Developing a Data handling pipeline with the aid of Big Data SQL Tools

Arinjay Jain (A20447307)

Ayush Dadhich (A20449379)

Keerthana N (A20443972)

Parth Gupta(A20449774)

CSP 554 Big Data Technologies

Prof. Adam McElhinney

Introduction

The entertainment industry is a booming and successful industry with billions of dollars going into it. A major chunk of this industry is the Movie industry, with an average of 700 films being released a year and the global box office making $41.7 billion in 2018 alone. Such a huge industry could benefit from knowing what aspects of a movie make it successful and more appealing to the audience. We wanted to study the various factors that go into making a movie and how much of an effect they have on the revenue generated by the movie, if at all.

On researching this topic, we found that most papers have worked on analyzing the reviews of movies and their sentiments to compare their ratings with the success of a movie. There were a few papers that discussed analyzing the pre-release information of a movie, predicting it's success and comparing it with the post-release movie data to see how successful their model was. Our main focus was to analyze every available movie release feature and see which ones have the most impact on the revenue of the movie. To do so, we had to find large enough datasets of movie information and a technology to handle this big data.

When it came to choosing the Big Data Technologies for this project, we decided to go with the most diverse framework, Spark, for our implementation. To implement our database, we looked through the most popular NoSQL databases Cassandra, MongoDB and HBase. On further study, we found that there have not been many implementations of HBase using Spark. We could only find one paper [1] where the authors implemented HBase using Spark. Since this was not well established ground, we decided to try and implement this. Since HBase uses Hadoop Distributed File System (HDFS) as its file system, we were able to take advantage of the distributed nature of data storage in HBase. We also used Tableau for the visualizations in our project to portray our findings in a more understandable and easy to follow manner.
Since implementing HBase using Spark is a relatively untrodden path, most of our focus in this project was devoted to implementing this.

Related Work

We researched six papers while preparing for this project.

a. Mining Chinese social media UGC: a big-data framework for analyzing Douban movie reviews [2]
This paper addresses common Big Data problems of time constraints and memory costs involved with using standard single-machine and software. The authors propose a novel big data processing framework to investigate a niche subset of user-generated popular culture content on Douban, a well-known Chinese-language online social network. Huge data samples were harvested via an asynchronous scraping crawler and the rest of the research was implemented on big data technologies. Their major contributions were:
i. An efficient framework implemented for large volumes of social media data processing based on the Hadoop platform. User-generated contents were collected, distributed, stored and processed on the Hadoop distributed file system (HDFS)
ii. An asynchronous scraping crawler was implemented via a multiple-task queue to collect data in an efficient and simultaneous manner.
iii. A novel extraction, transformation and load (ETL) process was introduced
Iv. An improved Apriori algorithm based on MapReduce was proposed to increase the flexibility and efficiency of Big Data Mining.

The authors conclude that the proposed framework offers a flexible capability and efficient applicability for the processing of large amounts of social media data that in turn can be fed back to producers and distributors of both commercial and user-generated digital media contents.

b. Scalable sentiment classification for Big Data analysis using Naive Bayes Classifier [3]

This paper evaluates the scalability of Naive Bayes classifier in large datasets to achieve fine-grained control of the analysis procedure. A Big Data analyzing system was also designed to help this study. A standard approach is to use Mahout, a machine learning library for clustering, classification and filtering and is implemented on top of Hadoop. The authors compare the performance of Naive Bayes with the implementation using Mahout and built a big data analyzing system. This system adds four modules on Hadoop: work flow controller, data parser, the user terminal and the result collector. Each of these modules were implemented using MapReduce frameworks and data transfer to and from HDFS. The model was implemented on Virtual Hadoop cluster to enable testing of the Hadoop program in the cloud. The authors conclude that as data increases, the Naive Bayes Classifier model has 82% accuracy. The Hadoop implemented model shows faster performance and lesser processing time as the amount of data used crosses 2000K cases.

c. Movie Popularity Classification based on Inherent Movie Attributes using C4.5, PART and Correlation Coefficient [4]

This paper talks about the surplus research being done on classifying movies for future selection based on the attributes of already released movies. The authors mention that much can be predicted by considering parameters such as actors, directors, languages, countries, etc. and is an aspect of movie data that can be taken advantage of in prediction how a movie would perform. In this paper, the authors propose to analyse details of movies prior to their release and predict the success, revenue, ratings, etc. of a movie and compare it to the same post-release. This would be greatly useful to producers, financiers, academics and even viewers to understand the contributing factors that lead to a movie's success. The objective of this paper was to provide a suitable approach along with the necessary factors that were to be considered for developing pre-release and post-release movie datasets using Internet Movie Database (IMDB), classify data and interpret future predictions. This was accomplished using tools JMDB, SQL, WEKA, PART and Decision Trees. The results showed that decision trees perform well in prediction, directors and budget together play an important role in the success of a movie and the correlation between budget and foreign revenues is significantly high.

d. **A Review Paper on Big Data: Technologies, Tools and Trends [5]**

This paper talks about the rapid increase in generation of data and increase in internet population, thus explaining the need for Big Data Technologies. This paper covers the history of Big Data, Definition, Characteristics and Generation of Big Data. This paper also mentions the various categories of Big Data, its Management and details the major tools of Big Data: Hadoop, HDFS, MapReduce Frameworks, YARN, Hbase, Pig, ZooKeeper, HCatalog, Hive, Mahout, Oozie, Kafka, Spark and many more. The authors conclude by mentioning the applications of Big Data and the various analyses that can be done on Big Data.

e. Early Prediction of Movie Box Office Success Based on Wikipedia Activity Big Data [6]

This paper aims to bridge the gap between 'real time monitoring' and 'early predicting' by building a minimalistic predictive model for the financial success of movies based on collective activity data of online users. The authors show that the popularity of a movie can be predicted much before its release by measuring and analyzing the activity level of editors and viewers of the corresponding entry to the movie in Wikipedia and Twitter. THe tools used in this research were Wikipedia, Toolserver, Mojo, Bots and used a 10-fold cross-validation to validate the multivariate linear regression model. They concluded that their model works more accurately for movies that are more popular and the volume of the related data is larger. They also concluded that most of the movies predicted by the Twitter method were among the successful ones.

f. **An implementation of a high throughput data ingestion system for machine logs in manufacturing industry [1]**

The authors of this paper presented a case study for designing and implementing a data ingestion system for manufacturers. Though this paper has nothing to do with movie data analysis, this paper was one of the few papers we found where the authors documented using HBase from Spark. They leveraged the power of open source frameworks like Apache Kafka, Apache Hadoop File System, Apache Flume and HBase.

Comparing HBase with Cassandra and MongoDB [8]

Before explaining our methodology, we would like to justify our choice of choosing HBase over the other NoSQL Database options for this project. Enumerated below are the advantages and disadvantages of each of the three NoSQL Databases.

Table 1

Cassandra	MongoDB	HBase
Advantages ● High scalability ● High availability due to lack of single point of failure ● Real-time analysis ● Durable ● Fault-tolerant	Advantages ● Schema-less database, stores data as documents ● Faster querying through indexes ● High availability and scalability	Advantages ● In-memory operation ● Uses HDFS as the distributed file system, can therefore handle billions of rows ● Since the data is distributed, it is cost-effective ● Follows immediate consistency ● Schema-less, only defines column families ● High availability and scalability
Disadvantages ● Inconsistency due to distributed architecture ● Depends on primary key to scan data. This leads to high read time penalties if primary keys are not known ● Lack of solid official documentation	Disadvantages ● Management operations are manual and time-consuming processes ● MapReduce implementations still remain a slow process ● Has memory hog issues when scaling up	Disadvantages ● Has a master-slave architecture and can be a single point of failure ● Does not have a query language ● Dependency on other systems like HDFS and ZooKeeper, making its architecture complex

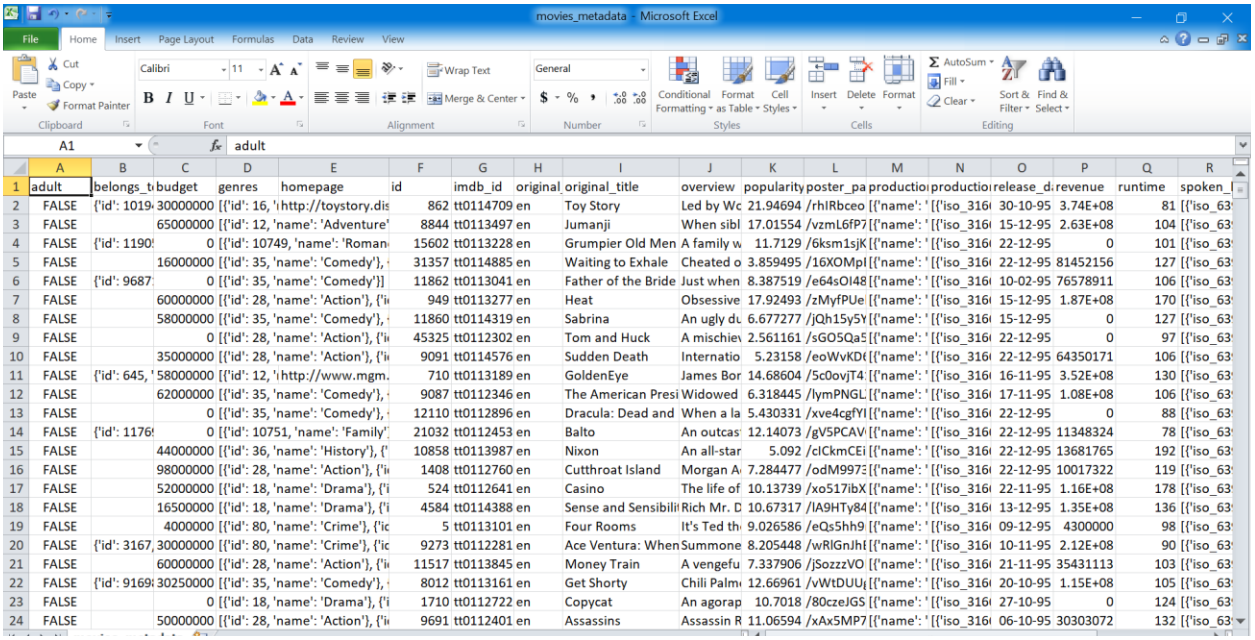
On further studying the various NoSQL Databases available, HBase was chosen for this project:

- It seemed most applicable to our project
- Not many papers or projects are available where HBase and Spark were used
- Since HBase uses HDFS as its file system, it is great for huge amounts of records
- Uses MapReduce in its job execution

Methodology

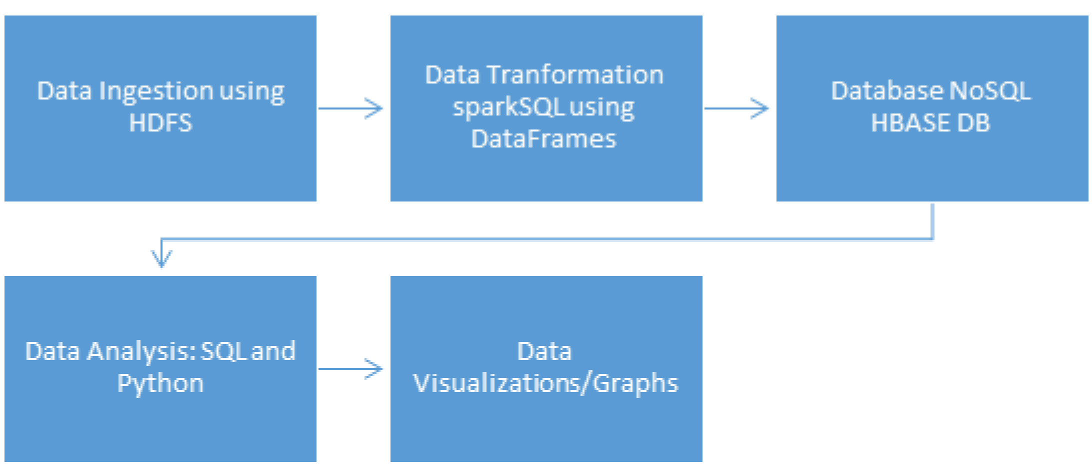
Dataset [7]

The dataset was taken from Kaggle. It has multiple files with reviews, ratings, actors, directors, etc. The file **movies_metadata.csv** was chosen to use for the analysis as it had the most data and was well organized.



	adult	belongs_t	budget	genres	homepage	id	imdb_id	original_	original_title	overview	popularity	poster_pa	productio	productio	release_d	revenue	runtime	spoken_
2	FALSE	{'id': 1019	30000000	[{'id': 16, '	http://toystory.dis	862	tt0114709	en	Toy Story	Led by Wc	21.94694	/rhIRbceo	[{'name': '	[{'iso_316	30-10-95	3.74E+08	81	[{'iso_63
3	FALSE		65000000	[{'id': 12, 'name': 'Adventure'		8844	tt0113497	en	Jumanji	When sibl	17.01554	/vzmL6fP7	[{'name': '	[{'iso_316	15-12-95	2.63E+08	104	[{'iso_63
4	FALSE	{'id': 1190	0	[{'id': 10749, 'name': 'Roman		15602	tt0113228	en	Grumpier Old Men	A family w	11.7129	/6ksm1sjK	[{'name': '	[{'iso_316	22-12-95	0	101	[{'iso_63
5	FALSE		16000000	[{'id': 35, 'name': 'Comedy'},		31357	tt0114885	en	Waiting to Exhale	Cheated o	3.859495	/16XOMpI	[{'name': '	[{'iso_316	22-12-95	81452156	127	[{'iso_63
6	FALSE	{'id': 9687	0	[{'id': 35, 'name': 'Comedy'}]		11862	tt0113041	en	Father of the Bride	Just when	8.387519	/e64sOI48	[{'name': '	[{'iso_316	10-02-95	76578911	106	[{'iso_63
7	FALSE		60000000	[{'id': 28, 'name': 'Action'}, {'i		949	tt0113277	en	Heat	Obsessive	17.92493	/zMyfPUe	[{'name': '	[{'iso_316	15-12-95	1.87E+08	170	[{'iso_63
8	FALSE		58000000	[{'id': 35, 'name': 'Comedy'},		11860	tt0114319	en	Sabrina	An ugly du	6.677277	/jQh15y5Y	[{'name': '	[{'iso_316	15-12-95	0	127	[{'iso_63
9	FALSE		0	[{'id': 28, 'name': 'Action'}, {'i		45325	tt0112302	en	Tom and Huck	A mischiev	2.561161	/sGO5Qa5	[{'name': '	[{'iso_316	22-12-95	0	97	[{'iso_63
10	FALSE		35000000	[{'id': 28, 'name': 'Action'}, {'i		9091	tt0114576	en	Sudden Death	Internatio	5.23158	/eoWvKD	[{'name': '	[{'iso_316	22-12-95	64350171	106	[{'iso_63
11	FALSE	{'id': 645, '	58000000	[{'id': 12, '	http://www.mgm.	710	tt0113189	en	GoldenEye	James Bor	14.68604	/5c0ovjT4	[{'name': '	[{'iso_316	16-11-95	3.52E+08	130	[{'iso_63
12	FALSE		62000000	[{'id': 35, 'name': 'Comedy'},		9087	tt0112346	en	The American Presi	Widowed	6.318445	/lymPNGL	[{'name': '	[{'iso_316	17-11-95	1.08E+08	106	[{'iso_63
13	FALSE		0	[{'id': 35, 'name': 'Comedy'},		12110	tt0112896	en	Dracula: Dead and	When a la	5.430331	/xve4cgfYI	[{'name': '	[{'iso_316	22-12-95	0	88	[{'iso_63
14	FALSE	{'id': 1176	0	[{'id': 10751, 'name': 'Family'		21032	tt0112453	en	Balto	An outcas	12.14073	/gV5PCAV	[{'name': '	[{'iso_316	22-12-95	11348324	78	[{'iso_63
15	FALSE		44000000	[{'id': 36, 'name': 'History'}, {'		10858	tt0113987	en	Nixon	An all-star	5.092	/cICkmCEi	[{'name': '	[{'iso_316	22-12-95	13681765	192	[{'iso_63
16	FALSE		98000000	[{'id': 28, 'name': 'Action'}, {'i		1408	tt0112760	en	Cutthroat Island	Morgan A	7.284477	/odM9973	[{'name': '	[{'iso_316	22-12-95	10017322	119	[{'iso_63
17	FALSE		52000000	[{'id': 18, 'name': 'Drama'}, {'i		524	tt0112641	en	Casino	The life of	10.13739	/xo517ibX	[{'name': '	[{'iso_316	22-11-95	1.16E+08	178	[{'iso_63
18	FALSE		16500000	[{'id': 18, 'name': 'Drama'}, {'i		4584	tt0114388	en	Sense and Sensibili	Rich Mr. D	10.67317	/lA9HTy84	[{'name': '	[{'iso_316	13-12-95	1.35E+08	136	[{'iso_63
19	FALSE		4000000	[{'id': 80, 'name': 'Crime'}, {'ic		5	tt0113101	en	Four Rooms	It's Ted th	9.026586	/eQs5hh9	[{'name': '	[{'iso_316	09-12-95	4300000	98	[{'iso_63
20	FALSE	{'id': 3167,	30000000	[{'id': 80, 'name': 'Crime'}, {'ic		9273	tt0112281	en	Ace Ventura: When	Summone	8.205448	/wRlGnJh	[{'name': '	[{'iso_316	10-11-95	2.12E+08	90	[{'iso_63
21	FALSE		60000000	[{'id': 28, 'name': 'Action'}, {'i		11517	tt0113845	en	Money Train	A vengefu	7.337906	/jSozzzVO	[{'name': '	[{'iso_316	21-11-95	35431113	103	[{'iso_63
22	FALSE	{'id': 9169	30250000	[{'id': 35, 'name': 'Comedy'},		8012	tt0113161	en	Get Shorty	Chili Palm	12.66961	/vWtDUU	[{'name': '	[{'iso_316	20-10-95	1.15E+08	105	[{'iso_63
23	FALSE		0	[{'id': 18, 'name': 'Drama'}, {'i		1710	tt0112722	en	Copycat	An agorap	10.7018	/80czeJGS	[{'name': '	[{'iso_316	27-10-95	0	124	[{'iso_63
24	FALSE		50000000	[{'id': 28, 'name': 'Action'}, {'i		9691	tt0112401	en	Assassins	Assassin R	11.06594	/xAx5MP7	[{'name': '	[{'iso_316	06-10-95	30303072	132	[{'iso_63

Project Flow Diagram



Data Cleaning

However, there were issues with reading in the data as the storage format of the data was difficult to query. There were also problems with reading the data from HBase if we imputed it as it was. We cleaned the dataset to make it easier to query.

The data was cleaned by using **pre_processing.sh** file, it handles null data sets and then imputed into HDFS from where the data was pushed to the HBase database

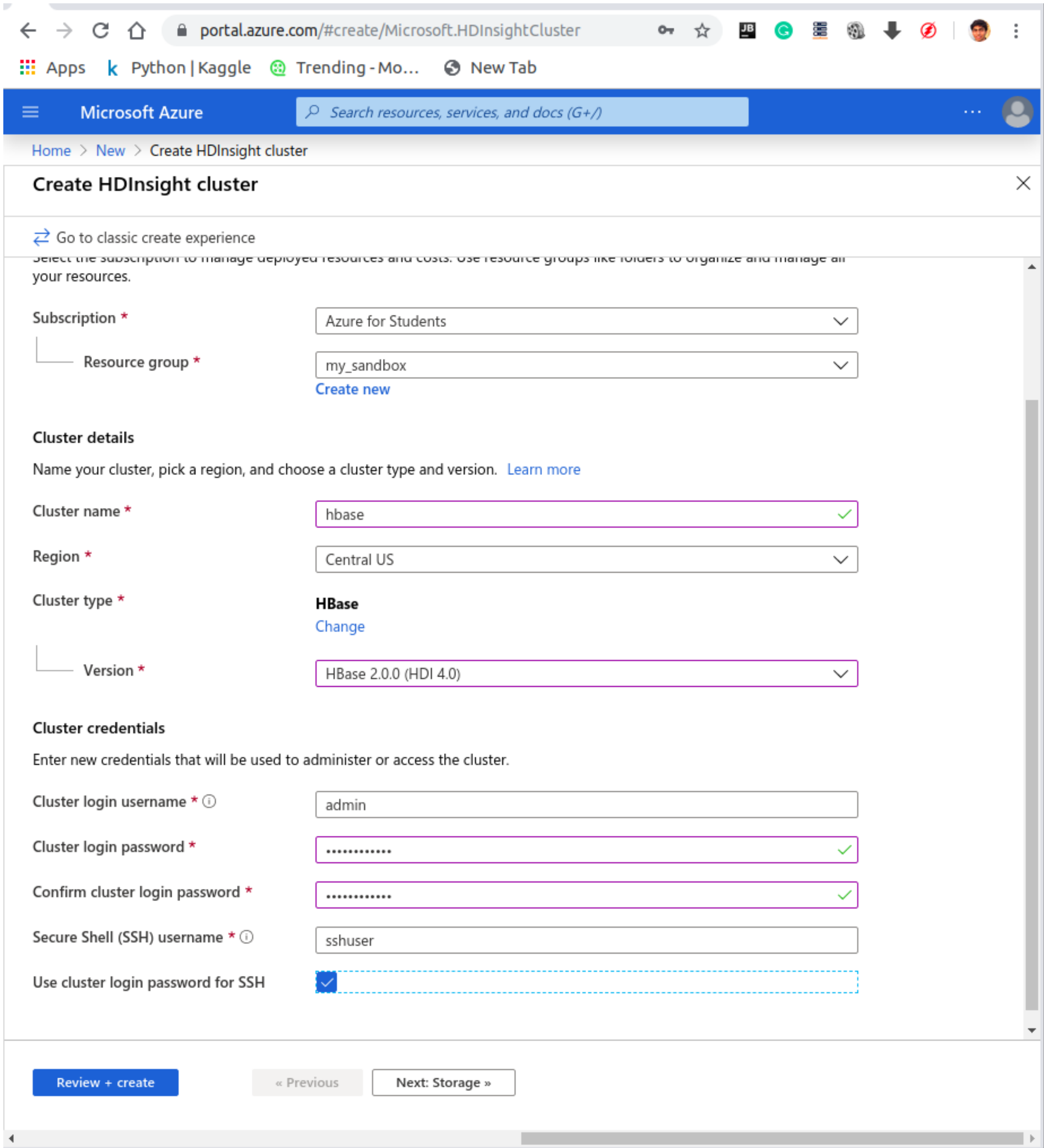
Data Statistics

The data was moderately distributed with a large enough number of rows (~45,467 rows). The dataset however did have an outlier and a high influential point. The movie 'Avatar' was released in 2009 and has the highest revenue of all time. This point does make the graphs in our analysis skewed and may not be the best movie to consider in our analysis of future movies. However, we have not excluded this point in our analysis as it does bring information of high budget and high revenue movies.

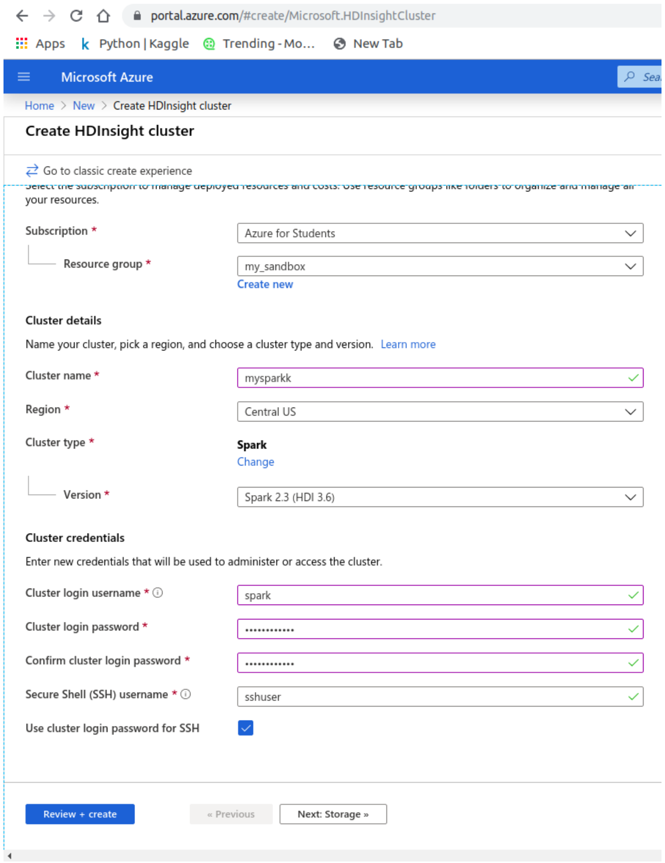
Methodology

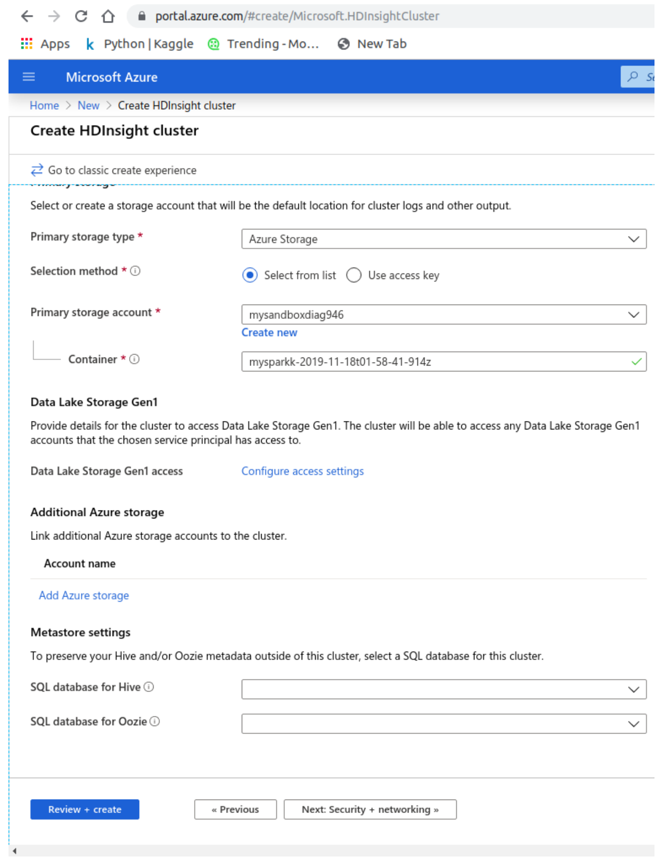
The methodology of our project can be explained under the following:

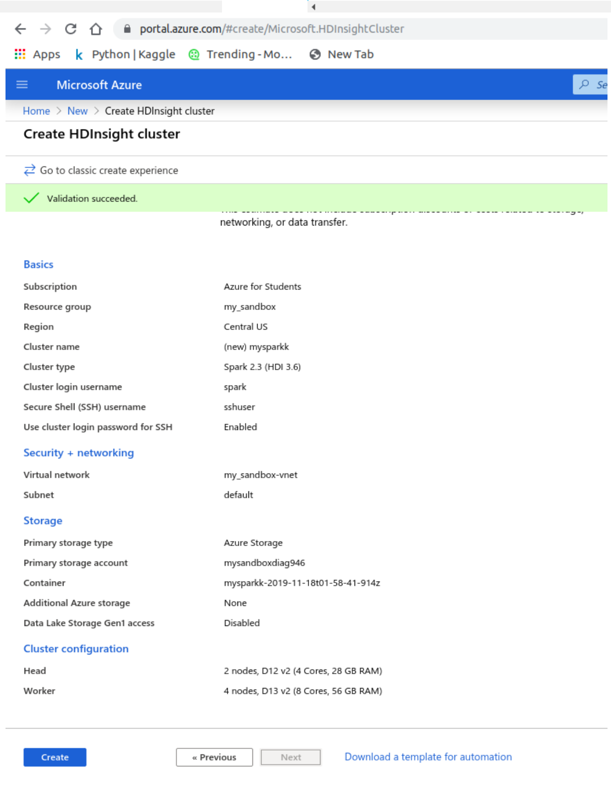
- Creating a cluster for HBase
  We first create an HBase cluster where the data will be moved for storage. This data will be accessed later for our querying and analysis.



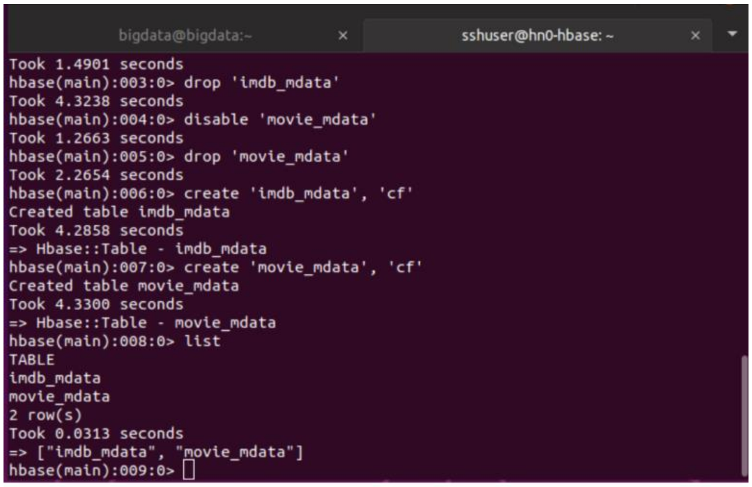
- Creating a cluster for Spark
  We next create a cluster for Spark. A node from this cluster will be used to access data from the HBase cluster.

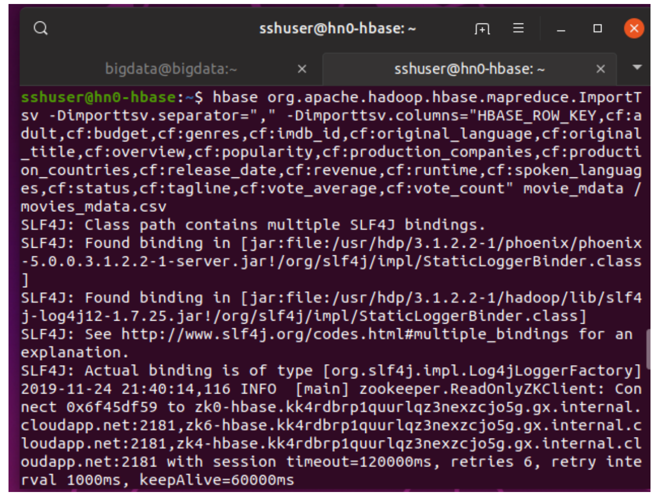


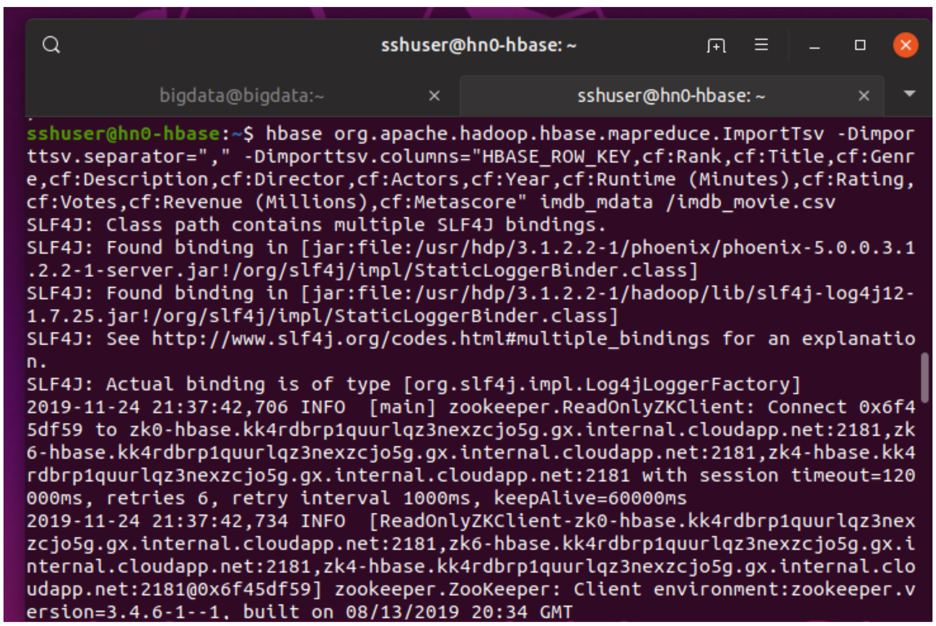




- Create tables in HBase
  Tables were created in HBase to read and store the movie data from the datasets.

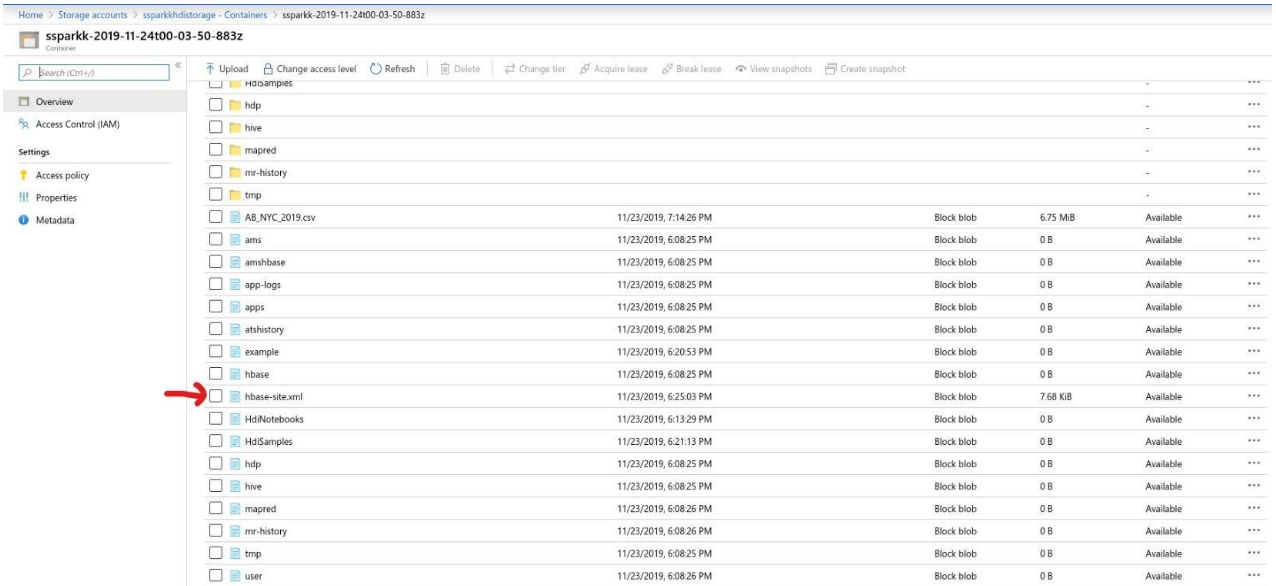






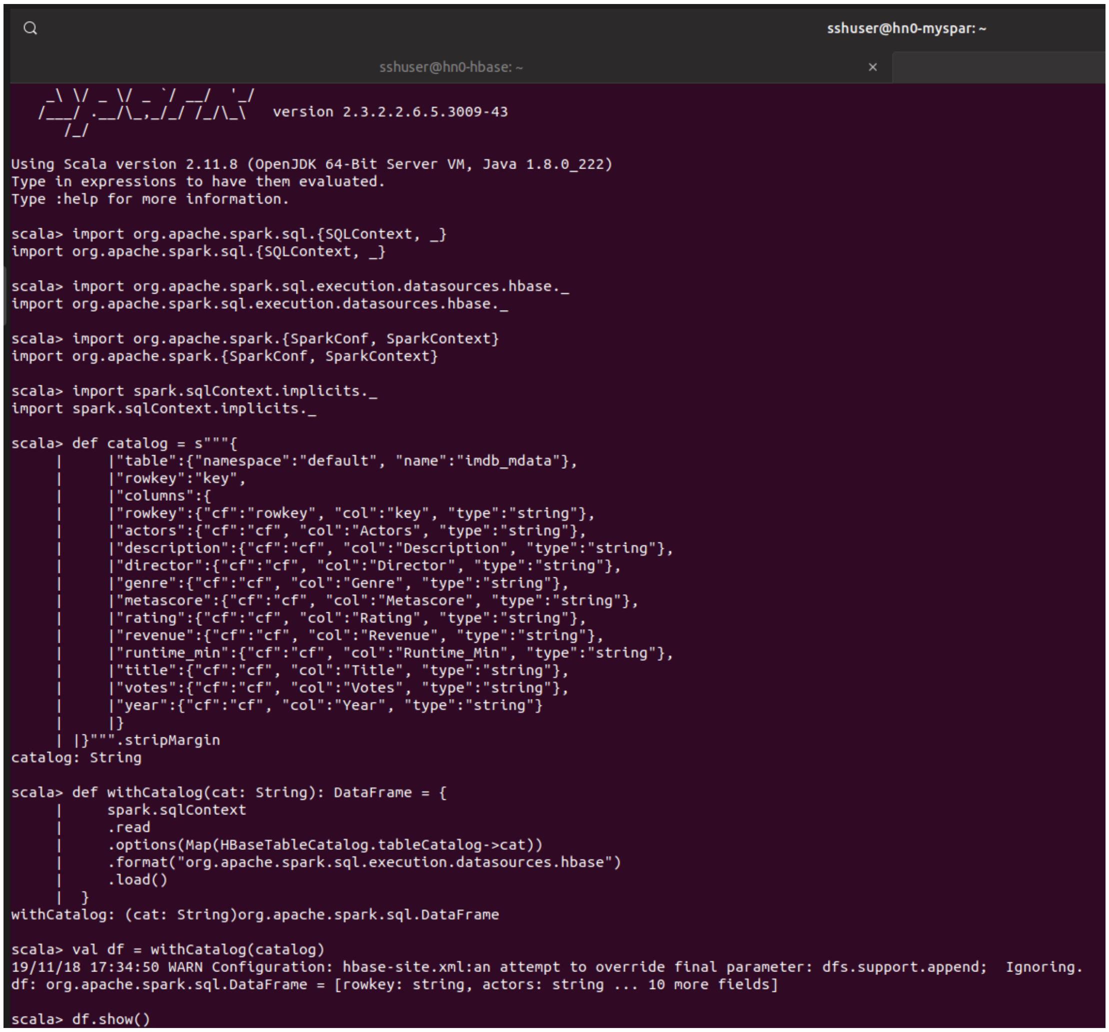
- Connecting the two clusters

We then connect the two clusters to enable communication between them by copying the file **hbase-site.xml** from the HBase cluster to the Spark cluster.



- Accessing HBase through the Spark cluster

We then setup the connection to the HBase cluster from the Spark cluster. The tables were then created and accessed as shown in the screenshots below.



```
sshuser@hn0-myspar: ~
sshuser@hn0-hbase: ~
   _\ \/ _ \/ _ `/ __/  '_/
  /___/ .__/\_,_/_/ /_/\_\   version 2.3.2.2.6.5.3009-43
     /_/

Using Scala version 2.11.8 (OpenJDK 64-Bit Server VM, Java 1.8.0_222)
Type in expressions to have them evaluated.
Type :help for more information.

scala> import org.apache.spark.sql.{SQLContext, _}
import org.apache.spark.sql.{SQLContext, _}

scala> import org.apache.spark.sql.execution.datasources.hbase._
import org.apache.spark.sql.execution.datasources.hbase._

scala> import org.apache.spark.{SparkConf, SparkContext}
import org.apache.spark.{SparkConf, SparkContext}

scala> import spark.sqlContext.implicits._
import spark.sqlContext.implicits._

scala> def catalog = s"""{
     |      |"table":{"namespace":"default", "name":"imdb_mdata"},
     |      |"rowkey":"key",
     |      |"columns":{
     |      |"rowkey":{"cf":"rowkey", "col":"key", "type":"string"},
     |      |"actors":{"cf":"cf", "col":"Actors", "type":"string"},
     |      |"description":{"cf":"cf", "col":"Description", "type":"string"},
     |      |"director":{"cf":"cf", "col":"Director", "type":"string"},
     |      |"genre":{"cf":"cf", "col":"Genre", "type":"string"},
     |      |"metascore":{"cf":"cf", "col":"Metascore", "type":"string"},
     |      |"rating":{"cf":"cf", "col":"Rating", "type":"string"},
     |      |"revenue":{"cf":"cf", "col":"Revenue", "type":"string"},
     |      |"runtime_min":{"cf":"cf", "col":"Runtime_Min", "type":"string"},
     |      |"title":{"cf":"cf", "col":"Title", "type":"string"},
     |      |"votes":{"cf":"cf", "col":"Votes", "type":"string"},
     |      |"year":{"cf":"cf", "col":"Year", "type":"string"}
     |      |}
     | |}""".stripMargin
catalog: String

scala> def withCatalog(cat: String): DataFrame = {
     |      spark.sqlContext
     |      .read
     |      .options(Map(HBaseTableCatalog.tableCatalog->cat))
     |      .format("org.apache.spark.sql.execution.datasources.hbase")
     |      .load()
     |   }
withCatalog: (cat: String)org.apache.spark.sql.DataFrame

scala> val df = withCatalog(catalog)
19/11/18 17:34:50 WARN Configuration: hbase-site.xml:an attempt to override final parameter: dfs.support.append;  Ignoring.
df: org.apache.spark.sql.DataFrame = [rowkey: string, actors: string ... 10 more fields]

scala> df.show()
```

```
sshuser@hn0-myspar: ~
sshuser@hn0-hbase: ~                    sshuser@hn0-myspar: ~
19/11/18 17:36:26 WARN Configuration: hbase-site.xml:an attempt to override final parameter: dfs.support.append;  Ignoring.
19/11/18 17:36:27 WARN Configuration: hbase-site.xml:an attempt to override final parameter: dfs.support.append;  Ignoring.
+------+--------------------+--------------------+-----------------+--------------------+---------+------+-------+-----------+--------------------+------+----+
|rowkey|              actors|         description|         director|               genre|metascore|rating|revenue|runtime_min|               title| votes|year|
+------+--------------------+--------------------+-----------------+--------------------+---------+------+-------+-----------+--------------------+------+----+
|     1|Chris Pratt; Vin ...|A group of interg...|       James Gunn|Action;Adventure;...|       76|   8.1| 333.13|        121|Guardians of the ...|757074|2014|
|    10|Jennifer Lawrence...|A spacecraft trav...|    Morten Tyldum|Adventure;Drama;R...|       41|     7| 100.01|        116|          Passengers|192177|2016|
|   100|Leonardo DiCaprio...|An undercover cop...|  Martin Scorsese|Crime;Drama;Thriller|       85|   8.5| 132.37|        151|        The Departed|937414|2006|
|  1000|Kevin Spacey; Jen...|A stuffy business...| Barry Sonnenfeld|Comedy;Family;Fan...|       11|   5.3|  19.64|         87|          Nine Lives| 12435|2016|
|   101|Tom Hardy; Emily ...|Identical twin ga...|  Brian Helgeland|Biography;Crime;D...|       55|     7|   1.87|        132|              Legend|108836|2015|
|   102|Chris Hemsworth; ...|The powerful but ...|  Kenneth Branagh|Action;Adventure;...|       57|     7| 181.02|        115|                Thor|570814|2011|
|   103|Matt Damon; Jessi...|An astronaut beco...|     Ridley Scott|Adventure;Drama;S...|       80|     8| 228.43|        144|          The Martian|556097|2015|
|   104|Mario Casas; Ana ...|A young businessm...|      Oriol Paulo|Crime;Mystery;Thr...|     null|   7.9|   null|        106|        Contratiempo|  7204|2016|
|   105|Henry Cavill; Arm...|In the early 1960...|      Guy Ritchie|Action;Adventure;...|       56|   7.3|  45.43|        116|The Man from U.N....|202973|2015|
|   106|Chris Pine; Ben F...|A divorced father...|  David Mackenzie|Crime;Drama;Thriller|       88|   7.7|  26.86|        102|  Hell or High Water|115546|2016|
|   107|Robert De Niro; L...|A look at the lif...|   Taylor Hackford|              Comedy|       40|   5.4|   1.66|        120|         The Comedian|  1954|2016|
|   108|Alexander Skarsgå...|Tarzan having acc...|      David Yates|Action;Adventure;...|       44|   6.3| 126.59|        110|The Legend of Tarzan|117590|2016|
|   109|Eve Lindley; Rich...|A mother struggle...|     Katie Holmes|               Drama|       48|   5.8|   null|        105|          All We Had|  1004|2016|
|    11|Eddie Redmayne; K...|The adventures of...|      David Yates|Adventure;Family;...|       66|   7.5| 234.02|        133|Fantastic Beasts ...|232072|2016|
|   110|Alicia Vikander; ...|A young programme...|     Alex Garland|Drama;Mystery;Sci-Fi|       78|   7.7|  25.44|        108|          Ex Machina|339797|2014|
|   111|John Gallagher Jr...|In a twisted soci...|      Greg McLean|Action;Horror;Thr...|       44|   6.3|  10.16|         89|The Belko Experiment|  3712|2016|
|   112|Chiwetel Ejiofor;...|In the antebellum...|    Steve McQueen|Biography;Drama;H...|       96|   8.1|  56.67|        134|    12 Years a Slave|486338|2013|
|   113|Keanu Reeves; Jas...|A dystopian love ...|Ana Lily Amirpour|      Romance;Sci-Fi|       65|   6.1|   null|        118|       The Bad Batch|   512|2016|
|   114|Gerard Butler; Le...|King Leonidas of ...|     Zack Snyder|  Action;Fantasy;War|       52|   7.7| 210.59|        117|                 300|637104|2006|
|   115|Daniel Radcliffe;...|Harry Ron and Her...|      David Yates|Adventure;Drama;F...|       87|   8.1| 380.96|        130|Harry Potter and ...|590595|2011|
+------+--------------------+--------------------+-----------------+--------------------+---------+------+-------+-----------+--------------------+------+----+
only showing top 20 rows


scala> df.createTempView("imdb_Mdata")

scala> spark.sqlContext.sql("select title, rating from imdb_Mdata").show
+--------------------+------+
|               title|rating|
+--------------------+------+
|Guardians of the ...|   8.1|
|          Passengers|     7|
|        The Departed|   8.5|
|          Nine Lives|   5.3|
|              Legend|     7|
|                Thor|     7|
|         The Martian|     8|
|        Contratiempo|   7.9|
|The Man from U.N....|   7.3|
|  Hell or High Water|   7.7|
|         The Comedian|   5.4|
|The Legend of Tarzan|   6.3|
|          All We Had|   5.8|
|Fantastic Beasts ...|   7.5|
|          Ex Machina|   7.7|
|The Belko Experiment|   6.3|
|    12 Years a Slave|   8.1|
|       The Bad Batch|   6.1|
|                 300|   7.7|
```

```
scala> def withCatalog(cat: String): DataFrame = {
     |     spark.sqlContext
     |     .read
     |     .options(Map(HBaseTableCatalog.tableCatalog->cat))
     |     .format("org.apache.spark.sql.execution.datasources.hbase")
     |     .load()
     |  }
withCatalog: (cat: String)org.apache.spark.sql.DataFrame

scala> val df = withCatalog(catalog)
19/11/18 21:51:01 WARN Configuration: hbase-site.xml:an attempt to override final parameter: dfs.support.append;  Ignoring.
df: org.apache.spark.sql.DataFrame = [rowkey: string, adult: string ... 17 more fields]

scala> df.show()
```

```
+-----+-----+--------+--------------------+---------+-----------------+--------------------+--------------------+----------+--------------------+--------------------+------------+---------+-------+-------
------------+--------+--------------------+------------+----------+
|   id|adult|  budget|              genres|  imdb_id|original_language|      original_title|            overview|popularity|production_companies|production_countries|release_date|  revenue|runtime|    spo
ken_languages|  status|             tagline|vote_average|vote_count|
|  862|FALSE|30000000|[{'id': 16 'name'...|tt0114709|               en|           Toy Story|Led by Woody Andy...| 21.946943|[{'name': 'Pixar ...|[{'iso_3166_1': '...|  10/30/1995|373554033|     81|[{'iso_
639_1': 'e...|Released|                null|         7.7|      5415|
| 8844|FALSE|65000000|[{'id': 12 'name'...|tt0113497|               en|             Jumanji|When siblings Jud...| 17.015539|[{'name': 'TriSta...|[{'iso_3166_1': '...|  12/15/1995|262797249|    104|[{'iso_
639_1': 'e...|Released|Roll the dice and...|         6.9|      2413|
|15602|FALSE|       0|[{'id': 10749 'na...|tt0113228|               en|    Grumpier Old Men|A family wedding ...|   11.7129|[{'name': 'Warner...|[{'iso_3166_1': '...|  12/22/1995|        0|    101|[{'iso_
639_1': 'e...|Released|Still Yelling. St...|         6.5|        92|
|31357|FALSE|16000000|[{'id': 35 'name'...|tt0114885|               en|   Waiting to Exhale|"Cheated on mistr...|  3.859495|[{'name': 'Twenti...|[{'iso_3166_1': '...|  12/22/1995| 81452156|    127|[{'iso_
639_1': 'e...|Released|Friends are the p...|         6.1|        34|
|11862|FALSE|       0|[{'id': 35 'name'...|tt0113041|               en|Father of the Bri...|Just when George ...|  8.387519|[{'name': 'Sandol...|[{'iso_3166_1': '...|   2/10/1995| 76578911|    106|[{'iso_
639_1': 'e...|Released|Just When His Wor...|         5.7|       173|
|  949|FALSE|60000000|[{'id': 28 'name'...|tt0113277|               en|                Heat|Obsessive master ...| 17.924927|[{'name': 'Regenc...|[{'iso_3166_1': '...|  12/15/1995|187436818|    170|[{'iso_
639_1': 'e...|Released|A Los Angeles Cri...|         7.7|      1886|
|11860|FALSE|58000000|[{'id': 35 'name'...|tt0114319|               en|             Sabrina|An ugly duckling ...|  6.677277|[{'name': 'Paramo...|[{'iso_3166_1': '...|  12/15/1995|        0|    127|[{'iso_
639_1': 'f...|Released|You are cordially...|         6.2|       141|
|45325|FALSE|       0|[{'id': 28 'name'...|tt0112302|               en|        Tom and Huck|A mischievous you...|  2.561161|[{'name': 'Walt D...|[{'iso_3166_1': '...|  12/22/1995|        0|     97|[{'iso_
639_1': 'e...|Released|The Original Bad ...|         5.4|        45|
| 9091|FALSE|35000000|[{'id': 28 'name'...|tt0114576|               en|        Sudden Death|International act...|   5.23158|[{'name': 'Univer...|[{'iso_3166_1': '...|  12/22/1995| 64350171|    106|[{'iso_
639_1': 'e...|Released|Terror goes into ...|         5.5|       174|
|  710|FALSE|58000000|[{'id': 12 'name'...|tt0113189|               en|           GoldenEye|James Bond must u...| 14.686036|[{'name': 'United...|[{'iso_3166_1': '...|  11/16/1995|352194034|    130|[{'iso_
639_1': 'e...|Released|No limits. No fea...|         6.6|      1194|
| 9087|FALSE|62000000|[{'id': 35 'name'...|tt0112346|               en|The American Pres...|Widowed U.S. pres...|  6.318445|[{'name': 'Columb...|[{'iso_3166_1': '...|  11/17/1995|107879496|    106|[{'iso_
639_1': 'e...|Released|Why can't the mos...|         6.5|       199|
|12110|FALSE|       0|[{'id': 35 'name'...|tt0112896|               en|Dracula: Dead and...|When a lawyer sho...|  5.430331|[{'name': 'Columb...|[{'iso_3166_1': '...|  12/22/1995|        0|     88|[{'iso_
639_1': 'e...|Released|                null|         5.7|       210|
|21032|FALSE|       0|[{'id': 10751 'na...|tt0112453|               en|               Balto|An outcast half-w...| 12.140733|[{'name': 'Univer...|[{'iso_3166_1': '...|  12/22/1995| 11348324|     78|[{'iso_
639_1': 'e...|Released|Part Dog. Part Wo...|         7.1|       423|
|10858|FALSE|44000000|[{'id': 36 'name'...|tt0113987|               en|               Nixon|An all-star cast ...|     5.092|[{'name': 'Hollyw...|[{'iso_3166_1': '...|  12/22/1995| 13681765|    192|[{'iso_
639_1': 'e...|Released|Triumphant in Vic...|         7.1|        72|
| 1408|FALSE|98000000|[{'id': 28 'name'...|tt0112760|               en|    Cutthroat Island|Morgan Adams and ...|  7.284477|[{'name': 'Le Stu...|[{'iso_3166_1': '...|  12/22/1995| 10017322|    119|[{'iso_
639_1': 'e...|Released|The Course Has Be...|         5.7|       137|
|  524|FALSE|52000000|[{'id': 18 'name'...|tt0112641|               en|              Casino|The life of the g...| 10.137389|[{'name': 'Univer...|[{'iso_3166_1': '...|  11/22/1995|116112375|    178|[{'iso_
639_1': 'e...|Released|No one stays at t...|         7.8|      1343|
| 4584|FALSE|16500000|[{'id': 18 'name'...|tt0114388|               en|Sense and Sensibi...|Rich Mr. Dashwood...| 10.673167|[{'name': 'Columb...|[{'iso_3166_1': '...|  12/13/1995|135000000|    136|[{'iso_
639_1': 'e...|Released|Lose your heart a...|         7.2|       364|
|    5|FALSE| 4000000|[{'id': 80 'name'...|tt0113101|               en|          Four Rooms|It's Ted the Bell...|  9.026586|[{'name': 'Mirama...|[{'iso_3166_1': '...|   12/9/1995|  4300000|     98|[{'iso_
639_1': 'e...|Released|Twelve outrageous...|         6.5|       539|
| 9273|FALSE|30000000|[{'id': 80 'name'...|tt0112281|               en|Ace Ventura: When...|Summoned from an ...|  8.205448|[{'name': 'O Ente...|[{'iso_3166_1': '...|  11/10/1995|212385533|     90|[{'iso_
639_1': 'e...|Released|New animals. New ...|         6.1|      1128|
+-----+-----+--------+--------------------+---------+-----------------+--------------------+--------------------+----------+--------------------+--------------------+------------+---------+-------+-------
------------+--------+--------------------+------------+----------+
only showing top 20 rows

>>> 
```

- Querying the data
  Querying was done to extract data for various features against revenue and other indicators of success of a movie.
  We compared multiple features to understand how much a feature affects the revenue.

SQL Results:

We first compared the number of movies released each year to see if more number of movies result in more revenue. This is a straightforward and direct analysis and the results agree.

-------movie count each release_year-------------

```
>>> result2 = spark.sql("SELECT _c6, count(*) as _c1 from df group by _c6 order by _c1 DESC")
>>> result2.collect()
[Row(_c6=u'2016', _c1=297), Row(_c6=u'2015', _c1=127), Row(_c6=u'2014', _c1=98), Row(_c6=u'2013', _c1=91), Row(_c6=u'2012', _c1=64), Row(_c6=u'2011', _c1=63), Row(_c6=u'2010', _c1=60), Row(_c6=u'2007', _c
1=53), Row(_c6=u'2008', _c1=52), Row(_c6=u'2009', _c1=51), Row(_c6=u'2006', _c1=44)]
>>>
>>>
>>>
>>>
>>>
>>>
```

We tried to compare if a director is an important factor in how well a movie runs.

-------Movie count each director directed--------------------

```
>>>
>>>
>>> result4 = spark.sql("SELECT _c4, count(*) as _c1 from df group by _c4 having _c1>4 order by _c1 DESC")
>>> result4.collect()
[Stage 18:==================================>                  (115 + 6) / 2[Stage 18:==================================================>          (165 + 7) / 2
                    [Stage 20:==========================================>              (145 + 4) / 2[Stage 20:=====================================================>      (181 + 4) / 2
                                        [Row(_c4=u'Ridley Scott', _c1=8), Row(_c4=u'Paul W.S. Anderson', _c1=6), Row(_c4=u'Michael Bay', _c1=6), Row(_c4=u'David Yates', _c1=6), Row(_c4=u'M.
Night Shyamalan', _c1=6), Row(_c4=u'Antoine Fuqua', _c1=5), Row(_c4=u'Zack Snyder', _c1=5), Row(_c4=u'Danny Boyle', _c1=5), Row(_c4=u'Justin Lin', _c1=5), Row(_c4=u'J.J. Abrams', _c1=5), Row(_c4=u'Woody A
llen', _c1=5), Row(_c4=u'Peter Berg', _c1=5), Row(_c4=u'David Fincher', _c1=5), Row(_c4=u'Christopher Nolan', _c1=5), Row(_c4=u'Martin Scorsese', _c1=5), Row(_c4=u'Denis Villeneuve', _c1=5)]
>>>
```

We also wanted to compare the ratings of movies over the years and see if higher ratings result in higher revenue

--------movie count each year with rating--------------------------------

```
>>> result6 = spark.sql("SELECT _c6, _c8, count(*) as _c1 from df group by _c6,_c8 having _c1>4 order by _c1 DESC")
>>> result6.collect()
[Stage 23:==========================================>          (156 + 4) /                              [Stage 25:======================================>
           (141 + 4) / [Stage 25:==========================================================>   (187 + 5) /                    [Row(_c6=u'2016', _c8=u'6.3'
, _c1=19), Row(_c6=u'2016', _c8=u'7.4', _c1=14), Row(_c6=u'2016', _c8=u'6.1', _c1=14), Row(_c6=u'2016', _c8=u'7.5', _c1=13), Row(_c6=u'2016', _c8=u'5.8', _c1=12), Row(_c6=u'2016', _c8=u'6.8', _c1=12), Row
(_c6=u'2016', _c8=u'7.1', _c1=11), Row(_c6=u'2016', _c8=u'6.7', _c1=11), Row(_c6=u'2016', _c8=u'7.2', _c1=11), Row(_c6=u'2016', _c8=u'6', _c1=10), Row(_c6=u'2016', _c8=u'6.5', _c1=10), Row(_c6=u'2016', _c
8=u'6.9', _c1=10), Row(_c6=u'2016', _c8=u'5.7', _c1=10), Row(_c6=u'2015', _c8=u'7.1', _c1=10), Row(_c6=u'2016', _c8=u'6.4', _c1=10), Row(_c6=u'2016', _c8=u'7.3', _c1=9), Row(_c6=u'2016', _c8=u'7', _c1=9),
 Row(_c6=u'2015', _c8=u'6.3', _c1=9), Row(_c6=u'2013', _c8=u'7', _c1=9), Row(_c6=u'2016', _c8=u'6.6', _c1=8), Row(_c6=u'2015', _c8=u'6.7', _c1=8), Row(_c6=u'2014', _c8=u'6.2', _c1=8), Row(_c6=u'2016', _c8
=u'6.2', _c1=8), Row(_c6=u'2016', _c8=u'7.9', _c1=8), Row(_c6=u'2007', _c8=u'7.1', _c1=7), Row(_c6=u'2016', _c8=u'5.6', _c1=7), Row(_c6=u'2014', _c8=u'8.1', _c1=7), Row(_c6=u'2015', _c8=u'7.3', _c1=7), Ro
w(_c6=u'2015', _c8=u'6.5', _c1=7), Row(_c6=u'2016', _c8=u'5.9', _c1=7), Row(_c6=u'2013', _c8=u'7.3', _c1=6), Row(_c6=u'2014', _c8=u'6', _c1=6), Row(_c6=u'2016', _c8=u'5.3', _c1=6), Row(_c6=u'2015', _c8=u'
6.6', _c1=6), Row(_c6=u'2014', _c8=u'6.7', _c1=6), Row(_c6=u'2013', _c8=u'6.5', _c1=6), Row(_c6=u'2013', _c8=u'7.8', _c1=6), Row(_c6=u'2007', _c8=u'7.2', _c1=6), Row(_c6=u'2015', _c8=u'6', _c1=6), Row(_c6
=u'2013', _c8=u'6.7', _c1=6), Row(_c6=u'2015', _c8=u'7', _c1=6), Row(_c6=u'2014', _c8=u'6.5', _c1=6), Row(_c6=u'2015', _c8=u'7.2', _c1=5), Row(_c6=u'2012', _c8=u'7', _c1=5), Row(_c6=u'2014', _c8=u'7.8', _
c1=5), Row(_c6=u'2008', _c8=u'6.6', _c1=5), Row(_c6=u'2007', _c8=u'7.5', _c1=5), Row(_c6=u'2013', _c8=u'7.5', _c1=5), Row(_c6=u'2011', _c8=u'7.1', _c1=5), Row(_c6=u'2013', _c8=u'6.2', _c1=5), Row(_c6=u'20
16', _c8=u'5.2', _c1=5), Row(_c6=u'2016', _c8=u'7.7', _c1=5), Row(_c6=u'2012', _c8=u'7.2', _c1=5), Row(_c6=u'2013', _c8=u'6.6', _c1=5), Row(_c6=u'2013', _c8=u'7.1', _c1=5), Row(_c6=u'2008', _c8=u'7.1', _c
1=5), Row(_c6=u'2016', _c8=u'5.4', _c1=5), Row(_c6=u'2010', _c8=u'6.8', _c1=5), Row(_c6=u'2014', _c8=u'6.3', _c1=5), Row(_c6=u'2015', _c8=u'5.7', _c1=5)]
```

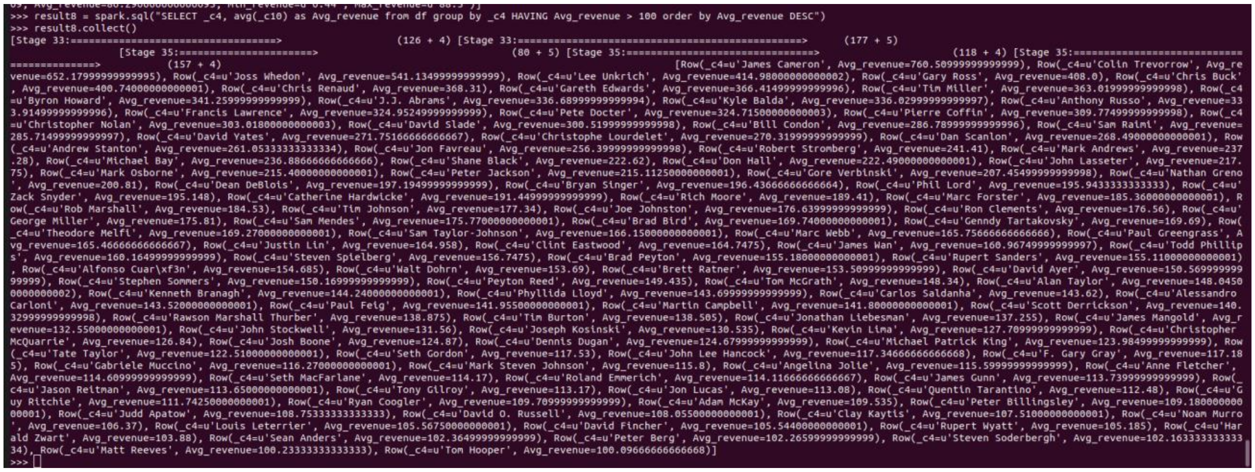
Since the number of movies and their revenues differ for every year, we tried to get the standardized values of the revenue by averaging them over the total number of movies that year.

--------Revenue Stylized Facts per year----------------

```
>>> result5 = spark.sql("SELECT _c6, _c8 count(*) as _c1 from df group>>> r>>> result7 = spark.sql("SELECT _c6, sum(_c10) as sum_revenue, avg(_c10) as Avg_revenue, min(_c10) as min_revenue, max(_c10) as m
ax_revenue from df group by _c6 order by _c6 DESC")
>>> result7.collect()
[Stage 28:=====================>                                   (79 + 4) /[Stage 28:=============================>                  (116 + 4) /[Stage 28:===================================>
         (142 + 4) /[Stage 28:=====================================================>     (179 + 4) /[Stage 28:=================================================> (193 + 4) /
                                         [Stage 30:==============================>               (114 + 4) /[Stage 30:========================================>              (149 + 4) /[Stage 30:==
===================================================>  (190 + 4) /                                          [Row(_c6=u'2016', sum_revenue=11211.650000000003, Avg_revenue=54.6
90975609756116, min_revenue=u'0', max_revenue=u'97.66'), Row(_c6=u'2015', sum_revenue=8854.1200000000026, Avg_revenue=78.355044247787632, min_revenue=u'0.01', max_revenue=u'936.63'), Row(_c6=u'2014', sum_
revenue=7997.3999999999978, Avg_revenue=85.0787234042553, min_revenue=u'0.01', max_revenue=u'91.12'), Row(_c6=u'2013', sum_revenue=7666.7199999999966, Avg_revenue=87.121818181818142, min_revenue=u'0.03',
max_revenue=u'98.9'), Row(_c6=u'2012', sum_revenue=6910.2900000000027, Avg_revenue=107.97328125000004, min_revenue=u'0.02', max_revenue=u'95.72'), Row(_c6=u'2011', sum_revenue=5431.9600000000009, Avg_reve
nue=87.612258064516141, min_revenue=u'0.03', max_revenue=u'85.46'), Row(_c6=u'2010', sum_revenue=5989.6500000000005, Avg_revenue=105.08157894736843, min_revenue=u'0.02', max_revenue=u'96.92'), Row(_c6=u'2
009', sum_revenue=5292.2600000000011, Avg_revenue=112.60127659574471, min_revenue=u'0.06', max_revenue=u'97.03'), Row(_c6=u'2008', sum_revenue=5053.2200000000021, Avg_revenue=99.082745098039254, min_reven
ue=u'0.07', max_revenue=u'9.03'), Row(_c6=u'2007', sum_revenue=4306.2300000000005, Avg_revenue=87.882244897959197, min_revenue=u'0.04', max_revenue=u'82.23'), Row(_c6=u'2006', sum_revenue=3624.46000000000
09, Avg_revenue=86.296666666666695, min_revenue=u'0.44', max_revenue=u'88.5')]
>>>
```

We extracted the revenues of every director

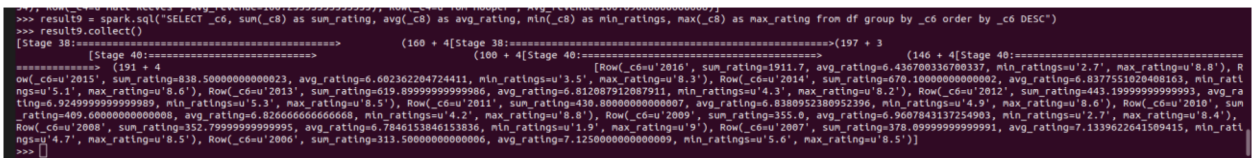
--------Avg. Revenue of movies per director---------------



```
>>> result8 = spark.sql("SELECT _c4, avg(_c10) as Avg_revenue from df group by _c4 HAVING Avg_revenue > 100 order by Avg_revenue DESC")
>>> result8.collect()
[Stage 33:===================================>                    (126 + 4) [Stage 33:==================================================>     (177 + 5)
               [Stage 35:=======================>                                (80 + 5) [Stage 35:==================================>                 (118 + 4) [Stage 35:=============================
==============>         (157 + 4)                                                            [Row(_c4=u'James Cameron', Avg_revenue=760.5099999999999), Row(_c4=u'Colin Trevorrow', Avg_re
venue=652.1799999999995), Row(_c4=u'Joss Whedon', Avg_revenue=541.1349999999999), Row(_c4=u'Lee Unkrich', Avg_revenue=414.98000000000002), Row(_c4=u'Gary Ross', Avg_revenue=408.0), Row(_c4=u'Chris Buck'
, Avg_revenue=400.74000000000001), Row(_c4=u'Chris Renaud', Avg_revenue=368.31), Row(_c4=u'Gareth Edwards', Avg_revenue=366.41499999999996), Row(_c4=u'Tim Miller', Avg_revenue=363.01999999999998), Row(_c4
=u'Byron Howard', Avg_revenue=341.2599999999999), Row(_c4=u'J.J. Abrams', Avg_revenue=336.68999999999994), Row(_c4=u'Kyle Balda', Avg_revenue=336.02999999999997), Row(_c4=u'Anthony Russo', Avg_revenue=33
3.91499999999996), Row(_c4=u'Francis Lawrence', Avg_revenue=324.9524999999999), Row(_c4=u'Pete Docter', Avg_revenue=324.71500000000003), Row(_c4=u'Pierre Coffin', Avg_revenue=309.77499999999998), Row(_c4
=u'Christopher Nolan', Avg_revenue=303.01800000000003), Row(_c4=u'David Slade', Avg_revenue=300.51999999999998), Row(_c4=u'Bill Condon', Avg_revenue=286.78999999999996), Row(_c4=u'Sam Raimi', Avg_revenue=
285.71499999999997), Row(_c4=u'David Yates', Avg_revenue=271.75166666666667), Row(_c4=u'Christophe Lourdelet', Avg_revenue=270.3199999999999), Row(_c4=u'Dan Scanlon', Avg_revenue=268.49000000000001), Row
(_c4=u'Andrew Stanton', Avg_revenue=261.05333333333334), Row(_c4=u'Jon Favreau', Avg_revenue=256.39999999999998), Row(_c4=u'Robert Stromberg', Avg_revenue=241.41), Row(_c4=u'Mark Andrews', Avg_revenue=237
.28), Row(_c4=u'Michael Bay', Avg_revenue=236.88666666666666), Row(_c4=u'Shane Black', Avg_revenue=222.62), Row(_c4=u'Don Hall', Avg_revenue=222.49000000000001), Row(_c4=u'John Lasseter', Avg_revenue=217.
75), Row(_c4=u'Mark Osborne', Avg_revenue=215.40000000000001), Row(_c4=u'Peter Jackson', Avg_revenue=215.11250000000001), Row(_c4=u'Gore Verbinski', Avg_revenue=207.45499999999998), Row(_c4=u'Nathan Greno
', Avg_revenue=200.81), Row(_c4=u'Dean DeBlois', Avg_revenue=197.1949999999999), Row(_c4=u'Bryan Singer', Avg_revenue=196.43666666666664), Row(_c4=u'Phil Lord', Avg_revenue=195.9433333333333), Row(_c4=u'
Zack Snyder', Avg_revenue=195.148), Row(_c4=u'Catherine Hardwicke', Avg_revenue=191.4499999999999), Row(_c4=u'Rich Moore', Avg_revenue=189.41), Row(_c4=u'Marc Forster', Avg_revenue=185.36000000000001), R
ow(_c4=u'Rob Marshall', Avg_revenue=184.53), Row(_c4=u'Tim Johnson', Avg_revenue=177.34), Row(_c4=u'Joe Johnston', Avg_revenue=176.6399999999999), Row(_c4=u'Ron Clements', Avg_revenue=176.56), Row(_c4=u'
George Miller', Avg_revenue=175.81), Row(_c4=u'Sam Mendes', Avg_revenue=175.77000000000001), Row(_c4=u'Brad Bird', Avg_revenue=169.74000000000001), Row(_c4=u'Genndy Tartakovsky', Avg_revenue=169.69), Row(
_c4=u'Theodore Melfi', Avg_revenue=169.27000000000001), Row(_c4=u'Sam Taylor-Johnson', Avg_revenue=166.15000000000001), Row(_c4=u'Marc Webb', Avg_revenue=165.75666666666666), Row(_c4=u'Paul Greengrass', A
vg_revenue=165.46666666666667), Row(_c4=u'Justin Lin', Avg_revenue=164.958), Row(_c4=u'Clint Eastwood', Avg_revenue=164.7475), Row(_c4=u'James Wan', Avg_revenue=160.96749999999997), Row(_c4=u'Todd Phillip
s', Avg_revenue=160.1649999999999), Row(_c4=u'Steven Spielberg', Avg_revenue=156.7475), Row(_c4=u'Brad Peyton', Avg_revenue=155.18000000000001), Row(_c4=u'Rupert Sanders', Avg_revenue=155.11000000000001)
, Row(_c4=u'Alfonso Cuar\xf3n', Avg_revenue=154.685), Row(_c4=u'Walt Dohrn', Avg_revenue=153.69), Row(_c4=u'Brett Ratner', Avg_revenue=153.5099999999999), Row(_c4=u'David Ayer', Avg_revenue=150.569999999
99999), Row(_c4=u'Stephen Sommers', Avg_revenue=150.1699999999999), Row(_c4=u'Peyton Reed', Avg_revenue=149.435), Row(_c4=u'Tom McGrath', Avg_revenue=148.34), Row(_c4=u'Alan Taylor', Avg_revenue=148.0450
0000000002), Row(_c4=u'Kenneth Branagh', Avg_revenue=144.24000000000001), Row(_c4=u'Phyllida Lloyd', Avg_revenue=143.6999999999999), Row(_c4=u'Carlos Saldanha', Avg_revenue=143.62), Row(_c4=u'Alessandro
Carloni', Avg_revenue=143.52000000000001), Row(_c4=u'Paul Feig', Avg_revenue=141.95500000000001), Row(_c4=u'Martin Campbell', Avg_revenue=141.80000000000001), Row(_c4=u'Scott Derrickson', Avg_revenue=140.
32999999999998), Row(_c4=u'Rawson Marshall Thurber', Avg_revenue=138.875), Row(_c4=u'Tim Burton', Avg_revenue=138.505), Row(_c4=u'Jonathan Liebesman', Avg_revenue=137.255), Row(_c4=u'James Mangold', Avg_r
evenue=132.55000000000001), Row(_c4=u'John Stockwell', Avg_revenue=131.56), Row(_c4=u'Joseph Kosinski', Avg_revenue=130.535), Row(_c4=u'Kevin Lima', Avg_revenue=127.7099999999999), Row(_c4=u'Christopher
McQuarrie', Avg_revenue=126.84), Row(_c4=u'Josh Boone', Avg_revenue=124.87), Row(_c4=u'Dennis Dugan', Avg_revenue=124.6799999999999), Row(_c4=u'Michael Patrick King', Avg_revenue=123.9849999999999), Row
(_c4=u'Tate Taylor', Avg_revenue=122.51000000000001), Row(_c4=u'Seth Gordon', Avg_revenue=117.53), Row(_c4=u'John Lee Hancock', Avg_revenue=117.34666666666668), Row(_c4=u'F. Gary Gray', Avg_revenue=117.18
5), Row(_c4=u'Gabriele Muccino', Avg_revenue=116.27000000000001), Row(_c4=u'Mark Steven Johnson', Avg_revenue=115.8), Row(_c4=u'Angelina Jolie', Avg_revenue=115.5999999999999), Row(_c4=u'Anne Fletcher',
Avg_revenue=114.6099999999999), Row(_c4=u'Seth MacFarlane', Avg_revenue=114.17), Row(_c4=u'Roland Emmerich', Avg_revenue=114.11666666666667), Row(_c4=u'James Gunn', Avg_revenue=113.7399999999999), Row(_
c4=u'Jason Reitman', Avg_revenue=113.65000000000001), Row(_c4=u'Tony Gilroy', Avg_revenue=113.17), Row(_c4=u'Jon Lucas', Avg_revenue=113.08), Row(_c4=u'Quentin Tarantino', Avg_revenue=112.48), Row(_c4=u'G
uy Ritchie', Avg_revenue=111.74250000000001), Row(_c4=u'Ryan Coogler', Avg_revenue=109.7099999999999), Row(_c4=u'Adam McKay', Avg_revenue=109.535), Row(_c4=u'Peter Billingsley', Avg_revenue=109.180000000
00001), Row(_c4=u'Judd Apatow', Avg_revenue=108.75333333333333), Row(_c4=u'David O. Russell', Avg_revenue=108.05500000000001), Row(_c4=u'Clay Kaytis', Avg_revenue=107.51000000000001), Row(_c4=u'Noam Murro
', Avg_revenue=106.37), Row(_c4=u'Louis Leterrier', Avg_revenue=105.56750000000001), Row(_c4=u'David Fincher', Avg_revenue=105.54400000000001), Row(_c4=u'Rupert Wyatt', Avg_revenue=105.185), Row(_c4=u'Har
ald Zwart', Avg_revenue=103.88), Row(_c4=u'Sean Anders', Avg_revenue=102.3649999999999), Row(_c4=u'Peter Berg', Avg_revenue=102.2659999999999), Row(_c4=u'Steven Soderbergh', Avg_revenue=102.163333333333
34), Row(_c4=u'Matt Reeves', Avg_revenue=100.23333333333333), Row(_c4=u'Tom Hooper', Avg_revenue=100.09666666666668)]
>>>
```

Since the number of ratings and movies differ every year, we found the standardized values of the ratings per year by averaging the total number of ratings for the number of movies that year.

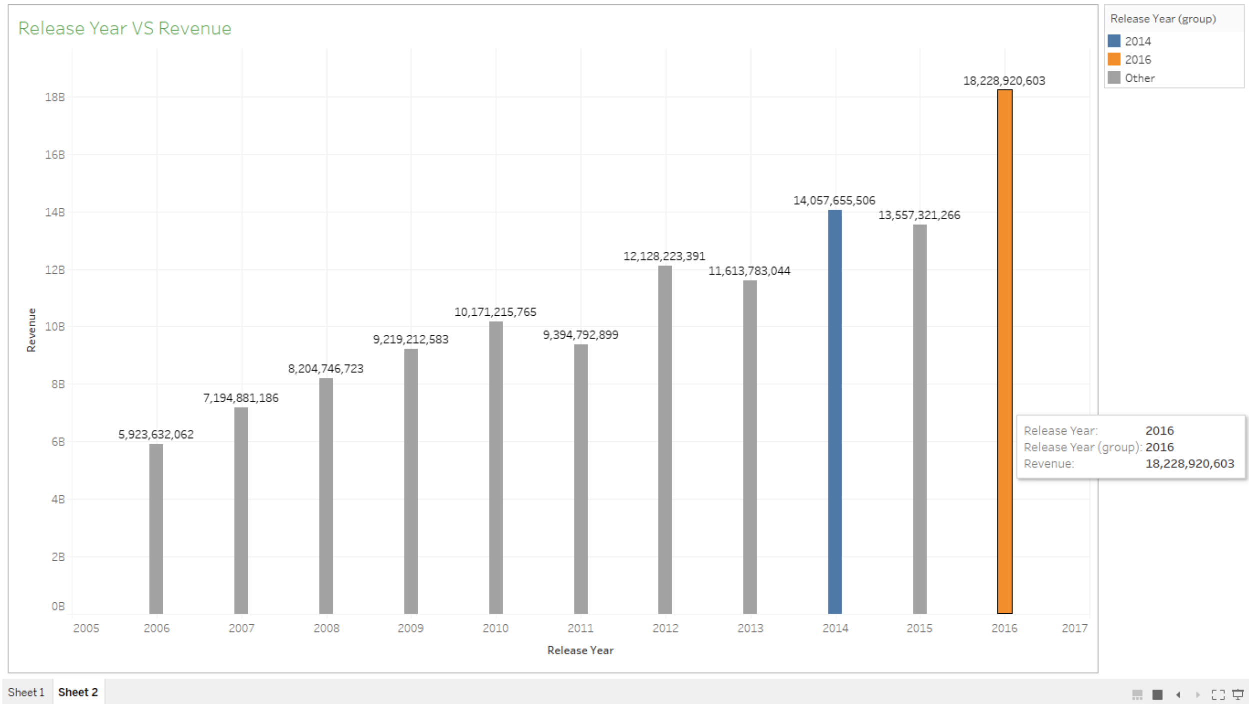
--------Ratings Stylized Facts per year---------------



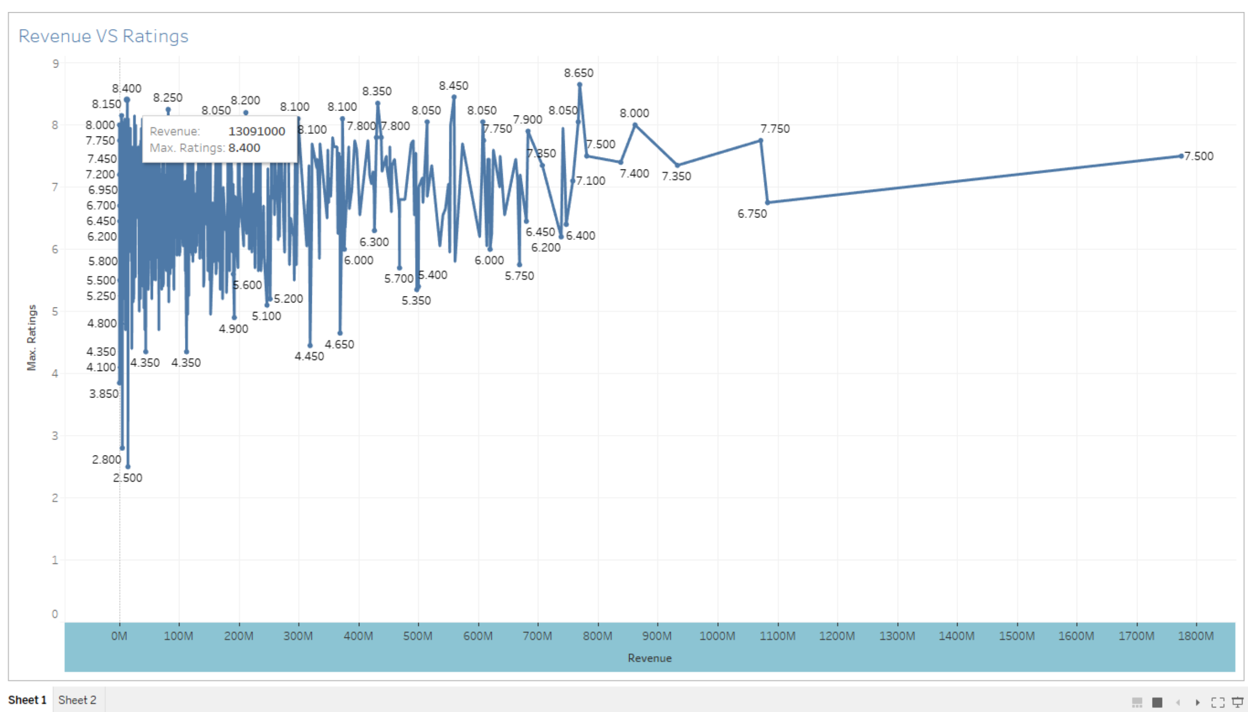
```
>>> result9 = spark.sql("SELECT _c6, sum(_c8) as sum_rating, avg(_c8) as avg_rating, min(_c8) as min_ratings, max(_c8) as max_rating from df group by _c6 order by _c6 DESC")
>>> result9.collect()
[Stage 38:============================================>          (160 + 4[Stage 38:====================================================>(197 + 3
            [Stage 40:============================>                          (100 + 4[Stage 40:=======================================>            (146 + 4[Stage 40:=======================================
=============>  (191 + 4                                                        [Row(_c6=u'2016', sum_rating=1911.7, avg_rating=6.436700336700337, min_ratings=u'2.7', max_rating=u'8.8'), R
ow(_c6=u'2015', sum_rating=838.50000000000023, avg_rating=6.602362204724411, min_ratings=u'3.5', max_rating=u'8.3'), Row(_c6=u'2014', sum_rating=670.10000000000002, avg_rating=6.8377551020408163, min_rati
ngs=u'5.1', max_rating=u'8.6'), Row(_c6=u'2013', sum_rating=619.89999999999986, avg_rating=6.812087912087911, min_ratings=u'4.3', max_rating=u'8.2'), Row(_c6=u'2012', sum_rating=443.19999999999993, avg_ra
ting=6.9249999999999989, min_ratings=u'5.3', max_rating=u'8.5'), Row(_c6=u'2011', sum_rating=430.80000000000007, avg_rating=6.8380952380952396, min_ratings=u'4.9', max_rating=u'8.6'), Row(_c6=u'2010', sum
_rating=409.60000000000008, avg_rating=6.826666666666668, min_ratings=u'4.2', max_rating=u'8.8'), Row(_c6=u'2009', sum_rating=355.0, avg_rating=6.9607843137254903, min_ratings=u'2.7', max_rating=u'8.4'),
Row(_c6=u'2008', sum_rating=352.79999999999995, avg_rating=6.7846153846153836, min_ratings=u'1.9', max_rating=u'9'), Row(_c6=u'2007', sum_rating=378.09999999999991, avg_rating=7.1339622641509415, min_rati
ngs=u'4.7', max_rating=u'8.5'), Row(_c6=u'2006', sum_rating=313.50000000000006, avg_rating=7.1250000000000009, min_ratings=u'5.6', max_rating=u'8.5')]
>>>
```

Data Visualizations

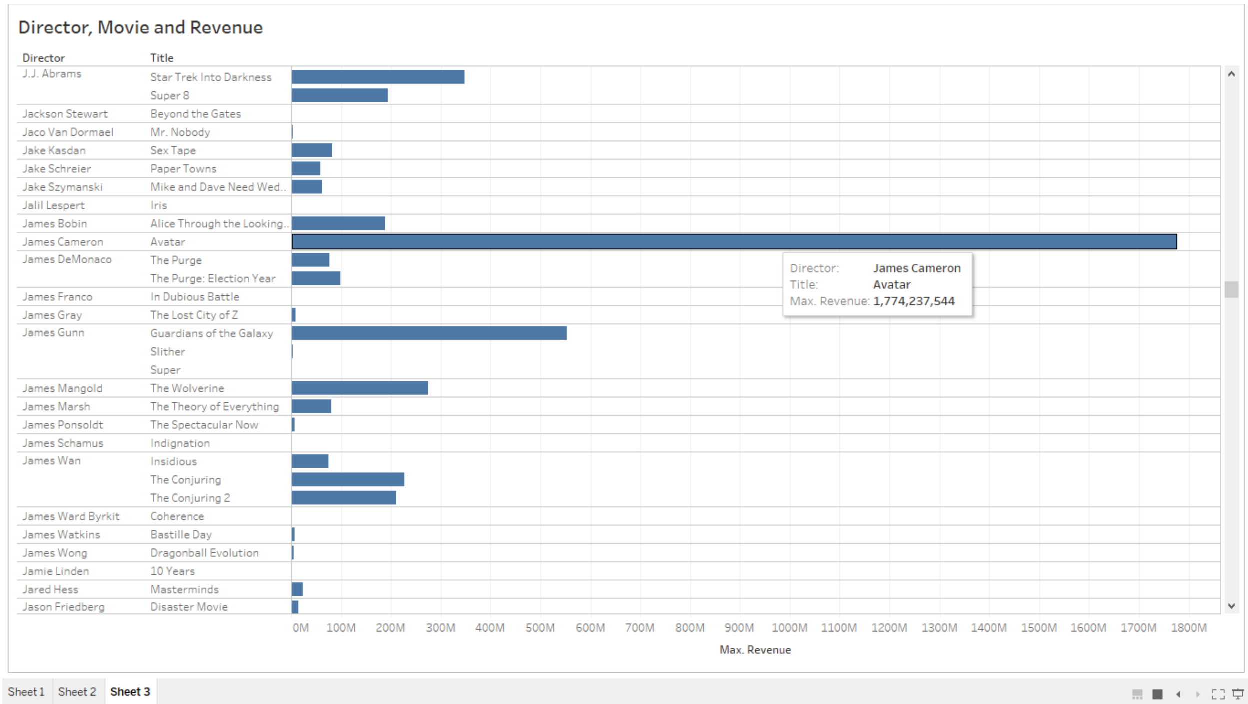
The following graphs show the relationship between the different features queried previously.



Above we see the distribution of the revenues of movies over the years 2006-2017. We see that there is not always a strict increase in revenue every year as seen in 2011, 2013 and 2015.

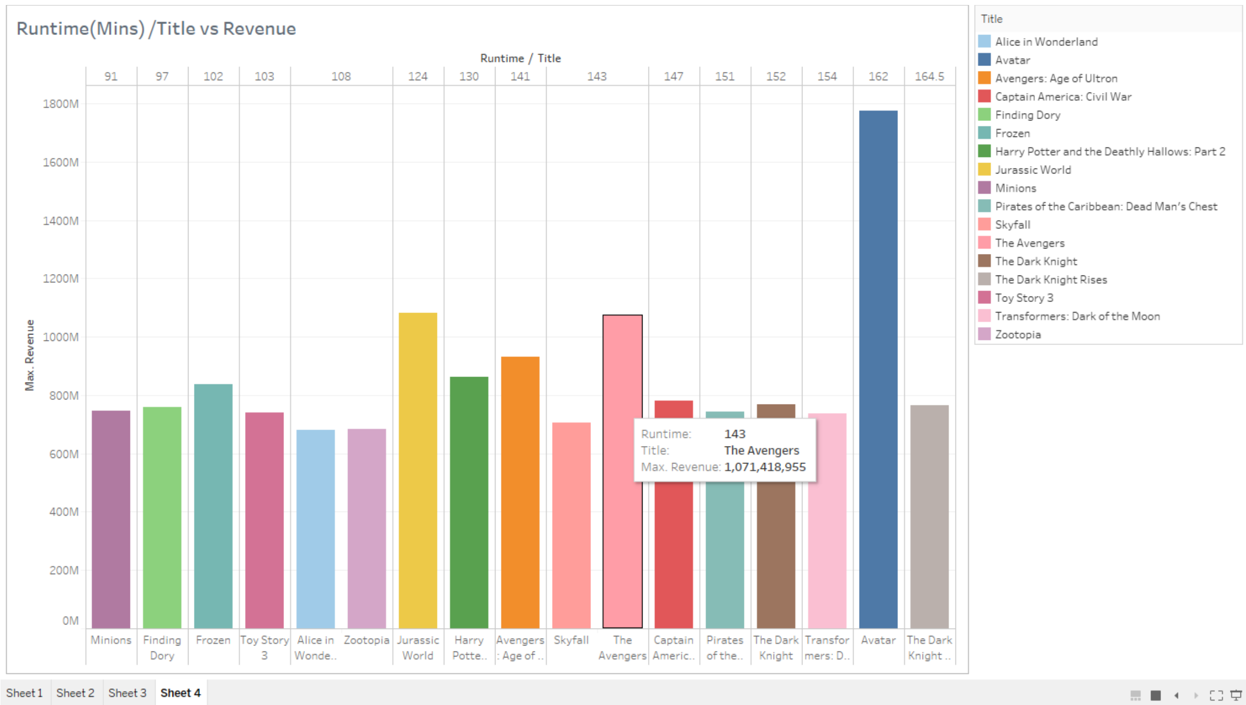


Here, we wanted to see if movies with higher ratings have more revenue. We see however that the ratings average at around 6.5 ~ 7 and there is no evidence of higher ratings resulting in higher revenue. There are movies with 8.4 and 8.6 ratings but they have revenues of <100M and 800M respectively. There are movies with lesser ratings of 6.7 rating but have very high revenue of ~1100M. This shows that the ratings alone cannot be considered while analysing the success of a movie.

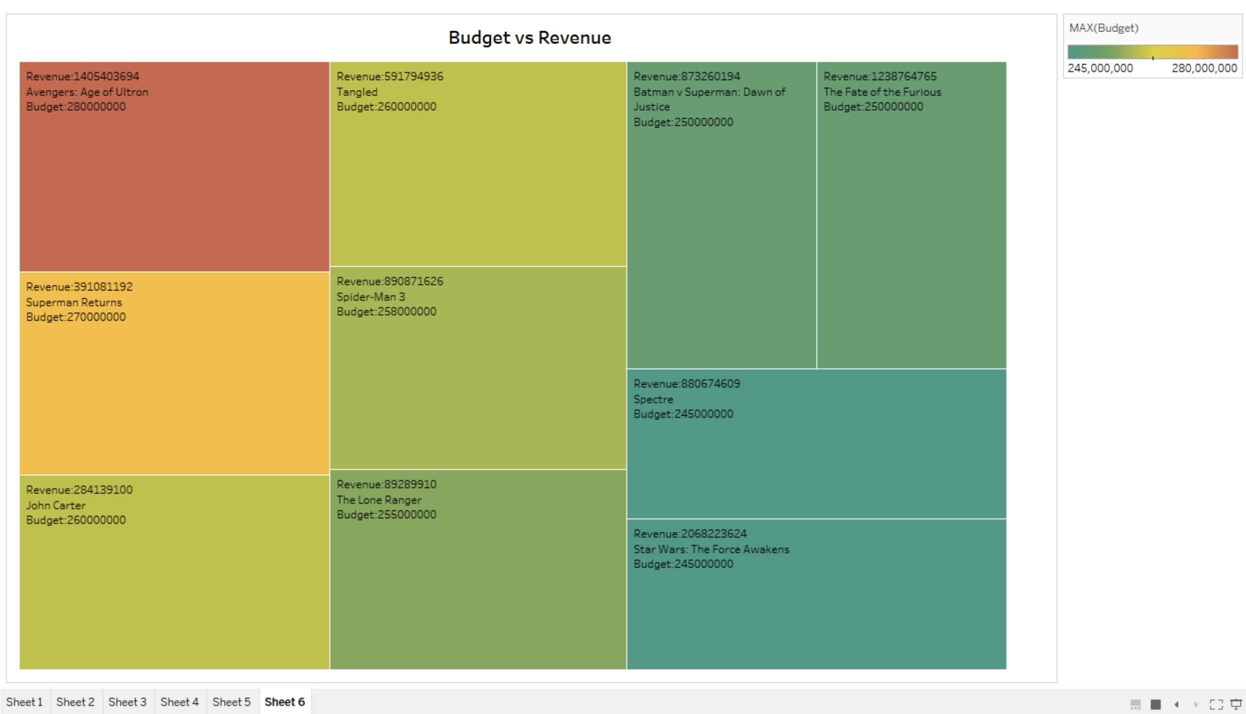


In this graph we see the effect of the high leverage point, 'Avatar'. It has the highest revenue of all time and thus results in James Cameron having directed the movie with the most revenue. However, that is his only movie in this dataset, while directors like James Gunn and J.J Abrams have more number of movies doing well in the box office.

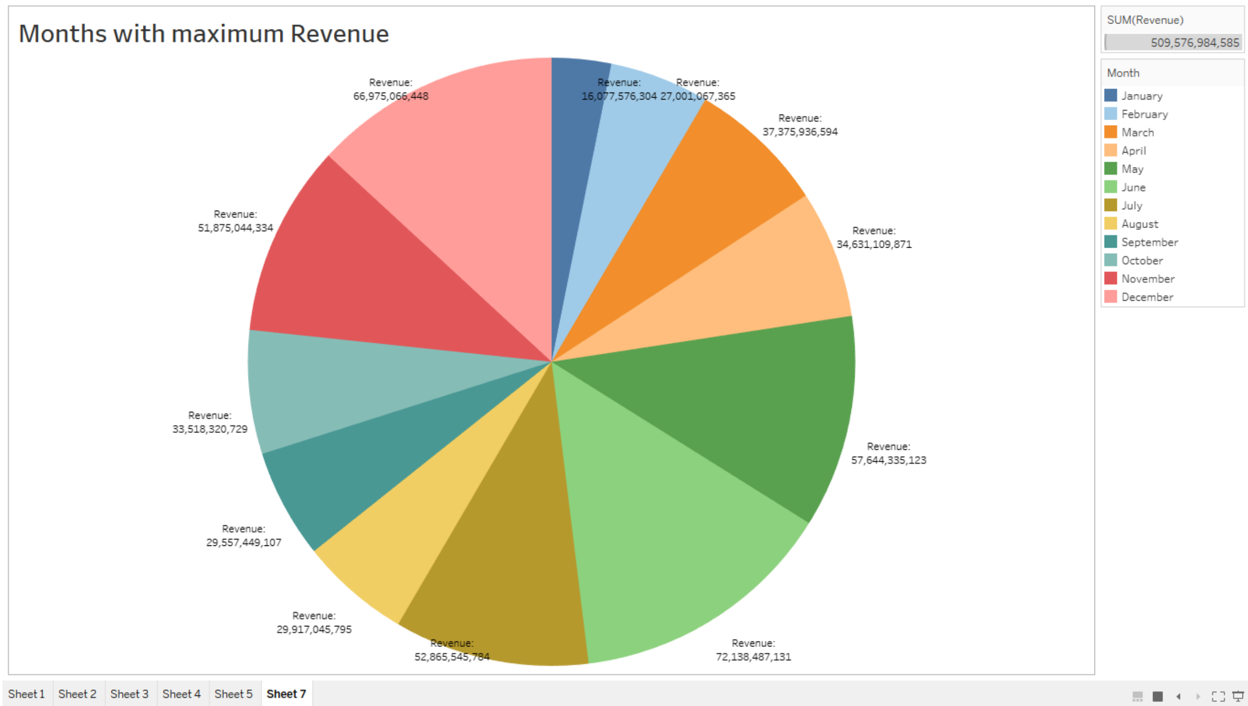
This graph visualizes the ten movies with the highest revenues in the dataset over the last 10 years. As we can see, besides Avatar, the other movies have about the same amount of success in terms of revenues.



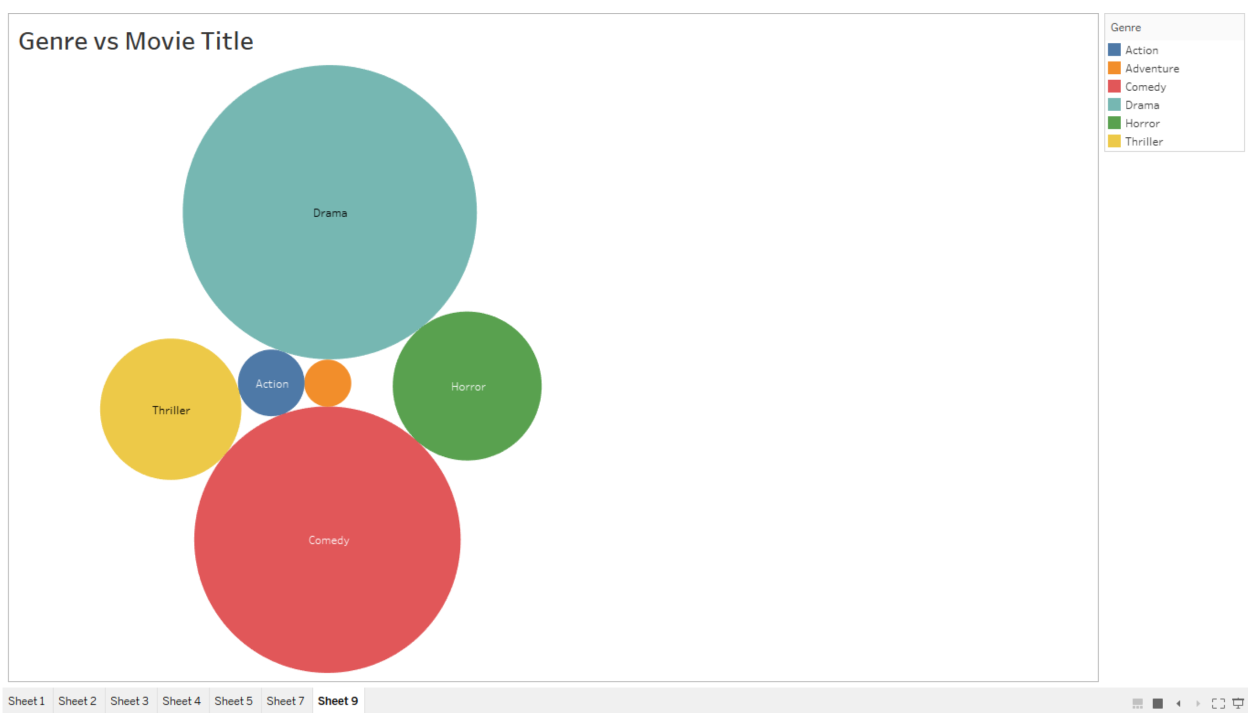
With this graph, we wanted to see if a longer movie would be better received by the audience. Again, the movie Avatar has the second highest run-time of 162 minutes and had the highest revenue. However, as can be seen, movies with longer runtimes do not necessarily do better. This can be seen with the movie Dark Knight (164.5 minutes) and the movie Transformers: Dark of the Moon (154 minutes).



This graph compares the budget and the revenue generated by movies. We tried to see if a high budget movie does well in the box office. This is not necessarily true, as seen with movies like The Lone Ranger.



We also wanted to see if when the movie is released has any effect on its revenue. As can be seen, there isn't a significant difference in the revenues to justify releasing movies in a certain month. However, it can be observed that movies released in January and mid-year do perform well.



This graph highlights the number of movies released in different genres. As we can see, Comedy and Drama have the highest number of movies in that genre. However, most movies come under more than one genre. Therefore, we cannot conclude if dramas and comedies perform better than the other genres.

Analysis and Results

We compared the revenue of a movie with various factors to see which ones had the most impact on the revenue. As explained with every graph above, the revenue of a movie depends on multiple factors on not on just one feature alone like Budget or Director. The factors that seemed to have the most relevance to the revenue of a movie is the Budget, Director, number of movies released that year and the ratings of the movie (to an extent).

Conclusions

This project aimed at implementing HBase through Spark and was successfully completed. By building two clusters, one for HBase and one for Spark and connecting them, we were able to access data stored in HBase through a Spark node and run queries on it.
Since running HBase through Spark is achieved, future projects could focus on comparing the performance of using HBase with other databases, keeping in mind the convenience and use of HDFS when using HBase. Another possible future application could be building and running machine learning models by accessing the data from HBase through Spark.

References

[1] Jaehui Park and Su-young Chi "An Implementation of a High Throughput Data Ingestion System for Machine Logs in Manufacturing Industry - IEEE Conference Publication"
https://ieeexplore.ieee.org/abstract/document/7536997

[2] Jie Yang & Brian Yecies "Mining Chinese social media UGC: a big-data framework for analyzing Douban movie reviews"
https://journalofbigdata.springeropen.com/articles/10.1186/s40537-015-0037-9

[3] Bingwei Liu, Erik Blasch, Yu Chen, Dan Shen and Genshe Chen "Scalable sentiment classification for Big Data analysis using Naïve Bayes Classifier"

https://ieeexplore.ieee.org/abstract/document/6691740

[4] Khalid Ibnal Asad, Tanvir Ahmed and Md. Saiedur Rahman "Movie Popularity Classification based on Inherent Movie Attributes using C4.5, PART and Correlation Coefficient" https://arxiv.org/ftp/arxiv/papers/1209/1209.6070.pdf

[5] Anurag Agrahari and Prof D.T.V. Dharmaji Rao "A Review paper on Big Data: Technologies, Tools and Trends" https://www.irjet.net/archives/V4/i10/IRJET-V4I10112.pdf

[6] Márton Mestyán, Taha Yasseri and János Kertész "Early Prediction of Movie Box Office Success Based on Wikipedia Activity Big Data" https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0071226

[7] Dataset : https://www.kaggle.com/rounakbanik/the-movies-dataset

[8] Comparison of Hbase, Cassandra and MongoDB
https://logz.io/blog/nosql-database-comparison/