**CS595 – Final Project Report**

* **Section – I: Project Details**

1. **Project Topic:** Build a Data processing pipeline using Big Data SQL tools
2. **Application subject area:** Entertainment & Movies
3. **Data set Source:** IMDB & Movie Lens datasets from Kaggle

<https://www.kaggle.com/tomiandrep/imdb-filmid/data>

Data set of 1,000 most popular movies on IMDB in the 10 years [2006-2016]. The data points included are: Title, Genre, Description, Director, Actors, Year, Runtime, Rating, Votes, Revenue, Metascore.

<https://www.kaggle.com/rounakbanik/the-movies-dataset/data>

The main Movies Metadata file. Contains information on 45,000 movies featured in the Full Movie Lens dataset. Features include posters, backdrops, budget, revenue, release dates, languages, production countries and companies.

1. **Relevant Question/Problem:**

Build a data processing pipeline using Big Data technologies to ingest, curate, transform, load and use techniques to explore data and derive insights on Movies data from IMDB.

1. **Proposed Approach: Tools for**

* + **Data Ingestion:** HDFS commands to move data on to Hadoop environment
  + **Data Curation & Transformation:** Spark SQL (using Data Frames)
  + **Database:** Hive
  + **Data Mining & Analysis:** HQL on Big Data and R
* **Section – II: Literature Review**

1. **Data Ingestion: HDFS**

The Hadoop Distributed File System (HDFS) is a distributed file system designed to run on commodity hardware. It has many similarities with existing distributed file systems. However, the differences from other distributed file systems are significant. HDFS is highly fault-tolerant and is designed to be deployed on low-cost hardware. HDFS provides high throughput access to application data and is suitable for applications that have large data sets.

HDFS relaxes a few POSIX requirements to enable streaming access to file system data. HDFS was originally built as infrastructure for the Apache Nutch web search engine project. HDFS is now an Apache Hadoop subproject.

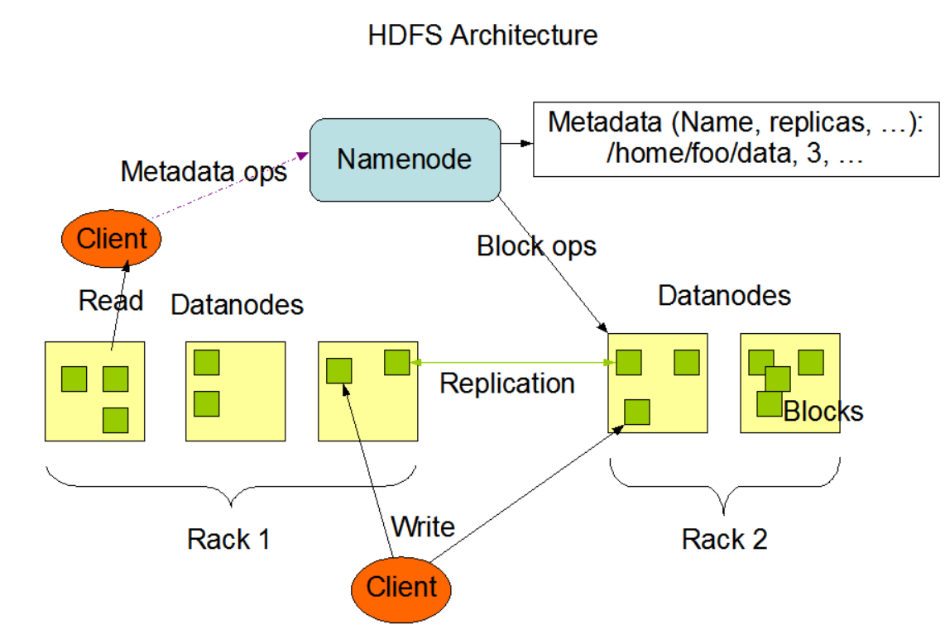
HDFS is built using the Java language; any machine that supports Java can run the NameNode or the DataNode software.

* 1. **NameNode and DataNodes**

HDFS has a master/slave architecture. An HDFS cluster consists of a single NameNode, a master server that manages the file system namespace and regulates access to files by clients. In addition, there are several Data Nodes, usually one per node in the cluster, which manage storage attached to the nodes that they run on.

HDFS exposes a file system namespace and allows user data to be stored in files. Internally, a file is split into one or more blocks and these blocks are stored in a set of Data Nodes. The NameNode executes file system namespace operations like opening, closing, and renaming files and directories. It also determines the mapping of blocks to Data Nodes.

The Data Nodes are responsible for serving read and write requests from the file system’s clients. The Data Nodes also perform block creation, deletion, and replication upon instruction from the NameNode.

****

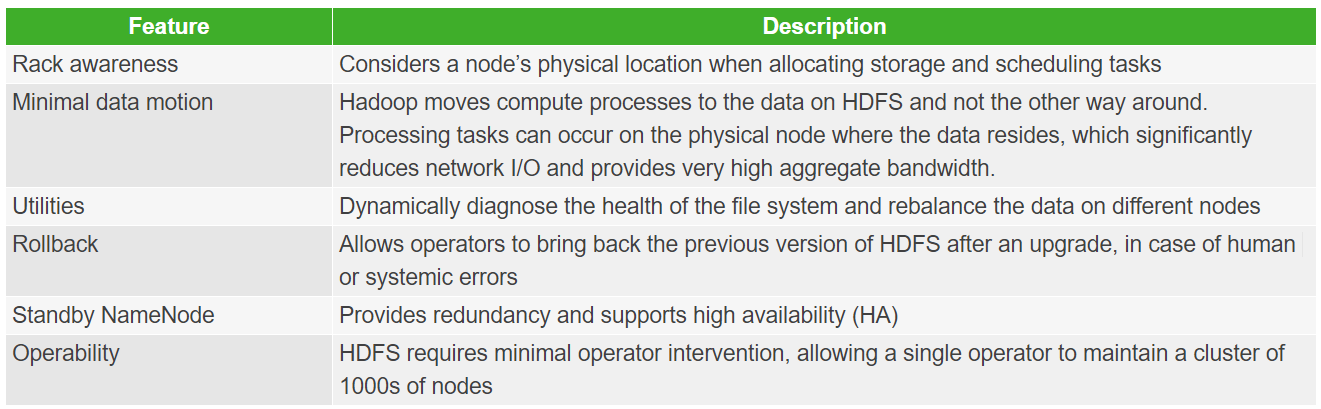
* 1. **Data Replication**

HDFS is designed to reliably store very large files across machines in a large cluster. It stores each file as a sequence of blocks; all blocks in a file except the last block are the same size. The blocks of a file are replicated for fault tolerance. The block size and replication factor are configurable per file. An application can specify the number of replicas of a file. The replication factor can be specified at file creation time and can be changed later. Files in HDFS are write-once and have strictly one writer at any time.

The NameNode maintains the file system namespace. Any change to the file system namespace or its properties is recorded by the NameNode. An application can specify the number of replicas of a file that should be maintained by HDFS. The number of copies of a file is called the replication factor of that file. This information is stored by the NameNode.

The NameNode makes all decisions regarding replication of blocks. It periodically receives a Heartbeat and a Blockreport from each of the DataNodes in the cluster. Receipt of a Heartbeat implies that the DataNode is functioning properly. A Blockreport contains a list of all blocks on a DataNode.

Following specific features ensure that data is stored efficiently in a Hadoop cluster and that it is highly available:



1. **Data Curation & Transformation: SparkSQL (using DataFrames)**

Spark SQL is a Spark module for structured data processing. Unlike the basic Spark RDD API, the interfaces provided by Spark SQL provide Spark with more information about the structure of both the data and the computation being performed. Internally, Spark SQL uses this extra information to perform extra optimizations. There are several ways to interact with Spark SQL including SQL and the Dataset API. When computing a result the same execution engine is used, independent of which API/language you are using to express the computation. This unification means that developers can easily switch back and forth between different APIs based on which provides the most natural way to express a given transformation.

* 1. **SQL**

Spark SQL is used to execute SQL queries. Spark SQL can also be used to read data from an existing Hive installation. When running SQL from within another programming language the results will be returned as a [Dataset/DataFrame](https://spark.apache.org/docs/latest/sql-programming-guide.html#datasets-and-dataframes).

* 1. **Datasets and DataFrames**

A Dataset is a distributed collection of data. Dataset is a new interface added in Spark 1.6 that provides the benefits of RDDs (strong typing, ability to use powerful lambda functions) with the benefits of Spark SQL’s optimized execution engine. A Dataset can be [constructed](https://spark.apache.org/docs/latest/sql-programming-guide.html#creating-datasets) from JVM objects and then manipulated using functional transformations (map, flatMap, filter, etc.).

A DataFrame is a Dataset organized into named columns. It is conceptually equivalent to a table in a relational database or a data frame in R/Python, but with richer optimizations under the hood. DataFrames can be constructed from a wide array of [sources](https://spark.apache.org/docs/latest/sql-programming-guide.html#data-sources) such as: structured data files, tables in Hive, external databases, or existing RDDs.

* 1. **Interoperating with RDDs**

Spark SQL supports two different methods for converting existing RDDs into Datasets. The first method uses reflection to infer the schema of an RDD that contains specific types of objects. This reflection-based approach leads to more concise code and works well when you already know the schema while writing your Spark application.

The second method for creating Datasets is through a programmatic interface that allows you to construct a schema and then apply it to an existing RDD. While this method is more verbose, it allows you to construct Datasets when the columns and their types are not known until runtime.

* + 1. **Inferring the Schema Using Reflection**

The Scala interface for Spark SQL supports automatically converting an RDD containing case classes to a DataFrame. The case class defines the schema of the table. The names of the arguments to the case class are read using reflection and become the names of the columns. Case classes can also be nested or contain complex types such as Seqs or Arrays. This RDD can be implicitly converted to a DataFrame and then be registered as a table. Tables can be used in subsequent SQL statements.

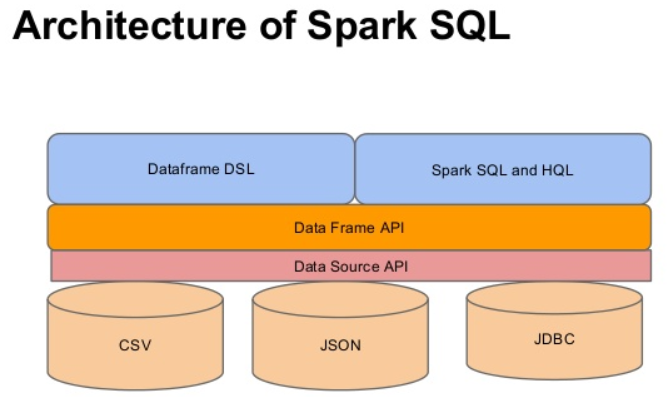
* + 1. **Programmatically Specifying the Schema**

When case classes cannot be defined ahead of time (for example, the structure of records is encoded in a string, or a text dataset will be parsed, and fields will be projected differently for different users), a DataFrame can be created programmatically with three steps:

Create an RDD of Rows from the original RDD;

Create the schema represented by a StructType matching the structure of Rows in the RDD created in Step 1.

Apply the schema to the RDD of Rows via createDataFrame method provided by SparkSession.

****

1. **Database: NoSQL HBASE DB**

HBASE is a non-relational (NoSQL) database that runs on top of HDFS

Apache HBase is an open source NoSQL database that provides real-time read/write access to those large datasets.

HBase scales linearly to handle huge data sets with billions of rows and millions of columns, and it easily combines data sources that use a wide variety of different structures and schemas. HBase is natively integrated with Hadoop and works seamlessly alongside other data access engines through YARN.

HBase is a column-oriented database management system that runs on top of Hadoop Distributed File System[(HDFS)](https://www.ibm.com/analytics/hadoop/hdfs). It is well suited for sparse data sets, which are common in many big data use cases.

Unlike relational database systems, HBase does not support a structured query language like SQL; in fact, HBase isn’t a relational data store at all. HBase applications are written in Java much like a typical [Apache™ MapReduce](https://www.ibm.com/analytics/hadoop/mapreduce) application. HBase does support writing applications in [Apache™ Avro™](https://www.ibm.com/analytics/hadoop/avro), REST, and Thrift.

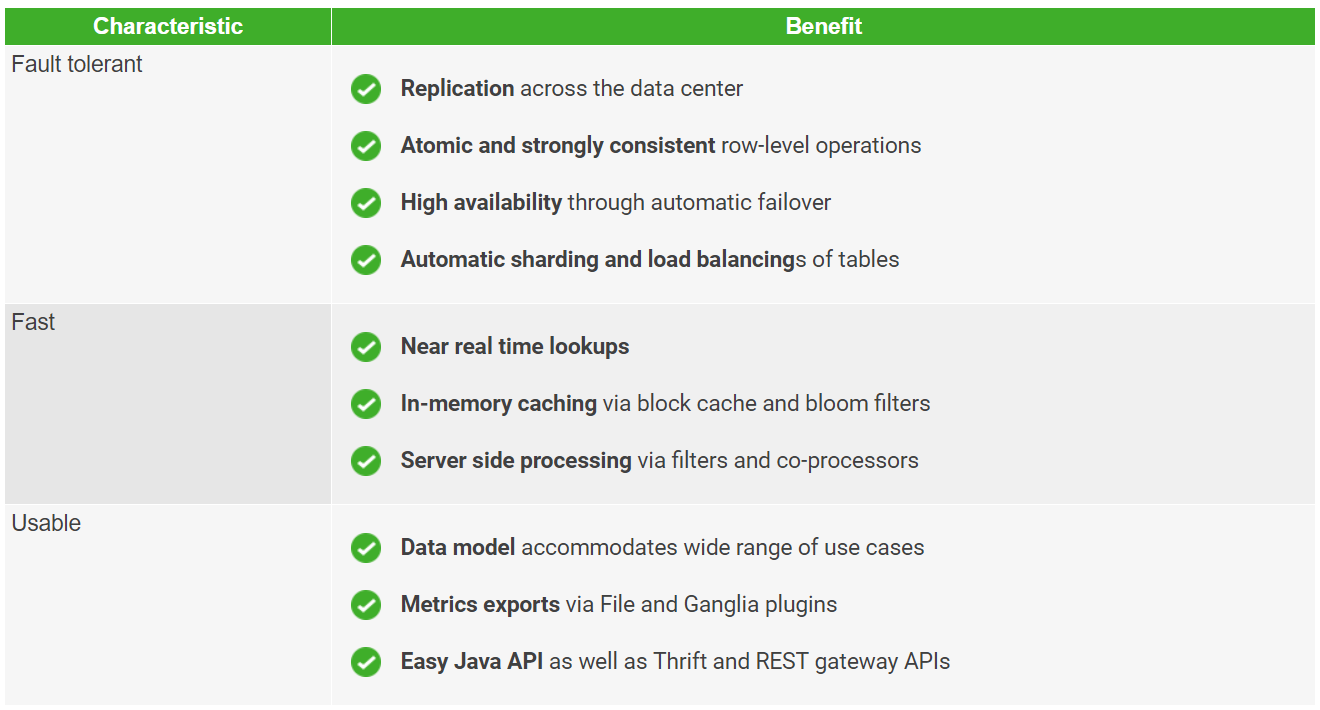
An HBase system comprises a set of tables. Each table contains rows and columns, much like a traditional database. Each table must have an element defined as a Primary Key, and all access attempts to HBase tables must use this Primary Key.

Avro, as a component, supports a rich set of primitive data types including: numeric, binary data and strings; and a number of complex types including arrays, maps, enumerations and records. A sort order can also be defined for the data.

* 1. **WHAT HBASE DOES**

Apache HBase provides random, real time access to your data in Hadoop. It was created for hosting very large tables, making it a great choice to store multi-structured or sparse data. Users can query HBase for a particular point in time, making “flashback” queries possible. These following characteristics make HBase a great choice for storing semi-structured data like log data and then providing that data very quickly to users or applications integrated with HBase.

Enterprises use Apache HBase’s low latency storage for scenarios that require real-time analysis and tabular data for end user applications. Apache HBase provides that super low-latency access over an enormous, rapidly changing data store.



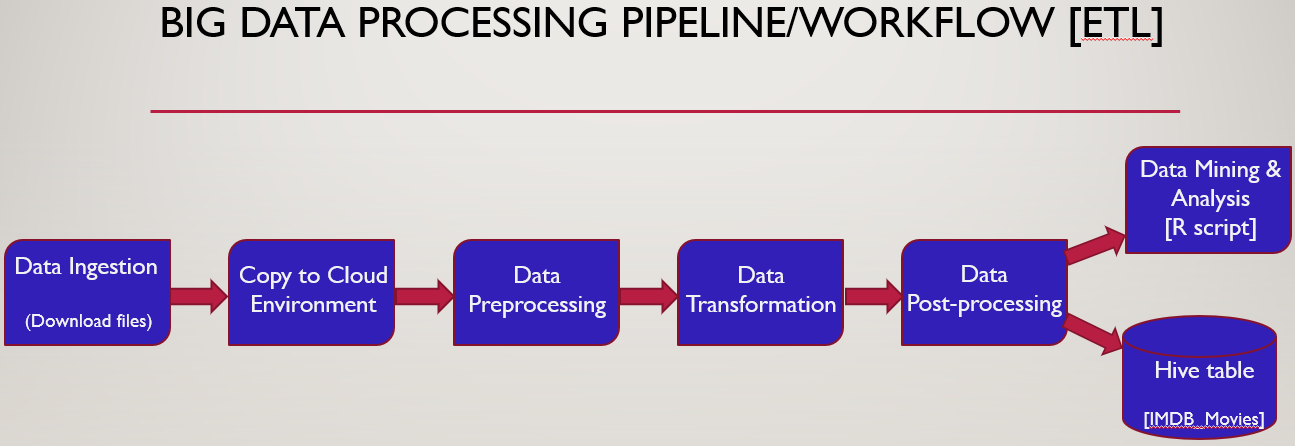
1. **Data Mining & Analysis: SQL and R on Big Data**

R is an open-source data analysis environment and programming language. The process of converting data into knowledge, insight and understanding is Data analysis, which is a critical part of statistics. For the effective processing and analysis of big data, it allows users to conduct a number of tasks that are essential. R consists of numerous ready-to-use statistical modelling algorithms and machine learning which allow users to create reproducible research and develop data products. Although big data processing may be accomplished with other tools as well, it is when one steps on to the data analysis that R really stands unique, owing to the huge amount of built-in statistical formulae and third-party algorithms available.

Business value is not generated by stored data and this is true as for traditional databases, data warehouses, also for the new technologies like Hadoop for storing big data. Once the data is appropriately stored, it can be analyzed, and thus immense value can be created. In-memory analytics, in database analytics and a variety of analysis, technologies and products have arrived that are mainly applicable to big data.

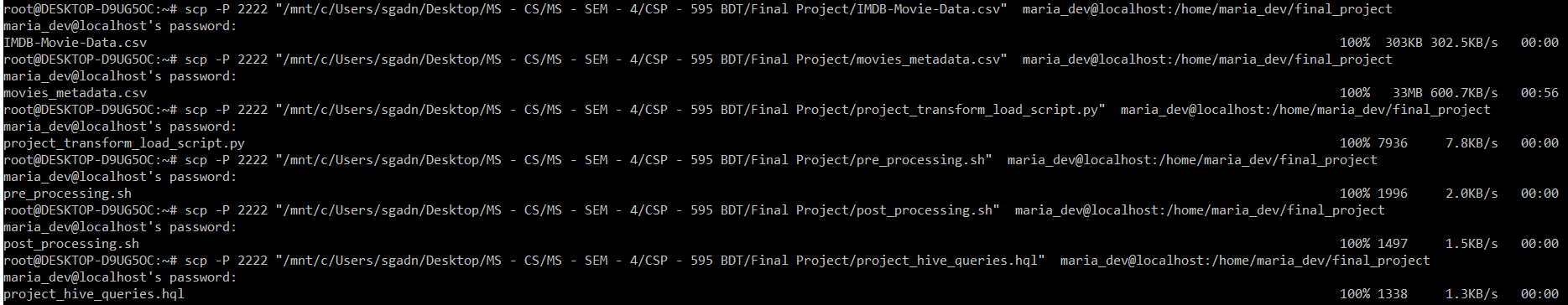
For statistical data analysis, R is an open source software platform. Largely because of its open source nature, R is speedily adopted by statistics departments in universities around the world, attracted by its extensible nature as a platform for academic research. Free in cost surely played a role as well. And it wasn’t long before researchers in data science, statistics and machine learning started to publish papers in academic journals along with R code applying their new methods. R builds this process very easily and anyone can produce an R package to CRAN that stands for Comprehensive R Archive Network and make it available to everyone. An excellent open-source interactive development environment has been created by R Studio for the R language, further boosting the productivity of R users everywhere.

* **Section – III: Project Execution/Results**

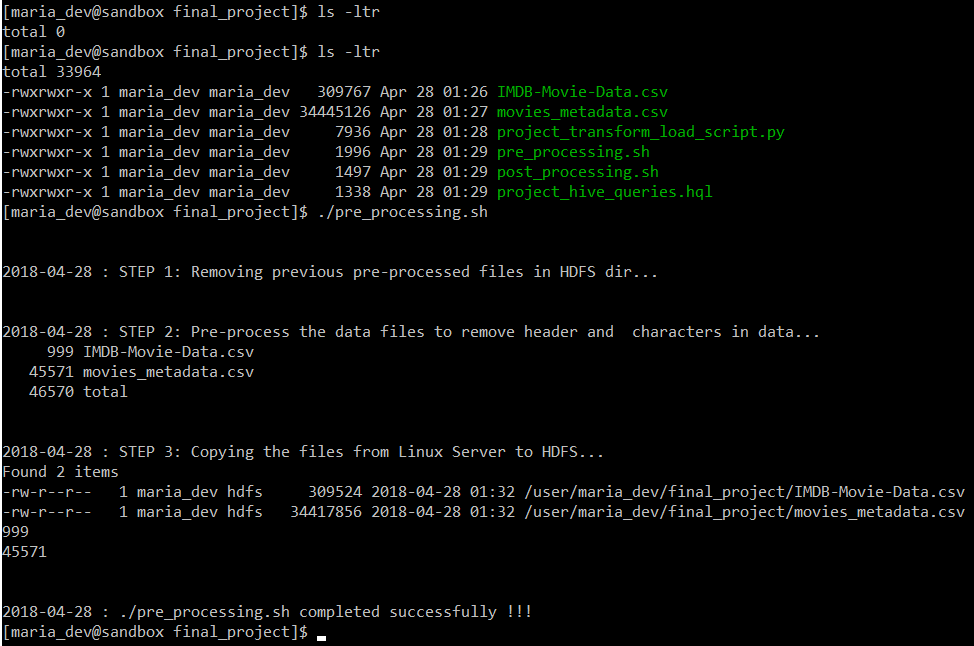
****

*NOTE: In the Initial Project Draft and the Literature Review, I had planned to use HBASE as the NoSQL database to load the transformed data, but during the development process, I discovered creating a Hive table was straight-forward with Spark SQL rather than dealing with an API to get the data loaded into a HBASE database. Hence, during the big data pipeline development Hive was used as the database to load the transformed data and enable HQL queries over it.*

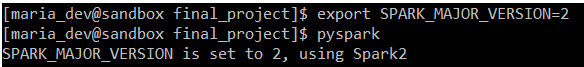
1. **STEP 1: Download the source dataset from Kaggle to Local Disk**
   1. IMDB dataset => https://www.kaggle.com/tomiandrep/imdb-filmid/data
   2. Movie Lens dataset => https://www.kaggle.com/rounakbanik/the-movies-dataset/data
2. **STEP 2: Copy the source datasets from Local Disk to Cloud environment [Linux Server]**

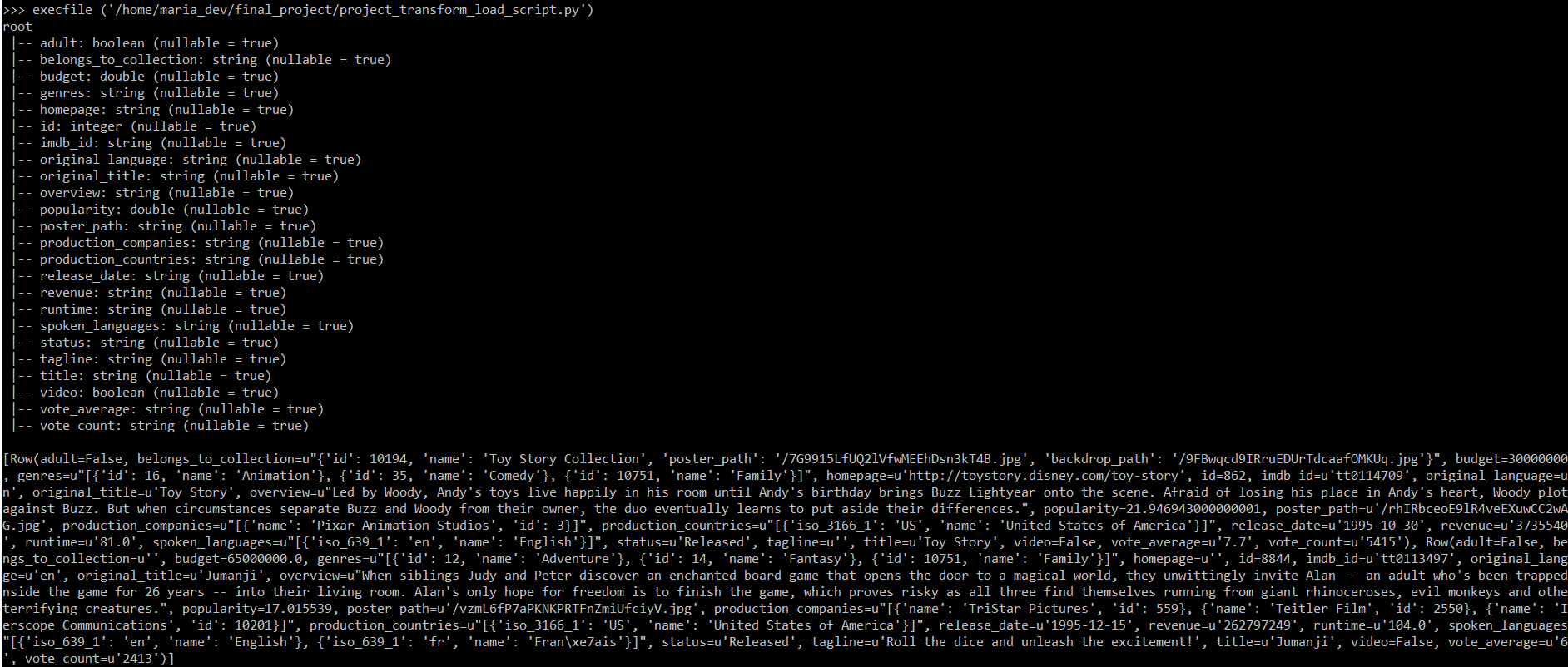
****

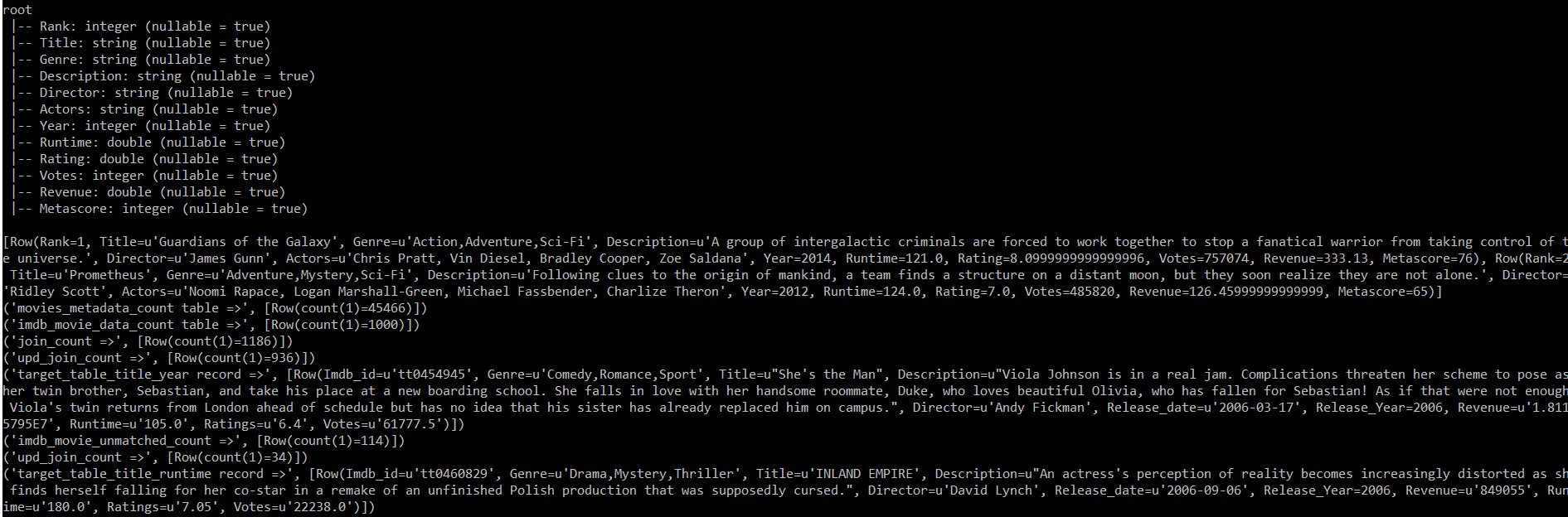
1. **STEP 3: Run Data Preprocessing script [Unix Shell Script]**

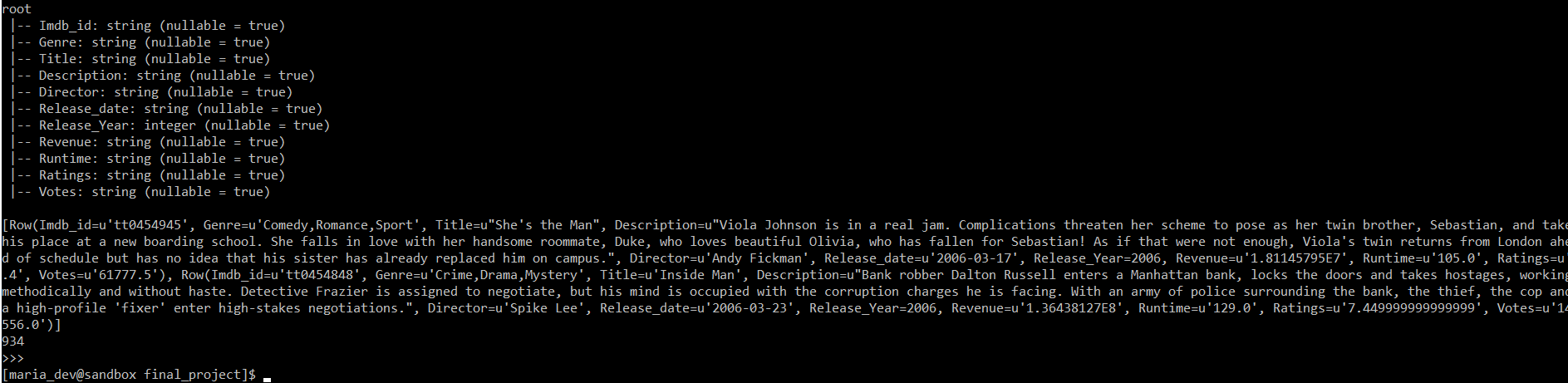
****

1. **STEP 4: Run the Data Transformation script [pyspark script]**

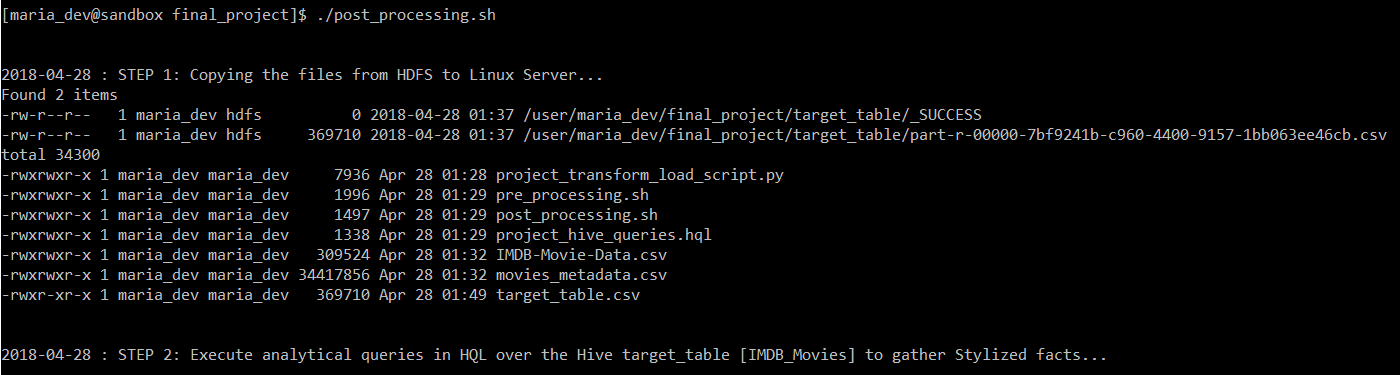
****

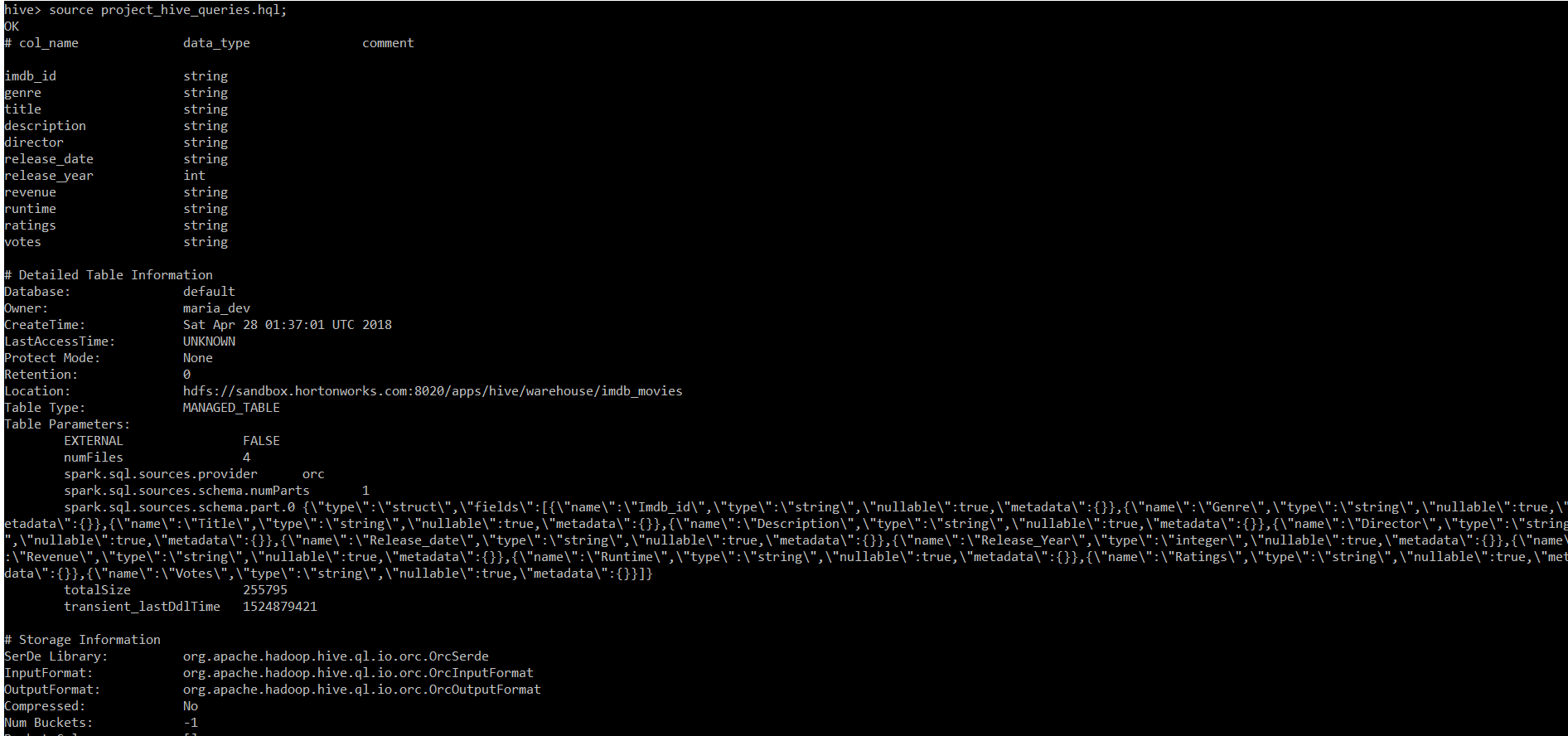
****

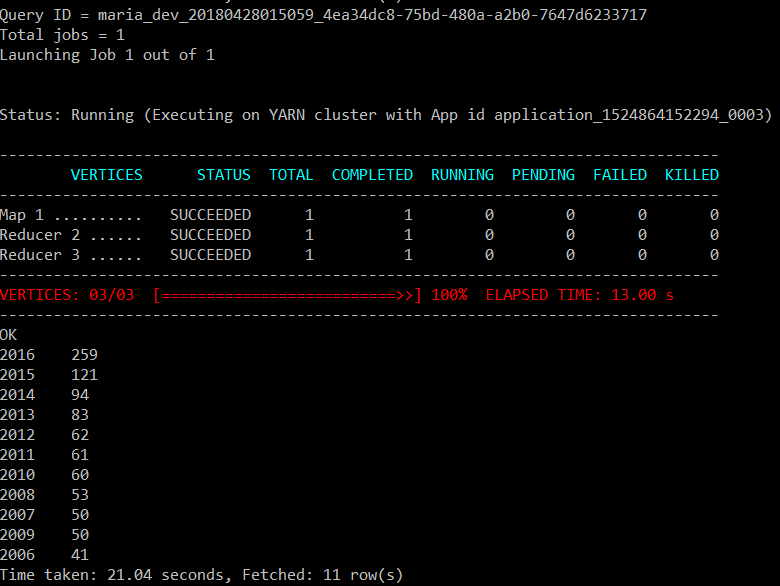
****

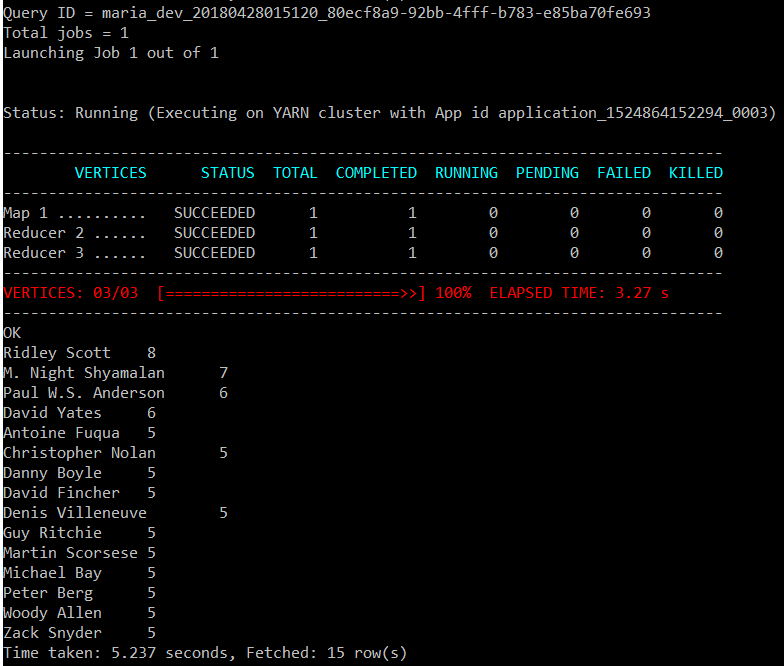
****

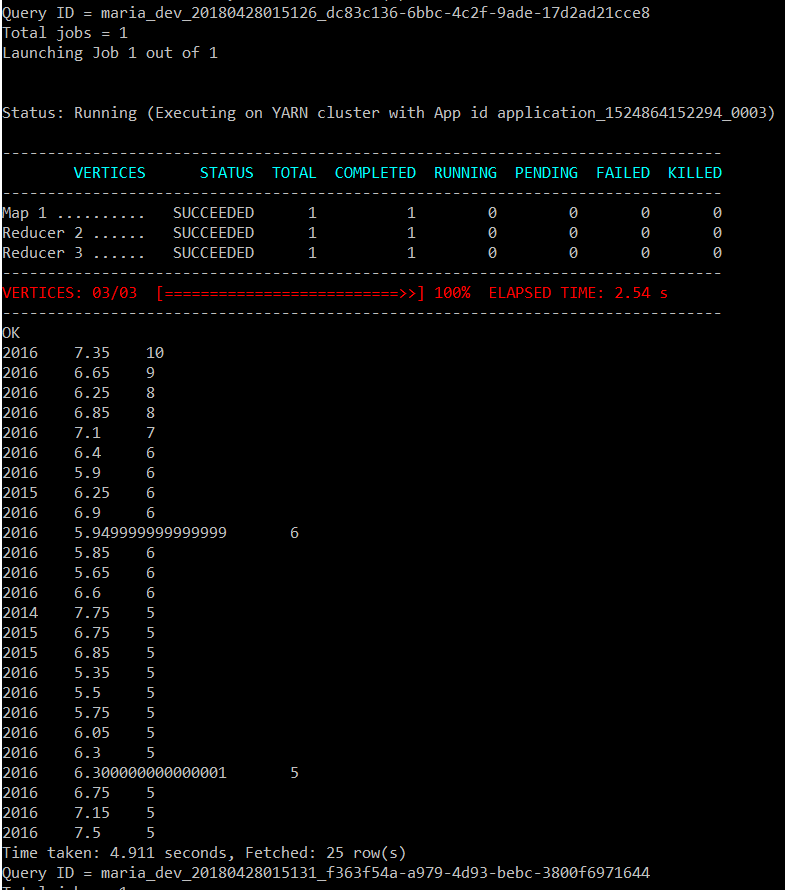
1. **STEP 5: Run Data Post-processing script [Unix Shell Script]**

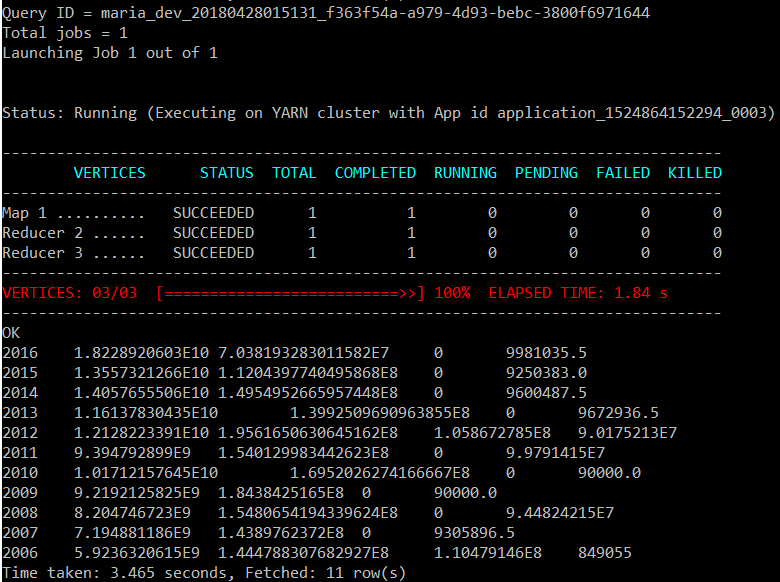
****

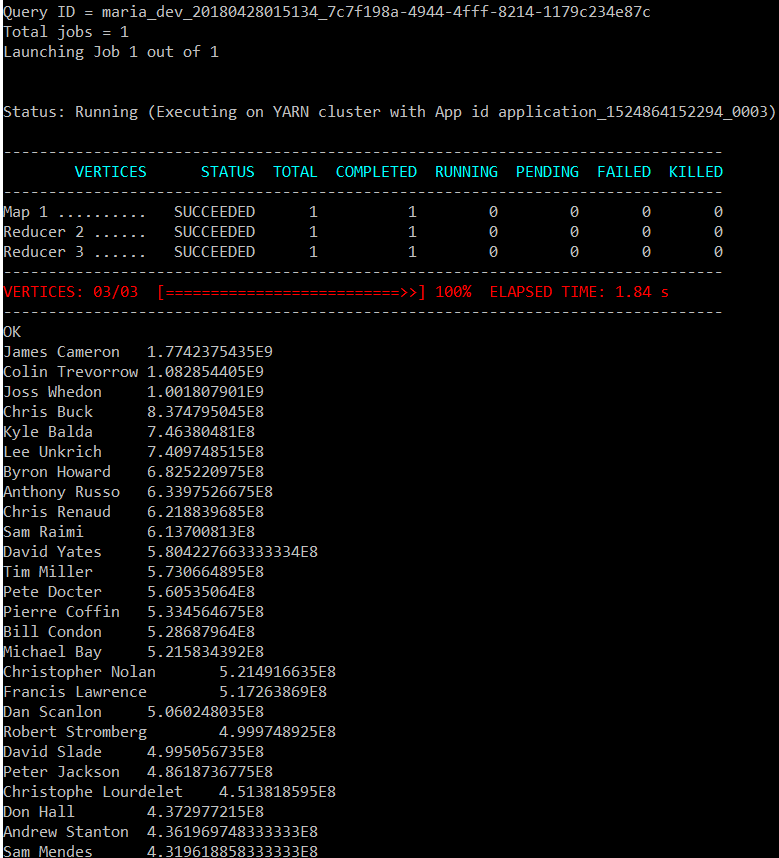
****

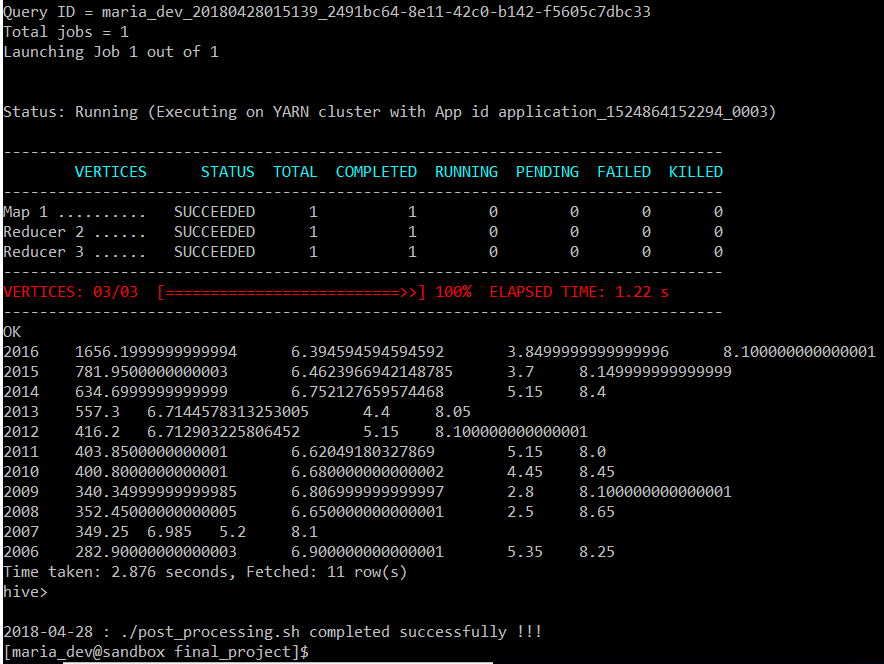
****

****

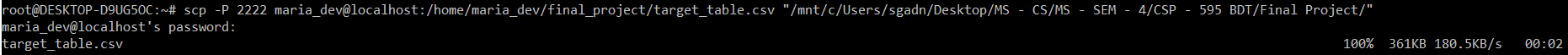
****

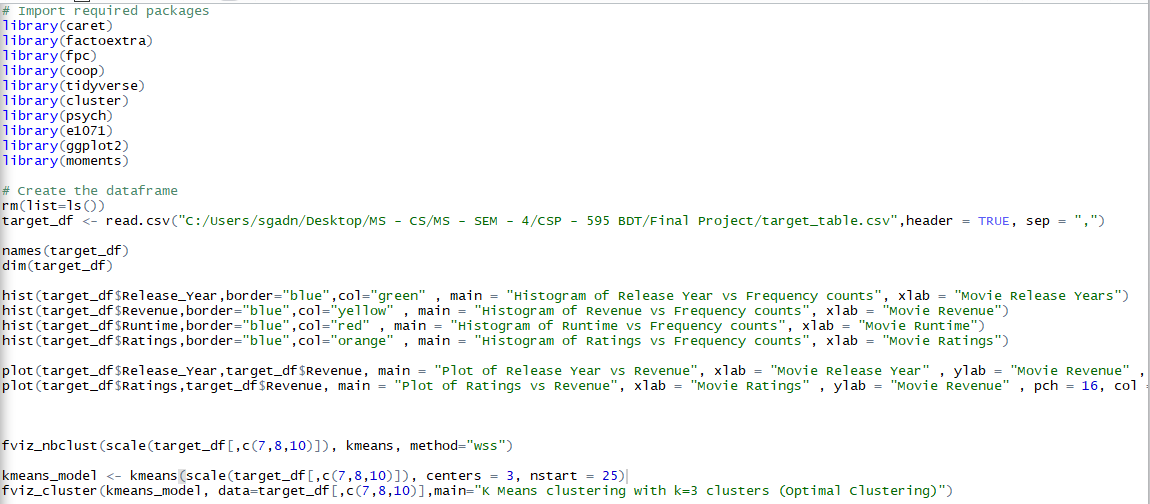
****

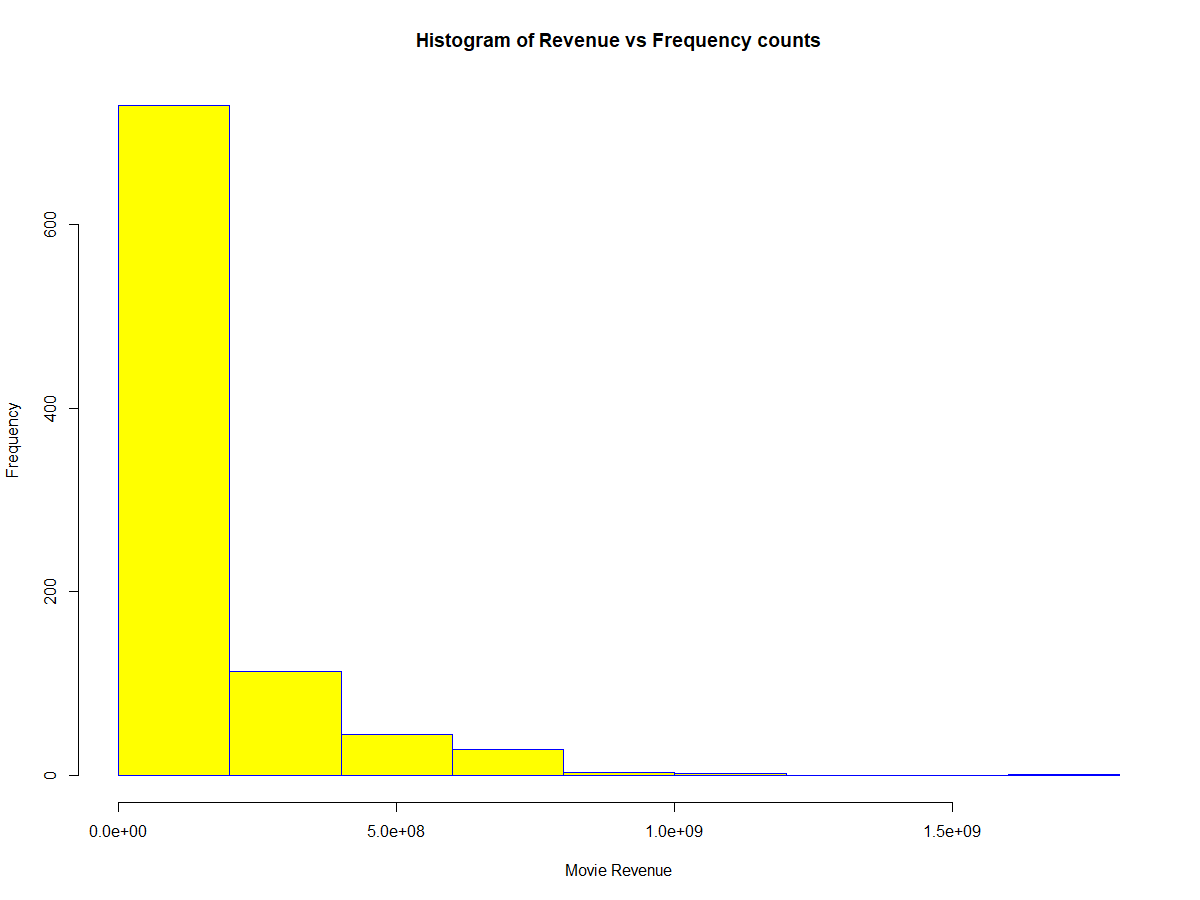
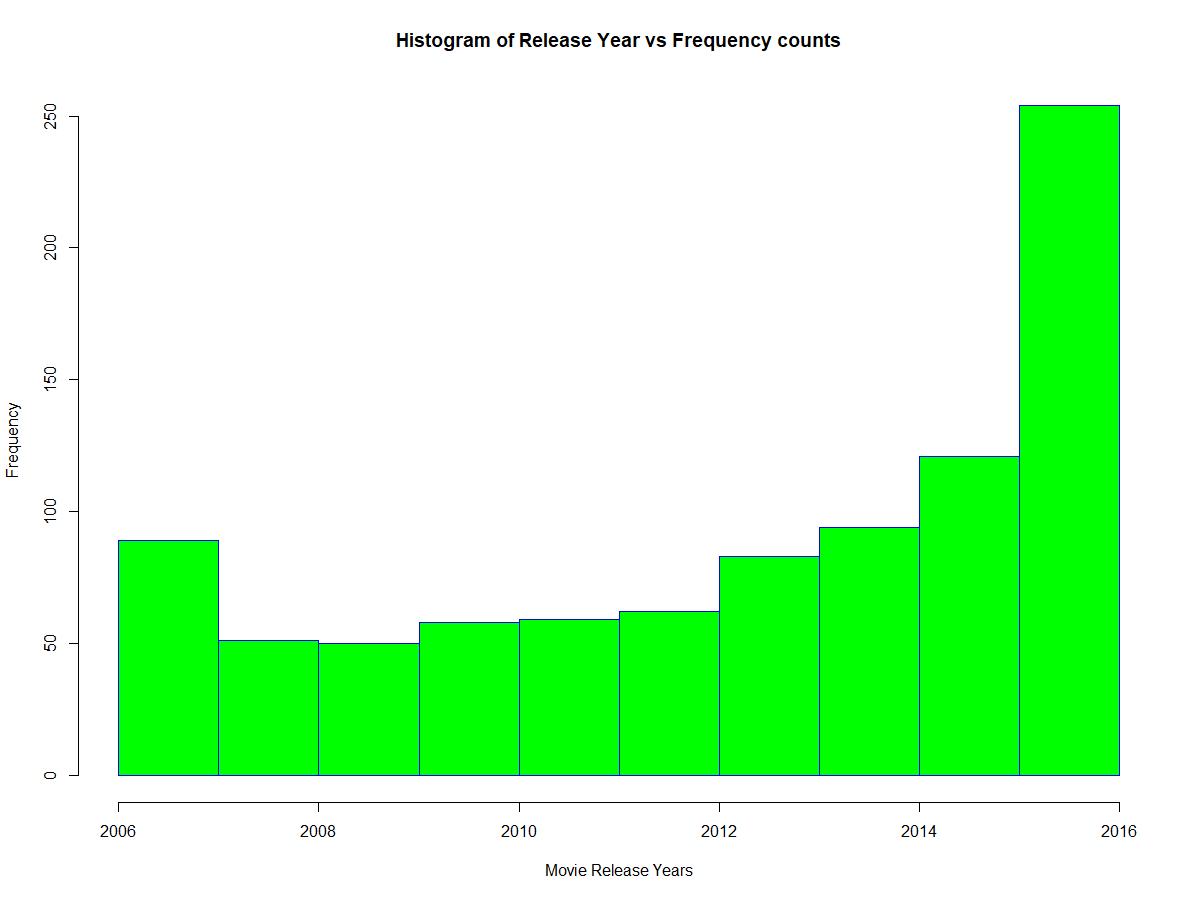
****

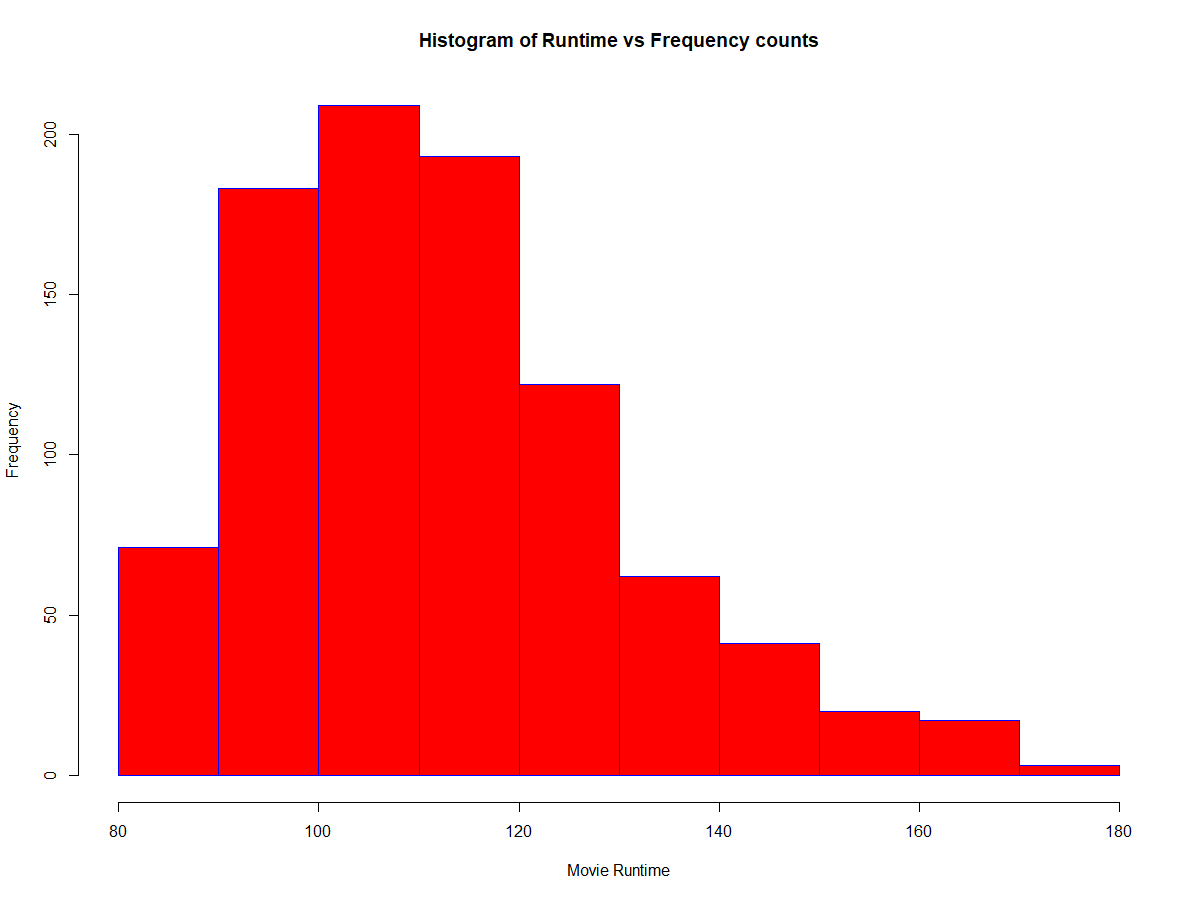
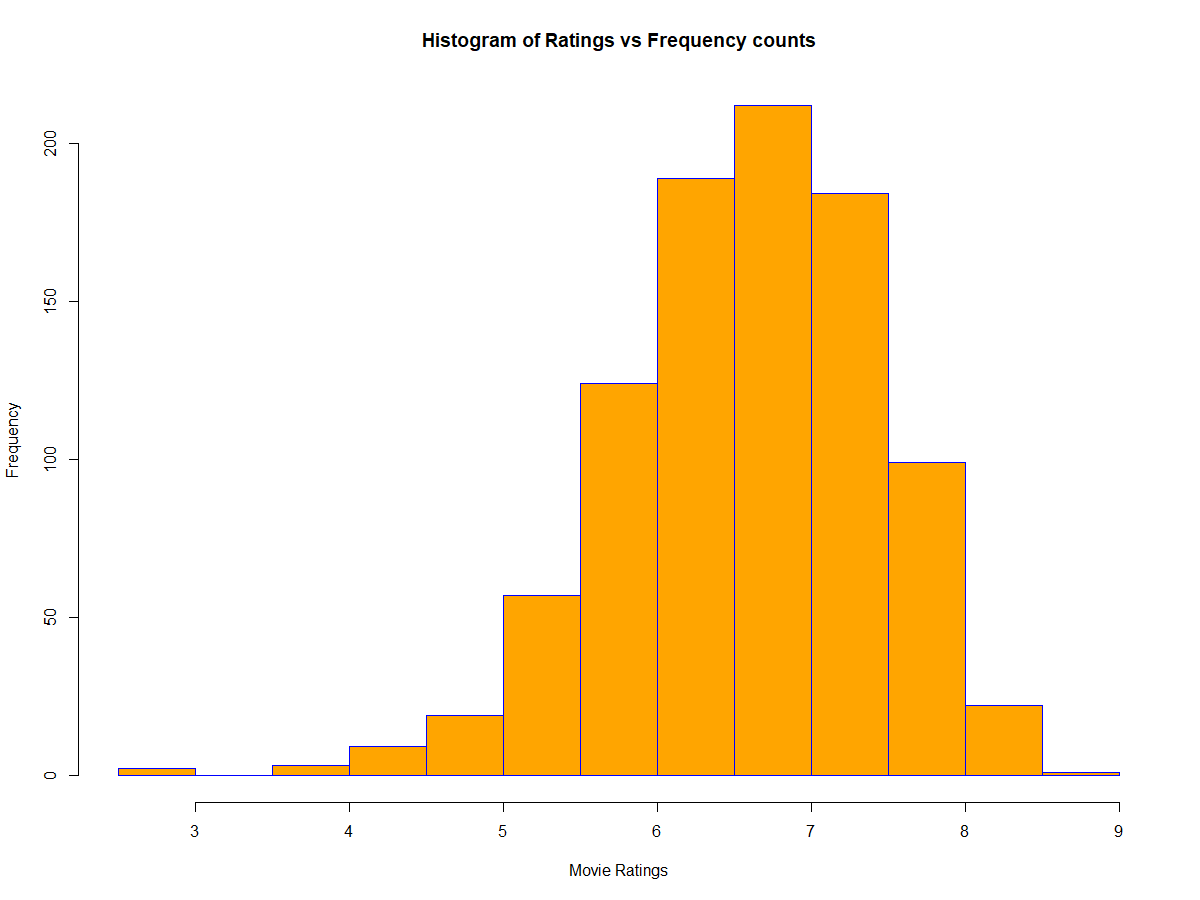
****

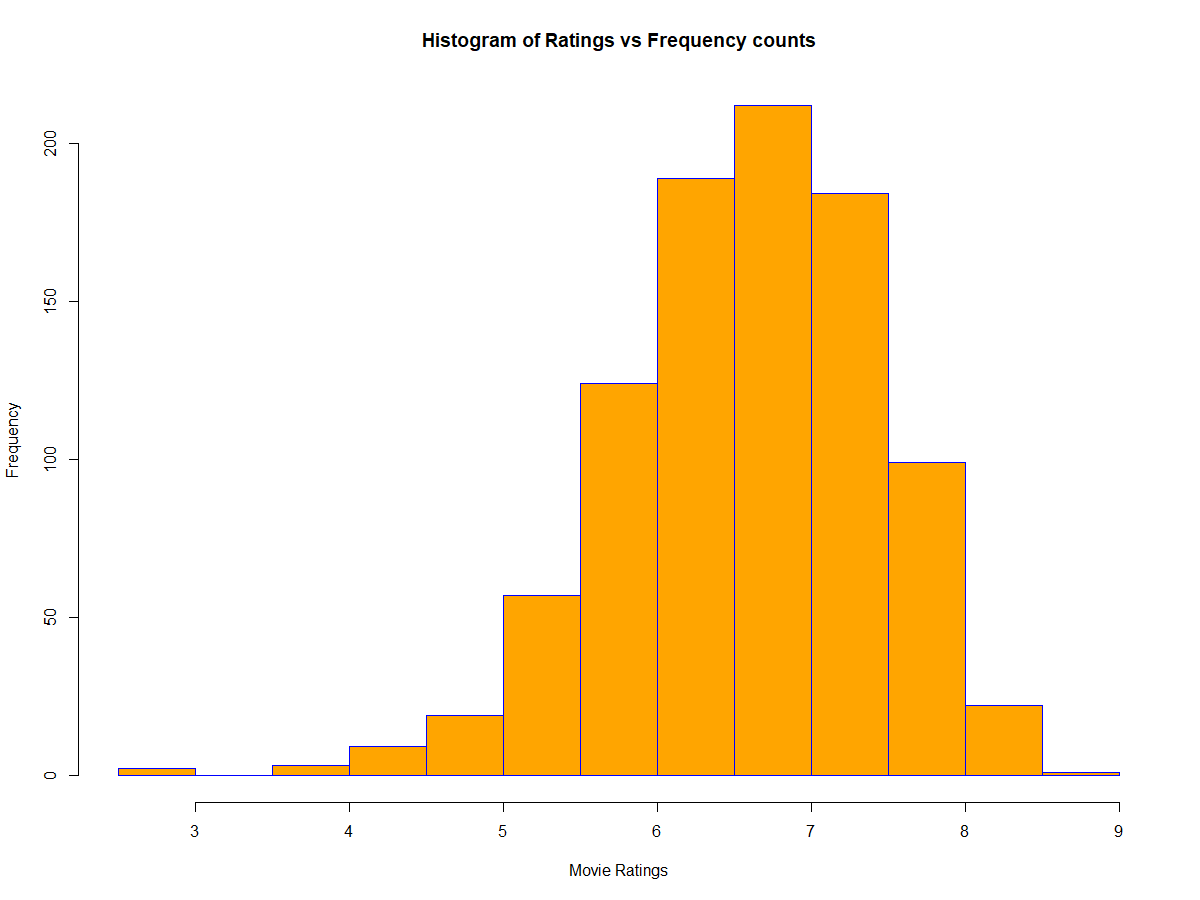
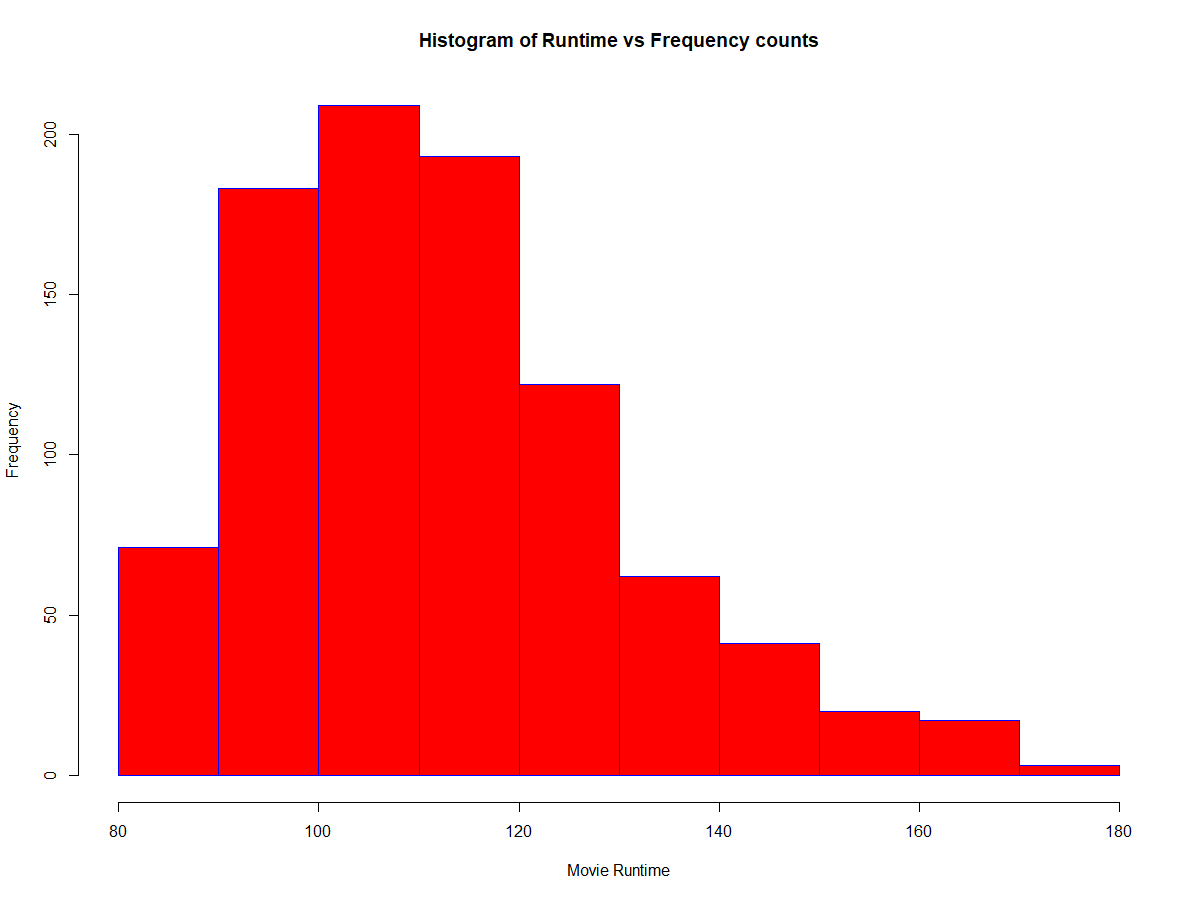
1. **STEP 6: Copy the target dataset from Cloud environment [Linux Server] to Windows & Execute R script to perform Exploratory Data Analysis, Unsupervised Learning [K-Means clustering]**

****

****

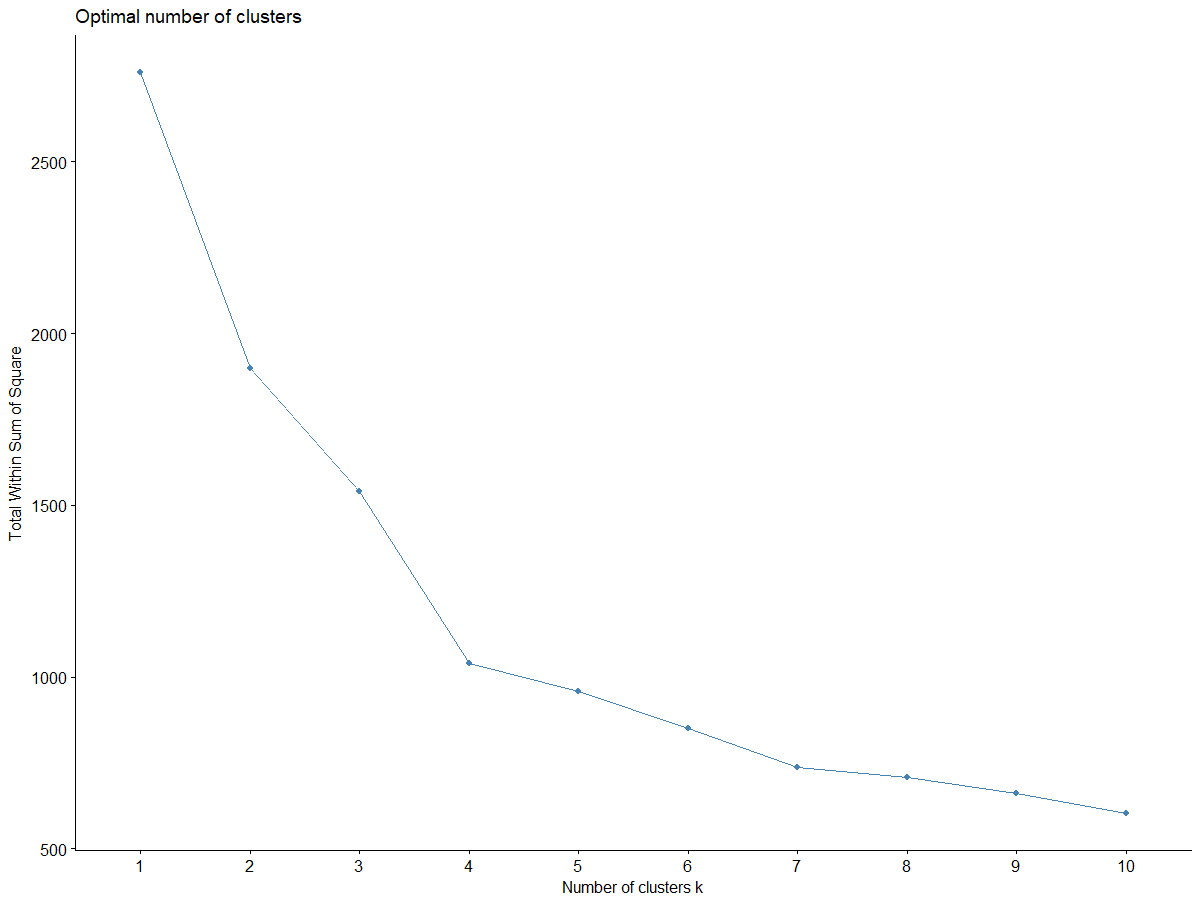
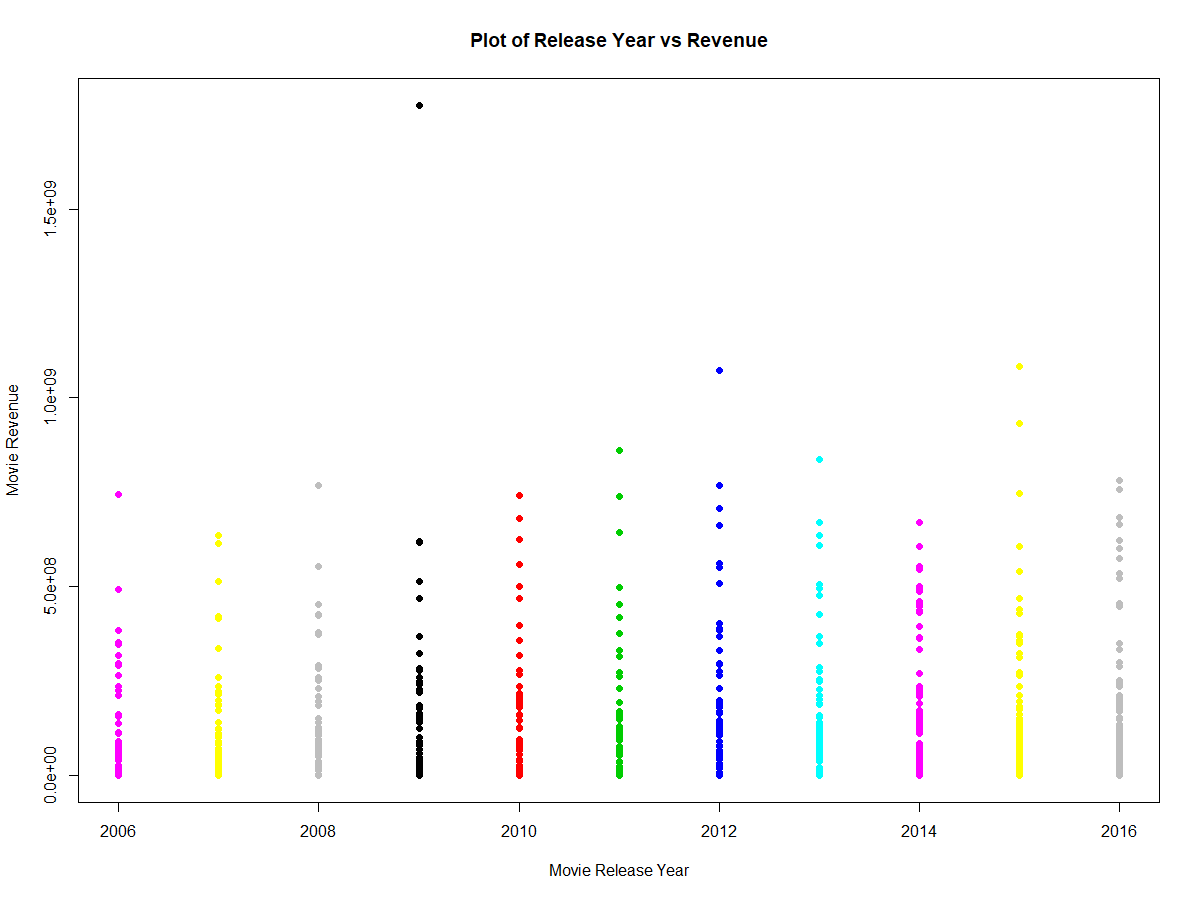
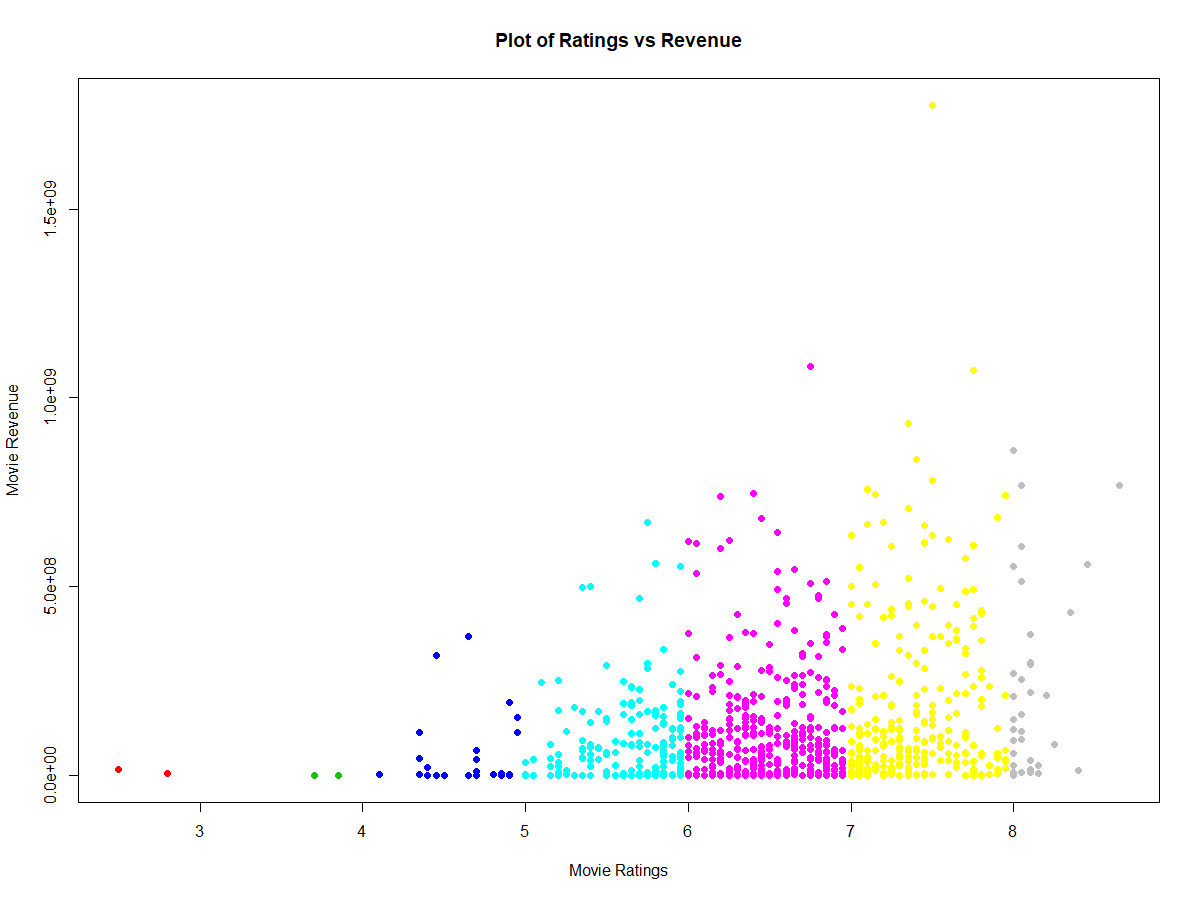
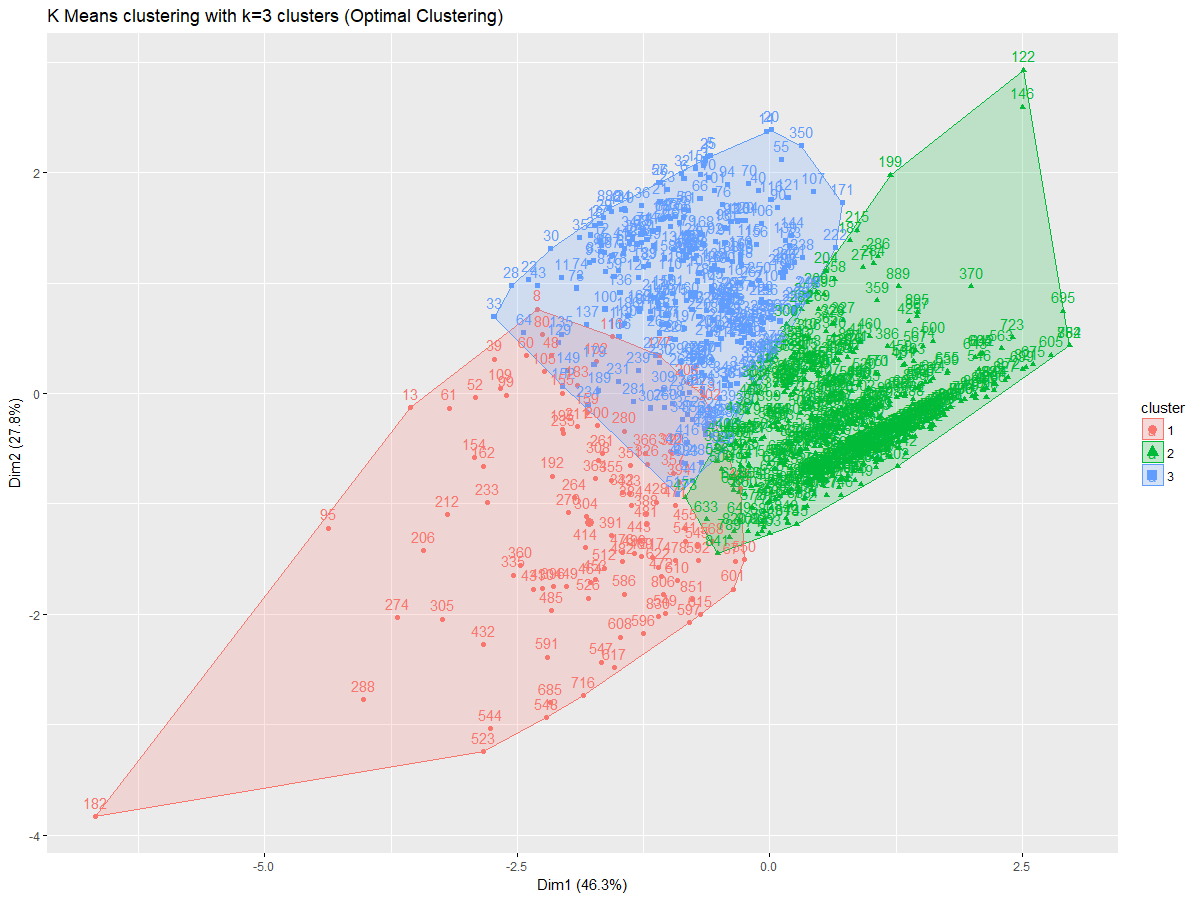
****

****

****

***Following are the details for the above Plots/Histograms:***

* *Top Left: Histogram of Release Year vs Count/Frequency of movies [Analysis/Distribution by year]*
* *Top Right: Histogram of Revenue vs Count/Frequency of movies [Analysis/Distribution by Revenue]*
* *Bottom Left: Histogram of Runtime vs Count/Frequency of movies [Analysis/Distribution by Runtime]*
* *Bottom Right: Histogram of Ratings vs Count/Frequency of movies [Analysis/Distribution by Rating]*

****

***Following are the details for the above plots:***

* *Top Left : Plot of Release Year vs Revenue of movies*
* *Bottom Left : Plot of Ratings vs Revenue of movies*
* *Top Right : Scree Plot to identify the optimal number of clusters*
* *Bottom Right : Clusters resulting on applying K-Means clustering [Unsupervised Learning] of the*

*result dataset.*

* **Section – IV: Reference resources**
* Some Kernels and existing sample code available on the respective datasets Kaggle link:
  + <https://www.kaggle.com/tomiandrep/imdb-filmid/data>
  + <https://www.kaggle.com/rounakbanik/the-movies-dataset/data>
  + [https://hadoop.apache.org/docs/r1.0.4/hdfs\_design.html#Introduction](https://hadoop.apache.org/docs/r1.0.4/hdfs_design.html%23Introduction)
  + <https://hortonworks.com/apache/hdfs/>
  + <https://spark.apache.org/docs/latest/sql-programming-guide.html>
  + <https://www.ibm.com/analytics/hadoop/hbase>
  + <https://hortonworks.com/apache/hbase/>
  + <https://irjet.net/archives/V3/i7/IRJET-V3I713.pdf>