**Note de qualité de code**

**Objectif**:

Ce document a pour objectif de cadrer les méthodes de développement et de qualité de code à utiliser pour les initiatives de développement python au sein de la direction Data et transformation SGCI

Sommaire

[Création du projet 1](#_Toc147680007)

[Création du répertoire distant sur SGitHub 1](#_Toc147680008)

[Arborescence par défaut 1](#_Toc147680009)

[Installation de l’environnement virtuel 2](#_Toc147680010)

[Installation du pré-commit 2](#_Toc147680011)

[Ecriture de code 2](#_Toc147680012)

[Protéger les branches prod et homo/test 3](#_Toc147680013)

[Coding 3](#_Toc147680014)

[Créer une merge request 3](#_Toc147680015)

# Création du projet

## Création du répertoire distant sur SGitHub

Pour ce faire, se rendre sur SGitHub, créer un répertoire interne. Par suite créer les branches test/homo et prod. Deux (2) en tout. Et enfin cloner le repo distant sur le serveur de dev.

## Arborescence par défaut

(Soit TEMPLATE\_PROJECT le projet template créé comme base pour toute nos initiatives)

Copier tout le contenu du projet TEMPLATE\_PROJECT (sans les dossiers .git et venv) à la racine du nouveau répertoire. Ce qui devrait donner une arborescence comme sur la figure ci-dessous

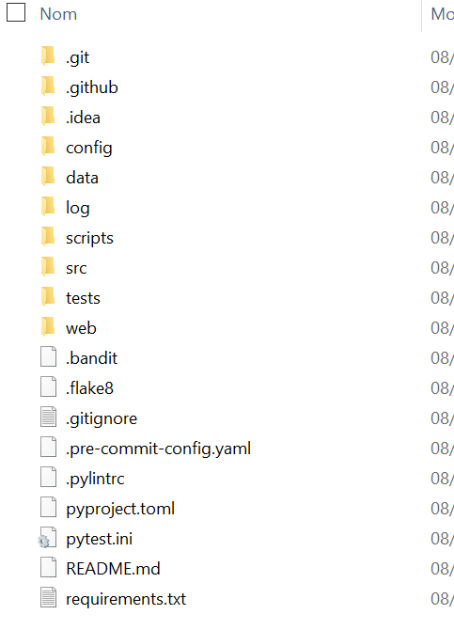


Figure 1: Arborescence par défaut du projet en création

Une fois l’arborescence en place, il convient de modifier le fichier README à la racine du répertoire pour y ajouter les mentions :

PROJECT\_NAME, PROJECT\_TITLE, URL\_TO\_PROJECT, …

## Installation de l’environnement virtuel

La commande de création de l’environnement virtuelle :

python -m venv venv

Puis

venv/Scripts/activate

Pour les distributions linux, la commande ci-dessus est remplacée par

source venv/bin/activate

Par la suite, il convient de renseigner le fichier « requirements.txt » avec la liste des packages à installer pour le projet et exécuter la commande ci-dessous pour effectivement installer ces packages

pip install -r requirements.txt

Notons que toutes ces étapes sont faisables en mode clic lorsque nous utilisons PyCharm.

## Installation du pré-commit

Pour installer nos règles de qualité de code il suffit d’exécuter les lignes ci-dessous

pip install pre-commit  
pre-commit install  
pre-commit autoupdate

Ceci étant, pour nous assurer que tout fonctionne comme prévu exécuter :

pre-commit run --all-files

# Ecriture de code

SI GITHub fonctionne

## Protéger les branches prod et homo/test

Sur SGitHub aller dans le répertoire distant du projet puis settings>branches>Add rule

Créer pour chacune des branches prod et homo une règle en cochant le checkbox **Require a pull request before merging**

## Coding

Pour chaque micro fonctionnalité à implémenter, créer une nouvelle branche avant tout développement.

Une bonne pratique consiste à normaliser le nom des branches selon le type de modifications à apporter au code :

* feat/ : rajout d’une nouvelle fonctionnalité
* fix/ : correction d’un bug
* poc/ : pour le développement de POC
* docs/ : Ajout ou modification de documentation
* opti/ : Amélioration des performances
* refactor/ : Modification n’ajoutant pas de fonctionnalités ni de correction de bug (renommage d’une variable, suppression de code redondant, simplification du code, etc.)
* test/ : Ajout ou modification de tests

Pour créer la branche exécuter les commandes ci-dessous

git checkout homo

git pull  
git checkout -b feat/branch\_new\_functionality homo  
git checkout feat/branch\_new\_functionality

Par la suite effectuer les modifications nécessaires au projet puis exécuter :

git add .  
git commit -m message\_du\_commit

En cas de problème de qualité de données lié au pré-commit installer, le commit ne fonctionnera pas. Il sera donc question de résoudre le problème avant de retenter un commit.

Après avoir réussi le commit, il convient de faire un push

git push

## Créer une merge request

Après avoir poussé son code, rendez-vous sur SGitHub et sur le répertoire de travail puis pull request> Create pull request.

COOKIECUTTER