
**MINISTERE DE L'ENSEIGNEMENT SUPERIEUR DE LA RECHERCHE ET DE L'INNOVATION**

********** ****** **********

# RAPPORT DE PROJET :

# « Traitement de Données avec Pandas, Seaborn, PyTorch et Keras »

**Présenté par :**

**DJIGUIMDE Oumaro Titans**

**DOUMBIA Cheick Oumar**

**FALL Ismael**

**Encadré par :**

**Pr Moustapha DER**

**'INGC2', Année académique :** *2025– 2026*

# Sommaire

# 1. INTRODUCTION ET OBJECTIFS

## 1.1 Objectifs pédagogiques

Ce projet a pour objectif principal d'acquérir une compréhension approfondie du workflow du traitement de données et du machine learning avec Python. Il permet de développer des compétences pratiques couvrant l'ensemble du pipeline, depuis les données brutes jusqu'à la modélisation et l'évaluation des performances.

Les bibliothèques étudiées sont :

- **Pandas** : manipulation et nettoyage des données,
- **Seaborn** : visualisation graphique et exploration,
- **Keras/TensorFlow** : développement rapide de modèles,
- **PyTorch** : contrôle avancé et architectures complexes.

Ce projet vise également à comparer les outils suivants : Pytorch et Keras afin d'en comprendre les forces et les limites en situation réelle.

## 1.2 Outils et technologies utilisés

Les principales bibliothèques employées sont :

```
# Pour l'utilisation de Pandas et Seaborn

import pandas as pd
import numpy as np
import seaborn as sns

# Pour l'utilisation de PyTorch
import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.utils.data import DataLoader, TensorDataset

# Pour l'utilisation de Keras
from tensorflow import keras
from tensorflow.keras import layers

# Pour l'Evaluation
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
```

```
from sklearn.metrics import accuracy_score, confusion_matrix,
classification_report
```

# 2. ENVIRONNEMENT ET PRÉREQUIS

## 2.1 Installation des dépendances

Les bibliothèques nécessaires sont installées avec la commande suivante :

**pip install pandas seaborn matplotlib numpy scikit-learn torch tensorflow**

## 2.2 Connaissances requises

Avant d'aborder ce projet, il est recommandé d'avoir :

- Un niveau intermédiaire en Python,
- Des bases en statistiques,
- Une connaissance générale du machine learning,
- Des notions élémentaires en réseaux neuronaux.

# 3. PRINCIPE GÉNÉRAL DU WORKFLOW

## 3.1 Chaîne de traitement

Le workflow général adopté est :

Données brutes → Exploration (Pandas/Seaborn) → Prétraitement (Pandas/scikit-learn)

→ Modélisation (Keras/PyTorch) → Évaluation → Déploiement

## 3.2 Description des étapes

Exploration : Étude statistique et visualisation pour comprendre la structure du dataset.

Prétraitement : Nettoyage, normalisation, encodage, split des données.

Modélisation : Choix des modèles selon les performances attendues.

Évaluation : Analyse via des métriques adaptées (MSE, accuracy, etc.).

## 3.3 Concepts fondamentaux pour le traitement de données

### 1. Ratios d'entraînement et de test

Les ratios de division des données en ensembles d'entraînement et de test varient en fonction de la taille totale de votre jeu de données et de la complexité du modèle. L'objectif est de fournir suffisamment de données pour que le modèle apprenne efficacement tout en conservant un ensemble de test suffisamment important pour une évaluation fiable.

Les répartitions couramment utilisées sont :

a. 80 % entraînement / 20 % test : C'est le ratio standard et le plus fréquent, offrant un bon compromis pour de nombreux ensembles de données de taille moyenne à grande.

b. 70 % entraînement / 30 % test : Utilisé lorsque l'on souhaite une évaluation plus robuste de la performance de généralisation, ou si le modèle est relativement simple.

c. 90 % entraînement / 10 % test : Souvent adopté pour les très grands ensembles de données, où même 10 % représentent un volume suffisant pour un test fiable, ou lorsque la validation croisée est utilisée.

Dans une configuration plus complète, on utilise souvent un troisième ensemble de validation pour le réglage des hyperparamètres, avec des ratios comme 70 % entraînement / 15 % validation / 15 % test.

### 2. Fonctions d'activation

Dans un réseau de neurones, la fonction d'activation détermine si un neurone doit être activé ou non et introduit une non-linéarité dans le modèle. Sans elles, le réseau se comporterait comme une simple régression linéaire, incapable d'apprendre des relations complexes dans les données, ce qui est essentiel pour des tâches comme la reconnaissance d'images ou le traitement du langage naturel.

Des exemples courants incluent ReLU, Sigmoïde, et Tanh.

## 3. Optimiseur

Un optimiseur est un algorithme qui ajuste les paramètres (poids et biais) du modèle pendant l'entraînement afin de minimiser la fonction de perte (ou fonction de coût). La fonction de perte mesure l'écart entre les prédictions du modèle et les valeurs réelles. L'optimiseur guide le processus d'apprentissage en calculant les gradients (dérivées partielles de la perte par rapport aux poids) et en déterminant comment mettre à jour les poids pour réduire l'erreur.

## 4. Pooling (ou échantillonnage)

Le pooling est une technique utilisée principalement dans les réseaux de neurones convolutifs (CNN) pour réduire la dimensionnalité des données (images, etc.) tout en conservant les informations essentielles.

À quoi il sert : Il permet de réduire la quantité de calculs, de contrôler le surapprentissage (overfitting) et de rendre le modèle plus robuste aux petites translations ou distorsions dans les données d'entrée.

Les types courants sont le Max Pooling (qui prend la valeur maximale dans une fenêtre spécifique) et l'Average Pooling (qui prend la valeur moyenne).

## 5. Adam (Optimiseur adaptatif)

Adam (Adaptive Moment Estimation) est un optimiseur très populaire et largement utilisé dans le deep learning.

Pourquoi il est utilisé : Il est apprécié pour son efficacité, sa convergence rapide et sa robustesse. Il combine les avantages de deux autres extensions de la descente de gradient : l'AdaGrad (qui s'adapte aux gradients rares) et le RMSProp (qui s'adapte aux gradients non stationnaires).

Qu'est-ce que c'est : Adam calcule les taux d'apprentissage individuels pour chaque paramètre du modèle en se basant sur la moyenne mobile des premiers moments (la moyenne des gradients) et des seconds moments (la variance au carré des

gradients). Il offre généralement de bonnes performances par défaut sur une grande variété de problèmes.

### 6. Tenseur

Un tenseur est la structure de données fondamentale utilisée dans les bibliothèques de machine learning comme TensorFlow ou PyTorch.

À quoi ça sert : C'est un conteneur générique pour les données. D'un point de vue mathématique, un tenseur est une généralisation des scalaires (nombres uniques, dimension 0), des vecteurs (tableaux 1D, dimension 1), et des matrices (tableaux 2D, dimension 2) à un nombre arbitraire de dimensions (3D, 4D, etc.). Dans le deep learning, toutes les données (entrées, poids, biais, activations intermédiaires) sont représentées sous forme de tenseurs, qui sont optimisés pour les calculs sur GPU.

# 4. ÉTUDE DE CAS : PRÉDICTION DU PRIX IMMOBILIER À DAKAR

## 4.1 Contexte

Objectif : prédire le prix de maisons à Dakar à partir de :

- Surface,
- Nombre de chambres,
- Quartier.

## 4.2 Construction et exploration du dataset

Un dataset fictif mais réaliste a été créé :

```
#2. Exemple de dataset (maison à Dakar)
data = {
```

```
    "quartier":
["Plateau","Almadies","Parcelles","Yoff","Pikine","Guediawaye","Mamelles","Libert
é 6","Ouakam","Medina"],
    "surface": [120, 250, 90, 110, 75, 80, 140, 100, 130, 85],#en m2
    "chambres": [3, 5, 2, 3, 2, 2, 4, 3, 3, 2],
    "prix": [150, 350, 120, 180, 60, 70, 220, 160, 200, 110]  # en millions FCFA
}

df = pd.DataFrame(data)
```

| | quartier | surface | chambres | prix |
|---|---|---|---|---|
| 0 | Plateau | 120 | 3 | 150 |
| 1 | Almadies | 250 | 5 | 350 |
| 2 | Parcelles | 90 | 2 | 120 |
| 3 | Yoff | 110 | 3 | 180 |
| 4 | Pikine | 75 | 2 | 60 |
| 5 | Guediawaye | 80 | 2 | 70 |
| 6 | Mamelles | 140 | 4 | 220 |
| 7 | Liberté 6 | 100 | 3 | 160 |
| 8 | Ouakam | 130 | 3 | 200 |
| 9 | Medina | 85 | 2 | 110 |

## 4.3 Prétraitement

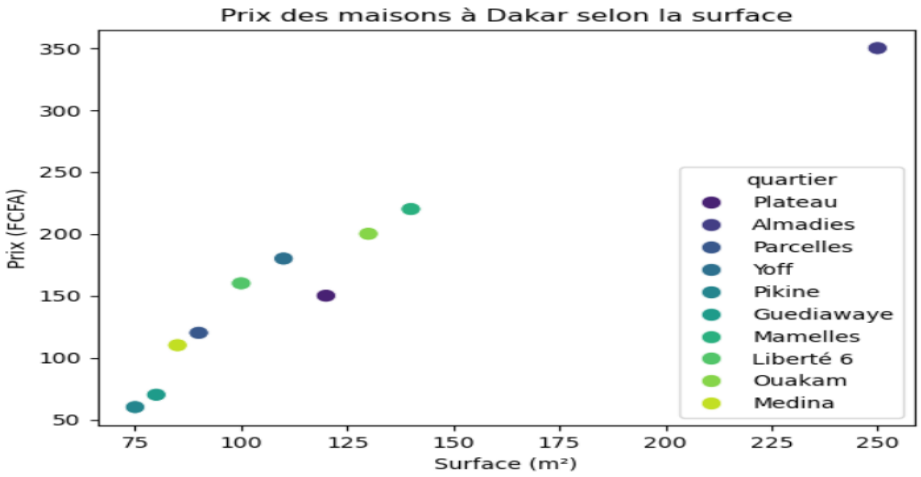
Encodage du quartier, sélection des features :

```
#3. Traitement de données (Pandas)
#3.1. Encodage du quartier (variable catégorielle → numérique)
df["quartier_code"] = df["quartier"].astype("category").cat.codes
df.head()
#3.2. Sélection des features
X = df[["surface", "chambres", "quartier_code"]]
y = df["prix"]
```

Visualisation :

```
#4-Visualisation
```

```
# Relation Surface vs Prix
sns.scatterplot(
    data=df,
    x="surface",
    y="prix",
    hue="quartier",
    palette="viridis",
    s=80
).set(
    title="Prix des maisons à Dakar selon la surface",
    xlabel="Surface (m²)",
    ylabel="Prix (FCFA)"
      )
```




## 4.4 Modèle Keras

Architecture utilisés :

- Dense(16) + ReLU
- Dense(1)

```
model_keras = keras.Sequential([
    keras.Input(shape=(3,)),
    layers.Dense(16, activation='relu'),
    layers.Dense(1)
])
model_keras.compile(optimizer='adam', loss='mse')
```

```
model_keras.fit(X, y, epochs=500, verbose=0)
```

### *4.5 Modele PyTorch*

```
model = nn.Sequential(
    nn.Linear(3, 16),
    nn.ReLU(),
    nn.Linear(16, 1)
)
```

# 5. TP : CLASSIFICATION DES FLEURS IRIS

## 5.1 Objectif

Classer des fleurs selon leurs caractéristiques botaniques (classe : setosa , virginica ,versicolor).

## 5.2 Exploration

```
#2. Chargement et exploration du dataset iris
from sklearn.datasets import load_iris

print('Type de fleur (0,1,2 → Setosa, Versicolor, Virginica)')
# Charger le dataset Iris
iris = load_iris()
df = pd.DataFrame(iris.data, columns=iris.feature_names)

# Ajouter les labels (0, 1, 2)
df['especes'] = iris.target

# Visualisation rapide
display(df.head())
print(df['especes'].value_counts())

# Pairplot pour visualisation (Seaborn uniquement)
sns.pairplot(df, hue='especes')
```

## 5.3 Préparation

- Normalisation,
- Split train/test.(Voir Code)

### 5.4 Modèle Keras

Réseau dense multiclasses avec Softmax.(Voir Code)

### 5.5 Modèle PyTorch

Implémentation avec DataLoader et CrossEntropyLoss.(Voir Code)

# 6.Comparaison des approches

## Keras(TensorFlow)

Avantages :

- API haut niveau très intuitive
- Développement rapide
- Documentation excellente
- Intégration TensorFlow complète
- Meilleur pour le prototypage

Inconvénients :

- Moins de flexibilité
- Debugging plus difficile
- Contrôle limité sur les détails

## PyTorch

Avantages :

- Contrôle granulaire
- Debugging facile (Pythonic)
- Flexibilité maximale

- Meilleur pour la recherche
- Computational graph dynamique

Inconvénients :

- Courbe d'apprentissage plus raide
- Plus de code boilerplate
- Documentation moins uniforme

Recommandations d'usage

- Débutants : Commencer par Keras
- Prototypage rapide : Keras
- Recherche/expérimentation : PyTorch
- Production : Les deux sont viables
- Contrôle maximal : PyTorch

# 7.Bonnes pratiques

La réussite d'un projet de machine learning repose sur l'adoption de méthodologies rigoureuses tout au long du pipeline de développement. Concernant la gestion des données, il est crucial de séparer les jeux d'entraînement et de test avant tout prétraitement, et de n'utiliser que les statistiques du set d'entraînement pour normaliser les données de test, évitant ainsi le data leakage. Pour la modélisation, l'implémentation de callbacks comme l'early stopping et la réduction dynamique du learning rate permet d'optimiser l'entraînement tout en prévenant le surapprentissage. Le debugging systématique inclut la vérification des shapes des

données, le monitoring des métriques d'entraînement en temps réel, et la sauvegarde régulière des modèles. Enfin, la documentation du code et la versioning des expériences garantissent la reproductibilité et la maintenabilité du projet sur le long terme.

# 8.Conclusion

Ce projet complet a démontré l'écosystème du machine learning et du deep learnng avec Python :

### Points clés à retenir :

1. **Pandas** est indispensable pour la manipulation et le nettoyage des données
2. **Seaborn** offre des visualisations puissantes pour l'exploration
3. **Keras** excelle pour le prototypage rapide et les applications standard
4. **PyTorch** offre un contrôle maximal pour la recherche et les architectures complexes

### Prochaines étapes :

- Explorer les réseaux convolutionnels (CNN) pour les images
- Découvrir les réseaux récurrents (LSTM) pour les séries temporelles
- Apprendre le transfer learning avec des modèles pré-entraînés
- Se familiariser avec le déploiement de modèles (TensorFlow Serving, TorchServe)

### Ressources recommandées :

- Documentation officielle de chaque bibliothèque
- Cours spécialisés sur les aspects théoriques du deep learning
- Projets pratiques sur des datasets variés (Kaggle)
- Participation à des compétitions pour améliorer les skills

Ce projet sert de fondation pour maîtriser l'écosystème du machine learning moderne avec Python et Keras.