SI 618 Project 1 Report

Yifei Sun

1. Motivation

As a movie lover, I am interested in movie genres and their revenues. Different movie genres usually have very different budgets and revenues, for example, big production movies tend to be action, war, science fiction, historical, etc.; while comedies and dramas tend to have lower investments, but some extremely good films often earn very high returns. With the introduction of film technology more than 100 years, the popularity of film genres is changing, once popular genres such as Western movies, etc., may not be so popular now. Figuring out film genres, budgets, revenues, and popularity can help us understand the development of movie industry and help movie studios make decisions.

Here I would like to answer the following three questions.

1. As times change, how did the number of different types of films change in this year as a percentage? (i.e., how does the popularity of different types of movies change with the times?)
2. how do the revenues, profits, budgets, return rates and ratings of different types of movies change with the era?
3. What are the movies with the highest net profits under the two prerequisites of considering inflation and not considering inflation? What are the highest-gross-profit movies? What are the films with the largest budgets? What are the movies with the highest return rate on investment? What are the movies that lost the most money?

2. Data Source

I used two data sources, both of them are from Kaggle. The first data source is The Movies Dataset, accessible at https://www.kaggle.com/datasets/rounakbanik/the-movies-dataset. It contains several csv files, I only used the csv file about metadata. In the csv file of metadata, there is information about the movie's genre, language, budget, revenue, number, name and many other information. This is based on a total of about 45,000 data, spanning over 100 years, from 1870 until 2017. However, the data before 1915 are sparse and incomplete. Since early films were not commercial projects but more scientific practices, budgets and revenues did not exist. Another problem is that a large number of film entries have a budget or revenue of 0. This is very unreasonable theoretically, and is most likely due to incomplete data collection, with the uncollected data being recorded as 0.

The other data I use is the US Consumer Price Index and Inflation (CPI), accessible at https://www.kaggle.com/datasets/tunguz/us-consumer-price-index-and-inflation-cpi. This is CPI data from the U.S. Department of Labor, from 1913 to 2021, on a monthly basis, and contains both date and index columns. The total number of data is about 1300. The data is in csv format. I use this data to represent inflation and currency devaluation.

3. Data Manipulation Methods

I need to be very specific about the data and do accurate operation to any of it. For CPI data, the processing is relatively simple. First I delete the header line in the csv, then use pyspark to read, and create two new columns, month and year. When reading, note that the numbers are expressed as integers. Then I use sparksql to build a dataframe for the operation. There is no missing or incomplete data and the data quality is good. CPI data needs to be converted to double decimals. The result of this step is a nicely constructed dataframe ready for joining and further manipulation.

Below is my codes for processing of CPI data:

```
input_file = sc.textFile("US CPI.csv")
input_file = input_file.flatMap(lambda line: [line.split(",")[:5]+line.split(",")[0].split("/")])
cpi_df = input_file.toDF()
cpi_df.registerTempTable("cpi_df")
cpi_df.show()
cpi_df.count()
```

This part is to create and filter the data I need. And create the dataframe for manipulation.

```
cpi_df_1 = sqlc.sql("""SELECT  cpi_df._1 as Date, CAST(cpi_df._2 as double) as CPI, CAST(cpi_df._3 as int) as Day, CAST(cpi_df._4 as int) as Month, CAST(cpi_df._5 as int) as Year FROM cpi_df
```

""")
cpi_df_1.registerTempTable("cpi_df_1")
cpi_df_1.show()

This part is to build SQL dataframe with related columns.

Below is the result of this step:

```
+---------+----+---+-----+----+
|     Date| CPI|Day|Month|Year|
+---------+----+---+-----+----+
| 1/1/1913| 9.8|  1|    1|1913|
| 1/2/1913| 9.8|  1|    2|1913|
| 1/3/1913| 9.8|  1|    3|1913|
| 1/4/1913| 9.8|  1|    4|1913|
| 1/5/1913| 9.7|  1|    5|1913|
| 1/6/1913| 9.8|  1|    6|1913|
| 1/7/1913| 9.9|  1|    7|1913|
```

The handling of metadata is very complicated. I encountered a lot of difficulties. Because this data is quite messy and missing in many places, it causes some problems for read csv. Parts that should be numbers, such as budget, revenue, rating, or year, appear as textual content read out from other parts. In addition, this csv file has several columns of elements are json file format, resulting in the problem of "comma in quotes". When reading the file, we should be careful to escape the comma in the quotes, and do not take them as delimiter. (This problem took me a long time in the MapReduce section) I also need to remove the header line of this file. For the data between 1870 and 1915, I removed the data because there is no budget or revenue. I also removed the entries that showed zero budget or revenue, because it was obvious that this was due to insufficient data collection, and keeping these data would have posed a great challenge to the accuracy and feasibility of my profit and return rate calculations later. For data that should be numbers, I converted them to decimals or integers accordingly, such as budget, revenue. To facilitate the operation, in the Spark and Sparksql part, I removed the unnecessary extra columns from the csv in advance and extracted the corresponding month, day and year from the date using excel as the new three columns. (I tried to operate in spark at first, but there would always be problems that I could not solve, probably caused by the inconsistent data format, which led to my inability to do ORDER BY and .count in the last part)
In the join section I performed another more complex operation. Using the SQL statement CASE WHEN THEN ELSE END, I tried to achieve the following: if the date is in the second half of the month, it will be classified as the first of the next month, and if the date is in the first half of the month, it will be classified as the first of the month. This is because my CPI data only contains data for the first day of each month, so I'm trying to find the CPI data corresponding to the day closest to the release date of this movie. In the JOIN step, I use the same year and month and LEFT JOIN to get the desired result. (Because not all months may correspond to movie releases, try to keep the information in the table on the left) From this I got the budget of each movie, the revenue and their corresponding CPI index at that time. This allows me to calculate the revenue and profit with inflation removed based on the CPI index.
Below is my codes for Spark manipulation of revenue and budget data:

```
input_file = sc.textFile("movies_metadata_2.csv")
input_file = input_file.flatMap(lambda line: [line.split(",")[:8]])
input_file = input_file.filter(lambda line: line[0].isnumeric() and line[3].isnumeric() and line[4].isnumeric() and line[5].isnumeric() and line[6].isnumeric() and line[7].isnumeric())
metadata_df = input_file.toDF()
metadata_df.registerTempTable("metadata_df")
metadata_df.show(20)
```

```
metadata_df.count()
```

This step is to create and filter the information I need. Filter out the data that has elements that are supposed to be number but are not numbers. And create the dataframe for manipulation.

```
metadata_df_2 = sqlc.sql("""
SELECT Title,Release_Date,Budget,Revenue,Profit,Return_Rate,Run_Time,Adjusted_Month_2 as Month,Adjusted_Year as Year FROM
(SELECT metadata_df._2 as Title, metadata_df._3 as Release_Date, CAST(metadata_df._1 as int) as Budget, CAST(metadata_df._4 as int) as Revenue, CAST(metadata_df._4 as int)-CAST(metadata_df._1 as int) as Profit, CAST(metadata_df._4 as int)/CAST(metadata_df._1 as int) as Return_Rate, CAST(metadata_df._5 as int) as Run_Time,
CAST(_6 as int) as Month, (CASE WHEN CAST(metadata_df._7 as int) > 15 THEN CAST(metadata_df._6 as int)+1 ELSE CAST(metadata_df._6 as int) END) as Adjusted_Month_1,
CAST(metadata_df._7 as int) as Day, CAST(metadata_df._8 as int) as Year, (CASE WHEN (CASE WHEN CAST(metadata_df._7 as int) > 15 THEN CAST(metadata_df._6 as int)+1 ELSE CAST(metadata_df._6 as int) END) > 12 THEN CAST(metadata_df._8 as int)+1 ELSE CAST(metadata_df._8 as int) END) as Adjusted_Year, (CASE WHEN (CASE WHEN CAST(metadata_df._7 as int) > 15 THEN CAST(metadata_df._6 as int)+1 ELSE CAST(metadata_df._6 as int) END) > 12 THEN (CASE WHEN CAST(metadata_df._7 as int) > 15 THEN CAST(metadata_df._6 as int)+1 ELSE CAST(metadata_df._6 as int) END)-12 ELSE
(CASE WHEN CAST(metadata_df._7 as int) > 15 THEN CAST(metadata_df._6 as int)+1 ELSE CAST(metadata_df._6 as int) END) END) as Adjusted_Month_2 FROM metadata_df) """)
metadata_df_2.registerTempTable("metadata_df_2")
metadata_df_2.show()
```

```
+--------------------+------------+--------+---------+---------+------------------+--------+-----+----+
|               Title|Release_Date|  Budget|  Revenue|   Profit|       Return_Rate|Run_Time|Month|Year|
+--------------------+------------+--------+---------+---------+------------------+--------+-----+----+
|           Toy Story|  10/30/1995|30000000|373554033|343554033|        12.4518011|      81|   11|1995|
|             Jumanji|  12/15/1995|65000000|262797249|197797249|         4.0430346|     104|   12|1995|
|  Waiting to Exhale|  12/22/1995|16000000| 81452156| 65452156|        5.09075975|     127|    1|1996|
|                Heat|  12/15/1995|60000000|187436818|127436818| 3.123946966666667|     170|   12|1995|
|        Sudden Death|  12/22/1995|35000000| 64350171| 29350171|1.8385763142857143|     106|    1|1996|
|           GoldenEye|  11/16/1995|58000000|352194034|294194034| 6.072310931034483|     130|   12|1995|
|The American Pres...|  11/17/1995|62000000|107879496| 45879496| 1.739991870967742|     106|   12|1995|
```

This step is to create the data I need such as return rate, profit. And change the month , and year of each data to the closest month and year of that date.

```
metadata_df_4 = sqlc.sql("""
SELECT Title, Release_Date, Budget, Revenue, Profit,Return_Rate, Adjusted_Budget, Adjusted_Revenue, Adjusted_Profit FROM
(SELECT A.Title as Title, A.Release_Date as Release_Date, A.Budget as Budget,A.Revenue as Revenue,A.Profit as Profit, A.Return_Rate as Return_Rate,
A.Run_Time as Run_Time,
B.CPI as CPI,
A.Budget/B.CPI as Adjusted_Budget,
A.Revenue/B.CPI as Adjusted_Revenue,
A.Profit/B.CPI as Adjusted_Profit
FROM
```

```
metadata_df_2 as A
LEFT JOIN cpi_df_1 as B
ON A.Year = B.Year AND A.Month = B.MONTH)
""")
metadata_df_4.registerTempTable("metadata_df_4")
metadata_df_4.show(20)
```

```
+--------------------+------------+---------+---------+---------+--------------------+-----------------+-----------------+-----------------
+
|                Title|Release_Date|   Budget|  Revenue|   Profit|         Return_Rate|  Adjusted_Budget|  Adj
usted_Revenue|     Adjusted_Profit|
+--------------------+------------+---------+---------+---------+--------------------+-----------------+-----------------+-----------------
+
| Across to Singapore|    4/7/1928|   290000|   596000|   306000|  2.0551724137931036| 16959.06432748538| 34
853.80116959064| 17894.736842105263|
|Captain America: ...|   3/20/2014|170000000|714766572|544766572|   4.204509247058824| 717081.7304447594|3014
976.7665519337|  2297895.036107174|
|                Noah|   3/20/2014|125000000|362637473|237637473|         2.901099784| 527265.9782682054|1
529651.2156644394| 1002385.2373962341|
| Muppets Most Wanted|   3/20/2014|  50000000|  80383290|  30383290|          1.6076658|210906.39130728217|3
39066.99230613484| 128160.60099885266|
|            Bad Words|    9/6/2013|  9500000|  7800000| -1700000|  0.8210526315789474|40572.456000239166|
33312.12176861742| -7260.334231621745|
|キャプテンハーロック|    9/7/2013|  30000000|  17137302|-12862698|           0.5712434|128123.54526391315|
73189.72961661164|-54933.815647301504|
```

This step is to get the final dataframe that I am using to get the results. All the unnecessary data are filtered out.

Here is one example of my Mapreduce codes. It includes mapper, combiner and reducer. I is used to get the total count and total revenue of a genre in a particular year:

```
import mrjob
import json
import csv
from mrjob.job import MRJob

class RevenueCount(MRJob):
    OUTPUT_PROTOCOL = mrjob.protocol.TextProtocol
    def mapper(self, _, line):
        try:
            # line = line.encode(encoding = 'UTF-8', errors = 'ignore')
            line_list = list(csv.reader([line], delimiter=',', quotechar='"'))[0]
            genres = line_list[3]
            id =  line_list[5]
            original_language = line_list[7]
            title = line_list[20]
            release_date = line_list[14]
            spoken_languages = line_list[17]
            release_year = release_date[0:4]
            revenue = int(line_list[15])
            if genres != "":
                genres_json_list = json.loads(genres.replace("'", '"'))
                for dictionary in genres_json_list:
                    genre = dictionary["name"]
                    yield (release_year,"Genre:",genre),(1,revenue)
            if spoken_languages != "":
                spoken_languages_json_list = json.loads(spoken_languages.replace("'", '"'))
                dictionary = spoken_languages_json_list[0]
                language = dictionary["name"]
```

```
                yield (release_year,"Language:",language),(1,revenue)
            yield (release_year,"All Languages All Genres:", ""),(1,revenue)
        except:
            pass
    def combiner(self, key, values):
        counts = 0
        revenues = 0
        for value in values:
            counts += value[0]
            revenues += value[1]
        yield key, (counts,revenues)
    def reducer(self, key, values):
        counts = 0
        revenues = 0
        for value in values:
            counts += value[0]
            revenues += value[1]
        yield key[0]+"\t\t"+key[1]+"\t"+key[2], "Counts:"+"\t"+str(counts)+"\t\t"+"Revenues:"+"\t"+str(revenues)
if __name__ == '__main__':
    RevenueCount.run()
```

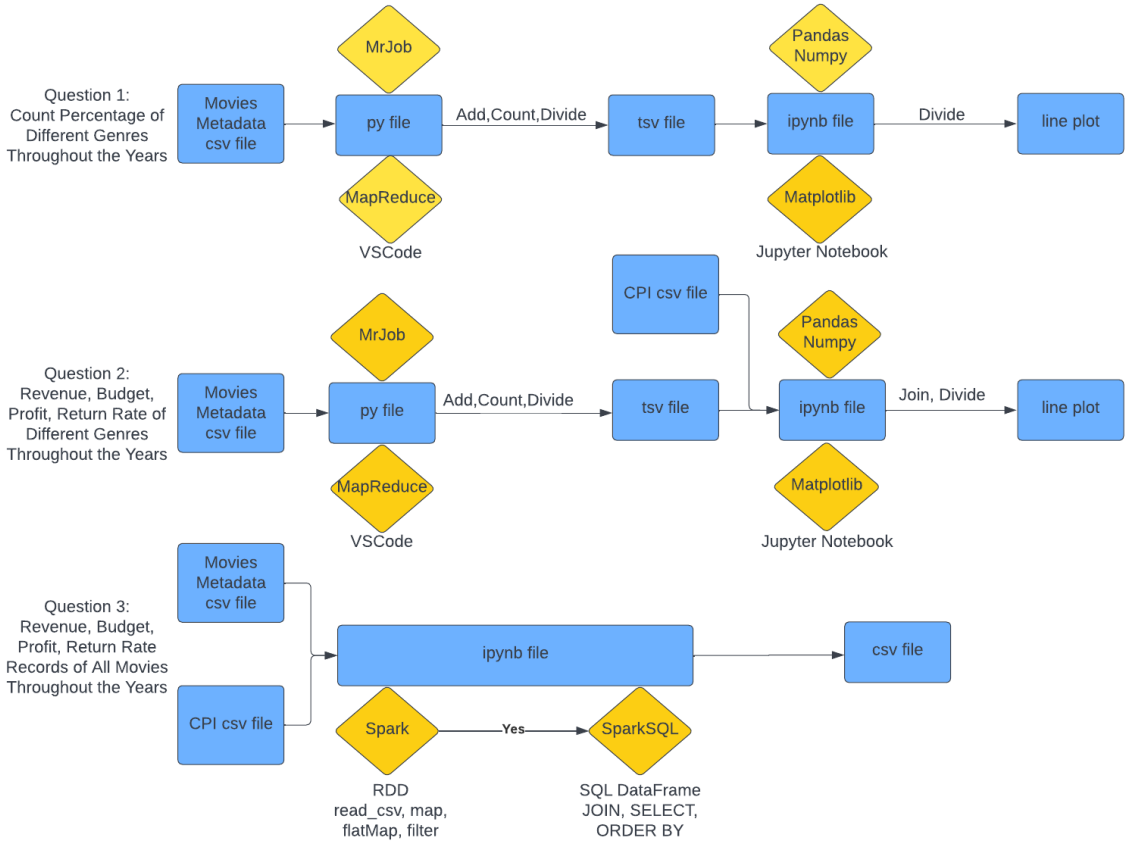
After I get the tsv files as the results of mapreduce process, I use pandas and matplotlib to visualize my results in line plots.

4. Analysis and Visualization

For the first question, I write several python codes separately and run them locally in my PC. After I get the result tsv files. I use pandas to read them into dataframes in Jupyter Notebook. I divide the count of every genre by the total count in that year and get the count percentage of every genre in that year. Then I use Matplotlib to plot the results.

For the second question, I write several python codes separately and run them locally in my PC. After I get the result tsv files. I use pandas to read them into dataframes in Jupyter Notebook, and also read the CPI data into dataframe. Them I merge these two dataframes by the same column “year”. And divide revenue, profit, budget by the CPI index to get the adjusted revenue, profit, budget. Then I use Matplotlib to plot the results.

For spark and sparksql data in the third question, I first build an RDD with spark to load the data, do the necessary splitting and filtering, splitting a row of data as a string into a list, then filtering out the undesired elements and keeping only the necessary columns of information. Then from the RDD, I build the Dataframe, and can perform SQL operations. The date of each movie is grouped into the nearest day by CPI data, and the CPI dataframe is LEFT JOIN to the metadata data by two conditions, year and month. The result is a dataframe containing each movie's revenue, budget and the corresponding CPI index at the time of its release. With this dataframe, I can calculate information such as profit as well as return, and profit and return excluding the inflation factor. With this dataframe, we can select the information we need and sort it into various categories to get the answers to the questions we want to know. All the operations are done in the Great Lakes cluster.
Here is a flowchart:




```
metadata_df_2 = sqlc.sql("""
SELECT Title,Release_Date,Budget,Revenue,Profit,Return_Rate,Run_Time,Adjusted_Month_2 as Month,Adjusted_Year as Year FROM
(SELECT
metadata_df._2 as Title,
metadata_df._3 as Release_Date,
CAST(metadata_df._1 as int) as Budget,
CAST(metadata_df._4 as int) as Revenue,
CAST(metadata_df._4 as int)-CAST(metadata_df._1 as int) as Profit,
CAST(metadata_df._4 as int)/CAST(metadata_df._1 as int) as Return_Rate,
CAST(metadata_df._5 as int) as Run_Time,
CAST(_6 as int) as Month,
(CASE WHEN CAST(metadata_df._7 as int) > 15 THEN CAST(metadata_df._6 as int)+1 ELSE CAST(metadata_df._6 as int) END)
as Adjusted_Month_1,
CAST(metadata_df._7 as int) as Day,
CAST(metadata_df._8 as int) as Year,
(CASE WHEN (CASE WHEN CAST(metadata_df._7 as int) > 15 THEN CAST(metadata_df._6 as int)+1
ELSE CAST(metadata_df._6 as int) END) > 12 THEN CAST(metadata_df._8 as int)+1 ELSE CAST(metadata_df._8 as int) END)
as Adjusted_Year,
(CASE WHEN
(CASE WHEN CAST(metadata_df._7 as int) > 15 THEN CAST(metadata_df._6 as int)+1 ELSE CAST(metadata_df._6 as int) END)
> 12
THEN
(CASE WHEN CAST(metadata_df._7 as int) > 15 THEN CAST(metadata_df._6 as int)+1 ELSE CAST(metadata_df._6 as int) END)-12
ELSE
(CASE WHEN CAST(metadata_df._7 as int) > 15 THEN CAST(metadata_df._6 as int)+1 ELSE CAST(metadata_df._6 as int) END)
END) as Adjusted_Month_2
FROM metadata_df)

""")
metadata_df_2.registerTempTable("metadata_df_2")
metadata_df_2.show()
```

This code is to get Profit, Rate of Return and the closest month and year of each movie.

Title	Release_Date	Budget	Revenue	Profit	Return_Rate	Run_Time	Month	Year
Toy Story	10/30/1995	30000000	373554033	343554033	12.4518011	81	11	1995
Jumanji	12/15/1995	65000000	262797249	197797249	4.0430346	104	12	1995
Waiting to Exhale	12/22/1995	16000000	81452156	65452156	5.09075975	127	1	1996
Heat	12/15/1995	60000000	187436818	127436818	3.123946966666667	170	12	1995
Sudden Death	12/22/1995	35000000	64350171	29350171	1.8385763142857143	106	1	1996
GoldenEye	11/16/1995	58000000	352194034	294194034	6.072310931034483	130	12	1995
The American Pres...	11/17/1995	62000000	107879496	45879496	1.739991870967742	106	12	1995
Nixon	12/22/1995	44000000	13681765	-30318235	0.31094920454545455	192	1	1996
Cutthroat Island	12/22/1995	98000000	10017322	-87982678	0.10221757142857144	119	1	1996
Casino	11/22/1995	52000000	116112375	64112375	2.2329302884615383	178	12	1995
Sense and Sensibi...	12/13/1995	16500000	135000000	118500000	8.181818181818182	136	12	1995
Four Rooms	12/9/1995	4000000	4300000	300000	1.075	98	12	1995
Ace Ventura: When...	11/10/1995	30000000	212385533	182385533	7.079517766666667	90	11	1995
Money Train	11/21/1995	60000000	35431113	-24568887	0.59051855	103	12	1995
Get Shorty	10/20/1995	30250000	115101622	84851622	3.805012297520661	105	11	1995
Assassins	10/6/1995	50000000	30303072	-19696928	0.60606144	132	10	1995
Leaving Las Vegas	10/27/1995	3600000	49800000	46200000	13.833333333333334	112	11	1995
Now and Then	10/20/1995	12000000	27400000	15400000	2.283333333333333	100	11	1995
La Cité des Enfan...	5/16/1995	18000000	1738611	-16261389	0.0965895	108	6	1995
Twelve Monkeys	12/29/1995	29500000	168840000	139340000	5.7233898305084745	129	1	1996

only showing top 20 rows

```
cpi_df_1 = sqlc.sql("""
SELECT
cpi_df._1 as Date,
CAST(cpi_df._2 as double) as CPI,
CAST(cpi_df._3 as int) as Day,
CAST(cpi_df._4 as int) as Month,
CAST(cpi_df._5 as int) as Year
FROM cpi_df
""")
cpi_df_1.registerTempTable("cpi_df_1")
cpi_df_1.show()
```

This code is to get the dataframe containing the year, month and the corresponding CPI index.

Date	CPI	Day	Month	Year
1/1/1913	9.8	1	1	1913
1/2/1913	9.8	1	2	1913
1/3/1913	9.8	1	3	1913
1/4/1913	9.8	1	4	1913
1/5/1913	9.7	1	5	1913
1/6/1913	9.8	1	6	1913
1/7/1913	9.9	1	7	1913
1/8/1913	9.9	1	8	1913
1/9/1913	10.0	1	9	1913
1/10/1913	10.0	1	10	1913
1/11/1913	10.1	1	11	1913
1/12/1913	10.0	1	12	1913
1/1/1914	10.0	1	1	1914
1/2/1914	9.9	1	2	1914
1/3/1914	9.9	1	3	1914
1/4/1914	9.8	1	4	1914
1/5/1914	9.9	1	5	1914
1/6/1914	9.9	1	6	1914
1/7/1914	10.0	1	7	1914
1/8/1914	10.2	1	8	1914

only showing top 20 rows

```
metadata_df_4 = sqlc.sql("""
SELECT Title, Release_Date, Budget, Revenue, Profit,Return_Rate, Adjusted_Budget, Adjusted_Revenue, Adjusted_Profit FROM
(SELECT A.Title as Title, A.Release_Date as Release_Date, A.Budget as Budget,A.Revenue as Revenue,A.Profit as Profit, A.Return_Rate a
A.Run_Time as Run_Time,
B.CPI as CPI,
A.Budget/B.CPI as Adjusted_Budget,
A.Revenue/B.CPI as Adjusted_Revenue,
A.Profit/B.CPI as Adjusted_Profit
FROM
metadata_df_2 as A
LEFT JOIN cpi_df_1 as B
ON A.Year = B.Year AND A.Month = B.MONTH)
""")
metadata_df_4.registerTempTable("metadata_df_4")
metadata_df_4.show(20)
```

This code is to get the final dataframe for analyzing containing, budgt, revenue, profit, rate of return, and their adjusted versions based on the corresponding CPI index.

```
+--------------------+------------+--------+--------+---------+--------------------+----------------+----------------+-----------------+
|               Title|Release_Date|  Budget| Revenue|   Profit|         Return_Rate|  Adjusted_Budget| Adjusted_Revenue|   Adjusted_Profit|
+--------------------+------------+--------+--------+---------+--------------------+----------------+----------------+-----------------+
| Across to Singapore|    4/7/1928|  290000|  596000|   306000|  2.0551724137931036| 16959.06432748538| 34853.80116959064| 17894.736842105263|
|Captain America: ...|   3/20/2014|170000000|714766572|544766572|   4.204509247058824| 717081.7304447594|3014976.7665519337|  2297895.036107174|
|                Noah|   3/20/2014|125000000|362637473|237637473|         2.901099784| 527265.9782682054|1529651.2156644394| 1002385.2373962341|
| Muppets Most Wanted|   3/20/2014| 50000000| 80383290| 30383290|          1.6076658|210906.39130728217|339066.99230613484| 128160.60099885266|
|           Bad Words|    9/6/2013|  9500000|  7800000| -1700000|  0.8210526315789474|40572.456000239166| 33312.12176861742| -7260.334231621745|
|キャプテンハーロック|   9/7/2013| 30000000| 17137302|-12862698|          0.5712434|128123.54526391315| 73189.72961661164|-54933.815647301504|
|          Blood Ties|   8/22/2013| 25500000|  2415472|-23084528| 0.09472439215686275|108905.01347432617|10315.961204190495| -98589.05227013568|
|             13 Sins|   4/18/2014|  5000000|    13809| -4986191|          0.0027618| 21017.23413198823| 58.04539722572509|-20959.188734762505|
|The Legend of Her...|   1/10/2014| 70000000| 61279452| -8720548|  0.8754207428571429|299252.72319978115|261972.04124557532|-37280.681954205786|
|300: Rise of an E...|    3/5/2014|110000000|337580051|227580051|  3.0689095545454546| 465523.7353624525|1428650.2393215203|  963126.5039590677|
|      Under the Skin|   3/14/2014| 13300000|  5380251| -7919749| 0.40453015037593987|56286.051639278354|22769.404933705187| -33516.64670557316|
|      Need for Speed|   3/13/2014| 66000000|203277636|137277636|  3.0799641818181818|279314.24121747154| 860277.8584215359|  580963.6172040644|
|       Veronica Mars|   3/13/2014|  6000000|  3485127| -2514873|          0.5808545|25392.203747042866| 14749.17581138671|-10643.027935656155|
|          Son of God|   2/28/2014| 22000000| 67800064| 45800064|   3.081821090909091| 93104.74707249051| 286932.1731917577| 193827.42611926718|
| That Awkward Moment|   1/29/2014|  8000000| 26049082| 18049082|         3.25613525|34074.307546181335|110950.55392046204|  76876.24637428072|
|      3 Days to Kill|   2/14/2014| 28000000| 52597999| 24597999|  1.8784999642857143|119260.07641163467| 224030.0492799673| 104769.97286833261|
|           Non-Stop|   1/26/2014| 50000000|222809600|172809600|           4.456192|212964.42216363334| 949010.3543302056|  736045.9321665722|
|Mr. Peabody & She...|    2/7/2014|145000000|272912430|127912430|  1.8821546896551724| 617596.8242745367|1162412.7591244606|   544815.934849924|
|            Barefoot|    2/2/2014|  6000000|    15071| -5984929|0.002511833333333...|   25555.730659636| 64.19173612856235| -25491.53892350744|
|       Best Man Down|  10/20/2012|  1500000|     1938| -1498062|           0.001292| 6515.478605340086| 8.417998358099391| -6507.060606981987|
+--------------------+------------+--------+--------+---------+--------------------+----------------+----------------+-----------------+
only showing top 20 rows
```

```
metadata_df_6 = sqlc.sql("""
SELECT Title, Release_Date, Budget, Revenue, Profit, Adjusted_Profit FROM
metadata_df_4
ORDER BY Adjusted_Profit DESC
""")
metadata_df_6.show(100)
metadata_df_6.coalesce(1).write.csv("removal Adjusted_Profit DESC.csv")
metadata_df_6
```

This is one example of my codes to analyze the final dataframe and get answers to my questions. For example, here we can get the most profitable film in the history adjusted by inflation.

```
+--------------------+------------+---------+----------+----------+--------------------+
|               Title|Release_Date|   Budget|   Revenue|    Profit|     Adjusted_Profit|
+--------------------+------------+---------+----------+----------+--------------------+
|  Gone with the Wind|  12/15/1939|  4000000| 400176459| 396176459|        2.82983185E7|
| Alice in Wonderland|    7/3/1951|  3000000| 572000000| 569000000| 2.196911196911197E7|
|               Bambi|   8/14/1942|   858000| 267447150| 266589150|1.6156918181818182E7|
|Snow White and th...|  12/20/1937|  1488423| 184925486| 183437063|1.2918103028169015E7|
|           Star Wars|   5/25/1977| 11000000| 775398007| 764398007|1.2593047891268533E7|
|          Cinderella|    3/4/1950|  2900000| 263591415| 260691415|1.1046246398305085E7|
|             Titanic|  11/18/1997|200000000|1845034188|1645034188|1.0198600049597023E7|
|        The Exorcist|  12/26/1973|  8000000| 441306145| 433306145|   9298415.128755365|
|  The Sound of Music|    3/2/1965|  8200000| 286214286| 278014286|   8882245.559105432|
|                Jaws|   6/18/1975|  7000000| 470654000| 463654000|    8554501.84501845|
|E.T. the Extra-Te...|    4/3/1982| 10500000| 792965326| 782465326|   8245156.227608008|
|Star Wars: The Fo...|  12/15/2015|245000000|2068223624|1823223624|   7708375.960257901|
|One Hundred and O...|   1/25/1961|  4000000| 215880014| 211880014|   7110067.583892617|
|The Empire Strike...|   5/17/1980| 18000000| 538400000| 520400000|  6292623.9419588875|
|     The Jungle Book|  10/17/1967|  4000000| 205843612| 201843612|   5971704.497041421|
|       Jurassic Park|   6/11/1993| 63000000| 920100000| 857100000|   5935595.567867036|
|            Pinocchio|   2/23/1940|  2600000|  84300000|  81700000|   5835714.285714285|
+--------------------+------------+---------+----------+----------+--------------------+
```

Below is a sample code to calculate the standard deviation and mean of the ratings of films of a genre in a year

```
import mrjob
import json
import csv
from mrjob.job import MRJob
from mrjob.step import MRStep

class RatingAvgStDev(MRJob):
    OUTPUT_PROTOCOL = mrjob.protocol.TextProtocol
    def mapper(self, _, line):
        try:
            # line = line.encode(encoding = 'UTF-8', errors = 'ignore')
            line_list = list(csv.reader([line], delimiter=',', quotechar='"'))[0]
            genres = line_list[3]
            id =  line_list[5]
            original_language = line_list[7]
            title = line_list[20]
            release_date = line_list[14]
            spoken_languages = line_list[17]
            release_year = release_date[0:4]
            revenue = int(line_list[15])
            vote_average = float(line_list[-2])
            if genres != "":
                genres_json_list = json.loads(genres.replace("'", '"'))
                for dictionary in genres_json_list:
                    genre = dictionary["name"]
                    yield (release_year,"Genre:",genre),(1,vote_average,vote_average**2)
            if spoken_languages != "":
                spoken_languages_json_list = json.loads(spoken_languages.replace("'", '"'))
```

```
                dictionary = spoken_languages_json_list[0]
                language = dictionary["name"]
                yield (release_year,"Language:",language),(1,vote_average,vote_average**2)
            yield (release_year,"All Languages All",""),(1,vote_average,vote_average**2)
        except:
            pass
    def combiner(self, key, values):
        counts = 0
        ratings = 0
        rating_sqs = 0
        for value in values:
            counts += value[0]
            ratings += value[1]
            rating_sqs += value[2]
        yield key, (counts,ratings,rating_sqs)
    def reducer(self, key, values):
        counts = 0
        ratings = 0
        rating_sqs = 0
        for value in values:
            counts += value[0]
            ratings += value[1]
            rating_sqs += value[2]
        yield key, (counts,ratings,rating_sqs)
    def st_dev_calculator(self, key, values):
        for value in values:
            n = value[0]
            if n >= 10:
                x_sum = value[1]
                x2_sum = value[2]
                avg = x_sum/n
                st_dev = ((x2_sum-(x_sum**2/n))/(n-1))**(1/2)
                yield key[0]+"\t\t"+key[1]+"\t"+key[2], "Mean:\t"+str(round(avg,2))+"\t\tStandard
Deviation:\t"+str(round(st_dev,2))

    def steps(self):
        return [
            MRStep(mapper = self.mapper,
                combiner = self.combiner,
                reducer = self.reducer),
            MRStep(reducer = self.st_dev_calculator)
        ]

if __name__ == '__main__':
    RatingAvgStDev.run()
```

Here is a sample result of the code above

```
1935    Language:       English Mean:   4.95
1936    All Languages All       Mean:   5.66
1936    Genre:  Comedy  Mean:   5.43
1936    Genre:  Crime   Mean:   6.46
1936    Genre:  Drama   Mean:   5.78
1936    Genre:  History Mean:   5.22
1936    Genre:  Music   Mean:   5.29
1936    Genre:  Mystery Mean:   6.41
1936    Genre:  Romance Mean:   5.77
1936    Genre:  Thriller        Mean:   6.78
1936    Genre:  Western Mean:   3.74
1936    Language:       English Mean:   5.54
1936    Language:       Français        Mean:   6.97
1937    All Languages All       Mean:   5.38
1937    Genre:  Action  Mean:   5.61
1937    Genre:  Adventure       Mean:   5.35
1937    Genre:  Comedy  Mean:   5.24
1937    Genre:  Crime   Mean:   6.48
1937    Genre:  Drama   Mean:   5.63
1937    Genre:  History Mean:   5.66
1937    Genre:  Music   Mean:   5.24
1937    Genre:  Mystery Mean:   5.84
1937    Genre:  Romance Mean:   5.56
1937    Genre:  Thriller        Mean:   6.5
1937    Language:       English Mean:   5.35
1938    All Languages All       Mean:   5.28
1938    Genre:  Adventure       Mean:   5.35
```

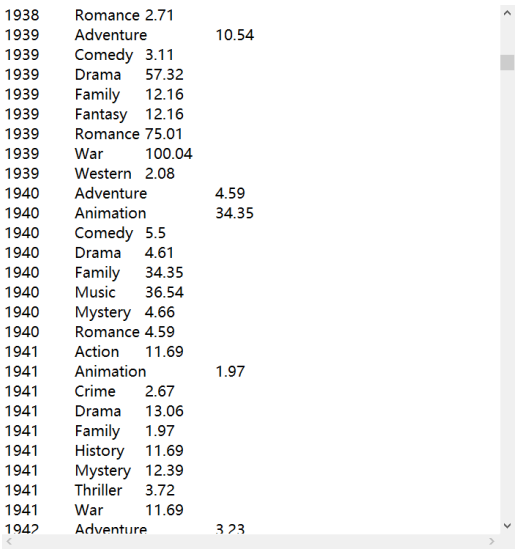
Here is another sample code I used to get the average rate of return of different genres throughout the years. Note I exclude any data that has 0 or null revenue or budget.

```
import mrjob
import json
```

```
import csv
from mrjob.job import MRJob


class AverageReturnRate(MRJob):
    OUTPUT_PROTOCOL = mrjob.protocol.TextProtocol
    def mapper(self, _, line):
        try:
            # line = line.encode(encoding = 'UTF-8', errors = 'ignore')
            line_list = list(csv.reader([line], delimiter=',', quotechar='"'))[0]
            genres = line_list[3]
            id =  line_list[5]
            original_language = line_list[7]
            title = line_list[20]
            release_date = line_list[14]
            spoken_languages = line_list[17]
            release_year = release_date[-4:]
            revenue = int(line_list[15])
            budget = int(line_list[2])
            if genres != ""  and revenue != 0 and revenue != None and budget != 0 and budget != None and
(release_year.startswith("19") or release_year.startswith("20")):
                genres_json_list = json.loads(genres.replace("'", '"'))
                for dictionary in genres_json_list:
                    genre = dictionary["name"]
                    yield (release_year,genre),(revenue,budget)
        except:
            pass
    def combiner(self, key, values):
        revenues = 0
        budgets = 0
        for value in values:
            revenues += value[0]
            budgets += value[1]
        yield key, (revenues,budgets)
    def reducer(self, key, values):
        revenues = 0
        budgets = 0
        for value in values:
            revenues += value[0]
            budgets += value[1]
        yield key[0]+"\t"+key[1], str(round((revenues/budgets),2))
if __name__ == '__main__':
    AverageReturnRate.run()
```

Here is one section of the result



1938	Romance	2.71
1939	Adventure	10.54
1939	Comedy	3.11
1939	Drama	57.32
1939	Family	12.16
1939	Fantasy	12.16
1939	Romance	75.01
1939	War	100.04
1939	Western	2.08
1940	Adventure	4.59
1940	Animation	34.35
1940	Comedy	5.5
1940	Drama	4.61
1940	Family	34.35
1940	Music	36.54
1940	Mystery	4.66
1940	Romance	4.59
1941	Action	11.69
1941	Animation	1.97
1941	Crime	2.67
1941	Drama	13.06
1941	Family	1.97
1941	History	11.69
1941	Mystery	12.39
1941	Thriller	3.72
1941	War	11.69
1942	Adventure	3.23

I used pandas and numpy to analyze and visualize the results above

```
import pandas as pd
import numpy as np
```

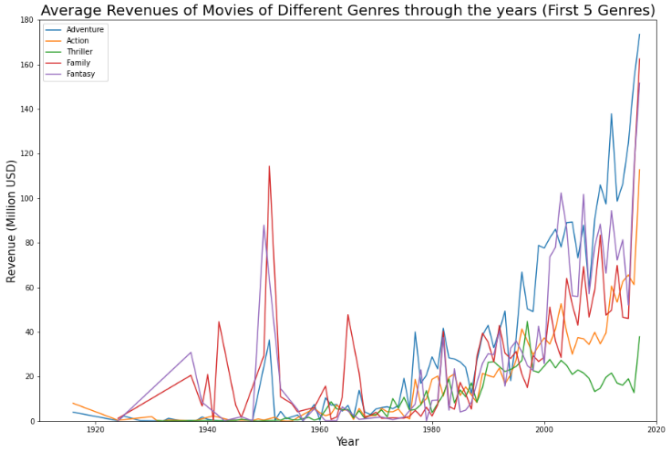
```
df_1 = pd.read_csv('yifeisun_si618_project1_RatingAvgStDev.tsv', sep='\t')
df_1 = df_1[["Unnamed: 0","All Languages All","Unnamed: 3","3.58","3.86"]]
df_1.columns = ["Year","Kind","Value","Mean","Standard Deviation"]
df_1 = df_1.dropna(subset=['Year'])
df_1["Year"] = df_1["Year"].astype(int)
df_1
```

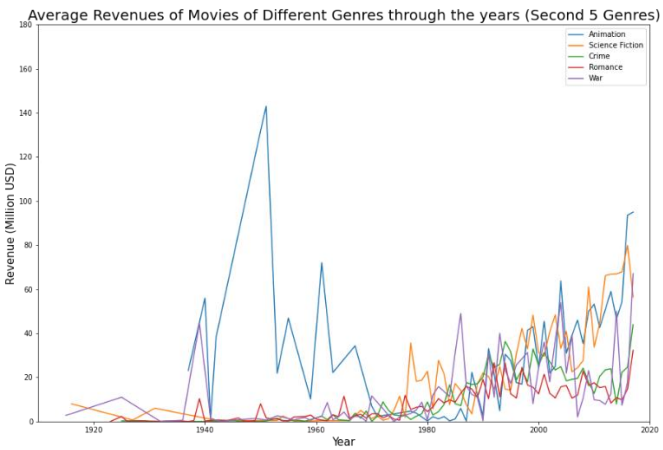
```
df_1_dict_year = {}
for year in list(set(list(df_1_genre["Year"]))):
    df_1_dict_year[year] = df_1_genre[df_1_genre["Year"] == year]
```

```
df_1_dict_genre = {}
for genre in list(set(list(df_1_genre["Genre"]))):
    df_1_dict_genre[genre] = df_1_genre[df_1_genre["Genre"] == genre]
```

```
import matplotlib.pyplot as plt
plt.figure(figsize=(15,10))
for genre in mean_list:
    if genre in mean_list_1:
        df = df_1_dict_genre[genre]
        plt.plot(df["Year"], df["Mean Rating"], label = genre)
plt.title("Average Rating of Movies of Different Genres through the years (Second 5 Genres)", fontsi
plt.xlabel("Year", fontsize=15)
plt.ylabel("Average Rating", fontsize=15)
plt.ylim([0, 10])
plt.xlim([1910, 2020])
plt.legend()
plt.show()
```

I looked at the relationship between different genres and their count percentage, average revenue, revenue percentage, average rating and rating standard deviation throughput the years. And also did some of the work to analyze different languages of films and their relationships, but the problem is majority of the films in the dataset are in English, and other languages are only accountable for a small percentage, therefore their could be huge deviation and biases, and also the data is very incomplete. So I forewent doing the language analysis, because the distribution of genres are more even. I also try to find the more profitable films and least profitable films in the history, and the results are pretty interesting.

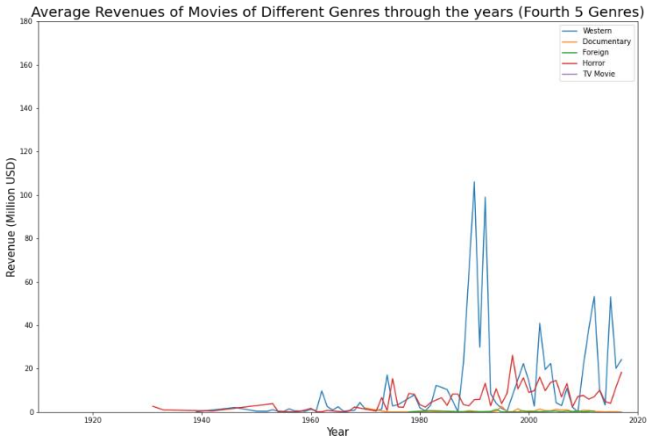


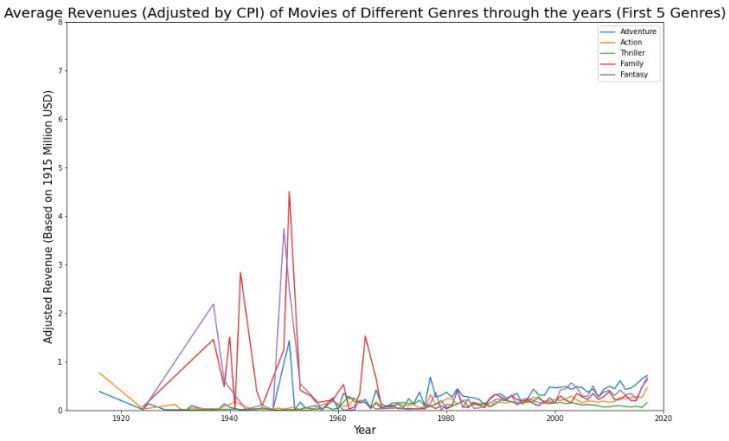


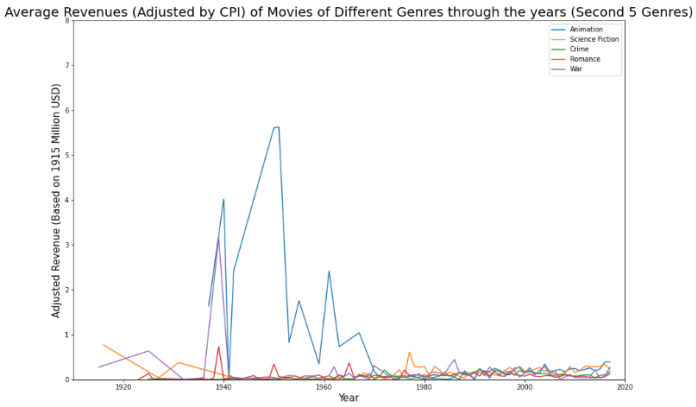
Here we can see there seems to be a spike of revenue of different films at around 1950. The revenue in USD is particularly high around 1950. For animation, fantasy and family genres, 1950 circa is a particular high era. This is pretty strange. I think this may be attributed to the value of USD being pretty low in that time.

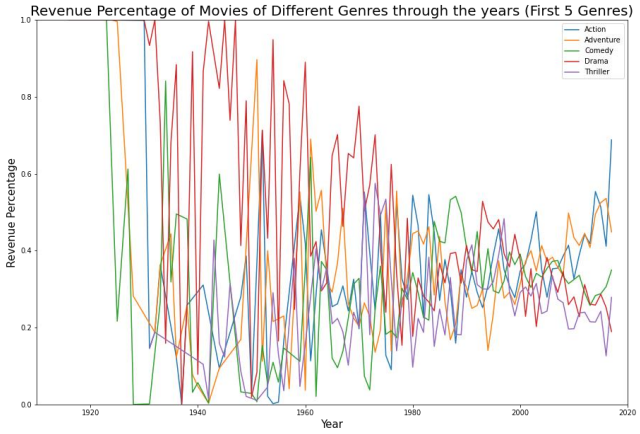
For western films, 1990s seems to be a golden era, with exceptional high revenues. Actually this surprises me, I thought the golden era of western films could be 1940s or earlier. This could be partially attributed to inflation.

After adjusted by CPI, we see a very different plot. Though the sheer USD of different genres went exponentially in the recent 4 decades, the real value of this revenues is actually a lot lower than the 30s, 40s and 50s. 1970s witnessed an exceptionally low movie revenues. Throughout the years, movie industry has been challenged by different new media and also helped by different new media, such as the popularization of television, the emergence of Netflix. New technology also helped the movie industry in recent decades.

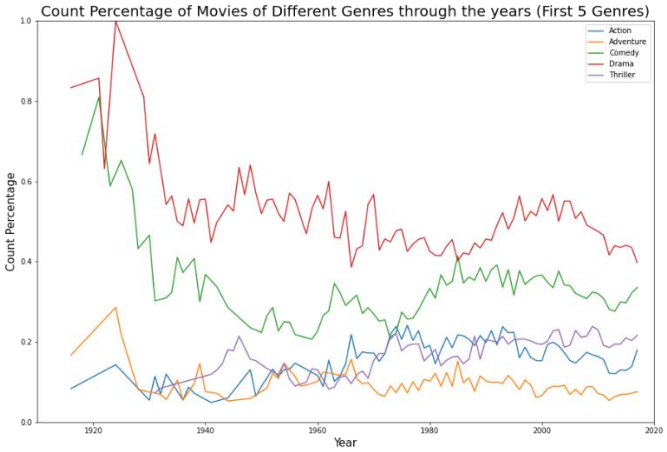


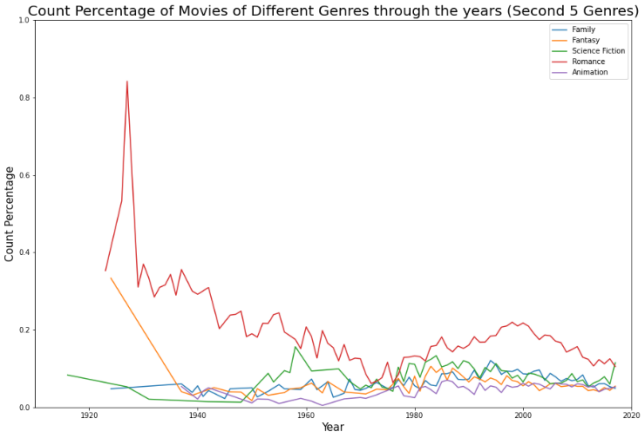




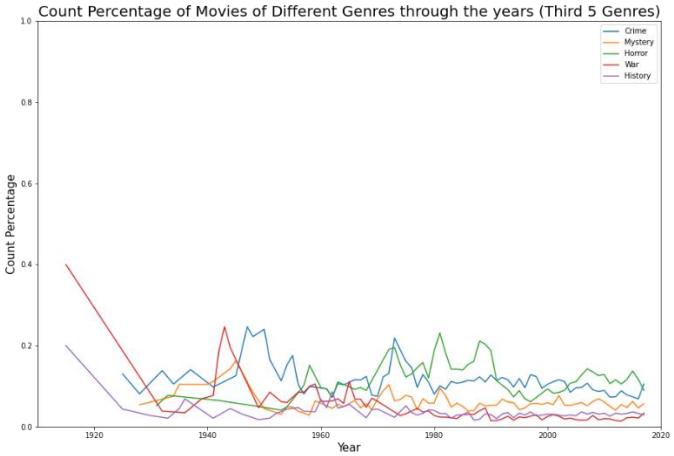


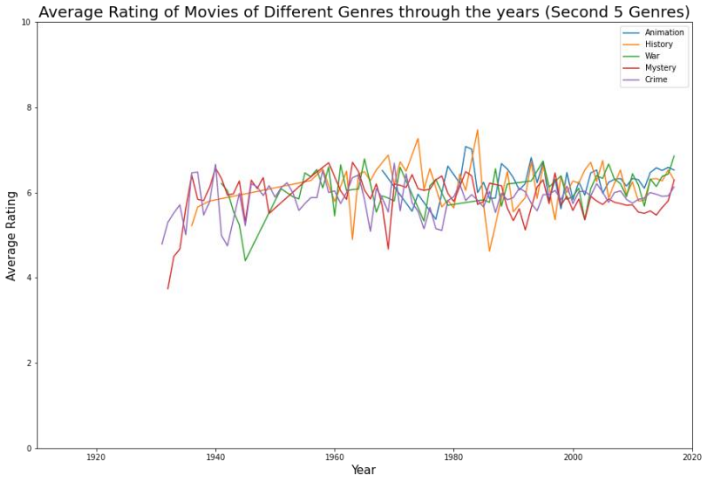
For the revenue share of different genres throughout the year, we saw a very chaotic plot here, looks like there is no obvious pattern, and the fluctuation is very large. Generally, we see drama genre have most portion of the revenue before 1980s and gradually declined in recent years. Action and adventure genres take more and more revenue shares in recent years.



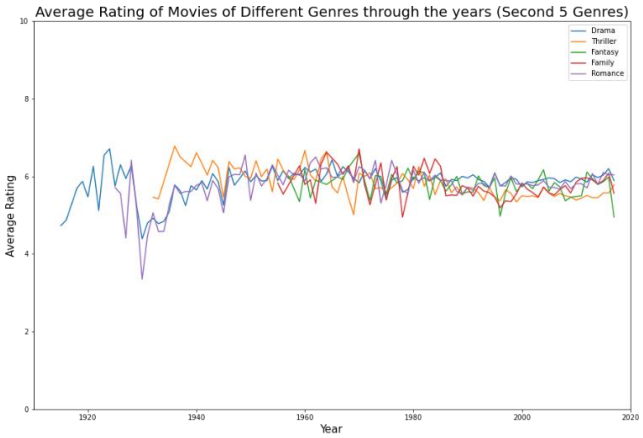


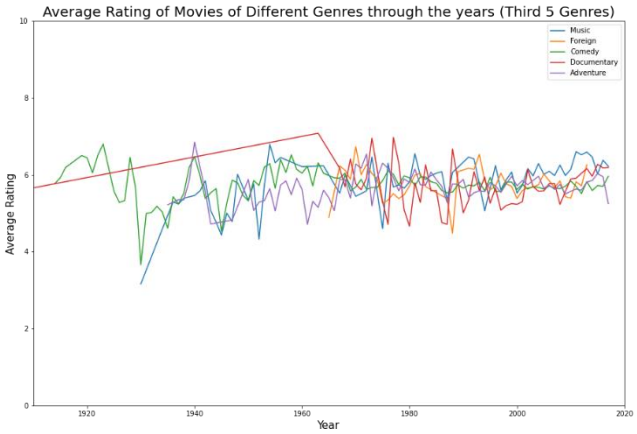
Drama and comedy are the most popular genres throughout the past 100 years, but their percentage in the whole movies are declining. Romance genre is also declining in the past 100 years.

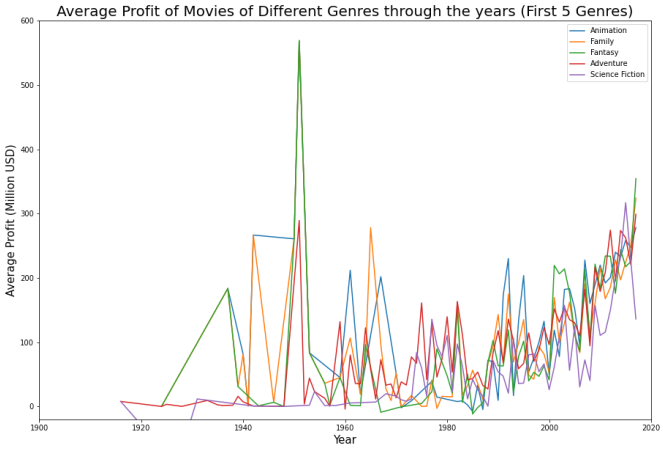


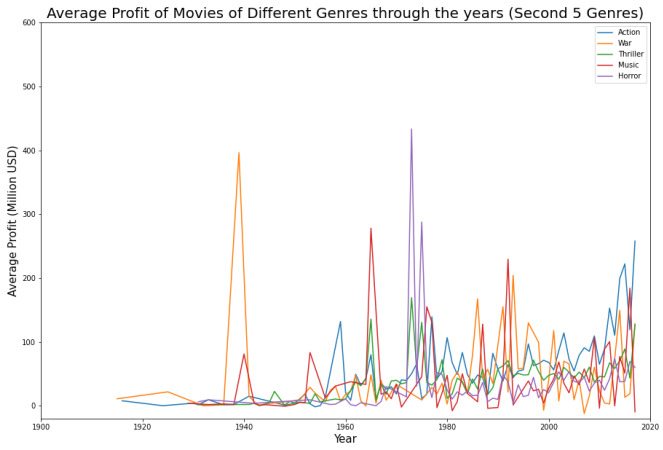


Though not obvious, the percentage of movies of history and war genres are also declining. And generally speaking, viewers seem to have lower ratings for older films and higher ratings for newer films.

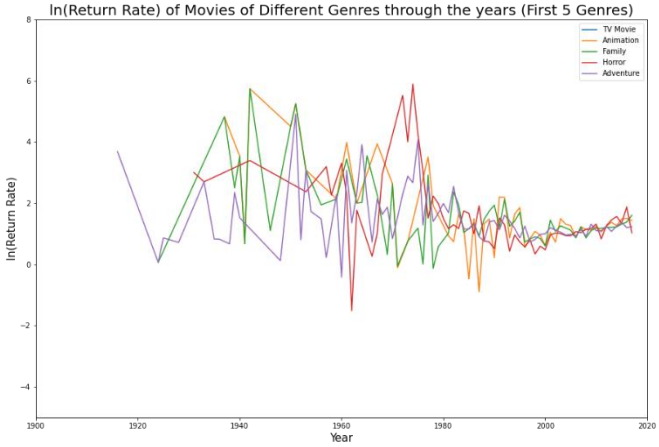


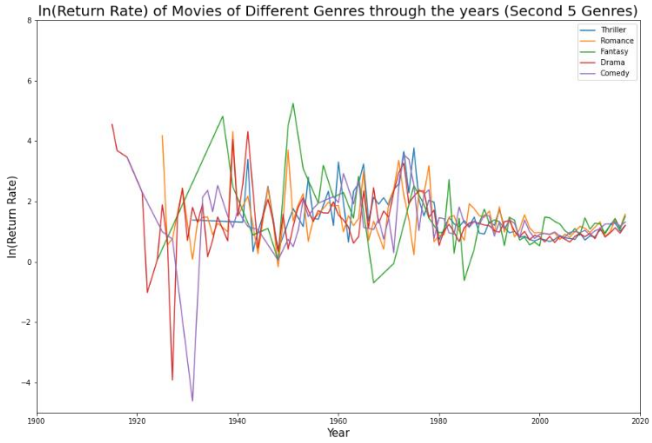






From these plots above we can see that in some years, the gross profit of certain kinds of movies is negative, which means they lose more money.





Here we can see that movies have the highest return rate are actually movies with small budget such as TV movies, etc.

Title	Release_Date	Budget	Revenue	Profit	Return_Rate
Less Than Zero	11/6/1987	1	12396383	12396382	1.2396383E7
Modern Times	2/5/1936	1	8500000	8499999	8500000.0
Aquí Entre Nos	3/30/2012	1	2755584	2755583	2755584.0
Nurse 3-D	9/28/2013	10	10000000	9999990	1000000.0
From Prada to Nada	1/28/2011	93	2500000	2499907	26881.720430107525
Paranormal Activity	9/14/2007	15000	193355800	193340800	12890.386666666667
Tarnation	10/19/2003	218	1162014	1161796	5330.339449541284
The Blair Witch P...	7/14/1999	60000	248000000	247940000	4133.333333333333
"口�?""While the K...	12/16/2015	5000	11083449	11078449	2216.6898
Eraserhead	3/19/1977	10000	7000000	6990000	700.0
猛龍過江	6/1/1972	130000	85000000	84870000	653.8461538461538
Pink Flamingos	3/12/1972	12000	6000000	5988000	500.0
Super Size Me	1/17/2004	65000	28575078	28510078	439.6165846153846
The Gallows	7/10/2015	100000	42664410	42564410	426.6441
Open Water	8/6/2004	130000	54667954	54537954	420.52272307692306
The Texas Chain S...	10/1/1974	85000	30859000	30774000	363.0470588235294
Bambi	8/14/1942	858000	267447150	266589150	311.709965034965
Night of the Livi...	10/1/1968	114000	30000000	29886000	263.1578947368421
Mad Max	4/12/1979	400000	100000000	99600000	250.0
Halloween	10/25/1978	300000	70000000	69700000	233.33333333333334

Movies ranked by Return Rate

Title	Release_Date	Budget	Revenue	Profit	Adjusted_Profit
Gone with the Wind	12/15/1939	4000000	400176459	396176459	2.82983185E7
Alice in Wonderland	7/3/1951	3000000	572000000	569000000	2.196911196911197E7
Bambi	8/14/1942	858000	267447150	266589150	1.6156918181818182E7
Snow White and th...	12/20/1937	1488423	184925486	183437063	1.2918103028169015E7
Star Wars	5/25/1977	11000000	775398007	764398007	1.2593047891268533E7
Cinderella	3/4/1950	2900000	263591415	260691415	1.1046246398305085E7
Titanic	11/18/1997	200000000	1845034188	1645034188	1.0198600049597023E7
The Exorcist	12/26/1973	8000000	441306145	433306145	9298415.128755365
The Sound of Music	3/2/1965	8200000	286214286	278014286	8882245.559105432
Jaws	6/18/1975	7000000	470654000	463654000	8554501.84501845
E.T. the Extra-Te...	4/3/1982	10500000	792965326	782465326	8245156.227608008
Star Wars: The Fo...	12/15/2015	245000000	2068223624	1823223624	7708375.960257901
One Hundred and O...	1/25/1961	4000000	215880014	211880014	7110067.583892617
The Empire Strike...	5/17/1980	18000000	538400000	520400000	6292623.9419588875
The Jungle Book	10/17/1967	4000000	205843612	201843612	5971704.497041421
Jurassic Park	6/11/1993	63000000	920100000	857100000	5935595.567867036
Pinocchio	2/23/1940	2600000	84300000	81700000	5835714.285714285
Fantasia	11/13/1940	2280000	83320000	81040000	5788571.428571428
The Godfather	3/14/1972	6000000	245066411	239066411	5774550.990338164
Jurassic World	6/9/2015	150000000	1513528810	1363528810	5713795.83301905
The Avengers	4/25/2012	220000000	1519557910	1299557910	5654800.208863651
Furious 7	4/1/2015	190000000	1506249360	1316249360	5563207.621333987
The Lord of the R...	12/1/2003	94000000	1118888979	1024888979	5560981.98046663
Return of the Jedi	5/23/1983	32350000	572700000	540350000	5430653.266331659
Harry Potter and ...	7/7/2011	125000000	1342000000	1217000000	5386814.918423173
The Lion King	6/23/1994	45000000	788241776	743241776	5008367.762803234
Star Wars: Episod...	5/19/1999	115000000	924317558	809317558	4869540.060168472
Frozen	11/27/2013	150000000	1274219009	1124219009	4823959.806735923
Harry Potter and ...	11/16/2001	125000000	976475550	851475550	4818763.723825693
Avengers: Age of ...	4/22/2015	280000000	1405403694	1125403694	4732464.388890057
Independence Day	6/25/1996	75000000	816969268	741969268	4725918.904458599
The Lord of the R...	12/18/2002	79000000	926287400	847287400	4663111.722619703
Finding Nemo	5/30/2003	94000000	940335536	846335536	4607161.328252586
Close Encounters ...	11/16/1977	20000000	303788635	283788635	4569865.297906602
Minions	6/17/2015	74000000	1156730962	1082730962	4536823.024126979
Beauty and the Beast	3/16/2017	160000000	1262886337	1102886337	4510339.831672965
Ben-Hur	12/26/1959	15000000	146900000	131900000	4501706.484641638
The Lord of the R...	12/18/2001	93000000	871368364	778368364	4395078.283455675

Movies ranked by Profit (Adjusted by CPI)

We can see that many movies of high return rate are not well-known, because they often have small budget, not the kind of blockbusters we all know. Some movies have budget of 1 USD, which is very confusing to me.

Many of the films that is most profitable are all very famous. Many of them are older films. The most profitable films in the history adjusted by inflation is actually Gone with the Wind. I already knew this years ago, and this calculation affirmed this fact.

Title	Release_Date	Budget	Revenue	Profit
Star Wars: The Fo...	12/15/2015	245000000	2068223624	1823223624
Titanic	11/18/1997	200000000	1845034188	1645034188
Jurassic World	6/9/2015	150000000	1513528810	1363528810
Furious 7	4/1/2015	190000000	1506249360	1316249360
The Avengers	4/25/2012	220000000	1519557910	1299557910
Harry Potter and ...	7/7/2011	125000000	1342000000	1217000000
Avengers: Age of ...	4/22/2015	280000000	1405403694	1125403694
Frozen	11/27/2013	150000000	1274219009	1124219009
Beauty and the Beast	3/16/2017	160000000	1262886337	1102886337
Minions	6/17/2015	74000000	1156730962	1082730962
The Lord of the R...	12/1/2003	94000000	1118888979	1024888979
Iron Man 3	4/18/2013	200000000	1215439994	1015439994
The Fate of the F...	4/12/2017	250000000	1238764765	988764765
Despicable Me 3	6/15/2017	80000000	1020063384	940063384
Transformers: Dar...	6/28/2011	195000000	1123746996	928746996
Skyfall	10/25/2012	200000000	1108561013	908561013
Captain America: ...	4/27/2016	250000000	1153304495	903304495
Despicable Me 2	6/25/2013	76000000	970761885	894761885
Transformers: Age...	6/25/2014	210000000	1091405097	881405097
Zootopia	2/11/2016	150000000	1023784195	873784195
Toy Story 3	6/16/2010	200000000	1066969703	866969703
Pirates of the Ca...	6/20/2006	200000000	1065659812	865659812
Jurassic Park	6/11/1993	63000000	920100000	857100000
Rogue One: A Star...	12/14/2016	200000000	1056057273	856057273
Harry Potter and ...	11/16/2001	125000000	976475550	851475550
The Lord of the R...	12/18/2002	79000000	926287400	847287400
Finding Nemo	5/30/2003	94000000	940335536	846335536
The Dark Knight R...	7/16/2012	250000000	1084939099	834939099
Finding Dory	6/16/2016	200000000	1028570889	828570889

Movies ranked by Profit

Title	Release_Date	Budget	Revenue	Profit	Adjusted_Profit
Avatar	12/10/2009	237000000	null	null	null
Metropolis	1/10/1927	92620000	650422	-91969578	-5255404.457142857
The Lone Ranger	7/3/2013	255000000	89289910	-165710090	-709387.5323207589
The Alamo	4/7/2004	145000000	25819961	-119180039	-633936.3776595745
The 13th Warrior	8/27/1999	160000000	61698899	-98301101	-585474.0976771888
Cutthroat Island	12/22/1995	98000000	10017322	-87982678	-569835.9974093264
Waterloo	10/26/1970	25000000	3052000	-21948000	-554242.4242424242
The Adventures of...	8/15/2002	100000000	7103973	-92896027	-514089.80077476485
Mars Needs Moms	3/9/2011	150000000	38992758	-111007242	-496750.0436306031
Heaven's Gate	11/19/1980	44000000	3484331	-40515669	-469474.7276940904
The Fall of the R...	3/24/1964	19000000	4750000	-14250000	-461165.04854368937
Town & Country	4/27/2001	90000000	10372291	-79627709	-448101.907709623
Supernova	1/14/2000	90000000	14828081	-75171919	-445331.2736966824
Valerian and the ...	7/20/2017	197471676	90024292	-107447384	-437633.6821182882
Flushed Away	10/22/2006	149000000	64459316	-84540684	-419556.74441687344
Monkeybone	2/23/2001	75000000	5409517	-69590483	-394951.6628830874
The Postman	12/25/1997	80000000	17626234	-62373766	-385976.27475247526
Australia	11/18/2008	130000000	49554002	-80445998	-382660.72074129037
Sphere	2/13/1998	75000000	13100000	-61900000	-382334.7745521927
A Sound of Thunder	5/15/2005	80000000	5989640	-74010360	-380711.72839506174
Lolita	9/27/1997	62000000	1060056	-60939944	-377103.61386138614
Virus	1/14/1999	75000000	14010690	-60989310	-371206.99939135724
Soldier	10/23/1998	75000000	14567883	-60432117	-368488.51829268294
Ishtar	5/15/1987	55000000	14375181	-40624819	-359193.8019451813
K-19: The Widowmaker	7/19/2002	100000000	35168966	-64831034	-358777.1665744328
The Great Raid	8/12/2005	80000000	10166502	-69833498	-355567.7087576375
Pirates	5/8/1986	40000000	1641825	-38358175	-352233.01193755737
Live by Night	12/25/2016	108000000	22678555	-85321445	-351349.8449590058
Hudson Hawk	5/23/1991	65000000	17218080	-47781920	-351337.64705882355
Chill Factor	9/1/1999	70000000	11263966	-58736034	-349827.48064324004

Movies ranked as the least profitable (Adjusted by CPI)

By sheer profit, the most profitable films include Titanic and many different blockbusters, most of them are recent films.

In the least profitable dataframe, Avatar must be a mistake as the revenue is empty in the dataset. Therefore, the least profitable movie in history would be Metropolis, an extremely old German film.

Overall, although movie revenues are increasing with the years, the importance of the movie industry is actually decreasing if you consider inflation, and people have more ways to be entertained. In the 1940s, movies were almost the only form of entertainment, which is why Gone with the Wind is the highest-grossing film in history when inflation is taken into account. As the times progressed, people became more and more interested in movies that were thrilling, such as adventure and action, and less and less interested in historical films.

I think the things that didn't work is my results about different languages of movies. I planned to do analysis of different languages, but the results are not ideal enough for me to do so. So I ended up forgoing all this despite the effort I have already made. Because most movies in this dataset are in English. For movies other than English, the data is incomplete and insufficient. The results would have a very high deviation and therefore meaningless.

5. Challenges

The first difficulty I encountered was in the mapreduce part, reading a very complex csv file. It took me a long time to figure out how to read it correctly because the csv file had commas between quotes and these commas were not delimiters. My solution was mainly to google it and experiment on my own. The second difficulty I encountered was in the SparkSQL section, where the dataframe I ended up with kept failing to ORDER BY, and kept showing errors. I checked on Google and found only one problem like mine. I have a feeling that there may be a problem with the datatype, but I have not been able to solve it, I finally bypassed the difficulty and did some pre-processing in Excel and finally managed to create the result.