Project Goal & Cleaning Data

The primary goal of this project is to build a recommendation system for steam gamess based on their metadata (Popularity, sales, ...) or features (Tags, Genres, ...).

This notebook focuses on cleaning the initial dataset (Which is very broken, the column names are wrong), cleaning the dataset, removing duplicates, creating new features to prepare the dataset for EDA.

In [1]:
```
import pandas as pd
import numpy as np
import ast
```

Cleaning

Fixing the columns, it's missing a comma between 2 columns

In [2]:
```
with open('Dataset/games.csv', encoding='utf-8') as f:
    s = f.read()

with open('Dataset/games_Fixed.csv', 'w', encoding='utf-8') as f:
    f.write(s.replace('DiscountDLC', 'Discount,DLC', 1))
```

In [3]:
```
df = pd.read_csv('Dataset/games_Fixed.csv')

df.head(5)
```

Out[3]:

	AppID	Name	Release date	Estimated owners	Peak CCU	Required age	Price	Discount	DLC count	A
0	20200	Galactic Bowling	Oct 21, 2008	0 - 20000	0	0	19.99	0	0	B exa anc
1	655370	Train Bandit	Oct 12, 2017	0 - 20000	0	0	0.99	0	0	T Lo sh
2	1732930	Jolt Project	Nov 17, 2021	0 - 20000	0	0	4.99	0	0	Jol r r
3	1355720	Henosis™	Jul 23, 2020	0 - 20000	0	0	5.99	0	0	H m 2D Pu
4	1139950	Two Weeks in Painland	Feb 3, 2020	0 - 20000	0	0	0.00	0	0	AB G. as

5 rows × 40 columns

In [4]:
```
print(df.columns)
```

```
Index(['AppID', 'Name', 'Release date', 'Estimated owners', 'Peak CCU',
       'Required age', 'Price', 'Discount', 'DLC count', 'About the game',
       'Supported languages', 'Full audio languages', 'Reviews',
       'Header image', 'Website', 'Support url', 'Support email', 'Windows',
       'Mac', 'Linux', 'Metacritic score', 'Metacritic url', 'User score',
       'Positive', 'Negative', 'Score rank', 'Achievements', 'Recommendations',
       'Notes', 'Average playtime forever', 'Average playtime two weeks',
       'Median playtime forever', 'Median playtime two weeks', 'Developers',
       'Publishers', 'Categories', 'Genres', 'Tags', 'Screenshots', 'Movies'],
      dtype='object')
```

Checking for duplicate games, there are none

In [5]:
```
df.duplicated(subset='AppID').sum()
```

Out[5]: 0

Initial Data Inspection

Before cleaning, we inspect the dataset to understand the structure, data types, missing values to plan to cleaning process

In [6]:
```
print("Info:")
# print(df.info().to_csv(index=False))

print("Missing Values:")
missing_values = df.isnull().sum()
display(missing_values[missing_values > 0].sort_values(ascending=False))

print("Basic Statistics:")
display(df[['Discount', 'User score']].describe())
```

```
Info:
Missing Values:
Score rank          97366
Metacritic url      93457
Reviews             87285
Notes               81936
Website             54673
Support url         51502
Tags                29763
Support email       16032
Movies               7891
Categories           5913
Publishers           5136
Developers           4876
About the game       4870
Genres               4841
Screenshots          2895
Name                    6
dtype: int64
Basic Statistics:
```

	Discount	User score
count	97410.0	97410.000000
mean	0.0	0.034791
std	0.0	1.674105
min	0.0	0.000000
25%	0.0	0.000000
50%	0.0	0.000000
75%	0.0	0.000000
max	0.0	100.000000

Dropping Rows and Columns

A variety of reasons to drop rows or columns, each of the reasons are mentioned in comments when dropping in the code cell.

In [7]:
```
columns_to_drop = [
    'Reviews', 'Metacritic url', 'Score rank', 'Notes', 'Metacritic score', # Hi
```

```
    'Website', 'Support url', 'Support email', # URLs/Emails
    'Header image', 'Screenshots', 'Movies', # Media links
    'Average playtime forever', 'Average playtime 2 weeks', 'Median playtime two
    'Discount', # No variance (always 0)
    'User score' # Unreliable (mostly 0)
]

df.drop(columns=columns_to_drop, inplace=True, errors='ignore')
print(f"Dropped {len(columns_to_drop)} columns.")

print(f"Remaining columns: {df.shape[1]}")
print("\nColumns after initial drop:")
print(df.columns.tolist())

print("Missing Values:")
missing_values = df.isnull().sum()
display(missing_values[missing_values > 0].sort_values(ascending=False))

drop_na_rows = [
    'Categories', 'Genres', 'Tags', # Since they are needed for analysis, we wil
    'Publishers', 'Developers', # Also cannot infer anything from them being emp
    "About the game", # We plan to do NLP on this column, so we drop empty rows 
]

print(f"Rows before dropping NaN: {df.shape[0]}")
df.dropna(subset=drop_na_rows, inplace=True)

print(f"Dropped {len(drop_na_rows)} columns with NaN values.")
print(f"Rows after dropping NaN: {df.shape[0]}")

print(f"New Shape: {df.shape}")
```

```
Dropped 16 columns.
Remaining columns: 25

Columns after initial drop:
['AppID', 'Name', 'Release date', 'Estimated owners', 'Peak CCU', 'Required age',
'Price', 'DLC count', 'About the game', 'Supported languages', 'Full audio langua
ges', 'Windows', 'Mac', 'Linux', 'Positive', 'Negative', 'Achievements', 'Recomme
ndations', 'Average playtime two weeks', 'Median playtime forever', 'Developers',
'Publishers', 'Categories', 'Genres', 'Tags']
Missing Values:
Tags              29763
Categories         5913
Publishers         5136
Developers         4876
About the game     4870
Genres             4841
Name                  6
dtype: int64
Rows before dropping NaN: 97410
Dropped 6 columns with NaN values.
Rows after dropping NaN: 66272
New Shape: (66272, 25)
```

Feature Engineering

Creating new features or transform existing ones to be more suitable for analysis and modeling

Game Age (Days) is created by the difference of the current date and the release date, then the original column is droped, this if for easier analysis later on

In [8]:
```
df['Release date'] = pd.to_datetime(df['Release date'], errors='coerce') # Conve

print(f"Earliest release date: {df['Release date'].min()}")

reference_date = df['Release date'].min()
df['Game Age (Days)'] = (df['Release date'] - reference_date).dt.days
median_age = df['Game Age (Days)'].median()

print(f"Number of missing values in 'Release date': {df['Release date'].isnull()

print(f"Median game age: {median_age} days")
df.drop(columns=['Release date'], inplace=True, errors='ignore') # No Longer nee

print(df['Game Age (Days)'].describe())
```

```
Earliest release date: 1997-06-30 00:00:00
Number of missing values in 'Release date': 0
Median game age: 8324.0 days
count    66272.000000
mean      8177.432068
std       1102.109736
min          0.000000
25%       7495.000000
50%       8324.000000
75%       8992.000000
max       9924.000000
Name: Game Age (Days), dtype: float64
```

is_indie is a binary feature for whether a game defined as indie in genre, this for testing if it makes a difference for recommendatinos later on.

In [9]:
```
def is_indie(genres):
    return 1 if 'Indie' in genres else 0

df['is_indie'] = df['Genres'].apply(is_indie)

print(f"Number of indie games: {df['is_indie'].sum()}")
print(f"Number of non-indie games: {df['is_indie'].count() - df['is_indie'].sum(
print(f"Proportion of indie games: {df['is_indie'].mean():.2%}")
```

```
Number of indie games: 47847
Number of non-indie games: 18425
Proportion of indie games: 72.20%
```

Since range of owners is a "Number - Number" (e.g "20000-50000"), we assign them to a single number instead for easier comparisons later on, with lower numbers representing lower range

In [10]:
```
print("Ranges of owners")
print(df['Estimated owners'].unique())
sorted_ranges = sorted(
    df['Estimated owners'].unique(),
    key=lambda x: int(x.split(' ')[0])
)
print(sorted_ranges)
```

```
ranges = {
    r: i for i, r in enumerate(sorted_ranges, start=1)
}

def convert_owners_to_category(owners):
    return ranges.get(owners, np.nan)

df['Owner range'] = df['Estimated owners'].apply(convert_owners_to_category)
print(df['Owner range'].head())

print(f"Number of unique owner ranges: {df['Owner range'].nunique()}")
df.drop(columns=['Estimated owners'], inplace=True, errors='ignore') # No longer
```

```
Ranges of owners
['0 - 20000' '50000 - 100000' '20000 - 50000' '200000 - 500000'
 '100000 - 200000' '2000000 - 5000000' '500000 - 1000000'
 '1000000 - 2000000' '20000000 - 50000000' '5000000 - 10000000'
 '10000000 - 20000000' '50000000 - 100000000' '100000000 - 200000000']
['0 - 20000', '20000 - 50000', '50000 - 100000', '100000 - 200000', '200000 - 500
000', '500000 - 1000000', '1000000 - 2000000', '2000000 - 5000000', '5000000 - 10
000000', '10000000 - 20000000', '20000000 - 50000000', '50000000 - 100000000', '1
00000000 - 200000000']
0    1
1    1
3    1
4    1
5    3
Name: Owner range, dtype: int64
Number of unique owner ranges: 13
```

Instead of raw counts of positive and negative, we calculate a review ratio, defaulting 0.5 if either to avoid dividing by 0. This will make it easier to compare between different games

In [11]:
```
df['Total Reviews'] = df['Positive'] + df['Negative'] # Adding up total positive
df['Review Ratio'] = np.where(
    df['Total Reviews'] > 0,
    df['Positive'] / df['Total Reviews'],
    0.5 # Assigning 0.5 when there are no reviews to avoid division by zero
)

df.drop(columns=['Positive', 'Negative'], inplace=True, errors='ignore') # No Lo
```

Supported Languages contain string representation of lists, this part parses the strings and extract only the useful information like number of languages and whether english is supported.

In [12]:
```
def parse_language_list(lang_str): # Convert string representation of list to ac
    if isinstance(lang_str, str) and lang_str.startswith('[') and lang_str.endsw
        try:
            return ast.literal_eval(lang_str)
        except (ValueError, SyntaxError):
            return []
    return []

df['Languages List'] = df['Supported languages'].apply(parse_language_list)

# Extracting the useful information from the language columns
```

```python
df['Num Languages'] = df['Languages List'].apply(len) # Extracting number of Lan
df['Is English Supported'] = df['Languages List'].apply(lambda x: 'English' in x

# Drop the intermediate and original language columns
df.drop(columns=['Supported languages', 'Full audio languages', 'Languages List'

display(df[['Num Languages', 'Is English Supported']].head())
df['Num Languages'].describe()
```

	Num Languages	Is English Supported
0	1	1
1	10	1
3	11	1
4	2	1
5	1	1

Out[12]:
```
count    66272.000000
mean         3.612129
std          4.960837
min          0.000000
25%          1.000000
50%          1.000000
75%          4.000000
max         39.000000
Name: Num Languages, dtype: float64
```

Processing the tags, genres and categories.

The tags, Genres and Categories columns contain comma-seperated strings, we need split them to seperate them, and also providing of preview of the types of content, we also remove Indie tags and genre since is_indie is already a feature.

In [13]:
```python
# tags = {tags count total.split(,) for tags count total in df['Tags']}
tags = {}
genres = {}
categories = {}

# Converting all tags, genres, and categories to seperate lists
for index, row in df.iterrows():
    tags_list = row['Tags'].split(',')
    genres_list = row['Genres'].split(',')
    categories_list = row['Categories'].split(',')

    for tag in tags_list:
        tag = tag.strip()
        tags[tag] = tags.get(tag, 0) + 1

    for genre in genres_list:
        genre = genre.strip()
        genres[genre] = genres.get(genre, 0) + 1

    for category in categories_list:
        category = category.strip()
        categories[category] = categories.get(category, 0) + 1
```

```
# Sorting the dictionaries by count
tags = dict(sorted(tags.items(), key=lambda item: item[1], reverse=True))
genres = dict(sorted(genres.items(), key=lambda item: item[1], reverse=True))
categories = dict(sorted(categories.items(), key=lambda item: item[1], reverse=T

# print number of unique values in Categories, Genres, Tags
print("Unique values in Categories:")
print(len(categories))
print("Unique values in Genres:")
print(len(genres))
print("Unique values in Tags:")
print(len(tags))

# remove Indie from genre and tags since not useful
tags.pop('Indie', None)
genres.pop('Indie', None)

print("Top 30 Tags:")
for tag, count in list(tags.items())[:30]:
    print(f"{tag}: {count}")

print()
print("Top 30 Genres:")
for genre, count in list(genres.items())[:30]:
    print(f"{genre}: {count}")

print()
print("Top 30 Categories:")
for category, count in list(categories.items())[:30]:
    print(f"{category}: {count}")
```

```
Unique values in Categories:
41
Unique values in Genres:
34
Unique values in Tags:
452
Top 30 Tags:
Singleplayer: 35664
Action: 30009
Casual: 28736
Adventure: 28219
2D: 18772
Strategy: 14336
Simulation: 14152
RPG: 12651
Puzzle: 12072
Atmospheric: 11997
3D: 11239
Early Access: 9852
Pixel Graphics: 9829
Story Rich: 9680
Colorful: 9617
Exploration: 8816
Cute: 8756
First-Person: 8424
Arcade: 8238
Multiplayer: 8165
Fantasy: 8161
Funny: 7147
Shooter: 6989
Horror: 6782
Retro: 6774
Platformer: 6715
Anime: 6413
Family Friendly: 6411
Sci-fi: 6379
Action-Adventure: 6274

Top 30 Genres:
Action: 28731
Casual: 26622
Adventure: 26514
Simulation: 13399
Strategy: 13330
RPG: 11954
Early Access: 8353
Free to Play: 3484
Sports: 3167
Racing: 2570
Massively Multiplayer: 1632
Violent: 512
Gore: 304
Utilities: 258
Design & Illustration: 155
Animation & Modeling: 132
Nudity: 116
Sexual Content: 105
Education: 98
Video Production: 69
Game Development: 66
```

```
Audio Production: 62
Software Training: 54
Web Publishing: 38
Photo Editing: 35
Accounting: 8
Movie: 2
Documentary: 1
Episodic: 1
Short: 1

Top 30 Categories:
Single-player: 63281
Steam Achievements: 32989
Steam Cloud: 17647
Full controller support: 14824
Multi-player: 13286
Steam Trading Cards: 9661
Partial Controller Support: 9164
PvP: 8106
Co-op: 6795
Steam Leaderboards: 6181
Online PvP: 5761
Remote Play Together: 5550
Shared/Split Screen: 5036
Online Co-op: 3765
Shared/Split Screen PvP: 3563
Stats: 3292
Family Sharing: 3020
Shared/Split Screen Co-op: 2921
Remote Play on TV: 2103
Cross-Platform Multiplayer: 1943
Includes level editor: 1822
Steam Workshop: 1782
In-App Purchases: 1461
Captions available: 1122
Remote Play on Tablet: 920
MMO: 863
Remote Play on Phone: 764
LAN PvP: 574
LAN Co-op: 527
VR Only: 388
```

We apply one-hot encoding to encode all the unique tags, genres and categories found in the dataset, each with it's own binary column indicating whether it is mentioned. This part is useful for machine learning models as it makes them easier to understand and use. This transformation allows each category to be treated independently without implying any false relationships between them.

In [14]:
```python
# one hot encoding first 100 tags, genres top 11, categories top 31
tags_to_encode = list(tags.keys())
genres_to_encode = list(genres.keys())
categories_to_encode = list(categories.keys())

# One-hot encoding the top tags, genres, and categories
def one_hot(column, vocab):
    data = { # True/False mask for every (row, token) pair, returned as a DataFr
        tok: column.apply(lambda s, t=tok: int(t in s))
        for tok in vocab
```

```
    }
    data = {f"{column.name}_{tok}": col for tok, col in data.items()} # Adding p
    return pd.DataFrame(data, index=column.index)

# Build three separate blocks
tags_block = one_hot(df["Tags"], tags_to_encode)
genres_block = one_hot(df["Genres"], genres_to_encode)
cats_block = one_hot(df["Categories"], categories_to_encode)

df = pd.concat( # Concatenate the original DataFrame with the one-hot encoded bl
        [df.drop(columns=["Tags", "Genres", "Categories"], errors="ignore"),
         tags_block, genres_block, cats_block],
        axis=1,
        copy=False
    )
```

Developer and Publisher Frequency, we tally the total number of games a developer/ publisher and made, which can be an indicator of their experience of size.

In [16]:
```
for col in ['Developers', 'Publishers']:
    if col in df.columns:
        # Calculate frequency of each column
        frequency_map = df[col].value_counts()

        # Map the frequency to the column
        df[f'{col} freq'] = df[col].map(frequency_map)

        display(df[[f'{col} freq']].head())
```

	Developers freq
0	1
1	4
3	1
4	1
5	1

	Publishers freq
0	1
1	5
3	1
4	1
5	1

Final checks of the dataset shape, types and missing values before saving it to a new CSV file to be processed by EDA.

In [17]:
```
# Final DataFrame Overview
print(f"Final DataFrame shape: {df.shape}")
print(df.dtypes.value_counts())
```

```python
final_missing = df.isnull().sum().sum()
print(f"Total missing values in final DataFrame: {final_missing}")

df.to_csv('Dataset/games_Cleaned.csv', index=False, encoding='utf-8') # Save the
```

```
Final DataFrame shape: (66272, 550)
int64      540
object       4
bool         3
float64      2
int32        1
dtype: int64
Total missing values in final DataFrame: 0
```