Anomaly Detection

Quincy

9/9/2021

Anomaly Detection

Context

We are to check whether there are any anomalies in the given sales dataset. The objective of this task being fraud detection.

Load data

```
# Installing anomalize package
#install.packages("anomalize",repos = "http://cran.us.r-project.org")
```

```
# Load tidyverse and anomalize
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.1 --
```

```
## v ggplot2 3.3.5     v purrr   0.3.4
## v tibble  3.1.4     v dplyr   1.0.7
## v tidyr   1.1.3     v stringr 1.4.0
## v readr   2.0.1     v forcats 0.5.1
```

```
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
library(anomalize)
```

```
## == Use anomalize to improve your Forecasts by 50%! =============================
## Business Science offers a 1-hour course - Lab #18: Time Series Anomaly Detection!
## </> Learn more at: https://university.business-science.io/p/learning-labs-pro </>
```

```
# read data
forecast <- read.csv('http://bit.ly/CarreFourSalesDataset')
View(forecast)
```

```r
# checking the structure of our data
str(forecast)
```

```
## 'data.frame':    1000 obs. of  2 variables:
##  $ Date : chr  "1/5/2019" "3/8/2019" "3/3/2019" "1/27/2019" ...
##  $ Sales: num  549 80.2 340.5 489 634.4 ...
```

```r
# checking the shape
dim(forecast)
```

```
## [1] 1000    2
```

We have 1000 observations and 2 variables.

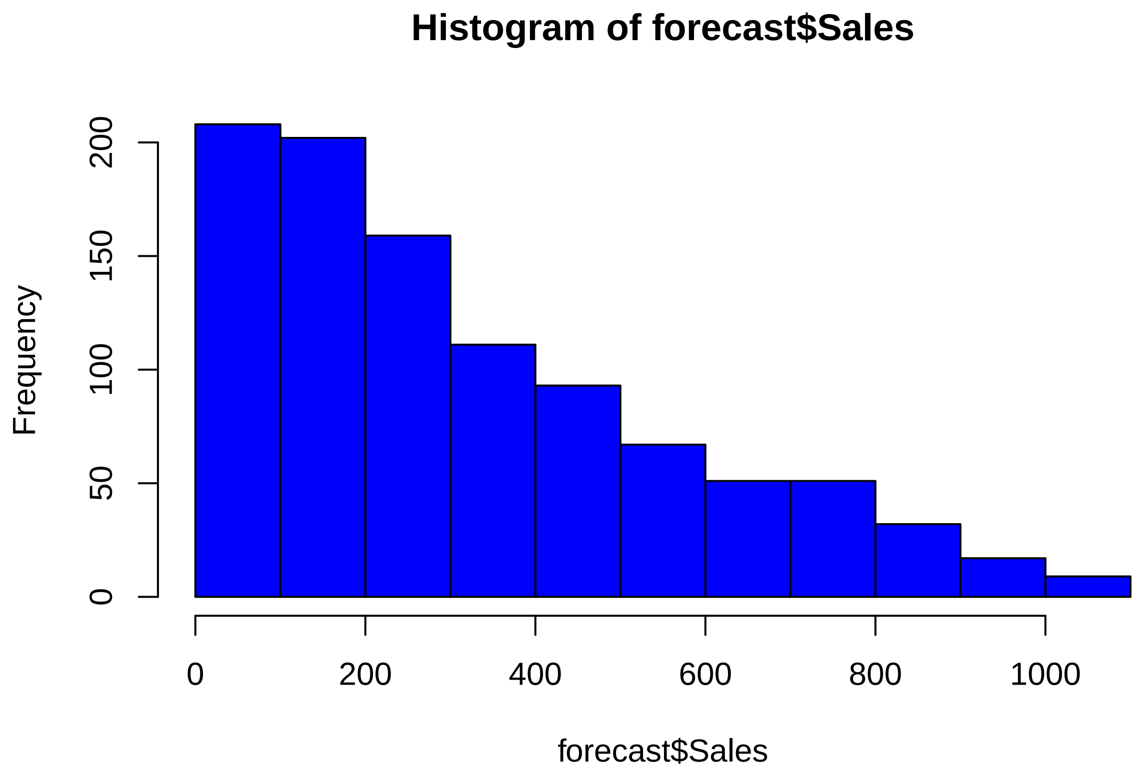
```r
# converting variables to our preferred format
forecast$Date <- as.Date(forecast$Date, "%m/%d/%Y")
```

```r
str(forecast)
```

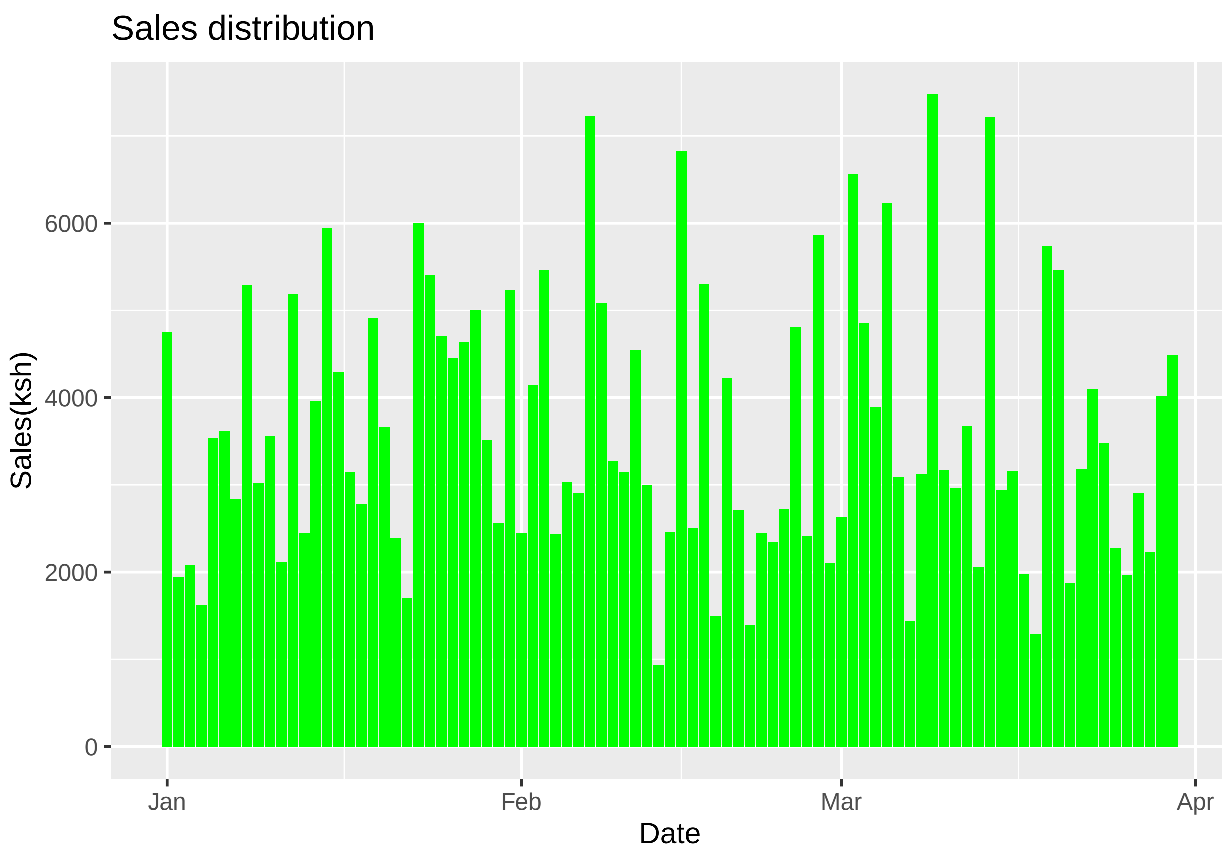
```
## 'data.frame':    1000 obs. of  2 variables:
##  $ Date : Date, format: "2019-01-05" "2019-03-08" ...
##  $ Sales: num  549 80.2 340.5 489 634.4 ...
```

Visualization

```r
# visualizing our sales
hist(forecast$Sales,col="blue")
```




```
# Sales distribution over time
library(ggplot2)
ggplot(data = forecast, aes(x = Date, y = Sales)) +
      geom_bar(stat = "identity", fill = "green") +
      labs(title = "Sales distribution",
           x = "Date", y = "Sales(ksh)")
```




```r
# Load libraries
library(tibbletime)
```

```
##
## Attaching package: 'tibbletime'

## The following object is masked from 'package:stats':
##
##     filter
```

```r
# Ordering the data by Date
forecast = forecast %>% arrange(Date)
head(forecast)
```

```
##         Date   Sales
## 1 2019-01-01 457.443
## 2 2019-01-01 399.756
## 3 2019-01-01 470.673
## 4 2019-01-01 388.290
## 5 2019-01-01 132.762
## 6 2019-01-01 132.027
```

```r
# Since our data has many records per day,
# We get the average per day, so that the data
forecast = aggregate(Sales ~ Date , forecast , mean)
head(forecast)
```

```
##         Date    Sales
## 1 2019-01-01 395.4318
## 2 2019-01-02 243.1879
## 3 2019-01-03 259.7661
## 4 2019-01-04 270.6148
## 5 2019-01-05 294.7236
## 6 2019-01-06 401.5783
```

```
# Converting data frame to a tibble time (tbl_time)
# tbl_time have a time index that contains information about which column
# should be used for time-based subsetting and other time-based manipulation,
forecast= tbl_time(forecast, Date)
class(forecast)
```

```
## [1] "tbl_time"   "tbl_df"     "tbl"        "data.frame"
```

We now use the following functions to detect and visualize anomalies;

The default values for time series decompose are method = "stl", which is just seasonal decomposition using a Loess smoother (refer to stats::stl()).

The frequency and trend parameters are automatically set based on the time scale (or periodicity)of the time series using tibbletime based function under the hood.

time_decompose() - this function would help with time series decomposition.

anomalize() - We perform anomaly detection on the decomposed data using the remainder column through the use of the anomalize() function which procides 3 new columns; remainder_l1" (lower limit), "remainder_l2" (upper limit), and "anomaly" (Yes/No Flag).

The default method is method = "iqr", which is fast and relatively accurate at detecting anomalies.

The alpha parameter is by default set to alpha = 0.05, but can be adjusted to increase or decrease the height of the anomaly bands, making it more difficult or less difficult for data to be anomalous.

The max_anoms parameter is by default set to a maximum of max_anoms = 0.2 for 20% of data that can be anomalous.

time_recompose()- We create the lower and upper bounds around the observed values through the use of the time_recompose() function, which recomposes the lower and upper bounds of the anomalies around the observed values.

We create new columns created: recomposed_l1(lower limit) and recomposed_l2 (upper limit).

plot_anomalies() - we now plot using plot_anomaly_decomposition() to visualize out data.

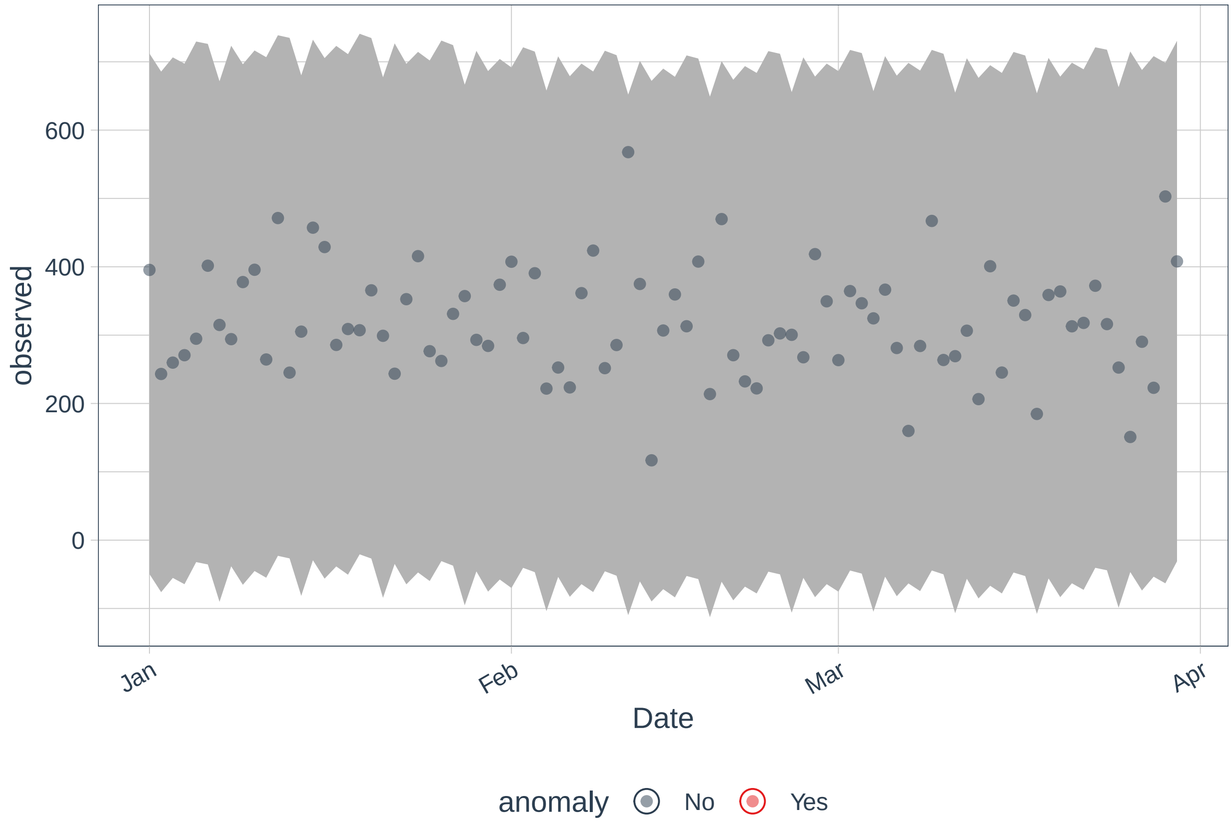
```
forecast %>%
    time_decompose(Sales) %>%
    anomalize(remainder) %>%
    time_recompose() %>%
    plot_anomalies(time_recomposed = TRUE, ncol = 3, alpha_dots = 0.5)
```

```
## frequency = 7 days
```

```
## trend = 30 days
```

```
## Registered S3 method overwritten by 'quantmod':
##   method            from
##   as.zoo.data.frame zoo
```

```
## Warning: ‘type_convert()‘ only converts columns of type ’character’.
## - ‘df‘ has no columns of type ’character’
```



Our data has no anomalies.