Exercices - 5

Préparation

1. Installez le package `devtools`.

Exercice

L'objectif de cet exercice sera de créer un package nous permettant de fournir des outils d'analyses des données de trajets vélos sur Nantes. Il consistera principalement à "packager" les fonctions développées lors des exercices des sessions précédentes. Réaliser des `commit` régulier avec des messages explicites sur vos modifications.

Initialisation

1. Dans une nouvelle session R, créer un nouveau projet utilisant la fonction `create_package` de la librairie `usethis`. Ce package sera appelé `firstlib.<name>`; vous replacerez `<name>` par votre nom. Pourquoi n'est-il pas nécessaire d'installer explicitement `usethis` ?
2. Dans votre projet RStudio, initialisez git et votre premier commit avec la commande `use_git` de la librairie `usethis`.
3. Configurez ensuite un nouveau projet vide sous GitHub qui s'appellera `firstlib.<name>`. En utilisant la commande `use_git_remote` de la librairie `usethis`, configurez le lien entre votre projet en local et votre répertoire distant. La commande `usethis::git_remotes()` doit vous retourner l'url de votre répertoire comme `origin`.
4. Dans le terminal RStudio, faites un `push` pour publier votre premier commit sur GitHub. Quelle est le nom de la branche principale ?
5. Éditez le fichier `DESCRIPTION` en renseignant les champs obligatoires et faites un nouveau `commit`/`push`.

Fonctions

6. Récupérez dans un script `R/velo.R` les fonctions préalablement implémentées lors de la partie 3 :

- `filtre_anomalie()`
- `compter_nombre_trajets()`
- `compter_nombre_boucle()`
- `trouver_trajet_max()`
- `calcul_distribution_semaine()`
- `plot_distribution_semaine()`

Pour chacune de ces fonctions, documentez et renseignez les dépendances en utilisant les notations `Roxygen2`. Soyez particulièrement attentifs à la déclaration des dépendances, si vous utilisez des librairies tierces pour réaliser vos calculs.

7. Une fois les fonctions documentées avec `Roxygen2`, générez la documentation en utilisant la fonction `devtools::document()`. Assurez-vous que les fonctions sont bien documentées en utilisant la fonction `devtools::load_all()`, puis en consultant l'aide de la fonction, par exemple `?compter_nombre_trajets`.

Dépendances

8. Dans le fichier `DESCRIPTION`, renseigner les dépendances nécessaires. Lancer la commande `devtools::check` pour vérifier la bonne gestion des dépendances. Ajuster si nécessaire.

Données

Afin de limiter la taille de notre package et de notre répertoire GitHub, nous allons intégrer uniquement les trajets des vacances de Toussaint 2025.

9. Télécharger le jeu de données ici, puis charger les données en utilisant la fonction `readr::read_delim`. En utilisant la fonction `use_data` de la librairie `usethis`, ajouter le jeu de données dans le package sous le nom `df_velo`. Assurez-vous que les données sont rendues disponibles dans la librairie.

10. Documentez le jeu de données comme décrit ici. Assurez-vous que la documentation soit rendue disponible dans la librairie avec la commande `?df_velo`.

11. Créer une nouvelle branche Git. Depuis cette branche, ajouter la fonction `filtrer_trajet()`, qui prendra en entrée un `data.frame` et un vecteur de numéro de boucle. Elle renverra en sortie le `data.frame` avec les boucles sélectionnées.

Exemple d'appel valide : `filtrer_trajet(trajet = df_velo, boucle = c("880", "881"))`

Une fois la fonction implémentée, récupérer les modifications sur `master` (ou `main`) via un `push` suivi d'un `merge` depuis la `remote` sur GitHub. Penser à `pull` depuis `master` avant de poursuivre vos modifications.

Tests

12. En utilisant `testthat`, écrivez des tests unitaires toutes les fonctions de votre package. Essayez de tester soit la validité des arguments fournis en entrée, soit la pertinence des résultats pour un jeu de données fixé en entrée. Vous pourrez créer les fichiers de tests en utilisant l'instruction `usethis::use_test()`.

13. Lancer la commande `devtools::test()` pour vous assurer que vos tests fonctionnent. Lancer les commandes `covr::package_coverage()` et `covr::report()` du package `{covr}` pour connaitre la couverture de test. Obtenir une couverture minimum de *50%*.

14. Initiation au *Test Driven Developement* : Ajouter un test qui s'assure que la fonction `filtrer_trajet()` renvoie un jeu de données non filtré si le paramètre `boucle` est `NULL`. Adapter votre fonction pour que ce nouveau test passe.

Partage

15. Créer une vignette qui décrit les commandes à effectuer pour réaliser un filtre sur une boucle puis un graphique à partir des données `df_velo`.

16. Créer une documentation HTML de votre package avec `build_site()` de `pkgdown`. Explorer la page crée, quels éléments de documentation retrouvez-vous ?

Installation

17. Assurez-vous d'avoir votre banche à jour, et réalisez un `commit`/`push` vers le répertoire distant.

18. Redémarrez une nouvelle session R. En utilisant la fonction `remotes::install_github("<username_github>/<nom_du_repo_github>")`, installez votre package depuis le répertoire distant. Chargez-le en utilisant l'instruction `library()`. Lancer la commande `plot_distribution_semaine(df_velo)`. Cela marche-t-il comme attendu ?

Félicitations, vous avez construit votre premier package !

19. Depuis une branche `feat`, ajouter à la fonction `calcul_distribution_semaine()` un nouveau paramètre `filtre` qui prend une valeur booléenne (`TRUE`/`FALSE`). Si `filtre = FALSE`, alors le jeu de données en entrée *ne sera pas filtré* avant calcul. Utiliser le conditionnement `if()` pour adapter la fonction.

20. Ajuster la documentation et les tests pour prendre en compte les changements apportés par la question précédente.

21. Appliquer ce changement sur `main` ou `master` depuis GitHub. Re-installer le package et vérifier que vous pouvez désormais utiliser ce nouveau paramètre.

Bonus

22. Faire en sorte que votre `devtools::check()` ne renvoie aucun warning, note ou erreur.

23. Faire en sorte d'avoir une couverture de test à *100%*.

24. Prendre exemple sur les répertoires GitHub de packages existants comme `{glue}`, et ajouter à votre package :

- un logo (*hex sticker*) sur le `README` (*cf.* hexmake)
- un fichier NEWS et une version 1.0.0
- une licence MIT
- un badge de version

Le package `usethis` propose des fonctions pour vous aider sur chacune de ces étapes, explorer la doc.

25. On peut implémenter des pipelines de CI/CD avec la commande `usethis::use_github_action()`. Explorer la doc de `usethis` et faire en sorte que votre documentation soit accessible sur une *GitHub Page* et automatiquement mise à jour à chaque `push` sur `master`. Pour cet exercice, vérifiez que vos paramètres GitHub sur le répertoire du package autorisent les **Actions** (*Settings* > *Actions* > *General*).

Tips

- Il est possible de réaliser la mise à jour des dépendances automatiquement via l'appel à `attachment::att_amend_desc()`.
- Il est possible de réaliser la documentation des données en partie automatiquement via l'appel à `checkhelper::use_data_doc()`.
- Vous pouvez avoir un aperçu de la couverture de vos tests avec la commande `covr::package_coverage()`.
- Pour tester l'installation de son package sans GitHub, on peut l'installer depuis sa version locale avec la commande `devtools::install_local`. Penser à bien vérifier dans quelle branche vous êtes et si vous êtes à jour avec la `remote`.
- Pour corriger les notes/warning du check, vous pouvez vous aider de `checkhelper::print_globals()` et le package `prefixer`