Hands-on practice on DT and KNN by using sklearn APIs. The notebook should include data preprocessing, model training, evaluation and visualization. Submit your notebook in PDF format to BS by the due date.

You may choose your dataset or use any below:

Iris https://bit.ly/3VqeyM8

Penguins https://bit.ly/3wXu4pE

Diabetes https://bit.ly/data-pi-diabetes

```
# Import necessary libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, classification_report
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.neighbors import KNeighborsClassifier
from sklearn.impute import SimpleImputer
```

```
url = "https://raw.githubusercontent.com/mdogy/dataForEng1999/master/pi_diabetes.csv"
df = pd.read_csv(url)
```

```
cols_to_clean = ['Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI']
df[cols_to_clean] = df[cols_to_clean].replace(0, np.nan)
imputer = SimpleImputer(strategy='median')
df[cols_to_clean] = imputer.fit_transform(df[cols_to_clean])
```

```
X = df[['Glucose', 'BMI']].values  # Using Glucose and BMI for visualization
y = df['Outcome'].values
```

```
print("First 5 rows of the Diabetes dataset:")
print(df[['Glucose', 'BMI', 'Outcome']].head())
```

```
First 5 rows of the Diabetes dataset:
   Glucose   BMI  Outcome
0    148.0  33.6        1
1     85.0  26.6        0
2    183.0  23.3        1
3     89.0  28.1        0
4    137.0  43.1        1
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

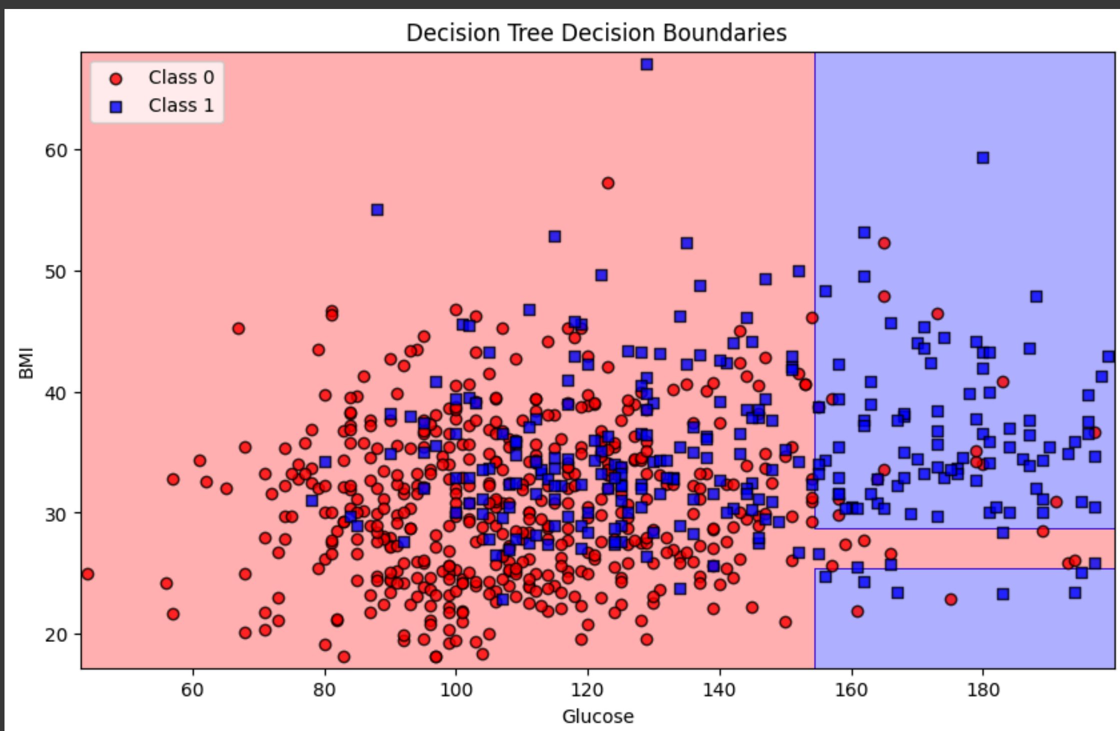
```
def plot_decision_regions(X, y, classifier, test_idx=None, resolution=0.02):
    # Setup markers and colors
    markers = ('o', 's')
    colors = ('red', 'blue')
    cmap = ListedColormap(colors[:len(np.unique(y))])

    # Plot decision surface
    x1_min, x1_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    x2_min, x2_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx1, xx2 = np.meshgrid(np.arange(x1_min, x1_max, resolution),
                          np.arange(x2_min, x2_max, resolution))

    lab = classifier.predict(np.array([xx1.ravel(), xx2.ravel()]).T)
    lab = lab.reshape(xx1.shape)
    plt.contourf(xx1, xx2, lab, alpha=0.3, cmap=cmap)
    plt.xlim(xx1.min(), xx1.max())
    plt.ylim(xx2.min(), xx2.max())

    # Plot class samples
    for idx, cl in enumerate(np.unique(y)):
        plt.scatter(x=X[y == cl, 0],
                    y=X[y == cl, 1],
                    alpha=0.8,
                    c=colors[idx],
                    marker=markers[idx],
```

```
                     label=f'Class {cl}',
                     edgecolor='black')
```

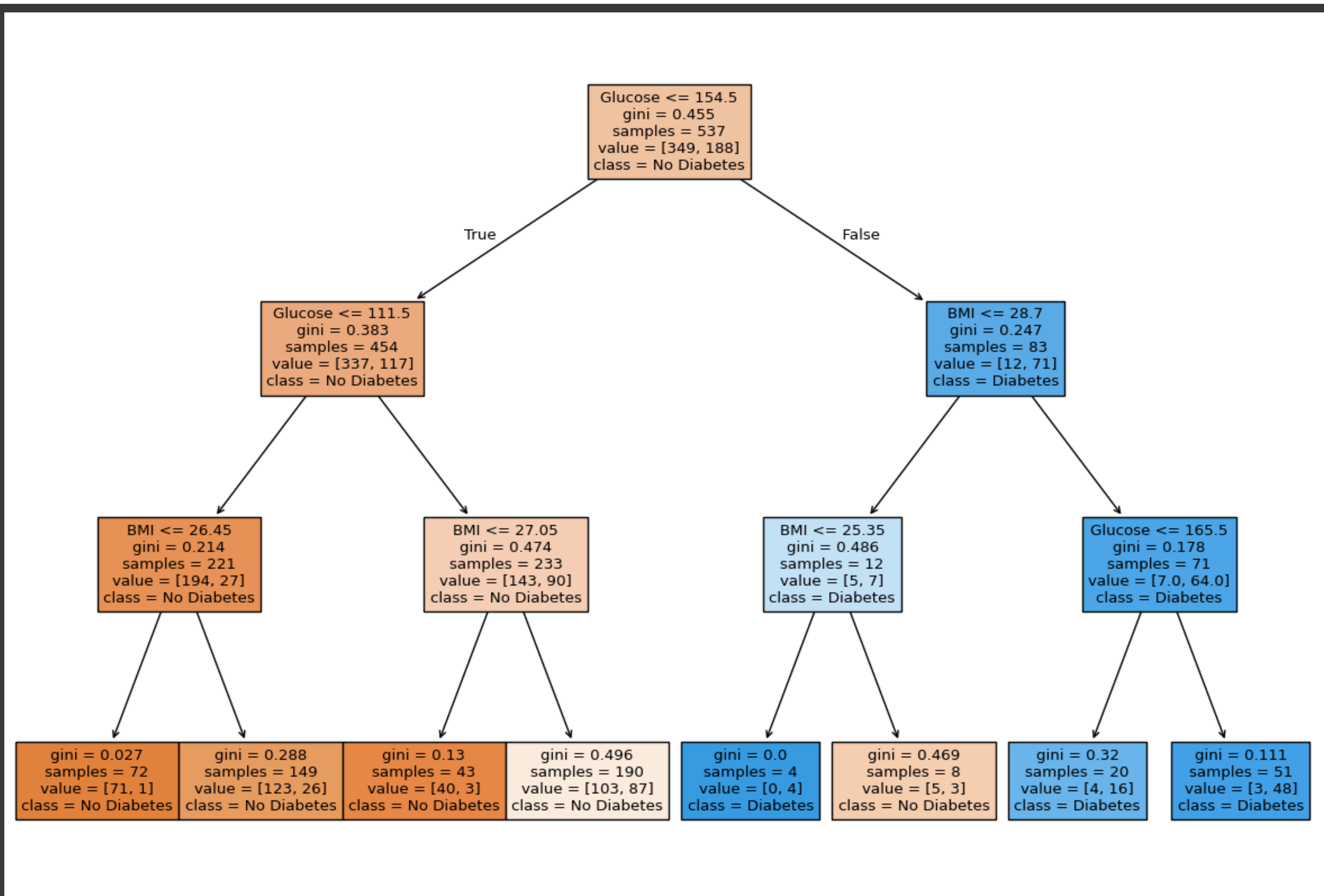
```
tree_model = DecisionTreeClassifier(max_depth=3, random_state=42)
tree_model.fit(X_train, y_train)
```

```
DecisionTreeClassifier
DecisionTreeClassifier(max_depth=3, random_state=42)
```

```
X_combined = np.vstack((X_train, X_test))
y_combined = np.hstack((y_train, y_test))
plt.figure(figsize=(10, 6))
plot_decision_regions(X_combined, y_combined, classifier=tree_model, )
plt.xlabel('Glucose')
plt.ylabel('BMI')
plt.title('Decision Tree Decision Boundaries')
plt.legend(loc='upper left')
plt.show()
```




```
plt.figure(figsize=(15, 10))
plot_tree(tree_model,
          feature_names=['Glucose', 'BMI'],
          class_names=['No Diabetes', 'Diabetes'],
          filled=True)
plt.show()
```




```
y_pred = tree_model.predict(X_test)
print("Decision Tree Performance:")
print(f"Accuracy: {accuracy_score(y_test, y_pred):.4f}")
print(classification_report(y_test, y_pred))
```

```
Decision Tree Performance:
Accuracy: 0.7186
              precision    recall  f1-score   support

           0       0.72      0.93      0.81       151
           1       0.71      0.31      0.43        80

    accuracy                           0.72       231
   macro avg       0.72      0.62      0.62       231
weighted avg       0.72      0.72      0.68       231
```

```
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

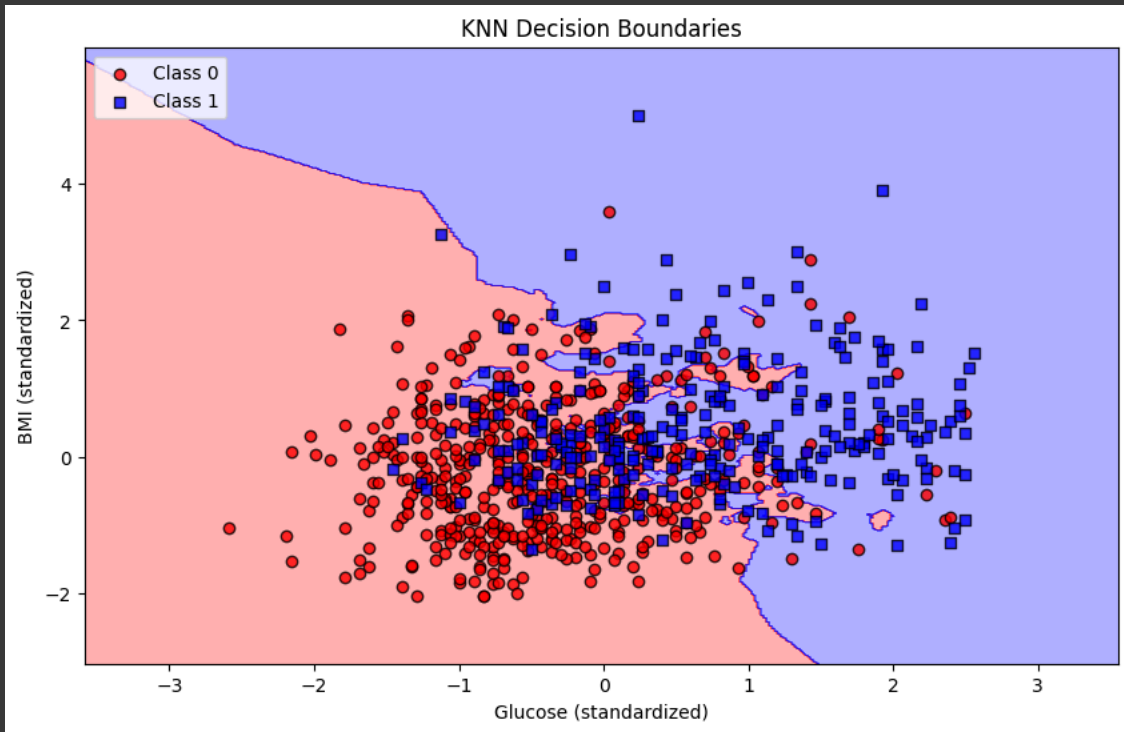
```
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train_scaled, y_train)
```

▾ KNeighborsClassifier

KNeighborsClassifier()

```
plt.figure(figsize=(10, 6))
plot_decision_regions(scaler.transform(X_combined), y_combined, classifier=knn)
plt.xlabel('Glucose (standardized)')
plt.ylabel('BMI (standardized)')
plt.title('KNN Decision Boundaries')
plt.legend(loc='upper left')
```

```
plt.show()
```




```
y_pred_knn = knn.predict(X_test_scaled)
print("\nKNN Performance:")
print(f"Accuracy: {accuracy_score(y_test, y_pred_knn):.4f}")
print(classification_report(y_test, y_pred_knn))
```

```
KNN Performance:
Accuracy: 0.7359
              precision    recall  f1-score   support

           0       0.78      0.83      0.81       151
           1       0.64      0.55      0.59        80

    accuracy                           0.74       231
   macro avg       0.71      0.69      0.70       231
weighted avg       0.73      0.74      0.73       231
```

```
# KNN Decision Boundary Plot (with standardized features)
plt.figure(figsize=(12, 8))

# Create mesh grid in original feature space
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.1),
                     np.arange(y_min, y_max, 0.1))

# Scale mesh grid points using the same scaler
mesh_points = np.c_[xx.ravel(), yy.ravel()]
mesh_points_scaled = scaler.transform(mesh_points)

# Predict class for scaled mesh points
Z = knn.predict(mesh_points_scaled)
Z = Z.reshape(xx.shape)

# Plot decision boundary
plt.contourf(xx, yy, Z, alpha=0.8, cmap=plt.cm.RdYlBu)

# Plot training data (inverse-transformed to original scale)
X_train_plot = scaler.inverse_transform(X_train_scaled)
scatter = plt.scatter(X_train_plot[:, 0], X_train_plot[:, 1], c=y_train,
                      cmap=plt.cm.RdYlBu, edgecolor='k', s=50)
plt.colorbar(scatter, ticks=[0, 1], label='Class (0: No Diabetes, 1: Diabetes)')
plt.xlabel('Glucose (original scale)')
plt.ylabel('BMI (original scale)')
```

```
plt.title('KNN Decision Boundary')
plt.show()
```