

MXB362 Case Study Project Final

Investigation & development for an improved method for visualising video game success.

HARRY JAMES PATRICK WRIGHT (11085614)

Executive Summary

This investigation delivers a comprehensive assessment of the performance, stability, and aesthetic attributes of the 'Steam Store Success Visualisation'. It's genesis emerged from a desire to enhance the existing iterations of models like *Item 1* and *Item 2* (Appendix 2) from Steam. Inspiration has also been drawn from the research paper titled 'How to launch a success video game: A framework' (Ahmad, Barakji, Shadada, & Anabtawi, 2017). The fundamental design was shifted towards interactivity, allowing users to engage more deeply. The design process encountered issues, particularly in managing a significant dataset of over 70,000 entries and ensuring optimal load times. Technical issues relating to data types and structures necessitated category reductions/adjustments to maintain performance and usability.

This has resulted in a stable customer focused visualisation. Despite eliminating several categories, the project effectively achieved the desired goals, underpinned by the visualisation principles of simplicity and accessibility. Notably if the user is unaware of the vastness of the data they're manipulating, it's a show of the method's simplicity. The minimal reliance on text, save for obligatory items, affirms its accessibility.

While the current visualisation has achieved these goals, there is room for further enhancement. The potential reintroduction of 'platform' categories could offer some users more comprehensive insight. Any further iteration will benefit immensely from staging a more comprehensive investigation of the data-type and structure of the categories in the dataset.

Data Sources

The dataset underpinning the ‘Steam Store Success Visualisation’ originates, from Steam, the largest gaming platform on PC. This restriction to only Steam as a platform is to maintain data integrity. Valve maintains consistent, strict policies regarding refunds, requiring a user to make the request before two weeks of purchase and with no more than two hours (120 minutes) playtime.

The “Steam Games Dataset” (Bustos, 2022) described is the result of a data-scraping program ‘Steam-Games-Scraper’, which pulls from an API provided by Steam. The dataset is primarily a mixture of quantitative and qualitative Split across 39 columns and 70,000+ rows, making data visualisation easier. The small ‘overview’ presented in *Table 1*, *Table 2* & *Table 3* of Appendix 1. It is important to note that not all categories hold equal relevance, necessitating more discerning selections for the preferred data visualisation outputs.

In the context of data visualisation, not all of the dataset is worth utilising either, as seen in *Table 4*. As a more niche example, there are categories which effectively achieve the function of another category, as seen in *Table 2*, which has parallels within Steam Categories (See *Table 3*) that indicate whether a game has achievements or not. Users only need to know whether the game has achievements, rather than how many. There are now seventeen (previously 22) categories/columns in the dataset which provide practical data for the purposes of an interactive visualisation. Additionally, six categories serve as ‘success’ metric, the variable which the names of video games can be ranked against.

Visualisation Techniques

The inherently quantitative and qualitative nature of the “Steam Games Dataset” lends itself to relatively simple visualisation techniques. While the dataset’s size presents challenges, focus on the user-centric design principles ensure a balance between simplicity and access. “Simplicity” in the context of the elements and the design serving immediately apparent purposes for the user to grasp. “Accessibility” in the context that no/few invisible - not explained- rules halt a user’s ability to manipulate the interactive visualisation. Serving both visualisation concepts is made more tactile by making elements consistent and using colour to highlight/‘code’ items of interest.

For these reasons, when ranking the ‘top’ games or if a graph is necessary, the ‘bar’ style graph is used. Bar graphs, exemplified in *Item 1* (Appendix 3), have been employed for their efficacy in representing cumulative data. Concurrently, dropdown menus, illustrated in *Item 3*, offer a pragmatic solution for representing multi-option categories, where a standard graph may be too large or obscuring. Colour coding will be used sparingly to enhance user comprehension, and colour selection will be made with the colour blind in mind. Blue is preferred in this regard, as it is not only a scientifically proven calming colour, but registers as the rarest form of achromatopsia.

Visualisation Environments and Tools

While design serves as the cornerstone of proficient visualisation -and is subject to the necessary techniques applied- its actualization necessitates coding proficiency. Producing an interactive visualisation is possible on most data-manipulation languages, such as R or MATLAB. It is also possible to produce the desired 'Steam Store Success Visualisation' within these languages, it is significantly easier within Python.

Raw Python is not an effective data manipulation or visualisation language, it can equal the capacities for manipulating 2D datasets present in R through modules. Within this project, the external packages utilised were *pandas* (2.1.0), *plotly* (5.16.1) and *dash* (2.13.0). Pandas provides Python the ability to load and process datasets from raw files, and enables surface level data massaging techniques, such as dropping duplicates or ensuring uniqueness within categories. Plotly and dash allow appropriate data to be visualised, presented and interacted with in the Python environment. Dash additionally allows HTML design features (such as divs and gridding) to be utilised for structuring and labelling purposes.

The development of the 'Steam Store Success Visualisation' followed a general theme of adding more and more of the previously 22 applicable categories. Throughout the development phase, iterative refinements were made, reducing the category count from the initial 22 to a more streamlined 17. Beyond mere redundancy, this pruning was also influenced by computational constraints associated with the dash module. Specifically within dash -the primary structuring & interface module- there exists an unchangeable 30 second time for initialising all elements in the visualisation. Because the process must run on other machines, removing categories were considered necessary.

This decision has not been considered a loss. The process of eliminating categories inadvertently birthed an innovative (novel) concept. Previously -before the progress report- the intention had been to represent all 6 success criteria simultaneously across multiple graphs. Specifics for how the concept has been implemented concurrently will be discussed -when applicable- in the following segment.

Results and Outputs

Throughout the development of the 'Steam Store Success Visualisation', the emphasis has consistently been on refining an existing model. Understanding the causes behind established success model's popularity has been a subject of contention during research. The intention behind this project wasn't just to emulate, but innovate and potentially replace existing models. Multiple iterations of these visualisations have been investigated, *Item 1* and *Item 2* (Appendix 2) represent standard success models from Steam. These models, although useful, often present a uniform view of success, mostly focusing on singular numeric metrics. The occasional lack of graphical representation or missing vital elements like axis values emphasise the need for a revamp.

This drive for innovation led to a 'How to launch a successful video game: A framework' (Ahmad, Barakji, Shadada, & Anabtawi, 2017). This paper advocated a more nuanced approach, emphasising both internal and external game factors. The difference between their proposed model (see *Item 3* Appendix 2) and existing ones is the breadth of factors considered and their representation, in addition to the respective audience. While conventional models do adjust based on the changing metrics, they often fail to accommodate diverse, fluctuating factors inherent to the gaming industry. This ultimately makes them more valuable to a customer-based audience -what this project focuses on-, rather than a developer oriented solution.

Based on these investigations, the finalised visualisation was envisioned as an interactive platform. This design would not only echo the nature of video games, but also empower users to craft their unique success criteria. These factors would be tied to a dataset which presents internal data relating directly to individual video games and will never change.

To bring the vision to life, data from a CSV file was manipulated within a Python environment. Data massaging & manipulation carried within the project's concurrent iterations is limited because of the selected dataset's predominantly categorical & numeric data structure, though it is conducted. An example is documented in *Code Segment 1* (Appendix 4) and represents the data manipulation of the 'Age' category, which is converted and visualised as the 'Maturity Rating' category.

The general algorithm for the program starts with the importation of libraries & the dataset from its csv format into a dataframe. General data massaging and basic data manipulation is then conducted, a second example being 'Positive' and 'Negative' category's conversion from two numbers to a 'Positive Review Percentage' is seen in *Code Segment 2*, Appendix 4). After this stage, (applicable) data is compiled as a HTML element through dash, containing the dropdown and a label for descriptive purposes and given a . There are two exceptions, one in 'Supported languages', which is instead manipulated as a plotly graph. This distinction is necessary, as the graph serves as a trigger for initialising an enabling 'Top Games Chart' to update according to the various other factors selected. The second exception, and arguably this visualisation's most novel is the 'Success Metric' dropdown. Integrating dash elements (dropdowns and labels) as objects, and the necessity to remove elements for improved performance inspired using a dropdown to enable multiple 'success metrics' within a single element. Prior to this discovery, 'Success Metric' would have operated as several 'Top Games' bar plots. Enabling the user to manipulate additional metric provides additional functionality to the design. Following on from there, after a user has selected the mandatory categories, the 'Top Games Chart' is given an update call. This code is visible in *Code Segment 4* (more in *Code Segment 5*) and shows how the filtered data will shift to compare against the new 'Success Metric'.

This general algorithm results in the finalised ‘Steam Store Success Visualisation’. The default output of running the program within Visual Studio (Code) on a ‘Python File’ debug configuration (see *Item 8* Appendix 3) is visible under Appendix 3 as *Item 5*. While a user may interact with the various categories from this page, only one will initiate the ‘Top Games Chart’, the ‘Language Counts’ graph. This requirement is made more obvious by the blue ‘alert’ text beneath the success metric & above the area the ‘Top Games Chart’ is present, in addition to plotly adding hover-over information by default. It should be noted that the colour blue has been used to signify importance within the visualisation, which distinguishes these items even more. A decision made more prominent by the presence of only four colours throughout the whole visualisation, white, black, blue and grey. This managed automatically by plotly (blue is the default colour for plots) and by passing ‘style’ objects when generating dropdowns.

After a language is selected, the ‘Top Games Chart’ will enable and become visible. Depending on the number of categories changed before doing so may affect the length of time to generate it. An example of a generated output, with various categories changed is visible as *Item 6*. The names of these ‘Top Games’ may shift from directly beneath to at an angle as seen in the ‘Language Counts’ graph to fit these results. As more categories, or more obscure options are toggled, it is entirely possible for the dataset to exhaust available options. Normally, this would produce an error from pandas within dash, but a simple check will instead disable the ‘Top Games Chart’ and display an alert such as *Item 7*. This is an attempt by the visualisation’s to remain accessible and ensure users understand the error without disrupting the entire process.

The process of generating the final visualisation has provided me with a wealth of knowledge. Programming the solution within python was not the original intention, but became necessary soon after coding commenced. Rstudio & MATLAB simply did not provide enough functional (and easily writable) interactivity within their environments to suit the project’s scope. While Python does not naturally support 2D datasets, installing modules is enough to provide this functionality to the chosen environment. The pivot from Rstudio and MATLAB to Python, while not intended, was a revelation. Python’s flexibility and extensibility, complemented by modules, enables far more creative problem solving.

Retrospectively, this project has been severed as a reminder of thoughtful tool and platform selection. While Python proved invaluable here, it's essential to remain open to alternatives in the future. The project also highlighted the balance needed between functionality and optimization. Going forward, a more pragmatic approach will guide future design decisions, ensuring both efficiency and user satisfaction.

Exegesis of the Effectiveness of Visualisation Outputs

Truly effective visualisation is the melding of simplicity and accessibility seamlessly. Reflecting on the objectives highlighted in “Results and Outputs” and the items presented in

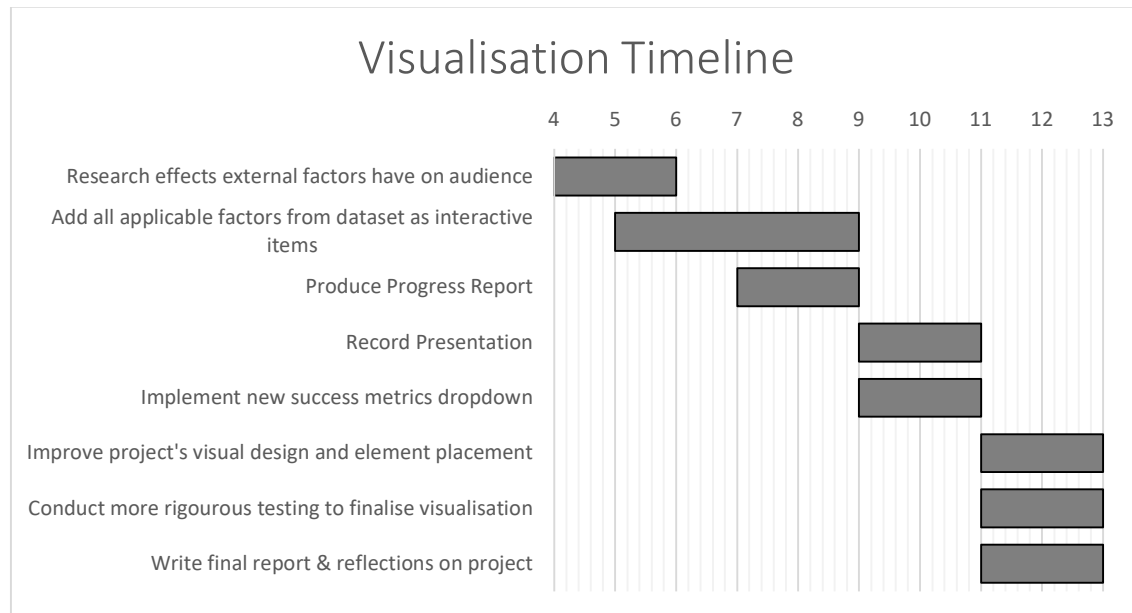
“Visualisation Techniques”, the implementation of these techniques stands as an achievement of the ‘Steam Store Success Visualisation’. To be more specific:

Simplicity: The essence of simplicity lies in directness and intuitive understanding. Within visual representation, this translates as clear, unambiguous graphics which convey an intended message without overwhelming the user. Where every element serves a purpose without redundancy. The ‘Steam Store Success Visualisation’ has been tailored to this principle at its very core. This is evident in *Items 5, 6, & 7*. These representations are devoid of clutter and present all elements within a single (2560 x 1440) screen. The consistent design, coupled with certain visual cues, guide the user’s attention to critical data points. Furthermore, if the user remains oblivious to the mammoth task of manipulating over 70,000 data entries, then the design’s simplicity has succeeded.

Accessibility: Beyond simplicity, a visualisation must be accessible. This doesn’t only concern the physical or technological requirements, but also involves the user’s own experiences. An accessible design is one where the user can engage without feeling restricted by invisible rules. The design should invite exploration and interaction. The final visualisation has become an antithesis of this purpose. The few existing rules, like the prerequisite language selection, are articulated clearly, leaving no room for ambiguity. Text should be reserved for genuine necessity, and the visualisation’s reliance on instructions for only critical points, such as when a user has either not made a language selection or has overly specific options, stands testament to its user-friendly approach.

The ‘Steam Store Successes Visualisation’ stands as a testament to the combination of an simple and accessible design. It not only meets the visual and interactive requirements of the modern user, but also respects their patience. Every decision, every iteration, and every design choice has been guided by these principles, resulting in a visualisation which is both aesthetically pleasing and functionally robust.

Updated Project Timeline



Problems

Throughout the development of any interactive visualisation, encountering challenges is something to be anticipated. However they should also be welcomed, as they can deeper understandings and inspire more robust solutions. The 'Steam Store Success Visualisation' has been no exception and presented its own fair share of obstacles.

Loading Times: During the course of this report, the persistent 30-second loading time introduced by Dash has been a recurring issue. The decision to resolve this issue by eliminating redundant or glitch-prone elements proved effective. The categories 'Windows', 'Linux' and 'Mac' initially offered Boolean options for platform compatibility, posed particularly frustrating challenges. Their inclusion introduced inexplicable errors, such as disabling the 'Top Games Chart' under specific conditions. The exact cause of the problem remains unknown, but there are suspicions towards the 'reduce' function, as detailed in *Code Segment 6* (Appendix 4). Instead of extending more time attempting to produce the final visualisation with these categories included, their removal enabled faster loading times & allowed for a better element positioning.

Data Type Amomalise: The other prevalent issue arose around data type inconsistencies in the now defunct categories. It's still unclear, but certain categories that appeared as numerical values in Excel underwent a transformation to object types once imported into Python through pandas. While it is likely that the failure lies with the project's code, it cannot be truly diagnosed, especially given that other numeric categories converted using similar processes to the problematic categories. The only other consistency with these categories are that they projected data which has since been considered redundant, such as

‘achievements count’ and ‘dlc count’. In practical terms, users are often more concerned with the presence of achievements or DLCs, rather than their exact quantities.

In summary, creating the ‘Steam Store Success Visualisation’ has been dotted with challenges. Each one has served as a learning opportunity to steer the project towards more refined solutions and, ultimately a more polished product. Any further development will undoubtedly benefit from these experiences.

Project Journal

- 8/08/23: Attended meeting with lecturer to discuss proposal.
- 14/08/23: Switched to a larger and more updated dataset (from ~27,000 to 70,000+)
- 15/08/23: Attended another lecture to finalize proposal.
- 28/08/2023: Through research on the different program options to produce the final figure, Matlab was found to not be open enough to add the level of interactivity necessary for the final product. Decided to look into various programming languages to see which would better accommodate the decision to move from Matlab.
- 29/08/2023: Out of the programming languages I’m competent in, python was decided upon for the final program, for its simplicity. C# was considered, but rejected on the grounds that an object-oriented design wasn’t necessary.
- 13/09/2023: Actively started to program the project. Established an initial system where the Peak CCU of various game titles would be ranked in descending order based on the language selected. Said languages were selectable from a histogram, which counted the number of times each language appeared in the dataframe.
- 14/09/2023: Out of the 39 columns in the csv file, 16 were considered either non-applicable for the project or too similar to other columns. Of the 24 remaining, 6 were considered to be ‘success’ metrics and a part of a new model concept for the project. Work started on adding the remaining 18 options for a user to manipulate as they saw fit. By the end of the day, 4 (Price, Categories, Genres, Tags) more items had been added as dropdowns which a user could manipulate, meaning 5 of the 18 had been implemented.
- 15/09/2023: Implemented five additional categories (Developers, Publishers, Windows, Mac, Linux). Had to update the code for Genres and Tags to be more similar to Categories.
- 16/09/2023: Implemented six additional categories (Required-age, DLC-count, Positive, Negative, Achievements, Recommendations). A consistent error has started appearing with the platforms (Windows, Mac, Linux) which causes them to not function properly. In that if Windows is selected, Mac must also be selected (switched to false/true). Additionally, DLC-count, Achievements and Recommendations have minor issues with converting string to integer values, there are plans to resolve this after the progress report.
- 17/09/2023: Added Labels and basic HTML divides to give the display significantly more clarity. Shrunk the maximum number of languages on the Languages Supported plot to 50, thereby making it significantly easier to navigate.
- 19/09/2023: Started Progress Report.
- 21/09/2023: Finished and submitted Progress Report.
- 23/09/2023: Implemented 4 of the 5 remaining success criteria. Fixed a recently discovered problem with the categories ‘Publishers’, ‘Developers’, ‘Categories’, ‘Genres’ and ‘Tags’,

which previously applied an OR check for the selected values, rather than AND. Currently running into issue where program takes longer than 30 seconds to load, current solution is to remove longer processes such as 'Maturity Rating' temporarily.

- 30/09/2023: Work started on project presentation. Draft of script written with 1127 words. No noteworthy progress made on fixing currently known issues or implementing the final success criteria yet.
- 9/10/2023: Submitted Project presentation.
- 10/10/2023: While the issue of loading times are still present, the previous 'temporary' approach of removing elements to improve loading times has been turned into a boon. 'No. of achievements' and 'No. of DLC's' are both made redundant because of the fact there are tags and genres which amount to a simpler way of portraying them, rather than selecting a specific number of them. Their removal has enabled 'Maturity Rating' to return. Also added a little text warning which reminds users that a language from the 'Language count' chart must be selected, otherwise the 'Top games' chart cannot be triggered.
- 11/10/2023: Merged the categories 'Positive' and 'Negative' into the category 'Percentage of Positive Reviews'. It is far more intuitive than having thousands of selectable numbers which typically only responded to one or two different games. Additionally, a previously manipulable factor has now been made a success factor, No. of recommendations. The 30 second error is much rarer now but is not gone.
- 12/10/2023: Finally restructured the elements to fit everything into one screen without needing to scroll down. All twelve selectable categories (including success criteria) are now grouped into two columns of six. Success criteria has been given a blue highlight to focus attention onto it, whereas the other categories have been given a light grey outline. At this point, the actual visualisation/project is effectively complete.
- 13/10/2023: After conducting some tests with audience for feedback, noticed a consistent error with the three 'platform' conditions (windows, mac & linux), where if windows and/or mac were selected, the 'Top games chart' would not load properly. Made the decision to remove them for a few reasons. Firstly -and most importantly-, with the 30 second loading time still being prevalent, trimming down options to improve loading times for users is necessary, as they may not have access to a particularly powerful PC. Secondly, with their removal the layout of the categories enables the 'Success Criteria' to be on it's own column, making it even more distinct from the other categories, which is important considering it is effectively 6 categories in one. Thirdly, with the time limit for the final report approaching, the project ultimately needs to be finalised now.
- 19/10/2023: Project Report finalised.

Conclusions

The project's primary objective has always been clear: to elevate and improve the existing steam video game success models. The final iteration of the 'Steam Store Success Visualisation' not only embodies this desire, but has also taken significant strides in achieving it. By providing users with an interactive platform, it empowers them to customise their personalised metric of success.

The project's success lies in its stability and adherence to the set objectives. However, some of the project's functionality was sacrificed due to the removal of multiple categories, a decision borne out of necessity rather than design.

Retrospection provides focus. Had the problematic categories been identified earlier, the development process would have been smoothed considerably. A more meticulous examination of the data types and structures for each category within the project environment, especially considering their inconsistencies, would have been beneficial. It may have also mitigated the challenges endured during the implementation phase.

With hindsight, the project has room for expansion and enhancement. The reintroduction of the 'platform' categories, especially in the context of a PC-based data source, holds value. Their exclusion was less a design choice and more a constraint of time. Given ample resources, integrating these categories could further improve the visualisation, ensuring it remains comprehensive and relevant to a larger customer audience.

If the project is to be developed further, beyond its current iteration, the option to implement the three 'platform' categories is one option available. Considering the project is based on data from a PC platform, it would be beneficial to allow users to manipulate these categories. The only reason these were not implemented was because of a lack of time available, so additional resources would enable their implementation into the design.

In conclusion, while the 'Steam Store Success Visualisation' is a successful iteration of existing designs, it also serves as a reminder that every project provides insight. Both in the issues inevitably faced and the solutions to them. Any future endeavor will invariably benefit from these insights, ensuring a continuous improvement towards bettering these visualisations.

References

- Ahmad, N. B., Barakji, S. A., Shadada, T. M., & Anabtawi, Z. A. (2017). How to launch a successful video game: A framework. *Entertainment Computing*, 1-11.
- Bustos, M. (2022). *Steam Games Dataset*. Retrieved from Kaggle:
<https://www.kaggle.com/datasets/fronkongames/steam-games-dataset>
- Value. (2023, October 15). *Steam Charts & Stats (Live)*. Retrieved from Steambase:
<https://steambase.io/steam-charts>

Appendix 1

[illegible]

Table 1 – Overview of first 50 entries in dataset (presented in excel)

Achievements
30
12
0
0
17
0
62
0
25
32

Table 2 – Example of numeric data structure

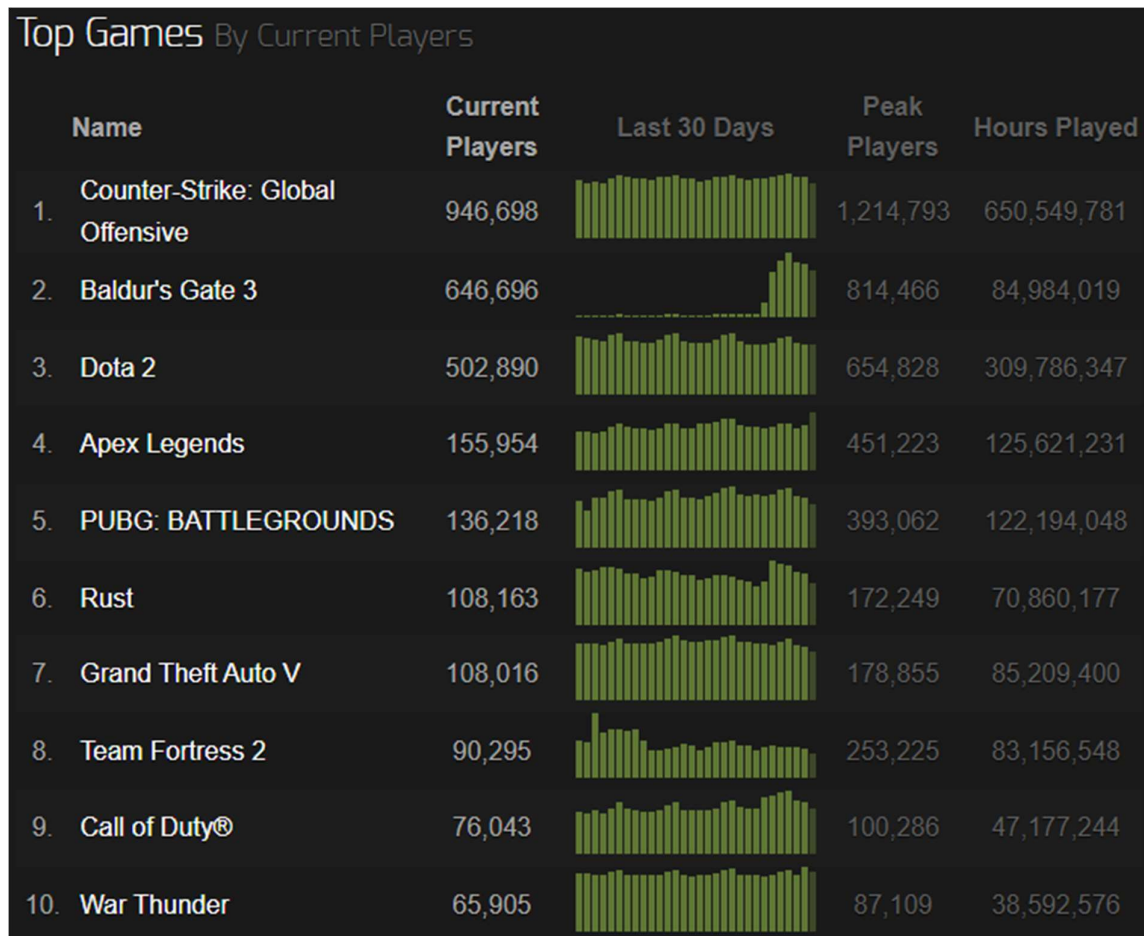
Categories	Supported languages
Single-player,Multi-player,Steam Achievements,Partial Controller Support	[English]
Single-player,Steam Achievements,Full controller support,Steam Leaderboards,Remote Play on Phone,Remote Play on Tablet,Remote Play on TV	[English, French, Italian, German, Spanish - Spain, Japanese, Portuguese - Brazil, Russian, Simplified Chinese, Traditional Chinese]
Single-player	[English, Portuguese - Brazil]
Single-player,Full controller support	[English, French, Italian, German, Spanish - Spain, Japanese, Korean, Portuguese, Russian, Simplified Chinese, Traditional Chinese]
Single-player,Steam Achievements	[English, Spanish - Spain]
Single-player,Multi-player,MMD,PoP,Online PoP,Co-op,Online Co-op,In-App Purchases	[English]
Single-player,Steam Achievements,Steam Cloud	[English, Russian, Danish]
Single-player,Steam Cloud	[English, German]
Single-player,Steam Achievements,Full controller support	[English, French, Italian, German, Spanish - Spain, Russian, Japanese, Simplified Chinese, Traditional Chinese, Korean]
Single-player,Steam Achievements,Steam Trading Cards,Partial Controller Support,Steam Cloud	[English, Polish, French, Italian, German, Spanish - Spain, Portuguese - Brazil, Russian, Japanese, Simplified Chinese, Traditional Chinese, Korean]

Table 3 – Examples of object structure present in dataset (presented in excel)

Movies
http://cdn.akamai.steamstatic.com/steam/apps/256863704/movie_max.mp4?t=1638854607
http://cdn.akamai.steamstatic.com/steam/apps/256691108/movie_max.mp4?t=1506089586
http://cdn.akamai.steamstatic.com/steam/apps/256847488/movie_max.mp4?t=1635980739 , http://cdn.akamai.steamstatic.com/steam/apps/256847487/movie_max.mp4?t=1635980747
http://cdn.akamai.steamstatic.com/steam/apps/256819153/movie_max.mp4?t=1611314333
http://cdn.akamai.steamstatic.com/steam/apps/256764430/movie_max.mp4?t=1580660973
http://cdn.akamai.steamstatic.com/steam/apps/256818274/movie_max.mp4?t=1611036715
http://cdn.akamai.steamstatic.com/steam/apps/256867570/movie_max.mp4?t=1641709460 , http://cdn.akamai.steamstatic.com/steam/apps/256843858/movie_max.mp4?t=1628006054
http://cdn.akamai.steamstatic.com/steam/apps/256881983/movie_max.mp4?t=1650185685
http://cdn.akamai.steamstatic.com/steam/apps/256776944/movie_max.mp4?t=1584591532
http://cdn.akamai.steamstatic.com/steam/apps/2035390/movie_max.mp4?t=1447364829

Table 4 – Example of poor visualisation data present in dataset (presented in excel)







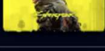

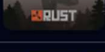

Appendix 2 – Existing Video Game Success Models/Frameworks



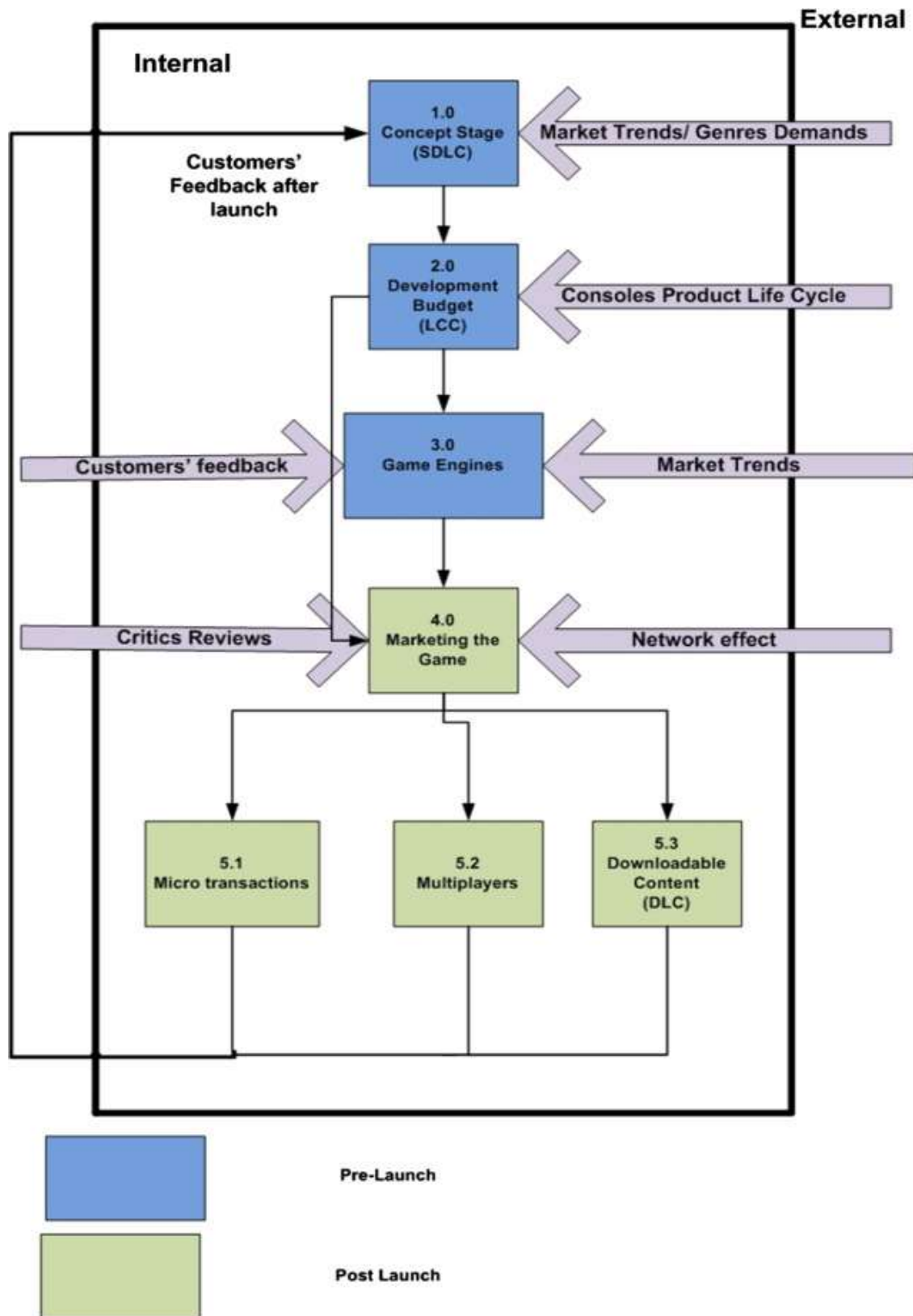
Item 1: Steam-charts 'Top Games Chart' (Value, 2023)

Most Played Steam Games

The top 100 **most played games** on Steam by number of players online and in-game. Game data and rankings are updated in real-time.

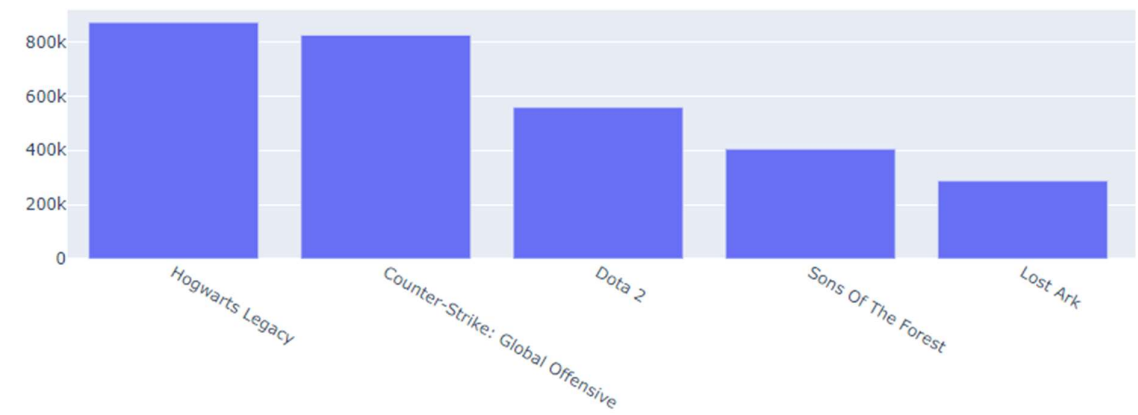
	Name	Current Players ↕	All-Time Peak ↕	Player Score ↕	
#1	 Counter-Strike 2	913,056	1,670,271	88 ↑	View
#2	 Dota 2	535,925	863,209	82 ↑	View
#3	 Baldur's Gate 3	239,895	873,941	95 ↑	View
#4	 Call of Duty®: Modern Warfare® II	170,426	189,557	59 ↑	View
#5	 Apex Legends™	141,807	454,151	79 ↑	View
#6	 PUBG: BATTLEGROUNDS	132,934	423,795	57 ↑	View
#7	 Cyberpunk 2077	122,139	274,526	80 ↑	View
#8	 Team Fortress 2	111,235	252,355	93 ↑	View
#9	 Rust	109,409	182,335	86 ↑	View
#10	 Grand Theft Auto V	99,651	204,660	86 ↑	View

Item 2: Steam-charts 'Top Games Chart' (Alternate Style) (Value, 2023)

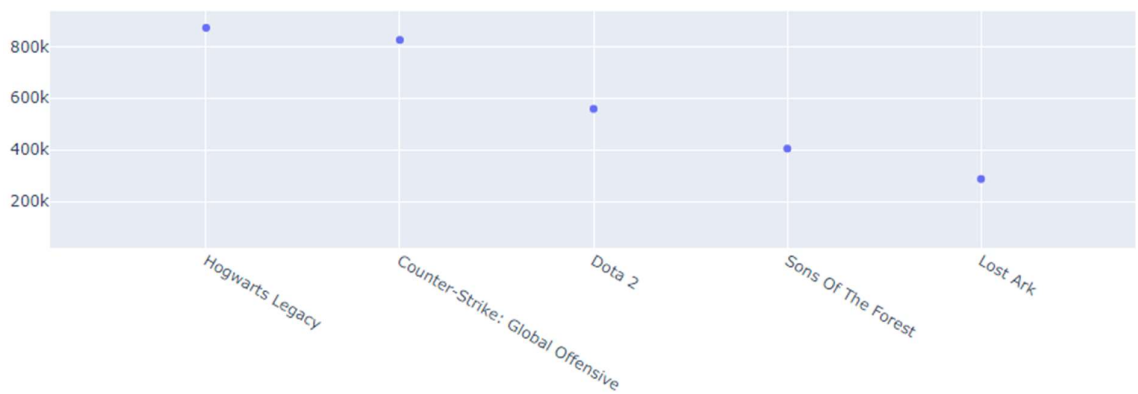


Item 3: Layered success framework w/ external & internal factors (Ahmad, Barakji, Shadada, & Anabtawi, 2017).

Appendix 3 – (Final) Visualisation Outputs



Item 1: (Reduced) Bar plot ‘Top Games Chart’



Item 2: (Reduced) Scatter plot ‘Top Games Chart’

Select Steam Categories

Select...

Single-player

Multi-player

Steam Achievements

Partial Controller Support

Item 3: “Steam Categories” Dropdown

Select Maturity Rating

Select...

Select Price Range

Select...

Select Percentage of Positive Reviews

Select...

Select Developer's

Select...

Select Success Metric

Select...

Select Steam Categories

Select...

Select Steam Genres

Select...

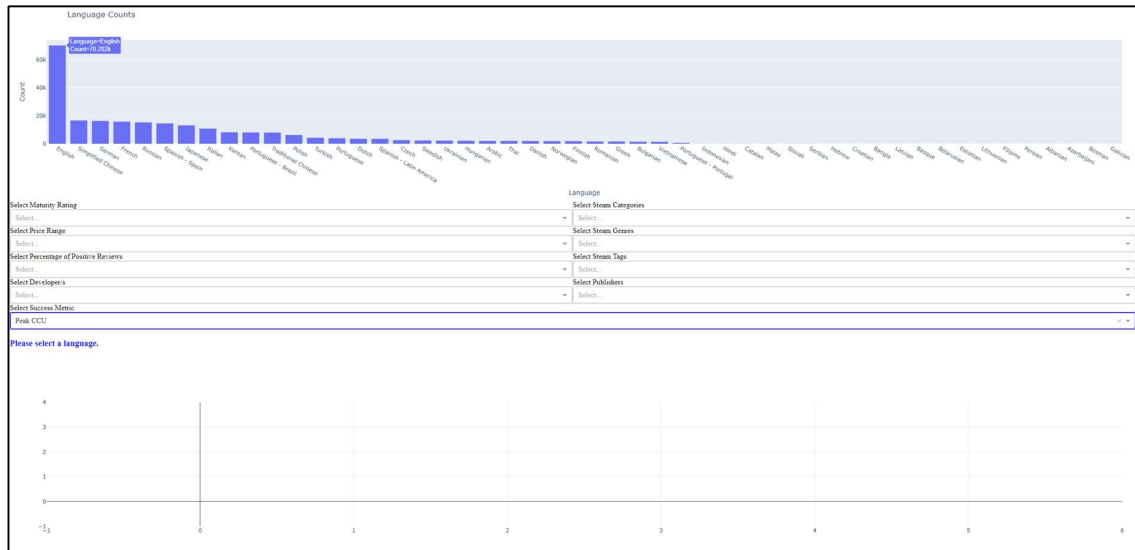
Select Steam Tags

Select...

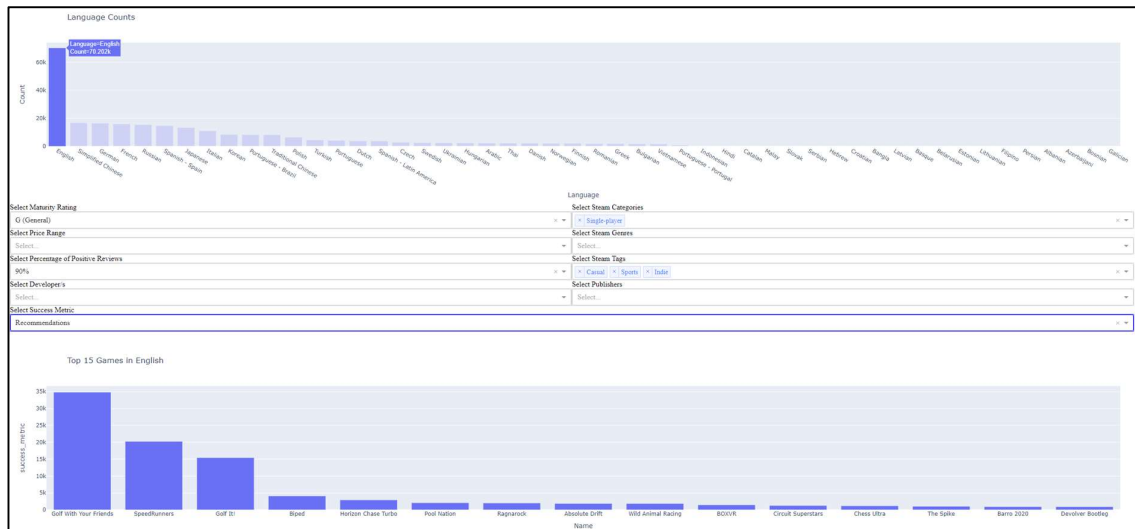
Select Publishers

Select...

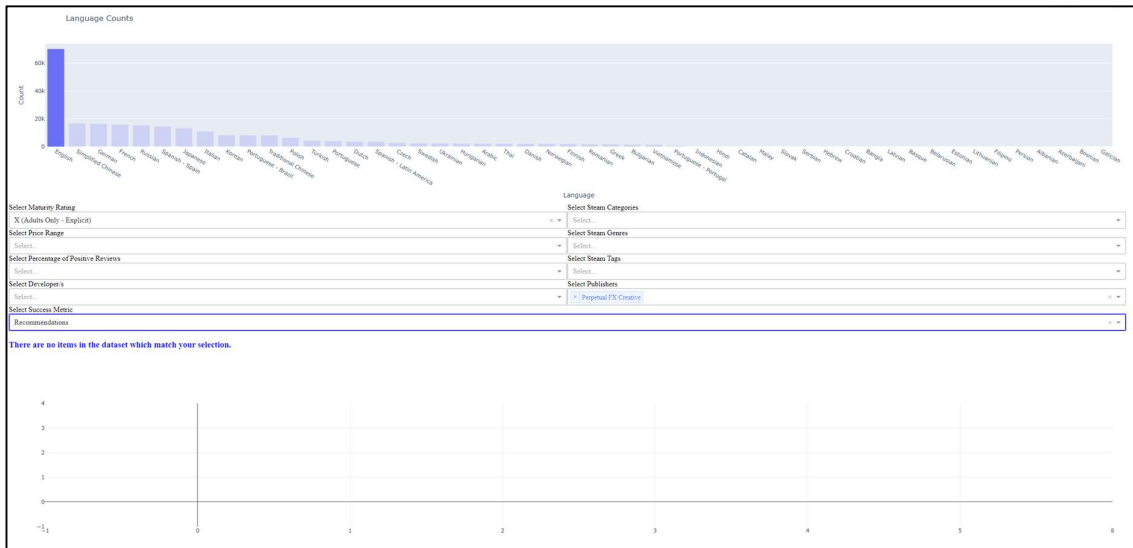
Item 4: Available Dropdown options



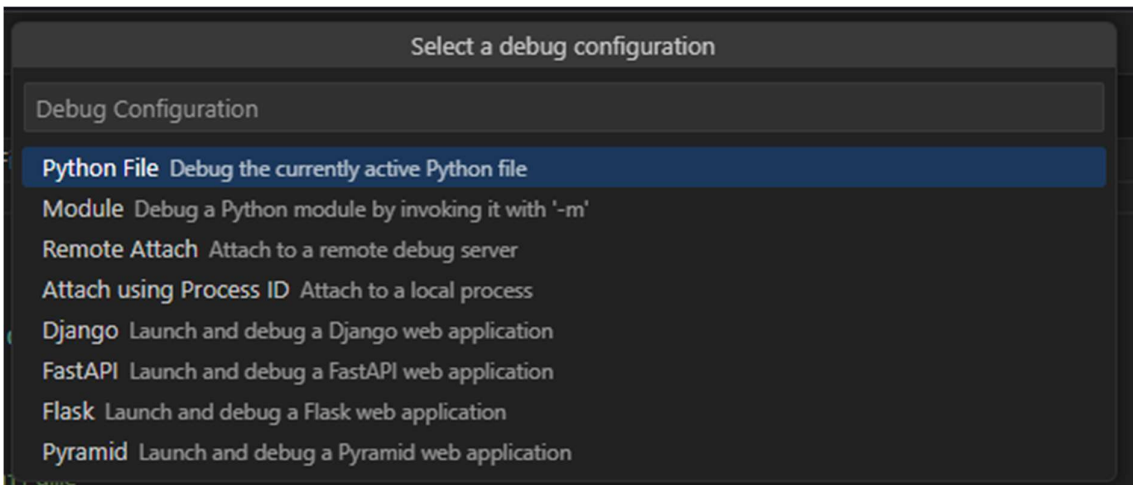
Item 5: Default Output (On 2560 x 1440 monitor)



Item 6: Output (W/ options selected) (On 2560 x 1440 monitor)



Item 7: Categories too specific (On 2560 x 1440 monitor)



Item 8: Debug configuration used for program.

Appendix 4 – Code Segments

```
# Define a function to map the age values to maturity ratings
def map_age_to_rating(age):
    if 0 <= age < 10:
        return 'G'
    elif 10 <= age < 13:
        return 'PG'
    elif 13 <= age < 15:
        return 'M'
    elif 15 <= age < 17:
        return 'MA'
    elif 17 <= age < 21:
        return 'R'
    else:
        return 'X'

# Apply the function to create a new column 'Maturity Rating'
steam_data['Maturity Rating'] = steam_data['Required
age'].apply(map_age_to_rating)

category_dropdown_style = {
    '#backgroundColor': 'white', # Background color
    'border': '1px solid lightgray', # Border style
    'color': 'black', # Text color
}

# Create a dropdown for selecting maturity ratings
maturity_rating_dropdown = html.Div([
    html.Label("Select Maturity Rating"),
    dcc.Dropdown(
        id='maturity-rating-dropdown',
        options=[
            {'label': 'G (General)', 'value': 'G'},
            {'label': 'PG (Parental Guidance)', 'value': 'PG'},
            {'label': 'M (Mature)', 'value': 'M'},
            {'label': 'MA (Mature 15+)', 'value': 'MA'},
            {'label': 'R (Adults Only)', 'value': 'R'},
            {'label': 'X (Adults Only - Explicit)', 'value': 'X'},
        ],
        multi=False, # Allow a single selection
        value=None, # Default to None
        style=category_dropdown_style
    )
])
```

Code Segment 1: ‘Age’ to ‘Maturity Rating’ conversion & dropdown compilation using dash, plotly, pandas and custom Python definitions.

```

category_dropdown_style = {
    #'backgroundColor': 'white', # Background color
    'border': '1px solid lightgray', # Border style
    'color': 'black', # Text color
}

# Calculate the merged reviews column (rounded to the nearest 10%)
steam_data['Merged Reviews'] = ((steam_data['Positive'] /
(steam_data['Positive'] + steam_data['Negative'])) * 10).round() * 10

# Create a dropdown for selecting the percentage of positive reviews
percentage_dropdown = html.Div([
    html.Label("Select Percentage of Positive Reviews"),
    dcc.Dropdown(
        id='percentage-dropdown',
        options=[
            {'label': f'{i}%', 'value': i}
            for i in range(10, 101, 10) # Generate options for 10%, 20%, ...,
100%
        ],
        multi=False,
        value=None, # Default to None
        style=category_dropdown_style
    )
])

```

Code Segment 2: 'Positive' & 'Negative' conversion to the 'Merged Reviews' category.

```

# Arrange dropdowns in two columns (excluding Success Metric)
dropdowns_column1 = html.Div([
    maturity_rating_dropdown,
    price_range_dropdown,
    percentage_dropdown,
    developers_dropdown,
], style={'width': '49%', 'display': 'inline-block'})

dropdowns_column2 = html.Div([
    categories_dropdown,
    genres_dropdown,
    tags_dropdown,
    publishers_dropdown,
], style={'width': '49%', 'display': 'inline-block'})

dropdowns_row = html.Div([
    success_metric_dropdown,
], style={'width': '98%', 'display': 'inline-block'})

# Define the layout with the language count chart and an empty top games chart
app.layout = html.Div([
    dcc.Graph(id='language-count-chart', figure=language_chart),
    dropdowns_column1,
    dropdowns_column2,
    dropdowns_row,
    alert_label,
    dcc.Graph(id='top-games-chart'),
])

```

Code Segment 3: Proper structuring of elements through dash html integration.

```

# Create a bar chart using Plotly Express for language counts
language_chart = px.bar(
    language_counts,
    x='Language',
    y='Count',
    title='Language Counts'
)

# Update chart layout for interactivity
language_chart.update_layout(
    clickmode='event+select',
    showlegend=False
)

@app.callback(
    [Output('top-games-chart', 'figure'),
     Output('alert_label', 'children')],
    [Input('language-count-chart', 'selectedData'),
     Input('maturity-rating-dropdown', 'value'),
     Input('price-range-dropdown', 'value'),
     Input('percentage-dropdown', 'value'),
     Input('developers-dropdown', 'value'),
     Input('publishers-dropdown', 'value'),
     Input('categories-dropdown', 'value'),
     Input('tags-dropdown', 'value'),
     Input('genres-dropdown', 'value'),
     Input('success-metric-dropdown', 'value')]
)
def update_top_games_chart(selectedData, selected_maturity_rating,
selected_price_range, selected_percentage, selected_developers,
selected_publishers, selected_categories, selected_genres, selected_tags,
selected_metric):

    if selectedData:
        # Extract the selected language
        selected_language = selectedData['points'][0]['x']
        print("Selected language:", selected_language)

        # Filter the DataFrame for the selected language
        filtered_df = steam_data[steam_data['Supported
languages'].str.contains(selected_language)]

        if selected_maturity_rating is not None:
            filtered_df = filtered_df[filtered_df['Maturity Rating'] ==
selected_maturity_rating]

        # Only apply the 'Price' filter if a price range is selected
        if selected_price_range is not None:

```

```

        filtered_df = filtered_df[filtered_df['Price'] ==
selected_price_range]

        if selected_percentage is not None:
            # Filter the DataFrame based on the selected percentage of positive
reviews
            min_positive_percentage = selected_percentage - 10 # Adjust the range
based on the selected percentage
            max_positive_percentage = selected_percentage

            filtered_df = filtered_df[(filtered_df['Merged Reviews'] >=
min_positive_percentage) & (filtered_df['Merged Reviews'] <=
max_positive_percentage)]
            # Filter the DataFrame based on the selected developers
            if selected_developers:
                developer_conditions = [] # List to store individual developer
conditions

                for selected_developer in selected_developers:
                    developer_conditions.append(
                        filtered_df['developers separated'].apply(
                            lambda devs: selected_developer in devs if
isinstance(devs, list) else False
                        )
                    )

                # Combine the individual developer conditions using the logical AND
operator
                if developer_conditions:
                    developer_filter = reduce(lambda x, y: x & y,
developer_conditions)
                    filtered_df = filtered_df[developer_filter]

            # Filter the DataFrame based on the selected publishers
            if selected_publishers:
                publisher_conditions = [] # List to store individual publisher
conditions

                for selected_publisher in selected_publishers:
                    publisher_conditions.append(
                        filtered_df['publishers separated'].apply(
                            lambda pubs: selected_publisher in pubs if
isinstance(pubs, list) else False
                        )
                    )

                # Combine the individual publisher conditions using the logical AND
operator

```



```

        if publisher_conditions:
            publisher_filter = reduce(lambda x, y: x & y,
publisher_conditions)
            filtered_df = filtered_df[publisher_filter]

        # Filter the DataFrame based on the selected categories
        if selected_categories:
            category_conditions = [] # List to store individual category
conditions

            for selected_category in selected_categories:
                category_conditions.append(
                    filtered_df['categories separated'].apply(
                        lambda cats: selected_category in cats if isinstance(cats,
list) else False
                    )
                )

            # Combine the individual category conditions using the logical AND
operator
            if category_conditions:
                category_filter = reduce(lambda x, y: x & y, category_conditions)
                filtered_df = filtered_df[category_filter]

        # Filter the DataFrame based on the selected Genres
        if selected_genres:
            genre_conditions = [] # List to store individual genre conditions

            for selected_genre in selected_genres:
                genre_conditions.append(
                    filtered_df['genres separated'].apply(
                        lambda gens: selected_genre in gens if isinstance(gens,
list) else False
                    )
                )

            # Combine the individual genre conditions using the logical AND
operator
            if genre_conditions:
                genre_filter = reduce(lambda x, y: x & y, genre_conditions)
                filtered_df = filtered_df[genre_filter]

        # Filter the DataFrame based on the selected Tags
        if selected_tags:
            # Use apply and lambda to handle lists in the 'tags separated' column
            filtered_df = filtered_df[filtered_df['tags separated'].apply(
                lambda tags: any(tag in selected_tags for tag in tags) if
isinstance(tags, list) else False

```

```

    )]

    # Update the 'success' metric column based on the selected metric
    if selected_metric == 'Peak CCU':
        filtered_df['success_metric'] = pd.to_numeric(filtered_df['Peak CCU'],
errors='coerce')
    elif selected_metric == 'Metacritic score':
        filtered_df['success_metric'] = pd.to_numeric(filtered_df['Metacritic
score'], errors='coerce')
    elif selected_metric == 'User score':
        filtered_df['success_metric'] = pd.to_numeric(filtered_df['User
score'], errors='coerce')
    elif selected_metric == 'Recommendations':
        filtered_df['success_metric'] =
pd.to_numeric(filtered_df['Recommendations'], errors='coerce')
    elif selected_metric == 'Average playtime forever':
        filtered_df['success_metric'] = pd.to_numeric(filtered_df['Average
playtime forever'], errors='coerce')
    else:
        filtered_df['success_metric'] = pd.to_numeric(filtered_df['Median
playtime forever'], errors='coerce')

    # Sort the filtered DataFrame by 'Peak CCU' in descending order
    top_15_games = filtered_df.sort_values(by='success_metric',
ascending=False).head(15)

    if top_15_games.empty:
        fig = {}
        alert_text = "There are no items in the dataset which match your
selection."
        return fig, alert_text
    elif selected_metric == "":
        fig = {}
        alert_text = "No 'Success Crtieria' has been selected."
        return fig, alert_text
    else:
        # Create a bar plot for the top 15 games
        fig = px.bar(
            top_15_games,
            x='Name',
            y='success_metric',
            title=f'Top 15 Games in {selected_language}',
            category_orders={"Name":
top_15_games.sort_values(by='success_metric', ascending=False)["Name"]} # Ensure
the x-axis order matches the DataFrame order
        )
        alert_text = ""
        return fig, alert_text

```

```
# If nothing is selected, return an empty figure, and the alert for the user  
to select a language  
return {}, "Please select a language."
```

Code Segment 4: 'Top Games Chart' updating method.

```

# Update the 'success' metric column based on the selected metric
    if selected_metric == 'Peak CCU':
        filtered_df['success_metric'] = pd.to_numeric(filtered_df['Peak CCU'],
errors='coerce')
    elif selected_metric == 'Metacritic score':
        filtered_df['success_metric'] = pd.to_numeric(filtered_df['Metacritic
score'], errors='coerce')
    elif selected_metric == 'User score':
        filtered_df['success_metric'] = pd.to_numeric(filtered_df['User
score'], errors='coerce')
    elif selected_metric == 'Recommendations':
        filtered_df['success_metric'] =
pd.to_numeric(filtered_df['Recommendations'], errors='coerce')
    elif selected_metric == 'Average playtime forever':
        filtered_df['success_metric'] = pd.to_numeric(filtered_df['Average
playtime forever'], errors='coerce')
    else:
        filtered_df['success_metric'] = pd.to_numeric(filtered_df['Median
playtime forever'], errors='coerce')

    # Sort the filtered DataFrame by 'Peak CCU' in descending order
    top_15_games = filtered_df.sort_values(by='success_metric',
ascending=False).head(15)
    if top_15_games.empty:
        fig = {}
        alert_text = "There are no items in the dataset which match your
selection."
        return fig, alert_text
    elif selected_metric == "":
        fig = {}
        alert_text = "No 'Success Crtieria' has been selected."
        return fig, alert_text
    else:
        # Create a bar plot for the top 15 games
        fig = px.bar(
            top_15_games,
            x='Name',
            y='success_metric',
            title=f'Top 15 Games in {selected_language}',
            category_orders={"Name":
top_15_games.sort_values(by='success_metric', ascending=False)["Name"]} # Ensure
the x-axis order matches the DataFrame order
        )
        alert_text = ""
        return fig, alert_text

```

Code Segment 5: Update “Success Metric” Category

```
# Filter the DataFrame based on the selected platforms
platform_conditions = []

if selected_windows is not None:
    platform_conditions.append(filtered_df['Windows'] ==
selected_windows)
if selected_mac is not None:
    platform_conditions.append(filtered_df['Mac'] == selected_mac)
if selected_linux is not None:
    platform_conditions.append(filtered_df['Linux'] == selected_linux)

if platform_conditions:
    # Use logical OR to combine platform conditions
    platform_filter = reduce(lambda x, y: x | y, platform_conditions)
    filtered_df = filtered_df[platform_filter]
```

Code Segment 6: Problematic Platform Code.