Recommendation System Project

By Brian Mallari

Project Description

The aim of this project is to create a simple recommendation system for a dataset relating to video games and users on the Steam platform.

Technologies Used

- Microsoft Windows 10 Home
- Python 3.10.2
- Microsoft Word 2016 (for tracking references)
- Spyder IDE Version 5.2.2
- Xfinity (for internet access)
- Jupyter Notebook (for creating this write-up)

Acknowledgements

This simple recommendation system was inspired by content by Mr. Aditya Sharma who created a couple of recommendation systems for movies on datacamp.com (URL = https://www.datacamp.com/community/tutorials/recommender-systems-python)

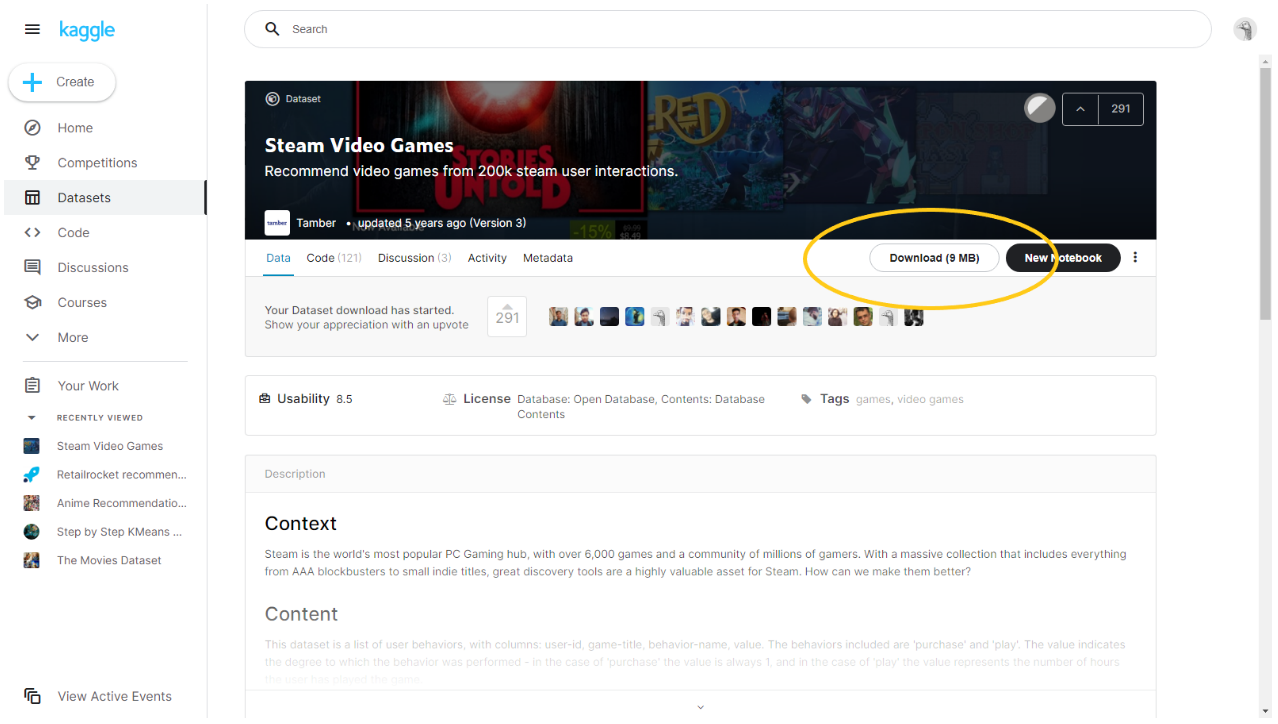
Acquiring the Data

The data used for this project can be found through the following URL: https://www.kaggle.com/tamber/steam-video-games/data

Registration will be needed in order to download the data. Once an account has been created at the website, the data can be downloaded by clicking "Download" towards the top of the webpage. A zip file should be downloaded to your machine.

In [1]:
```
from IPython import display
from base64 import b64decode
base64_data = "iVBORw0KGgoAAAANSUhEUgAABkAAAAOECAIAAAB2L2r1AAAAAXNSR0IArs4c6QAAAARnQU1B
display.Image(b64decode(base64_data))
```

Out[1]:



Steam Video Games

Recommend video games from 200k steam user interactions.

Your Dataset download has started.
Show your appreciation with an upvote

Usability 8.5 License Database: Open Database, Contents: Database Contents Tags games, video games

Description

Context

Steam is the world's most popular PC Gaming hub, with over 6,000 games and a community of millions of gamers. With a massive collection that includes everything from AAA blockbusters to small indie titles, great discovery tools are a highly valuable asset for Steam. How can we make them better?

Content

This dataset is a list of user behaviors, with columns: user-id, game-title, behavior-name, value. The behaviors included are 'purchase' and 'play'. The value indicates the degree to which the behavior was performed - in the case of 'purchase' the value is always 1, and in the case of 'play' the value represents the number of hours the user has played the game.

Exploratory Data Analysis

The pandas library for Python had been imported for this part of the project:

In [2]:
```python
import pandas as pd
```

Next, the Steam games data had been imported into a pandas object:

In [3]:
```python
steam_games_metadata = pd.read_csv('steam-200k.csv', header = None, low_memory = False)
```

The display configuration was then set to showcase all columns when viewing a pandas object:

In [4]:
```python
pd.options.display.max_columns = None
```

For good measure, the contents of the first few rows have been observed in order to confirm that things are going well so far:

In [5]:
```python
row_count = 5;
print(steam_games_metadata.head(row_count))
```

```
           0                            1         2      3  4
0  151603712  The Elder Scrolls V Skyrim  purchase    1.0  0
1  151603712  The Elder Scrolls V Skyrim      play  273.0  0
2  151603712                   Fallout 4  purchase    1.0  0
3  151603712                   Fallout 4      play   87.0  0
4  151603712                       Spore  purchase    1.0  0
```

The webpage that the Steam data came from said that starting from the far left, the columns represent user ID, game title, player behavior, and values assoicated with a particular player behavior. There's no mention on the webpage as to what the last column (the column labeled "4") represents; however, this column may very well still end up being helpful if any pattern can be discerned from the values of the column.

Now, for good measure the data type for the pandas object was reviewed in order confirm that things are going as expected:

In [6]:
```
print(type(steam_games_metadata))
```

```
<class 'pandas.core.frame.DataFrame'>
```

The shape of the DataFrame (i.e., the number of rows and columns of the DataFrame) had then been confirmed:

In [7]:
```
print(steam_games_metadata.shape)
```

```
(200000, 5)
```

So for this DataFrame, there are 200,000 rows and 5 columns.

The values in the column labeled "0" of the DataFrame were then explicitly set to be str data types in order for those values to be treated as user IDs instead of actual numbers:

In [8]:
```
steam_games_metadata_mod_1 = steam_games_metadata.astype({0: str})
```

The original DataFrame had then been deleted in order to save some RAM:

In [9]:
```
del steam_games_metadata
```

Next, some descriptive statistics were acquired for the columns with numeric values in the modified pandas object:

In [10]:
```
print(steam_games_metadata_mod_1.describe())
```

```
                   3         4
count  200000.000000  200000.0
mean       17.874384       0.0
std       138.056952       0.0
min         0.100000       0.0
25%         1.000000       0.0
50%         1.000000       0.0
75%         1.300000       0.0
max     11754.000000       0.0
```

It turns out that the column labeled "3" includes both values related to the "purchase" behavior of Steam users and values related to the "play" behavior of the same users, so the descriptive statistics for this column won't really mean much.

Moreover, the descriptive statistics for the column labeled "4" show that all of the values for this column are 0. It's not clear whether any value can be derived from 200,000 zeros, so it's been considered safe to just simply ignore this column.

Some descriptive statistics were then acquired for the columns of the latest modified pandas object with non-numeric values:

In [11]:

```
print(steam_games_metadata_mod_1.iloc[ : , [0, 1, 2]].describe())
```

```
               0       1         2
count     200000  200000    200000
unique     12393    5155         2
top     62990992  Dota 2  purchase
freq        1573    9682    129511
```

For the column labeled "0", there are 12393 unique user IDs with user 6299092 appearing the most often (1573 times), so this user is probably an avid gamer. In the column labeled "1", there are 5155 unique game titles with Dota 2 showing up most often (9682 times), so this game is likley to be one of the most popular. In the column labeled "2", there are two unique values for player behavior which makes sense because the choices are only "purchase" and "play". However, "purchase" shows up 129511 times which is more than half of the total number of rows in this column, 200000. This implies that there are have been more instances of game purchases than instances of game plays.

The count of "purchase" instances and the count of "play" instances were determined, and then the two values were compared in case the two counts are different:

In [12]:

```
print(steam_games_metadata_mod_1.iloc[ : , 2].value_counts())
```

```
purchase    129511
play         70489
Name: 2, dtype: int64
```

Now this confirms that there are actually more game purchases than there are game plays. This information can be useful later on.

Recap of the Different Modifications Done during the Course of This Exploratory Analysis

mod_1 => changed the data type for column with index 0 from int to str

Simple Recommender

This simple recommendation system was inspired by content by Mr. Aditya Sharma on datacamp.com
(URL = https://www.datacamp.com/community/tutorials/recommender-systems-python)

The data exploration and the simple recommender were each coded using different Python scripts in order to separate one task from the other. Therefore, the pandas library for Python had been imported once again for this part of the project:

In [13]:

```
import pandas as pd
```

The Steam games metadata had been reloaded for the simple recommender script. As we saw in the prior data exploration, this data doesn't have a header, so the data will be imported with the same function arguments as before:

In [14]:
```
steam_games_metadata = pd.read_csv('steam-200k.csv', header = None, low_memory = False)
```

Once again, the configuration was set to show all columns in a pandas object:

In [15]:
```
pd.options.display.max_columns = None
```

And like before, the content for the first few rows were printed to verify that things are going well so far:

In [16]:
```
row_count = 5;
print(steam_games_metadata.head(row_count))
```

```
           0                           1         2      3  4
0  151603712  The Elder Scrolls V Skyrim  purchase    1.0  0
1  151603712  The Elder Scrolls V Skyrim      play  273.0  0
2  151603712                   Fallout 4  purchase    1.0  0
3  151603712                   Fallout 4      play   87.0  0
4  151603712                       Spore  purchase    1.0  0
```

Like in the case of data exploration, the data type for the first column (the column labeled "0") in the pandas object steam_games_metadata was changed because the values for that were originally treated as numbers (int) instead of user IDs (str):

In [17]:
```
steam_games_metadata_mod_1 = steam_games_metadata.astype({0: str})
```

Also, the original pandas object was deleted in order to free up some RAM:

In [18]:
```
del steam_games_metadata
```

The data types for each column of the latest pandas object were then be identified in order to confirm that things are going as expected:

In [19]:
```
print(steam_games_metadata_mod_1.dtypes)
```

```
0     object
1     object
2     object
3    float64
4      int64
dtype: object
```

The "object" type for the column labeled "0" was interpreted to refer to a str data object because had the data in the column remained as integer values, then the type would come up as int64 instead.

Next, the last column (the column labeled "4") was removed because that has no important data as determined by the prior data exloration:

In [20]:

```
steam_games_metadata_mod_2 = steam_games_metadata_mod_1.drop(columns = 4)
```

Like in the case of the original pandas object, the pandas object after the first modificiation was deleted in order to save some RAM:

In [21]:
```
del steam_games_metadata_mod_1
```

For good measure, the content for the first few rows of the latest pandas object was printed to verify that things are going well so far:

In [22]:
```
row_count = 5;
print(steam_games_metadata_mod_2.head(row_count))
```

```
           0                            1         2      3
0  151603712  The Elder Scrolls V Skyrim  purchase    1.0
1  151603712  The Elder Scrolls V Skyrim      play  273.0
2  151603712                   Fallout 4  purchase    1.0
3  151603712                   Fallout 4      play   87.0
4  151603712                       Spore  purchase    1.0
```

Formal column names were then added to the pandas object based off of the description of the data found at https://www.kaggle.com/tamber/steam-video-games/data:

In [23]:
```
steam_games_metadata_mod_2.columns = ['user-id', 'game-title', 'behavior-name', 'value'
steam_games_metadata_mod_3 = steam_games_metadata_mod_2
```

The pandas object after the second modificiation was deleted in order to save some RAM:

In [24]:
```
del steam_games_metadata_mod_2
```

Once again, the content for the first few rows of the latest pandas object were printed to verify that things are going well so far:

In [25]:
```
row_count = 5;
print(steam_games_metadata_mod_3.head(row_count))
```

```
     user-id                  game-title behavior-name  value
0  151603712  The Elder Scrolls V Skyrim      purchase    1.0
1  151603712  The Elder Scrolls V Skyrim          play  273.0
2  151603712                   Fallout 4      purchase    1.0
3  151603712                   Fallout 4          play   87.0
4  151603712                       Spore      purchase    1.0
```

Next, the count of each unique user behavior item was grouped by game title because according to the prior data exploration, the number of purchases does not equal the number of plays:

In [26]:
```
steam_games_metadata_mod_4 = steam_games_metadata_mod_3.groupby(['game-title', 'behavio
```

And like before, the content of the first few rows of the latest pandas object was printed to confirm that things are going as expected:

```
In [27]: row_count = 10;
         print(steam_games_metadata_mod_4.head(row_count))
```

```
game-title                                              behavior-name
007 Legends                                             play             1
                                                        purchase         1
0RBITALIS                                               play             3
                                                        purchase         3
1... 2... 3... KICK IT! (Drop That Beat Like an Ugly Baby)  play         5
                                                        purchase         7
10 Second Ninja                                         play             2
                                                        purchase         6
10,000,000                                              play             1
                                                        purchase         1
Name: behavior-name, dtype: int64
```

Ten rows were printed this time around in order to showcase the first five pairs of purchase and play values for some of the games.

So here, we can see that in the case of one of the games, 1... 2... 3... KICK IT! (Drop That Beat Like an Ugly Baby), the value associated with the play behavior for this game (5) is different from the value assoicated with the purchase behavior (7). Another game, 10 Second Ninja, displays a similiar discrepency - the value associated with the play behavior for this game (2) is different from the value associated with the purchase behavior (6). These two cases align with the observation made during data exploration that there have been more instances of purchases for some games than actual plays of those games.

Line of Reasoning for the Simple Recommender

Because time and attention can be considered to be precious resources, the number of hours played for any game were treated as a measure of how much time and attention users were willing and able to allocate to a particular game as opposed to another game (or any other activity for that matter). However, not all games that have been purchased were necessarily played. Therefore, only games that users were willing and able to both purchase and play will be taken into consideration, and the games with the most user time and attention invested (i.e., total hours played) will be considered for recommendation.

Isolating Purchased Behavior

From here, just the instances of purchased behavior among all of the games had been isolated:

```
In [28]: steam_games_metadata_mod_5 = steam_games_metadata_mod_3[steam_games_metadata_mod_3['beh
```

And like before, the content for the first few rows of the latest pandas object were printed to verify that things are going well so far:

```
In [29]: row_count = 5;
         print(steam_games_metadata_mod_5.head(row_count))
```

```
     user-id                  game-title behavior-name  value
0  151603712  The Elder Scrolls V Skyrim      purchase    1.0
2  151603712                   Fallout 4      purchase    1.0
```

```
4  151603712                  Spore     purchase    1.0
6  151603712       Fallout New Vegas    purchase    1.0
8  151603712           Left 4 Dead 2    purchase    1.0
```

Next, the counts of purchased behavior by game title were grouped together:

In [30]:
```
steam_games_metadata_mod_6 = steam_games_metadata_mod_5.groupby(['game-title'])['behavi
```

And as a check, the content for the first few rows of this latest pandas object were printed to verify that things are going well so far:

In [31]:
```
row_count = 5;
print(steam_games_metadata_mod_6.head(row_count))
```

```
game-title
007 Legends                                                 1
0RBITALIS                                                   3
1... 2... 3... KICK IT! (Drop That Beat Like an Ugly Baby)    7
10 Second Ninja                                             6
10,000,000                                                  1
Name: behavior-name, dtype: int64
```

Next, the data type for this latest pandas object had been identified in order to figure out what actions can be taken next in terms of shaping the data for the purposes of this project:

In [32]:
```
print(type(steam_games_metadata_mod_6))
```

```
<class 'pandas.core.series.Series'>
```

This latest object had then been converted from a pandas Series to a pandas DataFrame for use later on:

In [33]:
```
steam_games_metadata_mod_7 = steam_games_metadata_mod_6.to_frame()
```

And to confirm that things are going well, the data type for this new pandas object had been confirmed:

In [34]:
```
print(type(steam_games_metadata_mod_7))
```

```
<class 'pandas.core.frame.DataFrame'>
```

Next, the shape of this DataFrame had been confirmed in order to figure out what operations can be done next to shape the data for further analysis:

In [35]:
```
print(steam_games_metadata_mod_7.shape)
```

```
(5155, 1)
```

Here we have a data frame with 5155 rows and a single column.

Next, the name of this one column had been identified. This information will be useful later on when shaping data for analysis:

In [36]:
```
print(steam_games_metadata_mod_7.columns)
```

```
Index(['behavior-name'], dtype='object')
```

The column name had then been changed to something that better explains what the column represents:

In [37]:
```
steam_games_metadata_mod_8 = steam_games_metadata_mod_7.rename(columns = {'behavior-nam
```

And like before, the content for the first few rows of the latest pandas object were printed in order to verify that things are going well so far:

In [38]:
```
row_count = 5;
print(steam_games_metadata_mod_8.head(row_count))
```

```
                                                    purchase-count
game-title
007 Legends                                                      1
0RBITALIS                                                        3
1... 2... 3... KICK IT! (Drop That Beat Like an...               7
10 Second Ninja                                                  6
10,000,000                                                       1
```

Isolating Played Behavior

Going forward, just like the case with the purchased behavior, the played behavior among all of the games had been isolated:

In [39]:
```
steam_games_metadata_mod_9 = steam_games_metadata_mod_3[steam_games_metadata_mod_3['beh
```

Next, the content for first few rows of this new pandas object were printed in order to verify that things are going well so far:

In [40]:
```
row_count = 5;
print(steam_games_metadata_mod_9.head(row_count))
```

```
     user-id                  game-title behavior-name  value
1  151603712  The Elder Scrolls V Skyrim          play  273.0
3  151603712                   Fallout 4          play   87.0
5  151603712                       Spore          play   14.9
7  151603712           Fallout New Vegas          play   12.1
9  151603712               Left 4 Dead 2          play    8.9
```

Like in the case of purchased behavior, the counts of played behavior had been grouped by game title:

In [41]:
```
steam_games_metadata_mod_10 = steam_games_metadata_mod_9.groupby(['game-title'])['behav
```

The content for the first few rows of this new pandas object was printed to verify that things are still going as intended:

In [42]:
```
row_count = 5;
```

```
print(steam_games_metadata_mod_10.head(row_count))
```

```
game-title
007 Legends                                              1
0RBITALIS                                                3
1... 2... 3... KICK IT! (Drop That Beat Like an Ugly Baby)    5
10 Second Ninja                                          2
10,000,000                                               1
Name: behavior-name, dtype: int64
```

The data type for steam_games_metadata_mod_10 had been identify to confirm that things are going well:

In [43]:
```
print(type(steam_games_metadata_mod_10))
```

```
<class 'pandas.core.series.Series'>
```

This object had then been converted from a pandas Series to a pandas DataFrame:

In [44]:
```
# convert steam_games_metadata_mod_10 from a pandas series to a pandas dataframe
steam_games_metadata_mod_11 = steam_games_metadata_mod_10.to_frame()
```

Next, the data type for steam_games_metadata_mod_11 had then been identified in order to confirm that things are going as intended:

In [45]:
```
print(type(steam_games_metadata_mod_11))
```

```
<class 'pandas.core.frame.DataFrame'>
```

As an additional check, the shape of this DataFrame had been examined:

In [46]:
```
print(steam_games_metadata_mod_11.shape)
```

```
(3600, 1)
```

Here, we have a DataFrame with 3600 rows and a single column. Contrast this with the case of purchased behavior where the DataFrame has 5155 rows and a single column.

Next, the name of the column for this "played behavior" DataFrame was identified:

In [47]:
```
print(steam_games_metadata_mod_11.columns)
```

```
Index(['behavior-name'], dtype='object')
```

Like in the case of "purchased behavior", the column name for this latest pandas object had been changed to something that better explains what the column actually represents:

In [48]:
```
steam_games_metadata_mod_12 = steam_games_metadata_mod_11.rename(columns = {'behavior-n
```

And like before, the content for the first few rows of this latest pandas object were printed to verify that things are going well so far:

In [49]:
```
row_count = 5;
```

```
print(steam_games_metadata_mod_12.head(row_count))
```

```
                                                      play-count
game-title                                                      
007 Legends                                                    1
0RBITALIS                                                      3
1... 2... 3... KICK IT! (Drop That Beat Like an...             5
10 Second Ninja                                                2
10,000,000                                                     1
```

And so far, things do appear to be going well.

Merging the Two Isolated Game Behaviors

Next, only the games that had been both purchased and played were isolated because these games were good enough to not only buy but also to spend time to actually play:

In [50]:
```
steam_games_metadata_mod_13 = steam_games_metadata_mod_8.join(steam_games_metadata_mod_
```

And as usual, content for the first few rows of the latest pandas object were printed to verify that things are going well so far:

In [51]:
```
row_count = 5;
print(steam_games_metadata_mod_13.head(row_count))
```

```
                                                      purchase-count  play-count
game-title                                                                      
007 Legends                                                        1           1
0RBITALIS                                                          3           3
1... 2... 3... KICK IT! (Drop That Beat Like an...                 7           5
10 Second Ninja                                                    6           2
10,000,000                                                         1           1
```

Next, some descriptive statistics for the latest pandas object, steam_games_metadata_mod_13, were acquired. This was to help confirm that things are going well, and ideally the counts for both purchased games and played games should be the same:

In [52]:
```
print(steam_games_metadata_mod_13.describe())
```

```
       purchase-count   play-count
count     3600.000000  3600.000000
mean        32.833889    19.580278
std        119.683883   104.314868
min          1.000000     1.000000
25%          3.000000     1.000000
50%          8.000000     4.000000
75%         26.000000    12.000000
max       4841.000000  4841.000000
```

The data type for each column of the same pandas ojbect was identified. This information will be useful in a bit:

In [53]:
```
print(steam_games_metadata_mod_13.dtypes)
```

```
purchase-count    int64
```

```
play-count        int64
dtype: object
```

And as a check, a simple test had been done to see if numeric equivalence works between an int and a float:

In [54]:
```
print(1 == 1.0)
print(type(1))
print(type(1.0))
```

```
True
<class 'int'>
<class 'float'>
```

Lo and behold, it looks like numeric equivalence really does work between an int and a float.

Next, the games with a purchase count that wasn't equal to their play count were filter out, or more specifically with a purchase count that was greater than the play count which would imply that people were willing and able to buy the game, but not willing and able to spend time playing the game as well:

In [55]:
```
steam_games_metadata_mod_14 = steam_games_metadata_mod_13[steam_games_metadata_mod_13['
```

And the content for the first few rows of the latest pandas object were once again printed to verify that things are going well so far:

In [56]:
```
row_count = 5;
print(steam_games_metadata_mod_14.head(row_count))
```

```
                         purchase-count  play-count
game-title
007 Legends                           1           1
0RBITALIS                             3           3
10,000,000                            1           1
15 Days                               1           1
1701 A.D. Sunken Dragon               1           1
```

Next, the games were grouped by the sum of hours played for each game title:

In [57]:
```
steam_games_metadata_mod_15 = steam_games_metadata_mod_9.groupby(['game-title'])['value
```

And another check was done by printing content for the first few rows of this latest pandas object:

In [58]:
```
row_count = 5;
print(steam_games_metadata_mod_15.head(row_count))
```

```
game-title
007 Legends                                              0.7
0RBITALIS                                                1.2
1... 2... 3... KICK IT! (Drop That Beat Like an Ugly Baby)    20.0
10 Second Ninja                                          5.9
10,000,000                                               3.6
Name: value, dtype: float64
```

The data type for this latest pandas ojbect, steam_games_metadata_mod_15, was identified. This information will also be helpful in a bit:

In [59]:
```
print(type(steam_games_metadata_mod_15))
```

```
<class 'pandas.core.series.Series'>
```

Here we have a pandas Series, and like before the Series was converted to a pandas DataFrame:

In [60]:
```
steam_games_metadata_mod_16 = steam_games_metadata_mod_15.to_frame()
```

The data type for this object, steam_games_metadata_mod_16, was then verified:

In [61]:
```
print(type(steam_games_metadata_mod_16))
```

```
<class 'pandas.core.frame.DataFrame'>
```

As an additional check, the shape of this pandas object was identified:

In [62]:
```
print(steam_games_metadata_mod_16.shape)
```

```
(3600, 1)
```

In other words, there are 3,600 rows and 1 column in this pandas ojbect.

The names of the columns were then identified for steam_games_metadata_mod_16:

In [63]:
```
print(steam_games_metadata_mod_16.columns)
```

```
Index(['value'], dtype='object')
```

This one column name was changed to something that better describes the content of the column itself:

In [64]:
```
steam_games_metadata_mod_17 = steam_games_metadata_mod_16.rename(columns = {'value': 'h
```

And like before, the content for the first few rows of this pandas object were viewed in order to verify that things are going well so far:

In [65]:
```
row_count = 5;
print(steam_games_metadata_mod_17.head(row_count))
```

```
                                                  hours-played
game-title                                                    
007 Legends                                                0.7
0RBITALIS                                                  1.2
1... 2... 3... KICK IT! (Drop That Beat Like an...        20.0
10 Second Ninja                                            5.9
10,000,000                                                 3.6
```

The pandas object with hours played for each game was joined to the pandas object where the count of purchases for each game is equal to the count of plays for the same game:

In [66]:
```
steam_games_metadata_mod_18 = steam_games_metadata_mod_14.join(steam_games_metadata_mod
```

The content for the first few rows of the latest pandas object was printed in order to verify that things are going well so far:

In [67]:
```
row_count = 5;
print(steam_games_metadata_mod_18.head(row_count))
```

```
                         purchase-count  play-count  hours-played
game-title
007 Legends                           1           1           0.7
0RBITALIS                             3           3           1.2
10,000,000                            1           1           3.6
15 Days                               1           1           0.5
1701 A.D. Sunken Dragon               1           1           0.6
```

Next, the game titles of this latest pandas object were sorted by total hours played in descending order:

In [68]:
```
steam_games_metadata_mod_19 = steam_games_metadata_mod_18.sort_values(by=['hours-played
```

And then the content for the first few rows of this pandas object were printed to verify that things are going well so far. In this case, we should end up with the top ten games to recommend:

In [69]:
```
row_count = 10;
print(steam_games_metadata_mod_19.head(row_count))
```

```
                           purchase-count  play-count  hours-played
game-title
Dota 2                               4841        4841      981684.6
Team Fortress 2                      2323        2323      173673.3
Football Manager 2013                 104         104       32308.6
DC Universe Online                    154         154        4674.7
Total War ATTILA                       36          36        2654.7
Mortal Kombat X                        38          38        2598.1
Pro Evolution Soccer 2015              12          12        2500.5
Zombie Panic Source                    86          86        2440.7
Football Manager 2016                  27          27        2296.6
The Sims(TM) 3                         44          44        2249.7
```

So there we have it - the top ten games to recommend based off of this dataset.

Recap of the Different Modifications Done with the Data while Building the Simple Recommender

mod_1 => changed the data type for column with index 0 from int to str

mod_2 => dropped column index 4 from the dataframe

mod_3 => added column names to the dataframe

mod_4 => grouped count of each user behavior by game title

mod_5 => isolated just the purchased behavior among all of the games

mod_6 => grouped the count of purchased behavior by game title

mod_7 => converted object from a pandas series to a pandas dataframe
mod_8 => renamed the column titled behavior-name to purchase-count

mod_9 => isolated just the play behavior among all of the games
mod_10 => grouped the count of played behavior by game title
mod_11 => converted object from a pandas series to a pandas dataframe
mod_12 => renamed the column titled behavior-name to play-count

mod_13 => join mod_8 and mod_12 dataframes along the indeces (game titles)
mod_14 => filter out the games where the purchase count and the play count aren't equal
mod_15 => grouped the sum of hours played by game title
mod_16 => converted object from a pandas series to a pandas dataframe
mod_17 => renamed the column titled value to hours-played

mod_18 => join mod_14 and mod_17 dataframes along the indeces (game titles)
mod_19 => sort the game titles of mod_18 by total hours played in descending order

Worth noting here is that unlike the case of the exploratory data analysis, many of the pandas objects weren't deleted while building the simple recommender. This was because it was not clear which intermediary pandas object would or wouldn't be needed when shaping the data to reach the final list of video games to recommend.