All python code

# Import necessary libraries for data analysis, visualization, and modeling

import pandas as pd                    # data manipulation

import numpy as np                     # numerical computations

import matplotlib.pyplot as plt        # plotting

import seaborn as sns                  # statistical data visualization

# Scikit-learn libraries for machine learning modeling and evaluation

from sklearn.model\_selection import train\_test\_split   # splitting data into train/test sets

from sklearn.preprocessing import StandardScaler       # feature scaling

from sklearn.tree import DecisionTreeClassifier, plot\_tree  # decision tree model and visualization

from sklearn.metrics import accuracy\_score, confusion\_matrix, classification\_report

from sklearn.metrics import ConfusionMatrixDisplay, roc\_curve, auc, RocCurveDisplay

# Load the Pima Indians Diabetes dataset

url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.csv"

# The dataset on the URL has no header row, so we specify column names manually

column\_names = ["Pregnancies", "Glucose", "BloodPressure", "SkinThickness",

                "Insulin", "BMI", "DiabetesPedigreeFunction", "Age", "Outcome"]

df = pd.read\_csv(url, header=None, names=column\_names)

# Preview the first 5 rows of the dataset

print("First 5 rows of the dataset:")

display(df.head())

# Display the shape of the dataset

print(f"Dataset shape (rows, columns): {df.shape}")

# Basic information about the dataset

print("\nDataset Info:")

df.info()

# Statistical summary of the dataset

print("\nStatistical Summary:")

display(df.describe())

# Define columns where 0 should be considered missing

cols\_with\_missing = ["Glucose", "BloodPressure", "SkinThickness", "Insulin", "BMI"]

# Replace zero values with NaN in the defined columns

for col in cols\_with\_missing:

    df[col] = df[col].replace(0, np.nan)

# Check the count of missing values in each column after replacement

print("\nMissing values per column after cleaning:")

print(df.isna().sum())

# Visualise the class distribution of the target variable 'Outcome'

sns.countplot(x='Outcome', data=df)

plt.title('Outcome Class Distribution')

plt.xlabel('Outcome (0 = No Diabetes, 1 = Diabetes)')

plt.ylabel('Count')

plt.show()

# Histogram of Glucose values

sns.histplot(df['Glucose'].dropna(), kde=True, color='skyblue')

plt.title('Distribution of Glucose Levels')

plt.xlabel('Glucose Level (mg/dL)')

plt.ylabel('Frequency')

plt.show()

# Histogram of BMI values

sns.histplot(df['BMI'].dropna(), kde=True, color='salmon')

plt.title('Distribution of BMI')

plt.xlabel('Body Mass Index (BMI)')

plt.ylabel('Frequency')

plt.show()

# Countplot of number of pregnancies

sns.countplot(x='Pregnancies', data=df, color='violet')

plt.title('Distribution of Number of Pregnancies')

plt.xlabel('Number of Pregnancies')

plt.ylabel('Count')

plt.show()

# Compute correlation matrix (fill missing values with median to avoid NaNs in correlation)

df\_corr = df.copy()

for col in cols\_with\_missing:

    df\_corr[col].fillna(df\_corr[col].median(), inplace=True)

corr\_matrix = df\_corr.corr()

# Plot a heatmap of the correlation matrix

plt.figure(figsize=(8,6))

sns.heatmap(corr\_matrix, annot=True, fmt=".2f", cmap='coolwarm')

plt.title('Correlation Heatmap of Features')

plt.show()

# Separate features and target

X = df.drop('Outcome', axis=1)

y = df['Outcome']

# Split into training and testing sets (80% train, 20% test)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

    X, y, test\_size=0.2, random\_state=42, stratify=y

)

print(f"Training set size: {X\_train.shape[0]} samples")

print(f"Test set size: {X\_test.shape[0]} samples")

# Impute missing values in training data with median, and apply to test data

for col in cols\_with\_missing:

    median\_val = X\_train[col].median()

    X\_train[col].fillna(median\_val, inplace=True)

    X\_test[col].fillna(median\_val, inplace=True)

# Feature Scaling (optional, especially for non-tree models)

scaler = StandardScaler()

X\_train\_scaled = pd.DataFrame(scaler.fit\_transform(X\_train), columns=X\_train.columns)

X\_test\_scaled  = pd.DataFrame(scaler.transform(X\_test), columns=X\_test.columns)

# Verify that no missing values remain after imputation

print("Missing values in X\_train after imputation:", X\_train\_scaled.isna().sum().sum())

print("Missing values in X\_test after imputation:", X\_test\_scaled.isna().sum().sum())

# Initialize and train the Decision Tree model

clf = DecisionTreeClassifier(random\_state=42)

clf.fit(X\_train\_scaled, y\_train)

# Optional: print tree depth and number of leaves to see model complexity

print(f"Trained Decision Tree depth: {clf.get\_depth()}, leaves: {clf.get\_n\_leaves()}")

from sklearn.tree import plot\_tree

# Plot the Decision Tree

plt.figure(figsize=(20,10))

plot\_tree(clf,

          feature\_names=X\_train.columns,

          class\_names=["No Diabetes", "Diabetes"],

          filled=True,

          rounded=True,

          fontsize=10)

plt.title("Decision Tree Classifier for Diabetes Prediction")

plt.show()

plt.figure(figsize=(80, 40))  # much larger canvas

plot\_tree(clf,

          feature\_names=X\_train.columns,

          class\_names=["No Diabetes", "Diabetes"],

          filled=True,

          rounded=True,

          fontsize=10)

plt.title("Full Decision Tree for Diabetes Prediction")

plt.savefig("full\_decision\_tree.png", dpi=300, bbox\_inches='tight')

plt.show()

from sklearn.tree import plot\_tree

plt.figure(figsize=(16, 10))

plot\_tree(clf,

          feature\_names=X\_train.columns,

          class\_names=["No Diabetes", "Diabetes"],

          filled=True,

          rounded=True,

          max\_depth=3,  # limit depth for readability

          fontsize=10)

plt.title("Simplified Decision Tree for Diabetes Prediction")

plt.show()

# Make predictions on the test set

y\_pred = clf.predict(X\_test\_scaled)

# Calculate accuracy

test\_accuracy = accuracy\_score(y\_test, y\_pred)

print(f"Test Accuracy: {test\_accuracy:.2%}")

# Display the confusion matrix

cm = confusion\_matrix(y\_test, y\_pred, labels=[0, 1])

disp = ConfusionMatrixDisplay(confusion\_matrix=cm, display\_labels=["No Diabetes", "Diabetes"])

disp.plot(cmap="Blues")

plt.title('Confusion Matrix')

plt.show()

# Print detailed classification report

print("Classification Report:")

print(classification\_report(y\_test, y\_pred, target\_names=["No Diabetes", "Diabetes"]))

# Compute predicted probabilities for the positive class

y\_proba = clf.predict\_proba(X\_test\_scaled)[:, 1]

# Calculate false positive rate and true positive rate for various thresholds

fpr, tpr, thresholds = roc\_curve(y\_test, y\_proba)

roc\_auc = auc(fpr, tpr)

print(f"AUC Score: {roc\_auc:.2f}")

# Plot the ROC curve

plt.plot(fpr, tpr, label=f'ROC Curve (AUC = {roc\_auc:.2f})', color='blue')

plt.plot([0, 1], [0, 1], 'k--')  # dashed diagonal line for random chance

plt.title('ROC Curve')

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.legend(loc='lower right')

plt.show()