M3T1_1.R

christiancobollogomez

2021-11-09

```
# install.packages("readr")
# install.packages("ggplot2")

library("readr")
library("ggplot2")

Data<-read.csv("cars.csv")

## Basic information

attributes(Data)#List your attributes within your data set.
```

```
## $names
## [1] "name.of.car"     "speed.of.car"    "distance.of.car"
##
## $class
## [1] "data.frame"
##
## $row.names
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
## [26] 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
```

```
summary(Data) #Prints the min, max, mean, median, and quartiles of each attribute.
```

```
##  name.of.car         speed.of.car  distance.of.car
##  Length:50          Min.   : 4.0   Min.   :  2.00
##  Class :character   1st Qu.:12.0   1st Qu.: 26.00
##  Mode  :character   Median :15.0   Median : 36.00
##                     Mean   :15.4   Mean   : 42.98
##                     3rd Qu.:19.0   3rd Qu.: 56.00
##                     Max.   :25.0   Max.   :120.00
```

```
str(Data) #Displays the structure of your data set.
```

```
## 'data.frame':    50 obs. of  3 variables:
##  $ name.of.car    : chr  "Ford" "Jeep" "Honda" "KIA" ...
##  $ speed.of.car   : int  4 4 7 7 8 9 10 10 10 11 ...
##  $ distance.of.car: int  2 4 10 10 14 16 17 18 20 20 ...
```

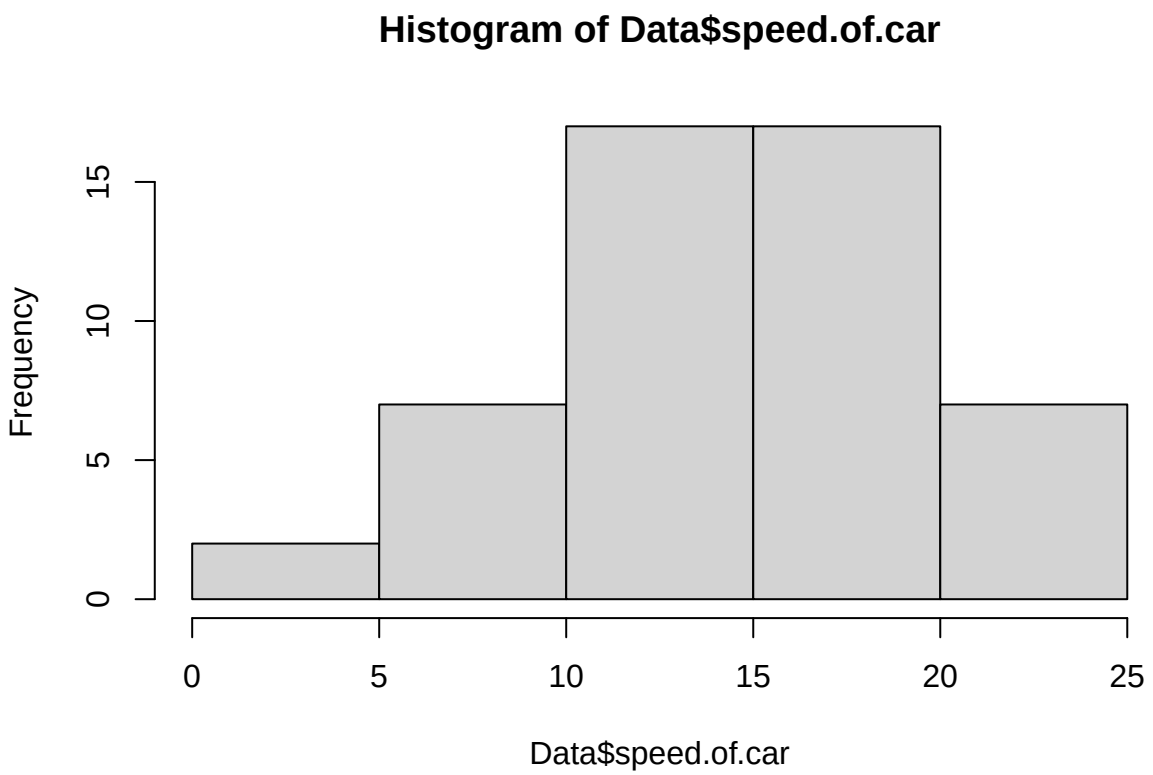
```
names(Data) #Names your attributes within your data set.
```

```
## [1] "name.of.car"     "speed.of.car"    "distance.of.car"
```

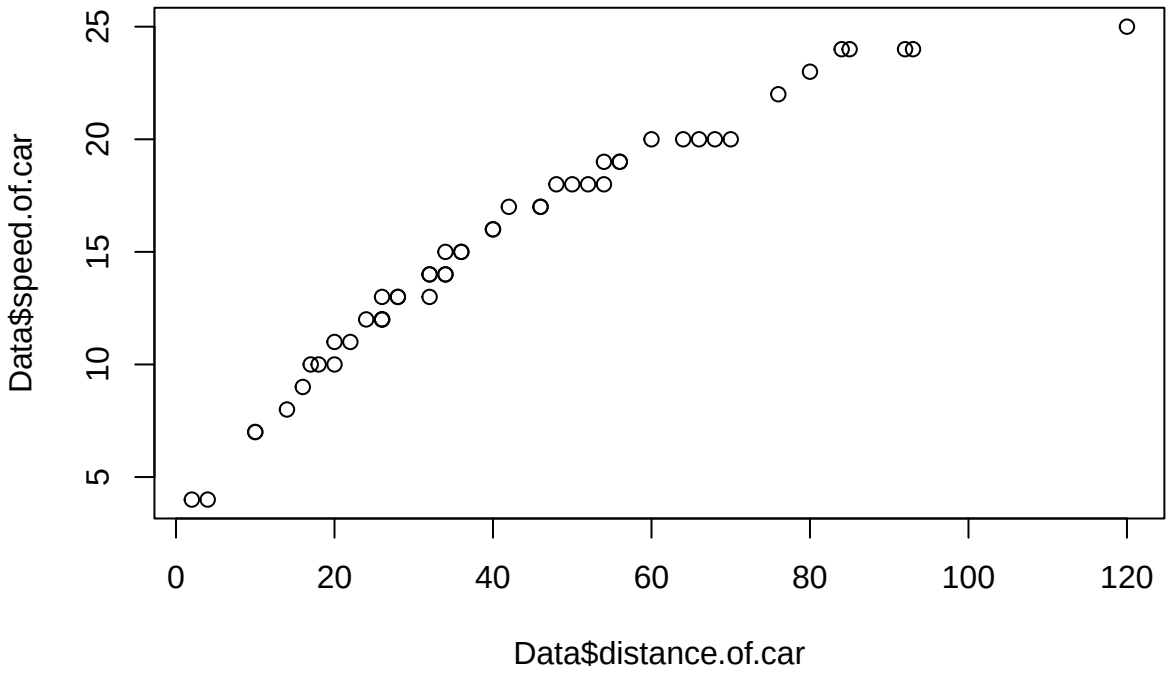
```
# Data$ColumnName Will print out the instances within that particular column in your data set.
Data$speed.of.car
```

```
##  [1]  4  4  7  7  8  9 10 10 10 11 11 12 12 12 12 13 13 13 13 14 14 14 14 15 15
## [26] 15 16 16 17 17 17 18 18 18 18 19 19 19 20 20 20 20 20 22 23 24 24 24 24 25
```

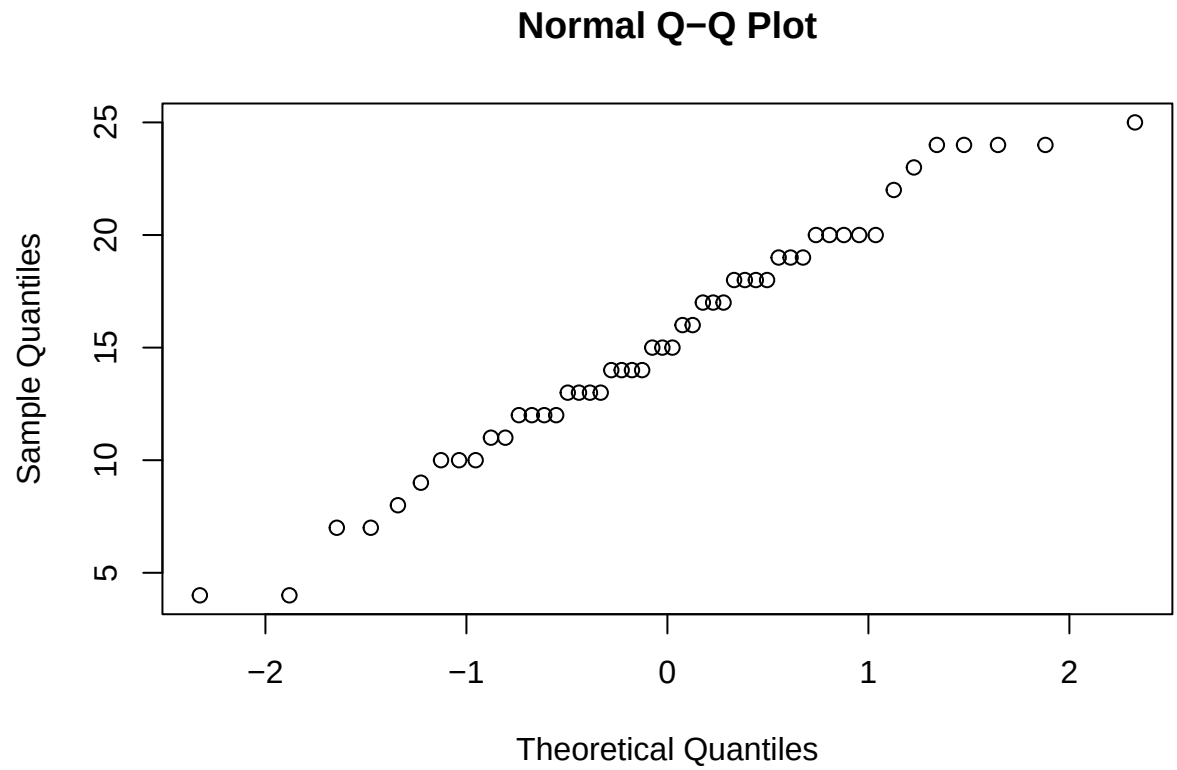
```
## Plots

# Histogram Plot

hist(Data$speed.of.car)
```




```
# Scatter (Box) Plot

plot(Data$distance.of.car,Data$speed.of.car)
```




```
# Normal Quantile Plot- is a way to see if your data is normally distributed.

qqnorm(Data$speed.of.car)
```




```
## Manage Data

# Do you see any data types that need changing within your data set? If so, how do
# you convert data types? Converting data types is a helpful skill to learn for
```

```
# this tutorial and future analyses. Here is an example
# of how one would change a column's data type within a data set:

#Data$ColumnName<-as.typeofdata(Data$ColumnName)


# Do the columns/attributes within your dataset need renaming?
# Pick short names, so you'll not have typing/spelling errors. You'll also want
# to name your columns in order as they appear in your dataset.
# To rename the attributes/columns in your dataset, you'll want to use the
# c() function, specifying a name for each column.

names(Data)<-c("name","speed","dist")


# Do any of your variables have missing values? How do you know if your dataset
# has any missing values? If you do not address missing values certain functions
# will not work properly, so it's smart to start the practice checking for
# missing values.  R labels missing as NA (Not Available).
# Here are two ways to know if you have any missing values:

summary(Data) #Will count how many NA's you have.
```

```
##      name               speed           dist
##  Length:50          Min.   : 4.0   Min.   :  2.00
##  Class :character   1st Qu.:12.0   1st Qu.: 26.00
##  Mode  :character   Median :15.0   Median : 36.00
##                     Mean   :15.4   Mean   : 42.98
##                     3rd Qu.:19.0   3rd Qu.: 56.00
##                     Max.   :25.0   Max.   :120.00
```

```
is.na(Data) #Will show your NA's through logical data. (TRUE if it's missing,
```

```
##        name speed  dist
##  [1,] FALSE FALSE FALSE
##  [2,] FALSE FALSE FALSE
##  [3,] FALSE FALSE FALSE
##  [4,] FALSE FALSE FALSE
##  [5,] FALSE FALSE FALSE
##  [6,] FALSE FALSE FALSE
##  [7,] FALSE FALSE FALSE
##  [8,] FALSE FALSE FALSE
##  [9,] FALSE FALSE FALSE
## [10,] FALSE FALSE FALSE
## [11,] FALSE FALSE FALSE
## [12,] FALSE FALSE FALSE
## [13,] FALSE FALSE FALSE
## [14,] FALSE FALSE FALSE
## [15,] FALSE FALSE FALSE
## [16,] FALSE FALSE FALSE
## [17,] FALSE FALSE FALSE
## [18,] FALSE FALSE FALSE
## [19,] FALSE FALSE FALSE
## [20,] FALSE FALSE FALSE
```

```
## [21,] FALSE FALSE FALSE
## [22,] FALSE FALSE FALSE
## [23,] FALSE FALSE FALSE
## [24,] FALSE FALSE FALSE
## [25,] FALSE FALSE FALSE
## [26,] FALSE FALSE FALSE
## [27,] FALSE FALSE FALSE
## [28,] FALSE FALSE FALSE
## [29,] FALSE FALSE FALSE
## [30,] FALSE FALSE FALSE
## [31,] FALSE FALSE FALSE
## [32,] FALSE FALSE FALSE
## [33,] FALSE FALSE FALSE
## [34,] FALSE FALSE FALSE
## [35,] FALSE FALSE FALSE
## [36,] FALSE FALSE FALSE
## [37,] FALSE FALSE FALSE
## [38,] FALSE FALSE FALSE
## [39,] FALSE FALSE FALSE
## [40,] FALSE FALSE FALSE
## [41,] FALSE FALSE FALSE
## [42,] FALSE FALSE FALSE
## [43,] FALSE FALSE FALSE
## [44,] FALSE FALSE FALSE
## [45,] FALSE FALSE FALSE
## [46,] FALSE FALSE FALSE
## [47,] FALSE FALSE FALSE
## [48,] FALSE FALSE FALSE
## [49,] FALSE FALSE FALSE
## [50,] FALSE FALSE FALSE
```

```
# FALSE if it's not.)

# How to address missing values? There are multiple ways to confront missing
# values in your dataset - all depend on how much they will affect your dataset.
# Here are a few options:

# Remove any observations containing missing data. (If the missing data is less
# than 10% of the total data and only after comparing the min/max of all
# the features both with and without the missing data.)

# na.omit(Data$ColumnName) #Drops any rows with missing values and omits them forever.

# na.exclude(Data$ColumnName) #Drops any rows with missing values, but keeps
# track of where they were.

# Replace the missing values with the mean, which is common technique, but
# something to use with care with as it can skew the data.

# Data$ColumnName[is.na(Data$ColumnName)]<-mean(Data$ColumnName,na.rm = TRUE)


## Creating Testing and Training sets.
```

```r
set.seed(123)


# How do you split the data into training and test sets? You'll now want to split
# your data into two sets for modeling. One is the training set and the other
# one being the test set. A common split is 70/30, which means that 70% of the
# data will be the training set's size and 30% of the data will be the test
# set's size. You will be using the 70/30 split, but another common split is 80/20.

# Setting the training set's size and the testing set's size can be done by
# performing these two lines of code. These two lines calculate the sizes
# of each set but do not create the sets:

trainSize<-round(nrow(Data)*0.7)

testSize<-nrow(Data)-trainSize

trainSize
```

```
## [1] 35
```

```r
testSize
```

```
## [1] 15
```

```r
# How do you create the training and test sets? It's now time for you to create
# the training and test sets. We also want these sets to be in a randomized order,
# which will create the most optimal model.

# To perform this, you need to run these three lines of code. Type in this code
# into R Script or Console:

training_indices<-sample(seq_len(nrow(Data)),size =trainSize)

trainSet<-Data[training_indices,]

testSet<-Data[-training_indices,]


# You're now ready to run your data through your modeling algorithm. The model
# that we will be using is the Linear Regression Model, which is helpful when
# trying to discover the relationship between two variables. These two variables
# represent the X and Y within the linear equation. The X variable is the predictor
# variable, also known as the independent variable because it doesn't depend on
# other attributes while making predictions. Y is the response variable, also
# known as the dependent variable because its value depends on the other variables.
# (We will be keeping this at a high level. If you'd like to discover more about
# this equation, please feel free to do your own research.) In our case, these
# two variables will be Speed and Distance. We are trying to predict Distance,
# so it is our dependent/response/Y variable. Speed is our independent/predictor/X
# variable.

# To create this model, we will be using the linear model function - lm(). Here
# is the basic line of code for the
# linear model function.
```

```r
LRmodel<-lm(dist~ speed, trainSet)

# The code above is without any parameter's or adjustments. If you'd like to
# experiment with options, this is a perfect time to do so. Did you create an
# optimal model? To see key metrics of your model, type in this code into R Script
# or Console:

summary(LRmodel)
```

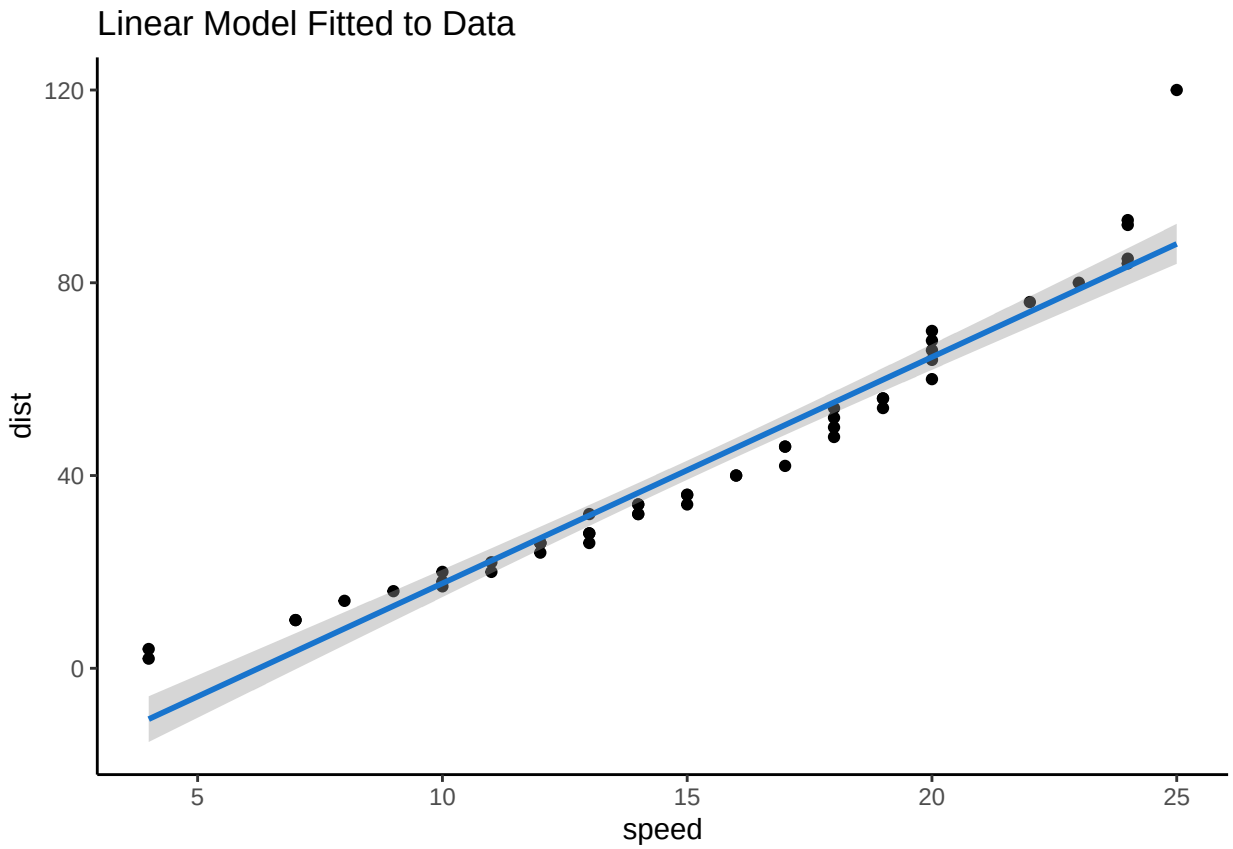
```
##
## Call:
## lm(formula = dist ~ speed, data = trainSet)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -9.0012 -5.0012 -0.5603  2.1458 28.4109
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -35.2481     4.0712  -8.658 5.25e-10 ***
## speed         5.0735     0.2519  20.143  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 7.18 on 33 degrees of freedom
## Multiple R-squared:  0.9248, Adjusted R-squared:  0.9225
## F-statistic: 405.7 on 1 and 33 DF,  p-value: < 2.2e-16
```

```r
## Predictions
# The next step is to predict the cars distances through the speed of the cars.
# To do this, we'll be using the
# prediction function - predict()

PredictLR <- predict(LRmodel,testSet)

# To view your predictions, type in this code into R Script or Console:

PredictLR
```

```
##         1         2         6        16        18        20        22        23
## -14.95415 -14.95415  10.41329  30.70724  30.70724  35.78073  35.78073  35.78073
##        34        35        38        39        44        46        47
##  56.07468  56.07468  61.14817  66.22166  76.36864  86.51561  86.51561
```

```r
## We represent the model vs the scatter plot of the data.
ggplot(data = Data, aes(x = speed, y = dist)) +
  geom_point() +
  stat_smooth(method = "lm", col = "dodgerblue3") +
  theme(panel.background = element_rect(fill = "white"),
        axis.line.x=element_line(),
        axis.line.y=element_line()) +
  ggtitle("Linear Model Fitted to Data")
```

```
## `geom_smooth()` using formula 'y ~ x'
```


Linear Model Fitted to Data
120
80
40
0
dist
5
10
15
20
25
speed