Clovis Deletre

Charles Vitry

# Introduction

Nous souhaitons réalisé l’**étude d’une série temporelle** et faire des prévisions sur celle-ci.

Cette série temporelle est le trafic mensuel d’une Compagnie aérienne de janvier 2011 à août 2019.

Nos prévisions portent sur les 8 mois de l’année 2019

# Représentation graphique de la série.

## Import des données

Import de la base, on sélectionne la colonne des valeurs

library(readr)  
data <- read\_delim("Trafic-voyageurs.csv",   
 delim = ";", locale = locale(encoding = "ISO-8859-1"))

## Rows: 104 Columns: 2  
## ── Column specification ────────────────────────────────────────────────────────  
## Delimiter: ";"  
## chr (1): dates  
## dbl (1): trafic  
##   
## ℹ Use `spec()` to retrieve the full column specification for this data.  
## ℹ Specify the column types or set `show\_col\_types = FALSE` to quiet this message.

data\_value <- data[,2]  
summary(data)

## dates trafic   
## Length:104 Min. :220876   
## Class :character 1st Qu.:297154   
## Mode :character Median :355178   
## Mean :354651   
## 3rd Qu.:407331   
## Max. :505190

## Création de notre série temporelle

Nous représentons nos données sous forme de série temporelle.

Une série temporelle est un ensemble de métrique mésurée sur des intervalles de temps réguliers.

Création de la série chronologique avec la librairie TSstudio :

* start : année de début,
* frequency : nbr de valeur par an => on en fréquence mensuelle donc 12

library(TSstudio)  
data\_ts <- ts(data\_value, start=2011, frequency=12)  
plot\_1\_TimeSeries(data\_ts)

##   
## Attachement du package : 'plotly'

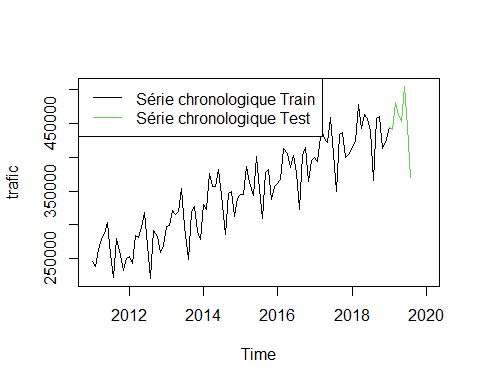
## L'objet suivant est masqué depuis 'package:ggplot2':  
##   
## last\_plot

## L'objet suivant est masqué depuis 'package:stats':  
##   
## filter

## L'objet suivant est masqué depuis 'package:graphics':  
##   
## layout

## Séparation jeu de données

#revoir l affichage car ca prend pas en compte tt 2019  
data\_ts\_train <- window(data\_ts, start = c(2011, 1), end = c(2018,12))  
data\_ts\_test <- window(data\_ts, start= c(2019,1), end = c(2019,8))  
  
names(data)[1] <- "ds"  
names(data)[2] <- "y"  
data\_train <- data[1:96,]  
data\_test <- data[97:104,]  
  
  
plot(data\_ts, xlim=c(2011,2020))  
lines(data\_ts\_test, col=3)  
legend("topleft", lty = 1, col=c(1,3), legend=c("Série chronologique Train", "Série chronologique Test"))



-> strong trend -> patern qui se repete, saisonnalité ?

## Représentation de la saisonnalité

Analyse de la saisonnalité en superposant chaque année (par mois):

-> en supprimant la tendance on voit bien la saisonnalité => saisonnalité régulière

## Analyse de notre série temporelle

Chaque point de notre série temporelle peut être exprimer comme une somme ou un produit de 3 composantes : - Saisonnalité (St), - Tendance (Tt), - Erreur (ϵt),

Yt=St+Tt+ϵt ou Yt=St×Tt×ϵt

La stationnarité d’une série signifique que le processus qui génère la série ne change pas dans le temps. Cela ne veut pas dire que la série ne change pas dans le temps, mais que la façon dont elle change, n’est pas modifié dans le temps.

Testons si la série est stationnaire :

library(tseries)  
adf.test(data\_ts) #p-value <0.5 => stationnaire

## Warning in adf.test(data\_ts): p-value smaller than printed p-value

##   
## Augmented Dickey-Fuller Test  
##   
## data: data\_ts  
## Dickey-Fuller = -5.4857, Lag order = 4, p-value = 0.01  
## alternative hypothesis: stationary

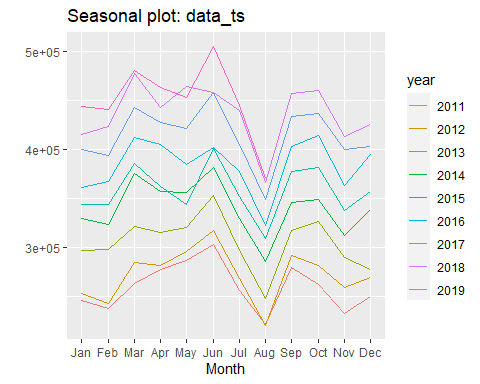
kpss.test(data\_ts)

## Warning in kpss.test(data\_ts): p-value smaller than printed p-value

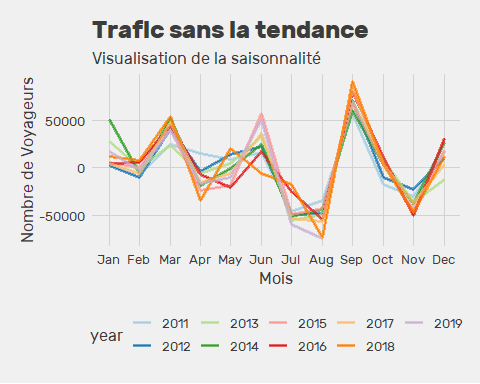
##   
## KPSS Test for Level Stationarity  
##   
## data: data\_ts  
## KPSS Level = 2.0803, Truncation lag parameter = 4, p-value = 0.01

Donc notre série est bien stationnaire et peut être étudiée facilement.

ggseasonplot(data\_ts)



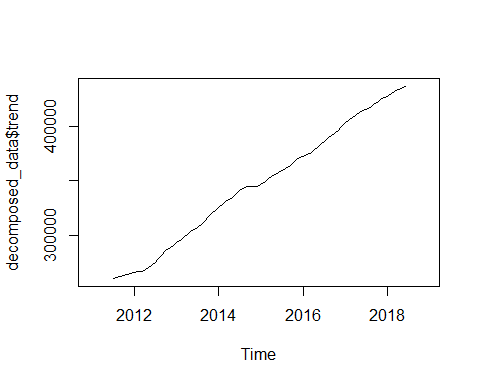
data\_ts\_without\_trend = diff(data\_ts)  
ggseasonplot(data\_ts\_without\_trend)+  
 labs(title = "Trafic sans la tendance",  
 subtitle = "Visualisation de la saisonnalité",  
 x="Mois",  
 y= "Nombre de Voyageurs")+  
 geom\_line(size=1.1, alpha = 0.65)+  
 theme\_fivethirtyeight()+  
 theme(axis.title = element\_text())+  
 scale\_color\_brewer(palette = "Paired") +  
 theme(axis.title = element\_text(), text = element\_text(family = "Rubik"))



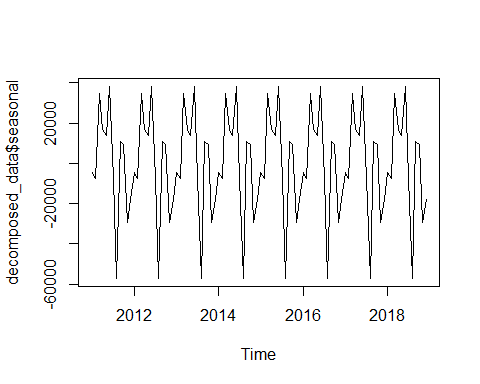
## Représentation des décompositions possibles

DECOMPOSITION : additive / Multiplicative Ts = Trend + Seasonal + Random / Ts = Trend \* Seasonal \* Random

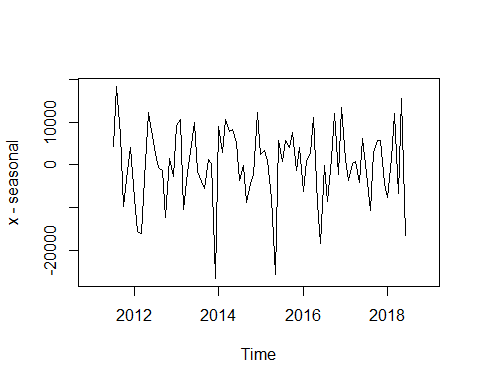
decomposed\_data <- decompose(data\_ts\_train, type="additive")  
plot(decomposed\_data$trend)



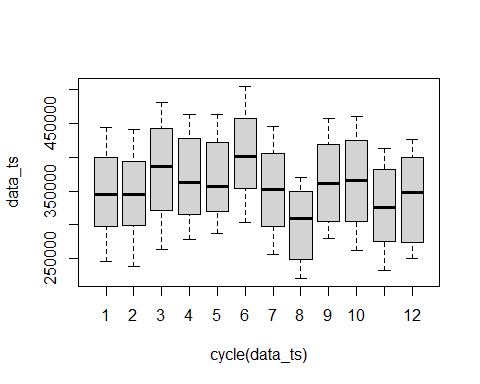
plot(decomposed\_data$seasonal)



plot(decomposed\_data$random)



boxplot(data\_ts ~ cycle(data\_ts))

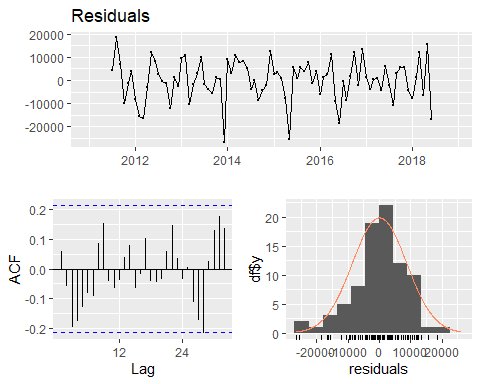


-> on distingue des saisonnalités => faire régression ca n’a pas de sens => modèle de Buys Ballot

-> bonne repartition du bruit -> quelques outliers

checkresiduals(remainder(decomposed\_data))

## Warning in modeldf.default(object): Could not find appropriate degrees of  
## freedom for this model.



On a tendances + saisonnalité

# Modèles espace-état

* meanf : Average Method : prend la valeur moyenne de toute les observations pour toutes les prédictions,
* naive : Naive Method : prend la dernière observation pour toutes les prédictions,
* drift : Drift Method : prend la première et la dernière observations et trace une lignes entre les deux, on utilise la courbe pour les prédictions,
* snaive : Seasonal Naive Forecast : Prend la dernière valeur de la saison précédente comme prédiction (ex : sept 2018 = sep 2019 + erreur)

library(forecast)  
mean <- meanf(data\_ts\_train, h=8)  
naivem <- naive(data\_ts\_train, h=8)  
driftm <- rwf(data\_ts\_train, h=8, drif=T)  
snaivem <- snaive(data\_ts\_train, h=8)

plot(mean, plot.conf = F, main="")

## Warning in plot.window(xlim, ylim, log, ...): "plot.conf" n'est pas un paramètre  
## graphique

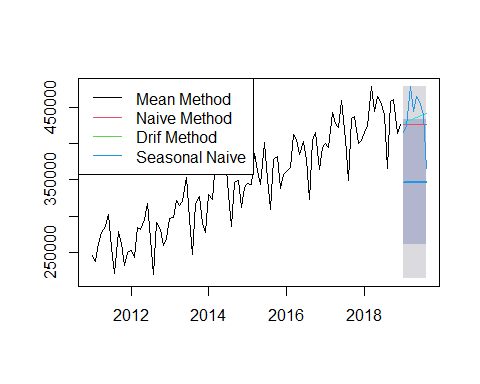
## Warning in title(main = main, xlab = xlab, ylab = ylab, ...): "plot.conf" n'est  
## pas un paramètre graphique

## Warning in axis(1, ...): "plot.conf" n'est pas un paramètre graphique

## Warning in axis(2, ...): "plot.conf" n'est pas un paramètre graphique

## Warning in box(...): "plot.conf" n'est pas un paramètre graphique

lines(naivem$mean, col=2, lty=1)  
lines(driftm$mean, col=5, lty=1)  
lines(snaivem$mean, col = 4, lty=1)  
legend("topleft", lty=1, col=c(1,2,3,4), legend=c("Mean Method", "Naive Method", "Drif Method", "Seasonal Naive"))



#comparaison :  
plot(snaivem, plot.conf = F, main="")

## Warning in plot.window(xlim, ylim, log, ...): "plot.conf" n'est pas un paramètre  
## graphique

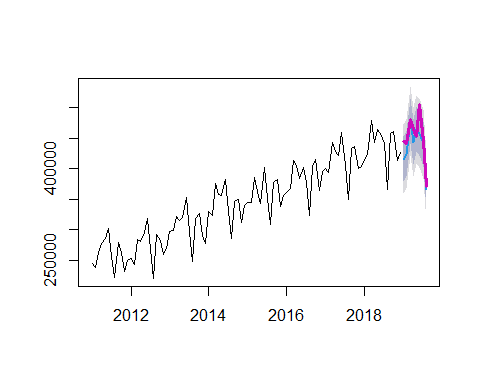
## Warning in title(main = main, xlab = xlab, ylab = ylab, ...): "plot.conf" n'est  
## pas un paramètre graphique

## Warning in axis(1, ...): "plot.conf" n'est pas un paramètre graphique

## Warning in axis(2, ...): "plot.conf" n'est pas un paramètre graphique

## Warning in box(...): "plot.conf" n'est pas un paramètre graphique

lines(data\_ts\_test, col = 6, lty=1, lwd=3)



plot(driftm, plot.conf = F, main="")

## Warning in plot.window(xlim, ylim, log, ...): "plot.conf" n'est pas un paramètre  
## graphique

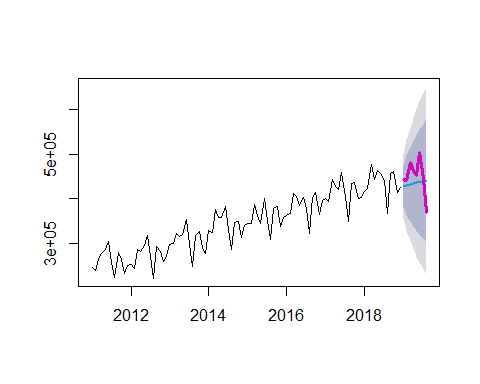
## Warning in title(main = main, xlab = xlab, ylab = ylab, ...): "plot.conf" n'est  
## pas un paramètre graphique

## Warning in axis(1, ...): "plot.conf" n'est pas un paramètre graphique

## Warning in axis(2, ...): "plot.conf" n'est pas un paramètre graphique

## Warning in box(...): "plot.conf" n'est pas un paramètre graphique

lines(data\_ts\_test, col = 6, lty=1, lwd=3)



On regarde : MAE : Mean Absolute Error : RMSE : Root Mean Squarred Error

MASE : Mean Absolute Scaled Error : MAPE : Mean Absolute Percentage Error :

res = pred - val MAE = sum(abs(res))/length(val) RSS = sum(res^2) MSE = RSS/length(val) RMSE = sqrt(MSE)

La plus populaire est la MAPE

MAPE(y\_pred, y\_true)

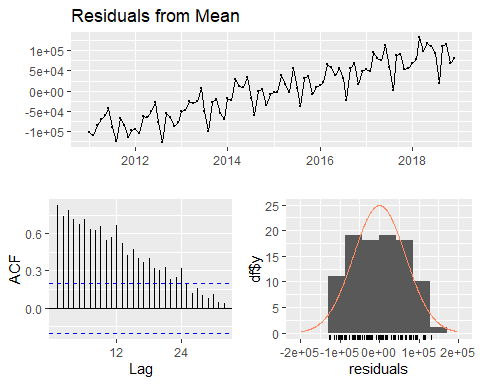
$MAPE = (1/n) \* Σ(|actual – forecast| / |actu0al|) \* 10

“a MAPE value of 6% means that the average difference between the forecasted value and the actual value is 6%”

print(summary(mean))

##   
## Forecast method: Mean  
##   
## Model Information:  
## $mu  
## [1] 346667.1  
##   
## $mu.se  
## [1] 6731.642  
##   
## $sd  
## [1] 65956.35  
##   
## $bootstrap  
## [1] FALSE  
##   
## $call  
## meanf(y = data\_ts\_train, h = 8)  
##   
## attr(,"class")  
## [1] "meanf"  
##   
## Error measures:  
## ME RMSE MAE MPE MAPE MASE  
## Training set 1.941958e-11 65611.93 55535.08 -3.855657 17.01186 2.16177  
## ACF1  
## Training set 0.8254447  
##   
## Forecasts:  
## Point Forecast Lo 80 Hi 80 Lo 95 Hi 95  
## Jan 2019 346667.1 261106.5 432227.7 215047 478287.2  
## Feb 2019 346667.1 261106.5 432227.7 215047 478287.2  
## Mar 2019 346667.1 261106.5 432227.7 215047 478287.2  
## Apr 2019 346667.1 261106.5 432227.7 215047 478287.2  
## May 2019 346667.1 261106.5 432227.7 215047 478287.2  
## Jun 2019 346667.1 261106.5 432227.7 215047 478287.2  
## Jul 2019 346667.1 261106.5 432227.7 215047 478287.2  
## Aug 2019 346667.1 261106.5 432227.7 215047 478287.2

checkresiduals(mean)



##   
## Ljung-Box test  
##   
## data: Residuals from Mean  
## Q\* = 731.64, df = 18, p-value < 2.2e-16  
##   
## Model df: 1. Total lags used: 19

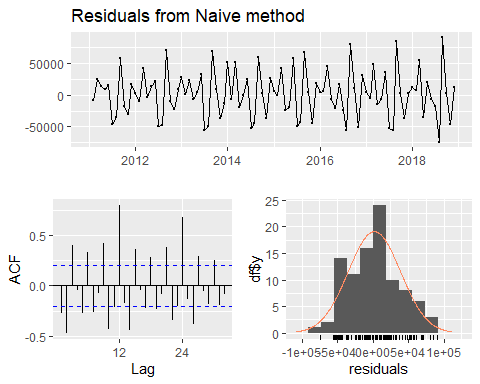
accuracy(mean, data\_ts\_test)

## ME RMSE MAE MPE MAPE MASE  
## Training set 1.941958e-11 65611.93 55535.08 -3.855657 17.01186 2.161770  
## Test set 1.037870e+05 110031.92 103787.04 22.486144 22.48614 4.040036  
## ACF1 Theil's U  
## Training set 0.8254447 NA  
## Test set 0.0485288 2.517689

print(summary(naivem))

##   
## Forecast method: Naive method  
##   
## Model Information:  
## Call: naive(y = data\_ts\_train, h = 8)   
##   
## Residual sd: 36679.9508   
##   
## Error measures:  
## ME RMSE MAE MPE MAPE MASE  
## Training set 1896.811 36679.95 29013.27 -0.02007386 8.597313 1.129377  
## ACF1  
## Training set -0.2744236  
##   
## Forecasts:  
## Point Forecast Lo 80 Hi 80 Lo 95 Hi 95  
## Jan 2019 426097 379089.8 473104.2 354205.6 497988.4  
## Feb 2019 426097 359618.7 492575.3 324427.2 527766.8  
## Mar 2019 426097 344678.1 507515.9 301577.5 550616.5  
## Apr 2019 426097 332082.5 520111.5 282314.2 569879.8  
## May 2019 426097 320985.6 531208.4 265343.0 586851.0  
## Jun 2019 426097 310953.2 541240.8 249999.8 602194.2  
## Jul 2019 426097 301727.5 550466.5 235890.3 616303.7  
## Aug 2019 426097 293140.4 559053.6 222757.5 629436.5

checkresiduals(naivem)



##   
## Ljung-Box test  
##   
## data: Residuals from Naive method  
## Q\* = 248.52, df = 19, p-value < 2.2e-16  
##   
## Model df: 0. Total lags used: 19

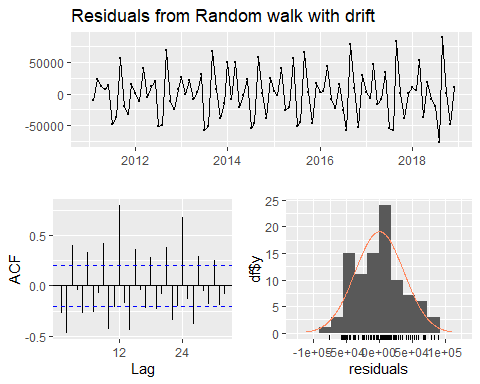
accuracy(naivem, data\_ts\_test)

## ME RMSE MAE MPE MAPE MASE  
## Training set 1896.811 36679.95 29013.27 -0.02007386 8.597313 1.129377  
## Test set 24357.125 43915.19 38328.62 4.72582164 8.499751 1.491988  
## ACF1 Theil's U  
## Training set -0.2744236 NA  
## Test set 0.0485288 1.063155

print(summary(driftm))

##   
## Forecast method: Random walk with drift  
##   
## Model Information:  
## Call: rwf(y = data\_ts\_train, h = 8, drift = T)   
##   
## Drift: 1896.8105 (se 3778.1861)  
## Residual sd: 36825.2032   
##   
## Error measures:  
## ME RMSE MAE MPE MAPE MASE  
## Training set 2.297696e-11 36630.87 28899.04 -0.5861884 8.591266 1.12493  
## ACF1  
## Training set -0.2744236  
##   
## Forecasts:  
## Point Forecast Lo 80 Hi 80 Lo 95 Hi 95  
## Jan 2019 427993.8 380800.4 475187.2 355817.7 500169.9  
## Feb 2019 429890.6 362798.7 496982.5 327282.4 532498.8  
## Mar 2019 431787.4 349190.1 514384.7 305465.7 558109.1  
## Apr 2019 433684.2 337818.7 529549.8 287070.6 580297.9  
## May 2019 435581.1 327854.7 543307.4 270827.8 600334.3  
## Jun 2019 437477.9 318875.0 556080.7 256090.5 618865.2  
## Jul 2019 439374.7 310630.0 568119.3 242476.7 636272.6  
## Aug 2019 441271.5 302958.0 579585.0 229739.3 652803.7

checkresiduals(driftm)



##   
## Ljung-Box test  
##   
## data: Residuals from Random walk with drift  
## Q\* = 248.52, df = 18, p-value < 2.2e-16  
##   
## Model df: 1. Total lags used: 19

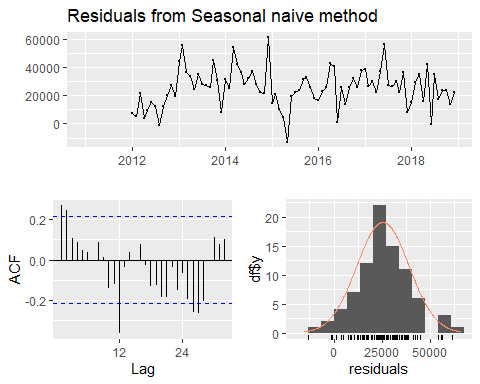
accuracy(driftm, data\_ts\_test)

## ME RMSE MAE MPE MAPE MASE  
## Training set 2.297696e-11 36630.87 28899.04 -0.5861884 8.591266 1.124930  
## Test set 1.582148e+04 41314.60 33586.60 2.7843152 7.582963 1.307399  
## ACF1 Theil's U  
## Training set -0.27442358 NA  
## Test set 0.06907259 1.007801

print(summary(snaivem))

##   
## Forecast method: Seasonal naive method  
##   
## Model Information:  
## Call: snaive(y = data\_ts\_train, h = 8)   
##   
## Residual sd: 28666.7301   
##   
## Error measures:  
## ME RMSE MAE MPE MAPE MASE ACF1  
## Training set 25337.46 28666.73 25689.63 7.101745 7.207375 1 0.2695124  
##   
## Forecasts:  
## Point Forecast Lo 80 Hi 80 Lo 95 Hi 95  
## Jan 2019 415292 378554.1 452029.9 359106.2 471477.8  
## Feb 2019 423665 386927.1 460402.9 367479.2 479850.8  
## Mar 2019 478207 441469.1 514944.9 422021.2 534392.8  
## Apr 2019 443548 406810.1 480285.9 387362.2 499733.8  
## May 2019 464162 427424.1 500899.9 407976.2 520347.8  
## Jun 2019 457944 421206.1 494681.9 401758.2 514129.8  
## Jul 2019 440436 403698.1 477173.9 384250.2 496621.8  
## Aug 2019 366272 329534.1 403009.9 310086.2 422457.8

checkresiduals(snaivem)



##   
## Ljung-Box test  
##   
## data: Residuals from Seasonal naive method  
## Q\* = 35.426, df = 19, p-value = 0.0124  
##   
## Model df: 0. Total lags used: 19

accuracy(snaivem, data\_ts\_test)

## ME RMSE MAE MPE MAPE MASE ACF1  
## Training set 25337.46 28666.73 25689.63 7.101745 7.207375 1.0000000 0.2695124  
## Test set 14263.38 22148.43 16960.88 3.053421 3.648407 0.6602226 -0.5427745  
## Theil's U  
## Training set NA  
## Test set 0.4792835

# Etude du Modèle de Buys-Ballot

## Modèle

<https://mpra.ub.uni-muenchen.de/77718/1/MPRA_paper_77718.pdf> page 175

L’approche de BUYS-BALLOT consiste à introduire des variables indicatrices correspondant à chaque saison définit par le cycle d’observation. Pour les données trimestrielles, on intègre 4 variables indicatrices. Et pour les données mensuelles, on intègre 12 variables indicatrices.

Le modèle doit alors être estimé (sans constante) avec ces variables indicatrices.

## Prédiction des valeurs de 2019

Préparation des données.

Annees=as.numeric(time(data\_ts\_train))  
ts\_DataFrame =data.frame(trafic=data\_ts\_train,X=as.numeric(Annees))

Création du modèle

Regression <- lm(trafic~X,data = ts\_DataFrame)

La tendance Prédiction sur les données futurs.

tendance=predict(Regression)  
  
AnneeMoisNumericFutur=seq(max(Annees)+1/12,length=8,by=1/12) #les 10 prochains mois  
  
tendance2=predict(Regression, newdata=data.frame(X=AnneeMoisNumericFutur))

ts\_DataFrame$trafic\_residual <- residuals(Regression)

Définissons le mois

ts\_DataFrame$mois <- round(ts\_DataFrame$X - trunc(ts\_DataFrame$X),digit=4)

Création du 2nd modèle avec les mois

Regression2 =lm(trafic\_residual~0+as.factor(mois),data=ts\_DataFrame)

Prédiction de la saisonnalité

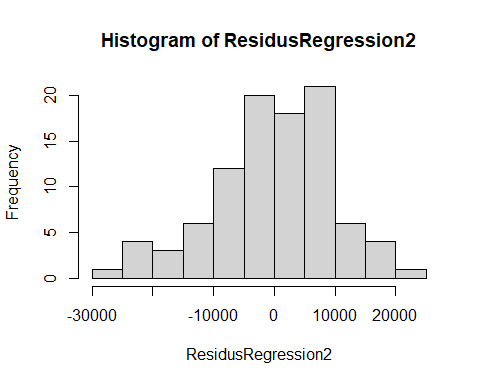
prediction2 =predict(Regression2)

Prédiction sur les mois

MoisNumeric= round(AnneeMoisNumericFutur - trunc(AnneeMoisNumericFutur  
 ),4)  
Prediction3 =predict( Regression2, newdata= data.frame(mois=MoisNumeric))

Calculons une région de confiance avec l’erreur d’ajustement

ResidusRegression2=residuals(Regression2)  
hist(ResidusRegression2)



1.96\*sqrt(var(ResidusRegression2))

## [1] 19226.16

## Auto corrélation de la série temporelle

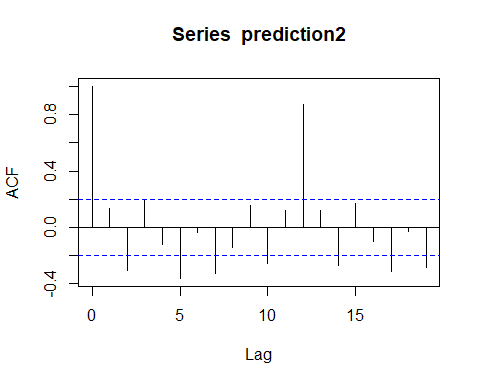
L’autocorrélation de notre série temporelle correspond à la corrélation entre une mesure du trafic et les mesures précédentes ou les mesures suivantes .

L’auto covariance d’une variable de moyenne et d’écart type à un décalage est donné par la formule

On en déduit l’autocorrélation correspondante :

Affichons les autocorrélations de la séries grâce à un corrélogramme

ACF\_Sur\_Valeurs\_Predites <- acf(prediction2)



Il est normal que la série soit autocorrélé totalement à elle avec un décalage nulle.

On observe une corrélation forte (0.87) avec un décalage (lag) de 12, cela correspond bien à une saisonnalité annuelle.

print(data.frame(ACF\_Sur\_Valeurs\_Predites$lag,ACF\_Sur\_Valeurs\_Predites$acf)[1:13,])

## ACF\_Sur\_Valeurs\_Predites.lag ACF\_Sur\_Valeurs\_Predites.acf  
## 1 0 1.00000000  
## 2 1 0.13556225  
## 3 2 -0.30353869  
## 4 3 0.19244891  
## 5 4 -0.11749420  
## 6 5 -0.36058135  
## 7 6 -0.03501791  
## 8 7 -0.32578902  
## 9 8 -0.14275858  
## 10 9 0.15718886  
## 11 10 -0.25819898  
## 12 11 0.12067802  
## 13 12 0.87497657

Recalculons la valeur d’auto-corrélation obtenu en appliquant la formule.

Observons l’application de la formule, en choisissant un décalage de 12

#Constantes  
Nombre\_Observations=96  
decalage=12  
  
#Estimations  
moyenneMu=mean(prediction2)  
sdSigma=sd(prediction2)  
  
  
Serie1=prediction2[(decalage+1): 96 ]  
Serie2=prediction2[ 1 :(96-decalage)]  
  
GammaDecalage12=mean((Serie1-moyenneMu)\*(Serie2-moyenneMu))\*((Nombre\_Observations-decalage)/(Nombre\_Observations))  
  
RhoDecalage12=GammaDecalage12/(sdSigma^2)  
RhoDecalage12

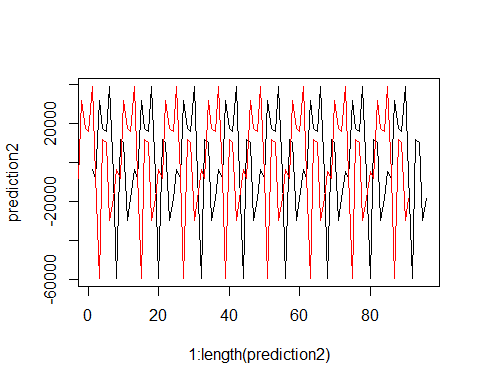
## [1] 0.8658622

Le résultat obtenu est correct. L’auto corrélation avec un décalage de 12 est donc très forte.

De plus cette auto corrélation étant positive, cela indique une tendance croissante.

la deuxième plus forte corrélation est obsersé avec un décalage de 5, observons cela graphiquement

plot ( 1:length(prediction2), prediction2,type="l")  
points((1:length(prediction2))-5,prediction2,type="l",col="red")



Cette corrélation est peu pertinente.

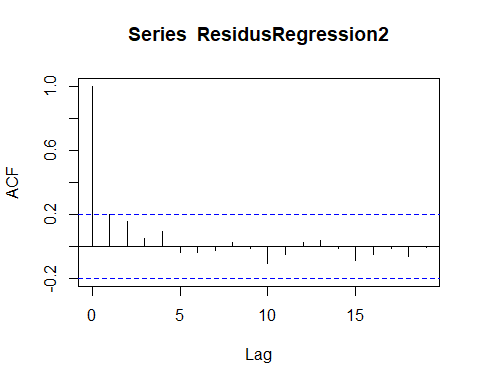
print(data.frame(ACF\_Sur\_Valeurs\_Predites$lag,ACF\_Sur\_Valeurs\_Predites$acf)[1:13,])

## ACF\_Sur\_Valeurs\_Predites.lag ACF\_Sur\_Valeurs\_Predites.acf  
## 1 0 1.00000000  
## 2 1 0.13556225  
## 3 2 -0.30353869  
## 4 3 0.19244891  
## 5 4 -0.11749420  
## 6 5 -0.36058135  
## 7 6 -0.03501791  
## 8 7 -0.32578902  
## 9 8 -0.14275858  
## 10 9 0.15718886  
## 11 10 -0.25819898  
## 12 11 0.12067802  
## 13 12 0.87497657

Après avoir étudier les auto-corrélations sur l’ensemble du modèle, Observons les auto-corrélations sur les résidus du modèle de Buys-Ballot.

* Texte pour dire que les accidents ne doivent pas être corrélés \*

plot(acf(ResidusRegression2))

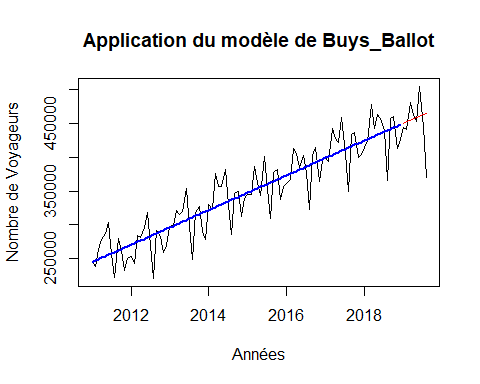


Pour notre modèle, il n’y a aucune auto-corrélation significative. (symbolisé par la ligne bleu)

## Comparaison des prédictions et des valeurs réelles

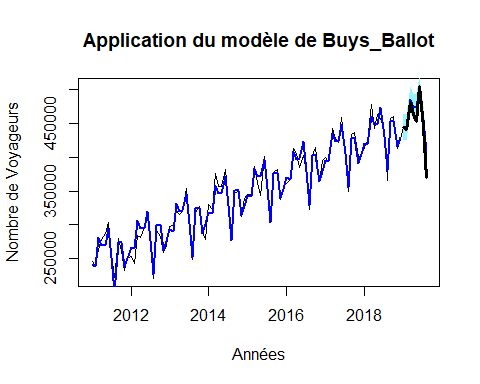
Affichage de la tendance

Buys\_ballot\_plot\_tendance <- plot(data\_ts,  
 main = "Application du modèle de Buys\_Ballot",  
 xlab = "Années",  
 ylab = "Nombre de Voyageurs")   
  
#droite de tendance  
lines(Annees,tendance,col="blue",lwd=2)   
  
#prédiction de la tendance futur  
lines(AnneeMoisNumericFutur,tendance2,col="red")



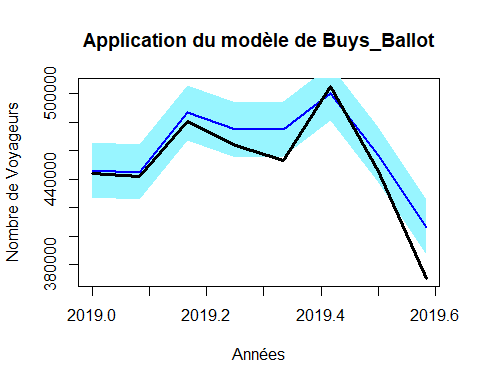
Affichage du modèle de Buys Ballot

Buys\_ballot\_plot <- plot(data\_ts,  
 main = "Application du modèle de Buys\_Ballot",  
 xlab = "Années",  
 ylab = "Nombre de Voyageurs")   
  
  
  
#prédiction du modèle de Buys ballot  
lines(Annees,tendance+prediction2,col="blue",lwd=2)  
  
#Interval de confiance  
 polygon(c(AnneeMoisNumericFutur,rev(AnneeMoisNumericFutur)),  
 c(tendance2+Prediction3-1.96\*sqrt(var(ResidusRegression2)),  
 rev(tendance2+Prediction3+1.96\*sqrt(var(ResidusRegression2)))),  
 col="cadetblue1",border=NA)  
   
 #Prediction des valeurs  
 lines(AnneeMoisNumericFutur,tendance2+Prediction3,col="blue",lwd=2)  
   
   
 lines(data\_ts\_test,col="black",lwd=3)



Affichage de la prédiction sur les 8 mois de 2020

Buys\_ballot\_plot <- plot(data\_ts\_test,  
 main = "Application du modèle de Buys\_Ballot",  
 xlab = "Années",  
 ylab = "Nombre de Voyageurs")   
  
  
  
#prédiction du modèle de Buys ballot  
lines(Annees,tendance+prediction2,col="blue",lwd=2)  
  
#Interval de confiance  
 polygon(c(AnneeMoisNumericFutur,rev(AnneeMoisNumericFutur)),  
 c(tendance2+Prediction3-1.96\*sqrt(var(ResidusRegression2)),  
 rev(tendance2+Prediction3+1.96\*sqrt(var(ResidusRegression2)))),  
 col="cadetblue1",border=NA)  
   
 #Prediction des valeurs  
 lines(AnneeMoisNumericFutur,tendance2+Prediction3,col="blue",lwd=2)  
   
   
 lines(data\_ts\_test,col="black",lwd=3)

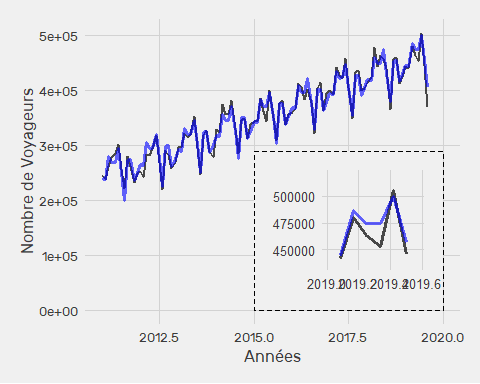


Préparation DataFrame pour affichage ggplot

DataAffichageGGplot = as.data.frame(data\_ts)  
DataAffichageGGplot$Annees = c(Annees, AnneeMoisNumericFutur)  
DataAffichageGGplot$AnneesRound = round(DataAffichageGGplot$Annees)  
DataAffichageGGplot$PredictionTendanceBuysBalot = c(tendance ,tendance2)  
DataAffichageGGplot$BuysBalotModele = c(tendance+prediction2,tendance2+Prediction3 )

Reproduisons les graphiques avec ggplot2 pour un résultat plus professsionnel.

Affichage\_Prediction(DataAffichageGGplot, DataAffichageGGplot$BuysBalotModele)



Nous avons réussi à ajuster une droite de régression. on remarque que la prédiction semble bien correspondre à la réalité si on fait abstraction du dernier mois où le nombre de voyageurs a bien plus chuté que la prédiction du modèle de Buys-Balot.

Comparons avec un ajustement local réalisé par lissage moyennes mobiles.

## Comparaison avec les valeurs observées

# Lissage moyenne mobile

## Définition

Mettre belle formule en latex ici

## Choix Moyenne mobiles

## Conservation & Annulation

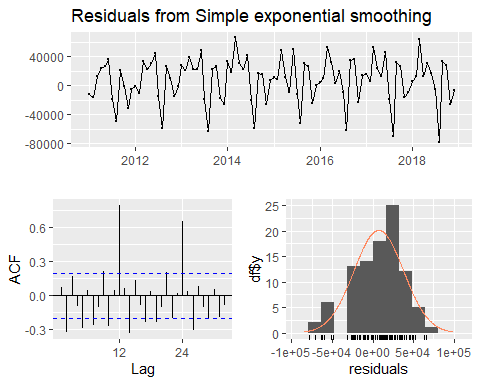
# Lissage exponentielle

## Lissage simple

fcst\_se <- ses(data\_ts\_train, h = 8)  
print(summary(fcst\_se))

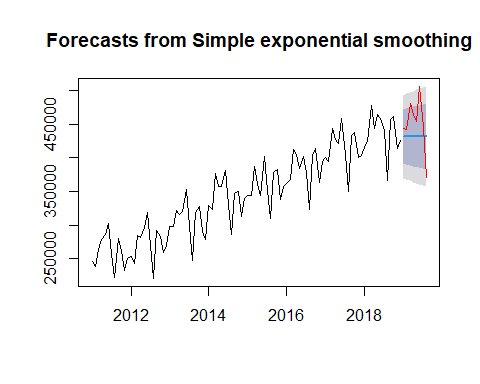
##   
## Forecast method: Simple exponential smoothing  
##   
## Model Information:  
## Simple exponential smoothing   
##   
## Call:  
## ses(y = data\_ts\_train, h = 8)   
##   
## Smoothing parameters:  
## alpha = 0.2559   
##   
## Initial states:  
## l = 258126.0245   
##   
## sigma: 31480.96  
##   
## AIC AICc BIC   
## 2430.727 2430.988 2438.420   
##   
## Error measures:  
## ME RMSE MAE MPE MAPE MASE ACF1  
## Training set 7057.127 31151.31 25752.89 1.326234 7.684321 1.002462 0.0711143  
##   
## Forecasts:  
## Point Forecast Lo 80 Hi 80 Lo 95 Hi 95  
## Jan 2019 431488.3 391143.9 471832.8 369786.8 493189.9  
## Feb 2019 431488.3 389843.9 473132.7 367798.7 495178.0  
## Mar 2019 431488.3 388583.3 474393.3 365870.8 497105.9  
## Apr 2019 431488.3 387358.8 475617.9 363998.0 498978.7  
## May 2019 431488.3 386167.3 476809.4 362175.7 500800.9  
## Jun 2019 431488.3 385006.3 477970.4 360400.2 502576.5  
## Jul 2019 431488.3 383873.6 479103.1 358667.9 504308.8  
## Aug 2019 431488.3 382767.3 480209.4 356975.9 506000.8

checkresiduals(fcst\_se)



##   
## Ljung-Box test  
##   
## data: Residuals from Simple exponential smoothing  
## Q\* = 144.66, df = 17, p-value < 2.2e-16  
##   
## Model df: 2. Total lags used: 19

plot(fcst\_se)  
lines(data\_ts\_test, col="red")



df\_se = as.data.frame(fcst\_se)  
predict\_value\_se <- df\_se$`Point Forecast`  
MAPE(predict\_value\_se, data\_ts\_test)\*100

## [1] 794.5782

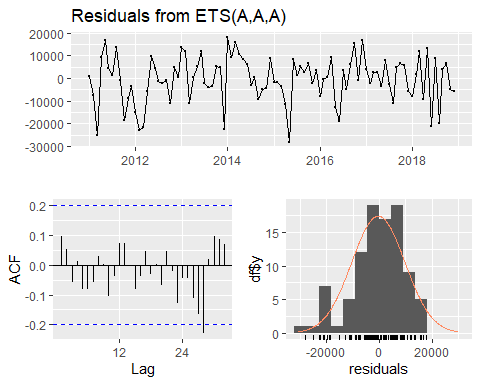
## Optimisation du modèle

Fit Exponential Smoothing model -> trouve le meilleur lissage expo

fit\_ets <- ets(data\_ts\_train)   
print(summary(fit\_ets))

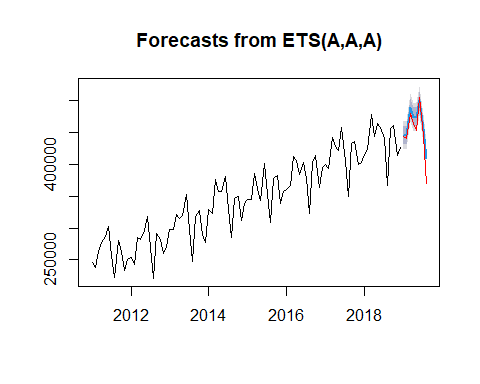
## ETS(A,A,A)   
##   
## Call:  
## ets(y = data\_ts\_train)   
##   
## Smoothing parameters:  
## alpha = 0.1568   
## beta = 1e-04   
## gamma = 1e-04   
##   
## Initial states:  
## l = 248267.1099   
## b = 2163.3982   
## s = -17928.3 -29535.73 9295.935 11005.81 -57117.85 -7708.17  
## 38272.64 14592.34 16899.53 34763.15 -7344.204 -5195.15  
##   
## sigma: 11014.45  
##   
## AIC AICc BIC   
## 2241.611 2249.458 2285.205   
##   
## Training set error measures:  
## ME RMSE MAE MPE MAPE MASE  
## Training set -458.6799 10054.77 7831.554 -0.2623253 2.371375 0.3048527  
## ACF1  
## Training set 0.09626331

checkresiduals(fit\_ets)



##   
## Ljung-Box test  
##   
## data: Residuals from ETS(A,A,A)  
## Q\* = 7.1794, df = 3, p-value = 0.06639  
##   
## Model df: 16. Total lags used: 19

fcst\_ets <- forecast(fit\_ets, h=8)  
plot(fcst\_ets)  
lines(data\_ts\_test, col="red")



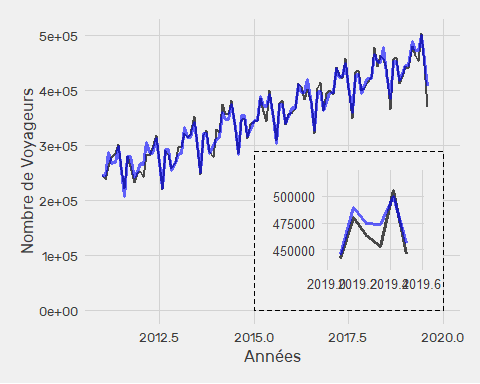
df\_ets = as.data.frame(fcst\_ets)  
predict\_value\_ets = df\_ets$`Point Forecast`  
MAPE(predict\_value\_ets, data\_ts\_test)\*100

## [1] 283.9718

Affichage GGplot

DataAffichageGGplot$ModeleLissageExponentielle <- c(fcst\_ets$fitted ,predict\_value\_ets )  
  
Affichage\_Prediction(DataAffichageGGplot, DataAffichageGGplot$ModeleLissageExponentielle)

## Warning: Removed 98 row(s) containing missing values (geom\_path).  
## Removed 98 row(s) containing missing values (geom\_path).



## Modèle ARIMA

A FAIRE

## Modèle ARIMA / SAMIRA Automatique

ARIMA : AutoRegressive Integrated Moving Average

Le modèle ARIMA est une combinaison du modèle ARMA combiné à une différentiation (le Integrated)

Différentiation = rétirer les tendances -> tendance linéaire : une différenciation -> tendance quadradique : deux différenciations

Le modèle SARIMA est une combinaison du modèle ARIMA qui prend en compte la composante saisonniaire.

auto.arima prend en compte les saisonnalités, comme on peut le voir dans le modèle selectionné : (0,1,1)(0,1,1)[12]

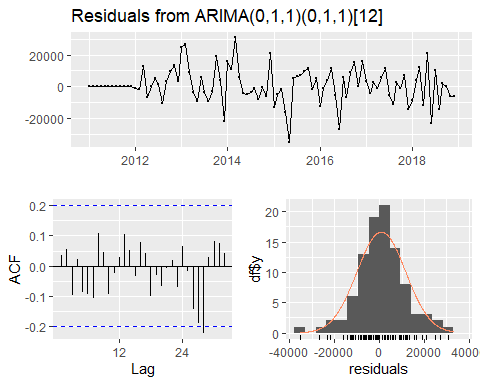
# retourne les meilleurs paramètres   
# d=1 enleve la tendance  
# D=1 enleve la saisonnalité   
# => avoir des données stationnaires  
# trace : voir les résultats  
fit\_arima <- auto.arima(data\_ts\_train, d=1, D=1, stepwise = FALSE, approximation = FALSE, trace=TRUE)

##   
## ARIMA(0,1,0)(0,1,0)[12] : 1846.398  
## ARIMA(0,1,0)(0,1,1)[12] : 1833.134  
## ARIMA(0,1,0)(0,1,2)[12] : 1835.211  
## ARIMA(0,1,0)(1,1,0)[12] : 1833.056  
## ARIMA(0,1,0)(1,1,1)[12] : 1835.09  
## ARIMA(0,1,0)(1,1,2)[12] : Inf  
## ARIMA(0,1,0)(2,1,0)[12] : 1835.207  
## ARIMA(0,1,0)(2,1,1)[12] : 1837.012  
## ARIMA(0,1,0)(2,1,2)[12] : 1836.461  
## ARIMA(0,1,1)(0,1,0)[12] : 1814.951  
## ARIMA(0,1,1)(0,1,1)[12] : 1801.155  
## ARIMA(0,1,1)(0,1,2)[12] : 1803.362  
## ARIMA(0,1,1)(1,1,0)[12] : 1803.592  
## ARIMA(0,1,1)(1,1,1)[12] : 1803.361  
## ARIMA(0,1,1)(1,1,2)[12] : Inf  
## ARIMA(0,1,1)(2,1,0)[12] : 1805.004  
## ARIMA(0,1,1)(2,1,1)[12] : 1805.397  
## ARIMA(0,1,1)(2,1,2)[12] : Inf  
## ARIMA(0,1,2)(0,1,0)[12] : 1816.915  
## ARIMA(0,1,2)(0,1,1)[12] : 1803.033  
## ARIMA(0,1,2)(0,1,2)[12] : 1805.296  
## ARIMA(0,1,2)(1,1,0)[12] : 1805.702  
## ARIMA(0,1,2)(1,1,1)[12] : 1805.295  
## ARIMA(0,1,2)(1,1,2)[12] : Inf  
## ARIMA(0,1,2)(2,1,0)[12] : 1807.026  
## ARIMA(0,1,2)(2,1,1)[12] : 1807.441  
## ARIMA(0,1,3)(0,1,0)[12] : 1817.787  
## ARIMA(0,1,3)(0,1,1)[12] : Inf  
## ARIMA(0,1,3)(0,1,2)[12] : Inf  
## ARIMA(0,1,3)(1,1,0)[12] : Inf  
## ARIMA(0,1,3)(1,1,1)[12] : Inf  
## ARIMA(0,1,3)(2,1,0)[12] : Inf  
## ARIMA(0,1,4)(0,1,0)[12] : 1820.052  
## ARIMA(0,1,4)(0,1,1)[12] : Inf  
## ARIMA(0,1,4)(1,1,0)[12] : Inf  
## ARIMA(0,1,5)(0,1,0)[12] : Inf  
## ARIMA(1,1,0)(0,1,0)[12] : 1825.579  
## ARIMA(1,1,0)(0,1,1)[12] : 1812.512  
## ARIMA(1,1,0)(0,1,2)[12] : 1814.657  
## ARIMA(1,1,0)(1,1,0)[12] : 1813.2  
## ARIMA(1,1,0)(1,1,1)[12] : 1814.614  
## ARIMA(1,1,0)(1,1,2)[12] : 1816.192  
## ARIMA(1,1,0)(2,1,0)[12] : 1815.227  
## ARIMA(1,1,0)(2,1,1)[12] : Inf  
## ARIMA(1,1,0)(2,1,2)[12] : 1817.796  
## ARIMA(1,1,1)(0,1,0)[12] : 1816.841  
## ARIMA(1,1,1)(0,1,1)[12] : 1802.853  
## ARIMA(1,1,1)(0,1,2)[12] : 1805.117  
## ARIMA(1,1,1)(1,1,0)[12] : 1805.653  
## ARIMA(1,1,1)(1,1,1)[12] : Inf  
## ARIMA(1,1,1)(1,1,2)[12] : Inf  
## ARIMA(1,1,1)(2,1,0)[12] : Inf  
## ARIMA(1,1,1)(2,1,1)[12] : Inf  
## ARIMA(1,1,2)(0,1,0)[12] : 1819.234  
## ARIMA(1,1,2)(0,1,1)[12] : 1805.22  
## ARIMA(1,1,2)(0,1,2)[12] : 1807.539  
## ARIMA(1,1,2)(1,1,0)[12] : 1807.381  
## ARIMA(1,1,2)(1,1,1)[12] : 1807.538  
## ARIMA(1,1,2)(2,1,0)[12] : 1808.925  
## ARIMA(1,1,3)(0,1,0)[12] : 1820.05  
## ARIMA(1,1,3)(0,1,1)[12] : 1806.055  
## ARIMA(1,1,3)(1,1,0)[12] : 1808.732  
## ARIMA(1,1,4)(0,1,0)[12] : Inf  
## ARIMA(2,1,0)(0,1,0)[12] : 1824.435  
## ARIMA(2,1,0)(0,1,1)[12] : 1811.07  
## ARIMA(2,1,0)(0,1,2)[12] : 1813.287  
## ARIMA(2,1,0)(1,1,0)[12] : 1811.619  
## ARIMA(2,1,0)(1,1,1)[12] : 1813.247  
## ARIMA(2,1,0)(1,1,2)[12] : Inf  
## ARIMA(2,1,0)(2,1,0)[12] : 1813.821  
## ARIMA(2,1,0)(2,1,1)[12] : 1815.872  
## ARIMA(2,1,1)(0,1,0)[12] : Inf  
## ARIMA(2,1,1)(0,1,1)[12] : Inf  
## ARIMA(2,1,1)(0,1,2)[12] : Inf  
## ARIMA(2,1,1)(1,1,0)[12] : Inf  
## ARIMA(2,1,1)(1,1,1)[12] : Inf  
## ARIMA(2,1,1)(2,1,0)[12] : Inf  
## ARIMA(2,1,2)(0,1,0)[12] : Inf  
## ARIMA(2,1,2)(0,1,1)[12] : Inf  
## ARIMA(2,1,2)(1,1,0)[12] : Inf  
## ARIMA(2,1,3)(0,1,0)[12] : Inf  
## ARIMA(3,1,0)(0,1,0)[12] : 1823.646  
## ARIMA(3,1,0)(0,1,1)[12] : 1808.49  
## ARIMA(3,1,0)(0,1,2)[12] : 1810.542  
## ARIMA(3,1,0)(1,1,0)[12] : 1808.594  
## ARIMA(3,1,0)(1,1,1)[12] : 1810.321  
## ARIMA(3,1,0)(2,1,0)[12] : 1810.708  
## ARIMA(3,1,1)(0,1,0)[12] : Inf  
## ARIMA(3,1,1)(0,1,1)[12] : Inf  
## ARIMA(3,1,1)(1,1,0)[12] : Inf  
## ARIMA(3,1,2)(0,1,0)[12] : Inf  
## ARIMA(4,1,0)(0,1,0)[12] : 1823.996  
## ARIMA(4,1,0)(0,1,1)[12] : 1810.199  
## ARIMA(4,1,0)(1,1,0)[12] : 1810.845  
## ARIMA(4,1,1)(0,1,0)[12] : Inf  
## ARIMA(5,1,0)(0,1,0)[12] : 1825.055  
##   
##   
##   
## Best model: ARIMA(0,1,1)(0,1,1)[12]

print(summary(fit\_arima))

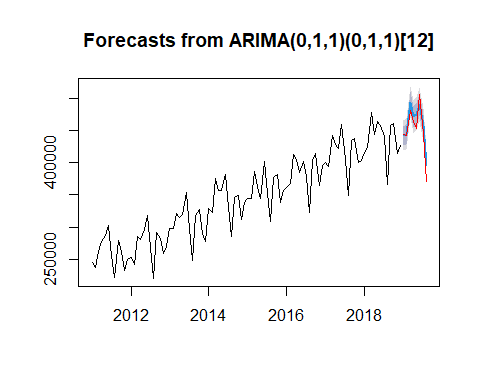
## Series: data\_ts\_train   
## ARIMA(0,1,1)(0,1,1)[12]   
##   
## Coefficients:  
## ma1 sma1  
## -0.7675 -0.5465  
## s.e. 0.0977 0.1295  
##   
## sigma^2 = 138827719: log likelihood = -897.43  
## AIC=1800.85 AICc=1801.16 BIC=1808.11  
##   
## Training set error measures:  
## ME RMSE MAE MPE MAPE MASE ACF1  
## Training set 805.2378 10822.93 7747.841 0.2158443 2.213481 0.3015941 0.03453594

checkresiduals(fit\_arima)



##   
## Ljung-Box test  
##   
## data: Residuals from ARIMA(0,1,1)(0,1,1)[12]  
## Q\* = 10.898, df = 17, p-value = 0.8618  
##   
## Model df: 2. Total lags used: 19

fcst\_arima <- forecast(fit\_arima, h=8)  
plot(fcst\_arima)  
lines(data\_ts\_test, col='red')



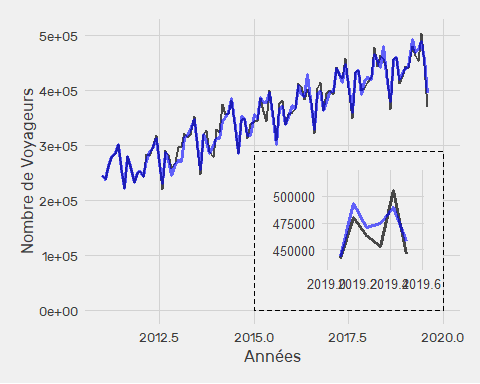
df\_arima = as.data.frame(fcst\_arima)  
predict\_value\_arima = df\_arima$`Point Forecast`  
MAPE(predict\_value\_arima, data\_ts\_test)\*100

## [1] 272.507

Affichage GGplot

DataAffichageGGplot$ModeleArima <- c(fit\_arima$fitted ,predict\_value\_arima )  
Affichage\_Prediction(DataAffichageGGplot, DataAffichageGGplot$ModeleArima)

## Warning: Removed 98 row(s) containing missing values (geom\_path).  
## Removed 98 row(s) containing missing values (geom\_path).



## PROPHET

Préparation données

library(zoo)

##   
## Attachement du package : 'zoo'

## Les objets suivants sont masqués depuis 'package:base':  
##   
## as.Date, as.Date.numeric

data\_train$ds <- as.Date( as.yearmon(time(data\_ts\_train)))

library(prophet)

## Le chargement a nécessité le package : Rcpp

## Le chargement a nécessité le package : rlang

model\_prophet <- prophet(data\_train)

## Disabling weekly seasonality. Run prophet with weekly.seasonality=TRUE to override this.

## Disabling daily seasonality. Run prophet with daily.seasonality=TRUE to override this.

forecast\_prophet <- make\_future\_dataframe(model\_prophet, periods = 8, freq = 'month')  
AAPLfc <- predict(model\_prophet, forecast\_prophet)  
tail(AAPLfc[c("ds", "yhat", "yhat\_lower", "yhat")])

## ds yhat yhat\_lower yhat.1  
## 99 2019-03-01 493929.1 481854.8 493929.1  
## 100 2019-04-01 477074.4 466088.6 477074.4  
## 101 2019-05-01 475880.5 464655.9 475880.5  
## 102 2019-06-01 503278.4 491246.8 503278.4  
## 103 2019-07-01 457437.7 444804.3 457437.7  
## 104 2019-08-01 412673.2 401441.3 412673.2

dyplot.prophet(model\_prophet, AAPLfc)

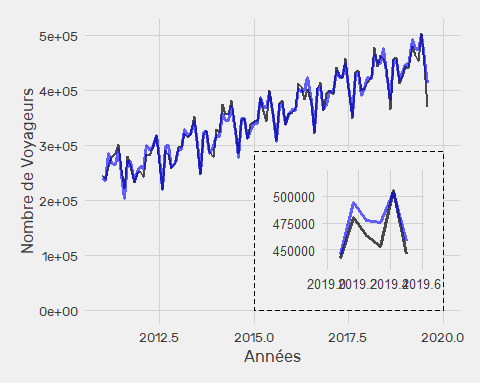
## Warning: `select\_()` was deprecated in dplyr 0.7.0.  
## Please use `select()` instead.  
## This warning is displayed once every 8 hours.  
## Call `lifecycle::last\_lifecycle\_warnings()` to see where this warning was generated.

# data\_pp <- subset(AAPLfc, select=c("yhat"))  
# data\_pp\_ts <- ts(data\_tttt, start=2011, frequency=12)  
# data\_pp\_ts\_w <- window(data\_pp\_ts, start= c(2019,1), end = c(2019,8))  
#   
#   
#   
# plot(data\_ts)  
# lines(data\_pp\_ts\_w, col="red")  
#   
# MAPE(data\_pp\_ts\_w, data\_ts\_test)\*100

Affichage GGplot

DataAffichageGGplot$ProphetModele <- AAPLfc$yhat  
Affichage\_Prediction(DataAffichageGGplot, DataAffichageGGplot$ProphetModele)

## Warning: Removed 98 row(s) containing missing values (geom\_path).  
## Removed 98 row(s) containing missing values (geom\_path).



## LSTM

scale\_factors <- c(mean(data$y), sd(data$y))  
scaled\_train <- data %>%  
 dplyr::select(y) %>%  
 dplyr::mutate(y = (y - scale\_factors[1]) / scale\_factors[2])  
scaled\_train

## # A tibble: 104 × 1  
## y  
## <dbl>  
## 1 -1.56   
## 2 -1.67   
## 3 -1.31   
## 4 -1.10   
## 5 -0.972  
## 6 -0.738  
## 7 -1.40   
## 8 -1.90   
## 9 -1.07   
## 10 -1.32   
## # … with 94 more rows

prediction <- 12  
lag <- prediction

On veut prendre l’année précedente pour apprendre > lag de 12, en réalité ca fait 12 - 1 pour avoir à chaque prédiction basée sur 12 valeurs

puis en transforme en array 3D car le modèle LSTM prendre un tensor de format 3D [samples, timesteps, features] samples : nbr d’observation par batchs timesteps : lag features : nbr de valeur predites

scaled\_train <- as.matrix(scaled\_train)  
   
# we lag the data 11 times and arrange that into columns  
x\_train\_data <- t(sapply(  
 1:(length(scaled\_train) - lag - prediction + 1),  
 function(x) scaled\_train[x:(x + lag - 1), 1]  
 ))  
   
# now we transform it into 3D form  
x\_train\_arr <- array(  
 data = as.numeric(unlist(x\_train\_data)),  
 dim = c(  
 nrow(x\_train\_data),  
 lag,  
 1  
 )  
)  
  
#(x\_train\_data)  
#length(x\_train\_arr)  
#head(x\_train\_arr)

y\_train\_data <- t(sapply(  
 (1 + lag):(length(scaled\_train) - prediction + 1),  
 function(x) scaled\_train[x:(x + prediction - 1)]  
))  
  
y\_train\_arr <- array(  
 data = as.numeric(unlist(y\_train\_data)),  
 dim = c(  
 nrow(y\_train\_data),  
 prediction,  
 1  
 )  
)  
  
#head(y\_train\_data)  
#head(y\_train\_arr)

x\_test <- data$y[(nrow(scaled\_train) - prediction + 1):nrow(scaled\_train)]  
  
x\_test\_scaled <- (x\_test - scale\_factors[1]) / scale\_factors[2]  
  
x\_pred\_arr <- array(  
 data = x\_test\_scaled,  
 dim = c(  
 1,  
 lag,  
 1  
 )  
)

lstm\_model <- keras\_model\_sequential()

## Loaded Tensorflow version 2.8.0

lstm\_model %>%  
 layer\_lstm(units = 50, # size of the layer  
 batch\_input\_shape = c(1, 12, 1), # batch size, timesteps, features  
 return\_sequences = TRUE,  
 stateful = TRUE) %>%  
 # fraction of the units to drop for the linear transformation of the inputs  
 layer\_dropout(rate = 0.5) %>%  
 layer\_lstm(units = 50,  
 return\_sequences = TRUE,  
 stateful = TRUE) %>%  
 layer\_dropout(rate = 0.5) %>%  
 time\_distributed(keras::layer\_dense(units = 1))  
  
lstm\_model %>%  
 compile(loss = 'mae', optimizer = 'adam', metrics = 'accuracy')  
  
summary(lstm\_model)

## Model: "sequential"  
## \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
## Layer (type) Output Shape Param #   
## ================================================================================  
## lstm\_1 (LSTM) (1, 12, 50) 10400   
## dropout\_1 (Dropout) (1, 12, 50) 0   
## lstm (LSTM) (1, 12, 50) 20200   
## dropout (Dropout) (1, 12, 50) 0   
## time\_distributed (TimeDistributed) (1, 12, 1) 51   
## ================================================================================  
## Total params: 30,651  
## Trainable params: 30,651  
## Non-trainable params: 0  
## \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

lstm\_model %>% fit(  
 x = x\_train\_arr,  
 y = y\_train\_arr,  
 batch\_size = 1,  
 epochs = 20,  
 verbose = 1,  
 shuffle = FALSE  
)

lstm\_forecast <- lstm\_model %>%  
 predict(x\_pred\_arr, batch\_size = 1) %>%  
 .[, , 1]  
   
# rescale en format basique  
lstm\_forecast <- lstm\_forecast \* scale\_factors[2] + scale\_factors[1]  
lstm\_forecast

## [1] 458724.6 462540.5 426429.5 443280.7 454624.5 452532.1 475679.8 467106.2  
## [9] 459594.9 486293.9 452061.4 377945.1

X résultats / prédictions par input donc > transforme pour une seule prédiciton

fitted <- predict(lstm\_model, x\_train\_arr, batch\_size = 1) %>%  
 .[, , 1]  
  
if (dim(fitted)[2] > 1) {  
   
 fit <- c(fitted[, 1], fitted[dim(fitted)[1], 2:dim(fitted)[2]])  
} else {  
 fit <- fitted[, 1]  
}  
  
# rescale final de nos données  
fitted <- fit \* scale\_factors[2] + scale\_factors[1]  
fitted

## [1] 266327.6 287497.6 307396.5 318650.7 326137.3 337562.4 300599.5 275437.5  
## [9] 316840.5 305088.6 282018.3 294295.3 296345.3 288891.1 323131.7 319457.7  
## [17] 331266.5 348360.7 306886.9 275227.4 324860.9 318839.8 299749.6 307930.9  
## [25] 330212.1 331093.1 351729.5 346061.0 353032.7 379796.8 329120.1 291576.0  
## [33] 350243.2 361374.7 324996.6 314069.9 362138.1 356666.7 400407.3 386910.8  
## [41] 388161.5 408708.3 363605.7 321708.2 380167.5 386963.1 346512.1 372889.6  
## [49] 381069.4 379518.8 416249.4 396415.3 376955.7 428310.7 382265.9 336048.5  
## [57] 408582.8 416928.4 366661.8 385637.7 394520.0 396067.2 436261.8 431088.0  
## [65] 409529.0 424832.8 399778.6 343240.9 426672.3 441126.6 381893.2 415211.4  
## [73] 424224.4 418044.9 455747.8 445130.4 438641.2 464603.6 419369.8 360070.5  
## [81] 446435.3 449536.9 415210.4 427506.5 436018.9 441899.4 473945.4 454682.5  
## [89] 467525.7 463348.2 450287.6 376817.8

fitted <- c(rep(NA, lag), fitted)  
fitted

## [1] NA NA NA NA NA NA NA NA  
## [9] NA NA NA NA 266327.6 287497.6 307396.5 318650.7  
## [17] 326137.3 337562.4 300599.5 275437.5 316840.5 305088.6 282018.3 294295.3  
## [25] 296345.3 288891.1 323131.7 319457.7 331266.5 348360.7 306886.9 275227.4  
## [33] 324860.9 318839.8 299749.6 307930.9 330212.1 331093.1 351729.5 346061.0  
## [41] 353032.7 379796.8 329120.1 291576.0 350243.2 361374.7 324996.6 314069.9  
## [49] 362138.1 356666.7 400407.3 386910.8 388161.5 408708.3 363605.7 321708.2  
## [57] 380167.5 386963.1 346512.1 372889.6 381069.4 379518.8 416249.4 396415.3  
## [65] 376955.7 428310.7 382265.9 336048.5 408582.8 416928.4 366661.8 385637.7  
## [73] 394520.0 396067.2 436261.8 431088.0 409529.0 424832.8 399778.6 343240.9  
## [81] 426672.3 441126.6 381893.2 415211.4 424224.4 418044.9 455747.8 445130.4  
## [89] 438641.2 464603.6 419369.8 360070.5 446435.3 449536.9 415210.4 427506.5  
## [97] 436018.9 441899.4 473945.4 454682.5 467525.7 463348.2 450287.6 376817.8

length(fitted)

## [1] 104

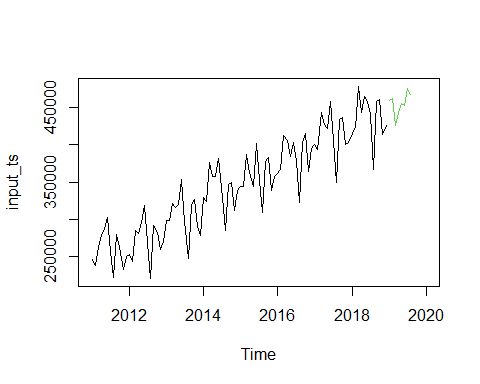
lstm\_forecast <- ts(lstm\_forecast,  
 start = c(2019, 1),  
 end = c(2019, 12),  
 frequency = 12  
)  
  
lstm\_forecast\_display <- window(lstm\_forecast, start= c(2019,1), end = c(2019,8))  
  
input\_ts <- ts(data$y,   
 start = c(2011, 1),   
 end = c(2018, 12),   
 frequency = 12)  
  
  
lstm\_forecast\_display

## Jan Feb Mar Apr May Jun Jul Aug  
## 2019 458724.6 462540.5 426429.5 443280.7 454624.5 452532.1 475679.8 467106.2

data\_ts\_test

## Jan Feb Mar Apr May Jun Jul Aug  
## 2019 443700 441499 480649 463680 453372 505190 445332 370211

plot(input\_ts, xlim=c(2011,2020))  
#lines(data\_ts\_test)  
lines(lstm\_forecast\_display, col=3)

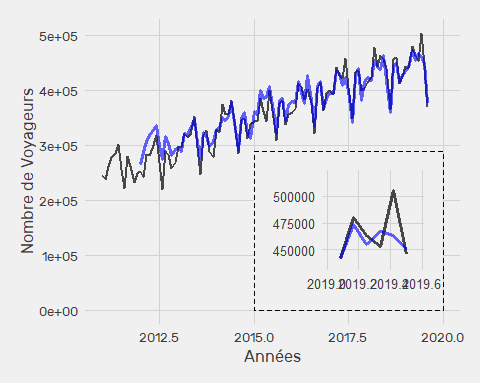


Affichage GGplot

DataAffichageGGplot$LSTM\_Modele <- fitted  
Affichage\_Prediction(DataAffichageGGplot, DataAffichageGGplot$LSTM\_Modele)

## Warning: Removed 98 row(s) containing missing values (geom\_path).  
## Removed 98 row(s) containing missing values (geom\_path).

## Warning: Removed 12 row(s) containing missing values (geom\_path).



# Comparaison Modèle

## Choix des indicateurs

Nous devons pouvoir comparer nos modèles entre eux, nous souhaitons pouvoir observer leurs performances globale, ainsi que spécifiquement leurs performances sur le jeu d’entraînement et le jeu de test (les 8 mois de 2019 à prédire).

Nous utiliserons une mesure largement utilisé lors de nos précédents travaux : le , cette mesure indique la proportion de la variance expliquée par le modèle.

* 0 % le modèle n’explique par la variable Y
* 100 % le modèle explique la variabilité de Y lié à la liaison linéaire des variables explicatives entièrement

Rappelons son explication mathématiques , soit SCE la somme des distances au carré entre chaque valeur prédite par le modèle et la moyenne des réponses

Nous obtenons alors la part de dispersion expliquée par le modèle.

Puis, nous calculons la dispersion totale des données nommé SCT

*Avec* *une valeur prise par une variable expliquée*

Nous obtenons alors le R² par la combinaisons des calculs précédents

Pour compléter cette mesure, nous utiliserons donc l’erreur absolue moyenne en pourcentage (MAPE en anglais) Il s’agit de la moyenne des écarts en valeur absolue par rapport aux valeurs observées.

C’est donc un pourcentage et par conséquent un indicateur pratique de comparaison.

**Exemple de mise en production**

Imaginons que l’on utilise nos modèles pour prédire le nombre de voyageurs, si l’on prévoit trop de passagers alors les moyens mis en place pour les accueillir sont partiellement utilisés, il en résulte un coût de 5 euros par passagers.

Si l’on a prédit moins de passagers que la réalité, notre compagnie doit commander en livraison urgente, il en résulte un coût de 10 euros par passagers.

Observons cette mise en production sur nos modèles.

## Calculs des performances

Dans les concerts, chaque individu chante comme une casserole, pourtant la foule chante systématiquement juste, suivant cet intuition réalisons une prédiction qui est une moyennne des différents modèles réalisés.

DataAffichageGGplot$MoyenneDesModele <- rowMeans(DataAffichageGGplot[,5:9])

Performances Modèles

noms\_modeles <- c("Buys Ballot ","Lissage exponentielle ","Arima","Prophet","LSTM","Moyenne Des Modèles")  
  
for (x in seq\_along(noms\_modeles)){  
 print("----------------------------")  
 print(noms\_modeles[x])  
   
 predictionglobale <- DataAffichageGGplot[13:104,4+x]  
 predictionEntrainement <- DataAffichageGGplot[13:96,4+x]  
 predictionTest <- DataAffichageGGplot[97:104,4+x]  
   
   
 cat("MAPE globale : ",MAPE(DataAffichageGGplot$trafic[13:104], predictionglobale),"\n")  
cat("R carré globale : ",Rcarre(DataAffichageGGplot$trafic[13:104], predictionglobale),"\n")  
  
cat("MAPE Entrainement : ",MAPE(DataAffichageGGplot$trafic[13:96], predictionEntrainement),"\n")  
cat("R carré Entrainement : ",Rcarre(DataAffichageGGplot$trafic[13:96], predictionEntrainement),"\n")  
  
cat("MAPE Test : ",MAPE(DataAffichageGGplot$trafic[97:104], predictionTest),"\n")  
cat("R carré Test : ",Rcarre(DataAffichageGGplot$trafic[97:104], predictionTest),"\n")  
  
MiseEnProductionDuModele <- CoutDesErreurs(DataAffichageGGplot$trafic[97:104], predictionTest)  
cat("Nombre de Passagers prévus en plus : ",MiseEnProductionDuModele[1],"\nNombre de Passagers prévus en moins : ",MiseEnProductionDuModele[2],"\nCout des erreurs en Euros : ",MiseEnProductionDuModele[3] )  
  
  
cat("\nDifférence en nbre de voyageurs sur le dernier mois : ",predictionglobale[92] - DataAffichageGGplot$trafic[104],"\n")  
   
}

## [1] "----------------------------"  
## [1] "Buys Ballot "  
## MAPE globale : 2.205698   
## R carré globale : 0.9740106   
## MAPE Entrainement : 2.142334   
## R carré Entrainement : 0.9747808   
## MAPE Test : 2.871024   
## R carré Test : 0.8036533   
## Nombre de Passagers prévus en plus : 4848.58   
## Nombre de Passagers prévus en moins : -91908.62   
## Cout des erreurs en Euros : 1016058  
## Différence en nbre de voyageurs sur le dernier mois : 36037.92   
## [1] "----------------------------"  
## [1] "Lissage exponentielle "  
## MAPE globale : 2.270427   
## R carré globale : 0.9725982   
## MAPE Entrainement : 2.200386   
## R carré Entrainement : 0.9736448   
## MAPE Test : 3.005848   
## R carré Test : 0.7863116   
## Nombre de Passagers prévus en plus : 5276.056   
## Nombre de Passagers prévus en moins : -96021.53   
## Cout des erreurs en Euros : 1065736  
## Différence en nbre de voyageurs sur le dernier mois : 38627.88   
## [1] "----------------------------"  
## [1] "Arima"  
## MAPE globale : 2.551599   
## R carré globale : 0.9655633   
## MAPE Entrainement : 2.526595   
## R carré Entrainement : 0.9629896   
## MAPE Test : 2.814135   
## R carré Test : 0.8420685   
## Nombre de Passagers prévus en plus : 17397.14   
## Nombre de Passagers prévus en moins : -81186.49   
## Cout des erreurs en Euros : 1159808  
## Différence en nbre de voyageurs sur le dernier mois : 24913.57   
## [1] "----------------------------"  
## [1] "Prophet"  
## MAPE globale : 2.0477   
## R carré globale : 0.9744149   
## MAPE Entrainement : 1.919235   
## R carré Entrainement : 0.9777721   
## MAPE Test : 3.396584   
## R carré Test : 0.7327864   
## Nombre de Passagers prévus en plus : 1911.595   
## Nombre de Passagers prévus en moins : -112557.7   
## Cout des erreurs en Euros : 1163809  
## Différence en nbre de voyageurs sur le dernier mois : 42462.18   
## [1] "----------------------------"  
## [1] "LSTM"  
## MAPE globale : 3.920528   
## R carré globale : 0.9274386   
## MAPE Entrainement : 4.062261   
## R carré Entrainement : 0.9175707   
## MAPE Test : 2.432328   
## R carré Test : 0.7936451   
## Nombre de Passagers prévus en plus : 65223.98   
## Nombre de Passagers prévus en moins : -26116.42   
## Cout des erreurs en Euros : 1565644  
## Différence en nbre de voyageurs sur le dernier mois : 6606.791   
## [1] "----------------------------"  
## [1] "Moyenne Des Modèles"  
## MAPE globale : 2.24867   
## R carré globale : 0.9741094   
## MAPE Entrainement : 2.21158   
## R carré Entrainement : 0.9735446   
## MAPE Test : 2.638107   
## R carré Test : 0.8422871   
## Nombre de Passagers prévus en plus : 14032.08   
## Nombre de Passagers prévus en moins : -76658.76   
## Cout des erreurs en Euros : 1047229  
## Différence en nbre de voyageurs sur le dernier mois : 29729.67

## Conclusion

lorem ipsum