IE Project

yunus emre erdogan , ugur uzunoglu

02 07 2021

```
##
## Attaching package: 'zoo'

## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric

## Loading required package: ggplot2

##
## Attaching package: 'plotly'

## The following object is masked from 'package:ggplot2':
##
##     last_plot

## The following object is masked from 'package:httr':
##
##     config

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

## The following object is masked from 'package:graphics':
##
##     layout

##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union

## NOTE: Either Arial Narrow or Roboto Condensed fonts are required to use these themes.
```

```
##       Please use hrbrthemes::import_roboto_condensed() to install Roboto Condensed and

##       if Arial Narrow is not on your system, please see https://bit.ly/arialnarrow

##
## Attaching package: 'lubridate'

## The following objects are masked from 'package:base':
##
##     date, intersect, setdiff, union

##
## Attaching package: 'data.table'

## The following objects are masked from 'package:lubridate':
##
##     hour, isoweek, mday, minute, month, quarter, second, wday, week,
##     yday, year

## The following objects are masked from 'package:dplyr':
##
##     between, first, last

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

## Loading required package: fma

## Loading required package: expsmooth

## Loading required package: lmtest

## Loading required package: tseries

##
## Attaching package: 'xts'

## The following objects are masked from 'package:data.table':
##
##     first, last

## The following objects are masked from 'package:dplyr':
##
##     first, last

## -- Attaching packages --------------------------------------- tidyverse 1.3.0 --

## v tibble  3.1.0     v stringr 1.4.0
## v readr   1.4.0     v forcats 0.5.1
## v purrr   0.3.4
```

```
## -- Conflicts ---------------------------------------- tidyverse_conflicts() --
## x lubridate::as.difftime() masks base::as.difftime()
## x data.table::between()    masks dplyr::between()
## x plotly::config()         masks httr::config()
## x lubridate::date()        masks base::date()
## x dplyr::filter()          masks plotly::filter(), stats::filter()
## x xts::first()             masks data.table::first(), dplyr::first()
## x purrr::flatten()         masks jsonlite::flatten()
## x data.table::hour()       masks lubridate::hour()
## x lubridate::intersect()   masks base::intersect()
## x data.table::isoweek()    masks lubridate::isoweek()
## x dplyr::lag()             masks stats::lag()
## x xts::last()              masks data.table::last(), dplyr::last()
## x data.table::mday()       masks lubridate::mday()
## x data.table::minute()     masks lubridate::minute()
## x data.table::month()      masks lubridate::month()
## x data.table::quarter()    masks lubridate::quarter()
## x data.table::second()     masks lubridate::second()
## x lubridate::setdiff()     masks base::setdiff()
## x purrr::transpose()       masks data.table::transpose()
## x lubridate::union()       masks base::union()
## x data.table::wday()       masks lubridate::wday()
## x data.table::week()       masks lubridate::week()
## x data.table::yday()       masks lubridate::yday()
## x data.table::year()       masks lubridate::year()

## Loading required package: lattice

##
## Attaching package: 'caret'

## The following object is masked from 'package:purrr':
##
##     lift

## The following object is masked from 'package:httr':
##
##     progress

##
## Attaching package: 'gridExtra'

## The following object is masked from 'package:dplyr':
##
##     combine

## Registered S3 method overwritten by 'GGally':
##   method from
##   +.gg   ggplot2

##
## Attaching package: 'GGally'
```

```
## The following object is masked from 'package:fma':
##
##     pigs

## ------------------------------------------------------------------------------

## You have loaded plyr after dplyr - this is likely to cause problems.
## If you need functions from both plyr and dplyr, please load plyr first, then dplyr:
## library(plyr); library(dplyr)

## ------------------------------------------------------------------------------

##
## Attaching package: 'plyr'

## The following object is masked from 'package:purrr':
##
##     compact

## The following object is masked from 'package:fma':
##
##     ozone

## The following objects are masked from 'package:dplyr':
##
##     arrange, count, desc, failwith, id, mutate, rename, summarise,
##     summarize

## The following objects are masked from 'package:plotly':
##
##     arrange, mutate, rename, summarise
```

Introduction

One holy question rises from the Misty Mountains : How many products will Trendyol sell each day? Trendyol is a online retail e-commerce company. One of the main challanges of their daily tasks is to forecast next day's sales quantity of their products and to take an action accordingly.In this project, we will try to develop forecasting methods to predict the next day's sales quantity of the products in Table 1.

```
Product_id = c("48740784","73318567","32737302","31515569","6676673","7061886","85004","4066298","32939

top_hier = c("Dis Giyim"," Plaj Giyim"," Plaj Giyim","Alt Giyim"," Elektronik Aksesuarlar","Elektrikli

bottom_hier = c("Mont","Bikini Ustu","Bikini Ustu","Tayt","Bluetooth Kulaklık"," Dik Supurge","Yuz Temiz

Brand = c("ALTINYILDIZ CLASSICS"," TRENDYOLMILLA"," TRENDYOLMILLA"," TRENDYOLMILLA","Xiaomi","Fakir","La

table_1 = cbind(Product_id,top_hier,bottom_hier,Brand)

print(table_1)
```

```
##       Product_id top_hier                  bottom_hier
##  [1,] "48740784" "Dis Giyim"               "Mont"
##  [2,] "73318567" " Plaj Giyim"             "Bikini Ustu"
##  [3,] "32737302" " Plaj Giyim"             "Bikini Ustu"
##  [4,] "31515569" "Alt Giyim"               "Tayt"
##  [5,] "6676673"  " Elektronik Aksesuarlar" "Bluetooth Kulaklık"
##  [6,] "7061886"  "Elektrikli Ev Aletleri"  " Dik Supurge"
##  [7,] "85004"    " Cilt Bakım"             "Yuz Temizleyici"
##  [8,] "4066298"  "Anne ve Bebek Bakım"     " Bebek Islak Mendil"
##  [9,] "32939029" "Agiz Bakım"              " Sarj Edebilir Dis Fırcası"
##       Brand
##  [1,] "ALTINYILDIZ CLASSICS"
##  [2,] " TRENDYOLMILLA"
##  [3,] " TRENDYOLMILLA"
##  [4,] " TRENDYOLMILLA"
##  [5,] "Xiaomi"
##  [6,] "Fakir"
##  [7,] "La Roche Posay"
##  [8,] "Sleepy"
##  [9,] "Oral-B"
```

The data taken from Trendyol contains daily sales of nine products and includes product, category level, brand and site level details. These additional details may help us to define useful regressors and to obtain better models. Now we should start with the investigation of data and some preprocessing issues.

Data Investigation and Preprocessing

Data is taken from an application specific API and every day new daily sales quantity and other details are added to the our core csv data. To get the data and for further submission purposes we need to add necessary functions to handle these issues :

```
get_token <- function(username, password, url_site){

  post_body = list(username=username,password=password)
  post_url_string = paste0(url_site,'/token/')
  result = POST(post_url_string, body = post_body)

  # error handling (wrong credentials)
  if(result$status_code==400){
    print('Check your credentials')
    return(0)
  }
  else if (result$status_code==201){
    output = content(result)
    token = output$key
  }

  return(token)
}

get_data <- function(start_date='2020-03-20', token, url_site){

  post_body = list(start_date=start_date,username=username,password=password)
```

```
  post_url_string = paste0(url_site,'/dataset/')

  header = add_headers(c(Authorization=paste('Token',token,sep=' ')))
  result = GET(post_url_string, header, body = post_body)
  output = content(result)
  data = data.table::rbindlist(output)
  data[,event_date:=as.Date(event_date)]
  data = data[order(product_content_id,event_date)]
  return(data)
}


send_submission <- function(predictions, token, url_site, submit_now=F){

  format_check=check_format(predictions)
  if(!format_check){
    return(FALSE)
  }

  post_string="list("
  for(i in 1:nrow(predictions)){
    post_string=sprintf("%s'%s'=%s",post_string,predictions$product_content_id[i],predictions$forecast[i
    if(i<nrow(predictions)){
      post_string=sprintf("%s,",post_string)
    } else {
      post_string=sprintf("%s)",post_string)
    }
  }

  submission = eval(parse(text=post_string))
  json_body = jsonlite::toJSON(submission, auto_unbox = TRUE)
  submission=list(submission=json_body)

  print(submission)
  # {"31515569":2.4,"32737302":2.4,"32939029":2.4,"4066298":2.4,"48740784":2.4,"6676673":2.4, "7061886"

  if(!submit_now){
    print("You did not submit.")
    return(FALSE)
  }


  header = add_headers(c(Authorization=paste('Token',token,sep=' ')))
  post_url_string = paste0(url_site,'/submission/')
  result = POST(post_url_string, header, body=submission)

  if (result$status_code==201){
    print("Successfully submitted. Below you can see the details of your submission")
  } else {
    print("Could not submit. Please check the error message below, contact the assistant if needed.")
  }

  print(content(result))
```

```
}


check_format <- function(predictions){

  if(is.data.frame(predictions) | is.data.frame(predictions)){
    if(all(c('product_content_id','forecast') %in% names(predictions))){
      if(is.numeric(predictions$forecast)){
        print("Format OK")
        return(TRUE)
      } else {
        print("forecast information is not numeric")
        return(FALSE)
      }
    } else {
      print("Wrong column names. Please provide 'product_content_id' and 'forecast' columns")
      return(FALSE)
    }

  } else {
    print("Wrong format. Please provide data.frame or data.table object")
    return(FALSE)
  }

}

# this part is main code
subm_url = 'http://46.101.163.177'

u_name = "Group2"
p_word = "T2MypIq2XP4MjH75"
submit_now = FALSE

username = u_name
password = p_word

token = get_token(username=u_name, password=p_word, url=subm_url)
```

After creating user specific token , we can obtain data and submit it during post-processing time.Here we took the core csv data and added new daily data on top of that. The 31th May was the last current day in the core data and we adjusted the event_date accordingly.The mixed columns of the new data table are also arranged. After all that we can combine rows of both sources together :

```
data = get_data(token=token,url=subm_url)

df2<-data[!(data$event_date <= "2021-05-31"),]

df2 <-df2 %>% arrange(desc(ymd(df2$event_date)))

df2=df2[,c(2,3,1,4,5,7,6,8,10,12,13,9,11)]

data_path='C:/Users/yeerd/OneDrive/Masaüstü/IE_project/ProjectRawData.csv'
trendyol=fread(data_path)
```

```
trendyol$event_date=as.Date(trendyol$event_date)
trendyol=rbind(df2,trendyol)
str(trendyol)
```

```
## Classes 'data.table' and 'data.frame':   4610 obs. of  13 variables:
##  $ event_date         : Date, format: "2021-07-01" "2021-07-01" ...
##  $ product_content_id : chr  "31515569" "32737302" "32939029" "4066298" ...
##  $ price              : num  44.9 61.6 141.2 70 700 ...
##  $ sold_count         : int  483 34 130 501 1 312 15 14 54 298 ...
##  $ visit_count        : int  10787 4125 3940 4938 57 11562 1074 4707 2801 9911 ...
##  $ basket_count       : int  2046 199 488 1265 4 930 51 87 403 1359 ...
##  $ favored_count      : int  649 293 875 303 2 777 103 255 415 867 ...
##  $ category_sold      : int  7463 6040 886 3949 1055 4839 886 6040 4320 7514 ...
##  $ category_visits    : int  418946 837980 32966 51553 199537 256832 64930 837980 171173 429371 ...
##  $ category_basket    : int  38687 29525 3093 11849 7083 21667 3324 29525 25721 39443 ...
##  $ category_favored   : int  37332 65643 4889 4284 13343 19158 5648 65643 26199 36388 ...
##  $ category_brand_sold: int  1510 3952 827 2362 3 752 184 3952 528 1115 ...
##  $ ty_visits          : int  106491398 106491398 106491398 106491398 106491398 106491398 106491398 1
##  - attr(*, ".internal.selfref")=<externalptr>
```

We see above 10 probable regressor candidates in the given data and we will play with them to get better results in a minute. We have some NA values, inconsistent data rows and some illogical(to the common sense) data points. To clear them up , we preprocessed it :

Undesired data is handled

```
trendyol = trendyol[!is.na(trendyol$event_date)]
trendyol = trendyol[!(ty_visits == 1)]


trendyol <- trendyol %>% mutate(soldout =
                                  case_when(price == -1 | is.na(price)==T~ 1,
                                            price > 0 | is.na(price)==F ~ 0

                                  )
)

trendyol$price[trendyol$price == -1 ] <- 0
trendyol$price[is.na(trendyol$price)] <- 0
```

New regressor detected ! : soldout We also added a new feature into the data: soldout. This soldout feature corresponds to a product's daily sales quantity being zero. By doing that we want to get the notion of no sales or out-of-product sense.We will add it again into the model but here we just introduced it.

The weekday and month of the daily data will also be quite useful , hence we decided to add 2 more columns containing the weekday names and month names.

Weekday and Month addition

```
trendyol[,wday:=weekdays(trendyol$event_date)]
trendyol[,mon:=months(trendyol$event_date)]
```

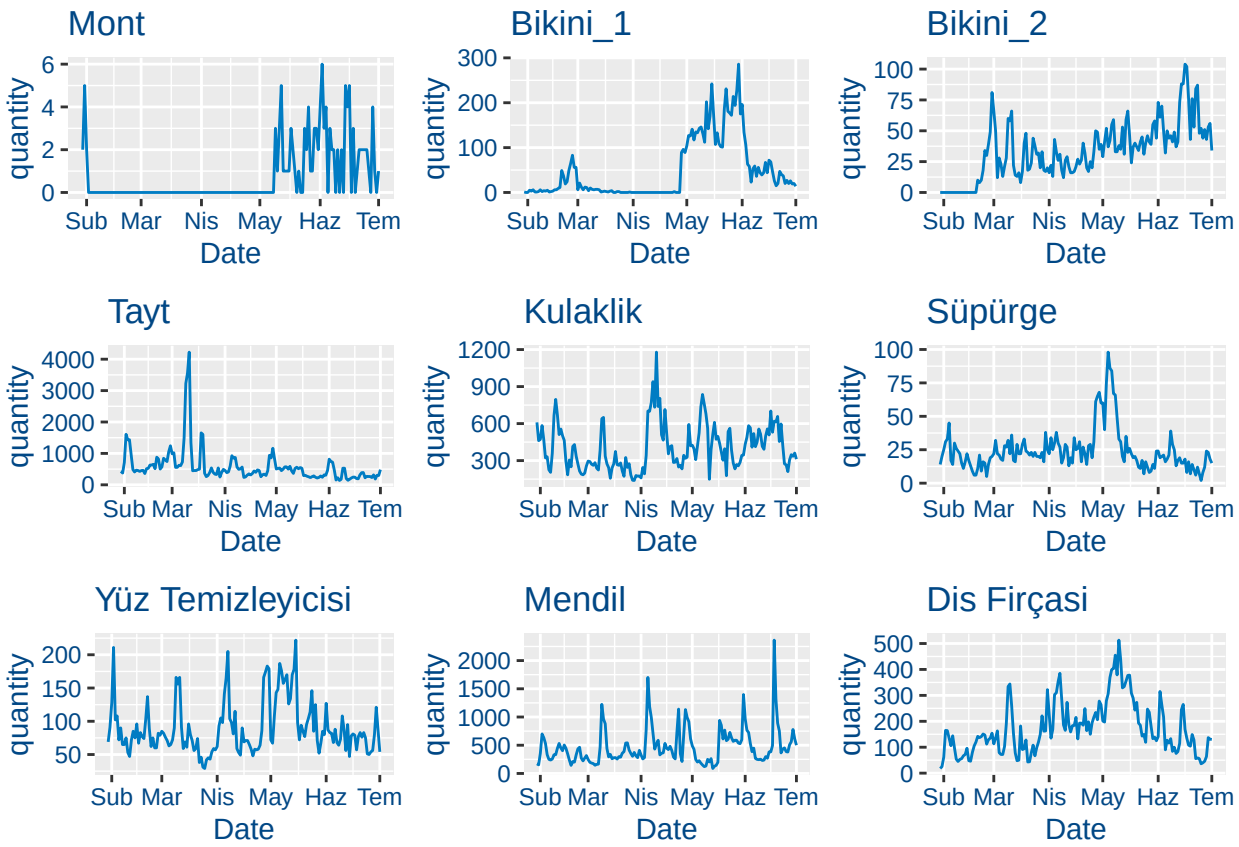
After preprocessing we shall continue with dividing the data by 9 according to product content id , since we will apply the model separately on them:

```
mont = trendyol[product_content_id=="48740784"]
bikini_1 = trendyol[product_content_id=="73318567"]
bikini_2 = trendyol[product_content_id=="32737302"]
tayt = trendyol[product_content_id=="31515569"]
kulaklık = trendyol[product_content_id=="6676673"]
supurge = trendyol[product_content_id=="7061886"]
yuz_temizleyicisi = trendyol[product_content_id=="85004"]
mendil = trendyol[product_content_id=="4066298"]
dis_fircasi = trendyol[product_content_id=="32939029"]
```

Now we can plot each product's sales quantity and get some visualisations on them :

9 products' sales quantity graphs

```
par(mfrow=c(3,3))
a<-ggplot(data = mont, aes(x = event_date, y = sold_count ,group=1))+
  geom_line(color = '#007cc3') + labs(title = 'Mont', x= "Date", y = "quantity") + theme(text=element_te

b<-ggplot(data = bikini_1, aes(x = event_date, y = sold_count ,group=1))+
  geom_line(color = '#007cc3') + labs(title = 'Bikini_1', x= "Date", y = "quantity") + theme(text=elemen

c<-ggplot(data = bikini_2, aes(x = event_date, y = sold_count ,group=1))+
  geom_line(color = '#007cc3') + labs(title = 'Bikini_2', x= "Date", y = "quantity") + theme(text=elemen

d<-ggplot(data = tayt, aes(x = event_date, y = sold_count ,group=1))+
  geom_line(color = '#007cc3') + labs(title = 'Tayt', x= "Date", y = "quantity") + theme(text=element_te

e<-ggplot(data = kulaklık, aes(x = event_date, y = sold_count ,group=1))+
  geom_line(color = '#007cc3') + labs(title = 'Kulaklık', x= "Date", y = "quantity") + theme(text=elemen

f<-ggplot(data = supurge, aes(x = event_date, y = sold_count ,group=1))+
  geom_line(color = '#007cc3') + labs(title = 'Süpürge', x= "Date", y = "quantity") + theme(text=element

g<-ggplot(data = yuz_temizleyicisi, aes(x = event_date, y = sold_count ,group=1))+
  geom_line(color = '#007cc3') + labs(title = 'Yüz Temizleyicisi', x= "Date", y = "quantity") + theme(te

h<-ggplot(data = mendil, aes(x = event_date, y = sold_count ,group=1))+
  geom_line(color = '#007cc3') + labs(title = 'Mendil', x= "Date", y = "quantity") + theme(text=element_

j<-ggplot(data = dis_fircasi, aes(x = event_date, y = sold_count ,group=1))+
  geom_line(color = '#007cc3') + labs(title = 'Diş Fırçası', x= "Date", y = "quantity") + theme(text=ele

grid.arrange(a,b,c,d,e,f,g,h,j ,
             ncol = 3, nrow = 3)
```



We see that there is a quite different range of sales quantity patterns. For example: mont and bikini_1 have some long zero timeline of sales, whereas bikini_2 seems to have an increasing trend and some seasonality.The other 6 products actually resemble each other in terms of noiseness.They all have single or multiple peaks and other than that they follow a white noisy pattern which is pretty good for modelling purposes but it harms our forecasting power, hence randomness effect play a significant role here by pushing us into this trade-off situation where modelling benefits from noiseness but forecasting takes damage from it.

Statistical Analysis

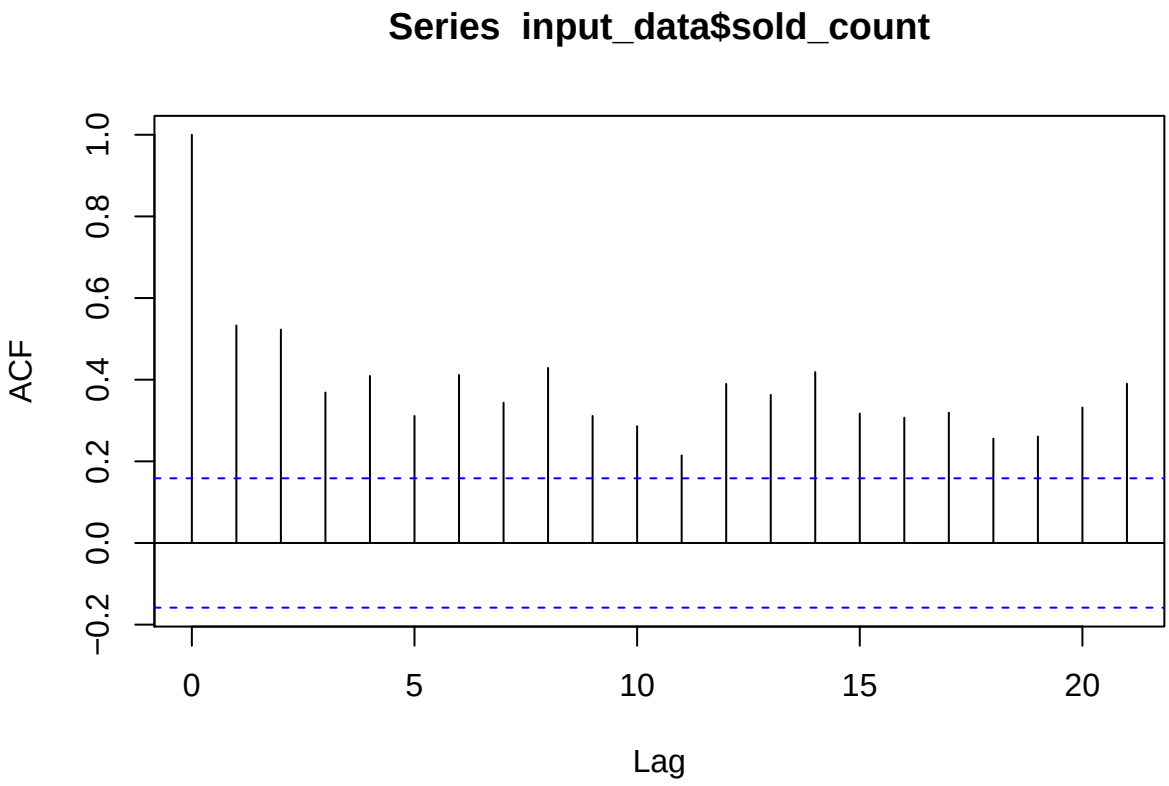
We now maybe interested in stationarity , partial correlation and autocorrelation relationships between lags , this will be useful in further statistical analysis and it will help us in arima part of the model.Here a handful function to study mentioned statistics :

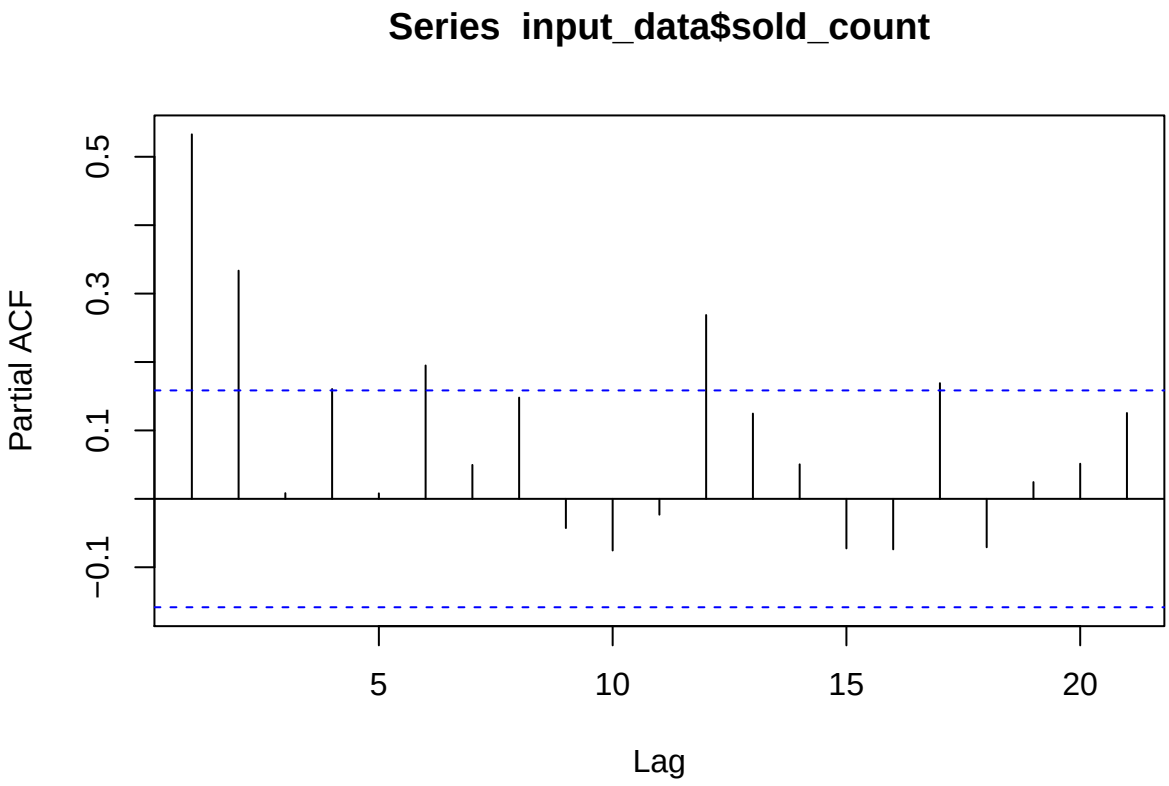
```
Statistical_Analysis <- function(X){
  input_data = X
  KPSS_test= ur.kpss(input_data$sold_count)
  print(summary(KPSS_test))
  acf(input_data$sold_count)
  pacf(input_data$sold_count)
}
```

Mont-statistics :

```
Statistical_Analysis(mont)
```

```
##
## ########################
## # KPSS Unit Root Test #
## ########################
##
## Test is of type: mu with 4 lags.
##
## Value of test-statistic is: 1.5466
##
## Critical value for a significance level of:
##                 10pct  5pct 2.5pct  1pct
## critical values 0.347 0.463  0.574 0.739
```

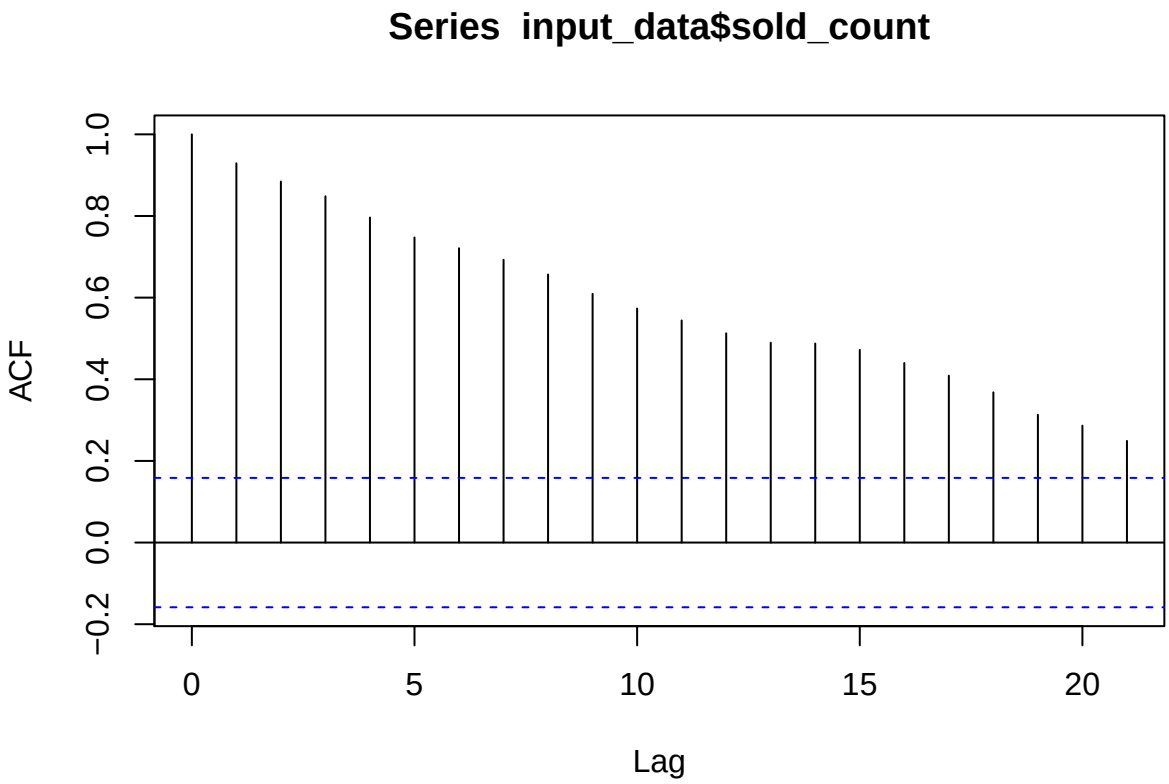


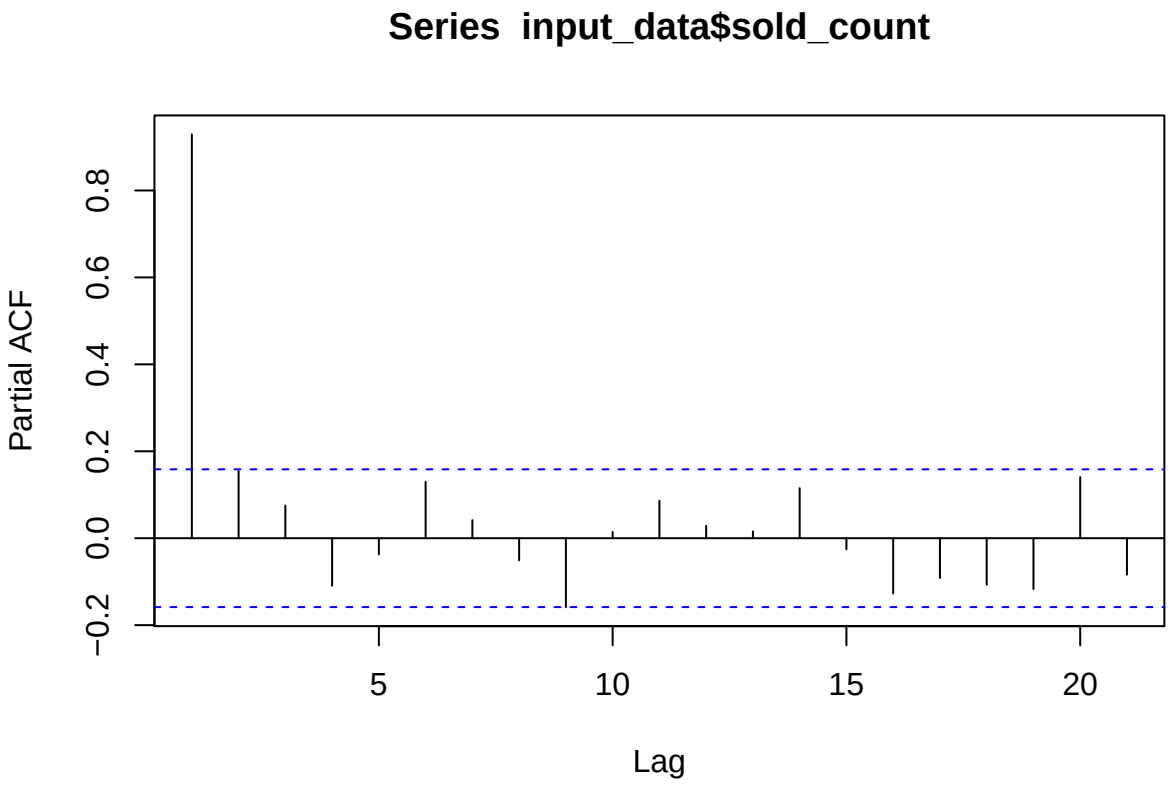


Bikini_1-statistics :

```
Statistical_Analysis(bikini_1)
```

```
##
## ########################
## # KPSS Unit Root Test #
## ########################
##
## Test is of type: mu with 4 lags.
##
## Value of test-statistic is: 1.1446
##
## Critical value for a significance level of:
##                 10pct  5pct 2.5pct  1pct
## critical values 0.347 0.463  0.574 0.739
```

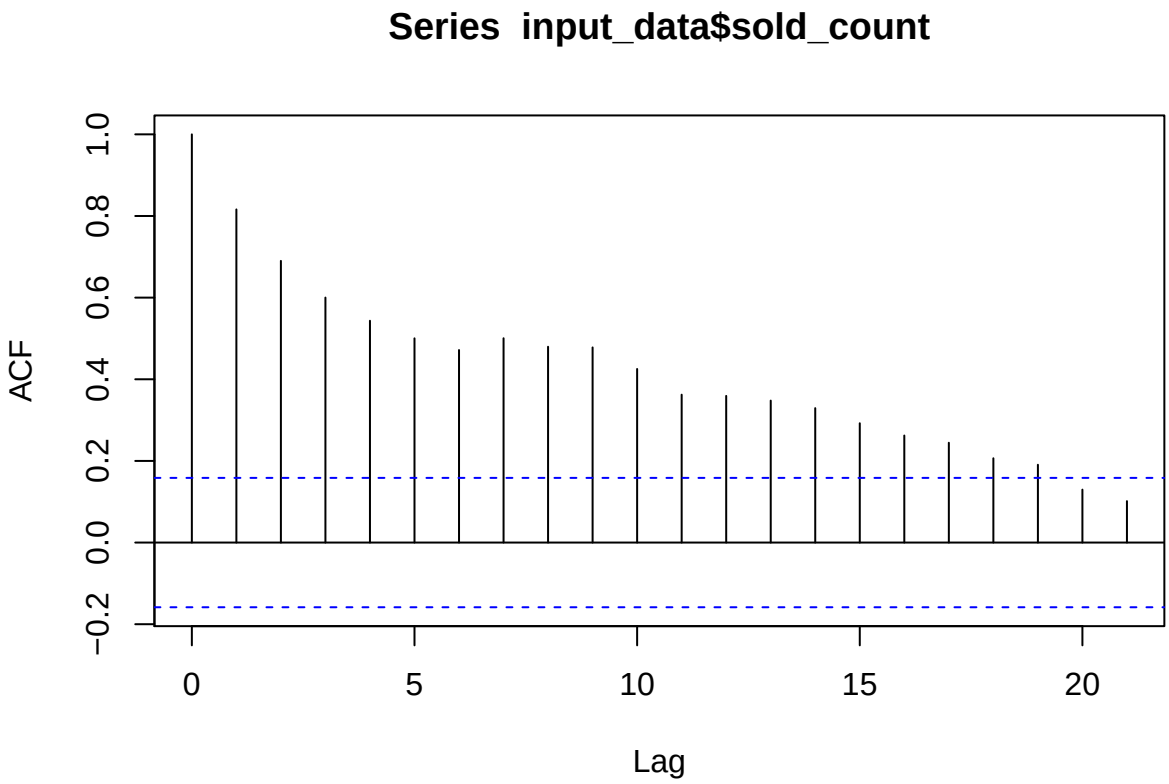

Series input_data$sold_count
ACF
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
0
5
10
15
20
Lag

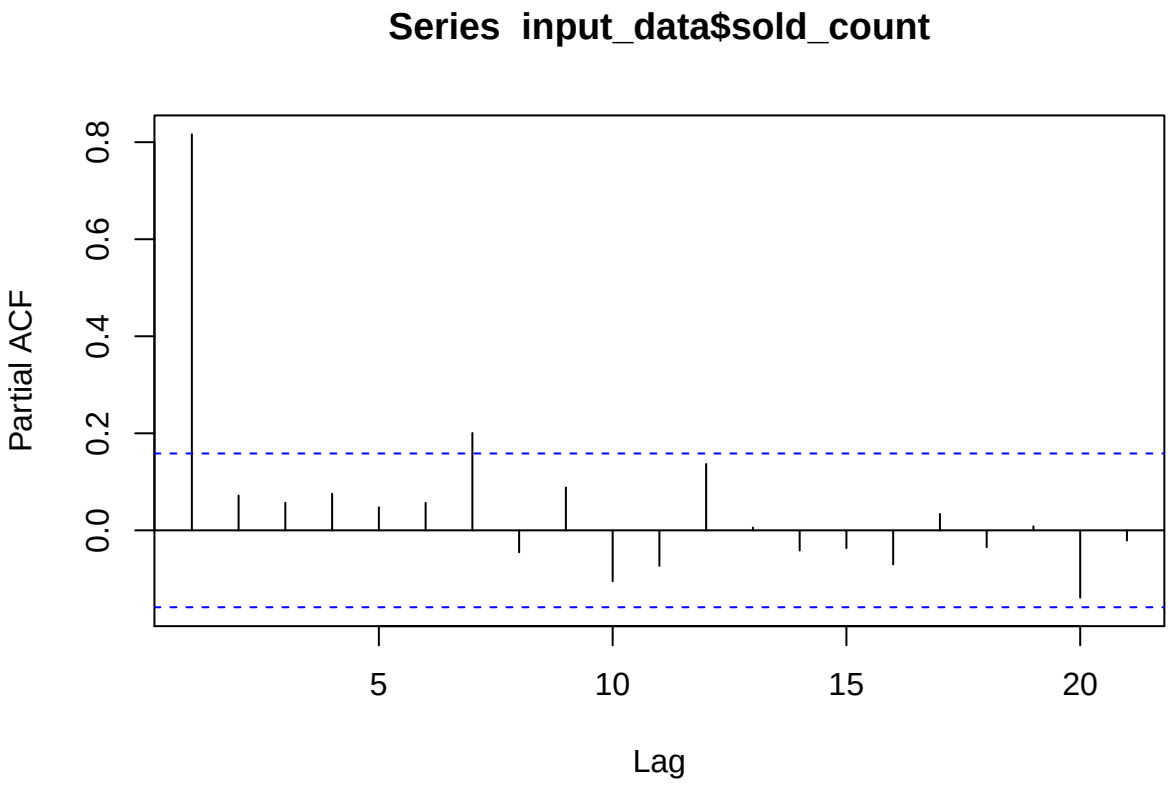


Bikini_2-statistics :

```
Statistical_Analysis(bikini_2)
```

```
##
## ########################
## # KPSS Unit Root Test #
## ########################
##
## Test is of type: mu with 4 lags.
##
## Value of test-statistic is: 1.9639
##
## Critical value for a significance level of:
##                 10pct  5pct 2.5pct  1pct
## critical values 0.347 0.463  0.574 0.739
```

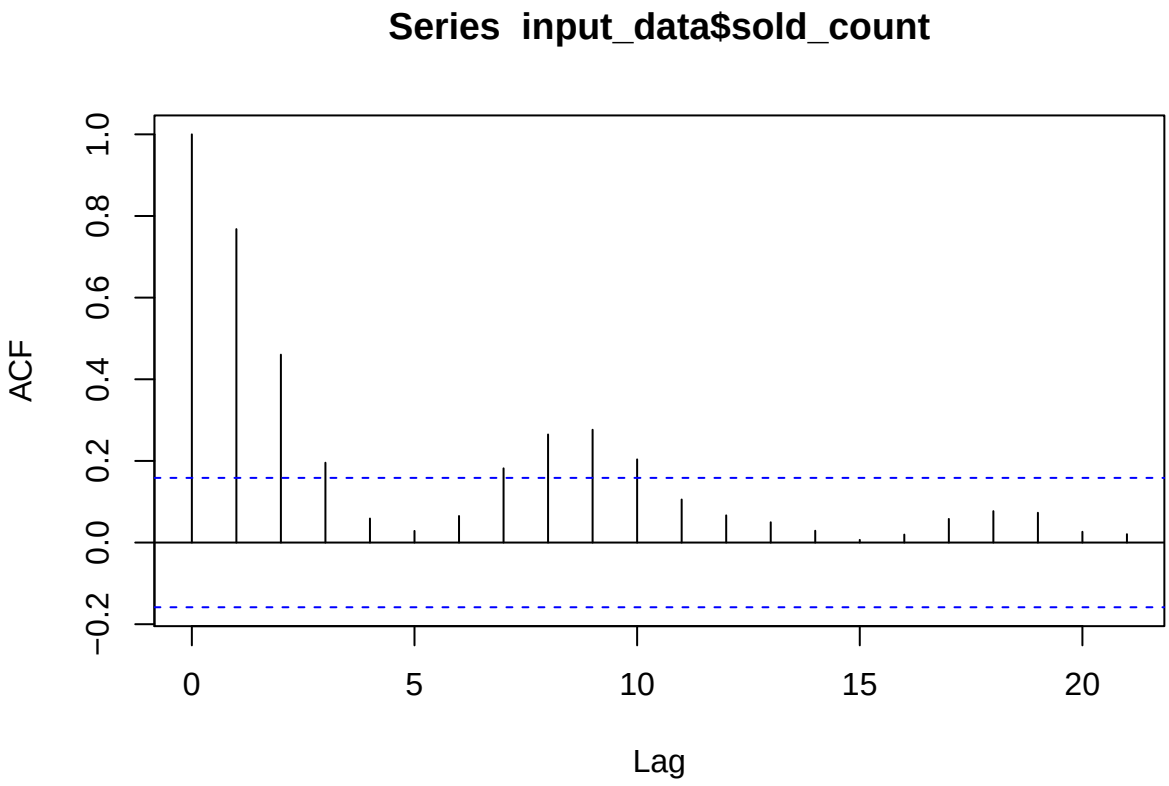

Series input_data$sold_count
ACF
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
0
5
10
15
20
Lag

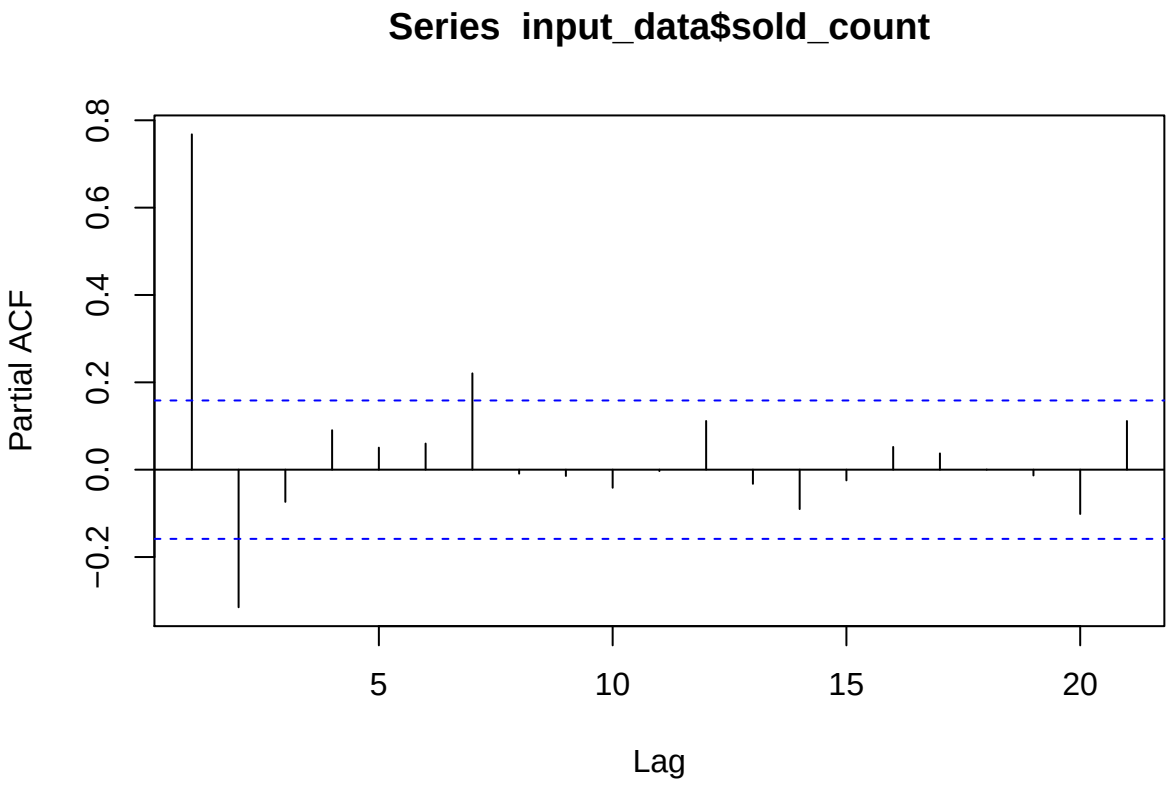


Tayt-statistics :

```
Statistical_Analysis(tayt)
```

```
##
## ########################
## # KPSS Unit Root Test #
## ########################
##
## Test is of type: mu with 4 lags.
##
## Value of test-statistic is: 0.8459
##
## Critical value for a significance level of:
##                 10pct  5pct 2.5pct  1pct
## critical values 0.347 0.463  0.574 0.739
```

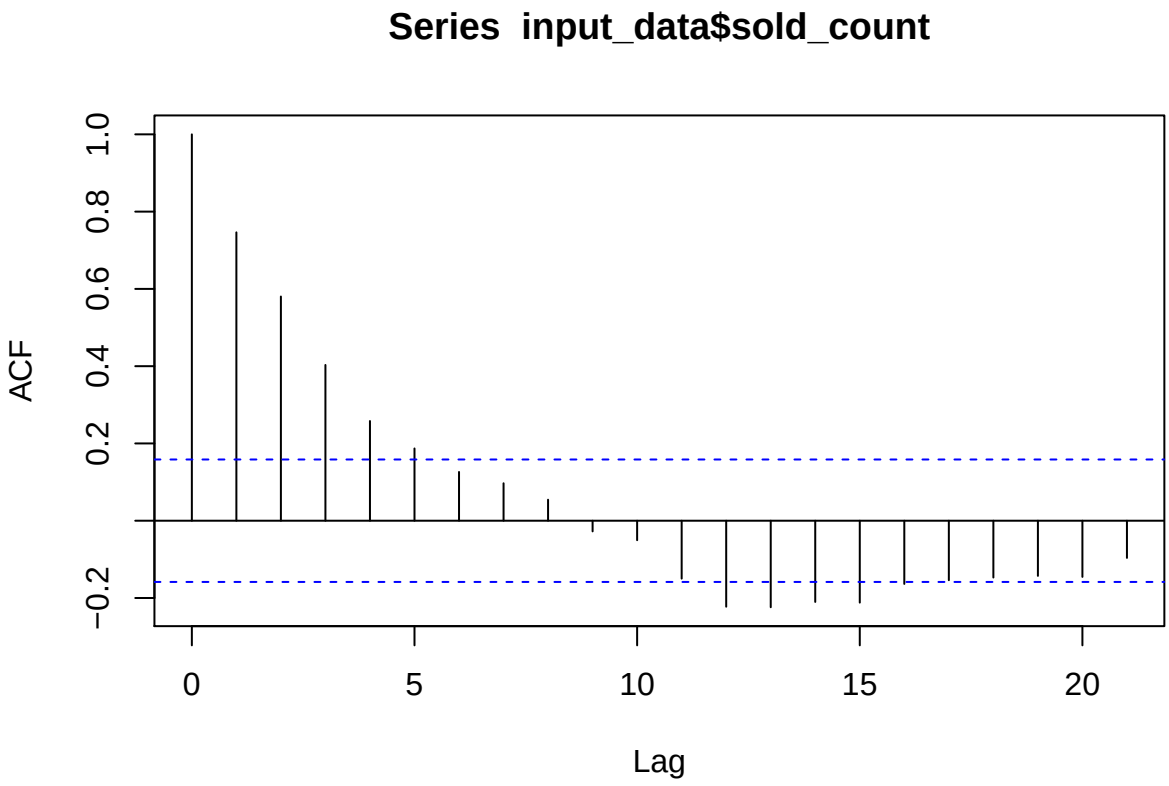

Series input_data$sold_count
ACF
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
0
5
10
15
20
Lag

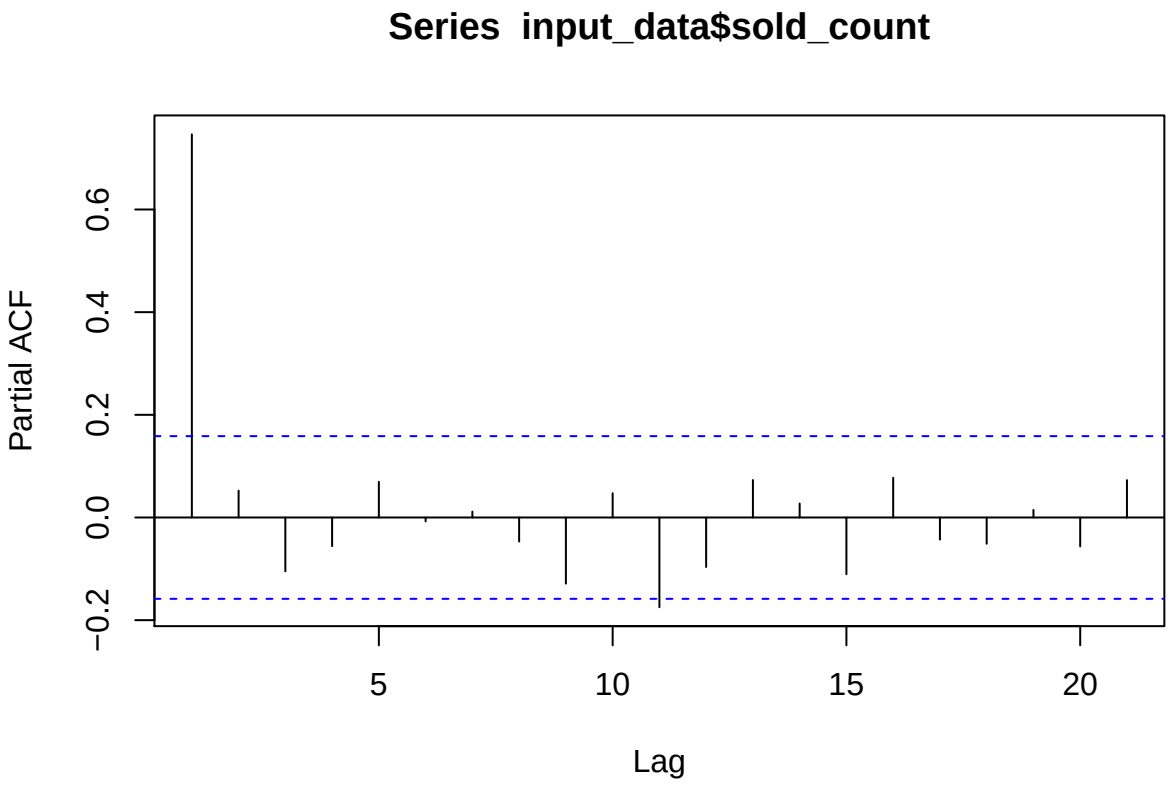


Kulaklik-statistics :

```
Statistical_Analysis(kulaklık)
```

```
##
## ########################
## # KPSS Unit Root Test #
## ########################
##
## Test is of type: mu with 4 lags.
##
## Value of test-statistic is: 0.2252
##
## Critical value for a significance level of:
##                 10pct  5pct 2.5pct  1pct
## critical values 0.347 0.463  0.574 0.739
```

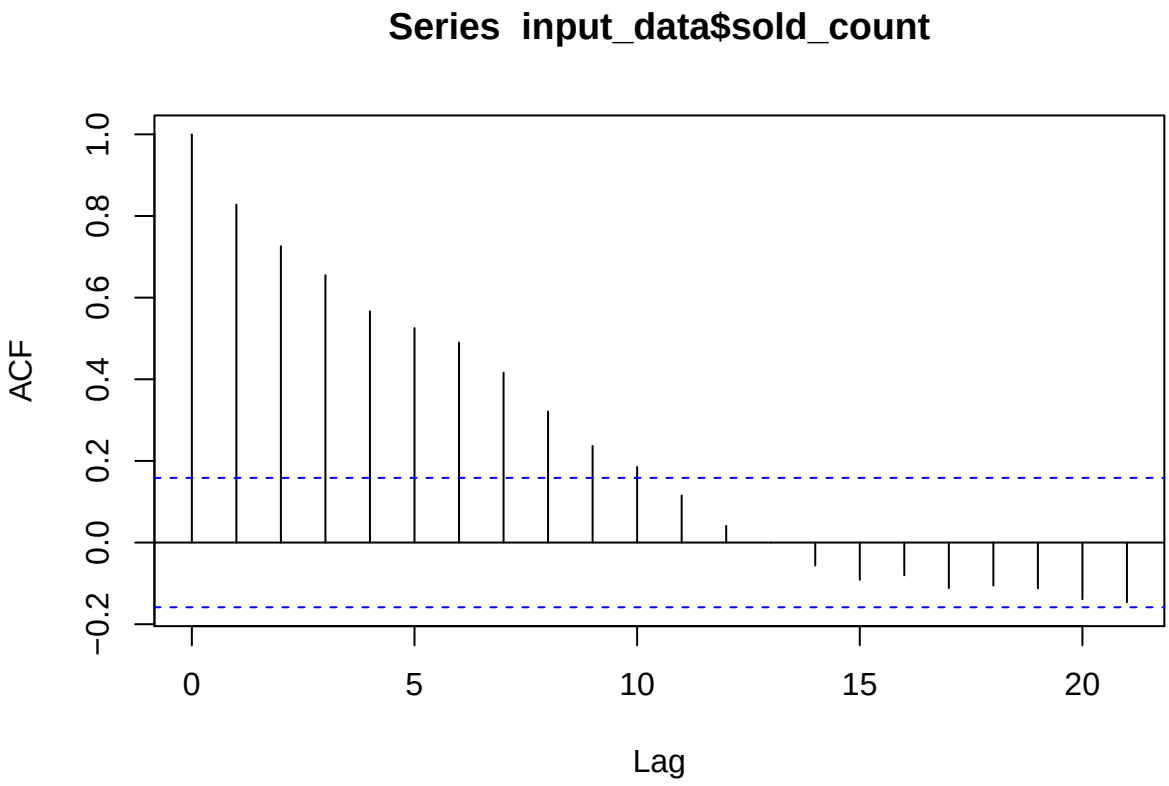

Series input_data$sold_count
ACF
1.0
0.8
0.6
0.4
0.2
−0.2
0
5
10
15
20
Lag

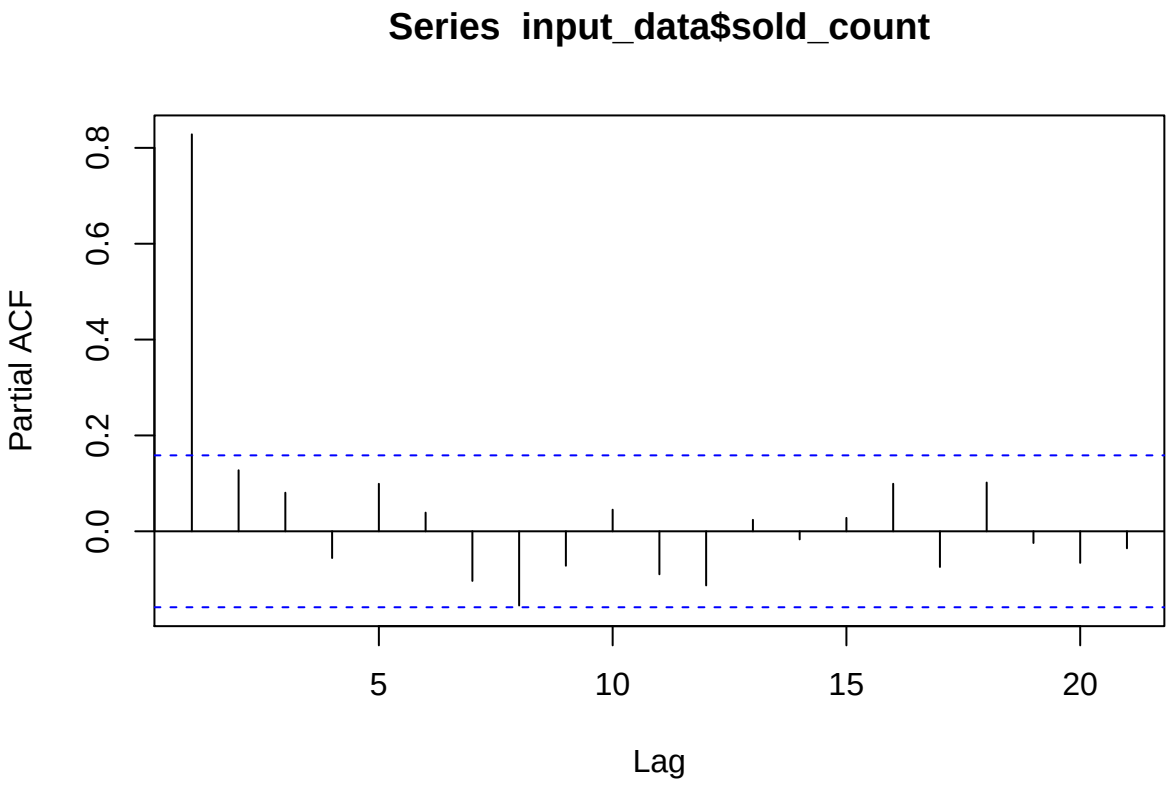


Supurge-statistics :

```
Statistical_Analysis(supurge)
```

```
##
## ########################
## # KPSS Unit Root Test #
## ########################
##
## Test is of type: mu with 4 lags.
##
## Value of test-statistic is: 0.2692
##
## Critical value for a significance level of:
##                 10pct  5pct 2.5pct  1pct
## critical values 0.347 0.463  0.574 0.739
```

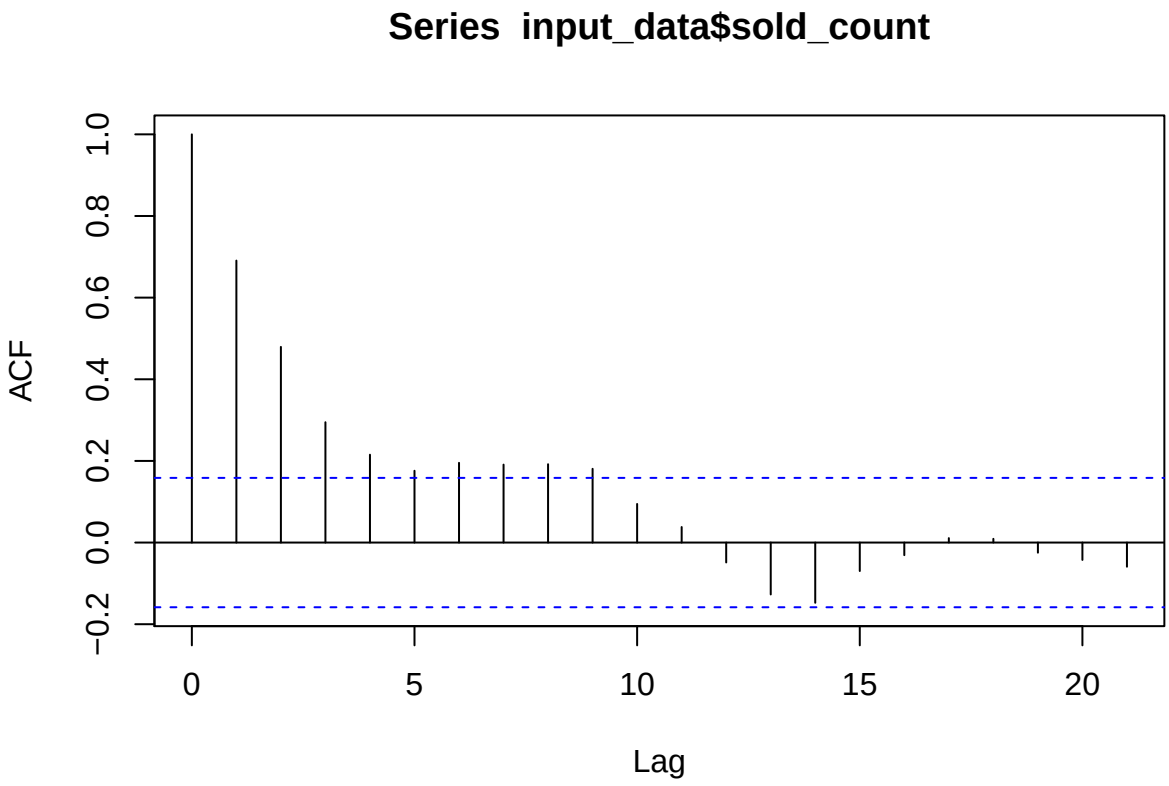

Series input_data$sold_count
ACF
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
0
5
10
15
20
Lag

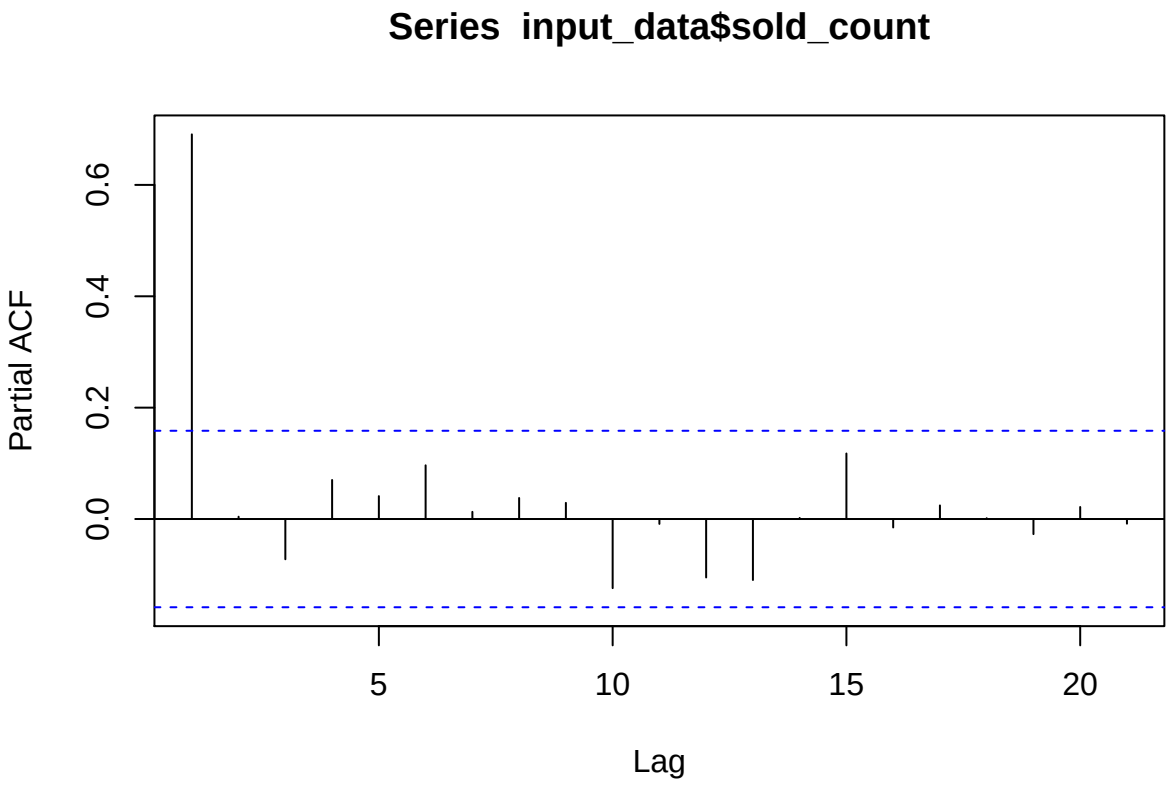


Yuz_temizleyicisi-statistics :

```
Statistical_Analysis(yuz_temizleyicisi)
```

```
##
## ########################
## # KPSS Unit Root Test #
## ########################
##
## Test is of type: mu with 4 lags.
##
## Value of test-statistic is: 0.1987
##
## Critical value for a significance level of:
##                 10pct  5pct 2.5pct  1pct
## critical values 0.347 0.463  0.574 0.739
```

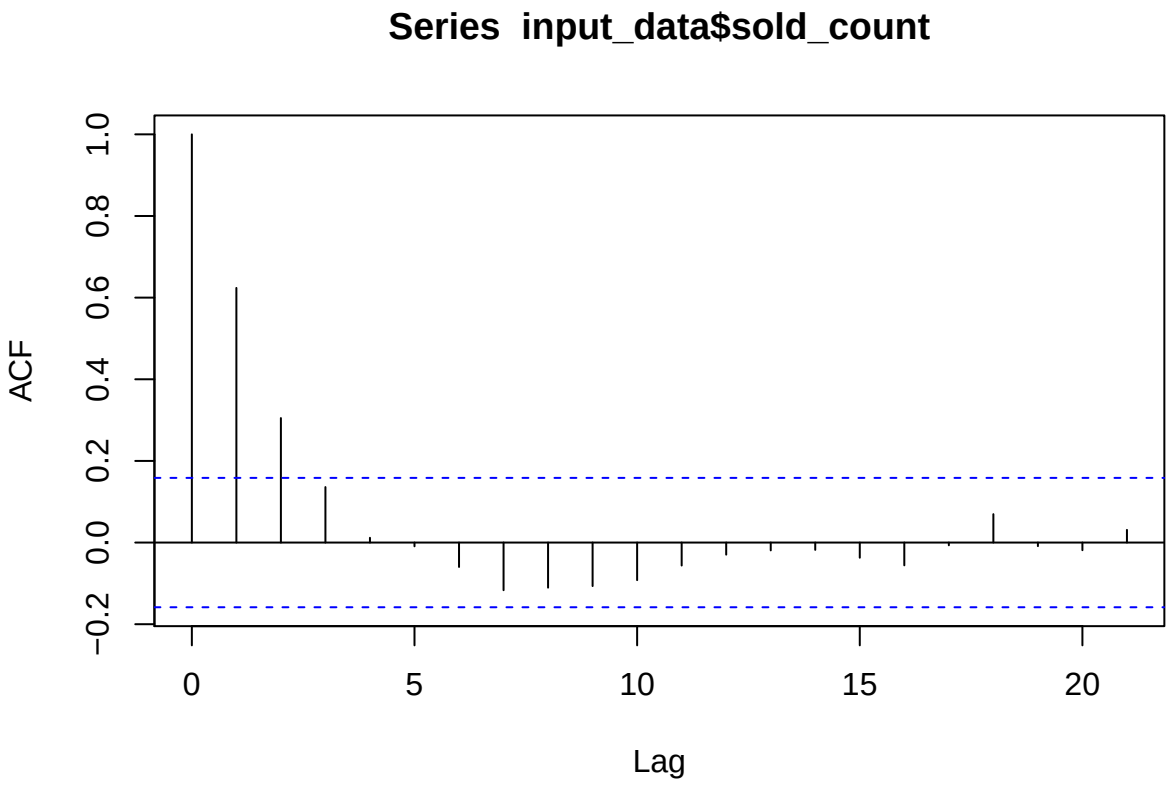

Series input_data$sold_count
ACF
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
0
5
10
15
20
Lag

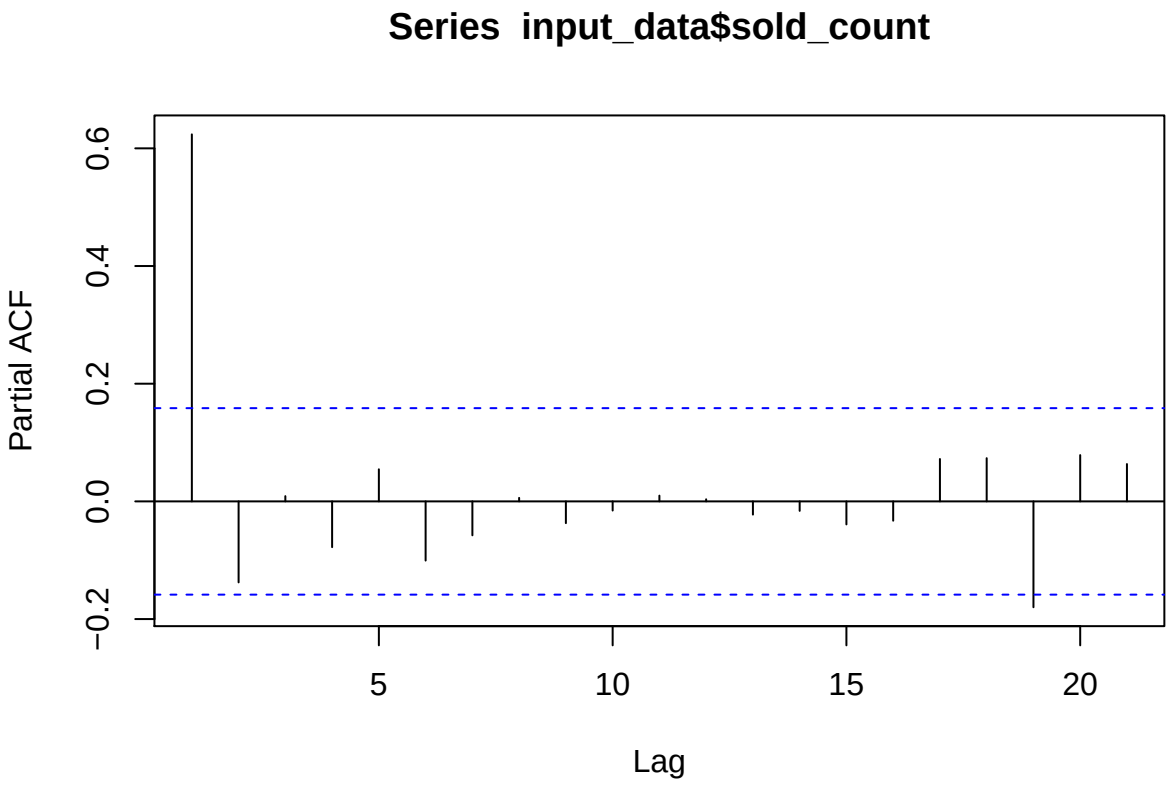


Mendil-statistics :

```
Statistical_Analysis(mendil)
```

```
##
## ########################
## # KPSS Unit Root Test #
## ########################
##
## Test is of type: mu with 4 lags.
##
## Value of test-statistic is: 0.3888
##
## Critical value for a significance level of:
##                 10pct  5pct 2.5pct  1pct
## critical values 0.347 0.463  0.574 0.739
```

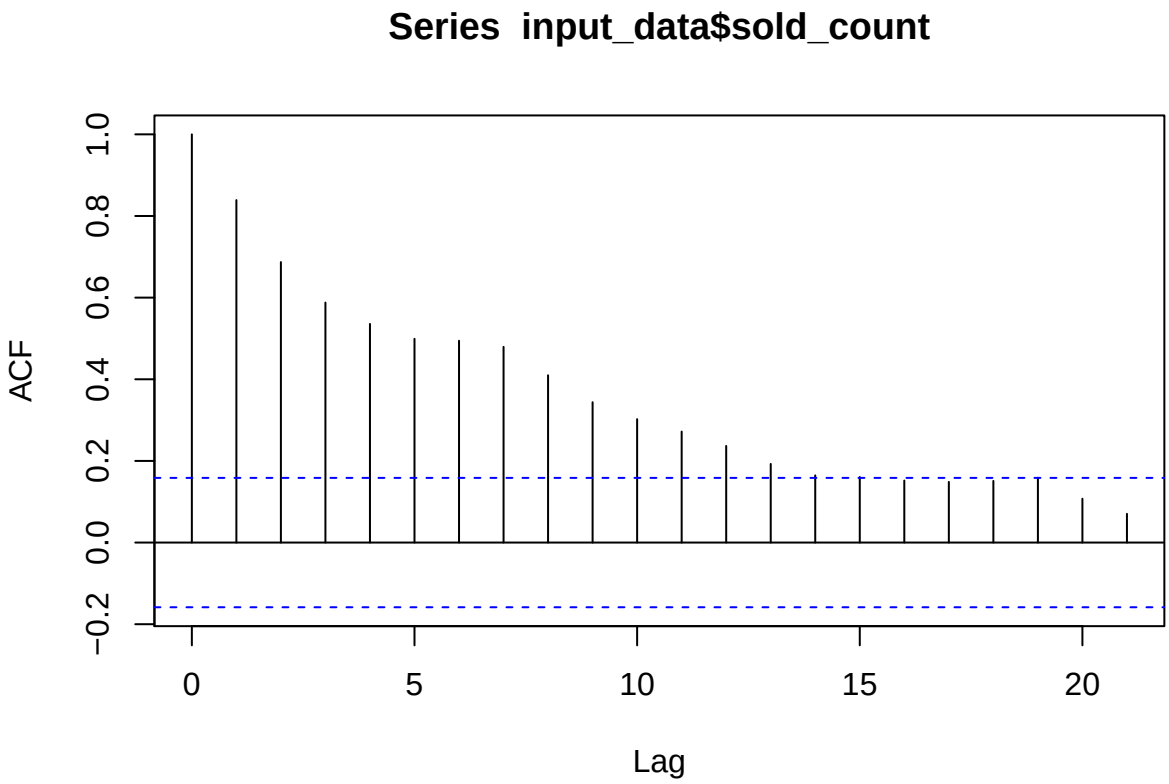

Series input_data$sold_count
ACF
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
0
5
10
15
20
Lag

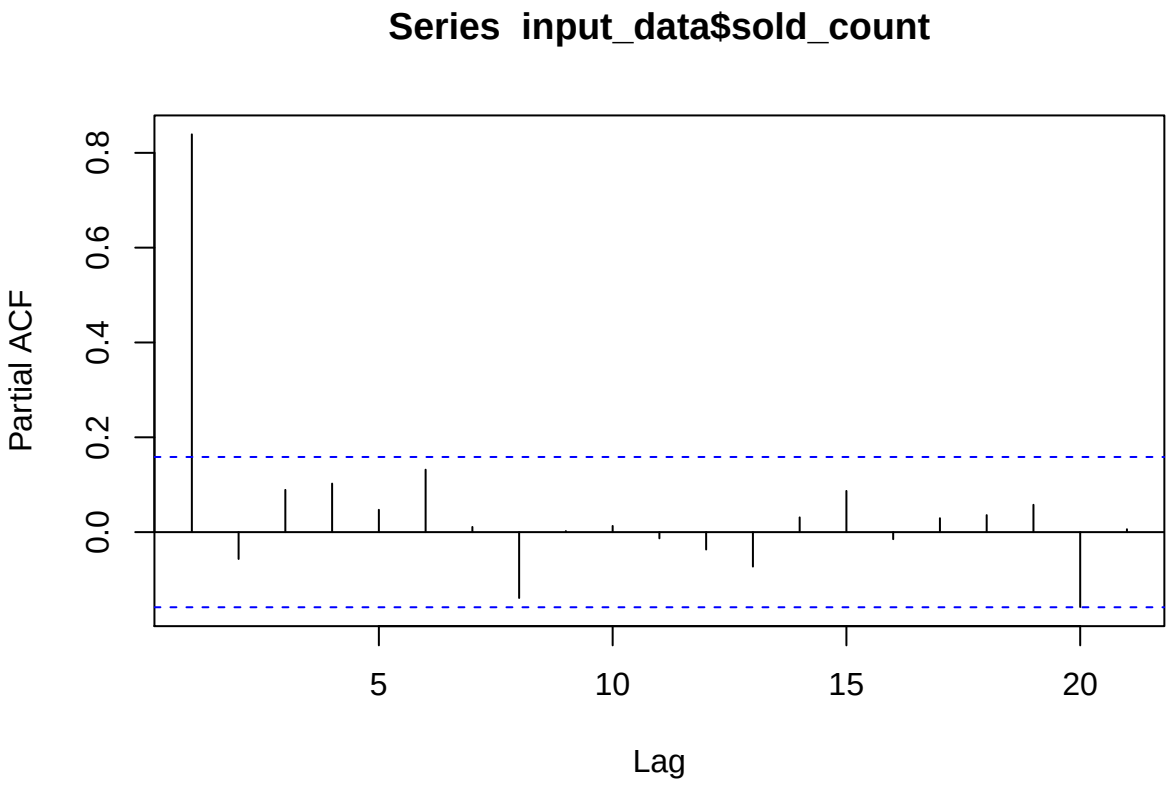


Dis_fircasi-statistics :

```
Statistical_Analysis(dis_fircasi)
```

```
##
## ########################
## # KPSS Unit Root Test #
## ########################
##
## Test is of type: mu with 4 lags.
##
## Value of test-statistic is: 0.6284
##
## Critical value for a significance level of:
##                 10pct  5pct 2.5pct  1pct
## critical values 0.347 0.463  0.574 0.739
```


Series input_data$sold_count
ACF
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
0
5
10
15
20
Lag



To evaluate each individual product in terms of stationarity and autocorrelations we need to mention a general approach. In KPSS test, we desire to approve the null hypothesis stating that the tested data is stationary, hence we want to get the lowest possible test-statistic even below 10-pct confidence interval. From ACF graph , we can get the notion of selecting correct number of moving average lags and from PACF graph , we can decide a sufficient p parameter for autoregression in ARIMA, but we wont do arima that way, its just a statistical perspective and this investigations help us to familiarize with the product sales data.

Interpretation of those statistics in a table form

The overall statistics interpretations are here :

```
product = c("mont","bikini_1","bikini_2 ","tayt","kulaklık","supurge","yuz_temizleyicisi","mendil","dis

stationarity = c("non-stationary","non-stationary","non-stationary","non-stationary","stationary","stati

p_parameter_for_arima = c("2","1","1","2","1","1","1","1","1")
q_parameter_for_arima = c("3","trend occurs due to decay","trend occurs due to decay","3","trend occurs

table_2=cbind(product,stationarity,p_parameter_for_arima,q_parameter_for_arima)
print(table_2)
```

```
##       product             stationarity                p_parameter_for_arima
##  [1,] "mont"              "non-stationary"            "2"
##  [2,] "bikini_1"          "non-stationary"            "1"
##  [3,] "bikini_2 "         "non-stationary"            "1"
```

```
##  [4,] "tayt"              "non-stationary"              "2"
##  [5,] "kulaklık"          "stationary"                  "1"
##  [6,] "supurge"           "stationary"                  "1"
##  [7,] "yuz_temizleyicisi" "stationary"                  "1"
##  [8,] "mendil"            "stationary(except in 10pct)" "1"
##  [9,] "dis_fircasi"       "stationary(only for 1pct)"   "1"
##       q_parameter_for_arima
##  [1,] "3"
##  [2,] "trend occurs due to decay"
##  [3,] "trend occurs due to decay"
##  [4,] "3"
##  [5,] "trend occurs due to decay"
##  [6,] "trend occurs due to decay"
##  [7,] "4"
##  [8,] "3"
##  [9,] "trend occurs due to decay"
```

Lag addition

Here we can conclude that lag1 is quite dominant in almost all product's sales quantity and more than half of them are exposed to a trend and some of the products already follow a white noisy pattern. Here we concluded that shifting sales quantities by 1,i.e. lag1, can help us to get rid of error accumulation and we can get better arima models in future. Lags are added :

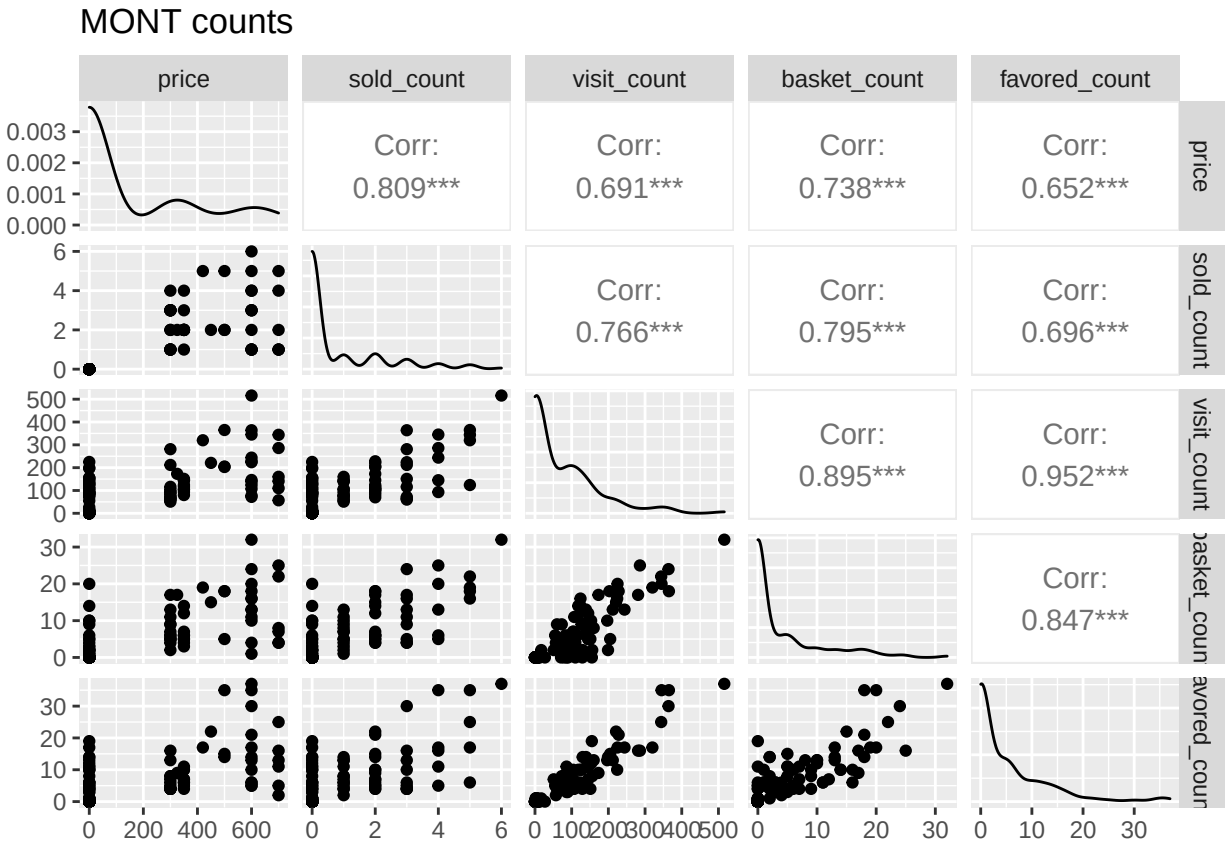
```
bikini_1[,lagg:=shift(x = bikini_1$sold_count,n = 1)]
bikini_1$lagg[1]=round(mean(bikini_1$lagg[2:4]))


bikini_2[,lagg:=shift(x = bikini_2$sold_count,n = 1)]
bikini_2$lagg[1]=round(mean(bikini_2$lagg[2:4]))


dis_fircasi[,lagg:=shift(x = dis_fircasi$sold_count,n = 1)]
dis_fircasi$lagg[1]=round(mean(dis_fircasi$lagg[2:4]))


kulaklık[,lagg:=shift(x = kulaklık$sold_count,n = 1)]
kulaklık$lagg[1]=round(mean(kulaklık$lagg[2:4]))


mendil[,lagg:=shift(x = mendil$sold_count,n = 1)]
mendil$lagg[1]=round(mean(mendil$lagg[2:4]))


mont[,lagg:=shift(x = mont$sold_count,n = 1)]
mont$lagg[1]=round(mean(mont$lagg[2:4]))


supurge[,lagg:=shift(x = supurge$sold_count,n = 1)]
supurge$lagg[1]=round(mean(supurge$lagg[2:4]))


tayt[,lagg:=shift(x = tayt$sold_count,n = 1)]
```

```
tayt$lagg[1]=round(mean(tayt$lagg[2:4]))


yuz_temizleyicisi[,lagg:=shift(x = yuz_temizleyicisi$sold_count,n = 1)]
yuz_temizleyicisi$lagg[1]=round(mean(yuz_temizleyicisi$lagg[2:5]))
```

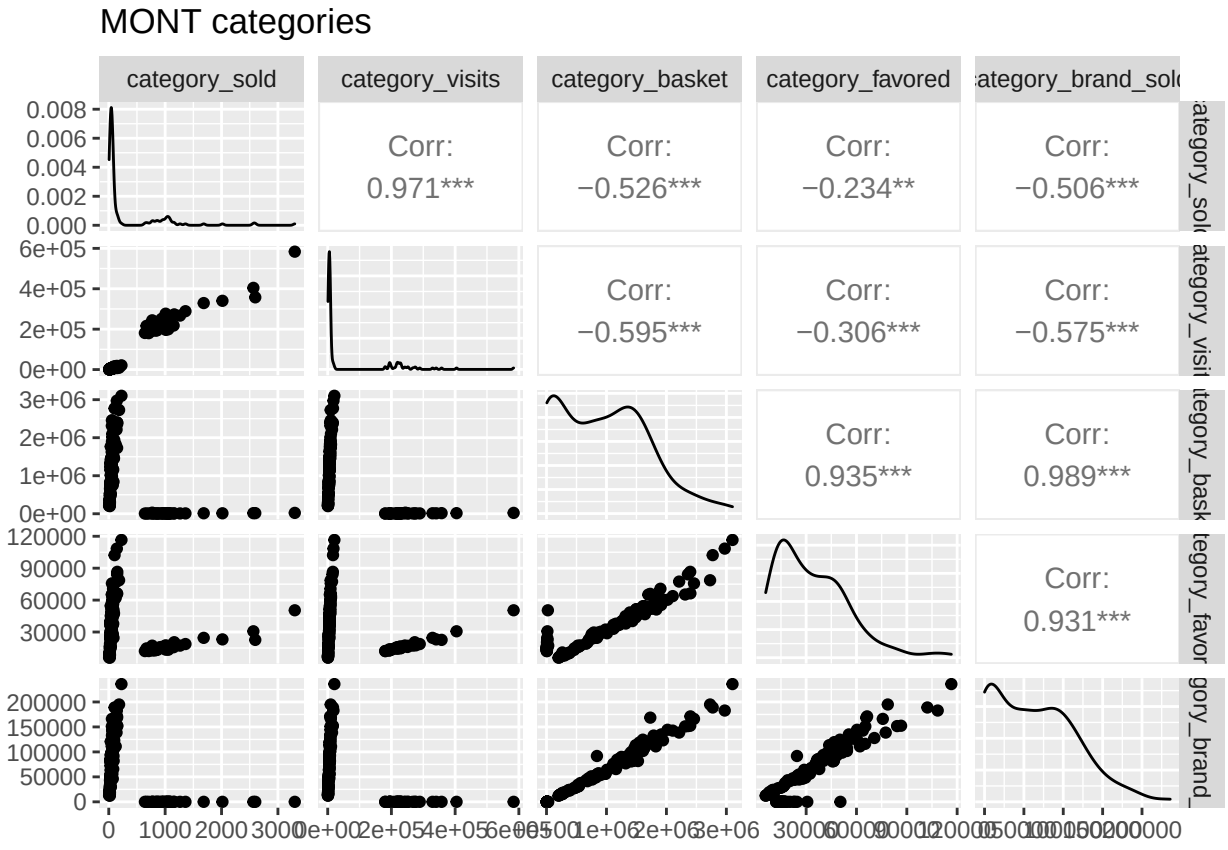
More detailed individual product investigation

We shall also look at correlations and plots of different regressors for each product.

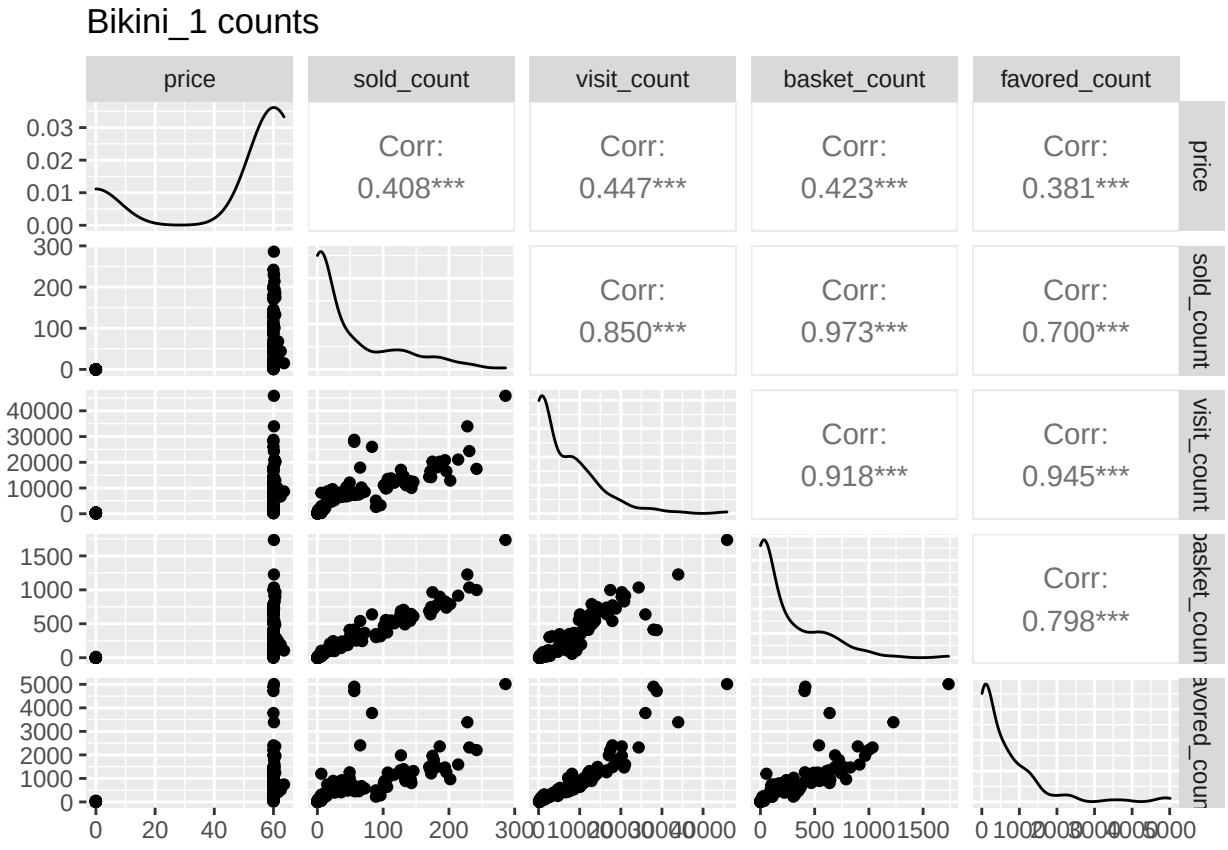
```
ggpairs(mont, columns = c(3,4,5,6,7),title = "MONT counts")
```



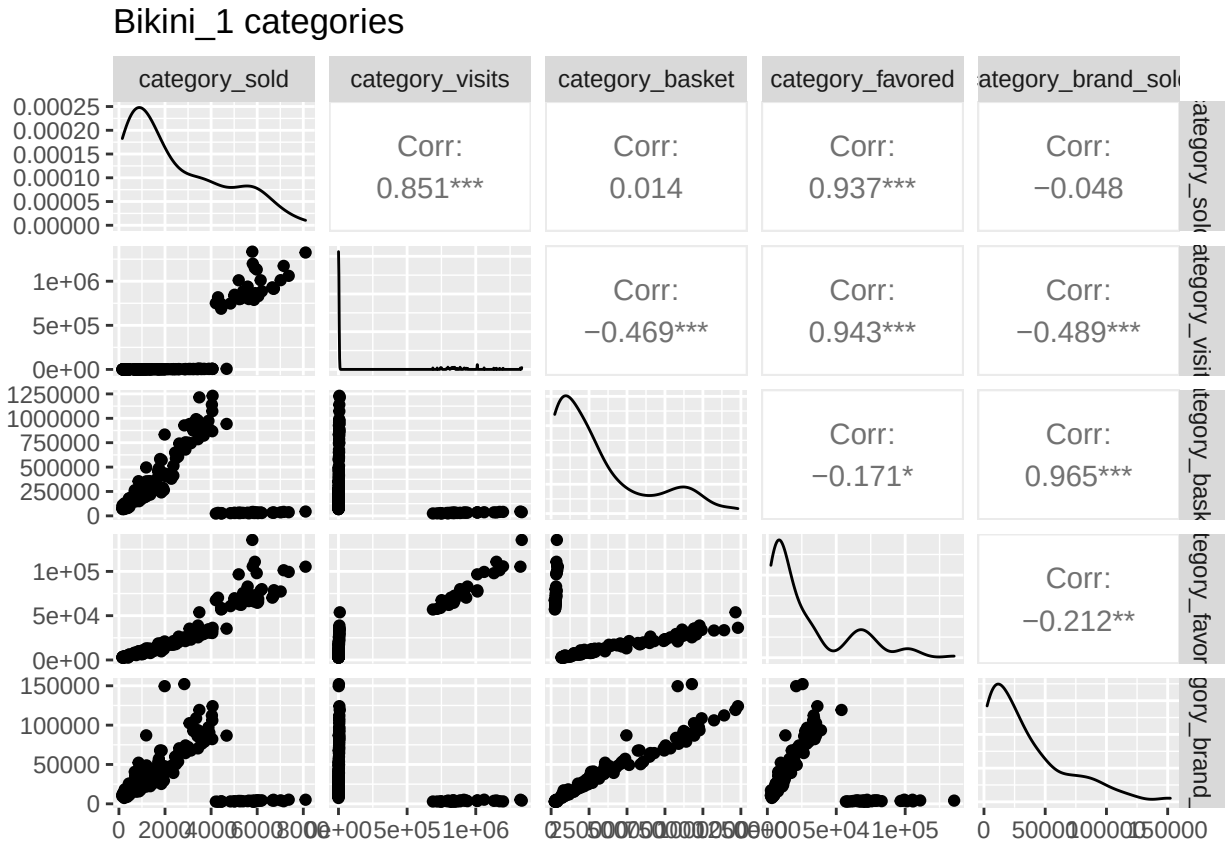

```
ggpairs(mont, columns = c(8,9,10,11,12),title ="MONT categories")
```



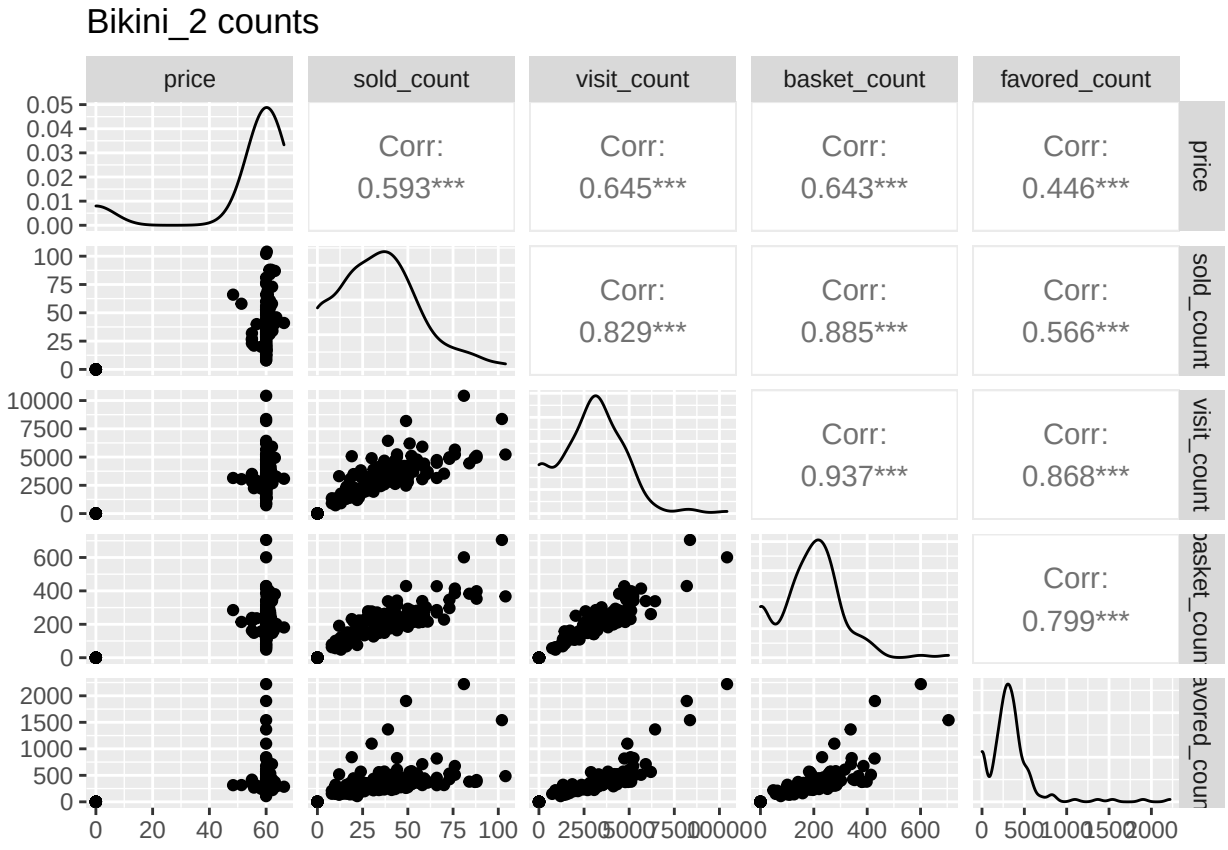

```
ggpairs(bikini_1, columns = c(3,4,5,6,7),title = "Bikini_1 counts")
```



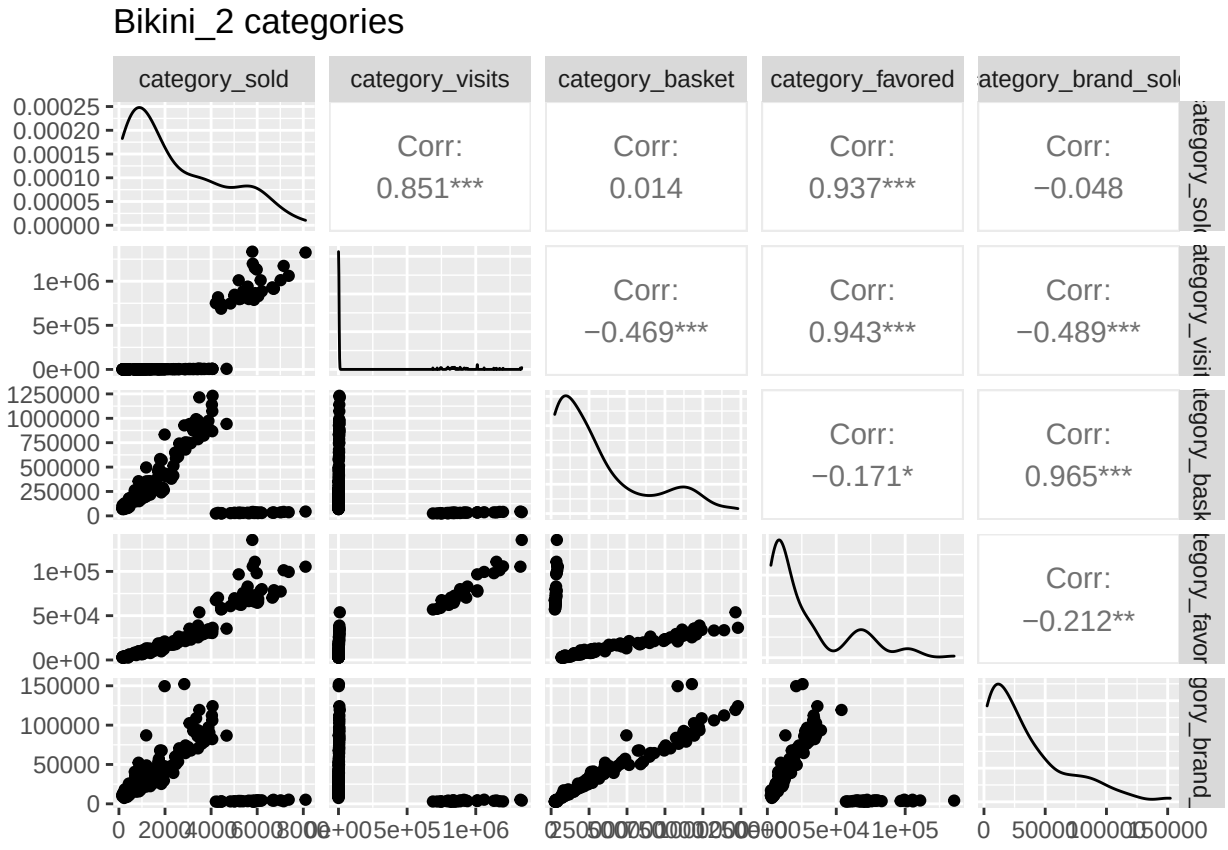

```
ggpairs(bikini_1, columns = c(8,9,10,11,12),title ="Bikini_1 categories")
```



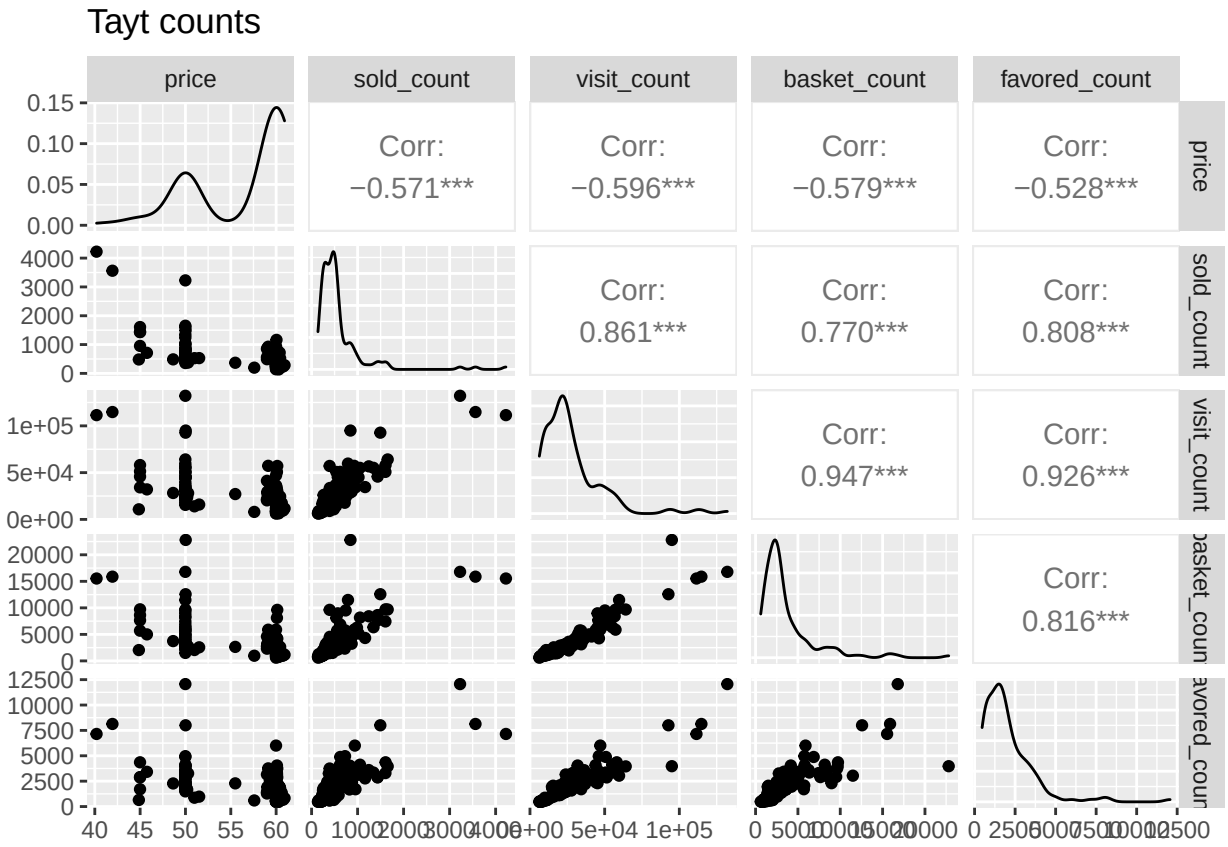

```
ggpairs(bikini_2, columns = c(3,4,5,6,7),title = "Bikini_2 counts")
```



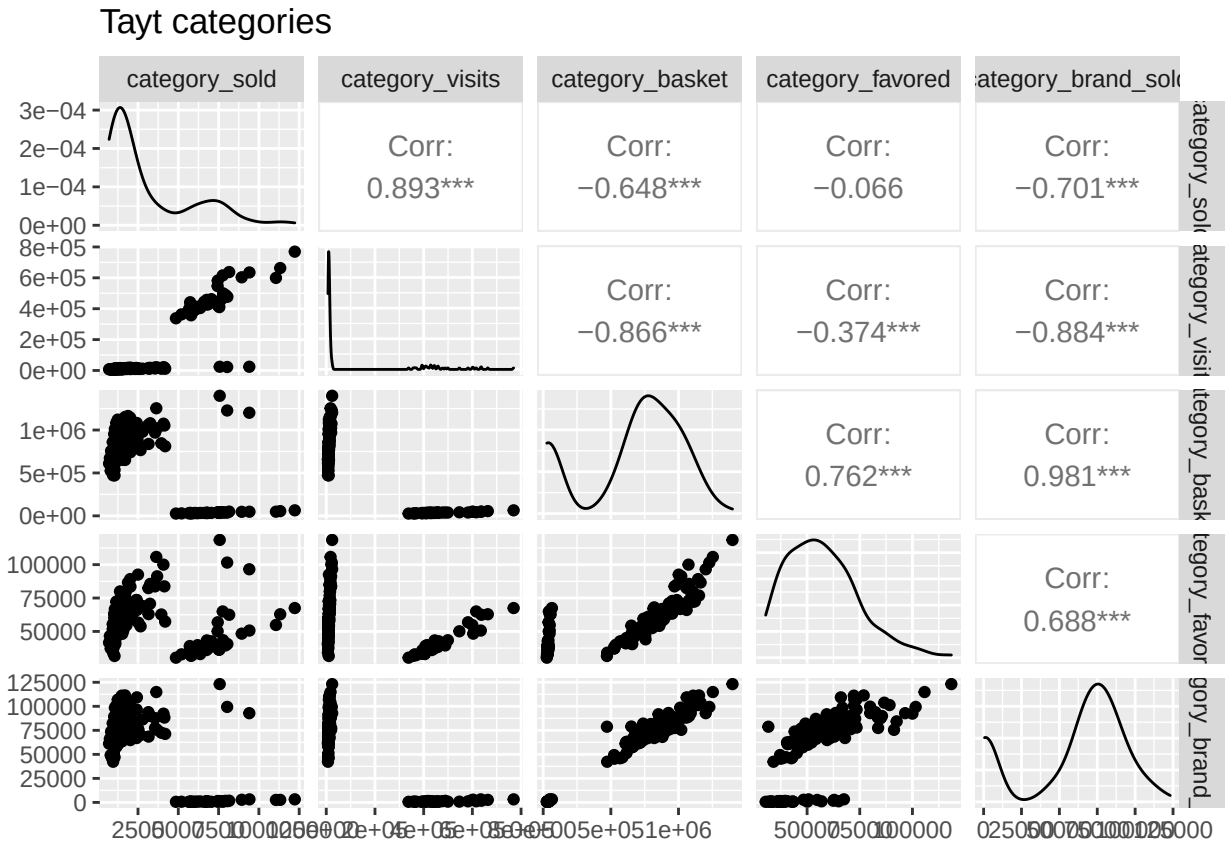

```
ggpairs(bikini_2, columns = c(8,9,10,11,12),title ="Bikini_2 categories")
```



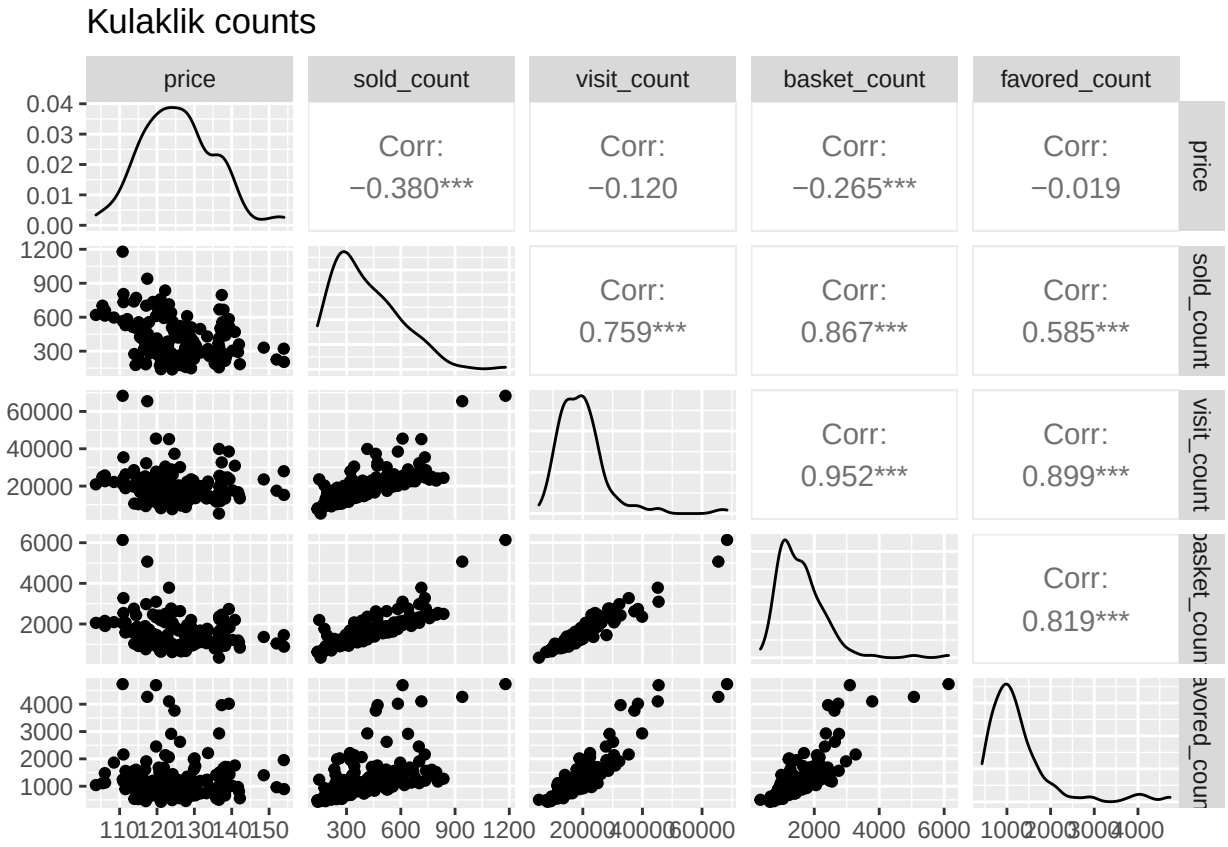

```
ggpairs(tayt, columns = c(3,4,5,6,7),title = "Tayt counts")
```



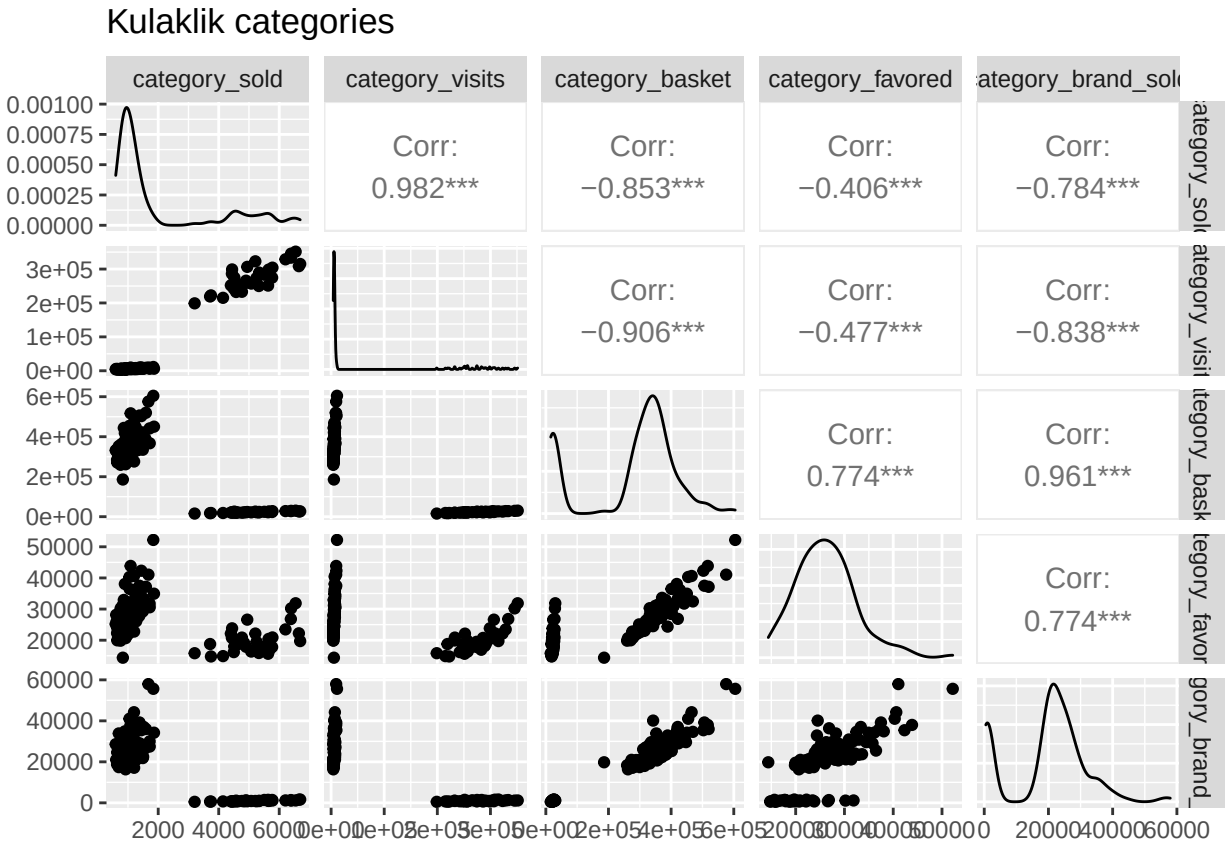

```
ggpairs(tayt, columns = c(8,9,10,11,12),title ="Tayt categories")
```



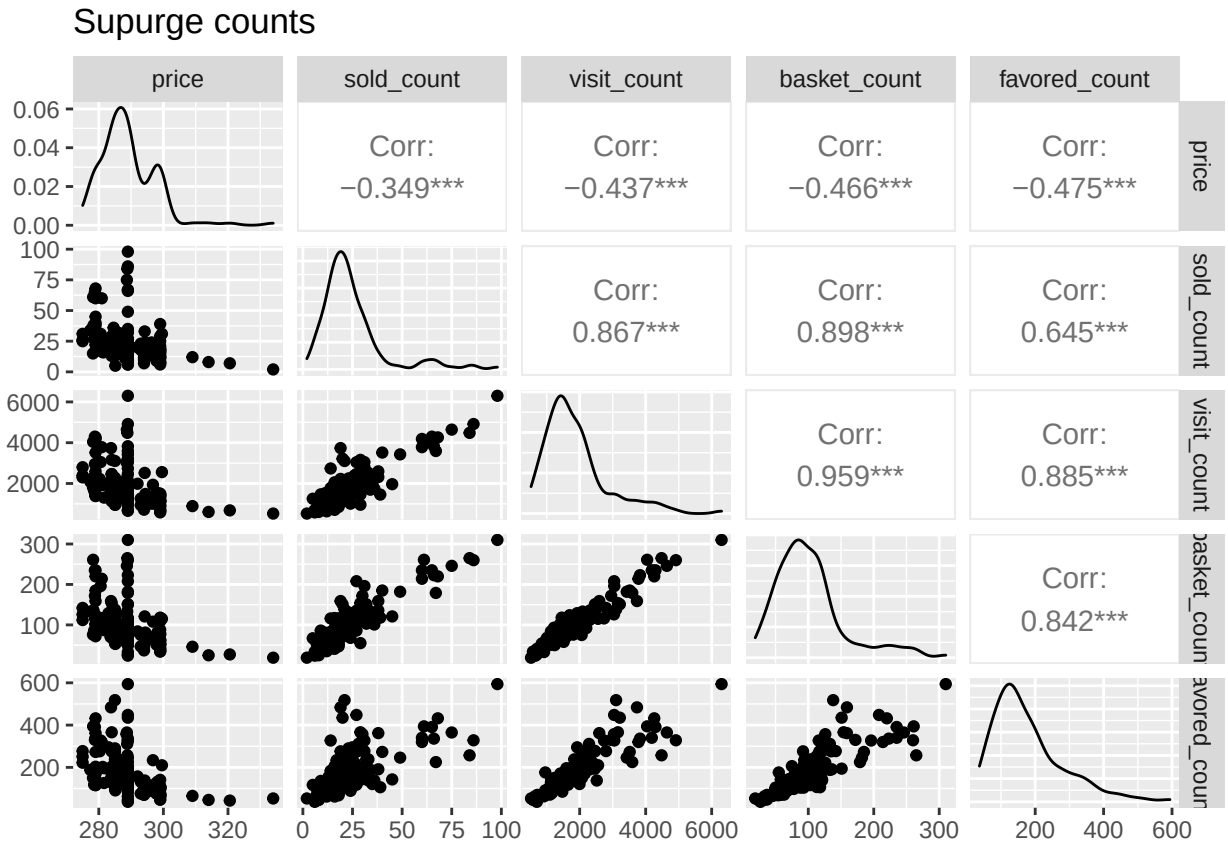

```
ggpairs(kulaklık, columns = c(3,4,5,6,7),title = "Kulaklık counts")
```



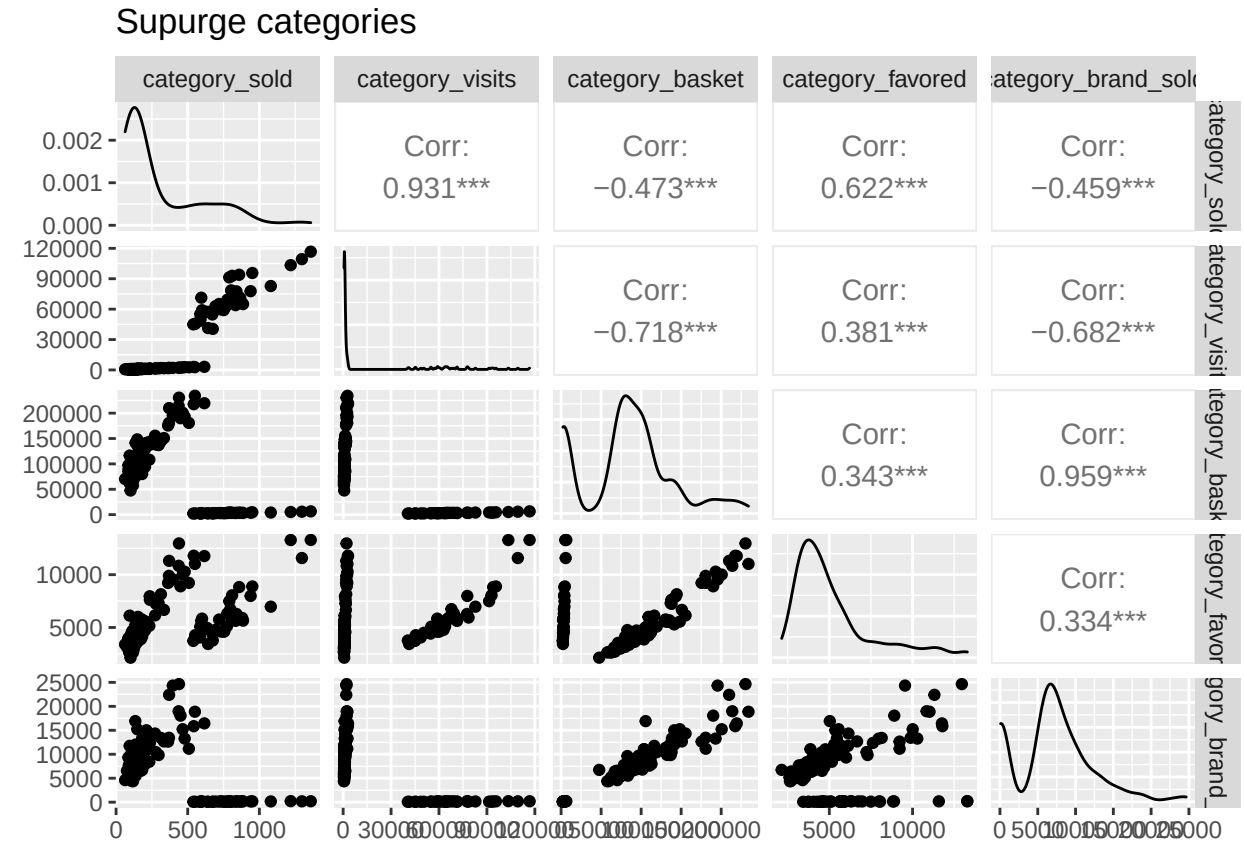

```
ggpairs(kulaklık, columns = c(8,9,10,11,12),title ="Kulaklık categories")
```



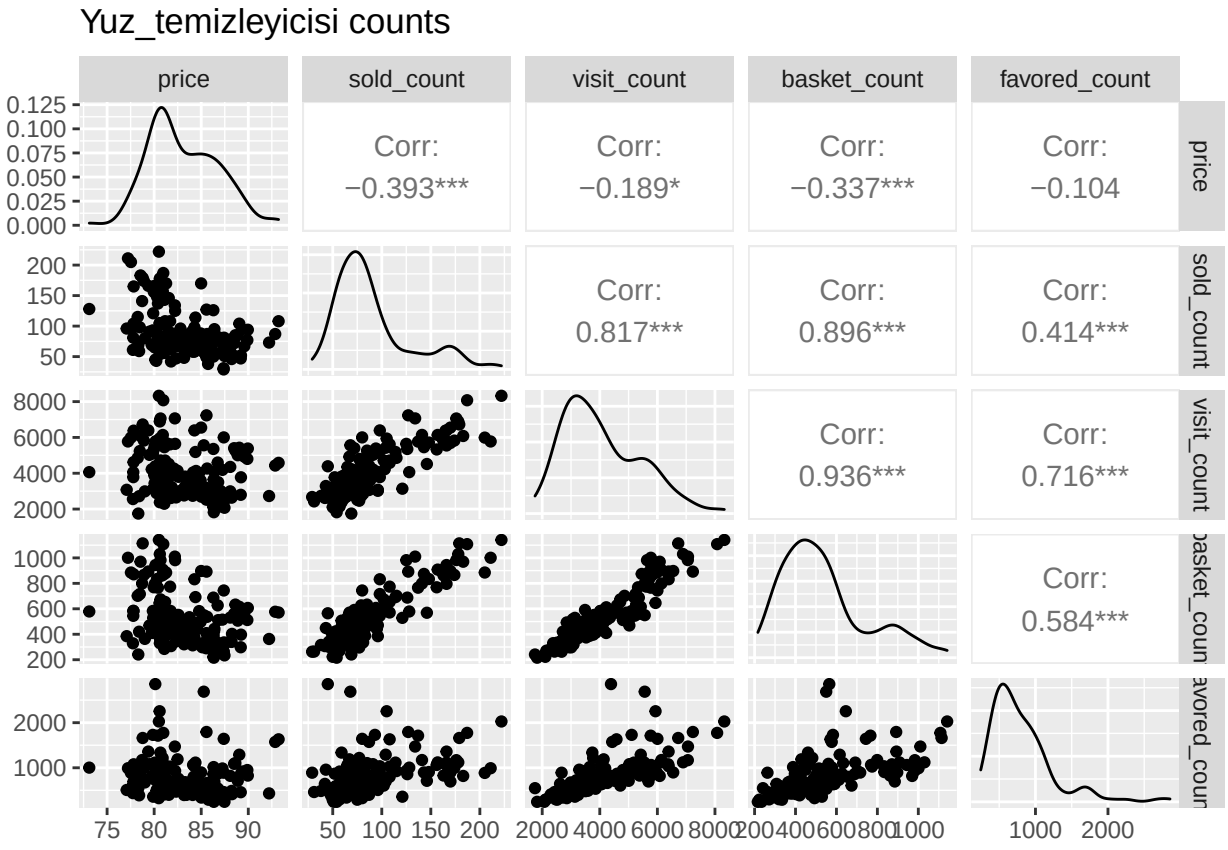

```
ggpairs(supurge, columns = c(3,4,5,6,7),title = "Supurge counts")
```



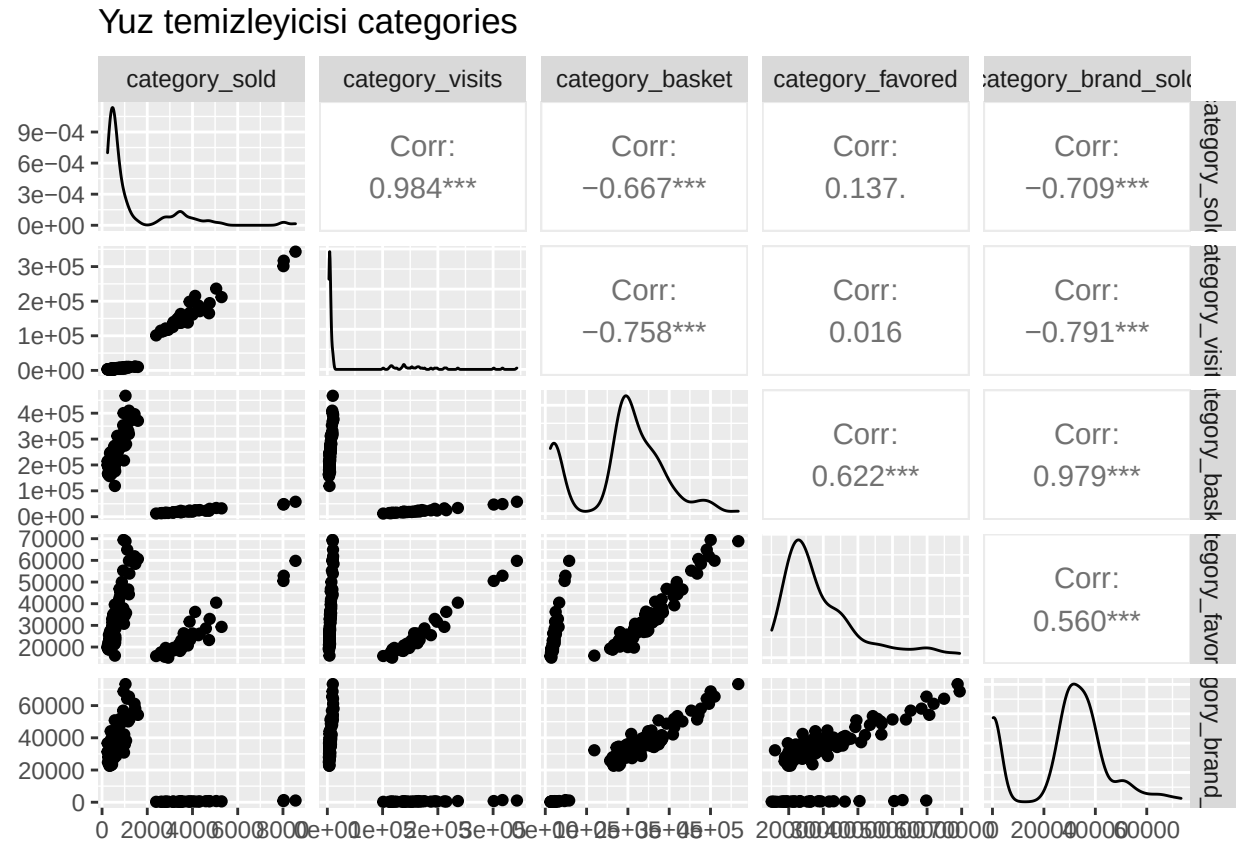

```
ggpairs(supurge, columns = c(8,9,10,11,12),title ="Supurge categories")
```



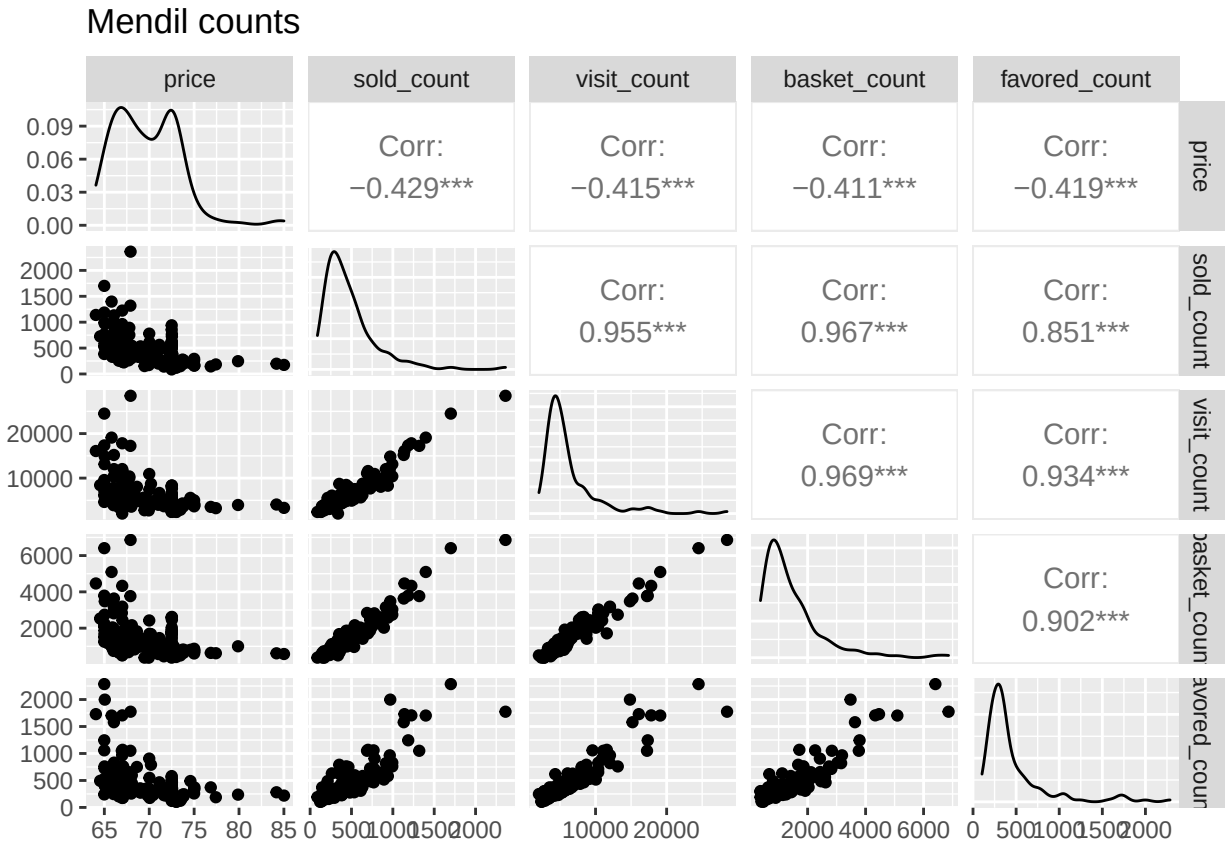

```
ggpairs(yuz_temizleyicisi, columns = c(3,4,5,6,7),title = "Yuz_temizleyicisi counts")
```



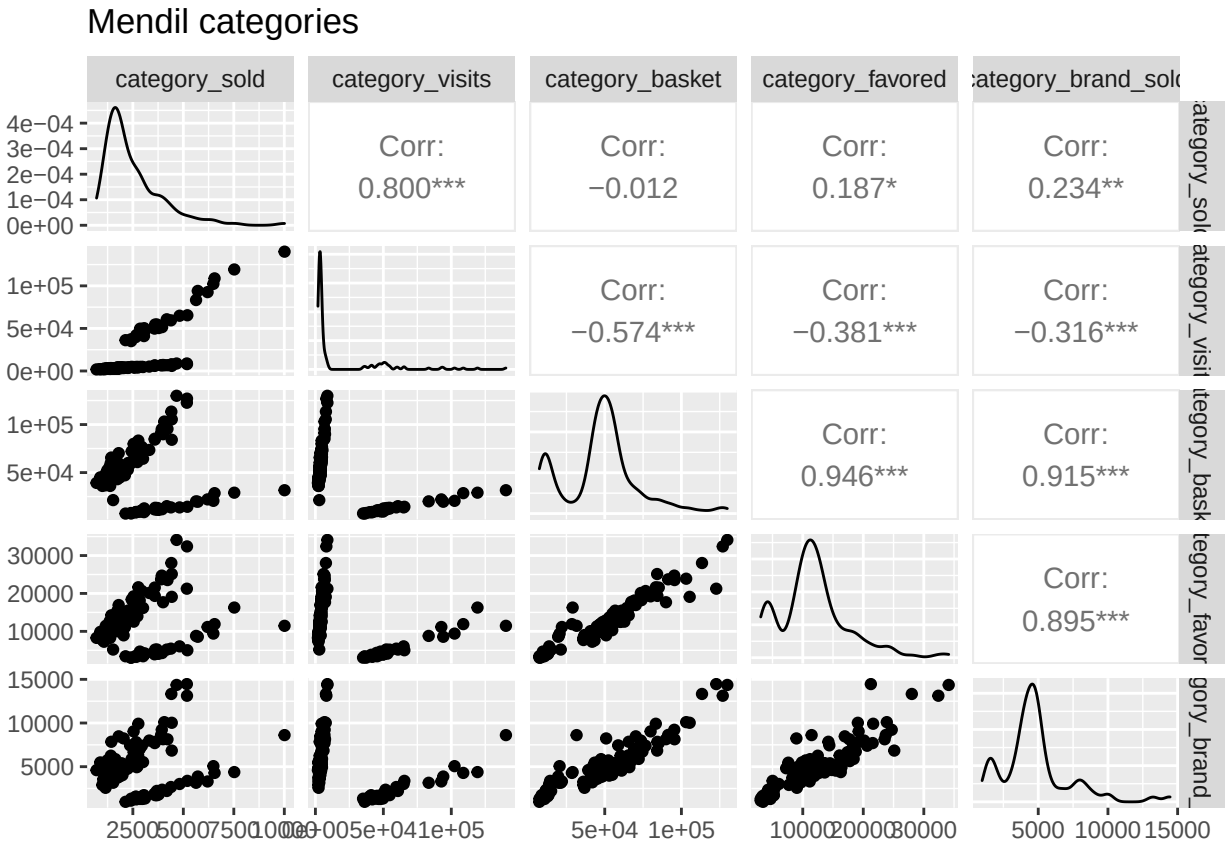

```
ggpairs(yuz_temizleyicisi, columns = c(8,9,10,11,12),title ="Yuz temizleyicisi categories")
```



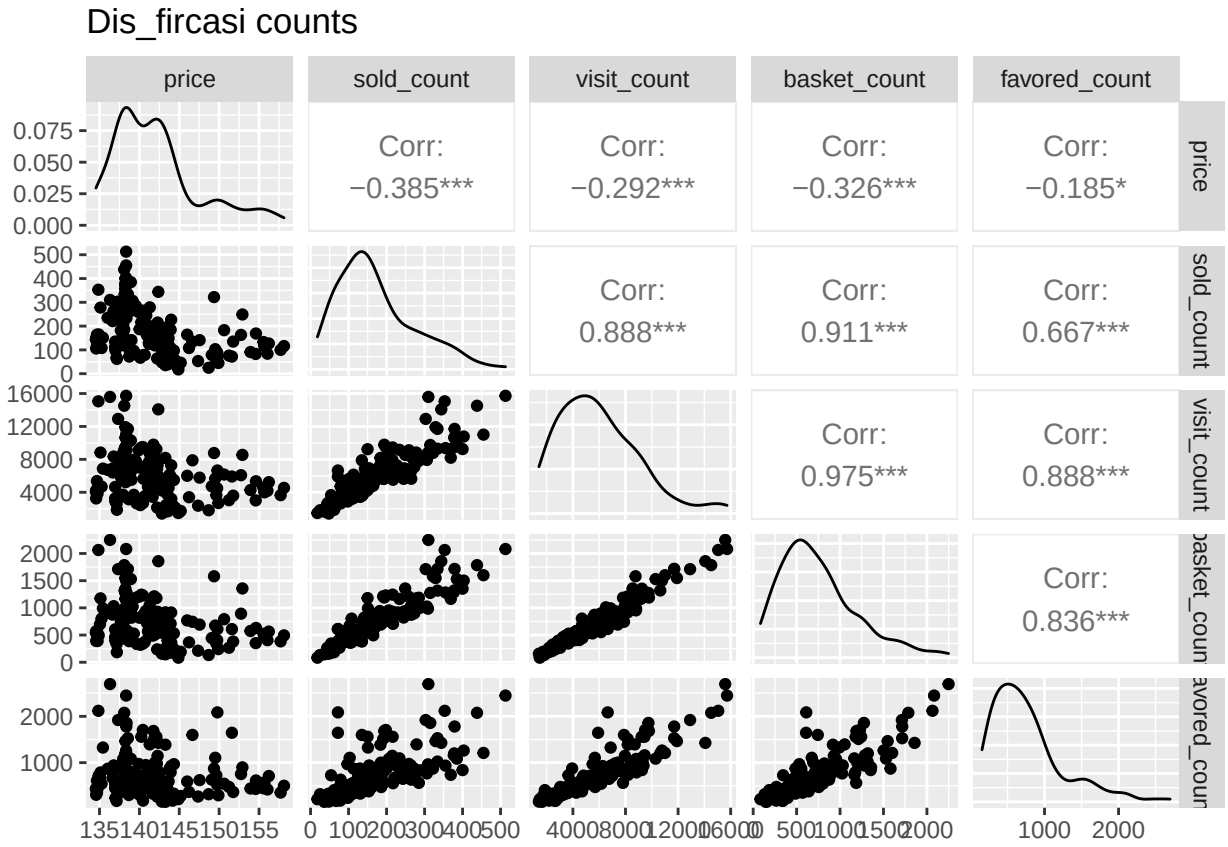

```
ggpairs(mendil, columns = c(3,4,5,6,7),title = "Mendil counts")
```



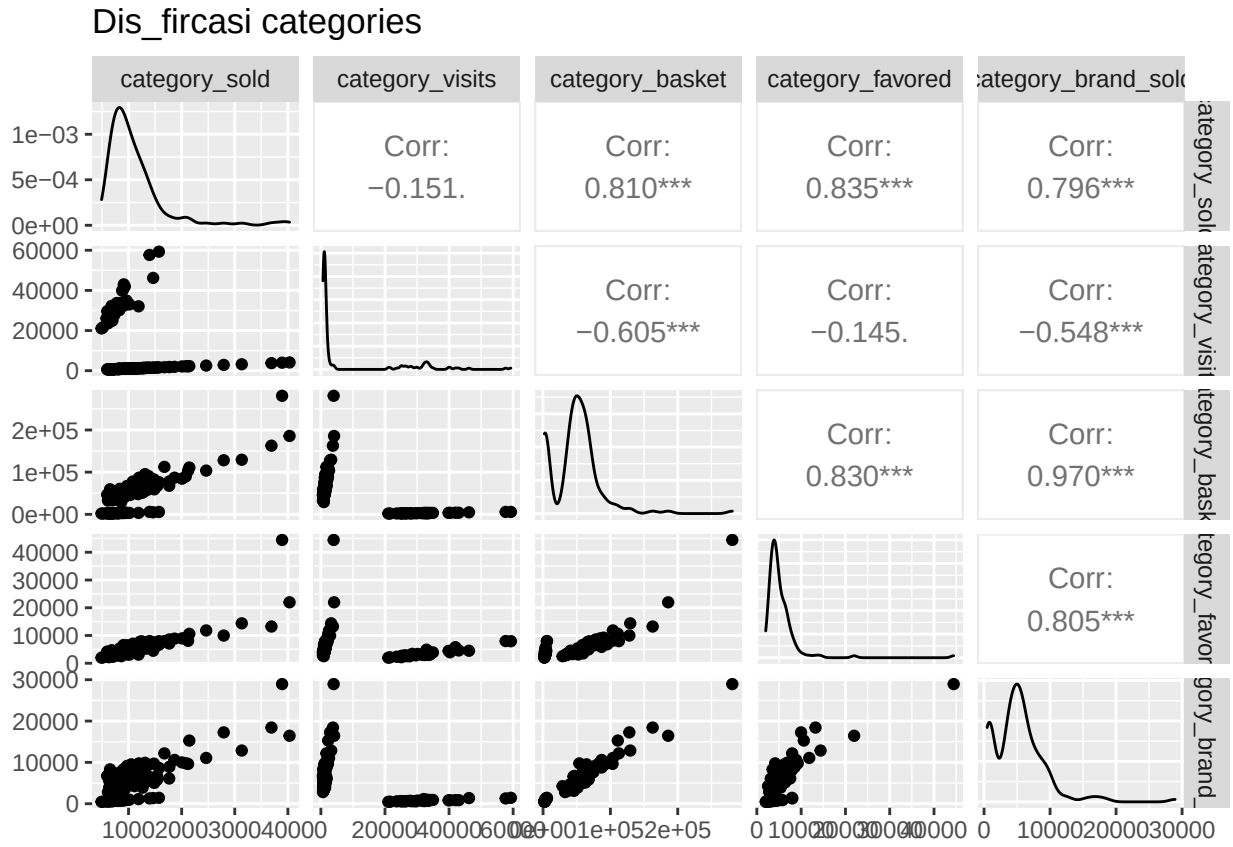

```
ggpairs(mendil, columns = c(8,9,10,11,12),title ="Mendil categories")
```




```
ggpairs(dis_fircasi, columns = c(3,4,5,6,7),title = "Dis_fircasi counts")
```




```
ggpairs(dis_fircasi, columns = c(8,9,10,11,12),title ="Dis_fircasi categories")
```



We see that price has a dominant negative correlation with sold_count in general and basket_count and visit_count regressors are most dominant positive correlated ones.

Related Literature

The linear model and arima model are the ones we used to predict next day's sales quantity but other approaches could be used. The main sources of our approach comes from IE_360 lectures and lecture videos in moodle system. No other external source method is used. One may also could use GAM method which is generialized additive models. Rather modelling linearly and discretely , we can model and fit the data by introducing contuinity in the data. SARIMAX or ARIMAX models would also perform efficiently. Our ARIMA model does not contain regressor or seasonality but we add these parts in the linear model so that arima deals only with the residuals. Our method is not the perfect one but when we search what is to be the best method , we encountered with the long short-term memory method.Long short-term memory (LSTM) is an artificial recurrent neural network (RNN) architecture used in the field of deep learning. Unlike standard feedforward neural networks, LSTM has feedback connections. It can not only process single data points (such as images), but also entire sequences of data (such as speech or video). Here is the wikipedia explanation and since it includes stuff that we have not learned in the class , we did not use this method. Xgboost usage seemed us to be a quick tricky way to extract the necessary features , hence we continued with our common sense and we discussed and found new regressors as we will see.

Approach

Our model we used in the API competition consists of 2 fundamental parts: Linear Model and ARIMA on residuals. We used the pipeline approach to code efficiently and to analyse easily. Our pipeline consists of 4 main functions : preperation, point forecast, performance and estimate. We will open up each of them as we move on.

Model Preperation Function

Here we introduce our preperation function which handles data row reversing, trend component addition ,traffic regressor addition and difference regressor addition. The function takes non-processed data and outputs the processed version of it.

```
preperation <- function(X){
  input_data = X
  len = length(input_data$price)
  input_data <- input_data %>% map_df(rev)
  trend = seq(1:len)
  input_data<-cbind(input_data,trend)
  input_data <- input_data %>% mutate(traffic =
                                        case_when(ty_visits >= mean(quantile(input_data$ty_visits)[3],q
                                                  ty_visits < mean(quantile(input_data$ty_visits)[3],qua
                                        )
  )

  difference <-diff(input_data$sold_count)
  difference[len]=0
  input_data<-cbind(input_data,difference)

  z<-input_data
  return (z)

}
```

Here there is a wider explanatory definitions :

data-row-reversing : before entering preperation function data rows are ordered by decreasing event date but we wanted to have latest data to be at the bottom.It is just a practice choice.

trend-component-addition : Since we saw that some of the products'sales quantity contains a trend in their acf graph, we decided to add a numeric trend colum.

New regressor detected ! : traffic traffic-regressor-addition : Thanks to the Trendyol, we also had total visit counts of the website. When the daily website visit counts lays in between 3rd and 4th quantiles of the data , we label those days as crowded days; therefore we named it as traffic. #### New regressor detected ! : difference difference-regressor-addition : This regressor represents the consecutive daily rise or fall in the sales quantity. It is deducted from the lag 1 observation and we will use it in the model.

Point Forecast Function

Since we will predict next day's sales quantity , we need to handle next day's regressors' values :

```
point_forecast <- function(X) {

  input_data = X
  len = length(input_data$price)

  new_price = forecast(input_data$price,1)$mean[1]
  new_visit_count = round(forecast(input_data$visit_count,1)$mean[1])
  new_basket_count = round(forecast(input_data$basket_count,1)$mean[1])
  new_favored_count = round(forecast(input_data$favored_count,1)$mean[1])
  new_cat_sold = round(forecast(input_data$category_sold,1)$mean[1])
  new_cat_visits = round(forecast(input_data$category_visits,1)$mean[1])
  new_cat_basket = round(forecast(input_data$category_basket,1)$mean[1])
  new_cat_favored = round(forecast(input_data$category_favored,1)$mean[1])
  new_cat_brand = round(forecast(input_data$category_brand_sold,1)$mean[1])
  new_ty = round(forecast(input_data$ty_visits,1)$mean[1])
  new_soldout = round(forecast(input_data$soldout,1)$mean[1])
  new_lagg = round(forecast(input_data$lagg,1)$mean[1])
  new_traffic = round(forecast(input_data$traffic,1)$mean[1])

  newdate = input_data$event_date[len]+1
  newwday = weekdays(newdate)
  newmon = months(newdate)

  newrow <- data.frame(event_date=newdate,stringsAsFactors=F)
  input_data <- rbind.fill(input_data,newrow)



  input_data$product_content_id[len+1] <-  input_data$product_content_id[2]
  input_data$wday[len+1] <- newwday
  input_data$mon[len+1] <- newmon
  input_data$price[len+1] <- new_price
  input_data$visit_count[len+1] <- new_visit_count
  input_data$basket_count[len+1] <- new_basket_count
  input_data$favored_count[len+1] <- new_favored_count
  input_data$category_sold[len+1] <- new_cat_sold
  input_data$category_visits[len+1] <- new_cat_visits
  input_data$category_favored[len+1] <- new_cat_favored
  input_data$category_basket[len+1] <- new_cat_basket
  input_data$category_brand_sold[len+1] <- new_cat_brand
  input_data$ty_visits[len+1] <- new_ty
  input_data$soldout[len+1] <- new_soldout
  input_data$lagg[len+1] <- new_lagg
  input_data$trend[len+1] <- len+1
  input_data$traffic[len+1] <- new_traffic


  z <- input_data
  return (input_data)
}
```

We used this function but it was essentially necessarily not so useful , since the point forecasting of regressors were adding up the error and error accumulation was inevitable.

Model Performance Function

This function is the core of the all processes. Here we apply linear model and after extracting the residuals from LM we use them in the ARIMA model.

```
performance <- function(X
){
  input_data =X

  if(deparse(substitute(X))=="mont"){
    fit <- lm(sold_count ~ price   +soldout+(sold_count/visit_count) , data = input_data)

  }
  else{

    if (sum(input_data$soldout)==0) {
      fit <- lm(sold_count ~ price   +input_data$difference+ lagg +(visit_count/category_visits)+wday+m
    } else {
      fit <- lm(sold_count ~ price   +input_data$difference+soldout+ lagg +(visit_count/category_visits
    }
  }



  summary(fit)
  checkresiduals(fit)

  model=auto.arima(residuals(fit),seasonal=F,
                   trace=T,stepwise=F,approximation=F)

  model_fitted = residuals(fit) - residuals(model)
  predicted=floor(predict(fit) + model_fitted)



  error=input_data$sold_count-predicted

  mean=mean(input_data$sold_count)
  MAD=sum(abs(error))/length(input_data$sold_count)
  WMAPE=MAD/mean
  my_list <- list("data" = input_data, "lm" = fit, "arima" = model,"wmape"=WMAPE)
  return(my_list)

}
```

Linear Model Part

Product Categories from our perspective We shall explain this function step by step since its most important one.We divided products into 3 categories : one being the mont itself, one being having no soldouts,i.e. the product is always sold given the time interval , one being having at least some soldouts hence the long zero sale-days happening. This was obtained from general investigation of the graphs above sections. Mont is treated as a special product since it only responded price,soldout and (sold_count/visit_count) regressors during testing.

New regressor detected ! : conversion Rate (Sold_count/visit_count) Sold_count/visit_count is also newly introduced regressor which represents the "conversion rate" as in the Key Performance Indicators(KPI) jargon.

New regressors detected ! : category_visit_conversion_rate , category_basket_conversion_rate , category_sold_conversion_rate ,category_favored_conversion_rate Here 4 similar regressors added to the linear model as well. We decided to represent the conversion rates of visits,baskets, sold_counts and favoring the item scaling them to their respective categories . By doing that , we were able to compare the given product to other items in the same category in terms of visits,basket numbers, favoring numbers and sold-counts.

Abbreviations :

category_visit_conversion_rate : (visit_count/category_visits), category_basket_conversion_rate : (basket_count/category_basket), category_sold_conversion_rate : (sold_count/(category_sold), category_favored_conversion_rate : (favored_count/category_favored)

All in all , the regressors being used are in general : price , lag(1),difference,traffic, wday(weekday),mon(month), trend, soldout, conversion_rate, category_visit_conversion_rate , category_basket_conversion_rate , category_sold_conversion_rate ,category_favored_conversion_rate

ARIMA Part

After fitting a linear model , now we extract the residuals of the model and process them in an ARIMA model. Since we will update our data in daily basis, the arima model will change with the new addition of data , the effect of change will not too much but since we have a limited data , we can use auto.arima to handle the new model parameters. As we investigated earlier , p parameter would range from 1-2 and q parameteter from 1-4, but the auto.arima will say something else and we will focus on this situation in a minute.

Evaluation / WMAPE

The model evaluation metric is chosen to be WMAPE(wieghted mean average percentage error). It is basically calculated as follows : we substract the actual value from predicted on to get the error. MAD(mean absolute deviation) is also calculated by taking mean of all daily error summation.At the end ,WMAPE equals to MAD divided by the mean of actual values.

The function takes each product as an input and gives a list of outputs.The list returns as an output containg : processed input , linear model itself, arima model itself and WMAPE result.

Estimate Function

Now its time to predict the next day's sales quantity.

```
estimate <- function(Y,linear,ariba
){
  input_data=Y
  len=length(input_data$price)
  fit = linear
  model = ariba
  model_fitted = residuals(fit) - residuals(model)
  future = predict(model, n.ahead =1)$pred
  res = mean(tail(residuals(fit),3))
```

```
  res = rep(res,1)
  future_fitted = res - future
  last_day=mean(predict(fit)[(length(input_data$sold_count)-1):(length(input_data$sold_count)-3)])
  estimated=abs(round(last_day +future_fitted))
  return (estimated)
}
```

The estimation is done as follows : we predict arima residual for 1 period of time and take the mean of last 3 days' residuals of linear model and substract them, so we obtain the predicted residuals. We also take the mean of last 3 days' predictions and add it to the predicted residuals. Here we see that we dont actually use point forecasting of regressors since the error accumulation was a big issue, here we use the last 3 predicted values from linear model and this will give us better results.

Overall Pipeline

Overall pipeline consists of all 4 functions and it also prints WMAPE and the predicted value for the next day's sales quantity.

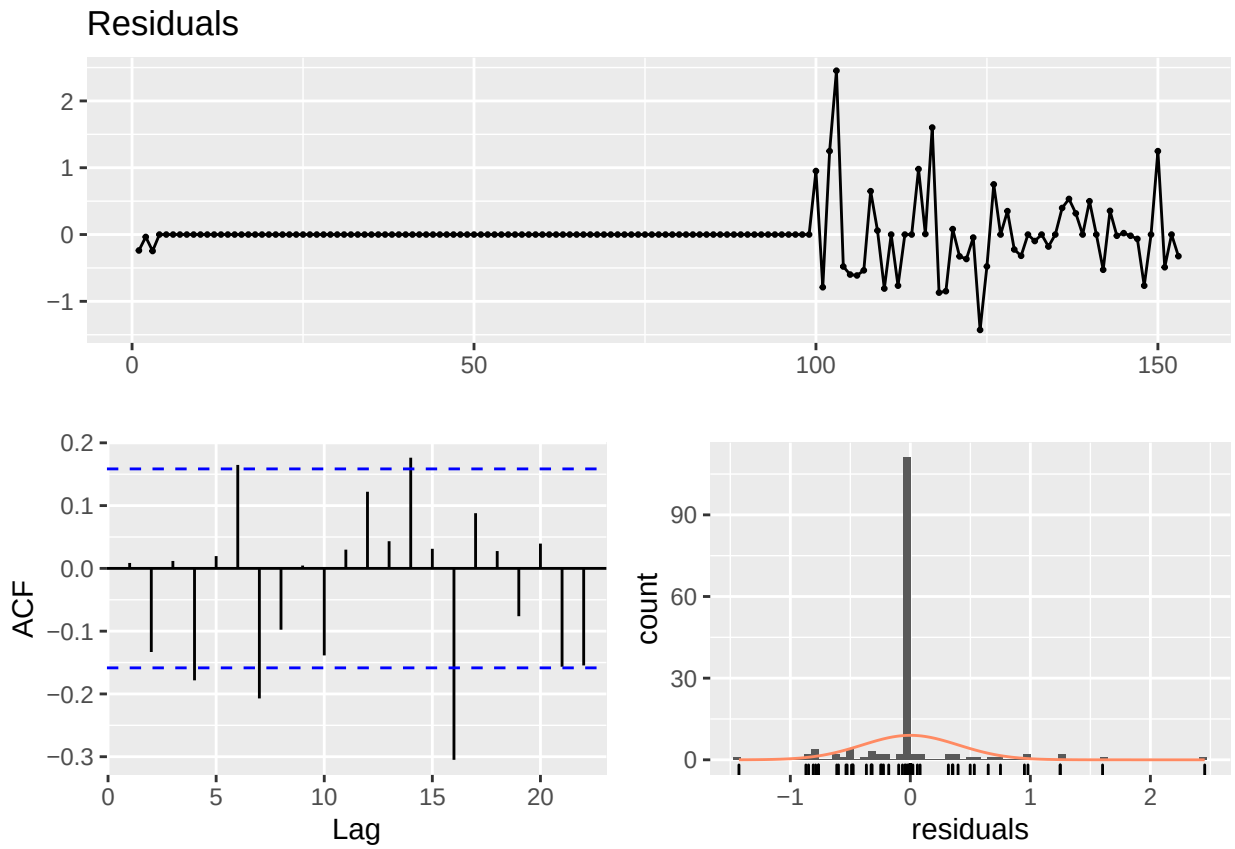
```
pipeout <-function(X){
  X_prep=preperation(X)
  X_perf=performance(X_prep)
  error = X_perf$wmape
  print(error)

  X_new<-point_forecast(X_prep)
  B=estimate(X_new,X_perf$lm,X_perf$arima)
  print(B[1])
}
```

Results

Here we will demonstrate 1 example pipeout for each 3 categories(mont,no soldouts,yes soldouts) that we mentioned.

Category 1 : Mont

```
mont <- preperation(mont)
A=performance(mont)
```




```
##
##  ARIMA(0,0,0) with zero mean     : 154.259
##  ARIMA(0,0,0) with non-zero mean : 156.3126
##  ARIMA(0,0,1) with zero mean     : 156.2969
##  ARIMA(0,0,1) with non-zero mean : 158.378
##  ARIMA(0,0,2) with zero mean     : 154.2706
##  ARIMA(0,0,2) with non-zero mean : 156.3792
##  ARIMA(0,0,3) with zero mean     : 156.3791
##  ARIMA(0,0,3) with non-zero mean : 158.5165
##  ARIMA(0,0,4) with zero mean     : 153.1455
##  ARIMA(0,0,4) with non-zero mean : 155.3047
##  ARIMA(0,0,5) with zero mean     : 155.245
##  ARIMA(0,0,5) with non-zero mean : 157.4326
##  ARIMA(1,0,0) with zero mean     : 156.3011
##  ARIMA(1,0,0) with non-zero mean : 158.3822
##  ARIMA(1,0,1) with zero mean     : 156.5724
##  ARIMA(1,0,1) with non-zero mean : 158.6814
##  ARIMA(1,0,2) with zero mean     : 151.5028
##  ARIMA(1,0,2) with non-zero mean : Inf
##  ARIMA(1,0,3) with zero mean     : Inf
##  ARIMA(1,0,3) with non-zero mean : Inf
##  ARIMA(1,0,4) with zero mean     : 155.2868
##  ARIMA(1,0,4) with non-zero mean : 157.4753
##  ARIMA(2,0,0) with zero mean     : 155.6576
##  ARIMA(2,0,0) with non-zero mean : 157.7666
##  ARIMA(2,0,1) with zero mean     : 156.5804
```

```
##  ARIMA(2,0,1) with non-zero mean : 158.7182
##  ARIMA(2,0,2) with zero mean     : 151.8068
##  ARIMA(2,0,2) with non-zero mean : Inf
##  ARIMA(2,0,3) with zero mean     : Inf
##  ARIMA(2,0,3) with non-zero mean : Inf
##  ARIMA(3,0,0) with zero mean     : 157.7344
##  ARIMA(3,0,0) with non-zero mean : 159.8721
##  ARIMA(3,0,1) with zero mean     : 158.7088
##  ARIMA(3,0,1) with non-zero mean : 160.876
##  ARIMA(3,0,2) with zero mean     : Inf
##  ARIMA(3,0,2) with non-zero mean : Inf
##  ARIMA(4,0,0) with zero mean     : 153.0572
##  ARIMA(4,0,0) with non-zero mean : 155.2242
##  ARIMA(4,0,1) with zero mean     : 155.1074
##  ARIMA(4,0,1) with non-zero mean : 157.3043
##  ARIMA(5,0,0) with zero mean     : 154.9516
##  ARIMA(5,0,0) with non-zero mean : 157.1485
##
##
##
##  Best model: ARIMA(1,0,2) with zero mean
```

```
print(summary(A$lm))
```

```
##
## Call:
## lm(formula = sold_count ~ price + soldout + (sold_count/visit_count),
##     data = input_data)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -1.428  0.000  0.000  0.000  2.453
##
## Coefficients:
##                           Estimate Std. Error t value Pr(>|t|)
## (Intercept)              2.0048345  0.1907470  10.510  < 2e-16 ***
## price                   -0.0011330  0.0004145  -2.733  0.00703 **
## soldout                 -2.0048345  0.1946898 -10.298  < 2e-16 ***
## sold_count:visit_count   0.0019713  0.0001048  18.807  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.4032 on 149 degrees of freedom
## Multiple R-squared:  0.9138, Adjusted R-squared:  0.9121
## F-statistic: 526.5 on 3 and 149 DF,  p-value: < 2.2e-16
```

```
error = A$wmape
mont<-point_forecast(mont)
B=estimate(mont,A$lm,A$arima)
print(error)
```

```
## [1] 0.3157895
```

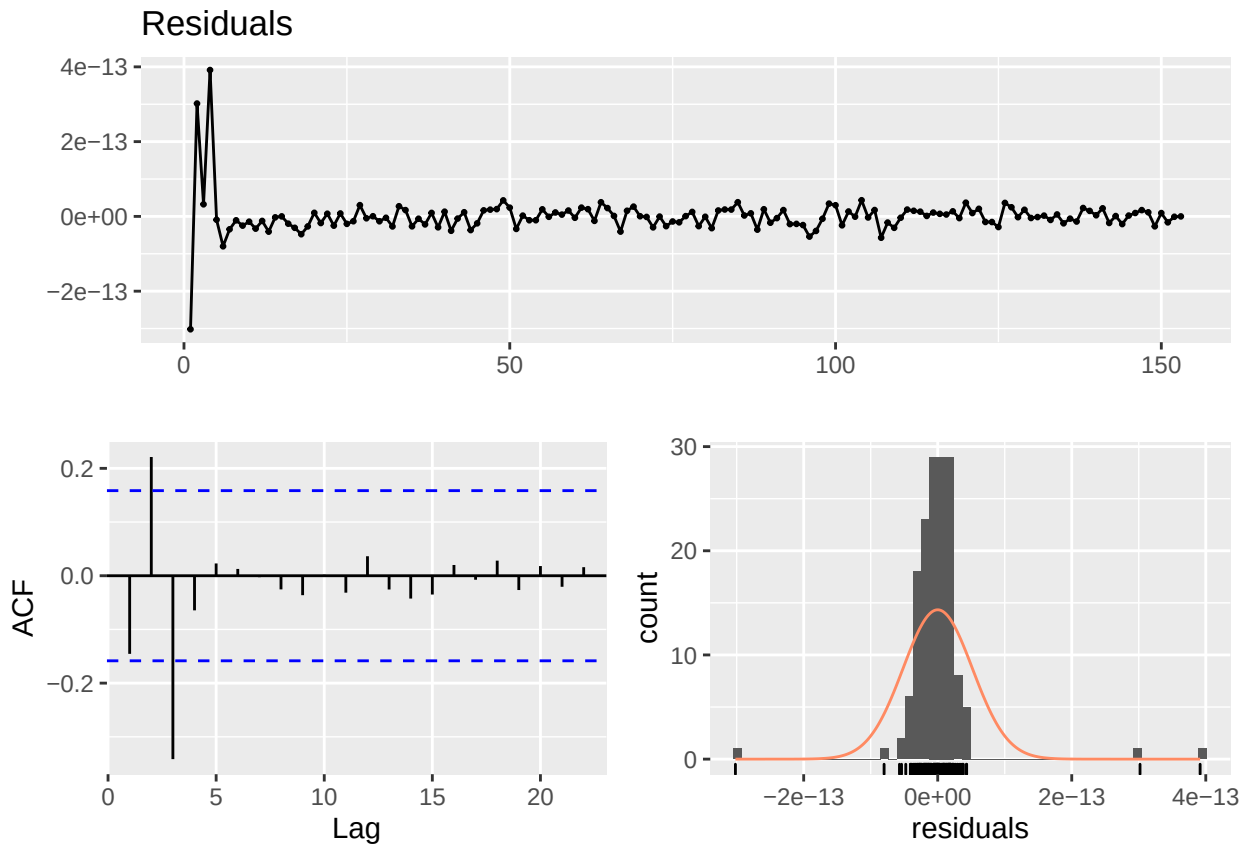
```
print(B[1])
```

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

We see that R-adjusted is 0.91 which is satisfactory. The WMAPE gave us 31 percent error and it predicted the next day's value as 1,i.e. 1 mont selling is predicted for the next day. The zero mean and constant variance error assumption holds, acf seems to be almost desired meaning that the lags' peaks dont cross the confidence interval. The gaussian distiribuiton error assumption seems to be vialoted. This may occur due to having too much zero-selling days.

Category 2: No Soldout(not including the 'soldout' regressor)

```
pipeout(dis_fircasi)
```

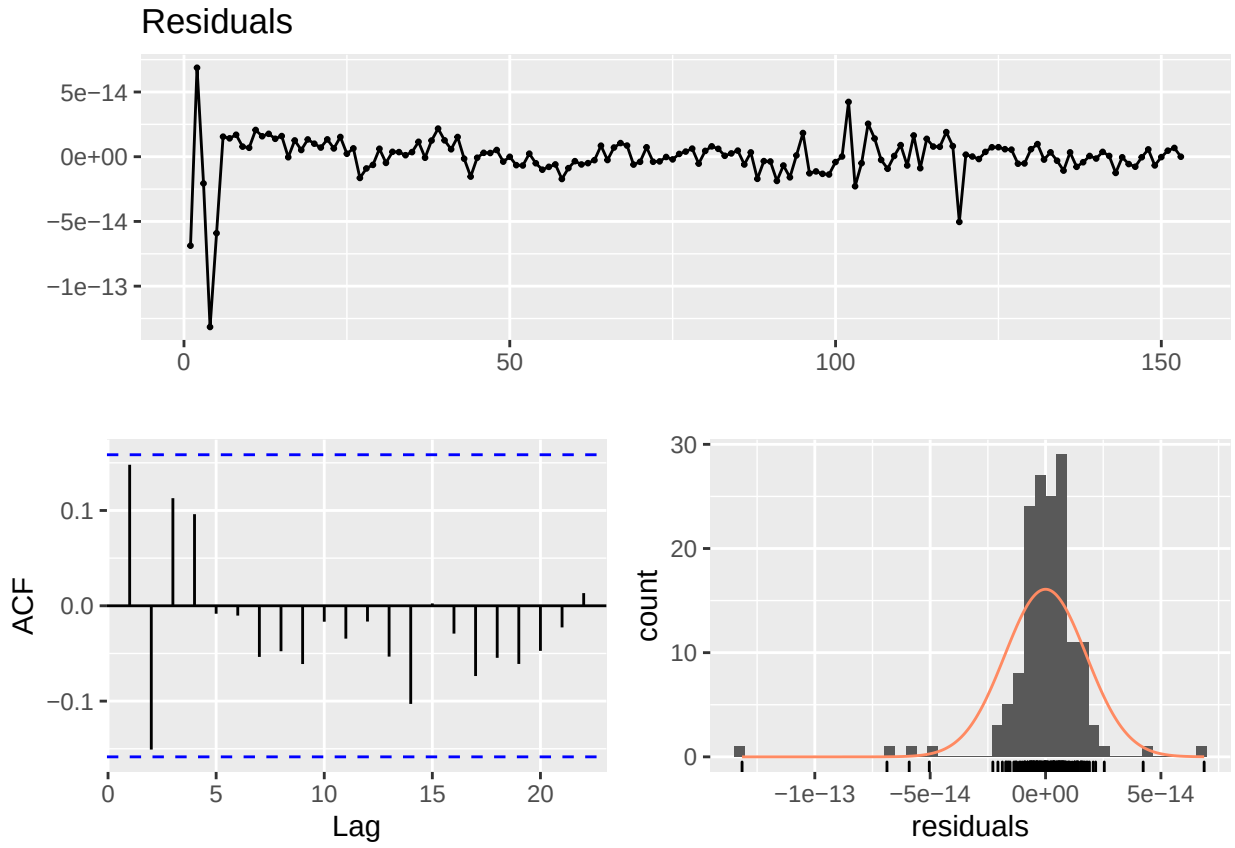



```
## [1] 0.003743902
## [1] 133
```

Overfitting issues Here we see that the prediction is 133 for the next day and WMAPE gave us 0.3 percent error meaning that we may ended up overfitting the model. Residuals assumptions are valid and gaussian distribution assumption almost holds , the acf seems to okay in terms of fitting in between dashed confidence interval lines. The overfitting problem comes from having a linear model with R-squared being 1.The reasons getting R-adjusted to be 1 may arises from applying too much regressors or the domination of 1 specific regressor in the model.

Category 3 : Yes soldout(including the 'soldout' regressor)

```
pipeout(bikini_1)
```




```
## [1] 0.007291385
## [1] 18
```

Overfitting Issues

Here we see that the prediction is 18 for the next day and WMAPE gave us 0.7 percent error meaning that we may ended up overfitting the model. Residuals assumptions are valid and gaussian distribution assumption almost holds , the acf seems to okay in terms of fitting in between dashed confidence interval lines. The overfitting problem comes from having a linear model with R-squared being 1.The reasons getting R-adjusted to be 1 may arises from applying too much regressors or the domination of 1 specific regressor in the model. We see that for the other 8 product the overfitting is a big issue. Overfitting gives us almost perfect models but the prediction becomes unreliable at some points.

Conclusion and Future Work

We succesfully identified each product's statistical characteristics and their relationships/correlations between other potential regressors. We created some new additional regressors using common sense and considering KPI approaches. The model we used was Linear Model + ARIMA . Linear model was overfitting the

data except for the 'mont' product. Our API competition performance was initially great , we were in top 3 for nearly first 10 days of the competition and in the last 6 days we were dreadful , this may occured from not updating the performance function to avoid the overfitting , the non-ideal approach of estimation may be another reason. Trusting the first 10 or-so days may boosted our confidence too much and our attention to improve the model decreased.One can also use the GAM approach or LSTM method to get highly accurate results if this individual makes quite dramatic decisions in finance,energy industry etc. Advanced method would give this indiviual much more reliable predictions. All in all, the competition was quite fun , the first thing that we've done in the morning was to predict the future during those 15-day period ! Here the overall competition results :

Competition	Group8	Group1	Group6	Group3	Group5	Group2	Group14	Group11	Group10	Group9	Group15	Group12	Group13	Group16	Group7	Group4
Day-12-June	18	16	10	14	15	19	17	7	9	13		8	11		12	
Day-13-June	17	19		11	15	12	14	16	7	9		10	8	13	6	18
Day-14-June	14	17	16	15	18	12	13	19	10	8		9	7	11	6	
Day-15-June	13	19	16	17	11	14	7	18	8	10		6	9	15	5	12
Day-16-June	18	14	19	17	13	15	11	16	7	9	12	10			6	8
Day-17-June	13	17	18		10	16	12	15	14	11	19	7	8	9	6	
Day-18-June	11		18	19	9	16	13	16	12	7	17	14	10	6	5	8
Day-19-June	16	9	7	14	12	15	17	6	11	8	10	18	19	13	4	5
Day-20-June	18	12	13	7	10	17	19	11	9	14	16	8	15			
Day-21-June	11	19	18	8	16	9	12	13	10	14	17	15		6	7	
Day-22-June	19	15	18	12	10	5	8	13	17	11	14	9	6	16	7	
Day-23-June	18	19	17	11	10	9	7	6	16	14	15	13	12		8	
Day-24-June	19	13	11	15	18	8	7	5	12	9	16	14	10	17	6	
Day-25-June	19	15	12	18	14	11	8	5	16	17	10	9	7	13	6	
Day-26-June	17	16	18	19	12	13	11	7	14	9	15	10	8		6	
Grand Total	**241**	**220**	**211**	**197**	**193**	**191**	**176**	**173**	**172**	**163**	**161**	**160**	**130**	**119**	**90**	**51**

Figure 1: the competition results.As Group2 we ended up in the 6th place

Code

Here is the R-script code