FM442: Quantitative Methods for Finance and Risk Analysis

Jon Danielsson
Michaelmas Term
2020

Seminar 1

In this class we will:

1. Familiarize ourselves with R and RStudio
2. Learn some basic commands
3. Download and import financial data
4. Create a simple plot

R and RStudio

What is R?

R is a language and environment for statistical computing and graphics. It is a very powerful tool that will allow us to analyze financial data and implement models to assess and quantify risk. We will use it throughout the course for performing analyses and creating plots. No prior programming experience is required. We will go through some basics of the languages that should be enough to let you start working with financial datasets.

Downloading R

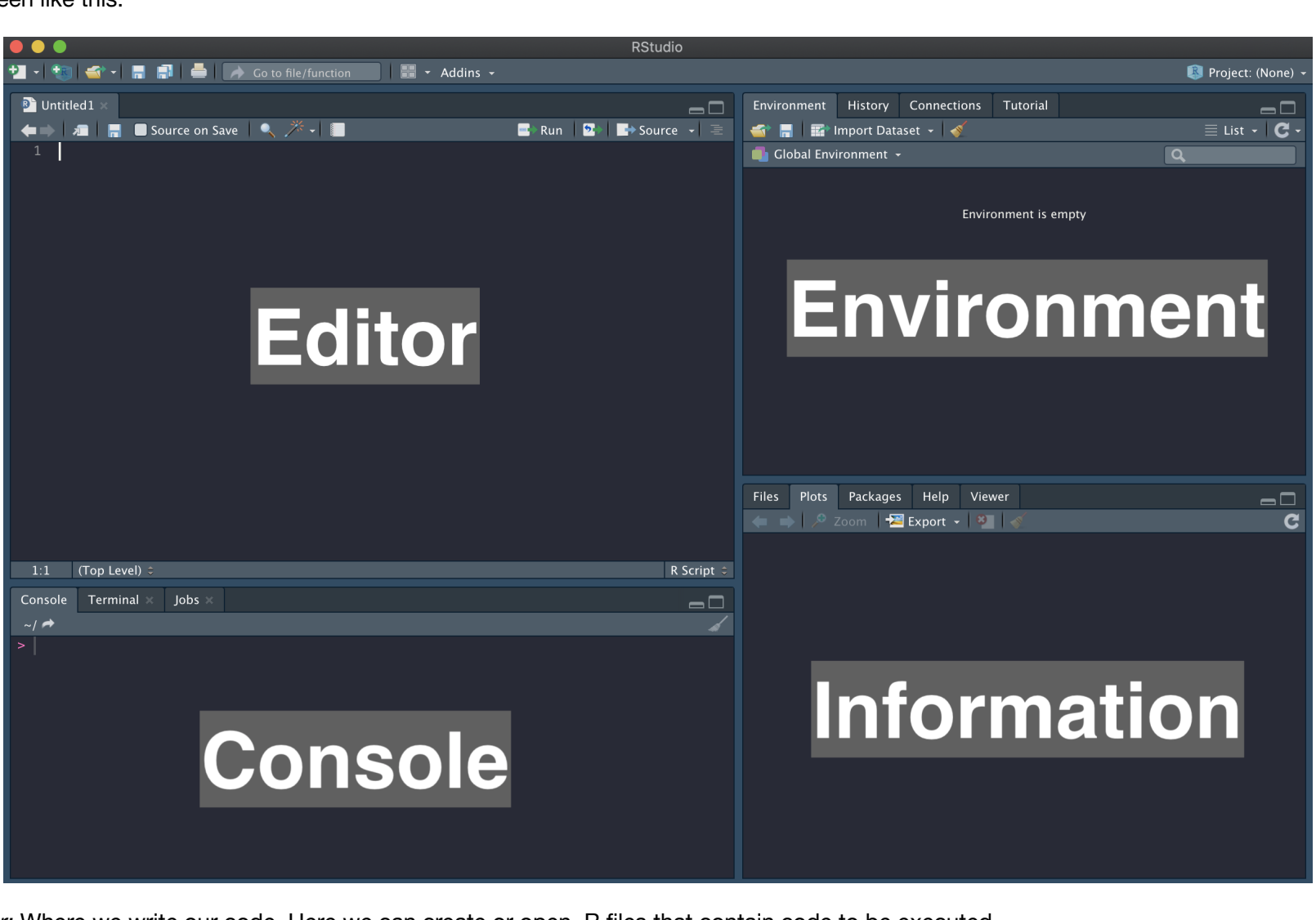
You can download R for free from https://www.r-project.org, by following these steps:

- Click *Download CRAN* in the left bar
- Choose a download site
- Choose your operating system
- Click on the latest release and the download should start

The software is open source, meaning that is supported by a community of developers. This is one of the main advantages of R, since it is constantly updated and offers a wide range of packages. A package is a bundle of code, data and documentation that can be easily downloaded and used in your own projects. We will use some packages for financial data.

RStudio

RStudio is an "Integrated Developer Environment" (IDE), which just means it is an application where you can write your code, execute it, visualize plots, and see the objects you have created. You can download it here: https://rstudio.com. Once you open RStudio, you will see a screen like this:



Editor: Where we write our code. Here we can create or open .R files that contain code to be executed.
Console: The output of the code executed will be shown here.
Information: Here we will see the packages we have installed, plots once we create them, and details and documentation.

Difference between R and RStudio

R and RStudio are two different things, but they work together. R is the programming language, and RStudio is the tool that helps you create programs using R. R can work without RStudio but not the other way around. RStudio can be thought of an interpreter to execute commands, create R scripts, manage R variables, reuse commands from history, visual debugging etc. Basically it lets you code in R easily.

Setting up the Working Directory and creating your first file

To start working in RStudio, we need to first set our *Working Directory*. For our purposes, this should be the folder where we are going to store our data to easily access it. You can set up the Working Directory by going on `Session -> Set Working Directory -> Choose Directory...`, or you can also type in the console `setwd("~/PATH")` with the path of the Directory.

To create your first file, choose `File -> New File -> R Script`. Now you are ready to write your code. To execute any part, you can select it and do `Shift + Enter` or use the button `Run`.

R file vs R Console

An R file (.R) is no more than a text file written in the R language. When we want to run an R file, it is executed in the R Console. You could directly type code into the console and run it, but an R file helps keep your program organized.

Some basic commands

You can use R as a calculator:

In [1]:
```r
# This is a comment, it won't produce any output
3 + 2
14 * 2
0.94^10
```

5

28

0.538615114094899

In R you can store variables, which are just pieces of information like numbers. You create variables to be able to use them in other parts of your code. To assign a value to a variable, you can either use an arrow `<-` or an equal sign `=`. The former is the more correct way, but I prefer using the latter. Once created, you can "call" them by their name:

In [2]:
```r
my_number <- 442
my_number

my_string <- "Hello world"
my_string
```

442

'Hello world'

We will work with vectors and matrices. They can only hold one type of data, either numbers or text. Vectors and matrices are created as follows:

In [3]:
```r
# This is a vector:
vec1 <- c(1,2,3)
vec2 <- c("FM", "442")

# This is a matrix:
mtx1 <- matrix(c(1,2,3,4), nrow = 2, ncol = 2)
mtx2 <- matrix(c("a", "b", "c", "d", "e","f"), nrow = 3, ncol = 2, byrow = TRUE)

vec1
vec2
mtx1
mtx2
```

1 2 3

'FM' '442'

1	3
2	4

a	b
c	d
e	f

In [4]:
```r
# Lenght of a vector
length(vec1)

# Dimensions of a matrix
dim(mtx1)
```

3

2 2

Accessing elements of vectors and matrices

We can access a single element or a subset of vectors and matrices using the brackets `[]` next to the variable's name and specifying the index of the desired elements. For matrices we need to specify the row followed by a comma and the column. If we want an entire row/column, we can leave the space blank:

In [5]:
```r
# Second element of vec2
vec2[2]

# Third element of the second column of mtx2
mtx2[3,2]
```

'442'

'f'

In [6]:
```r
# We can also change an element this way
vec2[1] <- "Finance"
mtx2[3,2] <- "X"

vec2
mtx2
```

'Finance' '442'

a	b
c	d
e	X

Sequences

The function `seq()` allows us to easily create a vector of evenly distributed numbers between a first and last element:

In [7]:
```r
# You need to specify the first and last element, and the increment
seq1 <- seq(2, 10, by = 2)

# With only one input it will create an integer sequence from 1
seq2 <- seq(5)

seq1
seq2
```

2 4 6 8 10

1 2 3 4 5

Data Frames

A data frame is similar to a matrix but it can hold data from different types and can have column names. It is the variable type that we will use the most through the course. You can transform a matrix into a data frame by passing the function `as.data.frame`, or create one from scratch by:

In [8]:
```r
# Create a data with two variables frame using data.frame
x <- data.frame("Stock" = c("A", "B"), "Price" = c(42,68))
x
```

Stock	Price
A	42
B	68

In [9]:
```r
# Accessing a column
x[,2]
x$Price
```

42 68

42 68

In [10]:
```r
# Creating a new column
x$Price_plus_1 <- x$Price + 1
x
```

Stock	Price	Price_plus_1
A	42	43
B	68	69

For-Loops and If-Statements

For-loops and If-Statements are a essential part of programming, allowing us to automate pieces of code.

A `for` loop repeats a piece of code for various elements of an array. The syntax is:

```
for (elements) {
    code to be repeated
}
```

Imagine we want to see the square of the first ten numbers:

In [11]:
```r
for (i in 1:10) { # For every element i in 1, 2, ... 10
    print(i^2) # print() displays the value in the console
}
```

```
[1] 1
[1] 4
[1] 9
[1] 16
[1] 25
[1] 36
[1] 49
[1] 64
[1] 81
[1] 100
```

An `if` statement evaluates a logical claim, which can be either `TRUE` or `FALSE`, and based on that condition executes a piece of code or another. A logical value can be:

- 5 < 10: `TRUE`
- pi = 3: `FALSE`

The syntax of a basic `if` statement is:

```
if (condition) {
    code to be executed if condition is TRUE
} else {
    code to be executed if condition is FALSE
}
```

For example:

In [12]:
```r
x <- 10

if (x > 0) {
    print("x has a positive value")
} else {
    print("x has a negative value")
}
```

```
[1] "x has a positive value"
```

Downloading, importing and manipulating financial data

We will download data on a number of stocks and manipulate it. The database to use is provided by the Center for Research in Security Prices, and is usually known as CRSP. You will access it through a provider called Wharton Research and Data Services (WRDS). To start with, create a student account at https://wrds-web.wharton.upenn.edu/wrds/, as it can take a few days, please do that early.

Ticker, Company Name, PERMNO

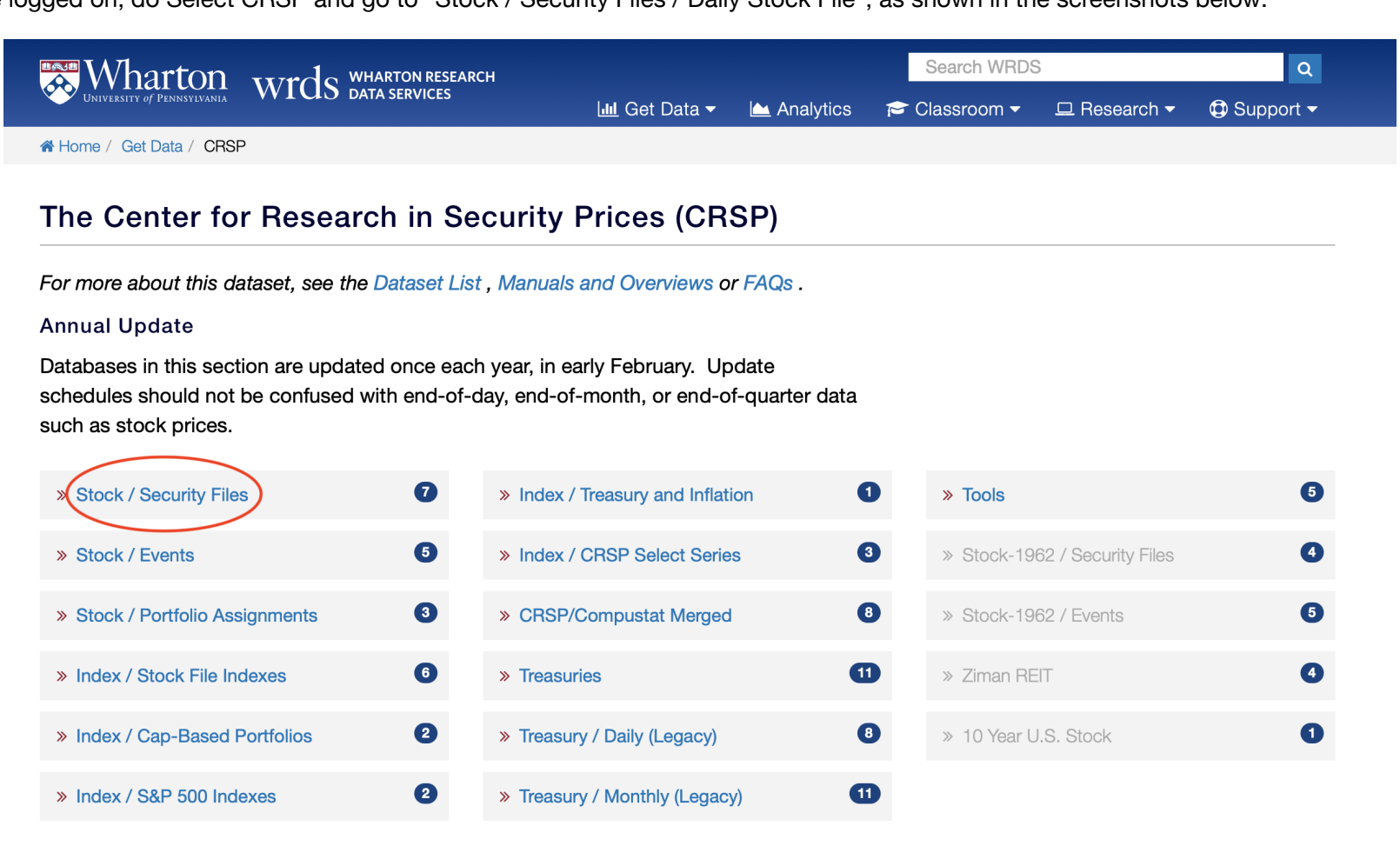
There are different ways of identifying a company in CRSP, and we need to be careful with what we choose. It is very common to associate a stock with its TICKER, but if the company has a merger, this might be subject to change. For example, if we consider JP Morgan (Ticker: JPM), historically it has been officially registered with different names before some mergers and acquisitions happened (Chemical Banking Corp, Chase Manhattan Corp, etc), each which a different ticker, but it is essentially the same company, and by specifying the Ticker JPM we would be losing years of financial data. For this reason, we work with the permanent company number, or PERMNO, which is mantained over time.

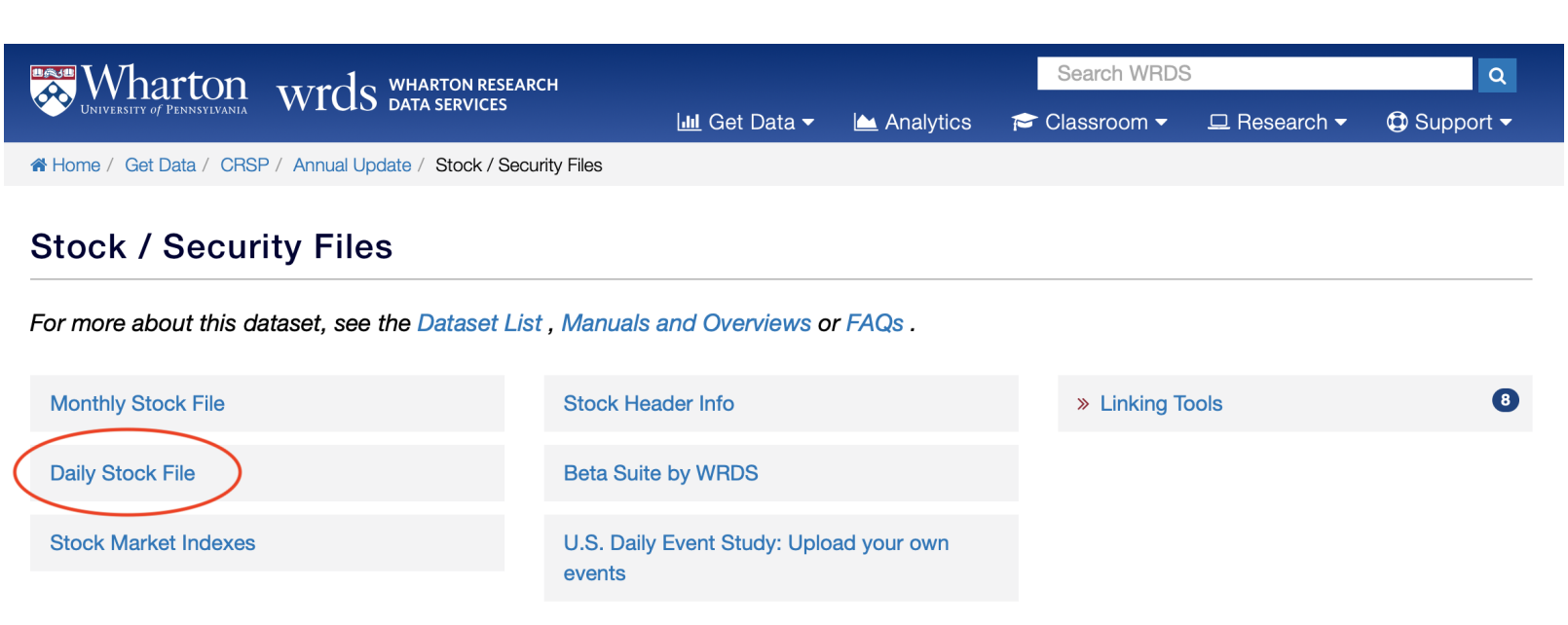
Note about data formats

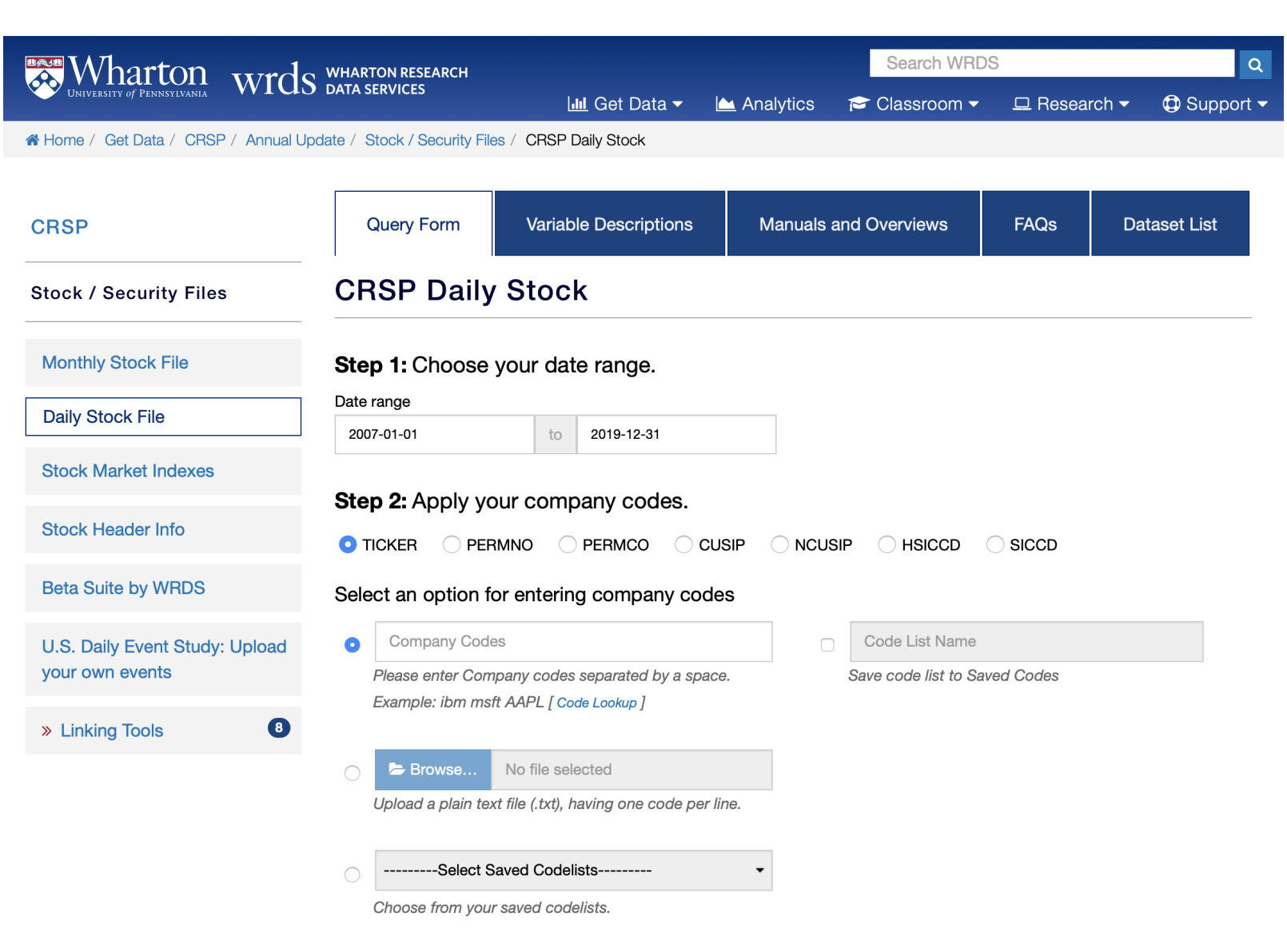
For the purpose of this course, we will mostly be working with *comma-separated values* , or *.csv* files.

Downloading the data

Once logged on, do Select CRSP and go to “Stock / Security Files / Daily Stock File”, as shown in the screenshots below:







Explore the page, and help provided. Then in the steps

1. Choose 1 January 1990 to 31 December 2019
2. Select ticker and codes for Microsoft MSFT, Exxon XOM, General Electric GE, JPMorgan Chase JPM, Intel INTC and Citigroup C
3. Select the following information:
   • From the identifying information: Company Name, Ticker;
   • From the time series information: Price, Holding Period Return;
   • From the distribution information: Cumulative Factor to Adjust Price;
4. Use comma-delimited text and default date format (YYYYMMDDn8) for the output.

Click “Submit Query”
Open the output file in Excel and look for unexpected output.

Redo the exercise but this time selecting PERMNO in Step 2, using 10107 59328 12060 47896 70519 11850
Explain the differences in the two output files.
Save the output into a file in some directory as ‘crsp.csv’

Variable description

All the details on the variables we download can be found in the *Variable Descripions* section of WRDS. It is important to distinguish between the type of returns we are using, whether we are including dividends or not. For example, the description of the time series variables we have downloaded is:

- PRC: "PRC is the closing price or the negative bid/ask average for a trading day. If the closing price is not available on any given trading day, the number in the price field has a negative sign to indicate that it is a bid/ask average and not an actual closing price [...]"
- RET: "A return is the change in the total value of an investment in a common stock over some period of time per dollar of initial investment. RET(I) is the return for a sale on day I. It is based on a purchase on the most recent time previous to I when the se curity had a valid price. Usually, this time is I - 1 [...]"

Importing our data into R

Open RStudio and select the directory you chose in the last step as the Working Directory. In your R script, write and execute:

In [13]:
```r
# This line will just make our lifes easier
options(stringsAsFactors = FALSE)

# Importing the downloaded data
data <- read.csv('crsp.csv')
```

In [14]:
```r
# Checking the dimensions
dim(data)
```

45354 7

In [15]:
```r
# First observations
head(data)
```

PERMNO	date	TICKER	COMNAM	PRC	RET	CFACPR
10107	19900102	MSFT	MICROSOFT CORP	88.750	0.020115	144
10107	19900103	MSFT	MICROSOFT CORP	89.250	0.005634	144
10107	19900104	MSFT	MICROSOFT CORP	91.875	0.029412	144
10107	19900105	MSFT	MICROSOFT CORP	89.625	-0.024490	144
10107	19900108	MSFT	MICROSOFT CORP	91.000	0.015342	144
10107	19900109	MSFT	MICROSOFT CORP	90.750	-0.002747	144

In [16]:
```r
# A single column
head(data$RET)
head(data[,6])
```

0.020115 0.005634 0.029412 -0.02449 0.015342 -0.002747

0.020115 0.005634 0.029412 -0.02449 0.015342 -0.002747

In [17]:
```r
# Names of the columns
names(data)
```

'PERMNO' 'date' 'TICKER' 'COMNAM' 'PRC' 'RET' 'CFACPR'

In [18]:
```r
# Getting unique values of PERMNO
unique(data$PERMNO)
```

10107 11850 12060 47896 59328 70519

In [19]:
```r
# Creating a variable for a company
    # We are filtering the dataset for the rows with the City PERMNO
citi <- data[data$PERMNO == 70519,]

# Dimension
dim(citi)
```

7559 7

In [20]:
```r
# Check the first few elements
head(citi)
```

	PERMNO	date	TICKER	COMNAM	PRC	RET	CFACPR
37796	70519	19900102	PA	PRIMERICA CORP NEW	29.375	0.030702	1.284085
37797	70519	19900103	PA	PRIMERICA CORP NEW	29.750	0.012766	1.284085
37798	70519	19900104	PA	PRIMERICA CORP NEW	29.375	-0.012605	1.284085
37799	70519	19900105	PA	PRIMERICA CORP NEW	29.625	0.008511	1.284085
37800	70519	19900108	PA	PRIMERICA CORP NEW	29.875	0.008439	1.284085
37801	70519	19900109	PA	PRIMERICA CORP NEW	29.500	-0.012552	1.284085

In [21]:
```r
# Check different tickers for the same PERMNO
unique(citi$TICKER)

# Why does this happen?
```

'PA' 'TRV' 'CCI' 'C'

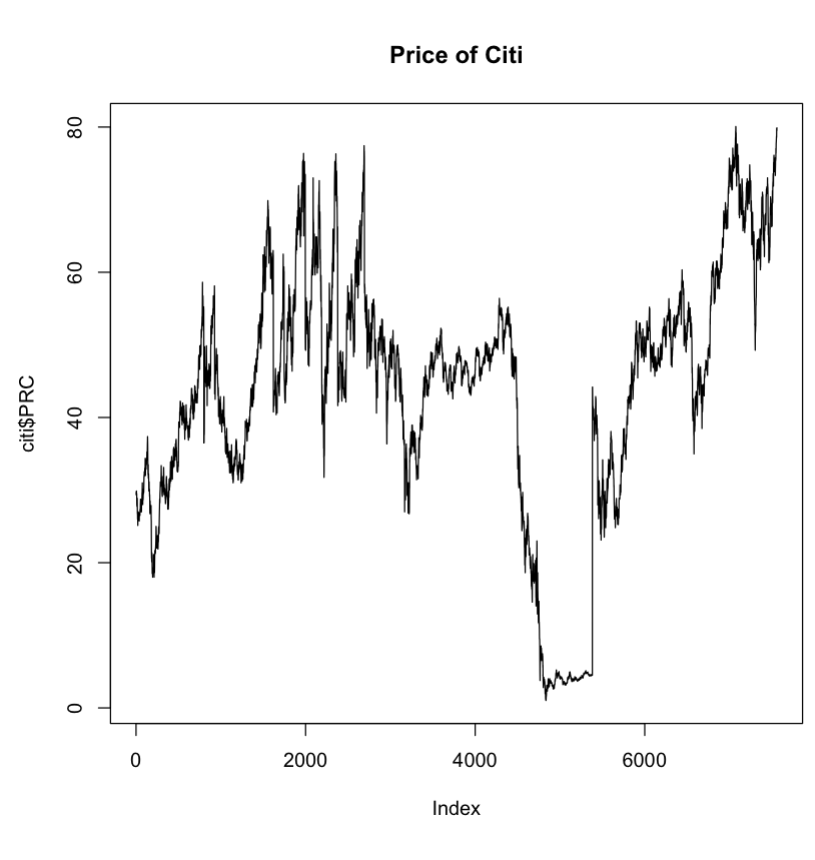
In [22]:
```r
# Highest return for Citi
highest_citi <- max(citi$RET) * 100
paste0("Highest return for Citi: ", highest_citi, "%")
```

'Highest return for Citi: 57.8249%'

Simple plots

R can easily build plots to visualize data. We will use the `plot` command for this. To read the documentation on the command, you can type `?plot` in the console and see all the options it includes.

In [23]:
```r
plot(citi$PRC, type = "l", main = "Price of Citi")
```



Recap

In this seminar we have covered:

- Downloading and working with R and RStudio
- Data types in R, basic operations, accessing elements
- For loops and if statements
- Downloading and importing data from CRSP into R
- Extracting columns from a data frame
- Finding the maximum value of a variable
- Making a simple plot

Some new functions used:

- `matrix()`
- `length()`
- `dim()`
- `seq()`
- `data.frame()`
- `print()`
- `read.csv()`
- `head()`
- `names()`
- `unique()`
- `paste0()`
- `plot()`

For more discussion on the material covered in this seminar, refer to *Chapter 1: Financial markets, prices and risk* on *Financial Risk Forecasting* by Jon Danielsson.

Acknowledgements: Thanks to Alvaro Aguirre for creating these notebooks