Decoding Gamer Preferences:
Steam Game Recommendations

Section 27 Final Project Group 3:
Mark Tian(522346)
Seojung Park (473169)
Yifan Xie(521597)
Young Kim (520193)
Lucas Yang（509806）

Executive Summary

Steam is an online platform providing games for users to download and purchase. Our report, Decoding Gamer Preferences: Steam Game Recommendations, analyzes Steam games using the Kaggle dataset to identify trends and user behavior that influence game recommendations and game reviews (Anton, 2023). The main method of data analysis was through PySpark and Linux, and we visualized the results through Tableau. There are two main purposes of the report: to analyze gaming trends on Steam, including buzzwords in game titles, and to explore the factors that influence user reviews and recommendations. Key findings include an increase in game prices, a surge in Windows platform usage, and a correlation between the number of games owned and the likelihood of leaving a review. We had five major findings: popular words used for the title of the game are related to the concept of the game, there has been an increase in game prices since 2021, there is the dominance of the Windows platform games provided on Steam, there is a link between user engagement and review frequency, and lastly, user engagement drives recommendation from the users. Through these findings, we recommend Steam implement a ceiling price to control the prices of the games and to designate more budget to incentivize users to write more reviews. Lastly, we recommend individual game creators hold game events to increase user engagement.

Data Description

The data contains information on approximately 12.8M user reviews on more than 50,000 games. The information is divided into three data frames: users.csv containing information about the users, games.csv containing information about games provided on Steam, and recommendations.csv which contains more detailed information about each review, such as whether users recommend the game or not. The data information is specified as follows:

Dataset	Size	Total number of records	Total columns	Column names	Data types

games.csv	4.86MB	50,872	13	app_id, title, date_release, win, mac, linux, rating, positive_ratio, user_reviews, price_final, price_original, discount, steam_deck	Numerical, Booleans, Strings
users.csv	185.66MB	13,786,778	3	user_id, products, reviews	Numerical
recommendations.csv	1.88GB	38,354,101	8	app_id, helpful, funny, date, is_recommended, hours, user_id, review_id	Numerical, Boolean, Strings

Why This is Big Data

This dataset as an exemplar of big data because of for its volume, variety, value, and variety, the details are as follows:

- Volume - Our dataset is 648MB and explores more than 22 aspects of the games provided on Steam and the reviews on these games, which captures the massive amount of data within the Steam gaming community.
- Variety - The dataset contains a variety of data types covering a range of information such as title, platform compatibility, and pricing details, providing multifaceted insights into the gaming ecosystem.
- Value - This data has great potential to extract valuable insights, for example, it allows us to understand the factors that influence user engagement and game success, not just pricing strategies.
- Variety- The gaming industry is known for its dynamic nature, and this dataset still captures this rapid evolution and reflects well the fast pace of changes in gaming trends and preferences.

Problem Statement

Our report's analysis is divided into four main areas: word counts for game titles, time-series analysis of game trends, user review patterns, and factors that influence game recommendations. The main goal is to provide a comprehensive analysis of games on the Steam platform and identify the key variables that drive users to write reviews and recommend games. Initial analysis of the data includes how word count correlates with other aspects and examining game trends over time. We dive deeper into finding the main drivers behind reviews and recommendations. This includes

exploring and evaluating whether game engagement influences review frequency, and identifying the factors that are most influential on game recommendations.

Method and results

Word Count Analysis

Pyspark and Linux were used to conduct a comprehensive analysis of the most frequently utilized term for the game title. Initially, Pyspark was used to split the title into individual rows while concurrently preserving relevant data (Table 1). Linux was used to perform word count on the title row storing the result to title_wordcount.csv (Table 2). Pyspark was used again to clean the format of the word count, convert boolean into numerical values, and inner join games.csv, recommendation.csv, and title_wordcount.csv (Table 3). In the final step of the analysis, Pyspark was used to find correlations between the word count and other variables (Table 4). The result of the data frames that were created by these processes was: wordcount_clean.csv which contained the clean table of the word count, and corr_matrix.csv which contained information about the correlation between the word count and other variables.

For visualization, we imported two data frames into Tableau. As shown in Table 5, the word count seems to have almost no correlation with other variables. The highest absolute correlation value was 0.024 with app_id – the native product ID of the game on Steam – being the highest absolute value. Notably, word count demonstrates a lower correlation to is_recommended (0.002) and user_reviews (-0.004).

Tables 6 and 7 present visualizations of the wordcount_clean.csv dataset, depicting the unfiltered and filtered versions, respectively. Due to the size of the data and diversity of the language and the combination of the words with numbers or symbols, words that had less than 150 counts were filtered out. When initially inputted into Tableau, the most frequently used words for the title of the game included numbers, prepositions, articles, and symbols. We have filtered out these elements, as they did not yield significant insights into the game's genre or nature. The most frequently used words were “Pack”(2,270), “Simulator”(1,030), “Edition” (1,013), and “VR”(924). Additionally, certain genre-related words also had relatively high word count, such as space (428),

war (464), and battle (371). However, words relating to the concept of the game predominated the words relating to the genre of the game.

Time Series Analysis

Through data on prices, operating systems, and year, we came up with the average price per year and the proportions of each operating system in the total number of released games every year. Table 8 shows that from 2013 to 2018, the price has been gradually decreasing, however, in the recent three years, the average game price has increased by approximately 11 dollars. The reason for this requires data on outside factors, such as macroeconomic variables and trends in the United States. One of the reasoning that could be a possible reasoning behind the increase in the average game price might be been affected by inflation after the pandemic of COVID.

Additionally, we have also analyzed the proportion of different Operating Systems – Mac, Windows, and Linux – between 2012 and 2017 (Table 9). The number of released games per year has significantly increased from 902 in 2012 to 4,279 in 2023. (Table 10)The domination of Windows-supporting games is emphasized by the proportion of Operating Systems that the games support. In 2012, games released on Mac and Linux accounted for approximately 22% and 17%. The proportion of Mac and Linux supported game have continuously decreased since 2012. In 2023, Mac and Linux have decreased to 12% and 9%, respectively. On the other hand, since 2017, the market share of Windows has become more and more predominant.

User Review Analysis

Using Pyspark, we created a linear regression model in order to predict the number of reviews by the number of games that a user owns.

The linear regression model is as follows:

$$\text{Number of Reviews} = 0.24793 + (\text{slope} \times \text{Number of Games Owned})$$

In the linear regression model provided, an intercept of 0.24793 means that if a user owns zero games, the model would predict that they would write approximately 0.24793 reviews. The intercept is a statistical artifact of the linear regression equation and does not have a practical real-world interpretation since it is not possible to write a fraction of a review. Furthermore, it is unlikely for someone to write a review if they do not own

any games. In statistics, the intercept is simply where the regression line crosses the y-axis. This is useful for the regression equation to make predictions within the scope of the data it was derived from, but one should interpret it with caution, particularly in contexts where the independent variable's value of zero doesn't make practical sense. The linear regression model provides a statistical model with a positive but weak correlation between the number of games owned and the number of reviews written, implying that owning more games is generally associated with writing more reviews, but that the model does not have a strong predictive relationship. Few outliers are noticeable where a small subset of users own and review many games, significantly diverging from the average user behavior.

Table 11 depicts a PCA scatterplot. PCA is a dimensionality-reduction method that is used to reduce the number of variables in a data set while preserving as much information as possible. PCA scatterplot shows that the majority of users are not prolific reviewers, owning and reviewing only a small number of games. We divided the scatterplot into three main clusters represented by three colors: Dark Blue (Cluster 0), Green (Cluster 1), and Yellow (Cluster 2). Cluster 0 represents users with fewer games and reviews, and Cluster 1 represents users with a moderate number of games and reviews, and lastly, Cluster 2 represents users with a large number of games and reviews. The clustering suggests that there are three distinct groups of users based on their purchasing and reviewing habits. From the PCA scatterplot, it is noticeable that there is a trend of people who own more products write more reviews.

The graph depicted in Table 12 shows the wide dispersion of data points, indicating the variability of user behavior. Some users own a lot of games and write a lot of reviews, but the trend is not very clear.

Trend: The analysis indicates a slight positive trend, showing that users typically write more reviews as they own more games, but the increase in reviews is not substantial with each additional game owned.

User Recommendation Analysis

The Initial search for insight into factors that correlate with users' recommendation rates was done through Tableau,in table 14, using "if_recommendated" as an attribute

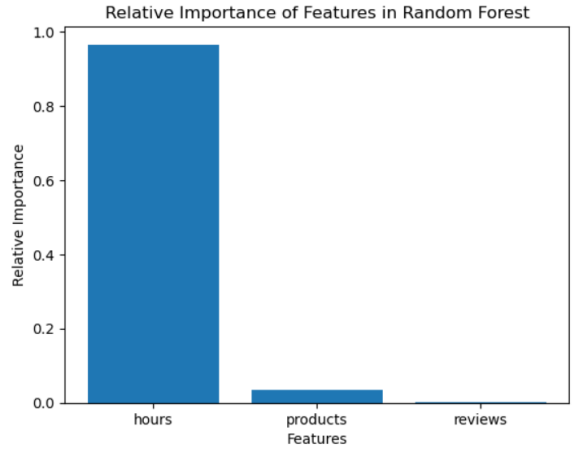
to do aggregate analysis. Three discernible patterns were found: products, hours, and reviews. The left histogram represents the group who does not recommend and the right-hand side histogram represents the group of users to recommend.

The data frames were uploaded to Pyspark, joined the user and recommendation data set in Pyspark, table 15, and used Pyspark to further analyze our problem. Because predicting "is_recommended" is a categorical problem and the data contains a low number of parameters and a high number of observations, a random forest model was used. Hours, products, and reviews were selected as features to predict, in table 16. The train data set is 70 percent of the data.

```
+----------+--------------------+------------------+
|prediction|is_recommended_index|          features|
+----------+--------------------+------------------+
|       0.0|                 0.0| [368.9,375.0,2.0]|
|       0.0|                 0.0| [83.5,358.0,20.0]|
|       0.0|                 0.0| [42.9,554.0,89.0]|
|       0.0|                 0.0|  [750.8,73.0,1.0]|
|       0.0|                 0.0| [503.1,122.0,1.0]|
|       0.0|                 0.0|[964.2,495.0,13.0]|
|       0.0|                 0.0|[117.0,227.0,29.0]|
|       0.0|                 0.0|  [988.9,38.0,1.0]|
|       0.0|                 0.0|  [720.0,37.0,1.0]|
|       0.0|                 0.0| [853.5,109.0,2.0]|
+----------+--------------------+------------------+
only showing top 10 rows
```

The test error shows the analysis is very accurate, with the test error of only 0.14. Finally, through relative importance analysis, we tried to find a factor that is the strongest predictor to affect users' decisions. The result shows hours the user played is the most important factor, which has 0.96 relative importance. To visualize the result,

we use matplotlib to plot to compare their relative importance.



Conclusion:

The graphs in this report illustrate the varying engagement levels of users within a gaming platform, with the majority being casual users and a minority being highly engaged. This information could be valuable for marketing strategies, customer engagement, and product development within the gaming industry. The linear regression model, while indicative of a trend, suggests that the number of games owned is not a strong predictor of the number of reviews written, highlighting the multifaceted nature of user engagement. From the time series analysis findings, we recommend Steam implement a ceiling price to control the prices of the games. User Review Analysis shows that in order to designate more budget to incentivize users to write more reviews. Lastly, User Recommendation analysis shows that it is recommended for individual game creators to hold game events to increase user engagement.

Appendices:

Table 1: Word Count Analysis - Splitting Title

```
import findspark
findspark.init()
```

```
Intel MKL WARNING: Support of Intel(R) Streaming SIMD Extensions 4.2 (Intel(R) SSE4.2) enabled only processors has
been deprecated. Intel oneAPI Math Kernel Library 2025.0 will require Intel(R) Advanced Vector Extensions (Intel(R)
AVX) instructions.
Intel MKL WARNING: Support of Intel(R) Streaming SIMD Extensions 4.2 (Intel(R) SSE4.2) enabled only processors has
been deprecated. Intel oneAPI Math Kernel Library 2025.0 will require Intel(R) Advanced Vector Extensions (Intel(R)
AVX) instructions.
```

```
import pyspark
from pyspark.sql import SparkSession
spark = (SparkSession.builder.master("local[*]").config("spark.driver.bindAddress", "127.0.0.1").appName('Final Proj
sc = spark.sparkContext

from pyspark.sql import functions as F
```

```
23/12/06 20:12:42 WARN Utils: Your hostname, MacBook-Pro-3.local resolves to a loopback address: 127.0.0.1; using 1
92.168.99.130 instead (on interface en0)
23/12/06 20:12:42 WARN Utils: Set SPARK_LOCAL_IP if you need to bind to another address
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
23/12/06 20:12:43 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-ja
va classes where applicable
```

```
df = spark.read.option('header','true').csv('games.csv', inferSchema = True)
title = df.withColumn('title', F.lower(df['title']))
title = title.withColumn('title', F.explode(F.split(title['title'], ' ')))
new_file_path = 'title.csv'
title.write.csv(new_file_path, header=True, mode='overwrite')
title.limit(5).toPandas()
```

	app_id	title	date_release	win	mac	linux	rating	positive_ratio	user_reviews	price_final	price_original	discount	steam_deck
0	13500	prince	2008-11-21	True	False	False	Very Positive	84	2199	9.99	9.99	0.0	True
1	13500	of	2008-11-21	True	False	False	Very Positive	84	2199	9.99	9.99	0.0	True
2	13500	persia:	2008-11-21	True	False	False	Very Positive	84	2199	9.99	9.99	0.0	True
3	13500	warrior	2008-11-21	True	False	False	Very Positive	84	2199	9.99	9.99	0.0	True
4	13500	within™	2008-11-21	True	False	False	Very Positive	84	2199	9.99	9.99	0.0	True

Table 2: Word Count Analysis - Linux Word Count

```
[seojung@ip-172-31-25-9 ~]$ cut -d"," -f2 title.csv | tail -n+2 | sort | uniq -c | sort -rn
   7512 -
   7377 the
   5294 of
   2270 pack
   1813 2
   1036 a
   1030 simulator
   1013 edition
    990 and
    924 vr
    765 soundtrack
    741 add-on
    741 &
    684 in
    650 2:
    636 game
    629 world
    619 train
    615 to
    535 3
    514 simulator:
    491 dlc
    464 war
    428 space
    395 for
    371 battle
    360 monster
    359 4
    344 hunter
    338 super
    333 dark
    328 ii
    323 adventure
    323 ~
    315 lost
    310 my
    298 dead
```

```
[seojung@ip-172-31-25-9 ~]$ cut -d"," -f2 title.csv | tail -n+2 | sort | uniq -c | sort -rn >
title_wordcount.csv
[seojung@ip-172-31-25-9 ~]$ ls
accidents.csv       fda_c_dead.csv       ls          reducer.py       title_unparsed.csv
CarSale.csv         hadoop-examples.jar  mapper.py   runmr.sh         title_wordcount.csv
DAT560M-Session27   lls                  Mybook.txt  shakespeare.txt  title_word.csv
disease.csv         lpwd                 put         title.csv
```

Table 3: Word Count Analysis - Pyspark Join Tables

```
data2 = spark.read.option('header', 'true').csv('title_wordcount.csv', inferSchema=True)
data2 = data2.withColumn('count', F.substring('weight', 1, 7))
data2 = data2.withColumn('title', F.substring('weight', 8, 100))
wordcount_clean = data2.select('count', 'title')
new_file_path = 'wordcount_clean.csv'
wordcount_clean.write.csv(new_file_path, header=True, mode='overwrite')
wordcount_clean.limit(5).toPandas()
```

	count	title
0	7512	-
1	7377	the
2	5294	of
3	2270	pack
4	1813	2

```
df3 = spark.read.option('header', 'true').csv('recommendations.csv', inferSchema=True)
df3 = df3.withColumn('is_recommended', F.when(df3['is_recommended'] == True, 1).otherwise(0))
df3 = df3.drop('date')
df3 = df3.drop('user_id')
df3 = df3.drop('review_id')
grouped_df3 = df3.groupBy('app_id').agg(
    F.avg('helpful').alias('avg_helpful'),
    F.avg('funny').alias('avg_funny'),
    F.avg('is_recommended').alias('avg_is_recommended'),
    F.avg('hours').alias('avg_hours'),
)
grouped_df3.limit(5).toPandas()
```

	app_id	avg_helpful	avg_funny	avg_is_recommended	avg_hours
0	1465360	1.957630	0.650643	0.885756	128.946229
1	1222670	3.929310	1.064085	0.871551	130.759899
2	1811260	2.897532	0.895798	0.511122	177.506055
3	4000	3.904900	2.023350	0.958332	303.907862
4	261550	2.944164	1.069474	0.874863	199.539138

```
df1 = spark.read.option('header', 'true').csv('title.csv', inferSchema=True)
df2 = spark.read.option('header', 'true').csv('wordcount_clean.csv', inferSchema=True)
wordcount_compiled = df1.join(df2, on='title', how='inner')
wordcount_compiled = wordcount_compiled.join(grouped_df3, on='app_id', how='inner')
new_file_path = 'wordcount_compiled.csv'
wordcount_compiled.write.csv(new_file_path, header=True, mode='overwrite')
wordcount_compiled.limit(5).toPandas()
```

	app_id	title	date_release	win	mac	linux	rating	positive_ratio	user_reviews	price_final	price_original	discount	steam_deck	count
0	1269310	masterplan	2021-01-26	True	True	True	Very Positive	90	64	19.99	19.99	0.0	True	3
1	1335560	unreal	2020-11-19	True	True	False	Overwhelmingly Positive	96	752	19.99	19.99	0.0	True	3
2	1335560	life	2020-11-19	True	True	False	Overwhelmingly Positive	96	752	19.99	19.99	0.0	True	265
3	32460	monkey	2010-07-07	True	False	False	Overwhelmingly Positive	95	2068	3.49	9.99	65.0	True	30
4	32460	island™	2010-07-07	True	False	False	Overwhelmingly Positive	95	2068	3.49	9.99	65.0	True	2

Table 4: Word Count Analysis - Pyspark Correlation

```python
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.stat import Correlation
import pandas as pd

boolean_columns = ['win', 'mac', 'linux', 'steam_deck']
for col in boolean_columns:
    df = df.withColumn(col, F.when(df[col] == True, 1).otherwise(0))

measure_cols = [col_name for col_name, col_type in wordcount_compiled.dtypes if col_type not in ('string', 'date')]
numeric_df = wordcount_compiled.select(measure_cols)

vector_col = 'corr_features'
assembler = VectorAssembler(inputCols=numeric_df.columns, outputCol=vector_col)
df_vector = assembler.transform(numeric_df).select(vector_col)

corr_matrix = Correlation.corr(df_vector, vector_col).collect()[0][0]

variables = numeric_df.columns

corr_df = pd.DataFrame(corr_matrix.toArray(), index=variables, columns=variables)

corr_long = pd.melt(corr_df.reset_index(), id_vars='index', var_name='Variable2', value_name='Correlation')
corr_long = corr_long.rename(columns={'index': 'Variable1'})
corr_long = corr_long.sort_values(by=['Variable1', 'Variable2']).reset_index(drop=True)

new_file_path = 'corr_matrix.csv'
corr_long.to_csv(new_file_path, index=False)

corr_long.head(8)
```

	Variable1	Variable2	Correlation
0	app_id	app_id	1.000000
1	app_id	avg_funny	-0.027199
2	app_id	avg_helpful	-0.049574
3	app_id	avg_hours	-0.123533
4	app_id	avg_is_recommended	0.169611
5	app_id	count	-0.023606
6	app_id	discount	0.030000
7	app_id	linux	-0.132542

Table 5: Correlation Matrix

Correlation Matrix

	Variable2														
Variable1	app_id	avg_fun..	avg_hel..	avg_hou..	avg_is_r..	count	discount	linux	mac	positive..	price_fin..	price_ori..	steam_d..	user_rev..	win
app_id	1.000	-0.027	-0.050	-0.124	0.170	-0.024	0.030	-0.133	-0.161	0.173	-0.021	-0.002	0.010	-0.040	0.021
avg_funny	-0.027	1.000	0.666	0.019	-0.047	-0.006	-0.006	-0.013	-0.021	-0.059	0.055	0.053	-0.002	0.015	-0.012
avg_helpful	-0.050	0.666	1.000	0.010	-0.137	0.013	-0.004	-0.006	-0.011	-0.161	0.124	0.129	0.001	0.003	-0.007
avg_hours	-0.124	0.019	0.010	1.000	0.106	-0.011	-0.049	0.002	0.029	0.093	0.311	0.228	-0.002	0.215	0.015
avg_is_rec..	0.170	-0.047	-0.137	0.106	1.000	0.002	-0.030	0.082	0.091	0.855	0.052	0.035	-0.002	0.021	-0.025
count	-0.024	-0.006	0.013	-0.011	0.002	1.000	0.001	0.006	0.016	-0.003	0.006	0.007	0.002	-0.004	-0.008
discount	0.030	-0.006	-0.004	-0.049	-0.030	0.001	1.000	-0.021	-0.039	-0.006	-0.153	0.054	0.002	-0.010	0.012
linux	-0.133	-0.013	-0.006	0.002	0.082	0.006	-0.021	1.000	0.602	0.084	-0.001	-0.005	0.003	0.022	0.013
mac	-0.161	-0.021	-0.011	0.029	0.091	0.016	-0.039	0.602	1.000	0.094	-0.003	-0.012	0.004	0.019	0.022
positive_ra..	0.173	-0.059	-0.161	0.093	0.855	-0.003	-0.006	0.084	0.094	1.000	0.028	0.013	-0.002	0.027	-0.023
price_final	-0.021	0.055	0.124	0.311	0.052	0.006	-0.153	-0.001	-0.003	0.028	1.000	0.896	0.000	0.036	0.024
price_origi..	-0.002	0.053	0.129	0.228	0.035	0.007	0.054	-0.005	-0.012	0.013	0.896	1.000	0.000	-0.015	0.024
steam_deck	0.010	-0.002	0.001	-0.002	-0.002	0.002	0.002	0.003	0.004	-0.002	0.000	0.000	1.000	0.000	0.000
user_revie..	-0.040	0.015	0.003	0.215	0.021	-0.004	-0.010	0.022	0.019	0.027	0.036	-0.015	0.000	1.000	0.001
win	0.021	-0.012	-0.007	0.015	-0.025	-0.008	0.012	0.013	0.022	-0.023	0.024	0.024	0.000	0.001	1.000

Table 6: Unfiltered Word Count

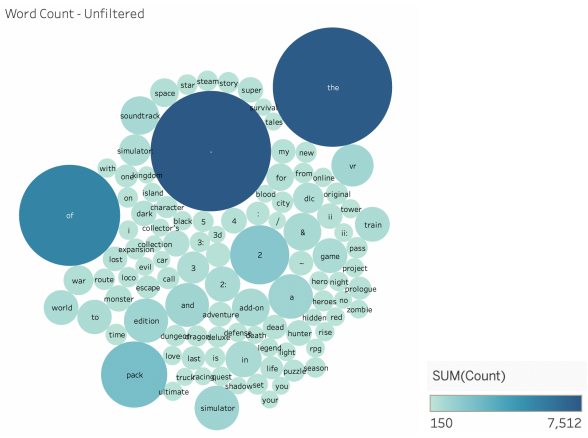


Table 7: Filtered Word Count

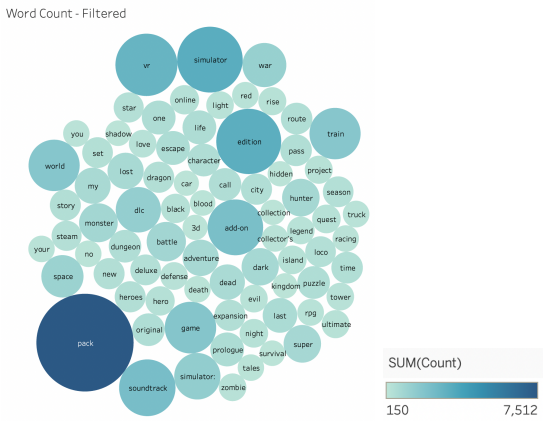


Table 8: Time Series Analysis - Pyspark

C. Converting boolean datatypes to numeric ones.

```
from pyspark.sql.functions import col

df = df.withColumn('win', col('win').cast('integer')) \
       .withColumn('mac', col('mac').cast('integer')) \
       .withColumn('linux', col('linux').cast('integer'))
```

A. Average Cost of a Steam Game Per Year

```
df = df.withColumn('Year', year('date_release'))
df_2013 = df.filter(df['Year'] >= 2012)
mean_original_prices = df_2013.groupBy('Year').avg('price_final').withColumnRenamed('avg(price_final)', 'average_price')

# For visualization
pandas_df = mean_original_prices.toPandas()
pandas_df.to_csv('visualizing_df.csv', index=False)
```

B. Steam Games Released Per Year

```
platform_counts_per_year = df.groupBy(year('date_release').alias('year'))\
                           .sum('win', 'mac', 'linux')\
                           .withColumnRenamed('sum(win)', 'Windows Games')\
                           .withColumnRenamed('sum(mac)', 'Mac Games')\
                           .withColumnRenamed('sum(linux)', 'Linux Games')

# For visualization
pandas_df2 = platform_counts_per_year.toPandas()
pandas_df2.to_csv('visual_df2.csv', index=False)
```

Table 9: Average price of Games

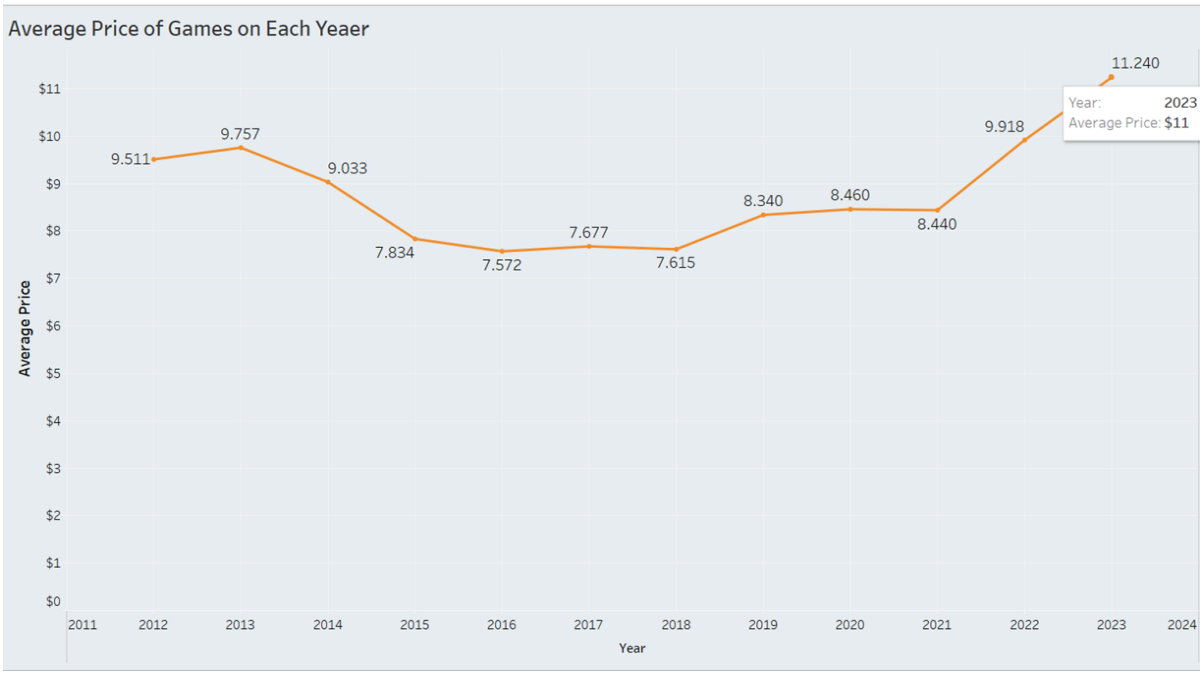


Table 10: Time Series Analysis

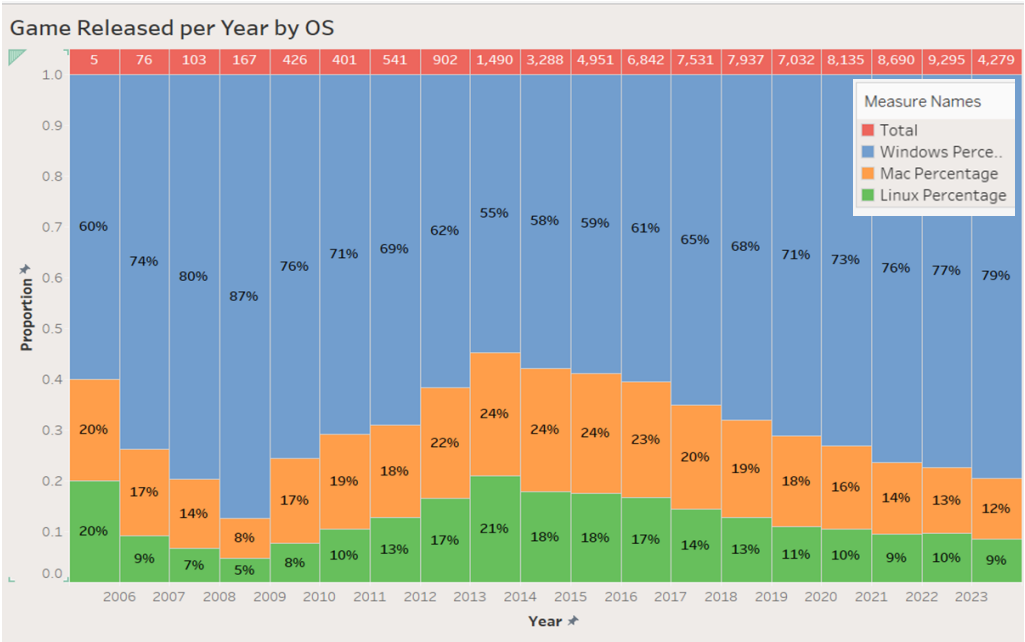


Table 11: PCA Analysis

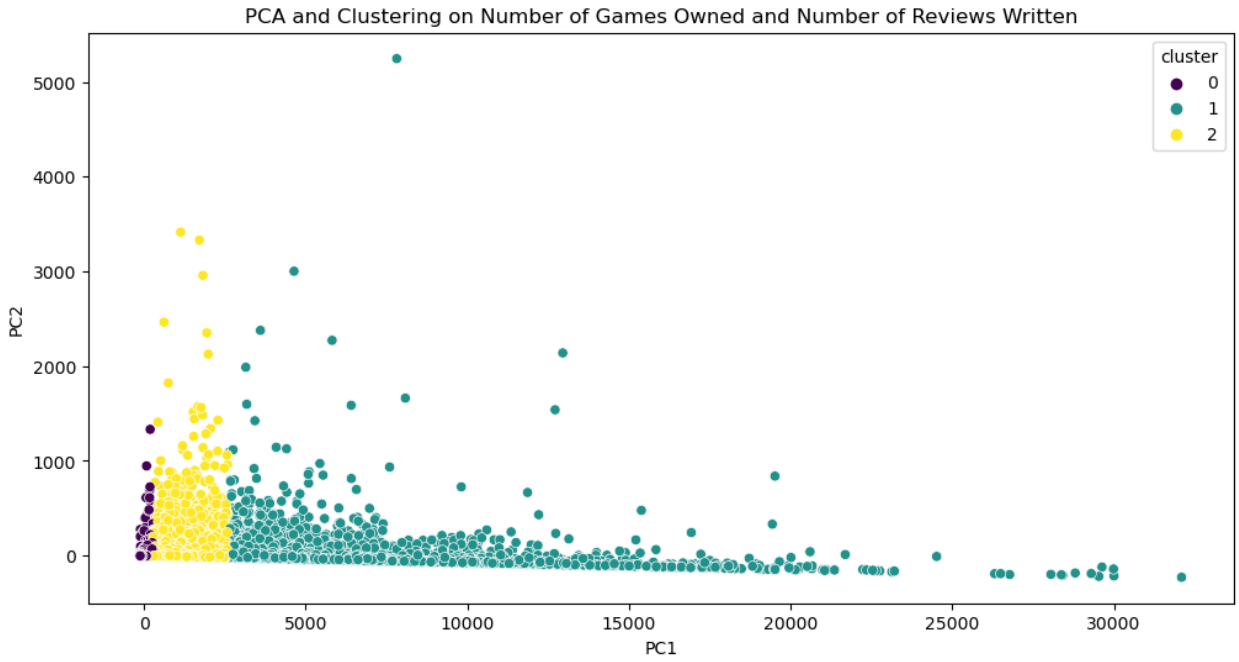


Table 12: User Review Analysis - Games Owned vs. Reviews Written

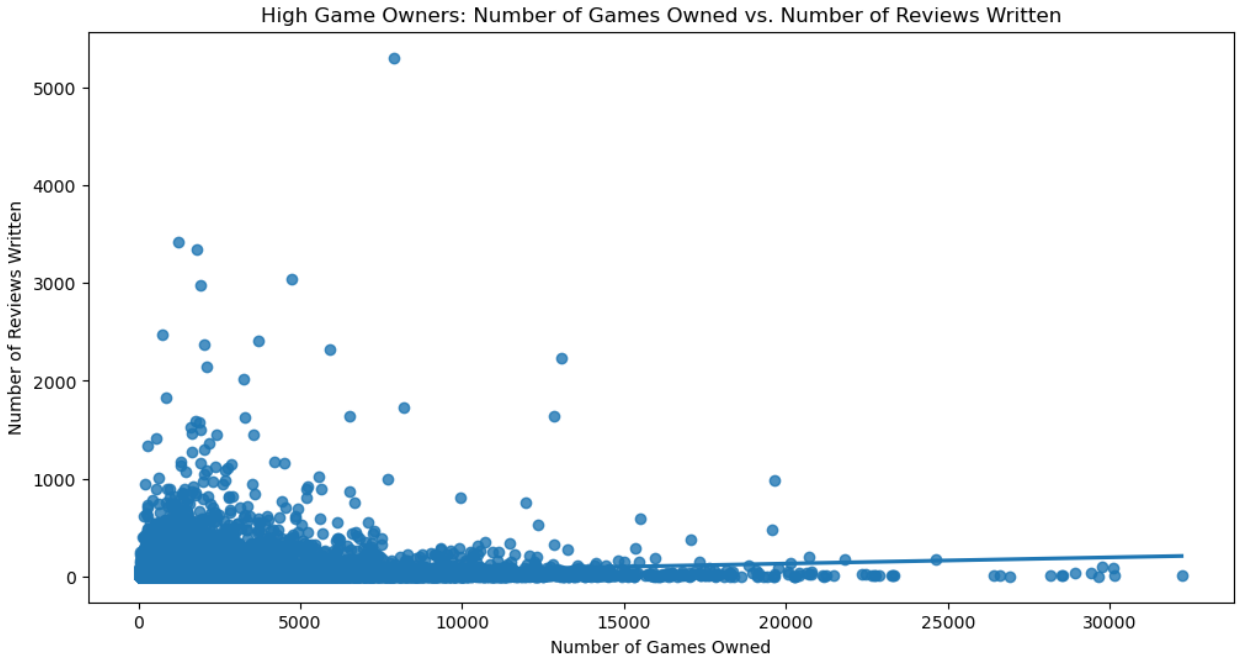


Table 13: PCA Analysis- Pyspark

```
pca = PCA(n_components=2)
principal_components = pca.fit_transform(data[['products', 'reviews']])
principal_df = pd.DataFrame(data=principal_components, columns=['PC1', 'PC2'])

kmeans = KMeans(n_clusters=3, n_init=n_init_value)
principal_df['cluster'] = kmeans.fit_predict(principal_df[['PC1', 'PC2']])

plt.figure(figsize=(12, 6))
sns.scatterplot(x='PC1', y='PC2', hue='cluster', data=principal_df, palette='viridis')
plt.title('PCA and Clustering on Number of Games Owned and Number of Reviews Written')
plt.show()
```

Table 14:Recommendation Analysis -Tableau

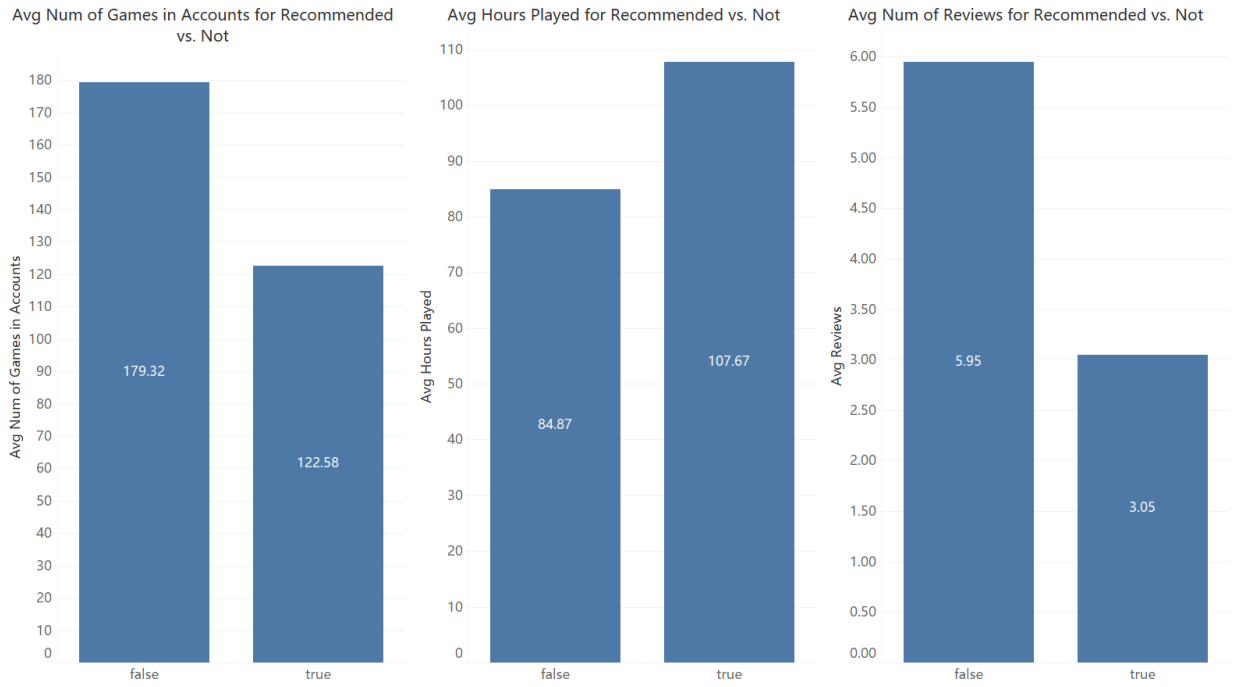


Table 15: User Recommendations Analysis - Pyspark

In [4]:
```
import findspark
import pyspark
from pyspark.sql import SparkSession
from pyspark.sql.functions import *
spark=SparkSession.builder.master("local").appName('Classification').getOrCreate()
sc=spark.sparkContext
```

In [5]:
```
df1 = spark.read.option('header', 'true').csv('data/recommendations.csv', inferSchema=True)
df2 = spark.read.option('header', 'true').csv('data/users.csv', inferSchema=True)
```

In [6]:
```
df2_alias = df2.select(col("user_id").alias("user_id_2"),col('products'),col('reviews'))
```

In [7]:
```
query = df1.join(df2_alias,df1.user_id == df2_alias.user_id_2, 'inner')
query = query.drop('user_id_2')
query.show(5)
```

```
+------+-------+-----+----------+--------------+-----+-------+---------+--------+-------+
|app_id|helpful|funny|      date|is_recommended|hours|user_id|review_id|products|reviews|
+------+-------+-----+----------+--------------+-----+-------+---------+--------+-------+
|105600|      6|    0|2014-06-21|          true|355.9|     31|  1292399|      63|      1|
|275850|      0|    0|2016-08-12|          true| 23.6|     53|   229463|     366|      9|
|107410|      0|    0|2014-04-25|          true| 22.9|     53|   600508|     366|      9|
|582660|      0|    0|2017-05-30|          true| 65.9|     53|  5683948|     366|      9|
|460920|      0|    0|2018-11-22|          true| 16.9|     53|  9158048|     366|      9|
+------+-------+-----+----------+--------------+-----+-------+---------+--------+-------+
only showing top 5 rows
```

In [15]:
```
from pyspark.ml.feature import StringIndexer, VectorAssembler
query = query.withColumn("is_recommended", col("is_recommended").cast("string"))
indexer=StringIndexer(inputCol='is_recommended',outputCol='is_recommended_index')
query = indexer.fit(query).transform(query)
query = query.drop('is_recommended')
```

Table 16: User Recommendations Analysis - Pyspark

```
In [22]: input_columns = ['hours', 'products', 'reviews']
feature = VectorAssembler(inputCols=input_columns,outputCol="features")
feature_vector= feature.transform(query)
(trainingData, testData) = feature_vector.randomSplit([0.7, 0.3])
```

```
In [23]: from pyspark.ml.classification import RandomForestClassifier
rf = RandomForestClassifier(labelCol="is_recommended_index", featuresCol="features")
rf_model = rf.fit(trainingData)
rf_prediction = rf_model.transform(testData)
rf_prediction.select("prediction", "is_recommended_index", "features").show(10)
```

```
+----------+--------------------+------------------+
|prediction|is_recommended_index|          features|
+----------+--------------------+------------------+
|       0.0|                 0.0| [368.9,375.0,2.0]|
|       0.0|                 0.0| [83.5,358.0,20.0]|
|       0.0|                 0.0| [42.9,554.0,89.0]|
|       0.0|                 0.0|  [750.8,73.0,1.0]|
|       0.0|                 0.0| [503.1,122.0,1.0]|
|       0.0|                 0.0|[964.2,495.0,13.0]|
|       0.0|                 0.0|[117.0,227.0,29.0]|
|       0.0|                 0.0|  [988.9,38.0,1.0]|
|       0.0|                 0.0|  [720.0,37.0,1.0]|
|       0.0|                 0.0| [853.5,109.0,2.0]|
+----------+--------------------+------------------+
only showing top 10 rows
```

Table 17: User Recommendations Analysis - Pyspark

```
from pyspark.ml.evaluation import MulticlassClassificationEvaluator


evaluator = MulticlassClassificationEvaluator(labelCol="is_recommended_index", predictionCol="prediction", metricName="accuracy")


rf_accuracy = evaluator.evaluate(rf_prediction)
print("Accuracy of RandomForestClassifier is = %g" % rf_accuracy)
print("Test Error of RandomForestClassifier  = %g" % (1.0 - rf_accuracy))
```

```
Accuracy of RandomForestClassifier is = 0.860446
Test Error of RandomForestClassifier  = 0.139554
```

```
importances = rf_model.featureImportances.toArray()
print("Feature Importances:")
for i, importance in enumerate(importances):
    print("Feature {}: {:.4f}".format(i + 1, importance))
```

```
Feature Importances:
Feature 1: 0.9649
Feature 2: 0.0337
Feature 3: 0.0014
```

```
import matplotlib.pyplot as plt

features = ["hours", "products",'reviews']

plt.bar(features, importances)
plt.xlabel("Features")
plt.ylabel("Relative Importance")
plt.title("Relative Importance of Features in Random Forest")
plt.show()
```

Works Cited:

Anton Kozyriev. (2023). Game Recommendations on Steam [Data set]. Kaggle. https://doi.org/10.34740/KAGGLE/DS/2871694