Documentation - Projet de Prédiction du Diabète

Contexte du Projet

Ce projet vise à développer un système intelligent de prédiction du risque de diabète en utilisant des critères cliniques. L'analyse exploratoire des données constitue la première étape cruciale pour comprendre la structure et les caractéristiques du dataset.

📊 Code d'Analyse Exploratoire

1. Importation des Bibliothèques

```python
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

Explication :

- `pandas` : Manipulation et analyse des données tabulaires
- `numpy` : Calculs numériques et opérations sur les arrays
- `seaborn` : Visualisations statistiques avancées
- `matplotlib.pyplot` : Création de graphiques et visualisations

---

2. Chargement et Inspection Initiale des Données

```python
print("CHARGEMENT DES DONNÉES...")
df = pd.read_csv('diabetes.csv')
print(f"Dimensions du dataset: {df.shape}")
print(f"Colonnes: {df.columns.tolist()}")
```

Objectif : Charger le dataset et obtenir une vue d'ensemble de sa structure

Informations extraites :

- Nombre de lignes et colonnes
- Liste des variables disponibles
- Vérification que le fichier est correctement chargé

---

3. Analyse Exploratoire Détaillée

```python
print("ANALYSE EXPLORATOIRE ...")
print("Information sur le dataset:")
print(df.info())

print("Valeurs manquantes par colonne:")
print(df.isnull().sum())

print("Nombre de doublons:", df.duplicated().sum())
```

Analyse des Résultats :

df.info()

- **Types de données** : Vérification que les variables numériques sont bien reconnues
- **Mémoire utilisée** : Estimation de la taille du dataset
- **Valeurs non-null** : Détection rapide des données manquantes

Valeurs manquantes

- **Identification** : Localisation des colonnes avec des données manquantes
- **Quantification** : Nombre exact de valeurs manquantes par variable
- **Impact** : Évaluation de la nécessité de traitement

Doublons

- **Détection** : Identification des lignes identiques
- **Nettoyage** : Préparation pour la suppression si nécessaire

---

4. Visualisation de la Distribution des Variables

```python
```

```python
print("Distribution des variables ...")
fig, axes = plt.subplots(3, 3, figsize=(15, 12))
axes = axes.ravel()

for i, column in enumerate(df.columns):
    if i < len(axes):
        df[column].hist(bins=30, ax=axes[i], alpha=0.7)
        axes[i].set_title(f'Distribution de {column}')
        axes[i].set_xlabel(column)
        axes[i].set_ylabel('Fréquence')

plt.tight_layout()
plt.show()
```

Analyse des Distributions :

Variable	Interprétation Attendue
Pregnancies	Distribution asymétrique, beaucoup de femmes avec 0-2 grossesses
Glucose	Distribution proche de la normale, avec des valeurs élevées indiquant un risque
BloodPressure	Distribution normale centrée autour de 70-80 mmHg
SkinThickness	Présence possible de valeurs nulles (0) à traiter
Insulin	Distribution très asymétrique avec beaucoup de zéros
BMI	Distribution légèrement asymétrique, centrée autour de 25-30
DiabetesPedigreeFunction	Distribution asymétrique avec queue étendue
Age	Distribution uniforme avec concentration chez les jeunes adultes
Outcome	Variable binaire (0/1) pour diabète/non-diabète

---

5. Analyse des Corrélations

```python
print("Matrice de corrélation ...")
plt.figure(figsize=(10, 8))
correlation_matrix = df.corr()
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', center=0)
plt.title('Matrice de corrélation des variables')
plt.show()
```

⚠ Correction du Code Original :

```python
```

```
# Code corrigé (erreur dans l'original)
plt.figure(figsize=(10, 8))  # Au lieu de plt.Figure
correlation_matrix = df.corr()  # Correction du nom de variable
```

Interprétation des Corrélations :

- **Corrélations fortes** (|r| > 0.5) : Relations significatives à explorer
- **Corrélations modérées** (0.3 < |r| < 0.5) : Relations intéressantes
- **Corrélations faibles** (|r| < 0.3) : Relations peu significatives

---

6. Détection des Valeurs Aberrantes (Outliers)

```
python

def detect_outliers(df, column):
    Q1 = df[column].quantile(0.25)
    Q3 = df[column].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    return (df[column] < lower_bound) | (df[column] > upper_bound)  # Correction: | au lieu de &

numerical_columns = ['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness',
                     'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age']
outlier_indices = set()

for column in numerical_columns:
    outliers = detect_outliers(df, column)
    outlier_indices.update(df[outliers].index.tolist())
    print(f"{column}: {outliers.sum()} outliers détectés")

df_clean = df.drop(index=list(outlier_indices)).reset_index(drop=True)
print(f"Dataset après suppression des outliers : {df_clean.shape}")
```

⚠️ Corrections Importantes :

1. **Opérateur logique** : `|` (OR) au lieu de `&` (AND) pour la détection d'outliers
2. **Nom de variable** : `outlier_indices` au lieu de `outlier_indice`

Méthode IQR (Interquartile Range) :

- **Q1** : Premier quartile (25%)
- **Q3** : Troisième quartile (75%)
- **IQR** : Q3 - Q1

- **Seuils** : Q1 - 1.5×IQR et Q3 + 1.5×IQR
- **Outliers** : Valeurs en dehors de ces seuils

---

7. Sélection des Variables pour le Clustering

```python
clustering_features = ['Glucose', 'BMI', 'Age', 'DiabetesPedigreeFunction']
clustering_x = df_clean[clustering_features].copy()  # Utilisation de df_clean

print(f"Variables sélectionnées pour le clustering : {clustering_features}")
print(f"Dimensions des données de clustering : {clustering_x.shape}")
```

Justification du Choix :

- **Glucose** : Indicateur direct du diabète
- **BMI** : Facteur de risque majeur
- **Age** : Facteur de risque croissant avec l'âge
- **DiabetesPedigreeFunction** : Prédisposition génétique

---

8. Visualisation des Relations entre Variables

```python
plt.figure(figsize=(12, 10))
g = sns.pairplot(df_clean[clustering_features], diag_kind='hist', height=2.5)
g.fig.suptitle('Relations entre les variables sélectionnées', y=1.02)
plt.show()
```

Analyse du Pairplot :

- **Diagonale** : Histogrammes des distributions individuelles
- **Hors diagonale** : Nuages de points pour les relations bivariées
- **Patterns** : Identification de clusters naturels ou de corrélations

---

📋 Résumé des Corrections Apportées

Problème Original	Correction
`plt.Figure`	`plt.figure` (minuscule)
`corrélation_matrice`	`correlation_matrix` (anglais)
`(df[column] < lower_bound) & (df[column] > upper_bound)`	`(df[column] < lower_bound) | (df[column] > upper_bound)`
`outlier_indice`	`outlier_indices` (pluriel)
`df[clustering_features]`	`df_clean[clustering_features]` (données nettoyées)

🎯 Étapes Suivantes

Après cette analyse exploratoire, les prochaines étapes du projet incluront :

1. **Prétraitement avancé** : Standardisation des variables
2. **Méthode du coude** : Détermination du nombre optimal de clusters
3. **Clustering K-Means** : Application de l'algorithme
4. **Réduction dimensionnelle** : PCA pour visualisation
5. **Interprétation** : Analyse des profils de risque par cluster

---

💡 Bonnes Pratiques Appliquées

- ✅ **Documentation claire** : Commentaires explicatifs
- ✅ **Gestion des erreurs** : Vérifications de cohérence
- ✅ **Visualisations informatives** : Graphiques avec titres et labels
- ✅ **Code modulaire** : Fonctions réutilisables
- ✅ **Nettoyage des données** : Suppression des outliers

🏗️ Module Core - Préprocesseur de Données

Fichier : `core/preprocessor.py`

```python
```

```python
import unittest
import pandas as pd
from core.preprocessor import Preprocessor

class TestPreprocessor(unittest.TestCase):
    def test_remove_outliers_iqr(self):
        data = {'A': [1,2,3,1000,5], 'B':[10,15,10,20,25]}
        df = pd.DataFrame(data)
        preproc = Preprocessor(df)
        df_clean = preproc.remove_outliers_iqr(['A'])
        self.assertTrue(df_clean['A'].max() < 1000)

if __name__ == '__main__':
    unittest.main()
```

Architecture du Module Preprocessor :

Méthode	Fonction	Usage
`__init__(df)`	Initialisation avec DataFrame	`preproc = Preprocessor(df)`
`remove_outliers_iqr(columns)`	Suppression outliers par IQR	`df_clean = preproc.remove_outliers_iqr(['A'])`

Test Unitaire Expliqué :

- **Données test** : Valeur aberrante (1000) dans colonne A
- **Assertion** : Vérification que l'outlier est supprimé
- **Validation** : `df_clean['A'].max() < 1000`

🎯 Module Core - Clustering

Fichier : `core/clusterer.py`

```python
```

```python
import sys
import os
import pandas as pd
import numpy as np
from sklearn.datasets import make_blobs

# Pour trouver le module core
sys.path.insert(0, os.path.abspath(os.path.join(os.path.dirname(__file__), '..')))
from core.clusterer import Clusterer

def test_clusterer_workflow():
    # Générer des données synthétiques
    X, _ = make_blobs(n_samples=100, centers=3, n_features=2, random_state=42)
    df = pd.DataFrame(X, columns=["Feature1", "Feature2"])

    clusterer = Clusterer(df)

    # Test du scale
    X_scaled = clusterer.scale_data()
    assert X_scaled.shape == df.shape, "Erreur de mise à l'échelle"

    # Test de la méthode du coude (ne plante pas)
    clusterer.elbow_method(max_k=5)

    # Test du fit kmeans
    clustered_df = clusterer.fit_kmeans(n_clusters=3)
    assert 'Cluster' in clustered_df.columns, "Clustering non effectué"

    # Test des moyennes par cluster
    means = clusterer.cluster_means()
    assert means.shape[0] == 3, "Nombre de clusters incorrect"

    print("Tous les tests ont réussi.")

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

Architecture du Module Clusterer :

Méthode	Fonction	Test de Validation
`scale_data()`	Standardisation des données	Shape identique au DataFrame original
`elbow_method(max_k)`	Méthode du coude pour k optimal	Exécution sans erreur
`fit_kmeans(n_clusters)`	Application K-Means	Colonne 'Cluster' créée
`cluster_means()`	Calcul moyennes par cluster	Nombre de lignes = nombre de clusters

Workflow de Test :

1. **Génération données synthétiques** : `make_blobs` avec 3 centres
2. **Validation séquentielle** : Chaque méthode testée individuellement
3. **Assertions robustes** : Vérification des formes et colonnes
4. **Random state** : Reproductibilité des résultats

🚂 Script d'Entraînement Principal

Fichier : `train_model.py`

```python
```

```python
import os
import pandas as pd
import matplotlib
matplotlib.use('Agg')  # ⚠ Avant pyplot !
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
import seaborn as sns
import matplotlib.pyplot as plt
import joblib

# 1. Chargement données
BASE_DIR = os.path.dirname(os.path.abspath(__file__))
DATA_PATH = os.path.join(BASE_DIR, '..', 'data', 'processed', 'diabetes_clean_no_outliers.csv')

df = pd.read_csv(DATA_PATH)
print(f"Données chargées : {df.shape}")

# 2. Sélection features importantes (sans Outcome)
features = ["Pregnancies", "Glucose", "BloodPressure", "SkinThickness", "Insulin",
            "BMI", "DiabetesPedigreeFunction", "Age"]
X = df[features]

# 3. Mise à l'échelle
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# 4. Méthode du coude pour choisir k optimal
inertias = []
K_range = range(1, 11)
for k in K_range:
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(X_scaled)
    inertias.append(kmeans.inertia_)

plt.figure(figsize=(8,5))
plt.plot(K_range, inertias, 'bo-')
plt.xlabel('Nombre de clusters k')
plt.ylabel('Inertie')
plt.title('Méthode du coude pour choisir k')
plt.grid(True)

output_dir = os.path.join(BASE_DIR, '..', 'outputs')
os.makedirs(output_dir, exist_ok=True)

plt.savefig(os.path.join(output_dir, "cluster_elbow_plot.png"))
```

```python
plt.close()

# 5. Clustering avec k choisi (exemple k=2)
k_optimal = 2
kmeans = KMeans(n_clusters=k_optimal, random_state=42)
clusters = kmeans.fit_predict(X_scaled)

# Ajout des clusters dans le DataFrame
df['Cluster'] = clusters

# 6. Réduction dimensionnelle pour visualiser les clusters
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)
df_pca = pd.DataFrame(X_pca, columns=['PC1', 'PC2'])
df_pca['Cluster'] = clusters

plt.figure(figsize=(8,6))
sns.scatterplot(data=df_pca, x='PC1', y='PC2', hue='Cluster', palette='Set1')
plt.title('Visualisation des clusters avec PCA')
plt.savefig(os.path.join(output_dir, "PCA_clusters.png"))
plt.close()

# 7. Analyse des clusters
print("Moyennes des caractéristiques par cluster:")
print(df.groupby('Cluster')[features].mean())
print("\nEffectifs par cluster:")
print(df['Cluster'].value_counts())

# 8. Sauvegarde des modèles scaler et kmeans + PCA
model_dir = os.path.join(BASE_DIR, '..', 'model')
os.makedirs(model_dir, exist_ok=True)

joblib.dump(scaler, os.path.join(model_dir, "scaler.pkl"))
joblib.dump(kmeans, os.path.join(model_dir, "kmeans_model.pkl"))
joblib.dump(pca, os.path.join(model_dir, "pca_model.pkl"))

print("Modèles scaler, kmeans et PCA sauvegardés avec succès.")
```

📊 Pipeline d'Entraînement Détaillé

Étape 1 : Configuration et Chargement

```python
matplotlib.use('Agg')  # Mode non-interactif pour serveurs
BASE_DIR = os.path.dirname(os.path.abspath(__file__))
```

Importance : Configuration backend matplotlib pour environnements sans interface graphique

Étape 2 : Sélection des Features

```python
features = ["Pregnancies", "Glucose", "BloodPressure", "SkinThickness",
            "Insulin", "BMI", "DiabetesPedigreeFunction", "Age"]
```

Justification : Exclusion de 'Outcome' pour apprentissage non-supervisé

Étape 3 : Standardisation

```python
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

Nécessité : Égalisation des échelles pour K-Means (sensible aux unités)

Étape 4 : Méthode du Coude

```python
for k in K_range:
    kmeans = KMeans(n_clusters=k, random_state=42)
    inertias.append(kmeans.inertia_)
```

Objectif : Détermination automatique du nombre optimal de clusters

Étape 5 : Clustering Final

```python
k_optimal = 2  # Basé sur le contexte médical (risque/non-risque)
clusters = kmeans.fit_predict(X_scaled)
```

Étape 6 : Visualisation PCA

```python
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)
```

But : Projection 2D pour visualisation des clusters

Étape 7 : Analyse des Profils

```python
print(df.groupby('Cluster')[features].mean())
```

Interprétation Médicale :

- **Cluster 0** : Profil faible risque
- **Cluster 1** : Profil haut risque (Glucose>126, BMI>30, DiabetesPedigreeFunction>0.5)

Étape 8 : Persistance des Modèles

```python
joblib.dump(scaler, "scaler.pkl")
joblib.dump(kmeans, "kmeans_model.pkl")
joblib.dump(pca, "pca_model.pkl")
```

🧪 Tests de Validation des Modèles

Test du Modèle Sauvegardé

```python
import joblib
import pandas as pd

model = joblib.load("model/model.pkl")

def test_prediction():
    sample = pd.DataFrame([{
        "Pregnancies": 1, "Glucose": 85, "BloodPressure": 66,
        "SkinThickness": 29, "Insulin": 0, "BMI": 26.6,
        "DiabetesPedigreeFunction": 0.351, "Age": 31
    }])
    pred = model.predict(sample)
    assert pred in [0, 1], "La prédiction doit être binaire"

test_prediction()
```

Analyse du Test :

- **Patient exemple** : Profil relativement normal
- **Valeurs attendues** : Glucose=85 (normal), BMI=26.6 (légèrement élevé)
- **Validation** : Prédiction binaire (0=non-diabète, 1=diabète)

📁 Structure du Projet

```
diabetes_prediction/
├── data/
│   ├── raw/
│   │   └── diabetes.csv
│   └── processed/
│       └── diabetes_clean_no_outliers.csv
├── core/
│   ├── __init__.py
│   ├── preprocessor.py
│   └── clusterer.py
├── model/
│   ├── scaler.pkl
│   ├── kmeans_model.pkl
│   ├── pca_model.pkl
│   └── model.pkl
├── outputs/
│   ├── cluster_elbow_plot.png
│   └── PCA_clusters.png
├── tests/
│   ├── test_preprocessor.py
│   ├── test_clusterer.py
│   └── test_model.py
├── train_model.py
└── README.md
```

🎯 Workflow Complet du Projet

Phase 1 : Exploration et Nettoyage

1. **EDA** : Analyse exploratoire (distributions, corrélations)
2. **Preprocessing** : Suppression outliers, standardisation
3. **Feature Selection** : Sélection variables pertinentes

Phase 2 : Modélisation

1. **Méthode du coude** : Détermination k optimal
2. **K-Means** : Clustering des patients
3. **PCA** : Réduction dimensionnelle pour visualisation

Phase 3 : Validation et Déploiement

1. **Tests unitaires** : Validation modules core

2. **Sauvegarde modèles** : Persistance avec joblib
3. **Tests d'intégration** : Validation pipeline complet

🔍 Critères d'Interprétation Médicale

Seuils de Risque Diabète

Variable	Seuil Critique	Interprétation
Glucose	> 126 mg/dL	Diabète diagnostiqué
BMI	> 30	Obésité (facteur de risque)
DiabetesPedigreeFunction	> 0.5	Prédisposition génétique élevée
Age	> 45 ans	Risque croissant

Classification des Clusters

```python
# Logique d'assignation du risque
def assign_risk_category(cluster_means):
    if (cluster_means['Glucose'] > 126 and
        cluster_means['BMI'] > 30 and
        cluster_means['DiabetesPedigreeFunction'] > 0.5):
        return "High Risk"
    else:
        return "Low Risk"
```

🚀 Instructions d'Exécution

Prérequis

```bash
pip install pandas numpy scikit-learn matplotlib seaborn joblib
```

Exécution Séquentielle

```bash
```

```
# 1. Tests des modules core
python tests/test_preprocessor.py
python tests/test_clusterer.py

# 2. Entraînement du modèle
python train_model.py

# 3. Validation du modèle
python tests/test_model.py
```

Sorties Générées

- **Graphiques** : `outputs/cluster_elbow_plot.png`, `outputs/PCA_clusters.png`
- **Modèles** : `model/scaler.pkl`, `model/kmeans_model.pkl`, `model/pca_model.pkl`
- **Logs** : Moyennes par cluster, effectifs, métriques de performance

---

💡 Recommandations et Bonnes Pratiques

Code Quality

- ✅ **Tests unitaires** pour chaque module
- ✅ **Gestion des chemins** avec `os.path.join()`
- ✅ **Random seeds** pour reproductibilité
- ✅ **Documentation inline** avec docstrings

Production Readiness

- ✅ **Gestion des erreurs** avec try/catch
- ✅ **Logging structuré** pour traçabilité
- ✅ **Configuration externalisée** (fichiers config)
- ✅ **Monitoring des performances** modèles

Évolutions Futures

- 🔄 **Cross-validation** pour validation robuste
- 🔄 **Hyperparameter tuning** avec GridSearch
- 🔄 **Ensemble methods** pour améliorer performances
- 🔄 **API REST** pour déploiement en production

Cette documentation complète fait partie du projet de prédiction du diabète développé dans le cadre d'un système intelligent pour laboratoire biomédical. Elle couvre l'ensemble du pipeline depuis l'exploration des

données jusqu'au déploiement des modèles.