SQL_TEST

May 2, 2022

Candidate: Andrés Felipe Bolaños Acosta

Position: Business Intelligence Developer

E-mail: afb.acosta@gmail.com

Mobile phone: +5511973662765

In order to execute SQL commands to query information from the database's table two functions were implemented "connect" and "search query". The first one allows us to connect directly to the mySQL data base and the second one allow us to make queries:

[7]:
```
import mysql.connector

def connect():
    connection = (mysql.connector.connect(
                                    user = 'looqbox-challenge',
                                    password = 'looq-challenge',
                                    host = '35.199.127.241',
                                    database = 'looqbox_challenge'))
    return connection

def search_query(query, conection_database):
    cursor = conection_database.cursor()
    cursor.execute(query)
    items_query = cursor.fetchall()

    for item in items_query:
        print(item)

    cursor.close()
```

[8]:
```
conection_database = connect()
```

1 What are the 10 most expensive products in the company?

[9]:
```
query = ("SELECT PRODUCT_NAME FROM data_product "
         "ORDER BY PRODUCT_VAL DESC LIMIT 10 ")
search_query(query, conection_database)
```

```
('Whisky Escoces THE MACALLAN Ruby Garrafa 700ml com Caixa',)
('Whisky Escoces JOHNNIE WALKER Blue Label Garrafa 750ml',)
('Cafeteira Expresso 3 CORACOES Tres Modo Vermelho',)
('Vinho Portugues Tinto Vintage QUINTA DO CRASTO Garrafa 750ml',)
('Escova Dental Eletrica ORAL B D34 Professional Care 5000 110v',)
('Champagne Rose VEUVE CLICQUOT PONSARDIM Garrafa 750ml',)
('Champagne Frances Brut Imperial MOET Rose Garrafa 750ml',)
('Conjunto de Panelas Allegra em Inox TRAMONTINA 5 Pecas Gratis Utensilios 5
Pecas',)
('Whisky Escoces CHIVAS REGAL 18 Anos Garrafa 750ml',)
('Champagne Frances Brut Imperial MOET & CHANDON Garrafa 750ml',)
```

2 What sections do the 'BEBIDAS' and 'PADARIA' departments have?:

[10]:
```
query = ("SELECT DISTINCT DEP_NAME, SECTION_NAME "
         "FROM data_product WHERE DEP_NAME = 'BEBIDAS' "
         "OR DEP_NAME = 'PADARIA'ORDER BY DEP_NAME ")
search_query(query, conection_database)
```

```
('BEBIDAS', 'BEBIDAS')
('BEBIDAS', 'CERVEJAS')
('BEBIDAS', 'REFRESCOS')
('BEBIDAS', 'VINHOS')
('PADARIA', 'DOCES-E-SOBREMESAS')
('PADARIA', 'GESTANTE')
('PADARIA', 'PADARIA')
('PADARIA', 'QUEIJOS-E-FRIOS')
```

The department 'BEBIDAS' has sections 'BEBIDAS', 'CERVEJAS', 'REFRESCOS' and 'VINHOS'. While, "DOCES-E-SOBREMESAS", "GESTANTE", "PADARIA" and "QUEIJOS-E-FRIOS" correspond to sections of 'PADARIA'

3 What was the total sale of products (in dollars) of each Business Area in the first quarter of 2019?

[11]:
```
query = ("SELECT A.BUSINESS_NAME, SUM(B.SALES_VALUE) "
         "FROM data_store_cad AS A "
         "INNER JOIN data_product_sales AS B "
         "ON A.STORE_CODE = B.STORE_CODE "
         "WHERE B.DATE >= '2019-01-01' AND B.DATE <= '2019-03-31' "
         "GROUP BY A.BUSINESS_NAME")
search_query(query, conection_database)
```

```
('Atacado', Decimal('80384884.60'))
('Farma', Decimal('81776691.73'))
('Posto', Decimal('32072326.40'))
('Proximidade', Decimal('80171122.80'))
('Varejo', Decimal('81032347.65'))
```

Making the previous result clear:

ATACADO: 80,384,884.60

FARMA: 81,776,691.73

POSTO: 32,072,326.40

PROXIMIDADE: 80,171,122.80

VAREJO: 81,032,347.65

The same values are also achieved if we apply another ”INNER JOIN” to the tables ”data_store_cad” and ”data_store_sales”:

[12]:
```
query = ("SELECT A.BUSINESS_NAME, SUM(B.SALES_VALUE) "
         "FROM data_store_cad AS A "
         "INNER JOIN data_store_sales AS B "
         "ON A.STORE_CODE = B.STORE_CODE "
         "WHERE B.DATE >= '2019-01-01' AND B.DATE <= '2019-03-31' "
         "GROUP BY A.BUSINESS_NAME ")
search_query(query, conection_database)
```

```
('Atacado', 80384884.6)
('Farma', 81776691.73)
('Posto', 32072326.4)
('Proximidade', 80171122.8)
('Varejo', 81032347.65)
```

It should be mentioned that “Atacado”, “Farma”, “Posto”, “Proximidade” and “Varejo” are all the business areas in the table “data_store_cad”. However, a particular case occurs when any of those business areas did not have sales in the first quarter of 2019, just for that case a “LEFT JOIN” (“data_store_cad” being the left table) is appropriate to show that any of those areas did not generated income (but this is not the case).

CASE_1

May 2, 2022

Candidate: Andrés Felipe Bolaños Acosta

Position: Business Intelligence Developer

E-mail: afb.acosta@gmail.com

Mobile phone: +5511973662765

1 Case

The Dev Team was tired of developing the same old queries just varying the filters accordingly to their boss demands:

As a new member of the crew, your mission now is to create a dynamic function, on the most flexible of ways, to produce queries and retrieve a dataframe based on three parameters:

- product_code: integer
- store_code: integer
- date: list of ISO-like strings

ANSWER:

[1]:
```
import mysql.connector
import pandas as pd
import seaborn as sns
from matplotlib import pyplot as plt
import warnings
warnings.filterwarnings('ignore')
```

The first solution is implemented by employing the tool "mySQL conector" and "pd.read_sql". Both allow us to establish a direct connection to the database and make queries:

[2]:
```
def retrieve_data(product_code, store_code, date):
    connection = (mysql.connector.connect(
                                    user = 'looqbox-challenge',
                                    password = 'looq-challenge',
                                    host = '35.199.127.241',
                                    database = 'looqbox_challenge'))

    query = ("SELECT STORE_CODE, PRODUCT_CODE, "
```

```
                    "DATE, SALES_VALUE, SALES_QTY "
              "FROM data_product_sales "
              "WHERE (STORE_CODE = '%s') "
              "AND (PRODUCT_CODE = '%s') "
              "AND (DATE BETWEEN '%s' AND '%s') "
              % (store_code, product_code, date[0], date[1]))

    df_query = pd.read_sql(query, con = connection)
    return df_query
```

Testing the first approach:

[3]:
```
product_code = 18
store_code = 18
date = ["2019-01-01", "2019-01-12"]
```

[4]:
```
retrieve_data(product_code, store_code, date).tail(3)
```

[4]:

	STORE_CODE	PRODUCT_CODE	DATE	SALES_VALUE	SALES_QTY
9	18	18	2019-01-10	1068.2	98.0
10	18	18	2019-01-11	1057.3	97.0
11	18	18	2019-01-12	806.6	74.0

However, a second approach has been implemented by just employing permitted actions from the tool "myqsl.connector", which becomes a sort of a raw version of the first function:

[5]:
```
def retrieve_data(product_code, store_code, date):
    connection = (mysql.connector.connect(
                                    user = 'looqbox-challenge',
                                    password = 'looq-challenge',
                                    host = '35.199.127.241',
                                    database = 'looqbox_challenge'))

    query = ("SELECT STORE_CODE, PRODUCT_CODE, "
                    "DATE, SALES_VALUE, SALES_QTY "
                    "FROM data_product_sales "
                    "WHERE (STORE_CODE = '%s') "
                    "AND (PRODUCT_CODE = '%s') "
                    "AND (DATE BETWEEN '%s' AND '%s') "
                    % (store_code, product_code,
                       date[0], date[1]))

    cursor = connection.cursor()
    cursor.execute(query)
    items_query = cursor.fetchall()
    df_items_query = pd.DataFrame(items_query)
```

```
    if len(df_items_query) == 0:
        print("There are not a sales!"
              " please check your query")
        return None

    else:
        query_columns = "SHOW COLUMNS FROM data_product_sales"
        cursor.execute(query_columns)
        query_columns = cursor.fetchall()
        cursor.close()

        lst_columns = []
        for i in range(len(query_columns)):
            lst_columns.append(query_columns[i][0])

        df_items_query.columns = lst_columns
        return df_items_query
```

Testing the implemented function:

[6]:
```
product_code = 20
store_code = 20
date = ["2019-01-01", "2019-01-12"]
```

[7]:
```
retrieve_data(product_code, store_code, date)
```

```
There are not a sales! please check your query
```

[8]:
```
product_code = 18
store_code = 18
date = ["2019-01-01", "2019-01-12"]
```

[9]:
```
retrieve_data(product_code, store_code, date).tail(3)
```

[9]:

	STORE_CODE	PRODUCT_CODE	DATE	SALES_VALUE	SALES_QTY
9	18	18	2019-01-10	1068.20	98
10	18	18	2019-01-11	1057.30	97
11	18	18	2019-01-12	806.60	74

CASE_2

May 2, 2022

Candidate: Andrés Felipe Bolaños Acosta

Position: Business Intelligence Developer

E-mail: afb.acosta@gmail.com

Mobile phone: +5511973662765

[2]:
```
import mysql.connector
import pandas as pd
import seaborn as sns
import warnings
from matplotlib import pyplot as plt

warnings.filterwarnings('ignore')
```

Connecting to the database:

[3]:
```
def connect():
    connection = (mysql.connector.connect(
                                  user = 'looqbox-challenge',
                                  password = 'looq-challenge',
                                  host = '35.199.127.241',
                                  database = 'looqbox_challenge'))
    return connection

conection_database = connect()
```

1 Case

1.1 A brand new client sent you two ready-to-go queries. Those are listed below:

Query 1:

[4]:
```
query_1 = ("SELECT "
               "STORE_CODE, "
               "STORE_NAME, "
```

```
            "START_DATE, "
            "END_DATE, "
            "BUSINESS_NAME, "
            "BUSINESS_CODE "
        "FROM data_store_cad ")
```

Query 2:

[5]:
```
query_2 = ("SELECT "
            "STORE_CODE, "
            "DATE, "
            "SALES_VALUE, "
            "SALES_QTY "
        "FROM data_store_sales "
        "WHERE DATE BETWEEN '2019-01-01' AND '2019-12-31' ")
```

In addition, he gave you this set of instructions:

- You must not modify my queries!
- Please filter the period between this given range:
  - ['2019-10-01','2019-12-31']

ANSWER:

First, it is necessary to execute the queries, which were sent by the client. For this purpose, "pd.read_sql" (it creates a dataframe from a query in the MySQL database) is employed as follows:

[6]:
```
df_store_cad = pd.read_sql(query_1, con = conection_database)
df_store_cad.head()
```

[6]:

	STORE_CODE	STORE_NAME	START_DATE	END_DATE	BUSINESS_NAME	BUSINESS_CODE
0	1	Sao Paulo	2006-10-01		Varejo	1
1	2	Chicago	2007-10-01		Varejo	1
2	3	Roma	2008-10-01		Varejo	1
3	4	Tokio	2009-10-01		Varejo	1
4	5	Paris	2019-01-01		Proximidade	2

[7]:
```
df_store_sales = pd.read_sql(query_2, con = conection_database)
df_store_sales.head()
```

[7]:

	STORE_CODE	DATE	SALES_VALUE	SALES_QTY
0	1	2019-01-01	196623.22	12838
1	10	2019-01-01	126795.44	4933
2	11	2019-01-01	223937.00	7724
3	12	2019-01-01	200251.80	7043
4	13	2019-01-01	196623.22	12838

Now, the filter dates ('2019-10-01','2019-12-31') are going to be formated:

[8]:
```
filter_date_initial = pd.to_datetime("2019-10-01").date()
filter_date_final = pd.to_datetime("2019-12-31").date()
```

Then, the filters are going to be applied to the dataframe that resulted from the client's query ("df_query_2").

[9]:
```
df_store_sales = df_store_sales[(df_store_sales.DATE
                                 >= filter_date_initial)
                                 & (df_store_sales.DATE
                                    <= filter_date_final)]
```

Just checking the boundaries of the filters within the dataframe:

[10]:
```
df_store_sales.head(3)
```

[10]:

	STORE_CODE	DATE	SALES_VALUE	SALES_QTY
5460	1	2019-10-01	187601.54	12160
5461	10	2019-10-01	139038.86	5223
5462	11	2019-10-01	252687.35	8481

[11]:
```
df_store_sales.tail(3)
```

[11]:

	STORE_CODE	DATE	SALES_VALUE	SALES_QTY
7297	7	2019-12-31	193619.94	12607
7298	8	2019-12-31	191704.64	12500
7299	9	2019-12-31	167081.21	5765

Now, a left join between "df_query_1" and "df_query_2" will be applied. The dataframe "df_query_1", which comes from the table "data_store_cad" in the MySQL database, correspond to the left table because there is a risk that certain shops as well as business areas did not sell products during the filtered period. In that case the values of the right table will represented by NaN values.

[12]:
```
df_business_cad = df_store_cad.merge(df_store_sales,
                                     on = "STORE_CODE",
                                     how = "left")
df_business_cad.head(3)
```

[12]:

	STORE_CODE	STORE_NAME	START_DATE	END_DATE	BUSINESS_NAME	BUSINESS_CODE	\
0	1	Sao Paulo	2006-10-01		Varejo	1	
1	1	Sao Paulo	2006-10-01		Varejo	1	
2	1	Sao Paulo	2006-10-01		Varejo	1	

	DATE	SALES_VALUE	SALES_QTY
0	2019-10-01	187601.54	12160
1	2019-10-02	332666.14	21643
2	2019-10-03	253401.57	16489

Looking for missing data in the dataframe:

[13]:
```
df_business_cad.isna().sum()
```

[13]:
```
STORE_CODE       0
STORE_NAME       0
START_DATE       0
END_DATE         0
BUSINESS_NAME    0
BUSINESS_CODE    0
DATE             0
SALES_VALUE      0
SALES_QTY        0
dtype: int64
```

There were not NaN values within the merged dataframe's dimensions. This means that values of "STORE_CODE" were in both tables, consequently, the obtained dataframe can be also attained by using an "inner join" instead.

At this time, it is suitable to observe the unique cities and businesses areas to develop some visualizations:

[14]:
```
df_business_cad.STORE_NAME.unique()
```

[14]:
```
array(['Sao Paulo', 'Chicago', 'Roma', 'Tokio', 'Paris', 'Berlin',
       'New York', 'Belem', 'London', 'Hong Kong', 'Rio de Janeiro',
       'Madri', 'Dubai', 'Bahia', 'Buenos Aires', 'Salvador', 'Sidney',
       'Bangkok', 'Miami', 'Vancouver'], dtype=object)
```

[15]:
```
df_business_cad.BUSINESS_NAME.unique()
```

[15]:
```
array(['Varejo', 'Proximidade', 'Farma', 'Atacado', 'Posto'], dtype=object)
```

1.1.1 Visualizations

Per business areas:

Sales: Before plotting a chart the data was grouped by the 'BUSINESS_NAME' in the filtered period.

[16]:
```
df_business = df_business_cad[["BUSINESS_NAME",
                               "SALES_VALUE",
                               "SALES_QTY"]].copy()
```

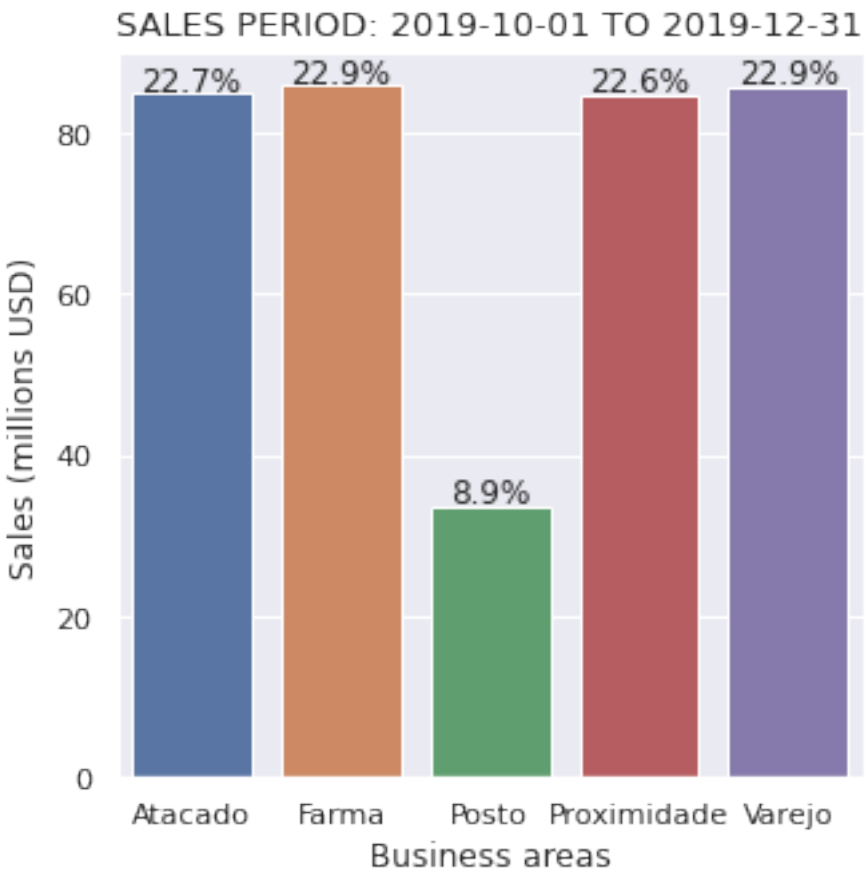
[17]:
```
df_business = df_business.groupby("BUSINESS_NAME",
                                  as_index = True).\
                                  agg({"SALES_VALUE": "sum",
```

```
                                        "SALES_QTY": 'sum'})
```

[18]:
```
df_business["SALES_VALUE"] = (df_business["SALES_VALUE"]
                              / 1e+06)
```

[19]:
```
df_business_per = (df_business.div(df_business.
                                   sum(axis = 0),
                                   axis = 1).multiply(100))
```

[20]:
```
plt.figure(figsize = (5, 5))
sns.set_theme(style = "darkgrid")
fig_obj = sns.barplot(x = df_business.index,
                      y = df_business.SALES_VALUE,
                      data = df_business)
patches = fig_obj.patches

for patch in range (len(patches)):
    x = (patches[patch].get_x()
         + patches[patch].get_width() / 2)
    y = patches[patch].get_height() + 0.5
    fig_obj.annotate('{:.1f}%'.format
                     (df_business_per.SALES_VALUE[patch]),
                     (x, y), ha = 'center')
fig_obj.set_title("SALES PERIOD: 2019-10-01 TO 2019-12-31",
                  fontsize = 13)
fig_obj.set_xlabel("Business areas")
fig_obj.set_ylabel("Sales (millions USD)")
plt.show()
```

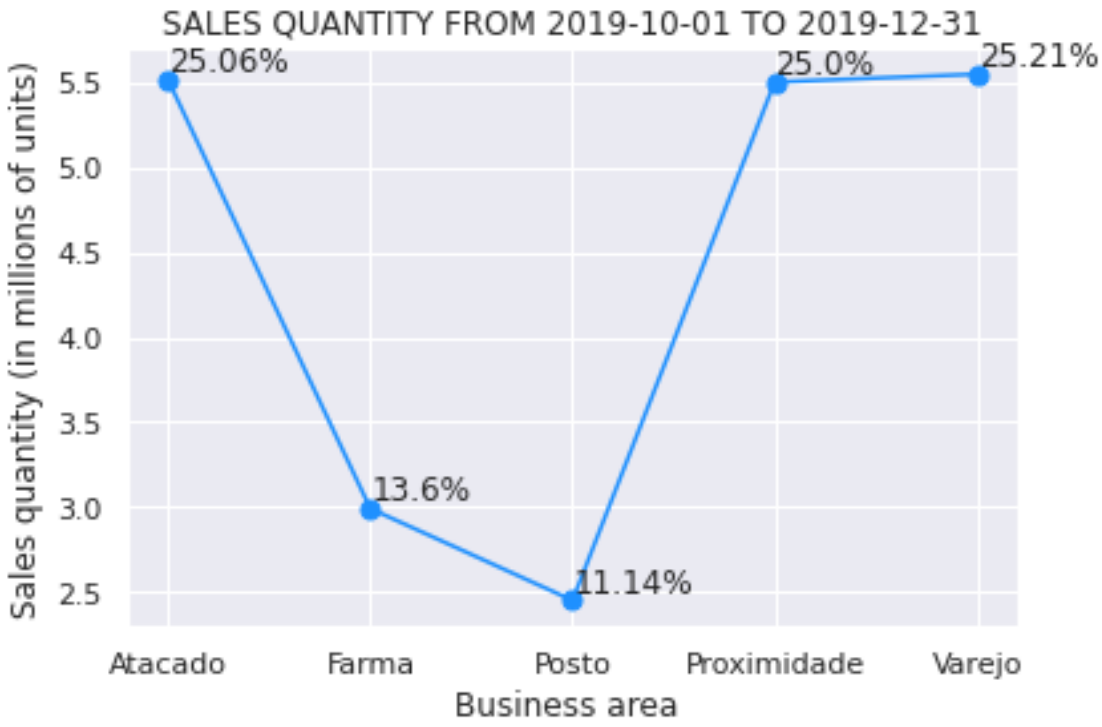


Quantity:

[21]:
```
text_percentages = []
for i in range(len(df_business_per.SALES_QTY)):
    round_val = round(df_business_per.SALES_QTY[i], 2)
    text_percentages.append(str(round_val) + "%")

text_x = df_business.index
text_y = (df_business.SALES_QTY / 1e+06
          + 0.05)
```

[22]:
```
plt.figure(figsize = (6, 4))
plt.title("SALES QUANTITY "
          "FROM 2019-10-01 TO 2019-12-31")


plt.xlabel("Business area ")
plt.ylabel("Sales quantity (in millions of units)")
plt.plot(df_business.index, df_business.SALES_QTY / 1e+06,
```

```
        marker = "o", ms = 7, color = 'dodgerblue',
        lw = 1.5)

for j in range(len(df_business.index)):
    plt.annotate(text_percentages[j],
                 xy = (text_x[j], text_y[j]))

plt.show()
```



Per store:

Sales: The df_business_cad's procedure was also applied to the dataframe that grouped both sales and quantity concerning the "STORE_NAME" as follows:

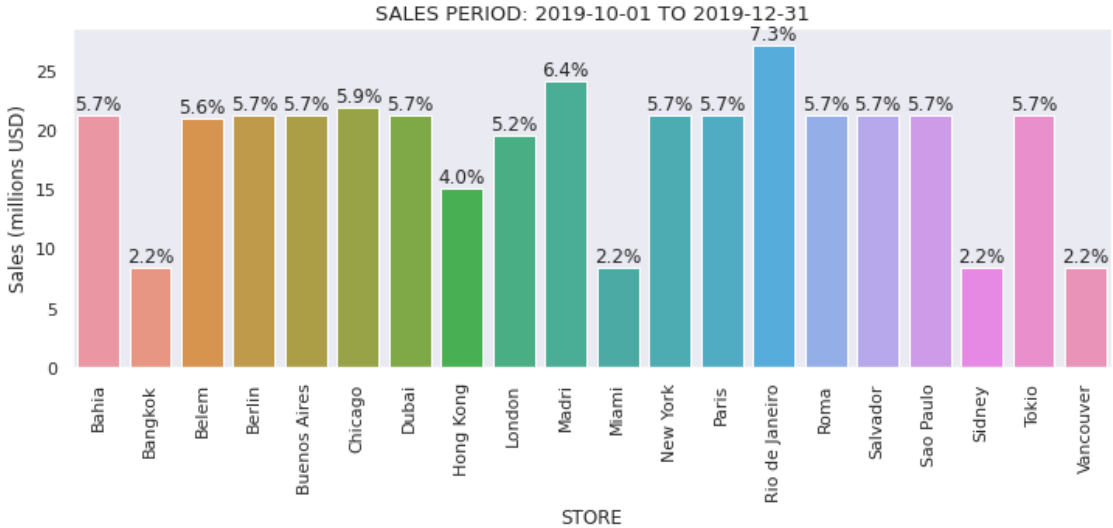
[23]:
```
df_store_sales = df_business_cad[["STORE_NAME",
                                  "SALES_VALUE",
                                  "SALES_QTY"]].copy()
```

[24]:
```
df_store_sales = (df_store_sales.groupby("STORE_NAME",
                  as_index = True).agg({"SALES_VALUE": "sum",
                                        "SALES_QTY": 'sum'}))
```

[25]:
```
df_store_sales["SALES_VALUE"] = (df_store_sales["SALES_VALUE"]
                                 / 1e+06)
```

[26]:
```
df_store_sales_per = (df_store_sales.div(df_business.
                                         sum(axis = 0),
                                         axis = 1).multiply(100))
```

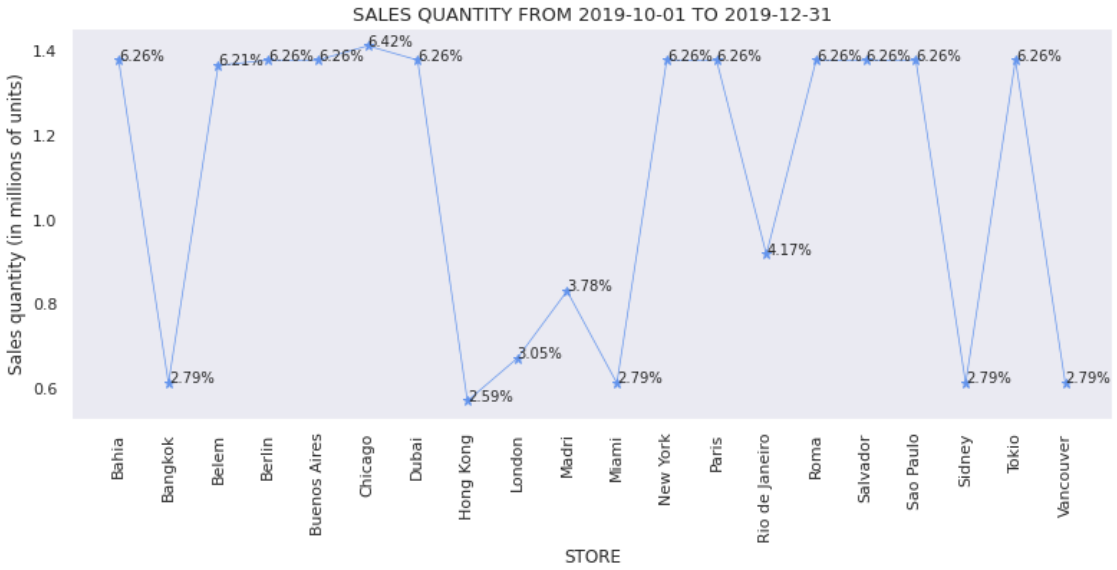
[27]:
```
plt.figure(figsize = (12, 4))
sns.set_theme(style = "dark")
fig_obj = sns.barplot(x = df_store_sales.index,
                      y = df_store_sales.SALES_VALUE,
                      data = df_store_sales)
patches = fig_obj.patches

for patch in range (len(patches)):
    x = (patches[patch].get_x()
         + patches[patch].get_width() / 2)
    y = patches[patch].get_height() + 0.5
    fig_obj.annotate('{:.1f}%'.format
                     (df_store_sales_per.SALES_VALUE[patch]),
                     (x, y), ha = 'center')

fig_obj.set_title("SALES PERIOD: 2019-10-01 TO 2019-12-31",
                  fontsize = 13)
fig_obj.set_xticklabels(fig_obj.get_xticklabels(),
                        rotation = 90)
fig_obj.set_xlabel("STORE")
fig_obj.set_ylabel("Sales (millions USD)")
plt.show()
```



Quantity:

[28]:
```
text_percentages = []
for i in range(len(df_store_sales_per.SALES_QTY)):
    round_val = round(df_store_sales_per.SALES_QTY[i], 2)
    text_percentages.append(str(round_val) + "%")

text_x = df_store_sales.index
text_y = (df_store_sales.SALES_QTY / 1e+06)
```

[29]:
```
plt.figure(figsize = (13, 5))
plt.title("SALES QUANTITY "
          "FROM 2019-10-01 TO 2019-12-31",
          fontsize = 13)


plt.xlabel("STORE")
plt.ylabel("Sales quantity (in millions of units)")
plt.plot(df_store_sales.index,
         df_store_sales.SALES_QTY / 1e+06,
         marker = "*", ms = 7,
         color = "cornflowerblue",
         lw = 0.7)
plt.xticks(rotation = 90)

for i in range(len(df_store_sales.index)):
    plt.annotate(text_percentages[i],
                 xy = (text_x[i], text_y[i]),
                 fontsize = 10)

plt.show()
```



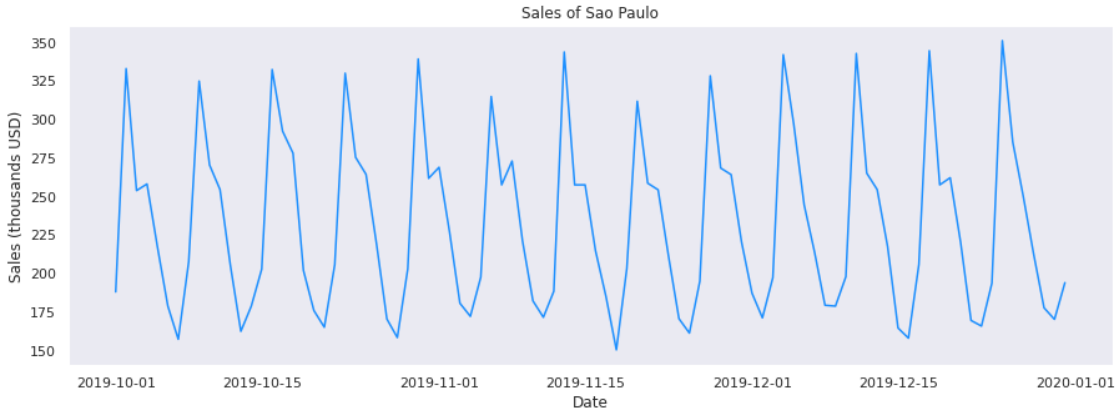
A dynamic function: Aside from the previous charts, a dynamic function is implemented "sales_chart", its inputs are "filter_type", which can be defined as "BUSINESS_NAME" or "STORE_NAME". In case of "BUSINESS_NAME", the filter_value is determined by "Varejo", "Atacado", etc. For "STORE_NAME", it receives values such as "Sao Paulo", "Sidney", etc. Finally, the last input corresponds to the table namely as "df_data":

[33]:
```
def sales_chart(filter_type, filter_value, df_data):
    df_data = df_data[(df_data[filter_type]
                       == filter_value)]

    plt.figure(figsize = (15, 5))
    sns.set_theme(style = "dark")
    fig = sns.lineplot(x = df_data.DATE,
                       y = (df_data.SALES_VALUE
                            / 1e03),
                       color = "dodgerblue",
                       data = df_data)

    fig.set_title("Sales of " + filter_value)
    fig.set_xlabel("Date")
    fig.set_ylabel("Sales (thousands USD)")

    plt.show()
```

[34]:
```
df_data = df_business_cad[["DATE",
                           "STORE_NAME",
                           "BUSINESS_NAME",
                           "SALES_VALUE"]].copy()
```

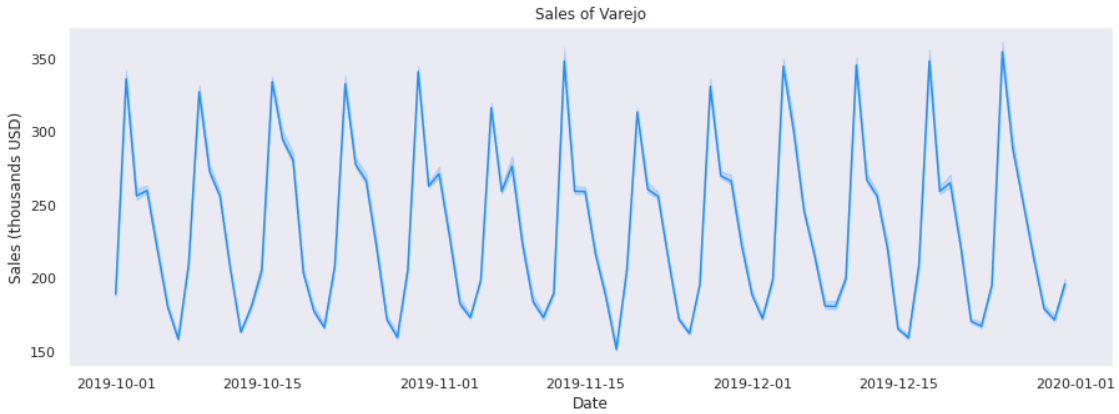
Illustrating the sales' historical data for the Sao Paulo's store:

[35]:
```
filter_type = "STORE_NAME"
filter_value = "Sao Paulo"
sales_chart(filter_type, filter_value, df_data)
```



Displaying the sales' historical data for the Sao Paulo's store:

[36]:
```
filter_type = "BUSINESS_NAME"
filter_value = "Varejo"
sales_chart(filter_type, filter_value, df_data)
```



CASE_3

May 2, 2022

Candidate: Andrés Felipe Bolaños Acosta

Position: Business Intelligence Developer

E-mail: afb.acosta@gmail.com

Mobile phone: +5511973662765

[1]:
```
import mysql.connector
import pandas as pd
import seaborn as sns
from matplotlib import pyplot as plt
import numpy as np
from dython.nominal import associations
import warnings

warnings.filterwarnings('ignore')
```

1 Case :

1.1 Building your own visualization:

Create at least one chart using the table IMDB_movies. The code must be in R or Python, and you are free to use any libraries, data in the table and graphic format. Explain why you chose the visualization (or visualizations) you are submitting.

ANSWER:

Since there is not an specific aim. The objective of the current analysis is to make insights in how a film should be produced to maximize its income:

Loading the database into a dataframe:

[2]:
```
def connect():
    connection = (mysql.connector.connect(
                                  user = 'looqbox-challenge',
                                  password = 'looq-challenge',
                                  host = '35.199.127.241',
                                  database = 'looqbox_challenge'))
```

```
    return connection
```

[3]:
```
conection_database = connect()
```

[4]:
```
query = ("SELECT Id, Title, Genre, Director, "
                  "Actors , Year , Runtime, Rating, "
                  "Votes , RevenueMillions, Metascore "
         "FROM IMDB_movies")
df_IMDB_movies = pd.read_sql(query, con = conection_database)
df_IMDB_movies.head(3)
```

[4]:
```
   Id                   Title                     Genre            Director  \
0   1  Guardians of the Galaxy   Action,Adventure,Sci-Fi          James Gunn
1   2               Prometheus  Adventure,Mystery,Sci-Fi        Ridley Scott
2   3                    Split           Horror,Thriller  M. Night Shyamalan

                                              Actors  Year  Runtime  Rating  \
0  Chris Pratt, Vin Diesel, Bradley Cooper, Zoe S…  2014      121     8.0
1  Noomi Rapace, Logan Marshall-Green, Michael Fa…  2012      124     7.0
2  James McAvoy, Anya Taylor-Joy, Haley Lu Richar…  2016      117     7.0

    Votes  RevenueMillions  Metascore
0  757074            333.0       76.0
1  485820            126.0       65.0
2  157606            138.0       62.0
```

Looking for missing data:

[5]:
```
df_IMDB_movies.isna().sum()
```

[5]:
```
Id                   0
Title                0
Genre                0
Director             0
Actors               0
Year                 0
Runtime              0
Rating               0
Votes                0
RevenueMillions    128
Metascore           64
dtype: int64
```

Replacing the ”ReveneuMillions” missing data with the mean value:

[6]:
```
mean_reveneu = (df_IMDB_movies.RevenueMillions.
                loc[(df_IMDB_movies.RevenueMillions)
                     != "NaN"].mean())
```

[7]:
```
df_IMDB_movies.RevenueMillions.fillna(
                        mean_reveneu, inplace = True)
```

Now, it should be stressed that a "METASCORE" is a grade determined by film's respected critics. consequently, it also can be filled with the mean value:

[8]:
```
mean_metascore = (df_IMDB_movies.Metascore.
                  loc[(df_IMDB_movies.Metascore)
                     != "NaN"].mean())
```

[9]:
```
df_IMDB_movies.Metascore.fillna(mean_metascore,
                                inplace = True)
```

Checking again the missing values:

[10]:
```
df_IMDB_movies.isna().sum()
```

[10]:
```
Id                  0
Title               0
Genre               0
Director            0
Actors              0
Year                0
Runtime             0
Rating              0
Votes               0
RevenueMillions     0
Metascore           0
dtype: int64
```

Now, according to the objective it is necessary to order the data in regard to the generated profit:

[11]:
```
df_IMDB_movies.sort_values(by = ["RevenueMillions"],
                           ascending = False,
                           inplace = True)
df_IMDB_movies.head(5)
```

[11]:

	Id	Title	Genre	\
50	51	Star Wars: Episode VII - The Force Awakens	Action,Adventure,Fantasy	
87	88	Avatar	Action,Adventure,Fantasy	
85	86	Jurassic World	Action,Adventure,Sci-Fi	
76	77	The Avengers	Action,Sci-Fi	
54	55	The Dark Knight	Action,Crime,Drama	

	Director	Actors	\
50	J.J. Abrams	Daisy Ridley, John Boyega, Oscar Isaac, Domhna…	
87	James Cameron	Sam Worthington, Zoe Saldana, Sigourney Weaver…	
85	Colin Trevorrow	Chris Pratt, Bryce Dallas Howard, Ty Simpkins,…	

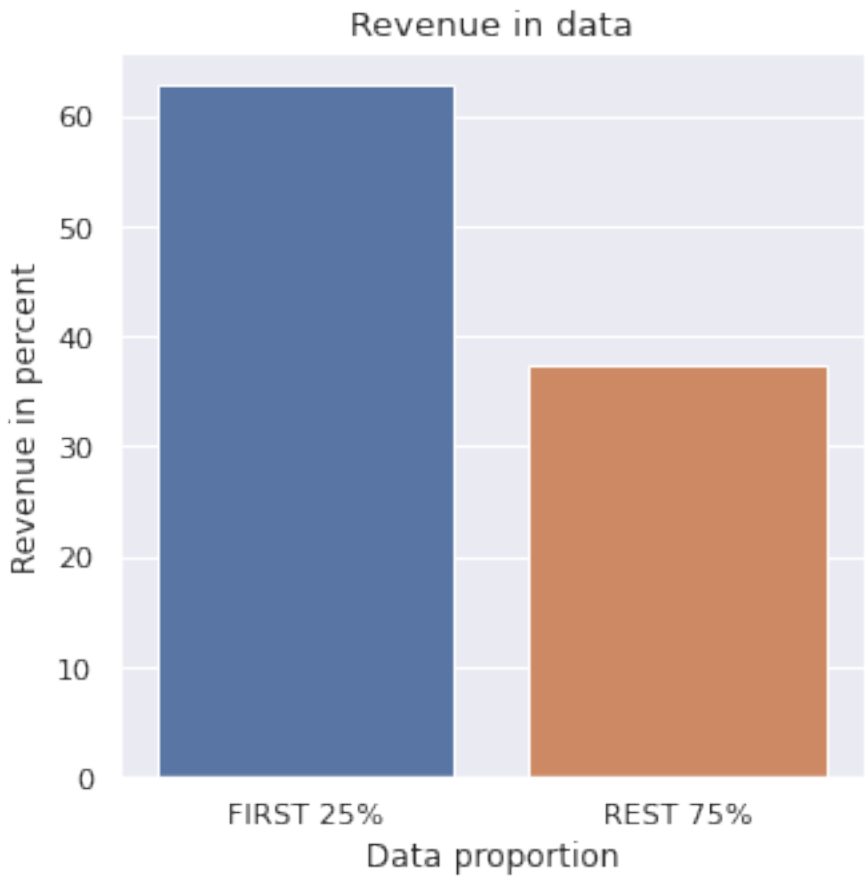
```
76        Joss Whedon  Robert Downey Jr., Chris Evans, Scarlett Johan…
54  Christopher Nolan  Christian Bale, Heath Ledger, Aaron Eckhart,Mi…

    Year  Runtime  Rating    Votes  RevenueMillions  Metascore
50  2015      136     8.0   661608            937.0       81.0
87  2009      162     8.0   935408            761.0       83.0
85  2015      124     7.0   455169            652.0       59.0
76  2012      143     8.0  1045588            623.0       69.0
54  2008      152     9.0  1791916            533.0       82.0
```

[12]:
```
df_films_25_per = df_IMDB_movies.head(250)
df_films_75_per = df_IMDB_movies.tail(750)
```

[13]:
```
total_revenue = df_IMDB_movies.RevenueMillions.sum()
revenue_25_percent = df_films_25_per.RevenueMillions.sum()
revenue_75_percent = df_films_75_per.RevenueMillions.sum()
```

[14]:
```
lst_revenue = [revenue_25_percent / total_revenue,
               revenue_75_percent / total_revenue]
lst_labels = ["FIRST 25%", "REST 75%"]
df_revenue_proportion = pd.DataFrame(lst_revenue,
                                     index = lst_labels,
                                     columns = ["revenue_per"])
```

[15]:
```
plt.figure(figsize = (5, 5))
sns.set_theme(style = "darkgrid")
fig_obj = sns.barplot(x = df_revenue_proportion.index,
                      y = df_revenue_proportion.revenue_per
                          * 100,
                      data = df_revenue_proportion)

fig_obj.set_title("Revenue in data",
                  fontsize = 13)
fig_obj.set_xlabel("Data proportion")
fig_obj.set_ylabel("Revenue in percent")
plt.show()
```



As observed the top first 25 percent of the data generates more than 60 percent of the income, this is an important fact because in order to characterize a new movie to maximize profit, the top 25 percent must be studied since the other 75 percent of the data is not relevant according to the main purpose of this analysis. Consequently, the first top 25 percent of the data has the most valuable information in regard to the profit and constitutes henceforth the analysed data:

1.1.1 Analysing per movie gender:

[16]:
```
df_films = df_films_25_per.copy()
df_films.reset_index(inplace = True, drop = True)
```

[17]:
```
def split_genre(str_movie_genre):
    split_genre = str_movie_genre.split(",")
    return split_genre

def genre_classifier(movie_genres, genre):
    if genre in movie_genres:
        return "YES"
```

```
    else:
        return "NO"
```

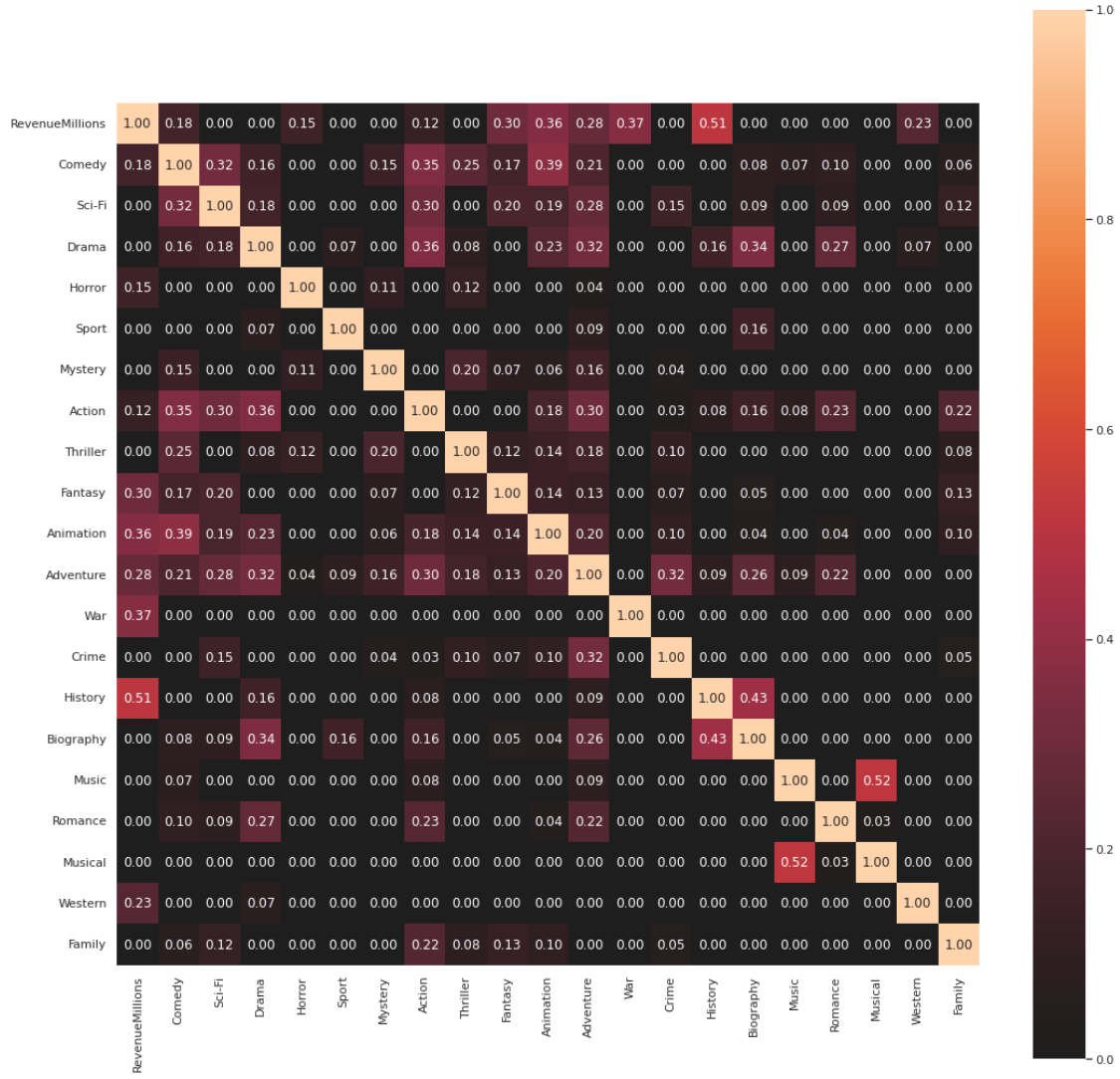
[18]:
```
lst_total_genres = []

for i in range(len(df_films.Genre)):
    genres_movie = split_genre(df_films.Genre[i])
    for j in range(len(genres_movie)):
        lst_total_genres.append(genres_movie[j])

unique_genres = set(lst_total_genres)
```

[19]:
```
df_films_genre = df_films.copy()
```

[20]:
```
for genre in unique_genres:
    df_films_genre[genre] = (df_films_genre.Genre.apply(lambda x:
                                        genre_classifier(x, genre)))
```

[21]:
```
df_films_genre.drop(["Id", "Title", "Genre",
                     "Director", "Actors", "Year",
                     "Runtime", "Rating", "Votes",
                     "Metascore"], axis=1, inplace = True)
```

[22]:
```
numer_col = 0
categ_col = list(range(1, 21))

associations(df_films_genre, nominal_columns = numer_col,
             numerical_columns = categ_col,
             nom_nom_assoc = "cramer", num_num_assoc = "pearson",
             nom_num_assoc = "correlation_ratio",
             figsize = (18, 18))
```



[22]: {'corr':

	RevenueMillions	Comedy	Sci-Fi	Drama	Horror
RevenueMillions	1.000000	0.178142	0.000000	0.000000	0.149335
Comedy	0.178142	1.000000	0.317459	0.158465	0.000000
Sci-Fi	0.000000	0.317459	1.000000	0.175874	0.000000
Drama	0.000000	0.158465	0.175874	1.000000	0.000000
Horror	0.149335	0.000000	0.000000	0.000000	1.000000
Sport	0.000000	0.000000	0.000000	0.074876	0.000000
Mystery	0.000000	0.147283	0.000000	0.000000	0.109131
Action	0.123574	0.354957	0.302281	0.357647	0.000000
Thriller	0.000000	0.249528	0.000000	0.081016	0.123367
Fantasy	0.300758	0.172184	0.199233	0.000000	0.000000
Animation	0.360328	0.388181	0.187722	0.230673	0.000000
Adventure	0.284027	0.208471	0.280660	0.319373	0.035065

```
War                  0.371614  0.000000  0.000000  0.000000  0.000000
Crime                0.000000  0.000000  0.154801  0.000000  0.000000
History              0.513405  0.000000  0.000000  0.156962  0.000000
Biography            0.000000  0.083749  0.092711  0.343752  0.000000
Music                0.000000  0.070779  0.000000  0.000000  0.000000
Romance              0.000000  0.100072  0.092711  0.266944  0.000000
Musical              0.000000  0.000000  0.000000  0.000000  0.000000
Western              0.231846  0.000000  0.000000  0.069570  0.000000
Family               0.000000  0.062921  0.121556  0.000000  0.000000

                      Sport   Mystery    Action  Thriller   Fantasy  … \
RevenueMillions  0.000000  0.000000  0.123574  0.000000  0.300758  …
Comedy           0.000000  0.147283  0.354957  0.249528  0.172184  …
Sci-Fi           0.000000  0.000000  0.302281  0.000000  0.199233  …
Drama            0.074876  0.000000  0.357647  0.081016  0.000000  …
Horror           0.000000  0.109131  0.000000  0.123367  0.000000  …
Sport            1.000000  0.000000  0.000000  0.000000  0.000000  …
Mystery          0.000000  1.000000  0.000000  0.195709  0.072758  …
Action           0.000000  0.000000  1.000000  0.000000  0.000000  …
Thriller         0.000000  0.195709  0.000000  1.000000  0.116820  …
Fantasy          0.000000  0.072758  0.000000  0.116820  1.000000  …
Animation        0.000000  0.064156  0.183922  0.141480  0.141480  …
Adventure        0.089468  0.157154  0.297137  0.179373  0.131978  …
War              0.000000  0.000000  0.000000  0.000000  0.000000  …
Crime            0.000000  0.044836  0.027332  0.104345  0.068849  …
History          0.000000  0.000000  0.075853  0.000000  0.000000  …
Biography        0.157130  0.000000  0.162961  0.000000  0.054490  …
Music            0.000000  0.000000  0.075853  0.000000  0.000000  …
Romance          0.000000  0.000000  0.233959  0.000000  0.000000  …
Musical          0.000000  0.000000  0.000000  0.000000  0.000000  …
Western          0.000000  0.000000  0.000000  0.000000  0.000000  …
Family           0.000000  0.000000  0.219776  0.083022  0.127436  …

                 Adventure       War     Crime   History  Biography     Music
\
RevenueMillions   0.284027  0.371614  0.000000  0.513405   0.000000  0.000000
Comedy            0.208471  0.000000  0.000000  0.000000   0.083749  0.070779
Sci-Fi            0.280660  0.000000  0.154801  0.000000   0.092711  0.000000
Drama             0.319373  0.000000  0.000000  0.156962   0.343752  0.000000
Horror            0.035065  0.000000  0.000000  0.000000   0.000000  0.000000
Sport             0.089468  0.000000  0.000000  0.000000   0.157130  0.000000
Mystery           0.157154  0.000000  0.044836  0.000000   0.000000  0.000000
Action            0.297137  0.000000  0.027332  0.075853   0.162961  0.075853
Thriller          0.179373  0.000000  0.104345  0.000000   0.000000  0.000000
Fantasy           0.131978  0.000000  0.068849  0.000000   0.054490  0.000000
Animation         0.203559  0.000000  0.104251  0.000000   0.044989  0.000000
Adventure         1.000000  0.000000  0.315780  0.089468   0.256288  0.089468
```

```
War                0.000000  1.000000  0.000000  0.000000   0.000000  0.000000
Crime              0.315780  0.000000  1.000000  0.000000   0.000000  0.000000
History            0.089468  0.000000  0.000000  1.000000   0.433862  0.000000
Biography          0.256288  0.000000  0.000000  0.433862   1.000000  0.000000
Music              0.089468  0.000000  0.000000  0.000000   0.000000  1.000000
Romance            0.221164  0.000000  0.000000  0.000000   0.000000  0.000000
Musical            0.000000  0.000000  0.000000  0.000000   0.000000  0.522538
Western            0.000000  0.000000  0.000000  0.000000   0.000000  0.000000
Family             0.000000  0.000000  0.047022  0.000000   0.000000  0.000000

                   Romance   Musical   Western    Family
RevenueMillions   0.000000  0.000000  0.231846  0.000000
Comedy            0.100072  0.000000  0.000000  0.062921
Sci-Fi            0.092711  0.000000  0.000000  0.121556
Drama             0.266944  0.000000  0.069570  0.000000
Horror            0.000000  0.000000  0.000000  0.000000
Sport             0.000000  0.000000  0.000000  0.000000
Mystery           0.000000  0.000000  0.000000  0.000000
Action            0.233959  0.000000  0.000000  0.219776
Thriller          0.000000  0.000000  0.000000  0.083022
Fantasy           0.000000  0.000000  0.000000  0.127436
Animation         0.044989  0.000000  0.000000  0.095830
Adventure         0.221164  0.000000  0.000000  0.000000
War               0.000000  0.000000  0.000000  0.000000
Crime             0.000000  0.000000  0.000000  0.047022
History           0.000000  0.000000  0.000000  0.000000
Biography         0.000000  0.000000  0.000000  0.000000
Music             0.000000  0.522538  0.000000  0.000000
Romance           1.000000  0.033918  0.000000  0.000000
Musical           0.033918  1.000000  0.000000  0.000000
Western           0.000000  0.000000  1.000000  0.000000
Family            0.000000  0.000000  0.000000  1.000000

[21 rows x 21 columns],
'ax': <AxesSubplot:>}
```

The previous chart's relation between the movie genres and the revenue, allow us to observe the correspondence between the genres and the revenue. "History", "War", "Adventure", "Animation", "Western", "Fantasy", "Comedy", "Horror" having a moderate correlation with the generated profits. This was carried out aiming to find a genre or genres strongly related with revenue.

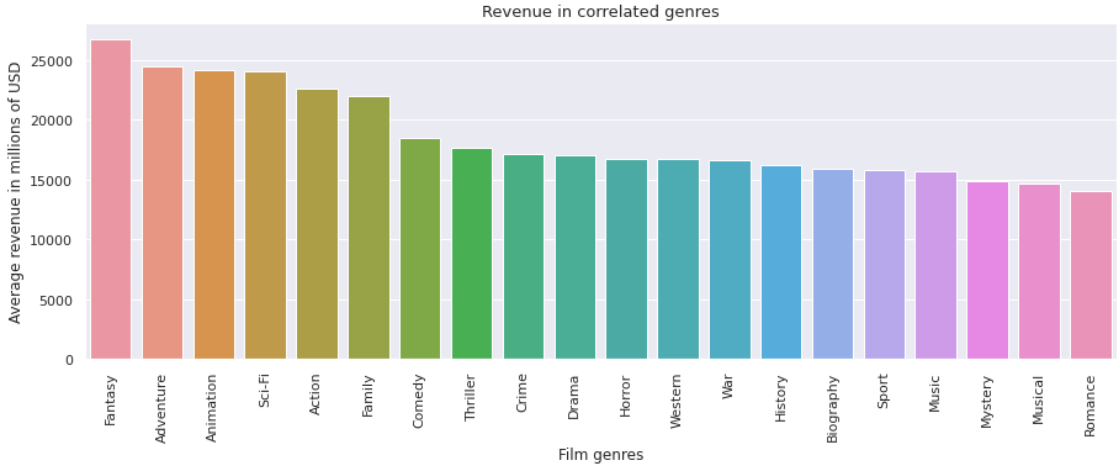
[23]:
```
df_films_genre.reset_index(inplace = True, drop = True)
```

[24]:
```
correlated_genres = ["History", "War", "Adventure",
                     "Animation", "Western", "Fantasy",
                     "Comedy", "Horror"]
lst_revenue = []
```

```
for genre in unique_genres:
    cor_movie_genre = (df_films_genre[
                        df_films_genre[genre] == "YES"])
    revenue_movie = cor_movie_genre.RevenueMillions.mean()
    lst_revenue.append(revenue_movie)

df_revenue = pd.DataFrame(
                lst_revenue, index = unique_genres,
                columns = ["Reveneu"])
```

[25]:
```
df_revenue.sort_values(by = ["Reveneu"],
                               ascending = False,
                               inplace = True)
```

[26]:
```
plt.figure(figsize = (15, 5))
sns.set_theme(style = "darkgrid")
fig_obj = sns.barplot(x = df_revenue.index,
                      y = df_revenue.Reveneu
                          * 100,
                      data = df_revenue)
fig_obj.set_title("Revenue in correlated genres",
                  fontsize = 13)
fig_obj.set_xlabel("Film genres")
fig_obj.set_ylabel("Average revenue "
                   "in millions of USD")
fig_obj.set_xticklabels(fig_obj.get_xticklabels(),
                        rotation = 90)
plt.show()
```



As observed, the relation between genre and revenue is not clear for this case ”History” had a

correlation of 0.51 but in the previous chart, its generated income is low.

Further, it is also required to study the behavior of the other categorical data in concerning "Director" and "Actors".

[27]:
```
unique_directors = (df_films.Director.unique())
```

[28]:
```
len(unique_directors)
```

[28]: 162

With respect to the directors, there are 162 directors in the top 25 percent (related to the highes revenue) of the data. An attempt to analyze directors and revenue was developed, but there was not computational capacity and the dython library did not converge.

Studying the actors within the top 25 percent:

[29]:
```
def split_actors(str_movie_actor):
    split_actor = str_movie_actor.split(",")
    return split_actor

lst_total_actors = []
for i in range(len(df_films.Actors)):
    actors_movie = split_actors(df_films.Actors[i])
    for j in range(len(actors_movie)):
        lst_total_actors.append(actors_movie[j])

unique_actors = set(lst_total_actors)
```

[30]:
```
len(unique_actors)
```

[30]: 679

There are 679 actors, there is no computational capacity to analyse their relation with the revenue from the Dython library.

1.1.2 Analysing numeric features:

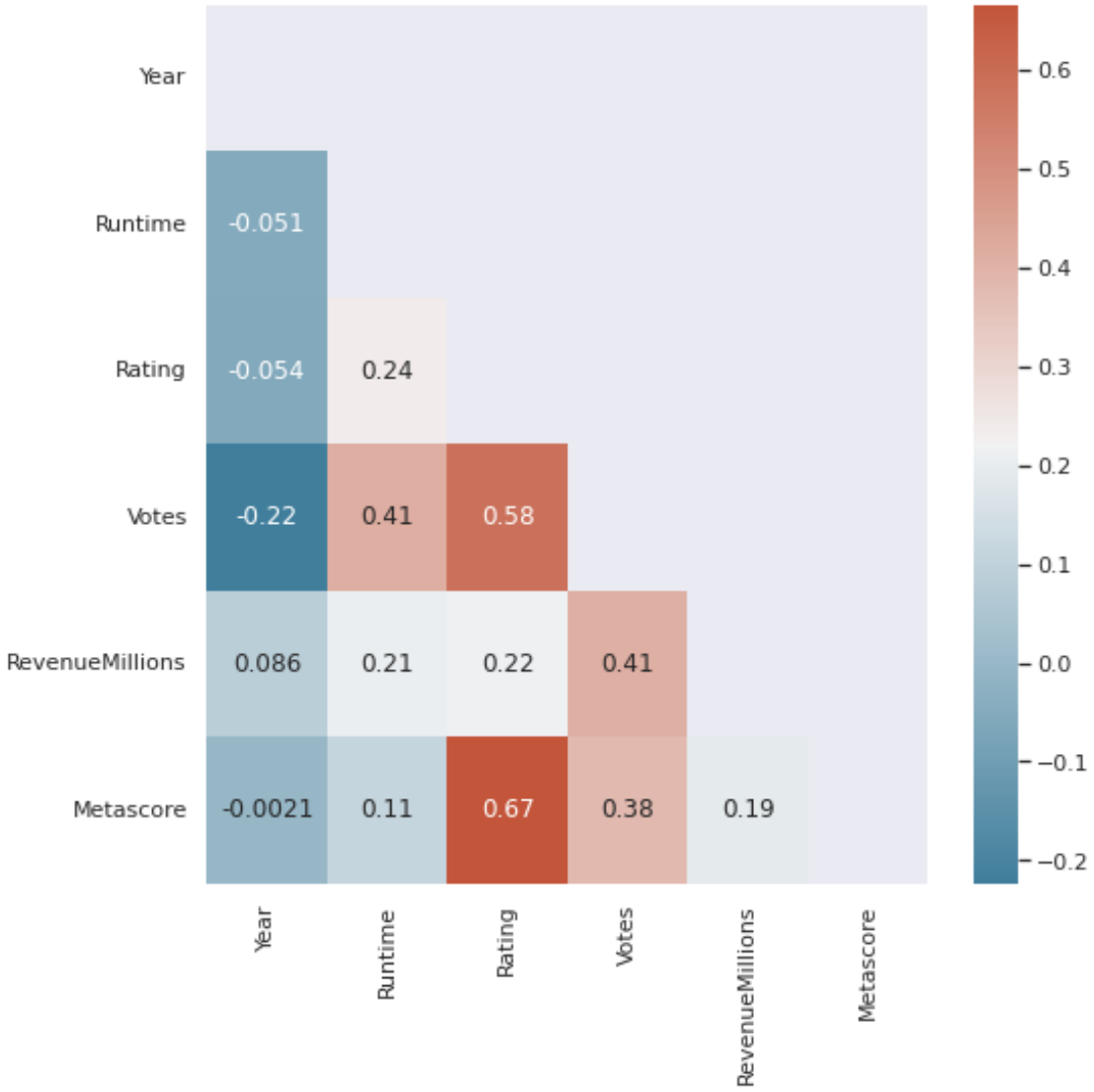
[31]:
```
df_films = df_films_25_per.copy()
df_films.reset_index(inplace = True, drop = True)
```

Developing a Pearson correlation chart:

[32]:
```
df_films_continous = df_films.copy()
df_films_continous.drop(["Id", "Title", "Genre",
                         "Director", "Actors"],
                         axis=1, inplace = True)
```

[33]:
```
f, ax = plt.subplots(figsize=(8, 8))
mask = np.triu(np.ones_like(df_films_continous.corr(),
                            dtype=bool))
cmap = sns.diverging_palette(230, 20, as_cmap = True)
sns.heatmap(df_films_continous.corr(), annot = True,
            mask = mask, cmap = cmap)
```

[33]: <AxesSubplot:>



With respect to the previous Pearson correlation in regard to the numerical variables, the renevue has a negligible correlation with respect to the film's year, and moderate degree of correlation with the number of votes. Low correlation with with the runtime and rating. However, Rating, Votes and Metascore had a large Correlation. Now it is necessary to explore the behavior of the numer of votes and the revenue, because it has the strongenst relation with the income feature (with respect to the analysed variables). With this purpose in mind, the data will be classied in three levels

according to the number of votes.

[34]:
```
df_film_votes = df_films.copy()
```

Determining the number of bins:

[35]:
```
bin_size = ((df_film_votes.Votes.max()
             - df_film_votes.Votes.min())
             / 4)
```

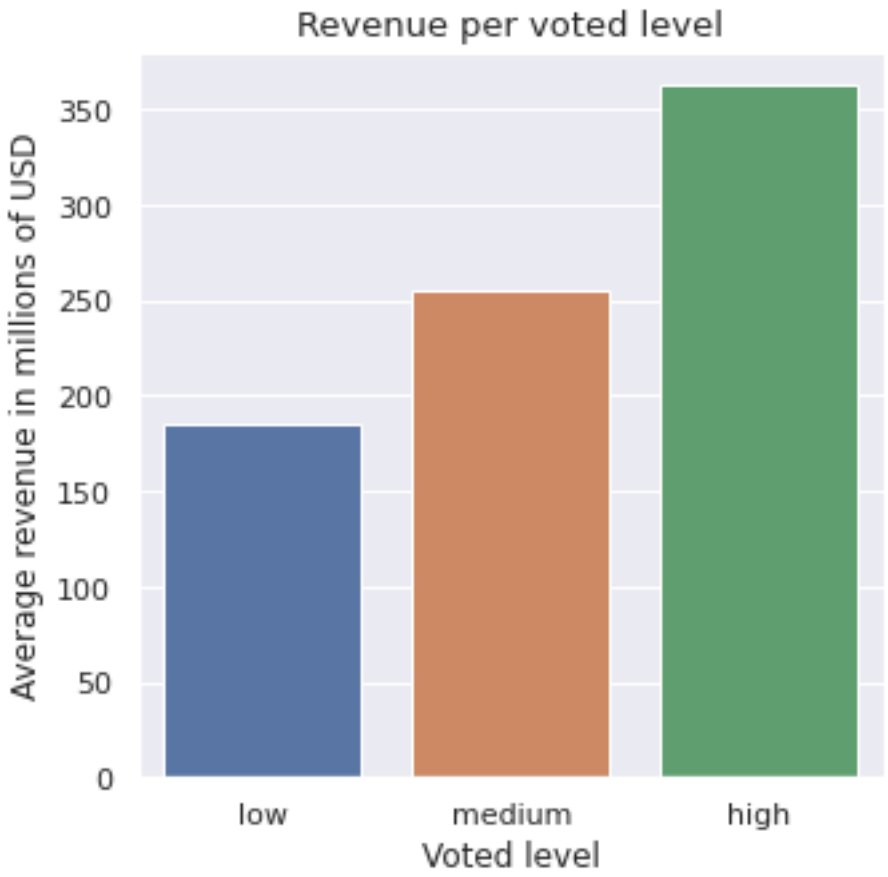
[36]:
```
bin_1 = int(df_film_votes.Votes.min())
bin_2 = int(bin_1 + bin_size)
bin_3 = int(bin_2 + bin_size)
bin_4 = int(df_film_votes.Votes.max())

bins = [bin_1, bin_2, bin_3, bin_4]
labels = ["low", "medium", "high"]
```

Classifying the data into three voted levels (low, medium, high):

[37]:
```
df_film_votes["Vote_level"] = (pd.cut(df_film_votes["Votes"],
                                bins = bins, labels = labels))
```

[38]:
```
group_vote = df_film_votes[["Vote_level",
                            "RevenueMillions"]].copy()
group_vote =  (group_vote.groupby("Vote_level",
               as_index = True).agg({"RevenueMillions":
                                     "mean"}))
```

[39]:
```
plt.figure(figsize = (5, 5))
sns.set_theme(style = "darkgrid")
fig_obj = sns.barplot(x = group_vote.index,
                      y = group_vote.RevenueMillions,
                      data = group_vote)

fig_obj.set_title("Revenue per voted level",
                              fontsize = 13)
fig_obj.set_xlabel("Voted level")
fig_obj.set_ylabel("Average revenue "
                   "in millions of USD")

plt.show()
```



The previous plot confirms the correlation between the votes and the revenue. Consequently, the study of the high voted films will give insights of the characteristics of a product that maximizes profit.

1.1.3 Analysing high voted films:

[40]:
```
df_high_voted = (df_film_votes[(
                 df_film_votes.Vote_level == 'high')])

df_high_voted.reset_index(inplace = True, drop = True)
```

[41]:
```
df_high_voted
```

[41]:

	Id	Title	Genre	Director	\
0	88	Avatar	Action,Adventure,Fantasy	James Cameron	
1	77	The Avengers	Action,Sci-Fi	Joss Whedon	
2	55	The Dark Knight	Action,Crime,Drama	Christopher Nolan	
3	125	The Dark Knight Rises	Action,Thriller	Christopher Nolan	

```
4   81              Inception   Action,Adventure,Sci-Fi  Christopher Nolan
5   37           Interstellar    Adventure,Drama,Sci-Fi  Christopher Nolan
6  145       Django Unchained             Drama,Western  Quentin Tarantino
7  100           The Departed      Crime,Drama,Thriller    Martin Scorsese
8   78   Inglourious Basterds       Adventure,Drama,War  Quentin Tarantino

                                              Actors  Year  Runtime  Rating  \
0  Sam Worthington, Zoe Saldana, Sigourney Weaver…  2009      162     8.0
1  Robert Downey Jr., Chris Evans, Scarlett Johan…  2012      143     8.0
2  Christian Bale, Heath Ledger, Aaron Eckhart,Mi…  2008      152     9.0
3  Christian Bale, Tom Hardy, Anne Hathaway,Gary …  2012      164     9.0
4  Leonardo DiCaprio, Joseph Gordon-Levitt, Ellen…  2010      148     9.0
5  Matthew McConaughey, Anne Hathaway, Jessica Ch…  2014      169     9.0
6  Jamie Foxx, Christoph Waltz, Leonardo DiCaprio…  2012      165     8.0
7  Leonardo DiCaprio, Matt Damon, Jack Nicholson,…  2006      151     9.0
8  Brad Pitt, Diane Kruger, Eli Roth,Mélanie Laurent  2009      153     8.0

     Votes  RevenueMillions  Metascore Vote_level
0   935408            761.0       83.0       high
1  1045588            623.0       69.0       high
2  1791916            533.0       82.0       high
3  1222645            448.0       78.0       high
4  1583625            293.0       74.0       high
5  1047747            188.0       74.0       high
6  1039115            163.0       81.0       high
7   937414            132.0       85.0       high
8   959065            121.0       69.0       high
```

Moreover, With respect to the genre in this group. Genres such as ”action”, ”Adventure” and ”Fantansy” are located and those also have the highest profit in the studied databaseit. The films runtime ranges from 143 to 169 min. Further, it is also observed that directors like James Cameron, Joss Whedon, Christopher Nolan, Quentin Tarantino and Martin Scorserse appear. Christopher Nolan being four times. It is desirable to analyse If they are have a relation with the highest income.

[42]:
```
def director_classifier(row, director):
    if director in row:
        return "YES"
    else:
        return "NO"
```

[43]:
```
df_directors = df_films.copy()
```

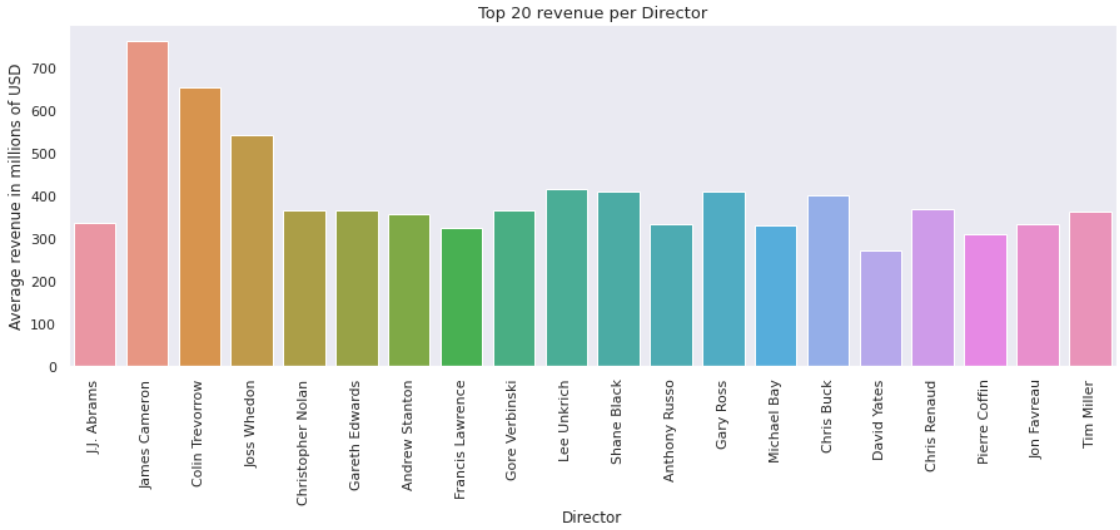
[44]:
```
for director in unique_directors:
    df_directors[director] = (df_directors.Director.apply(lambda x:
                              director_classifier(x, director)))
```

[45]:
```
df_directors.drop(["Id", "Title", "Genre",
                   "Director", "Actors", "Year",
                   "Runtime", "Rating", "Votes",
                   "Metascore"], axis=1, inplace = True)
```

[46]:
```
lst_rev_director = []

for director in unique_directors:
    cor_movie_direc = (df_directors[
                       df_directors[director] == "YES"])
    revenue_movie = cor_movie_direc.RevenueMillions.mean()
    lst_rev_director.append(revenue_movie)

df_rev_director = pd.DataFrame(
                  lst_rev_director,
                  index = unique_directors,
                  columns = ["Revenue"])
```

[47]:
```
directors_high_revenue = df_rev_director.head(20)
```

[48]:
```
plt.figure(figsize = (15, 5))
sns.set_theme(style = "dark")
fig_obj = sns.barplot(x = directors_high_revenue.index,
                      y = (directors_high_revenue.Revenue),
                      data = group_vote)

fig_obj.set_title("Top 20 revenue per Director",
                                  fontsize = 13)
fig_obj.set_xlabel("Director")
fig_obj.set_ylabel("Average revenue "
                   "in millions of USD")
fig_obj.set_xticklabels(fig_obj.get_xticklabels(),
                        rotation = 90)

plt.show()
```



The previous chart displays the first 20 directors with the highest profit. It is observed that James Cameron, Joss Whedon and Cristopher Nolan appear. Consequently, these directors increase the revenue regading the relations actor-revenue and votes-revenue.

Approaching the characteristics of the most voted segmentation in regard to the actors:

[49]:
```
lst_actors_voted = []

for i in range(len(df_high_voted.Actors)):
    actors_movie = split_actors(df_high_voted.Actors[i])
    for j in range(len(actors_movie)):
        lst_actors_voted.append(actors_movie[j])
```

[50]:
```
fequency_voted_actors = {i:lst_actors_voted.count(i)
                         for i in lst_actors_voted}
```

[51]:
```
df_voted_actors = pd.DataFrame.from_dict(fequency_voted_actors,
                                         orient = "index",
                                         columns = ["Frequency"])
df_voted_actors.sort_values(by = ["Frequency"],
                            ascending = False,
                            inplace = True)
```

[52]:
```
df_voted_actors.head()
```

[52]:

	Frequency
Leonardo DiCaprio	2
Anne Hathaway	2
Christian Bale	2
Sam Worthington	1

```
Leonardo DiCaprio          1
```

Famous actors such as Leonardo DiCaprio, Anne Hathaway and Christian Bale are present in the highest voted film associated to the largest income. Consequently, the list of actors who could be hired to produce a movie with a large profit are listed in the "df_voted_actors" dataframe.

1.1.4 Final remarks:

With the previous analysis, the insights that suggest a maximum film's revenue are:

- The director should be James Cameron, Joss Whedon or Cristopher Nolan (James Cameron principally).
- The movie should be based on action, adventure, Scify and fantasy since these genres had the highest profit in the studied database.
- Even the run time has a low correlation (but not negligible) with revenue of 0.21 according to the Pearson chart, it is desirable it ranges within 143 to 169 min.
- In regard to the actors it was impossible to analyse a correlation with the revenue due to the lack of computational capacity. Several attempts to analyse director and actors likewise genre were carried out but the Dython library did not converge. However, famous actors like Anne Hathaway, Christian Bale and Leonardo DiCrapio appear within the segmentation with the highest profit and highly vote. However, the list of such a segmentation's actors are available.