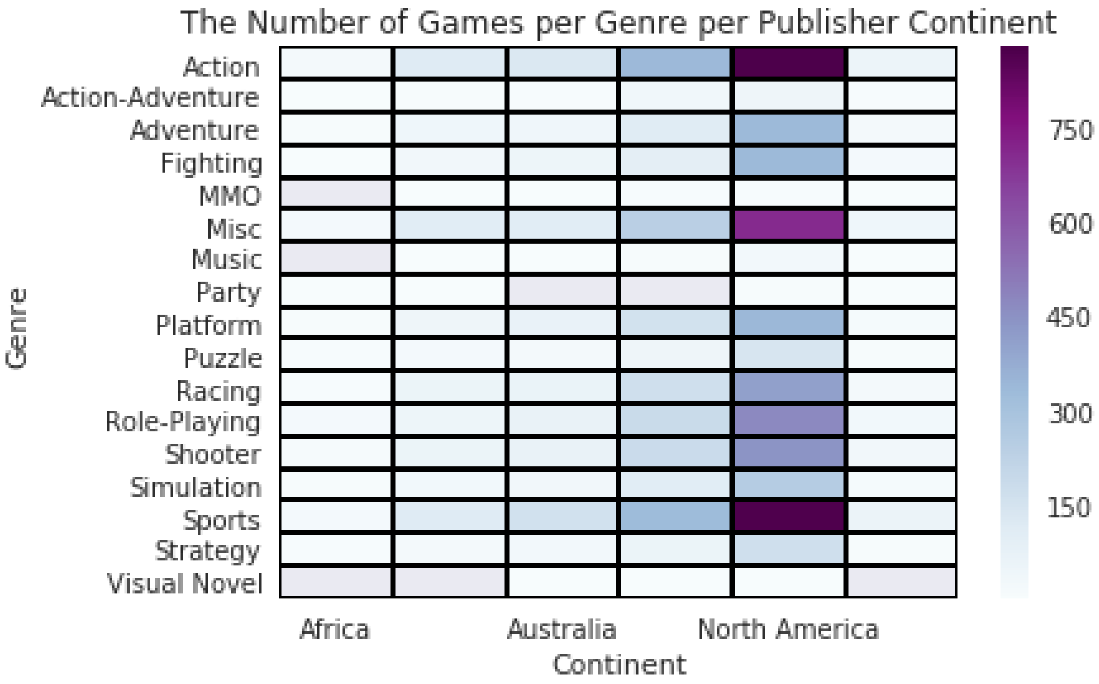
The majority of these games are in fact produced in Europe and in North America as shown in the interactive map below.

```
In [43]: import cufflinks as cf
         import plotly
         import plotly.offline as py
         import colorlover as cl
         from plotly.graph_objs import *

         cf.go_offline() # required to use plotly offline (no account required).
         py.init_notebook_mode()

         trace1 = {
           "hoverinfo": "lat+lon",
           "lat": sales['Publisher_Lat'],
             "line": {
             "color": "rgba(255, 153, 51, 1.0)",
             "dash": "solid",
             "width": 1.3
           },
           "locationmode": "USA-states",
           "lon": sales['Publisher_Long'],
             "marker": {
             "color": "rgba(44, 160, 44, 0.79)",
             "line": {"color": "rgba(44, 160, 44, 0.71)"}
           },
           "mode": "markers",
           "name": "y",
           "text": "",
           "type": "scattergeo",
           "uid": "c68755"
         }
         data = Data([trace1])
         layout = {
           "autosize": True,
           "geo": {
             "center": {
               "lat": 0,
               "lon": 0
             },
             "projection": {
               "rotation": {"lon": 0},
               "scale": 1,
               "type": "equirectangular"
             },
             "scope": "world"
           },
           "legend": {
             "bgcolor": "#F5F6F9",
             "font": {"color": "#4D5663"}
           },
           "paper_bgcolor": "#F5F6F9",
           "plot_bgcolor": "#F5F6F9",
           "titlefont": {"color": "#4D5663"}
         }
         fig = dict(data=data, layout=layout)

         py.iplot(fig)
```

What is in interesting though is that North America produces the most action, sports, and misc genre games. This result can be seen in the below's image. The darker the value, the higher it is. The continent in this heatmap refers to the continent of the publisher's location.

```
In [44]: count = pd.DataFrame({'Count' : sales.groupby(['Genre','Publisher_Continent']).size()
         df_heat2 = count.pivot("Genre","Publisher_Continent",'Count')
         sns.set()
         sns.heatmap(df_heat2,cmap="BuPu",linewidths=1, linecolor='black',xticklabels=2)
         plt.xlabel('Continent')
         plt.ylabel('Genre')
         plt.title('The Number of Games per Genre per Publisher Continent')
```

```
Out[44]: Text(0.5,1,'The Number of Games per Genre per Publisher Continent')
```



```
In [45]: import cufflinks as cf
         import plotly
         import plotly.offline as py
         from plotly.graph_objs import *

         cf.go_offline() # required to use plotly offline (no account required).
         py.init_notebook_mode() #

         layout = go.Layout(
             xaxis=dict(
                 title='Publishers',
             ),
             yaxis=dict(
                 title='Number of Videogames',
             ),
         )

         sales['Publisher'].value_counts().nlargest(20).iplot(kind='bar', color='#1F77B4')

In [46]: import pandas as pd
         import numpy as np
         import matplotlib
         import cufflinks as cf
         import plotly
         import plotly.offline as py
         from plotly.graph_objs import *

         cf.go_offline() # required to use plotly offline (no account required).
         py.init_notebook_mode() #

         layout = go.Layout(
             yaxis=dict(
                 title='Sales',
                 zeroline=False
             ),
         )
         sales[['NA_Sales','EU_Sales','JP_Sales','Global_Sales']].iplot(kind='box', boxpoints =

In [47]: df = sales[(sales.NA_Sales > 60)].dropna(how='all',axis=1).drop(columns=['Unnamed: 0']
         df.head(5)

Out[47]:                        Name    Platform               Publisher  User_Score  \
         0    Super Mario Bros.       Nintendo             Nintendo EAD        10.0
         2               Tetris       Nintendo    Bullet Proof Software         NaN
         4            Duck Hunt       Nintendo            Nintendo R&D1         NaN
         25          Wii Sports       Nintendo             Nintendo EAD         7.7

             Global_Sales    NA_Sales    EU_Sales   JP_Sales  Other_Sales  \
         0      60.312336   89.184016   16.740672  53.505894         0.77
         2      45.354157   71.150934   10.568134  33.156369         0.58
         4      42.431467   82.590287    2.945984   2.199949         0.47
```

```
25    123.877101  126.844941  135.655559  29.620737         8.51

   Year_of_Release     Genre    Year  Total_Tweets Publisher_Country  \
0       1985-10-18  Platform  1985.0           NaN                MX
2       1989-06-01    Puzzle  1989.0          19.0                DE
4       1985-10-15   Shooter  1985.0           NaN                RU
25      2006-11-19    Sports  2006.0      115679.0                DE

   Publisher_Continent  Publisher_Lat  Publisher_Long
0        North America       14.88102       -92.27582
2               Europe       51.47722        -0.06931
4               Europe       49.00519        11.92915
25              Europe       14.88102       -92.27582
```

Now, we are going to look at these same sales but split up by publisher location and genre.

```
In [48]: summary_sales = sales.groupby(['Genre','Publisher_Continent'])[['NA_Sales','EU_Sales'
         summary_sales = summary_sales.swaplevel('Publisher_Continent', 'Genre', axis=0)
         mi=summary_sales.columns
         ind = pd.Index([e[0] + e[1] for e in mi.tolist()])
         summary_sales.columns = ind
```

```
In [49]: import cufflinks as cf
         import plotly
         import plotly.offline as py
         from plotly.graph_objs import *

         cf.go_offline() # required to use plotly offline (no account required).
         py.init_notebook_mode() #

         data = pd.DataFrame(summary_sales.reset_index(),columns=['Genre','Publisher_Continent

         trace01 = go.Bar(x=data[data['Genre']=='Action'].loc[:,'Publisher_Continent'].values,
                       y = data[data['Genre']=='Action'].loc[:,'NA'].values,name='NA')
         trace02 = go.Bar(x=data[data['Genre']=='Action'].loc[:,'Publisher_Continent'].values,
                       y = data[data['Genre']=='Action'].loc[:,'JP'].values,name='JP')
         trace03 = go.Bar(x=data[data['Genre']=='Action'].loc[:,'Publisher_Continent'].values,
                       y = data[data['Genre']=='Action'].loc[:,'EU'].values,name='EU')
         trace11 = go.Bar(x=data[data['Genre']=='Adventure'].loc[:,'Publisher_Continent'].valu
                       y = data[data['Genre']=='Adventure'].loc[:,'NA'].values,name='NA')
         trace12 = go.Bar(x=data[data['Genre']=='Adventure'].loc[:,'Publisher_Continent'].valu
                       y = data[data['Genre']=='Adventure'].loc[:,'JP'].values,name='JP')
         trace13 = go.Bar(x=data[data['Genre']=='Adventure'].loc[:,'Publisher_Continent'].valu
                       y = data[data['Genre']=='Adventure'].loc[:,'EU'].values,name='EU')
         trace21 = go.Bar(x=data[data['Genre']=='Fighting'].loc[:,'Publisher_Continent'].value
                       y = data[data['Genre']=='Fighting'].loc[:,'NA'].values,name='NA')
         trace22 = go.Bar(x=data[data['Genre']=='Fighting'].loc[:,'Publisher_Continent'].value
                       y = data[data['Genre']=='Fighting'].loc[:,'JP'].values,name='JP')
```

```
trace23 = go.Bar(x=data[data['Genre']=='Fighting'].loc[:,'Publisher_Continent'].values
               y = data[data['Genre']=='Fighting'].loc[:,'EU'].values,name='EU')
trace31 = go.Bar(x=data[data['Genre']=='Misc']['Publisher_Continent'].values,
                y = data[data['Genre']=='Misc'].loc[:,'NA'].values,name='NA')
trace32 = go.Bar(x=data[data['Genre']=='Misc']['Publisher_Continent'].values,
                y = data[data['Genre']=='Misc'].loc[:,'JP'].values,name='JP')
trace33 = go.Bar(x=data[data['Genre']=='Misc']['Publisher_Continent'].values,
                y = data[data['Genre']=='Misc'].loc[:,'EU'].values,name='EU')
trace41 = go.Bar(x=data[data['Genre']=='Platform']['Publisher_Continent'].values,
                y = data[data['Genre']=='Platform'].loc[:,'NA'].values,name='NA')
trace42 = go.Bar(x=data[data['Genre']=='Platform']['Publisher_Continent'].values,
                y = data[data['Genre']=='Platform'].loc[:,'JP'].values,name='JP')
trace43 = go.Bar(x=data[data['Genre']=='Platform']['Publisher_Continent'].values,
                y = data[data['Genre']=='Platform'].loc[:,'EU'].values,name='EU')
trace51 = go.Bar(x=data[data['Genre']=='Puzzle']['Publisher_Continent'].values,
                y = data[data['Genre']=='Puzzle'].loc[:,'NA'].values,name='NA')
trace52 = go.Bar(x=data[data['Genre']=='Puzzle']['Publisher_Continent'].values,
                y = data[data['Genre']=='Puzzle'].loc[:,'JP'].values,name='JP')
trace53 = go.Bar(x=data[data['Genre']=='Puzzle']['Publisher_Continent'].values,
                y = data[data['Genre']=='Puzzle'].loc[:,'EU'].values,name='EU')
trace61 = go.Bar(x=data[data['Genre']=='Racing']['Publisher_Continent'].values,
                y = data[data['Genre']=='Racing'].loc[:,'NA'].values,name='NA')
trace62 = go.Bar(x=data[data['Genre']=='Racing']['Publisher_Continent'].values,
                y = data[data['Genre']=='Racing'].loc[:,'JP'].values,name='JP')
trace63 = go.Bar(x=data[data['Genre']=='Racing']['Publisher_Continent'].values,
                y = data[data['Genre']=='Racing'].loc[:,'EU'].values,name='EU')
trace71 = go.Bar(x=data[data['Genre']=='Role-Playing']['Publisher_Continent'].values,
                y = data[data['Genre']=='Role-Playing'].loc[:,'NA'].values,name='NA')
trace72 = go.Bar(x=data[data['Genre']=='Role-Playing']['Publisher_Continent'].values,
                y = data[data['Genre']=='Role-Playing'].loc[:,'JP'].values,name='JP')
trace73 = go.Bar(x=data[data['Genre']=='Role-Playing']['Publisher_Continent'].values,
                y = data[data['Genre']=='Role-Playing'].loc[:,'EU'].values,name='EU')
trace81 = go.Bar(x=data[data['Genre']=='Shooter']['Publisher_Continent'].values,
                y = data[data['Genre']=='Shooter'].loc[:,'NA'].values,name='NA')
trace82 = go.Bar(x=data[data['Genre']=='Shooter']['Publisher_Continent'].values,
                y = data[data['Genre']=='Shooter'].loc[:,'JP'].values,name='JP')
trace83 = go.Bar(x=data[data['Genre']=='Shooter']['Publisher_Continent'].values,
                y = data[data['Genre']=='Shooter'].loc[:,'EU'].values,name='EU')
trace91 = go.Bar(x=data[data['Genre']=='Simulation']['Publisher_Continent'].values,
                y = data[data['Genre']=='Simulation'].loc[:,'NA'].values,name='NA')
trace92 = go.Bar(x=data[data['Genre']=='Simulation']['Publisher_Continent'].values,
                y = data[data['Genre']=='Simulation'].loc[:,'JP'].values,name='JP')
trace93 = go.Bar(x=data[data['Genre']=='Simulation']['Publisher_Continent'].values,
                y = data[data['Genre']=='Simulation'].loc[:,'EU'].values,name='EU')
trace101 = go.Bar(x=data[data['Genre']=='Sports']['Publisher_Continent'].values,
                y = data[data['Genre']=='Sports'].loc[:,'NA'].values,name='NA')
trace102 = go.Bar(x=data[data['Genre']=='Sports']['Publisher_Continent'].values,
                y = data[data['Genre']=='Sports'].loc[:,'JP'].values,name='JP')
```

```
trace103 = go.Bar(x=data[data['Genre']=='Sports']['Publisher_Continent'].values,
                y = data[data['Genre']=='Sports'].loc[:,'EU'].values,name='EU')
trace111 = go.Bar(x=data[data['Genre']=='Strategy']['Publisher_Continent'].values,
                y = data[data['Genre']=='Strategy'].loc[:,'NA'].values,name='NA')
trace112 = go.Bar(x=data[data['Genre']=='Strategy']['Publisher_Continent'].values,
                y = data[data['Genre']=='Strategy'].loc[:,'JP'].values,name='JP')
trace113 = go.Bar(x=data[data['Genre']=='Strategy']['Publisher_Continent'].values,
                y = data[data['Genre']=='Strategy'].loc[:,'EU'].values,name='EU')

data2=([trace01,trace02,trace03,trace11,trace12,trace13,trace21,trace22,trace23,
       trace31,trace32,trace33,trace41,trace42,trace43,trace51,trace52,trace53,
       trace61,trace62,trace63,trace71,trace72,trace73,trace81,trace82,trace83,
       trace91,trace92,trace93,trace101,trace102,trace103,trace111,trace112,trace113]

# make figure
updatemenus = list([
    dict(active=1,
         buttons=list([
            dict(label = 'Action',
                 method = 'update',
                 args = [{'visible': [True, True, True, False,False,False,False,False
                                      False,False,False,False,False,False,False,False
                                      False,False,False,False,False,False,False,False
                                      False,False,False,False,False,False,False,False
                         {'title': 'Action'}]),
            dict(label = 'Adventure',
                 method = 'update',
                 args = [{'visible': [ False,False,False,True, True, True,False,False
                                      False,False,False,False,False,False,False,False
                                      False,False,False,False,False,False,False,False
                                      False,False,False,False,False,False,False,False
                         {'title': 'Adventure'}]),
            dict(label = 'Fighting',
                 method = 'update',
                 args = [{'visible': [ False,False,False,False,False,False,True, True
                                      False,False,False,False,False,False,False,False
                                      False,False,False,False,False,False,False,False
                                      False,False,False,False,False,False,False,False
                         {'title': 'Fighting'}]),
            dict(label = 'Misc',
                 method = 'update',
                 args = [{'visible': [False,False,False,False,False,False,False,False
                                      True, True, True, False,False,False,False,False
                                     False,False,False,False,False,False,False,False,
                                      False,False,False,False,False,False,False,False
                         {'title': 'Misc'}]),
            dict(label = 'Platform',
                 method = 'update',
                 args = [{'visible': [False,False,False,False,False,False,False,False
                                     False,False,False,True, True, True, False,False,
                                     False,False,False,False,False,False,False,False,
                                      False,False,False,False,False,False,False,False
                         {'title': 'Platform'}]),
            dict(label = 'Puzzle',
                 method = 'update',
                  args = [{'visible': [False,False,False,False,False,False,False,Fals
                                     False,False,False, False,False,False,True, True,
                                     False,False,False,False,False,False,False,False,
                                      False,False,False,False,False,False,False,False
                         {'title': 'Puzzle'}]),
            dict(label = 'Racing',
                 method = 'update',
                 args = [{'visible': [False,False,False,False,False,False,False,False
                                     False,False,False, False,False,False, False,Fals
                                      True, True, True,False,False,False,False,False,
                                      False,False,False,False,False,False,False,False
                         {'title': 'Racing'}]),
            dict(label = 'Role-playing',
                 method = 'update',
                 args = [{'visible': [False,False,False,False,False,False,False,False
                                      False,False,False, False,False,False, False,Fal
                                      False,False,False,True, True, True,False,False,
                                      False,False,False,False,False,False,False,False
                         {'title': 'Role-playing'}]),
             dict(label = 'Shooter',
                 method = 'update',
                 args = [{'visible': [False,False,False,False,False,False,False,False
                                      False,False,False, False,False,False, False,Fal
                                      False,False,False,False,False,False, True, True
                                      False,False,False,False,False,False,False,False
                         {'title': 'Shooter'}]),
             dict(label = 'Simulation',
                 method = 'update',
                 args = [{'visible': [False,False,False,False,False,False,False,False
                                      False,False,False, False,False,False, False,Fal
                                      False,False,False, False,False,False,
                                      True, True, True, False,False,False,False,False
                         {'title': 'Simulation'}]),
             dict(label = 'Sports',
                 method = 'update',
                 args = [{'visible': [False,False,False,False,False,False,False,False
                                      False,False,False, False,False,False, False,Fal
                                      False,False,False, False,False,False, False,Fal
                                      True, True, True, False, False,False,False,Fals
                         {'title': 'Sports'}]),
             dict(label = 'Strategy',
                 method = 'update',
                 args = [{'visible': [False,False,False,False,False,False,False,False
                                      False,False,False, False,False,False, False,Fal
                                      False,False,False,False,False,False,False,False
                                      False,False,False,False,False, True, True, True
                         {'title': 'Strategy'}]),

        ]),
    )
])

layout = dict(updatemenus=updatemenus,
             xaxis=dict(
        title='Publishers',
    ),
    yaxis=dict(
        title='Number of Videogames',
    )
             )

fig = Figure(data=data2, layout=layout)

py.iplot(fig)
```

Make sure to look at puzzles, strategy, and role-playing and compare these to the other genres. What do you notice? Japan sales are the highest in these three genres, suggesting that they like these type of games the most.

```
In [50]: import difflib
         import collections

         if not os.path.exists('Data/machine_learning_data.csv') or os.stat('Data/machine_lear
             sales_weekly['year_of_release'] = np.nan
             sales_weekly['name'] = sales_weekly['name'].astype(str)
             for index, row in sales.iterrows():
                 close_matches = difflib.get_close_matches(row['Name'],sales_weekly['name'],cu
                 for close_match in close_matches:
                     sales_weekly.loc[sales_weekly['name'] == close_match,'year_of_release'] =
             sales_weekly.to_csv('machine_learning_data.csv')
         else:
             sales_weekly = pd.read_csv('Data/machine_learning_data.csv').drop(columns=['index


         def diff(start, end):
             x = pd.to_datetime(end) - pd.to_datetime(start)
             return int(x / np.timedelta64(1, 'W'))

         sales_weekly = sales_weekly.dropna(thresh= 5)
```

```
for index,row in sales_weekly.iterrows():
    try:
        sales_weekly.loc[index,'time_difference'] = diff(row['year_of_release'],row['
    except:
         continue
sales_weekly = sales_weekly[sales_weekly['time_difference'] >= 0]
```

```
In [51]: sales_weekly['date'] = pd.to_datetime(sales_weekly['date'])
         sales_weekly['year_of_release'] = pd.to_datetime(sales_weekly['year_of_release'],forma
```

Before I apply machine learning methods, I want to see what my data looks like on plots.

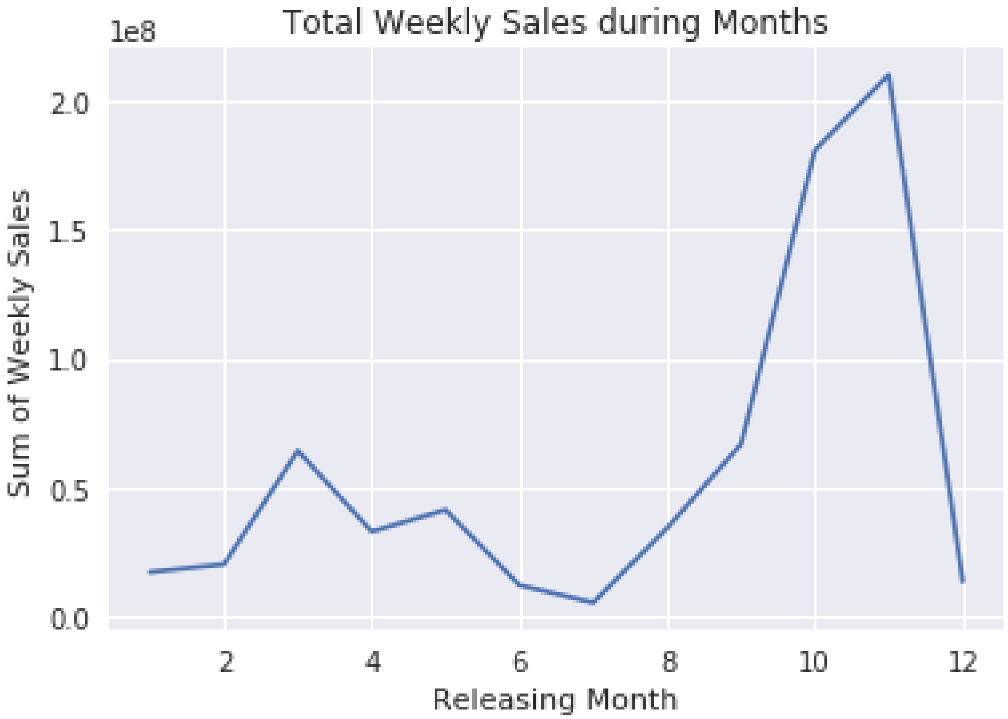
```
In [52]: sales_weekly['release_month'] = sales_weekly['year_of_release'].dt.month
```

```
In [53]: sales_weekly.groupby(['release_month']).size()
```

```
Out[53]: release_month
         1      519
         2     1138
         3     2525
         4     1491
         5     1554
         6      730
         7      343
         8     1134
         9     1596
         10    4748
         11    5223
         12     892
         dtype: int64
```

```
In [54]: sales_weekly.groupby('release_month').sum().plot(y='weekly_sales',legend=False)
         plt.xlabel('Releasing Month')
         plt.ylabel('Sum of Weekly Sales')
         plt.title('Total Weekly Sales during Months')
```

```
Out[54]: Text(0.5,1,'Total Weekly Sales during Months')
```



3.4 Machine Learning

```
In [55]: sales_weekly.dropna(how='all').drop('week_of_day',axis=1).corr().iplot(kind='heatmap'
```

3.4.1 Now time to predict weekly sales

```
In [56]: sales_weekly = pd.read_csv('Data/machine_learning_data.csv')

In [57]: sales_weekly.loc[:,'last_week_sales'] = np.nan
         sorted_sales_weekly = sales_weekly.sort_values(['name','location','date','console'])
         sorted_sales_weekly = sorted_sales_weekly.reset_index()

In [58]: for index,row in sorted_sales_weekly.iterrows():
             if index == 0:
                 sorted_sales_weekly.loc[index,'last_week_sales'] = 0.0
             elif sorted_sales_weekly.loc[index-1,'name'] != row['name']:
                 sorted_sales_weekly.loc[index,'last_week_sales'] = 0.0
             elif  sorted_sales_weekly.loc[index-1,'location'] != row['location']:
                 sorted_sales_weekly.loc[index,'last_week_sales'] = 0.0
             else:
                 sorted_sales_weekly.loc[index,'last_week_sales'] = sorted_sales_weekly.loc[ind

In [59]: sales_weekly=sorted_sales_weekly.set_index('index').drop(columns=['level0','Unnamed: (
In [60]: import datetime
         import dateutil.parser
         more_dates_df = pd.DataFrame(columns = sorted_sales_weekly.columns)
         for index,row in sales_weekly.iterrows():
             date = row.loc['start_date']
             for i in range(0,5):
                 new_row = row.copy(deep=True)
                 if i==1:
                     new_row.loc['last_week_sales'] = row['weekly_sales']
                 try:
                     new_row.loc['start_date'] = dateutil.parser.parse(date).date()
                     new_row.loc['date'] = dateutil.parser.parse(date).date()
                     date = dateutil.parser.parse(date).date() + datetime.timedelta(days=7)
                 except:
                     new_row.loc['start_date'] = date
                     new_row.loc['date'] = date
                     date = date + datetime.timedelta(days=7)
                 new_row.loc['time_difference'] = int(new_row['time_difference']) + i
                 more_dates_df = more_dates_df.append(new_row)
         more_dates_df['week'] = pd.to_datetime(more_dates_df["start_date"]).dt.week
         more_dates_df['month'] = pd.to_datetime(more_dates_df["start_date"]).dt.month
         more_dates_df['weekday'] = pd.to_datetime(more_dates_df["start_date"]).dt.weekday
         more_dates_df['year'] = pd.to_datetime(more_dates_df["start_date"]).dt.year
         more_dates_df = more_dates_df.dropna(how='all',axis=1)
```

```
In [61]: train_set = more_dates_df.iloc[::5, :].drop(columns=['start_date','position','end_date
         target = train_set['weekly_sales']
         first_set = more_dates_df.iloc[1::5, :].drop(columns=['start_date','position','end_da
         second_set = more_dates_df.iloc[2::5, :].drop(columns=['start_date','position','end_d
         third_set = more_dates_df.iloc[3::5, :].drop(columns=['start_date','position','end_da
         fourth_set = more_dates_df.iloc[4::5,:].drop(columns=['start_date','position','end_da
```

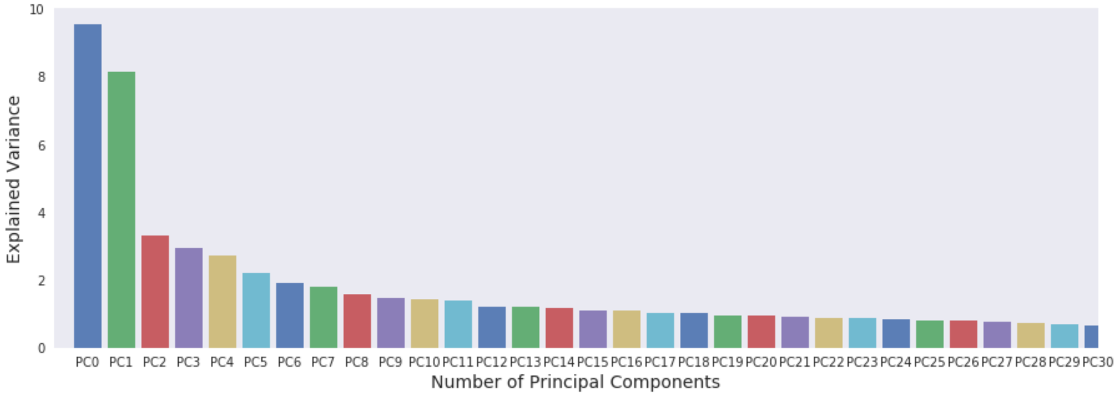
```
In [62]: sales_weekly = sorted_sales_weekly.reset_index(drop=True)
```

```
In [63]: from sklearn.preprocessing import StandardScaler
         for col in ['year', 'month', 'week', 'weekday','year_of_release','time_difference']:
             sales_weekly[col] = sales_weekly[col].astype('category')
         numerical = ['total_tweets','last_week_sales','weekly_sales']
         X = sales_weekly.loc[:,numerical]
         scaler  = StandardScaler().fit(X.values)
         Xscaled = scaler.transform(X.values).reshape(X.shape)
         Xscaled = pd.DataFrame(Xscaled,columns=numerical)
```

```
In [64]: target_dummies = Xscaled['weekly_sales']
         new_df = pd.concat([Xscaled[['total_tweets','last_week_sales']],sales_weekly.select_d
         new_df = new_df.drop(columns=['end_date','start_date'],errors='ignore')
         df_dummies = pd.get_dummies(new_df,drop_first = True)
         features_dummies = df_dummies.dropna()
```

Below are the calculated principal components.

```
In [65]: from sklearn.decomposition import PCA
         pca = PCA(n_components = 0.90)
         features_dummies_reduced = pca.fit(features_dummies).transform(features_dummies)
         fig = plt.figure(figsize=(15,5))
         ncol=np.arange(0,len(pca.components_),1)
         bar_labels = ['PC%s' %i for i in ncol]
         x_pos = list(ncol)
         for i in ncol:
             plt.bar(x=i, height=pca.explained_variance_ratio_[i]*100, align='center', alpha=0
         plt.xticks(x_pos, bar_labels)
         plt.xlabel('Number of Principal Components', fontsize=14)
         plt.ylabel('Explained Variance', fontsize=14)
         plt.xlim(-1, 30)
         plt.grid()
```



```
In [66]: from sklearn.model_selection import train_test_split
         from sklearn.linear_model import LinearRegression, ElasticNet
         from sklearn.ensemble import RandomForestRegressor
         from sklearn.ensemble import GradientBoostingRegressor
         from sklearn.ensemble import AdaBoostRegressor
         from sklearn.externals import joblib
         from sklearn.model_selection import GridSearchCV
         from sklearn import metrics
         X_train, X_test, y_train, y_test = train_test_split(features_dummies_reduced, target_

         search_space = [{"classifier":LinearRegression(),
                          'params':{'n_jobs':[-1]}},
                         {"classifier":ElasticNet(),
                          'params':{"alpha": np.arange(0.10,0.11,.01), "l1_ratio":  np.arange(0
                          {"classifier":RandomForestRegressor(n_jobs=-1),
                           "params":{"n_estimators": np.arange(8,9,1),"max_features":['sqrt', '
```

```
               {"classifier": GradientBoostingRegressor(),
                'params':{"alpha": np.arange(0.7,0.9,.1), "loss":['ls','quantile']}},
                {"classifier": AdaBoostRegressor(RandomForestRegressor(max_features='
                "params":{'loss':['linear', 'square', 'exponential']}}]

best_acc = 0.0
best_clf = 0
for row in search_space:
    random_search = GridSearchCV(row['classifier'],param_grid = row['params'], cv=5)
    random_search.fit(X_train, y_train.ravel())
    print (random_search.best_params_)
    predictions = random_search.predict(X_test)
    accuracy = metrics.r2_score(y_test, predictions)
    mse = metrics.mean_squared_error(y_test, predictions)
    print("mean squared error:", mse)
    print('Test set accuracy score for best params: %.3f ' % accuracy)
    if accuracy > best_acc:
          best_acc = accuracy
          best_clf = row['classifier']
print(best_acc,best_clf)
# Save to file in the current working directory
joblib_file = "joblib_model.pkl"
joblib.dump(best_clf, joblib_file)
```

```
{'n_jobs': -1}
mean squared error: 0.4933018161607935
Test set accuracy score for best params: 0.198
{'alpha': 0.1, 'l1_ratio': 0.1, 'max_iter': 1000}
mean squared error: 0.5165516708221671
Test set accuracy score for best params: 0.160
{'max_features': 'log2', 'n_estimators': 8}
mean squared error: 0.5702585858584472
Test set accuracy score for best params: 0.073
{'alpha': 0.7999999999999999, 'loss': 'quantile'}
mean squared error: 0.499029822951635
Test set accuracy score for best params: 0.189
{'loss': 'exponential'}
mean squared error: 0.3783568807280241
Test set accuracy score for best params: 0.385
0.384883095547078 AdaBoostRegressor(base_estimator=RandomForestRegressor(bootstrap=True, crite
           max_features='sqrt', max_leaf_nodes=None,
           min_impurity_decrease=0.0, min_impurity_split=None,
           min_samples_leaf=1, min_samples_split=2,
           min_weight_fraction_leaf=0.0, n_estimators=9, n_jobs=-1,
           oob_score=False, random_state=None, verbose=0, warm_start=False),
         learning_rate=1.0, loss='linear', n_estimators=50,
         random_state=None)
```

```
Out[66]: ['joblib_model.pkl']
```

```
In [67]: target_dummies.describe()
```

```
Out[67]: count    2.189300e+04
         mean     1.402967e-16
         std      1.000023e+00
         min     -2.270352e-01
         25%     -2.103277e-01
         50%     -1.715965e-01
         75%     -5.927510e-02
         max      7.054872e+01
         Name: weekly_sales, dtype: float64
```

3.4.2 Ready for predictions?

```
In [68]: from sklearn.preprocessing import StandardScaler
         for col in ['year', 'month', 'week','year_of_release','time_difference']:
             train_set[col] = train_set[col].astype('category',errors='ignore')
         numerical = ['last_week_sales','total_tweets','week_num']
         try:
             train_set[numerical] = train_set[numerical].astype('float64',errors='ignore')
         except:
             print('already done')
         X = train_set.loc[:,numerical].dropna(how='all',axis=1)
         scaler  = StandardScaler()
         Xscaled = scaler.fit_transform(X.values).reshape(X.shape)
         Xscaled = pd.DataFrame(Xscaled,columns=numerical)
```

```
In [69]: train_set['last_week_sales'] = Xscaled['last_week_sales'].values
         train_set['total_tweets'] = Xscaled['total_tweets'].values
         train_set['week_num'] = Xscaled['week_num'].values
         train_set = train_set.drop(columns=['weekly_sales'],axis=1,errors='ignore')
```

```
In [70]: train_set_dummies = pd.get_dummies(train_set,drop_first = True)
```

```
In [71]: first_set = first_set.drop(columns=['weekly_sales'],axis=1,errors='ignore')
         train_set_dummies.info()
         train_set_dummies.info()
         pca = PCA(n_components = 118)
         train_set_dummies_reduced = pca.fit(train_set_dummies).transform(train_set_dummies)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21893 entries, 0 to 21892
Columns: 2448 entries, index to time_difference_1307
dtypes: float64(3), int64(2), uint8(2443)
memory usage: 51.8 MB
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21893 entries, 0 to 21892
```

```
Columns: 2448 entries, index to time_difference_1307
dtypes: float64(3), int64(2), uint8(2443)
memory usage: 51.8 MB
```

```
In [72]: joblib_model = joblib.load(joblib_file)
         joblib_model.fit(train_set_dummies_reduced,target)
```

```
Out[72]: AdaBoostRegressor(base_estimator=RandomForestRegressor(bootstrap=True, criterion='mse
                    max_features='sqrt', max_leaf_nodes=None,
                    min_impurity_decrease=0.0, min_impurity_split=None,
                    min_samples_leaf=1, min_samples_split=2,
                    min_weight_fraction_leaf=0.0, n_estimators=9, n_jobs=-1,
                    oob_score=False, random_state=None, verbose=0, warm_start=False),
                  learning_rate=1.0, loss='linear', n_estimators=50,
                  random_state=None)
```

```
In [73]: from sklearn.preprocessing import StandardScaler
         def predicting(df):
             df = df.dropna(how='all')
             df = df.drop(columns=['weekly_sales'],errors='ignore')
             for col in ['year', 'month', 'week','year_of_release','time_difference']:
                 df[col] = df[col].astype('category',errors='ignore')
             numerical = ['last_week_sales','total_tweets','week_num']
             df[numerical] = df[numerical].astype('float64',errors='ignore')
             X = df.loc[:,numerical]
             scaler  = StandardScaler()
             Xscaled = scaler.fit_transform(X.values)
             Xscaled = pd.DataFrame(Xscaled,columns=numerical)
             df['last_week_sales'] = Xscaled['last_week_sales'].values
             df['total_tweets'] = Xscaled['total_tweets'].values
             df['week_num'] = Xscaled['week_num'].values
             df_dummies = pd.get_dummies(df,drop_first = True)
             df_dummies_reduced = pca.fit(df_dummies).transform(df_dummies)
             predictions = joblib_model.predict(df_dummies_reduced)
             return predictions
```

```
In [74]: first_set = first_set.drop(columns=['weekly_sales','index'],errors='ignore')
         second_set = second_set.drop(columns=['weekly_sales','index'],errors='ignore')
         third_set = third_set.drop(columns=['weekly_sales','index'],errors='ignore')
         fourth_set = fourth_set.drop(columns=['weekly_sales','index'],errors='ignore')
         print(fourth_set.shape,
              third_set.shape,
              second_set.shape)
```

```
(21893, 16) (21893, 16) (21893, 16)
```

```
In [75]: first_set['weekly_sales'] = predicting(first_set)
         second_set['last_week_sales'] = first_set['weekly_sales'].values
```

```
        second_set['weekly_sales'] = predicting(second_set)
        third_set['weekly_sales'] = predicting(third_set)
        fourth_set = fourth_set[:-1]
        fourth_set['weekly_sales'] = predicting(fourth_set)
        #third_set = third_set[:-1]
        third_set['last_week_sales'] = second_set['weekly_sales']
        fourth_set['last_week_sales'] = third_set['weekly_sales']
```

```
In [76]: first_set[['last_week_sales','total_tweets','week_num']] = scaler.inverse_transform(fi
         second_set[['last_week_sales','total_tweets','week_num']] = scaler.inverse_transform(s
         third_set[['last_week_sales','total_tweets','week_num']] = scaler.inverse_transform(th
         fourth_set[['last_week_sales','total_tweets','week_num']] = scaler.inverse_transform(f
```

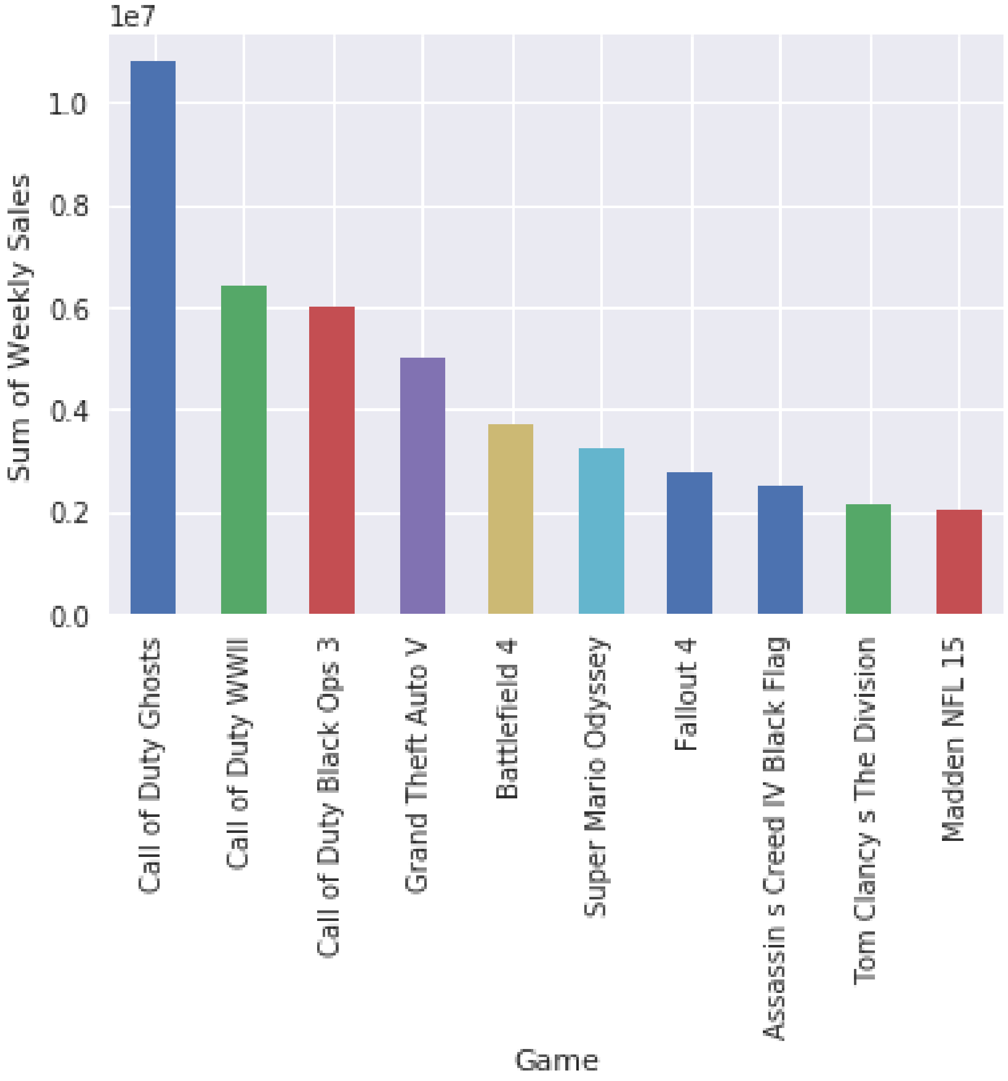
```
In [77]: USA_sales = sales_weekly[(sales_weekly['location'] == 'USA')].dropna(axis=1).drop(colu
```

```
In [78]: USA_sales.head()
```

```
Out[78]:     console         date   developer   genre location  \
         277    X360  05 Jul 2014  Electronic  Sports      USA
         278     PS3  12 Apr 2014  Electronic  Sports      USA
         279    X360  12 Apr 2014  Electronic  Sports      USA
         280    X360  28 Jun 2014  Electronic  Sports      USA
         350    WiiU  07 Nov 2015    Nintendo    Misc      USA

                                     name  position  total_sales  week_num  \
         277       2014 FIFA World Cup Brazil        26        66265        13
         278       2014 FIFA World Cup Brazil        12        15088         1
         279       2014 FIFA World Cup Brazil         9        23059         1
         280       2014 FIFA World Cup Brazil        27        59318        12
         350  Animal Crossing Amiibo Festival        11        72336         1

              weekly_sales       ...        year month  week  hour  weekday  total_tweets  \
         277          6947       ...        2014     7    27    19        4           0.0
         278         15088       ...        2014     4    15    21        4           0.0
         279         23059       ...        2014     4    15    21        4           0.0
         280          6562       ...        2014     6    26    13        4           0.0
         350         72336       ...        2015    11    45    21        4           0.0

                  year_of_release  time_difference week_of_day last_week_sales
         277  2014-04-15 00:00:00             11.0           5             0.0
         278  2014-04-15 00:00:00              0.0           5          6947.0
         279  2014-04-15 00:00:00              0.0           5         15088.0
         280  2014-04-15 00:00:00             10.0           5         23059.0
         350  2015-11-13 00:00:00              0.0           5             0.0

         [5 rows x 22 columns]
```

```
In [79]: USA_sales.groupby(["name"]).sum()['weekly_sales'].nlargest(10).plot(kind='bar')
         plt.xlabel('Game')
         plt.ylabel('Sum of Weekly Sales')
```

```
Out[79]: Text(0,0.5,'Sum of Weekly Sales')
```



```
In [80]: predictions = pd.DataFrame(sales_weekly,columns=first_set.columns)
         predictions = predictions.append(first_set)
         predictions = predictions.append(second_set)
         predictions = predictions.append(third_set)
         predictions = predictions.append(fourth_set)
         predictions = predictions.drop(columns=['index'],errors='ignore')
```

```
In [81]: predictions.groupby('name').head(10)
```

```
Out[81]:       console          date       developer          genre location  \
         0         PS4   28 Oct 2017    Namco Bandai   Role-Playing   Global
         1         PS4   04 Nov 2017    Namco Bandai   Role-Playing    Japan
```

2	PS4	11 Nov 2017	Namco Bandai	Role-Playing	Japan
3	PS4	18 Nov 2017	Namco Bandai	Role-Playing	Japan
4	PS4	28 Oct 2017	Namco Bandai	Role-Playing	Japan
5	NS	01 Jul 2017	Nintendo	Party	Europe
6	NS	02 Dec 2017	Nintendo	Party	Europe
7	NS	03 Feb 2018	Nintendo	Party	Europe
8	NS	03 Jun 2017	Nintendo	Party	Europe
9	NS	06 Jan 2018	Nintendo	Party	Europe
10	NS	08 Jul 2017	Nintendo	Party	Europe
11	NS	09 Dec 2017	Nintendo	Party	Europe
12	NS	10 Feb 2018	Nintendo	Party	Europe
13	NS	13 Jan 2018	Nintendo	Party	Europe
14	NS	15 Jul 2017	Nintendo	Party	Europe
157	PS3	03 May 2014	Electronic	Sports	Europe
158	X360	03 May 2014	Electronic	Sports	Europe
159	PS3	05 Jul 2014	Electronic	Sports	Europe
160	X360	05 Jul 2014	Electronic	Sports	Europe
161	PS3	07 Jun 2014	Electronic	Sports	Europe
162	X360	07 Jun 2014	Electronic	Sports	Europe
163	PS3	09 Aug 2014	Electronic	Sports	Europe
164	PS3	10 May 2014	Electronic	Sports	Europe
165	X360	10 May 2014	Electronic	Sports	Europe
166	PS3	12 Apr 2014	Electronic	Sports	Europe
281	PSP	13 Apr 2013	Sega	Role-Playing	Global
282	Wii	26 Jan 2013	Ubisoft	Misc	UK
283	PS4	02 Sep 2017	Studio	Action	Europe
284	PS4	02 Sep 2017	Studio	Action	France
285	PS4	09 Sep 2017	Studio	Action	France
...	...	...	...	...	...
19886	PS3	2013-08-10	Nippon Ichi	Action	Global
20633	PC	2015-03-14	Sega	Strategy	UK
20699	PS4	2015-10-31	Activision	Action	Global
20700	PS4	2015-10-31	Activision	Action	UK
20701	PS3	2012-09-22	Activision	Action	Global
20702	X360	2012-09-22	Activision	Action	Global
20708	PS3	2012-04-07	Sony Computer	Action	UK
20709	PS3	2012-04-14	Sony Computer	Action	UK
20754	PS4	2017-04-01	Capcom	Fighting	Japan
21104	PS4	2017-08-12	Arc System	Fighting	Japan
21165	PS4	2016-11-05	Sony Computer	Adventure	UK
21166	PS4	2016-11-12	Sony Computer	Adventure	UK
21167	PSV	2017-02-18	Broccoli	Adventure	Japan
21168	PSV	2017-02-25	Broccoli	Adventure	Japan
21169	PS4	2016-06-18	Sega	Role-Playing	Global
21170	PS4	2018-02-10	Sega	Role-Playing	Japan
21171	PSV	2017-02-04	Marvelous	Action	Japan
21172	PSV	2017-02-11	Marvelous	Action	Japan
21176	PS4	2015-11-07	Bigben	Sports	Europe

```
21177     PS4   2015-11-07          Bigben        Sports       UK
21178      PC   2017-05-20         Unknown      Strategy   France
21179      NS   2017-12-02           Tecmo        Action    Japan
21220     PS4   2017-03-25           Tecmo          Misc    Japan
21221     PSV   2017-03-25           Tecmo          Misc    Japan
21235     PSV   2012-03-17   Sony Computer        Racing       UK
21539     PS3   2014-03-22            Sega        Action   Global
21547     PS4   2017-02-18            Sega     Adventure   France
21548     PS4   2017-02-25            Sega     Adventure   France
21775    XOne   2016-05-21  Microsoft Game    Simulation       UK
21776    XOne   2016-05-28  Microsoft Game    Simulation       UK

                                     name     week_num  year  month  week  \
0                 .hack//G.U. Last Recode     1.000000  2017     10    43
1                 .hack//G.U. Last Recode     2.000000  2017     11    44
2                 .hack//G.U. Last Recode     3.000000  2017     11    45
3                 .hack//G.U. Last Recode     4.000000  2017     11    46
4                 .hack//G.U. Last Recode     1.000000  2017     10    43
5                              1-2-Switch    19.000000  2017      7    26
6                              1-2-Switch    41.000000  2017     12    48
7                              1-2-Switch    50.000000  2018      2     5
8                              1-2-Switch    15.000000  2017      6    22
9                              1-2-Switch    46.000000  2018      1     1
10                             1-2-Switch    20.000000  2017      7    27
11                             1-2-Switch    42.000000  2017     12    49
12                             1-2-Switch    51.000000  2018      2     6
13                             1-2-Switch    47.000000  2018      1     2
14                             1-2-Switch    21.000000  2017      7    28
157            2014 FIFA World Cup Brazil     4.000000  2014      5    18
158            2014 FIFA World Cup Brazil     4.000000  2014      5    18
159            2014 FIFA World Cup Brazil    13.000000  2014      7    27
160            2014 FIFA World Cup Brazil    13.000000  2014      7    27
161            2014 FIFA World Cup Brazil     9.000000  2014      6    23
162            2014 FIFA World Cup Brazil     9.000000  2014      6    23
163            2014 FIFA World Cup Brazil    18.000000  2014      8    32
164            2014 FIFA World Cup Brazil     5.000000  2014      5    19
165            2014 FIFA World Cup Brazil     5.000000  2014      5    19
166            2014 FIFA World Cup Brazil     1.000000  2014      4    15
281                    7th Dragon 2020-II     1.000000  2013      4    15
282                    ABBA You Can Dance    63.000000  2013      1     4
283                  ARK Survival Evolved     2.000000  2017      9    35
284                  ARK Survival Evolved     2.000000  2017      9    35
285                  ARK Survival Evolved     3.000000  2017      9    36
...                                   ...          ...   ...    ...   ...
19886                   Time and Eternity  1571.553527  2013      8    32
20633                    Total War Attila    63.128878  2015      3    11
20699            Transformers Devastation    63.128878  2015     10    44
20700            Transformers Devastation    63.128878  2015     10    44
```

```
20701      Transformers Fall of Cybertron   100.839494  2012      9    38
20702      Transformers Fall of Cybertron   100.839494  2012      9    38
20708                       Twisted Metal    63.128878  2012      4    14
20709                       Twisted Metal   100.839494  2012      4    15
20754        Ultimate Marvel vs. Capcom 3    63.128878  2017      4    13
21104     Under Night In-Birth Exe Latest    63.128878  2017      8    32
21165            Until Dawn Rush of Blood    63.128878  2016     11    44
21166            Until Dawn Rush of Blood   100.839494  2016     11    45
21167      Uta no Prince-Sama Repeat Love    63.128878  2017      2     7
21168      Uta no Prince-Sama Repeat Love   100.839494  2017      2     8
21169                 Valkyria Chronicles   628.788121  2016      6    24
21170                 Valkyria Chronicles  3871.901116  2018      2     6
21171            Valkyrie Drive Bhikkhuni  2212.634002  2017      2     5
21172            Valkyrie Drive Bhikkhuni  2250.344619  2017      2     6
21176  WRC 5 FIA World Rally Championship    63.128878  2015     11    45
21177  WRC 5 FIA World Rally Championship    63.128878  2015     11    45
21178     Warhammer 40000 Dawn of War III    63.128878  2017      5    20
21179                   Warriors Orochi 3    63.128878  2017     12    48
21220                 Winning Post 8 2017    63.128878  2017      3    12
21221                 Winning Post 8 2017    63.128878  2017      3    12
21235                        Wipeout 2048    63.128878  2012      3    11
21539                        Yakuza Ishin   100.839494  2014      3    12
21547       Yakuza Zero The Place of Oath    63.128878  2017      2     7
21548       Yakuza Zero The Place of Oath   100.839494  2017      2     8
21775                     Zoo Tycoon 2013  4852.377138  2016      5    20
21776                     Zoo Tycoon 2013  4890.087754  2016      5    21

       weekday  total_tweets      year_of_release time_difference week_of_day  \
0            4      0.000000  2017-11-03 00:00:00               0           5
1            2      0.000000  2017-11-03 00:00:00               0           5
2            3      0.000000  2017-11-03 00:00:00               1           5
3            4     55.000000  2017-11-03 00:00:00               2           5
4            4      0.000000  2017-11-03 00:00:00               0           5
5            4      0.000000  2017-03-03 00:00:00              17           5
6            4    222.000000  2017-03-03 00:00:00              39           5
7            4   2718.000000  2017-03-03 00:00:00              48           5
8            4      0.000000  2017-03-03 00:00:00              13           5
9            4   3494.000000  2017-03-03 00:00:00              44           5
10           4      0.000000  2017-03-03 00:00:00              18           5
11           4   1702.000000  2017-03-03 00:00:00              40           5
12           4   1292.000000  2017-03-03 00:00:00              49           5
13           4   2006.000000  2017-03-03 00:00:00              45           5
14           4      0.000000  2017-03-03 00:00:00              19           5
157          4      0.000000  2014-04-15 00:00:00               2           5
158          4      0.000000  2014-04-15 00:00:00               2           5
159          4      0.000000  2014-04-15 00:00:00              11           5
160          4      0.000000  2014-04-15 00:00:00              11           5
161          4      0.000000  2014-04-15 00:00:00               7           5
```

```
162        4      0.000000  2014-04-15 00:00:00              7            5
163        4      0.000000  2014-04-15 00:00:00             16            5
164        4      0.000000  2014-04-15 00:00:00              3            5
165        4      0.000000  2014-04-15 00:00:00              3            5
166        4      0.000000  2014-04-15 00:00:00              0            5
281        3      0.000000  2011-11-23 00:00:00             72            5
282        2      0.000000  2011-11-15 00:00:00             62            5
283        3      0.000000  2017-08-29 00:00:00              0            5
284        3      0.000000  2017-08-29 00:00:00              0            5
285        4      0.000000  2017-08-29 00:00:00              1            5
...      ...           ...                  ...            ...          ...
19886      5    333.362445  2013-07-16 00:00:00              4            5
20633      5    333.362445  2015-02-17 00:00:00              4            5
20699      5    333.362445  2015-10-06 00:00:00              4            5
20700      5    333.362445  2015-10-06 00:00:00              4            5
20701      5    333.362445  2012-08-21 00:00:00              4            5
20702      5    333.362445  2012-08-21 00:00:00              4            5
20708      5    333.362445  2012-02-14 00:00:00              7            5
20709      5    333.362445  2012-02-14 00:00:00              8            5
20754      5    333.362445  2017-03-07 00:00:00              4            5
21104      5    333.362445  2015-02-24 00:00:00            128            5
21165      5    333.362445  2016-10-13 00:00:00              4            5
21166      5    333.362445  2016-10-13 00:00:00              4            5
21167      5    333.362445  2011-08-11 00:00:00            288            5
21168      5    333.362445  2011-08-11 00:00:00            289            5
21169      5    333.362445  2016-05-17 00:00:00              4            5
21170      5  66904.227587  2016-05-17 00:00:00             90            5
21171      5    333.362445  2016-10-11 00:00:00             16            5
21172      5    333.362445  2016-10-11 00:00:00             17            5
21176      5    333.362445  2011-10-14 00:00:00            212            5
21177      5    333.362445  2011-10-14 00:00:00            212            5
21178      5    333.362445  2009-02-19 00:00:00            430            5
21179      5  15001.519171  2014-09-02 00:00:00            169            5
21220      5    333.362445  1993-09-10 00:00:00           1228            5
21221      5    333.362445  1993-09-10 00:00:00           1228            5
21235      5    333.362445  2012-02-15 00:00:00              4            5
21539      5    333.362445  2014-02-22 00:00:00              4            5
21547      5    333.362445  2015-03-12 00:00:00            101            5
21548      5    333.362445  2015-03-12 00:00:00            102            5
21775      5    333.362445  2013-11-22 00:00:00            130            5
21776      5    333.362445  2013-11-22 00:00:00            131            5

       last_week_sales  weekly_sales
0         0.000000e+00  89508.000000
1         0.000000e+00   6312.000000
2         6.312000e+03   3395.000000
3         3.395000e+03   2787.000000
4         2.787000e+03  57615.000000
```

```
5          0.000000e+00   8502.000000
6          8.502000e+03  31690.000000
7          3.169000e+04   9731.000000
8          9.731000e+03   9130.000000
9          9.130000e+03  11054.000000
10         1.105400e+04   9049.000000
11         9.049000e+03  37563.000000
12         3.756300e+04  12915.000000
13         1.291500e+04  12752.000000
14         1.275200e+04  10859.000000
157        0.000000e+00  19276.000000
158        1.927600e+04   9709.000000
159        9.709000e+03  17925.000000
160        1.792500e+04   6773.000000
161        6.773000e+03  18144.000000
162        1.814400e+04  14437.000000
163        1.443700e+04   6896.000000
164        6.896000e+03  17356.000000
165        1.735600e+04   8399.000000
166        8.399000e+03  38053.000000
281        0.000000e+00  74694.000000
282        0.000000e+00   3576.000000
283        0.000000e+00  13628.000000
284        0.000000e+00   2627.000000
285        2.627000e+03   2148.000000
...                 ...            ...
19886      5.787135e+09  32268.555556
20633      1.728459e+09  15199.111111
20699      1.017445e+10  36788.666667
20700      3.503677e+09  27022.555556
20701      6.676600e+09  38270.666667
20702      7.675110e+09  41691.777778
20708      2.593571e+09  19500.222222
20709      2.691435e+09  17709.333333
20754      6.314960e+09  18646.222222
21104      2.146077e+09  11868.777778
21165      3.705304e+09  16810.777778
21166      3.070080e+09  16588.333333
21167      3.966304e+09  16566.000000
21168      3.492828e+09  12123.333333
21169      7.881912e+09  28948.666667
21170      9.903405e+09  40160.666667
21171      4.030642e+09  34501.666667
21172      6.147206e+09  30291.000000
21176      2.466843e+09  25503.555556
21177      1.896514e+09  22819.444444
21178      1.600433e+09  17885.777778
21179      2.468774e+09  32971.444444
```

```
21220     5.142918e+09  24673.777778
21221     3.854906e+09  21163.222222
21235     2.186197e+09  14322.888889
21539     5.438788e+09  40480.555556
21547     4.621823e+09  18418.333333
21548     4.459018e+09  16494.333333
21775     3.449328e+09  28879.555556
21776     4.298960e+09  28516.333333

[6230 rows x 17 columns]
```

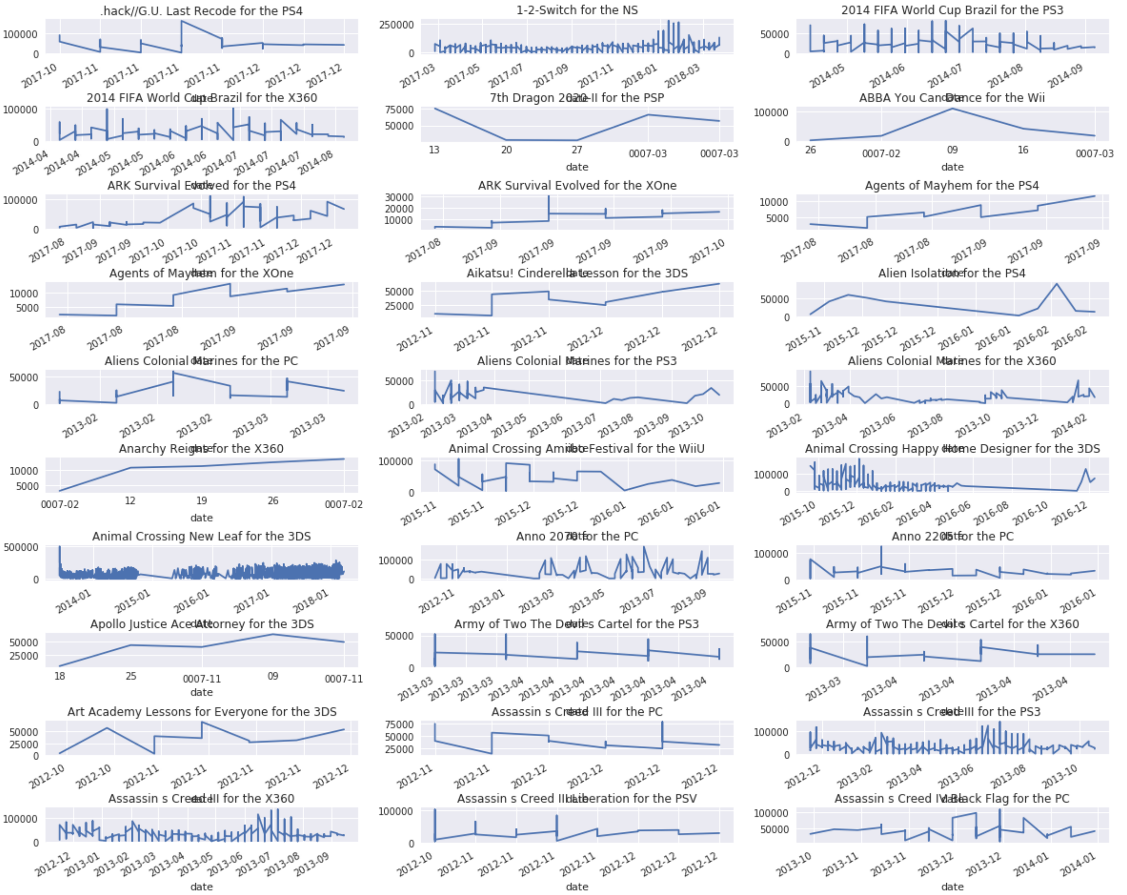
```
In [82]: predictions['time_difference'] = pd.to_numeric(predictions['time_difference'])
         predictions['weekly_sales'] = pd.to_numeric(predictions['weekly_sales'])
```

```
In [83]: first_set = first_set[:-1]
         second_set = second_set[:-1]
         third_set = third_set[:-1]
```

```
In [84]: predictions = pd.read_csv('predictions.csv')
```

```
In [85]: predictions['date'] = pd.to_datetime(predictions['date'])
```

```
In [86]: import matplotlib.dates as dates
         fig, axes = plt.subplots(10,3, figsize=(20,15))
         for (index, group), ax in zip(predictions.groupby(['name','console']), axes.flatten()
             group.sort_values(by='date').set_index('date').plot(y='weekly_sales', ax=ax, titl
             ax.xaxis.set_major_formatter(dates.DateFormatter("%Y-%m"))
             plt.subplots_adjust(top=1.5, bottom = 0.7 ,hspace = 1.5)
```


.hack//G.U. Last Recode for the PS4
100000
0
2017-10 2017-11 2017-11 2017-11 2017-11 2017-12 2017-12 2017-12
1-2-Switch for the NS
250000
0
2017-03 2017-05 2017-07 2017-09 2017-11 2018-01 2018-03
2014 FIFA World Cup Brazil for the PS3
50000
0
2014-05 2014-06 2014-07 2014-08 2014-09
2014 FIFA World Cup Brazil for the X360
100000
0
2014-04 2014-04 2014-05 2014-05 2014-06 2014-06 2014-07 2014-07 2014-07 2014-08
7th Dragon 2020-II for the PSP
75000
50000
13 20 27 0007-03 0007-03
date
ABBA You Can Dance for the Wii
100000
0
26 0007-02 09 16 0007-03
date
ARK Survival Evolved for the PS4
100000
0
2017-08 2017-09 2017-09 2017-10 2017-10 2017-11 2017-11 2017-12 2017-12
ARK Survival Evolved for the XOne
30000
20000
10000
2017-08 2017-09 2017-09 2017-09 2017-09 2017-10
Agents of Mayhem for the PS4
10000
5000
2017-08 2017-08 2017-08 2017-09 2017-09 2017-09
Agents of Mayhem for the XOne
10000
5000
2017-08 2017-08 2017-08 2017-09 2017-09 2017-09
Aikatsu! Cinderella Lesson for the 3DS
50000
25000
2012-11 2012-11 2012-11 2012-12 2012-12 2012-12
Alien Isolation for the PS4
50000
0
2015-11 2015-12 2015-12 2015-12 2016-01 2016-01 2016-02 2016-02
Aliens Colonial Marines for the PC
50000
0
2013-02 2013-02 2013-02 2013-03 2013-03
Aliens Colonial Marines for the PS3
50000
0
2013-02 2013-03 2013-04 2013-05 2013-06 2013-07 2013-08 2013-09 2013-10
Aliens Colonial Marines for the X360
50000
0
2013-02 2013-04 2013-06 2013-08 2013-10 2013-12 2014-02
Anarchy Reigns for the X360
10000
5000
0007-02 12 19 26 0007-02
date
Animal Crossing Amiibo Festival for the WiiU
100000
0
2015-11 2015-11 2015-12 2015-12 2016-01 2016-01 2016-01
Animal Crossing Happy Home Designer for the 3DS
100000
0
2015-10 2015-12 2016-02 2016-04 2016-06 2016-08 2016-10 2016-12
Animal Crossing New Leaf for the 3DS
500000
0
2014-01 2015-01 2016-01 2017-01 2018-01
Anno 2070 for the PC
100000
0
2012-11 2013-01 2013-03 2013-05 2013-07 2013-09
Anno 2205 for the PC
100000
0
2015-11 2015-11 2015-11 2015-12 2015-12 2016-01 2016-01
Apollo Justice Ace Attorney for the 3DS
50000
25000
18 25 0007-11 09 0007-11
date
Army of Two The Devils Cartel for the PS3
50000
0
2013-03 2013-03 2013-03 2013-04 2013-04 2013-04 2013-04 2013-04 2013-04 2013-04
Army of Two The Devils Cartel for the X360
50000
0
2013-03 2013-04 2013-04 2013-04 2013-04
Art Academy Lessons for Everyone for the 3DS
50000
25000
0
2012-10 2012-10 2012-11 2012-11 2012-11 2012-11 2012-12
Assassin s Creed III for the PC
75000
50000
25000
2012-11 2012-11 2012-12 2012-12 2012-12 2012-12
Assassin s Creed III for the PS3
100000
0
2012-12 2013-02 2013-04 2013-06 2013-08 2013-10
Assassin s Creed III for the X360
100000
0
2012-12 2013-01 2013-02 2013-03 2013-04 2013-05 2013-06 2013-07 2013-08 2013-09
date
Assassin s Creed III Liberation for the PSV
100000
0
2012-10 2012-11 2012-11 2012-11 2012-11 2012-12 2012-12 2012-12
date
Assassin s Creed IV Black Flag for the PC
100000
50000
2013-10 2013-11 2013-11 2013-12 2013-12 2014-01 2014-01
date