

ECE-653

Database Systems

Assignment 2

UNIVERSITY OF
WATERLOO



To study the Restaurant Check-in Times, Star Ratings for Restaurants and Building a Predictive Model to Predict Star Rating Based on Restaurant Check-in Time.

Submitted By

Rehan Shah

20616679

r46shah@uwaterloo.ca

TABLE OF CONTENTS

1. Hypothesis	3
2. Data Collection.....	3
3. Data Cleaning & Sanity Checking.....	5
Step 1 : Null values in Business Table.....	6
Step 2 : Keep the Businesses with Category “Restaurants”.....	7
Step 3 : Delete the N/A Values.....	8
Step 4 : Merge the Datasets.....	8
Step 5 : Aggregate Checkin Columns.....	8
Step 6 : Add Column Quartile Rating.....	9
3. Data Indexing.....	9
INDEX on Business Table.....	10
INDEX on JOIN Query.....	12
INDEX on Checkin Table.....	14
4. Data Analysis.....	15
4.1 Latent Dirichlet Allocation.....	15
4.1.1 Stars Distribution among the Restaurants.....	16
4.1.2 Check-in Times Distribution.....	19
4.1.3 Most Popular Check-in Hours.....	20
4.1.4 Least Popular Check-in Hours.....	23
4.1.5 Analysis of Check-in Hours.....	25
5. Data Validation.....	26
5.1 Building a Predictive Model using GLM.....	27
5.1.1 Subsample the Training Set	27
5.1.2 Subsample the Testing Set	27
5.1.3 Generalized linear model.....	28
5.1.4 Predicting Stars for a given Check-in Hour.....	28
5.1.5 Comparison between Training Data and Test Data Prediction.....	29
5.2 Performance of Generalized Linear Model.....	30
5.2.1 Comparison of GLM with other Classifiers	30
6. Conclusion.....	32

1.Hypothesis

Given a Checkin Time in a Restaurant, we can predict the Star Rating.

So, our goal is to:

- Study the Star Ratings given to Restaurants.
- Study the Restaurants Check-in Times
- Analyse the relation between Restaurant check-in times and star ratings for restaurants.
- To create a model for future prediction of Stars ratings based on the Check-in Times.

There is a useful Key Performance Indicator for decision making. The restaurants can use this relation in following ways :

- The restaurants can provide more incentives to customers in the low - rating hours.
- The restaurants can improve their service to improve their Yelp Rating.

Users of yelp would benefit from analyses of distributions of check in times / star ratings. A star rating by itself has little meaning without some sort of distribution backing it up. For example if every restaurant was rated 5 on yelp, then a 5 rated restaurant would not be a relatively good restaurant. Relative "goodness" is a more useful metric to a yelp user, which we can characterize after we attempt to fit a distribution to yelp restaurant stars.

As part of data analysis, following are the main steps I have covered:

- Data Collection
- Data Cleaning & sanity Checking
- Data Indexing
- Data Analysis

Data Analysis section further contains three sections:

- Analyzing Stars Distribution among the Restaurants
- Analyzing Checkin Times
- Build a Predictive Model to predict Stars Rating based on Checkin Time

2. Data Collection

The data I have used comes from the following link:

http://www.yelp.com/dataset_challenge

This data has been made available by Yelp for the purpose of the Yelp Dataset Challenge

The dataset is a single compressed file, composed of one json-object per line. Every object contains a 'type' field, which tells whether it is a business, a user, or a review. The data consists of Business objects that contain basic information about local businesses. The fields are as follows:

```
{ 'type': 'business', 'business_id': (encrypted business id),  
  'name': (business name),  
  'neighborhoods': [(hood names)],  
  'full_address': (localized address),  
  'city': (city), 'state': (state),  
  'latitude': latitude,  
  'longitude': longitude,  
  'stars': (star rating, rounded to half-stars),  
  'review_count': review count,  
  'categories': [(localized category names)] 'open': True / False (corresponds to closed,  
not business hours), 'hours': { (day_of_week): { 'open': (HH:MM), 'close': (HH:MM) },  
... }, 'attributes': { (attribute_name): (attribute_value), ... }, }
```

The raw data format is JSON. As a part of assignment 1 work, I had converted this JSON format data into the Comma Separated Values format, CSV.

Next, I worked on the dataset to clean the data, extract information about Stars rating, Checkin Hours and Predicting the Stars Distribution based on a given Checkin Hour.

3. Data Cleaning & Sanity Checking

We have created schema for Yelp Dataset in assignment 1. There, we have created Entity Relationship model for the Yelp dataset and created schema based on the ER-model. Here, based on our requirements, we are interested in Star Ratings and Check-in Times. I have considered two tables for this particular analysis :

- Business Dataset
- Checkin Dataset

In *Business* Table, we are interested in two columns, *Business_id* and *Stars*.

The *business_id* to identify a particular business and to merge the data with the checkin table. *Business_id* field is present in *Business* as well as in *Checkin* tables. So ,I have used *business_id* field to join these two tables, so that I can extract the useful information from the *Business* table. From this table, we will be getting the *stars* (Stars ratings) given to a business.

Also, the *Business* table has a field *stars*, which represent the Star Rating given to a particular business by the users. We need this column for our analysis to check the Star Ratings corresponding to a particular Checkin Time. I have extracted this column from the *Business* table and merged this data with *Checkin* table as explained in the following sections.

In *Checkin* Table, we have the *checkin_info*, *business_id* and *type* columns. *Checkin_info* column gives the count of customer checking into a particular *business_id* (Restaurants in our case) at a given time. For example, we have the following values from *Checkin* table :

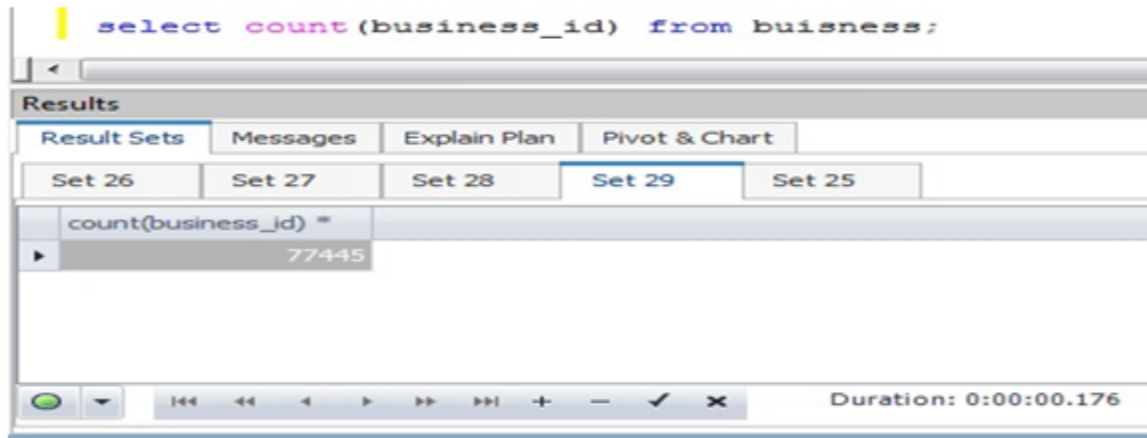
Checkin_info.9-5	Business_id
4	b9WZJp5L1RZr4F1nxclOoQ

It can be read as 4 customers have checked into Restaurant with ID 'b9WZJp5L1RZr4F1nxclOoQ' at 9 am on Friday.

So, we have the data from the Yelp dataset and we need to perform Data Cleaning steps, which I have explained in following sections.

Step 1 : Null values in Business Table

We have total Count of **77445** of distinct *business_id* values in *Business* table as shown below:

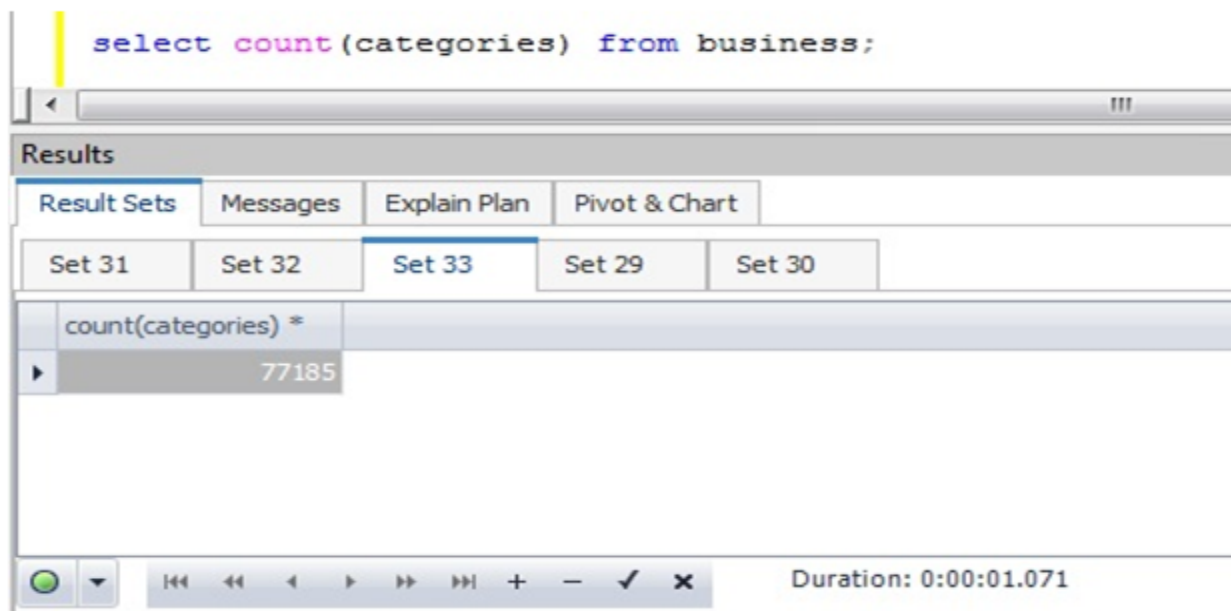


The screenshot shows a SQL query window with the following SQL statement: `select count(business_id) from buisness;`. The results pane displays a single row with the value 77445. The interface includes tabs for Result Sets, Messages, Explain Plan, and Pivot & Chart. Below the tabs are buttons for Set 26, Set 27, Set 28, Set 29 (selected), and Set 25. The results table has a header row `count(business_id) *` and a data row with the value 77445. The bottom status bar shows the duration as 0:00:00.176.

count(business_id) *
77445

Fig 1 : Checking Business_id Count in Business Table

While checking the corresponding values of distinct *categories* in *business* table, I got **77185** rows for column categories.



The screenshot shows a SQL query window with the following SQL statement: `select count(categories) from business;`. The results pane displays a single row with the value 77185. The interface includes tabs for Result Sets, Messages, Explain Plan, and Pivot & Chart. Below the tabs are buttons for Set 31, Set 32, Set 33 (selected), Set 29, and Set 30. The results table has a header row `count(categories) *` and a data row with the value 77185. The bottom status bar shows the duration as 0:00:01.071.

count(categories) *
77185

Fig 2 : Checking Categories Count in Business Table

To analyze the data, I needed *categories* as Restaurants. So I excluded these rows from the table *Business*, where we have no category corresponding to a *business_id*.

Step 2 : Keep the Businesses with Category “Restaurants”

Since we are interesting in Restaurants only, we need to filter out all non-restaurant related *check-in* times. SO, I have deleted all the non-restaurant type business records.

To check the restaurant businesses from the *Business* table, I used the following query :

```
Select Count(business_id) FROM Business WHERE buisness_id IN (SELECT b.buisness_id FROM Business b, Categories c where b.buisness_id=c.buisness_id AND c.category IN ('Restaurants'));
```

The count of *Businesses* with type Restaurants is **25141** as show below :

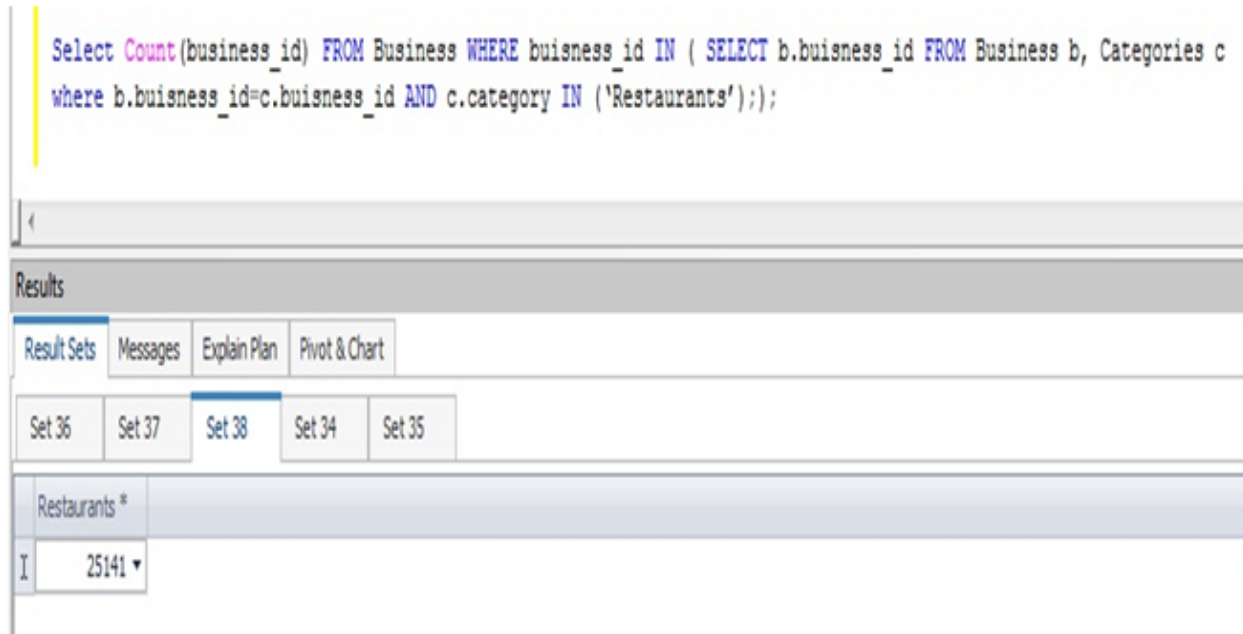


Fig 3 : Businesses With Type ‘Restaurant’

The following query is used to delete the non-restaurant type businesses:

```
DELETE FROM Business WHERE buisness_id IN ( SELECT b.buisness_id FROM Business b, Categories c where b.buisness_id=c.buisness_id AND c.category NOT IN ('Restaurants'));
```

Step 3 : Delete the N/A Values

There are lot of not-available *check-in* values. For our analysis, I am assuming that there is no customer checking-in at that particular time. So I have replaced all the not available values with the 0 for the analysis.

To replace the N/A values with zeros, I have used the following command :

```
checkin_stars[is.na(data_checkin_stars)] <- 0
```

Step 4 : Merge the Datasets

We have datasets from *Business* and *Checkin* tables. To have *check-in* Times, *stars* and *business_id* attributes, we need to merge the data from both the tables. We can do it with *buisness_id*, which is a common attribute.

The merge the columns from both tables, I have used the following commands:

```
loc <- grep("R,e,s,t,a,u", data_bus$categories, perl=TRUE, value=FALSE)
data_bus_subset <- data_bus[loc,]
data_subset = subset(data_bus_subset,select=c("stars","business_id"))
checkin_stars <- merge(data_checkin,data_subset,by="business_id")
```

Step 5 : Aggregate Checkin Columns

We have taken *buisness_id* attribute as Restaurant identifier as explained above. After that, I have aggregated the *Checkin* columns and the result of aggregation of the checkins is stored in one new column *Checkin_Total*. I have used this column to analyze the total checkins for a given *business_id*. For analyzing, we need to aggregate the individual checkins for different hours into a one column corresponding to a *business_id*.

Step 6 : Add Column Quartile Rating

After adding the *Checkin_Total* column, I have added one new column *Quartile_Rating*. The value of the *Quartile_Ratng* column could be any of these mentioned below :

- High : top 25 %
- Medium : Middle 50 %
- Low : Bottom 25 %

The following commands are used to perform the above operation :

```
checkin_stars$quant_rating <- "Medium"
```

```
checkin_stars[good,]$quant_rating<-"High"
```

```
checkin_stars[bad,]$quant_rating<-"Low"
```

By performing above Data Cleaning and enhancement, we have the following format of the data for analysis :

ROWS : The rows in our final dataset represented each restaurant. One additional row was added, which was aggregation of checkins per time slice across Restaurants.

Columns : The columns represented a *check_in* time for a particular hour/day of the week, in the form of *checkin_info_18.5* where 18 indicates the hour (6 pm) and 5 represents the day of the week (Saturday), from this we have 7*24 or 168 columns.

Three Additional Columns were added, *total_checkins_rest* for a restaurant along with its *star* rating(1-5), an overall *quant_rating* ("High", "Med", "Low").

3. Data Indexing

Indexes are one of the best ways to improve performance in a database application. In Database, a table scan happens when there is no index available to help a query. In a table scan, SQL Server examines every row in the table to satisfy the query results. Table scans are sometimes unavoidable, but on large tables, scans have a terrific impact on performance. An index helps when you are looking for a specific record or set of records with a WHERE clause. This includes queries looking for a range of values, queries designed to match a specific value, and queries performing a Join on two tables.

The tables I used for the analysis are *Business* and *Checkin*. In the *Business* table, As explained earlier, I needed two columns from Business table for the analysis purposes, *business_id* and *stars*.

The queries I used for cleaning and analysis purposes were the select and order by queries. To analyze the performance of Indexes on the table *Business*, I executed the steps as explained below.

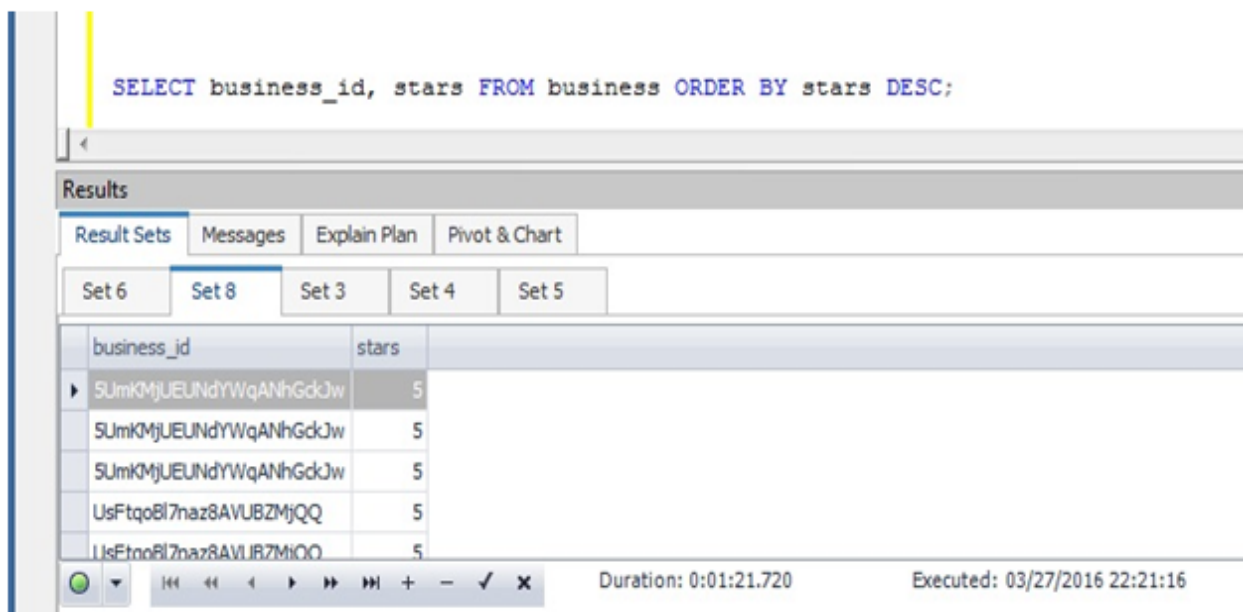
3.1 INDEX on Business Table

First, I created INDEX on stars column by running the following query:

```
CREATE INDEX stars ON business(stars);
```

Then I run the query to select the *business_id* and *stars* from *Business* table in descending order of *stars*:

```
SELECT business_id, stars FROM business ORDER BY stars DESC;
```



The screenshot shows a SQL query window with the following text: `SELECT business_id, stars FROM business ORDER BY stars DESC;`. Below the query window, the 'Results' tab is active, displaying a table with two columns: *business_id* and *stars*. The table contains five rows of data. The first three rows have the same *business_id* and a *stars* value of 5. The last two rows have different *business_id* values and a *stars* value of 5. At the bottom of the results window, the 'Duration' is 0:01:21.720 and the 'Executed' time is 03/27/2016 22:21:16.

business_id	stars
SUmKMjUEUNdYWqANhGckJw	5
SUmKMjUEUNdYWqANhGckJw	5
SUmKMjUEUNdYWqANhGckJw	5
UsFtqoBl7naz8AVUBZMjQQ	5
UsFtqoBl7naz8AVUBZMiOO	5

Fig 4 : Select Query With INDEX

The time taken by the query to execute is **0:01:21:720**

After noting the execution time of the select query, the next step is to drop the Primary Key and Foreign Key constraints and drop the Index. , I removed the Primary Key and Foreign Key constraints and also dropped the index *stars* , I had created earlier on the *stars* column from *business* table as:

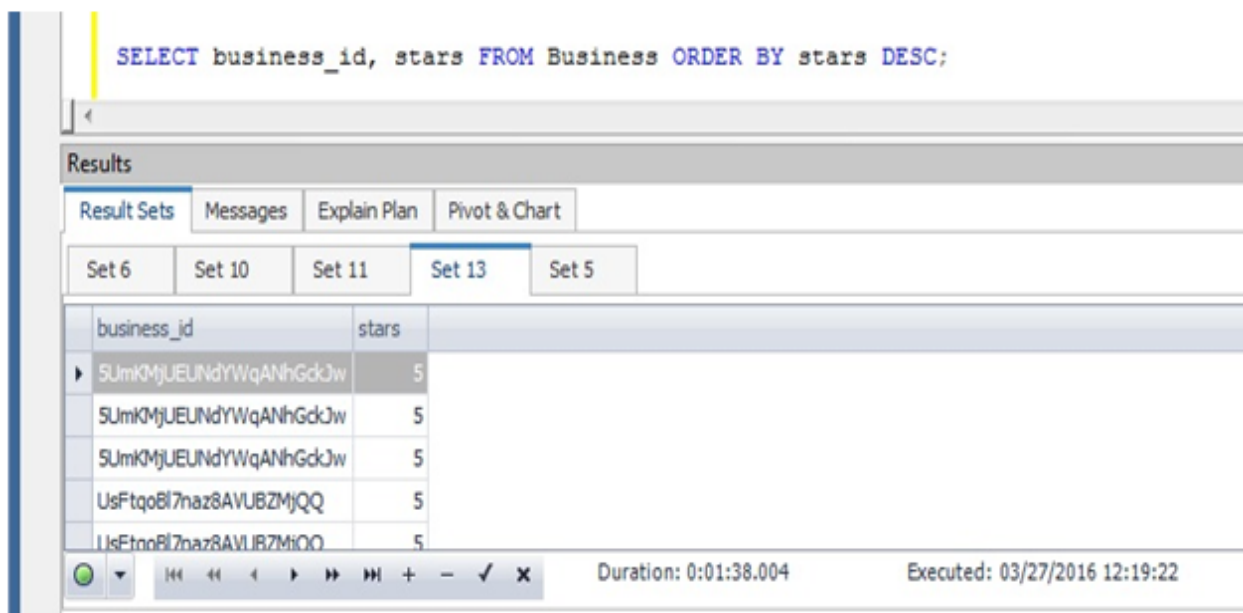
```
ALTER TABLE BUSINESS DROP CONSTRAINT pk_business_id;
```

```
ALTER TABLE BUSINESS DROP CONSTRAINT fk_category_id;
```

```
ALTER TABLE business DROP INDEX stars;
```

After removing Primary Key & Foreign Key constraints and dropping index *stars* from the *Business* table , I executed the select query, where I selected the *business_id* and the *stars* rating in descending order of *stars*. The query is as given below :

```
SELECT business_id, stars FROM business ORDER BY stars DESC;
```



The screenshot shows the SQL Server Enterprise Manager interface. The query editor at the top contains the query: `SELECT business_id, stars FROM Business ORDER BY stars DESC;`. Below the query editor, the 'Results' tab is selected, showing a table with two columns: *business_id* and *stars*. The table contains five rows, all with a *stars* value of 5. The *business_id* values are truncated. The status bar at the bottom indicates the query duration is 0:01:38.004 and it was executed on 03/27/2016 at 12:19:22.

business_id	stars
5UmKMjUEUNdYWqANhGdcJw	5
5UmKMjUEUNdYWqANhGdcJw	5
5UmKMjUEUNdYWqANhGdcJw	5
UsFtqbI7naz8AVUBZMjQQ	5
UsFtqbI7naz8AVUBZMjQQ	5

Fig 5 : Select Query Without INDEX, Primary Key & Foreign Key

As we can see, the time taken by the query to execute is **0:01:38:004**

Since index entries are stored in sorted order, indexes help when processing ORDER BY clauses. Without an index the database has to load the records and sort them during execution. An index on *stars* allowed the database to process the above query by simply scanning the index and fetching rows as they are referenced.

To order the records in descending order, the database can simply scan the index in reverse.

3.2 INDEX on JOIN Query

To check the performance of Join queries, I created INDEX on columns and then checked the execution time to analyze the performance of INDEX. We need to Join two tables, *Business* and *Categories* to extract the Restaurants from all other categories. Following is the query I am using for extracting the Restaurants from all categories:

```
SELECT b.business_id FROM Business b, Categories c where b.business_id=c.business_id AND c.category IN ('Restaurants');
```

Here , the filter is on column *business_id* of table *Business*. So, the table *Business* needs an index on column *business_id*. Following query is used to create INDEX on *business_id* column of *Business* table:

```
CREATE INDEX business ON Business(business_id);
```

Similarly, table *Categories* needs an INDEX on column *business_id*. Following query is used to create INDEX on *business_id* column of *Categories* table:

```
CREATE INDEX business_categories ON Categories(business_id);
```

MySQL read each row of table *Business* using the INDEX on *b.business_id*. MySQL then used the INDEX on *c.business_id* to join *Categories* to *Business*. We run the query to check the businesses with category 'Restaurants':

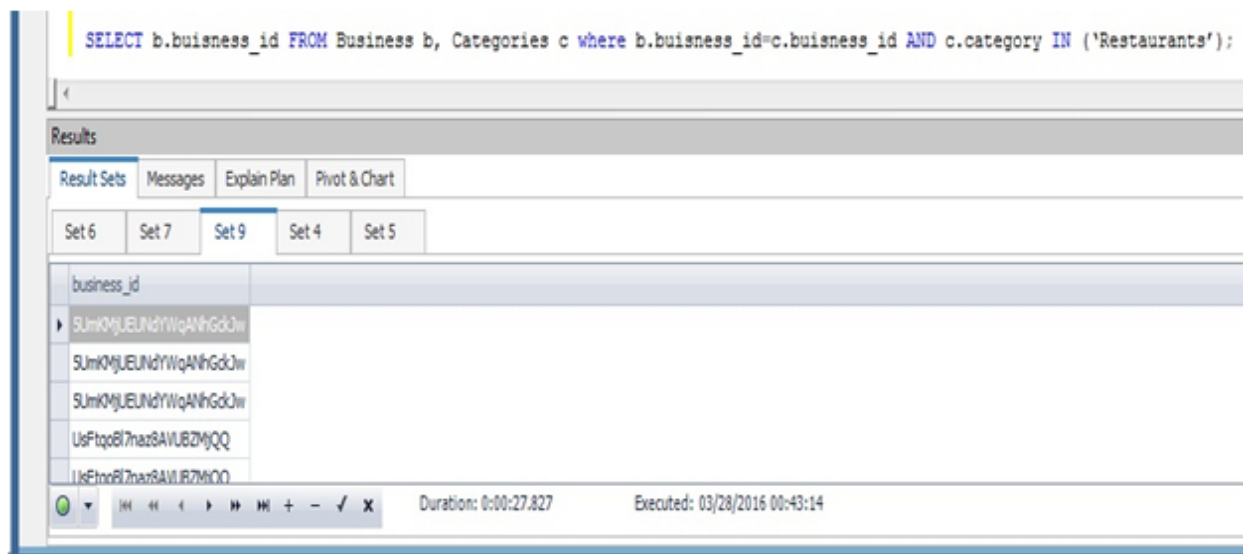


Fig 6 : Select Query With INDEX

The time taken by the query to execute is **0:00:27:827**

After this, I removed all the Primary Key and Foreign Key constraints and dropped the INDEX from the *business_id* field in the Business table by running the following query:

```
ALTER TABLE Business DROP INDEX business;
```

Also, I deleted the INDEX from the *Categories* table:

```
ALTER TABLE categories DROP INDEX business_categories ;
```

Then I run the same query to select the only businesses with category 'Restaurants':

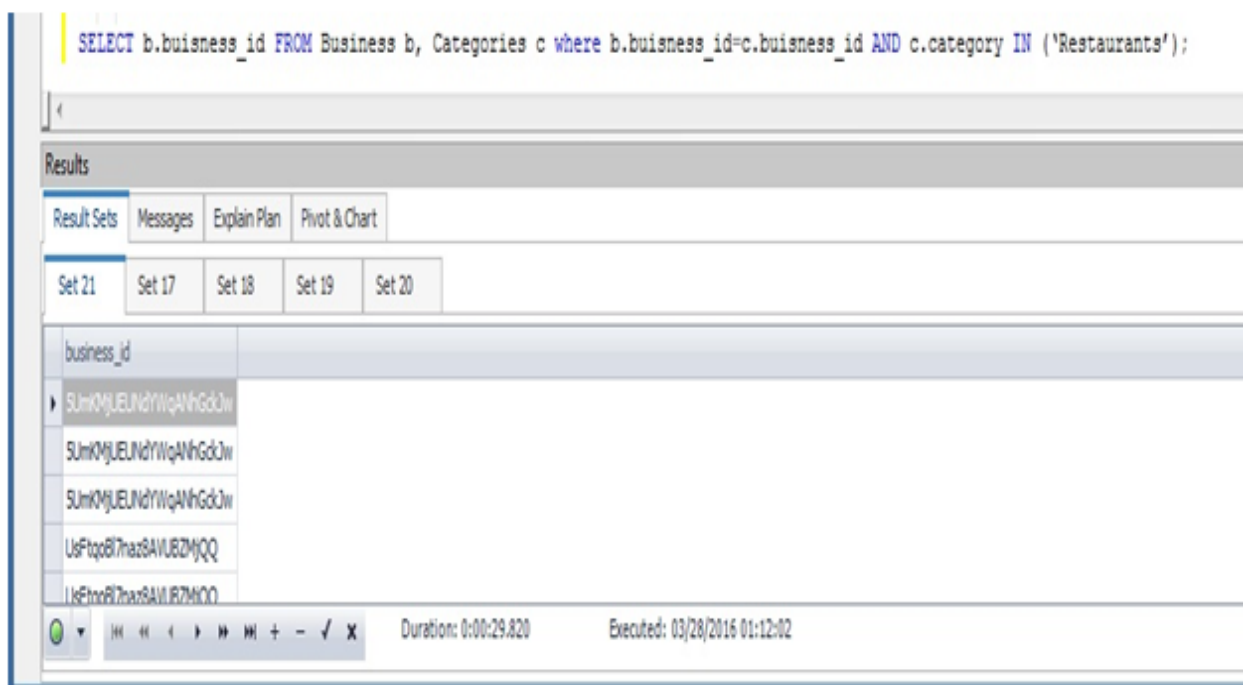


Fig 7 : Select Query Without INDEX, Primary Key & Foreign Key

As we can see, the time taken by the query to execute is **0:00:29:820**

3.3 INDEX on Checkin Table

Next, I studied the impact of INDICES on the Checkin table. The following query was used to check the number of total checkins in *checkin_info.18-4*. While analyzing, I have found that maximum number of checkins happened at 6pm, Thursday. The corresponding query to check the total number of checkins for *checkin_info.18-4* :

```
select sum('checkin_info.18-4') as Maximum_Checkins from checkin where business_id in (SELECT b.buisness_id FROM Business b, Categories c where b.buisness_id=c.buisness_id AND c.category IN ('Restaurants'));
```

Before checking the sum of checkins in *checkin_info.18-4*, I created INDEX on business_id by running the following query :

```
CREATE INDEX business ON Checkin(business_id);
```

Then I run the above mentioned query to check the total checkins in *checkin_info.18-4* as shown below :

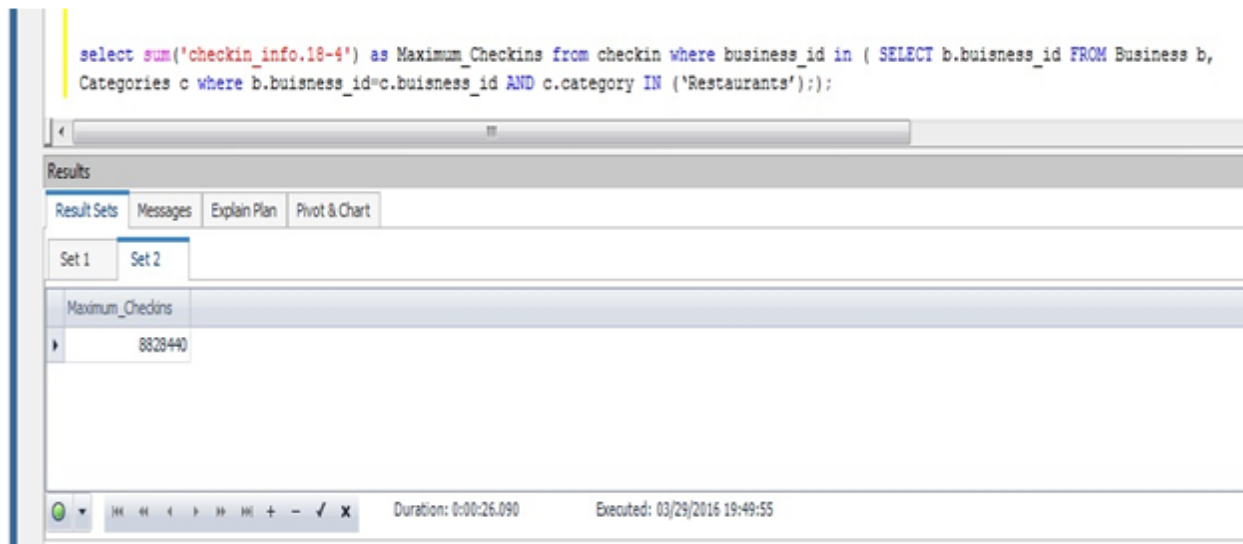


Fig 8 : Select Query With INDEX

The execution time of the select query is **0:00:26:090** as shown above.

After this, I removed all the Primary Key and Foreign Key constraints and dropped the index from the *buisness_id* field in the Checkin table by running the following query:

```
ALTER TABLE Checkin DROP INDEX business;
```

Then I run the same query to check the total checkins in *checkin_info.18-4* as shown below:

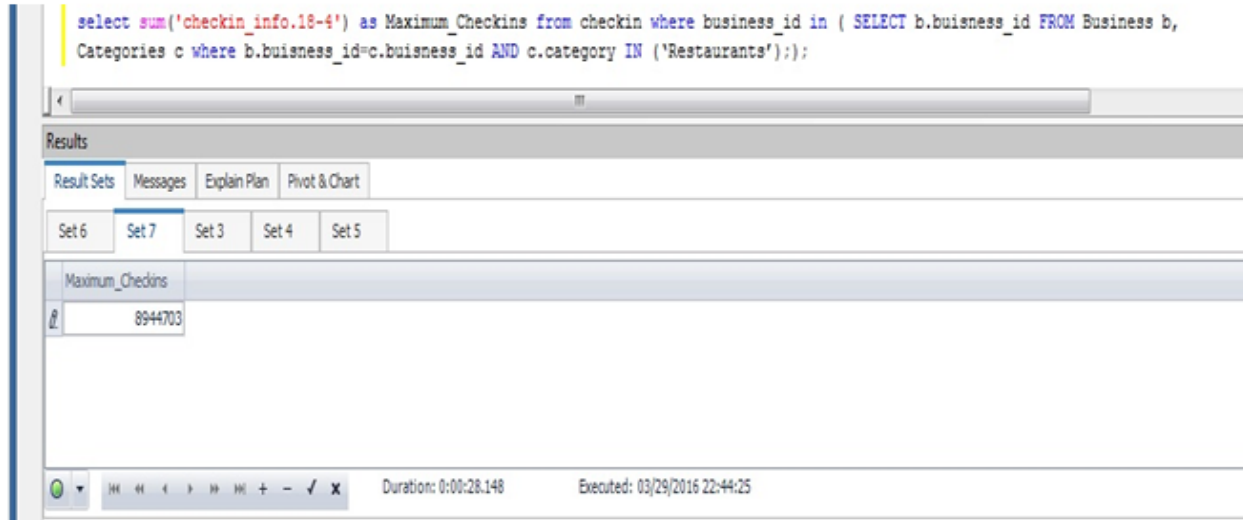


Fig 9 : Select Query Without INDEX, Primary Key & Foreign Key

The execution time of the select query is **0:00:28:148** as shown above.

It can be seen that by creating indexes on columns which are frequently used in where conditions, the execution time of the select queries has improved. I have retained Indices on *business_id* and categories table (*business_id*) for query execution time improvement.

4. Data Analysis

By Data Cleaning & Enhancement process, we have prepared our data for the analysis. Now, we will analyse the data based on our Key Performance Indicator, which is "Relation between Check-in Times and Stars Rating". I studied the distribution of stars among restaurants,

What are the most popular check-in hours based on the check-in data by the users and the stars ratings, I tried to perform Latent Dirichlet Allocation (LDA) for topic modelling.

4.1 Latent Dirichlet Allocation (LDA)

Using Latent Dirichlet Allocation, we can infer topic distribution to build a model to classify whether a check-in hour is among the popular check-in hours.

- Fit the distribution of Stars Rating to Restaurants.
- Fit the distribution of Check-in Times .

4.1.1 Stars Distribution among the Restaurants

I tried to fit normal as well as coucy distribution.

```
checkin_stars_only <- data_checkin_stars[1:169]
star_dist <- fitdistr(data_checkin_stars_only$stars, "normal")
star_dist1 <- fitdistr(data_checkin_stars_only$stars, "cauchy")

qqplot(data_checkin_stars$stars, Cauchy_dist_quartiles,xlab="Cauchy-Fit QQ Plot"
        ,ylim=ylim,ylab="Sampled Quartiles",main = "Cauchy QQ Plot")
abline(0,1)

qqplot(data_checkin_stars$stars, normal_dist_quartiles,xlab="Normal-Fit QQ Plot"
        ,ylim=ylim,ylab="Sampled Quartiles",main = "Normal QQ Plot")
abline(0,1)
```

Based on above distribution analysis, I have got the following Quartile distribution of Stars among the Restaurants :

Quartile	Stars Rating
Minimum	1.000
1st Quartile	3.000
Median	3.500

Quartile	Stars Rating
Mean	3.459
2nd Quartile	4.000
Maximum	5.000

Table 1: Distribution of Star Rating Among Restaurants

The table shows that Mean value of Star Rating distribution among Restaurants is 3.459. The Median of Star Rating distribution is 3.500, first 25 % lies in Star Rating 3.000, while Star Rating 4.000 is for 2nd Quartile.

The plot of Star Ratings given to Restaurants Vs. Frequency of Restaurants is as shown below. We are comparing the Normal Distribution Fit and the cauchy Distribution Fit.

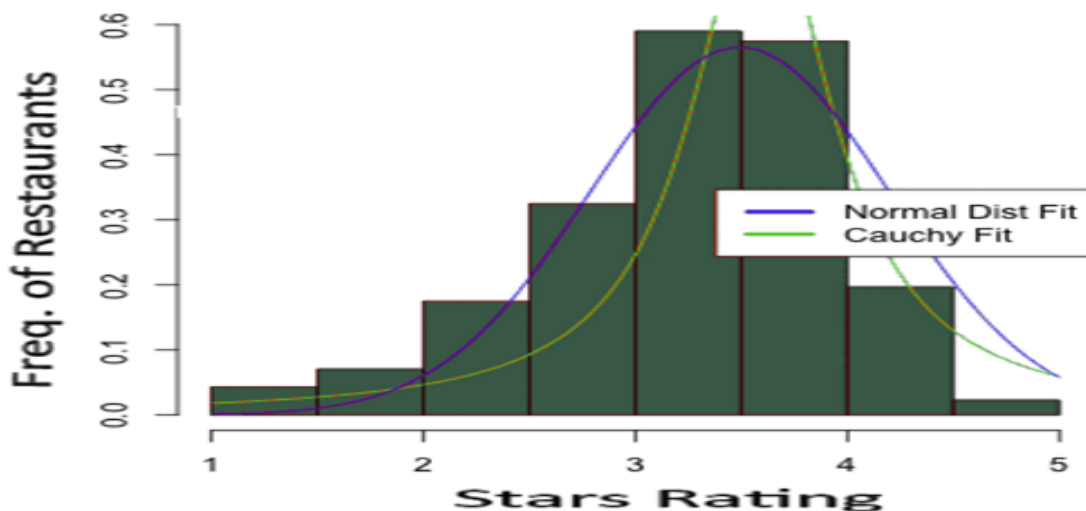


Fig 10 : Star rating and Frequency of Restaurants

I have categorized the Normal distribution of Stars rating, with Maximum Likelihood Estimation values as below :

Mean : 3.46

S.E. : 0.73

The quartiles matching up on the QQ plot, small standard errors for parameter estimates, and the fairly symmetrical distribution (median = 3.5, mean=3.46) all signify that **$N(3.46, .73)$** is a good fit. Attempts to fit a distribution with a fatter tail, e.g. Cauchy, yielded a similar, but worse fit overall along with higher avg. errors, and worse QQ-plot fits. Furthermore, the Cauchy distribution would sometimes predict numbers such as -32 which could never possibly be in the range of the 1-5 star rating yelp system.

People tend to rate restaurants higher than they are "supposed to be rated"

I expected the average restaurant to get a review of 3 (the average of 1 and 5), but it is actually 3.5, suggesting people tend to rate restaurants higher than they are "supposed to be rated". Furthermore, by using properties of the normal distribution, namely the 3 sigma rule, I can say that stars roughly translate into these percentiles:

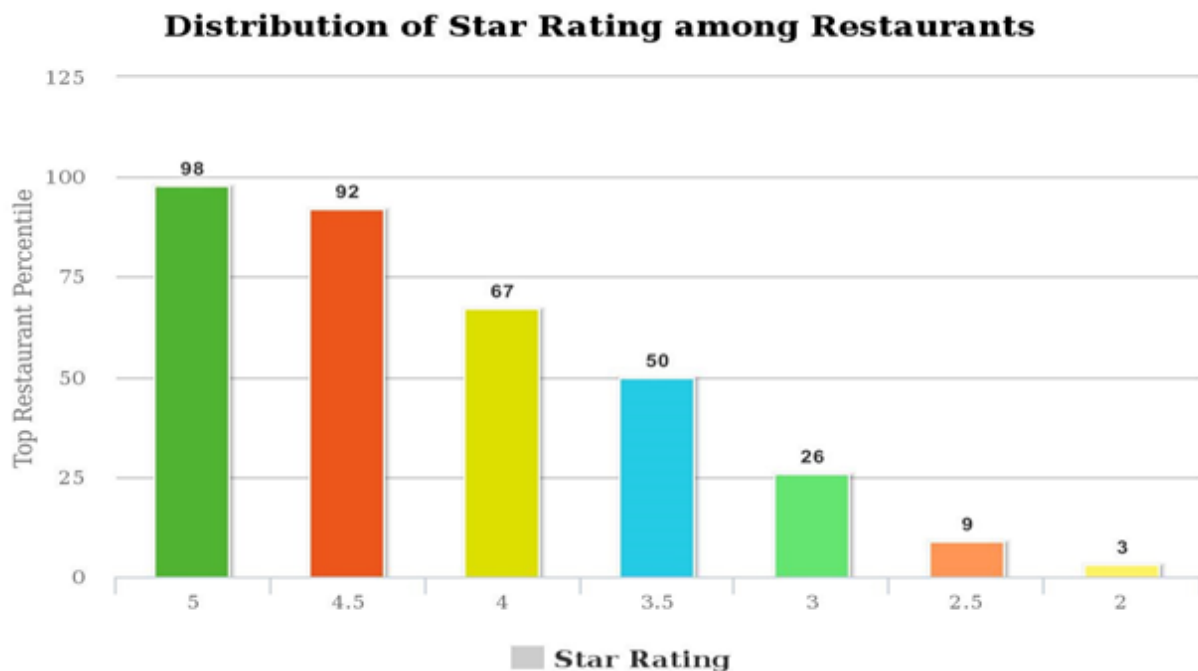


Fig 11 : Distribution of Star Rating Among Restaurants

The values in the table are consistent with the distribution, but on the tails, the results I got are higher than the distribution fit. I have also tried Cauchy Distribution with fatter tails, but the results are not better than the normal fit. I am concluding that the distribution is normal with slightly fatter tails, due to outliers or noisy data.

4.1.2 Check-in Times Distribution

I have analyzed the distribution of Number of Check-in's per hour. The results of distribution from histogram are :

- Log - Normal
- Exponential
- Gamma Distribution

I did not consider the Log- Normal plot of the ranked data by plotting the log-log plot. The plot is not linear, as we can see an exponential relationship. This exponential relationship describes the number of times events occurs in a particular time slice.

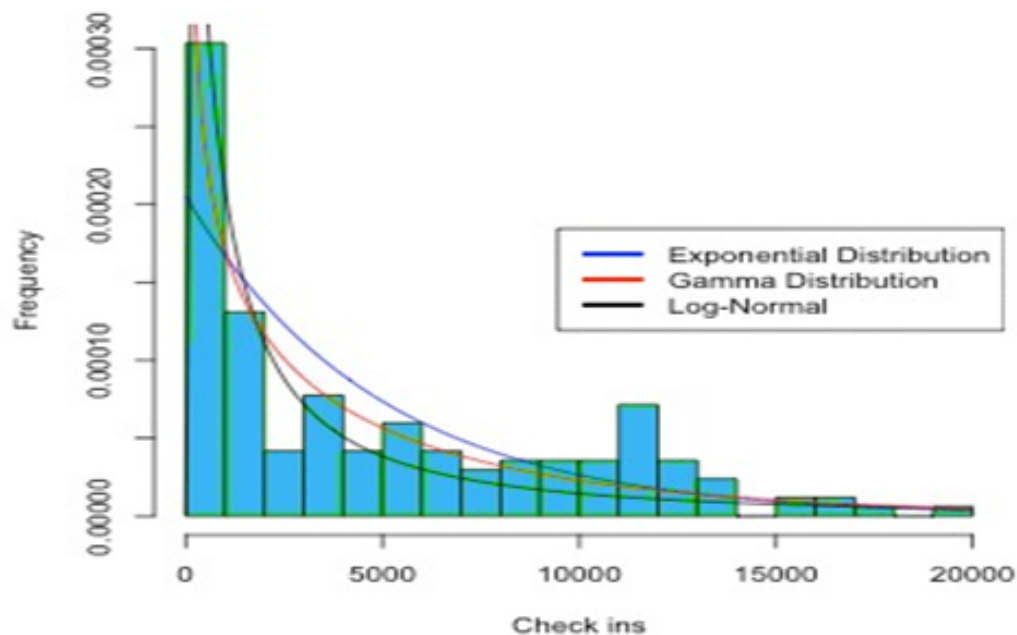


Fig 12 : Number of Check-ins in a given Time Slice

By aggregating the exponentially distributed variables, we can infer that this is Gamma Distribution of Check-in Times. From the QQ plots, we can see that the gamma QQ plot is almost a perfect fit until we reach roughly the third quartile. The sharp change in slope indicates that there reaches a point where the distribution changes from a gamma to one with fatter tails, reflecting a power law for the top 25 percentile of check in times.

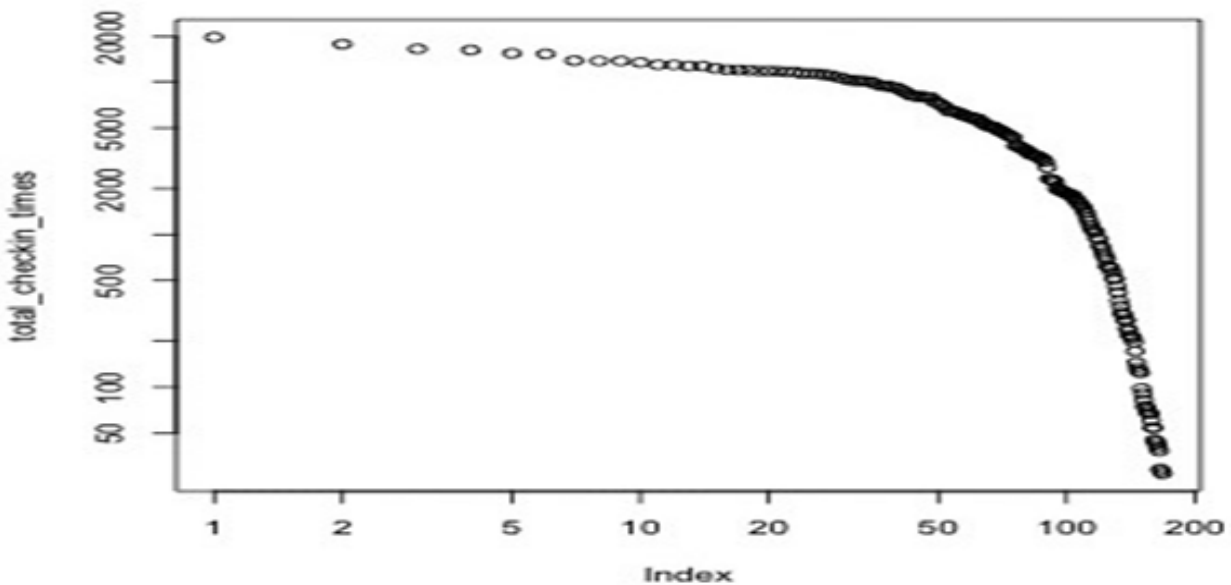


Fig 13 : Log-Log Plot of Total Checkins vs Index

4.1.3 Most Popular Check-in Hours

After analyzing the Check-in Times distribution, I analyzed the Number of Check-in Times to see how many customers checked into the restaurant's time/days wise. From this, I tried to analyze the most popular times and days among the customers. We have got the check-in time data, where the number of customers checking into a particular restaurant at a particular time is given. From this data, we can analyze the popularity of different time slices among the people.

Also, from the results, restaurants can analyze what are the factors behind the popularity of particular time slices and days and figure out what needs to be improved to increase the customers check-in in the other days as well.

To plot the 10 Most popular Check-in Times, I have used the following script :

%%R

unranked_total_checkin_times <- colSums(raw_checkin_times)

plot(Total_Checkin_Times, main= " Plot")

plot.ts(cbind(unranked_total_checkin_times[15:84]), main= "Partial Time series Plot")

Based on the analysis work done to check the Most Popular Check-in Times, I have got the following results :

No	Check-in Time	Number of Check-ins
1	checkin_info_18.4	21056
2	checkin_info_19.4	19113
3	checkin_info_17.4	18165
4	checkin_info_18.5	17524
5	checkin_info_12.4	16298
6	checkin_info_19.5	15049
7	checkin_info_18.3	13825
8	checkin_info_12.5	12441
9	checkin_info_11.4	11578
10	checkin_info_17.5	11142

Table 2 : Most Popular Checkin Hours

The Most Popular Check-in Hours are plotted as below :

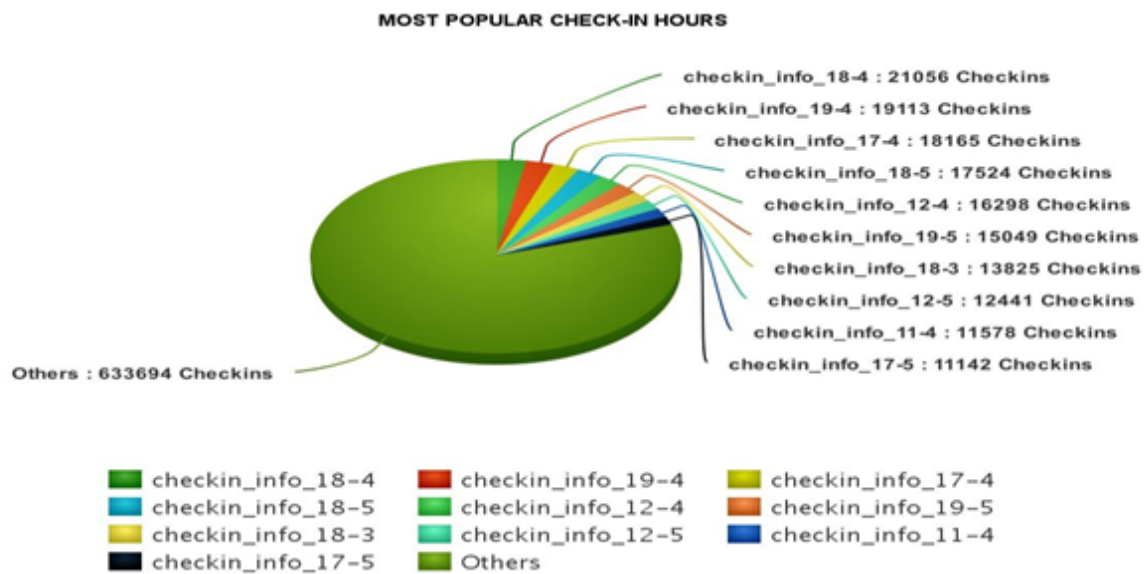


Fig 14 : Most Popular Checkin Hours

Total number of Check-ins : **789885**

Total number of Check-ins in 10 Most popular Check-in Hours : **156191**

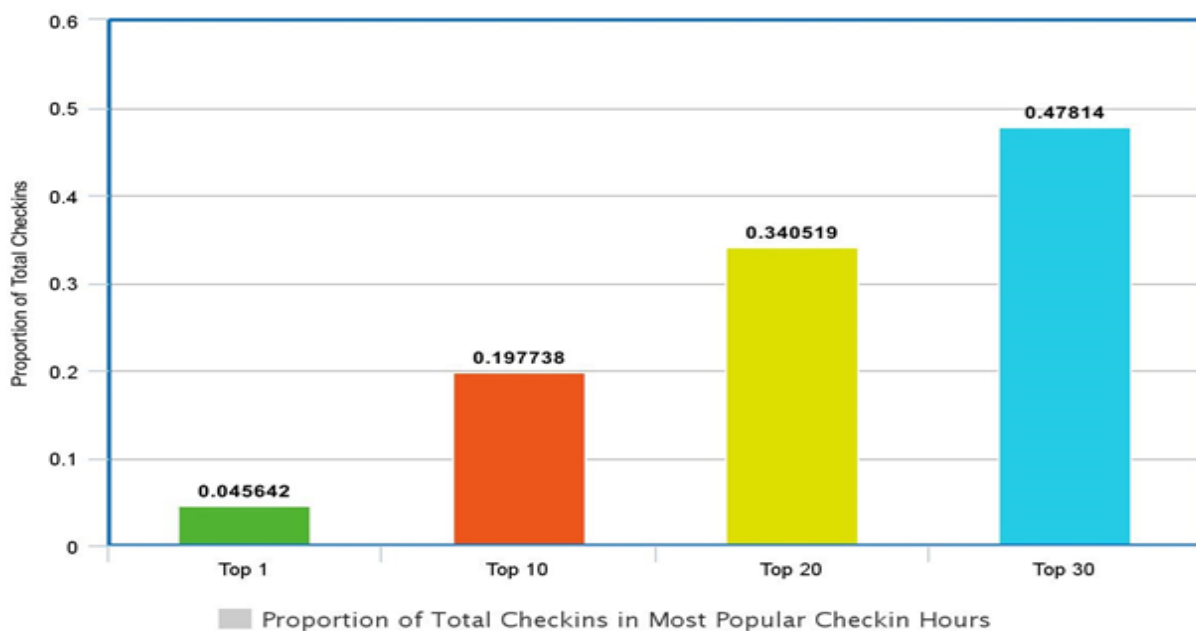


Fig 15 : Proportion of Total Checkins in Most Popular Checkin Hours

So, around **19.8 %** of the total Check-ins were registered in the 10 Most Popular Check-in Hours.

4.1.4 Least Popular Check-in Hours

Next, we analyzed the least popular Check-in Hours. To plot the least popular check-in hours, following script was used:

```
%%R
```

```
unranked_total_checkin_times <- colSums(raw_checkin_times)
```

```
plot(Total_Checkin_Times, main= "Ranked Plot")
```

```
plot.ts(cbind(unranked_total_checkin_times[15:84]), main= "Partial Time series Plot")
```

```
cat("10 Least popular check in hours: n");print(total_checkin_times[158:168])
```

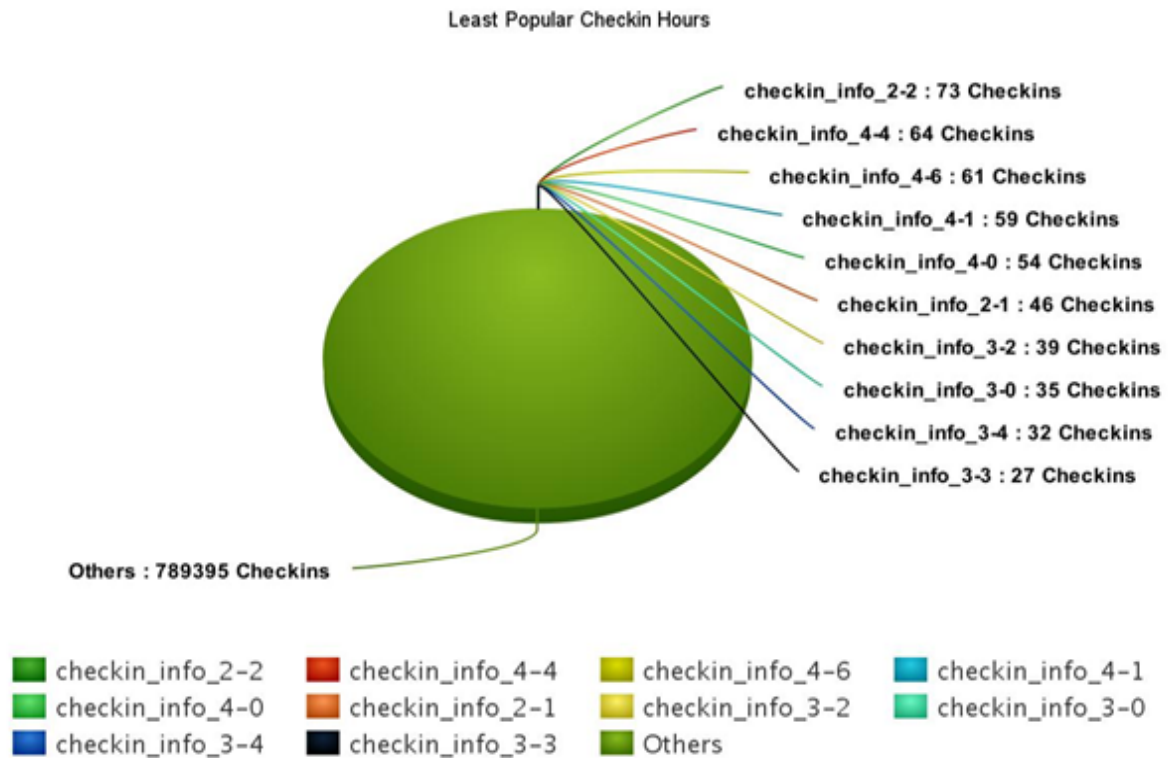
For the Least Popular Check-in Hours, I have got the following results :

No.	Check-in Time	Number of Check-ins
1	checkin_info_2.2	73
2	checkin_info_4.4	64
3	checkin_info_4.6	61
4	checkin_info_4.1	59
5	checkin_info_4.0	54
6	checkin_info_2.1	46
7	checkin_info_3.2	39

No.	Check-in Time	Number of Check-ins
8	checkin_info_3.0	35
9	checkin_info_3.4	32
10	checkin_info_3.3	27

Table 3 : Least Popular Checkin Hours

Based on the above data, we can plot the Least popular Check-in Hours as below :



meta-chart.com

Fig 16 : Least Popular Checkin Hours

The Quartile distribution of Check-ins in Most and Least Popular Check-in Hours is as given below :

Popularity	Number of Checkins
Minimum	27
1st Quartile	610
Median	3278
Mean	4874
3rd Quartile	8313
Maximum	21056

Table 4 :Quartile Distribution of Checkins

4.1.5 Analysis of Check-in Hours

- The most popular check in hour is **checkin_info_18.4**, corresponding to Friday at 6 pm. There are **21056** check-ins during this time, which take up nearly 5% of all the check-ins during the week. Nearly six times the median of people going out in that particular hour compared to the rest of the week's hours. The restaurants can focus more on providing better facilities and food services so as to generate more revenue in these busy hours.
- **Friday** is by far the most popular day of the week that people check in on. Four out of the top five check in hours occur on Friday, and these four hours on Friday (4,5,6, and noon) consist of nearly **10%** of all check ins.
- Examining the least popular check in hours, they seem more correlated to time of the day rather than day of the week. It seems 3 am overall is the least popular check in time, with a weekend day even showing up as an unpopular time.
- Minimum number of checkins are 27, in **checkin_info_3.3** hour.

5. Data Validation

The final step in our Data Analysis process is to validate the data. We need to see if we can predict the Stars rating correctly based on the given Checkin Times, which was our Hypothesis. We need to divide the data into two parts , 50% -50%, at random. We will try to predict the tsar rating for a given checkin Time from both, Training data set as well as Test data set to check whether our hypothesis is True or not.

To perform the data validation, I divided the data into two sets at random:

- Training Set
- Testing Set

The 50 % data is the Training Data set and the other 50 % is the Test Data set.

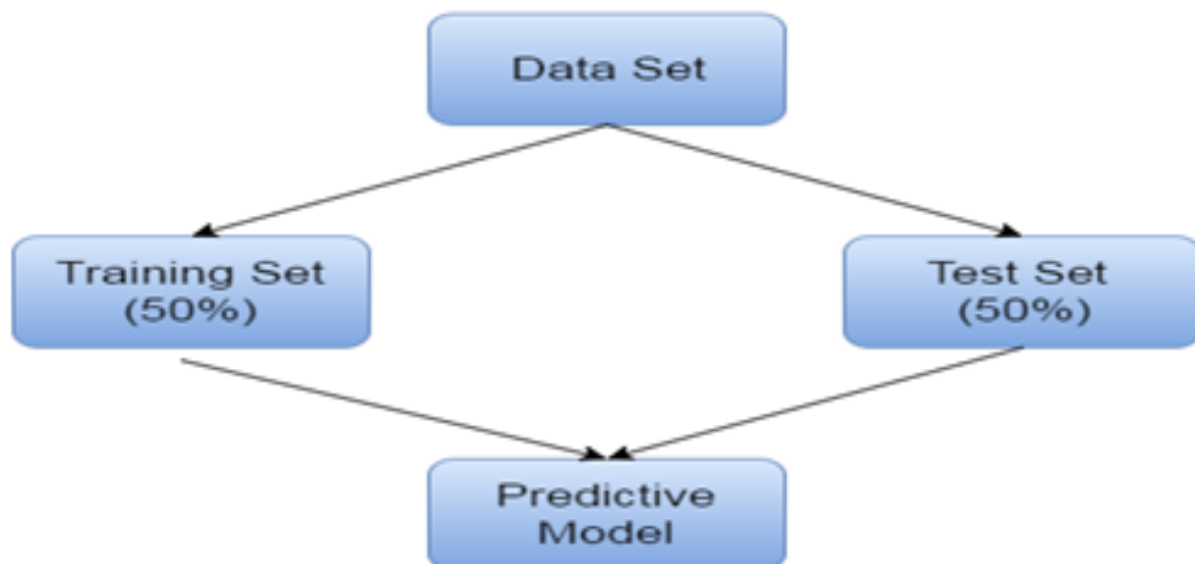


Fig 17 :Building a Predictive Model

I have used the following script to divide the data set into Training data Set and Test data Set:

```
index -sample(nrow(data_review_restaurant), nrow(data_review_restaurant)*.5)
data_review_restaurant.train <- data_review_restaurant[index, ]
data_review_restaurant.test <- data_review_restaurant[-index, ]
cat("Training set: "); print(dim(data_review_restaurant.train))
cat("Testing set: "); print(dim(data_review_restaurant.test))
```

5.1 Building a Predictive Model using GLM

Based on the analysis I have done, I tried to build a Predictive model. Based on this model, I tried to predict the Star Ratings given by customers to restaurants for a given Check-in Hour. I have followed the following steps to prepare my predictive model.

5.1.1 Subsample the Training Set

The first step in building predictive model is to sub sample the training data set. This is executed by the following script :

```
train_checkin_indices <- sample(nrow(Checkin_Stars))
train_checkin <- Checkin_Stars[train_checkin_indices, ]
write.csv(train_checkin, file = "train_yelp_checkins_stars.csv")
```

5.1.2 Subsample the Testing Set

After subsampling the training data set, the next step is to sub-sample the testing data set. Following script was used to perform the subsampling :

```
test_checkin <- Checkin_Stars[-train_checkin_indices, ]
write.csv(test_checkin, file = "test_yelp_checkins_stars.csv")
```

5.1.3 Generalized Linear Model

The **Generalized Linear Model (GLM)** is a flexible generalization of ordinary linear regression that allows for response variables that have error distribution models other than a normal distribution. The GLM generalizes linear regression by allowing the linear model to be related to the response variable via a *link function* and by allowing the magnitude of the variance of each measurement to be a function of its predicted value.

I have used the Generalized Linear Model to predict the relationship between the Check-in Hours and the Stars Rating. Our aim is to predict the Star rating based on a given Check-in Hour.

(Reference: http://www.sagepub.com/sites/default/files/upm-binaries/21121_Chapter_15.pdf)

The following Script was used to prepare the histogram results to study the relationship between Stars Rating and Check-in Hours :

```
no_rating <- train_checkin[,1:169]
checkin.glm <- glm(stars~.,data=no_rating,na.action=na.omit)

test_predictions <- predict(checkin.glm , test_checkin)
training_predictions <- predict(checkin.glm , train_checkin)

total_error <- abs(test_predictions - test_checkin$stars)
cat("Quartile summary of abs error");print(summary(total_error))
```

5.1.4 Predicting Stars for a given Check-in Hour

Based on the given Check-in Hours, we try to predict the Star Rating. Using the above scripts, I have got the following results :

Summary of abs error in Predicting Stars :

Minimum	0.000425
1st Quartile	0.146500
Median	0.450700
Mean	0.528600
3rd Quartile	0.764200
Maximum	2.433000

Table 5 : Error in Predicting Stars

On an average, we have an **absolute error = 0.5286067** in predicting Star Ratings for a given Check-in Hour.

5.1.5 Comparison between Training Data and Test Data Prediction

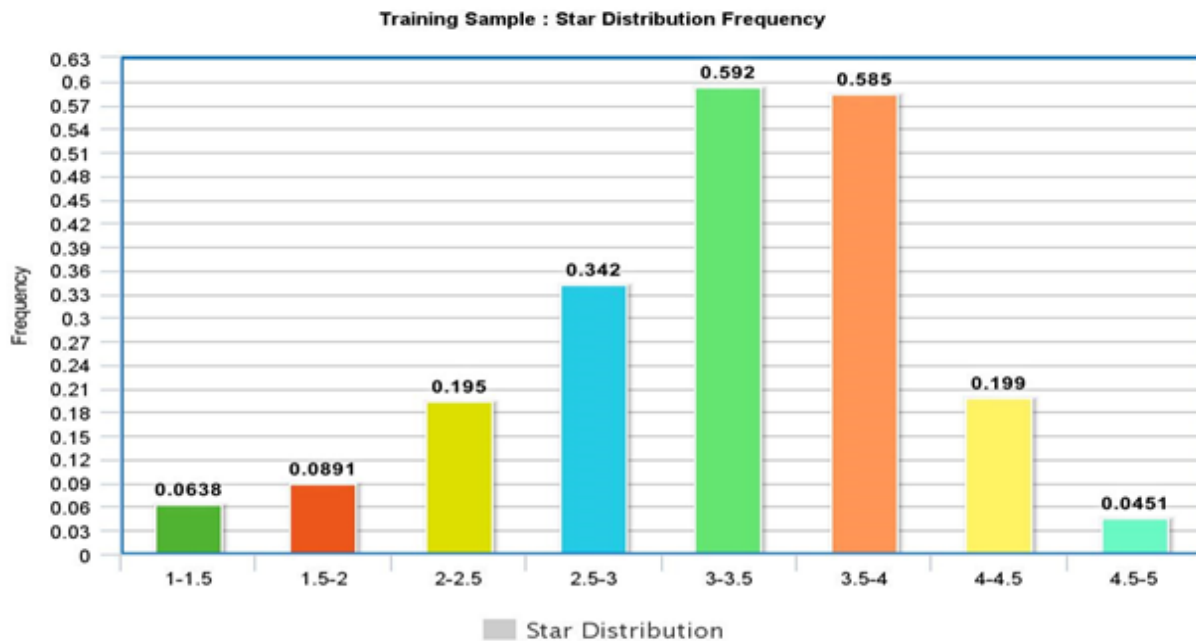


Fig 18 : Frequency of Star Distribution for Training data Sample

The corresponding Star Rating Prediction from the the **Test Data**, with **absolute error** of **0.5286067** is as shown below :

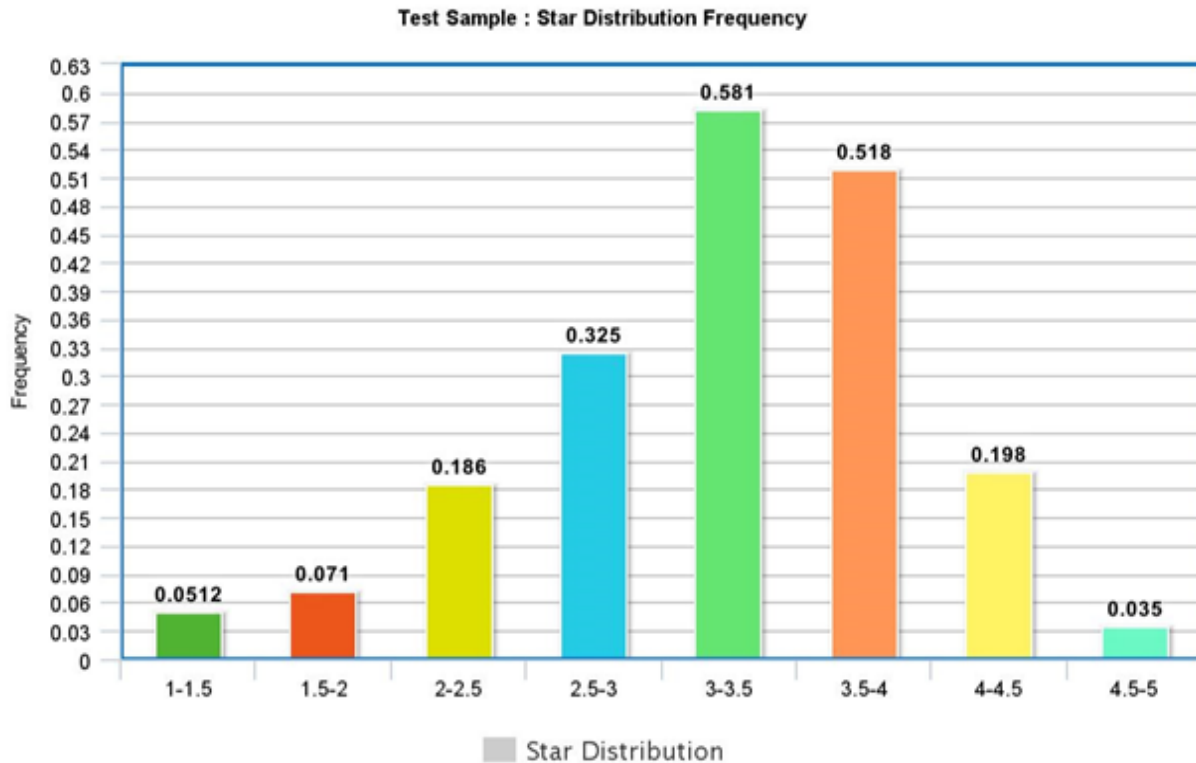


Fig 19 : Frequency of Star Distribution for Test data Sample

5.2 Performance of Generalized Linear Model

As explained above, I tried to predict the Star Ratings for a given Check-in Hour by using the Generalized linear model .On an average, we have an **absolute error = 0.5286067** in predicting Star Ratings for a given Check-in Hour. Also, the **standard deviation is only .70**.

The Generalized Linear Model really gives us average predictive power as a model for predicting star ratings.

5.2.1 Comparison of GLM with other Classifiers :

I have used the Generalized linear model for predicting Stars Rating based on Check-in Hours. I also try to run LDA mode for the same, but it did not give lower absolute error than the Generalized Linear Model. When I tried the feature selection tool, it led to a different set of features every time; meaning even if a star classifier could be built, it would not be robust and lead to different results. Based on these issues, I believe that there is not enough information / check-in info to make accurate predictions of star category. I reduced the original star classification problem to a binary classification problem where we try to predict whether a restaurant is "good" or "bad", meaning that it falls into the bottom or top quartile based on check in info.

```
data_checkin_stars$stars <- NULL
```

```
drop_indices <- data_checkin_stars$quant_rating=="medium"
```

```
binary_dataset <- data_checkin_stars[!drop_indices,]
```

I tried to project the data onto a lower dimensional subspace using PCA in order to visually examine the data. As the graphs shown below, there is very little information we can use to distinguish in between "good" and "bad" restaurants, given check-in info.

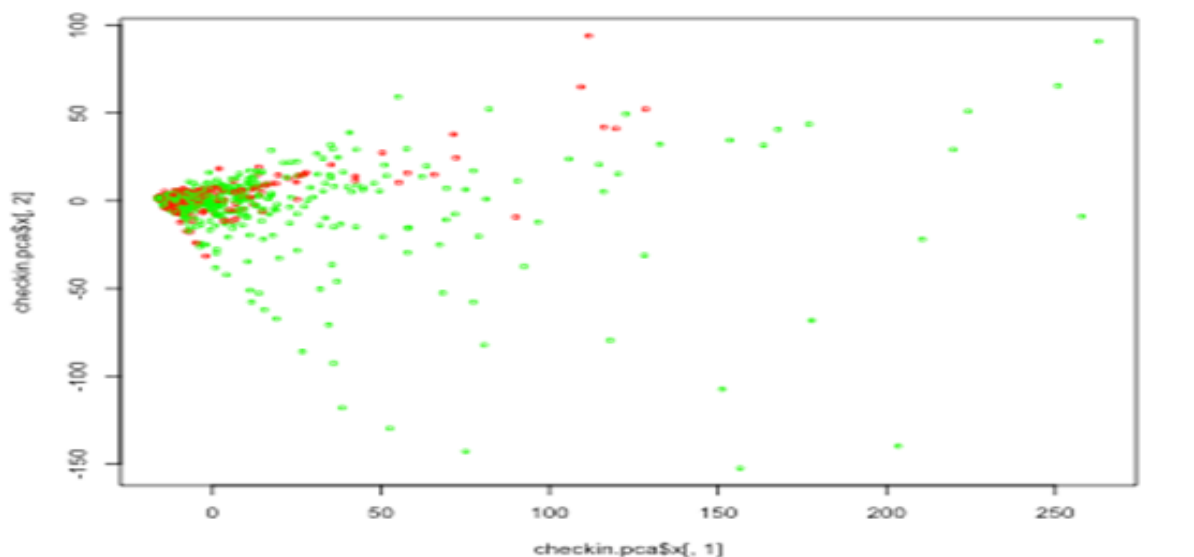


Fig 20 :Building a Predictive Model

6. Conclusion

Based on the above analysis, I have come to the following conclusions :

- We can predict Star Ratings for a given Checkin Time with the absolute error within a factor of **0.5411736** on average using check-in info.
- About 10 % of the all Checkins in the Restaurants are on Fridays, between 5 pm to 7 pm.
- Restaurant star reviews are distributed $N\sim(3.46,.73)$, but with slightly heavier tails than the normal distribution would explain. This is not too big of a problem since we are restricted in our star rating domain in between 1-5 stars and in our distribution over 99% of the points fall within these boundaries.
- The most popular check in hour is **checkin_info_18.4**, corresponding to Friday at 6 pm.
- Minimum number of checkins are 27, in **checkin_info_3.3** hour.
- People tend to rate restaurants higher than they are "supposed to be rated"