

Data Analytics Engineering

For Accountants and Auditors

Stewart Li

2024-01-29

Table of contents

Preface	3
I Infrastructure	4
1 Local	6
2 ELT	20
3 Example 1	22
II Data tools	26
4 Polars	30
5 Analysis	35
5.1 IO	35
5.2 Cleaning	36
5.3 Validate	36
5.4 Munging	37
5.5 EDA	38
5.6 Model	40
5.7 Report	41
6 Example 2	43
6.1 Cleaning	43
6.2 Procedure	45
6.3 Enhanced	46
References	50

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 `ws12`, and Linux distribution systems. Edit terminal theme/font, dotfiles of Bash/Tmux/Vim, and env variables. Install Git/GitBash and Docker/Podman if needed.

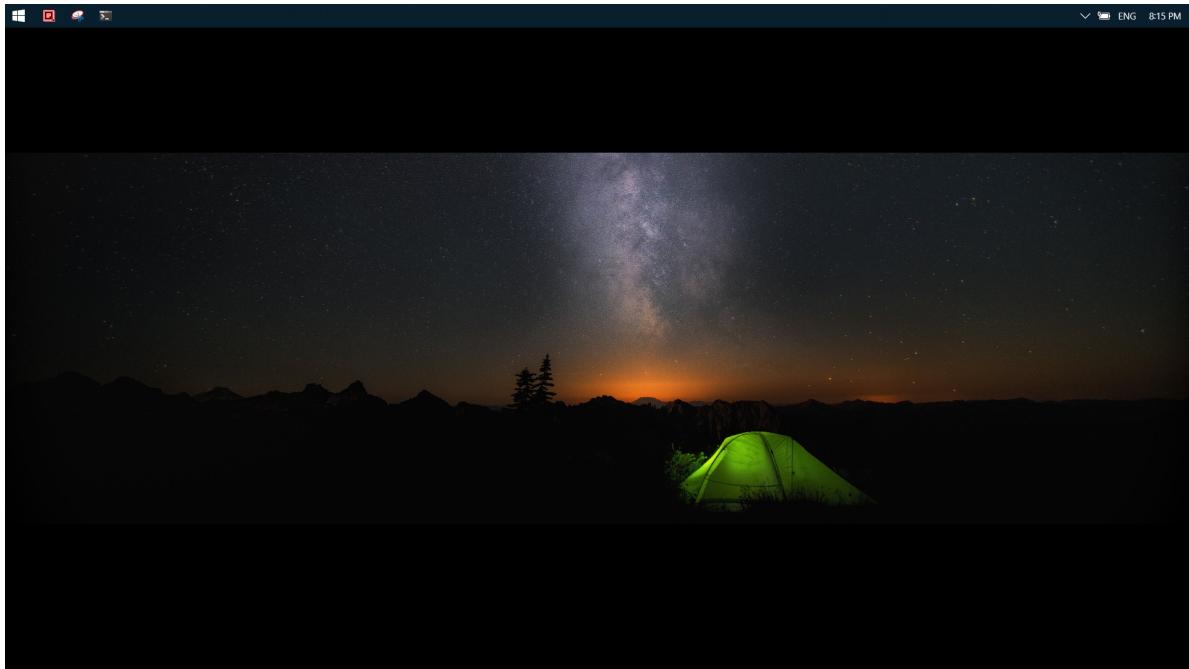


Figure 1.1: Desktop

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.

```

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

```

1 library(data.table)
2
3 head(mtcars)
4 #> #   mpg cyl disp hp drat wt qsec vs am gear carb
5 #> #   Mazda RX4   21.0   6 160 108 3.90 2.620 16.46 0 1 4 4
6 #> #   Mazda RX4 Wag 21.0   6 160 110 3.90 2.675 17.02 0 1 4 4
7 #> #   Datsun 710  22.8   4 108 93 3.85 2.320 18.61 1 1 4 1
8 #> #   Hornet 4 Drive 21.4   6 258 110 3.08 3.215 19.44 1 0 3 1
9 #> #   Hornet Sportabout 18.7   8 360 175 3.15 3.440 17.02 0 0 3 2
10 #> #   Valiant 18.1   6 225 105 2.76 3.460 20.22 1 0 3 1
11
12 mydf <- mean(mtcars[mpg]) > paste0()
13 print(mydf)
14
15 cat('Hello world from R')
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
599
600
601
602
603
604
605
606
607
608
609
609
610
611
612
613
614
615
616
617
618
619
619
620
621
622
623
624
625
626
627
628
629
629
630
631
632
633
634
635
636
637
638
639
639
640
641
642
643
644
645
646
647
648
649
649
650
651
652
653
654
655
656
657
658
659
659
660
661
662
663
664
665
666
667
668
669
669
670
671
672
673
674
675
676
677
678
679
679
680
681
682
683
684
685
686
687
687
688
689
689
690
691
692
693
694
695
696
697
697
698
699
699
700
701
702
703
704
705
706
707
708
709
709
710
711
712
713
714
715
716
717
717
718
719
719
720
721
722
723
724
725
726
727
727
728
729
729
730
731
732
733
734
735
736
737
737
738
739
739
740
741
742
743
744
745
746
746
747
748
748
749
749
750
751
752
753
754
755
756
757
757
758
759
759
760
761
762
763
764
765
766
766
767
768
768
769
769
770
771
772
773
774
775
776
777
777
778
779
779
780
781
782
783
784
785
786
787
787
788
789
789
790
791
792
793
794
795
796
797
797
798
799
799
800
801
802
803
804
805
806
807
808
809
809
810
811
812
813
814
815
816
817
817
818
819
819
820
821
822
823
824
825
826
827
827
828
829
829
830
831
832
833
834
835
836
837
837
838
839
839
840
841
842
843
844
845
846
846
847
848
848
849
849
850
851
852
853
854
855
856
857
857
858
859
859
860
861
862
863
864
865
866
866
867
868
868
869
869
870
871
872
873
874
875
876
877
877
878
879
879
880
881
882
883
884
885
886
887
887
888
889
889
890
891
892
893
894
895
896
897
897
898
899
899
900
901
902
903
904
905
906
907
907
908
909
909
910
911
912
913
914
915
916
916
917
918
918
919
919
920
921
922
923
924
925
926
927
927
928
929
929
930
931
932
933
934
935
936
937
937
938
939
939
940
941
942
943
944
945
946
946
947
948
948
949
949
950
951
952
953
954
955
956
957
957
958
959
959
960
961
962
963
964
965
966
966
967
968
968
969
969
970
971
972
973
974
975
976
977
977
978
979
979
980
981
982
983
984
985
986
987
987
988
989
989
990
991
992
993
994
995
996
997
998
999
999
1000
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1009
1010
1011
1012
1013
1014
1015
1016
1016
1017
1018
1018
1019
1019
1020
1021
1022
1023
1024
1025
1026
1026
1027
1028
1028
1029
1029
1030
1031
1032
1033
1034
1035
1036
1036
1037
1038
1038
1039
1039
1040
1041
1042
1043
1044
1045
1046
1046
1047
1048
1048
1049
1049
1050
1051
1052
1053
1054
1055
1056
1056
1057
1058
1058
1059
1059
1060
1061
1062
1063
1064
1065
1066
1066
1067
1068
1068
1069
1069
1070
1071
1072
1073
1074
1075
1076
1076
1077
1078
1078
1079
1079
1080
1081
1082
1083
1084
1085
1086
1086
1087
1088
1088
1089
1089
1090
1091
1092
1093
1094
1095
1095
1096
1097
1097
1098
1098
1099
1099
1100
1101
1102
1103
1104
1105
1105
1106
1107
1107
1108
1108
1109
1109
1110
1111
1112
1112
1113
1114
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1120
1121
1121
1122
1122
1123
1123
1124
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1130
1131
1131
1132
1132
1133
1133
1134
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1140
1141
1141
1142
1142
1143
1143
1144
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1150
1151
1151
1152
1152
1153
1153
1154
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1160
1161
1161
1162
1162
1163
1163
1164
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1170
1171
1171
1172
1172
1173
1173
1174
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1180
1181
1181
1182
1182
1183
1183
1184
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1190
1191
1191
1192
1192
1193
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1200
1201
1201
1202
1202
1203
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1210
1211
1211
1212
1212
1213
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1220
1221
1221
1222
1222
1223
1223
1224
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1230
1231
1231
1232
1232
1233
1233
1234
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1240
1241
1241
1242
1242
1243
1243
1244
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1250
1251
1251
1252
1252
1253
1253
1254
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1260
1261
1261
1262
1262
1263
1263
1264
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1270
1271
1271
1272
1272
1273
1273
1274
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1280
1281
1281
1282
1282
1283
1283
1284
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1290
1291
1291
1292
1292
1293
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1300
1301
1301
1302
1302
1303
1303
1304
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1310
1311
1311
1312
1312
1313
1313
1314
1314
1315
1315
1316
1316
1317
1317
1318
1318
1319
1319
1320
1320
1321
1321
1322
1322
1323
1323
1324
1324
1325
1325
1326
1326
1327
1327
1328
1328
1329
1329
1330
1330
1331
1331
1332
1332
1333
1333
1334
1334
1335
1335
1336
1336
1337
1337
1338
1338
1339
1339
1340
1340
1341
1341
1342
1342
1343
1343
1344
1344
1345
1345
1346
1346
1347
1347
1348
1348
1349
1349
1350
1350
1351
1351
1352
1352
1353
1353
1354
1354
1355
1355
1356
1356
1357
1357
1358
1358
1359
1359
1360
1360
1361
1361
1362
1362
1363
1363
1364
1364
1365
1365
1366
1366
1367
1367
1368
1368
1369
1369
1370
1370
1371
1371
1372
1372
1373
1373
1374
1374
1375
1375
1376
1376
1377
1377
1378
1378
1379
1379
1380
1380
1381
1381
1382
1382
1383
1383
1384
1384
1385
1385
1386
1386
1387
1387
1388
1388
1389
1389
1390
1390
1391
1391
1392
1392
1393
1393
1394
1394
1395
1395
1396
1396
1397
1397
1398
1398
1399
1399
1400
1400
1401
1401
1402
1402
1403
1403
1404
1404
1405
1405
1406
1406
1407
1407
1408
1408
1409
1409
1410
1410
1411
1411
1412
1412
1413
1413
1414
1414
1415
1415
1416
1416
1417
1417
1418
1418
1419
1419
1420
1420
1421
1421
1422
1422
1423
1423
1424
1424
1425
1425
1426
1426
1427
1427
1428
1428
1429
1429
1430
1430
1431
1431
1432
1432
1433
1433
1434
1434
1435
1435
1436
1436
1437
1437
1438
1438
1439
1439
1440
1440
1441
1441
1442
1442
1443
1443
1444
1444
1445
1445
1446
1446
1447
1447
1448
1448
1449
1449
1450
1450
1451
1451
1452
1452
1453
1453
1454
1454
1455
1455
1456
1456
1457
1457
1458
1458
1459
1459
1460
1460
1461
1461
1462
1462
1463
1463
1464
1464
1465
1465
1466
1466
1467
1467
1468
1468
1469
1469
1470
1470
1471
1471
1472
1472
1473
1473
1474
1474
1475
1475
1476
1476
1477
1477
1478
1478
1479
1479
1480
1480
1481
1481
1482
1482
1483
1483
1484
1484
1485
1485
1486
1486
1487
1487
1488
1488
1489
1489
1490
1490
1491
1491
1492
1492
1493
1493
1494
1494
1495
1495
1496
1496
1497
1497
1498
1498
1499
1499
1500
1500
1501
1501
1502
1502
1503
1503
1504
1504
1505
1505
1506
1506
1507
1507
1508
1508
1509
1509
1510
1510
1511
1511
1512
1512
1513
1513
1514
1514
1515
1515
1516
1516
1517
1517
1518
1518
1519
1519
1520
1520
1521
1521
1522
1522
1523
1523
1524
1524
1525
1525
1526
1526
1527
1527
1528
1528
1529
1529
1530
1530
1531
1531
1532
1532
1533
1533
1534
1534
1535
1535
1536
1536
1537
1537
1538
1538
1539
1539
1540
1540
1541
1541
1542
1542
1543
1543
1544
1544
1545
1545
1546
1546
1547
1547
1548
1548
1549
1549
1550
1550
1551
1551
1552
1552
1553
1553
1554
1554
1555
1555
1556
1556
1557
1557
1558
1558
1559
1559
1560
1560
1561
1561
1562
1562
1563
1563
1564
1564
1565
1565
1566
1566
1567
1567
1568
1568
1569
1569
1570
1570
1571
1571
1572
1572
1573
1573
1574
1574
1575
1575
1576
1576
1577
1577
1578
1578
1579
1579
1580
1580
1581
1581
1582
1582
1583
1583
1584
1584
1585
1585
1586
1586
1587
1587
1588
1588
1589
1589
1590
1590
1591
1591
1592
1592
1593
1593
1594
1594
1595
1595
1596
1596
1597
1597
1598
1598
1599
1599
1600
1600
1601
1601
1602
1602
1603
1603
1604
1604
1605
1605
1606
1606
1607
1607
1608
1608
1609
1609
1610
1610
1611
1611
1612
1612
1613
1613
1614
1614
1615
1615
1616
1616
1617
1617
1618
1618
1619
1619
1620
1620
1621
1621
1622
1622
1623
1623
1624
1624
1625
1625
1626
1626
1627
1627
1628
1628
1629
1629
1630
1630
1631
1631
1632
1632
1633
1633
1634
1634
1635
1635
1636
1636
1637
1637
1638
1638
1639
1639
1640
1640
1641
1641
1642
1642
1643
1643
1644
1644
1645
1645
1646
1646
1647
1647
1648
1648
1649
1649
1650
1650
1651
1651
1652
1652
1653
1653
1654
1654
1655
1655
1656
1656
1657
1657
1658
1658
1659
1659
1660
1660
1661
1661
1662
1662
1663
1663
1664
1664
1665
1665
1666
1666
1667
1667
1668
1668
1669
1669
1670
1670
1671
1671
1672
1672
1673
1673
1674
1674
1675
1675
1676
1676
1677
1677
1678
1678
1679
1679
1680
1680
1681
1681
1682
1682
1683
1683
1684
1684
1685
1685
1686
1686
1687
1687
1688
1688
1689
1689
1690
1690
1691
1691
1692
1692
1693
1693
1694
1694
1695
1695
1696
1696
1697
1697
1698
1698
1699
1699
1700
1700
1701
1701
1702
1702
1703
1703
1704
1704
1705
1705
1706
1706
1707
1707
1708
1708
1709
17
```

WSL at < bash * MEM: 9.5% | 1/4GB * 8ms
~ 15:25 | q ➔ testvnm

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,
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

```
> write.csv(mtcars, "mtcars.csv")
>
```

WSL at < bash * MEM: 10.6% | 1/4GB * 9ms
~ 15:29 | q ➔ testvnm

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	32.000000	32.000000
mean	20.898625	6.187500	236.721875	146.597500	...	0.477500	0.406250	3.737500	2.8125
std	6.076942	1.785922	123.938694	66.592688	...	0.504016	0.499391	0.737804	1.6357
min	10.400000	4.000000	71.000000	57.000000	...	0.000000	0.000000	3.000000	1.0000
25%	15.425000	4.000000	120.825000	96.500000	...	0.000000	0.000000	3.000000	2.0000
50%	19.200000	6.000000	196.300000	123.000000	...	0.000000	0.000000	4.000000	2.0000
75%	22.800000	6.000000	326.000000	186.000000	...	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
~ 15:25 | q ➔ testvnm

ls

a.R b.py mtcars.csv

WSL at < bash * MEM: 10.52% | 1/4GB * 11ms
~ 15:27 | q ➔ testvnm

WSL at < bash * MEM: 12.3% | 1/4GB * 9ms
~ 15:32 | q ➔

/duckdb

— Loading resources from /home/stli/.duckdbrc

v0.8.1 6536a77232

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("stripes/testvnm/mtcars.csv") LIMIT 3;
```

column0	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
Mazda RX4	21.0	6	160.0	110	3.9	2.62	16.46	0	1	4	4
Mazda RX4 Wag	21.0	6	160.0	110	3.9	2.875	17.02	0	1	4	4
Datsun 710	22.8	4	108.0	93	3.85	2.32	18.61	1	1	4	1

Figure 1.6: R, Python, DuckDB

The screenshot shows a Windows desktop environment with several open windows. In the center is a RStudio interface displaying an R script named 'testvwm.R'. The script includes code to load the 'mtcars' dataset, calculate mean mpg, print the results, and then display the 'head' of the 'mtcars' dataset. To the right of the RStudio window is a terminal window titled 'full_ada - RStudio' showing the output of the R script. The terminal output includes the R version information, copyright notice, platform details, and various help and documentation links. It also shows the 'head' command output for the 'mtcars' dataset, which contains columns: mpg, cyl, disp, hp, drat, wt, qsec, vs, am, gear, carb. Below the terminal window is a status bar with the text 'vertical resize -2'.

```
library(data.table)
# Load the mtcars dataset
# Calculate mean mpg
mydf <- mean(mtcars$mpg) > paste0()
# Print the result
print(mydf)
# Print the head of the mtcars dataset
cat('hello world from R')
>
> head(mtcars)
# Output of the head command
  mpg cyl disp hp drat wt qsec vs am gear carb
Mazda RX4     21.0   6 160 110 3.90 2.620 16.46  0  1   4   4
Mazda RX4 Wag 21.0   6 160 110 3.90 2.625 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   6 258 110 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

```
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
```

b.py 13,0-1 All python3 "b.py" [finished] 2,17 All

Figure 1.8: Vim - Python

The screenshot shows a Windows taskbar at the top with icons for File Explorer, Task View, Start, and a search bar. Below the taskbar is a horizontal tab bar with tabs for 'stl' (active), 'Neo-tree', 'Cargo.toml', 'use_std::f_i', 'Heart FM Online | Inte...', and 'NORMAL'. The main area contains four code editors:

- Neo-tree**: Shows the file structure of a Rust project with files like `main.rs` and `Cargo.toml`.
- stl**: Shows a C++ file with code related to `std::fs` and `std::path`.
- Cargo.toml**: Shows the configuration file for the project, specifying the package name as "testrust", version as "0.1.0", edition as "2021", and dependencies including `anyhow = "1.0.75"`. A note at the bottom says "# See more keys and their definitions at https://doc.rust-lang.org/cargo/".
- Cpp**: Shows a C++ file with imports from `os` and a function `some()` that prints the current directory.

At the bottom, there are two terminal windows:

- NORMAL**: Contains the command `fs::dir_tree("./")`.
- Bash**: Contains the command `fs::dir_tree("./")`.

Figure 1.9: Tmux - Nvim 1

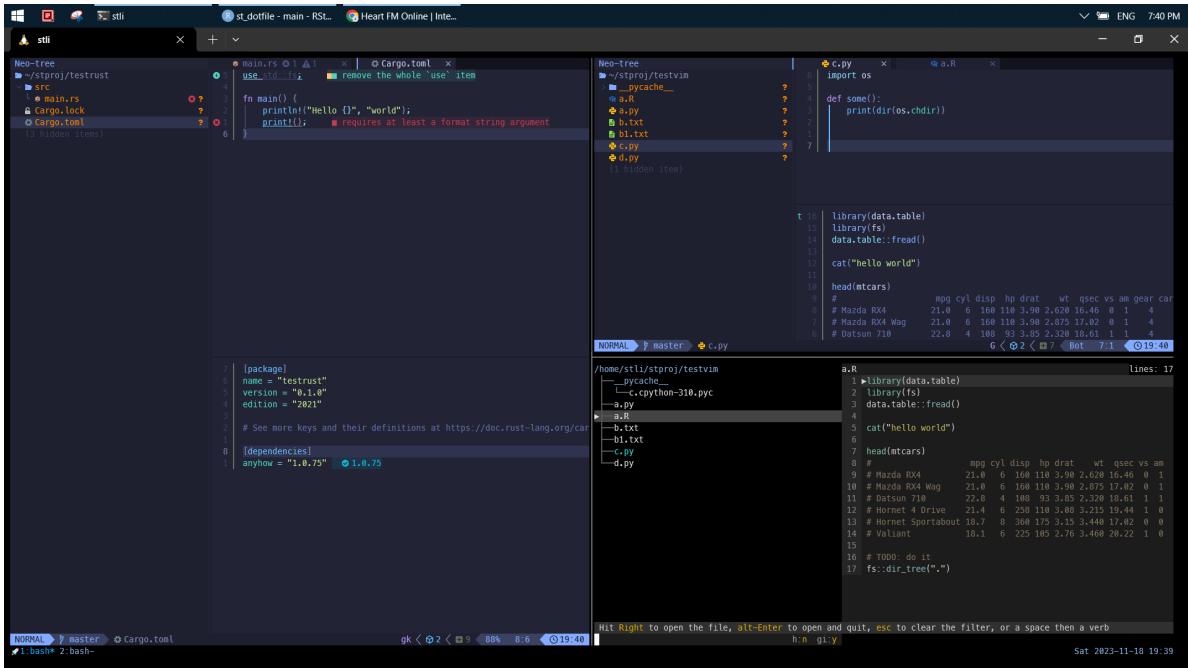


Figure 1.10: Tmux -Nvim 2

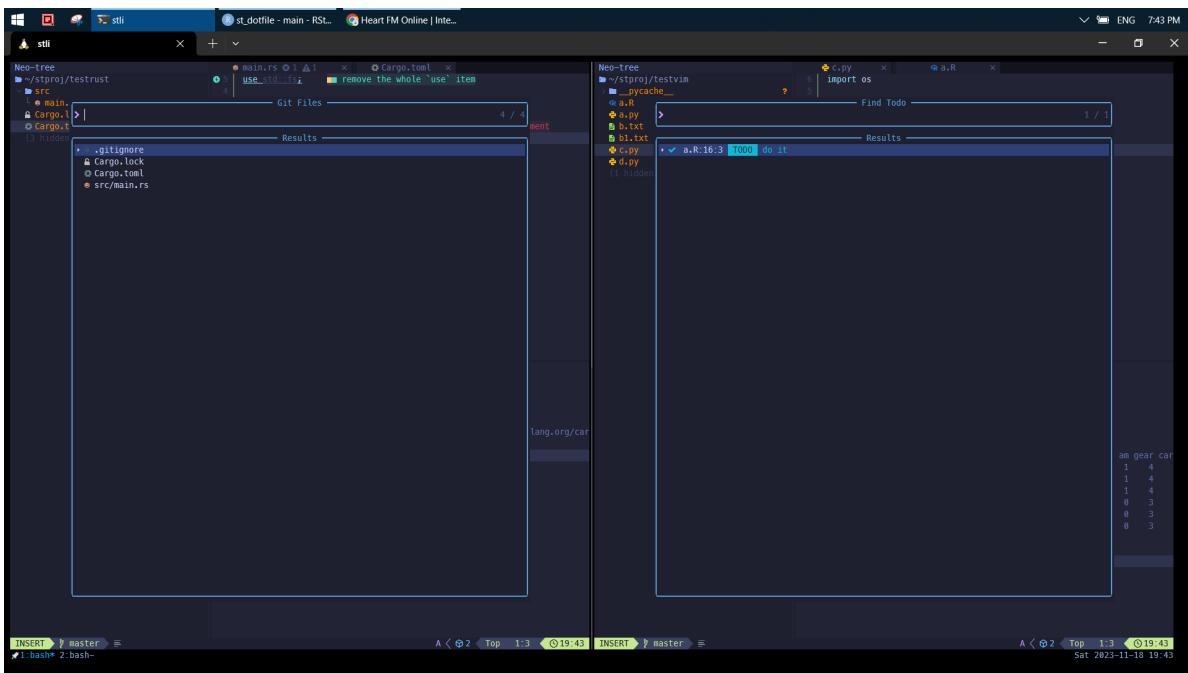


Figure 1.11: Tmux - Nvim 3

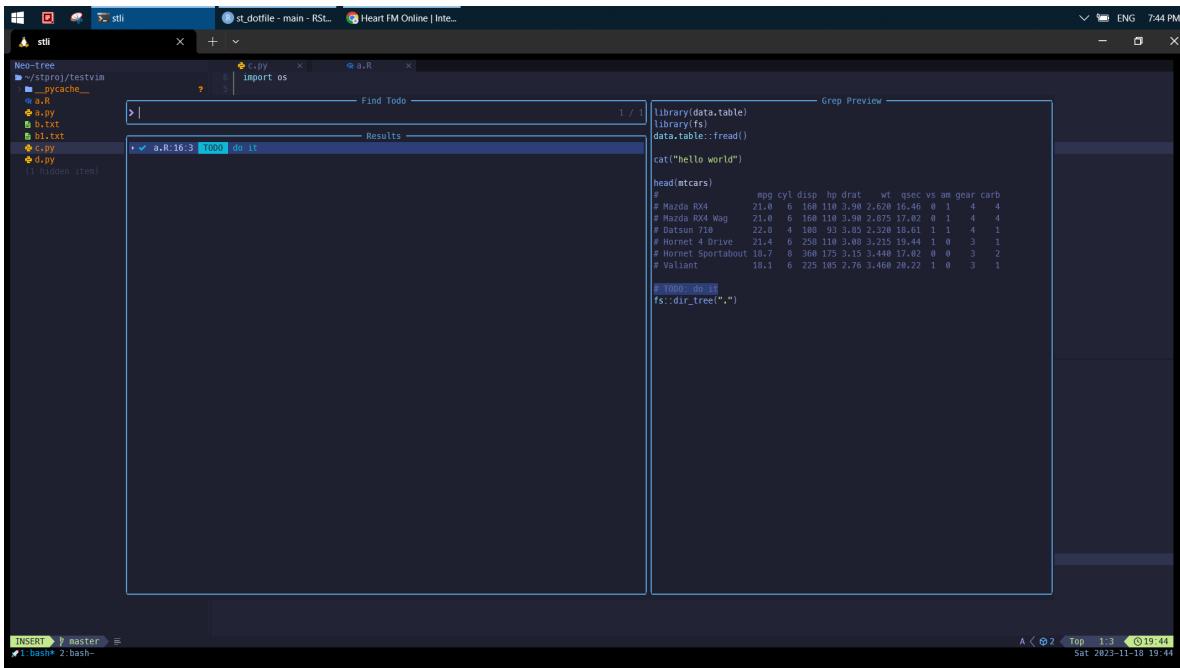


Figure 1.12: Tmux -Nvim 4

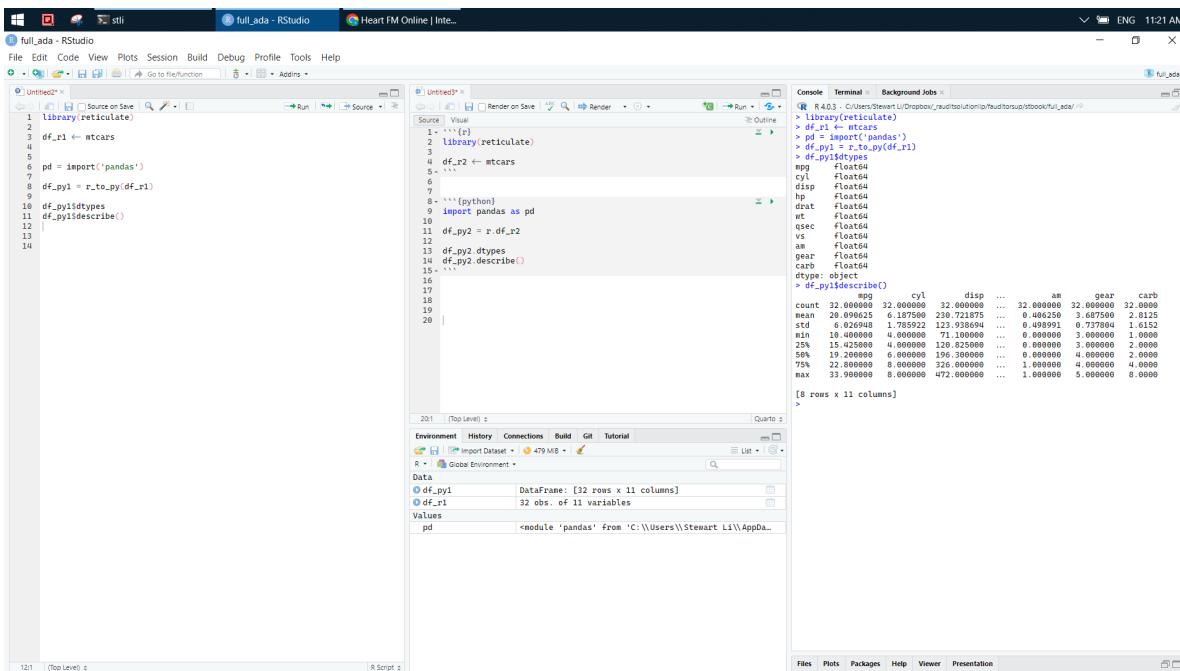


Figure 1.13: RStudio - R

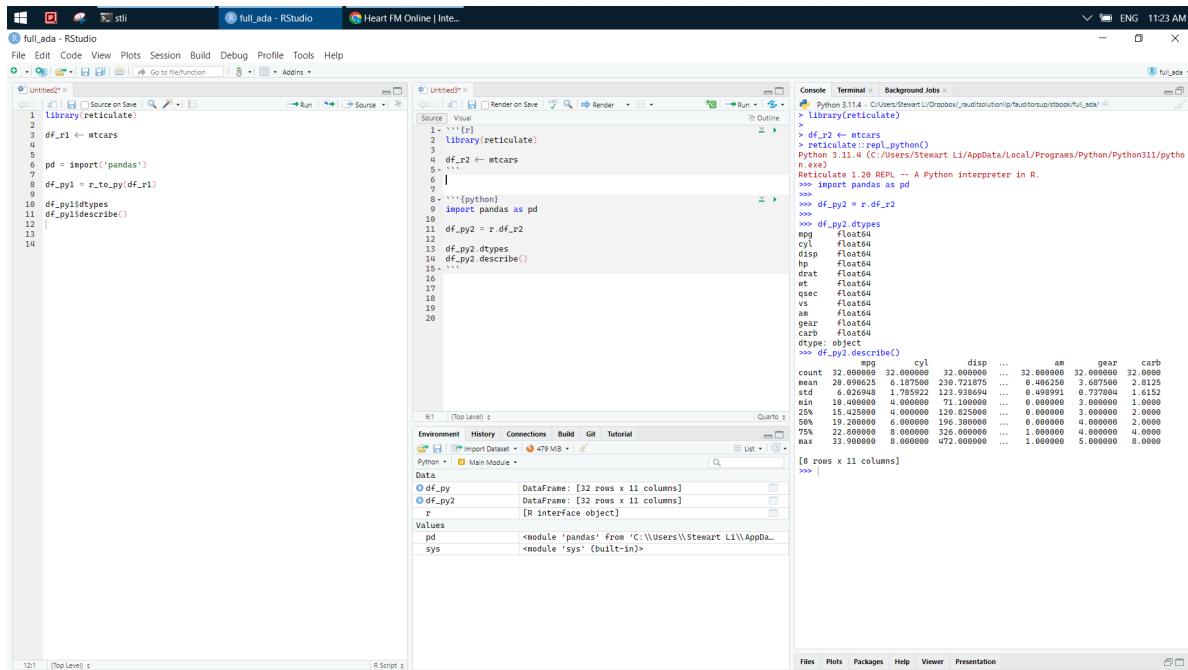


Figure 1.14: RStudio - Python

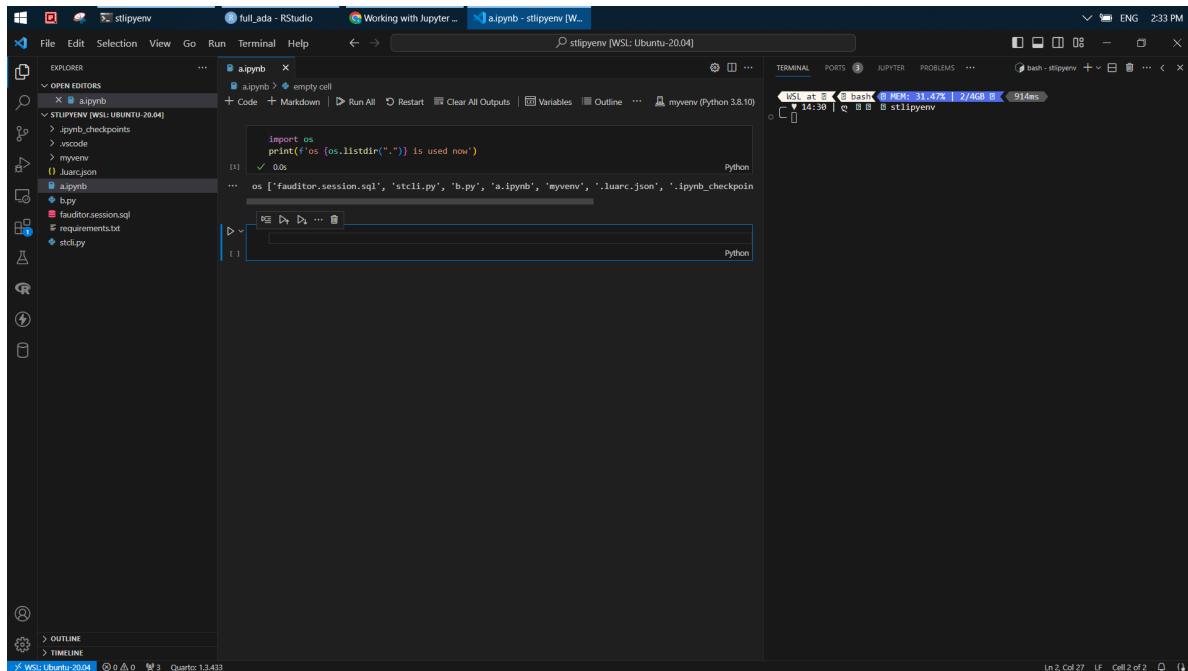


Figure 1.15: VS Code in Linux - Jupyter

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 has a dark theme. On the left is the Explorer sidebar with a tree view of files and folders, including 'OPEN EDITORS' (b.py), 'GROUP 1' (b.py), 'GROUP 2' (Interactive - b.py), and 'STLIPYENV (WSL: UBUNTU-20.04)' (jupyter_checkpoints, .vscode, myenv, Juancion, a.ipynb, b.py, fauditor.session.sql, requirements.txt, stdlib). The center-left is the code editor with a file named 'b.py' containing the following 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
```

The center-right is the 'Interactive' panel titled 'Interactive - b.py X'. It shows the message 'Connected to myenv (Python 3.8.10)'. Below it, a command history shows:

```
✓ import pandas as pd...\n... pandas 2.0.2 is used now\n\n1+1\n✓ 0.05\n...\n2
```

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

Figure 1.16: VS Code in Linux - Interactive cell

A screenshot of the Visual Studio Code interface running on a Windows system. The title bar shows multiple tabs: 'full_ada - RStudio', 'Heart FM Online | Inte...', 'b.py - testvim - Visual...', and 'testvim'. The main area has a dark theme. On the left is the Explorer sidebar with a tree view of files and folders, including 'OPEN EDITORS' (b.py), 'TESTVIM' (a.R, b.py), and 'b.py'. The center-left is the code editor with a file named 'b.py' containing the following 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\n12 # %%
```

The center-right is the 'TERMINAL' panel showing the output of the script execution:

```
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\nStewart: L1@DESKTOP-HCEU07A MINGW64 ~/Desktop/testvim\n$
```

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

Figure 1.17: VS Code in Windows - Script

A screenshot of the Visual Studio Code interface on Windows. The title bar shows multiple tabs: 'full_ada - RStudio', 'Heart FM Online | Inte...', 'b.py - testvim - Visual...', and 'testvim'. The left sidebar has an 'EXPLORER' view with files 'a.R' and 'b.py' listed under 'TESTVIM'. The main area has two code editors: one with 'b.py' containing Python code and another with 'pandas' imports. To the right is an 'Interactive' cell window titled 'Interactive - b.py X'. It contains a command-line interface with the output of running the code. At the bottom, there's a status bar with 'R (not attached) Ln3, Col1 Spaces:4 UTF-8 CRLF Python 3.11.4 64-bit Go Live Prettier'.

Figure 1.18: VS Code in Windows - Interactive cell

A screenshot of the Visual Studio Code interface on Windows. The title bar shows multiple tabs: 'full_ada - RStudio', 'Heart FM Online | Inte...', 'a.R - testvim - Visual...', and 'R Graphics: Device 2 (ACTIVE)'. The left sidebar has an 'EXPLORER' view with files 'a.R' and 'b.py' listed under 'TESTVIM'. The main area shows an R session with code like 'library(dplyr)', 'head(mtcars)', and 'plot(mtcars)'. Below the session is a 'TERMINAL' tab showing the output of 'head(mtcars)' which lists car models and their attributes. To the right is an 'R Graphics: Device 2 (ACTIVE)' window displaying a correlation matrix plot for the mtcars dataset, showing a grid of scatter plots for each variable against every other variable.

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.

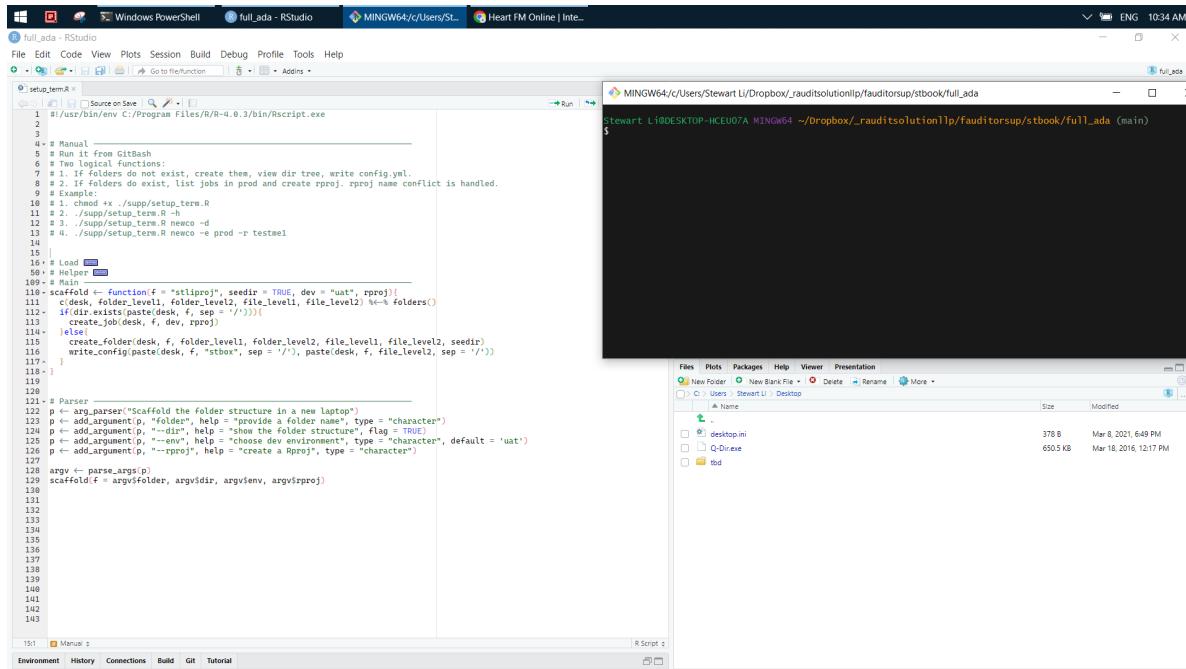


Figure 1.20: CLI R - GitBash 1

The screenshot shows a Windows desktop with several open windows. In the foreground, there's an RStudio interface with a code editor containing R script code. The code is related to setting up a folder structure for a project, involving functions like `create_folder`, `write_config`, and `scaffold`. The RStudio interface includes tabs for Environment, History, Connections, Build, Git, and Tutorial. Below the RStudio window is a taskbar with icons for File Explorer, Task View, and other system tools. In the background, there's a terminal window titled "MINGW64/c/Users/Stewart Li/Desktop/raudit solution/lp/fauditorsup/stbook/full_ada" showing command-line usage for a script named `setup_term.R`. The terminal also displays the output of running the script with various flags.

```
#!/usr/bin/env R --R-4.0.3/bin/Rscript.exe
# Manual
# Run this 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.
# 3. If folders do exist, write config.yml
# 4. If folders do exist, list jobs in prod and create testmvel
# Load
# Helper
# Helper
# scaffold = function(f = "stlproj", seedir = TRUE, dev = "uat", rproj) {
#   c(desk, folder.level1, folder.level2, file.level1, file.level2) <-> folders()
#   if(dir.exists(paste(desk, f, sep = '/'))){
#     create.job(desk, f, dev, rproj)
#   }
#   create.folder(desk, f, folder.level1, folder.level2, file.level1, file.level2, seedir)
#   write.config(paste(desk, f, "tbox", sep = '/'), paste(desk, f, file.level2, sep = '/'))
# }
# Parser
# arg_parser("Scaffold the folder structure in a new laptop")
# p <- add_argument(p, "folder", help = "the folder name", type = "character")
# p <- add_argument(p, "dev", help = "the dev environment", type = "character")
# p <- add_argument(p, "--env", help = "choose dev environment", type = "character", default = 'uat')
# p <- add_argument(p, "--rproj", help = "create a Rproj", type = "character")
# parse_args(p)
# scaffold(f = args$folder, arg$dir, arg$env, arg$rproj)
# Environment History Connections Build Git Tutorial
```

Figure 1.21: CLI R - GitBash 2

The screenshot shows the RStudio interface with two main panes. The left pane displays the R script 'setup_term.R'. The right pane shows a terminal window with the command 'cd /Users/Stewart.Li/Desktop/newco -d' followed by a listing of files and directories. The bottom navigation bar includes tabs for Environment, History, Connections, Build, Git, and Tutorial.

```
#!/usr/bin/env R --vanilla
# setup_term.R

1 #!/usr/bin/env C:/Program Files/R/R-4.0.3/bin/Rscript.exe
2
3 # Manual
4 # Run from GitBash
5 # Run logical functions:
6 # 1. If folders do not exist, create them, view dir tree, write config.yml
7 # 2. If folders do exist, list jobs in prod and create rproj. rproj name conflict is handled.
8 # 9
9 # 10
10 # 11 chmod +x ./setup_term.R
11 # 12 ./setup_term.R -h
12 # 13 ./setup_term.R newco -d
13 # 14 ./setup_term.R newco -e prod -r testmeil
14
15 # Load
15 # Helper
16 # Mail
17 # Scaffold
18 scaffold <- function(f = "scaffold", seedin = TRUE, dev = "uat", rproj) {
19   cdesk, folder.level1, folder.level2, file.level1, file.level2) <- %> folders()
20   if(!dir.exists(paste(desk, f, sep = '/'))){
21     create_job(desk, f, dev, rproj)
22   }
23   create.folder(desk, f, folder.level1, folder.level2, file.level1,
24                 file.level2, seedin)
25   write.config(paste(desk, f, "box", sep = '/'), paste(desk, f, file.level2, sep = '/'))
26 }
27
28 # Parser
29 p <- arg_parser("Scaffold the folder structure in a new laptop")
30 p <- add_argument(p, "folders", help = "the path to the folder name", type = "character")
31 p <- add_argument(p, "seedin", help = "set the seed environment flag = TRUE")
32 p <- add_argument(p, "--env", help = "choose dev environment", type = "character", default = 'uat')
33 p <- add_argument(p, "--rproj", help = "create a Rproj", type = "character")
34
35 args <- parse_args(p)
36 scaffold(f = args$folders, args$dir, args$env, args$rproj)
37
38
```

```
MINGW64/c/Users/Stewart.Li/Desktop/_raudsolution1lp/fauditorsup/stbook/full_ada
$ ./setup_term.R newco -d
C:/Users/Stewart.Li/Desktop/newco
|-- prod
|   |-- config.yml
|   |-- data
|   |-- proj
|   |-- res
|-- README.qmd
|-- sbin
|-- stdbox
|-- stdimg
|-- tbd
`-- uat
    '-- proj

Stewart.Li@DESKTOP-HCEU07A MINGW64 ~ /Dropbox/_raudsolution1lp/fauditorsup/stbook/full_ada
$
```

Files Plots Packages Help Viewer Presentation
New Folder New Blank File Delete Rename More
C:/Users/Stewart.Li/Desktop/newco/prod
Name Size Modified
config.yml 200 B Oct 12, 2023, 10:37 AM
data
proj
raw
res

Figure 1.22: 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 a terminal window titled "MINGW64/c/Users/Stewart Li/Desktop/full_ada" where the script is being run. The terminal output shows the script executing and creating a folder named "testmel". A file browser window is also visible at the bottom, showing files like "ignore", "R", and "testmel.Rproj".

```

#!/usr/bin/env 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 = "stliproj", 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
args <- parse.args(p)
scraftold = args$folder
args$folder = args$dev
args$dev = args$rproj

```

Figure 1.23: CLI R - GitBash 4

This screenshot is identical to Figure 1.23, showing the RStudio interface with the `setup.Term.R` script in the left pane and a terminal window in the right pane. The terminal shows the script running and creating a "testmel" folder. The file browser at the bottom is also the same.

```

#!/usr/bin/env 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 = "stliproj", 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
args <- parse.args(p)
scraftold = args$folder
args$folder = args$dev
args$dev = args$rproj

```

Figure 1.24: CLI R - GitBash 5

The screenshot shows the RStudio interface with an R script named `setup.R` open. The script contains code for setting up a folder structure, handling command-line arguments, and creating R projects. To the right of the RStudio window is a Windows PowerShell window showing the command `.\run_setup.bat newco -d` being run, which generates a folder structure. Below the PowerShell window is a file explorer showing the created directory structure.

```

# 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 testmel
# Load
# Helper
# Main
scffold <- 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")
scffold(f = argv$folder, argv$dev, argv$env, argv$rproj)

```

Figure 1.25: CLI R - PowerShell 1

This screenshot is similar to Figure 1.25, showing the RStudio interface with the `setup.R` script and the resulting folder structure in Windows PowerShell and file explorer. The command run is `PS C:\Users\Stewart Li\Desktop\newco> .\run_setup.bat newco -d`.

```

# 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 testmel
# Load
# Helper
# Main
scffold <- 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")
scffold(f = argv$folder, argv$dev, argv$env, argv$rproj)

```

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`.

The screenshot shows a web-based data exploration environment. On the left, there's a sidebar titled "My databases" containing a tree view of database structures. The main area is titled "My Notebook" and contains a code editor with the following SQL query:

```
1 SELECT text, by, id, COUNT(*) AS n
2 FROM sample_data.hn.hacker_news GROUP BY ALL
3 ORDER BY n DESC
4 LIMIT 6;
```

Below the code editor, a message states "Query executed in 1.39 s. Row count: 6". A table displays the results:

text	by	id	n
In addition to being quite possibly the best free introduc...	sn9	31,213,459	1
>Especially seems unnecessary->p;An option that sh...	forgotpwd16	31,038,085	1
What you#x27;ve mentioned is objectively wrong and ...	fzeroracer	31,650,610	1
Oh i love talcscale, it is so performant and just works (m...	wjokinen	29,935,984	1
I'll be watching some YouTube with kids, or some ...	Borganicbits	32,765,091	1
I have the same issue(s) on my third set of earpods and...	arthurmorgan	32,765,097	1

Figure 2.1: DuckDB cloud

The screenshot shows a terminal window titled 'stli' running on a Windows operating system. The window displays the following command-line session:

```

d:\stli>true ~./testf > cd
d:\stli>true ~ > ./duckdb md:
-- Loading resources from /home/stli/.duckdbrc
v0.9.2 3c695d9ba9
Enter ".help" for usage hints.
==> show databases;
  database_name
  varchar
my_db
sample_data
  ==> SELECT text, by, id, COUNT(*) AS n
> FROM sample_data.hn.hacker_news GROUP BY ALL
> ORDER BY n DESC
> LIMIT 6;

```

Below the command history, a table is displayed with the following data:

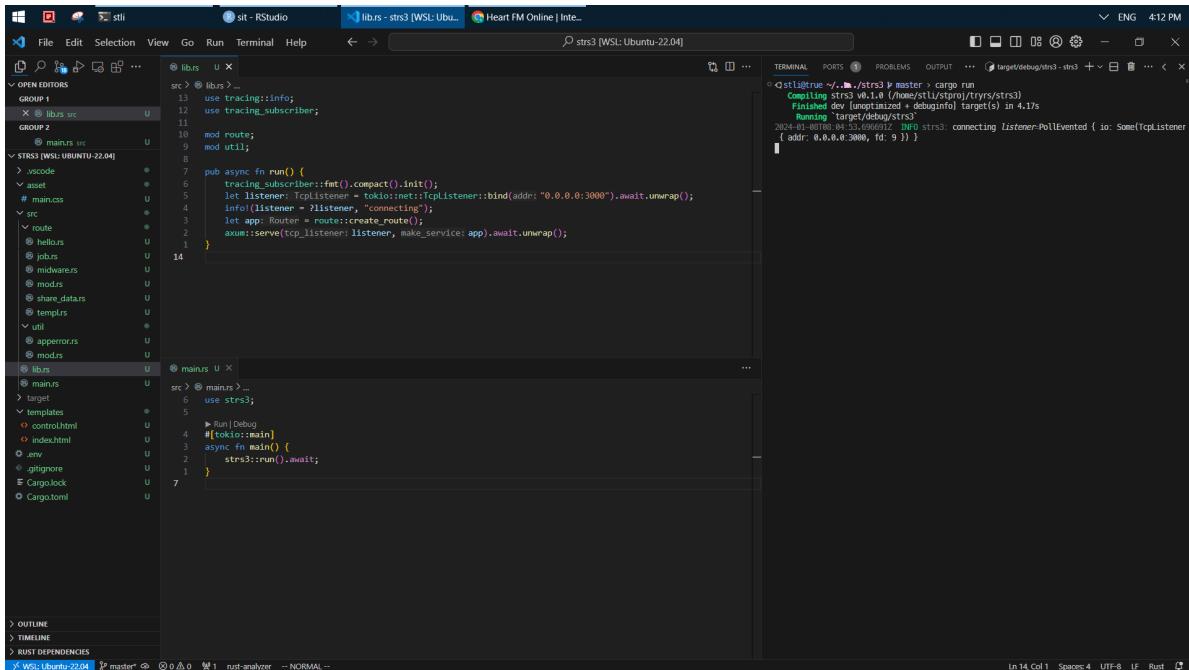
text	by	id	n
varchar	varchar	int64	int64
Roberts raise up.	rednerrus	32201944	1
Yep, I use this to get vimium-FF on Fennec. It's a little fiddly to setup but it's worth it if you're going for the cellphone model of ownership. Pay large for L.	re10	29798774	1
I do this, and it's very nice to tell who lost'f sold my email. psAlso I...	Shmueler2020	29798771	1
I have a learning disability related to some incredibly common cognitive issues t.	Shared444	32683683	1
You can theoretically send email to an IP address directly, like someone@[127.0...	DrewWebDesign	33807785	1
	csande17	33305117	1

The terminal window also shows the status bar at the bottom indicating 'Thu 2024-01-25 17:08'.

Figure 2.2: DuckDB terminal

3 Example 1

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



```
git clone https://github.com/rust-lang/rustls.git
cd rustls
cargo build --release
cargo run
```

The screenshot shows a terminal window with the following output:

```
git clone https://github.com/rust-lang/rustls.git
cd rustls
cargo build --release
cargo run
```

The terminal shows the command being run and its output, indicating the application is running at port 3000.

Figure 3.1: Web server

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

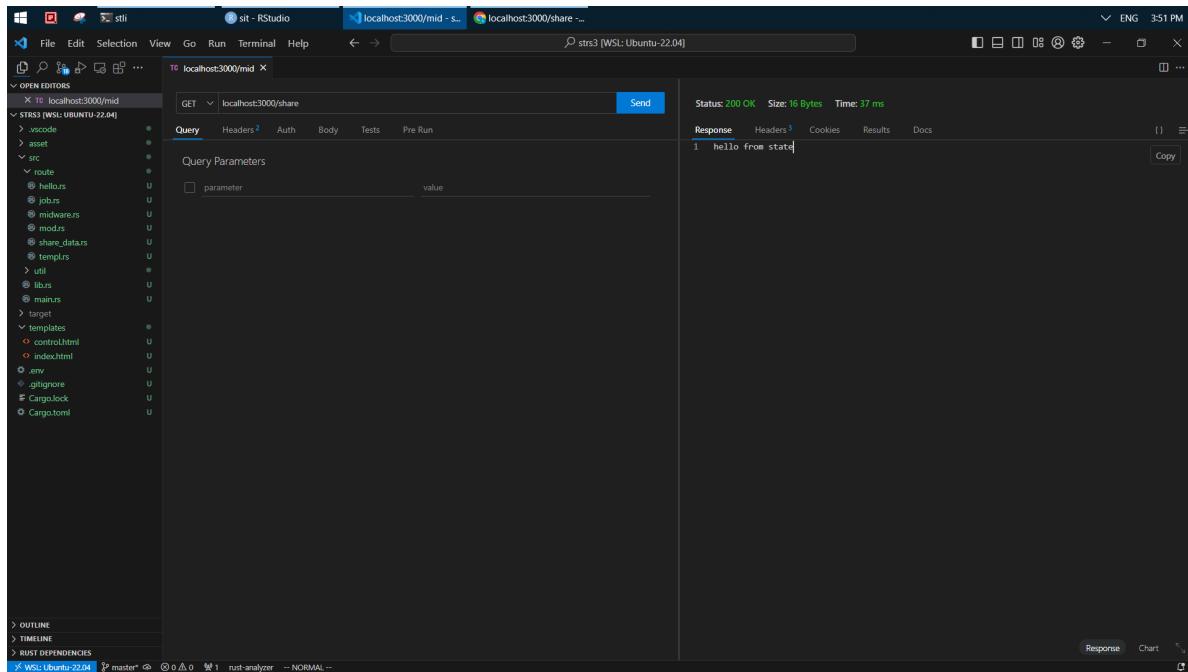


Figure 3.2: Get

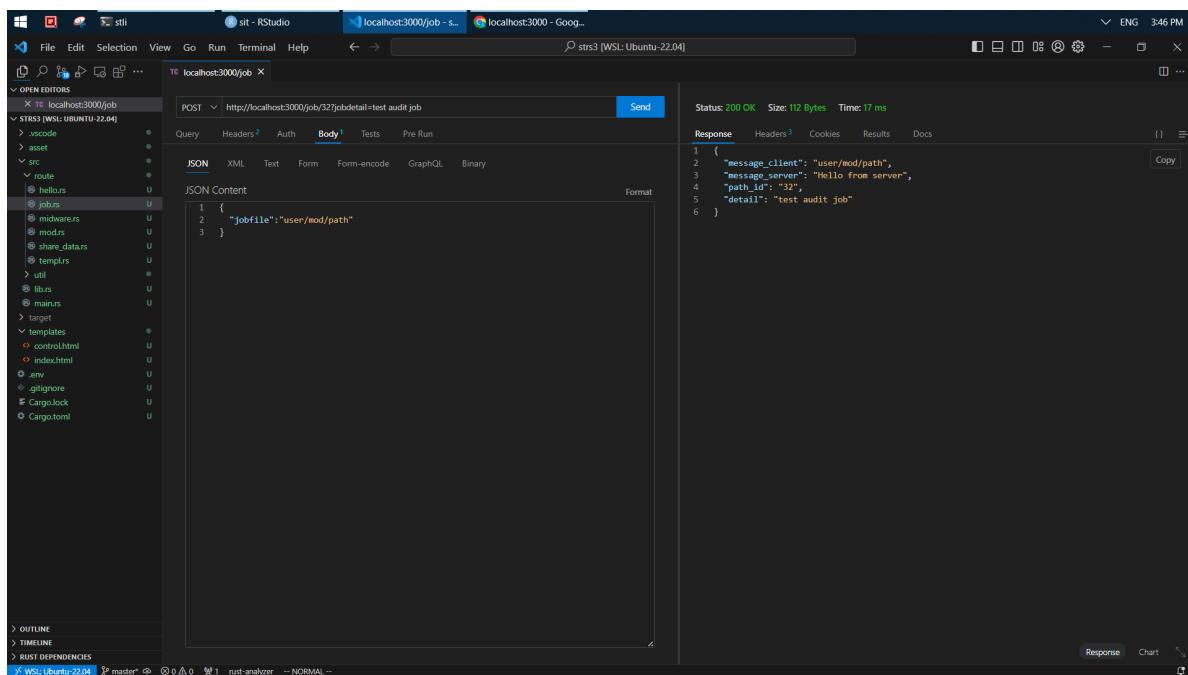


Figure 3.3: Post

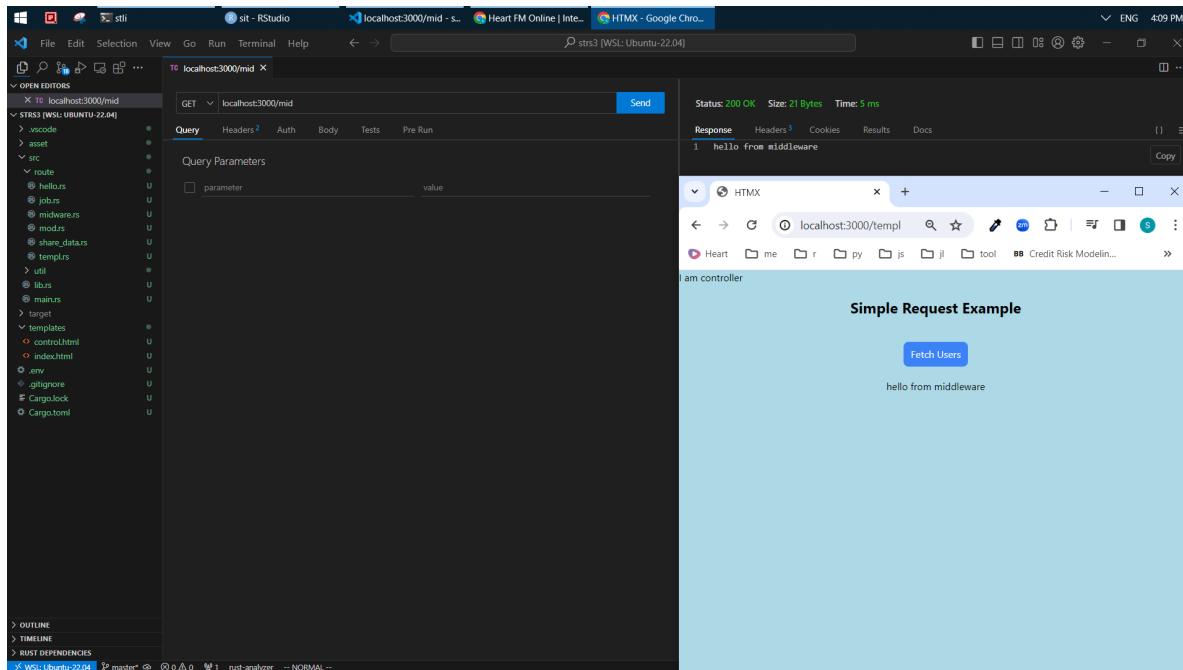


Figure 3.4: Template

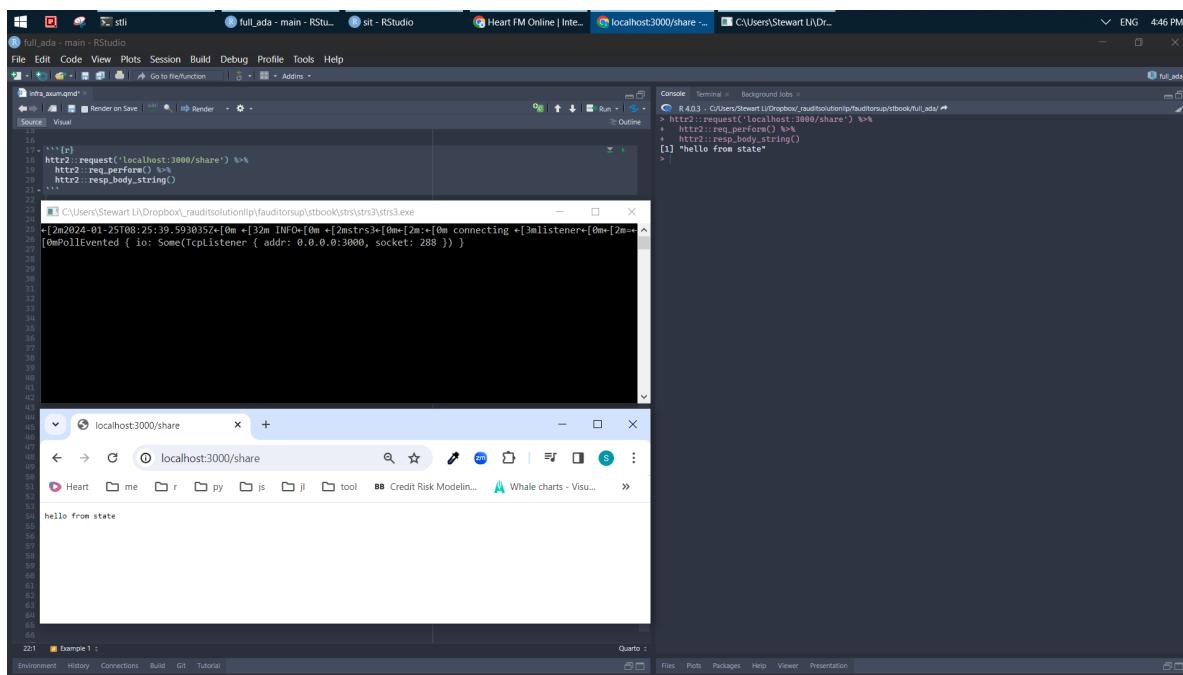


Figure 3.5: R client

The screenshot shows a Windows desktop environment with several open windows. In the foreground, a terminal window titled 'sts - RStudio' displays Rust code. The code includes imports for `std::io` and `Box`, defines a struct `mainus` with fields `stock` and `datas`, and implements a `main` function. The `main` function uses `tokio::run` to handle an asynchronous loop. Inside the loop, it reads command-line arguments, processes them, and then performs a series of operations involving `StringTreeNodes` and `Value` objects, ultimately printing the result to the console.

```
src/main.rs
1 use mainus;
2
3 fn main() {
4     use serde_json::Value;
5     use send_to_trees::FormatCharacters, StringTreeNode, TreeFormatting;
6
7     let stock: String = std::env::args().skip(1).collect::();
8     let datas: Value = manifest_from_server(stock).await.unwrap();
9
10
11     let mut tree: TreeNode<String> = StringTreeNode::new();
12     datas.as_array().unwrap().first().unwrap()["symbol"].Value
13         .as_str().Option<&str>
14         .unwrap() &str
15         .to_string(),
16     );
17
18     tree.push(
19         datas.as_array().unwrap().first().unwrap()["entityRegistrantName"].Value
20             .as_str().Option<&str>
21             .unwrap() &str
22             .to_string(),
23     );
24     tree.push(
25         datas.as_array().unwrap().first().unwrap()["auditorName"].Value
26             .as_str().Option<&str>
27             .unwrap() &str
28             .to_string(),
29     );
30     tree.push(
31         datas.as_array().unwrap().first().unwrap()["netIncomeloss"].Value
32             .as_id().Option<i64>
33             .unwrap() i64
34             .to_string(),
35     );
36
37     println!(
38         "{}",
39         tree.to_string_with_format(TreeFormatting::dir_tree(FormatCharacters::box_chars()))
40             .unwrap()
41     );
42 }
43
44 async fn manifest_from_server(ticker: String) -> Result<Value, Box<dyn std::error::Error>> {
45     let url: String =
46         "https://financialmodelingprep.com/api/v3/financial-statement-full-as-reported/".to_owned();
47     let client = Client::new();
48     let response = client.get(url).header("User-Agent", "Mozilla/5.0").send().await;
49     if response.is_err() {
50         return Err("Failed to fetch financial statement data".into());
51     }
52     let body = response.unwrap().text().await;
53     if body.is_err() {
54         return Err("Failed to parse JSON response".into());
55     }
56     let value: Value = serde_json::from_str(&body.unwrap()).unwrap();
57     Ok(value)
58 }
```

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 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`,

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.

The screenshot shows a Windows desktop environment. On the left, there is a code editor window titled "stcli.py" containing Python code. The code uses the argparse module to parse command-line arguments and the duckdb module to interact with a DuckDB database. It defines functions for filtering and summarizing data from a table named "finsample". The terminal window on the right shows the execution of the script and displays the resulting data frame. The data frame has columns: segment, country, product, cogs, profit, and date timestamp. It contains 3 rows of data for Government, Canada, and Carretera products.

```
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
```

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

Figure 4.1: CLI - argparse 1

Figure 4.2: CLI - argparse 2

Figure 4.3: CLI - argparse 3

```

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"])
73     @click.argument(["-red", "-green"], type=click.Choice(["red", "green"]))
74     @click.argument(["-help"], help="highlight data based on the provided threshold",
75     )
76     def highlight(res, output, highlight):
77         if highlight == "red":
78             res = res.with_columns(pl.col("new").map(lambda x: "(033[91m" + x + "\033[0m"))
79         else:
80             res = res.with_columns(pl.col("new").map(lambda x: "\033[96m" + x + "\033[0m"))

81     # click.echo(output)
82     click.echo(res)

83     if __name__ == "__main__":
84         main()

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

101    # click.echo(output)
102    click.echo(res)

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

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

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

```

Figure 4.4: CLI - click 1

```

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"])
73     @click.argument(["-red", "-green"], type=click.Choice(["red", "green"]))
74     @click.argument(["-help"], help="highlight data based on the provided threshold",
75     )
76     def highlight(res, output, highlight):
77         if highlight == "red":
78             res = res.with_columns(pl.col("new").map(lambda x: "(033[91m" + x + "\033[0m"))
79         else:
80             res = res.with_columns(pl.col("new").map(lambda x: "\033[96m" + x + "\033[0m"))

81     # click.echo(output)
82     click.echo(res)

83     if __name__ == "__main__":
84         main()

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

101    # click.echo(output)
102    click.echo(res)

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

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

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

```

Figure 4.5: CLI - click 2

```

stli@stli:~/stliproj/testv1m$ 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

```

Figure 4.6: CLI - click 3

```

stli@stli:~/stliproj/testv1m$ stpolars "/home/stli/stliproj/testv1m/mtcars.csv" about
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

```

Figure 4.7: CLI - click 4

```

fauditor# select * from client;
name | year | joblist | auditor | status
(0 rows)

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

fauditor#

```

The screenshot shows a terminal window titled 'stli' running on WSL: Ubuntu-20.04. It displays the output of a SQL query against a database named 'testtomi'. The first query returns no rows. The second query returns one row with the data: name='clientA', year=2023, joblist='clientA_2023', auditor='stewartli', and status='f'.

Figure 4.8: CLI - clap 1

```

WSL at 25 ~ bash @ WSL:55.52% 3/4GB B in 27s 630ms master w/ 24
└─ 16:45 | Q B B B testtomi
  cargo run -- -n clientA -y 2023 -a stewartli init
  Finished dev [unoptimized + debuginfo] target(s) in 0.95s
  Running `target/debug/testtomi` ...
  clientA_2023 -> stewartli init
  Initialized in /home/stli/stliproj/trystrust/testtomi
WSL at 25 ~ bash @ WSL:55.52% 3/4GB B in 298ms master w/ 24
└─ 17:03 | Q B B B testtomi
  cargo run -- -n clientA -y 2023 -a stewartli new -p /mnt/c/Users/Stewart_Li/Desktop/mpbc/
  Finished dev [unoptimized + debuginfo] target(s) in 0.12s
  Running `target/debug/testtomi` ...
  You currently have 1 jobs in total. They are [
    clientA_2023 ,
  ]
This job exists, cd to that folder
WSL at 25 ~ bash @ WSL:55.52% 3/4GB B in 431ms master w/ 25
└─ 17:04 | Q B B B testtomi
  source ./stliproj/myvenv/bin/activate
WSL at 25 ~ bash @ WSL:55.52% 3/4GB B in 16ms master w/ 25
└─ 17:07 | Q B B B testtomi
  cargo run -- -n clientA -y 2023 -a stewartli new -p /mnt/c/Users/Stewart_Li/Desktop/mpbc/
  Finished dev [unoptimized + debuginfo] target(s) in 0.12s
  Running `target/debug/testtomi` ...
  AuditLog - Audit Data Analytics
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
WSL at 25 ~ bash @ WSL:56.35% 3/4GB B in 377ms master w/ 25
└─ 17:20 | Q B B B testtomi

```

The screenshot shows a terminal window titled 'awp.R - testtomi [WSL]' running on WSL: Ubuntu-20.04. It displays the output of a 'cargo run' command followed by a 'source ./stliproj/myvenv/bin/activate' command. The terminal then shows the usage information for the 'testtomi' command, including its subcommands like 'init', 'new', and 'plan', and its options like '-n', '-y', '-a', '-s', '-h', and '-V'.

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.

```

# 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.

```

# 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'))

```

5.5 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 %>%

```

The screenshot shows an RStudio session titled "full_ada - RStudio". The Editor tab contains R code for data munging, including grouping by country code, summarizing sales, calculating first disputed late, and creating four categories based on credit policy. The Console tab displays a large data frame with columns like customer_id, disputed, invoice_number, invoice_amount, invoice_date, and various late metrics. The data frame has 106 rows and 13 columns.

```

#> dtsq_database.qnd %>
#> df_clean %>
#> group_by(country_code) %>
#> arrange(invoice_date) %>
#> summarise(
#>   sales = sum(invoice_amount),
#>   first_disputed_late = first(islate(disputed == 'Yes')),
#>   first_disputed_late_d = first(islate(disputed == 'Yes')),
#>   largest_late = max(late[disputed == 'Yes']),
#>   largest_inv_amt = invoice_amount[late == max(late)], .groups = 'drop')
#> )
#> df_clean %>
#> mutate(
#>   n_sales = sales,
#>   first_disputed_late = first(islate(disputed == 'Yes')),
#>   first_disputed_late_d = first(islate(disputed == 'Yes')),
#>   largest_late = max(late[disputed == 'Yes']),
#>   largest_inv_amt = invoice_amount[late == max(late)], .groups = 'drop')
#> )
#> 
```

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())

```

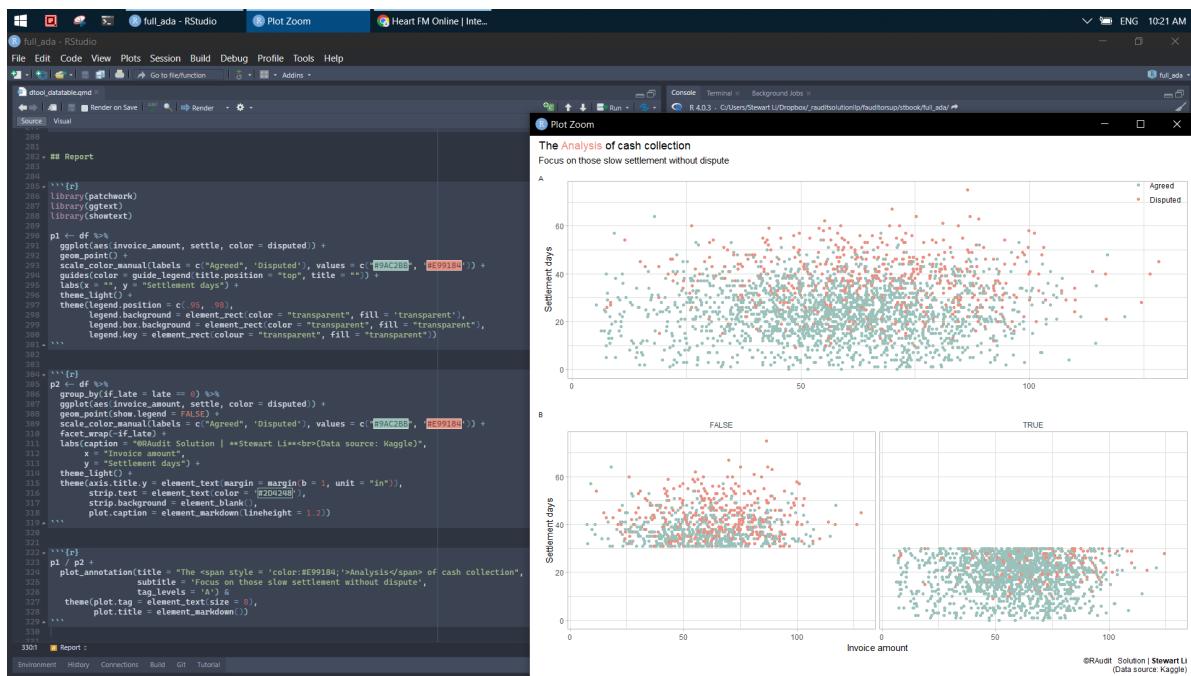


Figure 5.2: Combined plot

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 `lm` 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()
```

```

stli@true:~/.../testf> bat a.R
File: a.R
1 cat(cli::bg_red("Hello world - audit data analytics in R\n"))
2
3 library(data.table)
4
5 # Part 1 - IO
6 df <- fread(cmd = "grep -v Merc 'C:/Users/Stewart Li/Desktop/tbd/a.csv'", select = 2:13, colClasses = list(character = c("car"), numeric = 3:13))
7
8 dim(df)
9 glimpse(df)
10 df[sample(1:nrow(df), 3, replace = TRUE)]
11
12 # Part 2 - Count
13 df[, .N., .by = .(am, gear)]
14 df[, sum(mpg > mean(mpg)), .by = .(am, gear)]
15
16 # Part 3 - Summarize
17 df[, .SD(), by = .(am, gear)]
18 df[, lapply(.SD, max), .by = .(am, gear)]
19
20 with(df, tapply(mpg, list(am, gear), max, default = 0))
21 with(df, by(mpg, c(gear), summary))
22 aggregate(mpg ~ am * gear, data = df, FUN = median, subset = df$hp > 150)
23
24 # Part 4 - Reshape
25 melt(df, id = c("am", "gear"), measure = c("mpg", "hp"))
26 dcast(df, am * gear ~ carb, value.var = "mpg", fun.aggregate = list(min, mean, max), fill = 0)
27
28 # Part 5 - Filter
29 df[between(mpg, 25, 30)]
30 df[mpg > 20 & hp < 100, ]
31
32 # Part 6 - Mutate
33 df[, created_date := seq(as.Date("2021-12-1"), by = "month", length.out = nrow(df))]
34 df[, :=(new_mpg = (mpg %/% 2) * 2, new_mpg_hp = mpg/hp)]
35 df[, c("var1", "var2") := tstrsplit(displ, ".", fixed = TRUE, fill = 0)][]
36 df[, (mpg, new_mpg_if) = fcase(mpg < 15, "small", mpg > 30, "large", default = "medium")]
37
38 # Part 7 - Reshape
39 melt(df, id = c("am", "gear"), measure = c("mpg", "hp"))
40 dcast(df, am * gear ~ carb, value.var = "mpg", fun.aggregate = list(min, mean, max), fill = 0)
41
42 # Part 8 - Filter
43 df[between(mpg, 25, 30)]
44 df[mpg > 20 & hp < 100, ]
45
46 # Part 9 - Mutate
47 df[, created_date := seq(as.Date("2021-12-1"), by = "month", length.out = nrow(df))]

stli@true:~/.../testf> bat a1.R
File: a1.R
1 cat(cli::bg_red("Hello world - audit data analytics in Python\n"))
2
3 library(data.table)
4
5 # Part 1 - IO
6 df <- fread(cmd = "grep -v Merc 'C:/Users/Stewart Li/Desktop/tbd/a.csv'", select = 2:13, colClasses = list(character = c("car"), numeric = 3:13))
7
8 dim(df)
9 glimpse(df)
10 df[sample(1:nrow(df), 3, replace = TRUE)]
11
12 # Part 2 - Count
13 df[, .N., .by = .(am, gear)]
14 df[, sum(mpg > mean(mpg)), .by = .(am, gear)]
15
16 # Part 3 - Summarize
17 df[, .SD(), by = .(am, gear)]
18 df[, lapply(.SD, max), .by = .(am, gear)]
19
20 with(df, tapply(mpg, list(am, gear), max, default = 0))
21 with(df, by(mpg, c(gear), summary))
22 aggregate(mpg ~ am * gear, data = df, FUN = median, subset = df$hp > 150)
23
24 # Part 4 - Reshape
25 melt(df, id = c("am", "gear"), measure = c("mpg", "hp"))
26 dcast(df, am * gear ~ carb, value.var = "mpg", fun.aggregate = list(min, mean, max), fill = 0)
27
28 # Part 5 - Filter
29 df[between(mpg, 25, 30)]
30 df[mpg > 20 & hp < 100, ]
31
32 # Part 6 - Mutate
33 df[, created_date := seq(as.Date("2021-12-1"), by = "month", length.out = nrow(df)))
34 df[, :=(new_mpg = (mpg %/% 2) * 2, new_mpg_hp = mpg/hp)]
35 df[, c("var1", "var2") := tstrsplit(displ, '.', fixed = TRUE, fill = 0)][]
36 df[, (mpg, new_mpg_if) = fcase(mpg < 15, "small", mpg > 30, "large", default = "medium")]

stli@true:~/.../testf>

```

Thu 2024-01-25 15:10

Figure 6.1: Diff 1

```

stli@true:~/.../testf> bat a.R
File: a.R
1 cat(cli::bg_red("Hello world - audit data analytics in R\n"))
2
3 library(data.table)
4
5 # Part 1 - IO
6 df <- fread(cmd = "grep -v Merc 'C:/Users/Stewart Li/Desktop/tbd/a.csv'", select = 2:13, colClasses = list(character = c("car"), numeric = 3:13))
7
8 dim(df)
9 glimpse(df)
10 df[sample(1:nrow(df), 3, replace = TRUE)]■ Trailing whitespace is superfluous.
11
12 # Part 2 - Count
13 df[, .N., .by = .(am, gear)]
14 df[, sum(mpg > mean(mpg)), .by = .(am, gear)]
15
16 # Part 3 - Summarize
17 df[, .SD(), by = .(am, gear)]
18 df[, lapply(.SD, max), .by = .(am, gear)]
19
20 with(df, tapply(mpg, list(am, gear), max, default = 0))
21 with(df, by(mpg, c(gear), summary))
22 aggregate(mpg ~ am * gear, data = df, FUN = median, subset = df$hp > 150)■ Lines should not be here
23
24 # Part 4 - Reshape
25 melt(df, id = c("am", "gear"), measure = c("mpg", "hp"))
26 dcast(df, am * gear ~ carb, value.var = "mpg", fun.aggregate = list(min, mean, max), fill = 0)■ Lines should not be here
27
28 # Part 5 - Filter
29 df[between(mpg, 25, 30)]
30 df[mpg > 20 & hp < 100, ]
31
32 # Part 6 - Mutate
33 df[, created_date := seq(as.Date("2021-12-1"), by = "month", length.out = nrow(df))]■ Lines should not be here
34 df[, :=(new_mpg = (mpg %/% 2) * 2, new_mpg_hp = mpg/hp)]■ Put spaces around all infix operators.
35 df[, c("var1", "var2") := tstrsplit(displ, '.', fixed = TRUE, fill = 0)][]■ Trailing whitespace is superfluous.
36 df[, (mpg, new_mpg_if) = fcase(mpg < 15, "small", mpg > 30, "large", default = "medium")]■ Trailing whitespace is superfluous.
37
38 # Part 7 - Reshape
39 melt(df, id = c("am", "gear"), measure = c("mpg", "hp"))

stli@true:~/.../testf> bat a1.R
File: a1.R
1 cat(cli::bg_red("Hello world - audit data analytics in Python\n"))
2
3 library(data.table)
4
5 # Part 1 - IO
6 df <- fread(cmd = "grep -v Merc 'C:/Users/Stewart Li/Desktop/tbd/a.csv'", select = 2:13, colClasses = list(character = c("car"), numeric = 3:13))
7
8 dim(df)
9 glimpse(df)
10 df[sample(1:nrow(df), 3, replace = TRUE)]■ Trailing whitespace is superfluous.
11
12 # Part 2 - Count
13 df[, .N., .by = .(am, gear)]
14 df[, sum(mpg > mean(mpg)), .by = .(am, gear)]
15
16 # Part 3 - Summarize
17 df[, .SD(), by = .(am, gear)]
18 df[, lapply(.SD, max), .by = .(am, gear)]
19
20 with(df, tapply(mpg, list(am, gear), max, default = 0))
21 with(df, by(mpg, c(gear), summary))
22 aggregate(mpg ~ am * gear, data = df, FUN = median, subset = df$hp > 150)
23
24 # Part 4 - Reshape
25 melt(df, id = c("am", "gear"), measure = c("mpg", "hp"))
26 dcast(df, am * gear ~ carb, value.var = "mpg", fun.aggregate = list(min, mean, max), fill = 0)
27
28 # Part 5 - Filter
29 df[between(mpg, 25, 30)]
30 df[mpg > 20 & hp < 100, ]
31
32 # Part 6 - Mutate
33 df[, created_date := seq(as.Date("2021-12-1"), by = "month", length.out = nrow(df))]■ Lines should not be here
34 df[, :=(new_mpg = (mpg %/% 2) * 2, new_mpg_hp = mpg/hp)]■ Put spaces around all infix operators.
35 df[, c("var1", "var2") := tstrsplit(displ, '.', fixed = TRUE, fill = 0)][]■ Trailing whitespace is superfluous.
36 df[, (mpg, new_mpg_if) = fcase(mpg < 15, "small", mpg > 30, "large", default = "medium")]■ Lines should not be here
37
38 # Part 7 - Reshape
39 melt(df, id = c("am", "gear"), measure = c("mpg", "hp"))

stli@true:~/.../testf>

```

Thu 2024-01-25 15:20

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(), '.xlsx'))
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))

```

The screenshot shows a Windows desktop environment with several open windows. At the top, there's a taskbar with icons for File Explorer, Task View, and other system tools. Below the taskbar, there are four RStudio windows side-by-side, each displaying different code snippets and data frames. The first window contains code for calculating payroll amounts against financial amounts. The second window shows a data frame for staff members. The third window contains code for identifying multiple claims per expense. The fourth window displays a data frame for claims. A fifth window, titled 'Heart FM Online | Inte...', is visible at the top right. At the bottom of the screen, there's a navigation bar with links like Environment, History, Connections, Git, and Tutorial.

```
audit_stg.rd x sti full_add - main - RS... sit - RStudio Heart FM Online | Inte... ENG 241 PM
```

File Edit Code View Plots Session Build Debug Profile Tools Help Go to Function Addins +

Sources: Visual

audit_stg.rd

```
81 #> ````{c}
82 # cross check payroll amount against finance amount
83 df_clean %>
84   group_by(staff_id, staff_name) %>
85   summarise(amt_exp = sum(amount_s.x),
86             amt_paid = sum(amount_s.y) / n(),
87             amt_diff = amt_exp - amt_paid,
88             .groups = 'drop')
89 ````
```

```
90 #> ````{c}
91 # compare date to ensure no claim happens before incurred or after resigned
92 df_clean %>
93   dplyr::filter(claim_date > expense_date)
94 
```

```
95 df_clean %>
96   dplyr::filter(claim_date > last_date | claim_date == on_leave)
97 
```

```
98 #> ````{c}
99 # identify multiple claims for the same expense
100 df_clean %>
101   group_by(staff_id, staff_name, purpose, amount_s.x) %>
102   dplyr::filter(n() > 1)
103 
```

```
104 #> ````{c}
105 # ensure staff name and their bank account number updated timely
106 df_clean %>
107   dplyr::filter(is_na(edits_to_hr_data),
108                 bank_account_no.x == bank_account_no.y)
109 
```

```
110 df_clean %>
111   dplyr::filter(name.x != name.y)
112 
```

```
113 #> ````{c}
114 # process audit working paper
115 library(pointblank)
116 
```

```
117 ag = df_clean %>
118   create_agent_label(%>% "very simple example", tbl_name = "payroll") %>
119   col_vals_between(columns = claim_date, left = vars(expense_date), right = vars(last_date)) %>
120   interrogate()
121 
```

```
122 ag
123 
```

```
124 #> Procedures :
```

Figure 6.3: Audit Procedure 1

Figure 6.4: Audit Procedure 2

```

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')

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))

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)

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>.