Time Series Analysis in R

Endri Raco

2024-09-28

Introduction

Endri Raço

Course Objective

Course Objective: Understanding the key concepts of time series, summary statistics, trends, and visual representation.

Key Skills

- Retrieving summary statistics for time series data
- Recognizing trends in time series
- Using window functions to focus on specific time periods
- Imputing missing values with constant fill, LOCF (Last Observation Carried Forward), and linear interpolation

Let's get started

What Is Time Series Data?

What is Time Series Data?

Let's begin by defining a *time series*.

- A **time series** is a collection of data points ordered sequentially over time.
- Observations are recorded at regular intervals (e.g., daily, monthly, yearly).

Endri Raço

What is Time Series Data?

- Time series datasets are ubiquitous in the real world – a time series analysis studies how **a variable changes over time**, rather than **only measuring the variable at different points** in time.

What is Time Series Data?

- Let's look at some real-world uses of time series!

What is Time Series Data?

- One field that makes frequent use of time series data is *marketing and analytics* — for example, the monthly count of passengers in an international airline.

Endri Raço

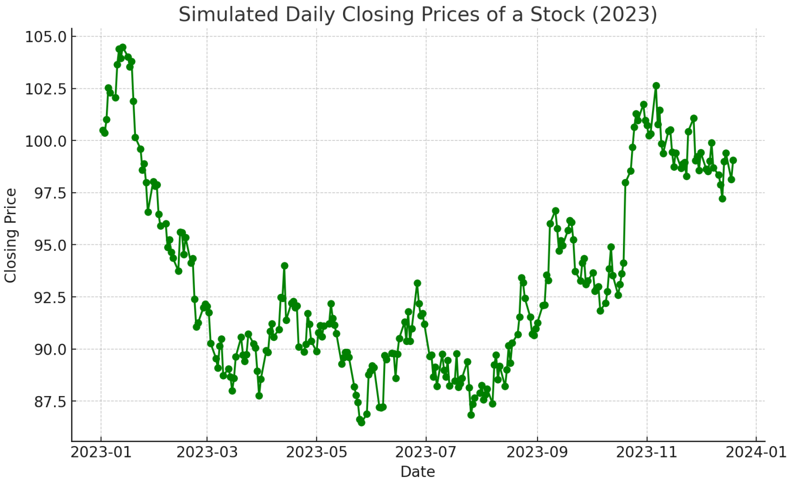
What is Time Series Data?

Endri Raço

What is a time series?

- Another example is **finance** — the closing price of a particular stock market each business day.

What is Time Series Data?

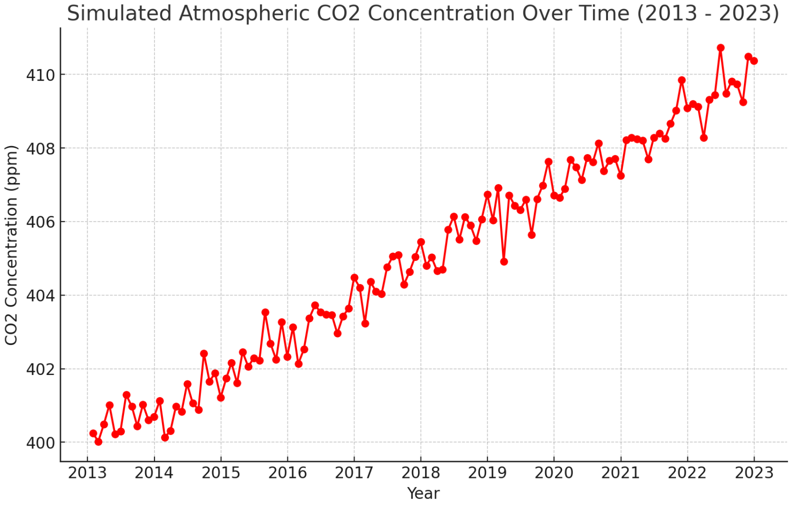


Endri Raço

What is a time series?

- There's also **scientific research** — this last graph shows the concentration of carbon dioxide in the atmosphere over time.

What is Time Series Data?



Endri Raço

Time Series Overview

- The first step in time series analysis is visualizing the dataset.
- The `print()` function:
- Displays the **Start**, **End**, and **Frequency** of your data.
- Also shows the observations.

Endri Raço

Checking Dataset Size

- Use `length()` to get the total number of observations.
- Ideal for long datasets where full preview is unnecessary.

Previewing Subsets

- For subset previews, use:
- $head(data, n =)$ to view the first `n` elements.
- $tail(data, n =)$ to view the last `n` elements.

Example: Nile Dataset

```r
print(Nile)
```

Endri Raço

Example: Nile Dataset

```
Time Series:
Start = 1871
End = 1970
Frequency = 1
  [1] 1120 1160  963 1210 1160 1160  813 1230 1370 1140   99
 [16]  960 1180  799  958 1140 1100 1210 1150 1250 1260 122
 [31]  874  694  940  833  701  916  692 1020 1050  969   83
 [46] 1120 1100  832  764  821  768  845  864  862  698   84
 [61]  781  865  845  944  984  897  822 1010  771  676   64
 [76] 1040  860  874  848  890  744  749  838 1050  918   98
 [91] 1020  906  901 1170  912  746  919  718  714  740
```

Previewing Subsets

‘- The **Nile** dataset contains annual streamflow data.

- `print(Nile)` output:
  - **Start** = 1871 (first observation year).
  - **End** = 1970 (last observation year).

Number of Observations

- Use `length()` to list the number of observations.

```
length(Nile)
```

```
[1] 100
```

Previewing Data: First 10 Elements

- Use `head()` to display the first 10 elements of the Nile dataset.

```r
head(Nile, n = 10)
```

```
 [1] 1120 1160  963 1210 1160 1160  813 1230 1370 1140
```

Previewing Data: Last 12 Elements

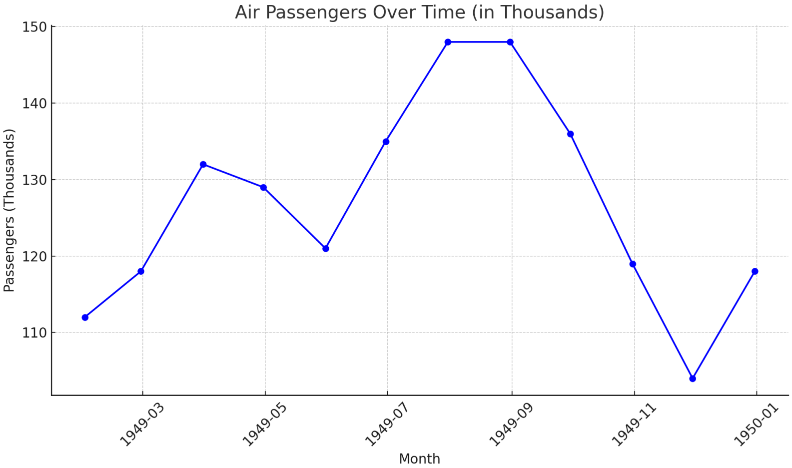
- Use `tail()` to display the last 12 elements of the dataset.

```r
tail(Nile, n = 12)
```

```
 [1]  975  815 1020  906  901 1170  912  746  919  718  71
```

Time series in R

Here's a plot of AirPassengers, representing passengers in thousands.



Endri Raço

Summary statistics

- Our dataset tracks monthly passengers from January 1949 to December
  1960.
- The first observation has 112 thousand passengers, the next has 118 thousand, and so on.

Summary statistics

Let's get some more information about the dataset.

- Calling the **summary** function, we get some summary statistics about the dataset, including the mean, median, and quartiles.

Endri Raço

Summary statistics

```
summary(AirPassengers)
```

Endri Raço

Summary statistics

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
    104     180     266     280     360     622
```

Exploring Time Series Data

- Commands like `summary()`, `print()`, `length()`, `head()`, and `tail()` help explore raw time series data.
- A more visual approach to data exploration is plotting.

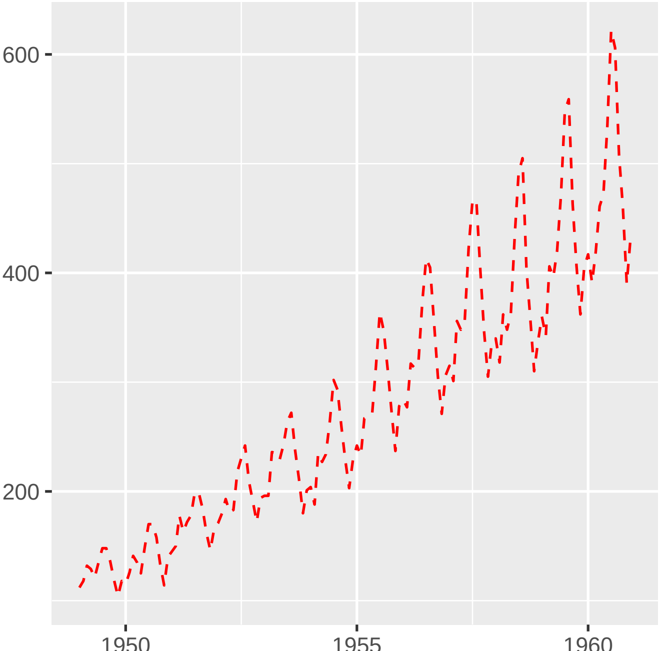
Example: AirPassengers Dataset

- Monthly total international airline passengers (in thousands) from **1949 to 1960**.

Plotting with autoplot

```r
library(ggfortify)
autoplot(AirPassengers, ts.colour = "red", ts.linetype = "dashed")
```

Plotting with autoplot



ui Raço

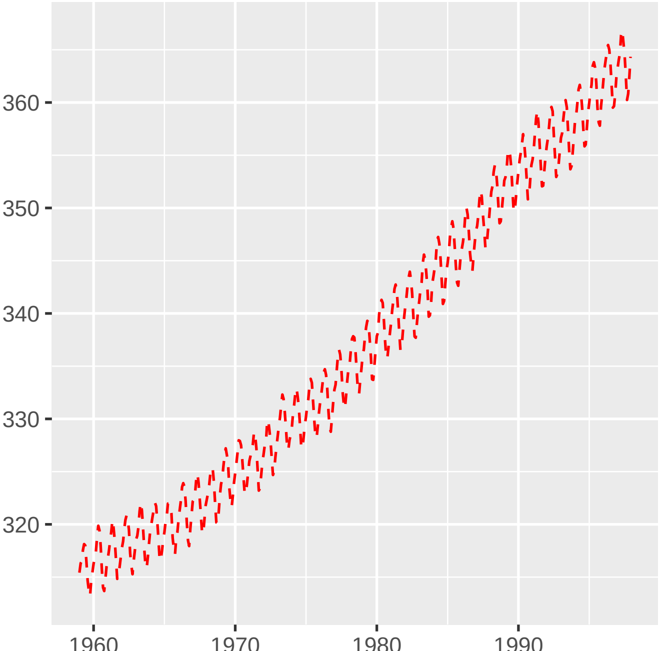
Practice

Generate an autoplot of the **co2** time series.

Solution

```r
library(ggfortify)
autoplot(co2, ts.colour = "red", ts.linetype = "dashed")
```

Solution



ui Raço

Practice

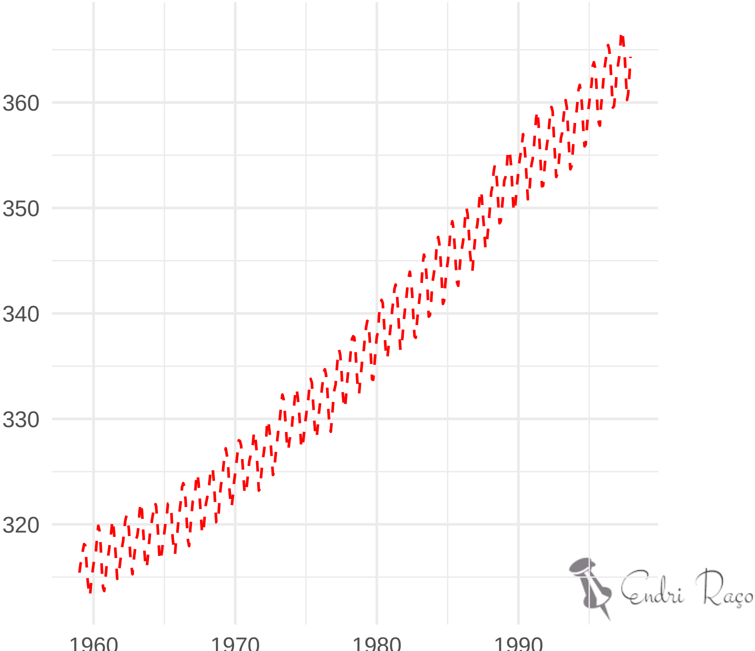
Create a second autoplot, using the minimal theme from ggplot2

Solution

```r
library(ggfortify)
autoplot(co2, ts.colour = "red", ts.linetype = "dashed") + theme_minimal()
```

Endri Raço

Solution



Practice

Return the summary statistics from the **co2** time series.

Solution

```r
# Determine the summary statistics of co2
summary(co2)
```

Endri Raço

Solution

```
Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 313     324     335     337     350     367
```

Practice

- When visualizing time series data, it is vital to be able to interpret the different features and attributes of the plots of your data.
- Determining the properties of your data from a plot allows you to better prepare the tools needed in your analysis.
- In this exercise, you'll interpret some of the properties of the **ftse** time series, based on its plot.

Endri Raço

Practice

Raço

True or False

The axis labeled ‘Index’ provides the time of each observation.

True or False

Within each year, there is only a single observation of data.

True or False

This plot shows an upwards trend in the data

True or False

There is little ‘noise’ in the data; the plot is very smooth line

Endri Raço

True or False

The average value of data is approximately 3500

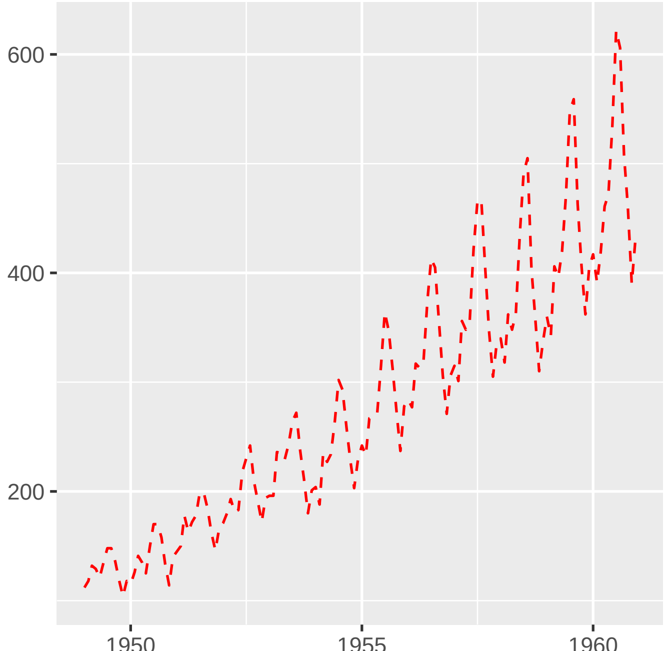
True or False

The smallest value of data is approximately 5000

Plotting with autoplot

```r
library(ggfortify)
autoplot(AirPassengers, ts.colour = "red", ts.linetype = "dashed")
```

Plotting with autoplot



ui Raço

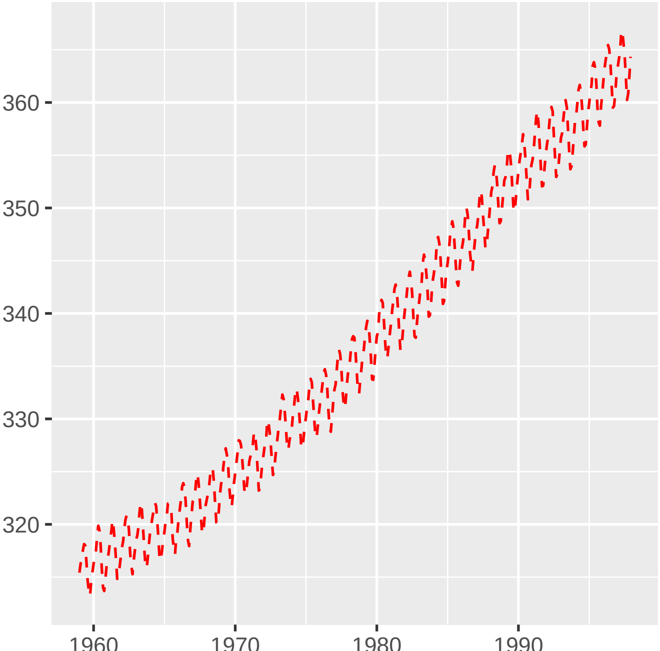
Practice

Generate an autoplot of the **co2** time series.

Solution

```r
library(ggfortify)
autoplot(co2, ts.colour = "red", ts.linetype = "dashed")
```

Solution



ui Raço

Practice

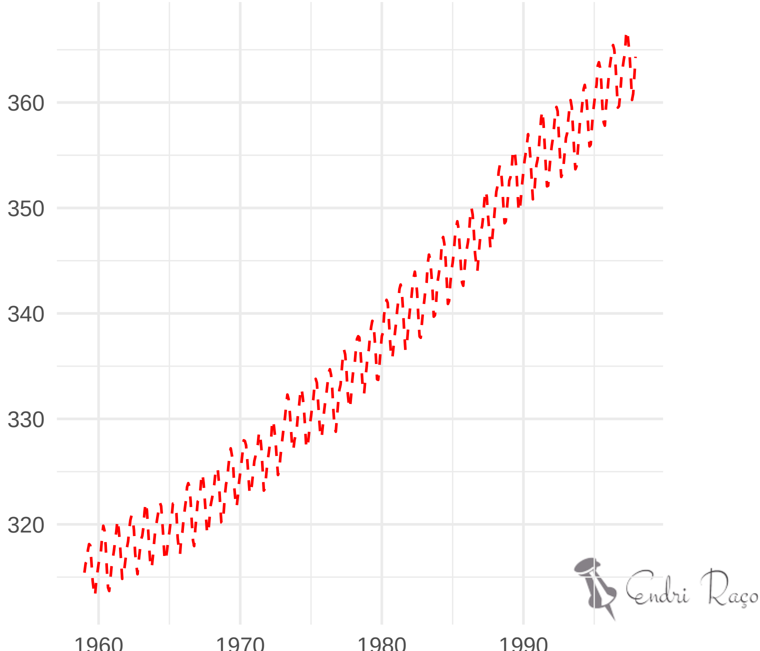
Create a second autoplot, using the minimal theme from ggplot2

Solution

```r
library(ggfortify)
autoplot(co2, ts.colour = "red", ts.linetype = "dashed") + theme_minimal()
```

Endri Raço

Solution



Practice

Return the summary statistics from the **co2** time series.

Solution

```r
# Determine the summary statistics of co2
summary(co2)
```

Solution

```
Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 313     324     335     337     350     367
```

Practice

- When visualizing time series data, it is vital to be able to interpret the different features and attributes of the plots of your data.
- Determining the properties of your data from a plot allows you to better prepare the tools needed in your analysis.
- In this exercise, you'll interpret some of the properties of the **ftse** time series, based on its plot.

Practice

Raço

True or False

The axis labeled ‘Index’ provides the time of each observation.

True or False

Within each year, there is only a single observation of data.

True or False

This plot shows an upwards trend in the data

True or False

There is little ‘noise’ in the data; the plot is very smooth line

True or False

The average value of data is approximately 3500

True or False

The smallest value of data is approximately 5000

Sampling frequency

Additional Operations on Time Series Data

- Beyond visualization, time series analysis involves several operations:
  - **start()**: returns time index of the first observation.
  - **end()**: returns time index of the last observation.
  - **time()**: computes vector of time indices for the entire series.

Time Interval & Frequency Functions

- **deltat()**: Returns the fixed time interval between observations.
- **frequency()**: Number of observations per unit time.
- **cycle()**: Position of each observation in the cycle.

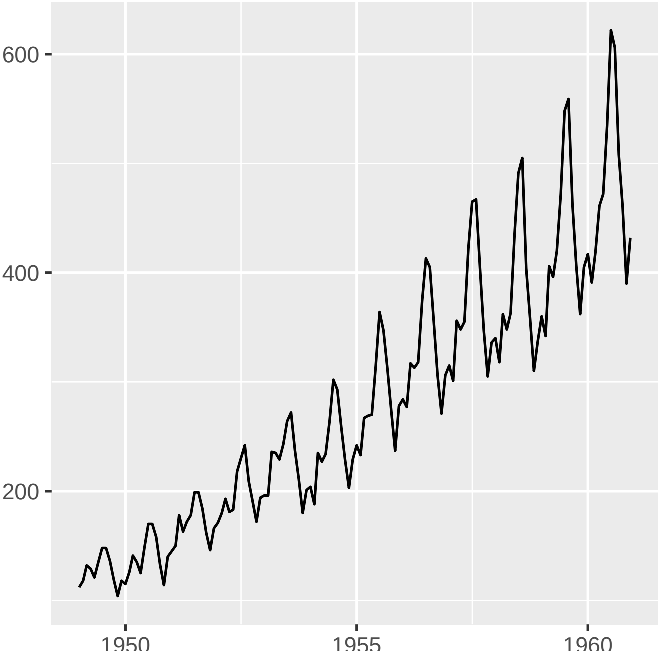
Example: AirPassengers Dataset

AirPassengers dataset reports monthly airline passengers (in thousands) from 1949 to 1960.

Example: AirPassengers Dataset

```r
# Plot AirPassengers data
autoplot(AirPassengers)
```

Example: AirPassengers Dataset



ui Raço

Sampling frequency: exact

- Some time series data is exactly evenly spaced.
- For example, hourly temperature measurements for every hour in a day.

Endri Raço

Sampling frequency: approximate

- Some time series data is only approximately evenly spaced.
- For example, temperature measurements recorded every time you check your email.

Sampling frequency: missing values

- Some time series data is evenly spaced, but with missing values.
- For example, hourly temperature measurements while you are awake.

Basic assumptions

The analysis of time series data proceeds with some simplifying assumptions:

- The first assumption is that consecutive observations are equally spaced.
- Secondly, a discrete-time observation index is applied.

Basic assumptions

- In practice, this may only hold approximately, and sometimes data may be missing.
- For example, daily log returns on a stock may only be available for weekdays, and data may not be available for certain holidays.
- Monthly CPI (Consumer Price Index) values are equally spaced by month, but not by days.

Endri Raço

Sampling frequency: R functions

- You can apply the **start()** function to the hourly temperature measurements series to confirm that it begins on day one at hour one.
- Similarly, applying the **end()** function confirms that the series last observation is on day one at hour 24.

Sampling frequency: R functions

```r
# View the start and end dates of AirPassengers
start(AirPassengers)
```

```
[1] 1949    1
```

```r
end(AirPassengers)
```

```
[1] 1960   12
```

Endri Raço

Sampling frequency: R functions

- The **frequency()** function reports that 24 observations are made each day, and the **deltat()** function notes that observations are made every 0.0417 days, that is, the time increment between observations is 1 over 24.

Sampling frequency: R functions

```r
# Use time(), deltat(), frequency(), and cycle() with
# AirPassengers
deltat(AirPassengers)
```

```
[1] 0.08333
```

Endri Raço

Sampling frequency: R functions

```r
# Use time(), deltat(), frequency(), and cycle() with
# AirPassengers
frequency(AirPassengers)
```

```
[1] 12
```

Sampling frequency: R functions

```r
# Use time(), deltat(), frequency(), and cycle() with
# AirPassengers
time(AirPassengers)
```

```
      Jan  Feb  Mar  Apr  May  Jun  Jul  Aug  Sep  Oct  Nov  Dec
1949 1949 1949 1949 1949 1949 1949 1950 1950 1950 1950 1950 1950
1950 1950 1950 1950 1950 1950 1950 1951 1951 1951 1951 1951 1951
1951 1951 1951 1951 1951 1951 1951 1952 1952 1952 1952 1952 1952
1952 1952 1952 1952 1952 1952 1952 1953 1953 1953 1953 1953 1953
1953 1953 1953 1953 1953 1953 1953 1954 1954 1954 1954 1954 1954
1954 1954 1954 1954 1954 1954 1954 1955 1955 1955 1955 1955 1955
1955 1955 1955 1955 1955 1955 1955 1956 1956 1956 1956 1956 1956
1956 1956 1956 1956 1956 1956 1956 1957 1957 1957 1957 1957 1957
1957 1957 1957 1957 1957 1957 1957 1958 1958 1958 1958 1958 1958
1958 1958 1958 1958 1958 1958 1958 1959 1959 1959 1959 1959 1959
1959 1959 1959 1959 1959 1959 1959 1960 1960 1960 1960 1960 1960
1960 1960 1960 1960 1960 1960 1960 1961 1961 1961 1961 1961 1961
```

Sampling frequency: R functions

```r
# Use time(), deltat(), frequency(), and cycle() with
# AirPassengers
cycle(AirPassengers)
```

```
     Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
1949   1   2   3   4   5   6   7   8   9  10  11  12
1950   1   2   3   4   5   6   7   8   9  10  11  12
1951   1   2   3   4   5   6   7   8   9  10  11  12
1952   1   2   3   4   5   6   7   8   9  10  11  12
1953   1   2   3   4   5   6   7   8   9  10  11  12
1954   1   2   3   4   5   6   7   8   9  10  11  12
1955   1   2   3   4   5   6   7   8   9  10  11  12
1956   1   2   3   4   5   6   7   8   9  10  11  12
1957   1   2   3   4   5   6   7   8   9  10  11  12
1958   1   2   3   4   5   6   7   8   9  10  11  12
1959   1   2   3   4   5   6   7   8   9  10  11  12
1960   1   2   3   4   5   6   7   8   9  10  11  12
```

Endri Raço

Missing values

- Sometimes there are missing values in time series data, denoted **NA** in R, and it is useful to know their locations.
- It is also important to know how missing values are handled by various R functions.

Endri Raço

Missing values

Sometimes we may want to ignore any missingness, but other times we may wish to impute or estimate the missing values.

Missing values

- Let's again consider the monthly **AirPassengers** dataset, but now the data for the year **1956** are missing.

Endri Raço

Missing values

- Let's use **plot()** to display a simple plot of **AirPassengers**.
  - Note the missing data for 1956.

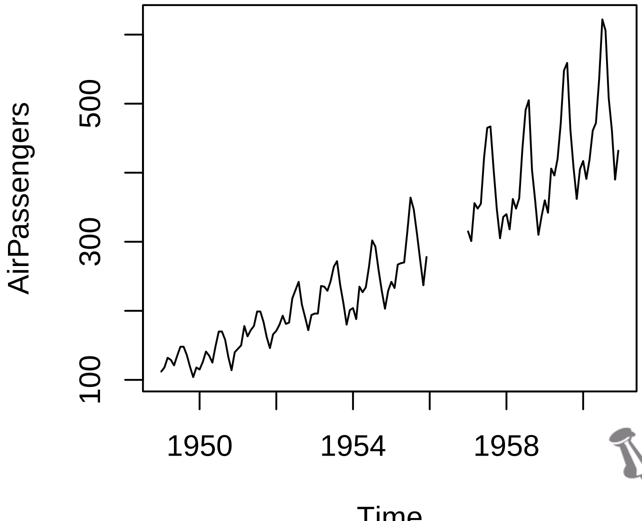
Missing values

```r
# Get the time index for AirPassengers
time_index <- time(AirPassengers)

# Set the data for the year 1956 to NA
AirPassengers[which(floor(time_index) == 1956)] <- NA

# Plot the AirPassengers data
plot(AirPassengers)
```

Missing values



Endri Raço

Missing values

- We'll explore the implications of this missing data and impute some new data to solve the problem.

Missing values

- The **mean()** function calculates the sample mean, but it fails in the presence of any **NA** values.
- We use **mean(____, na.rm = TRUE)** to calculate the mean with all missing values removed.

Endri Raço

Missing values

```
# Compute the mean of AirPassengers without using na.rm
mean(AirPassengers)
```

```
[1] NA
```

```
# Compute the mean of AirPassengers using na.rm
mean(AirPassengers, na.rm = TRUE)
```

```
[1] 275.9
```

Missing values

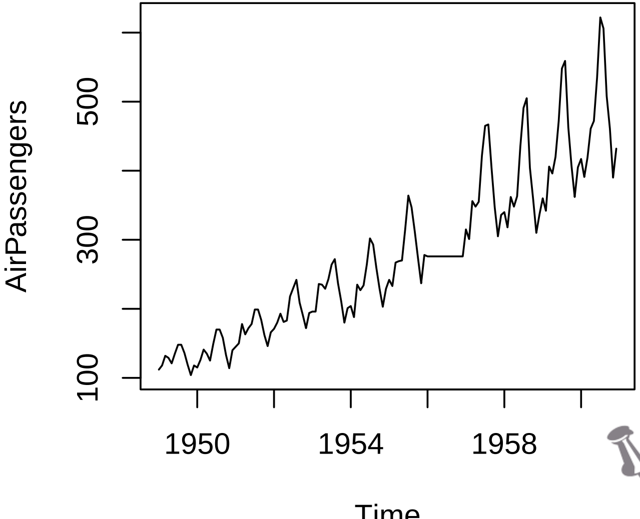
- It is common to replace missing values with the mean of the observed values.

Endri Raço

Missing values

```r
# Impute mean values to NA in AirPassengers
AirPassengers[85:96] <- mean(AirPassengers, na.rm = TRUE)
# Generate another plot of AirPassengers
plot(AirPassengers)
```

Missing values



Missing values

Does this simple data imputation scheme appear adequate when applied the the AirPassengers dataset?

Based on our plot, it seems that simple data imputation using the mean is not a great method to approximate what's really going on in the AirPassengers data.

Missing values

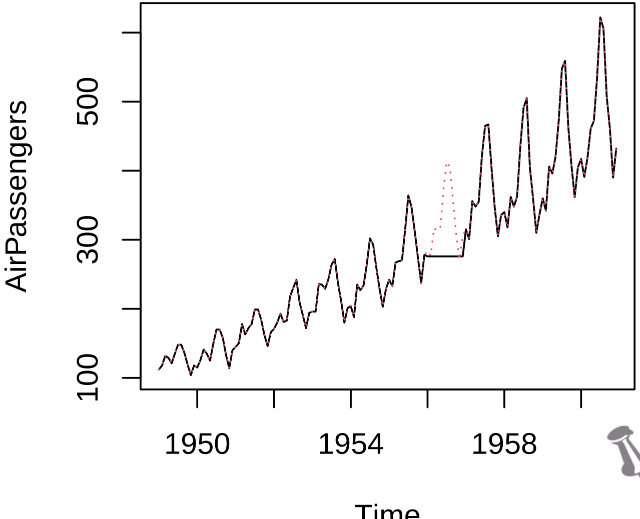
```r
# Impute mean values to NA in AirPassengers
AirPassengers[85:96] <- mean(AirPassengers, na.rm = TRUE)

# Generate another plot of AirPassengers
plot(AirPassengers)

# Add the complete AirPassengers data to your plot
rm(AirPassengers)
points(AirPassengers, type = "l", col = 2, lty = 3)
```

Endri Raço

Missing values



Endri Raço

Time Series objects

Endri Raço

Basic time series objects

A time series is more than a vector of numbers, it also includes *the time indices* for each observation.

Building ts() objects

- Start with a vector of numbers
- Apply the **ts()** function to create a time series object.
- Such objects are of the `ts` class.
- They represent data that is at least approximately evenly spaced over time.

Endri Raço

Building ts() objects

- Consider the following *data_vector* which has just eight observations.

Example

```r
data_vector <- c(10, 6, 11, 8, 10, 3, 6, 9)
data_vector
```

```
[1] 10  6 11  8 10  3  6  9
```

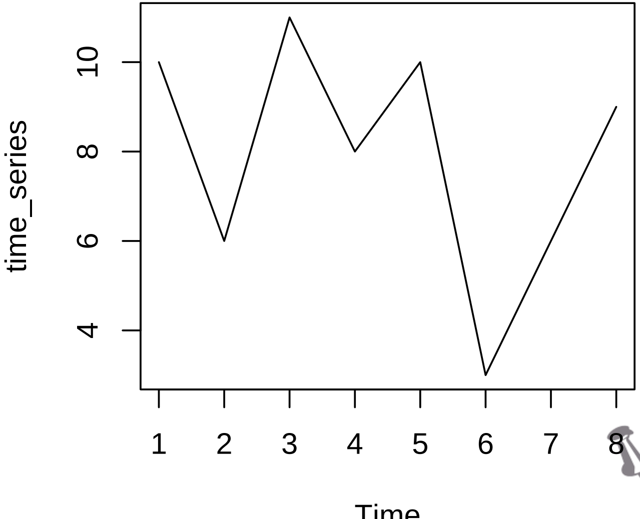
Building ts() objects

To make this vector a time series object you apply the **ts()** function
When you plot the result using the **plot()** function the time index and label is automatically added to the horizontal axis.

Example

```
time_series <- ts(data_vector)
plot(time_series)
```

Example



Building ts() objects

By default, R uses a simple observation index starting from 1 as the time index.

Building ts() objects

If you want the time series to start in the year 2001 with 1 observation per year you should apply the ts() function with the additional arguments *start* $= 2001$ and *frequency* $= 1$ as shown.

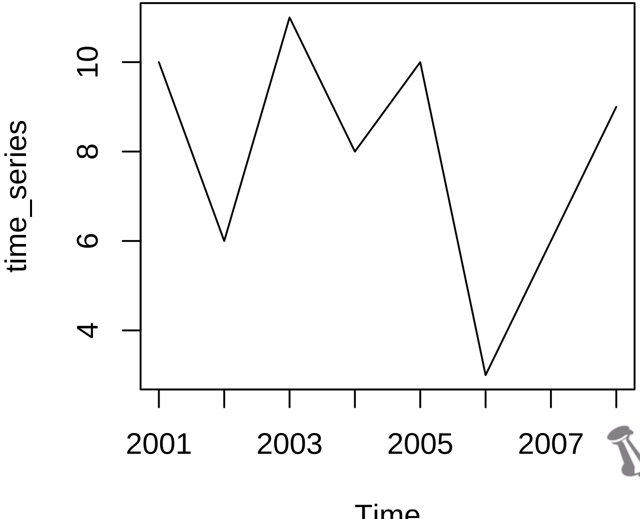
Example

```r
time_series <- ts(data_vector, start = 2001, frequency = 1)
plot(time_series)
```

Example

Now when you plot the result you can see an updated time axis, running from 2001 through 2008.

Example



Using is.ts()

- You can use the function **is.ts()** to check whether a given object is a time series.

Example

```r
is.ts(data_vector)
```

```
[1] FALSE
```

```r
is.ts(time_series)
```

```
[1] TRUE
```

Using is.ts()

As you can see, it reports **FALSE** for the **data_vector** and **true** for the **time_series** that were just created.

Why ts() objects?

Why do you want to create and work with time series objects of the **ts** class?

- Improved plotting
- Access to time index information
- Model estimation and forecasting

Components of a time series

Components of a time series

A time series can be considered composed of 4 main parts:

1. **trend**
2. **cycle**,
3. **seasonality**
4. **irregular** or remainder/residual part.

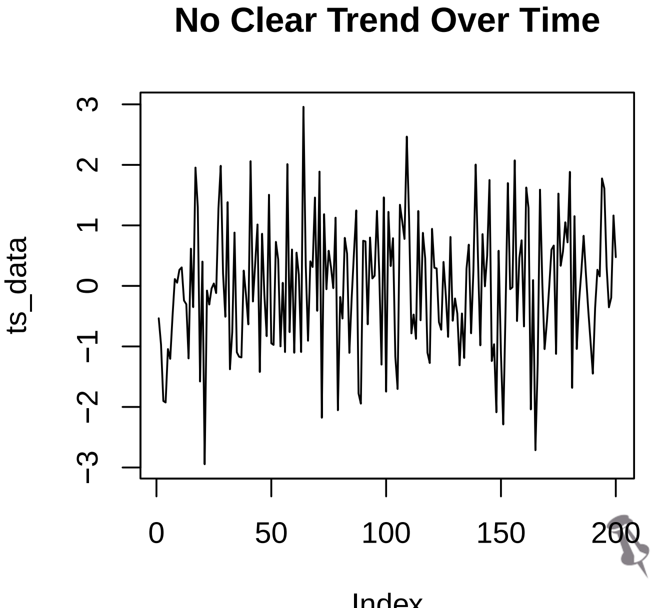
Time Series Trends: No Clear Trend

- Some time series data do not exhibit any clear or consistent trend over time.
- They fluctuate around a constant mean, with no obvious increase or decrease.
- **Example**:
  - Random fluctuations over time without a specific direction.

Time Series Trends: No Clear Trend

```r
# Simulating no clear trend
ts_data <- rnorm(200)
plot(ts_data, type = "l", main = "No Clear Trend Over Time"
```

Time Series Trends: No Clear Trend



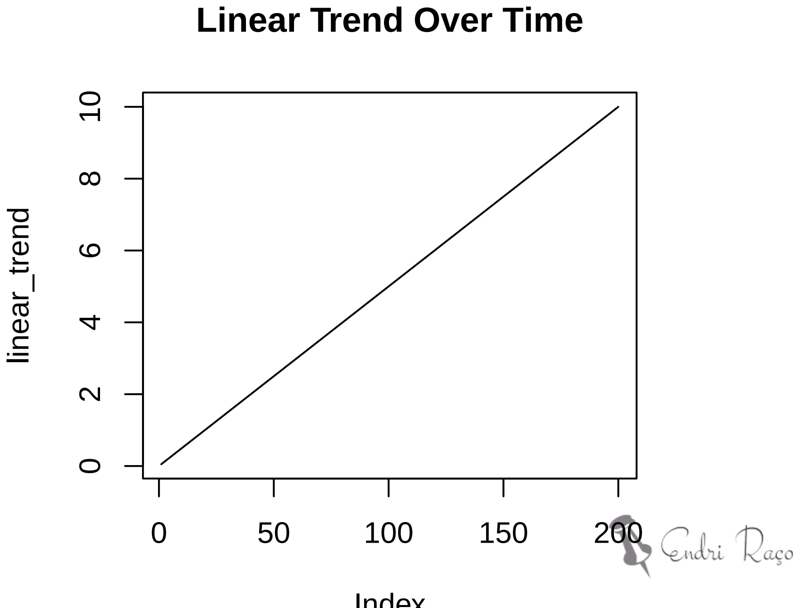
Time Series Trends: Linear

- A **linear trend** is a consistent increase or decrease over time.
- This type of trend is represented by a straight line when plotted.
- **Example**:
  - Data increases or decreases at a constant rate over time.

Time Series Trends: Linear

```r
# Simulating linear trend
time <- 1:200
linear_trend <- 0.05 * time
plot(linear_trend, type = "l", main = "Linear Trend Over Time")
```

Time Series Trends: Linear



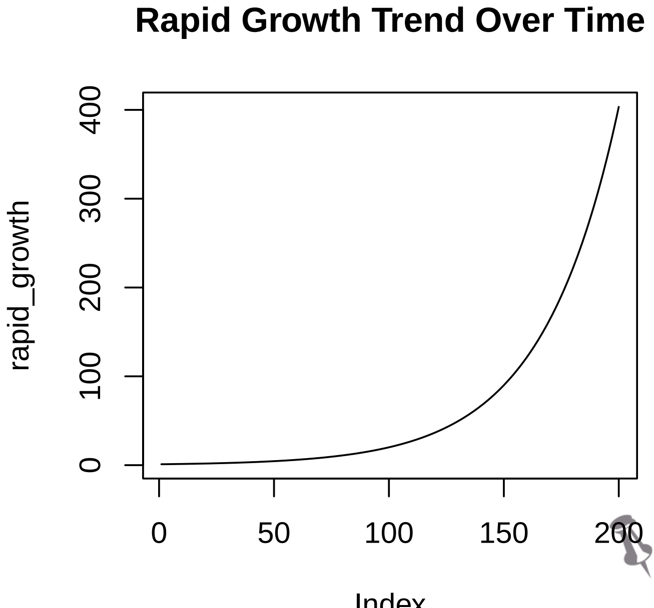
Time Series Trends: Rapid Growth

- **Rapid growth trends** refer to exponential-like increases over time.
- These trends grow quickly after a certain point.
- **Example**:
  - A rapid increase in a dataset, such as population growth.

Time Series Trends: Rapid Growth

```r
# Simulating rapid growth
time <- 1:200
rapid_growth <- exp(0.03 * time)
plot(rapid_growth, type = "l", main = "Rapid Growth Trend Over Time")
```

Time Series Trends: Rapid Growth



Time Series Trends: Periodic

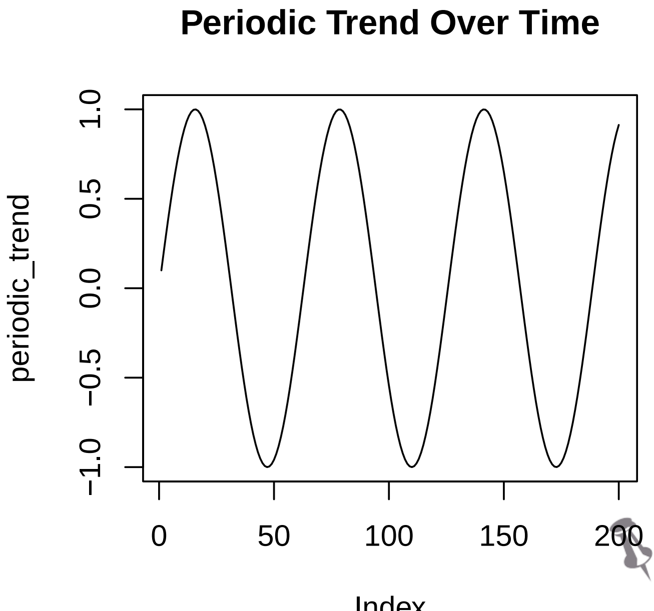
- **Periodic trends** are repeated patterns that recur over time.
- Typically represented as sinusoidal or oscillating movements.
- **Example**:
  - Seasonal changes in temperature or sales cycles.

Endri Raço

Time Series Trends: Periodic

```r
# Simulating periodic trend
time <- 1:200
periodic_trend <- sin(0.1 * time)
plot(periodic_trend, type = "l", main = "Periodic Trend Over Time")
```

Time Series Trends: Periodic



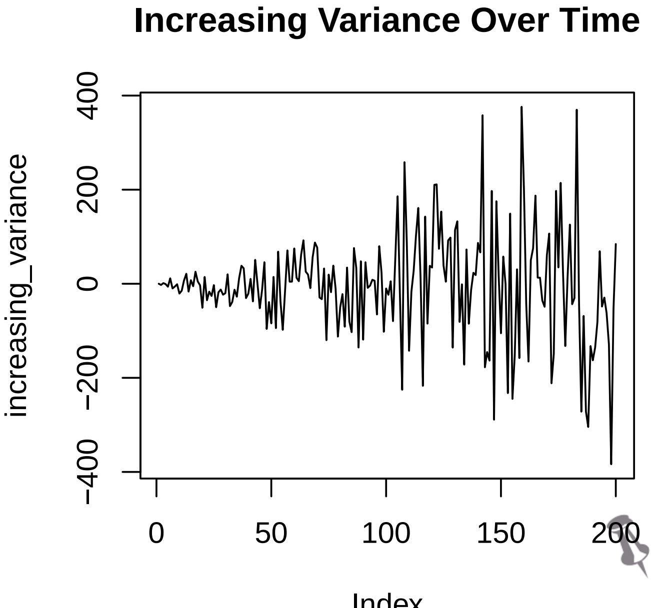
Time Series Trends: Variance

- Some time series exhibit increasing or decreasing **variance** over time.
- This means the spread or volatility of the data changes as time progresses.
- **Example**:
  - Financial time series where volatility increases over time.

Time Series Trends: Variance

```r
# Simulating increasing variance
time <- 1:200
increasing_variance <- time * rnorm(200)
plot(increasing_variance, type = "l", main = "Increasing Variance Over Time")
```

Time Series Trends: Variance



Endri Raço

Why Use Sample Transformations?

Simplify Data:

- Some time series exhibit complex patterns like exponential growth, periodic trends, or increasing variance.
- Transformations can simplify these patterns, making analysis easier.

Why Use Sample Transformations?

Make Data Stationary:

- Many time series models require data to be stationary (constant mean, variance, and covariance).
- Transformations such as differencing or logging help achieve stationarity.

Why Use Sample Transformations?

Remove Trends:

- By removing linear or seasonal trends, transformations allow us to focus on the true underlying behavior of the data, such as fluctuations or noise.
- **Stabilize Variance**: Transformations like logarithms can help stabilize variance in cases where the spread of data changes over time, improving the accuracy of models.

Endri Raço

Why Use Sample Transformations?

Improve Interpretability:

- Transformed data often reveals clearer insights and patterns, aiding better decision-making and predictions.

Endri Raço

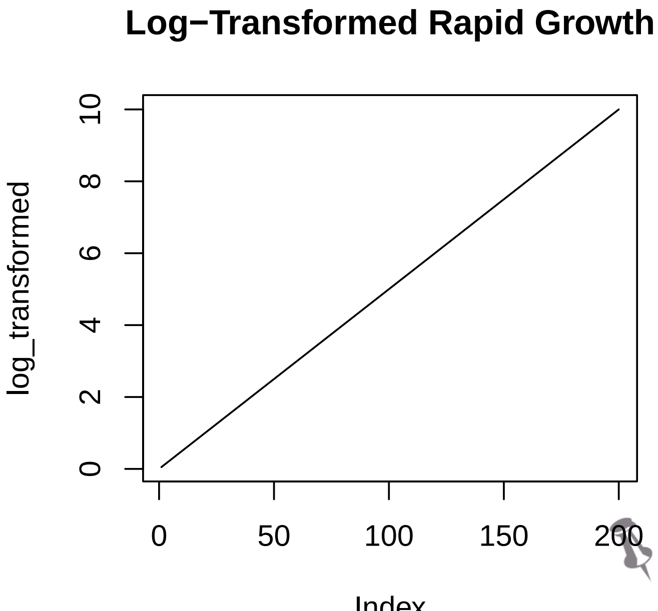
Sample Transformations: Log()

- The **log()** transformation can linearize data with rapid growth.
- By taking the logarithm of values, rapid exponential growth becomes more linear.
- **Use case**:
  - Suitable for data with exponential growth patterns, such as financial data.

Sample Transformations: Log()

```r
# Log transformation to linearize rapid growth
rapid_growth <- exp(0.05 * time)
log_transformed <- log(rapid_growth)
plot(log_transformed, type = "l", main = "Log-Transformed Rapid Growth")
```

Sample Transformations: Log()



Sample Transformations: Diff()

- The **diff()** function computes the difference between successive values.
- The first difference transformation of a time series z_t consists of the differences (changes) between successive observations over time, that is:

$$\Delta z_t = z_t - z_{t-1}$$

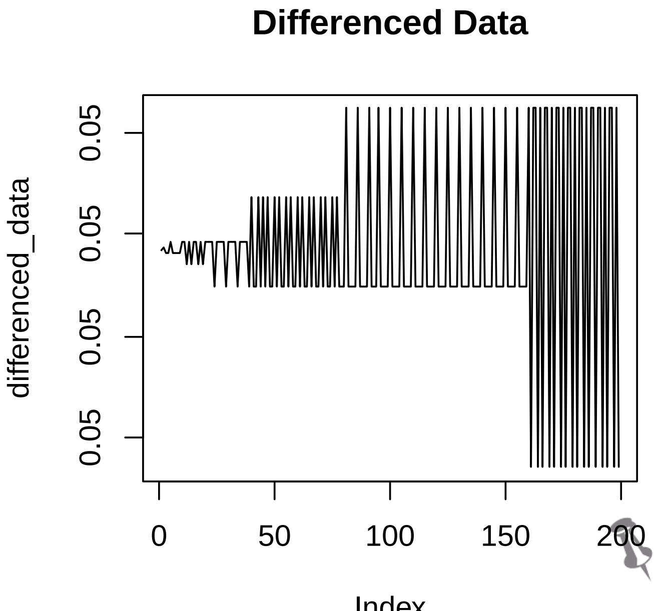
where Δz_t represents the first difference at time t.

- This can help remove linear trends, making data easier to analyze.
- **Use case**:
  - Often used in financial data to remove trends and focus on changes over time.

Sample Transformations: Diff()

```r
# Differencing to remove linear trend
linear_trend <- 0.05 * time
differenced_data <- diff(linear_trend)
plot(differenced_data, type = "l", main = "Differenced Data")
```

Sample Transformations: Diff()



Sample Transformations: Seasonal Differencing

- **Seasonal differencing** removes periodic trends by differencing the series at seasonal intervals.
  - Helps focus on variations within the same cycle.
- **Use case**:
  - Used when data shows clear seasonal or cyclical patterns, such as monthly sales.

Sample Transformations: Seasonal Differencing

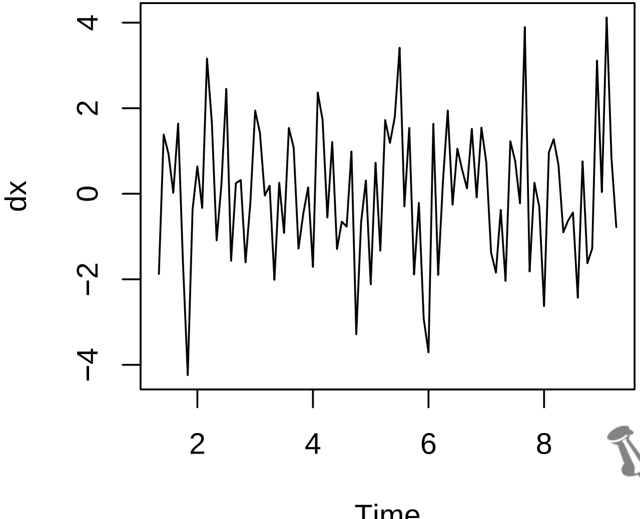
- The function `diff(..., lag = s)` will calculate the lag s difference or length s seasonal change series.
- For monthly or quarterly data, an appropriate value of s would be 12 or 4, respectively.
- The `diff()` function has `lag = 1` as its default for first differencing.
- Similar to before, a seasonally differenced series will have s fewer observations than the original series.

Sample Transformations: Seasonal Differencing

```
# Create a sample time series 'x' to use in seasonal
# differencing
x <- ts(rnorm(100), frequency = 12)  # Simulate a monthly time s
# Seasonal differencing with lag = 4
dx <- diff(x, lag = 4)
ts.plot(dx)
```

Endri Raço

Sample Transformations: Seasonal Differencing



White Noise (WN) Model

White noise is the simplest example of a stationary process. A weak white noise (WN) process has the following characteristics:

- A fixed, constant mean.
- A fixed, constant variance.
- No correlation over time.

Simulating the White Noise Model

- The white noise (WN) model is a basic time series model and serves as the foundation for more elaborate models.
- We will focus on the simplest form of WN, where the data are independent and identically distributed (iid).

Endri Raço

Simulating the White Noise Model

- The `arima.sim()` function can be used to simulate data from various time series models.
- `ARIMA` is an abbreviation for *autoregressive integrated moving average*, a class of models that needs to be studied in detail.

Simulating the White Noise Model

An `ARIMA(p, d, q)` model has three components:

- *p*: The autoregressive order.
- *d*: The order of integration (or differencing).
- *q*: The moving average order.

ARIMA Model Components

An `ARIMA(p, d, q)` model has three key components:

- p: The autoregressive order
- d: The order of integration (or differencing)
- q: The moving average order

Explanation

The ARIMA model is used to predict time series data, taking into account the past values, trends, and errors. Let's break down each of its components:

Endri Raço

Autoregressive (AR) Order

- **Definition**: The number of past observations used to predict the current value.
- **Explanation**: If $p = 2$, the model uses the last two values of the series to predict the next one.

It's similar to saying, "Today's value depends on the last two values."

$$X_t = \alpha + \beta_1 X_{t-1} + \beta_2 X_{t-2} + \epsilon_t$$

Differencing Order

- **Definition**: The number of times the data has been differenced to make it stationary.
- **Explanation**: Differencing removes trends to stabilize the data.
  - If $d = 1$, the model looks at changes between consecutive values instead of raw data.

For example, subtracting X_{t-1} from X_t:

$$\Delta X_t = X_t - X_{t-1}$$

Endri Raço

Moving Average (MA) Order

- **Definition**: The number of past errors used to predict the current value.
- **Explanation**: The model uses past forecast errors to adjust the current prediction.
  - If $q = 1$, it uses the last error to improve today's forecast.

$$X_t = \mu + \epsilon_t + \theta_1 \epsilon_{t-1}$$

Endri Raço

Example of ARIMA(1,1,1)

This would indicate:

- $p = 1$: Use the last value for prediction.
- $d = 1$: Use the first difference of the series.
- $q = 1$: Use the last forecast error.

$$\Delta X_t = \alpha + \beta X_{t-1} + \epsilon_t + \theta \epsilon_{t-1}$$

Endri Raço

Simulating the White Noise Model

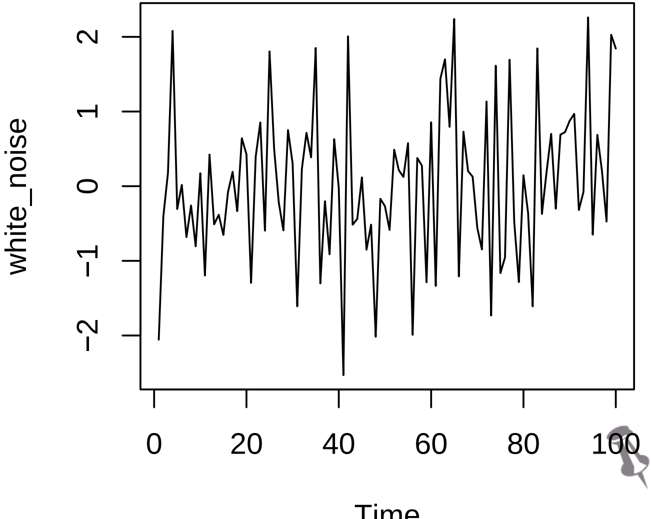
- In this exercise, we focus on the simplest model, `ARIMA(0, 0, 0)`, where all parameters are set to zero.
- This corresponds to a white noise model.

White Noise (WN) Model

```
# Simulate WN model
white_noise <- arima.sim(model = list(order = c(0, 0, 0)), n = 1
ts.plot(white_noise)
```

Endri Raço

White Noise (WN) Model



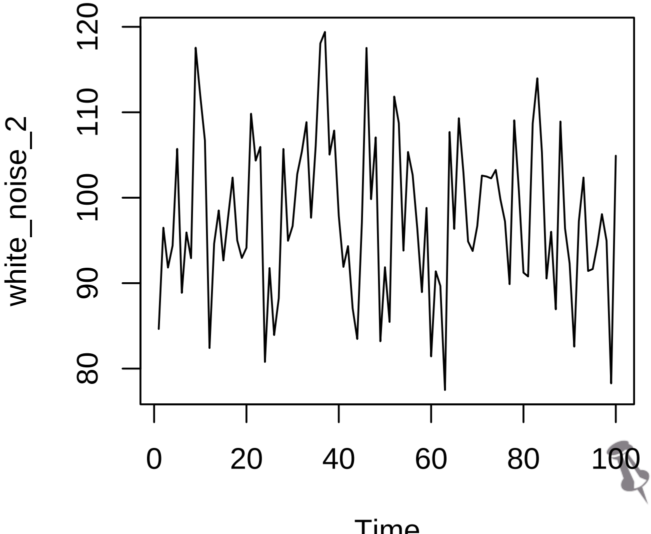
Simulating White Noise with Mean and Variance

- White noise can be generated with a specific mean and variance.

```
# Simulate WN with mean = 100, sd = 10
white_noise_2 <- arima.sim(model = list(order = c(0, 0, 0)),
    n = 100, mean = 100, sd = 10)
ts.plot(white_noise_2)
```

Endri Raço

Simulating White Noise with Mean and Variance



Model future behavior

- By fitting a model to the data (e.g., using ARIMA), we can use the estimated parameters to make predictions about future observations.
- For example, estimating parameters for a white noise model allows us to predict future values based on the randomness of the process.

Endri Raço

Estimating the White Noise Model

- **Estimate WN** using *arima*() with $order = c(0, 0, 0)$.
- The model estimates the mean (intercept) and variance ($sigma^2$).

Estimating the White Noise Model

```r
# Simulate a white noise series for y
y <- arima.sim(model = list(order = c(0, 0, 0)), n = 100)
# Fit WN model
arima(y, order = c(0, 0, 0))
# Fit WN model
arima(y, order = c(0, 0, 0))
mean(y)
var(y)
```

Estimating the White Noise Model

```
Call:
arima(x = y, order = c(0, 0, 0))

Coefficients:
      intercept
          0.104
s.e.      0.105

sigma^2 estimated as 1.09:  log likelihood = -146.3,  aic = 296.
```


```
Call:
arima(x = y, order = c(0, 0, 0))

Coefficients:
      intercept
          0.104
s.e.      0.105

sigma^2 estimated as 1.09:  log likelihood = -146.3,  aic = 296.
```

Stationary Processes

- **Stationary models** are parsimonious (balance simplicity and explanatory power).
- **Distributional stability** over time: mean, variance, and covariance are constant.
- **Observed time series**:
  - Fluctuate randomly.
- Behave similarly from one time period to the next.

Non - Stationary Processes

- A non-stationary process is a time series where statistical properties like the mean, variance, or covariance change over time.
- One of the simplest examples of a non-stationary process is a random walk.

The Random Walk (RW) Model

- A *random walk* (RW) is a simple example of a non-stationary process.
- A random walk has the following properties:
  - No specified mean or variance.
  - Strong dependence over time.
  - Its changes or increments are white noise (WN).

The Random Walk Recursion

Recursion means that each step in the process depends on the result of the previous step. In the case of a random walk, it means the current value Y_t is calculated by adding a random change ϵ_t to the previous value Y_{t-1}.

$$Y_t = Y_{t-1} + \epsilon_t$$

The Random Walk Recursion

The random walk recursion is given by:

$$\text{Today} = \text{Yesterday} + \text{Noise}$$

or more formally:

$$Y_t = Y_{t-1} + \epsilon_t$$

where ϵ_t is mean zero white noise (WN).

Endri Raço

The Random Walk Recursion

Simulation requires an initial point Y_0 and only one parameter, the WN variance σ_ϵ^2. Since:

$$Y_t - Y_{t-1} = \epsilon_t$$

the first difference diff(Y) is white noise.

Random Walk with Drift

- A random walk with drift is a random walk where, in addition to the random change, there is a constant trend (called “drift”).
- This means that over time, the values tend to move in a particular direction (upward or downward) instead of just randomly fluctuating around a fixed value.

Random Walk with Drift

A random walk with drift can be described as:

$$\text{Today} = \text{Constant} + \text{Yesterday} + \text{Noise}$$

or more formally:

$$Y_t = c + Y_{t-1} + \epsilon_t$$

where c is the drift (constant), and ϵ_t is white noise with variance σ_ϵ^2. In this case:

$$Y_t - Y_{t-1} = ?$$

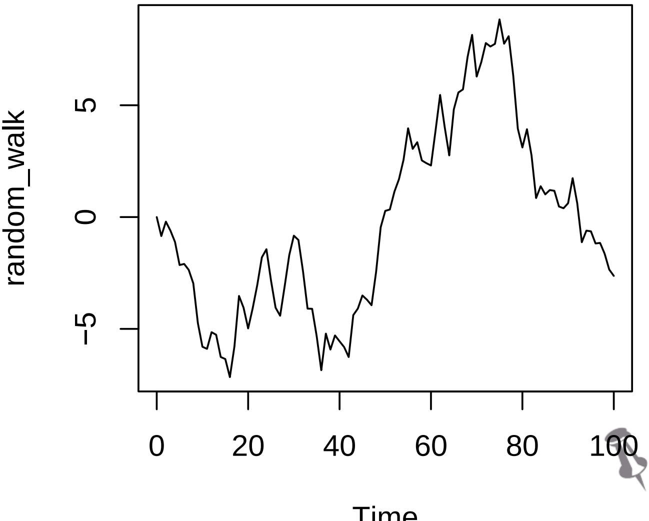
which means that diff(Y) is white noise with mean c.

The Random Walk (RW) Model

```
# Simulate RW model
random_walk <- arima.sim(model = list(order = c(0, 1, 0)), n = 1
ts.plot(random_walk)
```

Endri Raço

The Random Walk (RW) Model



Simulating the Random Walk Model with Drift

- A random walk (RW) need not wander around zero; it can have an upward or downward trajectory, i.e., a drift or time trend.
- This can be achieved by including an intercept in the RW model, which corresponds to the slope of the RW time trend.

Endri Raço

Simulating the Random Walk Model with Drift

For an alternative formulation, you can take the cumulative sum of a constant mean white noise (WN) series, where the mean corresponds to the slope of the RW time trend.

Simulating the Random Walk Model with Drift

- To simulate data from the RW model with a drift, you can use the `arima.sim()` function with the `model = list(order = c(0, 1, 0))` argument.
- Additionally, you should include the argument `mean = ...` to specify the drift variable, or the intercept.

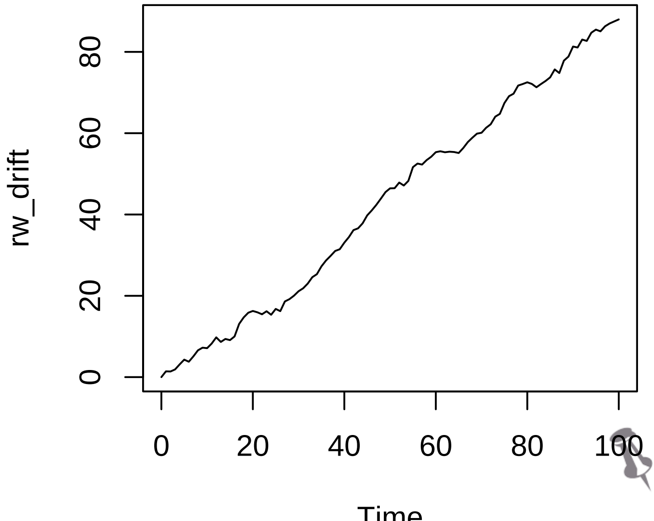
Simulating the Random Walk Model with Drift

- A **random walk with drift** includes a constant trend.
- **Formula**: $Y_t = c + Y_{t-1} + \epsilon_t$

Simulating the Random Walk Model with Drift

```
# Simulate RW model with drift
rw_drift <- arima.sim(model = list(order = c(0, 1, 0)), n = 100,
    mean = 1)
ts.plot(rw_drift)
```

Simulating the Random Walk Model with Drift



Estimating the Random Walk Model

- To estimate the random walk, first difference the data and fit a white noise model.

```r
# Fit WN model to first differenced RW
rw_diff <- diff(rw_drift)
model_wn <- arima(rw_diff, order = c(0, 0, 0))
```

Estimating the Random Walk Model

- **Key point**: The estimated drift (intercept) is shown in the model output.

Endri Raço

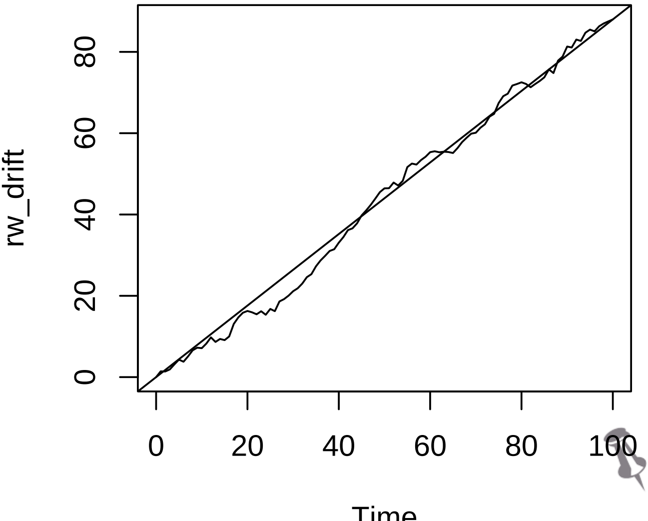
Visualizing the Random Walk with a Time Trend

Goal: Visualize the estimated time trend (drift) in the random walk data.

```
# Plot RW data and add time trend
ts.plot(rw_drift)
abline(0, model_wn$coef[1])  # Add time trend line
```

Endri Raço

Visualizing the Random Walk with a Time Trend



Endri Raço

Weak Stationarity

- A process $Y_1, Y_2, \ldots$ is **weakly stationary** if:
- Mean μ of Y_t is constant for all t.
- Variance σ^2 of Y_t is constant for all t.
- Covariance of Y_t and Y_s depends only on $|t - s| = h$.

Weak Stationarity

```
# Example covariance
cov(Y_3, Y_7) = cov(Y_7, Y_11)
```

Stationarity: When?

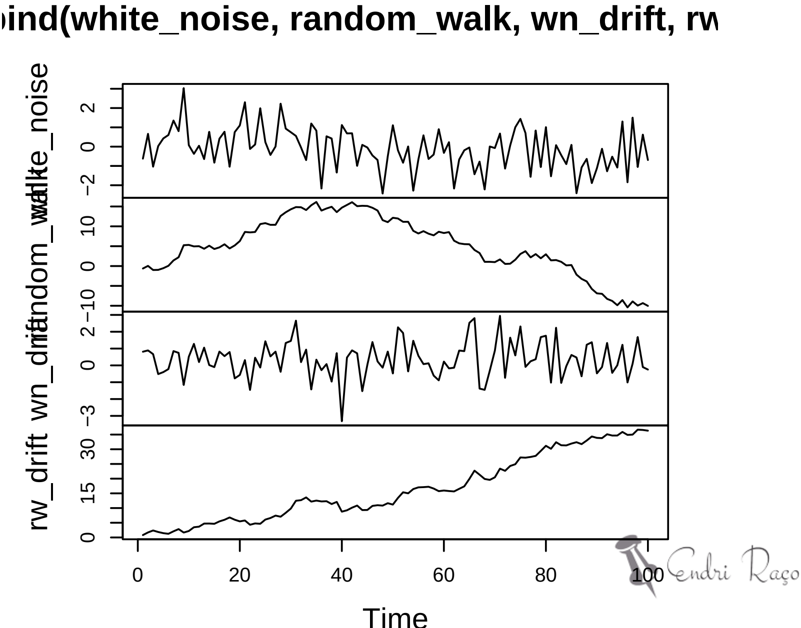
- Financial time series often not stationary, but changes are approximately stationary.
- Stationary series oscillates around a fixed level (mean-reversion).
- White noise (WN) is stationary, random walk (RW) is not.

Endri Raço

Stationarity Simulation

```
# White noise simulation
white_noise <- arima.sim(model = list(order = c(0, 0, 0)), n = 1
# Random walk from white noise
random_walk <- cumsum(white_noise)
# White noise with drift
wn_drift <- arima.sim(model = list(order = c(0, 0, 0)), n = 100,
    mean = 0.4)
# Random walk with drift
rw_drift <- cumsum(wn_drift)
# Plot
plot.ts(cbind(white_noise, random_walk, wn_drift, rw_drift))
```

Stationarity Simulation



The Goal of Investing

Goal of Investing:

- To make a profit.

Profit depends on:

- Amount invested

Changes in prices

Asset returns measure:

- Changes in price as a fraction of the initial price.

Calculated over a given time horizon (e.g., one business day).

What are Financial Returns?

- Financial returns are a way to measure profit or loss.
- They show how much the price of an asset has changed.
- Common time frames: daily, weekly, monthly.

Endri Raço

EU Stocks Dataset

- *eu_stocks* reports index values (prices).
- Indices are not investable but many assets (like mutual funds) track them.

Asset Prices vs. Returns

- Prices show the value of an asset at a given time.
- Returns show how much the price has changed relative to its starting value.

Endri Raço

Log Returns

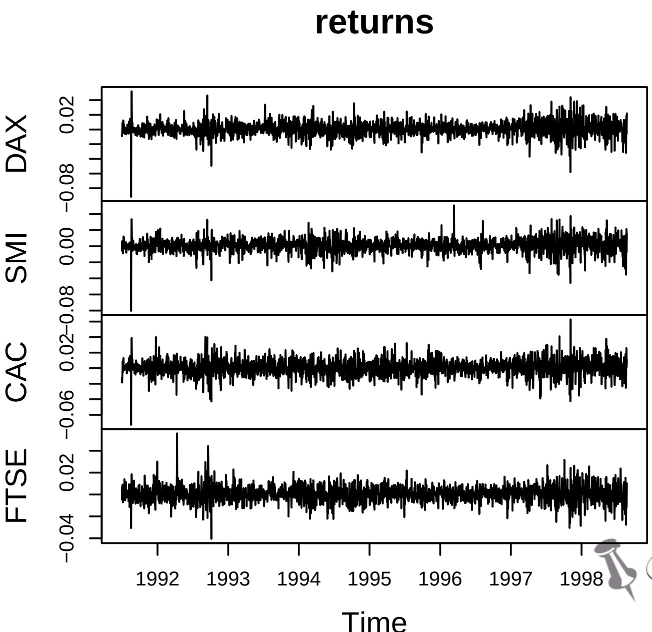
- Log returns are also called continuously compounded returns.
- They simplify multi-period calculations.

$$Logreturns = logofgrossreturns$$

Prices and Returns in eu_stocks

```r
eu_stocks <- EuStockMarkets
# Convert prices to returns
returns <- eu_stocks[-1, ]/eu_stocks[-nrow(eu_stocks), ] - 1
returns <- ts(returns, start = c(1991, 130), frequency = 260)
# Plot returns
plot(returns)
```

Prices and Returns in eu_stocks

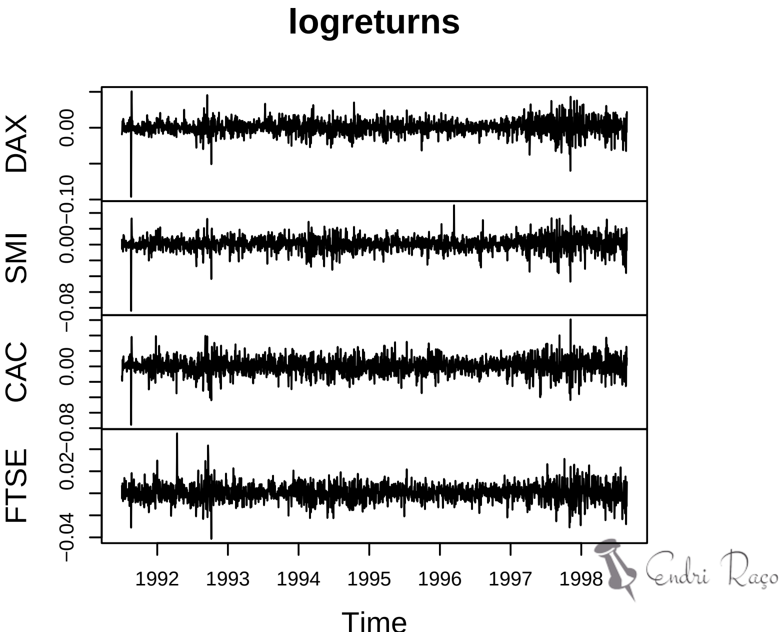


Prices and Returns in eu_stocks

```r
# Log returns
logreturns <- diff(log(eu_stocks))
# Plot log returns
plot(logreturns)
```

Endri Raço

Prices and Returns in eu_stocks



Financial Time Series Characteristics

- Small daily returns, close to zero average.
- Large variances and occasional extreme returns (outliers).
- Heavy-tailed, non-normal distribution.

Endri Raço

Statistical Summary of Returns

```r
eu_percentreturns <- returns * 100
# Means, Variance, SD
colMeans(eu_percentreturns)
apply(eu_percentreturns, MARGIN = 2, FUN = var)
apply(eu_percentreturns, MARGIN = 2, FUN = sd)
```

Statistical Summary of Returns

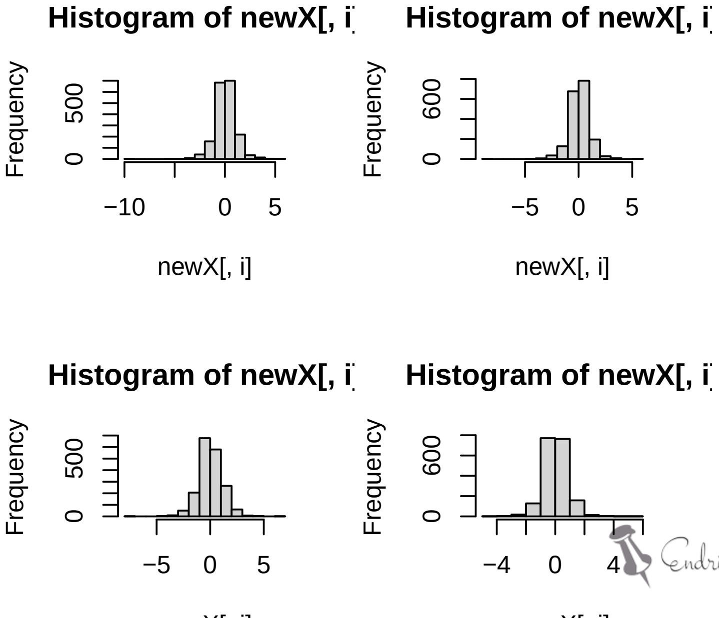
```
    DAX     SMI     CAC    FTSE
0.07052 0.08609 0.04979 0.04637
   DAX    SMI    CAC   FTSE
1.0570 0.8524 1.2159 0.6345
   DAX    SMI    CAC   FTSE
1.0281 0.9232 1.1027 0.7965
```

Statistical Summary of Returns

```
# Histograms and Quantile plots
par(mfrow = c(2, 2))
apply(eu_percentreturns, 2, hist)
apply(eu_percentreturns, 2, qqnorm)
qqline(eu_percentreturns)
```

Endri Raço

Statistical Summary of Returns



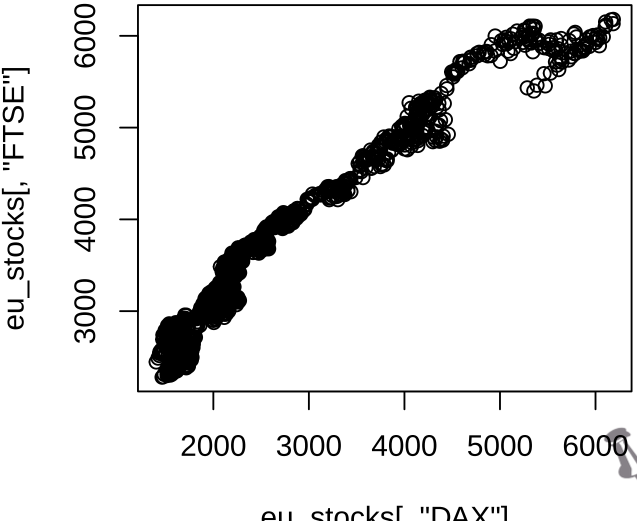
Scatterplots: Asset Prices vs Log Returns

- Scatterplots: relationship between pairs of time series.
- Log returns often show elliptical patterns in pairs.

Scatterplots: Asset Prices vs Log Returns

```r
# Scatterplot of DAX and FTSE
plot(eu_stocks[, "DAX"], eu_stocks[, "FTSE"])
```

Scatterplots: Asset Prices vs Log Returns

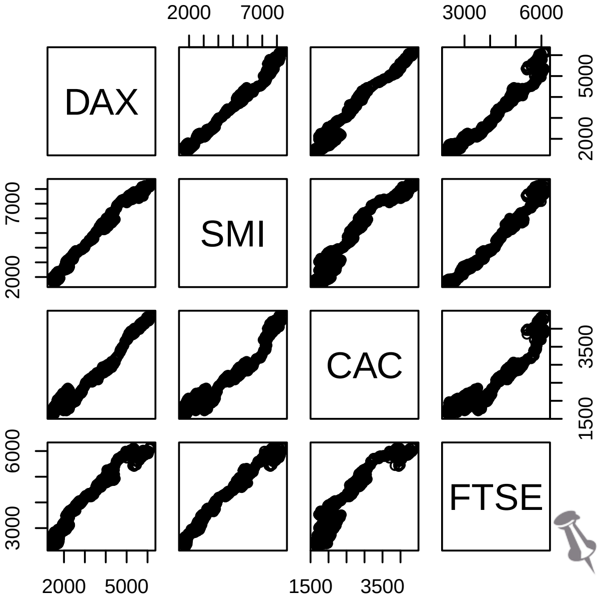


Scatterplots: Asset Prices vs Log Returns

```r
# Scatterplot matrix
pairs(eu_stocks)
pairs(logreturns)
```

Endri Raço

Scatterplots: Asset Prices vs Log Returns



Endri Raço

Covariance and Correlation

- Covariance: relationship between variables, scale-dependent.
- Correlation: standardized covariance, ranges [-1, 1].

Covariance and Correlation

```r
DAX_logreturns <- diff(log(eu_stocks[, "DAX"]))
FTSE_logreturns <- diff(log(eu_stocks[, "FTSE"]))
# Covariance and correlation
cov(DAX_logreturns, FTSE_logreturns)
cor(DAX_logreturns, FTSE_logreturns)
cor(logreturns)
```

Covariance and Correlation

```
[1] 0.00005242

[1] 0.6395

        DAX    SMI    CAC   FTSE
DAX  1.0000 0.7031 0.7344 0.6395
SMI  0.7031 1.0000 0.6160 0.5848
CAC  0.7344 0.6160 1.0000 0.6486
FTSE 0.6395 0.5848 0.6486 1.0000
```

Autocorrelation: Lag Dependence

Autocorrelation: correlation between values at different lags.

```r
# Lag-1 autocorrelation manually
x_t0 <- x[-1]
x_t1 <- x[-length(x)]
cor(x_t0, x_t1)
# ACF calculation
acf(x, lag.max = 10, plot = FALSE)
```

Autocorrelation: Lag Dependence

```
[1] -0.03914

Autocorrelations of series 'x', by lag

0.0000 0.0833 0.1667 0.2500 0.3333 0.4167 0.5000 0.5833 0.6667 0
 1.000 -0.039  0.059  0.123 -0.066 -0.164 -0.094  0.029 -0.181 -
```

Endri Raço

Autoregressive (AR) Model

$$ARrecursion : Today = Slope * Yesterday + Noise$$

Persistence is higher with large slope values.

Endri Raço

Autoregressive (AR) Model

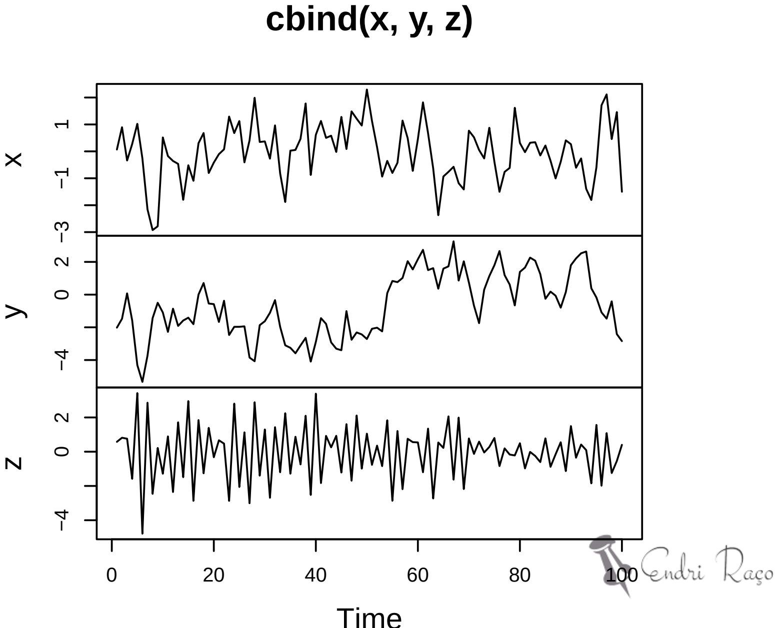
```r
# Simulate AR models
x <- arima.sim(model = list(ar = 0.5), n = 100)
y <- arima.sim(model = list(ar = 0.9), n = 100)
z <- arima.sim(model = list(ar = -0.75), n = 100)
```

Autoregressive (AR) Model

```r
# Plot
plot.ts(cbind(x, y, z))
acf(x)
acf(y)
acf(z)
```

Endri Raço

Autoregressive (AR) Model



MA vs AR Models

- MA: short-run dependence, AR: persistent dependence.
- AIC/BIC help compare model fit (lower values are better).

Moving Average (MA) Model

MA process: current value depends on current & previous noise.

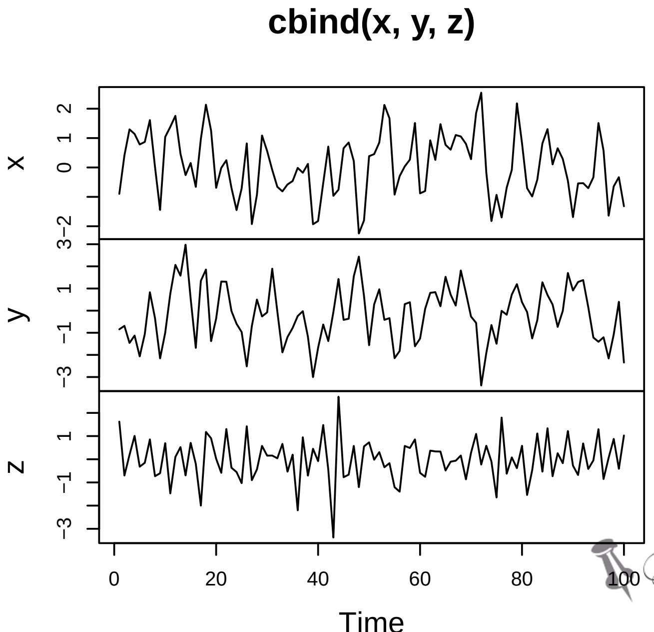
```
# Simulate MA models
x <- arima.sim(model = list(ma = 0.5), n = 100)
y <- arima.sim(model = list(ma = 0.9), n = 100)
z <- arima.sim(model = list(ma = -0.5), n = 100)
```

Moving Average (MA) Model

```r
# Plot and ACF
plot.ts(cbind(x, y, z))
acf(x)
acf(y)
acf(z)
```

Endri Raço

Moving Average (MA) Model

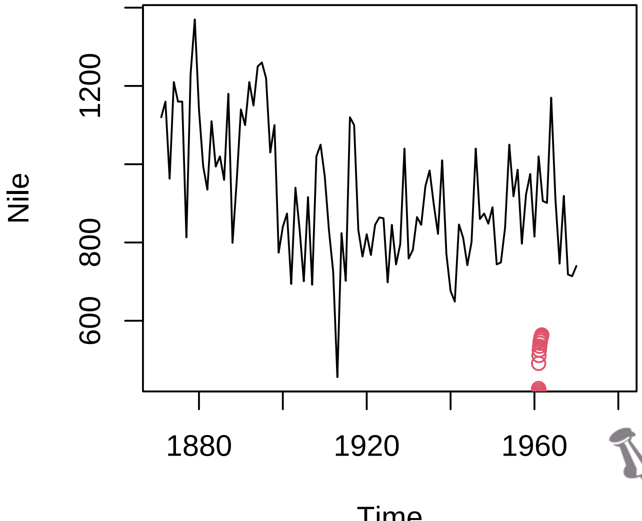


Forecasting with AR/MA Models

Predict future observations based on AR/MA models.

```
AR <- arima(AirPassengers, order = c(1, 0, 0))
# Forecasts from AR model
AR_forecast <- predict(AR, n.ahead = 10)
# Plot with prediction intervals
ts.plot(Nile, xlim = c(1871, 1980))
points(AR_forecast$pred, col = 2)
points(AR_forecast$pred - 2 * AR_forecast$se, col = 2, lty = 2)
points(AR_forecast$pred + 2 * AR_forecast$se, col = 2, lty = 2)
```

Forecasting with AR/MA Models



Conclusion

- Key models: White noise, random walk, AR, MA.
- Use AIC/BIC to evaluate model fit.
- Time series tools: arima.sim(), acf(), predict().

Endri Raço

Thank You!