Unit-V

1.Overview of R Shiny

2.R Hadoop

3.Case Study - Hypothesis Generation, Importing Data set and Basic Data Exploration, Feature Engineering, Model Building**.**

1. Overview of R Shiny

Writing codes for plotting graphs in R time & again can get very tiring. Also, it is very difficult to create an interactive visualization for story narration using above packages. These problems can be resolved by dynamically creating interactive plots in R using Shiny with minimal effort.

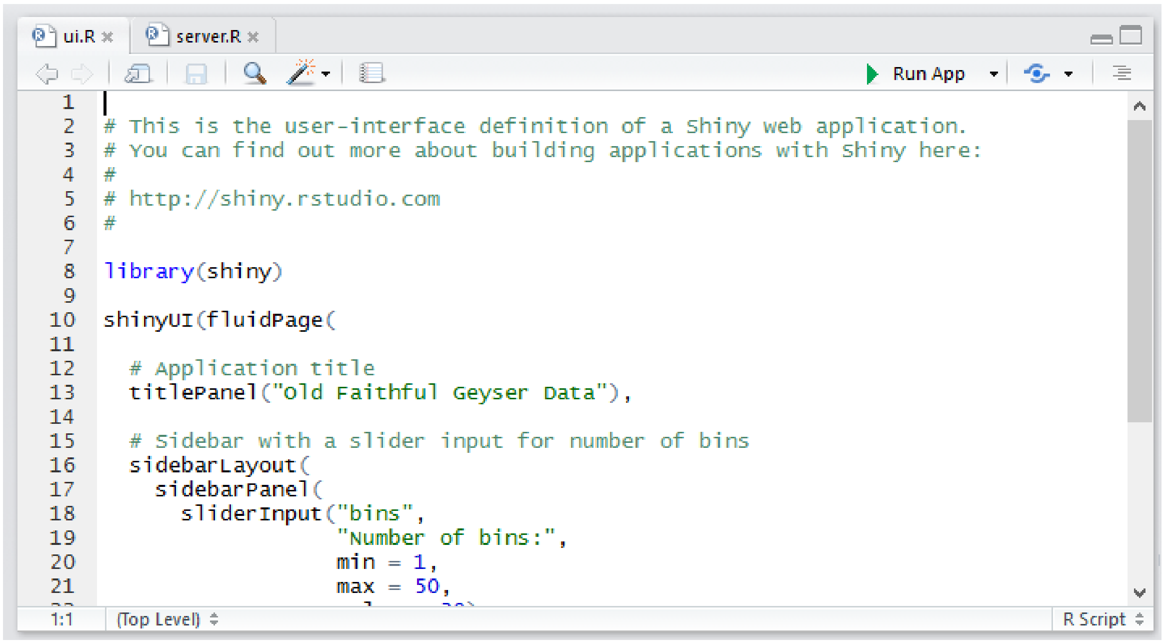
If you use R, chances are that you might have come across Shiny. It is an open package from RStudio, used to build interactive web pages with R. It provides a very powerful way to share your analysis in an interactive manner with the community.

Shiny is an open package from RStudio, which provides a web application framework to create interactive web applications (visualization) called "Shiny apps". The ease of working with Shiny has what popularized it among R users. These web applications seamlessly display R objects (like plots, tables etc.) and can also be made live to allow access to anyone.

Shiny provides automatic reactive binding between inputs and outputs which we will be discussing in the later parts of this article. It also provides extensive pre-built widgets which make it possible to build elegant and powerful applications with minimal effort.

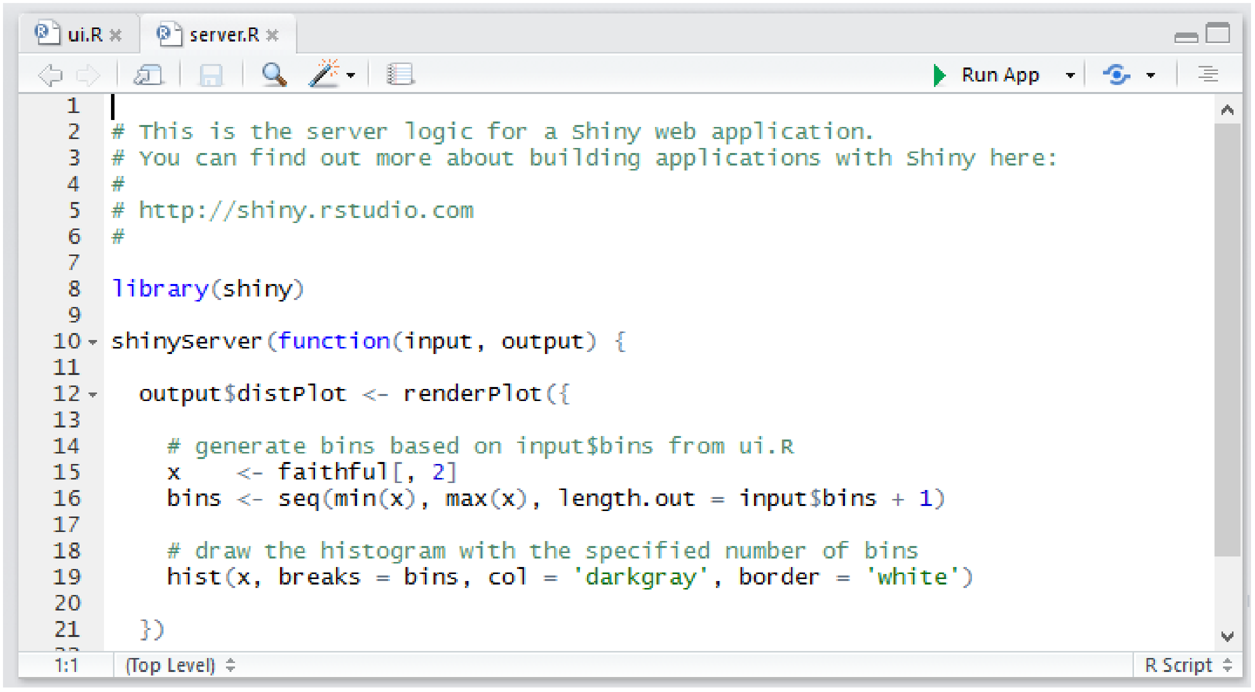
Components of R Shiny

1.UI.R: This file creates the user interface in a shiny application. It provides interactivity to the shiny app by taking the input from the user and dynamically displaying the generated output on the screen.



```r

# This is the user-interface definition of a Shiny web application.
# You can find out more about building applications with Shiny here:
#
# http://shiny.rstudio.com
#

library(shiny)

shinyUI(fluidPage(

  # Application title
  titlePanel("Old Faithful Geyser Data"),

  # Sidebar with a slider input for number of bins
  sidebarLayout(
    sidebarPanel(
      sliderInput("bins",
                  "Number of bins:",
                  min = 1,
                  max = 50,
```

2. Server.R: This file contains the series of steps to convert the input given by user into the desired output to be displayed.

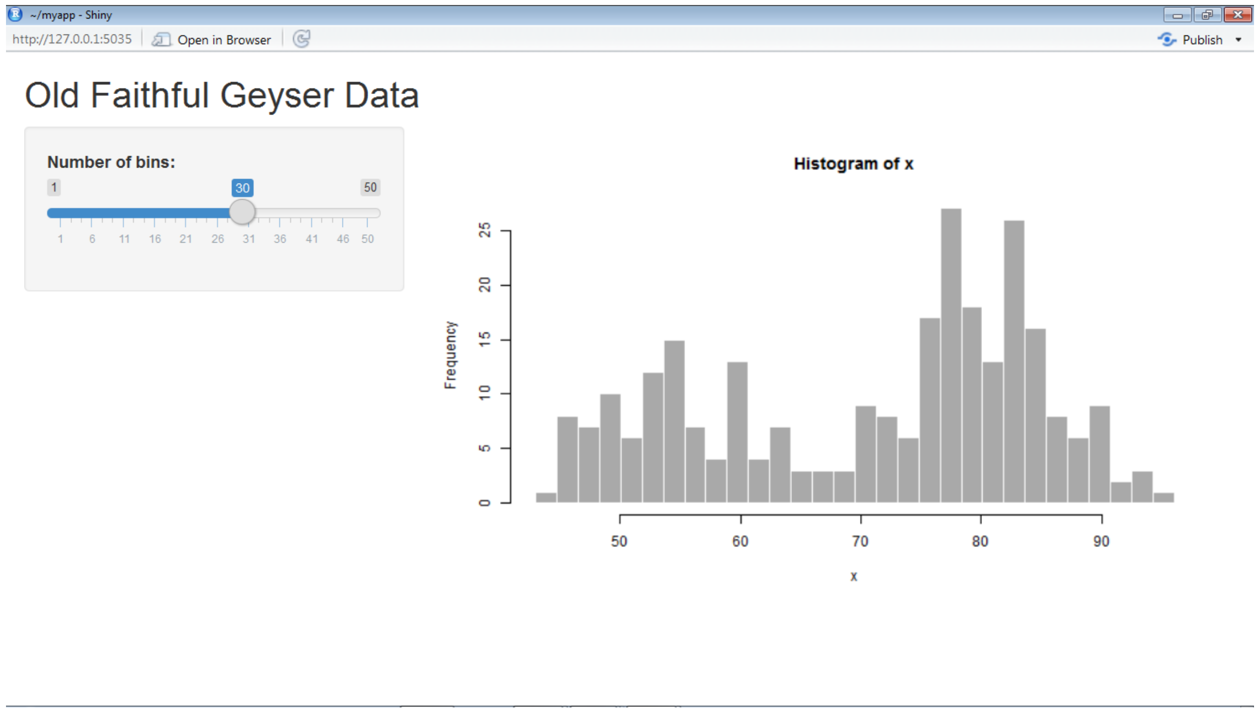


```r

# This is the server logic for a Shiny web application.
# You can find out more about building applications with Shiny here:
#
# http://shiny.rstudio.com
#

library(shiny)

shinyServer(function(input, output) {

  output$distPlot <- renderPlot({

    # generate bins based on input$bins from ui.R
    x    <- faithful[, 2]
    bins <- seq(min(x), max(x), length.out = input$bins + 1)

    # draw the histogram with the specified number of bins
    hist(x, breaks = bins, col = 'darkgray', border = 'white')

  })
```

Creation of Simple RShiny Application

Writing "ui.R"

If you are creating a shiny application, the best way to ensure that the application interface runs smoothly on different devices with different screen resolutions is to create it using fluid page. This ensures that the page is laid out dynamically based on the resolution of each device.



The user interface can be broadly divided into three categories:

- **Title Panel:** The content in the title panel is displayed as metadata, as in top left corner of above image which generally provides name of the application and some other relevant information.

- **Sidebar Layout:** Sidebar layout takes input from the user in various forms like text input, checkbox input, radio button input, drop down input, etc. It is represented in dark background in left section of the above image.

- **Main Panel:** It is part of screen where the output(s) generated as a result of performing a set of operations on input(s) at the server.R is / are displayed.

Let’s understand UI.R and Server.R with an example:

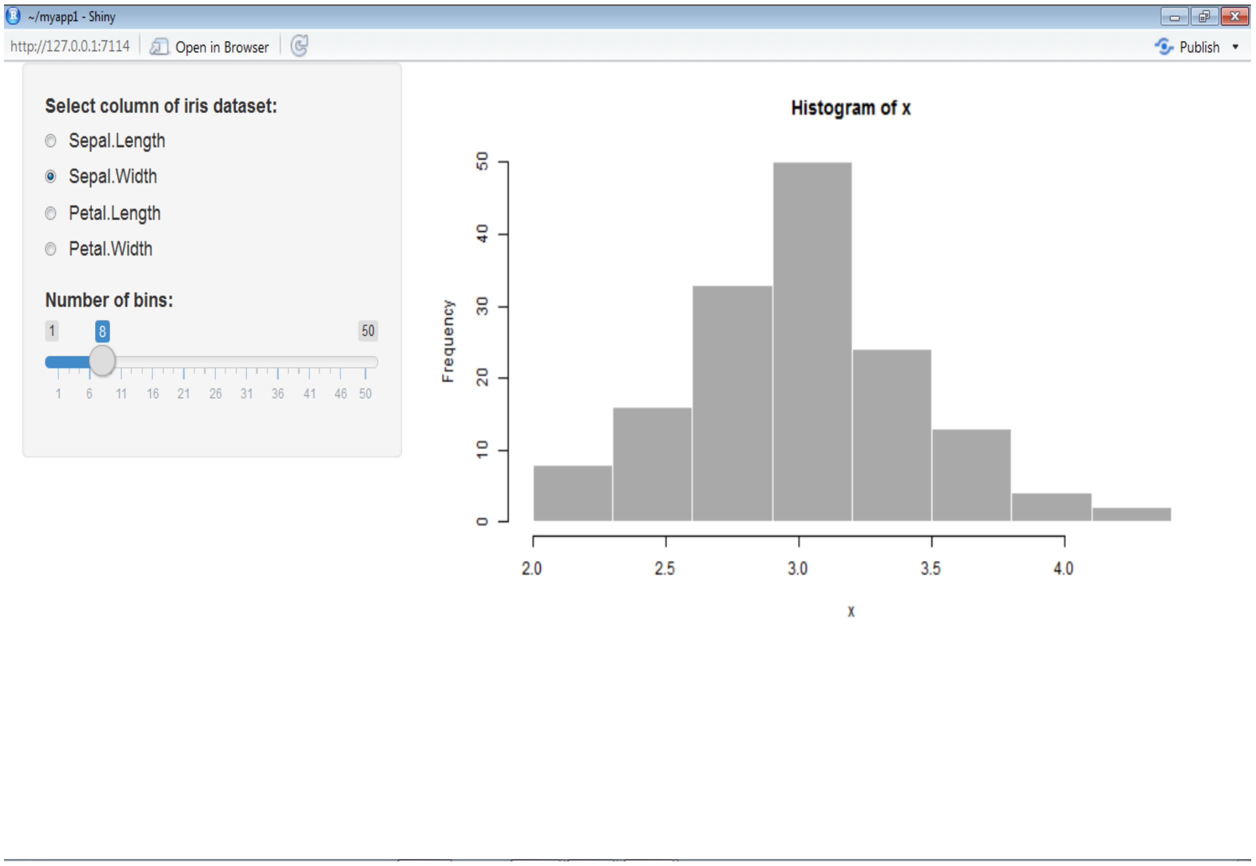
```
#UI.R

#loading shiny library

library(shiny)


shinyUI(fluidPage(

#fluid page for dynamically adapting to screens of different resolutions.

 titlePanel("Iris Dataset"),

 sidebarLayout(

 sidebarPanel  (

     #implementing radio buttons

     radioButtons("p", "Select column of iris dataset:",

                 list("Sepal.Length"='a', "Sepal.Width"='b', "Petal.Length"='c', "Petal.Width"='d')),

                #slider input for bins of histogram

                sliderInput("bins",  "Number of bins:",  min = 1, max = 50, value = 30)
   ),

 mainPanel(plotOutput("distPlot")))

   ))
```

Writing SERVER.R

This acts as the brain of web application. The server.R is written in the form of a function which maps input(s) to the output(s) by some set of logical operations. The inputs taken in ui.R file are accessed using $ operator (input$InputName). The outputs are also referred using the $ operator (output$OutputName). We will be discussing a few examples of server.R in the coming sections of the article for better understanding.

```
#SERVER.R
library(shiny)

#writing server function
shinyServer(function(input, output) {

#referring output distPlot in ui.r as output$distPlot
  output$distPlot <- renderPlot({

#referring input p in ui.r as input$p
    if(input$p=='a'){
      i<-1
    }

    if(input$p=='b'){
      i<-2
    }

    if(input$p=='c'){
      i<-3
    }

    if(input$p=='d'){
      i<-4
    }

    x    <- iris[, i]

#referring input bins in ui.r as input$bins
    bins <- seq(min(x), max(x), length.out = input$bins + 1)

#producing histogram as output
    hist(x, breaks = bins, col = 'darkgray', border = 'white')
  })

})
```

Output:

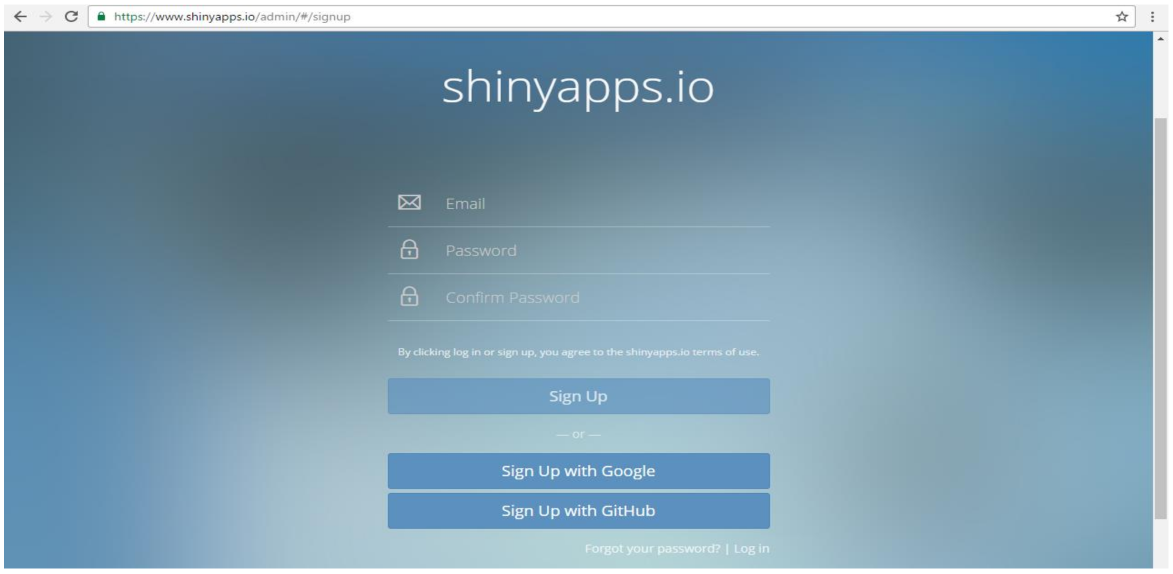


Deploying the Shiny app on the Web

The shiny apps which you have created can be accessed and used by anyone only if, it is deployed on the web. You can host your shiny application on “Shinyapps.io”. It provides free of cost platform as a service [PaaS] for deployment of shiny apps, with some restrictions though like only 25 hours of usage in a month, limited memory space, etc. You can also use your own server for deploying shiny apps.

Steps for using shiny cloud:

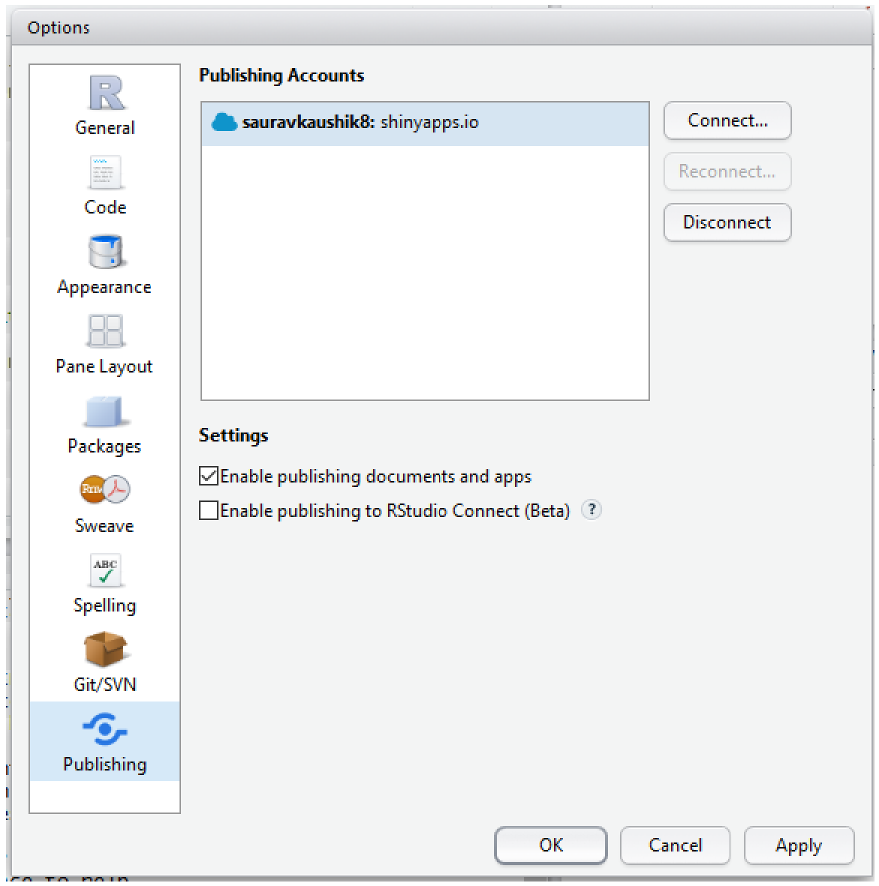
Step 1: Sign up on shinyapps.io


https://www.shinyapps.io/admin/#/signup
shinyapps.io
Email
Password
Confirm Password
By clicking log in or sign up, you agree to the shinyapps.io terms of use.
Sign Up
— or —
Sign Up with Google
Sign Up with GitHub
Forgot your password? | Log in

Step 2: Go to Tools in R Studio.

Step 3: Open global options.

Step 4: Open publishing tab



Publishing Accounts

sauravkaushik8: shinyapps.io

Settings

[✓] Enable publishing documents and apps

[ ] Enable publishing to RStudio Connect (Beta)

Step 5: Manage your account(s).

Shiny App for displaying summary of the given data set:

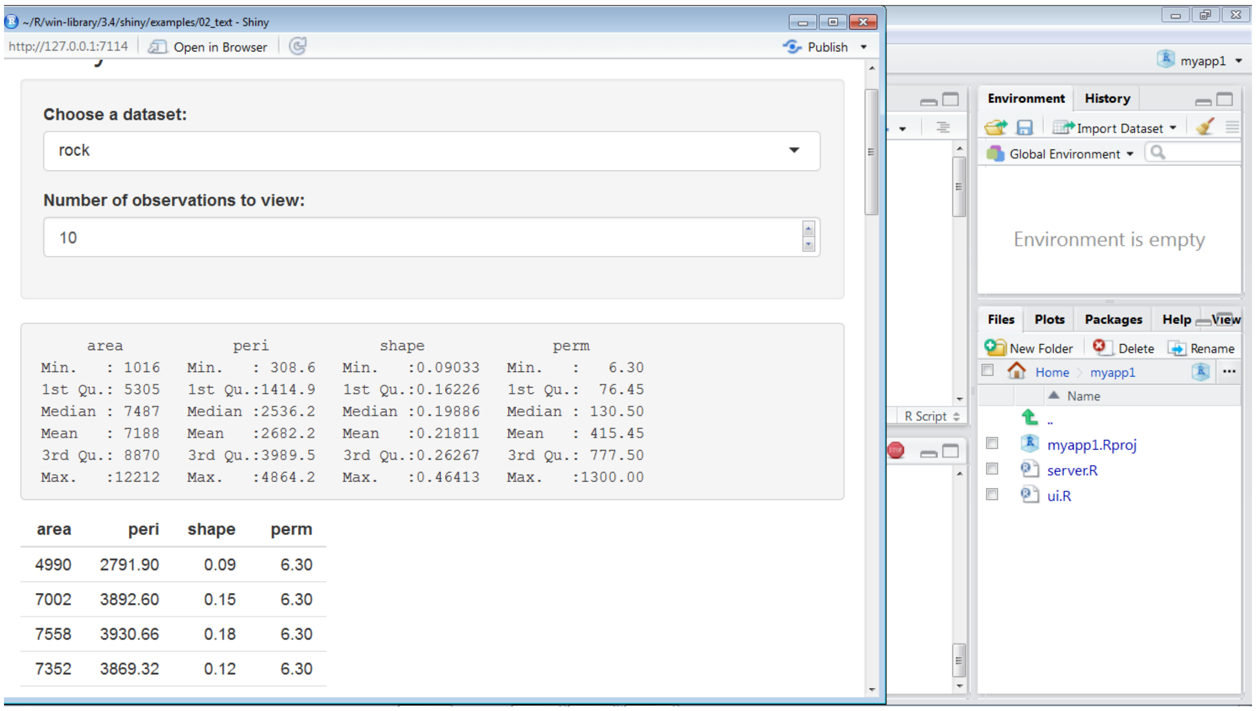
Server.R

```
library(shiny)
library(datasets)

# Define server logic required to summarize and view the selected dataset
shinyServer(function(input, output) {

  # Return the requested dataset
  datasetInput <- reactive({
    switch(input$dataset,
           "rock" = rock,
           "pressure" = pressure,
           "cars" = cars)
  })

  # Generate a summary of the dataset
  output$summary <- renderPrint({
    dataset <- datasetInput()
    summary(dataset)
  })

  # Show the first "n" observations
  output$view <- renderTable({
    head(datasetInput(), n = input$obs)
  })
})
```

ui.R

```
library(shiny)

# Define UI for dataset viewer application
shinyUI(fluidPage(

  # Application title
  titlePanel("Shiny Text"),

  # Sidebar with controls to select a dataset and specify the number
  # of observations to view
  sidebarLayout(
    sidebarPanel(
      selectInput("dataset", "Choose a dataset:",
                  choices = c("rock", "pressure", "cars")),

      numericInput("obs", "Number of observations to view:", 10)
    ),

    # Show a summary of the dataset and an HTML table with the requested
    # number of observations
    mainPanel(
      verbatimTextOutput("summary"),

      tableOutput("view")
    )
  )
))
```

Output:



Choose a dataset: rock

Number of observations to view: 10

```
     area            peri            shape              perm
Min.   : 1016   Min.   : 308.6   Min.   :0.09033   Min.   :   6.30
1st Qu.: 5305   1st Qu.:1414.9   1st Qu.:0.16226   1st Qu.:  76.45
Median : 7487   Median :2536.2   Median :0.19886   Median : 130.50
Mean   : 7188   Mean   :2682.2   Mean   :0.21811   Mean   : 415.45
3rd Qu.: 8870   3rd Qu.:3989.5   3rd Qu.:0.26267   3rd Qu.: 777.50
Max.   :12212   Max.   :4864.2   Max.   :0.46413   Max.   :1300.00
```

area	peri	shape	perm
4990	2791.90	0.09	6.30
7002	3892.60	0.15	6.30
7558	3930.66	0.18	6.30
7352	3869.32	0.12	6.30

2. R HADOOP

R is an amazing data science programming tool to run statistical data analysis on models and translating the results of analysis into colourful graphics. There is no doubt that R is the most preferred programming tool for statisticians, data scientists, data analysts and data architects but it falls short when working with large datasets. One major drawback with R programming language is that all objects are loaded into the main memory of a single machine. Large datasets of size petabytes cannot be loaded into the RAM memory; this is when Hadoop integrated with R language, is an ideal solution. To adapt to the in-memory, single machine limitation of R programming language, data scientists have to limit their data analysis to a sample of data from the large data set. This limitation of R programming language comes as a major hindrance when dealing with big data. Since, R is not very scalable, the core R engine can process only limited amount of data.

To the contrary, distributed processing frameworks like Hadoop are scalable for complex operations and tasks on large datasets (petabyte range) but do not have strong statistical analytical capabilities. As Hadoop is a popular framework for big data processing, integrating R with Hadoop is the next logical step. Using R on Hadoop will provide highly scalable data analytics platform which can be scaled depending on the size of the dataset. Integrating Hadoop with R lets data scientists run R in parallel on large dataset as none of the data science libraries in R language will work on a dataset that is larger than its memory. Big Data analytics with R and Hadoop competes with the cost value return offered by commodity hardware cluster for vertical scaling.

Methods of Integrating R and Hadoop Together

Data analysts or data scientists working with Hadoop might have R packages or R scripts that they use for data processing. To use these R scripts or R packages with Hadoop, they need to rewrite these R scripts in Java programming language or any other language that implements Hadoop MapReduce. This is a burdensome process and could lead to unwanted errors. To integrate Hadoop with R programming language, we need to use a software that already is written for R language with the data being stored on the distributed storage Hadoop. There are many solutions for using R language to perform large computations but all these solutions require that the data be loaded into the memory before it is distributed to the computing nodes. This is not an ideal solution for large datasets. Here are some commonly used methods to integrate Hadoop with R to make the best use of the analytical capabilities of R for large datasets-

1) RHADOOP –Install R on Workstations and Connect to Data in Hadoop

The most commonly used open source analytics solution to integrate R programming language with Hadoop is RHadoop. RHadoop developed by Revolution Analytics lets users directly ingest data from HBase database subsystems and HDFS file systems. Rhadoop package is the ‘go-to’ solution for using R on Hadoop because of its simplicity and cost advantage. Rhadoop is a collection of 5 different packages which allows Hadoop users to manage and analyse data using R programming language. RHadoop package is compatible with open source Hadoop and as well with popular Hadoop distributions- Cloudera, Hortonworks and MapR.

rhbase – rhbase package provides database management functions for HBase within R using Thrift server. This package needs to be installed on the node that will run R client. Using rhbase, data engineers and data scientists can read, write and modify data stored in HBase tables from within R.

rhdfs –rhdfs package provides R programmers with connectivity to the Hadoop distributed file system so that they read, write or modify the data stored in Hadoop HDFS.

plyrmr – This package supports data manipulation operations on large datasets managed by Hadoop. Plyrmr (plyr for MapReduce) provides data manipulation operations present in popular packages like reshape2 and plyr. This package depends on Hadoop MapReduce to perform operations but abstracts most of the MapReduce details.

ravro –This package lets users read and write Avro files from local and HDFS file systems.

rmr2 (Execute R inside Hadoop MapReduce) – Using this package, R programmers can perform statistical analysis on the data stored in a Hadoop cluster. Using rmr2 might be a cumbersome process to integrate R with Hadoop but many R programmers find using rmr2 much easier than depending on Java based Hadoop mappers and reducers. rmr2 might be a little tedious but it eliminates data movement and helps parallelize computation to handle large datasets.

2) RHIPE – Execute R inside Hadoop Map Reduce

RHIPE (**"R and Hadoop Integrated Programming Environment")** is an R library that allows users to run Hadoop MapReduce jobs within R programming language. R programmers just have to write R map and R reduce functions and the RHIPE library will transfer them and invoke the corresponding Hadoop Map and Hadoop Reduce tasks. RHIPE uses a protocol buffer encoding scheme to transfer the map and reduce inputs. The advantage of using RHIPE over other parallel R packages is, that it integrates well with Hadoop and provides a data distribution scheme using HDFS across a cluster of machines - which provides fault tolerance and optimizes processor usage.

3) R and Hadoop Streaming

Hadoop Streaming API allows users to run Hadoop MapReduce jobs with any executable script that reads data from standard input and writes data to standard output as mapper or reducer. Thus, Hadoop Streaming API can be used along R programming scripts in the map or reduce phases. This method to integrate R, Hadoop does not require any client side integration because streaming jobs are launched through Hadoop command line. MapReduce jobs submitted undergo data transformation through UNIX standard streams and serialization to ensure Java complaint input to Hadoop, irrespective of the language of the input script provided by the programmer.

4) RHIVE –Install R on Workstations and Connect to Data in Hadoop

If you want your Hive queries to be launched from R interface then RHIVE is the go-to package with functions for retrieving metadata like database names, column names, and table names from Apache Hive. RHIVE provides rich statistical libraries and algorithms available in R programming language to the data stored in Hadoop by extending HiveQL with R language functions. RHIVE functions allow users to apply R statistical learning models to the data stored in Hadoop cluster that has been catalogued using Apache Hive. The advantage of using RHIVE for Hadoop R integration is that it parallelizes operations and avoids data movement because data operations are pushed down into Hadoop.

5) ORCH – Oracle Connector for Hadoop

ORCH can be used on non-oracle Hadoop clusters or on any other Oracle big appliance. Mappers and Reducers are written in R programming language and MapReduce jobs are executed from the R environments through a high level interface. With ORCH for R Hadoop integration, R programmers do not have to learn a new programming language like Java for getting into the details of Hadoop environment like Hadoop Cluster hardware or software. ORCH connector also allows users to test the ability of Map Reduce programs locally, through the same function call, much before they are deployed to the Hadoop cluster.

The number of open source options for performing big data analytics with R and Hadoop is continuously expanding but for simple Hadoop MapReduce jobs, R and Hadoop Streaming still proves to be the best solution. The combination of R and Hadoop together is a must have toolkit for professionals working with big data to create fast, predictive analytics combined with performance, scalability and flexibility you need.

Most Hadoop users claim that the advantage of using R programming language is its exhaustive list of data science libraries for statistics and data visualization. However, the data science libraries in R language are non-distributed in nature which makes data retrieval a time consuming affair. However, this is an in-built limitation of R programming language, but if we just ignore it, then R and Hadoop together can make big data analytics an ecstasy!

3.Case Study

Data set used: https://archive.ics.uci.edu/ml/machine-learning-databases/00275/

Step 1. Hypothesis Generation

Before exploring the data to understand the relationship between variables, I'd recommend you to focus on hypothesis generation first. Now, this might sound counter-intuitive for solving a data science problem. Before exploring data, think about the business problem, gain the domain knowledge.

How does it help? This practice usually helps you form better features later on, which are not biased by the data available in the dataset. At this stage, you are expected to posses structured thinking i.e. a thinking process which takes into consideration all the possible aspects of a particular problem.

Here are some of the hypothesis which I thought could influence the demand of bikes:

Hourly trend: There must be high demand during office timings. Early morning and late evening can have different trend (cyclist) and low demand during 10:00 pm to 4:00 am.

Daily Trend: Registered users demand more bike on weekdays as compared to weekend or holiday.

Rain: The demand of bikes will be lower on a rainy day as compared to a sunny day. Similarly, higher humidity will cause to lower the demand and vice versa.

Temperature: In India, temperature has negative correlation with bike demand. But, after looking at Washington's temperature graph, I presume it may have positive correlation.

Pollution: If the pollution level in a city starts soaring, people may start using Bike (it may be influenced by government / company policies or increased awareness).

Time: Total demand should have higher contribution of registered user as compared to casual because registered user base would increase over time.

Traffic: It can be positively correlated with Bike demand. Higher traffic may force people to use bike as compared to other road transport medium like car, taxi etc

Step 2. Understanding the Data Set

The dataset shows hourly rental data for two years (2011 and 2012). The training data set is for the first 19 days of each month. The test dataset is from 20th day to month's end. We are required to predict the total count of bikes rented during each hour covered by the test set.

In the training data set, they have separately given bike demand by registered, casual users and sum of both is given as count.

Training data set has 12 variables (see below) and Test has 9 (excluding registered, casual and count).

Independent Variables

datetime: date and hour in "mm/dd/yyyy hh:mm" format

season: Four categories-> 1 = spring, 2 = summer, 3 = fall, 4 = winter

holiday: whether the day is a holiday or not (1/0)

workingday: whether the day is neither a weekend nor holiday (1/0)

weather: Four Categories of weather

1-> Clear, Few clouds, Partly cloudy, Partly cloudy

2-> Mist + Cloudy, Mist + Broken clouds, Mist + Few clouds, Mist

3-> Light Snow and Rain + Thunderstorm + Scattered clouds, Light Rain + Scattered clouds

4-> Heavy Rain + Ice Pallets + Thunderstorm + Mist, Snow + Fog

temp: hourly temperature in Celsius

atemp: "feels like" temperature in Celsius

humidity: relative humidity

windspeed: wind speed

Dependent Variables

registered: number of registered user

casual: number of non-registered user

count: number of total rentals (registered + casual)

3. Importing Data set and Basic Data Exploration

For this solution, R (R Studio 0.99.442) in Windows Environment has been used.

Below are the steps to import and perform data exploration.

1. **Import Train and Test Data Set**

```
setwd("E:/kaggle data/bike sharing")

train=read.csv("train_bike.csv")

test=read.csv("test_bike.csv")
```

2. **Combine both Train and Test Data set (to understand the distribution of independent variable together).**

```
test$registered=0

test$casual=0

test$count=0

data=rbind(train,test)
```

Before combing test and train data set, I have made the structure similar for both.

3. **Variable Type Identification**

str(data)

```
'data.frame':  17379 obs. of  12 variables:

$ datetime  : Factor w/ 17379 levels "2011-01-01 00:00:00",..: 1 2 3 4 5 6 7 8 9 10
...

$ season    : int  1 1 1 1 1 1 1 1 1 1 ...

$ holiday   : int  0 0 0 0 0 0 0 0 0 0 ...

$ workingday: int  0 0 0 0 0 0 0 0 0 0 ...

$ weather   : int  1 1 1 1 1 2 1 1 1 1 ...

$ temp      : num  9.84 9.02 9.02 9.84 9.84 ...

$ atemp     : num  14.4 13.6 13.6 14.4 14.4 ...

$ humidity  : int  81 80 80 75 75 75 80 86 75 76 ...
$ windspeed : num  0 0 0 0 0 ...
$ casual    : num  3 8 5 3 0 0 2 1 1 8 ...
$ registered: num  13 32 27 10 1 1 0 2 7 6 ...
$ count     : num  16 40 32 13 1 1 2 3 8 14 ...
```

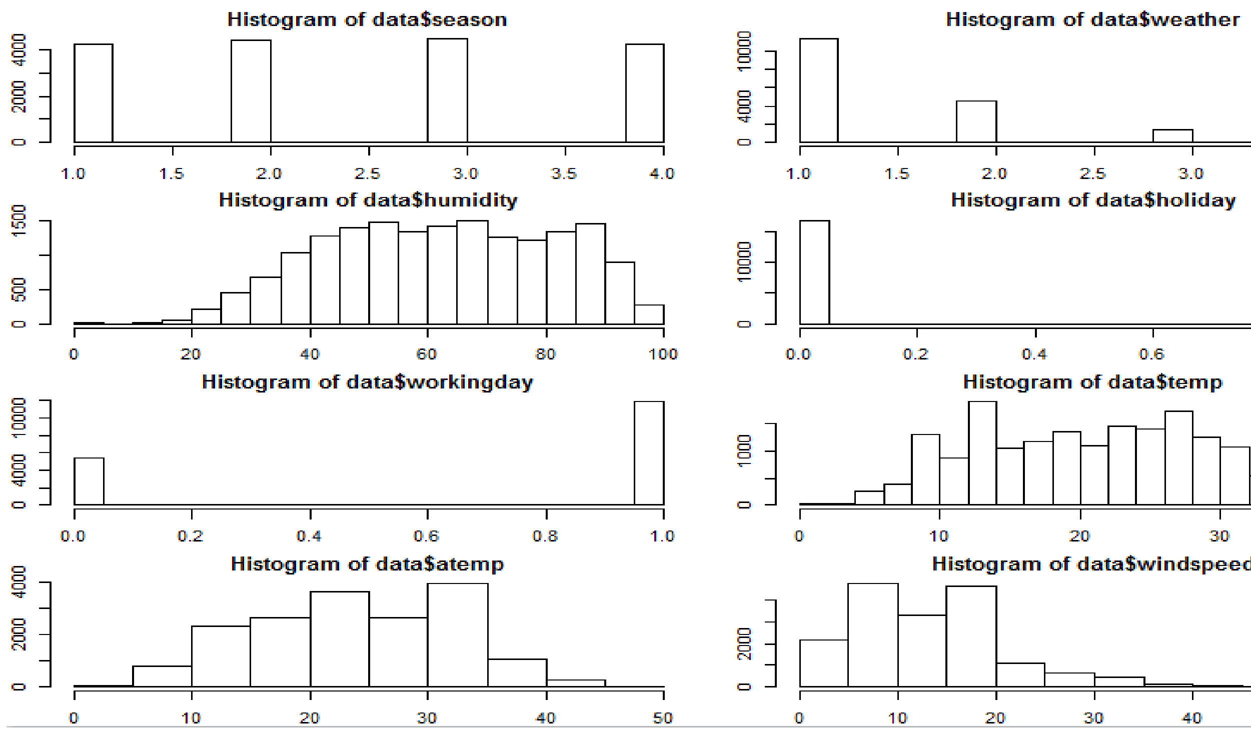
Find missing values in data set if any.

```
table(is.na(data))
 FALSE
208548
```

Above you can see that it has returned no missing values in the data frame.

4. **Understand the distribution of numerical variables and generate a frequency table for numeric variables. Now, I'll test and plot a histogram for each numerical variables and analyze the distribution.**

```
par(mfrow=c(4,2))
par(mar = rep(2, 4))
hist(data$season)
hist(data$weather)
hist(data$humidity)
hist(data$holiday)
hist(data$workingday)
hist(data$temp)
hist(data$atemp)
hist(data$windspeed)
```



Few inferences can be drawn by looking at the these histograms:

- Season has four categories of almost equal distribution
- Weather 1 has higher contribution i.e. mostly clear weather.

prop.table(table(data$weather))

```
1    2     3     4

0.66  0.26   0.08   0.00
```

As expected, mostly working days and variable holiday is also showing a similar inference. You can use the code above to look at the distribution in detail. Here you can generate a variable for weekday using holiday and working day. Incase, if both have zero values, then it must be a working day.Variables temp, atemp, humidity and windspeed looks naturally distributed.

5. **Convert discrete variables into factor (season, weather, holiday, workingday)**

```
data$season=as.factor(data$season)

data$weather=as.factor(data$weather)

data$holiday=as.factor(data$holiday)

data$workingday=as.factor(data$workingday)
```

4. Hypothesis Testing (using multivariate analysis)

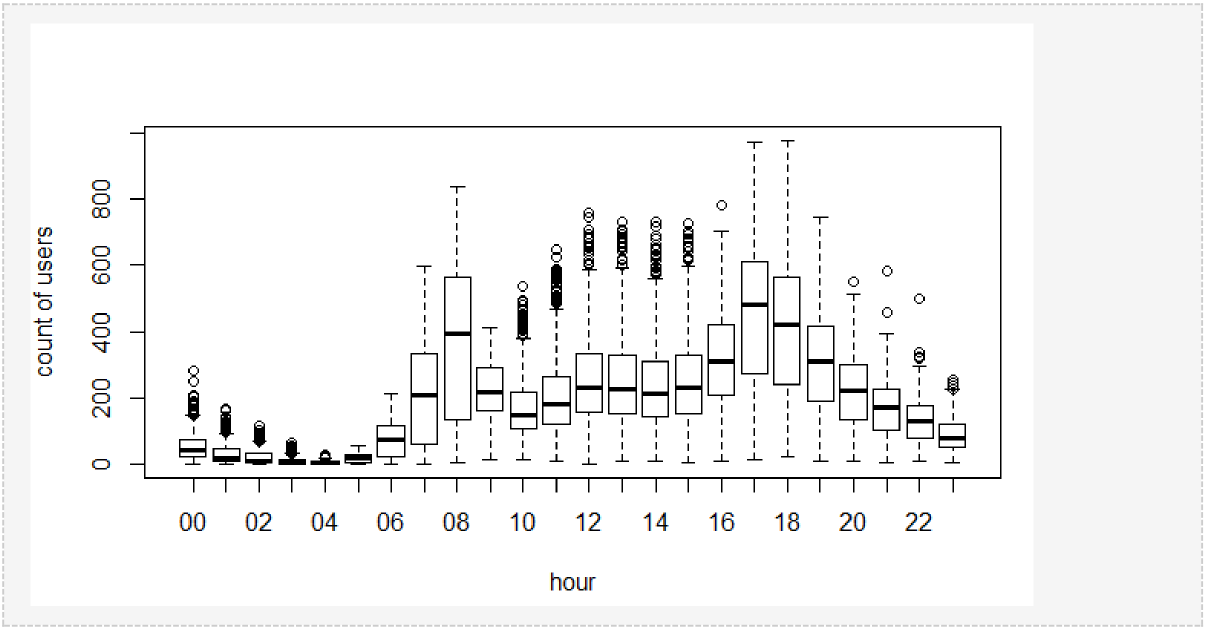
Till now, we have got a fair understanding of the data set. Now, let's test the hypothesis which we had generated earlier. Here I have added some additional hypothesis from the dataset. Let's test them one by one:

- **Hourly trend**: We don't have the variable 'hour' with us right now. But we can extract it using the datetime column.

```
data$hour=substr(data$datetime,12,13)

data$hour=as.factor(data$hour)
```

Let's plot the hourly trend of count over hours and check if our hypothesis is correct or not. We will separate train and test data set from combined one.

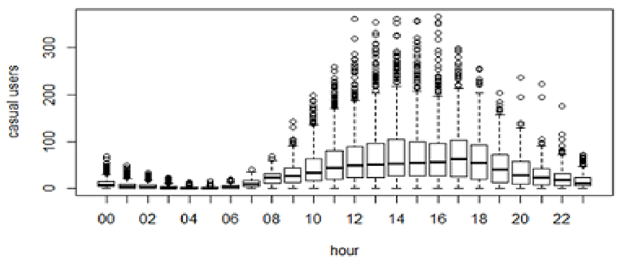
```
train=data[as.integer(substr(data$datetime,9,10))<20,]

test=data[as.integer(substr(data$datetime,9,10))>19,]

boxplot(train$count~train$hour,xlab="hour", ylab="count of users")
```

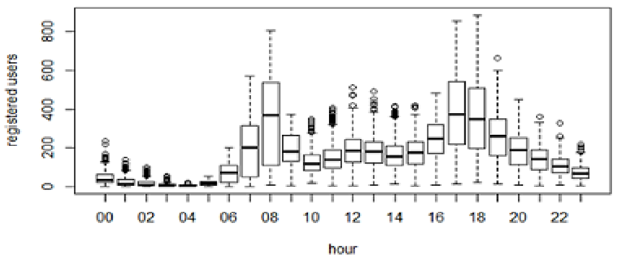


Above, you can see the trend of bike demand over hours. Quickly, I'll segregate the bike demand in three categories:

High	: 7-9 and 17-19 hours
Average	: 10-16 hours
Low	: 0-6 and 20-24 hours

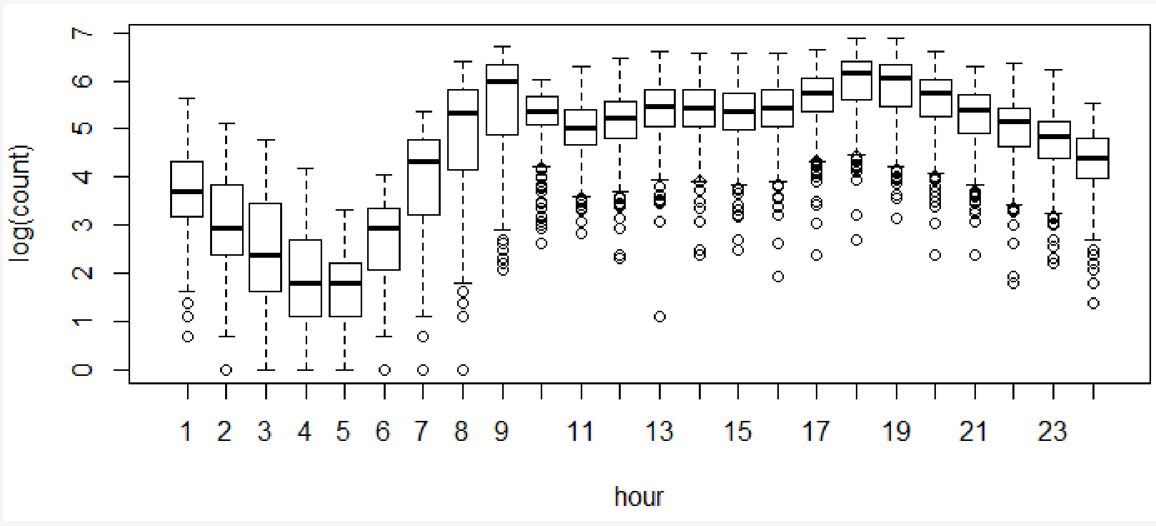
Here we have analyzed the distribution of total bike demand. Let's look at the distribution of registered and casual users separately.





Above you can see that registered users have similar trend as count. Whereas, casual users have different trend. Thus, we can say that 'hour' is significant variable and our hypothesis is 'true'.

You might have noticed that there are a lot of outliers while plotting the count of registered and casual users. These values are not generated due to error, so we consider them as natural outliers. They might be a result of groups of people taking up cycling (who are not registered). To treat such outliers, we will use logarithm transformation. Let's look at the similar plot after log transformation.

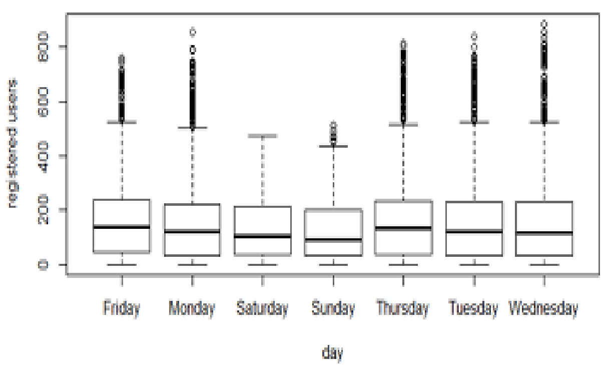


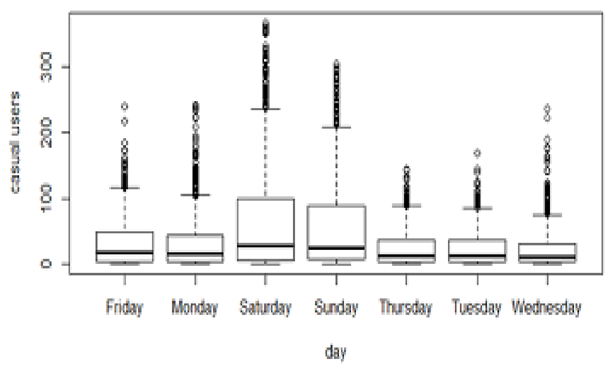
```
boxplot(log(train$count)~train$hour,xlab="hour",ylab="log(count)")
```

Daily Trend: Like Hour, we will generate a variable for day from datetime variable and after that we'll plot it.

```
date=substr(data$datetime,1,10)

days<-weekdays(as.Date(date))

data$day=days
```

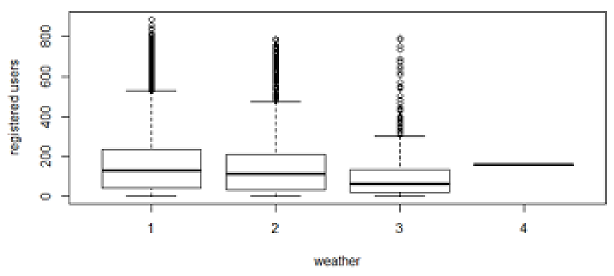
Plot shows registered and casual users' demand over days.

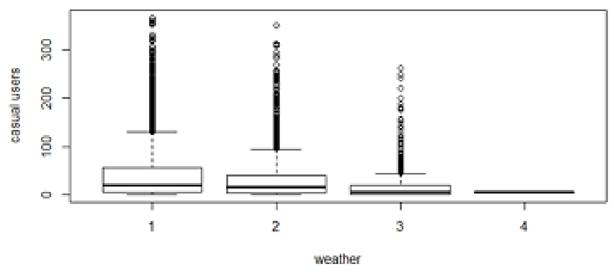




While looking at the plot, I can say that the demand of causal users increases over weekend.

Rain: We don't have the 'rain' variable with us but have 'weather' which is sufficient to test our hypothesis. As per variable description, weather 3 represents light rain and weather 4 represents

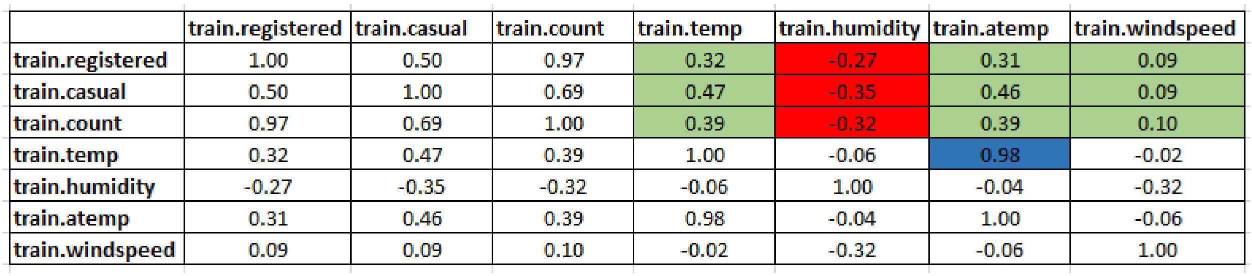




heavy rain. Take a look at the plot: It is clearly satisfying our hypothesis.

Temperature, Windspeed and Humidity: These are continuous variables so we can look at the correlation factor to validate hypothesis.

```
sub=data.frame(train$registered,train$casual,train$count,train$temp,train$humidity,train
$atemp,train$windspeed)
```



	train.registered	train.casual	train.count	train.temp	train.humidity	train.atemp	train.windspeed
train.registered	1.00	0.50	0.97	0.32	-0.27	0.31	0.09
train.casual	0.50	1.00	0.69	0.47	-0.35	0.46	0.09
train.count	0.97	0.69	1.00	0.39	-0.32	0.39	0.10
train.temp	0.32	0.47	0.39	1.00	-0.06	0.98	-0.02
train.humidity	-0.27	-0.35	-0.32	-0.06	1.00	-0.04	-0.32
train.atemp	0.31	0.46	0.39	0.98	-0.04	1.00	-0.06
train.windspeed	0.09	0.09	0.10	-0.02	-0.32	-0.06	1.00

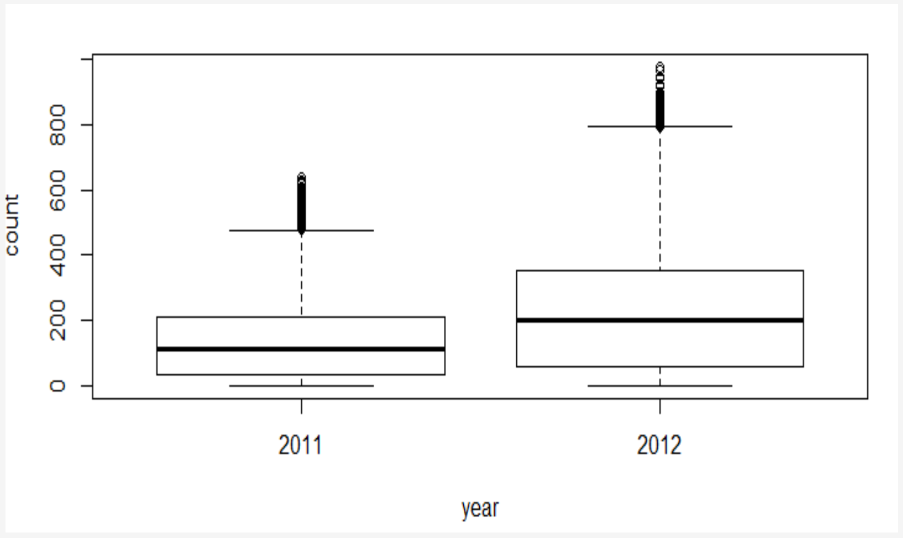
```
cor(sub)
```

Here are a few inferences you can draw by looking at the above histograms:

- Variable temp is positively correlated with dependent variables (casual is more compare to registered)
- Variable atemp is highly correlated with temp.
- Wind speed has lower correlation as compared to temp and humidity

Time: Let’s extract year of each observation from the date time column and see the trend of bike demand over year.

```
data$year=substr(data$datetime,1,4)

data$year=as.factor(data$year)

train=data[as.integer(substr(data$datetime,9,10))<20,]

test=data[as.integer(substr(data$datetime,9,10))>19,]

boxplot(train$count~train$year,xlab="year", ylab="count")
```



We can see that 2012 has higher bike demand as compared to 2011.

Pollution & Traffic: We don't have the variable related with these metrics in our data set so we cannot test this hypothesis.

5. Feature Engineering

In addition to existing independent variables, we will create new variables to improve the prediction power of model. Initially, you must have noticed that we generated new variables like hour, month, day and year.

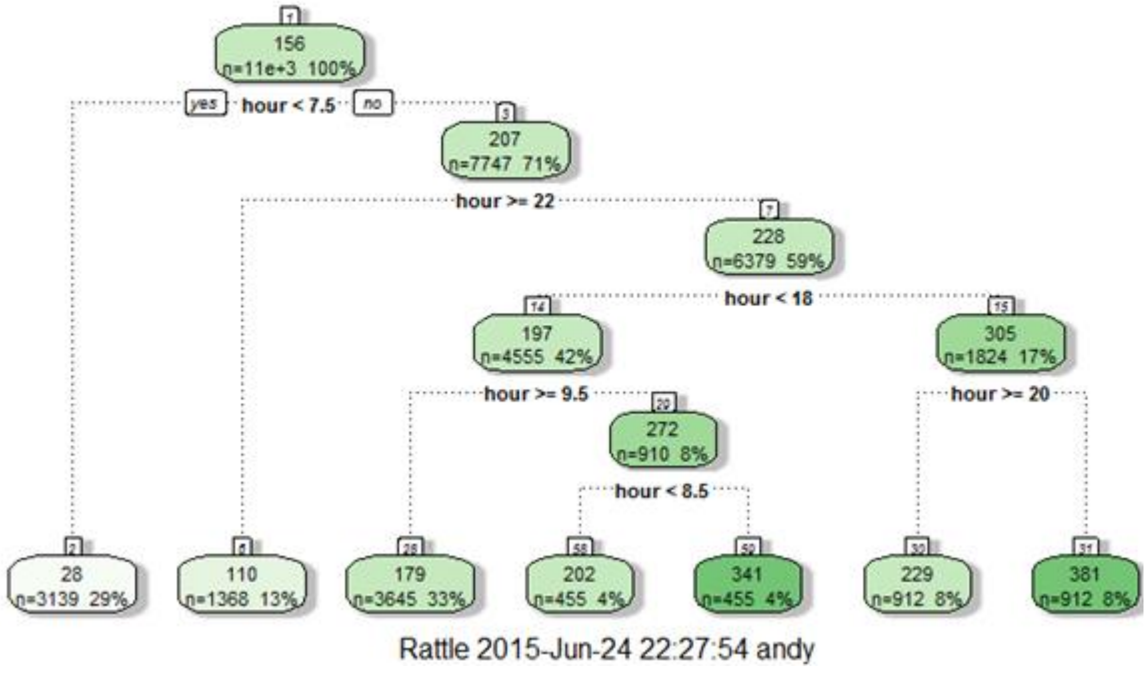
Here we will create more variables, let's look at the some of these:

Hour Bins: Initially, we have broadly categorize the hour into three categories. Let's create bins for the hour variable separately for casual and registered users. Here we will use decision tree to find the accurate bins.

```
train$hour=as.integer(train$hour) # convert hour to integer

test$hour=as.integer(test$hour) # modifying in both train and test data set
```

We use the library rpart for decision tree algorithm.

```
library(rpart)
library(rattle) #these libraries will be used to get a good visual plot for the decision tree model.

library(rpart.plot)

library(RColorBrewer)

d=rpart(registered~hour,data=train)

        fancyRpartPlot(d)
```



Now, looking at the nodes we can create different hour bucket for registered users.

```
data=rbind(train,test)

data$dp_reg=0

data$dp_reg[data$hour<8]=1

data$dp_reg[data$hour>=22]=2

data$dp_reg[data$hour>9 & data$hour<18]=3

data$dp_reg[data$hour==8]=4
```

```
data$dp_reg[data$hour==9]=5

data$dp_reg[data$hour==20 | data$hour==21]=6

data$dp_reg[data$hour==19 | data$hour==18]=7
```

Similarly, we can create day_part for casual users also (dp_cas).

Temp Bins: Using similar methods, we have created bins for temperature for both registered and casuals users. Variables created are (temp_reg and temp_cas).

Year Bins: We had a hypothesis that bike demand will increase over time and we have proved it also. Here I have created 8 bins (quarterly) for two years. Jan-Mar 2011 as 1.Oct-Dec2012 as 8.

```
data$year_part[data$year=='2011']=1

data$year_part[data$year=='2011' & data$month>3]=2

data$year_part[data$year=='2011' & data$month>6]=3

data$year_part[data$year=='2011' & data$month>9]=4

data$year_part[data$year=='2012']=5

data$year_part[data$year=='2012' & data$month>3]=6

data$year_part[data$year=='2012' & data$month>6]=7

data$year_part[data$year=='2012' & data$month>9]=8

table(data$year_part)
```

Day Type: Created a variable having categories like “weekday”, “weekend” and “holiday”.

```
data$day_type=""

data$day_type[data$holiday==0 & data$workingday==0]="weekend"

data$day_type[data$holiday==1]="holiday"

data$day_type[data$holiday==0 & data$workingday==1]="working day"
```

Weekend: Created a separate variable for weekend (0/1)

```
data$weekend=0
```

```
data$weekend[data$day=="Sunday" | data$day=="Saturday" ]=1
```

6. Model Building

Before executing the random forest model code, I have followed following steps:

- Convert discrete variables into factor (weather, season, hour, holiday, working day, month, day)

```
train$hour=as.factor(train$hour)

test$hour=as.factor(test$hour)
```

- As we know that dependent variables have natural outliers so we will predict log of dependent variables.
- Predict bike demand registered and casual users separately.
  y1=log(casual+1) and y2=log(registered+1), Here we have added 1 to deal with zero values in the casual and registered columns.

```
#predicting the log of registered users.

set.seed(415)

fit1 <- randomForest(logreg ~ hour +workingday+day+holiday+ day_type
+temp_reg+humidity+atemp+windspeed+season+weather+dp_reg+weekend+yea
r+year_part, data=train,importance=TRUE, ntree=250)

pred1=predict(fit1,test)

test$logreg=pred1

#predicting the log of casual users.

set.seed(415)

fit2 <- randomForest(logcas ~hour +
day_type+day+humidity+atemp+temp_cas+windspeed+season+weather+holiday
+workingday+dp_cas+weekend+year+year_part, data=train,importance=TRUE,
ntree=250)

pred2=predict(fit2,test)

test$logcas=pred2
```

Re-transforming the predicted variables and then writing the output of count to the file submit.csv

```
test$registered=exp(test$logreg)-1

test$casual=exp(test$logcas)-1

test$count=test$casual+test$registered

s<-data.frame(datetime=test$datetime,count=test$count)

write.csv(s,file="submit.csv",row.names=FALSE)
```