Data Analytics Engineering

For Accountants and Auditors

Stewart Li

2024-01-29

Table of contents

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, financial statement preparation, insurance data analysis, 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 Window manager `komorebi`, 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.

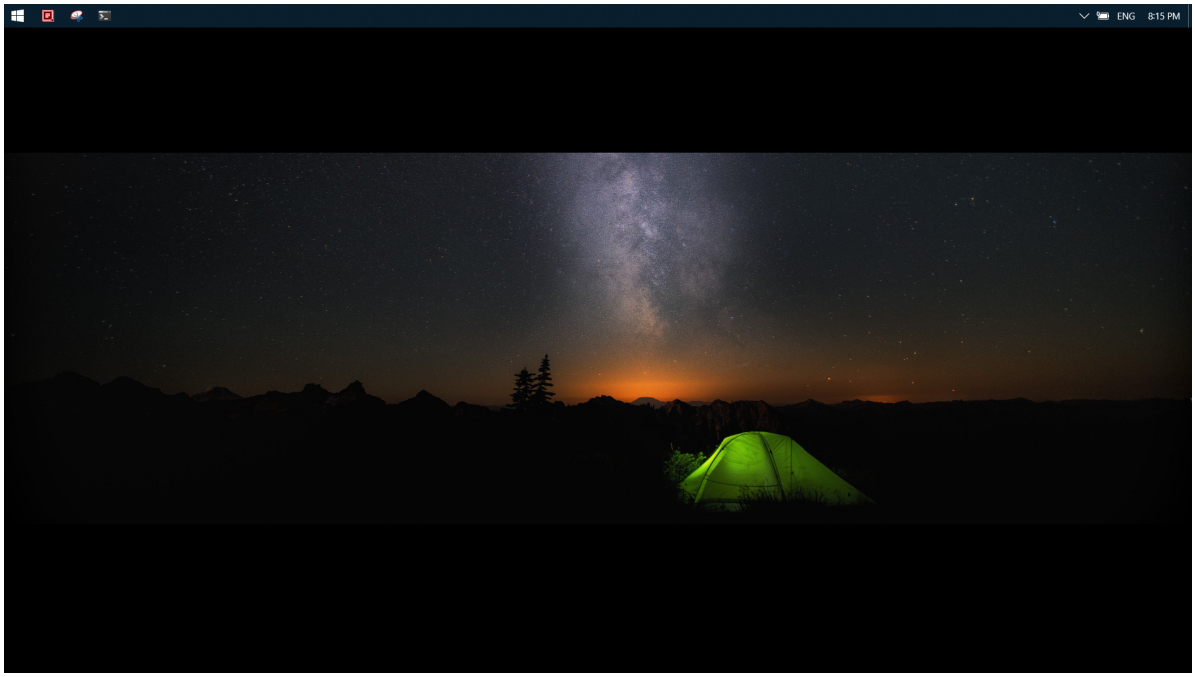

Figure 1.1: Desktop

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 `rofi` applications, manage `pass`, `rsync` files, `quarto` markdown, `sftp` to server, `ssh` into remote machines, 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.

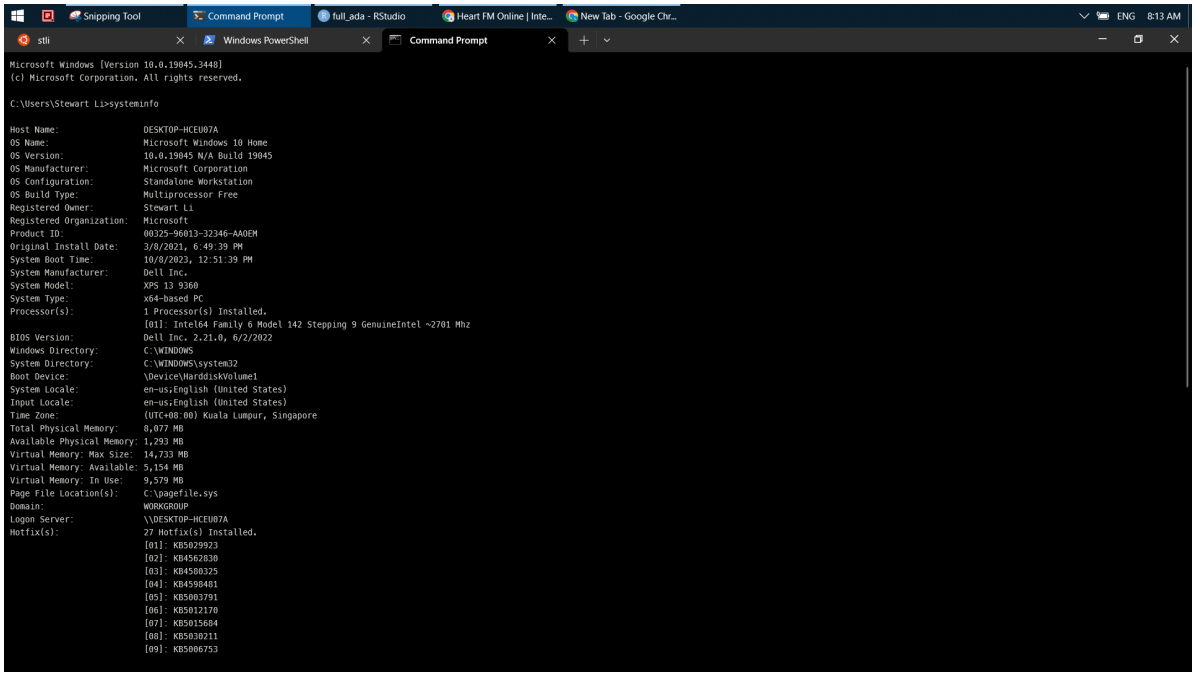

Figure 1.2: CMD

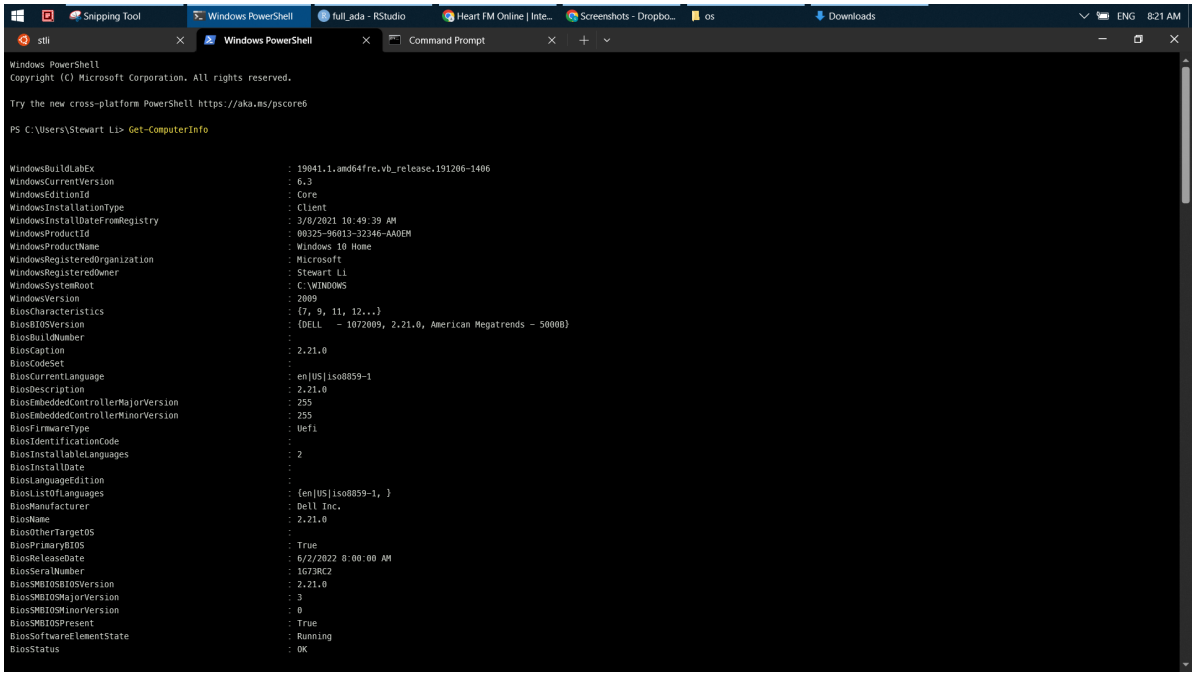

Figure 1.3: PowerShell

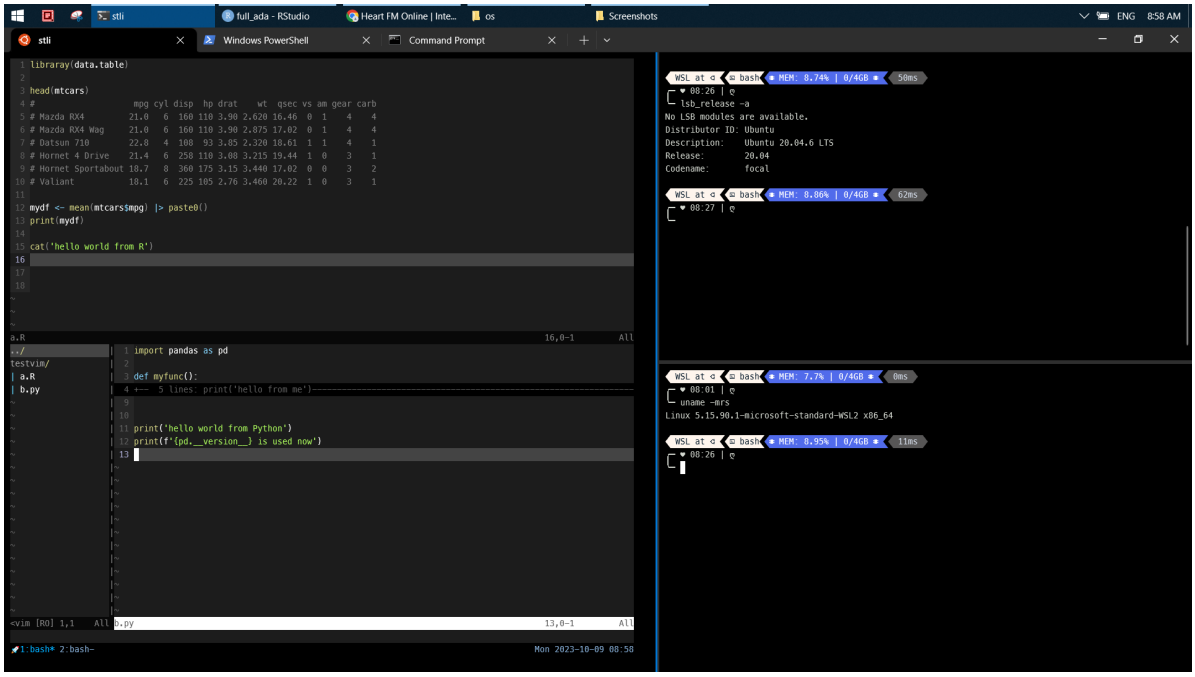

Figure 1.4: Ubuntu

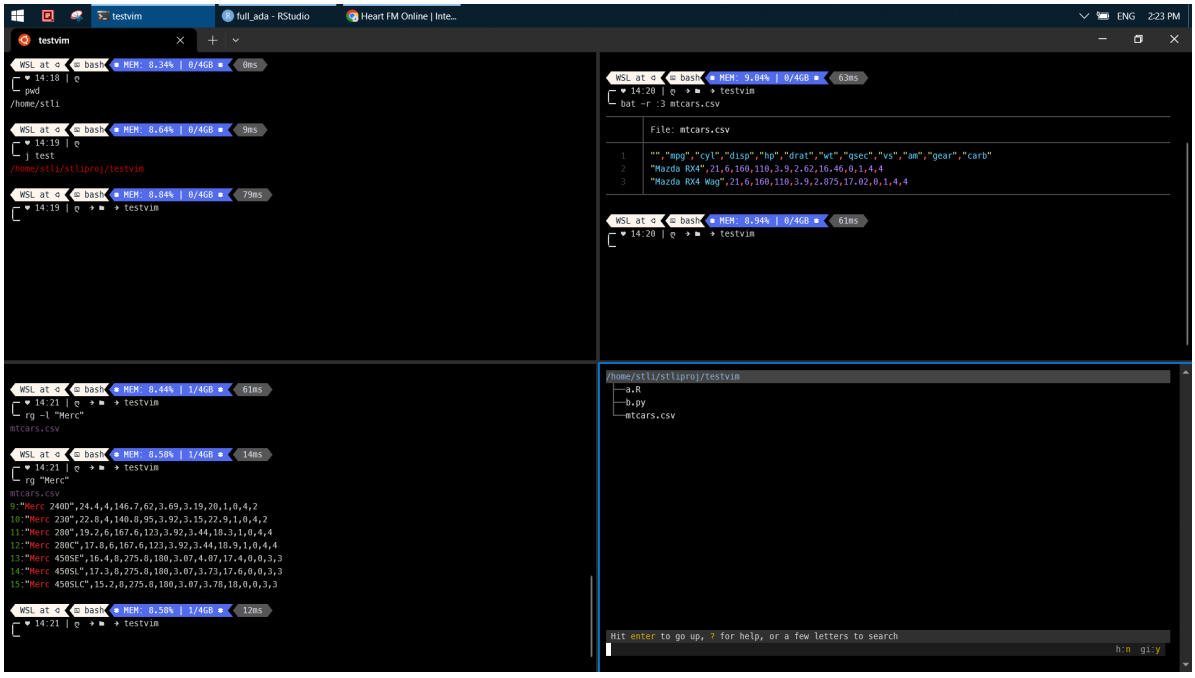

Figure 1.5: Terminal tools

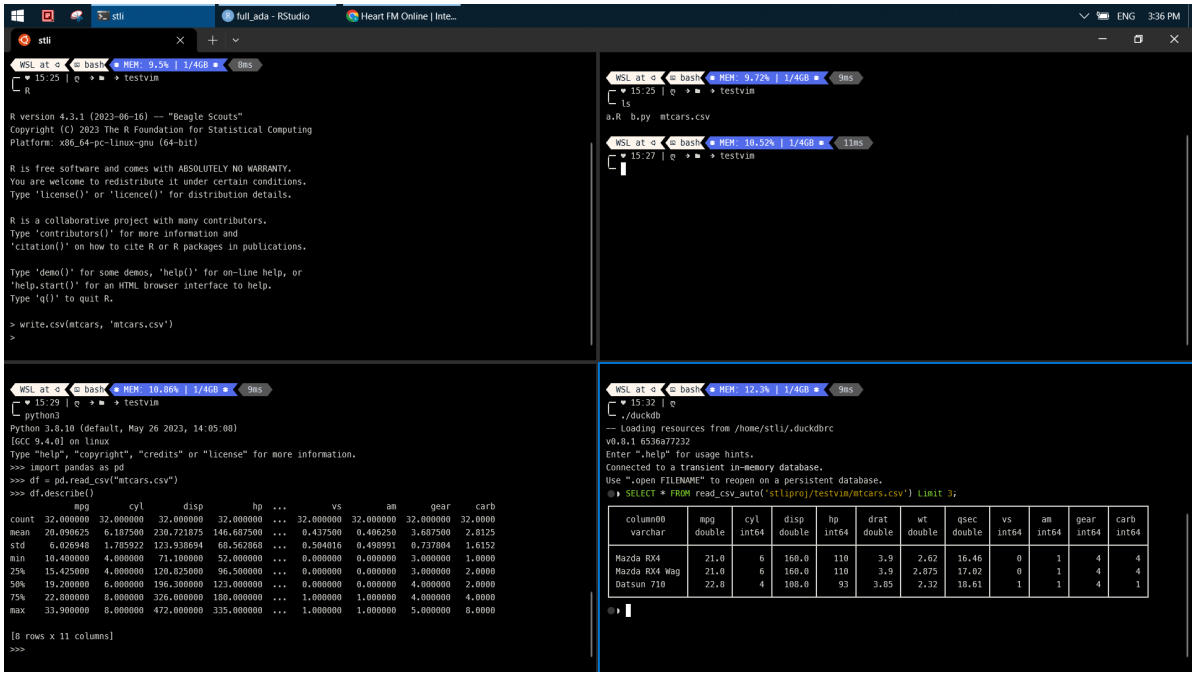

Figure 1.6: R, Python, DuckDB

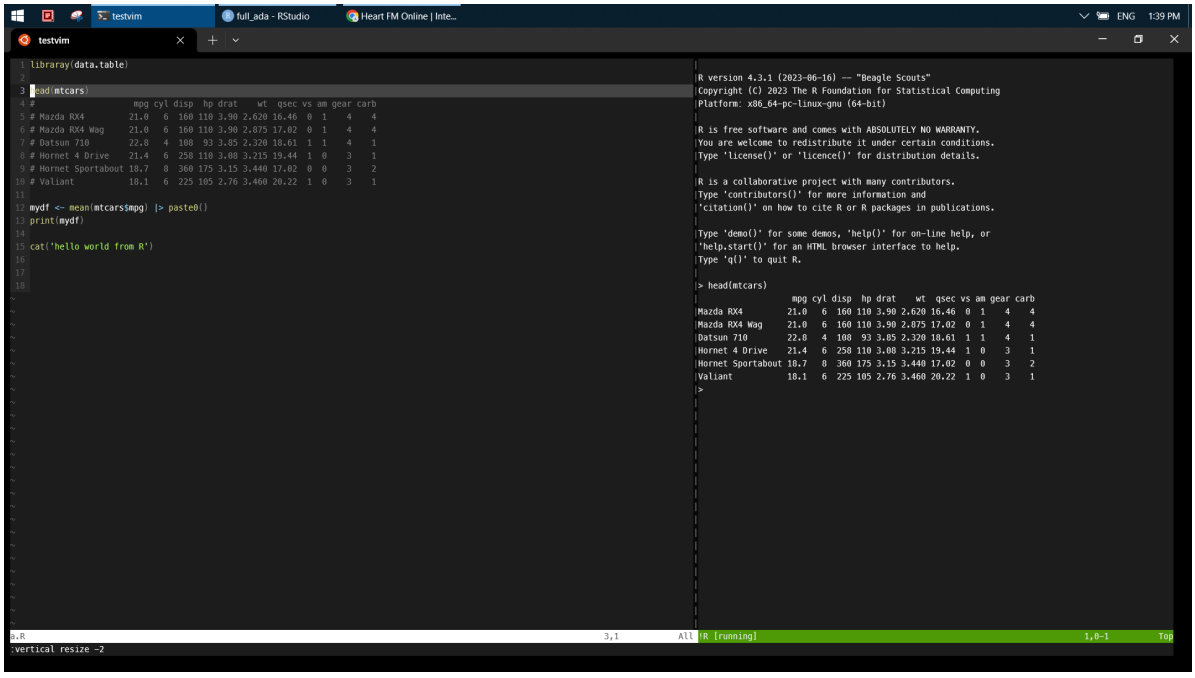

Figure 1.7: Vim - R

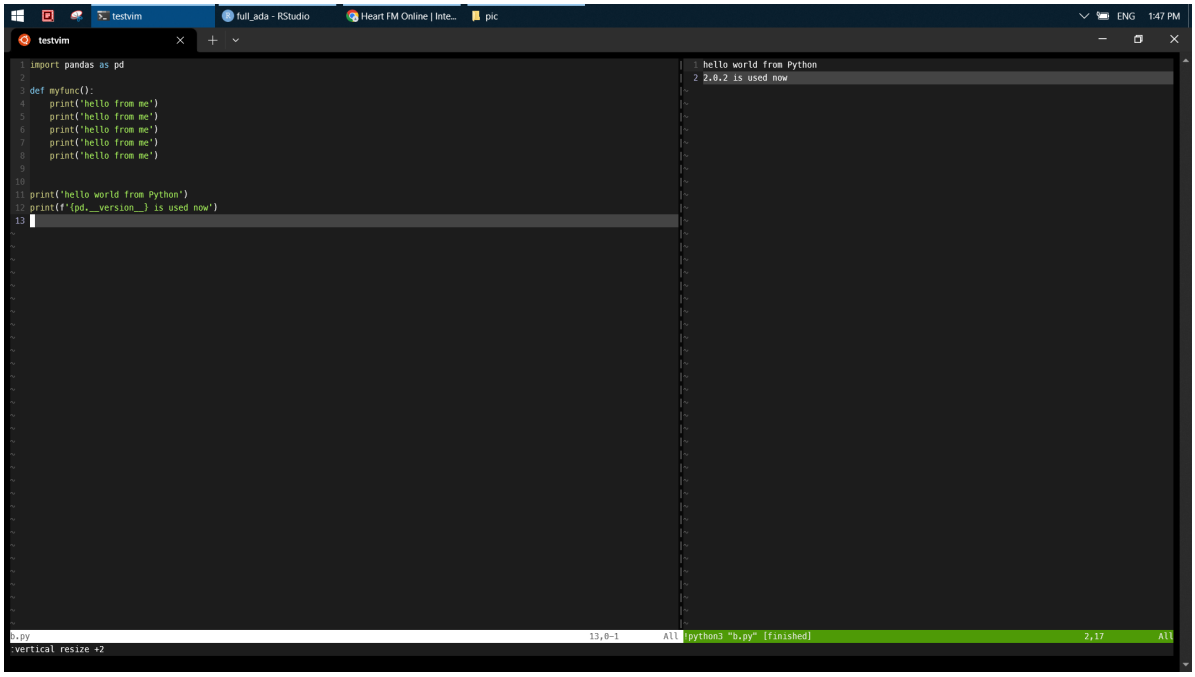

Figure 1.8: Vim - Python

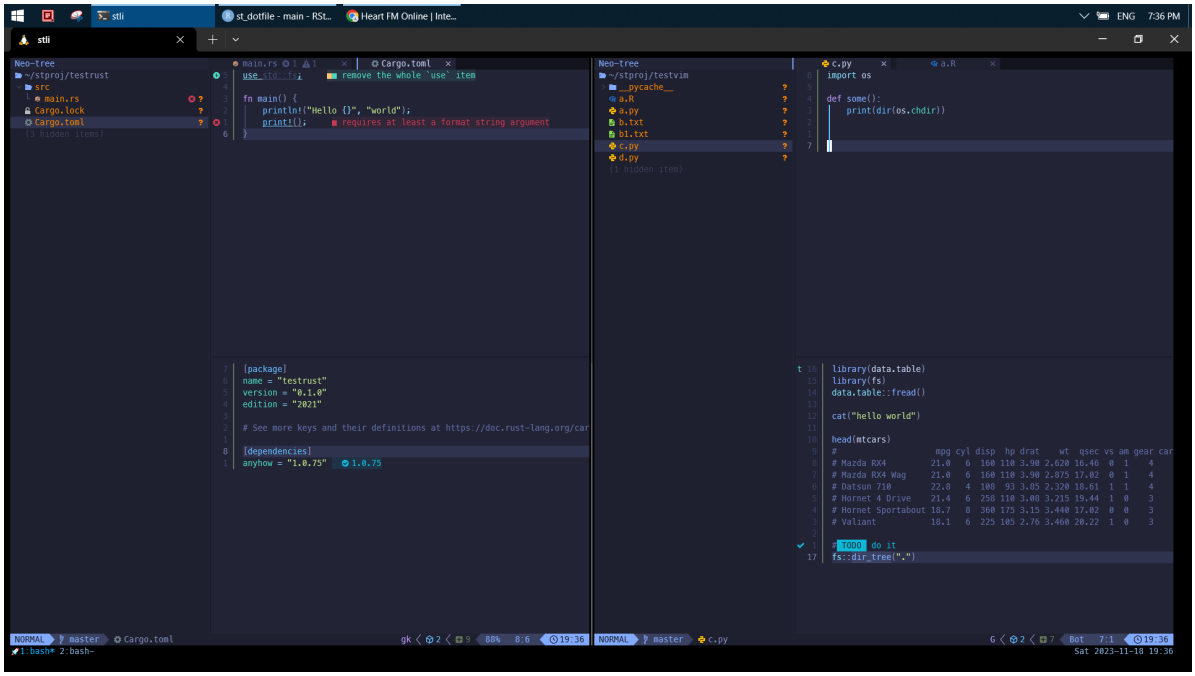

Figure 1.9: Tmux - Nvim 1

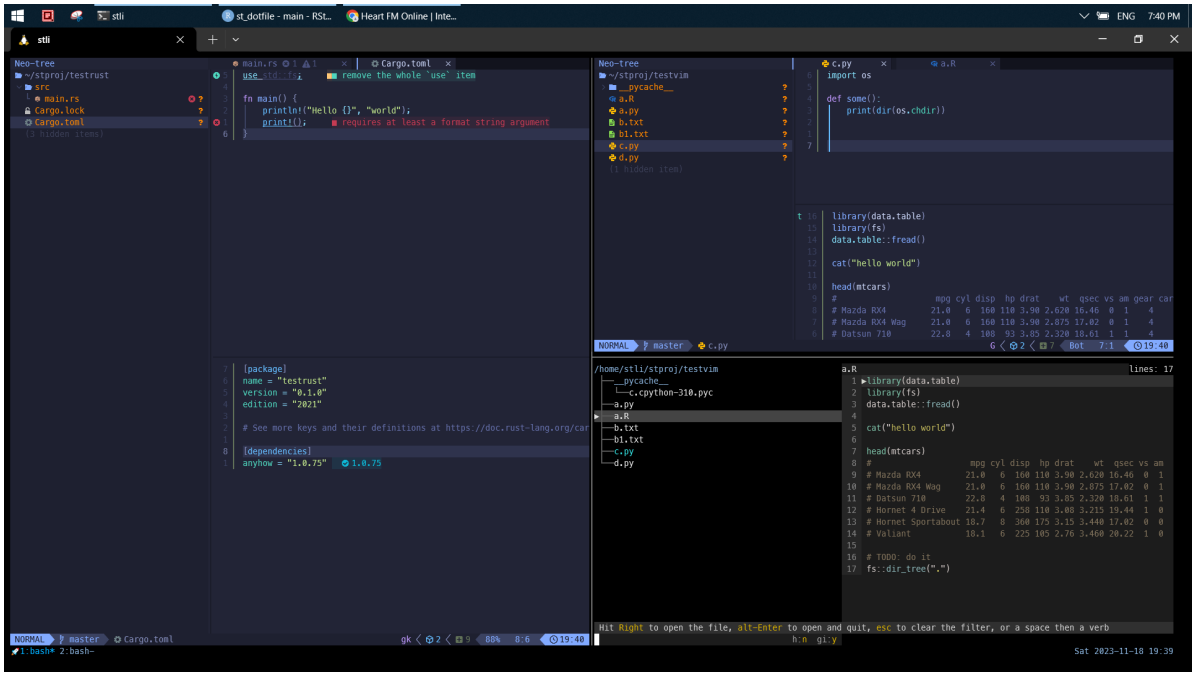

Figure 1.10: Tmux -Nvim 2

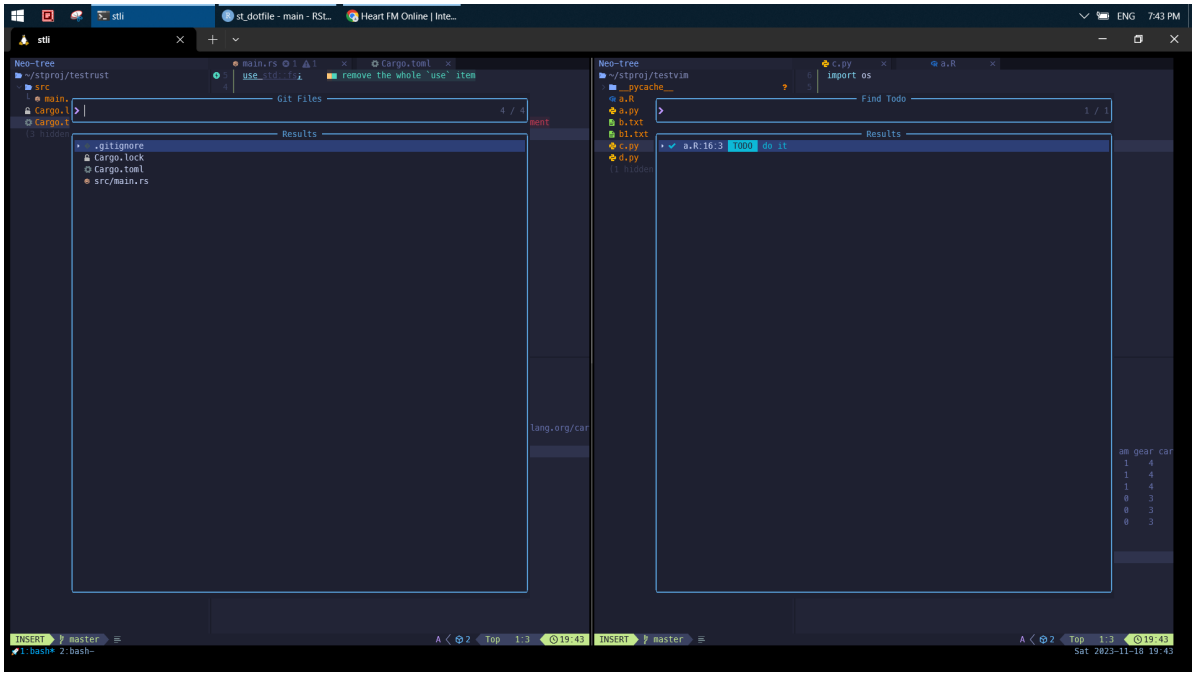

Figure 1.11: Tmux - Nvim 3

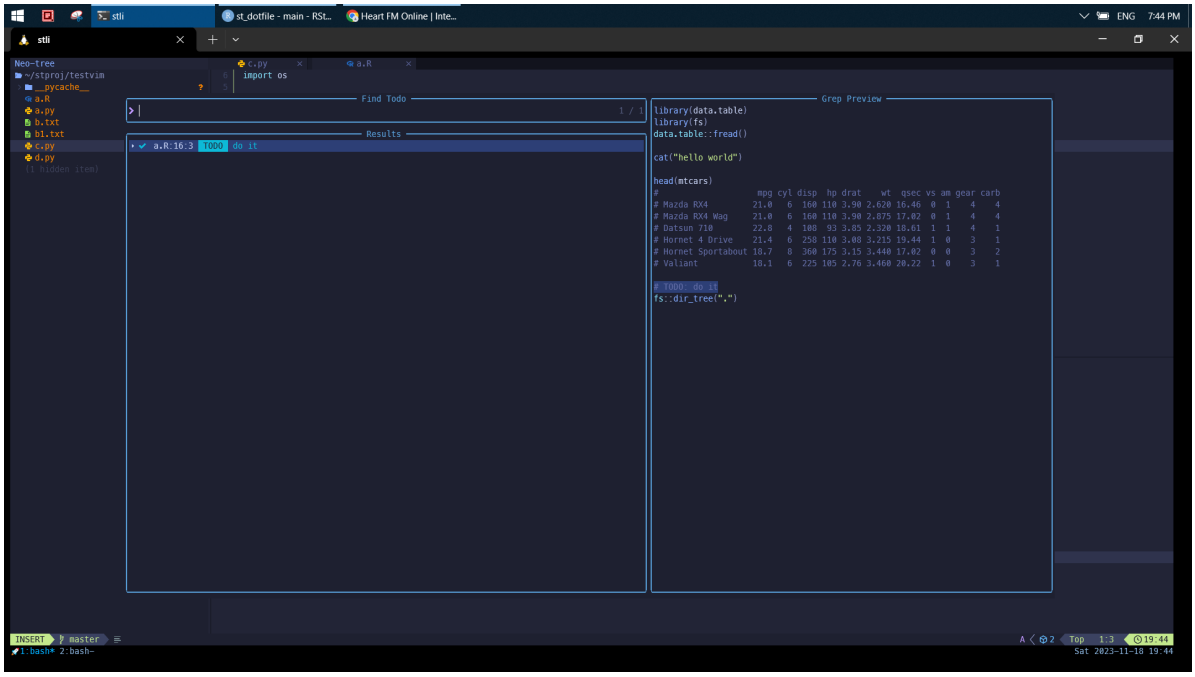

Figure 1.12: Tmux -Nvim 4

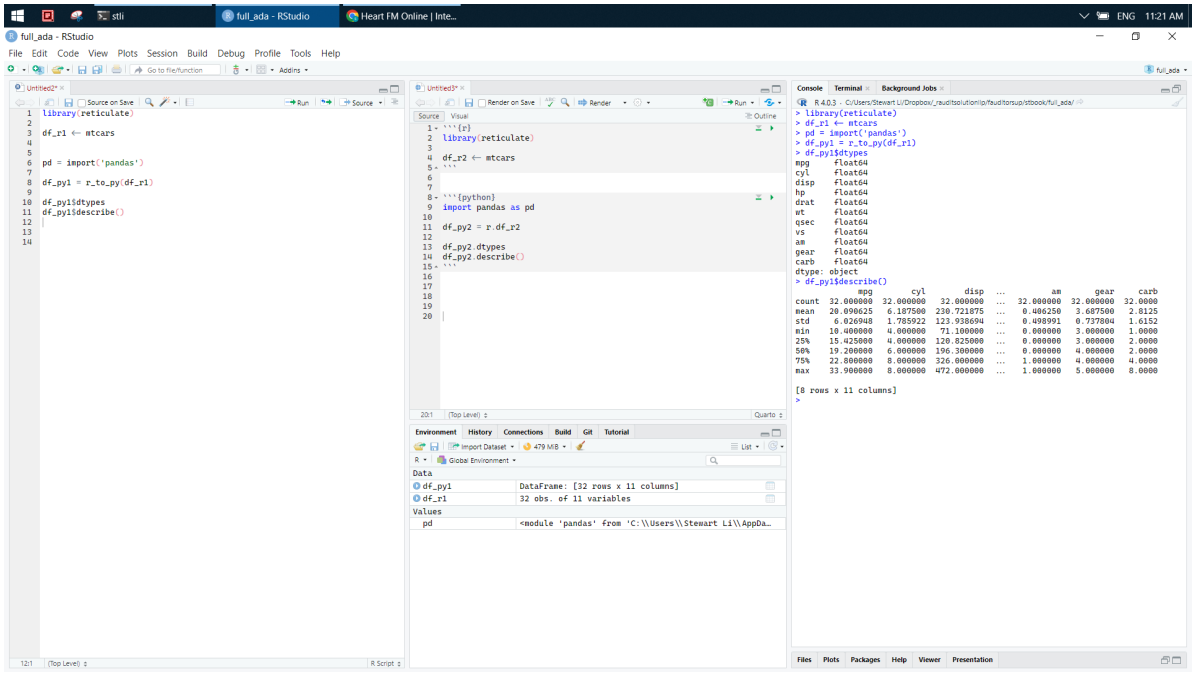

Figure 1.13: RStudio - R

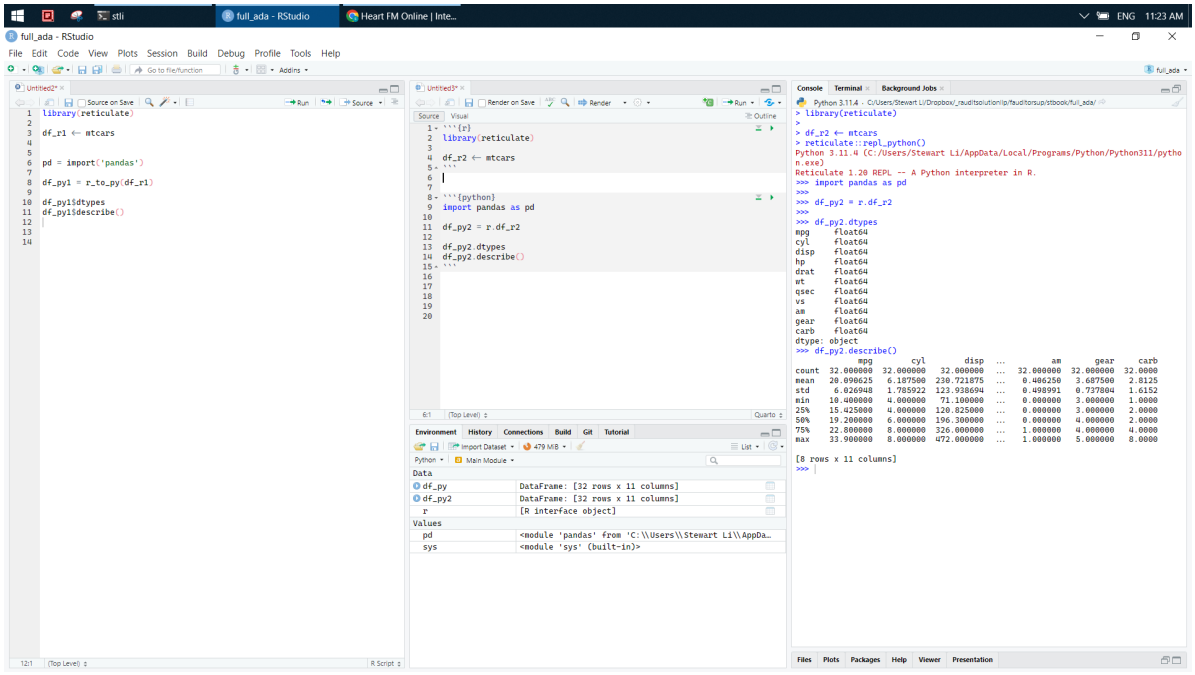

Figure 1.14: RStudio - Python

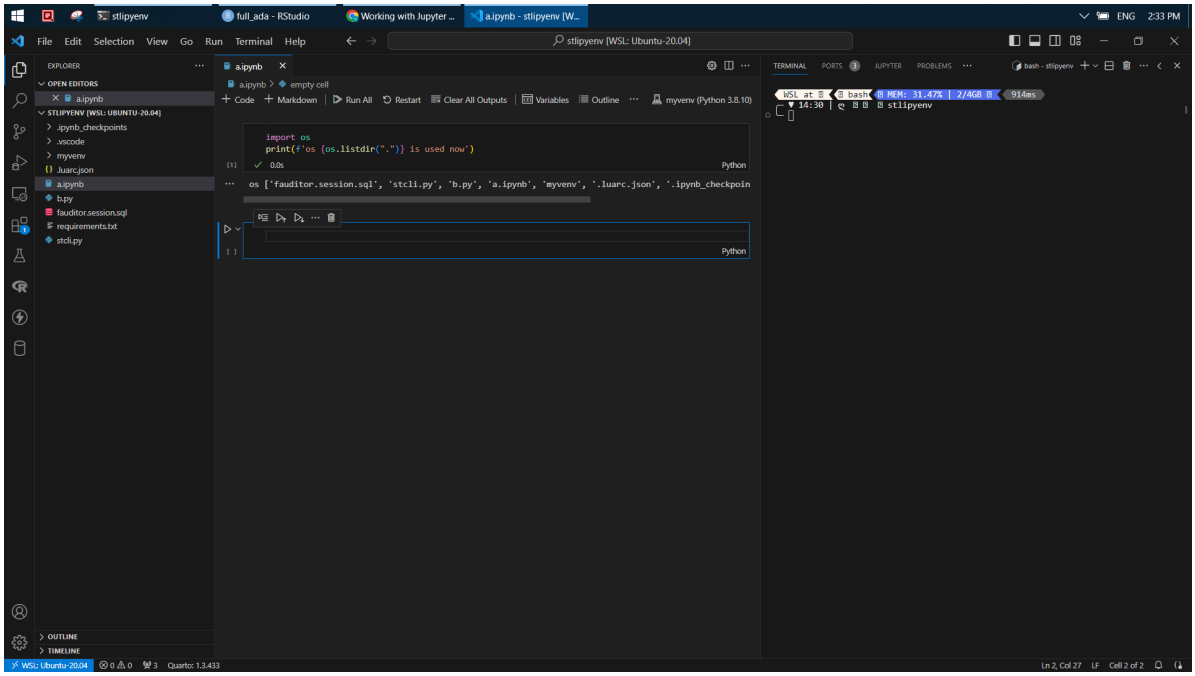

Figure 1.15: VS Code in Linux - Jupyter

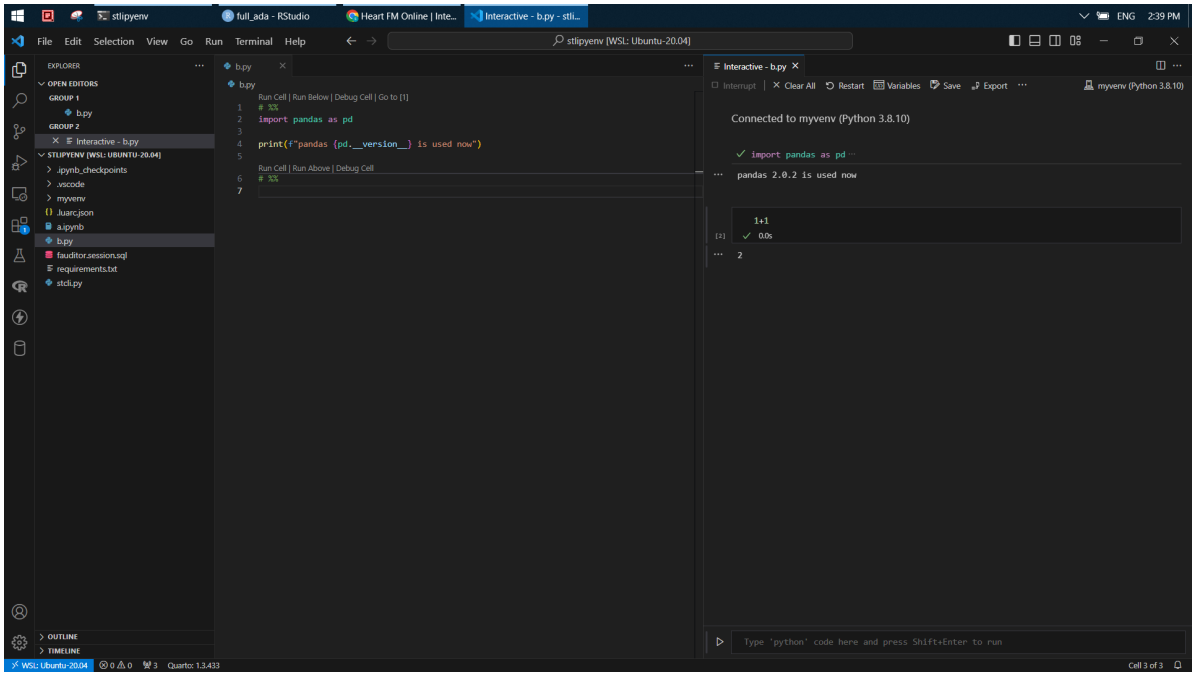

Figure 1.16: VS Code in Linux - Interactive cell

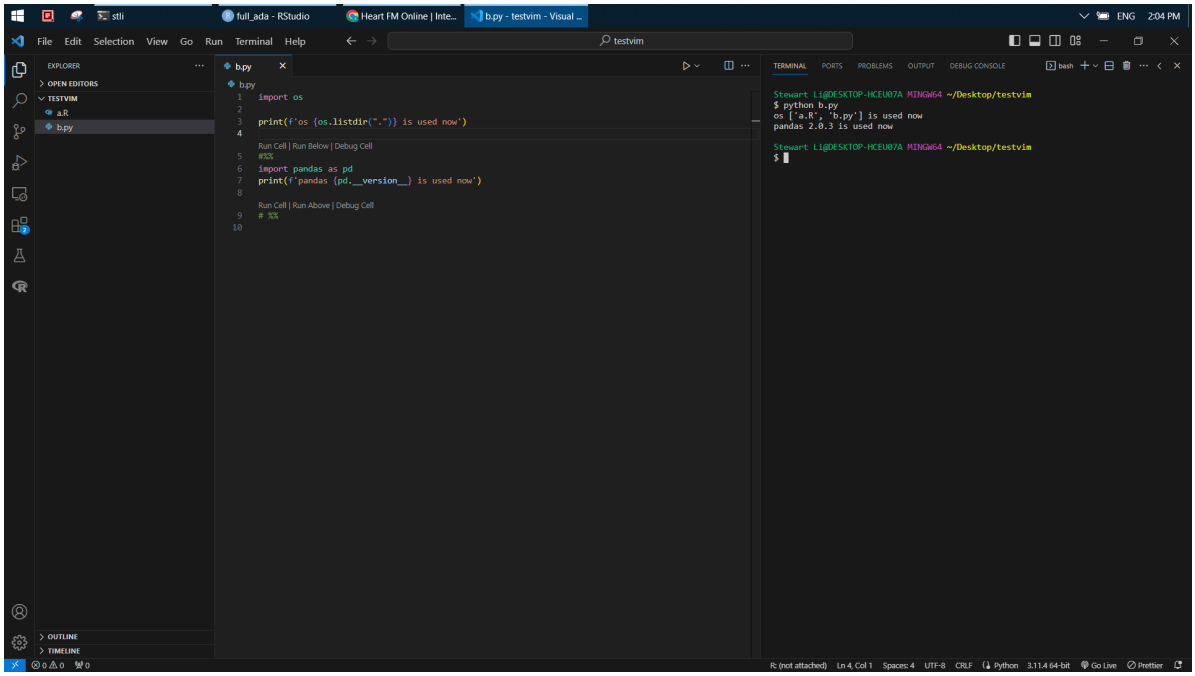

Figure 1.17: VS Code in Windows - Script

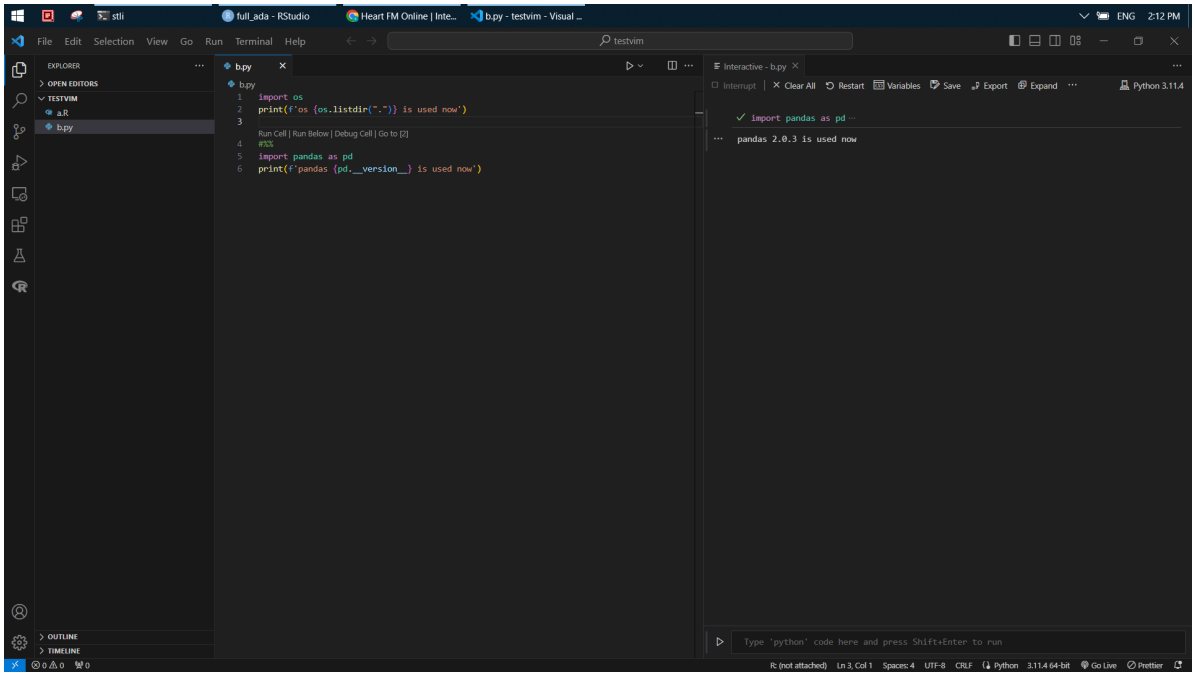

Figure 1.18: VS Code in Windows - Interactive cell

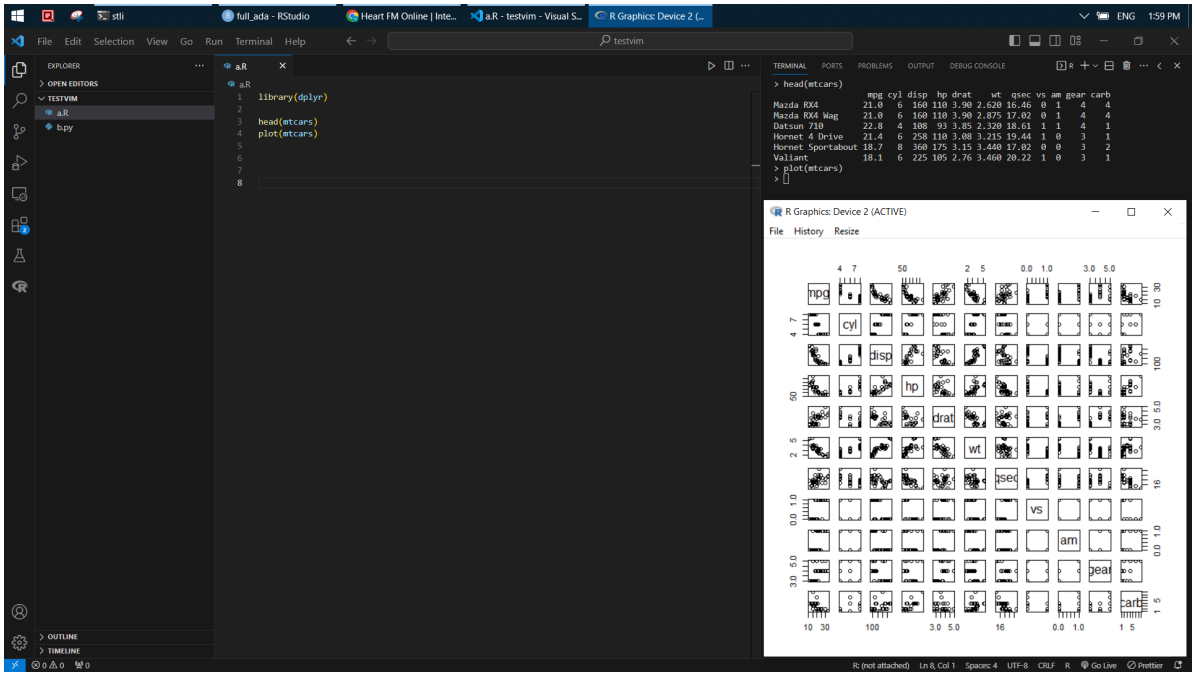

Figure 1.19: VS Code in Windows - R

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.

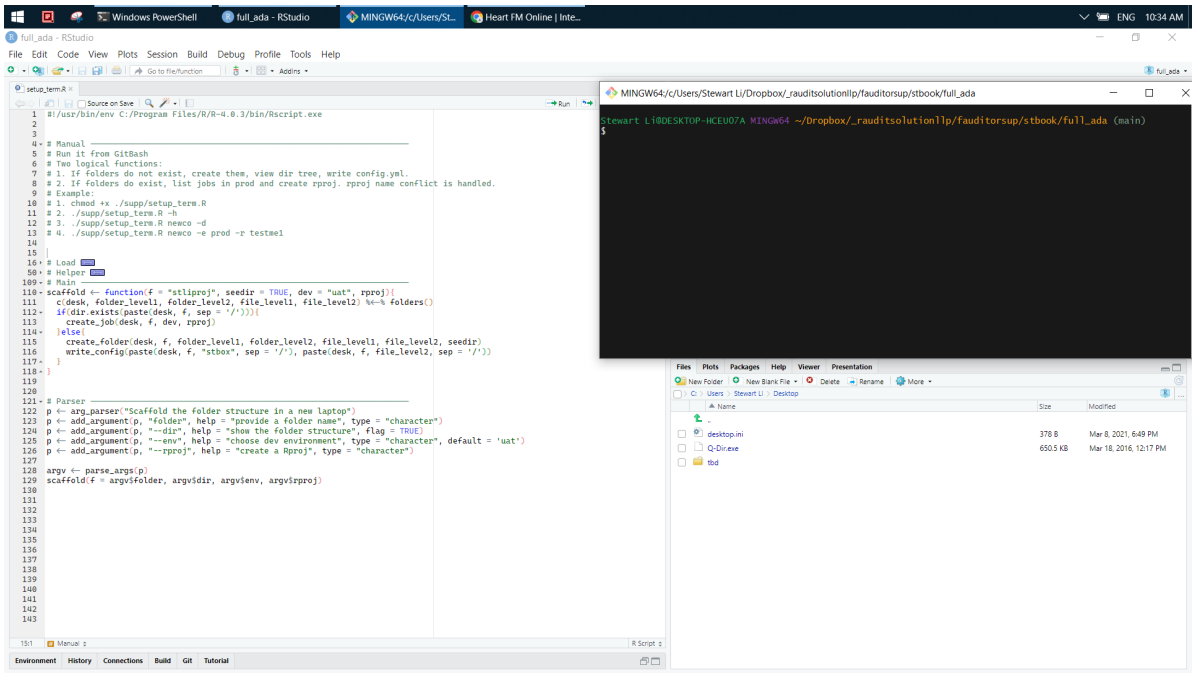

Figure 1.20: CLI R - GitBash 1

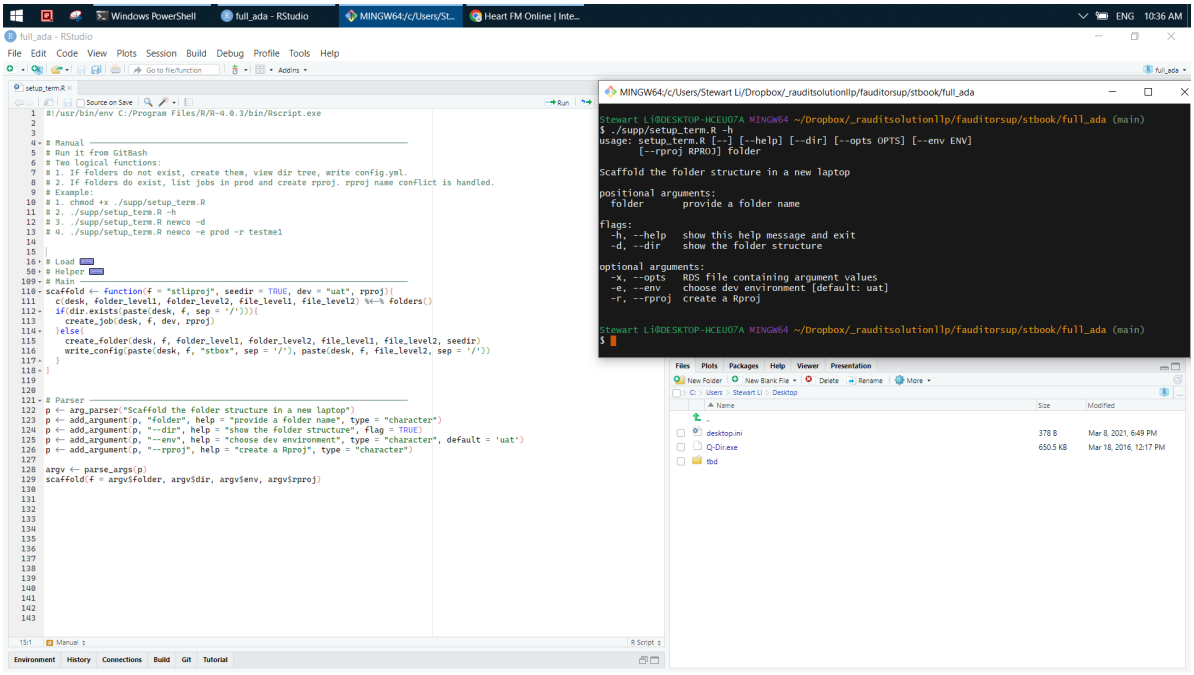

Figure 1.21: CLI R - GitBash 2

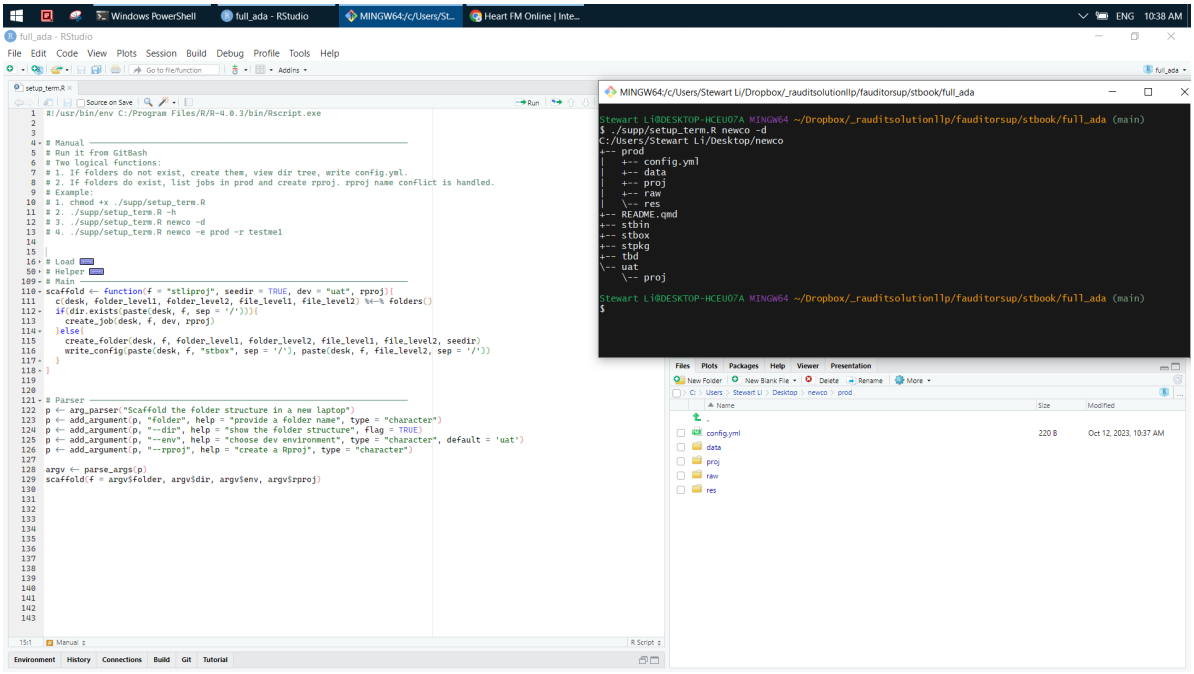

Figure 1.22: CLI R - GitBash 3

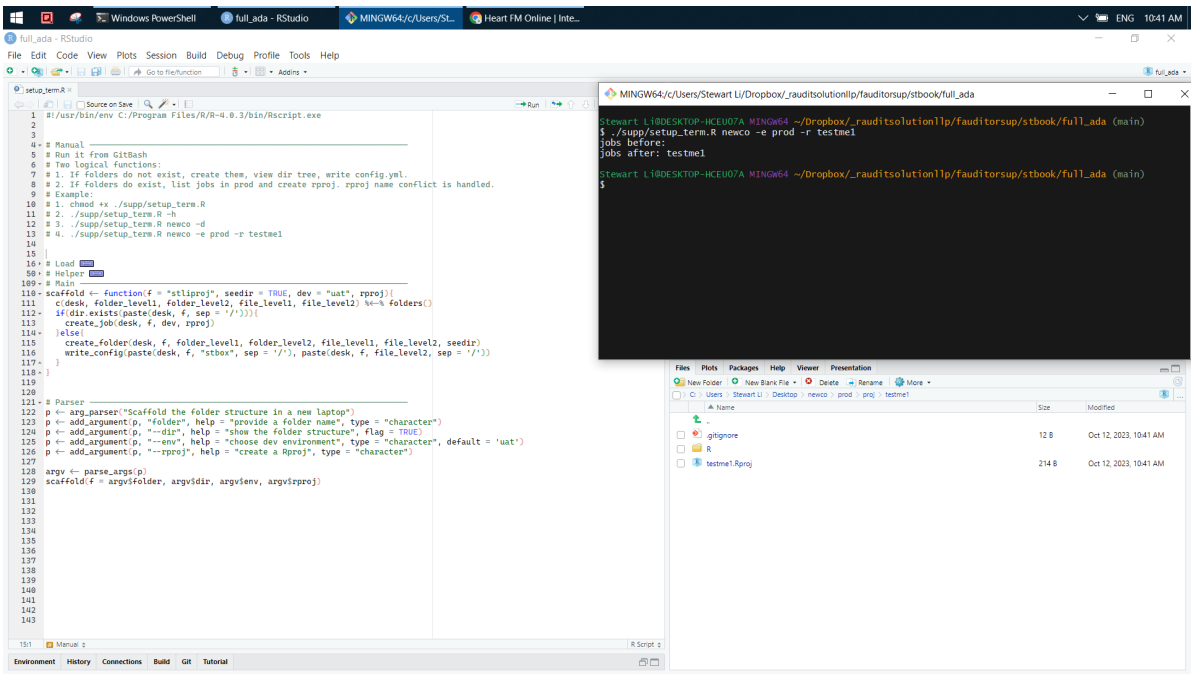

Figure 1.23: CLI R - GitBash 4

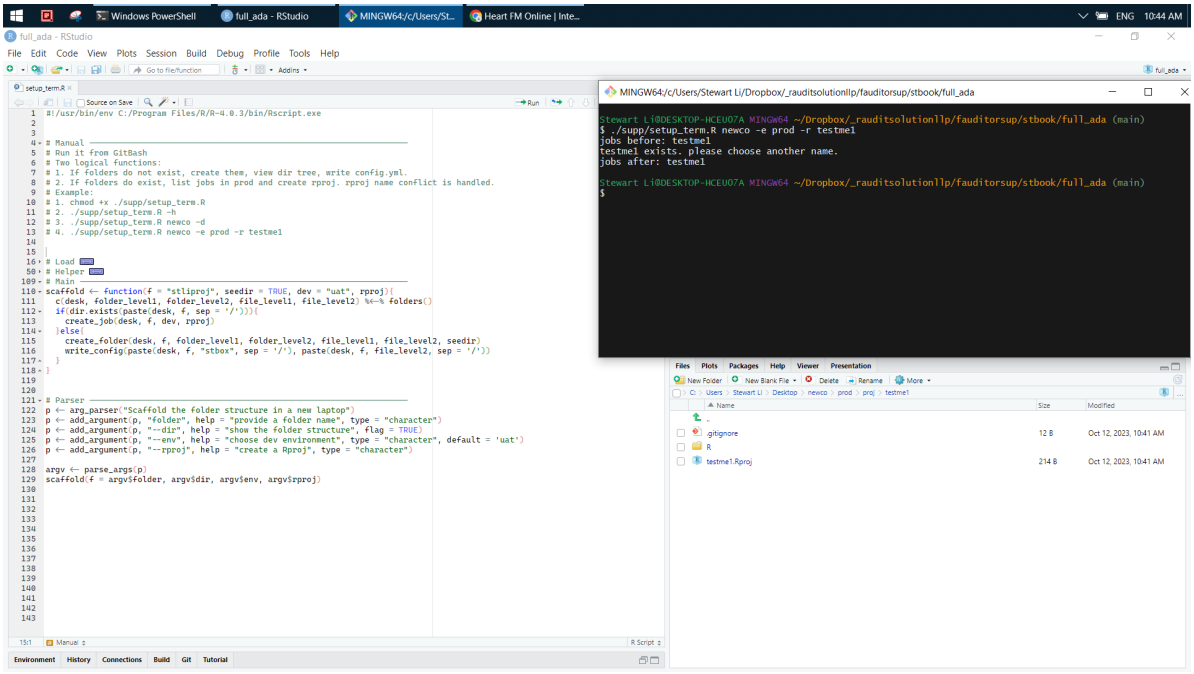

Figure 1.24: CLI R - GitBash 5

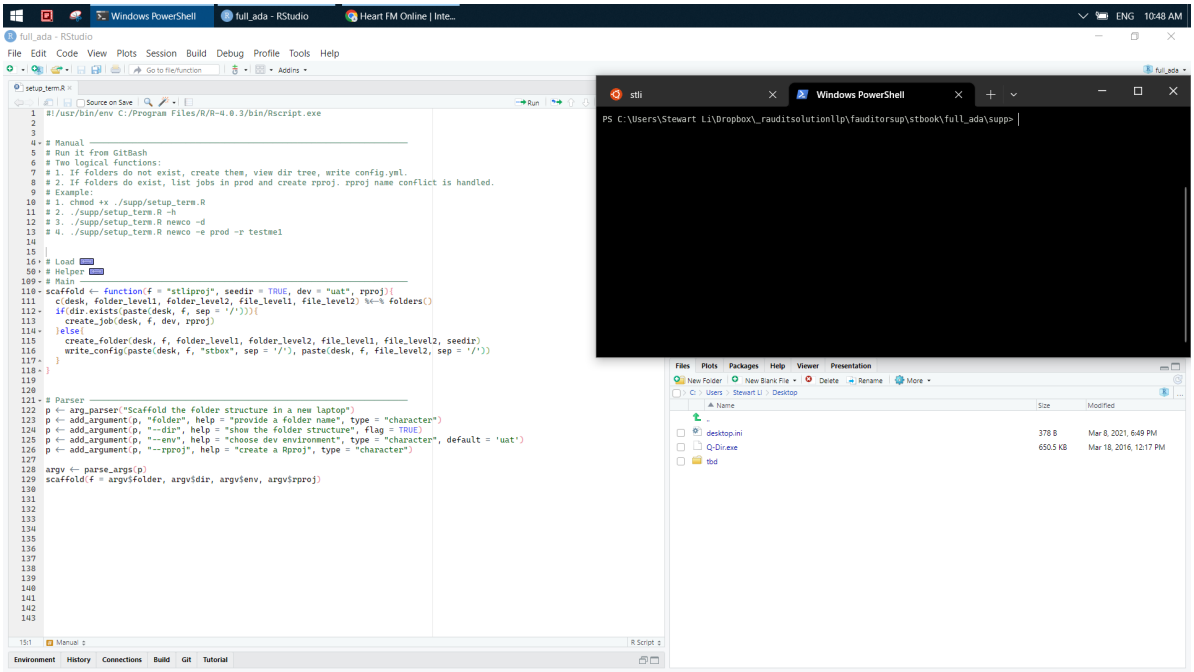

Figure 1.25: CLI R - PowerShell 1

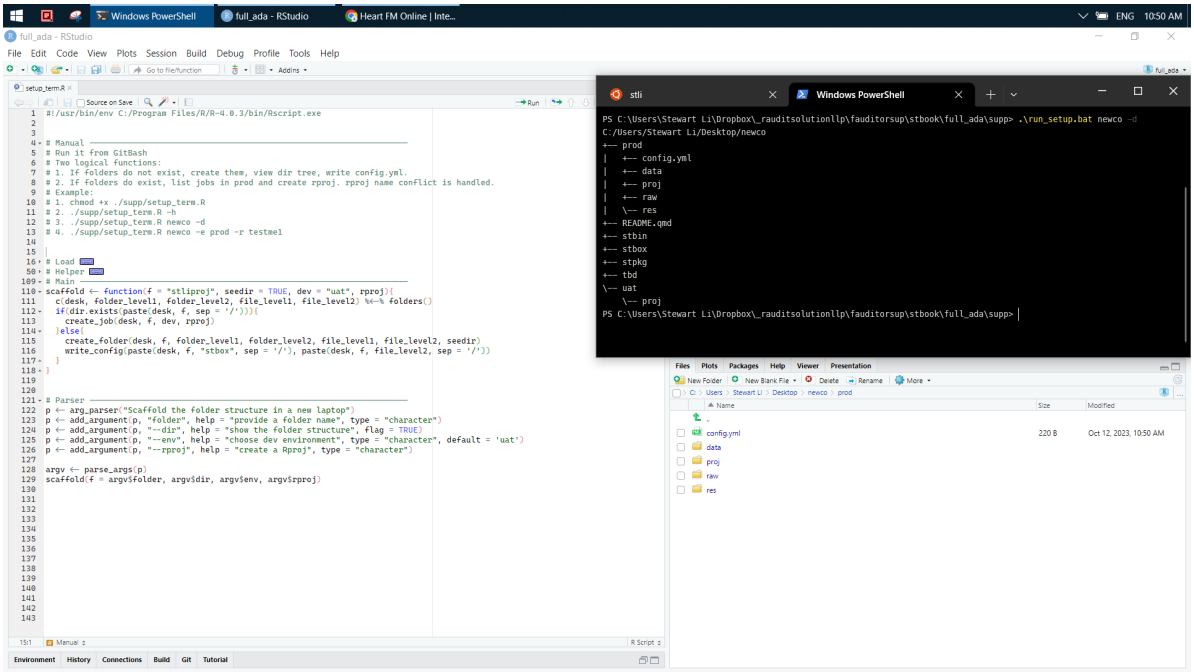

Figure 1.26: CLI R - PowerShell 2

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`.

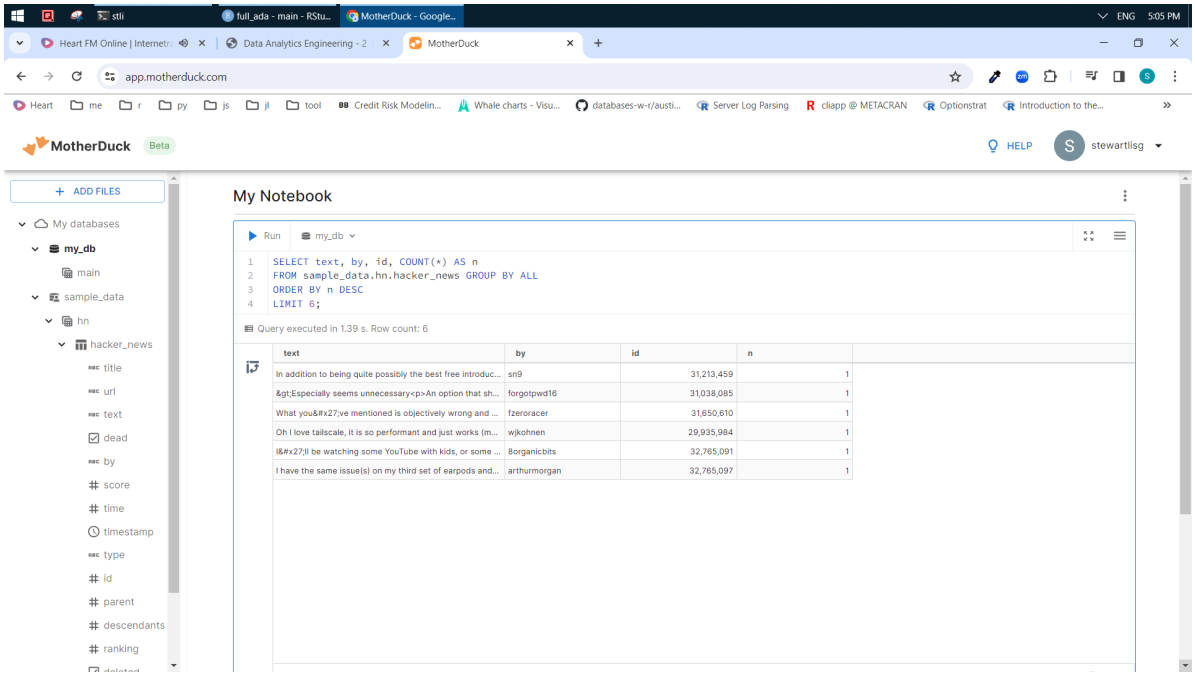

Figure 2.1: DuckDB cloud

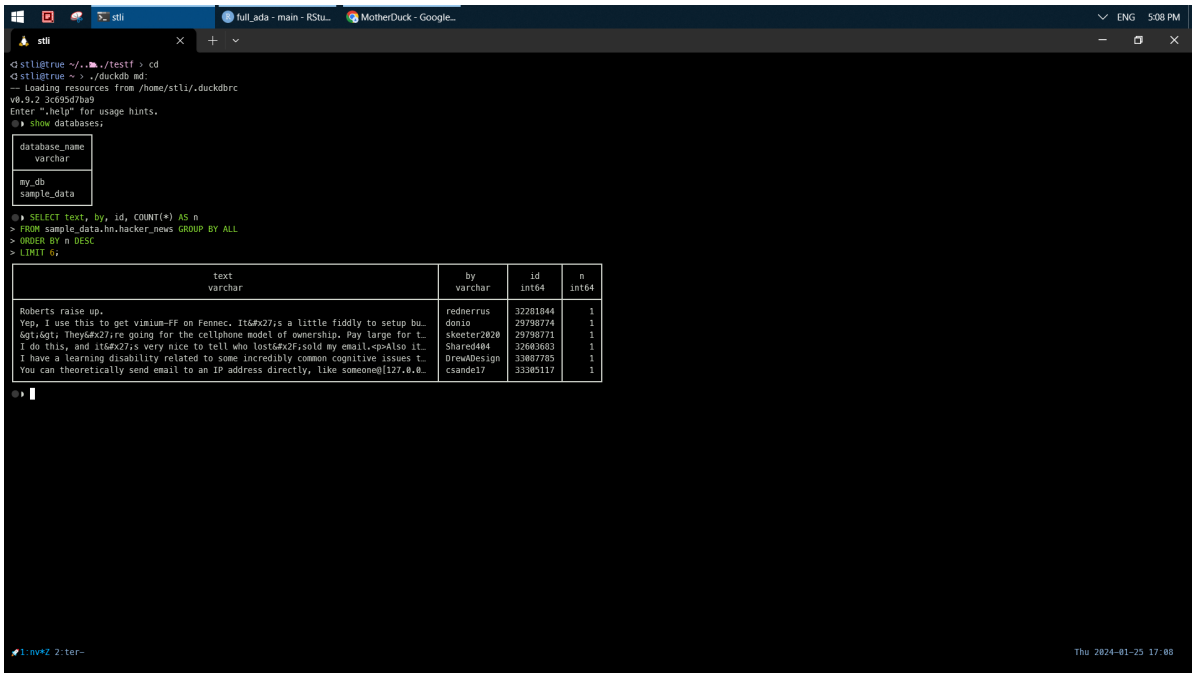

Figure 2.2: DuckDB terminal

3 Example 1

It is very useful to create a micro service `API` internally.

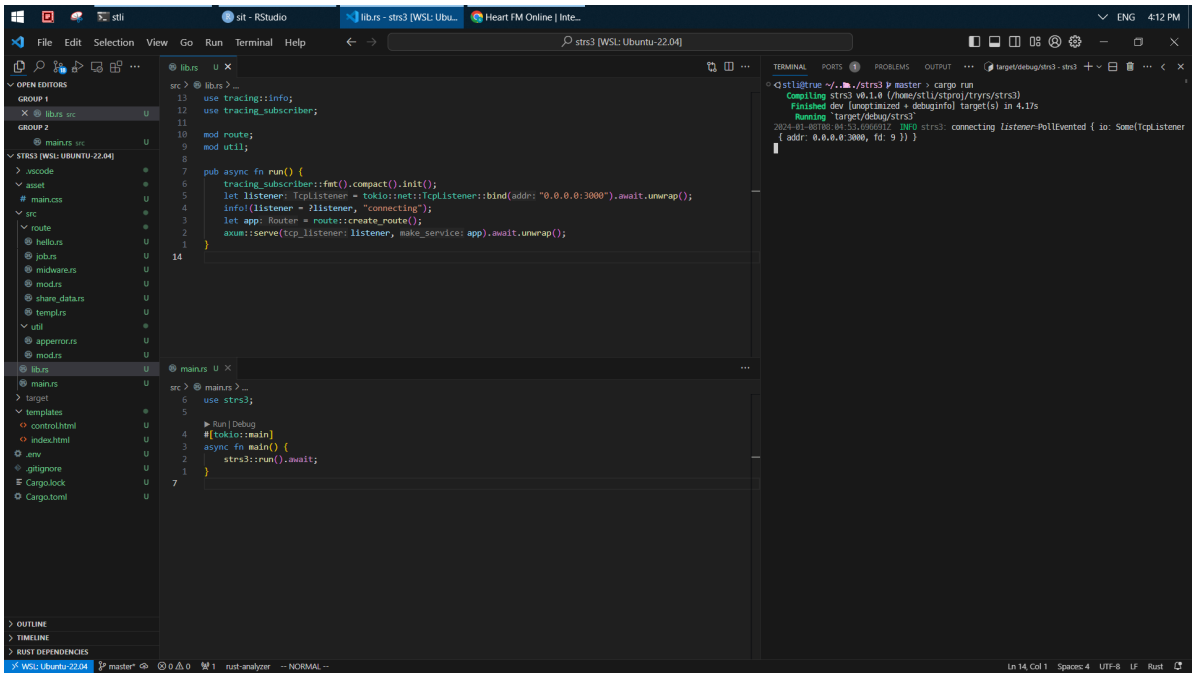

Figure 3.1: Web server

```
httr2::request('localhost:3000/share') %>%
  httr2::req_perform() %>%
  httr2::resp_body_string()
```

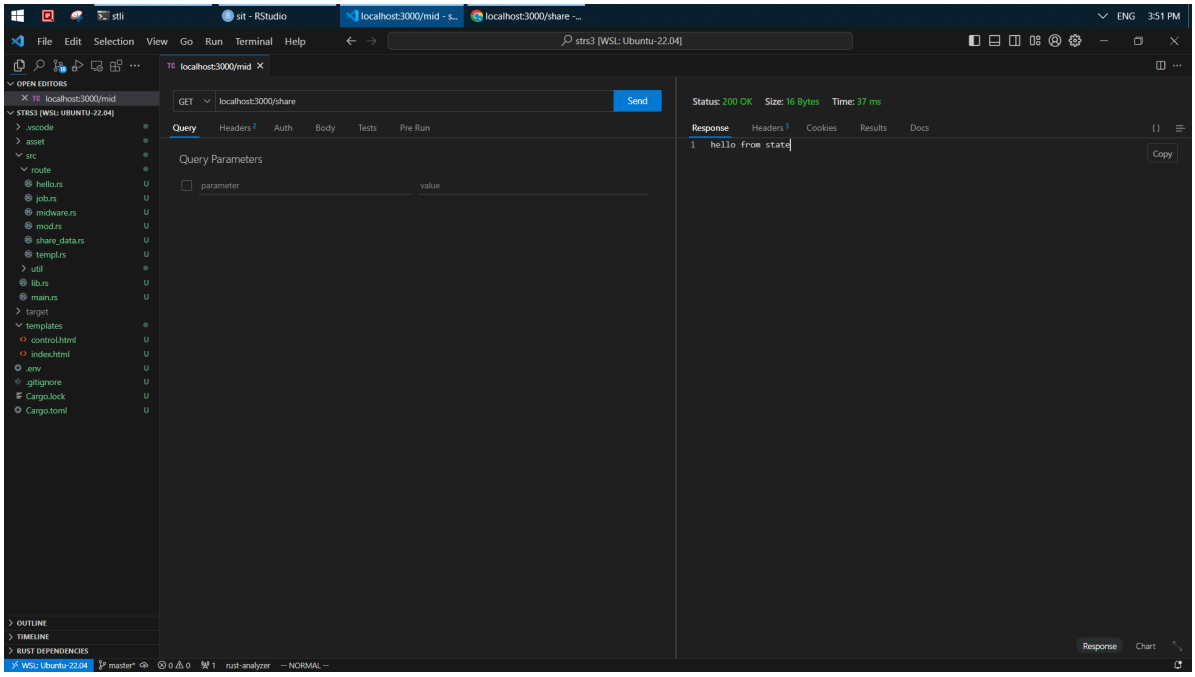

Figure 3.2: Get

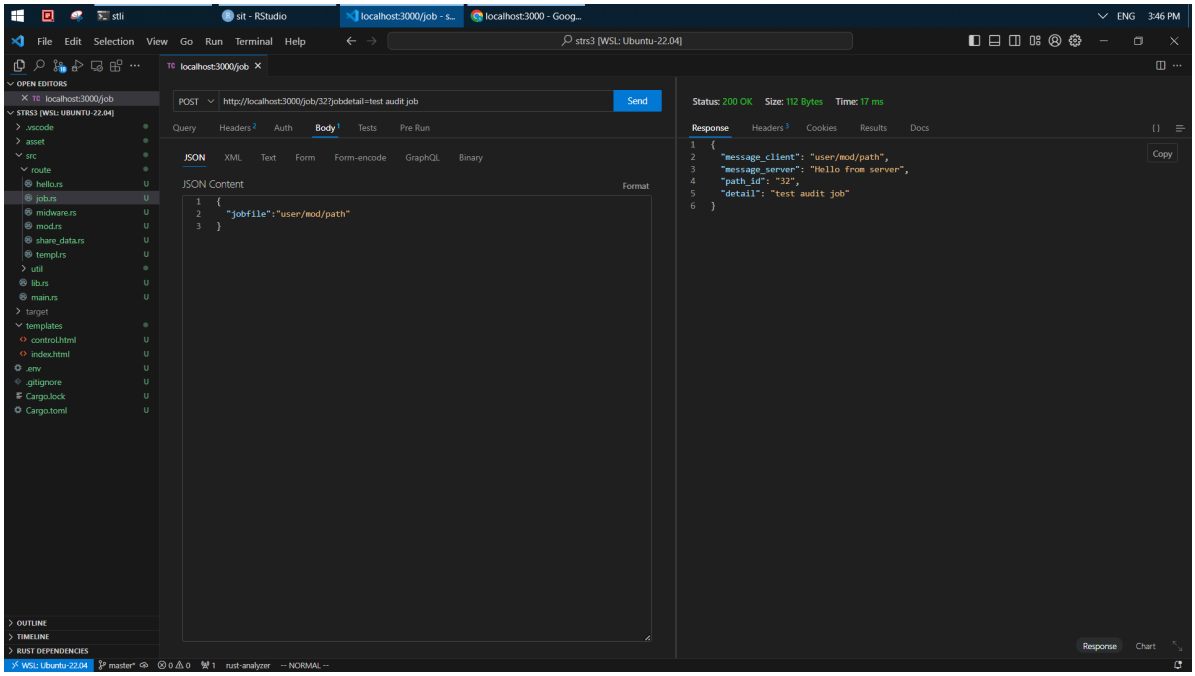

Figure 3.3: Post

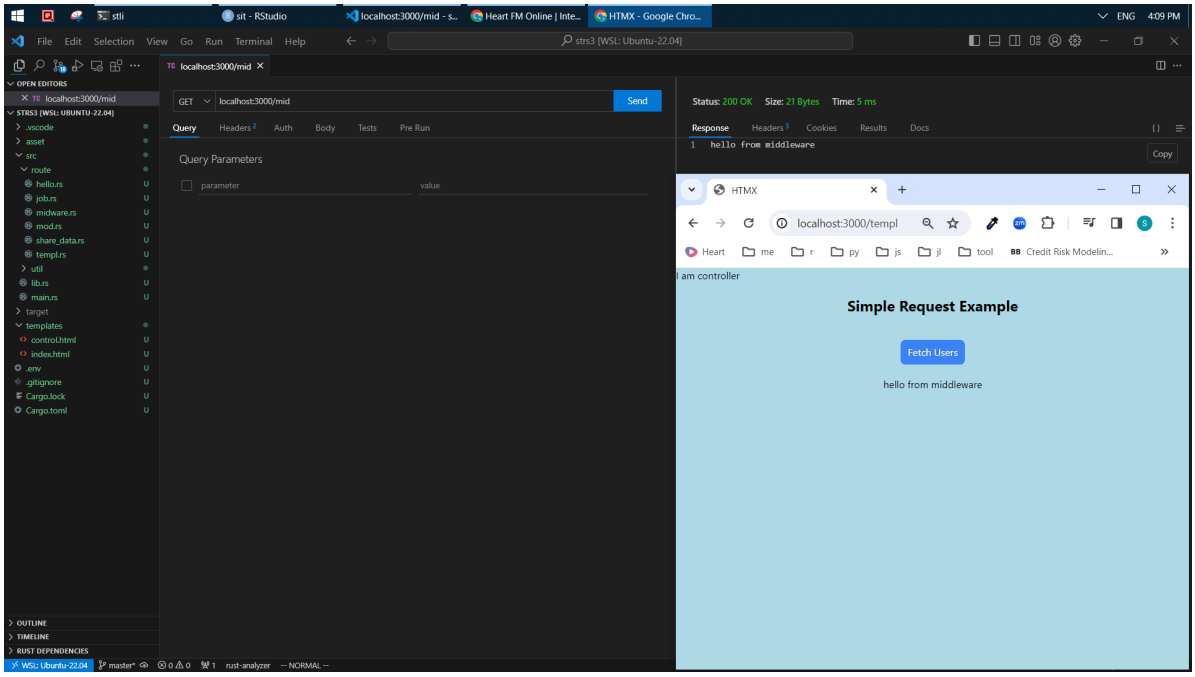

Figure 3.4: Template

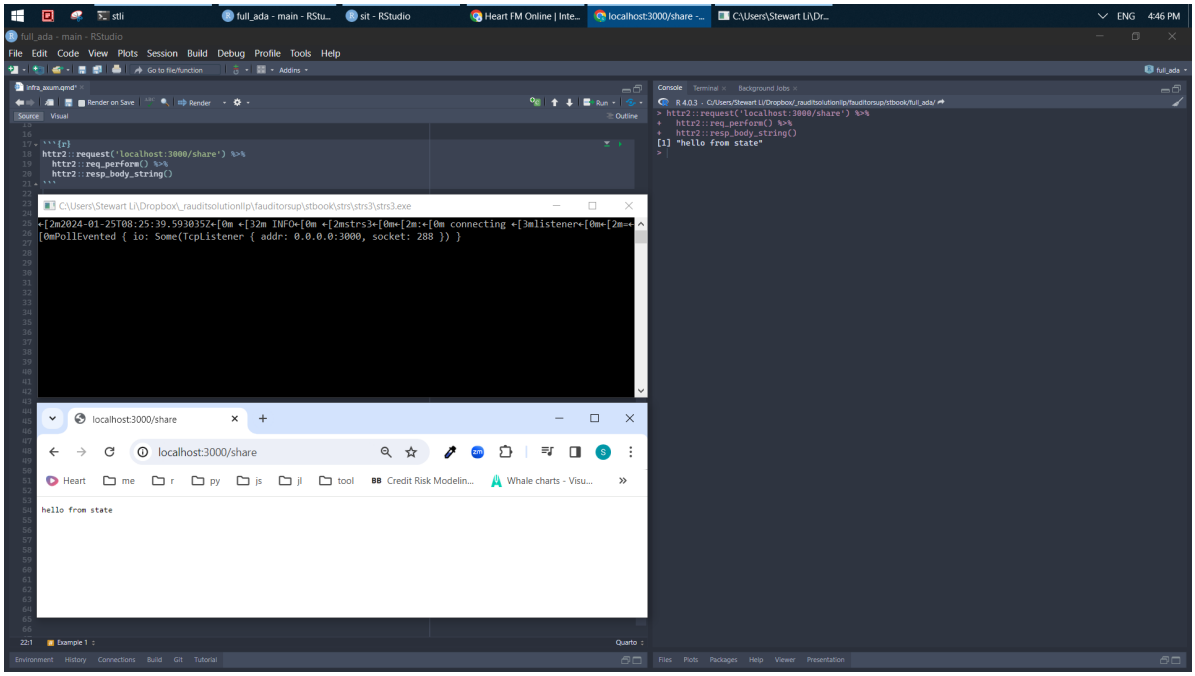

Figure 3.5: R client

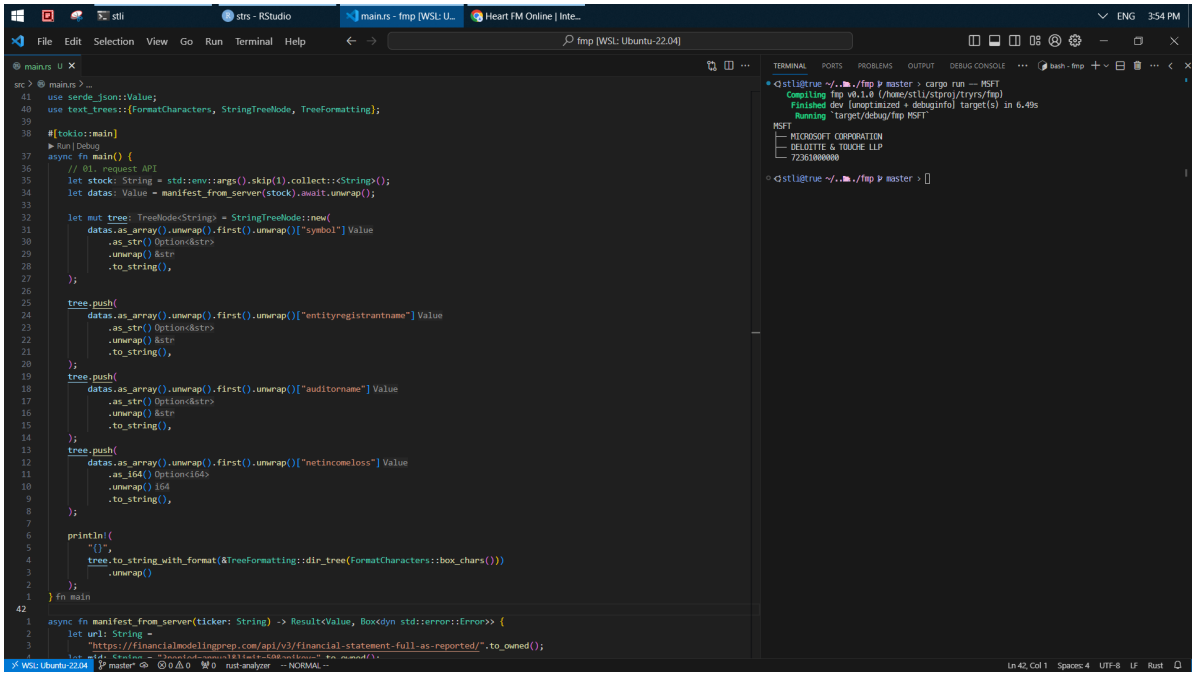

Figure 3.6: Request CLI 1

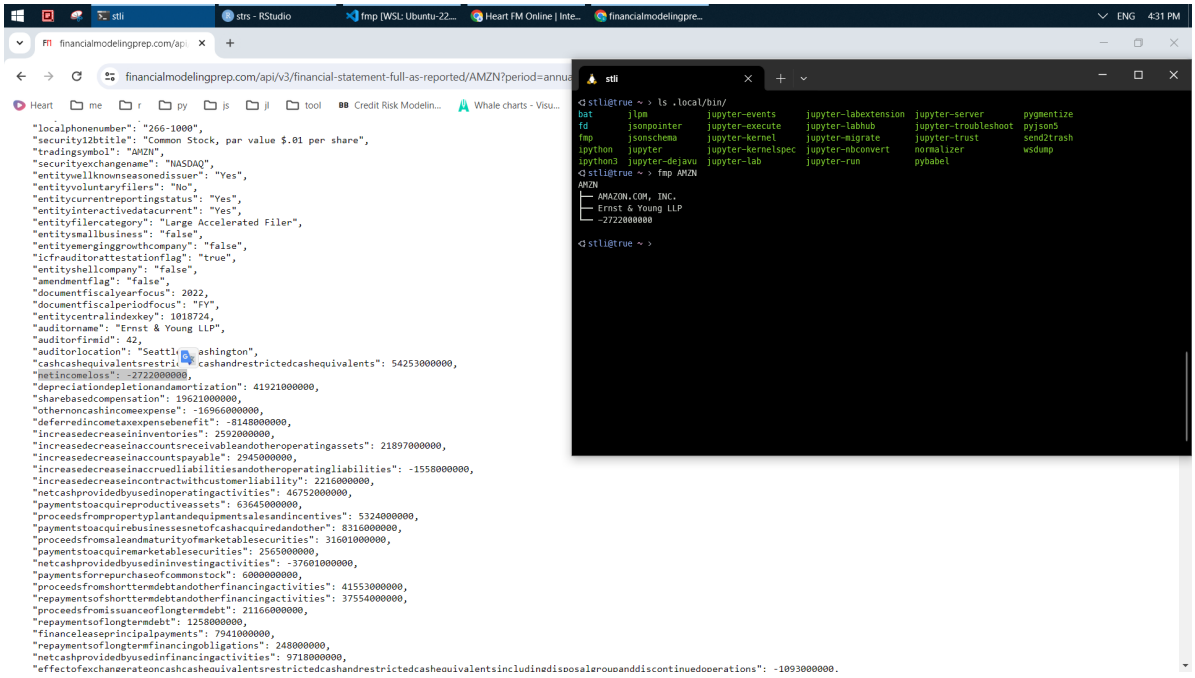

Figure 3.7: Request CLI 2

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`, `pandera`, `greate expectation`, `pydantic`,
4. Data visualization[1]:
- `grid`, `patchwork`, `ggfx`, `ggtext`, `showtext`,
- `ragg`, `scales`, `formattable`, `sparkline`,
- `gghighlight`, `ggforce`,
- `imager`, `imagerExtra`, `ggimage`, `ggpubr`,
- `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`,

[1] `ggplot2` (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, extendr, pyo3, bindgen,

4 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`.

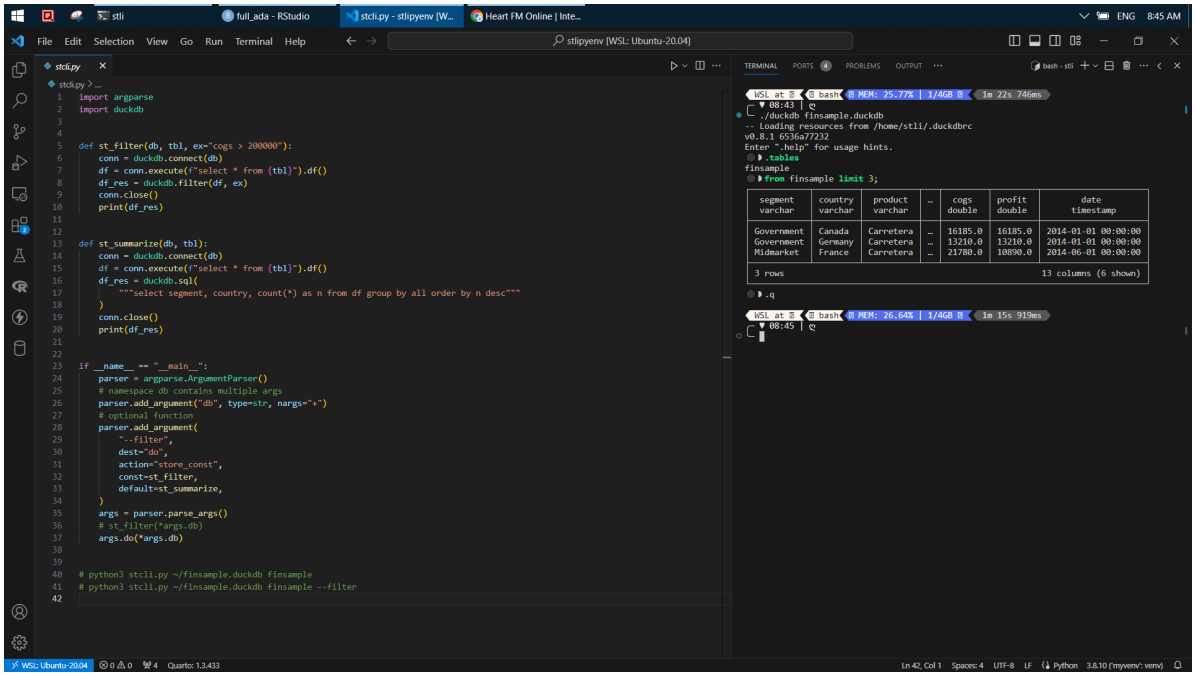

Figure 4.1: CLI - argparse 1

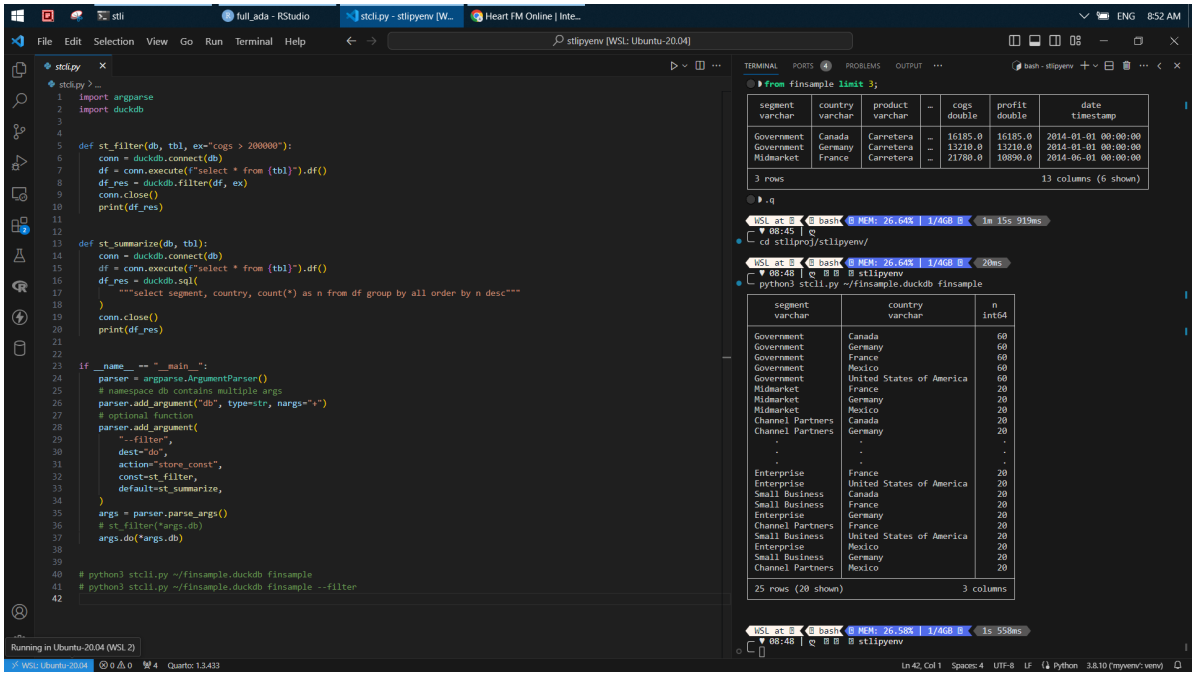

Figure 4.2: CLI - argparse 2

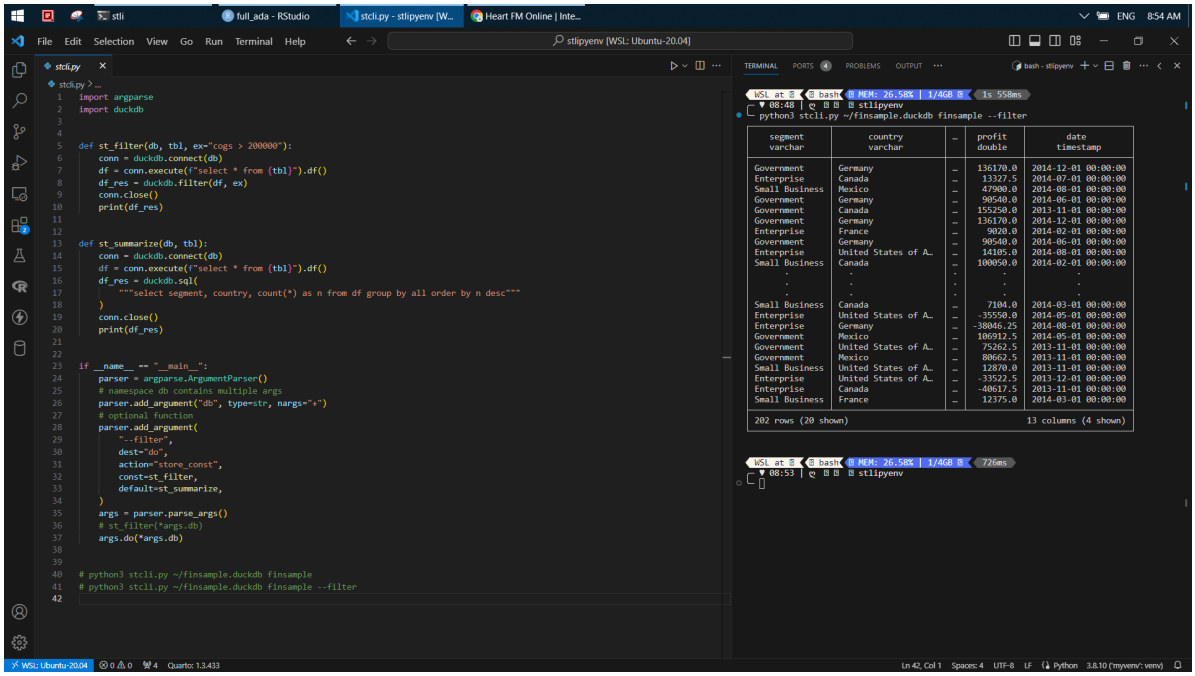

Figure 4.3: CLI - argparse 3

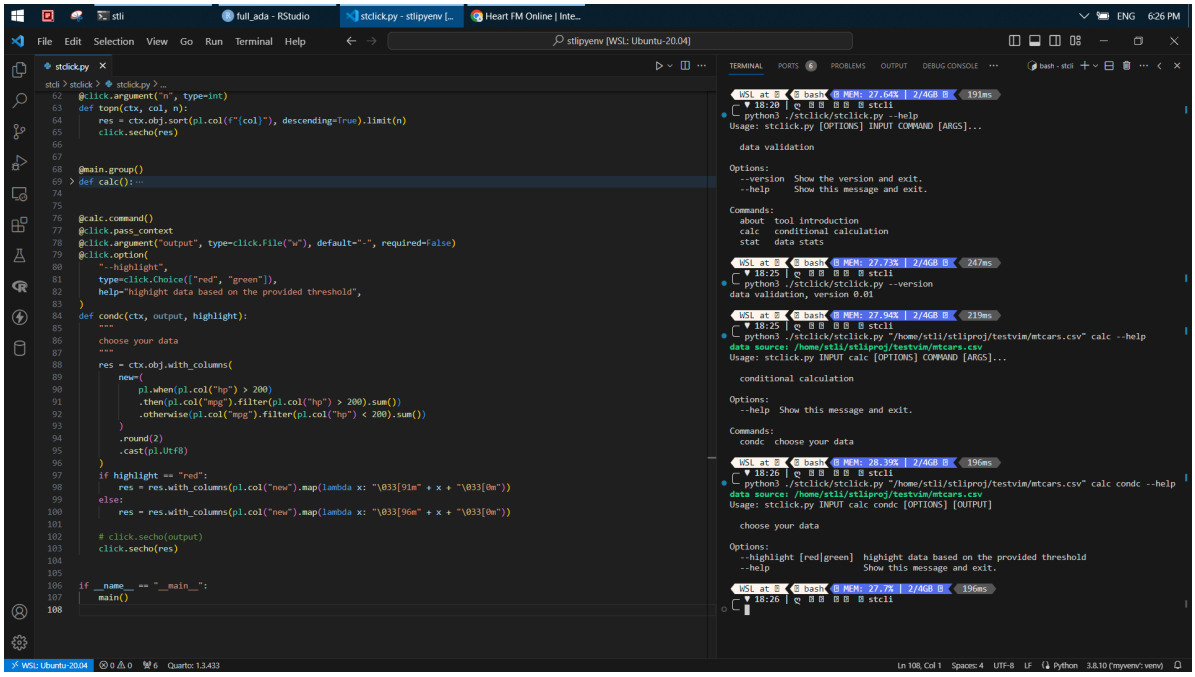

Figure 4.4: CLI - click 1

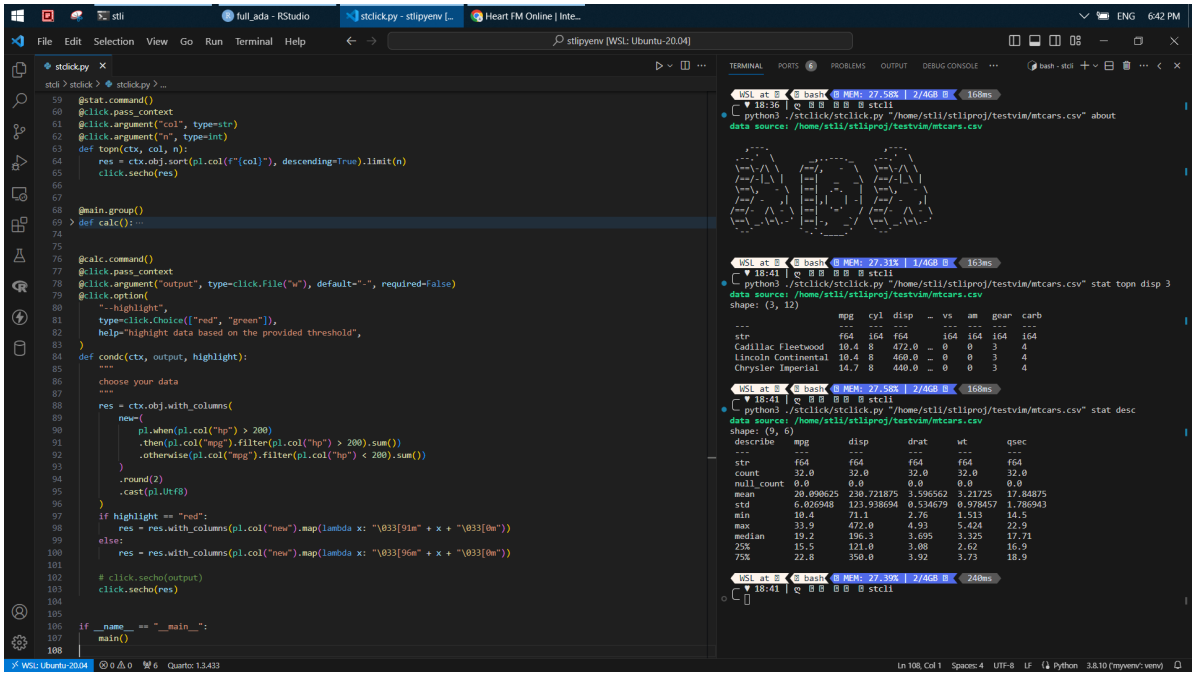

Figure 4.5: CLI - click 2

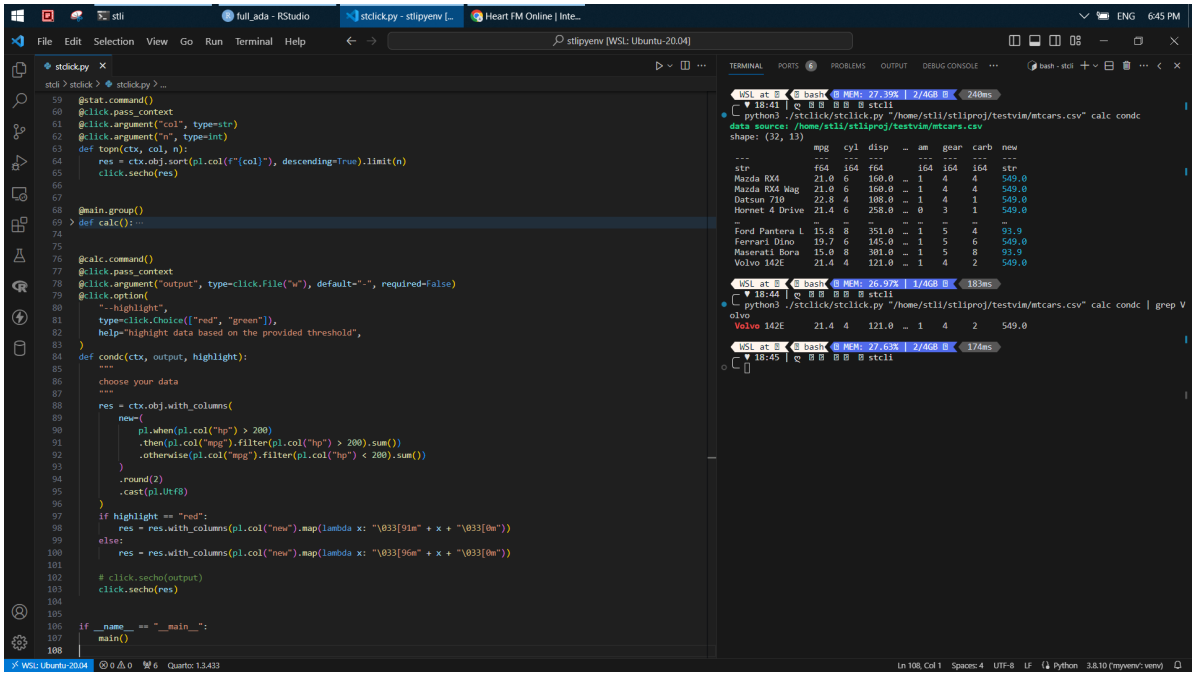

Figure 4.6: CLI - click 3

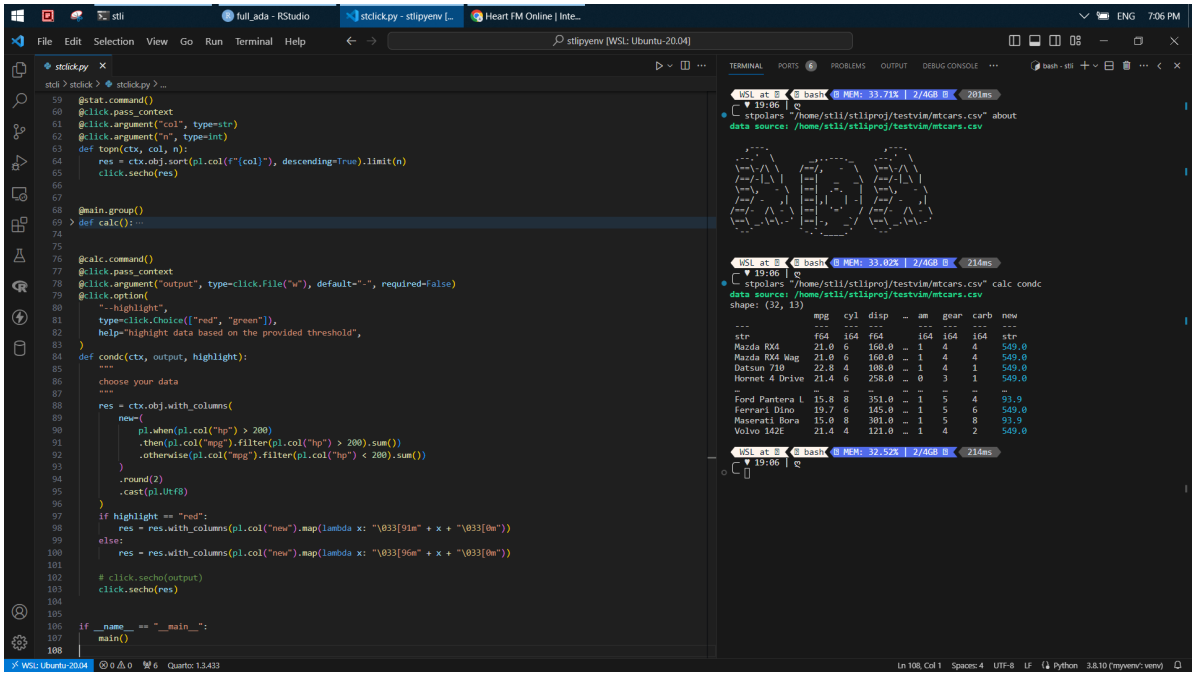

Figure 4.7: CLI - click 4

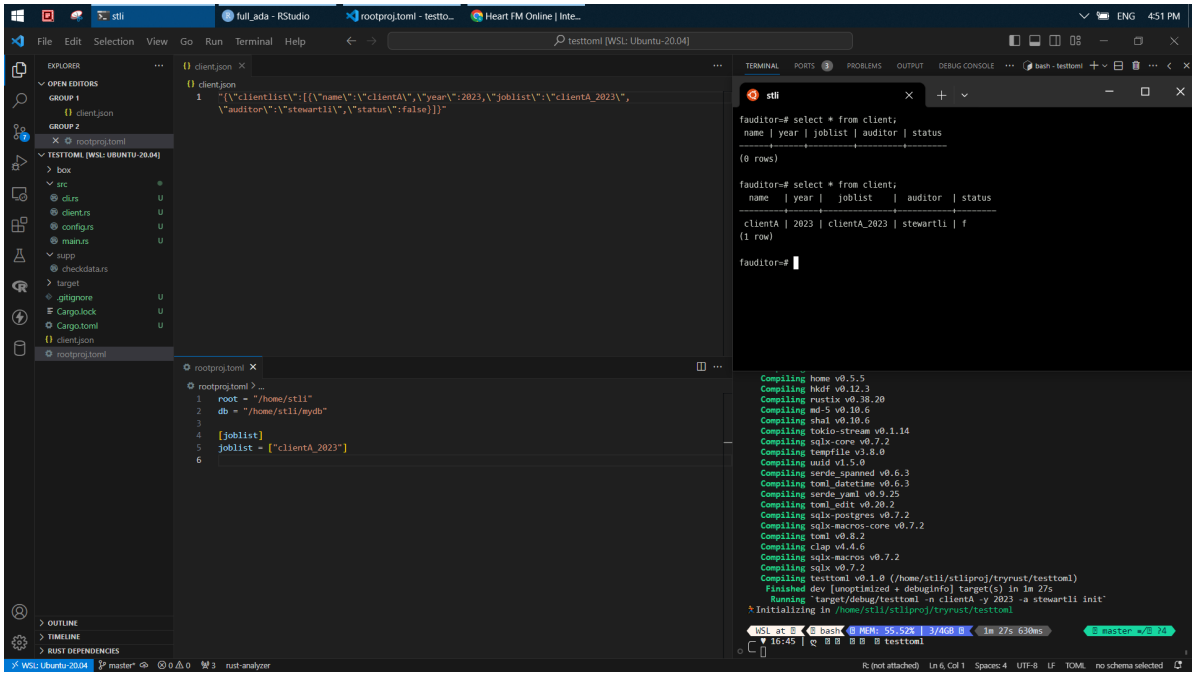

Figure 4.8: CLI - clap 1

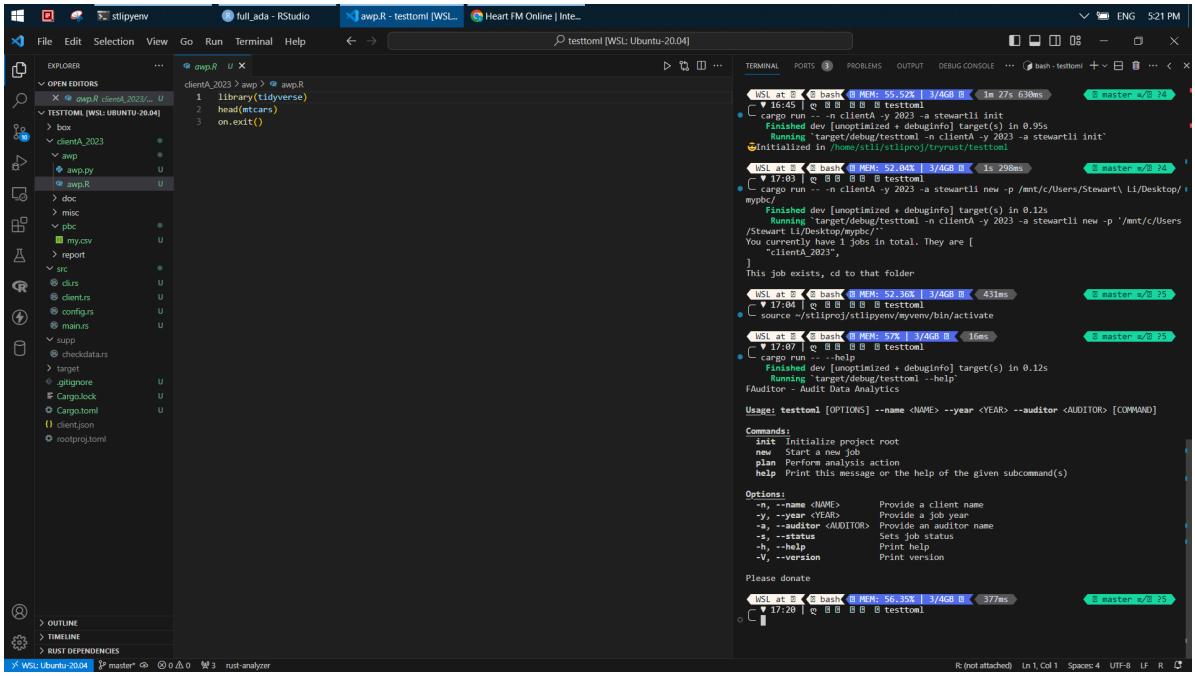

Figure 4.9: CLI - clap 2

5 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.

5.1 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)
```

5.2 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))
```

5.3 Validate

Validate data if it is received from other team members.

```r
# 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))
```

5.4 Munging

Ask reasonable questions via slice dice.

```r
# window operation: lag, first, nth,
df_clean %>%
  arrange(invoice_date) %>%
  group_by(country_code) %>%
  mutate(increase = invoice_amount - dplyr::lag(invoice_amount, default = 0),
         indcator = 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.

```r
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'))
```

5.5 EDA

Focus on a handful of variables after dropped others.

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

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

# descriptive stats
df %>%
```

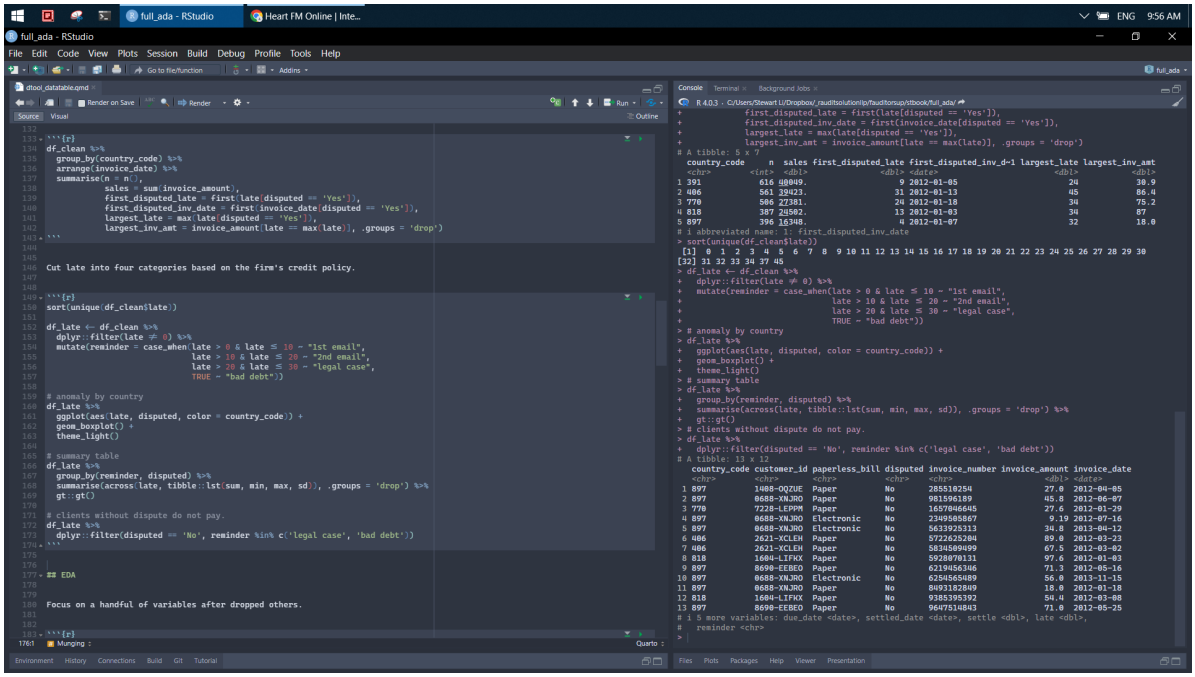

Figure 5.1: Data munging

```
  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()
```

5.6 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()
```

5.7 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())
```

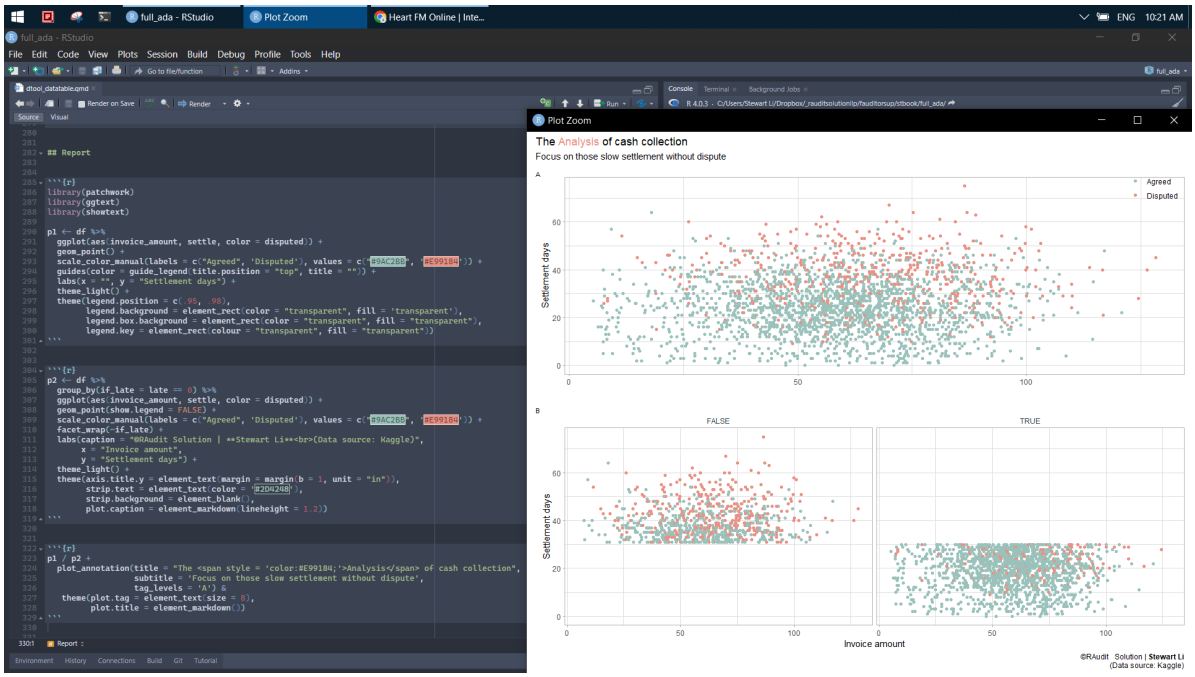

Figure 5.2: Combined plot

6 Example 2

[To my understanding] Audit includes **tools and work** stipulated by Standards. Audit Data Analytics (ADA) replaces excel-related tools with R/Python to improve efficiency/effectiveness. It does not necessarily reduce audit work required by ISCA. The following example is to audit expense claim based on data from payroll, hr, and finance departments, which demonstrates ADA is a vital move for auditors from all possible perspectives.

Compared to excel-related tools, it could be easily used to test audit assertions (e.g., occurrence, existence, completeness, cut-off, valuation, classification) after reconciled in terms of P2P, O2C, Payroll, R2R, GL.
1. benefit: version control `diff`, lightweight `size`, powerful `1m` rows, automation `script`.
2. pattern recognition: spot deviation and inconsistency.

It also addresses common mistakes throughout the audit process. For instance,
1. version control: which version of PBC data is the latest?
2. reproducible: my result is different from yours after rerun.
3. report: check if number in working papers tally to those in financial statement.
4. automation: roll out audit work next year by copy+paste.

6.1 Cleaning

```
exp_claim_raw <- readxl::read_excel("isca_cpe_2023/1. Anomalies in Payroll data.xlsx",
                                    sheet = 1,
                                    range = "A1:G33") %>%
  janitor::clean_names()

hr_data_raw <- readxl::read_excel("isca_cpe_2023/1. Anomalies in Payroll data.xlsx",
                                  sheet = 2) %>%
  janitor::clean_names()

pay_data_raw <- readxl::read_excel("isca_cpe_2023/1. Anomalies in Payroll data.xlsx",
                                   sheet = 3,
                                   skip = 2, range = "A3:D25") %>%
  janitor::clean_names()
```

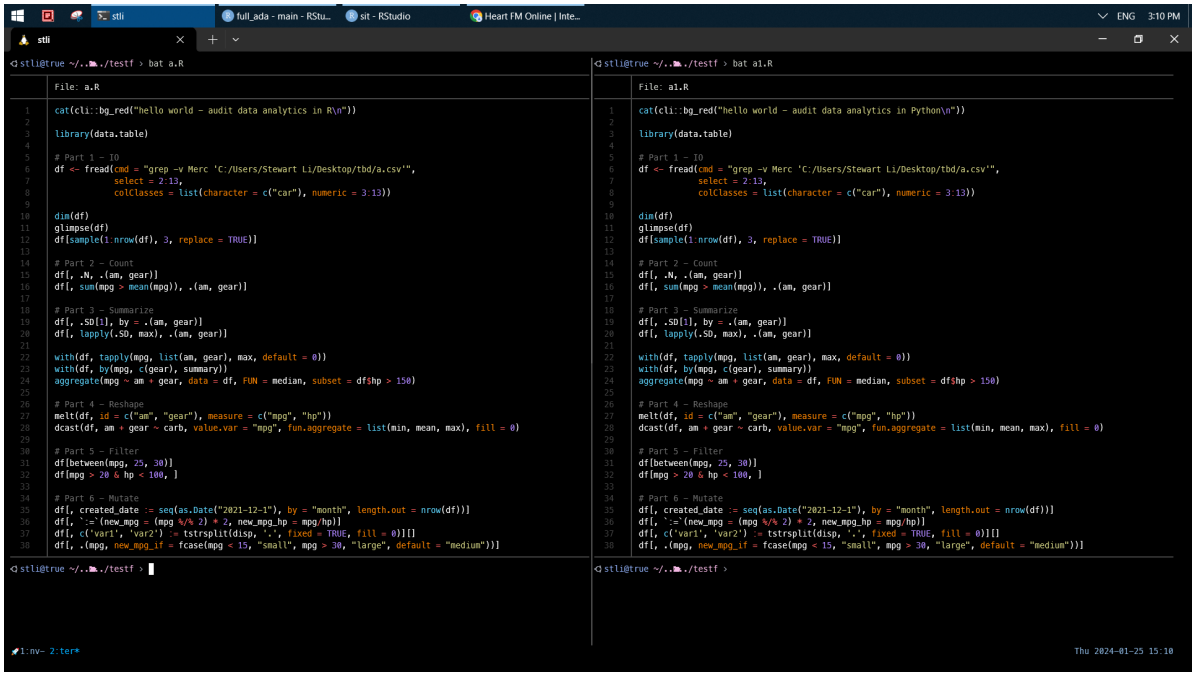

Figure 6.1: Diff 1

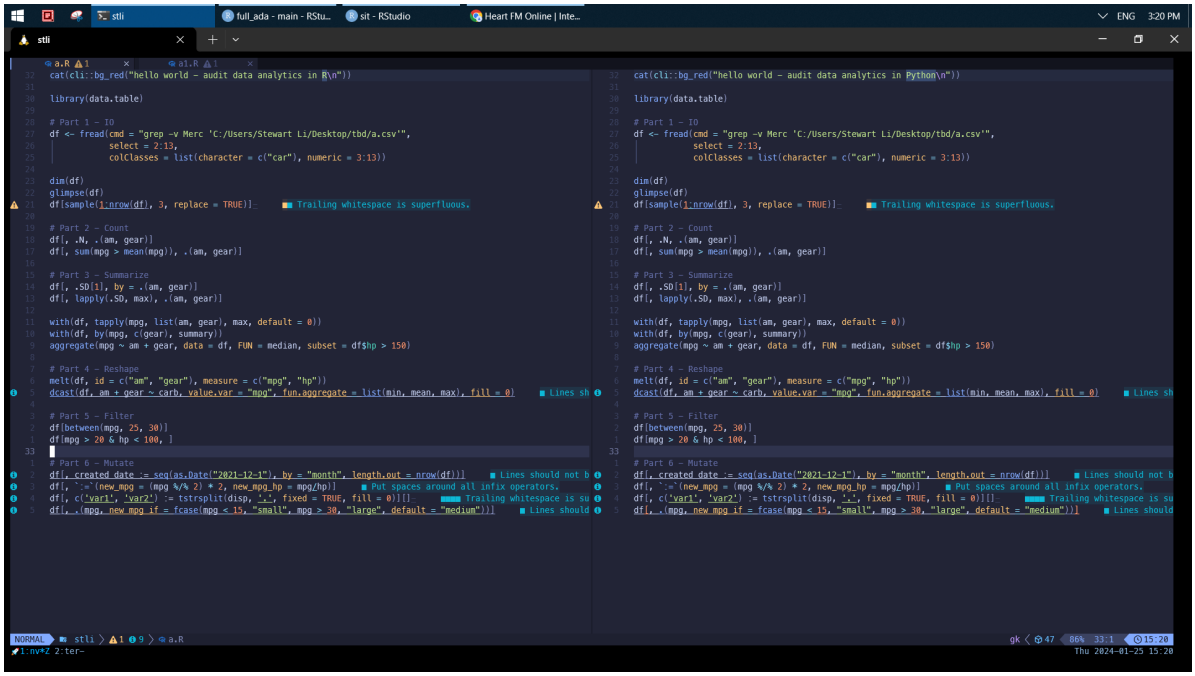

Figure 6.2: Diff 2

```
df_comb <- exp_claim_raw %>%
  full_join(hr_data_raw, by = c('staff_id' = 'staff_id')) %>%
  left_join(pay_data_raw, by = c('staff_id' = 'staff_id'))
```

```
df_clean <- df_comb %>%
  mutate(across(contains("date"), lubridate::dmy)) %>%
  mutate(on_leave = lubridate::dmy(on_leave)) %>%
  mutate(staff_name = coalesce(staff_name, name.x))

# check if amount is correct
sum(df_clean$amount_s.x, na.rm = TRUE)

df_clean %>%
  distinct(staff_id, amount_s.y) %>%
  summarise(app_c = sum(amount_s.y, na.rm = TRUE))
```

```
sheets <- list("comb" = df_comb, "clean" = df_clean)
writexl::write_xlsx(sheets, here::here(paste0('audit_sit/audit_payroll', Sys.Date(), '.xls
openxlsx::openXL(here::here("audit_sit/audit_payroll2023-12-22.xlsx"))

df_clean <- readxl::read_excel(here::here("audit_sit/audit_payroll2023-12-22.xlsx")) %>%
  mutate(across(c(contains("date"), on_leave), lubridate::dmy))
```

6.2 Procedure

```
# cross check payroll amount against finance amount
df_clean %>%
  group_by(staff_id, staff_name) %>%
  summarise(amt_exp = sum(amount_s.x),
            amt_paid = sum(amount_s.y) / n(),
            amt_diff = amt_exp - amt_paid,
            .groups = 'drop')
```

```
# compare date to ensure no claim happens before incurred or after resigned
df_clean %>%
  dplyr::filter(claim_date > expense_date)
```

```
df_clean %>%
  dplyr::filter(claim_date > last_date | claim_date == on_leave)
```

```
# identify multiple claims for the same expense
df_clean %>%
  group_by(staff_id, staff_name, purpose, amount_s.x) %>%
  dplyr::filter(n() > 1)
```

```
# ensure staff name and their bank account number updated timely
df_clean %>%
  dplyr::filter(!is.na(edits_to_hr_data),
                bank_account_no.x == bank_account_no.y)

df_clean %>%
  dplyr::filter(name.x != name.y)
```

```
# produces audit working paper
library(pointblank)

ag <- df_clean %>%
  create_agent(label = "A very *simple* example.", tbl_name = "payroll") %>%
  col_vals_between(columns = claim_date, left = vars(expense_date), right = vars(last_date
  interrogate()

ag
```

6.3 Enhanced

```
df_clean %>%
  count(staff_name, sort = TRUE)

df_clean %>%
  dplyr::filter(grepl("\\d+?", purpose)) %>%
  mutate(purpose = gsub("\\d+?", "", purpose)) %>%
  mutate(across(where(is.character), ~na_if(., "AB99"))) %>%
  mutate(staff_id = replace_na(staff_id, 0))
```

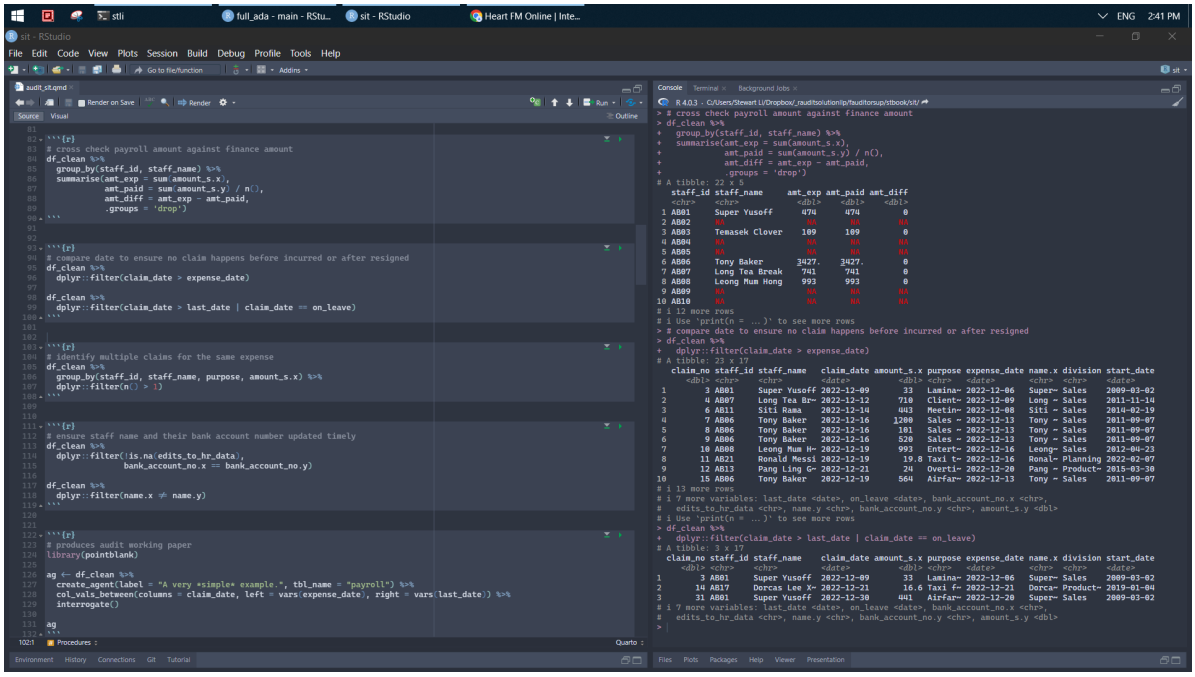

Figure 6.3: Audit Procedure 1

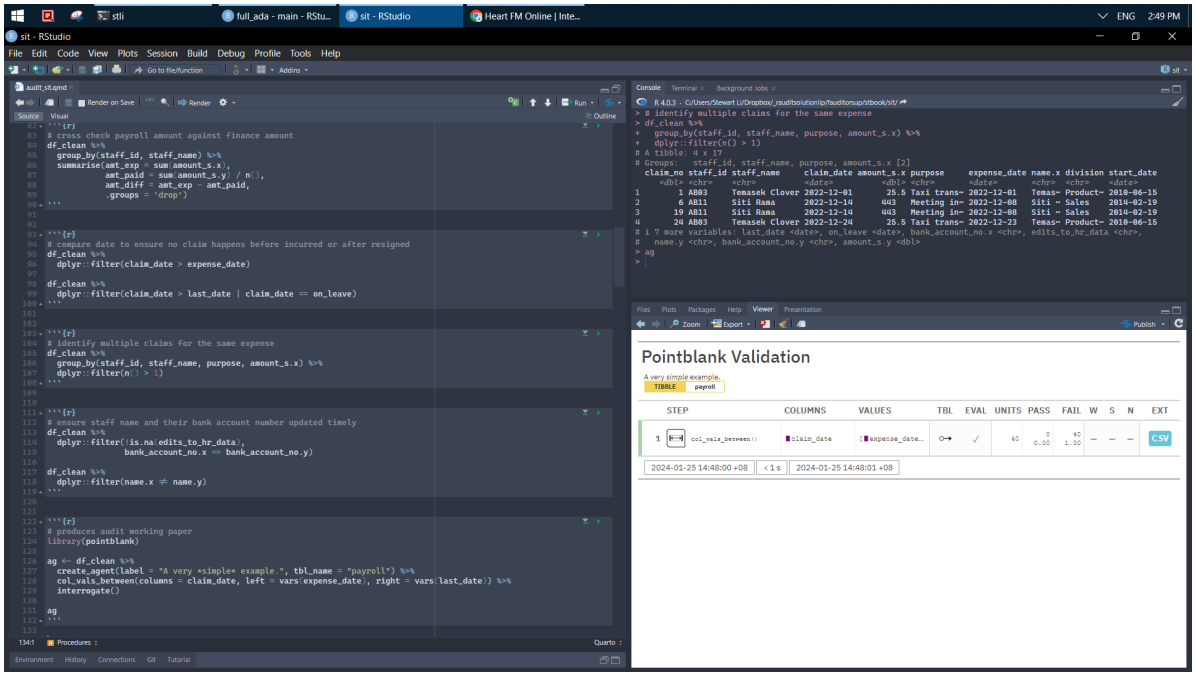

Figure 6.4: Audit Procedure 2

```r
df_clean %>%
  select(contains("date"), purpose) %>%
  mutate(if_taxi = case_when(str_detect(purpose, "Taxi") ~ "taxi",
                             TRUE ~ "other"),
         total_date = lubridate::floor_date(claim_date, "week"),
         first_date = first(total_date)) %>%
  slice_max(order_by = claim_date, n = 3)

df_clean %>%
  dplyr::filter(!is.na(amount_s.x)) %>%
  mutate(new = (amount_s.x %/% 100) * 100) %>%
  group_by(new, amount_s.x > 300) %>%
  summarise(new1 = mean(amount_s.x), .groups = 'drop')
```

```r
df_clean %>%
  dplyr::filter(!is.na(staff_name)) %>%
  group_nest(staff_id, staff_name) %>%
  mutate(new = map(data, ~pluck(.x, 4))) %>%
  mutate(new1 = map(new, ~paste(.x, collapse = '|'))) %>%
  select(-data, -new) %>%
  unnest(new1)

df_clean %>%
  dplyr::filter(!is.na(staff_name)) %>%
  select(staff_id, staff_name, purpose) %>%
  summarise(new1 = paste(purpose, collapse = '|'), .by = c(staff_id, staff_name))
```

```r
df_clean %>%
  select(staff_id, staff_name, division, purpose, amount_s.x) %>%
  dplyr::filter(!is.na(purpose)) %>%
  separate(purpose, into = c("type", "info"),
           extra = 'merge', remove = FALSE, fill = 'right') %>%
  group_by(division, type) %>%
  summarise(n = n(),
            amt_type = sum(amount_s.x), .groups = 'drop') %>%
  arrange(-amt_type)
```

```r
library(lubridate)

df_clean %>%
```

```
pivot_longer(cols = where(is.Date),
             names_to = 'activity_date',
             values_to = 'detail_date',
             names_pattern = "(.*)_.*",
             names_transform = list(activity_date = toupper))
```

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.