Final Project Proposal

Fruity Robo

Project Title:
Understanding the Commercial Success of Steam Games: A Data-Driven Analysis

Team Members:
- Yuxun He
- Tingjun Chen
- Boran Zhang

1. Research Question

The goal of this project is to study the **commercial success** of video games published on *Steam*, the largest digital PC gaming marketplace. And because Steam does not release official sales data, we use widely accepted **proxy variables** for commercial success, such as:

- Number of user reviews or recommendations
- Metacritic score
- User rating categories

Our research questions include:

1. **How do pricing and discount strategies influence the commercial success of a game?**
2. **Do certain game genres or tags cluster into groups with similar commercial outcomes?**
3. **Are there seasonal or temporal patterns—e.g., release month or quarter—that predict success?**
4. **Do large publishers systematically perform better than independent developers?**

Understanding these patterns is important to both the gaming industry and to statistical practice. This project connects real-world economic behavior, user engagement, and digital marketplace dynamics with statistical modeling, visualization, and modern data engineering.

2. Planned Work

We plan to:

1. **Construct a large dataset** of Steam games using:
    - The Steam Web API
    - Web scraping of store pages for tags and user review counts, if needed
2. **Perform exploratory data analysis**, including:
    - Distribution of prices, discounts, ratings
    - Yearly and monthly release patterns
3. **Build visualizations** to examine relationships among:
    - Price vs. user review volume
    - Discount percentage vs. commercial success
    - Publisher size vs. user ratings
4. **Cluster games** using features such as genre, tag vectors, pricing, and platform support.
5. **Generate a final data analysis product**, including:
    - Reproducible code
    - Clean dataset
    - Visual summary report

3. Data Sources

We will obtain game-level data from two complementary and fully accessible sources:
(1) the SteamSpy API and
(2) web scraping of the Steam Store.
Both sources have been tested successfully, and the data retrieval procedures are documented in the Appendix.

(A) SteamSpy API

SteamSpy is a publicly accessible service that aggregates large-scale statistics across the Steam platform. It provides structured JSON data containing high-value quantitative variables, including:

- game identifiers (`appid`, `name`)
- developer and publisher information
- estimated ownership ranges
- long-term and recent player activity
- price and discount information
- user ratings (positive/negative counts)
- community-assigned tags

We have verified that the SteamSpy API can be queried programmatically and returns a clean and well-structured dataset. (See Appendix for the working retrieval script.)

(B) Web Scraping of the Steam Store

Some important commercial attributes—especially those related to user perception and pricing display—are not available through SteamSpy. To supplement these missing fields, we scrape individual Steam Store pages using `rvest`.

From each game page, we can extract:

- overall and recent user review summaries
- displayed price or discounted price
- release date
- developer and publisher fields
- user-assigned tags (e.g., *MOBA*, *Open World*, *Indie*)

We have verified that the Steam Store HTML is fully accessible from our environment and that these variables can be reliably extracted. (Full scraping implementation is provided in the Appendix.)

Notice: The specific data collection workflow may be adjusted as the project evolves and additional analytical needs arise.

4. Programming Paradigms

We will combine:

1. **Functional programming** for data extraction and cleaning
2. **Vectorized operations** for efficient manipulation
3. **Object-oriented patterns** (class-based data collectors, if needed)
4. **Pipeline-style workflows** using tidyverse principles

This combination makes sense because:

- API calls are repetitive → functional approach improves clarity
- Data wrangling with tidyverse is highly efficient for tabular datasets
- Web scraping benefits from modularity and reusable functions
- Statistical modeling fits naturally into tidy workflows

5. Software and Packages

We plan to use:

R

- `httr2` / `httr` — API calls
- `jsonlite` — JSON extraction
- `rvest` — web scraping
- `dplyr`, `tidyr`, `purrr` — data wrangling
- `ggplot2` — data visualization

- `cluster`, `factoextra` — clustering
- `glmnet` or `tidymodels` — modeling framework
- `lubridate` — handling dates

Additional packages and tools may be incorporated as needed during data extraction, modeling, and visualization.

6. Planned Data Analytic Product

Our final product will include:

- A **fully reproducible dataset** including plenty of Steam games with all key features
- A portfolio of exploratory visualizations
- A clustering analysis of game genres and commercial outcomes
- A final written report and presentation summarizing insights

7. Tentative Timeline

Week	Task
Week 5	Data collection scripts (API + scraping), preliminary dataset
Week 6	Cleaning, wrangling, and exploratory visualizations
Week 7	Clustering analysis and predictive modeling
Week 8	Presentation, and finalization of the project

8. Task Division

- **Yuxun He**: Data acquisition through Steam API, data preprocessing, exploratory data analysis.
- **Tingjun Chen**: Web scraping for supplemental game metadata, feature engineering, and clustering analysis.

- **Boran Zhang**: Data cleaning and integration, visualization development, and preparation of the final analytical outputs and documentation.

We will collaborate closely throughout the project to ensure consistency across data processing, analysis, and interpretation. But the division of tasks may be adjusted as needed based on the actual progress and evolving requirements of the project.

Appendix: Fetch Game Details by R

The code block below demonstrates that we successfully retrieved a structured dataset of Steam games using the SteamSpy API.

```
library(httr2)
library(jsonlite)
library(dplyr)

# Demonstration: retrieve Steam game data successfully
resp <- request("https://steamspy.com/api.php?request=all&page=1") |>
  req_perform()

data_raw <- resp |> resp_body_json()
steam_data <- bind_rows(lapply(data_raw, as_tibble))

# inspect structure
glimpse(steam_data)
```

```
Rows: 1,000
Columns: 17
$ appid           <int> 9450, 921060, 490220, 1301720, 274520, 70000, 239200, ~
$ name            <chr> "Warhammer 40,000: Dawn of War - Soulstorm", "Modern C~
$ developer       <chr> "Relic Entertainment", "Gameloft", "Lunarch Studios", ~
$ publisher       <chr> "SEGA", "Gameloft", "Lunarch Studios", "Bitbeast Games~
$ score_rank      <chr> "", "", "", "", "", "", "", "", "", "", "", "", "", ""~
$ positive        <int> 16764, 3130, 970, 9872, 20529, 12659, 7886, 13456, 154~
$ negative        <int> 847, 2595, 217, 2233, 1089, 2391, 3457, 2672, 3149, 23~
$ userscore       <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~
$ owners          <chr> "1,000,000 .. 2,000,000", "1,000,000 .. 2,000,000", "1~
$ average_forever <int> 2212, 269, 472, 86, 464, 241, 147, 673, 528, 39, 1064,~
$ average_2weeks  <int> 230, 0, 0, 0, 3, 0, 0, 0, 222, 0, 388, 0, 522, 74, 0, ~
```

```
$ median_forever  <int> 597, 48, 432, 76, 132, 114, 45, 409, 48, 46, 941, 316,~
$ median_2weeks   <int> 267, 0, 0, 0, 3, 0, 0, 0, 319, 0, 311, 0, 618, 106, 0,~
$ price           <chr> "0", "0", "0", "0", "449", "999", "1999", "3999", "0",~
$ initialprice    <chr> "0", "0", "0", "0", "1499", "999", "1999", "3999", "0"~
$ discount        <chr> "0", "0", "0", "0", "70", "0", "0", "0", "0", "0", "0"~
$ ccu             <int> 673, 1, 6, 63, 86, 3, 10, 145, 993, 0, 4144, 21, 744, ~
```

To supplement the API data, we also verified that web scraping from the Steam Store front-end is fully functional.

```r
library(rvest)
library(dplyr)
library(stringr)
library(purrr)
library(tidyr)

# Web Scraping Function for single game
scrape_steam_game <- function(appid) {
  url <- paste0("https://store.steampowered.com/app/", appid)

  page <- tryCatch(read_html(url), error = function(e) return(NULL))
  if (is.null(page)) return(NULL)

  game_name <- page %>%
    html_node(".apphub_AppName") %>%
    html_text(trim = TRUE)

  overall_review <- page %>%
    html_node(".game_review_summary") %>%
    html_text(trim = TRUE)

  review_block <- page %>% html_nodes(".user_reviews_summary_row")

  recent_review <- review_block %>%
    .[2] %>%
    html_node(".game_review_summary") %>%
    html_text(trim = TRUE)

  release_date <- page %>%
    html_node(".release_date .date") %>%
    html_text(trim = TRUE)
```

```
  price <- page %>%
    html_node(".game_purchase_price") %>%
    html_text(trim = TRUE)

  if (is.na(price) || price == "") {
    price <- page %>%
      html_node(".discount_final_price") %>%
      html_text(trim = TRUE)
  }

  tags <- page %>%
    html_nodes(".app_tag") %>%
    html_text(trim = TRUE) %>%
    str_squish() %>%
    paste(collapse = ", ")

  dev <- page %>%
    html_nodes(".dev_row") %>%
    html_nodes("a") %>%
    html_text(trim = TRUE)

  developer <- dev[1] %||% NA
  publisher <- dev[2] %||% NA

  tibble(
    appid = appid,
    game_name = game_name,
    overall_review = overall_review,
    recent_review = recent_review,
    price = price,
    release_date = release_date,
    tags = tags,
    developer = developer,
    publisher = publisher
  )
}

`%||%` <- function(x, y) if (!is.null(x) && length(x) > 0) x else y

scrape_steam_game(570)
```

```
# A tibble: 1 x 9
```

```
  appid game_name overall_review recent_review price        release_date tags
  <dbl> <chr>     <chr>          <chr>         <chr>        <chr>        <chr>
1   570 Dota 2    Very Positive  Very Positive Free To Play Jul 9, 2013  Free t~
# i 2 more variables: developer <chr>, publisher <chr>
```