

Data Science Project

Analysis of Video Game Sales



April 24, 2023

**Group 10**

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **ID** | **Contribation** | **E-mail** |
| Ahmed Khaled | **185272** | data encoding ,aggregation, Data Visualization – Scrape , Clean, Integrate all relevant datasets | [Ahmed185272@bue.edu.eg](mailto:Ahmed185272@bue.edu.eg) |
| Ahmed Ramie | **217380** | Clean and tidy the data set - Scrape , Clean, Integrate all relevant datasets | [Ahmed217380@bue.edu.eg](mailto:Ahmed217380@bue.edu.eg) |
| Mariam | **196292** | data encoding ,aggregation, Data Visualization - answer the remaining questions with visualizations and formulate hypothesis | [Mariam196292@bue.edu.eg](mailto:Mariam196292@bue.edu.eg) |
| Mohamed Hany | **206457** | Clean and tidy the data set - - answer the remaining questions with visualizations and formulate hypothesis | [Mohamed206457@bue.edu.eg](mailto:Mohamed206457@bue.edu.eg) |

**PART ONE: The Project Proposal**

*description*

The dataset found at <https://www.kaggle.com/code/upadorprofzs/eda-video-game-sales> is a collection of video game sales data spanning several years and regions. The data contains information about various video games, including their titles, release dates, platforms, genres, publishers, and sales figures. The dataset consists of a single CSV file, which can be loaded into a data analysis tool or programming language for analysis.

The dataset contains the following columns:

1. Rank: The ranking of the game based on global sales.
2. Name: The name of the game.
3. Platform: The platform on which the game was released.
4. Year: The year in which the game was released.
5. Genre: The genre of the game.
6. Publisher: The publisher of the game.
7. NA\_Sales: The sales figures for the game in North America, measured in millions of units.
8. EU\_Sales: The sales figures for the game in Europe, measured in millions of units.
9. JP\_Sales: The sales figures for the game in Japan, measured in millions of units.
10. Other\_Sales: The sales figures for the game in regions other than North America, Europe, and Japan, measured in millions of units.
11. Global\_Sales: The total global sales figures for the game, measured in millions of units.

The data spans from 1980 to 2016, with a total of 16,598 entries. The dataset provides valuable insights into the video game industry, including which platforms and genres are most popular, and which publishers and games have had the most success in terms of sales. The dataset can be used for exploratory data analysis, predictive modeling, and other data-driven tasks related to the video game industry.

Goals

* **What is the top-selling game genres across different consoles?**
* Use groupby and aggregation functions to calculate total sales by genre and console and sort the results to identify the top-selling genres across different consoles.
* **Is there a correlation between a game's publisher and its total sales?**
* Use correlation analysis and visualization techniques to explore the relationship between a game's publisher and its total sales.
* **What is the distribution of total sales for each game genre?**
* Use visualization techniques such as histograms or box plots to visualize the distribution of total sales for each game genre.
* **Which consoles have the highest total sales over the years?**
* Use groupby and aggregation functions to calculate total sales by console over time and visualize the results using line charts or bar charts to identify the consoles with the highest total sales.
* **Are there any trends or patterns in the release of video games over time?**

Use time series analysis and visualization techniques to explore how the number of games released each year has changed over time, and consider the impact of major industry events on release trends.

**PART Two: Data Preparation**

Data Cleaning

**Text

Description automatically generated**

**This code loads a video game sales dataset from a CSV file using Pandas library, drops unnecessary columns, and renames the remaining columns to more descriptive names. It then drops rows with missing data in critical columns (Title, Genre, Publisher) and fills missing values in the 'Year' column with the median value. The 'Year' column is then converted to an integer data type. Duplicate rows are dropped, and the index is reset. Finally, the cleaned data set is exported to a new CSV file called 'cleaned\_vgsales.csv'.**

**Data encoding and aggregation**

**A picture containing text

Description automatically generated**

**the code encodes the categorical variables in the data set using one-hot encoding with Pandas' get\_dummies() function. The encoded variables are then concatenated with the original data set. The code then aggregates the data by year and console using Pandas' groupby() and agg() functions, which calculate the sum of 'Total\_Sales' for each unique combination of 'Year' and 'Console'. The resulting data frame is then reset and renamed before being exported to a new CSV file called 'cleaned\_and\_aggregated\_vgsales.csv'.**

**Finally, the cleaned and aggregated data set is printed to the console using the head() method to confirm that the data cleaning and preprocessing steps have been performed correctly.**

**PART Three: Data Visualization and Interpretation**

**Data Visualization and interpretation**

* **Q.1 What is the top-selling game genres across different consoles?**

**Scatter chart

Description automatically generated with low confidence**

**Chart, bar chart

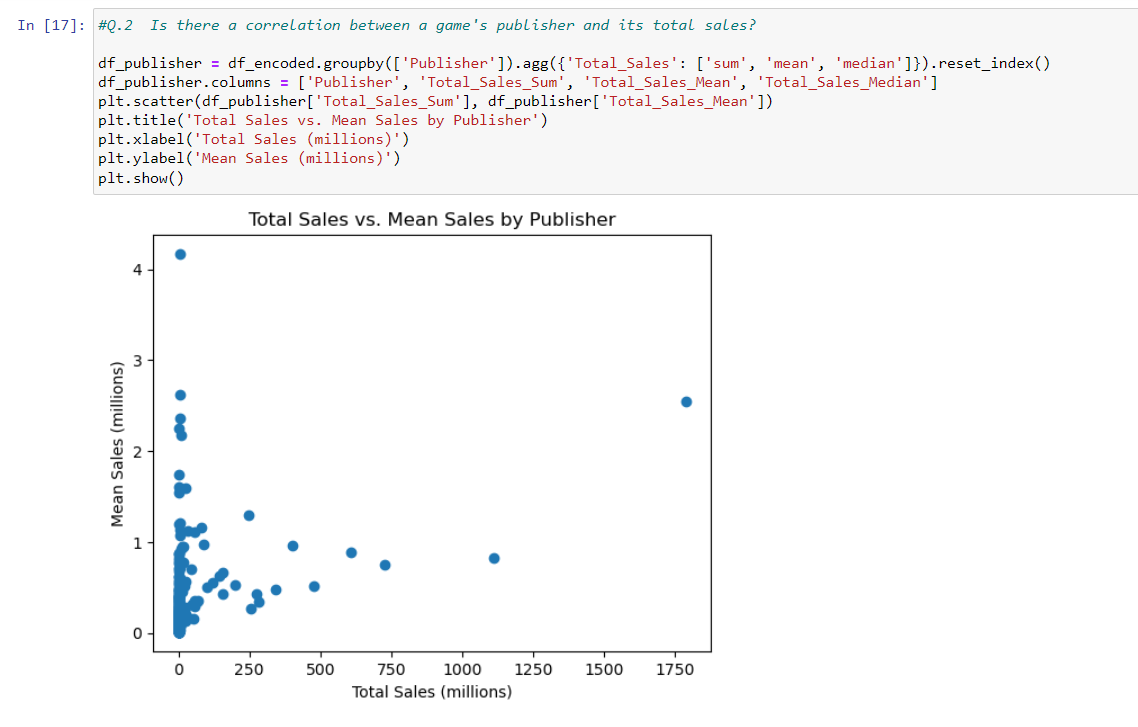
Description automatically generated**

**First, the code groups the cleaned and encoded data set by genre and console and aggregates the total sales for each group. The resulting table is then pivoted to reshape the data, with genres as the index and consoles as the columns.**

**Finally, a stacked bar chart is created using the pivoted table, where each stack represents the total sales of a console for a particular genre. The chart is plotted with the title "Total Sales by Genre and Console", x-axis labeled as "Genre", y-axis labeled as "Total Sales (millions)", and a size of 10x6.**

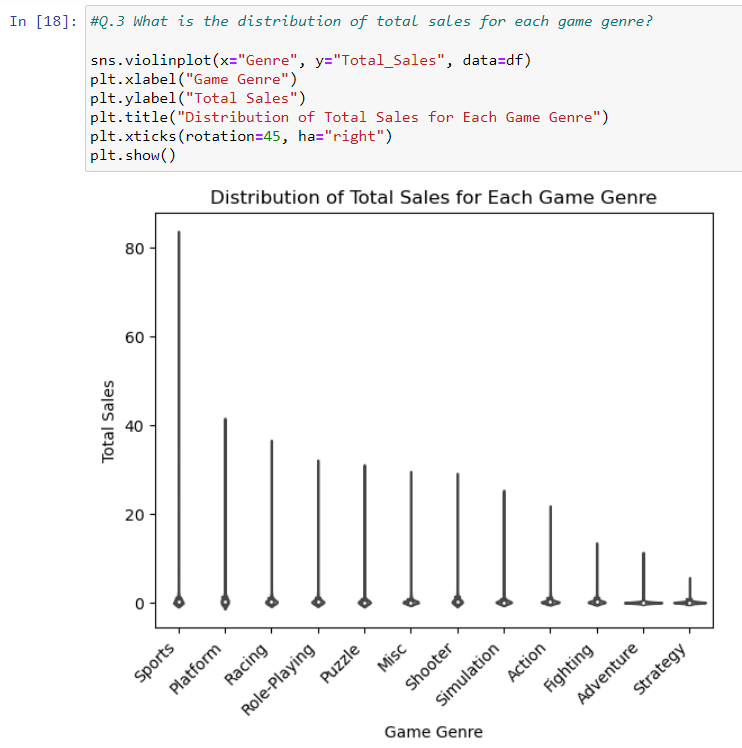
**The stacked bar chart can reveal which genres and consoles have higher total sales compared to others, and can help identify trends or patterns in the video game market.**

**Q.2 Is there a correlation between a game's publisher and its total sales?**

****

**This code loads the cleaned and encoded video game sales dataset, groups the data by publisher and calculates the total sales, mean and median of the sales for each publisher. Then it creates a scatter plot to visualize the correlation between the total sales and mean sales for each publisher.**

**Q.3 What is the distribution of total sales for each game genre?**

****

**This code creates a box plot using Seaborn library to show the distribution of total sales for each game genre in the dataset. The x-axis represents the game genre, and the y-axis represents the total sales. The plot shows the median value, the interquartile range, and any potential outliers for each game genre. The rotation and horizontal alignment of the x-axis labels are adjusted using plt.xticks() function. The labels and title are set using plt.xlabel(), plt.ylabel(), and plt.title() functions.**

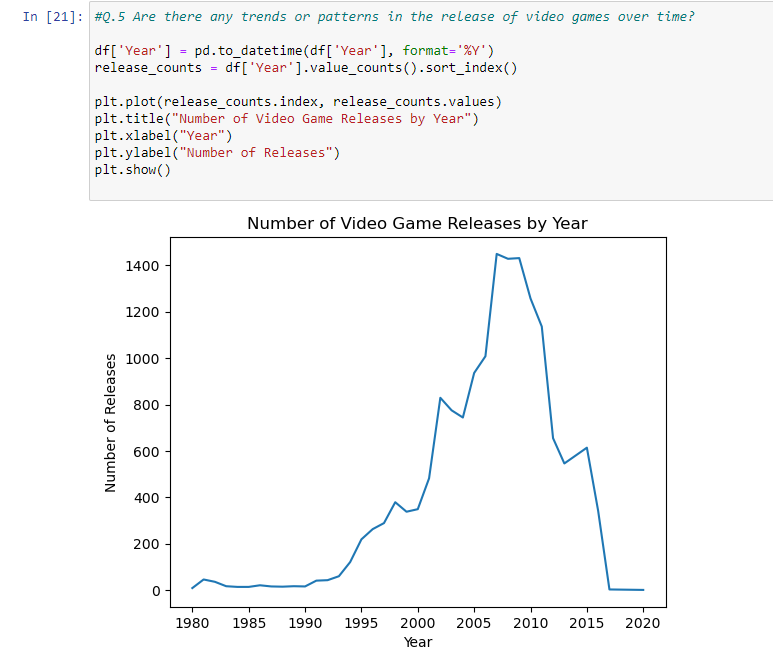
**Q.4 Which consoles have the highest total sales over the years?**

**Chart, histogram

Description automatically generated**

**This code first groups the cleaned and aggregated video game sales data by console and year, and then calculates the sum of total sales for each console and year combination. It then pivots the resulting table to create a new table with years as rows, consoles as columns, and total sales as values. Finally, it creates a line plot to visualize the total sales by console over time. The resulting plot shows how total sales have changed over time for each console. The x-axis shows the years, and the y-axis shows the total sales in millions of units. Each console is represented by a different colored line on the plot. The plot can help identify trends or patterns in the release of video games over time for each console.**

**Q.5 Are there any trends or patterns in the release of video games over time?**

****

**This code reads in a pandas dataframe from a CSV file and then converts the 'Year' column to a datetime object using the pd.to\_datetime() function. It then counts the number of video game releases by year using the value\_counts() function and plots the results using plt.plot(). The resulting plot displays the number of video game releases over time.**

**Phase 2**

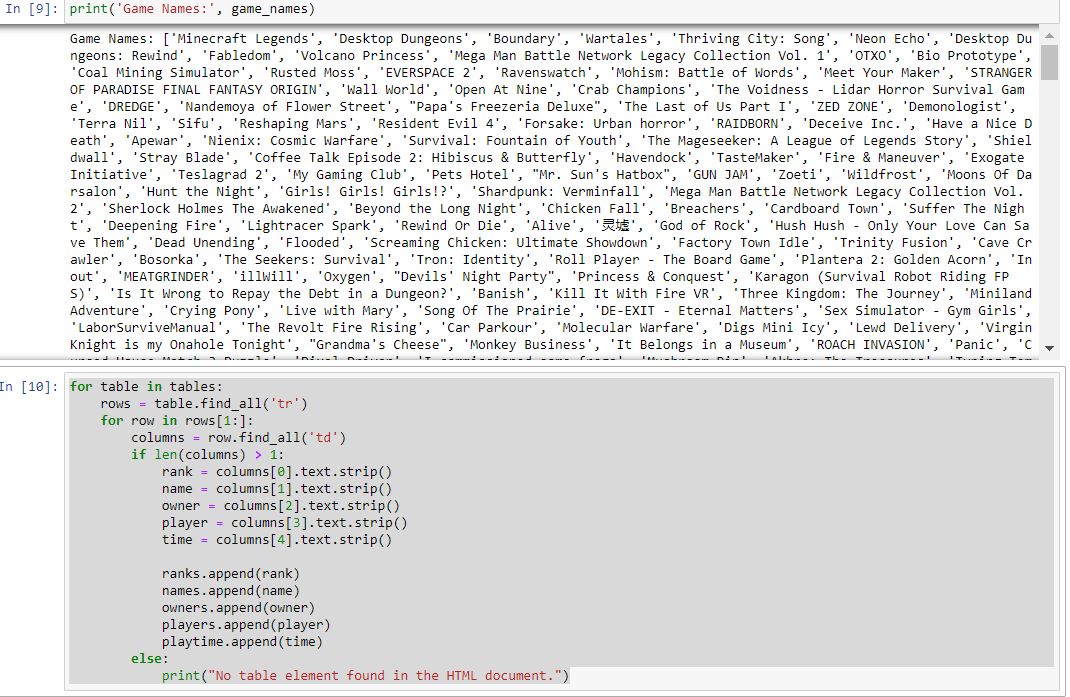
**Step 1:** **Scrape the extra data from a related HTML pages.  
  
**

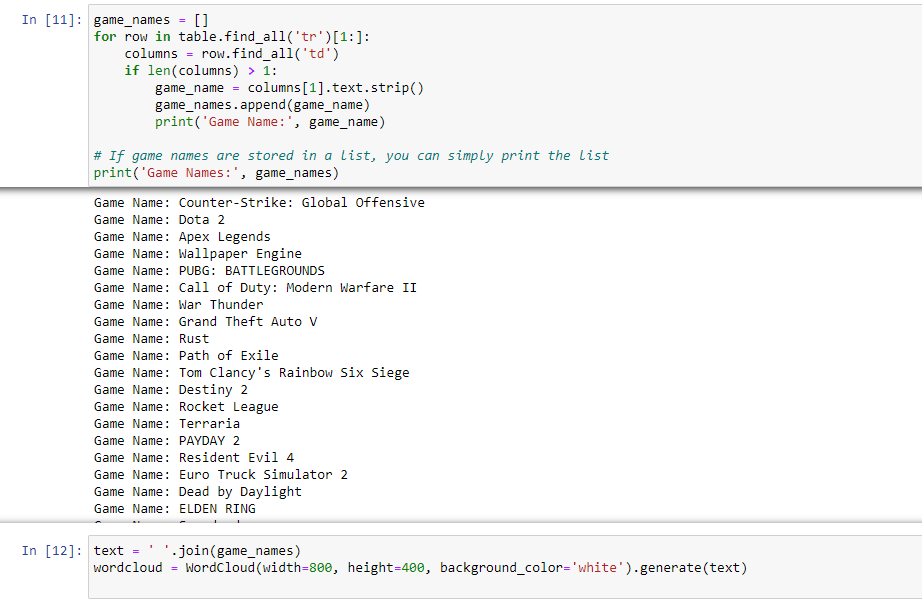
**These are Python commands that install two libraries named "wordcloud" and "missingno" using pip, which is a package installer for Python. The "wordcloud" library is used for creating word clouds, which are visual representations of text data. The library provides various customization options, such as changing the font size and color scheme, to create visually appealing word clouds. The "missingno" library is used for visualizing missing data in a dataset. It provides several types of plots, such as bar plots and matrix plots, to show where the missing data is located in the dataset. This library is helpful in understanding the quality and completeness of a dataset.**

**Graphical user interface, text, application, email

Description automatically generated**

**This code performs web scraping of the SteamSpy website. It starts by importing the required libraries, such as `requests`, `BeautifulSoup`, and installing additional libraries such as `wordcloud` and `missingno`. Then, the code specifies the URL of the website to scrape, which is https://steamspy.com/. The `requests.get()` function sends a GET request to the URL, and the response is stored in the `response` variable. The `BeautifulSoup()` function is used to parse the HTML content of the response, and the resulting object is stored in the `soup` variable.The code then searches for all tables on the page using the `find\_all()` method of the `soup` object, and stores them in the `tables` variable. It then searches for the table with the id `games`, and stores it in the `table` variable.Next, the code creates empty lists to store the scraped data, such as `game\_names`, `ranks`, `names`, `owners`, `players`, and `playtime`. Finally, the code uses a `for` loop to find all `a` tags on the page that contain the substring `/app/` in their `href` attribute, which are the links to individual game pages. For each such link found, it extracts the game name from the `text` attribute of the `a` tag, and appends it to the `game\_names` list. The game name is also printed to the console using the `print()` function.**

**  
This code iterates through all the tables found in the HTML document and then iterates through each row (except the first row) in each table. It then extracts the text content of each column in the row and stores it in separate lists for the rank, name, owner, player count, and playtime of each game. If there are no table elements found in the HTML document, it prints the message "No table element found in the HTML document."**

****

**The code is creating a word cloud visualization of the game names obtained from the SteamSpy website.**

**First, an empty list called game\_names is created to store the game names scraped from the table. Then, a loop is used to iterate through each row of the table, skipping the header row (hence table.find\_all('tr')[1:]). For each row, the code checks if there are at least 2 columns (i.e. if the row contains actual data), and if so, it extracts the game name from the second column (columns[1]) and appends it to the game\_names list.**

**Once all the game names have been extracted and stored in the game\_names list, the code creates a single string called text by joining all the game names together with a space separator using the join() method. This is done so that the WordCloud object can take in a single string of text to create the visualization.Finally, a WordCloud object is created with a specified width, height, and background color. The generate() method is used to generate the word cloud visualization from the text string, and the resulting image is stored in the wordcloud variable.**

****

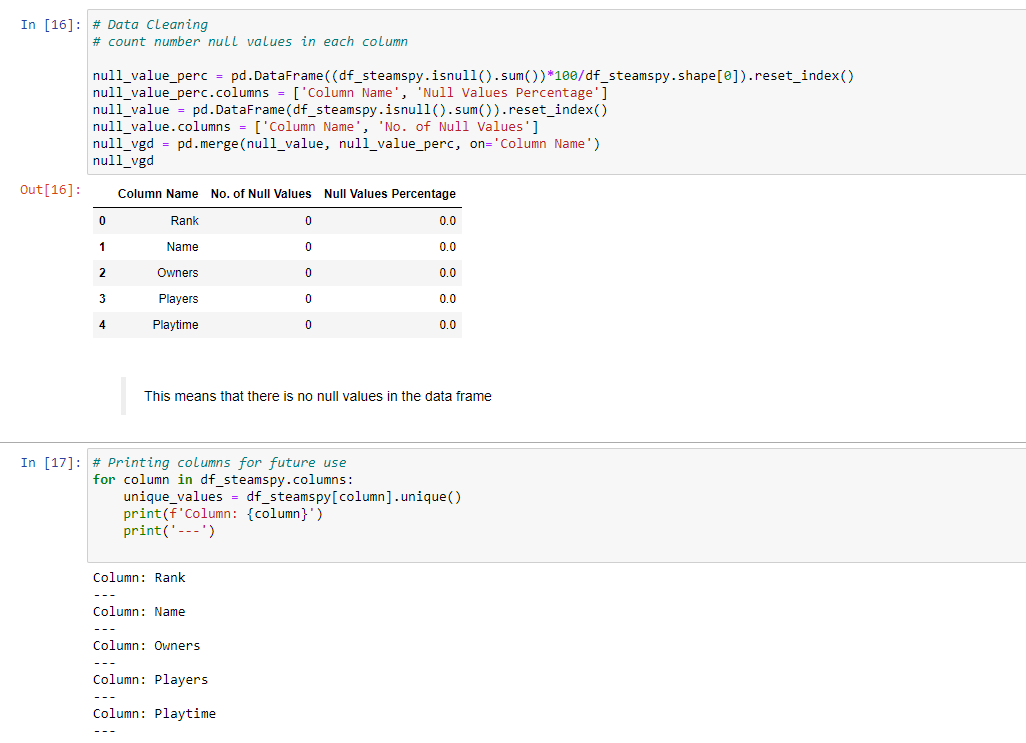
**This code generates a plot of the word cloud that was created earlier using the WordCloud library. The plt.figure() function specifies the size of the figure in inches, while plt.imshow() displays the image of the word cloud. plt.axis('off') removes the axis ticks and labels from the plot. Finally, plt.title() sets the title of the plot, and plt.show() displays the plot.**

**A picture containing graphical user interface

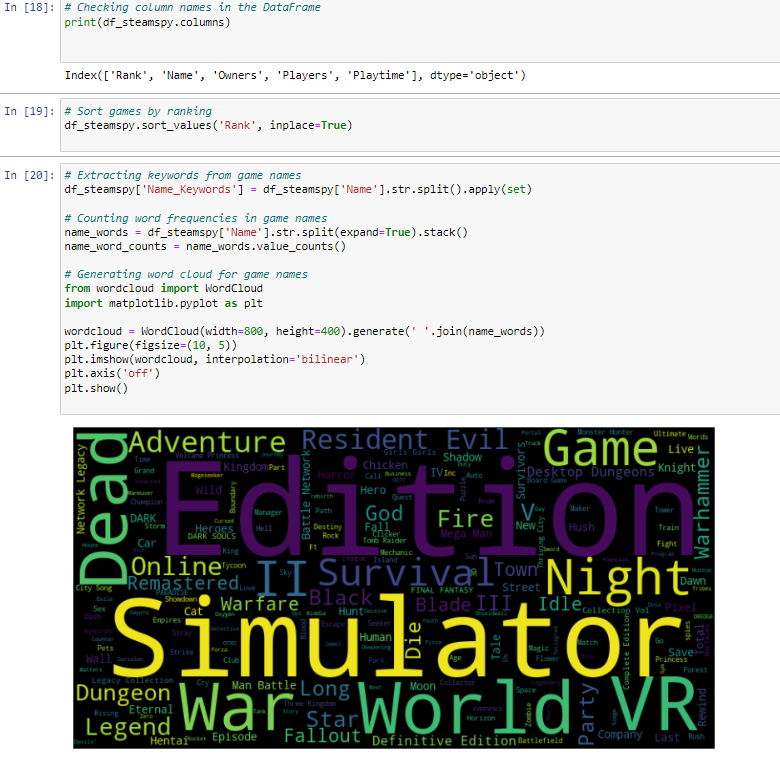
Description automatically generated**

**This code creates a pandas DataFrame from the lists that were populated with data scraped from the SteamSpy website using BeautifulSoup. The DataFrame has five columns named "Rank", "Name", "Owners", "Players", and "Playtime", and the data in each column corresponds to the respective lists that were populated during the scraping process. The head() method is called on the DataFrame to display the first few rows of the data in a tabular format.**

**Step 2: Clean and Tidy the Data and represent it with visualization**

****

**This code is performing data cleaning operations on the DataFrame df\_steamspy created in the previous code block. It calculates the percentage of null values in each column using pd.DataFrame.isnull().sum() and then creates two DataFrames: one containing the number of null values in each column, and another containing the percentage of null values in each column. These two DataFrames are then merged on the 'Column Name' column using pd.merge(), and the resulting DataFrame is printed. After that, a for loop is used to iterate over all columns in the df\_steamspy DataFrame, and for each column, the unique values are printed. This is likely done for future reference, so that the user can see the range of values present in each column.**

****

1. **Extracts keywords from the 'Name' column of the DataFrame using the split function and applies the set function to remove duplicates. The resulting sets are stored in a new column called 'Name\_Keywords'.**
2. **Counts the frequency of each word in the 'Name' column of the DataFrame using the value\_counts function. The resulting counts are stored in the 'name\_word\_counts' variable.**
3. **Generates a word cloud for the game names using the WordCloud library. The 'join' function is used to combine all words from the 'Name' column into a single string, which is then passed to the WordCloud function. The resulting word cloud is plotted using the Matplotlib library.**

**Text

Description automatically generated**

**This code snippet performs data cleaning and analysis on the df\_steamspy DataFrame. It first converts the Rank, Owners, Players, and Playtime columns to float values using the pd.to\_numeric() function with errors='coerce' to convert non-numeric values to NaN.**

**Then it calculates the average number of owners per game using the .mean() method of the Owners column. It also identifies the games with the highest and lowest number of owners using boolean indexing with .max() and .min() methods on the Owners column.**

**Finally, it creates a bar chart to visualize the ownership distribution of the games using the matplotlib.pyplot library. The x-axis shows the game names and the y-axis shows the number of owners for each game. The plt.xticks(rotation=90) command rotates the x-axis labels by 90 degrees to avoid overlap.**

**A picture containing graphical user interface

Description automatically generated**

**The code is dropping rows that have NaN values in the 'Playtime' column using the dropna() function from Pandas. The subset parameter is used to specify the column to check for NaN values. After dropping the NaN values, the code creates a histogram to visualize the distribution of playtime values. The hist() function from Matplotlib is used to create the histogram, with the bins parameter specifying the number of bins to use. The x-axis shows the playtime values in hours, and the y-axis shows the frequency of those values in the dataset. Overall, the code is checking for NaN values in the 'Playtime' column and dropping them, then creating a histogram to visualize the distribution of playtime values in the dataset.**

**Step 3: Integrate all relevant datasets (scraped and main), considering the best matching between the two data sets, and discuss any possible data loss as result of integration.**

**Text

Description automatically generated**

**This code creates a Pandas DataFrame from four lists: game\_names, owners, players, and playtime. First, it ensures that all the lists have the same length by padding the shorter lists with empty strings ('') or None values, as appropriate. Then, it creates the DataFrame using the pd.DataFrame() constructor, passing a dictionary with keys corresponding to the column names and values corresponding to the lists. Optionally, the code drops any rows with missing values in the Name column, and resets the index to start from 0. Finally, the resulting DataFrame is printed using the print() function.**

**Text

Description automatically generated with medium confidence**

**The code loads a dataset from a CSV file using pd.read\_csv function and assigns it to the df\_csv variable. It then prints the length of four arrays: game\_names, owners, players, and playtime. Next, a new DataFrame is created using these arrays and assigned to the variable df\_bs. Finally, the two DataFrames df\_csv and df\_bs are concatenated along the rows (axis=1) using the pd.concat function and assigned to the variable df\_combined. The resulting combined DataFrame is then printed. Note that the ignore\_index=True parameter ensures that the resulting DataFrame has a new index that is unique for all rows.**