R & Python

Crash Course for Finance Students

A Hands-On Guide to Two Languages That Will Power Your Career

Prepared by: Dr. Benyawarath “Yaa” Nithithanatchinnapat
Penn State University • Black School of Business, Erie
Spring 2022

Contents

Section 1: What Are R and Python?

If you're going to work in finance, data analytics, or economics, you'll encounter two programming languages again and again: **R** and **Python**. Think of them as two different power tools in your toolkit. Both can get the job done, but each was designed with a slightly different purpose in mind.

R: The Statistician's Language

R was created *by statisticians, for statisticians*. It was born in the early 1990s at the University of Auckland and quickly became the go-to language in academic research, biostatistics, and econometrics. If you're running a regression, building a time-series forecast, or creating a publication-quality chart, R feels like home.

R's Strengths

- **Statistical powerhouse:** Over 20,000 packages on CRAN (the official R package repository), many created by the world's leading statisticians.
- **Beautiful visualizations:** The ggplot2 package is widely regarded as the best data visualization library in any language. Period.
- **Data manipulation made easy:** The tidyverse family of packages (dplyr, tidyr, etc.) makes cleaning and transforming data almost feel like writing English.
- **Finance-ready:** Packages like quantmod, PerformanceAnalytics, and rugarch are specifically built for financial analysis.
- **Excellent for reproducible research:** RMarkdown lets you combine code, output, and narrative in one document.

R's Limitations

- Steeper learning curve if you've never coded before — the syntax can feel quirky at first.
- Not ideal for building web apps, automating IT tasks, or working with APIs at scale.
- Can be slower with very large datasets compared to Python (though this is rarely an issue for typical finance work).
- Smaller general-purpose community — Stack Overflow help is abundant for statistics, but thinner for other tasks.

Python: The Swiss Army Knife

Python was created in 1991 by Guido van Rossum as a *general-purpose* programming language. It wasn't designed for data science — it was designed to be readable, flexible, and fun to write. But its simplicity attracted data scientists, and now it's the most popular language in data science, machine learning, and AI.

Python's Strengths

- **Beginner-friendly syntax:** Python reads almost like plain English. If you've never coded before, this is the gentler on-ramp.
- **Massive ecosystem:** From web development to machine learning to automation, Python does it all.
- **Machine learning dominance:** Libraries like scikit-learn, TensorFlow, and PyTorch make Python the undisputed champion of ML and AI.
- **API and automation:** Need to pull data from Bloomberg, scrape SEC filings, or automate a daily report? Python excels here.
- **Industry adoption:** Most fintech companies, hedge funds, and tech firms use Python as their primary language.

Python's Limitations

- Statistical packages aren't quite as deep as R's — for advanced econometrics, R still has the edge.
- Visualization (matplotlib) is powerful but less elegant than ggplot2 out of the box.
- Package management can be messy — virtual environments and dependency conflicts are a rite of passage.
- For quick exploratory statistics, R's workflow is often faster and more intuitive.

The Bottom Line: You don't have to choose one. The best data professionals know when to reach for R (deep stats, beautiful charts, econometrics) and when to reach for Python (automation, ML, API integrations, building apps). Learning both makes you versatile and highly employable.

Head-to-Head Comparison

Feature	R	Python
Created For	Statistical computing & research	General-purpose programming
Syntax Style	Functional, can feel quirky	Clean, reads like English
Best For	Econometrics, time series, statistical modeling, publication-quality charts	Automation, ML/AI, API integrations, web scraping, full-stack apps
Visualization	ggplot2 (gold standard)	matplotlib, seaborn, plotly
Data Wrangling	tidyverse (dplyr, tidyr)	pandas, polars
Finance Packages	quantmod, TTR, rugarch, PerformanceAnalytics	yfinance, pandas-datareader, QuantLib, zipline
Learning Curve	Moderate — quirky syntax but powerful once learned	Gentle — closest to natural language

Feature	R	Python
Job Market	Strong in research, insurance, pharma, central banking	Dominant in tech, fintech, hedge funds, consulting

Section 2: Recommended Tools and IDEs

Before you write a single line of code, you need a place to write it. Here's what we recommend for each language, along with the reasoning behind each choice.

For R: RStudio (Now Called Posit)

This is the undisputed champion for R development. RStudio gives you a **console** (where code runs), a **script editor** (where you write and save code), an **environment panel** (showing your variables and data), and a **plots/help panel** — all in one window. It's free, it's beautiful, and it just works.

Getting Started with RStudio

1. **Install R first:** Go to r-project.org and download R for your operating system.
2. **Install RStudio:** Go to posit.co and download RStudio Desktop (free edition).
3. **Open RStudio:** You'll see four panels. The bottom-left is your console — type code there and hit Enter to run it.

For Python: Three Great Options

Option A: Google Colab (Recommended for Beginners)

If you want to start coding Python *right now* without installing anything, Google Colab is your best friend. It's a free, browser-based notebook that runs on Google's servers. You write code in "cells," hit play, and see results instantly. It even comes with popular data science libraries pre-installed.

- **Pros:** Zero setup, free GPU access, easy sharing via Google Drive, perfect for learning.
- **Cons:** Requires internet, sessions expire after inactivity, not ideal for large production projects.
- **Get started:** Go to colab.research.google.com and sign in with your Google account.

Option B: Jupyter Notebook (Local Version of Colab)

Jupyter is essentially what Google Colab is based on, but it runs on your own computer. Same cell-based approach, same interactive style — but you own your data and don't need internet.

- **Install:** The easiest way is to install Anaconda (anaconda.com), which bundles Python + Jupyter + all the data science libraries together.

Option C: VS Code (For Students Who Want to Level Up)

Visual Studio Code is a free, powerful code editor from Microsoft. It's what professional developers use. It handles Python, R, SQL, JavaScript, and just about every other language. If

you plan to pursue a career in tech, data engineering, or software development, learning VS Code now is a smart investment.

- **Install:** Download from code.visualstudio.com, then add the Python extension from the marketplace.

🎓 **Recommendation for This Course:** Start with Google Colab for Python (zero friction) and RStudio for R (purpose-built). Once you're comfortable, explore VS Code as your all-in-one environment.

Section 3: Getting Started with R

Let's learn R by doing. Every example below is code you can type directly into the RStudio console and run. We'll build up from the absolute basics to finance-relevant tasks.

3.1 Variables and Data Types

In R, you assign values using `<-` (the assignment arrow). Think of a variable as a labeled box where you store information.

```
# Assigning variables in R
stock_price <- 152.75       # numeric
ticker <- "AAPL"            # character (text)
is_profitable <- TRUE       # logical (TRUE/FALSE)
shares_owned <- 100L        # integer (the L makes it an integer)

# Check the type of a variable
class(stock_price)   # Returns: "numeric"
class(ticker)        # Returns: "character"
```

3.2 Vectors: Your First Data Structure

A vector is a list of values, all the same type. It's the fundamental building block in R. You create one with `c()` (which stands for "combine").

```
# Create a vector of daily stock prices
prices <- c(150.25, 152.50, 148.75, 155.00, 153.25)

# Basic operations work on the whole vector at once
mean(prices)          # Average: 151.95
max(prices)           # Highest: 155.00
min(prices)           # Lowest: 148.75
sd(prices)            # Standard deviation: 2.39

# Calculate daily returns
returns <- diff(prices) / prices[-length(prices)]
returns               # Percentage changes between each day
```

3.3 Data Frames: Working with Tables

A data frame is R's version of a spreadsheet. Columns can hold different types of data (numbers, text, dates), and each row is an observation. If you're coming from Excel, this will feel very familiar.

```
# Create a small portfolio data frame
portfolio <- data.frame(
  ticker    = c("AAPL", "MSFT", "JPM", "JNJ"),
  shares    = c(50, 30, 100, 75),
  buy_price = c(145.00, 310.50, 155.75, 170.25),
  sector    = c("Tech", "Tech", "Finance", "Healthcare")
)
```

```
# View the data frame
print(portfolio)

# Access a single column with $
portfolio$ticker         # Returns all ticker symbols
portfolio$buy_price      # Returns all buy prices

# Calculate total investment per stock
portfolio$total_cost <- portfolio$shares * portfolio$buy_price
```

3.4 Reading Data from a CSV File

In the real world, you'll rarely type data in by hand. You'll load it from a file. Here's how:

```
# Read a CSV file into a data frame
stock_data <- read.csv("stock_prices.csv")

# Quick look at the data
head(stock_data)         # First 6 rows
str(stock_data)          # Structure (column types)
summary(stock_data)      # Statistical summary of each column
nrow(stock_data)         # Number of rows
```

3.5 The Tidyverse: Data Manipulation Made Beautiful

The tidyverse is a collection of R packages that makes data wrangling feel almost like writing a sentence. The key package is `dplyr`, and the magic ingredient is the pipe operator `|>` (or `%>%`), which chains operations together.

```
# Install and load tidyverse (one-time install)
install.packages("tidyverse")
library(tidyverse)

# Filter, transform, and summarize in one pipeline
portfolio |>
  filter(sector == "Tech") |>              # Keep only tech stocks
  mutate(current_val = shares * 165.00) |>  # Add a new column
  summarize(
    total_invested = sum(total_cost),
    total_current  = sum(current_val)
  )
```

3.6 Your First Chart with ggplot2

One of R's superpowers is visualization. The ggplot2 package uses a "grammar of graphics" approach — you build charts layer by layer, like painting on a canvas.

```
library(ggplot2)

# Simple line chart of stock prices over time
dates <- as.Date(c("2025-01-06", "2025-01-07", "2025-01-08",
```

```
                    "2025-01-09", "2025-01-10"))
price_data <- data.frame(date = dates, price = prices)

ggplot(price_data, aes(x = date, y = price)) +
  geom_line(color = "steelblue", linewidth = 1.2) +
  geom_point(color = "steelblue", size = 3) +
  labs(title = "AAPL Stock Price (5-Day)",
       x = "Date", y = "Price ($)") +
  theme_minimal()
```

Section 4: Getting Started with Python

Now let's cover the same ground in Python. You'll notice the logic is very similar — the syntax just looks a little different. That's the whole point of learning both: the concepts transfer, only the "spelling" changes.

4.1 Variables and Data Types

Python uses = for assignment (no arrow needed). Variables don't need type declarations — Python figures it out automatically.

```
# Assigning variables in Python
stock_price = 152.75        # float (decimal number)
ticker = "AAPL"             # string (text)
is_profitable = True        # boolean (True/False)
shares_owned = 100          # integer

# Check the type
type(stock_price)   # Returns: <class 'float'>
type(ticker)        # Returns: <class 'str'>
```

4.2 Lists: Python's Flexible Container

A list in Python is similar to R's vector, but more flexible — it can hold mixed types. For numerical work, we'll mostly use `numpy` arrays (coming up next).

```
# A simple list
prices = [150.25, 152.50, 148.75, 155.00, 153.25]

# Basic list operations
len(prices)          # Length: 5
max(prices)          # Highest: 155.00
min(prices)          # Lowest: 148.75
sum(prices) / len(prices)  # Average: 151.95

# NumPy makes math much easier
import numpy as np
prices_arr = np.array(prices)
np.mean(prices_arr)   # 151.95
np.std(prices_arr)    # Standard deviation
```

4.3 DataFrames with pandas

The `pandas` library gives Python its version of data frames. If you know Excel, pandas will feel very natural.

```
import pandas as pd

# Create a portfolio DataFrame
portfolio = pd.DataFrame({
    'ticker':    ['AAPL', 'MSFT', 'JPM', 'JNJ'],
```

```
    'shares':    [50, 30, 100, 75],
    'buy_price': [145.00, 310.50, 155.75, 170.25],
    'sector':    ['Tech', 'Tech', 'Finance', 'Healthcare']
})

# View the DataFrame
print(portfolio)

# Access a column
portfolio['ticker']
portfolio['buy_price']

# Calculate total cost
portfolio['total_cost'] = portfolio['shares'] * portfolio['buy_price']
```

4.4 Reading Data from a CSV File

```
import pandas as pd

# Read a CSV file
stock_data = pd.read_csv('stock_prices.csv')

# Quick look at the data
stock_data.head()            # First 5 rows
stock_data.info()            # Column types and memory
stock_data.describe()        # Statistical summary
len(stock_data)              # Number of rows
```

4.5 Filtering and Transforming Data

Python's pandas gives you powerful filtering and transformation, similar to the tidyverse in R:

```
# Filter for tech stocks only
tech_stocks = portfolio[portfolio['sector'] == 'Tech']

# Add a new column for current value
portfolio['current_val'] = portfolio['shares'] * 165.00

# Group by sector and summarize
sector_summary = portfolio.groupby('sector').agg({
    'total_cost': 'sum',
    'current_val': 'sum'
}).reset_index()

print(sector_summary)
```

4.6 Your First Chart with matplotlib

Python's `matplotlib` is the foundational plotting library. It's not as pretty as ggplot2 out of the box, but it's highly customizable:

```
import matplotlib.pyplot as plt

dates = ['Jan 6', 'Jan 7', 'Jan 8', 'Jan 9', 'Jan 10']
prices = [150.25, 152.50, 148.75, 155.00, 153.25]

plt.figure(figsize=(10, 5))
plt.plot(dates, prices, color='steelblue', linewidth=2, marker='o')
plt.title('AAPL Stock Price (5-Day)', fontsize=14)
plt.xlabel('Date')
plt.ylabel('Price ($)')
plt.grid(True, alpha=0.3)
plt.tight_layout()
plt.show()
```

Section 5: Side-by-Side — The Same Task in Both Languages

This is where it all clicks. Below, we show the same real finance tasks done in both R and Python. Notice how the logic is identical — only the syntax changes. This is the skill that makes you versatile.

Task 1: Download Stock Price Data

The scenario: You want to pull Apple's historical stock prices for 2024.

In R (using quantmod)

```
library(quantmod)

# Download AAPL stock data
getSymbols("AAPL", src = "yahoo",
           from = "2024-01-01", to = "2024-12-31")

# View the first few rows
head(AAPL)

# Plot it
chartSeries(AAPL, theme = chartTheme("white"))
```

In Python (using yfinance)

```
import yfinance as yf

# Download AAPL stock data
aapl = yf.download('AAPL', start='2024-01-01', end='2024-12-31')

# View the first few rows
print(aapl.head())

# Plot it
aapl['Close'].plot(title='AAPL Closing Price 2024', figsize=(10,5))
```

Task 2: Calculate Daily Returns

The scenario: You have closing prices and need to compute percentage returns.

In R

```
# Using quantmod's built-in function
daily_returns <- dailyReturn(Cl(AAPL))

# Or manually
closing <- as.numeric(Cl(AAPL))
```

```
manual_returns <- diff(closing) / closing[-length(closing)]

# Summary statistics
mean(daily_returns)      # Average daily return
sd(daily_returns)        # Daily volatility
```

In Python

```
# Using pandas pct_change()
daily_returns = aapl['Close'].pct_change().dropna()

# Summary statistics
daily_returns.mean()     # Average daily return
daily_returns.std()      # Daily volatility
```

Task 3: Run a Simple Linear Regression

The scenario: You want to test whether a stock's returns are related to market returns (a basic CAPM-style regression).

In R

```
# Assume you have data frame with stock_return and market_return columns
model <- lm(stock_return ~ market_return, data = returns_df)

# View regression results
summary(model)

# The beta coefficient
coef(model)["market_return"]
```

In Python

```
import statsmodels.api as sm

# Add a constant for the intercept
X = sm.add_constant(returns_df['market_return'])
y = returns_df['stock_return']

# Fit the model
model = sm.OLS(y, X).fit()

# View results
print(model.summary())

# The beta coefficient
model.params['market_return']
```

Task 4: Track Portfolio Value Over Time

The scenario: You own multiple stocks and want to visualize your total portfolio value.

In R

```
library(tidyverse)

# Assume portfolio_data has columns: date, ticker, value
total_by_date <- portfolio_data |>
  group_by(date) |>
  summarize(total_value = sum(value))

ggplot(total_by_date, aes(x = date, y = total_value)) +
  geom_line(color = "darkgreen", linewidth = 1) +
  geom_area(fill = "lightgreen", alpha = 0.3) +
  labs(title = "Portfolio Value Over Time",
       y = "Total Value ($)") +
  scale_y_continuous(labels = scales::dollar) +
  theme_minimal()
```

In Python

```
import matplotlib.pyplot as plt

# Assume portfolio_data has columns: date, ticker, value
total_by_date = portfolio_data.groupby('date')['value'].sum()

plt.figure(figsize=(10, 5))
plt.fill_between(total_by_date.index, total_by_date.values,
                 color='lightgreen', alpha=0.3)
plt.plot(total_by_date.index, total_by_date.values,
         color='darkgreen', linewidth=1)
plt.title('Portfolio Value Over Time')
plt.ylabel('Total Value ($)')
plt.tight_layout()
plt.show()
```

Section 6: Finance and Economics Use Cases

Here's where things get exciting. Both R and Python are used every day by financial analysts, economists, quants, and risk managers. Below is a guide to what each language excels at in real-world finance.

Use Cases Where R Shines

Use Case	What You'd Do	Key Packages
Econometric Modeling	Run OLS, panel data regressions, IV models for policy analysis	fixest, plm, AER, sandwich
Time Series Forecasting	Forecast stock prices, interest rates, GDP growth with ARIMA, GARCH	forecast, rugarch, tseries
Portfolio Analytics	Calculate risk metrics, Sharpe ratios, drawdowns, VaR	PerformanceAnalytics, PortfolioAnalytics
Statistical Visualization	Create publication-quality charts for reports or research papers	ggplot2, plotly, highcharter
Academic Research	Reproduce studies, run simulations, produce RMarkdown reports	rmarkdown, knitr, stargazer

Use Cases Where Python Shines

Use Case	What You'd Do	Key Libraries
Automated Trading	Build algorithmic trading strategies, backtest them, deploy to production	zipline, backtrader, alpaca-trade-api
Credit Risk / ML Models	Predict loan defaults, fraud detection, customer segmentation	scikit-learn, XGBoost, LightGBM
Financial Data Pipelines	Pull data from APIs (Bloomberg, FRED, SEC), clean and store it	requests, pandas-datareader, fredapi
NLP for Finance	Analyze earnings call transcripts, sentiment from news, SEC filings	spaCy, transformers, nltk
Web Scraping	Scrape financial data from websites, PDF reports, databases	BeautifulSoup, Selenium, tabula-py
Dashboard / Reporting	Build interactive dashboards for stakeholders	Streamlit, Dash, Panel

When Finance Teams Use Both

In practice, many finance teams use R and Python together. Here's a common workflow:

1. **Data Collection (Python):** Use Python to pull data from APIs, scrape websites, and build automated data pipelines.
2. **Statistical Analysis (R):** Pass the cleaned data to R for econometric modeling, hypothesis testing, and GARCH volatility estimation.
3. **Machine Learning (Python):** Build predictive models (default prediction, asset pricing) in Python using scikit-learn or XGBoost.
4. **Visualization and Reporting (R):** Create publication-quality charts in ggplot2 and produce polished reports with RMarkdown.
5. **Deployment (Python):** Deploy the final model or dashboard using Streamlit or Flask for stakeholders to interact with.

Section 7: Quick Reference Cheat Sheet

Keep this page bookmarked. When you forget how to do something, look here first.

Task	R	Python
Assign a variable	`x <- 42`	`x = 42`
Create a list/vector	`c(1, 2, 3)`	`[1, 2, 3]`
Print to console	`print(x)`	`print(x)`
Read a CSV	`read.csv("file.csv")`	`pd.read_csv('file.csv')`
First 5 rows	`head(df)`	`df.head()`
Number of rows	`nrow(df)`	`len(df)`
Filter rows	`filter(df, col > 5)`	`df[df['col'] > 5]`
Add a column	`df$new <- df$a * 2`	`df['new'] = df['a'] * 2`
Group & summarize	`group_by() |> summarize()`	`df.groupby().agg()`
Average	`mean(x)`	`np.mean(x)`
Standard deviation	`sd(x)`	`np.std(x)`
Correlation	`cor(x, y)`	`np.corrcoef(x, y)`
Linear regression	`lm(y ~ x, data)`	`sm.OLS(y, X).fit()`
Line chart	`ggplot() + geom_line()`	`plt.plot()`
Install a package	`install.packages("pkg")`	`pip install pkg`
Load a package	`library(pkg)`	`import pkg`

Section 8: What's Next?

You've now got a working foundation in both R and Python. That's no small thing. Here's how to keep building momentum:

Immediate Next Steps

1. **Practice with real data:** Download stock data using quantmod (R) or yfinance (Python) and explore it. Calculate returns, plot charts, run a regression. The more you practice with real financial data, the faster you'll improve.
2. **Pick one language to go deeper in:** You don't need to master both simultaneously. Pick the one that aligns with your career goals, go deep, and circle back to the other later.

3. **Join the community:** Follow #rstats and #python on X/Twitter. Subscribe to R-bloggers.com and realpython.com. Ask questions on Stack Overflow. The communities for both languages are incredibly helpful.

Recommended Free Resources

For R

- *R for Data Science* by Hadley Wickham (free online at r4ds.hadley.nz)
- *Swirl* — an R package that teaches R interactively inside RStudio
- *Tidy Finance with R* (free online at tidy-finance.org) — finance-specific R tutorials

For Python

- *Python for Finance* by Yves Hilpisch — the definitive guide
- *Automate the Boring Stuff with Python* (free online) — great for general Python skills
- *Kaggle Learn* — free, bite-sized Python courses with hands-on exercises

🚀 **Remember:** The goal isn't to memorize syntax. It's to understand the logic of working with data. Once you grasp the concepts — variables, data frames, filtering, visualization, modeling — switching between R and Python becomes like switching between Spanish and Portuguese. Different words, same ideas.

Happy coding!

Penn State Behrend • Spring 2022