

Projet Covid

Margaux Bailleul / Oriane Duclos / Marie Guibert

2023-05-07

```
library(tidyverse)
library(forecast)
library(tidyquant)
```

```
## Warning: le package 'tidyquant' a été compilé avec la version R 4.2.3
```

```
## Warning: le package 'PerformanceAnalytics' a été compilé avec la version R 4.2.3
```

```
library(caschnono)
```

```
## Warning: le package 'caschnono' a été compilé avec la version R 4.2.3
```

```
library(tseries)
```

Importations des données

Tout d'abord, on importe les données et on sélectionne les données concernant la France.

```
donnees_fr <- read.csv("covid_france.csv", sep=",")
summary(donnees_fr)
```

```
##      date      new_cases
## Length:1203    Min.   :    0
## Class :character 1st Qu.: 2968
## Mode  :character Median :12174
##                Mean   :32315
##                3rd Qu.:32913
##                Max.   :500563
##                NA's   :1
```

Les données ci-dessus comprennent une variable temporelle et une variable caractérisée par un enregistrement journalier des nouveaux cas de Covid-19 en France.

```
min(donnees_fr$date)
```

```
## [1] "2020-01-03"
```

```
max(donnees_fr$date)
```

```
## [1] "2023-04-19"
```

Grâce à cette étape, nous pouvons observer que notre série temporelle débute le 1er Mars 2020 et se termine le 19 Avril 2023. Notre étude a donc une plage d'environ de 3 ans.

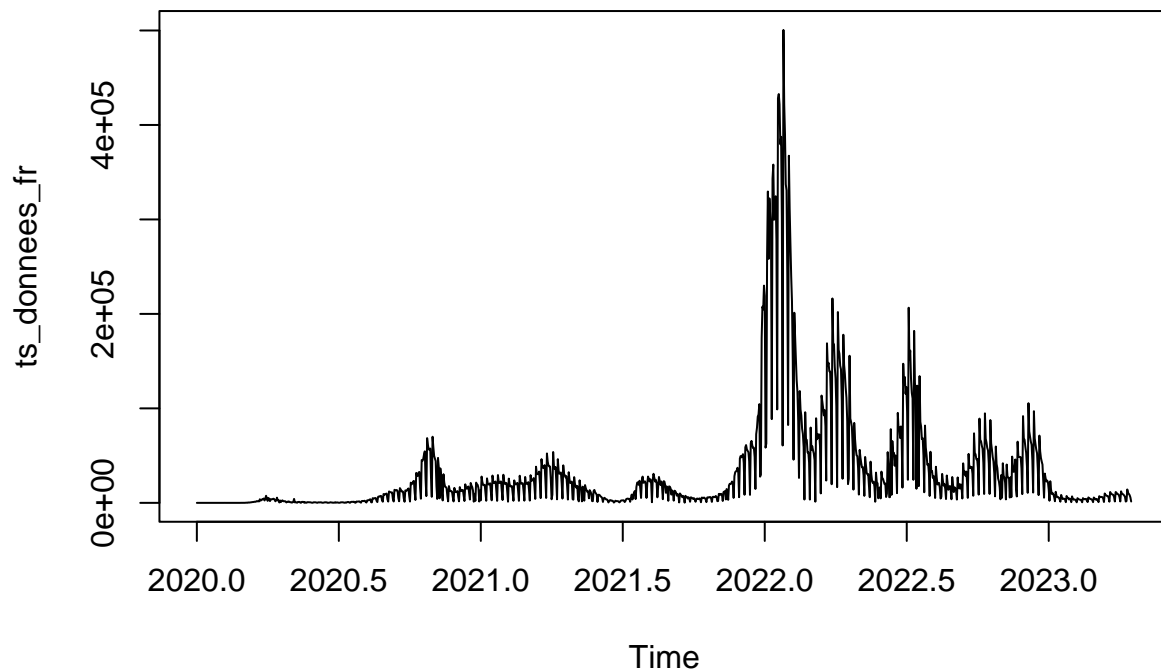
Transformation des données en série temporelle

Premièrement, nous allons transformer nos données en séries temporelles pour pouvoir réaliser notre analyse.

```
ts_donnees_fr <- ts(donnees_fr$new_cases, start = c(2020,1,3), frequency = 365)
# class(ts_donnees_fr)
```

Première partie

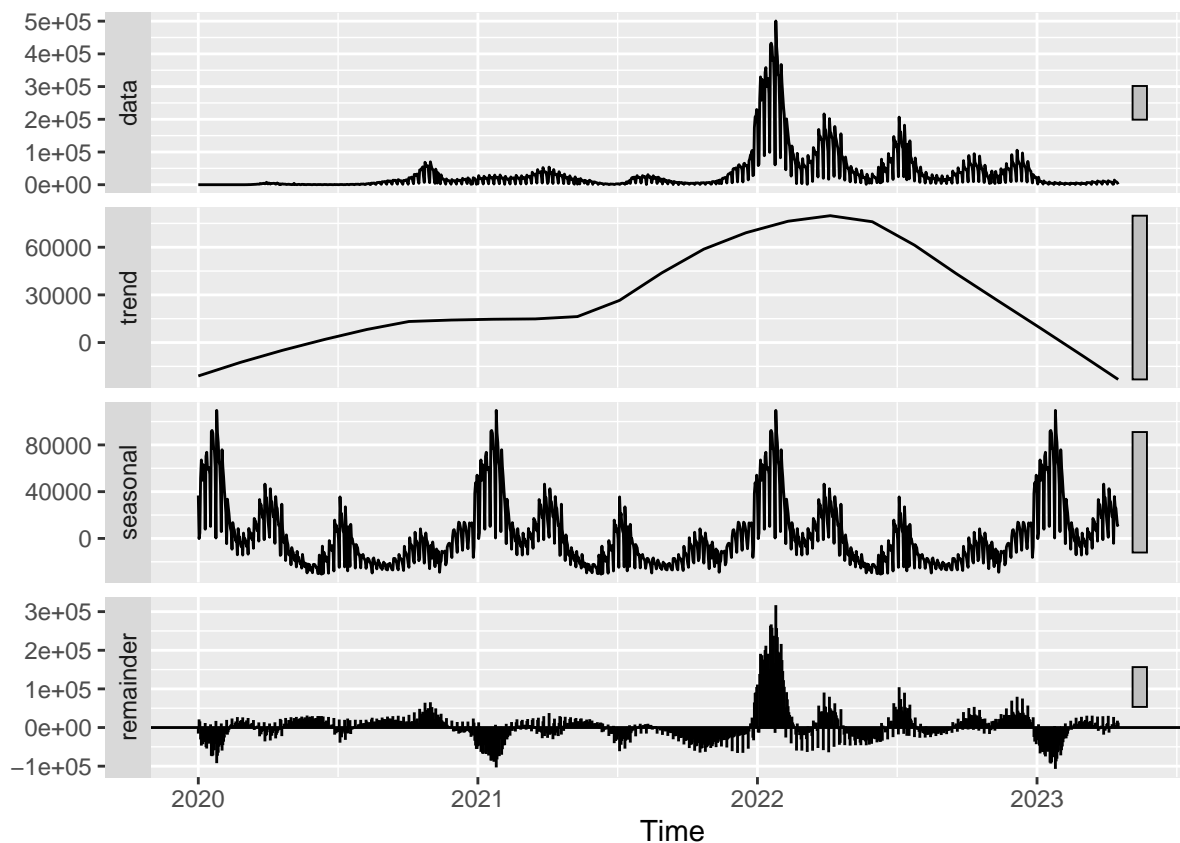
```
plot(ts_donnees_fr)
```



Ce premier graphique nous montre une hausse brutale des nouveaux cas de covid en 2022. Afin de pouvoir continuer notre analyse de façon cohérente, nous allons diviser notre série en 3 parties : avant, pendant et après ce choc en 2022.

Application de la décomposition saisonnière à la série temporelle pour visualiser les tendances et les motifs saisonniers. Nous supprimons les données manquantes afin de ne garder que celles qui sont pertinentes.

```
decomp_ts <- stl(na.omit(ts_donnees_fr), s.window = "periodic")
autoplot(decomp_ts)
```



Cette étape nous permet d'observer une tendance à la hausse entre 2020 et 2022, puis à partir de 2022, une tendance à la baisse.

De plus, nous pouvons voir une saisonnalité annuelle.

Division de notre série

Nous décidons de créer trois sous-séries de notre série initiale afin de pouvoir réaliser le traitement des données. Notre objectif est d'isoler le cas particulier de l'année 2022 pour avoir une étude correcte.

```
serie1 <- donnees_fr |>
  filter(date<="2021-12-22")
# serie1
ts_serier1 <- ts(serie1$new_cases, start = c(2020,1,3), frequency = 365)

serie2 <- donnees_fr |>
  filter(date>"2021-12-22", date<="2023-01-05")
# serie2
```

```
ts_serie2 <- ts(serie2$new_cases,start = c(2021,31,12), frequency = 365)
```

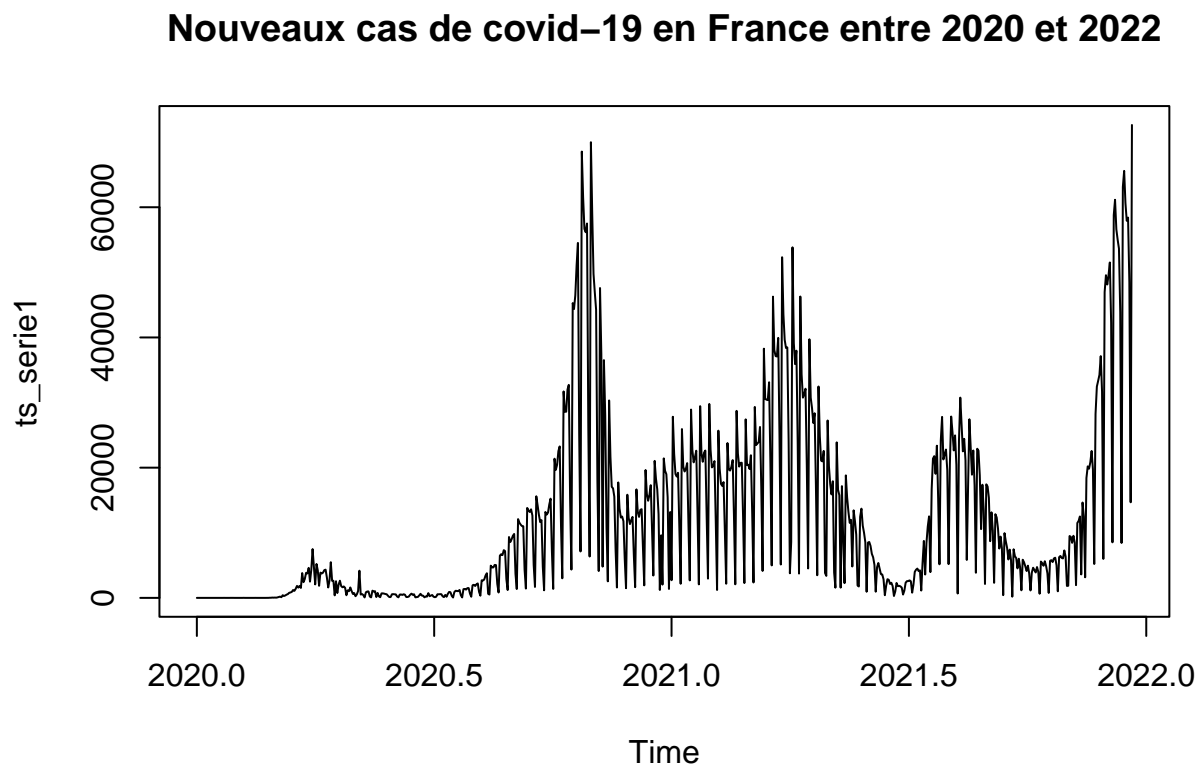
```
serie3 <- donnees_fr |>  
  filter(date>"2023-01-05")
```

```
ts_serie3 <- ts(serie3$new_cases,start = c(2023,2,5), frequency = 365)
```

Nous avons choisi de scinder notre série en trois périodes : - avant le 22 Décembre 2021 - entre le 23 Décembre 2021 et le 5 Janvier 2023 - après le 6 Janvier 2023

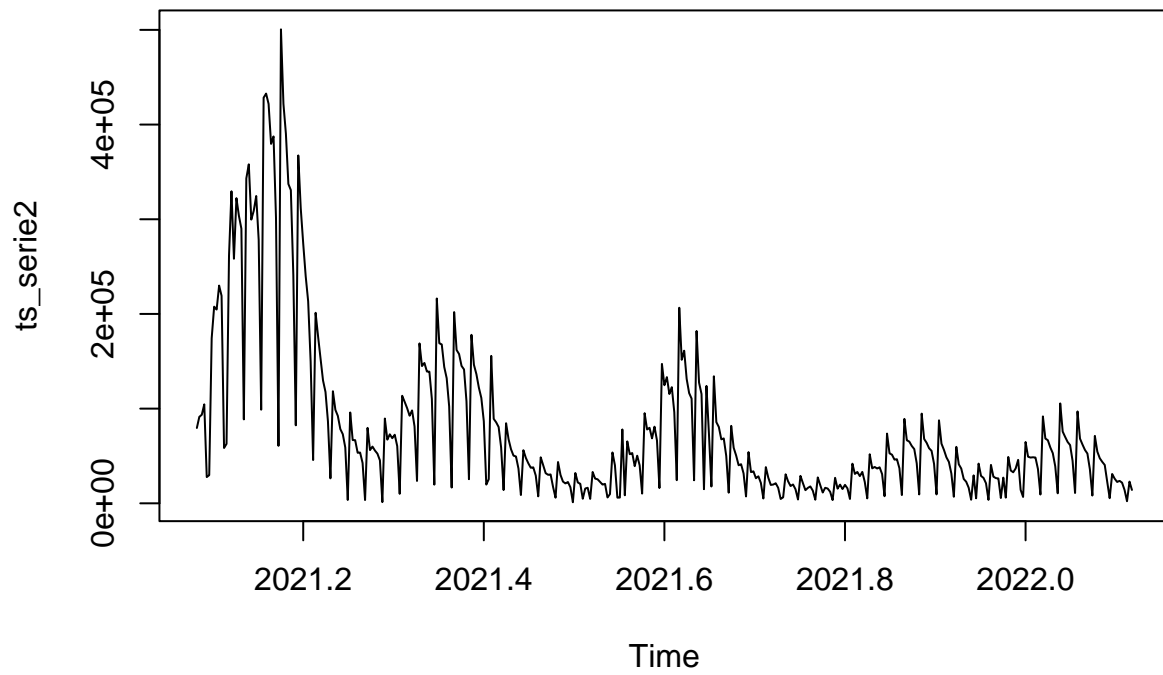
Nous pouvons maintenant les visualiser :

```
plot(ts_serie1,main="Nouveaux cas de covid-19 en France entre 2020 et 2022")
```



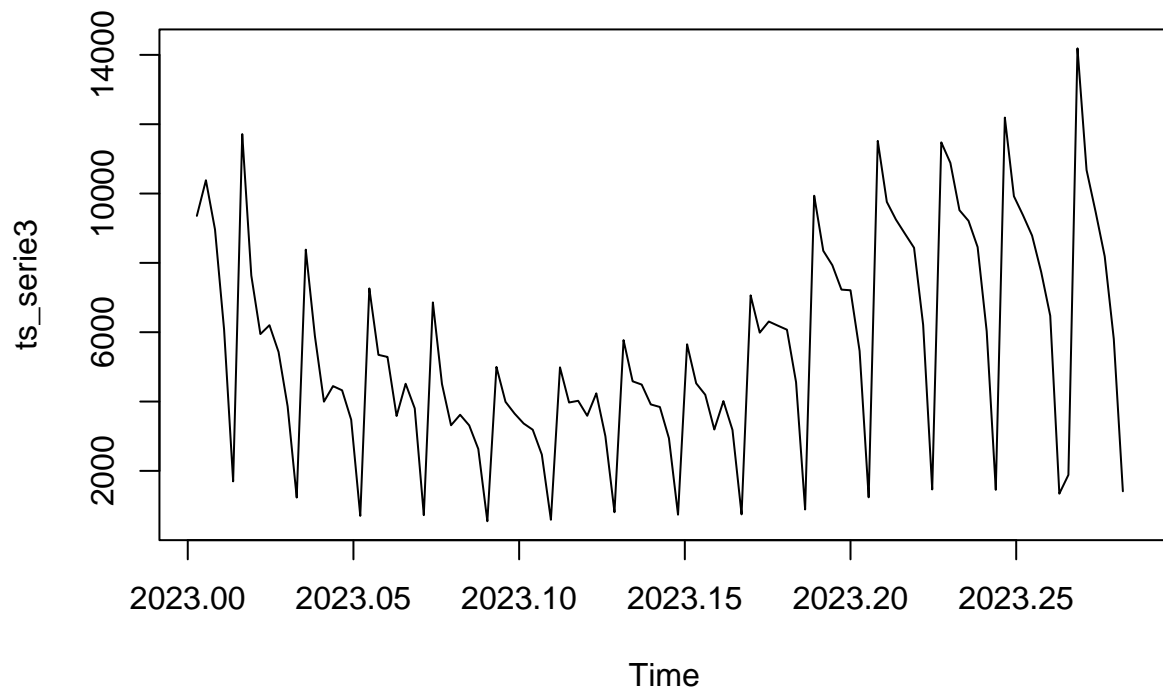
```
plot(ts_serie2,main="Nouveaux cas de covid-19 en France entre 2022 et 2023")
```

Nouveaux cas de covid-19 en France entre 2022 et 2023



```
plot(ts_serie3,main="Nouveaux cas de covid-19 en France en 2023")
```

Nouveaux cas de covid-19 en France en 2023



Grâce à cette division, nous allons pouvoir étudier chaque sous-série pertinemment.

Analyse de la première sous-série

Nous avons décidé de nous focaliser sur la première sous-série.

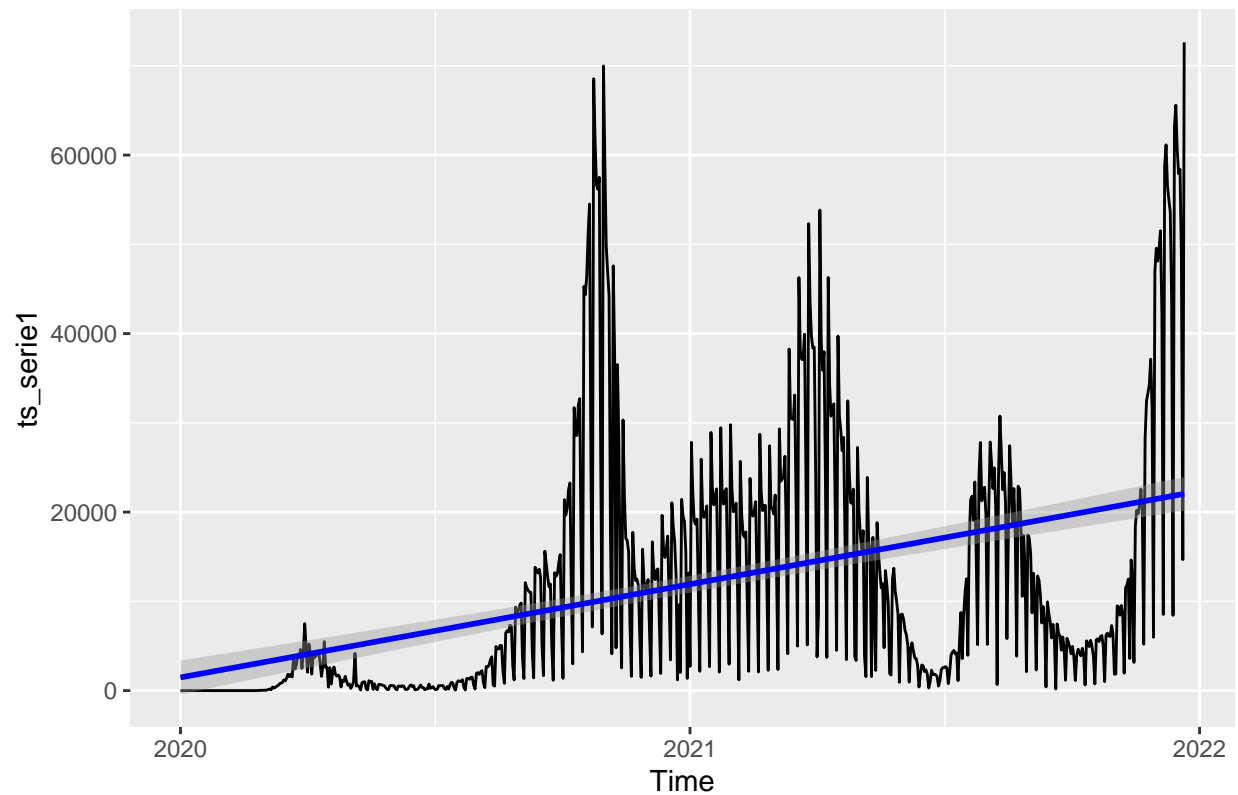
Ce choix est expliqué grâce à notre connaissance des événements durant cette année particulière. En effet, les confinements ont pu avoir des conséquences sur notre série et nos données. Notre étude commence donc le 1er Mars 2020 et s'étend jusqu'au 22 décembre 2021.

Pour rappel, notre série présente une tendance à la hausse comme le montre le graphique ci-dessous. Elle présente aussi une saisonnalité, mais elle n'est pas régulière. En effet, les différentes hausses de nouveaux cas de covid dépendent des confinements et des mesures sanitaires mises en place.

```
autoplot(ts_serie1)+  
  geom_smooth(method = lm,color="blue")+  
  ggtitle("Nouveaux cas de covids en France entre 2020 et 2022")
```

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

Nouveaux cas de covids en France entre 2020 et 2022



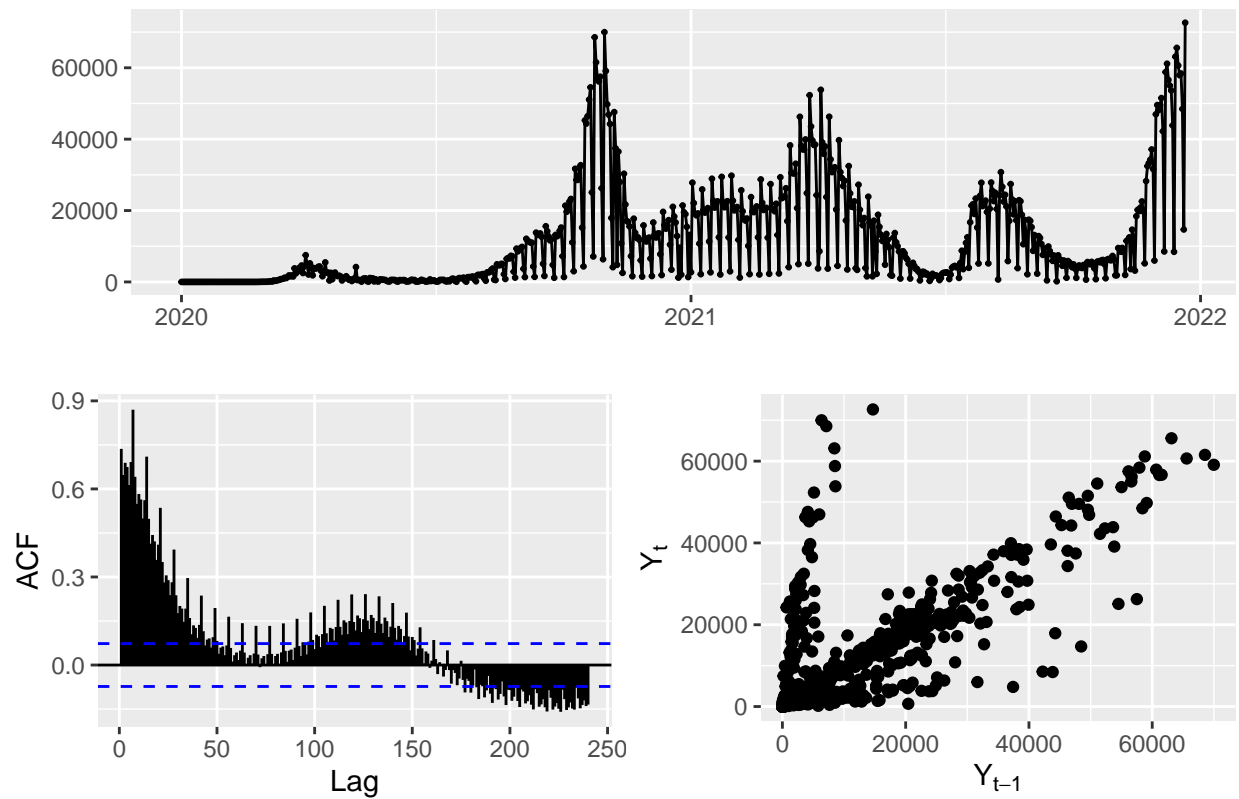
On va d'abord chercher à décrire notre série grâce à des indicateurs descriptifs simples.

```
mean(ts_serie1)
```

```
## [1] 11763.34
```

Entre le 1er Mars 2020 et le 22 Décembre 2022, la moyenne des nouveaux cas de covids par jour était de 11 763 cas en France.

```
ts_serie1 |>  
  ggtsdisplay(plot.type = "scatter",smooth=FALSE)
```

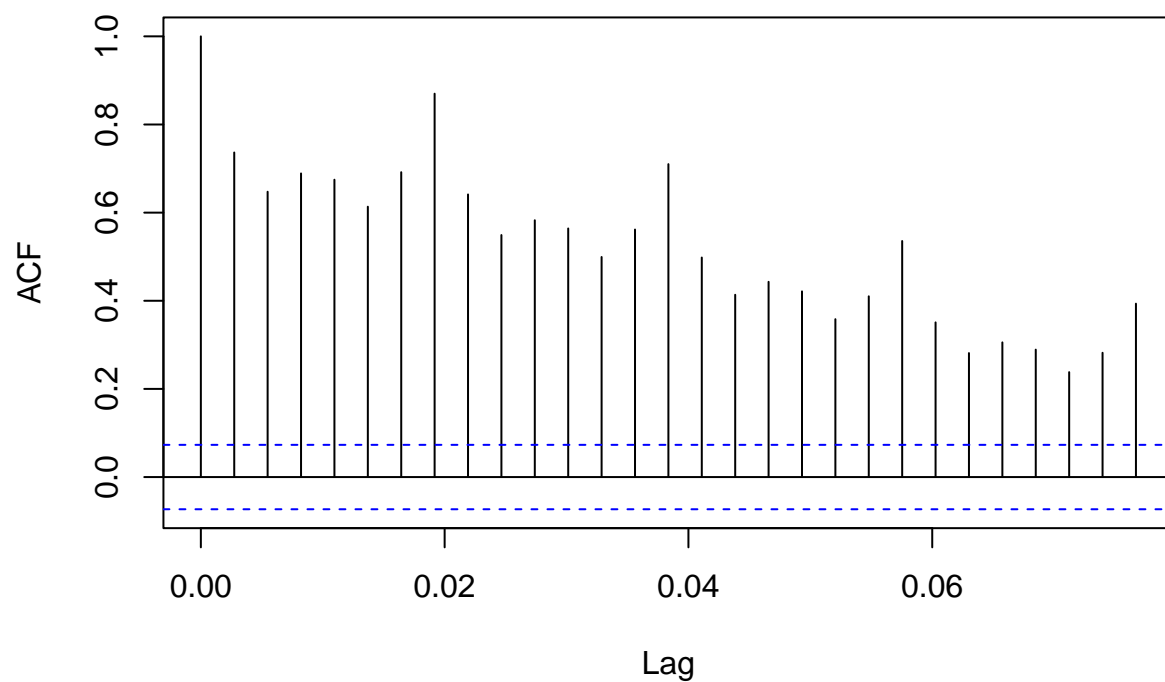


Nous pouvons faire quelques observations sur le graphique de l'ACF. Ce graphique nous permet de détecter une structure de corrélation du réseau. Dans notre cas, plusieurs autocorrélations présentent des valeurs significativement non nulles, ce qui signifie que la série chronologique n'est pas aléatoire.

Afin d'avoir une analyse plus exhaustive, nous pouvons analyser l'ACF et la PACF

```
acf <- acf(ts_serie1)
```


Series ts_serie1



```
print(data.frame(acf$lag,acf$acf))
```

```
##      acf.lag  acf.acf
## 1 0.00000000 1.000000
## 2 0.002739726 0.7365088
## 3 0.005479452 0.6474386
## 4 0.008219178 0.6890097
## 5 0.010958904 0.6747133
## 6 0.013698630 0.6134778
## 7 0.016438356 0.6918098
## 8 0.019178082 0.8699193
## 9 0.021917808 0.6413418
## 10 0.024657534 0.5491586
## 11 0.027397260 0.5827231
## 12 0.030136986 0.5640830
## 13 0.032876712 0.4993613
## 14 0.035616438 0.5617146
## 15 0.038356164 0.7100309
## 16 0.041095890 0.4981678
## 17 0.043835616 0.4136221
## 18 0.046575342 0.4429968
## 19 0.049315068 0.4213644
## 20 0.052054795 0.3582470
## 21 0.054794521 0.4101775
## 22 0.057534247 0.5356155
```

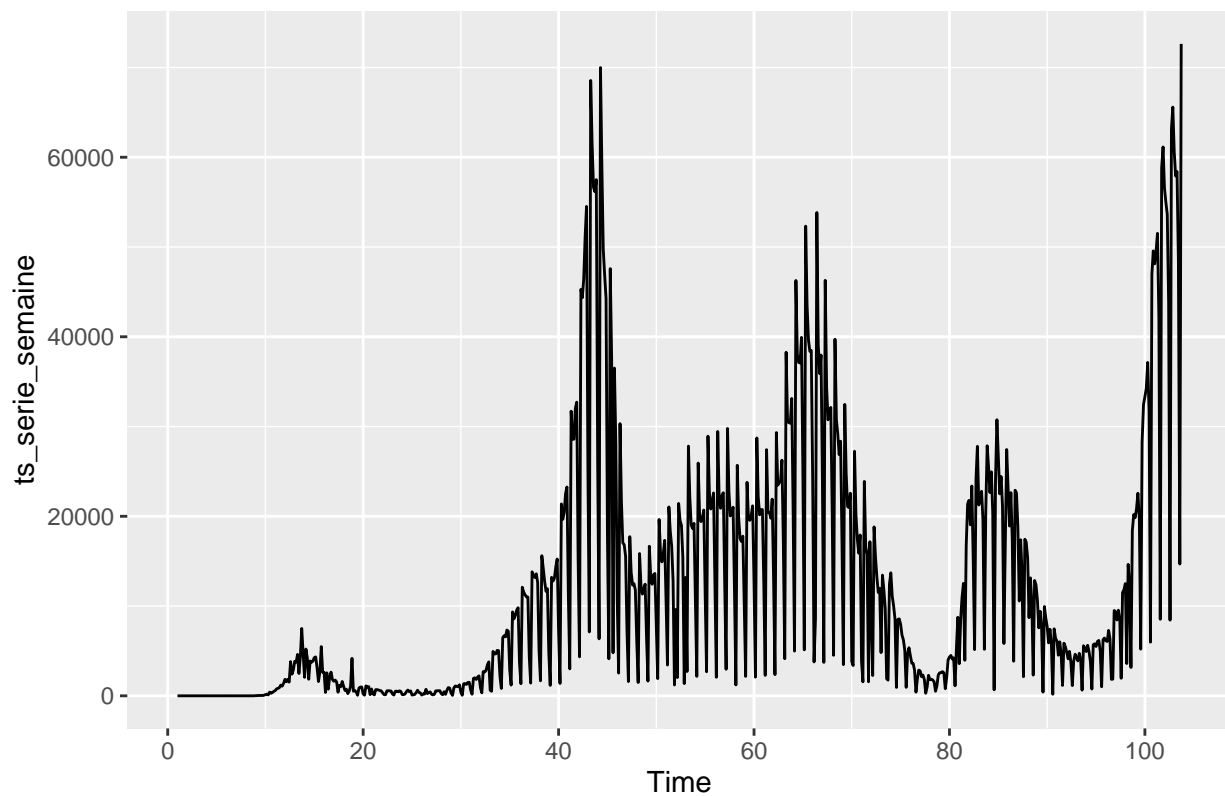
```
## 23 0.060273973 0.3509638
## 24 0.063013699 0.2813732
## 25 0.065753425 0.3056083
## 26 0.068493151 0.2891480
## 27 0.071232877 0.2380992
## 28 0.073972603 0.2820158
## 29 0.076712329 0.3934595
```

```
# plot(data.frame(acf$lag,acf$acf))
```

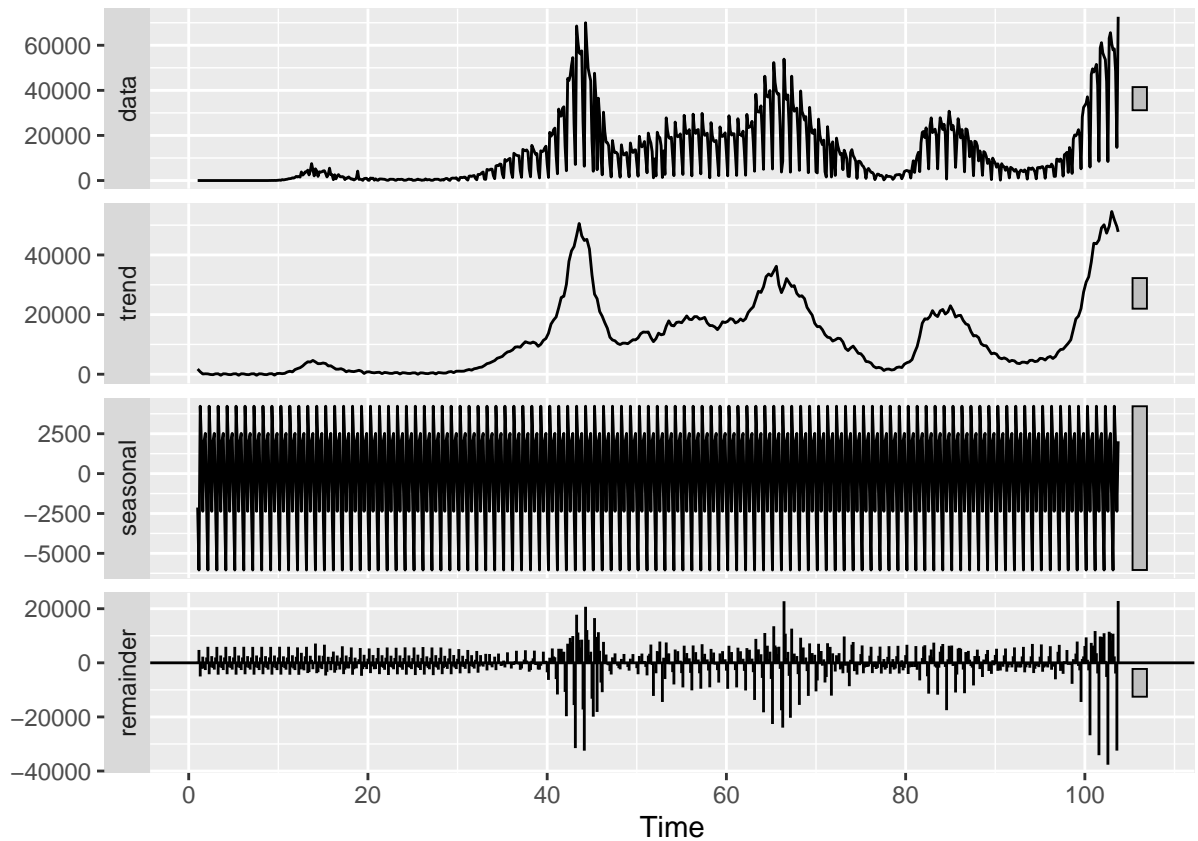
Ce graphique nous permet d'observer une corrélation hebdomadaire. En effet, nous pouvons remarquer un pic plus élevé tous les 7 jours.

Puisque notre série montre une corrélation hebdomadaire, nous avons choisi d'étudier une série temporelle avec une fréquence de 7 jours.

```
ts_serie_semaine <- ts(ts_serie1, frequency = 7)
autoplot(ts_serie_semaine) # Visualisation des données
```



```
decomp_ts <- stl(na.omit(ts_serie_semaine), s.window = "periodic")
autoplot(decomp_ts)
```



Retrait de la tendance / saisonnalité

Nous cherchons à nous ramener à une série sans tendance. Afin de la retirer de notre série, nous allons utiliser l'opérateur diff. Nous allons donc construire un filtre permettant de construire notre analyse.

Tout d'abord, nous avons décidé de ne pas transformer notre série en logarithme. Celle-ci nous permettrait de réduire sa variance mais puisque la série présente des valeurs nulles, cette transformation n'est pas pertinente.

Nous allons donc différencier la série.

```
ndiffs(ts_serie_semaine) # On nous conseille de différencier la série 1 fois
```

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

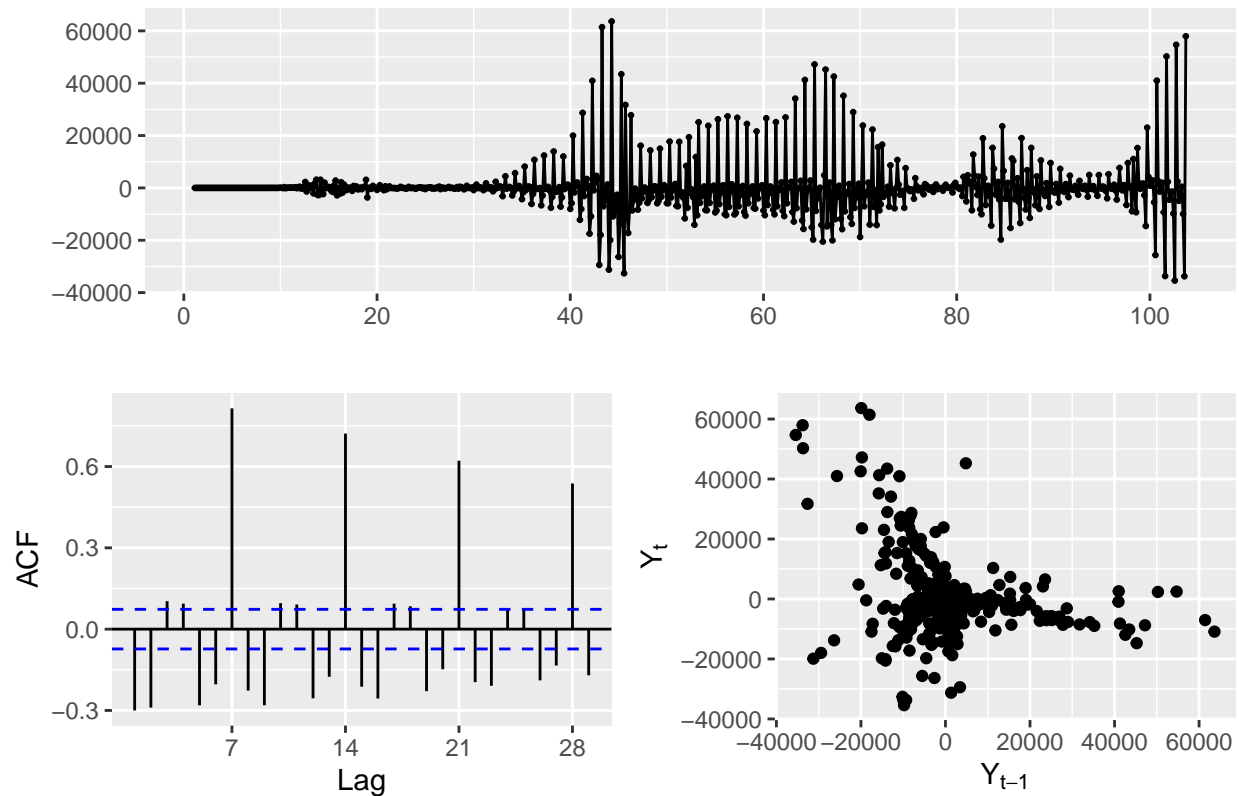
```
serie_lissee <- ts_serie_semaine |>
  diff(lag=1)
```

```
ndiffs(serie_lissee) # la série a été différencié comme il faut
```

```
## [1] 0
```

Puisque l'on différencie une seule fois, la variance augmente mais reste plus faible que si l'on avait différencié 3 ou 4 fois par exemple.

```
serie_lissee |>
  ggtsdisplay(plot.type = "scatter", smooth=FALSE)
```



Grâce à cette méthode, nous pouvons faire plusieurs constats : - **Le chronogramme** : Notre différenciation nous permet de stationariser notre série et supprimer la tendance. Nous pouvons donc émettre l'hypothèse d'un modèle polynomial.

- **L'ACF** : L'auto-corrélation décroît car on fait des moyennes sur de moins en moins de valeurs. De plus, nous pouvons observer une saisonnalité hebdomadaire avec un pic tous les 7 jours (lag).
- **Corrélation** : Le nuage de points ne présente pas de direction.

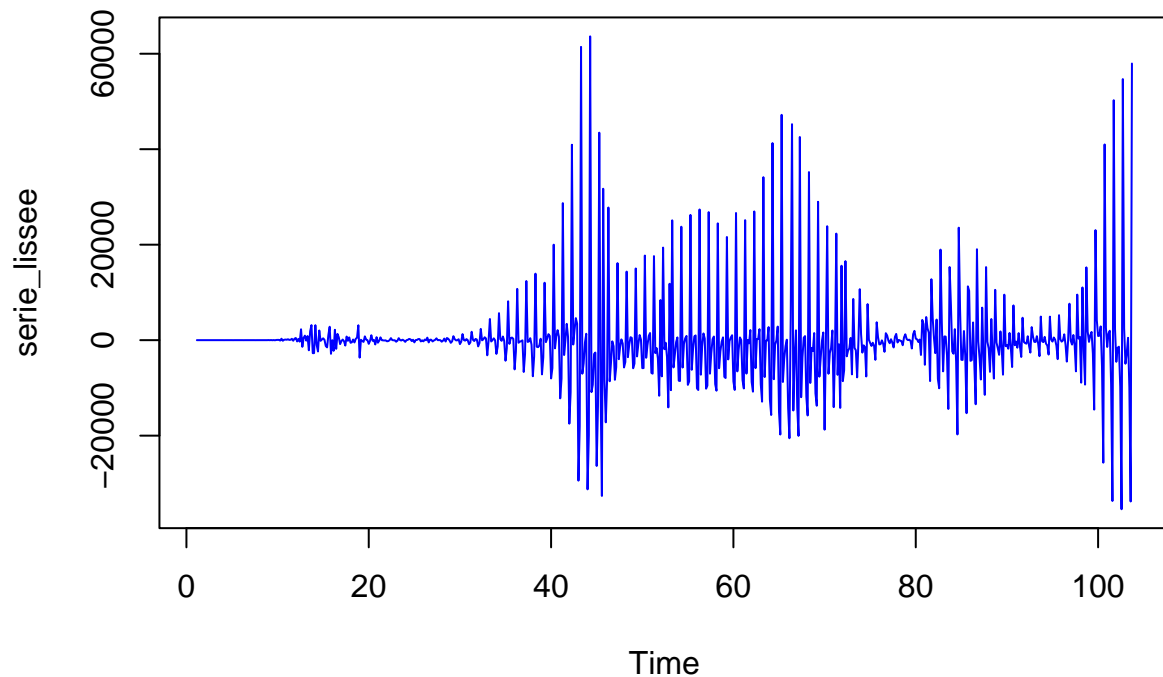
Retrait de la saisonnalité

Pour pouvoir réaliser des tests de stationnarité, nous devons d'abord avoir une série sans tendance ni de saisonnalité.

Ici, notre série n'a plus de tendance, il faut maintenant enlever la saisonnalité.

```
ts.plot(serie_lissee, main = "Serie différenciée", col = "blue")
```

Serie différenciée



Stationnarité

Ensuite, nous allons tester la stationnarité de notre série :

```
kpss.test(serie_lissee)
```

```
## Warning in kpss.test(serie_lissee): p-value greater than printed p-value
```

```
##  
## KPSS Test for Level Stationarity  
##  
## data: serie_lissee  
## KPSS Level = 0.26302, Truncation lag parameter = 6, p-value = 0.1
```

Nous avons une p-value associée au test supérieure à 5%, nous rejettons donc l'hypothèse nulle de stationnarité. Notre série n'est donc pas stationnaire.

```
adf.test(serie_lissee)
```

```
## Warning in adf.test(serie_lissee): p-value smaller than printed p-value
```

```
##  
## Augmented Dickey-Fuller Test
```

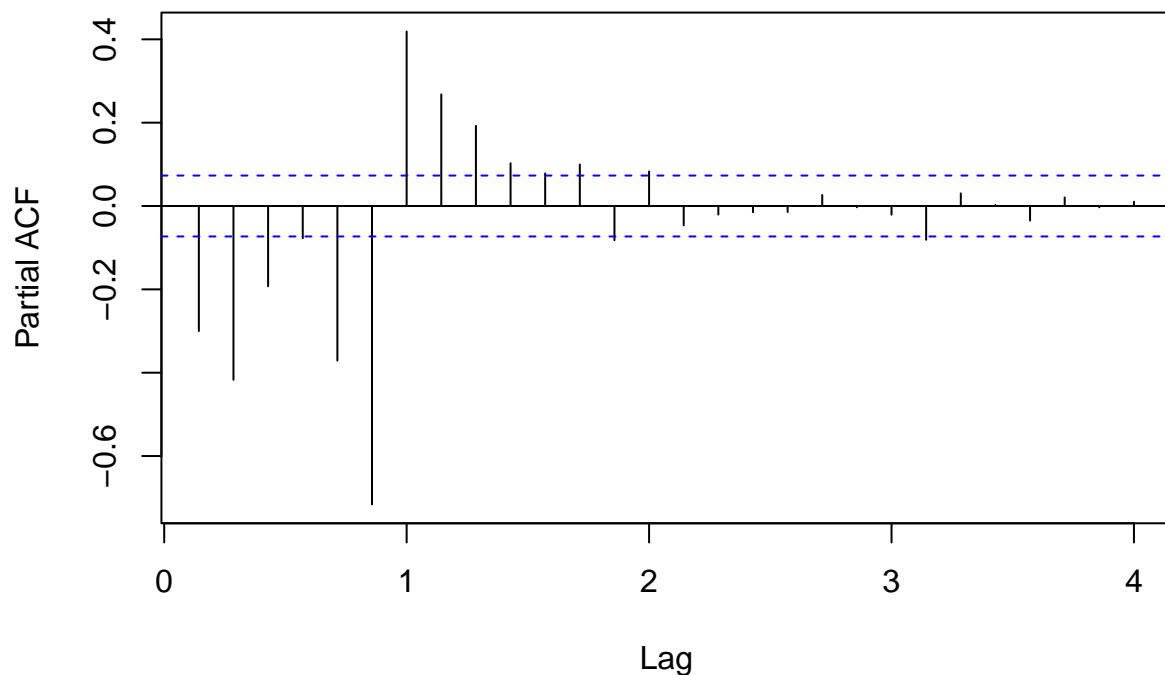
Lors de ce test, nous obtenons une p-value inférieure à 5%. Nous acceptons l'hypothèse de non-stationnarité. Ce test nous permet de confirmer notre hypothèse précédente.

En conclusion, notre série n'est pas **stationnaire**.

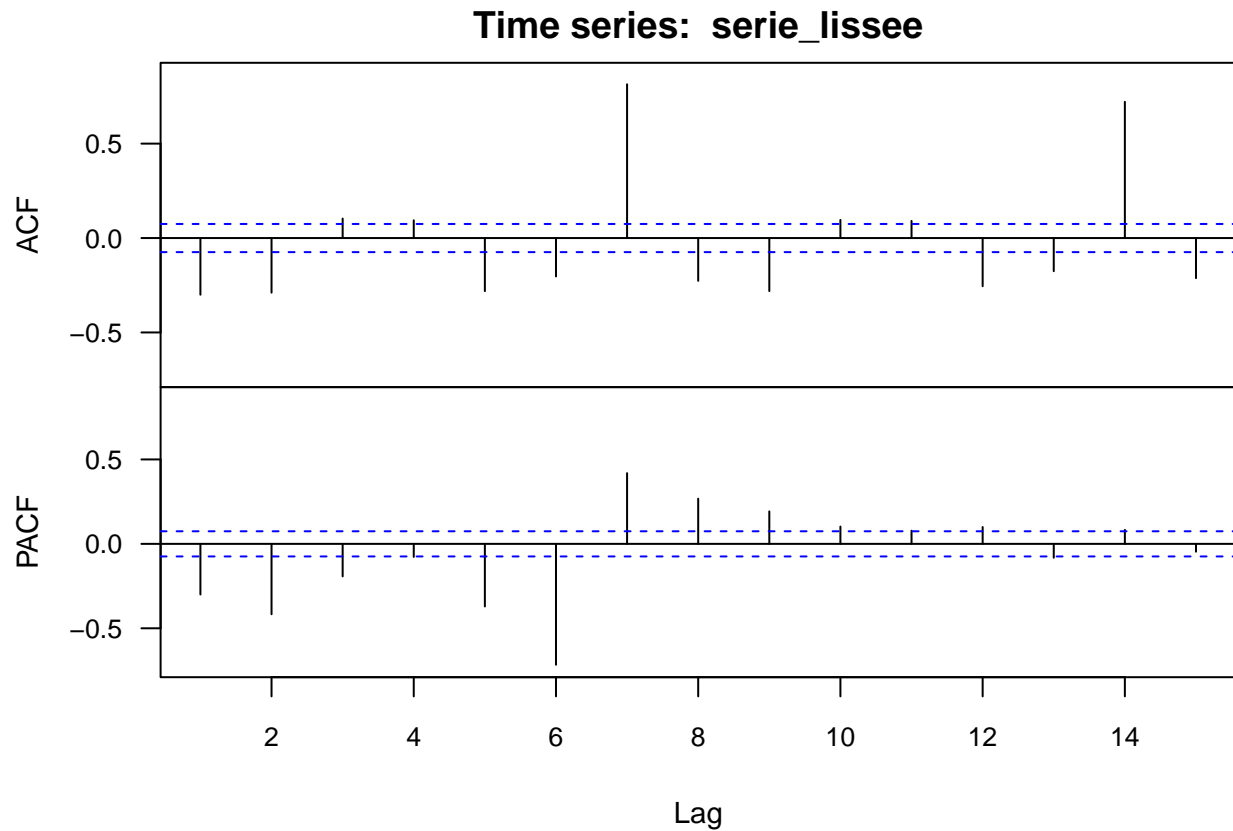
Afin de détecter la présence d'autocorrélation partielle, nous allons utiliser la fonction `pacf`. Elle nous permet de mesurer l'aurocorrélation d'un signal pour un décalage k "indépendamment" des autocorrélations pour les décalages inférieurs.

Nous avons choisi d'utiliser cette fonction car le corrélogramme produit par la fonction `ggdisplay` montre des autocorrélations fortes à répétition.

Series serie_lissee



```
acf2y(serie_lissee, lag.max = 15)
```



```
##      LAG      ACF1      PACF
## [1,]  1 -0.30006607 -0.30006607
## [2,]  2 -0.28932338 -0.41690062
## [3,]  3  0.10323365 -0.19270179
## [4,]  4  0.09411426 -0.07714966
## [5,]  5 -0.28121680 -0.37099581
## [6,]  6 -0.20356944 -0.71598887
## [7,]  7  0.81450655  0.41882851
## [8,]  8 -0.22680907  0.26782417
## [9,]  9 -0.28089902  0.19242924
## [10,] 10  0.09603546  0.10251255
## [11,] 11  0.09084251  0.07823376
## [12,] 12 -0.25529630  0.09978066
## [13,] 13 -0.17562952 -0.08206719
## [14,] 14  0.72149907  0.08291360
## [15,] 15 -0.21237168 -0.04646638
```

Estimation du modèle

Nous allons essayer de choisir le meilleur modèle afin d'estimer notre série.

Afin de pouvoir estimer le modèle, nous avons utilisé la fonction `auto.arima()` du package `forecast` qui permet d'effectuer une modélisation automatique. En précisant les arguments `trace=T` et `ic=aic`, nous avons donné la main au logiciel R de sélectionner le meilleur modèle sur la base du critère AIC

```
model_arima <- auto.arima(serie_lissee, trace=T, ic = "aic")
```

ARIMA

```
##
## Fitting models using approximations to speed things up...
##
## ARIMA(2,0,2)(1,1,1)[7] with drift : 13847.67
## ARIMA(0,0,0)(0,1,0)[7] with drift : 14195.3
## ARIMA(1,0,0)(1,1,0)[7] with drift : 13985.9
## ARIMA(0,0,1)(0,1,1)[7] with drift : 13852.64
## ARIMA(0,0,0)(0,1,0)[7] : 14193.3
## ARIMA(2,0,2)(0,1,1)[7] with drift : 13847.43
## ARIMA(2,0,2)(0,1,0)[7] with drift : 13882.05
## ARIMA(2,0,2)(0,1,2)[7] with drift : 13849.42
## ARIMA(2,0,2)(1,1,0)[7] with drift : 13855.43
## ARIMA(2,0,2)(1,1,2)[7] with drift : Inf
## ARIMA(1,0,2)(0,1,1)[7] with drift : 13857.44
## ARIMA(2,0,1)(0,1,1)[7] with drift : 13854.16
## ARIMA(3,0,2)(0,1,1)[7] with drift : Inf
## ARIMA(2,0,3)(0,1,1)[7] with drift : 13847.45
## ARIMA(1,0,1)(0,1,1)[7] with drift : 13855.53
## ARIMA(1,0,3)(0,1,1)[7] with drift : 13846.98
## ARIMA(1,0,3)(0,1,0)[7] with drift : 13881.03
## ARIMA(1,0,3)(1,1,1)[7] with drift : 13846.76
## ARIMA(1,0,3)(1,1,0)[7] with drift : 13855.17
## ARIMA(1,0,3)(2,1,1)[7] with drift : 13846.9
## ARIMA(1,0,3)(1,1,2)[7] with drift : 13857.54
## ARIMA(1,0,3)(0,1,2)[7] with drift : 13848.97
## ARIMA(1,0,3)(2,1,0)[7] with drift : 13863.74
## ARIMA(1,0,3)(2,1,2)[7] with drift : 13847.32
## ARIMA(0,0,3)(1,1,1)[7] with drift : 13845.78
## ARIMA(0,0,3)(0,1,1)[7] with drift : 13850.7
## ARIMA(0,0,3)(1,1,0)[7] with drift : 13858.58
## ARIMA(0,0,3)(2,1,1)[7] with drift : 13849.71
## ARIMA(0,0,3)(1,1,2)[7] with drift : Inf
## ARIMA(0,0,3)(0,1,0)[7] with drift : 13878.07
## ARIMA(0,0,3)(0,1,2)[7] with drift : 13852.59
## ARIMA(0,0,3)(2,1,0)[7] with drift : 13867.4
## ARIMA(0,0,3)(2,1,2)[7] with drift : 13849.79
## ARIMA(0,0,2)(1,1,1)[7] with drift : 13847.62
## ARIMA(0,0,4)(1,1,1)[7] with drift : 13846.85
## ARIMA(1,0,2)(1,1,1)[7] with drift : 13850.64
## ARIMA(1,0,4)(1,1,1)[7] with drift : Inf
## ARIMA(0,0,3)(1,1,1)[7] : 13843.91
## ARIMA(0,0,3)(0,1,1)[7] : 13848.74
## ARIMA(0,0,3)(1,1,0)[7] : 13856.61
## ARIMA(0,0,3)(2,1,1)[7] : 13847.82
## ARIMA(0,0,3)(1,1,2)[7] : Inf
## ARIMA(0,0,3)(0,1,0)[7] : 13876.1
## ARIMA(0,0,3)(0,1,2)[7] : 13850.63
## ARIMA(0,0,3)(2,1,0)[7] : 13865.43
```



```
## ARIMA(0,0,3)(2,1,2)[7] : 13847.9
## ARIMA(0,0,2)(1,1,1)[7] : 13845.76
## ARIMA(1,0,3)(1,1,1)[7] : 13844.88
## ARIMA(0,0,4)(1,1,1)[7] : 13844.98
## ARIMA(1,0,2)(1,1,1)[7] : 13848.81
## ARIMA(1,0,4)(1,1,1)[7] : Inf
##
## Now re-fitting the best model(s) without approximations...
##
## ARIMA(0,0,3)(1,1,1)[7] : 13960.51
##
## Best model: ARIMA(0,0,3)(1,1,1)[7]
```

Modèle identifier : ARIMA(4,0,4)

```
summary(model_arima)
```

```
## Series: serie_lissee
## ARIMA(0,0,3)(1,1,1)[7]
##
## Coefficients:
##          ma1          ma2          ma3          sar1          sma1
##      -0.7378  -0.0419   0.0793   0.7009  -0.8852
## s.e.    0.0380   0.0462   0.0394   0.0554   0.0379
##
## sigma^2 = 18925739: log likelihood = -6974.25
## AIC=13960.51 AICc=13960.63 BIC=13987.92
##
## Training set error measures:
##              ME      RMSE      MAE MPE MAPE      MASE      ACF1
## Training set 53.96374 4313.916 2168.68 NaN  Inf 0.9191823 0.001809301
```

```
t_stat(model_arima)
```

```
##              ma1          ma2          ma3          sar1          sma1
## t.stat -19.44142 -0.906692  2.013372 12.65305 -23.37542
## p.val    0.00000  0.364570 0.044075  0.00000  0.00000
```

Le modèle n'est pas simplifiable

```
# auto.arima(serie_lissee,d=1,D=1,stepwise = FALSE,approximation=FALSE,trace=TRUE)
```

Estimation par régression linéaire (MCO)

```
t <- 1:length(ts_serie1)
x <- outer(t,1:6)*(pi/6)
df <- data.frame(ts_serie1,t,cos(x),sin(x[,-6]))
ts_serie1_lm <- lm(data=df,ts_serie1~.)
summary(ts_serie1_lm)
```

```
##
## Call:
## lm(formula = ts_serie1 ~ ., data = df)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -19426  -7072  -2665   4848  60392
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  1456.745    971.698   1.499   0.134
## t             28.590     2.335  12.243 <2e-16 ***
## X1          -157.233    686.311  -0.229   0.819
## X2           355.712    686.311   0.518   0.604
## X3            6.227    686.311   0.009   0.993
## X4           274.870    686.311   0.401   0.689
## X5           241.654    686.311   0.352   0.725
## X6            61.208    485.294   0.126   0.900
## X1.1         -228.513    686.363  -0.333   0.739
## X2.1         -177.238    686.319  -0.258   0.796
## X3.1          -80.877    686.311  -0.118   0.906
## X4.1          -61.130    686.309  -0.089   0.929
## X5.1          188.947    686.308   0.275   0.783
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 13020 on 707 degrees of freedom
## Multiple R-squared:  0.1761, Adjusted R-squared:  0.1621
## F-statistic: 12.59 on 12 and 707 DF, p-value: < 2.2e-16
```

Etude des résidus

-> test de blancheur : box.test -> test de durbin watson : test auto corrélation des erreurs du modele arima
->