МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования

Национальный исследовательский ядерный университет «МИФИ»

ИНСТИТУТ ИНТЕЛЛЕКТУАЛЬНЫХ КИБЕРНЕТИЧЕСКИХ СИСТЕМ

Кафедра №75 «Финансовый мониторинг»

ОТЧЕТ

по лабораторной работе №2

«Изучение основ работы по преобразованию данных и их хранению в БД средствами языка Python»

вариант 3

по курсу

«Информационные ресурсы в финансовом мониторинге»

Выполнили: Васильев Алексей,
Гусак Павел,
Макаров Максим,
Михненко Ольга,
студенты группы С15-501,
Проверила: Давыденко В.И.

Москва, 2019 г

Постановка задачи

1. Скачать с сайта набор со статистическими данными, посвященный каталогу фильмов IMDB.
2. Написать программу на языке Python, выполняющую следующие действия:
    a. Создает в СУБД SQLiteили Oraclenабор таблиц для хранения данных рассматриваемого набора. При этом исключите из исходного набора данных все переменные кроме director_name, budget, imdb_score, title_year.
    b. Загружает данные из набора в созданные таблицы.
    c. Выполняет очистку данных, удаляя все строки с пустыми или нереальными данными.
    d. Выполняет частотный анализ по переменной director_name.
    e. Удаляет из загруженных данных все строки, содержащие директоров с экстремальным числом фильмов. Экстремальным числом будем считать такую величину x, которая стоит в первых и последних 10% по порядку наблюдения. Причем есть такая величина y (может быть равна x), стоящая ближе к центру, чем x и разрыв между которой и соседней с ней по порядку следования к центру составляет более 10% от общего разброса. Также должны быть удалены директора с числом фильмов, меньшим 3.
    f. Строит график зависимости imdb_score от бюджета фильма.
    g. На основе полученных данных проведите простейшее исследование наличия факта корреляции между бюджетом фильма и его рейтингом.

Выполнение работы

1. ***Скачать с сайта набор со статистическими данными, посвященный каталогу фильмов IMDB.***

   Чтобы **корректно** открыть csv – файл в Excel, необходимо перейти во вкладку Данные и сделать импорт данных из текста, указав соответствующий файл movie_metadata.csv и запятую в качестве разделителя.

color	director_name	num_critic_for_reviews	duration	director_facebook_likes	actor_3_facebook_likes	actor_2_name	actor_1_facebook_lik
Color	James Cameron	723	178	0	855	Joel David Moore	1
Color	Gore Verbinski	302	169	563	1000	Orlando Bloom	40
Color	Sam Mendes	602	148	0	161	Rory Kinnear	11
Color	Christopher Nolan	813	164	22000	23000	Christian Bale	27
	Doug Walker			131		Rob Walker	
Color	Andrew Stanton	462	132	475	530	Samantha Morton	
Color	Sam Raimi	392	156	0	4000	James Franco	24
Color	Nathan Greno	324	100	15	284	Donna Murphy	
Color	Joss Whedon	635	141	0	19000	Robert Downey Jr.	26
Color	David Yates	375	153	282	10000	Daniel Radcliffe	25
Color	Zack Snyder	673	183	0	2000	Lauren Cohan	15
Color	Bryan Singer	434	169	0	903	Marlon Brando	18
Color	Marc Forster	403	106	395	393	Mathieu Amalric	
Color	Gore Verbinski	313	151	563	1000	Orlando Bloom	40
Color	Gore Verbinski	450	150	563	1000	Ruth Wilson	40

Рис.1 “movie_metadata.csv”

Правильно скачанный файл movie_metadata.csv в Excel выглядит так (рис.1)

2. ***Написать программу на языке Python, выполняющую следующие действия:***
   a. ***Создает в СУБД SQLite или Oracleнабор таблиц для хранения данных рассматриваемого набора. При этом исключите из исходного набора данных все переменные кроме director_name, budget, imdb_score, title_year.***

   Подключаем библиотеку pandas и выводим содержимое файла movie_metadata.csv

```
In [74]: import pandas as pd
         data = pd.read_csv ("movie_metadata.csv")
         data
```

Out[74]:

	color	director_name	num_critic_for_reviews	duration	director_facebook_likes	actor_3_facebook_likes	actor_2_name	actor_1_facebook_likes	gr
0	Color	James Cameron	723.0	178.0	0.0	855.0	Joel David Moore	1000.0	7605058
1	Color	Gore Verbinski	302.0	169.0	563.0	1000.0	Orlando Bloom	40000.0	3094041
2	Color	Sam Mendes	602.0	148.0	0.0	161.0	Rory Kinnear	11000.0	2000741
3	Color	Christopher Nolan	813.0	164.0	22000.0	23000.0	Christian Bale	27000.0	4481306
4	NaN	Doug Walker	NaN	NaN	131.0	NaN	Rob Walker	131.0	
5	Color	Andrew Stanton	462.0	132.0	475.0	530.0	Samantha Morton	640.0	730586
6	Color	Sam Raimi	392.0	156.0	0.0	4000.0	James Franco	24000.0	3365303
7	Color	Nathan Greno	324.0	100.0	15.0	284.0	Donna Murphy	799.0	2008072
8	Color	Joss Whedon	635.0	141.0	0.0	19000.0	Robert Downey Jr.	26000.0	4589915

Рис.2 – Загрузка файла в Python

Оставляем в файле только нужные переменные.

```
In [75]: new_data = data [["director_name", "budget", "imdb_score", "title_year"]]
         new_data
```

Out[75]:

	director_name	budget	imdb_score	title_year
0	James Cameron	237000000.0	7.9	2009.0
1	Gore Verbinski	300000000.0	7.1	2007.0
2	Sam Mendes	245000000.0	6.8	2015.0
3	Christopher Nolan	250000000.0	8.5	2012.0
4	Doug Walker	NaN	7.1	NaN
5	Andrew Stanton	263700000.0	6.6	2012.0
6	Sam Raimi	258000000.0	6.2	2007.0
7	Nathan Greno	260000000.0	7.8	2010.0
8	Joss Whedon	250000000.0	7.5	2015.0
9	David Yates	250000000.0	7.5	2009.0
10	Zack Snyder	250000000.0	6.9	2016.0
11	Bryan Singer	209000000.0	6.1	2006.0

Рис.3 – Усеченный датафрейм

b. Загружает данные из набора в созданные таблицы.

```
import sqlite3
conn = sqlite3.connect("lab2.db") # открывает соединение
cursor = conn.cursor()
```

```
new_data.to_sql ("movies", conn, if_exists="replace")
```

```
cursor.execute ('select * from movies')
cur = cursor.fetchall()
print (cur)
df_2ab = pd.DataFrame(data = cur, columns =['id','director_name','budget','imdb_score','title_year'])
df_2ab
```

```
[(0, 'James Cameron', 237000000.0, 7.9, 2009.0), (1, 'Gore Verbinski', 300000000.0, 7.1, 2007.0), (2, 'Sam Mendes', 24500000.0, 6.8, 2015.0), (3, 'Christopher Nolan', 250000000.0, 8.5, 2012.0), (4, 'Doug Walker', None, 7.1, None), (5, 'Andrew Stanton', 263700000.0, 6.6, 2012.0), (6, 'Sam Raimi', 258000000.0, 6.2, 2007.0), (7, 'Nathan Greno', 260000000.0, 7.8, 2010.0), (8, 'Joss Whedon', 250000000.0, 7.5, 2015.0), (9, 'David Yates', 250000000.0, 7.5, 2009.0), (10, 'Zack Snyder', 250000000.0, 6.9, 2016.0), (11, 'Bryan Singer', 209000000.0, 6.1, 2006.0), (12, 'Marc Forster', 200000000.0, 6.7, 2008.0), (13, 'Gore Verbinski', 225000000.0, 7.3, 2006.0), (14, 'Gore Verbinski', 215000000.0, 6.5, 2013.0), (15, 'Zack Snyder', 225000000.0, 7.2, 2013.0), (16, 'Andrew Adamson', 225000000.0, 6.6, 2008.0), (17, 'Joss Whedon', 220000000.0, 8.1, 2012.0), (18, 'Rob Marshall', 250000000.0, 6.7, 2011.0), (19, 'Barry Sonnenfeld', 225000000.0, 6.8, 2012.0), (20, 'Peter Jackson', 250000000.0, 7.5, 2014.
```

Рис.4 – Создание соединения и выполнение запроса

Cur – представление ответа на запрос, представлен в виде списка из кортежей. Данное представление не очень удобно для восприятия, поэтому будем в дальнейшем обертывать его в датафрейм (последние 2 строки кода):

	id	director_name	budget	imdb_score	title_year
0	0	James Cameron	237000000.0	7.9	2009.0
1	1	Gore Verbinski	300000000.0	7.1	2007.0
2	2	Sam Mendes	245000000.0	6.8	2015.0
3	3	Christopher Nolan	250000000.0	8.5	2012.0
4	5	Andrew Stanton	263700000.0	6.6	2012.0
5	6	Sam Raimi	258000000.0	6.2	2007.0
6	7	Nathan Greno	260000000.0	7.8	2010.0
7	8	Joss Whedon	250000000.0	7.5	2015.0
8	9	David Yates	250000000.0	7.5	2009.0
9	10	Zack Snyder	250000000.0	6.9	2016.0
10	11	Bryan Singer	209000000.0	6.1	2006.0
11	12	Marc Forster	200000000.0	6.7	2008.0
12	13	Gore Verbinski	225000000.0	7.3	2006.0
13	14	Gore Verbinski	215000000.0	6.5	2013.0
14	15	Zack Snyder	225000000.0	7.2	2013.0
15	16	Andrew Adamson	225000000.0	6.6	2008.0
16	17	Joss Whedon	220000000.0	8.1	2012.0

Рис.5 – Обертка в датафрейм

c. *Выполняет очистку данных, удаляя все строки с пустыми или нереальными данными.*

Сначала удалим строки с пустыми значениями:

```
cursor.execute ('delete from movies where director_name is NULL or budget is NULL or imdb_score is NULL or title_year is NULL')
conn.commit()
cursor.execute ('select * from movies')
cur = cursor.fetchall()
df_2c1 = pd.DataFrame(data = cur, columns =['id','director_name','budget','imdb_score','title_year'])
df_2c1
```

Рис.6 – Удаление строк с пустыми значениями

Получаем:

	id	director_name	budget	imdb_score	title_year
0	0	James Cameron	237000000.0	7.9	2009.0
1	1	Gore Verbinski	300000000.0	7.1	2007.0
2	2	Sam Mendes	245000000.0	6.8	2015.0
3	3	Christopher Nolan	250000000.0	8.5	2012.0
4	5	Andrew Stanton	263700000.0	6.6	2012.0
5	6	Sam Raimi	258000000.0	6.2	2007.0
6	7	Nathan Greno	260000000.0	7.8	2010.0
7	8	Joss Whedon	250000000.0	7.5	2015.0
8	9	David Yates	250000000.0	7.5	2009.0
9	10	Zack Snyder	250000000.0	6.9	2016.0
10	11	Bryan Singer	209000000.0	6.1	2006.0
11	12	Marc Forster	200000000.0	6.7	2008.0
12	13	Gore Verbinski	225000000.0	7.3	2006.0
13	14	Gore Verbinski	215000000.0	6.5	2013.0
14	15	Zack Snyder	225000000.0	7.2	2013.0
15	16	Andrew Adamson	225000000.0	6.6	2008.0
16	17	Joss Whedon	220000000.0	8.1	2012.0
17	18	Rob Marshall	250000000.0	6.7	2011.0

4536	5033	Shane Carruth	7000.0	7.0	2004.0
4537	5034	Neill Dela Llana	7000.0	6.3	2005.0
4538	5035	Robert Rodriguez	7000.0	6.9	1992.0
4539	5036	Anthony Vallone	3250.0	7.8	2005.0
4540	5037	Edward Burns	9000.0	6.4	2011.0
4541	5040	Benjamin Roberds	1400.0	6.3	2013.0
4542	5042	Jon Gunn	1100.0	6.6	2004.0

4543 rows × 5 columns

Рисунок 7 –Результат удаления

Видно, что примерно 500 строк удалились по условию: «Пустое значение».

Теперь удалим строки с нереальными данными.Для этого проведем примитивный анализ для числовых данных (считаем, что имена режиссеров корректные):

```
1 new_data.describe()
```

	budget	imdb_score	title_year
count	4.551000e+03	5043.000000	4935.000000
mean	3.975262e+07	6.442138	2002.470517
std	2.061149e+08	1.125116	12.474599
min	2.180000e+02	1.600000	1916.000000
25%	6.000000e+06	5.800000	1999.000000
50%	2.000000e+07	6.600000	2005.000000
75%	4.500000e+07	7.200000	2011.000000
max	1.221550e+10	9.500000	2016.000000

Рисунок 8 –Статистические данные

Рейтинг и год корректны, а бюджет в 12 млрд. выглядит нереалистично (при этом фильм с бюджетом в 218 $ реально есть, проверено Google), поэтому выведем самые дорогие фильмы:

```
new_data.dropna().sort_values( by = 'budget',ascending = False)
```

	director_name	budget	imdb_score	title_year
2988	Joon-ho Bong	1.221550e+10	7.0	2006.0
3859	Chan-wook Park	4.200000e+09	7.7	2005.0
3005	Lajos Koltai	2.500000e+09	7.1	2005.0
2323	Hayao Miyazaki	2.400000e+09	8.4	1997.0
2334	Katsuhiro Ôtomo	2.127520e+09	6.9	2004.0
3423	Katsuhiro Ôtomo	1.100000e+09	8.1	1988.0
4542	Takao Okawara	1.000000e+09	6.0	1999.0
3851	Carlos Saura	7.000000e+08	7.2	1998.0
3075	Karan Johar	7.000000e+08	6.0	2006.0
3273	Anurag Basu	6.000000e+08	6.0	2010.0
1338	John Woo	5.536320e+08	7.4	2008.0
3311	Chatrichalerm Yukol	4.000000e+08	6.6	2001.0
1016	Luc Besson	3.900000e+08	6.4	1999.0
2740	Tony Jaa	3.000000e+08	6.2	2008.0
1	Gore Verbinski	3.000000e+08	7.1	2007.0
5	Andrew Stanton	2.637000e+08	6.6	2012.0
7	Nathan Greno	2.600000e+08	7.8	2010.0

Рисунок 9 – Самые дорогие фильмы

По-видимому, всё, что до строчки с индексом 1 (судя по данным, это «Пираты карибского моря. На краю света»),либо неверно посчитано, либо посчитано в другой валюте (какой-то азиатской, судя по режиссерам), поэтому удаляем все фильмы с бюджетом больше 300 млн.

```
cursor.execute ('delete from movies where budget > 300000000')
conn.commit()
cursor.execute ('select * from movies')
cur = cursor.fetchall()
df_2c = pd.DataFrame(data = cur, columns =['id','director_name','budget','imdb_score','title_year'])
df_2c
```

	id	director_name	budget	imdb_score	title_year
0	0	James Cameron	237000000.0	7.9	2009.0
1	1	Gore Verbinski	300000000.0	7.1	2007.0
2	2	Sam Mendes	245000000.0	6.8	2015.0
3	3	Christopher Nolan	250000000.0	8.5	2012.0
4	5	Andrew Stanton	263700000.0	6.6	2012.0
5	6	Sam Raimi	258000000.0	6.2	2007.0
6	7	Nathan Greno	260000000.0	7.8	2010.0
7	8	Joss Whedon	250000000.0	7.5	2015.0
8	9	David Yates	250000000.0	7.5	2009.0

...

4523	5033	Shane Carruth	7000.0	7.0	2004.0
4524	5034	Neill Dela Llana	7000.0	6.3	2005.0
4525	5035	Robert Rodriguez	7000.0	6.9	1992.0
4526	5036	Anthony Vallone	3250.0	7.8	2005.0
4527	5037	Edward Burns	9000.0	6.4	2011.0
4528	5040	Benjamin Roberds	1400.0	6.3	2013.0
4529	5042	Jon Gunn	1100.0	6.6	2004.0

4530 rows × 5 columns

Рисунок 10 – Окончательный результат задания 2c

Удалилось еще 13 строчек по этому условию.

d. Выполняет частотный анализ по переменной director_name.

В данном задании просто нужно написать грамотный SQL-запрос

```
In [87]: cursor.execute ('SELECT director_name,count(*)  FROM movies group by director_name')
         cur = cursor.fetchall()
         df_2d = pd.DataFrame(data = cur, columns =['director_name','count of movies']).sort_values (by = 'count of movies')
         df_2d
```

Out[87]:

	director_name	count of movies
0	A. Raven Cruz	1
1272	Marc Abraham	1
1271	Mamoru Hosoda	1
1270	Malcolm Goodwin	1
1268	Maksim Fadeev	1
1267	Majid Majidi	1
1266	Maggie Greenwald	1
1265	Maggie Carey	1
1264	Mabrouk El Mechri	1
1263	Mabel Cheung	1
1260	Lynne Ramsay	1
1259	Lynn Shelton	1
1257	Luke Dye	1
1256	Luis Valdez	1
...		
1823	Sam Raimi	13
1756	Robert Zemeckis	13
1748	Robert Rodriguez	13
978	John Carpenter	13
1780	Ron Howard	13
1365	Michael Bay	13
148	Barry Levinson	13
1528	Oliver Stone	14
1670	Renny Harlin	15
1887	Spike Lee	16
1941	Steven Soderbergh	16
1999	Tim Burton	16
1707	Ridley Scott	17
1323	Martin Scorsese	18
347	Clint Eastwood	20
2148	Woody Allen	22
1942	Steven Spielberg	26

2168 rows × 2 columns

Рисунок 11 – Результаты частотного анализа

e. Удаляет из загруженных данных все строки, содержащие директоров с экстремальным числом фильмов. Экстремальным числом будем считать такую величину x, которая стоит в первых и последних 10% по порядку

наблюдения. Причем есть такая величина y (может быть равна x), стоящая ближе к центру, чем x и разрыв между которой и соседней с ней по порядку следования к центру составляет более 10% от общего разброса. Также должны быть удалены директора с числом фильмов, меньшим 3.

```
cursor.execute ('SELECT director_name,count(*)  FROM movies group by director_name order by count(*)') # как прошлое
cur = cursor.fetchall() # это список из кортежей, с которым мы будем дальше работать
```

```
razbros = cur [len(cur)-1][1]- cur[0][1] # разброс по фильмам (26-1)
```

```
n = len (cur) # по сути число режиссеров
```

```
left_border_default = int(0.1*n)
right_border_default = int(0.9*n) # тут я запомнил индексы, которые ограничивают 10 процентов с двух сторон
```

```
left_border_real = 0
for i in range (n//2,0,-1):
    if cur[i][1] - cur[i-1][1] > 0.1*razbros:
        left_border_real = i  # тут я запоминаю индекс, который будет соответствовать провалу > 0.1*разброс
        break

left_border = min (left_border_default,left_border_real) # окончательная граница, по которой мы обрежем список
```

```
right_border_real = n-1 # это зеркальная штука для правого конца
for i in range (n//2,n-1):
    if abs(cur[i][1] - cur[i+1][1]) > 0.1*razbros:
        right_border_real = i
        break

right_border = max (right_border_default,right_border_real)
```

Рисунок 12 – Выполнение пункта 2e

```
lst_cleared_directors = []
lst_cleared = cur[left_border:right_border+1] # обрезка изначального списка по найденным границам (границы ВКЛЮЧАЮТСЯ)
for pair in lst_cleared:
    lst_cleared_directors.append(pair[0]) # в этот список добавляем только режиссеров
```

```
lst_more2 = df_2d['director_name'] [df_2d['count of movies'] > 2] # список режиссеров с числом фильмов > 2
```

```
tup_cleared_directors = tuple (lst_cleared_directors)
tup_more2 = tuple (lst_more2) # надо переделать в кортежи, потому что sql понимает только круглые скобки
cursor.execute ('SELECT * FROM movies where director_name in {} and director_name in {}'.format(tup_cleared_directors,tup_mo
cur = cursor.fetchall()
df_2e = pd.DataFrame(data = cur, columns =['id','director_name','budget','imdb_score','title_year'])
df_2e
```

	id	director_name	budget	imdb_score	title_year
0	0	James Cameron	237000000.0	7.9	2009.0
1	1	Gore Verbinski	300000000.0	7.1	2007.0
2	2	Sam Mendes	245000000.0	6.8	2015.0
3	3	Christopher Nolan	250000000.0	8.5	2012.0
4	5	Andrew Stanton	263700000.0	6.6	2012.0
5	6	Sam Raimi	258000000.0	6.2	2007.0
6	8	Joss Whedon	250000000.0	7.5	2015.0
7	9	David Yates	250000000.0	7.5	2009.0

...

2496	5015	Richard Linklater	23000.0	7.1	1991.0
2497	5018	Alex Kendrick	20000.0	6.9	2003.0
2498	5021	Jay Duplass	15000.0	6.6	2005.0
2499	5025	John Waters	10000.0	6.1	1972.0
2500	5035	Robert Rodriguez	7000.0	6.9	1992.0
2501	5037	Edward Burns	9000.0	6.4	2011.0
2502	5042	Jon Gunn	1100.0	6.6	2004.0

2503 rows x 5 columns

Рисунок 13 – Выполнение пункта 2e

Теперь строк всего 2503 - за счет того, что все директоры с числом фильмов меньше 3, а также Стивен Спилберг, который оказался экстремальным (у него 26 фильмов, разрыв до ближайшего - 4, что больше, чем 0.1*(26-1) = 2,5), вылетели.

```
1  df_2e[df_2e.director_name == 'Steven Spielberg']
```

id	director_name	budget	imdb_score	title_year

Рисунок 14 – Отсутствие результатов по Спилбергу

f. Строит график зависимости imdb_score от бюджета фильма.

```
# Функция тренда
def Func(x, trend):
    return [sum([a * xi ** i for i, a in enumerate(trend[::-1])]) for xi in x]
```

```

trend = numpy.polyfit(df_2f['budget'], df_2f['imdb_score'], deg=1)
trend_y = Func(df_2f['budget'], trend)
```

```
import matplotlib.pyplot as plt #тут вроде понятно

df_2f = df_2e.sort_values(by = 'budget')

plt.figure(figsize=(20,15))

plt.title('ГРАФИК ЗАВИСИМОСТИ РЕЙТИНГА ОТ БЮДЖЕТА', fontsize=30)
plt.ylabel('IMDB SCORE', fontsize=30)
plt.xlabel('BUDGET', fontsize=30)

plt.plot (df_2f['budget'], df_2f['imdb_score'])
plt.plot(df_2f['budget'], trend_y, "r--")

plt.tick_params(axis='both', labelsize=20)

plt.grid()
plt.show()
```

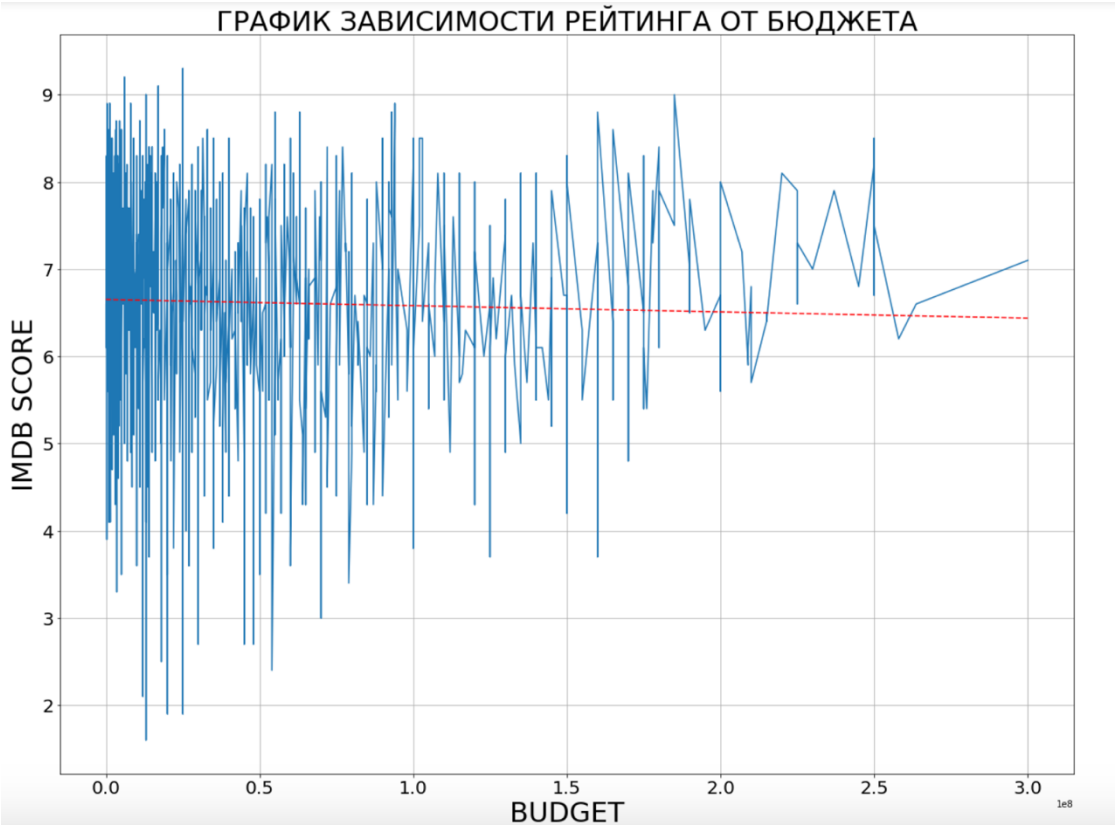

Рисунок 15 – График зависимости рейтинга от бюджета

g. На основе полученных данных проведите простейшее исследование наличия факта корреляции между бюджетом фильма и его рейтингом.

Для выявления корреляции были рассчитаны коэф-т корреляции Пирсона, а также ранговые коэф-ты корреляции Спирмена и Кендалла.

Все они находятся в библиотеке scipy.stats, каждая функция возвращает два числа – непосредственно сам коэф-т и значение p-value.

```
[ ]: 1 from scipy.stats import spearmanr,kendalltau,pearsonr

[ ]: 1 spearmanr(df_2f.imdb_score,df_2f.budget)

[ ]: SpearmanrResult(correlation=-0.16743550625375395, pvalue=3.3921610517020045e-17)

[ ]: 1 kendalltau (df_2f.imdb_score,df_2f.budget)

[ ]: KendalltauResult(correlation=-0.11455832777160319, pvalue=4.350722485999532e-17)

[ ]: 1 pearsonr (df_2f.imdb_score,df_2f.budget)
     2

[ ]: (-0.03278563628194725, 0.1010287514522426)
```

Рисунок 16 – Расчет коэф-тов корреляции

Таким образом тесты Спирмена и Кендалла подтверждают наличие слабой отрицательной корреляции на уровне значимости 0.05 , тест Пирсона отвергает наличие корреляции. По графику наличие или отсутствие корреляции определить трудно, поэтому была добавлена линия тренда, которая показала наличие слабой отрицательной корреляции.

Вывод

В ходе данной работы приобретены навыки в работе с базами данных с помощью средств языка Python.

Изначально был скачан файл с данными и отброшены лишние признаки (при помощи библиотеки **pandas**). Далее был импортирован модуль **sqlite3** и установлена связь с БД. После этого стало возможным писать запросы, в частности были сформированы запросы:

- На создание таблицы и загрузку данных
- На удаление пустых строк
- На удаление строк с нереальными значениями (анализ на нереальность проводился с помощью библиотеки **pandas**)
- С группировкой по одному из атрибутов
- С условиями на вхождение атрибута в кортеж (с помощью форматирования строк)

С помощью библиотеки **matplotlib** был построен график зависимости одного атрибута от другого.

С помощью библиотеки **scipy.stats** был проведен простой корреляционный анализ. Тесты Спирмена и Кендалла подтвердили корреляцию, тест Пирсона опроверг (это может быть вызвано тем, что тест Пирсона работает тем лучше, чем более нормальное распределение имеют атрибуты, что неочевидно).