In [104]:

```
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"

%matplotlib inline

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

import seaborn as sns
sns.set_style("whitegrid")
sns.set_context("notebook")
#sns.set_context("poster")
```

Create

In [105]:

```
import pandas as pd

# Wikipedia 2017 - entry "Economy of Europe"
# Creation of a Data Frame using a Dictionnary
#
eu_metro_areas=pd.DataFrame({"City": ["London","Paris","Moscow","Madrid","Barcelona",
                                      "Rome","Milan","Vienna","Lisbon","Athens","Berlin","E
                            "Rank": [1,2,3,4,5,6,7,8,9,10,11,12],
                            "GDP" : [732,669,520,230,177,144,136,122,98,96,95,51],
                            "Population": [11.9,11.5,11.5,5.8,4.97,4.34,3.2,2.18,2.44,4.01
                            "GDP per Capita": [61.5,62.4,45.2,39.7,35.6,41.6,44.2,56.0,40.
                            "Eurozone": ["N","Y","N","Y","Y","Y","Y","Y","Y","Y","Y","N"]}
```

In [106]:

```
eu_metro_areas
```

Out[106]:

	City	Rank	GDP	Population	GDP per Capita	Eurozone
0	London	1	732	11.90	61.5	N
1	Paris	2	669	11.50	62.4	Y
2	Moscow	3	520	11.50	45.2	N
3	Madrid	4	230	5.80	39.7	Y
4	Barcelona	5	177	4.97	35.6	Y
5	Rome	6	144	4.34	41.6	Y
6	Milan	7	136	3.20	44.2	Y
7	Vienna	8	122	2.18	56.0	Y
8	Lisbon	9	98	2.44	40.2	Y
9	Athens	10	96	4.01	23.9	Y
10	Berlin	11	95	4.97	19.1	Y
11	Bucharest	12	51	2.30	23.3	N

In [107]:

```
#Creation of a Data Frame using lists
#
eu_metro_areas1=pd.DataFrame([["London",1,732,11.9,61.5,"N"],
                              ["Paris",2,669,11.5,62.4,"Y"],
                              ["Moscow",3,520,11.5,62.4,"Y"],
                              ["Madrid",4,230,5.8,39.7,"Y"],
                              ["Barcelona",5,177,4.97,35.6,"Y"],
                              ["Rome",6,144,4.34,41.6,"Y"],
                              ["Milan",7,136,3.2,44.2,"Y"],
                              ["Vienna",8,122,2.18,56.0,"Y"],
                              ["Lisbon",9,98,2.44,40.2,"Y"],
                              ["Athens",10,96,4.01,23.9,"Y"],
                              ["Berlin",11,95,4.97,19.1,"Y"],
                              ["Bucharest",12,51,2.3,23.3,"N"]],
                              columns=["City","Rank","GDP","Population","GDP per Capita","E
eu_metro_areas1
```

Out[107]:

	City	Rank	GDP	Population	GDP per Capita	Eurozone
0	London	1	732	11.90	61.5	N
1	Paris	2	669	11.50	62.4	Y
2	Moscow	3	520	11.50	62.4	Y
3	Madrid	4	230	5.80	39.7	Y
4	Barcelona	5	177	4.97	35.6	Y
5	Rome	6	144	4.34	41.6	Y
6	Milan	7	136	3.20	44.2	Y
7	Vienna	8	122	2.18	56.0	Y
8	Lisbon	9	98	2.44	40.2	Y
9	Athens	10	96	4.01	23.9	Y
10	Berlin	11	95	4.97	19.1	Y
11	Bucharest	12	51	2.30	23.3	N

Import

In [108]:

```
#Import from a csv file
#
eu_metro_areas3=pd.read_csv("eu_metro_areas.csv",sep=";")
eu_metro_areas3
```

Out[108]:

	City	Rank	GDP	Population	GDP per Capita	Eurozone
0	London	1	732	11.90	61.5	N
1	Paris	2	669	11.50	62.4	Y
2	Moscow	3	520	11.50	45.2	N
3	Madrid	4	230	5.80	39.7	Y
4	Barcelona	5	177	4.97	35.6	Y
5	Rome	6	144	4.34	41.6	Y
6	Milan	7	136	3.20	44.2	Y
7	Vienna	8	122	2.18	56.0	Y
8	Lisbon	9	98	2.44	40.2	Y
9	Athens	10	96	4.01	23.9	Y
10	Berlin	11	95	4.97	19.1	Y
11	Bucharest	12	51	2.30	23.3	N

In [109]:

```
# Importing from an Excel file
#
eu_metro_areas4=pd.read_excel("eu_metro_areas.xlsx")
eu_metro_areas4
```

Out[109]:

	City	Rank	GDP	Population	GDP per Capita	Eurozone
0	London	1	732	11.90	61.5	N
1	Paris	2	669	11.50	62.4	Y
2	Moscow	3	520	11.50	45.2	N
3	Madrid	4	230	5.80	39.7	Y
4	Barcelona	5	177	4.97	35.6	Y
5	Rome	6	144	4.34	41.6	Y
6	Milan	7	136	3.20	44.2	Y
7	Vienna	8	122	2.18	56.0	Y
8	Lisbon	9	98	2.44	40.2	Y
9	Athens	10	96	4.01	23.9	Y
10	Berlin	11	95	4.97	19.1	Y
11	Bucharest	12	51	2.30	23.3	N

In [110]:

```
# Managing the data frame interactively
# ! pip install qgrid
# ! jupyter nbextension enable --py --sys-prefix qgrid
# ! jupyter nbextension enable --py --sys-prefix widgetsnbext3ension
#
# if you are using jupyterlab
# ! pip install jupyterlab
# ! jupyter labextension install @jupyter-widgets/jupyterlab-manager

# Install the qgrid-jupyterlab extension and enable:
# ! jupyter labextension install qgrid

# Requirements:
#   ipywidgets -> pip install ipywidgets (or pip install --upgrade ipywidgets)

import ipywidgets as widgets
import qgrid


#qgrid.show_grid(eu_metro_areas, show_toolbar=True)
qgrid_widget=qgrid.show_grid(eu_metro_areas)
qgrid_widget

# if you want to assign the result of a filtering or the changes
# your_new_df =qgrid_wodget.get_changed_df()
```

A Jupyter widget could not be displayed because the widget state could not be found. This could happen if the kernel storing the widget is no longer available, or if the widget state was not saved in the notebook. You may be able to create the widget by running the appropriate cells.

Mission 1

a) Create two imaginary datasets. One with a dictionnary and the other using lists

b) Import the Excel file VideoGameSales that you downloaded from the Moodle

In [111]:

```
# a) Create two imaginary datasets. One with a dictionnary and the other using lists

# Dataframe using dictionary method

fruit_dic = pd.DataFrame({"Name": ["Banana", "Apple", "Orange", "Mandarine", "Strawberry",
                     "Color": ["Yellow", "Red", "Orange", "Orange", "Red", "Green", "Orange
                     "Deliciousness": [7, 6, 8, 9, 7, 7, 6]})
fruit_dic

# Dataframe using list method

fruit_list = pd.DataFrame([["Banana", "Yellow", 7],
                      ["Apple", "Red", 6],
                      ["Orange", "Orange", 8],
                      ["Mandarine", "Orange", 9],
                      ["Strawberry", "Red", 7],
                      ["Melon", "Green", 7],
                      ["Papaya", "Orange", 6]],
                     columns=["Name", "Color", "Deliciousness"])
fruit_list
```

Out[111]:

	Name	Color	Deliciousness
0	Banana	Yellow	7
1	Apple	Red	6
2	Orange	Orange	8
3	Mandarine	Orange	9
4	Strawberry	Red	7
5	Melon	Green	7
6	Papaya	Orange	6

Out[111]:

	Name	Color	Deliciousness
0	Banana	Yellow	7
1	Apple	Red	6
2	Orange	Orange	8
3	Mandarine	Orange	9
4	Strawberry	Red	7
5	Melon	Green	7
6	Papaya	Orange	6

In [112]:

```
# b) Import the Excel file VideoGameSales that you downloaded from the Moodle

vgs = pd.read_excel("VideoGamesSales.xlsx")
```

Explore

In [113]:

```
# Summary information of the dataframe

eu_metro_areas.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 12 entries, 0 to 11
Data columns (total 6 columns):
City              12 non-null object
Rank              12 non-null int64
GDP               12 non-null int64
Population        12 non-null float64
GDP per Capita    12 non-null float64
Eurozone          12 non-null object
dtypes: float64(2), int64(2), object(2)
memory usage: 656.0+ bytes
```

In [114]:

```
# If you only want to know the dimmensions

eu_metro_areas.shape
```

Out[114]:

```
(12, 6)
```

In [115]:

```
# Examine first rows

eu_metro_areas.head()
```

Out[115]:

	City	Rank	GDP	Population	GDP per Capita	Eurozone
0	London	1	732	11.90	61.5	N
1	Paris	2	669	11.50	62.4	Y
2	Moscow	3	520	11.50	45.2	N
3	Madrid	4	230	5.80	39.7	Y
4	Barcelona	5	177	4.97	35.6	Y

In [116]:

```
# Examine last rows

eu_metro_areas.tail(7)
```

Out[116]:

	City	Rank	GDP	Population	GDP per Capita	Eurozone
5	Rome	6	144	4.34	41.6	Y
6	Milan	7	136	3.20	44.2	Y
7	Vienna	8	122	2.18	56.0	Y
8	Lisbon	9	98	2.44	40.2	Y
9	Athens	10	96	4.01	23.9	Y
10	Berlin	11	95	4.97	19.1	Y
11	Bucharest	12	51	2.30	23.3	N

In [117]:

```
# Describe

eu_metro_areas.describe()
```

Out[117]:

	Rank	GDP	Population	GDP per Capita
count	12.000000	12.000000	12.000000	12.000000
mean	6.500000	255.833333	5.759167	41.058333
std	3.605551	240.674746	3.720795	14.296246
min	1.000000	51.000000	2.180000	19.100000
25%	3.750000	97.500000	3.010000	32.675000
50%	6.500000	140.000000	4.655000	40.900000
75%	9.250000	302.500000	7.225000	47.900000
max	12.000000	732.000000	11.900000	62.400000

In [118]:

```
# ---- Functions ----
#   count()      Number or non null observations
#   sum()        Sum of values
#   mean()       Mean of values
#   median()     Arithmetic median of values
#   min()        Minimum
#   max()        Maximum
#   std()        Unbiased standard deviation
#   var()        Unbiased variance

eu_metro_areas.mean()
eu_metro_areas.mean(axis=1) #row wise - in this case has no sense
eu_metro_areas["GDP"].mean()
eu_metro_areas["GDP"].median()
eu_metro_areas["GDP"].std()*2
eu_metro_areas["GDP"].quantile(0.8)
```

Out[118]:

```
Rank               6.500000
GDP              255.833333
Population         5.759167
GDP per Capita    41.058333
dtype: float64
```

Out[118]:

```
0     201.6000
1     186.2250
2     144.9250
3      69.8750
4      55.6425
5      48.9850
6      47.6000
7      47.0450
8      37.4100
9      33.4775
10     32.5175
11     22.1500
dtype: float64
```

Out[118]:

```
255.83333333333334
```

Out[118]:

```
140.0
```

Out[118]:

```
481.34949188020687
```

Out[118]:

```
462.00000000000017
```

Select

In [119]:

```
# Data selection

col1=["GDP","Population"]
eu_metro_areas[col1]

eu_metro_areas["GDP"]
eu_metro_areas[["GDP","Population"]]
eu_metro_areas[:7]
```

Out[119]:

	GDP	Population
0	732	11.90
1	669	11.50
2	520	11.50
3	230	5.80
4	177	4.97
5	144	4.34
6	136	3.20
7	122	2.18
8	98	2.44
9	96	4.01
10	95	4.97
11	51	2.30

Out[119]:

```
0     732
1     669
2     520
3     230
4     177
5     144
6     136
7     122
8      98
9      96
10     95
11     51
Name: GDP, dtype: int64
```

Out[119]:

	GDP	Population
0	732	11.90
1	669	11.50
2	520	11.50
3	230	5.80
4	177	4.97

	GDP	Population
5	144	4.34
6	136	3.20
7	122	2.18
8	98	2.44
9	96	4.01
10	95	4.97
11	51	2.30

Out[119]:

	City	Rank	GDP	Population	GDP per Capita	Eurozone
0	London	1	732	11.90	61.5	N
1	Paris	2	669	11.50	62.4	Y
2	Moscow	3	520	11.50	45.2	N
3	Madrid	4	230	5.80	39.7	Y
4	Barcelona	5	177	4.97	35.6	Y
5	Rome	6	144	4.34	41.6	Y
6	Milan	7	136	3.20	44.2	Y

In [120]:

```
# Filtering

# Metro areas with a population > 3M

eu_metro_areas[eu_metro_areas["Population"]>3]

eu_metro_areas[(eu_metro_areas["Population"]>5) & (eu_metro_areas["GDP"]>150)]

criteria=(eu_metro_areas["Population"]>6) & (eu_metro_areas["GDP"]>150)
eu_metro_areas[criteria]

eu_metro_areas[eu_metro_areas["Population"]>3].head() #or .tail()

# .isnull() is very interesting in pre-processing
```

Out[120]:

	City	Rank	GDP	Population	GDP per Capita	Eurozone
0	London	1	732	11.90	61.5	N
1	Paris	2	669	11.50	62.4	Y
2	Moscow	3	520	11.50	45.2	N
3	Madrid	4	230	5.80	39.7	Y
4	Barcelona	5	177	4.97	35.6	Y
5	Rome	6	144	4.34	41.6	Y
6	Milan	7	136	3.20	44.2	Y
9	Athens	10	96	4.01	23.9	Y
10	Berlin	11	95	4.97	19.1	Y

Out[120]:

	City	Rank	GDP	Population	GDP per Capita	Eurozone
0	London	1	732	11.9	61.5	N
1	Paris	2	669	11.5	62.4	Y
2	Moscow	3	520	11.5	45.2	N
3	Madrid	4	230	5.8	39.7	Y

Out[120]:

	City	Rank	GDP	Population	GDP per Capita	Eurozone
0	London	1	732	11.9	61.5	N
1	Paris	2	669	11.5	62.4	Y
2	Moscow	3	520	11.5	45.2	N

Out[120]:

	City	Rank	GDP	Population	GDP per Capita	Eurozone
0	London	1	732	11.90	61.5	N

	City	Rank	GDP	Population	GDP per Capita	Eurozone
1	Paris	2	669	11.50	62.4	Y
2	Moscow	3	520	11.50	45.2	N
3	Madrid	4	230	5.80	39.7	Y
4	Barcelona	5	177	4.97	35.6	Y

In [121]:

```
# Reversing rows
eu_metro_areas[::-1]
```

Out[121]:

	City	Rank	GDP	Population	GDP per Capita	Eurozone
11	Bucharest	12	51	2.30	23.3	N
10	Berlin	11	95	4.97	19.1	Y
9	Athens	10	96	4.01	23.9	Y
8	Lisbon	9	98	2.44	40.2	Y
7	Vienna	8	122	2.18	56.0	Y
6	Milan	7	136	3.20	44.2	Y
5	Rome	6	144	4.34	41.6	Y
4	Barcelona	5	177	4.97	35.6	Y
3	Madrid	4	230	5.80	39.7	Y
2	Moscow	3	520	11.50	45.2	N
1	Paris	2	669	11.50	62.4	Y
0	London	1	732	11.90	61.5	N

In [122]:

```
# Reversing rows of a single column

eu_metro_areas["City"][::-1]
```

Out[122]:

```
11    Bucharest
10       Berlin
9        Athens
8        Lisbon
7        Vienna
6         Milan
5          Rome
4     Barcelona
3        Madrid
2        Moscow
1         Paris
0        London
Name: City, dtype: object
```

In [123]:

```
# Reversing colums
eu_metro_areas[eu_metro_areas.columns[::-1]]
```

Out[123]:

	Eurozone	GDP per Capita	Population	GDP	Rank	City
0	N	61.5	11.90	732	1	London
1	Y	62.4	11.50	669	2	Paris
2	N	45.2	11.50	520	3	Moscow
3	Y	39.7	5.80	230	4	Madrid
4	Y	35.6	4.97	177	5	Barcelona
5	Y	41.6	4.34	144	6	Rome
6	Y	44.2	3.20	136	7	Milan
7	Y	56.0	2.18	122	8	Vienna
8	Y	40.2	2.44	98	9	Lisbon
9	Y	23.9	4.01	96	10	Athens
10	Y	19.1	4.97	95	11	Berlin
11	N	23.3	2.30	51	12	Bucharest

In [124]:

```
# Filtering and presenting only some columns

eu_metro_areas[eu_metro_areas["Eurozone"]=="Y"][["City","Population"]]
```

Out[124]:

	City	Population
1	Paris	11.50
3	Madrid	5.80
4	Barcelona	4.97
5	Rome	4.34
6	Milan	3.20
7	Vienna	2.18
8	Lisbon	2.44
9	Athens	4.01
10	Berlin	4.97

Mission 2

a) List all Nintendo Games published after 2010 for WIIU
b) List all Video Games with Global Sales larger than the average
c) List all Video Games with Global Sales larger than the average with a User and Critic score also higher than the average

In [125]:

```
# Before working with the database, we should first drop null values

vgs.dropna(inplace = True)
vgs["User_Score"]= pd.to_numeric(vgs["User_Score"], errors="coerce")
```

In [126]:

```
# a) List all Nintendo Games published after 2010 for WIIU

vgs_m2a = vgs[(vgs["Platform"]=="WiiU") & (vgs["Year_of_Release"]>2010) & (vgs["Publisher"]
vgs_m2a
```

Out[126]:

	Name	Platform	Year_of_Release	Genre	Publisher	NA_Sales	EU_Sales	JP_Sale
94	Mario Kart 8	WiiU	2014.0	Racing	Nintendo	3.11	2.09	1.2
162	New Super Mario Bros. U	WiiU	2012.0	Platform	Nintendo	2.30	1.31	1.2
215	Nintendo Land	WiiU	2012.0	Misc	Nintendo	2.52	1.10	0.4
218	Splatoon	WiiU	2015.0	Shooter	Nintendo	1.53	1.15	1.4
229	Super Mario 3D World	WiiU	2013.0	Platform	Nintendo	2.10	1.12	0.7
380	Super Mario Maker	WiiU	2015.0	Platform	Nintendo	1.17	0.87	0.9
642	New Super Luigi U	WiiU	2013.0	Platform	Nintendo	1.25	0.62	0.1
909	Wii Party U	WiiU	2013.0	Misc	Nintendo	0.30	0.54	0.8
1083	Mario Party 10	WiiU	2015.0	Misc	Nintendo	0.68	0.51	0.2
1131	Donkey Kong Country: Tropical Freeze	WiiU	2014.0	Platform	Nintendo	0.69	0.53	0.1
1369	Yoshi's Woolly World	WiiU	2015.0	Platform	Nintendo	0.60	0.47	0.1
1456	Hyrule Warriors	WiiU	2014.0	Action	Nintendo	0.58	0.42	0.1
1537	Captain Toad: Treasure Tracker	WiiU	2014.0	Puzzle	Nintendo	0.53	0.37	0.1
1659	Pikmin 3	WiiU	2013.0	Strategy	Nintendo	0.43	0.31	0.2
1689	LEGO City Undercover	WiiU	2013.0	Platform	Nintendo	0.47	0.41	0.1
1860	The Legend of Zelda: Twilight Princess HD	WiiU	2016.0	Action	Nintendo	0.53	0.30	0.0
2237	Xenoblade Chronicles X	WiiU	2015.0	Role-Playing	Nintendo	0.35	0.28	0.1

	Name	Platform	Year_of_Release	Genre	Publisher	NA_Sales	EU_Sales	JP_Sale
2453	Bayonetta 2	WiiU	2014.0	Action	Nintendo	0.32	0.28	0.1
3589	Kirby and the Rainbow Curse	WiiU	2015.0	Platform	Nintendo	0.22	0.13	0.1
4086	Mario Tennis Ultra Smash	WiiU	2015.0	Sports	Nintendo	0.12	0.15	0.1
4351	Animal Crossing: Amiibo Festival	WiiU	2015.0	Misc	Nintendo	0.19	0.11	0.0
4416	The Wonderful 101	WiiU	2013.0	Action	Nintendo	0.19	0.10	0.0
4664	Wii Sports Club	WiiU	2014.0	Sports	Nintendo	0.17	0.14	0.0
4805	Star Fox: Zero	WiiU	2016.0	Shooter	Nintendo	0.16	0.10	0.0
5745	NES Remix	WiiU	2014.0	Action	Nintendo	0.17	0.00	0.0
5875	Sing Party	WiiU	2012.0	Misc	Nintendo	0.13	0.12	0.0
6227	Paper Mario: Color Splash	WiiU	2016.0	Role-Playing	Nintendo	0.12	0.06	0.0
11942	Devil's Third	WiiU	2015.0	Action	Nintendo	0.04	0.01	0.0
15576	Mario vs. Donkey Kong: Tipping Stars	WiiU	2015.0	Puzzle	Nintendo	0.00	0.00	0.0
15668	Art Academy: Home Studio	WiiU	2015.0	Misc	Nintendo	0.00	0.00	0.0

In [127]:

```
# b) List all Video Games with Global Sales larger than the average

vgs_m2b = vgs[vgs["Global_Sales"] > vgs["Global_Sales"].mean()]
vgs_m2b
```

Out[127]:

	Name	Platform	Year_of_Release	Genre	Publisher	NA_Sales	EU_Sales	J
0	Wii Sports	Wii	2006.0	Sports	Nintendo	41.36	28.95	
1	Mario Kart Wii	Wii	2008.0	Racing	Nintendo	15.67	12.75	
2	Wii Sports Resort	Wii	2009.0	Sports	Nintendo	15.61	10.92	
3	New Super Mario Bros.	DS	2006.0	Platform	Nintendo	11.28	9.14	
4	Wii Play	Wii	2006.0	Misc	Nintendo	13.96	9.18	
5	New Super Mario Bros. Wii	Wii	2009.0	Platform	Nintendo	14.42	6.94	
7	Mario Kart DS	DS	2005.0	Racing	Nintendo	9.71	7.47	
8	Wii Fit	Wii	2007.0	Sports	Nintendo	8.92	8.03	
9	Wii Fit Plus	Wii	2009.0	Sports	Nintendo	9.01	8.49	
10	Kinect Adventures!	X360	2010.0	Misc	Microsoft Game Studios	14.98	4.88	
11	Grand Theft Auto V	PS3	2013.0	Action	Take-Two Interactive	7.01	9.04	
12	Grand Theft Auto: San Andreas	PS2	2004.0	Action	Take-Two Interactive	9.43	0.40	
13	Brain Age: Train Your Brain in Minutes a Day	DS	2005.0	Misc	Nintendo	4.74	9.20	
15	Grand Theft Auto V	X360	2013.0	Action	Take-Two Interactive	9.65	5.12	
16	Grand Theft Auto: Vice City	PS2	2002.0	Action	Take-Two Interactive	8.41	5.49	
18	Brain Age 2: More Training in Minutes a Day	DS	2005.0	Puzzle	Nintendo	3.43	5.35	
20	Gran Turismo 3: A-Spec	PS2	2001.0	Racing	Sony Computer Entertainment	6.85	5.09	
21	Call of Duty: Modern Warfare 3	X360	2011.0	Shooter	Activision	9.03	4.22	

	Name	Platform	Year_of_Release	Genre	Publisher	NA_Sales	EU_Sales	J
22	Call of Duty: Black Ops	X360	2010.0	Shooter	Activision	9.68	3.67	
25	Call of Duty: Black Ops II	PS3	2012.0	Shooter	Activision	4.99	5.73	
26	Call of Duty: Black Ops II	X360	2012.0	Shooter	Activision	8.25	4.23	
27	Call of Duty: Modern Warfare 2	X360	2009.0	Shooter	Activision	8.52	3.58	
28	Call of Duty: Modern Warfare 3	PS3	2011.0	Shooter	Activision	5.54	5.73	
29	Grand Theft Auto III	PS2	2001.0	Action	Take-Two Interactive	6.99	4.51	
30	Super Smash Bros. Brawl	Wii	2008.0	Fighting	Nintendo	6.61	2.54	
31	Call of Duty: Black Ops	PS3	2010.0	Shooter	Activision	5.99	4.36	
32	Mario Kart 7	3DS	2011.0	Racing	Nintendo	4.77	3.93	
33	Grand Theft Auto V	PS4	2014.0	Action	Take-Two Interactive	3.88	6.04	
34	Animal Crossing: Wild World	DS	2005.0	Simulation	Nintendo	2.50	3.45	
35	Halo 3	X360	2007.0	Shooter	Microsoft Game Studios	7.97	2.81	
...	...	...	...	...	...	...	...	
1986	Teenage Mutant Ninja Turtles	GBA	2003.0	Action	Unknown	0.67	0.25	
1987	Bayonetta	X360	2009.0	Action	Sega	0.51	0.25	
1988	MLB 07: The Show	PS2	2007.0	Sports	Sony Computer Entertainment	0.77	0.03	
1989	Cars	GC	2006.0	Racing	THQ	0.72	0.19	
1990	Build-A-Bear Workshop	DS	2007.0	Simulation	Game Factory	0.85	0.01	
1991	Overwatch	XOne	2016.0	Shooter	Activision	0.57	0.27	
1992	Kingdom Hearts Re:coded	DS	2010.0	Role-Playing	Square Enix	0.53	0.09	
1993	Operation Flashpoint: Dragon Rising	X360	2009.0	Shooter	Codemasters	0.36	0.45	
1995	Wipeout: In The Zone	X360	2011.0	Misc	Activision	0.87	0.00	

	Name	Platform	Year_of_Release	Genre	Publisher	NA_Sales	EU_Sales	J
1997	Lego Batman 3: Beyond Gotham	PS4	2014.0	Action	Warner Bros. Interactive Entertainment	0.37	0.40	
1998	Assassin's Creed III	PC	2012.0	Action	Ubisoft	0.28	0.53	
1999	Teenage Mutant Ninja Turtles	PS2	2003.0	Action	Konami Digital Entertainment	0.45	0.35	
2000	Top Gun: Combat Zones	PS2	2001.0	Simulation	Titus	0.45	0.35	
2001	NCAA Football 11	X360	2010.0	Sports	Electronic Arts	0.86	0.00	
2002	Dragon Age: Inquisition	XOne	2014.0	Role-Playing	Electronic Arts	0.56	0.28	
2003	007: The World is not Enough	PS	2000.0	Action	Electronic Arts	0.51	0.35	
2004	Brothers in Arms: Hell's Highway	PS3	2008.0	Shooter	Ubisoft	0.44	0.34	
2007	Phoenix Wright: Ace Attorney	DS	2005.0	Adventure	Capcom	0.44	0.05	
2008	Borderlands 2	PC	2012.0	Shooter	Take-Two Interactive	0.42	0.41	
2009	Disney Magical World	3DS	2013.0	Adventure	Nintendo	0.17	0.20	
2012	South Park: The Stick of Truth	X360	2014.0	Role-Playing	Ubisoft	0.57	0.26	
2014	MLB 15: The Show	PS4	2015.0	Sports	Sony Computer Entertainment	0.67	0.08	
2015	MLB 12: The Show	PS3	2012.0	Sports	Sony Computer Entertainment	0.86	0.00	
2016	SpongeBob SquarePants: The Yellow Avenger	PSP	2006.0	Action	THQ	0.55	0.21	
2017	Dante's Inferno	X360	2010.0	Action	Electronic Arts	0.63	0.20	
2018	South Park: The Stick of Truth	PS3	2014.0	Role-Playing	Ubisoft	0.43	0.34	
2019	WWE SmackDown vs. Raw 2009	X360	2008.0	Fighting	THQ	0.58	0.26	
2020	Wii Play: Motion	Wii	2011.0	Misc	Nintendo	0.23	0.42	
2021	Sonic Riders	GC	2006.0	Racing	Sega	0.71	0.18	

	Name	Platform	Year_of_Release	Genre	Publisher	NA_Sales	EU_Sales	J
2024	Blue Dragon	X360	2006.0	Role-Playing	Microsoft Game Studios	0.30	0.32	

1467 rows × 16 columns

In [128]:

```
# c) List all Video Games with Global Sales larger than the average with a User and Critic

# Calculate global sales mean

gsm = vgs["Global_Sales"].mean()
print(f"Global Sales mean is equal to: {gsm:.3f}")

# Calculate user score mean

usm = vgs["User_Score"].mean()
print(f"User Score mean is equal to: {usm:.3f}")

# Calculate critic score mean

csm = vgs["Critic_Score"].mean()
print(f"Critic Score mean is equal to: {csm:.3f}")

# Filter the dataframe

vgs_m2c = vgs[(vgs["Global_Sales"]> gsm) & ((vgs["User_Score"]>usm) & (vgs["Critic_Score"]>
vgs_m2c
```

```
Global Sales mean is equal to: 0.920
User Score mean is equal to: 7.236
Critic Score mean is equal to: 71.272
```

Out[128]:

	Name	Platform	Year_of_Release	Genre	Publisher	NA_Sales	EU_Sales
0	Wii Sports	Wii	2006.0	Sports	Nintendo	41.36	28.95
1	Mario Kart Wii	Wii	2008.0	Racing	Nintendo	15.67	12.75
2	Wii Sports Resort	Wii	2009.0	Sports	Nintendo	15.61	10.92
3	New Super Mario Bros.	DS	2006.0	Platform	Nintendo	11.28	9.14
5	New Super Mario Bros. Wii	Wii	2009.0	Platform	Nintendo	14.42	6.94
7	Mario Kart DS	DS	2005.0	Racing	Nintendo	9.71	7.47
8	Wii Fit	Wii	2007.0	Sports	Nintendo	8.92	8.03
9	Wii Fit Plus	Wii	2009.0	Sports	Nintendo	9.01	8.49
11	Grand Theft Auto V	PS3	2013.0	Action	Take-Two Interactive	7.01	9.04
12	Grand Theft Auto: San Andreas	PS2	2004.0	Action	Take-Two Interactive	9.43	0.40
13	Brain Age: Train Your Brain in Minutes a Day	DS	2005.0	Misc	Nintendo	4.74	9.20

	Name	Platform	Year_of_Release	Genre	Publisher	NA_Sales	EU_Sales
15	Grand Theft Auto V	X360	2013.0	Action	Take-Two Interactive	9.65	5.12
16	Grand Theft Auto: Vice City	PS2	2002.0	Action	Take-Two Interactive	8.41	5.49
20	Gran Turismo 3: A-Spec	PS2	2001.0	Racing	Sony Computer Entertainment	6.85	5.09
29	Grand Theft Auto III	PS2	2001.0	Action	Take-Two Interactive	6.99	4.51
30	Super Smash Bros. Brawl	Wii	2008.0	Fighting	Nintendo	6.61	2.54
32	Mario Kart 7	3DS	2011.0	Racing	Nintendo	4.77	3.93
33	Grand Theft Auto V	PS4	2014.0	Action	Take-Two Interactive	3.88	6.04
34	Animal Crossing: Wild World	DS	2005.0	Simulation	Nintendo	2.50	3.45
35	Halo 3	X360	2007.0	Shooter	Microsoft Game Studios	7.97	2.81
36	Super Mario 64	N64	1996.0	Platform	Nintendo	6.91	2.85
38	Gran Turismo 4	PS2	2004.0	Racing	Sony Computer Entertainment	3.01	0.01
40	Super Mario Galaxy	Wii	2007.0	Platform	Nintendo	6.06	3.34
41	Grand Theft Auto IV	X360	2008.0	Action	Take-Two Interactive	6.76	3.07
42	Gran Turismo	PS	1997.0	Racing	Sony Computer Entertainment	4.02	3.87
43	Super Mario 3D Land	3DS	2011.0	Platform	Nintendo	4.89	2.99
44	Gran Turismo 5	PS3	2010.0	Racing	Sony Computer Entertainment	2.96	4.82
46	Grand Theft Auto IV	PS3	2008.0	Action	Take-Two Interactive	4.76	3.69
50	Just Dance 3	Wii	2011.0	Misc	Ubisoft	5.95	3.11
51	Mario Kart 64	N64	1996.0	Racing	Nintendo	5.55	1.94
...	...	...	...	...	...	...	...
1921	ESPN NBA 2K5	XB	2004.0	Sports	Global Star	0.88	0.04

	Name	Platform	Year_of_Release	Genre	Publisher	NA_Sales	EU_Sales
1925	The Witcher 2: Assassins of Kings	PC	2011.0	Action	Namco Bandai Games	0.25	0.56
1929	Tom Clancy's Splinter Cell: Chaos Theory	PS2	2005.0	Action	Ubisoft	0.36	0.45
1932	PlayStation All-Stars Battle Royale	PS3	2012.0	Action	Sony Computer Entertainment	0.50	0.27
1938	Kessen	PS2	2000.0	Strategy	Electronic Arts	0.27	0.21
1941	Driver: San Francisco	PS3	2011.0	Racing	Ubisoft	0.24	0.52
1946	Army of Two: The 40th Day	X360	2010.0	Shooter	Electronic Arts	0.62	0.24
1953	The SpongeBob SquarePants Movie	GC	2004.0	Platform	THQ	0.73	0.19
1956	Castlevania: Lament of Innocence	PS2	2003.0	Action	Konami Digital Entertainment	0.46	0.36
1958	Tom Clancy's Splinter Cell: Blacklist	PS3	2013.0	Action	Ubisoft	0.34	0.39
1960	Star Fox 64 3D	3DS	2011.0	Shooter	Nintendo	0.48	0.27
1967	Brutal Legend	PS3	2009.0	Action	Electronic Arts	0.54	0.27
1970	Bully	PS2	2006.0	Action	Take-Two Interactive	0.75	0.03
1971	Prince of Persia: The Sands of Time	XB	2003.0	Action	Ubisoft	0.57	0.33
1974	MLB 06: The Show	PS2	2006.0	Sports	Sony Computer Entertainment	0.78	0.03
1976	Naruto Shippuden: Ultimate Ninja Storm 3	PS3	2013.0	Fighting	Namco Bandai Games	0.32	0.32
1978	Tony Hawk's Pro Skater 4	XB	2002.0	Sports	Activision	0.59	0.30
1984	Guitar Hero: Metallica	Wii	2009.0	Misc	Activision	0.40	0.43
1987	Bayonetta	X360	2009.0	Action	Sega	0.51	0.25
1988	MLB 07: The Show	PS2	2007.0	Sports	Sony Computer Entertainment	0.77	0.03

	Name	Platform	Year_of_Release	Genre	Publisher	NA_Sales	EU_Sales
1993	Operation Flashpoint: Dragon Rising	X360	2009.0	Shooter	Codemasters	0.36	0.45
2004	Brothers in Arms: Hell's Highway	PS3	2008.0	Shooter	Ubisoft	0.44	0.34
2007	Phoenix Wright: Ace Attorney	DS	2005.0	Adventure	Capcom	0.44	0.05
2008	Borderlands 2	PC	2012.0	Shooter	Take-Two Interactive	0.42	0.41
2012	South Park: The Stick of Truth	X360	2014.0	Role-Playing	Ubisoft	0.57	0.26
2014	MLB 15: The Show	PS4	2015.0	Sports	Sony Computer Entertainment	0.67	0.08
2015	MLB 12: The Show	PS3	2012.0	Sports	Sony Computer Entertainment	0.86	0.00
2017	Dante's Inferno	X360	2010.0	Action	Electronic Arts	0.63	0.20
2018	South Park: The Stick of Truth	PS3	2014.0	Role-Playing	Ubisoft	0.43	0.34
2024	Blue Dragon	X360	2006.0	Role-Playing	Microsoft Game Studios	0.30	0.32

917 rows × 16 columns

Data Manipulation & Cleaning

In [129]:

```
# Manipulating data

eu_metro_areas["GDP"]/eu_metro_areas["Population"]
eu_metro_areas[["City","GDP per Capita"]]
```

Out[129]:

```
0     61.512605
1     58.173913
2     45.217391
3     39.655172
4     35.613682
5     33.179724
6     42.500000
7     55.963303
8     40.163934
9     23.940150
10    19.114688
11    22.173913
dtype: float64
```

Out[129]:

	City	GDP per Capita
0	London	61.5
1	Paris	62.4
2	Moscow	45.2
3	Madrid	39.7
4	Barcelona	35.6
5	Rome	41.6
6	Milan	44.2
7	Vienna	56.0
8	Lisbon	40.2
9	Athens	23.9
10	Berlin	19.1
11	Bucharest	23.3

In [130]:

```
z=eu_metro_areas["Population"].apply(np.sqrt)
z
eu_metro_areas["Population"].apply(lambda d: d**2)
eu_metro_areas["High Population"]=eu_metro_areas["Population"].apply(lambda x: ("Yes" if x>
eu_metro_areas
eu_metro_areas["Population"]/100
```

Out[130]:

```
0     3.449638
1     3.391165
2     3.391165
3     2.408319
4     2.229350
5     2.083267
6     1.788854
7     1.476482
8     1.562050
9     2.002498
10    2.229350
11    1.516575
Name: Population, dtype: float64
```

Out[130]:

```
0     141.6100
1     132.2500
2     132.2500
3      33.6400
4      24.7009
5      18.8356
6      10.2400
7       4.7524
8       5.9536
9      16.0801
10     24.7009
11      5.2900
Name: Population, dtype: float64
```

Out[130]:

	City	Rank	GDP	Population	GDP per Capita	Eurozone	High Population
0	London	1	732	11.90	61.5	N	Yes
1	Paris	2	669	11.50	62.4	Y	Yes
2	Moscow	3	520	11.50	45.2	N	Yes
3	Madrid	4	230	5.80	39.7	Y	No
4	Barcelona	5	177	4.97	35.6	Y	No
5	Rome	6	144	4.34	41.6	Y	No
6	Milan	7	136	3.20	44.2	Y	No
7	Vienna	8	122	2.18	56.0	Y	No
8	Lisbon	9	98	2.44	40.2	Y	No
9	Athens	10	96	4.01	23.9	Y	No
10	Berlin	11	95	4.97	19.1	Y	No

	City	Rank	GDP	Population	GDP per Capita	Eurozone	High Population
11	Bucharest	12	51	2.30	23.3	N	No

Out[130]:

```
0     0.1190
1     0.1150
2     0.1150
3     0.0580
4     0.0497
5     0.0434
6     0.0320
7     0.0218
8     0.0244
9     0.0401
10    0.0497
11    0.0230
Name: Population, dtype: float64
```

In [131]:

```
# We can also use full functions

eu_metro_areas.drop("High Population", axis=1, inplace=True)

def high_population(x):
    if x>=10:
        return "Yes"
    else:
        return "No"

eu_metro_areas["High Population"]=eu_metro_areas["Population"].apply(high_population)
eu_metro_areas
```

Out[131]:

	City	Rank	GDP	Population	GDP per Capita	Eurozone	High Population
0	London	1	732	11.90	61.5	N	Yes
1	Paris	2	669	11.50	62.4	Y	Yes
2	Moscow	3	520	11.50	45.2	N	Yes
3	Madrid	4	230	5.80	39.7	Y	No
4	Barcelona	5	177	4.97	35.6	Y	No
5	Rome	6	144	4.34	41.6	Y	No
6	Milan	7	136	3.20	44.2	Y	No
7	Vienna	8	122	2.18	56.0	Y	No
8	Lisbon	9	98	2.44	40.2	Y	No
9	Athens	10	96	4.01	23.9	Y	No
10	Berlin	11	95	4.97	19.1	Y	No
11	Bucharest	12	51	2.30	23.3	N	No

In [132]:

```
# encoding

z=eu_metro_areas.copy()
map_eurozone={"Y":1,"N":0}
map_highpopulation={"Yes":1,"No":0}

z["Eurozone"]=z["Eurozone"].map(map_eurozone)
z["High Population"]=z["High Population"].map(map_highpopulation)
z
```

Out[132]:

	City	Rank	GDP	Population	GDP per Capita	Eurozone	High Population
0	London	1	732	11.90	61.5	0	1
1	Paris	2	669	11.50	62.4	1	1
2	Moscow	3	520	11.50	45.2	0	1
3	Madrid	4	230	5.80	39.7	1	0
4	Barcelona	5	177	4.97	35.6	1	0
5	Rome	6	144	4.34	41.6	1	0
6	Milan	7	136	3.20	44.2	1	0
7	Vienna	8	122	2.18	56.0	1	0
8	Lisbon	9	98	2.44	40.2	1	0
9	Athens	10	96	4.01	23.9	1	0
10	Berlin	11	95	4.97	19.1	1	0
11	Bucharest	12	51	2.30	23.3	0	0

In [133]:

```
# Adding a new colum

eu_metro_areas["GDP Norm"]=eu_metro_areas["GDP"]/eu_metro_areas["GDP"].max()
eu_metro_areas
```

Out[133]:

	City	Rank	GDP	Population	GDP per Capita	Eurozone	High Population	GDP Norm
0	London	1	732	11.90	61.5	N	Yes	1.000000
1	Paris	2	669	11.50	62.4	Y	Yes	0.913934
2	Moscow	3	520	11.50	45.2	N	Yes	0.710383
3	Madrid	4	230	5.80	39.7	Y	No	0.314208
4	Barcelona	5	177	4.97	35.6	Y	No	0.241803
5	Rome	6	144	4.34	41.6	Y	No	0.196721
6	Milan	7	136	3.20	44.2	Y	No	0.185792
7	Vienna	8	122	2.18	56.0	Y	No	0.166667
8	Lisbon	9	98	2.44	40.2	Y	No	0.133880
9	Athens	10	96	4.01	23.9	Y	No	0.131148
10	Berlin	11	95	4.97	19.1	Y	No	0.129781
11	Bucharest	12	51	2.30	23.3	N	No	0.069672

In [134]:

```
# Dropping a column

eu_metro_areas.drop("GDP Norm", axis=1, inplace=True)
eu_metro_areas
```

Out[134]:

	City	Rank	GDP	Population	GDP per Capita	Eurozone	High Population
0	London	1	732	11.90	61.5	N	Yes
1	Paris	2	669	11.50	62.4	Y	Yes
2	Moscow	3	520	11.50	45.2	N	Yes
3	Madrid	4	230	5.80	39.7	Y	No
4	Barcelona	5	177	4.97	35.6	Y	No
5	Rome	6	144	4.34	41.6	Y	No
6	Milan	7	136	3.20	44.2	Y	No
7	Vienna	8	122	2.18	56.0	Y	No
8	Lisbon	9	98	2.44	40.2	Y	No
9	Athens	10	96	4.01	23.9	Y	No
10	Berlin	11	95	4.97	19.1	Y	No
11	Bucharest	12	51	2.30	23.3	N	No

In [135]:

```
# Dropping a column

eu_metro_areas.drop("High Population", axis=1)
eu_metro_areas
```

Out[135]:

	City	Rank	GDP	Population	GDP per Capita	Eurozone
0	London	1	732	11.90	61.5	N
1	Paris	2	669	11.50	62.4	Y
2	Moscow	3	520	11.50	45.2	N
3	Madrid	4	230	5.80	39.7	Y
4	Barcelona	5	177	4.97	35.6	Y
5	Rome	6	144	4.34	41.6	Y
6	Milan	7	136	3.20	44.2	Y
7	Vienna	8	122	2.18	56.0	Y
8	Lisbon	9	98	2.44	40.2	Y
9	Athens	10	96	4.01	23.9	Y
10	Berlin	11	95	4.97	19.1	Y
11	Bucharest	12	51	2.30	23.3	N

Out[135]:

	City	Rank	GDP	Population	GDP per Capita	Eurozone	High Population
0	London	1	732	11.90	61.5	N	Yes
1	Paris	2	669	11.50	62.4	Y	Yes
2	Moscow	3	520	11.50	45.2	N	Yes
3	Madrid	4	230	5.80	39.7	Y	No
4	Barcelona	5	177	4.97	35.6	Y	No
5	Rome	6	144	4.34	41.6	Y	No
6	Milan	7	136	3.20	44.2	Y	No
7	Vienna	8	122	2.18	56.0	Y	No
8	Lisbon	9	98	2.44	40.2	Y	No
9	Athens	10	96	4.01	23.9	Y	No
10	Berlin	11	95	4.97	19.1	Y	No
11	Bucharest	12	51	2.30	23.3	N	No

In [136]:

```
# Append a row

z=eu_metro_areas.append({"City": "Nowhere","Eurozone":"N","GDP":100,"GDP per Capita": 10,"P
                         ignore_index = True)
z
```

Out[136]:

	City	Rank	GDP	Population	GDP per Capita	Eurozone	High Population
0	London	1	732	11.90	61.5	N	Yes
1	Paris	2	669	11.50	62.4	Y	Yes
2	Moscow	3	520	11.50	45.2	N	Yes
3	Madrid	4	230	5.80	39.7	Y	No
4	Barcelona	5	177	4.97	35.6	Y	No
5	Rome	6	144	4.34	41.6	Y	No
6	Milan	7	136	3.20	44.2	Y	No
7	Vienna	8	122	2.18	56.0	Y	No
8	Lisbon	9	98	2.44	40.2	Y	No
9	Athens	10	96	4.01	23.9	Y	No
10	Berlin	11	95	4.97	19.1	Y	No
11	Bucharest	12	51	2.30	23.3	N	No
12	Nowhere	13	100	10.00	10.0	N	NaN

In [137]:

```
# Drop a row

z.drop(12, axis=0, inplace = True)
z
```

Out[137]:

	City	Rank	GDP	Population	GDP per Capita	Eurozone	High Population
0	London	1	732	11.90	61.5	N	Yes
1	Paris	2	669	11.50	62.4	Y	Yes
2	Moscow	3	520	11.50	45.2	N	Yes
3	Madrid	4	230	5.80	39.7	Y	No
4	Barcelona	5	177	4.97	35.6	Y	No
5	Rome	6	144	4.34	41.6	Y	No
6	Milan	7	136	3.20	44.2	Y	No
7	Vienna	8	122	2.18	56.0	Y	No
8	Lisbon	9	98	2.44	40.2	Y	No
9	Athens	10	96	4.01	23.9	Y	No
10	Berlin	11	95	4.97	19.1	Y	No
11	Bucharest	12	51	2.30	23.3	N	No

In [138]:

```
# Dropping null values

z=eu_metro_areas.append({"City": "Nowhere","Eurozone":"N","Rank":13},
                        ignore_index = True)
# if you don't use copy, you'll be referencing the same data frame
# a pointer and not a different data frame
z1=z.copy()
z

z.dropna(inplace=True)
z

z1.dropna( subset=["GDP"], inplace=True)
z1
```

Out[138]:

	City	Rank	GDP	Population	GDP per Capita	Eurozone	High Population
0	London	1	732.0	11.90	61.5	N	Yes
1	Paris	2	669.0	11.50	62.4	Y	Yes
2	Moscow	3	520.0	11.50	45.2	N	Yes
3	Madrid	4	230.0	5.80	39.7	Y	No
4	Barcelona	5	177.0	4.97	35.6	Y	No
5	Rome	6	144.0	4.34	41.6	Y	No
6	Milan	7	136.0	3.20	44.2	Y	No
7	Vienna	8	122.0	2.18	56.0	Y	No
8	Lisbon	9	98.0	2.44	40.2	Y	No
9	Athens	10	96.0	4.01	23.9	Y	No
10	Berlin	11	95.0	4.97	19.1	Y	No
11	Bucharest	12	51.0	2.30	23.3	N	No
12	Nowhere	13	NaN	NaN	NaN	N	NaN

Out[138]:

	City	Rank	GDP	Population	GDP per Capita	Eurozone	High Population
0	London	1	732.0	11.90	61.5	N	Yes
1	Paris	2	669.0	11.50	62.4	Y	Yes
2	Moscow	3	520.0	11.50	45.2	N	Yes
3	Madrid	4	230.0	5.80	39.7	Y	No
4	Barcelona	5	177.0	4.97	35.6	Y	No
5	Rome	6	144.0	4.34	41.6	Y	No
6	Milan	7	136.0	3.20	44.2	Y	No
7	Vienna	8	122.0	2.18	56.0	Y	No
8	Lisbon	9	98.0	2.44	40.2	Y	No
9	Athens	10	96.0	4.01	23.9	Y	No

	City	Rank	GDP	Population	GDP per Capita	Eurozone	High Population
10	Berlin	11	95.0	4.97	19.1	Y	No

Out[138]:

	City	Rank	GDP	Population	GDP per Capita	Eurozone	High Population
0	London	1	732.0	11.90	61.5	N	Yes
1	Paris	2	669.0	11.50	62.4	Y	Yes
2	Moscow	3	520.0	11.50	45.2	N	Yes
3	Madrid	4	230.0	5.80	39.7	Y	No
4	Barcelona	5	177.0	4.97	35.6	Y	No
5	Rome	6	144.0	4.34	41.6	Y	No
6	Milan	7	136.0	3.20	44.2	Y	No
7	Vienna	8	122.0	2.18	56.0	Y	No
8	Lisbon	9	98.0	2.44	40.2	Y	No
9	Athens	10	96.0	4.01	23.9	Y	No
10	Berlin	11	95.0	4.97	19.1	Y	No
11	Bucharest	12	51.0	2.30	23.3	N	No

In [139]:

```
# Imputing null values with the mean of the column

z=eu_metro_areas.append({"City": "Nowhere","Eurozone":"N","Rank":13},
                        ignore_index = True)
z
z["GDP"].fillna(z["GDP"].mean(), inplace=True)
z
```

Out[139]:

	City	Rank	GDP	Population	GDP per Capita	Eurozone	High Population
0	London	1	732.0	11.90	61.5	N	Yes
1	Paris	2	669.0	11.50	62.4	Y	Yes
2	Moscow	3	520.0	11.50	45.2	N	Yes
3	Madrid	4	230.0	5.80	39.7	Y	No
4	Barcelona	5	177.0	4.97	35.6	Y	No
5	Rome	6	144.0	4.34	41.6	Y	No
6	Milan	7	136.0	3.20	44.2	Y	No
7	Vienna	8	122.0	2.18	56.0	Y	No
8	Lisbon	9	98.0	2.44	40.2	Y	No
9	Athens	10	96.0	4.01	23.9	Y	No
10	Berlin	11	95.0	4.97	19.1	Y	No
11	Bucharest	12	51.0	2.30	23.3	N	No
12	Nowhere	13	NaN	NaN	NaN	N	NaN

Out[139]:

	City	Rank	GDP	Population	GDP per Capita	Eurozone	High Population
0	London	1	732.000000	11.90	61.5	N	Yes
1	Paris	2	669.000000	11.50	62.4	Y	Yes
2	Moscow	3	520.000000	11.50	45.2	N	Yes
3	Madrid	4	230.000000	5.80	39.7	Y	No
4	Barcelona	5	177.000000	4.97	35.6	Y	No
5	Rome	6	144.000000	4.34	41.6	Y	No
6	Milan	7	136.000000	3.20	44.2	Y	No
7	Vienna	8	122.000000	2.18	56.0	Y	No
8	Lisbon	9	98.000000	2.44	40.2	Y	No
9	Athens	10	96.000000	4.01	23.9	Y	No
10	Berlin	11	95.000000	4.97	19.1	Y	No
11	Bucharest	12	51.000000	2.30	23.3	N	No
12	Nowhere	13	255.833333	NaN	NaN	N	NaN

In [140]:

```
# Replacing weird characters

z.replace({"Nowhere":"A city"}, inplace=True)
z
```

Out[140]:

	City	Rank	GDP	Population	GDP per Capita	Eurozone	High Population
0	London	1	732.000000	11.90	61.5	N	Yes
1	Paris	2	669.000000	11.50	62.4	Y	Yes
2	Moscow	3	520.000000	11.50	45.2	N	Yes
3	Madrid	4	230.000000	5.80	39.7	Y	No
4	Barcelona	5	177.000000	4.97	35.6	Y	No
5	Rome	6	144.000000	4.34	41.6	Y	No
6	Milan	7	136.000000	3.20	44.2	Y	No
7	Vienna	8	122.000000	2.18	56.0	Y	No
8	Lisbon	9	98.000000	2.44	40.2	Y	No
9	Athens	10	96.000000	4.01	23.9	Y	No
10	Berlin	11	95.000000	4.97	19.1	Y	No
11	Bucharest	12	51.000000	2.30	23.3	N	No
12	A city	13	255.833333	NaN	NaN	N	NaN

In [141]:

```
# Changing column names

z.rename(columns={"GDP_per_Capita":"GDP/capita", "High Population":"Big"}, inplace=True)
z
```

Out[141]:

	City	Rank	GDP	Population	GDP per Capita	Eurozone	Big
0	London	1	732.000000	11.90	61.5	N	Yes
1	Paris	2	669.000000	11.50	62.4	Y	Yes
2	Moscow	3	520.000000	11.50	45.2	N	Yes
3	Madrid	4	230.000000	5.80	39.7	Y	No
4	Barcelona	5	177.000000	4.97	35.6	Y	No
5	Rome	6	144.000000	4.34	41.6	Y	No
6	Milan	7	136.000000	3.20	44.2	Y	No
7	Vienna	8	122.000000	2.18	56.0	Y	No
8	Lisbon	9	98.000000	2.44	40.2	Y	No
9	Athens	10	96.000000	4.01	23.9	Y	No
10	Berlin	11	95.000000	4.97	19.1	Y	No
11	Bucharest	12	51.000000	2.30	23.3	N	No
12	A city	13	255.833333	NaN	NaN	N	NaN

In [142]:

```
# Writing the result in a csv o Excell file

eu_metro_areas.to_csv('My_metro_areas.csv', sep=',')

eu_metro_areas.to_excel("My_metro_areas.xlsx")
```

Sorting & Grouping

In [143]:

```
# Sorting

eu_metro_areas.sort_values(by="Population", ascending = False)
eu_metro_areas.sort_values(by="City", ascending = True)
eu_metro_areas
```

Out[143]:

	City	Rank	GDP	Population	GDP per Capita	Eurozone	High Population
0	London	1	732	11.90	61.5	N	Yes
1	Paris	2	669	11.50	62.4	Y	Yes
2	Moscow	3	520	11.50	45.2	N	Yes
3	Madrid	4	230	5.80	39.7	Y	No
4	Barcelona	5	177	4.97	35.6	Y	No
10	Berlin	11	95	4.97	19.1	Y	No
5	Rome	6	144	4.34	41.6	Y	No
9	Athens	10	96	4.01	23.9	Y	No
6	Milan	7	136	3.20	44.2	Y	No
8	Lisbon	9	98	2.44	40.2	Y	No
11	Bucharest	12	51	2.30	23.3	N	No
7	Vienna	8	122	2.18	56.0	Y	No

Out[143]:

	City	Rank	GDP	Population	GDP per Capita	Eurozone	High Population
9	Athens	10	96	4.01	23.9	Y	No
4	Barcelona	5	177	4.97	35.6	Y	No
10	Berlin	11	95	4.97	19.1	Y	No
11	Bucharest	12	51	2.30	23.3	N	No
8	Lisbon	9	98	2.44	40.2	Y	No
0	London	1	732	11.90	61.5	N	Yes
3	Madrid	4	230	5.80	39.7	Y	No
6	Milan	7	136	3.20	44.2	Y	No
2	Moscow	3	520	11.50	45.2	N	Yes
1	Paris	2	669	11.50	62.4	Y	Yes
5	Rome	6	144	4.34	41.6	Y	No
7	Vienna	8	122	2.18	56.0	Y	No

Out[143]:

	City	Rank	GDP	Population	GDP per Capita	Eurozone	High Population
0	London	1	732	11.90	61.5	N	Yes
1	Paris	2	669	11.50	62.4	Y	Yes
2	Moscow	3	520	11.50	45.2	N	Yes
3	Madrid	4	230	5.80	39.7	Y	No
4	Barcelona	5	177	4.97	35.6	Y	No
5	Rome	6	144	4.34	41.6	Y	No
6	Milan	7	136	3.20	44.2	Y	No
7	Vienna	8	122	2.18	56.0	Y	No
8	Lisbon	9	98	2.44	40.2	Y	No
9	Athens	10	96	4.01	23.9	Y	No
10	Berlin	11	95	4.97	19.1	Y	No
11	Bucharest	12	51	2.30	23.3	N	No

In [144]:

```
eu_metro_areas.sort_values(by="Population", ascending = False, inplace=True)
eu_metro_areas
eu_metro_areas.sort_index(axis=0, ascending=True, inplace=True)
eu_metro_areas
```

Out[144]:

	City	Rank	GDP	Population	GDP per Capita	Eurozone	High Population
0	London	1	732	11.90	61.5	N	Yes
1	Paris	2	669	11.50	62.4	Y	Yes
2	Moscow	3	520	11.50	45.2	N	Yes
3	Madrid	4	230	5.80	39.7	Y	No
4	Barcelona	5	177	4.97	35.6	Y	No
10	Berlin	11	95	4.97	19.1	Y	No
5	Rome	6	144	4.34	41.6	Y	No
9	Athens	10	96	4.01	23.9	Y	No
6	Milan	7	136	3.20	44.2	Y	No
8	Lisbon	9	98	2.44	40.2	Y	No
11	Bucharest	12	51	2.30	23.3	N	No
7	Vienna	8	122	2.18	56.0	Y	No

Out[144]:

	City	Rank	GDP	Population	GDP per Capita	Eurozone	High Population
0	London	1	732	11.90	61.5	N	Yes
1	Paris	2	669	11.50	62.4	Y	Yes
2	Moscow	3	520	11.50	45.2	N	Yes
3	Madrid	4	230	5.80	39.7	Y	No
4	Barcelona	5	177	4.97	35.6	Y	No
5	Rome	6	144	4.34	41.6	Y	No
6	Milan	7	136	3.20	44.2	Y	No
7	Vienna	8	122	2.18	56.0	Y	No
8	Lisbon	9	98	2.44	40.2	Y	No
9	Athens	10	96	4.01	23.9	Y	No
10	Berlin	11	95	4.97	19.1	Y	No
11	Bucharest	12	51	2.30	23.3	N	No

In [145]:

```
# Grouping

eu_metro_areas[["Eurozone", "GDP","Population"]].groupby(["Eurozone"]).sum()
eu_metro_areas[["Eurozone", "GDP","Population"]].groupby(["Eurozone"]).mean()
```

Out[145]:

Eurozone	GDP	Population
N	1303	25.70
Y	1767	43.41

Out[145]:

Eurozone	GDP	Population
N	434.333333	8.566667
Y	196.333333	4.823333

Mission 3

a) Short the names of Publishers (ex. "Warner Bros. Interactive Entertainment" by "Warner Bros")
b) List the sales of Nintendo Games grouped by year
c) List sales grouped by Publisher and year

In [146]:

```
# a) Short the names of Publishers (ex. "Warner Bros. Interactive Entertainment" by "Warner
# Also done for two other studios ("Microsoft Game Studios" to "Microsoft" and "Sony Comput

vgs["Publisher"] = vgs["Publisher"].replace("Warner Bros. Interactive Entertainment", "Warr
vgs["Publisher"] = vgs["Publisher"].replace("Microsoft Game Studios", "Microsoft")
vgs["Publisher"] = vgs["Publisher"].replace("Sony Computer Entertainment", "Sony")
vgs
```

Out[146]:

	Name	Platform	Year_of_Release	Genre	Publisher	NA_Sales	EU_Sale
0	Wii Sports	Wii	2006.0	Sports	Nintendo	41.36	28.9
1	Mario Kart Wii	Wii	2008.0	Racing	Nintendo	15.67	12.7
2	Wii Sports Resort	Wii	2009.0	Sports	Nintendo	15.61	10.9
3	New Super Mario Bros.	DS	2006.0	Platform	Nintendo	11.28	9.1
4	Wii Play	Wii	2006.0	Misc	Nintendo	13.96	9.1
5	New Super Mario Bros. Wii	Wii	2009.0	Platform	Nintendo	14.42	6.9
7	Mario Kart DS	DS	2005.0	Racing	Nintendo	9.71	7.4
8	Wii Fit	Wii	2007.0	Sports	Nintendo	8.92	8.0
9	Wii Fit Plus	Wii	2009.0	Sports	Nintendo	9.01	8.4
10	Kinect Adventures!	X360	2010.0	Misc	Microsoft	14.98	4.8
11	Grand Theft Auto V	PS3	2013.0	Action	Take-Two Interactive	7.01	9.0
12	Grand Theft Auto: San Andreas	PS2	2004.0	Action	Take-Two Interactive	9.43	0.4
13	Brain Age: Train Your Brain in Minutes a Day	DS	2005.0	Misc	Nintendo	4.74	9.2
15	Grand Theft Auto V	X360	2013.0	Action	Take-Two Interactive	9.65	5.1
16	Grand Theft Auto: Vice City	PS2	2002.0	Action	Take-Two Interactive	8.41	5.4
18	Brain Age 2: More Training in Minutes a Day	DS	2005.0	Puzzle	Nintendo	3.43	5.3
20	Gran Turismo 3: A-Spec	PS2	2001.0	Racing	Sony	6.85	5.0
21	Call of Duty: Modern Warfare 3	X360	2011.0	Shooter	Activision	9.03	4.2

	Name	Platform	Year_of_Release	Genre	Publisher	NA_Sales	EU_Sale
22	Call of Duty: Black Ops	X360	2010.0	Shooter	Activision	9.68	3.6
25	Call of Duty: Black Ops II	PS3	2012.0	Shooter	Activision	4.99	5.7
26	Call of Duty: Black Ops II	X360	2012.0	Shooter	Activision	8.25	4.2
27	Call of Duty: Modern Warfare 2	X360	2009.0	Shooter	Activision	8.52	3.5
28	Call of Duty: Modern Warfare 3	PS3	2011.0	Shooter	Activision	5.54	5.7
29	Grand Theft Auto III	PS2	2001.0	Action	Take-Two Interactive	6.99	4.5
30	Super Smash Bros. Brawl	Wii	2008.0	Fighting	Nintendo	6.61	2.5
31	Call of Duty: Black Ops	PS3	2010.0	Shooter	Activision	5.99	4.3
32	Mario Kart 7	3DS	2011.0	Racing	Nintendo	4.77	3.9
33	Grand Theft Auto V	PS4	2014.0	Action	Take-Two Interactive	3.88	6.0
34	Animal Crossing: Wild World	DS	2005.0	Simulation	Nintendo	2.50	3.4
35	Halo 3	X360	2007.0	Shooter	Microsoft	7.97	2.8
...	...	...	...	...	...	...	
15621	Myst	PSP	2006.0	Adventure	Midway Games	0.00	0.0
15623	Sherlock Holmes: The Devil's Daughter	PC	2016.0	Adventure	Bigben Interactive	0.00	0.0
15637	Juiced 2: Hot Import Nights	PC	2007.0	Racing	THQ	0.00	0.0
15638	Dungeon Explorer: Warriors of Ancient Arts	PSP	2007.0	Role-Playing	Rising Star Games	0.01	0.0
15663	Pro Evolution Soccer 2010	PC	2009.0	Sports	Konami Digital Entertainment	0.00	0.0
15666	Hoshigami: Ruining Blue Earth Remix	DS	2007.0	Role-Playing	505 Games	0.00	0.0
15668	Art Academy: Home Studio	WiiU	2015.0	Misc	Nintendo	0.00	0.0
15670	Alone in the Dark	PC	2008.0	Adventure	Atari	0.00	0.0

	Name	Platform	Year_of_Release	Genre	Publisher	NA_Sales	EU_Sale
15671	Clive Barker's Jericho	PC	2007.0	Shooter	Codemasters	0.00	0.0
15680	Madagascar: Escape 2 Africa	PC	2008.0	Action	Activision	0.01	0.0
15682	Wade Hixton's Counter Punch	GBA	2004.0	Sports	Destination Software, Inc	0.01	0.0
15685	Sega Rally Revo	PC	2007.0	Racing	Sega	0.00	0.0
15699	Egg Mania: Eggstreme Madness	GC	2002.0	Puzzle	Kemco	0.01	0.0
15706	The Eye of Judgment: Legends	PSP	2010.0	Strategy	Sony	0.00	0.0
15717	King's Bounty: Armored Princess	PC	2009.0	Role-Playing	1C Company	0.00	0.0
15722	Transformers: Fall of Cybertron	PC	2012.0	Action	Activision	0.01	0.0
15727	Micro Machines V4	PS2	2006.0	Racing	Codemasters	0.01	0.0
15750	Legacy of Kain: Defiance	PC	2003.0	Action	Eidos Interactive	0.00	0.0
15751	Dragon Ball Z for Kinect	X360	2012.0	Fighting	Namco Bandai Games	0.01	0.0
15761	Super Stardust Ultra VR	PS4	2016.0	Shooter	Sony	0.00	0.0
15762	Karnaaj Rally	GBA	2003.0	Racing	Jaleco	0.01	0.0
15772	S√©bastien Loeb Rally Evo	XOne	2016.0	Racing	Milestone S.r.l	0.00	0.0
15774	Trine	PC	2009.0	Action	Nobilis	0.00	0.0
15785	Hospital Tycoon	PC	2007.0	Strategy	Codemasters	0.00	0.0
15793	Quantum Break	PC	2016.0	Action	Microsoft	0.00	0.0
15800	E.T. The Extra-Terrestrial	GBA	2001.0	Action	NewKidCo	0.01	0.0

	Name	Platform	Year_of_Release	Genre	Publisher	NA_Sales	EU_Sale
15813	Mortal Kombat: Deadly Alliance	GBA	2002.0	Fighting	Midway Games	0.01	0.0
15834	Breach	PC	2011.0	Shooter	Destineer	0.01	0.0
15839	STORM: Frontline Nation	PC	2011.0	Strategy	Unknown	0.00	0.0
15845	Metal Gear Solid V: Ground Zeroes	PC	2014.0	Action	Konami Digital Entertainment	0.00	0.0

5596 rows × 16 columns

In [147]:

```
#b) List the sales of Nintendo Games grouped by year

vgs_nintendo = vgs[vgs["Publisher"] == "Nintendo"]
vgs_m3b = vgs_nintendo[["Year_of_Release", "NA_Sales", "EU_Sales", "JP_Sales", "Other_Sales
vgs_m3b
```

Out[147]:

Year_of_Release	NA_Sales	EU_Sales	JP_Sales	Other_Sales	Global_Sales
1996.0	15.00	5.61	4.78	0.44	25.82
1997.0	12.78	4.11	2.64	0.34	19.85
1998.0	9.27	4.34	3.24	0.37	17.22
1999.0	9.91	2.53	4.44	0.20	17.08
2000.0	9.38	3.25	6.29	0.59	19.53
2001.0	18.84	6.68	7.01	1.19	33.69
2002.0	16.64	5.32	4.76	0.87	27.55
2003.0	10.13	3.30	4.95	0.51	18.87
2004.0	14.13	5.00	8.35	0.83	28.28
2005.0	32.26	30.30	24.69	7.07	94.36
2006.0	82.76	54.35	22.95	16.62	176.62
2007.0	39.39	29.68	15.47	7.93	92.47
2008.0	34.43	22.84	17.46	6.60	81.38
2009.0	45.20	30.82	18.99	8.31	103.32
2010.0	15.98	11.30	8.98	2.79	39.05
2011.0	15.72	10.32	7.92	2.62	36.59
2012.0	16.56	11.34	13.06	2.51	43.50
2013.0	9.67	8.81	7.05	1.64	27.16
2014.0	7.74	5.59	6.49	1.29	21.08
2015.0	7.68	5.96	5.84	1.28	20.81
2016.0	1.13	0.63	0.81	0.18	2.75

In [148]:

```
#c) List sales grouped by Publisher and year

vgs_m3c = vgs[["Publisher", "Year_of_Release", "NA_Sales", "EU_Sales", "JP_Sales", "Other_S
vgs_m3c.groupby(["Publisher", "Year_of_Release"]).sum()
```

Out[148]:

Publisher	Year_of_Release	NA_Sales	EU_Sales	JP_Sales	Other_Sales	Global_Sales
10TACLE Studios	**2006.0**	0.01	0.01	0.00	0.00	0.02
1C Company	**2009.0**	0.00	0.01	0.00	0.00	0.01
	2011.0	0.01	0.03	0.00	0.01	0.05
3DO	**2000.0**	0.74	0.33	0.00	0.05	1.13
	2001.0	0.26	0.21	0.00	0.08	0.53
	2002.0	0.25	0.19	0.00	0.07	0.49
	2003.0	0.18	0.14	0.00	0.05	0.36
505 Games	**2003.0**	0.05	0.04	0.00	0.01	0.10
	2004.0	0.28	0.22	0.00	0.07	0.58
	2005.0	0.29	0.09	0.08	0.04	0.52
	2006.0	3.32	1.94	0.16	0.59	6.04
	2007.0	3.67	2.63	0.19	0.75	7.25
	2008.0	0.54	0.00	0.01	0.05	0.61
	2009.0	1.48	0.93	0.06	0.29	2.75
	2010.0	2.26	0.85	0.00	0.31	3.42
	2011.0	0.68	0.53	0.00	0.18	1.39
	2012.0	0.41	0.45	0.06	0.14	1.07
	2013.0	1.01	0.81	0.09	0.25	2.13
	2014.0	0.43	0.53	0.01	0.15	1.12
	2015.0	0.03	0.01	0.00	0.00	0.04
	2016.0	0.03	0.28	0.00	0.05	0.36
5pb	**2011.0**	0.00	0.00	0.04	0.00	0.04
AQ Interactive	**2008.0**	0.19	0.01	0.00	0.02	0.22
ASCII Entertainment	**1997.0**	0.11	0.07	0.29	0.03	0.50
Acclaim Entertainment	**1997.0**	1.84	0.52	0.09	0.05	2.50
	2000.0	1.15	0.24	0.00	0.04	1.43
	2001.0	4.39	3.12	0.08	0.84	8.44
	2002.0	2.98	1.57	0.00	0.42	5.00
	2003.0	1.43	0.65	0.00	0.17	2.23
	2004.0	0.43	0.22	0.00	0.06	0.72
...	**...**	...	...	...	...	...

Publisher	Year_of_Release	NA_Sales	EU_Sales	JP_Sales	Other_Sales	Global_Sales
Warner Bros	**2007.0**	0.73	0.25	0.00	0.15	1.10
	2008.0	5.70	2.93	0.00	1.44	10.06
	2009.0	4.99	2.32	0.05	0.87	8.24
	2010.0	5.72	3.49	0.00	0.99	10.19
	2011.0	11.52	5.05	0.17	2.13	18.85
	2012.0	4.92	3.31	0.02	1.13	9.37
	2013.0	8.00	5.88	0.13	1.96	15.95
	2014.0	4.86	5.01	0.12	1.43	11.40
	2015.0	8.43	5.63	0.27	2.25	16.61
	2016.0	0.40	0.34	0.01	0.07	0.82
XS Games	**2004.0**	0.03	0.01	0.00	0.00	0.04
	2009.0	0.20	0.00	0.00	0.02	0.21
Xicat Interactive	**2002.0**	0.02	0.01	0.00	0.00	0.03
Xplosiv	**2005.0**	0.10	0.07	0.00	0.03	0.20
	2008.0	0.06	0.39	0.00	0.01	0.47
Xseed Games	**2012.0**	0.31	0.00	0.21	0.03	0.56
	2014.0	0.02	0.00	0.04	0.01	0.07
Yacht Club Games	**2015.0**	0.03	0.03	0.00	0.01	0.08
	2016.0	0.01	0.00	0.00	0.00	0.01
Zoo Digital Publishing	**2002.0**	0.04	0.03	0.12	0.01	0.21
	2003.0	0.75	0.57	0.00	0.18	1.50
	2004.0	1.00	0.36	0.00	0.14	1.48
	2005.0	0.05	0.02	0.00	0.00	0.07
	2006.0	0.13	0.00	0.00	0.01	0.15
	2007.0	0.53	0.00	0.00	0.03	0.57
	2009.0	0.12	0.00	0.00	0.01	0.13
Zoo Games	**2008.0**	0.04	0.00	0.00	0.00	0.04
	2009.0	1.21	0.00	0.00	0.09	1.30
bitComposer Games	**2009.0**	0.00	0.03	0.00	0.01	0.03
id Software	**1992.0**	0.02	0.00	0.00	0.00	0.03

878 rows × 5 columns

Pivot Tables

pandas pivot_table explained

	Account	Name	Rep	Manager	Product	Quantity	Price	Status
0	714466	Trantow-Barrows	Craig Booker	Debra Henley	CPU	1	30000	presented
1	714466	Trantow-Barrows	Craig Booker	Debra Henley	Software	1	10000	presented
2	714466	Trantow-Barrows	Craig Booker	Debra Henley	Maintenance	2	5000	pending
3	737550	Fritsch, Russel and Anderson	Craig Booker	Debra Henley	CPU	1	35000	declined
4	146832	Kiehn-Spinka	Daniel Hilton	Debra Henley	CPU	2	65000	won

```
pd.pivot_table(df,
    index=["Manager","Status"],
    columns=["Product"],
    aggfunc=[np.sum],
    values=["Price"],
    fill_value=0,
    margins=True,
    dropna=True)
```

Can also use a dictionary:
`aggfunc={"Quantity":len, "Price":[np.sum,np.mean]}`

		sum				
		Price				
	Product	CPU	Maintenance	Monitor	Software	All
Manager	Status					
Debra Henley	declined	70000	0	0	0	70000
	pending	40000	10000	0	0	50000
	presented	30000	0	0	20000	50000
	won	65000	0	0	0	65000
Fred Anderson	declined	65000	0	0	0	65000
	pending	0	5000	0	0	5000
	presented	30000	0	5000	10000	45000
	won	165000	7000	0	0	172000
All		465000	22000	5000	30000	522000

pbpython.com

In [149]:

```
# Pivot tables

eu_metro_areas.pivot_table(columns=["City"])
```

Out[149]:

City	Athens	Barcelona	Berlin	Bucharest	Lisbon	London	Madrid	Milan	Moscow	Pa
GDP	96.00	177.00	95.00	51.0	98.00	732.0	230.0	136.0	520.0	66
GDP per Capita	23.90	35.60	19.10	23.3	40.20	61.5	39.7	44.2	45.2	6
Population	4.01	4.97	4.97	2.3	2.44	11.9	5.8	3.2	11.5	1
Rank	10.00	5.00	11.00	12.0	9.00	1.0	4.0	7.0	3.0	

In [150]:

```
# Pivot tables
# http://pbpython.com/pandas-pivot-table-explained.html

eu_metro_areas.pivot_table(index=["Eurozone", "City"], values=["Population","GDP"])
```

Out[150]:

		GDP	Population
Eurozone	**City**		
N	**Bucharest**	51	2.30
	London	732	11.90
	Moscow	520	11.50
Y	**Athens**	96	4.01
	Barcelona	177	4.97
	Berlin	95	4.97
	Lisbon	98	2.44
	Madrid	230	5.80
	Milan	136	3.20
	Paris	669	11.50
	Rome	144	4.34
	Vienna	122	2.18

In [151]:

```
eu_metro_areas.pivot_table(columns=["Eurozone"], values=["GDP"], aggfunc=np.sum)
eu_metro_areas.pivot_table(columns=["Eurozone"], values=["GDP"], aggfunc=np.mean)
```

Out[151]:

Eurozone	N	Y
GDP	1303	1767

Out[151]:

Eurozone	N	Y
GDP	434.333333	196.333333

In [152]:

```
eu_metro_areas["Count"]=1
eu_metro_areas.head()

eu_metro_areas.pivot_table(columns=["Eurozone"], values=["Count"], aggfunc=np.sum)
```

Out[152]:

	City	Rank	GDP	Population	GDP per Capita	Eurozone	High Population	Count
0	London	1	732	11.90	61.5	N	Yes	1
1	Paris	2	669	11.50	62.4	Y	Yes	1
2	Moscow	3	520	11.50	45.2	N	Yes	1
3	Madrid	4	230	5.80	39.7	Y	No	1
4	Barcelona	5	177	4.97	35.6	Y	No	1

Out[152]:

Eurozone	N	Y
Count	3	9

Mission 4

a) List the Publishers in the columns and the different sales in the rows
b) Count the number of game titles per publisher
c) Display Platform in columns and sales in rows

In [153]:

```
#a) List the Publishers in the columns and the different sales in the rows

vgs_m4a = vgs.pivot_table(columns = ["Publisher"], values = ["EU_Sales", "Global_Sales", "J
vgs_m4a
```

Out[153]:

Publisher	10TACLE Studios	1C Company	3DO	505 Games	5pb	AQ Interactive	ASCII Entertainment	Accl Entertainm
EU_Sales	0.01	0.04	0.87	9.31	0.00	0.01	0.07	
Global_Sales	0.02	0.06	2.51	27.38	0.04	0.22	0.50	2
JP_Sales	0.00	0.00	0.00	0.66	0.04	0.00	0.29	
NA_Sales	0.01	0.01	1.43	14.48	0.00	0.19	0.11	1
Other_Sales	0.00	0.01	0.25	2.88	0.00	0.02	0.03	

5 rows × 237 columns

In [154]:

```
#b) Count the number of game titles per publisher

vgs_m4b = vgs[["Publisher", "Name"]].groupby("Publisher").count()
vgs_m4b.columns = ["Count"]
vgs_m4b.sort_values(by = "Count", ascending = False, inplace = True)
vgs_m4b
```

Out[154]:

Publisher	Count
Electronic Arts	819
Activision	423
Ubisoft	397
Nintendo	304
Sony	269
THQ	250
Take-Two Interactive	228
Sega	225
Konami Digital Entertainment	194
Namco Bandai Games	184
Capcom	164
Atari	135
Microsoft	120
Square Enix	115
Warner Bros	109
Tecmo Koei	108
Vivendi Games	94
Eidos Interactive	94
Midway Games	88
Codemasters	83
Disney Interactive Studios	64
505 Games	59
Deep Silver	58
Acclaim Entertainment	55
LucasArts	55
Nippon Ichi Software	46
Bethesda Softworks	41
Rising Star Games	38
Focus Home Interactive	26
D3Publisher	24

Publisher	Count
...	...
Popcorn Arcade	1
Pinnacle	1
Phantagram	1
Pack In Soft	1
Pacific Century Cyber Works	1
PM Studios	1
Ocean	1
Number None	1
Nihon Falcom Corporation	1
NewKidCo	1
Navarre Corp	1
NDA Productions	1
Myelin Media	1
Mercury Games	1
Media Rings	1
Maxis	1
Max Five	1
Little Orbit	1
Kool Kizz	1
Irem Software Engineering	1
Introversion Software	1
Imagineer	1
Illusion Softworks	1
Human Entertainment	1
Hudson Entertainment	1
Home Entertainment Suppliers	1
Hello Games	1
Havas Interactive	1
Hasbro Interactive	1
id Software	1

237 rows × 1 columns

In [155]:

```
#c) Display Platform in columns and sales in rows

vgs_m4c = vgs.pivot_table(columns = ["Platform"], values = ["EU_Sales", "JP_Sales", "NA_Sal
vgs_m4c
```

Out[155]:

Platform	3DS	DC	DS	GBA	GC	N64	PC	PS	PS2	PS3	PS
EU_Sales	33.31	0.29	93.27	30.89	28.32	23.81	93.09	63.12	245.22	266.24	101.1
Global_Sales	120.40	4.11	362.07	125.00	145.77	118.51	169.41	202.65	906.56	748.95	220.4
JP_Sales	33.57	2.22	82.08	15.74	15.90	22.23	0.17	37.92	71.76	51.02	8.4
NA_Sales	46.04	1.54	158.33	74.85	97.52	70.08	59.85	90.59	452.59	321.27	76.5
Other_Sales	7.37	0.06	28.33	3.54	4.10	2.36	15.83	11.11	137.00	110.72	34.2

Plots

In [156]:

```
# distribution plots
sns.distplot(eu_metro_areas["GDP"])
plt.figure()
sns.distplot(eu_metro_areas["GDP"],bins=7,rug=True)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\matplotlib\axes\_axes.py:6462: Us
erWarning: The 'normed' kwarg is deprecated, and has been replaced by the 'd
ensity' kwarg.
  warnings.warn("The 'normed' kwarg is deprecated, and has been "
```

Out[156]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1c4459e4898>
```

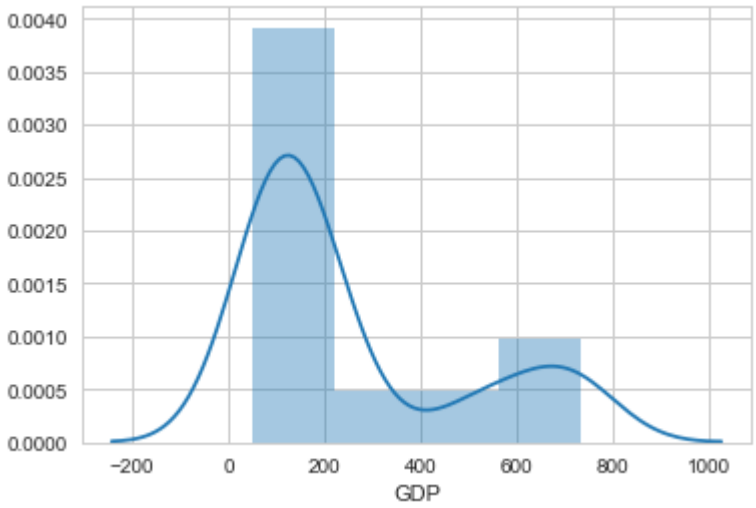
Out[156]:

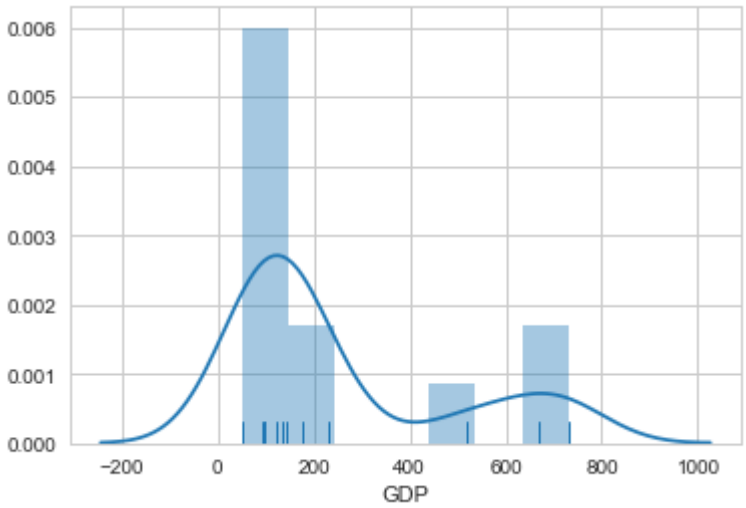
```
<Figure size 432x288 with 0 Axes>
```

```
C:\ProgramData\Anaconda3\lib\site-packages\matplotlib\axes\_axes.py:6462: Us
erWarning: The 'normed' kwarg is deprecated, and has been replaced by the 'd
ensity' kwarg.
  warnings.warn("The 'normed' kwarg is deprecated, and has been "
```

Out[156]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1c445f05358>
```





In [157]:

```
# Joint Plots
sns.jointplot(x="GDP",y="Population",data=eu_metro_areas, kind="reg")
sns.jointplot(x="GDP per Capita",y="Population",data=eu_metro_areas, kind="reg")
```

```
C:\ProgramData\Anaconda3\lib\site-packages\matplotlib\axes\_axes.py:6462: Us
erWarning: The 'normed' kwarg is deprecated, and has been replaced by the 'd
ensity' kwarg.
  warnings.warn("The 'normed' kwarg is deprecated, and has been "
C:\ProgramData\Anaconda3\lib\site-packages\matplotlib\axes\_axes.py:6462: Us
erWarning: The 'normed' kwarg is deprecated, and has been replaced by the 'd
ensity' kwarg.
  warnings.warn("The 'normed' kwarg is deprecated, and has been "
```

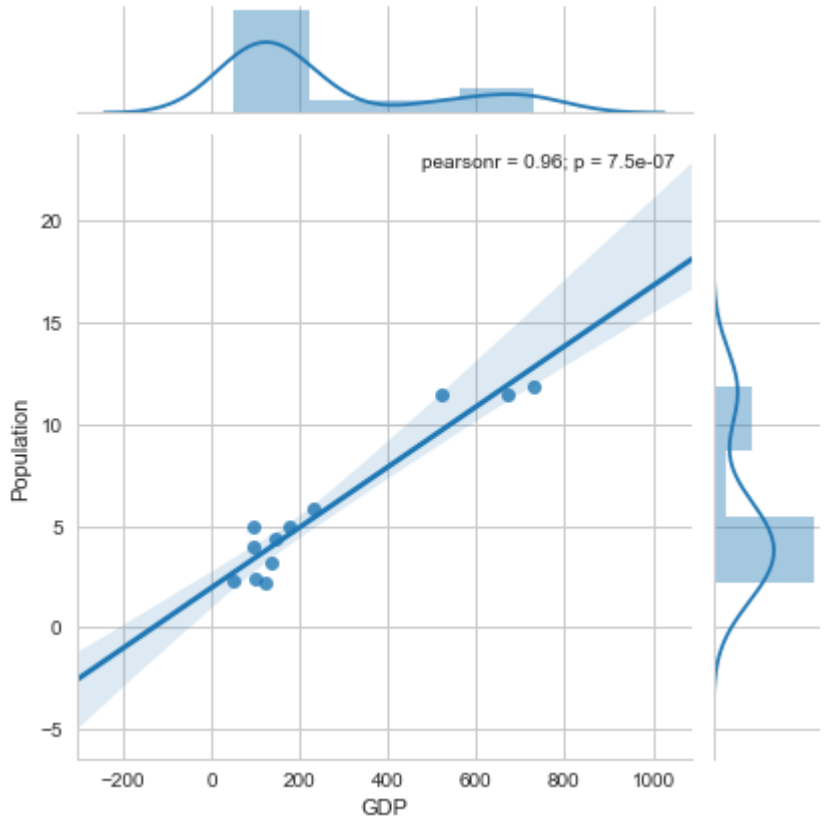
Out[157]:

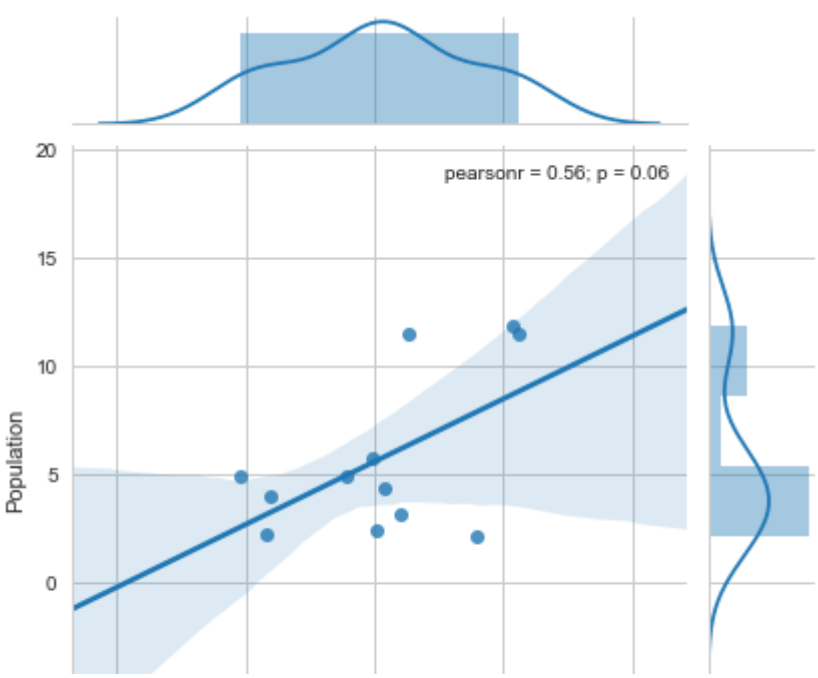
```
<seaborn.axisgrid.JointGrid at 0x1c4464d0400>
```

```
C:\ProgramData\Anaconda3\lib\site-packages\matplotlib\axes\_axes.py:6462: Us
erWarning: The 'normed' kwarg is deprecated, and has been replaced by the 'd
ensity' kwarg.
  warnings.warn("The 'normed' kwarg is deprecated, and has been "
C:\ProgramData\Anaconda3\lib\site-packages\matplotlib\axes\_axes.py:6462: Us
erWarning: The 'normed' kwarg is deprecated, and has been replaced by the 'd
ensity' kwarg.
  warnings.warn("The 'normed' kwarg is deprecated, and has been "
```

Out[157]:

```
<seaborn.axisgrid.JointGrid at 0x1c4464f8d30>
```




pearsonr = 0.56; p = 0.06
Population
20
15
10
5
0

In [158]:

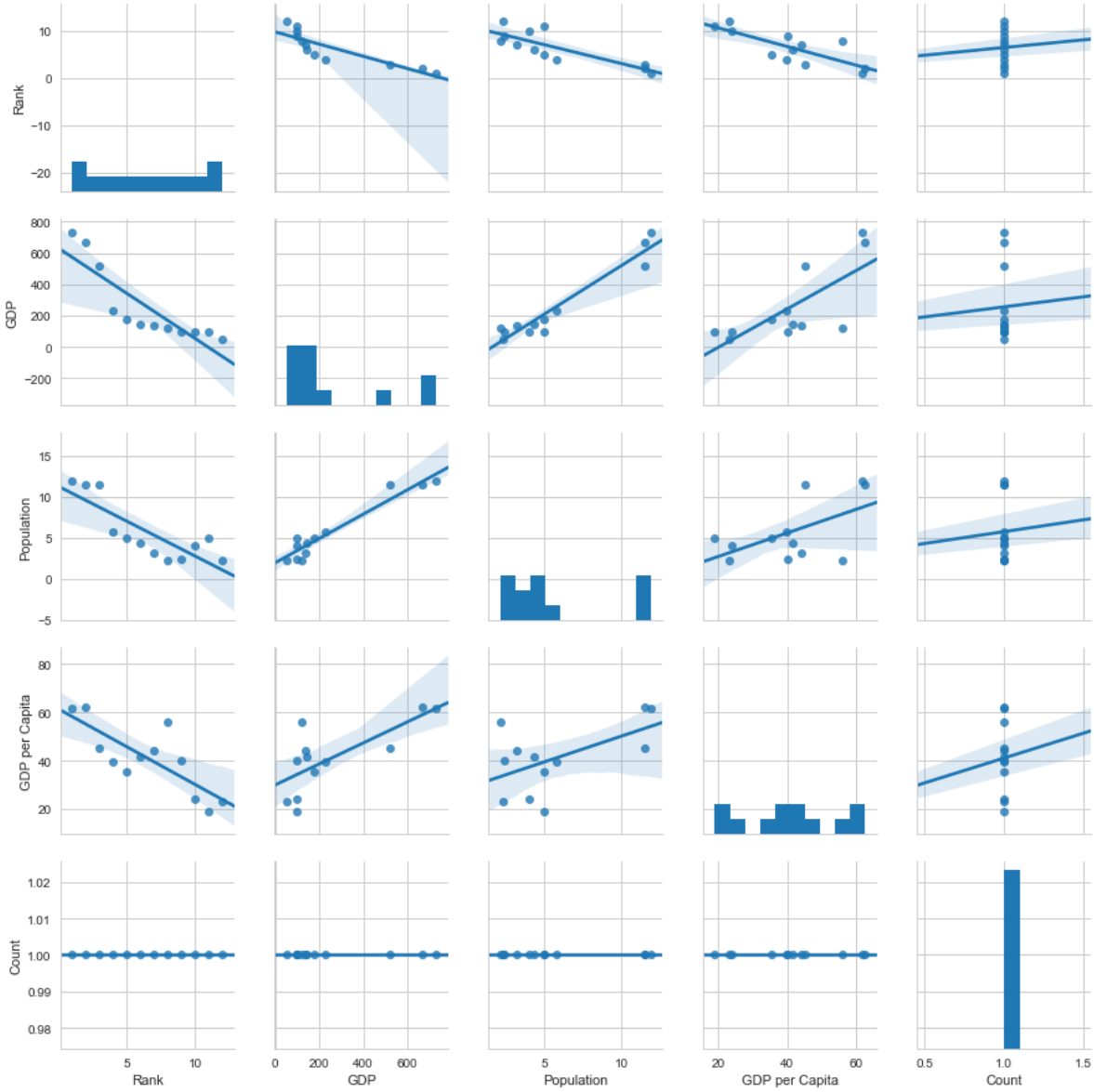
```
# Pair plots

sns.pairplot(eu_metro_areas, kind="reg")
sns.pairplot(eu_metro_areas, kind="reg",diag_kind="kde") # kernel density estimation
```

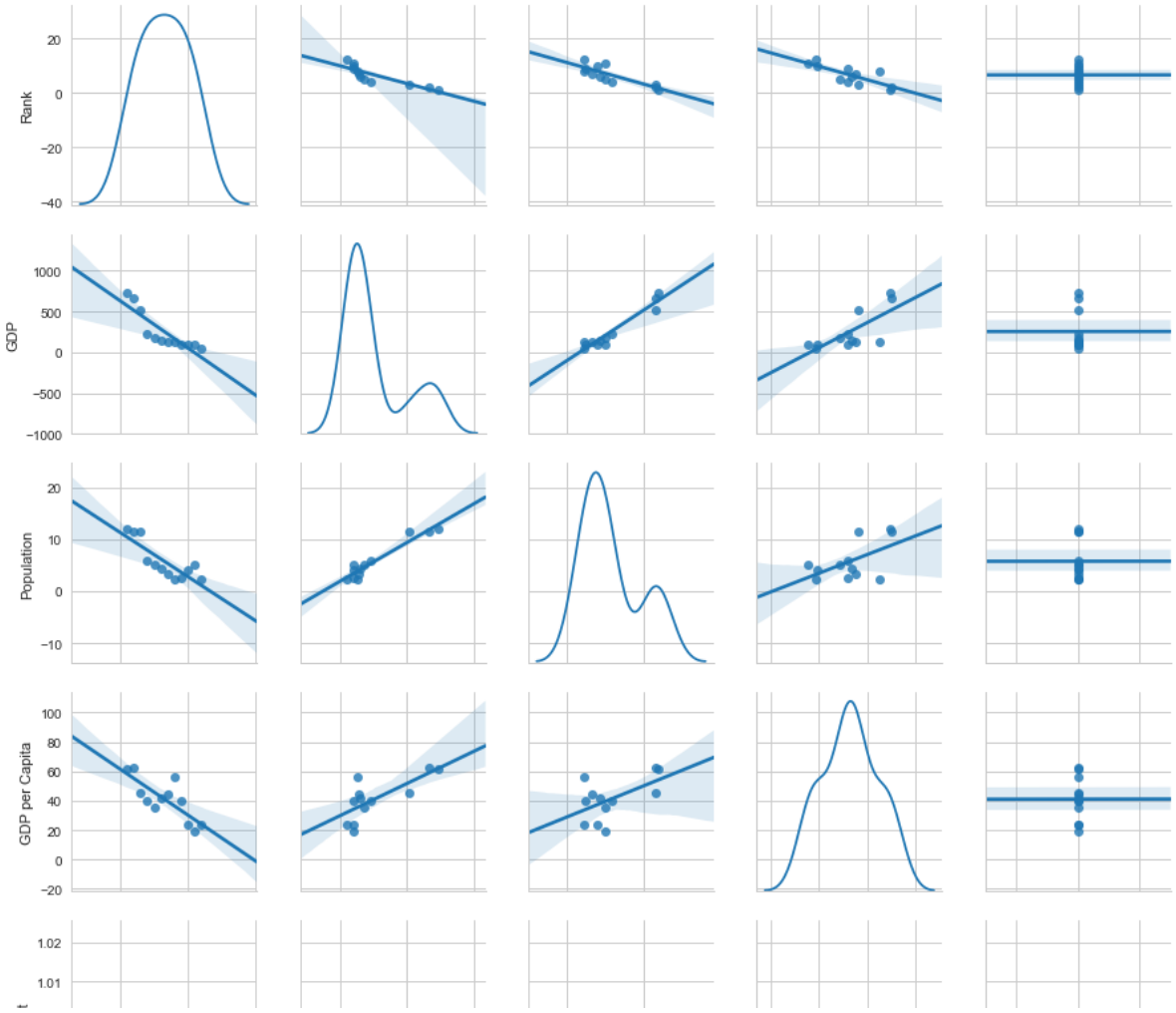
Out[158]:

```
<seaborn.axisgrid.PairGrid at 0x1c442f3f2e8>

C:\ProgramData\Anaconda3\lib\site-packages\statsmodels\nonparametric\kde.py:
488: RuntimeWarning: invalid value encountered in true_divide
  binned = fast_linbin(X, a, b, gridsize) / (delta * nobs)
C:\ProgramData\Anaconda3\lib\site-packages\statsmodels\nonparametric\kdetool
s.py:34: RuntimeWarning: invalid value encountered in double_scalars
  FAC1 = 2*(np.pi*bw/RANGE)**2
```

Out[158]:

```
<seaborn.axisgrid.PairGrid at 0x1c442f2ce48>
```




Rank
GDP
Population
GDP per Capita
t
20
0
−20
−40
1000
500
0
−500
−1000
20
10
0
−10
100
80
60
40
20
0
−20
1.02
1.01

In [159]:

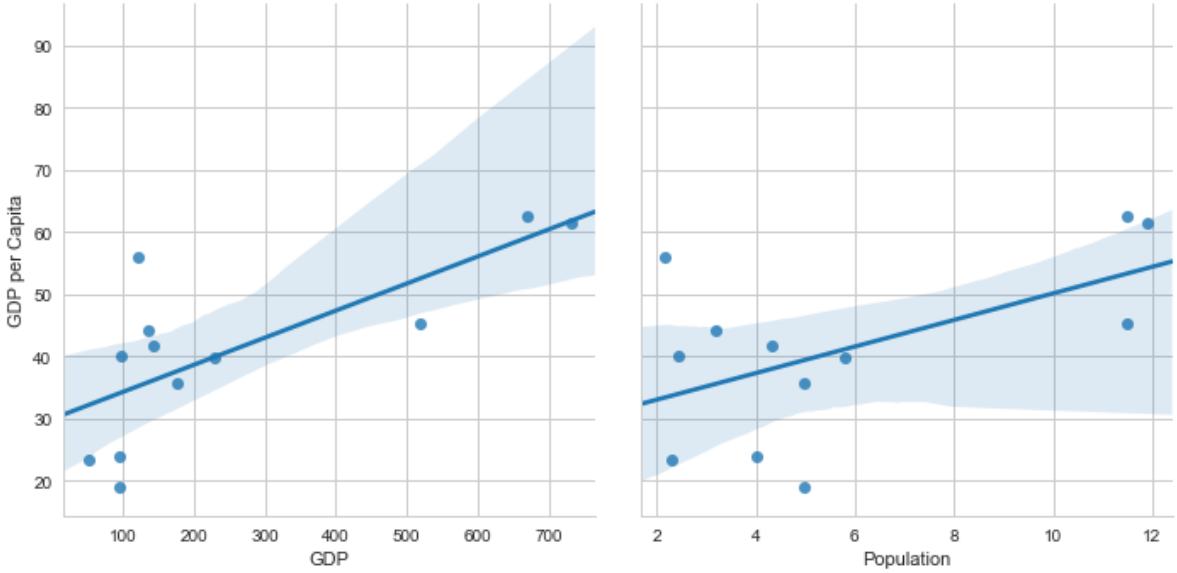
```
# Pair plots only with some variables

plt.figure(figsize=(10,10))

sns.pairplot(eu_metro_areas, x_vars=["GDP","Population"], y_vars=["GDP per Capita"], kind="
```

Out[159]:

```
<Figure size 720x720 with 0 Axes>
```

Out[159]:

```
<seaborn.axisgrid.PairGrid at 0x1c448bd6be0>

<Figure size 720x720 with 0 Axes>
```



In [160]:

```
plt.figure(figsize=(10,10))
flights = sns.load_dataset("flights")
flights.head()
flights = flights.pivot("month", "year", "passengers")
flights.head()
sns.heatmap(flights)
```

Out[160]:

```
<Figure size 720x720 with 0 Axes>
```

Out[160]:

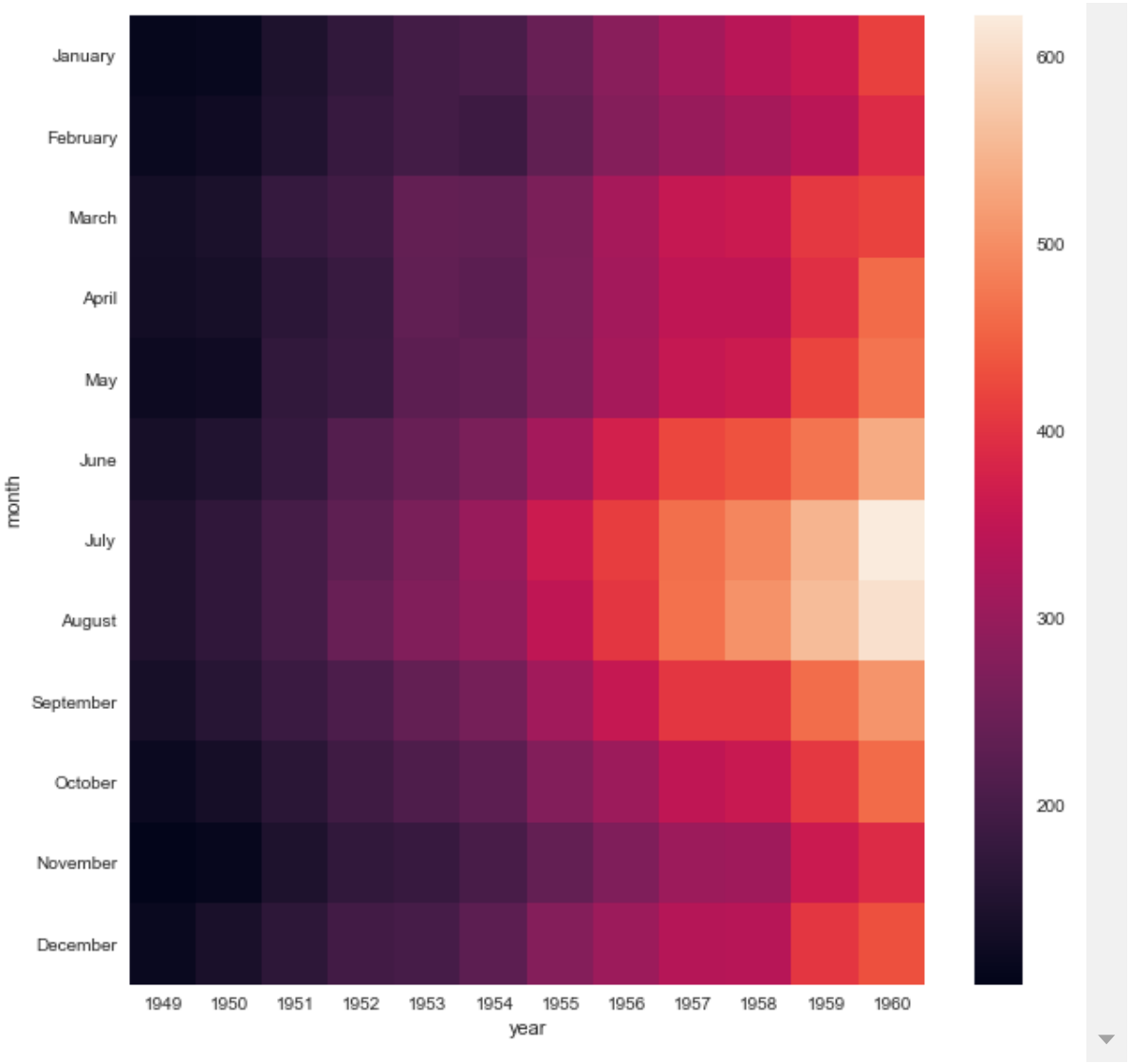
	year	month	passengers
0	1949	January	112
1	1949	February	118
2	1949	March	132
3	1949	April	129
4	1949	May	121

Out[160]:

year	1949	1950	1951	1952	1953	1954	1955	1956	1957	1958	1959	1960
month												
January	112	115	145	171	196	204	242	284	315	340	360	417
February	118	126	150	180	196	188	233	277	301	318	342	391
March	132	141	178	193	236	235	267	317	356	362	406	419
April	129	135	163	181	235	227	269	313	348	348	396	461
May	121	125	172	183	229	234	270	318	355	363	420	472

Out[160]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1c448f76160>
```


January
February
March
April
May
June
July
August
September
October
November
December
month
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
year
600
500
400
300
200

In [161]:

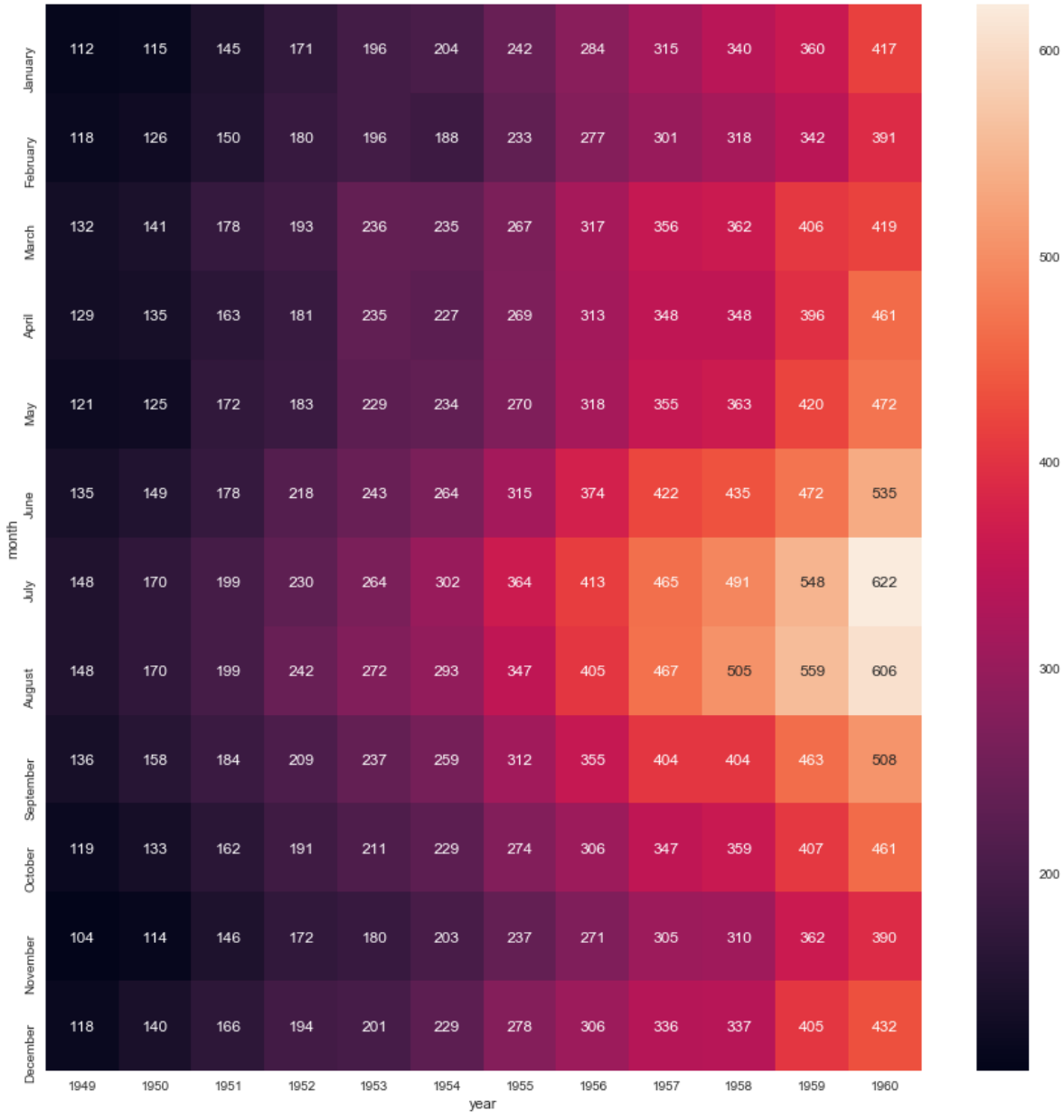
```
plt.figure(figsize=(15,15))
sns.heatmap(flights, annot=True, fmt="d")
```

Out[161]:

```
<Figure size 1080x1080 with 0 Axes>
```

Out[161]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1c4486fdc18>
```



In [162]:

```
cmap=sns.choose_colorbrewer_palette("s") # sequential, diverging or qualitative
```

A Jupyter widget could not be displayed because the widget state could not be found. This could happen if the kernel storing the widget is no longer available, or if the widget state was not saved in the notebook. You may be able to create the widget by running the appropriate cells.

In [163]:

```
plt.figure(figsize=(12,12))
sns.heatmap(flights, annot=True, fmt="d", cmap=cmap)
```

...

In [164]:

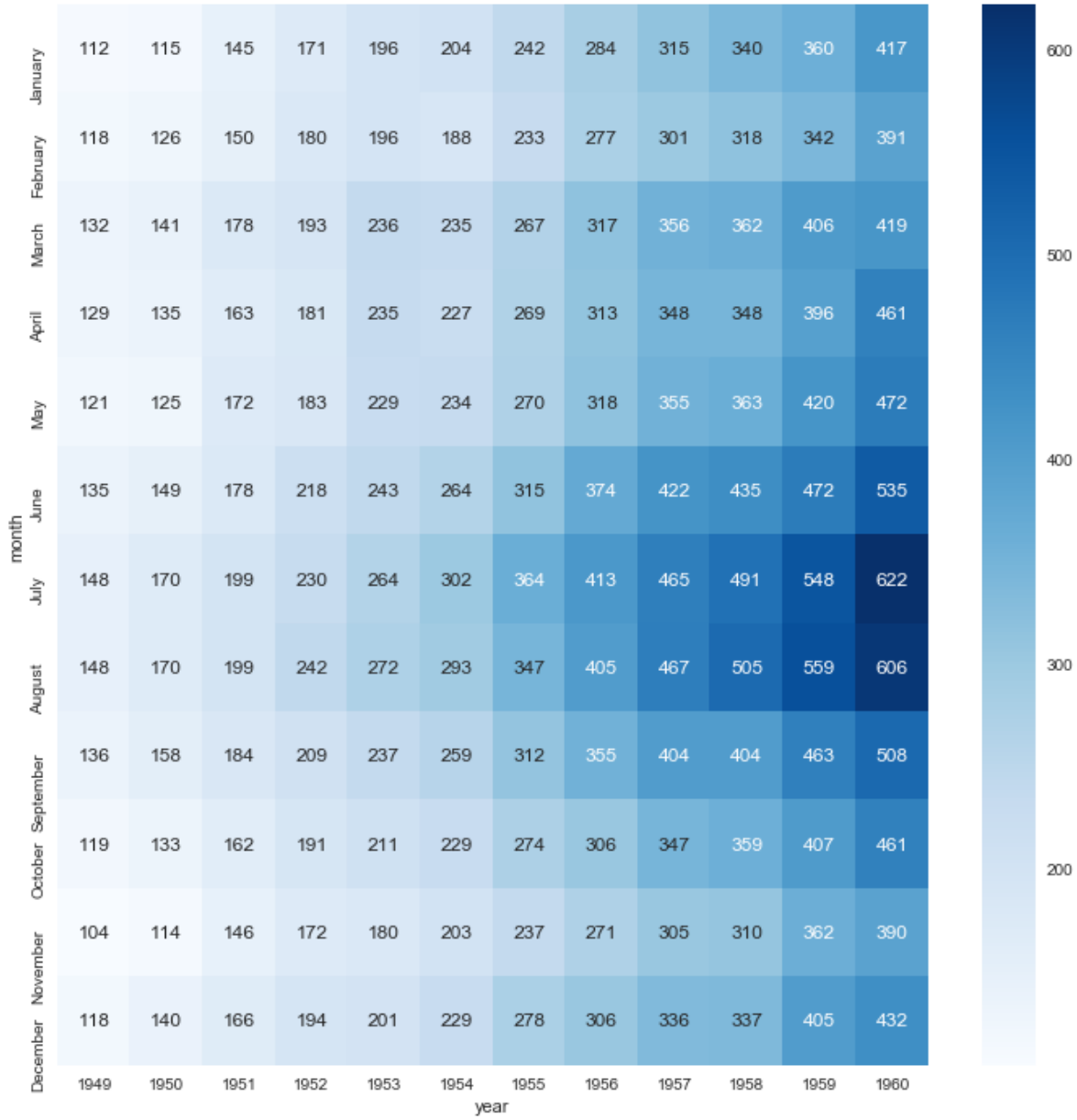
```
#seq_col_brew = sns.color_palette("Blues", 12)
#sns.set_palette(seq_col_brew)

plt.figure(figsize=(12,12))
sns.heatmap(flights, annot=True, fmt="d", cmap="Blues")
```

Out[164]:

```
<Figure size 864x864 with 0 Axes>
```

Out[164]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1c4499d2198>
```



Mission 5

a) List sales (all) by year and by genre
b) For the last 5 years, sales by platform and sales by publisher
c) We want to explore a relationship between sales (all) and critic.

In [165]:

```
# a) List sales (all) by year and by genre
# All sales were included, global and by region in separate columns
# The table is more clear when displaying the data by genre first and year of release secon

vgs_m5a = vgs[["Year_of_Release", "Genre", "EU_Sales", "JP_Sales", "NA_Sales", "Other_Sales
vgs_m5a = vgs_m5a.groupby(["Genre", "Year_of_Release"]).sum()
vgs_m5a
```

Out[165]:

		EU_Sales	JP_Sales	NA_Sales	Other_Sales	Global_Sales
Genre	**Year_of_Release**					
Action	**1996.0**	3.20	1.24	5.30	0.97	10.71
	1997.0	1.04	0.52	1.76	0.20	3.52
	1998.0	7.26	4.25	10.29	1.07	22.87
	1999.0	4.09	1.04	5.63	0.55	11.33
	2000.0	7.35	3.04	11.82	1.06	23.26
	2001.0	15.77	5.29	23.13	3.89	48.10
	2002.0	22.25	3.64	37.42	6.18	69.50
	2003.0	15.34	3.54	25.19	4.22	48.26
	2004.0	11.37	4.46	28.59	15.39	59.89
	2005.0	17.57	3.69	35.67	6.43	63.39
	2006.0	12.28	2.96	27.47	5.82	48.49
	2007.0	19.23	3.61	37.85	11.20	71.94
	2008.0	34.63	3.82	58.15	15.41	111.99
	2009.0	25.60	5.34	45.87	10.50	87.29
	2010.0	24.24	3.60	42.12	8.87	78.92
	2011.0	33.51	5.52	42.15	11.03	92.26
	2012.0	34.84	5.90	39.07	11.63	91.35
	2013.0	37.32	6.71	46.13	12.87	102.97
	2014.0	25.98	2.67	24.87	7.75	61.29
	2015.0	13.63	2.40	15.28	4.57	36.02
	2016.0	5.25	0.86	4.49	1.44	12.14
Adventure	**1997.0**	0.03	0.00	0.12	0.00	0.15
	1999.0	0.25	0.38	0.52	0.04	1.20
	2000.0	0.22	0.10	0.32	0.05	0.69
	2001.0	0.48	0.03	0.76	0.14	1.40
	2002.0	2.07	1.54	3.66	0.55	7.77
	2003.0	0.35	0.07	0.58	0.09	1.11
	2004.0	1.25	0.48	2.56	0.23	4.54
	2005.0	0.59	0.51	1.70	0.19	2.96

Genre	Year_of_Release	EU_Sales	JP_Sales	NA_Sales	Other_Sales	Global_Sales
	2006.0	1.06	0.96	2.79	0.59	5.45
...	**...**	...	...	...	...	...
Sports	**2006.0**	35.03	6.38	65.71	12.37	119.53
	2007.0	19.43	5.76	34.20	11.89	71.38
	2008.0	14.32	1.32	28.84	7.55	52.08
	2009.0	31.48	6.49	46.49	8.78	93.20
	2010.0	21.25	1.81	27.01	7.21	57.28
	2011.0	5.55	0.73	16.33	2.32	24.94
	2012.0	5.08	0.93	14.41	2.22	22.65
	2013.0	12.80	0.24	16.30	4.52	33.86
	2014.0	10.79	0.42	11.62	3.52	26.30
	2015.0	11.41	0.36	13.52	4.08	29.39
	2016.0	9.06	0.23	5.71	2.35	17.34
Strategy	**1998.0**	0.09	0.00	0.04	0.01	0.14
	1999.0	0.10	0.00	0.04	0.01	0.14
	2000.0	0.82	1.70	1.61	0.38	4.50
	2001.0	0.61	0.76	1.41	0.09	2.89
	2002.0	0.33	0.35	1.46	0.07	2.24
	2003.0	1.88	0.54	2.45	0.27	5.15
	2004.0	0.91	0.86	1.49	0.19	3.47
	2005.0	0.73	0.19	1.82	0.24	2.98
	2006.0	0.30	0.00	1.11	0.18	1.64
	2007.0	1.26	0.39	2.54	0.56	4.74
	2008.0	1.62	0.03	2.03	0.32	3.98
	2009.0	3.51	0.07	3.05	0.70	7.36
	2010.0	3.27	0.03	4.50	1.05	8.88
	2011.0	2.39	0.05	2.89	0.74	6.08
	2012.0	0.82	0.00	0.87	0.23	1.92
	2013.0	1.99	0.29	1.69	0.42	4.38
	2014.0	0.00	0.00	0.03	0.01	0.04
	2015.0	0.74	0.02	0.30	0.09	1.13
	2016.0	0.23	0.00	0.11	0.03	0.35

243 rows × 5 columns

In [166]:

```
# b) For the last 5 years, sales by platform and sales by publisher
# Most recent year is considered to be 2016, since there is very little data for 2017 and 2

vgs_m5b = vgs[["Year_of_Release", "Platform", "Publisher", "EU_Sales", "JP_Sales", "NA_Sale
vgs_m5b = vgs_m5b[vgs_m5b["Year_of_Release"] > 2011].groupby(["Platform", "Publisher", "Yea
vgs_m5b
```

Out[166]:

			EU_Sales	JP_Sales	NA_Sales	Other_Sales	Global_S
Platform	**Publisher**	**Year_of_Release**					
3DS	**Activision**	**2012.0**	0.20	0.00	0.38	0.05	
		2013.0	0.12	0.00	0.10	0.02	
	Agatsuma Entertainment	**2013.0**	0.00	0.03	0.00	0.00	
	Aksys Games	**2016.0**	0.00	0.01	0.05	0.01	
	Atlus	**2013.0**	0.01	0.41	0.38	0.04	
		2015.0	0.01	0.05	0.16	0.02	
	Capcom	**2012.0**	0.22	0.30	0.30	0.05	
		2014.0	0.00	0.02	0.00	0.00	
		2016.0	0.00	0.05	0.14	0.02	
	Electronic Arts	**2012.0**	0.20	0.00	0.06	0.03	
	FuRyu	**2015.0**	0.01	0.10	0.12	0.01	
	Konami Digital Entertainment	**2012.0**	0.11	0.10	0.15	0.02	
	Marvelous Entertainment	**2012.0**	0.17	0.29	0.41	0.05	
		2014.0	0.00	0.06	0.00	0.00	
	Namco Bandai Games	**2012.0**	0.04	0.15	0.18	0.02	
		2013.0	0.04	0.28	0.05	0.01	
		2014.0	0.00	0.00	0.01	0.00	
		2015.0	0.03	0.16	0.12	0.02	
	Nintendo	**2012.0**	7.67	9.87	9.66	1.57	
		2013.0	5.71	4.80	4.93	0.92	
		2014.0	1.76	4.52	2.17	0.37	
		2015.0	2.28	2.60	2.78	0.48	
		2016.0	0.17	0.61	0.32	0.05	
	Nippon Ichi Software	**2014.0**	0.07	0.27	0.33	0.04	
		2015.0	0.01	0.13	0.12	0.01	
	Rising Star Games	**2012.0**	0.00	0.00	0.03	0.00	
		2013.0	0.13	0.04	0.24	0.03	
		2015.0	0.00	0.03	0.01	0.00	

Platform	Publisher	Year_of_Release	EU_Sales	JP_Sales	NA_Sales	Other_Sales	Global_S
	Screenlife	2013.0	0.00	0.42	0.00	0.00	
	Sega	2013.0	0.14	0.07	0.24	0.03	
...	...	...	...	...	...	...	
XOne	Konami Digital Entertainment	2014.0	0.11	0.00	0.15	0.02	
		2015.0	0.32	0.01	0.41	0.07	
		2016.0	0.03	0.00	0.01	0.00	
	Majesco Entertainment	2013.0	0.05	0.00	0.17	0.02	
	Microsoft	2013.0	1.57	0.02	3.10	0.47	
		2014.0	2.48	0.06	3.58	0.57	
		2015.0	2.45	0.06	5.90	0.86	
		2016.0	0.87	0.01	1.27	0.21	
	Milestone S.r.l	2016.0	0.01	0.00	0.00	0.00	
	Namco Bandai Games	2015.0	0.59	0.00	0.85	0.14	
		2016.0	0.15	0.00	0.32	0.04	
	Revolution Software	2015.0	0.01	0.00	0.00	0.00	
	Sega	2014.0	0.24	0.00	0.22	0.04	
	Slightly Mad Studios	2015.0	0.13	0.00	0.09	0.02	
	Square Enix	2014.0	0.17	0.00	0.18	0.03	
		2015.0	0.65	0.02	0.54	0.10	
	Stainless Games	2016.0	0.01	0.00	0.01	0.00	
	Take-Two Interactive	2013.0	0.11	0.00	0.70	0.09	
		2014.0	2.48	0.00	4.38	0.68	
		2015.0	0.70	0.00	2.87	0.39	
		2016.0	0.41	0.00	1.12	0.16	
	Tecmo Koei	2015.0	0.02	0.00	0.03	0.00	
	Telltale Games	2016.0	0.02	0.00	0.02	0.00	
	Ubisoft	2013.0	0.18	0.00	0.41	0.06	
		2014.0	2.96	0.01	4.88	0.77	
		2015.0	1.07	0.00	1.60	0.26	
		2016.0	0.96	0.00	1.70	0.27	
	Warner Bros	2014.0	0.92	0.01	1.42	0.23	
		2015.0	1.33	0.01	3.29	0.48	
		2016.0	0.22	0.00	0.32	0.05	

439 rows × 5 columns

In [167]:

```
#c) We want to explore a relationship between sales (all) and critic & user scores.

# Select the data that is of interest

vgs_m5c = vgs[["Global_Sales", "Critic_Score", "User_Score"]]
vgs_m5c.dropna(inplace = True)

# Understand the data

sns.jointplot(x = "Critic_Score", y = "Global_Sales", data = vgs_m5c, kind = "reg")
sns.jointplot(x = "User_Score", y = "Global_Sales", data = vgs_m5c, kind = "reg")

print("Global sales has an exponential relationship with both Critic Scores and User scores
print("Both relationships are positive but weak, since the correlation between Global Sales
print("The relationship between Global Sales and Critic Score is stronger, with a correlati
```

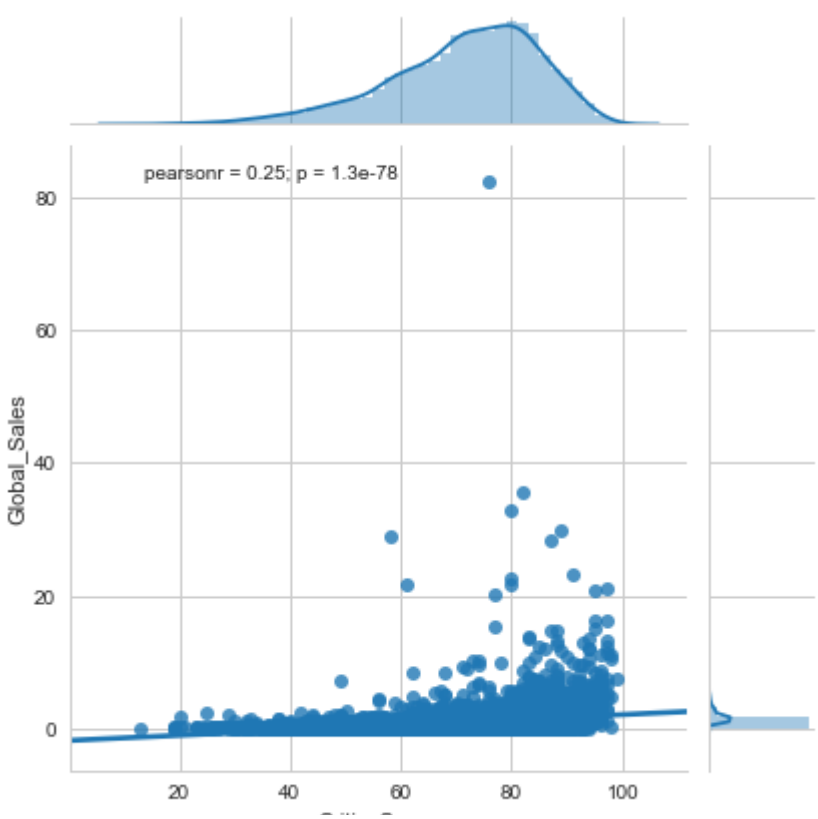
```
C:\ProgramData\Anaconda3\lib\site-packages\ipykernel_launcher.py:6: SettingW
ithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/s
table/indexing.html#indexing-view-versus-copy (http://pandas.pydata.org/pand
as-docs/stable/indexing.html#indexing-view-versus-copy)

C:\ProgramData\Anaconda3\lib\site-packages\matplotlib\axes\_axes.py:6462: Us
erWarning: The 'normed' kwarg is deprecated, and has been replaced by the 'd
ensity' kwarg.
  warnings.warn("The 'normed' kwarg is deprecated, and has been "
C:\ProgramData\Anaconda3\lib\site-packages\matplotlib\axes\_axes.py:6462: Us
erWarning: The 'normed' kwarg is deprecated, and has been replaced by the 'd
ensity' kwarg.
  warnings.warn("The 'normed' kwarg is deprecated, and has been "
```

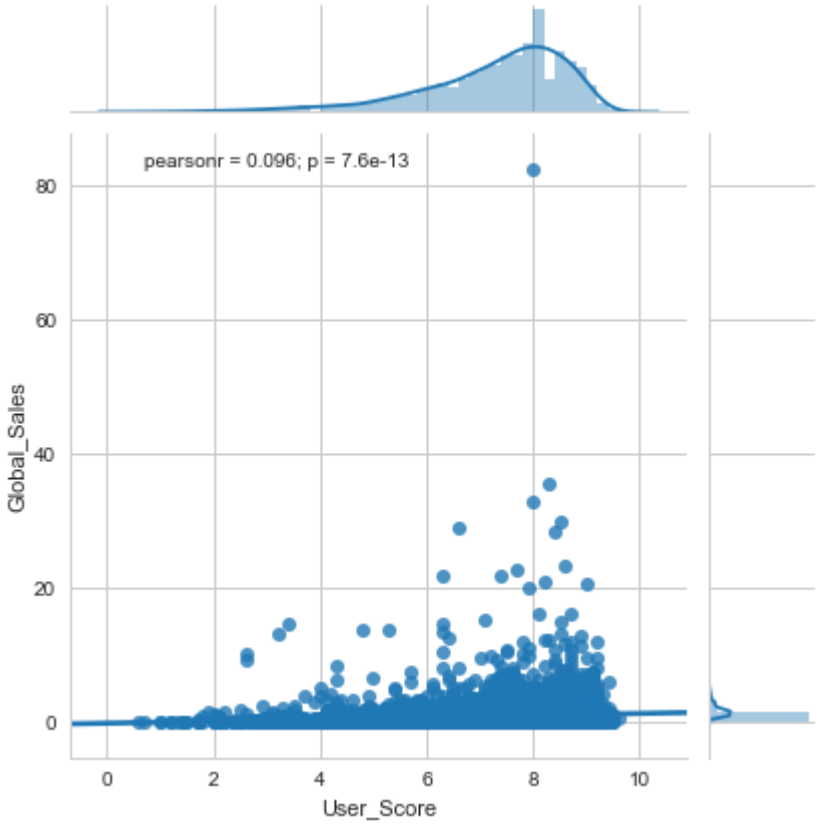
Out[167]:

```
<seaborn.axisgrid.JointGrid at 0x1c4483fc668>

C:\ProgramData\Anaconda3\lib\site-packages\matplotlib\axes\_axes.py:6462: Us
erWarning: The 'normed' kwarg is deprecated, and has been replaced by the 'd
ensity' kwarg.
  warnings.warn("The 'normed' kwarg is deprecated, and has been "
C:\ProgramData\Anaconda3\lib\site-packages\matplotlib\axes\_axes.py:6462: Us
erWarning: The 'normed' kwarg is deprecated, and has been replaced by the 'd
ensity' kwarg.
  warnings.warn("The 'normed' kwarg is deprecated, and has been "
```

Out[167]:

```
<seaborn.axisgrid.JointGrid at 0x1c445d78d30>

Global sales has an exponential relationship with both Critic Scores and Use
r scores.
Both relationships are positive but weak, since the correlation between Glob
al Sales and the two variables are under 0.30.
The relationship between Global Sales and Critic Score is stronger, with a c
orrelation of 0.25 compared to 0.096 for User Scores.
```


pearsonr = 0.25; p = 1.3e-78
Global_Sales
0
20
40
60
80
20
40
60
80
100


pearsonr = 0.096; p = 7.6e-13
Global_Sales
0
20
40
60
80
0
2
4
6
8
10
User_Score

Join & Merge

Visualizing Joins

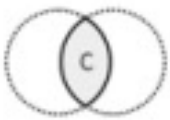
Dataset 1

	name	size
0	A	1
1	B	2
2	B	3
3	C	4

Dataset 2

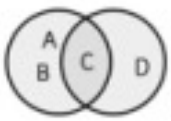
	name	value
3	C	10
1	C	9
2	D	8
4	D	7

Inner



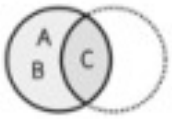
	name	size	value
0	C	4	10
1	C	4	9

Outer



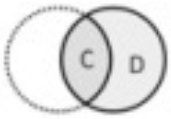
	name	size	value
0	A	1.00	nan
1	B	2.00	nan
2	B	3.00	nan
3	C	4.00	10.00
4	C	4.00	9.00
5	D	nan	8.00
6	D	nan	7.00

Left



	name	size	value
0	A	1	nan
1	B	2	nan
2	B	3	nan
3	C	4	10.00
4	C	4	9.00

Right



	name	size	value
0	C	4.00	10
1	C	4.00	9
2	D	nan	8
3	D	nan	7

In [168]:

```
# joins are the basic tool for querying databases
# relational databases are in fact based in joins
# with Data Frames we can do exactly the same
# let's first have two hypothetical tables

df1 = pd.DataFrame({'name':['John', 'George','Ringo'],'color': ['Blue','Blue','Purple']})
df2 = pd.DataFrame({'name':['Paul','George','Ringo'],'carcolor':['Red','Blue',np.nan]},inde
df1
df2
```

Out[168]:

	name	color
0	John	Blue
1	George	Blue
2	Ringo	Purple

Out[168]:

	name	carcolor
3	Paul	Red
1	George	Blue
2	Ringo	NaN

In [169]:

```
# our first operation is not a join,
# but a concatenation of the two tables
# by defalt concat preserves index values
pd.concat([df1,df2])

# but we can ignore them if we want
pd.concat([df1,df2], ignore_index=True)

# we can also concat by rows instead of columns
pd.concat([df1,df2],axis=1)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\ipykernel_launcher.py:4: FutureWa
rning: Sorting because non-concatenation axis is not aligned. A future versi
on
of pandas will change to not sort by default.

To accept the future behavior, pass 'sort=True'.

To retain the current behavior and silence the warning, pass sort=False

  after removing the cwd from sys.path.
```

Out[169]:

	carcolor	color	name
0	NaN	Blue	John
1	NaN	Blue	George
2	NaN	Purple	Ringo
3	Red	NaN	Paul
1	Blue	NaN	George
2	NaN	NaN	Ringo

```
C:\ProgramData\Anaconda3\lib\site-packages\ipykernel_launcher.py:7: FutureWa
rning: Sorting because non-concatenation axis is not aligned. A future versi
on
of pandas will change to not sort by default.

To accept the future behavior, pass 'sort=True'.

To retain the current behavior and silence the warning, pass sort=False

  import sys
```

Out[169]:

	carcolor	color	name
0	NaN	Blue	John
1	NaN	Blue	George
2	NaN	Purple	Ringo
3	Red	NaN	Paul
4	Blue	NaN	George

	carcolor	color	name
5	NaN	NaN	Ringo

Out[169]:

	name	color	name	carcolor
0	John	Blue	NaN	NaN
1	George	Blue	George	Blue
2	Ringo	Purple	Ringo	NaN
3	NaN	NaN	Paul	Red

In [170]:

```
d1=pd.DataFrame({'name':['A','B','B','C'],'size':[1,2,3,4]})

d2=pd.DataFrame({'name':['C','C','D','D'],'value':[10,9,8,7]},index=[3,1,2,4])

d1
d2

d1.merge(d2, on='name')                    # inner join
d1.merge(d2, how='outer', on='name')       #outer join
d1.merge(d2, how='left',  on='name')       #left join
d1.merge(d2, how='right', on='name')       # right join
```

Out[170]:

	name	size
0	A	1
1	B	2
2	B	3
3	C	4

Out[170]:

	name	value
3	C	10
1	C	9
2	D	8
4	D	7

Out[170]:

	name	size	value
0	C	4	10
1	C	4	9

Out[170]:

	name	size	value
0	A	1.0	NaN
1	B	2.0	NaN
2	B	3.0	NaN
3	C	4.0	10.0
4	C	4.0	9.0
5	D	NaN	8.0
6	D	NaN	7.0

Out[170]:

	name	size	value
0	A	1	NaN
1	B	2	NaN
2	B	3	NaN
3	C	4	10.0
4	C	4	9.0

Out[170]:

	name	size	value
0	C	4.0	10
1	C	4.0	9
2	D	NaN	8
3	D	NaN	7

In [171]:

```
#You can also use SQL with pandasql
#
#!pip install pandasql
import pandasql

pandasql.sqldf("select * from eu_metro_areas;")

pandasql.sqldf("select * from eu_metro_areas where Eurozone='Y';")
```

Out[171]:

	City	Rank	GDP	Population	GDP per Capita	Eurozone	High Population	Count
0	London	1	732	11.90	61.5	N	Yes	1
1	Paris	2	669	11.50	62.4	Y	Yes	1
2	Moscow	3	520	11.50	45.2	N	Yes	1
3	Madrid	4	230	5.80	39.7	Y	No	1
4	Barcelona	5	177	4.97	35.6	Y	No	1
5	Rome	6	144	4.34	41.6	Y	No	1
6	Milan	7	136	3.20	44.2	Y	No	1
7	Vienna	8	122	2.18	56.0	Y	No	1
8	Lisbon	9	98	2.44	40.2	Y	No	1
9	Athens	10	96	4.01	23.9	Y	No	1
10	Berlin	11	95	4.97	19.1	Y	No	1
11	Bucharest	12	51	2.30	23.3	N	No	1

Out[171]:

	City	Rank	GDP	Population	GDP per Capita	Eurozone	High Population	Count
0	Paris	2	669	11.50	62.4	Y	Yes	1
1	Madrid	4	230	5.80	39.7	Y	No	1
2	Barcelona	5	177	4.97	35.6	Y	No	1
3	Rome	6	144	4.34	41.6	Y	No	1
4	Milan	7	136	3.20	44.2	Y	No	1
5	Vienna	8	122	2.18	56.0	Y	No	1
6	Lisbon	9	98	2.44	40.2	Y	No	1
7	Athens	10	96	4.01	23.9	Y	No	1
8	Berlin	11	95	4.97	19.1	Y	No	1

Mission 6

For this and the next mission we will use data from Kaggle In concrete from the World University Rankings Competition https://www.kaggle.com/mylesoneill/world-university-rankings (https://www.kaggle.com/mylesoneill/world-university-rankings)
In this first mission with the rankings we will focus on the Times Ranking

a) Find the top 50 universities of the top 10 countries and see how do they compare. Are their rankings clustered? Are there very few good ones or there is a high dispersion.

In [172]:

```
# Read university data

uni = pd.read_excel("timesData.xlsx")[["university_name", "country", "total_score", "year"]

# Convert total_score to numeric

uni["total_score"] = pd.to_numeric(uni["total_score"], errors="coerce")

# Drop null values

uni.dropna(inplace = True)

# Only select instances of the last year, being 2016

uni = uni[uni["year"] == uni["year"].max()]

# Create country filter

cou = uni[["country", "total_score"]].groupby("country").mean().sort_values(by = "total_sco

# Filter the universities by the country filter

uni_f = uni[uni["country"].isin(cou["country"]) == True]

# Explore score distribution for top 50 universities, regardless of country

import numpy as np

uni_disp = uni["total_score"].plot.hist(bins = np.arange(45,100,5))
uni_max = uni["total_score"].max()

# Start exploring by country. Metrics per country gathered will be average score of all uni
# of all the universities in the top 50, the number of universities in the top 50 and the a
# in the top 50.

uni_tot = uni_f.head(50)[["country", "university_name", "total_score"]].groupby("country").
uni_count = uni_f.head(50)[["country", "university_name"]].groupby("country").count().sort_
uni_avg = uni_f.head(50)[["country", "university_name", "total_score"]].groupby("country").
uni_stats = cou.merge(uni_tot, how = "outer", on = "country").merge(uni_count, how = "outer
uni_stats.columns = ["country", "avg_score", "top_50_agg_score", "top_50_unis", "top_50_avg
uni_stats.sort_values(by = "top_50_agg_score", ascending = False).fillna(0)

print("The US seems to dominates the top 50 rankings, having the highest aggregate score an
print("However, of the universities in the top 50, the UK and Switzerland have a higher ave
print("All other countries have a couple of universities in the top 50, with an average sco
print("Finland is the only country with no universities in the top 50, indicating that thei

print("\n")

print("As we can see from the histogram below, most universities in the top 50 have a score
print(f"Very few universities actually have a score of 70 and above, the max being {uni_max
```

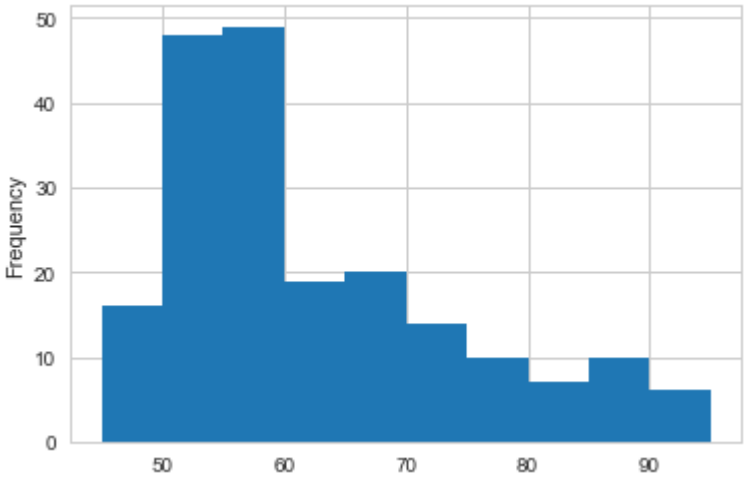
Out[172]:

	country	avg_score	top_50_agg_score	top_50_unis	top_50_avg_score
2	United States of America	67.431746	2205.3	27.0	81.677778

	country	avg_score	top_50_agg_score	top_50_unis	top_50_avg_score
8	United Kingdom	62.020588	671.1	8.0	83.887500
4	Canada	64.657143	231.8	3.0	77.266667
5	Australia	64.237500	213.0	3.0	71.000000
7	Switzerland	63.642857	164.4	2.0	82.200000
0	Singapore	73.700000	147.4	2.0	73.700000
1	China	71.000000	142.0	2.0	71.000000
6	Hong Kong	64.233333	138.2	2.0	69.100000
3	Japan	65.500000	71.1	1.0	71.100000
9	Finland	61.900000	0.0	0.0	0.000000

The US seems to dominates the top 50 rankings, having the highest aggregate score and number of univeristies in the top 50.
However, of the universities in the top 50, the UK and Switzerland have a higher average score
All other countries have a couple of universities in the top 50, with an average score in the low 70s or high 60s
Finland is the only country with no universities in the top 50, indicating that their university scores are balanced

As we can see from the histogram below, most universities in the top 50 have a score around 50 or 60.
Very few universities actually have a score of 70 and above, the max being 95.2.



Mission 7

In this last mission we will compare two rankings: Shangai and Times. The objective is to find out how well they correlate per country and globally. We will use a sample and focus on the best 100 universities.

In [173]:

```
# Load both datasets.

uni_t = pd.read_excel("timesData.xlsx")[["university_name", "country", "total_score", "year
uni_s = pd.read_excel("shanghaiData.xlsx")[["university_name", "total_score", "year"]]

# Convert total score on the times dataset to numeric

uni_t["total_score"] = pd.to_numeric(uni_t["total_score"], errors = "coerce")

# Drop null values from both datasets

uni_t.dropna(inplace = True)
uni_s.dropna(inplace = True)

# First select the data for the same year in both tables.
# Since the latest data on the shangai dataset is from 2015, this will be the year use for

uni_t_2015 = uni_t[uni_t["year"] == 2015]
uni_s_2015 = uni_s[uni_s["year"] == 2015]

# Select columns of interest (university_name, total_score and country in the times dataset

uni_t_sel = uni_t_2015[["university_name", "country", "total_score"]]
uni_s_sel = uni_s_2015[["university_name", "total_score"]]

# Merge both datasets, using the times dataset as the main one, since country data is avail

uni_both = uni_t_sel.merge(uni_s_sel, how = "left", on = "university_name")
uni_both.columns = ["university_name", "country", "total_score_times", "total_score_shangai

# Drop null values.

uni_both.dropna(inplace = True)
uni_both = uni_both[:-2]

uni_both

# Some data has been lost, but we still have 60 entries of data, which is enough to run a c

plot = sns.jointplot(x = "total_score_times", y = "total_score_shangai", data = uni_both, k
r = uni_both[["total_score_times", "total_score_shangai"]].corr().iloc[0][1]
print("\n")
print(f"Correlation between the two datasets is positive and very strong: {r:.3f}")
print("This similarity indicates that while both rankings may have similarities in their ap
print("\n")
```

Out[173]:

	university_name	country	total_score_times	total_score_shangai
0	California Institute of Technology	United States of America	94.3	59.6
1	Harvard University	United States of America	93.3	100.0
2	University of Oxford	United Kingdom	93.2	56.6

	university_name	country	total_score_times	total_score_shangai
3	Stanford University	United States of America	92.9	73.3
4	University of Cambridge	United Kingdom	92.0	68.8
6	Princeton University	United States of America	90.9	61.0
7	University of California, Berkeley	United States of America	89.5	69.6
9	Yale University	United States of America	87.5	54.5
10	University of Chicago	United States of America	87.1	57.1
11	University of California, Los Angeles	United States of America	85.5	50.7
13	Columbia University	United States of America	84.4	58.8
14	Johns Hopkins University	United States of America	83.0	46.3
15	University of Pennsylvania	United States of America	81.0	46.1
17	Duke University	United States of America	79.9	38.0
18	Cornell University	United States of America	79.4	50.5
19	University of Toronto	Canada	79.3	40.6
20	Northwestern University	United States of America	79.2	38.8
21	University College London	United Kingdom	78.7	44.5
23	Carnegie Mellon University	United States of America	74.3	29.2
25	University of Washington	United States of America	73.2	47.8
29	University of Illinois at Urbana-Champaign	United States of America	71.9	38.6
31	University of British Columbia	Canada	71.8	34.1
36	University of California, Santa Barbara	United States of America	70.0	34.6
37	New York University	United States of America	69.9	38.8
38	McGill University	Canada	69.6	28.5
40	University of California, San Diego	United States of America	68.6	48.7
43	Karolinska Institute	Sweden	66.8	31.9
46	University of North Carolina at Chapel Hill	United States of America	65.9	34.4
53	Brown University	United States of America	64.1	27.0
54	KU Leuven	Belgium	63.7	24.7
...	...	...	...	...

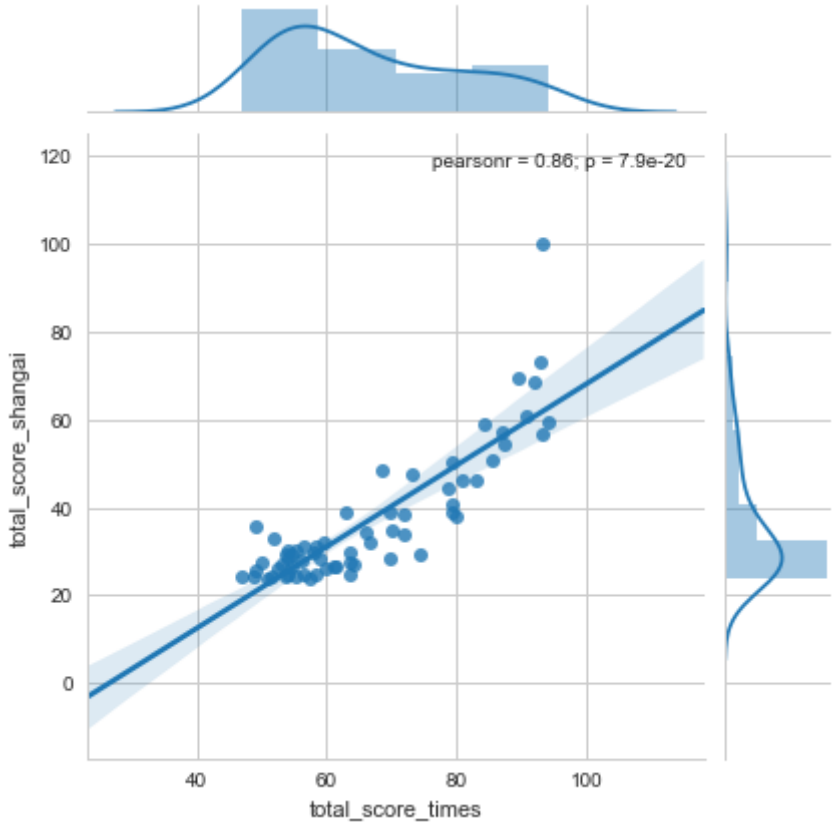
	university_name	country	total_score_times	total_score_shangai
64	The University of Queensland	Australia	61.2	26.7
68	Rice University	United States of America	59.8	26.0
69	Heidelberg University	Germany	59.6	32.2
73	University of Bristol	United Kingdom	58.9	28.3
74	University of Basel	Switzerland	58.4	24.9
75	University of Southern California	United States of America	58.4	31.3
78	Utrecht University	Netherlands	58.0	29.8
81	Michigan State University	United States of America	57.3	24.0
85	University of Arizona	United States of America	56.5	24.7
88	University of California, Irvine	United States of America	56.4	31.0
89	Ghent University	Belgium	56.2	27.8
93	McMaster University	Canada	55.3	24.4
95	Vanderbilt University	United States of America	55.2	30.2
97	Stockholm University	Sweden	54.6	26.7
99	Uppsala University	Sweden	54.6	29.2
103	University of Helsinki	Finland	53.9	28.2
104	University of Warwick	United Kingdom	53.9	24.6
105	University of Zurich	Switzerland	53.9	30.1
107	University of Geneva	Switzerland	53.8	29.4
108	University of California, Santa Cruz	United States of America	53.7	24.5
116	University of Groningen	Netherlands	53.1	27.0
127	University of Florida	United States of America	52.5	26.2
132	University of Maryland, College Park	United States of America	51.9	32.8
136	VU University Amsterdam	Netherlands	51.4	24.2
141	Texas A&M University	United States of America	50.9	23.9
152	Aarhus University	Denmark	49.9	27.3
156	Osaka University	Japan	49.1	25.7
159	University of Copenhagen	Denmark	49.0	35.7
161	University of Utah	United States of America	48.6	24.5
181	Arizona State University	United States of America	46.9	24.5

64 rows × 4 columns

```
C:\ProgramData\Anaconda3\lib\site-packages\matplotlib\axes\_axes.py:6462: Us
erWarning: The 'normed' kwarg is deprecated, and has been replaced by the 'd
```

```
ensity' kwarg.
  warnings.warn("The 'normed' kwarg is deprecated, and has been "
C:\ProgramData\Anaconda3\lib\site-packages\matplotlib\axes\_axes.py:6462: Us
erWarning: The 'normed' kwarg is deprecated, and has been replaced by the 'd
ensity' kwarg.
  warnings.warn("The 'normed' kwarg is deprecated, and has been "
```

```
Correlation between the two datasets is positive and very strong: 0.861
This similarity indicates that while both rankings may have similarities in
their approach
```



In [ ]: