```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, recall_score
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer

# Load Dataset
url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv"
# NOTE: We are simulating a 'Stroke' dataset using the Pima dataset structure for stability
names = ['feature1', 'feature2', 'feature3', 'feature4', 'feature5', 'mass', 'pedi', 'age', 'class']
df = pd.read_csv(url, names=names)

print("Environment Ready.")
```

```
Environment Ready.
```

```
X = df.drop('class', axis=1)
Y = df['class']

X_train, X_test, y_train, y_test = train_test_split(
    X, Y, test_size=0.25, random_state=42
)

print("Number of rows in X_test:", len(X_test))
```

```
Number of rows in X_test: 192
```

```
knn_pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('knn', KNeighborsClassifier(n_neighbors=5))
])
```

```
knn_pipeline.fit(X_train, y_train)

y_pred = knn_pipeline.predict(X_test)


accuracy = accuracy_score(y_test, y_pred)
print("Accuracy Score of KNN model:", round(accuracy, 2))
```

```
Accuracy Score of KNN model: 0.68
```

```
recall = recall_score(y_test, y_pred, pos_label=1)
print("Recall Score for positive class:", round(recall, 2))
```

```
Recall Score for positive class: 0.54
```

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

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

```
y_pred = dt_model.predict(X_test)


accuracy = accuracy_score(y_test, y_pred)
print("Accuracy Score of Decision Tree:", round(accuracy, 2))
```

```
Accuracy Score of Decision Tree: 0.71
```

```
importances = dt_model.feature_importances_

max_importance = max(importances)
print("Highest feature importance value:", round(max_importance, 2))
```

```
Highest feature importance value: 0.71
```