Video Game Sales Analysis – Report
MTH412 Spring 2023

Instructor: James Livsey
Report Written by: Kevin Nguyen

Abstract

In this project, a dataset of video game sales ranging from 1980 to 2020 was explored and analyzed. The focus of this project was narrowed down to three main questions: are Sports games more profitable than Action games on average, are Nintendo's Platform games more profitable than their Role-Playing games on average, and how accurately are we able to model and predict the number of Action games sold in a year. To answer these questions, the methods of hypothesis testing, permutation testing, and linear regression were applied. There was strong evidence to suggest that Sports games are not more profitable than Action games, but there was marginal evidence to suggest that Nintendo's Platform games were not more profitable than their Role-Playing games. Future analysis should include more comprehensive data of Nintendo's games (i.e., more recent releases) to perhaps swing the margins one way or the other. Furthermore, it was found that a linear regression model does not accurately capture the sales trajectory of Action games, leading to the use of a cubic regression model that described the data very well but was a poor predictor of future sales. As before, more recent data could be used to more accurately describe the performance of the model as a predictor.

Data and Motivation

The data used for analysis was the "Video Games Sales" dataset, which was uploaded to Kaggle (https://www.kaggle.com/datasets/ulrikthygepedersen/video-games-sales). The dataset contains information about 16,600 different games, including their name, publisher, release year, platform, genre, rank, and sales by region (North America, Europe, Japan, or other), and global sales.

The release years range from 1980 to 2020, but after processing the data, some years were missing entries or had a single entry with negligible sales data. To deal with this problem, the dataset was modified to not include such entries, which included the entries from the years 2017 and 2020 while the years 2018 and 2019 had no entries to begin with. There were also some entries with missing information: publisher, year, or both. In the case where the publisher was missing, it was filled in with "Unknown Publisher" and in the case where the year was missing, the entry was removed from the dataset entirely after checking they would be negligible, because it would have interfered with the goal of analyzing yearly sales data.

The sales categories measured how many copies of the games were sold, in millions. This category implies that this dataset is not entirely comprehensive of all games released, as it would not include any data for games that were free to download.

name	platform	year	genre	publisher	rank	na_sales	eu_sales	jp_sales	other_sales	global_sales
Wii Sports	Wii	2006.0	Sports	Nintendo	1	41.49	29.02	3.77	8.46	82.74
Super Mario Bros.	NES	1985.0	Platform	Nintendo	2	29.08	3.58	6.81	0.77	40.24
Mario Kart Wii	Wii	2008.0	Racing	Nintendo	3	15.85	12.88	3.79	3.31	35.82
Wii Sports Resort	Wii	2009.0	Sports	Nintendo	4	15.75	11.01	3.28	2.96	33.00
Pokemon Red/Pokemon Blue	GB	1996.0	Role-Playing	Nintendo	5	11.27	8.89	10.22	1.00	31.37

Figure 1. Top 5 video games, by global sales.

From the figure, it can be observed that the top publisher is Nintendo and North America was generally the better market for these top games. After performing the data cleaning, a numeric summary of the data can be seen in the next figure.

	year	rank	na_sales	eu_sales	jp_sales	other_sales	global_sales
count	15720.000000	15720.000000	15720.000000	15720.000000	15720.000000	15720.000000	15720.000000
mean	2006.298982	7984.755534	0.275555	0.153152	0.081497	0.050190	0.560686
std	5.852258	4614.625014	0.835647	0.517682	0.317173	0.193275	1.592144
min	1980.000000	1.000000	0.000000	0.000000	0.000000	0.000000	0.020000
25%	2003.000000	3984.750000	0.000000	0.000000	0.000000	0.000000	0.070000
50%	2007.000000	7986.500000	0.080000	0.030000	0.000000	0.010000	0.190000
75%	2010.000000	11977.500000	0.250000	0.120000	0.040000	0.040000	0.500000
max	2016.000000	15982.000000	41.490000	29.020000	10.220000	10.570000	82.740000

Figure 2. Numeric summary of the modified data.

It is worth noting that the data had many games towards the bottom of the list that had global_sales values of 0.01, which would drastically reduce the mean of global_sales which can be observed by the mean of approximately 0.5607. This observation is further reinforced by creating a histogram of global_sales, with a bin size of 2.

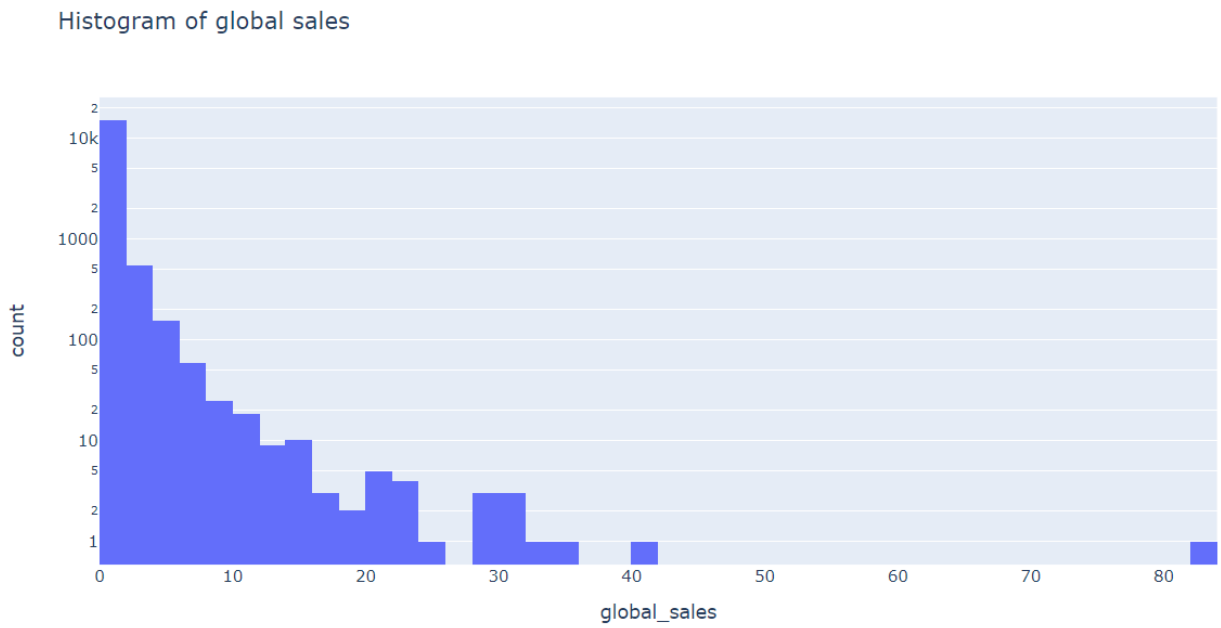

Figure 3. Histogram of global_sales, bin size of 2.

The histogram shows that most of the data lies in the range $[0, 10)$ with the modal bin being the range $[0, 2)$, which would explain why the mean of global_sales is a low number.

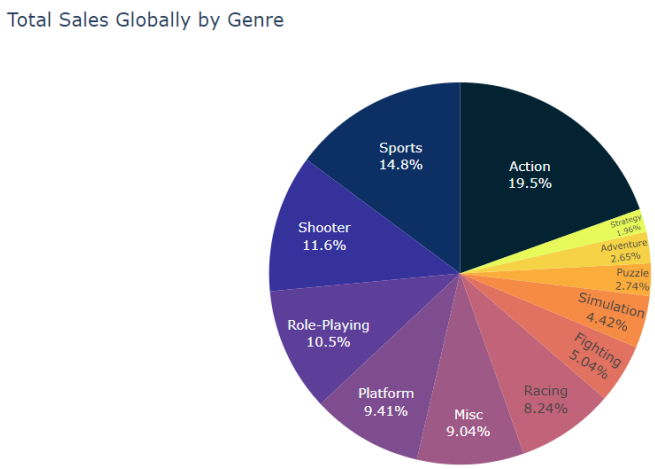

Figure 4. Pie chart of global sales share, by genre

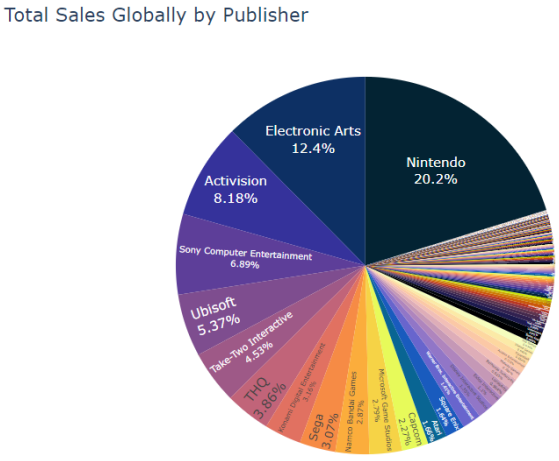

Figure 5. Pie chart of global sales share, by publisher

Figure 4 shows that the top two genres are Sports and Action, but there is a noticeable gap between the two genres' shares of sales. In fact, further exploration of the data shows that the average global sales of Action games is approximately 0.547 while the average of Sports games is about 0.580. Figure 5 shows that the top publisher is Nintendo by a wide margin. These two observations motivate a path forward for our explorations.

We will seek to answer three questions. First, we noticed that Action was on top in terms of total global sales followed by Sports games, but upon further inspection we found

that the average of Sports games was higher than Action games. This raises the question of whether Sports games are more profitable, on average, compared to Action games. Furthermore, the top publisher was Nintendo, but its average sales of each genre varied widely between genres. We will look at two genres: Platform and Role-Playing and try to determine if Platform games are more profitable, on average, than Role-Playing games for Nintendo. Lastly, as Action is the top genre, we will try to use its data to make predictions about its future yearly sales.

Methods

To help us answer the first two questions, we will be performing a hypothesis test. A hypothesis test tests a null hypothesis against an alternative hypothesis. A null hypothesis essentially claims that a difference in results is due to chance, while the alternative hypothesis claims that there is a statistically significant difference in results. After determining the null and alternative hypotheses, the distribution of a test statistic under the null hypothesis needs to be chosen as well. The distribution we will be using is the t-distribution, as we cannot assume the population standard deviation. Using this distribution, we will determine a critical region for which our null hypothesis should be rejected if the observed difference falls in this region. The critical region will be given by the significance level of the test.

To determine whether Sports games are more profitable than Action games on average, we can apply these principles. In this case, the null hypothesis would be that the difference in average global sales of the two genres are the same and the alternative hypothesis would be that the average global sales of Sports games is greater than the average global sales of Action games. We can restate this more precisely with mathematical symbols. Let μ_A be the average global sales of Action games, μ_S be the average global sales of Sports games, and H_0, H_A be the null and alternative hypotheses, respectively. Then,

$$H_0 : \mu_S = \mu_A$$

$$H_A : \mu_S > \mu_A$$

The setup to answer the second question is similar. Let N_P be the average global sales of Nintendo's Platform games and let N_R be the average global sales of Nintendo's Role-Playing games. Then the hypothesis test for this question is the.

$$H_0 : N_P = N_R$$

$$H_A : N_P > N_R$$

We will also conduct the hypothesis tests an alternative way, without a distribution. Using a permutation test, we can achieve essentially the same results as the hypothesis test. A permutation test reassigns labels of the data we are interested in and calculates a new observed difference with the shuffled labels. Doing this a large number of times, such as 9999, we can then obtain a p-value that is defined as the proportion of permutations with an observed difference larger than the original observed difference. If the p-value obtained is

smaller than the level of significance we choose, then we will reject the null hypothesis in favor of the alternative hypothesis.

The last problem of predicting future yearly Action game sales, we must use a different method. For this problem, we will use a linear regression model and a cubic regression model, then compare their accuracy and performance.

A linear regression model is a model which fits data points to a line. Mathematically, a line is determined by the equation $y = \alpha_0 + \alpha_1 x$, where y is the output, α_0 is the y-intercept, α_1 is the slope of the line, and x is the input. When we fit data to a line, we can calculate the residual of each prediction, which is defined as the distance between the actual data point and the predicted value. The lower the residual, the better fit the line has. Therefore, the line of best fit is the line with α_0 and α_1 which minimizes the square of these residuals.

Furthermore, the correlation coefficient, r, of the model determines the correlation between the two variables. It is bounded between -1 and 1, where a number close to 1 means that there is a strong positive correlation and a number close to -1 means there is a strong negative correlation. The square of the correlation coefficient, which is known as the coefficient of determination r^2, gives us information about how good of a fit the model is.

Besides this, we can further quantify the accuracy of the model using the squared error (SE) and root-mean-squared error (RMSE). The SE is simply a sum of the squares of the residuals while the RMSE is the square root of the SE divided by the number of data points, or $\sqrt{\frac{SE}{n}}$. The lower the SE and RMSE, the better fit the model has. The cubic regression model is similar, except it is defined as $y = b_0 + b_1 x + b_2 x^2 + b_3 x^3$. Since r (and r^2 by extension) do not exist for polynomial regression models (with degree greater than 1), we can only compare these two models using the SE and RMSE.

Results

For our hypothesis tests, a significance level of 95% was chosen. This means that if a p-value less than 0.05 was obtained, the null hypothesis should be rejected in favor of accepting the alternative hypothesis.

The first hypothesis test yielded $p = 0.2497$. At the significance level chosen, this strongly suggests that the null hypothesis is true. In other words, there is not enough evidence to conclude that Sports games are more profitable than Action games on average. The result of the permutation test was very similar, with a $p = 0.2478$, which leads to the same conclusion as the classic hypothesis test.

The second hypothesis test gives a little bit more hope towards favoring the alternative hypothesis. A p-value of 0.0669 was obtained and at the 95% level chosen, this would mean the alternative hypothesis would be rejected. However, due to how close p is to 0.05, this is only marginal evidence that Nintendo's Platform games were more profitable to their Role-

Playing games on average. Like for the first test, the permutation test had a similar result, with $p = 0.0655$. Perhaps more data could be included to round out the data set of Nintendo games, such as more recent data, to obtain a more confident result.

The final problem to discuss is the problem of fitting the yearly global sales of Action games to a linear and cubic regression model. From Figure 6, it can be observed that the data displays a linear relationship, but only up to about 2009. We can also see that the linear regression model will be unable to capture the drop in sales after the linear relationship seems to change directions, in other words the drop in sales after about 2010. Figure 7 shows the line of best fit obtained from the linear regression model.

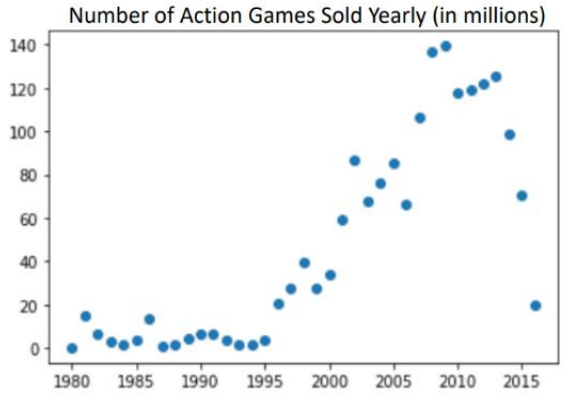

Figure 6. Scatter plot of action games sold per year

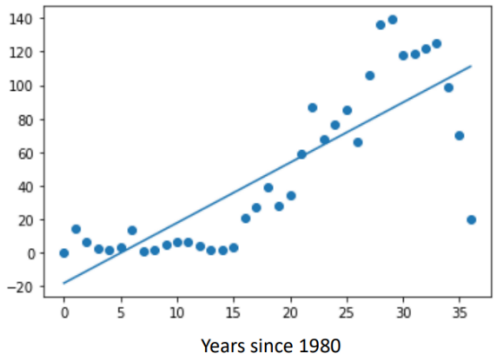

Figure 7. Line of best fit for action games sold per year

The equation representing this line is $y = 3.5923163x - 18.1257468$ and the correlation coefficient and coefficient of determination are $r = 0.8178$, $r^2 = 0.6688$ respectively. We can see that r, which describes the correlation in the data, is indeed positive and relatively close to 1. This is most likely due to the partially positive and negative linear relationship before 2009 and after 2010, respectively. However, the value of r^2 suggests that the model is only moderately strong at fitting the data.

The shape of our data suggests that a cubic regression may better fit the data. As we can see in Figure 8, this seems to be the case. Not only is the increase in sales captured, but the decrease in sales after 2010 is also captured by the curve. The equation of this curve is given by $y = -0.0203879606x^3 + 1.12811360x^2 - 13.0231436x + 31.2496471$.

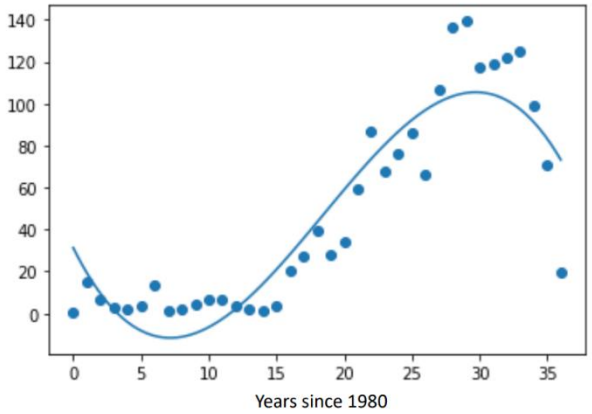

Figure 7. Cubic fit of action games sold per year

Calculating the residuals, we obtain a squared error of 12717.43 and a root-mean-squared error of 18.54. Doing the same for the linear regression model, we obtain $SE = 26949.84$ and $RMSE = 26.99$. Between the two results, the lower RMSE is 18.54, coming from the cubic fit. Does this indeed support our observation that the cubic fit was a better fit for the data than the linear fit. However, the cubic regression model assumes that the number of yearly sales for Action games will continue to decline after 2016. This may not be the case, though, and if we were to introduce more recent data into the dataset, it could be true that the number sold increased in recent years. This is the problem of overfitting our data: the cubic regression model is very fit and can well-describe the current dataset, but an introduction of new data may not be as predictable.

Appendix – Code

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import plotly.express as px
```

```
games_data = pd.read_csv("video_games_sales.csv")

# reorder columns of csv, so index does not precede ranking
cols = ['name', 'platform', 'year', 'genre', 'publisher',
        'rank', 'na_sales', 'eu_sales', 'jp_sales', 'other_sales', 'global_sales']
games_data = games_data[cols]
```

```
games_data
```

	name	platform	year	genre	publisher	rank	na_sales	eu_sales	jp_sales	other_sales	global_sales
0	Wii Sports	Wii	2006.0	Sports	Nintendo	1	41.49	29.02	3.77	8.46	82.74
1	Super Mario Bros.	NES	1985.0	Platform	Nintendo	2	29.08	3.58	6.81	0.77	40.24
2	Mario Kart Wii	Wii	2008.0	Racing	Nintendo	3	15.85	12.88	3.79	3.31	35.82
3	Wii Sports Resort	Wii	2009.0	Sports	Nintendo	4	15.75	11.01	3.28	2.96	33.00
4	Pokemon Red/Pokemon Blue	GB	1996.0	Role-Playing	Nintendo	5	11.27	8.89	10.22	1.00	31.37
...	...	...	...	...	...	...	...	...	...	...	...
16593	Woody Woodpecker in Crazy Castle 5	GBA	2002.0	Platform	Kemco	16596	0.01	0.00	0.00	0.00	0.01
16594	Men in Black II: Alien Escape	GC	2003.0	Shooter	Infogrames	16597	0.01	0.00	0.00	0.00	0.01
16595	SCORE International Baja 1000: The Official Game	PS2	2008.0	Racing	Activision	16598	0.00	0.00	0.00	0.00	0.01
16596	Know How 2	DS	2010.0	Puzzle	7G//AMES	16599	0.00	0.01	0.00	0.00	0.01
16597	Spirits & Spells	GBA	2003.0	Platform	Wanadoo	16600	0.01	0.00	0.00	0.00	0.01

```
# summary of each numeric column
games_data.describe()
```

	year	rank	na_sales	eu_sales	jp_sales	other_sales	global_sales
count	16327.000000	16598.000000	16598.000000	16598.000000	16598.000000	16598.000000	16598.000000
mean	2006.406443	8300.605254	0.264667	0.146652	0.077782	0.048063	0.537441
std	5.828981	4791.853933	0.816683	0.505351	0.309291	0.188588	1.555028
min	1980.000000	1.000000	0.000000	0.000000	0.000000	0.000000	0.010000
25%	2003.000000	4151.250000	0.000000	0.000000	0.000000	0.000000	0.060000
50%	2007.000000	8300.500000	0.080000	0.020000	0.000000	0.010000	0.170000
75%	2010.000000	12449.750000	0.240000	0.110000	0.040000	0.040000	0.470000
max	2020.000000	16600.000000	41.490000	29.020000	10.220000	10.570000	82.740000

```
# check for missing values
games_data.isnull().sum()

# missing years are 2018, 2019; 2020 and 2017 have negligible entries (single digit number)
```

```
name            0
platform        0
year          271
genre           0
publisher      58
rank            0
na_sales        0
eu_sales        0
jp_sales        0
other_sales     0
global_sales    0
dtype: int64
```

```
games_data.loc[games_data["year"] == 2020]
```

	name	platform	year	genre	publisher	rank	na_sales	eu_sales	jp_sales	other_sales	global_sales
5957	Imagine: Makeup Artist	DS	2020.0	Simulation	Ubisoft	5959	0.27	0.0	0.0	0.02	0.29

```
games_data.drop([14390, 16241, 16438, 5957], axis = 0, inplace = True)
games_data["publisher"].fillna(value = "Unknown", inplace = True)
games_data.dropna(inplace = True)
games_data.reset_index(inplace = True, drop = True)
```

```
trunc_data = games_data.loc[(games_data["global_sales"] > 0.01)]
```

```
# numeric summary after changes made
trunc_data.describe()
```

	year	rank	na_sales	eu_sales	jp_sales	other_sales	global_sales
count	15720.000000	15720.000000	15720.000000	15720.000000	15720.000000	15720.000000	15720.000000
mean	2006.298982	7984.755534	0.275555	0.153152	0.081497	0.050190	0.560686
std	5.852258	4614.625014	0.835647	0.517682	0.317173	0.193275	1.592144
min	1980.000000	1.000000	0.000000	0.000000	0.000000	0.000000	0.020000
25%	2003.000000	3984.750000	0.000000	0.000000	0.000000	0.000000	0.070000
50%	2007.000000	7986.500000	0.080000	0.030000	0.000000	0.010000	0.190000
75%	2010.000000	11977.500000	0.250000	0.120000	0.040000	0.040000	0.500000
max	2016.000000	15982.000000	41.490000	29.020000	10.220000	10.570000	82.740000

```
# hist global sales
fig = px.histogram(trunc_data, x = "global_sales", nbins=50 ,log_y=True, title = "Histogram of global sales")
fig.show()
```

```
# hist of publishers by global sales
top_publishers = trunc_data.groupby("publisher").sum().sort_values(by = "global_sales", ascending = False)[["na_sales", "eu_sales
actual_top_pub = top_publishers.loc[top_publishers["global_sales"] >= 100]
fig = px.bar(actual_top_pub, x = "publisher", y = "global_sales")
fig.show()
```

```
# same for top platforms
top_platforms = trunc_data.groupby("platform").sum().sort_values(by = "global_sales", ascending = False)[["na_sales", "eu_sales",
fig = px.bar(top_platforms.loc[top_platforms["global_sales"] > 50], x = "platform", y = "global_sales")
fig.show()
```

```
# same for top genre
top_genres = trunc_data.groupby("genre").sum().sort_values(by = "global_sales", ascending = False)[["na_sales", "eu_sales", "jp_s
fig = px.bar(top_genres, x = "genre", y = "global_sales")
fig.show()
```

```python
def aggregate_regional_summary(feature):
    """
    Gets regional sum and average, grouped by << feature >> (genre, publisher, year).

    Parameters:
    feature <str>: category of interest in dataframe

    Returns:
    agg_sales <dataframe>: dataframe of regional sum and average, grouped by feature
    """

    # get na_sales
    na_sales = trunc_data.groupby(feature)["na_sales"].agg(["sum", "mean"]).reset_index()
    na_sales.columns = [feature, "sum_na", "avg_na"]

    # get eu_sales
    eu_sales = trunc_data.groupby(feature)["eu_sales"].agg(["sum", "mean"]).reset_index()
    eu_sales.columns = [feature, "sum_eu", "avg_eu"]

    # get jp_sales
    jp_sales = trunc_data.groupby(feature)["jp_sales"].agg(["sum", "mean"]).reset_index()
    jp_sales.columns = [feature, "sum_jp", "avg_jp"]

    # get other_sales
    other_sales = trunc_data.groupby(feature)["other_sales"].agg(["sum", "mean"]).reset_index()
    other_sales.columns = [feature, "sum_other", "avg_other"]

    # get global_sales
    global_sales = trunc_data.groupby(feature)["global_sales"].agg(["sum", "mean"]).reset_index()
    global_sales.columns = [feature, "sum_global", "avg_global"]

    # merge all sales tables on the category of interest
    agg_sales = na_sales.merge(eu_sales, on = feature).merge(jp_sales, on = feature).merge(other_sales, on = feature).merge(globa

    return agg_sales
```

```python
# genre summary for each sales region
agg_genre_sales = aggregate_regional_summary("genre")
agg_genre_sales
```

```python
# create pie chart
fig = px.pie(agg_genre_sales, values = "sum_global", names = "genre", color_discrete_sequence = px.colors.sequential.thermal,
             title = "Total Sales Globally by Genre")
fig.update_traces(textposition = "inside", textinfo = "label+percent")
```

```python
# publisher summary by region
agg_pub_sales = aggregate_regional_summary("publisher")

# create pie chart
fig = px.pie(agg_pub_sales, values = "sum_global", names = "publisher", color_discrete_sequence = px.colors.sequential.thermal,
             title = "Total Sales Globally by Publisher")
fig.update_traces(textposition = "inside", textinfo = "label+percent")
```

```python
from scipy.stats import permutation_test, ttest_ind
```

```python
action_games = trunc_data.loc[trunc_data["genre"] == "Action"]["global_sales"]
sports_games = trunc_data.loc[trunc_data["genre"] == "Sports"]["global_sales"]
```

```python
action_games.describe()
```

```
count    3148.000000
mean        0.546960
std         1.180286
min         0.020000
25%         0.080000
50%         0.205000
75%         0.510000
max        21.400000
```

```python
# compute difference in means
def statistic(x, y, axis):
    return np.mean(x, axis=axis) - np.mean(y, axis=axis)
```

```python
# t - test
ttest_ind(action_games, sports_games, alternative = "less", equal_var = False)
```

```
Ttest_indResult(statistic=-0.6755367603317097, pvalue=0.24969146392880254)
```

```python
# resample
permutation_test((action_games, sports_games), statistic = statistic, vectorized = True, alternative = "less")
```

```
PermutationTestResult(statistic=-0.03341696554599827, pvalue=0.2478, null_distribution=array([-0.061822  , -0.02429874,  0.0132
702 , ..., -0.07310943,
       -0.04907322,  0.04949194]))
```

```
nint_plat = trunc_data.loc[(trunc_data["publisher"] == "Nintendo") & (trunc_data["genre"] == "Platform")]["global_sales"]
nint_rpg = trunc_data.loc[(trunc_data["publisher"] == "Nintendo") & (trunc_data["genre"] == "Role-Playing")]["global_sales"]
```

```
# t - test
ttest_ind(nint_plat, nint_rpg, alternative = "greater", equal_var = False)
```

```
Ttest_indResult(statistic=1.5053114703612713, pvalue=0.06688225533125643)
```

```
# permutation test
permutation_test((nint_plat, nint_rpg), statistic = statistic, vectorized = True, alternative = "greater")
```

```
PermutationTestResult(statistic=1.1640822510822524, pvalue=0.0655, null_distribution=array([-0.41798268,  0.86084848,  0.891190
48, ...,  0.04794372,
       -0.93658874, -3.19027273]))
```

```
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
```

```
yearly_sales = trunc_data.groupby(["year", "genre"]).sum().reset_index() # sales by year and genre
yearly_action_sales = yearly_sales.loc[yearly_sales["genre"] == "Action"].drop(columns = ["genre", "rank"]) # yearly action sales
yearly_action_sales.index = yearly_action_sales["year"] # shift year to be index of df
```

```
# scatter plot of global action game sales vs year
plt.scatter(x = yearly_action_sales["global_sales"].index, y = yearly_action_sales["global_sales"].values)
```

```
# linear regression model
lin_reg = LinearRegression()
years = np.array(yearly_action_sales["global_sales"].index - 1980).reshape(-1, 1)
sales = np.array(yearly_action_sales["global_sales"].values)

# fit model to data
lin_reg.fit(years, sales)
```

```
# plot line of best fit on top of scatter data
plt.plot(years, lin_reg.predict(years))
plt.scatter(years, sales)
```

```
# correlation coefficient and coefficient of determination
r_sq = lin_reg.score(years, sales)
r = np.sqrt(r_sq)
r, r_sq
```

```
(0.8178311186710656, 0.6688477386667666)
```

```
# slope and intercept
print(f"b0 = {lin_reg.intercept_}\nb1 = {lin_reg.coef_[0]}")
```

```
b0 = -18.12574679943097
b1 = 3.5923162636320507
```

```
# cubic fit function
def f(x):
    # get coefficients of x. order goes from highest degree to lowest (x^3 -> constant)
    coeffs = np.polyfit(years.reshape(-1,), sales, deg = 3)
    return coeffs[0] * x**3 + coeffs[1] * x**2 + coeffs[2] * x + coeffs[3]
```

```
X = np.linspace(0,36,100) # x values (years since 1980)
predicted_y = [f(x) for x in X] # predicted y values
plt.plot(X, predicted_y) # plot predicted curve
plt.scatter(years, sales) # scatter plot of actual data
```

```
# compute residuals of fit
residuals = [sales[k] - predicted_y[k] for k in range(len(predicted_y))]
residuals
```

```
# compute squared error (SE)
SE = 0
for r in residuals:
    SE += r**2
print(f"SE is {SE}")

# compute root mean squared error
RMSE = np.sqrt(SE / len(y_vals))
print(f"RMSE is {RMSE}")
```

```
SE is 12717.428903923963
RMSE is 18.53953329241725
```

```
# do same as above to compute residuals for linear fit
lin_fit = lin_reg.predict(years)
residuals_lin = [sales[k] - lin_fit[k] for k in range(len(y_vals))]
residuals_lin
```

```
# lin fit SE and RMSE
# squared error (SE)
SE_lin = 0
for r in residuals_lin:
    SE_lin += r**2
print(f"SE is {SE_lin}")

# root mean squared error
RMSE_lin = np.sqrt(SE_lin / len(y_vals))
print(f"RMSE is {RMSE_lin}")
```

```
SE is 26949.838461996216
RMSE is 26.988405148996563
```