

Data Integreation

Data integration is the process of combining data from different sources into a unified view so it can be easily analyzed or used by applications.

Example:

A company might have:

- Customer data in a CRM system (Salesforce)
- Sales data in an Excel sheet
- Product data in a database

Data integration means bringing all this together so you can answer questions like:

“Which customers bought which products last month

Why We Need Data Integration

- Data Heterogeneity
 - “Heterogeneous” = “different kinds.”
 - Different sources store data in different formats, structures, and systems.
 - Example: One system might use “customer_id” while another uses “custID”—they mean the same thing but are stored differently.
 - So integration is needed to make all that data work together.
- Knowledge Discovery
 - You can’t find useful patterns or insights if your data is scattered across many systems.
 - Integration allows for data mining and analytics across all data sources.
 - Example: Detecting that “customers who buy X also buy Y” requires all purchase data to be in one place.
- Data ≠ Knowledge
 - Just collecting data doesn’t give you value.
 - The value comes from analyzing and interpreting the data — turning it into knowledge or actionable insights.
 - Analogy:
Data = raw ingredients
Knowledge = cooked meal
- Heterogeneity Challenges
 - The main difficulties in data integration come from differences in:
 - Architectures: (relational DB, NoSQL, files, APIs)
 - Access methods: (SQL queries, REST APIs, manual CSV uploads)
 - Data structures: (different column names, data types, hierarchies)

Helper Functions when Connecting to Postgres from R

The Helper Script: psql_queries.R

- psql_select()
 - Purpose: Retrieve (select) data from Postgres into R.
 - What it does: Executes an SQL SELECT query and returns the result as an R data.frame.

```
r  
  
data <- psql_select("SELECT * FROM customers;")  
head(data)
```

 Code kopieren

- Use when you want to *read* or *analyze* data.

- psql_manipulate()
 - Purpose: Perform operations that *change* data or structure in the database.
 - Examples of use:
 - Create tables or schemas
 - Insert, update, or delete row

```
r  
  
psql_manipulate("CREATE TABLE sales (id SERIAL, amount NUMERIC);")
```

 Code kopieren

- Use when you want to *modify* the database (not fetch data).

- psql_append_df()
 - Purpose: Quickly insert a whole R data.frame into a Postgres table.
 - Why useful: If you have large datasets in R and want to upload them into Postgres.

```
r  
  
psql_append_df(my_dataframe, "sales")
```

 Code kopieren

- Use when you want to *upload* R data directly to the database.

The Credential Script: .credentials.R

- This file stores your database login information (username, password, host, port, etc.) for different Postgres servers.
- Important Notes:
 - Never store passwords as plain text in a shared or public environment.
 - The dot (.) at the beginning of .credentials.R makes it a hidden file in RStudio.
 - To see it: Go to the *Files* pane → click *More* → select *Show Hidden Files*.
 - If you're using Git, make sure .credentials.R is listed in .gitignore
 - This prevents accidentally uploading your login credentials to GitHub or GitLab.

Connecting through the docker network

Check the available networks in Docker

Docker network ls

```
*** System restart required ***
root@ubuntu-s-lvcpu-lgb-fral-01:~# docker network ls
NETWORK ID     NAME      DRIVER      SCOPE
3aa02e02c60a   bridge    bridge      local
72bf3977433b   db_r_shiny bridge      local
b8cf7fbef2f2   host      host       local
63b3015476f2   none      null      local
root@ubuntu-s-lvcpu-lgb-fral-01:~#
```

Previously, we created a Docker network (db_r_shiny) where our postgres and rstudio containers are attached.

inspect the db_r_shiny network by running

docker network inspect db_r_shiny

```
root@ubuntu-s-lvcpu-lgb-fral-01:~# docker network inspect db_r_shiny
[
    {
        "Name": "db_r_shiny",
        "Id": "72bf3977433b7dad563e545a13ec866902d910799a362fdc3b68c12ba8192140",
        "Created": "2025-09-06T10:02:07.167438936Z",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv4": true,
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": {},
            "Config": [
                {
                    "Subnet": "10.0.42.0/24",
                    "IPRange": "10.0.42.128/25"
                }
            ]
        },
        "Internal": false,
        "Attachable": true,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {
            "16da47902cd8f0510d3bb5cf47c81cled4ce093e87581571a45071ca068559c": {
                "Name": "postgres",
                "EndpointID": "3a75a7ba7837e13ab4517f91b1baab421b0c6413ee00f87f1dfbccaf2398baa7",
                "MacAddress": "46:8e:78:a9:c3:44",
                "IPv4Address": "10.0.42.130/24",
                "IPv6Address": ""
            },
            "1d90cf679838c43fc03bcccea4e240a6517af54493f3a7bd077b913ff400b0ea6": {
                "Name": "rstudio",
                "EndpointID": "3c6166f65df9a6cecd3b406b59a96d2069b36b53b432cc76e94883971ae517bc",
                "MacAddress": "fe:d5:24:49:a8:49",
                "IPv4Address": "10.0.42.129/24",
                "IPv6Address": ""
            }
        },
        "Options": {},
        "Labels": {}
    }
]
```

Both your Postgres container and RStudio container are attached to it.

This means they can communicate directly without exposing ports publicly.

When two containers are on the same network (db_r_shiny), you have two options for the host when connecting from R to Postgres:

- *Option 1: Use the container name*
 - If your Postgres container is named postgres, you can connect like this:

```
r
host = "postgres"
```

Code kopieren

- *Option 2: Use the internal IP address*
 - You can also use the IP shown in the network inspect output:

```
r
host = "10.0.42.129"
```

Code kopieren

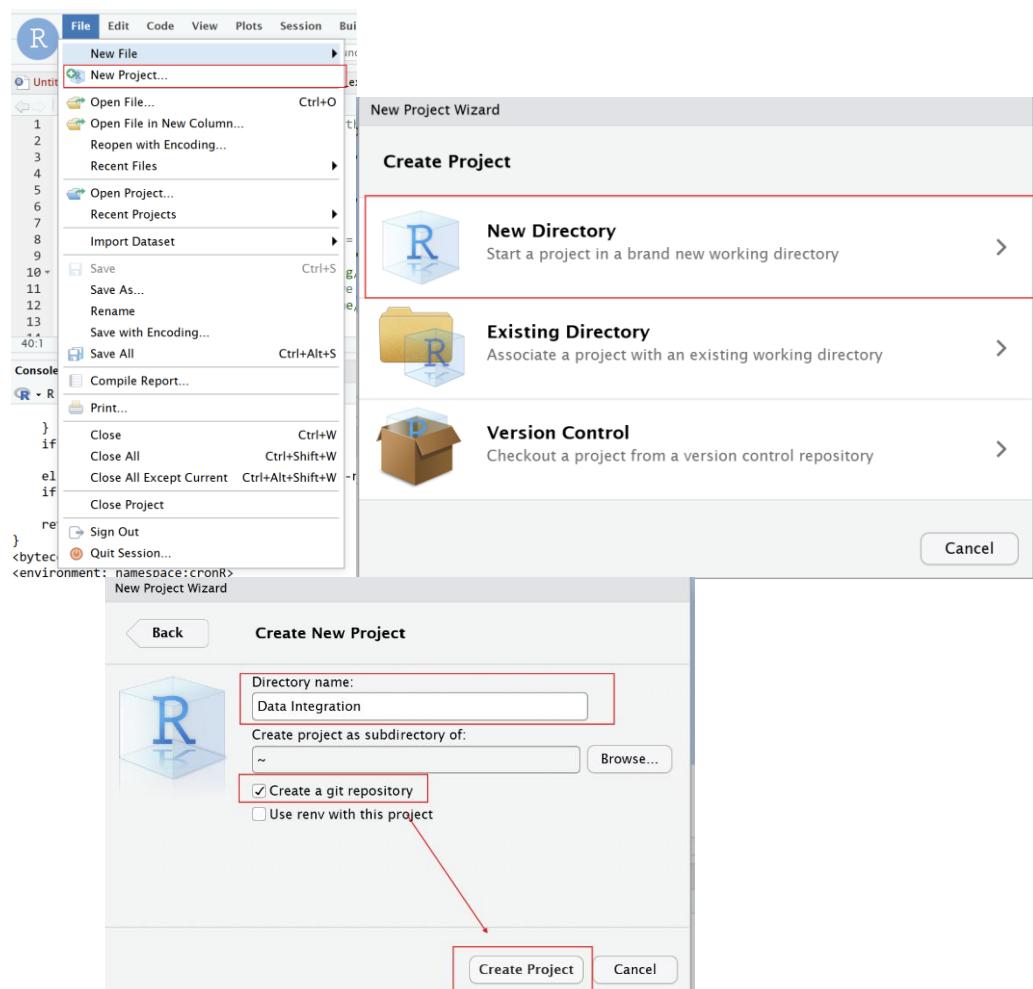
Connecting to Postgres from R

Example of ".credentials.R file"

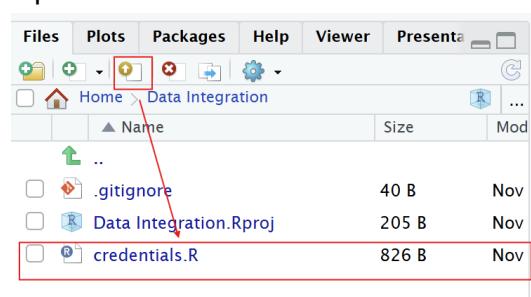
```
cred_psycopg_docker <- list(dbname = "postgres",
                           host   = "postgres",
                           user   = "postgres",
                           pass   = 'you-own-password-for-postgres',
                           port   = 5432)

cred_psycopg_150 <- list(dbname = "databasename-from-brightspace",
                         host   = "159.233.30.236",
                         user   = 'username-from-brightspace',
                         pass   = 'password-from-brightspace',
                         port   = 6432)
cred_psycopg_150_dvd <- list(dbname = "dvrental",
                           host   = "159.233.30.236",
                           user   = 'username-from-brightspace',
                           pass   = 'password-from-brightspace',
                           port   = 6432)
```

Create a new project



Upload ".credentials.R file" to R



Modify the credentials.R file

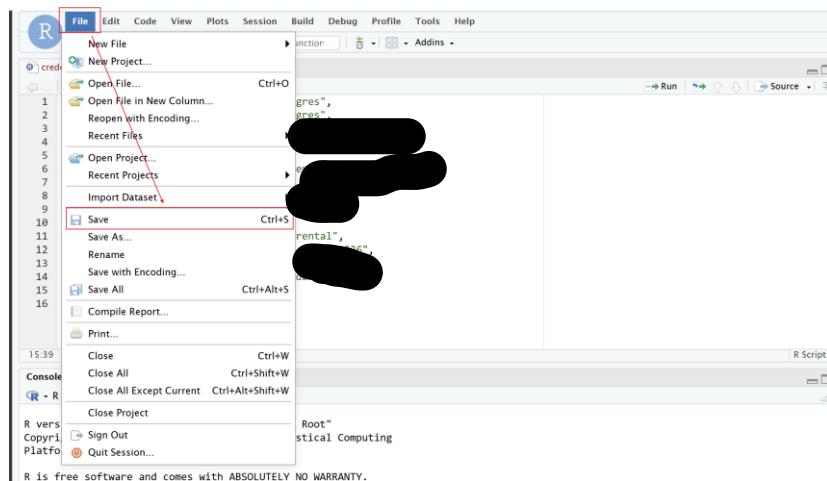
```

1 cred_pgsql_docker <- list(dbname = "postgres",
2   host = [REDACTED], host [REDACTED] "The host from inspecting Docker"
3   user = [REDACTED], user [REDACTED]
4   pass = 'g[REDACTED]', pass [REDACTED] "The Password we used when we created Docker"
5   port = 5432) port = 5432)
6
7 cred_pgsql_150 <- list(dbname = "databasename-from-brightspace",
8   host = "159.223.30.236",
9   user = [REDACTED], user [REDACTED]
10  pass = 'g[REDACTED]', pass [REDACTED] "The credentials we got from the Professor"
11  port = 6432) port = 6432)
12 cred_pgsql_150_dvd <- list(dbname = "dyvrental",
13   host = "159.223.30.236",
14   user = [REDACTED], user [REDACTED]
15   pass = 'g[REDACTED]', pass [REDACTED] "The credentials we got from the Professor"
16   port = 6432) port = 6432)
17

```

Karagic, Dino | user66 | user66db | dbmng_dv_66

Save changes



add file to the gitignore

```

1 .Rproj.user
2 .Rhistory
3 .RData
4 .Ruserdata
5 credentials.R

```

Gitignore: .gitignore

```

Environment History Connections Git Tutori... (no branch)
Staged Status Path
.gitignore
Data Integration.Rproj

Files Plots Packages Help Viewer Present...
Name Size Mod
.gitignore 53 B Nov
credentials.R 726 B Nov
Data Integration.Rproj 205 B Nov

```

Console Terminal Background Jobs

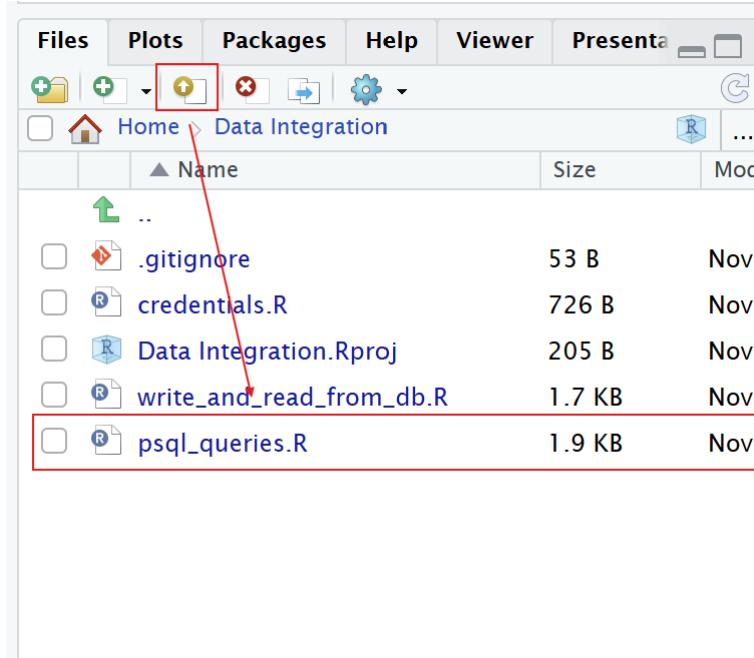
```

Terminal 1 ~$ git rm --cached credentials.R
fatal: No pathspec was given. Which files should I remove?

```

Read and write to Postgres

Upload the psql_queris.R



First load the librarys and Function the 2 files credentials.R & psql_queries.R

```
1 library(RPostgres)
2 library(DBI)
3
4 # Put the credentials in this script
5 # Never push credentials to git!! --> use .gitignore on .credentials.R
6 source("credentials.R")
7
8 # Function to send queries to Postgres
9 source("psql_queries.R")
10
```

Create a Schema

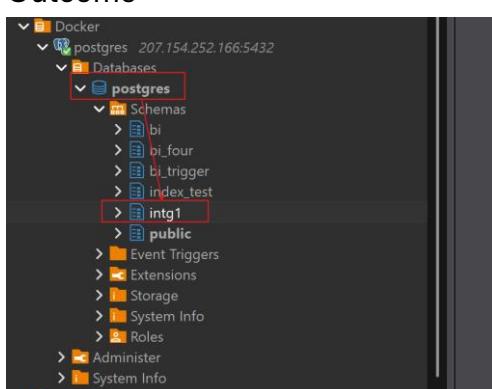
```
12 psql_manipulate(cred = cred_psql_docker,
13   query_string = "CREATE SCHEMA intg1;")
```

cred_psql_docker is the name we gave the postgres database in the credentials File

FYI: cred_psql_docker from the credentials file

```
1 cred_psql_docker <- list(dbname =
2   host =
3   user =
4   pass =
5   port = 5432)
```

Outcome



Create a table in the new schema

```
15 # Create a table in the new schema
16 psql_manipulate(cred = cred_psql_docker,
17                   query_string =
18 "create table intg1.Department (
19   department_code serial primary key,
20   department_name varchar(255),
21   department_location varchar(255),
22   last_update timestamp(0) without time zone default current_timestamp(0)
23 );")
```

Outcome

department
123 department_code
AZ department_name
AZ department_location
⌚ last_update

Write rows in the new table

```
25 # Write rows in the new table
26 psql_manipulate(cred = cred_psql_docker,
27                   query_string =
28 "insert into intg1.Department
29   values (default, 'Computer Science', 'Aarhus C')
30         ,(default, 'Economics and Business Economics', 'Aarhus V')
31         ,(default, 'Law', 'Aarhus C')
32         ,(default, 'Medicine', 'Aarhus C');")
```

Outcome

⌚	123 ↗ department_code	AZ department_name	AZ department_location	⌚ last_update
1	1	Computer Science	Aarhus C	2025-11-04 09:48:01.000
2	2	Economics and Business Economics	Aarhus V	2025-11-04 09:48:01.000
3	3	Law	Aarhus C	2025-11-04 09:48:01.000
4	4	Medicine	Aarhus C	2025-11-04 09:48:01.000

Create an R dataframe

```
35 df <- data.frame(department_name = c("Education", "Chemistry"),
36                     department_location = c("Aarhus N", "Aarhus C"))
• A data.frame is like a table in Excel or a database table.
• It has rows and columns, where each column can have a different type (numbers, text, etc.).
• We do not specify any values for "department_code" and "last_update" as these have default values in the Postgres table.
```

```
> df
  department_name department_location
  1      Education          Aarhus N
  2      Chemistry          Aarhus C
```

A data.frame in R like your df does not automatically appear in DBeaver

To see it in DBeaver, you need to insert it into a Postgres table. Here's how it works:

```
39 # Write the data.frame to a postgres table (columns with default values are skipped)
40 department <- psql_append_df(cred = cred_psql_docker,
41                               schema_name = "intg1",
42                               tab_name = "department",
43                               df = df)
```

```
[1] "Number of rows inserted: 2"
```

	department_code	department_name	department_location	last_update
1	1	Computer Science	Aarhus C	2025-11-04 09:48:01.000
2	2	Economics and Business Economics	Aarhus V	2025-11-04 09:48:01.000
3	3	Law	Aarhus C	2025-11-04 09:48:01.000
4	4	Medicine	Aarhus C	2025-11-04 09:48:01.000
5	5	Education	Aarhus N	2025-11-04 09:55:03.000
6	6	Chemistry	Aarhus C	2025-11-04 09:55:03.000

Read and write to Postgres

Fetch rows into R

```
45 # Fetching rows into R
46 psql_select(cred = cred_pgsql_docker,
47               query_string = "select * from intg1.department;")
```

Outcome

```
> # Fetching rows into R
> psql_select(cred = cred_pgsql_docker,
+               query_string = "select * from intg1.department;")
   department_code      department_name department_location      last_update
1                  1  Computer Science        Aarhus C 2025-11-04 09:48:01
2          2 Economics and Business Economics        Aarhus V 2025-11-04 09:48:01
3                  3                 Law        Aarhus C 2025-11-04 09:48:01
4                  4                Medicine        Aarhus C 2025-11-04 09:48:01
5                  5              Education        Aarhus N 2025-11-04 09:55:03
6                  6             Chemistry        Aarhus C 2025-11-04 09:55:03
```

ETL with R

- On Rapidapi.com we will utilize the API called "Alpha Vantage" • From this API we aim to get intraday stock prices for Microsoft
- First, we will investigate if Alpha Vantage has prices for Microsoft

Go to Rapid.API and subscribe to Alpha Vantage

The screenshot shows the Alpha Vantage API page on RapidAPI. It displays four pricing plans: BA... Current (\$0.00 / mo), PRO (\$49.99 / mo), ULTRA (\$149.99 / mo), and MEGA (\$249.99 / mo). The 'Overview' tab is selected. In the 'Provider Info' section, it shows the API Creator is Alpha Vantage and there are 32521 Subscribers.

Select search endpoint and ask chatgpt to change to httr2

The screenshot shows the Postman interface. A GET request is being prepared to https://alpha-vantage.p.rapidapi.com/query. The 'Code Snippets' tab is active, showing R code for httr2. The 'Headers' section includes an X-RapidAPI-Key header with a redacted value. The 'Body' section is set to 'raw' with JSON content type, containing a query object with 'function': 'SYMBOL_SEARCH' and 'symbol': 'MSFT'.

```
library(httr2)

url <- "https://alpha-vantage.p.rapidapi.com/query"

# Create the request
req <- request(url) %>
  req_headers(
    # Redacted
  ) %>
  req_url_query(
    datatype = "json",
    keywords = "microsoft",
    function = "SYMBOL_SEARCH"
  ) %>
  req_body_raw(content_type = "application/octet-stream")

# Perform the request
resp <- req_perform(req)

# Get response as text
resp_text <- resp_body_string(resp)
print(resp_text)
```

Copy code into R

The screenshot shows the RStudio code editor with the copied R code pasted into the script pane. The code uses httr2 to make a GET request to the Alpha Vantage API, specifying the X-RapidAPI-Key header and the query parameters for Microsoft's symbol.

```
library(RPostgres)
library(DBI)
library(tidyverse)
library(httr2)
library(lubridate)
# Investigate which symbols we can search for -----
library(httr2)
url <- "https://alpha-vantage.p.rapidapi.com/query"
# Create the request
library(httr2)
url <- "https://alpha-vantage.p.rapidapi.com/query"
req <- request(url) %>
  req_headers(
    `X-RapidAPI-Key` = # Redacted,
    `X-RapidAPI-Host` = # Redacted
  ) %>
  req_url_query(
    datatype = "json",
    keywords = "microsoft",
    function = "SYMBOL_SEARCH"
  )
# Print the Result
resp <- req_perform(req)
resp_text <- resp_body_string(resp)
print(resp_text)
```

Convert to json

```
27 # convert to json  
28 symbols <- resp %>%  
29   resp_body_json()
```

See if there is a match

```
31 #see if there is a Match  
32 symbols$bestMatches[[1]]  
33 symbols$bestMatches[[2]]  
24
```

Outcome

```
> symbols$bestMatches[[1]]  
$`1. symbol`  
[1] "MSFO.FRK"  
  
$`2. name`  
[1] "MICROSOFT CORP. CDR"  
  
$`3. type`  
[1] "Equity"  
  
$`4. region`  
[1] "Frankfurt"  
  
$`5. marketOpen`  
[1] "08:00"  
  
$`6. marketClose`  
[1] "20:00"  
  
$`7. timezone`  
[1] "UTC+02"  
  
$`8. currency`  
[1] "EUR"  
  
$`9. matchScore`  
[1] "0.6429"
```

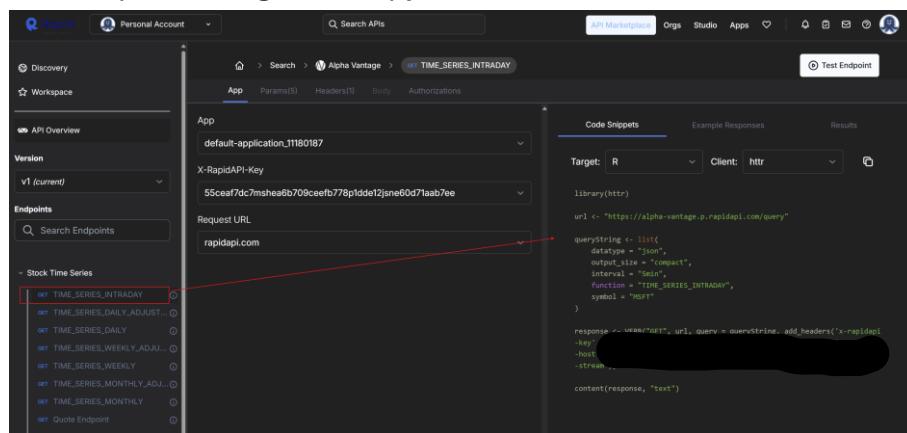


```
> symbols$bestMatches[[2]]  
$`1. symbol`  
[1] "MSFT"  
  
$`2. name`  
[1] "Microsoft Corporation"  
  
$`3. type`  
[1] "Equity"  
  
$`4. region`  
[1] "United States"  
  
$`5. marketOpen`  
[1] "09:30"  
  
$`6. marketClose`  
[1] "16:00"  
  
$`7. timezone`  
[1] "UTC-04"  
  
$`8. currency`  
[1] "USD"  
  
$`9. matchScore`  
[1] "0.6154"
```

➔ We can see alpha Vantage has prices for the Microsoft stock

Extract with R

Go to Alpha Vantage and copy the code so we can check it minutey



The screenshot shows the RapidAPI interface for the Alpha Vantage API. The endpoint selected is `TIME_SERIES_INTRADAY`. The request URL is `https://alpha-vantage.p.rapidapi.com/query`. The request body contains the following R code:

```
url1 <- "https://alpha-vantage.p.rapidapi.com/query"  
queryString <- list(  
  datatype = "json",  
  outputsize = "compact",  
  interval = "5min",  
  function = "TIME_SERIES_INTRADAY",  
  symbol = "MSFT"  
)  
  
response <- uread("url1", url, query = querystring, add_headers("x-rapidapi-key", host, stream, ...))  
content(response, "text")
```

Ask chatgpt to make it httr and output as json

Here's the equivalent httr2 code:

```
r
library(httr2)

url <- "https://alpha-vantage.p.rapidapi.com/query"

# Create the request
req <- request(url) %>%
  req_headers(
    `x-rapidapi-key` = [REDACTED],
    `x-rapidapi-host` = [REDACTED]
  ) %>%
  req_url_query(
    datatype = "json",
    output_size = "compact",
    interval = "5min",
    `function` = "TIME_SERIES_INTRADAY", # backticks because "function" is reserved
    symbol = "MSFT"
  )

# Perform the request
resp <- req_perform(req)

# Get response as JSON
data <- resp_body_json(resp)

# Inspect the result
str(data)
```

Copy to R we want minute so we change interval to 1min

```
46 # Create the request
47 req <- request(url) %>%
48   req_headers(
49     `x-rapidapi-key` = [REDACTED],
50     `x-rapidapi-host` = [REDACTED]
51   ) %>%
52   req_url_query(
53     datatype = "json",
54     output_size = "compact",
55     interval = "1min",
56     "function" = "TIME_SERIES_INTRADAY", # backticks because "function" is reserved
57     symbol = "MSFT"
58   )
59 # Perform the request
60 resp <- req_perform(req)
61 # Get response as JSON
62 data <- resp_body_json(resp)
63 # Inspect the result
64 str(data)
```

Outcome

```
..$ 2025-11-03 18:23:00:List of 5
...$ 1. open : chr "516.7900"
...$ 2. high : chr "516.8200"
...$ 3. low : chr "516.6200"
...$ 4. close : chr "516.8200"
...$ 5. volume: chr "146"
..$ 2025-11-03 18:22:00:List of 5
...$ 1. open : chr "516.7850"
...$ 2. high : chr "516.9700"
...$ 3. low : chr "516.6250"
...$ 4. close : chr "516.7900"
...$ 5. volume: chr "25"
..$ 2025-11-03 18:21:00:List of 5
...$ 1. open : chr "516.8500"
...$ 2. high : chr "516.8500"
...$ 3. low : chr "516.6700"
...$ 4. close : chr "516.8000"
...$ 5. volume: chr "37"
.. [list output truncated]
```

The Microsoft prices are given in America/New_York time. As we might wish to have other price data (given in other time zones), we transform the timestamps to UTC time.

```

67 # TRANSFORM timestamp to UTC time
68 timestamp <- lubridate::ymd_hms(names(dat$`Time Series (1min)`), tz = "America/New_York")
69 timestamp <- format(timestamp, tz = "UTC") Converts the timestamps from New York time to UTC, which is standard for storing time-series financial data.
70 # Prepare an empty data.frame to hold results
71 df <- tibble(timestamp = timestamp,
72               open = NA, high = NA, low = NA, close = NA, volume = NA)
73 # TRANSFORM data into a data.frame
74 for (i in 1:nrow(df)) { What's happening:
75   df[i, 1] <- as.data.frame(dat$`Time Series (1min)`[[i]]) dat$Time Series (1min)[[i]] → grabs the i-th timestamp's data. Example:
76 } So after the loop: df$timestamp → UTC timestamps as.data.frame(...) → converts this small list into a one-row data frame.
77 df$open, df$high, df$low, df$close, df$volume → the actual stock data for each minute df[i, -1] < ... → assigns the values to the i-th row, excluding the timestamp column (-1).

```

Intermediate step –

creating a Postgres table (Normally, the table would already exist when doing ETL)

```

78 # Create table in Postgres -----
79 # Put the credentials in this script
80 # Never push credentials to git!! --> use .gitignore on .credentials.R
81 source("credentials.R")
82 # Function to send queries to Postgres
83 source("psql_queries.R")
84 # Create a new schema in Postgres on docker
85 psql_manipulate(cred = cred_psql_docker,
86                 query_string = "CREATE SCHEMA intg2;")
87 # Create a table in the new schema
88 psql_manipulate(cred = cred_psql_docker,
89                 query_string =
90                   "create table intg2.prices (
91                     id serial primary key,
92                     timestamp timestamp(0) without time zone ,
93                     open numeric(30,4),
94                     high numeric(30,4),
95                     low numeric(30,4),
96                     close numeric(30,4),
97                     volume numeric(30,4));")

```

Outcome

	prices
123	id
123	timestamp
123	open
123	high
123	low
123	close
123	volume

Load price data into the Table

```
99 # LOAD price data -----
100 psql_append_df(cred = cred_psql_docker,
101     schema_name = "intg2",
102     tab_name = "prices",
103     df = df)
```

[1] "Number of rows inserted: 100"

Reminder: df is the table we created above

```
# Prepare an empty data.frame to hold results
df <- tibble(timestamp = timestamp,
             open = NA, high = NA, low = NA, close = NA, volume = NA)
# TRANSFORM data into a data.frame
for (i in 1:nrow(df)) {
  df[i,-1] <- as.data.frame(dat$`Time Series (1min)`[[i]])
}
```

Outcome

Grid	① id	⌚ timestamp	⌚ open	⌚ high	⌚ low	⌚ close	⌚ volume
Text	1	2025-11-04 00:59:00.000	516.35	516.35	516.01	516.11	420
Text	2	2025-11-04 00:58:00.000	516.01	516.35	516.0001	516.0001	667
Text	3	2025-11-04 00:57:00.000	516.2	516.2	516	516.0006	208
Text	4	2025-11-04 00:56:00.000	516.125	516.2499	516	516.2	436
Text	5	2025-11-04 00:55:00.000	516.25	516.2987	516.125	516.125	340
Text	6	2025-11-04 00:54:00.000	516.3	516.3	516.2	516.3	82
Text	7	2025-11-04 00:53:00.000	516.35	516.35	516.25	516.25	111

Check that we can fetch the data again in R

```
105 # Check results -----
106 # Check that we can fetch the data again
107 psql_select(cred = cred_psql_docker,
108             query_string =
109             "select * from intg2.prices")
```

Outcome

Console	Terminal	Background Jobs
R 4.5.1 · ~/Data Integration/ ↗		
93	93	2025-11-03 23:27:00 517.0250 517.2300 516.9600 516.9600 1226
94	94	2025-11-03 23:26:00 516.8700 517.0300 516.8201 517.0150 840
95	95	2025-11-03 23:25:00 516.8200 516.9900 516.6300 516.8212 341
96	96	2025-11-03 23:24:00 516.8200 516.8200 516.6200 516.7200 36
97	97	2025-11-03 23:23:00 516.7900 516.8200 516.6200 516.8200 146
98	98	2025-11-03 23:22:00 516.7850 516.9700 516.6250 516.7900 25
99	99	2025-11-03 23:21:00 516.8500 516.8500 516.6700 516.8000 37
100	100	2025-11-03 23:20:00 516.6700 516.9800 516.6200 516.8400 43

Delete the schema if wished

```
--#
111 # If you wish, you can delete the schema (all the price data) from Postgres
112 psql_manipulate(cred = cred_psql_docker,
113                   query_string = "drop SCHEMA intg2 cascade;")
```

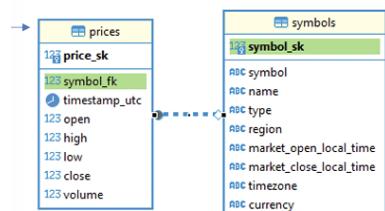
Data Integration II

ETL with Scheduling and Visualization

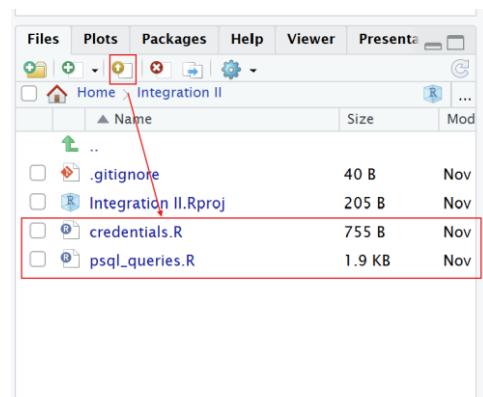
- Goal
 - You want to create a workflow that:
 - Extracts Microsoft (MSFT) and Tesla (TSLA) stock prices from the internet (e.g., Alpha Vantage API).
 - Transforms the data into a clean, usable format.
 - Loads the data into a database (Postgres in your case).
 - Visualizes the stock prices in charts.
 - Automatically updates every 15 minute

Steps of the Integration and Visualization

1. Prepare the Tables in Postgres



Upload the files credentials and psql_queries into the Project

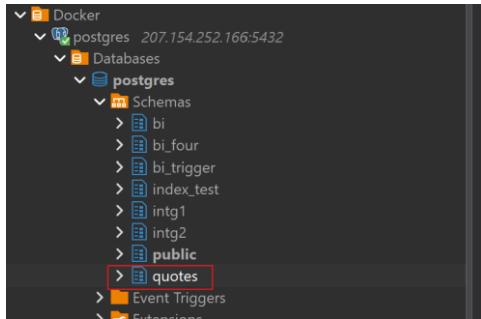


Modify the credentials file

```
1 cred_psql_docker <- list(dbname = 'mydb', host = '127.0.0.1', user = 'myuser', pass = 'mypassword', port = 5432)
2
3 cred_psql_150 <- list(dbname = 'mydb', host = '127.0.0.1', user = 'myuser', pass = 'mypassword', port = 6432)
4
5 cred_psql_150_dvd <- list(dbname = 'mydb', host = '127.0.0.1', user = 'myuser', pass = 'mypassword', port = 6432)
```

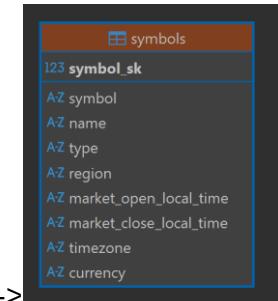
Create a new schema in Postgres on docker

```
12 psql_manipulate(cred = cred_psql_docker,
13                     query_string = "CREATE SCHEMA quotes;")
```



Create Tables

```
15 # Create symbols table with symbols metadata
16 psql_manipulate(cred = cred_psql_docker,
17                     query_string =
18                         "create table quotes.symbols (
19     symbol_sk serial primary key,
20     symbol varchar(50),
21     name varchar(150),
22     type varchar(50),
23     region varchar(100),
24     market_open_local_time varchar(20),
25     market_close_local_time varchar(20),
26     timezone varchar(50),
27     currency varchar(20));")
```



Populate table with metainformation about Microsoft and Tesla

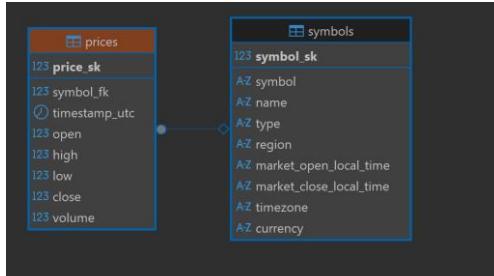
```
28 # Populate table with metainformation about Microsoft and Tesla -----
29 psql_manipulate(cred = cred_psql_docker,
30                     query_string =
31                         "insert into quotes.symbols
32                             values (default, 'MSFT', 'Microsoft Corporation', 'Equity', 'United States', '09:30', '16:00', 'UTC-04', 'currency'),
33                             (default, 'TSLA', 'Tesla Inc', 'Equity', 'United States', '09:30', '16:00', 'UTC-04', 'currency');")
```

The screenshot shows the 'symbols' table in pgAdmin with the following data:

	symbol_sk	symbol	name	type	region	market_open_local_time
1	1	MSFT	Microsoft Corporation	Equity	United States	09:30
2	2	TSLA	Tesla Inc	Equity	United States	09:30

Create table to hold price data

```
39 # Create table to hold price data -----
40 psql_manipulate(cred = cred_psql_docker,
41                     query_string =
42                         "create table quotes.prices (
43     price_sk serial primary key,
44     symbol_fk integer,
45     timestamp_utc timestamp(0) without time zone ,
46     open numeric(30,4),
47     high numeric(30,4),
48     low numeric(30,4),
49     close numeric(30,4),
50     volume numeric(30,4),
51     constraint fk_symbol foreign key (symbol_fk)
52     references quotes.symbols(symbol_sk));")
```



Populate price table with Microsoft prices

The screenshot shows the Alpha Vantage API documentation interface. On the left, there's a sidebar with 'Discovery', 'Workspace', 'API Overview', 'Version' (set to 'v1 (current)'), and 'Endpoints'. Under 'Endpoints', there's a 'Stock Time Series' section with several endpoints listed: 'TIME_SERIES_INTRADAY', 'TIME_SERIES_DAILY_ADJUSTED', 'TIME_SERIES_DAILY', 'TIME_SERIES_WEEKLY_ADJUSTED', 'TIME_SERIES_WEEKLY', 'TIME_SERIES_MONTHLY_ADJUSTED', and 'TIME_SERIES_MONTHLY'. On the right, under 'App', the 'default-application_11180187' app is selected. The 'X-RapidAPI-Key' field contains a placeholder key. The 'Request URL' dropdown is set to 'rapidapi.com'. Below these, the 'Code Snippets' tab is active, showing R code to query the API. The target is set to 'R' and the client to 'httr'. The code uses the httr library to build a query string and then performs the request to get a JSON response.

(change to httr2 and json output)

```
In [R]:  
r  
library(httr2)  
library(jsonlite)  
  
url <- "https://alpha-vantage.p.rapidapi.com/query"  
  
# Build the request  
req <- request(url) %>  
req_url_query(  
  datatype = "json",  
  output_size = "compact",  
  interval = "5min",  
  function = "TIME_SERIES_INTRADAY",  
  symbol = "MSFT")  
req %>  
req_headers(  
  
)  
  
# Perform the request  
resp <- req_perform(req)  
  
# Parse JSON response  
data <- resp %>% resp_body_json()  
  
# View the output  
print(data)
```

53 # Populate price table with Microsoft prices -----
54 url <- "https://alpha-vantage.p.rapidapi.com/query"
55 # Build the request
56 req <- request(url) %>
57 req_url_query(
 datatype = "json",
 output_size = "compact",
 interval = "15min",
 function = "TIME_SERIES_INTRADAY",
 symbol = "MSFT")
58 req %>
req_headers(
59)
60 # Perform the request
61 resp <- req_perform(req)
62 # Parse JSON response
63 data <- resp %>% resp_body_json()
64 # View the output
65 print(data)

(transform data to UTC)

```
75 # Transform timestamp to UTC time  
76 timestamp <- lubridate::ymd_hms(names(data$`Time Series (15min)`), tz = "America/New_York")  
77 timestamp <- format(timestamp, tz = "UTC")
```

(Prepare data.frame to hold results)

```
79 df <- tibble(timestamp_utc = timestamp,  
80                 symbol_fk = 1, #  
81                 open = NA, high = NA, low = NA, close = NA, volume = NA)
```

(TRANSFORM data into a data.frame)

```
82 # TRANSFORM data into a data.frame  
83 for (i in 1:nrow(df)) {  
84   df[i,-c(1,2)] <- as.data.frame(data$`Time Series (15min)`[[i]])  
85 }
```

(Load Microsoft price data into prices)

```
86 # Load Microsoft price data into prices  
87 psql_append_df(cred = cred_psql_docker,  
88                 schema_name = "quotes",  
89                 tab_name = "prices",  
90                 df = df)
```

Do the same with Tesla
(ask chatgpt to change the Rapidapi to Tesla)

```
f
library(httr)
library(jsonlite)

url <- "https://alpha-vantage.p.rapidapi.com/query"

# Build the request
req <- request(url) %>%
  req_url_query(
    datatype = "json",
    output_size = "compact",
    interval = "5min",
    function = "TIME_SERIES_INTRADAY",
    symbol = "TSLA" # changed to Tesla
  ) %>%
  req_headers(
    [REDACTED]
  )

# Perform the request
resp <- req_perform(req)

# Parse JSON response
data <- resp %>% resp_body_json()

# View the output
print(data)

~ | Populate price table with Tesla prices -----
91 url <- "https://alpha-vantage.p.rapidapi.com/query"
92 # Build the request
93 req <- request(url) %>%
94   req_url_query(
95     datatype = "json",
96     output_size = "compact",
97     interval = "15min",
98     function = "TIME_SERIES_INTRADAY",
99     symbol = "TSLA" # changed to Tesla
100 ) %>%
101 req_headers(
102   'x-rapidapi-key': [REDACTED],
103   'x-rapidapi-host': [REDACTED]
104 )
105 # Perform the request
106 resp <- req_perform(req)
107 # Parse JSON response
108 data <- resp %>% resp_body_json()
109 # View the output
110 print(data)
111
112
```

Transform timestamp to UTC time

```
114 # Transform timestamp to UTC time
115 timestamp <- lubridate::ymd_hms(names(data$`Time Series (15min)`), tz = "America/New_York")
116 timestamp <- format(timestamp, tz = "UTC")
```

Prepare data.frame to hold results

```
117 # Prepare data.frame to hold results
118 df <- tibble(timestamp_utc = timestamp,
119               symbol_fk = 2, #
120               open = NA, high = NA, low = NA, close = NA, volume = NA)
```

transform data into a data.frame

```
121 # transform data into a data.frame
122 for (i in 1:nrow(df)) {
123   df[i,-c(1,2)] <- as.data.frame(data$`Time Series (15min)`[[i]])
124 }
```

Load Tesla price data into prices

```
125 # Load Tesla price data into prices
126 psql_append_df(cred = cred_psql_docker,
127                 schema_name = "quotes",
128                 tab_name = "prices",
129                 df = df)
```

Check if it worked

	id	symbol_fk	timestamp_utc	open	high	low	close	
Text	1	1	2025-11-05 00:45:00.000	512.7	513.39	512.17	512.58	
Text	2	1	2025-11-05 00:30:00.000	513.275	513.45	512.4	512.78	
Text	3	1	2025-11-05 00:15:00.000	513.99	514.35	513.0514	513.275	
Text	4	1	2025-11-05 00:00:00.000	514.7	514.75	513.32	513.99	
Text	5	1	2025-11-04 23:45:00.000	515.2	515.325	514.3	514.406	
Text	6	1	2025-11-04 23:30:00.000	514.42	515.8	514.39	515.2	
Text	7	1	2025-11-04 23:15:00.000	514.9727	514.99	514.3	514.39	
Text	8	1	2025-11-04 23:00:00.000	514.67	515.57	514.0665	514.9727	

Or

```
130 # Check that we have populated the tables as intended -----
131 psql_select(cred = cred_psql_docker,
132             query_string = "select * from quotes.prices")
133
```

```
R 4.5.1 . ~/Integration II/ 
108   108    2 2025-11-04 23:00:00 443.4900 444.0000 442.0000 442.7700 34320
109   109    2 2025-11-04 22:45:00 443.4000 443.9000 443.2100 443.4700 36284
110   110    2 2025-11-04 22:30:00 444.2632 444.3100 443.2100 443.4200 54370
111   111    2 2025-11-04 22:15:00 444.2000 463.1606 415.8999 444.2600 52774
112   112    2 2025-11-04 22:00:00 443.8000 444.5000 443.8000 444.1500 96877
113   113    2 2025-11-04 21:45:00 443.0400 444.3900 442.3800 443.8650 133682
114   114    2 2025-11-04 21:30:00 442.8000 445.2856 442.0000 443.0500 163903
115   115    2 2025-11-04 21:15:00 444.2600 449.9690 442.1000 442.7200 173487
116   116    2 2025-11-04 21:00:00 444.2200 468.3700 442.0000 442.3400 12144173
117   117    2 2025-11-04 20:45:00 448.5764 449.4617 443.6000 444.2300 5279086
118   118    2 2025-11-04 20:30:00 448.5200 448.8476 447.1875 448.5501 2083579
119   119    2 2025-11-04 20:15:00 447.3900 448.8500 446.6100 448.5000 1882475
120   120    2 2025-11-04 20:00:00 446.4100 448.1000 446.0300 447.3800 2929479
121   121    2 2025-11-04 19:45:00 446.1600 446.7400 443.8600 446.4200 3381652
122   122    2 2025-11-04 19:30:00 449.2800 449.4300 446.0110 446.1900 2076762
123   123    2 2025-11-04 19:15:00 448.2900 450.5500 448.0600 449.3300 1865227
124   124    2 2025-11-04 19:00:00 450.2800 450.9700 448.1450 448.3292 1819178
125   125    2 2025-11-04 18:45:00 448.3350 451.5400 447.8800 450.2700 2513351
[ reached 'max' / getOption("max.print") -- omitted 75 rows ]
```

2. Script to be scheduled

Load functions to communicate with Postgres

```
6 # Load credentials
7 source("credentials.R")
8 # Load functions to communicate with Postgres
9 source("psql_queries.R")
```

Extract Microsoft prices

```
library(httr)
url <- "https://alpha-vantage.p.rapidapi.com/query"
queryString <- list(
  datatype = "json",
  output_size = "compact",
  interval = "1min",
  function = "TIME_SERIES_INTRADAY",
  symbol = "MSFT"
)
response <- httr::GET(url, query = queryString)
content(response, "text")
```

(change to httr2 and json output)

```

In [5]: r
Code kopieren

library(httr2)
library(jsonlite)

url <- "https://alpha-vantage.p.rapidapi.com/query"

# Build the request
req <- request(url) %>%
  req_url_query(
    datatype = "json",
    output_size = "compact",
    interval = "5min",
    function = "TIME_SERIES_INTRADAY",
    symbol = "MSFT"
  ) %>%
  req_headers(
    # ...
  )

# Perform the request
resp <- req_perform(req)

# Parse JSON response
data <- resp %>% resp_body_json()

# View the output
print(data)

```

53+ # Populate price table with Microsoft prices
54 url <- "https://alpha-vantage.p.rapidapi.com/query"
55 # Build the request
56 req <- request(url) %>%
57 req_url_query(
58 datatype = "json",
59 output_size = "compact",
60 interval = "5min",
61 function = "TIME_SERIES_INTRADAY",
62 symbol = "MSFT"
63) %>%
64 req_headers(
65 # ...
66)
67 # ...
68 # Perform the request
69 resp <- req_perform(req)
70 # Parse the response
71 data <- resp %>% resp_body_json()
72 # View the output
73 print(data)

(transform data to UTC)

```

75 # Transform timestamp to UTC time
76 timestamp <- lubridate::ymd_hms(names(data$`Time Series (15min)`), tz = "America/New_York")
77 timestamp <- format(timestamp, tz = "UTC")

```

(Prepare data.frame to hold results)

```

79 df <- tibble(timestamp_utc = timestamp,
80                 symbol_fk = 1, #
81                 open = NA, high = NA, low = NA, close = NA, volume = NA)

```

(TRANSFORM data into a data.frame)

```

82 # TRANSFORM data into a data.frame
83 for (i in 1:nrow(df)) {
84   df[i,-c(1,2)] <- as.data.frame(data$`Time Series (15min)`[[i]])
85 }

```

From our database, we fetch the most recent timestamp for a Microsoft price

```

44 latest_tmstmp <- psql_select(cred = cred_psql_docker,
45                               query_string =
46                               "select timestamp_utc
47                               from quotes.prices
48                               where symbol_fk = 1
49                               order by timestamp_utc desc
50                               limit 1;")

```

Limit 1

Only new datapoints should be loaded to database

```

51 # Only new datapoints should be loaded to database
52 df <- df[df$timestamp_utc > latest_tmstmp[[1]],]

```

Insert new Microsoft price data into the database

```

53 # Load price data
54 print(paste0(round(Sys.time()), ": Updating Microsoft prices"))
55
56 psql_append_df(cred = cred_psql_docker,
57                 schema_name = "quotes",
58                 tab_name = "prices",
59                 df = df)

```

Do the same with Tesla
(ask chatgpt to change the Rapidapi to Tesla)

```
f
library(httr2)
library(jsonlite)

url <- "https://alpha-vantage.p.rapidapi.com/query"

# Build the request
req <- request(url) %>%
  req_url_query(
    datatype = "json",
    output_size = "compact",
    interval = "5min",
    function = "TIME_SERIES_INTRADAY",
    symbol = "TSLA" # changed to Tesla
  ) %>%
  req_headers(
    'X-RapidAPI-Host' = "alpha-vantage.p.rapidapi.com"
  )

# Perform the request
resp <- req_perform(req)

# Parse JSON response
data <- resp %>% resp_body_json()

# View the output
print(data)

~ | [ Populate price table with Tesla prices -----
91 url <- "https://alpha-vantage.p.rapidapi.com/query"
92 # Build the request
93 req <- request(url) %>%
94   req_url_query(
95     datatype = "json",
96     output_size = "compact",
97     interval = "15min",
98     "function" = "TIME_SERIES_INTRADAY",
99     symbol = "TSLA" # changed to Tesla
100 ) %>%
101 req_b
102 'X-RapidAPI-Host' = "alpha-vantage.p.rapidapi.com"
103 'X-RapidAPI-Key' = "REDACTED"
104 )
105
106 # Perform the request
107 resp <- req_perform(req)
108 # Parse JSON response
109 data <- resp %>% resp_body_json()
110 # View the output
111 print(data)
```

Transform timestamp to UTC time

```
114 # Transform timestamp to UTC time
115 timestamp <- lubridate::ymd_hms(names(data$`Time Series (15min)`), tz = "America/New_York")
116 timestamp <- format(timestamp, tz = "UTC")
```

Prepare data.frame to hold results

```
117 # Prepare data.frame to hold results
118 df <- tibble(timestamp_utc = timestamp,
119               symbol_fk = 2, #
120               open = NA, high = NA, low = NA, close = NA, volume = NA)
```

transform data into a data.frame

```
121 # transform data into a data.frame
122 for (i in 1:nrow(df)) {
123   df[i,-c(1,2)] <- as.data.frame(data$`Time Series (15min)`[[i]])
124 }
```

Get most recent datapoint from database

```
-- | 93 # Get most recent datapoint from database
94 latest_tmstmp <- psql_select(cred = cred_psql_docker,
95                               query_string =
96                               "select timestamp_utc
97                               from quotes.prices
98                               where symbol_fk = 2
99                               order by timestamp_utc desc
100                               limit 1;")
```

Only new datapoints should be loaded to database

```
df <- df[df$timestamp_utc > latest_tmstmp[[1]],]
```

Load price data

```
104 print(paste0(round(Sys.time()), ": Updating Tesla prices"))
105
106 psql_append_df(cred = cred_psql_docker,
107   schema_name = "quotes",
108   tab_name = "prices",
109   df = df)
```

3. Test the script to be scheduled

Before sending a code to production, you should always perform tests to validate the code

Load functions

```
6 # Load credentials
7 source("credentials.R")
8 # Load functions to communicate with Postgres
9 source("psql_queries.R")
```

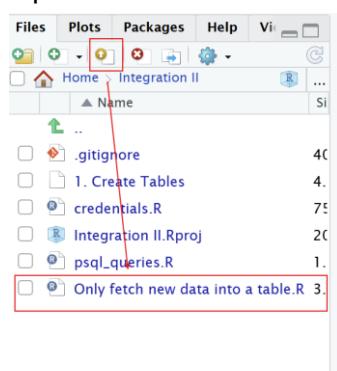
Delete some observations from the table Microsoft

```
12 psql_manipulate(cred = cred_psql_docker,
13   query_string =
14     "delete
15       from quotes.prices
16       where
17         price_sk = (select price_sk
18                     from quotes.prices
19                     where symbol_fk = 1
20                     order by timestamp_utc desc
21                     limit 1);")
```

Check most recent prices

```
22 # Check most recent prices
23 psql_select(cred = cred_psql_docker,
24   query_string =
25     "select timestamp_utc
26       from quotes.prices
27       where symbol_fk = 1
28       order by timestamp_utc desc
29       limit 5;")
```

Upload the file we created above for fetching new data



Run the scheduled function 1 time

```
31 source("Only fetch new data into a table.R")
```

Check that things has been reinserted

```
33 psql_select(cred = cred_psql_docker,
34             query_string =
35             "select *
36             from quotes.prices
37             where symbol_fk = 1
38             order by timestamp_utc desc
39             limit 5;")  
+-----+
| price_sk symbol_fk      timestamp_utc      open      high      low      close     volume |
| 1        201            1 2025-11-05 00:45:00 512.700 513.390 512.1700 512.580 13871   |
| 2        2               1 2025-11-05 00:30:00 513.275 513.450 512.4000 512.780 6338    |
| 3        3               1 2025-11-05 00:15:00 513.990 514.350 513.0514 513.275 6467    |
| 4        4               1 2025-11-05 00:00:00 514.700 514.750 513.3200 513.990 4816    |
| 5        5               1 2025-11-04 23:45:00 515.200 515.325 514.3000 514.406 6431    |
> |
```

Check updating Tesla

Delete some observations from the table

```
41 # Delete some observations from the table
42 psql_manipulate(cred = cred_psql_docker,
43                   query_string =
44                   "delete
45                   from quotes.prices
46                   where
47                   price_sk = (select price_sk
48                               from quotes.prices
49                               where symbol_fk = 2
50                               order by timestamp_utc desc
51                               limit 1);")
```

Check most recent prices

```
52 # Check most recent prices
53 psql_select(cred = cred_psql_docker,
54             query_string =
55             "select *
56             from quotes.prices
57             where symbol_fk = 2
58             order by timestamp_utc desc
59             limit 5;")  
+-----+
| price_sk symbol_fk      timestamp_utc      open      high      low      close     volume |
| 1        102            2 2025-11-05 00:30:00 441.050 441.1100 440.00 440.00 90483   |
| 2        103            2 2025-11-05 00:15:00 441.500 441.8111 440.91 441.09 59937   |
| 3        104            2 2025-11-05 00:00:00 441.949 442.2200 441.00 441.50 66933   |
| 4        105            2 2025-11-04 23:45:00 442.000 442.3500 441.42 441.90 70457   |
| 5        106            2 2025-11-04 23:30:00 442.570 442.7267 441.66 441.96 122507   |
```

Run the scheduled function 1 time

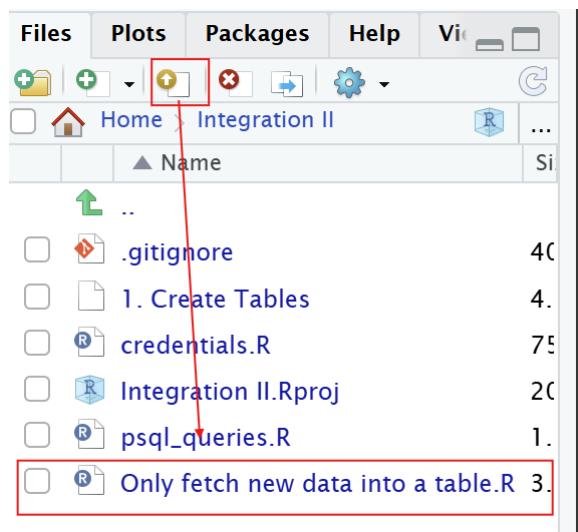
```
60 # Run the scheduled function 1 time
61 source("Only fetch new data into a table.R")
[1] "2025-11-05 17:13:06: Updating Tesla prices"
[1] "Number of rows inserted: 1"
```

Check that things has been reinserted

```
63 psql_select(cred = cred_psql_docker,
64             query_string =
65             "select *
66             from quotes.prices
67             where symbol_fk = 2
68             order by timestamp_utc desc
69             limit 5;")
   price_sk symbol_fk      timestamp_utc    open    high   low  close volume
1       202            2 2025-11-05 00:45:00 440.070 441.000 439.45 439.50 175999
2       102            2 2025-11-05 00:30:00 441.050 441.1100 440.00 440.00 90483
3       103            2 2025-11-05 00:15:00 441.500 441.8111 440.91 441.09 59937
4       104            2 2025-11-05 00:00:00 441.949 442.2200 441.00 441.50 66933
5       105            2 2025-11-04 23:45:00 442.000 442.3500 441.42 441.90 70457
```

4. Setting the schedule

Upload the script from step 2



Add cron job for fetching price data

```
3 cmd <- cron_rscript(rscript = "Only fetch new data into a table.R")
```

Must run every 15 minutes

```
5 cron_add(cmd, frequency = '0,15,30,45 * * * *', id = 'job3')
```

Check the schedule

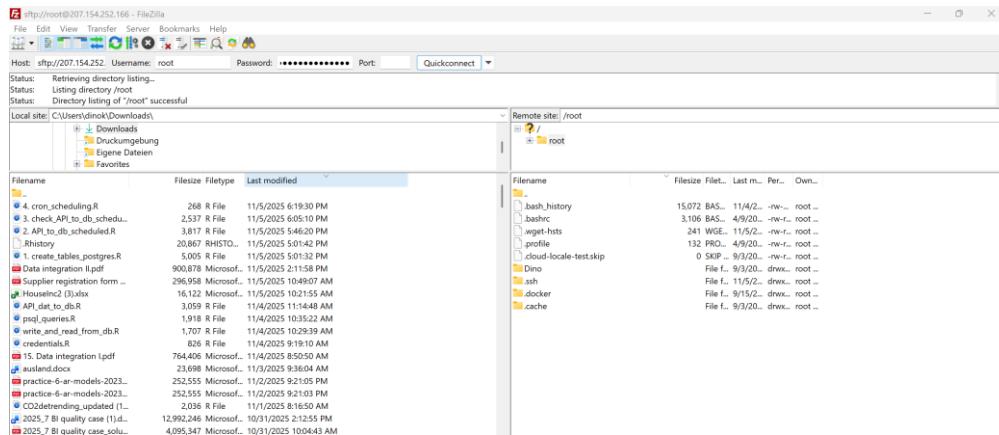
```
7 cron_ls()
```

```
> cron_ls()
## cronR job
## id: job3
## tags:
## desc:
0,15,30,45 * * * * /usr/local/lib/R/bin/Rscript '/home/rstudio/Integration II/Only fetch new data into a table.R' >> '/home/rstudio/Integration II/Only fetch new data into a table.log' 2>&1
```

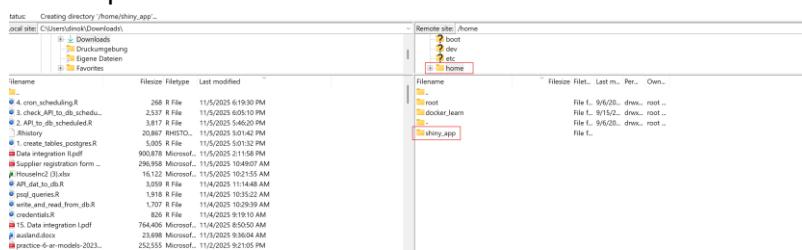
- We have now automated the data integration task
- We will automatically get new price data written to our database
- More symbols could be added (but then we would likely reach our API limit for our free API)
- To complete the pipeline, we will set up a shiny server to visualize our stock data
- It is not important how the Shinyapp itself is setup – you will get more into this when Martin takes over next week
- The way we build the new image and run the container is, however, instructive for this part of the course
- The Shiny server will fetch the data from the PostgreSQL server and expose the data on a web address

5. Setting up the Shiny server

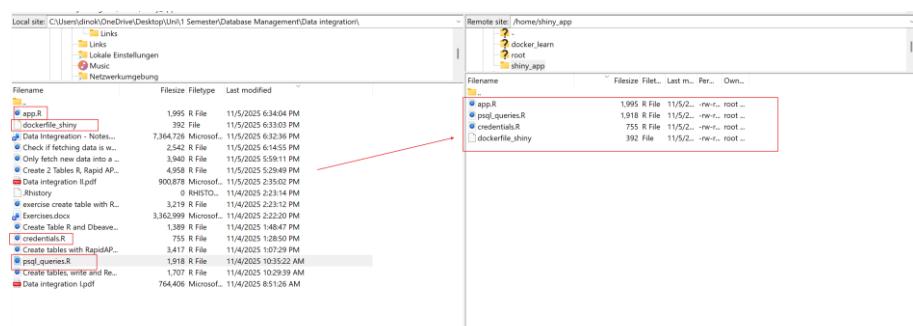
Connect to filezilla



Create a directory (e.g. using FileZilla) called “shiny_app” under the “home” directory on the droplet



Move the files “.credentials.R”, “app.R”, “dockerfile_shiny”, and “psql_queries.R” into this folder



If you have an existing “rshiny” container running, then remove it using “docker rm -rshiny” (using the droplet console)

```
Run 'docker --help' for more information
root@ubuntu-s-1vcpu-1gb-fral-01:/home/shiny_app# docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS               NAMES
1d90cf679838        rstudio:1.0.3    "/init"            7 days ago         Up 7 days          0.0.0.0:8787->8787/tcp, ::1:8787->8787/tcp   rstudio
16da47902ddc        postgres           "docker-entrypoint.s_..." 6 weeks ago       Up 6 weeks          0.0.0.0:5432->5432/tcp, ::1:5432->5432/tcp   postgres
7e30e982ce93        rocker/shiny      "/init"            2 months ago       Up 2 months         0.0.0.0:3838->3838/tcp, ::1:3838->3838/tcp   rshiny
```

→ R shiny container there so remove

Navigate to “/home/shiny_app”

```
root@ubuntu-s-1vcpu-1gb-fral-01:/# cd ..
root@ubuntu-s-1vcpu-1gb-fral-01:/# ls -l
total 64
lrwxrwxrwx  1 root root    7 Apr  9  2025 bin  -> usr/bin
drwxr-xr-x  5 root root  4096 Nov  5 06:49 boot
drwxr-xr-x 18 root root  3920 Sep  3 10:44 dev
drwxr-xr-x 117 root root  4096 Oct 31 06:27 etc
drwxr-xr-x  6 root root  4096 Nov  5 17:31 home
lrwxrwxrwx  1 root root    7 Apr  9  2025 lib  -> usr/lib
lrwxrwxrwx  1 root root    9 Apr  9  2025 lib64 -> usr/lib64
drwx----- 2 root root 16384 Jul  1 19:57 lost+found
drwxr-xr-x  2 root root  4096 Jul  1 19:54 media
drwxr-xr-x  2 root root  4096 Jul  1 19:54 mnt
drwxr-xr-x  4 root root  4096 Sep  3 12:40 opt
dr-xr-xr-x 201 root root     0 Sep  3 10:44 proc
drwx----- 6 root root  4096 Sep 15 08:52 root
drwxr-xr-x 34 root root 1080 Nov  5 17:37 run
lrwxrwxrwx  1 root root    8 Apr  9  2025 sbin -> usr/sbin
drwxr-xr-x  2 root root  4096 Oct 29 16:17 scheduling
drwxr-xr-x  2 root root  4096 Sep  3 10:44 snap
drwxr-xr-x  2 root root  4096 Jul  1 19:54 srv
dr-xr-xr-x 13 root root     0 Sep  6 06:51 sys
drwxrwxrwt  7 root root  140 Nov  5 15:26 tmp
drwxr-xr-x 12 root root  4096 Sep  6 07:35 usr
drwxr-xr-x 13 root root  4096 Sep  3 10:44 var
root@ubuntu-s-1vcpu-1gb-fral-01:/# cd home
root@ubuntu-s-1vcpu-1gb-fral-01:/home# ls -l
total 16
drwxr-xr-x 2 root root 4096 Sep  6 12:44 -
drwxr-xr-x 2 root root 4096 Sep 15 20:05 docker_learn
drwxr-xr-x 3 root root 4096 Sep  6 09:10 root
drwxr-xr-x 2 root root 4096 Nov  5 17:36 shiny_app
root@ubuntu-s-1vcpu-1gb-fral-01:/home# cd shiny_app
root@ubuntu-s-1vcpu-1gb-fral-01:/home/shiny_app#
```

build the image by running “docker build --tag shiny:1.0.0 -f dockerfile_shiny .”

```
root@ubuntu-s-1vcpu-1gb-fral-01:/home/shiny_app# docker build --tag shiny:1.0.0 -f dockerfile_shiny .
[+] Building 28.5s (7/7) FINISHED
--> [internal] load build definition from dockerfile_shiny
--> [internal] load metadata for docker.io/rocker/shiny:latest
--> [internal] load .dockerignore
--> [internal] transfer context to build environment
--> [1/3] FROM docker.io/rocker/shiny:latest
--> [2/3] RUN apt-get update && apt-get install -y libpq-dev
--> [3/3] RUN R -e 'install.packages(c("DBI", "RPostgres", "ggplot2"))'
--> exporting to image
--> writing manifest to /sha256:dd61c07037e30b4d94710f7ff7dd30ce22dc05c51c60d8046602585666d0bc04
--> naming to docker.io/shiny:1.0.0
```

Run a shiny container using “docker run -d -p 3838:3838 --network db_r_shiny --name rshiny -v /home/shiny_app:/srv/shiny-server shiny:1.0.0”

```
root@ubuntu-s-1vcpu-1gb-fral-01:/home/shiny_app# docker run -d -p 3838:3838 --network db_r_shiny --name rshiny -v /home/shiny_app:/srv/shiny-server shiny:1.0.0
1ad08269d531129382b32329330df563995e54796b4fa86baff993a73cc86f79
```

- In the above command, we mounted the host directory “/home/shiny_app” to the Shiny server container.
- “/home/shiny_app” is the directory where you located the files “.credentials.R”, “app.R”, and “psql_queries.R”. The Shiny-server uses these files to create the Shinyapp.

Copy the ipv from droplet

[← Back to Droplets](#)



Paste into URL with port : 3838 for rshiny

