この記事は R と競馬データで学ぶ統計学シリーズの一部です。

はじめに

R と競馬データで学ぶ統計学シリーズ第 4 回は、R の機能を強化する「パッケージ」について紹介します。無償で使用できる数千を超えるパッケージの中から、競馬データを処理する際にあると便利なパッケージを取り上げます。

この記事では、はじめにパッケージを使用するための一般的な手順を紹介し、次にデータを効率的に処理するための dplyr パッケージと RcppRoll パッケージの使用法について紹介します。

パッケージの活用

R には、インストールしたての状態でも、多数の関数が用意されています。しかし、「やりたいこと」がそれらの関数だけでは実現できない場合、

- 自分で所望の処理を R 言語のプログラムとして実装する
- やりたいこと（の一部）を機能として提供しているパッケージを探し、導入する

のいずれかのアプローチを取ることになります。このうち、自分で処理を実装する方法は、もちろん何でも自由に実現できますが、R 言語に関する知識に加え、対象となる処理、アルゴリズムについての深い理解が必要になります。一方、これから紹介するパッケージを使う方法は、詳細を知らなくても、「やりたいことができる関数」を使えば、所望の結果が得られます。

パッケージと CRAN（と GitHub）

ここまで明確な説明なしに「パッケージ」を連呼してきましたが、ここでパッケージとは何か、どこで手に入るのかについて紹介します。

■パッケージとは

パッケージとは、開発者が作成した関数やデータセットなどをまとめて、所定のフォルダ構造で配置したファイル群です。ユーザーはパッケージをダウンロードし、R がアクセス可能なパスにインストールすることで、その機能を利用できます。

■ CRAN（Comprehensive R Archive Network）

パッケージは、基本的に CRAN という Web サイトからダウンロードします。CRAN には、R プロジェクトの審査を通過した、（ある程度）信頼できるパッケージが登録されています。

なお、Microsoft R Open においては、Microsoft がある時点で凍結した CRAN のコピー（MRAN; Microsoft R Application Network）からダウンロードするよう設定されています（変更は可能）。

■パッケージのインストール

R には、CRAN / MRAN から指定のパッケージを取得し、自動的にインストールしてくれる install.packages 関数があります。引数に、インストールしたいパッケージ名を指定して実行すると、ダウンロード、所定のフォルダへのインストールまで自動的に行います。「所定のフォルダ」は、設定を変更しない限り C:\Program Files\Microsoft\MRO-3.3.2\library などになります。

例えば、本記事で詳しく紹介する、データを効率的に加工するための dplyr パッケージをインス

トールするには、以下のようにします。

```
install.packages("dplyr") # パッケージ名をダブルクォーテーションで囲う
```

なお、Mac OS X や Unix / Linux などの環境では、パッケージのソースコードがダウンロードされ、手元でコンパイルが始まります。そのため、gcc などの開発環境を用意する必要があります。

■ GitHub

パッケージの多くは CRAN に登録されますが、近年では GitHub 上に最新の開発版や、CRAN に登録するまでもない（と作者が考えているかもしれない）パッケージが多く公開されています。

GitHub 上で公開されているパッケージをインストールするには、あらかじめ CRAN から devtools パッケージをインストールしておく必要があります。devtools パッケージが提供する install_github 関数の引数に、“ GitHub ユーザー名 / レポジトリ名 ”と指定することで、パッケージをインストールできます。

また、日本の著名な R ユーザー達による“ 匿名知的集団ホクソエム ”が開発した githubinstall パッケージも CRAN に登録されています。こちらは、あいまいなパッケージ名のサジェストなどにも対応した、より高機能なパッケージです。

■パッケージの探し方

ここでは、導入したいパッケージの名前がわかっているものとして説明しましたが、実際には、「○○分析をしたい」というニーズに対応するパッケージ名がわからない場合が多いでしょう。現在のところ、便利な逆引き機能や Web サイトは（少なくとも日本語では）整っておらず、Web 検索などを通じてパッケージ名を特定することになります。R ユーザーのブログなどで実際の使用感などを確認して導入すると良いでしょうが、その際には記事の日付情報を確認し、できるだけ最新の情報源をもとに判断しましょう。特に機械学習などの進展が著しい分野では、1 年前に主流だったパッケージがもう古いものになっている場合があります。

英語では、CRAN Task Views や Awesome R など、ある程度所望のパッケージに行き着く手段がありますが、それでも、“ 検索力 ” を試される状況は変わりません。

パッケージの読み込み

パッケージは、インストールしただけでは使用できません。library 関数または require 関数で読み込んではじめて、使用できるようになります。2 つの関数はほぼ同じ機能なので、好きなほう（あれば）を使えばよいでしょう。一般的には、library 関数を使う例が多いようです。

```
library( パッケージ名 ) # ダブルクォーテーションはつけない
```

library 関数でパッケージを読み込む際には、パッケージ名はダブルクォーテーションで囲みませんので注意してください。

ここまで、パッケージを活用するための一般論を紹介しました。ここからは、競馬データを処理する際に便利な、dplyr パッケージと RcppRoll パッケージの使いかたを紹介します。

dplyr パッケージ

dplyr（ディープライヤー）パッケージは、データフレームから必要なデータをを柔軟に抽出、加工、集計するための、たいへん有用なパッケージです。開発者の Hadley Wickham は、革新的なパッケージを数多く開発し、R のデータサイエンス領域における活用に大きく寄与しています。日本の R ユーザーの間では、“ Hadley 神 ” とも呼ばれています。

dplyr パッケージは、Hadley が提唱する “ Split-Apply-Combine ” アプローチによるデータ処理を R で効率的に行うために開発、提供されています。

以前は plyr というパッケージで同等の機能を提供していましたが、現在は dplyr パッケージに進化しています。また、Hadley の思想自体も進化して、現在では “ Tidy Data ” という概念を提唱しています。Tidy Data を実現するために、dplyr だけでなく、tidyr、purrr など様々なパッケージが Hadley らにより開発されています。それらの、有用なパッケージをひとまとめにインストールするためのメタパッケージとして、tidyverse パッケージが提供されています。

dplyr あるいは tidyverse パッケージをインストールするには以下のようにします。

```
install.packages("dplyr") # dplyr パッケージのみをインストール
```

または

```
install.packages("tidyverse") # Hadley らによる便利なパッケージ群をまとめてインストール
```

本記事では、dplyr パッケージの多様な機能のうち、CSV などのテキストファイルで記録された競馬データを、条件ごとに抽出し、何らかの加工を施し、集計する作業を行う際に便利な機能を中心に紹介します。

なお、dplyr パッケージに関する大変よくまとまった記事として、以下があります。

- dplyr を使いこなす！基礎編
- dplyr を使いこなす！ Window 関数編
- dplyr を使いこなす！ JOIN 編

パイプ（%>%）演算子

パイプ（%>%）演算子は、dplyr パッケージの根幹をなす機能です。パイプ演算子を使うことで、データに対してある関数を適用した結果を、次の関数へ、さらに次の関数へ、と受け渡すことができます。

パイプ演算子を使うことで、ある関数の実行結果を一時オブジェクトに保存し、そのオブジェクトに対しまた別の関数を適用する、という手間（とメモリの無駄）を省けるようになりました。パイプ演算子自体は、データを次の関数に受け渡す機能しか持たないので、実際のデータ抽出、加工、集計は、以下のそれぞれの関数で行います。

なお、パイプ演算子を使う場合、前の関数から後の関数へのデータの受け渡しは、暗黙のうちにおこなわれる場合と、「.」で指定する必要がある場合があります。

```
データ %>% group_by( 競馬場 ) %>% select( 必要なデータ ) %>% do(model = lm( タイム ~ 予測に使用するデー
```

```
タ , data = .))
```

例えば、上記のような処理（競馬場ごとにタイムの線形回帰による予測モデルを作成する）を行う場合、dplyr パッケージに含まれる group_by 関数や select 関数（後述）は、暗黙のうちに前の関数からデータを受け取っていますが、lm 関数においては、data = . と、使用するデータは前の関数から受け取った「.」であると宣言する必要があります。

select 関数

select 関数は、データフレームから特定の列だけを取り出す、あるいは除外する際に使います。

```
select( オブジェクト , 条件 )
 （例）
library(dplyr) # すでに読み込んでいる場合は不要
jra_data <- read.csv("jra_race_result.csv", header = TRUE) # 中央競馬データの読み込み
jra_data$ 開催日 <- as.Date(jra_data$ 開催日 )

jra_result_odds <- jra_data %>% select( 着順 , オッズ ) # データから、着順と（単勝）オッズだけを取り出す

# 欠損値を除外して相関係数を算出してみる
cor(na.omit(jra_result_odds), method = "spearman")
             着順     オッズ
着順   1.0000000 0.5799023 # “ 中程度の相関 ” が見られる
オッズ 0.5799023 1.0000000
```

この例では、select 関数で JRA の競走成績データ（全 30 列）から「着順」と「オッズ」の 2 列だけを取り出しています。そして、得られたデータについて相関係数（第 3 回参照）を算出しています。結果は、およそ 0.58 と、“ 中程度の相関 ” があると主張できる値になっています。つまり、オッズが低い（人気がある）馬は、良い着順になりがちである、という当たり前の結果です。逆に、人気と実際の結果の間には、中程度の関係性しかないという捉え方もできます。

なお、上記のプログラムは、相関係数の算出を別行で実行していますが、パイプ演算子を使って、以下のように書くこともできます。

```
na.omit(jra_data) %>% select( 着順 , オッズ ) %>% cor(., method = "spearman")
```

filter 関数

filter 関数は、データから条件に合致する行だけを抽出します。条件にはさまざまな演算子が指定できます。一般的な >, >=, <=, <, ==, != に加え、「いずれかに一致する」%in% 演算子などがあります。また、複数の条件を & または ,（AND 条件）あるいは |（OR 条件）演算子で指定できます。

```
filter( 条件 )
```

例えば、「一本被り」（いっぽんかぶり）と呼ばれるような、オッズが 2 倍を切るようなケースだけ抽出するには、以下のようにします。

```
# データから、単勝オッズ 2 倍を切るケースのオッズと着順だけを取り出す
jra_result_odds <- jra_data %>% select( 着順 , オッズ ) %>% filter( オッズ <2)

# 欠損値を除外して順位相関係数を算出してみる
na.omit(jra_data) %>% select( 着順 , オッズ ) %>% filter( オッズ < 2) %>% cor(., method = "spearman")
             着順     オッズ
着順   1.0000000 0.1849791
オッズ 0.1849791 1.0000000
```

なんと、相関係数はさらに小さくなってしまいました。オッズが 2 倍を切るということは、(単勝を買う）多くの人が「この馬が 1 着」と思っているにもかかわらず、実際の結果はそうなりにくいということです。

group_by 関数

group_by 関数は、データを指定した列に含まれる水準でグループ化するために使います。例えば、同じ開催日、同じ騎手、同じ馬などの基準でデータをまとめることができます。group_by 関数を単体で使うことはあまりなく、大抵はグループに対して何らかの集計処理（グループに含まれる値の合計、平均など）を続けて行います。競馬に限らず、一定量以上のデータを扱う場合、データの特性をグループ別に把握したい場合が多くあります。group_by 関数は大量データを扱う際に有用な関数です。

```
group_by( 列名 )
```

ここでは、JRA の競馬データを騎手ごとにグループ化し、単勝オッズ 2 倍を切るケースにどれくらい騎乗しているかを調べてみましょう。

```
jra_result_low_odds <- jra_data %>% filter( 開催日 >= "2016-01-01") %>% group_by( 騎手 ) %>%
filter( オッズ < 2) %>% select( 騎手 , 着順 )
jra_result_low_odds$ 騎手 <- droplevels(jra_result_low_odds$ 騎手 )
table(jra_result_low_odds)
            着順
騎手          1  2  3  4  5  6  7  8  9 10 11 12 13 15 16
C. ルメール   42 11  8  6  6  3  1  1  1  0  0  1  1  0  0
M. デムーロ   28 15  6  5  6  2  1  1  0  0  1  1  1  0  0
岩田 康誠     10  6  3  3  0  0  0  0  0  0  0  0  0  0  0
戸崎 圭太     39 17 10  7  2  2  1  4  0  0  0  0  0  0  0
川田 将雅     28  7  4  0  4  1  3  0  1  0  2  1  0  0  0
武 豊         15  7  2  2  1  0  0  0  0  0  0  0  0  0  0
福永 祐一     11  3  5  5  2  2  0  1  2  0  1  1  0  0  0
蛯名 正義      9  1  2  1  0  1  0  0  1  1  0  0  0  0  0
...
```

droplevels 関数は、factor 型の変数について、値の存在しない（0 件 =2 倍を切るオッズの馬に騎乗していない）水準を削除します。table 関数については第 3 回を参照してください。

summarize 関数

summarize 関数（summarise でも動作します）は、データに引数に指定した処理を適用し、要約・集計するために使います。group_by 関数や filter 関数などで抽出した集計対象のグループに対して、合計や平均、その他任意の統計量を算出して返します。

```
summarize( 列名 = 処理 )
```

例えば、JRA の競馬データから、2016 年度の 11 月時点での騎手別勝利数（リーディング）を集計するには以下のようなプログラムを実行します。

```
# 開催日を Date 型に変換しておく必要がある（上記参照）
jra_data %>% filter( 開催日 >= "2016-01-01") %>% group_by( 騎手 ) %>% select( 着順 ) %>%
summarize( 勝利数 = sum( 着順 == 1)) %>% arrange(desc( 勝利数 ))
# A tibble: 182 × 2
       関取焔 勝利数 # Windows 環境ではグループ化対象の変数名で文字化けが発生する
       <fctr>  <int>
1   戸崎 圭太     170
2  C. ルメール     164
3  M. デムーロ     127
4   川田 将雅     121
```

```
5   福永 祐一    105
6   内田 博幸     83
7   田辺 裕信     75
8       武 豊     68
9   岩田 康誠     65
10  和田 竜二     64
# ... with 172 more rows
```

また、arrange 関数は引数に指定した列をキーにデータを並べ替えます。desc 関数は降順にするために指定します。

また、もうひとつの例として、先程と同様、単勝オッズが 2 倍を切るケースでの勝率を算出してみましょう。

```
na.omit(jra_data) %>% select( 着順 , オッズ ) %>%  filter( オッズ < 2) %>% summarize( 勝率 = sum( 着順 == 1) / nrow(.))
      勝率
1 0.4921021
```

nrow 関数は、データフレームの行数を算出します。ここでは、JRA の競馬データのうち、(1) 着順とオッズを選択、(2) オッズが 2 倍を切ったケースのみ抽出、(3) 対象馬が 1 着になった確率を算出する処理を行っています。なお、nrow(.) は前の処理から受け取ったデータ（オッズが 2 倍を切ったケース）の行数を意味します。

この結果からは、オッズが 2 倍を切る一本被りのケースでも、実際に 1 着になるのはその半分に満たないことがわかります。筆者自身も、「こんなにオッズが低いなら、頭（1 着）に固定して大丈夫だろう」と三連単などを買い、大失敗することがよくあります……。

mutate 関数

mutate 関数は、データに新たに列を追加します。何らかのフラグや、関数の適用結果などを既存のデータと結合したい場合に使用します。

```
mutate( 列名 = 処理 )
```

ここでは、既存のデータに新たに、「馬券内フラグ」を追加してみます。馬券内とは、1 着から 3 着に入着することです。

```
jra_result_flg <- jra_data %>% select( 着順 , オッズ ) %>% mutate( 馬券内フラグ = ifelse( 着順 <= 3, 1, 0))
head(jra_result_flg)
  着順 オッズ 馬券内フラグ
1    1    1.9            1
2    2   44.1            1
3    3   11.8            1
4    4    5.3            0
5    5    6.1            0
6    6   19.0            0
...
```

ifelse 関数は、ifelse(データ , 条件 , 真の場合の値 , 偽の場合の値) という書式で使います。データは、パイプ演算子により前の処理から暗黙のうちに引き継がれています。

さて、ここまで dplyr パッケージの概要と select 関数、filter 関数、summarize 関数、mutate 関数を紹介してきました。最後に、合わせ技として、以下のようなプログラムを用意しました。

```
na.omit(jra_data) %>% select( 着順 , オッズ ) %>% mutate( 馬券内フラグ = ifelse( 着順 <= 3, 1, 0)) %>%
filter( オッズ < 2) %>% summarize( 馬券内率 = sum( 馬券内フラグ == 1) / nrow(.))
   馬券内率
1 0.8010791
```

これは、JRA の競馬データから（1）着順とオッズを選択、（2）着順から馬券内フラグを追加、（3）オッズが 2 倍を切ったケースのみ抽出、（4）オッズが 2 倍以下の馬が 3 着内に入った確率を算出する、という処理をしています。

結果は、80.1% と高い値になりました。高い支持を得た馬は、勝たないまでも 3 着内には大抵の場合入るという（当たり前の）ことがわかりました。

ここまで例としてみてきた、「単勝オッズが 2 倍を切る場合の成績」については、少し古いデータですが、JRA-VAN のページでも集計されています。

ちなみに、地方競馬データで同様のことをしてみると、以下のようになります。

```
nar_data <- read.csv("nar_race_result.csv")
# オッズが 2 倍を切ったケースの 1 着率
na.omit(nar_data) %>% select( 着順 , オッズ ) %>%  filter( オッズ < 2) %>% summarize( 優勝率 = sum( 着順
== 1) / nrow(.))
     優勝率
1 0.5468185

# オッズが 2 倍を切ったケースの 3 着内率
na.omit(nar_data) %>% select( 着順 , オッズ ) %>% mutate( 馬券内フラグ = ifelse( 着順 <= 3, 1, 0)) %>%
filter( オッズ < 2) %>% summarize( 馬券内率 = sum( 馬券内フラグ == 1) / nrow(.))
   馬券内率
1 0.8541233
```

地方競馬のほうが、一本被りになった場合の信頼度はやや高いようです。

RcppRoll パッケージ

続いて、RcppRoll パッケージを紹介します。このパッケージは、「ウィンドウ関数」のうち、「ローリング処理」を提供します。ウィンドウ関数とは、データ列に対して、一定の大きさの“窓”をかけて、窓の範囲内のデータに対して何らかの処理を適用するものです。ローリング処理は、その窓を少しずつずらしていく、というものです。以下に、例として 2016 年の菊花賞、有馬記念を制したサトノダイヤモンド号の戦績（2016 年 11 月時点）について、窓のサイズを 2 として平均着順を算出するイメージを示しています。

ここでは、まずデビュー戦と 2 戦目で 1 つ目の平均着順（いずれも 1 着なので 1）を算出します。次に、2 戦目と 3 戦目で平均着順（これも 1）を算出、3 戦目と 4 戦目で平均着順を算出（1 着と 3 着で 2）……というように、窓をずらしながら特定の処理を繰り返していきます。

RcppRoll パッケージは、ローリング処理を実現する関数を提供します。RcppRoll パッケージで提供される関数のうち、代表的なものを以下に列挙します。

- roll_max: 窓の範囲内での最大値を返す
- roll_mean: 〃平均を返す
- roll_median: 〃中央値を返す
- roll_min: 〃最小値を返す

・ roll_prod: 〃総乗を返す
・ roll_sd: 〃標準偏差を返す
・ roll_sum: 〃総和を返す
・ roll_var: 〃分散を返す

さらに、これらの関数は r または l を末尾に付与することで、窓の“ 位置 ”を指定できます。この“ 位置 ”については、以下のイメージ図を見ていただくと理解できるでしょうか（説明放棄）。

競馬予想において、このようなローリング処理は、「平均着順」や「上り 3F 平均」などの指標を算出するために広く使われています。筆者も、予測のファクターとして平均着順を算出するため、roll_meanr 関数を使っています。

roll_meanr 関数

roll_meanr 関数は、窓の範囲内の平均を返します。その際、「次に処理する値が窓の右端にくる」ように窓を移動させます。これは、説明よりも実際に結果を見たほうが理解しやすいと思いますので、上記と同様サトノダイヤモンド号の成績を使い、実際に平均着順を算出してみます。

```
jra_data %>% filter(馬名 =="サトノダイヤモンド") %>% select(開催日 ,レース名 ,距離 ,騎手 ,馬体重 ,着順 ) %>%
mutate(平均着順 =roll_meanr(着順 ,n=3,fill=着順 [n()]))
      開催日              レース名 距離       騎手 馬体重 着順 平均着順
1 2015-11-08        サラ系 2 歳新馬 2000 C. ルメール    502    1 1.000000
2 2015-12-26 サラ系 2 歳 500 万円以下 2000 C. ルメール    500    1 1.000000
3 2016-02-07   きさらぎ賞 (GIII) 1800 C. ルメール    498    1 1.000000
4 2016-04-17         皐月賞 (GI) 2000 C. ルメール    504    3 1.666667
5 2016-05-29       東京優駿 (GI) 2400 C. ルメール    500    2 2.000000
6 2016-09-25    神戸新聞杯 (GII) 2400 C. ルメール    500    1 2.000000
7 2016-10-23         菊花賞 (GI) 3000 C. ルメール    498    1 1.333333
```

ちなみに、roll_mean 関数および roll_meanl 関数をそれぞれ適用すると、以下のようになります。

```
# roll_mean 関数
jra_data %>% filter(馬名 =="サトノダイヤモンド") %>% select(開催日 ,レース名 ,距離 ,騎手 ,馬体重 ,着順 ) %>%
mutate(平均着順 =roll_mean(着順 ,n=3,fill=着順 [n()]))
      開催日              レース名 距離       騎手 馬体重 着順 平均着順
1 2015-11-08        サラ系 2 歳新馬 2000 C. ルメール    502    1 1.000000
2 2015-12-26 サラ系 2 歳 500 万円以下 2000 C. ルメール    500    1 1.000000
3 2016-02-07   きさらぎ賞 (GIII) 1800 C. ルメール    498    1 1.666667
4 2016-04-17         皐月賞 (GI) 2000 C. ルメール    504    3 2.000000
5 2016-05-29       東京優駿 (GI) 2400 C. ルメール    500    2 2.000000
6 2016-09-25    神戸新聞杯 (GII) 2400 C. ルメール    500    1 1.333333
7 2016-10-23         菊花賞 (GI) 3000 C. ルメール    498    1 1.000000

# roll_meanl 関数
jra_data %>% filter(馬名 =="サトノダイヤモンド") %>% select(開催日 ,レース名 ,距離 ,騎手 ,馬体重 ,着順 ) %>%
mutate(平均着順 =roll_meanl(着順 ,n=3,fill=着順 [n()]))
      開催日              レース名 距離       騎手 馬体重 着順 平均着順
1 2015-11-08        サラ系 2 歳新馬 2000 C. ルメール    502    1 1.000000
2 2015-12-26 サラ系 2 歳 500 万円以下 2000 C. ルメール    500    1 1.666667
3 2016-02-07   きさらぎ賞 (GIII) 1800 C. ルメール    498    1 2.000000
4 2016-04-17         皐月賞 (GI) 2000 C. ルメール    504    3 2.000000
5 2016-05-29       東京優駿 (GI) 2400 C. ルメール    500    2 1.333333
6 2016-09-25    神戸新聞杯 (GII) 2400 C. ルメール    500    1 1.000000
7 2016-10-23         菊花賞 (GI) 3000 C. ルメール    498    1 1.000000
```

いずれも、「そのレースの時点までの平均着順」として考えると、違和感があります（3 連勝しているのに平均着順が 1.666... など）。なんと説明すればよいのか明確でないですが、“ 違和感のない平均着順 ”を出すためには、roll_meanr 関数を使います。

まとめ

本記事では、R の機能を拡張するパッケージの一般的な使用法と、競馬データを処理する際に有用な dplyr、RcppRoll パッケージの使用例を紹介しました。

記事中で紹介したように、R には数千を超えるパッケージが提供されています。自分の「やりたいこと」に合わせて最適なパッケージを選択することで、処理速度や精度、プログラムの見通しが良くなるでしょう。

注釈