Here’s a structured approach to tackle the assignment step by step:

**Q1: Import and Examine the Dataset**

**Tasks:**

1. Import the dataset using libraries like pandas.
2. Display summary statistics for each variable using .describe() and .info().
3. Use visualizations to understand variable distributions and relationships. Suggested plots:
   * Histograms for individual variable distributions.
   * Pair plots or correlation heatmaps for relationships between variables.
   * Boxplots for detecting outliers.

**Example Code:**

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

# Load the dataset

data = pd.read\_csv("diabetes.csv")

# Descriptive statistics

print(data.info())

print(data.describe())

# Visualizations

sns.histplot(data, x="Glucose", kde=True, bins=30)

plt.title("Glucose Distribution")

plt.show()

sns.pairplot(data, hue="Outcome")

plt.show()

sns.heatmap(data.corr(), annot=True, cmap="coolwarm")

plt.title("Correlation Heatmap")

plt.show()

**Q2: Data Preprocessing**

**Tasks:**

1. **Handle Missing Values**:
   * Replace missing values with mean/median or use imputation techniques.
2. **Outlier Removal**:
   * Use IQR or Z-score methods for continuous variables.
3. **Transform Categorical Variables**:
   * If there are any, convert them into dummy variables using pd.get\_dummies().

**Example Code:**

# Check for missing values

print(data.isnull().sum())

# Replace zero values in critical columns with NaN and handle them

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

data[cols\_with\_zeros] = data[cols\_with\_zeros].replace(0, pd.NA)

# Impute missing values with median

for col in cols\_with\_zeros:

data[col].fillna(data[col].median(), inplace=True)

# Outlier detection using IQR

for col in cols\_with\_zeros:

Q1 = data[col].quantile(0.25)

Q3 = data[col].quantile(0.75)

IQR = Q3 - Q1

data = data[~((data[col] < (Q1 - 1.5 \* IQR)) | (data[col] > (Q3 + 1.5 \* IQR)))]

# Final data check

print(data.info())

**Q3: Split the Dataset**

**Tasks:**

* Use train\_test\_split from sklearn to create training and test sets with a reproducible random seed.

**Example Code:**

from sklearn.model\_selection import train\_test\_split

# Split the data

X = data.drop("Outcome", axis=1)

y = data["Outcome"]

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

**Q4: Train a Decision Tree Model**

**Tasks:**

1. Use DecisionTreeClassifier from sklearn.
2. Optimize hyperparameters using cross-validation (e.g., grid search or random search).

**Example Code:**

from sklearn.tree import DecisionTreeClassifier

from sklearn.model\_selection import GridSearchCV

# Define the model

dt = DecisionTreeClassifier(random\_state=42)

# Hyperparameter tuning

param\_grid = {

"criterion": ["gini", "entropy"],

"max\_depth": [3, 5, 10, None],

"min\_samples\_split": [2, 5, 10]

}

grid\_search = GridSearchCV(estimator=dt, param\_grid=param\_grid, cv=5, scoring="accuracy")

grid\_search.fit(X\_train, y\_train)

# Best model

best\_tree = grid\_search.best\_estimator\_

print("Best Parameters:", grid\_search.best\_params\_)

**Q5: Evaluate the Model**

**Tasks:**

* Calculate metrics like accuracy, precision, recall, and F1 score.
* Plot confusion matrices and ROC curves.

**Example Code:**

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score, confusion\_matrix, roc\_curve, auc

import matplotlib.pyplot as plt

# Predictions

y\_pred = best\_tree.predict(X\_test)

# Metrics

print("Accuracy:", accuracy\_score(y\_test, y\_pred))

print("Precision:", precision\_score(y\_test, y\_pred))

print("Recall:", recall\_score(y\_test, y\_pred))

print("F1 Score:", f1\_score(y\_test, y\_pred))

# Confusion matrix

sns.heatmap(confusion\_matrix(y\_test, y\_pred), annot=True, fmt="d", cmap="Blues")

plt.title("Confusion Matrix")

plt.show()

# ROC curve

y\_pred\_prob = best\_tree.predict\_proba(X\_test)[:, 1]

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

roc\_auc = auc(fpr, tpr)

plt.plot(fpr, tpr, label=f"AUC = {roc\_auc:.2f}")

plt.xlabel("False Positive Rate")

plt.ylabel("True Positive Rate")

plt.title("ROC Curve")

plt.legend()

plt.show()

**Q6: Interpret the Decision Tree**

**Tasks:**

* Visualize the tree using plot\_tree or export it as text.
* Identify key variables and interpret their importance.

**Example Code:**

from sklearn.tree import plot\_tree

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

plot\_tree(best\_tree, feature\_names=X.columns, class\_names=["Non-Diabetic", "Diabetic"], filled=True)

plt.show()

# Feature importance

importances = best\_tree.feature\_importances\_

sorted\_indices = importances.argsort()[::-1]

for idx in sorted\_indices:

print(f"{X.columns[idx]}: {importances[idx]:.2f}")

**Q7: Validate the Model**

**Tasks:**

* Test with new data (if available).
* Conduct sensitivity analysis by altering variables slightly and observing outcomes.

**Example Code:**

# Sensitivity analysis

new\_data = X\_test.copy()

new\_data.iloc[0] += 0.1 # Modify one observation slightly

print("Original Prediction:", best\_tree.predict(X\_test.iloc[0:1]))

print("Modified Prediction:", best\_tree.predict(new\_data.iloc[0:1]))