End-to-end analytics workflow sample for Steam data

Case submission for Data Analytics Engineer role

Baris Bilen
03.06.2025

1) ETL & pre-modelling

1.1) Data cleaning efforts

In consulting projects, before working on the data, it is crucial to first take a look at what we are provided by the client (ideally we should also show the data to them in the beginning to align). We should observe the data quality, summarize key inputs (row count, count of unique identifiers) and even do some simple visualizations that would reveal insights that are already there. Thus, ***[src/exploration_notebook.ipynb]*** is to explore the data first.

Please observe the notebook directly to see my findings regarding data quality, uniqueness, completeness, and missing information. In short, below is a summary of what I did in ***[src/ETL_and_enrich.py]*** after my observations in the notebook.

- assign column names to tables where they are missing
- uniqueness/distinct count checks on dimension and fact tables (otherwise the duplicate records will distort the big picture)
- remove or impute NaN entries in tables, recover the duplicate app records and remove some illogical entries
- column type conversions/recastings
- melt the genre and language tables (reason explained in the next slide)
- drop some columns that can easily be derived from the other columns (Finalprice field in price info is example) to make it more efficient
- aggregate top player counts as hourly granularity is quite high for analysis purposes, then concatenate bottom player counts and top player counts

With ***[src/ETL_and_enrich.py]*** I processed the raw files and wrote the cleaned data to my local as .csv, then uploaded them to BigQuery.

2) Building a data model

- **Requirements**: First thing to consider when modelling our data is what we expect to do with this data, as our expectations will drive requirements. For this case, I came up with a data model that will;
    - support both dashboarding & reporting and machine learning/prediction modelling pipelines
    - ensure referential integrity by enforcing a foreign key based on appid field in app_info table
    - enable user to analyze and filter fact table on a single view stored within the data platform, reducing the need for unnecessary joins and querying on business user end

- **Data modelling approach:** I have used STAR schema as we have a fact table surrounded by various dimension tables, it provides faster aggregation and filtering as joins are simplified, and is easy to interpret by different stakeholders.

2) Building a data model

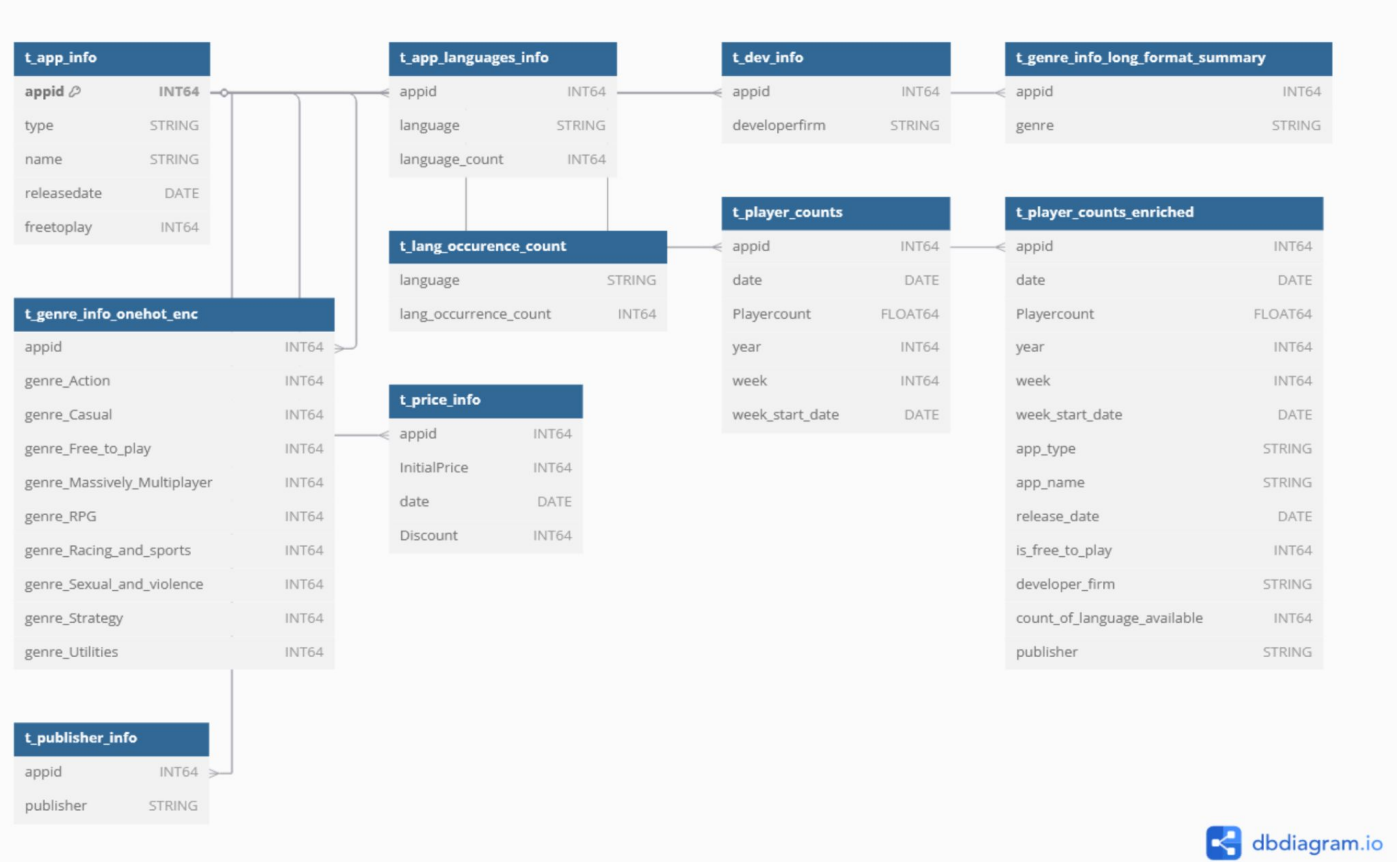
1. Dimension tables

- t_app_info: Contains key metadata for each app: ID, name, type, release date, free-to-play flag
- t_dev_info, t_publisher_info — link apps to publishers and developers
- t_app_languages_info — app-specific languages
  t_lang_occurence_count — language popularity across apps
- Genre Representation:
  t_genre_info_long_format_summary — long format (for filtering, grouping)
  t_genre_info_onehot_enc — one-hot encoded format (binary variables to represent genres, for features of ML mode)
- t_price_info: price and promotion information per app per date

2. Fact table(s)

- t_player_counts: Daily player count per app
- t_player_counts_enriched: Pre-joined view (on BigQuery) with metadata for efficient analysis

2) Building a data model

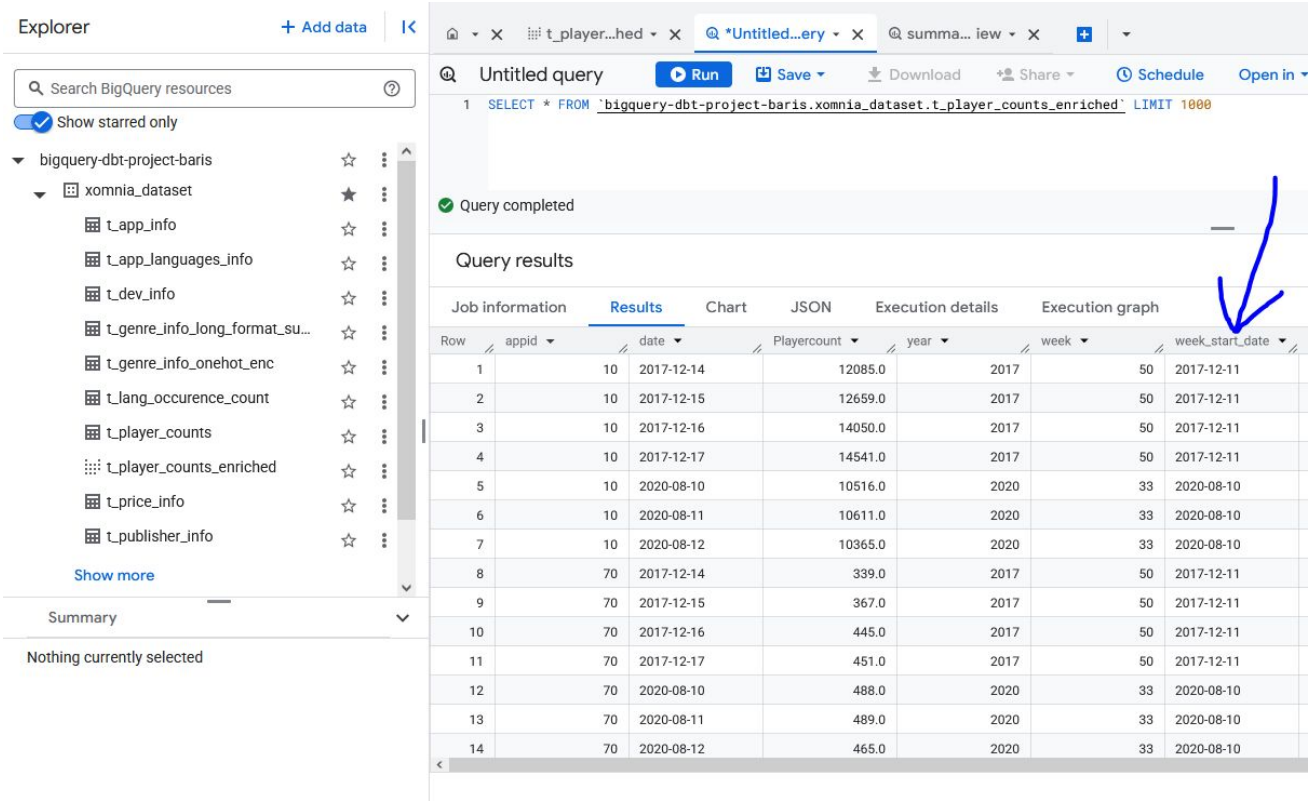


1) All appid's in all tables should be contained in t_app_info (referential integrity)

2) t_player_counts_enriched is a view on BigQuery that is enriched version of our main fact table

View creation query can be found in **create_bigquery_view.sql.** Notice that all tables are left joined but t_app_info is INNER JOINed in order to ensure that in our view we don't have any record for an app that does not exist in t_app_info

2) Building a data model

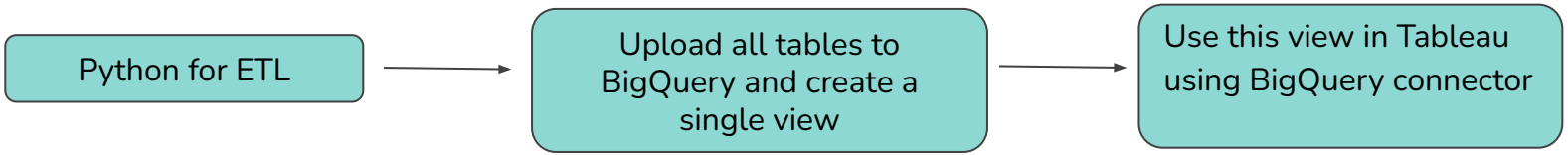


Row	appid	date	Playercount	year	week	week_start_date
1	10	2017-12-14	12085.0	2017	50	2017-12-11
2	10	2017-12-15	12659.0	2017	50	2017-12-11
3	10	2017-12-16	14050.0	2017	50	2017-12-11
4	10	2017-12-17	14541.0	2017	50	2017-12-11
5	10	2020-08-10	10516.0	2020	33	2020-08-10
6	10	2020-08-11	10611.0	2020	33	2020-08-10
7	10	2020-08-12	10365.0	2020	33	2020-08-10
8	70	2017-12-14	339.0	2017	50	2017-12-11
9	70	2017-12-15	367.0	2017	50	2017-12-11
10	70	2017-12-16	445.0	2017	50	2017-12-11
11	70	2017-12-17	451.0	2017	50	2017-12-11
12	70	2020-08-10	488.0	2020	33	2020-08-10
13	70	2020-08-11	489.0	2020	33	2020-08-10
14	70	2020-08-12	465.0	2020	33	2020-08-10

I have also created a “week_start_date” column to identify that record’s week, to be able to aggregate the to week level when desired

3) Dashboarding

Overall this is the pipeline I built for reporting:



Why did I upload all the tables to BigQuery separately and did the joins there, instead of joining them beforehand in Python or joining them within the Tableau report with setting up the relations between them? With this way,

→ scalability for large fact tables (joins within Tableau for big tables take long)

→ maintainability (tables stay modular and reusable, otherwise need to run Python scripts every time to renew the final data)

→ better Tableau performance (especially with extract, so that the user can play around the report more easily)

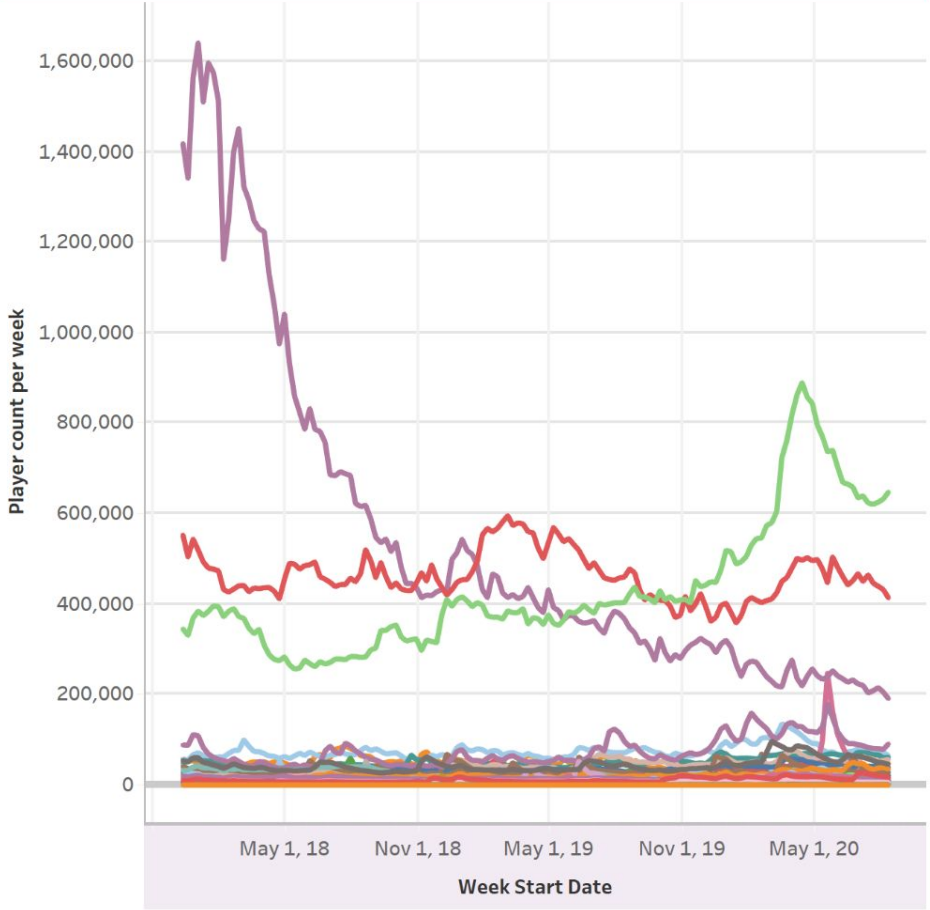
3) Dashboarding

I have created 4 worksheets (piece of dashboards) in Tableau:

1) **Player Count per week per game:** shows average player counts per game per week (day level is too granular for this report so I decided to use week as my time dimension). This dashboard can be filtered only basically any applicable information such as game, developer firm, publisher, free_to_play indicator, etc. Once a filter is applied, the plot shows only the game contained in that subset.
2) **Quarterly Average Player Count:** very simple dashboard that shows the change in average player count per quarter. This dashboard is more sort of a high level indicator of the trends in player counts. We can see a clear decline over years.
3) **Game count per language:** Again a simple dashboard that shows game count per language. We see that most games are offered in English already.
4) **Top played games:** Very important dashboard that shows the top 5 performers in terms of average player count over 3 years.
5) **Games player per week:** Distinct count of games that has at least one player who played it

⇒ Please find a screenshot from my Tableau report in the next slide. **The report is also available in my submission and its data is "Extract", meaning that its data is attached to the report and it does not require any connection to read the data from any external source (this way the report is also faster)**

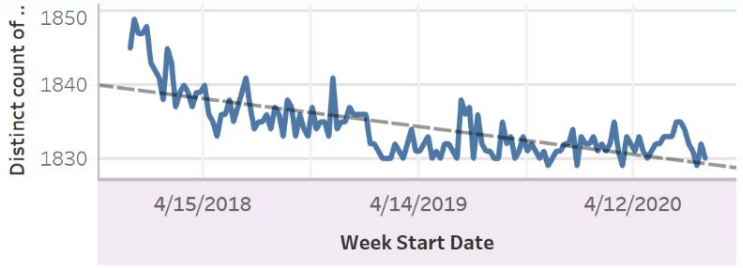
Player Count per week per game



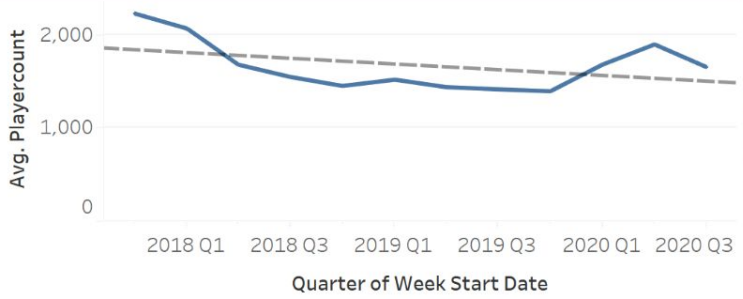
Top played games

name	
PLAYERUNKNOWN'S BATTLEGROUNDS	539,414
Dota 2	463,419
Counter-Strike: Global Offensive	430,713
Tom Clancy's Rainbow Six Siege	74,860
Grand Theft Auto V	74,843

Games played per week



Quarterly Average Player Count



is_promotion
(All)

Developer Firm
(All)

Is Free To Play
(All)

Year of Release Date
(All)

App Name
(All)

App Type
(All)

Publisher
(All)

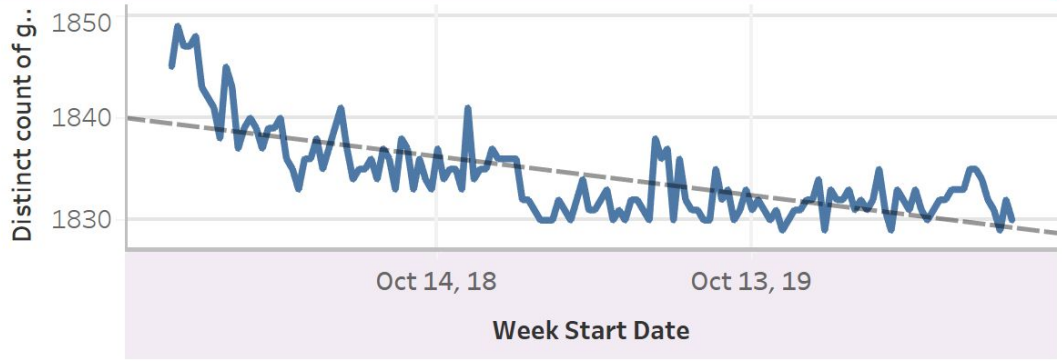
App Name
3DMark Demo
7 Days to Die
9Dragons
12 is Better ..
20XX
60 Seconds!
90 Minute Fe..
100% Orang..

is_played
(All)
0
1

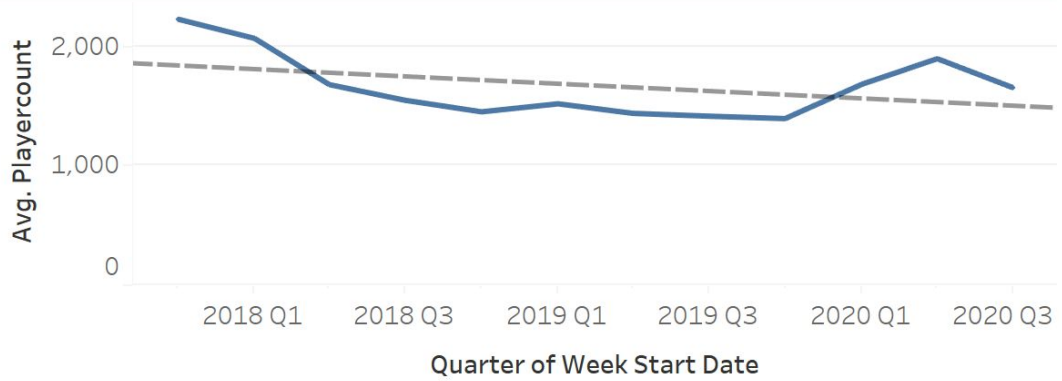
4) Insights

1) There is a clear seasonality in player counts, and each game's seasonality is different. This information will be useful when planning for promotion periods.
2) Top 3 games with the highest average player count over three years are PUBG, DOTA 2 and Counter Strike, averaging 539k, 463k, 430k weekly players respectively.
3) Most games are available in English. After that, the most common languages in the platform are German, French and Spanish, where a bit more than half of the games is offered in these 3 languages.
4) I have created a calculated field called is_played. This takes a value of 1 if a game's player count is more than zero on that day and 0 otherwise. When I plot the distinct count of games where is_player = 1 over weeks, we see that the # of games played is decreasing over years (from 1846 to 1830 levels).
5) I have created a worksheet called "Quarterly Average Player Count" that gives high level overview of average player count per quarter. As one can observe the trend line, this value is also decreasing. Combining this with #4 above, we can say that Steam might need to improve their retention a bit, to prevent business from shrinking. See next slide for visual.
6) It's impossible not to notice significant upward trend for Counter Strike: Global Offensive and downward trend for PUBG. See slide 12.

Games played per week



Quarterly Average Player Count



Player Count per week per game

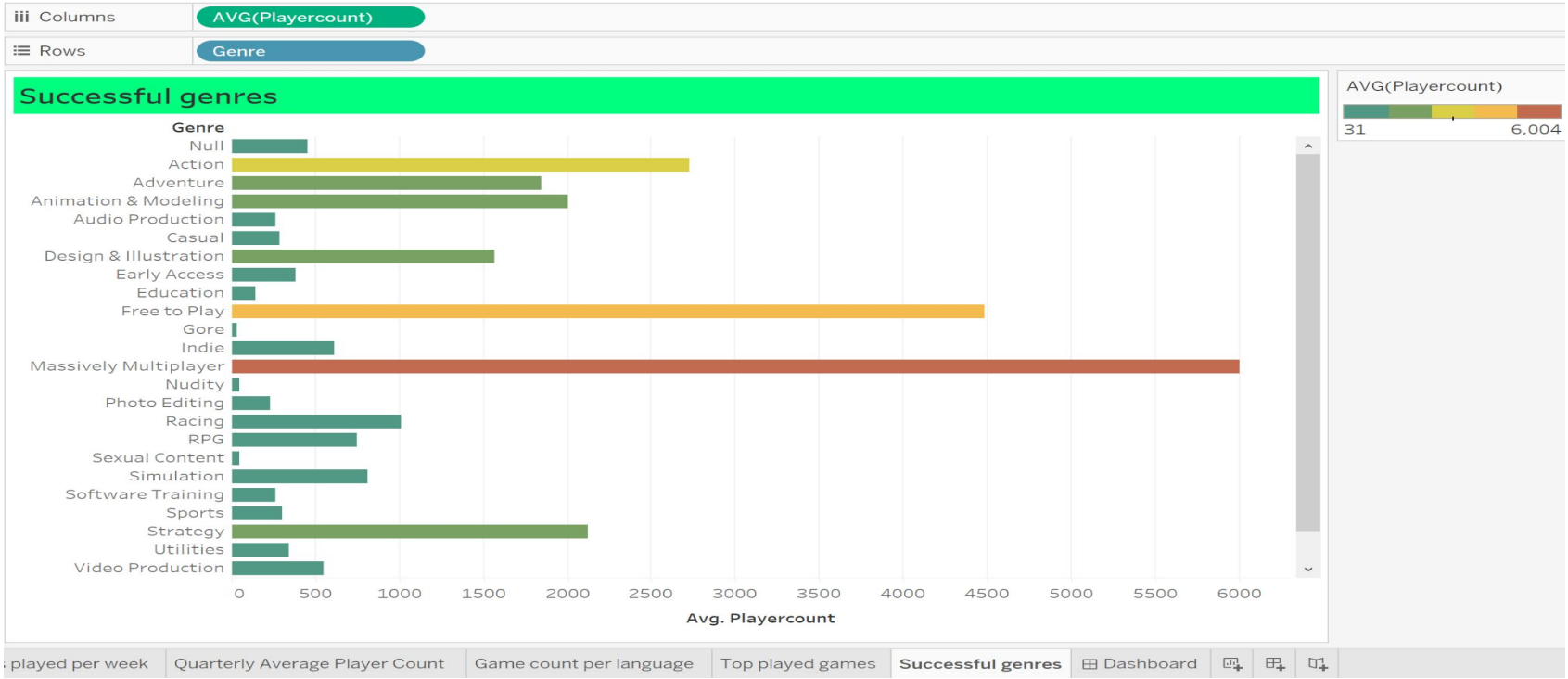
App Name

(Multiple values)

App Name

Counter-Strike: Global ..

PLAYERUNKNOWN'S BA..



7) The genres whose average player counts are the highest are **action**, **massively multiplayer**, **strategy.** When we also look at the genres of top 3 most successful games, which are PUBG, CS:GO and DOTA 2, we see that these three also have the aforementioned genres. There is clearly a positive relation between these genres and a game's success.

5) Bonus section

What type of data and analysis would you recommend Valve to undertake after the creation of your dashboard? What more would you do if you had more time?

A) Leveraging predictive modelling

1) I'd recommend trying simple machine learning models to understand the relation between the attributes of a game and it's player count. Available language count, publisher firm, year/week information (for seasonality) seems to be important variables driving the player count.

2) For #1, I'd recommend employ outlier detection on the fact tables (player_counts) retrieved from the source. For reporting purposes (presenting what has happened) I did not want to find and exclude the outliers, however in predictive modelling these outliers, if there are any, will distort and reduce the accuracy of the model's predictive power. Data engineers/analysts can assign a "is_usable_ML" flag to each record, where outliers will be labeled as 0, and reporting pipes will use all data while ML pipes will use only the "is_usable_ML"=1 ones.

B) Pipelining, increasing data quality and query performance (for product management and future analyses)

1) Storing and enforcing all schemas and column types using a config file (such as schema_config.yml)
2) Partition the fact tables using year-week columns or (date), to have faster queries (or put INDEX to date field if the data is inserted to a db)
3) Write the intermediate (silver) tables in Parquet instead of csv, and use PySpark for ETL if the data gets bigger at some point
4) Employ unit testing on top of linting
5) Employ more logical checks on data (such as checking a if a play date is before than releasedate (I checked this in my notebook)
6) Put the codes to a remote repo and employ version controlling like I did, and schedule the codes on Airflow for automation and reproducibility of the analyses

5) Bonus section

What type of data and analysis would you recommend Valve to undertake after the creation of your dashboard? What more would you do if you had more time?

C) Bring more useful data

Player counts are definitely valuable piece of information but I think, for any business, the most important info is revenue or profit. If possible, I'd work on a table such as "Purchases" that has the purchase information and shows the price of the game at the time of purchase. This way we can find the games which brings us the most and least revenue.

→ Ultimately, we can see what attributes are common in top-performer games and what attributes of them differentiate from the games with least revenue to understand what brings financial success to a game and focus on creating games with such attributes.

D) Running correlation analysis to understand the effect of promotions

Currently our data is sufficient for this but unfortunately I don't have more time to work on this. Ideally one can see that applying a promotion will increase the player count for a short period, and Valve can use promotions to attract more players and generate more revenue out of games. Probably it's not a good idea to apply discounts right after a game is launched.

6) Tech stack

- Programming languages
  - **Python** (*initial data exploration, ETL (cleaning and normalization), and enrichment in local environment)*
  - **SQL** (*querying and creating views for reports)*
- Ensuring code quality/linting
  - **flake8 & autoflake** *(PEP-8 compliance check and reformatting)*
- Version control
  - **Git & GitHub** *(version controlling and code repository)*
- Compute tools
  - **Local environment** *(running Python codes inside a virtualenv)*
  - **BigQuery Sandbox environment** *(data-platforming, querying, and reporting purposes)*
- Reporting tools
  - **Tableau Desktop** *(dashboarding)*
- Other
  - dbdiagram.io *(visualizing my data model)*
  - Jupyter notebooks *(initial exploration)*

Thank you