SY09 Printemps 2016
Éléments d'aide pour l'analyse des données de paris

Éléments uniques d'un vecteur

La fonction `unique` permet de récupérer les éléments uniques apparaissant dans un vecteur (c'est-à-dire après suppression des valeurs redondantes) : par exemple, on peut construire un vecteur contenant les identifiants des différents matches par la commande :

```
unique(books.sel$match_uid)
```

Notons que lorsque la variable d'intérêt est déclarée sous R comme variable qualitative, la fonction `levels` permet de récupérer ses modalités :

```
levels(books.sel$match_uid)
```

La différence est que `unique` ne renverra que les éléments différents *présents* dans le vecteur, alors que `levels` renverra l'*ensemble des modalités déclarées* (même absentes du vecteur) ; par exemple, on verra la différence si l'on compare les (tailles des) vecteurs obtenus par les commandes

```
unique(books.sel$winner)
levels(books.sel$winner)
```

Identifier les répétitions dans un vecteur

La fonction `duplicated` permet d'identifier les éléments d'un vecteur qui sont des répétitions (déjà apparus dans le vecteur) ; elle renvoie un `logical` (donc une valeur égale à `TRUE` ou `FALSE`). Sa négation (obtenue en utilisant l'opérateur logique `!`) permet donc de déterminer les premières occurrences d'une valeur dans le vecteur :

```
test <- c(rep(1,3),rep(2,4),rep(3,3))
duplicated(test)
!duplicated(test)
```

L'utilisation de la fonction `which` permettra de récupérer les indices correspondants :

```
which(!duplicated(test))
```

Remarquons qu'on peut donc retrouver avec `duplicated` le résultat obtenu par la commande `unique` ; on peut ainsi vérifier que les deux résultats sont équivalents :

```
identical(books.sel$winner[which(!duplicated(books.sel$winner))],
    unique(books.sel$winner))
```

D'un tableau de données à un autre

Création explicite On peut créer un nouveau jeu de données, par exemple relatif aux joueurs, avec la fonction `data.frame` :

```
players <- data.frame(
    player_uid=factor(levels(matches$winner), levels=levels(matches$winner)),
    champ2=...,
    champ3=...,
    ...)
```

On remarquera ici que la première variable (`player_uid`) a été définie comme variable qualitative, et qu'on a explicitement ordonné ses modalités dans l'ordre des modalités du tableau de données `matches`. L'objectif est ici de préserver au maximum la cohérence entre les différents tableaux de données que l'on manipule.

De manière alternative, on pourrait également utiliser la fonction `table`, qui construit un tableau de contingence à partir d'une variable (généralement qualitative ou quantitative discrète). Dans le cas d'une variable qualitative, `table` respecte l'ordre des modalités ; en revanche, *elle se base sur les modalités et non sur les valeurs apparaissant dans le vecteur qu'on lui transmet.* Par exemple, la commande suivante compte le nombre de paris par match y compris pour les matches non joués :

```
table(books.sel$match_uid)
```

En effet, ces matches ont été supprimés du tableau `books.sel`, mais leurs modalités apparaissent encore dans `levels(books.sel$match_uid)` et sont donc prises en compte.

Duplications La fonction `duplicated` peut être utilisée pour extraire un ensemble de données relatif aux matches à partir du tableau `books.sel`.

En effet, on peut sélectionner dans `books.sel` les lignes correspondant à la première occurrence de chaque valeur de `match_uid` : cela revient à ne garder qu'une seule ligne, correspondant au premier des paris relatifs à ce match. Notons qu'on supprime également les colonnes (variables) correspondant aux paris pour ne garder que les informations relatives aux matches (lesquelles, comme l'année ou le gagnant, sont identiques pour tous les paris correspondant au même match) :

```
matches <- books.sel[which(!duplicated(books.sel$match_uid)),
    -c(1,2,3,4,5,7,8,9,10,11,12)]
matches <- matches[sort.int(as.character(matches$match_uid), index.return=T)$ix,]
```

Attention — Le nouveau tableau de données ainsi formé (`matches`) comporte autant de lignes qu'il apparaît de valeurs de la variable `match_uid` dans l'ancien (`books.sel`). Ainsi, les modalités de `match_uid` absentes du tableau `books.sel` (ici, les matches annulés ou avec des cotes initiales atypiques, exclus lors des prétraitements, mais toujours présents dans `levels(books.sel$match_uid)`) ne seront donc pas présentes dans le tableau `matches`. Cela peut poser problème si l'on souhaite ensuite remplir des champs du tableau `matches` au moyen de la commande `table`, laquelle considère toutes les modalités.

La seconde ligne de commande exécutée ci-dessus consiste à ré-ordonner les lignes en fonction de l'ordre des modalités de `match_uid`. En effet, sans cela, les matches décrits dans le tableau `matches` seraient classés dans l'ordre où les valeurs de `match_uid` apparaissent dans le tableau `books.sel`, ce qui pourrait également poser des problèmes de cohérence. — Attention

Agrégations La fonction `aggregate` (utilisée notamment dans le script de prétraitements) offre d'autres possibilités pour former un nouveau tableau de données à partir des informations contenues dans un tableau existant (par exemple, pour constituer un tableau relatif aux matches à partir d'un

tableau relatif aux paris). Elle agrège les données en appliquant une fonction (à préciser) à un ou plusieurs champs du tableau de données initial, en fonction d'une variable.

Par exemple, la commande suivante compte, pour chaque match, le nombre de paris ayant évolué en faveur du vainqueur final :

```
aggregate(moved_towards_winner~match_uid, data=books.sel, FUN=sum)[,2]
```

Soulignons que le vecteur ou le tableau de données obtenu via cette fonction respecte l'ordre des modalités de la variable `match_uid` du tableau `books.sel`.

Copier des données d'un tableau à un autre Supposons que l'on dispose d'un tableau de données `players` contenant des informations relatives aux joueurs, dont notamment l'identifiant du joueur (variable qualitative nominale, stockée dans la variable `player_uid`), et son niveau de jeu (variable qualitative ordinale, stockée dans une variable nommée `play_level`).

Imaginons que l'on souhaite intégrer cette information de niveau de jeu dans le tableau `books.sel` : pour chaque pari, on souhaite disposer du niveau de jeu du joueur gagnant. Cette opération peut être réalisée facilement de la manière suivante :

```
books.sel$winner_level <- players$play_level[books.sel$winner]
```

Attention — Pour que cette opération donne un résultat correct, il est impératif que les indices suivant lesquels on extrait les informations, obtenus à partir des modalités de `books.sel$winner`, correspondent bien aux lignes désirées du tableau `players`. Il faudra ici vérifier que les modalités de `books.sel$winner` sont bien identiques aux identifiants `players$player_uid)` :

```
identical(as.character(players$player_uid),levels(books.sel$winner))
```

Dans le cas contraire, le transtypage de `books.sel$winner` (dont les valeurs sont transformées en une série d'entiers utilisés comme indices) donne des indices de ligne qui ne figurent pas dans le tableau `players`. On pourra comprendre ce phénomène sur l'exemple suivant :

```
paris <- data.frame(
    id_pari = as.factor(1:10),
    id_match = as.factor(c(rep(2,3),rep(1,4),rep(3,3))),
    id_gagnant = factor(c("B","B","B","B","B","B","B","B","B","B"),
        levels=c("A","B","C","D","E")),
    id_perdant = factor(c("A","A","A","E","E","E","E","C","C","C"),
        levels=c("A","B","C","D","E"))
)

joueurs <- data.frame(
    id_joueur = factor(c("A","B","C","E"),
        levels=c("A","B","C","D","E")),
    nb_gagnes = c(0,3,0,0),
    nb_perdus = c(1,0,1,1),
    niveau = c(2,1,3,4)
)

paris$niveau_gagnant <- joueurs$niveau[paris$id_gagnant]
paris$niveau_perdant <- joueurs$niveau[paris$id_perdant]
joueurs
paris
```

— Attention