ai_notebook

September 24, 2024

1 Exploratory Data Analysis

The dataset that I am going to be using for creating an AI Recommendation Engine is -

Steam Video Games from Kaggle - https://www.kaggle.com/datasets/tamber/steam-video-games

About the Dataset

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.

This dataset is generated entirely from public Steam data.

[1]:
```python
#Data manipulation imports
import numpy as np
import pandas as pd

#Graphing imports
import plotly.express as px
import plotly.io as pio
import plotly.graph_objects as go
pio.renderers.default = "notebook_connected+pdf"

#AI imports
import tensorflow.compat.v1 as tf
tf.disable_v2_behavior()

import warnings
warnings.filterwarnings('ignore')



import custom_theme #need to have custom_theme.py file in same directory
```

```
WARNING:tensorflow:From
C:\Users\Vikram\AppData\Local\Temp\ipykernel_32248\3314047346.py:13: The name
tf.disable_v2_behavior is deprecated. Please use
tf.compat.v1.disable_v2_behavior instead.
```

```
WARNING:tensorflow:From c:\Users\Vikram\anaconda3\envs\the_vault_env\lib\site-
packages\tensorflow\python\compat\v2_compat.py:98: disable_resource_variables
(from tensorflow.python.ops.resource_variables_toggle) is deprecated and will be
removed in a future version.
Instructions for updating:
non-resource variables are not supported in the long term
WARNING:tensorflow:From c:\Users\Vikram\anaconda3\envs\the_vault_env\lib\site-
packages\tensorflow\python\compat\v2_compat.py:98: disable_resource_variables
(from tensorflow.python.ops.resource_variables_toggle) is deprecated and will be
removed in a future version.
Instructions for updating:
non-resource variables are not supported in the long term
```

[2]:
```python
#Reading the CSV file using pandas

steam_raw = pd.read_csv("gamedata.
↪csv",usecols=[0,1,2,3],names=['userid','game','behavior','hoursplayed'])
steam_raw.head()
```

[2]:

	userid	game	behavior	hoursplayed
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

[3]:
```python
#Checking if any null values are present in any row and column

steam_raw.isnull().values.any()
```

[3]: False

[4]:
```python
#Converting the 'userid' column's values to string

steam_raw['userid'] = steam_raw.userid.astype(str)
```

[5]:
```python
steam_raw.describe()
```

[5]:

	hoursplayed
count	200000.000000
mean	17.874384
std	138.056952
min	0.100000
25%	1.000000
50%	1.000000
75%	1.300000
max	11754.000000

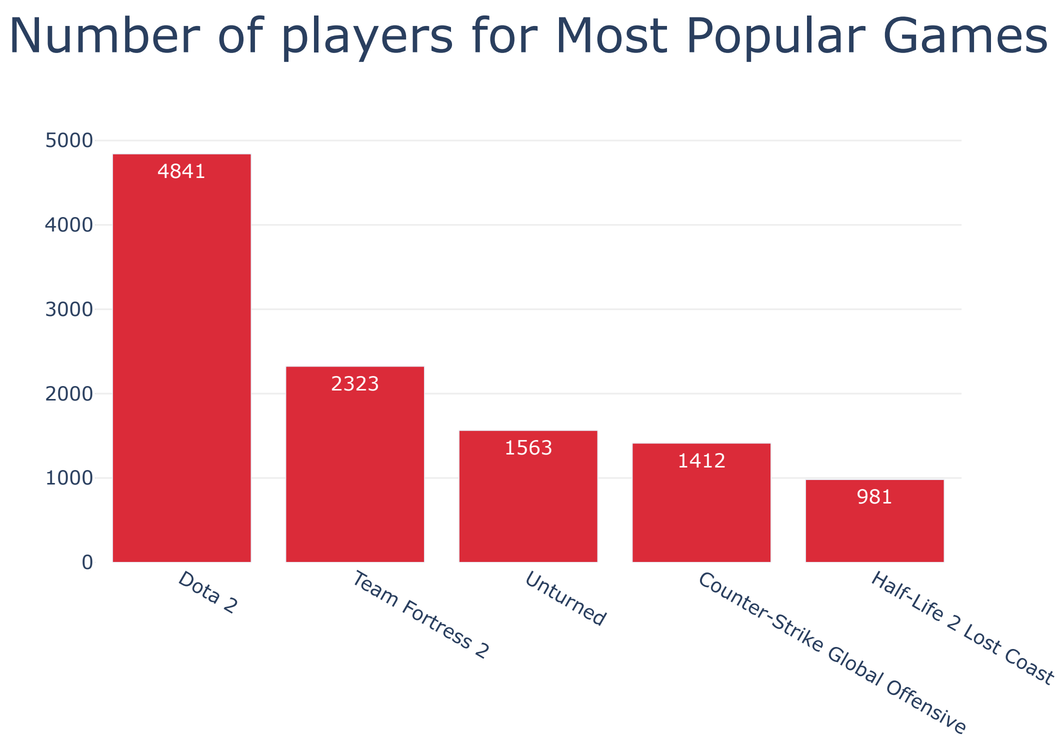
[6]:
```python
len(steam_raw['game'].unique()), len(steam_raw['userid'].unique())
```

[6]: (5155, 12393)

There are 5155 unique games and 12393 unique players in the dataset.

[7]:
```python
#Setting custom plotting theme as default

def set_theme():
    pio.templates['my_theme'] = custom_theme.my_theme
    pio.templates.default = 'my_theme'

set_theme()
# config = {'displayModeBar':False}
```

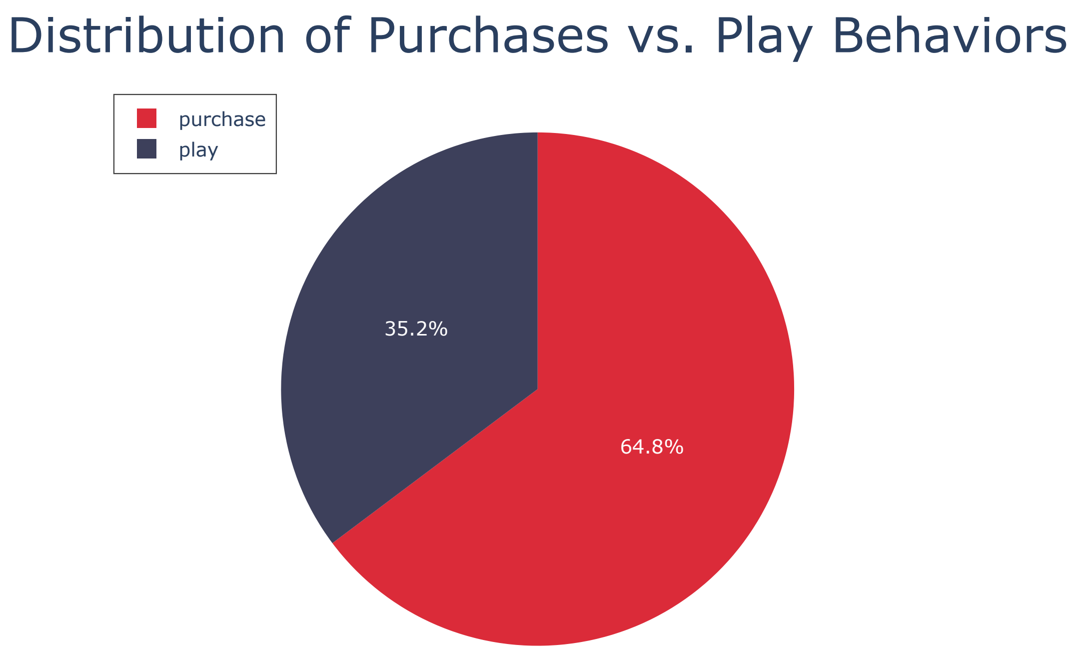
[8]:
```python
# Grouping the data to get unique user count per game
gb = steam_raw.groupby('game')['userid'].nunique().sort_values(ascending=False).
 ↪head()

# Convert to DataFrame for Plotly compatibility
gb_df = gb.reset_index(name='No. of players')

# Create the bar plot
fig = px.bar(gb_df,
             x='game',
             y='No. of players',
             title='Number of players for Most Popular Games',
             labels={'game': 'Game', 'No. of players': 'No. of players'},
             text='No. of players')  # Adds labels to bars

# Update layout for better visualization
fig.update_layout(xaxis_title='Game',
                  yaxis_title='No. of players')

# Show the figure
fig.show()
```



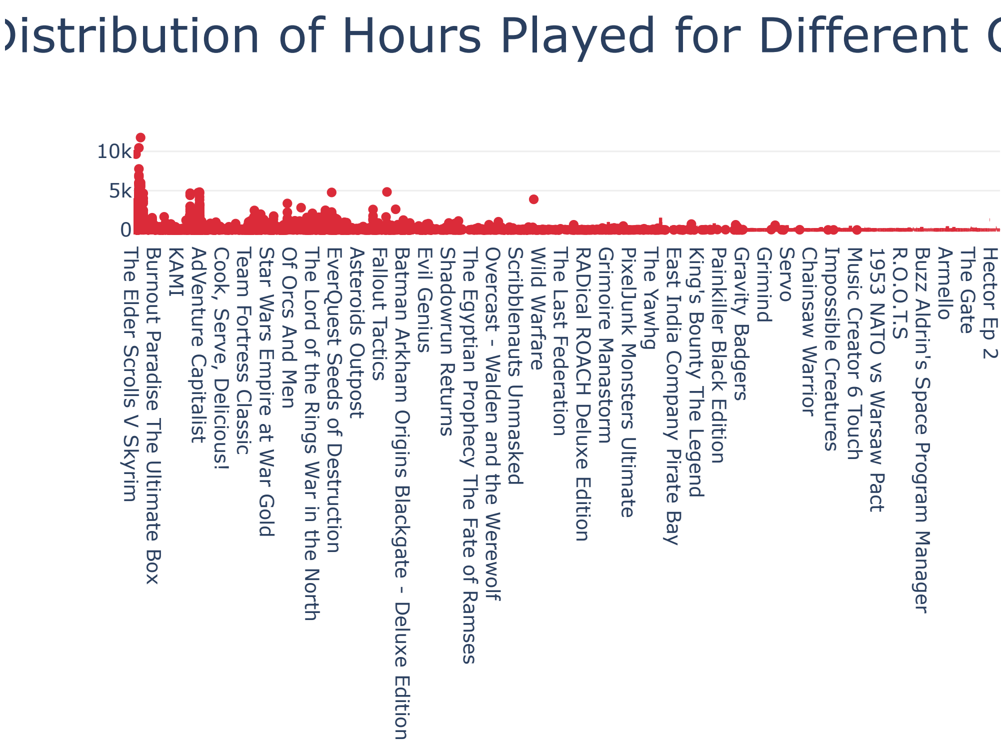
[9]:
```
steam_raw
```

[9]:
```
           userid                        game  behavior  hoursplayed
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
…             …                           …       …            …
199995  128470551                 Titan Souls      play          1.5
199996  128470551  Grand Theft Auto Vice City  purchase          1.0
199997  128470551  Grand Theft Auto Vice City      play          1.5
199998  128470551                        RUSH  purchase          1.0
199999  128470551                        RUSH      play          1.4

[200000 rows x 4 columns]
```

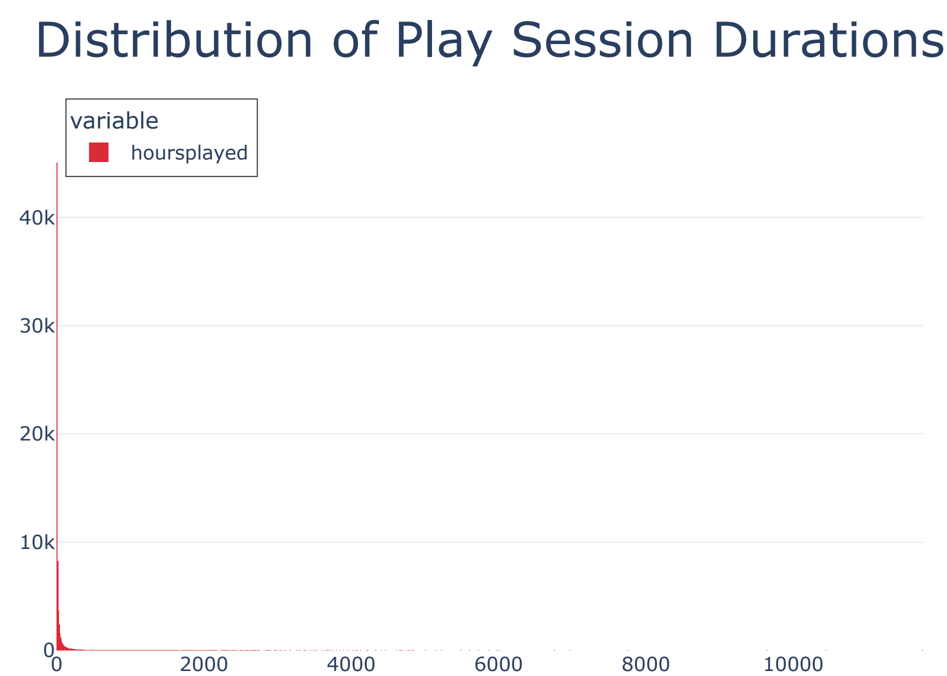
[10]:
```
#Pie chart showing distribution of purchases vs. play behaviors
behavior_counts = steam_raw['behavior'].value_counts()
fig2 = px.pie(values=behavior_counts.values, names=behavior_counts.index,
    title='Distribution of Purchases vs. Play Behaviors')
fig2.show()
```



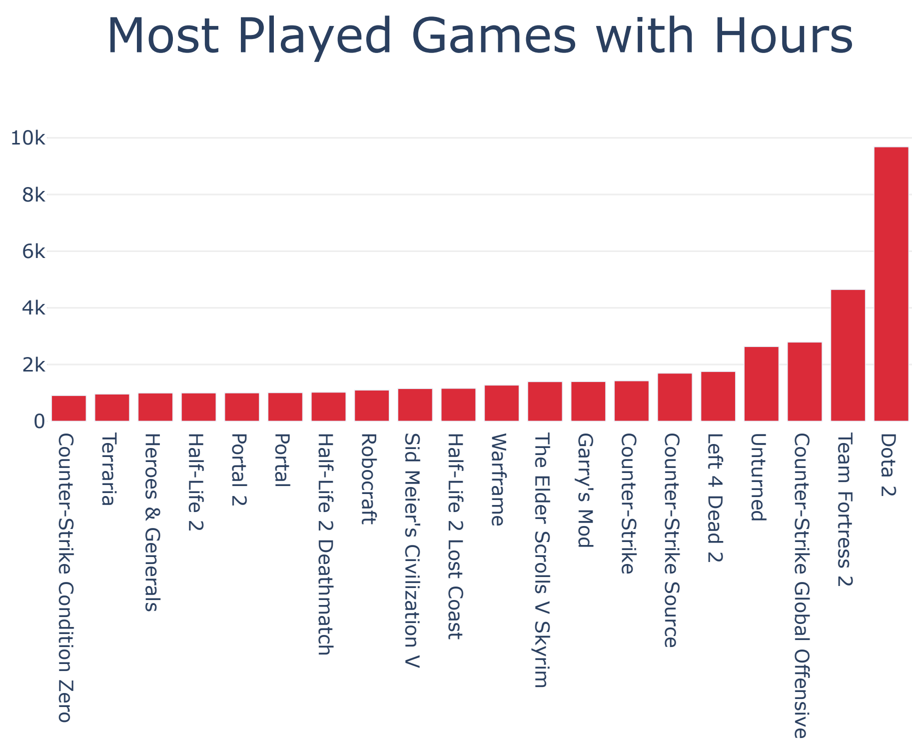
[11]:
```
#Box plot of hours played for different games
fig4 = px.box(steam_raw[steam_raw['behavior'] == 'play'], x='game',␣
↪y='hoursplayed', title='Distribution of Hours Played for Different Games')
fig4.show()
```



[12]:
```python
#Histogram of play session durations
play_sessions = steam_raw[steam_raw['behavior'] == 'play']['hoursplayed']
fig6 = px.histogram(play_sessions, title='Distribution of Play Session␣
  ↪Durations')
fig6.show()
```



[13]:
```
a = steam_raw.groupby('game').count().reset_index()
px.bar(a.sort_values('hoursplayed').tail(20), x='game', y='hoursplayed',␣
 ↪title='Most Played Games with Hours')
```



1.1 Feature Engineering and Metrics

Supposedly if a user plays a game for more than 40 hours, then the user enjoys the game. Thus, we define a binary column "like" that indicates 1 if the user enjoys the game, and 0 if he/she doesn't.

[14]:
```
steam_df = steam_raw.copy()
steam_df['like'] = [1 if x > 40 else 0 for x in steam_df['hoursplayed']]
steam_df['like'].value_counts()
```

[14]:
```
like
0    189067
1     10933
Name: count, dtype: int64
```

[15]:
```
steam_df.head()
```

[15]:

	userid	game	behavior	hoursplayed	like
0	151603712	The Elder Scrolls V Skyrim	purchase	1.0	0
1	151603712	The Elder Scrolls V Skyrim	play	273.0	1
2	151603712	Fallout 4	purchase	1.0	0
3	151603712	Fallout 4	play	87.0	1

```
4  151603712                        Spore  purchase          1.0     0
```

[16]:
```
bg=steam_df.groupby('game')['like'].apply(lambda x: (x==1).sum()).
 ↪sort_values(ascending=False)
bg.head()
```

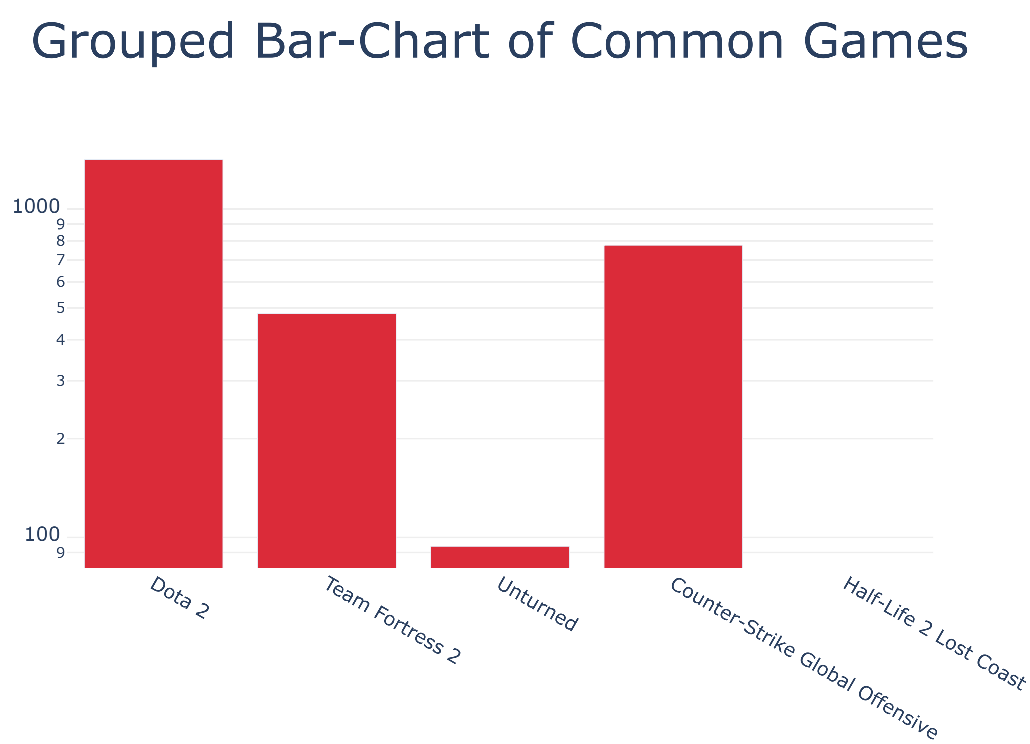
[16]:
```
game
Dota 2                                1417
Counter-Strike Global Offensive        776
Team Fortress 2                        480
The Elder Scrolls V Skyrim             362
Sid Meier's Civilization V             265
Name: like, dtype: int64
```

[17]:
```
gb.head()
```

[17]:
```
game
Dota 2                                4841
Team Fortress 2                       2323
Unturned                              1563
Counter-Strike Global Offensive       1412
Half-Life 2 Lost Coast                 981
Name: userid, dtype: int64
```

[18]:
```
#Plot grouped bar-chart of common games
gbbg = pd.merge(gb, bg, on='game')

# Plotly grouped bar chart
fig = go.Figure()

# Add bar traces for each column in the merged DataFrame
for column in gbbg.columns[1:]:  # Skip 'game' column for x-axis
    fig.add_trace(go.Bar(
        x=gbbg.index,
        y=gbbg[column],
        name=column
    ))

# Set the layout and enable logarithmic scale for y-axis
fig.update_layout(
    barmode='group',  # Grouped bars
    title='Grouped Bar-Chart of Common Games',
    xaxis_title='Game',
    yaxis_title='Values',
    yaxis_type='log',  # Set y-axis to log scale
)
```

```
# Show the figure
fig.show()
```



From the graph, Half-Life 2 Lost Coast had one of the highest unique players of 981 (purchased and played) but none of them played the game more than 40 hours. Now, let's find those who purchased a game and didn't play it at all. We would want to reassign hoursplayed for these players to 0 instead of 1. And change the behavior to play and finally drop rows that are purchase. This would leave the dataframe to only containing play behaviors and if those that are purchased and not played, the hoursplayed will be 0.

[19]:
```
x = steam_df.groupby(['userid', 'game'])['behavior'].size()
s = x[x == 1]

len(s), len(x)
```

[19]:
```
(57904, 128804)
```

[20]:
```
boolean_index = steam_df.groupby(['userid','game'])['behavior'].
 ↪transform('size') < 2
steam_df.loc[boolean_index,'hoursplayed'] = 0
steam_df.loc[steam_df['hoursplayed']==0]
```

[20]:

```
           userid                                         game  behavior  \
52      151603712                                    Alan Wake  purchase
53      151603712                                  BioShock 2  purchase
54      151603712                                Fallen Earth  purchase
55      151603712           Fallout New Vegas Courier's Stash  purchase
56      151603712                Fallout New Vegas Dead Money  purchase
…             …                                          …       …
199947   99096740       The Elder Scrolls V Skyrim - Hearthfire  purchase
199956  176449171                               Counter-Strike  purchase
199957  176449171                Counter-Strike Condition Zero  purchase
199958  176449171  Counter-Strike Condition Zero Deleted Scenes  purchase
199959  176449171                        Counter-Strike Source  purchase

        hoursplayed  like
52              0.0     0
53              0.0     0
54              0.0     0
55              0.0     0
56              0.0     0
…                …   …
199947          0.0     0
199956          0.0     0
199957          0.0     0
199958          0.0     0
199959          0.0     0

[57904 rows x 5 columns]
```

[21]:

```python
steam_df.loc[steam_df.hoursplayed==0,'behavior'] = 'play'
steam_df.loc[steam_df['hoursplayed'] ==0]
```

[21]:

```
           userid                                         game behavior  \
52      151603712                                    Alan Wake     play
53      151603712                                  BioShock 2     play
54      151603712                                Fallen Earth     play
55      151603712           Fallout New Vegas Courier's Stash     play
56      151603712                Fallout New Vegas Dead Money     play
…             …                                          …      …
199947   99096740       The Elder Scrolls V Skyrim - Hearthfire     play
199956  176449171                               Counter-Strike     play
199957  176449171                Counter-Strike Condition Zero     play
199958  176449171  Counter-Strike Condition Zero Deleted Scenes     play
199959  176449171                        Counter-Strike Source     play

        hoursplayed  like
52              0.0     0
53              0.0     0
```

```
54                0.0     0
55                0.0     0
56                0.0     0
…               …   …
199947            0.0     0
199956            0.0     0
199957            0.0     0
199958            0.0     0
199959            0.0     0

[57904 rows x 5 columns]
```

[22]:
```
steam_df = steam_df[steam_df.behavior != 'purchase']
```

There are 57904 games purchased that have not been played yet. Next, we define the metrics to calculate a simple recommendation based on popularity and what other players like.

[23]:
```
# Create a new dataframe to store metrics
d = {'like':'Sum Likes','hoursplayed':'Avg Hours Played'}
metrics_df = steam_df.groupby(['game'], as_index=False).agg({'like':
 ↪'sum','hoursplayed':'mean'}).rename(columns=d)
metrics_df.loc[metrics_df['game'] == "Dota 2"] #Check Dota 2
```

[23]:
```
        game  Sum Likes  Avg Hours Played
1336  Dota 2       1417        202.785499
```

[24]:
```
# Calculate mean of Hours Played average
c = metrics_df['Avg Hours Played'].mean()
print("Average hours played across all games is " + str(round(c,2)))
```

```
Average hours played across all games is 6.78
```

[25]:
```
# Calculate the minimum number of likes required, set to 95 percentile
m = metrics_df['Sum Likes'].quantile(0.95)
print("Minimum number of likes for a game is " + str(m))
```

```
Minimum number of likes for a game is 5.0
```

Here the cut-off for the minimum number of likes is 5, this mean that there should be at least 5 user that played the game for more than 40 hours. If a game has no more than 5 likes, we wouldn't recommend it to others. Now, we can proceed to trim and filter out the dataframe that meet this minimum number of likes.

[26]:
```
metrics_df.shape
```

[26]:
```
(5155, 3)
```

[27]:
```
metrics_df = metrics_df.loc[metrics_df['Sum Likes'] >= m]
metrics_df.shape
```

[27]: (266, 3)

[28]:
```
metrics_df.head()
```

[28]:
```
                          game  Sum Likes  Avg Hours Played
38               7 Days to Die         22         39.567961
81                APB Reloaded         17         35.256489
84         ARK Survival Evolved         61         83.393252
109        AdVenture Capitalist         33         27.331982
174  Age of Empires II HD Edition         33         28.817227
```

1.2 Simple Recommender

Next, we will create the scoring system for each game. Define the score as Average Hours Played for the Game multiplied by Sum Likes Fraction Add Average Hours Across Games multilpied by minimum number of Likes Fraction

[29]:
```
def weighted_rating(df, m=m, C=c):
    l = df['Sum Likes']
    a = df['Avg Hours Played']
    return (l/(l+m) * a) + (m/(l+m) * C)

metrics_df['score'] = metrics_df.apply(weighted_rating, axis=1)
metrics_df.head()
```

[29]:
```
                          game  Sum Likes  Avg Hours Played      score
38               7 Days to Die         22         39.567961  33.495568
81                APB Reloaded         17         35.256489  28.783886
84         ARK Survival Evolved         61         83.393252  77.588993
109        AdVenture Capitalist         33         27.331982  24.627384
174  Age of Empires II HD Edition         33         28.817227  25.917202
```

[30]:
```
metrics_df.sort_values(by=['score'],ascending=False).head(15)
```

[30]:
```
                              game  Sum Likes  Avg Hours Played  \
1762          Football Manager 2012         64        385.572500
1764          Football Manager 2014         60        382.185000
1763          Football Manager 2013         77        310.659615
1760          Football Manager 2010         23        345.439474
1765          Football Manager 2015         58        307.381013
1761          Football Manager 2011         24        333.435294
981     Counter-Strike Global Offensive        776        228.591785
1336                            Dota 2       1417        202.785499
1620  FINAL FANTASY XIV A Realm Reborn          9        264.740000
3825         Sid Meier's Civilization V        265        167.485403
```

1559	Europa Universalis IV	24	187.673077
978	Counter-Strike	191	156.847079
2807	Mount & Blade Warband	52	158.744615
329	Arma 3	77	149.414286
3271	Pro Evolution Soccer 2015	8	208.375000

	score
1762	358.123553
1764	353.307464
1763	292.130190
1760	284.964039
1765	283.523554
1761	277.114905
981	227.171716
1336	202.096299
1620	172.610371
3825	164.509322
1559	156.484105
978	153.018762
2807	145.414126
329	140.716893
3271	130.837322

Using the Simple Recommender score, the top games are

1. Football Manager,
2. CSGO,
3. and Dota 2.

This yields the most popular games/games that are well-liked by others.

1.3 Restricted Boltzman Machine

Develop RBM a stochastic ANN to generate construct recommendations.

[31]:
```
steam_df
```

[31]:

	userid	game	behavior	hoursplayed	like
1	151603712	The Elder Scrolls V Skyrim	play	273.0	1
3	151603712	Fallout 4	play	87.0	1
5	151603712	Spore	play	14.9	0
7	151603712	Fallout New Vegas	play	12.1	0
9	151603712	Left 4 Dead 2	play	8.9	0
…	…	…	…	…	…
199991	128470551	Fallen Earth	play	2.4	0
199993	128470551	Magic Duels	play	2.2	0
199995	128470551	Titan Souls	play	1.5	0
199997	128470551	Grand Theft Auto Vice City	play	1.5	0
199999	128470551	RUSH	play	1.4	0

```
[128393 rows x 5 columns]
```

[32]:
```python
len(steam_df['game'].unique()), len(steam_df['userid'].unique()), len(steam_df)
```

[32]:
```
(5155, 12392, 128393)
```

[33]:
```python
games_df = pd.DataFrame(steam_df.game.unique(), columns=['game'])
games_df['index_col'] = games_df.index
games_df
```

[33]:
```
                                     game  index_col
0             The Elder Scrolls V Skyrim          0
1                              Fallout 4          1
2                                  Spore          2
3                      Fallout New Vegas          3
4                          Left 4 Dead 2          4
…                                      …          …
5150                   Warriors & Castles       5150
5151  Romance of the Three Kingdoms Maker       5151
5152                         Space Colony       5152
5153                         Life is Hard       5153
5154                    Executive Assault       5154

[5155 rows x 2 columns]
```

[34]:
```python
steam_df = steam_df.merge(games_df, on='game')
steam_df.head()
```

[34]:
```
      userid                        game behavior  hoursplayed  like  \
0  151603712  The Elder Scrolls V Skyrim     play        273.0     1
1  151603712                   Fallout 4     play         87.0     1
2  151603712                       Spore     play         14.9     0
3  151603712           Fallout New Vegas     play         12.1     0
4  151603712               Left 4 Dead 2     play          8.9     0

   index_col
0          0
1          1
2          2
3          3
4          4
```

[35]:
```python
steam_df['hoursplayed'].std()
steam_df['hoursplayed'].mean()
```

[35]:
```
26.834529919855445
```

[36]:
```
usergroup = steam_df.groupby('userid')
usergroup.head()
```

[36]:
```
           userid                         game behavior  hoursplayed  like  \
0       151603712   The Elder Scrolls V Skyrim     play        273.0     1
1       151603712                    Fallout 4     play         87.0     1
2       151603712                        Spore     play         14.9     0
3       151603712            Fallout New Vegas     play         12.1     0
4       151603712                Left 4 Dead 2     play          8.9     0
…             …                            …       …            …     …
128377  128470551  The Binding of Isaac Rebirth     play        291.0     1
128378  128470551                Path of Exile     play         42.0     1
128379  128470551              Arma 2 DayZ Mod     play         22.0     0
128380  128470551                  Antichamber     play         16.8     0
128381  128470551                 Risk of Rain     play         15.4     0

        index_col
0               0
1               1
2               2
3               3
4               4
…             …
128377        500
128378          6
128379        279
128380        292
128381        246

[32437 rows x 6 columns]
```

[37]:
```
noOfUsers = 1000

train_list = []

i = 0
# For each user in the group
for userID, cur in usergroup:
    # Create a temp that stores every game's hours played
    temp = [0]*len(games_df)
    # For each game in list
    for no, game in cur.iterrows():
        temp[game['index_col']] = game['hoursplayed']
        i+=1
    train_list.append(temp)

    if noOfUsers == 0:
```

```
            break
        noOfUsers -= 1
```

[38]:
```
# Setting the models Parameters
hiddenUnits = 50
visibleUnits = len(steam_raw['game'].unique())
vb = tf.placeholder(tf.float32, [visibleUnits])
hb = tf.placeholder(tf.float32, [hiddenUnits])
W = tf.placeholder(tf.float32, [visibleUnits, hiddenUnits])

# Phase 1: Input Processing
v0 = tf.placeholder("float", [None, visibleUnits])
_h0 = tf.nn.sigmoid(tf.matmul(v0, W) + hb)
h0 = tf.nn.relu(tf.sign(_h0 - tf.random_uniform(tf.shape(_h0))))

# Phase 2: Reconstruction
_v1 = tf.nn.sigmoid(tf.matmul(h0, tf.transpose(W)) + vb)
v1 = tf.nn.relu(tf.sign(_v1 - tf.random_uniform(tf.shape(_v1))))
h1 = tf.nn.sigmoid(tf.matmul(v1, W) + hb)

# Learning rate
alpha = 1

# Create the gradients
w_pos_grad = tf.matmul(tf.transpose(v0), h0)
w_neg_grad = tf.matmul(tf.transpose(v1), h1)

# Calculate the Contrastive Divergence to maximize
CD = (w_pos_grad - w_neg_grad) / tf.to_float(tf.shape(v0)[0])

# Create methods to update the weights and biases
update_w = W + alpha * CD
update_vb = vb + alpha * tf.reduce_mean(v0 - v1, 0)
update_hb = hb + alpha * tf.reduce_mean(h0 - h1, 0)

# Set the error function, here we use Mean Absolute Error Function
err = v0 - v1
err_sum = tf.reduce_mean(err*err)

err_sum
```

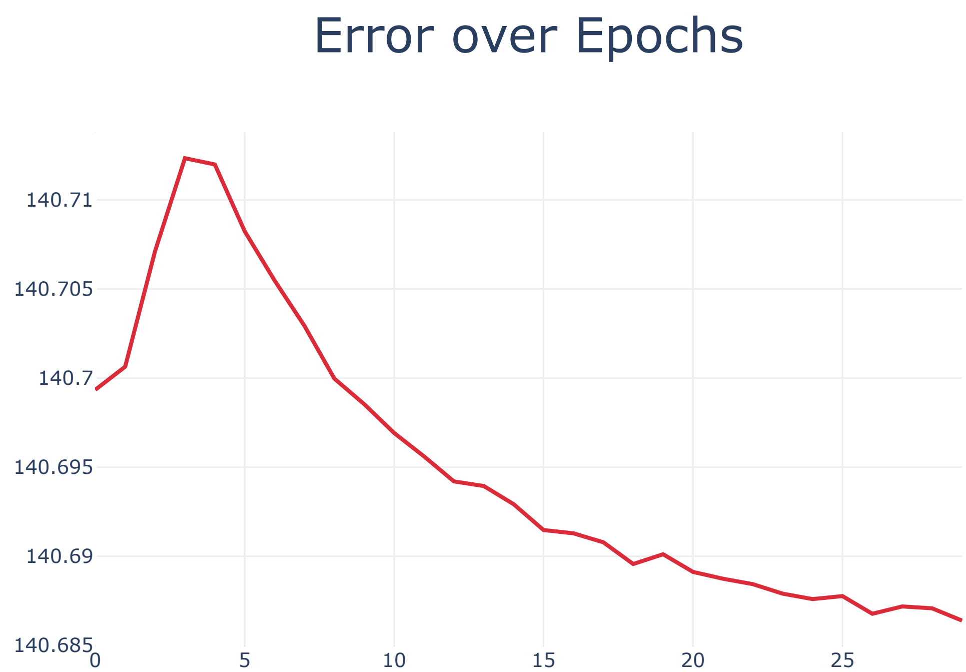
```
WARNING:tensorflow:From c:\Users\Vikram\anaconda3\envs\the_vault_env\lib\site-
packages\tensorflow\python\util\dispatch.py:1260: to_float (from
tensorflow.python.ops.math_ops) is deprecated and will be removed in a future
version.
Instructions for updating:
Use `tf.cast` instead.
```

[38]: <tf.Tensor 'Mean_2:0' shape=() dtype=float32>

[39]:
```
# Initialize variables
cur_w = np.zeros([visibleUnits, hiddenUnits], np.float32)
cur_vb = np.zeros([visibleUnits], np.float32)
cur_hb = np.zeros([hiddenUnits], np.float32)
prv_w = np.zeros([visibleUnits, hiddenUnits], np.float32)
prv_vb = np.zeros([visibleUnits], np.float32)
prv_hb = np.zeros([hiddenUnits], np.float32)

# Create a TensorFlow session and initialize global variables
sess = tf.Session()
sess.run(tf.global_variables_initializer())

# Parameters
epochs = 30
batchsize = 150
errors = []

# Training loop
for i in range(epochs):
    for start, end in zip(range(0, len(train_list), batchsize),␣
 ↪range(batchsize, len(train_list), batchsize)):
        batch = train_list[start:end]
        cur_w = sess.run(update_w, feed_dict={v0: batch, W: prv_w, vb: prv_vb,␣
 ↪hb: prv_hb})
        cur_vb = sess.run(update_vb, feed_dict={v0: batch, W: prv_w, vb:␣
 ↪prv_vb, hb: prv_hb})
        cur_hb = sess.run(update_hb, feed_dict={v0: batch, W: prv_w, vb:␣
 ↪prv_vb, hb: prv_hb})
        prv_w = cur_w
        prv_vb = cur_vb
        prv_hb = cur_hb

    # Append errors for each epoch
    errors.append(sess.run(err_sum, feed_dict={v0: train_list, W: cur_w, vb:␣
 ↪cur_vb, hb: cur_hb}))
    print(errors[-1])

# Plot errors using Plotly
fig = go.Figure()

# Add a line plot for errors over epochs
fig.add_trace(go.Scatter(x=list(range(epochs)), y=errors, mode='lines',␣
 ↪name='Error'))

# Update layout to add titles and axis labels
```

```python
fig.update_layout(
    title='Error over Epochs',
    xaxis_title='Epoch',
    yaxis_title='Error',
)

# Show the figure
fig.show()
```

```
140.69934
140.70064
140.70712
140.71234
140.71199
140.70824
140.70547
140.70293
140.69997
140.69853
140.69691
140.6956
140.6942
140.69394
140.69292
140.69147
140.69128
140.69078
140.68956
140.69011
140.68912
140.68874
140.68843
140.6879
140.68759
140.68776
140.68677
140.68718
140.68707
140.68639
```



[40]:

```
# Select mock user input
inputUser = [train_list[150]]

# Compute hidden and visible layer activations
hh0 = tf.nn.sigmoid(tf.matmul(v0, W) + hb)
vv1 = tf.nn.sigmoid(tf.matmul(hh0, tf.transpose(W)) + vb)

# Run session to calculate hidden activations and reconstructed input
feed = sess.run(hh0, feed_dict={v0: inputUser, W: prv_w, hb: prv_hb})
rec = sess.run(vv1, feed_dict={hh0: feed, W: prv_w, vb: prv_vb})

# Add recommendation scores and show top 10 games
inputuser_games = games_df
inputuser_games["Recommendation Score"] = rec[0]
inputuser_games.sort_values(["Recommendation Score"], ascending=False).head(10)

# Get user ID and find the games they have played
userid = steam_df.iloc[150]['userid']
muser_df = steam_df.loc[(steam_df['userid'] == userid) &␣
 ↪(steam_df['hoursplayed'] > 0)]
```

```
muser_df
```

[40]:

```
        userid                               game behavior  hoursplayed  \
84   53875128                   Grand Theft Auto V     play         86.0
85   53875128                           Insurgency     play         72.0
86   53875128                        Left 4 Dead 2     play         71.0
87   53875128  METAL GEAR SOLID V THE PHANTOM PAIN     play         59.0
88   53875128   S.T.A.L.K.E.R. Shadow of Chernobyl     play         54.0
..        …                                    …        …            …
276  53875128                  Metro Last Light Redux     play          0.1
277  53875128       Crimzon Clover  WORLD IGNITION     play          0.1
278  53875128                    Sonic Generations     play          0.1
279  53875128                  Ethan Meteor Hunter     play          0.1
280  53875128                                 Reus     play          0.1

     like  index_col
84      1         75
85      1         76
86      1          4
87      1         77
88      1         78
..    …         …
276     0        254
277     0        255
278     0        256
279     0        257
280     0        258

[197 rows x 6 columns]
```

[42]:

```
#Doing a left merge
df_all = inputuser_games.merge(muser_df, how='left', indicator=True)
unplayed_games = df_all[df_all['_merge']=='left_only']

#Any Top 5 recommended games for input user which he hasn't played
unplayed_games.loc[:,['game','Recommendation Score']].
↪sort_values(['Recommendation Score'], ascending=False).head(5)
```

[42]:

```
                                  game  Recommendation Score
0              The Elder Scrolls V Skyrim                   1.0
1226       Medieval II Total War Kingdoms                   1.0
995                                Arma 3                   1.0
1017  Total War ROME II - Emperor Edition                   1.0
1022                   XCOM Enemy Unknown                   1.0
```

The top 5 recommended games for this user are The Elder Scrolls V Skyrim, Warframe, Arma 3, Counter-Strike and APB Reloaded.