17 プログラミングの環境管理

花澤楓 学籍番号: 2125242

2023/12/10

1　分類と順序

世の中には膨大な数のデータ・要素が存在し、個々の人間はそれらの中から必要な情報のみを取り出して処理している。その中でも、人間が同時に、認知的に考えることができる対象数は 7 ± 2 程度と言われている(magic number Z)。情報の整理について、情報の分類、順序づけの 2 つが考えられる。。

1. 「近いもの」をグループにして分類するこのタスクでは、与えられた情報の要素を、ある特定の基準に沿ってグループ分けする。例えば、買い物リストとして以下の情報を考える。

- コーヒー
- サラダ
- ソーセージ
- お米
- ケーキ
- 惣菜
- パン
- 人参
- のり

この時、例として{コーヒー、ケーキ、パン}を洋食、{サラダ、ソーセージ、人参}を食材、{お米、惣菜、のり}を和食として分類することができる。

2. 順序に並べて示すこのタスクでは、情報をその文脈において意味のある順番に並べ替える。例えば、買い物リストについて、「料理」の文脈で順番を考えると、{サラダ、ソーセージ、人参}を食材、{お米、パン}を主食、{惣菜、ケーキ、コーヒー}を食後、{のり}などをその他として順序に並べることができる。

また、データ・情報が複数あり、それぞれが部分的に関係しれいる場合、ピラミッド構造で情報を整理することが重要である。

2　フォルダー構造の整理

データ分析研究プロジェクトを行う上で、さまざまな種類のファイルが発生し、それらを効率的に管理する必要がある。データファイル、データの cleaning 用プログラム、分析実行用プログラム、結果の出力ファイルなど、構造的に整理しなければごちゃごちゃになってしまう。

ファイルの適切な分類・順序づけの方法としては以下の通りである。

1. ファイルの分類方法
    1. by topic/ function（トピックごと、機能ごとに分類）
    2. by time（時系列順）
    3. by type（ファイル形式、.pdf, .R, .csv など）
2. フォルダーの順序
    1. by importantce（重要度で並び替え）
    2. by time（時系列順）
    3. by structure（何らかの構造に沿って）

特に、フォルダーやファイルの整理には、以下のような命名をすることで任意の順序を実現することができる。

- 01_raw
- 02_clean
- 03_analyze
- 04_report

フォルダの構造を木構造で出力してくれるコマンドが mac にはある。まず、terminal にて

```
$ brew install tree
```

を実行して、Tree パッケージをインストールする。あとは、次のコマンドを、表示したいディレクトリに移動した上で実行するだけで出力結果が `tree.txt` に保存される。

```
$ tree > tree.txt
```

自分の環境での実行結果（tree.txt）：

```
.
├── 17_プログラミングの環境管理.Rmd
├── 17_プログラミングの環境管理.pdf
├── tree.txt
└── 中級ミクロデータサイエンス.Rproj

1 directory, 4 files
```

注意として、1 階層にあるフォルダーの数と、階層の深さの間にはトレードオフがあるため、状況に応じて適切（だと感じるよう）なフォルダー構造を構築する必要がある。

3　バージョン管理と Git

プログラムやレポートを書く際には、次の流れが一般的である。

- 新規文書 → 下書き → 清書 → 加筆

プロジェクトを進めていく上で、複数のバージョンを保持することは重要である。なぜなら、一旦加筆したとしても、加筆前の清書の状態に戻りたい、といった状況が往々にして発生するためである。そのため、以下の 2 点を意識することが必要である。

1. 過去のバージョンを回復できるようにすること
2. 最新のバージョンを明確に区別すること

フォルダとファイル名を工夫してバージョンを管理するアプローチをとると、以下のような問題が発生する。

1. ファイル名の命名が複雑になりかねない
2. 最新ファイルがどれなのか、明確にわからない場合がある
3. データ・ファイル数が膨大になる
4. いつ最新版として保存するのか

Git* によるアプローチ*:

Git では、ユーザーによる作業ディレクトリ（working directory）とリポジトリ（repository）によってファイルのバージョンが管理される。

ユーザーは、作業ディレクトリでファイルに関する作業を実施したのちに、リポジトリにそのファイルを送信する。これを受け、リポジトリ上でファイルが保存される。この際、Git によるファイル管理には以下のような特徴がある。

1. 変更内容に、コメントを追加し、リポジトリにコミット（commit）する
2. ファイルの変更歴はリポジトリに保存される
3. テキストファイル[*1]について変更について変更点（diff）のみ保存

ただし、ファイルをいつ保存（リポジトリにコミット）するべきかの明確な答えはない。状況に応じて変えるべきだが、基本的には変更後すぐにコミットすることが推奨されている。

[*1] .R, .csv, .md などの、テキスト情報から内容が読み取れるファイル。反対に、バイナリーファイル（.png, .jpg, .pdf, .docx, .xlsx）と呼ばれる文字列で構成されていないファイル、つまり 0 と 1 のみで情報が構築されているファイルについては Git は、その変更点について差分を取ることができないので、データ量が膨大になってしまう。この問題は、`.gitignore` にコミットしたくないファイルの形式を記述することで対処可能。

4　クラウド保存と GitHub

クラウド保存（cloud storage）とは、世界中に散らばっているデータセンターに、インターネットを介してデータを保存することをいう。例えば、`OneDrive`, `DropBox`, `GoogleDrive` など。自分なりのバックアップと、クラウド上にデータを保存することの特徴を示す。

1. 自分なりのバックアップ

- 安全性を保証しづらい
  これは、自らで物理的にデータを保存していると、何らかの理由（パソコンの故障など）でデータが破損してしまう可能性が高いことを意味する。
- 他人とのデータの共有が不便
  自らのパソコン上でのみデータを保存している場合、他人とデータを共有する際に、USB メモリなど物理的な方法で共有しなければならない場合が出てくる。
- インフラが必要
  個人の場合はハードディスク、大規模なプロジェクトであれば膨大な量のデータを保存するためのインフラが必要になり、その設置、保全にも多大な費用がかかってしまう。

2. クラウド保存

- 複数のデータセンターにデータが複製されている
  クラウドに保存されているデータは一つのデータセンターに保存されているわけではなく、世界中にあるデータセンターに複製され、保存されている。これによって、もし 1 箇所のデータセンターが破壊されたとしてもクラウド保存しているデータが失われることはない。
- 他人との共有を、許可することで可能
  クラウド上にあるデータは、他人でも許可をもらうことで簡単にアクセスすることができる。大規模研究プロジェクトなどでは、それぞれ個人が同じファイルを扱いたいケースがあるため、この機能によってそれが簡単に実現される。
- 自前のインフラが不要
  クラウド運用会社がデータセンターにてデータを管理するので、自前でインフラを管理するコストがかからない

Github:

Github とは、「リポジトリの DropBox+ マニュアル型の機能」のようなもので、クラウド上にあるリモート・リポジトリにファイルを `stage`, `commit`, `push`（アップロード）することで変更を反映させる。また、研究プロジェクトなどにおいて、他のメンバーが更新した情報を、ローカル・リポジトリに反映させたい場合、`pull` コマンド、`clone` コマンドを使うことで可能となる。

5　GitHub と RStudio による Git の活用

Rstudio のコンソールで、git のアカウントを登録する。

```
install.packages("usethis")
library(usethis)
usethis::use_git_config(user.name = "Hanazawa-Kaede",
                user.email = "hanazawa-kaede-ds@ynu.jp")
```

Git レポジトリの作成 →Rstudio と紐付け → ファイルの `commit, push`→`pull` で作業内容を更新（リモートレポジトリの内容をローカルに反映）

Rstudio で視覚的に Git を使えることは知らなかったので勉強になりました。