**Data Analysis and Visualisations using R**

*Step by Step Guide for Beginners*

Are you starting your journey in the field of Data Science? Do you need to know how to get started with R? Are you intrigued by Data Visualisations? If yes, then this tutorial is meant for you!

**Overview & Purpose**

With this article, we’d learn how to do basic exploratory analysis on a data set, create visualisations and draw inferences.

**What we’d be covering**

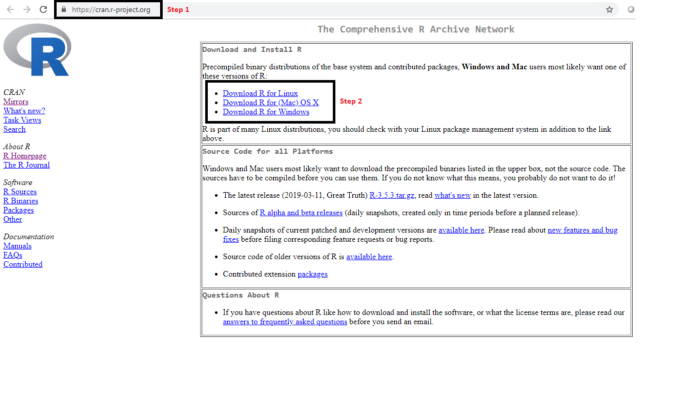
1. Getting Started with R
2. Understanding your Data Set
3. Analysing & Building Visualisations

**1. Getting Started with R**

**1.1 Download and Install R | R Studio**

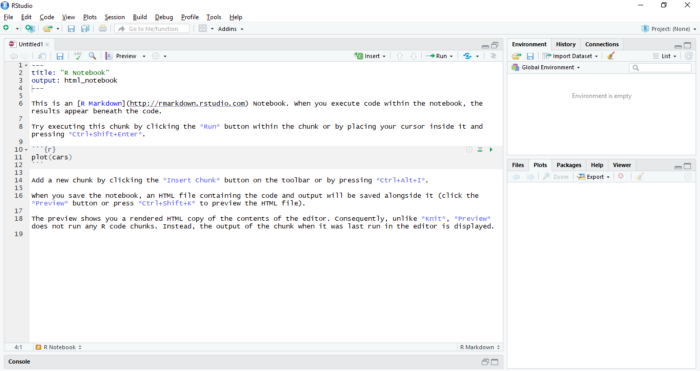
R programming offers a set of inbuilt libraries that help build visualisations with minimal code and flexibility.

You can download R easily from the [R Project Website](https://www.r-project.org/). While downloading you would need to choose a mirror. Choose R depending on your operating system, such as Windows, Mac or Linux.



It is super easy to install R. Just follow through the basic installation steps and you’d be good to go.

For an easy way to write scripts, I recommend using **R Studio**. It is an open source environment which is known for its simplicity and efficiency.



Launch Screen after starting R Studio

**1.2 Install R packages**

Packages are the fundamental units created by the community that contains reproducible R code. These include reusable R functions, documentation that describes how to use them and sample data.

The directory where packages are stored is called the library. R comes with a standard set of packages. Others are available for download and installation. Once installed, they have to be loaded into the session to be used.

To install a package in R, we simply use the command

***install.packages****(“Name of the Desired Package”)*

**1.3 Loading the Data set**

There are some data sets that are already pre-installed in R. Here, we shall be using **The Titanic**data set that comes built-in R in the Titanic Package.

While using any external data source, we can use the read command to load the files(Excel, CSV, HTML and text files etc.)

This data set is also available at [Kaggle](https://www.kaggle.com/c/titanic/data" \t "_blank). You may download the data set, both train and test files. In this tutorial, we’d be just using the train data set.

*titanic <-****read.csv****(“C:/Users/Desktop/titanic.csv”, header=TRUE, sep=”,”)*

The above code reads the file titanic.csv into a dataframe **titanic**. With Header=TRUE we are specifying that the data includes a header(column names) and sep=”,” specifies that the values in data are comma separated.

**2. Understanding the Data set**

We have used the Titanic data set that contains historical records of all the passengers who on-boarded the Titanic. Below is a brief description of the 12 variables in the data set :

* PassengerId: Serial Number
* Survived: Contains binary Values of 0 & 1. Passenger did not survive — 0, Passenger Survived — 1.
* Pclass — Ticket Class | 1st Class, 2nd Class or 3rd Class Ticket
* Name — Name of the passenger
* Sex — Male or Female
* Age — Age in years — Integer
* SibSp — No. of Siblings / Spouses — brothers, sisters and/or husband/wife
* Parch — No. of parents/children — mother/father and/or daughter, son
* Ticket — Serial Number
* Fare — Passenger fare
* Cabin — Cabin Number
* Embarked — Port of Embarkment | C- Cherbourg, Q — Queenstown, S — Southhampton

**2.1 Peek at your Data**

Before we begin working on the dataset, let’s have a good look at the raw data.

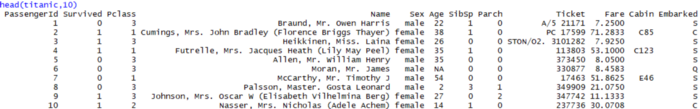
view(titanic)

This helps us in familiarising with the data set.

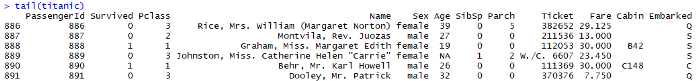


***head(titanic,n) | tail(titanic,n)***

In order to have a quick look at the data, we often use the head()/tail().



Top 10 rows of the data set.



Bottom 5 rows of the data set.

In case we do not explicitly pass the value for n, it takes the default value of 5, and displays 5 rows.

**names(titanic)**

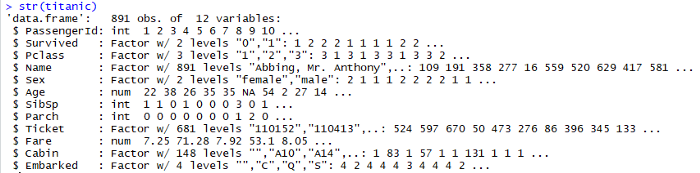
This helps us in checking out all the variables in the data set.

https://miro.medium.com/max/700/1*Dq-3wd_5PEpZ1AAs6Prv6w.png

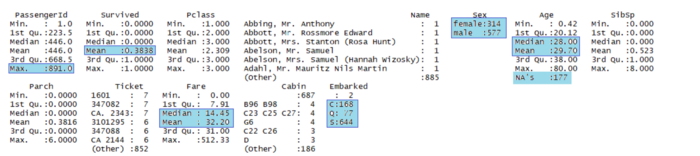
Familiarising with all the Variables/Column Names

**str(titanic)**

This helps in understanding the structure of the data set, data type of each attribute and number of rows and columns present in the data.



**summary(titanic)**



A cursory look at the data

**Summary()** is one of the most important functions that help in summarising each attribute in the dataset. It gives a set of descriptive statistics, depending on the type of variable:

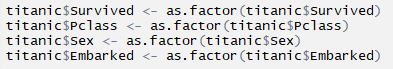
* In case of a Numerical Variable -> Gives Mean, Median, Mode, Range and Quartiles.
* In case of a Factor Variable -> Gives a table with the frequencies.
* In case of Factor + Numerical Variables -> Gives the number of missing values.
* In case of character variables -> Gives the length and the class.

In case we just need the summary statistic for a particular variable in the dataset, we can use

*summary(datasetName$VariableName) -> summary(titanic$Pclass)*

**as.factor(dataset$ColumnName)**

There are times when some of the variables in the data set are factors but might get interpreted as numeric. For example, the Pclass(Passenger Class) tales the values 1, 2 and 3, however, we know that these are not to be considered as numeric, as these are just levels. In order to such variables treated as factors and not as numbers we need explicitly convert them to factors using the function **as.factor()**



**3. Analysis & Visualisations**

Data Visualisation is an art of turning data into insights that can be easily interpreted. In this tutorial, we’ll analyse the survival patterns and check for factors that affected the same.

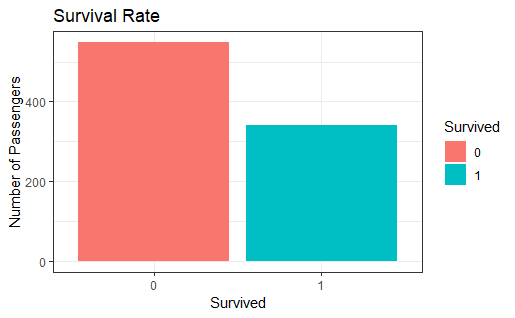
**Points to think about**

Now that we have an understanding of the dataset, and the variables, we need to identify the variables of interest. Domain knowledge and the correlation between variables help in choosing these variables. To keep it simple, we have chosen only 3 such variables, namely Age, Gender, Pclass.

**What was the survival rate?**

When talking about the Titanic data set, the first question that comes up is “How many people did survive?”. Let’s have a simple Bar Graph to demonstrate the same.

***ggplot(titanic, aes(x=Survived)) + geom\_bar()***



On the X-axis we have the survived variable, 0 representing the passengers that did not survive, and 1 representing the passengers who survived. The Y -axis represents the number of passengers. Here we see that over 550 passenger did not survive and ~ 340 passengers survived.

Let’s make is more clear by using checking out the percentages

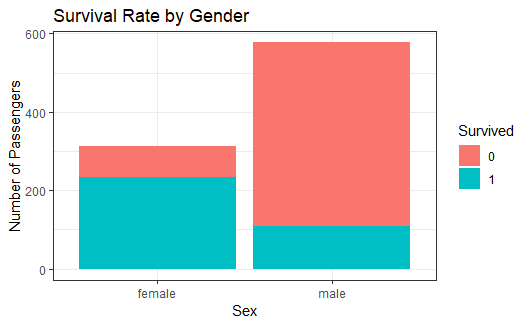
***prop.table(table(titanic$Survived))***

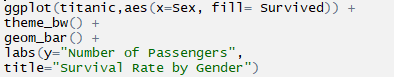
https://miro.medium.com/max/309/1*WXLUuMmp705PFUimYXlcbA.png

Only 38.38% of the passengers who on-boarded the titanic did survive.

**Survival rate basis Gender**

It is believed that in case of rescue operations during disasters, woman’s safety is prioritised. Did the same happen back then?

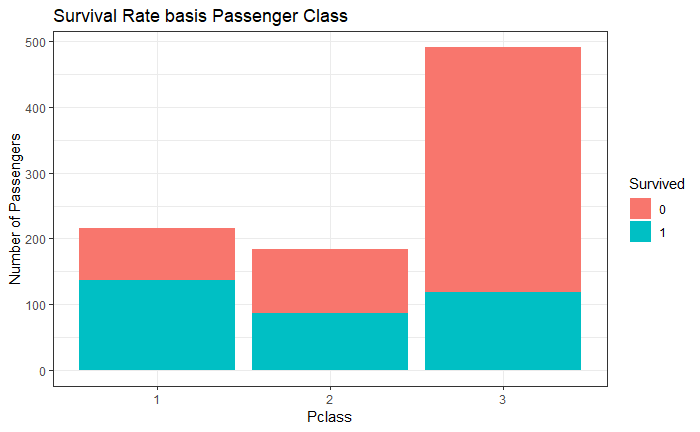




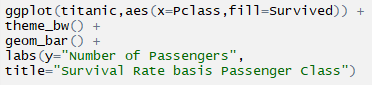
*We see that the survival rate amongst the women was significantly higher when compared to men. The survival ratio amongst women was around 75%, whereas for men it was less than 20%.*

**Survival Rate basis Class of tickets (Pclass)**

There were 3 segments of passengers, depending upon the class they were travelling in, namely, 1st class, 2nd class and 3rd class. We see that over 50% of the passengers were travelling in the 3rd class.



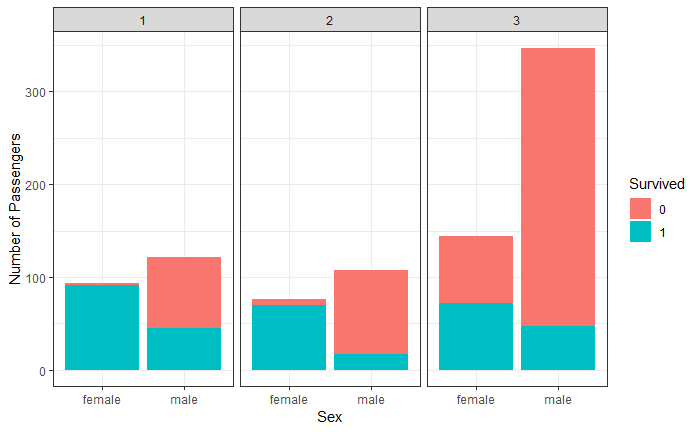
Survival Rate basis Passenger Class



1st and 2nd Class passengers disproportionately survived, with over 60% survival rate of the 1st class passengers, around 45–50% of 2nd class, and less than 25% survival rate of those travelling in 3rd class.

I’ll leave you at the thought… Was it because of a preferential treatment to the passengers travelling eliteclass, or the proximity, as the 3rd class compartments were in the lower deck?

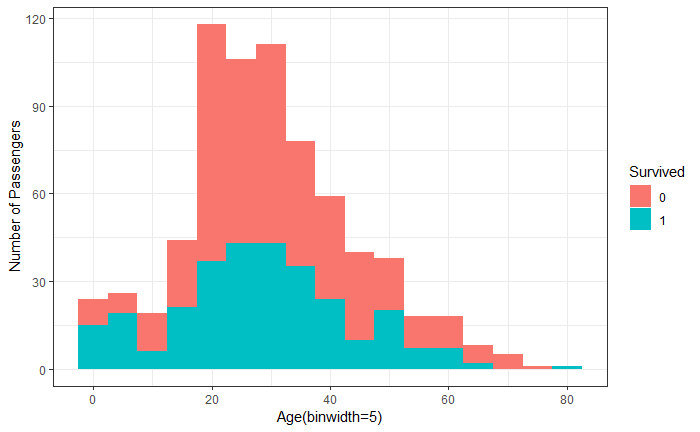
**Survival Rate basis Class of tickets and Gender*(pclass)***

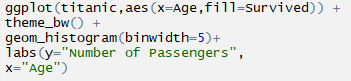


**We see that the females in the 1st and 2nd class had a very high survival rate. The survival rate for the females travelling in 1st and 2nd class was 96% and 92% respectively, corresponding to 37% and 16% for men. The survival rate for men travelling 3rd class was less than 15%.**

Till now it is evident that the Gender and Passenger class had significant impact on the survival rates. Let’s now check the impact of passenger’s Age on Survival Rate.

**Survival rates basis age**



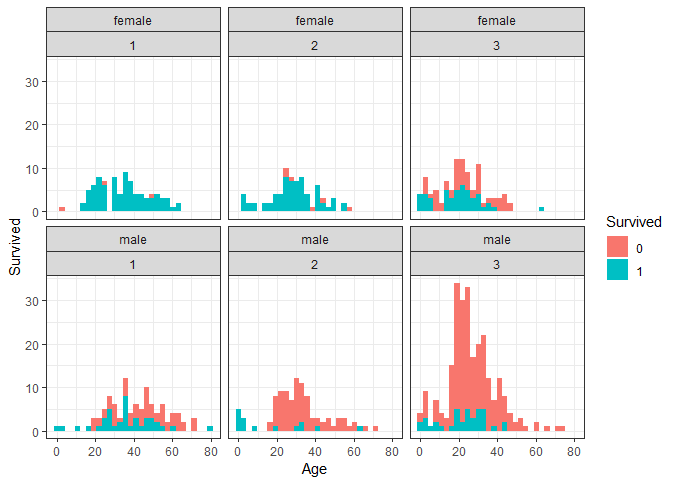


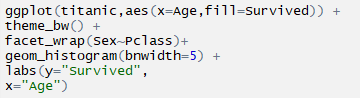
Looking at the age<10 years section in the graph, we see that the survival rate is high. And the survival rate is low and drops beyond the age of 45.

Here we have used bin width of 5, you may try out different values and see, how the graph changes.

**Survival Rate basis Age, Gender and Class of tickets**

This graph helps identify the survival patterns considering all the three variables.





The top 3 sections depict the female survival patterns across the three classes, while the bottom 3 represent the male survival patterns across 3 classes. On the x-axis we have the Age.

It is evident that the survival rate of children, across 1st and 2nd class was the highest. Except for 1 girl child all children travelling 1st and 2nd class survived. The survival rates were lowest for men travelling 3rd class.

# Data Visualization in R

* **Difficulty Level :** [Expert](https://www.geeksforgeeks.org/expert/)
* **Last Updated :** 26 Apr, 2022

**Data visualization** is the technique used to deliver insights in data using visual cues such as graphs, charts, maps, and many others. This is useful as it helps in intuitive and easy understanding of the large quantities of data and thereby make better decisions regarding it.

## Data Visualization in R Programming Language

The popular data visualization tools that are available are Tableau, Plotly, R, Google Charts, Infogram, and Kibana. The various data visualization platforms have different capabilities, functionality, and use cases. They also require a different skill set. This article discusses the use of R for data visualization.

R is a language that is designed for statistical computing, graphical data analysis, and scientific research. It is usually preferred for data visualization as it offers flexibility and minimum required coding through its packages.

### Consider the following *airquality* data set for visualization in R:

| Ozone | Solar R. | Wind | Temp | Month | Day |
| --- | --- | --- | --- | --- | --- |
| 41 | 190 | 7.4 | 67 | 5 | 1 |
| 36 | 118 | 8.0 | 72 | 5 | 2 |
| 12 | 149 | 12.6 | 74 | 5 | 3 |
| 18 | 313 | 11.5 | 62 | 5 | 4 |
| NA | NA | 14.3 | 56 | 5 | 5 |
| 28 | NA | 14.9 | 66 | 5 | 6 |

## Types of Data Visualizations

Some of the various types of visualizations offered by R are:

### Bar Plot

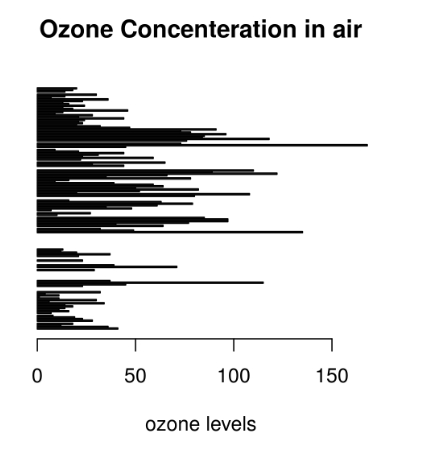
There are two types of bar plots- horizontal and vertical which represent data points as horizontal or vertical bars of certain lengths proportional to the value of the data item. They are generally used for continuous and categorical variable plotting. By setting the **horiz** parameter to true and false, we can get horizontal and vertical bar plots respectively.

**Example 1:**

* R

|  |
| --- |
| # Horizontal Bar Plot for  # Ozone concentration in air  barplot(airquality$Ozone,          main = 'Ozone Concenteration in air',          xlab = 'ozone levels', horiz = TRUE) |

**Output:**

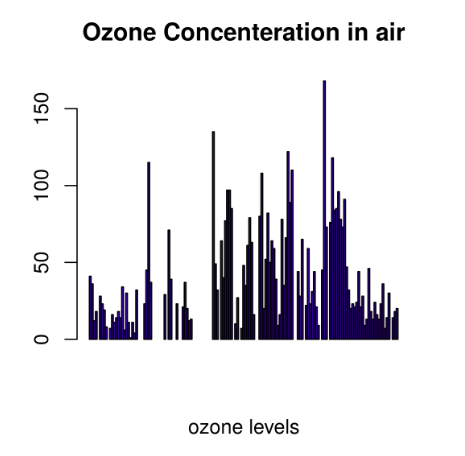


**Example 2:**

* R

|  |
| --- |
| # Vertical Bar Plot for  # Ozone concentration in air  barplot(airquality$Ozone, main = 'Ozone Concenteration in air',          xlab = 'ozone levels', col ='blue', horiz = FALSE) |

**Output:**



Bar plots are used for the following scenarios:

* To perform a comparative study between the various data categories in the data set.
* To analyze the change of a variable over time in months or years.

### Histogram

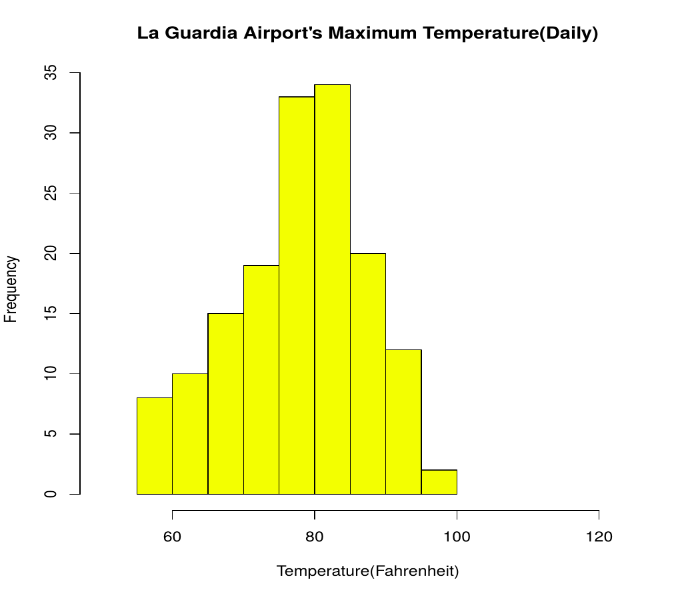
A histogram is like a bar chart as it uses bars of varying height to represent data distribution. However, in a histogram values are grouped into consecutive intervals called bins. In a Histogram, continuous values are grouped and displayed in these bins whose size can be varied.

**Example:**

* R

|  |
| --- |
| # Histogram for Maximum Daily Temperature  data(airquality)    hist(airquality$Temp, main ="La Guardia Airport's\  Maximum Temperature(Daily)",      xlab ="Temperature(Fahrenheit)",      xlim = c(50, 125), col ="yellow",      freq = TRUE) |

**Output:**



For a histogram, the parameter **xlim** can be used to specify the interval within which all values are to be displayed.   
Another parameter **freq** when set to *TRUE* denotes the frequency of the various values in the histogram and when set to *FALSE*, the probability densities are represented on the y-axis such that they are of the histogram adds up to one.

**Histograms are used in the following scenarios:**

* To verify an equal and symmetric distribution of the data.
* To identify deviations from expected values.

### Box Plot

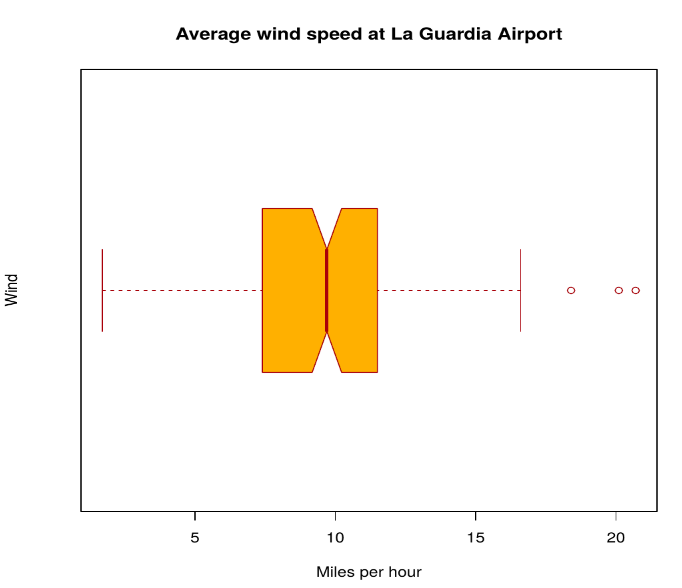
The statistical summary of the given data is presented graphically using a boxplot. A boxplot depicts information like the minimum and maximum data point, the median value, first and third quartile, and interquartile range.

**Example:**

* R

|  |
| --- |
| # Box plot for average wind speed  data(airquality)    boxplot(airquality$Wind, main = "Average wind speed\  at La Guardia Airport",          xlab = "Miles per hour", ylab = "Wind",          col = "orange", border = "brown",          horizontal = TRUE, notch = TRUE) |

**Output:**



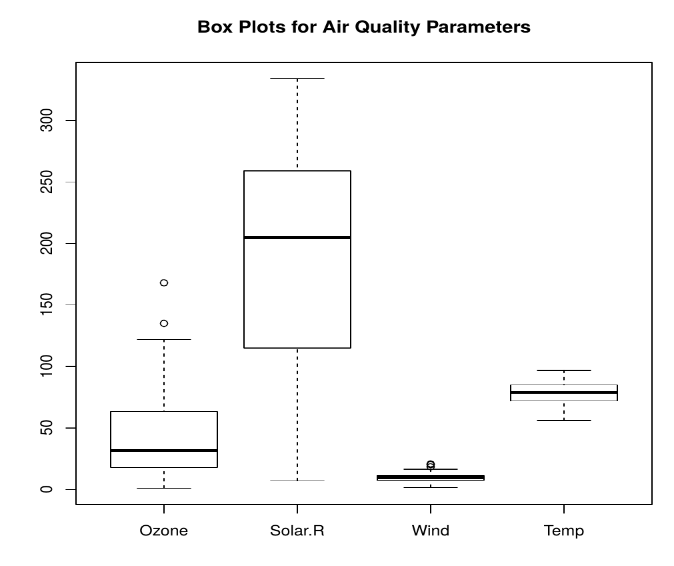
Multiple box plots can also be generated at once through the following code:

**Example:**

* R

|  |
| --- |
| # Multiple Box plots, each representing  # an Air Quality Parameter  boxplot(airquality[, 0:4],          main ='Box Plots for Air Quality Parameters') |

**Output:**



**Box Plots are used for:**

* To give a comprehensive statistical description of the data through a visual cue.
* To identify the outlier points that do not lie in the inter-quartile range of data.

### Scatter Plot

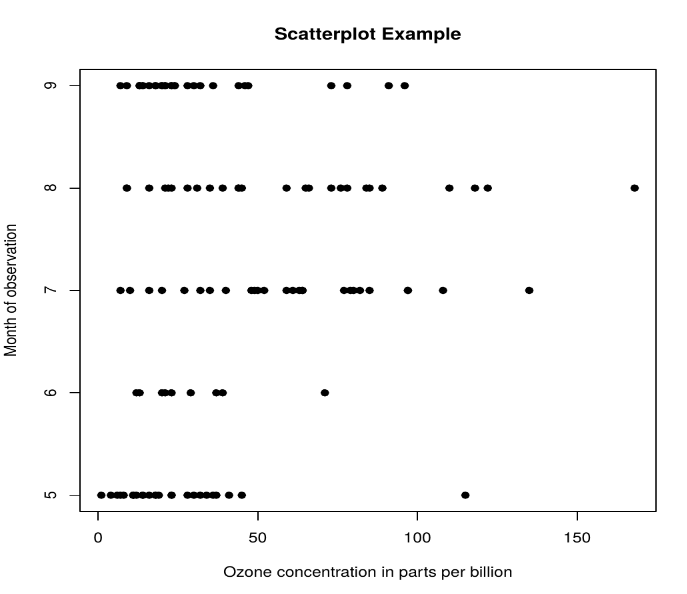
A scatter plot is composed of many points on a Cartesian plane. Each point denotes the value taken by two parameters and helps us easily identify the relationship between them.

**Example:**

* R

|  |
| --- |
| # Scatter plot for Ozone Concentration per month  data(airquality)    plot(airquality$Ozone, airquality$Month,       main ="Scatterplot Example",      xlab ="Ozone Concentration in parts per billion",      ylab =" Month of observation ", pch = 19) |

**Output:**



**Scatter Plots are used in the following scenarios:**

* To show whether an association exists between bivariate data.
* To measure the strength and direction of such a relationship.

### Heat Map

Heatmap is defined as a graphical representation of data using colors to visualize the value of the matrix. heatmap() function is used to plot heatmap.

***Syntax:****heatmap(data)*

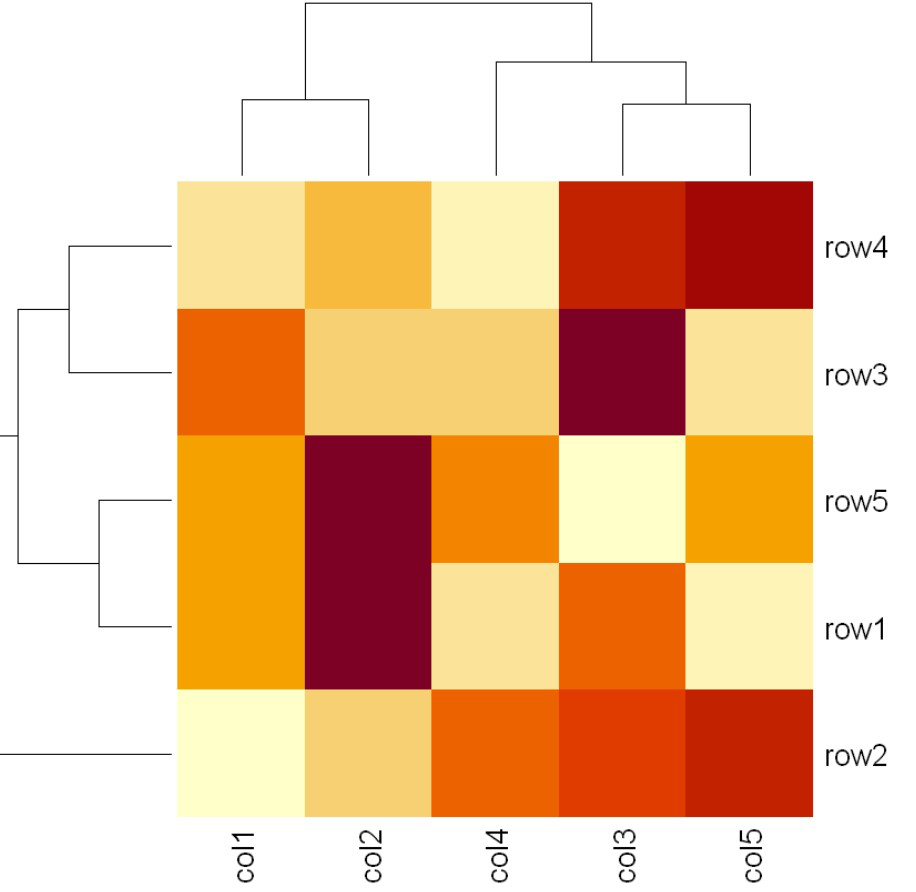
***Parameters:****data: It represent matrix data, such as values of rows and columns*

***Return:****This function draws a heatmap.*

* R

|  |
| --- |
| # Set seed for reproducibility  # set.seed(110)    # Create example data  data <- matrix(rnorm(50, 0, 5), nrow = 5, ncol = 5)    # Column names  colnames(data) <- paste0("col", 1:5)  rownames(data) <- paste0("row", 1:5)    # Draw a heatmap  heatmap(data) |

**Output:**



### Map visualization in R

Here we are using maps package to visualize and display geographical maps using an R programming language.

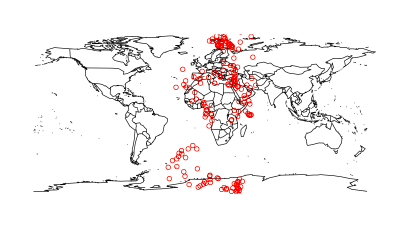
install.packages("maps")

Link of the dataset: [worldcities.csv](https://drive.google.com/file/d/1H_CyZpNY4Ku2MRckLreqIitcFHDvIHg1/view)

* R

|  |
| --- |
| # Read dataset and convert it into  # Dataframe  data <- read.csv("worldcities.csv")  df <- data.frame(data)    # Load the required libraries  library(maps)  map(database = "world")    # marking points on map  points(x = df$lat[1:500], y = df$lng[1:500], col = "Red") |

**Output:**



### 3D Graphs in R

Here we will use preps() function, This function is used to create 3D surfaces in perspective view. This function will draw perspective plots of a surface over the x–y plane.

***Syntax:****persp(x, y, z)*

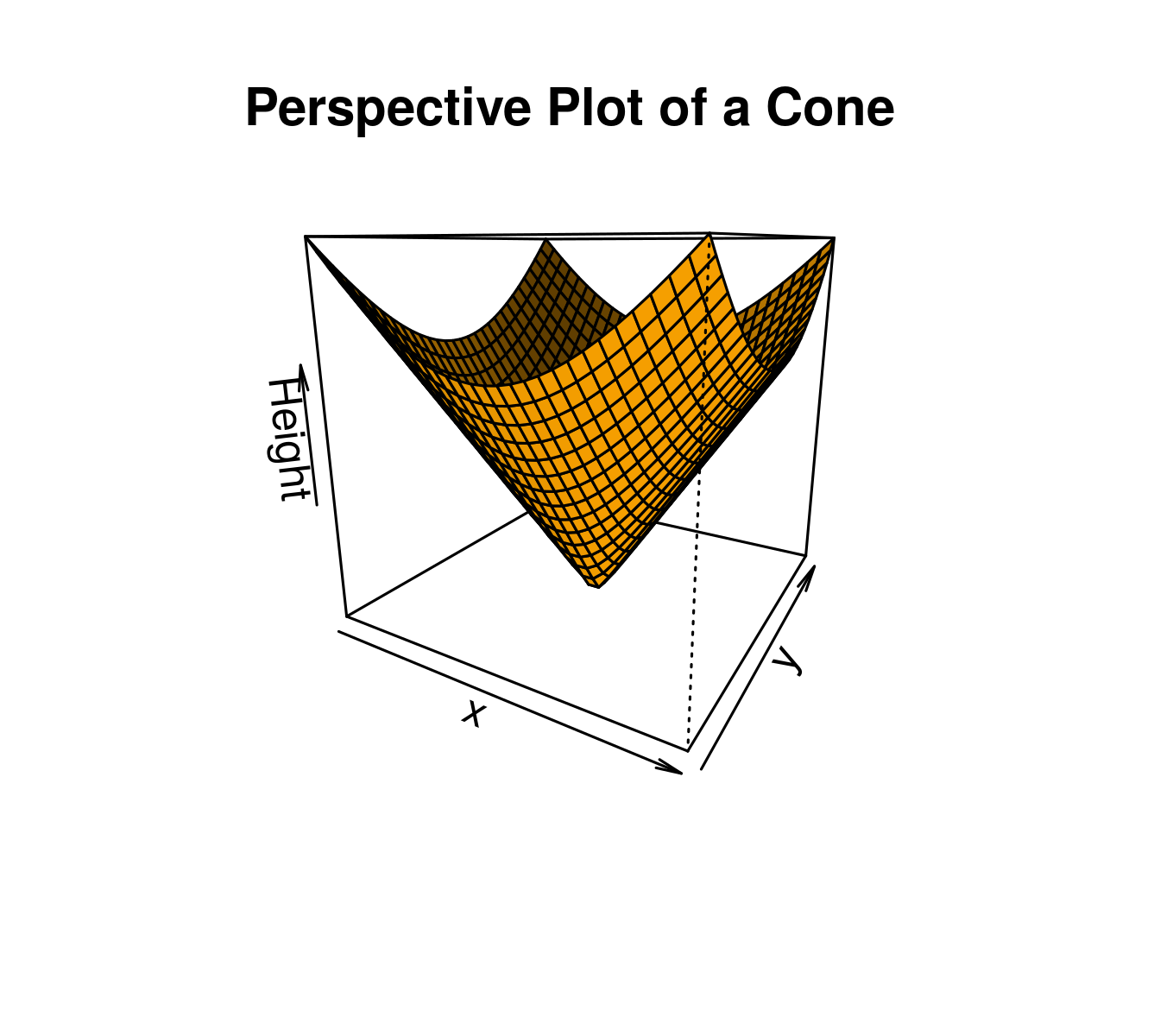
***Parameter:****This function accepts different parameters i.e. x, y and z where x and y are vectors defining the location along x- and y-axis. z-axis will be the height of the surface in the matrix z.*

***Return Value:****persp() returns the viewing transformation matrix for projecting 3D coordinates (x, y, z) into the 2D plane using homogeneous 4D coordinates (x, y, z, t).*

* R

|  |
| --- |
| # Adding Titles and Labeling Axes to Plot  cone <- function(x, y){  sqrt(x ^ 2 + y ^ 2)  }    # prepare variables.  x <- y <- seq(-1, 1, length = 30)  z <- outer(x, y, cone)    # plot the 3D surface  # Adding Titles and Labeling Axes to Plot  persp(x, y, z,  main="Perspective Plot of a Cone",  zlab = "Height",  theta = 30, phi = 15,  col = "orange", shade = 0.4) |

**Output:**



## ****Advantages of Data Visualization in R:****

R has the following advantages over other tools for data visualization:

* R offers a broad collection of visualization libraries along with extensive online guidance on their usage.
* R also offers data visualization in the form of 3D models and multipanel charts.
* Through R, we can easily customize our data visualization by changing axes, fonts, legends, annotations, and labels.

## ****Disadvantages of Data Visualization in R:****

R also has the following disadvantages:

* R is only preferred for data visualization when done on an individual standalone server.
* Data visualization using R is slow for large amounts of data as compared to other counterparts.

## ****Application Areas:****

* Presenting analytical conclusions of the data to the non-analysts departments of your company.
* Health monitoring devices use data visualization to track any anomaly in blood pressure, cholesterol and others.
* To discover repeating patterns and trends in consumer and marketing data.
* Meteorologists use data visualization for assessing prevalent weather changes throughout the world.
* Real-time maps and geo-positioning systems use visualization for traffic monitoring and estimating travel time.

# Creating Visualizations

* [Bar plot](https://db.rstudio.com/best-practices/visualization/#bar-plot)
* [Histogram](https://db.rstudio.com/best-practices/visualization/#histogram)
* [Raster Plot](https://db.rstudio.com/best-practices/visualization/#raster-plot)
* [Use an R package](https://db.rstudio.com/best-practices/visualization/#use-an-r-package)

Typically, a function that produces a plot in R performs the data crunching and the graphical rendering. For example, geom\_histogram() calculates the bin sizes and the count per bin, and then it renders the plot. Plotting functions usually require that 100% of the data be passed to them. This is a problem when working with a database. The best approach is to move the data transformation to the database, and then use a graphing function to render the results.

This article has two goals:

* Demonstrate a practical implementation of the **“Transform in database, plot in R”** concept by showing how to visualize a categorical variable using a Bar plot, a single continuous variable using a Histogram, and two continuous variables using a Raster plot, all using data in a database
* Introduce a technique that simplifies the use of complex formulas that are required to move the calculations of the plot to the database

An alternative, is to use a helper R package that already implements the principles shared in this article, please see the [dbplot page](https://db.rstudio.com/dbplot) for more info.

## Bar plot

A Bar plot is intended to measure and compare categorical data. Passing the category to geom\_bar() as x will automatically calculate the height of the bars based on the row count per category. Here is the code of a typical bar plot using ggplot2:

ggplot(data = flights) +

geom\_bar(aes(x = origin), stat = "count")

### DATA TRANSFORMATION

Because dplyr is being used to compute the count per category inside the database, the discrete values are separated using group\_by(), followed by tally() to obtain the row count per category. Lastly, collect() downloads the results into R:

df <- tbl(con, "flights") %>%

group\_by(origin) %>%

tally() %>%

collect()

df

**#***# # A tibble: 3 x 2*

**#***# origin n*

**#***# <chr> <int>*

**#***# 1 LGA 104662*

**#***# 2 EWR 120835*

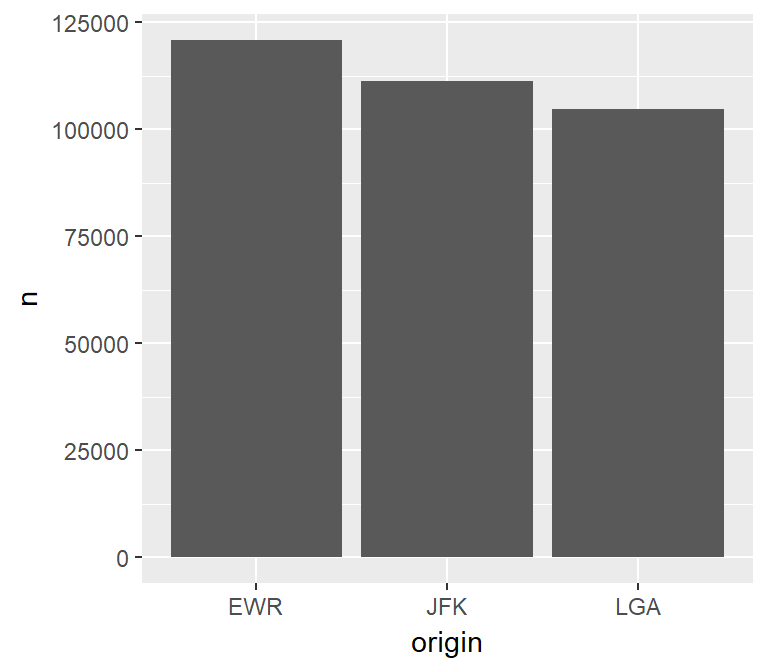
**#***# 3 JFK 111279*

### PLOTTING RESULTS IN R

The results of the Data Transformation step can now be used in ggplot2 to render the plot. This time, geom\_col() is used instead of geom\_bar() because the height of the bars have been pre-calculated by dplyr:

ggplot(data = df) +

geom\_col(aes(x = origin, y = n))



### TRANSFORM AND PLOT

The plot can be created using a single piped line of code. This is particularly useful when performing exploratory data analysis because it is easy to add or remove filters, or to change the variable that is being analyzed.

tbl(con, "flights") %>%

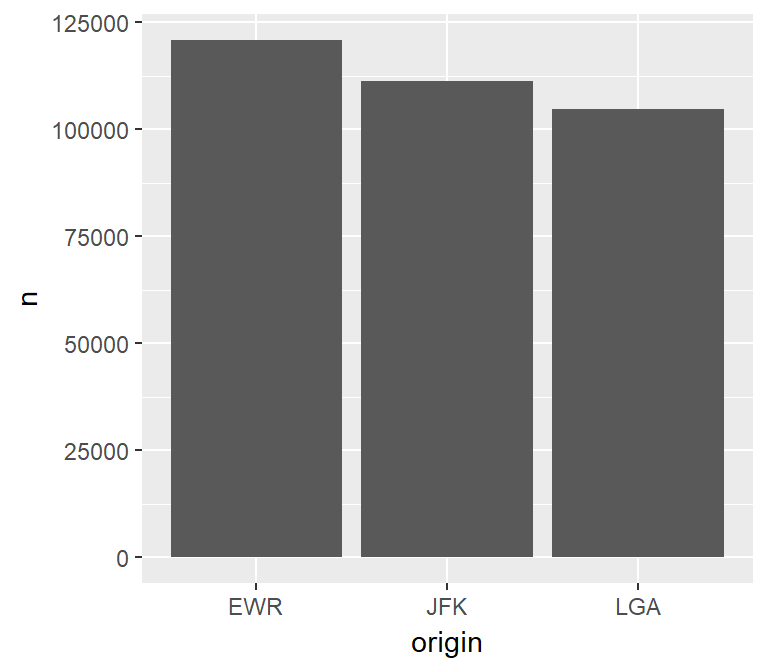
group\_by(origin) %>%

tally() %>%

collect() %>%

ggplot() +

geom\_col(aes(x = origin, y = n))



## Histogram

The histogram is intended to visualize the distribution of the values of a continuous variable. It does this by grouping the values into bins with the same range of values. In essence, a histogram converts a continuous variable to a discrete variable by splitting and placing the variable’s values into multiple bins.

### CALCULATIONS

The following breakdown of the calculation needed to create a histogram is intended to highlight the complexity of moving its processing to the database.

For example, if a histogram with 20 bins is needed, and the variable has a minimum value of 1 and a maximum value of 101, then each bin needs to be 5.

* 101 (Max value) - 1 (Min value) = 100
* 100 / 20 (Number of bins) = 5

The first bin will have a range of 1 to 6, the second 7 to 12, etc.

After that, the count of values that are inside each range needs to be determined. In this example, there may be two rows that have a value between 1 and 6 and five rows with values between 7 and 12.

Any formula used to create a Histogram will need to calculate the bins, place the values inside the bins, and only call math functions supported by the database in use.

### USING A HELPER FUNCTION

An advantage of using dplyr to convert the continuous variable into a discrete variable is that one solution can be applied to multiple database types. This is possible if the resulting formula is made of basic functions that most SQL databases support and is expressed in R, so that dplyr can translate it into the proper SQL syntax.

Unfortunately, the formula is rather long and mistakes can be made if used in multiple locations, because any corrections to the formula may not be propagated to all of the instances. To solve this, a helper function can be used.

In the following helper function, the var input is used to build the formula in an **unevaluated** R code format. When used inside dplyr, it will return the assembled formula which will then be **evaluated** as inside the verb command. Feel free to copy this function into your script or R Notebook.

The function has two other arguments:

* bins - this allows the number of bins to be customized. It defaults to 30
* binwidth - this is used to specify the size of the bin. It overrides any value passed to the bins argument.

**library**(rlang)

db\_bin <- **function**(var, bins = 30, binwidth = NULL) {

var <- enexpr(var)

range <- expr((max(!! var, na.rm = TRUE) - min(!! var, na.rm = TRUE)))

**if** (is.null(binwidth)) {

binwidth <- expr((!! range / !! bins))

} **else** {

bins <- expr(as.integer(!! range / !! binwidth))

}

*# Made more sense to use floor() to determine the bin value than*

*# using the bin number or the max or mean, feel free to customize*

bin\_number <- expr(as.integer(floor((!! var - min(!! var, na.rm = TRUE)) / !! binwidth)))

*# Value(s) that match max(x) will be rebased to bin -1, giving us the exact number of bins requested*

expr(((!! binwidth) \*

ifelse(!! bin\_number == !! bins, !! bin\_number - 1, !! bin\_number)) + min(!! var, na.rm = TRUE))

}

Notice that the function returns a **quosure** containing the **unevaluated** R code that calculates the bins. To read more about how this kind of approach works, please refer to this article: [Programming with dplyr](http://dplyr.tidyverse.org/articles/programming.html).

It is important to note that the database in use needs to support the functions called in the formula, such as min() and max().

Here is an example of the function’s output. Notice that a fictitious field called any\_field is used, and no “missing field” error is generated. That is because the formula has not yet been evaluated.

db\_bin(any\_field)

**#***# (((max(any\_field, na.rm = TRUE) - min(any\_field, na.rm = TRUE))/30) \**

**#***# ifelse(as.integer(floor((any\_field - min(any\_field, na.rm = TRUE))/((max(any\_field,*

**#***# na.rm = TRUE) - min(any\_field, na.rm = TRUE))/30))) ==*

**#***# 30, as.integer(floor((any\_field - min(any\_field, na.rm = TRUE))/((max(any\_field,*

**#***# na.rm = TRUE) - min(any\_field, na.rm = TRUE))/30))) -*

**#***# 1, as.integer(floor((any\_field - min(any\_field, na.rm = TRUE))/((max(any\_field,*

**#***# na.rm = TRUE) - min(any\_field, na.rm = TRUE))/30))))) +*

**#***# min(any\_field, na.rm = TRUE)*

This is an example of the function using binwidth. The resulting formula is a little different.

db\_bin(any\_field, binwidth = 300)

**#***# (300 \* ifelse(as.integer(floor((any\_field - min(any\_field, na.rm = TRUE))/300)) ==*

**#***# as.integer((max(any\_field, na.rm = TRUE) - min(any\_field,*

**#***# na.rm = TRUE))/300), as.integer(floor((any\_field - min(any\_field,*

**#***# na.rm = TRUE))/300)) - 1, as.integer(floor((any\_field - min(any\_field,*

**#***# na.rm = TRUE))/300)))) + min(any\_field, na.rm = TRUE)*

### DATA TRANSFORMATION

The data processing is very simple when using the helper function. The db\_bin function is used inside group\_by(). There are a couple of **must-do’s** to keep in mind:

* **Specify the name of the field that uses the**db\_bin()**function** - If a name is not specified, dplyr will use the long formula text as the default name of the field, which in most cases breaks the database’s field name length rules.
* **Prefix**!!**to the**db\_bin()**function** - This triggers the processing, or evaluation, of the function, which returns the complex formula.

df <- tbl(con, "flights") %>%

group\_by(x = !! db\_bin(sched\_dep\_time, bins = 10)) %>%

tally() %>%

collect()

head(df)

**#***# # A tibble: 6 x 2*

**#***# x n*

**#***# <dbl> <int>*

**#***# 1 782. 51999*

**#***# 2 557. 48864*

**#***# 3 1007. 38889*

**#***# 4 106. 1*

**#***# 5 331. 1861*

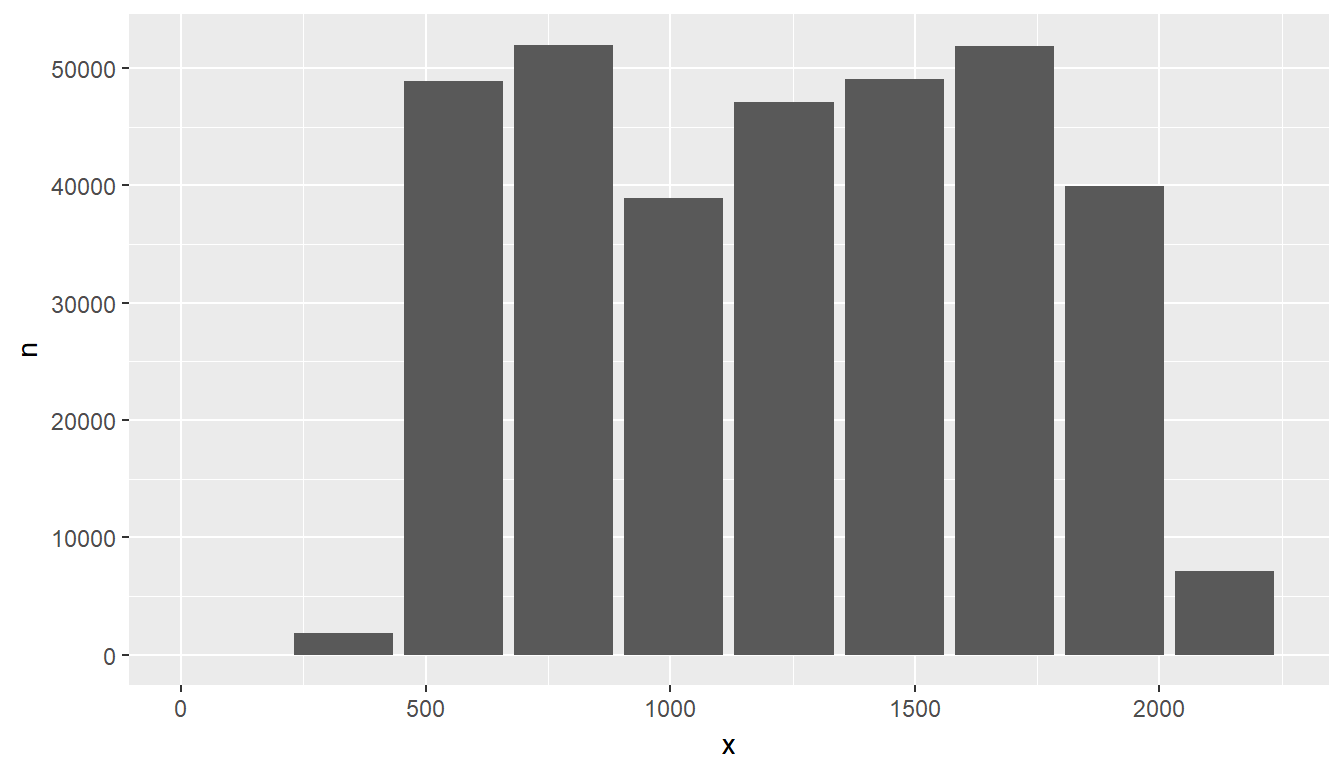
**#***# 6 1908. 39942*

### PLOTTING RESULTS IN R

Because the bins have been pre-processed on and collected from the database, the results are easily plotted using geom\_col(). The resulting bin values are x and the count per bin is y:

ggplot(data = df) +

geom\_col(aes(x = x, y = n))



### TRANSFORM AND PLOT

Just like with the Bar plot, the entire process can be piped. Here is an example of using the binwidth argument instead of bins; additionally, the bin size is widened to 300-minute intervals:

tbl(con, "flights") %>%

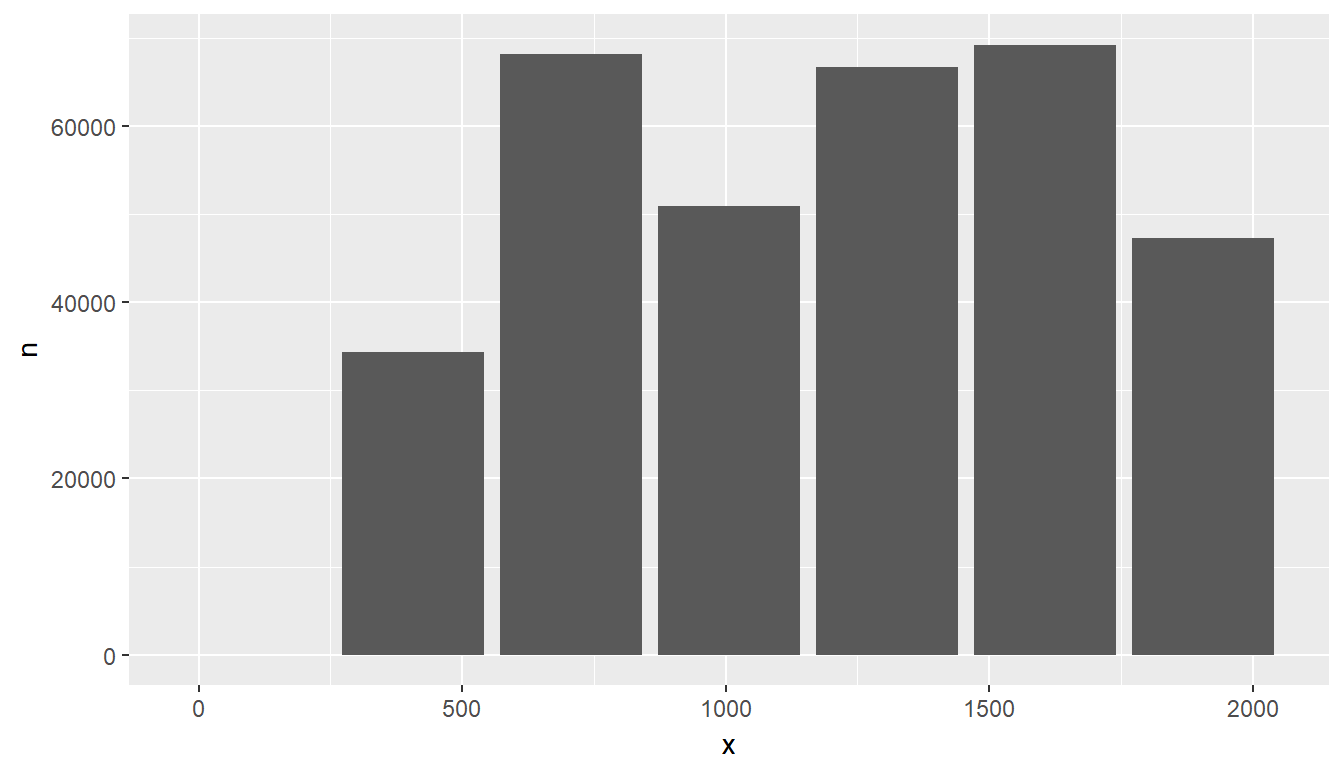
group\_by(x = !! db\_bin(sched\_dep\_time, binwidth = 300)) %>%

tally() %>%

collect() %>%

ggplot() +

geom\_col(aes(x = x, y = n))



## Raster Plot

To visualize two continuous variables, we typically resort to a Scatter plot. However, this may not be practical when visualizing millions or billions of dots representing the intersections of the two variables. A Raster plot may be a better option, because it concentrates the intersections into squares that are easier to parse visually.

A Raster plot basically does the same as a Histogram. It takes two continuous variables and creates discrete 2-dimensional bins represented as squares in the plot. It then determines either the number of rows inside each square or processes some aggregation, like an average.

### DATA TRANSFORMATION

The same helper function used to create the Histogram can be used to create the squares. The db\_bin() function is used for each continuous variable inside group\_by(), but in this case the number if bins is increased to 50:

df <- tbl(con, "flights") %>%

group\_by(

sc\_dep\_time = !! db\_bin(sched\_dep\_time, bins = 50),

sc\_arr\_time = !! db\_bin(sched\_arr\_time, bins = 50)

) %>%

summarise(avg\_distance = mean(distance)) %>%

collect()

**#***# Warning: Missing values are always removed in SQL.*

**#***# Use `AVG(x, na.rm = TRUE)` to silence this warning*

head(df)

**#***# # A tibble: 6 x 3*

**#***# # Groups: sc\_dep\_time [6]*

**#***# sc\_dep\_time sc\_arr\_time avg\_distance*

**#***# <dbl> <dbl> <dbl>*

**#***# 1 1953. 2170. 596.*

**#***# 2 2044. 2123. 201.*

**#***# 3 1728. 1887. 496.*

**#***# 4 1863. 2076. 687.*

**#***# 5 1638. 1793. 422.*

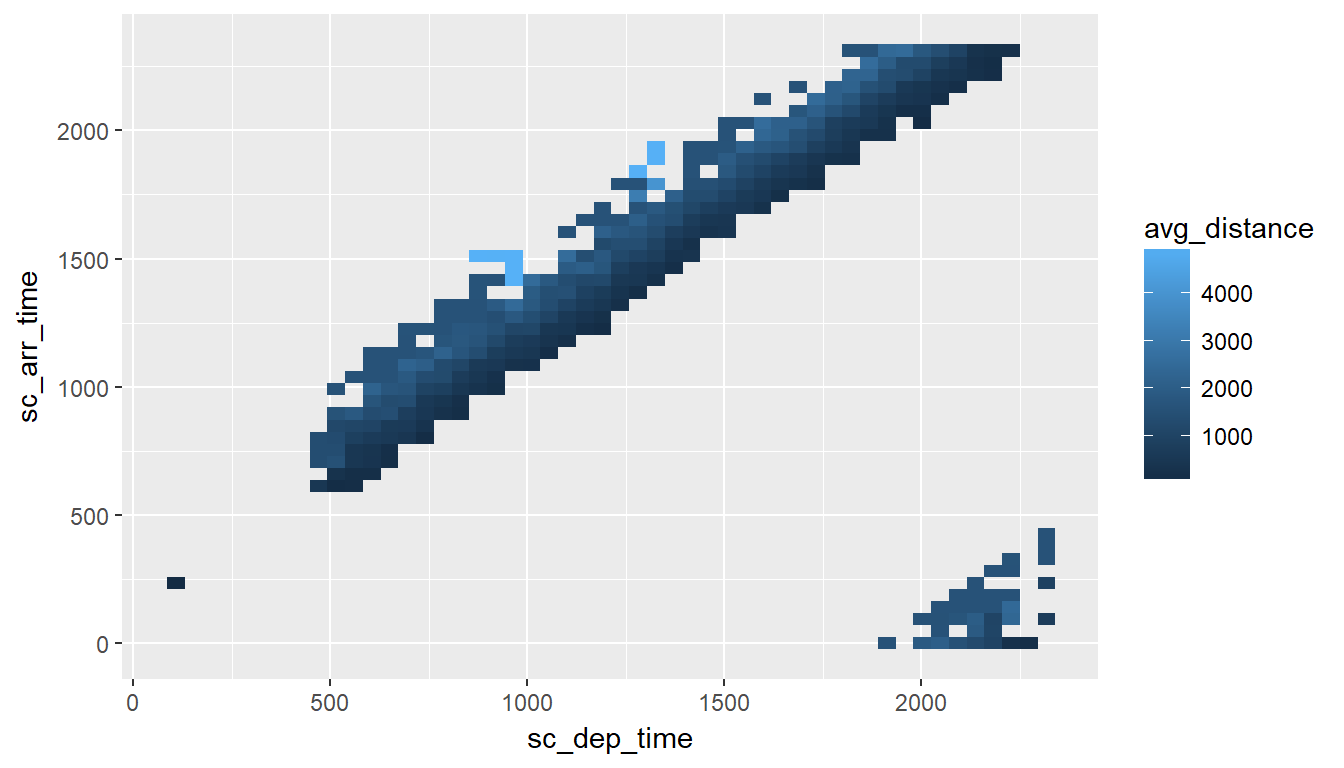
**#***# 6 737. 944. 593.*

### PLOTTING RESULTS IN R

The plot can now be built using geom\_raster(). Assigning x and y to each of the continuous variables will depend on what makes more sense for a given visualization. The result of each intersection is passed as the color of the square using fill.

ggplot(data = df) +

geom\_raster(aes(x = sc\_dep\_time, y = sc\_arr\_time, fill = avg\_distance))



### CONSIDERATIONS

There are two considerations when using a Raster plot with a database. Both considerations are related to the size of the results downloaded from the database:

* The number of bins requested: The higher the bins value is, the more data is downloaded from the database.
* How concentrated the data is: This refers to how many intersections return a value. The more intersections without a value, the less data is downloaded from the database.

In the previous example, there is a maximum of 2,500 rows (50 x 50). Because the data is highly concentrated, only 353 records are returned. This means that the data will be transmitted over the network quickly, but the trade-off is that the picture definition may not be ideal to gain insights about the data.

In the following example, the “definition” is set at 100 x 100. This improves the resolution but it quadruples the number of records that could potentially be downloaded.

tbl(con, "flights") %>%

group\_by(

sc\_dep\_time = !! db\_bin(sched\_dep\_time, bins = 100),

sc\_arr\_time = !! db\_bin(sched\_arr\_time, bins = 100)

) %>%

summarise(avg\_distance = mean(distance)) %>%

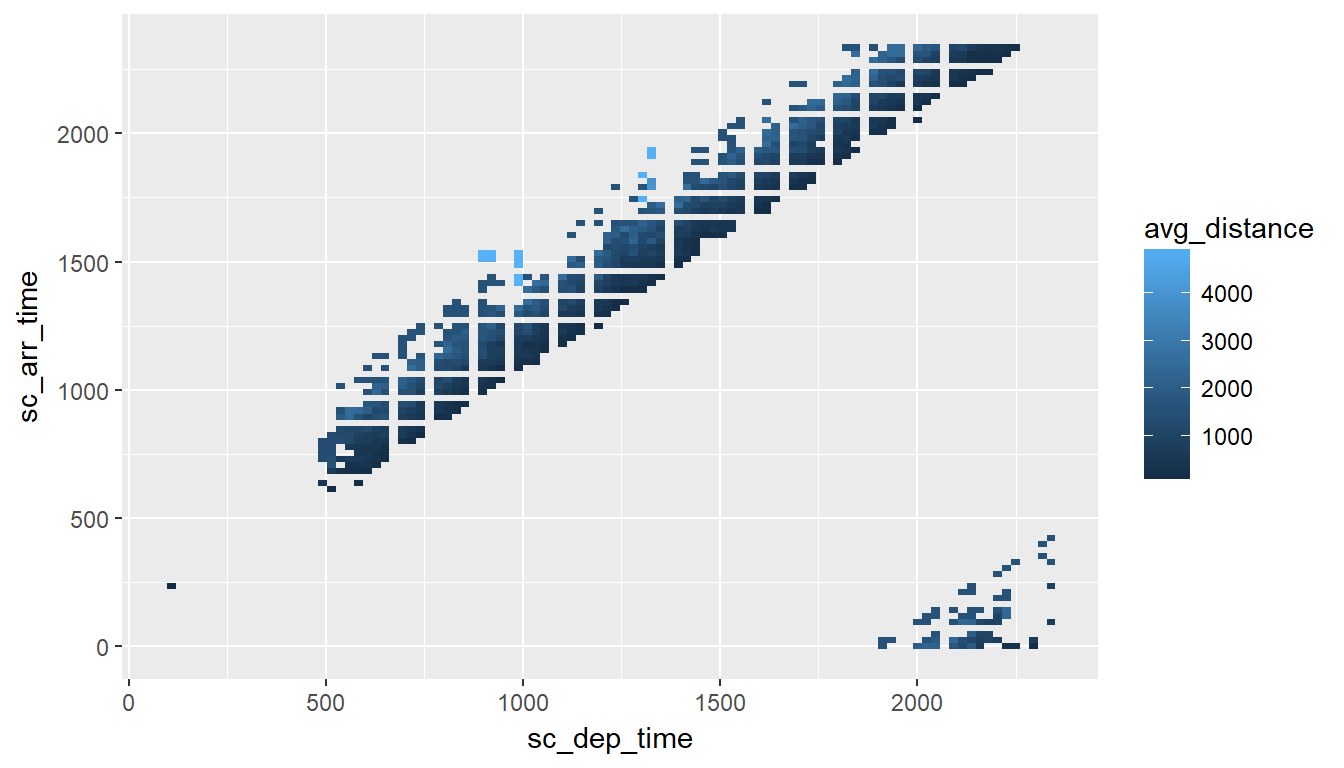
collect() %>%

ggplot() +

geom\_raster(aes(x = sc\_dep\_time, y = sc\_arr\_time, fill = avg\_distance))

**#***# Warning: Missing values are always removed in SQL.*

**#***# Use `AVG(x, na.rm = TRUE)` to silence this warning*



## Use an R package

The [dbplot](https://db.rstudio.com/dbplot) package provides helper functions that automate the aggregation and plotting steps. For more info, visit the [dbplot article](https://db.rstudio.com/dbplot) in this website.