Sumedh Kumar Prasad Mini project (Data science using Python)

Part - 1

We shall now test your skills in using Pandas package. We will be using the games Dataset (https://www.kaggle.com/gutsyrobot/games-data/data) from Kaggle.

Answer each question asked below wrt the games dataset.

Import pandas as pd.

In [2]:
```
import numpy as np
import pandas as pd
```

Read games.csv as a dataframe called games.

In [3]:
```
df = pd.read_csv('game1.csv')
```

Check the head of the DataFrame.

In [4]:
```
df.head()
```

Out[4]:

	id	type	name	yearpublished	minplayers	maxplayers	playingtime	minpla
0	1	boardgame	Twilight Struggle	2005	2	2	180	180
1	2	boardgame	Terra Mystica	2012	2	5	150	60
2	3	boardgame	Caverna: The Cave Farmers	2013	1	7	210	30
3	4	boardgame	Through the Ages: A Story of Civilization	2006	2	4	240	240
4	5	boardgame	Puerto Rico	2002	2	5	150	90

Use .info() method to find out total number of entries in dataset

In [5]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2665 entries, 0 to 2664
Data columns (total 20 columns):
id                      2665 non-null int64
type                    2665 non-null object
name                    2665 non-null object
yearpublished           2665 non-null int64
minplayers              2665 non-null int64
maxplayers              2665 non-null int64
playingtime             2665 non-null int64
minplaytime             2665 non-null int64
maxplaytime             2665 non-null int64
minage                  2665 non-null int64
users_rated             2665 non-null int64
average_rating          2665 non-null float64
bayes_average_rating    2665 non-null float64
total_owners            2665 non-null int64
total_traders           2665 non-null int64
total_wanters           2665 non-null int64
total_wishers           2665 non-null int64
total_comments          2665 non-null int64
total_weights           2665 non-null int64
average_weight          2665 non-null float64
dtypes: float64(3), int64(15), object(2)
memory usage: 416.5+ KB
```

In this data set there is 20 Columns and 2665 rows is there.

What is the mean playin time for all games put together ?

In [6]:
```
df.playingtime.mean()
```

Out[6]:
```
105.03489681050657
```

Mean playing time for all the games is 105.034869 units.

What is the highest number of comments received for a game?

In [7]:
```
df[df.total_comments==df.total_comments.max()][['name','total_comments']]
```

Out[7]:

	name	total_comments
165	Catan	11798

In [8]:
```
df[df.total_comments==df.total_comments.max()]['total_comments']
```

Out[8]:
```
165    11798
Name: total_comments, dtype: int64
```

Catan Game received most number of comments having 11798

What is the name of the game with id 1500?

In [9]:
```
df[df.id==1500][['name','id']]
```

Out[9]:

	name	id
1499	El Grande: Knig & Intrigant	1500

El Grande: Knig & Intrigant game having id = 1500

And which year was it published?

In [10]:
```
df[df.id==1500][['id','name','yearpublished']]
```

Out[10]:

	id	name	yearpublished
1499	1500	El Grande: Knig & Intrigant	1997

El Grande: Knig & Intrigant game having id = 1500 had published in 1997

Which game has received highest number of comments?

In [11]:
```
df[df.total_comments==df.total_comments.max()][['name','total_comments']]
```

Out[11]:

	name	total_comments
165	Catan	11798

Caten game received maximum number of comments

Which games have received least number of comments?

In [12]:
```
df[df.total_comments==df.total_comments.min()][['name','total_comments']]
```

Out[12]:

	name	total_comments
1572	Fire On The Suns: Tactical Command Fleet Book 1	0
1584	Hossa! Seefahrer-Erweiterung	0
1665	Worlds of Heroes & Tyrants: Hell Card Expansio...	0
1750	Aura Battler Dunbine: Wing Caliver	0
1751	Mobile Suit Z Gundam: Gate of Zedan	0
1752	Mobile Suit Gundam: White Base	0
1753	L-Gaim Mark II	0
1768	Lemlican Series 3: Arms Collection	0
1769	Lemlican Series 2: Lemlican Monsters	0
1770	Wizardry Card Game Wiz Ball Expansion Kit	0
1772	Gal Master 2	0
1776	Super Nova: Event Horizon	0
1785	Aventuras Hericas: A Revelao da Princesa / O E...	0
1811	Speed Circuit Tournament Tracks	0
1889	Sturmovik, Clash of Eagles Expansion	0
1933	Forte Trivia Cards Volume II	0
1934	Zeppelin	0
1961	Love, Sex and Romance Trivia Card Set	0
1966	Trivia Adventure Plus	0
1989	Mayhem	0
2065	Entrepreneur's Accessory to Monopoly	0
2092	Crusade	0
2098	Starfleet Wars: Observer's Directory & Identif...	0
2100	Suomi: A module for Clash of Eagles	0
2142	Bar-barians!	0
2169	Carnage: Skullbrawl	0
2173	Horsepower: 5000 Expansion Set 1 World Rails	0
2183	Alexander	0
2188	Mobile Suit Gundam ZZ: Core 3	0
2191	Renegade Legion: Interceptor The Fire Eagles	0
...	...	...
2335	Napoleonic Scenarios Volume 2	0

	name	total_comments
2345	No Pasaran!: La Quinta del Biberon Balaguer 1938	0
2383	Cranium New York Booster Pack	0
2394	Floaties & Sinkies!	0
2397	Hossa!: Arbeiterlieder	0
2429	Meck Wars	0
2430	Train Raider: Europe Expansion	0
2448	Tank War: Expansion A	0
2455	Avanti Savoia: Intelligence Handbook on Italia...	0
2545	History of War: "Fall Gelb" Edition	0
2547	Vampire Wars: The Antagonists	0
2557	Krebiz-4	0
2606	Overlords Monsters Packs	0
2607	Piglings Revenge, PigWars Expansion Module #1	0
2621	Mad Scientist University Course Packet: Anthro...	0
2622	Mad Scientist University Course Packet: Chemistry	0
2623	Mad Scientist University Course Packet: Astronomy	0
2624	Mad Scientist University Course Packet: Computers	0
2625	Mad Scientist University Course Packet: Dorm Life	0
2626	Mad Scientist University Course Packet: Greek ...	0
2627	Mad Scientist University Course Packet: Physic...	0
2628	Mad Scientist University Course Packet: Medicine	0
2629	Mad Scientist University Course Packet: Physics	0
2631	Mad Scientist University Course Packet: Theater	0
2632	Mad Scientist University Course Packet: Winter...	0
2633	Mad Scientist University Course Packet: Zoology	0
2635	Mad Scientist University Course Packet: PSI PHI	0
2636	Mad Scientist University Course Packet: Indepe...	0
2637	Mad Scientist University Course Packet: Home E...	0
2649	La die Kirche ins Dorf Erweiterung	0

65 rows × 2 columns

Above listed games had least or zero comments in it .

This o/p show both game list and minimum number of comments

What was the average minage of all games per game "type"? (boardgame & boardgameexpansion)

```
In [13]: df.groupby('type').minage.mean()
```

```
Out[13]: type
         boardgame            10.718499
         boardgameexpansion    10.633419
         Name: minage, dtype: float64
```

How many unique games are there in the dataset?

```
In [14]: df.name.nunique()
```

```
Out[14]: 2657
```

there are 2657 unique game is there in dataset

How many boardgames and boardgameexpansions are there in the dataset?

```
In [15]: df.type.value_counts()
```

```
Out[15]: boardgame            1492
         boardgameexpansion    1173
         Name: type, dtype: int64
```

for broadgame 1492 datasets is there and for broadgamexpansion 1173 dataset is there

Is there a correlation between playing time and total comments for the games? - Use the .corr() function

```
In [16]: df2=df[['playingtime','total_comments']]
```

```
In [17]: correlation = df2.corr(method='pearson')
```

```
In [18]: print(correlation)
```

```
                 playingtime  total_comments
playingtime         1.000000       -0.010367
total_comments     -0.010367        1.000000
```

Correlation coefficient between playing time and total comments = -0.010367, there is no correlation between playing time and total comments.

Part 2

Inferential Statistical Analysis

A Brand new Gaming Design Company has entered the Market in year 2015. They are interested in Analyzing the Amount of Time Gamers spend for each Game Type and check if the players spend equal Time on both Types.

Analyze the same using a suitable Two Sample Test at 95% Confidence.

The following points are necessarily to be adhered.

- Clean Data
- Transform Data(as required)
- Visualize Data(Draw as many patterns, not limited to Histograms & Boxplots only) - Minimum Requirement of 5 different Visualizations apart from Histograms & Boxplots to be created.
- Model the Data with the correct hypothesis statement
- Conclude on your results based on the outcome of experiment
- Calculate power of test and interpret the same

The END

In [19]:
```
df.describe()
```

Out[19]:

	id	yearpublished	minplayers	maxplayers	playingtime	minplaytim
count	2665.000000	2665.000000	2665.000000	2665.000000	2665.000000	2665.00000
mean	1333.000000	1982.877298	2.056285	5.258537	105.034897	94.924953
std	769.463558	209.866182	0.721703	6.426498	302.791737	187.967350
min	1.000000	-3000.000000	0.000000	0.000000	0.000000	0.000000
25%	667.000000	1996.000000	2.000000	3.000000	45.000000	30.000000
50%	1333.000000	2004.000000	2.000000	5.000000	60.000000	60.000000
75%	1999.000000	2009.000000	2.000000	6.000000	120.000000	120.000000
max	2665.000000	2015.000000	10.000000	100.000000	12000.000000	6000.00000

year cannot be negative in year published so year should be greater than zero
https://www.technologyreview.com/s/404233/the-start-of-computer-games with the help of this source computer game originated nearly at early 1960 therefore we should remove the rows less than 1960

In [20]:
```
df3=df[df.yearpublished>1960]
```

In [21]:
```
df3.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2624 entries, 0 to 2664
Data columns (total 20 columns):
id                      2624 non-null int64
type                    2624 non-null object
name                    2624 non-null object
yearpublished           2624 non-null int64
minplayers              2624 non-null int64
maxplayers              2624 non-null int64
playingtime             2624 non-null int64
minplaytime             2624 non-null int64
maxplaytime             2624 non-null int64
minage                  2624 non-null int64
users_rated             2624 non-null int64
average_rating          2624 non-null float64
bayes_average_rating    2624 non-null float64
total_owners            2624 non-null int64
total_traders           2624 non-null int64
total_wanters           2624 non-null int64
total_wishers           2624 non-null int64
total_comments          2624 non-null int64
total_weights           2624 non-null int64
average_weight          2624 non-null float64
dtypes: float64(3), int64(15), object(2)
memory usage: 430.5+ KB
```

In [22]:
```
df3.describe()
```

Out[22]:

	id	yearpublished	minplayers	maxplayers	playingtime	minplaytim
count	2624.000000	2624.000000	2624.000000	2624.000000	2624.000000	2624.00000
mean	1331.831936	2001.763338	2.054116	5.274009	105.620427	95.375381
std	769.467898	9.659116	0.720675	6.465368	304.999887	189.213599
min	1.000000	1961.000000	0.000000	0.000000	0.000000	0.000000
25%	663.750000	1997.000000	2.000000	3.000000	45.000000	30.000000
50%	1332.500000	2004.000000	2.000000	5.000000	60.000000	60.000000
75%	1998.250000	2009.000000	2.000000	6.000000	120.000000	120.000000
max	2665.000000	2015.000000	10.000000	100.000000	12000.000000	6000.00000

In [23]:
```
df3['id'].is_unique
```

Out[23]: True

In [29]:
```
df4 = df3.set_index('id')
```

There should be alteast one player is requried to play the game in both the cases minplayers and maxplayers so we have to those rows which is having zero players in these two columns.

There should be alteast some age is requried to play the game it should not be zero.

In [31]:
```
df5 = df4[(df4.minplayers>0) & (df4.maxplayers>0)& (df4.minage>0)]
```

In [32]:
```
df5.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2476 entries, 1 to 2665
Data columns (total 19 columns):
type                    2476 non-null object
name                    2476 non-null object
yearpublished           2476 non-null int64
minplayers              2476 non-null int64
maxplayers              2476 non-null int64
playingtime             2476 non-null int64
minplaytime             2476 non-null int64
maxplaytime             2476 non-null int64
minage                  2476 non-null int64
users_rated             2476 non-null int64
average_rating          2476 non-null float64
bayes_average_rating    2476 non-null float64
total_owners            2476 non-null int64
total_traders           2476 non-null int64
total_wanters           2476 non-null int64
total_wishers           2476 non-null int64
total_comments          2476 non-null int64
total_weights           2476 non-null int64
average_weight          2476 non-null float64
dtypes: float64(3), int64(14), object(2)
memory usage: 386.9+ KB
```

In [33]:
```
df5.describe()
```

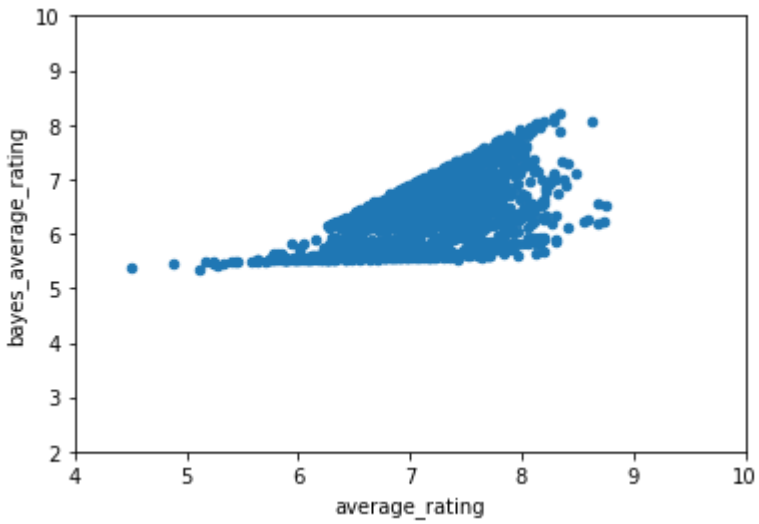
Out[33]:

	yearpublished	minplayers	maxplayers	playingtime	minplaytime	maxplaytin
count	2476.000000	2476.000000	2476.000000	2476.000000	2476.000000	2476.00000
mean	2001.924071	2.092892	5.385299	106.836834	96.197496	106.836834
std	9.645677	0.663214	6.310051	312.527785	192.531309	312.527785
min	1961.000000	1.000000	1.000000	0.000000	0.000000	0.000000
25%	1997.000000	2.000000	4.000000	45.000000	30.000000	45.000000
50%	2004.000000	2.000000	5.000000	60.000000	60.000000	60.000000
75%	2009.000000	2.000000	6.000000	120.000000	120.000000	120.000000
max	2015.000000	10.000000	100.000000	12000.000000	6000.000000	12000.0000

In [34]:
```
import seaborn as sns
import matplotlib.pyplot as plt
```

In [35]:
```
df5.plot(kind="scatter", x="average_rating", y="bayes_average_rating",xlim=(4,
10),ylim=(2,10))
```

Out[35]: <matplotlib.axes._subplots.AxesSubplot at 0x2433ceac2b0>

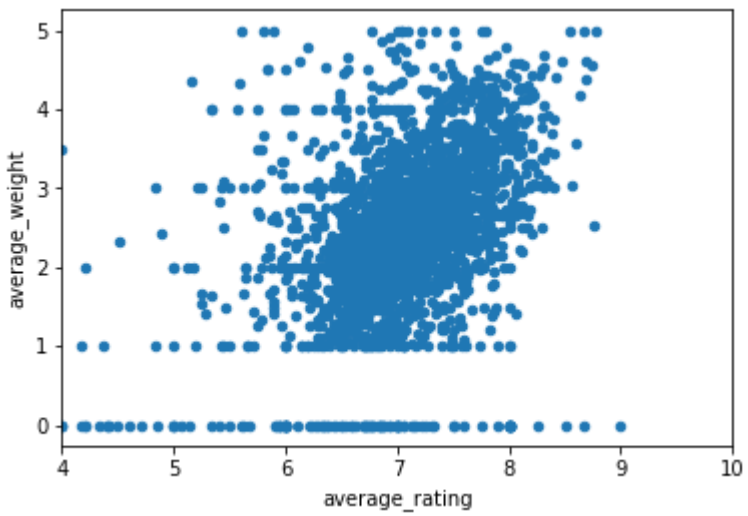


Average rating and bayes average rating not much correlated to each other but not highly correlated to each other

.

In [36]:
```
df5.plot(kind="scatter", x="average_rating", y="average_weight",xlim=(4,10))
```

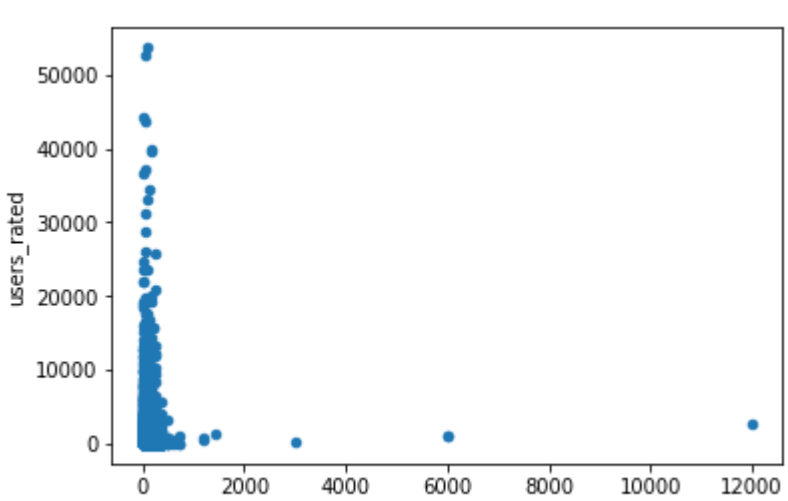
Out[36]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x2433cf91630>
```



Average rating is not vary much as it varies between 6 to 8 for most of the cases but average weight vary from 1 to 5 So , variation is more in average weight compare to average rating

In [139]:
```
df5.plot(kind="scatter", x="playingtime", y="users_rated",xlim=())
```

Out[139]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1806a737c18>
```



Data is not correlated between playingtime and user rated

In [37]:
```
df5.type.value_counts()
```

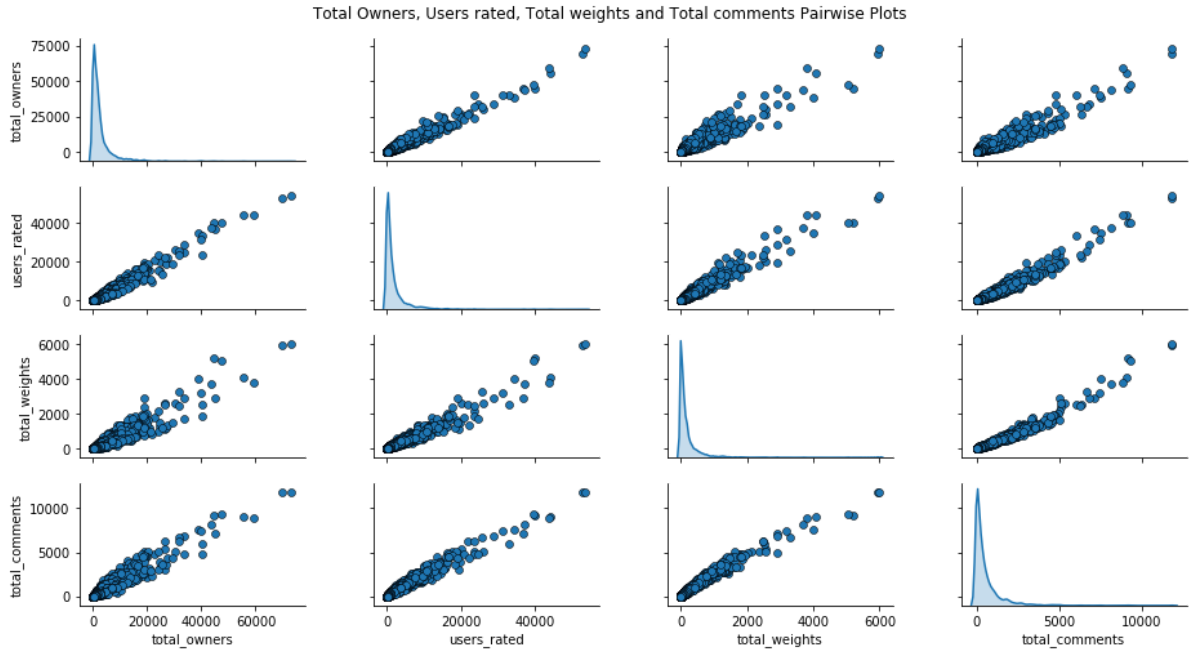
Out[37]:
```
boardgame            1429
boardgameexpansion    1047
Name: type, dtype: int64
```

In [141]:
```
correlation = df5.corr(method='pearson')
```

In [38]:
```
print(correlation)
```

```
                playingtime  total_comments
playingtime        1.000000       -0.010367
total_comments    -0.010367        1.000000
```

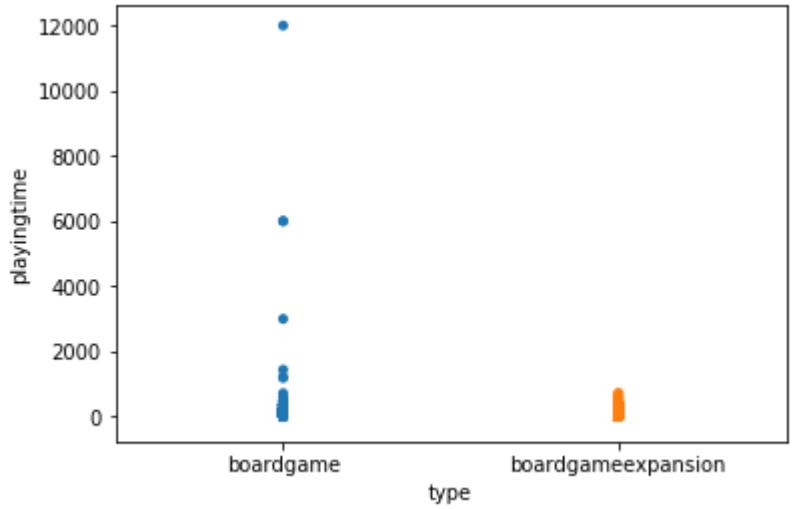
In [144]:
```
cols = ['total_owners', 'users_rated', 'total_weights', 'total_comments']
pp = sns.pairplot(df5[cols], size=1.8, aspect=1.8,
                  plot_kws=dict(edgecolor="k", linewidth=0.5),
                  diag_kind="kde", diag_kws=dict(shade=True))

fig = pp.fig
fig.subplots_adjust(top=0.93, wspace=0.3)
t = fig.suptitle('Total Owners, Users rated, Total weights and Total comments
 Pairwise Plots', fontsize=12)
```



Total Owners, Users rated, Total weights and Total comments are correlated to each other but indiviually they are right skewed it shows that data is not normal.

In [39]:
```
sns.stripplot(df5['type'], df5['playingtime'])
```

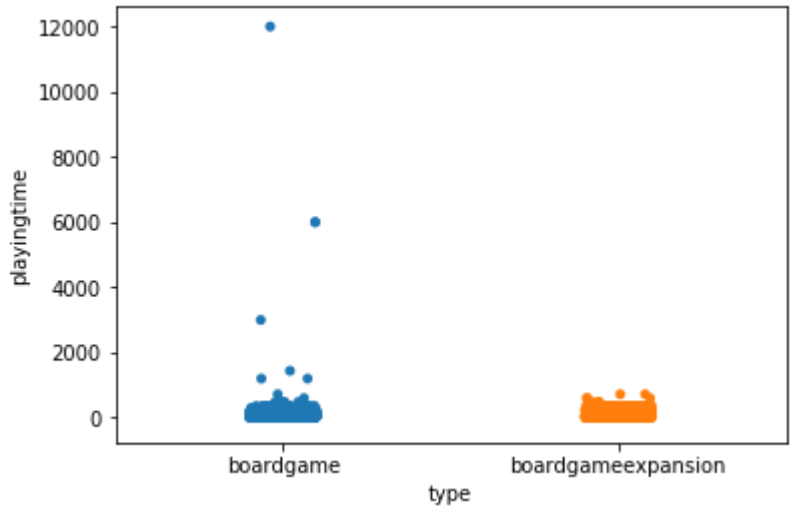
Out[39]: <matplotlib.axes._subplots.AxesSubplot at 0x2433caada20>



Variation is more in the broadgame compare to the broadgameexpansion

In [40]:
```
sns.stripplot(df5['type'], df5['playingtime'],jitter=True)
```

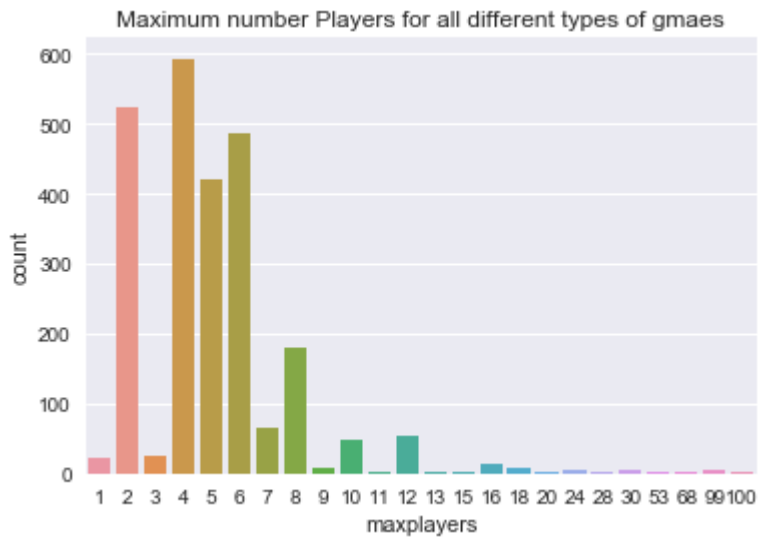
Out[40]: <matplotlib.axes._subplots.AxesSubplot at 0x2433cff36a0>



In [ ]:
```
# There is more variation in the broadgame.
```

In [71]:
```
import matplotlib.pyplot as plt
import seaborn as sns
sns.set(color_codes=True)
sns.countplot(df5['maxplayers'])
plt.title('Maximum number Players for all different types of gmaes')
```

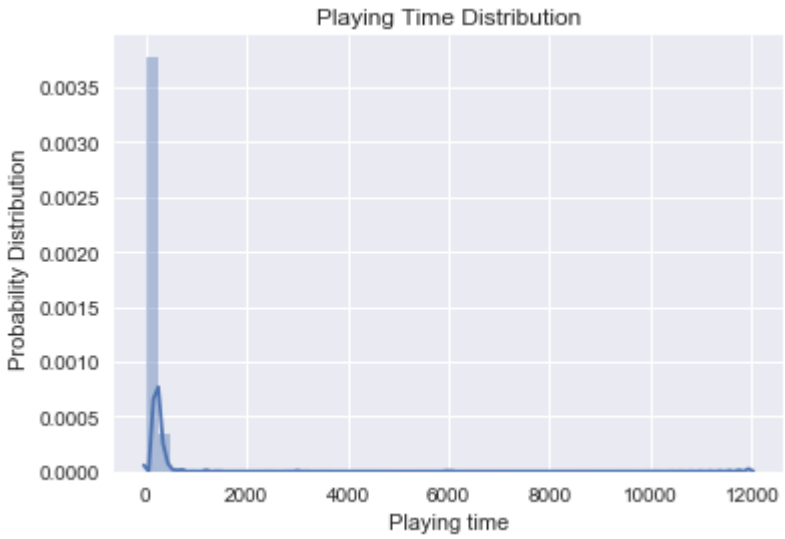
Out[71]: Text(0.5,1,'Maximum number Players for all different types of gmaes')



In most of the games maximum number of players is between 2 and 6. Also there are games with maximum number of players upto 100.

In [75]:
```
sns.distplot(df5['playingtime'])
plt.title('Playing Time Distribution')
plt.xlabel('Playing time')
plt.ylabel('Probability Distribution ')
```

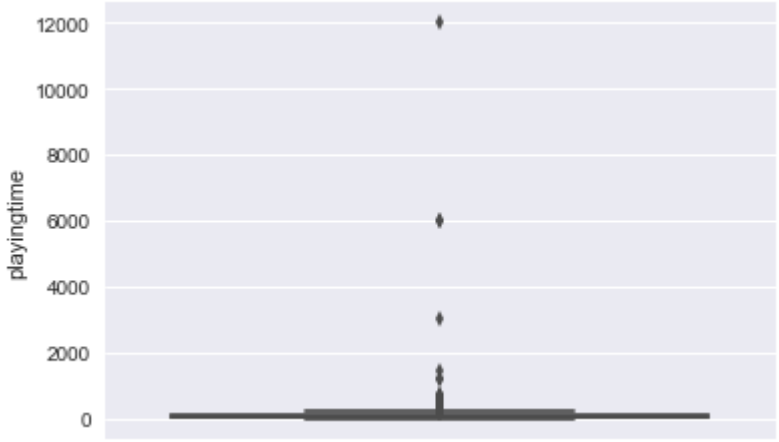
Out[75]: Text(0,0.5,'Probability Distribution ')



Playing time is not normal and it is right skewed due to presence of outliers

In [81]:
```
sns.boxplot(df5['playingtime'],hue=df.type,orient="v")
```

Out[81]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x2433d56da20>
```

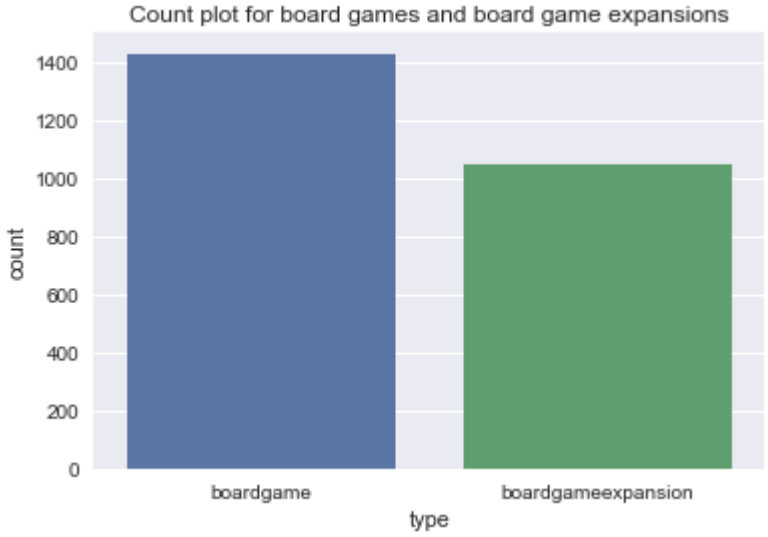


From the box plot it is clear that for most of the data points are around mean and median and there are some outliers at 12000 and 6000 .

From the existing dataset 2665 observation is there after considering the valid and with supporting resaons it cames around the 2476. There fore total no of outlier is 189 with valid reason and support.So, my df5 contain the 2476 nos. of observation.

In [85]:
```
sns.countplot(df5['type'])
plt.title('Count plot for board games and board game expansions')
```

Out[85]:
```
Text(0.5,1,'Count plot for board games and board game expansions')
```

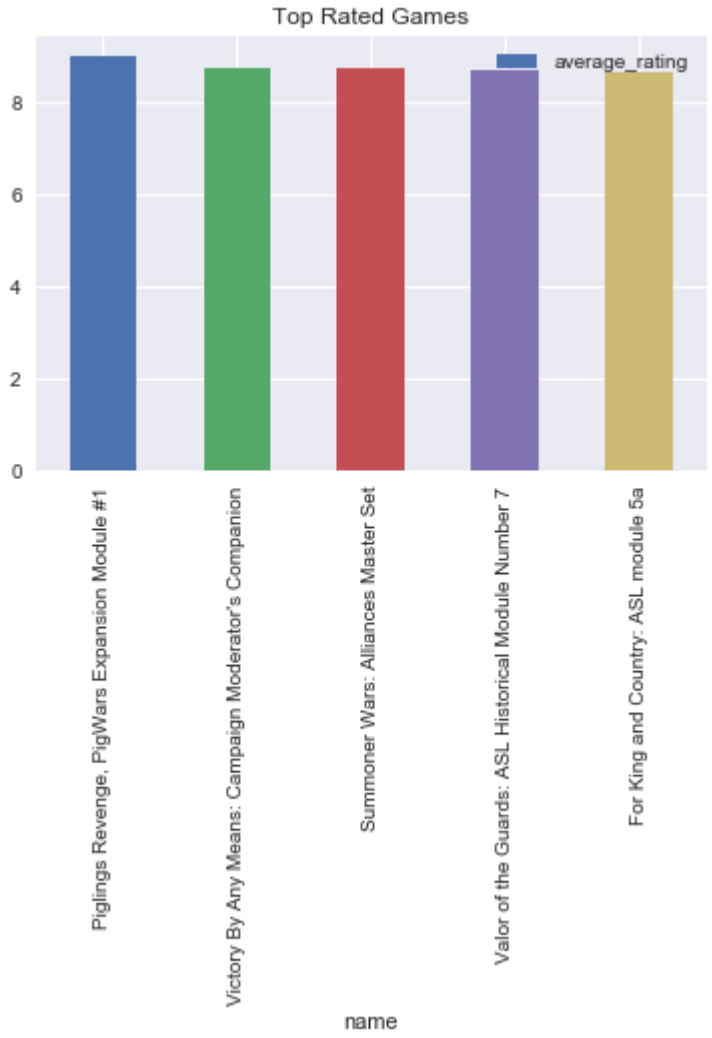


In [86]:
```
df5.type.value_counts()
```

Out[86]:
```
boardgame            1429
boardgameexpansion    1047
Name: type, dtype: int64
```

In [88]:
```
df5.sort_values("average_rating", ascending=False).head(5).plot('name','averag
e_rating',kind='bar')
plt.title('Top Rated Games')
```

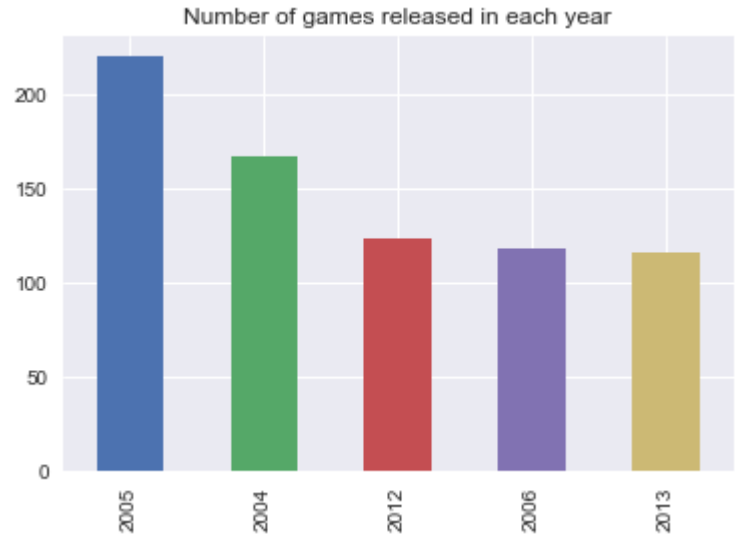
Out[88]: Text(0.5,1,'Top Rated Games')



Top rated game is Piglings Revenge, PigWars Expansion Module 1, followed by Victory By Any Means: Campaign Moderator's Companion Summoner Wars: Alliances Master Set , Victor of the Guards and Far king and Country

In [90]:
```
df5.yearpublished.value_counts().head(5).plot(kind='bar')
plt.title('Number of games released in each year')
```

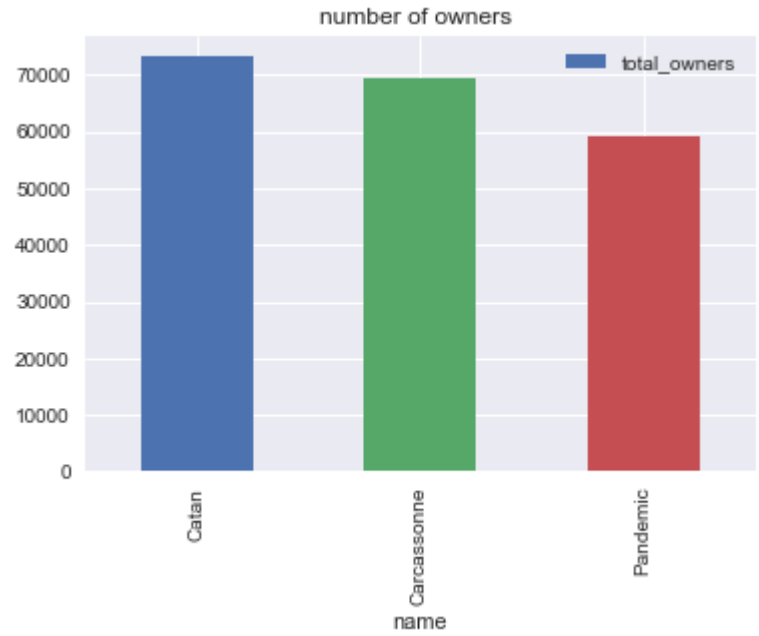
Out[90]: Text(0.5,1,'Number of games released in each year')



Number of games published is hieghest in the year 2005 followed by 2004 and 2012

In [96]:
```
df5.sort_values("total_owners", ascending=False).head(3).plot('name','total_owners',kind='bar')
plt.title('number of owners ')
```

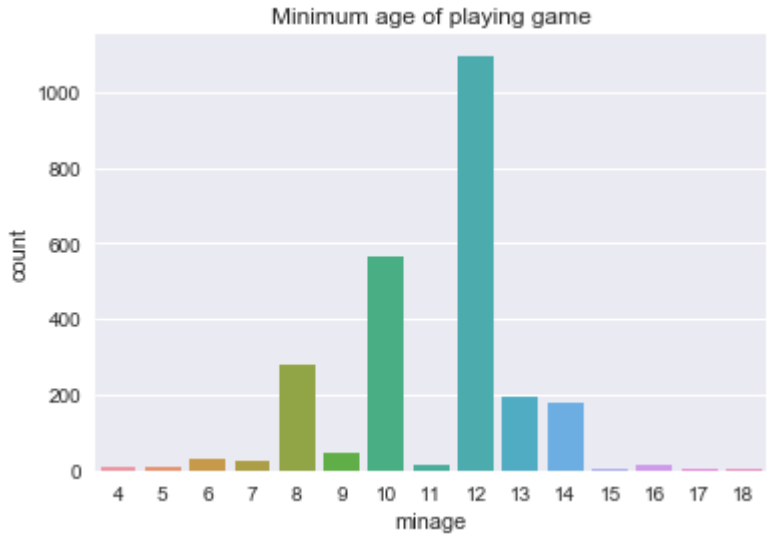
Out[96]: Text(0.5,1,'number of owners ')



While checking the top three owners, it is found that top owned game is Catan followed by Carcassonne and Pandemic.

In [98]:
```
sns.countplot(df5['minage'])
plt.title('Minimum age of playing game')
```

Out[98]: Text(0.5,1,'Minimum age of playing game')



For most of the games minmum age required for playing game is 12 followed by 10 and it varies from 4 to 18 years

Insnight from the Visualization

- Variation is more in the broadgame compare to the broadgameexpansion when it is compare with playing time.
- Average rating is not vary much as it varies between 6 to 8 for most of the cases but average weight vary from 1 to 5 So , variation is more in average weight compare to average rating
- Total Owners, Users rated, Total weights and Total comments are correlated to each other but indiviually they are right skewed it shows that data is not normal.
- Data is not correlated between playingtime and user rated/
- Playing time is not normal and it is right skewed due to presence of outliers.
- The number of board games are higher than boargames expansion.
- Number of games published is hieghest in the year 2005 followed by 2004 and 2012.
- While checking the top three owners, it is found that top owned game is Catan followed by Carcassonne and Pandemic.
- For most of the games minmum age required for playing game is 12 followed by 10 and it varies from 4 to 18 years.

Going for Hypothesis testing

Here the problem is whether the amount of time gamers spend for each game type is same or not Null Hypothesis, H0 = Amount of time spend in each game type is same Alternate Hypothesis, HA = Amount of time spend in each game type is different

Making two groups to do the further analysis

In [46]:
```
boardgames=df5[df5['type'] == 'boardgame']
boardgamesexp=df5[df5['type'] == 'boardgameexpansion']
```

In [48]:
```
boardgames.describe()
```

Out[48]:

	yearpublished	minplayers	maxplayers	playingtime	minplaytime	maxplaytin
count	1429.000000	1429.000000	1429.000000	1429.000000	1429.000000	1429.00000
mean	2005.603219	2.077677	4.944017	107.156753	89.019594	107.156753
std	8.868477	0.665463	5.232114	403.187846	239.533863	403.187846
min	1962.000000	1.000000	1.000000	0.000000	0.000000	0.000000
25%	2003.000000	2.000000	4.000000	45.000000	30.000000	45.000000
50%	2008.000000	2.000000	4.000000	60.000000	60.000000	60.000000
75%	2012.000000	2.000000	5.000000	120.000000	90.000000	120.000000
max	2015.000000	8.000000	99.000000	12000.000000	6000.000000	12000.0000

In [49]:
```
boardgamesexp.describe()
```

Out[49]:

	yearpublished	minplayers	maxplayers	playingtime	minplaytime	maxplaytime
count	1047.000000	1047.000000	1047.000000	1047.000000	1047.000000	1047.000000
mean	1996.902579	2.113658	5.987584	106.400191	105.994269	106.400191
std	8.309044	0.659882	7.497408	95.828616	95.982093	95.828616
min	1961.000000	1.000000	1.000000	0.000000	0.000000	0.000000
25%	1991.000000	2.000000	2.000000	45.000000	45.000000	45.000000
50%	1999.000000	2.000000	6.000000	90.000000	90.000000	90.000000
75%	2004.000000	2.000000	7.000000	120.000000	120.000000	120.000000
max	2009.000000	10.000000	100.000000	720.000000	720.000000	720.000000

Here the data is unbalanced we have to do non parametric test only. Anyway just checking for shapiro test.

Performing shapiro and levenes test to confirm assumptions of Normality .

Shapiro Test

Ho : Null Hypothesis - boardgame Data is normally distributed, Ha : Alternate Hypothesis - boardgame Data is not normally distributed

In [67]:
```
import matplotlib.pyplot as plt
from scipy.stats import levene, shapiro, mannwhitneyu
shapiro(boardgames.playingtime)
```

Out[67]: (0.10300081968307495, 0.0)

p value < 0.05 we reject the null hypothesis i.e. data is normally distributed

In [51]:
```
shapiro(boardgamesexp.playingtime)
```

Out[51]: (0.8309653997421265, 7.320831292380178e-32)

P-value < 0.05 hence data is not normal, we reject the null hypothesis

Two Sample t test (independent one)

Here samples are not normally distributed, so going forward with non parametric test, within that Mann Whitney U test for two sample t test (independent one)

In [53]:
```
u, p_value = mannwhitneyu(boardgames.playingtime, boardgamesexp.playingtime)
print ("two-sample mannwhitneyu-test p-value=", p_value)
```

```
two-sample mannwhitneyu-test p-value= 4.20308334150003e-08
```

p value < 0.05, hence null hypothesis is rejected. That is the amount of time gamers spend for each game type is different.

In [54]:
```
(np.mean(boardgames.playingtime) - np.mean(boardgamesexp.playingtime)) / np.sq
rt(((1429-1)*np.var(boardgames.playingtime)+(1047-1)*np.var(boardgamesexp.play
ingtime)) / 1429+1047-2)
```

Out[54]:
```
0.0018344281377945728
```

In [56]:
```
from statsmodels.stats.power import ttest_power
print(ttest_power(0.0018344281377945728, nobs=2476, alpha=0.05, alternative='t
wo-sided'))
```

```
0.050954253401354885
```

Actual power of the test is coming around only 5 % it shows that model is weak.

In [57]:
```
#Power of Test conducted to calculate the ideal sample size(n) for 2 data vari
ables
import scipy
#We shall create a custom function to identify the need for a larger sample si
ze value

def sample_power_difftest(d, s, power=0.8, sig=0.05):
    z = scipy.stats.norm.isf([sig/2])                  #Inverse Survival Func
tion for calculation
    zp = -1 * scipy.stats.norm.isf([power])
    n = (2*(s**2)) * ((zp + z)**2) / (d**2)
    return int(round(n[0]))


# Given d (difference in means),
# s (pooled standard deviation),
# sig (significance level, typically .05),
# power (typically .80)
```

In [58]:
```
# Let us calculate the necessary data in order to run our power test

#Firstly, calculate the means of the 2 variables Current and New
mean_boardgame=boardgames.playingtime.mean()
mean_boardgamesexp=boardgamesexp.playingtime.mean()

#Secondly, calculate the standard deviations of the 2 variables
std_boardgame=boardgames.playingtime.std()
std_boardgamesexp=boardgamesexp.playingtime.std()
```

In [59]:
```
mean_boardgame
```

Out[59]: 107.15675297410776

In [60]:
```
mean_boardgamesexp
```

Out[60]: 106.40019102196753

In [61]:
```
std_boardgame
```

Out[61]: 403.1878457949021

In [62]:
```
std_boardgamesexp
```

Out[62]: 95.8286159393457

In [63]:
```
#Now, let us calculate the difference in their means
d = mean_boardgame - mean_boardgamesexp

#Next, we calculate the pooled Standard Deviation for the 2 variables using th
e following formula
s = np.sqrt(((std_boardgame**2)+(std_boardgamesexp**2))/2)
```

In [64]:
```
#Now we calculate the sample size needed for a power of 0.8
n = sample_power_difftest(d,s,power=0.8,sig=0.05)
```

In [65]:
```
n
```

Out[65]: 2355045

We can observe from above result, that in order for the hypothesis test to have a power of confidence of atleast 80%, atleast 2355045 observations are needed.

However, our dataset has only 2476 observations, hence a larger sample size would be needed to increase the power of our Model.Power is the probability that the test correctly rejects the Null Hypothesis if the Alternative Hypothesis is true.

Conclusion

For two sample t test

As the two group data is unbalanced and shapiro test fail for both the group from that it is clear that data is not parametric we have to go for non parametric test.

Here problem is to find the wheather playing time for both game type is same or not after doing the Mann Whitney U test it is found that amount of time spend by gamers or players is different.

But after checking the power of test results is showing up approx 5 % . It means that only 5 % of probability that rejecting the null hypothesis when it is actually false.It implies that we have a weak model.

For getting the power of test 80% minimum number of observation required is 2355045.