

GESTION DE PROJET & PRODUIT DIGITAL

Morgan COUSIN
morgan27cousin@gmail.com

M2 SEP
ANNÉE UNIVERSITAIRE 2025-2026

HI



PRÉSENTATIONS

MON PARCOURS



Depuis 2025-04: Lead Data & Data Engineer Senior → **advizeo**

2022-01 à 2025-02: Squad Tech Lead & Data Engineer → **JELLYSMACK**

2021-01 à 2021-12: Lead Data Analytics

2018-10 à 2020-12: Data Scientist & Scrum Master → **GreenFlex**

2018-05 à 2018-09: Data Analyst Intern



Depuis 2020: Master SEP - Introduction à la Gestion de Produit Digital

2018-09 à 2023-06: L3 EG - Introduction à la Programmation en **R & python**



2018: Master 2 Statistique pour l'Évaluation et la Prospective (**SEP**)

2017: Master 2 Management & Gestion des Ressources Humaines, **EM Strasbourg**

2016: Master 1 Economie Appliquée

2015: Licence Economie–Gestion, parcours Analyse Économique

PROGRAMME DU MODULE

- Objectifs & Mise en pratique (**30'**)
- Produit & Digital & Data (**40'**)

Break (10')

- Les outils de l'équipe produit (**20'**)
- Structurer son projet python (**30'**)
- En route vers le projet + Q&A (**15'**)

QUESTIONS



MISE EN PRATIQUE



CONTEXTE & OBJECTIFS

A la croisée des chemins: 1 projet = 2 matières évaluées



Réalisation par équipe d'un **Produit Applicatif Analytics**

- Livraison d'interfaces utilisateurs permettant de récolter les informations et de visualiser les résultats
- Intégration au sein de l'application d'un algorithme d'**apprentissage supervisé**

M. KEZIOU représentera le client à qui vous devrez vendre et faire adopter votre application

N'hésitez pas à lui demander conseils & feedbacks tout au long du cycle de développement!

Les outils à disposition



- Interfaces utilisateurs d'entrée et de sorties, à développer soit en **Excel/VBA**, soit en **python**
- Algorithme d'apprentissage supervisé développé en **python** (*utilisation de pycharm ou VSCode recommandée*)
- Un projet **github public** contenant tout le code et les fichiers utilisés (*tous langages confondus*)

Rendus

- Tout le code versionné au travers du répo github (*les diffs de code & contributions individuelles seront pris en compte*)
- Toutes les démos clients (*demos intermédiaires + démo finale*) -> **2 notes: Moyenne des Sprint Review & Note Finale**
- Support écrit présentant:



- Le contexte métier de la démarche et la justification du besoin
- La présentation du produit en tant que réponse au besoin
- Une présentation exhaustive des données utilisées et leur provenance
- Les résultats statistiques obtenus et commentés au fil des étapes
- La présentation de la roadmap et des différentes releases



MODALITÉS D'ÉVALUATION

Qualité des productions (8 points)

- **Qualité & structure du code** au sein du répo git
(tests unitaires, docstring, type hinting, gestion des dépendances, conventions de nommage, README, découpage en sous-modules)
- **Un rapport complet et soigné**
(logigramme, présentation de la base de données, stats descriptives, data dictionnaire, data préparation, roadmap, limites & ouvertures, biblio, qualité rédactionnelle & uniformité)
- Qualité **scientifique** des productions statistiques
- Un **produit qui fonctionne**, simple à prendre en main, avec un parcours fluide et avec des fonctionnalités analytiques pertinentes au regard du besoin client

Prise en main des outils de gestion & de suivi AGILE (6 points)

- La Roadmap construite détaille clairement les fonctionnalités livrées à chaque sprint & intègre les feedbacks
- Les objectifs de sprints respectent les standards **SMART** et les **DoD** permettent une mesure des écarts **transparente**
- Des livraisons en phase avec les attentes à chaque sprint & des Reviews complètes qui rendent **visible** l'ensemble du travail effectué ainsi que la progression sur la Roadmap

Implication individuelle & collective (6 points)

- **Contributions individuelles** sur la codebase disponible sur **git**
- **Implication** dans le sondage de fin de module & dans les feedbacks donnés pour améliorer le module
- **Curiosité & Proactivité** dans la recherche de feedbacks pour améliorer le produit
- Appréciation générale quant à la **dynamique collective**

JOUR DE REVIEW

Chaque jour de démo, vous devrez envoyer par mail à 10h au plus tard

Support écrit avec les nouveautés de la quinzaine (*dossier final en construction*)

Support de **démo** de la quinzaine

Lien du projet **git**

 morgan27cousin@gmail.com

Les **objectifs** & la **DoD** du prochain sprint devront, eux, être envoyés le **VENDREDI** après la démo (à 18h au plus tard)

Rappel du sommaire d'une démo client



Rappel du contexte et du **besoin** client

Rappel des **objectifs de sprint** et analyse de l'atteinte des objectifs

Les **faits marquants** de l'itération

Présentation des **nouveautés** de la quinzaine

Récupération des **feedbacks** client

Point sur la **roadmap** et présentation des objectifs du prochain sprint

SPRINT #1

Objectifs de sprint



- Concevoir un **produit analytics** en réponse à un besoin client identifié
- Construire la **roadmap** produit découpant les étapes en différents lots de valeur
- Trouver un premier **jeu de données** permettant de traiter la problématique
- Valider la prise en main du **setup de développeur**

Definition of Done



1. Un logigramme est construit permettant de comprendre l'ensemble des étapes potentielles de l'algorithme. Les erreurs potentielles sont intégrées au schéma.
2. Une roadmap produit détaillée par responsabilités est construite & découpée dans le temps en différents lots de valeur. A chaque étape de la Roadmap, le programme pourra être exécuté.
3. Un jeu de données est constitué et documenté au sein du rapport. La provenance des données et les définitions des champs de la table sont détaillées.
4. Un premier script python executable est disponible au sein du projet github validant l'autonomie de l'équipe dans l'utilisation des outils de développements

PLAN DES SÉANCES



Sprint Reviews

Jeudi 2025-10-23 (18h00-20h00): Review 1

Jeudi 2025-11-06 (18h00-20h00): Review 2

Jeudi 2025-11-20 (18h00-20h00): Review 3

Jeudi 2025-12-04 (18h00-20h00): Review 4

Jeudi 2025-12-18 (18h00-20h00): Review 5



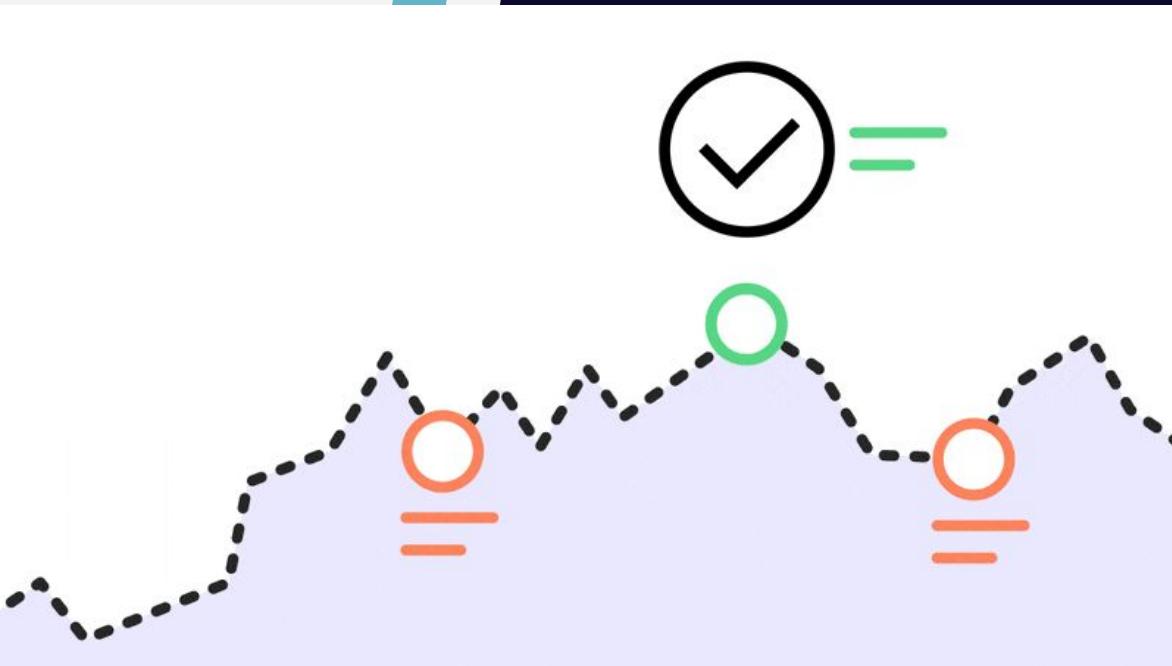
Soutenances

Lundi 2026-01-05 (17h30-20h00): Soutenances (date provisoire)

QUESTIONS



PRODUIT & DIGITAL & DATA



SIMPLE, BASIQUE

Qu'est-ce qu'un produit ?



Un vecteur de valeur en tant que réponse à un besoin identifié

Une production démarre toujours en réponse à un besoin !

Il peut être matériel ou immatériel

D'où vient le besoin ?



Le besoin naît d'un manque

Le manque vient de la connaissance



Peut-on créer le besoin ?



Un besoin ne se créé pas de façon ad hoc, il se révèle !

Comment ça ? Qui révèle quoi ?



La conception d'un produit ne peut pas se faire sans le client

Questionner le pourquoi de sa demande, révéler la « root cause », le problème à résoudre

LES GRANDS AXES D'UN PRODUIT DIGITAL



Solidité

L'application est-elle souvent sujette à des bugs ?



Ergonomie

Le parcours utilisateur est-il simple, rapide et fluide ?



Evolutivité

L'application s'adapte-t-elle à son écosystème technique et fonctionnel ?



Accessibilité

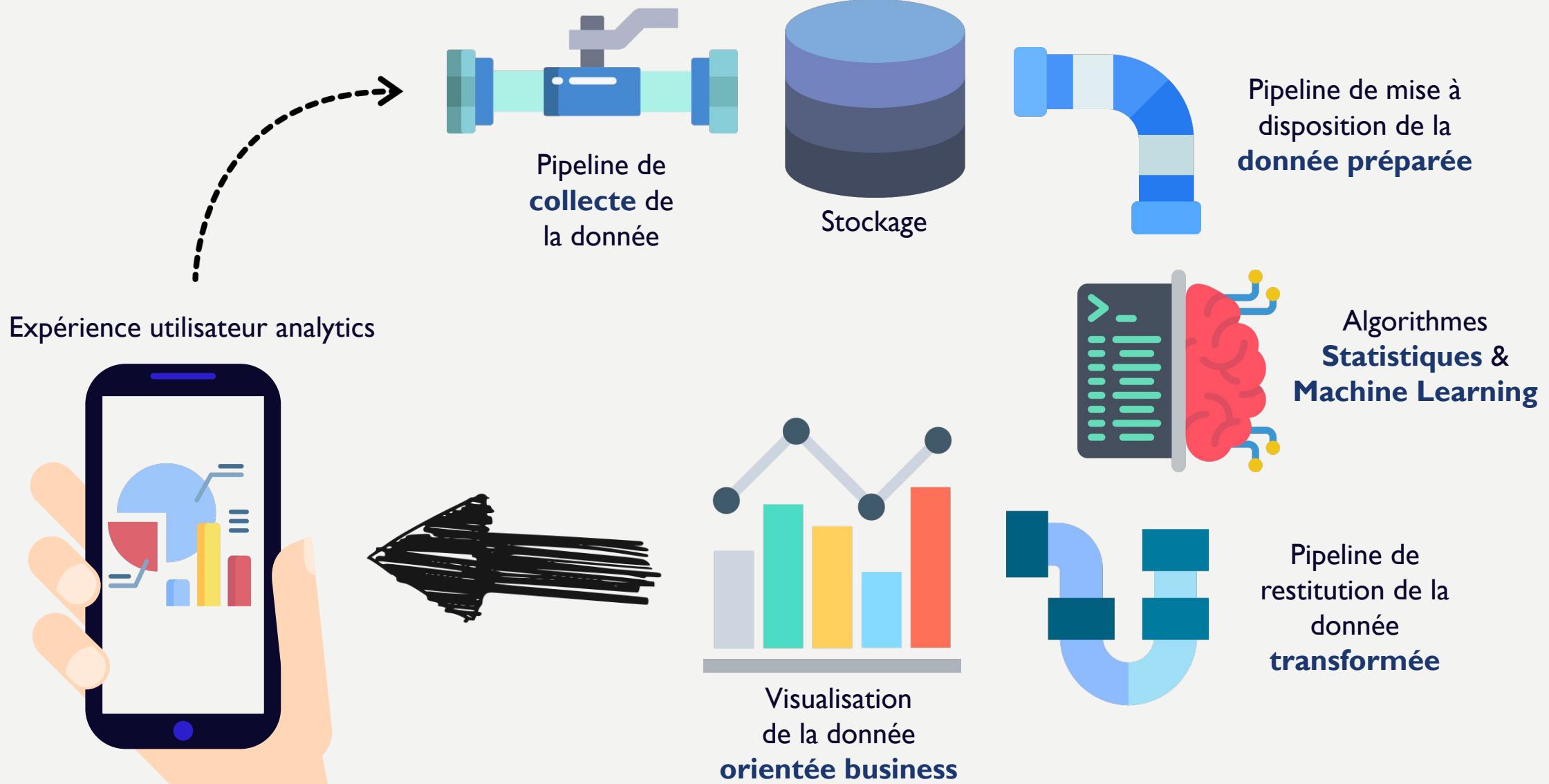
L'application est-elle compatible avec différents supports ?



Sécurité

Les données contenues au sein de l'application sont-elles suffisamment protégées ?

LA DATA EN APPLICATION



LE CYCLE DE LA DATA ANALYTICS



La data analytics n'est pas une recette linéaire !

Chaque étape du process est susceptible de remettre en question la précédente



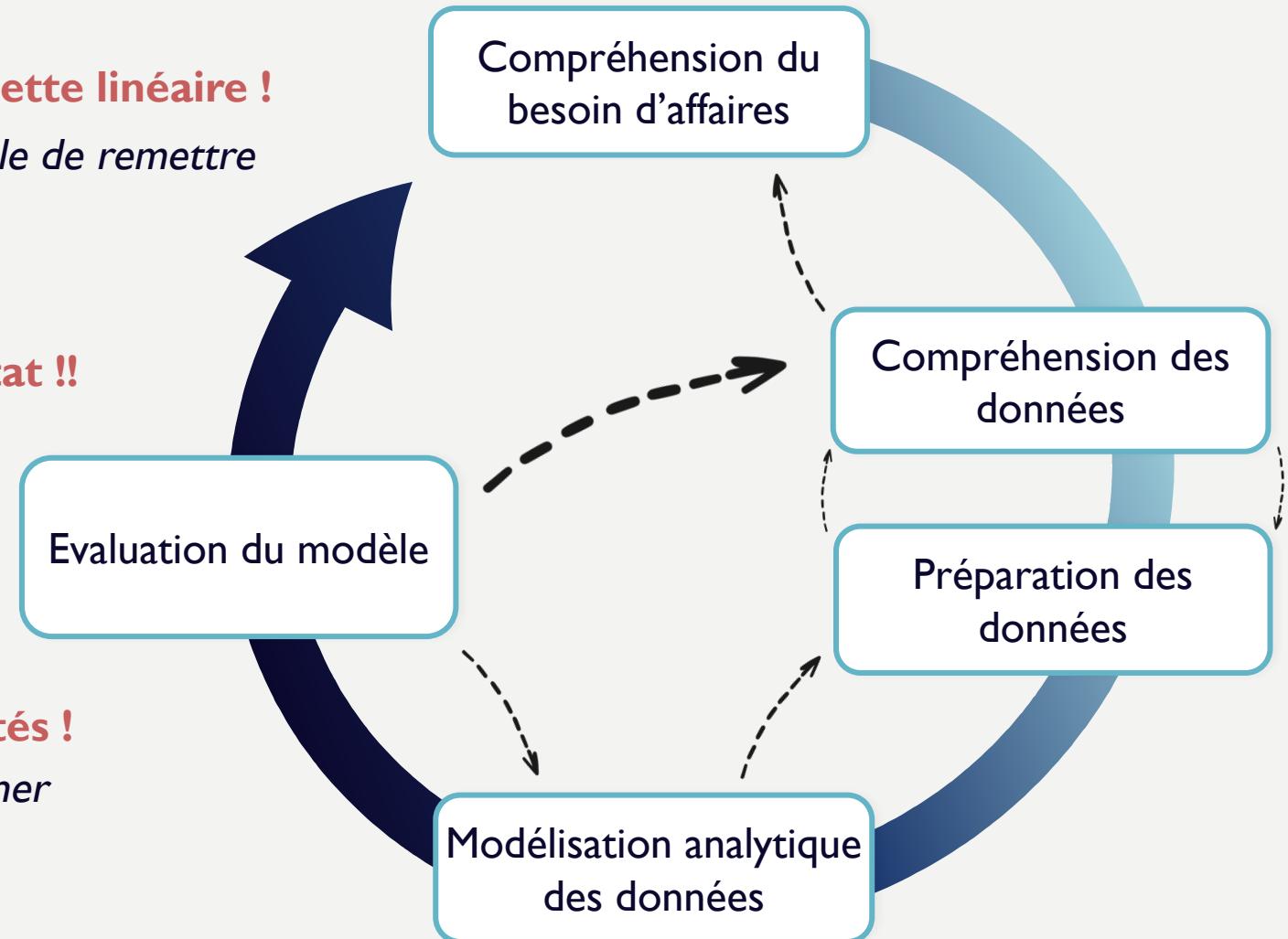
L'absence de résultat est un résultat !!

Ayez toujours en tête ce possible résultat, prévoyez des hypothèses alternatives et rechallengez régulièrement vos hypothèses

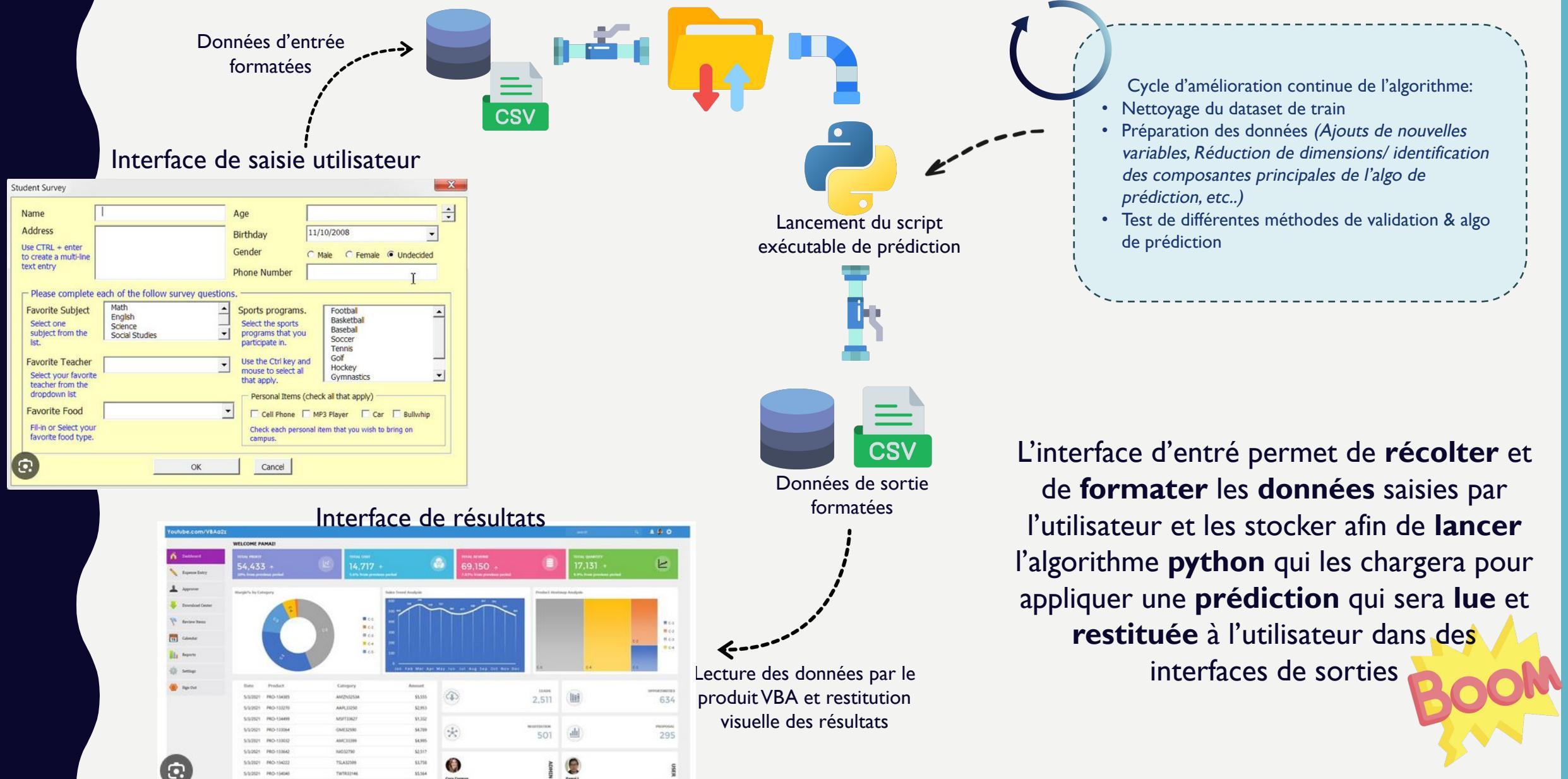


Les croyances ne sont pas des vérités !

L'anti-pattern par excellence est de mener une analyse en cherchant à tout prix à démontrer ce que le business demande

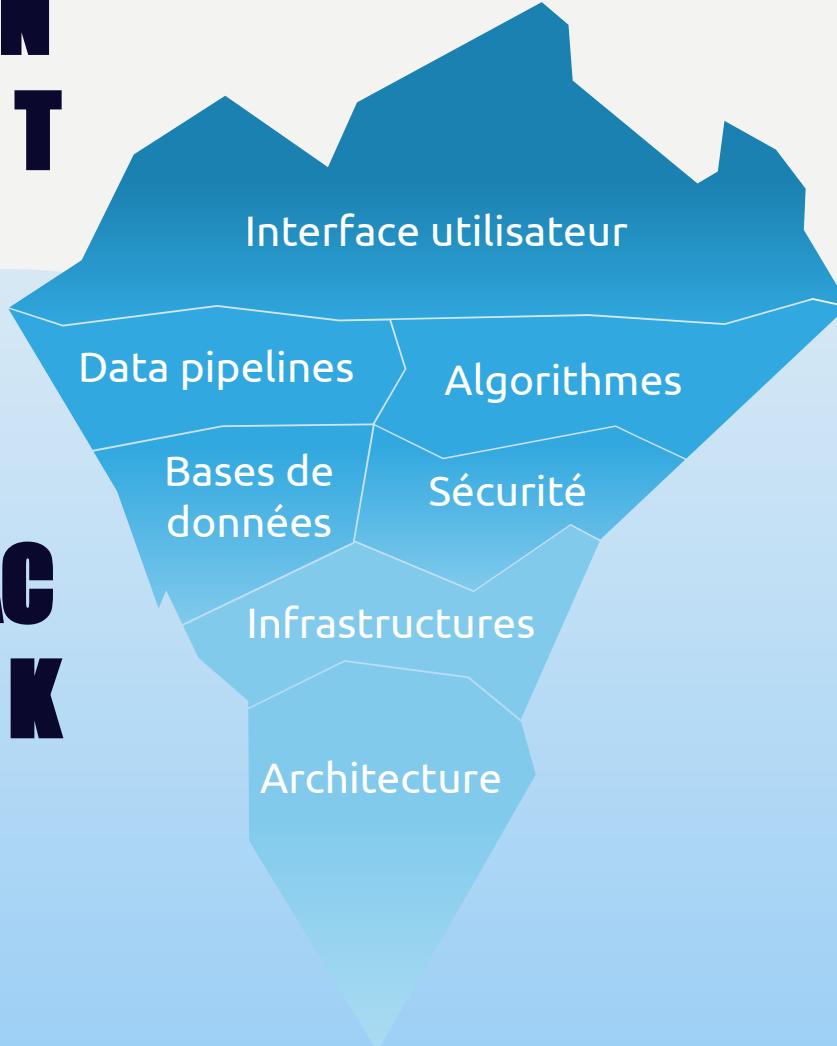


EXEMPLE DE PROTOTYPAGE SIMPLE



LA FACE CACHÉE DE L'ICEBERG

FRON
T



Tout ce que l'**utilisateur voit**
Tout ce qui permet à l'utilisateur d'interagir
avec le produit

Tout ce que l'**utilisateur ne voit pas**
Tout ce qui permet à l'application de
fonctionner et de fournir les services à
l'utilisateur

1 PRODUIT = DES EXPERTISES

Autour d'un logiciel, il n'y a pas une seule famille de métiers mais une **multitude** (*souvent éclipsée par l'écran*)

Front-end developers: La partie visible de l'application, l'interface avec laquelle l'humain va interagir

Back-end developers: Mettre en place l'architecture & communiquer avec le front (*architecture, calculs, règles de gestion, api, etc..*)

Data Integration: Construction des bases de données, des normes de stockage et mise en place des pipelines de données massifs: Extraction, Transformation & Mise à disposition (*ETL*)

Data Analytics: Valorisation des données par des calculs plus complexes

Data Visualization: Restitution visuelles des données pour une meilleure appropriation métier

Data Architect & Data Governance: Architecture & Gouvernance de données

DevOps: Permettre la rapidité et la solidité des développements et des contributions permanentes

UX/UI designers: Améliorer l'expérience l'utilisateur centré sur son utilisation de l'appli

TROUVER SA PLACE DANS LA DATA

Vous touchez un peu à tout et avez le sens de la débrouille. Vous aimez fouiller la data pour répondre à des besoins clients et vous savez adapter vos communications à vos interlocuteurs. Enfin, vous tirez une grande satisfaction de voir des décisions prises en vous appuyant sur vos rapport d'analyses.

Vous êtes passionné(e)s par la recherche appliquée, les articles scientifiques sont des sources inépuisables d'idées et vous aimez mener vos expériences sur machines.

Vous êtes persuadés que la data n'a aucune valeur tant qu'elle n'est pas restituée correctement au client. Pour vous la prise de décision n'est rien sans une valorisation visuelle simple et efficace.

Vous êtes convaincu(e)s que l'algorithme d'IA le plus sophistiqué n'est rien tant qu'il n'est pas correctement intégré à la machine. Votre priorité c'est avant tout une application analytics qui repose sur des fondations solides. Pour vous le code c'est de l'art.

Votre priorité est de maîtriser la donnée, de savoir d'où elle vient et comment elle est transformée. Vous en avez marre des études qui ne veulent rien dire parce que personne ne peut vous dire d'où viennent les données ni comment elles ont été récoltées !

Vous souhaitez mettre la data dans les mains des utilisateurs. Les besoins client sont synonymes de nouvelles fonctionnalités, de fluidité et d'ergonomie. Pour ce faire, vous adorez coordonner des équipes multi-compétences autour d'une vision commune pour créer la prochaine application analytics phare.

?

Data Analyst

?

**Data Scientist,
Research & Applied Scientist**

?

Dataviz/BI Developer

?

Data Engineer, Data Ops, MLOps

data engineers vs data scientists



?

Data Governance

?

Product Owner Data

QUESTIONS





L'ÉQUIPE PRODUIT

L'EQUIPE PRODUIT, ÇA RESSEMBLE À QUOI ?

Les parties prenantes



Product Owner

L'équipe Scrum



Scrum Master



Développeurs



ScrumFact

Il est généralement recommandé de ne pas dépasser les 9 personnes

Les équipes de 7 sont souvent privilégiées (*idéal pour se la jouer « 2 pizzas-team » à la Amazon*)



MON SPRINT TYPE

Semaine 1					Semaine 2						
Lundi	Mardi	Mercredi	Jeudi	Vendredi	Lundi	Mardi	Mercredi	Jeudi	Vendredi		
Sprint Review (SM)	Daily Meeting (ALL)										
Retro (SM)											
Sprint Planning (PO)	Grooming (PO)					Grooming (PO)	Deploy PROD & Demo Setup (ALL)				

LA PLANIFICATION DE SPRINT



Pourquoi ???

Définir l'objectif du sprint aka **c'est quoi LE truc qui doit absolument être fait !**

En définissant un objectif à atteindre, vous donnez du sens à votre sprint



Comment faire ?

1. Rappeler la localisation du sprint sur la **roadmap** et la **release** associée
2. Faire le tour des **capacités** & disponibilités de chacun
3. Statuer sur la **Definition of Done** du sprint
4. Lister les **tâches à réaliser** durant le sprint pour atteindre cette DoD
5. Évaluer collectivement la **complexité** des différentes tâches
6. Se **répartir** les tâches entre développeurs dans la limite de la capacité définie en amont
7. Adapter la DoD en **arbitrant** éventuellement sur ce qui ne peut pas être embarqué



Combien de temps ?

Autant de temps qu'il faut pour lever les inconnues et être serein pour livrer l'objectif durant le sprint à venir

En général, **2h** suffisent à planifier un sprint de 2 semaines pour une équipe de 7

Quand ? 1 fois par sprint: le premier jour du sprint

LE DAILY SCRUM MEETING



Pourquoi ???

S'assurer que l'objectif n'est pas mis à risque et si des bloqueurs apparaissent, mettre immédiatement en place les plans d'actions pour les résoudre

Moment pour l'équipe ! Synchronisation et communication: ce n'est pas un reporting !

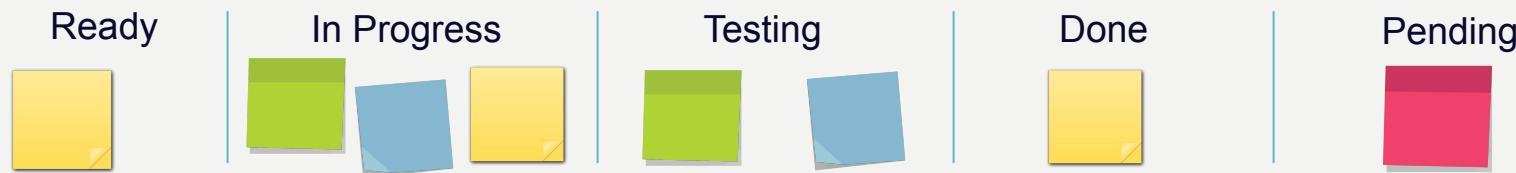


Comment faire ?

Souvent le daily se construit autour de 3 questions auxquelles chaque développeur répond:

1. Qu'est-ce que j'ai fait hier qui a aidé l'équipe à atteindre **l'objectif du Sprint** ?
2. Que ferai-je aujourd'hui pour aider l'équipe à atteindre **l'objectif du Sprint** ?
3. Est-ce que je vois des obstacles qui empêchent l'équipe de respecter **l'objectif du Sprint** ?

Il s'accompagne souvent d'un **board de sprint** (vous pouvez utiliser [miro](#))



Combien de temps ?

Le Scrum Guide est assez précis sur la timebox du daily scrum meeting: **15min maximum**

Quand ? Tous les jours (en début de journée idéalement)

LA REVUE DE SPRINT



Pourquoi ???

Donner de la visibilité aux parties prenantes sur l'atteinte de l'objectif de sprint **#Transparence**
Récolter des Feedbacks sur l'incrément livré et ajouter de la valeur au produit ! **#Adaptation**

RENDRE VISIBLE L'INVISIBLE



FAIL FAST = LEARN FAST !



Comment faire ?

1. Rappel du contexte et du **besoin** client
2. Rappel des **objectifs de sprint** et analyse de l'atteinte des objectifs
3. Les **faits marquants** de l'itération (*problèmes rencontrés/résolus*)
4. Présentation des **nouveautés** de la quinzaine
5. Récupération des **feedbacks** client
6. Point sur la **roadmap** et partage de la vision autour du prochain sprint

Pour éviter le « bug » le jour de démo, nous enregistrons souvent nos démos avec [loom](#) ⚡



Combien de temps ?

En général, **1h** suffit à restituer le contenu du sprint, faire les démos et récolter le feedback client
Quand ? 1 fois par sprint: le dernier jour du sprint

LA RÉTROSPECTIVE DE SPRINT



Pourquoi ???

Travailler collectivement à rendre l'équipe plus efficace pour livrer dans les temps des incrémentums de valeurs adaptés au besoin client

*Il s'agit vraiment d'un moment à part, dédié à l'équipe pour parler en toute transparence et bienveillance
Et même quand tout se passe bien, il est toujours possible d'identifier un axe aurait permis de faire mieux !*



Comment faire ?

Différents formats existent au sein de la communauté Agile mais tous cherchent à identifier:

- Les *likes* (*ce que les développeurs ont aimé lors du sprint*)
- Les forces/points positifs du sprint (*ce qui a bien fonctionné*)
- Les problèmes rencontrés lors du sprint (*ce qui n'a pas fonctionné ou qui a ralenti l'équipe*)
- Les risques à venir par rapport aux objectifs de roadmap (*ce qui pourrait menacer le produit*)

A l'issue de l'exercice un **plan d'améliorations** est mis en place. Il sera suivi par le Scrum Master

Des boards de rétrospective sont disponibles dans [miro](#)



Combien de temps ?

En général, **1h** suffit pour brainstormer, partager son ressenti et définir les plans d'actions

Quand ? 1 fois par sprint: le dernier jour du sprint

SCRUM IN A NUTSHELL

SCRUM PROCESS





LES OUTILS DE L'ÉQUIPE PRODUIT

WELCOME TO THE MACHINE !



THINK before ACT

En algorithmie, avant de partir tête baissée et d'écrire une tonne de ligne de code, il est indispensable de passer par la phase de **CONCEPTION**

Cette étape permet de schématiser l'ensemble des processus et ainsi mieux anticiper les cas d'erreur !



Construisez votre logigramme

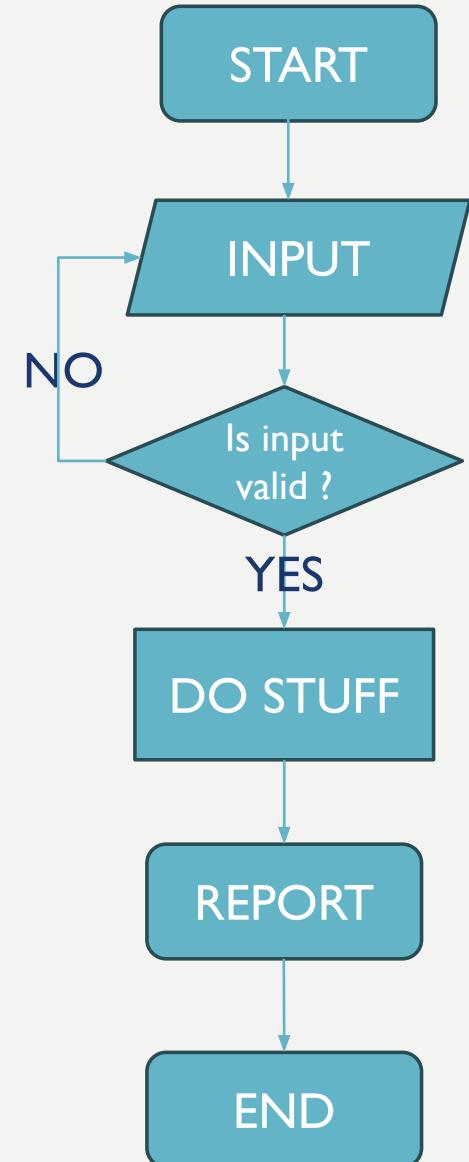
Le **logigramme** est un outil très fréquemment utilisé car il est simple à réaliser et il permet de visualiser de façon séquentielle et logique l'ensemble des actions réalisées par le programme.

Il s'agit également d'un élément de documentation parfait pour expliquer le comportement de votre algorithme à toute personne extérieure à votre code !



Comment fait-on ?

Il existe de nombreux outils disponibles en ligne comme draw.io ou whimsical, plus ou moins gratuits !



Exemple de logigramme générique

EXEMPLE DE ROADMAP MATRICIELLE



Octobre

Novembre

Décembre

Interfaces Utilisateur

Release I



Release II



Release III



Release IV



Release V



Data pipelines

Release I



Release II



Release III



Release IV



Release V



Algo de prédition

Release I



Release II



Release III



Release IV



Release V



CONSTRUIRE UN OBJECTIF

EXEMPLE

Construire un jeu de données

Commencer l'écriture du rapport

Se documenter sur les méthodes

Faire l'analyse du jeu de données

Préparer la présentation

Faire le tour des projets existants sur Kaggle



1. Impossible de se projeter sur le **rendu attendu**
2. Impossible de **mesurer précisément** l'atteinte de l'objectif
3. Impossible de savoir le **temps et les ressources nécessaires**
4. Impossible de délimiter la **faisabilité** de l'objectif
5. Impossible de définir l'**échéance** de livraison

Exemple d'un objectif de sprint pour un produit analytics

En tant que propriétaire de plusieurs appartements et utilisateur de l'application SmartEnergy, je peux choisir sélectionner un de mes appartements pour lancer un apprentissage de son profil de consommation énergétique journalier en fonction des données de températures externes afin d'avoir une prédition de ma consommation de la semaine à venir. Cette prévision me sera restituée dans une interface graphique en incluant un intervalle de confiance de 5% autour de la prédition.

BE S.M.A.R.T.



Spécifique: Votre objectif doit être clairement défini, simple et compréhensible de tous



Mesurable: Un seuil quantitatif est disponible et permet de conclure de façon binaire sur l'atteinte de votre objectif



Atteignable: Votre objectif doit être cohérent au regard de votre temps et de vos compétences disponibles



Réaliste: Ne visez pas la lune ! Soyez ambitieux mais fixez vous des objectifs que vous pourrez tenir et non un défi qui vous démotivera et que vous abandonnerez



Temporellement défini: Votre objectif doit être délimité dans le temps avec une échéance précise

STRUCTURER SON CODE PYTHON



THINK BEFORE ACT



*“Any fool can write code that a computer can understand.
Good programmers write code that humans can understand.” — Martin Fowler*

Avant de foncer tête baissée sur du code, il est vraiment conseillé de se poser pour établir la gouvernance technique du projet:

- **Etablir un process de contribution clair (gitlab, trunk-base/feature-branch)**
- **Organiser son projet de façon**
- **Gérer ses dépendances**
- **Structurer, Découper, Nommer correctement et surtout Tester ses fonctions**
- **Appliquer des contrôles de qualité de code automatisés**
- **Rendre son environnement de travail reproduitible**



Bienvenus dans le monde du Génie Logiciel ! a.k.a software engineering !



La grande majorité des projets data qui échouent ont négligé ces composantes de gouvernance et de mise en production, souvent par une méconnaissance de ses enjeux et donc par un contexte IT qui ne permet pas d'accompagner l'ambition data du business

VERSION CONTROL

Il était une fois git (2005)



Répertoire (*repo*) de travail décentralisé (*remote*) qui vous permettra de stocker vos projets et de suivre facilement les contributions individuelles locales

Outil de **versioning** qui vous permet de

- Savoir qui a modifié le code, quand et comment ? (*vous pouvez voir les différences par rapport à la version précédente*)
- Relire facilement et valider la fusion de la *branch candidate* avec la *branch principale* du produit
- **Code Review** : « *Seul on va vite, à plusieurs on va loin !* » (*bénéficiez des retours sur vos codes par des Lead Devs expérimentés*)
- Faire facilement marche arrière en restaurant une version du code précédente



github

Hub qui vous permettra d'héberger vos projets (*public & private mode*)

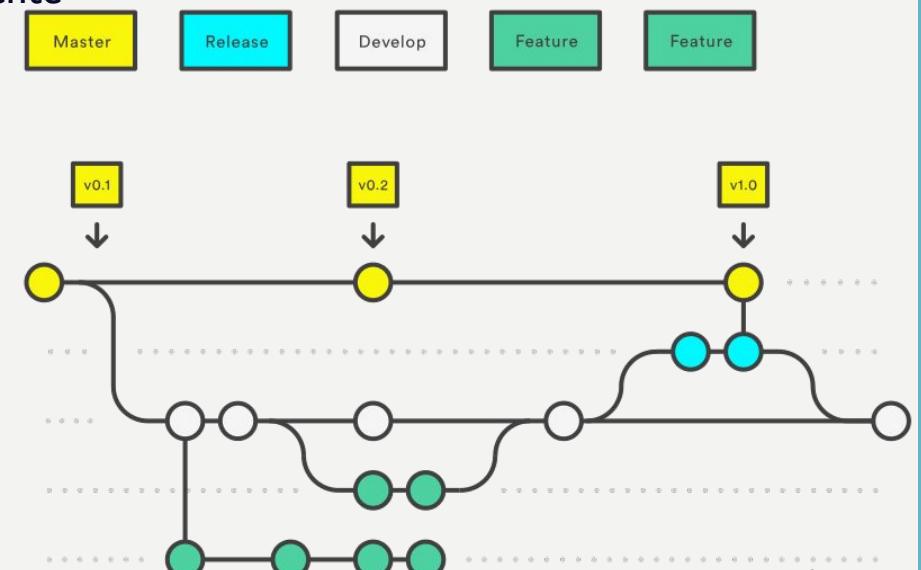


gitlab

Plateforme dédiée au déploiement & à l'intégration continue des logiciels

Lexique Top 5

- **branch**: Historique de développement
- **commit**: Enregistre les modifications dans le dépôt
- **push**: Met à jour la *remote* avec les différences locales enregistrées
- **merge**: Fusionne 2 ou + historique de développement
- **pull**: Cherche les diffs d'un *repo* en *remote* et les intègre sur votre version locale



Exemple de giflow

SRC LAYOUT vs FLAT LAYOUT

src layout

Structure avec un dossier `src/` qui contient le code source

```
iac-microservices-alerts/
├── src/
│   └── alerts/
│       ├── __init__.py
│       └── ...
└── tests/
    └── pyproject.toml / setup.py
```

✓ Avantages

- Les imports `from alerts import ...` ne peuvent pas fonctionner sans l'installation du module. Offre une sécurité supplémentaire.
- Recommandé pour le packaging et la distribution car le code source est isolé des outils externes
- Permet de construire plusieurs sous-packages si besoin de redécouper les responsabilités

✗ Limites

- Ajoute une couche supplémentaire, moins immédiat pour lire

flat layout

Structure où le code source est à la racine du projet

```
iac-microservices-alerts/
├── alerts/
│   ├── __init__.py
│   └── ...
└── tests/
    └── pyproject.toml / setup.py
```

✓ Avantages

- Structure plus direct et assez simple pour démarrer un projet ou un petit outil

✗ Limites

- Les imports fonctionnent sans installation du module, ce qui peut masquer de potentiels problèmes d'import
- Le code source est au même niveau que les outils externes (`Dockerfile`, `gitlab-ci`, etc...)
- Pas modulable: 1 seul module

GESTION DES DÉPENDANCES

3 principaux package managers pour les projets python

Artisanale

- **pip**: utilisation d'un **requirements.txt** à la racine du projet, souvent généré par un **pip freeze**

Industrielle (*Nécessite une installation*)

- **poetry**
- **uv** (*projet open-source codé en Rust, développé par Astral, ceux qui ont fait le linter ruff*)

Les gros avantages de **poetry** et **uv**:

- Gestion centralisée via un **pyproject.toml**, fournissant un **.lock file**
- Gestion des packages avec **packages = [{ include = "alerts", from = "src" }]**
- Environnement virtuel intégré et automatiquement détecté par Docker sans configuration supplémentaire
- Possibilité de centraliser les fichiers **.ini** dans le **pyproject.toml**
- Temps de résolution des dépendances sur un mini projet perso: **pip=88s | poetry=45s | uv=20s** (*pour un install des dépendances sur un mini projet perso*)
- A priori les images Docker sont moins lourdes avec **uv** qu'avec **poetry**
- Possibilité de facilement passer de **poetry** à **uv** à l'avenir donc pas d'inquiétude si besoin de changer quand **uv** sera plus mature

LES TESTS UNITAIRES



Les tests quoi ?

- Procédure automatique vérifiant qu'une **partie précise** du code **fonctionne correctement**
- Lorsque vous codez, vous avez la **responsabilité** de garantir que vos transformations fournissent toujours les bons résultats

✓ `assert days_in_year(year=2019) = 365`
✓ `assert days_in_year(year=2100) = 365`
✓ `assert days_in_year(year='Test') is None`

✓ `assert days_in_year(year=2020) = 366`
✓ `assert days_in_year(year=2000) = 366`
✓ `assert days_in_year(year=-4) = 366`

Ceci est un tableau de cas de test !!!

En somme, les test unitaires sont:

 Des gages de **confiance** pour les utilisateurs: le code est testé pour garantir sa **stabilité** aux utilisateurs
De la **documentation** pour toute personne se demandant ce que renvoie la fonction selon son utilisation

Dans le monde industriel:

 Les tests sont **intégrés aux protocoles de déploiement** et sont lancés systématiquement à chaque **révision** de modification avant toute activation au grand public !
Les tests sont écrits avant la fonction à implémenter: on parle de **Test-Driven Development** 

STRUCTURER SES TESTS UNITAIRES

pytest est un framework de test python qui complète **unittest** (*built-in*)

- Test Discovery basé sur les noms de fichiers et de fonctions
- Fixtures à scopes variables, réutilisables et paramétrables en dehors des class via des `@pytest.fixture` automatiquement chargeables depuis le `conftest.py`
- Nombreuses extensions: `pytest-cov`, `pytest-mock`, `pytest-asyncio`

Conventions pour bien structurer ses tests unitaires

- Structure du dossier tests en miroir par rapport au module applicatif
- 1 fonction = 1 class de test contenant 1 méthode de test par cas de test
- Les tests unitaires doivent pouvoir être exécutés dans n'importe quel ordre. Un test ne doit pas dépendre de l'état de la session de test résultant de l'exécution d'un autre test
- Les tests unitaires ne dépendent pas de services externes (*ex: base de données de dev*), ces derniers sont mockés
- Tester uniquement la valeur ajoutée : On ne re-teste pas une fonction appelée dans notre fonction, on la considère déjà couverte

The image shows a file explorer interface with the following structure:

- src**:
 - app**
 - tools**:
 - cleaning.py**

```
def clean_series():
    pass
```
 - ...
- tests**:
 - test_app**
 - test_tools**:
 - test_cleaning.py**

```
class TestCleanSeries:

    def test_case_empty(self):
        assert ...

    def test_full_series(self):
        assert ...
```

Contrôle de code quality (p1)

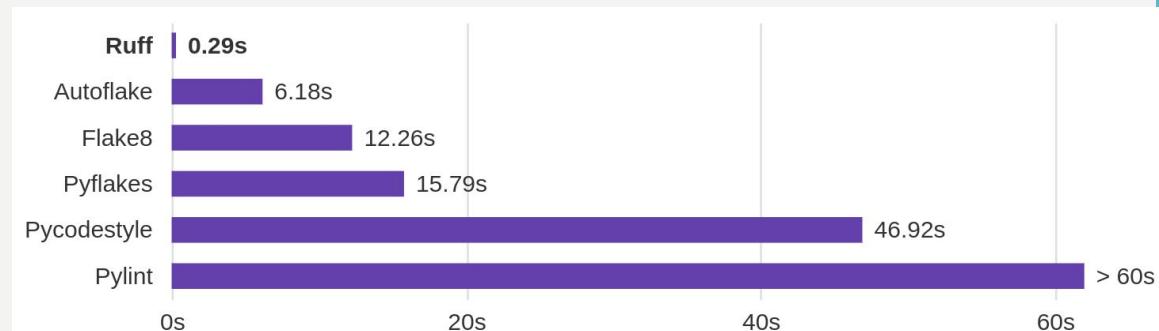
- **ruff** est un code formatter codé en **Rust** par Astral qui se veut 10 à 100x plus rapide que les autres linters python disponibles

- **Utilisation des configurations par défaut de ruff**

Par exemple:

- line-length = **88**
- indent-width = **4**
- indent-style = "**space**"
- quote-style = "**double**"

(default identiques à *Black*)



Linting the CPython codebase from scratch.

Ajout de sécurités pour garantir la conformité du projet aux critères de ruff :

- Mise en place de **pre-commit**: Validation de la conformité du code avant de figer les diffs dans un commit.
Configuration pre-commit faite avec ruff (*cf pre-commit-config.yaml*)
- Activation de l'**autoSaveFormat** de **VSCode** sur la configuration **ruff** pour qu'à chaque enregistrement des fichiers le linter soit appliqué automatiquement

Contrôle de code quality (p2)

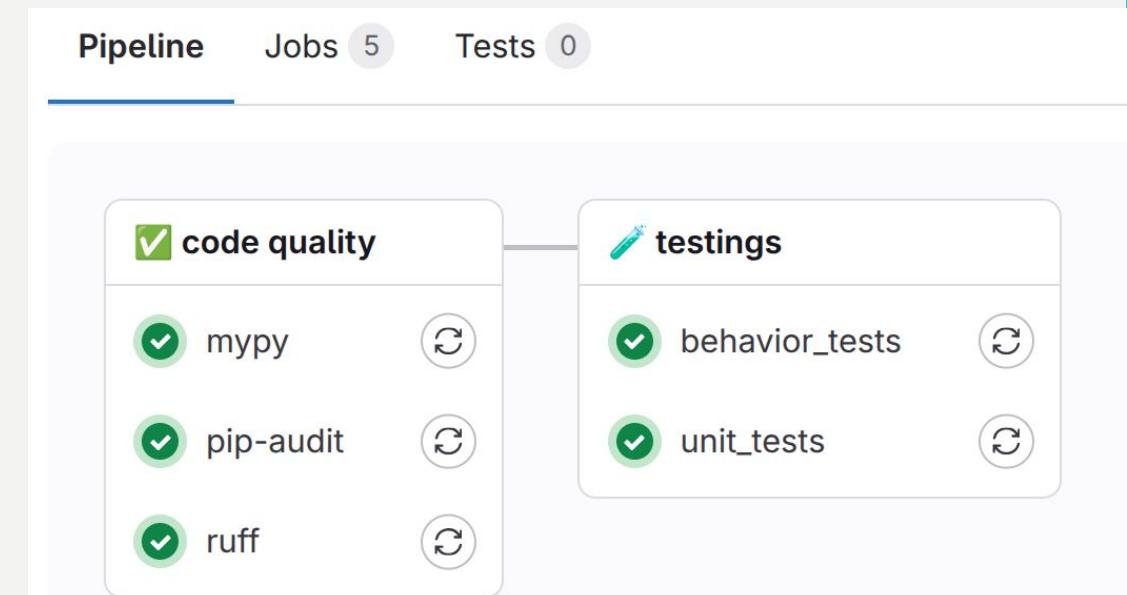
Code quality

- [mypy](#) : Type checking
- [pip-audit](#) : Scan Python packages vulnerabilities
- [ruff](#) : Code Formatter

Testings

- [behavior_testing](#): Tests de comportement de bout en bout avec le module [behave](#)
- [unit_testing](#): Tests unitaires avec le module [pytest](#)

Définition des étapes dans le [gitlab-ci.yml](#)



ENVIRONNEMENT DE TRAVAIL REPRODUCTIBLE

Mise en place d'un dossier `.devcontainer`

- **Dockerfile** : fichier de configuration décrivant comment construire l'environnement d'exécution de l'application dans l'image Docker
- `docker-compose.yml` : permet de définir et lancer plusieurs conteneurs en même temps, avec leurs liens, réseaux, volumes et les variables d'environnement
- **devcontainer.json** : fichier utilisé par **VSCODE** pour configurer automatiquement un conteneur de développement

✓ Avantages

- Environnement de travail **100 % reproductible et identique** pour tous les développeurs: Adieu le classique "*ça marche sur ma machine*" même dans une équipe hétérogène (*os différentes, projets multiples sur des versions de python différentes, etc...*). L'environnement n'est plus fonction des configurations individuelles de son environnement local.
- Permet de travailler à l'intérieur d'un container que l'on souhaite au plus proche de l'image qui sera déployée en production
- **Virtualisation légère:** Une fois votre travail terminé, vous pouvez détruire le container et libérer les ressources de votre ordinateur contrairement aux installations en dur qui provisionnent des ressources qui ne sont plus disponibles pour d'autres usages même si vous ne vous servez pas de l'application

✗ Limites

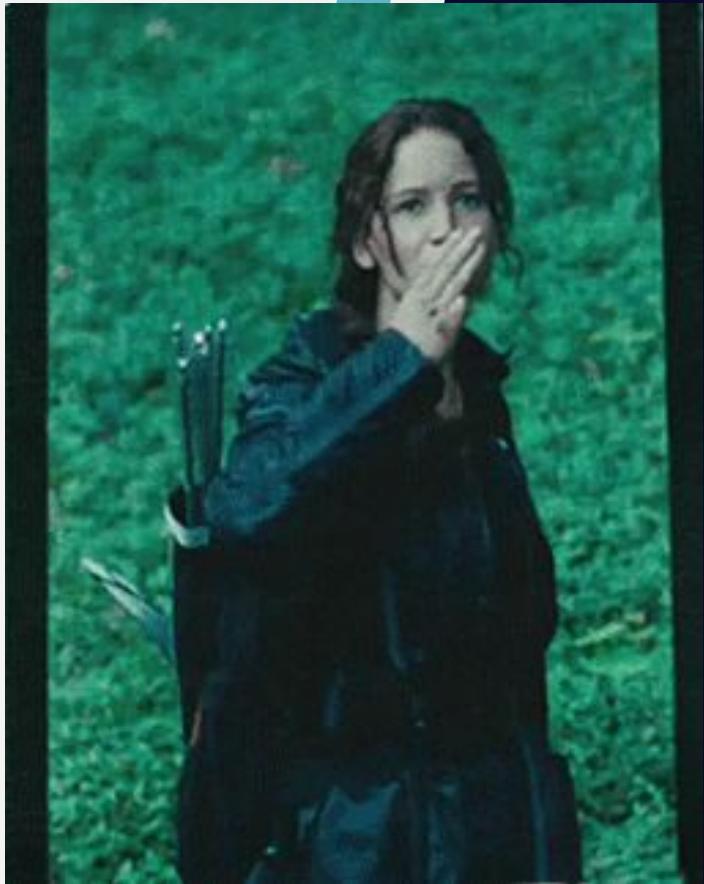
- Peut sembler plus complexe notamment pour la mise en place du Dockerfile et des services via compose
- Le temps de build : même si c'est un peu un "faux" argument au regard des 2/3 minutes d'installation, fait uniquement au moment d'installer les services (*bien de rebuild uniquement si vous toucher aux configurations du container*)

QUESTIONS



QUESTIONS





**EN ROUTE VERS LE
PROJET!**

PUISSE LE SORT VOUS ÊTRE FAVORABLE !

Vous réaliserez ce projet en équipes:

- 27 étudiants : 16 APE + 11 Maths répartis en **6 équipes de 4 à 5 étudiants**
- Les groupes seront **formés librement** à condition de respecter le ratio suivant:
 - **2/3 profils ECO + 1/2 profils Maths**
- Les différentes équipes ont **48h** pour me fournir le lien du répertoire de **travail github ou gitlab** (*1 mail par groupe suffira*)
 - **ATTENTION: en cas d'utilisation du gitlab de l'URCA, il est impératif de rendre le répo public (ouvert à tous) dans les paramètres du répo !**
- Les 5 prochains cours seront au format **Sprint Review** i.e des soutenances de présentation des avancées intermédiaires soumis à évaluation ! (*les consignes sont à retrouver slide 9*)
- A chaque Sprint Review, plusieurs groupes seront choisis pour présenter leurs avancés, me permettant ainsi de vous donner des feedbacks pour améliorer les roadmaps et produits
- Les objectifs du premier sprint sont à retrouver en slide 10

QUESTIONS



THANK YOU

