Learning and Predicting from Video Game Sales History

A project by Martin Blurton-Jones for CMSC320

Why Video Games? \ While many see video games as just a hobby, or worse, just a mindless distraction, like many hobbies (and mindless distractions) there is a lot of profit to be made from it. Even within the relatively short existance of home video game consoles (having been only around for a little over half a decade), developers, ranging from independent programmers to massive companies, have published tens of thousands of games for people to play around the world. Video game development in itself has become a significant industry. \ \ According to this wikipedia article here, in 2017, the US video game industry is estimated to have about 66,000 direct employed workers, and over 220,000 employed workers when counting indirect employment. Additionally, in 2018, the US video game industry earned about $43 Billion USD, matching the revenue of the US film industry. With video games on the rise, it is important to understand historical trends to try and predict where the industry may be headed. \ \ **Today's Goal:** \ The goal of this tutorial is to use video game sales data scraped from vgchartz.com to try and make insights about the video game industry based on historical patterns, and eventually to train two machine learning models to make predictions about what video games would sell more than others.

Imports: \ The most important libraries we will be using are: \ BeautifulSoup, an incredibly helpful library for scraping data from html websites and reading them into nicer datastructures. \ Numpy, a library that provides many essential mathematical functions and processes. \ Pandas, a library that provides very useful datastructures for storing tables in rows and columns. \ Seaborn, an elegant and effective way to graph and plot various forms of data. \ Scikit-Learn, a powerful tool for implementing machine learning in a project, including everything from preparing the training data to providing models to train.

In [175…

```python
import requests
from bs4 import BeautifulSoup
import numpy as np
import pandas as pd
import re
import datetime
import seaborn as sns
import matplotlib.pyplot as plt
import sklearn
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import cross_validate
from sklearn.model_selection import KFold
from sklearn.model_selection import GridSearchCV
```

Parsing Onwards

The first step of analyzing data is gathering the data together. \ The table that this project will be using is available at vgchartz.com. \ \ **Reading from the site:** \ While BeautifulSoup allows us to read a lot of the data directly from the table into a nice pandas dataframe, there are some things about the table that make it difficult to do so. First, the maximum size for each page is 200 results, meaning each page must be parsed individually and then combined into one big dataframe. As for how many pages will be counted, that can be set with the pageCount variable (for performance reasons I reccomend about 10-15 pages). The next issue is that the table includes a consoles entry, which is a helpful piece of data to have about a game. However, each entry is actually a tiny icon that represents a console. To account for this, we can look at the url of the image that is called on each console listing, and then, using a regular expression, we can parse the part of the url that contains the name of the console.

In [176…

```python
#Define the number of pages to scrape (each page has 200 listings, there are 309 page
#Lower numbers are better/recommended if processing power is limited
#25 gives about 5000 listings (hard)
#10 gives about 2000 listings (medium)
#5 gives about 1000 listings (easy)
pageCount = 10

#Prepare the dataframe that will hold all the scraped data
vgData = pd.DataFrame()

#Go through the table page by page
for page in (range(1, pageCount + 1)):
    #Print out the current page to keep track of progress
    print("On page: " + str(page))

    #Creating the URL to get the current page of the table
    vgchartz = requests.get('https://www.vgchartz.com/games/games.php?page=' + str(pa

    #Making the soup
    vg_soup = BeautifulSoup(vgchartz.text, 'html.parser')

    #Parsing the table from the page
    table = vg_soup.find('div',{"id":"generalBody"})

    #Reading the table into a pandas dataframe, labelling its columns
    pageData = pd.read_html(str(table))[0]
    pageData.columns = ['Pos','GamePic','Title','Console','Publisher','Developer','Cr

    #The table uses small icons to represent which console the game belongs to,
    #So we can find each time one of the console pictures is called, read its filenam
    consoleRE = re.compile('consoles/[0-9a-zA-Z_]+\\.png')
    consolepicsS = re.findall(consoleRE, str(table))
    consoleNames = []
    for i in consolepicsS:
        consoleNames.append(i[9:len(i)-6])
    #Replaces the empty Console column with the newly scraped values
    pageData['Console'] = consoleNames

    #Attatch this page's data to the main dataframe
    vgData = vgData.append(pageData, ignore_index=True)

#After adding each page, print the head of the dataframe to preview the newly generat
print(vgData.head(25))
```

```
On page: 1
On page: 2
On page: 3
On page: 4
On page: 5
On page: 6
On page: 7
On page: 8
On page: 9
On page: 10
    Pos  GamePic                                         Title Console  \
0     1      NaN                                    Wii Sports     Wii
1     2      NaN                             Super Mario Bros.     NES
2     3      NaN                   Mario Kart Wii Read the review     Wii
3     4      NaN              Wii Sports Resort Read the review     Wii
4     5      NaN               Pokémon Red / Green / Blue Version      GB
5     6      NaN                                        Tetris      GB
6     7      NaN                         New Super Mario Bros.      DS
7     8      NaN                     Wii Play Read the review     Wii
8     9      NaN     New Super Mario Bros. Wii Read the review     Wii
9    10      NaN                                     Duck Hunt     NES
10   11      NaN                                    Nintendogs      DS
11   12      NaN                                 Mario Kart DS      DS
12   13      NaN                Pokémon Gold / Silver Version      GB
13   14      NaN                       Wii Fit Read the review     Wii
14   15      NaN             Kinect Adventures! Read the review    X360
15   16      NaN                                  Wii Fit Plus     Wii
16   17      NaN                  Grand Theft Auto: San Andreas     PS2
17   18      NaN                             Super Mario World    SNES
18   19      NaN                            Grand Theft Auto V     PS3
19   20      NaN  Brain Age: Train Your Brain in Minutes a Day      DS
20   21      NaN                            Grand Theft Auto V     PS4
21   22      NaN                Pokémon Diamond / Pearl Version      DS
22   23      NaN                             Super Mario Land      GB
23   24      NaN                           Super Mario Bros. 3     NES
24   25      NaN                   Grand Theft Auto: Vice City     PS2

                 Publisher             Developer  Critic Score  User Score  \
0                 Nintendo          Nintendo EAD           7.7         NaN
1                 Nintendo          Nintendo EAD          10.0         8.2
2                 Nintendo          Nintendo EAD           8.2         9.1
3                 Nintendo          Nintendo EAD           8.0         8.8
4                 Nintendo            Game Freak           9.4         NaN
5                 Nintendo  Bullet Proof Software          NaN         NaN
6                 Nintendo          Nintendo EAD           9.1         8.1
7                 Nintendo          Nintendo EAD           5.9         4.5
8                 Nintendo          Nintendo EAD           8.6         9.2
9                 Nintendo          Nintendo R&D1          NaN         NaN
10                Nintendo          Nintendo EAD           8.4         NaN
11                Nintendo          Nintendo EAD           9.1         9.4
12                Nintendo            Game Freak           9.2         NaN
13                Nintendo          Nintendo EAD           7.9         NaN
14  Microsoft Game Studios    Good Science Studio           6.7         NaN
15                Nintendo          Nintendo EAD           8.0         NaN
16          Rockstar Games        Rockstar North           9.5         NaN
17                Nintendo          Nintendo EAD           8.5         NaN
18          Rockstar Games        Rockstar North           9.4         NaN
19                Nintendo          Nintendo SDD           8.1         NaN
20          Rockstar Games        Rockstar North           9.7         NaN
21                Nintendo            Game Freak           8.6         NaN
22                Nintendo          Nintendo R&D1          NaN         NaN
23                Nintendo          Nintendo R&D2          NaN         NaN
24          Rockstar Games        Rockstar North           9.6         NaN

   Total Sales NA Sales PAL Sales Japan Sales Other Sales Release Date  \
0       82.65m   41.36m    29.02m       3.77m       8.51m   19th Nov 06
1       40.24m   29.08m     3.58m       6.81m       0.77m   18th Oct 85
2       35.98m   15.91m    12.92m       3.80m       3.35m   27th Apr 08
3       32.90m   15.61m    10.99m       3.29m       3.02m   26th Jul 09
4       31.37m   11.27m     8.89m      10.22m       1.00m   30th Sep 98
5       30.26m   23.20m     2.26m       4.22m       0.58m   31st Jul 89
6       29.85m   11.28m     9.19m       6.50m       2.89m   15th May 06
7       28.92m   13.96m     9.18m       2.93m       2.85m   12th Feb 07
8       28.51m   14.53m     7.01m       4.70m       2.27m   15th Nov 09
9       28.31m   26.93m     0.63m       0.28m       0.47m   15th Oct 85
10      24.68m    9.05m    10.96m       1.93m       2.74m   23rd Aug 05
11      23.26m    9.71m     7.51m       4.13m       1.91m   14th Nov 05
12      23.10m    9.00m     6.18m       7.20m       0.71m   14th Oct 00
13      22.70m    8.92m     8.03m       3.60m       2.15m   21st May 08
14      22.10m   15.09m     4.97m       0.24m       1.81m   04th Nov 10
15      21.81m    9.01m     8.51m       2.53m       1.76m   04th Oct 09
16      20.81m    9.43m     0.40m       0.41m      10.57m   26th Oct 04
17      20.61m   12.78m     3.75m       3.54m       0.55m   13th Aug 91
18      20.32m    6.37m     9.85m       0.99m       3.12m   17th Sep 13
19      20.16m    4.74m     9.21m       4.16m       2.04m   17th Apr 06
20      19.39m    6.06m     9.71m       0.60m       3.02m   18th Nov 14
21      18.25m    6.38m     4.47m       6.04m       1.36m   28th Apr 07
22      18.14m   10.83m     2.71m       4.18m       0.42m   01st Aug 89
23      17.28m    9.54m     3.44m       3.84m       0.46m   12th Feb 90
24      16.15m    8.41m     5.49m       0.47m       1.78m   28th Oct 02

   Last Update  Misread
0          NaN      NaN
1          NaN      NaN
2    11th Apr 18      NaN
3          NaN      NaN
4          NaN      NaN
5          NaN      NaN
6          NaN      NaN
7          NaN      NaN
8          NaN      NaN
9          NaN      NaN
10         NaN      NaN
11         NaN      NaN
12         NaN      NaN
13         NaN      NaN
14   29th Sep 18      NaN
15         NaN      NaN
16         NaN      NaN
17   10th Aug 18      NaN
18         NaN      NaN
19         NaN      NaN
20   03rd Jan 18      NaN
21         NaN      NaN
22         NaN      NaN
23   09th Aug 18      NaN
24         NaN      NaN
```

Tidying up the Table: \ As you may have noticed, the resulting table contains a lot of strange and confusing data.\ The next step is of course, to clean up all this distorted data.\ \ The first step of the cleaning is to get rid of some excess columns that contained erroneous or not very useful information, in this case 'Pos', 'GamePic', and 'Misread' which contained the position of the row in the original table, an icon for the original game, and a column of all NaN's, respectively. \ After those are dropped, all of the numerical sales values need to be adjusted. Each numerical sales value in this table is reported in millions with an 'm' at the end. So the function dropM goes through a column and removes the m and converts the remaining decimal into a float. \ Next, vgchartz.com links to reviews on some articles, which means many of the titles need "Read the review" trimmed off of them. This is an easy fix under the assumption that there is no game named "Read the review". \ Then, the next stage involves converting the reported release dates to datetime objects for convenience, however some values needed manual correcting because pandas' .todatetime reads the year '70 as 2070 instead of 1970. \ Finally, the last change is the removal of "Series" and "All" entries from the table, which both correspond to entire game series or multi-platform releases (individual titles are already counted on their own).

In [177…

```python
#These axis can be removed entirely as they serve no purpose
#(Pos is just the position in the table, Gamepic links to the game's box art which is
vgData = vgData.drop(['Pos','GamePic','Misread'], axis=1)

#This function takes in a column name and for each item in that column it will remove
#and converts the value into a float
def dropM(col):
    for i, row in vgData.iterrows():
        if type(vgData.at[i,col]) != float:
            vgData.at[i,col] = float(vgData.at[i, col][0:(len(vgData.at[i, col])-1)])

#The dropM function is called on each column that represents sales in the millions
dropM('Total Sales')
dropM('NA Sales')
dropM('PAL Sales')
dropM('Japan Sales')
dropM('Other Sales')

#Removes the 'Read the review' tag that was added to each title where the website had
for i, row in vgData.iterrows():
    vgData.at[i, 'Title'] = vgData.at[i, 'Title'].replace(' Read the review', '')

#Converts the Release Date and Last Update values into datetime objects for convenien
#Also, because the years are only 2 digits, sometimes pandas makes assumptions like '
#This corrects for that
vgData['Release Date'] = pd.to_datetime(vgData['Release Date'], errors='coerce')
for i, row in vgData.iterrows():
    if vgData.at[i, 'Release Date'] > datetime.datetime.now():
        vgData.at[i, 'Release Date'] = vgData.at[i, 'Release Date'].replace(year = vgI
vgData['Last Update'] = pd.to_datetime(vgData['Last Update'], errors='coerce')
#Remove any titles with null Release Dates
vgData = vgData.drop(vgData[pd.isnull(vgData['Release Date'])].index)

#This table includes some entries representing entire series as opposed to titles and
#to compare individual game sales to entire series (the individual members of the ser
#It also includes some listings that amalgamate a game's sales across all consoles
#but this could interfere with speculation about consoles later on, so it is easier t
for i, row in vgData.iterrows():
    if vgData.at[i, 'Console'] == 'Series' or vgData.at[i, 'Console'] == 'All':
        vgData.drop(i)

#Print the head again now that it's nice and clean
print(vgData.head(25))
```

```
                                           Title Console  \
0                                     Wii Sports     Wii
1                              Super Mario Bros.     NES
2                                 Mario Kart Wii     Wii
3                              Wii Sports Resort     Wii
4             Pokémon Red / Green / Blue Version      GB
5                                         Tetris      GB
6                          New Super Mario Bros.      DS
7                                       Wii Play     Wii
8                      New Super Mario Bros. Wii     Wii
9                                      Duck Hunt     NES
10                                    Nintendogs      DS
11                                 Mario Kart DS      DS
12                 Pokémon Gold / Silver Version      GB
13                                       Wii Fit     Wii
14                            Kinect Adventures!    X360
15                                  Wii Fit Plus     Wii
16                 Grand Theft Auto: San Andreas     PS2
17                             Super Mario World    SNES
18                            Grand Theft Auto V     PS3
19  Brain Age: Train Your Brain in Minutes a Day      DS
20                            Grand Theft Auto V     PS4
21               Pokémon Diamond / Pearl Version      DS
22                              Super Mario Land      GB
23                           Super Mario Bros. 3     NES
24                   Grand Theft Auto: Vice City     PS2

                 Publisher             Developer  Critic Score  User Score  \
0                 Nintendo          Nintendo EAD           7.7         NaN
1                 Nintendo          Nintendo EAD          10.0         8.2
2                 Nintendo          Nintendo EAD           8.2         9.1
3                 Nintendo          Nintendo EAD           8.0         8.8
4                 Nintendo            Game Freak           9.4         NaN
5                 Nintendo  Bullet Proof Software          NaN         NaN
6                 Nintendo          Nintendo EAD           9.1         8.1
7                 Nintendo          Nintendo EAD           5.9         4.5
8                 Nintendo          Nintendo EAD           8.6         9.2
9                 Nintendo          Nintendo R&D1          NaN         NaN
10                Nintendo          Nintendo EAD           8.4         NaN
11                Nintendo          Nintendo EAD           9.1         9.4
12                Nintendo            Game Freak           9.2         NaN
13                Nintendo          Nintendo EAD           7.9         NaN
14  Microsoft Game Studios    Good Science Studio           6.7         NaN
15                Nintendo          Nintendo EAD           8.0         NaN
16          Rockstar Games        Rockstar North           9.5         NaN
17                Nintendo          Nintendo EAD           8.5         NaN
18          Rockstar Games        Rockstar North           9.4         NaN
19                Nintendo          Nintendo SDD           8.1         NaN
20          Rockstar Games        Rockstar North           9.7         NaN
21                Nintendo            Game Freak           8.6         NaN
22                Nintendo          Nintendo R&D1          NaN         NaN
23                Nintendo          Nintendo R&D2          NaN         NaN
24          Rockstar Games        Rockstar North           9.6         NaN

   Total Sales NA Sales PAL Sales Japan Sales Other Sales Release Date  \
0        82.65    41.36     29.02        3.77        8.51   2006-11-19
1        40.24    29.08      3.58        6.81        0.77   1985-10-18
2        35.98    15.91     12.92         3.8        3.35   2008-04-27
3         32.9    15.61     10.99        3.29        3.02   2009-07-26
4        31.37    11.27      8.89       10.22         1.0   1998-09-30
5        30.26     23.2      2.26        4.22        0.58   1989-07-31
6        29.85    11.28      9.19         6.5        2.89   2006-05-15
7        28.92    13.96      9.18        2.93        2.85   2007-02-12
8        28.51    14.53      7.01         4.7        2.27   2009-11-15
9        28.31    26.93      0.63        0.28        0.47   1985-10-15
10       24.68     9.05     10.96        1.93        2.74   2005-08-23
11       23.26     9.71      7.51        4.13        1.91   2005-11-14
12        23.1      9.0      6.18         7.2        0.71   2000-10-14
13        22.7     8.92      8.03         3.6        2.15   2008-05-21
14        22.1    15.09      4.97        0.24        1.81   2010-11-04
15       21.81     9.01      8.51        2.53        1.76   2009-10-04
16       20.81     9.43       0.4        0.41       10.57   2004-10-26
17       20.61    12.78      3.75        3.54        0.55   1991-08-13
18       20.32     6.37      9.85        0.99        3.12   2013-09-17
19       20.16     4.74      9.21        4.16        2.04   2006-04-17
20       19.39     6.06      9.71         0.6        3.02   2014-11-18
21       18.25     6.38      4.47        6.04        1.36   2007-04-28
22       18.14    10.83      2.71        4.18        0.42   1989-08-01
23       17.28     9.54      3.44        3.84        0.46   1990-02-12
24       16.15     8.41      5.49        0.47        1.78   2002-10-28

   Last Update
0          NaT
1          NaT
2   2018-04-11
3          NaT
4          NaT
5          NaT
6          NaT
7          NaT
8          NaT
9          NaT
10         NaT
11         NaT
12         NaT
13         NaT
14  2018-09-29
15         NaT
16         NaT
17  2018-08-10
18         NaT
19         NaT
20  2018-01-03
21         NaT
22         NaT
23  2018-08-09
24         NaT
```

Viewing the Industry as a Whole

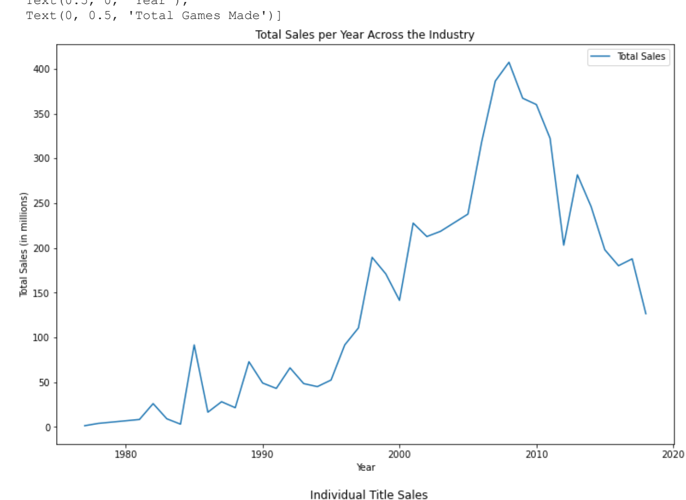
Now that the data is settled, let's start by taking a look at the industry as a whole, as opposed to looking at individual titles, publishers, or developers. That way we can learn about general sales trends in the industry over time, get a better sense of the magnitude of the video game industry, and better understand this specific dataset.\ \ **Calculating the Yearly Stats:** \ Calculating data in terms of the year is a great way to measure the change over time. To expedite the process of totaling up several values for every year, the function "yearly_total(col, count)" will take in the name of a column and return a dataframe that contains the sum of that column for each year. \ The count parameter is just for a special case where instead of summing it just counts how many entries that column ha for each year. \ For example, yearly_total('Total Sales', False) would give a dataframe indexed by years with a column of the total sales across all titles each year. \ With this new function, we can calculate the total sales from games released each year and the total number of games created per year. \ \ **Graphing the Industry:** \ Once we have the means to calculate the totals, we can graph them to have a better look at them. In this case, we create a line graph of the total sales from games released each year, a scatterplot of each title's sales and release date, and a line graph of the total number of games released each year. \ What these graphs tell us is that there is a large climb overall, indicating there are more video games being released and sold as the industry grows. You may notice that there is a large peak just before 2010, which might seem to imply that gaming is dying out, but what it really shows is that our dataset is biased against newer entries, which have had less time to get and report clean sales estimates. Also, the scatterplot shows that there are several very strong outliers. Sometimes outside factors can affect a games' sales, for example, the highest point overall belongs to Wii Sports, which technically sold a huge ammount of copies because it was included with the Nintendo Wii.

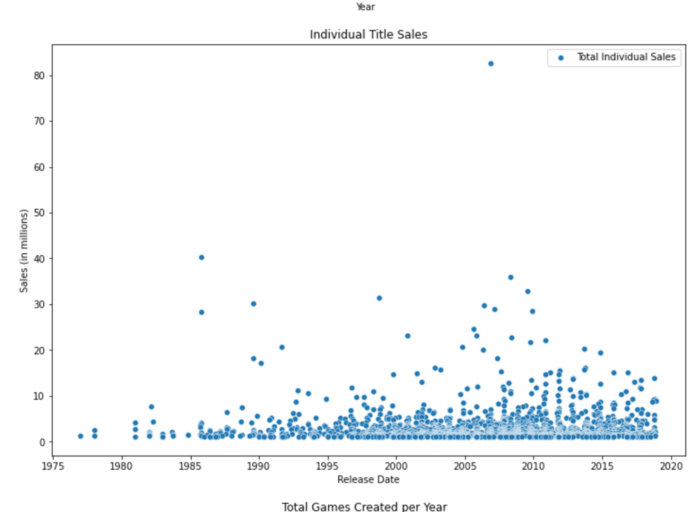
In [178…

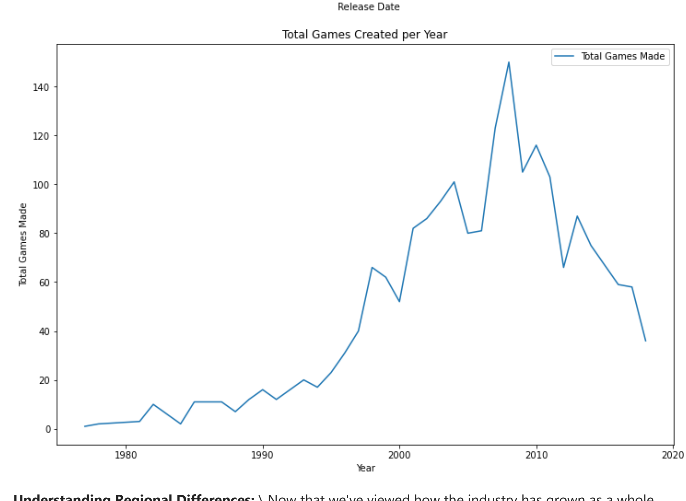
```python
#Generates a sum of the values in col for each year
#Count switches from a sum to a count of entries
def yearly_total(col, count):
    yearly = {}
    for i, row in vgData.iterrows():
        year = vgData.at[i,'Release Date'].year
        if not np.isnan(year):
            year = int(year)
            if not year in yearly:
                yearly[year] = 0
            if count:
                yearly[year] += 1
            else:
                if not np.isnan(vgData.at[i, col]):
                    yearly[year] += vgData.at[i, col]

    yearly = pd.DataFrame.from_dict(yearly, orient = 'index')
    yearly = yearly.sort_index()
    yearly.columns = [col]
    yearly.index.set_names(["Year"])
    return yearly

#Calculate the total sales from games released that year, then graph it
yearly_sales = yearly_total("Total Sales", False)
tsalesline, ax1 = plt.subplots()
tsalesline.set_size_inches(12, 8)
sns.lineplot(x = yearly_sales.index, y = yearly_sales["Total Sales"], label="Total Sa

#Graph each individual title by its release date and total sales
tsalesscatter, ax2 = plt.subplots()
tsalesscatter.set_size_inches(12, 8)
sns.scatterplot(x=vgData['Release Date'],y=vgData['Total Sales'], label="Total Indivi

#Calculate the total number of games released per year, then graph it
yearly_games = yearly_total("Total Sales", True)
yearly_games.columns = ["Total Games"]
tgsalesline, ax3 = plt.subplots()
tgsalesline.set_size_inches(12, 8)
sns.lineplot(x = yearly_games.index, y = yearly_games["Total Games"], label="Total Gam
```

Out[178…

```
[Text(0.5, 1.0, 'Total Games Created per Year'),
 Text(0.5, 0, 'Year'),
 Text(0, 0.5, 'Total Games Made')]
```







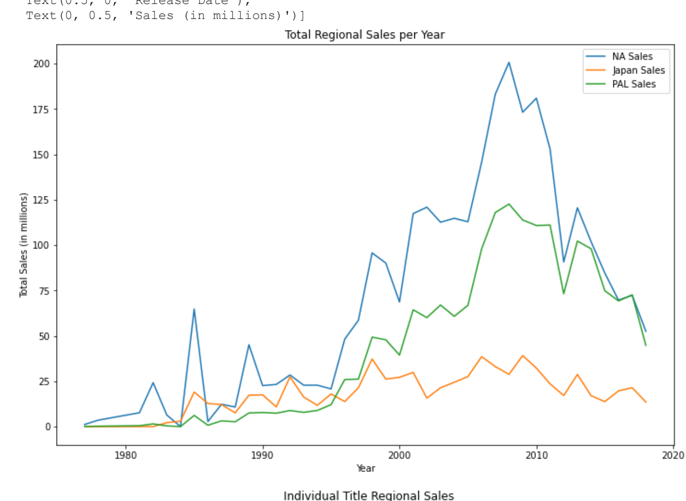
Understanding Regional Differences: \ Now that we've viewed how the industry has grown as a whole, we can take a closer look to see if there are any significant regional differences. \ This dataset includes three main sales regions: \ **North America**, which, as you might assume, covers North America, particularly the US, Canada, and Mexico. It is significant because of the US' strong prescence in the video game industry. \ **Japan**, which, again fairly self explanatory, covers Japan. Not unlike the US, Japan also plays a major role in both video game development and purchases, with possibly the most notable company being Nintendo, who has long been a major player in video game and video game console development. \ **PAL**, which may be less obvious. The PAL region covers multiple continents, ranging from Asia to Africa to Europe, South America, and Oceania. It was a designation that originally was used in broadcasting as those regions used the "Phase Alternating Line" television standard. Now many of the countries in the region no longer use the standard, as it was outdated (and actually significantly hindered video game performance), but in terms of video games the region is still based on the old members. \ \ Now that the regions are established, we can graph them. \ First we plot the sales per region per year. This graph shows us two things especially. One, that all three regions follow the same peak structure around 2010, which adds to the previous idea of it likely being a bias in the data as opposed to something like a spike in a specific regional market. Secondly, it shows the scale of each market, with NA being the largest market, PAL following second, and Japan ranking third. \ The second plot shows how individual titles rank in sales in each region, with the data once again showing the pre-2010 peak (which, the scatterplot shows is largely due to several high performing individual titles, such as Wii Sports).

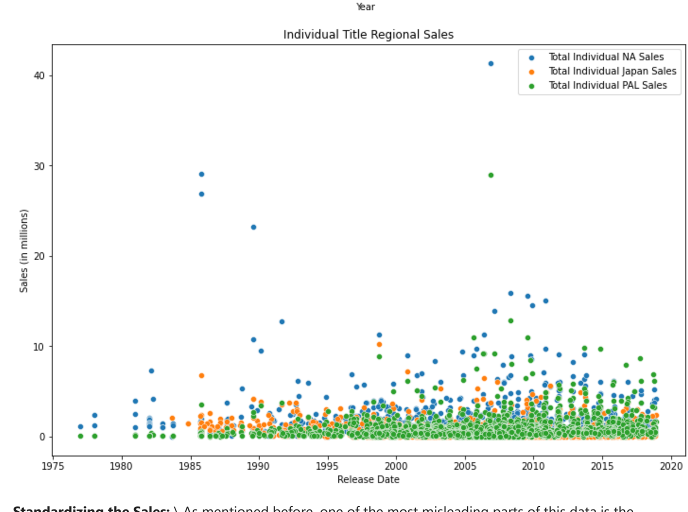
In [179…

```python
#Using the yearly_total formula from before find the yearly totals for each region
yearly_NA_sales = yearly_total("NA Sales", False)
yearly_JP_sales = yearly_total("Japan Sales", False)
yearly_PAL_sales = yearly_total("PAL Sales", False)

#Set up the first graph, which plots each region's sales by year
trsalesline, ax4 = plt.subplots()
trsalesline.set_size_inches(12, 8)
sns.lineplot(x = yearly_NA_sales.index, y = yearly_NA_sales["NA Sales"], label="NA Sa
sns.lineplot(x = yearly_JP_sales.index, y = yearly_JP_sales["Japan Sales"], label="Ja
sns.lineplot(x = yearly_PAL_sales.index, y = yearly_PAL_sales["PAL Sales"], label="PA

#Sets up the second graph, which plots individual titles by their release date and re
trsalesscatter, ax5 = plt.subplots()
trsalesscatter.set_size_inches(12, 8)
sns.scatterplot(x=vgData['Release Date'],y=vgData['NA Sales'], label="Total Individua
sns.scatterplot(x=vgData['Release Date'],y=vgData['Japan Sales'], label="Total Indivi
sns.scatterplot(x=vgData['Release Date'],y=vgData['PAL Sales'], label="Total Individu
```

Out[179…

```
[Text(0.5, 1.0, 'Individual Title Regional Sales'),
 Text(0.5, 0, 'Release Date'),
 Text(0, 0.5, 'Sales (in millions)')]
```





Standardizing the Sales: \ As mentioned before, one of the most misleading parts of this data is the influence that a couple hit sellers have on the total sales. To combat this, we can standardize the sales with the mean and standard deviation to reframe each game's performance in relation to its peers, as opposed to raw sales numbers. \ The formula for this will be: \

$$StandardizedSales = \frac{(IndividualSales - YearlyMeanSales)}{YearlySalesStandardDeviation}$$

In [180…

```python
#Combines all the yearly metrics into one year-indexed dataframe
yearly_combined = yearly_sales
yearlies = [yearly_games, yearly_NA_sales, yearly_JP_sales, yearly_PAL_sales]
for x in yearlies:
    yearly_combined = yearly_combined.join(x)

#Calculate the yearly mean by dividing the total sales by the total number of games r
yearly_combined['Total Average'] = yearly_combined['Total Sales']/yearly_combined['To

#Grab every game for every year to find the standard deviation in sales for that year
for i, row in yearly_combined.iterrows():
    games = []
    for j, row2 in vgData.iterrows():
        if vgData.at[j, 'Release Date'].year == i:
            games.append(vgData.at[j, 'Total Sales'])
    stdev = np.std(games)
    yearly_combined.at[i,'Total Stdev'] = stdev

#Print out the new yearly dataframe to check it looks right
print(yearly_combined.head(10))

#Calculate a new row of adjusted total sales
for i, row in vgData.iterrows():
    if not np.isnan(vgData.at[i, 'Release Date'].year):
        vgData.at[i,'Total Sales Adj'] = (vgData.at[i, 'Total Sales'] - yearly_combine

#Print out the dataframe with the new column to make sure it all looks right
print(vgData.head(10))
```

```
      Total Sales  Total Games  NA Sales  Japan Sales  PAL Sales  \
1977         1.25            1      1.17         0.00       0.07
1978         3.83            2      3.57         0.00       0.22
1981         8.22            3      7.63         0.00       0.50
1982        25.95           10     24.21         0.00       1.47
1983         8.98            6      6.38         2.13       0.39
1984         2.97            2      0.00         2.97       0.00
1985        91.53           11     64.79        19.01       6.17
1986        16.36           11      2.79        12.73       0.74
1987        28.05           11     12.31        12.12       3.15
1988        21.34            7     10.77         7.57       2.66

      Total Average  Total Stdev
1977       1.250000     0.000000
1978       1.915000     0.615000
1981       2.740000     1.290142
1982       2.595000     1.950391
1983       1.496667     0.346442
1984       1.485000     0.035000
1985       8.320909    12.536178
1986       1.487273     0.494462
1987       2.550000     1.386657
1988       3.048571     2.026851
                                Title Console Publisher  \
0                          Wii Sports     Wii  Nintendo
1                   Super Mario Bros.     NES  Nintendo
2                      Mario Kart Wii     Wii  Nintendo
3                   Wii Sports Resort     Wii  Nintendo
4  Pokémon Red / Green / Blue Version      GB  Nintendo
5                              Tetris      GB  Nintendo
6               New Super Mario Bros.      DS  Nintendo
7                            Wii Play     Wii  Nintendo
8           New Super Mario Bros. Wii     Wii  Nintendo
9                           Duck Hunt     NES  Nintendo

               Developer  Critic Score  User Score Total Sales NA Sales  \
0           Nintendo EAD           7.7         NaN       82.65    41.36
1           Nintendo EAD          10.0         8.2       40.24    29.08
2           Nintendo EAD           8.2         9.1       35.98    15.91
3           Nintendo EAD           8.0         8.8        32.9    15.61
4             Game Freak           9.4         NaN       31.37    11.27
5  Bullet Proof Software           NaN         NaN       30.26     23.2
6           Nintendo EAD           9.1         8.1       29.85    11.28
7           Nintendo EAD           5.9         4.5       28.92    13.96
8           Nintendo EAD           8.6         9.2       28.51    14.53
9          Nintendo R&D1           NaN         NaN       28.31    26.93

  PAL Sales Japan Sales Other Sales Release Date Last Update  Total Sales Adj
0     29.02        3.77        8.51   2006-11-19         NaT         8.192748
1      3.58        6.81        0.77   1985-10-18         NaT         2.546158
2     12.92         3.8        3.35   2008-04-27  2018-04-11         9.114139
3     10.99        3.29        3.02   2009-07-26         NaT         6.283591
4      8.89       10.22         1.0   1998-09-30         NaT         7.148735
5      2.26        4.22        0.58   1989-07-31         NaT         2.825718
6      9.19         6.5        2.89   2006-05-15         NaT         2.697931
7      9.18        2.93        2.85   2007-02-12         NaT         7.221932
8      7.01         4.7        2.27   2009-11-15         NaT         5.345430
9      0.63        0.28        0.47   1985-10-15         NaT         1.594512
```

```
<ipython-input-180-576e5d78554f>:25: RuntimeWarning: invalid value encountered in doub
le_scalars
  vgData.at[i,'Total Sales Adj'] = (vgData.at[i, 'Total Sales'] - yearly_combined.at[v
gData.at[i, 'Release Date'].year, 'Total Average']) / yearly_combined.at[vgData.at[i,
'Release Date'].year, 'Total Stdev']
```

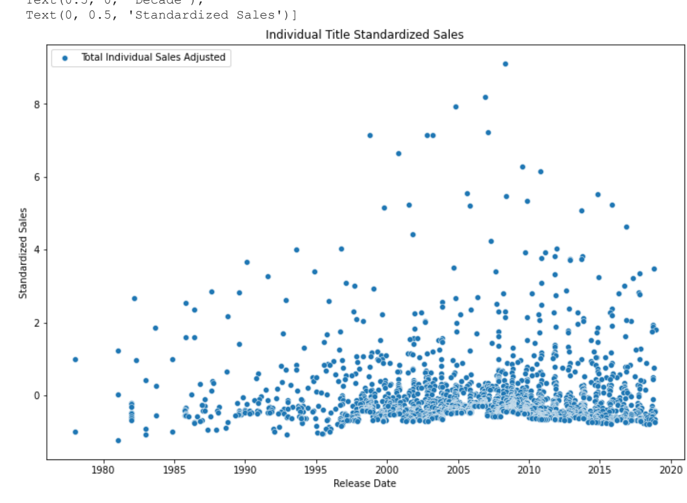
Graphing Again: \ Now that we have the adjusted sales values, we can regraph the scatterplot to get a better sense of the data spread. \ From the scatterplot, we can see that there are many values that congregate around the mean, and a relatively smaller ammount of values that rise significantly above it. What this means in terms of video game sales is that the majority of games sell a normal ammount, while a small percentage find huge success.\ \ We can push this idea furth by introducing violin graphs. \ By grouping each title by its decade, we can better see how sales vary within a set time period. \ Agreeing with what the scatterplot indicated before, the violin plot shows that for every decade the vast majority of values are close to the mean, with a thin trail of positive outliers, representing games that sold exceptionally well.\ Even after cropping out some of the outliers in a second violin plot, the shapes remain the same, showing a density of points at the mean.\ \ These graphs show that historically, game sales are less of a linear trend, and more of a question of wether a game is considered an average title or a big hit. Instead of seeing a gradual trend of points above the mean, there is a stark contrast between the large number of points at the mean and those that sold well beyond it.

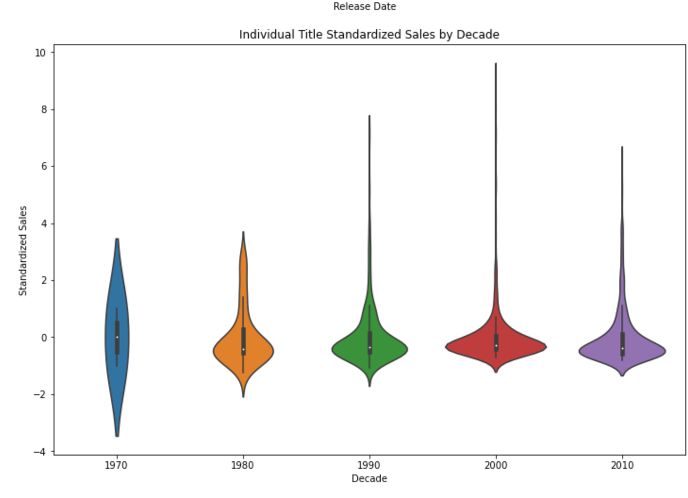
In [181…

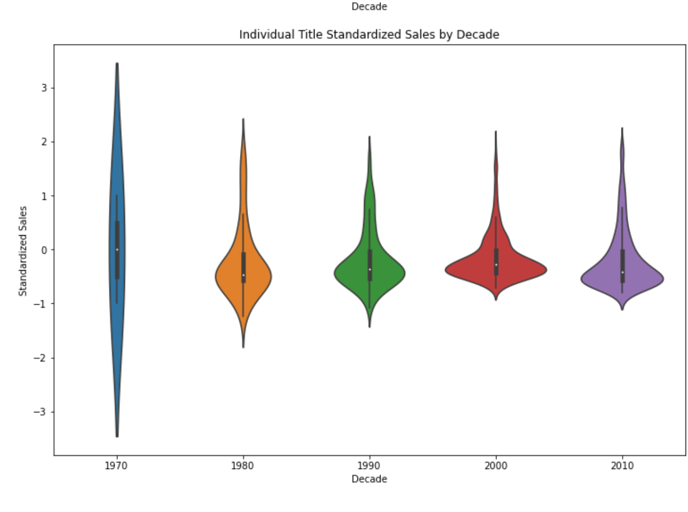
```python
#Graphing another scatterplot of individual values, this time with the adjusted total
tadjsalesscatter, ax6 = plt.subplots()
tadjsalesscatter.set_size_inches(12, 8)
sns.scatterplot(x=vgData['Release Date'],y=vgData['Total Sales Adj'], label="Total In

#Graphing a violin plot to see how much individual sales vary within a given decade
tadjsalesviolin, ax7 = plt.subplots()
tadjsalesviolin.set_size_inches(12, 8)
vgData['Decade'] = (vgData['Release Date'].dt.year - vgData['Release Date'].dt.year %
sns.violinplot(x=vgData['Decade'],y=vgData['Total Sales Adj'], label="Total Individua

#Graphing the same violin plot but with a more limited range to better cut out outlie
tadjsalesviolinzoomed, ax8 = plt.subplots()
tadjsalesviolinzoomed.set_size_inches(12, 8)
sns.violinplot(x=vgData.loc[vgData['Total Sales Adj'] < 2].loc[vgData['Total Sales Ad
```

Out[181…

```
[Text(0.5, 1.0, 'Individual Title Standardized Sales by Decade'),
 Text(0.5, 0, 'Decade'),
 Text(0, 0.5, 'Standardized Sales')]
```







Clash of the Industry Titans

A large part of any industry is obviously going to be the major companies involved. The gaming industry is no different. People have famously joked about "The Console Wars," where people will argue about which company's console is the best. Competition like that fuels favoritism, and you would be hard pressed to find a frequent video game player who doesn't have a favorite developer, whether it be a little known indie darling or a name brand like Nintendo. Even just identifying industry leaders through sales can prove helpful for understanding the industry as a whole. \ \ **Finding the Big Earners:** \ First, I have defined a similar function to the yearly_total() from before, called company_total, the difference being it will go through a column and add to a publisher company based total. \ With this function, we can easily find the total number of sales per company, which can in turn be sorted to find the top 10 companies by sales. \ Having some knowledge of the gaming industry, many of the top 10 are not surprising, with Nintendo being a household name, Activision publishing the famous Call of Duty Series, and Sony being the developers of the Playstation, one of the top leading console brands. But the metric of total sales is fairly skewed, particularly in favor of game companies that have been around long enough to rack up sales (like Nintendo) or have developed far more titles than other companies (like Nintendo). We will revisit this later.

In [182…

```
#Very similar function to yearly_total(), but this time it works with companies instea
def company_total(col, count):
    companies = {}
    for i, row in vgData.iterrows():
        publisher = vgData.at[i,'Publisher']
        if not publisher in companies:
            companies[publisher] = 0
        if count:
            companies[publisher] += 1
        else:
            if not np.isnan(vgData.at[i, col]):
                companies[publisher] += vgData.at[i, col]

    companies = pd.DataFrame.from_dict(companies, orient = 'index')
    companies = companies.sort_index()
    companies.columns = [col]
    companies.index.set_names(["Company"])
    return companies

#Uses company_total to find the Total Sales of each company, then sorts it to find the
company_sales = company_total('Total Sales', False)
company_sales = company_sales.sort_values(by='Total Sales',ascending=False)
print(company_sales.head(10))
```

```
                              Total Sales
Nintendo                          1741.05
Activision                         464.80
Electronic Arts                    427.91
Sony Computer Entertainment        390.69
EA Sports                          316.10
Ubisoft                            278.88
Rockstar Games                     245.12
Capcom                             165.13
THQ                                143.64
Sega                               128.58
```

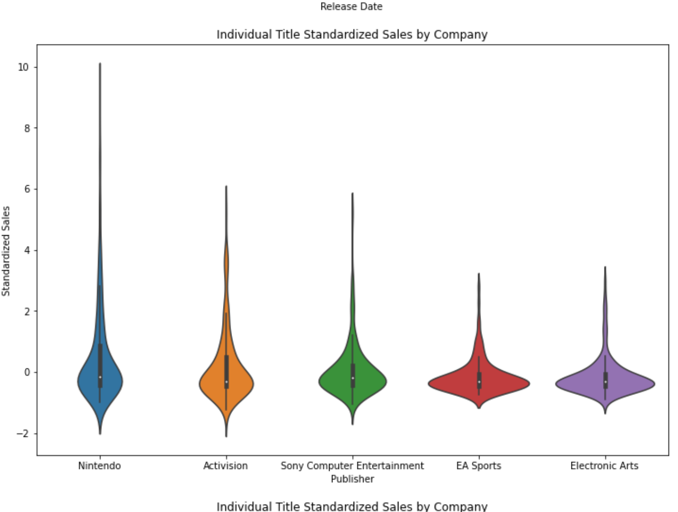
Graphing the Big Earners: \ With this list of the top 10 sellers, we can now graph just their sales as opposed to the whole industry. \ \ First, we graph a new scatterplot, which shows each companies' titles' sales (not standardized sales, just total sales). This graph shows that even the top companies follow the same trend of primarily normal-performing titles with several high performing hits. \ Then, we graph two violin plots (split up for visibility) to better represent the distribution of sales for each companies' titles (this time using the standardized sales to emphasize the variation of sales). Again this corroborates with what we've seen so far, the majority of titles sell around the mean with a few hits that sell hard. Just like the rest of the industry, the big players in the industry perform the same.

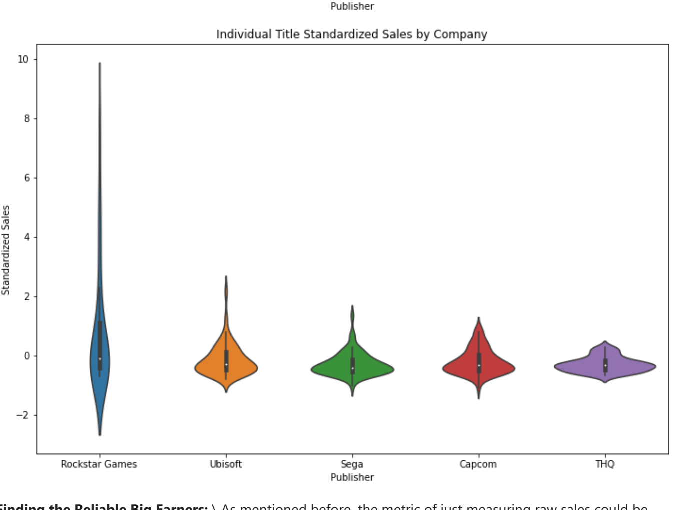
In [183…

```
#Isolate the highest 10 companies as the top_companies
top_companies = company_sales.head(10).index

#Function to check if a company is in the top 10
def top_company_check(name, start=0, end=10):
    for comp in top_companies[start:end]:
        if comp == name:
            return True
    return False

#Create new dataframe that has only the top companies in it
topc_vgData = vgData.loc[vgData['Publisher'].apply(top_company_check)]

#Graph a scatterplot of each companies' titles
toptadjsalesline, ax11 = plt.subplots()
toptadjsalesline.set_size_inches(12, 8)
for i in range(0,10):
    sns.scatterplot(x=topc_vgData.loc[topc_vgData['Publisher']==top_companies[i]]['Re

#Graph a violin plot for each companies' titles (only graphs 5 at a time for visibili
toptadjsalesviolin, ax9 = plt.subplots()
toptadjsalesviolin.set_size_inches(12, 8)
sns.violinplot(x=topc_vgData.loc[topc_vgData['Publisher'].apply(lambda x: top_company_
#Second violin graph
toptadjsalesviolin2, ax10 = plt.subplots()
toptadjsalesviolin2.set_size_inches(12, 8)
sns.violinplot(x=topc_vgData.loc[topc_vgData['Publisher'].apply(lambda x: top_company_
```

Out[183…

```
[Text(0.5, 1.0, 'Individual Title Standardized Sales by Company'),
 Text(0.5, 0, 'Publisher'),
 Text(0, 0.5, 'Standardized Sales')]
```





Finding the Reliable Big Earners: \ As mentioned before, the metric of just measuring raw sales could be biased as some companies have simply released more titles than others, and by our current metric if a company released 100 mediocre games they would beat a company that released 5 really good games. Instead, we can take the average sales/game per company by dividing each publisher's total sales by the total number of games they have produced. \ \ After doing this, we can see that the list has significantly changed with new companies entering the list and some companies falling off. This metric better protects against companies that rely on just pushing out enough normal-performing games as opposed to reliably releasing high earners. However, several members remain on both lists, such as Nintendo and Rockstar Games (most known as the creators of the Grand Theft Auto series).

In [184…

```
#Calculate the number of games produced per company and then combines it with the tota
#into a brand new company dataframe
company_gamecount = company_total("Total Sales", True)
company_gamecount.columns = ["Games Produced"]
company_totals = company_sales.join(company_gamecount)

#Calculates the average Sales/Game
company_totals['Average Sales/Game'] = company_totals["Total Sales"]/company_totals["(

#Prints the Average Sales/Game of the current top 10 companies by sales
print("Top 10 Companies by Sales' Average Sales/Game:")
top10sales = company_totals.head(10)
print(top10sales)

#Prints the top 10 companies by Average Sales/Game
print("\nTop 10 Companies by Average Sales/Game:")
company_totals = company_totals.sort_values(by='Average Sales/Game', ascending=False)
top10asg = company_totals.head(10)
print(top10asg)

#Finds and prints any companies that lead in both total sales and average sales/game
consistent_champs = []
for i in top10sales.index:
    for j in top10asg.index:
        if not i in consistent_champs and i == j:
            consistent_champs.append(i)
print("\nConsistant Top Performers:")
print(consistent_champs)
```

```
Top 10 Companies by Sales' Average Sales/Game:
                              Total Sales  Games Produced  Average Sales/Game
Nintendo                          1741.05             344            5.061192
Activision                         464.80             132            3.521212
Electronic Arts                    427.91             180            2.377278
Sony Computer Entertainment        390.69             129            3.028605
EA Sports                          316.10             128            2.469531
Ubisoft                            278.88             106            2.630943
Rockstar Games                     245.12              47            5.215319
Capcom                             165.13              70            2.359000
THQ                                143.64              76            1.890000
Sega                               128.58              61            2.107869

Top 10 Companies by Average Sales/Game:
                                      Total Sales  Games Produced  \
Rockstar Games                             245.12              47
Nintendo                                  1741.05             344
Microsoft Game Studios                     120.10              25
Sony Computer Entertainment America         18.33               4
Square EA                                   18.22               4
Sony Interactive Entertainment              46.96              12
Activision                                 464.80             132
Valve Corporation                            3.52               1
RedOctane                                   27.18               8
Microsoft                                   16.92               5

                                      Average Sales/Game
Rockstar Games                                  5.215319
Nintendo                                        5.061192
Microsoft Game Studios                          4.804000
Sony Computer Entertainment America             4.582500
Square EA                                       4.555000
Sony Interactive Entertainment                  3.913333
Activision                                      3.521212
Valve Corporation                               3.520000
RedOctane                                       3.397500
Microsoft                                       3.384000

Consistant Top Performers:
['Nintendo', 'Activision', 'Rockstar Games']
```

Predicting (or at least tricking a computer into predicting)

Now that we have a better understanding of the industry, we can try training a model to predict roughly how well a game will sell given some basic information about it. A model like this could be useful in the real world for predicting the potential success of a game before or during development (although from an artistic standpoint I really hope there is more driving the creation of a game than a computer telling a company they can make money). \ \ **Preparing the Training Data:** \ From the dataset we have been working with, it wouldn't make sense to predict a game's sales with regional sales data (because it wouldn't be much of a prediction more just basic addition). The model has to be limited to data that a game would have before or at least close to launch. The console, publisher, developer, and the release year can all be determined ahead of time, and critic scores are often released close enough to launch as critics get their copies ahead of time that its fair to predict from. \ \ To prepare the data for training, each of the aforementioned columns is added to a dataframe, and the console, publisher, and developer are all encoded with one-hot encoding in order to be compatible with sci-kit learn's models (this results in a comically large dataframe). The release date is simplified down to a year, and the critic score is set to 0 if the game does not have one.\ The standardized total sales (which are being predicted in this case) are labeled as "Strongly Positive", "Positive", "Negative", or "Strongly Negative"\ After all the data is prepared, the data is split into 75% training data and 25% testing data

In [185…

```
#Begin preparing the X data by one hot encoding
X = vgData[['Console', 'Publisher', 'Developer', 'Critic Score', 'Release Date']]
X = X.join(pd.get_dummies(X['Console']))
X.pop('Console')
X = X.join(pd.get_dummies(X['Publisher']),lsuffix="Pub")
X.pop('Publisher')
X = X.join(pd.get_dummies(X['Developer']),lsuffix="Dev")
X.pop('Developer')
#Simplify release date to year
X['Release Date'] = X['Release Date'].dt.year
#Replace NaN critic scores with 0
X['Critic Score'] = X['Critic Score'].apply(lambda x: 0 if np.isnan(x) else x)
X.fillna(0, inplace=True)

#label the standardized sales to be guessed
def label_sales(sales):
    if sales > 2:
        return "Strongly Positive"
    elif sales > 0:
        return "Positive"
    elif sales > -2:
        return "Negative"
    else:
        return "Strongly Negative"
ts = vgData['Total Sales Adj'].apply(label_sales)
ts.fillna(0, inplace=True)

# Splits up the data for training (75%) and testing (25%)
X_train, X_test, ts_train, ts_test = train_test_split(X, ts, test_size = 0.25)
```

Model 1: Random Forest: \ This model comes from sci-kit learn and works by fitting several decision trees to different portions of the training data. It makes sense for this goal because of the decent number of features being inputted into the model (random forests work best with more features). The n_estimators is optimized using a GridSearchCV.

In [186…

```
#Training the random forest and predicting with it
forest_tuned_parameters = [{'n_estimators' : range(20,200,40)}]
forest = GridSearchCV(RandomForestClassifier(), forest_tuned_parameters, cv=10)
#forest = RandomForestClassifier(n_estimators = 200)
forest.fit(X_train, ts_train)
forest_predictions = forest.predict(X_test)
```

```
C:\Users\bloo1\anaconda3\lib\site-packages\sklearn\model_selection\_split.py:666: User
Warning: The least populated class in y has only 1 members, which is less than n_split
s=10.
  warnings.warn(("The least populated class in y has only %d"
```

Model 2: K Nearest Neighbors: \ This model comes from sci-kit learn and it works by identifying the k nearest datapoints or 'neighbors' to a point and classifying based on their values. It fits for this project because it accounts for brand value in that if a game is made by the same developer and publisher as a previous game, it is likely a sequel or in the same brand. The n_neighbors parameter is optimized via a GridSearchCV.

In [187…

```
#Training the k nearest neighbors and determine the best value for k
tuned_parameters = [{'n_neighbors' : range(1,100,10)}]
knnr = GridSearchCV(KNeighborsClassifier(), tuned_parameters, cv=10)
knnr.fit(X_train, ts_train);
```

```
C:\Users\bloo1\anaconda3\lib\site-packages\sklearn\model_selection\_split.py:666: User
Warning: The least populated class in y has only 1 members, which is less than n_split
s=10.
  warnings.warn(("The least populated class in y has only %d"
```

Judgement Day: \ After running both a manual check on the random forest and the 10-fold cross-validation on both models, both prove to work fairly effectively, earning an average score of above 70% and low standard deviations. The random forest model seems to be more effective overall, but both prove quite effective overall. This shows that it is possible to predict game sales based on prior history given some descriptive features of the game.

In [188…

```
#Manually check the accuracy of the random forest
forest_prediction_accuracy = 0
for i in range(0, len(forest_predictions)):
    if forest_predictions[i] == ts_test.iloc[i]:
        forest_prediction_accuracy += 1
print("Manually Checked Random Forest Accuracy: ")
print(forest_prediction_accuracy/len(forest_predictions))

#Performs a 10-fold cross-validation on both the forest and the k nearest neighbors
tenfold = KFold(n_splits=10, shuffle=True)
forest_scores = cross_validate(forest, X, ts, cv=tenfold)
knnr_scores = cross_validate(knnr, X, ts, cv=tenfold)

# Prints the mean of each algorithm's scores
print("Average Random Forest Score", (np.mean(forest_scores['test_score'])))
print("Average K Nearest Neighbors Score", (np.mean(knnr_scores['test_score'])))
# Prints the standard deviation of each algorithm's scores
print("Standard Deviation of Random Forest's Scores", (np.std(forest_scores['test_sco
print("Standard Deviation of K Nearest Neighbors' Scores", (np.std(knnr_scores['test_s
```

```
Manually Checked Random Forest Accuracy:
0.754
```

```
C:\Users\bloo1\anaconda3\lib\site-packages\sklearn\model_selection\_split.py:666: User
Warning: The least populated class in y has only 1 members, which is less than n_split
s=10.
  warnings.warn(("The least populated class in y has only %d"
C:\Users\bloo1\anaconda3\lib\site-packages\sklearn\model_selection\_split.py:666: User
Warning: The least populated class in y has only 1 members, which is less than n_split
s=10.
  warnings.warn(("The least populated class in y has only %d"
C:\Users\bloo1\anaconda3\lib\site-packages\sklearn\model_selection\_split.py:666: User
Warning: The least populated class in y has only 1 members, which is less than n_split
s=10.
  warnings.warn(("The least populated class in y has only %d"
C:\Users\bloo1\anaconda3\lib\site-packages\sklearn\model_selection\_split.py:666: User
Warning: The least populated class in y has only 1 members, which is less than n_split
s=10.
  warnings.warn(("The least populated class in y has only %d"
C:\Users\bloo1\anaconda3\lib\site-packages\sklearn\model_selection\_split.py:666: User
Warning: The least populated class in y has only 1 members, which is less than n_split
s=10.
  warnings.warn(("The least populated class in y has only %d"
C:\Users\bloo1\anaconda3\lib\site-packages\sklearn\model_selection\_split.py:666: User
Warning: The least populated class in y has only 1 members, which is less than n_split
s=10.
  warnings.warn(("The least populated class in y has only %d"
C:\Users\bloo1\anaconda3\lib\site-packages\sklearn\model_selection\_split.py:666: User
Warning: The least populated class in y has only 1 members, which is less than n_split
s=10.
  warnings.warn(("The least populated class in y has only %d"
C:\Users\bloo1\anaconda3\lib\site-packages\sklearn\model_selection\_split.py:666: User
Warning: The least populated class in y has only 1 members, which is less than n_split
s=10.
  warnings.warn(("The least populated class in y has only %d"
C:\Users\bloo1\anaconda3\lib\site-packages\sklearn\model_selection\_split.py:666: User
Warning: The least populated class in y has only 1 members, which is less than n_split
s=10.
  warnings.warn(("The least populated class in y has only %d"
C:\Users\bloo1\anaconda3\lib\site-packages\sklearn\model_selection\_split.py:666: User
Warning: The least populated class in y has only 1 members, which is less than n_split
s=10.
  warnings.warn(("The least populated class in y has only %d"
C:\Users\bloo1\anaconda3\lib\site-packages\sklearn\model_selection\_split.py:666: User
Warning: The least populated class in y has only 1 members, which is less than n_split
s=10.
  warnings.warn(("The least populated class in y has only %d"
C:\Users\bloo1\anaconda3\lib\site-packages\sklearn\model_selection\_split.py:666: User
Warning: The least populated class in y has only 1 members, which is less than n_split
s=10.
  warnings.warn(("The least populated class in y has only %d"
C:\Users\bloo1\anaconda3\lib\site-packages\sklearn\model_selection\_split.py:666: User
Warning: The least populated class in y has only 1 members, which is less than n_split
s=10.
  warnings.warn(("The least populated class in y has only %d"
C:\Users\bloo1\anaconda3\lib\site-packages\sklearn\model_selection\_split.py:666: User
Warning: The least populated class in y has only 1 members, which is less than n_split
s=10.
  warnings.warn(("The least populated class in y has only %d"
C:\Users\bloo1\anaconda3\lib\site-packages\sklearn\model_selection\_split.py:666: User
Warning: The least populated class in y has only 1 members, which is less than n_split
s=10.
  warnings.warn(("The least populated class in y has only %d"
C:\Users\bloo1\anaconda3\lib\site-packages\sklearn\model_selection\_split.py:666: User
Warning: The least populated class in y has only 1 members, which is less than n_split
s=10.
  warnings.warn(("The least populated class in y has only %d"
C:\Users\bloo1\anaconda3\lib\site-packages\sklearn\model_selection\_split.py:666: User
Warning: The least populated class in y has only 1 members, which is less than n_split
s=10.
  warnings.warn(("The least populated class in y has only %d"
C:\Users\bloo1\anaconda3\lib\site-packages\sklearn\model_selection\_split.py:666: User
Warning: The least populated class in y has only 1 members, which is less than n_split
s=10.
  warnings.warn(("The least populated class in y has only %d"
```

```
Average Random Forest Score 0.7453718592964824
Average K Nearest Neighbors Score 0.7263743718592964
Standard Deviation of Random Forest's Scores 0.02444434333465721
Standard Deviation of K Nearest Neighbors' Scores 0.01909155622924868
```

Conclusions

After scraping, analyzing, and training with this video game sales data, I hope you have learned a lot, both about the video game industries' historical performance and (arguably more importantly) about the data science pipeline and the process of data analysis and model training. Being able to apply these skills in industries can be very beneficial, as shown by the model we trained, and I hope that you will be able to apply these skills to projects of your own, wether for profit or for fun.\ \ **Thank you for reading through this tutorial, and I hope you learned a lot! \ - Martin Blurton-Jones**

In [ ]: