**Prologue**

Package comperank is on CRAN now. It offers consistent implementations of several ranking and rating methods. Originally, it was intended to be my first CRAN package when I started to build it 13 months ago. Back then I was very curious to learn about different ranking and rating methods that are used in sport. This led me to two conclusions:

These discoveries motivated me to write my first ever CRAN package. Things didn’t turn out the way I was planning, and now comperank is actually my fourth. After spending some time writing it I realized that most of the package will be about storing and manipulating competition results in consistent ways. That is how comperes was born.

After diverging into creating this site and ruler in pair with [keyholder](https://echasnovski.github.io/keyholder/), a few months ago I returned to competition results and rankings. Gained experience helped me to improve functional API of both packages which eventually resulted into submitting them to CRAN.

Ruler Package Details

ruler offers a set of tools for creating tidy data validation reports using dplyr grammar of data manipulation. It is structured to be flexible and extendable in terms of creating rules and using their output.

To fully use this package a solid knowledge of dplyr is required. The key idea behind ruler’s design is to validate data by modifying regular dplyr code with as little overhead as possible.

Some functionality is powered by the keyholder package. It is highly recommended to use its supported functions during rule construction. All one- and two-table dplyr verbs applied to local data frames are supported and considered the most appropriate way to create rules.

This README is structured as follows:

* **Installation** shows ways to install package.
* **Example** shows the basic usage of ruler for exploration of obeying user-defined rules and its automatic validation.
* **Overview** explains basic data and function types with design behind them.
* **Usage** describes ruler’s capabilities in more detail.
* **Other packages for validation and assertions** lists alternatives for described tasks.

## Installation

You can install current stable version from CRAN with:

install.packages(**"ruler"**)

## Example

# Utilities functions

is\_integerish **<- function**(x) {

all(x == as.integer(x))

}

z\_score **<- function**(x) {

abs(x - mean(x)) / sd(x)

}

# Define rule packs

my\_packs **<-** list(

data\_packs(

dims = . **%>%** summarise(nrow\_low = nrow(.) >= 10, nrow\_high = nrow(.) <= 15,

ncol\_low = ncol(.) >= 20, ncol\_high = ncol(.) <= 30)

),

group\_packs(

vs\_am\_num = . **%>%** group\_by(vs, am) **%>%** summarise(vs\_am\_low = n() >= 7),

.group\_vars = c(**"vs"**, **"am"**)

),

col\_packs(

enough\_col\_sum = . **%>%**

summarise\_if(is\_integerish, rules(is\_enough = sum(.) >= 14))

),

row\_packs(

enough\_row\_sum = . **%>%**

filter(vs == 1) **%>%**

transmute(is\_enough = rowSums(.) >= 200)

),

cell\_packs(

dbl\_not\_outlier = . **%>%**

transmute\_if(is.numeric, rules(is\_not\_out = z\_score(.) < 1)) **%>%**

slice(-(1:5))

)

)

# Expose data to rules

mtcars\_exposed **<-** mtcars **%>%** as\_tibble() **%>%**

expose(my\_packs)

# View exposure

mtcars\_exposed **%>%** get\_exposure()

#> Exposure

#>

#> Packs info:

#> # A tibble: 5 x 4

#> name type fun remove\_obeyers

#> <chr> <chr> <list> <lgl>

#> 1 dims data\_pack <data\_pck> TRUE

#> 2 vs\_am\_num group\_pack <grop\_pck> TRUE

#> 3 enough\_col\_sum col\_pack <col\_pack> TRUE

#> 4 enough\_row\_sum row\_pack <row\_pack> TRUE

#> 5 dbl\_not\_outlier cell\_pack <cell\_pck> TRUE

#>

#> Tidy data validation report:

#> # A tibble: 117 x 5

#> pack rule var id value

#> <chr> <chr> <chr> <int> <lgl>

#> 1 dims nrow\_high .all 0 FALSE

#> 2 dims ncol\_low .all 0 FALSE

#> 3 vs\_am\_num vs\_am\_low 0.1 0 FALSE

#> 4 enough\_col\_sum is\_enough am 0 FALSE

#> 5 enough\_row\_sum is\_enough .all 19 FALSE

#> 6 dbl\_not\_outlier is\_not\_out mpg 15 FALSE

#> # … with 111 more rows

# Assert any breaker

invisible(mtcars\_exposed **%>%** assert\_any\_breaker())

#> Breakers report

#> Tidy data validation report:

#> # A tibble: 117 x 5

#> pack rule var id value

#> <chr> <chr> <chr> <int> <lgl>

#> 1 dims nrow\_high .all 0 FALSE

#> 2 dims ncol\_low .all 0 FALSE

#> 3 vs\_am\_num vs\_am\_low 0.1 0 FALSE

#> 4 enough\_col\_sum is\_enough am 0 FALSE

#> 5 enough\_row\_sum is\_enough .all 19 FALSE

#> 6 dbl\_not\_outlier is\_not\_out mpg 15 FALSE

#> # … with 111 more rows

#> Error: assert\_any\_breaker: Some breakers found in exposure.

## Overview

**Rule** is a function which converts data unit of interest (data, group, column, row, cell) to logical value indicating whether this object satisfies certain condition.

**Rule pack** is a function which combines several rules into one functional block. The recommended way of creating rules is by creating packs right away with the use of dplyr and magrittr’s pipe operator.

**Exposing** data to rules means applying rules to data, collecting results in common format and attaching them to the data as an exposure attribute. In this way actual exposure can be done in multiple steps and also be a part of a general data preparation pipeline.

**Exposure** is a format designed to contain uniform information about validation of different data units. For reproducibility it also saves information about applied packs. Basically exposure is a list with two elements:

1. **Packs info**: a tibble with the following structure:
   * name <chr> : Name of the pack. If not set manually it will be imputed during exposure.
   * type <chr> : Name of pack type. Indicates which data unit pack checks.
   * fun <list> : List of rule pack functions.
   * remove\_obeyers <lgl> : Whether rows about obeyers (data units that obey certain rule) were removed from report after applying pack.
2. **Tidy data validation report**: a tibble with the following structure:
   * pack <chr> : Name of rule pack from column ‘name’ in packs info.
   * rule <chr> : Name of the rule defined in rule pack.
   * var <chr> : Name of the variable which validation result is reported. Value ‘.all’ is reserved and interpreted as ‘all columns as a whole’. **Note** that var doesn’t always represent the actual column in data frame: for group packs it represents the created group name.
   * id <int> : Index of the row in tested data frame which validation result is reported. Value 0 is reserved and interpreted as ‘all rows as a whole’.
   * value <lgl> : Whether the described data unit obeys the rule.

There are four basic combinations of var and id values which define five basic data units:

* var == '.all' and id == 0: Data as a whole.
* var != '.all' and id == 0: Group (var shouldn’t be an actual column name) or column (var should be an actual column name) as a whole.
* var == '.all' and id != 0: Row as a whole.
* var != '.all' and id != 0: Described cell.

With exposure attached to data one can perform different kinds of actions: exploration, assertion, imputation and so on.

## Usage

### Creating packs

#### **Data packs**

# List of two rule packs for checking data properties

my\_data\_packs **<-** data\_packs(

# data\_dims is a pack name

data\_dims = . **%>%** summarise(

# ncol and nrow are rule names

ncol = ncol(.) == 12,

nrow = nrow(.) == 32

),

# Data after subsetting should have number of rows in between 10 and 30

# Rules are applied separately

vs\_1 = . **%>%** filter(vs == 1) **%>%**

summarise(

nrow\_low = nrow(.) > 10,

nrow\_high = nrow(.) < 30

)

)

#### **Group packs**

# List of one nameless rule pack for checking group property

my\_group\_packs **<-** group\_packs(

# Name will be imputed during exposure

. **%>%** group\_by(vs, am) **%>%**

summarise(any\_cyl\_6 = any(cyl == 6)),

# One should supply grouping variables for correct interpretation of output

.group\_vars = c(**"vs"**, **"am"**)

)

#### **Column packs**

# rules() is a dplyr::funs() with necessary name imputations

# In column packs it should always be used instead of dplyr::funs()

# List of two rule pack for checking certain columns' properties

my\_col\_packs **<-** col\_packs(

sum\_bounds = . **%>%** summarise\_at(

# Check only columns with names starting with 'c'

vars(starts\_with(**"c"**)),

rules(sum\_low = sum(.) > 300, sum\_high = sum(.) < 400)

),

# In the edge case of checking one column with one rule there is a need

# for forcing inclusion of names in the output of summarise\_at().

# This is done with naming argument in vars()

vs\_mean = . **%>%** summarise\_at(vars(vs = vs), rules(mean(.) > 0.5))

)

#### **Row packs**

z\_score **<- function**(x) {

(x - mean(x)) / sd(x)

}

# List of one rule pack checking certain rows' property

my\_row\_packs **<-** row\_packs(

row\_mean = . **%>%** mutate(rowMean = rowMeans(.)) **%>%**

transmute(is\_common\_row\_mean = abs(z\_score(rowMean)) < 1) **%>%**

# Check only rows 10-15

# Values in 'id' column of report will be based on input data (i.e. 10-15)

# and not on output data (1-6)

slice(10:15)

)

#### **Cell packs**

is\_integerish **<- function**(x) {

all(x == as.integer(x))

}

# List of two cell pack checking certain cells' property

my\_cell\_packs **<-** cell\_packs(

my\_cell\_pack\_1 = . **%>%** transmute\_if(

# Check only integer-like columns

is\_integerish,

rules(is\_common = abs(z\_score(.)) < 1)

) **%>%**

# Check only rows 20-30

slice(20:30),

# The same edge case as in column rule pack

vs\_side = . **%>%** transmute\_at(vars(vs = **"vs"**), rules(. > mean(.)))

)

### Exposing

By default exposing removes obeyers.

mtcars **%>%**

expose(my\_data\_packs, my\_group\_packs) **%>%**

get\_exposure()

#> Exposure

#>

#> Packs info:

#> # A tibble: 3 x 4

#> name type fun remove\_obeyers

#> <chr> <chr> <list> <lgl>

#> 1 data\_dims data\_pack <data\_pck> TRUE

#> 2 vs\_1 data\_pack <data\_pck> TRUE

#> 3 group\_pack\_\_1 group\_pack <grop\_pck> TRUE

#>

#> Tidy data validation report:

#> # A tibble: 3 x 5

#> pack rule var id value

#> <chr> <chr> <chr> <int> <lgl>

#> 1 data\_dims ncol .all 0 FALSE

#> 2 group\_pack\_\_1 any\_cyl\_6 0.0 0 FALSE

#> 3 group\_pack\_\_1 any\_cyl\_6 1.1 0 FALSE

One can leave obeyers by setting .remove\_obeyers to FALSE.

mtcars **%>%**

expose(my\_data\_packs, my\_group\_packs, .remove\_obeyers = **FALSE**) **%>%**

get\_exposure()

#> Exposure

#>

#> Packs info:

#> # A tibble: 3 x 4

#> name type fun remove\_obeyers

#> <chr> <chr> <list> <lgl>

#> 1 data\_dims data\_pack <data\_pck> FALSE

#> 2 vs\_1 data\_pack <data\_pck> FALSE

#> 3 group\_pack\_\_1 group\_pack <grop\_pck> FALSE

#>

#> Tidy data validation report:

#> # A tibble: 8 x 5

#> pack rule var id value

#> <chr> <chr> <chr> <int> <lgl>

#> 1 data\_dims ncol .all 0 FALSE

#> 2 data\_dims nrow .all 0 TRUE

#> 3 vs\_1 nrow\_low .all 0 TRUE

#> 4 vs\_1 nrow\_high .all 0 TRUE

#> 5 group\_pack\_\_1 any\_cyl\_6 0.0 0 FALSE

#> 6 group\_pack\_\_1 any\_cyl\_6 0.1 0 TRUE

#> # … with 2 more rows

By default expose() guesses the pack type if ‘not-pack’ function is supplied. This behaviour has some edge cases but is useful for interactive use.

mtcars **%>%**

expose(

some\_data\_pack = . **%>%** summarise(nrow = nrow(.) == 10),

some\_col\_pack = . **%>%** summarise\_at(vars(vs = **"vs"**), rules(is.character(.)))

) **%>%**

get\_exposure()

#> Exposure

#>

#> Packs info:

#> # A tibble: 2 x 4

#> name type fun remove\_obeyers

#> <chr> <chr> <list> <lgl>

#> 1 some\_data\_pack data\_pack <data\_pck> TRUE

#> 2 some\_col\_pack col\_pack <col\_pack> TRUE

#>

#> Tidy data validation report:

#> # A tibble: 2 x 5

#> pack rule var id value

#> <chr> <chr> <chr> <int> <lgl>

#> 1 some\_data\_pack nrow .all 0 FALSE

#> 2 some\_col\_pack rule\_\_1 vs 0 FALSE

To write strict and robust code one can set .guess to FALSE.

mtcars **%>%**

expose(

some\_data\_pack = . **%>%** summarise(nrow = nrow(.) == 10),

some\_col\_pack = . **%>%** summarise\_at(vars(vs = **"vs"**), rules(is.character(.))),

.guess = **FALSE**

) **%>%**

get\_exposure()

#> Error in expose\_single.default(X[[i]], ...): There is unsupported class of rule pack.

### Acting after exposure

General actions are recommended to be done with act\_after\_exposure(). It takes two arguments:

* .trigger - a function which takes the data with attached exposure and returns TRUE if some action should be made.
* .actor - a function which takes the same argument as .trigger and performs some action.

If trigger didn’t notify then the input data is returned untouched. Otherwise the output of .actor() is returned. **Note** that act\_after\_exposure() is often used for creating side effects (printing, throwing error etc.) and in that case should invisibly return its input (to be able to use it with pipe).

trigger\_one\_pack **<- function**(.tbl) {

packs\_number **<-** .tbl **%>%**

get\_packs\_info() **%>%**

nrow()

packs\_number > 1

}

actor\_one\_pack **<- function**(.tbl) {

cat(**"More than one pack was applied.\n"**)

invisible(.tbl)

}

mtcars **%>%**

expose(my\_col\_packs, my\_row\_packs) **%>%**

act\_after\_exposure(

.trigger = trigger\_one\_pack,

.actor = actor\_one\_pack

) **%>%**

invisible()

#> More than one pack was applied.

ruler has function assert\_any\_breaker() which can notify about presence of any breaker in exposure.

mtcars **%>%**

expose(my\_col\_packs, my\_row\_packs) **%>%**

assert\_any\_breaker()

#> Breakers report

#> Tidy data validation report:

#> # A tibble: 4 x 5

#> pack rule var id value

#> <chr> <chr> <chr> <int> <lgl>

#> 1 sum\_bounds sum\_low cyl 0 FALSE

#> 2 sum\_bounds sum\_low carb 0 FALSE

#> 3 vs\_mean rule\_\_1 vs 0 FALSE

#> 4 row\_mean is\_common\_row\_mean .all 15 FALSE

#> Error: assert\_any\_breaker: Some breakers found in exposure.

Keyholder Package Details

keyholder is a package for storing information (*keys*) about rows of data frame like objects. The common use cases are to track rows of data without modifying it and to backup and restore information about rows. This is done with creating a class **keyed\_df** which has special attribute “keys”. Keys are updated according to changes in rows of reference data frame.

keyholder is designed to work tightly with dplyr package. All its one- and two-table verbs update keys properly.

Installation

You can install current stable version from CRAN with:

[install.packages](https://rdrr.io/r/utils/install.packages.html)(**"keyholder"**)

Usage

keyholder provides a set of functions to work with keys:

* Set keys with assign\_keys() and key\_by().
* Get all keys with keys(). Get one specific key with pull\_key().
* Restore information stored in certain keys with restore\_keys() and its scoped variants (\*\_all(), \*\_if() and \*\_at()).
* Rename certain keys with rename\_keys() and its scoped variants.
* Remove certain keys with remove\_keys() and its scoped variants. Completely unkey object with unkey().
* Track rows with use\_id() and special .id key.

For more detailed explanations and examples see package vignettes and documentation.

Common use cases

[**library**](https://rdrr.io/r/base/library.html)([dplyr](https://dplyr.tidyverse.org/))

[**library**](https://rdrr.io/r/base/library.html)([keyholder](https://echasnovski.github.io/keyholder/))

mtcars\_tbl **<-** mtcars [**%>%**](https://magrittr.tidyverse.org/reference/pipe.html) [as\_tibble](https://tibble.tidyverse.org/reference/as_tibble.html)()

* Track rows without modifying data:

mtcars\_tbl\_id **<-** mtcars\_tbl [**%>%**](https://magrittr.tidyverse.org/reference/pipe.html)

# Creates a key '.id' with row index

[use\_id](https://echasnovski.github.io/keyholder/reference/keyholder-id.html)() [**%>%**](https://magrittr.tidyverse.org/reference/pipe.html)

[filter](https://dplyr.tidyverse.org/reference/filter.html)(vs == 1, gear == 4)

mtcars\_tbl\_id

#> # A keyed object. Keys: .id

#> # A tibble: 10 × 11

#> mpg cyl disp hp drat wt qsec vs am gear carb

#> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>

#> 1 22.8 4 108 93 3.85 2.32 18.6 1 1 4 1

#> 2 24.4 4 147. 62 3.69 3.19 20 1 0 4 2

#> 3 22.8 4 141. 95 3.92 3.15 22.9 1 0 4 2

#> # … with 7 more rows

mtcars\_tbl\_id [**%>%**](https://magrittr.tidyverse.org/reference/pipe.html) [pull\_key](https://echasnovski.github.io/keyholder/reference/keys-manipulate.html)(.id)

#> [1] 3 8 9 10 11 18 19 20 26 32

* Backup and restore information:

mtcars\_tbl\_keyed **<-** mtcars\_tbl [**%>%**](https://magrittr.tidyverse.org/reference/pipe.html)

# Backup

[key\_by](https://echasnovski.github.io/keyholder/reference/keys-set.html)(vs, am, gear) [**%>%**](https://magrittr.tidyverse.org/reference/pipe.html)

# Modify

[mutate](https://dplyr.tidyverse.org/reference/mutate.html)(vs = am) [**%>%**](https://magrittr.tidyverse.org/reference/pipe.html)

[group\_by](https://dplyr.tidyverse.org/reference/group_by.html)(vs) [**%>%**](https://magrittr.tidyverse.org/reference/pipe.html)

[mutate](https://dplyr.tidyverse.org/reference/mutate.html)(gear = [max](https://rdrr.io/r/base/Extremes.html)(gear))

# Restore with recomputing groups

mtcars\_tbl\_restored **<-** mtcars\_tbl\_keyed [**%>%**](https://magrittr.tidyverse.org/reference/pipe.html) [restore\_keys\_all](https://echasnovski.github.io/keyholder/reference/restore-keys-scoped.html)()

mtcars\_tbl\_grouped **<-** mtcars\_tbl [**%>%**](https://magrittr.tidyverse.org/reference/pipe.html) [group\_by](https://dplyr.tidyverse.org/reference/group_by.html)(vs)

[all.equal](https://rdrr.io/r/base/all.equal.html)(

[as.data.frame](https://rdrr.io/r/base/as.data.frame.html)(mtcars\_tbl\_restored),

[as.data.frame](https://rdrr.io/r/base/as.data.frame.html)(mtcars\_tbl\_grouped),

check.attributes = **FALSE**

)

#> [1] TRUE

[all.equal](https://rdrr.io/r/base/all.equal.html)(

[group\_indices](https://dplyr.tidyverse.org/reference/group_data.html)(mtcars\_tbl\_restored),

[group\_indices](https://dplyr.tidyverse.org/reference/group_data.html)(mtcars\_tbl\_grouped)

)

#> [1] TRUE

# Restore with renaming

mtcars\_tbl\_keyed [**%>%**](https://magrittr.tidyverse.org/reference/pipe.html)

[restore\_keys\_at](https://echasnovski.github.io/keyholder/reference/restore-keys-scoped.html)(**"vs"**, .funs = [list](https://rdrr.io/r/base/list.html)(~ [paste0](https://rdrr.io/r/base/paste.html)(., **"\_old"**)))

#> # A keyed object. Keys: vs, am, gear

#> # A tibble: 32 × 12

#> # Groups: vs [2]

#> mpg cyl disp hp drat wt qsec vs am gear carb vs\_old

#> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>

#> 1 21 6 160 110 3.9 2.62 16.5 1 1 5 4 0

#> 2 21 6 160 110 3.9 2.88 17.0 1 1 5 4 0

#> 3 22.8 4 108 93 3.85 2.32 18.6 1 1 5 1 1

#> # … with 29 more rows

* As a special case of previous usage one can also hide columns for convenient use of dplyr’s \*\_if scoped variants of verbs:

# Restored key goes to the end of the tibble

mtcars\_tbl [**%>%**](https://magrittr.tidyverse.org/reference/pipe.html)

[key\_by](https://echasnovski.github.io/keyholder/reference/keys-set.html)(mpg, .exclude = **TRUE**) [**%>%**](https://magrittr.tidyverse.org/reference/pipe.html)

[mutate\_if](https://dplyr.tidyverse.org/reference/mutate_all.html)(is.numeric, round, digits = 0) [**%>%**](https://magrittr.tidyverse.org/reference/pipe.html)

[restore\_keys\_all](https://echasnovski.github.io/keyholder/reference/restore-keys-scoped.html)()

#> # A keyed object. Keys: mpg

#> # A tibble: 32 × 11

#> cyl disp hp drat wt qsec vs am gear carb mpg

#> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>

#> 1 6 160 110 4 3 16 0 1 4 4 21

#> 2 6 160 110 4 3 17 0 1 4 4 21

#> 3 4 108 93 4 2 19 1 1 4 1 22.8

#> # … with 29 more rows

Harry Potter and Competiotion results with survey data

We will need the following setup:

**library**(dplyr)

**library**(tidyr)

**library**(rlang)

**library**(stringr)

**library**(ggplot2)

**library**(comperes)

set.seed(201805)

theme\_set(theme\_bw())

# Authenticity palette

hp\_pal **<-** c(Gryff = **"#D02037"**, Huffl = **"#F0C346"**,

Raven = **"#2450A8"**, Raven\_light = **"#0088FF"**,

Slyth = **"#09774A"**)

# For less noisy bar charts

theme\_bar **<- function**() {

list(theme(panel.grid.major.x = element\_blank(),

panel.grid.minor.x = element\_blank()))

}

# **Exploration**

## Data preparation

hp\_suvery is a tibble (enhanced data frame) and has the following columns:

* **person** <int> : Identifier of a person.
* **book** <chr> : Identifier of a Harry Potter book. Its values are of the form “HP\_x” where “x” represents book’s number in the series (from 1 to 7).
* **score** <chr> : Book’s score. Can be one of “1 - Poor”, “2 - Fair”, “3 - Good”, “4 - Very Good”, “5 - Excellent”.

For exploration, let’s transform hp\_survey for more expressive code and results:

* Convert scores to numerical.
* Add book names.

book\_names **<-** c(

**"Philosopher's (Sorcerer's) Stone (#1)"**,

**"Chamber of Secrets (#2)"**,

**"Prisoner of Azkaban (#3)"**,

**"Goblet of Fire (#4)"**,

**"Order of the Phoenix (#5)"**,

**"Half-Blood Prince (#6)"**,

**"Deathly Hallows (#7)"**

)

book\_name\_tbl **<-** tibble(

book = paste0(**"HP\_"**, 1:7),

book\_name = factor(book\_names, levels = book\_names)

)

hp **<-** hp\_survey **%>%**

# Extract numerical score

rename(score\_chr = score) **%>%**

mutate(score = as.integer(gsub(**"[^0-9].\*$"**, **""**, score\_chr))) **%>%**

# Add book names

left\_join(y = book\_name\_tbl, by = **"book"**)

hp

## # A tibble: 657 x 5

## person book score\_chr score book\_name

## <int> <chr> <chr> <int> <fct>

## 1 1 HP\_6 5 - Excellent 5 Half-Blood Prince (#6)

## 2 1 HP\_7 5 - Excellent 5 Deathly Hallows (#7)

## 3 2 HP\_1 3 - Good 3 Philosopher's (Sorcerer's) Stone (#1)

## 4 2 HP\_4 5 - Excellent 5 Goblet of Fire (#4)

## 5 2 HP\_5 2 - Fair 2 Order of the Phoenix (#5)

## # ... with 652 more rows

## Subset uniformity

The first step in the survey was to choose the first element in the randomly shuffled list to simulate generation of random subset from all books. Each of 127 list element was connected to one subset. Lets visualize subset frequency to ensure a good faith of respondents:

# Compute subset representations

hp\_subsets **<-** hp **%>%**

arrange(person, book) **%>%**

group\_by(person) **%>%**

summarise(subset = paste0(book, collapse = **"-"**))

# Compute the number of actually picked subsets

n\_distinct(hp\_subsets$subset)

## [1] 95

# Visualize

hp\_subsets **%>%**

ggplot(aes(subset)) +

geom\_bar(fill = hp\_pal[**"Gryff"**]) +

labs(

x = **"Subset"**, y = **"Number of times subset was picked"**,

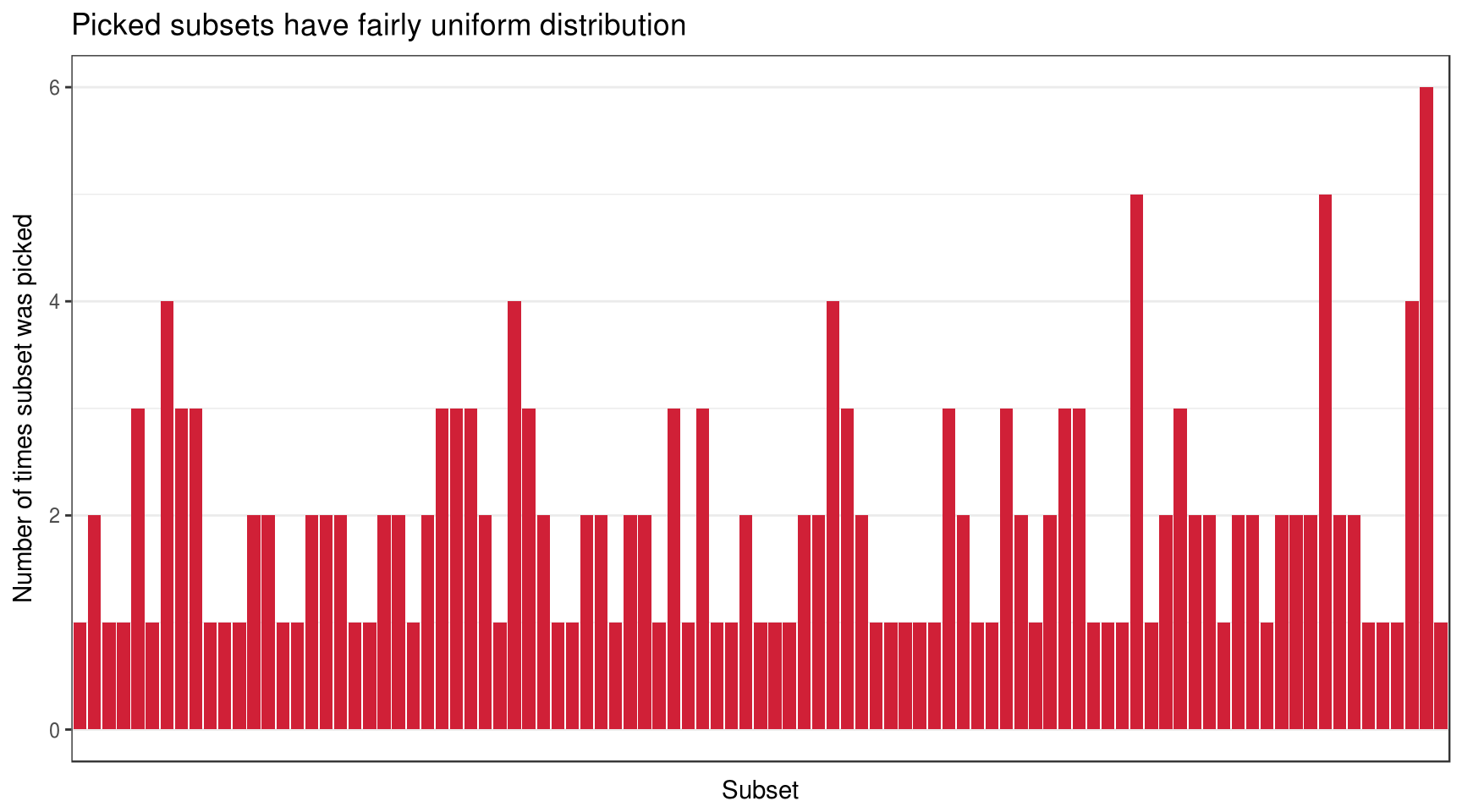
title = **"Picked subsets have fairly uniform distribution"**

) +

scale\_x\_discrete(labels = **NULL**) +

theme\_bar() +

theme(axis.ticks.x = element\_blank())



So there are 95 subsets actually picked and their distribution seems reasonably uniform. This is enough for me to confirm that randomization for subsets was successful.

## Book presence

Other important thing to explore is number of times book was actually rated:

hp **%>%**

ggplot(aes(book\_name)) +

geom\_bar(fill = hp\_pal[**"Huffl"**]) +

# Cool way to wrap labels for a given width

scale\_x\_discrete(labels = **function**(x) str\_wrap(x, width = 15)) +

labs(

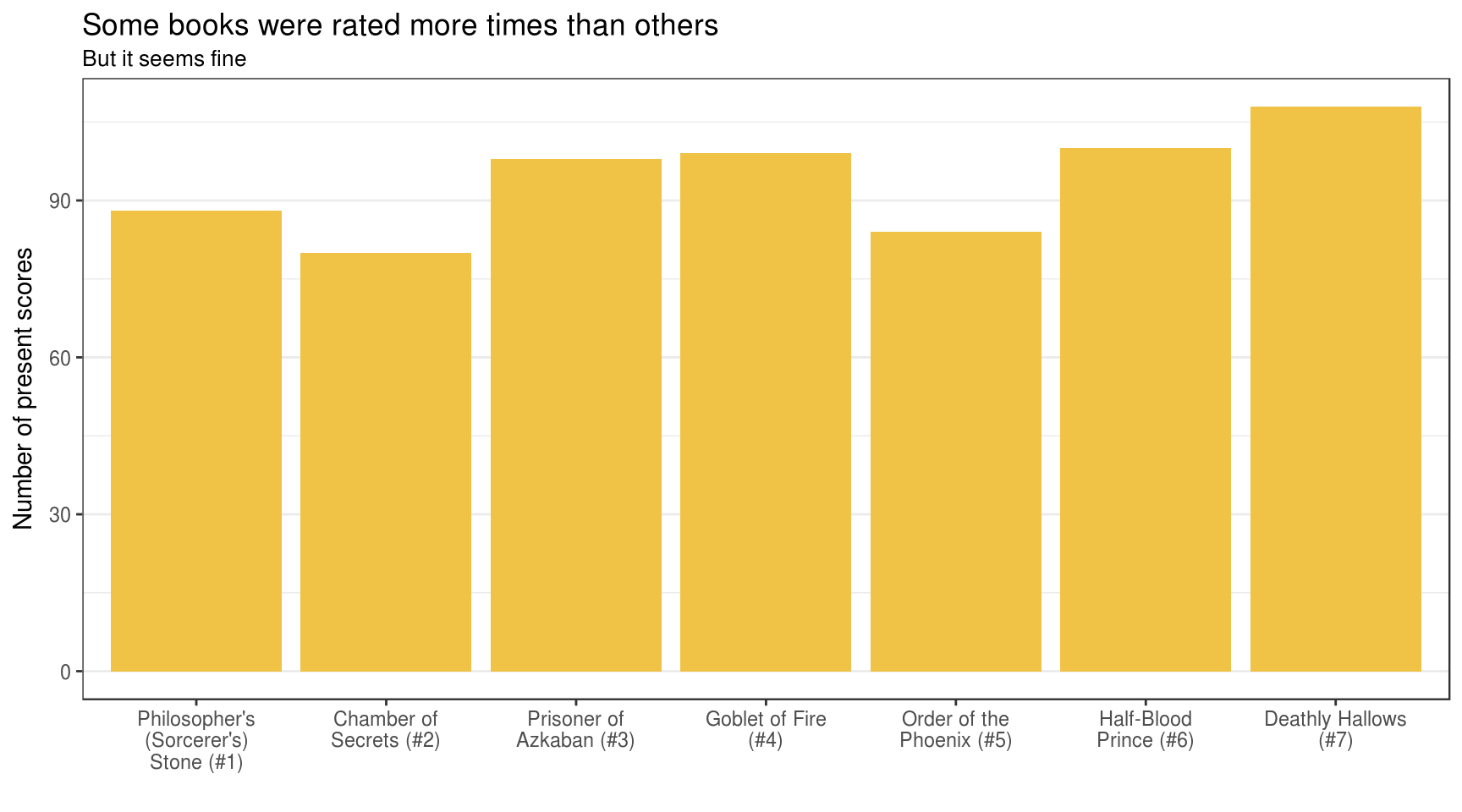
x = **""**, y = **"Number of present scores"**,

title = **"Some books were rated more times than others"**,

subtitle = **"But it seems fine"**

) +

theme\_bar()



## Book scores

The most obvious way to summarise book “performance” is its mean score of numerical representation of scale. Using mean is not harmful in this study as no outlier can be present.

hp\_book\_score **<-** hp **%>%**

group\_by(book\_name) **%>%**

summarise(mean\_score = round(mean(score), digits = 2)) **%>%**

arrange(desc(mean\_score))

hp\_book\_score

## # A tibble: 7 x 2

## book\_name mean\_score

## <fct> <dbl>

## 1 Prisoner of Azkaban (#3) 4.19

## 2 Half-Blood Prince (#6) 4.13

## 3 Goblet of Fire (#4) 4.00

## 4 Deathly Hallows (#7) 3.96

## 5 Philosopher's (Sorcerer's) Stone (#1) 3.91

## 6 Order of the Phoenix (#5) 3.90

## 7 Chamber of Secrets (#2) 3.55

**So, “the best” book seems to be “Harry Potter and the Prisoner of Azkaban (#3)”**.

For more understanding of results, lets also visualize score distribution.

hp **%>%**

# Compute share of score per book

count(book\_name, score) **%>%**

group\_by(book\_name) **%>%**

mutate(share = n / sum(n)) **%>%**

ungroup() **%>%**

# Visualize

ggplot() +

geom\_col(

aes(score, share, colour = score, fill = score),

show.legend = **FALSE**

) +

geom\_text(

data = hp\_book\_score,

mapping = aes(label = paste0(**"Mean = "**, mean\_score)),

x = -**Inf**, y = **Inf**,

hjust = -0.05, vjust = 1.3

) +

facet\_wrap(~ book\_name) +

scale\_x\_continuous(

breaks = 1:5,

labels = c(**"1\nPoor"**, **"2\nFair"**, **"3\nGood"**,

**"4\nVery\nGood"**, **"5\nExcellent"**)

) +

scale\_fill\_gradient(low = hp\_pal[**"Raven"**], high = hp\_pal[**"Raven\_light"**]) +

scale\_colour\_gradient(low = hp\_pal[**"Raven"**], high = hp\_pal[**"Raven\_light"**]) +

labs(

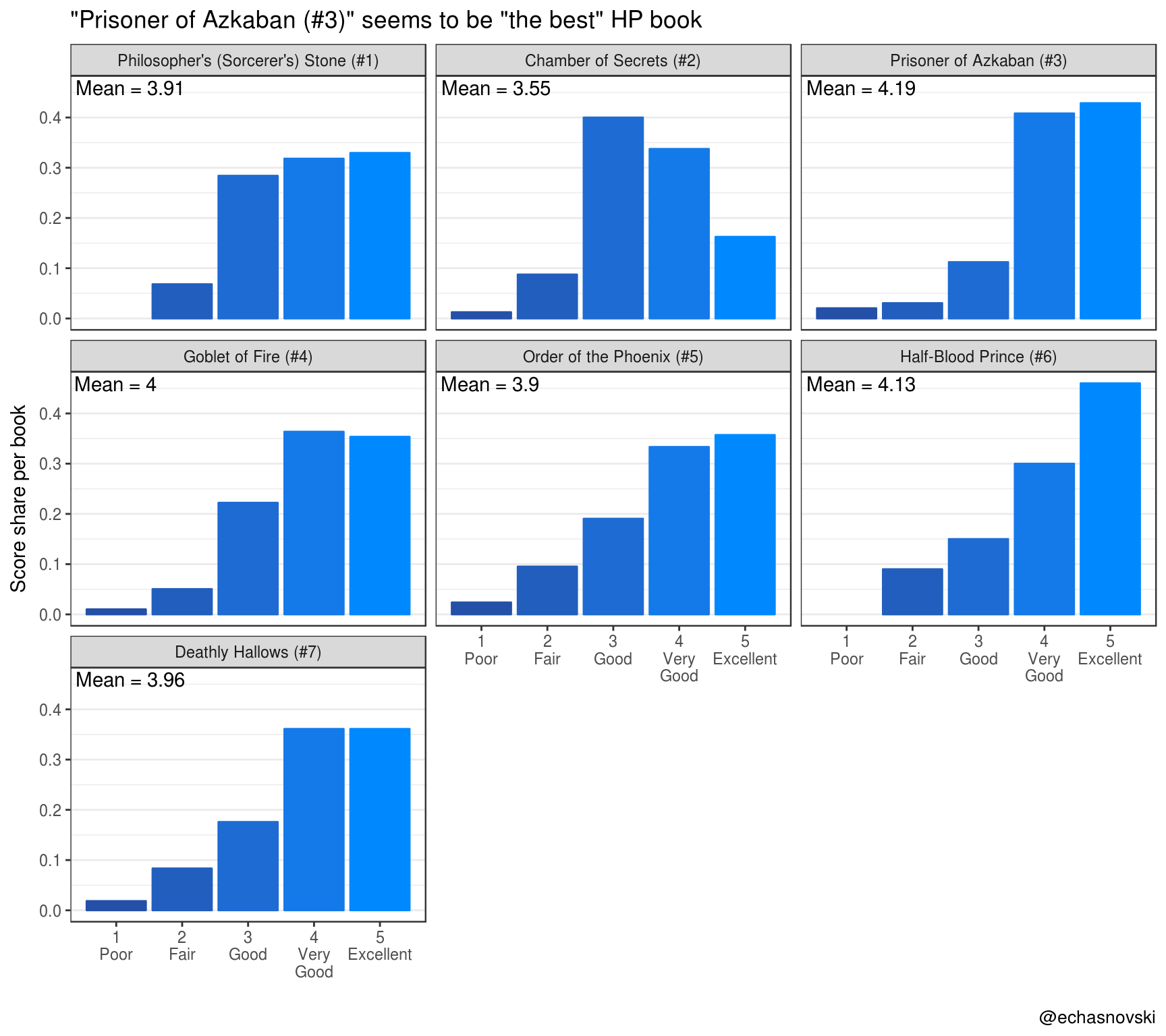
x = **""**, y = **"Score share per book"**,

title = **'"Prisoner of Azkaban (#3)" seems to be "the best" HP book'**,

caption = **"@echasnovski"**

) +

theme\_bar()



# **Competition results**

## Formats of comperes

Understanding of **competition** is quite general: it is a set of **games** (abstract event) in which **players** (abstract entity) gain some abstract **scores** (typically numeric). Inside games all players are treated equally. The most natural example is sport results, however not the only one. For example, product rating can be considered as a competition between products as “players”. Here a “game” is a customer that reviews a set of products by rating them with numerical “score” (stars, points, etc.).

In case of Harry Potter Books Survey results “game” is an act of respondent taking part in survey, “player” - Harry Potter book, “score” - discrete scale values converted to numerical score from 1 to 5.

In comperes there are two supported formats of competition results:

* **Long format**. It is the most abstract way of presenting competition results. Basically, it is a data frame (or tibble) with columns game (game identifier), player (player identifier) and score where each row represents the score of particular player in particular game. One game can consist from **variable** number of players which makes this format more usable. Extra columns are allowed.
* **Wide format** is a more convenient way to store results with **fixed** number of players in a game. Each row represents scores of all players in particular game. Data should be organized in pairs of columns “player”-“score”. Identifier of a pair should go after respective keyword and consist only from digits. For example: player1, score1, player2, score2. Order doesn’t matter. Column game is optional. Extra columns are also allowed.

Programmatically these formats are implemented as S3 classes longcr and widecr respectively. Essentially, they are tibbles with fixed structure. Objects of these classes should be created using functions as\_longcr() and as\_widecr() which also do conversions to other format.

## Conversion

hp\_survey presents results in **long format**.

hp\_cr **<-** hp\_survey **%>%**

transmute(

game = person, player = book,

score = as.integer(gsub(**"[^0-9].\*$"**, **""**, score))

) **%>%**

as\_longcr()

hp\_cr

## # A longcr object:

## # A tibble: 657 x 3

## game player score

## <int> <chr> <int>

## 1 1 HP\_6 5

## 2 1 HP\_7 5

## 3 2 HP\_1 3

## 4 2 HP\_4 5

## 5 2 HP\_5 2

## # ... with 652 more rows

Here is the demonstration of conversion to **wide format**. It detects the maximum number of players in a game, which is 7, and assumes that data is missing in games with less number of players.

as\_widecr(hp\_cr)

## # A widecr object:

## # A tibble: 182 x 15

## game player1 score1 player2 score2 player3 score3 player4 score4

## <int> <chr> <int> <chr> <int> <chr> <int> <chr> <int>

## 1 1 HP\_6 5 HP\_7 5 <NA> NA <NA> NA

## 2 2 HP\_1 3 HP\_4 5 HP\_5 2 HP\_6 4

## 3 3 HP\_1 3 HP\_3 4 HP\_5 1 <NA> NA

## 4 4 HP\_6 5 HP\_7 5 <NA> NA <NA> NA

## 5 5 HP\_4 4 HP\_5 3 <NA> NA <NA> NA

## # ... with 177 more rows, and 6 more variables: player5 <chr>,

## # score5 <int>, player6 <chr>, score6 <int>, player7 <chr>, score7 <int>

# **Head-to-Head**

## Functionality of comperes

Head-to-Head value is a **summary statistic of direct confrontation between two players**. It is assumed that this value can be computed based only on the players’ matchups (results for ordered pairs of players from one game). In other words, every game is converted into series of “subgames” between ordered pairs of players (including selfplay) which is stored as widecr object. After that, summary of item, defined by columns player1 and player2, is computed.

comperes has function get\_matchups() for computing matchups:

get\_matchups(hp\_cr)

## # A widecr object:

## # A tibble: 2,697 x 5

## game player1 score1 player2 score2

## <int> <chr> <int> <chr> <int>

## 1 1 HP\_6 5 HP\_6 5

## 2 1 HP\_6 5 HP\_7 5

## 3 1 HP\_7 5 HP\_6 5

## 4 1 HP\_7 5 HP\_7 5

## 5 2 HP\_1 3 HP\_1 3

## # ... with 2,692 more rows

To compute multiple Head-to-Head values, use h2h\_long() supplying competition results and summarizing expressions in dplyr::summarise() fashion. They will be applied to a data frame of matchups.

hp\_cr\_h2h **<-** hp\_cr **%>%** h2h\_long(

# Number of macthups

n = n(),

# Number of wins plus half the number of ties

# num\_wins() is a function from comperes to compute number of times

# first score is bigger than second one

num\_wins = num\_wins(score1, score2, half\_for\_draw = **TRUE**),

# Mean rating of a book scored in matchups with other books

mean\_score = mean(score1),

# Mean rating difference of books scored in direct matchups

mean\_score\_diff = mean(score1 - score2)

) **%>%**

mutate\_if(is.numeric, funs(round(., 2)))

hp\_cr\_h2h

## # A long format of Head-to-Head values:

## # A tibble: 49 x 6

## player1 player2 n num\_wins mean\_score mean\_score\_diff

## <chr> <chr> <dbl> <dbl> <dbl> <dbl>

## 1 HP\_1 HP\_1 88. 44.0 3.91 0.

## 2 HP\_1 HP\_2 42. 29.5 3.88 0.500

## 3 HP\_1 HP\_3 51. 19.5 3.92 -0.390

## 4 HP\_1 HP\_4 48. 24.0 3.79 0.0400

## 5 HP\_1 HP\_5 42. 21.5 3.79 0.

## # ... with 44 more rows

So here we see, for example, that HP\_1 and HP\_2 had 42 matchups, i.e. they were rated by the same person 42 times. HP\_1 “won” 29.5 (respecting ties) times, gained mean score of 3.88 in those matchups and had, on average, 0.5 points more.

There is also an h2h\_mat() function which computes a matrix of Head-to-Head values for one expression.

hp\_cr **%>%** h2h\_mat(num\_wins(score1, score2, half\_for\_draw = **TRUE**))

## # A matrix format of Head-to-Head values:

## HP\_1 HP\_2 HP\_3 HP\_4 HP\_5 HP\_6 HP\_7

## HP\_1 44.0 29.5 19.5 24.0 21.5 17.0 24.0

## HP\_2 12.5 40.0 12.0 11.5 10.5 12.0 19.0

## HP\_3 31.5 32.0 49.0 31.5 28.0 25.0 33.5

## HP\_4 24.0 33.5 26.5 49.5 23.5 30.5 31.5

## HP\_5 20.5 25.5 15.0 24.5 42.0 23.0 24.5

## HP\_6 25.0 30.0 20.0 27.5 24.0 50.0 34.0

## HP\_7 26.0 34.0 21.5 29.5 25.5 26.0 54.0

For more convenient usage, comperes has a list h2h\_funs of some common Head-to-Head functions stored as expressions. To use them you need a little bit of rlang’s unquoting magic.

h2h\_funs[1:3]

## $mean\_score\_diff

## mean(score1 - score2)

##

## $mean\_score\_diff\_pos

## max(mean(score1 - score2), 0)

##

## $mean\_score

## mean(score1)

hp\_cr **%>%** h2h\_long(!!! h2h\_funs)

## # A long format of Head-to-Head values:

## # A tibble: 49 x 11

## player1 player2 mean\_score\_diff mean\_score\_diff\_pos mean\_score

## <chr> <chr> <dbl> <dbl> <dbl>

## 1 HP\_1 HP\_1 0. 0. 3.91

## 2 HP\_1 HP\_2 0.500 0.500 3.88

## 3 HP\_1 HP\_3 -0.392 0. 3.92

## 4 HP\_1 HP\_4 0.0417 0.0417 3.79

## 5 HP\_1 HP\_5 0. 0. 3.79

## # ... with 44 more rows, and 6 more variables: sum\_score\_diff <int>,

## # sum\_score\_diff\_pos <dbl>, sum\_score <int>, num\_wins <dbl>,

## # num\_wins2 <dbl>, num <int>

## Harry Potter books

Head-to-Head “performance” of Harry Potter books is summarised in the following plot:

hp\_cr\_h2h **%>%**

gather(h2h\_fun, value, -player1, -player2) **%>%**

# Manually produce a dummy colour variable to use in facets

group\_by(h2h\_fun) **%>%**

mutate(col = (value - min(value)) / (max(value) - min(value))) **%>%**

ungroup() **%>%**

# Make factors for correct orders

mutate(

player1 = factor(player1, levels = rev(sort(unique(player1)))),

player2 = factor(player2, levels = sort(unique(player2))),

h2h\_fun = factor(h2h\_fun,

levels = c(**"n"**, **"num\_wins"**,

**"mean\_score"**, **"mean\_score\_diff"**)),

h2h\_fun = recode(

h2h\_fun,

n = **"Number of matchups (ratings by common person)"**,

num\_wins = **'Number of "wins" in matchups (half for ties)'**,

mean\_score = **"Mean score in matchups"**,

mean\_score\_diff = **"Mean score difference in matchups"**

)

) **%>%**

# Visualize

ggplot(aes(player1, player2)) +

geom\_text(

aes(label = value, colour = col),

size = 5, fontface = **"bold"**, show.legend = **FALSE**

) +

facet\_wrap(~ h2h\_fun, scales = **"free"**) +

# To coordinate well with matrix form of Head-to-Head results

coord\_flip() +

scale\_colour\_gradient(low = hp\_pal[**"Slyth"**], high = hp\_pal[**"Gryff"**]) +

labs(

x = **""**, y = **""**,

title = **"Head-to-Head performance of Harry Potter books"**,

subtitle = paste0(

**'"HP\_x" means Harry Potter book number "x" in series\n'**,

**"Numbers are Head-to-Head values of book in row against book in column"**

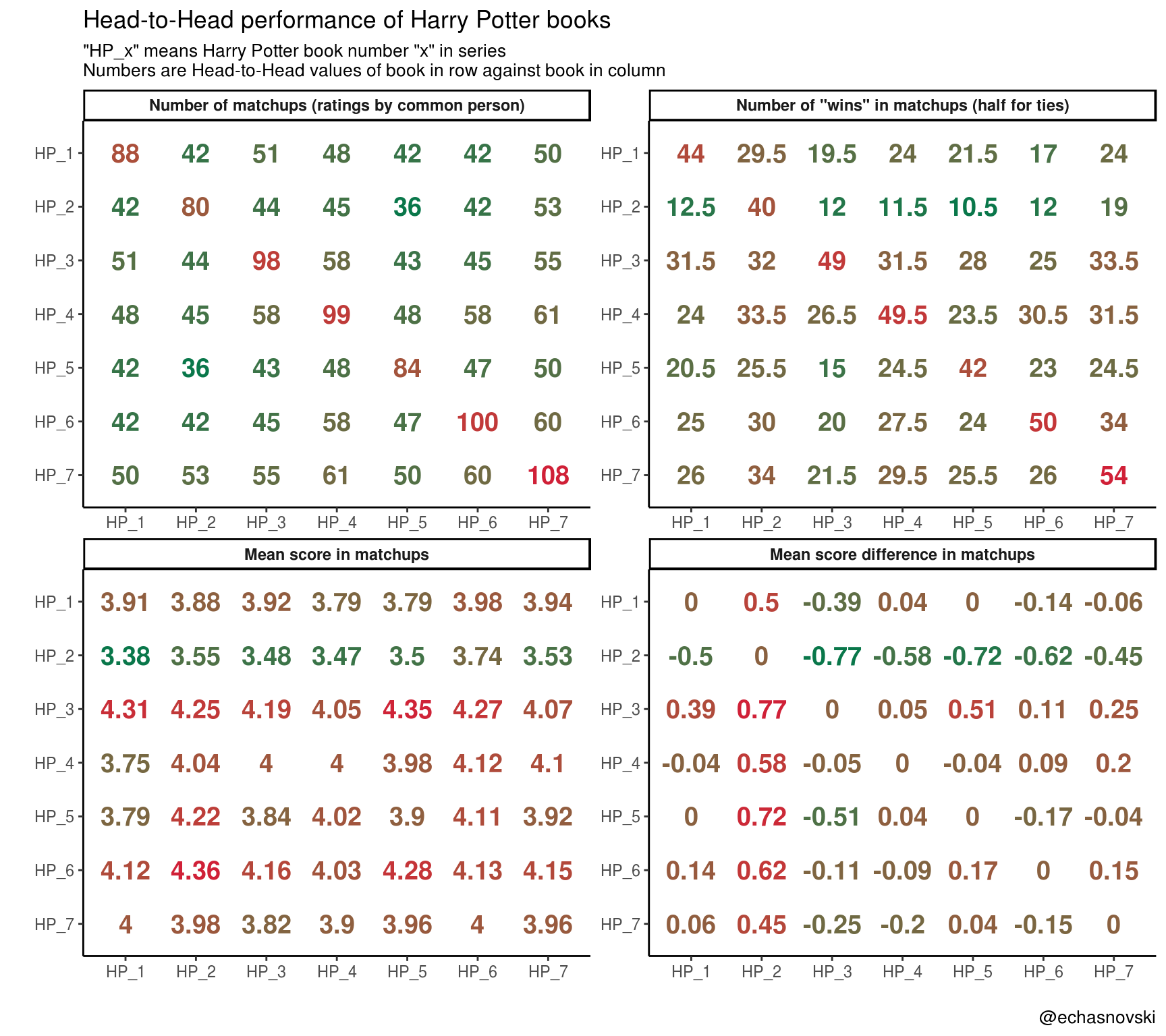
),

caption = **"@echasnovski"**

) +

theme\_classic() +

theme(strip.text = element\_text(face = **"bold"**))



There is a lot of information hidden in this plot. The most obvious discoveries:

* It happened that book “HP\_7” (“Deathly Hallows”) was rated with “HP\_4” (“Goblet of Fire”) by one person the most: 61 times.
* “HP\_7” scored over “HP\_2” (“Chamber of Secrets”) the most wins (34, half for ties) as did “HP\_6” (“Half-Blood Prince”) over “HP\_7”.
* Book “HP\_6” made the highest mean score of 4.36 in matchups with “HP\_2”, which is bigger by 0.23 from its overall mean score.
* In terms of score differences, “HP\_3” (“Prisoner of Azkaban”) did best in matchups with “HP\_2”, scoring on average 0.77 points more. This pair also represents “the best” and “the worst” books in terms of mean score.

**Overview**

This post, has two goals:

* Explore different types of rankings on Harry Potter Books Survey results (data provided by comperes).
* // Function to generate all non-empty subsets of array
* **function** generatePowerSet(array) {
* **var** result = [];
* **for** (**var** i = 1; i < (1 << array.length); i++) {
* **var** subset = [];
* **for** (**var** j = 0; j < array.length; j++)
* **if** (i & (1 << j))
* subset.push(array[j]);
* result.push(subset);
* }
* **return** result;
* }
* // Function to create target survey
* **function** createHPSurvey() {
* **var** form = FormApp.create(**'Harry Potter Books Survey'**)
* .setAllowResponseEdits(**false**)
* .setCollectEmail(**false**)
* .setLimitOneResponsePerUser(**true**);
* // Add select list
* **var** selectList = form.addListItem()
* .setTitle(**'Choose first listed number'**)
* .setHelpText(**'This simulates random subsetting of books.'**)
* .setRequired(**true**);
* // Initialize main questions data
* **var** questionSingular = **'What is your impression of this Harry Potter BOOK?'**;
* **var** questionPlural = **'What is your impression of these Harry Potter BOOKS?'**;
* **var** likertScale = [**'1 - Poor'**, **'2 - Fair'**, **'3 - Good'**,
* **'4 - Very Good'**, **'5 - Excellent'**];
* **var** books = [**"HP and the Philosopher's (Sorcerer's) Stone (#1)"**,
* **"HP and the Chamber of Secrets (#2)"**,
* **"HP and the Prisoner of Azkaban (#3)"**,
* **"HP and the Goblet of Fire (#4)"**,
* **"HP and the Order of the Phoenix (#5)"**,
* **"HP and the Half-Blood Prince (#6)"**,
* **"HP and the Deathly Hallows (#7)"**];
* **var** allSubsets = generatePowerSet(books);
* // Create pages with all subsets
* **var** pages = []; // for collecting the choices in the list item
* **for** (**var** n = 0; n < allSubsets.length; n++) {
* // Make a section for current subset
* **var** newPage = form.addPageBreakItem()
* .setTitle(**'Rate books'**);
* // Set the section to submit after completing (rather than next subset section)
* newPage.setGoToPage(FormApp.PageNavigationType.SUBMIT)
* // Add question for current subset with scale
* **var** question = form.addGridItem()
* .setRows(allSubsets[n])
* .setColumns(likertScale)
* .setRequired(**true**);
* **if** (allSubsets[n].length == 1) {
* question.setTitle(questionSingular);
* } **else** {
* question.setTitle(questionPlural);
* }
* // Push our choice to the list select
* pages.push(selectList.createChoice(n + 1, newPage));
* }
* // Add all subsets to select list
* selectList.setChoices(pages);
* }
* This code should be run into Google Apps Script project. It creates a Google Form named “Harry Potter Books Survey” and stores it on Google Drive.
* Demonstrate basic functionality of comperank package. We will cover the following topics:
* Short notes about **functionality of comperank**.
* **Exploration ranking** with ranking based on mean book score. No comperank package functionality is required.
* **Rankings with fixed Head-to-Head structure**. This will cover Massey and Colley ranking methods.
* **Rankings with variable Head-to-Head structure**. This will cover Keener, Markov and Offense-Defense ranking methods.
* **Combined rankings** in which average ranks will be computed using all described comperank methods.

Another very interesting set of ranking methods implemented in comperank are methods with iterative nature. However, their usage with mentioned Harry Potter Books Survey dataset is meaningless as temporal ordering of games (acts of book scoring by one person) should make sense, which it doesn’t.

The idea behind converting survey results into competition results is described in aforementioned post. We will need the following setup:

library(dplyr)

library(purrr)

library(rlang)

# This will automatically load {comperes}

library(comperank)

# Create competition results from hp\_survey

hp\_cr <- hp\_survey %>%

transmute(

game = person, player = book,

score = as.integer(gsub("[^0-9].\*$", "", score))

) %>%

as\_longcr()

**Functionality of comperank**

**Rating** is considered to be a list (in the ordinary sense) of numerical values, one for each player, or the numerical value itself. Its interpretation depends on rating method: either bigger value indicates better player performance or otherwise.

**Ranking** is considered to be a rank-ordered list (in the ordinary sense) of players: rank 1 indicates player with best performance.

comperank leverages the tidyverse ecosystem of R packages. Among other things, it means that the main output format is tibble.

There are three sets of functions:

* rate\_\*() (\* stands for ranking method short name). Its output is a tibble with columns player (player identifier) and at least one rating\_\* (rating value). Names of rating columns depend on rating method.
* rank\_\*(). Its default output is similar to previous one, but with ranking\_\* instead of rating columns. It runs rate\_\*() and does ranking with correct direction. One can use option keep\_rating = TRUE to keep rating columns in the output.
* add\_\*\_ratings(). These functions are present only for algorithms with iterative nature and competition results with games only between two players. They return tibble with row corresponding to a game and extra columns indicating ratings of players before and after the game.

**Exploration ranking**

Previously we established that “Harry Potter and the Prisoner of Azkaban” seems to be “the best” book and “Harry Potter and the Chamber of Secrets” comes last. This was evaluated by mean score:

hp\_rank\_explore <- hp\_cr %>%

summarise\_player(rating\_explore = mean(score)) %>%

# round\_rank() is a function from {comperank} package for doing ranking

mutate(ranking\_explore = round\_rank(rating\_explore))

hp\_rank\_explore

## # A tibble: 7 x 3

## player rating\_explore ranking\_explore

##

## 1 HP\_1 3.91 5

## 2 HP\_2 3.55 7

## 3 HP\_3 4.19 1

## 4 HP\_4 4 3

## 5 HP\_5 3.90 6

## 6 HP\_6 4.13 2

## 7 HP\_7 3.96 4

As simple as it is, this approach might leave some available information unused. Survey originally was designed to obtain information not only about books performance as separate objects, but also to learn about possible pair relationships between them. Maybe some book is considered generally “not the best” but it “outperforms” some other “better” book. This was partially studied in “Harry Potter and competition results with comperes” by computing different Head-to-Head values and manually studying them.

Here we will attempt to summarise books performance based on their Head-to-Head relationships.

**Rankings with fixed H2H structure**

In comperank there are two methods which operate on fixed Head-to-Head structure: **Massey** and **Colley**. Both of them are designed for competitions where:

* Games are held only between two players.
* It is assumed that score is numeric and higher values indicate better player performance in a game.

Being very upset for moment, we realize that in dataset under study there are games with different number of players. Fortunately, comperes package comes to rescue: it has function to\_pairgames() just for this situation. It takes competition results as input and returns completely another (strictly speaking) competition results where “crowded” games are split into small ones. More strictly, games with one player are removed and games with three and more players are converted to multiple games between all unordered pairs of players. The result is in wide format:

hp\_cr\_paired <- to\_pairgames(hp\_cr)

# For example, second game was converted to a set of 10 games

hp\_cr %>% filter(game == 2)

## # A longcr object:

## # A tibble: 5 x 3

## game player score

##

## 1 2 HP\_1 3

## 2 2 HP\_4 5

## 3 2 HP\_5 2

## 4 2 HP\_6 4

## 5 2 HP\_7 5

hp\_cr\_paired %>% slice(2:11)

## # A widecr object:

## # A tibble: 10 x 5

## game player1 score1 player2 score2

##

## 1 2 HP\_1 3 HP\_4 5

## 2 3 HP\_1 3 HP\_5 2

## 3 4 HP\_1 3 HP\_6 4

## 4 5 HP\_1 3 HP\_7 5

## 5 6 HP\_4 5 HP\_5 2

## 6 7 HP\_4 5 HP\_6 4

## 7 8 HP\_4 5 HP\_7 5

## 8 9 HP\_5 2 HP\_6 4

## 9 10 HP\_5 2 HP\_7 5

## 10 11 HP\_6 4 HP\_7 5

**Massey method**

Idea of Massey method is that difference in ratings should be proportional to score difference in direct confrontations. Bigger value indicates better player competition performance.

hp\_cr\_massey <- hp\_cr\_paired %>% rank\_massey(keep\_rating = TRUE)

hp\_cr\_massey

## # A tibble: 7 x 3

## player rating\_massey ranking\_massey

##

## 1 HP\_1 -0.00870 5

## 2 HP\_2 -0.514 7

## 3 HP\_3 0.293 1

## 4 HP\_4 0.114 3

## 5 HP\_5 0.00195 4

## 6 HP\_6 0.124 2

## 7 HP\_7 -0.00948 6

**Colley method**

Idea of Colley method is that ratings should be proportional to share of player’s won games. Bigger value indicates better player performance.

hp\_cr\_colley <- hp\_cr\_paired %>% rank\_colley(keep\_rating = TRUE)

hp\_cr\_colley

## # A tibble: 7 x 3

## player rating\_colley ranking\_colley

##

## 1 HP\_1 0.497 5

## 2 HP\_2 0.326 7

## 3 HP\_3 0.599 1

## 4 HP\_4 0.534 3

## 5 HP\_5 0.505 4

## 6 HP\_6 0.542 2

## 7 HP\_7 0.497 6

Both Massey and Colley give the same result differing from Exploration ranking in treating “HP\_5” (“Order of the Phoenix”) and “HP\_7” (“Deathly Hallows”) differently: “HP\_5” moved up from 6-th to 4-th place.

**Rankings with variable H2H structure**

All algorithms with variable Head-to-Head structure depend on user supplying custom Head-to-Head expression for computing quality of direct confrontations between all pairs of players of interest.

There is much freedom in choosing Head-to-Head structure appropriate for ranking. For example, it can be “number of wins plus half the number of ties” (implemented in h2h\_funs[["num\_wins2"]] from comperes) or “mean score difference from direct matchups” (h2h\_funs[["mean\_score\_diff"]]). In this post we will use the latter one. Corresponding Head-to-Head matrix looks like this:

hp\_h2h <- hp\_cr %>%

h2h\_mat(!!! h2h\_funs[["mean\_score\_diff"]]) %>%

round(digits = 2)

# Value indicates mean score difference between "row-player" and

# "column-player". Positive - "row-player" is better.

hp\_h2h

## # A matrix format of Head-to-Head values:

## HP\_1 HP\_2 HP\_3 HP\_4 HP\_5 HP\_6 HP\_7

## HP\_1 0.00 0.50 -0.39 0.04 0.00 -0.14 -0.06

## HP\_2 -0.50 0.00 -0.77 -0.58 -0.72 -0.62 -0.45

## HP\_3 0.39 0.77 0.00 0.05 0.51 0.11 0.25

## HP\_4 -0.04 0.58 -0.05 0.00 -0.04 0.09 0.20

## HP\_5 0.00 0.72 -0.51 0.04 0.00 -0.17 -0.04

## HP\_6 0.14 0.62 -0.11 -0.09 0.17 0.00 0.15

## HP\_7 0.06 0.45 -0.25 -0.20 0.04 -0.15 0.00

**Keener method**

Keener method is based on the idea of “relative strength” – the strength of the player relative to the strength of the players he/she has played against. This is computed based on provided Head-to-Head values and some flexible algorithmic adjustments to make method more robust. Bigger value indicates better player performance.

hp\_cr\_keener <- hp\_cr %>%

rank\_keener(!!! h2h\_funs["mean\_score\_diff"], keep\_rating = TRUE)

hp\_cr\_keener

## # A tibble: 7 x 3

## player rating\_keener ranking\_keener

##

## 1 HP\_1 0.147 5

## 2 HP\_2 0.0816 7

## 3 HP\_3 0.191 1

## 4 HP\_4 0.150 4

## 5 HP\_5 0.153 3

## 6 HP\_6 0.155 2

## 7 HP\_7 0.122 6

Results for Keener method again raised “HP\_5” one step up to third place.

**Markov method**

The main idea of Markov method is that players “vote” for other players’ performance. Voting is done with Head-to-Head values and the more value the more “votes” gives player2 (“column-player”) to player1 (“row-player”). For example, if Head-to-Head value is “number of wins” then player2 “votes” for player1 proportionally to number of times player1 won in a matchup with player2.

Actual “voting” is done in [Markov chain](https://en.wikipedia.org/wiki/Markov_chain) fashion: Head-to-Head values are organized in stochastic matrix which vector of stationary probabilities is declared to be output ratings. Bigger value indicates better player performance.

hp\_cr\_markov <- hp\_cr %>%

rank\_markov(!!! h2h\_funs["mean\_score\_diff"], keep\_rating = TRUE)

hp\_cr\_markov

## # A tibble: 7 x 3

## player rating\_markov ranking\_markov

##

## 1 HP\_1 0.140 5

## 2 HP\_2 0.0500 7

## 3 HP\_3 0.196 1

## 4 HP\_4 0.168 2

## 5 HP\_5 0.135 6

## 6 HP\_6 0.167 3

## 7 HP\_7 0.143 4

We can see that Markov method put “HP\_4” (“Goblet of Fire”) on second place. This is due to its reasonably good performance against the leader “HP\_3” (“Prisoner of Azkaban”): mean score difference is only 0.05 in “HP\_3” favour. Doing well against the leader in Markov method has a great impact on output ranking, which somewhat resonates with common sense.

**Offense-Defense method**

The idea of Offense-Defense (OD) method is to account for different abilities of players by combining different ratings:

* For player which can achieve *high* Head-to-Head value (even against the player with strong defense) it is said that he/she has **strong offense** which results into *high* offensive rating.
* For player which can force their opponents into achieving *low* Head-to-Head value (even if they have strong offense) it is said that he/she has **strong defense** which results into *low* defensive rating.

Offensive and defensive ratings describe different skills of players. In order to fully rate players, OD ratings are computed: offensive ratings divided by defensive. The more OD rating the better player performance.

hp\_cr\_od <- hp\_cr %>%

rank\_od(!!! h2h\_funs["mean\_score\_diff"], keep\_rating = TRUE)

print(hp\_cr\_od, width = Inf)

## # A tibble: 7 x 7

## player rating\_off rating\_def rating\_od ranking\_off ranking\_def

##

## 1 HP\_1 5.42 1.03 5.29 5 5

## 2 HP\_2 1.45 1.88 0.771 7 7

## 3 HP\_3 7.91 0.522 15.1 1 1

## 4 HP\_4 6.51 0.869 7.49 3 3

## 5 HP\_5 5.30 0.888 5.97 6 4

## 6 HP\_6 6.59 0.809 8.14 2 2

## 7 HP\_7 5.54 1.05 5.29 4 6

## ranking\_od

##

## 1 5

## 2 7

## 3 1

## 4 3

## 5 4

## 6 2

## 7 6

All methods give almost equal results again differing only in ranks of “HP\_5” and “HP\_7”.

**Combined rankings**

To obtain averaged, and hopefully less “noisy”, rankings we will combine rankings produced with comperank by computing their mean.

list(hp\_cr\_massey, hp\_cr\_colley, hp\_cr\_keener, hp\_cr\_markov, hp\_cr\_od) %>%

# Extract ranking column

map(. %>% select(., player, starts\_with("ranking"))) %>%

# Join all ranking data in one tibble

reduce(left\_join, by = "player") %>%

# Compute mean ranking

transmute(player, ranking\_combined = rowMeans(select(., -player))) %>%

# Join exploration rankings for easy comparison

left\_join(y = hp\_rank\_explore %>% select(-rating\_explore), by = "player")

## # A tibble: 7 x 3

## player ranking\_combined ranking\_explore

##

## 1 HP\_1 5 5

## 2 HP\_2 7 7

## 3 HP\_3 1 1

## 4 HP\_4 3 3

## 5 HP\_5 4.43 6

## 6 HP\_6 2.14 2

## 7 HP\_7 5.43 4

As we can see, although different ranking methods handle results differently for books with “middle performance”, combined rankings are only slightly different from exploration ones. Only notable difference is in switched rankings of “Order of the Phoenix” and “Deathly Hallows”.

**Conclusion**

* “Harry Potter and the Prisoner of Azkaban” still seems to be considered “best” among R users. And yet “Harry Potter and the Chamber of Secrets” still suffers the opposite fate.
* Using different ranking methods is a powerful tool in analyzing Head-to-Head performance. This can be done in very straightforward manner with new addition to CRAN – comperank package.

sessionInfo()

## R version 3.4.4 (2018-03-15)

## Platform: x86\_64-pc-linux-gnu (64-bit)

## Running under: Ubuntu 16.04.4 LTS

##

## Matrix products: default

## BLAS: /usr/lib/openblas-base/libblas.so.3

## LAPACK: /usr/lib/libopenblasp-r0.2.18.so

##

## locale:

## [1] LC\_CTYPE=ru\_UA.UTF-8 LC\_NUMERIC=C

## [3] LC\_TIME=ru\_UA.UTF-8 LC\_COLLATE=ru\_UA.UTF-8

## [5] LC\_MONETARY=ru\_UA.UTF-8 LC\_MESSAGES=ru\_UA.UTF-8

## [7] LC\_PAPER=ru\_UA.UTF-8 LC\_NAME=C

## [9] LC\_ADDRESS=C LC\_TELEPHONE=C

## [11] LC\_MEASUREMENT=ru\_UA.UTF-8 LC\_IDENTIFICATION=C

##

## attached base packages:

## [1] methods stats graphics grDevices utils datasets base

##

## other attached packages:

## [1] bindrcpp\_0.2.2 comperank\_0.1.0 comperes\_0.2.0 rlang\_0.2.0

## [5] purrr\_0.2.4 dplyr\_0.7.5

##

## loaded via a namespace (and not attached):

## [1] Rcpp\_0.12.17 knitr\_1.20 bindr\_0.1.1 magrittr\_1.5

## [5] tidyselect\_0.2.4 R6\_2.2.2 stringr\_1.3.1 tools\_3.4.4

## [9] xfun\_0.1 utf8\_1.1.3 cli\_1.0.0 htmltools\_0.3.6

## [13] yaml\_2.1.19 rprojroot\_1.3-2 digest\_0.6.15 assertthat\_0.2.0

## [17] tibble\_1.4.2 crayon\_1.3.4 bookdown\_0.7 tidyr\_0.8.1

## [21] glue\_1.2.0 evaluate\_0.10.1 rmarkdown\_1.9 blogdown\_0.6

## [25] stringi\_1.2.2 compiler\_3.4.4 pillar\_1.2.2 backports\_1.1.2

## [29] pkgconfig\_2.0.1