**Yelp Data Challenge 2016**

Real People Real Reviews

By: Bhatkal, Apurva

Dalal, Vivek

Kallianpur, Vinit

Under the guidance of Prof. Shrikant Krishnamurthi

;

**Contents**

[**INTRODUCTION 4**](#_Toc469689728)

[**Data Format for YELP JSON FILES 5**](#_Toc469689729)

[**1. Business 5**](#_Toc469689730)

[**2. Review 5**](#_Toc469689731)

[**3. User 6**](#_Toc469689732)

[**4. Check-in 6**](#_Toc469689733)

[**5. Tip 6**](#_Toc469689734)

[**Data wrangling and cleaning 7**](#_Toc469689735)

[**Reading data from JSON to csv format 7**](#_Toc469689736)

[**R code with select queries for obtaining data only for Yelp restaurants: 8**](#_Toc469689737)

[**R code for shuffling and joining the dataset 9**](#_Toc469689738)

[**R code for merging all cities: 10**](#_Toc469689739)

[**UI Implementation and deployment for the Yelp dataset 11**](#_Toc469689740)

[**Exploratory Data Analysis 12**](#_Toc469689741)

[**Modeling in Azure: 13**](#_Toc469689742)

[**Recommendation Systems: 13**](#_Toc469689743)

[**DATA: 13**](#_Toc469689744)

[**MODEL: 14**](#_Toc469689745)

[**RESULTS: 15**](#_Toc469689746)

[**PREDICTION RATINGS: 16**](#_Toc469689747)

[**TESTING: 18**](#_Toc469689748)

[**DEPLOY AS WEBSERVICE: 19**](#_Toc469689749)

[**Live Text Analysis on Reviews 21**](#_Toc469689750)

[**Introduction: 21**](#_Toc469689751)

[**Step 1: Clean and preprocess text dataset 21**](#_Toc469689752)

[**Step 2 : Extract numeric feature vectors from pre-processed text 22**](#_Toc469689753)

[**Step 3: Train classification or regression model 24**](#_Toc469689754)

[**Step 4: Score and validate the model 25**](#_Toc469689755)

[**Step 5: Deploy the model to production 26**](#_Toc469689756)

[**Fetching Live Data: 26**](#_Toc469689757)

[**Scraping the Data from Yelp website 26**](#_Toc469689758)

[**Review Predictions 28**](#_Toc469689759)

# **INTRODUCTION**

Yelp releases a data challenge every year for improving its functionalities’. It released its 8th challenge in 2016 which is due on December 31st 2016. The project has utilized this dataset which has the data in JSON format. We have extracted the data from JSON to csv files and then used the data for modeling purposes.

Yelp provides user, business data for check in of users and the tip information along with user check in information.

Road map used for loading data to the DB and utilizing it to make informed decisions:

1. Convert the data from JSON format to csv files
2. Data cleaning, wrangling and exploratory analysis
3. Model Building in Azure
4. Application deployment using Flask on AWS EC2 instance

The data is provided in JSON format for 6 entities. We have faced -issues in dealing with such huge data so we have confined the scope of this project to only the states within the US in place of US, Canada, Germany and UK.

## **Data Format for YELP JSON FILES**

### **Business**

{

'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),

...

},

}

### **Review**

{

'type': 'review',

'business\_id': (encrypted business id),

'user\_id': (encrypted user id),

'stars': (star rating, rounded to half-stars),

'text': (review text),

'date': (date, formatted like '2012-03-14'),

'votes': {(vote type): (count)},

}

### **User**

{

'type': 'user',

'user\_id': (encrypted user id),

'name': (first name),

'review\_count': (review count),

'average\_stars': (floating point average, like 4.31),

'votes': {(vote type): (count)},

'friends': [(friend user\_ids)],

'elite': [(years\_elite)],

'yelping\_since': (date, formatted like '2012-03'),

'compliments': {

(compliment\_type): (num\_compliments\_of\_this\_type),

...

},

'fans': (num\_fans),

}

### **Check-in**

{

'type': 'checkin',

'business\_id': (encrypted business id),

'checkin\_info': {

'0-0': (number of checkins from 00:00 to 01:00 on all Sundays),

'1-0': (number of checkins from 01:00 to 02:00 on all Sundays),

'14-4': (number of checkins from 14:00 to 15:00 on all Thursdays),

...

'23-6': (number of checkins from 23:00 to 00:00 on all Saturdays)

}, # if there was no checkin for a hour-day block it will not be in the dict

}

### **Tip**

{

'type': 'tip',

'text': (tip text),

'business\_id': (encrypted business id),

'user\_id': (encrypted user id),

'date': (date, formatted like '2012-03-14'),

'likes': (count),

}

## **Data wrangling and cleaning**

1. The review file has around 1.8 GB of data; this made loading the data difficult. We took a subset of the dataset to load the data on the local machines for exploratory analysis.
2. The JSON files has levels of nesting that had to be extracted appropriately to ensure that no data is lost.
3. This review table was inner joined with the other tables User, Review and business using business id for business and review table and user id for user and review table.
4. We call it the URB subset (User, Review and Business)
5. The dataset provided by Yelp contains data about various business’s like Automobile, Doctors, etc. apart from Restaurants. We have filtered the data based on the category of Restaurants and reduced the number of entries to 26k records in the business dataset.
6. There are 1017 categories of Businesses in Yelp and 45% of them comprise of Yelp Restaurants

## **Reading data from JSON to csv format**

data<-fromJSON(file = /PATH FOR THE FILE/ ,method="C")

data<-lapply(data,function(x)

{

  x[sapply(x,is.null)]<-NA

  unlist(x)

})

do.call("rbind",data)

data<-ldply(data,rbind)

write.csv(data,file='Business.csv')

## **R code with select queries for obtaining data only for Yelp restaurants:**

# read the json files directly in R for exploratory analysis

library(jsonlite)

# for nested json files

yelp <- stream\_in(file("yelp\_academic\_dataset\_business.json"))

# converting the dataframe for the 7 days which are data frames

yelp\_flat <- flatten(yelp)

# only displays 10 values

library(tibble)

yelp\_tbl\_display10 <- as\_data\_frame(yelp\_flat)

# remove hours of operation

# for %>%

require(magrittr)

install.packages('magrittr')

# for select

require(dplyr)

yelp\_tbl\_display10 %>% select(-starts\_with("hours"), -starts\_with("attribute"))

# get all the resturants from the category data

library(stringr)

number\_restaurants\_1<-yelp\_tbl\_display10 %>% select(-starts\_with("hours"), -starts\_with("attribute")) %>%

filter(str\_detect(categories, "Restaurant"))

# make a row for each category

library(tidyr)

restaurant\_cat<-yelp\_tbl\_display10 %>% select(-starts\_with("hours"), -starts\_with("attribute")) %>%

filter(str\_detect(categories, "Restaurant")) %>%

unnest(categories) %>%

select(name, categories)

# get rid of the others apart restaurant like tyre, vets, etc.

restaurant\_rid<-yelp\_tbl\_display10 %>% select(-starts\_with("hours"), -starts\_with("attribute")) %>%

filter(str\_detect(categories, "Restaurant")) %>%

unnest(categories) %>%

filter(categories != "Restaurants") %>%

count(categories) %>%

arrange(desc(n))

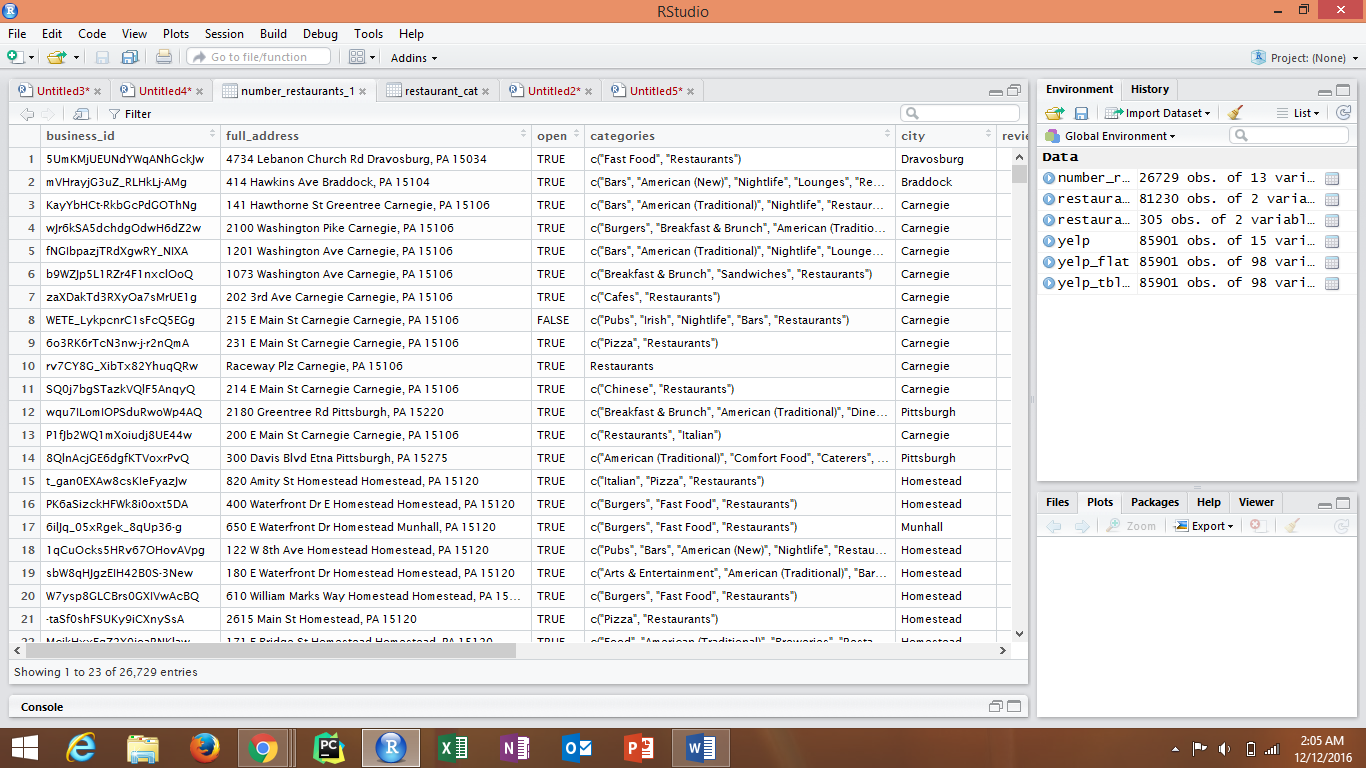


Figure: Dataset with information only about Restaurants and no other business’s

## **R code for shuffling and joining the dataset**

# 1. review

review<-read.csv(file = 'review.csv')

# as review is too huge, the data set is divided further into 5 subsets

shuffle<-runif(nrow(review))

review\_1<-review[order(shuffle),]

review\_part1<-review\_1[1:20000,]

write.csv(review\_part1, file = "review\_small.csv")

inner\_join<-merge(x = review\_small, y = business\_for\_modeling, by = "business\_id")

1. We have filtered out data from the business file for columns where majority; 90% of the data is missing; NA values exist. This is done by getting a summary of the dataset.

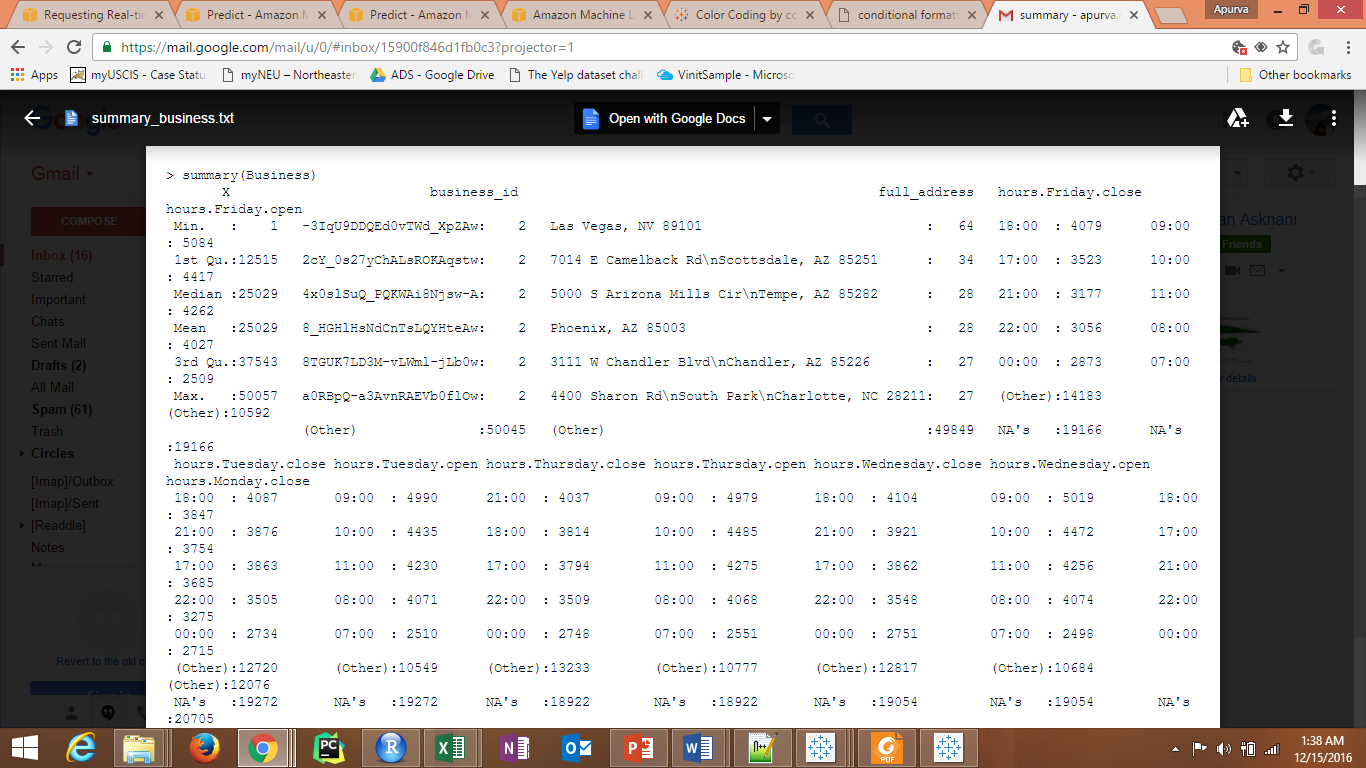


Figure: The summary describes the column hours.Friday.close has 19166 NA values which is majority of the data.

1. We have also used Apache drill to aid the data cleaning process. Apache drill allows users to analyze the data without any schema definition or ETL transformations enabling a user to get preliminary insights on data with SQL like querying.

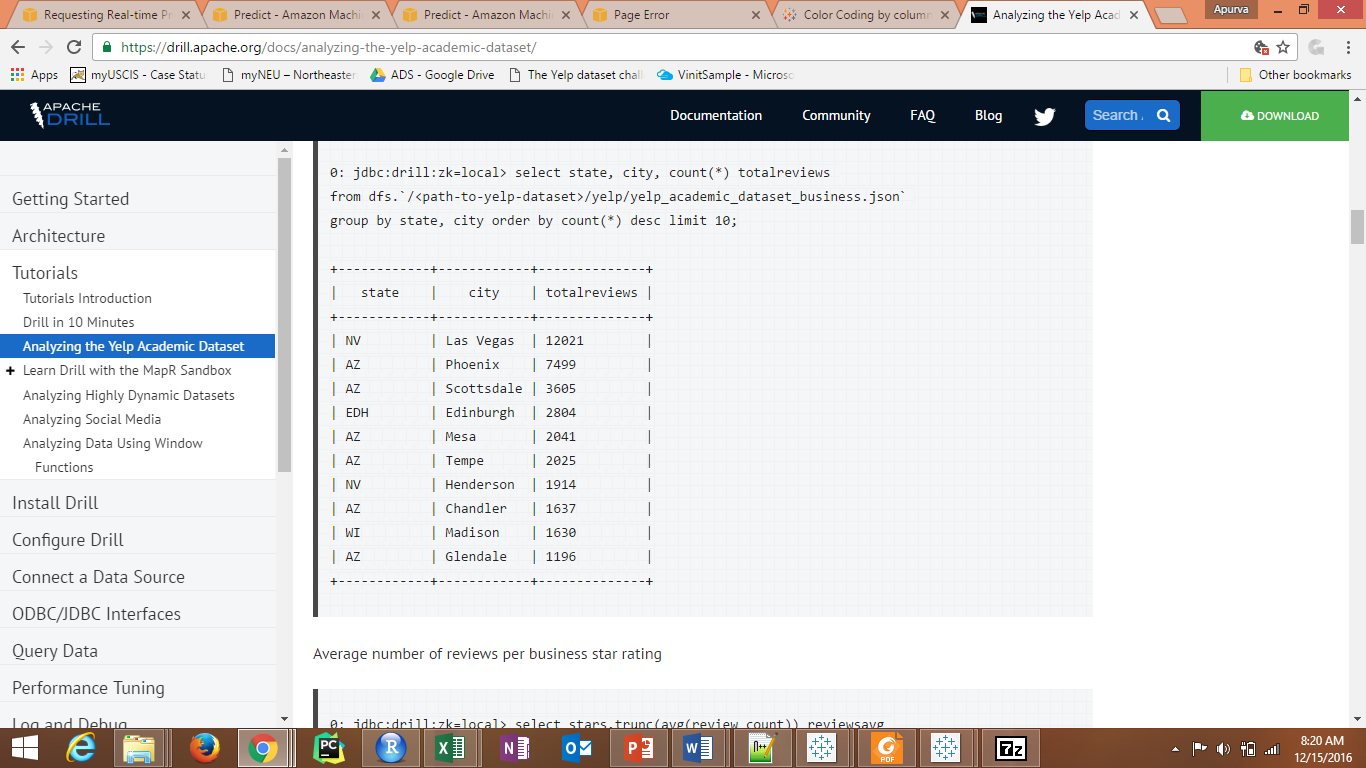


Figure: We get the total reviews for the state by city and state

1. The data for the cities have spelling errors and hence can be grouped to one city. For example: Cities like Las Vegas are also having entries as LV or LasVegas. We have merged the cities on those criteria’s too. The same has been manually checked for states too.

## **R code for merging all cities:**

if(df$city[i] =='Las Vegas' | df$city[i] =='LasVegas'){

value[i]<-'Lass Vegas'

}

# **Exploratory Data Analysis in Tableau**

1. The data provided by Yelp is in JSON format which is nested. Loading such a huge dataset and joining data in Tableau wasn’t isn’t the right approach.
2. If we load the data that was cleaned in CSV format, we do not get a holistic view as we have dropped a lot of unwanted columns in the data cleaning step for modeling purposes.
3. We initially connected to Tableau using Apache Drill to get an initial overview about data. Drill could read the JSON data but we were not able to join all the data as the size of the data is too huge.
4. Later, as the complexity of the JSON nesting increased with the join operations, hence we moved from using Tableau 10.01 to tableau 10.1 as it supports JSON format.
5. Tableau does not allow joining of JSON files, hence we took an extract of the JSON files and then joined these extracts. This approach took lesser time to join data and build the graphs.

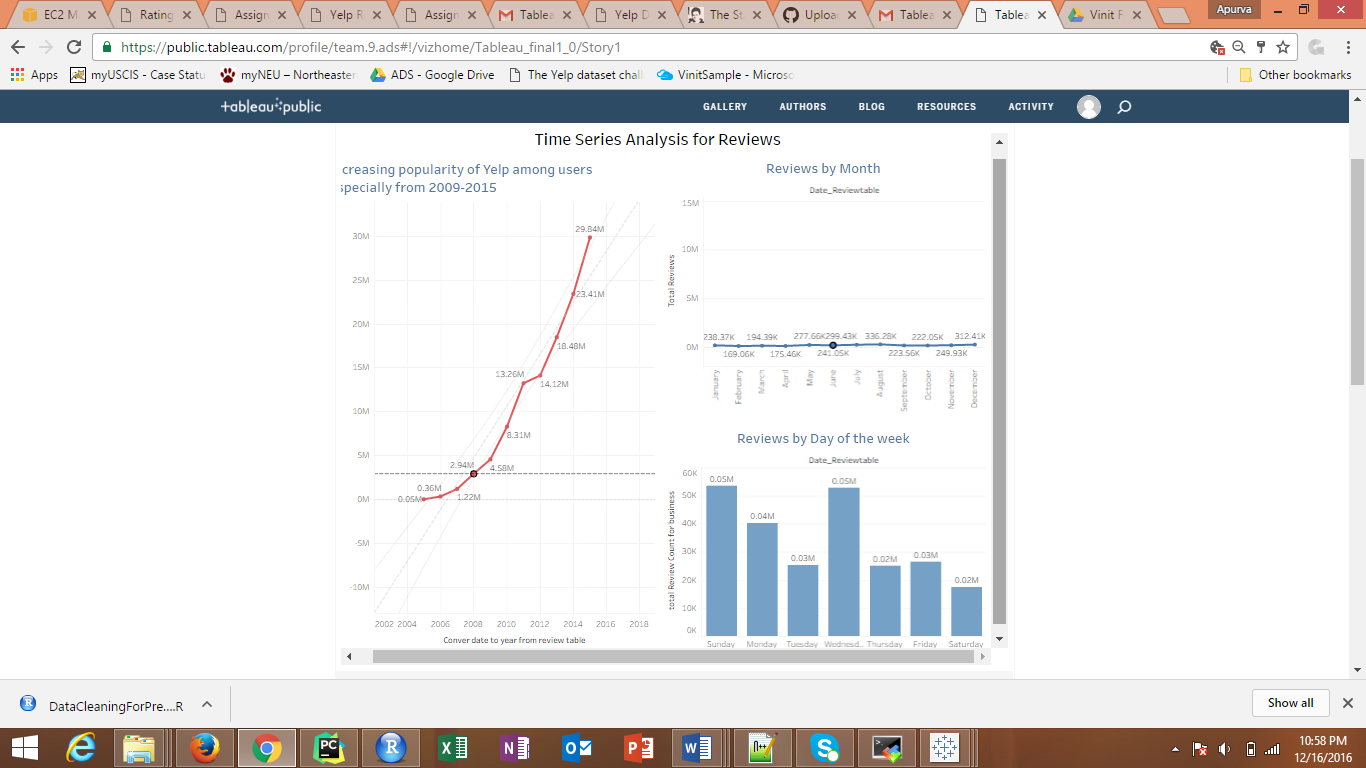


Figure: Time Series Analysis for Review

Here, the first graph shows the increasing trend amongst users. This can be said as the number of reviews is increasing year over year. The other 2 graphs show the trends by Month and by Day. These trends show that the number of reviews increase over the summer period and people tend to write reviews on Sundays and Wednesdays.



Figure: Categorization of Businesses that deal with food on Yelp

Yelp has businesses apart from restaurants like doctors, automobile companies, etc. The data provided by Yelp shows that around 45% of the categories are related to Food and Restaurants.



Figure: The top 10 best Restaurants for the cities with most restaurants

The graph portrays that Las Vegas has the most restaurants and gives each city with the top 10 restaurants for the selected city based on the criteria’s selected.

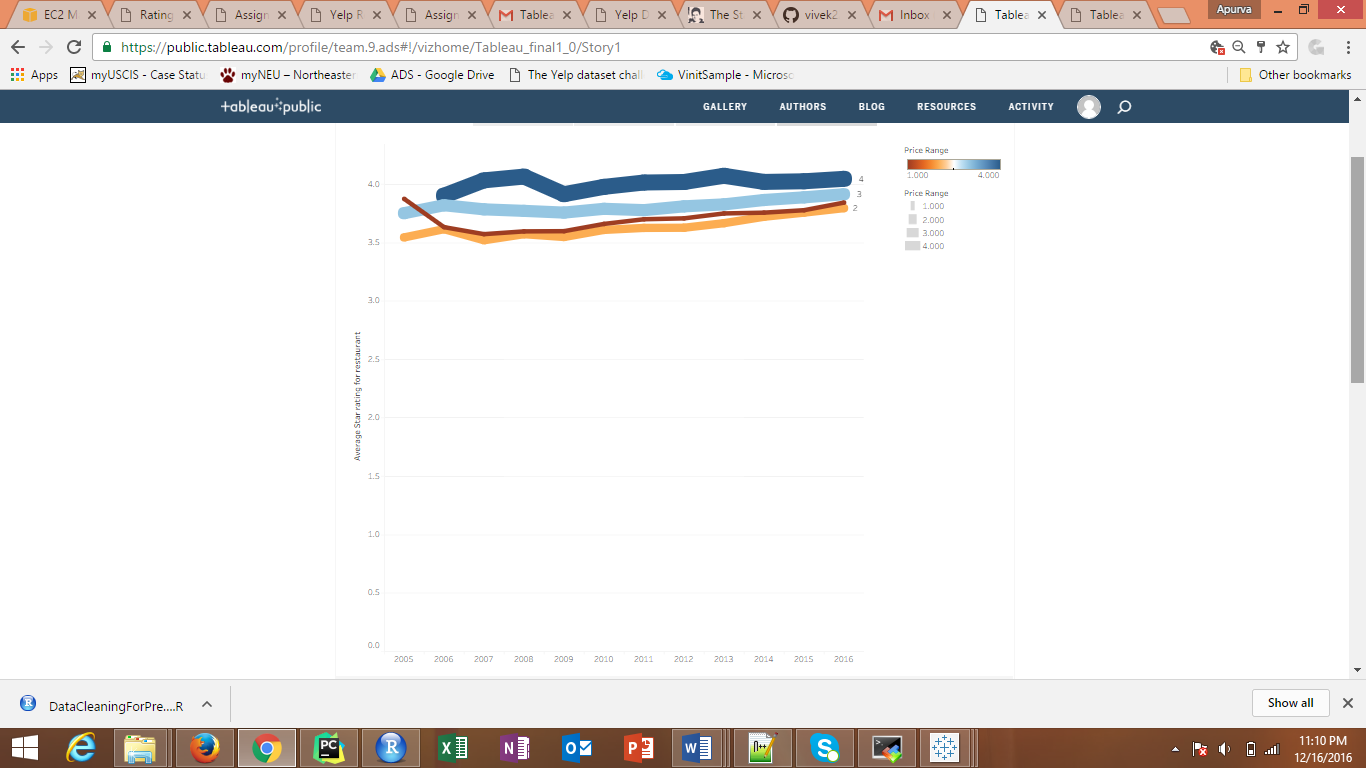


Figure: Represents the star rating of the restaurant and the value you get for the money you pay at the restaurant

The thickness of the graph gives the price range of the restaurant and the 4 colors give the cost. It shows that there are hotels where you pay less but the food is better.

The other charts are explained and posted on the tableau public link as below:

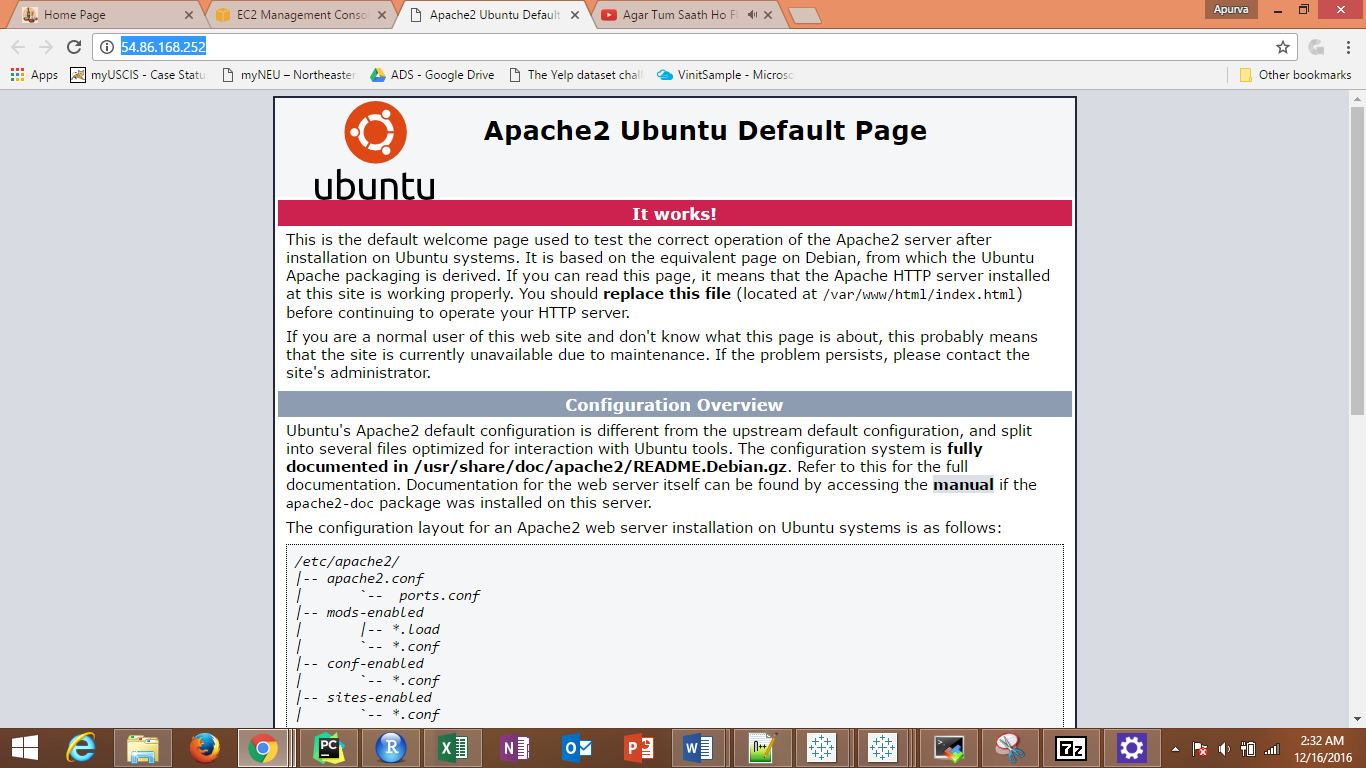
<https://public.tableau.com/profile/team.9.ads#!/vizhome/shared/W6T4WXGD9>

# **UI Implementation and deployment for the Yelp dataset**

Our link to the application is :

<http://54.85.250.226/>

1. We have launched an EC2 instance on an Ubuntu machine on AWS. This instance is not a Community version and is launched from scratch by loading Apache on the Ubuntu image.
2. We have created a flask application that consumes the response from the web service that is built on Azure using the Request Response Service provided by AWS.



1. Our flask application needs to be deployed on to Apache on Ubuntu. For this, we load python and flask on Apache.

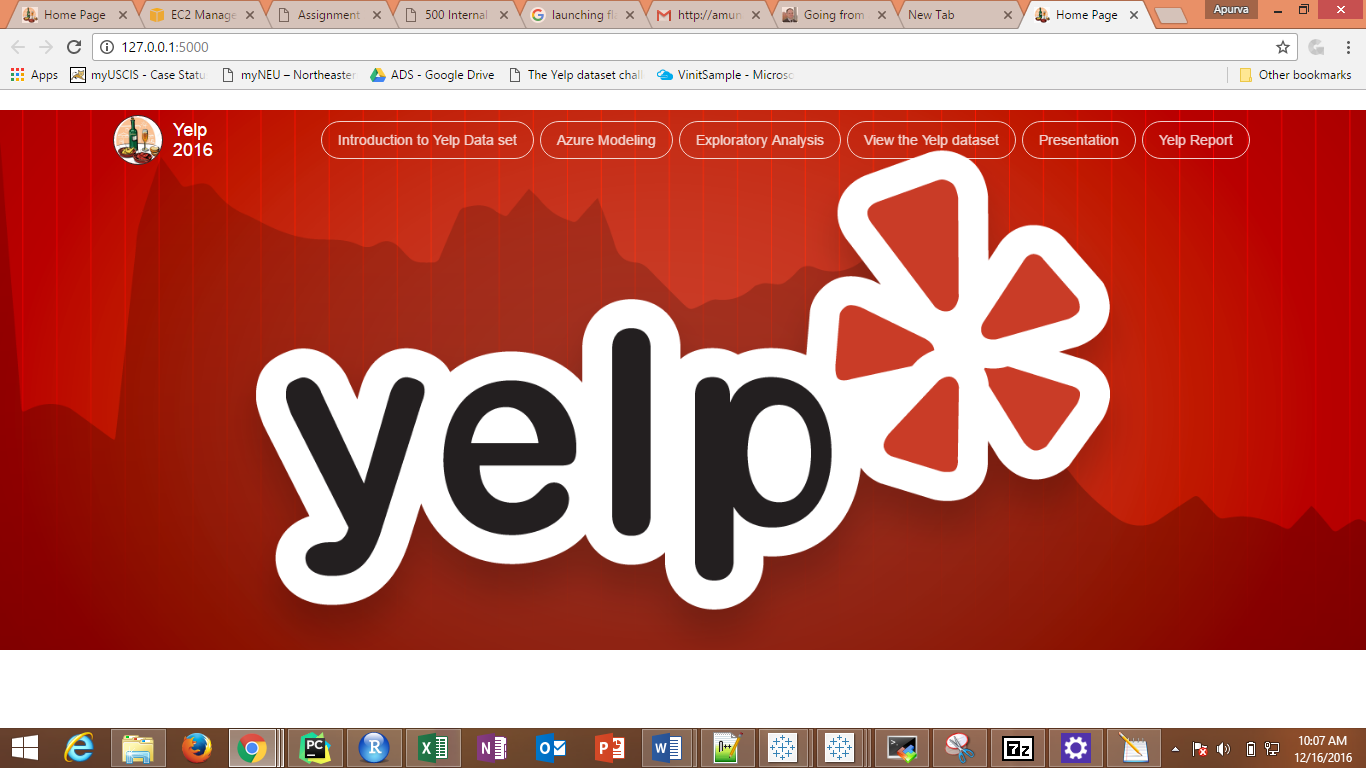


Figure: Screenshot for our Yelp project

1. Once Apache is loaded, we deploy our project folder by SSH ing into the Ubuntu machine. The project needs to be built with the right directory structure on the fresh EC2 instance else there will be issues faced with the deployment.

# **Modeling in Azure:**

## Recommendation Systems:

Login credentials on AzureML Studio:

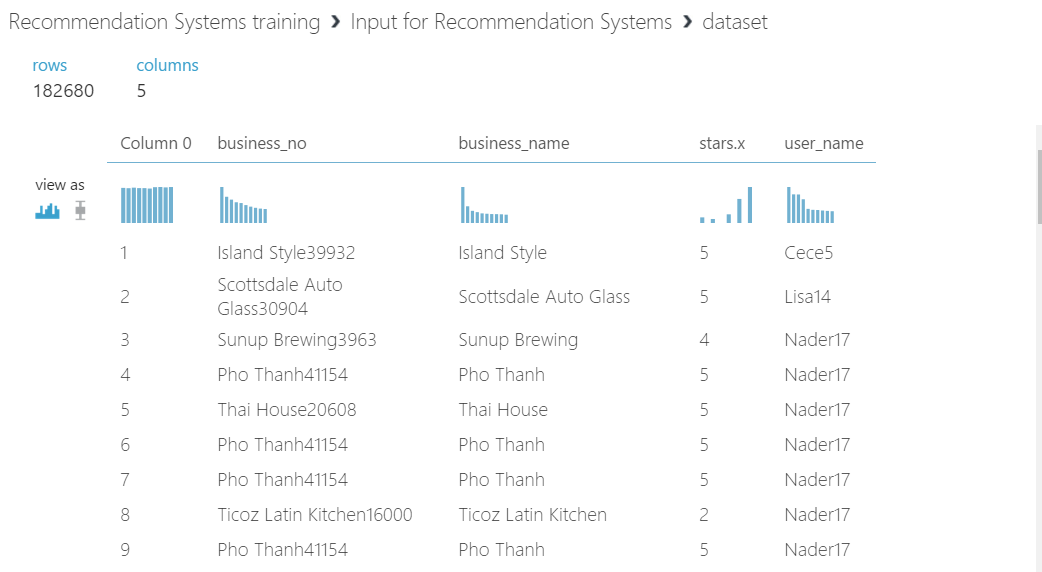
UserId: [adsteam9project2@gmail.com](mailto:adsteam9project2@gmail.com)

Password: polop@123

This engine demonstrates the use of the Matchbox recommender modules to train a business recommender engine. We use a pure collaborative filtering approach: the model learns from a set of users who have all rated a subset of a catalog of business on Yelp. Matrix factorization allows us to infer from this latent user preferences and his business visit traits. These preferences and traits are then used to predict what rating a user will give to an unvisited business, so that we can recommend businesses that the user is most likely to visit.

### Data:

Data is approximately 180,000 records with 25000 records of users and 52000 records of businesses extracted from Yelp dataset. Each user is giving tips, stars(ratings) for businesses and they are unique.

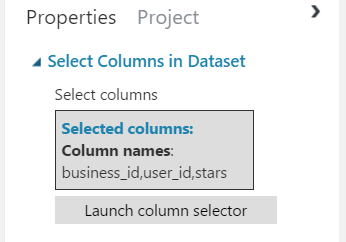


### Model:

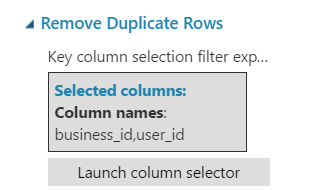
First, we need to prepare the data for use with the **Train Recommender** module. It requires triplets in this format: <user\_id, business\_id, stars>.

The **Train Recommender** module is more lenient with respect to the user and business identifiers. One would typically use integer IDs for these too (and use those as keys for names, tips, reviews and other metadata in the presentation layer), but to make our results easier to work with this.

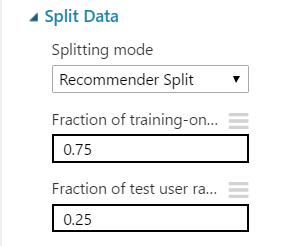
The **Train Recommender** module requires that the input contain three fields used for training, so we use **Project Columns** to select only the **user\_id**, **business\_id**, and **stars fields**.



This dataset contains a few conflicting ratings for the same user\_id, business\_id pair. Thus introducing noise in our training and evaluation, so we remove the duplicates, arbitrarily retaining only the first occurrence of each user-movie pair we encounter.

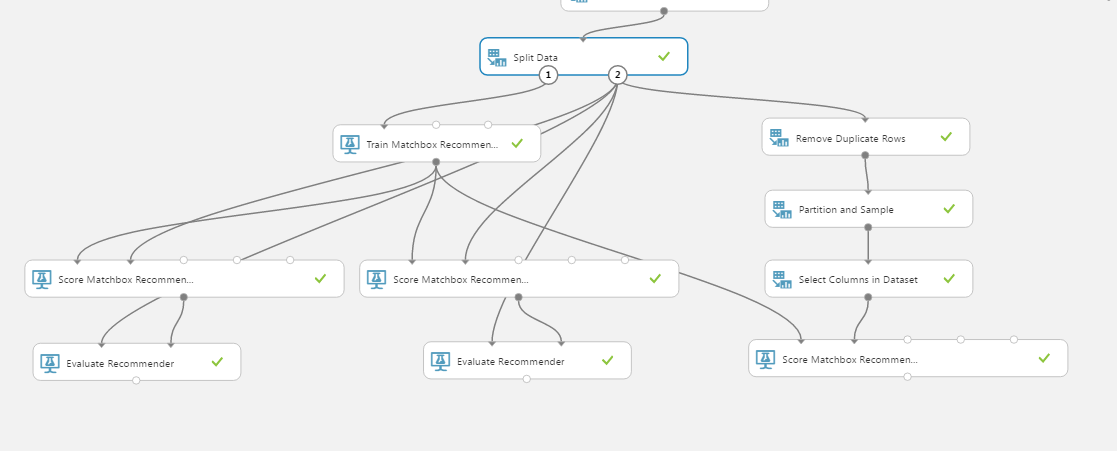


Split the dataset to train and test, using recommender split.



The **Train Recommender** module requires two parameters:

* Number of features: This determines the number of latent parameters that will be learned for each user and each item. (Technically, the recommender algorithm is based on factorizing the user-movie interaction matrix. This parameter determines the rank of the approximation.) More features make for a more powerful model, but risks over fitting the training data.
* Number of iterations: Model parameters are found by random initialization, followed by minimizing a residual error (difference between the true and predicted ratings for each user-movie pair) using an iterative gradient descent technique.



### Results:

In our model, we demonstrated 3 different ways to use the trained recommender model:

* To predict ratings.
* To predict top-*n* movies from a list already rated by each user
* 3. To make *n* recommendations from the full catalog for each user

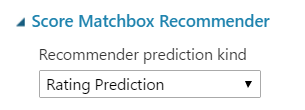
The first two methods are used simply to evaluate the performance of the learned model, while the last method represents a typical production use case. To perform all three different types of predictions, we used the **Score Recommender** module.

The module has two required and two optional inputs.

* The first required input is a trained model. In this case, we have directly connected the output of the trainer, but for production one would save the trained model and then connect this saved model to the scorer.
* The second input is a dataset to be scored. The format of this dataset will depend on the task, as we have described below.
* The two optional ports are for user and business metadata, like the optional inputs when training. We used thee inputs when training, and provided the same data when scoring.

### Prediction Rating:

Prediction is a straightforward task. We provided an input dataset for which we wanted to get scores, using the three-item tuple format used for training. The **Score Recommender** module used the trained model to predict a rating for each user-business pair, and outputs a tuple consisting of <user, business and predicted rating>.



To evaluate the accuracy of predictions, we used the **Evaluate Recommender** module. The first input is the testing dataset, containing tuples (business\_id-user\_id-stars) similar to those provided for training.

Evaluate Recommender module required two parameters:

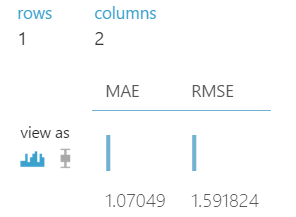
* Minimum number of businesses
* Minimum number of users

1. Prediction Ratings

By using these parameters, we limited the evaluation to users who have rated at least n items; and businesses that have been rated by at least m users, respectively.

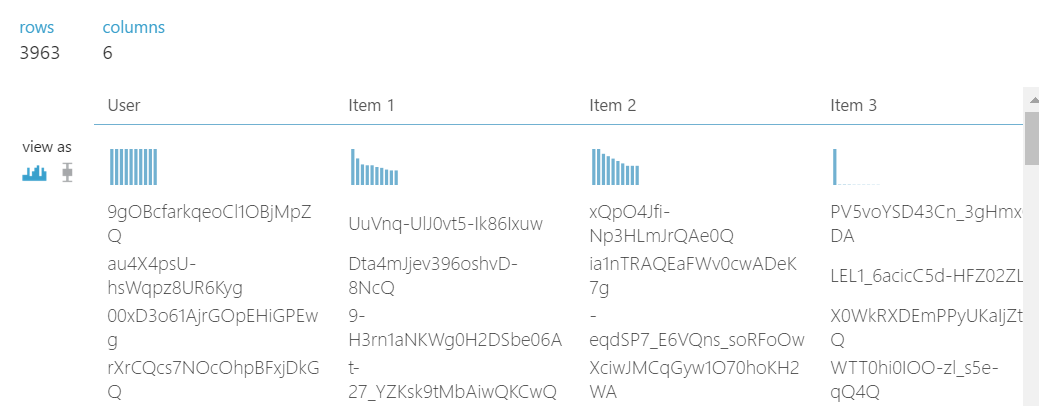
In this experiment, the second input contained the same set of tuples that we used earlier for training the model; therefore, evaluation will compare the predicted ratings with the actual ratings, using these two metrics:

* RMSE
* MAE



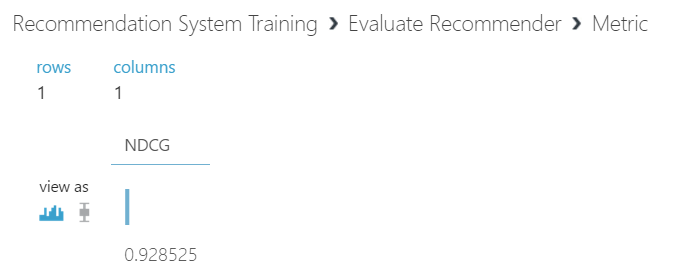
1. Recommend Businesses from Test

In this part of the experiment, we use the model to create a rank-ordered list of the top *n* Businesses for each user, but selecting *only* from movies that have already been rated. The input is the same Business-user-stars format that was used for training. The **Score Recommender** module will use the tuples to extract the set of users, and then for each user, create a set of movies to use in building the rank-ordered list.



For our final evaluation, we again used the **Evaluate Recommender** module. The first input is our test split of the data, but this time the scored dataset is recognized as a per-user recommendation

1. For our final evaluation, we again use the Evaluate Recommender module. The first input is our test split of the data, but this time the scored dataset is recognized as a per-user recommendation, so the module will calculate Normalized Discounted Cumulative Gain (NDCG) instead.

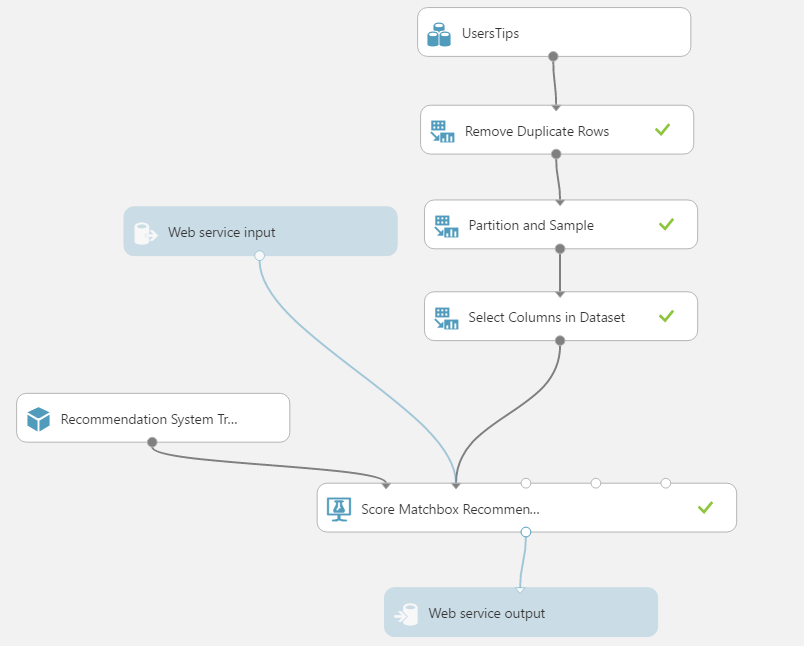


**SAVE TRAINED MODEL**



### TESTING:

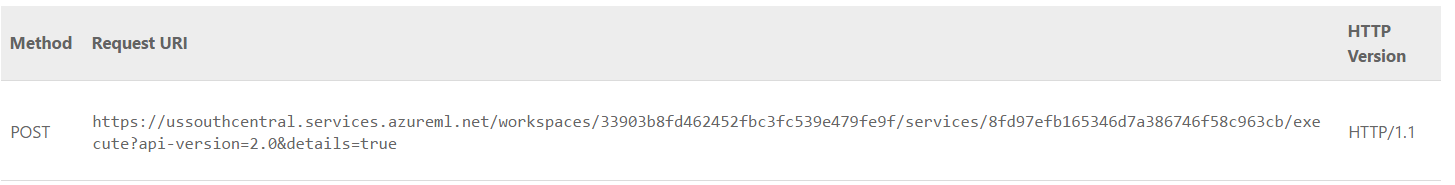
We then create a new experiment that has only the scoring module, and added the saved model. To specify the Web service entry and exit points, use the special Web Service modules. Note that the Web service input module is attached to the node where input data would enter the experiment.



### Deployment of Web Service:

After successfully running the experiment, it can be published by clicking **Publish Web Service** at the bottom of the experiment canvas.





Enter these on the Azure Portal and deploy it on the Azure Portal website and it will provide you with the link:

The steps to recreate this is been mentioned above.

<http://yelprecommendationystems.azurewebsites.net/>

## Live Text Analysis on Reviews

Login credentials on AzureML Studio:

UserId: [adsteam9project@gmail.com](mailto:adsteam9project@gmail.com)

Password: polop@123

### Introduction:

We used Azure Machine Learning to build and operationalize text analytics models. This model helped us solve, for example, document classification or sentiment analysis problems.

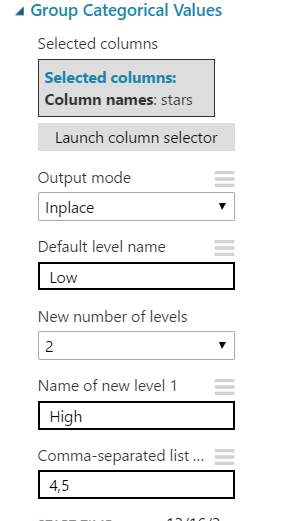
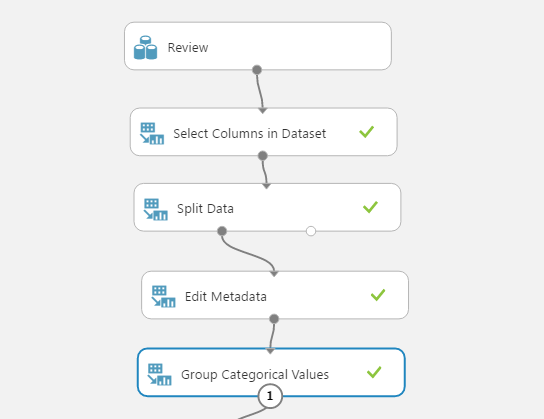
We followed the following steps to build our Model:

1. Clean and preprocess text dataset
2. Extract numeric feature vectors from pre-processed text
3. Train classification model
4. Score and validate the model
5. Deploy the model to production as a Web Service.
6. Scraping Data from the yelp website for Live predictions

We performed Text Analysis on the reviews table

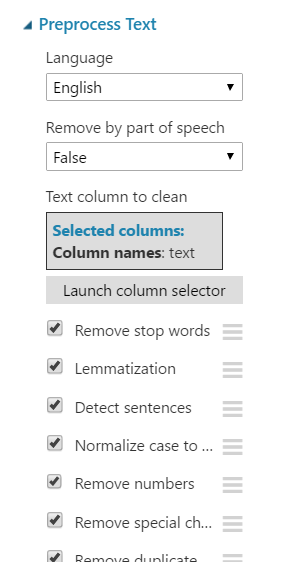
### Step 1: Clean and preprocess text dataset

We begin the experiment by dividing the review scores into categorical low and high buckets to formulate the problem as two-class classification. We use Edit Metadata and Group Categorical Values modules.



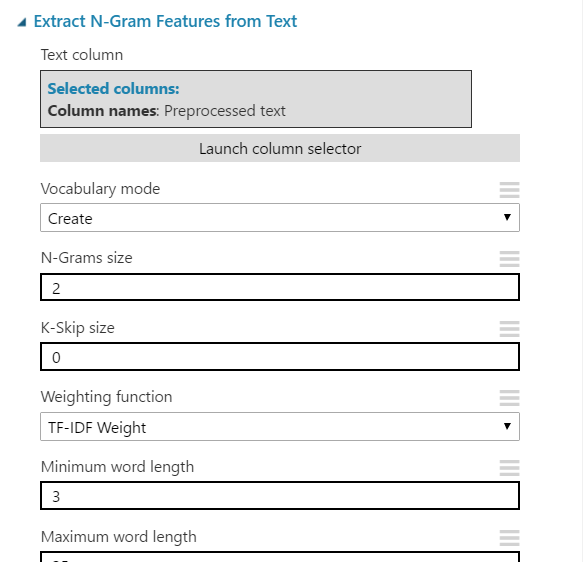
### Step 2 : Extract numeric feature vectors from pre-processed text

Then, we clean the text using Preprocess Text module. We also convert the text to lowercase, lemmatize the words, and detect sentence boundaries that are then indicated by "|||" symbol in pre-processed text.



We also customized the stop words.

To build a model for text data, we had to convert free-form text into numeric feature vectors. In this example, we used Extract N-Gram Features from Text module to transform the text data to such format. This module took a column of words and computed a dictionary of words, or N-grams of words, that appeared in our dataset. Then, it counts how many times each word, or N-gram, appears in each record, and creates feature vectors from those counts. We set N-gram size to 2, so our feature vectors include single words and combinations of two subsequent words.

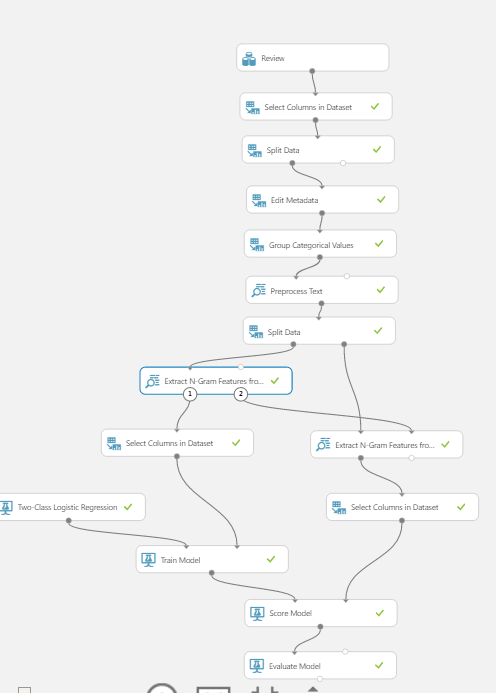


We applied TF\*IDF (Term Frequency Inverse Document Frequency) weighting to N-gram counts. This approach added weight of words that appeared frequently in a single record but are rare across the entire dataset. Other options include binary, TF, and graph weighing. But this gave us the best predictions.

### Step 3: Train classification or regression model

Now that the text has been transformed to numeric feature columns. The dataset still contains string columns from previous stages, so we use Select Columns in Dataset to exclude them.

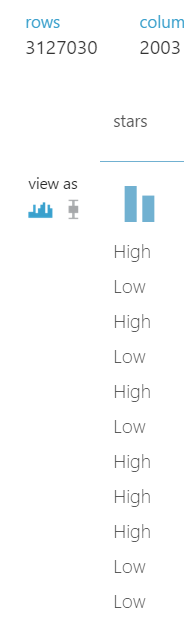
We then used Two-Class Logistic Regression to predict our target: high or low review score. We used the tools available in Azure Machine Learning to improve the model. For example, experimented with different classifiers to find out how accurate results they gave and used hyperparameter tuning to improve the accuracy.



### Step 4: Score and validate the model

We scored it against the test dataset and evaluated the accuracy. However, the model learned the vocabulary of N-grams and their weights from the training dataset.

We then used that vocabulary and those weights by extracting features from test data, as opposed to creating the vocabulary anew. Therefore, we added Extract N-Gram Features module to the scoring branch of the experiment, connected the output vocabulary from training branch.



### Step 5: Deploy the model to production

The model is almost ready to be deployed to production. When deployed as web service, it takes free-form text string as input, and return a prediction "high" or "low." It uses the learned N-gram vocabulary to transform the text to features, and trained logistic regression model to make a prediction from those features.

This is the core of the Modelling. Now the below mentioned steps are for fetching Live data and predicting it.

## Live Text Analysis

### Scraping the Data from Yelp website

1. The reviews for conducting this analysis were obtained from scrapping Yelp website for a restaurant.



Figure: Scrapping of data for a restaurant based on the user choice

Input

About 100 current reviews are extracted and is fed through the above program to the Sentimental Analysis Model for predicting the sentiment. The values are fed as below by web consumption. The values are fed by the web consumption.

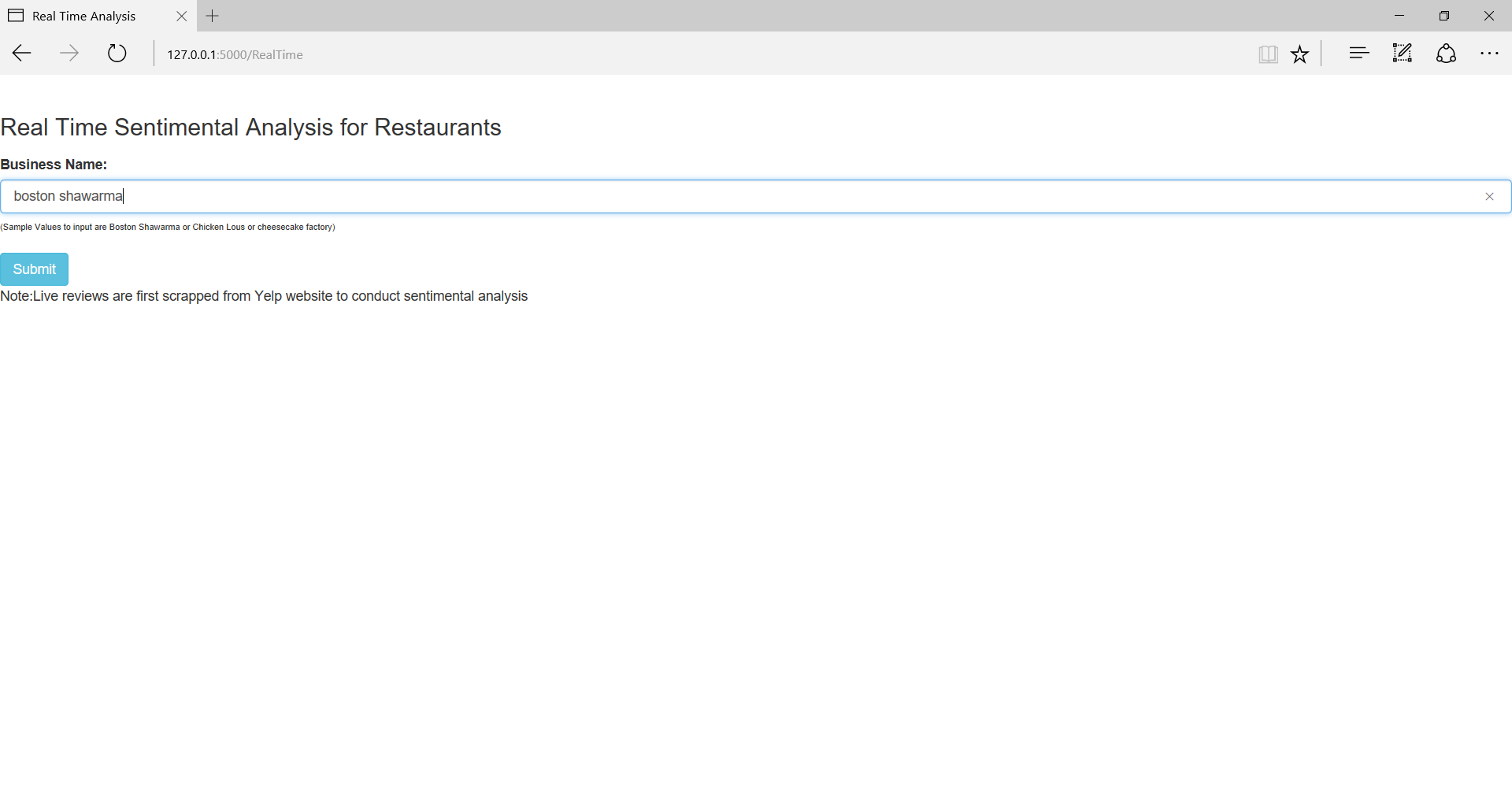


Figure: User Input for restaurant

Output

The output that is generated predicts the sentiment for the given restaurant choice.

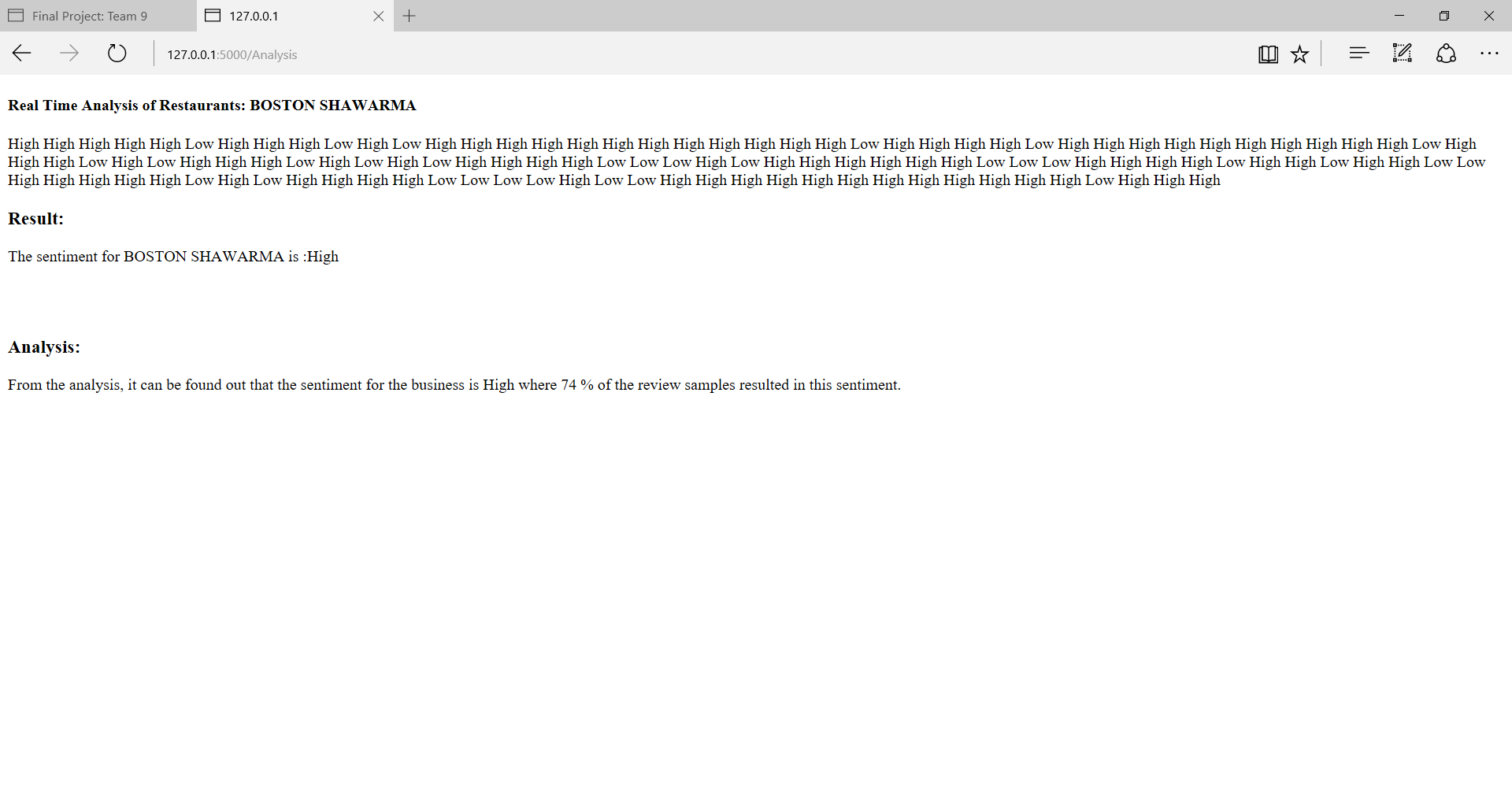


Figure: Output sentiment for a restaurant

## Prediction of Ratings for new Business

1. The cleaned data was loaded on Azure and the best predictors were chosen using Chi square method.

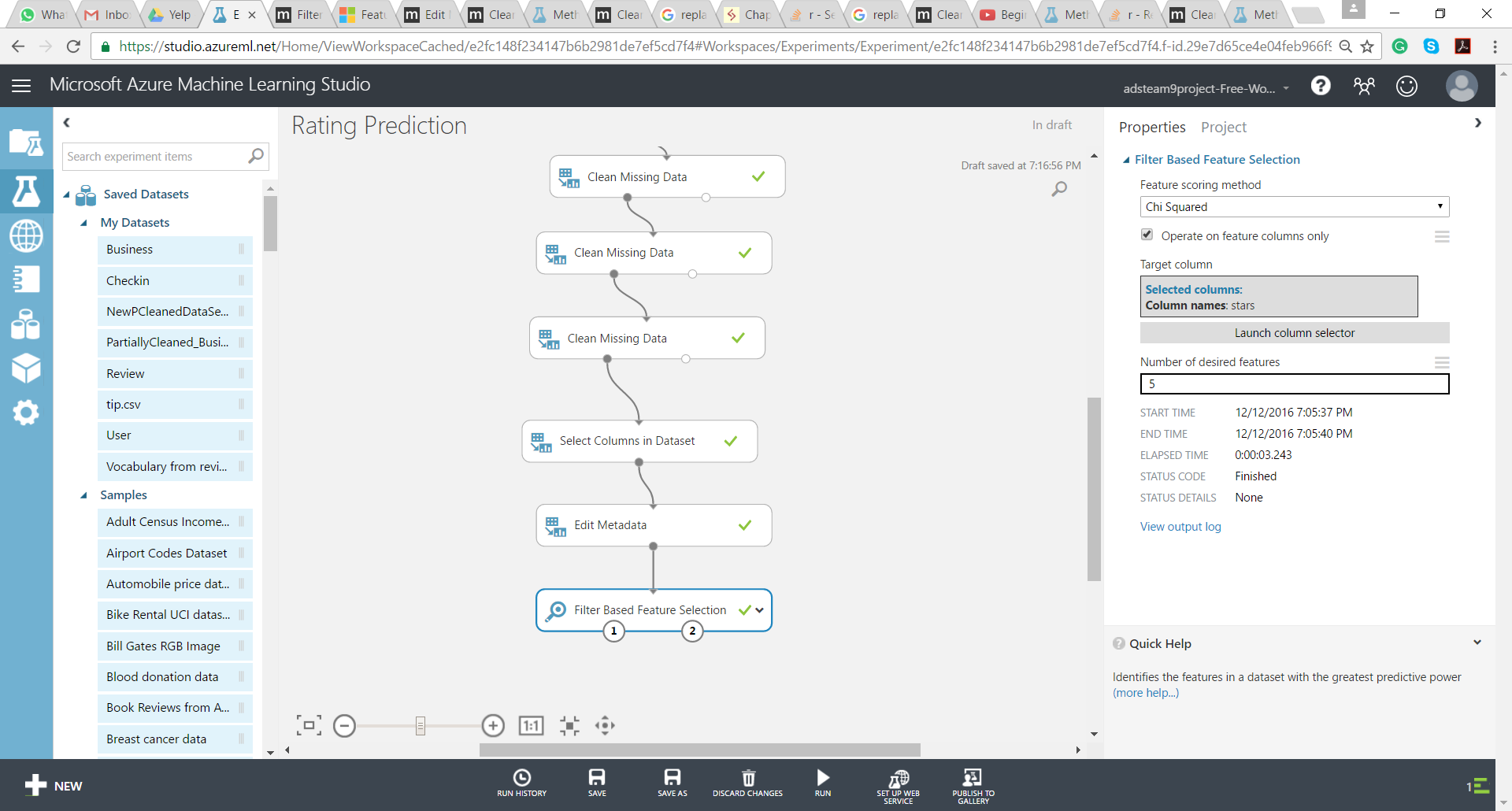
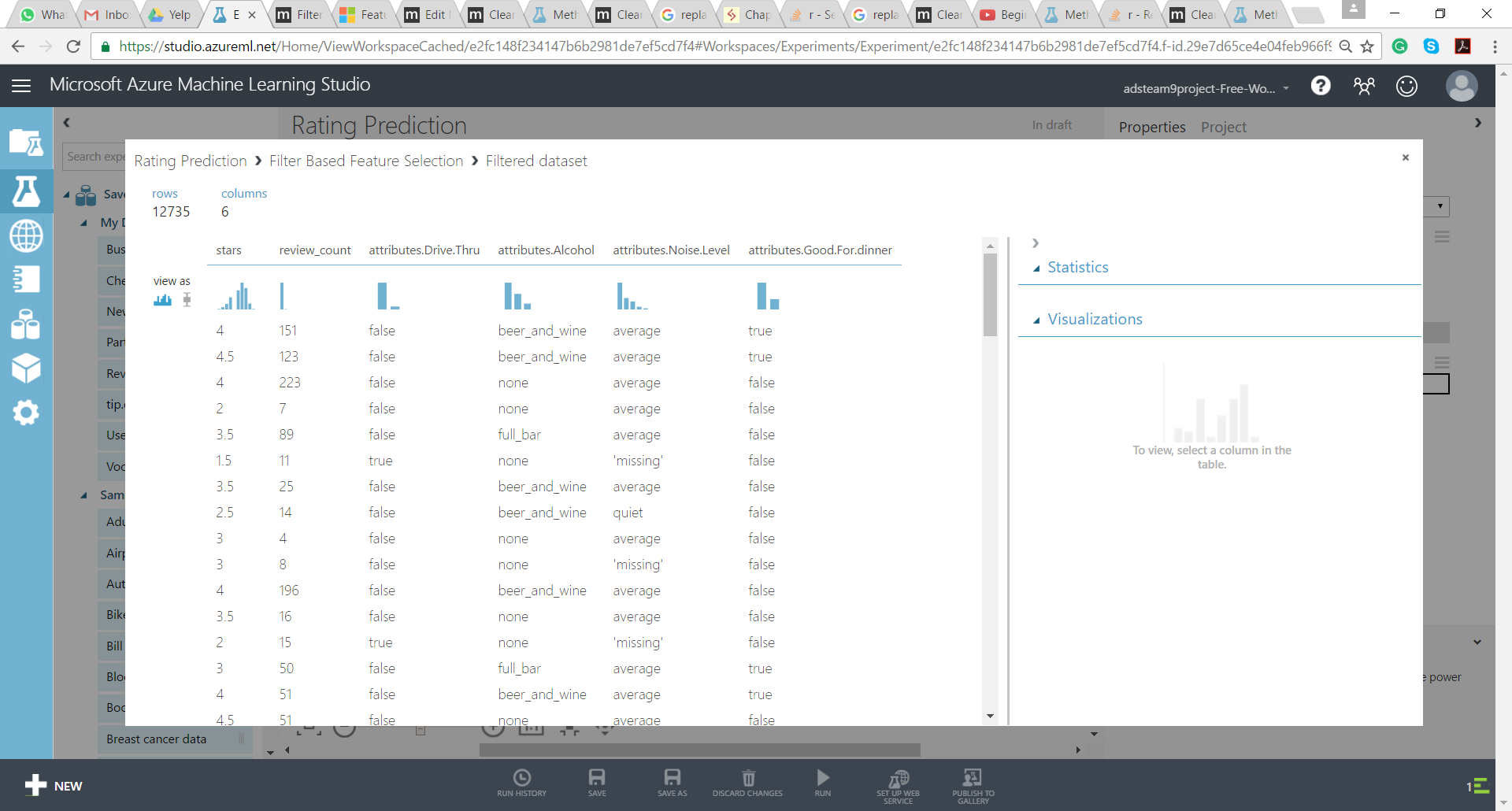
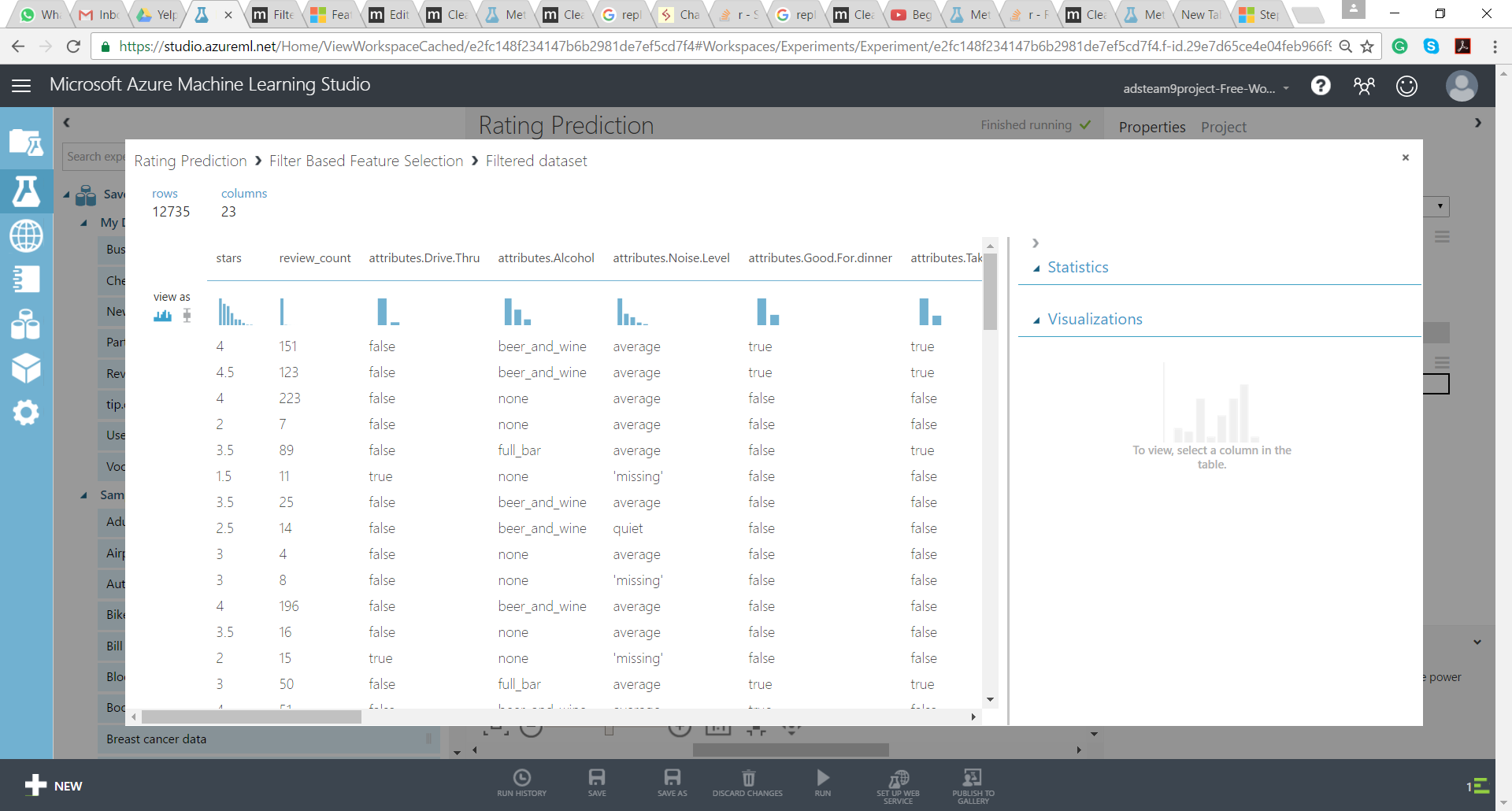


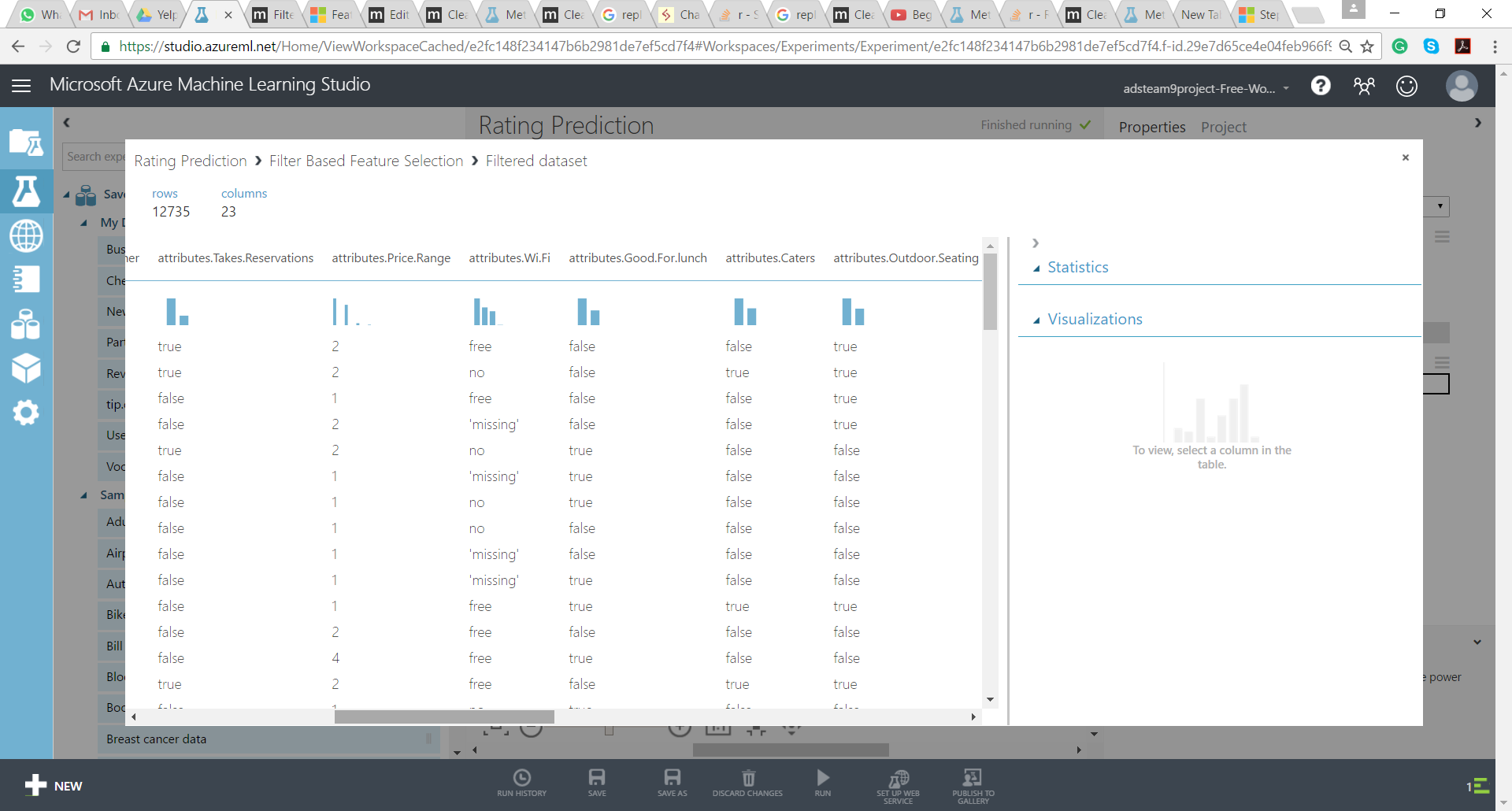
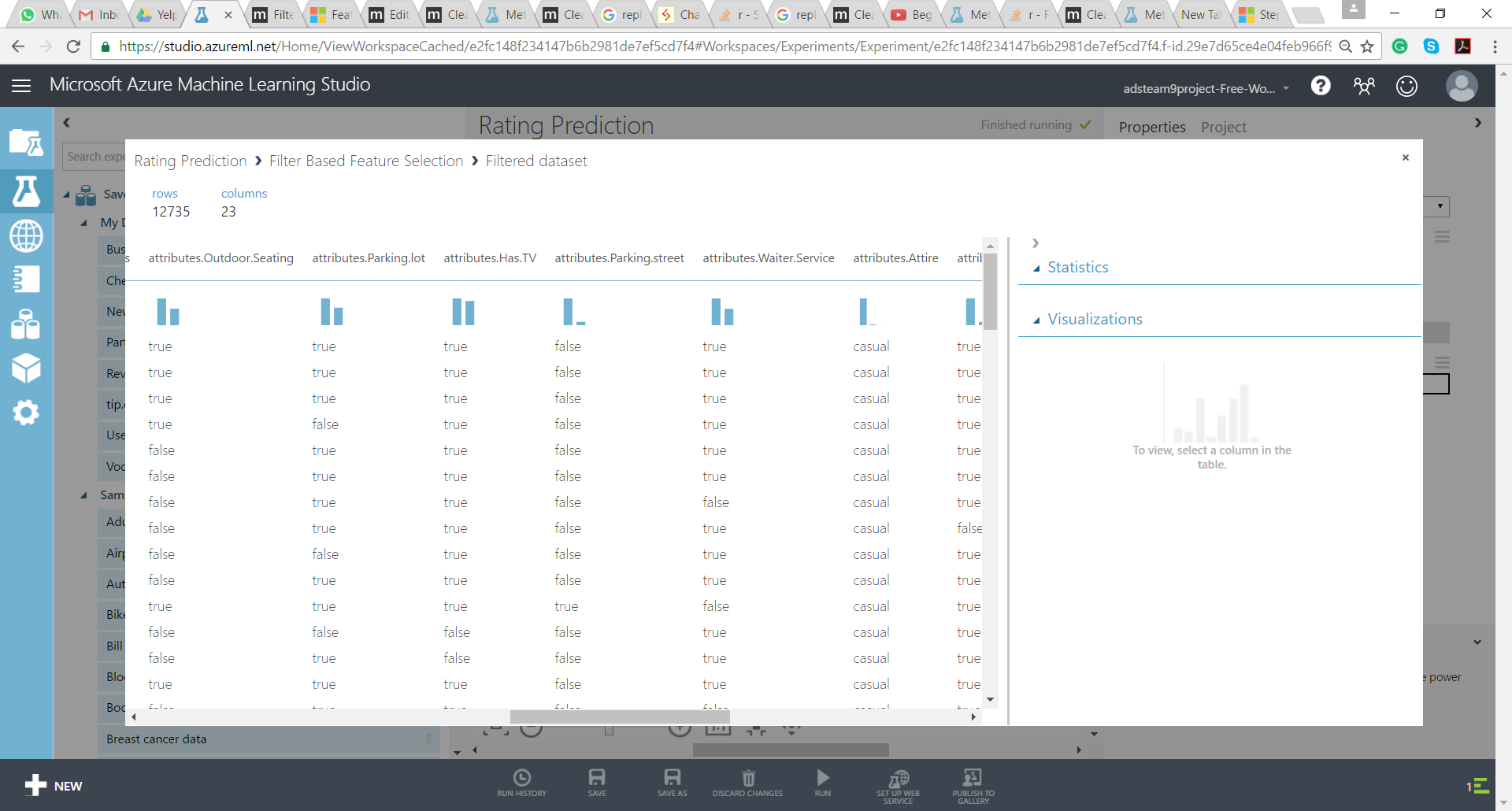
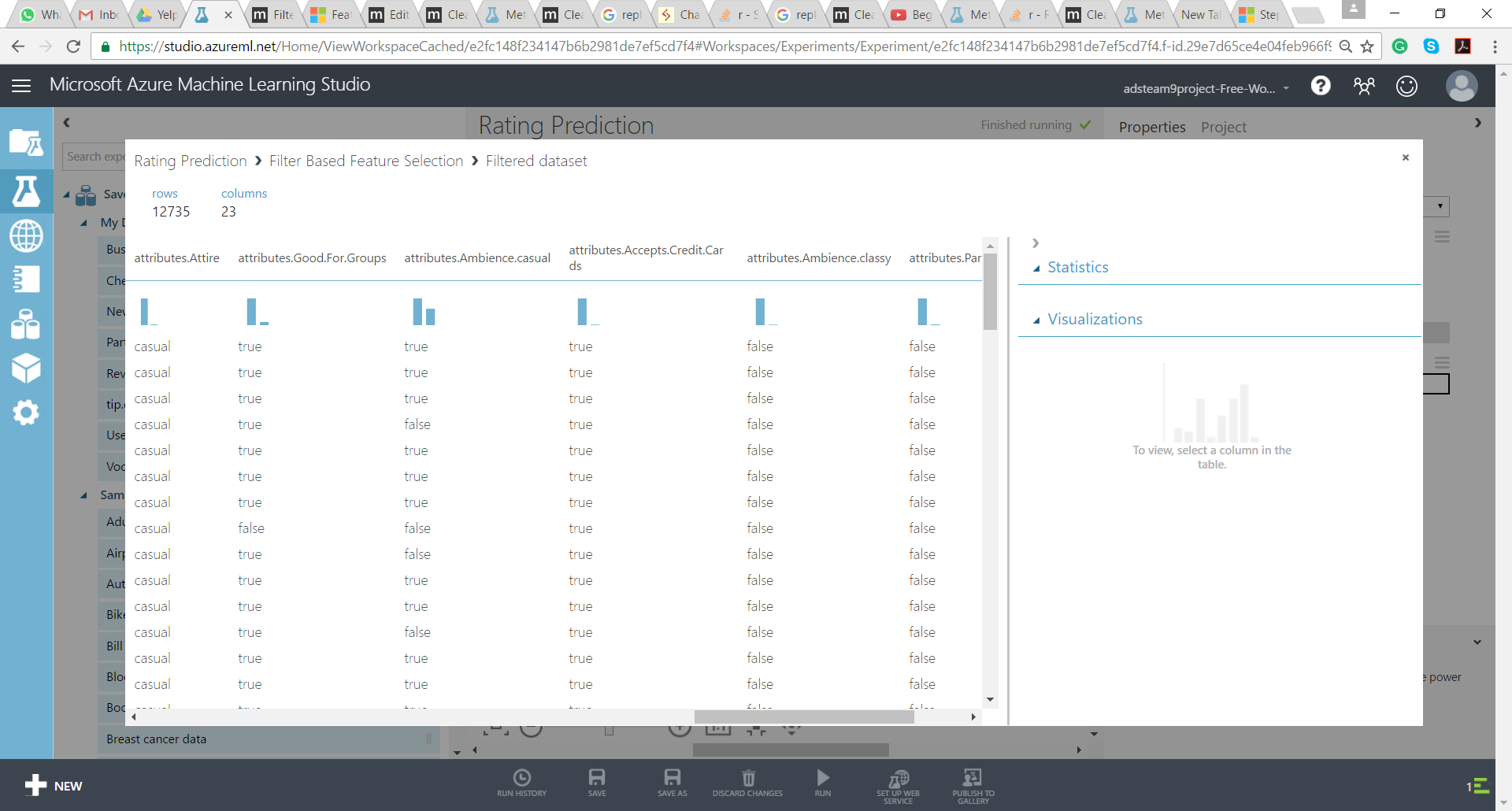
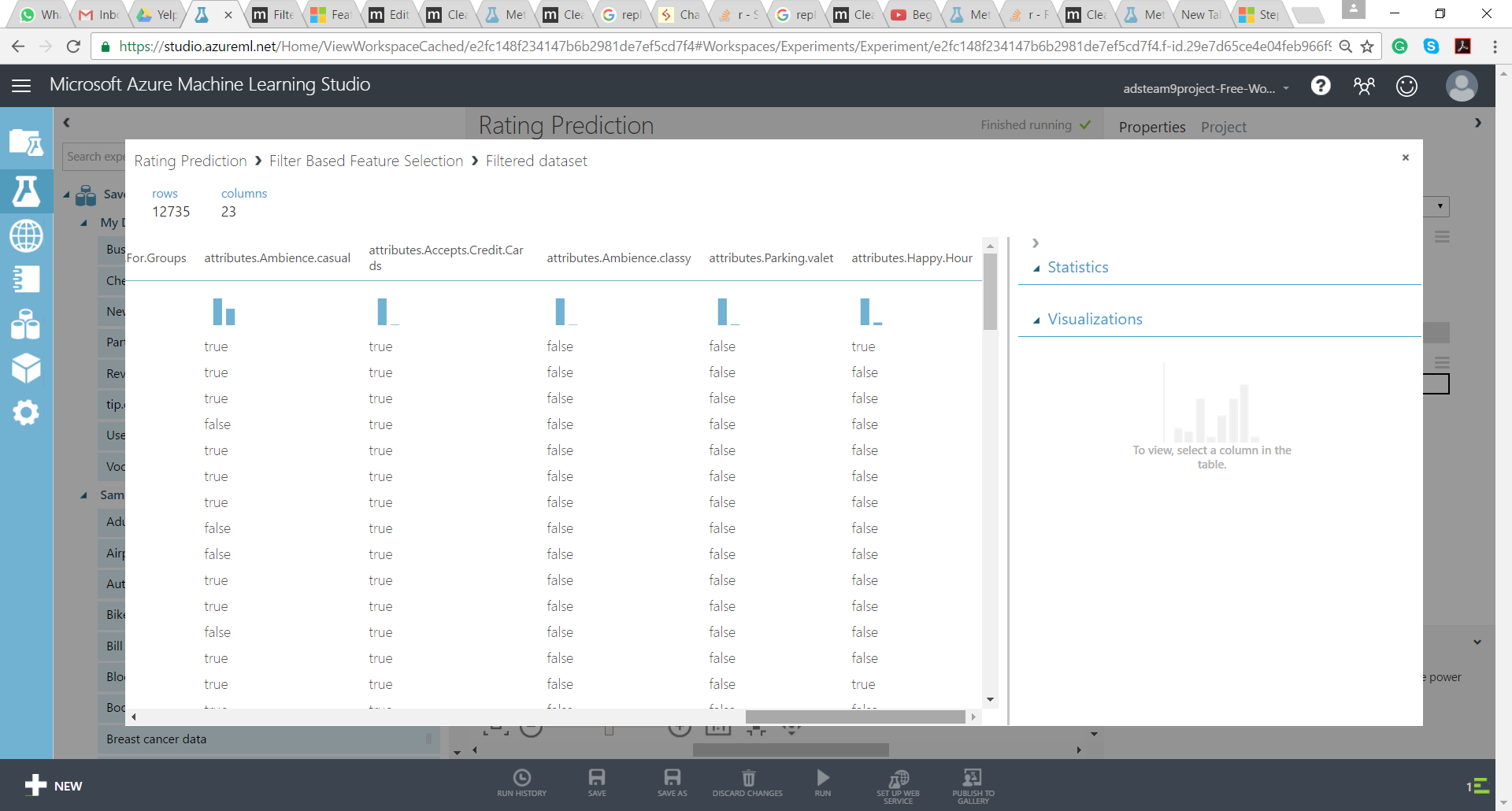
Figure: Feature selection based on Chi square method

1. Based on the feature selection and on desired no of features i.e 5 the following features were identified by Chi Squared method

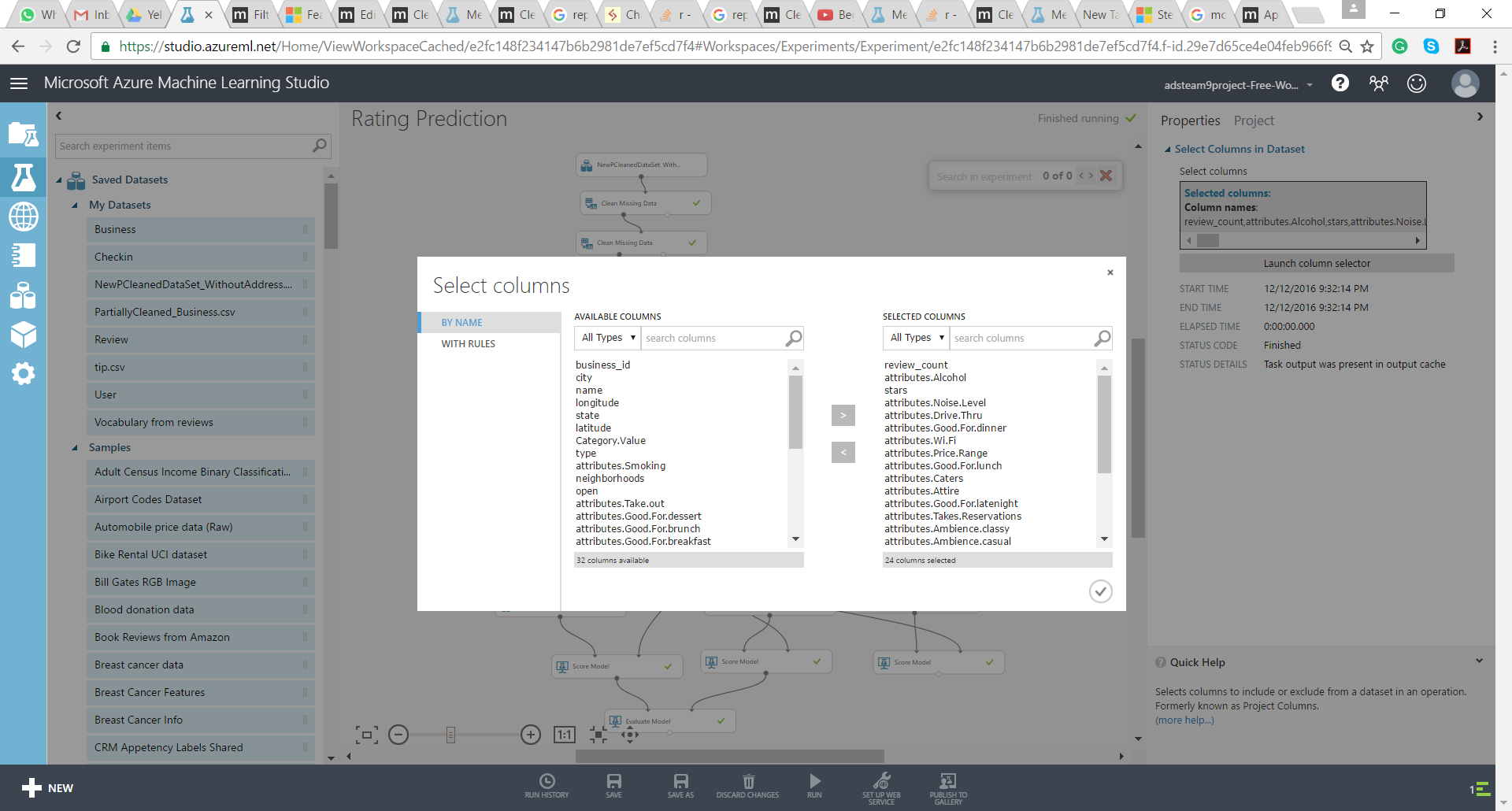


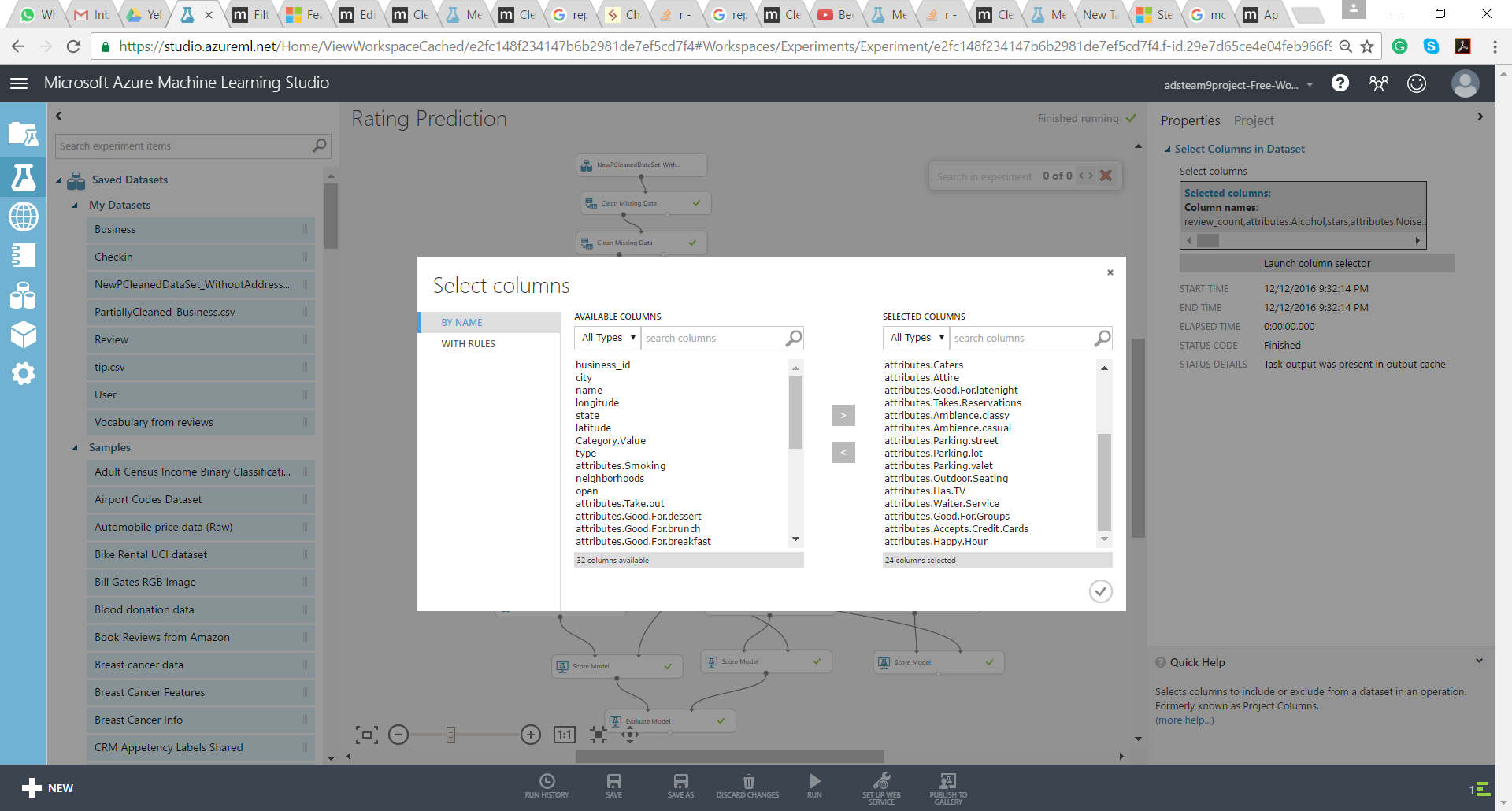
1. Based on the feature selection and on desired no of features i.e 24 the following features were identified by Chi Squared method



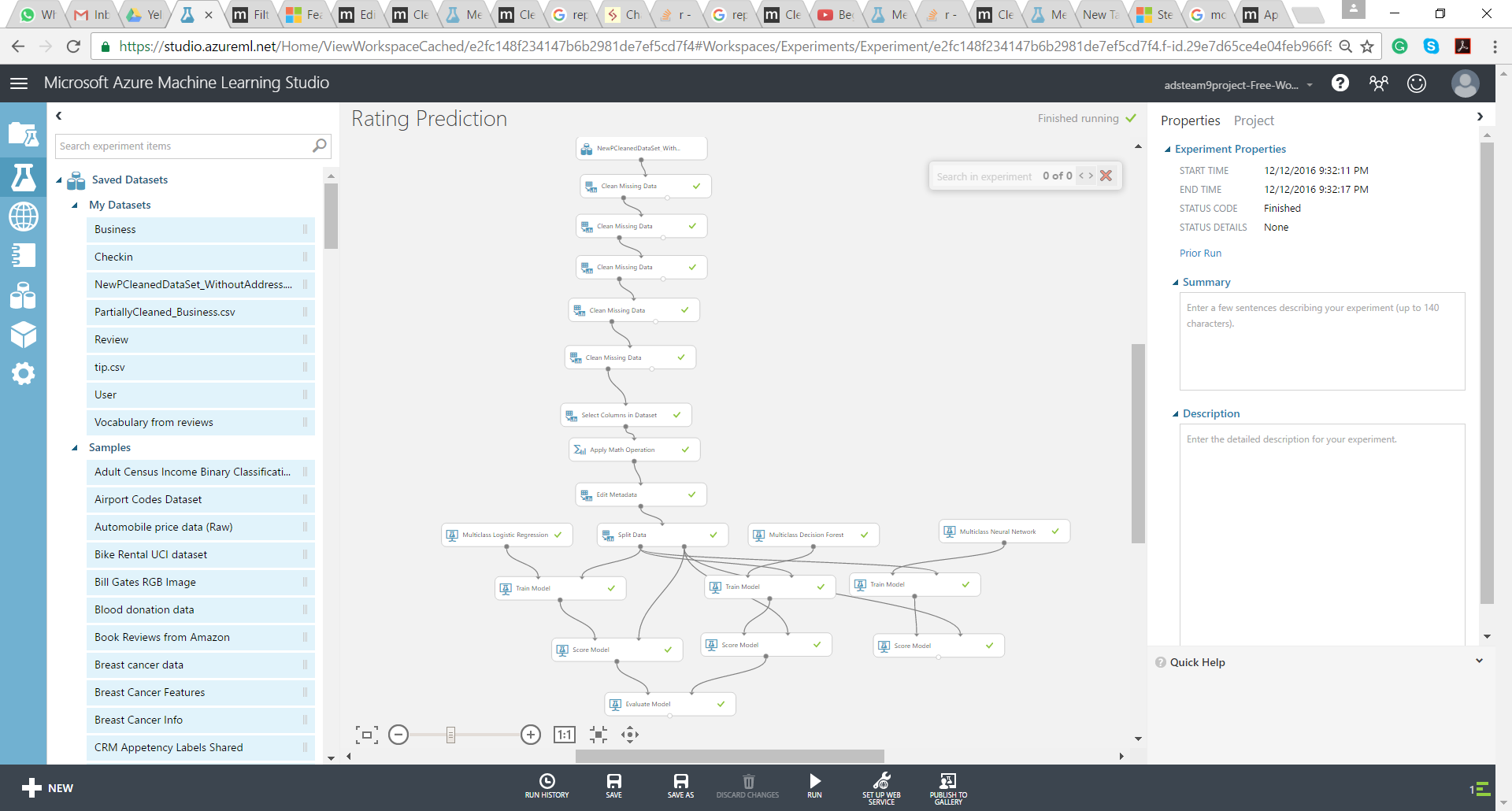
1. Selecting columns based on feature selection result as follows:

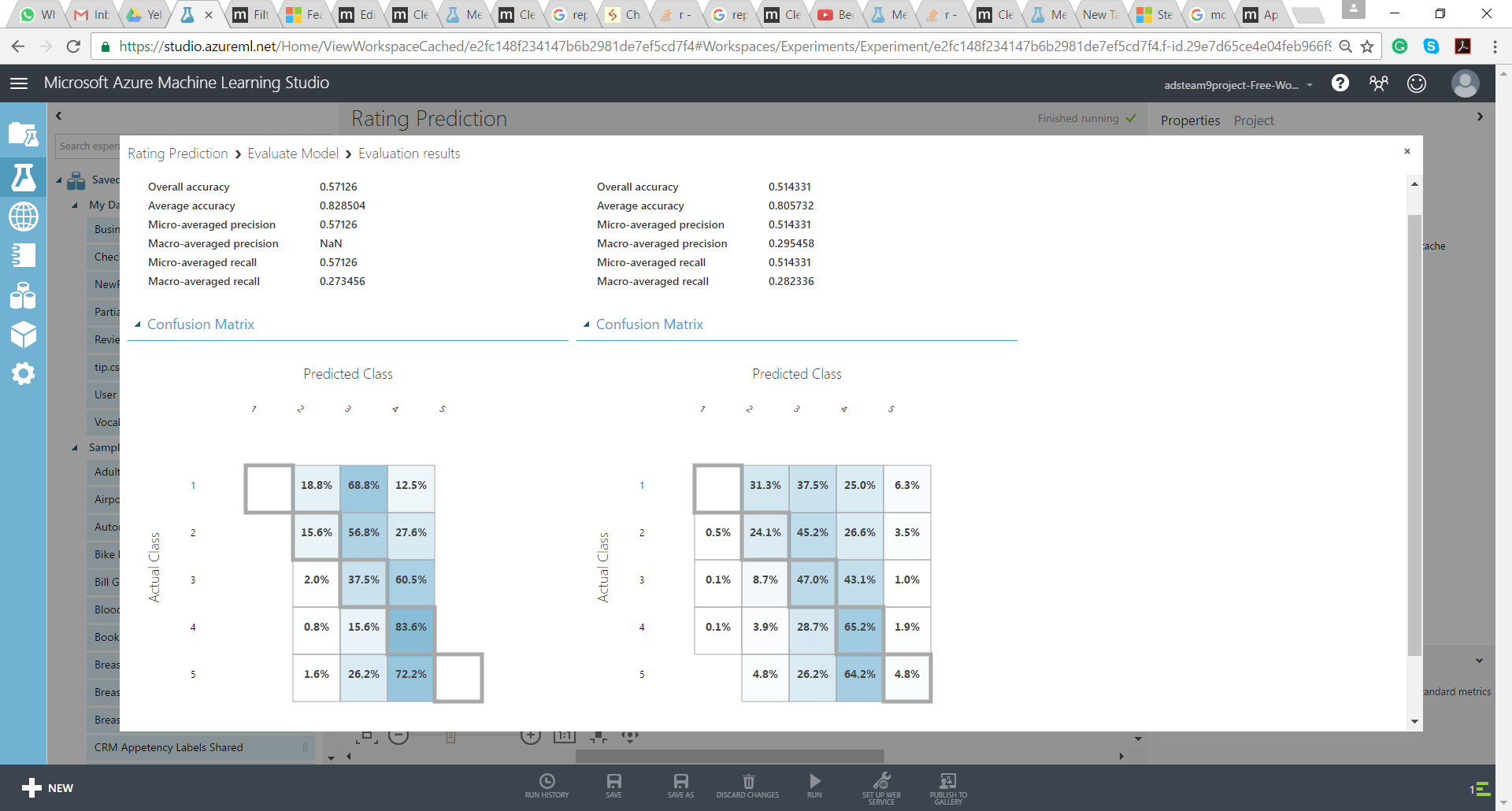




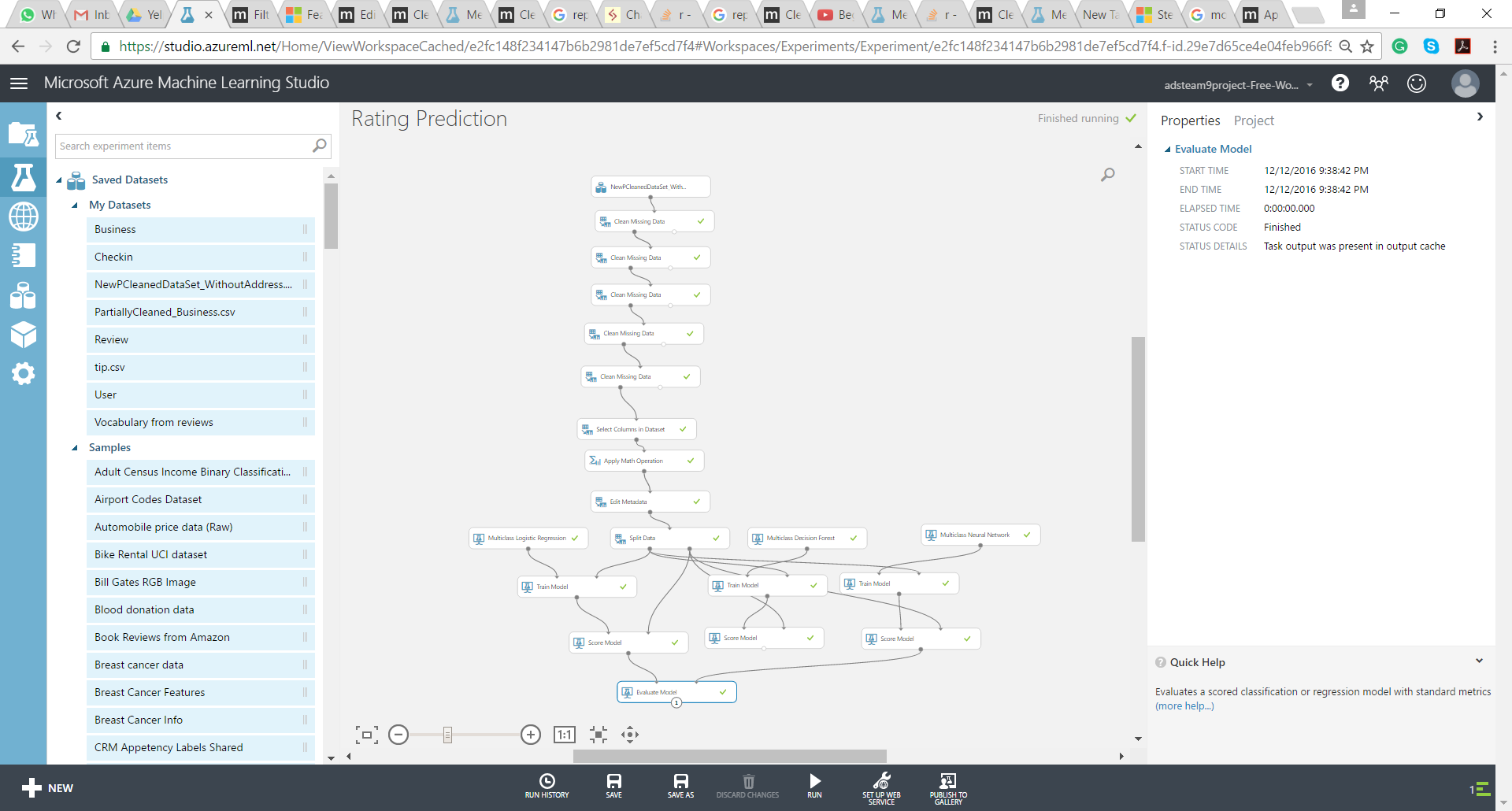
1. Evaluating different models:

* Multiclass Logistic
* Regression and Multiclass Decision Forest comparison





1. Multiclass Logistic Regression and Multiclass Neural Network





1. Based on the above result seems Multiclass Logistic Regression is apt for this prediction. 