Notes 10: Time Series

Contents

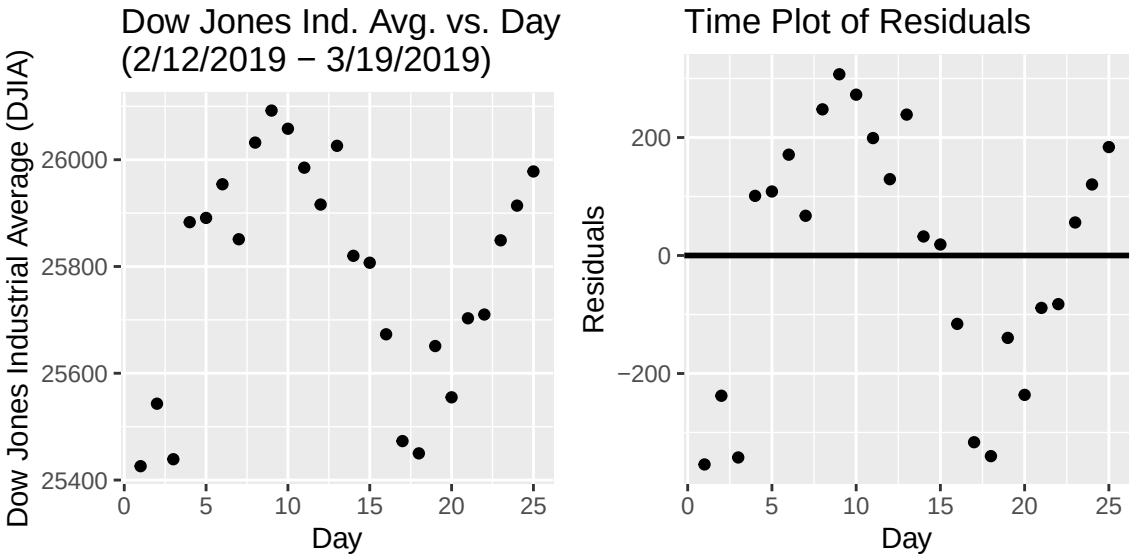
Introduction

Time series analysis is extremely common in many disciplines. It is often the case that we want to predict future events from past ones or model the trend and error of data separately.

The key aspect of time series data is that of **autocorrelation**. It is usually the case that one observation will be dependent on a previous observation or observations.

Example: Dow Jones

Consider the following data on the Dow Jones Industrial Average (DJIA) for stocks:



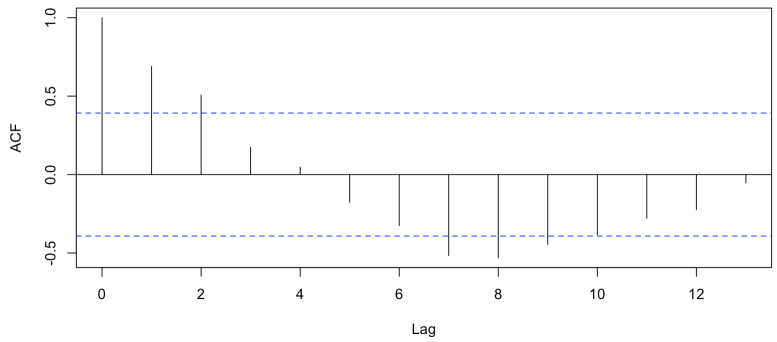
In this example, there is definitely some correlation in the values. If the previous value was large, the next is more likely to be large. Because of this (and the fact that the relationship is not linear), a regression model would not be appropriate.

When this is the case, a **time series model** is more appropriate since these models assume the data are ordered and dependent on time. A linear regression model assumes the data are independent and random.

ACF

We can plot the autocorrelation using the `acf()` function.

```
DJIA <- c(25426, 25543, 25439, 25883, 25891, 25954, 25851, 26032,
          26092, 26058, 25985, 25916, 26026, 25820, 25807, 25673,
          25473, 25450, 25651, 25555, 25703, 25710, 25849, 25914, 25978)
day <- 1:length(DJIA)
acf(DJIA)
```



The autocorrelation at lag k can be computed using

$$\hat{\rho}_k = \frac{\sum_{t=k+1}^{n}(x_t - \bar{x})(x_{t-k} - \bar{x})}{\sum_{t=1}^{n}(x_t - \bar{x})^2}$$

where $\bar{x}$ is the average value of the time series and x_t is the tth value of the time series.

```
# ACF at lag 1
sum((DJIA[2:25] - mean(DJIA)) * (DJIA[1:24] - mean(DJIA)))/
  sum((DJIA - mean(DJIA))^2)
```

```
## [1] 0.6903616
```

```
# ACF at lag 2
sum((DJIA[3:25] - mean(DJIA)) * (DJIA[1:23] - mean(DJIA)))/
  sum((DJIA - mean(DJIA))^2)
```

```
## [1] 0.5067019
```

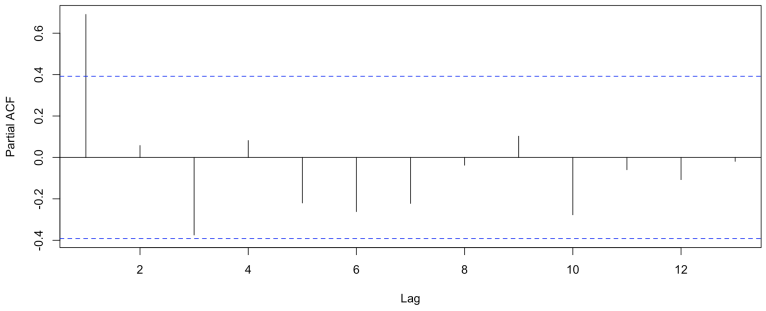
```
# Verify
acf(DJIA, lag.max = 2, plot = F)
```

```
##
## Autocorrelations of series 'DJIA', by lag
##
##     0     1     2
## 1.000 0.690 0.507
```

PACF

We can plot the partial autocorrelation function (PACF) values using the `pacf()` function. The PACF is useful because it controls for autocorrelation at other lags. So, the kth value of the PACF is the correlation between the time series values separated by a lag of k while controlling for all lags between them.

```
pacf(DJIA)
```



Time Series Model

The main idea of a time series model is that the data, which are represented by x_t, can be decomposed into several components:

1. The trend, m_t.
2. The seasonality, s_t.
3. The error, e_t. This is usually a sequence of correlated random variables with mean zero.

Then there are two primary types of time series models:

Additive: $x_t = m_t + s_t + e_t$

Multiplicative: $x_t = m_t \cdot s_t \cdot e_t$

Of course, like linear regression, transformations can be made on x_t as well.

An **additive model**, $x_t = m_t + s_t + e_t$, is appropriate if the magnitude of the seasonal fluctuations does not vary with the level of time series.

A **multiplicative model**, $x_t = m_t \cdot s_t \cdot e_t$ is appropriate if the seasonal fluctuations increase or decrease proportionally with increases and decreases in the level of the series.

Taking the log of a multiplicative model makes it additive:

$$\log(x_t) = \log(m_t \cdot s_t \cdot e_t)$$
$$\log(x_t) = \log(m_t) + \log(s_t) + \log(e_t)$$

Air Passengers Example

Consider the data set `AirPassengers` on the number of US airline passengers from 1949 through 1960. This data set is built into **R** and can be loaded by typing `data(AirPassengers)`.

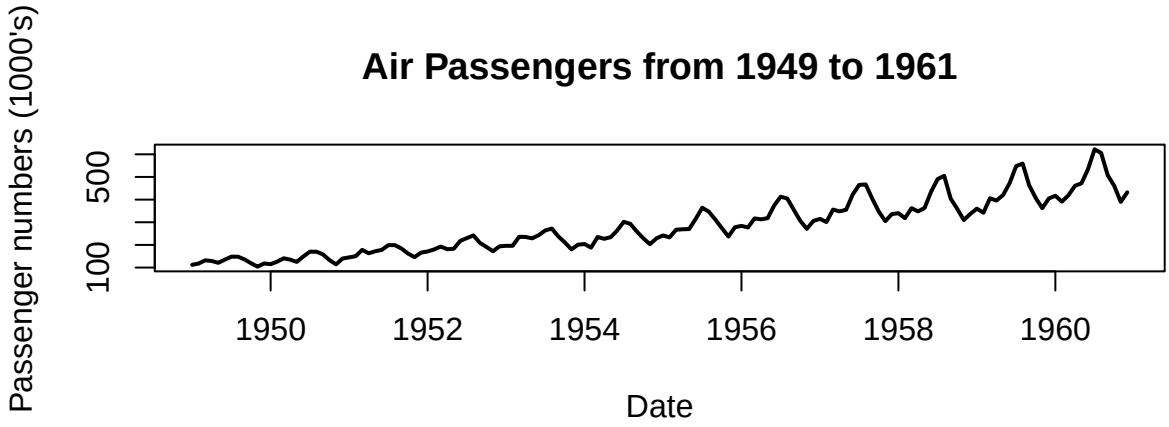
This data set is a `ts` or time series object. That means that some function that can be used on this data set are:

- `frequency()`: tells how often it was sampled per year.

- `cycle()`: gives the cycle value for each year.
- `time()`: gives the time of each sample.
- `deltat()`: gives the difference in time between samples.
- `start()`: gives the first date in the time series.
- `end()`: gives the last date in the time series.

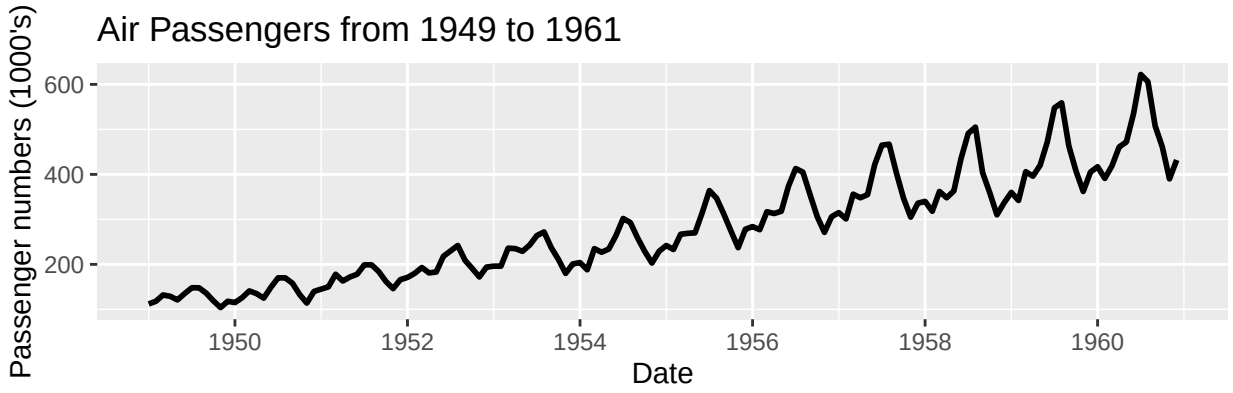
We can plot time series objects with the `plot.ts()` function. If you just use the `plot()` function, it will realize that the object is a time series and automatically use `plot.ts()`:

```
data(AirPassengers)
plot(AirPassengers, xlab = "Date", ylab = "Passenger numbers (1000's)",
     main = "Air Passengers from 1949 to 1961", lwd = 2)
```



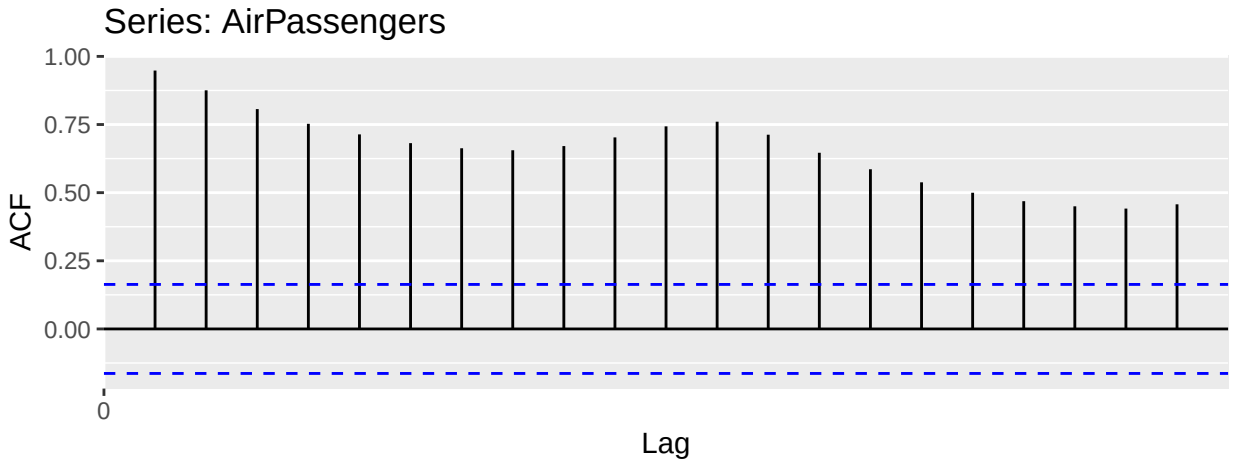
We can plot it using `ggplot2` formatting using the `autoplot()` function in the `ggfortify` package.

```
library(ggfortify)
autoplot(AirPassengers, size = 1) +
  labs(x = "Date", y = "Passenger numbers (1000's)",
       title = "Air Passengers from 1949 to 1961")
```



Additionally, we can pipe the `acf()` function into the `autoplot()` function to make the acf plot look better.

```
acf(AirPassengers, plot = F) |>
  autoplot()
```



When plotting a time series object with `autoplot()`, there is not much customization we can do. We can manipulate the time series object more easily by converting it to a `tsibble`, which is just like a `tibble` but for time series:

```
library(tsibble)
AP <- as_tsibble(AirPassengers) |>
  rename(date = index, num_pass = value) |>
    mutate(lpass = log(num_pass))
head(AP)
```

```
## # A tsibble: 6 x 3 [1M]
##       date num_pass lpass
##      <mth>    <dbl> <dbl>
## 1 1949 Jan      112  4.72
## 2 1949 Feb      118  4.77
## 3 1949 Mar      132  4.88
## 4 1949 Apr      129  4.86
## 5 1949 May      121  4.80
## 6 1949 Jun      135  4.91
```

Another nice advantage tsibbles have is that they work with these functions from the `lubridate` package that is a part of the tidyverse:

- `year()`: gives the year of that observation in the time series.
- `month()`: gives the month of that observation in the time series.
- `quarter()`: gives the quarter of that observation in the time series.
- `week()`: gives the week of that observation in the time series.
- `day()`: gives the day of that observation in the time series.
- `hour()`: gives the hour of that observation in the time series.
- `minute()`: gives the minute of that observation in the time series.
- `second()`: gives the second of that observation in the time series.

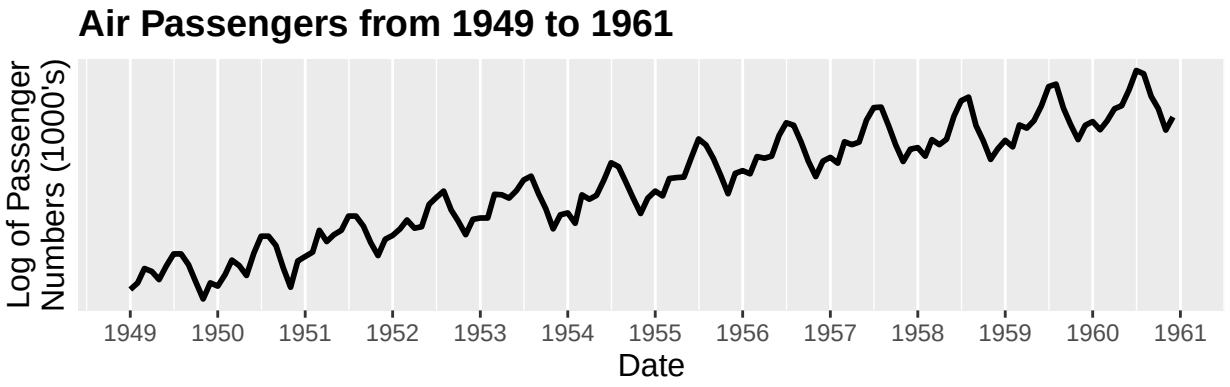
Not all of these functions are approriate for each time series. For the air passengers time series, `year()` and `month()` would be most useful.

```
head(month(AP$date), 14)
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12  1  2
```

Now that it is a tsibble, we can plot it with `ggplot()`:

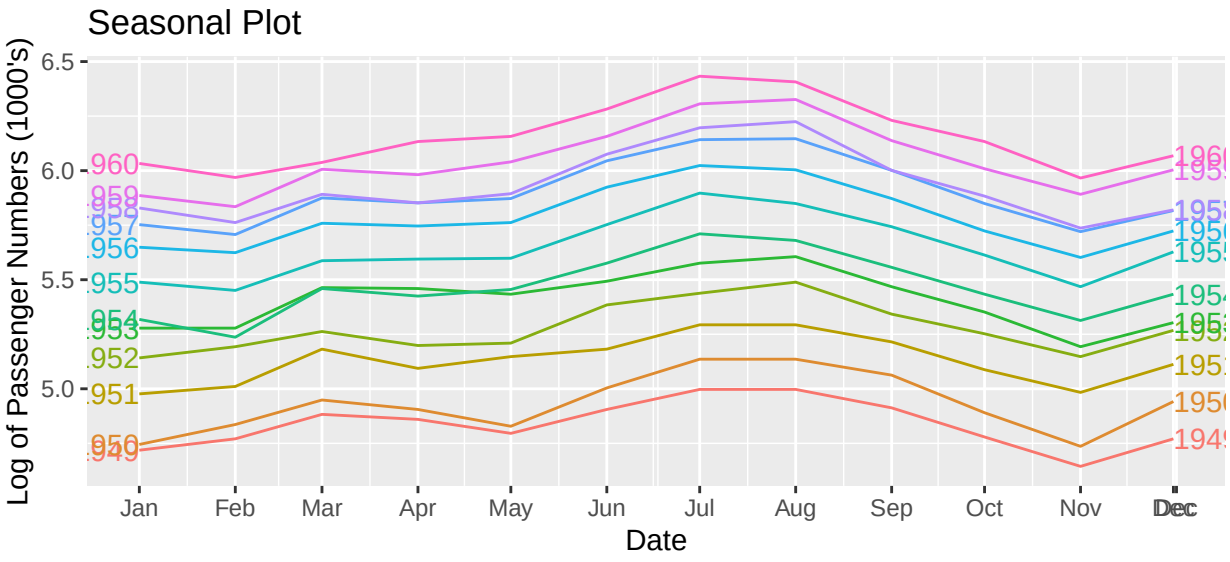
```
ggplot(AP, aes(x = date, y = lpass)) +
  geom_line(linewidth = 1) +
  scale_x_yearmonth(date_breaks = "1 year", date_labels = "%Y") +
  scale_y_continuous(breaks = seq(100, 600, 100)) +
  labs(x = "Date", y = "Log of Passenger\nNumbers (1000's)",
       title = "Air Passengers from 1949 to 1961") +
  theme(axis.title = element_text(size = 12),
        plot.title = element_text(size = 14, face = "bold"))
```



Air Passengers Example - Seasonality

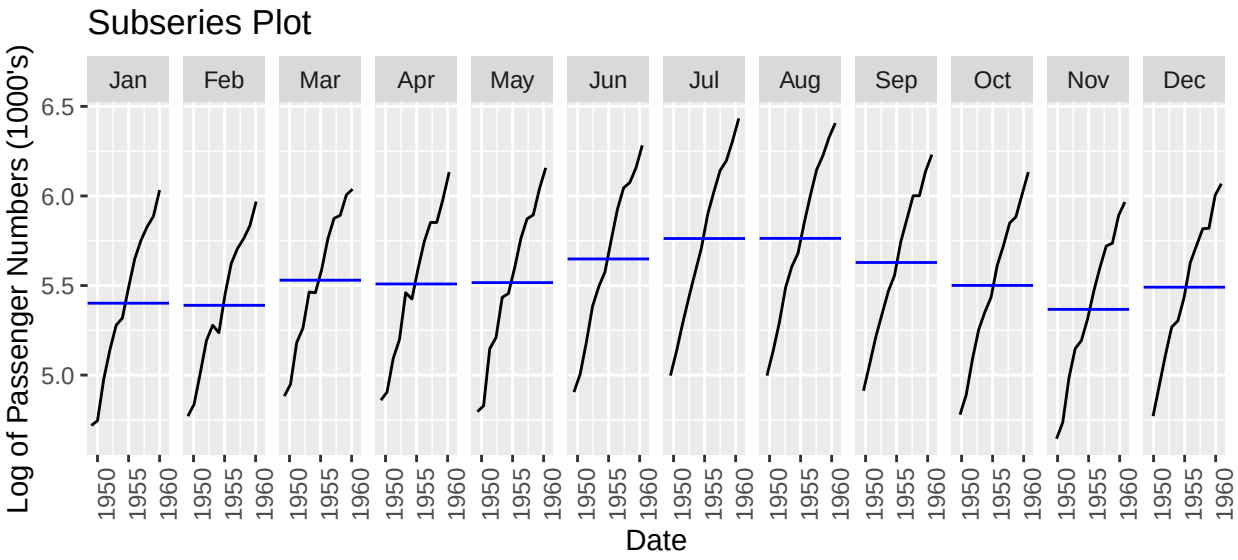
We can see that there is a cyclic nature to this time series (as is often the case) that repeats each year. This is known as the *seasonality* component. Let's check out how air passenger numbers look by month. For many nice plotting functions, we can use the `feasts` library.

```
library(feasts)
gg_season(AP, y = lpass, labels = "both") +
  labs(x = "Date", y = "Log of Passenger Numbers (1000's)",
       title = "Seasonal Plot")
```



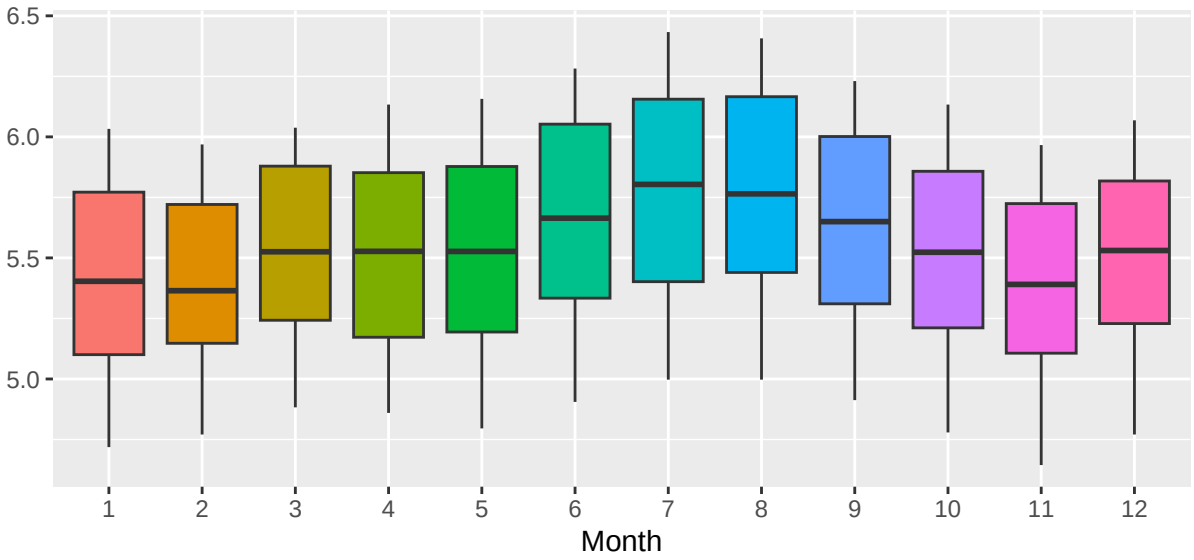
We can use the `gg_subseries()` to obtain another potentially useful plot. The average for each month is shown with the blue line.

```
gg_subseries(AP, y = lpass) +
  labs(x = "Date", y = "Log of Passenger Numbers (1000's)",
       title = "Subseries Plot")
```



Finally, one other familiar way to show approximately the same information is to create side-by-side boxplots by month.

```
AP |> mutate(month = factor(month(date))) |>
  ggplot(aes(x = month, y = lpass, fill = month)) +
    geom_boxplot() +
    theme(legend.position = "none") +
    labs(x = "Month", y = "")
```



Time Series Modeling

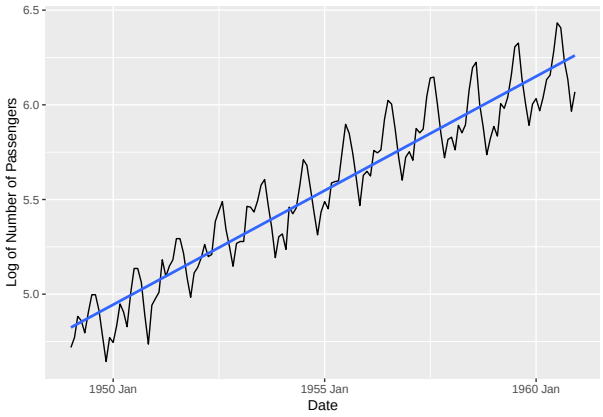
When modeling a time series, we are going to want to estimate the trend (m_t), the seasonality (s_t), and the error (e_t). When we do this, we will assume an additive log model, which is equivalent to a multiplicative model, so the data can be written as

$$\log(x_t) = \log(m_t) + \log(s_t) + \log(e_t)$$

The trend is usually the simplest to estimate. Plotting a line through the plot, we can see that there seems to be a fairly linear increase in the number of passengers. So let's use linear regression to estimate and then remove the trend.

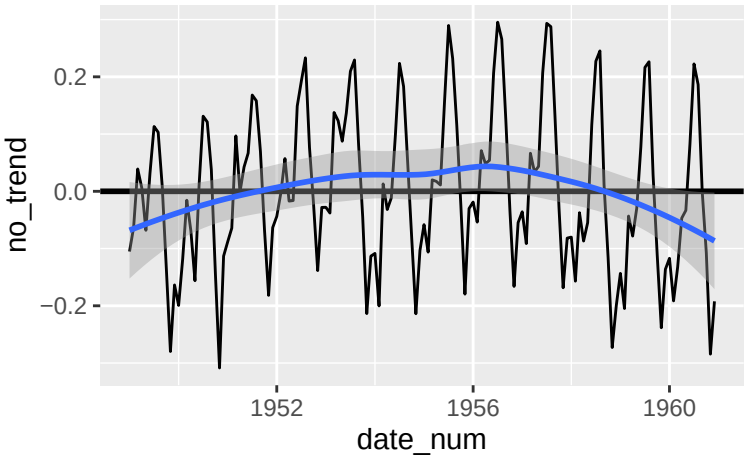
Estimating the Trend - Linear

```
ggplot(AP, aes(x = date, y = lpass)) +
  geom_line() +
  geom_smooth(method = "lm", se = F) +
  labs(
    x = "Date",
    y = "Log of Number of Passengers"
    )
```



Let's plot the de-trended data value to see how they look.

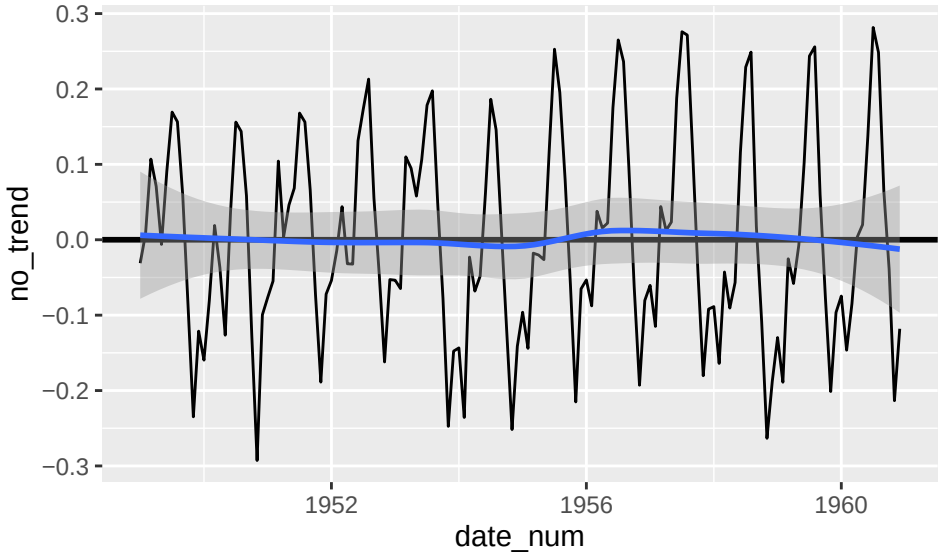
```
# Convert date to numeric
date_num <- year(AP[[1]]) + (month(AP[[1]]) - 1)/12
trend_mod <- lm(lpass ~ date_num, data = AP)
xt <- AP$num_pass
logxt <- AP$lpass
logmt <- trend_mod$fitted.values
# The de-trended values can be found with trend_mod$residuals
AP |> mutate(no_trend = logxt - logmt) |>
  ggplot(aes(x = date_num, y = no_trend)) +
  geom_line() +
  geom_hline(yintercept = 0, linewidth = 1)+
  geom_smooth()
```



Estimating the Trend - Quadratic

Let's try it again using quadratic regression:

```
trend_mod2 <- lm(lpass ~ poly(date_num, degree = 2, raw = T), data = AP)
logxt <- AP$lpass
logmt <- trend_mod2$fitted.values
# The de-trended values can also be found with trend_mod$residuals
AP |> mutate(no_trend = logxt - logmt) |>
  ggplot(aes(x = date_num, y = no_trend)) +
  geom_line() +
  geom_hline(yintercept = 0, linewidth = 1)+
  geom_smooth()
```



Removing the Trend

Now that we have removed the trend, we are still left with modeling the seasonality and error.

$$x_t = m_t + s_t + e_t$$
$$x_t - m_t = s_t + e_t$$

Now we can say that $y_t = x_t - m_t$ (what I labeled `no_trend`). This gives the model

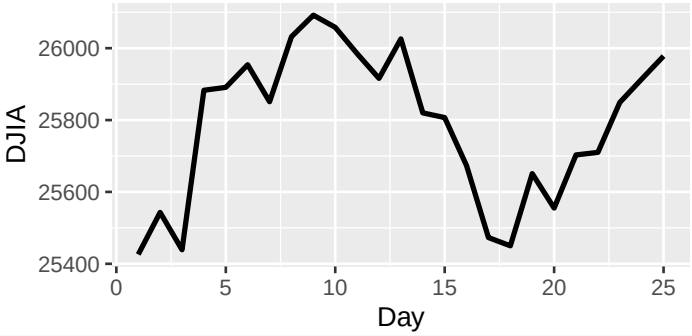
$$y_t = s_t + e_t$$

Clearly the plot on the previous slide is not linear, but there is a pattern. We need a new type of model for modeling the *seasonality*.

Other Non-linear Trends

Before we get to seasonality, however, it should be noted that many trends are not (approximately) linear. The Air Passengers was modeled decently with a linear trend, but a quadratic one was better. The Dow Jones example, as we saw earlier, does not have a linear trend at all. Let's see that one again:

We can still fit a regression model here, but it would not be linear regression. Let's use the `poly()` function to fit a cubic curve to this:

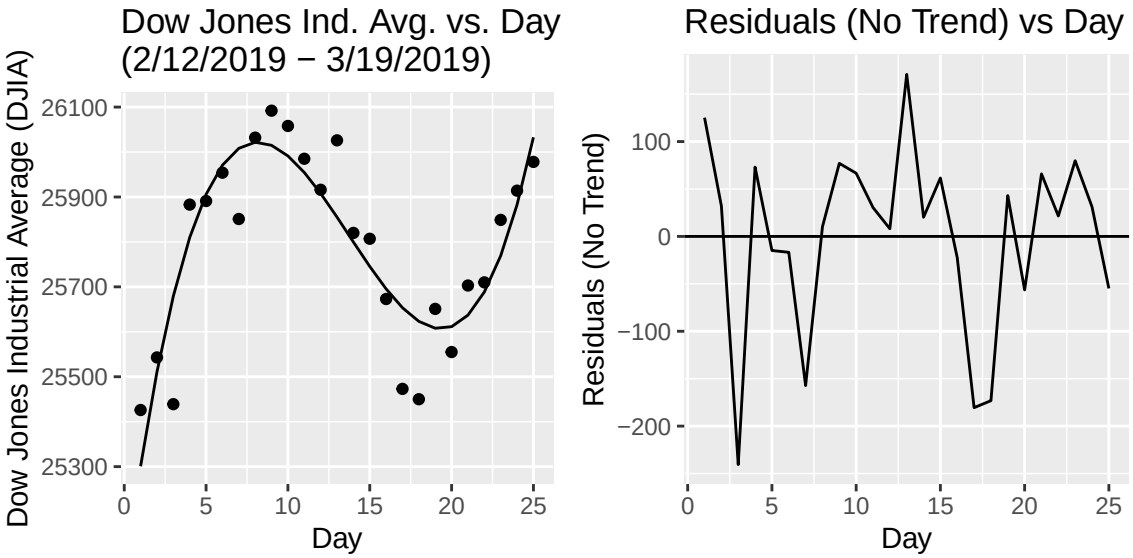



```
DJIA <- c(25426, 25543, 25439, 25883, 25891, 25954, 25851, 26032,
          26092, 26058, 25985, 25916, 26026, 25820, 25807, 25673,
          25473, 25450, 25651, 25555, 25703, 25710, 25849, 25914, 25978)
day <- 1:length(DJIA)

DJ_mod_nl <- lm(DJIA ~ poly(day, degree = 3, raw = T))
DJ_df <- tibble(DJIA, day, fitted =  DJ_mod_nl$fitted.values,
                w_t = DJIA - DJ_mod_nl$fitted.values)
p3 <- ggplot(DJ_df) +
  geom_point(aes(x = day, y = DJIA)) +
  geom_line(aes(x = day, y = fitted)) +
  labs(x = "Day", y = "Dow Jones Industrial Average (DJIA)") +
  ggtitle("Dow Jones Ind. Avg. vs. Day\n(2/12/2019 - 3/19/2019)")

p4 <- ggplot(DJ_df) +
  geom_line(aes(x = day, y = w_t)) +
  geom_hline(yintercept = 0) +
  labs(x = "Day", y = "Residuals (No Trend)") +
  ggtitle("Residuals (No Trend) vs Day")

grid.arrange(p3, p4, ncol = 2)
```



Model: $\hat{m}_t = 25025.44 + 279.1(\text{Day}) - 24.37(\text{Day})^2 + 0.5913(\text{Day})^3$

Estimating the Seasonality

Now we need to discuss how to model and remove the seasonality. Again, there are many methods for this, but the most common (and easiest) is to model it with the following:

$$\hat{s}_k = a_k - \frac{\sum_{i=1}^{d} a_i}{d}, \quad k = 1, \ldots, d$$

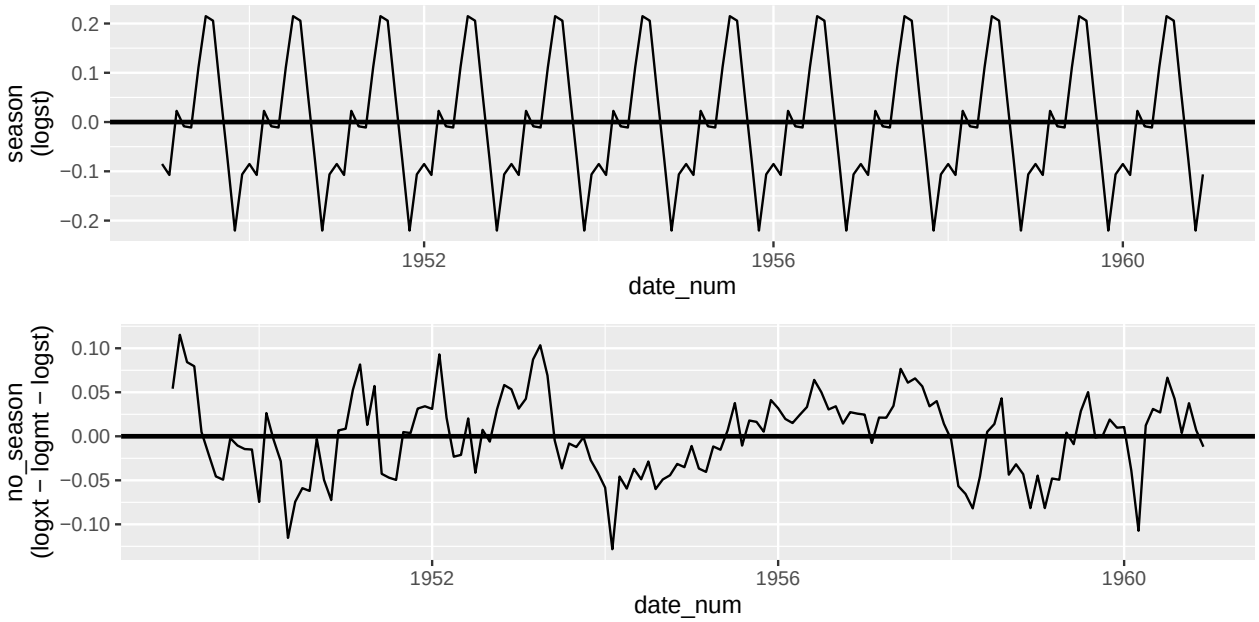
where d is the period (how many terms it takes to repeat the pattern) and a_k is the average of the deviations for the kth part of the period. Then we have $\hat{s}_k = \hat{s}_{k-d}$ for $k > d$. That is, this pattern repeats.

Note that the reason we need to subtract the mean of the a_k values, $\frac{\sum_{i=1}^{d} a_i}{d}$, is just because there is no guarantee that the average of the a_k is zero, so we subtract off the average.

Estimating the Seasonality - Air Passengers

For the air passengers example, our period, $d = 12$ since the seasonality repeats each year. Now, we will consider the de-trended data. a_1 will be the average for all the January months, a_2 will be the average for all February, and so on.

```
logyt = logxt - logmt
a1 = mean(logyt[seq(1, length(logyt), by = 12)])
a2 = mean(logyt[seq(2, length(logyt), by = 12)])
# Etc... Quicker to do this:
a <- logyt |>
  matrix(nrow = 12) |>
  rowMeans()
# Then estimate seasonality:
logst <- rep(a - mean(a), 12)
```



Time Series Models

There are three primary models for modeling the errors of a time series. The errors of a time series can also be thought of as a time series in their own right. They are just simpler since they do not have a trend or seasonality component.

Time series of this natures are called **stationary** is the mean and variance is constant across the series. There are several ways to model time series:

- Moving average (MA) models.
- Autoregressive (AR) models.
- Autoregressive moving average (ARMA) models.
- Autoregressive integrated moving average (ARIMA) models. This is for non-stationary models that need to be **differenced** to become stationary.

Moving Average (MA) Models

Moving average (MA) models of degree q are those in which the stationary time series is a weighted average of q white-noise variables. This looks like

$$X_t = Z_t + \theta_1 Z_{t-1} + \cdots + \theta_q Z_{t-q}$$

where $\{Z_t\} \sim WN(0, \sigma^2)$ and $\theta_1, \ldots, \theta_q$ are constants.

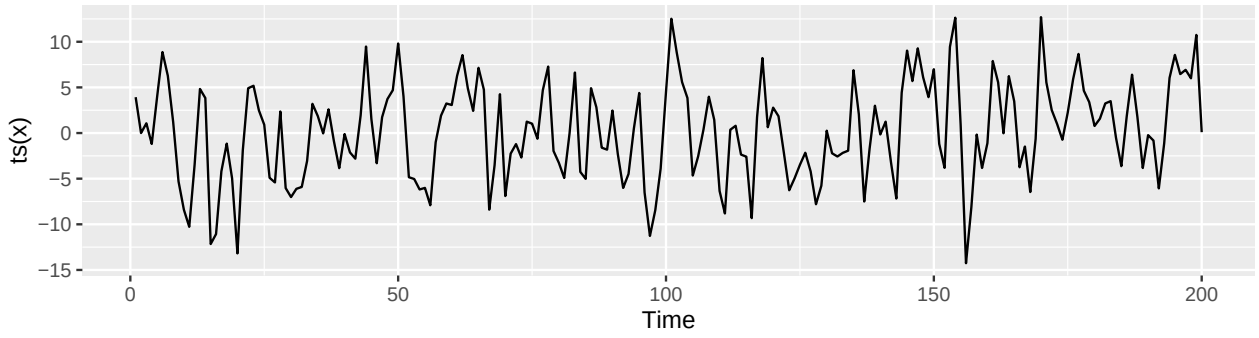
White noise is simply a term for all variables being drawn independently and identically distributed (iid) from a distribution. The most common white noise is **Gaussian white noise** where all variables come from the same normal distribution.

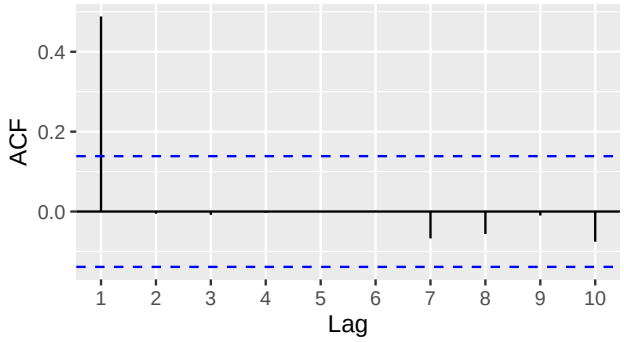
MA Model Examples - MA(1)

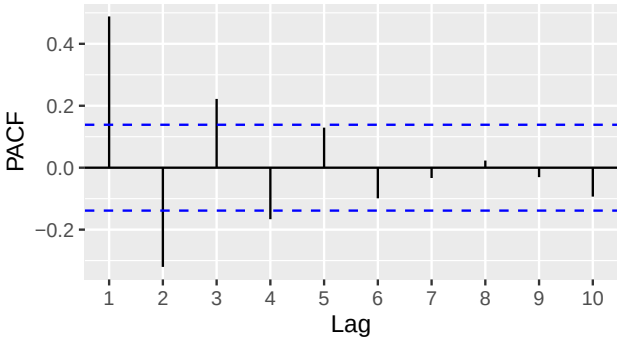
Let's simulate what a few MA time series may look like:

```
# Degree 1
n = 200
set.seed(2024)
x <- rep(0, n)
z <- rnorm(n, 0, 4) # Gaussian WN with sigma = 4
theta1 <- 0.8
x[1] <- z[1]
for (i in 3:n) {
  x[i] <- z[i] + theta1*z[i-1]
}
p1 <- autoplot(ts(x)) +
  labs(title = expr(paste(theta[1], " = ", !!theta1)))
p2 <- acf(x, plot = F, lag.max = 10) |>
  autoplot() + labs(title = "")
p3 <- pacf(x, plot = F, lag.max = 10) |>
  autoplot() +
  labs(y = "PACF", title = "")
grid.arrange(grobs = list(p1, p2, p3),
             layout_matrix = rbind(c(1, 1), c(2, 3)))
```

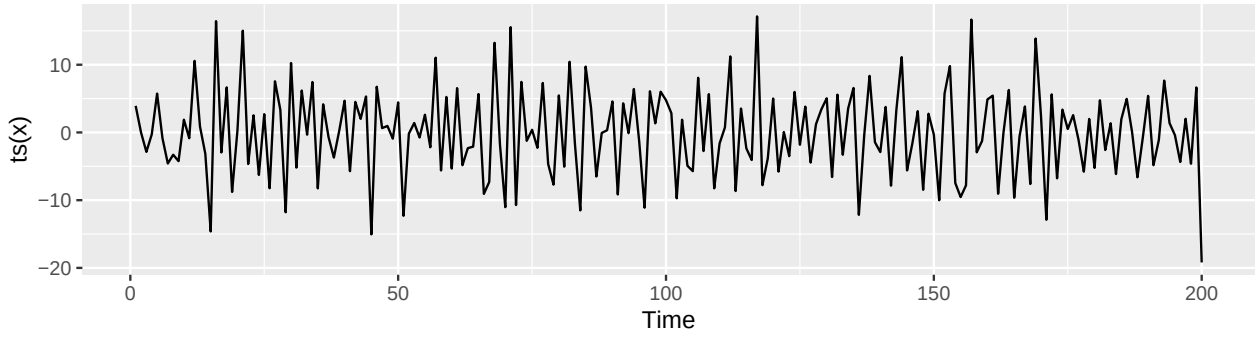
$\theta_1 = 0.8$

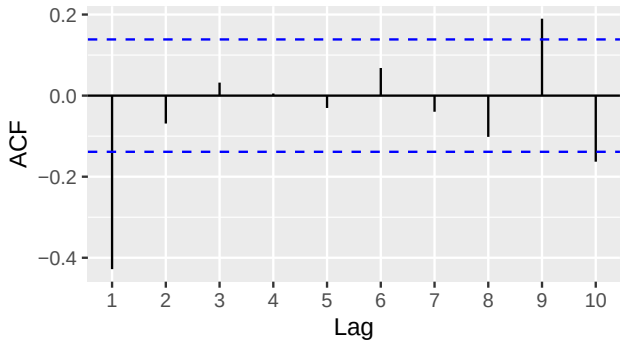


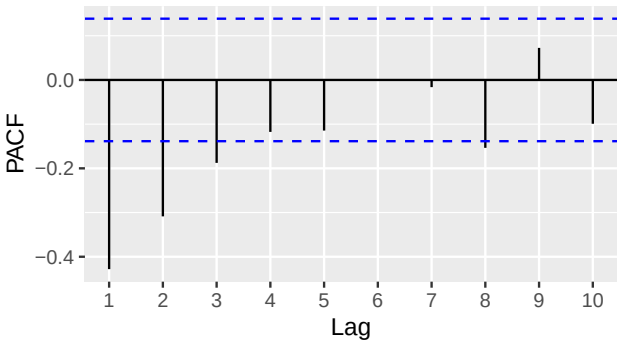




$\theta_1 = -1.3$

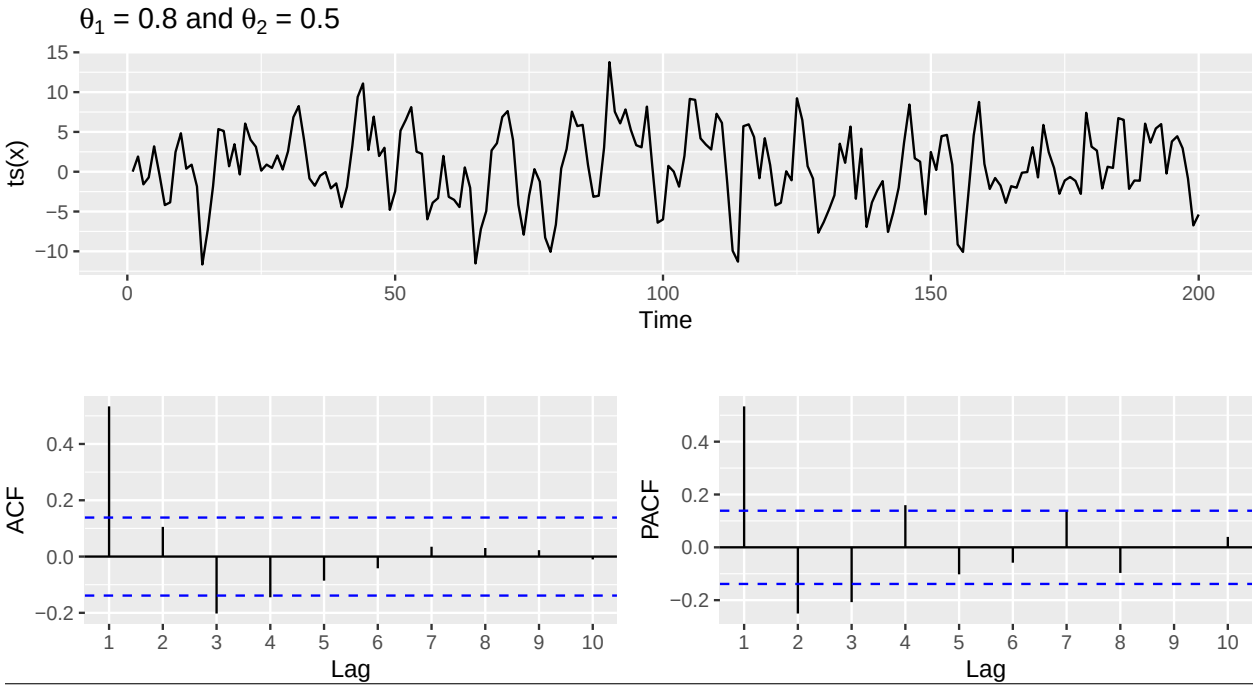


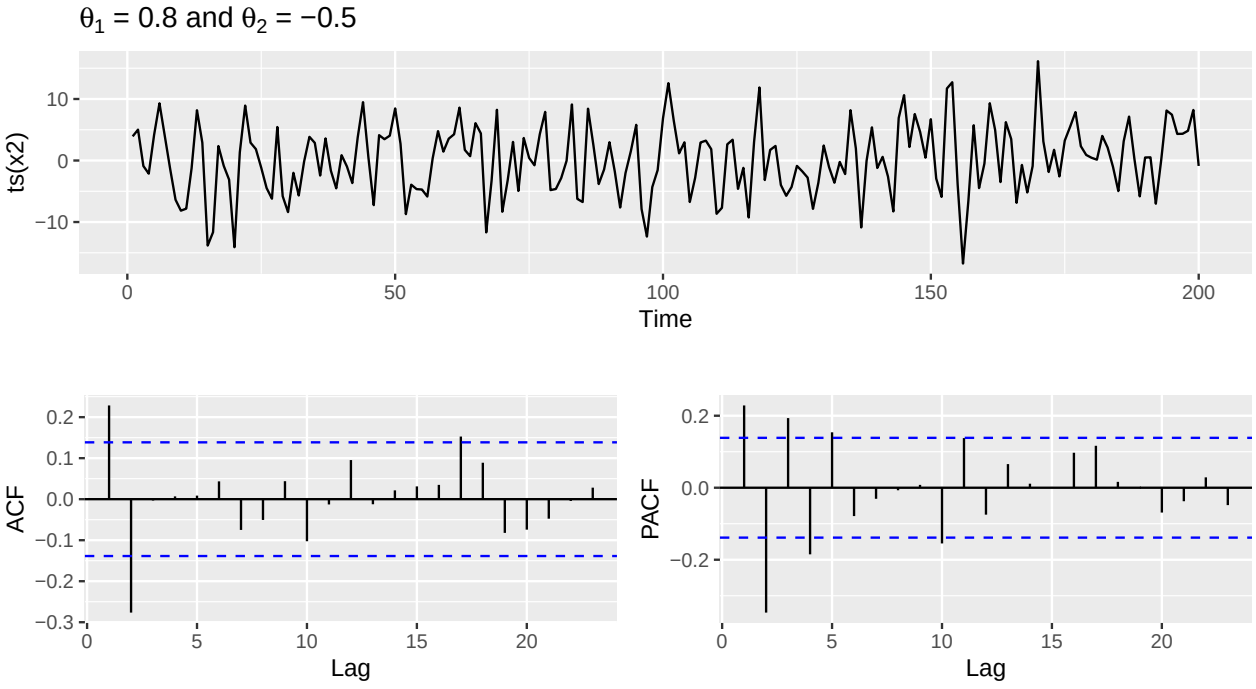




MA Model Examples - MA(2)

```
n <- 200; x <- rep(0, n); z <- rnorm(n, 0, 4) # Gaussian WN with sigma = 4
set.seed(2024)
theta1 <- 0.8; theta2 <- 0.5
x[1] <- z[1]; x[2] <- z[2] + theta1*z[1]
for (i in 3:n) {
  x[i] <- z[i] + theta1*z[i-1] + theta2*z[i-2]
}
```





Autoregressive (AR) Models

An autoregression or autoregressive model of degree p, an AR(p), is one in which the time series is a weighted average of p of its previous terms plus noise. This looks like

$$X_t = \phi_1 X_{t-1} + \phi_2 X_{t-2} + \cdots + \phi_p X_{t-p} + Z_t$$

where $\{Z_t\} \sim WN(0, \sigma^2)$ and $\phi_1, \ldots, \phi_p$ are constants.

For the AR(p) model to be stationary, we need the roots of the polynomial $1 - \phi_1 z - \cdots - \phi_p z^p$ to be outside the unit circle. That is, $|z_i| > 1$ for all i.

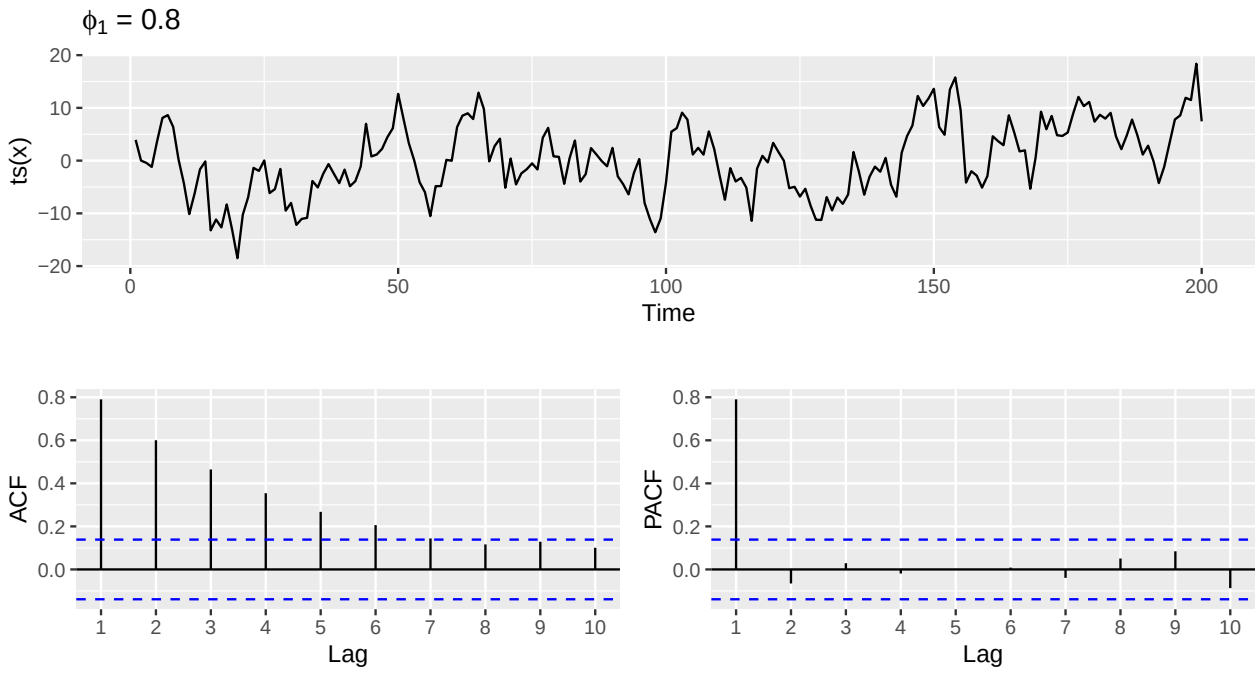
This can alternatively be written

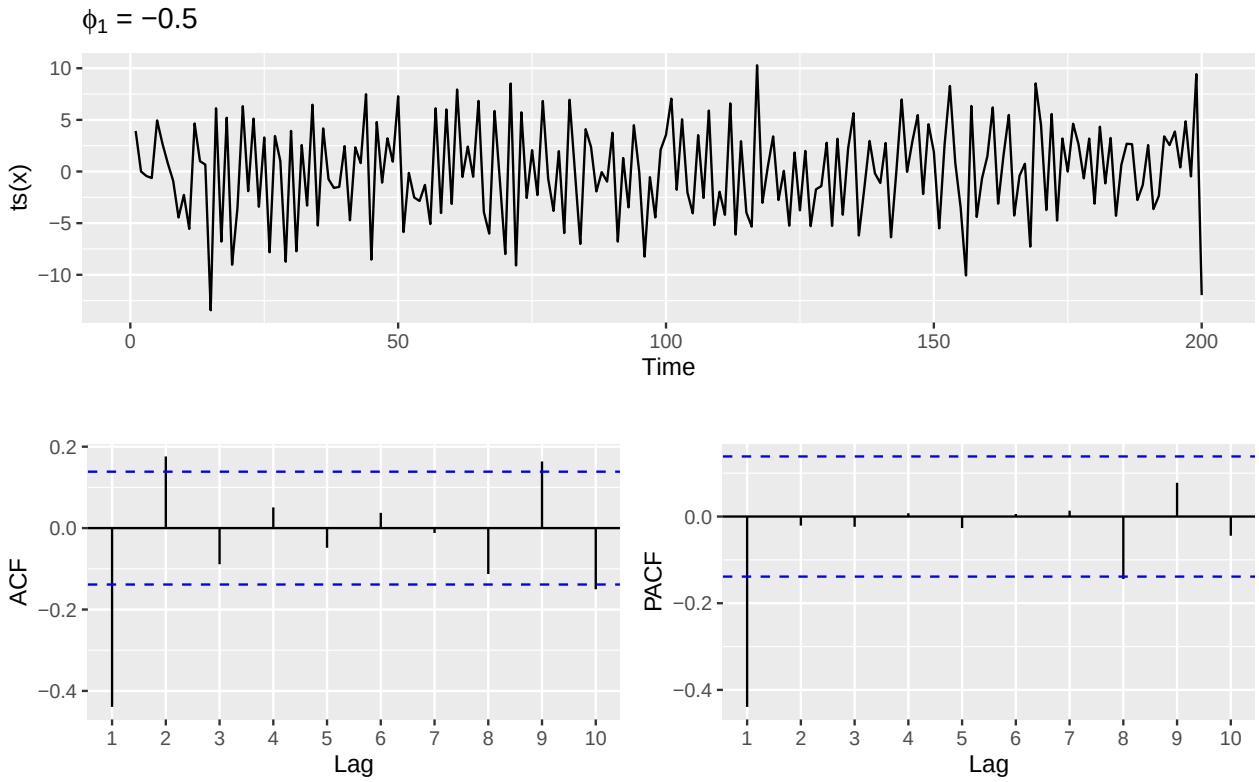
$$X_t - \phi_1 X_{t-1} - \phi_2 X_{t-2} - \cdots - \phi_p X_{t-p} = Z_t$$

AR Model Examples - AR(1)

Let's simulate what a few AR time series may look like:

```
# Degree 1
n = 200
set.seed(2024)
x <- rep(0, n)
z <- rnorm(n, 0, 4) # Gaussian WN with sigma = 4
phi1 <- 0.8
x[1] <- z[1]
for (i in 3:n) {
  x[i] <- phi1*x[i-1] + z[i]
}
```

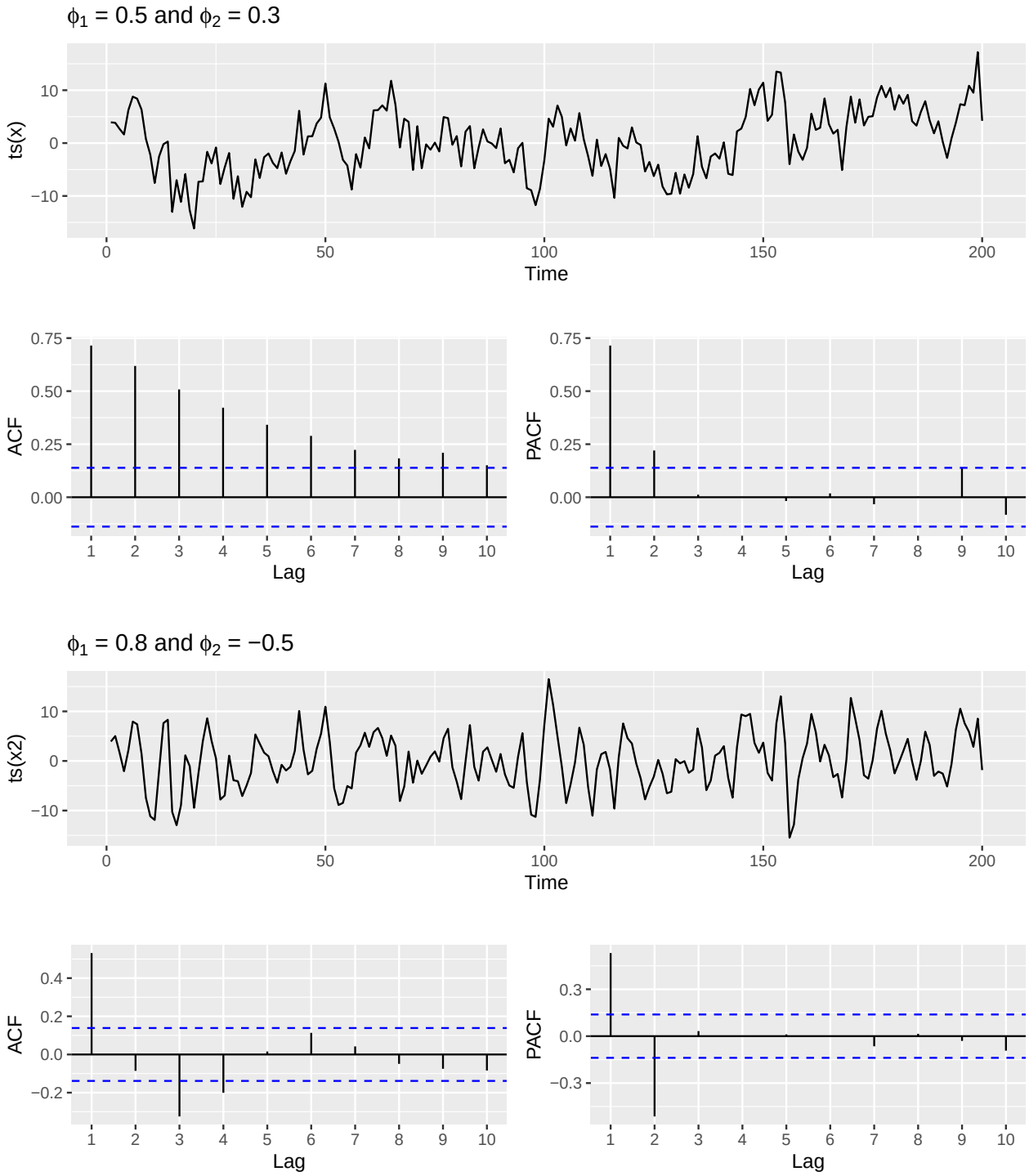




AR Model Examples - AR(2)

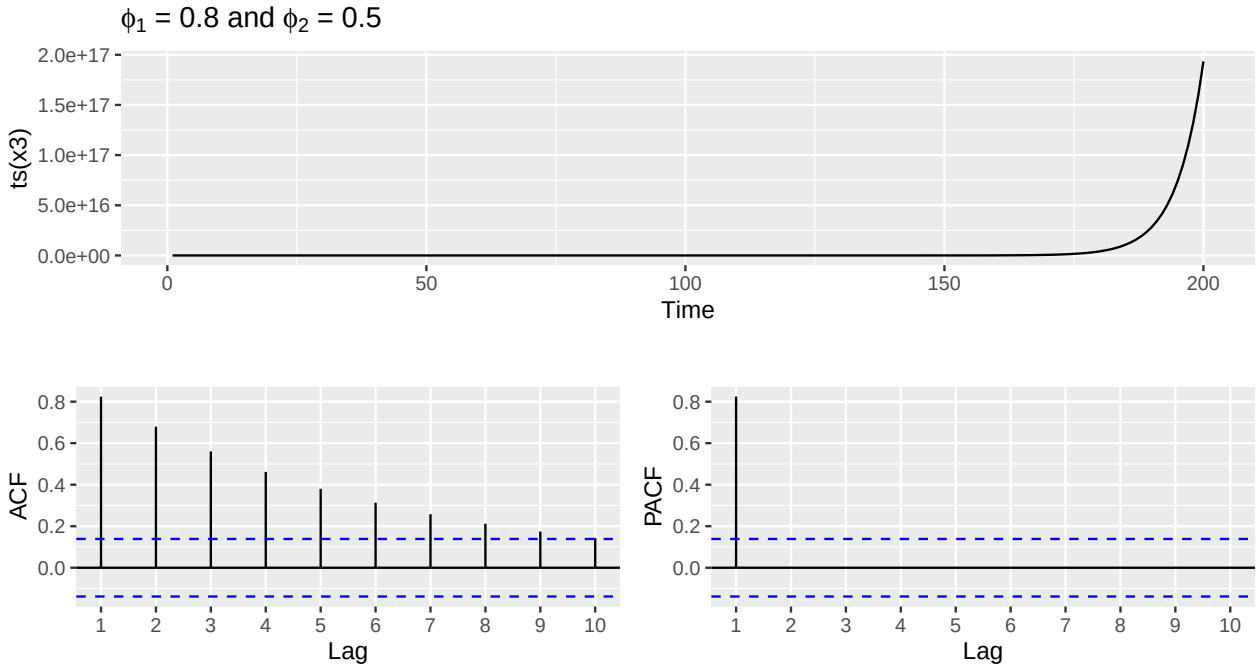
```
# Degree 2
n = 200
set.seed(2024)
x <- rep(0, n)
z <- rnorm(n, 0, 4) # Gaussian WN with sigma = 4
phi1 <- 0.5
phi2 <- 0.3
x[1] <- z[1]
x[2] <- phi1*x[1] + z[2]
for (i in 3:n) {
  x[i] <- phi1*x[i-1] + phi2*x[i-2] + z[i]
}
```

Stationary. The roots of $1 - 0.5z - 0.3z^2$ are -2.84 and 1.174, each of which is larger than 1 in absolute value.



This is also stationary. The roots of $1 - 0.8z + 0.5z^2$ are $0.8 - 1.17i$ and $0.8 + 1.17i$, each with modulus $\sqrt{0.8^2 + 1.17^2} = 1.42$, which is larger than 1.

AR Model Examples - Not Stationary



This time series is **not stationary**. The roots of $1 - 0.8z - 0.5z^2$ are -2.425 and 0.825. The second root is less than 1.

Autoregressive Moving-Average (ARMA) Models

The autoregressive moving-average model is a combination of an AR(p) and an MA(q) models. This is abbreviated ARMA(p,q). The model can be written

$$X_t = \phi_1 X_{t-1} + \cdots + \phi_p X_{t-p} + Z_t + \theta_1 Z_{t-1} + \cdots + \theta_q Z_{t-q}$$

or

$$X_t = Z_t + \sum_{i=1}^{p} \phi_i X_{t-i} + \sum_{j=1}^{q} \theta_j Z_{t-j}$$

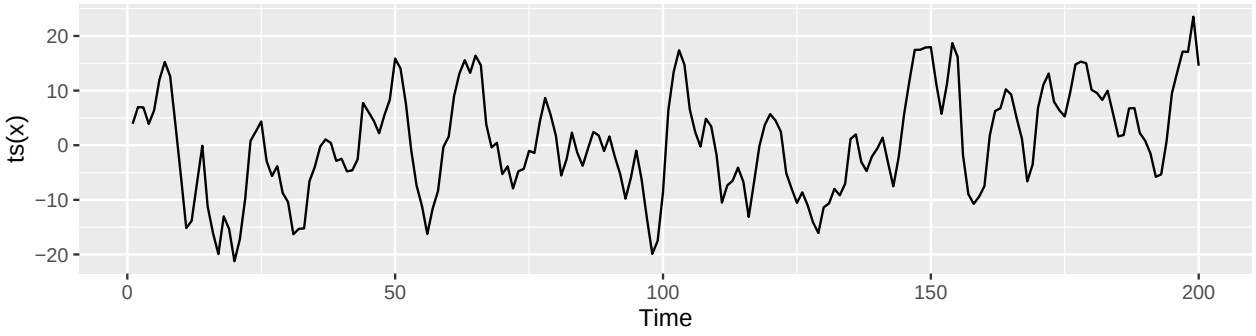
This is sometimes written as

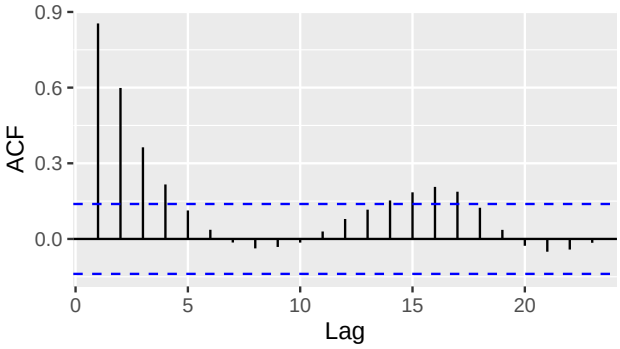
$$X_t - \phi_1 X_{t-1} - \cdots - \phi_p X_{t-p} = Z_t + \theta_1 Z_{t-1} + \cdots + \theta_q Z_{t-q}$$

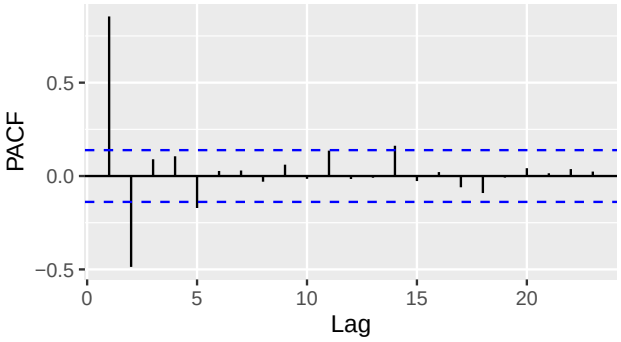
ARMA Examples - ARMA(2,2)

```
n <- 200; set.seed(2024)
x <- rep(0, n); z <- rnorm(n, 0, 4) # Gaussian WN with sigma = 4
phi1 <- 0.5; phi2 <- 0.1; theta1 <- 0.8; theta2 <- 0.5
x[1] <- z[1]; x[2] <- phi1*x[1] + z[2] + theta1*z[1]
for (i in 3:n) {
  x[i] <- phi1*x[i-1] + phi2*x[i-2] + z[i] +
    theta1*z[i-1] + theta2*z[i-2]
}
```

$\phi_1 = 0.5$, $\phi_2 = 0.1$, $\theta_1 = 0.8$, and $\theta_2 = 0.5$

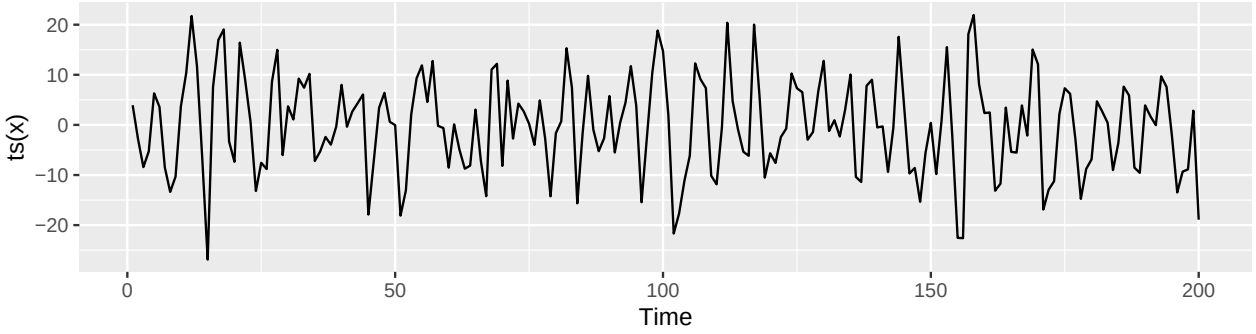


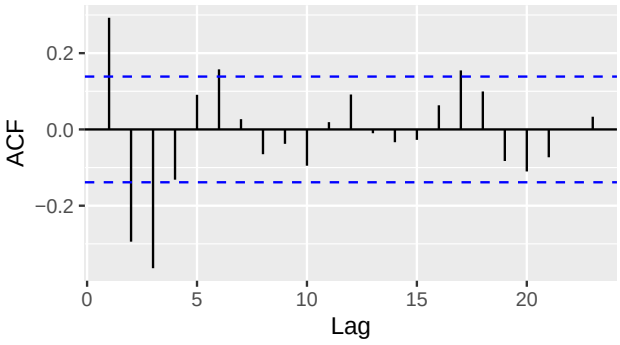


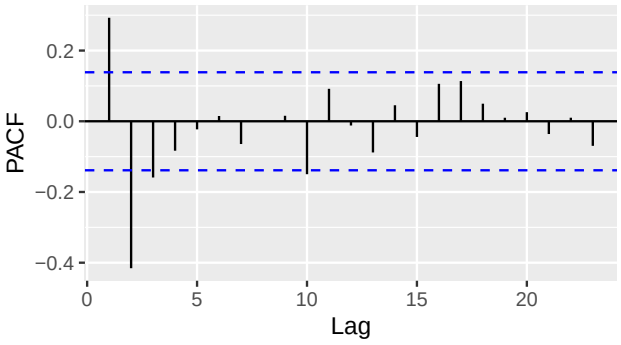


ARMA Examples - ARMA(2,1)

$\phi_1 = 0.8$, $\phi_2 = -0.5$, and $\theta_1 = -2$

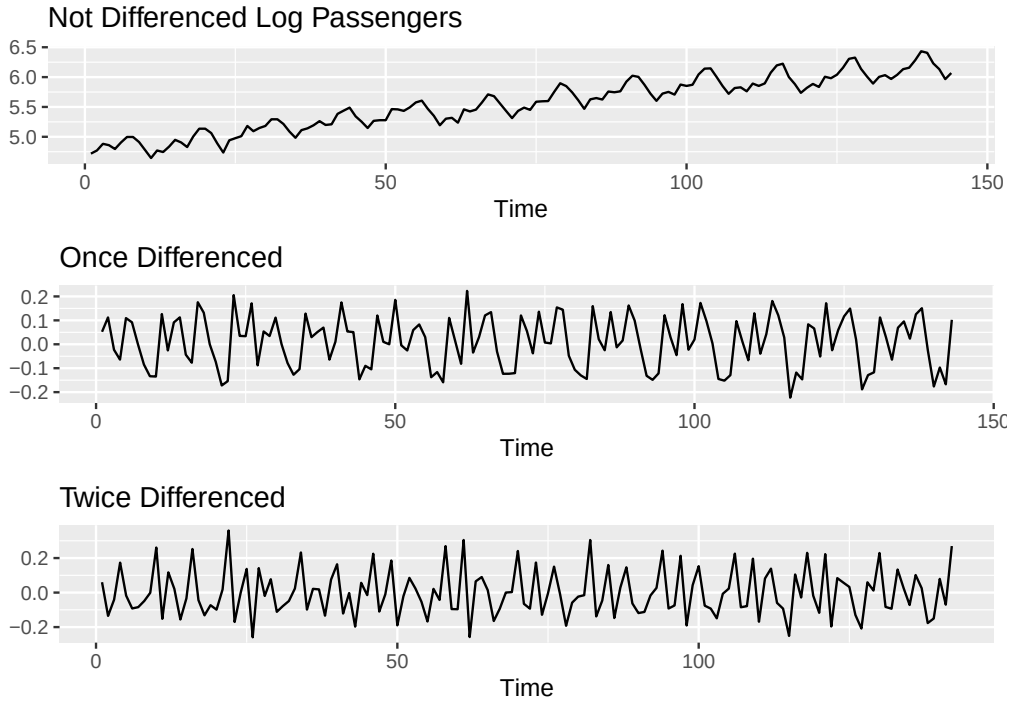






Differencing

Sometimes differencing a time series can make it stationary and make the model fit better. Consider the Air Passengers data set with and without differencing.



Fitting an ARMA Model

Of course, our goal is to try to model the errors by fitting an ARMA model to them. Then we can add the seasonality and trend back to get a model we can use for **forecasting**.

There are three things we should do when fitting an ARMA model to a time series:

1. Make sure the resulting series is likely stationary.
2. Determine the order of the ARMA(p,q) process. That is, determine p and q.
3. Determine the proper ϕ and θ value(s). This is most commonly done with maximum likelihood, but we will not get into that here.

Determining if Series is Stationary

A common test for stationarity is the augmented Dickey-Fuller test. This test can be written as

$$H_0 : \text{The time series is not stationary}$$
$$H_A : \text{The time series is stationary}$$

So, we would like to reject the null hypothesis here.

Testing for Stationarity

We can use the `adf.test()` function in the `tseries` library to test this. If there is seasonality in the time series, then it is best to set `k` equal to the seasonality period. We have removed the trend and seasonality in the Air Passengers time series, so we will leave this as the default.

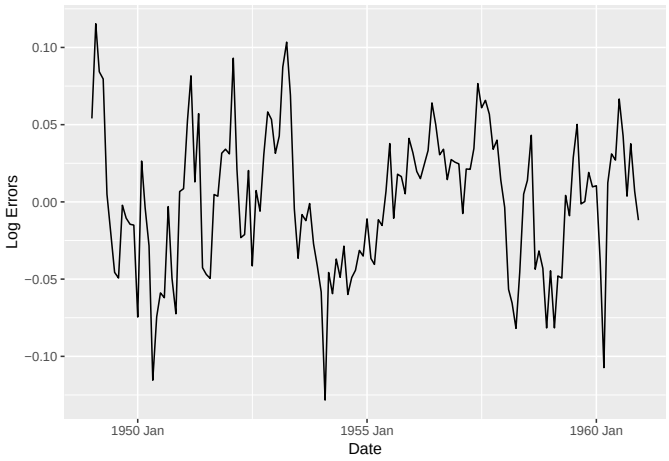
```
library(tseries)
loget <- logxt - logmt - logst
adf.test(loget)
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  loget
## Dickey-Fuller = -3.555, Lag order = 5, p-value = 0.0399
## alternative hypothesis: stationary
```

Fitting an ARMA Model

Let's take another look at the log error data set.

```
AP2 <- AP |>
  mutate(logxt, logmt, logst, loget)
ggplot(
  AP2,
  aes(x = date, y = loget)
  ) +
  geom_line() +
  labs(x = "Date", y = "Log Errors")
```



We can fit an ARMA model to the data using the `arima()` function in the `stats` library that is already pre-loaded into **R**.

Note: the ARIMA is a slightly different model than the ARMA. The ARIMA allows for the data to be differenced to make it stationary if it isn't already. Ours already is.

In the `arima()` function, we need to specify the order we want to fit. This should be a vector with three arguments: (p, d, q). p is the order of the AR part of the model, d is the number of times the time series should be differenced to make it stationary, and q is the order of the MA part of the model.

Let's start with fitting an AR(1) model. Setting `include.mean = F` will omit an intercept term.

```
# Fit an AR(1)
arima(loget, order = c(1, 0, 0), include.mean = F)
```

```
##
## Call:
## arima(x = loget, order = c(1, 0, 0), include.mean = F)
##
## Coefficients:
##          ar1
##       0.6734
## s.e.  0.0612
##
## sigma^2 estimated as 0.001143:  log likelihood = 283.08,  aic = -562.15
```

```r
# Fit an MA(2)
arima(loget, order = c(0, 0, 2), include.mean = F))
# Fit an ARMA(2,1)
arima(loget, order = c(2, 0, 1), include.mean = F))
```

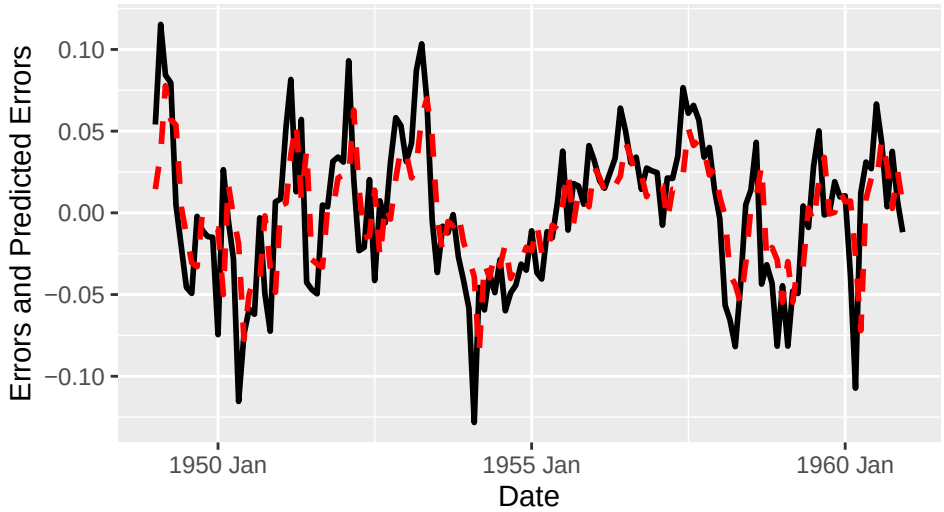
Model Fit Table

Model	AIC	BIC
AR(1)	-562.16	-556.21
AR(2)	-560.91	-552.00
AR(3)	-560.79	-548.91
MA(1)	-530.76	-524.82
MA(2)	-550.02	-541.11
MA(3)	-566.16	-554.29
ARMA(1, 1)	-560.71	-551.80
ARMA(2, 1)	-559.38	-547.50
ARMA(1, 2)	-562.41	-550.54
ARMA(2, 2)	-560.88	-546.03
ARMA(3, 2)	-564.18	-546.37
ARMA(2, 3)	-562.40	-544.58
ARMA(3, 3)	-562.19	-541.40

We can obtain these figures using the `AIC()` and `BIC()` functions. Not `extractAIC()`.

Using an AR(1) model, let's plot the errors and the modeled values:

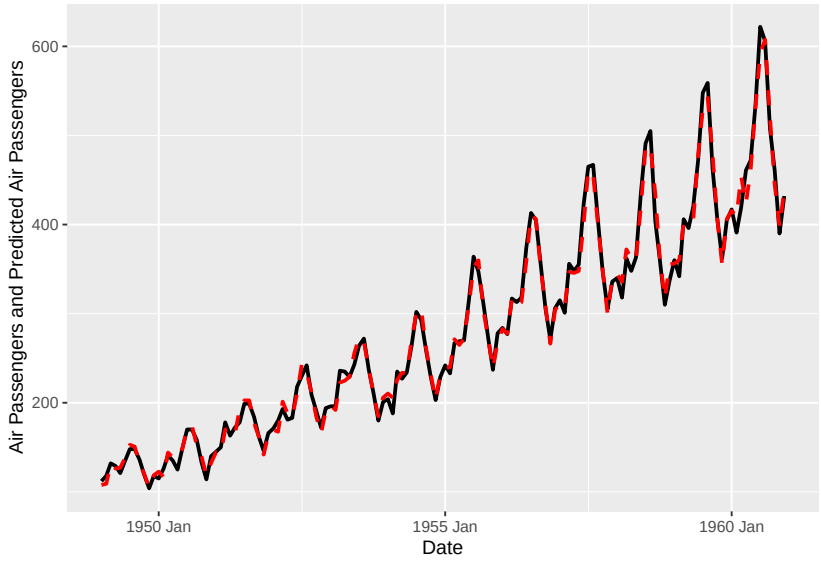
```r
library(forecast) # Allows us to get fitted values from arima model
fit_lerror <- arima(loget, order = c(1, 0, 0), include.mean = T)
AP2 |> mutate(fitted = fitted.values(fit_lerror)) |>
  ggplot() +
  geom_line(aes(x = date, y = loget), linewidth = 1) +
  geom_line(aes(x = date, y = fitted), color = "red",
            linetype = "dashed", linewidth = 1) +
  labs(x = "Date", y = "Errors and Predicted Errors")
```



Transforming Back

Now we can transform all the way back!

```
predicted_series <- exp(logmt + logst +
                          fitted.values(fit_lerror))
AP2 |>
  mutate(pred = predicted_series) |>
  ggplot() +
  geom_line(aes(x = date, y = num_pass), linewidth = 1) +
  geom_line(aes(x = date, y = pred), color = "red",
            linetype = "dashed", linewidth = 1) +
  labs(x = "Date",
       y = "Air Passengers and Predicted Air Passengers")
```

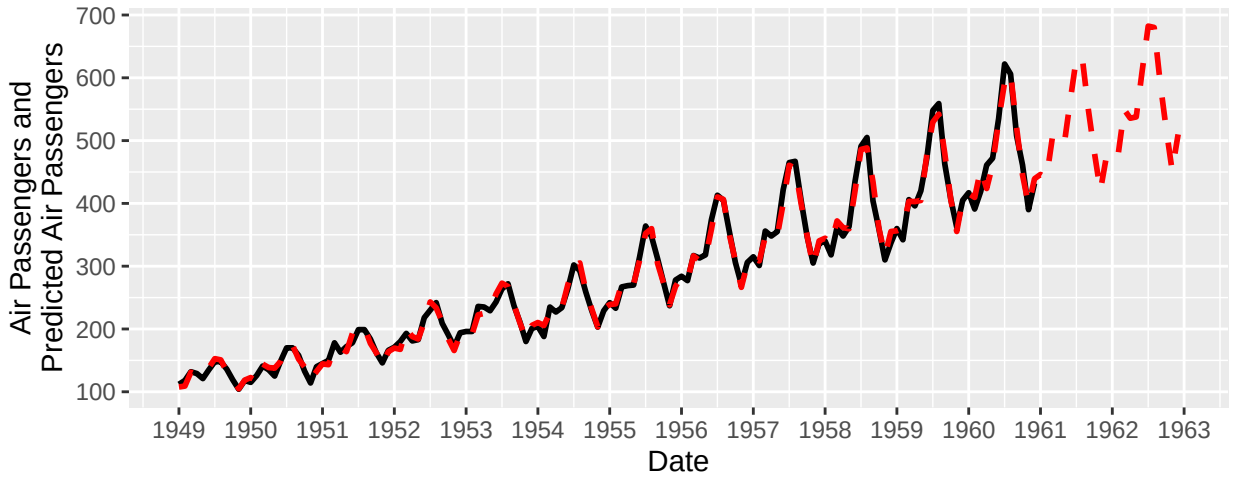


Forecasting

Now that we have everything modeled, we can use this model to predict future values of our time series. We will need the `forecast()` function in the `forecast` library **R** package for this.

```
ltrend_pred <- predict(
  trend_mod2, data.frame(date_num = seq(1961, 1962.95, by = 1/12))
  )
lseason_pred <- logst[1:24] # Seasonality repeats
# h controls how many points in the future are forecasted
lerror_pred <- as.vector(
  forecast(fit_lerror, level = 95, h = 24)$mean
  )
forecasted_series <- exp(ltrend_pred + lseason_pred + lerror_pred)
AP3 <- tibble(newdate = c(date_num, seq(1961, 1962.95, by = 1/12)),
              pred = c(predicted_series, forecasted_series),
              num_pass = c(AP2$num_pass, rep(NA, 24)))
```

```
ggplot(AP3) +
  geom_line(aes(x = newdate, y = num_pass), linewidth = 1) +
  geom_line(aes(x = newdate, y = pred), color = "red",
            linetype = "dashed", linewidth = 1) +
  labs(x = "Date",
       y = "Air Passengers and\nPredicted Air Passengers") +
  scale_x_continuous(breaks = 1948:1963) +
  scale_y_continuous(breaks = seq(100, 700, by = 100))
```



Forecasting Confidence Intervals

Of course, it is always the case that predicting future values comes with error. It would be nice to get some confidence intervals for these new predictions.

The way we just walked through this example is also a bit clunky. We had to model each piece separately and then put it back together.

We can solve both of these issues by fitting a model to the original time series using the `auto.arima()` function in the `forecast` package.

`auto.arima()` Function

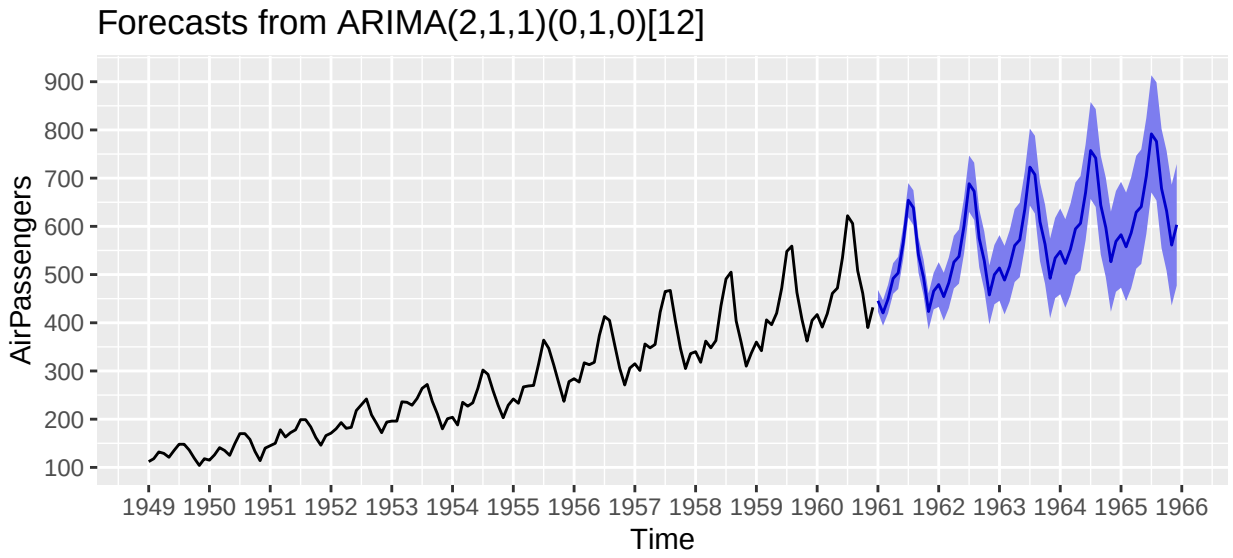
```
AP_mod <- auto.arima(AirPassengers)
AP_mod
```

```
## Series: AirPassengers
## ARIMA(2,1,1)(0,1,0)[12]
##
## Coefficients:
##          ar1     ar2      ma1
##       0.5960  0.2143  -0.9819
## s.e.  0.0888  0.0880   0.0292
##
## sigma^2 = 132.3:  log likelihood = -504.92
## AIC=1017.85   AICc=1018.17   BIC=1029.35
```

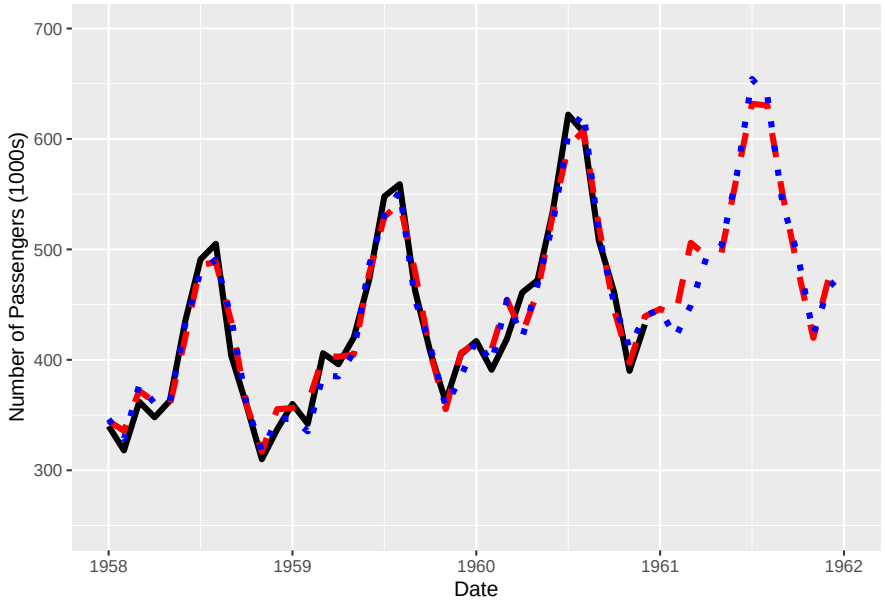
Forecasting Confidence Intervals

```
# Predict 5 years out
AP_forecast <- forecast(AP_mod, level = 95, h = 60)

# Predict 5 years out
autoplot(AP_forecast) +
  scale_x_continuous(breaks = 1948:1966) +
  scale_y_continuous(breaks = seq(100, 900, 100))
```



Compare this new forecast (with the blue dotted line) to the original forecast (with the red dashed line). The black line is the actual values.



Diagnostic Plots

If the model is a good fit, then the residuals of the model should act like random noise. That is, the residuals should not be correlated. In addition, the residuals should be normal since we used the `auto.arima()` function that uses maximum likelihood estimation for choosing the parameters. That function requires the residuals be normal. So, many of the plots for regression diagnostics apply here. That is, we should check the following:

- A residual plot.
  - Should not have patterns or fanning.
- An autocorrelation plot for the residuals.
  - The correlations should be within or close to the intervals bands for all lags greater than zero.
- A QQ plot of the residuals.
  - Should be close to the line.

```
resids <- AP_mod$residuals
p <- autoplot(resids) +
  labs(y = "Residuals")
p2 <- acf(resids, plot = F) |>
  autoplot()
p3 <- ggplot(data.frame(resids), aes(sample = resids)) +
  geom_qq() +
  geom_qq_line(linewidth = 1)
grid.arrange(p, p2, p3, nrow = 3)
```