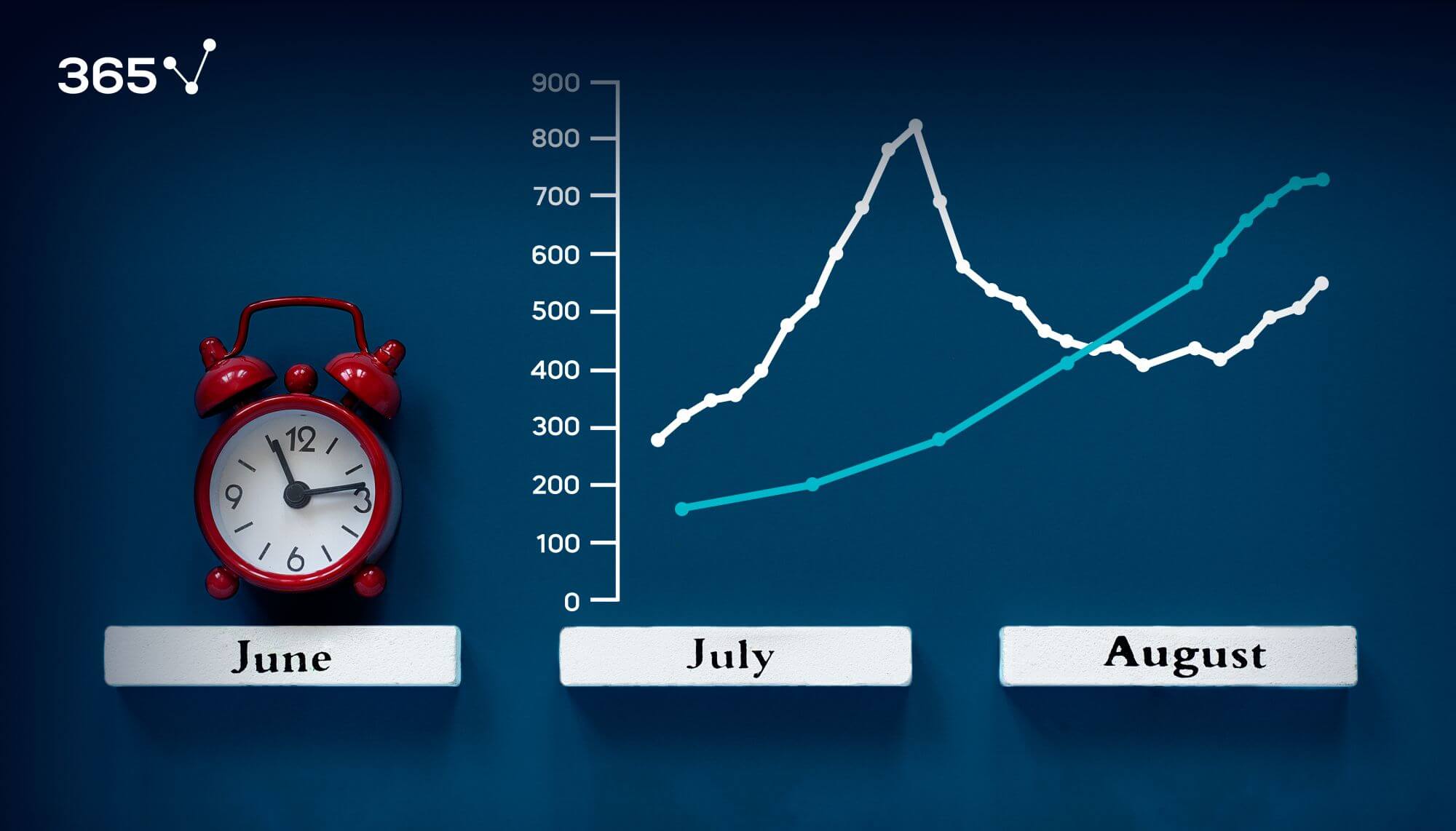
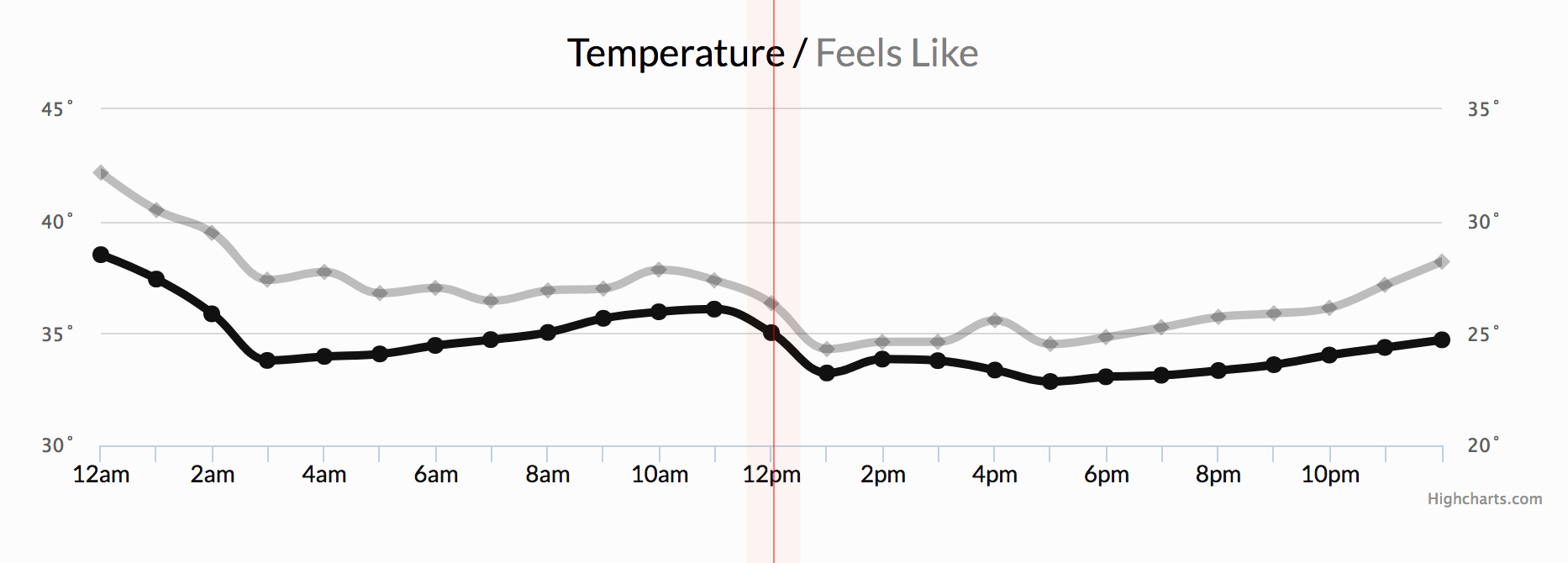
**Module-2**

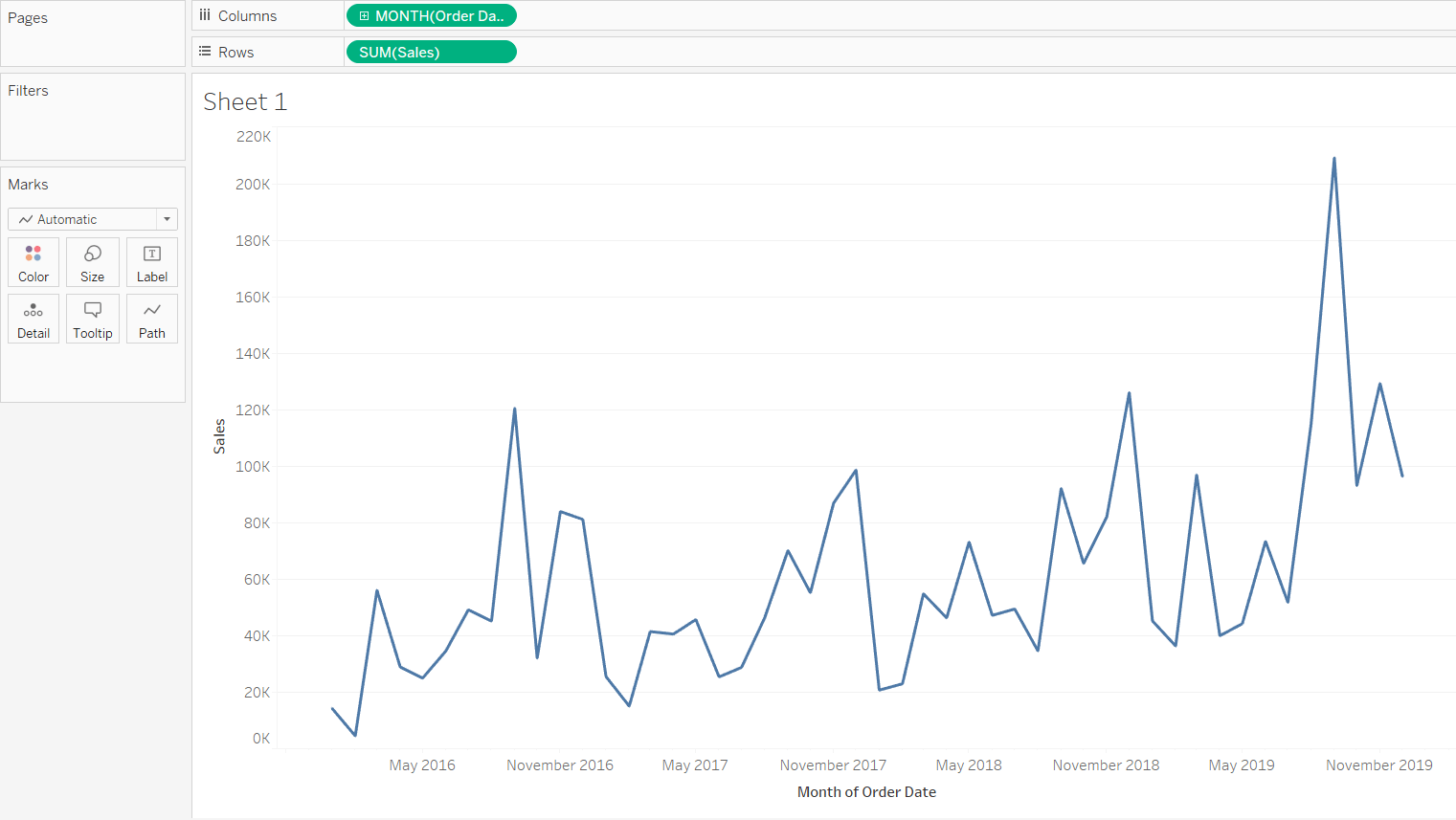
Time series analysis is a specific way of analyzing a sequence of data points collected over an interval of time. In time series analysis, analysts record data points at consistent intervals over a set period of time rather than just recording the data points intermittently or randomly. However, this type of analysis is not merely the act of collecting data over time.



Time series analysis typically requires a large number of data points to ensure consistency and reliability. An extensive data set ensures you have a representative sample size and that analysis can cut through noisy data. It also ensures that any trends or patterns discovered are not outliers and can account for seasonal variance. Additionally, time series data can be used for forecasting—predicting future data based on historical data.



Time series analysis is used for non-stationary data—things that are constantly fluctuating over time or are affected by time. Industries like finance, retail, and economics frequently use time series analysis because currency and sales are always changing. Stock market analysis is an excellent example of time series analysis in action, especially with automated trading algorithms. Likewise, time series analysis is ideal for forecasting weather changes, helping meteorologists predict everything from tomorrow’s weather report to future years of climate change.



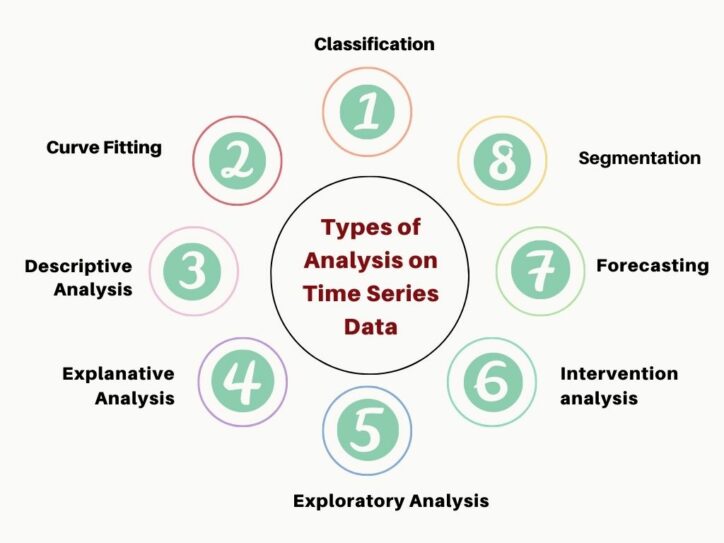
**Examples** of time series analysis in action include:

* Weather data
* Rainfall measurements
* Temperature readings
* Heart rate monitoring (EKG)
* Brain monitoring (EEG)
* Quarterly sales
* Stock prices
* Automated stock trading
* Industry forecasts
* Interest rates

## **Time Series Analysis Types**

Models of time series analysis include:

* **Classification:** Identifies and assigns categories to the data.
* **Curve fitting:** Plots the data along a curve to study the relationships of variables within the data.
* **Descriptive analysis:** Identifies patterns in time series data, like trends, cycles, or seasonal variation.
* **Explanative analysis:** Attempts to understand the data and the relationships within it, as well as cause and effect.
* **Exploratory analysis:** Highlights the main characteristics of the time series data, usually in a visual format.
* **Forecasting:** Predicts future data. This type is based on historical trends. It uses the historical data as a model for future data, predicting scenarios that could happen along future plot points.
* **Intervention analysis:** Studies how an event can change the data.
* **Segmentation:** Splits the data into segments to show the underlying properties of the source information.



### **Data classification**

Further, time series data can be classified into two main categories:

* **Stock time series data** means measuring attributes at a certain point in time, like a static snapshot of the information as it was.
* **Flow time series data** means measuring the activity of the attributes over a certain period, which is generally part of the total whole and makes up a portion of the results.

### **Data variations**

In time series data, variations can occur sporadically throughout the data:

* **Functional analysis** can pick out the patterns and relationships within the data to identify notable events.
* **Trend analysis** means determining consistent movement in a certain direction. There are two types of trends: deterministic, where we can find the underlying cause, and stochastic, which is random and unexplainable.
* **Seasonal variation** describes events that occur at specific and regular intervals during the course of a year. Serial dependence occurs when data points close together in time tend to be related.

### Time Series Analysis: Definition, Types & Techniques | Tableau

### **Important Considerations for Time Series Analysis**

While time series data is data collected over time, there are different types of data that describe how and when that time data was recorded. For example:

* **Time series data** is data that is recorded over consistent intervals of time.
* **Cross-sectional data** consists of several variables recorded at the same time.
* **Pooled data** is a combination of both time series data and cross-sectional data.

# Processing data with R

* Steps in Data Preprocessing
  + Step 1: Importing the Dataset
  + Step 2: Handling the Missing Data
* Step 3: Encoding Categorical Data.
  + Output
* Step 4: Splitting the Dataset into the Training and Test sets
  + Training set
  + Test set
* Step 5: Feature Scaling
  + training\_set
  + test\_set

#### **Step 1: Importing the dataset**

Before we start preparing our data, first we need to dowload it from [here](https://github.com/Daniel695/data) and load it in RStudio IDE.

Here is how to achieve this.

Dataset = read\_csv('data.csv')

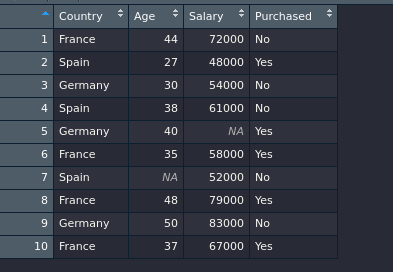
This code imports our data stored in CSV format.

We can have a look at our data using the ‘view()’ function:

view(Dataset)

Upon executing we obtain our dataset as below.

**Output**



#### **Step 2: Handling the missing data**

From the dataset, the Age and Salary column report missing data. Before implementing our machine learning models, this problem needs to be solved, otherwise it will cause a serious problem to our machine learning models. Therefore, it’s our responsibility to ensure this missing data is eliminated from our dataset using the most appropriate technique.

Here are two techniques we can use to handle missing data:

1. **Delete the observation reporting the missing data:**

This technique is suitable when dealing with big datasets and with very few missing values i.e. deleting one row from a dataset with thousands of observations can not affect the quality of the data. When the dataset reports many missing values, it can be very dangerous to use this technique. Deleting many rows from a dataset can lead to the loss of crucial information contained in the data.

To ensure this does not happen, we make use of an appropriate technique that has no harm to the quality of the data.

1. **Replace the missing data with the average of the feature in which the data is missing:**

This technique is the best way so far to deal with the missing values. Many statisticians make use of this technique over that of the first one.

Now that we know the techniques used to treat the missing data, let’s solve this problem from our data. In our case, we shall make use of the second technique.

Let’s start by replacing the missing data in the Age column with the mean of that column.

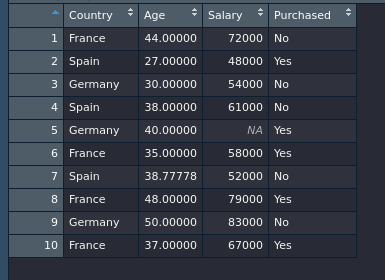
The code below carries out such a task.

Dataset$Age = ifelse(is.na(Dataset$Age),

ave(Dataset$Age, FUN = function (x)mean(x, na.rm = TRUE)),

Dataset$Age)

Executing the code we obtain:



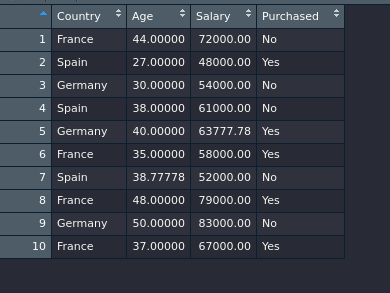
The missing value that was in the Age column of our data set has successfully been replaced with the mean of the same column.

We do the same for the Salary column by executing the code below:

Dataset$Salary = ifelse(is.na(Dataset$Salary),

ave(Dataset$Salary, FUN = function (x)mean(x, na.rm = TRUE)),

Dataset$Salary)



The missing value that was in the Salary column was successfully replaced with the mean of the same column.

### **Step 3: Encoding categorical data**

Encoding refers to transforming text data into numeric data. Encoding Categorical data simply means we are transforming data that fall into categories into numeric data.

In our dataset, the Country column is Categorical data with 3 levels i.e. France, Spain, and Germany. The purchased column is Categorical data as well with 2 categories, i.e. YES and NO.

The machine models we built on our dataset are based on mathematical equations and it’s only take numbers in those equations.

Keeping texts of a categorical variable in the equation can cause some troubles to the machine learning models and this why we encode those variables. To transform a categorical variable into numeric, we use the factor() function.

Let start by encoding the Country column.

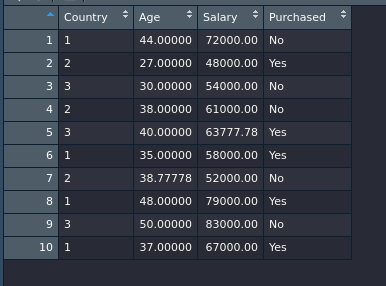
Dataset$Country = factor(Dataset$Country,

levels = c('France','Spain','Germany'),

labels = c(1.0, 2.0 , 3.0 ))

Executing the code above we obtain.

**Output**



Our country names were successfully replaced with numbers.

We do the same for the purchased column.

Dataset$Purchased = factor(Dataset$Purchased,

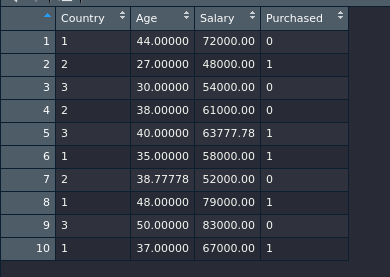
levels = c('No', 'Yes'),

labels = c(0, 1))

Dataset$Purchased[is.na(Dataset$Purchased)] <- 0

as.factor(Dataset$Purchased)

Using the view() function we obtain.



Our purchased column was successfully encoded into 0,s, and 1,s.

### **Step 4: Splitting the dataset into the training and test set**

In machine learning, we split data into two parts:

* Training set: The part of the data that we implement our machine learning model on.
* Test set: The part of the data that we evaluate the performance of our machine learning model on.

The reason we split this data is to ensure that our machine learning model does not overlearn the correlation of data it’s trained on. If we let it learn too much on the data, it may perform poorly when tested on a new dataset with a different correlation.

Therefore, whenever we are building a machine learning model, the idea is to implement it on the training set and evaluate it on the test set. We expect the performance in the training set and test set to be different and if this is the case the model can adapt to new datasets.

Using our dataset, let’s split it into the training and test sets.

To begin with, we first load the required library.

library(caTools)# required library for data splition

set.seed(123)

split = sample.split(Dataset$Purchased, SplitRatio = 0.8)# returns true if observation goes to the Training set and false if observation goes to the test set.

#Creating the training set and test set separately

training\_set = subset(Dataset, split == TRUE)

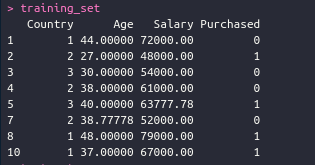
test\_set = subset(Dataset, split == FALSE)

training\_set

test\_set

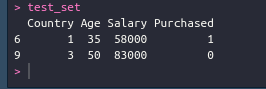
Executing our code yields:

**Training Set**:



From the results it clear that eight observations, 0.8 of our dataset observations, were split into the training set.

**Test Set**:



From the output it clear that two observations went to the test set.

### **Step 5: Feature scaling**

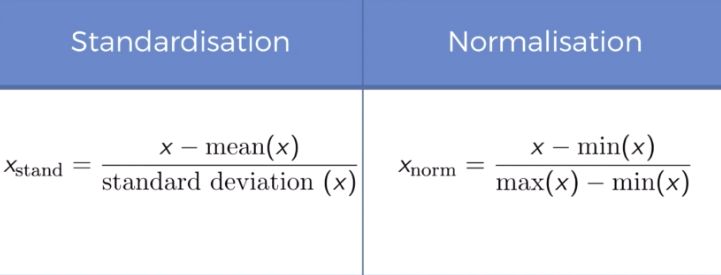
It’s a common case that in most datasets, features also known as inputs, are not on the same scale. Many machine learning models are Euclidian distant-based.

It happens that, the features with the large units dominate those with small units when it comes to calculation of the Euclidian distance and it will be as if those features with small units do not exist.

To ensure this does not occur, we need to encode our features so that they all fall in the range between -3 and 3. There are several ways we can use to scale our features. The most used one is the standardization and normalization technique.

The normalization technique is used when the data is normally distributed while standardization works with both normally distributed and the data that is not normally distributed.

The formula for these two techniques is shown below.



Now, let’s scale both the training set and test set of our dataset separately.

Here is how we achieve this:

training\_set[, 2:3] = scale(training\_set[, 2:3])

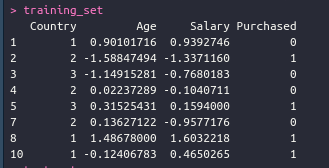
test\_set[, 2:3] = scale(test\_set[, 2:3])

training\_set

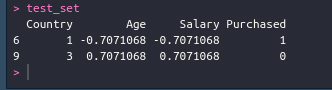
test\_set

Executing our code we obtain:

**Training Set**:



**Test Set**:



Our training and test set were successfully scaled.

Note that in our code we specified the columns to be scale.

If we fail to do so, R will show us an error.

Such as:

training\_set = scale(training\_set)# returns an error

The reason is that our encoded columns are not treated as numeric entries.

# R to Organize and Manipulate Data

## **Creating a Data Frame**

To create a data frame we begin by creating vectors for each of the variables. Note that letters is a constant in R that contains the 26 lower case letters of the Roman alphabet: here we are using just the first six letters for the bag ids.

bag\_id = letters[1:6]  
year = c(2006, 2006, 2000, 2000, 1994, 1994)  
weight = c(1.74, 1.74, 0.80, 0.80, 10.0, 10.0)  
type = c("peanut", "peanut", "plain", "plain", "plain", "plain")   
number\_yellow = c(2, 3, 1, 5, 56, 63)  
percent\_red = c(27.8, 4.35, 22.7, 20.8, 23.0, 21.9)  
total = c(18, 23, 22, 24, 331, 333)  
rank = c("sixth", "fourth", "fifth", "third", "second", "first")

To create the data frame, we use R’sdata.frame()function, passing to it the names of our vectors, each of which must be of the same length. There is an option within this function to treat variables whose values are character strings as factors—another name for a categorical variable—by using the argument stringsAsFactors = TRUE. As the default value for this argument depends on your version of R, it is useful to make your choice explicit by including it in your code, as we do here.

mm\_data = data.frame(bag\_id, year, weight, type, number\_yellow, percent\_red, total, rank, stringsAsFactors = TRUE)  
mm\_data

bag\_id year weight type number\_yellow percent\_red total rank

1 a 2006 1.74 peanut 2 27.80 18 sixth

2 b 2006 1.74 peanut 3 4.35 23 fourth

3 c 2000 0.80 plain 1 22.70 22 fifth

4 d 2000 0.80 plain 5 20.80 24 third

5 e 1994 10.00 plain 56 23.00 331 second

6 f 1994 10.00 plain 63 21.90 333 first

If we examine the structure of this data set using R’sstr()function, we see that bag\_id, type, and rank are factors and year, weight, number\_yellow, percent\_red, and total arenumerical variables, assignments that are consistent with our earlier analysis of the data.

str(mm\_data)

'data.frame': 6 obs. of 8 variables:

$ bag\_id : Factor w/ 6 levels "a","b","c","d",..: 1 2 3 4 5 6

$ year : num 2006 2006 2000 2000 1994 ...

$ weight : num 1.74 1.74 0.8 0.8 10 10

$ type : Factor w/ 2 levels "peanut","plain": 1 1 2 2 2 2

$ number\_yellow: num 2 3 1 5 56 63

$ percent\_red : num 27.8 4.35 22.7 20.8 23 21.9

$ total : num 18 23 22 24 331 333

$ rank : Factor w/ 6 levels "fifth","first",..: 5 3 1 6 4 2

Finally, we can use the functionas.factor()to have R treat a numerical variable as a categorical variable, as we do here for year. Why we might wish to do this is a topic we will return to in later chapters.

mm\_year\_as\_factor = data.frame(bag\_id, as.factor(year), percent\_red, total)

str(mm\_year\_as\_factor)

'data.frame': 6 obs. of 4 variables:

$ bag\_id : Factor w/ 6 levels "a","b","c","d",..: 1 2 3 4 5 6

$ as.factor.year.: Factor w/ 3 levels "1994","2000",..: 3 3 2 2 1 1

$ percent\_red : num 27.8 4.35 22.7 20.8 23 21.9

$ total : num 18 23 22 24 331 33

## Creating a New Data Frame by Subsetting an Existing Data Frame

In Chapter 1.2 we learned how to retrieve individual rows or columns from a data frame and assign them to a new object. Here we learn how to use R’s more flexible subset() function to accomplish the same thing. Here, for example, we retrieve only the data for plain M&Ms.

plain\_mm = subset(mm\_data, type == "plain")

plain\_mm

bag\_id year weight type number\_yellow percent\_red total rank

3 c 2000 0.8 plain 1 22.7 22 fifth

4 d 2000 0.8 plain 5 20.8 24 third

5 e 1994 10.0 plain 56 23.0 331 second

6 f 1994 10.0 plain 63 21.9 333 first

Note that type == "plain"uses a relational operator to choose only those rows in which the variable type has the value plain. Here is a list of relational operators:

|  |  |
| --- | --- |
| Table 2.2.2 | |
|  | |
| . Relational Operators in R. | | | |
| **operator** | **usage** | | **meaning** |
| < | x < y | | x is less than y |
| > | x > y | | x is greater than y |
| <= | x <= y | | x is less than or equal to y |
| >= | x >= y | | x is greater than or equal to y |
| == | x == y | | x is exactly equal to y |
| != | x != y | | x is not equal to y |

We can string variables together using the logical & operator.

mm\_plain10 = subset(mm\_data, (weight == 10.0 & type == "plain"))

mm\_plain10

bag\_id year weight type number\_yellow percent\_red total rank

5 e 1994 10 plain 56 23.0 331 second

6 f 1994 10 plain 63 21.9 333 first

We also can narrow the number of variables returned using the subset() function’s select argument. In this example we exclude samples collected before the year 2000 and return only the year, the number of yellow M&Ms, and the percentage of red M&Ms.

mm\_20xx = subset(mm\_data, year >= 2000, select = c(year, number\_yellow, percent\_red))

mm\_20xx

year number\_yellow percent\_red

1 2006 2 27.80

2 2006 3 4.35

3 2000 1 22.70

4 2000 5 20.80

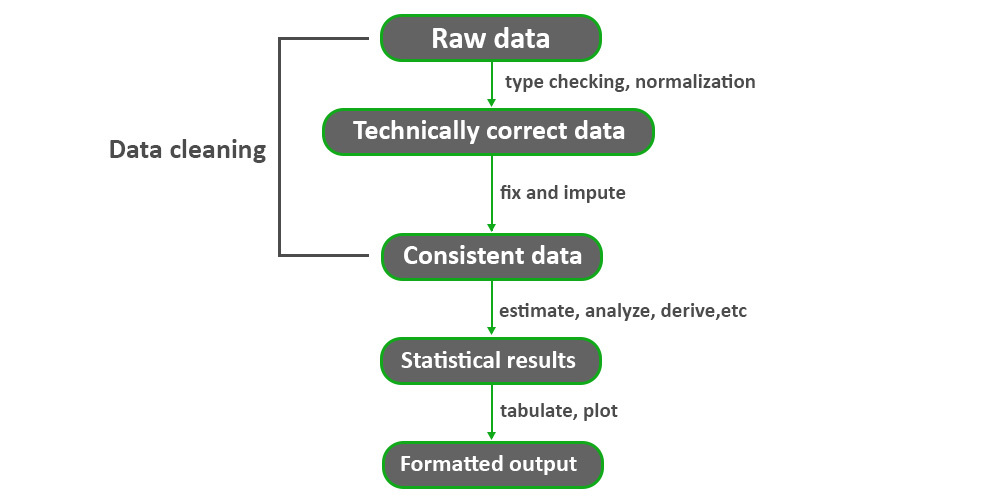
## **Data Cleaning in R**

Data Cleaning is the process to transform raw data into consistent data that can be easily analyzed. It is aimed at filtering the content of statistical statements based on the data as well as their reliability. Moreover, it influences the statistical statements based on the data and improves your data quality and overall productivity.

## **Purpose of Data Cleaning**

The following are the various purposes of data cleaning:

* Eliminate Errors
* Eliminate Redundancy
* Increase Data Reliability
* Delivery Accuracy
* Ensure Consistency
* Assure Completeness
* Standardize your approach



## Let’s Start the implementation of Data Cleaning in R

For this, we will use inbuilt datasets(air quality datasets) which are available in R.

|  |
| --- |
| head(airquality) |

**Output:**



In the above dataset, we can clearly see the NA value inside the columns which will generate the error or not produce the accurate predictions for Machine Learning Model.

## **Handling missing value in R**

To handle the missing value we will check the columns of the datasets, if we found some missing data inside the columns then this generates the NA values as an output, which can be not good for every model. So let’s check it using [mean()](https://www.geeksforgeeks.org/calculate-arithmetic-mean-in-r-programming-mean-function/) methods.

|  |
| --- |
| mean(airquality$Solar.R) |

**Output:**

<NA>

**Checking another column**

|  |
| --- |
| mean(airquality$Ozone) |

**Output:**

<NA>

**Checking another column**

Here we get the mean value of Wind Columns which means it doesn’t have any missing value in this column.

|  |
| --- |
| mean(airquality$Wind) |

**Output:**

9.95751633986928

### **Handling NA values**

Handling NA value using na.rm in both columns.

|  |
| --- |
| mean(airquality$Solar.R, na.rm = TRUE) |

**Output:**

185.931506849315

Also performing the same operation on another column.

|  |
| --- |
| mean(airquality$Ozone, na.rm = TRUE) |

**Output:**

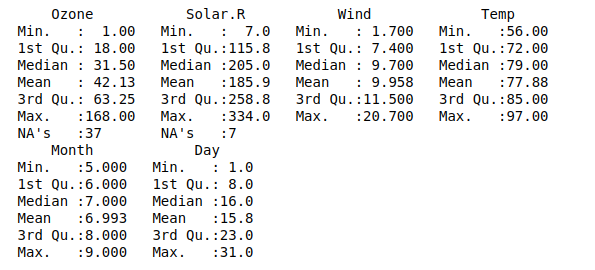
42.1293103448276

## **Data Cleaning Operation**

After checking the summary of the dataset and we found the  number on NA in two columns(Ozone and Solar.R)

|  |
| --- |
| summary(airquality) |

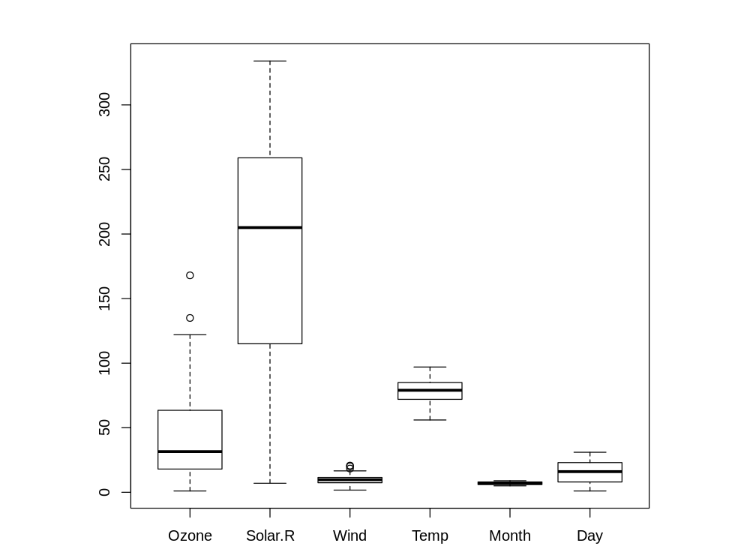
**Output:**



We can get a clear visual of the irregular data using a boxplot.

|  |
| --- |
| boxplot(airquality) |

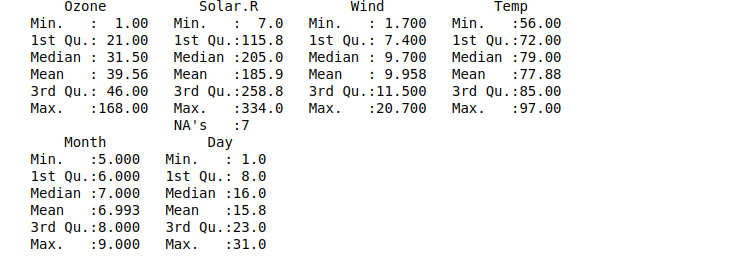
**Output:**



**Removing irregularities data with** [**is.na()**](https://www.geeksforgeeks.org/how-to-use-is-na-in-r/) **methods.**

|  |
| --- |
| New\_df = airquality    New\_df$Ozone = ifelse(is.na(New\_df$Ozone),                        median(New\_df$Ozone,                               na.rm = TRUE),                        New\_df$Ozone) |

**Output:**



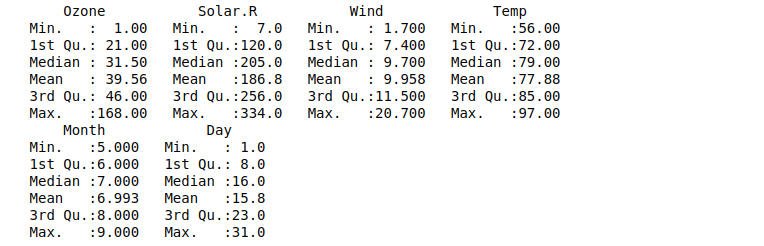
Performing the same operation in another column.

|  |
| --- |
| New\_df$Solar.R = ifelse(is.na(New\_df$Solar.R),                          median(New\_df$Solar.R,                                 na.rm = TRUE),                          New\_df$Solar.R) |

Now can clearly see that we don’t have any unclean data using summary methods.

|  |
| --- |
| summary(New\_df) |

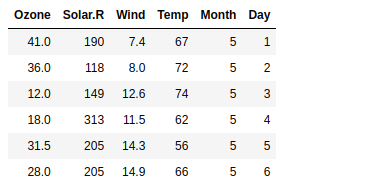
**Output:**



We can clearly see that we don’t have any missing data inside data frame.

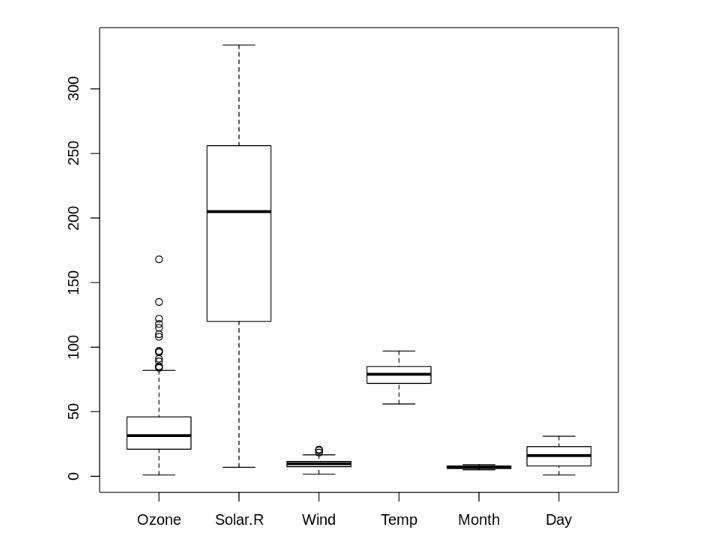
|  |
| --- |
| head(New\_df) |

**Output:**



Now our boxplot outliers also show no errors.

|  |
| --- |
| boxplot(New\_df) |



# Outlier Treatment

Outliers in data can distort predictions and affect the accuracy, if you don’t detect and handle them appropriately especially in regression models.

## **Why outliers detection is important?**

Treating or altering the outlier/extreme values in genuine observations is not a standard operating procedure. However, it is essential to understand their impact on your predictive models. It is left to the best judgement of the investigator to decide whether treating outliers is necessary and how to go about it.

So, why identifying the extreme values is important? Because, it can drastically bias/change the fit estimates and predictions. Let me illustrate this using the cars dataset.

To better understand the implications of outliers better, I am going to compare the fit of a simple linear regression model on cars dataset with and without outliers. In order to distinguish the effect clearly, I manually introduce extreme values to the original cars dataset. Then, I predict on both the datasets.

# Inject outliers into data.

cars1 <- cars[1:30, ] # original data

cars\_outliers <- data.frame(speed=c(19,19,20,20,20), dist=c(190, 186, 210, 220, 218)) # introduce outliers.

cars2 <- rbind(cars1, cars\_outliers) # data with outliers.

# Plot of data with outliers.

par(mfrow=c(1, 2))

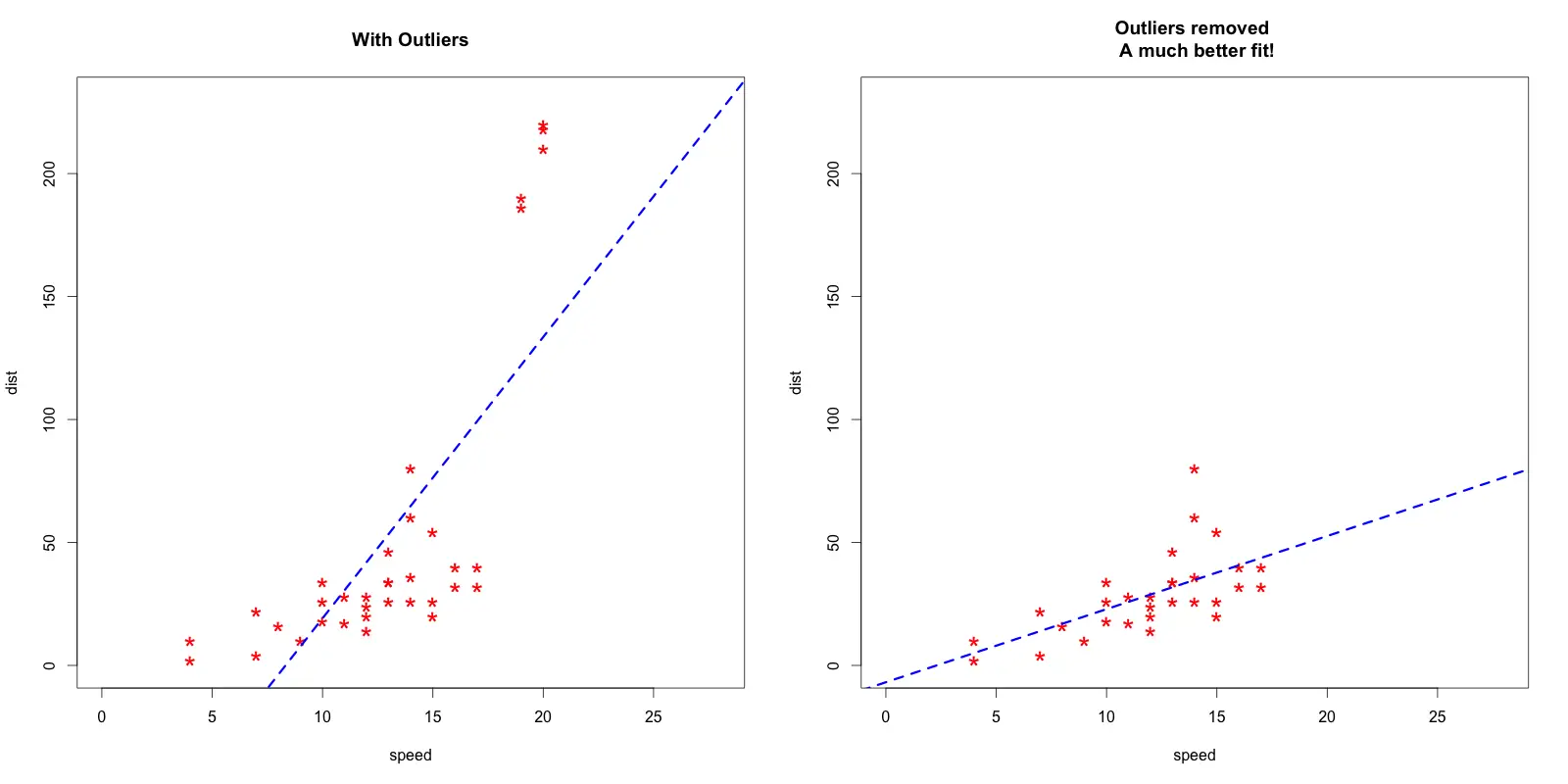
plot(cars2$speed, cars2$dist, xlim=c(0, 28), ylim=c(0, 230), main="With Outliers", xlab="speed", ylab="dist", pch="\*", col="red", cex=2)

abline(lm(dist ~ speed, data=cars2), col="blue", lwd=3, lty=2)

# Plot of original data without outliers. Note the change in slope (angle) of best fit line.

plot(cars1$speed, cars1$dist, xlim=c(0, 28), ylim=c(0, 230), main="Outliers removed \n A much better fit!", xlab="speed", ylab="dist", pch="\*", col="red", cex=2)

abline(lm(dist ~ speed, data=cars1), col="blue", lwd=3, lty=2)



Notice the change in slope of the best fit line after removing the outliers. Had we used the outliers to train the model(left chart), our predictions would be exagerated (high error) for larger values of speed because of the larger slope.

## **Detect outliers**

## **Univariate approach**

For a given continuous variable, outliers are those observations that lie outside 1.5 \* IQR, where IQR, the ‘Inter Quartile Range’ is the difference between 75th and 25th quartiles. Look at the points outside the whiskers in below box plot.

url <- "http://rstatistics.net/wp-content/uploads/2015/09/ozone.csv"

# alternate source: https://raw.githubusercontent.com/selva86/datasets/master/ozone.csv

inputData <- read.csv(url) # import data

outlier\_values <- boxplot.stats(inputData$pressure\_height)$out # outlier values.

boxplot(inputData$pressure\_height, main="Pressure Height", boxwex=0.1)

mtext(paste("Outliers: ", paste(outlier\_values, collapse=", ")), cex=0.6)

## Bivariate approach

Visualize in box-plot of the X and Y, for categorical X’s

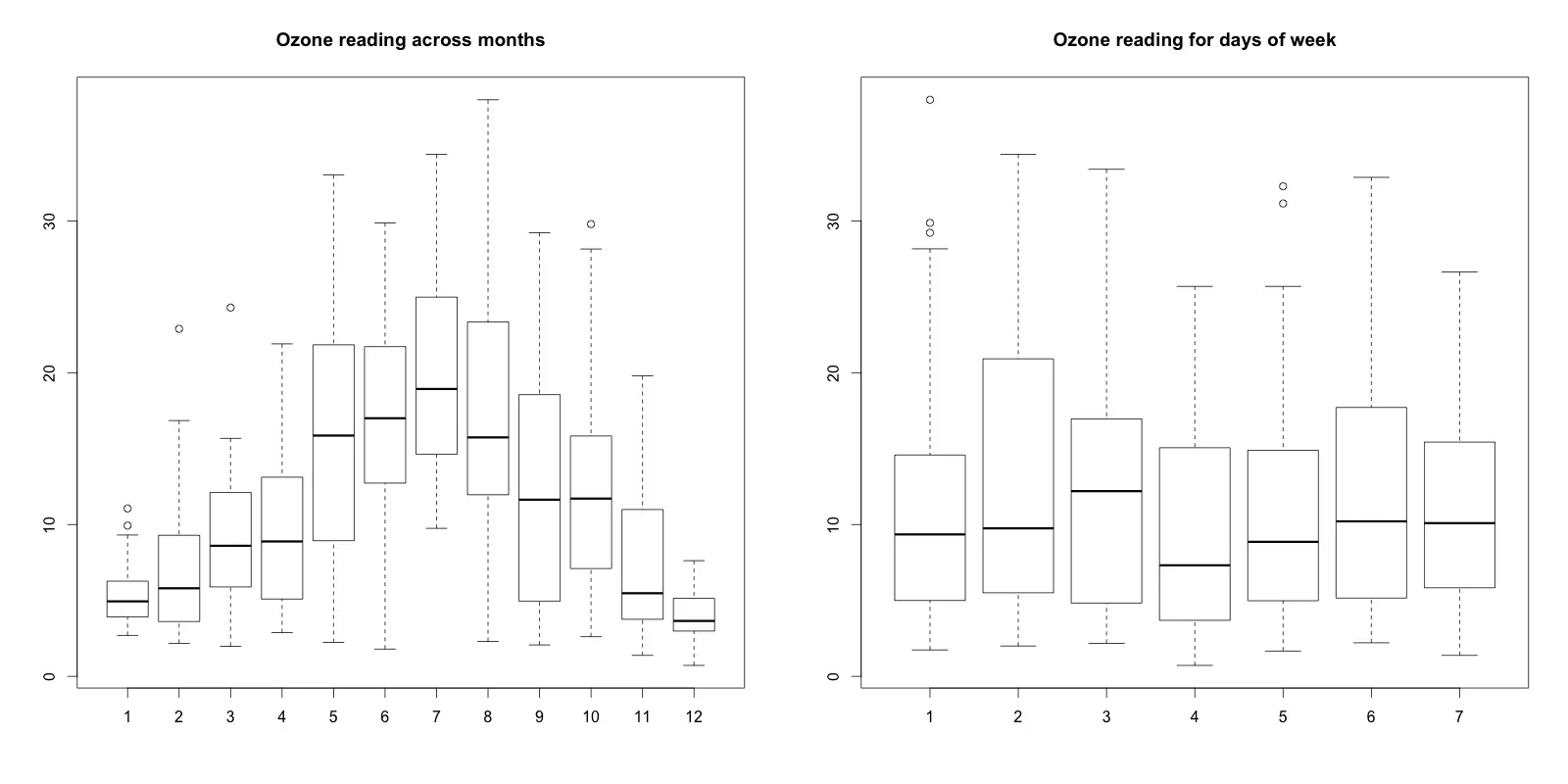
url <- "http://rstatistics.net/wp-content/uploads/2015/09/ozone.csv"

ozone <- read.csv(url)

# For categorical variable

boxplot(ozone\_reading ~ Month, data=ozone, main="Ozone reading across months") # clear pattern is noticeable.

boxplot(ozone\_reading ~ Day\_of\_week, data=ozone, main="Ozone reading for days of week") # this may not be significant, as day of week variable is a subset of the month var.

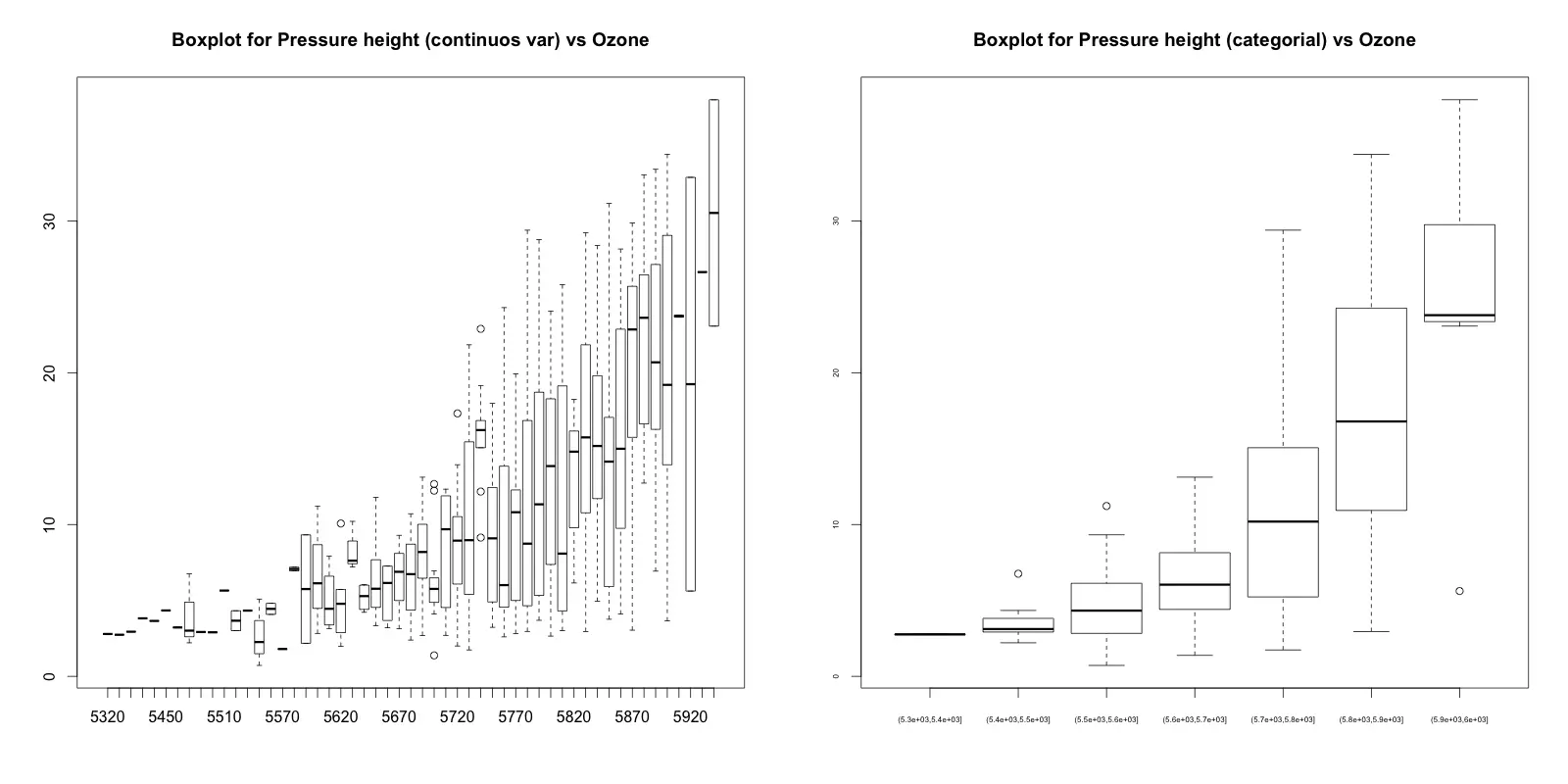


What is the inference? The change in the level of boxes suggests that Month seem to have an impact in ozone\_reading while Day\_of\_week does not. Any outliers in respective categorical level show up as dots outside the whiskers of the boxplot.

# For continuous variable (convert to categorical if needed.)

boxplot(ozone\_reading ~ pressure\_height, data=ozone, main="Boxplot for Pressure height (continuos var) vs Ozone")

boxplot(ozone\_reading ~ cut(pressure\_height, pretty(inputData$pressure\_height)), data=ozone, main="Boxplot for Pressure height (categorial) vs Ozone", cex.axis=0.5)



You can see few outliers in the box plot and how the ozone\_reading increases with pressure\_height. Thats clear.

## **Multivariate Model Approach**

Declaring an observation as an outlier based on a just one (rather unimportant) feature could lead to unrealistic inferences. When you have to decide if an individual entity (represented by row or observation) is an extreme value or not, it better to collectively consider the features (X’s) that matter. Enter Cook’s Distance.

#### **Cooks Distance**

Cook’s distance is a measure computed with respect to a given regression model and therefore is impacted only by the X variables included in the model. But, what does cook’s distance mean? It computes the influence exerted by each data point (row) on the predicted outcome.

The cook’s distance for each observation i measures the change in *Y*^

(fitted Y) for all observations with and without the presence of observation i, so we know how much the observation i impacted the fitted values. Mathematically, cook’s distance Di for observation i is computed as:

*Di*=∑*nj*=1(*Y*^*j*−*Y*^*j*(*i*))2*p*×*MSE*

where,

* *Y*^*j*

 is the value of jth fitted response when all the observations are included.

 *Y*^*j*(*i*)

* is the value of jth fitted response, where the fit does not include observation i.
* MSE is the mean squared error.
* p is the number of coefficients in the regression model.

mod <- lm(ozone\_reading ~ ., data=ozone)

cooksd <- cooks.distance(mod)

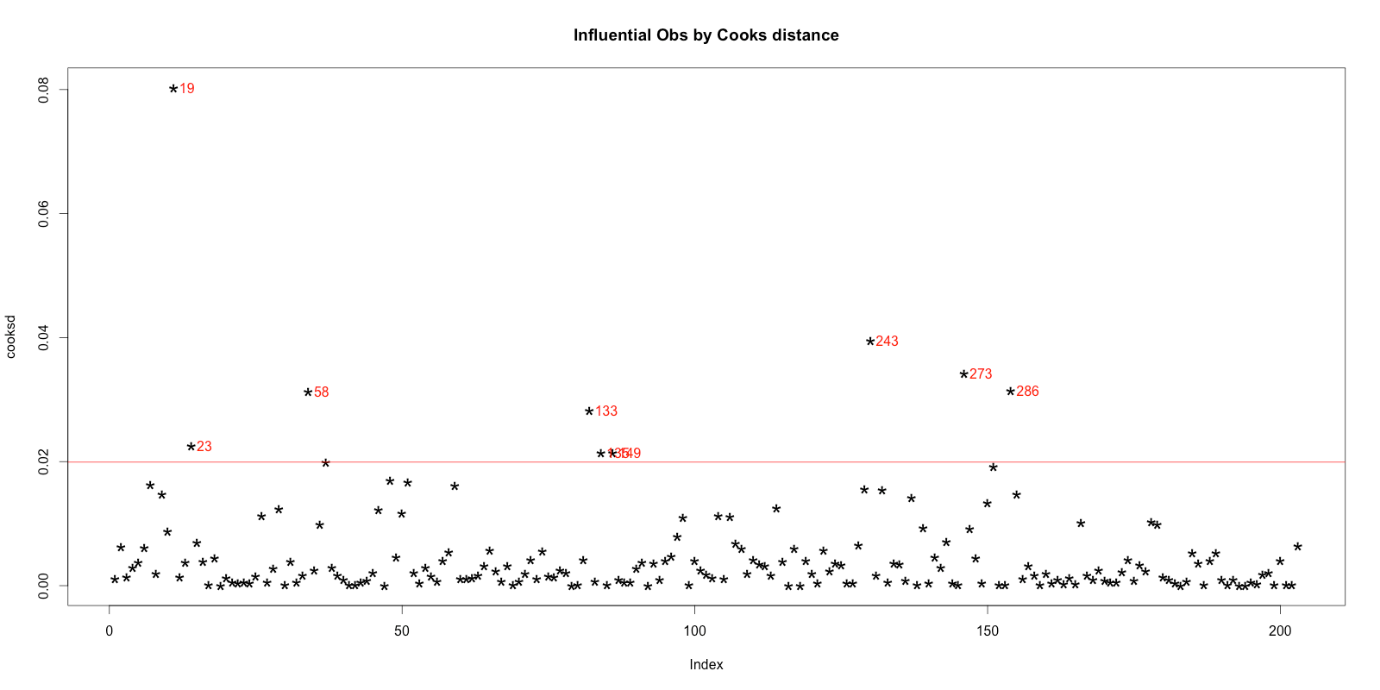
##### **Influence measures**

In general use, those observations that have a cook’s distance greater than 4 times the mean may be classified as influential. This is not a hard boundary.

plot(cooksd, pch="\*", cex=2, main="Influential Obs by Cooks distance") # plot cook's distance

abline(h = 4\*mean(cooksd, na.rm=T), col="red") # add cutoff line

text(x=1:length(cooksd)+1, y=cooksd, labels=ifelse(cooksd>4\*mean(cooksd, na.rm=T),names(cooksd),""), col="red") # add labels



Now lets find out the influential rows from the original data. If you extract and examine each influential row 1-by-1 (from below output), you will be able to reason out why that row turned out influential. It is likely that one of the X variables included in the model had extreme values.

influential <- as.numeric(names(cooksd)[(cooksd > 4\*mean(cooksd, na.rm=T))]) # influential row numbers

head(ozone[influential, ]) # influential observations.

#> Month Day\_of\_month Day\_of\_week ozone\_reading pressure\_height Wind\_speed Humidity

#> 19 1 19 1 4.07 5680 5 73

#> 23 1 23 5 4.90 5700 5 59

#> 58 2 27 5 22.89 5740 3 47

#> 133 5 12 3 33.04 5880 3 80

#> 135 5 14 5 31.15 5850 4 76

#> 149 5 28 5 4.82 5750 3 76

#> Temperature\_Sandburg Temperature\_ElMonte Inversion\_base\_height Pressure\_gradient

#> 19 52 56.48 393 -68

#> 23 69 51.08 3044 18

#> 58 53 58.82 885 -4

#> 133 80 73.04 436 0

#> 135 78 71.24 1181 50

#> 149 65 51.08 3644 86

#> Inversion\_temperature Visibility

#> 19 69.80 10

#> 23 52.88 150

#> 58 67.10 80

#> 133 86.36 40

#> 135 79.88 17

#> 149 59.36 70

Lets examine the first 6 rows from above output to find out why these rows could be tagged as influential observations.

* Row 58, 133, 135 have very high ozone\_reading.
* Rows 23, 135 and 149 have very high Inversion\_base\_height.
* Row 19 has very low Pressure\_gradient.

## **Outliers Test**

The function outlierTest from car package gives the most extreme observation based on the given model. Here’s an example based on the mod linear model object we’d just created.

car::outlierTest(mod)

#> No Studentized residuals with Bonferonni p < 0.05

#> Largest |rstudent|:

#> rstudent unadjusted p-value Bonferonni p

#> 243 3.045756 0.0026525 0.53845

This output suggests that observation in row 243 is most extreme.

## **outliers package**

The [outliers package](https://cran.r-project.org/web/packages/outliers/index.html) provides a number of useful functions to systematically extract outliers. Some of these are convenient and come handy, especially the outlier() and scores() functions.

##### outliers

outliers gets the extreme most observation from the mean. If you set the argument opposite=TRUE, it fetches from the other side.

set.seed(1234)

y=rnorm(100)

outlier(y)

#> [1] 2.548991

outlier(y,opposite=TRUE)

#> [1] -2.345698

dim(y) <- c(20,5) # convert it to a matrix

outlier(y)

#> [1] 2.415835 1.102298 1.647817 2.548991 2.121117

outlier(y,opposite=TRUE)

#> [1] -2.345698 -2.180040 -1.806031 -1.390701 -1.372302

##### scores

There are two aspects to the scores() function.

1. Compute the normalised scores based on “z”, “t”, “chisq” etc
2. Find out observations that lie beyond a given percentile based on a given score.

set.seed(1234)

x = rnorm(10)

scores(x) # z-scores => (x-mean)/sd

scores(x, type="chisq") # chi-sq scores => (x - mean(x))^2/var(x)

#> [1] 0.68458034 0.44007451 2.17210689 3.88421971 0.66539631 . . .

scores(x, type="t") # t scores

scores(x, type="chisq", prob=0.9) # beyond 90th %ile based on chi-sq

#> [1] FALSE FALSE FALSE TRUE FALSE FALSE FALSE FALSE FALSE FALSE

scores(x, type="chisq", prob=0.95) # beyond 95th %ile

scores(x, type="z", prob=0.95) # beyond 95th %ile based on z-scores

scores(x, type="t", prob=0.95) # beyond 95th %ile based on t-scores

## **Treating the outliers**

Once the outliers are identified and you have decided to make amends as per the nature of the problem, you may consider one of the following approaches.

#### **1. Imputation**

Imputation with mean / median / mode. This method has been dealt with in detail in the discussion about treating missing values.

#### **2. Capping**

For missing values that lie outside the 1.5 \* IQR limits, we could cap it by replacing those observations outside the lower limit with the value of 5th %ile and those that lie above the upper limit, with the value of 95th %ile. Below is a sample code that achieves this.

x <- ozone$pressure\_height

qnt <- quantile(x, probs=c(.25, .75), na.rm = T)

caps <- quantile(x, probs=c(.05, .95), na.rm = T)

H <- 1.5 \* IQR(x, na.rm = T)

x[x < (qnt[1] - H)] <- caps[1]

x[x > (qnt[2] + H)] <- caps[2]

#### **3. Prediction**

In yet another approach, the outliers can be replaced with missing values (NA) and then can be predicted by considering them as a response variable. We already discussed [how to predict missing values](http://r-statistics.co/Missing-Value-Treatment-With-R.html#4.%20Prediction).

## **What Is an Autoregressive Integrated Moving Average (ARIMA)?**

An autoregressive integrated moving average, or ARIMA, is a statistical analysis model that uses [time series data](https://www.investopedia.com/terms/t/timeseries.asp) to either better understand the data set or to predict future trends.

A statistical model is autoregressive if it predicts future values based on past values. For example, an ARIMA model might seek to predict a stock's future prices based on its past performance or forecast a company's earnings based on past periods.

### **Key Takeaways**

* Autoregressive integrated moving average (ARIMA) models predict future values based on past values.
* ARIMA makes use of lagged moving averages to smooth time series data.
* They are widely used in technical analysis to forecast future security prices.
* Autoregressive models implicitly assume that the future will resemble the past.
* Therefore, they can prove inaccurate under certain market conditions, such as financial crises or periods of rapid technological change.

## **Understanding Autoregressive Integrated Moving Average (ARIMA)**

An autoregressive integrated moving average model is a form of [regression analysis](https://www.investopedia.com/terms/r/regression.asp) that gauges the strength of one dependent variable relative to other changing variables. The model's goal is to predict future securities or financial market moves by examining the differences between values in the series instead of through actual values.

An ARIMA model can be understood by outlining each of its components as follows:

* [**Autoregression (AR)**](https://www.investopedia.com/terms/a/autoregressive.asp): refers to a model that shows a changing variable that regresses on its own lagged, or prior, values.
* **Integrated (I)**: represents the differencing of raw observations to allow the time series to become stationary (i.e., data values are replaced by the difference between the data values and the previous values).
* [**Moving average (MA)**](https://www.investopedia.com/terms/m/movingaverage.asp):  incorporates the dependency between an observation and a residual error from a moving average model applied to lagged observations.

## **ARIMA Parameters**

Each component in ARIMA functions as a parameter with a standard notation. For ARIMA models, a standard notation would be ARIMA with p, d, and q, where integer values substitute for the parameters to indicate the type of ARIMA model used. The parameters can be defined as:

* p: the number of lag observations in the model, also known as the lag order.
* d: the number of times the raw observations are differenced; also known as the degree of differencing.
* q: the size of the moving average window, also known as the order of the moving average.

## **Pros and Cons of ARIMA**

ARIMA models have strong points and are good at forecasting based on past circumstances, but there are more reasons to be cautious when using ARIMA. In stark contrast to investing disclaimers that state "past performance is not an indicator of future performance...," ARIMA models assume that past values have some residual effect on current or future values and use data from the past to forecast future events.

The following table lists other ARIMA traits that demonstrate good and bad characteristics.

Pros

* Good for short-term forecasting
* Only needs historical data
* Models non-stationary data

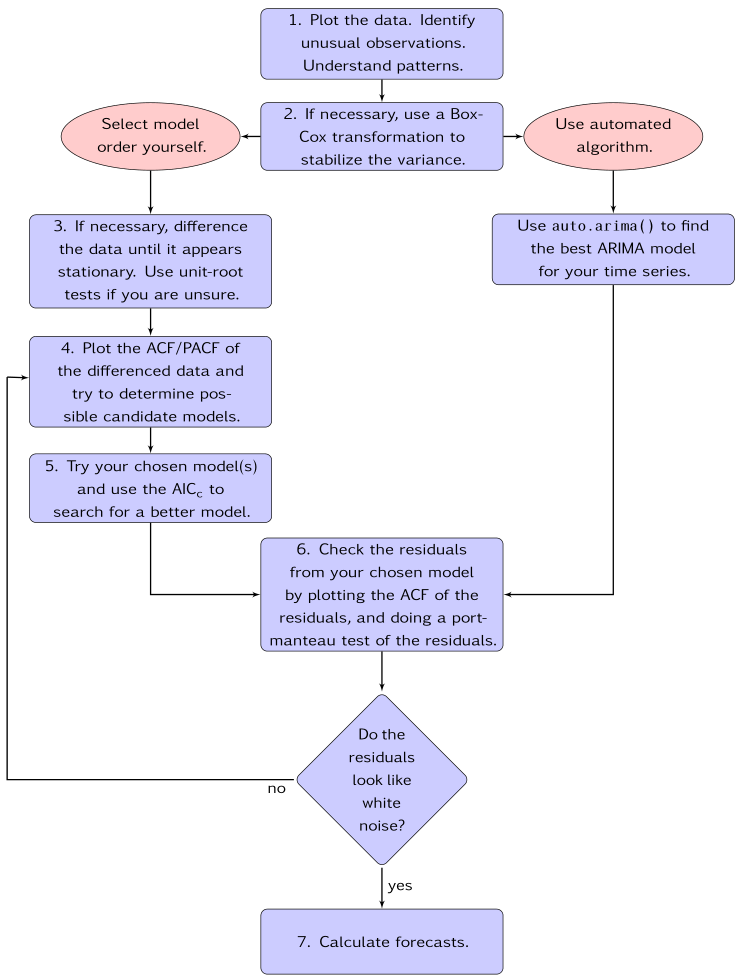
Cons

* Not built for long-term forecasting
* Poor at predicting turning points
* Computationally expensive
* Parameters are subjective

## **What Is ARIMA Used for?**

ARIMA is a method for forecasting or predicting future outcomes based on a historical time series. It is based on the statistical concept of serial correlation, where past data points influence future data points.

### **Modelling procedure**



### **Portmanteau tests of residuals for ARIMA models**

With ARIMA models, more accurate portmanteau tests are obtained if the degrees of freedom of the test statistic are adjusted to take account of the number of parameters in the model. Specifically, we use ℓ−K

degrees of freedom in the test, where K is the number of AR and MA parameters in the model. So for the non-seasonal models, we have considered so far, K=p+q. The correct value of K

is automatically determined in the checkresiduals() function.

### Example: Seasonally adjusted electrical equipment orders

We will apply this procedure to the seasonally adjusted electrical equipment orders data shown in Figure [8.12](https://otexts.com/fpp2/arima-r.html#fig:ee1).

elecequip %>% stl(s.window='periodic') %>% seasadj() -> eeadj

autoplot(eeadj)

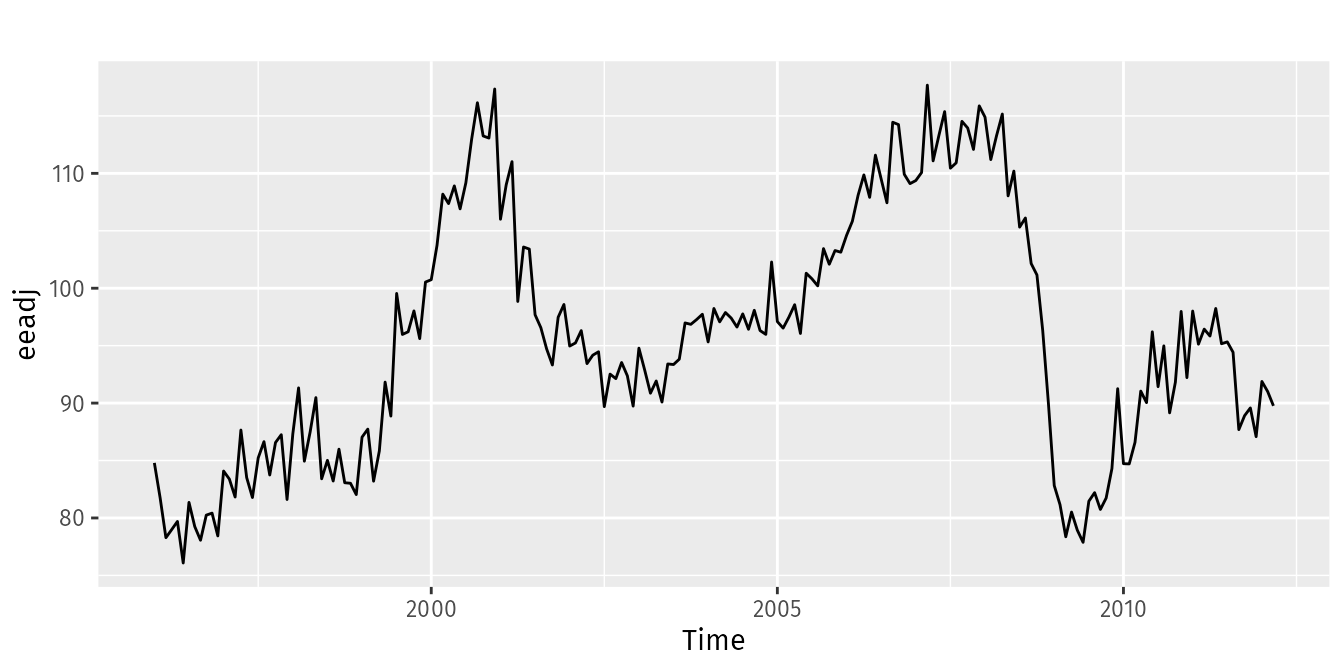


Figure 8.12: Seasonally adjusted electrical equipment orders index in the Euro area.

1. The time plot shows some sudden changes, particularly the big drop in 2008/2009. These changes are due to the global economic environment. Otherwise there is nothing unusual about the time plot and there appears to be no need to do any data adjustments.
2. There is no evidence of changing variance, so we will not do a Box-Cox transformation.
3. The data are clearly non-stationary, as the series wanders up and down for long periods. Consequently, we will take a first difference of the data. The differenced data are shown in Figure [8.13](https://otexts.com/fpp2/arima-r.html#fig:ee2). These look stationary, and so we will not consider further differences.

 eeadj %>% diff() %>% ggtsdisplay(main="")

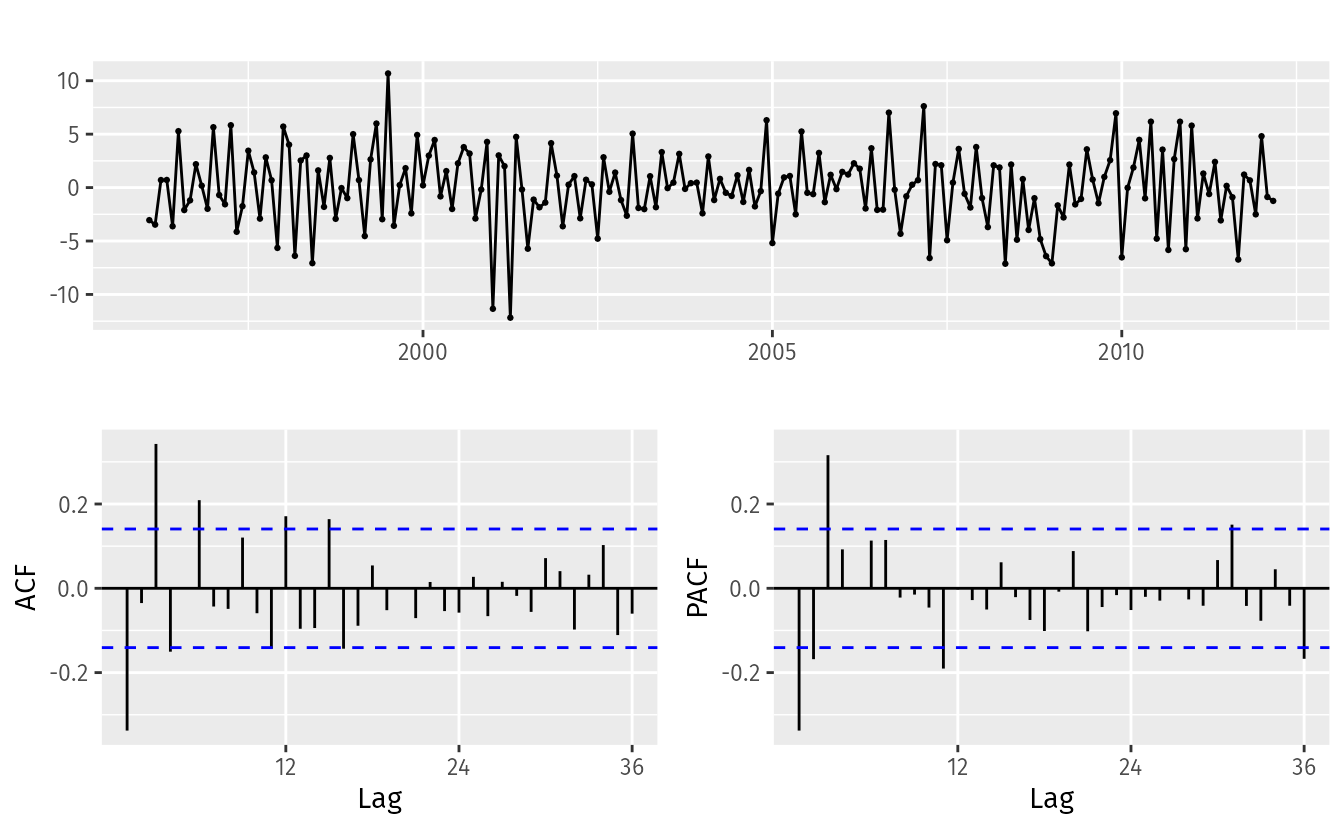


Figure 8.13: Time plot and ACF and PACF plots for the differenced seasonally adjusted electrical equipment data.

 The PACF shown in Figure [8.13](https://otexts.com/fpp2/arima-r.html#fig:ee2) is suggestive of an AR(3) model. So an initial candidate model is an ARIMA(3,1,0). There are no other obvious candidate models.

 We fit an ARIMA(3,1,0) model along with variations including ARIMA(4,1,0), ARIMA(2,1,0), ARIMA(3,1,1), etc. Of these, the ARIMA(3,1,1) has a slightly smaller AICc value.

 (fit <- Arima(eeadj, order=c(3,1,1)))

#> Series: eeadj

#> ARIMA(3,1,1)

#>

#> Coefficients:

#> ar1 ar2 ar3 ma1

#> 0.004 0.092 0.370 -0.392

#> s.e. 0.220 0.098 0.067 0.243

#>

#> sigma^2 = 9.58: log likelihood = -492.7

#> AIC=995.4 AICc=995.7 BIC=1012

 The ACF plot of the residuals from the ARIMA(3,1,1) model shows that all autocorrelations are within the threshold limits, indicating that the residuals are behaving like white noise. A portmanteau test returns a large p-value, also suggesting that the residuals are white noise.

 checkresiduals(fit)

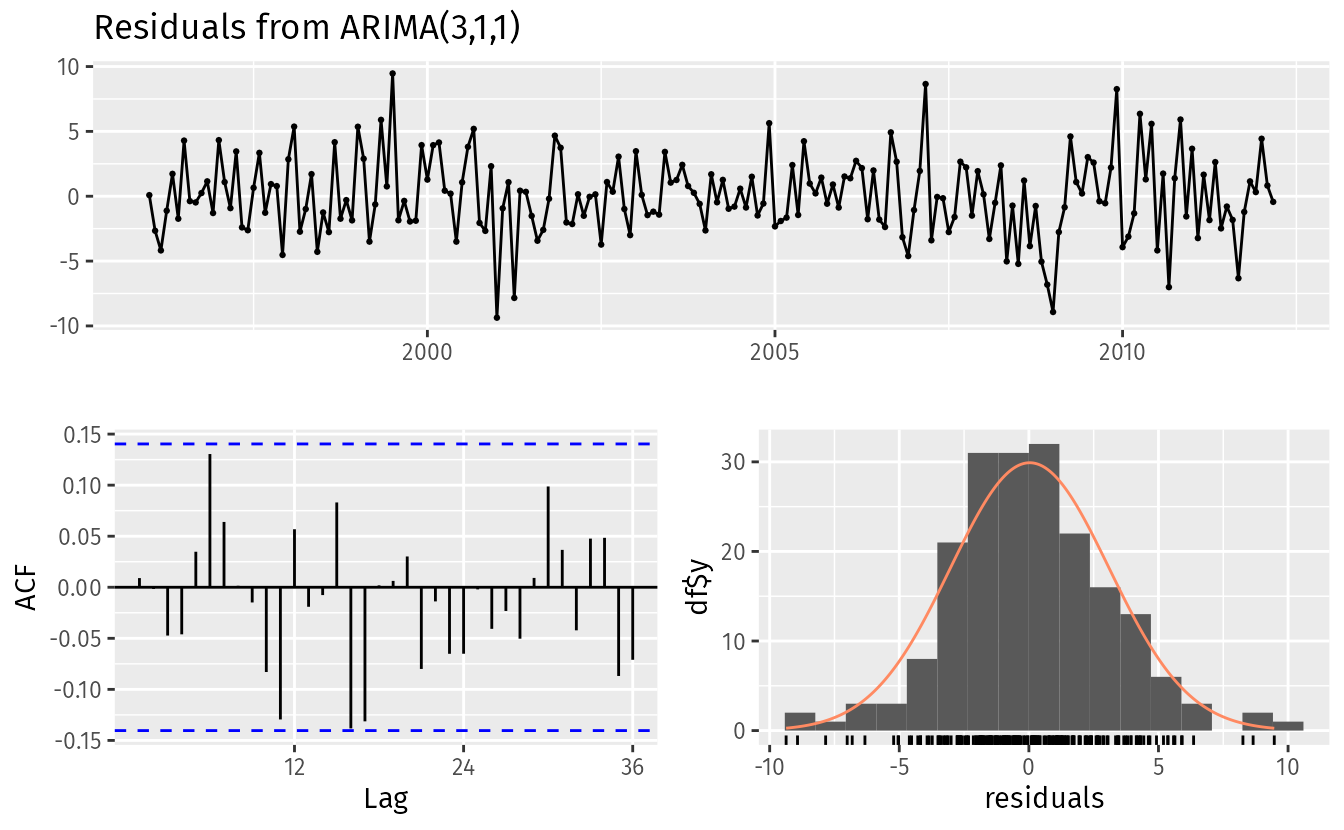


Figure 8.14: Residual plots for the ARIMA(3,1,1) model.

#>

#> Ljung-Box test

#>

#> data: Residuals from ARIMA(3,1,1)

#> Q\* = 24, df = 20, p-value = 0.2

#>

#> Model df: 4. Total lags used: 24

 Forecasts from the chosen model are shown in Figure [8.15](https://otexts.com/fpp2/arima-r.html#fig:ee4).

1. autoplot(forecast(fit))

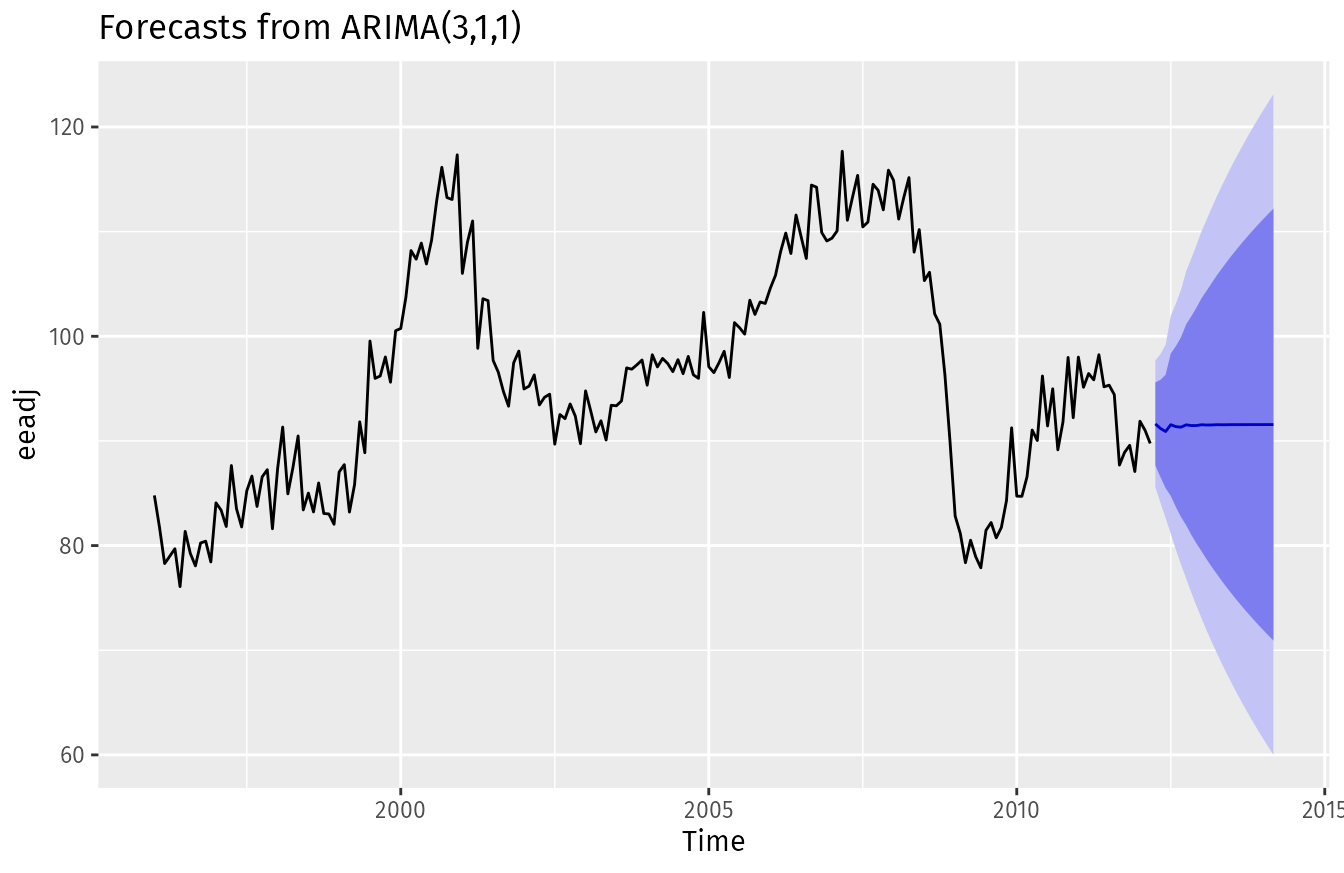


Figure 8.15: Forecasts for the seasonally adjusted electrical orders index.

If we had used the automated algorithm instead, we would have obtained an ARIMA(3,1,0) model using the default settings, but the ARIMA(3,1,1) model if we had set approximation=FALSE.

### How does auto.arima() work?

The auto.arima() function in R uses a variation of the Hyndman-Khandakar algorithm ([Hyndman & Khandakar, 2008](https://otexts.com/fpp2/arima-r.html#ref-HK08)), which combines unit root tests, minimisation of the AICc and MLE to obtain an ARIMA model. The arguments to auto.arima() provide for many variations on the algorithm.

# Forecasting Using ARIMA

Let’s first expand our dataset to include 365 days instead of 30 .

data = df[:365]['rainfall'].values

1. We then split the data into train (66%) and test set (34%).

train\_size = int(len(data) \* 0.66)  
train, test = data[0:train\_size], data[train\_size:len(data)]

2. And initialize the historical and prediction values for comparison purposes

history = [x for x in train]  
predictions = list()

3. Now we train the model and make future forecast as stored in the test data

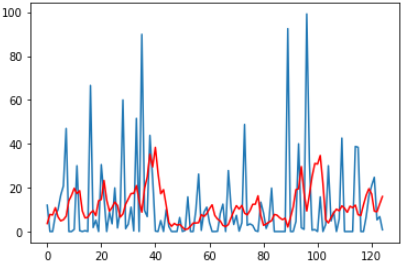
for t in range(len(test)): model = ARIMA(history, order=(5,1,0))  
 model\_fit = model.fit()  
 pred = model\_fit.forecast()  
 yhat = pred[0]  
 predictions.append(yhat) # Append test observation into overall record  
 obs = test[t]  
 history.append(obs)

4. Lets evaluate our performance

from sklearn.metrics import mean\_squared\_error  
from math import sqrtrmse = sqrt(mean\_squared\_error(test, predictions))print('Test RMSE: %.3f' % rmse)  
>>> Test RMSE: 20.664

Pretty huge RMSE! Definitely there’s room for improvement here.

We can also plot the difference between observations and predictions, and compare how both are similar (or diverging!)



Evaluation (Image by Author)

Not too bad! Our prediction (red) closely resembles the observations (blue) for some days (except in extreme rainfall cases with noticeably high peaks).