# **CIS4560 Term Project Tutorial**

#### **Authors:** [**Cesar Lopez**](https://www.linkedin.com/in/cesarlopez200/) **, Michael Cheung, Brian Wonse, Tan Huynh Instructor:** [**Jongwook Woo**](https://www.linkedin.com/in/jongwook-woo-7081a85) **Date: 05/18/2017**

**Lab Tutorial**

Cesar Lopez (clope151@calstatela.edu)

12/01/2018

**IMDB: Intro to Data Analysis using Hive**

**Objectives**

In this hands-on lab, you will be introduced to data analysis related topics with HDFS and HIVE:

* Download online files with command-line utility wget.
* Unzip files with gzip.
* Rename files with mv.
* Create Directories in HDFS.
* Put files into HDFS Directory.
* List data in HDFS Directory.
* Create tables in HDFS.
* Load data into HDFS tables.
* Join HDFS Tables
* Place files in HDFS directory into local server.
* Download data from local server to local machine

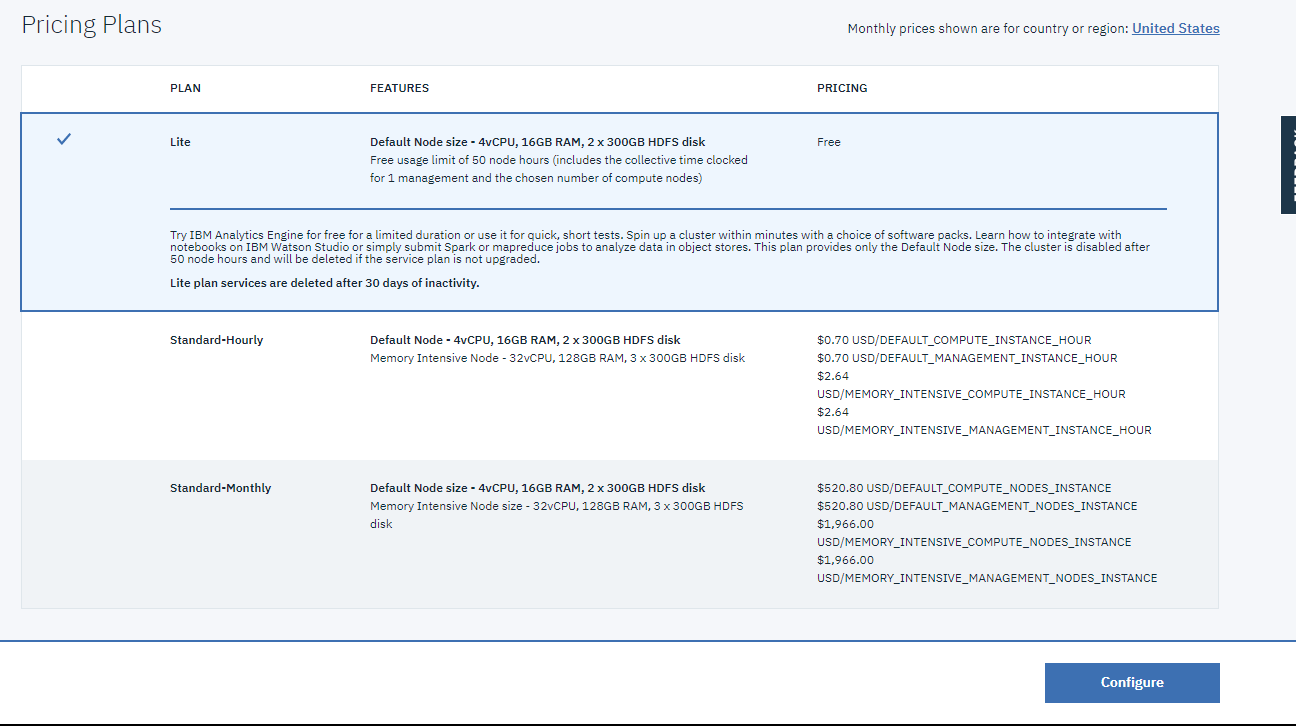
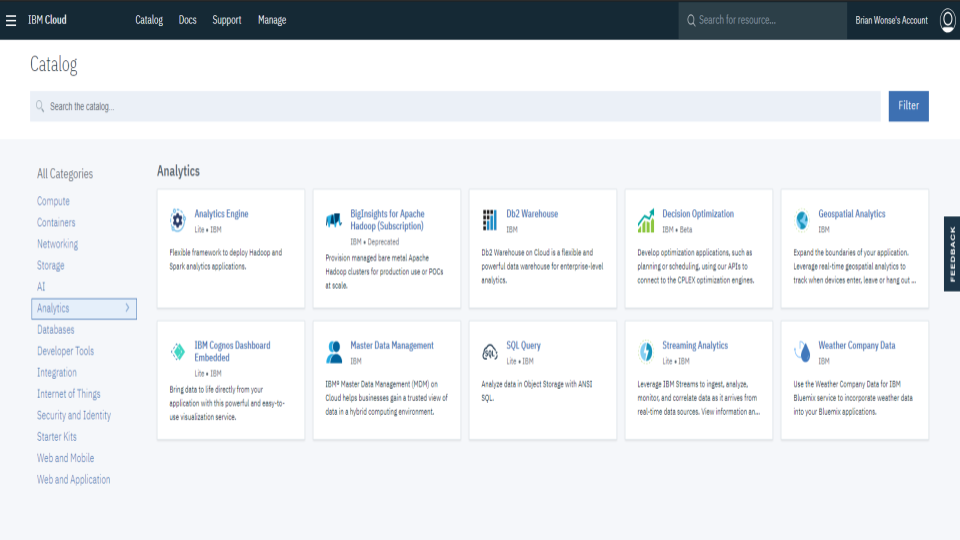
**Platform Spec**

* IBM Cloud Analytics Engine: Lite Plan
* RAM Size: 16GB
* # of CPU cores: 4
* # of nodes: 1
* Total Memory Size: 600GB HDFS Space

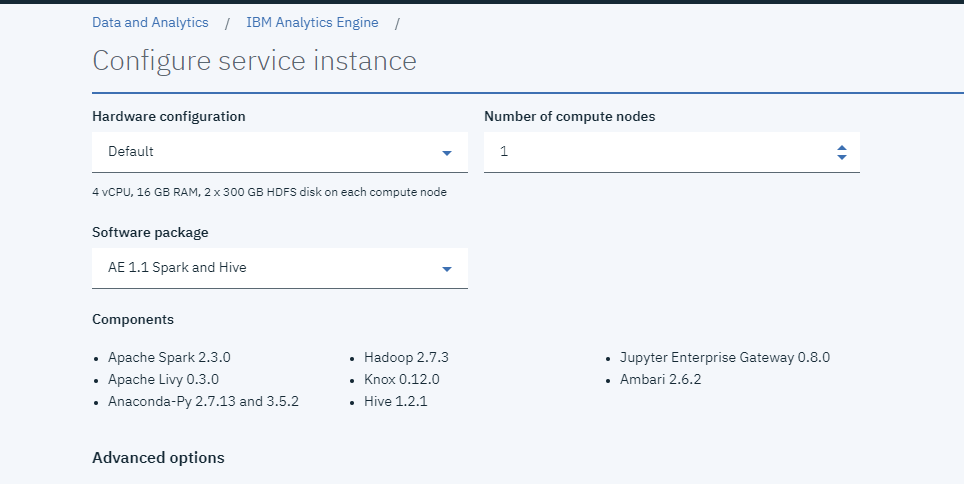
Step 0: Creating Analytics Engine on IBM Cloud

**IBM Cloud’s Analytic Engine is a server preset with Hive and Spark Services. Once we create our engine and enable**

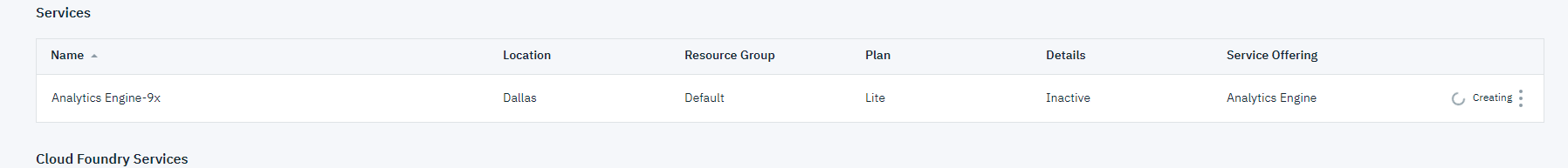
1. In IBM Cloud Catalog. Go to Analytics category. And select Analytics Engine. Then Choose the Lite Plan

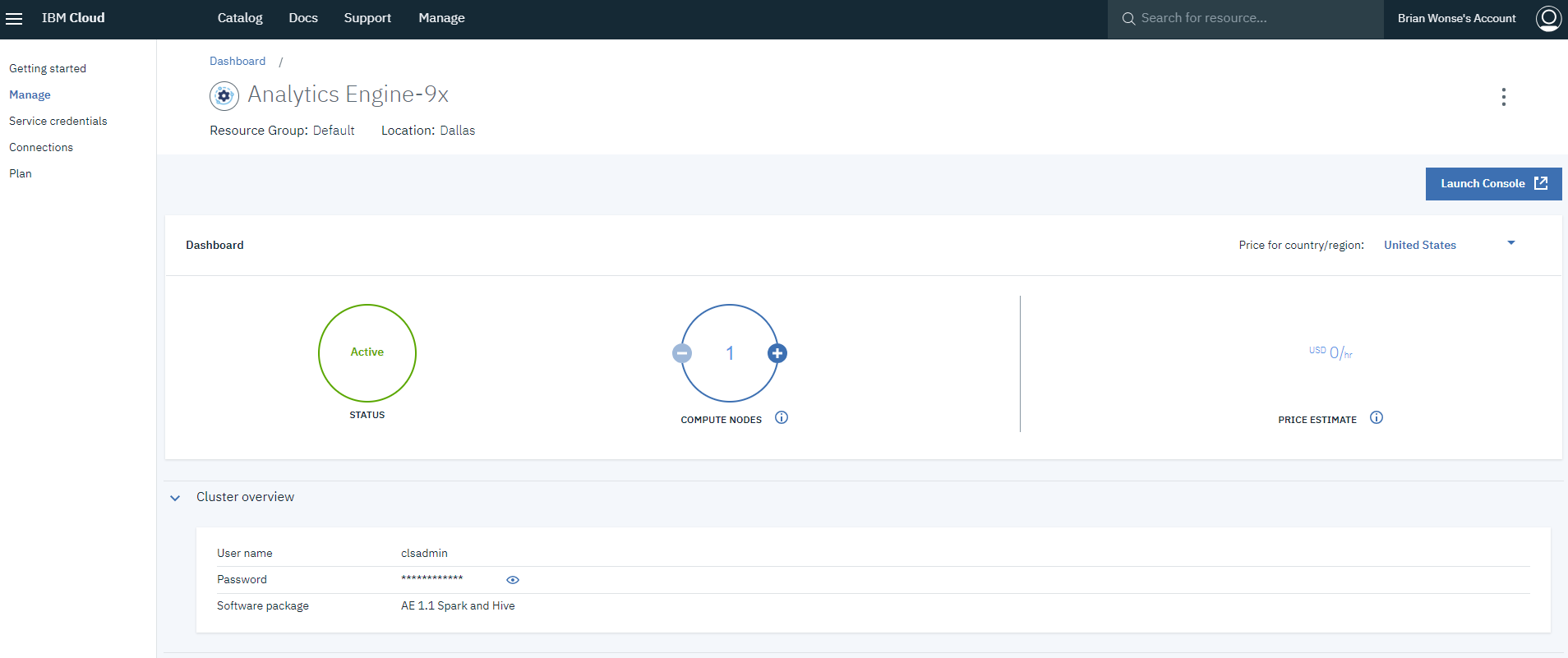


1. When configuring service instance make sure to include the software package that includes: Spark and Hive.

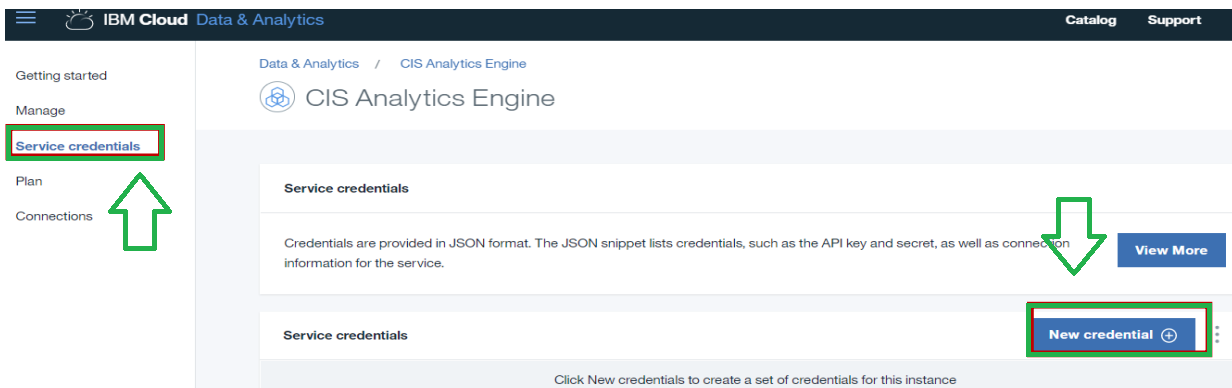


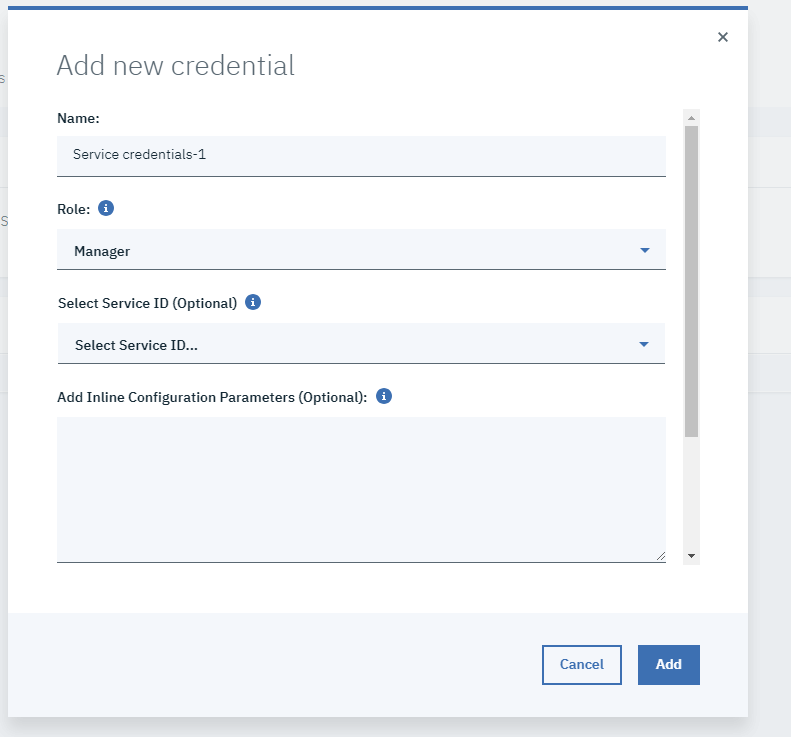
1. Creating the Analytics Engine





1. Creating Credentials





1. Viewing credentials will give us the ssh statement needed to remotely access the analytics engine.

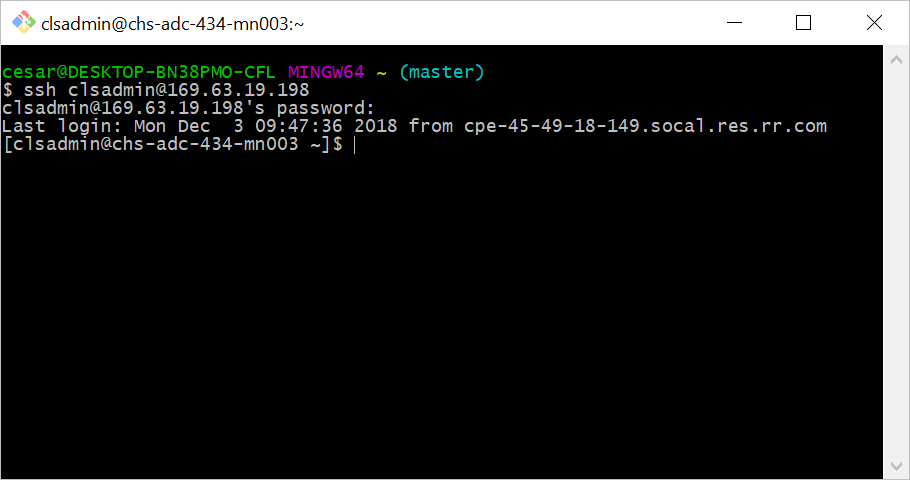


Step 1: Download online files through shell client

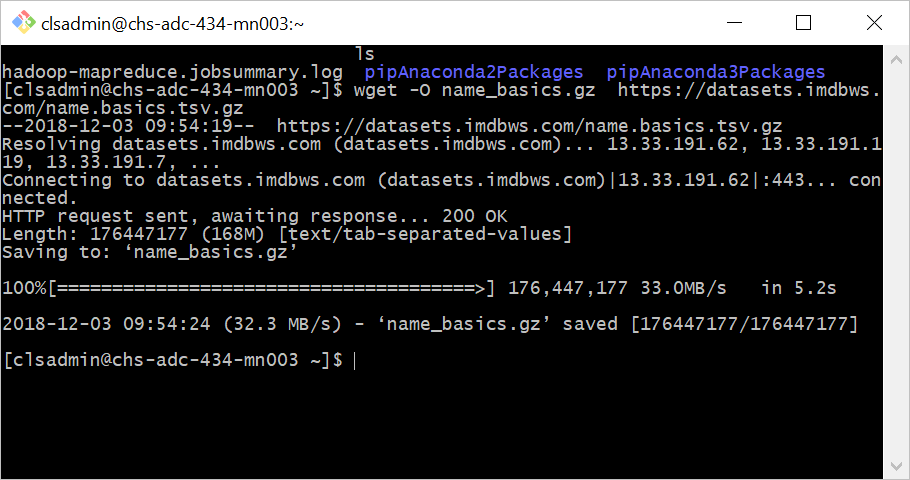
**Before any data can be analyzed, we must download our files. We will be using Git Bash terminal as our working environment.**

1. Open Shell Client Git Bash
2. Connect to cluster through SSH. This requires the ssh command and the location of the host name.

$ ssh <host Name Location>



1. Login to client with password (if required) Note: password doesn't appear while typed.
2. Download files with wget command. Wget stands for “Web Get” and will be used to pull online files into our cluster.



$ wget -O name\_basics.gz

https://datasets.imdbws.com/name.basics.tsv.gz

$ wget -O title\_akas.gz

https://datasets.imdbws.com/title.akas.tsv.gz

$ wget -O title\_basics.gz

https://datasets.imdbws.com/title.basics.tsv.gz

$ wget -O title\_crew.gz

https://datasets.imdbws.com/title.crew.tsv.gz

$ wget -O title\_episode.gz

https://datasets.imdbws.com/title.episode.tsv.gz

$ wget -O title\_principals.gz

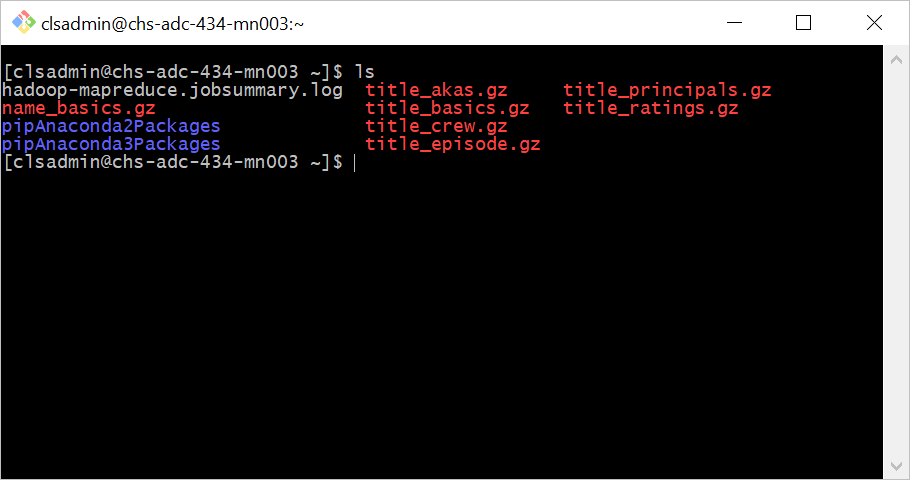
https://datasets.imdbws.com/title.principals.tsv.gz

$ wget -O title\_ratings.gz

https://datasets.imdbws.com/title.ratings.tsv.gz

1. List files in local server. Notice that files are compressed

$ ls



Step 2: Preparing Downloaded files

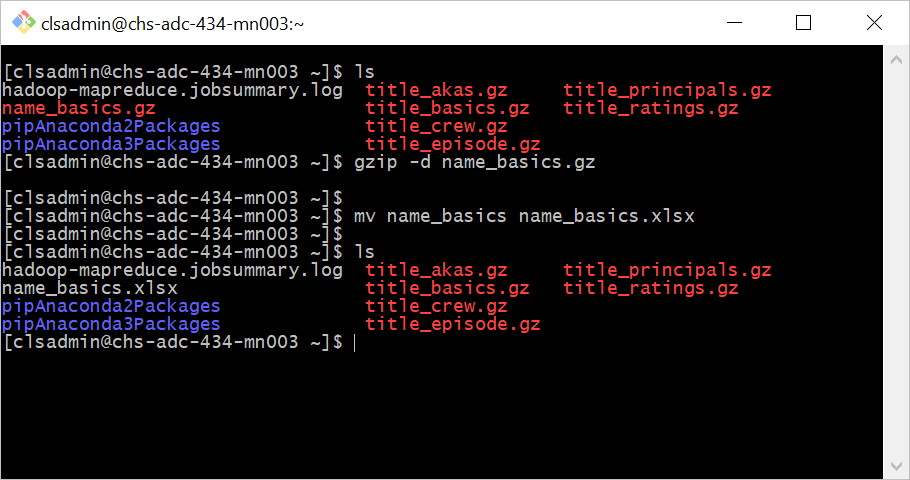
**Now that we downloaded our data, we must prepare the files so that they can be analyzed better. We downloaded our files as compressed files. We must decompress and rename each file that was downloaded.**

1. Decompress files with gzip command

$ gzip -d name\_basics.gz

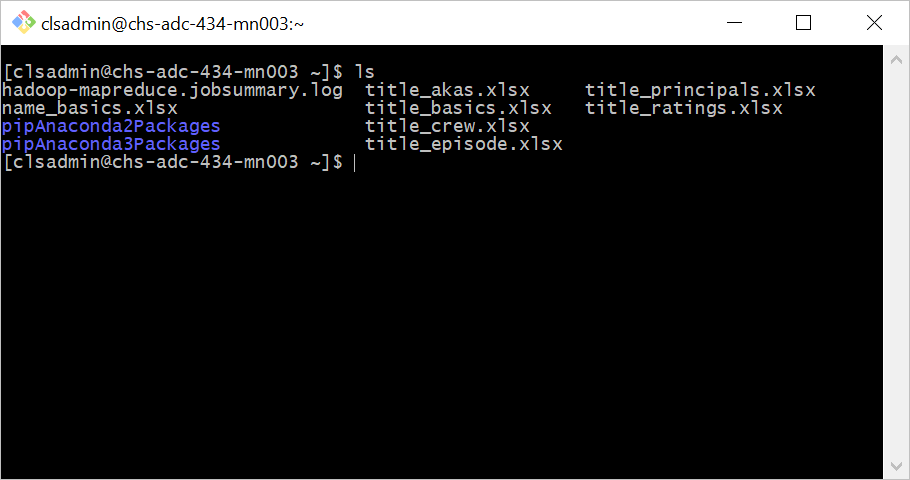
1. Rename files with mv command.

$ mv name\_basics name\_basics.xlsx



1. Repeat steps for:
   1. title\_akas.gz
   2. title\_basics.gz
   3. title\_crew.gz
   4. title\_episode.gz
   5. title\_principals.gz
   6. Title\_ratings.gz
2. List files in local server. Notice that files that were compressed are no longer in system. Only de-compressed files are in our local server.

$ ls



Step 3: Moving Files to HDFS

**We now have our files prepared in our clusters default file system. We want to move our files to the Hadoop File System (HDFS). Once a file is in HDFS it is split into blocks and distributed to different nodes in the current Hadoop cluster.**

1. Make a directory in hdfs

$ hdfs dfs -mkdir original\_data

1. Place files into HDFS directory

$ hdfs dfs -put name\_basics.xlsx original\_data

1. Repeat steps for:
   1. title\_akas.xlsx
   2. title\_basics.xlsx
   3. title\_crew.xlsx
   4. title\_episode.xlsx
   5. title\_principals.xlsx
   6. title\_ratings.xlsx
2. List data in HDFS directory. This is just to check if data has been placed correctly.

$ hdfs dfs -ls original\_data



Step 4: Connecting to Beeline (Skip if Hive installed locally on server)

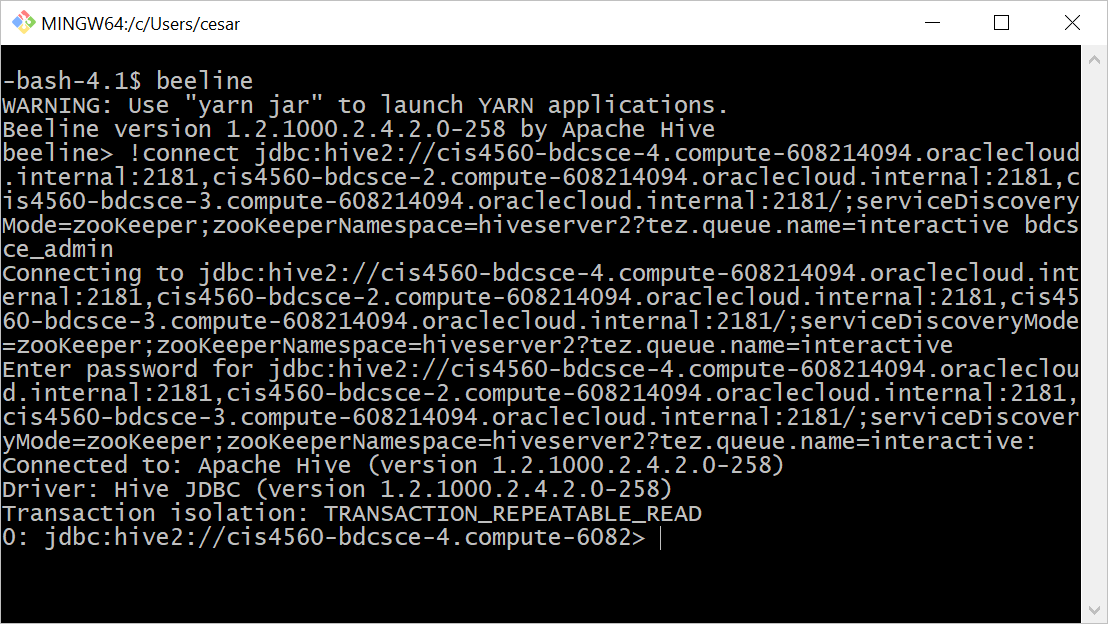
Unlike the hive client, beeline connects to HiveServer2 and does not require the installation of Hive libraries on the same machine as the client.

1. Open beeline client

$ beeline

1. Connect to HiveServer2 with connection string. Password may be required.

beeline> !connect jdbc:hive2://cis4560-bdcsce-4.compute-608214094.oraclecloud.internal:2181,cis4560-bdcsce-2.compute-608214094.oraclecloud.internal:2181,cis4560-bdcsce-3.compute-608214094.oraclecloud.internal:2181/;serviceDiscoveryMode=zooKeeper;zooKeeperNamespace=hiveserver2?tez.queue.name=interactive bdcsce\_admin



Step 5: Creating Tables

**Hive Tables are HDFS directories made up of one or more files. Each individual table will be designed based off the different files we downloaded earlier.**

1. Create a table with CREATE TABLE statement

CREATE EXTERNAL TABLE IF NOT EXISTS data\_name\_basics

(nConst STRING,

primaryName STRING,

birthYear STRING,

deathYear STRING,

primaryProfession STRING,

knownForTitles STRING)

ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t'

STORED AS TEXTFILE

~~LOCATION '/user/clope151/original\_data/name\_basics.xlsx'~~

TBLPROPERTIES ('skip.header.line.count'=’1');

1. Because location is not specified, location will be located at the default path: /user/hive/warehouse/**data\_name\_basics**
2. Create tables for all other files in HDFS directory.

CREATE EXTERNAL TABLE IF NOT EXISTS data\_title\_akas

(titleId STRING, ordering SMALLINT, title STRING,

region STRING, language STRING, types STRING, attributes STRING, isOriginalTitle SMALLINT)

ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t'

STORED AS TEXTFILE

TBLPROPERTIES ('skip.header.line.count'='1');

CREATE EXTERNAL TABLE IF NOT EXISTS data\_title\_basics

(tconst STRING, titleType STRING, primaryTitle STRING, originalTitle STRING, isAdult INT, startYear INT, endYear INT,runtimeMinutes INT, genres STRING)

ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t'

STORED AS TEXTFILE

TBLPROPERTIES ('skip.header.line.count'='1');

CREATE EXTERNAL TABLE IF NOT EXISTS data\_title\_crew

(tconst STRING, directors STRING, writers STRING)

ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t'

STORED AS TEXTFILE

TBLPROPERTIES ('skip.header.line.count'='1');

CREATE EXTERNAL TABLE IF NOT EXISTS data\_title\_episode

(tconst STRING, parentTconst STRING, seasonNumber STRING, episodeNumber SMALLINT)

ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t'

STORED AS TEXTFILE

TBLPROPERTIES ('skip.header.line.count'='1');

CREATE EXTERNAL TABLE IF NOT EXISTS data\_title\_principals

(tconst STRING, parentTconst STRING, seasonNumber STRING,episodeNumber SMALLINT)

ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t'

STORED AS TEXTFILE

TBLPROPERTIES ('skip.header.line.count'='1');

CREATE EXTERNAL TABLE IF NOT EXISTS data\_title\_ratings

(tconst STRING, averageRating DOUBLE, numVotes INT)

ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t'

STORED AS TEXTFILE

TBLPROPERTIES ('skip.header.line.count'='1');

1. Show table data with SELECT statement. Notice how the tables are empty.

SELECT \* FROM data\_name\_basics;

Step 6: Inserting Data Into Tables

**So far we have created empty tables. We will insert data into the tables from the files we moved into the HDFS directory.**

1. Insert data into tables with LOAD DATA INPATH command

LOAD DATA INPATH

'/user/clsadmin/original\_data/name\_basics.xlsx'

INTO TABLE data\_name\_basics;

LOAD DATA INPATH

'/user/clsadmin/original\_data/title\_akas.xlsx'

INTO TABLE data\_title\_akas;

LOAD DATA INPATH

'/user/clsadmin/original\_data/title\_basics.xlsx'

INTO TABLE Data\_title\_basics;

1. Repeat for:
   1. title\_crew.xlsx INTO TABLE data\_title\_crew
   2. title\_episode.xlsx INTO TABLE data\_title\_episode
   3. title\_principals.xlsx INTO TABLE data\_title\_principals;
   4. title\_ratings.xlsx INTO TABLE data\_title\_ratings;

1. Show table data with SELECT statement. Notice how the tables are no longer empty.

SELECT \* FROM data\_name\_basics;

Step 7: Cleaning Up Data

**Downloading big data online is fun, but not all of it is necessary when analyzing specific areas. Cleaning up data by removing any unnecessary columns and rows will allow us to more easily view smaller files. We will be creating tables with less columns from our tables we created earlier.**

1. Create a new table based of data\_title\_akas with rows: titleId, title, and region

CREATE TABLE data\_title\_akas\_limit

AS SELECT titleId, title, region

FROM data\_title\_akas;

1. Create a new table based of data\_title\_basics with rows: tconst, titleType, primaryTitle, startYear, genres

CREATE TABLE data\_title\_basics\_limit

AS SELECT tconst, titleType, primaryTitle,

startYear, genres

FROM data\_title\_basics;

Step 8: Joining Tables

**Now that we have simplified our tables, we can join the columns we find useful from both tables. Remember Hive tables are made up of files. So when we create a new table by joining two tables, we are creating a new file in Hive.**

1. Join tables using INNER JOIN statement. We are joining the limited tables we created in step 7.

CREATE TABLE data\_title\_limit\_join\_innerjoin

AS

SELECT distinct \*

FROM data\_title\_akas\_limit takas

INNER JOIN data\_title\_basics\_limit tbasics

ON (takas.titleId = tbasics.tconst);

Step 9: Preparing Data to be Analyzed

**Earlier we omitted the storage location of our tables. Currently tables are stored in a private warehouse. Lets create a new table in a specified HDFS directory. This new table we be better formatted to be analyzed.**

1. Create a new HDFS directory to store our new data.

hdfs dfs -mkdir new\_data

1. Create table into specified HDFS directory.

CREATE TABLE IF NOT EXISTS data\_title\_analyze

ROW FORMAT DELIMITED

FIELDS TERMINATED BY '\t'

STORED AS TEXTFILE

LOCATION '/user/clsadmin/new\_data'

TBLPROPERTIES ('skip.header.line.count'='1')

AS

SELECT titleId, title, region, titleType, startYear, genres

FROM data\_title\_limit\_join\_innerjoin;

Step 10: Downloading Data

**Now that our new table has been stored into our HDFS directory, we can move our table’s file into our local server. Once in our local directory we can securely copy files into new directories.**

1. List files in HDFS directory we created in step 9. Notice how our file how been broken up to several files.

hdfs dfs -ls new\_data

1. Get files in HDFS directory and place into new directory. In the statement below the asteriks ( \* ) is used as a wildcard to equal any character value. This is used to get multiple files in one statement.

hdfs dfs -get new\_data/00000\*\_0

1. List files in local server. Notice how we moved multiple files.

ls

1. Concatenate (combine) files into one file called: 000000\_0\_comb

cat 000000\_0 000001\_0 000002\_0 000003\_0>000000\_0\_comb

1. Securely copy files from local server (server connected to through SSH) into local machine (computer user is controlling). You can control the location of where you want to store the file.
   1. Note: This command should be done on shell that is connected to your local machine. Unlike other commands that were done in shell connected to local server.

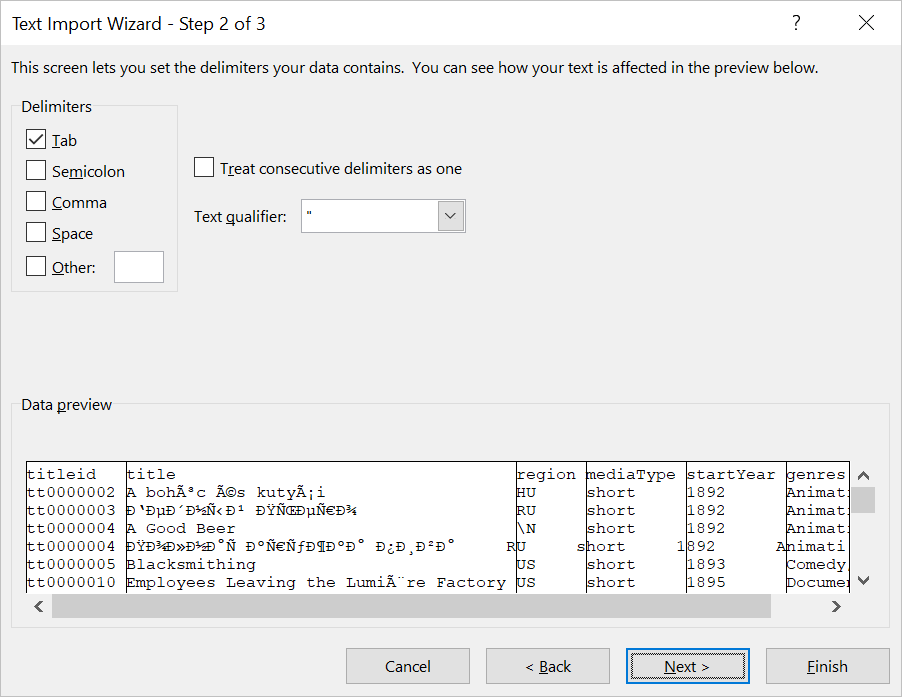
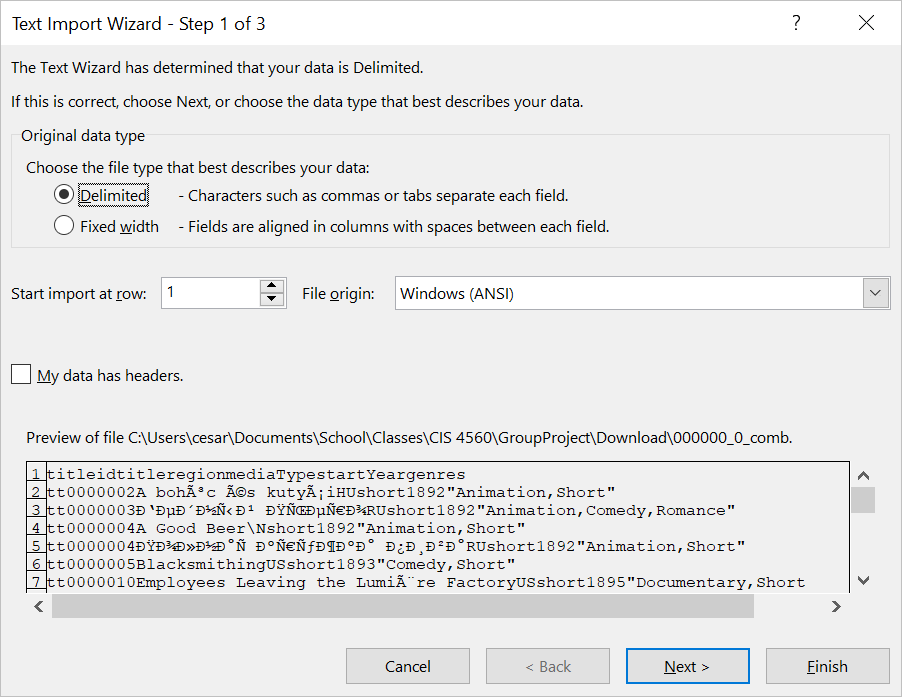
scp clsadmin@169.63.19.198:/home/clsadmin/000000\_0\_comb

/c/Users/cesar/Documents/School/Classes/

Step 11: Preparing File for PowerBi

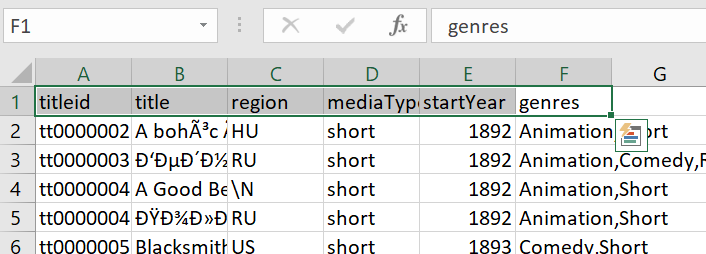
**In order to be able to load our excel data into PowerBi, we must format our data into a flat table. Each column should follow the same data type and there should be a header for each column. All data should also be formatted as a table in excel.**

1. Find file that was copied into local machine. Save file location.
2. Open Excel. In File tab, click open and find file that was downloaded
3. Make sure file is delimited by tabs. If we were to open up our file we would see the file organized by tabs.

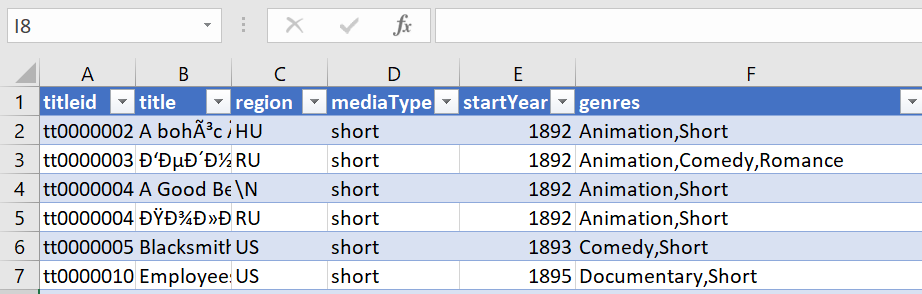




1. Insert New Row in Cell Row 1. Note: The very last row may have to be deleted to fit the header



1. Format all cells containing data into a table. Save file as csv



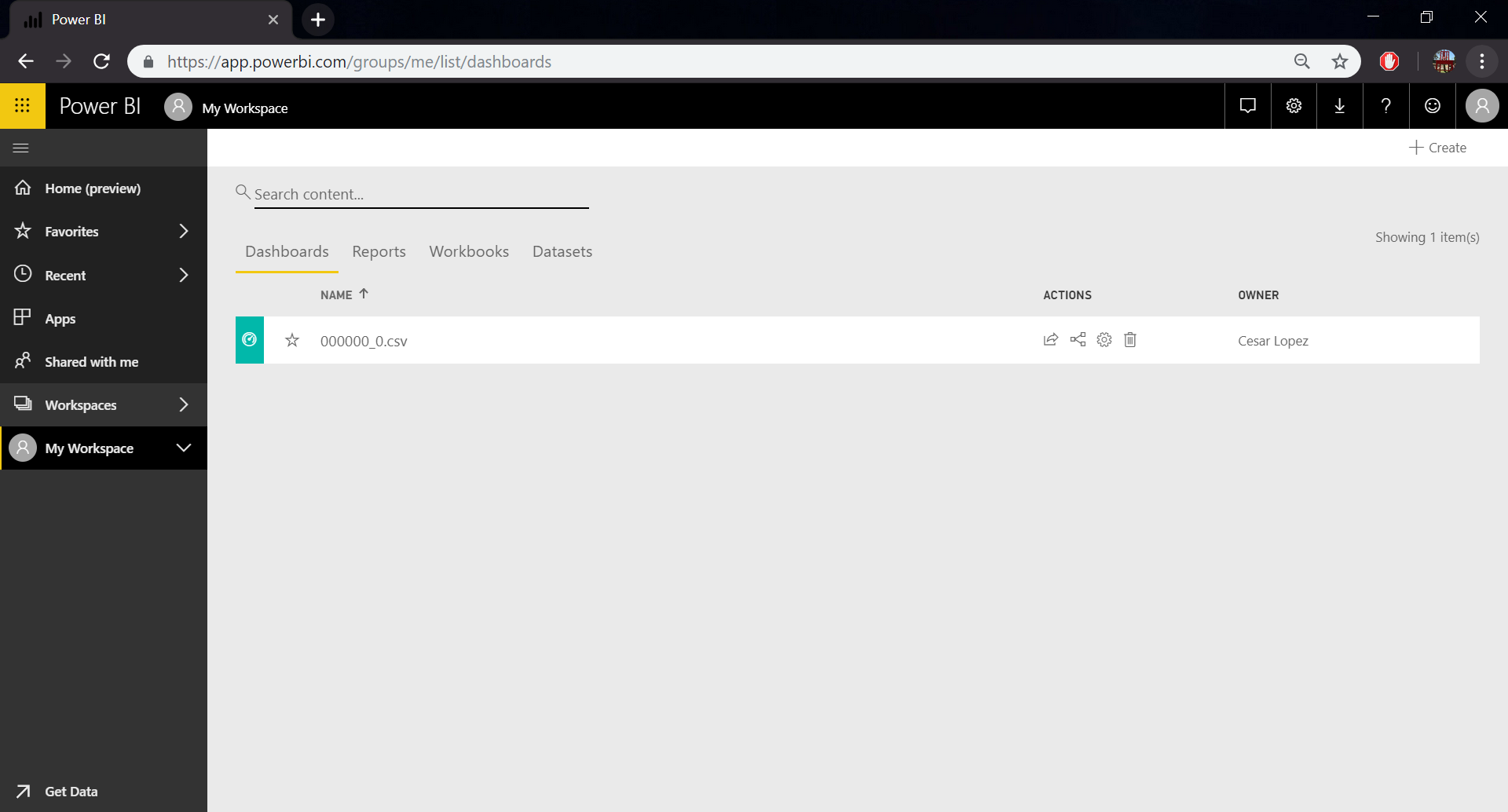
Step 12: PowerBi Visualization

**PowerBi is a business analytics service that provides us with interactive visualizations based off our data. First we need to upload our formated file into PowerBi. Then we can create multiple visual reports based off our uploaded data.**

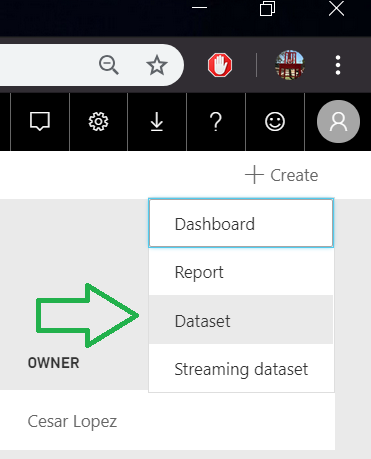
1. Login Into PowerBi. Image below is an example of a home page.



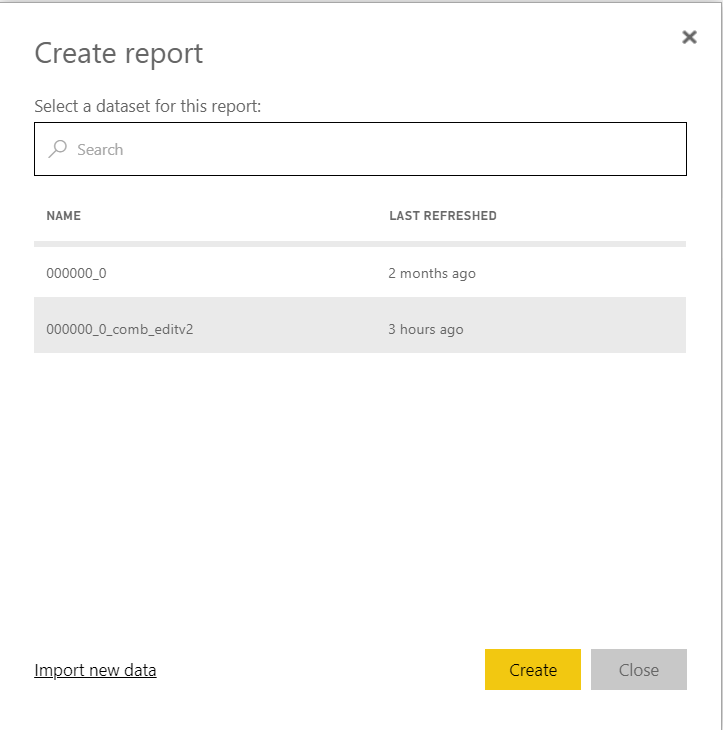
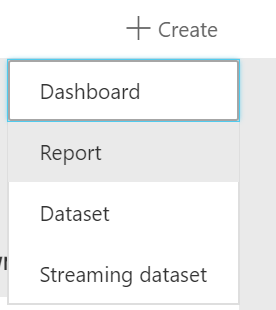
1. Go to My Workspace



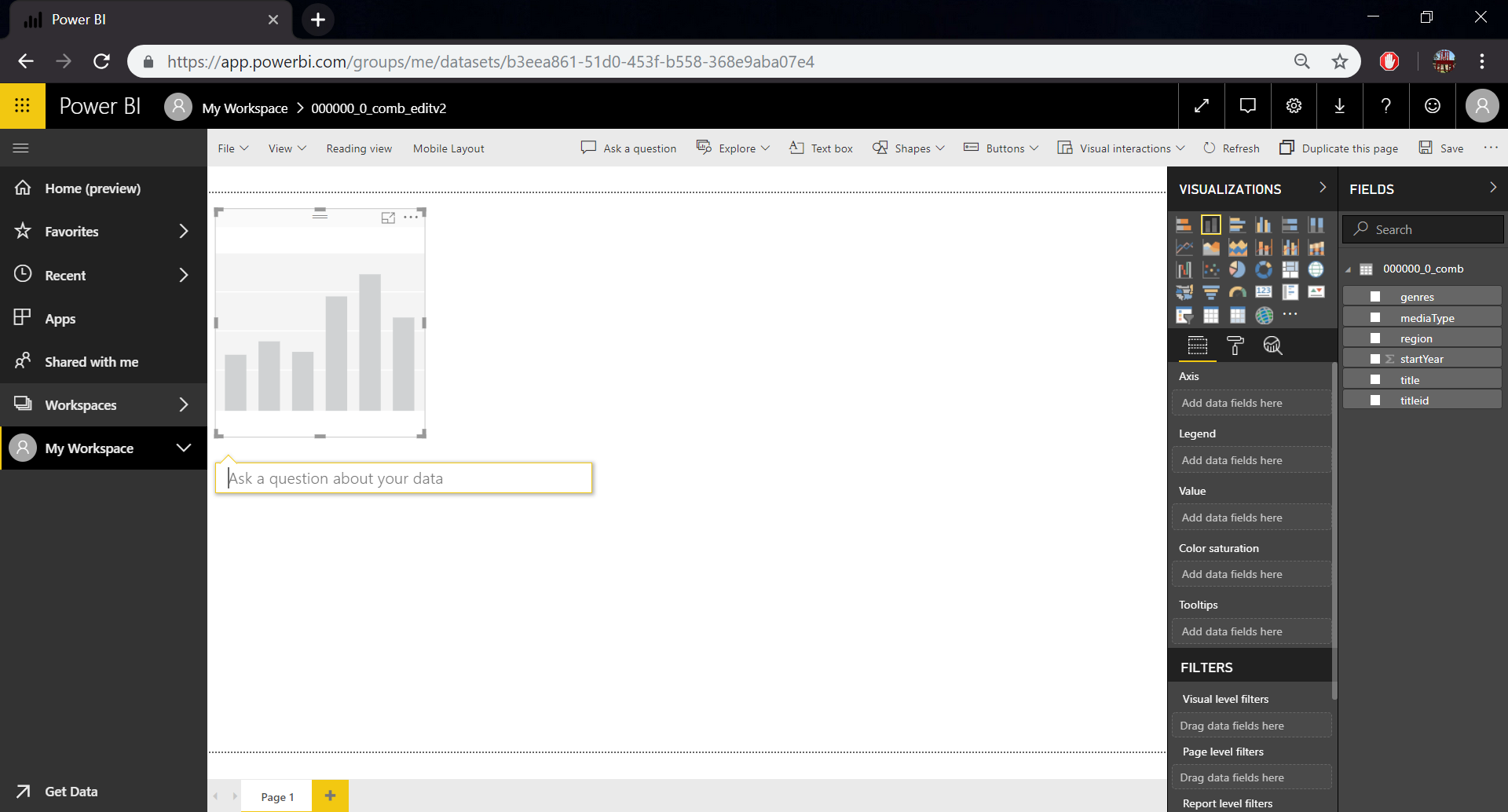
1. Create New Dataset. Click the +Create button to see a drop menu of creatable options. Click Dataset.

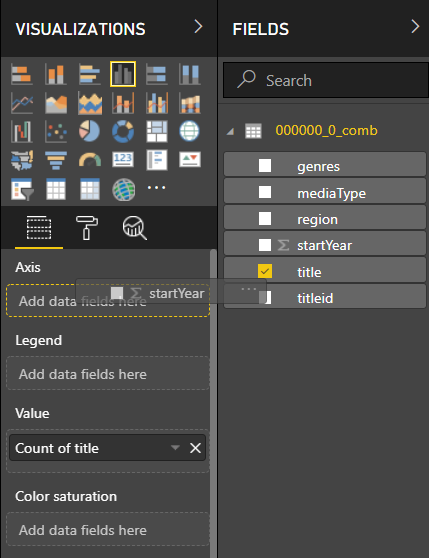


1. Create New Report based off data set we just uploaded. Click +Create button again. When creating report make sure to select the dataset we just uploaded.

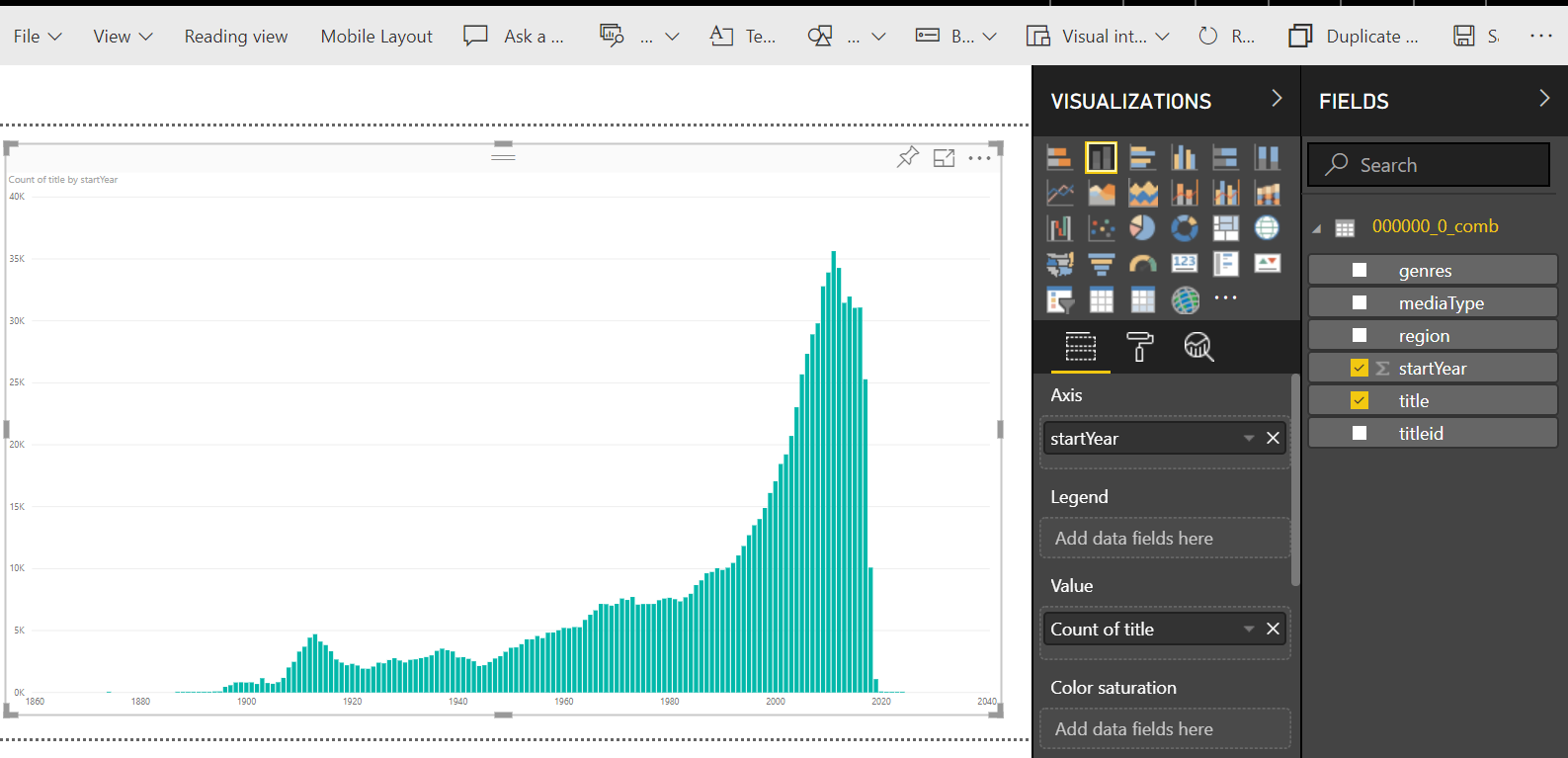


1. Familiarize yourself with the report Editor. The visualizations tab has different tables and charts we can use based off our data. Selecting one of the icons under the visualization tab will create new visual objects. We then can use the fields tab to select field check boxes and drag fields into forms appropriate to the visual object we just created.

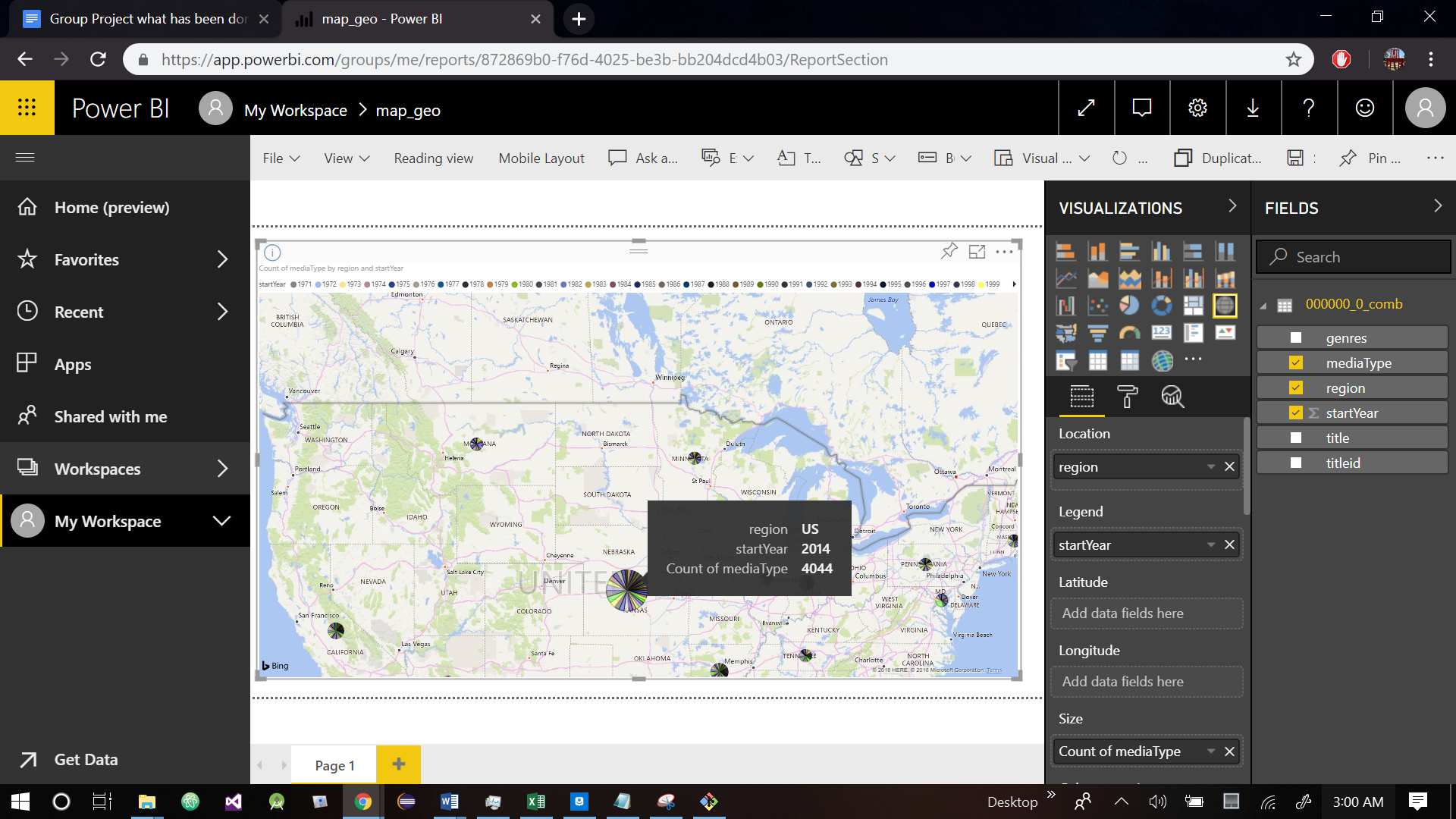


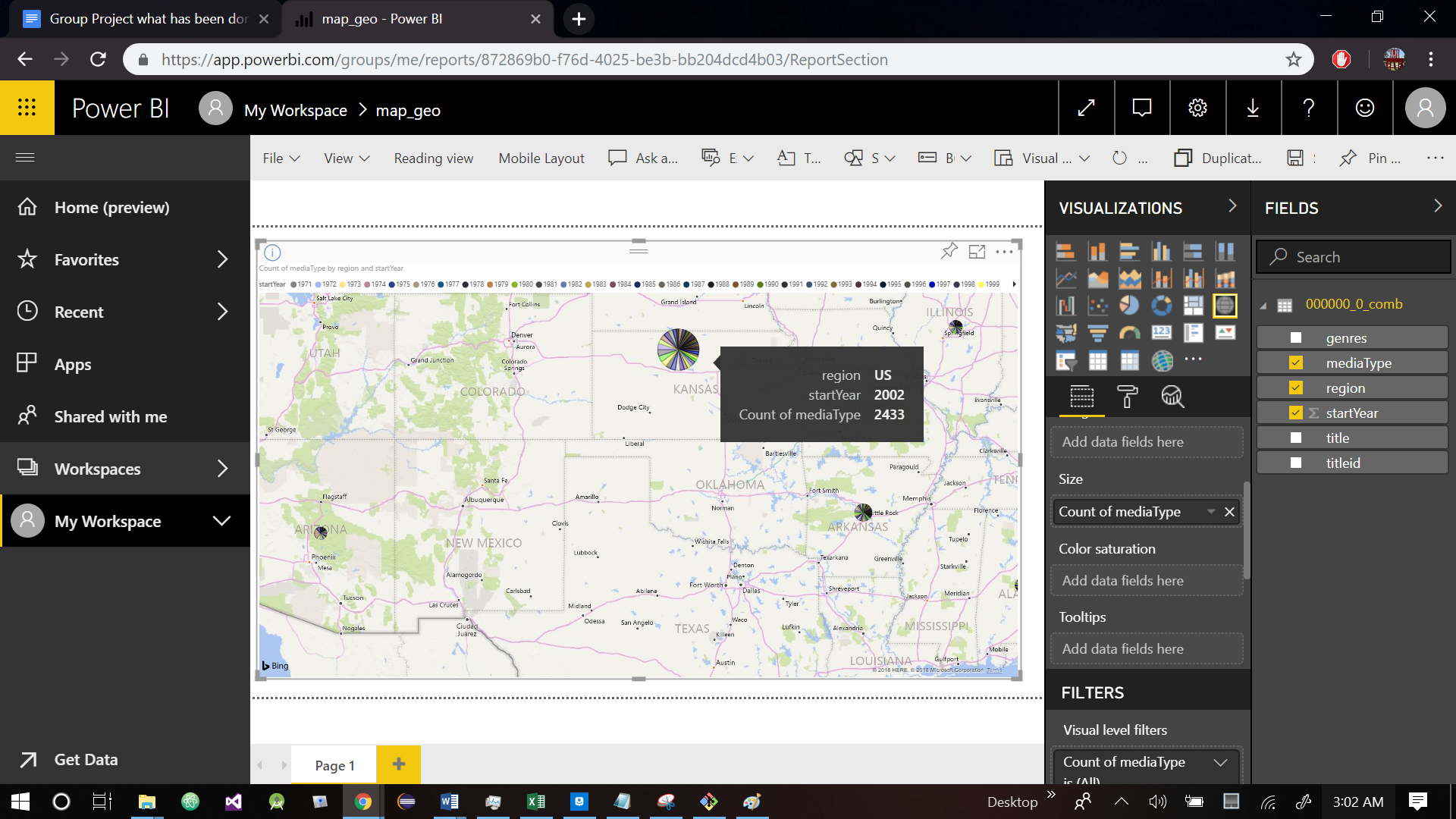


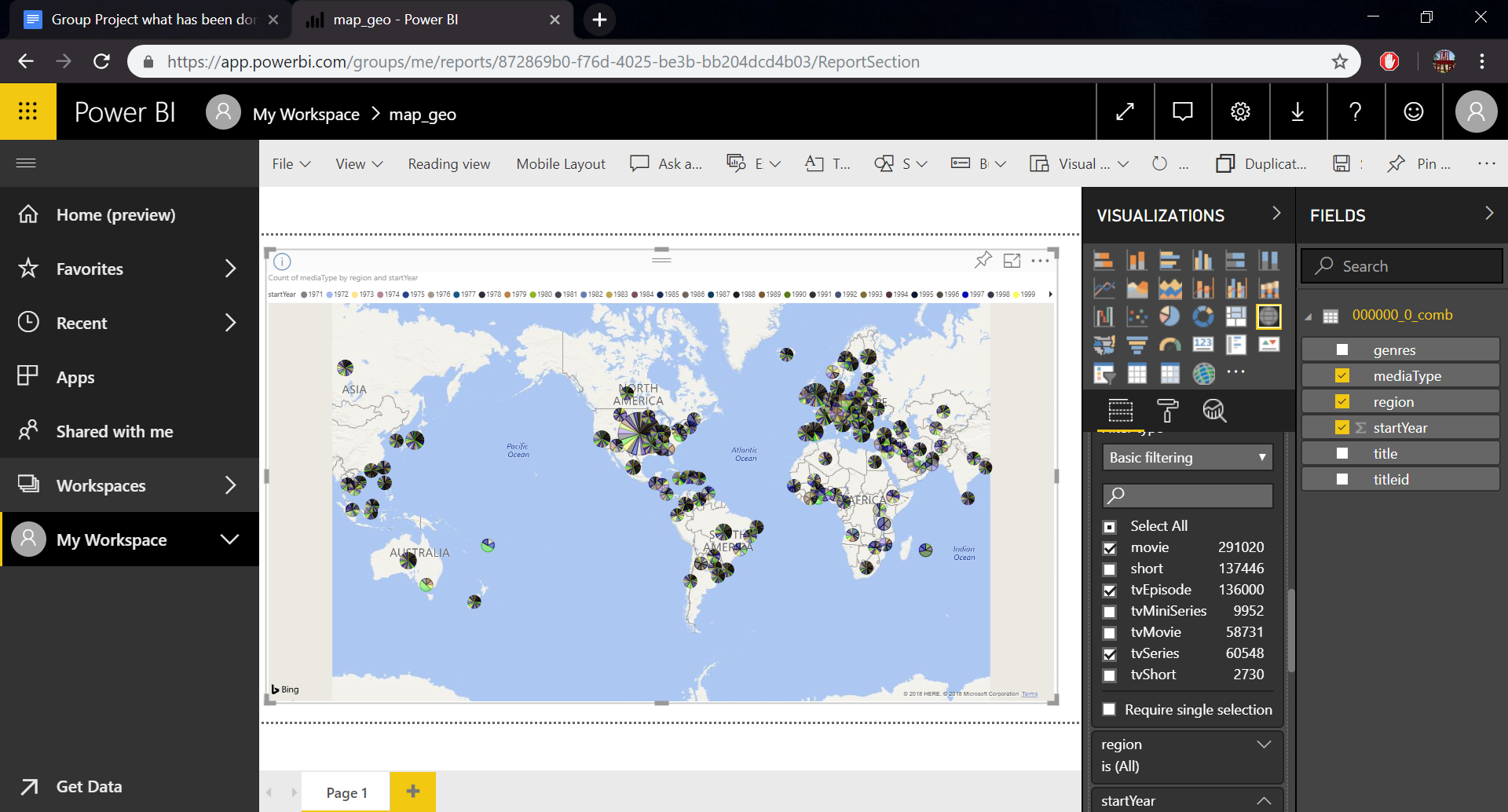
1. Create a simple Stacked Column Chart by selecting the second icon. Under Axis drag startYear. Under Value drag title. Right away we get a chart that visualizes the total count of all the media released in different regions based off year.

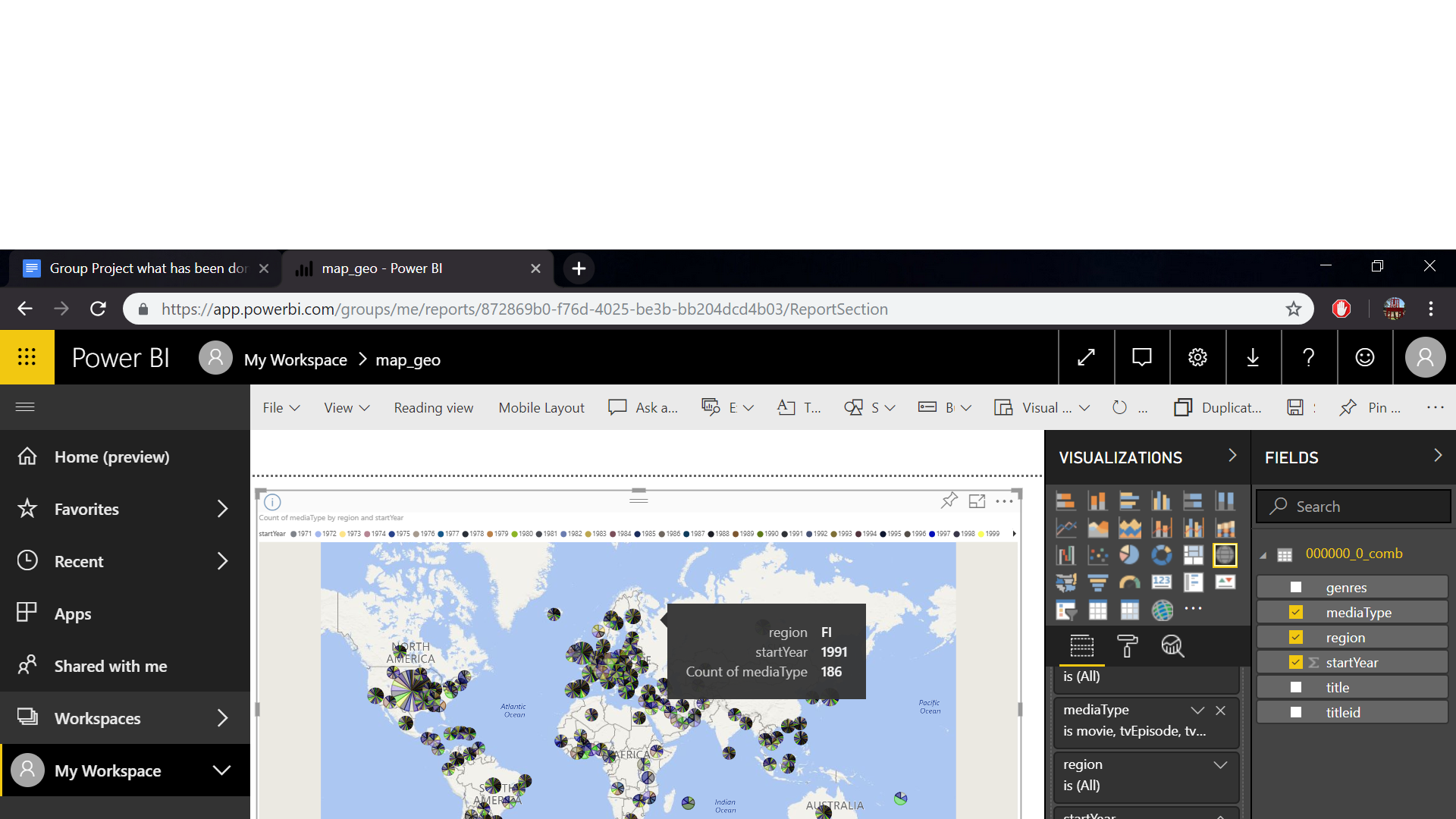


1. Creating a Map for Geo-Spatial Visualization. Select the Map Icon. Under location drag the region field. Then under Legend add startYear. Also add mediaType under size. We then can filter results based off the different media types. Select Movies, tvEpidsode, tvSeries, and videoGames and of media released after 1970









Step 13: Continued Analysis

**Now that we know how to visualize our data, we can continue to create different queries that will help us analyze our data. By altering titleId, title, region, startYear, genres, titleType in our query we can obtain different data. We can search for movies that came out in a specific region and that came out after a specific year. We can count the number of movies that were released a specific period. Count the different total titleTypes released for each title. This can also be counted by year and have genres limited to specific topic.**

1. Basic query we can use to analyze data. By changing region, startyear, genres, and titleType in the WHERE clause, we can drastically change the data that is outputted.

SELECT titleId, title, region, startYear, genres, titleType

FROM data\_title\_limit\_join\_innerjoin

WHERE region = 'US'

AND startYear > 1970

LIKE concat('%', 'Comedy', '%')

AND titleType = 'movie';

1. Count movie titles grouped by their genres and released after 1990

SELECT COUNT(titleId), genres

FROM data\_title\_limit\_join\_innerjoin

WHERE region = 'US'

AND startYear > 1990

AND titleType = 'movie'

GROUP BY genres

LIMIT 1000;

1. Count US movies that were released each year

SELECT startYear, COUNT(titleId)  
 FROM data\_title\_limit\_join\_innerjoin   
 WHERE region = 'US'   
 AND titleType = 'movie'  
 GROUP BY startYear;

1. Count US movies by titletype

SELECT titleType, COUNT(titleId)  
FROM data\_title\_limit\_join\_innerjoin  
WHERE region = 'US'   
GROUP BY titleType;

1. Count US movies by titletype and year

SELECT titleType, startYear, COUNT(titleId)  
FROM data\_title\_limit\_join\_innerjoin   
WHERE region = 'US'  
GROUP BY titleType, startYear;

1. Count US movies by titletype and year where genres has horror. This table is organized by the startYear. The like statement only looks for genres that have the word horror included in their record.

SELECT titleType, startYear, COUNT(titleId)

FROM data\_title\_limit\_join\_innerjoin

WHERE region = 'US' and genres

LIKE concat('%', 'Horror', '%')

GROUP BY titleType, startYear

ORDER BY startYear;

Step 14: Preparing Analysis with Views

**Now that we have a better idea in how to analyze our data we can continue to use Hive features to prepare data. Once data has been manipulated we can gain different insights with more analysis. Views help simplify query results. We will use views to store useful tables created from queries.**

1. We joined two tables but some columns from the two tables have repeatable and unnecessary information. Create a VIEW to save a simplistic version of the table.

CREATE VIEW data\_title\_imp\_limit

AS

SELECT titleId, title, region, startYear, genres, titleType

FROM data\_title\_limit\_join\_innerjoin

WHERE region = 'US';

1. Hive has support for arrays as a data type. We can convert a string delimited by commas ( , ) into an array with the split function. Separating each genre will help us ensure every genre can be accessed for each record. We will store this result in another view to be able to access these useful results again

CREATE VIEW data\_title\_split

AS

SELECT titleId, title, region, startYear, split(genres, ',') as genres, titleType

FROM data\_title\_limit\_join\_innerjoin

WHERE region = 'US';

1. Looking at the different combinations of genres for each movie can look confusing. To ensure we get every movie genre we will expand each row. This view will have a lot of repeated values. We can then select all the distinct values from this view to get every genre in the table.

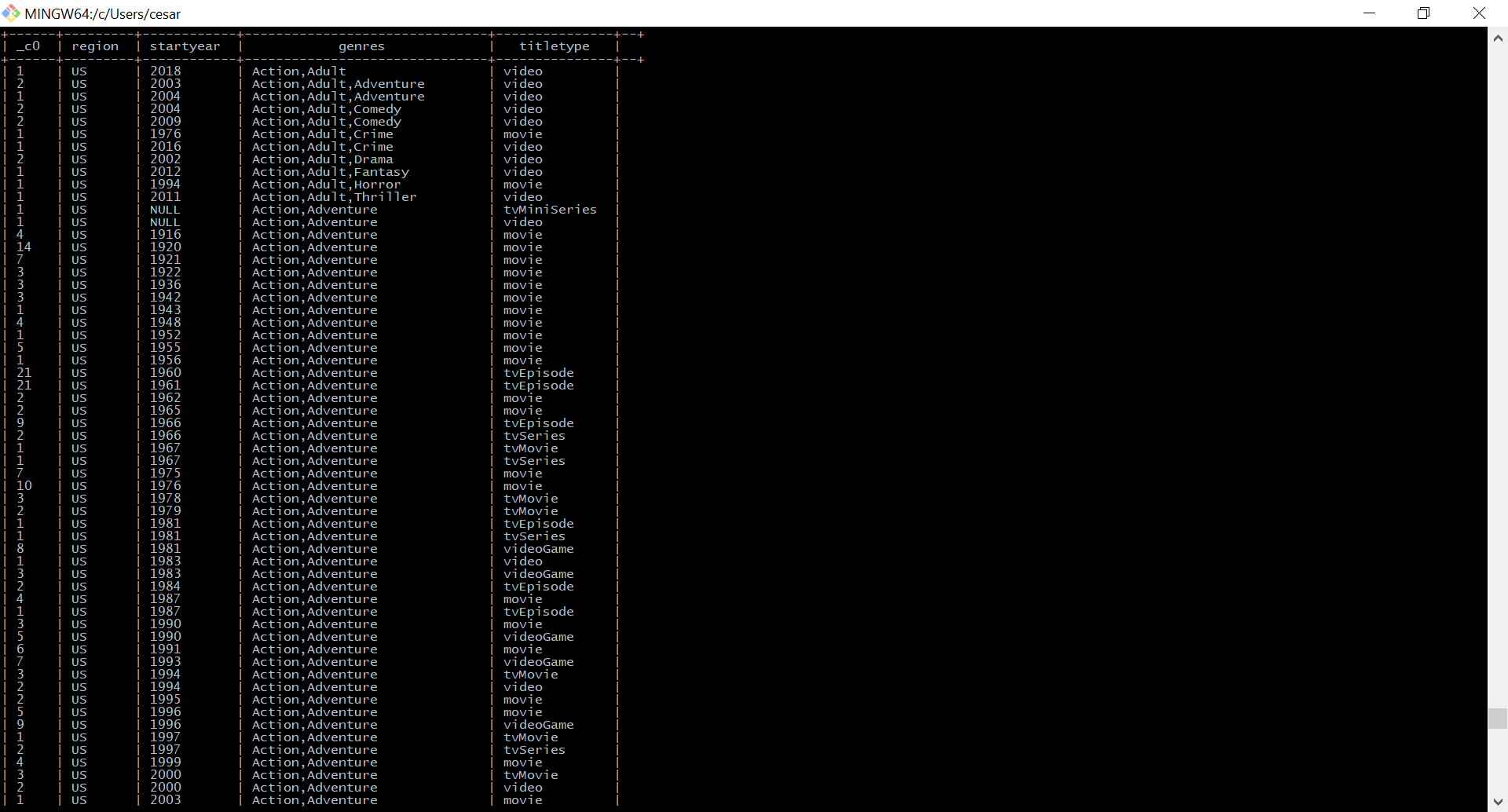
CREATE VIEW data\_title\_split\_genres\_ungrouped  
 AS  
 Select explode(genres)  
 FROM data\_title\_split;

Select distinct \*   
 FROM data\_title\_split\_genres\_ungrouped;

+--------------------------+ | Romance |  
| Genres | | Adult |  
+--------------------------+ | Animation |  
| Action | | Documentary |  
| Biography | | Drama |  
| Crime | | Fantasy |  
| History | | Film-Noir |  
| Horror | | Game-Show |  
| Music | | News |  
| Musical | | Reality-TV |  
| Mystery | | Short |  
| Sci-Fi | | Sport |  
| Western | | Talk-Show |  
| Adventure | | Thriller |  
| Comedy | | War |  
| Family | +--------------------------+

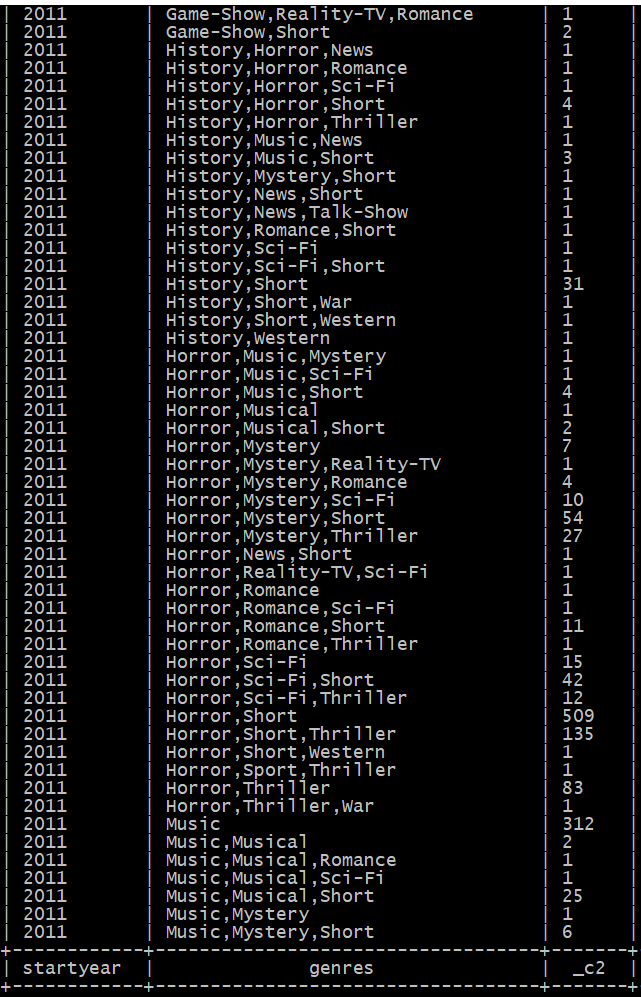
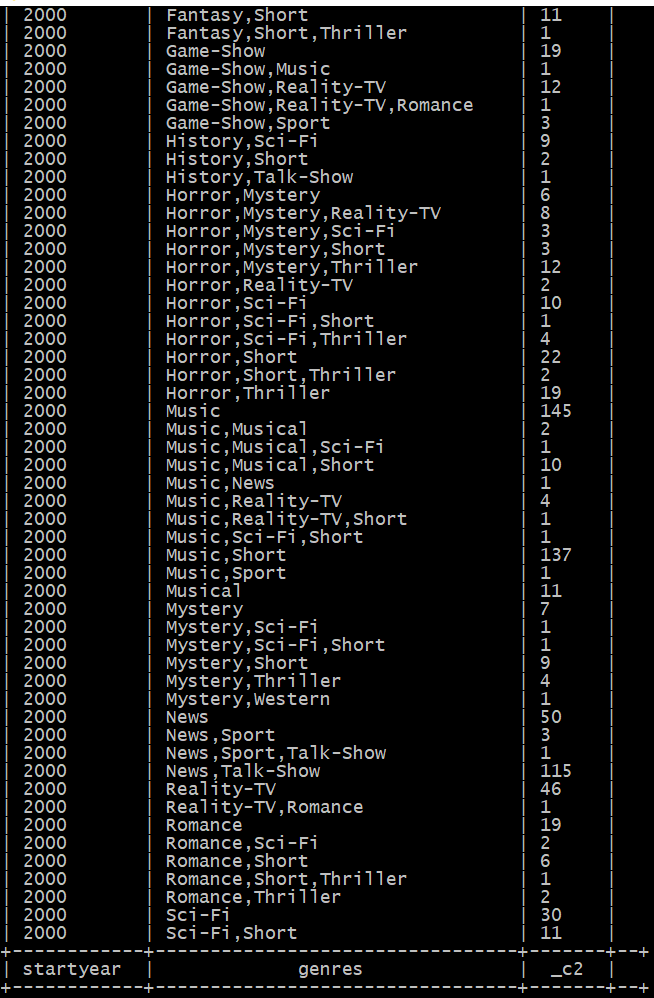
1. We can count the number of specific media type by specific detailed genres (multiple genres). This allows us to see the number of media with specific genres that came out within the same year. In this query we organize our results by genre names then startYear. This helps us see all the same genres with their concurrent year.

SELECT count(titleId), region, startYear, genres, titleType  
 FROM data\_title\_imp\_limit  
 GROUP BY genres, startYear, titleType, region  
 HAVING genres LIKE concat('%', 'Action', '%') OR  
 genres LIKE concat('%', 'Biography ', '%') OR  
 genres LIKE concat('%', 'Crime ', '%') OR  
 genres LIKE concat('%', 'History ', '%') OR  
 genres LIKE concat('%', 'Horror ', '%') OR  
 genres LIKE concat('%', 'Music', '%') OR  
 genres LIKE concat('%', 'Musical', '%') OR  
 genres LIKE concat('%', 'Mystery', '%') OR  
 genres LIKE concat('%', 'Sci-Fi', '%') OR  
 genres LIKE concat('%', 'Western', '%') OR  
 genres LIKE concat('%', 'Adventure', '%') OR  
 genres LIKE concat('%', 'Comedy', '%') OR  
 genres LIKE concat('%', 'Family', '%') OR  
 genres LIKE concat('%', 'Romance', '%') OR  
 genres LIKE concat('%', 'Adult', '%') OR  
 genres LIKE concat('%', 'Animation', '%') OR  
 genres LIKE concat('%', 'Documentary', '%') OR  
 genres LIKE concat('%', 'Drama', '%') OR  
 genres LIKE concat('%', 'Fantasy', '%') OR  
 genres LIKE concat('%', 'Film-Noir', '%') OR  
 genres LIKE concat('%', 'Game-Show', '%') OR  
 genres LIKE concat('%', 'News', '%') OR  
 genres LIKE concat('%', 'Reality-TV', '%') OR  
 genres LIKE concat('%', 'Short', '%') OR  
 genres LIKE concat('%', 'Sport', '%') OR  
 genres LIKE concat('%', 'Talk-Show', '%') OR  
 genres LIKE concat('%', 'Thriller', '%') OR  
 genres LIKE concat('%', 'War', '%');



1. This query groups all media types together and counts them. I have limited the query to the year 2000 but this value can be changed (e.g. 2011) to get a different table.

SELECT startYear, genres, count(titleId)  
 FROM data\_title\_imp\_limit  
 WHERE startyear = 2000  
 GROUP BY genres, startYear  
 HAVING genres LIKE concat('%', 'Action', '%') OR  
 genres LIKE concat('%', 'Biography ', '%') OR  
 genres LIKE concat('%', 'Crime ', '%') OR  
 genres LIKE concat('%', 'History ', '%') OR  
 genres LIKE concat('%', 'Horror ', '%') OR  
 genres LIKE concat('%', 'Music', '%') OR  
 genres LIKE concat('%', 'Musical', '%') OR  
 genres LIKE concat('%', 'Mystery', '%') OR  
 genres LIKE concat('%', 'Sci-Fi', '%') OR  
 genres LIKE concat('%', 'Western', '%') OR  
 genres LIKE concat('%', 'Adventure', '%') OR  
 genres LIKE concat('%', 'Comedy', '%') OR  
 genres LIKE concat('%', 'Family', '%') OR  
 genres LIKE concat('%', 'Romance', '%') OR  
 genres LIKE concat('%', 'Adult', '%') OR  
 genres LIKE concat('%', 'Animation', '%') OR  
 genres LIKE concat('%', 'Documentary', '%') OR  
 genres LIKE concat('%', 'Drama', '%') OR  
 genres LIKE concat('%', 'Fantasy', '%') OR  
 genres LIKE concat('%', 'Film-Noir', '%') OR  
 genres LIKE concat('%', 'Game-Show', '%') OR  
 genres LIKE concat('%', 'News', '%') OR  
 genres LIKE concat('%', 'Reality-TV', '%') OR  
 genres LIKE concat('%', 'Short', '%') OR  
 genres LIKE concat('%', 'Sport', '%') OR  
 genres LIKE concat('%', 'Talk-Show', '%') OR  
 genres LIKE concat('%', 'Thriller', '%') OR  
 genres LIKE concat('%', 'War', '%')  
 ORDER BY startyear, genres;



Step 15: More Analysis

**We can always alter queries to display useful information to us.**

1. Count the total media that has came out each year. This query does not separate results from there genre or media type.

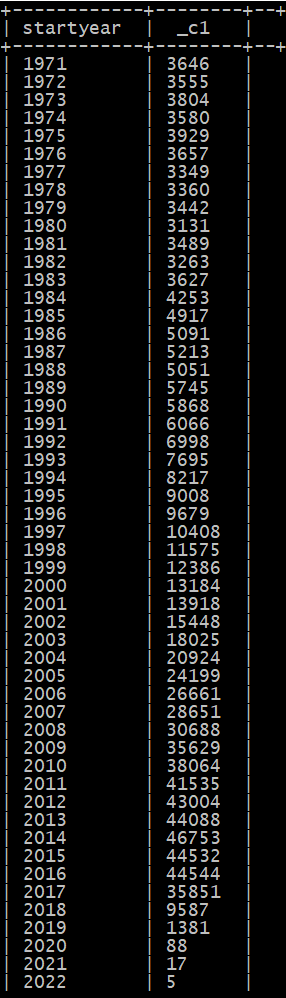
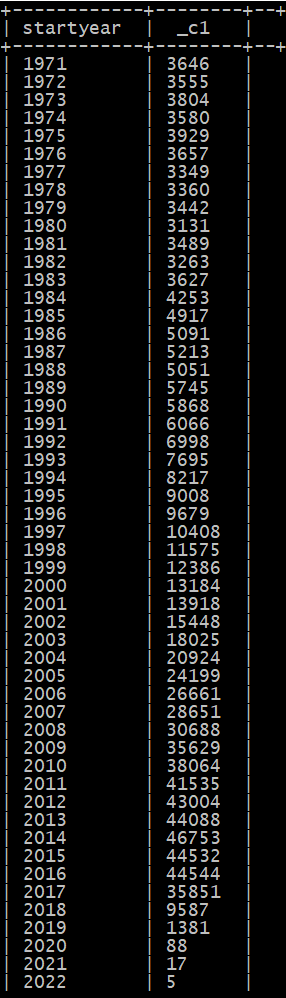
SELECT startYear, count(titleId)

FROM data\_title\_imp\_limit

WHERE startyear> 1970

GROUP BY startYear

ORDER BY startyear;



1. This query only counts media with the titletype movie that came out after 1990

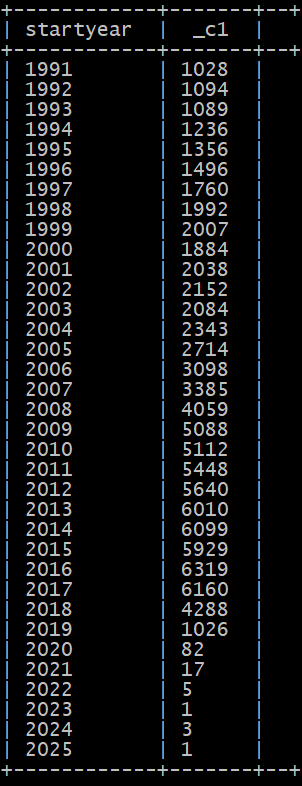
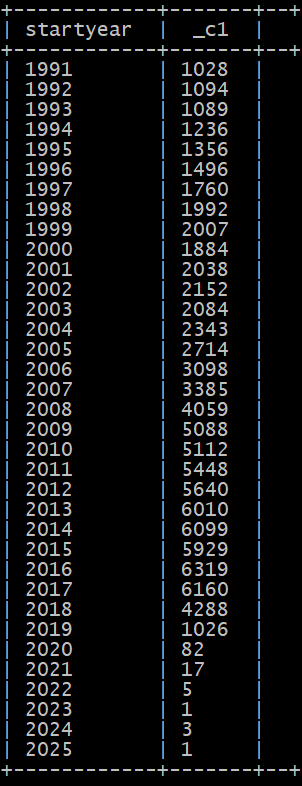
SELECT startYear, count(titleId)

FROM data\_title\_imp\_limit

WHERE startyear> 1990 and titletype = 'movie'

GROUP BY startYear

ORDER BY startyear;



1. We can specify our query again by limiting results by genres

SELECT startYear, count(titleId)

FROM data\_title\_imp\_limit

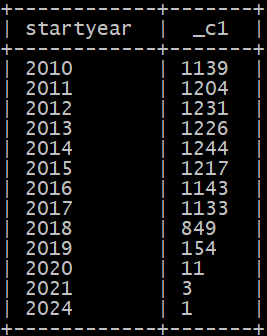
WHERE startyear > 2009

AND titletype = 'movie'

AND genres LIKE concat('%', 'Comedy', '%')

GROUP BY startYear

ORDER BY startyear;



References

* URL of Data Source,<https://datasets.imdbws.com/>
* URL of our Github: <https://github.com/SelipeProducts/IMDB-Intro-to-Data-Analysis-using-Hive>
* URL of References
  + <https://www.imdb.com/interfaces/>
  + <https://console.bluemix.net/catalog/services/analytics-engine>
  + <https://www.ibm.com/cloud/analytics-engine>
  + <https://docs.microsoft.com/en-us/power-bi/service-from-excel-to-stunning-report>
  + <https://www.youtube.com/watch?v=Qgam9M8I0xA>