CA1

February 14, 2021

1 Compulsory Assignment 1 - Pandas and visualizations

Nida Grønbekk

1.1 Imports

[1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
from matplotlib import pyplot as plt
from copy import copy
```

1.2 Loading and exploring the dataset

1. Load the dataset named `videogames.csv` and store it in a dataframe called `raw_df`

[2]:
```
# Insert your code below
# ======================
raw_df = pd.read_csv('videogames.csv')
```

2. Print the first `five` rows of the dataframe

[3]:
```
# Insert your code below
# ======================
raw_df.head(5)
```

[3]:

	name	platform	year	genre	publisher	north_america_sales	europe_sales
0	Imagine: Fashion Stylist	DS	2010.0	Simulation	Ubisoft	0.13	0.00
1	Monsters vs. Aliens	Wii	2009.0	Action	Activision	0.14	0.09
2	LEGO Racers	PS	1999.0	Racing	LEGO Media	0.20	0.14
3	The X-Factor	PS3	2010.0	Misc			
4	Houkago Shounen	DS	2008.0	Misc			

```
3                   Deep Silver                 0.00          0.10
4  Konami Digital Entertainment                 0.00          0.00

   japan_sales  global_sales
0         0.00          0.14
1         0.00          0.26
2         0.00          0.36
3         0.00          0.14
4         0.05          0.05
```

3. How many unique values exist in each of the columns genre and publisher?

[4]:
```
# Insert your code below
# ======================
num_genres = raw_df['genre'].nunique(dropna=True) # do not count NaN values
num_publisher = raw_df['publisher'].nunique(dropna=True) # do not count NaN␣
 ↪values
print('Number of unique values of genre: ', num_genres,
      '. Number of unique values of publisher: ', num_publisher)
```

```
Number of unique values of genre:  12 . Number of unique values of publisher:
486
```

4. Identify missing (NaN) values in the dataset

[5]:
```
# Insert your code below
# ======================
raw_df.isnull().values.any() # check if there is any nan values, True, find␣
 ↪where
```

[5]:
```
True
```

[6]:
```
raw_df.isnull().any() #which column has nan values.
```

[6]:
```
name                 False
platform             False
year                 False
genre                False
publisher             True
north_america_sales  False
europe_sales         False
japan_sales          False
global_sales         False
dtype: bool
```

We saw from the command above that the column 'publisher' is the only column that contains nan values. Let us see which rows of publisher (id) has nan values.

[7]:
```
nan_values_publisher = list(raw_df[raw_df['publisher'].isnull()].index)
nan_values_publisher
```

[7]:
```
[159,
 173,
 370,
 1359,
 2075,
 2211,
 2417,
 3161,
 3240,
 4973,
 5190,
 5474,
 5564,
 5808,
 5956,
 6393,
 8075,
 8123,
 8786,
 9159,
 9641,
 10509,
 10552,
 10672,
 11435,
 11504,
 11949]
```

5. Create a copy of `raw_df` named `df`. Remove any rows containing NaN values in the new dataframe. What is the shape of `df` before and after removing the NaN values?

[8]:
```
# Insert your code below
# ======================
df = raw_df.copy()
df.shape # shape before removing rows with nan values
```

[8]:
```
(12441, 9)
```

[9]:
```
df = df.drop(nan_values_publisher)
df.shape # shape after dropping rows with nan values
```

[9]:
```
(12414, 9)
```

The number of rows were reduced from 12441 to 12414 when dropping NaN values.

6. Which `platform`, `genre` and `publisher` is the most popular (by number of instances)

of all time? Print the name and count of each

Hint: The output should look something like this:

```
Column: [col], Most popular: [name], Count: [count]
Column: [col], Most popular: [name], Count: [count]
Column: [col], Most popular: [name], Count: [count]
```

[10]:
```
# Insert your code below
# ======================
columns = ['platform', 'genre', 'publisher']
for column in columns:
    pop = df[column].value_counts().idxmax()
    count = df[df[column]==pop].shape[0]
    print('Column: ', column, ', Most popular: ', pop, ', Count: ', count)
```

```
Column:  platform , Most popular:  PS2 , Count:  2120
Column:  genre , Most popular:  Action , Count:  2039
Column:  publisher , Most popular:  Electronic Arts , Count:  1087
```

7. What is the most popular game for each region? E.g. North America, Europe and Japan and globally

[11]:
```
# Insert your code below
# ======================
# We first find the row number of the most popular game in each region:
game_id_north_america = df['north_america_sales'].idxmax()
game_id_europe = df['europe_sales'].idxmax()
game_id_japan = df['japan_sales'].idxmax()
game_id_global = df['global_sales'].idxmax()

# We then find the name of the most popular game in each region:
game_north_america = df.loc[game_id_north_america, 'name']
game_europe = df.loc[game_id_europe, 'name']
game_japan = df.loc[game_id_japan, 'name']
game_global = df.loc[game_id_global, 'name']

print(' North Americas most popular game: ',game_north_america,
      '\n Europes most popular game: ', game_europe,
      '\n Japans most popular game: ', game_japan,
     '\n The most popular game globally: ',game_global)
```

```
North Americas most popular game:  Wii Sports
Europes most popular game:  Wii Sports
Japans most popular game:  Pokemon Red/Pokemon Blue
The most popular game globally:  Wii Sports
```

8. Create a new dataframe called `df_pokemon` containing all games with "pokemon" in the `name`

Hint: Make sure your filtering is NOT case-sensitive

a. What is the most-sold (globally) Pokemon game of all time?
b. How many Pokemon-games are in the dataset?

[12]:
```
# Insert your code below
# ======================
# Assume we are going to filter the df dataframe, not raw_df.
import re # need this to ignore case
data = df[df['name'].str.contains('pokemon',flags=re.IGNORECASE)]
df_pokemon = pd.DataFrame(data)
```

To find the most sold pokemon game globally we find the max in the column 'global_sales' in df_pokemon which gives us index, then find name of this game.

[13]:
```
pokemon_id = df_pokemon['global_sales'].idxmax()
pokemon_name = df_pokemon.loc[pokemon_id, 'name']
print('The most popular pokemon game globally is ',pokemon_name)
```

```
The most popular pokemon game globally is  Pokemon Red/Pokemon Blue
```

To find how many pokemon games are in the dataset we find the number of unique names in df_pokemon. Which is the same as number of rows in the dataset.

[14]:
```
num_pokemon_games = len(df_pokemon['name'].unique())
num_pokemon_games
```

[14]: 25

1.3 Visualizing the dataset

9. Create plot with 2 vertical axes and one horizontal axes. The plot should display a barchart containing the `count` of the `10 most frequent` genres and platforms, each in its own subplot. The bars should be sorted in descending order.

Hint: It is recommended to use the `Barplot` function built into Seaborn for barcharts.

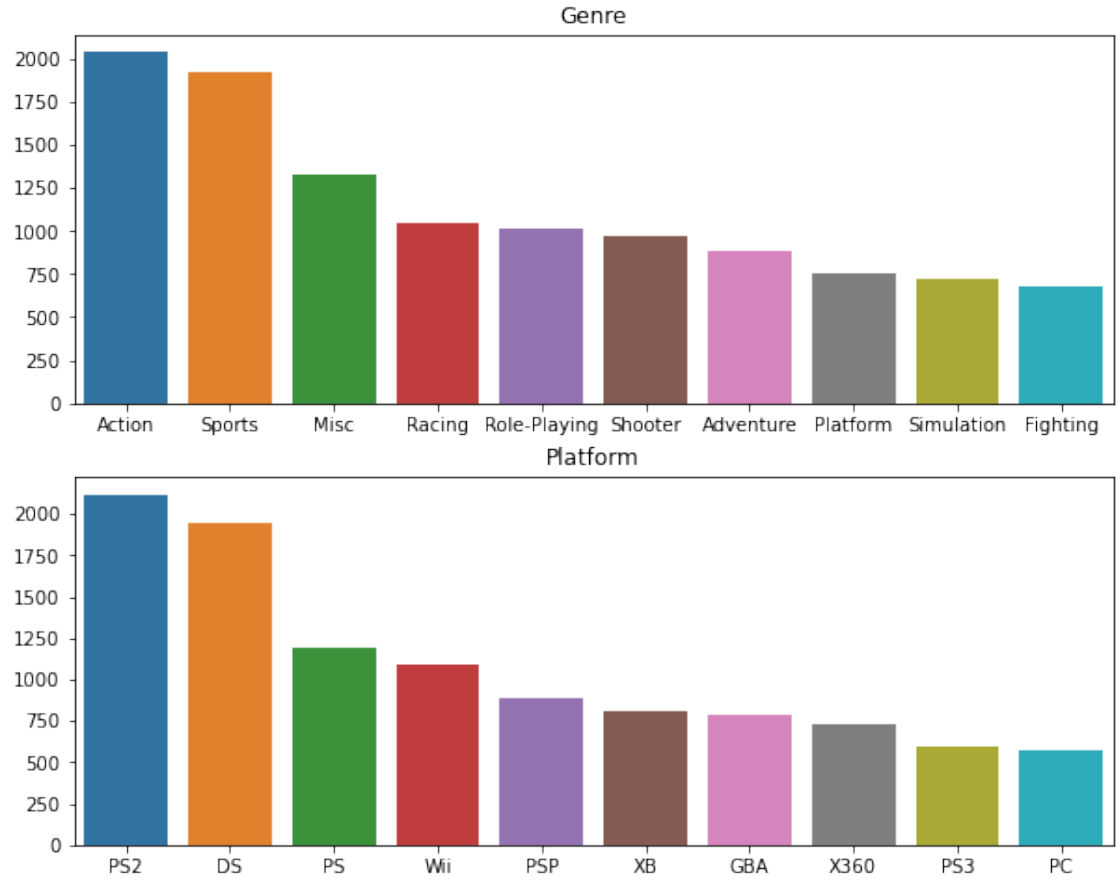
The output should look something like this:

PS: Disregard the color scheme of the example image.

[15]:
```
# Insert your code below
# ======================
# 10 most frequent genre in descending order
frequent_genre = list(df['genre'].value_counts()[:10].index)
# their counts
genre_counts = list(df['genre'].value_counts()[:10])
#10 most frequent platform in descending order
frequent_platform = list(df['platform'].value_counts()[:10].index)
#their counts
platform_counts = list(df['platform'].value_counts()[:10])
```

```
# plot
fig, axs = plt.subplots(2, figsize=(10,8))
axs[0].set_title('Genre')
sns.barplot(ax=axs[0], x=frequent_genre, y=genre_counts)
axs[1].set_title('Platform')
sns.barplot(ax=axs[1],x=frequent_platform, y=platform_counts)
```

[15]: <matplotlib.axes._subplots.AxesSubplot at 0x27050267048>



10. Group sales in `df` for the four different regions *by year* into a new dataframe; `yearly_sales`

Hint: Sales should be aggregated by the `sum` of all sales within the year.

The first five rows of the new dataframe should look like this:

[16]:
```
# Insert your code below
# ======================
grouped_df = df.groupby(df['year']).agg(list)
```

[17]:
```
# set the index of yearly_saels to be the year.
yearly_sales = pd.DataFrame(index=grouped_df.index)
# let the values of the columns *_sales be the sum of all sales that year.
yearly_sales['north_america_sales'] = [sum(x) for x in␣
 ↪grouped_df['north_america_sales']]
yearly_sales['europe_sales'] = [sum(x) for x in grouped_df['europe_sales']]
yearly_sales['japan_sales'] = [sum(x) for x in grouped_df['japan_sales']]
yearly_sales['global_sales'] = [sum(x) for x in grouped_df['global_sales']]
yearly_sales.head()
```

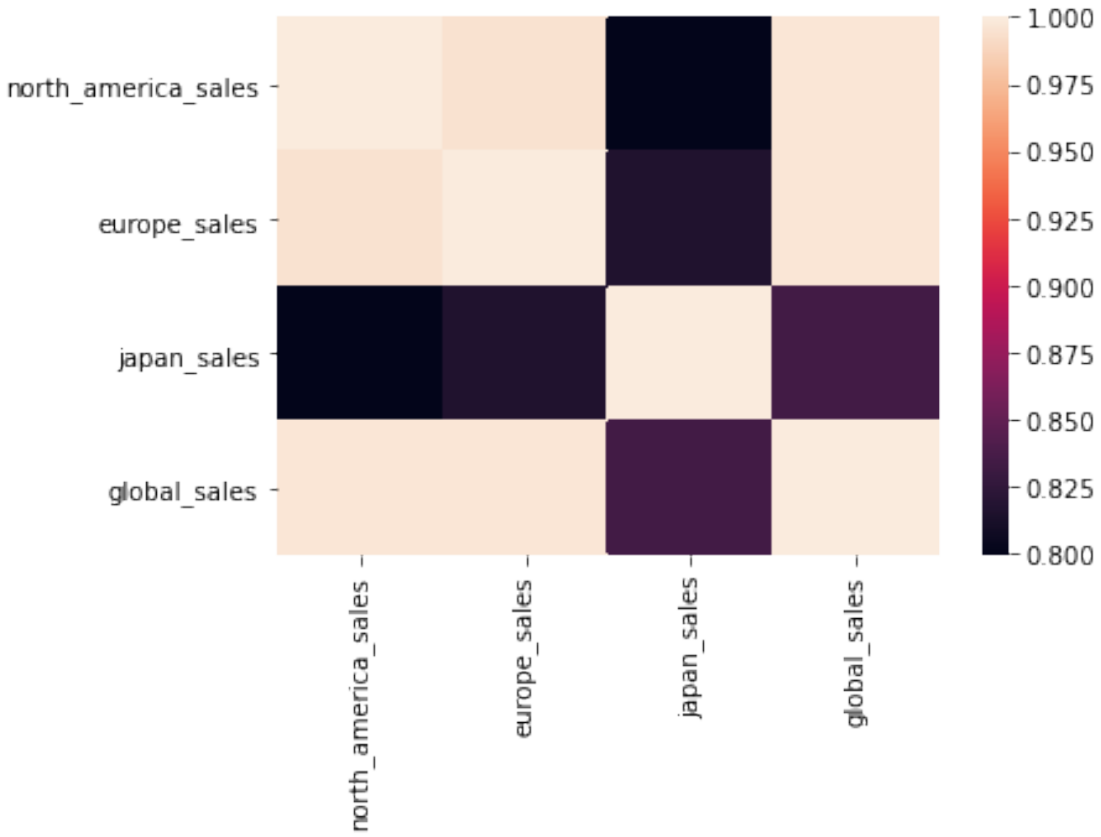
[17]:

year	north_america_sales	europe_sales	japan_sales	global_sales
1980.0	10.59	0.67	0.00	11.38
1981.0	33.40	1.96	0.00	35.77
1982.0	26.92	1.65	0.00	28.86
1983.0	7.76	0.80	8.10	16.79
1984.0	33.28	2.10	14.27	50.36

11. Create a correlation matrix (based on the aggregated sales data) and plot it as a heatmap using Seaborn. What does the plot tell you about correlation between sales in the given regions?

[18]:
```
# Insert your code below
# ======================
correlations = yearly_sales.corr()
sns.heatmap(correlations)
```

[18]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x270502b4438>
```



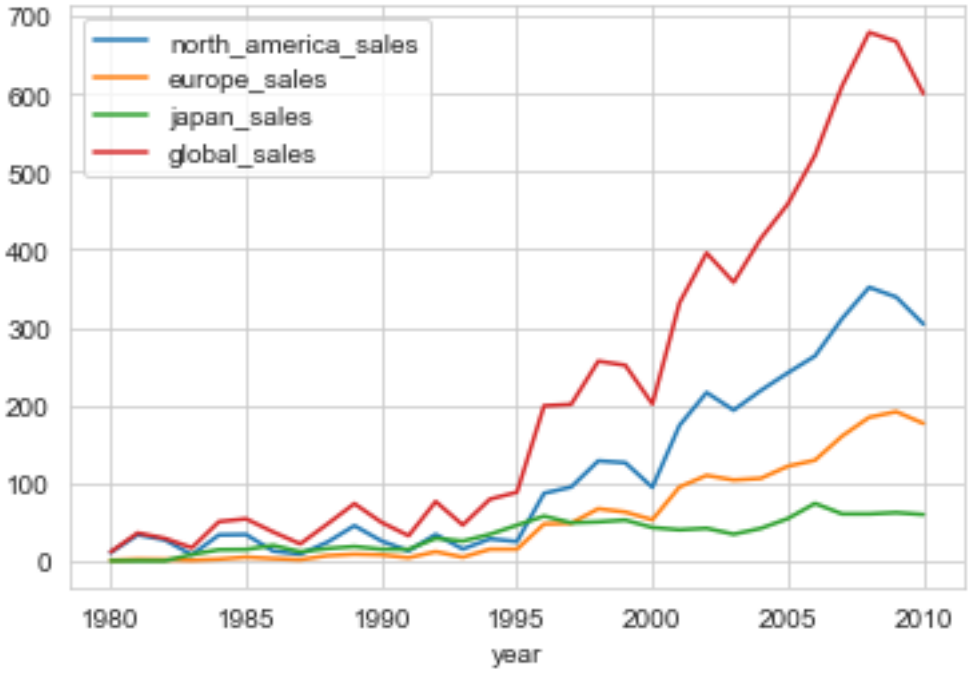
We can see the amount of correlation between the markets in north america, europe, japan and globally by examining the colors of the heatmap above. The correlation between all the markets except from the one in Japan are highly correlated, correlation is close to 1 (between 0.975 and 1). The market in Japan seems to be more independent from the others as the correlation is lower, however there is still a high correlation (more than 0.8).

12. Create a lineplot showing sales per year for all four regions over the entire period

The end result should look something like this:

[19]:
```
sns.set_style('whitegrid')
sns.lineplot(data=yearly_sales, dashes=False)
```

[19]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x2705035ab38>
```


north_america_sales
europe_sales
japan_sales
global_sales
700
600
500
400
300
200
100
0
1980
1985
1990
1995
2000
2005
2010
year