Financial Risk Forecasting
Seminar Week 1: Introduction to R and Financial Data Analysis

Jon Danielsson
London School of Economics

Version 4.0

1 R, data and sample statistics

1.1 Why R and financial data analysis matter for financial risk

R is the industry standard for quantitative finance and risk management. Financial institutions use R for portfolio optimisation, risk measurement, regulatory reporting and algorithmic trading. The ability to download, manipulate and analyse financial data programmatically is essential for modern risk management practices.

Learning R and financial data fundamentals provides the foundation for all subsequent risk modelling techniques we will develop throughout this course.

1.2 Introduction

The software we use in this course is R, which is a powerful open-source language for statistics, data analysis and simulation. It is widely used in finance, economics and academia. We recommend using RStudio as a user-friendly interface that makes it easier to write, test and debug code, with built-in tools for plots, packages and version control.

Please download R and RStudio and familiarise yourself with the basic setup. This RStudio for the Total Beginner might be of help and you can find many others on YouTube.

For financial data, we will use the EOD Historical Data API (or EODHD) through the `eodhdR2` package. This provides a simple interface to access stock prices, dividends and splits directly from R. More information can be found at https://eodhd.com/financial-apis/r-library-v-2-for-financial-data-by-eodhd-2024.

For more information about R and especially how it is used in forecasting risk, refer to the R risk forecasting notebook. You can find references to resources for learning R in the notebook.

1.3 The plan for this week

1. Familiarisation with R and RStudio
2. Learn basic R commands
3. Set up EOD Historical Data access
4. Download, manipulate and save financial data
5. Create and customise plots
6. Calculate basic financial statistics

1.4 Basic R commands

We can enter some basic R commands into the `console` and you see the output immediately printed on the console.

```r
1+1
```

```
[1] 2
```

```r
2^5
```

```
[1] 32
```

```r
a=2
b=100
exp(a)+b/10
```

```
[1] 17.38906
```

```r
result=exp(a)+b
cat("the answer is:",log(a),"or",result,"\n")
```

```
the answer is: 0.6931472 or 107.3891
```

Usually it is best to create a file for all the code you want to execute, perhaps `seminar1.R`. You can then execute the commands in this file either one by one or all at once. We use scripts and notebooks — not the console alone.

Also, it is best to keep the data files you create below in the same directory (folder).

R needs to be able to find your data and for that you may need to set the `working directory`. You can do that with the RStudio menu item `Session/Set Working Directory` or the R command `setwd()`.

1.5 Setting up EOD Historical Data Access

1.5.1 Installing Required Packages

The `eodhdR2` package relies on several supporting packages that need to be installed first:

You only need to run `install.packages()` once.

```r
# Install supporting packages
install.packages(c("httr", "jsonlite", "lubridate", "data.table", "readr"), repos = "https

# Install the EODHD R package
install.packages('eodhdR2', repos = "https://cloud.r-project.org/")
```

Every time you run this code, you need to load the package (also known as a library).

```r
# Load the package
library(eodhdR2)
```

1.5.2 API Token Setup

An API token is like a digital key that identifies you to the EODHD service. The token is a unique string of characters that tells EODHD who you are and what data you're allowed to access.

If you are in my courses, you will get access to EODHD. For others, you can register at eodhistoricaldata.com for a free or paid account.

In the code below, if you have a personal API key: Replace "YOUR_API_KEY_HERE" in the code below with your actual key. Else, if you are using the demo key, you can only access basic price data for major stocks. Functions like `get_splits()` will not work.

1.6 Downloading Financial Data

1.6.1 Setup

Set up your API token

```r
token = "YOUR_API_KEY_HERE"
set_token(token)
```

1.6.2 What works with demo vs personal tokens

Demo tokens can access: basic price data (`get_prices()`) and ividend data (`get_dividends()`), but not stock splits data (`get_splits()`) and some advanced market data.

Personal token can access all functions without restrictions. However, there are tiers in access.

With EODHD, we can download data directly in R using ticker symbols. The `get_prices()` function takes two main parameters: the stock ticker symbol and the exchange code. For US stocks, we use "US" as the exchange. Start with a single stock, Apple, whose ticker is `AAPL`:

```r
# Download Apple stock data
prices = get_prices("AAPL", "US")
```

```
-- retrieving price data for ticker AAPL|US ------------------------------------
```

```
! Quota status: 71|100000, refreshing in 15.3 hours

i cache file AAPL_US_eodhd_prices.rds saved

v got 11255 rows of prices

i got daily data from 1980-12-12 to 2025-08-08
```

1.6.3 Understanding the Data Structure

The EOD price data includes:

- `date`: Trading date
- `open`: Opening price for the day
- `high`: Highest price during the day
- `low`: Lowest price during the day
- `close`: Closing price for the day
- `adjusted_close`: Close price adjusted for splits and dividends
- `volume`: Number of shares traded
- `ticker`: Symbol
- `exchange`: Where traded
- `ret_adj_close`: Simple returns for the adjusted closeing price

1.7 Exploring the Data

1.7.1 Description

```r
# Explore the data structure
dim(prices)
```

```
[1] 11255    10
```

```r
head(prices)
```

```
        date    open    high     low   close adjusted_close    volume ticker
1 1980-12-12 28.7392 28.8736 28.7392 28.7392         0.0986 469033600   AAPL
2 1980-12-15 27.3728 27.3728 27.2608 27.2608         0.0935 175884800   AAPL
3 1980-12-16 25.3792 25.3792 25.2448 25.2448         0.0866 105728000   AAPL
4 1980-12-17 25.8720 26.0064 25.8720 25.8720         0.0887  86441600   AAPL
5 1980-12-18 26.6336 26.7456 26.6336 26.6336         0.0914  73449600   AAPL
6 1980-12-19 28.2464 28.3808 28.2464 28.2464         0.0969  48630400   AAPL
  exchange ret_adj_close
1       US            NA
2       US   -0.05172414
3       US   -0.07379679
4       US    0.02424942
5       US    0.03043968
6       US    0.06017505
```

```r
names(prices)
```

```
 [1] "date"           "open"           "high"           "low"           
 [5] "close"          "adjusted_close" "volume"         "ticker"        
 [9] "exchange"       "ret_adj_close" 
```

```
# Summary statistics
summary(prices$adjusted_close)
```

```
    Min.  1st Qu.   Median     Mean  3rd Qu.     Max. 
  0.0377   0.2449   0.4627  25.9844  20.3285 258.3967 
```

1.7.2 Make returns

```
y=diff(log(prices$adjusted_close))
```

We could add y to `prices` but that will not work.

```
# prices$y=y  # This would cause an error
```

It gives an error `replacement has 11236 rows, data has 11237`. This occurs because the first observation in `y` will be

```
# This shows the formula for the first return (not executed):
# y[1] = log(prices$adjusted_close[2]) - log(prices$adjusted_close[1])
```

So the first observation in `y` corresponds to the second day.

One way to deal with this is by

```
# First, make sure y is properly calculated as the full vector of returns
y=diff(log(prices$adjusted_close))
length(c(NA,y))
```

```
[1] 11255
```

```
prices$y=c(NA,y)
```

And, perhaps, to then remove the first row altogether:

```
prices=prices[2:nrow(prices),]
```

```
summary(prices$y)
```

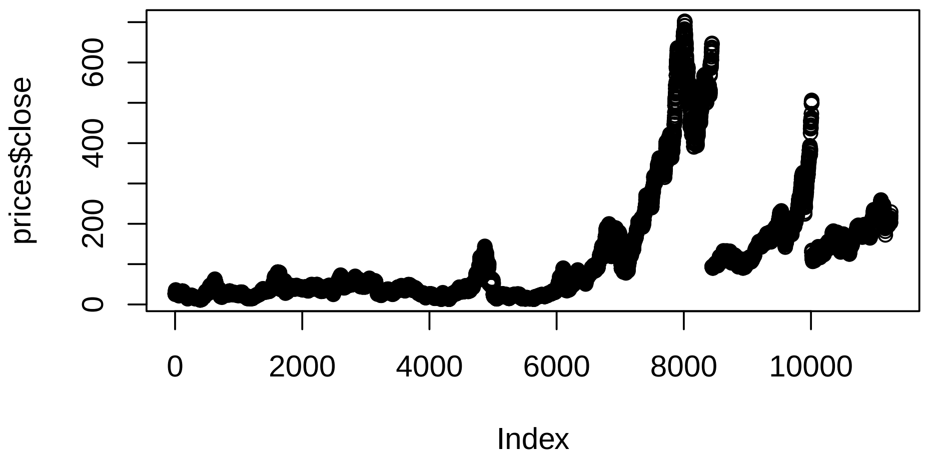
```
      Min.    1st Qu.     Median       Mean    3rd Qu.       Max. 
-0.7314330 -0.0126420  0.0000000  0.0006888  0.0142034  0.2868394 
```

1.8 Visualizing Stock Data

R can easily build plots to visualise data. We use the `plot` command for this. See Plottin in the R Notebook.

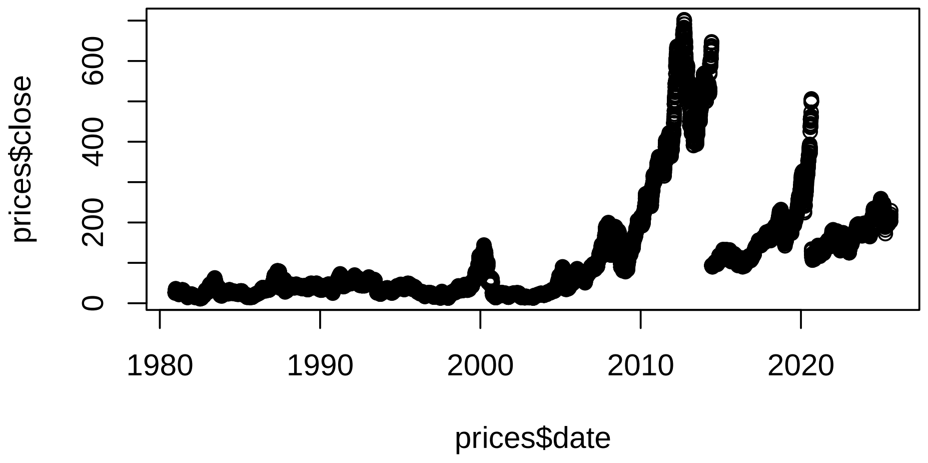
We use base R plots instead of ggplot2 because the code is shorter and simpler for the types of plots we make in financial analysis. Base plots are built into R and do not require additional packages, while producing comparable visual quality for our purposes.

```
plot(prices$close)
```



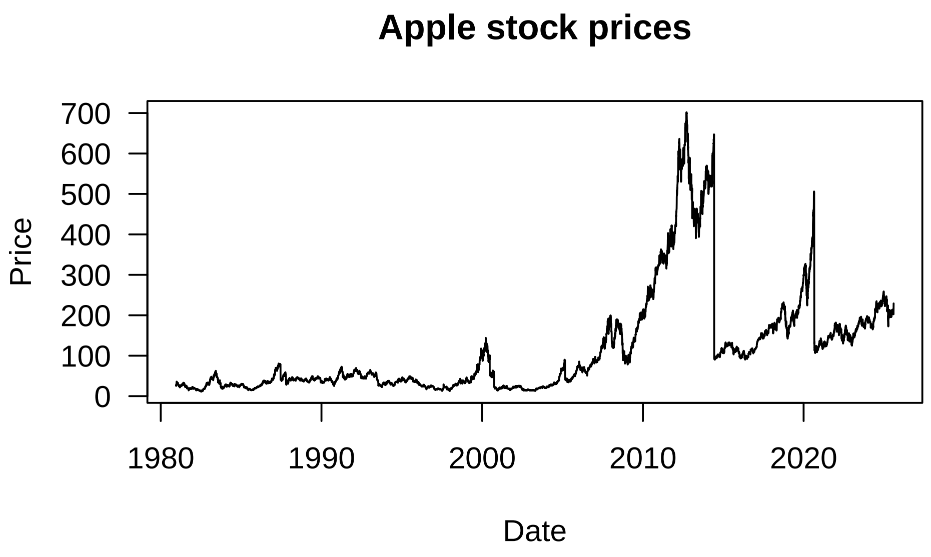
This plot needs improvement. To begin with, we want dates on the x-axis.

```r
plot(prices$date,prices$close)
```



We can do better

```r
plot(prices$date,prices$close,
     type = 'l',
     las = 1,
     main = "Apple stock prices",
     xlab = "Date",
     ylab = "Price")
```



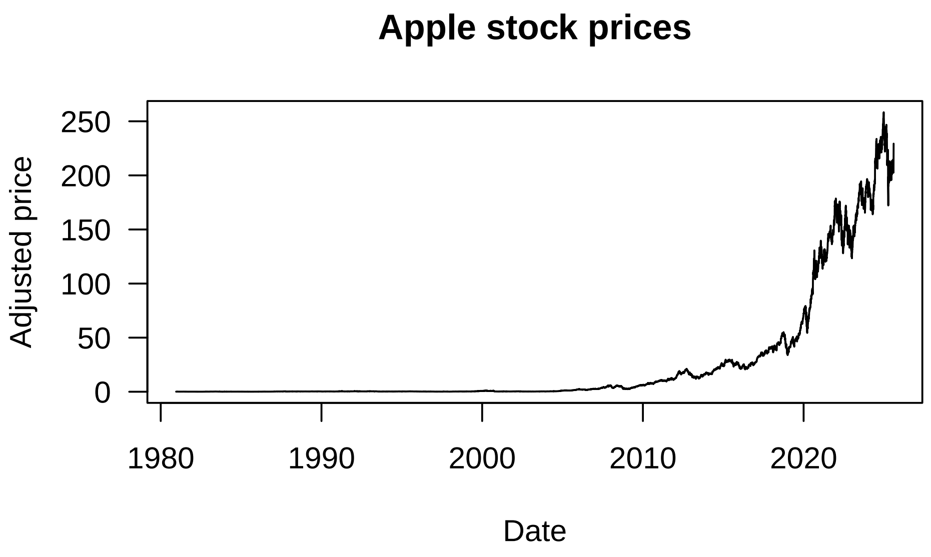
But this does not look very realistic. Why?

The regular closing price shows sudden jumps and drops that are not related to company performance — these are caused by stock splits and dividend payments. For example, Apple had a 4-for-1 stock split on August 31, 2020, where the price suddenly dropped from around $500 to $125, creating an artificial drop that has nothing to do with the company's value.

Adjusted prices correct for these corporate actions, giving us the true economic picture of how the stock has performed. This is crucial for financial analysis because we want to measure actual returns, not accounting artifacts.

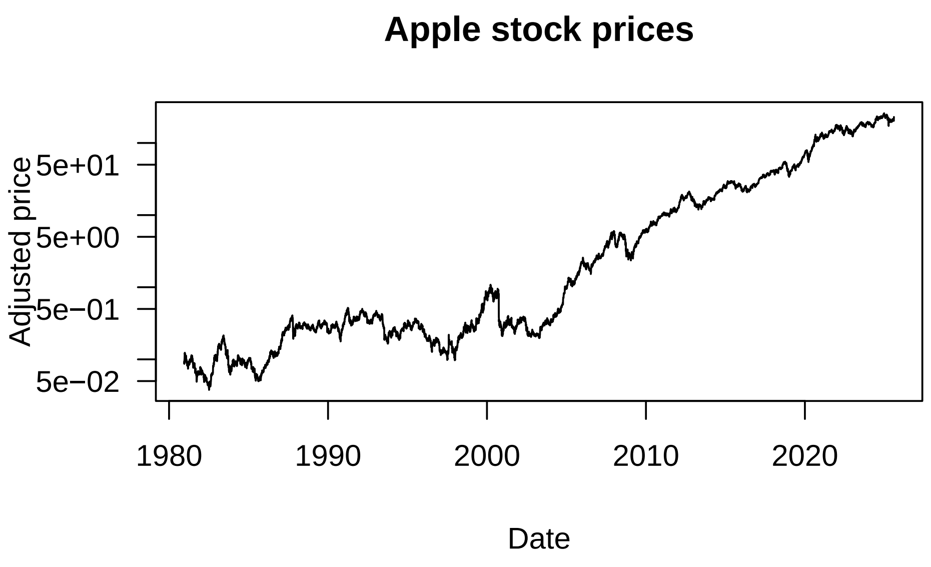
However, not all large changes in the closing price are due to splits. Apple's price halved on 2000-06-21 due to a bad earnings announcement.

```r
plot(prices$date,prices$adjusted_close,
     type = 'l',
     las = 1,
     main = "Apple stock prices",
     xlab = "Date",
     ylab = "Adjusted price"
)
```



Or perhaps with log y-axis:

```
plot(prices$date,prices$adjusted_close,
     type = 'l',
     las = 1,
     main = "Apple stock prices",
     xlab = "Date",
     ylab = "Adjusted price",
     log='y'
)
```



1.9 Accessing Additional Data

EOD also provides dividend and split information:

```r
# Get dividend data for Apple
dividends = get_dividends("AAPL", "US")
```

```
-- retrieving dividends for ticker AAPL|US ------------------------------------

! Quota status: 72|100000, refreshing in 15.3 hours

i cache file AAPL_US_eodhd_dividends.rds saved

v got 88 rows of dividend data
```

```r
head(dividends)
```

```
        date ticker exchange declarationDate recordDate paymentDate period
1 1987-05-11   AAPL       US            <NA>       <NA>        <NA>   <NA>
2 1987-08-10   AAPL       US            <NA>       <NA>        <NA>   <NA>
3 1987-11-17   AAPL       US            <NA>       <NA>        <NA>   <NA>
4 1988-02-12   AAPL       US            <NA>       <NA>        <NA>   <NA>
5 1988-05-16   AAPL       US            <NA>       <NA>        <NA>   <NA>
6 1988-08-15   AAPL       US            <NA>       <NA>        <NA>   <NA>
    value unadjustedValue currency
1 0.00054         0.12096      USD
2 0.00054         0.06048      USD
3 0.00071         0.07952      USD
4 0.00071         0.07952      USD
5 0.00071         0.07952      USD
6 0.00071         0.07952      USD
```

```r
# splits = get_splits("AAPL", "US")  # Only works with a personal token
# head(splits)
```

1.10 Recap

1.10.1 In this seminar we have covered:

- Installing and setting up R and RStudio
- Installing and configuring the eodhdR2 package
- Understanding API tokens and how to use them securely
  - Understanding demo token limitations
- Downloading and exploring stock price data
- Understanding the difference between regular and adjusted prices
- Calculating log returns from price data and handling dimension mismatches
- Creating progressively better visualisations:
  - Basic plots
  - Adding dates to axes
  - Improving labels and formatting
  - Using log scale for better long-term visualisation
- Accessing additional data (dividends and splits)

1.10.2 Key skills learned:

- Managing package installation with `install.packages()` and `library()`
- Working with data frames and handling NA values
- Using `diff()` and `log()` for return calculations
- Debugging common issues (like dimension mismatches)
- Customizing plots with multiple parameters
- Understanding why adjusted prices matter for analysis

1.10.3 Some new functions used:

- `install.packages()` — installs R packages from CRAN
- `library()` — loads an installed package into the current session
- `set_token()` — configures your API authentication token
- `get_prices()` — downloads historical price data for a stock
- `get_dividends()` — retrieves dividend payment history
- `get_splits()` — fetches stock split information
- `dim()` — shows dimensions (rows and columns) of a data frame
- `head()` — displays the first few rows of data
- `nrow()` — returns the number of rows in a data frame
- `names()` — lists column names of a data frame
- `summary()` — provides statistical summary of data
- `diff()` — calculates differences between consecutive elements
- `log()` — computes natural logarithm
- `c()` — combines values into a vector or list
- `length()` — returns the number of elements in a vector
- `plot()` — creates various types of graphs and charts
- `cat()` — prints output to the console
- `exp()` — calculates exponential (e^x)
- `setwd()` — sets the working directory
- `help()` — displays documentation for functions

1.11 Optional exercises

1. **Data exploration**: Download data for Microsoft (MSFT) and create a complete analysis:
   - Calculate log returns using `diff(log(adjusted_close))`
   - Find the highest and lowest daily returns
   - Create a plot with proper labels and log scale
   - Compare the results with Apple
2. **Handling data issues**:
   - Download JPMorgan (JPM) data and calculate returns
   - What happens if you do not use `c(NA, y)` when adding returns to the dataframe?
   - Practice removing the first row after adding returns
   - Verify your results using `head()` and `dim()`
3. **Visualization practice**:
   - Create plots for any three stocks showing both regular and adjusted closing prices
   - Explain why the regular closing prices might show sudden jumps
   - Use different colors for each stock (hint: `col = "red"`, etc.)

- Add a legend to identify each line

4. **Working with dividends**:
   - Use `get_dividends()` for Apple, Microsoft and Exxon (XOM)
   - Which company pays dividends most frequently?
   - Create a plot showing dividend amounts over time
   - Calculate the total dividends paid by each company
5. **Summary statistics**:
   - For any five stocks of your choice:
   - Calculate mean and standard deviation of returns
   - Use `summary()` on both prices and returns
   - Create a table comparing the volatility (standard deviation) of each stock
   - Which stock would be considered most risky based on volatility?