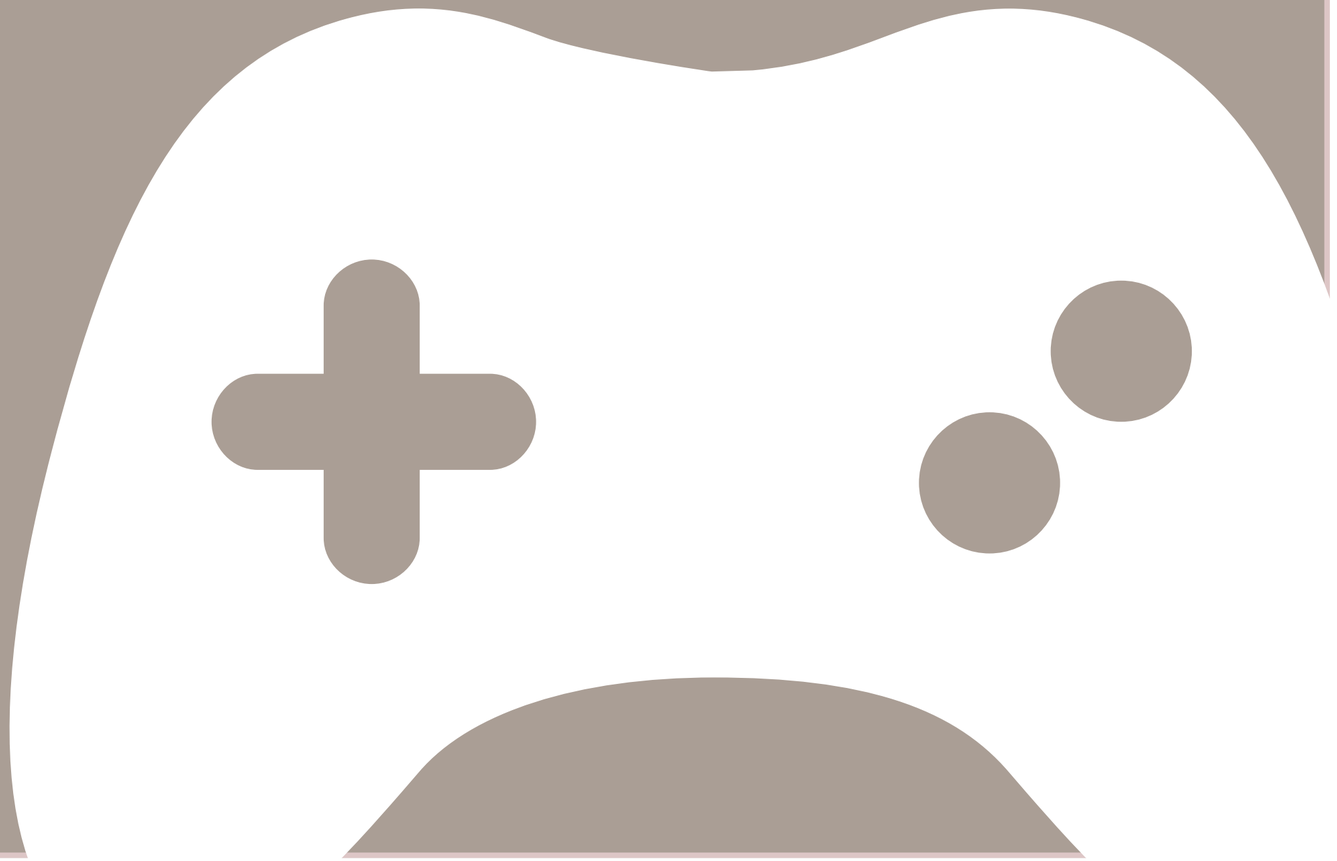
STEAM GAME EXPLORER – WHAT MAKES A GAME SUCCESSFUL?

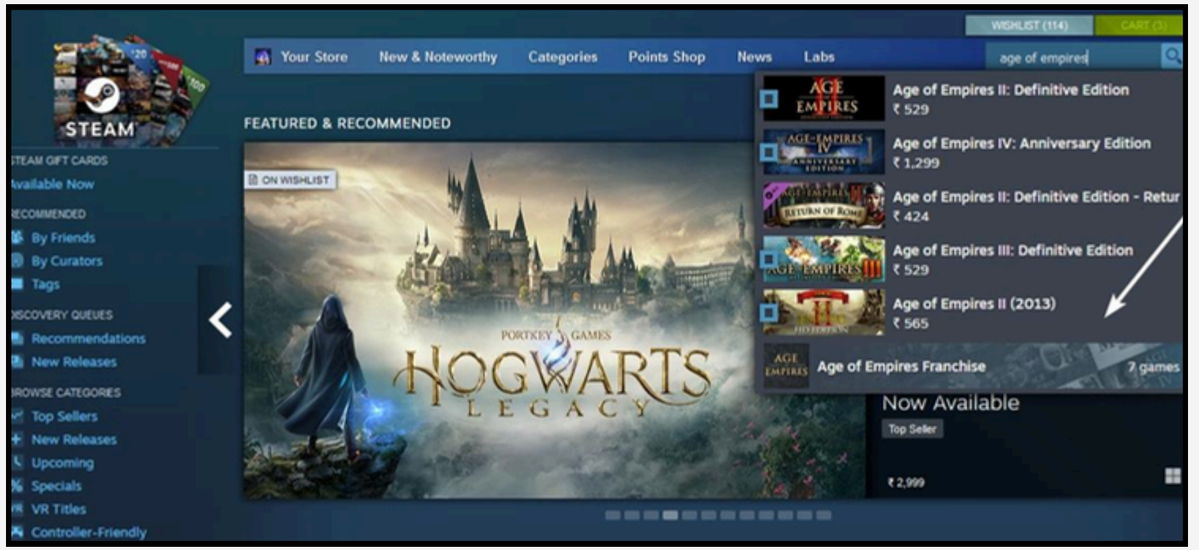
BY: JEZLEA ORTEGA
& ESTHER MALLEN



STEAM LANDSCAPE OVERVIEW

WHAT IS STEAM?

Steam is the world's largest digital distribution platform for PC gaming. It offers a marketplace where players can purchase, review, and discuss games, while developers can publish titles independently without needing a traditional publisher. With features like community reviews, tagging, curated lists, and downloadable content, Steam has become the central hub for PC gaming and the primary launchpad for indie developers.



LANDSCAPE OVERVIEW:

The Steam marketplace contains tens of thousands of games released across a wide range of genres and price points. Its open publishing model has accelerated growth, but it has also created an extremely crowded ecosystem where most games compete for limited user attention. Discoverability is increasingly driven by engagement metrics, including reviews, ratings, tagging behavior, and playtime. As a result, high-quality titles without early exposure often struggle to surface, making it critical to analyze how visibility forms and why some games rise while others remain hidden.

WHY IS THIS AN ISSUE TO INDIE DEVELOPERS?

ISSUE	DESCRIPTION	WHY THIS AN ISSUE
Oversaturation	Thousands of new releases every year, with many launching on the same day.	Indie creators don’t have the marketing power or established fanbases to stand out. Smaller titles are immediately pushed out of the “new releases” window before players even see them.
Heavy Reliance On Early Reviews	Discovery algorithms are driven by user signals such as review count, review positivity, and engagement.	Indie studios struggle to gather early reviews simply because fewer players know about the game. Without a surge of early interaction, the algorithm deprioritizes the game.
Favoritism Toward High-Engagement Games	Steam promotes games with high review counts, high playtime, and high wish list activity.	Larger studios easily generate engagement through marketing, influencers, and brand recognition, allowing their games to dominate recommendation areas. Indie games get overshadowed because they can't inflate engagement.

WHY IS THIS AN ISSUE TO INDIE DEVELOPERS?

ISSUE	DESCRIPTION	WHY THIS AN ISSUE
Limited Marketing Budgets	Big studios spend heavily on ads, trailers, influencer campaigns, and store placement.	Indie teams usually have small or nonexistent marketing budgets. Without sponsored visibility, trailers, or paid promotion, players never hear about the game.
High Competition Within Genres	Popular genres have thousands of competing indie titles.	The algorithm tends to boost titles with existing traction. New indie games struggle to differentiate themselves.
Lack of Publisher Support	Publishers handle marketing, PR, and platform negotiations.	Without a publisher, indie developers must handle all discovery, promotion, and community management themselves. This splits their limited time.
Visibility Snowball Effect	Games with early visibility gain even more visibility through Steam's automatic systems.	If an indie game doesn't receive early traction, popular games get pushed, while theirs sinks further.

PREVIOUS ATTEMPTS AT A SOLUTION

SOLUTION	HOW IT TRIES TO SOLVE THE ISSUE	WHERE IT FAILS
Curator System (2014)	Steam introduced “Curators”, aka influencers, reviewers, or organizations who recommend games to their followers.	Curators with large followings ended up favoring already-popular games, and it did not help unknown indie games gain initial traction.
Steam Tags (2014)	Developers and players could add descriptive tags to games, improving searchability.	Tags became oversaturated, and top-tagged games overshadowed smaller ones.
Main Capsule Rotation Algorithm (2017–2020)	Implemented rotating the “front-page capsule” so more games could appear and be featured.	Games with poor early metrics were still rarely shown, and rotation did not overcome the popularity snowball effect. Only a tiny fraction of indie titles saw meaningful exposure.

PREVIOUS ATTEMPTS AT A SOLUTION

SOLUTION	HOW IT TRIES TO SOLVE THE ISSUE	WHERE IT FAILS
Steam Spy (2015)	Estimates game ownership, sales numbers, and player demographics to give developers transparency into how well games were selling, trending genres and tags, competition, and how pricing affected performance.	Data became highly unreliable after 2018 when Steam restricted profile visibility. Estimates for smaller games were also often inaccurate or missing entirely. The tool provides post-release metrics, not predictive visibility guidance.
SteamCharts (2012)	Tracks concurrent player counts over time to try and offer visibility into how active a game is, whether a game is growing or declining, seasonal or event-driven player spikes, and engagement.	Fails to show why a game is or isn't discovered. Games with low visibility don't appear because they have no player data to graph. It also measures players, not sales, tags, reviews, or Steam algorithm behavior.

OUR STEAM EXPLORER FEATURES AND BENEFITS

1. **VISIBILITY RANKING** — A ranking within each genre, which lets developers compare how games perform relative to direct competitors.
2. **GENRE VISIBILITY TRENDS** — Charts that show how visibility scores change across genres over time to help identify which genres are oversaturated, rising, or struggling.
3. **HIDDEN GEMS DETECTOR** — A table of games with high ratings but low review counts to surface overlooked indie titles and shows where the algorithm is failing them.
4. **RELEASE-YEAR HEATMAP** — A genre × year heatmap, built from genre-year aggregates, that will reveal long-term trends, market shifts, and periods of high competition.
5. **PRICE BUCKET ANALYSIS** — Visuals showing how pricing affects visibility and review accumulation to help choose price ranges that maximize early traction.

OUR STEAM EXPLORER FEATURES AND BENEFITS

6 **TAG USAGE INSIGHTS**
Charts summarizing tag counts and common tag patterns to help devs tag their games strategically to increase searchability and discoverability.

7 **AGE VS. VISIBILITY SCATTERPLOTS**
Plots showing how game age relates to review count and visibility score to show whether games peak early or build traction slowly.

8 **TOP VISIBLE GAMES LEADERBOARD**
A sortable leaderboard showing the games with the highest visibility score that highlights what successful games have in common (tags, prices, genres).

TECHNOLOGIES UTILIZED:

OUR PYTHON CODE

DATA CLEANING AND LOADING CODE:

- Importing Python libraries needed for file handling, dates, arrays, and dataframes.
- Defining the path to the steam dataset from kaggle (games.csv) and the folders where processed files will be saved.
- Automatically creating the folders processed, aggregates, and discoverability so the program can store its outputs.
- Loading the raw Steam dataset into a pandas DataFrame.
- Defining a list of columns that the project actually needs (reviews, price, tags, genres, playtimes, etc.).
- Filtering the dataset so it only keeps the columns that exist in the file, preventing errors if a column is missing.
- Produces a clean, trimmed-down dataset.

```python
import os
from datetime import datetime
import numpy as np
import pandas as pd

PATH = "games.csv"
PROCESSED_DIR = "processed"
AGG_DIR = "aggregates"
DISC_DIR = "discoverability"
CURRENT_YEAR = datetime.now().year

for d in [PROCESSED_DIR, AGG_DIR, DISC_DIR]: #making directories for my csv files
    os.makedirs(d, exist_ok=True)

def load_and_clean():
    df = pd.read_csv(PATH)
    true_name = df["AppID"]
    true_release_date = df["Name"]
    true_estimated_owners = df["Release date"]
    true_price = df["Required age"]
    df["Name"] = true_name
    df["Release date"] = true_release_date
    df["Estimated owners"] = true_estimated_owners
    df["Price"] = true_price
    df = df.drop(columns=["AppID"])

    columns = [
        "Name",
        "Release date",
        "Estimated owners",
        "Price",
        "Genres",
        "Tags",
        "Positive",
        "Negative",
        "Average playtime forever",
        "Average playtime two weeks",
        "Median playtime forever",
        "Median playtime two weeks",
        "Developers",
        "Publishers",
        "Categories",
        "Recommendations",
    ]
    columns = [c for c in columns if c in df.columns]
    df = df[columns]
```

OUR PYTHON CODE

DATE PARSING & NUMERIC CLEANUP

- Checking if the dataset has a Release date column before trying to process it.
- Defining a safe parse_date() function that converts strings into real dates and returns NaT (missing) if the format is invalid.
- Applying the parse_date() function to every row of the Release date column.
- Extracting the release_year from each parsed date and storing it as a new column.
- If the Release date column does not exist, assigning NaN for release_year.
- Looping through important numeric columns and converting any string values into actual numbers.
- Dropping rows with no Name.
- Replacing missing or invalid values in Price, Positive, and Negative with 0 to keep the dataset consistent.

```
if "Release date" in df.columns:
     year_str = df["Release date"].astype(str).str[-4:]
     df["Release year"] = pd.to_numeric(year_str, errors="coerce").astype("Int64")
     df = df.drop(columns=["Release date"])
else:
     df["Release year"] = pd.NA

for col in [
    "Price",
    "Positive",
    "Negative",
    "Average playtime forever",
    "Average playtime two weeks",
    "Median playtime forever",
    "Median playtime two weeks",
]:
    if col in df.columns:
        df[col] = pd.to_numeric(df[col], errors="coerce")
df = df.dropna(subset=["Name"])

for col in ["Price", "Positive", "Negative"]:
    if col in df.columns:
        df[col] = df[col].fillna(0)

outpath = f"{PROCESSED_DIR}/steam_clean.csv"
df.to_csv(outpath, index=False)
return df
```

OUR PYTHON CODE

FEATURE ENGINEERING CODE:

- Creating a copy of the cleaned DataFrame to safely add new columns.
- Generating review_count by adding Positive + Negative reviews.
- Creating rating, calculated as positive reviews/total reviews.
- Computing log_review_count using log1p to scale large review counts.
- Building visibility_score by multiplying rating × log_review_count.
- Counting how many tags each game has and stores the value in a new tag_count column.
- Checking each game's genre list for the word "Indie" and stores the result as a boolean column is_indie.
- Creating a new price_bucket column by grouping games into price ranges (Very Cheap, Cheap, Mid, Expensive).
- Calculating each game's age by subtracting its release year from the current year, storing the result in age years.

```python
#feature engineering
def engineer_features(df_clean: pd.DataFrame):
    df = df_clean.copy()

    if "Positive" in df.columns and "Negative" in df.columns:
        df["review_count"] = df["Positive"] + df["Negative"] #New total number of reviews column will be positive + negative
    else:
        raise ValueError("Columns 'Positive' and 'Negative' are required to compute review_count.")

    df["rating"] = df["Positive"] / df["review_count"].replace(0, np.nan) #New column called rating will positive / (positive + negative)
    df["log_review_count"] = np.log1p(df["review_count"]) #New column, computing log of the review count log1p(x) = log(1 + x)
    df["visibility_score"] = df["rating"] * df["log_review_count"] #New column, computing visibility (how popular and well reviewed the game is) = ratin

    if "Tags" in df.columns: #new column, counting how many tags each game has
        df["tag_count"] = df["Tags"].fillna("").apply(
            lambda x: len([t for t in str(x).split(",") if t.strip()])
        )
    else:
        df["tag_count"] = 0
    if "Genres" in df.columns: #new column, checking if a game's genre list includes indie
        df["is_indie"] = df["Genres"].fillna("").str.contains("Indie", case=False)
    else:
        df["is_indie"] = False

    if "Price" in df.columns:
        def bucket_price(p): #sorting the game into price categories
            if p <= 5:
                return "Very Cheap (<= $5)"
            elif p <= 15:
                return "Cheap ($5-15)"
            elif p <= 40:
                return "Mid ($15-40)"
            else:
                return "Expensive ($40+)"
        df["price_bucket"] = df["Price"].apply(bucket_price) #new column
    else:
        df["price_bucket"] = "Unknown"
    if "Release year" in df.columns: #new column, checking how old each game is
        df["Age years"] = CURRENT_YEAR - df["Release year"]
    else:
        df["Age years"] = np.nan
```

OUR PYTHON CODE

AGGREGATION CODE:

Genre Level Aggregates:

- Checking if the Genres column exists and grouping the dataset by genre.
- For each genre, calculating how many games are in that genre, the average rating of the genre, and the average visibility score of the genre

Year-Level Aggregates

- Checks if the release year column exists.
- Groups the dataset by release year.
- For each year, calculating how many games were released that year and the average rating of that year's releases.

Returns All Aggregates

- Returns genre results and year results,

.

```
    outpath = f"{PROCESSED_DIR}/steam_features.csv"
    df.to_csv(outpath, index=False)
    return df

f build_aggregates(df_features: pd.DataFrame):
    df = df_features.copy()

    if "Genres" in df.columns: #grouping names by genre and then calculating the number of games, average rating, and average visibility of the group
        genre_stats = (
            df.groupby("Genres")
            .agg(
                n_games=("Name", "count"),
                avg_rating=("rating", "mean"),
                avg_visibility=("visibility_score", "mean")
            )
            .reset_index()
        )
        genre_stats.to_csv(f"{AGG_DIR}/genre_stats.csv", index=False) #saving the data to a new csv file
    else:
        genre_stats = None

    if "Release year" in df.columns: #grouping by year to get how many games were released and average rating per year
        year_stats = (
            df.groupby("Release year")
            .agg(
                n_games=("Name", "count"),
                avg_rating=("rating", "mean")
            )
            .reset_index()
        )
        year_stats.to_csv(f"{AGG_DIR}/year_stats.csv", index=False) #saving to new csv file
    else:
        year_stats = None
```

OUR PYTHON CODE

AGGREGATION, DISCOVERABILITY, AND MAIN CODE (1):

Genre-Year Aggregation:

- Checking that Genres and release_year exist before running.
- Grouping the dataset by genre and release year.
- Computes for each pair: number of games released in that genre/year, the average rating of that group, and the average visibility score of that group.

Discoverability Function;

- If Genres exist, creating a new column visibility_rank_in_genre.
- Ranking games within their genre based on visibility_score.
- If Genres missing, assigning NaN.

Hidden Gems Detection:

- Defines thresholds: high_rating = 0.9, low_reviews = 40
- Filtering games where: rating ≥ 0.9, review_count > 0, review_count ≤ 40

```
    if "Genres" in df.columns and "Release year" in df.columns:
        genre_year_stats = (
            df.groupby(["Genres", "Release year"]) #grouping my dataset by genre and release year
            .agg( #calculating how many games came out and average visibility score for each genre year pair.
                n_games=("Name", "count"),
                avg_visibility=("visibility_score", "mean")
            )
            .reset_index()
        )
        genre_year_stats.to_csv(f"{AGG_DIR}/genre_year_stats.csv", index=False) #saving the results in a new csv
    else:
        genre_year_stats = None
    return genre_stats, year_stats, genre_year_stats
```

```
def discoverability(df_features: pd.DataFrame):
    df = df_features.copy()

    if "Genres" in df.columns: #ranking visibility of games within each genre
        df["visibility_rank_in_genre"] = (
            df.groupby("Genres")["visibility_score"]
            .rank(ascending=False, method="dense")
        )
    else:
        df["visibility_rank_in_genre"] = np.nan
    #Looking to see if there are any well-rated games with not that many reviews, but with at least 1
    high_rating = 0.9
    low_reviews = 40

    hidden_gems = df[
        (df["rating"] >= high_rating) &
        (df["review_count"] > 0) &
        (df["review_count"] <= low_reviews)
    ].copy()

    hidden_gems.to_csv(f"{DISC_DIR}/hidden_gems.csv", index=False)
    top_visible = df.sort_values("visibility_score", ascending=False).head(200) #most visible games overall
    top_visible.to_csv(f"{DISC_DIR}/top_visible_games.csv", index=False)
    return hidden_gems, top_visible
```

```
def main(): #running all my code
    df_clean = load_and_clean()
    df_feat = engineer_features(df_clean)
    build_aggregates(df_feat)
    discoverability(df_feat)
main()
```

OUR PYTHON CODE

AGGREGATION, DISCOVERABILITY, AND MAIN CODE (2):

Top Visible Games Table

- Sorting all games by visibility_score (highest to lowest).
- Selecting the top 200 most visible games.

Main Function:

- Calling load_and_clean() to read the raw dataset and clean it.
- Calling engineer_features() to add new columns like rating, review_count, visibility_score, etc.
- Sending the engineered dataset to build_aggregates() to generate grouped summaries (genre stats, year stats, genre-year stats).
- Sending the same dataset into discoverability() to produce hidden gems and top visibility rankings.
- Runs everything automatically by calling main() at the end of the script.

```
    if "Genres" in df.columns and "Release year" in df.columns:
        genre_year_stats = (
            df.groupby(["Genres", "Release year"]) #grouping my dataset by genre and release year
            .agg( #calculating how many games came out and average visibility score for each genre year pair.
                n_games=("Name", "count"),
                avg_visibility=("visibility_score", "mean")
            )
            .reset_index()
        )
        genre_year_stats.to_csv(f"{AGG_DIR}/genre_year_stats.csv", index=False) #saving the results in a new csv
    else:
        genre_year_stats = None
    return genre_stats, year_stats, genre_year_stats
```

```
def discoverability(df_features: pd.DataFrame):
    df = df_features.copy()

    if "Genres" in df.columns: #ranking visibility of games within each genre
        df["visibility_rank_in_genre"] = (
            df.groupby("Genres")["visibility_score"]
            .rank(ascending=False, method="dense")
        )
    else:
        df["visibility_rank_in_genre"] = np.nan
    #Looking to see if there are any well-rated games with not that many reviews, but with at least 1
    high_rating = 0.9
    low_reviews = 40

    hidden_gems = df[
        (df["rating"] >= high_rating) &
        (df["review_count"] > 0) &
        (df["review_count"] <= low_reviews)
    ].copy()

    hidden_gems.to_csv(f"{DISC_DIR}/hidden_gems.csv", index=False)
    top_visible = df.sort_values("visibility_score", ascending=False).head(200) #most visible games overall
    top_visible.to_csv(f"{DISC_DIR}/top_visible_games.csv", index=False)
    return hidden_gems, top_visible
```

```
def main(): #running all my code
    df_clean = load_and_clean()
    df_feat = engineer_features(df_clean)
    build_aggregates(df_feat)
    discoverability(df_feat)
main()
```

OUR TABLEAU INTERFACE

VISIBILITY RANKING AND GENRE VISIBILTY TRENDS:

Visibility Ranking

- Shows the games from highest to lowest rank in their genre
- Shows the visibility score
- Each game includes age, price, rating, and review count

Genre Visibility Trends

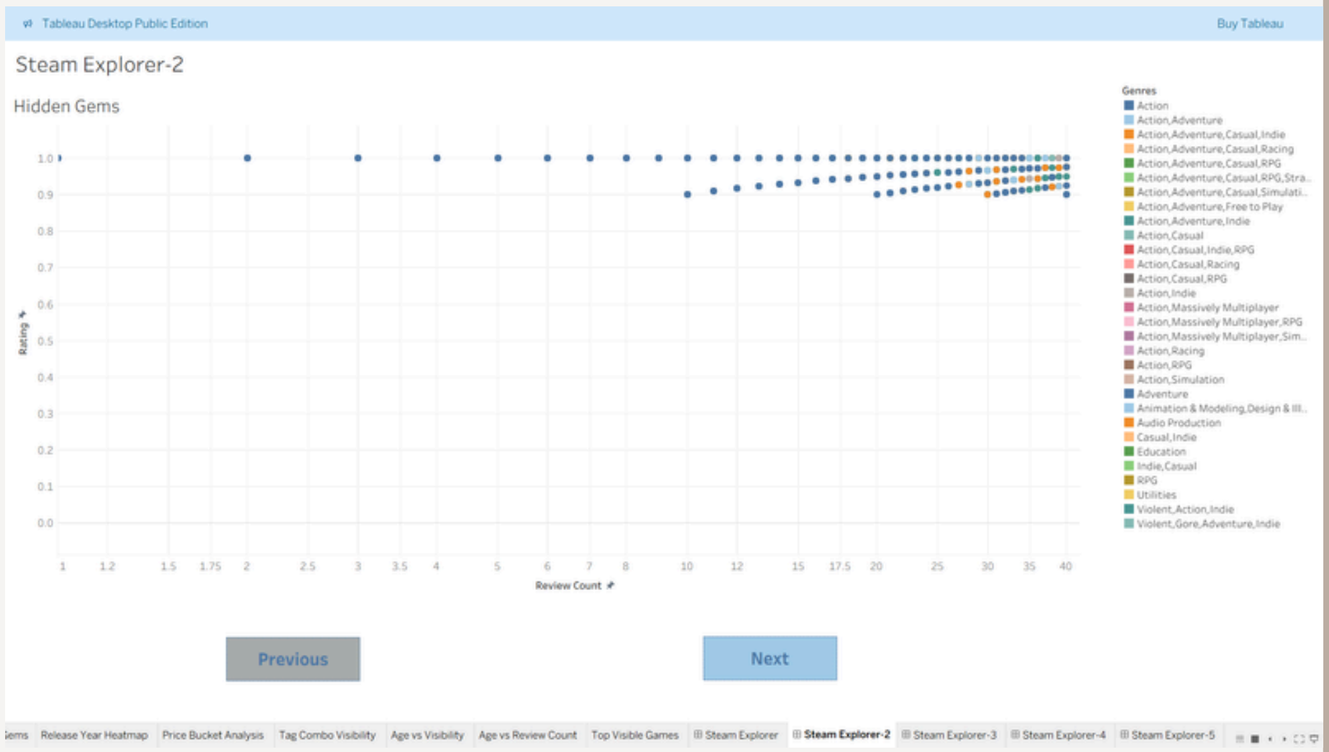
- Compares the average visibility score by year from all the genres in the dataset
- Shows the top 10 genres with highest accumulated visibility score.
- This chart can filter any data from the visibility ranking chart

OUR TABLEAU INTERFACE

HIDDEN GEMS:

Hidden Gems

- Shows the games that have high ratings but low review count
- In the right side is the genre of such games, sorted but color.
- This are the games that get buried because of little marketing or small budget. Sometimes the release date is at the same time as big games, so people forget about them.



OUR TABLEAU INTERFACE

RELEASE YEAR HEATMAP AND PRICE BUCKET ANALYSIS:

Release Year Heatmap

- Shows genres by year
- Color shows the amount of games releases under that genre (darker color = higher quantity of games)
- The chart is sorted by descending order showing the genre with most games released at the top

Price Bucket Analysis

- There are four price buckets and each is shown with the review count and average visibility score
- The color indicates the average ratings among those price buckets (darker color = higher avg. rating)

OUR TABLEAU INTERFACE

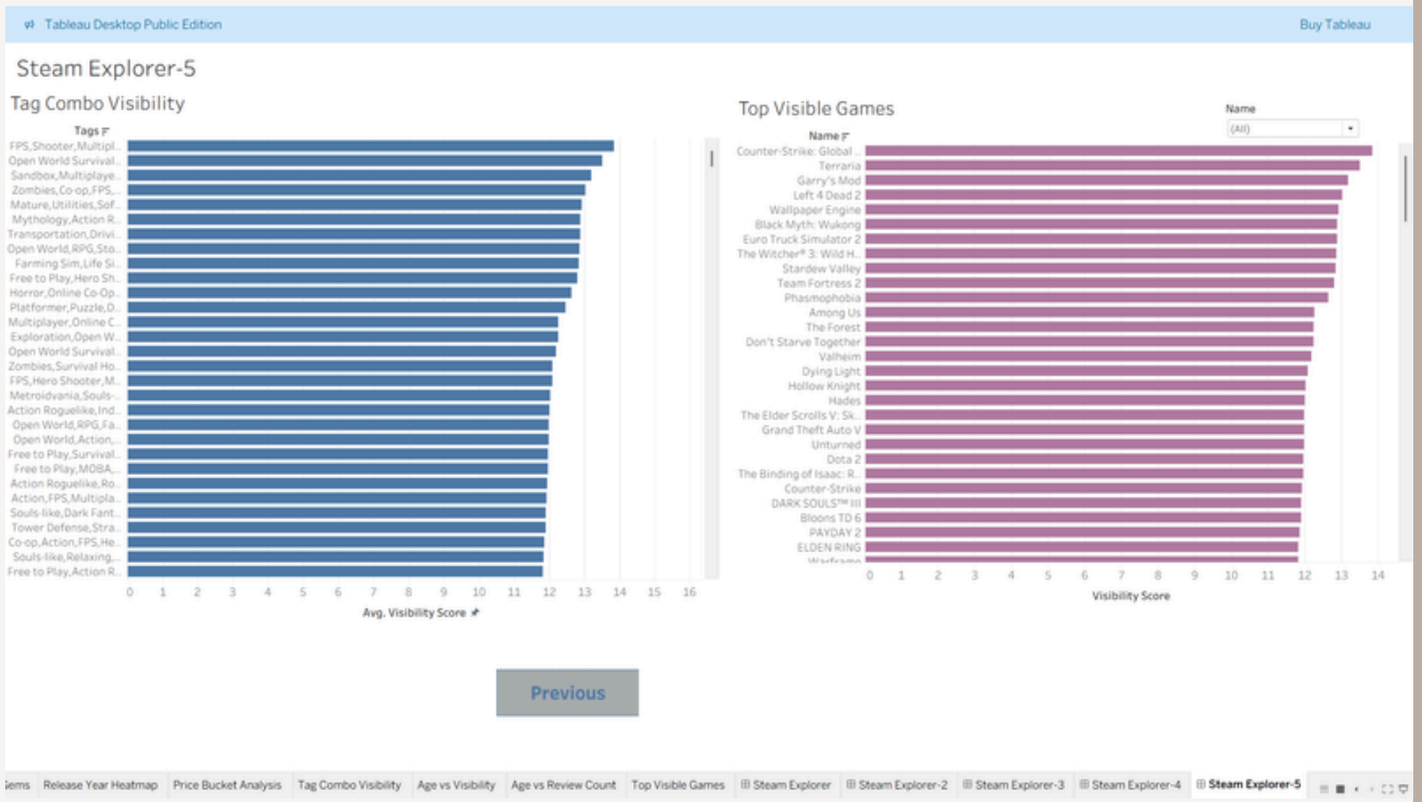
TAG COMBO VISIBILITY AND TOP VISIBLE GAMES:

Tag Combo Visibility

- Shows the top game tags that have more visibility on Steam

Top Visible Games

- Displays the top visible games based on the visibility score
- When a game is selected the tag combo chart will show the tag combos of that specific game



END RESULTS & SIGNIFICANCE

Findings

- Casual, Indie is the genre with most games and it falls in the cheap price bucket
- Over the years visibility of games based on genres decreases due to oversaturation
- A game visibility is affected by the review count and ratings even if it’s old

Limitations

- Some data got lost while creating the visualizations
- Some rows are null even though they have data in the processed CSVs
- Tableau has a limit for the data size
- Some filters don’t work very well with some data

DASHBOARD DEMO