Introduction to Time Series Analysis

Kritchavat Ploddi

Section 1

ARIMA and Exponential Smoothing Models

Packages requirement

Before proceeding, please download and install following packages:

```
library(forecast)
library(tseries)
library(ggplot2)
library(dplyr)
```

Stationary

Another important concept in time series analysis is stationary.

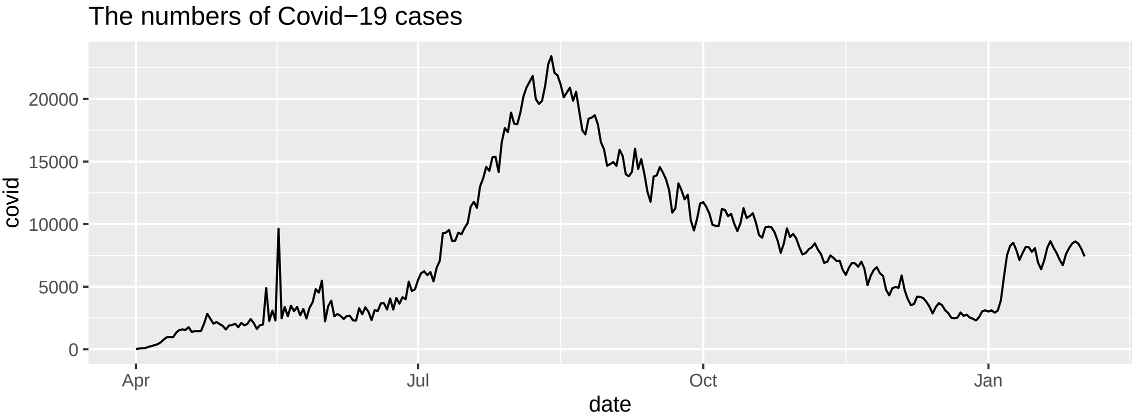
This is because most time series forecasting models require stationary assumptions.

Stationary

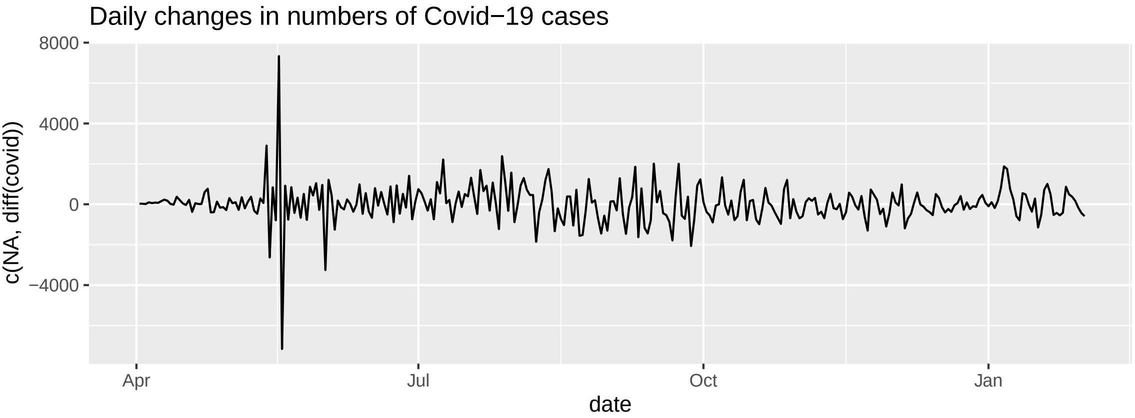
A time series Y is said to be stationary if all of its values y_t do not depend on time t.

In other words, the distribution of y_t has constant mean and variance.

Stationary ?


The numbers of Covid−19 cases
covid
20000
15000
10000
5000
0
Apr
Jul
Oct
Jan
date

Stationary ?



Daily changes in numbers of Covid−19 cases
8000
4000
0
−4000
c(NA, diff(covid))
Apr
Jul
Oct
Jan
date

How to transform non-stationary time series to be stationary

Stationary is characterized by constant mean and variance.

Transformations help to stabilize the variance.

Difference help to stabilize the mean

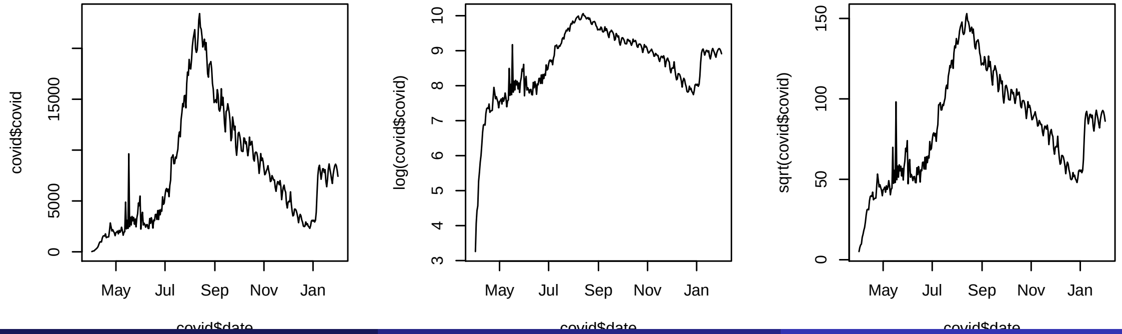
Transformations

Variance could be stabilized by taking:

- (Natural) Logarithm (log)
- Squared root (sqrt)

Transformations

```
par(mfrow=c(1,3))
plot(covid$date, covid$covid, type='l')
plot(covid$date, log(covid$covid), type='l')
plot(covid$date, sqrt(covid$covid), type='l')
```



Transformations

We found that taking natural logarithm on Covid cases mostly stabilize the variance.

```
covid$log_covid <- log(covid$covid)
head(covid[, c('date', 'covid', 'log_covid')])
```

```
##         date covid log_covid
## 1 2021-04-01    26  3.258097
## 2 2021-04-02    58  4.060443
## 3 2021-04-03    84  4.430817
## 4 2021-04-04    96  4.564348
## 5 2021-04-05   194  5.267858
## 6 2021-04-06   250  5.521461
```

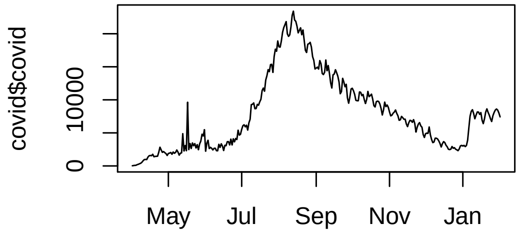
Differencing

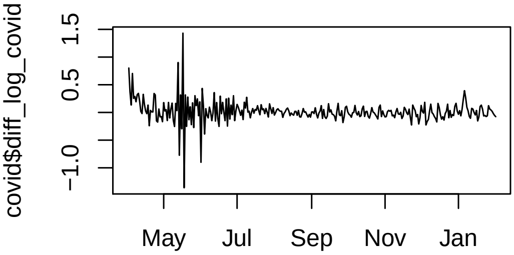
Difference helps stabilize the mean.

In R, we could use the command diff() (plus NA offset at the first index).

```
covid$diff_log_covid <- c(NA, diff(covid$log_covid))

par(mfrow=c(1,2))
plot(covid$date, covid$covid, type='l')
plot(covid$date, covid$diff_log_covid, type='l')
```





How could we statistcally test if a series is stationary?

We could use KPSS test (Kwiatkowski–Phillips–Schmidt–Shin test) to test ifa series is stationary.

Using the function kpss.test() from tseries package.

```
print(kpss.test((covid$diff_log_covid))) # Log transform with differencing
```

```
##
##  KPSS Test for Level Stationarity
##
## data:  (covid$diff_log_covid)
## KPSS Level = 0.9128, Truncation lag parameter = 5, p-value = 0.01
```

How could we statistcally test if a series is stationary?

This suggested that log-transformed diferencing is not enough.

We have to take another differencing:

```
diff_diff_log_covid <- diff(na.omit(covid$diff_log_covid))
print(kpss.test(diff_diff_log_covid)) # Log transform with second differencin
```

```
##
##  KPSS Test for Level Stationarity
##
## data:  diff_diff_log_covid
## KPSS Level = 0.068379, Truncation lag parameter = 5, p-value = 0.1
```

How could we statistcally test if a series is stationary?

We could use KPSS test (Kwiatkowski–Phillips–Schmidt–Shin test) to test ifa series is stationary.

Using the function kpss.test() from tseries package.

```
print(kpss.test(covid$log_covid)) # Log tranform
```

```
##
##  KPSS Test for Level Stationarity
##
## data:  covid$log_covid
## KPSS Level = 1.6925, Truncation lag parameter = 5, p-value = 0.01
```

How could we statistcally test if a series is stationary?

We could use KPSS test (Kwiatkowski–Phillips–Schmidt–Shin test) to test ifa series is stationary.

Using the function kpss.test() from tseries package.

```
print(kpss.test(covid$covid)) # Original series
```

```
##
##  KPSS Test for Level Stationarity
##
## data:  covid$covid
## KPSS Level = 1.105, Truncation lag parameter = 5, p-value = 0.01
```

ARIMA Models

Autoregressive Integrated Moving Average (ARIMA) Models

An ARIMA model is rarely interpretable in terms of visible data structures like trend and seasonality. But it can capture a huge range of time series patterns.

ARIMA Models

Autoregressive Integrated Moving Average (ARIMA) Models

An ARIMA model is rarely interpretable in terms of visible data structures like trend and seasonality. But it can capture a huge range of time series patterns.

It is composed of an Autoregressive (AR) and Moving Average (MA) models

Autoregressive models (AR)

$$y_t = c + \phi_1 y_{t-1} + \phi_2 y_{t-2} + \cdots + \phi_p y_{t-p} + \varepsilon_t,$$

This is called AR(p) model.

AR models predict future values based on previous values (at p lags).

Moving Average Models (MA)

$$y_t = c + \varepsilon_t + \theta_1 \varepsilon_{t-1} + \theta_2 \varepsilon_{t-2} + \cdots + \theta_q \varepsilon_{t-q},$$

This is called MA(q) model.

MA models predict future values based on previous errors (at q lags).

ARMA and ARIMA models

ARMA(p,q) is a linear combination of AR(p) and MA(q). ARMA predict future values based on both previous values and errors.

ARIMA(p,n,q) is similar to ARMA(p,q), except only n difference has been taken. This is done for taking an account for non-stationary time series.

Forecasting Dengue with ARIMA models

```
dhfrain <- read.csv('data/dhfrain2.csv')
dhfrain$date <- as.Date(dhfrain$date)
dhfrain[, c("date", "cases")]
```

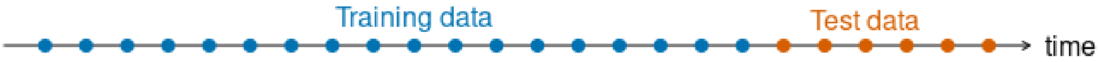
```
##            date cases
## 1    2009-01-01  2614
## 2    2009-02-01  2057
## 3    2009-03-01  2324
## 4    2009-04-01  2947
## 5    2009-05-01  6234
## 6    2009-06-01  8569
## 7    2009-07-01  7184
## 8    2009-08-01  7302
## 9    2009-09-01  5016
## 10   2009-10-01  4640
## 11   2009-11-01  4723
```

Forecasting Dengue with ARIMA models

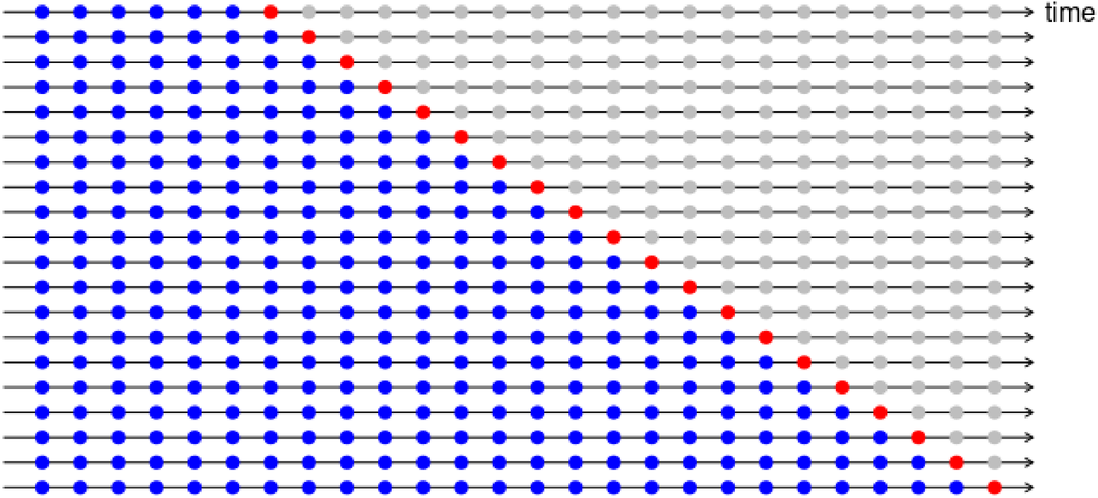
Train-test splitting

```
train <- dhfrain[dhfrain$year < 2018, ]
test <- dhfrain[dhfrain$year == 2018, ]
```

Train-test splitting

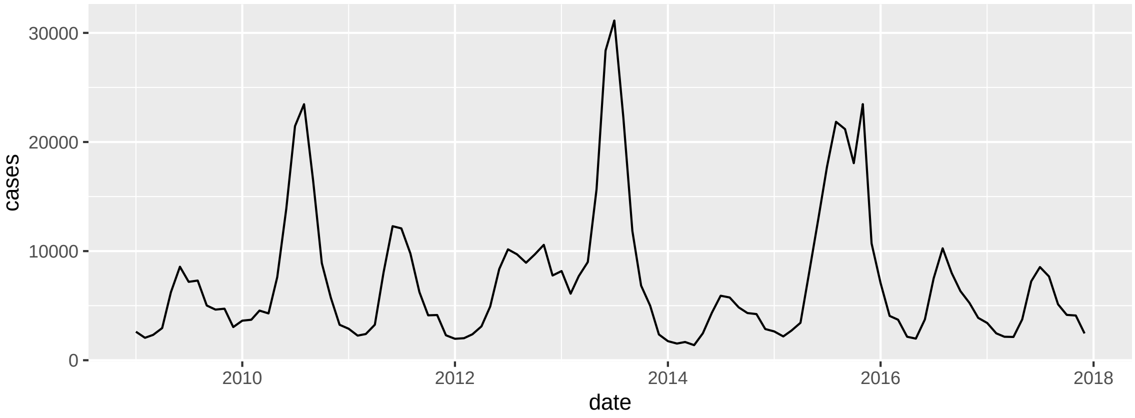


Cross-validation



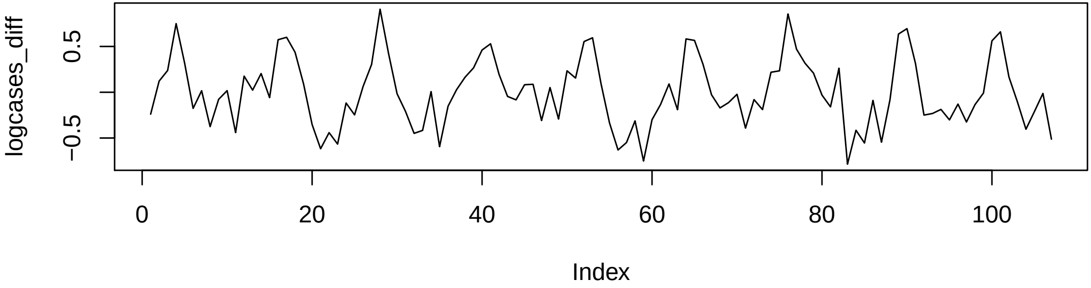
Forecasting Dengue with ARIMA models

```
ggplot(train) + geom_line(aes(x=date, y=cases))
```



Forecasting Dengue with ARIMA models

```
train$logcases <- log(train$cases)
logcases_diff <- diff(train$logcases)
plot(logcases_diff, type='l')
```



Forecasting Dengue with ARIMA models

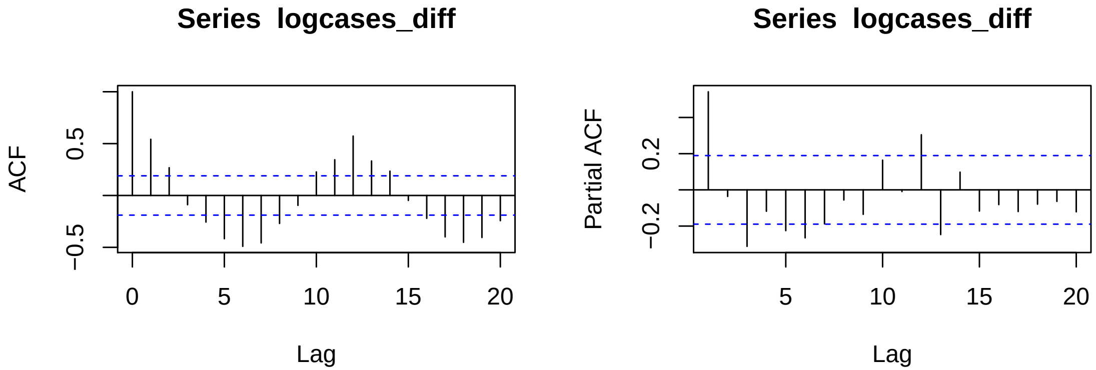
KPSS test for stationary

```
library(tseries)
kpss.test(logcases_diff)
```

```
##
##  KPSS Test for Level Stationarity
##
## data:  logcases_diff
## KPSS Level = 0.038409, Truncation lag parameter = 4, p-value = 0.1
```

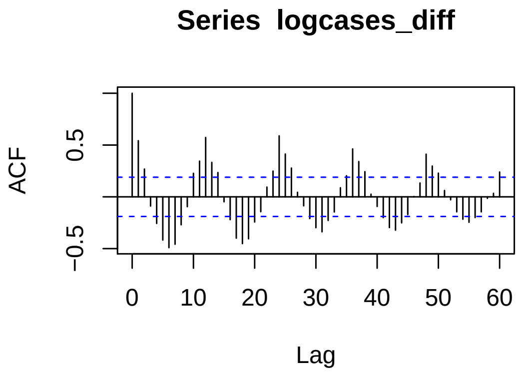
ACF and PACF plot

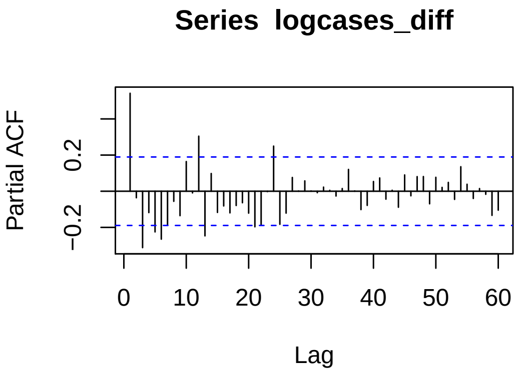
```
par(mfrow=c(1,2))
acf(logcases_diff)
pacf(logcases_diff)
```



ACF and PACF plot (seasonal)

```
par(mfrow=c(1,2))
acf(logcases_diff, lag=60)
pacf(logcases_diff, lag=60)
```





Forecasting Dengue with ARIMA models

So the ARIMA model would be: (S)ARIMA (1,1,0) (2,1,0), 12

```
manual <- arima(train$logcases,
                order=c(1,1,0),
                seasonal=list(order=c(3,1,0), period=12))
```

Auto ARIMA

```r
library(forecast)
auto <- auto.arima(train$logcases)
```

Models evaluation

```
test$forecast_manual <- exp(forecast(manual, h=12)$mean)
test$forecast_auto <- exp(forecast(auto, h=12)$mean)
```

Models evaluation: metrics

Error $Error = Y_{actual} - Y_{forecast}$

Absolute Error $AbsError = Abs(Error)$

Absolute Percentage Error $AbsPctError = Abs(Error)/Y_{actual} \times 100$

Squred Error $SqError = Error^2$

Mean Squared Error $MSE = mean(SqError)$

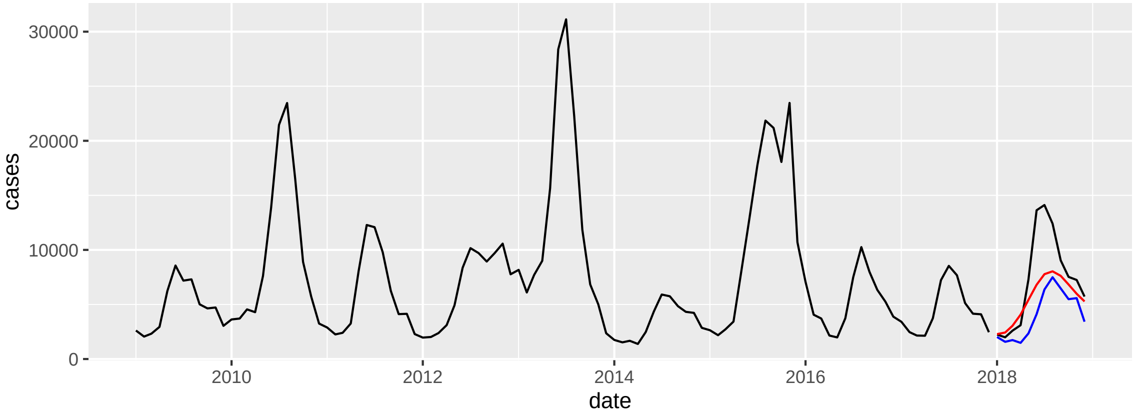
Root mean squred error $RMSE = \sqrt{MSE}$

Mean Absolute Error $MAE = mean(AbsError)$

Mean Absolute Percentage Error $MAPE = mean(AbsPctError)$

Models evaluation

```
ggplot() +
  geom_line(data = train, aes(x=date, y=cases)) +
  geom_line(data = test, aes(x=date, y=cases)) +
  geom_line(data = test, aes(x=date, y=forecast_auto), color='red') +
  geom_line(data = test, aes(x=date, y=forecast_manual), color='blue')
```



Models evaluation

Root mean squared error:

```
error_manual <- with(test, forecast_manual - cases)
error_auto <- with(test, forecast_auto - cases)

# SE

sqerror_manual <- error_manual^2
sqerror_auto <- error_auto^2

# MSE
mse_manual <- mean(sqerror_manual)
mse_auto <- mean(sqerror_auto)

# RMSE
rmse_manual <- sqrt(mse_manual)
rmse_auto <- sqrt(mse_auto)

c(manual=rmse_manual, auto=rmse_auto)
```

```
##   manual     auto
## 4298.099 3096.161
```

Models evaluation

Mean absolute error: (MAE)

```
error_manual <- with(test, forecast_manual - cases)
error_auto <- with(test, forecast_auto - cases)

# ABS Error

abserror_manual <- abs(error_manual)
abserror_auto <- abs(error_auto)

# MAE
mae_manual <- mean(abserror_manual)
mae_auto <- mean(abserror_auto)


c(manual=mae_manual, auto=mae_auto)
```

```
##   manual     auto
## 3234.929 2094.105
```

Model evaluation

Mean absolute percentage error: (MAE)

```
error_manual <- with(test, forecast_manual - cases)
error_auto <- with(test, forecast_auto - cases)

# ABS Error

abserror_manual <- abs(error_manual)
abserror_auto <- abs(error_auto)

# Abs Percentage Error
abs_pct_error_manual <- abserror_manual/test$cases * 100
abs_pct_error_auto <- abserror_auto/test$cases * 100

# MAPE
mape_manual <- mean(abs_pct_error_manual)
mape_auto <- mean(abs_pct_error_auto)

c(manual=mape_manual, auto=mape_auto)
```

```
##   manual     auto
## 38.90675 23.20156
```

Section 2

Exponential smoothing methods

Naïve average

Using the naïve method, all forecasts for the future are equal to the last observed value of the series,

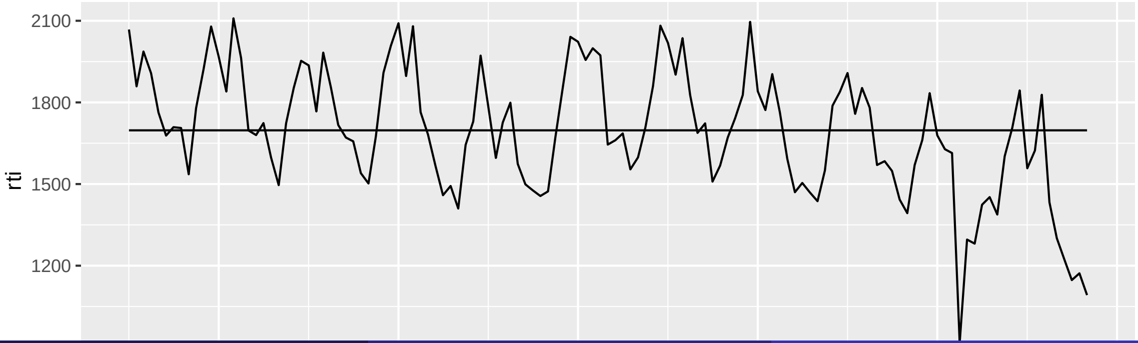
$$\hat{y}_{T+h|T} = y_T$$

```
##          date  rti Naïve
## 1  2011-01-01 2068    NA
## 2  2011-02-01 1859  2068
## 3  2011-03-01 1987  1859
## 4  2011-04-01 1907  1987
## 5  2011-05-01 1763  1907
## 6  2011-06-01 1678  1763
## 7  2011-07-01 1709  1678
## 8  2011-08-01 1706  1709
## 9  2011-09-01 1536  1706
## 10 2011-10-01 1779  1536
## 11 2011-11-01 1925  1779
## 12 2011-12-01 2079  1925
## 13 2012-01-01 1968  2079
## 14 2012-02-01 1840  1968
## 15 2012-03-01 2109  1840
## 16 2012-04-01 1963  2109
## 17 2012-05-01 1697  1963
## 18 2012-06-01 1680  1697
## 19 2012-07-01 1724  1680
## 20 2012-08-01 1597  1724
```

Average method

Using the average method, all future forecasts are equal to a simple average of the observed data,

$$\hat{y}_{T+h|T} = \frac{1}{T}\sum_{t=1}^{T} y_t,$$



Simple Exponential Smoothing

$$\hat{y}_{T+1|T} = \alpha y_T + \alpha(1-\alpha) y_{T-1} + \alpha(1-\alpha)^2 y_{T-2} + \cdots$$

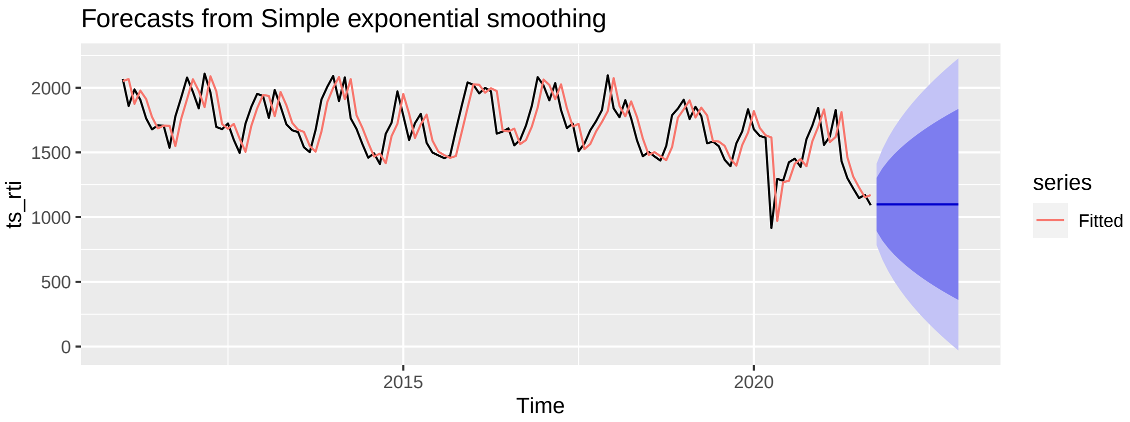
Simple Exponential Smoothing

```
rti <- read.csv("data/rti.csv")
rti$date <- as.Date(rti$date)
ts_rti <- ts(rti$rti, start=c(2011,1), freq=12)
ts_rti
```

```
##       Jan  Feb  Mar  Apr  May  Jun  Jul  Aug  Sep  Oct  Nov  Dec
## 2011 2068 1859 1987 1907 1763 1678 1709 1706 1536 1779 1925 2079
## 2012 1968 1840 2109 1963 1697 1680 1724 1597 1496 1723 1853 1953
## 2013 1936 1767 1983 1856 1717 1671 1657 1540 1502 1675 1909 2008
## 2014 2091 1897 2080 1764 1682 1565 1459 1493 1410 1644 1730 1972
## 2015 1786 1596 1726 1799 1574 1499 1477 1456 1473 1672 1860 2041
## 2016 2023 1956 1999 1973 1645 1661 1686 1554 1598 1708 1860 2082
## 2017 2018 1902 2036 1827 1688 1723 1509 1568 1671 1742 1827 2096
## 2018 1841 1772 1904 1763 1593 1470 1504 1469 1437 1550 1788 1839
## 2019 1908 1758 1853 1781 1570 1584 1548 1443 1393 1570 1662 1834
## 2020 1679 1628 1614  916 1296 1281 1424 1452 1388 1602 1707 1844
```

Simple Exponential Smoothing

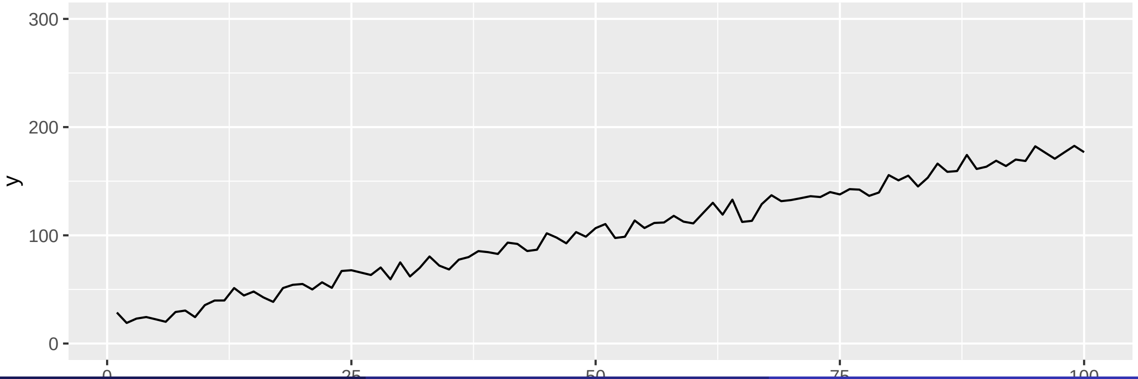
```
forecast_rti <- ses(ts_rti, h=15)
autoplot(forecast_rti) +
  autolayer(fitted(forecast_rti), series="Fitted")
```



Additive and multiplicative trend

Additive trend

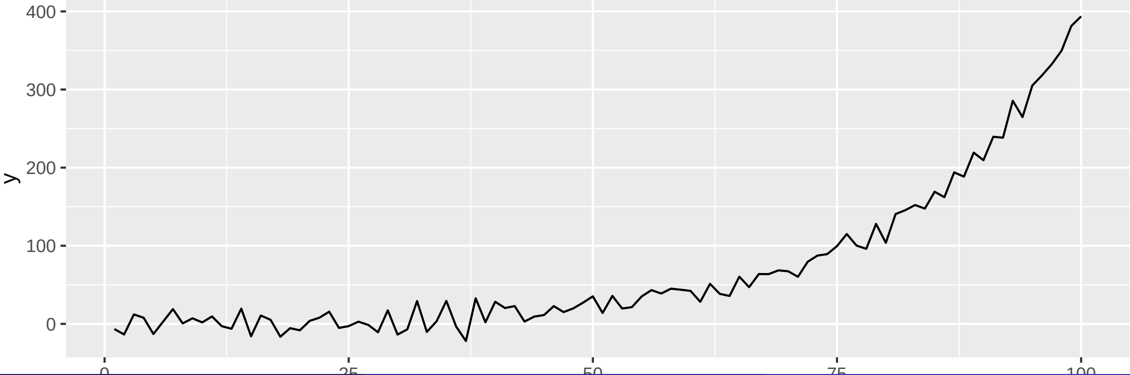
```
data.frame(t=1:100, e=rnorm(100,0,5)) %>%
  mutate(y=20+1.6*t+e) %>%
  ggplot() + geom_line(aes(x=t, y=y)) + ylim(0,300)
```



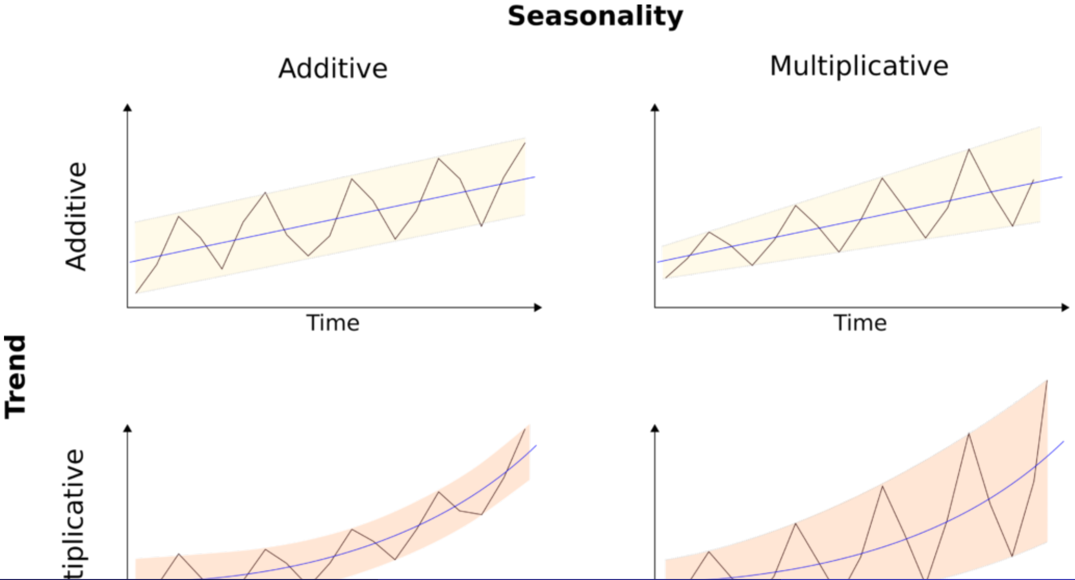
Additive and multiplicative trend

Multiplicative trend

```
data.frame(t=1:100, e=rnorm(100,0,10)) %>%
  mutate(y=exp(0.06*t)+e) %>%
  ggplot() + geom_line(aes(x=t, y=y))
```



Additive and multiplicative seasonal



Classification of exponential smooting method

Trend Component	Seasonal Component		
	N (None)	A (Additive)	M (Multiplicative)
N (None)	(N,N)	(N,A)	(N,M)
A (Additive)	(A,N)	(A,A)	(A,M)
A_d (Additive damped)	(A_d,N)	(A_d,A)	(A_d,M)

Some of these methods we have already seen using other names:

Short hand	Method
(N,N)	Simple exponential smoothing
(A,N)	Holt's linear method
(A_d,N)	Additive damped trend method
(A,A)	Additive Holt-Winters' method
(A,M)	Multiplicative Holt-Winters' method
(A_d,M)	Holt-Winters' damped method

Fitting exponential smooting method in R

In R, using ets() function from forecast package to fit an exponential smoothing

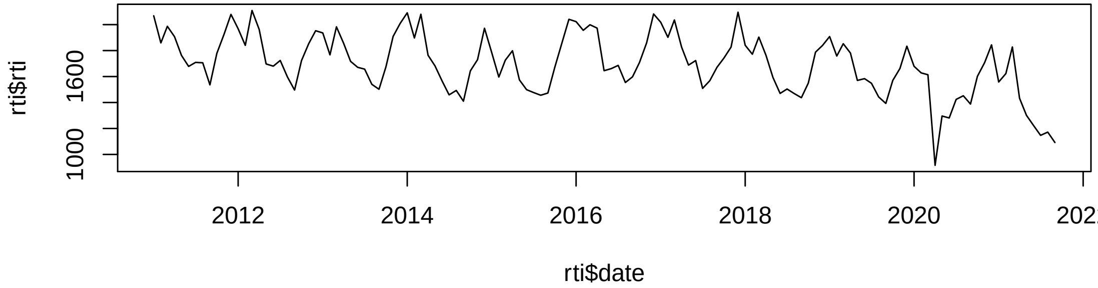
We can specify the type of ets using the model parameters.

- "NNN" Simple Average
- "NAN" Additive Trend without seasonal
- "NMN" Multiplicative Trend without seasonal
- "NAA" Additive Trend and seasonal (Additive Holt-Winter's)
- "NMM" Multiplicative Trend and seasonal (Multiplicative Holt-Winter's)

We can also use "Z", e.g. "ZZZ" to let R to find the best fitted ETS models.

Fitting exponential smooting method in R

For example, if we wish to fit Additive Holt-Winter methods (Additive trend and seasonality) and Multiplicative Holt-Winter methods (Multiplicative trend and seasonality)

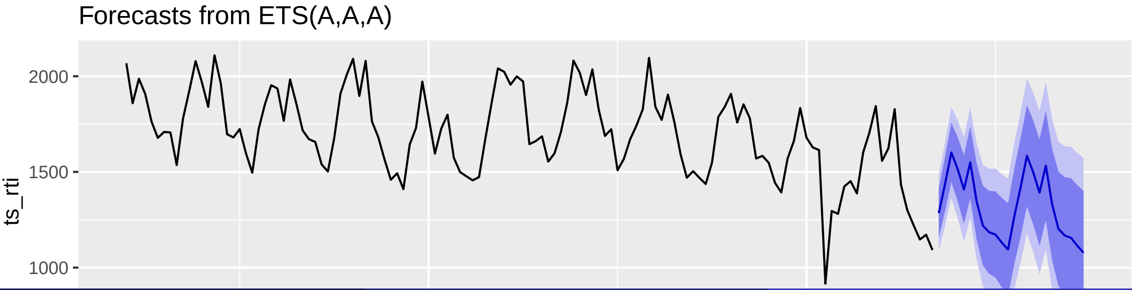


Fitting exponential smooting method in R

Fitting Additive Holt-Winter methods (Additive trend and seasonality), “ZAA’

Fitting Multiplicative Holt-Winter methods (Multiplicative trend and seasonality), “ZMM’

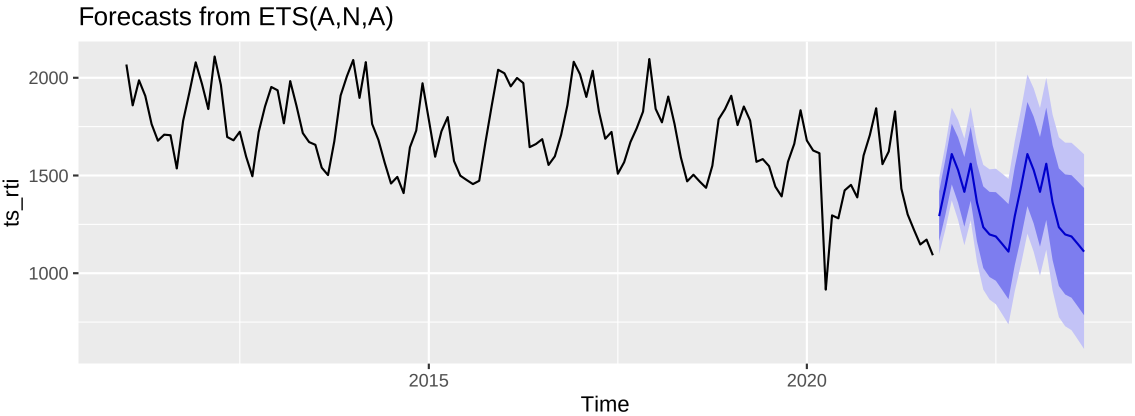
```
zaa_rti <- ets(ts_rti, model="ZAA") # Additive
zmm_rti <- ets(ts_rti, model="ZMM") # Multiplicative
par(mfrow=c(1,2))
autoplot(forecast(zaa_rti), h=16)
```



Fitting exponential smooting method in R

We can also use “ZZZ” to let R automatically fit and Exponential Smoothing Models

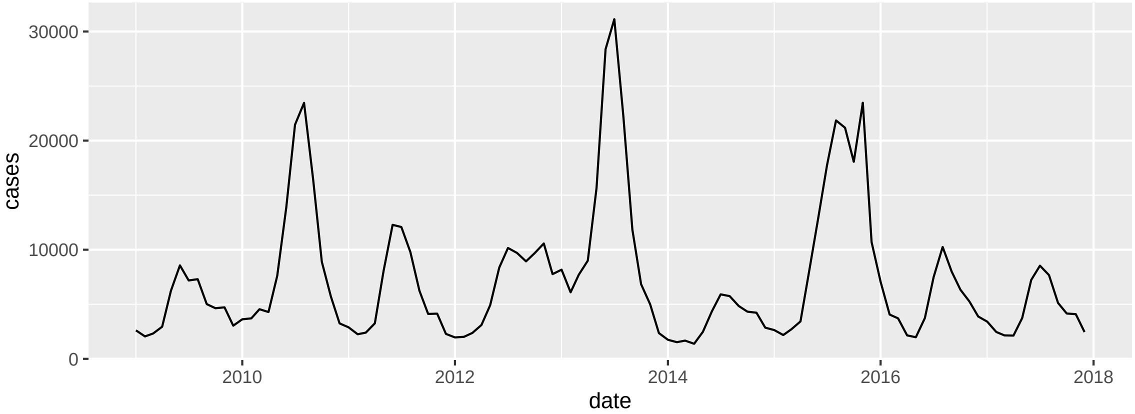
```
zzz_rti <- ets(ts_rti, model="ZZZ")
autoplot(forecast(zzz_rti), h=16)
```



Fitting an ETS with Dengue data

We would be manually fitting an ETS with only multiplicative seasonal with training data of Dengue (2009-2017).

We would also use an automated methods.

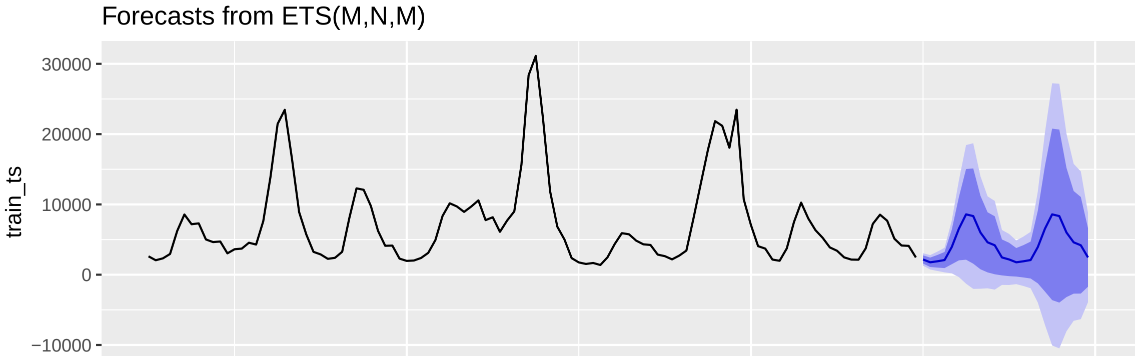


Fitting an ETS with Dengue data

Using “ZNM” for multiplicative seasonal ETS

Using “ZZZ” for automated ETS

```
train_ts <- ts(train$cases, start=c(2009,1), freq=12)
znm_train <- ets(train_ts, model="ZNM")
zzz_train <- ets(train_ts, model="ZZZ") # Auto
par(mfrow=c(1,2))
autoplot(forecast(znm_train), h=12)
```



Models evaluation

Mean Absolute Percentage Errors

```
forecast_test_znm <- forecast(znm_train, h=12)$mean
forecast_test_zzz <- forecast(zzz_train, h=12)$mean

abs_error_znm <- abs(forecast_test_znm - test$cases)
abs_error_zzz <- abs(forecast_test_zzz - test$cases)

abs_pct_error_znm  <- abs_error_znm/test$cases * 100
abs_pct_error_zzz  <- abs_error_zzz/test$cases * 100

mape_znm <- mean(abs_pct_error_znm)
mape_zzz <- mean(abs_pct_error_zzz)

c(ZNM=mape_znm, AUTO=mape_zzz)
```

```
##      ZNM     AUTO
## 34.62963 34.62963
```

Models evaluation

Comparing MAPE with previous ARIMA models

```
c(AUTO_ETS=mape_zzz, manual_ARIMA=mape_manual, AUTO_ARIMA=mape_auto)
```

```
##     AUTO_ETS manual_ARIMA   AUTO_ARIMA
##     34.62963     38.90675     23.20156
```