TP7

Introduction:

Structure du projet:

```
tp-dvc-mlflow/
│
├── params.yaml
├── dvc.yaml
├── pyproject.toml
├── src/
│   ├── data/
│   │   ├── download.py
│   │   ├── preprocess.py
│   │   └── split.py
│   ├── train/
│   │   └── train.py
│   └── infer/
│       └── predict.py
│
├── data/
│   ├── raw/
│   ├── processed/
│   └── splits/
│
├── models/
├── reports/
└── mlruns/
```

Dataset utilisé

Adult Income Dataset (UCI Machine Learning Repository)
Problème de classification binaire : prédire si le revenu annuel est >50K ou <=50K.

Sources :

https://archive.ics.uci.edu/ml/machine-learning-databases/adult/adult.data
https://archive.ics.uci.edu/ml/machine-learning-databases/adult/adult.names

Parameters globales **params.yaml**:

```yaml
seed: 42

dataset:
  url: "https://archive.ics.uci.edu/ml/machine-learning-databases/adult/adult.data"
  raw_filename: "adult.csv"

preprocess:
  drop_missing: true
  drop_duplicates: true
  normalize_numeric: true
  categorical_encoding: "onehot"

split:
  train_ratio: 0.7
  val_ratio: 0.15
  test_ratio: 0.15
  stratify: true
```

Pour Lecture des paramètres YAML:

```python
import yaml

with open("params.yaml", "r") as f:
    params = yaml.safe_load(f)

seed = params["seed"]
train_ratio = params["split"]["train_ratio"]
```

Étape 1 – Téléchargement des données

Script : **src/data/download.py**

- Télécharger le dataset depuis l'URL définie dans params.yaml
- Sauvegarder le fichier CSV dans data/raw/

code :

```
import pandas as pd

url = params["dataset"]["url"]
columns = [
   "age", "workclass", "fnlwgt", "education", "education_num",
   "marital_status", "occupation", "relationship", "race", "sex",
   "capital_gain", "capital_loss", "hours_per_week",
   "native_country", "income"
]

df = pd.read_csv(url, header=None, names=columns)
df.to_csv("data/raw/adult.csv", index=False)
```

Le fichier brut doit être suivi par DVC.

```
dvc add data/raw/adult.csv
git add data/raw/adult.csv.dvc
Git commit -m 'add raw dataset'
```

Étape 2 – Prétraitement et split

Script : **src/data/download.py**

Transformations:

- Remplacement de ? par NaN
- Suppression des doublons
- Suppression ou conservation des lignes incomplètes
- Normalisation des colonnes numériques

```
from __future__ import annotations
from pathlib import Path
```

```python
import json
import yaml
import numpy as np
import pandas as pd
from sklearn.preprocessing import StandardScaler

def load_params(path: str = "params.yaml") -> dict:
    with open(path, "r", encoding="utf-8") as f:
        return yaml.safe_load(f)
params = load_params()

raw_path = Path("data/raw") / params["dataset"]["raw_filename"]
out_dir = Path("data/processed")
out_dir.mkdir(parents=True, exist_ok=True)
out_path = out_dir / "adult_clean.csv"

df = pd.read_csv(raw_path)

df.replace(" ?", np.nan, inplace=True)

if params["preprocess"]["drop_duplicates"]:
    df = df.drop_duplicates()

if params["preprocess"]["drop_missing"]:
    df = df.dropna()

from sklearn.preprocessing import StandardScaler

num_cols = df.select_dtypes(include="number").columns
scaler = StandardScaler()
df[num_cols] = scaler.fit_transform(df[num_cols])

df.columns = [c.strip().lower().replace(" ", "_") for c in df.columns]

df.to_csv(out_path, index=False)
```

Script : **src/data/split.py**

```python
params = load_params()
```

```
seed = int(params["seed"])
split_cfg = params["split"]
train_ratio = float(split_cfg["train_ratio"])
val_ratio = float(split_cfg["val_ratio"])
test_ratio = float(split_cfg["test_ratio"])
stratify = bool(split_cfg.get("stratify", True))

total = train_ratio + val_ratio + test_ratio
if abs(total - 1.0) > 1e-9:
    raise ValueError(f"Split ratios must sum to 1.0, got {total}")

in_path = Path("data/processed") / "adult_clean.csv"
df = pd.read_csv(in_path)

if "income" not in df.columns:
    raise ValueError("Target column 'income' not found. Check preprocessing output.")

X = df.drop(columns=["income"])
y = df["income"]

X_train, X_temp, y_train, y_temp = train_test_split(
    X,
    y,
    test_size=1.0 - train_ratio,
    random_state=seed,
    stratify=y if stratify else None,
)

# Split temp into val and test
val_ratio_adjusted = val_ratio / (val_ratio + test_ratio)
X_val, X_test, y_val, y_test = train_test_split(
    X_temp,
    y_temp,
    test_size=1.0 - val_ratio_adjusted,
    random_state=seed,
    stratify=y_temp if stratify else None,
)

out_dir = Path("data/splits")
out_dir.mkdir(parents=True, exist_ok=True)
```

```
    train_df = X_train.copy()
    train_df["income"] = y_train.values
    val_df = X_val.copy()
    val_df["income"] = y_val.values
    test_df = X_test.copy()
    test_df["income"] = y_test.values

    train_path = out_dir / "train.csv"
    val_path = out_dir / "val.csv"
    test_path = out_dir / "test.csv"

    train_df.to_csv(train_path, index=False)
    val_df.to_csv(val_path, index=False)
    test_df.to_csv(test_path, index=False)

    def dist(series: pd.Series) -> dict:
        vc = series.value_counts(normalize=True)
        return {str(k): float(v) for k, v in vc.items()}

    print(f"[split] Input: {in_path} shape={df.shape}")
    print(f"[split] seed={seed} stratify={stratify}")
    print(f"[split] train={len(train_df)} val={len(val_df)} test={len(test_df)}")
    print(f"[split] train class dist: {dist(train_df['income'])}")
    print(f"[split] val   class dist: {dist(val_df['income'])}")
    print(f"[split] test  class dist: {dist(test_df['income'])}")
    print(f"[split] Saved: {train_path}, {val_path}, {test_path}")


if __name__ == "__main__":
    main()
```

Étape 3 – Train

Script : **src/train/train.py**

```
from __future__ import annotations
from pathlib import Path
import json
import yaml
import pandas as pd

from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import OneHotEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

import joblib
import mlflow
import mlflow.sklearn


def load_params(path: str = "params.yaml") -> dict:
    with open(path, "r", encoding="utf-8") as f:
        return yaml.safe_load(f)


def main() -> None:
    params = load_params()
    seed = int(params["seed"])

    train_path = Path("data/splits/train.csv")
    test_path = Path("data/splits/test.csv")

    df_train = pd.read_csv(train_path)
    df_test = pd.read_csv(test_path)

    target = "income"
    X_train, y_train = df_train.drop(columns=[target]), df_train[target]
    X_test, y_test = df_test.drop(columns=[target]), df_test[target]

    cat_cols = X_train.select_dtypes(include=["object"]).columns.tolist()
    num_cols = [c for c in X_train.columns if c not in cat_cols]

    preprocessor = ColumnTransformer(
        transformers=[
            ("cat", OneHotEncoder(handle_unknown="ignore"), cat_cols),
            ("num", "passthrough", num_cols),
        ]
    )

    clf = Pipeline(
        steps=[
            ("prep", preprocessor),
            ("model", LogisticRegression(max_iter=1000, random_state=seed)),
        ]
    )

    mlflow.set_tracking_uri("file:./mlruns")
    mlflow.set_experiment("adult-income-dvc-mlflow")

    Path("models").mkdir(parents=True, exist_ok=True)
    Path("reports").mkdir(parents=True, exist_ok=True)

    with mlflow.start_run():
        mlflow.log_param("seed", seed)

        clf.fit(X_train, y_train)
        preds = clf.predict(X_test)
        acc = accuracy_score(y_test, preds)

        mlflow.log_metric("test_accuracy", float(acc))

        model_path = Path("models/model.joblib")
        joblib.dump(clf, model_path)
        mlflow.sklearn.log_model(clf, "model")

        metrics_path = Path("reports/metrics.json")
        with open(metrics_path, "w", encoding="utf-8") as f:
            json.dump({"test_accuracy": float(acc)}, f, indent=2)
        mlflow.log_artifact(str(metrics_path))

        print("[train] test_accuracy =", float(acc))
        print("[train] saved model:", model_path)
        print("[train] saved metrics:", metrics_path)
```

```
if __name__ == "__main__":
    main()
```

Avant d'exécuter ce script lancer mlflow dans le même répertoire:

```
> [poetry run] mlflow ui --backend-store-uri ./mlruns --port 5000
```

Étape 4 – dvc pipeline

Script : **dvc.yaml**

```
stages:
  download:
    cmd: python src/data/download.py
    outs:
      - data/raw/adult.csv

  preprocess:
    cmd: python src/data/preprocess.py
    deps:
      - data/raw/adult.csv
    params:
      - preprocess
    outs:
      - data/processed/adult_clean.csv

  split:
    cmd: python src/data/split.py
    deps:
      - data/processed/adult_clean.csv
    params:
      - seed
      - split
    outs:
      - data/splits/train.csv
      - data/splits/val.csv
      - data/splits/test.csv
```

Exécuter Dvc repro pour lancer tous les etapes et entrainer

```
dvc remove .\data\raw\adult.csv.dvc
dvc repro
```