

Data Analytics Engineering

For Accountants and Auditors

Stewart Li

2023-11-09

Table of contents

Preface	3
I Infrastructure	4
1 Local	6
2 ELT	18
II Data tools	19
3 Polars	23
4 Analysis	28
4.1 IO	28
4.2 Cleaning	29
4.3 Validate	29
4.4 Munging	30
4.5 EDA	31
4.6 Model	33
4.7 Report	34
References	36

Preface

This book documents the data analytics engineering workflow, which contains two parts namely infrastructure and tools. It focuses on its implementation instead of its setup. macOS is left out as Windows OS is widely used in the business setting. Pick the preferred tools after considered your career path. For instance, data/dev ops, data/analytic/ML engineer, and data analyst/scientist. My goal is to have a better solution to do auditing/accounting job easily (powerful tools), accurately (reproducible process), and automatically (job scheduler). If you don't know what I am talking about, watch [data firm](#) and [financial statement preparation](#), and read the paper (Li, Fisher, and Falta 2020).

You might ask how it relates to you. Generally, CFO is charge of COA, Audit partner emphasize accounting treatments, and staffs do their job at the transactional level. You need much better tools to pan out at work. For instance,

1. New job requires the strong analytic mind. Excel or similar tools are not sufficient for pattern recognition.
2. A higher staff turnover is caused by pressure and boredom. You need to be efficient by automating repetitive work such as reconciliation.

Part I

Infrastructure

The knowledge of linux (Ubuntu LTS) terminal will be beneficial when you use remote AWS services. For instance,

1. `awscli`, `terraform`,
2. `docker`, `podman`, `k8`,

ELT seems better than ETL as you normally don't know the part of transformation upfront.

1 Local

My **OS** is Windows 11. Install Windows Terminal, WSL2, and Linux distribution systems. Edit terminal theme/font, dotfiles of Bash/Tmux/Vim, and env variables. Install Git/GitBash and Docker/Podman if needed.

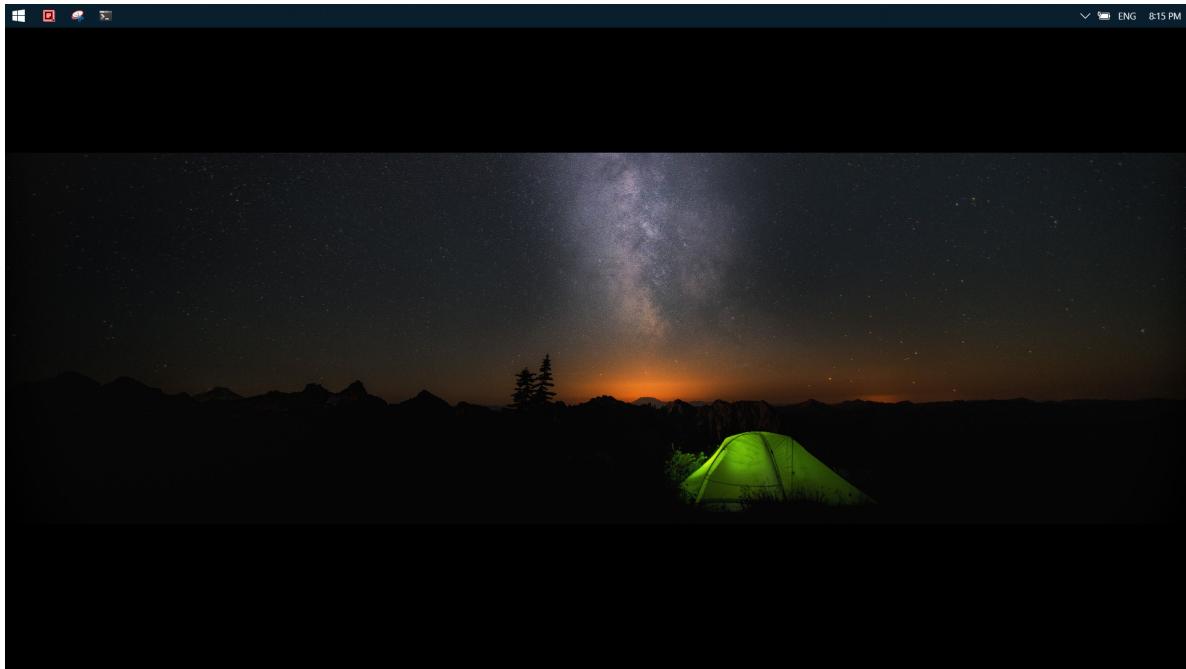


Figure 1.1: Desktop

Install **programming** languages R/Python/DuckDB/Rust/Go. R is a language designed to get shit done ([@hadleywickham](#)). Python is a glue language. Rust is a decent language for software engineering. I often live in terminal to manage `pass`, `rsync` files, `quarto` markdown, `sftp` to server, `ssh` into remote machine, and do a quick analysis for ad hoc tasks.

Editors like nano (Linux) and notepad (Windows) can be used for their simplicity. However, appropriate **IDE** helps you organize your project better. I choose Vim (Linux), RStudio (Windows), and VS Code (Both) based on the active development environment. Of course, RStudio can be launched in Linux as well.

```

Microsoft Windows [Version 10.0.19045.3440]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Stewart Li>systeminfo

Host Name: DESKTOP-HCEU07A
OS Name: Microsoft Windows 10 Home
OS Version: 10.0.19045 N/A Build 19045
OS Manufacturer: Microsoft Corporation
OS Configuration: Standalone Workstation
OS Build Type: Multiprocessor Free
Registered Owner: Stewart Li
Registered Organization: Microsoft
Product ID: K8S25-96013-32346-AA0EM
Original Install Date: 3/8/2021, 6:49:39 PM
System Boot Time: 10/9/2023, 12:31:39 PM
System Manufacturer: Dell Inc.
System Model: XPS 13 9360
System Type: x64-based PC
Processor(s): 1 Processor(s) Installed.
[001]: Intel® Family 6 Model 142 Stepping 9 GenuineIntel ~2701 MHz
BIOS Version: Dell Inc. 2.21.0, 6/2/2022
Windows Directory: C:\WINDOWS
System Directory: C:\WINDOWS\system32
Boot Device: \Device\harddiskVolume1
System Locale: en-US (English (United States))
Input Locale: en-US (English (United States))
Time Zone: (UTC+08:00) Kuala Lumpur, Singapore
Total Physical Memory: 8,077 MB
Available Physical Memory: 1,293 MB
Virtual Memory: Max Size: 14,733 MB
Virtual Memory: Available: 5,154 MB
Virtual Memory: In Use: 9,579 MB
Page File Location(s): C:\pagefile.sys
Domain: WORKGROUP
Logon Server: \DESKTOP-HCEU07A
Hotfix(s): 27 Hotfix(s) Installed.
[001]: KB5029932
[002]: KB5062430
[003]: KB5062525
[004]: KB5089481
[005]: KB5003791
[006]: KB5012170
[007]: KB5015684
[008]: KB5030211
[009]: KB5006753

```

Figure 1.2: CMD

```

Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\Stewart Li> Get-ComputerInfo

WindowsBuildLabEx : 19041.1.amd64fre.vb_release.191206-1406
WindowsCurrentVersion : 6.3
WindowsEditionId : Core
WindowsInstallationType : Client
WindowsInstallDateFromRegistry : 3/8/2021 10:49:39 AM
WindowsProductId : 00325-96013-32346-AA0EM
WindowsProductName : Windows 10 Home
WindowsRegisteredOrganization : Microsoft
WindowsRegisteredOwner : Stewart Li
WindowsSystemRoot : C:\WINDOWS
WindowsVersion : 2009
BiosCharacteristics : {7, 9, 11, 12...}
BiosTosVersion : (DELL - 1072009, 2.21.0, American Megatrends - 50008)
BiosNumber : 
BiosCaption : 
BiosCodeSet : 2.21.0
BiosCurrentLanguage : en[US]iso8859-1
BiosDescription : 2.21.0
BiosEmbeddedControllerMajorVersion : 255
BiosEmbeddedControllerMinorVersion : 255
BiosFirmwareType : Uefi
BiosIdentificationCode : 
BiosInstallableLanguages : 2
BiosInstallDate : 
BiosLanguageEdition : 
BiosListofLanguages : {en[US]iso8859-1, }
BiosManufacturer : Dell Inc.
BiosName : 
BiosOtherTargetOS : True
BiosPrimaryBios : 6/2/2022 8:00:00 AM
BiosReleaseDate : 1G73RC2
BiosSerialNumber : 2.21.0
BiosMBIOSIOSVersion : 
BiosSMBIOSMajorVersion : 3
BiosSMBIOSMinorVersion : 0
BiosMBIOSPresent : True
BiosSoftwareElementState : Running
BiosStatus : OK

```

Figure 1.3: PowerShell

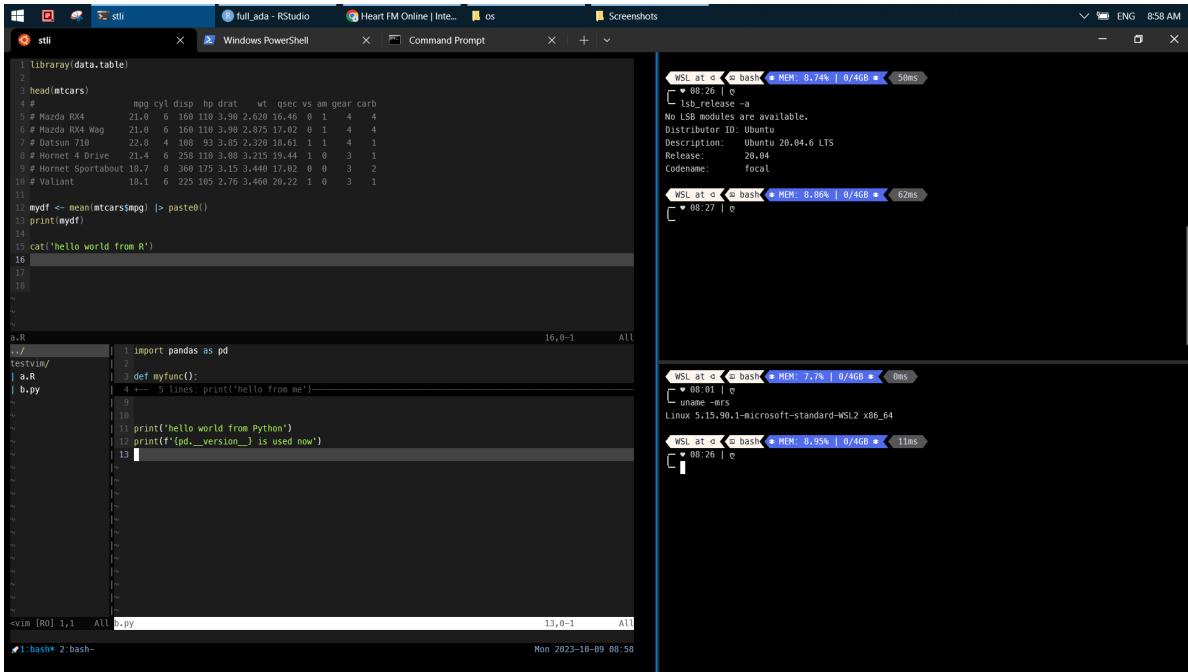


Figure 1.4: Ubuntu

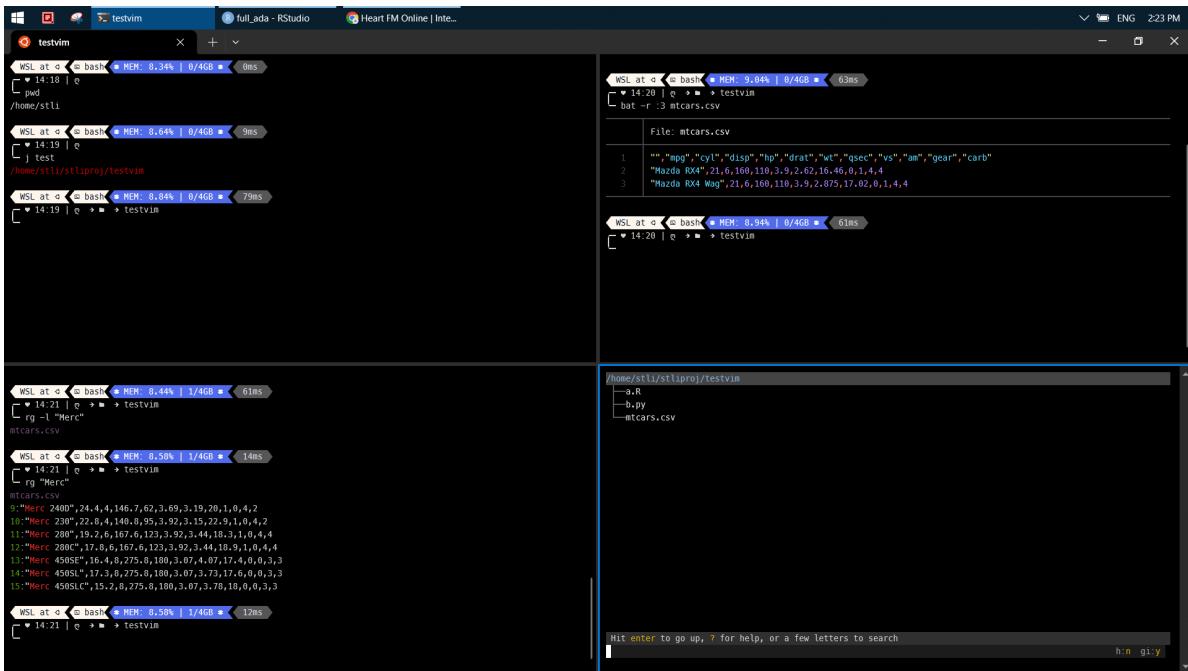


Figure 1.5: Terminal tools

WSL at < bash * MEM: 9.5% | 1/4GB * 8ms
 15:23 | & testvim

```
R
R version 4.3.1 (2023-06-16) -- "Beagle Scouts"
Copyright (C) 2023 The R Foundation for Statistical Computing
Platform: x86_64-pc-linux-gnu (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> write.csv(mtcars, 'mtcars.csv')
>

WSL at < bash * MEM: 10.0% | 1/4GB * 9ms  

  15:29 | & testvim
```

python3
 Python 3.8.10 (default, May 26 2023, 14:05:08)
 [GCC 9.4.0] on linux
 Type "help", "copyright", "credits" or "license" for more information.
 >>> import pandas as pd
 >>> df=pd.read_csv("mtcars.csv")
 >>> df.describe()
 mpg cyl disp hp ... vs am gear carb
 count 32.000000 32.000000 32.000000 32.000000 32.000000 32.000000
 mean 20.090625 6.167500 230.721675 146.687500 ... 0.437500 0.406250 3.667500 2.8125
 std 6.026948 1.785922 123.330694 56.562864 ... 0.538016 0.498950 0.737800 1.6152
 min 12.000000 4.000000 71.0 52.0 130.0 ... 0.0 0.0 0.0 0.0
 25% 15.425000 4.000000 128.025000 95.500000 ... 0.0 0.0 0.0 0.0
 50% 19.200000 6.000000 156.000000 123.000000 ... 0.0 0.0 0.0 0.0
 75% 22.000000 8.000000 326.000000 180.000000 ... 1.000000 1.000000 4.000000 4.0000
 max 33.900000 8.000000 472.000000 355.000000 ... 1.000000 1.000000 5.000000 8.0000

 [8 rows x 11 columns]
 >>>

WSL at < bash * MEM: 9.72% | 1/4GB * 9ms
 15:29 | & testvim

```
a.R b.py mtcars.csv
WSL at < bash * MEM: 10.52% | 1/4GB * 11ms  

  15:27 | & testvim
```

WSL at < bash * MEM: 12.3% | 1/4GB * 9ms
 15:32 | & duckdb

```
-- Loading resources from /home/stli/.duckdbrc
v0.8.1 6556a72232
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
>>> SELECT * FROM read_csv_auto('stlipm/testvim/mtcars.csv') LIMIT 3;
column00
varchar
mpg
double
cyl
int64
disp
double
hp
int64
drat
double
wt
double
qsec
int64
vs
int64
am
int64
gear
int64
carb
int64
Mazda RX4
21.0
6
160
108
3.93
2.620
16.46
0
1
4
4
Mazda RX4 Wag
21.0
6
160
108
3.93
2.875
17.02
0
1
4
4
Datsun 710
22.8
4
108
93
3.85
2.320
18.61
1
1
4
1
Hornet 4 Drive
21.4
6
250
110
3.08
3.215
19.44
1
0
3
1
Hornet Sportabout
18.7
8
360
175
3.15
3.440
17.02
0
0
3
2
Valiant
18.1
6
225
105
2.76
3.460
20.22
1
0
3
1
```

Figure 1.6: R, Python, DuckDB

```
library(data.table)
# read(mtcars)
# Mazda RX4
# Mazda RX4 Wag
# Datsun 710
# Hornet 4 Drive
# Hornet Sportabout
# Valiant
mydf <- mean(mtcars$mpg) |> paste0()
print(mydf)
cat('Hello world from R')
head(mtcars)
```

WSL at < bash * MEM: 9.72% | 1/4GB * 9ms
 15:29 | & testvim

```
R
R version 4.3.1 (2023-06-16) -- "Beagle Scouts"
Copyright (C) 2023 The R Foundation for Statistical Computing
Platform: x86_64-pc-linux-gnu (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> head(mtcars)
mpg cyl disp hp drat wt qsec vs am gear carb
Mazda RX4 21.0 6 160 108 3.93 2.620 16.46 0 1 4 4
Mazda RX4 Wag 21.0 6 160 108 3.93 2.875 17.02 0 1 4 4
Datsun 710 22.8 4 108 93 3.85 2.320 18.61 1 1 4 1
Hornet 4 Drive 21.4 6 250 110 3.08 3.215 19.44 1 0 3 1
Hornet Sportabout 18.7 8 360 175 3.15 3.440 17.02 0 0 3 2
Valiant 18.1 6 225 105 2.76 3.460 20.22 1 0 3 1
```

Figure 1.7: Vim - R

A screenshot of the Vim editor interface. On the left is the code editor window titled 'testvim' containing Python code. The code defines a function 'myfunc' that prints 'Hello from me' five times, followed by a print statement for the pandas version. On the right is the terminal window showing the output: 'hello world from Python' and '2.8.2 is used now'. The status bar at the bottom indicates the file is 'python3 "b.py"' and has been finished.

```

1 import pandas as pd
2
3 def myfunc():
4     print('Hello from me')
5     print('Hello from me')
6     print('Hello from me')
7     print('Hello from me')
8     print('Hello from me')
9
10
11 print('Hello world from Python')
12 print(f'{pd.__version__} is used now')
13

```

Figure 1.8: Vim - Python

A screenshot of the RStudio interface. On the left is the code editor window titled 'Untitled2' containing R code that imports the 'mtcars' dataset from R, converts it to a pandas DataFrame in Python using reticulate, and then prints the DataFrame. On the right is the R console window showing the resulting DataFrame. The console output includes column names like mpg, cyl, disp, am, gear, and carb, along with their respective data values. The status bar at the bottom indicates the file is 'R Script'.

```

1 library(reticulate)
2 df_r1 <- mtcars
3
4 pd = import('pandas')
5 df_py1 = r_to_py(df_r1)
6 df_py1.dtypes
7 df_py1.describe()
8
9
10
11
12
13
14

```

	mpg	cyl	disp	am	gear	carb
count	32.000000	32.000000	32.000000	32.000000	32.000000	32.000000
min	20.09025	6.187500	236.721875	0.466250	4.67500	2.8125
q1	22.800000	6.580000	160.890000	0.400000	3.000000	3.000000
median	24.345000	7.0	200.617500	0.400000	3.000000	4.000000
q3	26.000000	8.0	283.500000	0.400000	3.000000	6.000000
max	33.900000	8.000000	472.000000	1.000000	4.000000	8.000000

Figure 1.9: RStudio - R

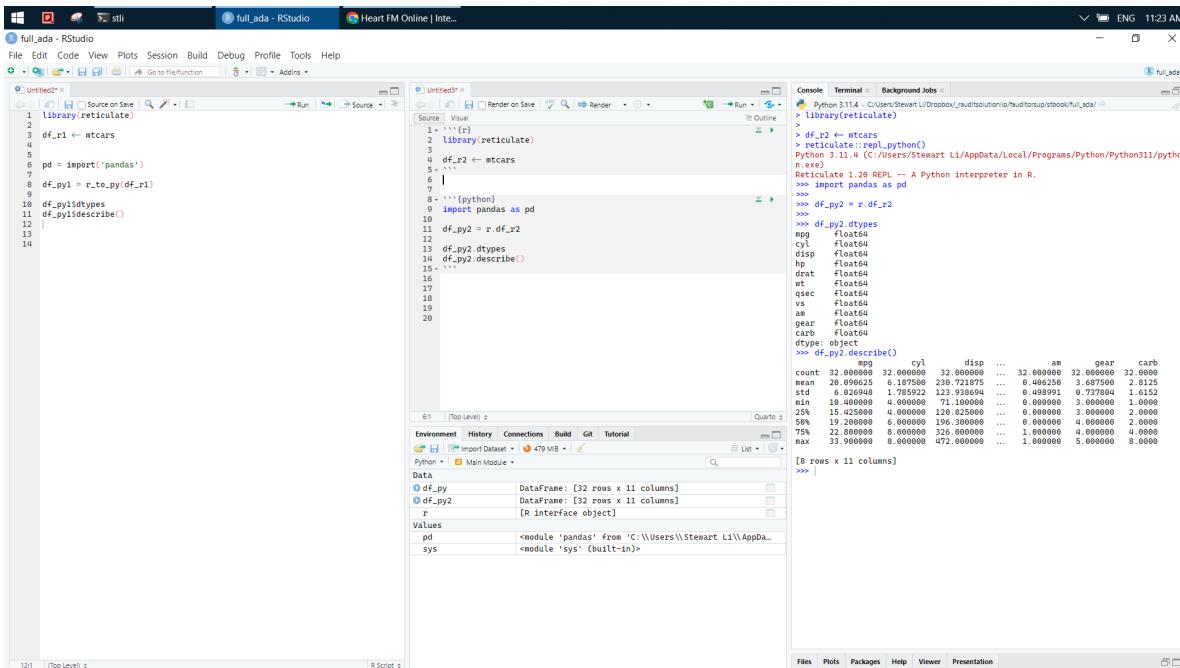


Figure 1.10: RStudio - Python

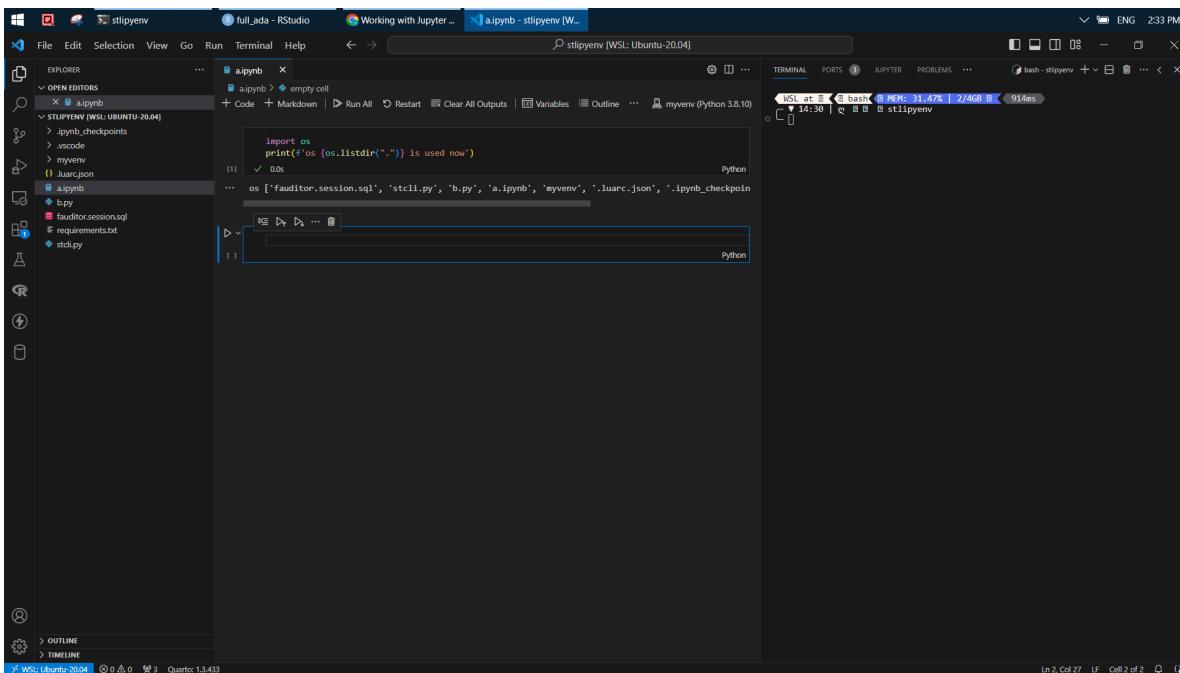


Figure 1.11: VS Code in Linux - Jupyter

A screenshot of the Visual Studio Code interface running on a Linux system. The title bar shows multiple tabs: 'full_ada - RStudio', 'Heart FM Online | Inte...', 'Interactive - b.py - stlipyenv', and 'stlipyenv [WSL: UBUNTU-20.04]'. The main area features a dark theme. On the left is the Explorer sidebar with a tree view of files and folders, including 'GROUP 1' and 'GROUP 2' sections, and a 'STLIPYENV (WSL: UBUNTU-20.04)' section containing 'jupyter_checkpoints', '.vscode', 'myenv', 'Juarcion', 'a.ipynb', 'b.ipynb', 'fauditor.session.sql', 'requirements.txt', and 'stdiply'. The center-left is the 'OPEN EDITORS' panel showing the file 'b.py' with code:

```
1 # %%\n2 import pandas as pd\n3\n4 print(f"pandas [{pd.__version__}] is used now")\n5\n6 # %%\n7 Run Cell | Run Above | Debug Cell
```

. To the right is the 'Interactive - b.py - stlipyenv' panel, which displays the output of the code execution:

```
Connected to myenv (Python 3.8.10)\n✓ import pandas as pd...\n... pandas 2.0.2 is used now\n\n1+1\n✓ 0.05\n...\n2
```

. At the bottom, there is a terminal window with the text 'Type "python" code here and press Shift+Enter to run'.

Figure 1.12: VS Code in Linux - Interactive cell

A screenshot of the Visual Studio Code interface running on Windows. The title bar shows multiple tabs: 'full_ada - RStudio', 'Heart FM Online | Inte...', 'b.py - testvim - Visual...', and 'testvim'. The main area features a dark theme. On the left is the Explorer sidebar with a tree view of files and folders, including 'TESTVIM' and 'a.R', 'b.py'. The center-left is the 'OPEN EDITORS' panel showing the file 'b.py' with code:

```
1 import os\n2\n3 print(f"os.listdir('.') is used now")\n4\n5 # %%\n6 import pandas as pd\n7 print(f"pandas [{pd.__version__}] is used now")\n8\n9 # %%\n10 Run Cell | Run Below | Debug Cell\n11 Run Cell | Run Above | Debug Cell
```

. To the right is the 'TERMINAL' panel, which shows the command line output:

```
Stewart: L1@DESKTOP-HCEU07A MINGW64 ~/Desktop/testvim\n$ python b.py\nos ['a.R', 'b.py'] is used now\npandas 2.0.3 is used now\n$
```

. At the bottom, there is a status bar with the text 'R (not attached) In 4, Col 1 Spaces:4 UTF-8 CR LF Python 3.11.4 64-bit Go Live Prettier'.

Figure 1.13: VS Code in Windows - Script

A screenshot of the Visual Studio Code interface on Windows. The window title is "full_ada - RStudio" and "Heart FM Online | Inte...". The left sidebar shows an "EXPLORER" view with files "a.R" and "b.py" under "TESTVIM". The main editor area has an "Interactive" tab open, showing the following Python code:

```
1 import os
2 print(f'os.listdir(".") is used now')
3
4 #%%%
5 import pandas as pd
6 print(f'pandas ({pd.__version__}) is used now')
```

The status bar at the bottom indicates "R (not attached) Ln3, Col1 Spaces:4 UTF-8 CRLF Python 3.11.4 Go Live Prettier".

Figure 1.14: VS Code in Windows - Interactive cell

A screenshot of the Visual Studio Code interface on Windows. The window title is "full_ada - RStudio" and "Heart FM Online | Inte...". The left sidebar shows an "EXPLORER" view with files "a.R" and "b.py" under "TESTVIM". The main editor area has an "Interactive" tab open, showing R code and output:

```
> head(mtcars)
#> #> library(dplyr)
#> #> head(mtcars)
#> #> plot(mtcars)
```

The output shows the first few rows of the mtcars dataset:

mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
21.0	6	160	110	3.98	2.620	16.46	0	1	4	4
Mazda RX4										
21.0	6	160	110	3.98	2.875	17.02	0	1	4	4
Mazda RX4 Wag										
21.0	6	160	110	3.98	2.432	17.82	0	1	4	4
Datsun 710										
21.4	6	258	110	3.08	3.215	19.44	1	0	3	2
Hornet 4 Drive										
21.4	6	258	110	3.08	3.215	19.44	1	0	3	2
Hornet Sportabout										
18.7	8	360	175	3.15	3.440	17.02	0	0	3	2
Honda Civic										
18.1	6	225	105	2.76	3.460	20.22	1	0	3	1
Valiant										
> plot(mtcars)										
>										

The status bar at the bottom indicates "R (not attached) Ln8, Col1 Spaces:4 UTF-8 CRLF R Go Live Prettier". To the right of the terminal, there is a "R Graphics: Device 2 (ACTIVE)" window showing a correlation matrix plot for the mtcars dataset.

Figure 1.15: VS Code in Windows - R

It is vital to create a proper **folder** structure along with config file as you are able to move quickly and organize your scripts better. I run a command line tool (written in R) from GitBash and PowerShell to do it.

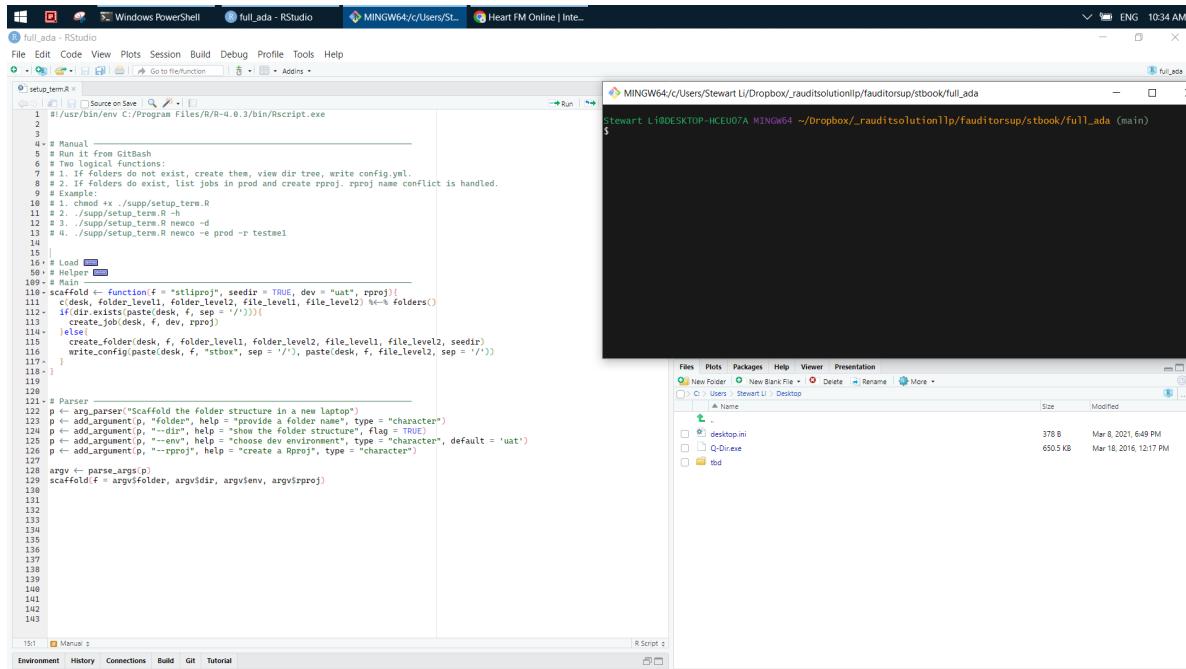


Figure 1.16: CLI R - GitBash 1

The screenshot shows the RStudio interface with the 'full_ada' project open. The left pane displays the `setup.Term.R` script, which is a shell script for setting up a folder structure. The right pane shows a terminal window running under GitBash, displaying the command-line usage of the script.

```

#!/usr/bin/env C:/Program Files/R/R-4.0.3/bin/Rscript.exe
# Manual
# Run it from GitBash
# Two logical functions:
# 1. If folders do not exist, create them, view dir tree, write config.yml.
# 2. If folders do exist, list jobs in prod and create rproj. rproj name conflict is handled.
# Examples:
# 1. chmod +x ./supp/setup.Term.R
# 2. ./supp/setup.Term.R -h
# 3. ./supp/setup.Term.R newco -d
# 4. ./supp/setup.Term.R newco -e prod -r testmle
# Load
# Helper
# Main
scatfold <- function(f = "stlproj", seedir = TRUE, dev = "uat", rproj){
  cdesk, folder_level1, folder_level2, file_level1, file_level2, sep = "%-%"
  if(dir.exists(paste(desk, f, sep = '/'))){
    create.folder(desk, f, dev, rproj)
  }
  else{
    create.folder(desk, f, folder_level1, folder_level2, file_level1, file_level2, seedir)
    write.config(paste(desk, f, "stbox", sep = '/'), paste(desk, f, file_level2, sep = '/'))
  }
}
# Parser
args <- parse_args()
scatfold(f = argv$folder, argv$dev, argv$env, argv$rproj)
# Manual

```

Terminal Output:

```

Stewart Li@DESKTOP-HCEU07A MINGW64 ~/Dropbox/_rauditsolution1lp/fauditorsup/stbook/full_ada (main)
$ ./supp/setup.Term.R -h
usage: setup.Term.R [-] [--help] [--dir] [--opts OPTS] [--env ENV]
                     [--rproj RPROJ] folder
Scaffold the folder structure in a new laptop
positional arguments:
  folder   provide a folder name
optional arguments:
  -h, --help  show this help message and exit
  -d, --dir   show the folder structure
  --opts    file containing argument values
  --env    choose dev environment [default: uat]
  --rproj   create a Rproj
$ 
```

Figure 1.17: CLI R - GitBash 2

This screenshot is similar to Figure 1.17, showing the RStudio interface and terminal window. The terminal output shows a different folder structure being scaffolded, specifically a 'newco' folder structure.

```

#!/usr/bin/env C:/Program Files/R/R-4.0.3/bin/Rscript.exe
# Manual
# Run it from GitBash
# Two logical functions:
# 1. If folders do not exist, create them, view dir tree, write config.yml.
# 2. If folders do exist, list jobs in prod and create rproj. rproj name conflict is handled.
# Examples:
# 1. chmod +x ./supp/setup.Term.R
# 2. ./supp/setup.Term.R -h
# 3. ./supp/setup.Term.R newco -d
# 4. ./supp/setup.Term.R newco -e prod -r testmle
# Load
# Helper
# Main
scatfold <- function(f = "stlproj", seedir = TRUE, dev = "uat", rproj){
  cdesk, folder_level1, folder_level2, file_level1, file_level2, sep = "%-%"
  if(dir.exists(paste(desk, f, sep = '/'))){
    create.folder(desk, f, dev, rproj)
  }
  else{
    create.folder(desk, f, folder_level1, folder_level2, file_level1, file_level2, seedir)
    write.config(paste(desk, f, "stbox", sep = '/'), paste(desk, f, file_level2, sep = '/'))
  }
}
# Parser
args <- parse_args()
scatfold(f = argv$folder, argv$dev, argv$env, argv$rproj)
# Manual

```

Terminal Output:

```

Stewart Li@DESKTOP-HCEU07A MINGW64 ~/Dropbox/_rauditsolution1lp/fauditorsup/stbook/full_ada (main)
$ ./supp/setup.Term.R newco -d
C:/Users/Stewart Li/Desktop/newco
  +- proj
    +- config.yml
    +- data
    +- pro
    +- res
    +- README.qmd
  +- stbox
  +- stpkg
  +- tbd
  +- uat
  \-- proj
$ 
```

Figure 1.18: CLI R - GitBash 3

The screenshot shows the RStudio interface with two panes. The left pane displays the `setup.Term.R` script, which is a series of R code for managing folder structures and environments. The right pane shows the terminal window running under GitBash, where the script is executed. The terminal output indicates that a file named `testmel` already exists, prompting the user to choose another name. A file browser window is also visible at the bottom, showing files like `ignore`, `R`, and `testme1.Rproj`.

```

#!/usr/bin/env C:/Program Files/R/R-4.0.3/bin/Rscript.exe
# Manual
# Run it from GitBash
# Logical functions:
# 1. If folders do not exist, create them, view dir tree, write config.yml.
# 2. If folders do exist, list jobs in prod and create rproj. rproj name conflict is handled.
# Examples:
# 1. chmod +x ./supp/setup.Term.R
# 2. ./supp/setup.Term.R -h
# 3. ./supp/setup.Term.R newco -d
# 4. ./supp/setup.Term.R newco -e prod -r testmel
# Load
# Helper
# Main
scraftold <- function(f = "stlproj", seedir = TRUE, dev = "uat", rproj) {
  cdesk, folder_level1, folder_level2, file_level1, file_level2, %<-% folders()
  if(dir.exists(paste(desk, f, sep = '/'))) {
    create.folder(desk, f, dev, rproj)
  } else {
    create.folder(desk, f, folder_level1, folder_level2, file.level1, file.level2, seedir)
    write.config(paste(desk, f, "stbox", sep = '/'), paste(desk, f, file.level2, sep = '/'))
  }
}
# Parser
p <- arg_parser("Scaffold the folder structure in a new laptop")
p <- add_argument(p, "folder", help = "provide a folder name", type = "character")
p <- add_argument(p, "dev", help = "choose dev environment", flag = TRUE)
p <- add_argument(p, "--env", help = "choose dev environment", type = "character", default = "uat")
p <- add_argument(p, "--rproj", help = "create a Rproj", type = "character")
# Arguments
args <- parse_args(p)
scraftold = args$folder, arg$dev, arg$env, arg$rproj

```

Figure 1.19: CLI R - GitBash 4

This screenshot shows the same RStudio setup as Figure 1.19, but the terminal output is different. It shows the command `./supp/setup.Term.R newco -e prod -r testmel` being run, and the terminal responds with "testmel exists. Please choose another name." A file browser window at the bottom shows files `ignore`, `R`, and `testme1.Rproj`.

```

#!/usr/bin/env C:/Program Files/R/R-4.0.3/bin/Rscript.exe
# Manual
# Run it from GitBash
# Logical functions:
# 1. If folders do not exist, create them, view dir tree, write config.yml.
# 2. If folders do exist, list jobs in prod and create rproj. rproj name conflict is handled.
# Examples:
# 1. chmod +x ./supp/setup.Term.R
# 2. ./supp/setup.Term.R -h
# 3. ./supp/setup.Term.R newco -d
# 4. ./supp/setup.Term.R newco -e prod -r testmel
# Load
# Helper
# Main
scraftold <- function(f = "stlproj", seedir = TRUE, dev = "uat", rproj) {
  cdesk, folder_level1, folder_level2, file_level1, file_level2, %<-% folders()
  if(dir.exists(paste(desk, f, sep = '/'))) {
    create.folder(desk, f, dev, rproj)
  } else {
    create.folder(desk, f, folder_level1, folder_level2, file.level1, file.level2, seedir)
    write.config(paste(desk, f, "stbox", sep = '/'), paste(desk, f, file.level2, sep = '/'))
  }
}
# Parser
p <- arg_parser("Scaffold the folder structure in a new laptop")
p <- add_argument(p, "folder", help = "provide a folder name", type = "character")
p <- add_argument(p, "--dev", help = "show the folder structure", flag = TRUE)
p <- add_argument(p, "--env", help = "choose dev environment", type = "character", default = "uat")
p <- add_argument(p, "--rproj", help = "create a Rproj", type = "character")
# Arguments
args <- parse_args(p)
scraftold = args$folder, arg$dev, arg$env, arg$rproj

```

Figure 1.20: CLI R - GitBash 5

The screenshot shows an RStudio interface with a code editor containing R script code. The code defines a function `stlproj` that handles command-line arguments for creating a folder structure. It includes logic for handling subfolders, creating jobs, and writing configuration files. Below the code editor is a terminal window showing the command `PS C:\Users\Stewart Li\Desktop\full_ada\supp> stlproj` being run. To the right of the terminal is a file explorer window showing the directory structure created by the script, including subfolders like `prod`, `data`, and `raw`.

```

#> stlproj <- function(f = "stlproj", seedir = TRUE, dev = "uat", rproj) {
  # Manual
  # Run it from Gitbash
  # Two logical functions:
  # 1. If folders do not exist, create them, view dir tree, write config.yml.
  # 2. If folders do exist, list jobs in prod and create rproj. rproj name conflict is handled.
  # Examples:
  # 1. chmod +x ./supp/setup_term.R
  # 2. ./supp/setup_term.R -h
  # 3. ./supp/setup_term.R newco -d
  # 4. ./supp/setup_term.R newco -e prod -r testml
  # Load
  # Helper
  # Main
  scaffold <- function(f = "stlproj", seedir = TRUE, dev = "uat", rproj) {
    cdesk, folder_level1, folder_level2, file_level1, file_level2, sep = "%-%"
    if(dir.exists(paste(desk, f, sep = '/'))){
      create_job(desk, f, dev, rproj)
    } else {
      create_folder(desk, f, folder_level1, folder_level2, file_level1, file_level2, seedir)
      write_config(paste(desk, f, "stbox", sep = '/'), paste(desk, f, file_level2, sep = '/'))
    }
  }
  # Parser
  p <- arg_parser("Scaffold the folder structure in a new laptop")
  p <- add_argument(p, "folder", help = "provide a folder name", type = "character")
  p <- add_argument(p, "dev", help = "choose the folder structure", flag = TRUE)
  p <- add_argument(p, "--env", help = "choose dev environment", type = "character", default = "uat")
  p <- add_argument(p, "--rproj", help = "create a Rproj", type = "character")
  # Load args
  scaffold(f = argv$folder, argv$dev, argv$env, argv$rproj)
}

arg <- parse_args(p)
scaffold(f = argv$folder, argv$dev, argv$env, argv$rproj)

```

Figure 1.21: CLI R - PowerShell 1

This screenshot is similar to Figure 1.21, showing the RStudio interface and terminal window. The terminal window now shows the command `PS C:\Users\Stewart Li\Desktop\full_ada\supp> ./run_setup.bat newco -d` being run. The file explorer window on the right shows the newly created directory structure under the `newco` folder, including subfolders like `prod`, `data`, `raw`, `stbin`, `stbox`, `stpkgs`, `tbd`, and `uat`.

```

#> stlproj <- function(f = "stlproj", seedir = TRUE, dev = "uat", rproj) {
  # Manual
  # Run it from Gitbash
  # Two logical functions:
  # 1. If folders do not exist, create them, view dir tree, write config.yml.
  # 2. If folders do exist, list jobs in prod and create rproj. rproj name conflict is handled.
  # Examples:
  # 1. chmod +x ./supp/setup_term.R
  # 2. ./supp/setup_term.R -h
  # 3. ./supp/setup_term.R newco -d
  # 4. ./supp/setup_term.R newco -e prod -r testml
  # Load
  # Helper
  # Main
  scaffold <- function(f = "stlproj", seedir = TRUE, dev = "uat", rproj) {
    cdesk, folder_level1, folder_level2, file_level1, file_level2, sep = "%-%"
    if(dir.exists(paste(desk, f, sep = '/'))){
      create_job(desk, f, dev, rproj)
    } else {
      create_folder(desk, f, folder_level1, folder_level2, file_level1, file_level2, seedir)
      write_config(paste(desk, f, "stbox", sep = '/'), paste(desk, f, file_level2, sep = '/'))
    }
  }
  # Parser
  p <- arg_parser("Scaffold the folder structure in a new laptop")
  p <- add_argument(p, "folder", help = "provide a folder name", type = "character")
  p <- add_argument(p, "--dir", help = "show the folder structure", flag = TRUE)
  p <- add_argument(p, "--env", help = "choose dev environment", type = "character", default = "uat")
  p <- add_argument(p, "--rproj", help = "create a Rproj", type = "character")
  # Load args
  scaffold(f = argv$folder, argv$dev, argv$env, argv$rproj)
}

arg <- parse_args(p)
scaffold(f = argv$folder, argv$dev, argv$env, argv$rproj)

```

Figure 1.22: CLI R - PowerShell 2

2 ELT

Consider the following examples to establish a data pipeline.

1. A zip file lands in data lake (`s3/minio`) daily.
2. Execute scripts in the server (`ec2`) to download/unzip/select/upload files based on `mtime`. It produces a file (`csv`) to track work done at the agreed cut-off time (`cron`). AWS `lambda` is another option.
3. `snowflake` external stage (`s3`) is triggered by a file (`txt`) to kicks off `snowpipe` and ingest data to DB as `variant`. Similar storage are `databrick`, `dremio`, `clickhouse`. The preferred formats are `parquet`, `iceberg`, `ADBC`.
4. Move data between platforms via `airbyte`.
5. Validate and transform DB raw to DB mart through `dbt`.
6. Automatize the process by a task scheduler `prefect`, `airflow`, `dagster`.
7. Create a dashboard for DB mart via `metabase`, `superset`.

Part II

Data tools

SQL, R, Python, Julia, Rust, and JavaScript can be used interchangeably to perform data work at most of the time. Choose programming languages and relevant packages based on your needs and personal preference.

Assess your IO scenario after considered the followings.

How big is data?

1. Memory:

- `datatable`, `collapse`, `duckdb`, `polars`,
- `ibis`, `DataFusion`, `deltalake`,

2. Hard disk:

- `arrow`,

3. Cluster:

- `spark`, `dask`,

Where data lives?

1. DB:

- `DBI`, `odbc`, `SQLAlchemy`, `connectorx`, `sqlx`,

2. SFTP:

- `RCurl`, `paramiko`,

3. Blob:

- `pins`, `aws.s3`, `s3fs`, `boto3`,

In what form? The preferred file types are `txt`, `csv`, `parquet`, `feather`.

1. Excel:

- `tidyxl`, `unpivotr`, `openxlsx`, `openpyxl`,

2. Word:

- `officer`, `docx`,

3. PPT:

- `officer`, `python-pptx`,

4. PDF:

- `pdftools`, `PDFminer`, `PyPDF2`, `pdfplumber`,

5. SAS:

- `haven`,

6. Image:

- `magick`, `tesseract`, `pillow`, `cv2`,

7. Geo:

- `sf`, `countrycode`,

8. API:

- `httr2`, `request`, `reqwest`,

- `jsonlite`, `yaml`, `toml`,

9. Website:

- `html`, `xml`, `rvest`, `bs4`,

- `v8`, `chromote`, `selenium`, `playwright`,

In what data structure and type?

1. Data type:

- numeric, string, bool, factor, date,

2. Data collection:

- list, vector, data.frame (cell/0 row/1 column),

3. Verb:

- count/sort/select/filter/mutate/summarize/pivot/join,

Analysis work is to produce meaningful insight via slice dice. Classify a set of tools based on the following analytics steps. To reduce repetitive work, you can create functions, OOP, box, package, and cli.

1. Interact with DB:

- dbplyr, dbplot, dbcooper,

2. Data cleaning:

- base, tidyverse, pandas,

- janitor, glue, tidylog,

- waldo, diffobj, compareDF,

3. Data validation:

- pointblank, validate, greate expectation, pydantic,

4. Data visualization¹:

- grid, patchwork, ggfx, ggtext, showtext,

- ragg, scales, formattable, sparkline,

- gghighlight, ggforce,

- imager, imagerExtra, ggimage, ggpibr,

- igraph, ggraph, tidygraph, networkD3, visNetwork,

- DiagrammeR, UpSetR, tmap,

5. Table:

- gt, gtExtras, gtsummary, modelsummary,

- flextable, kableExtra,

6. EDA:

- skimr, naniar, visdat, inspectdf,

7. Stats:

- corrplot, tidylo, widyr, broom,

8. Report:

- quarto, whisker, target, jinja2,

9. API deploy:

- vetiver, plumber, fastapi,

10. Dashboard:

- shiny, htmltools, htmlwidgets, crosstalk, leaflet,

- bslib, thematic, sass,

- DT, reactable, reactablefmtr,

- plotly, echarts4r, bokeh,

¹ggplot2 (Wickham 2016)

- `dash`, `streamlit`,
- 11. WASM:
 - `webr`, `pyodide`, `wasm_bindgen`,
- 12. GUI:
 - [PyAutoGUI](#),
 - `Tkinter`, [PyQt5](#),

Consider other utility tools when necessary.

1. Environment:

- `rvenv`, `venv`,

2. Helper:

- `cli`, `crayon`,

- `clipr`, `withr`, `callr`, `pingr`, `curl`,

3. Email:

- `blastula`, `emayili`, `smtplib`, `pywin32`,

4. Unzip:

- `archive`, `zipfile`,

5. FFI:

- `rlang`, `vctrs`, `lobstr`, `S7`,

- `cpp11`, `Rcpp`, `extindr`, `pyo3`, `bindgen`,

3 Polars

Command line tools allow you to do those repetitive data work easily. The following three examples are.

1. argparse and duckdb.
2. click and polars.
3. clap and polars.

The screenshot shows a Windows desktop environment with a terminal window open. The terminal window title is "stcli.py - stlipyenv [W...]" and the subtitle bar shows "Heart FM Online | Inte...". The terminal content displays a Python script named "stcli.py" and its execution output. The script uses argparse to handle command-line arguments, connects to a DuckDB database, and performs a query to filter rows where "cogs > 200000". The output shows the resulting DataFrame with columns: segment, country, product, cogs, profit, and date timestamp. The data includes rows for Government, Canada, Carretera, and Government, Germany, Carretera, among others. The terminal also shows the command "stcli.py --filter" being run again.

```
stcli.py
1 import argparse
2 import duckdb
3
4
5 def st_filter(db, tbl, ex="cogs > 200000"):
6     conn = duckdb.connect(db)
7     df = conn.execute(f"select * from {tbl}").df()
8     df_res = duckdb.filter(df, ex)
9     conn.close()
10    print(df_res)
11
12
13 def st_summarize(db, tbl):
14     conn = duckdb.connect(db)
15     df = conn.execute(f"select * from {tbl}").df()
16     df_res = duckdb.sql(
17         """select segment, country, count(*) as n from df group by all order by n desc"""
18     )
19     conn.close()
20     print(df_res)
21
22
23 if __name__ == "__main__":
24     parser = argparse.ArgumentParser()
25     # namespace db contains multiple args
26     parser.add_argument("db", type=str, nargs="+")
27     # optional function
28     parser.add_argument(
29         "--filter",
30         dest="do",
31         action="store_const",
32         const=st_filter,
33         default=st_summarize,
34     )
35     args = parser.parse_args()
36     # st_filter(*args[0])
37     args.do(*args.db)
38
39
40 # python3 stcli.py ~/finsample.duckdb finsample
41 # python3 stcli.py ~/finsample.duckdb finsample --filter
42
WSL:at: 25.77% | 1/4GB | 1m 22s 746ms
./duckdb finsample.duckdb
-- Loading resources from /home/stli/.duckdbrc
v0.8.1 6536a7732
Enter "help" for usage hints.
> .table
finsample
> .from finsample limit 3;
+-----+-----+-----+-----+-----+-----+
| segment | country | product | cogs | profit | date timestamp |
+-----+-----+-----+-----+-----+-----+
| Government | Canada | Carretera | 36185.0 | 16185.0 | 2014-01-01 00:00:00 |
| Government | Germany | Carretera | 13210.0 | 13210.0 | 2014-01-01 00:00:00 |
| Midmarket | France | Carretera | 21780.0 | 10890.0 | 2014-06-01 00:00:00 |
+-----+-----+-----+-----+-----+-----+
3 rows
13 columns (6 shown)
WSL:at: 26.64% | 1/4GB | 1m 15s 919ms
0 88.45% |
```

Figure 3.1: CLI - argparse 1

```

stcli.py
1 import argparse
2 import duckdb
3
4 def st_filter(db, tbl, ex="cogs > 200000"):
5     conn = duckdb.connect(db)
6     df = conn.execute(f"select * from {tbl}").df()
7     df_res = duckdb.filter(df, ex)
8     conn.close()
9     print(df_res)
10
11
12 def st_summarize(db, tbl):
13     conn = duckdb.connect(db)
14     df = conn.execute(f"select * from {tbl}").df()
15     df_res = duckdb.sql(
16         """select segment, country, count(*) as n from df group by all order by n desc"""
17     )
18     conn.close()
19     print(df_res)
20
21
22 if __name__ == "__main__":
23     parser = argparse.ArgumentParser()
24     # namespace db contains multiple args
25     parser.add_argument("db", type=str, nargs="+")
26     # optional function
27     parser.add_argument(
28         "--filter",
29         dest="do",
30         action="store_const",
31         const=st_filter,
32         default=st_summarize,
33     )
34
35     args = parser.parse_args()
36     # st_filter(*args.db)
37     args.do(*args.db)
38
39
40 # python3 stcli.py ~/finsample.duckdb finsample
41 # python3 stcli.py ~/finsample.duckdb finsample --filter
42

```

Running in Ubuntu-20.04 (WSL 2)

In 42, Col 1 Spaces:4 UFF:8 If Python 3.8.10 (myvenv:venv) □

Figure 3.2: CLI - argparse 2

```

stcli.py
1 import argparse
2 import duckdb
3
4 def st_filter(db, tbl, ex="cogs > 200000"):
5     conn = duckdb.connect(db)
6     df = conn.execute(f"select * from {tbl}").df()
7     df_res = duckdb.filter(df, ex)
8     conn.close()
9     print(df_res)
10
11
12 def st_summarize(db, tbl):
13     conn = duckdb.connect(db)
14     df = conn.execute(f"select * from {tbl}").df()
15     df_res = duckdb.sql(
16         """select segment, country, count(*) as n from df group by all order by n desc"""
17     )
18     conn.close()
19     print(df_res)
20
21
22 if __name__ == "__main__":
23     parser = argparse.ArgumentParser()
24     # namespace db contains multiple args
25     parser.add_argument("db", type=str, nargs="+")
26     # optional function
27     parser.add_argument(
28         "--filter",
29         dest="do",
30         action="store_const",
31         const=st_filter,
32         default=st_summarize,
33     )
34
35     args = parser.parse_args()
36     # st_filter(*args.db)
37     args.do(*args.db)
38
39
40 # python3 stcli.py ~/finsample.duckdb finsample
41 # python3 stcli.py ~/finsample.duckdb finsample --filter
42

```

Running in Ubuntu-20.04 (WSL 2)

In 42, Col 1 Spaces:4 UFF:8 If Python 3.8.10 (myvenv:venv) □

Figure 3.3: CLI - argparse 3

```

stclick > stclick > stclick.py > ...
59 @stat.command()
60 @click.pass_context
61 @click.argument('col', type=str)
62 @click.argument('n', type=int)
63 def topn(ctx, col, n):
64     res = ctx.obj.sort(pl.col(f'{col}'), descending=True).limit(n)
65     click.echo(res)

66 @main.group()
67 > def calc():
68     ...

69     @click.command()
70     @click.pass_context
71     @click.argument("output", type=click.File("w"), default="-", required=False)
72     @click.argument("highlight", type=click.Choice(["red", "green"]),
73                    help="highlight data based on the provided threshold",
74                )
75     def main_group():
76         ...

77     @click.command()
78     @click.argument("output", type=click.File("w"), default="-", required=False)
79     @click.argument("highlight", type=click.Choice(["red", "green"]),
80                    help="highlight data based on the provided threshold",
81                )
82     def condc(ctx, output, highlight):
83         ...

84     def cond(ctx, output, highlight):
85         ...

86     choose your data
87     ...
88     res = ctx.obj.with_columns(
89         new(
90             pl.when(pl.col("hp") > 200
91                   .then(pl.col("mpg").filter(pl.col("hp") > 200).sum())
92                   .otherwise(pl.col("mpg").filter(pl.col("hp") < 200).sum())
93             )
94             .round(2)
95             .cast(pl.Utf8)
96         )
97         if highlight == "red":
98             res = res.with_columns(pl.col("new").map(lambda x: f"\033[91m{x} + \x1b[0m"))
99         else:
100             res = res.with_columns(pl.col("new").map(lambda x: f"\033[96m{x} + \x1b[0m"))
101
102     # click.echo(output)
103     click.echo(res)

104     if __name__ == "__main__":
105         main()

106     if __name__ == "__main__":
107         main()

108

```

Figure 3.4: CLI - click 1

```

stclick > stclick > stclick.py > ...
59 @stat.command()
60 @click.pass_context
61 @click.argument('col', type=str)
62 @click.argument('n', type=int)
63 def topn(ctx, col, n):
64     res = ctx.obj.sort(pl.col(f'{col}'), descending=True).limit(n)
65     click.echo(res)

66 @main.group()
67 > def calc():
68     ...

69     @click.command()
70     @click.pass_context
71     @click.argument("output", type=click.File("w"), default="-", required=False)
72     @click.argument("highlight", type=click.Choice(["red", "green"]),
73                    help="highlight data based on the provided threshold",
74                )
75     def main_group():
76         ...

77     @click.command()
78     @click.argument("output", type=click.File("w"), default="-", required=False)
79     @click.argument("highlight", type=click.Choice(["red", "green"]),
80                    help="highlight data based on the provided threshold",
81                )
82     def condc(ctx, output, highlight):
83         ...

84     def cond(ctx, output, highlight):
85         ...

86     choose your data
87     ...
88     res = ctx.obj.with_columns(
89         new(
90             pl.when(pl.col("hp") > 200
91                   .then(pl.col("mpg").filter(pl.col("hp") > 200).sum())
92                   .otherwise(pl.col("mpg").filter(pl.col("hp") < 200).sum())
93             )
94             .round(2)
95             .cast(pl.Utf8)
96         )
97         if highlight == "red":
98             res = res.with_columns(pl.col("new").map(lambda x: f"\033[91m{x} + \x1b[0m"))
99         else:
100             res = res.with_columns(pl.col("new").map(lambda x: f"\033[96m{x} + \x1b[0m"))
101
102     # click.echo(output)
103     click.echo(res)

104     if __name__ == "__main__":
105         main()

106     if __name__ == "__main__":
107         main()

108

```

Figure 3.5: CLI - click 2

```

WSL at stli [M: 27.0% | 2/4GB B] 249ms
python3 ./stclick/stclick.py "/home/stli/stliproj/testv1m/mtcars.csv" calc condc
data source: /home/stli/stliproj/testv1m/mtcars.csv
shape: (32, 13)
  mpg cyl disp  am gear carb new
  ...
str   f64 164 f64  i64 164 i64 str
Mazda RX4 21.0 6 160.0 - 1 4 4 549.0
Mazda RX4 Wag 21.0 6 160.0 - 1 4 4 549.0
Datsun 710 22.8 4 108.0 - 1 4 1 549.0
Hornet 4 Drive 21.4 6 258.0 - 0 3 1 549.0
...
Ford Pantera L 15.8 8 351.0 - 1 5 4 93.9
Ferrari Dino 19.7 6 145.0 - 1 5 6 549.0
Maserati Bora 15.0 8 301.0 - 1 5 8 93.9
Volvo 142E 21.4 4 121.0 - 1 4 2 549.0

WSL at stli [M: 26.9% | 1/4GB B] 189ms
python3 ./stclick/stclick.py "/home/stli/stliproj/testv1m/mtcars.csv" calc condc | grep Volvo
Volvo 142E 21.4 4 121.0 - 1 4 2 549.0

WSL at stli [M: 27.0% | 2/4GB B] 174ms
python3 ./stclick/stclick.py "/home/stli/stliproj/testv1m/mtcars.csv" calc condc | grep V
o

```

Figure 3.6: CLI - click 3

```

WSL at stli [M: 33.7% | 2/4GB B] 281ms
stpolars "/home/stli/stliproj/testv1m/mtcars.csv" about
data source: /home/stli/stliproj/testv1m/mtcars.csv
shape: (32, 13)

WSL at stli [M: 33.0% | 2/4GB B] 214ms
stpolars "/home/stli/stliproj/testv1m/mtcars.csv" calc condc
data source: /home/stli/stliproj/testv1m/mtcars.csv
shape: (32, 13)
  mpg cyl disp  am gear carb new
  ...
str   f64 164 f64  i64 164 i64 str
Mazda RX4 21.0 6 160.0 - 1 4 4 549.0
Mazda RX4 Wag 21.0 6 160.0 - 1 4 4 549.0
Datsun 710 22.8 4 108.0 - 1 4 1 549.0
Hornet 4 Drive 21.4 6 258.0 - 0 3 1 549.0
...
Ford Pantera L 15.8 8 351.0 - 1 5 4 93.9
Ferrari Dino 19.7 6 145.0 - 1 5 6 549.0
Maserati Bora 15.0 8 301.0 - 1 5 8 93.9
Volvo 142E 21.4 4 121.0 - 1 4 2 549.0

WSL at stli [M: 32.5% | 2/4GB B] 214ms

```

Figure 3.7: CLI - click 4

The screenshot shows a terminal window titled 'testtomi [WSL: Ubuntu-20.04]' with the following content:

```

fauditor# select * from client;
name | year | joblist | auditor | status
clientA | 2023 | clientA_2023 | stewartli | f
(1 row)

fauditor# 

```

Below the terminal, the output of a Rust build is shown:

```

Compiling home v0.5.5
Compiling hdk v0.12.3
Compiling stdcrt v0.38.20
Compiling md-5 v0.10.6
Compiling sha1 v0.10.6
Compiling md5-simd v0.1.14
Compiling sqlx-core v0.7.2
Compiling sqlx v0.7.2
Compiling tempfile v3.8.0
Compiling uuid v1.5.0
Compiling tokio-util v0.6.3
Compiling tracemac v0.6.3
Compiling serde_yaml v0.9.25
Compiling toml_edit v0.20.2
Compiling tokio-postgres v0.7.2
Compiling sqlx-core v0.7.2
Compiling toml v0.8.2
Compiling clap v4.4.6
Compiling futures-v0.7.2
Compiling sqlx v0.7.2
Compiling testtomi v0.1.0 (/home/stili/stiliproj/trystrust/testtomi)
Finished dev [unoptimized + debuginfo] target(s) in 1m 27s 630ms
Running "target/debug/testtomi" -n clientA -y 2023 -a stewartli init"
Initialization in /home/stili/stiliproj/trystrust/testtomi

```

The bottom status bar indicates the terminal is not attached.

Figure 3.8: CLI - clap 1

The screenshot shows a terminal window titled 'testtomi [WSL: Ubuntu-20.04]' with the following content:

```

WSL at 2023-07-17 16:45:38 [WSL:Ubuntu-20.04] master w/ 24
cargo run -- -n clientA -y 2023 -a stewartli init
Initialization in /home/stili/stiliproj/trystrust/testtomi

```

Below the terminal, the output of a help command is shown:

```

Usage: testtomi [OPTIONS] <NAME> <NAME> --year <YEAR> --auditor <AUDITOR> [COMMAND]
Commands:
  init   Initialize project root
  new    Start a new job
  plan   Plan a new analysis action
  help   Print this message or on the help of the given subcommand(s)
Options:
  -n <NAME> <NAME>      Provide a client name
  -y, --year <YEAR>       Provide a job year
  -a, --auditor <AUDITOR> Provide an auditor name
  -s, --status             Sets job status
  -h, --help               Print help
  -V, --version            Print version
Please donate

```

The bottom status bar indicates the terminal is not attached.

Figure 3.9: CLI - clap 2

4 Analysis

Factored Accounts Receivable - The biggest challenge of Factoring is to predict if and when invoices will be paid. The factor provides funds against this future payment to the business by buying their invoice. The factor then collects the payment and charges their interest rate. If the invoice isn't paid, the factor loses their advanced funds. Try using this data set for predicting when payments will be made. Get the data [here](#).

4.1 IO

```
df_raw <- read_csv(here::here('data/factor_ar.csv')) %>%
  janitor::clean_names()

glimpse(df_raw)
```

`data.table` is the fastest IO tool if your data can fit in the memory.

```
library(data.table)

# read in
data.table::fread("grep -v '770' ./data/factor_ar.csv")[, .N, by = countryCode]

# write out
df_dt <- as.data.table(df_raw)

df_dt[, 
       fwrite(data.table(.SD),
              paste0("C:/Users/Stewart Li/Desktop/res/",
                     paste0(country_code, ".csv"))), by = country_code]

# read in
data.table(
  country_code.csv = Sys.glob("C:/Users/Stewart Li/Desktop/res/*.csv")
)[, fread(country_code.csv), by = country_code.csv]
```

Get to know your data. For instance, any missing value, counting variables, and others.

```
# no NA
sapply(df_raw, function(x) {sum(is.na(x)) / nrow(df_raw)}) %>%
  enframe() %>%
  mutate(value = formattable::percent(value))

naniar::gg_miss_var(df_raw)
naniar::vis_miss(df_raw)

# no duplicate
df_raw %>% count(invoice_number, sort = TRUE)

# overview of data
skimr::skim(df_raw)
```

4.2 Cleaning

After having a basic understanding about data, do the followings to clean it up.

1. cast data types.
2. 30 days credit term is allowed. drop it subsequently (constant).
3. drop column (paperless_date).
4. rename and rearrange columns.

```
df_clean <- df_raw %>%
  mutate(across(contains("date"), lubridate::mdy),
        across(c(country_code, invoice_number), as.character)) %>%
  mutate(credit = as.numeric(due_date - invoice_date)) %>%
  select(c(country_code, customer_id, paperless_bill, disputed,
           invoice_number, invoice_amount, invoice_date, due_date, settled_date,
           settle = days_to_settle, late = days_late))

setdiff(colnames(df_raw), colnames(df_clean))
```

4.3 Validate

Validate data if it is received from other team members.

```

# data type
df_clean %>%
  select(contains("date")) %>%
  pointblank::col_is_date(columns = everything())

# cross checking
df_clean %>%
  mutate(settle1 = as.numeric(settled_date - invoice_date),
         late1 = as.numeric(settled_date - due_date),
         late1 = if_else(late1 < 0, 0, late1)) %>%
  summarise(late_sum = sum(late1) - sum(late),
            settle_sum = sum(settle1) - sum(settle))

```

4.4 Munging

Ask reasonable questions via slice dice.

```

# window operation: lag, first, nth,
df_clean %>%
  arrange(invoice_date) %>%
  group_by(country_code) %>%
  mutate(increase = invoice_amount - dplyr::lag(invoice_amount, default = 0),
         indicator = ifelse(increase > 0, 1, 0)) %>%
  ungroup() %>%
  mutate(settle_grp = (settle %% 10) * 10)

df_clean %>%
  group_by(country_code) %>%
  arrange(invoice_date) %>%
  summarise(n = n(),
            sales = sum(invoice_amount),
            first_disputed_late = first(late[disputed == 'Yes']),
            first_disputed_inv_date = first(invoice_date[disputed == 'Yes']),
            largest_late = max(late[disputed == 'Yes']),
            largest_inv_amt = invoice_amount[late == max(late)],
            .groups = 'drop')

```

Cut late into four categories based on the firm's credit policy.

```

sort(unique(df_clean$late))

df_late <- df_clean %>%
  dplyr::filter(late != 0) %>%
  mutate(reminder = case_when(late > 0 & late <= 10 ~ "1st email",
                               late > 10 & late <= 20 ~ "2nd email",
                               late > 20 & late <= 30 ~ "legal case",
                               TRUE ~ "bad debt"))

# anomaly by country
df_late %>%
  ggplot(aes(late, disputed, color = country_code)) +
  geom_boxplot() +
  theme_light()

# summary table
df_late %>%
  group_by(reminder, disputed) %>%
  summarise(across(late, tibble::lst(sum, min, max, sd)),
            .groups = 'drop') %>%
  gt::gt()

# clients without dispute do not pay.
df_late %>%
  dplyr::filter(disputed == 'No', reminder %in% c('legal case', 'bad debt'))

```

4.5 EDA

Focus on a handful of variables after dropped others.

```

df <- df_clean %>%
  select(-c(contains('date'), invoice_number))

# freq table
with(df, table(disputed, country_code) %>% addmargins())
tapply(df$invoice_amount, list(df$disputed, df$country_code), median)

# descriptive stats
df %>%

```

Figure 4.1: Data munging

```
select(where(is.numeric)) %>%
summary()

# normal distribution
df %>%
  ggplot(aes(invoice_amount, fill = disputed)) +
  geom_histogram(bins = 10, position = 'dodge') +
  geom_vline(xintercept = median(df$invoice_amount), color = 'red',
             size = 3, linetype = "dashed") +
  theme_light()

# correlation
df %>%
  select(where(is.numeric)) %>%
  cor() %>%
  corrplot::corrplot(method = 'color', order = 'FPC', type = 'lower', diag = FALSE)

df %>%
  select(where(is.numeric)) %>%
```

```

corrr::correlate() %>%
corrr::rearrange() %>%
corrr::shave() %>%
corrr::fashion()

```

4.6 Model

Read more about logistic regression [here](#), [here](#), and [here](#).

```

# easy stats plot
df %>%
  mutate(prob = ifelse(disputed == "Yes", 1, 0)) %>%
  ggplot(aes(late, prob)) +
  geom_point(alpha = .2) +
  geom_smooth(method = "glm", method.args = list(family = "binomial")) +
  theme_light()

# model comparison
df_mod <- df %>%
  mutate(disputed = as.factor(disputed))

mod1 <- glm(disputed ~ late, family = "binomial", data = df_mod)
mod2 <- glm(disputed ~ late + settle + invoice_amount,
             family = "binomial", data = df_mod)

summary(mod1)
anova(mod1, mod2, test = "Chisq")

# model diagnostic
df_mod_res <- broom::augment(mod1, df_mod) %>%
  mutate(pred = ifelse(.fitted > .5, "Yes", "No") %>% as.factor())

# confusion matrix
df_mod_res %>%
  yardstick::conf_mat(disputed, pred) %>%
  autoplot()

# plot pred
df_mod_res %>%

```

```

mutate(res = disputed == pred) %>%
ggplot(aes(invoice_amount, settle, color = res)) +
geom_point() +
theme_light()

df_mod_res %>%
ggplot(aes(invoice_amount, settle, color = disputed)) +
geom_point() +
facet_wrap(~pred) +
theme_light()

```

4.7 Report

```

library(patchwork)
library(ggtext)
library(showtext)

p1 <- df %>%
ggplot(aes(invoice_amount, settle, color = disputed)) +
geom_point() +
scale_color_manual(labels = c("Agreed", 'Disputed'),
values = c("#9AC2BB", '#E99184')) +
guides(color = guide_legend(title.position = "top", title = ""))
labs(x = "", y = "Settlement days") +
theme_light() +
theme(
  legend.position = c(.95, .98),
  legend.background = element_rect(color = "transparent", fill = 'transparent'),
  legend.box.background = element_rect(color = "transparent", fill = "transparent"),
  legend.key = element_rect(colour = "transparent", fill = "transparent")
)

p2 <- df %>%
group_by(if_late = late == 0) %>%
ggplot(aes(invoice_amount, settle, color = disputed)) +
geom_point(show.legend = FALSE) +
scale_color_manual(labels = c("Agreed", 'Disputed'),
values = c("#9AC2BB", '#E99184')) +
facet_wrap(~if_late) +

```

```

  labs(caption = "@RAudit Solution | **Stewart Li**<br>(Data source: Kaggle)",
       x = "Invoice amount",
       y = "Settlement days") +
  theme_light() +
  theme(
    axis.title.y = element_text(margin = margin(b = 1, unit = "in")),
    strip.text = element_text(color = '#2D4248'),
    strip.background = element_blank(),
    plot.caption = element_markdown(lineheight = 1.2)
  )
)

p1 / p2 +
  plot_annotation(
    title = "The <span style = 'color:#E99184;'>Analysis</span> of cash collection",
    subtitle = 'Focus on those slow settlement without dispute',
    tag_levels = 'A'
  ) &
  theme(plot.tag = element_text(size = 8),
        plot.title = element_markdown())

```

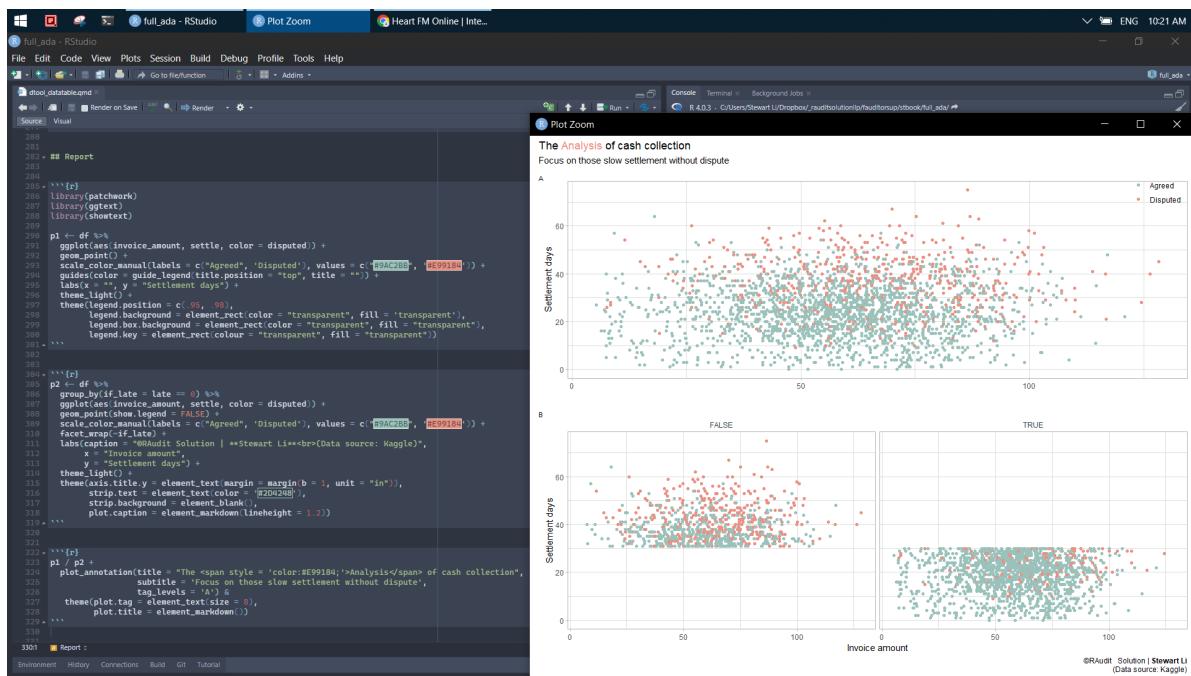


Figure 4.2: Combined plot

References

- Li, Stewart, Richard Fisher, and Michael Falta. 2020. “The Effectiveness of Artificial Neural Networks Applied to Analytical Procedures Using High Level Data: A Simulation Analysis.” *Meditari Accountancy Research* 29 (6): 1425–50. <https://doi.org/10.1108/medar-06-2020-0920>.
- Wickham, Hadley. 2016. *Ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York. <https://ggplot2.tidyverse.org>.