Partie 1 : collect des données

Importation des pakages

import numpy as np  
import pandas as pd  
import matplotlib.pyplot as plt  
import seaborn as sns  
from sklearn.model\_selection import StratifiedShuffleSplit  
from sklearn.linear\_model import LogisticRegression  
from sklearn.tree import DecisionTreeClassifier  
from sklearn.neighbors import KNeighborsClassifier  
from sklearn.metrics import accuracy\_score,classification\_report, confusion\_matrix  
import joblib  
from sklearn.model\_selection import train\_test\_split  
from sklearn.ensemble import RandomForestClassifier

Importation du fichier CSV

# chemin vers le fichier csv  
df = pd.read\_csv("C:/Users/Khalil/Documents/ProjetDataDiabete/DataPredDiabete/diabete.csv")  
df

|  | n\_pregnant | glucose | tension | thickness | insulin | bmi | pedigree | age | diabete |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 6 | 148 | 72 | 35 | 0 | 33.6 | 0.627 | 50 | 1 |
| 1 | 1 | 85 | 66 | 29 | 0 | 26.6 | 0.351 | 31 | 0 |
| 2 | 8 | 183 | 64 | 0 | 0 | 23.3 | 0.672 | 32 | 1 |
| 3 | 1 | 89 | 66 | 23 | 94 | 28.1 | 0.167 | 21 | 0 |
| 4 | 0 | 137 | 40 | 35 | 168 | 43.1 | 2.288 | 33 | 1 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 763 | 10 | 101 | 76 | 48 | 180 | 32.9 | 0.171 | 63 | 0 |
| 764 | 2 | 122 | 70 | 27 | 0 | 36.8 | 0.340 | 27 | 0 |
| 765 | 5 | 121 | 72 | 23 | 112 | 26.2 | 0.245 | 30 | 0 |
| 766 | 1 | 126 | 60 | 0 | 0 | 30.1 | 0.349 | 47 | 1 |
| 767 | 1 | 93 | 70 | 31 | 0 | 30.4 | 0.315 | 23 | 0 |

Affichage des 10 premières lignes

pd.set\_option('display.max\_rows',10)  
df

|  | n\_pregnant | glucose | tension | thickness | insulin | bmi | pedigree | age | diabete |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 6 | 148 | 72 | 35 | 0 | 33.6 | 0.627 | 50 | 1 |
| 1 | 1 | 85 | 66 | 29 | 0 | 26.6 | 0.351 | 31 | 0 |
| 2 | 8 | 183 | 64 | 0 | 0 | 23.3 | 0.672 | 32 | 1 |
| 3 | 1 | 89 | 66 | 23 | 94 | 28.1 | 0.167 | 21 | 0 |
| 4 | 0 | 137 | 40 | 35 | 168 | 43.1 | 2.288 | 33 | 1 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 763 | 10 | 101 | 76 | 48 | 180 | 32.9 | 0.171 | 63 | 0 |
| 764 | 2 | 122 | 70 | 27 | 0 | 36.8 | 0.340 | 27 | 0 |
| 765 | 5 | 121 | 72 | 23 | 112 | 26.2 | 0.245 | 30 | 0 |
| 766 | 1 | 126 | 60 | 0 | 0 | 30.1 | 0.349 | 47 | 1 |
| 767 | 1 | 93 | 70 | 31 | 0 | 30.4 | 0.315 | 23 | 0 |

Vérification des informations du DataFrame

df.info()

<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 768 entries, 0 to 767  
Data columns (total 9 columns):  
 # Column Non-Null Count Dtype   
--- ------ -------------- -----   
 0 n\_pregnant 768 non-null int64   
 1 glucose 768 non-null int64   
 2 tension 768 non-null int64   
 3 thickness 768 non-null int64   
 4 insulin 768 non-null int64   
 5 bmi 768 non-null float64  
 6 pedigree 768 non-null float64  
 7 age 768 non-null int64   
 8 diabete 768 non-null int64   
dtypes: float64(2), int64(7)  
memory usage: 54.1 KB

Partie 2 : Nettoyage des données

Vérification des valeurs manquantes

df.isnull().sum().sort\_values(ascending=False)

n\_pregnant 0  
glucose 0  
tension 0  
thickness 0  
insulin 0  
bmi 0  
pedigree 0  
age 0  
diabete 0  
dtype: int64

Statistiques descriptives

df.describe()

|  | n\_pregnant | glucose | tension | thickness | insulin | bmi | pedigree | age | diabete |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| count | 768 | 768 | 768 | 768 | 768 | 768 | 768 | 768 | 768 |
| mean | 3.84505 | 120.895 | 69.1055 | 20.5365 | 79.7995 | 31.9926 | 0.471876 | 33.2409 | 0.348958 |
| std | 3.36958 | 31.9726 | 19.3558 | 15.9522 | 115.244 | 7.88416 | 0.331329 | 11.7602 | 0.476951 |
| min | 0 | 0 | 0 | 0 | 0 | 0 | 0.078 | 21 | 0 |
| 25% | 1 | 99 | 62 | 0 | 0 | 27.3 | 0.24375 | 24 | 0 |
| 50% | 3 | 117 | 72 | 23 | 30.5 | 32 | 0.3725 | 29 | 0 |
| 75% | 6 | 140.25 | 80 | 32 | 127.25 | 36.6 | 0.62625 | 41 | 1 |
| max | 17 | 199 | 122 | 99 | 846 | 67.1 | 2.42 | 81 | 1 |

Renommage des colonnes

df = pd.read\_csv("C:/Users/Khalil/Documents/ProjetDataDiabete/DataPredDiabete/diabete.csv")  
# Renommer les colonnes  
df = df.rename(columns={  
 "n\_pregnant": "nbr\_grossesses",  
 "thickness": "epaisseur\_peau",  
 "bmi": "IMC",  
 "pedigree": "Coef\_hereditaire",  
   
})  
  
# Vérification après modification des noms  
print("Noms des colonnes après modification :")  
print(df.columns)  
# Enregistrer le DataFrame modifié dans un nouveau fichier CSV  
df.to\_csv('C:/Users/Khalil/Documents/ProjetDataDiabete/DataPredDiabete/diabete\_nettoye.csv', index=False)

Noms des colonnes après modification :  
Index(['nbr\_grossesses', 'glucose', 'tension', 'epaisseur\_peau', 'insulin',  
 'IMC', 'Coef\_hereditaire', 'age', 'diabete'],  
 dtype='object')

Vérification des doublons

duplicates = df.duplicated()  
# Afficher les doublons  
print("Lignes dupliquées :")  
print(df[duplicates])  
# Compter les doublons  
num\_duplicates = duplicates.sum()  
print(f"Nombre de lignes dupliquées : {num\_duplicates}")

Lignes dupliquées :  
Empty DataFrame  
Columns: [nbr\_grossesses, glucose, tension, epaisseur\_peau, insulin, IMC, Coef\_hereditaire, age, diabete]  
Index: []  
Nombre de lignes dupliquées : 0

df = pd.read\_csv("C:/Users/Khalil/Documents/ProjetDataDiabete/DataPredDiabete/diabete\_nettoye.csv")  
df

|  | nbr\_grossesses | glucose | tension | epaisseur\_peau | insulin | IMC | Coef\_hereditaire | age | diabete |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 6 | 148 | 72 | 35 | 0 | 33.6 | 0.627 | 50 | 1 |
| 1 | 1 | 85 | 66 | 29 | 0 | 26.6 | 0.351 | 31 | 0 |
| 2 | 8 | 183 | 64 | 0 | 0 | 23.3 | 0.672 | 32 | 1 |
| 3 | 1 | 89 | 66 | 23 | 94 | 28.1 | 0.167 | 21 | 0 |
| 4 | 0 | 137 | 40 | 35 | 168 | 43.1 | 2.288 | 33 | 1 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 763 | 10 | 101 | 76 | 48 | 180 | 32.9 | 0.171 | 63 | 0 |
| 764 | 2 | 122 | 70 | 27 | 0 | 36.8 | 0.340 | 27 | 0 |
| 765 | 5 | 121 | 72 | 23 | 112 | 26.2 | 0.245 | 30 | 0 |
| 766 | 1 | 126 | 60 | 0 | 0 | 30.1 | 0.349 | 47 | 1 |
| 767 | 1 | 93 | 70 | 31 | 0 | 30.4 | 0.315 | 23 | 0 |

Partie 3 : Exploration et Analyse des Données

Spécification de la cible (diabète)

# Définir la colonne cible (target) et les facteurs   
target = df["diabete"] # Colonne cible  
facteur = df.drop(columns=["diabete"]) # Toutes les autres colonnes  
# Affichage des résultats  
print("Colonne cible (target) :")  
display(target)  
print("\nFacteurs:")  
display(facteur)

Colonne cible (target) :

0 1  
1 0  
2 1  
3 0  
4 1  
 ..  
763 0  
764 0  
765 0  
766 1  
767 0  
Name: diabete, Length: 768, dtype: int64

Facteurs:

|  | nbr\_grossesses | glucose | tension | epaisseur\_peau | insulin | IMC | Coef\_hereditaire | age |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 6 | 148 | 72 | 35 | 0 | 33.6 | 0.627 | 50 |
| 1 | 1 | 85 | 66 | 29 | 0 | 26.6 | 0.351 | 31 |
| 2 | 8 | 183 | 64 | 0 | 0 | 23.3 | 0.672 | 32 |
| 3 | 1 | 89 | 66 | 23 | 94 | 28.1 | 0.167 | 21 |
| 4 | 0 | 137 | 40 | 35 | 168 | 43.1 | 2.288 | 33 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 763 | 10 | 101 | 76 | 48 | 180 | 32.9 | 0.171 | 63 |
| 764 | 2 | 122 | 70 | 27 | 0 | 36.8 | 0.340 | 27 |
| 765 | 5 | 121 | 72 | 23 | 112 | 26.2 | 0.245 | 30 |
| 766 | 1 | 126 | 60 | 0 | 0 | 30.1 | 0.349 | 47 |
| 767 | 1 | 93 | 70 | 31 | 0 | 30.4 | 0.315 | 23 |

Analyse de la variable cible

#on va commencer pa le variable target  
target.value\_counts()

diabete  
0 500  
1 268  
Name: count, dtype: int64

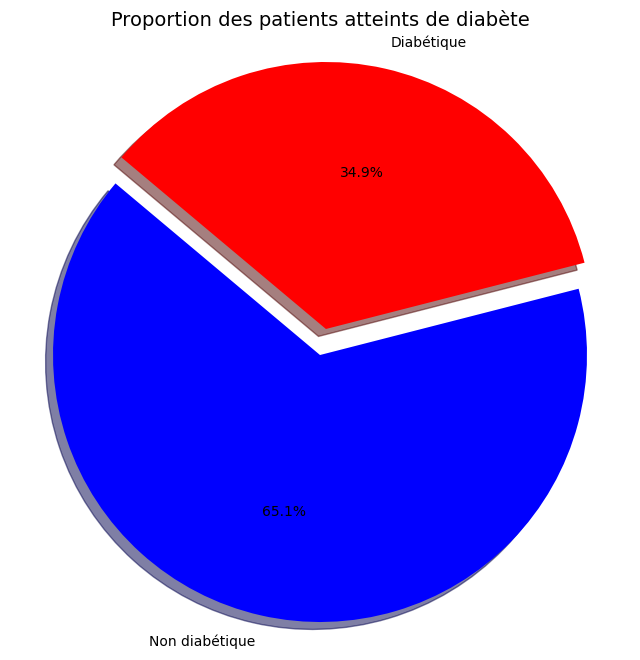
Calcul des proportions des classes

class\_counts = target.value\_counts(normalize=True)  
diabetes\_yes = class\_counts.get(1, 0) # Proportion de la classe 1 (diabète présent)  
diabetes\_no = class\_counts.get(0, 0) # Proportion de la classe 0 (diabète absent)  
  
# Affichage des proportions avec formatage  
print(f'Proportion des patients atteints de diabète : {diabetes\_yes:.2%}')  
print(f'Proportion des patients non atteints de diabète : {diabetes\_no:.2%}')

Proportion des patients atteints de diabète : 34.90%  
Proportion des patients non atteints de diabète : 65.10%

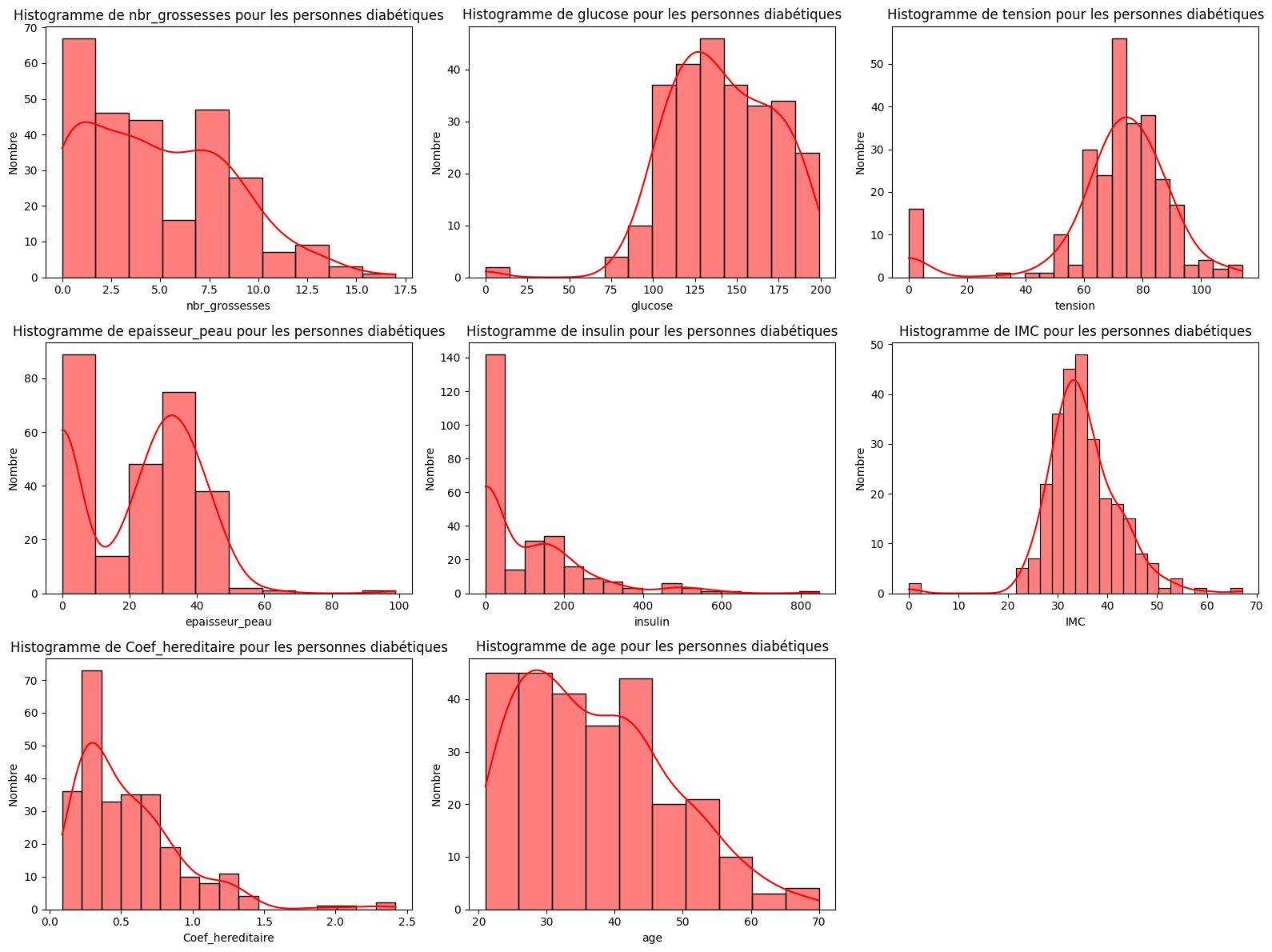
Diagramme circulaire des proportions

# Préparation des données pour le diagramme circulaire  
labels = ['Non diabétique', 'Diabétique']  
sizes = [diabetes\_no \* 100, diabetes\_yes \* 100] # Convertir les proportions en pourcentages  
colors = ['blue', 'red'] # Couleurs pour le diagramme circulaire  
explode = (0.1, 0) # Exploser la première tranche pour la mettre en évidence  
  
# Création du diagramme circulaire  
plt.figure(figsize=(8, 8))  
plt.pie(sizes, explode=explode, labels=labels, colors=colors, autopct='%1.1f%%', shadow=True, startangle=140)  
  
# Ajout d'un titre au diagramme circulaire  
plt.title("Proportion des patients atteints de diabète", fontsize=14)  
  
# Assure que le diagramme circulaire est un cercle  
plt.axis('equal')  
  
# Affichage du diagramme circulaire  
plt.show()



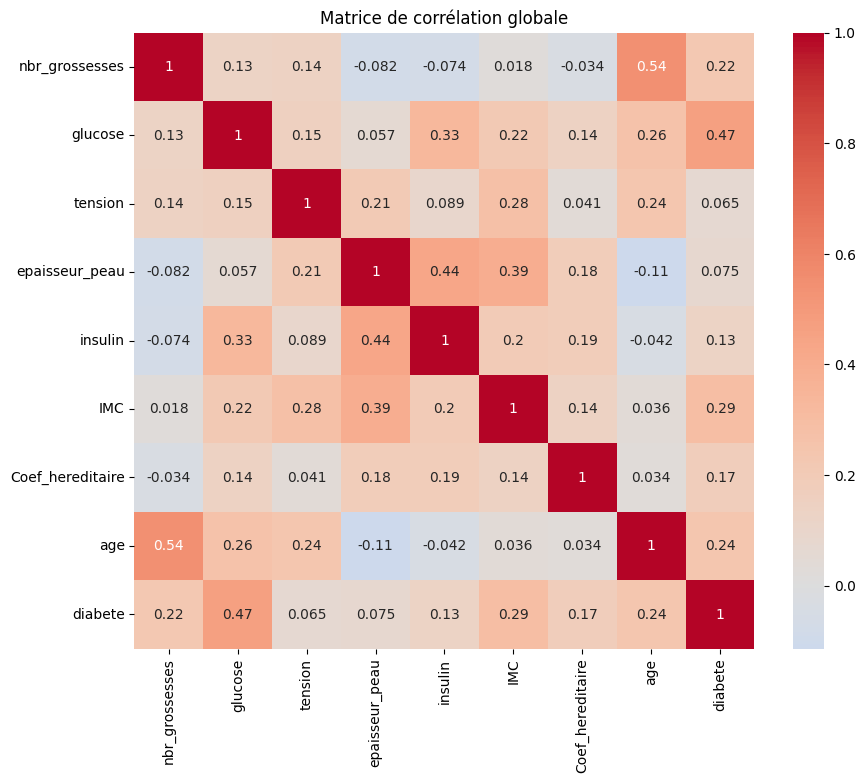
Histogrammes des variables pour les personnes diabétiques

# Définition des colonnes à tracer  
variables = ['nbr\_grossesses', 'glucose', 'tension', 'epaisseur\_peau',  
 'insulin', 'IMC', 'Coef\_hereditaire', 'age']  
  
# Filtrer les données pour les personnes atteintes de diabète  
df\_diabete = df[df['diabete'] == 1] # Sélection des cas où 'diabete' == 1  
  
# Configuration des graphiques avec matplotlib  
plt.figure(figsize=(16, 12)) # Taille de la figure  
  
# Boucle pour créer un histogramme pour chaque variable  
for i, column in enumerate(variables, 1):  
 plt.subplot(3, 3, i) # Crée une grille 3x3 pour les sous-graphiques  
 sns.histplot(data=df\_diabete, x=column, kde=True, color='red') # Histogramme filtré  
 plt.title(f'Histogramme de {column} pour les personnes diabétiques')  
 plt.xlabel(column)  
 plt.ylabel('Nombre')  
  
# Ajuster l'affichage  
plt.tight\_layout()  
plt.show()



Matrice de corrélation

# Calculer la matrice de corrélation  
correlation\_matrix = df.corr()  
# Visualiser la matrice de corrélation  
plt.figure(figsize=(10, 8))  
sns.heatmap(correlation\_matrix, annot=True, cmap='coolwarm', center=0)  
plt.title("Matrice de corrélation globale")  
plt.show()



Partie 4 : Modélisation

Séparation des données

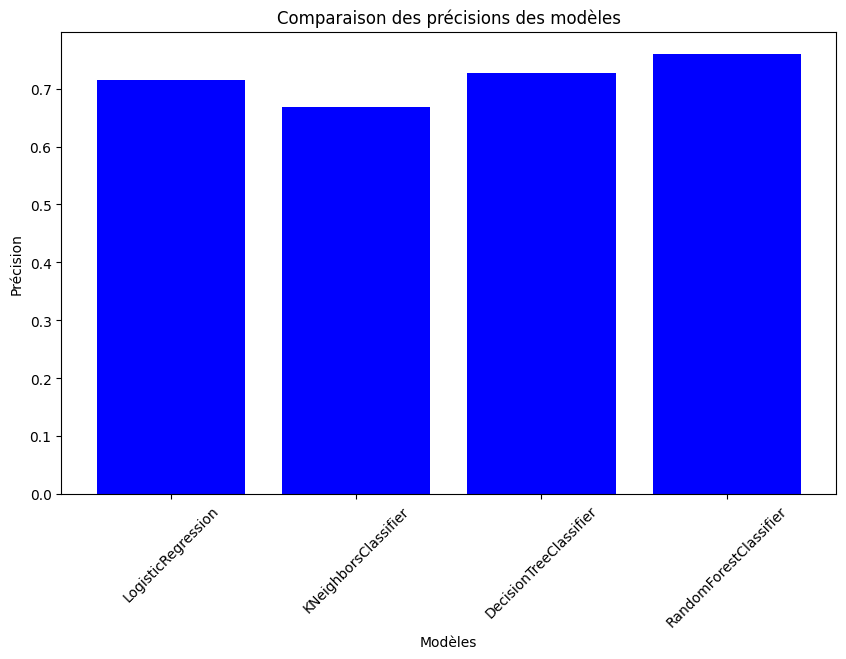
X = df.drop('diabete', axis=1) # Supposons que 'Diabete' est la colonne cible  
y = df['diabete'] # La colonne cible est 'Diabete'  
  
# Diviser la base de données en test et entraînement  
sss = StratifiedShuffleSplit(n\_splits=1, test\_size=0.2, random\_state=42)  
for train, test in sss.split(X, y):  
 X\_train, X\_test = X.iloc[train], X.iloc[test]  
 y\_train, y\_test = y.iloc[train], y.iloc[test]  
  
print('X\_train taille :', X\_train.shape)  
print('X\_test taille :', X\_test.shape)  
print('y\_train taille :', y\_train.shape)  
print('y\_test taille :', y\_test.shape)

X\_train taille : (614, 8)  
X\_test taille : (154, 8)  
y\_train taille : (614,)  
y\_test taille : (154,)

Application des algorithmes de Machine Learning

# Définition des modèles  
models = {  
 'LogisticRegression': LogisticRegression(max\_iter=1000, random\_state=42),  
 'KNeighborsClassifier': KNeighborsClassifier(n\_neighbors=5),  
 'DecisionTreeClassifier': DecisionTreeClassifier(random\_state=42),  
 'RandomForestClassifier': RandomForestClassifier(n\_estimators=100, random\_state=42)  
}  
  
# Fonction pour calculer la précision  
def accury\_true(y\_true, y\_pred):  
 return accuracy\_score(y\_true, y\_pred)  
  
# Fonction pour entraîner, évaluer les modèles et afficher un graphique  
def train\_test\_eval(models, X\_train, y\_train, X\_test, y\_test):  
 accuracies = [] # Liste pour stocker les précisions  
 for name, model in models.items():  
 print(f"{name} :")  
 model.fit(X\_train, y\_train) # Entraînement du modèle  
 y\_pred = model.predict(X\_test) # Prédiction sur les données de test  
 accuracy = accury\_true(y\_test, y\_pred) # Calcul de la précision  
 accuracies.append(accuracy) # Ajout de la précision à la liste  
 print(f"Précision : {accuracy:.2f}")  
 print('-' \* 30)  
  
 # Tracer les précisions des modèles  
 plt.figure(figsize=(10, 6))  
 plt.bar(models.keys(), accuracies, color='blue')  
 plt.xlabel('Modèles')  
 plt.ylabel('Précision')  
 plt.title('Comparaison des précisions des modèles')  
 plt.xticks(rotation=45)  
 plt.show()  
  
# Appel de la fonction pour appliquer les modèles et afficher le graphique  
train\_test\_eval(models, X\_train, y\_train, X\_test, y\_test)

LogisticRegression :  
Précision : 0.71  
------------------------------  
KNeighborsClassifier :  
Précision : 0.67  
------------------------------  
DecisionTreeClassifier :  
Précision : 0.73  
------------------------------  
RandomForestClassifier :  
Précision : 0.76  
------------------------------



Sélection du modèle choisi :RandomForestClassifier

X = df.drop('diabete', axis=1) # 'Diabete' est la colonne cible  
y = df['diabete']  
  
# Séparation des données  
X = df.drop('diabete', axis=1) # 'Diabete' est la colonne cible  
y = df['diabete']  
  
# Séparation en ensembles d'entraînement et de test (80% / 20%)  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)  
  
# Définition du modèle RandomForestClassifier  
rf\_model = RandomForestClassifier(n\_estimators=100, max\_depth=5, random\_state=42)  
  
# Entraînement du modèle avec les données d'entraînement  
rf\_model.fit(X\_train, y\_train)  
  
# Prédiction sur les données de test  
y\_pred = rf\_model.predict(X\_test)  
  
# Calcul de la précision du modèle  
accuracy = accuracy\_score(y\_test, y\_pred)  
print(f"La précision du modèle RandomForestClassifier est: {accuracy}")  
  
# Rapport détaillé  
print("\nRapport de classification :")  
print(classification\_report(y\_test, y\_pred))  
  
# Matrice de confusion  
print("\nMatrice de confusion :")  
print(confusion\_matrix(y\_test, y\_pred))

La précision du modèle RandomForestClassifier est: 0.7662337662337663  
  
Rapport de classification :  
 precision recall f1-score support  
  
 0 0.81 0.84 0.82 99  
 1 0.69 0.64 0.66 55  
  
 accuracy 0.77 154  
 macro avg 0.75 0.74 0.74 154  
weighted avg 0.76 0.77 0.76 154  
  
  
Matrice de confusion :  
[[83 16]  
 [20 35]]

Évaluation du modèle : Calcul du taux d'erreur

# Calcul du taux d'erreur  
error\_rate = 1 - accuracy  
print(f'Le taux d\'erreur du modèle est: {error\_rate}')

Le taux d'erreur du modèle est: 0.23376623376623373

Déploiement du modèle

# Sauvegarde du modèle dans un fichier  
joblib.dump(rf\_model, 'random\_forest\_model.joblib')  
print("Le modèle a été sauvegardé sous 'random\_forest\_model.joblib'.")  
  
# Charger le modèle sauvegardé  
model\_loaded = joblib.load('random\_forest\_model.joblib')  
  
# Vérification de la précision avec le modèle chargé  
y\_pred\_loaded = model\_loaded.predict(X\_test)  
accuracy\_loaded = accuracy\_score(y\_test, y\_pred\_loaded)  
print(f"La précision du modèle chargé est: {accuracy\_loaded}")  
  
# Exemple de nouvelles données pour prédiction (sans la colonne cible 'diabete')  
new\_data = pd.DataFrame({  
 'nbr\_grossesses': [5, 2],  
 'glucose': [500, 60],  
 'tension': [80, 60],  
 'epaisseur\_peau': [15, 0],  
 'insulin': [300, 90],  
 'IMC': [40.5, 10],  
 'Coef\_hereditaire': [0.2, 0.5],  
 'age': [20, 60]  
})  
  
# Afficher les nouvelles données pour vérification  
print("\nNouvelles données pour prédiction :")  
display(new\_data)  
  
# Prédiction avec le modèle chargé sur les nouvelles données  
new\_predictions = model\_loaded.predict(new\_data)  
  
# Affichage des prédictions  
for i, prediction in enumerate(new\_predictions):  
 print(f"Prédiction pour la ligne {i+1} : {'Diabétique' if prediction == 1 else 'Non diabétique'}")

Le modèle a été sauvegardé sous 'random\_forest\_model.joblib'.  
La précision du modèle chargé est: 0.7662337662337663  
  
Nouvelles données pour prédiction :

Prédiction pour la ligne 1 : Diabétique  
Prédiction pour la ligne 2 : Non diabétique

|  | nbr\_grossesses | glucose | tension | epaisseur\_peau | insulin | IMC | Coef\_hereditaire | age |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 5 | 500 | 80 | 15 | 300 | 40.5 | 0.2 | 20 |
| 1 | 2 | 60 | 60 | 0 | 90 | 10 | 0.5 | 60 |

Installation de pypandoc pour la conversion de documents

import pypandoc  
pypandoc.download\_pandoc()  
import warnings  
warnings.simplefilter(action='ignore', category=FutureWarning)