INFO 7390

Final Project Report

Processor: Sri Krishnamurthy

Group 1

Chang Yu, Songzhe Zhang, You Li

Pre-process

The data pro-process includes related material adding and data cleaning. In order to promote the precise of our model. The content list below will describe how we add weather, address, workdays together in to our final dataset.

In the first part, we gather the trips of Uber from January to July with the trip's location information.

```
numDispach<-read.csv("D:/myspace/R/final/uber-raw-data-janjune-15.csv")

LocationList<-read.csv("D:/myspace/R/final/ZipCode.csv")
```

After we gather two datasets together, we get all the location variables connected with its trips numbers.

According to the graph we can easily distinguish the changes from month to month in one year, and changes from day by day in each month. The next step for our analysis is to get the whole trend of energy consumptions in the whole year on the unit of date.

To get the related weather data, we try to use the weather API from www.wundergound.com. However, the price to get the history is about 3000 dollars. Obviously, it's impossible for us to purchase so much money for so precious data. So we

```
String a="https://www.wunderground.com/history/airport/KNYC/2015/3/"+day+"/DailyHistory.html?req_city=New+York&req_state=NY&req_statename=Ne
url = new URL(a);
URLConnection conn = url.openConnection();

// open the stream and put it into BufferedReader
BufferedReader br = new BufferedReader(
                    new InputStreamReader(conn.getInputStream()));

String inputLine;
while ((inputLine = br.readLine()) != null) {
```

use this program get the hourly data and daily data from wundergound.com.

We modulate the change of the date through java programming and output this information into our weather.csv files.

The output files are in different csv files with same headline.

Coming these data together need the help of these code:

```r
##merge data in 2015

a = list.files("test")   #list.files命令将blog文件夹下所有文件名输入a

dir = paste("./test/",a,sep="")     #用paste命令构建路径变量dir

n = length(dir)              #读取dir长度，也就是文件夹下的文件个数

merge.data = read.csv(file = dir[1],header=T,sep=",")
#读入第一个文件内容（可以不用先读一个，但是为了简单，省去定义data.frame的时间
#，我选择先读入一个文件。


for (i in 2:n){
  new.data = read.csv(file = dir[i], header=T, sep=",")
  merge.data = rbind(merge.data,new.data)
}
```

No-work day include weekend and holidays, to get the weekend of different week. We write a java program to generate the weekend and combine it with the holiday we searched from online making it our no-workday.

```java
public class workday
{
    public static void main(String arg[])
    {
        for( int i =1; i<=365; i++)
        {
            if(i%7==4||i%7==5)
            {
              System.out.println("0,") ;
            } else System.out.println("1,");
        }
    }
}
```

Through combining the data in the scope of areas, we get the dataset which can be used to predict the trend to Uber trips in hours, days, months. The model for servicing drivers end is mainly developed from these data sets.

```r
holiday<-cbind(getOne,year,month,day)

material1<-sqldf("select * from deal join holiday where deal.year= holiday.year
                 and deal.month = holiday.month and deal.day= holiday.day ")

write.csv(material1,"chedan1.csv")

material1=read.csv("chedan1.csv")


HourBase<-sqldf("select spec.Dispatching_base_num, count(*) as numOutput,spec.year,
                spec.month, spec.day, spec.hour, spec.Borough , spec.Zone ,
                spec.ZipCode,material1.Date, material1.TemperatureF,material1.Preci,
                material1.Events  from spec join material1 where spec.month=material1.month
                and spec.day = material1.day and spec.hour = material1.hour
                group by spec.Dispatching_base_num,spec.year,spec.month, spec.day, spec.hour")


write.csv(HourBase,"2.csv")
```

On the other way, we join these datasets restricted on the scope Base and group the results by hour, day.

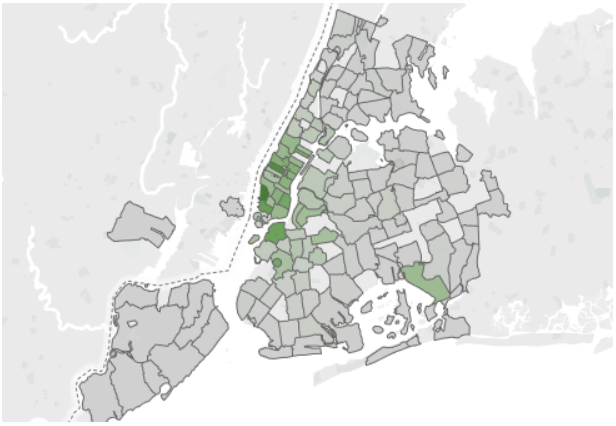
This dataset will help us analysis the number dispatched by Bases. Through the prediction on the macro scope， Uber can get the model to predict its trend of its business and design related strategies to adapt to that.

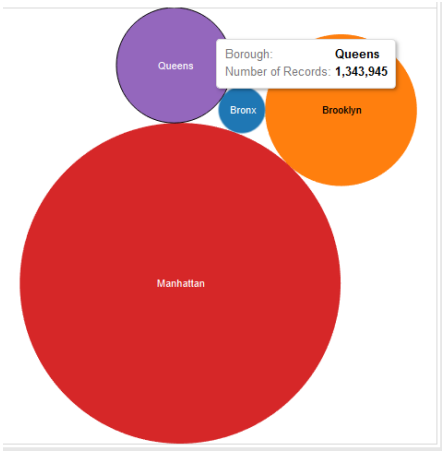
On the other hand, the prediction mode can also get the potential maximal number of the service number for each Base. So if maintenance is needed for one of the base, the replace plan can be designed through this model.

Visualization Analytic

The distribution of Uber trips in different zip area

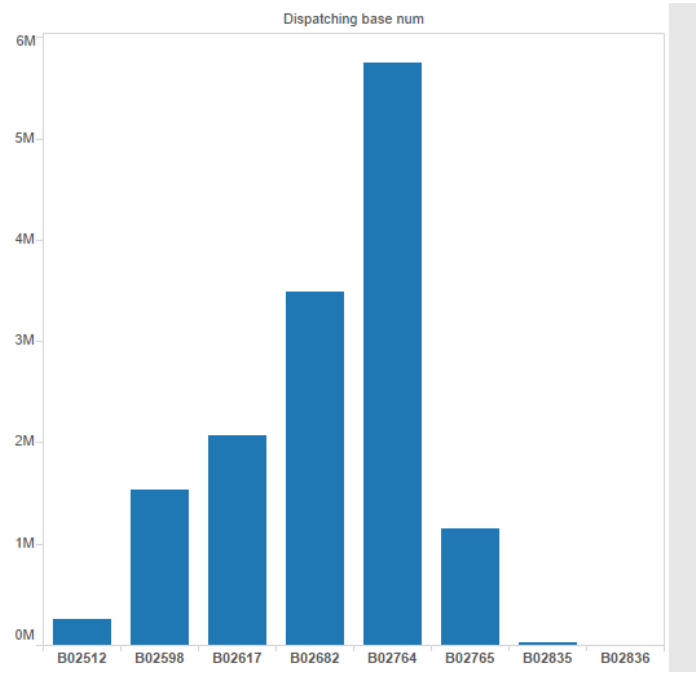
The graph will show the number of Uber trips from the darker or lighter of the color in these zips. We can easily tell that the number of trips is affect by its area. So it's necessary to take the variables zip code in to consideration.





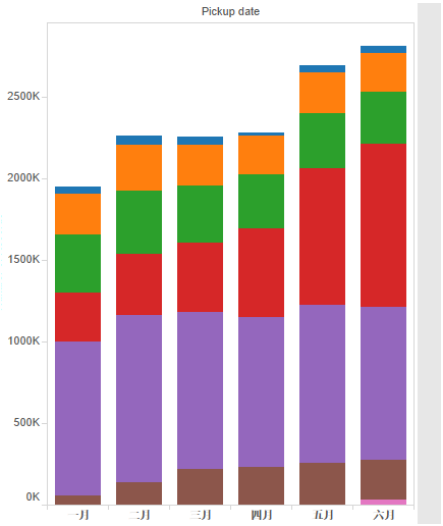
Different Dispatch Base undertake different levels of tasks.

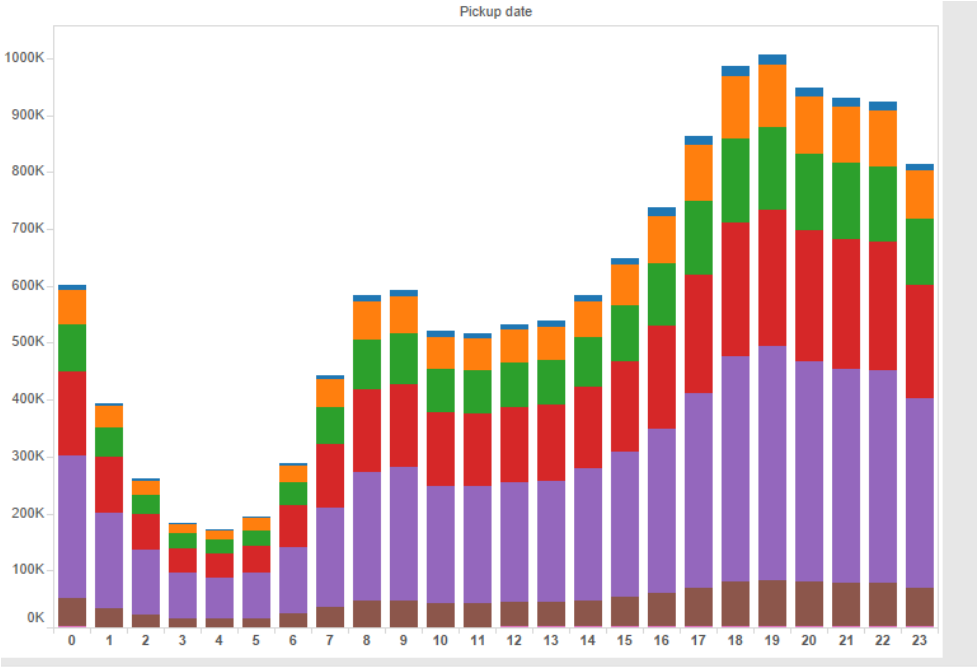
As what you can see, the different number of dispatch's tasks is totally different, it is necessary to make them into categories before we cast them in to analysis.



When we put the number of bases in to time scope. Increase of trips mainly happened on B02683 and B02765. The chang of other parts seems not so obvious.

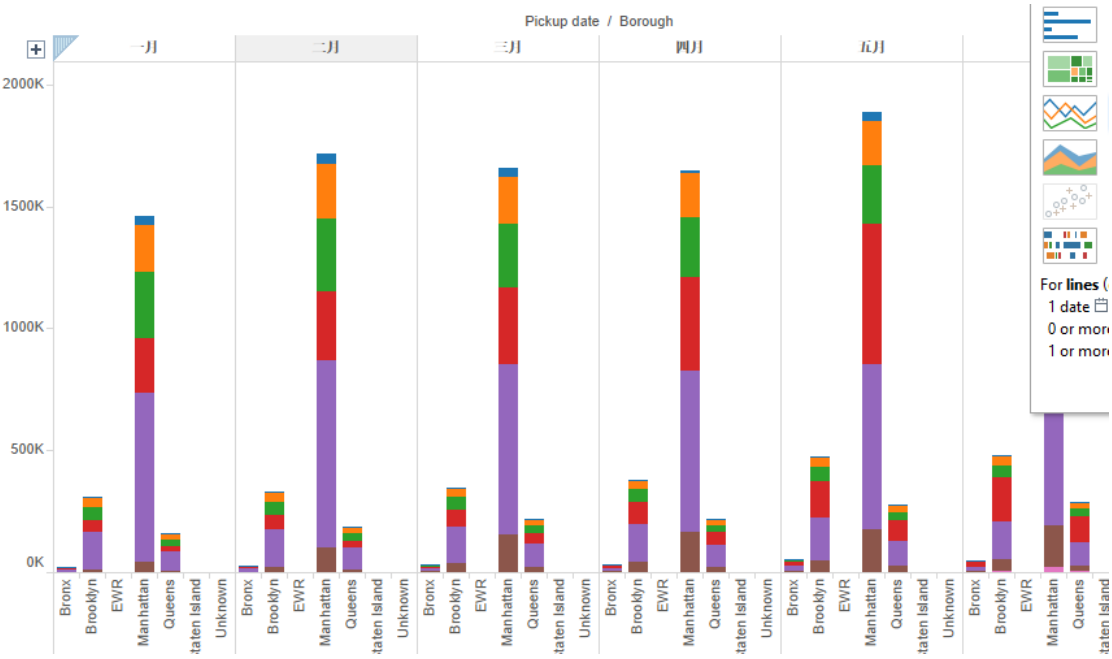
We also compare the trend to Dispatch Bases in hours of a day for similar analysis.





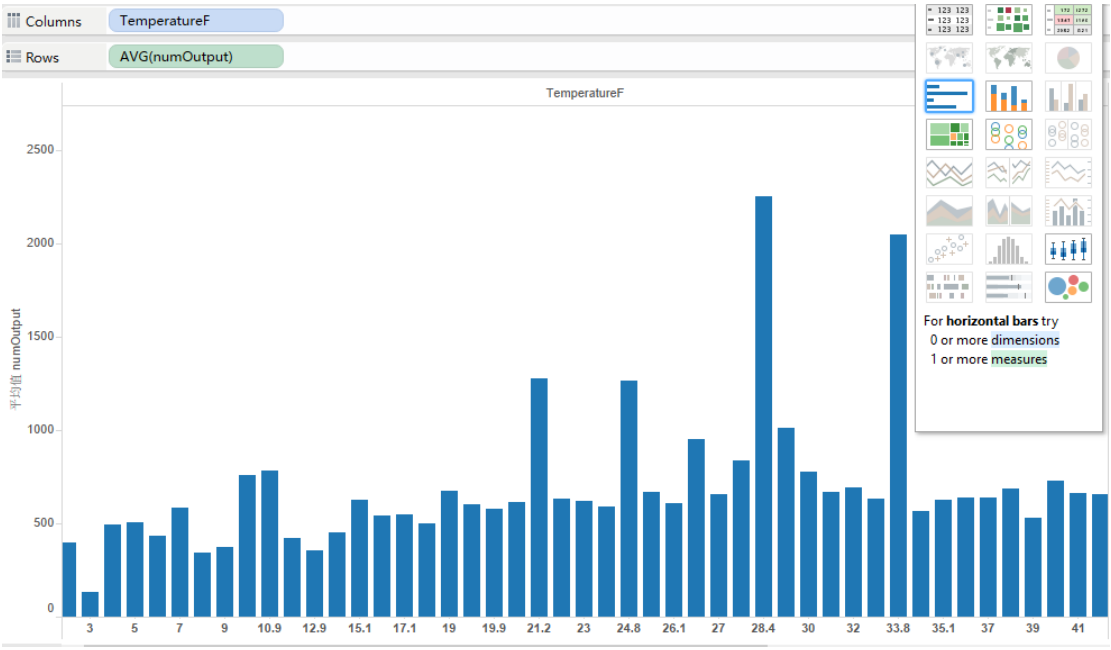
Relationship between area, Dispatch Base and Months

We gather Zone, Base and Date based on month together to get the graph to display their relationship.



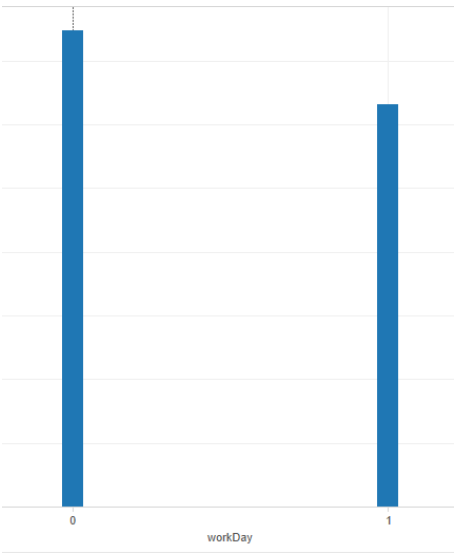
Temperature with trips

Through the graph of temperature with number of trips. We can see their relationship is not so obvious.



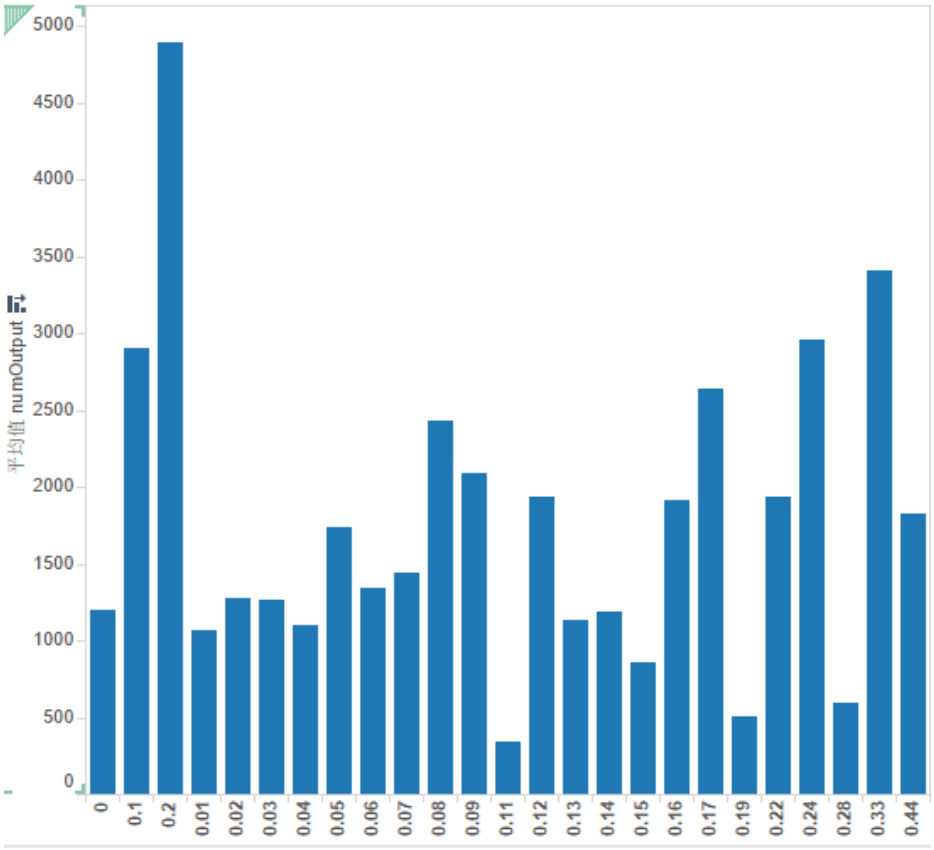
Workday with trips

In this graph, we can see the relationship between workday and number of trips. In no-workday, we can see the increase of trips is so obvious. We really should take it in to our analysis.



Participation with trips

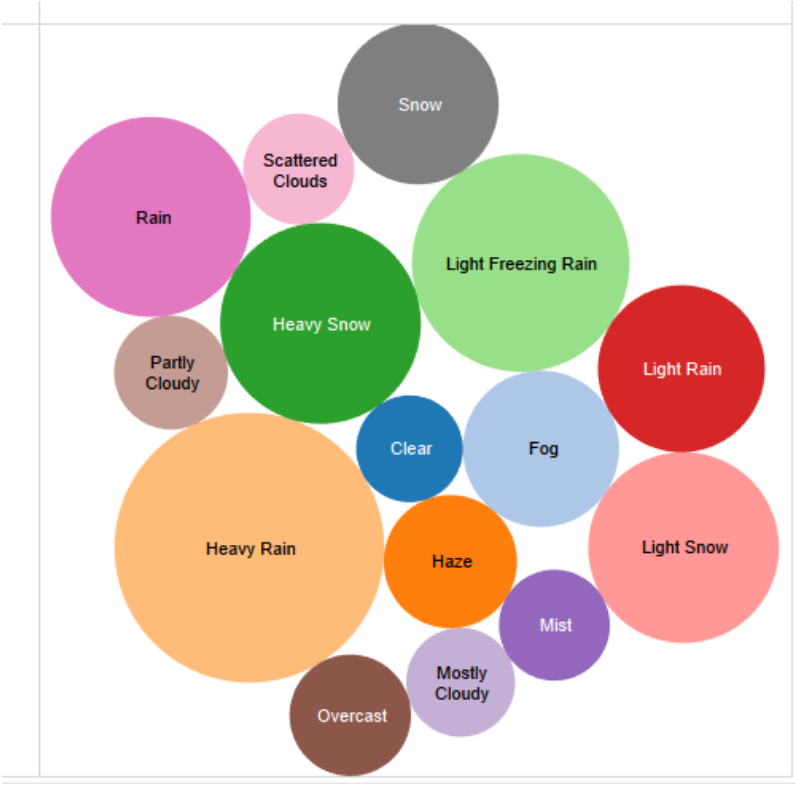
The volume of perception is also displayed blew. However, as the grown of the volume, the number trips still in the condition from rise to fall. The relationship is really hard to tell through visualization, so we keep these data and test it in Model Part.



Weather Conditions with trips

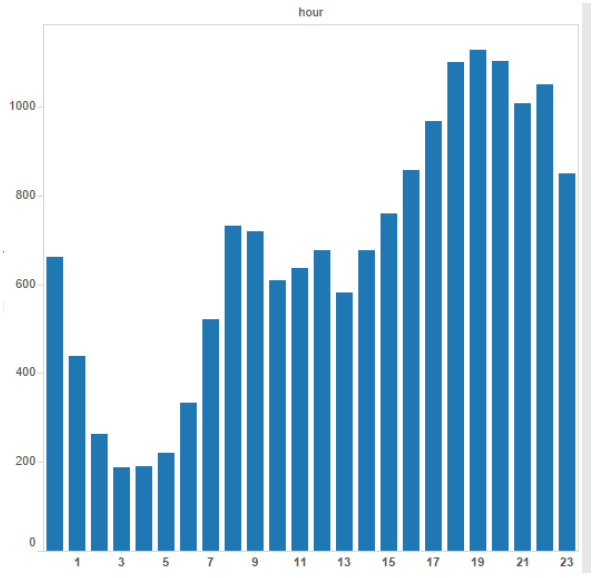
In the terrible weather such as heavy rain, heavy snow and so on, the trips obviously increase a lot. In the conditions such as rain or snow, there is also an incensement.

Surely, we need to take consideration of this in our final project.



Different hours during the day with Uber trips

Looking through the change our hours in a day. We can see that the number of trips becomes higher in a 7-9 am, 4-7pm. This trend tells us why the cost of Uber is higher than average time in commuting time. The indicate us the range of daily hours should also be considerate here.



Model

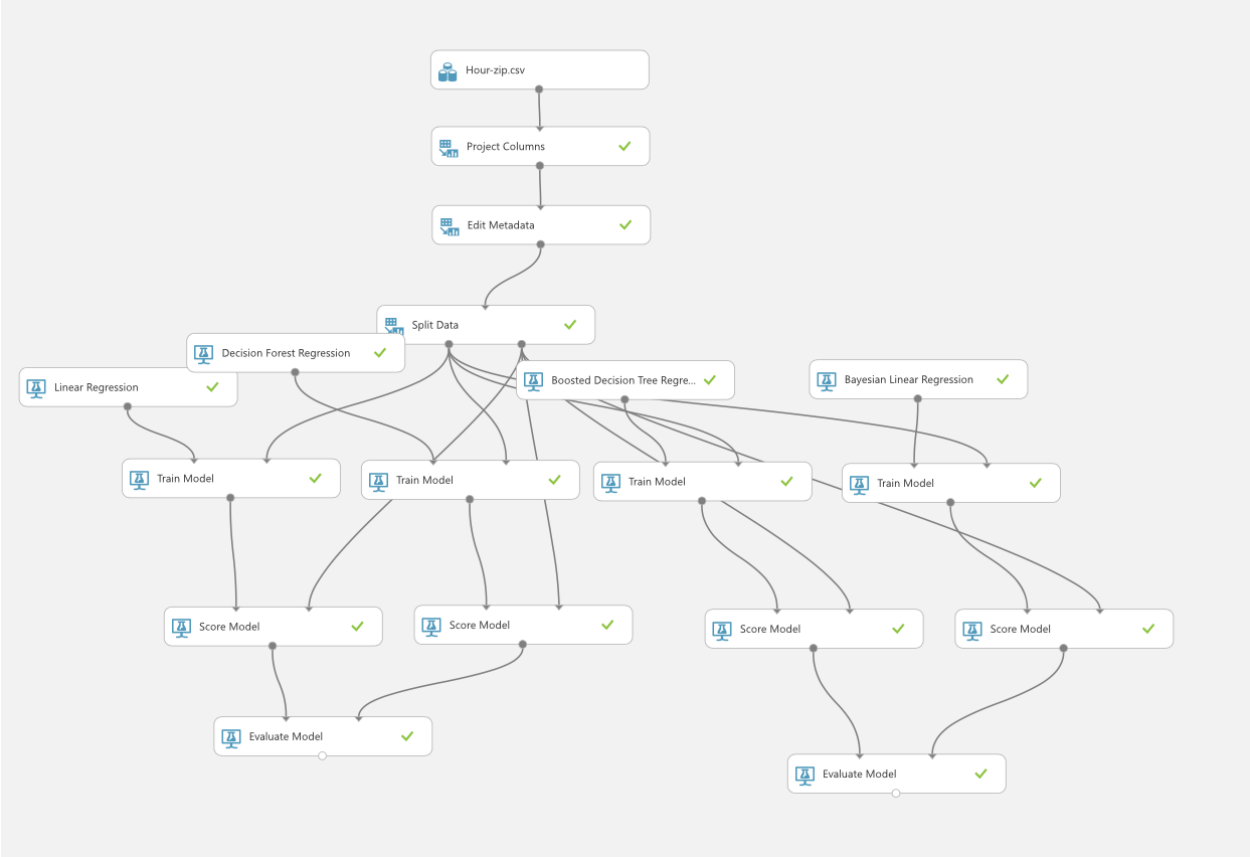
Problem Statement

Our project problem is try to use the 2015 Uber trip records to predict the Uber trips in the future day or hour in specific zone. Because the trips are a continuous numeric data, so we should choose a regression algorithm to solve this problem.

According to our problem, we decide to use Multiple Linear Regression, Boosted Decision Tree, Decision Forest Regression and Bayesian Linear Regression. We compare the evaluation of each and try to find out the best Algorithm for our problem.

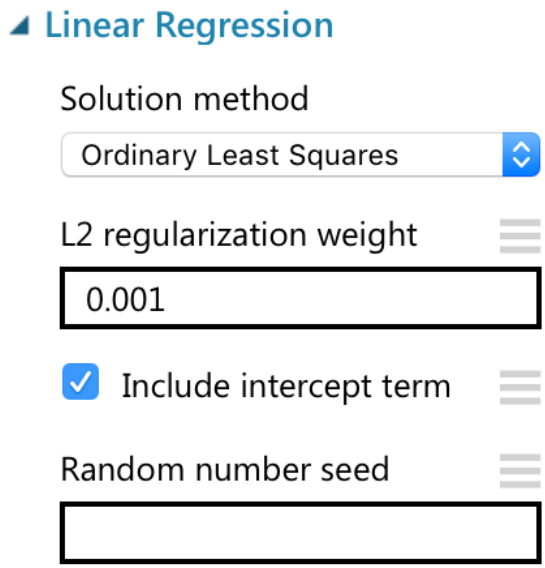
Model Hour-ZipCode

In this model, we count the records grouped by zip code and hour and join with weather, work day dataset. The inputs are date time, temperature, weather condition, work day and zip code. In the Azure Studio the training process like this:

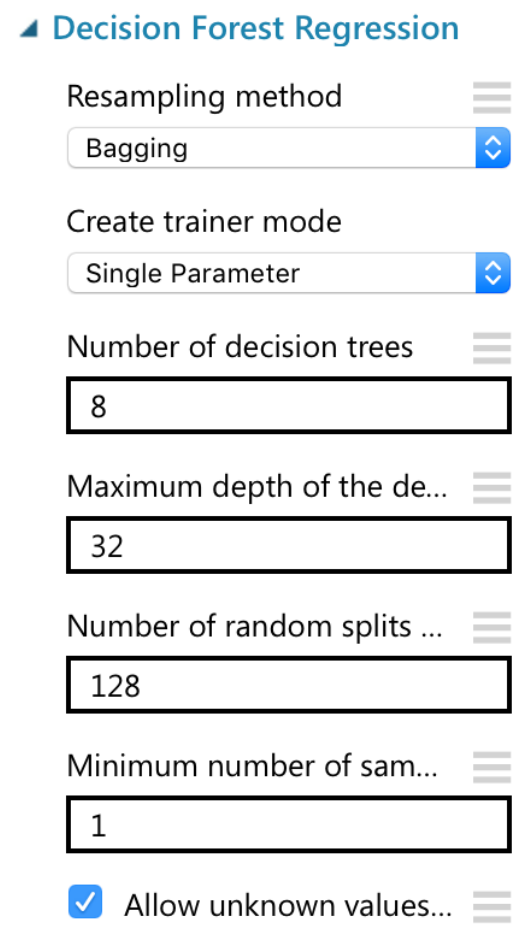


Model setting:

Linear Regression



Decision Forest Tree



Boosted Decision Tree

Boosted Decision Tree Regressi...

Create trainer mode

Single Parameter

Maximum number of leav...

20

Minimum number of sam...

10

Learning rate

0.2

Total number of trees cons...

100

Random number seed

Allow unknown categ...

Bayesian Leaner Regression

Bayesian Linear Regression

Regularization weight

1

Allow unknown categ...

START TIME	4/30/2016 ...
END TIME	4/30/2016 ...
ELAPSED TIME	0:00:00.000
STATUS CODE	Finished
STATUS DETAILS	Task output was present in output cache

Evaluation

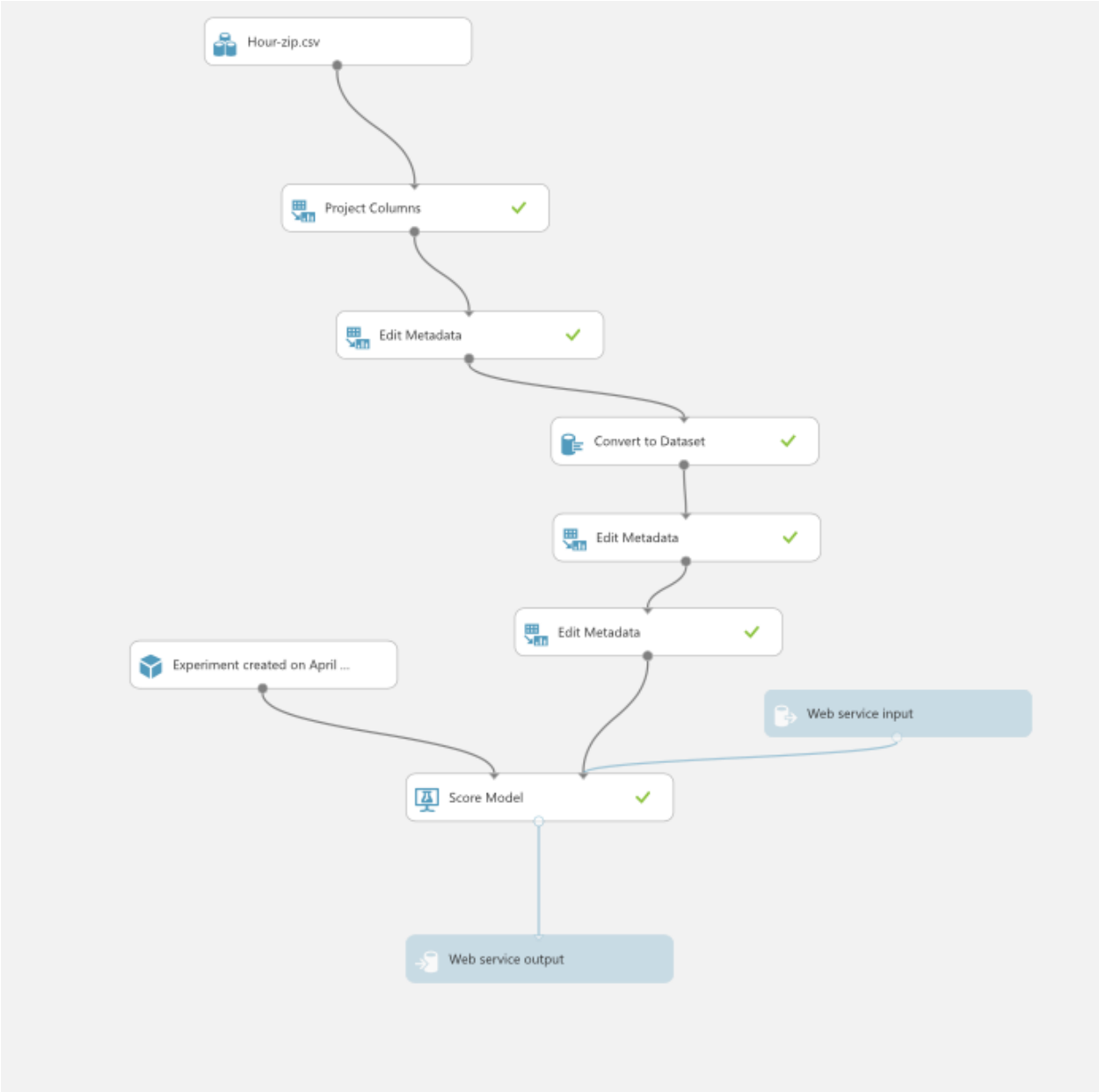
view as

Negative Log Likelihood	Mean Absolute Error	Root Mean Squared Error	Relative Absolute Error	Relative Squared Error	Coefficient of Determination
Infinity	12.303254	25.359921	0.608758	0.588842	0.411158
1144303.829743	12.233462	23.944698	0.605305	0.524954	0.475046

view as

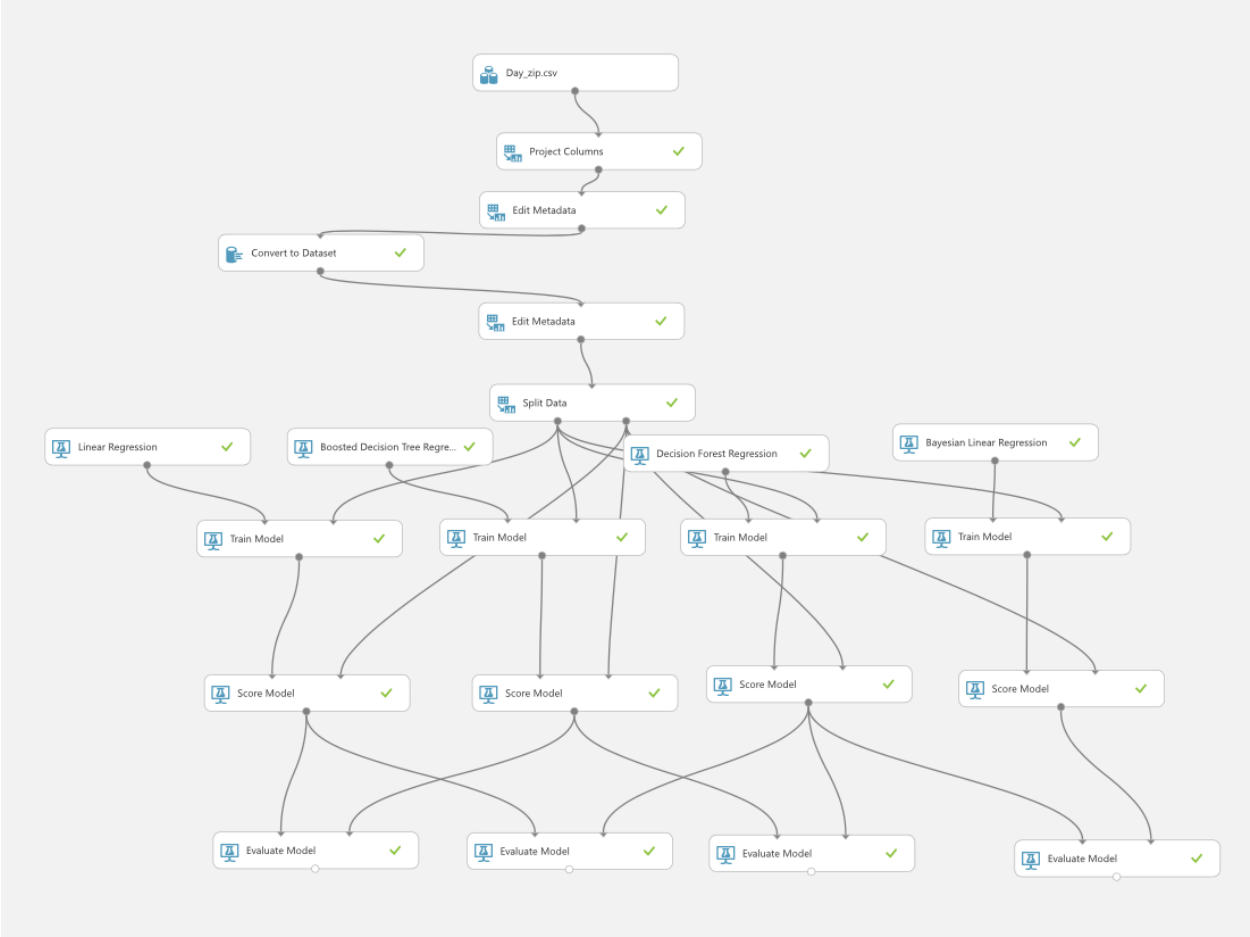
Negative Log Likelihood	Mean Absolute Error	Root Mean Squared Error	Relative Absolute Error	Relative Squared Error	Coefficient of Determination
Infinity	11.863739	24.163102	0.587011	0.534574	0.465426
1847448.803997	12.238018	25.433771	0.60553	0.592276	0.407724

Compare the Root Mean Squared Error, we can see that the smallest RMS is 23.944. As we know the smaller RMS the better the Model is, so we choose the model which RMS is 23.994—Boosted Decision Tree. Then we use the trained model to implement a service.



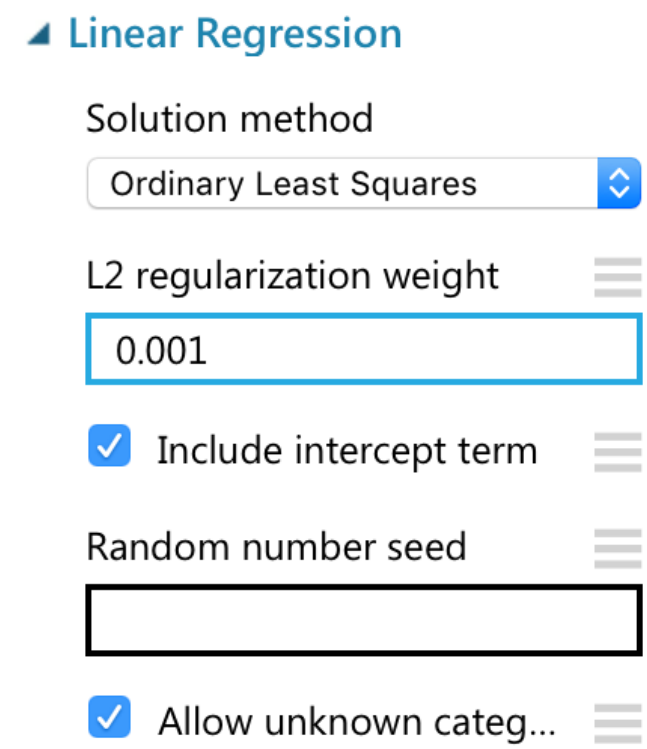
Model Day-ZipCode

In this model, we used large time scale to count trips. Inputs are date, weather, zip code, the output is a prediction of a specific zone' s trip in one day.

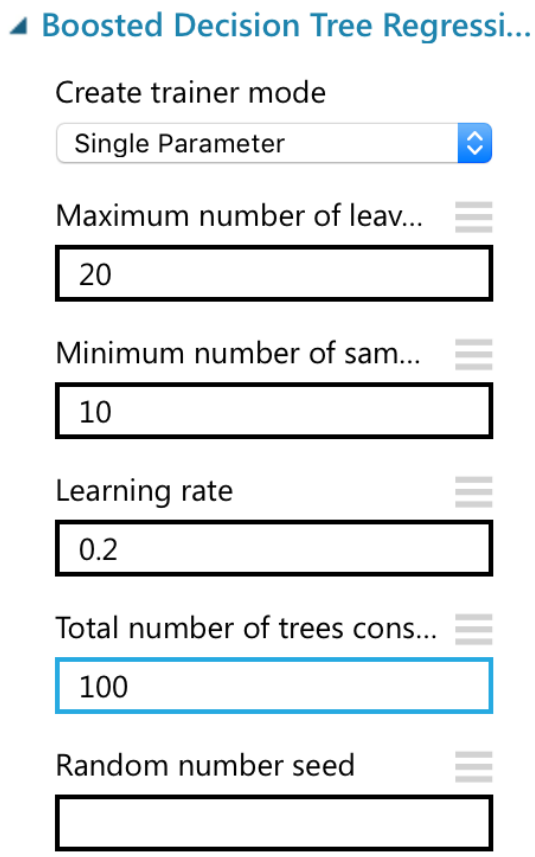


Algorithm Setting

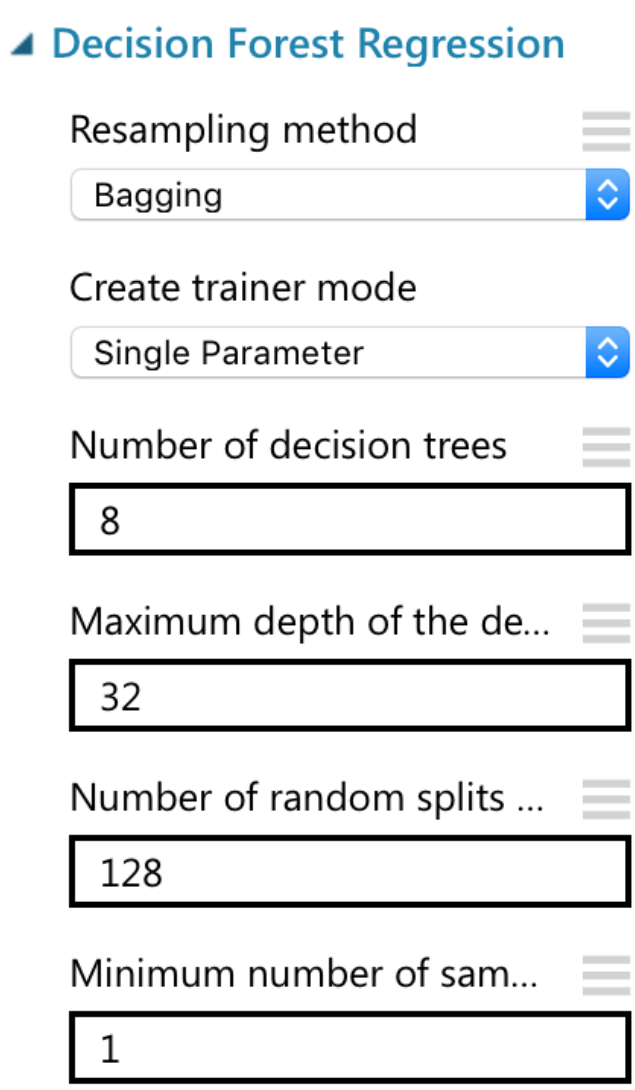
Linear Regression



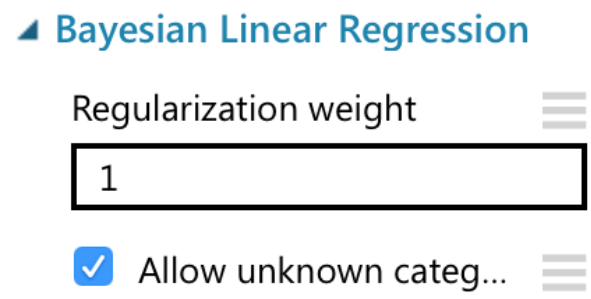
Boosted Decision Tree



Decision Forest Tree



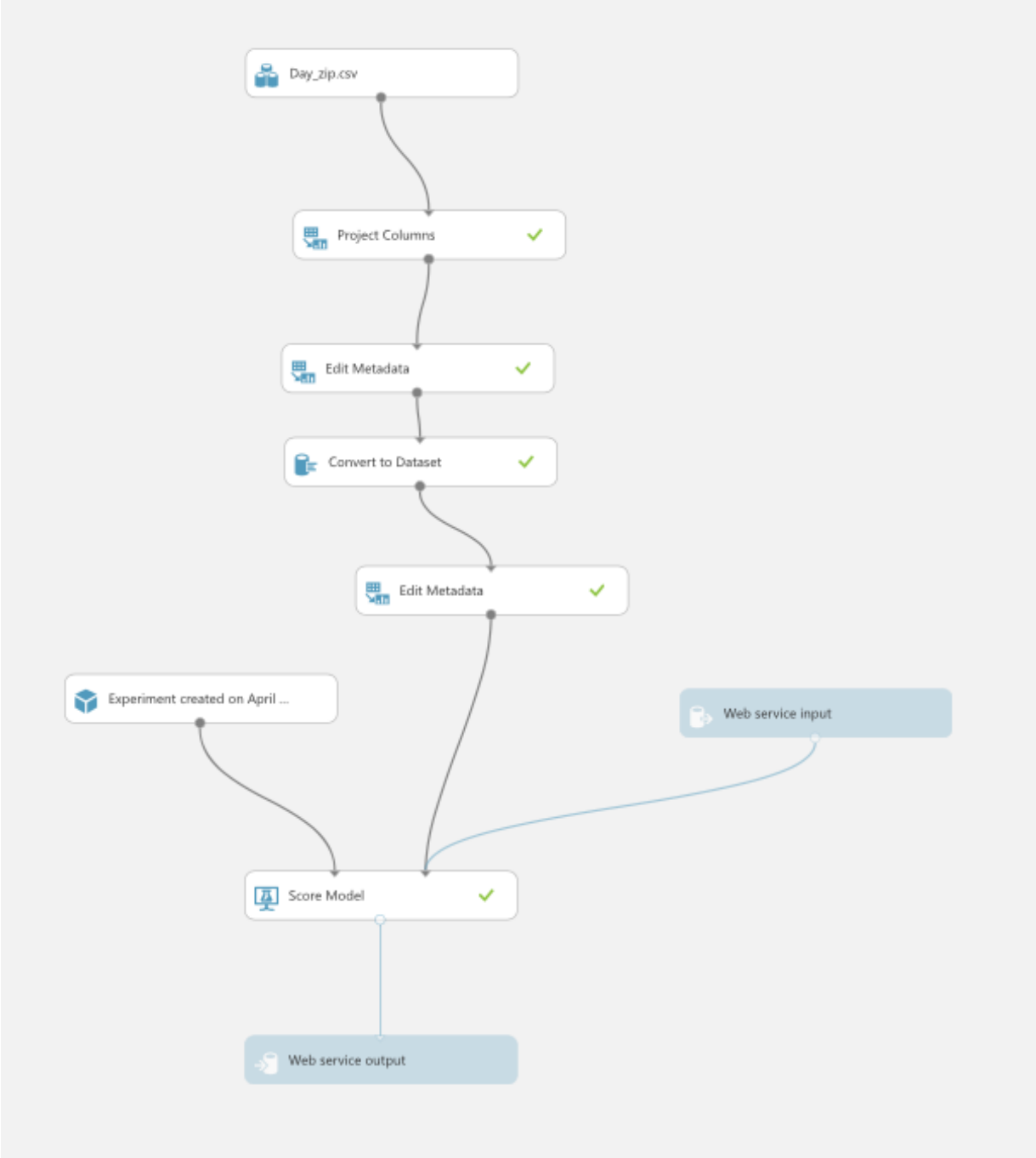
Bayesian Linear Regression



Evaluation

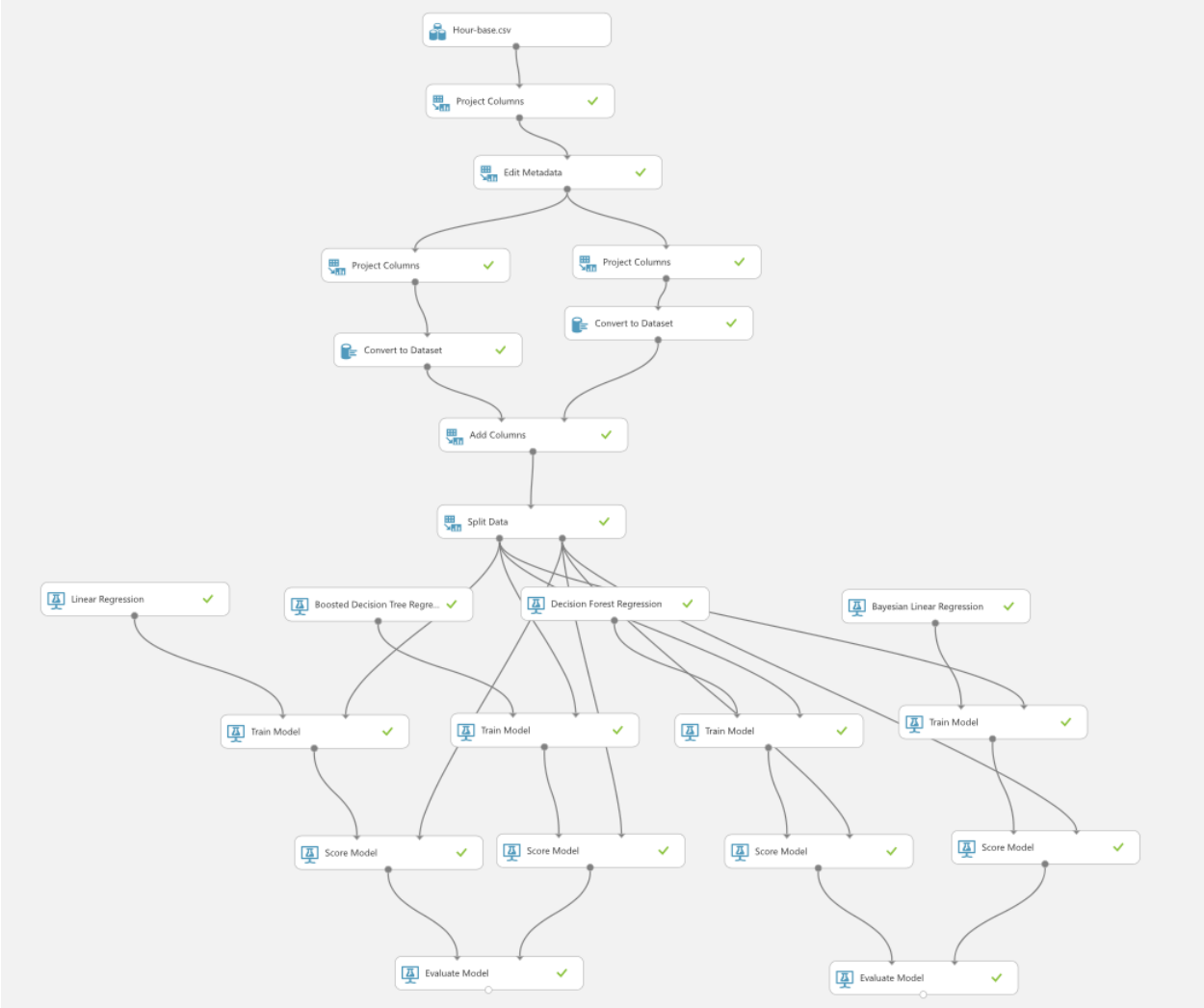
	Linear Regression	Boosted Decision Tree	Decision Forest Tree	Bayesian Linear Regression
Root Mean Squared Error	268.40	260.26	290.73	268.59

We also chose the smallest RMS Algorithm which is Boosted Decision Tree. We also built a service for this model.



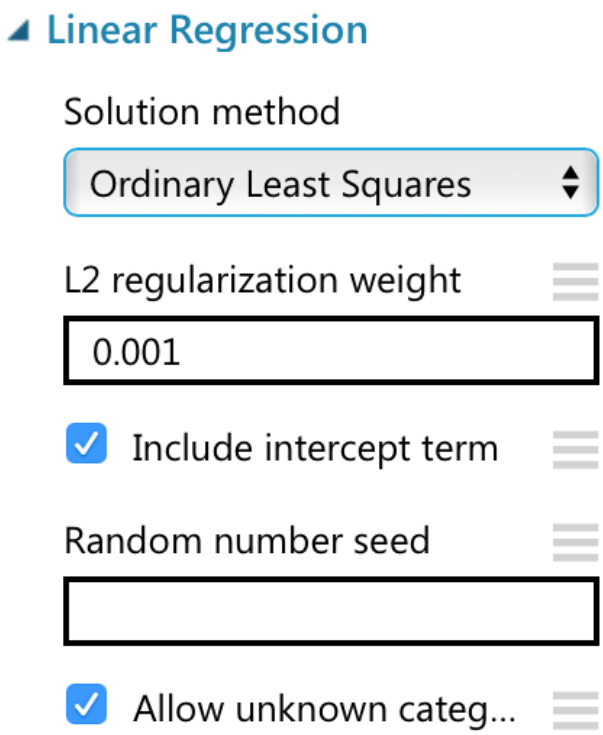
Model Hour-Base

For Uber manager to predict the usage of their base, we try to build a model to help them. We count trips according to the despatcher base. Inputs of the model are weather, date time, work day and number of despatcher base, the output is trips of this hour. We also used the four regression algorithm to train model.

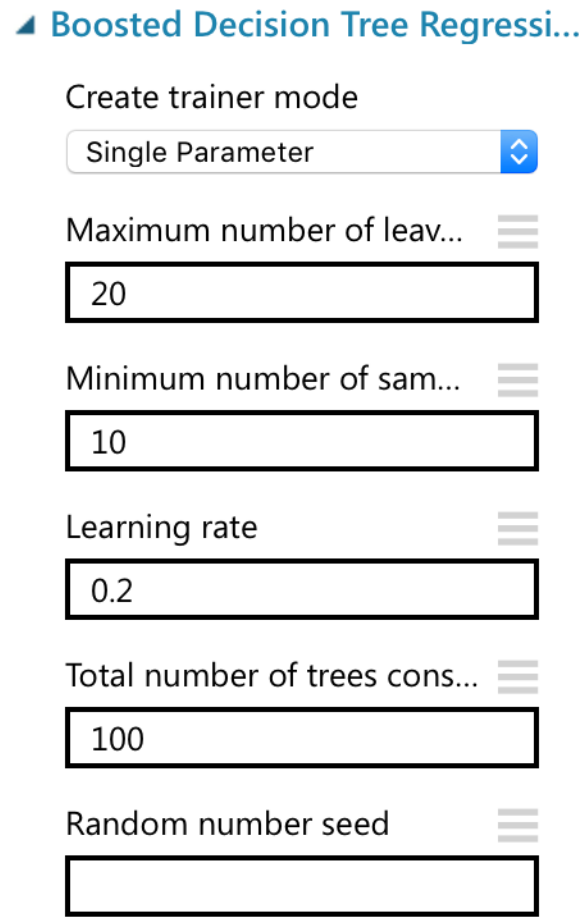


Algorithm Setting

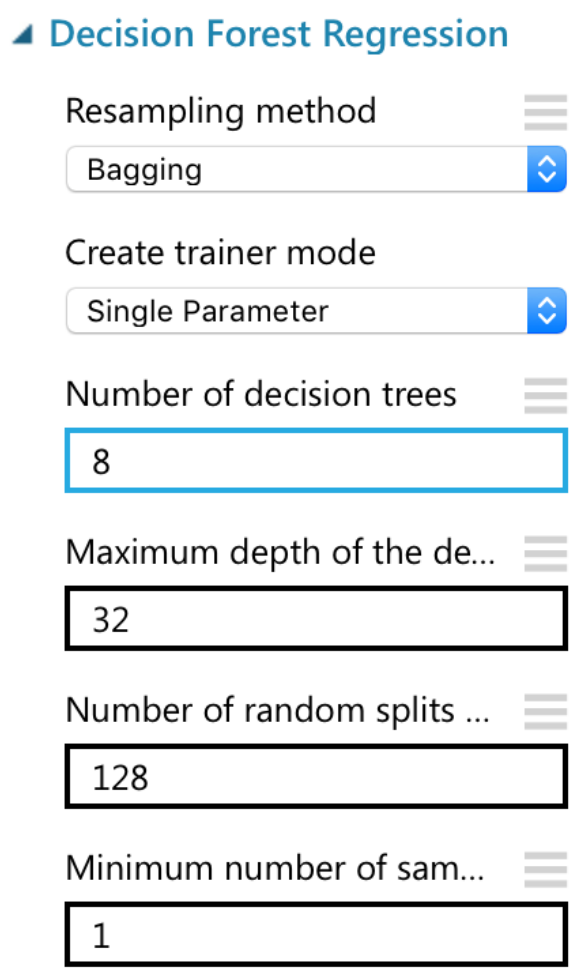
Linear Regression



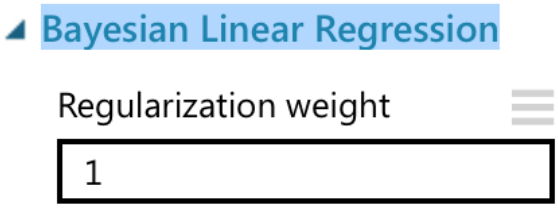
Boosted Decision Tree



Decision Forest Tree



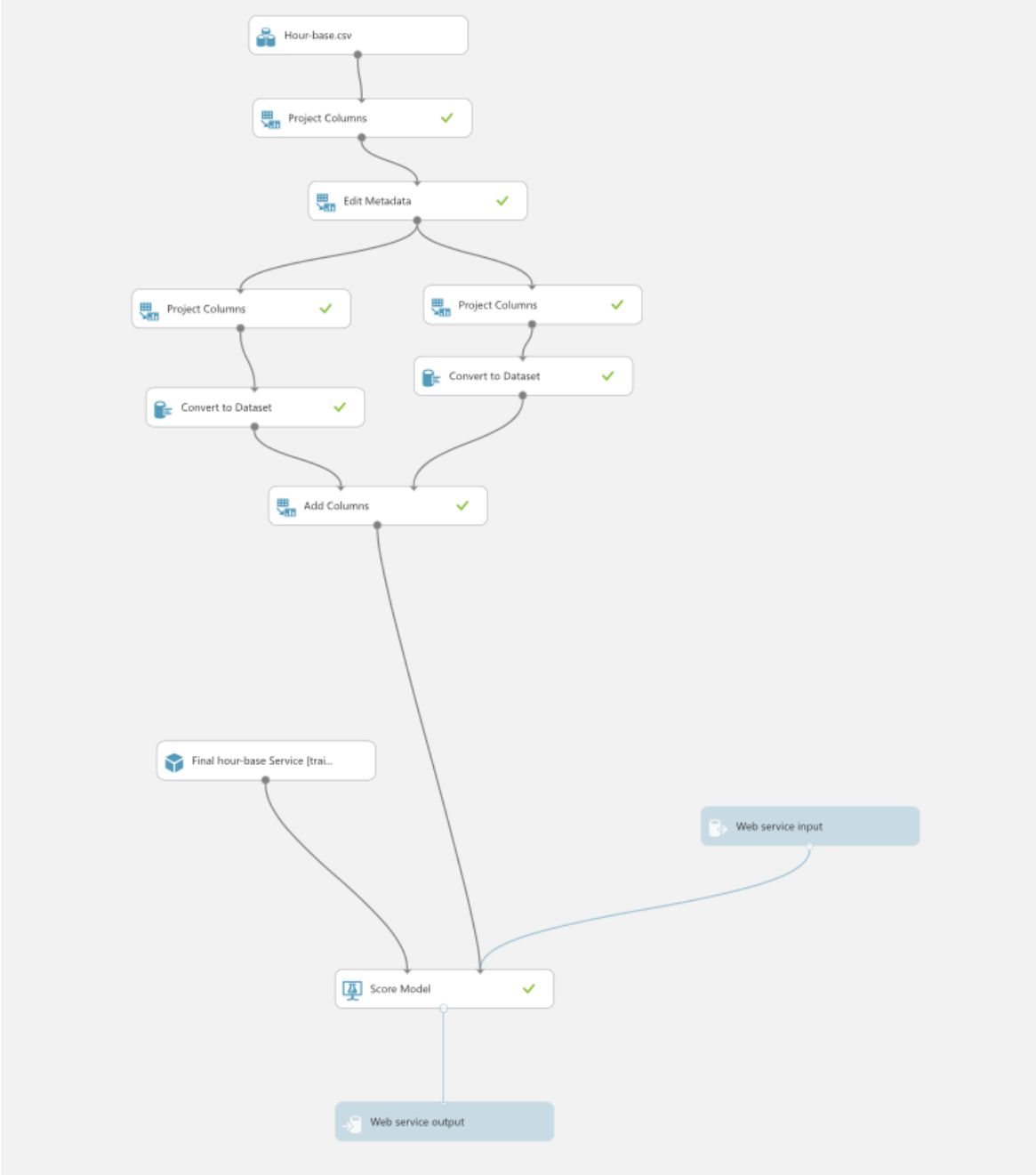
Bayesian Linear Regression



Evaluation

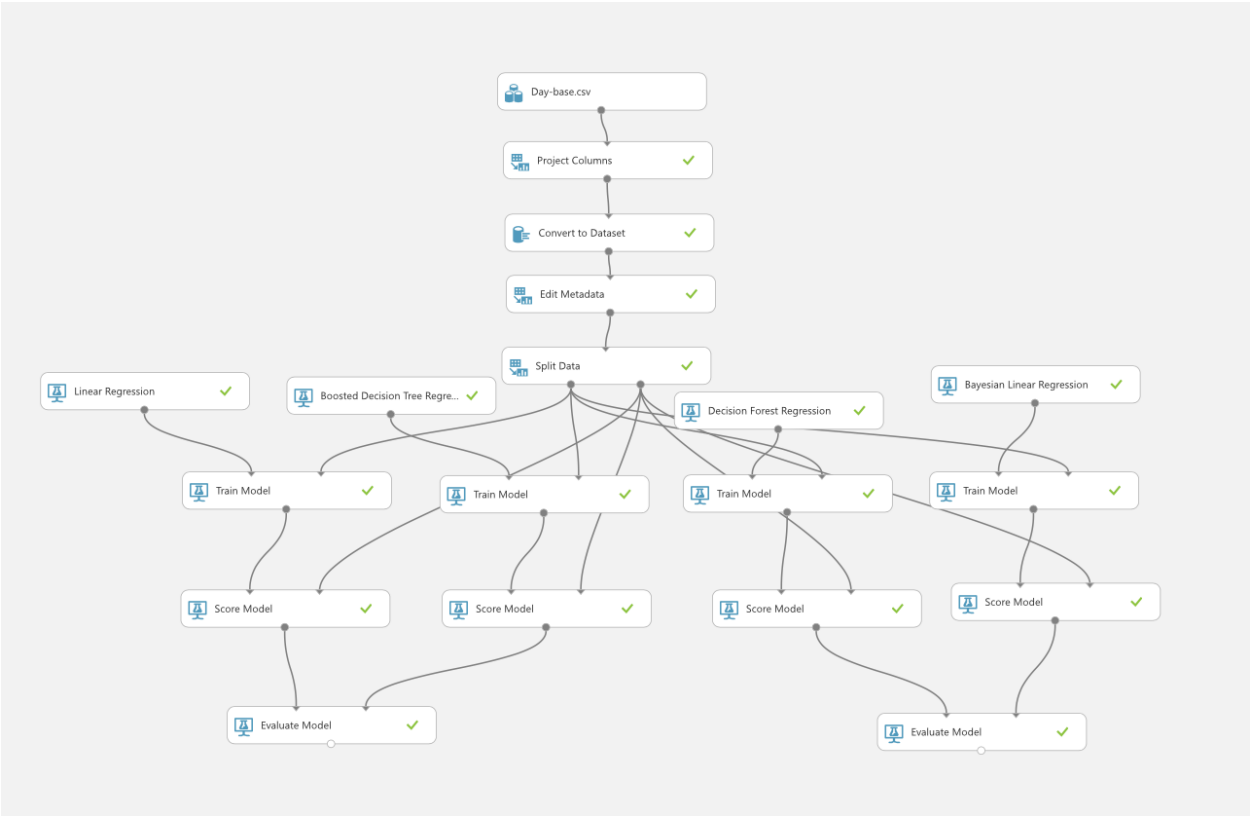
	Linear Regression	Boosted Decision Tree	Decision Forest Tree	Bayesian Linear Regression
Root Mean Squared Error	735.87	625.34	685.73	736.46

Compare the RMS we chose the Boosted Decision Tree algorithm to training our model and implement a service.



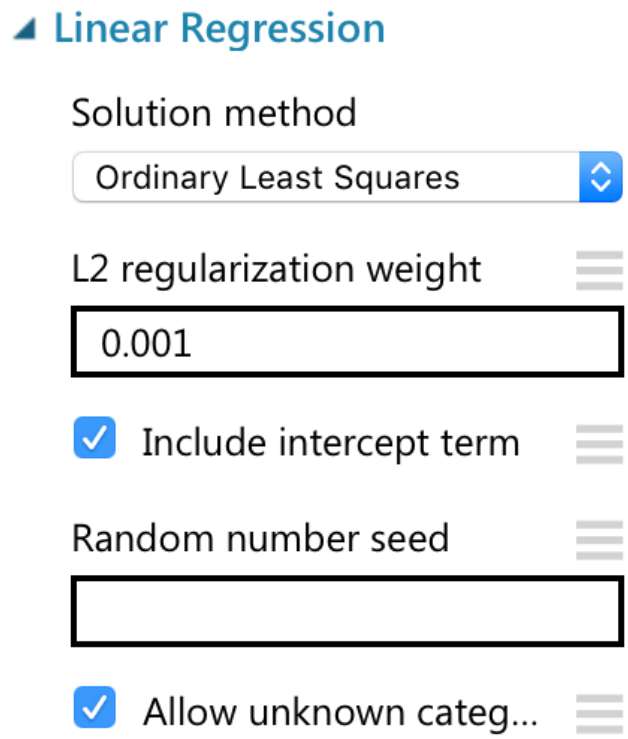
Model Day-Base

This model is use day scale to count the trip which we wish to give Uber business Macro-Prediction for the usage of Base.

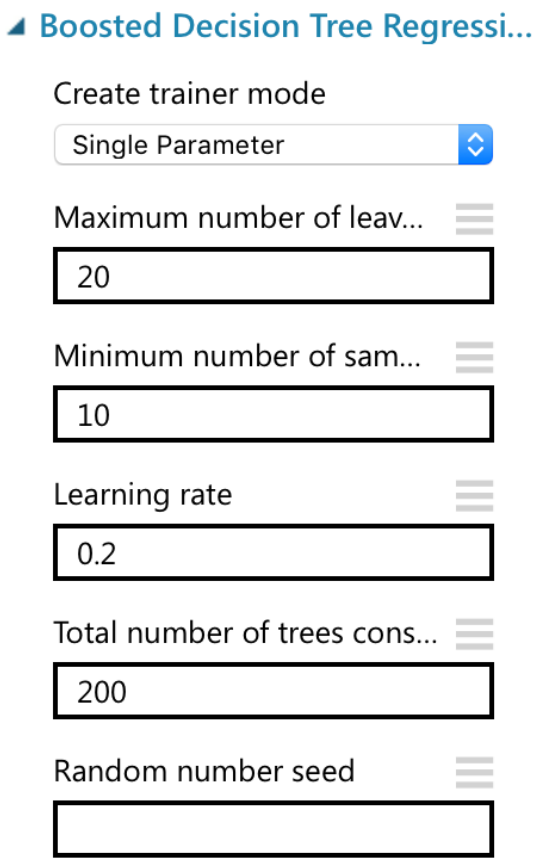


Algorithm Setting

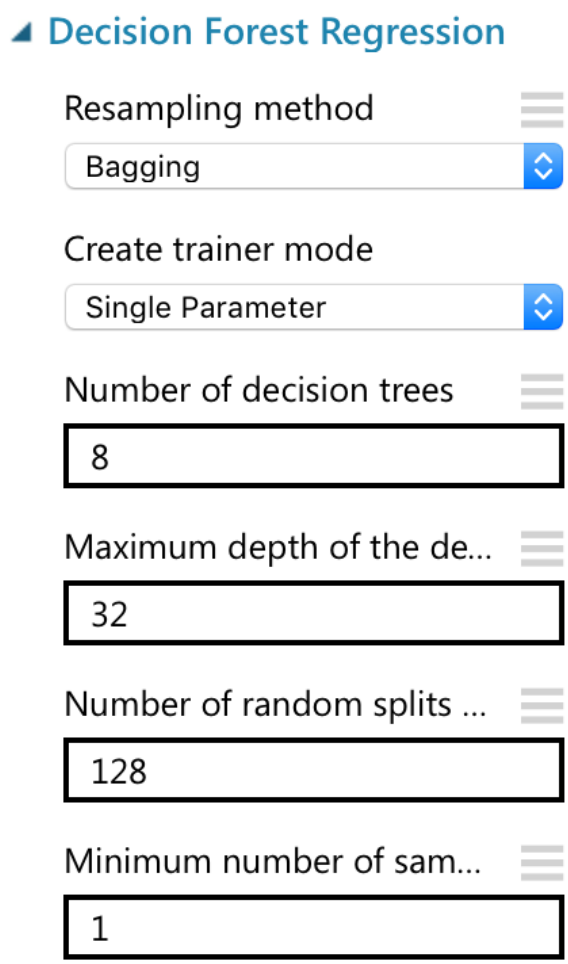
Linear Regression



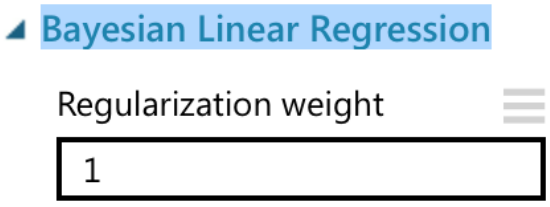
Boosted Decision Tree Regression



Decision Forest Regression



Bayesian Linear Regression

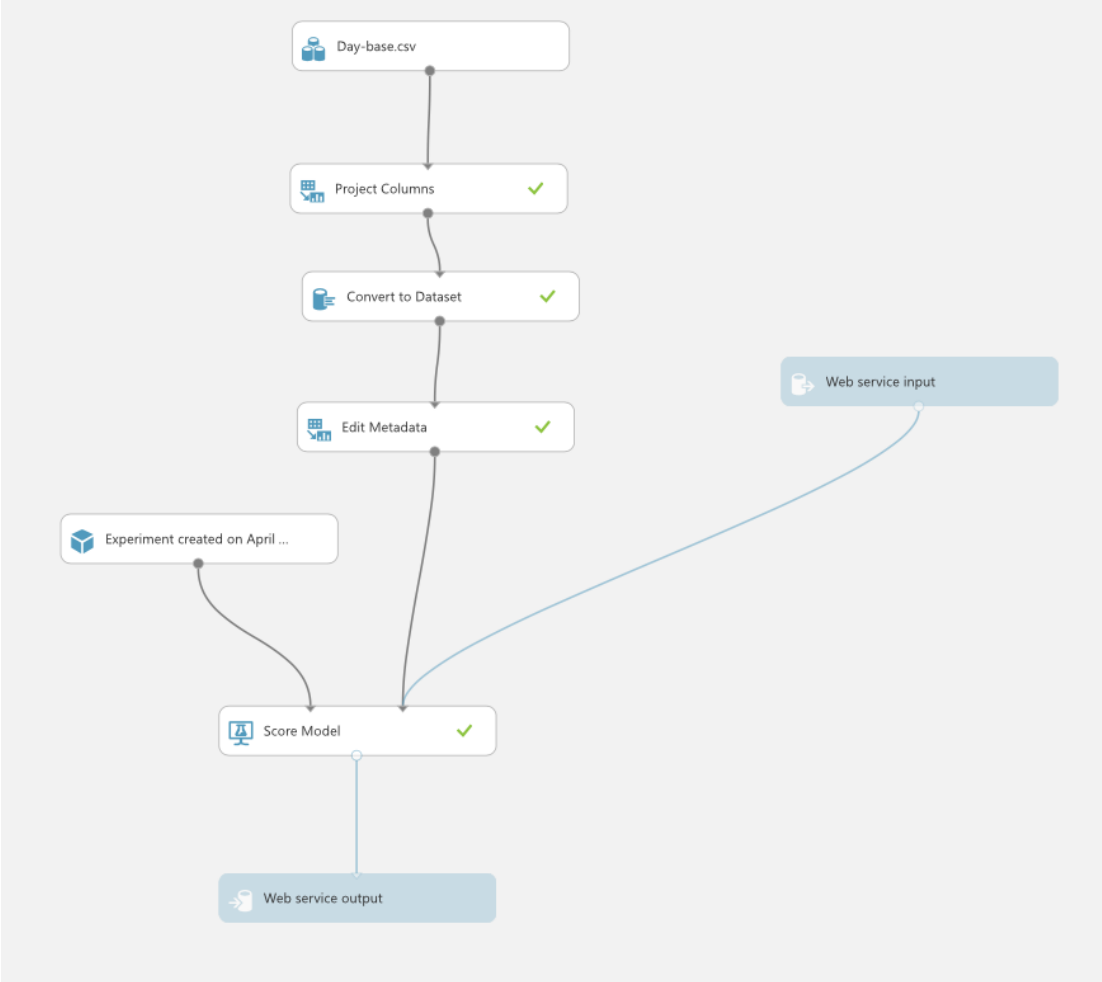


Evaluation

	Linear Regression	Boosted Decision Tree	Decision Forest Tree	Bayesian Linear Regression

Root Mean Squared Error	4647.20	2183.96	1682.73	3104.10

Compare with each model’ s RMS, we chose Decision Forest Tree as our final model. The service is like this.



Model Summary

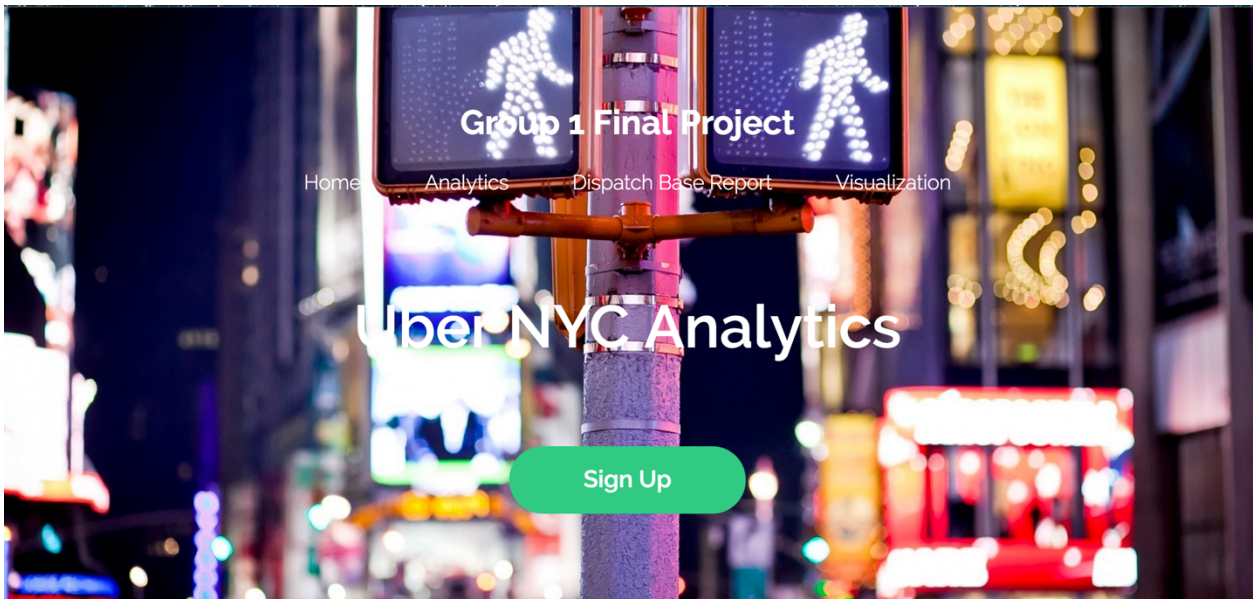
Depends on the time scale we wish to predict, we built 4 models for each scale that could much accurate than only build one model to predict different time scale. From the models, we also can see that, decision tree and decision forest tree have a better model than the other linear algorithm, that may mean our data is not a linear relationship.

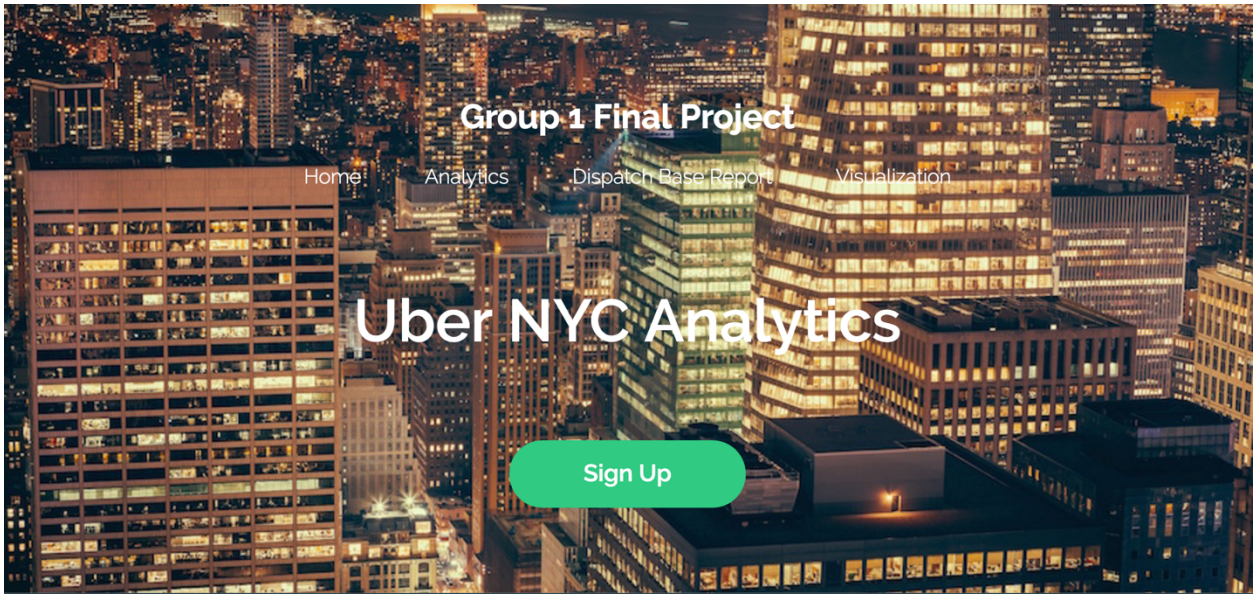
Web Service

4.1 The Web Architecture

We built the front end web page in Bootstrap framework. The web page used a form to collect the input information from users, then the form submit the input to the backend, which is implemented by Spring MVC. Spring MVC parsed the data into JSON format and then sent the data to the RESTful service. The Spring MVC also received the response from the Microsoft Azure, parsed the data and picked the output, then sent the output to the front page to users' side. The Spring MVC also received the Dynamic RESTful Weather Data API from openweathermap. At last, we deployed the website on AWS BeanStalk.

4.2 Front End





We use Bootstrap to create front end, the visualization tag is the link to the tableau page. And we use a simple form to collect the user's input.

4.3 Spring MVC

We used Spring MVC to parse the user's input data into JSON format. And then sent the request to the Azure model. The response will be sent back in JSON format. We extract the data we need and presented on the front end.

4.3.1 Set the header for building Connections

According to the API document, we have to add header to the request.

```
HttpPost createConnectivity(String restUrl)
{
    HttpPost post = new HttpPost(restUrl);
//      String auth=new StringBuffer(username).append(":").append(password).toString();
//      byte[] encodedAuth = Base64.encodeBase64(auth.getBytes(Charset.forName("US-ASCII")));
    String authHeader = "Bearer " + "VEu4ZU1S1ndCerZ54HQioAANthTtsJe8xImSJAn377OV9jjfAoPJf8BlS0ENU3oRRtX+Esuh-
    post.setHeader("AUTHORIZATION", authHeader);
    post.setHeader("Content-Type", "application/json");
        post.setHeader("Accept", "application/json");
        post.setHeader("X-Stream" , "true");
    return post;
}
```

Firstly, we set the post's header by setting parameters and connect the author Header to our API. The format of our data are setting in JSON.

```
@SuppressWarnings("finally")
String executeReq(String jsonData, HttpPost httpPost)
{
    String predict = null;
    try{
         predict = executeHttpRequest(jsonData, httpPost);

    }
    catch (UnsupportedEncodingException e){
        System.out.println("error while encoding api url : "+e);
    }
    catch (IOException e){
        System.out.println("ioException occured while sending http request : "+e);
    }
    catch(Exception e){
        System.out.println("exception occured while sending http request : "+e);
    }
    finally{
        httpPost.releaseConnection();
        return predict;
    }
}
```

```
String executeHttpRequest(String jsonData,  HttpPost httpPost)  throws UnsupportedEncodingException, IOE
{
    HttpResponse response=null;
    String line = "";
    StringBuffer result = new StringBuffer();
    httpPost.setEntity(new StringEntity(jsonData));
    HttpClient client = HttpClientBuilder.create().build();
    response = client.execute(httpPost);
    System.out.println("Post parameters : " + jsonData );
    System.out.println("Response Code : " + response.getStatusLine().getStatusCode());
/       response.getEntity().getContent()
    BufferedReader reader = new BufferedReader(new InputStreamReader(response.getEntity().getContent()))
    while ((line = reader.readLine()) != null){ result.append(line); }
    System.out.println(result.toString());
```

4.3.2 Executing process.

In the Home Controller, we firstly direct get the data from front page and transform it into JSON format.

```
public ModelAndView addInput (HttpServletRequest req, HttpServletResponse res) {

    ModelAndView mv= new ModelAndView();
    String restUrl="https://ussouthcentral.services.azureml.net/workspaces/ad7eba57cb134c21be182a9ac192
/     JSONObject user=new JSONObject();
    JSONObject user=new JSONObject();
    JSONObject inputa=new JSONObject();
/   JSONObject user=new JSONObject();

    System.out.println("inside homecontroller addinput");
```

Since we have already define the Http post method, what we need to do next is to send the data into Azure through our API key. Then print the value of its output.

```
            System.out.println("test");
//            String jsonData=user.toString();
            HttpPostReq httpPostReq=new HttpPostReq();
            HttpPost httpPost = httpPostReq.createConnectivity(restUrl);
            String predictOutput = httpPostReq.executeReq(userInput, httpPost);
            System.out.println("this is in the home controller -> "+predictOutput);
//            return null;
            mv.addObject("output", predictOutput);
            mv.setViewName("try2");
            return mv;
        }
```