In [1]:
```
import re
import os
import glob
import zipfile
import requests
from urllib.request import urlretrieve
import json
import pandas as pd
```

In [2]:
```
%cd /Users/Bruce/DSCI525
```

```
C:\Users\Bruce\DSCI525
```

In [3]:
```
article_id = 14096681  # this is the unique identifier of the article on figshare
url = f"https://api.figshare.com/v2/articles/{article_id}"
headers = {"Content-Type": "application/json"}
output_directory = "figsharedailyrain/"
```

In [4]:
```
response = requests.request("GET", url, headers=headers)
data = json.loads(response.text)  # this contains all the articles data, feel free
files = data["files"]             # this is just the data about the files, which is
files
```

Out[4]:

```
[{'id': 26579150,
  'name': 'daily_rainfall_2014.png',
  'size': 58863,
  'is_link_only': False,
  'download_url': 'https://ndownloader.figshare.com/files/26579150',
  'supplied_md5': 'fd32a2ffde300a31f8d63b1825d47e5e',
  'computed_md5': 'fd32a2ffde300a31f8d63b1825d47e5e'},
 {'id': 26579171,
  'name': 'environment.yml',
  'size': 192,
  'is_link_only': False,
  'download_url': 'https://ndownloader.figshare.com/files/26579171',
  'supplied_md5': '060b2020017eed93a1ee7dd8c65b2f34',
  'computed_md5': '060b2020017eed93a1ee7dd8c65b2f34'},
 {'id': 26586554,
  'name': 'README.md',
  'size': 5422,
  'is_link_only': False,
  'download_url': 'https://ndownloader.figshare.com/files/26586554',
  'supplied_md5': '61858c6cc0e6a6d6663a7e4c75bbd88c',
  'computed_md5': '61858c6cc0e6a6d6663a7e4c75bbd88c'},
 {'id': 26766812,
  'name': 'data.zip',
  'size': 814041183,
  'is_link_only': False,
  'download_url': 'https://ndownloader.figshare.com/files/26766812',
  'supplied_md5': 'b517383f76e77bd03755a63a8ff83ee9',
  'computed_md5': 'b517383f76e77bd03755a63a8ff83ee9'},
 {'id': 26766815,
  'name': 'get_data.py',
  'size': 4113,
  'is_link_only': False,
  'download_url': 'https://ndownloader.figshare.com/files/26766815',
  'supplied_md5': '7829028495fd9dec9680ea013474afa6',
  'computed_md5': '7829028495fd9dec9680ea013474afa6'}]
```


In [5]:
```
%%time
files_to_dl = ["data.zip"]  # feel free to add other files here
for file in files:
    if file["name"] in files_to_dl:
        os.makedirs(output_directory, exist_ok=True)
        urlretrieve(file["download_url"], output_directory + file["name"])
```

```
CPU times: total: 3.25 s
Wall time: 2min 12s
```

In [6]:
```
%%time
with zipfile.ZipFile(os.path.join(output_directory, "data.zip"), 'r') as f:
    f.extractall(output_directory)
```

```
CPU times: total: 17.5 s
Wall time: 36.4 s
```

In [8]:
```
use_cols = ["time", "lat_min", "lat_max", "lon_min","lon_max","rain (mm/day)"]
df = pd.read_csv("figsharedailyrain/EC-Earth3-Veg-LR_daily_rainfall_NSW.csv", useco
df
```

Out[8]:

	time	lat_min	lat_max	lon_min	lon_max	rain (mm/day)
0	1889-01-01 12:00:00	-35.887649	-34.766162	141.1875	142.3125	4.139137e+00
1	1889-01-02 12:00:00	-35.887649	-34.766162	141.1875	142.3125	7.438660e-03
2	1889-01-03 12:00:00	-35.887649	-34.766162	141.1875	142.3125	-5.490768e-20
3	1889-01-04 12:00:00	-35.887649	-34.766162	141.1875	142.3125	4.148483e-03
4	1889-01-05 12:00:00	-35.887649	-34.766162	141.1875	142.3125	4.291534e-04
...	...	...	...	...	...	...
3037315	2014-12-27 12:00:00	-30.280211	-29.158723	152.4375	153.5625	5.162060e+01
3037316	2014-12-28 12:00:00	-30.280211	-29.158723	152.4375	153.5625	4.504323e+01
3037317	2014-12-29 12:00:00	-30.280211	-29.158723	152.4375	153.5625	1.230073e+01
3037318	2014-12-30 12:00:00	-30.280211	-29.158723	152.4375	153.5625	1.943111e-01
3037319	2014-12-31 12:00:00	-30.280211	-29.158723	152.4375	153.5625	7.665157e-01

3037320 rows × 6 columns

In [18]:
```
%%time
## here we are using a normal python way for merging the data
import pandas as pd
use_cols = ["time", "lat_min", "lat_max", "lon_min","lon_max","rain (mm/day)"]
files = glob.glob('figsharedailyrain/*.csv')
df = pd.concat((pd.read_csv(file, index_col=0, usecols=use_cols)
                .assign(model=re.findall(r'^([^_]*)', file_name)[0])
                for file in files)
               )
df.to_csv("figsharedailyrain/combined_data.csv")
```

```
CPU times: total: 6min 9s
Wall time: 12min 4s
```

In [20]:
```
len(df)
```

Out[20]:
```
62467843
```

In [46]:
```
%%time
use_cols = ["time", "lat_min", "lat_max", "lon_min","lon_max","rain (mm/day)"]
df = pd.read_csv("figsharedailyrain/combined_data.csv",usecols=use_cols)
print(df['time'].value_counts())
```

```
1889-01-01 12:00:00    1330
1973-01-12 12:00:00    1330
1972-12-23 12:00:00    1330
1972-12-24 12:00:00    1330
1972-12-25 12:00:00    1330
                       ...
1931-01-03               28
1931-01-04               28
1931-01-05               28
1931-01-06               28
2014-12-31               28
Name: time, Length: 92010, dtype: int64
CPU times: total: 1min 21s
Wall time: 1min 27s
```

In [ ]:
```
use_cols = ['time','rain (mm/day)']
df = pd.read_csv("figsharedailyrain/combined_data.csv",usecols=use_cols)
print(df['time'].astype('float32', errors='ignore').value_counts())
```

In [24]:
```
df['rain (mm/day)'].dtype
```

Out[24]:
```
dtype('float64')
```

In [27]:
```
print(f"Memory usage with float64: {df[['lat_min', 'lat_max', 'lon_min','lon_max','
print(f"Memory usage with float32: {df[['lat_min', 'lat_max', 'lon_min','lon_max','
```

```
Memory usage with float64: 2498.71 MB
Memory usage with float32: 1249.36 MB
```

In [32]:
```
import pyarrow.dataset as ds
import pyarrow as pa
import pandas as pd
import pyarrow
from pyarrow import csv
import rpy2_arrow.pyarrow_rarrow as pyra
```

In [33]:
```
filepathcsv = "/Users/Bruce/DSCI525/figsharedailyrain/combined_data.csv"
filepathparquet = "/Users/Bruce/DSCI525/figsharedailyrain/combined_data.parquet"
filepathparquetr = "/Users/Bruce/DSCI525/figsharedailyrain/combined_data_r.parquet"
```

In [34]:
```
%%time
dataset = ds.dataset(filepathcsv, format="csv")
# Converting the `pyarrow dataset` to a `pyarrow table`
table = dataset.to_table()
# Converting a `pyarrow table` to a `rarrow table`
r_table = pyra.converter.py2rpy(table)
```

```
CPU times: total: 30.7 s
Wall time: 57.4 s
```

In [36]:
```
%load_ext rpy2.ipython
```

```
C:\Users\Bruce\miniconda3\envs\525_2023\lib\site-packages\rpy2\robjects\packages.p
y:367: UserWarning: The symbol 'quartz' is not in this R namespace/package.
  warnings.warn(
```

In [37]:
```
%%time
%%R -i r_table
start_time <- Sys.time()
suppressMessages(library(dplyr))
result <- r_table %>% count(time)
end_time <- Sys.time()
print(result %>% collect())
print(end_time - start_time)
```

```
# A tibble: 92,010 x 2
   time                    n
   <dttm>              <int>
 1 1889-01-01 04:00:00  1330
 2 1889-01-02 04:00:00  1330
 3 1889-01-03 04:00:00  1330
 4 1889-01-04 04:00:00  1330
 5 1889-01-05 04:00:00  1330
 6 1889-01-06 04:00:00  1330
 7 1889-01-07 04:00:00  1330
 8 1889-01-08 04:00:00  1330
 9 1889-01-09 04:00:00  1330
10 1889-01-10 04:00:00  1330
# ... with 92,000 more rows
# i Use `print(n = ...)` to see more rows
Time difference of 0.857393 secs
CPU times: total: 1.34 s
Wall time: 2.66 s
```

In [39]:
```
len(r_table)
```

Out[39]:
```
37
```

In [40]:
```
%%time
%%R -i r_table
start_time <- Sys.time()
suppressMessages(library(dplyr))
result <- r_table %>% select(time,model)
end_time <- Sys.time()
print(result %>% collect())
print(end_time - start_time)
```

```
# A tibble: 62,467,843 x 2
   time                model
   <dttm>              <chr>
 1 1889-01-01 04:00:00 BCC-ESM1
 2 1889-01-02 04:00:00 BCC-ESM1
 3 1889-01-03 04:00:00 BCC-ESM1
 4 1889-01-04 04:00:00 BCC-ESM1
 5 1889-01-05 04:00:00 BCC-ESM1
 6 1889-01-06 04:00:00 BCC-ESM1
 7 1889-01-07 04:00:00 BCC-ESM1
 8 1889-01-08 04:00:00 BCC-ESM1
 9 1889-01-09 04:00:00 BCC-ESM1
10 1889-01-10 04:00:00 BCC-ESM1
# ... with 62,467,833 more rows
# i Use `print(n = ...)` to see more rows
Time difference of 0.0197351 secs
CPU times: total: 3.28 s
Wall time: 5.28 s
```

In [44]:

```
%%time
%%R -i r_table
start_time <- Sys.time()
suppressMessages(library(dplyr))
result <- r_table %>% group_by(time) %>% summarize(average_time=mean(time))
end_time <- Sys.time()
print(result %>% collect())
print(end_time - start_time)
```

```
R[write to console]: Warning:
R[write to console]:  Expression as.double("rain (mm/day)") not supported in Arro
w; pulling data into R
```

```
# A tibble: 62,467,843 x 8
   time                lat_min lat_max lon_min lon_max rain (mm/~1 model as.do~2
   <dttm>                <dbl>   <dbl>   <dbl>   <dbl>       <dbl> <chr>   <dbl>
 1 1889-01-01 04:00:00   -36.2     -35    141.    142.    3.29e-13 BCC-~      NA
 2 1889-01-02 04:00:00   -36.2     -35    141.    142.    0        BCC-~      NA
 3 1889-01-03 04:00:00   -36.2     -35    141.    142.    0        BCC-~      NA
 4 1889-01-04 04:00:00   -36.2     -35    141.    142.    0        BCC-~      NA
 5 1889-01-05 04:00:00   -36.2     -35    141.    142.    1.05e- 2 BCC-~      NA
 6 1889-01-06 04:00:00   -36.2     -35    141.    142.    3.29e- 2 BCC-~      NA
 7 1889-01-07 04:00:00   -36.2     -35    141.    142.    8.91e- 2 BCC-~      NA
 8 1889-01-08 04:00:00   -36.2     -35    141.    142.    3.16e- 2 BCC-~      NA
 9 1889-01-09 04:00:00   -36.2     -35    141.    142.    3.11e- 2 BCC-~      NA
10 1889-01-10 04:00:00   -36.2     -35    141.    142.    3.30e- 2 BCC-~      NA
# ... with 62,467,833 more rows, and abbreviated variable names
#   1: `rain (mm/day)`, 2: `as.double("rain (mm/day)")`
# i Use `print(n = ...)` to see more rows
Time difference of 1.211628 secs
CPU times: total: 5.33 s
Wall time: 9.96 s
```

Reasoning:

We will choose Arrow exchange since Arrow is optimized for performance and can take advantage of multi-core processors and distributed systems to speed up data transfer. One

of the key benefits of Arrow is that it supports zero-copy data sharing between different programming languages. This means that the data can be transferred between Python and R without the need for copying or converting the data, which can be time-consuming and memory-intensive. Besides, Arrow has good support for both Python and R, with libraries available for both languages, which means that it's easy to integrate Arrow into existing Python and R workflows.

One disadvantage of using Pandas exchange is that it can be memory-intensive, especially for large datasets. When transferring data between Python and R using Pandas exchange, the entire dataset must be loaded into memory on both sides, which is not available for our dataset with 6GB. Moreover, Pandas exchange is not as efficient as Arrow exchange since it relies on serializing and deserializing data, which can be slower and less efficient than zero-copy data sharing.

The disadvantage of Parquet file in our dataset could be the low efficiency of using Parquet file for our dataset as it spends much more time than the Arrow exchange. Moreover, the Arrow exchange uses a flat memory layout, which means that the memory required to store the data is minimized, whereas Parquet file has a more complex memory layout that requires more memory to store the same data.

In [ ]: