**Data Collection Process**

The data for this project was gathered from [Gamalytic](https://gamalytic.com), focusing on the top 150 games ranked by revenue between January 1, 2024, and September 15, 2024 (in international date format).

After identifying the game names, I created individual folders for each game using Python. Each game contains 11 CSV files representing different aspects of game data, including:

* **Overview**
* **Copies Sold**
* **Revenue**
* **Top Seller Rank**
* **CCU (Concurrent Users)**
* **Price**
* **Review**
* **Followers**
* **Average Playtime**
* **Outstanding Wishlist**
* **Positive Reviews**

Out of these, only the "overview" file came pre-named, while the other 10 files were labeled as "historical data." I manually renamed and sorted all files—one by one—for each of the 150 games, resulting in 1650 correctly named and organized files over the course of four days.

**01\_main\_df\_and\_foldering notebook file:**

This code processes and organizes raw data for the 150 best Steam games. It begins by loading multiple CSV files into a unified DataFrame, then cleans game names by replacing Roman numerals with Arabic numbers and removing any non-alphanumeric characters. After cleaning, the data is sorted alphabetically by game name, and folders are automatically created for each game in a specified directory. The code also converts the release dates into a more readable format and finally saves the cleaned data to a new CSV file for future use.

**Cell 1: Importing Libraries**

In the first cell, we’re importing the necessary libraries.

pandas is our main tool for handling the dataset—it allows us to easily manipulate and analyze the CSV files.

os is there because we’ll need to interact with the operating system, especially when we create folders for each game.

re is used for working with regular expressions, which helps when we clean up the game names (like replacing Roman numerals).

glob will help us find all the CSV files in a directory automatically, so we don’t need to specify each file manually.

time isn’t really used yet, but it could be helpful if we need to measure performance later on.

**Cell 2: Loading and Merging CSV Files**

Here, we're grabbing all the CSV files from a specific folder (the one containing the data about the top 150 games). Instead of loading one file at a time, we use glob to fetch them all. We load each file into a list of DataFrames and then combine those DataFrames into one large DataFrame using pd.concat().

This gives us a unified DataFrame with all the game data across the files, making it easier to process everything together.

**Cell 3: Cleaning Game Names**

Next, we’re tackling the game name cleaning. The main issue is that some game names contain Roman numerals or special characters (like symbols or punctuation), which we don’t want. So, we defined a function that:

Replaces Roman numerals with standard Arabic numbers (e.g., "II" becomes "2").

Removes special characters that aren’t alphanumeric or spaces (this cleans up things like hearts, stars, or other symbols).

Removes extra spaces that might exist after cleaning.

After defining this function, we apply it to the game name column, and the result is a new column called cleaned\_name that has the cleaned-up versions of the game names.

**Cell 4: Dropping and Renaming Columns**

Now that we have a cleaned\_name column, we no longer need the original name column. So, we drop it.

Next, we move the cleaned\_name column to the front of the DataFrame (to make it the first column), and we rename it back to just name—this makes it look like the original, but now it's cleaned up. This will be useful for creating the folder names in the next step.

**Cell 5: Sorting and Creating Folders**

Here, we're sorting the DataFrame by the cleaned game names (the new name column). Once sorted alphabetically, we’re ready to create folders for each game.

We define the base directory where the folders should go, and then we loop through each game in the name column. For each game, we create a folder with the game's name. The use of os.makedirs() ensures that the folder is created if it doesn’t already exist.

At the end, we print a message to confirm that all the folders have been created successfully.

**Cell 6: Converting the Release Date**

This step converts the firstReleaseDate column from a timestamp (in milliseconds) to a readable date format like YYYY-MM-DD. Right now, the date is stored as milliseconds, which isn’t user-friendly.

We convert it into a format that only shows the year, month, and day (removing the time, hours, or any extra details). After this conversion, the firstReleaseDate will be more intuitive and easy to read.

**Cell 7: Saving the Cleaned Data**

Finally, we save the cleaned DataFrame to a new CSV file. This ensures that all the work we’ve done so far—like cleaning game names, reorganizing columns, sorting data, and formatting release dates—is preserved. Now, we have a cleaned version of the dataset that can be used for further analysis or sharing.

This breakdown covers how we’re preparing the game data, cleaning it up, organizing it into folders, and ensuring everything is saved in a usable format. We’re structuring the data to make it easier to work with in the next steps.

**02\_testing\_files.ipynb:**

This project is designed to handle raw data related to game performance from a library of folders containing various CSV files. The project involves checking for missing files, organizing data, transforming it, and saving the results. The key operations are data validation, filling missing values, and converting data formats for further analysis.

**Dependencies**

To run this project, the following Python libraries are required:

pandas: for data manipulation and analysis

os: for interacting with the operating system (managing directories and files)

re: for using regular expressions

glob: for file matching and pathname pattern expansion

time: for handling time-related operations

openpyxl: for writing data to Excel files (install using pip install openpyxl)

**Data Files**

The project processes several key CSV files within multiple folders in a library, with each folder representing data for a different game. The main folder (../data/raw/game lib) contains subfolders for each game, and these subfolders contain the following files:

overview.csv

copies sold.csv

revenue.csv

top seller rank.csv

CCU.csv

price.csv

review.csv

followers.csv

average playtime.csv

outstanding wishlist.csv

positive reviews.csv

Some files may be missing or mistyped, and this script helps identify and report such issues.

**Code Breakdown**

**Step 1: Loading CSV Files**

The script loads several CSV files from the raw data directory, including overview.csv, copies sold.csv, revenue.csv, and others, into corresponding DataFrames using pandas.read\_csv().

**Step 2: Emptying DataFrames**

Except for df\_price, all other DataFrames are cleared using the iloc[0:0] method to keep the structure intact but remove all data.

**Step 3: Converting and Expanding Dates in df\_price**

Convert the x column (milliseconds) to a proper date format.

Create a date range from 01.01.2024 to 09.15.2024 using pd.date\_range().

Merge this date range with df\_price to fill any missing dates.

Forward-fill missing prices using the fillna() method.

**Step 4: Rotating and Cleaning Data**

The df\_price DataFrame is rotated (transposed) to match the format of df\_copies\_sold. It’s then cleaned by removing unnecessary columns and resetting the index.

**Step 5: Handling Missing Data and Saving**

Missing values are filled with 0.

The cleaned DataFrame is saved as price\_0.csv.

Step 6: Checking for Missing Files

The script scans each folder in the game library (../data/raw/game lib) and checks whether all required files are present. If any files are missing, the folder name and missing files are recorded.

The result is saved as an Excel file (missing\_files\_report\_2.xlsx), listing the folder name and the missing files for each game folder.

**Step 7: Running the Code Twice**

To ensure accuracy, the script was run twice. The first run generated an initial missing files report, and the second run generated missing\_files\_report\_2.xlsx to verify if any files were still missing after corrections.

**Output Files**

price\_0.csv: Cleaned df\_price DataFrame with filled missing values and rotated structure.

missing\_files\_report.xlsx: An Excel report listing folders and missing files in the game library.

missing\_files\_report\_2.xlsx: Ensuring there are no missing files left.

**Very Importan update:**

After this point I've encountered a huge problem. Github does not allow me to upload all of my project files. So I decid to shrink my project from 150 best revenue Steam games to 75 best revenue Steam game.

This script is designed to shrink the number of game folders in the main library to the top 75 games based on revenue. It first selects the top 76 games from df\_main to ensure 75 valid folders are retained, considering one folder was excluded due to non-English characters in its name. The script then checks each folder in the main library and compares the folder names to the list of top 75 games. If a folder does not match any of the top 75 game names, it is deleted from the directory. This ensures only the most profitable games remain in the library, while others are removed.

**Importan note 2:**

Fixed and set same format as priceed games 'price' files, same date format but price column is 0.

**Cell 1: Finding and Arranging overview.csv Files**

**Summary**: This cell collects and concatenates all overview.csv files from each game folder into a single massive dataframe.

1. **Imports and Folder Setup**:
   * Imports the necessary libraries, os for file handling and pandas for data manipulation. The main\_folder is set to the directory containing game folders.
2. **Creating an Empty List**:
   * Initializes an empty list, df\_list, to store dataframes created from individual overview.csv files.
3. **Loop Through Game Folders**:
   * Iterates through each game folder in the main directory and constructs the path to overview.csv for each game.
4. **Check if overview.csv Exists**:
   * For each game, checks if the overview.csv file exists. If it does, the file is read into a dataframe and appended to df\_list.
5. **Concatenate DataFrames**:
   * Combines all dataframes in df\_list into one large dataframe, df\_mass\_game, holding the overview data of all games.

**Cell 2: Extracting and Organizing Unique Tags**

**Summary**: This cell extracts the tags column, collects all unique tags from every game, and saves them in a text file.

1. **Initialize Empty Tag List**:
   * Creates an empty list tag\_list to store unique tags across all games.
2. **Iterate Over DataFrame**:
   * Loops through each row in the dataframe and splits the tags column by commas to extract individual tags.
3. **Add Only Unique Tags**:
   * Strips excess spaces from each tag and adds it to tag\_list only if it hasn’t been added before, ensuring no duplicates.
4. **Convert List to String**:
   * Once all tags are collected, converts the list into a string of comma-separated tags.
5. **Save Tags to Text File**:
   * Writes the resulting string of unique tags to a file named tags.txt for future use.

**Cell 3: Categorizing Tags and Adding Columns**

**Summary**: This cell categorizes the tags from the tags column into four new columns (tag game type, tag style, tag mechanics, and tag perspective), and removes the original tags column.

1. **Define Tag Categories**:
   * Predefines four tag categories (game type, style, mechanics, and perspective) using a dictionary. Each category contains a list of relevant tags that match its theme.
2. **Categorize Tags Function**:
   * Defines a function to categorize tags based on these predefined lists. Tags from the dataframe are split and matched with the category lists.
3. **Handle Empty Matches**:
   * Ensures that if no tag matches a particular category for a game, a dash '-' is added to that category to avoid missing values.
4. **Apply Function to DataFrame**:
   * The categorization function is applied to the tags column of df\_overview for each game. The results are stored in four new columns, one for each tag category.
5. **Drop Original Tags Column**:
   * After the new columns are created, the original tags column is dropped from the dataframe, simplifying the structure.

**Overall Summary:**

This code prepares a large game overview dataframe by aggregating individual overview.csv files, extracts and organizes unique game tags, and classifies these tags into four distinct categories for easier analysis.

**03\_processing\_data.ipynb:**

**Cell 1**:  
We first import the required data from CSV files related to a game called *Anomaly Agent*. These include various game statistics like copies sold, revenue, playtime, reviews, etc. Each CSV is loaded into a Pandas DataFrame.

**Cell 2**:  
We start processing the df\_price DataFrame, which contains price data in a slightly different format compared to other data files. We rename the columns to date and price, convert the date format, and then fill in missing dates by creating daily entries for the price.

* **Steps within Cell 2**:
  1. Rename columns x to date and y to price.
  2. Convert date from milliseconds to human-readable format.
  3. Ensure that for each day between two recorded price points, the price is copied forward to fill any gaps.
  4. Remove duplicate dates, keeping only the first occurrence, and drop the last row (to avoid any outlier or incomplete data).

**Cell 3-11**:  
Each of these cells follows a similar process to handle the other game-related files (copies sold, revenue, rank, etc.).

* **Common Steps**:
  1. Transpose the DataFrame, switching rows with columns to make dates as rows.
  2. Clean and reset the index to ensure that date becomes the main column.
  3. Drop unnecessary characters from the date column (first four characters).
  4. Rename the first data column to match the file’s content (e.g., copies sold, revenue, etc.).

**Cell 12**:  
This cell handles the processing of tags, which are pieces of metadata that describe game attributes (like genres, mechanics, perspectives, etc.). A function is used to assign each tag to a category, which is then split into four columns:

* tag game type
* tag style
* tag mechanics
* tag perspective

If no relevant tag is found for a category, a single '-' is assigned to represent missing data. After processing, the original 'tags' column is removed from the DataFrame.

**Cell 13**:  
This step focuses on cleaning up the df\_overview DataFrame and ensuring proper column alignment:

* Removing redundant columns like price and languages.
* Splitting the genres column into genre 1, genre 2, and genre 3.
* Reordering columns to make the DataFrame more organized, placing genre-related information next to developers.
* The df\_overview is repeated to match the length of the df\_price DataFrame so that it can be merged seamlessly later.

Finally, all cleaned and processed DataFrames are concatenated together, ensuring that the information from various sources (price, revenue, reviews, etc.) is aligned on the same date column, creating the final df\_game DataFrame with all relevant game information.

**04\_main\_function.ipynb**

**Game Data Processing**

This script processes game data from multiple CSV files stored in separate folders for each game. It combines these files into a single cohesive DataFrame and generates a summary of each game's data. The script ensures the integrity of the data by checking for required files, processing various columns, and adjusting formats when necessary. The final output consists of individual CSV files for each game and a comprehensive df\_steamgame.csv that contains data for all the games.

**Key Functions**

* **Folder and File Structure**: The script loops through the ../data/raw/game lib folder, identifying subfolders for each game. It expects each folder to contain the following required CSV files:
  + overview.csv
  + copies sold.csv
  + revenue.csv
  + top seller rank.csv
  + ccu.csv
  + price.csv
  + review.csv
  + followers.csv
  + average playtime.csv
  + outstanding wishlist.csv
  + positive reviews.csv
* **File Validation**: It checks that each game folder contains all required CSV files. If all files are present, the game data is processed.
* **Data Processing**: The script processes each of the CSV files into a consistent format, handling date and value columns, filling gaps in the price.csv when necessary. It also extracts and categorizes tags from the overview.csv, splitting them into predefined categories such as 'game type', 'style', 'mechanics', and 'perspective'.
* **df\_price Specific Handling**: The price.csv can have two different formats. In some cases, it contains only two columns (date and price), which require special processing to fill in missing dates. In other cases, it follows the same structure as the smaller data files (revenue, outstanding wishlist, positive reviews, etc.), making it easier to process similarly to other data frames.
* **Daily Changes**: New columns are added to reflect daily changes in copies sold, revenue, reviews, and outstanding wishlist.
* **Final Output**: After processing the data, individual gameinfo.csv files are saved for each game. All game data is then consolidated into a massive df\_steamgame.csv, which contains the combined data for all games.

**Summary Files**

* **game\_info\_summary.csv**: Contains basic metadata for each game, including the folder location and processed file paths.
* **df\_steamgame.csv**: A comprehensive CSV containing all processed game data.

**Price Data**

* **df\_price Processing**: The script handles two different formats of price data. In one format, the file contains just two columns for date and price, which requires interpolation to fill in missing dates. In the other format, it matches the structure of other small data frames, making it simpler to process. Both formats are handled appropriately to ensure consistency across the dataset.