Data Science & Analysis Foundations with R and Python — Deep Understanding Guide

Stage 1: Foundations — Statistics, R & Python Basics, Visualisation, Exercises

Author: Maingo Israel

Table of Contents

1. Introduction to Data Science

Data science is the process of extracting knowledge and insights from data. It combines elements of statistics,
programming, domain expertise, and communication. The typical workflow is: define the question, gather data, clean and
preprocess, explore (EDA), model, evaluate, and communicate results.

In this guide we cover foundational statistics and programming skills in both R and Python, provide code examples, and
give exercises that use finance, health, and general datasets. Aim for deep understanding: for each method, ask "why it
works", "what assumptions it makes", and "how to check those assumptions".

2. Core Statistics Refresher

Descriptive statistics summarize data: mean (average), median (middle), variance and standard deviation (spread),
skewness (asymmetry), and kurtosis (tailedness). Use visual and numeric summaries together.

Probability: random variables, expectation, variance. Common distributions: Normal, Student-t, Poisson, Binomial. The
Central Limit Theorem explains why sample means tend toward Normal when sample size is large.

Estimation and hypothesis testing: point and interval estimates, confidence intervals, p-values, and statistical power.
Understand Type I and Type II errors. Always interpret tests within context and effect sizes.

Resampling methods: bootstrap (for estimating sampling distributions) and cross-validation (for model evaluation). These
methods are powerful when analytic solutions are hard or model assumptions are uncertain.

Code example:

```
# R: quick descriptive stats
x <- c(1.2, 3.4, 2.5, 4.0, 5.1)
mean(x); median(x); sd(x); var(x)
# Python equivalent
import numpy as np
x = np.array([1.2, 3.4, 2.5, 4.0, 5.1])
x.mean(), np.median(x), x.std(ddof=1), x.var(ddof=1)
```

3. R Fundamentals

R is a language for statistics and data analysis. Start with RStudio as an IDE. Key data types: numeric, integer,
character, logical, factor, Date. Important data structures: vector, matrix, data.frame, list, tibble.

The tidyverse (dplyr, tidyr, ggplot2) provides consistent data manipulation and plotting grammar. Learn piping (%>%) to
chain operations for readable code.

Best practices: comment code, use meaningful variable names, save scripts and RMarkdown for reproducible reports, and
use version control (Git).

Code example:

```
# Install tidyverse
install.packages("tidyverse")
library(tidyverse)

# Read CSV and basic EDA
df <- read.csv("data.csv")
glimpse(df)
summary(df)
df %>% select(Date, Close) %>% mutate(Return = diff(log(Close))) %>% head()

# Simple ggplot
library(ggplot2)
ggplot(df, aes(x=Date, y=Close)) + geom_line() + labs(title="Price over time")
```

4. Python Fundamentals

Python is a general-purpose language widely used in data science. Use Anaconda and Jupyter or VS Code. Core libraries:
pandas (dataframes), numpy (numerical operations), matplotlib/seaborn/plotly (visualization), scikit-learn (machine
learning), statsmodels (statistical models).

Pandas DataFrame is central: read_csv, head(), info(), describe(), groupby(), merge(), pivot_table(). Practice indexing
and boolean masks—they're essential.

Best practices: virtual environments, notebooks for exploration, scripts/modules for production, docstrings, and tests
for critical code.

Code example:

```
# Python: basic pandas usage
import pandas as pd
import numpy as np

df = pd.read_csv("data.csv", parse_dates=['Date'])
df.info()
df['Return'] = np.log(df['Close']).diff()
df[['Date','Close','Return']].head()

# Simple matplotlib plot
import matplotlib.pyplot as plt
plt.figure()
plt.plot(df['Date'], df['Close'])
plt.title("Price over time")
plt.xlabel("Date")
plt.ylabel("Close")
plt.show()
```

5. Practical Visualization (R & Python)

Visualization is essential for EDA. Use plots to reveal trends, seasonality, outliers, and relationships. Line plots are
for time series; histograms/density plots for distribution; scatter plots for relationships; boxplots for spread and
outliers.

When plotting time series, always check frequency, missing dates, and annotate events. Combine plots with summary
statistics.

Code example:

```
# R: time series plot with ggplot2
library(ggplot2)
df$Date <- as.Date(df$Date)
ggplot(df, aes(x=Date, y=Return)) + geom_line() + geom_smooth(method='loess') + labs(title="Log returns'

# Python: histogram + KDE
df['Return'].hist(bins=50)
df['Return'].plot(kind='kde')
```

6. Practice Exercises

Exercises are designed to force active learning. For each exercise, write code in both R and Python and explain results
in one paragraph.

Exercise 1: Compute descriptive statistics (mean, median, std, skewness, kurtosis) for daily returns of a chosen stock
or commodity. Plot histogram and time series.

Exercise 2: Use bootstrap to create a 95% CI for the mean return. Explain the bootstrap procedure and interpret the
interval.

Exercise 3: Load a public health dataset (e.g., patient wait times). Clean missing values, compute summaries by group,
and visualize differences.

Exercise 4: Split a dataset into train/test, fit a simple linear regression and report RMSE. Then compare with a tree-
based model (random forest).

7. Mini Projects & Suggested Datasets

Mini Project A (Finance): Exploratory Analysis of JSE ALSI (or alternative index). Compute log returns, plot volatility,
fit ARIMA and simple GARCH(1,1) in R (rugarch) or Python (arch package). Compare models and write concise
interpretation.

Mini Project B (Health): Analyze hospital admission times. Identify peak hours/days, model counts with Poisson or
negative binomial regression, and visualize results.

Mini Project C (Economic Indicators): Compare GDP growth and unemployment across countries; create small dashboards with
plotly or R Shiny.

Suggested datasets: Kaggle, Yahoo Finance (CSV), World Bank (CSV), UCI repository, South African data portals
(data.gov.za).

Appendix: Useful Resources and Next Steps

Appendix: Recommended textbooks and online courses: 'An Introduction to Statistical Learning' (ISLR), 'Elements of Statistical Learning' (Hastie et al.), Andrew Ng's ML course, 'Python for Data Analysis' (Wes McKinney). For time series: 'Time Series Analysis' (Box & Jenkins), 'Forecasting: principles and practice' (Hyndman & Athanasopoulos).

Next steps (after Stage 1): Stage 2 — Data Wrangling & Visualization deep dive; Stage 3 — Machine Learning fundamentals; Stage 4 — Time Series & GARCH; Stage 5 — Projects & Portfolio.

Appendix: Quick Reference - Selected Code Snippets

R: tidy data and dplyr

```
library(tidyverse)
df %>% drop_na() %>% mutate(Return = log(Close) - lag(log(Close))) %>% filter(!is.na(Return)) %>% summar
```

Python: pandas groupby and pivot

```
# groupby example
df.groupby('Country')['GDP'].mean().sort_values(ascending=False).head()

# pivot example
df.pivot_table(index='Year', columns='Country', values='GDP')
```

R: simple ARIMA with forecast

```
library(forecast)
tsr <- ts(df$Return, frequency=252) # approx trading days
fit <- auto.arima(tsr)
forecast::forecast(fit, h=10)
```

Python: simple scikit-learn regression

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
X = df[['Feature1','Feature2']].fillna(0)
y = df['Target']
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2,random_state=42)
model = LinearRegression().fit(X_train,y_train)
mean_squared_error(y_test, model.predict(X_test), squared=False)
```

Final notes

Study actively: read, code, and explain results in writing. Use both R and Python for the same tasks to build transferable intuition.

Keep a GitHub repo with notebooks, README, and clear explanations. Include small reports (RMarkdown or Jupyter Notebooks) for each mini project.