Project-2 Ishan Patwa(65461485) , Bharath Yarlagadda(61319913)

Enron Data Set Analysis:

Aim : With the analysis of enron email data set we have tried to identify "positive reinforcement" behavior amongst the top level employee of enron. We have done this by doing a sentiment analysis of the email context for a particular subset of emails in the enronset. Details of how the subsets were chosen are describe later in the report.

Motivation : In large industries such as enron, a malpractice has many adverse consequences both to the individual and the company. In such situations as was that of enron the top down communication plays an very important role in boosting companies morale. Such "positive reinforcements" however are used not only to add strength to the morale of the company , but sometimes also to curtain discrepancies of harmful nature. We believe doing analysis of such nature will provide us with two very important insights and learning : 1) Was positive reinforcement used to avoid any suspicions of fraud? 2) were the top down communications low on sentiment ? indicating a bankruptcy ? if yes , then was this a good strategy.

Analysis: Lets us consider a minimum prior knowledge of the enron timeline to test our hypothesis. Now, to determine the subset of emails which will play an important role in our analysis we start of by doing a frequency analysis on the data set to identify the deviants. We also perform the frequency analysis on the senders/receivers to identify the most active members of the company.

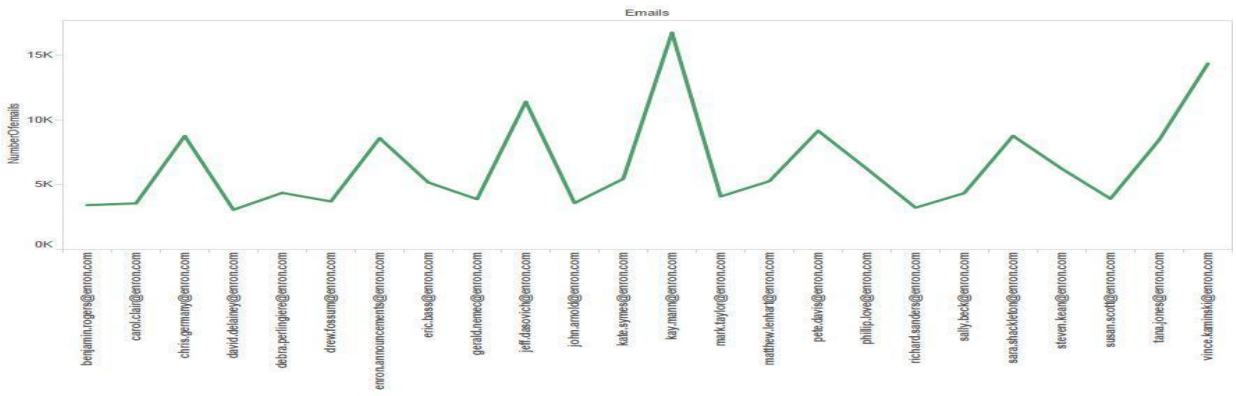
We define our data set in the following way , with the table "enronset" holding our data.

create external table enronset (eid STRING,timestamp STRING,from_id STRING,to_id STRING,cc STRING,subject STRING,context STRING) COMMENT 'enron data_set' ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t'STORED AS TEXTFILE LOCATION 's3n://spring-2014-ds/enron_dataset/';

To begin with we identify the people sending the maximum number of emails by the following query :

select from_id , count(from_id) as counts from enronset group by from_id order by counts desc limit 10;

This gives us the frequency of emails sent by the employees, also information like the person sending the maximum number of emails. Some of our results show the top 25 person sending the emails:

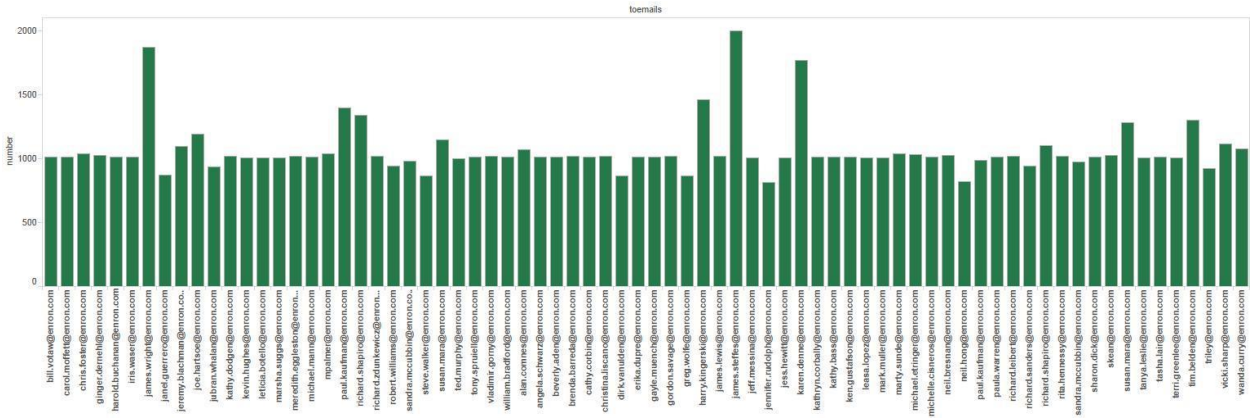


From the results above we found out that following people sending the maximum number of emails:
kay.mann@enron.com=16721 vince.kaminski@enron.com=14329 jeff.dasovich@enron.com=11387
pete.davis@enron.com=9149 sara.shackleton@enron.com=8757

After careful analysis, we see a couple of very interesting observations :

1. **Andrew Fastov one of the main accused in the fraud has a very small communication within or outside the company infact there are only 9 emails that he sent that are present in the data set;**
2. **Although Kay McCall Mann and Vince Kaminski were the lead senders, Jeff Dasovich exchanged emails with the maximum number of people suggesting he was in apublic relations officer or a spokesperson in the company.**

A closer look at the email distribution of Jeff Dasovich gives the following distribution.

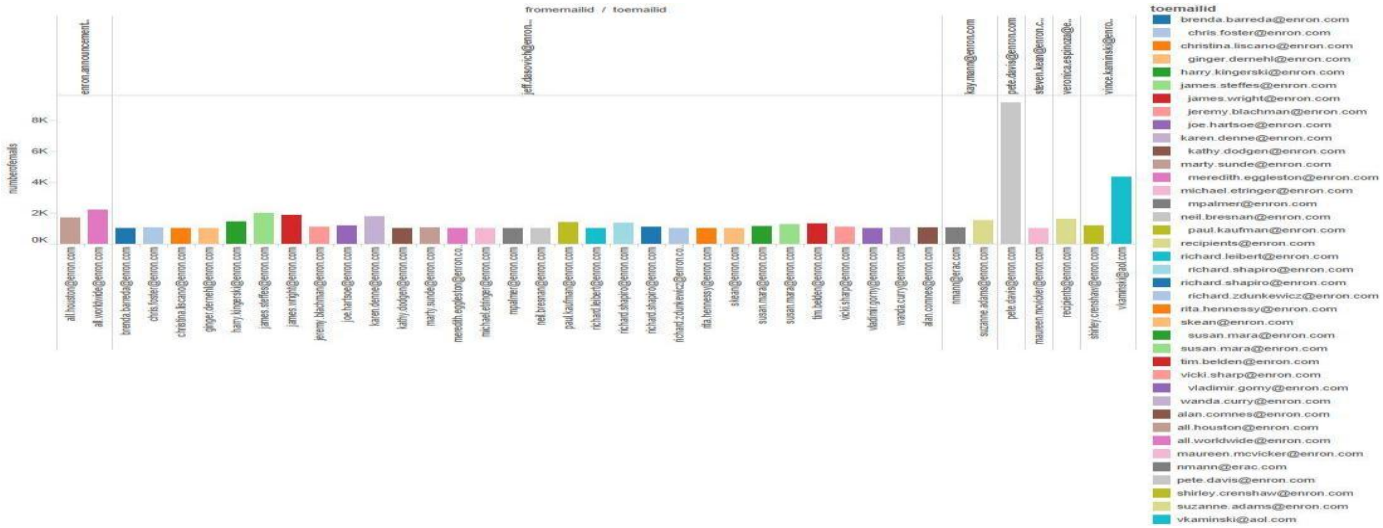


We used the following query to extract recipients of Jeff's emails and the number of emails sent:

select from_id ,splitstring, count(1) as counts
from enronset
Lateral view explode(split(to_id,',')) mytable as splitstring
where from_id="jeff.dasovich@enron.com"
group by from_id , splitstring
order by counts desc;
This query essentially splits the to_id of the enronset and then laterally explodes the to_id column to get pairs of from_id and to_id , finally they are grouped to get the total count.

We also performed an analysis on pairs of senders/receivers frequency as well, giving us the information on which two email id's saw the most number of email exchanges between them.
insert overwrite directory 's3n://ishan-bucket/results/'
select from_id ,splitstring, count(1) as counts
from enronset
Lateral view explode(split(to_id,',')) mytable as splitstring
group by from_id , splitstring
order by counts desc;

To achieve this we first make the to_id column array of strings , then explode those to create from_id to_id count(1) pair for all the possible combinations then we just group them by from_id , to_id first to aggregate unique sending entries second grouping to aggregate duplicates.
A visualization of pair of sender/receiver frequency is shown below:

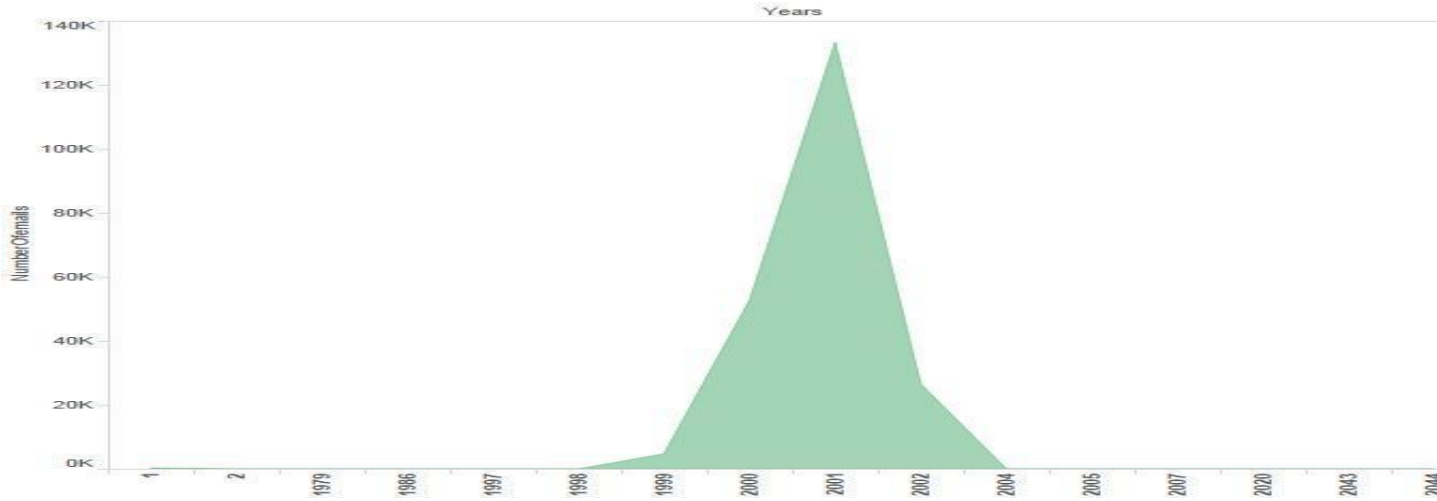


After the analysis of the enron email set by "from_id" i.e the sender's perspective , we narrowed down the span of sentiment analysis to people having the most communication in the company.
kay.mann@enron.com=16721 vince.kaminski@enron.com=14329 jeff.dasovich@enron.com=11387 pete.davis@enron.com 9149 sara.shackleton@enron.com=8757

As you can see from the data retrieved above the size of the corpus was still too huge and would contain irrelevant and probably misleading information. To narrow the scope of our data set we did a thorough time-frequency analysis of the email set.
To begin we plotted the year and month/year distribution of the email sets using the following queries:
/** getting the frequency distribution on the number of emails sent per year **/

```
insert overwrite directory 's3n://ishan-bucket/results/'
select  dates , count(dates) as numEmails
from (
select split(timestamp," ")[3] as dates , count(timestamp)
from enronset
group by timestamp) mailfreq
group by dates
order by numEmails desc;
```

In the above query we use "split" inbuilt function of hive to select the to_id coumn into array<string> and retrieve the 4th element of the array i.e the year. And group them to get the count followed the sorting in descending order using the inbuilt function "order by".
.



We could clearly see a major deviation from trend in the year **2000-2002** with the number of emails exchanged peaking at **2001**.

We also made couple of very interesting observations by this analysis:

1) There were many emails with a invalid year for example year "1" and "2".

2) We also found some outliers with the email timestamp referencing emails of future for example "2044"

3) The data between the years "1987-1996" and "1980-1986" we missing.

We were also able to extract the email frequency based on year and date using the following query.

```
select dates , count(dates) from
(select concat_ws(" " ,split(timestamp," ")[0],split(timestamp," ")[1],split(timestamp," ")[2],split(timestamp," ")[3]) as dates , count(timestamp)
from enronset
group by timestamp) mailfreq
group by dates;
```

Having the year of deviation we decided to analyse the sentiments of emails exchanged by the top 5 senders in the year 2001, In this report we have shown all the queries for vince kaminski. First we created a new table to store emails exchanged by vince in year 2001

```
create table if not exists vinceemails ( from_id string , to_id string , timestamp string ,context string , hashvalue int ) row format delimited fields terminated by "\t" stored as textfile;
```

Please note a new column hashvalue was added to this table to tag each email with a unique , the purpose of which is explained ahead. The table was populated with the following query:

```
insert overwrite table vinceemails
select from_id,to_id, concat_ws(" " ,split(timestamp," ")[0],split(timestamp," ")[1],split(timestamp," ")[2],split(timestamp," ")[3]), regexp_replace(context,"\\s+",","),
hash(from_id,to_id,length(context),split(timestamp," ")[4]) as identifier
from enronset
where split(timestamp," ")[3] = 2001 and
from_id="vince.kaminski@enron.com";
```

Here a new inbuilt function “hash” was used to create a unique key from the from_id,to_id ,length of context and timestamp. regexp_replace was used to clean the context column. To perform the sentiment analysis of the words AFINN dataset was used to assign a sentiment to word raning from -5 to 5 , -5 being the lowest and negative and 5 being the highest and positive/good sentiment. Another table vinceemailswordssplits was created to store context as array<string> using the following query:

create table vinceemailswordssplits if not exists (from_id string , to_id string , timestamp string , context array<string> , hashvalue int) row format delimited fields terminate by “\t”;

It was populated by the following query:
insert overwrite table vinceemailswordssplits select from_id, to_id,timestamp, split(context,",") , hashvalue from vinceemails;

We created a AFINN table using the following query , which was populated from the afinn wordlist.
create table afinn (words string , sentiment int) row format delimited fields terminated by "\t" stored as textfile;

Another table was created to get each word sent by vince in the email context:
create table if not exists vincewordsexploded (from_id string , to_id string , timestamp string, word string , hashvalue int , sentiment int) row format delimited fields terminated by "\t";

The table was popluated with the sentiment's initial value of 0 and exploded words:
insert overwrite table vincewordsexploded
select from_id ,to_id,timestamp,viral ,hashvalue, 0
from vinceemailswordssplits
lateral view explode(context) mytable as viral ;

Another table called sentiment was created which stored a word, tag(to identify in which email it appeared) and its sentiment.
create table if not exists sentiment (word string, timestamp string, sentiment float , hashvalue int)
row format delimited
fields terminated by “\t” ;

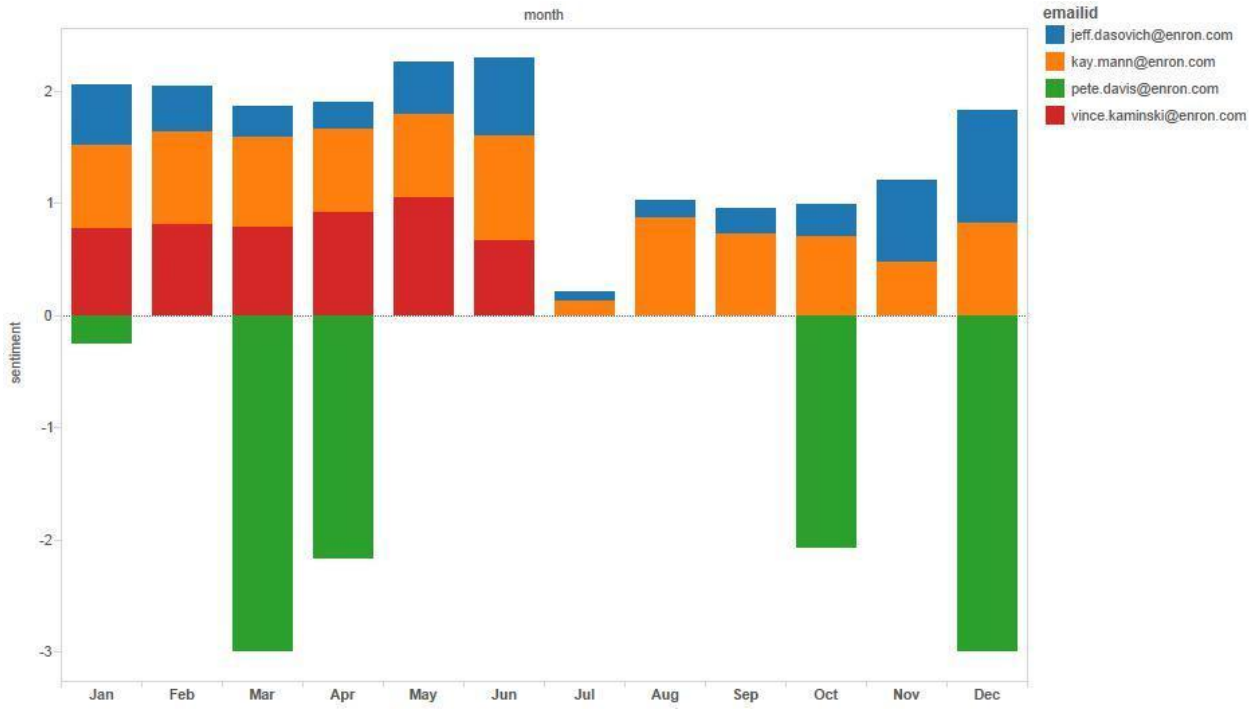
Table was populated with the following query and next we took a join of this table with AFINN list to assign a sentiment to each word:
insert overwrite table sentiment
select word,timestamp,vincewordsexploded.sentiment + afinn.sentiment as total , hashvalue
from vincewordsexploded
join afinn on (vincewordsexploded.word = afinn.words) ;

To calculate the sentiment the following formula was used

$$S(e) = \sum(S(w))/ Num(w);$$

where S(e) is the sentiment of an email , S(w) sentiment of a word and Num(w) number of words in that email.
this was achieved by the following query:

insert overwrite local directory '/home/hduser/Documents/temp/sentiments' select split(timestamp," ")[2] as mon, avg(sentiment) as avg from sentiment group by split(timestamp," ")[2] ;



One important observation that we made while running this query was the lack of data for all the critical months that is the months laters than august - december when the enron was sued for vince kaminiski, A few important observations also include the following:

1) Amongst the most frequent senders, pete davis was the only one with negative sentiments in his emails throughout the year.

2) Our theory of positive reinforcement was proved right, as 3 out of top 4 senders has a positive sentiment throughout even during bankruptcy.

3) If the distribution for Kay McCall Mann is observed we can see a clear decrease in sentiments post July , hinting that he found out about the malpractices around that timeframe.

4) Jeff dasovich followed the same trend but with a higher average sentiment value, which could possibly attribute to his job profile.

Netflix DataSet Analysis:

Aim: *To find meaningful patterns or information in the netflix data set, consisting of movies and their year of release and customer ratings for few years.*
Motivation: *We wanted to understand the way ratings are given and possibly find some patterns in the way rating are gives.. We also wanted to mine some important data from the data set.*
Note: *We reduced the number of records to be plotted by considering only top rated movies or top rating customers.*
These queries are used to import the text data into tables.

```
// import the files
create external table movietitles (
mid int, yearofrelease int, title string
) row format delimited
fields terminated by ','
stored as textfile location 's3n://spring-2014-ds/movie_dataset/movie_titles';

create external table movieratings (
mid int, customerid int, rating int, date string
) row format delimited
fields terminated by ','
stored as textfile location 's3n://spring-2014-ds/movie_dataset/movie_ratings';
```

In First query we want to find the top rated and low rated movies till now. For this we create a table so that we could use this data further in other queries as well. We initially wanted to use views but there are really slow and are calculated on the fly which is time consuming. So we created a table and stored the data and select queries are used to access this data later.

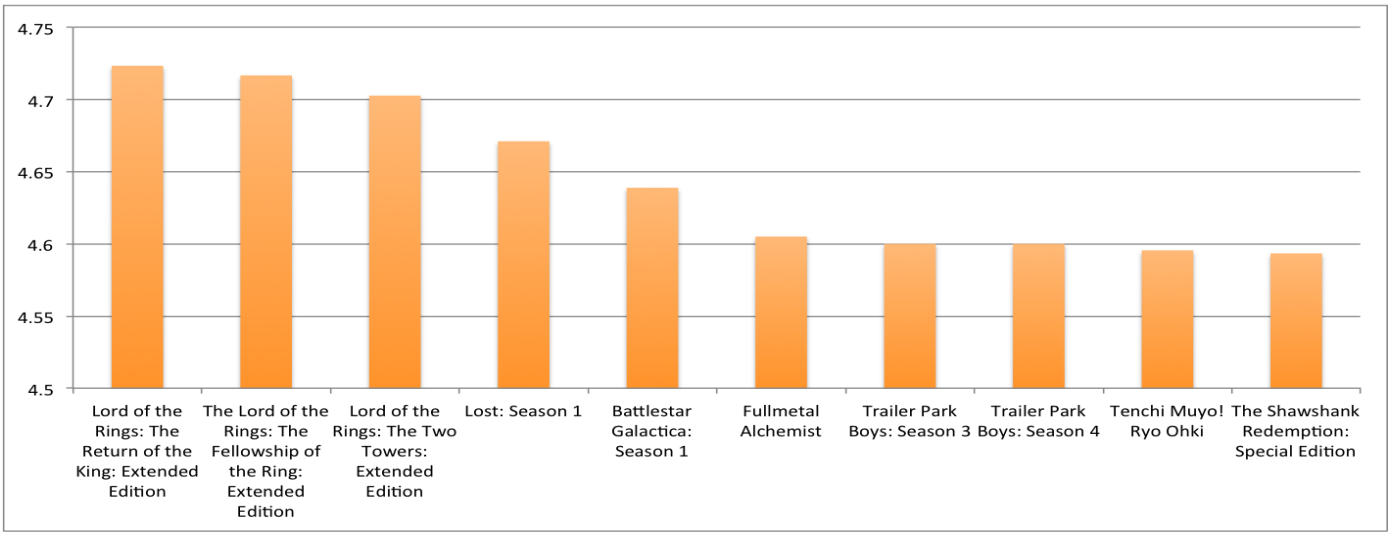
```
create table currentavgratings (mid int, title string, avgrat float);

// Total avg per movie.
insert into table currentavgratings
select mr.mid, mt.title, mr.avgrat from movietitles mt
 join (
        select mid, avg(rating) as avgrat from movieratings mr
        group by mid
) mr on (mr.mid = mt.mid);

// Highest and lowest rated movies.
```

This query gives us movies that are top rated.

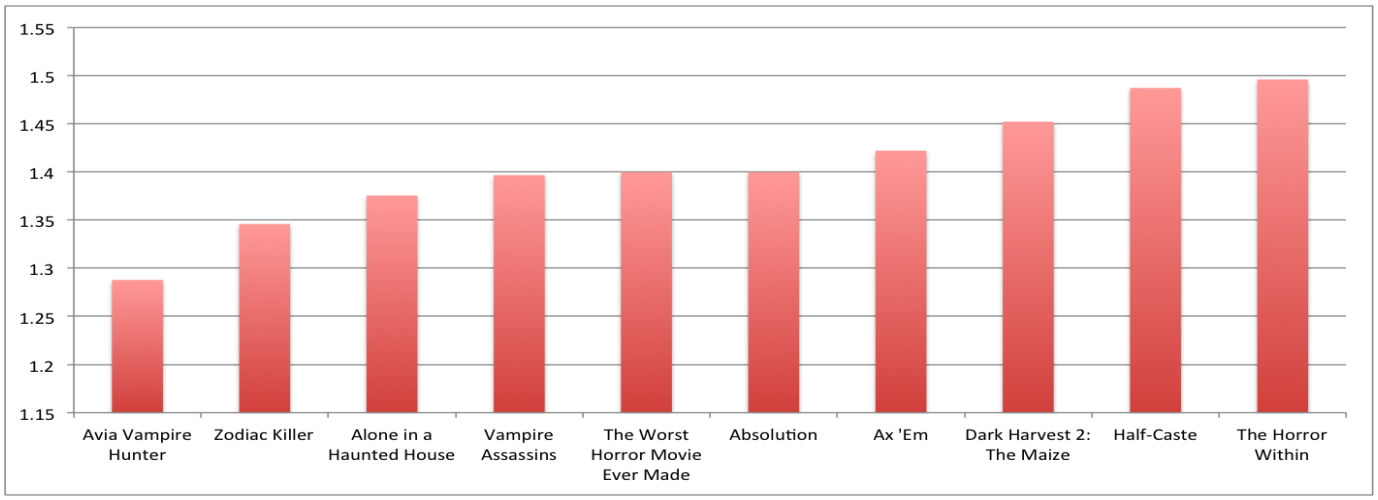
```
insert overwrite directory 's3://netflix-hive/output/highlow'
select * from currentavgratings order by avgrat desc limit 10;
```



From the above graph we can see the top rated movies by the customers

This query gives us movies that are low rated.

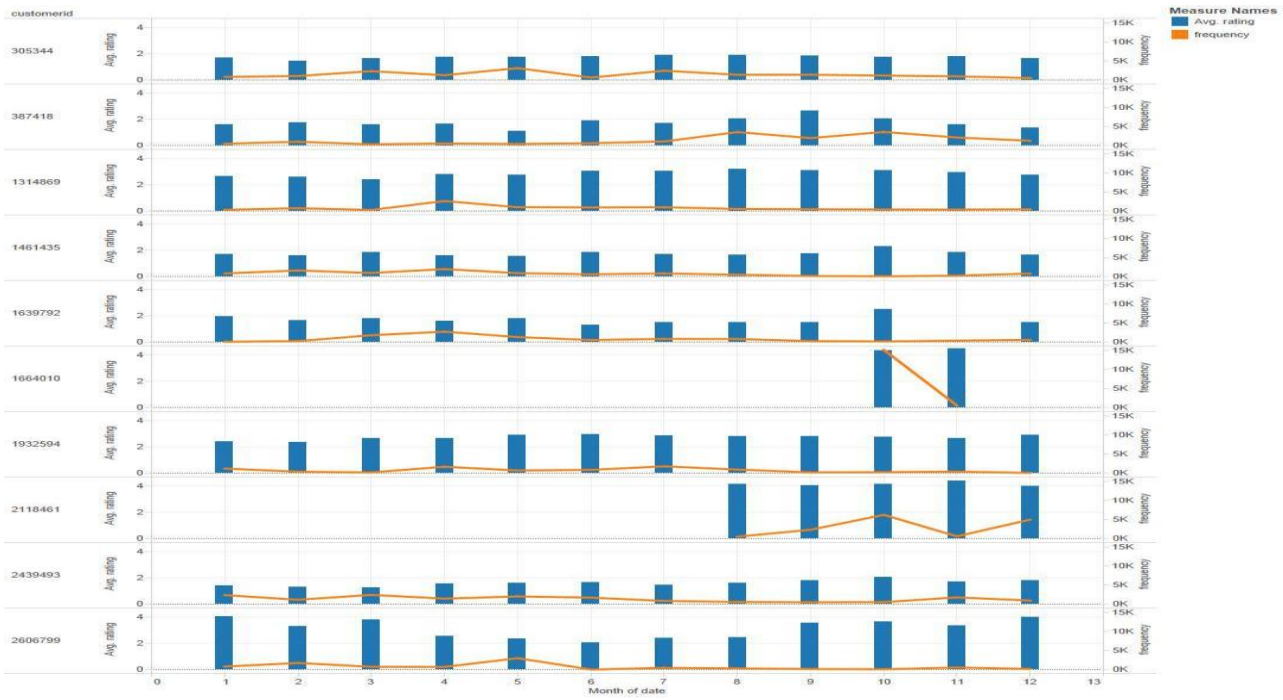
insert overwrite directory 's3://netflix-hive/output/highlow1'
select * from currentavgratings order by avgrat limit 10;



From the above graph we get the least favourite movies for the customers.

In second query we wanted to find some top reviewers from the data set and understand their daily avg for movie ratings.

//customers with most reviews date wise
insert overwrite directory 's3://netflix-hive/output/customerdatewise'
select mr.customerid, count(mr.mid), avg(mr.rating), mr.date from movieratings mr
join (
    select customerid, count(rating) as countrating from movieratings group by customerid order by countrating desc limit 10
) cus on mr.customerid = cus.customerid
group by mr.customerid, mr.date;

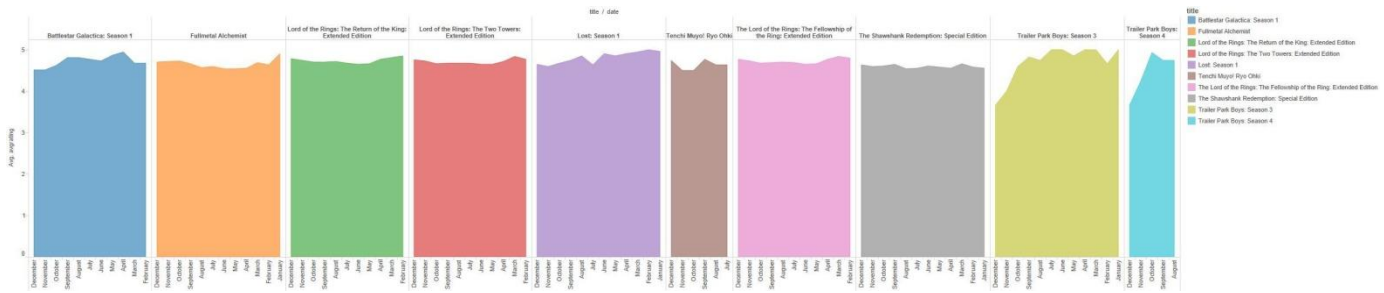


since the data is for plotting is large we had to decrease the resolution to months so that it can clearly viewed.

Third query: we wanted to find the avgrating a top rated movie gets per day.

// date wise avg ratings for movies with high ratings
insert overwrite directory 's3://netflix-hive/output/datewise'
select mt.mid, mt.title, mr.date, mr.avgrat from movietitles mt
join (
select mid, date, avg(rating) as avgrat from movieratings group by mid, date
) mr on (mt.mid = mr.mid)
join (
select * from currentavgratings order by avgrat desc limit 10
) ca on ca.mid = mt.mid ;

Note: since there are many records the tablueau couldn't generate a graph that big. so we decreased the resolution to months.



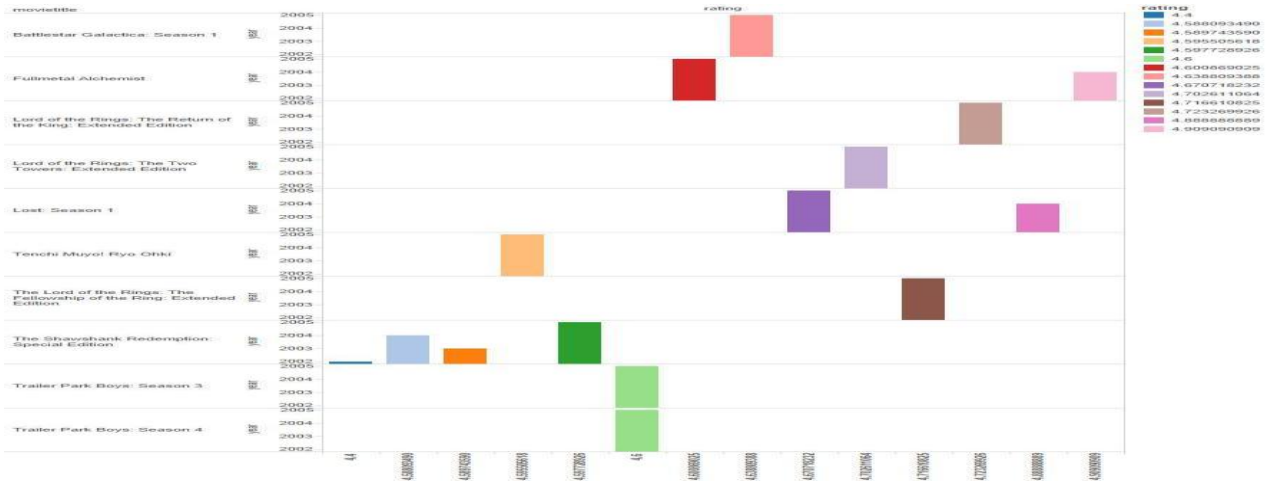
We modified the above query and decreased the resolution to find avgrating per year.

This query is to find the top movies in the year they were released.

insert overwrite directory 's3://netflix-hive/output/topmoviereleasedyearwise' select tab.mid, avg(tab.rating) as rat, count(1), tab.year as yearval from (select mr.mid, mt.title, mr.rating, mt.yearofrelease as year from movieratings mr join movietitles mt on mr.mid = mt.mid where year(mr.date) = mt.yearofrelease) tab group by tab.year, tab.mid order by yearval, rat desc;

```
// year wise avg ratings for a movies for movies with high ratings
insert overwrite directory 's3://netflix-hive/output/yearwise'
select mt.mid, mt.title, mr.year, mr.avgrat from movietitles mt
 join (
        select mid, year(date) as year, avg(rating) as avgrat from movieratings
        group by mid, year(date)
) mr on (mt.mid = mr.mid)
join (
        select * from currentavgratings order by avgrat desc limit 10
) ca on ca.mid = mt.mid ;
```

From the graph next page we can see how the ratings for the shawshank redemption changed year to year. Its interesting to know that the ratings of this movie increased in the later years. Since we have dataset for a small time period from 2002 to 2005 it should be an interesting observation if the trend followed further in the coming years. The other movies like lost 1 and full metal alchemist saw a drop in their ratings.



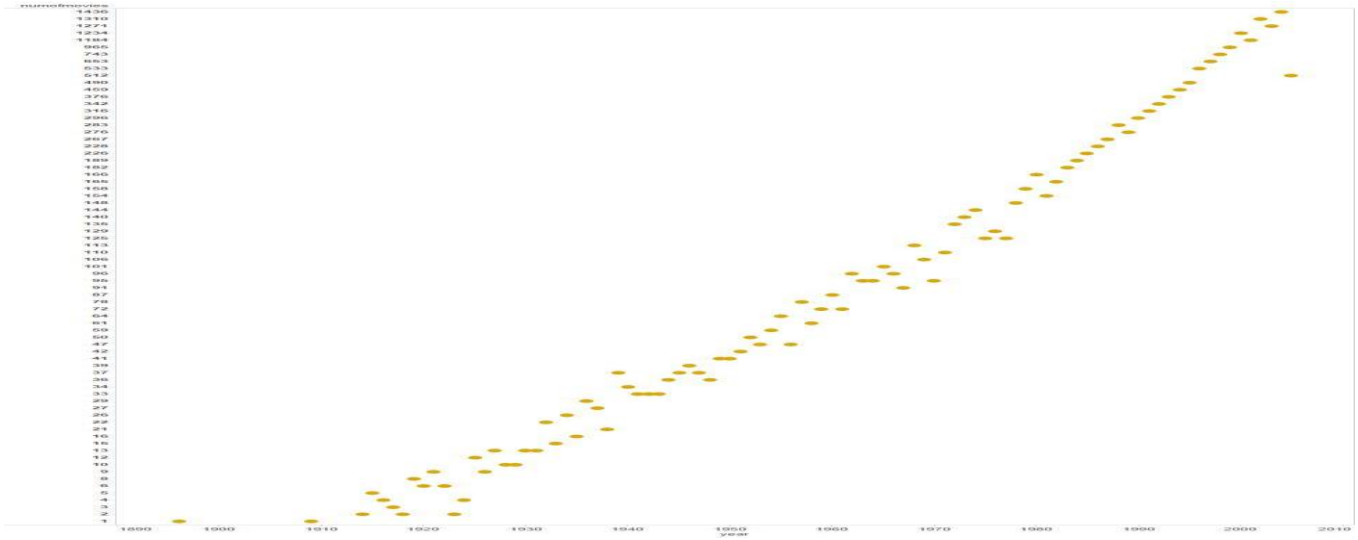
We wanted to find out the number of movies released per year

// find total movies of that year of release

```
insert overwrite directory 's3://netflix-hive/output/yearwisemoviesreleased'
select yearofrelease, count(mid) from movietitles
```

group by yearofrelease;

It is interesting to see that as years passed the number of movies coming out every year have increased significantly. One interesting thing to note here is the number of movies made during the 1939 and 1945 have decreased and stayed constant. This was the time of worldwar 2 which could have made an impact on the movie industry.



Our query next was to find out how was the rating like in the year the movie was released. To achieve this we first have to group by yearofrelease of the movie and take avg of all the ratings for movie in that year.

```
insert overwrite directory 's3://netflix-hive/output/topmoviereleasedyearwise'
select tab.mid, avg(tab.rating), count(1), tab.year from (
        select mr.mid, mt.title, mr.rating, mt.yearofrelease as year from movieratings mr
         join movietitles mt on mr.mid = mt.mid
        where year(mr.date) = mt.yearofrelease
) tab;
```

We wanted to check how the movies did in the year they were released.
There can be good reviewers or bad reviewers looking at just the numbers for ratings this cannot be understood but what we can understand from these ratings numbers are how many times a reviewer deviated further from the general opinion of other reviewers. So we can find those people that give odd reviews compared to their peers multiple times. To achieve this we found the deviation of every rating ever given by a customer from avg rating for the movie and we counted the number of times a customer is found deviated from the avg.

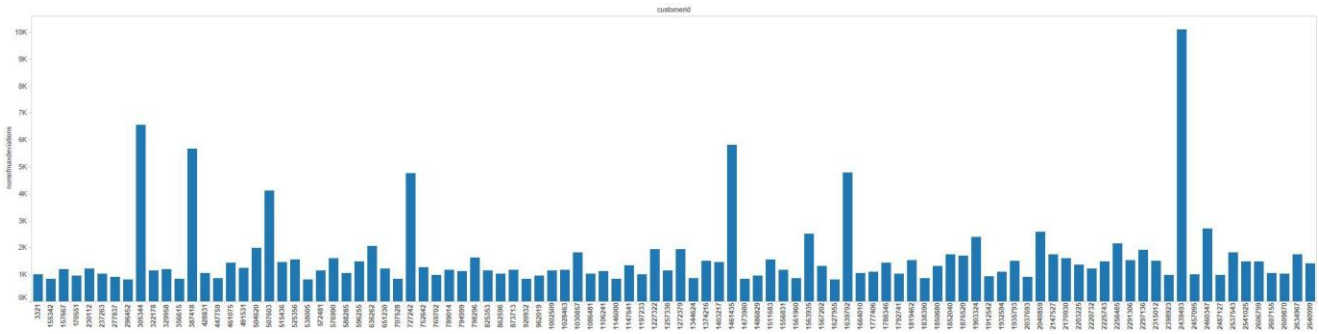
```
// find deviants in the rating
create table ratingdeviation (mid int, avgrat float, customerid int, rating int, deviation int);

insert into table ratingdeviation
select x.mid, x.avgrat, mr.customerid, mr.rating, abs(x.avgrat - mr.rating) from currentavgratings x
```

join movieratings mr on x.mid = mr.mid;

create table maxdeviation (mid int, maxdeviation float);
insert into table maxdeviation select mid, max(deviation) as negdiff from ratingdeviation group by mid;

insert overwrite directory 's3://netflix-hive/output/customerdeviation'
select z.customerid, count(z.customerid) as frequency from maxdeviation md
join ratingdeviation z on md.mid = z.mid and md.maxdeviation = z.deviation
group by z.customerid
order by frequency desc limit 100;



We plotted the top 100 customers who have consistently deviated from the avg rating for multiple movies. It is interesting because these customers may be genuinely didn't like what avg crowd likes or hates or are putting in wrong ratings.

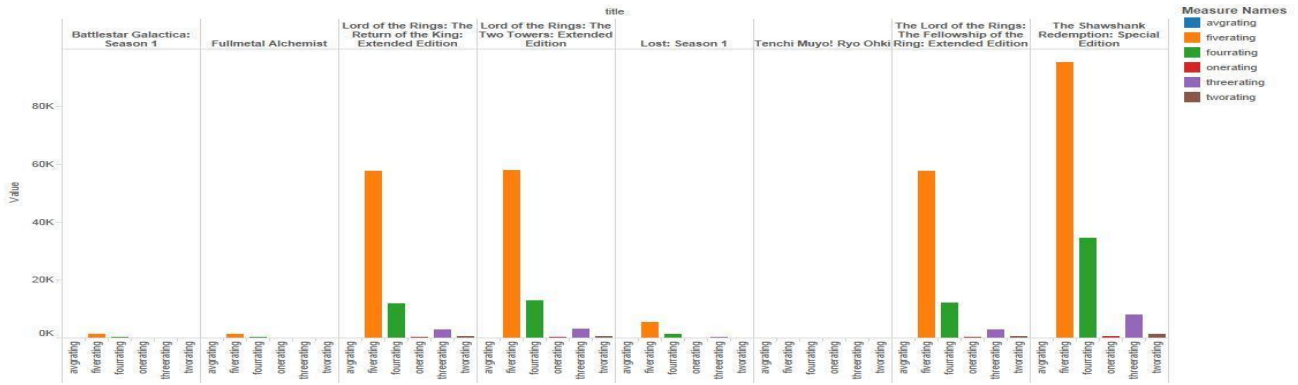
The movie ratings consist of a few low ratings, avg ratings, high ratings. We wanted to find the frequency at which a rating number is given to a particular movie. To achieve this we have to do five joins that calculate the number of times a particular rating is given to that movie.

// movie ratings frequency
create table ratingfrequency (mid int, one int, two int, three int, four int, five int);

insert into table ratingfrequency
select r1.mid, r1.countrating, r2.countrating, r3.countrating, r4.countrating, r5.countrating from (
    select mid, count(mid) as countrating from movieratings
    where rating = 1 group by mid, rating
) r1 join (
    select mid, count(mid) as countrating from movieratings
    where rating = 2
    group by mid, rating
) r2 on r1.mid = r2.mid join (
    select mid, count(mid) as countrating from movieratings
    where rating = 3 group by mid, rating
) r3 on r1.mid = r3.mid join (
    select mid, count(mid) as countrating from movieratings

where rating = 4 group by mid, rating
) r4 on r1.mid = r4.mid join (
select mid, count(mid) as countrating from movieratings
where rating = 5 group by mid, rating
) r5 on r1.mid = r5.mid;

This gives the actual frequency for each type of rating for the top rated movies.
insert overwrite directory 's3://netflix-hive/output/ratingfrequency'
select * from ratingfrequency rf join (
select * from currentavgratings
order by avgrat desc limit 10
) ca on rf.mid = ca.mid;



We were interested in knowing how the ratings were distributed for top rated movies so we queried the frequency of each type of rating and plotted it here. It is also intersting to know that lost season1, battlestar galactica , full metal alchemist although are highly rated they should be in the top 10 because the number of ratings for these movies are low.

This query gives the movies with high probabilty at which a low rating is given or will be given.
insert overwrite directory 's3://netflix-hive/output/ratio1'
select title from ratingfrequency
join movietitles on ratingfrequency.mid = movietitles.mid
order by one/(one+two+three+four+five) desc limit 10;

We also calculated the probabilty with which a movie gets a low rating (1). This can be used to calculate for any type of rating.

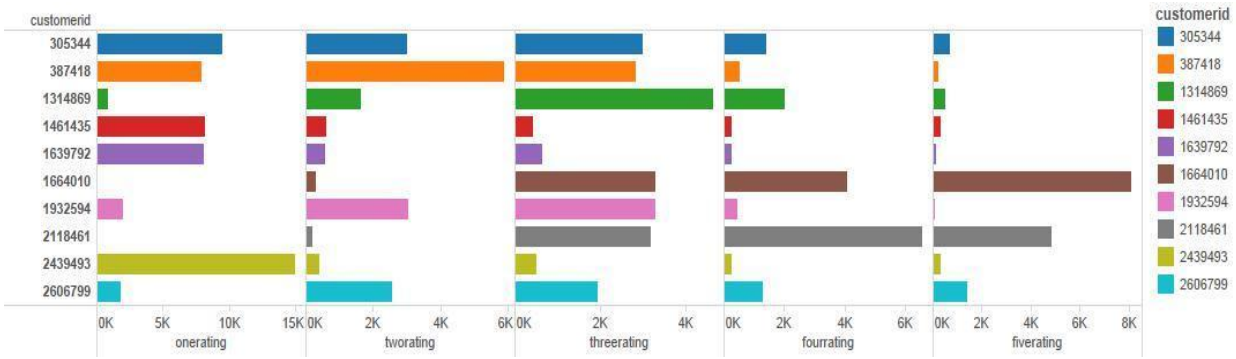
Similarly we wanted to find the frequency at which a customer gives a particular type of rating. To achieve this we have to five joins where each join finds the count of particular type of rating given by the user and combined at the end into single record.

```
// customer ratings frequency
create table customerratingfrequency (customerid int, one int, two int, three int, four int, five int);

insert into table customerratingfrequency
select r1.customerid, r1.countrating, r2.countrating, r3.countrating, r4.countrating, r5.countrating
from (
        select customerid, count(customerid) as countrating from movieratings
        where rating = 1 group by customerid, rating
) r1 join (
        select customerid, count(customerid) as countrating from movieratings
        where rating = 2
        group by customerid, rating
) r2 on r1.customerid = r2.customerid  join (
        select customerid, count(customerid) as countrating from movieratings
        where rating = 3
        group by customerid, rating
) r3 on r1.customerid = r3.customerid join (
        select customerid, count(customerid) as countrating from movieratings
        where rating = 4
        group by customerid, rating
) r4 on r1.customerid = r4.customerid join (
        select customerid, count(customerid) as countrating from movieratings
        where rating = 5
        group by customerid, rating
) r5 on r1.customerid = r5.customerid;

insert overwrite directory 's3://netflix-hive/output/topcustomerratingfrequency'
select * from customerratingfrequency rf join (
        select customerid, count(rating) as countrating from movieratings
        group by customerid
        order by countrating desc limit 10
) ca on rf.customerid = ca.customerid;
```

This graph below shows the frequency of type of ratings that top reviewers give. Its interesting to see that the customer 2439483 rates a lot of one's compared to other top reviewers. It may be conincidence that he/she always watches bad movies or he/she always consistently deviates from the crowd so we looked at the previous customer deviation graph and found that this customer constantly deviates from the avg crowd.



Similarly we found the high probability at which a customer gives a low rating for a movie.

insert overwrite directory 's3://netflix-hive/output/ratio3'
select customerid from customerratingfrequency
order by one/(one+two+three+four+five) desc limit 10;

This query can be used to estimate the probability of giving a one to movie.

some useful resources :
https://cwiki.apache.org/confluence/display/Hive/LanguageManual+Joins
http://stackoverflow.com/questions/38549/difference-between-inner-and-outer-join
http://hortonworks.com/blog/hive-cheat-sheet-for-sql-users/
http://www.youtube.com/watch?v=ShjrtAQmIVg