Scalable Databases

Data analysis on Steam library

Midterm Project (Group 5)

By

Somil H. Saparia (U01876683)

Kunj D. Patel (U01869538)

Yash D. Bhavsar(U01873220)

Seidenberg School of Computer and Information Systems, Pace University

Table of Contents

Problem Definition

This analysis's objective is to learn more about the video games that are offered on the Steam platform. We specifically want to investigate the distribution of games according to age restrictions and look at the connections between other characteristics like price, ratings, and its categories. By comprehending these patterns, we can spot trends and assist customers, publishers, and game creators in making decisions.

Introduction

One of the top digital distribution channels for PC gaming is Steam, which is a great illustration of this situation. Steam requires a strong and scalable database system to handle the enormous amount of data it generates due to its massive game library and millions of active users.

In this project, we'll explore the difficulties of creating a recommended scalable database system for a gaming platform like Steam. We will look into the essential components that such a system needs, patterns, we can spot trends and assist customers, publishers, and game creators in making decisions. We will also consider and assess the various database technologies accessible databases, and graph databases, for this application.

This project's overall objective is to provide a thorough study of the database requirements and drift for a massively multiplayer online game like Steam and to recommend a viable database design that can handle the needs of millions of players and data-intensive transactions.

Libraries used in the project.

NumPy- The Python library NumPy is used to manipulate arrays. Moreover, it has matrices, Fourier transform, and functions for working in the area of linear algebra. "Numerical Python" is the term. It offers effective multi-dimensional array and matrix implementations, as well as operations on these arrays. Applications involving scientific computing, data analysis, and machine learning frequently employ NumPy.

Panda- Pandas is a popular open-source Python library for data manipulation and analysis. For processing structured data, it offers simple data structures and data analysis tools. Pandas is frequently employed in data science and analytical projects and is especially beneficial when handling tabular data.
Series and Data Frame are the two primary data structures offered by Pandas. A Series is a universally compatible one-dimensional labeled array. Similar to a spreadsheet or a SQL table, a Data Frame is a two-dimensional labeled data structure that can hold several Series. Moreover, Pandas has strong tools for data alignment, merging and joining, reshaping, grouping, filtering, and aggregation.

Matplotlib- A Python data visualization library is called Matplotlib. One of the most popular libraries for Python's interactive, animated, and static visualizations. With the help of Matplotlib's extensive collection of plotting functions, users may make line plots, scatter plots, bar plots, histograms, 3D graphs, and a variety of other visualizations.

Seaborn- A Matplotlib-based Python data visualization library is called Seaborn. It provides a complex user interface for producing interesting and useful statistical visuals. Seaborn is based on Matplotlib and works well with Pandas and NumPy, two additional Python data analysis packages. The ability of Seaborn to build sophisticated visualizations with little coding work is one of its primary advantages. It offers lots of customizability choices, like color schemes, themes, and plot styles, which make it simple to produce excellent visualizations. Moreover, Seaborn comes with several built-in datasets for practicing statistical analysis and data visualization.

.

Cv2- A software library for computer vision and machine learning is called OpenCV (Open-Source Computer Vision Library). It is a well-known library for computer vision tasks and is frequently used in processes for images and videos, object detection, facial recognition, and other related things.
The cv2 module makes available Python bindings for the library, which is written in C++. The OpenCV library functions can be accessed by Python programmers via the cv2 module. The cv2 module is a robust tool for creating image and video processing applications since it offers a large range of functions for computer vision and machine learning tasks in Python.

URL lib- A Python package for working with URLs is called URL lib (Uniform Resource Locators). It has several modules that let you carry out a range of operations on URLs, such as parsing URLs and retrieving data from web sites. Overall, URL lib is a strong Python module for working with URLs and has several features that are helpful for web development, web scraping, and other web-related tasks.

Essential key steps to demonstrated in our Python Notebook-Loading data into Data Frames. Integration of SQL and Python

```
# Importing Libraries

import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import sqlite3
import cv2
from google.colab.patches import cv2_imshow
import urllib.request
```

This imports the libraries matplotlib with the alias plt, NumPy with the alias np, and pandas with the alias pd, which are frequently used to make plots and visualizations.

```
# Loading files into dataframe

steam_df = pd.read_csv('/content/sample_data/steam.csv')
#steam_description_df = pd.read_csv('/content/sample_data/steam_description_data.csv')
#steam_media_df = pd.read_csv('/content/sample_data/steam_media_data.csv')
steam_requirements_df = pd.read_xml('/content/sample_data/steam_requirements_data.xml')
steam_support_df = pd.read_json('/content/sample_data/steam_support_info.json')
steam_tag_df = pd.read_excel('/content/sample_data/steamspy_tag_data.xls')
```

Using the read csv() method from the pandas package, this code reads multiple CSV files into Data Frames. The data files can be found in the subfolder /content/sample data/.

The steam.csv file contains details on Steam titles, including the title, launch date, and the amount of favorable and unfavorable reviews.

The minimum and recommended system requirements for Steam games are detailed in the steam requirements data.xml file.

Information on Steam game support, such as the number of supported languages and if the game supports controllers, is contained in the steam support_info.jason file.

The steam tag data.xls file contains details on game tags, including the total number of games with a specific tag and the owners of those games.

Drop any NULL,missing values or unwanted columns.

```
# steam dataframe information

print("Info\n")
steam_df.info()

print("\n\n\n")

print("Is Null\n")
steam_df.isnull().sum()
```

```
Info

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 27075 entries, 0 to 27074
Data columns (total 18 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   appid             27075 non-null  int64
 1   name              27075 non-null  object
 2   release_date      27075 non-null  object
 3   english           27075 non-null  int64
 4   developer         27075 non-null  object
 5   publisher         27075 non-null  object
 6   platforms         27075 non-null  object
 7   required_age      27075 non-null  int64
 8   categories        27075 non-null  object
 9   genres            27075 non-null  object
 10  steamspy_tags     27075 non-null  object
 11  achievements      27075 non-null  int64
 12  positive_ratings  27075 non-null  int64
 13  negative_ratings  27075 non-null  int64
 14  average_playtime  27075 non-null  int64
 15  median_playtime   27075 non-null  int64
 16  owners            27075 non-null  object
 17  price             27075 non-null  float64
dtypes: float64(1), int64(8), object(9)
memory usage: 3.7+ MB
```

```
Is Null

appid               0
name                0
release_date        0
english             0
developer           0
publisher           0
platforms           0
required_age        0
categories          0
genres              0
steamspy_tags       0
achievements        0
positive_ratings    0
negative_ratings    0
average_playtime    0
median_playtime     0
owners              0
price               0
dtype: int64
```

Using the isnull() and sum() methods, this code determines the number of null values (missing values) for each column in the steam df DataFrame.

Drop duplicate values.

```
duplicates = steam_df.duplicated()

# Count the number of duplicated values
num_duplicates = duplicates.sum()

print(f'The dataset contains {num_duplicates} duplicate values.')
```

```
The dataset contains 0 duplicate values.
```

Using the duplicated () method from pandas, this code examines the steam df Data Frame for duplicate rows.

Check for outliers using a box plot or histogram.

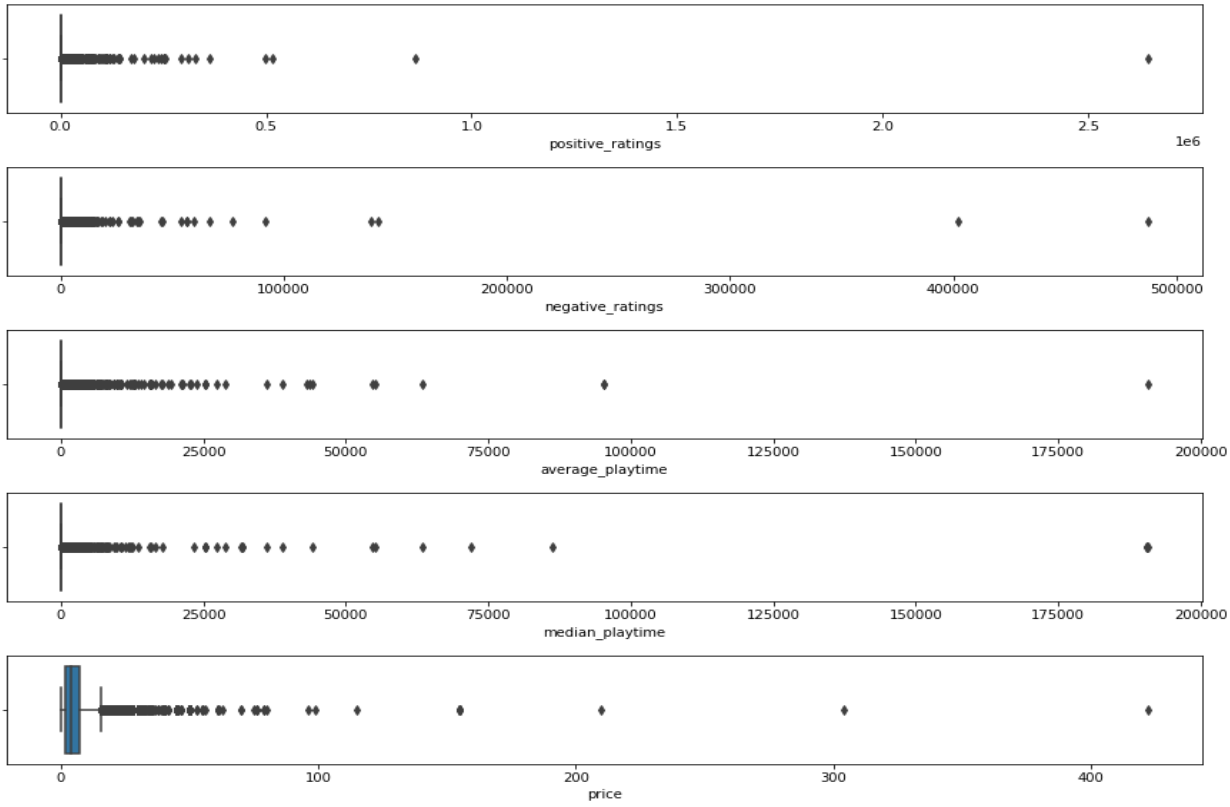
Using boxplot

```
# Box plot represenattion

cols = ['positive_ratings', 'negative_ratings', 'average_playtime', 'median_playtime','price']
plt.figure(figsize = (15,22))
plt.subplots_adjust(hspace = 0.5)
for i, col in enumerate(cols, 1):
  plt.subplot(9,1,i)
  sns.boxplot(steam_df[col])
```

This code creates a box plot for each of the columns specified in the cols list using the boxplot () function from the seaborn library, which is imported as sns.

Overall, this code generates a figure with five subplots arranged in a vertical layout, each showing a box plot for one of the selected columns from the steam_df DataFrame.

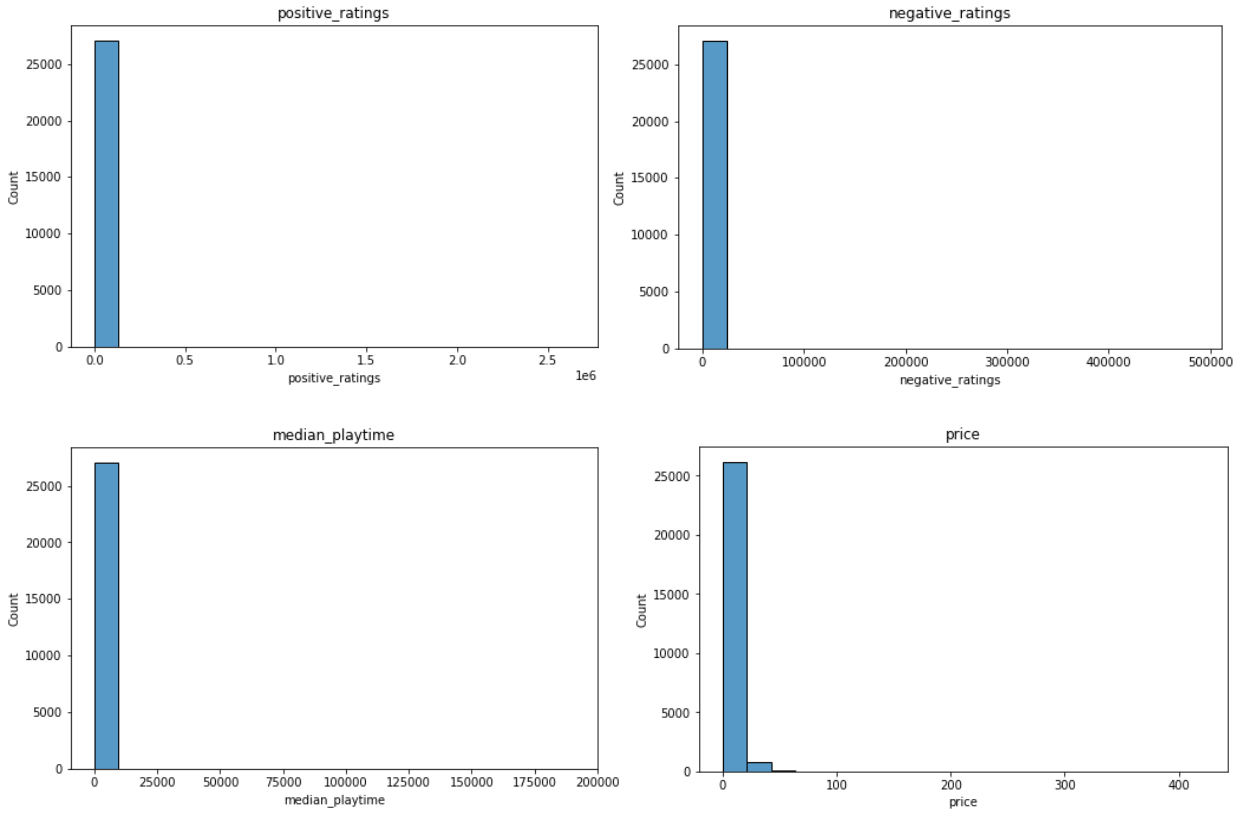
Output:



By Using Histogram:

```
# Histogram representation
cols = ['positive_ratings', 'negative_ratings', 'average_playtime', 'median_playtime','price']
for col in cols:
  plt.figure(figsize = (8,5))
  plt.subplots_adjust(hspace = 0.5)
  sns.histplot(x=col, data=steam_df, bins=20)
  plt.title(col)
  plt.show()
```

Output:


positive_ratings
Count
positive_ratings
1e6
negative_ratings
Count
negative_ratings
median_playtime
Count
median_playtime
price
Count
price

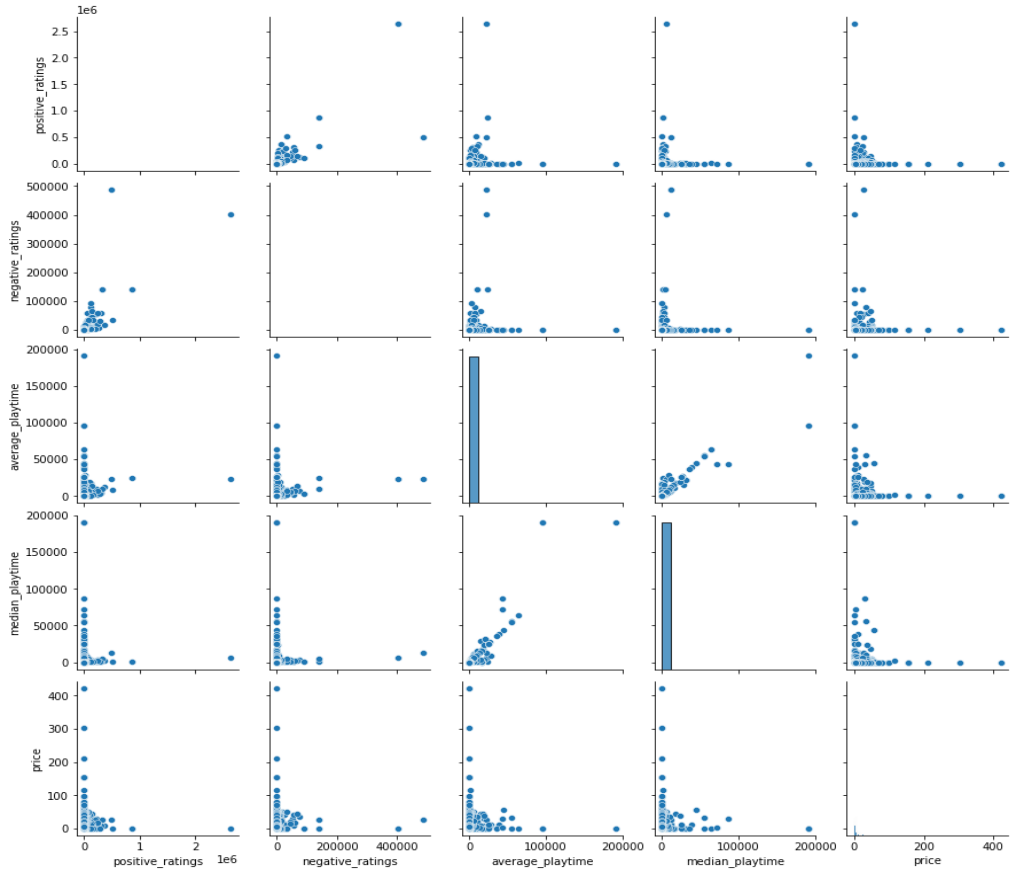
Plot features against each other using a pair plot.

```
# Pair plot representation

cols = ['positive_ratings', 'negative_ratings', 'average_playtime', 'median_playtime','price']

# Generate pair plot
sns.pairplot(data=steam_df[cols])

# Display plot
plt.show()
```

Using the pair plot() function, this code creates a pairwise scatter plot matrix for the columns listed in the cols list. The subset of columns to be included in the plot is specified by the data parameter, which is set to steam[cols].

The resulting plot displays pairwise scatter plots for each combination of the chosen columns, along with diagonal histograms for each variable. This kind of figure is helpful for illustrating the connections and correlations between different variables in a dataset.

Output:

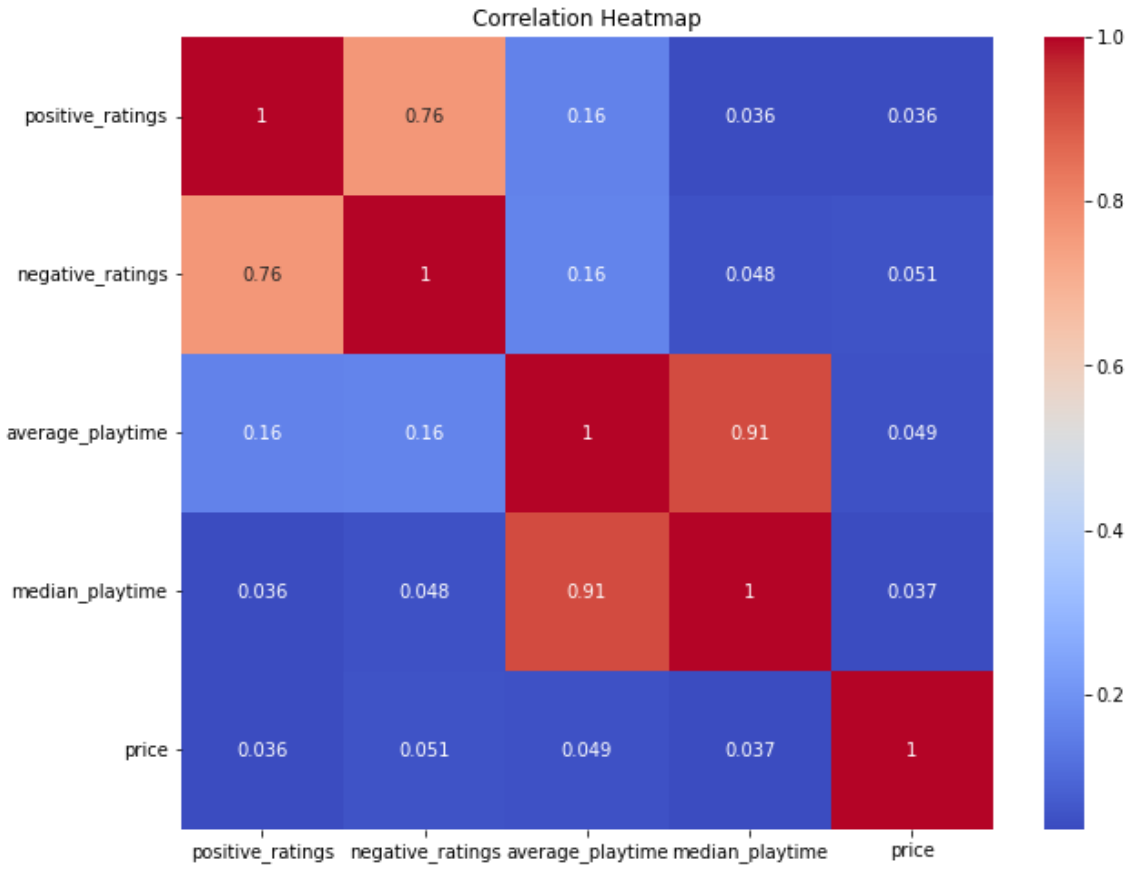


Use a Heatmap for finding the correlation between the features (Feature to Feature).

```
# Heat map representation

cols = ['positive_ratings', 'negative_ratings', 'average_playtime', 'median_playtime','price']

corr = steam_df[cols].corr()
plt.figure(figsize=(10, 8))
sns.heatmap(corr, cmap="coolwarm", annot=True)
plt.title("Correlation Heatmap")
plt.show()
```

Using the corr() method from pandas, this code generates the correlation matrix for the columns given in the cols list. The heatmap() function from the seaborn library, is then used to show the resulting correlation matrix as a heatmap. The heatmap's color map is specified by the cmap parameter's value of "coolwarm," and the annot parameter's value of True makes it possible to see the correlation coefficients for each cell.

Output:

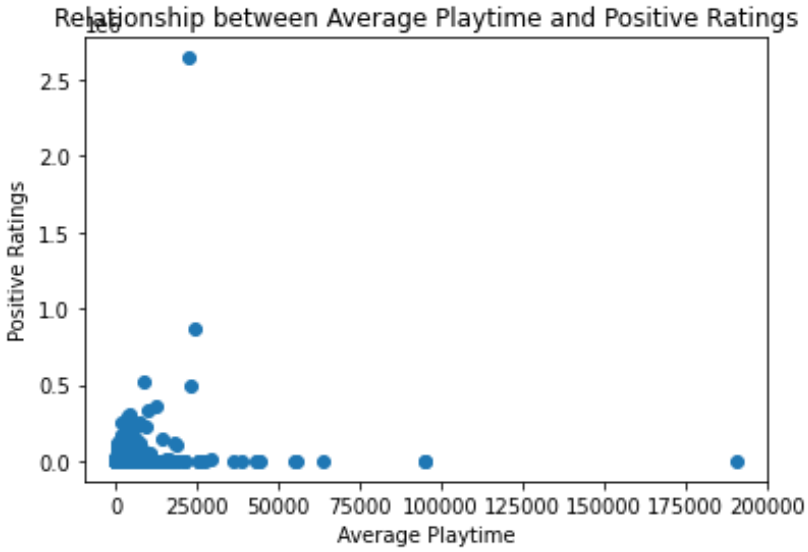


Use a scatter plot to show the relationship between 2 variables.

```
# Scatter plot representation to sgow relationship between average playtime and positive ratings

x = steam_df["average_playtime"]
y = steam_df["positive_ratings"]
plt.scatter(x, y)
plt.xlabel("Average Playtime")
plt.ylabel("Positive Ratings")
plt.title("Relationship between Average Playtime and Positive Ratings")
plt.show()
```

This code provides a way to visualize the relationship between two variables in the dataset and can be useful for identifying potential patterns or trends. In this case, the scatter plot shows a slight positive relationship between average playtime and positive ratings for Steam games.

Output:



Merging two Data Frames.

Steam and steam_tags_data are the two data frames merged and the header of the data is displayed.

```
merged_df = pd.merge(steam, steam_tags_df, on="appid")
```

```
merged_df.head()
```

	appid	name	release_date	english	developer	publisher	platforms	required_age	categories	genres	...	warhammer_40k
0	10	Counter-Strike	2000-11-01	1	Valve	Valve	windows;mac;linux	0	Multi-player;Online Multi-Player;Local Multi-P...	Action	...	0
1	20	Team Fortress Classic	1999-04-01	1	Valve	Valve	windows;mac;linux	0	Multi-player;Online Multi-Player;Local Multi-P...	Action	...	0
2	30	Day of Defeat	2003-05-01	1	Valve	Valve	windows;mac;linux	0	Multi-player;Valve Anti-Cheat enabled	Action	...	0
3	40	Deathmatch Classic	2001-06-01	1	Valve	Valve	windows;mac;linux	0	Multi-player;Online Multi-Player;Local Multi-P...	Action	...	0
4	50	Half-Life: Opposing Force	1999-11-01	1	Gearbox Software	Valve	windows;mac;linux	0	Single-player;Multi-player;Valve Anti-Cheat en...	Action	...	0

Checking for any NULL or missing values in the data.

```
merged_df.isnull().sum()
```

```
appid           0
name            0
release_date    0
english         0
developer       0
               ..
world_war_i     0
world_war_ii    0
wrestling       0
zombies         0
e_sports        0
Length: 390, dtype: int64
```

Slicing Data of a particular column value (like year, month, filter values depending on the categorical data)

```
# Selecting top 20 most common tags from 'steam_merged' dataframe

tag_counts = steam_merged_df.iloc[:, 19:].sum().sort_values(ascending=False)
print(tag_counts.head(20))
```

```
action              728416
indie               709652
adventure           540123
multiplayer         397327
casual              318387
rpg                 299875
open_world          294938
free_to_play        261649
first_person        203780
fps                 198945
co_op               197215
atmospheric         190813
great_soundtrack    177153
early_access        128743
horror              116094
funny               106462
2d                  102290
online_co_op         92518
difficult            92102
puzzle               91406
dtype: int64
```

This code finds the top 20 most popular tags for Steam games by first picking the columns in the merged df that include the tag information using the.iloc[:, 19:] indexing syntax, which selects all rows and columns beginning with the 19th column and going to the end.

The total number of games with each tag is then determined by using the.sum() method on the subset of columns. The sort values() method is used to sort the resulting counts in descending order with the ascending=False option.

Finally, the top 20 most popular tags are chosen from the sorted tag counts by using the.head(20)method.

```
# print top 10 genres with most tags
genre_counts = merged_df.groupby('genres').sum().iloc[:, 1:].sum(axis=1).sort_values(ascending=False)
print(genre_counts.head(10))
```

```
genres
Action                                  4907057.13
Action;Free to Play                     4795118.00
Action;Adventure                        3673127.09
Action;RPG                              1741369.27
RPG                                     1721547.29
Action;Indie                            1709738.42
Adventure;Indie                         1416920.54
Action;Adventure;Massively Multiplayer  1357258.94
Action;Adventure;Indie                  1353129.58
Strategy                                1257342.94
dtype: float64
```

Representing data in matrix form.

```
# Extract the required columns
matrix_df = steam[['name', 'release_date', 'genres', 'positive_ratings', 'negative_ratings']]
# Convert genres column to matrix form
genres_matrix = matrix_df['genres'].str.get_dummies(sep=', ')
# Concatenate the matrix columns
final_matrix = pd.concat([matrix_df[['name', 'release_date', 'positive_ratings', 'negative_ratings']], genres_matrix], axis=1)
# Display the final matrix
final_matrix.head()
```

This code creates a final matrix of features for Steam games by first extracting the required columns from the steam DataFrame using the .loc[] indexing syntax. The resulting subset of columns includes the game name, release date, genres, positive ratings, and negative ratings.

Output:

	name	release_date	positive_ratings	negative_ratings	Accounting;Animation & Modeling;Audio Production;Design & Illustration;Education;Photo Editing;Software Training;Utilities;Video Production;Web Publishing	Accounting;Education;Software Training;Utilities;Early Access	Action
0	Counter-Strike	2000-11-01	124534	3339	0	0	1
1	Team Fortress Classic	1999-04-01	3318	633	0	0	1
2	Day of Defeat	2003-05-01	3416	398	0	0	1
3	Deathmatch Classic	2001-06-01	1273	267	0	0	1
4	Half-Life: Opposing Force	1999-11-01	5250	288	0	0	1

5 rows × 1556 columns

Upload data to Numerical Python (NumPy)

```
steam_np = steam.to_numpy()
# Print the shape of NumPy array
print("Shape of steam_np:", steam_np.shape)
```

```
Shape of steam_np: (27075, 19)
```

The .to NumPy () method is used in this code to convert the steam Data Frame to a NumPy array, which results in a new array with the same data as the original Data Frame. The original Data Frame's rows and columns are all present in the resulting steam np array.

Select a slice or part of the data and display.

```
# Select a slice of the data

slice_df = steam_df.loc[10:20, ['name', 'release_date', 'positive_ratings']]
slice_df
```

	name	release_date	positive_ratings
10	Counter-Strike: Source	2004-11-01	76640
11	Half-Life: Source	2004-06-01	3767
12	Day of Defeat: Source	2010-07-12	10489
13	Half-Life 2: Deathmatch	2004-11-01	6020
14	Half-Life 2: Lost Coast	2005-10-27	5783
15	Half-Life Deathmatch: Source	2006-05-01	1362
16	Half-Life 2: Episode One	2006-06-01	7908
17	Portal	2007-10-10	51801
18	Half-Life 2: Episode Two	2007-10-10	13902
19	Team Fortress 2	2007-10-10	515879
20	Left 4 Dead	2008-11-17	17951

The names of the three columns ['name','release date', 'positive ratings'] are used and the resulting slice_df DataFrame contains only the specified rows and columns.

Use conditions and segregate the data based on the condition.

```
# Condition to segregate data
condition = steam['positive_ratings'] > 100000
# Segregate data based on condition
segregated_data = steam[condition]
# Display the segregated data
segregated_data
```

In this case the positive ratings column must contain a value larger than 100000 for the condition to apply. Games with lower positive ratings are successfully filtered out since only rows with a positive ratings column value larger than 100000 are included in the segregated data Frame.

Output:

	steamspy_tags	achievements	positive_ratings	negative_ratings	average_playtime	median_playtime	owners	price	age_category
	Action;FPS;Multiplayer	0	124534	3339	17612	317	10000000-20000000	7.19	All Ages
	Free to Play;Multiplayer;FPS	520	515879	34036	8495	623	20000000-50000000	0.00	All Ages
	Zombies;Co-op;FPS	70	251789	8418	1615	566	10000000-20000000	7.19	All Ages
	Free to Play;MOBA;Strategy	0	863507	142079	23944	801	100000000-200000000	0.00	All Ages
	Puzzle;Co-op;First-Person	51	138220	1891	1102	520	10000000-20000000	7.19	All Ages
	FPS;Multiplayer;Shooter	167	2644404	402313	22494	6502	50000000-100000000	0.00	All Ages
	Sandbox;Multiplayer;Funny	29	363721	16433	12422	1875	10000000-20000000	6.99	All Ages

Use mathematical and statistical functions using libraries.

```
# Calculate the mean, median, and standard deviation of the positive_ratings column
positive_ratings = steam["positive_ratings"]
mean = np.mean(positive_ratings)
median = np.median(positive_ratings)
std_dev = np.std(positive_ratings)
# Calculate the correlation coefficient between the positive_ratings and negative_ratings columns
positive_ratings = steam["positive_ratings"]
negative_ratings = steam["negative_ratings"]
correlation_coefficient = np.corrcoef(positive_ratings, negative_ratings)[0, 1]
# Print the results
print("Positive Ratings Mean:", mean)
print("Positive Ratings Median:", median)
print("Positive Ratings Standard D (type alias) correlation_coefficient: Any
print("Correlation Coefficient:", correlation_coefficient)
```

```
Positive Ratings Mean: 1000.5585226223453
Positive Ratings Median: 24.0
Positive Ratings Standard Deviation: 18988.373766546247
Correlation Coefficient: 0.7628042705085482
```

This code calculates some descriptive statistics for the positive_ratings column and the correlation coefficient between the positive_ratings and negative_ratings columns in the steam DataFrame. The numpy library is used to calculate the mean, median, and standard deviation of the positive_ratings column.

Select data based on a category (categorical data based).

```
# Select all rows where the genre is 'Action'
action_df = steam[steam_df['genres'].str.contains('Action')]

# Display the first 5 rows of the resulting dataframe
action_df.head()
```

With this code, a new dataframe called action_df will be created that contains all rows that have the word "Action" in the genres column. Only games that fall under the "Action" genre will be included in the final dataframe. The first five rows of the generated dataframe are then shown using the head() method.

Output:

	appid	name	release_date	english	developer	publisher	platforms	required_age	categories	genres	steamspy_tags
0	10	Counter-Strike	2000-11-01	1	Valve	Valve	windows;mac;linux	0	Multi-player;Online Multi-Player;Local Multi-P...	Action	Action;FPS;Multiplayer
1	20	Team Fortress Classic	1999-04-01	1	Valve	Valve	windows;mac;linux	0	Multi-player;Online Multi-Player;Local Multi-P...	Action	Action;FPS;Multiplayer
2	30	Day of Defeat	2003-05-01	1	Valve	Valve	windows;mac;linux	0	Multi-player;Valve Anti-Cheat enabled	Action	FPS;World War II;Multiplayer
3	40	Deathmatch Classic	2001-06-01	1	Valve	Valve	windows;mac;linux	0	Multi-player;Online Multi-Player;Local Multi-P...	Action	Action;FPS;Multiplayer
4	50	Half-Life: Opposing Force	1999-11-01	1	Gearbox Software	Valve	windows;mac;linux	0	Single-player;Multi-player;Valve Anti-Cheat en...	Action	FPS;Action;Sci-fi

Write your own functions using *args and handle exceptions in the functions.

```
def get_average_playtime(*games):
    try:
        playtime_list = []
        for game in games:
            playtime = steam_df.loc[steam_df['name'] == game, 'average_playtime'].values
            if len(playtime) > 0:
                playtime_list.append(playtime[0])
        if len(playtime_list) > 0:
            return sum(playtime_list) / len(playtime_list)
        else:
            return 0
    except pd.errors.EmptyDataError as e:
        print(f"Error: {e}. The dataset is empty.")
    except pd.errors.ParserError as e:
        print(f"Error: {e}. There was an error parsing the dataset.")
    except Exception as e:
        print(f"Error: {e}. An unexpected error occurred.")

# Call the function with a list of game names
games = ["Counter-Strike: Global Offensive", "Dota 2", "Team Fortress 2"]
average_playtime = get_average_playtime(*games)
print("Average playtime for the games is:",average_playtime)
```

```
Average playtime for the games is: 18311.0
```

This function is used to calculate the average playtime of the games that are passed in the parameters.
Moreover, this functions also handles all types of exceptions such as empty dataset and parsing error.

About Image Resizing

Image Resizing using SRC

```
# Image resizing using SRC
def image_resizing_SRC(src, percent):
  img = cv2.imread(src)

  print('Original Dimensions : ',img.shape)

  scale_percent = percent
  width = int(img.shape[1] * scale_percent / 100)
  height = int(img.shape[0] * scale_percent / 100)
  dim = (width, height)

  resized = cv2.resize(img, dim)

  print('Resized Dimensions : ',resized.shape)

  cv2_imshow(resized)
```

```
image_resizing_SRC('/content/sample_data/Valorant Image.jpg', 200)
```

Output

```
Original Dimensions :  (2160, 3840, 3)
Resized Dimensions :  (4320, 7680, 3)
```

We have an unstructured dataframe 'steam_media' which contains the URL of all the images and screenshots of the games, so we have used cv2 library for image processing to resize the given image, as we can see in this code we have to provide the image path and the percentage according to the original resolution of the image. For example, over here we have given 200 percent so it will increases its resolution by 2 times of the original image.

Image Resizing using URL

```
# Image resizing using URL
def image_resizing_URL(url, percent):

  req = urllib.request.urlopen(url)
  arr = np.asarray(bytearray(req.read()), dtype=np.uint8)
  img = cv2.imdecode(arr, -1)

  print('Original Dimensions : ',img.shape)

  scale_percent = percent
  width = int(img.shape[1] * scale_percent / 100)
  height = int(img.shape[0] * scale_percent / 100)
  dim = (width, height)

  resized = cv2.resize(img, dim)

  print('Resized Dimensions : ',resized.shape)

  cv2_imshow(resized)
```

```
image_resizing_URL('https://steamcdn-a.akamaihd.net/steam/apps/10/header.jpg?t=1528733245', 200)
```

Output:

```
Original Dimensions :  (215, 460, 3)
Resized Dimensions :  (430, 920, 3)
```

This is the same as the above function, but instead of passing the image location we have to pass the image URL during the function call, we have used the urllib.request library to fetch the image from the given URL.

SQL Integration

Created steam_library database using sql lite3

```
# Create database

conn = sqlite3.connect('Steam_Library.db')
```

Finally, after all the processsing and visualizations, we have integrated SQL to stores the results. We have used sqlite3 library to create the database 'steam_library'. However, before importing the data into the SQL tables we have changed their null values of the 'steam_requirements' dataframe to np.nan values using the .replace() method, as we can see in the images.

```
# View 'steam_requirements' dataframe

steam_requirements_df
```

	steam_appid	pc_requirements	mac_requirements	linux_requirements
0	10	{'minimum': '\r\n\t\t\t<p>Minimum:</st...	{'minimum': 'Minimum: OS X Snow Leopard 10.6....	{'minimum': 'Minimum: Linux Ubuntu 12.04, Dual...
1	20	{'minimum': '\r\n\t\t\t<p>Minimum:</st...	{'minimum': 'Minimum: OS X Snow Leopard 10.6....	{'minimum': 'Minimum: Linux Ubuntu 12.04, Dual...
2	30	{'minimum': '\r\n\t\t\t<p>Minimum:</st...	{'minimum': 'Minimum: OS X Snow Leopard 10.6....	{'minimum': 'Minimum: Linux Ubuntu 12.04, Dual...
3	40	{'minimum': '\r\n\t\t\t<p>Minimum:</st...	{'minimum': 'Minimum: OS X Snow Leopard 10.6....	{'minimum': 'Minimum: Linux Ubuntu 12.04, Dual...
4	50	{'minimum': '\r\n\t\t\t<p>Minimum:</st...	{'minimum': 'Minimum: OS X Snow Leopard 10.6....	{'minimum': 'Minimum: Linux Ubuntu 12.04, Dual...
...	...	...	...	...
27314	1065230	{'minimum': 'Minimum: <ul ...	[]	[]
27315	1065570	{'minimum': 'Minimum: <ul ...	[]	[]
27316	1065650	{'minimum': 'Minimum: <ul ...	[]	[]

As we can see steam_requirements data frame has null values in the form of '[]' in mac_requiremnets and Linux requirements columns.

Replacing ‘[]’ with np.nan

```
# Replacing '[]' values with np.nan

steam_requirements_df = steam_requirements_df.replace(r'[[]]', np.nan, regex=True)
```

Using this code we have replaced ’[]’ with np.nan, which is used to represent ‘not available’ values in the data.

```
# Review 'steam_requirements' dataframe

steam_requirements_df
```

	steam_appid	pc_requirements	mac_requirements	linux_requirements
0	10	{'minimum': '\r\n\t\t\t<p>Minimum:</st...	{'minimum': 'Minimum: OS X Snow Leopard 10.6....	{'minimum': 'Minimum: Linux Ubuntu 12.04, Dual...
1	20	{'minimum': '\r\n\t\t\t<p>Minimum:</st...	{'minimum': 'Minimum: OS X Snow Leopard 10.6....	{'minimum': 'Minimum: Linux Ubuntu 12.04, Dual...
2	30	{'minimum': '\r\n\t\t\t<p>Minimum:</st...	{'minimum': 'Minimum: OS X Snow Leopard 10.6....	{'minimum': 'Minimum: Linux Ubuntu 12.04, Dual...
3	40	{'minimum': '\r\n\t\t\t<p>Minimum:</st...	{'minimum': 'Minimum: OS X Snow Leopard 10.6....	{'minimum': 'Minimum: Linux Ubuntu 12.04, Dual...
4	50	{'minimum': '\r\n\t\t\t<p>Minimum:</st...	{'minimum': 'Minimum: OS X Snow Leopard 10.6....	{'minimum': 'Minimum: Linux Ubuntu 12.04, Dual...
...	...	...	...	...
27314	1065230	{'minimum': 'Minimum: <ul ...	NaN	NaN
27315	1065570	{'minimum': 'Minimum: <ul ...	NaN	NaN
27316	1065650	{'minimum': 'Minimum: <ul ...	NaN	NaN
27317	1066700	{'minimum': 'Minimum: <ul ...	{'minimum': 'Minimum: <ul ...	NaN

Replacing ‘Blank values’ with np.nan

```
# View 'steam_support' dataframe

steam_support_df
```

	steam_appid	website	support_url	support_email
0	10		http://steamcommunity.com/app/10	
1	30	http://www.dayofdefeat.com/		
2	50		https://help.steampowered.com	
3	70	http://www.half-life.com/	http://steamcommunity.com/app/70	
4	80		http://steamcommunity.com/app/80	
...	...	...	...	...
27131	1065230		https://goldlogsh.wixsite.com/myapps	zhang_frank@hotmail.com
27132	1065570			pizirоggg@gmail.com
27133	1065650	http://entwickler-x.de/super-star-blast	http://www.entwickler-x.de	mail@entwickler-x.de
27134	1066700	https://steamcommunity.com/groups/alawargames	http://www.alawar.com	steam@alawar.com
27135	1069460	https://steamcommunity.com/groups/alawargames	http://www.alawar.com	steam@alawar.com

27136 rows × 4 columns

Using this code we have filled Blank values with np.nan

Replacing ‘Blank values’ with np.nan

```
# View 'steam_support' dataframe

steam_support_df
```

	steam_appid	website	support_url	support_email
0	10		http://steamcommunity.com/app/10	
1	30	http://www.dayofdefeat.com/		
2	50		https://help.steampowered.com	
3	70	http://www.half-life.com/	http://steamcommunity.com/app/70	
4	80		http://steamcommunity.com/app/80	
...	...	...	...	...
27131	1065230		https://goldlogsh.wixsite.com/myapps	zhang_frank@hotmail.com
27132	1065570			pizirоggg@gmail.com
27133	1065650	http://entwickler-x.de/super-star-blast	http://www.entwickler-x.de	mail@entwickler-x.de
27134	1066700	https://steamcommunity.com/groups/alawargames	http://www.alawar.com	steam@alawar.com
27135	1069460	https://steamcommunity.com/groups/alawargames	http://www.alawar.com	steam@alawar.com

27136 rows × 4 columns

Using this code we have filled Blank values with np.nan

```
# Replacing null values with np.nan

steam_support_df = steam_support_df.replace(r'^\s*$', np.nan, regex=True)
```

Output:

```
# Review 'steam_support' dataframe

steam_support_df
```

	steam_appid	website	support_url	support_email
0	10	NaN	http://steamcommunity.com/app/10	NaN
1	30	http://www.dayofdefeat.com/	NaN	NaN
2	50	NaN	https://help.steampowered.com	NaN
3	70	http://www.half-life.com/	http://steamcommunity.com/app/70	NaN
4	80	NaN	http://steamcommunity.com/app/80	NaN
...	...	...	...	...
27131	1065230	NaN	https://goldlogsh.wixsite.com/myapps	zhang_frank@hotmail.com
27132	1065570	NaN	NaN	pizirоggg@gmail.com
27133	1065650	http://entwickler-x.de/super-star-blast	http://www.entwickler-x.de	mail@entwickler-x.de
27134	1066700	https://steamcommunity.com/groups/alawargames	http://www.alawar.com	steam@alawar.com
27135	1069460	https://steamcommunity.com/groups/alawargames	http://www.alawar.com	steam@alawar.com

27136 rows × 4 columns

Importing data into Sql tables from dataframe

```
# Creating data tables and importing data from dataframes

steam_df.to_sql('steam', conn, if_exists='replace', index = False)
steam_requirements_df.to_sql('steam_requirements', conn, if_exists='replace', index = False)
steam_support_df.to_sql('steam_support', conn, if_exists='replace', index = False)
steam_tag_df.to_sql('steam_support', conn, if_exists='replace', index = False)
steam_merged_df.to_sql('steam_merged', conn, if_exists='replace', index = False)
```

```
27075
```

Output:

```
cursor = conn.execute('''
  Select * from steam_merged;
''')
for row in cursor:
  print(row)
```

```
(10, 'Counter-Strike', '2000-11-01', 1, 'Valve', 'Valve', 'windows;mac;linux', 0, 'Multi-player;Online Mu
(20, 'Team Fortress Classic', '1999-04-01', 1, 'Valve', 'Valve', 'windows;mac;linux', 0, 'Multi-player;On
(30, 'Day of Defeat', '2003-05-01', 1, 'Valve', 'Valve', 'windows;mac;linux', 0, 'Multi-player;Valve Anti
(40, 'Deathmatch Classic', '2001-06-01', 1, 'Valve', 'Valve', 'windows;mac;linux', 0, 'Multi-player;Onlin
(50, 'Half-Life: Opposing Force', '1999-11-01', 1, 'Gearbox Software', 'Valve', 'windows;mac;linux', 0, '
(60, 'Ricochet', '2000-11-01', 1, 'Valve', 'Valve', 'windows;mac;linux', 0, 'Multi-player;Online Multi-Pl
(70, 'Half-Life', '1998-11-08', 1, 'Valve', 'Valve', 'windows;mac;linux', 0, 'Single-player;Multi-player;
(80, 'Counter-Strike: Condition Zero', '2004-03-01', 1, 'Valve', 'Valve', 'windows;mac;linux', 0, 'Single
(130, 'Half-Life: Blue Shift', '2001-06-01', 1, 'Gearbox Software', 'Valve', 'windows;mac;linux', 0, 'Sin
(220, 'Half-Life 2', '2004-11-16', 1, 'Valve', 'Valve', 'windows;mac;linux', 0, 'Single-player;Steam Achi
(240, 'Counter-Strike: Source', '2004-11-01', 1, 'Valve', 'Valve', 'windows;mac;linux', 0, 'Multi-player;
(280, 'Half-Life: Source', '2004-06-01', 1, 'Valve', 'Valve', 'windows;mac;linux', 0, 'Single-player', 'A
(300, 'Day of Defeat: Source', '2010-07-12', 1, 'Valve', 'Valve', 'windows;mac;linux', 0, 'Multi-player;C
(320, 'Half-Life 2: Deathmatch', '2004-11-01', 1, 'Valve', 'Valve', 'windows;mac;linux', 0, 'Multi-player
(340, 'Half-Life 2: Lost Coast', '2005-10-27', 1, 'Valve', 'Valve', 'windows;mac;linux', 0, 'Single-playe
(360, 'Half-Life Deathmatch: Source', '2006-05-01', 1, 'Valve', 'Valve', 'windows;mac;linux', 0, 'Multi-p
(380, 'Half-Life 2: Episode One', '2006-06-01', 1, 'Valve', 'Valve', 'windows;mac;linux', 0, 'Single-play
(400, 'Portal', '2007-10-10', 1, 'Valve', 'Valve', 'windows;mac;linux', 0, 'Single-player;Steam Achieveme
(420, 'Half-Life 2: Episode Two', '2007-10-10', 1, 'Valve', 'Valve', 'windows;mac;linux', 0, 'Single-play
(440, 'Team Fortress 2', '2007-10-10', 1, 'Valve', 'Valve', 'windows;mac;linux', 0, 'Multi-player;Cross-P
(500, 'Left 4 Dead', '2008-11-17', 1, 'Valve', 'Valve', 'windows;mac', 0, 'Single-player;Multi-player;Co-
(550, 'Left 4 Dead 2', '2009-11-19', 1, 'Valve', 'Valve', 'windows;mac;linux', 0, 'Single-player;Multi-pl
(570, 'Dota 2', '2013-07-09', 1, 'Valve', 'Valve', 'windows;mac;linux', 0, 'Multi-player;Co-op;Steam Trad
(620, 'Portal 2', '2011-04-18', 1, 'Valve', 'Valve', 'windows;mac;linux', 0, 'Single-player;Co-op;Steam A
(630, 'Alien Swarm', '2010-07-19', 1, 'Valve', 'Valve', 'windows', 0, 'Single-player;Multi-player;Co-op;S
```

Conclusion

We successfully investigated the distribution of games according to age limits and looked at the relationships between other factors like price, ratings, and its other categories in order to anticipate the drift about the various types of games.
Overall, a Python and SQL project employing the Steam dataset might offer insightful analysis of user behavior and the gaming business, which could be useful for game designers and marketers.