Pourquoi utiliser Python pour des projets data

Pourquoi choisir Python pour les projets data ?

Python est devenu le langage de référence dans le domaine de la data, que ce soit pour la data analysis, le machine learning, le traitement de données massives ou encore le déploiement d'APIs de données. Voici pourquoi.

Simplicité et lisibilité

- Python a une syntaxe claire, proche du langage naturel.
- Il est facile à lire, à apprendre et à maintenir, ce qui est crucial quand plusieurs personnes (scientifiques, analystes, ingénieurs) collaborent sur les mêmes scripts.

Écosystème riche et mature

- Python dispose d'une immense bibliothèque d'outils pour la data :
  - **NumPy** et **Pandas** : traitement et manipulation de données.
  - **Matplotlib**, **Seaborn**, **Plotly** : visualisation.
  - **Scikit-learn**, **XGBoost** : machine learning.
  - **TensorFlow**, **PyTorch** : deep learning.
  - **SQLAlchemy**, **Psycopg2**, **Pymongo** : connexion aux bases de données.
  - **Airflow**, **Luigi** : orchestration de pipelines.
  - **Dask**, **PySpark** : traitement distribué.

Interfaçage facile avec d'autres outils

- Python s'intègre facilement avec des bases de données, des APIs REST, des outils cloud, des scripts shell ou du code C/C++.
- Il est compatible avec tous les grands services cloud (AWS, Azure, GCP).

Un langage adapté à toutes les étapes de la data

- Du **nettoyage des données** à la **modélisation**, en passant par la **visualisation** et le **déploiement**, Python couvre l'ensemble de la chaîne de valeur data.
- Il est utilisé aussi bien par les **data analysts**, **data engineers** que par les **data scientists**.

Une grande communauté

- Des milliers de ressources, tutoriels, notebooks et projets open-source sont disponibles.
- Des réponses quasi instantanées sur StackOverflow, GitHub ou Medium.

Exemple : un petit projet Python de data exploration

Imaginons qu'on veuille charger des données CSV, les analyser, les nettoyer, puis les enregistrer dans une base PostgreSQL. Voici un exemple simple avec `pandas` et `sqlalchemy`.

Structure du projet

```
mon-projet-data/
│
├── app/
│   ├── main.py
│   └── requirements.txt
│
├── data/
│   └── ventes.csv
│
├── Dockerfile
└── docker-compose.yml
```

Contenu du `ventes.csv`

```csv
date,produit,quantite,prix_unitaire
2024-01-01,Stylo,10,1.5
2024-01-01,Cahier,5,2.0
...
```

Contenu du `main.py`

```python
import pandas as pd
from sqlalchemy import create_engine

# Chargement du CSV
df = pd.read_csv("data/ventes.csv")

# Nettoyage rapide
df.dropna(inplace=True)
df['total'] = df['quantite'] * df['prix_unitaire']

# Connexion à PostgreSQL via SQLAlchemy
engine = create_engine("postgresql://myuser:mypassword@db:5432/mydb")
df.to_sql("ventes", engine, if_exists="replace", index=False)

print("Données enregistrées avec succès.")

```

Contenu du requirements.txt

```
pandas
sqlalchemy
psycopg2-binary

```

Contenu du `Dockerfile`

```
FROM python:3.11-slim

WORKDIR /app

COPY app/requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

COPY app/ .
COPY data/ ../data/

CMD ["python", "main.py"]

```

Contenu du docker-compose.yml

```yaml
version: '3.9'

services:
  app:
    build: .
    depends_on:
      - db
    volumes:
      - ./data:/data
    networks:
      - backend

  db:
    image: postgres:15
    environment:
      POSTGRES_DB: mydb
      POSTGRES_USER: myuser
      POSTGRES_PASSWORD: mypassword
    volumes:
      - pgdata:/var/lib/postgresql/data
    networks:
      - backend

volumes:
  pgdata:

networks:
  backend:

```

Lancement du projet

```
[redacted] git:(master) docker-compose up --build
```

Conclusion

Python s’impose dans les projets data pour sa **simplicité**, sa **puissance**, et son **écosystème extrêmement riche**. Il est à la fois un outil d’exploration, d’analyse, de traitement, de modélisation et de déploiement. Il permet de transformer rapidement une idée en solution opérationnelle, même à grande échelle.