2016.10.27.THU

Rで実践！データサイエンス

～初めの一歩から高度な応用まで～

株式会社ef-prime
鈴木 了太

ef-prime, inc.

はじめに

- 本セッションで扱う内容
  - Rを利用する上での基本知識の紹介から
    ビジネス展開を意識した予測分析への応用まで

- 対象
  - Rや予測分析の基礎を学びたい、確認したい方
  - 業務での実践に近い形で予測分析を学びたい方

ef-prime, inc.

構成

- 導入編
  - 予測分析の基本的な考え方
  - ソフトウェアの入手およびインストール方法
- 基本編
  - Rの基礎と予測分析の実行方法
- 実践編
  - マーケティングデータを用いた実践的な演習

ef-prime, inc.

進行予定（改定）

予定時刻	内容	所要時間（目安）
13:00 - 13:40	導入編	40分
	（休憩）	10分
13:50 - 14:55	基本編	65分
	（休憩）	10分
15:05 - 15:45	実践編	40分
15:45 - 16:00	予備時間（質疑、ソフトウェアデモ）	15分

ソフトウェアと実行環境

- 利用するソフトウェア
  - 統計解析環境R
  - 作業環境と補助ツール
    - RStudio
    - R AnalyticFlow
    - Natto

ソフトウェアの入手・インストール方法については後ほど解説します。
インストール関連のトラブルに備え、資料はセッション後に改めて
復習いただけるよう作成しておりますのでご安心ください。

ソフトウェアと実行環境

- 推奨OS環境
  - Windows (64bit / 32bit)
  - Mac
    - Mac OS X 10.6 (Snow Leopard)以上
  - Linux
    - インストールの難易度はディストリビューションによって異なる
      - RedHat系やUbuntuではyumやapt-getで導入可能
      - 公式で配布されているバイナリはDebian、RedHat、SUSE、Ubuntu
      - （Linuxに限らず）ソースからのビルドも可能だが、環境によって特定の機能がオフになるなどトラブルも多い。まずはバイナリを推奨

資料とデータ

- 本セッションの資料とデータ
  - 分析には公開データを加工したサンプルデータを使用します。
  - 資料とデータは以下のURLで配布しています：

ef-prime.com/cdd2016/

ef-prime, inc.

導入編

予測分析クイックツアー

予測分析とはどういうもので、何ができるのか？
簡単な例をご覧いただきながら、基本的な考え方をご説明します。

サンプルデータ

- アヤメのデータ
  - （おそらく世界一有名な）iris（アイリス）データ
    - アヤメの花について、がく片（Sepal）と
      花弁（Petal）の長さ・幅を計測。
      これと種（Species）の関係を調べたデータ。
      - 種は3つあり、Setosa、Versicolor、Virginica
      - Setosaには和名があり、「ヒオウギアヤメ」
    - 数値データの出典はFisher (1936)
      - Anderson (1936) の研究で計測されたデータ

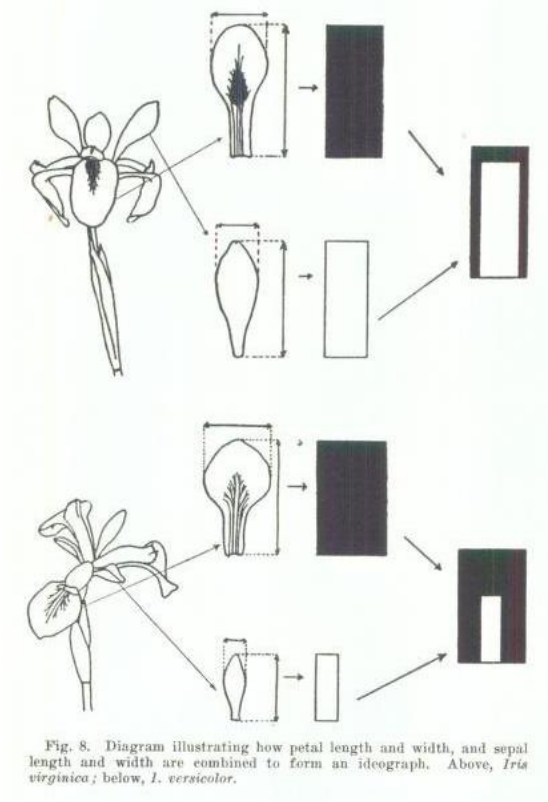


Fig. 8. Diagram illustrating how petal length and width, and sepal length and width are combined to form an ideograph. Above, *Iris virginica*; below, *I. versicolor*.

引用元: http://biostor.org/reference/11559

サンプルデータ（抜粋）

がく片の長さ	がく片の幅	花弁の長さ	花弁の幅	種
5.1	3.5	1.4	0.2	Setosa
4.9	3	1.4	0.2	Setosa
7	3.2	4.7	1.4	Versicolor
6.4	3.2	4.5	1.5	Versicolor
6.3	3.3	6	2.5	Virginica
5.8	2.7	5.1	1.9	Virginica

テーブル（表）

分析に用いるデータは、一般的にこのような表形式にまとめます。
表形式のデータを以下ではテーブルと呼ぶことにします。

がく片の長さ	がく片の幅	花弁の長さ	花弁の幅	種
5.1	3.5	1.4	0.2	Setosa
4.9	3	1.4	0.2	Setosa
7	3.2	4.7	1.4	Versicolor
6.4	3.2	4.5	1.5	Versicolor
6.3	3.3	6	2.5	Virginica
5.8	2.7	5.1	1.9	Virginica

行

テーブルの横のまとまりを行（Row）と呼びます。
分析対象の最小単位に対応し、ここではひとつひとつの花を表します。

がく片の長さ	がく片の幅	花弁の長さ	花弁の幅	種
5.1	3.5	1.4	0.2	Setosa
4.9	3	1.4	0.2	Setosa
7	3.2	4.7	1.4	Versicolor
6.4	3.2	4.5	1.5	Versicolor
6.3	3.3	6	2.5	Virginica
5.8	2.7	5.1	1.9	Virginica

行数はサンプルサイズとも呼ばれます。
この例ではサンプルサイズ6、または $n = 6$ のように記載します。
行数が多いことを「サンプルサイズが大きい」などといいます。

列

テーブルの縦のまとまりを列（Column）と呼びます。
観測値に対応し、変数または属性、項目などともいいます。

がく片の長さ	がく片の幅	花弁の長さ	花弁の幅	種
5.1	3.5	1.4	0.2	Setosa
4.9	3	1.4	0.2	Setosa
7	3.2	4.7	1.4	Versicolor
6.4	3.2	4.5	1.5	Versicolor
6.3	3.3	6	2.5	Virginica
5.8	2.7	5.1	1.9	Virginica

行数 n に対して列数は p で表し、この例では $p = 4$ となります。

目的変数

予測分析では、ある変数（列）の値を他の変数から予測します。
このとき、予測対象となる変数を目的変数と呼びます。

がく片の長さ	がく片の幅	花弁の長さ	花弁の幅	種
5.1	3.5	1.4	0.2	Setosa
4.9	3	1.4	0.2	Setosa
7	3.2	4.7	1.4	Versicolor
6.4	3.2	4.5	1.5	Versicolor
6.3	3.3	6	2.5	Virginica
5.8	2.7	5.1	1.9	Virginica

被説明変数、出力変数、応答変数、従属変数などとも呼ばれます。
記号を割り当てる際には Y を用いるのが一般的です。

説明変数

予測を行うときに参照する変数を説明変数と呼びます。

がく片の長さ	がく片の幅	花弁の長さ	花弁の幅	種
5.1	3.5	1.4	0.2	Setosa
4.9	3	1.4	0.2	Setosa
7	3.2	4.7	1.4	Versicolor
6.4	3.2	4.5	1.5	Versicolor
6.3	3.3	6	2.5	Virginica
5.8	2.7	5.1	1.9	Virginica

入力変数、あるいは独立変数と呼ばれることもあります。
記号を割り当てる際には X を用いるのが一般的です。

予測

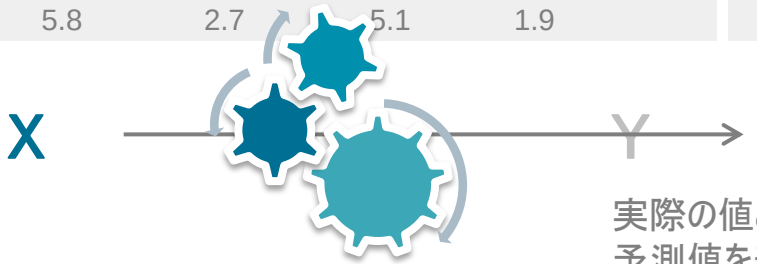
説明変数（X）の値から目的変数（Y）の値を予測します。
すなわち、Xの値だけを見て未知のYを言い当てることを考えます。

がく片の長さ	がく片の幅	花弁の長さ	花弁の幅	種
5.1	3.5	1.4	0.2	?
4.9	3	1.4	0.2	
7	3.2	4.7	1.4	
6.4	3.2	4.5	1.5	
6.3	3.3	6	2.5	
5.8	2.7	5.1	1.9	

予測モデル

説明変数の値を入力すると、目的変数の予測値を出力する数式やアルゴリズムを予測モデルと呼びます。

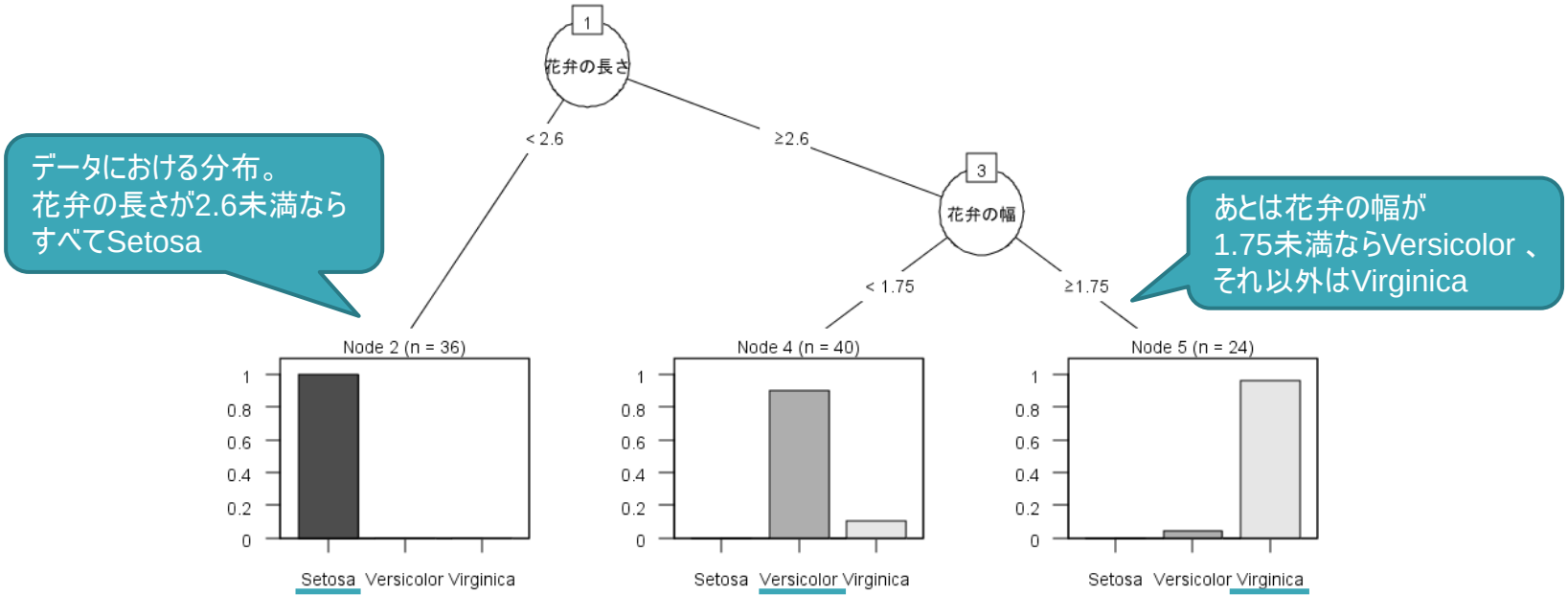
がく片の長さ	がく片の幅	花弁の長さ	花弁の幅	種	予測値
5.1	3.5	1.4	0.2		
4.9	3	1.4	0.2		Setosa
7	3.2	4.7	1.4		
6.4	3.2	4.5	1.5		
6.3	3.3	6	2.5		
5.8	2.7	5.1	1.9		



実際の値と区別するため、Y に ^ をつけて予測値を表し、ワイハットと読みます。

予測モデルの例：決定木

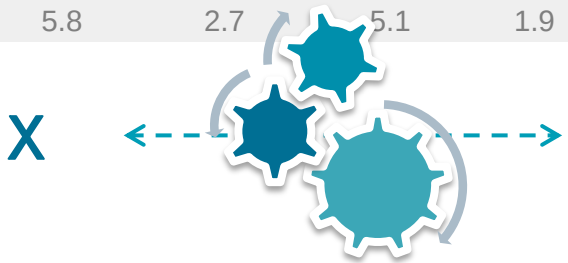
決定木モデルは説明変数を用いて自動的にルールを生成します。
ルールは木構造で表され、上から順に辿ることで予測値が得られます。



モデルの学習

決定木モデルにおけるルールの自動生成のように、
データから予測の仕組みを構築することをモデルの学習といいます。
このとき用いるデータを学習用データと呼びます。

がく片の長さ	がく片の幅	花弁の長さ	花弁の幅	種
5.1	3.5	1.4	0.2	Setosa
4.9	3	1.4	0.2	Setosa
7	3.2	4.7	1.4	Versicolor
6.4	3.2	4.5	1.5	Versicolor
6.3	3.3	6	2.5	Virginica
5.8	2.7	5.1	1.9	Virginica

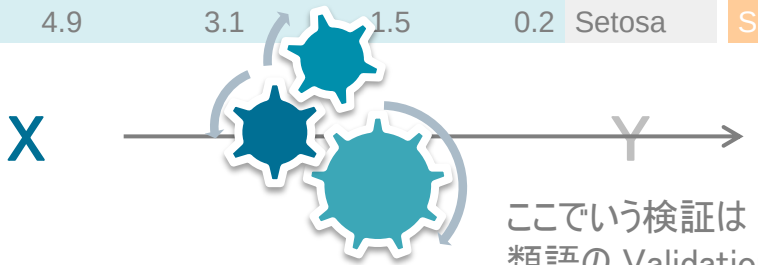


学習とは Training の訳で、訓練またはトレーニングと呼ばれることもあります。
モデルの当てはめ（Fit）ともいいます。

モデルの評価

モデルに説明変数だけを与えて予測値を出力し、
実際の値と比較することで予測精度を評価します。
このとき用いるデータを検証用データまたはテストデータと呼びます。

がく片の長さ	がく片の幅	花弁の長さ	花弁の幅	種	予測値	
4.7	3.2	1.3	0.2	Setosa	Setosa	✔
6.3	2.9	5.6	1.8	Virginica	Virginica	✔
5.5	2.3	4	1.3	Versicolor	Virginica	✘
7.1	3	5.9	2.1	Virginica	Versicolor	✘
6.9	3.1	4.9	1.5	Versicolor	Versicolor	✔
4.9	3.1	1.5	0.2	Setosa	Setosa	✔

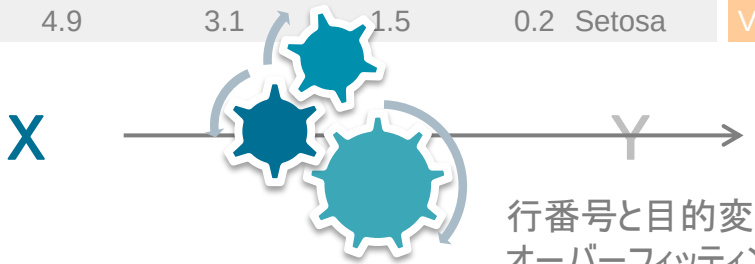


ここでいう検証は Test の訳です。
類語の Validation を検証と訳す場合もあります。

過学習（オーバーフィッティング）

モデルが学習用データに特化してしまい、他のデータに対する予測が外れる現象を過学習（オーバーフィッティング、過剰適合）といいます。
あらかじめデータを学習用と検証用に分けておくことで回避できます。

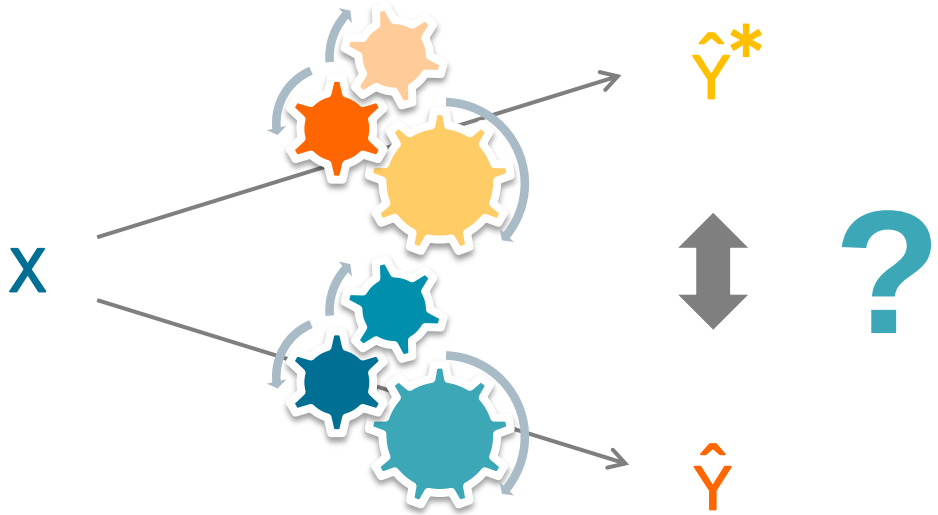
行番号	がく片の長さ	がく片の幅	花弁の長さ	花弁の幅	種	予測値	
1	4.7	3.2	1.3	0.2	Setosa	Setosa	✔
2	6.3	2.9	5.6	1.8	Virginica	Setosa	✘
3	5.5	2.3	4	1.3	Versicolor	Versicolor	✔
4	7.1	3	5.9	2.1	Virginica	Versicolor	✘
5	6.9	3.1	4.9	1.5	Versicolor	Virginica	✘
6	4.9	3.1	1.5	0.2	Setosa	Virginica	✘



行番号と目的変数の対応を学習してしまった
オーバーフィッティングの例。学習用データでは
100%正解するが、他のデータには当てはまらない。

モデル選択

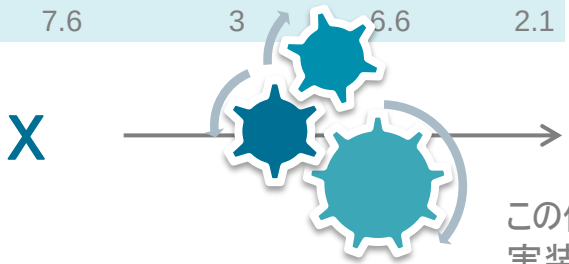
さまざまな予測手法があり、設定によっても予測値は異なります。
予測が目的の場合、基本的には精度の高いモデルを選択します。
検証用データで算出したモデルの予測精度を比較します。



予測の運用

目的変数が未知のデータについて予測を実施し、
得られた予測値に基づいて意思決定などを行います。

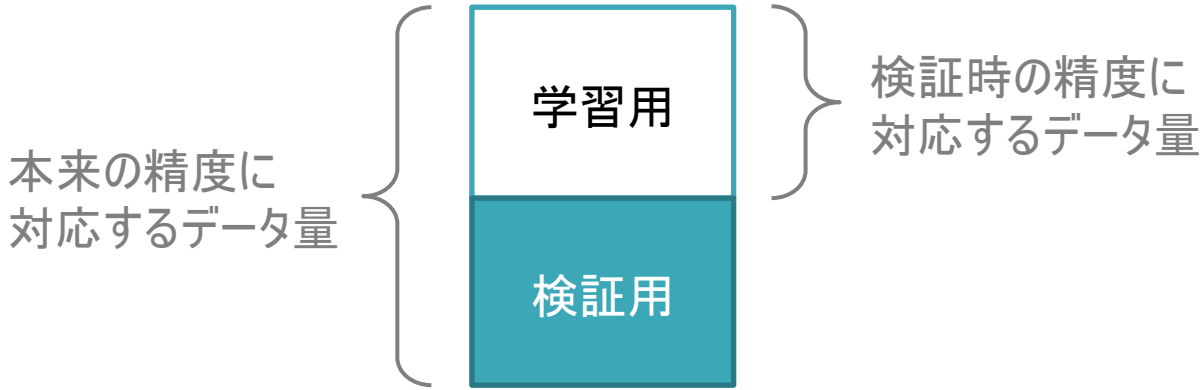
がく片の長さ	がく片の幅	花弁の長さ	花弁の幅	予測値
6.5	2.8	4.6	1.5	Versicolor
5.4	3.9	1.7	0.4	Setosa
6.5	3	5.8	2.2	Virginica
5.7	2.8	4.5	1.3	Versicolor
5	3.6	1.4	0.2	Setosa
7.6	3	6.6	2.1	Virginica



この例では、品種の自動仕分け機能を
実装するなどの使い方が考えられます。

参考：予測精度の過小評価

一般的に予測モデルは使えるデータが多いほど精度が高くなるため、
データを学習用と検証用に分割すると本来の予測精度が得られません。



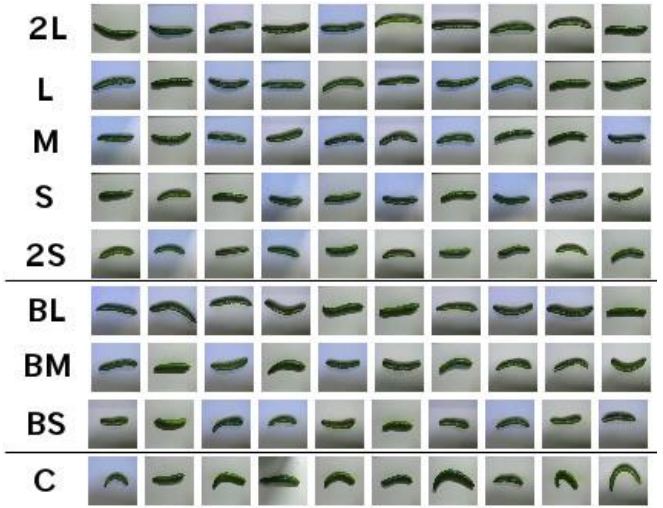
参考：クロスバリデーション

クロスバリデーション法では検証用として使うデータを少しずつずらし、
本来のデータ量に近い状態での検証を実現します。
分割数が多いほど計算時間がかかるため、5～10分割がよく使われます。

本セッションではこれ以上取り扱いませんが、
知識として覚えておいていただければと思います。

応用例の紹介

- キュウリの自動仕分け
  - 話題になった例として、画像から
    キュウリを等級別に自動仕分けする
    試みを紹介します。
    - Makoto Koike さんがウェブで公開
    - 画像データから判別するモデルを作成
    - キュウリを設置すると自動的にカメラで
      撮影、等級を判別して箱に投入



画像引用元　http://workpiles.com/2016/02/tensorflow-cnn-cucumber/

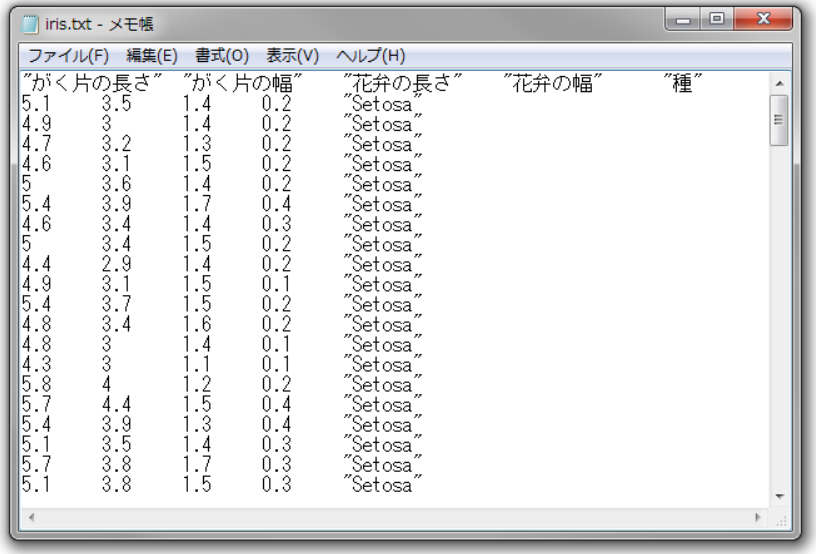
予測分析デモ

ここまで紹介してきた一連の流れを実際にご覧いただきます。

- 分析デモについて
  - 配布サンプルデータのうち iris.txt を使用します。
  - ソフトウェアはR AnalyticFlowを使用します。
    - ソフトウェアの入手やインストール、詳しい利用方法については後ほど説明します。

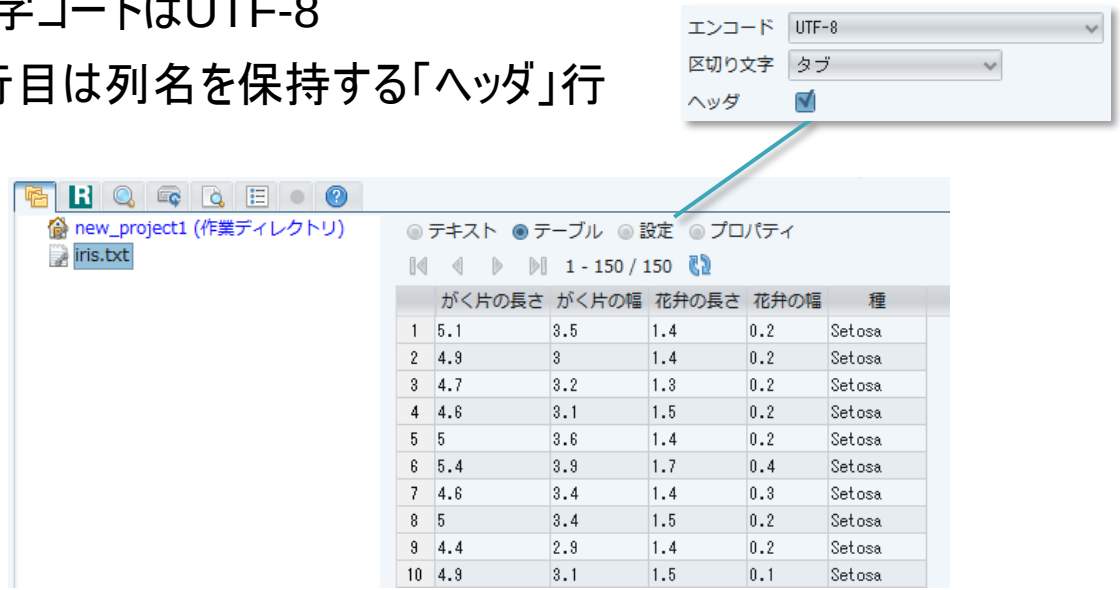
サンプルデータ

- iris.txt
  - テキストファイル



データ形式の確認

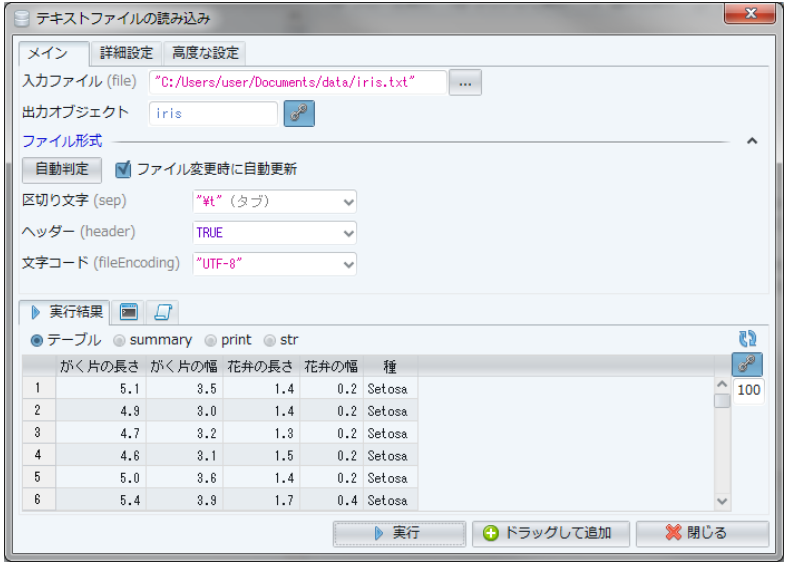
- データ形式
  - タブ区切りのテキストデータ
  - 文字コードはUTF-8
  - 1行目は列名を保持する「ヘッダ」行



	がく片の長さ	がく片の幅	花弁の長さ	花弁の幅	種
1	5.1	3.5	1.4	0.2	Setosa
2	4.9	3	1.4	0.2	Setosa
3	4.7	3.2	1.3	0.2	Setosa
4	4.6	3.1	1.5	0.2	Setosa
5	5	3.6	1.4	0.2	Setosa
6	5.4	3.9	1.7	0.4	Setosa
7	4.6	3.4	1.4	0.3	Setosa
8	5	3.4	1.5	0.2	Setosa
9	4.4	2.9	1.4	0.2	Setosa
10	4.9	3.1	1.5	0.1	Setosa

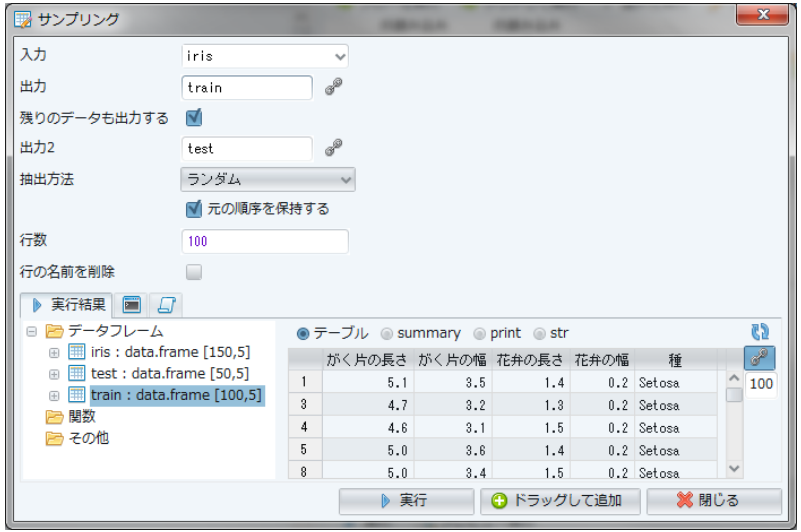
データの読み込み

- 設定を入力し、オブジェクト `iris` として読み込み
  - Rではデータや関数は全て「オブジェクト」



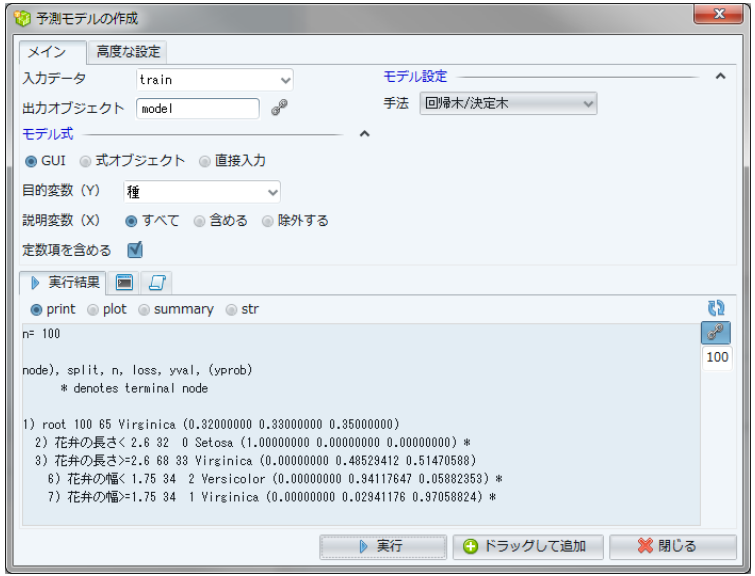
学習用と検証用に分割

- データの一部を取り出すサンプリング機能を利用
  - 乱数を発生させて無作為に抽出します（ランダムサンプリング）。
  - 学習用を `train`、検証用を `test` とします。



モデリング

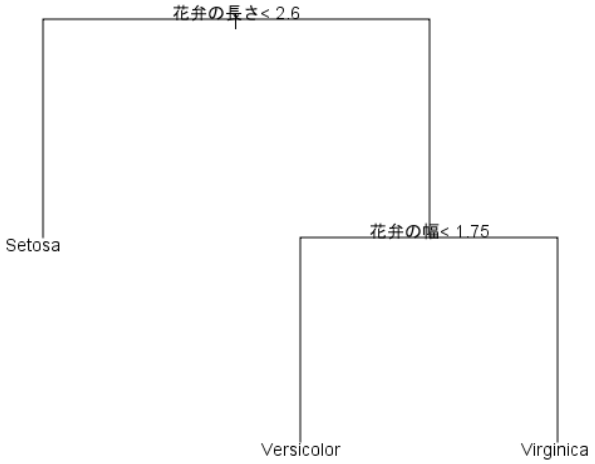
- 予測モデルの作成
  - 学習用データ(`train`)を入力とし、オブジェクト `model` に出力
  - 目的変数は種、手法を回帰木/決定木に設定



作成した予測モデル

モデルはテキスト形式で表示され、グラフィック表現も可能です。

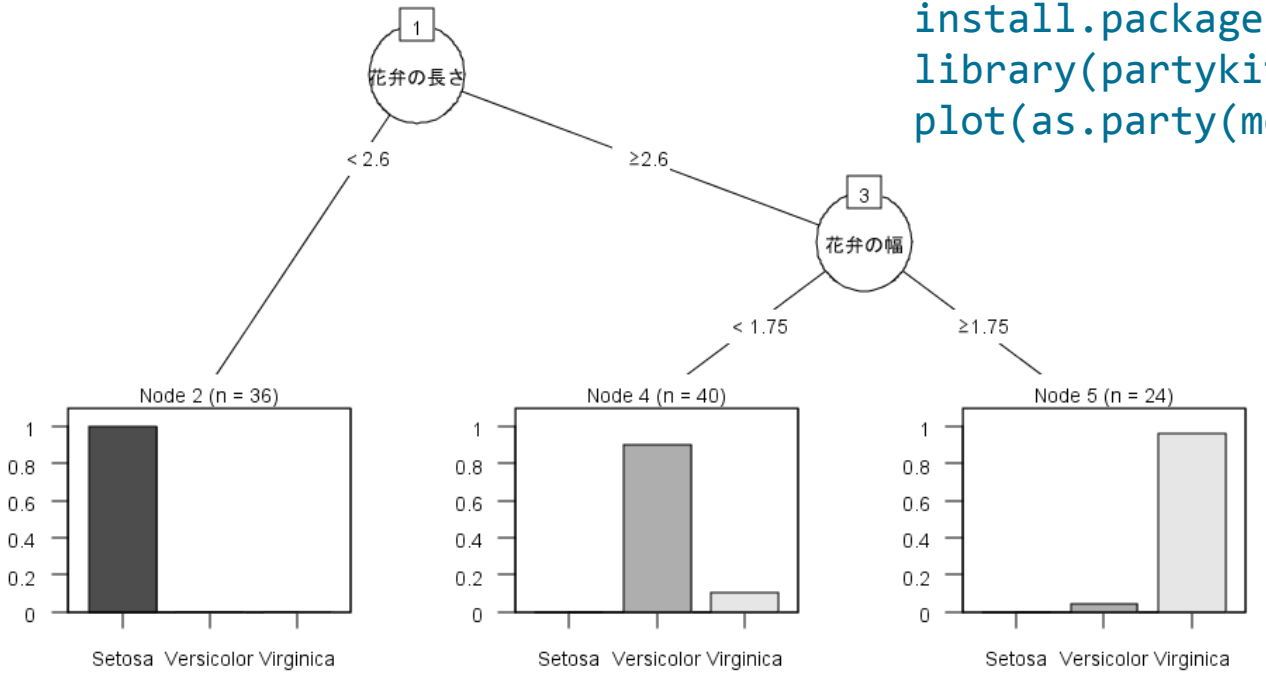
```
n= 100

node), split, n, loss, yval, (yprob)
      * denotes terminal node

1) root 100 63 Versicolor (0.36000000 0.37000000 0.27000000)
  2) 花弁の長さ< 2.6 36  0 Setosa (1.00000000 0.00000000 0.00000000) *
  3) 花弁の長さ>=2.6 64 27 Versicolor (0.00000000 0.57812500 0.42187500)
    6) 花弁の幅< 1.75 40  4 Versicolor (0.00000000 0.90000000 0.10000000) *
    7) 花弁の幅>=1.75 24  1 Virginica (0.00000000 0.04166667 0.95833333) *
```



作成した予測モデル

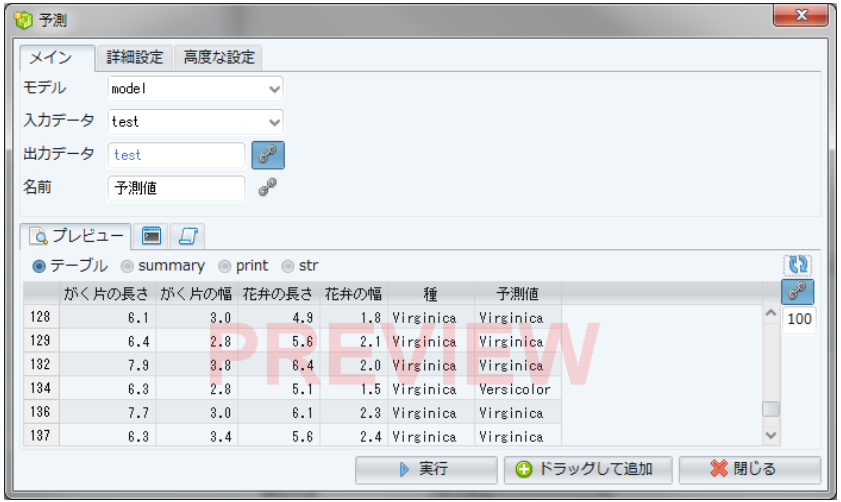
下図はRの追加パッケージ partykit を用いた作図例です。

```
install.packages("partykit")
library(partykit)
plot(as.party(model))
```



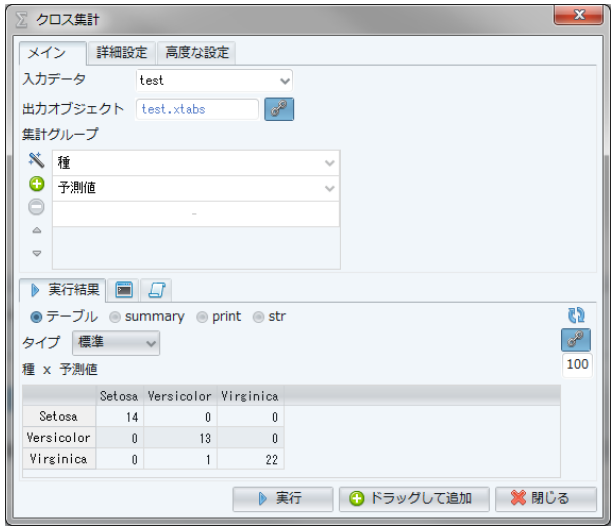
予測

- 検証用データに予測を適用
  - 作成したモデル(`model`)と検証用データ(`test`)を指定し、詳細設定で「値の予測」を選択します。
    - 多くのケースで実際の値(種)と予測値が合致している



モデルの評価

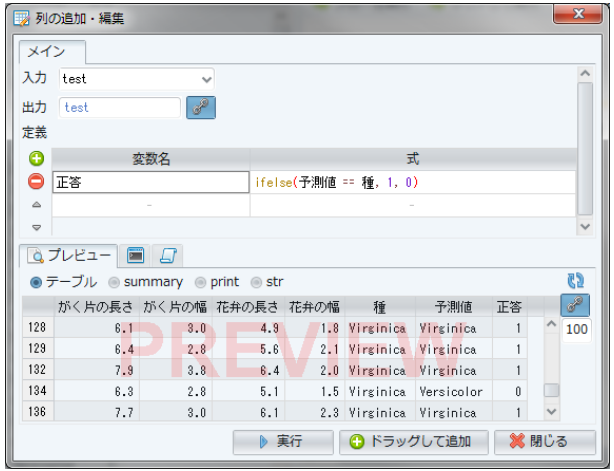
- 混同行列（Confusion Matrix）
  - クロス集計機能を使って、実績値と予測値の対応を集計します。
    - 本来はVirginicaのときに1件だけ誤ってVersicolorと予測していますが、非常によく予測できているようです。

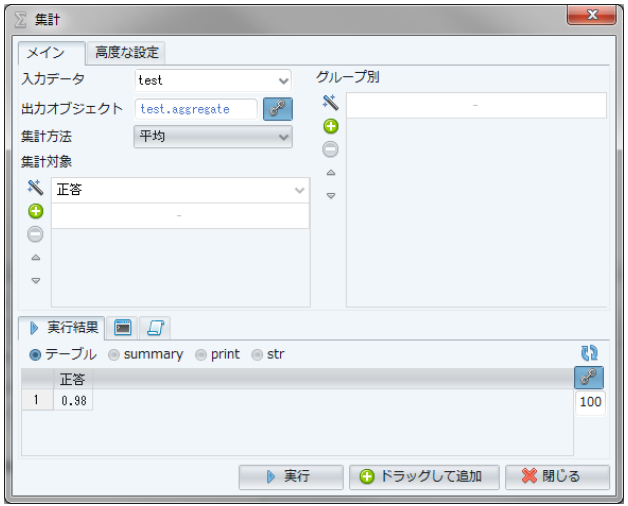


		予測値		
		Setosa	Versicolor	Virginica
種	Setosa	14	0	0
	Versicolor	0	13	0
	Virginica	0	1	22

モデルの評価

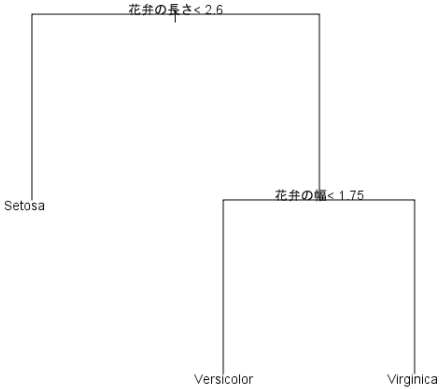
- 正答率の算出
  - 予測値と種が一致したとき1、それ以外は0となる変数を作成
    - `ifelse(予測値 == 値, 1, 0)`
    - 集計機能で平均値を計算　⇒　この例では 0.98 (98%)





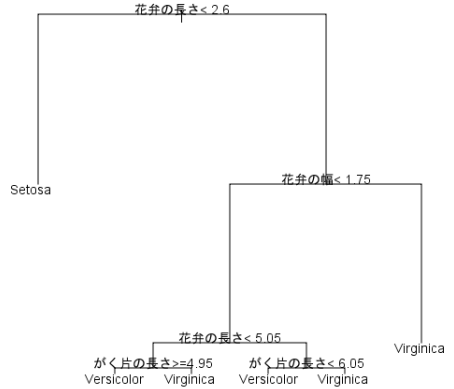
モデル選択

- モデルの比較と選択
  - 複数のモデルを作成した場合、検証用データでの予測精度を比較してより精度の高いモデルを選択します。
    - 以下の例ではよりシンプルなモデルAの方が高い精度を示しています。



モデルA	モデルB
96%	94%

※検証用データでの正答率



まとめ

- 予測とは
  - 説明変数（X）の値をみて、目的変数（Y）の値を当てること
- 予測モデル
  - 説明変数の値から目的変数の予測値を出力するもの
  - 学習用データから学習し、予測の仕組みを構築する
- モデルの評価と選択
  - 検証用データに予測を適用し、予測精度の高いモデルを選択

ソフトウェアと作業環境の準備

本セッションではRをはじめとする様々なソフトウェアを利用します。これらの位置づけについて説明し、入手方法やインストール方法についても紹介します。

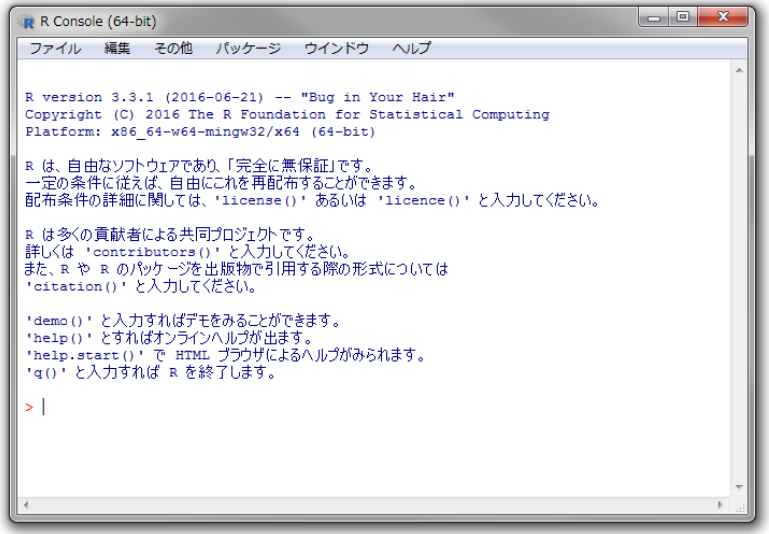
本セッションで利用するソフトウェア

- 実行環境（必須）
  - R
- 作業環境（いずれか推奨）
  - RStudio
    - デファクトスタンダードのR統合開発環境（IDE）
  - R AnalyticFlow
    - マウス操作で分析を実施、スクリプトエディタ機能もあり。日本語対応
- 補助ツール（任意）
  - Natto
    - 対話型データ分析ツール（Windows用）

セッション後に改めてインストール、
復習いただく形でも差し支えありません。

Rとは

「統計計算とグラフィックスのためのフリーなソフトウェア環境」。
対話的なコマンド入力やスクリプトの実行、外部プログラムとの連携によって
様々な統計計算やグラフィック描画を実現

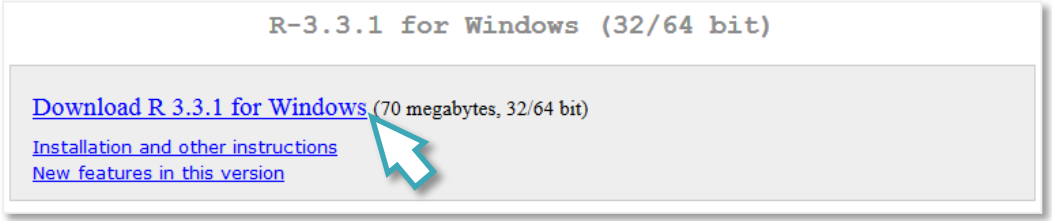


Rのインストール

- インストールの概要
  - Windows / Mac
    - 公式ミラーサイトからインストーラをダウンロードし、
      指示に従ってインストールします。
  - Linux
    - 主要なディストリビューションではyumやapt-getを使って
      比較的容易にインストールが可能です。

Rのインストール：Windows

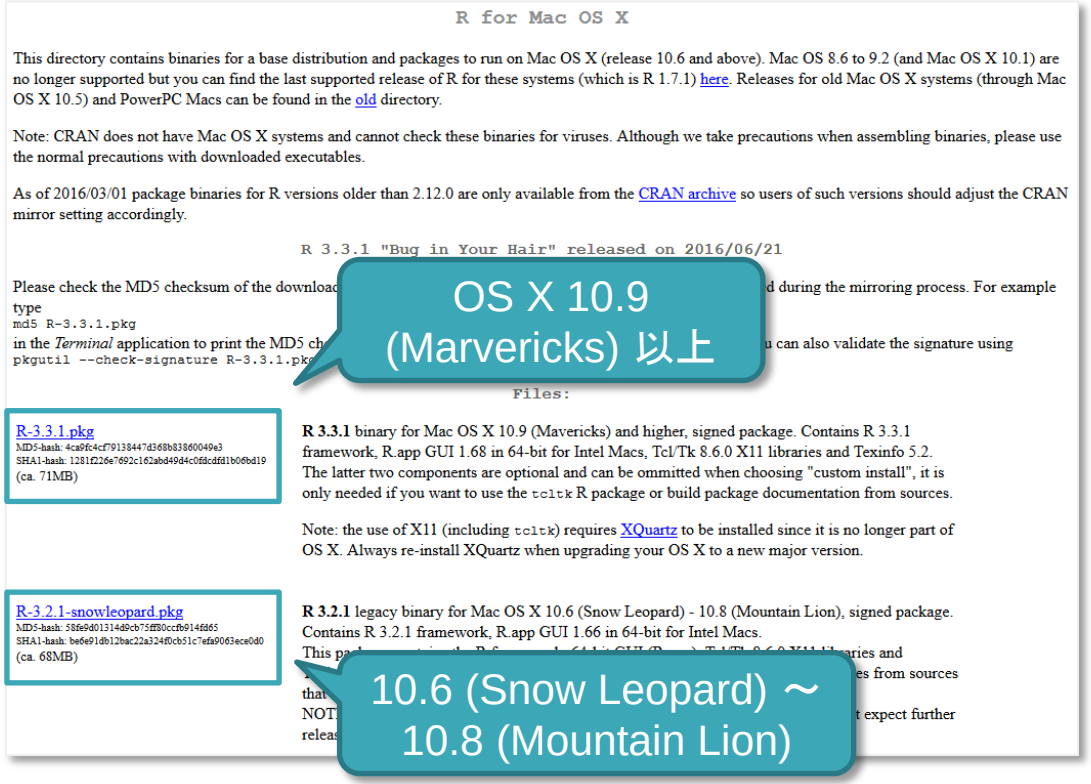
- 公式ミラーのダウンロードページ
  - https://cloud.r-project.org/bin/windows/base/
    - cloud.r-project.orgは適切なサーバへの自動リダイレクト
      - トップページからは Download R for Windows > install R for the first time とリンクを辿ると上記ダウンロードページに辿り着く
  - ダウンロードリンクをクリックしてインストーラを取得
    - インストーラを起動し、指示に従ってインストール
      - 設定は通常、既定値のままで問題ない



Rのインストール：Mac

- 公式ミラーのダウンロードページ
  - https://cloud.r-project.org/bin/macosx/
    - cloud.r-project.orgは適切なサーバへの自動リダイレクト
      - トップページからは Download R for (Mac) OS X リンクで辿り着ける
  - ダウンロードリンクをクリックしてインストーラを取得
    - OSのバージョンによってパッケージが異なるため注意（次ページ参照）
    - インストーラはWindows同様、起動して既定値のまま
      指示に従えば通常は問題ない

Rのインストール：Mac



R for Mac OS X

This directory contains binaries for a base distribution and packages to run on Mac OS X (release 10.6 and above). Mac OS 8.6 to 9.2 (and Mac OS X 10.1) are no longer supported but you can find the last supported release of R for these systems (which is R 1.7.1) here. Releases for old Mac OS X systems (through Mac OS X 10.5) and PowerPC Macs can be found in the old directory.

Note: CRAN does not have Mac OS X systems and cannot check these binaries for viruses. Although we take precautions when assembling binaries, please use the normal precautions with downloaded executables.

As of 2016/03/01 package binaries for R versions older than 2.12.0 are only available from the CRAN archive so users of such versions should adjust the CRAN mirror setting accordingly.

R 3.3.1 "Bug in Your Hair" released on 2016/06/21

Please check the MD5 checksum of the downloa d during the mirroring process. For example type
`md5 R-3.3.1.pkg`
in the *Terminal* application to print the MD5 ch u can also validate the signature using
`pkgutil --check-signature R-3.3.1.pk`

OS X 10.9 (Marvericks) 以上

Files:

R-3.3.1.pkg
MD5-hash: 4ca9fc4cf79138447d368b83860049e3
SHA1-hash: 1281f226e7692c162abd49d4c0fdcdfd1b06bd19
(ca. 71MB)

R 3.3.1 binary for Mac OS X 10.9 (Mavericks) and higher, signed package. Contains R 3.3.1 framework, R.app GUI 1.68 in 64-bit for Intel Macs, Tcl/Tk 8.6.0 X11 libraries and Texinfo 5.2. The latter two components are optional and can be ommitted when choosing "custom install", it is only needed if you want to use the `tcltk` R package or build package documentation from sources.

Note: the use of X11 (including `tcltk`) requires XQuartz to be installed since it is no longer part of OS X. Always re-install XQuartz when upgrading your OS X to a new major version.

R-3.2.1-snowleopard.pkg
MD5-hash: 58fe9d01314d9cb75ff80ccfb914fd65
SHA1-hash: be6e91db12bac22a324f0cb51c7efa9063ece0d0
(ca. 68MB)

R 3.2.1 legacy binary for Mac OS X 10.6 (Snow Leopard) - 10.8 (Mountain Lion), signed package. Contains R 3.2.1 framework, R.app GUI 1.66 in 64-bit for Intel Macs.
This pa aries and
es from sources
that
NOT t expect further
releas

10.6 (Snow Leopard) ～ 10.8 (Mountain Lion)

Rのインストール：Linux

- Fedora
  - `yum install R`

- RedHat系（Fedora以外）
  - https://fedoraproject.org/wiki/EPEL/ja
    - EPEL: Extra Packages for Enterprise Linux
      - CentOSは `yum install epel-release` と実行するだけでEPELレポジトリが利用できる。あとは `yum install R` するだけ
      - Red Hat Enterprise Linux、Oracle Linux などは上記ページに従ってレポジトリを設定し、同様に `yum install R`

Rのインストール：Linux

- Ubuntu
  - 基本的には `apt-get install r-base r-base-dev`
    - インストールに失敗、または古いバージョンが入る場合は以下を参照：
      - https://cloud.r-project.org/bin/linux/ubuntu/
        - 上記サイト中の <my.favorite.cran.mirror> は cloud.r-project.org などに置き換える
      - 本資料作成時点では3.3.1が最新。バージョン確認はコンソールから `R --version`

- その他Debian系
  - https://cloud.r-project.org/bin/linux/debian/

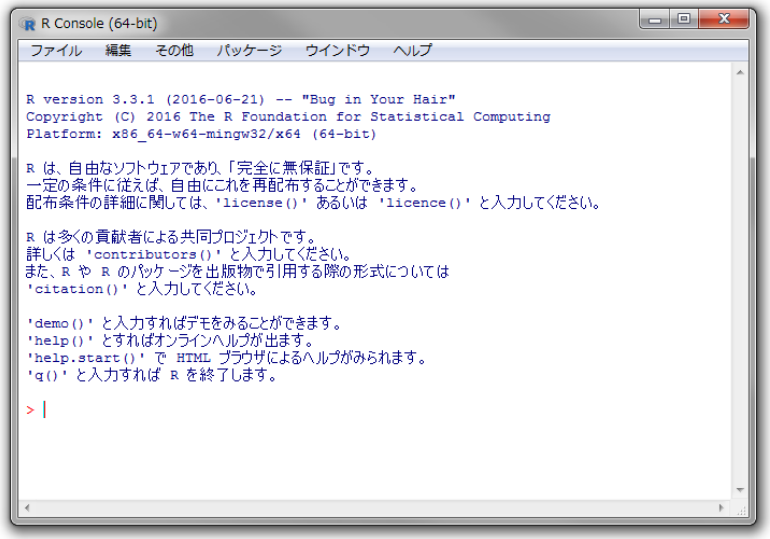
Rの起動

- Windows / Mac
  - デスクトップショートカット、アプリケーション一覧などから起動

- Linux
  - ターミナルから R とタイプ
    - ディストリビューションとインストール方法によっては Windows/Macと同様にアイコンから実行できる場合があります。

Rの起動

インターフェースはOSやインストール方法によって異なりますが、
コマンドライン形式のインターフェースが表示されます。



Rパッケージのインストール（任意）

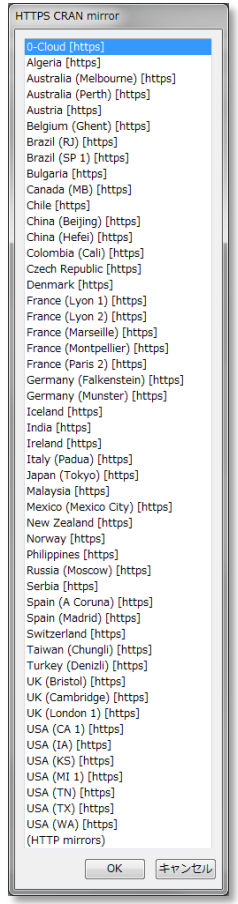
- パッケージについて
  - Rには様々な機能を実現する「関数」が用意されており、
    すべての関数はパッケージによって管理されています。

  - パッケージをインストールすることで機能の追加が可能です。
    - 標準で搭載されている基本パッケージに加えて、9,000を超える
      膨大な公式パッケージがオンラインで利用可能
    - R本体を配布しているThe Comprehensive R Archive Network、
      略してCRAN（シーラン、又はクラン）で管理・配布

Rパッケージのインストール（任意）

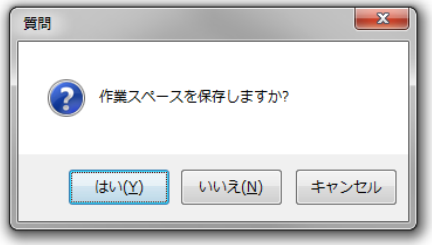
- 本セッションで利用するパッケージ
  - 標準パッケージのみで一通りの作業が可能ですが、応用例として以下の追加パッケージを利用します。

- インストール方法
  - Rを起動し、プロンプトから以下のように実行します：
    - `install.packages(c("ranger", "xgboost"))`
      - 上記コマンドを入力し、Enterキーで実行
    - 右のようなミラーサイトの選択画面が表示された場合、0-Cloud または Japan (Tokyo) などを選択してください。



Rの終了

- Rの終了方法
  - Rプロンプトから以下を実行します：
    - `q()`
      - Windows、Macではメニューから終了を選ぶなどの方法でも上記のコードが実行されます。
    - 続いて以下のようなダイアログ等が表示されるので、通常は「いいえ」を選択（またはNキーを押す）してください。
      - 「はい」の場合メモリ内容を自動で全てファイルに書き出し、次回起動時に自動で読み込むため注意が必要です。



作業環境について

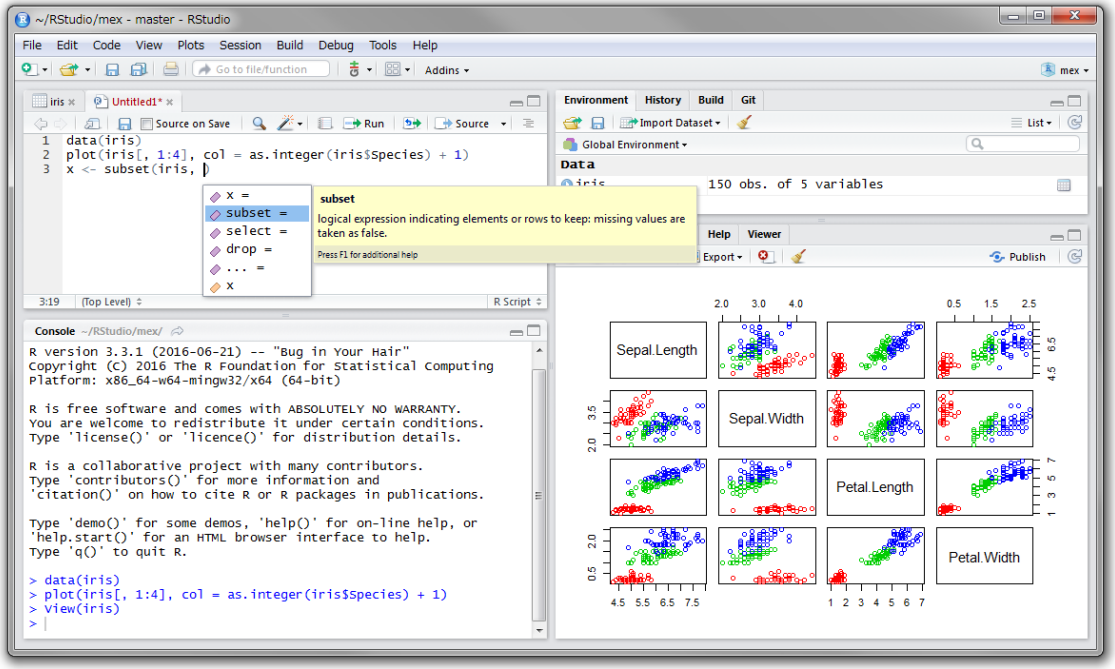
Rとテキストエディタがあれば分析を実施できますが、
専用の作業環境を導入することで効率よく作業を行うことができます。

- RStudio
  - 統合開発環境（IDE）タイプのGUIで、
    Rプログラミングやドキュメントの生成に最適です。

- R AnalyticFlow
  - データ分析に特化したGUIで、コーディング補助のほか
    マウス操作のみでも分析を実施できます。
  - 本セッションではこちらを中心的に使用しますが、
    ほとんどの内容はRStudioでも実行できます。

RStudioとは

Rと連携して動作するGUIで、統合開発環境（IDE）的な性質が強い。
他にもさまざまなエディタがあるが、現在のデファクトスタンダード

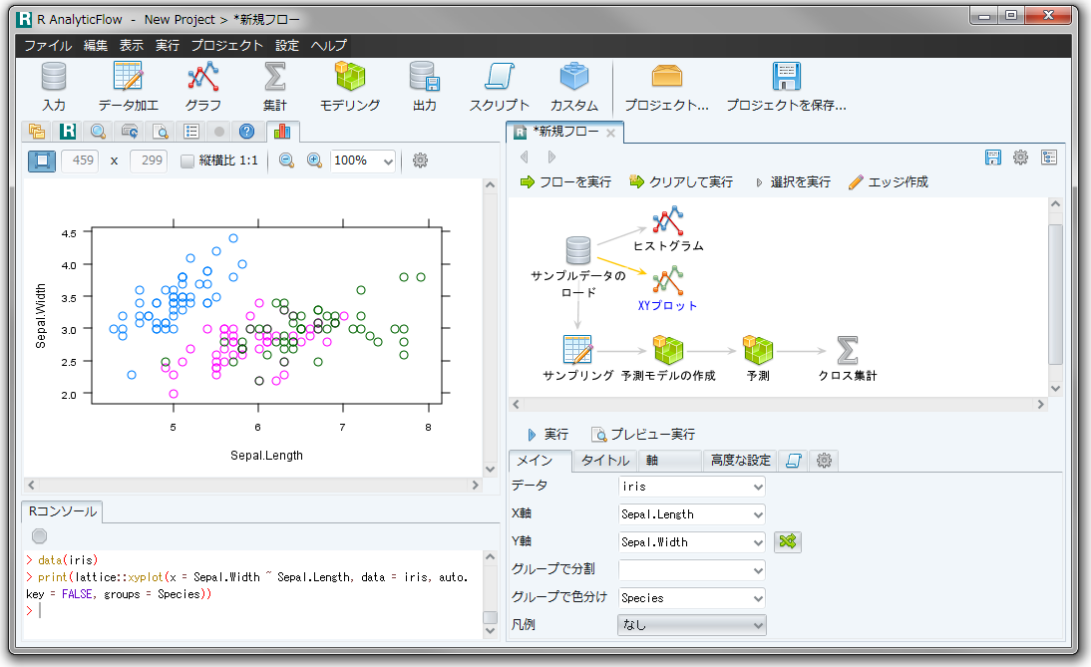


RStudioのインストール

- インストール方法
  - Server版とDesktop版があり、本セッションではDesktop版を使用
  - 下記サイトからインストーラを取得し、インストール
    - https://www.rstudio.com/products/rstudio/download/
    - 対応OS
      - Windows
      - Mac
      - Ubuntu, Debian
      - Fedora, RedHat, OpenSUSE

R AnalyticFlowとは

データ解析のためのR GUIで、ワークフロー形式で分析プロセスを記述。
マウス操作で分析を実施でき、スクリプトも記述可能。日本語対応



R AnalyticFlowのインストール

- インストールの概要（Rインストール済みの場合）
  - Windows
    - インストーラをダウンロードし、指示に従ってインストール
  - Mac
    - 必要なRパッケージをインストール
    - アプリケーションをダウンロードして実行
  - Linux
    - JDK (Java Development Kit) および
      必要なRパッケージをインストール
    - アプリケーションをダウンロードして実行

R AnalyticFlowのインストール

- インストール方法
  - プログラムの入手
    - http://r.analyticflow.com/ja/download/
    - お使いのシステムに適したバージョンをダウンロード
  - インストール方法
    - 上記ダウンロードページ、または下記ドキュメントページから「スタートガイド」ドキュメントを参照
      - http://r.analyticflow.com/ja/document/

ef-prime, inc.

R AnalyticFlowのインストール

- トラブルシューティング
  - Linuxでインターフェースが文字化けする場合
    - システムまたはJavaのフォント設定を変更することで解決します。

    - CentOS 7（GNOMEデスクトップ）の例
      - アプリケーション > ユーティリティ > Tweak Tool
      - 「インターフェース」のフォントを「VLゴシック」に変更することで解決

    - Ubuntu 16（Unityデスクトップ）の例
      - Ubuntu Software から Unity Tweak Tool をインストール
      - 「既定のフォント」を「Takao Pゴシック Regular」に変更することで解決

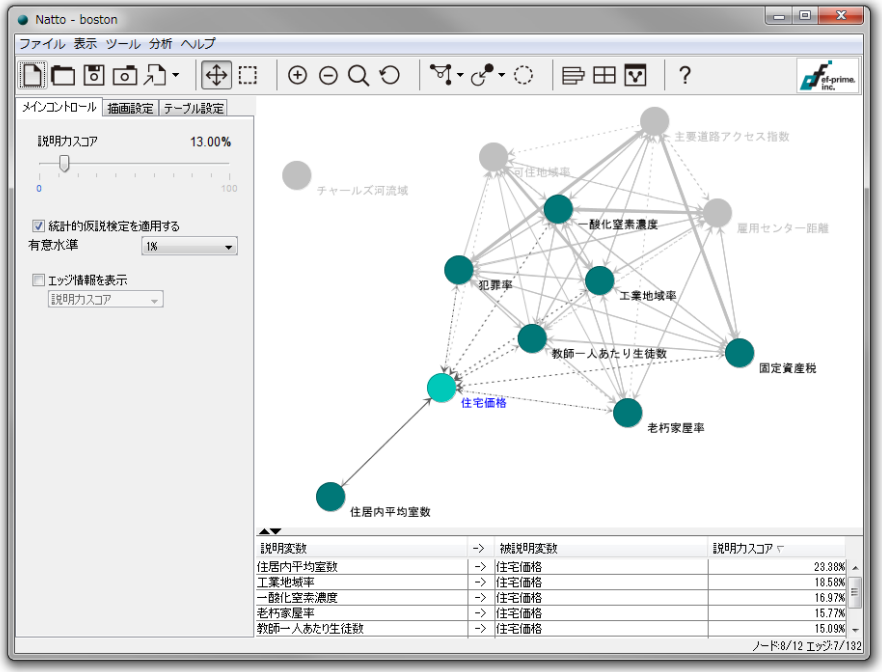
補助ツールについて

本セッションではほとんどの作業をRを用いて実施しますが、
補助的に以下のツールを使用します。

- Natto
  - Windows専用のデータ解析ツール
  - インストールは必須ではありませんが、
    分析の初期段階で役立つため一部で活用します。

Nattoとは

インタラクティブなデータ解析ツール。データに含まれる関係性を視覚化し、直感的な操作で発見することができる



Nattoのインストール

- インストール方法（Windowsのみ）
  - プログラムの入手
    - http://www.ef-prime.com/products/natto/
    - ダウンロードページからインストーラを入手
  - インストール方法
    - ダウンロードしたインストーラを実行し、
      指示に従ってインストール

休憩

サンプルデータ　http://ef-prime.com/cdd2016/

キュウリの動画　https://youtu.be/Nho2yyCdb3A

ef-prime, inc.

基本編

作業の準備

Rを使った作業は専用のインターフェースがあると便利です。
ここではR AnalyticFlowおよびRStudioのそれぞれに関して、
スクリプト作成にあたっての準備について説明します。

演習の準備

ここからはRを使って実際に手を動かしていきます。
ソフトウェアとサンプルデータを準備しましょう。

- ソフトウェア
  - Rのインストールは必須です。効率的に進めるため、
    R AnalyticFlowまたはRStudioの利用をお勧めします。

- サンプルデータ
  - 以下のURLからダウンロードできます：

ef-prime.com/cdd2016/

R AnalyticFlowの利用

- R AnalyticFlowについて
  - データ分析に特化したインターフェースで、スクリプトを含めた分析プロセスをワークフロー形式で整理することができます。
    - 弊社（ef-prime）が開発・公開しており、日本語に対応しています。
  - 以下のサイトから入手可能です：

http://r.analyticflow.com/

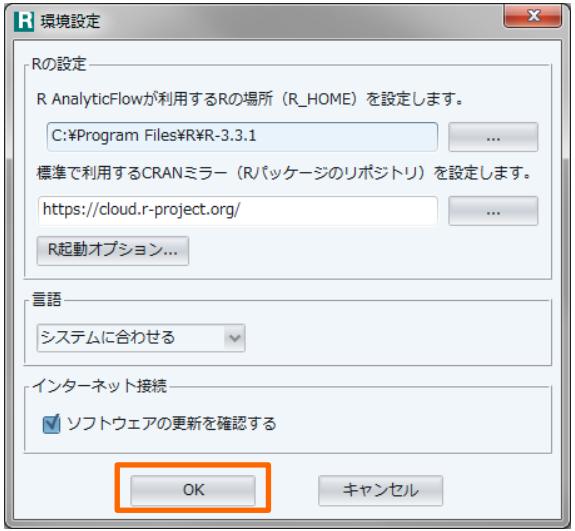
- 本セッションでの使用
  - 分析デモではGUIを用いた作業も行いますが、主にRスクリプトの記述と実行に用います。

R AnalyticFlowの利用

- ソフトウェアの起動
  - WindowsとMacではアイコンから、Linuxではスクリプトで起動します。

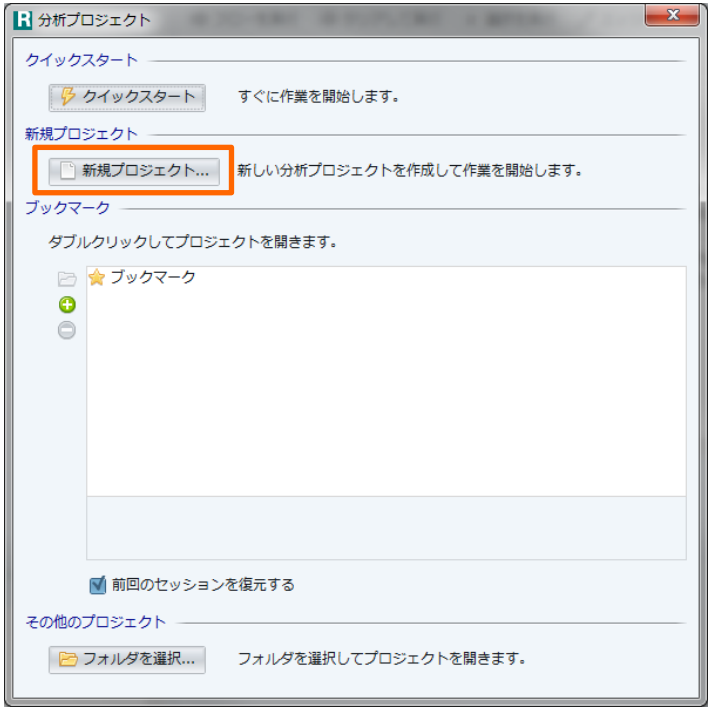
```
$ cd RAnalyticFlow_3
$ ./rflow &
```

  - 初回起動時は環境設定ダイアログが表示されます。
    - OSによって表示内容は異なります。
    - Rが正しくインストールされていれば、通常は既定値のままで構いません。



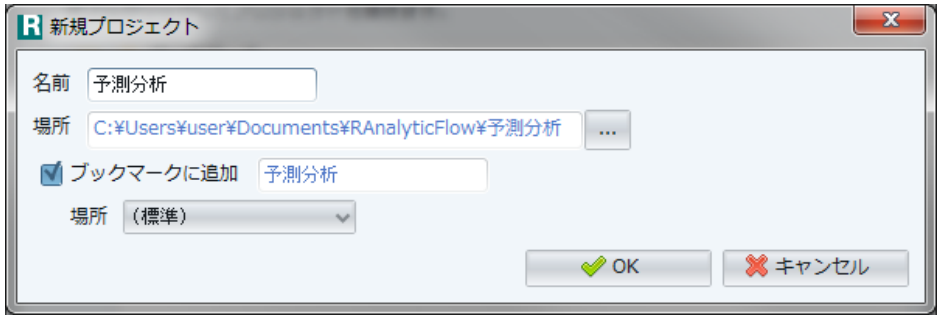
R AnalyticFlowの利用

- プロジェクトの作成
  - 続いてプロジェクトの選択ダイアログが表示されます。
    - プロジェクトを作成すると、分析プロセスやデータをまとめてひとつのフォルダで管理できます。
  - 今回は「新規プロジェクト」を選択してプロジェクトを作成しましょう。
    - 「クイックスタート」を選択して、一時的な作業フォルダを作成して分析を開始することもできます。

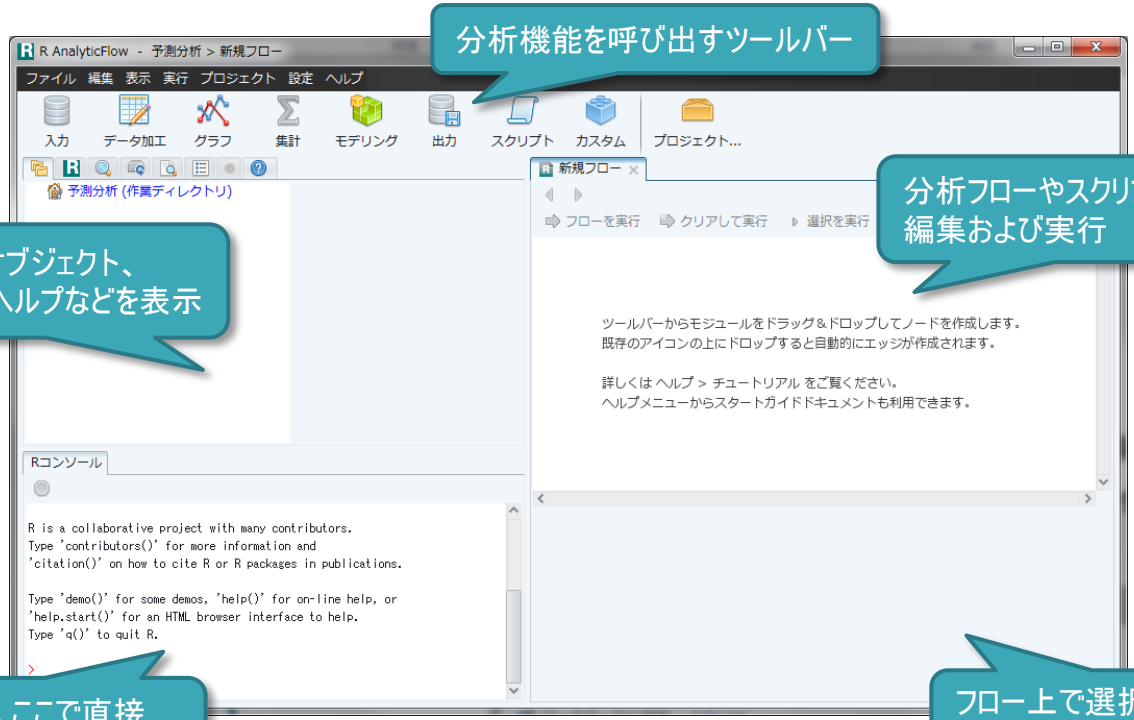


R AnalyticFlowの利用

- プロジェクトの設定
  - プロジェクトには名前を付けることができます。
  - 名前に対応して専用のフォルダが作成されます。
    - 作成するフォルダを変更したい場合、「場所」を編集してください。
    - 「ブックマークに追加」しておくと、プロジェクト選択ダイアログに表示され次回の利用がスムーズです。

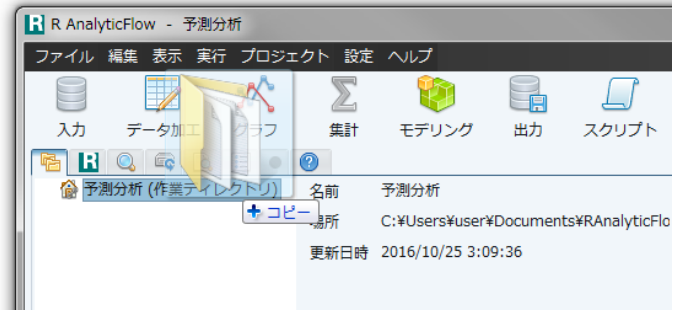


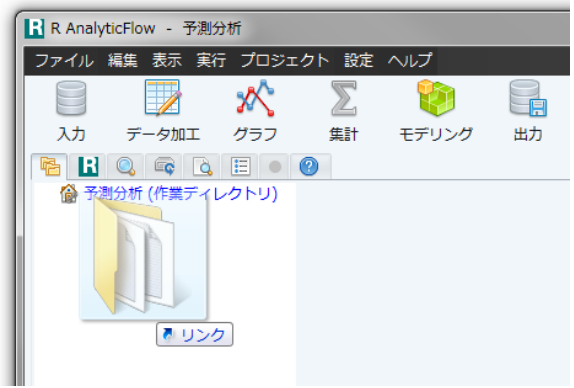
R AnalyticFlowの画面構成



サンプルデータの準備

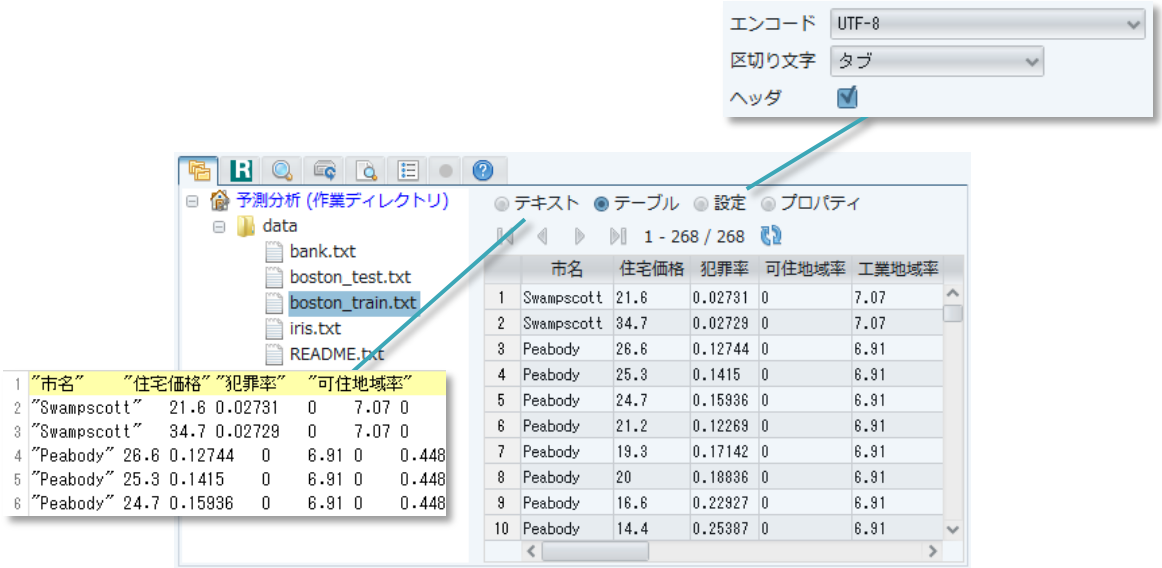
- サンプルデータのコピー
  - ダウンロードした data.zip を展開し、dataフォルダを選択して右上図の場所（フォルダ名の上）にドラッグします。
  - Ctrlキー（Macの場合はalt/optionキー）を押しながらドロップします。
    - そのままドロップすると移動されます。
  - 右下図の場所（空白エリア）にドロップするとリンクが作成されます。
    - 移動またはコピーせずに参照します。今回はコピーして利用します。





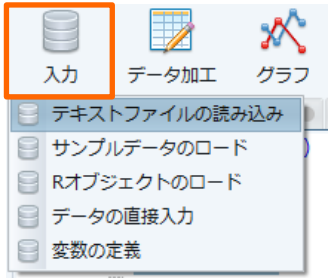
サンプルデータの参照

データファイルを選択すると、内容を参照することができます。
boston_train.txt を選択してみましょう。
ファイル形式を自動的に判別し、テーブル形式で表示します。



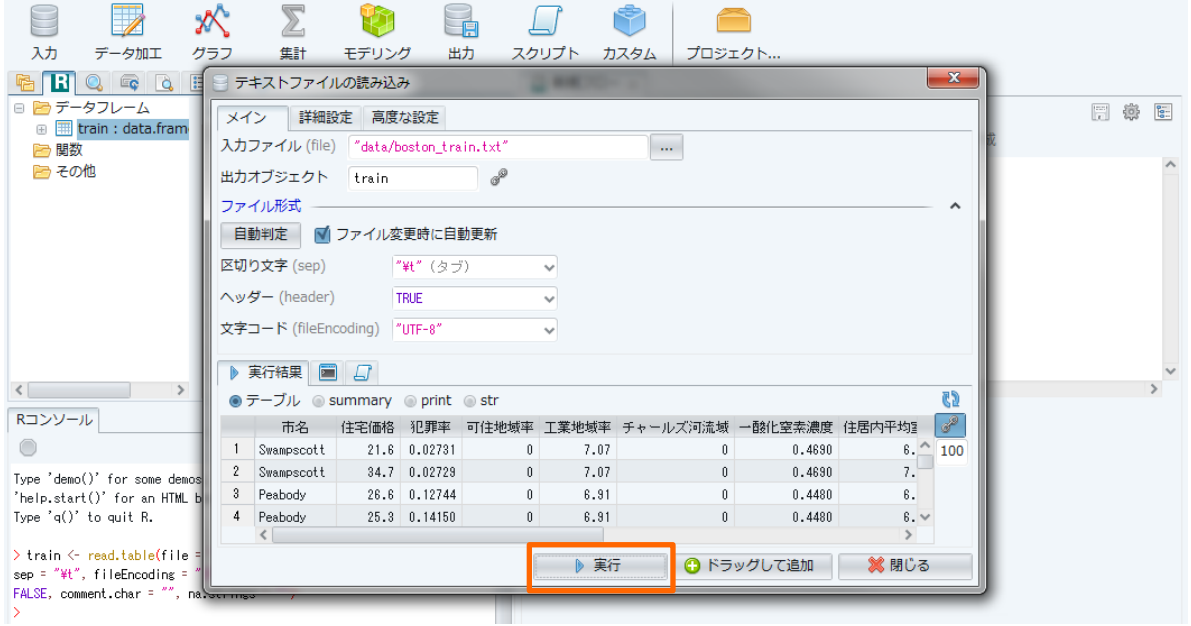
学習用データの読み込み

boston_train.txt を選択した状態で、ツールバーの入力をクリックします。
テキストファイルの読み込みをクリックしてダイアログを開きます。
文字コードなどの設定を行い、プレビューを確認してみましょう。



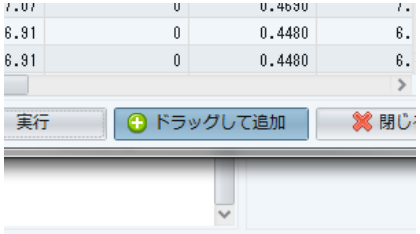
学習用データの読み込み

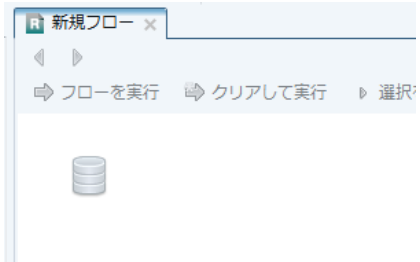
実行をクリックすると実際にデータファイルが読み込まれます。
表示されるデータもプレビューから実行結果に切り替わります。

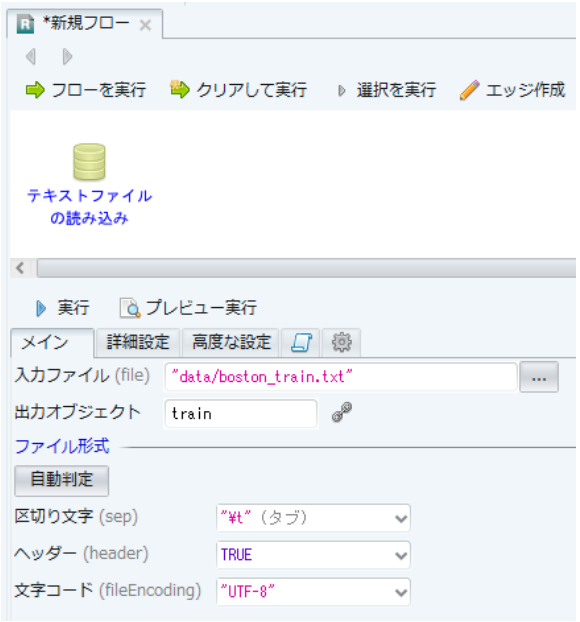


ノードの作成

ドラッグして追加ボタンを押して分析フロー上にドラッグします。
ドロップするとノードが作成されます。

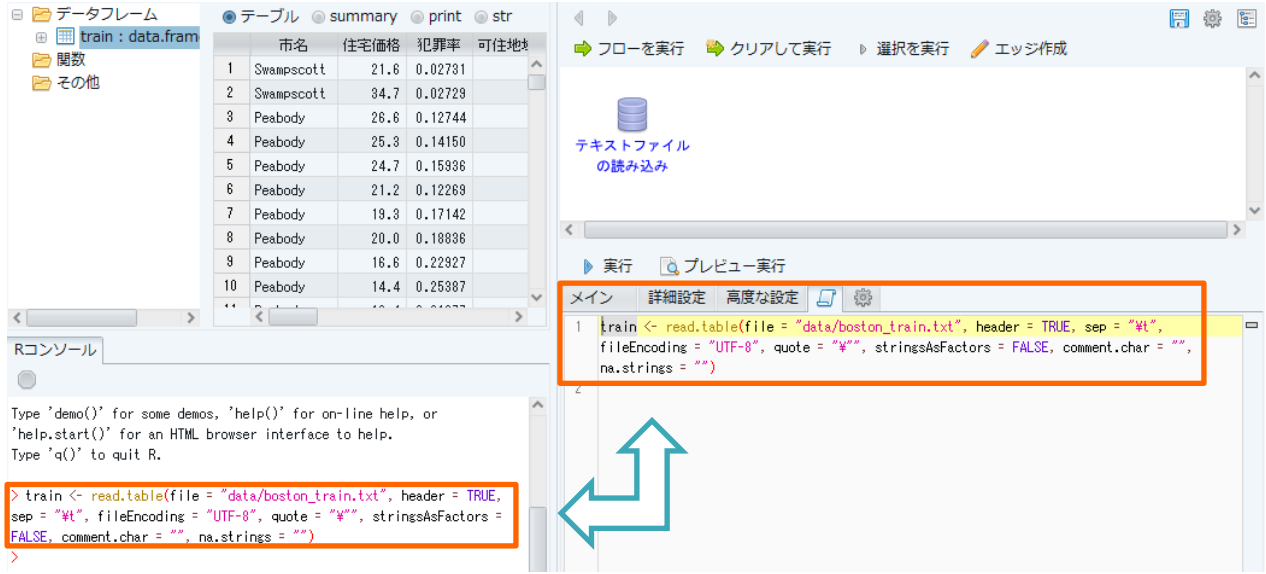






生成されたRスクリプト

ノード内のスクリプトタブをクリックするとRスクリプトが表示され、
実際に実行されたコードと一致することが確認できます。
マウス操作でスクリプトを生成・実行できるのがR AnalyticFlowの特徴です。

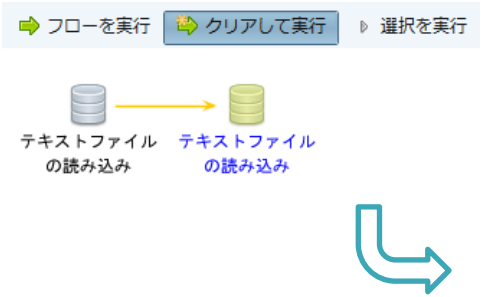


検証用データの読み込み

同様に検証用データ boston_test.txt についても読み込みます。
ドラッグして追加ボタンを押し、作成済みのノードの上にドロップします。
ノードが作成され、矢印（エッジ）で接続されます。

フローの実行

ノードを選択し、クリアして実行ボタンを押してみましょう。
作成されたオブジェクトをすべてクリアし、ノードを順に実行します。
このように、作成した分析フローは簡単に再実行できます。

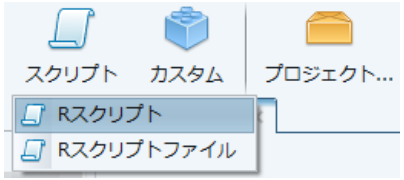


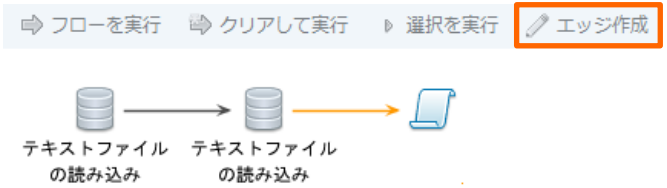
Rコンソール

```
> .clearObjects()
> train <- read.table(file = "data/boston_train.txt", header = TRUE, sep = "¥t", fileEncoding = "UTF-8",
quote = "¥"", stringsAsFactors = FALSE, comment.char = "", na.strings = "")
> test <- read.table(file = "data/boston_test.txt", header = TRUE, sep = "¥t", fileEncoding = "UTF-8",
quote = "¥"", stringsAsFactors = FALSE, comment.char = "", na.strings = "")
>
```

Rスクリプトノードの作成

分析フローにRスクリプトを記述するには、ツールバーのスクリプトから
Rスクリプトをドラッグ＆ドロップするか、フロー上をダブルクリックします。



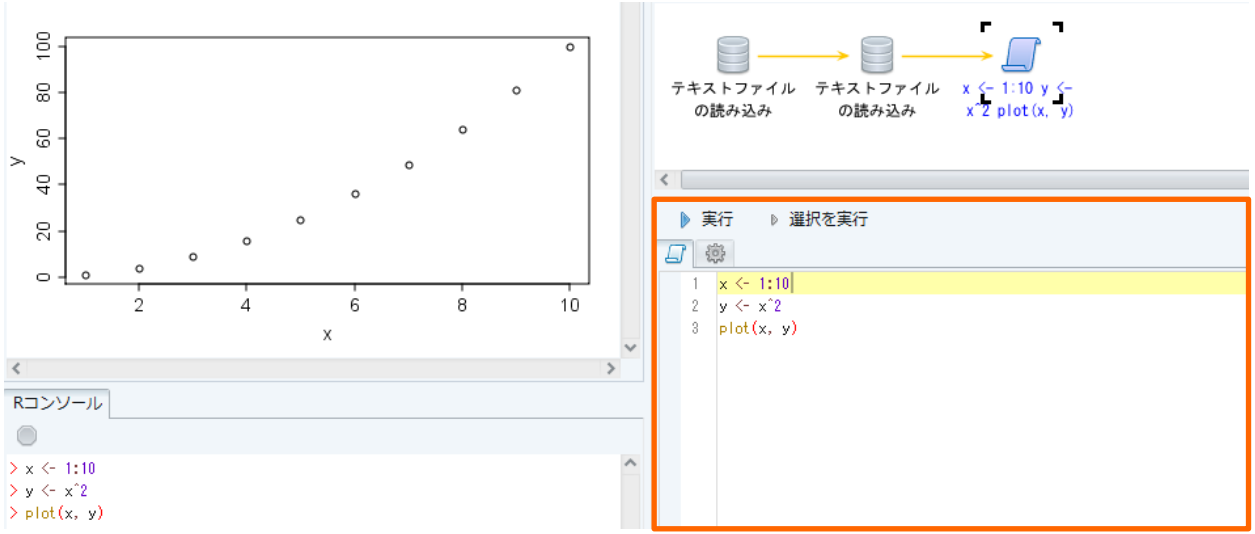


フロー上の空白部分をダブルクリックしても
スクリプトノードを作成することができます。

エッジの作成はマウスの右ボタンでドラッグします。
またはノードを選択してエッジ作成ボタンをクリックし、
宛先となるノードをクリックすることでも作成できます。

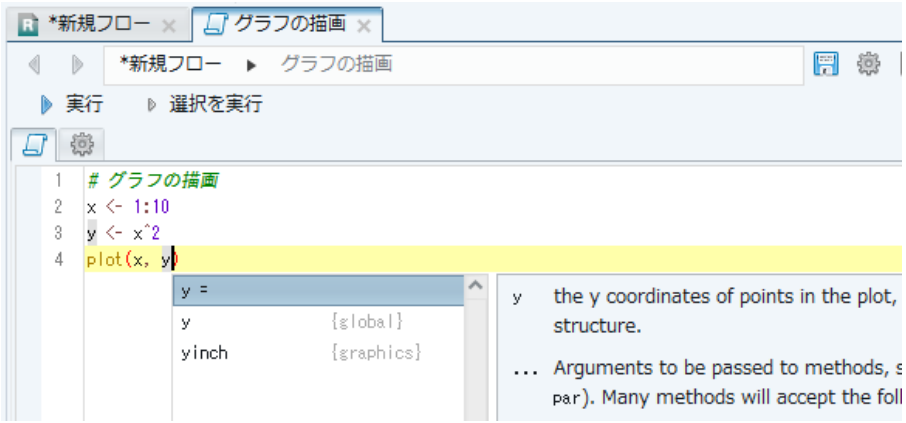
Rスクリプトの記述と実行

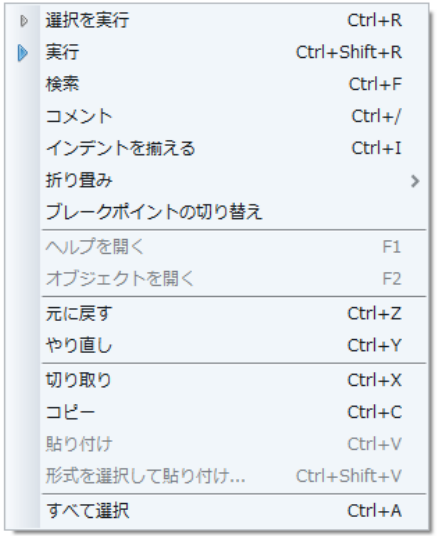
Rスクリプトノードを選択してスクリプトを記述します。
選択を実行ボタンで選択行、実行ボタンでノード全体を実行できます。
Ctrl + R（Macでは command + R）が選択を実行に対応します。



ノードをタブで編集

ノードをダブルクリックすると、新しいタブを開いて編集することができます。
Ctrl + スペース（Macでは alt/option + スペース）で呼び出せる
コード補完など、さまざまな機能が用意されています。

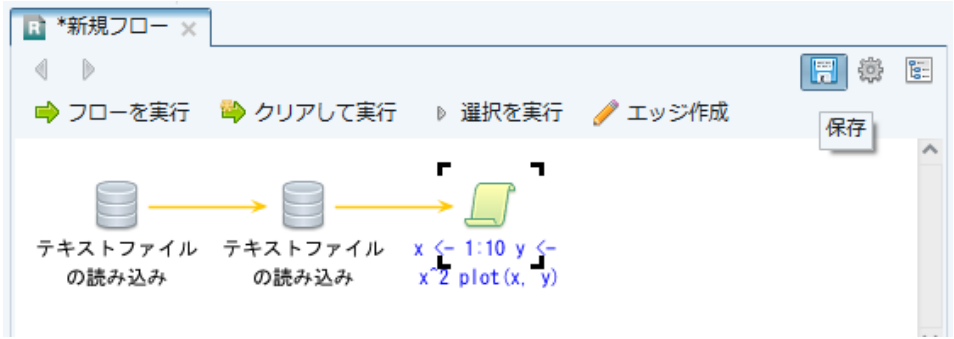




ショートカットの一覧はコンテキストメニュー（右クリックメニュー）から確認することができます。

分析フローの保存

作成した分析フローは自動で随時バックアップされますが、
念のためこまめに保存しておきましょう。
保存ボタンをクリックするか、フローのタブを選択して
Ctrl + S（Macではcommand + S）ショートカットで保存できます。



RStudioの利用

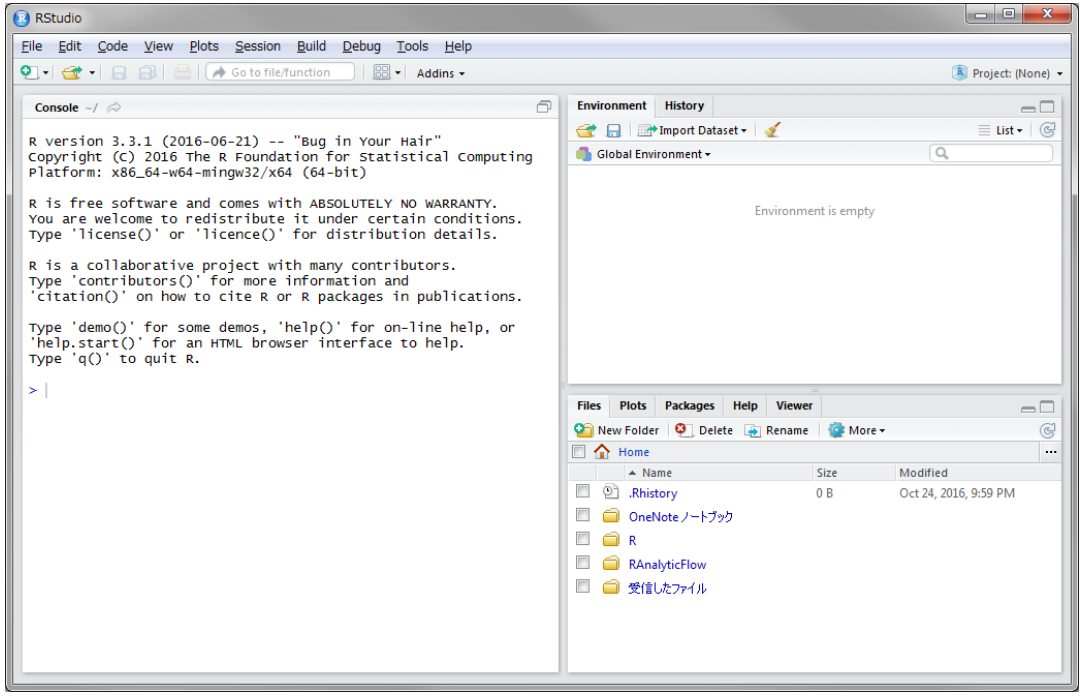
- RStudioについて
  - 統合開発環境（IDE）タイプのRインターフェースで、
    Rプログラミング環境のデファクトスタンダードといえます。
    - インターフェースは英語、データ中の日本語はおおむね利用可
  - 以下のサイトから入手可能です：

    https://www.rstudio.com/

- 本セッションでの使用
  - 以降の内容は主にRスクリプトの記述と実行がメインのため、
    R AnalyticFlowの代わりに利用できます。

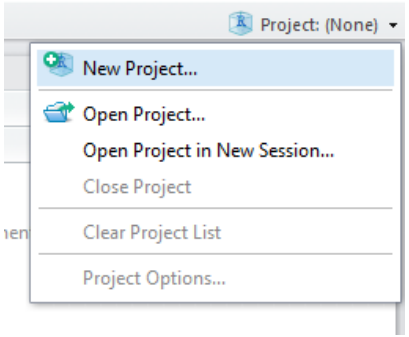
RStudioの利用

アイコンから起動すると、以下のような画面が表示されます。



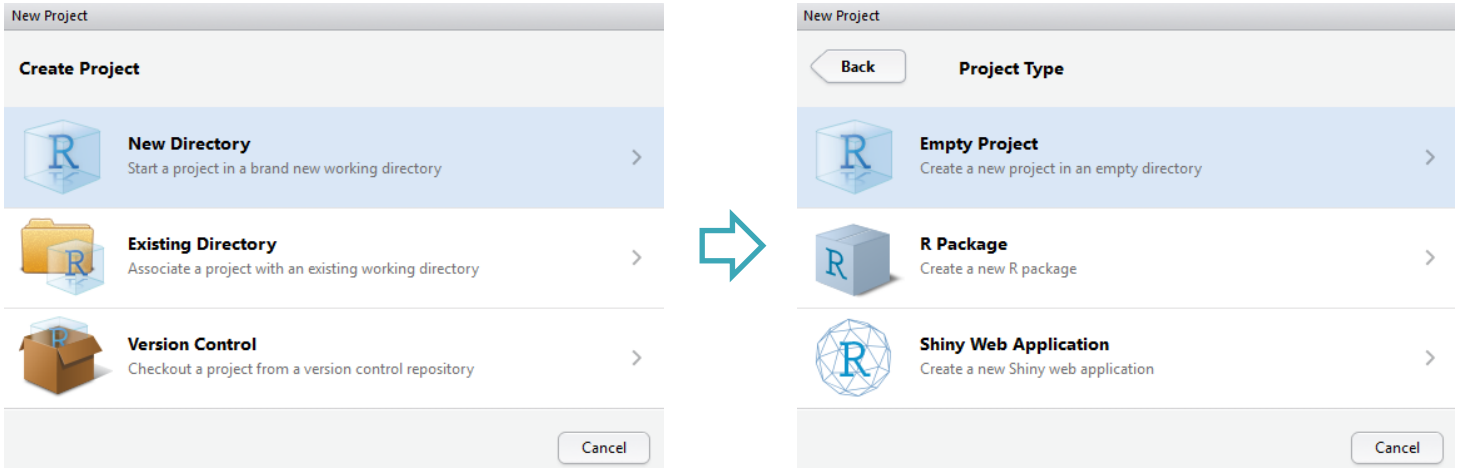
RStudioの利用

- プロジェクトの作成
  - RStudioでもプロジェクト単位でスクリプトやデータを管理します。
    - Git連携機能も搭載しており、バージョン管理も可能です。
  - 画面右上の Project: (None) と表示されている部分をクリックし、続いてNew Project... をクリックします。



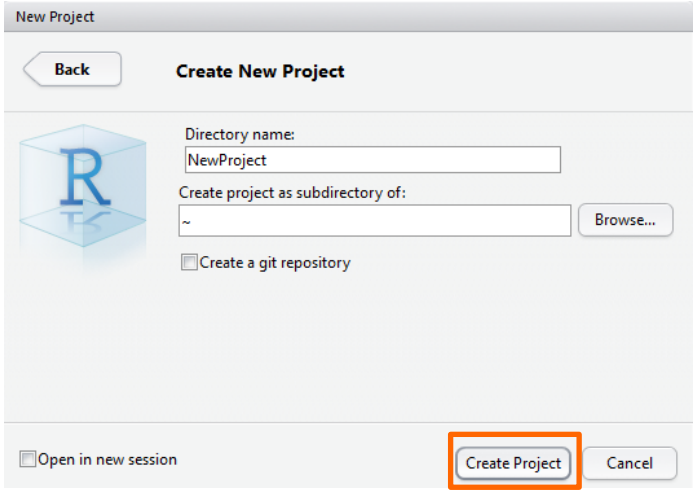
RStudioの利用

- プロジェクトの作成
  - ここでは新規フォルダに空のプロジェクトを作成することにします。
  - New Directoryを選択し、続いてEmpty Projectを選択します。



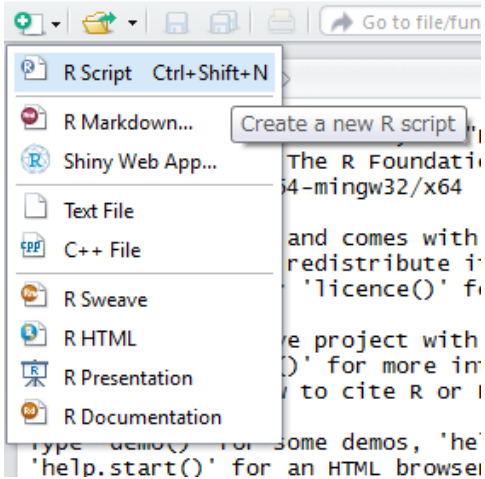
RStudioの利用

- プロジェクトの作成
  - Directory nameにフォルダ名を入力します。
  - 親となるフォルダを指定し、Create Projectボタンをクリックします。
    - 親フォルダは標準でホームディレクトリやマイドキュメントとなっています。

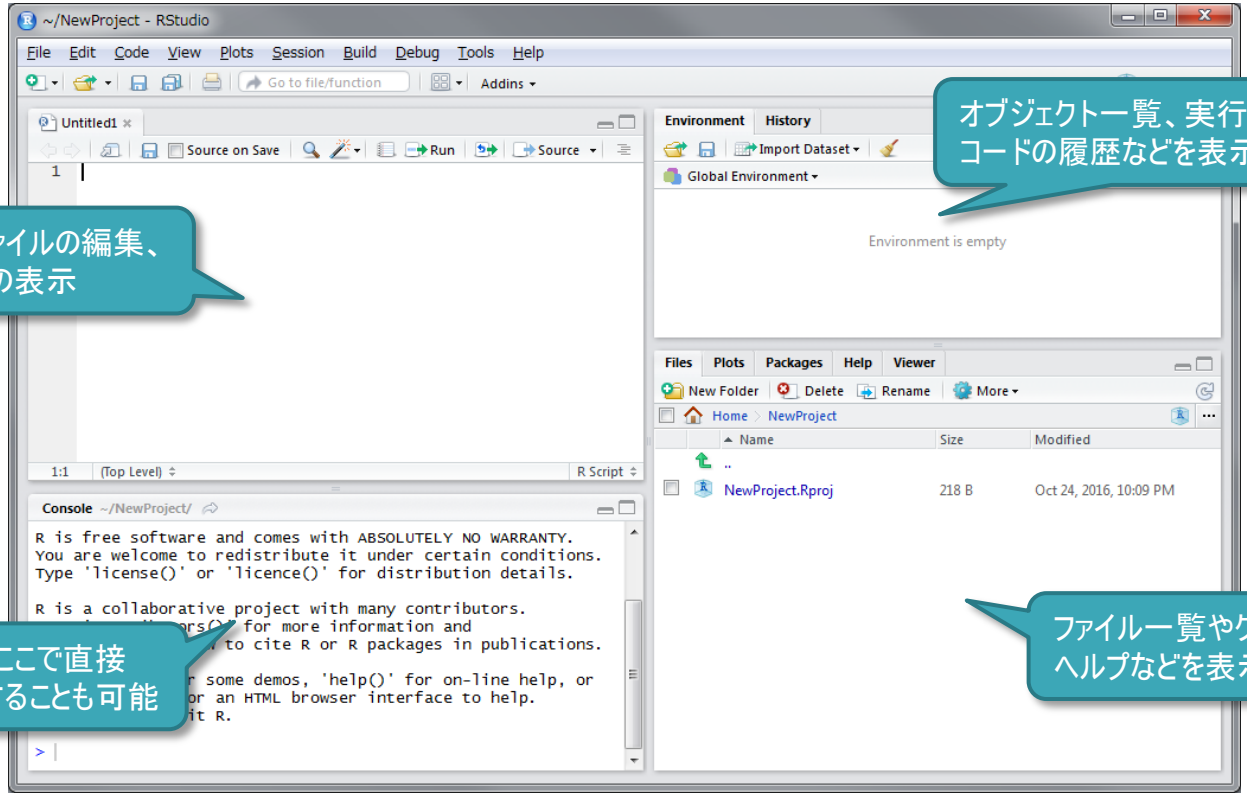


RStudioの利用

- スクリプトの作成
  - プロジェクトの作成に続いて、Rスクリプトを作成しましょう。
  - 画面左上の新規作成アイコンからR Scriptをクリックします。

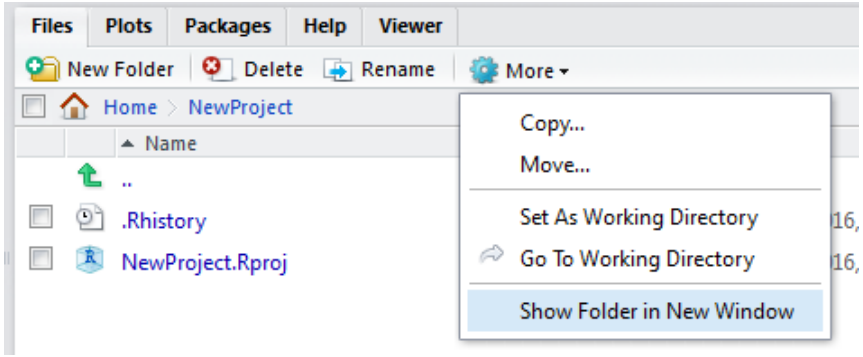


RStudioの画面構成



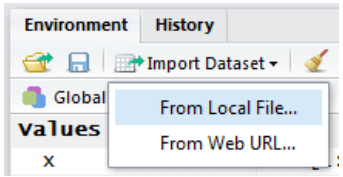
サンプルデータの準備

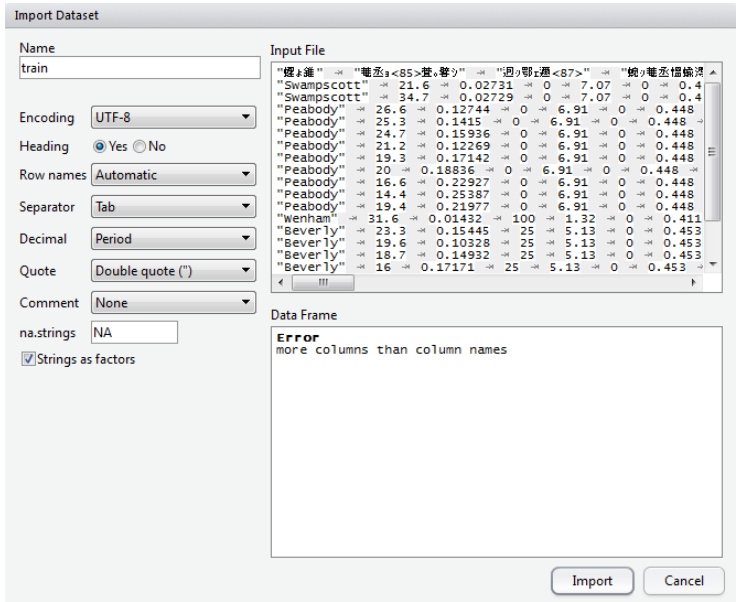
- サンプルデータのコピー
  - ダウンロードした data.zip を展開し、dataフォルダを プロジェクトのフォルダにコピーします。
    - ファイルのドラッグ＆ドロップには対応していませんが、画面右下 ファイル一覧の More > Show Folder in New Window から エクスプローラ等でフォルダを表示することができます。



サンプルデータの読み込み

画面右上Environmentタブにある Import Dataset から
データ読み込みインターフェースを利用できます。



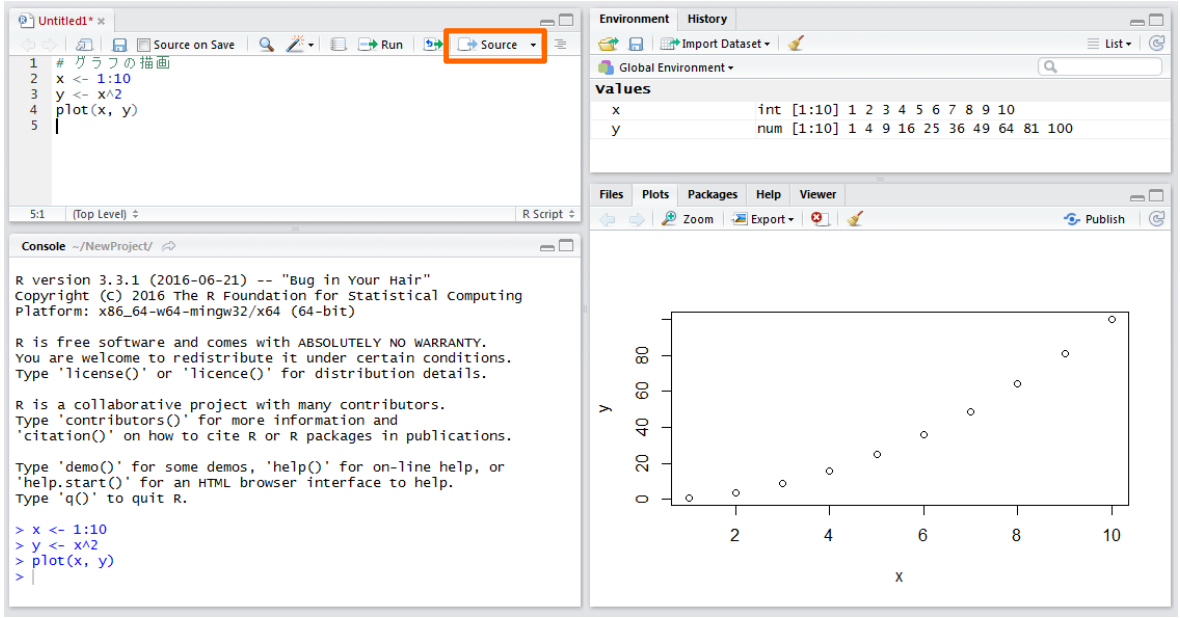


ただし、現状では（おそらくOS非標準の）
文字コード指定に対応できないようです。

後ほど説明するように、Rスクリプトを
直接記述する方法が確実です。

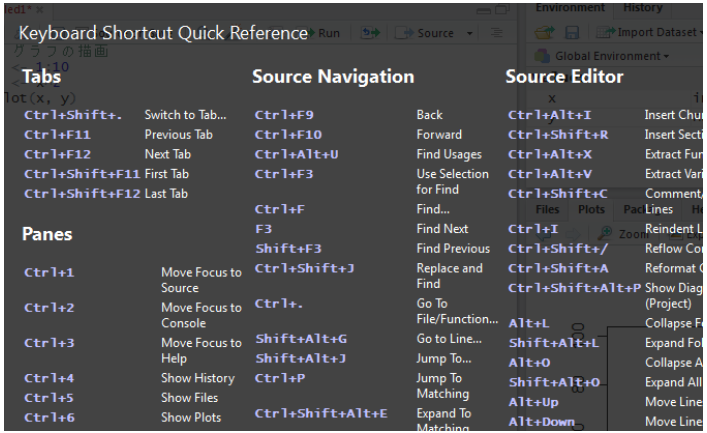
Rスクリプトの記述と実行

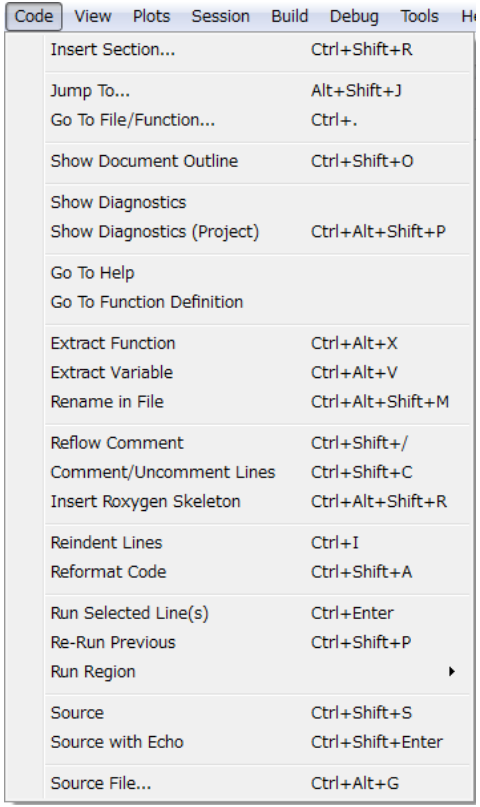
行ごとの実行はCtrl + Enter（Macではcommand + Enter）で行い、Sourceボタンでスクリプトの一括実行も可能です。



Rスクリプトの記述と実行

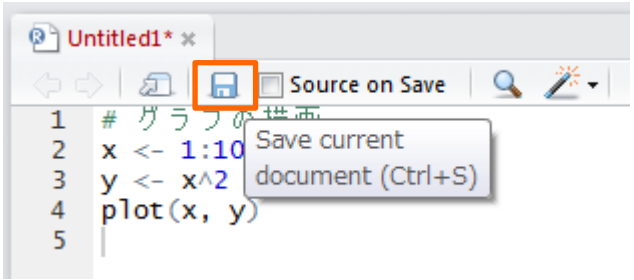
- 充実したキーボードショートカット
  - 代表的なものはメニューから確認できます。
    - Tools > Keyboard Shortcut Help で膨大な一覧が表示されます。カスタマイズも可能です。





Rスクリプトの保存

作成中のスクリプトはプログラムを終了しても保持されますが、
念のためこまめに保存しておきましょう。保存ボタンをクリックするか、
Ctrl + S（Macではcommand + S）ショートカットで保存できます。



速習R

Rによる予測分析を始める前に、まずは基本を学びましょう。
基礎となる概念は確実に押さえつつ、
予測分析に必要となる内容をコンパクトに説明します。

速習R

- はじめに
  - 以降、スクリプトを参照しながらRの基本を学びます。
  - スクリプトはRコンソールに直接入力・実行しても構いませんが、専用エディタ上で編集しながら実行するのが便利です。

- 記法について
  - スクリプト部分のフォントは `Consolas` で表示します。
  - Rプロンプトは色つきの `>` で表します。
  - コンソール出力とコメントは `薄文字` 、エラーは `強調色` で表します。

オブジェクトと値の代入

Rにおけるデータはオブジェクトと呼ばれ、宣言なしで使うことができます。
値の代入は = でも可能ですが、通常は <- を使います。

```
> x <- 10
> y <- 123.4
> y <- "文字列" # 種類の異なるデータを代入してもよい
```

以降の文字列はコメントとなります。

式の評価と結果の表示

式を評価すると結果が表示されます。オブジェクト名だけでも式となります。
値は複数の場合もあるため、1番目の要素であることを示す[1]が併記されます。

```
> 1 + 2
[1] 3

> x <- 10 # 代入は表示を伴わない
> x
[1] 10

> month.abb # 月の名前を表す組み込みデータ
 [1] "Jan" "Feb" "Mar" "Apr" "May" "Jun"
 [7] "Jul" "Aug" "Sep" "Oct" "Nov" "Dec"
```

応用的な記法

セミコロン(`;`)で区切ることで、1行に複数の式を記述することができます。
同じ値をまとめて代入することができます(代入が値を返すため)。

```
> x <- 1; y <- "文字列"
```

```
> a <- b <- 1 # a, bいずれも1を代入
```

`<-` による代入はほとんど慣習といって差し支えありませんが、
対話的な利用などでメリットが(ほんの少し)あります:

```
> 10 / 2.5 -> a # 左から右に代入できる！
```

関数

Rにおける処理は関数によって定義され、`f(x)` のように呼び出します。
関数の多くは値を返し、戻り値をオブジェクトに代入することができます。

```
> round(2.75)
[1] 3
> x <- round(2.75)
```

引数は名前を指定すれば任意の順序で指定でき、引数名を省略すると
定義上の順序で解釈されます。指定しなかった引数は既定値となります。

```
> # 以下と round(2.75, 1) は同義
> round(digits = 1, x = 2.75)
[1] 2.75
```

ヘルプの参照

help関数を使用して関数などのヘルプを参照することができます。
省略記法として ? を使うこともできます。

```
> help(round)
> ?cat
```

help.searchまたは ?? はヘルプ検索を行う関数です。

```
> help.search("example") # 要クォーテーション
> ??example
```

オブジェクトの操作

ls関数でオブジェクトの一覧を取得することができます。
消去はrm関数で行います。

```
> ls()
[1] "a" "b" "x" "y"
> rm(a, b)

> ls()
[1] "x" "y"
```

オブジェクトの操作

コピーした値を変更しても元のオブジェクトには影響ありません。

```
> x <- 3
> y <- x        # xをyにコピー
> y <- y * 10 # yの値を10倍

> x             # xの値はそのまま
[1] 3
```

実際は値を書き換えるタイミングでコピーが生成され、
不要なコピーを防いでいるようです。

オブジェクトのクラス

オブジェクトはクラスをもち、class関数で取得することができます。
基本的な型として、整数、実数、論理値、文字列などが利用できます。

```
> class(1L)      # 整数
[1] "integer"
> class(123.45) # 実数
[1] "numeric"
> class(TRUE)    # 論理値。TRUEまたはFALSE
[1] "logical"
> class("abc")   # 文字列
[1] "character"
```

クラスの変換

演算によってクラスは自動的に変換されることがあります。
変換用の関数を用いて明示的に変換することも可能です。

```
> class(4L / 1L) # 整数を割り算した結果は実数
[1] "numeric"
> as.integer(123.45)
[1] 123
> as.numeric("3.5")
[1] 3.5
> as.logical(0:3)
[1] FALSE  TRUE  TRUE  TRUE
> as.character(-3.60)
[1] "-3.6"
```

値の表示

式を評価したときの値の表示はprint関数によって行われます。
print関数はジェネリック関数のひとつで、クラスに応じた処理を呼び出します。

```
> print(x)
[1] 3
> x # print(x) と同じ
[1] 3

> help # help関数の定義を表示
function (topic, package = NULL,...(省略)

> class(help) # 関数もfunctionクラスのオブジェクト
[1] "function"
```

ベクトル

Rの基本データ形式はベクトルです。関数 c は複数の値を連結します。

```
> x <- c(1, 3, 5) # 長さ1のベクトル3つを連結
> print(x)
[1] 1 3 5
> length(x) # ベクトルのサイズ（長さ）
[1] 3
```

コロン（:）演算子は1刻みの整数ベクトルを返します。

```
> 1:20
 [1]  1  2  3  4  5  6  7  8  9 10
[11] 11 12 13 14 15 16 17 18 19 20
```

数値演算

加減乗除(+ - * /)や累乗(^)などの数値演算が可能です。
演算もベクトル単位で行われます。

```
> 1 + 2 * 3 - 4
[1] 3
> (1:5)^2
[1]  1  4  9 16 25
```

ベクトル同士の演算で長さが異なる場合、短い方が反復されます。

```
> 1:10 / c(1, 2)
 [1] 1 1 3 2 5 3 7 4 9 5
```

論理演算

比較(== != > < >= <=)は真偽値ベクトルを返します。
論理積 &, 論理和 |, 否定 ! による論理演算が可能です。

```
> x <- 1:5 > 3
> x
[1] FALSE FALSE FALSE  TRUE  TRUE
> !x
[1]  TRUE  TRUE  TRUE FALSE FALSE
> y <- c(T, F, F, F, T) # TRUE/FALSEの省略表記
> x & y
[1] FALSE FALSE FALSE FALSE  TRUE
> x | y
[1]  TRUE FALSE FALSE  TRUE  TRUE
```

欠損値

欠損値、すなわち「値がないこと」をNA（Not Available）で示します。

```
> x <- 1:5
> x[5] <- NA # 5番目を欠損値にする
> x
[1]  1  2  3  4 NA
```

欠損値に対する演算結果も欠損値になりますが、
is.na関数で「欠損値かどうか」を判定することができます。

```
> x <= 3
[1]  TRUE  TRUE  TRUE FALSE    NA
> is.na(x)
[1] FALSE FALSE FALSE FALSE  TRUE
```

ベクトルの要素

`x[i]`とすることでxのi番目の要素を取り出すことができます。
Rでは添え字は0ではなく1から始まります。

```
> x <- c(1, 3, 5, 7, 9)
> x[2]
[1] 3
```

添え字をベクトルで指定することもできます。

```
> x[c(1, 3)]
[1] 1 5

> x[2:4]
[1] 3 5 7
```

要素への代入

ベクトルの要素に値を代入することができます。
異なる型のデータを代入すると、型が変換されることに注意しましょう。

```
> x[2] <- 30
> x
[1]  1 30  5  7  9

> x[2:4] <- 0:-2 # 0:-2 = c(0, -1, -2)
> x
[1]  1  0 -1 -2  9

> x[1:3] <- "abc" # x全体が文字列ベクトルになる
> x
[1] "abc" "abc" "abc" "-2"  "9"
```

名前つきのベクトル

ベクトルの要素に名前をつけて、名前で要素を指定することができます。
名前はnames関数で取り出したり、作成や変更も可能です。

```
> x <- c(a = 1, b = 2, c = 3)
> names(x)
[1] "a" "b" "c"
> x["b"]
b
2

> names(x) <- c("x", "y", "z")
> x
x y z
1 2 3
```

論理値による要素の指定

論理値のベクトルを用いて要素を指定することができます。
同じ長さの論理値ベクトルを与え、TRUEに対応する要素が指定されます。

```
> x <- 1:5
> y <- x^2
> x > 3
[1] FALSE FALSE FALSE  TRUE  TRUE

> y[x > 3]
[1] 16 25
```

マッチング

x %in% y で「各xの値がyに含まれるか」を判定することができます。

```
> c("a", "b", "c") %in% c("c", "d")
[1] FALSE FALSE  TRUE
```

特定の名前に当てはまる要素のみ置換する、といった応用が可能です。

```
> x <- c(a = 1, b = 2, c = 3)
> x[names(x) %in% c("c", "d")] <- 0
> x
a b c
1 2 0
```

ベクトルと条件式

`ifelse(条件, 真の場合, 偽の場合)`を用いて、ベクトルの要素ごとに条件分岐を記述することができます。

```
> x <- 1:5
> ifelse(x > 3, "BIG", "small")
[1] "small" "small" "small" "BIG"   "BIG"
```

以下と同様の結果をよりシンプルに得ることができます。

```
> x <- 1:5
> y <- rep("small", 5) # "small"を5回繰り返し
> y[x > 3] <- "BIG"
> y
[1] "small" "small" "small" "BIG"   "BIG"
```

サンプルデータ

data関数を用いてサンプルデータを読み込むことができます。
前出のirisデータを読み込んで、head関数でデータの一部を表示します。
headはジェネリック関数で、データフレームの場合は始めの数行を返します。

```
> data(iris)
> ?iris
> head(iris, 5) # 最初の5行を取り出す
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.1         3.5          1.4         0.2  setosa
2          4.9         3.0          1.4         0.2  setosa
3          4.7         3.2          1.3         0.2  setosa
4          4.6         3.1          1.5         0.2  setosa
5          5.0         3.6          1.4         0.2  setosa
```

データフレーム

irisはテーブルを表すデータフレーム(data.frame)クラスのデータです。
class関数で確認してみましょう。

```
> class(iris)
[1] "data.frame"
```

データフレームのサイズは以下のように確認することができます。

```
> nrow(iris) # 行数, number of rows
[1] 150
> ncol(iris) # 列数, number of columns
[1] 5
> dim(iris)  # 行数と列数, dimensions
[1] 150   5
```

列の取得

データフレームは列単位でデータを保持しています。
列名はnames関数で確認することができ、
データ$列名の形で列のベクトルを取り出すことができます。

```
> names(iris)
[1] "Sepal.Length" "Sepal.Width"  "Petal.Length"
[4] "Petal.Width"  "Species"

> mean(iris$Sepal.Length) # meanは平均値を求める関数
[1] 5.843333
```

列の追加・編集

データ$列名 <- 定義の形で新しい列を追加したり、
既存の列を書き換えることができます。

```
> x <- iris
> x$dummy <- x$Sepal.Length > 5

> names(x)
[1] "Sepal.Length" "Sepal.Width"  "Petal.Length"
[4] "Petal.Width"  "Species"      "dummy"
```

部分データの抽出

データ[行,], データ[,列], データ[行,列]といった指定も可能です。

```
> iris[1:2,] # 1,2行目
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.1         3.5          1.4         0.2  setosa
2          4.9         3.0          1.4         0.2  setosa

> iris[3, c(T, T, F, F, T)] # 3行目, 1,2,5列目
 Sepal.Length Sepal.Width Species
3          4.7         3.2  setosa

> iris[4, -(2:3)] # 4行目, 2,3列目を除く
Sepal.Length Petal.Width Species
4          4.6         0.2  setosa
```

部分データの抽出

データ[,列]の形で列をひとつだけ取り出した場合、結果はベクトルとなります。
1列だけを持つデータフレームが必要な場合、drop=FALSEを指定します。

```
> iris[, "Petal.Length"] # iris$Petal.Length と同じ
  [1] 1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 1.5 ...（省略）

> iris[, "Petal.Length", drop=FALSE] # 結果をデータフレームで返す
Petal.Length
1            1.4
2            1.4
3            1.3
4            ...（省略）
```

部分データの置換

指定した要素のみを置換することも可能です。
以下では値を欠損値に置換していますが、実際の運用では逆に
欠損値を別の値（例：0，"不明"など）に置き換えることが多いです。

```
> x <- iris
> x[3, "Sepal.Length"] <- NA # 欠損値に置き換え

> x[3,]
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
3           NA         3.2          1.3         0.2  setosa
```

factorクラス

アヤメの種を表すSpecies列は文字列ではなく、factorとなっています。
factorは取り得る値のセットを保持したクラスです。

```
> class(iris$Species)
[1] "factor"

> iris$Species[1:3]
[1] setosa setosa setosa
Levels: setosa versicolor virginica
```

factorの特徴

factorの要素に範囲外の値を入力すると、欠損値に置き換えられます。

```
> iris$Species[1] <- "new value"
Warning message:
... （省略） ...
invalid factor level, NA generated

> iris$Species[1:3]
[1] <NA>   setosa setosa
Levels: setosa versicolor virginica
```

factorはデータファイルの読み込み時に自動的に設定されることも多く、想定外のタイミングで欠損が生じないように注意が必要です。

テキストデータの読み込み

タブ区切りやカンマ区切りなどのテキストファイルを
データフレームとして読み込むことができます。
この例では最初の行が列名を表すヘッダ行、区切り文字はタブ(¥t)、
エンコーディングはUTF-8を指定しています。

```
> train <- read.table("data/boston_train.txt",
                      header = TRUE, sep = "¥t",
                      fileEncoding = "utf-8")
> head(train)
  市名  住宅価格   犯罪率  可住地域率  工業地域率
1 Swampscott     21.6 0.02731           0       7.07
2 Swampscott     34.7 0.02729           0       7.07
...(省略)
```

テキストデータの読み込み

read.tableは標準で文字列をfactorとして読み込みます。
文字列のままで読み込むには、stringsAsFactors = FALSEとします。

```
> class(train$市名)
[1] "factor"
> train2 <- read.table("data/boston_train.txt",
                       header = TRUE, sep = "¥t",
                       fileEncoding = "utf-8",
                       stringsAsFactors = FALSE)
> class(train2$市名)
[1] "character"
```

with関数とsubset関数

with関数を用いることで、データフレームの列をいちいち取り出さず
通常のRオブジェクトであるかのように式を記述することができます。

```
> x <- with(iris, Sepal.Length > Sepal.Width * 1.5)
> sum(x) # logicalの和はTRUEの数に等しい
[1] 117
```

subset関数もこれと似たような書式で、条件式を記述すると
条件を満たす行だけを抽出します。

```
> s <- subset(iris, Sepal.Length > Sepal.Width * 1.5)
> dim(s) # 抽出したデータの行数と列数を調べる
[1] 117   5
```

パッケージ

Rにおけるさまざまな機能はパッケージ単位で管理されています。
`library`関数を使ってパッケージを読み込むことで機能を有効可します。

```
> model <- rpart(Species ~ ., data = iris) # 決定木モデル
Error: could not find function "rpart"

> library(rpart) # 決定木パッケージを読み込み
> model <- rpart(Species ~ ., data = iris)
```

標準ではないパッケージについては事前にインストールが必要です。
パッケージのインストールには `install.packages` 関数を使います。

予測モデルの作成

目的変数と説明変数、データを与えることで予測モデルを作成します。
まずは Species を予測する決定木モデルを作成しましょう。

```
> library(rpart) # 決定木パッケージを読み込み
> model <- rpart(Species ~ ., data = iris)
> plot(model); text(model) # 決定木の描画
```

目的変数として数値や真偽値を指定することもできます。
以下は数値を予測する例で、この場合は回帰木と呼ばれます。

```
> model <- rpart(Sepal.Length ~ ., data = iris)
> plot(model); text(model)
```

予測モデルの作成

Species ~ . の部分は formula といい、目的変数を左辺、
説明変数を右辺に記述します。ドット(.) は「すべての変数」を表します。
y ~ x1 + x2 + x3 のように、一部の変数を指定することもできます。

```
> model <- rpart(Species ~ Petal.Length + Petal.Width,
                 data = iris)
> plot(model); text(model)
```

マイナス(-)を使うと、除外する変数を指定することができます。

```
> model <- rpart(Species ~ . - Sepal.Length, data = iris)
> plot(model); text(model)
```

予測

予測の実施は predict 関数で行います。
モデルと予測対象のデータを指定します。

```
> model <- rpart(Species ~ ., data = iris)
> predict(model, newdata = iris[100:102,])
    setosa versicolor  virginica
100      0 0.90740741 0.09259259
101      0 0.02173913 0.97826087
102      0 0.02173913 0.97826087
```

この場合、予測値はそのクラスである確率の予測値を示します。
例えば100行目はversicolorである確率が0.907、すなわち90.7%となります。

予測

予測値を確率ではなく値で得たい場合、type = "class" を指定します。

```
> predict(model, newdata = iris[100:102,], type = "class")
      100         101         102
versicolor  virginica  virginica
Levels: setosa versicolor virginica
```

予測値をデータフレームの列にすることもできます。

```
> x <- iris
> model <- rpart(Species ~ ., data = x)
> x$pred <- predict(model, newdaa = x, type = "class")
> head(x)
```

Oracle R Enterprise

Oracle R Enterpriseを使うと、Oracleデータベースに接続して
データベース上のテーブルをデータフレームのように扱うことができます。

```
> ore.create(iris, table="IRIS") # データをOracle DBにコピー
> head(IRIS)
 Sepal.Length Sepal.Width Petal.Length Petal.Width   Species
1          6.7         3.3          5.7         2.5 virginica
2          6.7         3.0          5.2         2.3 virginica
...（省略）
```

ここでIRISの実体はデータベース上のテーブルです。
Rで記述した処理が自動的にSQLに変換され、まるでデータフレームを
扱っているかのように結果が表示されています。

Oracle R Enterprise

データベース上のテーブルを使って予測モデルを作成したり、
モデルを使って予測を実行することもできます。

```
> model <- ore.odmDT(Species ~ ., data = IRIS) # 決定木モデル
> PRED <- predict(model, newdata = IRIS)
> head(PRED)
  'setosa' 'versicolor' 'virginica' PREDICTION
1        0   0.02173913   0.9782609  virginica
2        0   0.02173913   0.9782609  virginica
...(省略)
```

ef-prime, inc.

休憩

ef-prime, inc.

実践編

Rで予測分析

いよいよRで予測分析を実行します。
既に学んだRの基礎を復習しながら、スクリプトを記述して
サンプルデータを分析してみましょう。

ボストンの住宅価格データ

- データについて
  - 概要
    - 米国ボストンの506地点について、一戸建て家屋の住宅価格と関連する様々な指標を調査したデータです。
    - オリジナルデータを学習用と検証用に分割しています。
  - 形式
    - タブ区切りのヘッダ付きテキストデータで、文字コードはUTF-8です。
  - ファイル
    - boston_train.txt　（学習用）
    - boston_test.txt　（検証用）

ボストンの住宅価格データ

- 課題
  - データファイルをRのデータフレームとして読み込みます。
    - オブジェクト名は学習用を `train` 、検証用を `test` とします。
  - 目的変数を住宅価格とし、学習用データを使って
    回帰木モデルを作ってみましょう。
    - 作成したモデルを表示してみましょう。
      ここでは`plot`よりも`print`関数が見やすいかも知れません。
  - 検証用データに対して予測を適用してみましょう。
    - 結果はオブジェクトに代入せず、そのまま表示して構いません。
    - 何が起きましたか？

解答例

```
train <- read.table("data/boston_train.txt", sep = "¥t",
       header = TRUE, fileEncoding = "utf-8")
test <- read.table("data/boston_test.txt", sep = "¥t",
       header = TRUE, fileEncoding = "utf-8")

library(rpart)
model <- rpart(住宅価格 ~ ., data = train)

plot(model); text(model)
print(model)

predict(model, newdata = test)
```

ボストンの住宅価格データ

- 課題
  - 問題のある変数を除いてモデルを作成してみましょう。
    - 残念ながら、rpartでは `y ~ . - z` の形式で
      うまく動作しない部分があります。
    - 以下のように列ごと除いた学習用データを作って対応します：

      ```
      > train.sub <- train[, names(train) != "問題の変数"]
      ```
  - うまく動作したら、検証用データに予測値の列を作りましょう。
    - 変数名は 予測価格 とします。
  - 予測誤差を表す列を作成し、mean関数で平均値を求めましょう。
    - 絶対誤差 = abs(予測価格 - 住宅価格)

解答例

```
train.sub <- train[, names(train) != "市名"]
model <- rpart(住宅価格 ~ ., data = train.sub)

test$予測価格 <- predict(model, newdata = test)
test$絶対誤差 <- with(test, abs(予測価格 - 住宅価格))

mean(test$絶対誤差)
```

ボストンの住宅価格データ

- 課題
  - 同様に、0/1変数を予測する決定木を作ってみましょう。
    目的変数は チャールズ河流域 とします。
    - 実用上の意味はなく、あくまで練習問題です。
  - 検証用データに予測を適用し、予測確率という列に代入します。
    - 目的変数が真偽値または0/1変数のとき、
      標準では1（またはTRUE）となる確率が出力されます。
  - 予測確率が0.5以上のとき1となる予測値という列を作成しましょう。
    - 予測値と実際の値が一致するとき1、それ以外は0となる列
      「正答」を作成しましょう。
    - 続いてmean関数で平均値を求め、正答率を算出しましょう。

解答例

```
model <- rpart(チャールズ河流域 ~ ., data = train.sub)

test$予測確率 <- predict(model, newdata = test)
test$予測値 <- ifelse(test$予測確率 >= 0.5, 1, 0)
test$正答 <- with(test, ifelse(予測値 == チャールズ河流域, 1, 0))

mean(test$正答)
```

実践ビジネス予測分析

実際のマーケティングデータを用いて予測分析を実施します。

銀行のテレマーケティングデータ

- データについて
  - 概要
    - ポルトガルの銀行で実施されたダイレクトマーケティングのキャンペーンに関するデータです。
    - キャンペーンは定期預金の申し込みを訴求するもので、電話によって行われました。申し込みの有無が判明するまで、一人の顧客に対して複数回のコンタクトが試みられるケースも多くあります。
  - 形式
    - タブ区切りのヘッダ付きテキストデータで、文字コードはUTF-8です。
  - ファイル
    - bank.txt

銀行のテレマーケティングデータ

- 課題
  - データファイルをRのデータフレームとして読み込みます。
  - データ項目を眺めてみましょう。
    - 「本件」とあるのはこのキャンペーンでの通話等に関する項目、
      「前回」はそのひとつ前のキャンペーンに関する項目です。
  - データを学習用と検証用に分割しましょう。
    - 以下のコードを参考にしてください。ランダムサンプリングを行う
      `sample`関数、割り算の商を求める `%/%` 演算子を使っています。

```
> smpl <- sample(nrow(bank), size = nrow(bank) %/% 2)
> train <- bank[sort(smpl),]
> test  <- bank[-smpl,]
```

解答例

```
bank <- read.table("data/bank.txt", sep = "¥t",
        header = TRUE, fileEncoding = "utf-8")

smpl <- sample(nrow(bank), size = nrow(bank) %/% 2)
train <- bank[sort(smpl),]
test  <- bank[-smpl,]
```

銀行のテレマーケティングデータ

- 課題
  - 学習用データで予測モデルを作ってみましょう。
    モデルを見て何か気付くことはありませんか？

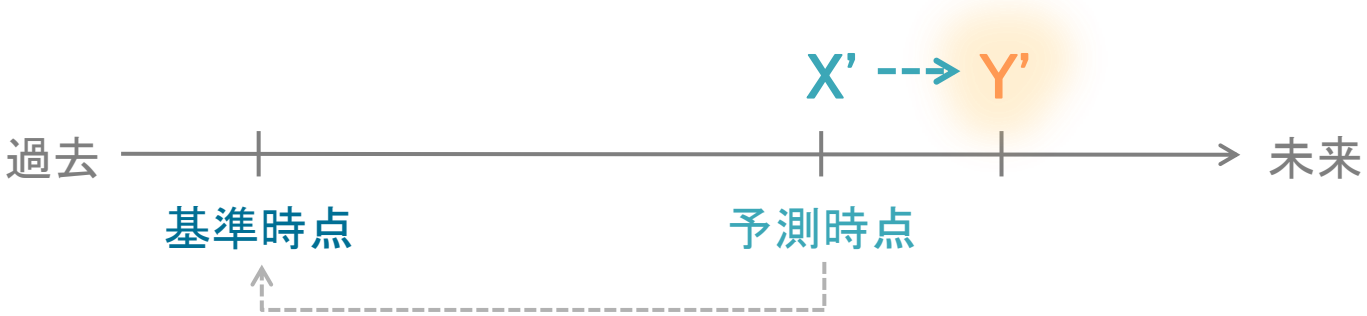
```
model <- rpart(定期預金申込有無 ~ ., data = train)
plot(model); text(model)
```

データの時点

予測対象が未来の値であるとき、分析用データの作成に注意が必要です。
ある時点（予測時点）で既知の情報 X’ から将来の Y’ を予測するとします。

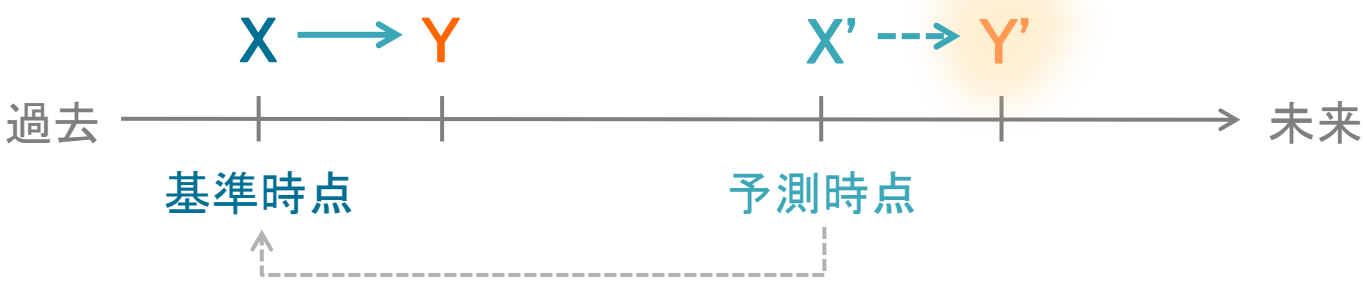
データの時点

これに対応する過去データを取得するため、予測時点から
過去に時間を遡って基準時点（例：1年前の同じ日付）を定めます。



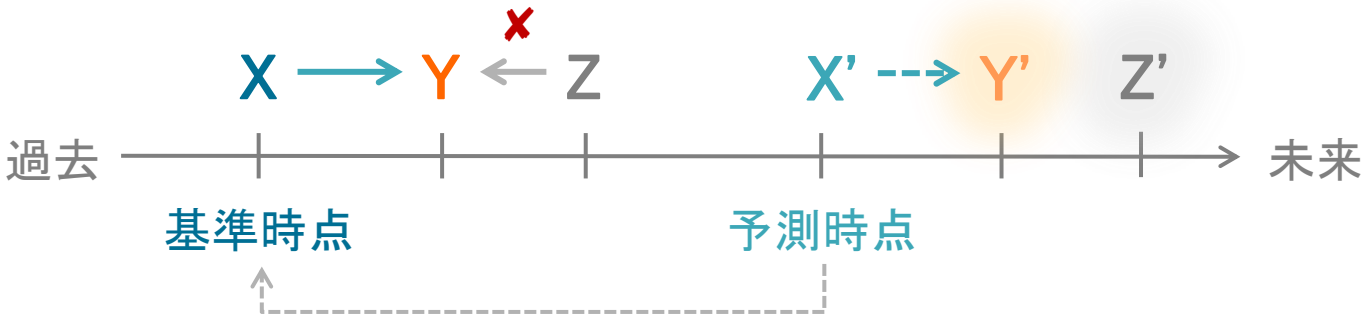
データの時点

当時の時点で既知の情報 X と、将来にあたる時点の Y からなるデータを作成し、X から Y を予測するモデルを作成します。
作成したモデルに予測時点の X' を与えて、将来の Y' を予測します。



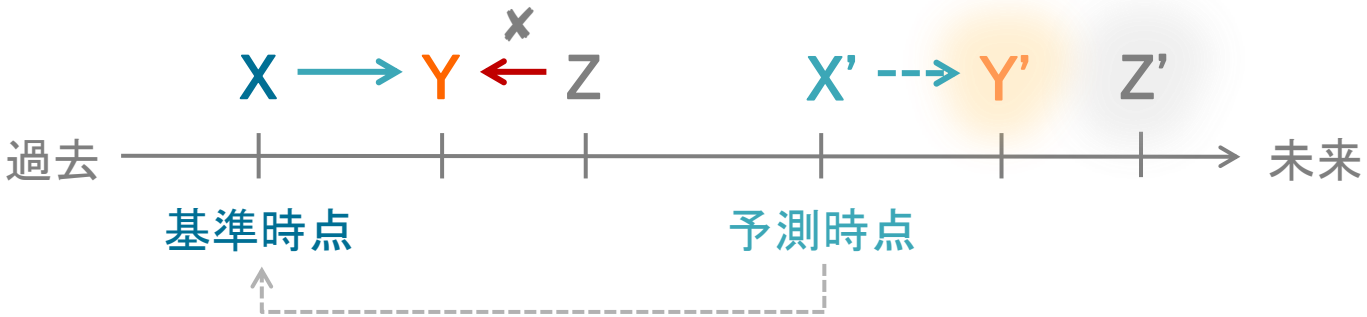
データの時点

分析用データの設計を誤ると、基準時点においては
未知であるはずの情報 Z が紛れ込んでしまう場合があります。
Z に対応する情報 Z' は予測時点で未知なので、使ってはいけません。



データの時点

ところが Z は Y より後の時点で得られる情報であることから、
Z を含むモデルは高い予測精度を達成してしまいます。
しかし当然、実際の予測ではそのような精度を得ることはできません。



Leakage

このような「使ってはいけないデータ」が含まれてしまう現象は
Leakage（リーク、漏洩）と呼ばれ、データ設計時には特に注意が必要です。
到底起こりえないように思えますが、以下のように比較的発生しやすい現象です。

- Leakageの例
  - マスタテーブルの値が変わってしまう場合
    - 無料会員登録の時点では存在しなかったクレジットカード情報。
      クレジットカード情報がある場合、全員が購入ありと予測されてしまう
    - 性別。見込みの時点では未登録が存在するが、正式入会時には
      必ず入力する場合、性別未登録は全員が入会なしと予測される
  - トランザクションが残らない情報
    - 購入をキャンセルした場合にフラグが立つが、いつキャンセルしたか
      わからない場合。当時キャンセル済みだったかどうか判別できない

銀行のテレマーケティングデータ

- 課題
  - 「使ってはいけないデータ」を除外してモデルを作ってみましょう。
    - 「使ってはいけない変数」はどれでしょうか？
    - 変数以外にも、使ってはいけないデータはないでしょうか？
  - 予測確率、予測値を算出し、正答率を出してみましょう。

解答例

```
train.sub <- subset(train, 本件コンタクトタイプ != "不明")
test.sub  <- subset(test,  本件コンタクトタイプ != "不明")
train.sub <- train.sub[, !(names(train.sub) %in%
       c("本件コンタクトタイプ", "本件最終通話日", "本件最終通話月",
       "本件最終通話秒数", "本件コンタクト回数"))]

model <- rpart(定期預金申込有無 ~ ., data = train.sub)
plot(model); text(model)

test$予測確率 <- predict(model, newdata = test)
test$予測値 <- ifelse(test$予測確率 >= 0.5, 1, 0)
test$正答 <- with(test, ifelse(予測値 == 定期預金申込有無, 1, 0))
mean(test$正答)
```

銀行のテレマーケティングデータ

- より進んだ応用
  - 以下のモデルを作ってみましょう:
  - rangerパッケージによるランダムフォレストモデル
  - xgboostパッケージによるブースティングモデル

  - ランダムフォレスト、ブースティングとは？
    - 複数の決定木、回帰木を作成し、予測値を組み合わせるモデル
      - ランダムフォレストは元データのランダムな複製に基づいて木を作る
      - ブースティングは予測精度を順次向上させるように木を作る

解答例

```
library(ranger)
model.RF <- ranger(定期預金申込有無 ~ ., data = train.sub)

test$予測確率.RF <- predict(model.RF, data = test)$predictions
...
```

xgboostの場合はもう少しややこしいが、基本は同じ