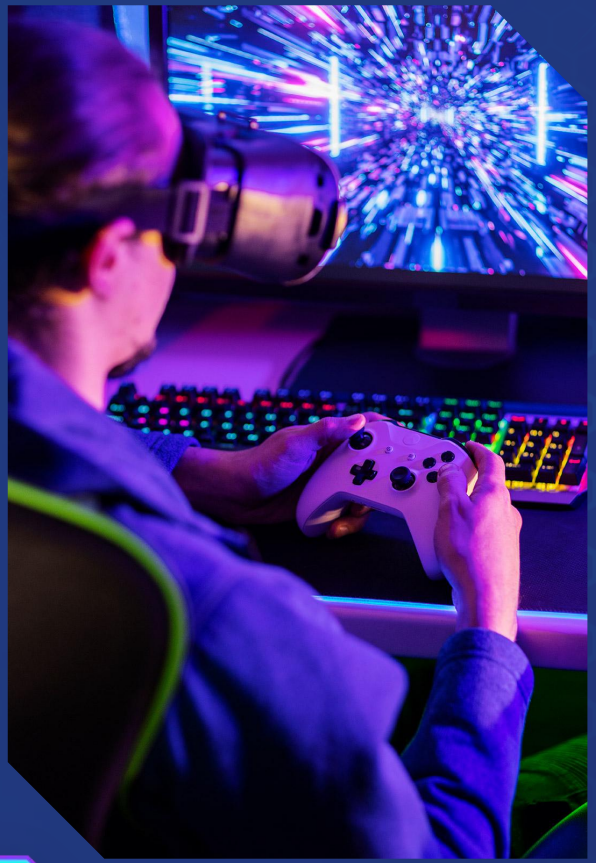
Steam Game Prediction

Lab Group: FCSE Group 6
Ngo Zong Han (U2321758A)
Rochelle Tan Tyen Xyn (U2321456F)

Table of contents



01 Motivation

1. Motivation

Skull & Bones

- Singapore's 1st major game title
- US$200 million and 10 years to develop
- Priced at $80

Motivation

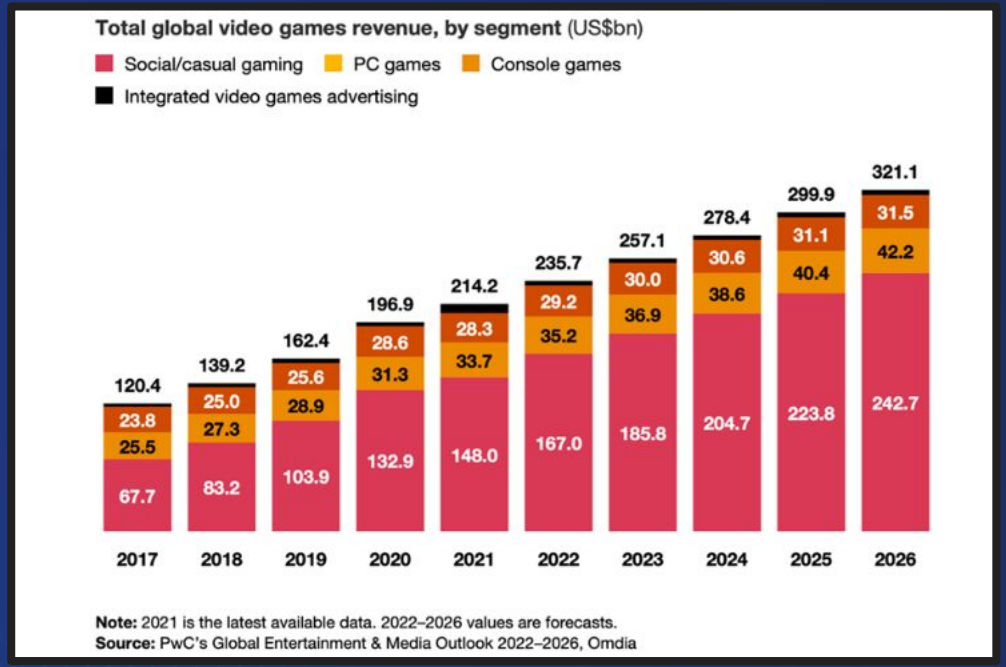
Large Funds + **Long Development Time** + **Expensive**

How profitable will the game be?

How successful will the game be?

Motivation

- Saturated with games
- Annual revenue US$78 billion to US$137 billion
- More than Hollywood and American music industry



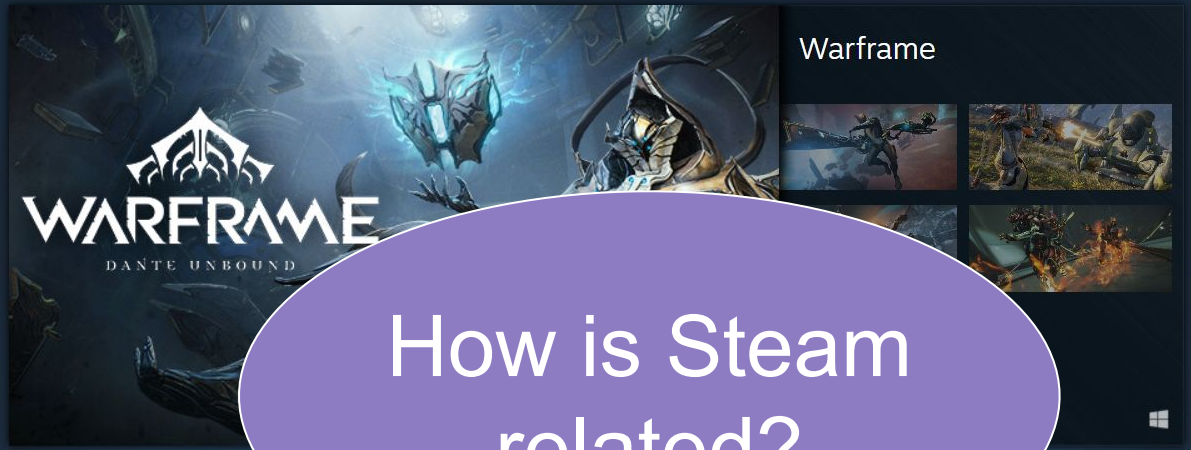
FEATURED & RECOMMENDED

STEAM GIFT CARDS
Give the Gift of Game

RECOMMENDED
By Friends
By Curators
Tags

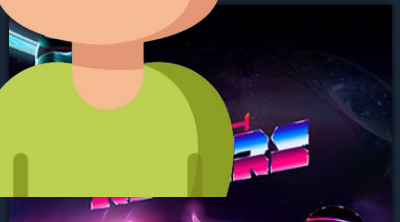
BROWSE CATEGORIES
Top Sellers
New Releases
Upcoming
Specials
VR Titles
Controller-Friendly
Great on Deck

BROWSE BY GENRE
Free to Play
Early Access
Action
Adventure
Casual
Indie
Massively Multiplayer
Racing
RPG
Simulation
Sports
Strategy



How is Steam related?

BROWSE MORE



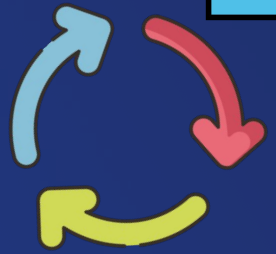
Today's deal! -25% S$22.00 S$16.50

Motivation

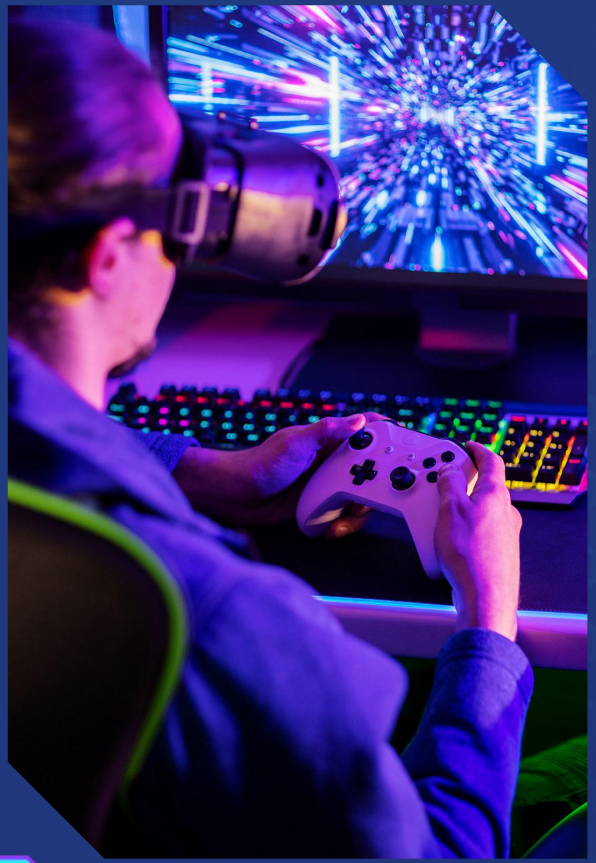
2015: 8.4 million
2023: 33 million

Trend in game increases

People and developers trying to capitalize



Long time and large funds required

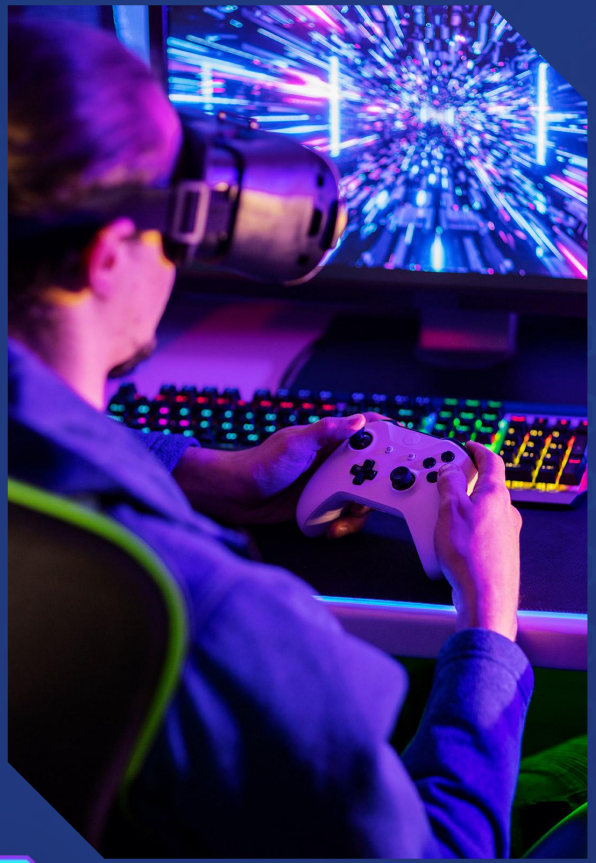


02

Aim

Project Aim:

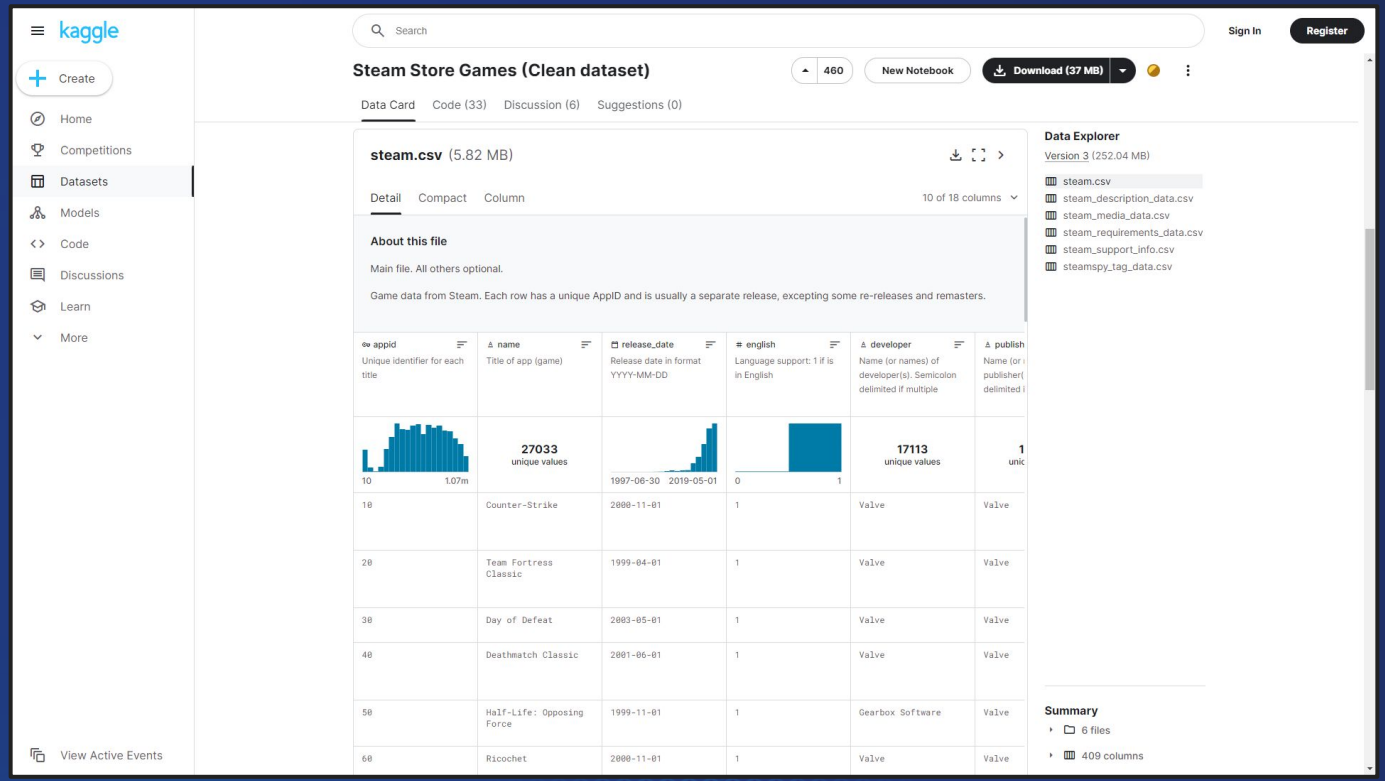
Maximise developers profits producing Steam Games by predicting the 'success' probability of game



03

Data Collection

Steam Dataset on Kaggle



3. Steam Spy Collection

JSON requests + web API

```python
response_details = requests.get("https://steamspy.com/api.php?request=appdetails&appid=10")

# Check if the request was successful (status code 200)
if response_details.status_code == 200:
    print("Request was successful.")
    print("Fields under 'data':")
    print("-----")

    # Access the 'data' dictionary within the JSON response
    data = response_details.json()

    # Iterate over the keys of the 'data' dictionary and print them
    for key in data.keys():
        print(key)
else:
    print("Error:", response_details.status_code)
```

SteamSpy Features

```
Request was successful.
Fields under 'data':
-----
appid
name
developer
publisher
score_rank
positive
negative
userscore
owners
average_forever
average_2weeks
median_forever
median_2weeks
price
initialprice
discount
ccu
languages
genre
tags
```

3. Steam Web Collection

```
type
name
steam_appid
required_age
is_free
detailed_description
about_the_game
short_description
supported_languages
header_image
capsule_image
capsule_imagev5
website
pc_requirements
mac_requirements
linux_requirements
developers
publishers
price_overview
packages
package_groups
platforms
metacritic
categories
genres
screenshots
recommendations
release_date
support_info
background
background_raw
content_descriptors
ratings
```

Steam Store Features

Eliminating features we deem unnecessary

```
type
name
steam_appid
required_age
is_free
detailed_description
about_the_game
short_description
supported_languages
header_image
capsule_image
capsule_imagev5
website
pc_requirements
mac_requirements
linux_requirements
developers
publishers
price_overview
packages
package_groups
platforms
metacritic
categories
genres
screenshots
recommendations
release_date
support_info
background
background_raw
content_descriptors
ratings
```

3. AppID Collection

```python
# Use steamspy API to retrieve SteamApp ID and Name
url = "https://steamspy.com/api.php"

#Loop through the 71 pages of SteamData using the page parameter
for i in range(0, 72, 1):
    pageNum = i;
    parameters = {"request": "all", "page": pageNum}

    # request 'all&page=i' using the parameters from steamspy and parse into dataframe
    json_data = get_request(url, parameters=parameters)
    steam_spy_all = pd.DataFrame.from_dict(json_data, orient='index')
    time.sleep(1) #pause

    # generate sorted app_list from steamspy data
    app_list = steam_spy_all[['appid', 'name']].sort_values('appid').reset_index(drop=True)

    # export to csv files based on their page number, formatted-strings
    app_list.to_csv(f"AppList/app_list_{i}.csv", index=False)
```

```python
# folder name
path = r'AppList'

# combine into a group
all_files = glob.glob(path + "/*.csv")
# empty array
li = []
# loop through all files in group and append
for filename in all_files:
    df = pd.read_csv(filename, index_col=None, header=0)
    li.append(df)
```

Excel Output

appid	name
10	Counter-Strike
20	Team Fortress Classic
30	Day of Defeat
40	Deathmatch Classic
50	Half-Life: Opposing Force
60	Ricochet
70	Half-Life
80	Counter-Strike: Condition Zero
130	Half-Life: Blue Shift
220	Half-Life 2
240	Counter-Strike: Source
280	Half-Life: Source
300	Day of Defeat: Source
320	Half-Life 2: Deathmatch
340	Half-Life 2: Lost Coast
360	Half-Life Deathmatch: Source
380	Half-Life 2: Episode One
400	Portal
420	Half-Life 2: Episode Two
440	Team Fortress 2
500	Left 4 Dead
550	Left 4 Dead 2
570	Dota 2

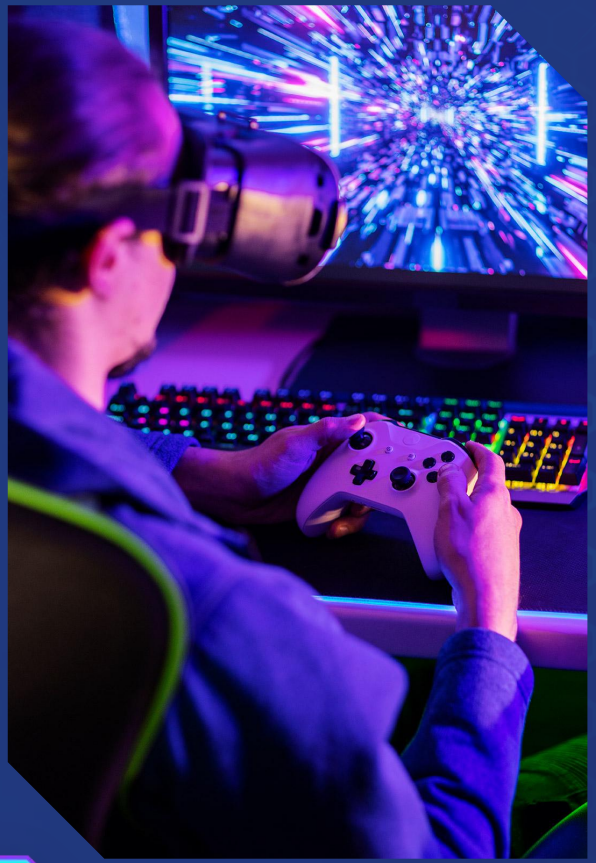
3. Function to get Data

```python
def parse_steamspy_request(appid, name):
    """Parser to handle SteamSpy API data."""
    url = "https://steamspy.com/api.php"
    parameters = {"request": "appdetails", "appid": appid}

    json_data = get_request(url, parameters)
    return json_data


# set files and columns
download_path = 'Test'
steamspy_data = 'steamspy_data.csv'
steamspy_index = 'steamspy_index.txt'

steamspy_columns = [
    'appid', 'name', 'developer', 'publisher', 'score_rank', 'positive',
    'negative', 'userscore', 'owners', 'average_forever', 'average_2weeks',
    'median_forever', 'median_2weeks', 'price', 'initialprice', 'discount',
    'languages', 'genre', 'ccu', 'tags'
]
```

```python
index = get_index(download_path, steamspy_index)

# Wipe data file if index is 0
prepare_data_file(download_path, steamspy_data, index, steamspy_columns)

process_batches(
    parser=parse_steamspy_request,
    mainAppList=mainAppList,
    download_path=download_path,
    data_filename=steamspy_data,
    index_filename=steamspy_index,
    columns=steamspy_columns,
    begin=index,
    #change end to indicate stopping index
    #end=40,
    batchsize=5,
    pause=0.3
)
```

Excel Output

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
1	appid	name	developer	publisher	score_ran	positive	negative	userscore	owners	average_f	average_2	median_f	median_2	price	initialpric	discount	languages	genre	ccu	tags
2	10	Counter-S	Valve	Valve		229766	5992	0	10,000,000	9847	205	169	139	999	999	0	English, Fr	Action	12919	{'Action': 5468,



04

Data Cleaning

4. Data Cleaning Pipeline

1. Identifying Null Values & Missing Values
2. Removing irrelevant and duplicated information
3. Pre - Processing & combing dataset

4.1 Handling Missing Data

Steam App

1. controller_support
2. dlc
3. full_game
4. legal & drm & user notice
5. demos
6. metacritic & reviews
7. recommendations

Steam Spy

1. score rank
2. genre
3. publisher
4. developer

4.2 Dropping Columns

Steam App

1. controller_support
2. dlc &full_game
4. legal & drm & user notice
5. demos
6. metacritic & reviews
7. recommendations
8. required_age, ratings
9. support info
10. pc & mac & linux requirements

Steam Spy

1. score rank
2. genre
3. publisher & developer
4. genre
5. average & median 2 weeks
6. ccu
7. price initial & price
8. score rank & userscore

4.3 Price Cleaning

	name	currency	price
0	Counter-Strike	SGD	1000
1	Team Fortress Classic	SGD	525
2	Day of Defeat	SGD	525
3	Deathmatch Classic	SGD	525
4	Half-Life: Opposing Force	SGD	525
...	...	...	...
71188	Windowkill	SGD	525
71189	SHIJIE XIUXIAN	SGD	215
71190	Knowledge, or know Lady	SGD	900
71191	Area 19	SGD	1200
71192	Darkness Ritual: Impasse	SGD	850

64561 rows × 3 columns

	name	currency	price
6	Half-Life	EUR	819
517	Prince of Persia®	EUR	999
906	FINAL FANTASY XIV Online	USD	1999
1479	Tomb Raider	USD	1499
1517	Dishonored	EUR	999
...	...	...	...
67931	Beautiful Mystic Survivors	EUR	399
68072	Fleet Commander: Pacific	USD	2499
68294	Stellar Orphans	USD	1699
68394	No Son Of Mine	EUR	1249
68881	StreamWare	USD	599

225 rows × 3 columns

	name	price
0	Counter-Strike	10.00
1	Team Fortress Classic	5.25
2	Day of Defeat	5.25
3	Deathmatch Classic	5.25
4	Half-Life: Opposing Force	5.25

4.3 Platform Cleaning

```
0    {'windows': True, 'mac': True, 'linux': True}
1    {'windows': True, 'mac': True, 'linux': True}
2    {'windows': True, 'mac': True, 'linux': True}
3    {'windows': True, 'mac': True, 'linux': True}
4    {'windows': True, 'mac': True, 'linux': True}
Name: platforms, dtype: object
```

```
platforms
windows                 48278
windows;mac;linux        7557
windows;mac              6859
windows;linux            2068
linux                      12
mac                        11
mac;linux                   1
Name: count, dtype: int64
```

4.3 Language Cleaning

```
supported_languages
English
English<strong>*</strong><br><strong>*</strong>languages with full a
English, Russian
English, Simplified Chinese
English, Japanese
Simplified Chinese
Simplified Chinese<strong>*</strong><br><strong>*</strong>languages
English<strong>*</strong>, Russian<strong>*</strong><br><strong>*</s
English, Portuguese - Brazil
English, French
Name: count, dtype: int64
```

```
english
1    58946
0     1993
Name: count, dtype: int64
```

4.3 Categories & Genres Cleaning

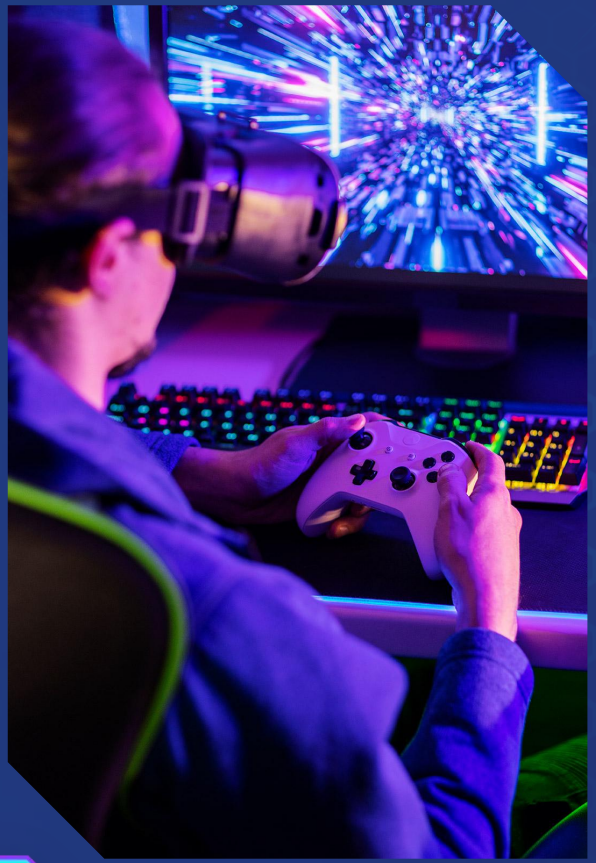
```
0    {'Action': 5468, 'FPS': 4894, 'Multiplayer': 3...
1    {'Action': 762, 'FPS': 326, 'Multiplayer': 276...
2    {'FPS': 797, 'World War II': 268, 'Multiplayer...
3    {'Action': 634, 'FPS': 149, 'Classic': 114, 'M...
4    {'FPS': 915, 'Action': 348, 'Classic': 279, 'S...
Name: tags, dtype: object
```

```
0                Action;FPS;Multiplayer
1                Action;FPS;Multiplayer
2    FPS;World War II;Multiplayer
3                   Action;FPS;Classic
4                   FPS;Action;Classic
Name: tags, dtype: object
```

4.3 Owners Cleaning

```
0    10,000,000 .. 20,000,000
1     5,000,000 .. 10,000,000
2     5,000,000 .. 10,000,000
3     5,000,000 .. 10,000,000
4      2,000,000 .. 5,000,000
Name: owners, dtype: object
```

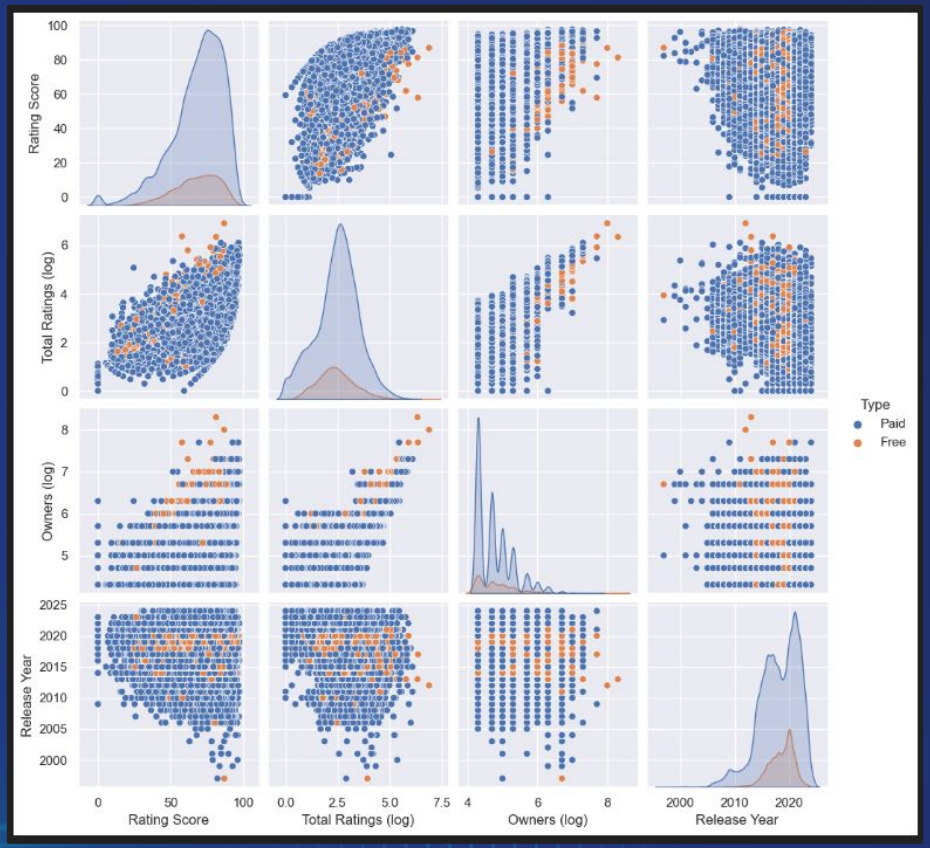
```
0    10000000-20000000
1     5000000-10000000
2     5000000-10000000
3     5000000-10000000
4      2000000-5000000
Name: owners, dtype: object
```



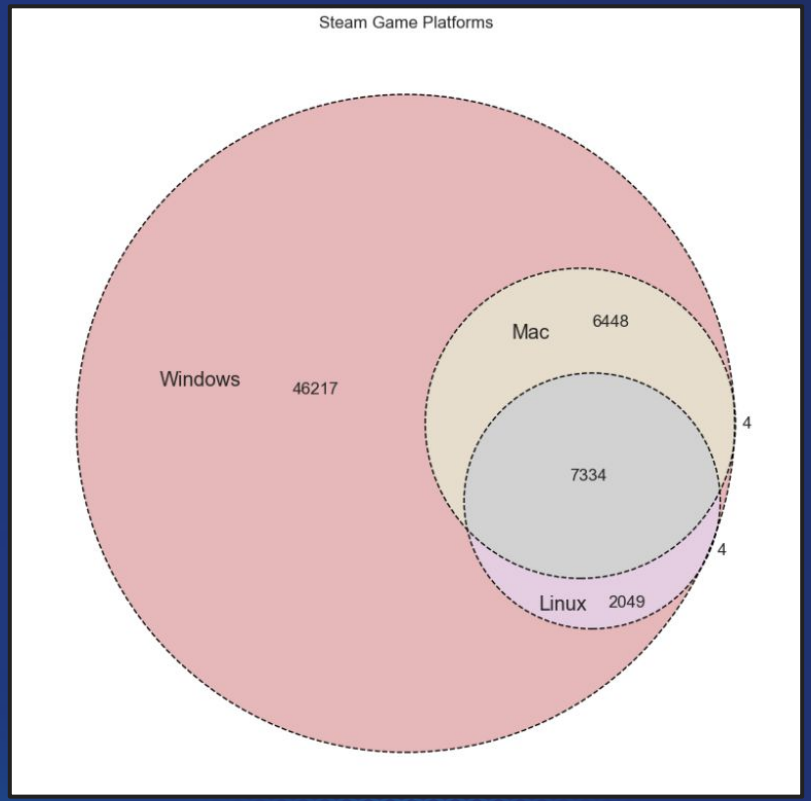
05

Exploratory Data Analysis (EDA)

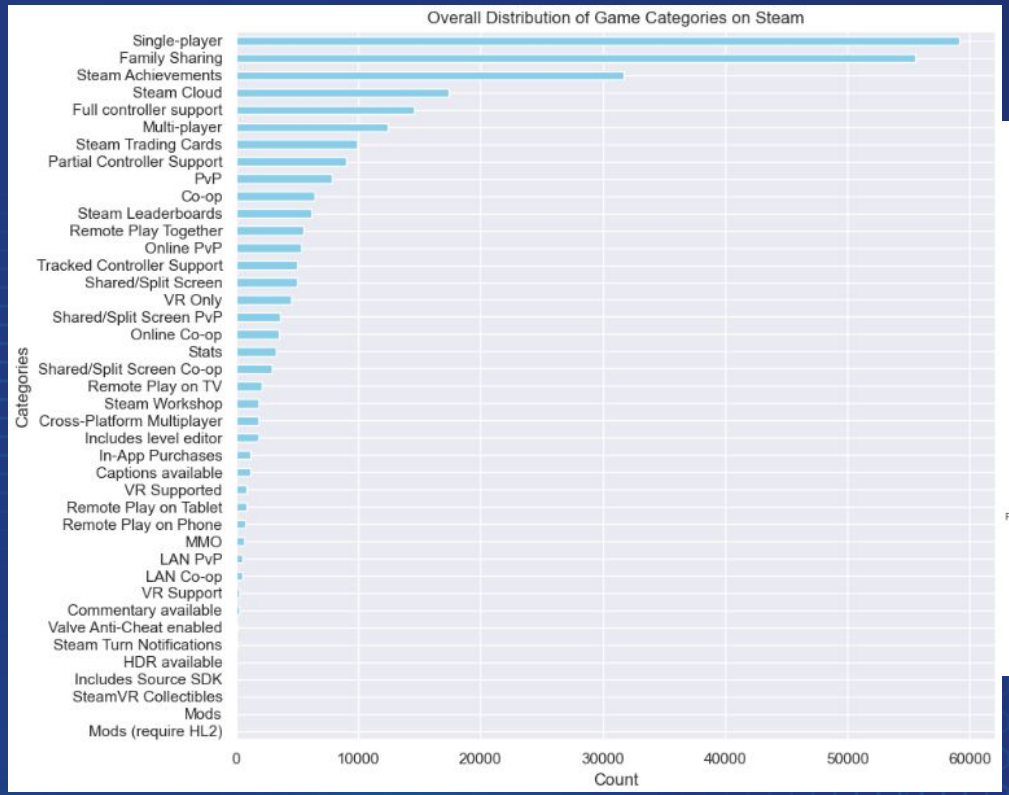
5. EDA

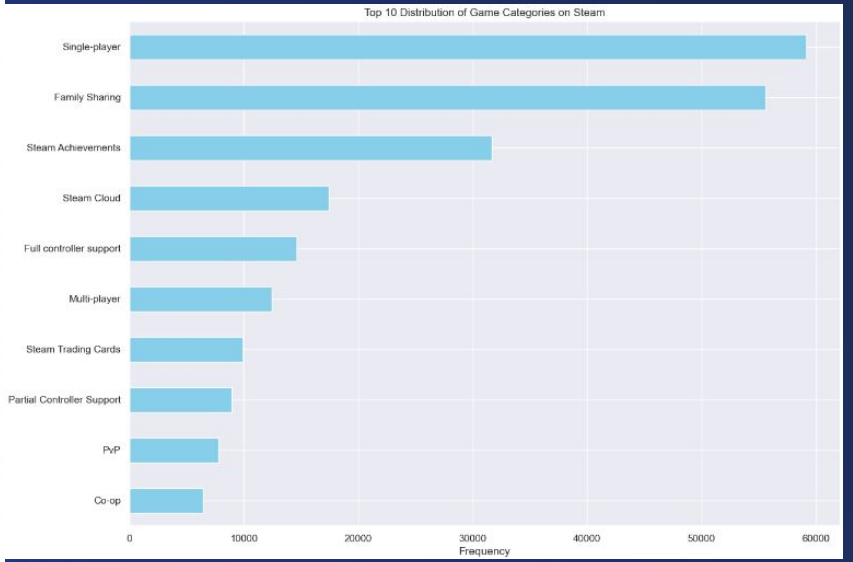

Rating Score
Total Ratings (log)
Owners (log)
Release Year
Type
Paid
Free

5.1 Platforms

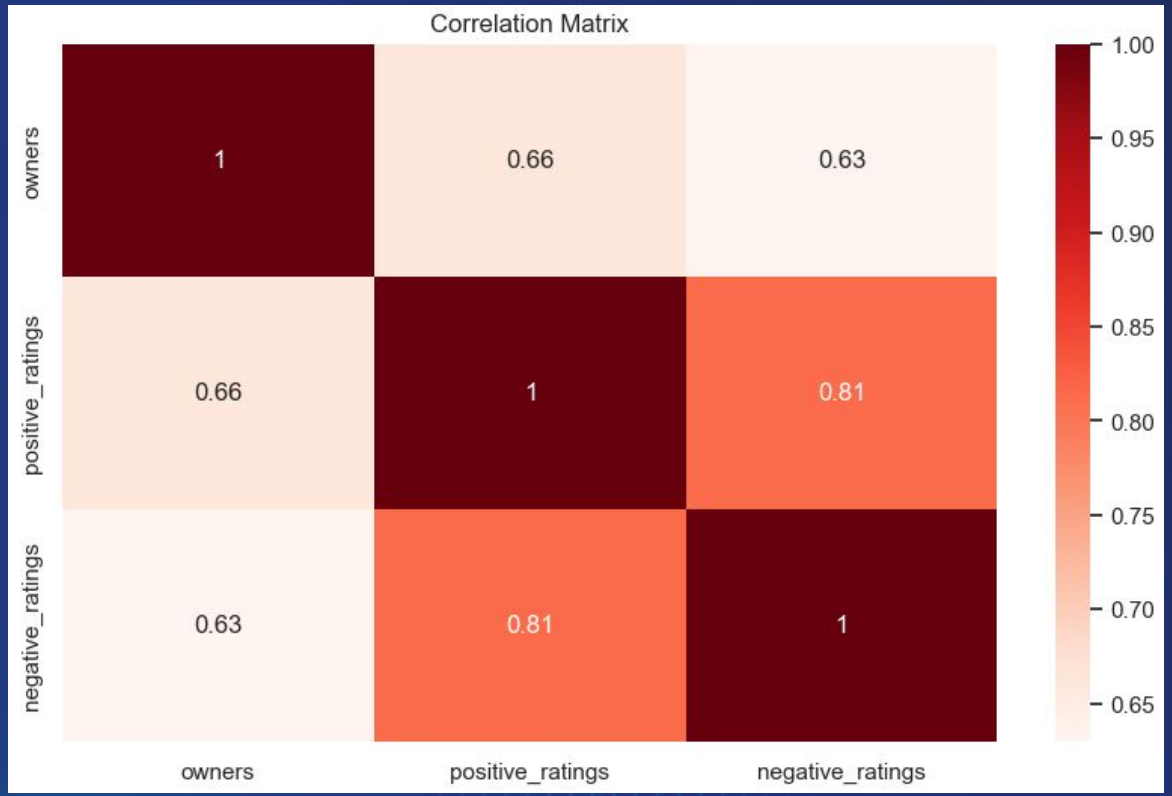


5.2 Categories vs Owners

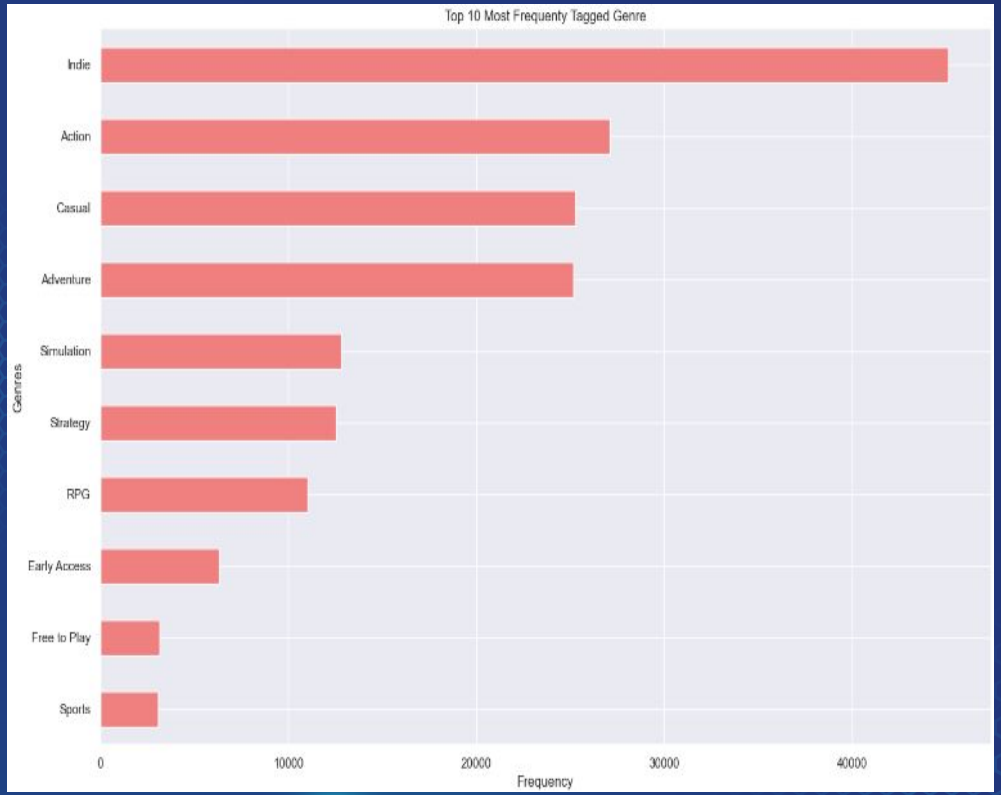


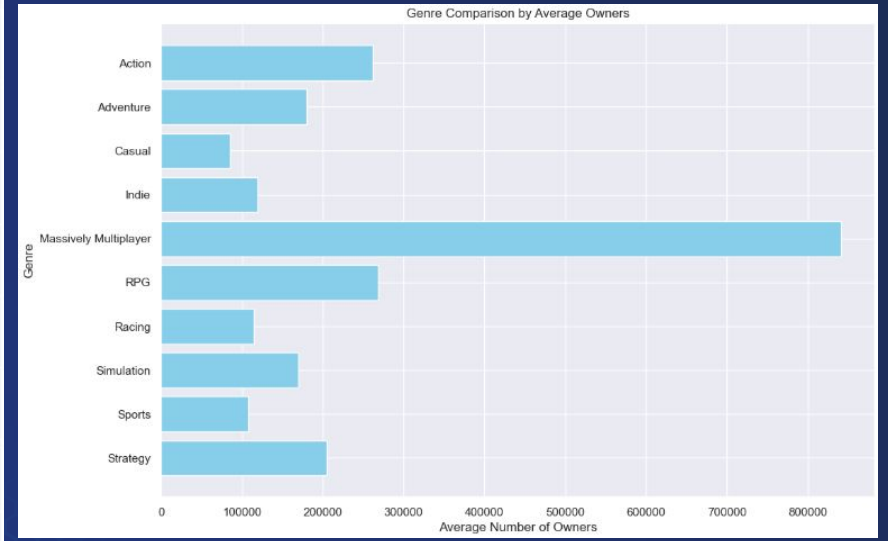


5.3 Owners vs Ratings

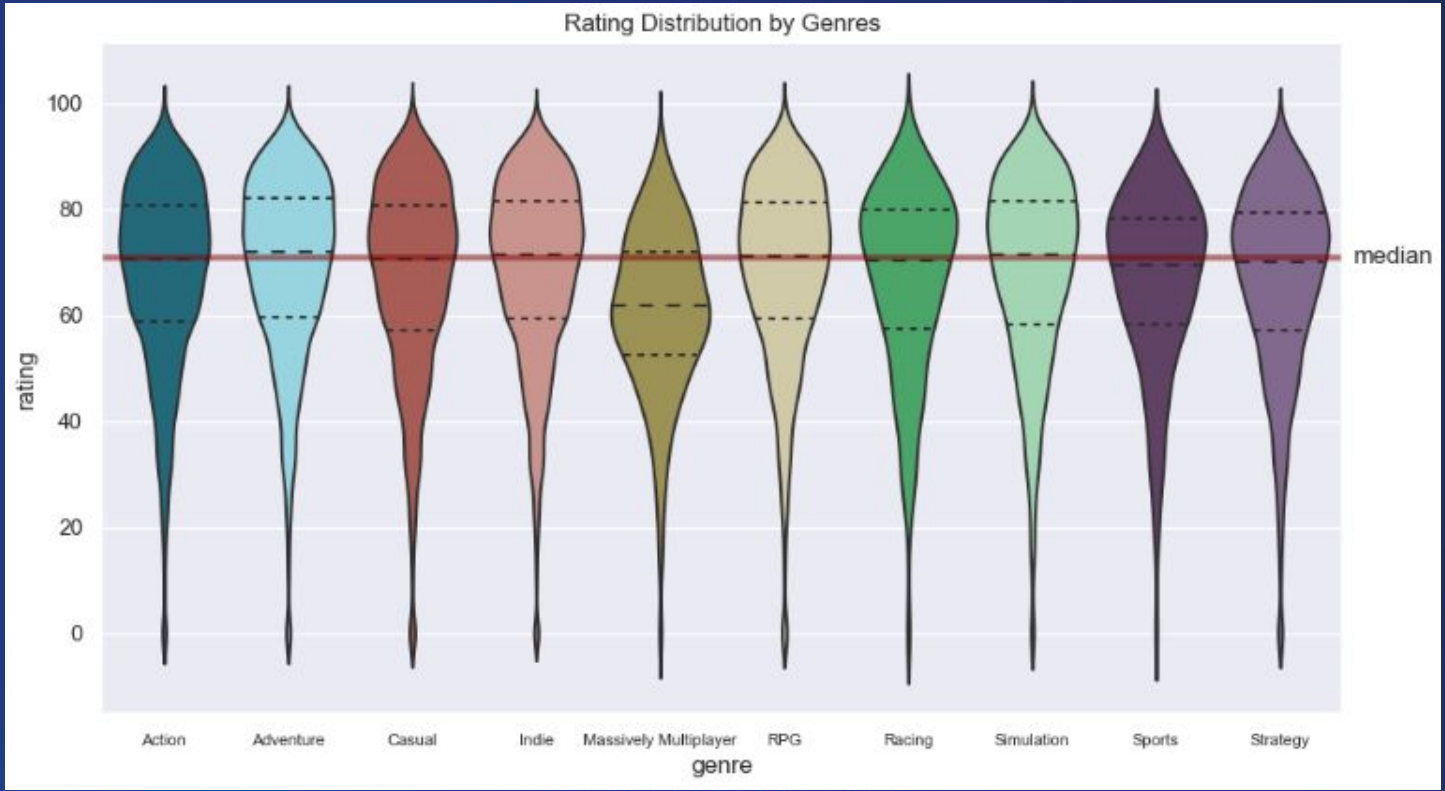


5.4 Genres

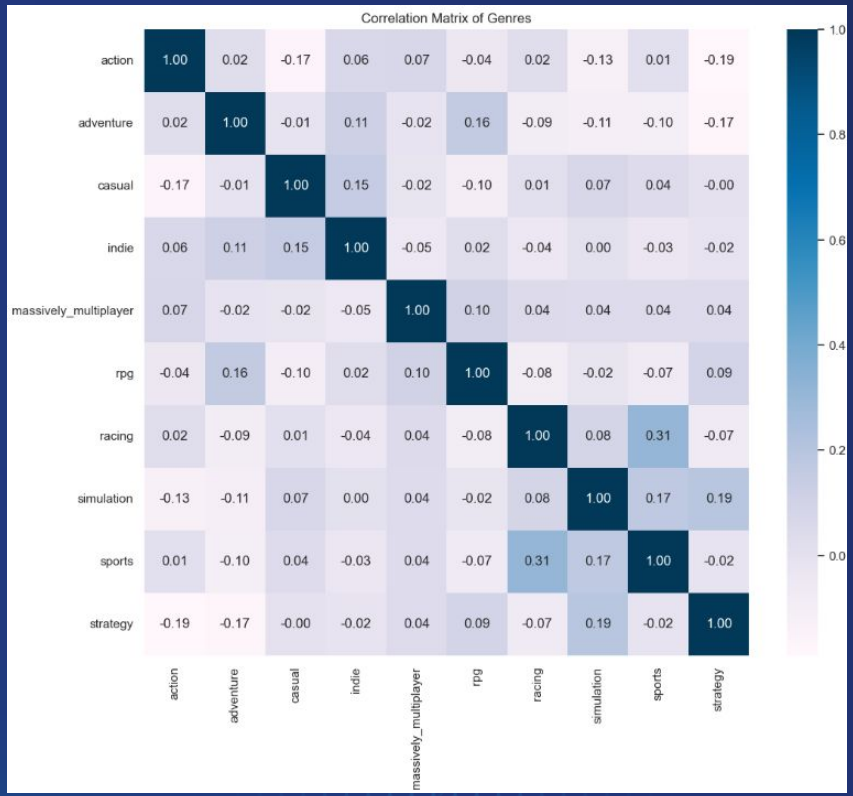

Top 10 Most Frequenty Tagged Genre
Indie
Action
Casual
Adventure
Simulation
Strategy
RPG
Early Access
Free to Play
Sports
Genres
0
10000
20000
30000
40000
Frequency

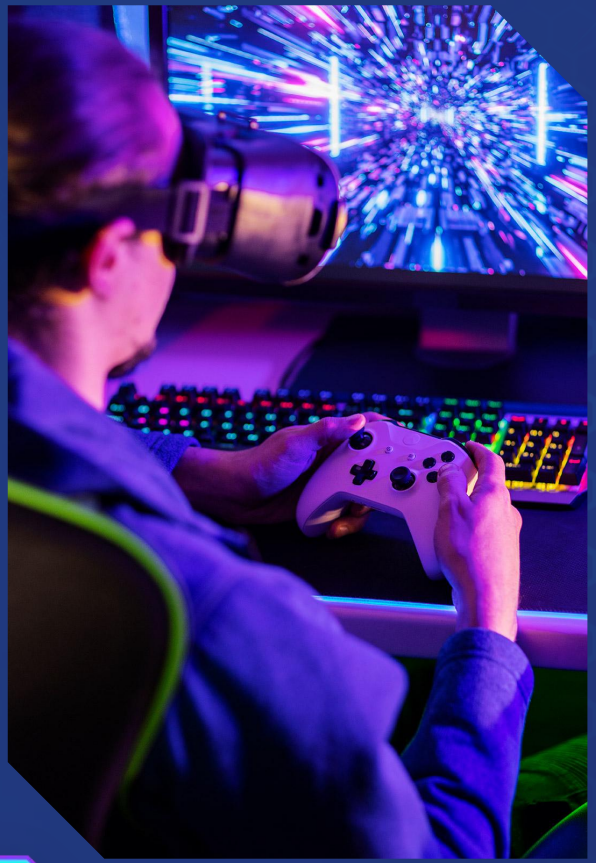

Genre Comparison by Average Owners
Action
Adventure
Casual
Indie
Massively Multiplayer
RPG
Racing
Simulation
Sports
Strategy
Genre
0
100000
200000
300000
400000
500000
600000
700000
800000
Average Number of Owners

5.4.1 Genres vs Rating



5.4.2 Genres Correlation





06

Machine Learning

Overview of Machine Techniques

Hierarchical Clustering

- Unsupervised Learning Method

Classification

- One Hot Encoding & Binary Classification

Linear Regression:

- Gradient Boosting Regressor

Hierarchical Clustering

Clustering SteamSpyTags:

- Number of player votes of a ‘genre’ per game

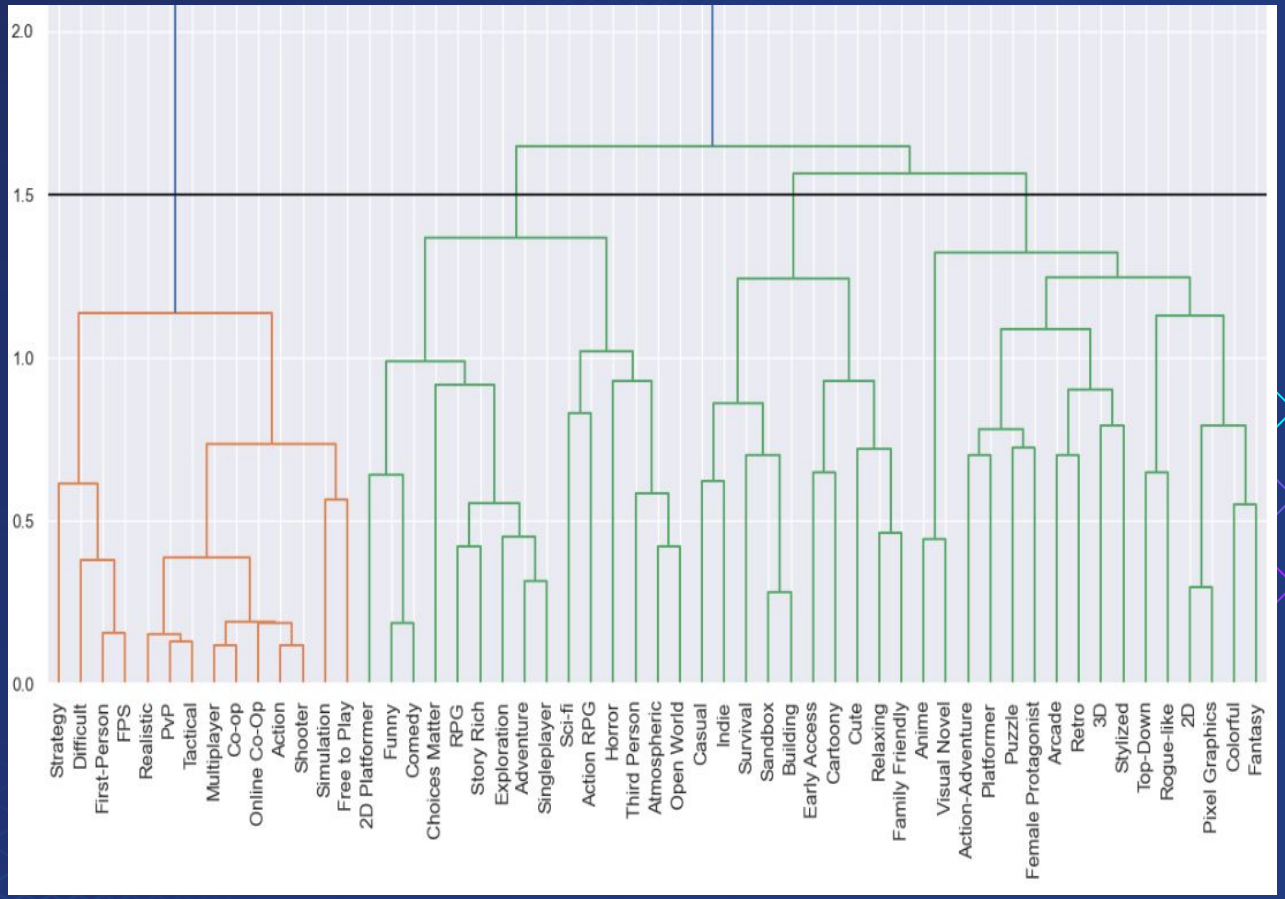
Ward’s Method:

- A criterion applied in hierarchical cluster

Dendrograms:

- Visualization of Hierarchy Tree

Hierarchical Dendrogram with Ward Linkage

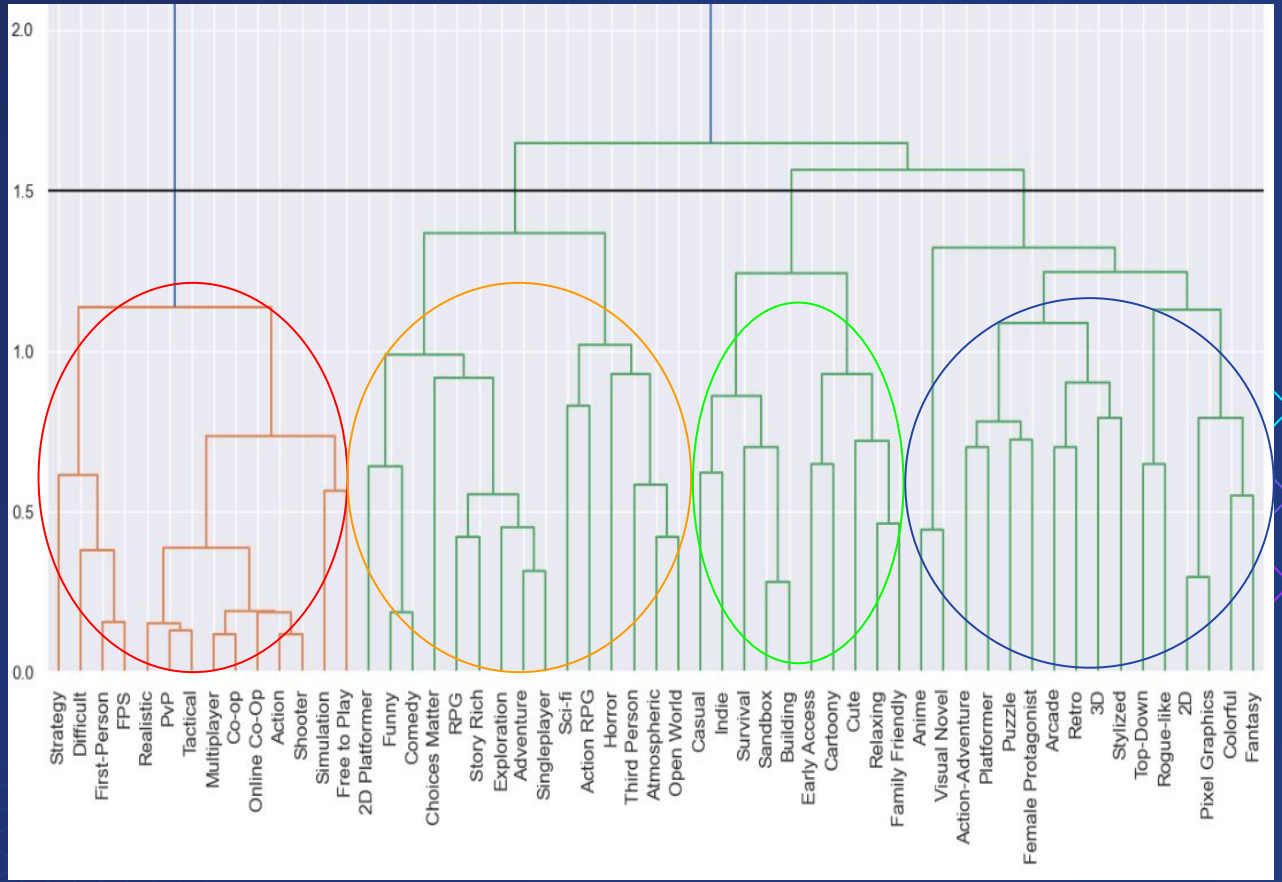


Cluster Evaluation

Identified 4 distinct groups under the 1.5 threshold

Cluster 1 -Competitive Action
Cluster 2 - Story Adventures
Cluster 3 - Casual Relaxed
Cluster 4 - Indie Anime

Hierarchical Dendrogram with Ward Linkage



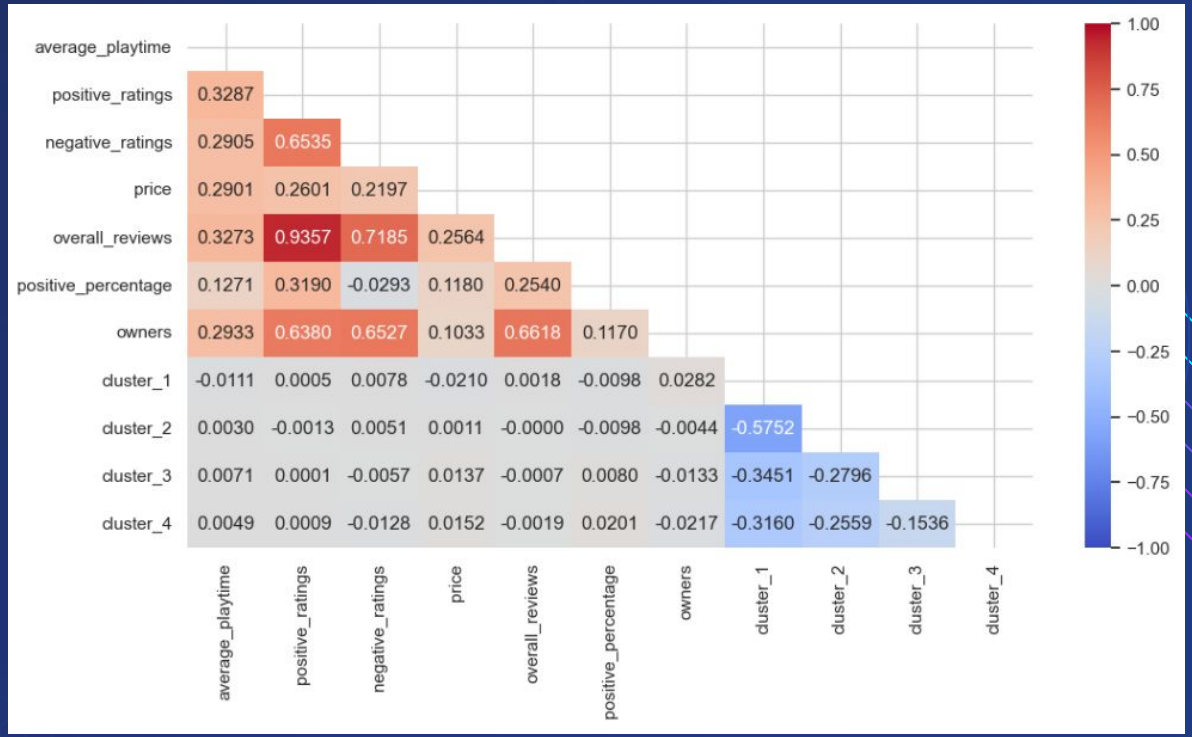
Correlations

There are low correlations with the clusters and other variables

Limited Predictive Power:
- This suggests tags and clusters might be noisy or even less relevant for prediction

Conclusion:
Besides genres, developers should focus on other factors of what can make their game more popular or successful.

Correlation with the clustered tags and the other variables



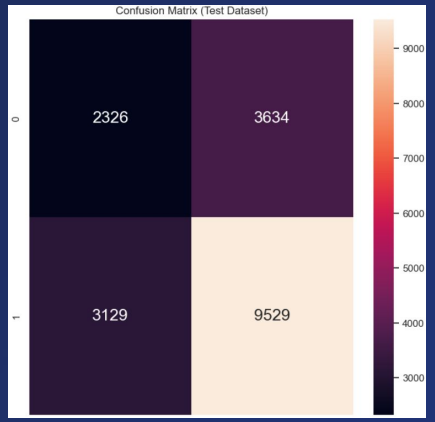
OHE Classification

0: Indicates Unsuccessful
1: Indicates Successful
- ratings > 80
- playtime > 0
- owners > 20,000

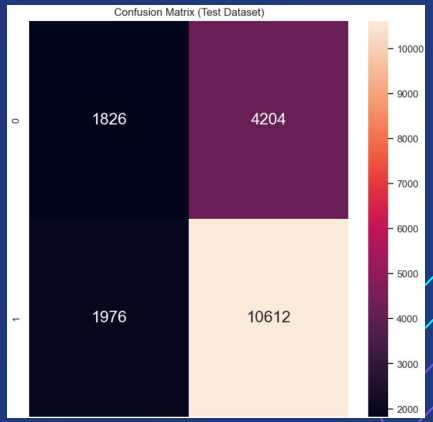
```
# create 'success' column
steam_df['success'] = np.nan

for row in range(len(steam_df)):
    success = (steam_df['rating'][row] > 80.0 or
               steam_df['average_playtime'][row] > 0.0 or
               steam_df['upper_own'][row] > 20000)

    if success:
        steam_df['success'][row] = 1
    else:
        steam_df['success'][row] = 0
```

Classification Models

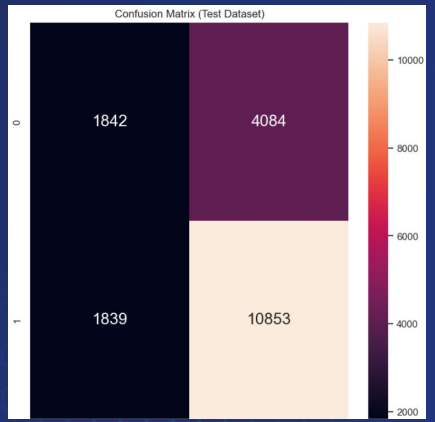
Decision Tree
F1: 73.8%



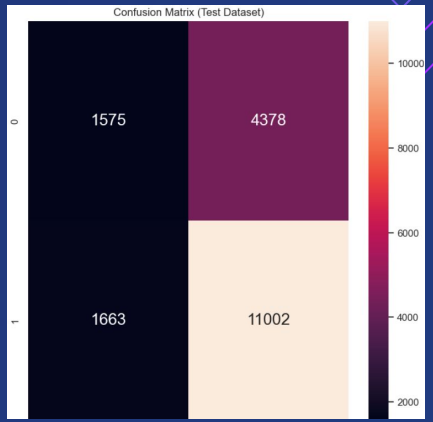
Random Forest V1
F1: 77.44%



Random Forest V2
F1: 78.56%

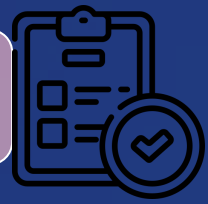


Random Forest V3
F1: 78.45%



Linear Regression

Making Prediction Models:



Response Variable:
“Average_playtime”

- Player retention and how long they spent playing the game

Predictors:

Top few numerical variables correlated to “average_playtime”

Linear Regression: Gradient Boosting

Feature	Linear Regression	Gradient Boosting Regressor
Model Type	Linear model	Ensemble of Decision Trees
Relationship	Assumes linear relationship	Captures non-linear relationships
Purpose	General-purpose regression	Improve accuracy, reduce variance, lower Mean Squared Error (MSE)
Strengths	Easy to interpret, computationally efficient	Handles complex relationships, captures interactions
Weaknesses	Limited to linear relationships	More complex, prone to overfitting, less interpretable (with many trees)

Beware of "overfitting"!
Use goodness of fit and monitor MSE!

Model 1: Numerical Variables with high correlation to “average_playtime”

```
steam_df.corr().average_playtime.sort_values(ascending = False).head(15)
```

```
selected_features = [
    "average_playtime",
    #"release_year",
    #"median_playtime",
    "price",
    "owners",
    "genre_casual",
    #"positive_ratings",
    "genre_rpg",
    "genre_indie",
    #"negative_ratings",
    "genre_early_access",
    "english",
    "genre_massively_multiplayer",
    "genre_strategy",
]
```

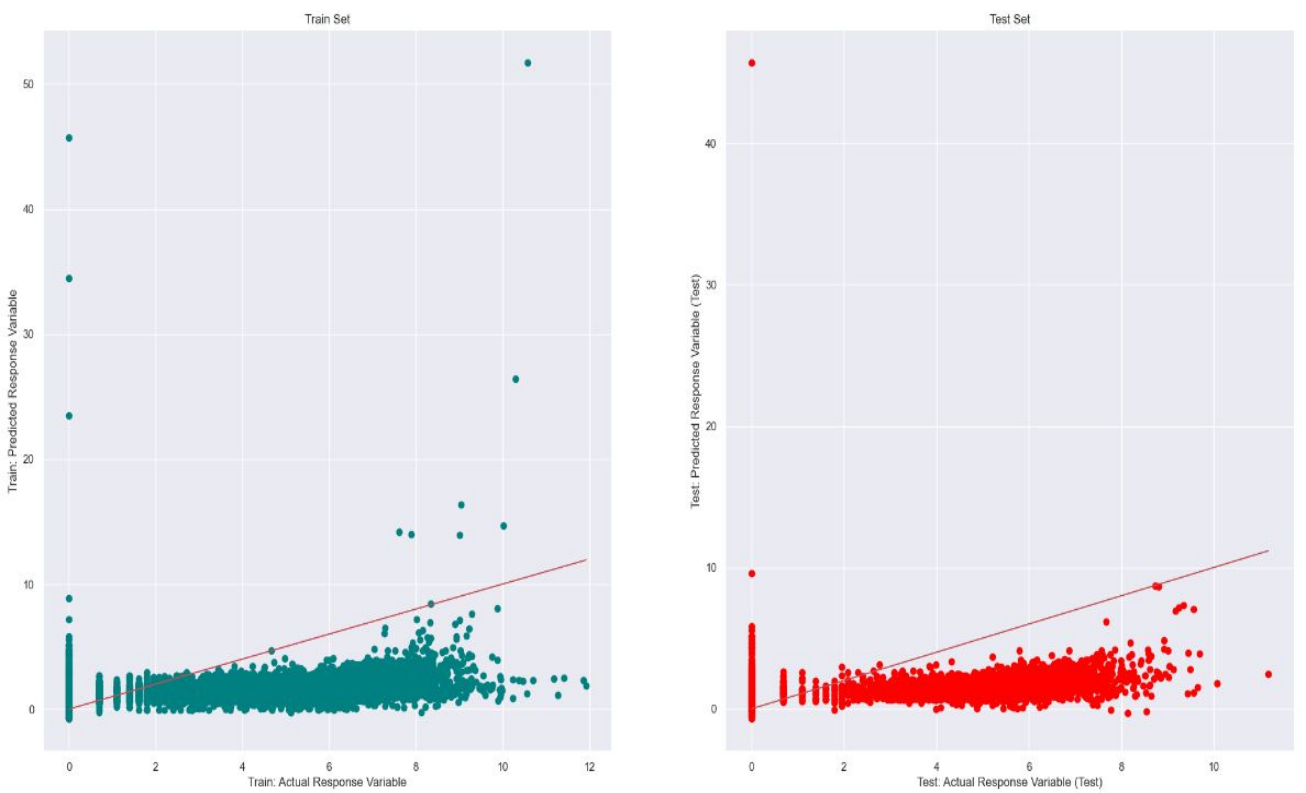
Selecting Features:

- Searching for top numerical variables that are correlated with the predictor.

Purpose of the Model:

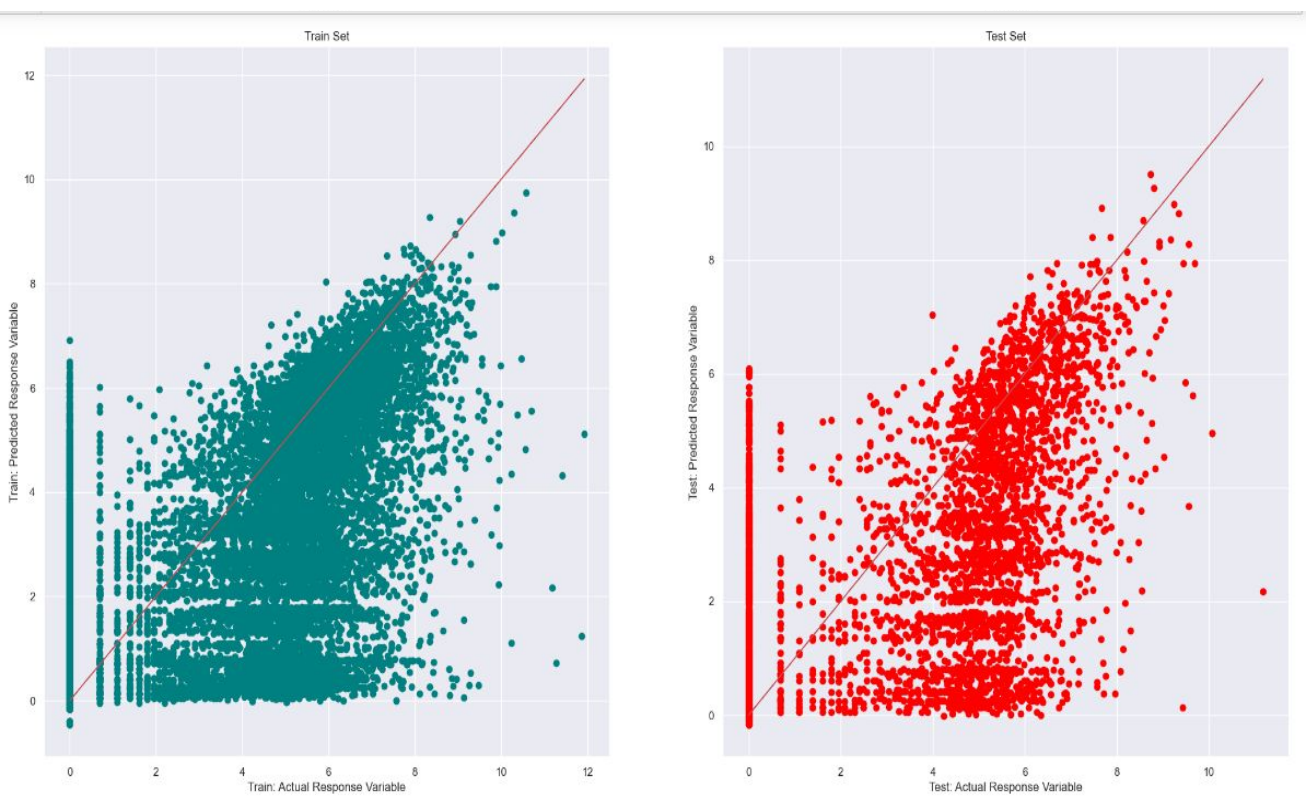
- The Model 1 is just an general overview of factors
- Consists of a mix of genres and other variables.

Linear Regression Model 1:




```
Linear Regression Model 1 Goodness of Fit:

Train Dataset:

Train Variance: 0.08649750514776988
Train Mean Square Error:  5.096174231253164

Test Dataset:

Test Variance: 0.06828544211003196
Test Mean Square Error:  4.99304938948119
```

Gradient Boosting Regressor Model 1:




```
GradientBoostingRegressor Model 1 Goodness of Fit

Train Dataset:

Train Variance: 0.5402530974721877
Train Mean Square Error:  2.5647990353214056

Test Dataset:

Test Variance: 0.5333638381725132
Test Mean Square Error:  2.500698720645873
```

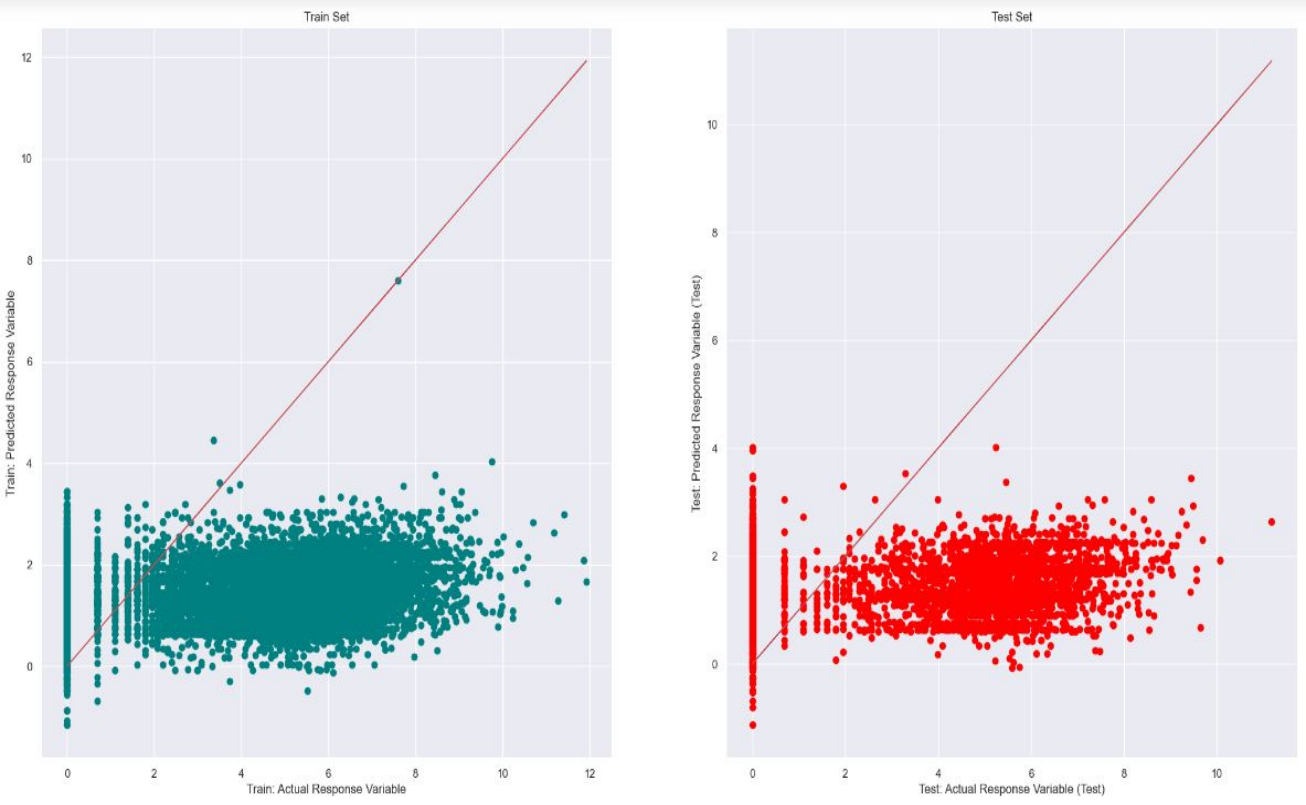
Model 2: Genres Only

```
selected_features = [
    "average_playtime",
    # Try Genre
    "genre_action",
    "genre_adventure",
    "genre_animation_and_modeling",
    "genre_audio_production",
    "genre_casual",
    "genre_design_and_illustration",
    "genre_early_access",
    "genre_education",
    "genre_free_to_play",
    "genre_game_development",
    "genre_gore",
    "genre_indie",
    "genre_massively_multiplayer",
    "genre_movie",
    "genre_nudity",
    "genre_photo_editing",
    "genre_rpg",
```

Selecting Features:

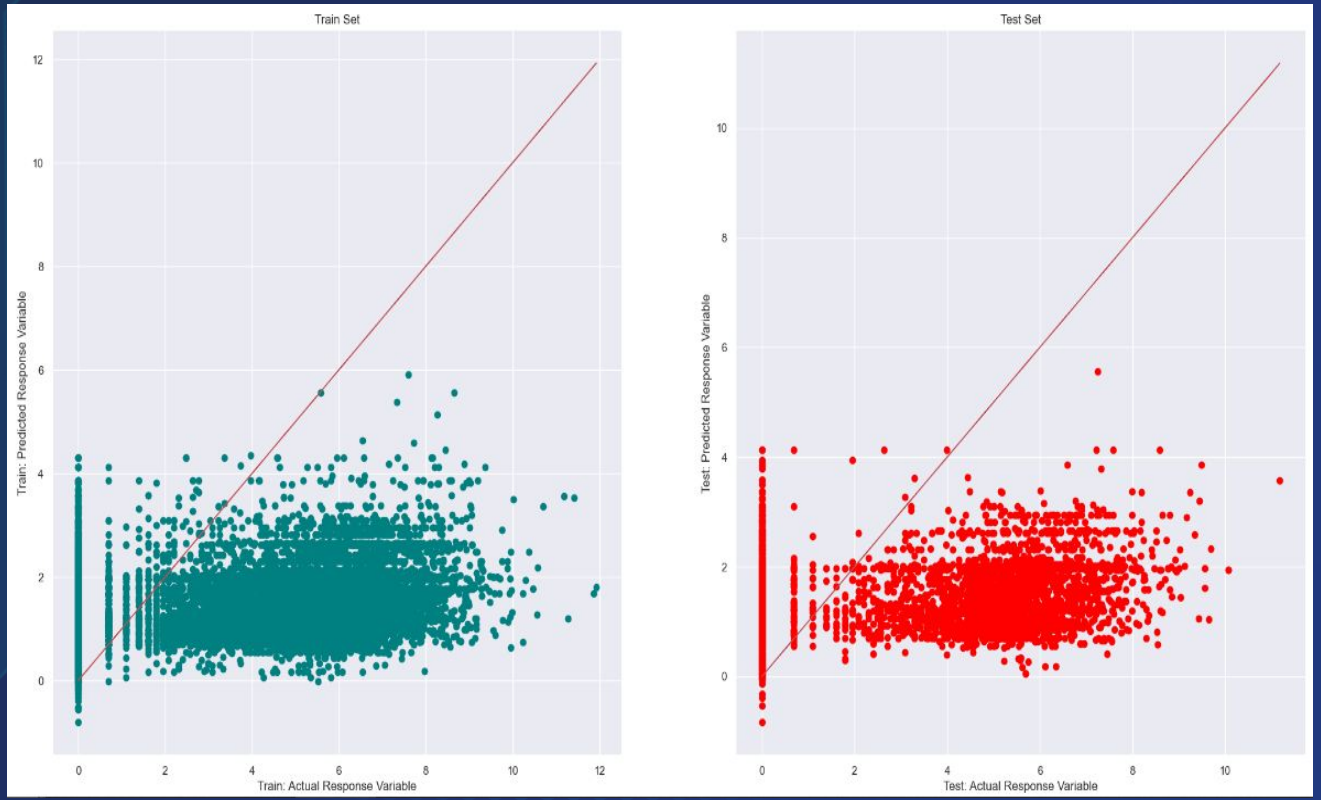
- Genre column unzipped into individual genres
- “genre_”

Linear Regression Model 2:




```
Linear Regression Model 2 Goodness of Fit:

Train Dataset:

Train Variance: 0.046506339006501474
Train Mean Square Error:  5.319273731818692

Test Dataset:

Test Variance: 0.04179761907601143
Test Mean Square Error:  5.134997379355059
```

Gradient Boosting Regressor Model 3:




```
GradientBoostingRegressor Model 2 Goodness of Fit

Train Dataset:

Train Variance: 0.06278718730242083
Train Mean Square Error:  5.228447445063965

Test Dataset:

Test Variance: 0.05515508501402522
Test Mean Square Error:  5.06341484736387
```

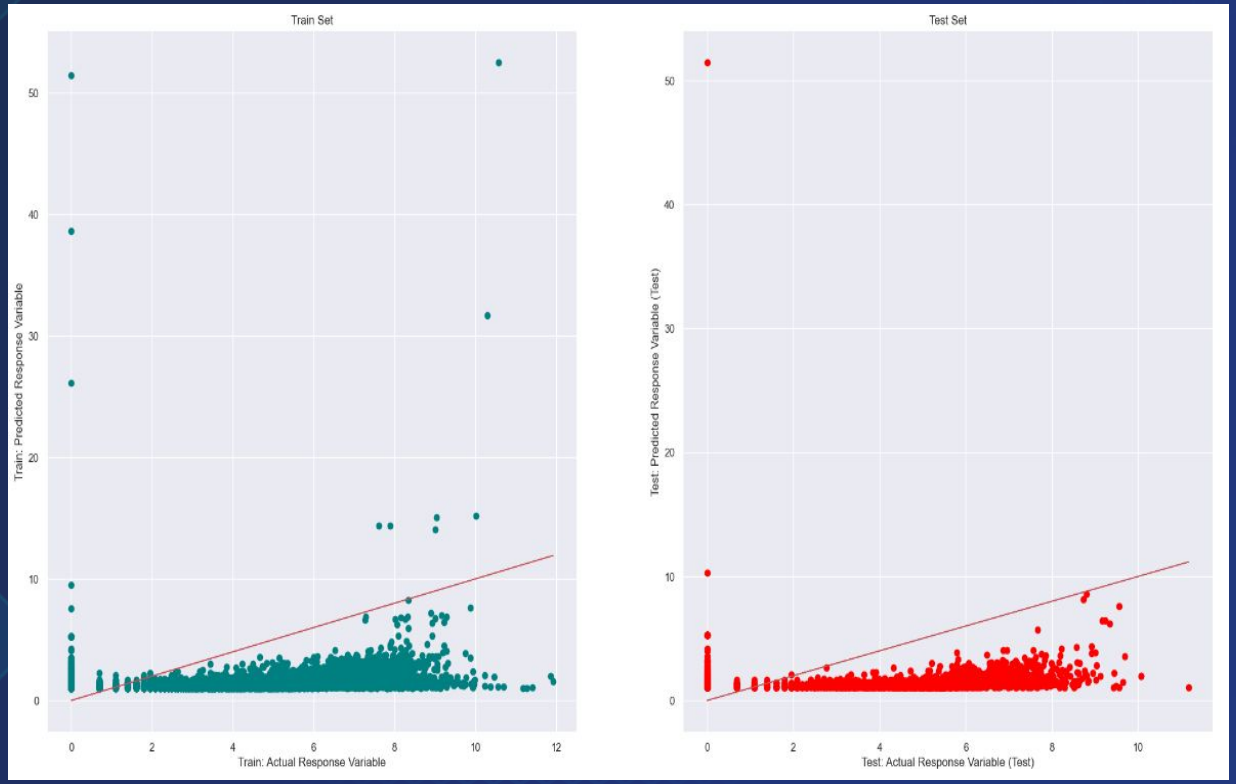
Model 3: `owners`, `overall_reviews`, `price`

```
selected_features = [
    "average_playtime",
    "owners",
    "overall_reviews",
    "price"
]
```

Selecting Features:

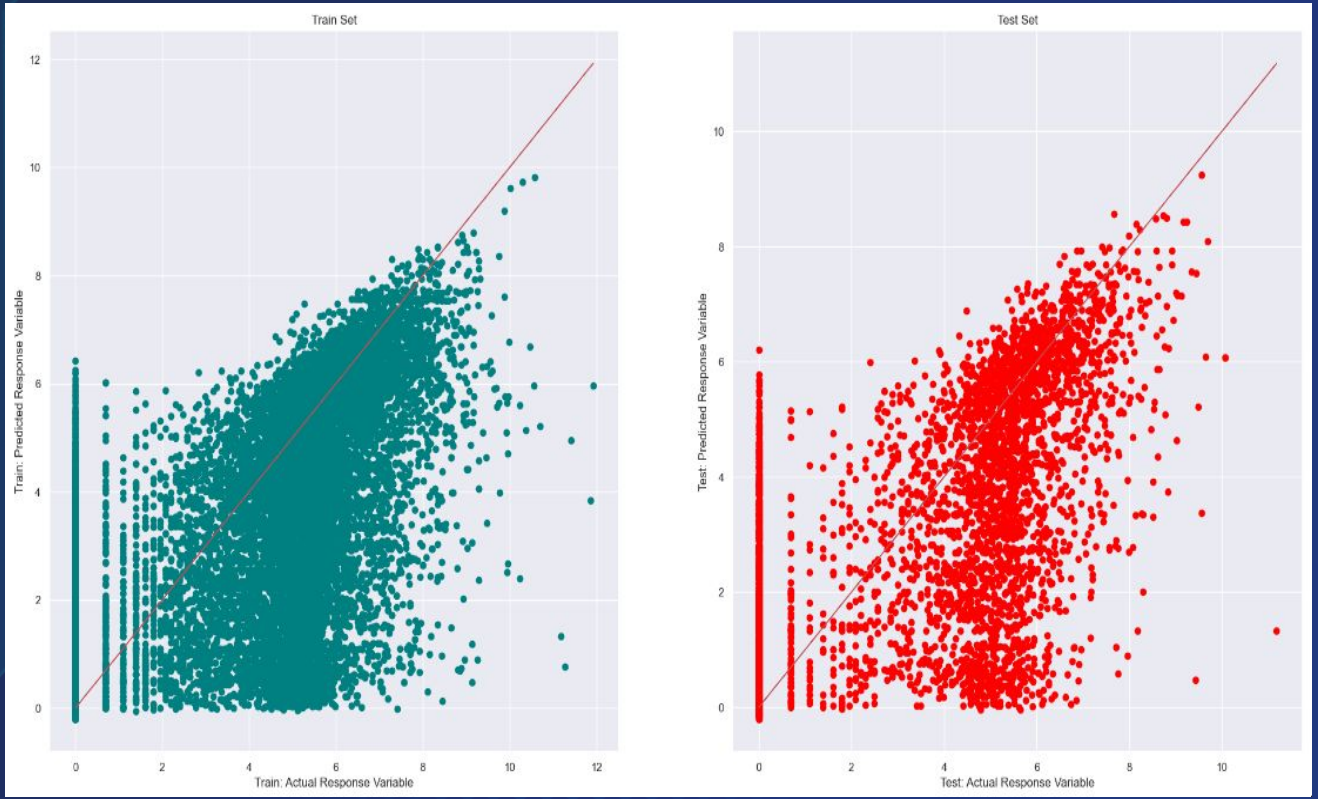
- Features that seem important: `owners`, `overall_reviews`, `price`

Linear Regression Model 3:




```
Linear Regression Model 3 Goodness of Fit:

Train Dataset:

Train Variance: 0.052081627543885234
Train Mean Square Error:  5.288170760632376

Test Dataset:

Test Variance: 0.03423966664710554
Test Mean Square Error:  5.175500374012928
```

Gradient Boosting Regressor Model 3:




```
GradientBoostingRegressor Model 2 Goodness of Fit

Train Dataset:

Train Variance: 0.06278718730242083
Train Mean Square Error:  5.228447445063965

Test Dataset:

Test Variance: 0.05515508501402522
Test Mean Square Error:  5.06341484736387
```

Comparing the Gradient Boosted Models

Model 1

Top variables correlated to average_playtime

Moderate Variance ~0.50

Model 2

Only Genres

Low Variance ~0.05, under 0.01

Model 3

`owners` , `overall_reviews` , `price`

Moderate High Variance ~0.65

Key Takeaways

Number of Owners:

- Popular games with more owners tend to have higher playtime

Overall Reviews:

- Games with more reviews suggest higher engagement → longer playtime

Price:

- Lower prices might attract more players
- “Higher-priced games as having higher value”
- Motivation to spend more time playing to justify the investment.

Genre:

- The diversity in player preferences leads to varying playtime across genres
- Resulting in a weaker correlation between genre and playtime.

7. Conclusion

Learning Points

Data Collection:

- Gathering Data using JSON and Web APIs

Data Cleaning:

- Handling multiple Data formats in excel
- Processing different data using python

Data Visualisation & Analysis:

- Applying course knowledge
- Learning new visualisation techniques

Machine Learning:

- Models:
  - Clustering, Linear Regression, Gradient Boosting Regressor
- Classification:
  - F1 Score, Accuracy

References

APIs:

1. https://partner.steamgames.com/doc/webapi_overview
2. https://steamspy.com/api.php
3. https://steamspy.com/

Motivation:

1. https://www.channelnewsasia.com/singapore/skull-and-bones-video-game-pirates-ubisoft-singapore-ps5-xbox-pc-4126386
2. https://www.washingtonpost.com/video-games/esports/2020/01/28/2010s-were-banner-decade-big-money-tech-esports-reaped-rewards/
3. https://www.statista.com/statistics/1330211/steam-peak-concurrent-players/

People:

1. https://nik-davis.github.io/posts/2019/steam-data-collection/
2. https://aldenyuan.medium.com/exploratory-data-analysis-on-steam-store-games-dataset-f4ff06da44eb