クラウド＆リモート時代のデータ分析環境

株式会社 ef-prime

鈴木 了太

2020年12月19日

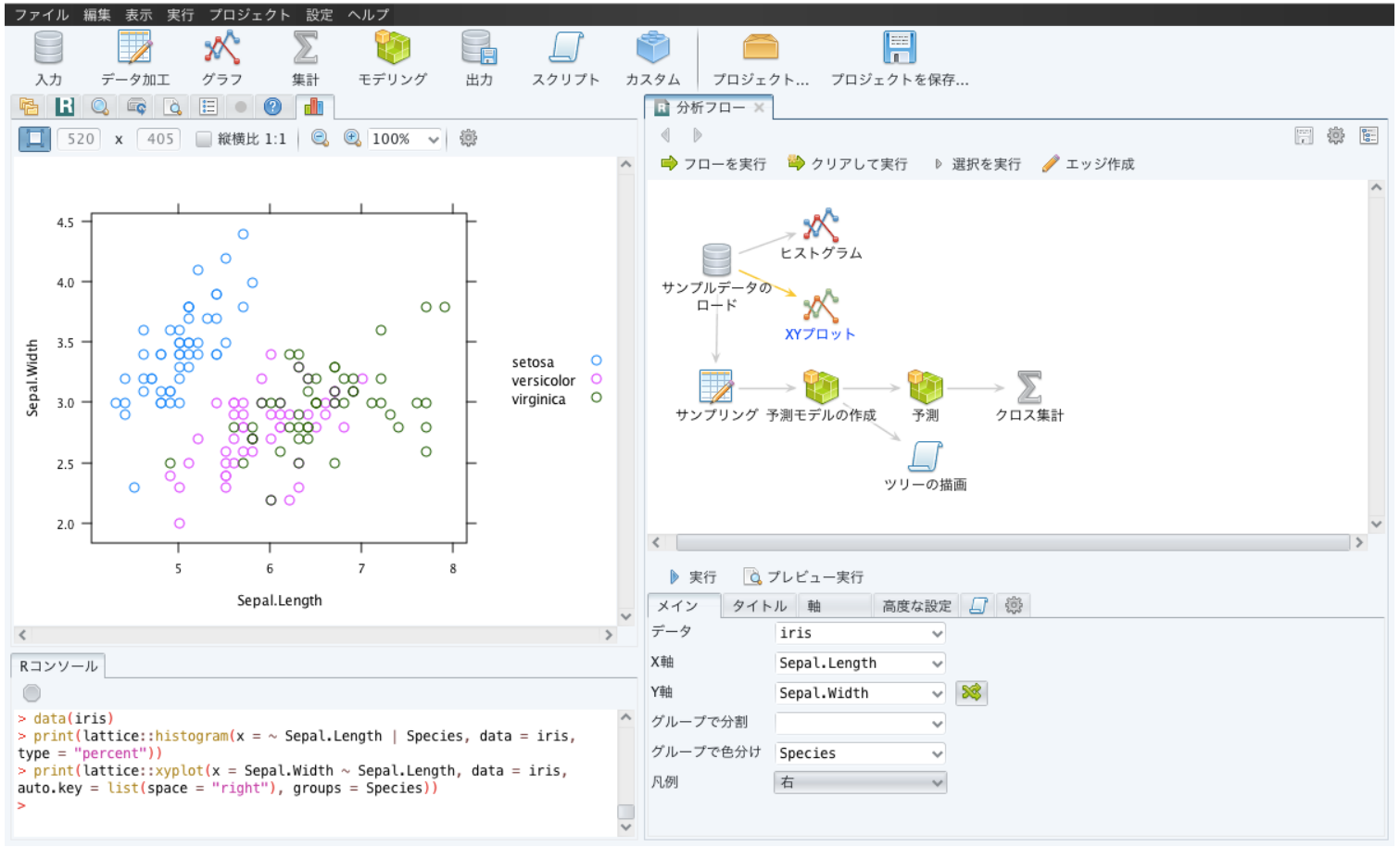
自己紹介

鈴木 了太

- 株式会社 ef-prime 代表
- おもに企業向けのデータ分析サービス
- データ分析ツール **R AnalyticFlow** の開発
- CRANパッケージpvclustのメンテナ（個人）

R AnalyticFlow (r.analyticflow.com)

ワークフロー形式のデータ分析GUI（Win/Mac/Linux対応）
Rスクリプトを直接実行することも可能



本日の話題

クラウド＆リモート時代のデータ分析環境

- COVID-19の影響でリモートでの業務遂行が重要に
- クラウドの活用が進む中、データ分析環境をどう拡張・移行するか
- 組織レベルの対応に加えて、小規模チームや個人でも可能な工夫

分析コンピューティング環境

スタンドアロン

- PC上のデータ、ソフトウェア

オンプレミス（イントラネット）

- ファイルサーバ、データベースサーバ
- オンプレミスの計算サーバ

クラウド

- クラウドストレージ、クラウドデータベース
- クラウド上の仮想マシンや分析ツール

スタンドアロンのデータ分析環境

データ

- プレーンテキスト
- Excelなどのファイル

分析環境

- PC
  - R (+ RStudio, R AnalyticFlow)
  - Excel, Access, SQLiteなど

オンプレミスのデータ分析環境

イントラネット内の計算資源でデータや分析環境を共有

データ

- ファイルサーバ上の共有ファイル
- データベースサーバ上のデータ

分析環境

- イントラネットに接続されたPC
- オンプレミスの分析用サーバ
    - PCからリモートデスクトップやSSHで接続

クラウドのデータ分析環境

データ

- クラウドストレージ上のファイル
- クラウドデータベース上のデータ

分析環境

- インターネットに接続されたPC、または計算サーバ
- 高性能な分析用**仮想マシン**
- 分析用の**クラウドサービス**

ユーザから見たオンプレミス分析環境

ファイルサーバ

- PC上のファイルと同じ感覚で利用可能
- チーム内でデータやスクリプトを共有

データベースサーバ

- 分析ツール側で対応
  - 例：RODBC, RJDBC, RPostgreSQLなどをインストール
  - dbplyrでテーブルをデータフレームのように扱うことも
- チーム内でデータが共有でき、大規模なデータも扱いやすい

ユーザから見たオンプレミス分析環境

分析用サーバ

- リモートデスクトップ、SSH、X11転送などで接続
- 高速大容量なディスク、メニーコアCPU、大容量メモリ、GPUなど共用
  - ユーザー間でリソースを奪い合う側面も
- 管理者が構築した環境でラクに分析
  - 分析ソフトウェア、ファイル共有、データベース接続など

まとめ

- 接続やツール側の設定ができればスタンドアロンと同じ感覚
- 分析環境そのものを共有することも可能だが、リソースの奪い合いも

リモートワークへの対応

- ユーザの手元にあるのはインターネットに接続されたPCのみ
- イントラネット内の環境に依存していた業務をどうするか？

対応方法

1. オンプレミスネットワークにVPN接続する
2. データや分析環境をクラウドに移行する
3. クライアントPCのリソースを活用する

1. オンプレミスネットワークにVPN接続する

メリット

- 既存のオンプレミス環境を原則そのまま利用できる
  - ユーザの業務フローをほとんど変えなくてよい

デメリット

- 環境構築の手間（および利用にかかる費用）
- 接続速度
  - 共有フォルダのデータを有線LANの感覚では利用しづらい
  - 手元のPCではなくリモート接続した現地のPCで作業するなど工夫
- 障害発生時に現地で作業する必要が生じる
- ネットワーク障害で業務が止まる

2. データや分析環境をクラウドに移行する

- データをクラウドストレージやクラウドデータベースに格納
- 仮想マシンやクラウド分析サービスを活用

メリット

- 導入・管理の手間が少ない
  - ブラウザ経由の設定だけでストレージやデータベースが利用できる
  - 機器の設定やOS・ソフトウェアのインストール、保守も不要
- リソースの柔軟な調整
  - 必要なときだけ従量課金でハイパフォーマンス仮想マシンを起動
  - リソースの奪い合いも解消しやすい
- オンプレミス環境とのハイブリッド運用も可能（サイト間VPN）

2. データや分析環境をクラウドに移行する

デメリット

- 従量課金の費用
  - 予算アラートでクラウド破産を防止
  - ただし予算超過を理由に業務を止めるわけにもいかない
- 接続速度
  - クラウドストレージをネットワークドライブとして利用する場合など、VPN経由のオンプレミス環境と類似の問題が生じる
  - 大容量データを頻繁に送受信する使い方には注意が必要
- ネットワーク障害で業務が止まる
- ユーザ側の業務フローや使用ツールに変更を要する場合も

3. クライアントPCのリソースを活用する

データやスクリプトはリモートに置き、可能な限りPC上で計算を行う

- バージョン管理システムを利用する
- PCのストレージをキャッシュとして使う
- コンテナ型の仮想環境を活用する

バージョン管理システムを利用する

- データやスクリプトをGitなどを用いたバージョン管理システムで管理
- 原則ローカルで作業し、データの送受信は作業の合間に行う
- 大きなファイルは Git LFS (Large File System) で管理

選択肢

1. GitHub, GitLab.com などのホスティング型ウェブサービス
2. クラウド仮想マシンやオンプレミスサーバにインストール
    - GitLab（GitHub風のオープンソースソフトウェア）, Subversionなど
3. ネットワーク共有フォルダをGitのベアレポジトリとして使用
    - `git init --bare`

バージョン管理システムを利用する

メリット

- 通信を最小限に抑えることができる
- 作業中にユーザ間での変更の衝突が起きない
- 作業ブランチの変更や課題管理が可能
    - バージョン管理システム・サービス自体の利点

デメリット

- 変更の衝突に対処する必要がある
- ソフトウェア開発のワークフローに慣れる必要がある
    - 現在では必須とも言えるが、バックグラウンドによってはハードルが高い
    - SourcetreeのようなGUIツールを併用すると便利

PCのストレージをキャッシュとして使う

- PC上のフォルダをオンラインストレージと自動的に同期する
- 同期済みのファイルはローカルから高速に読み込むことができる
- Microsoft OneDrive, Google Drive File Streamなど

例：Microsoft Teams (SharePoint Online)

- OneDriveアプリケーションをインストールし、共有フォルダを「同期」する
- ローカルでは標準で以下のパスが生成され、アクセスできる

  `C:\Users\user\組織名\チーム名 - フォルダ名`

PCのストレージをキャッシュとして使う

メリット

- オンプレミス環境の共有ファイルと同じ感覚で利用できる
- 基本的にはRでも問題なく利用できる
    - パッケージによってはスペースや日本語を含むパスで問題が生じる可能性も

デメリット

- 変更の衝突
    - 同じフォルダを多人数で使用すると衝突しやすい
    - （OneDriveの場合）共有ファイル内でGitを利用すると頻繁に衝突する

コンテナ型の仮想環境を活用する

Docker

- コンテナ型の仮想環境（仮想マシンのようなもの）
- Docker Hubから構築済みの「イメージ」をダウンロードして利用
- Dockerfileを記述することでカスタマイズできる

例：RStudio

- Docker Desktopをインストール（Windows/Mac/Linux）
- コマンドを実行（ `pass` は任意のパスワード、必須）

  ```
  docker run -it -p 8787:8787 -e PASSWORD=pass rocker/rstudio
  ```

  - 初回のみイメージのダウンロードに時間を要する
- ブラウザで `http://localhost:8787` を開き、ユーザー名 `rstudio` でログイン

コンテナ型の仮想環境を活用する

例：Jupyter Notebook (Jupyter Lab)

- コマンドを実行

```
docker run -it -p 8888:8888 jupyter/datascience-notebook jupyter lab
```

- コンソールに表示されたURLにアクセス

コンテナ型の仮想環境を活用する

例：ローカルPCのデータを読み書き

- Dockerコンテナは「揮発性」で変更が保存されない
- ローカルPC内のデータを参照して読み書き（Docker Desktopで権限設定が必要）
- 以下はWindows用の起動バッチ例（`run_docker.bat`）
  - `-v` オプションで `C:\Users\user\mywork` を `mywork` ディレクトリにマウント
  - `jovyan` は既定のユーザー名

```
docker run -it ^
         -p 8888:8888 ^
         -v %USERPROFILE%/mywork:/home/jovyan/mywork ^
    jupyter/datascience-notebook ^
    jupyter lab

pause
```

コンテナ型の仮想環境を活用する

例：Dockerファイルによるカスタマイズ

- `Dockerfile` という名称のテキストファイルを用意し、
  `docker build -t {イメージ名}` で自作イメージを作成（ビルドは初回のみ）
- root権限でパッケージをインストールする例
  - Jupyter Notebookイメージをベースにカスタマイズ
  - Microsoftが提供しているCRANミラーを利用して、2020/12/1時点のスナップショットからインストール

```
FROM jupyter/datascience-notebook

USER root
RUN R -e "install.packages('dplyr', repos='https://mran.microsoft.com/snapshot/2020-12-01/')"

USER jovyan
```

コンテナ型の仮想環境を活用する

メリット

- 仮想環境をコードとして管理し、配布・共有することができる
- 共通の環境を個々のPC上で実行することができる
- 「まっさらな環境」を保持できるため、破壊的な実験もしやすい
- プロジェクトごとに環境を構築することができ、設定がコードとして残る

デメリット

- GUIを利用する場合、ウェブベースのツールに偏りやすい
    - X11転送でウィンドウごと飛ばす方法も

まとめ

- スタンドアロン、オンプレミス、クラウドの違い
- VPNやクラウドの活用でリモートワークに対応
- バージョン管理システムや仮想環境なども活用
- ウェブベースのシステムだけではなく、PC上のリソースも活用