1. Dataset Download

```
def install_and_download_dataset():
    import kagglehub
    path = kagglehub.dataset_download("jillanisofttech/brain-stroke-dataset")
    return path

dataset_path = install_and_download_dataset()
print("Dataset path:", dataset_path)
```

```
Downloading from https://www.kaggle.com/api/v1/datasets/download/jillanisofttech/brain-stroke-dataset?dataset_version_number=1...
100%|██████████| 47.2k/47.2k [00:00<00:00, 5.83MB/s]Extracting files...
Dataset path: /root/.cache/kagglehub/datasets/jillanisofttech/brain-stroke-dataset/versions/1
```

2. Data Loading

```
def load_dataset(path):
    import pandas as pd
    csv_path = path + "/brain_stroke.csv"
    dataframe = pd.read_csv(csv_path)
    return dataframe

df = load_dataset(dataset_path)
print(df.head())
```

```
   gender   age  hypertension  heart_disease ever_married      work_type  \
0    Male  67.0             0              1          Yes        Private
1    Male  80.0             0              1          Yes        Private
2  Female  49.0             0              0          Yes        Private
3  Female  79.0             1              0          Yes  Self-employed
4    Male  81.0             0              0          Yes        Private

  Residence_type  avg_glucose_level   bmi   smoking_status  stroke
0          Urban             228.69  36.6  formerly smoked       1
1          Rural             105.92  32.5     never smoked       1
2          Urban             171.23  34.4           smokes       1
3          Rural             174.12  24.0     never smoked       1
4          Urban             186.21  29.0  formerly smoked       1
```

3. Initial Data Exploration

```
def explore_dataset(df):
    print("Data Info:")
    print(df.info())

    print("\nMissing Values:")
    print(df.isnull().sum())

    print("\nStatistical Summary:")
    print(df.describe(include='all'))

explore_dataset(df)
```

```
 10  stroke             4981 non-null   int64
dtypes: float64(3), int64(3), object(5)
memory usage: 428.2+ KB
None

Missing Values:
gender               0
age                  0
hypertension         0
heart_disease        0
```

std	NaN	22.662755	0.294848	0.228412	NaN
min	NaN	0.080000	0.000000	0.000000	NaN
25%	NaN	25.000000	0.000000	0.000000	NaN
50%	NaN	45.000000	0.000000	0.000000	NaN
75%	NaN	61.000000	0.000000	0.000000	NaN
max	NaN	82.000000	1.000000	1.000000	NaN

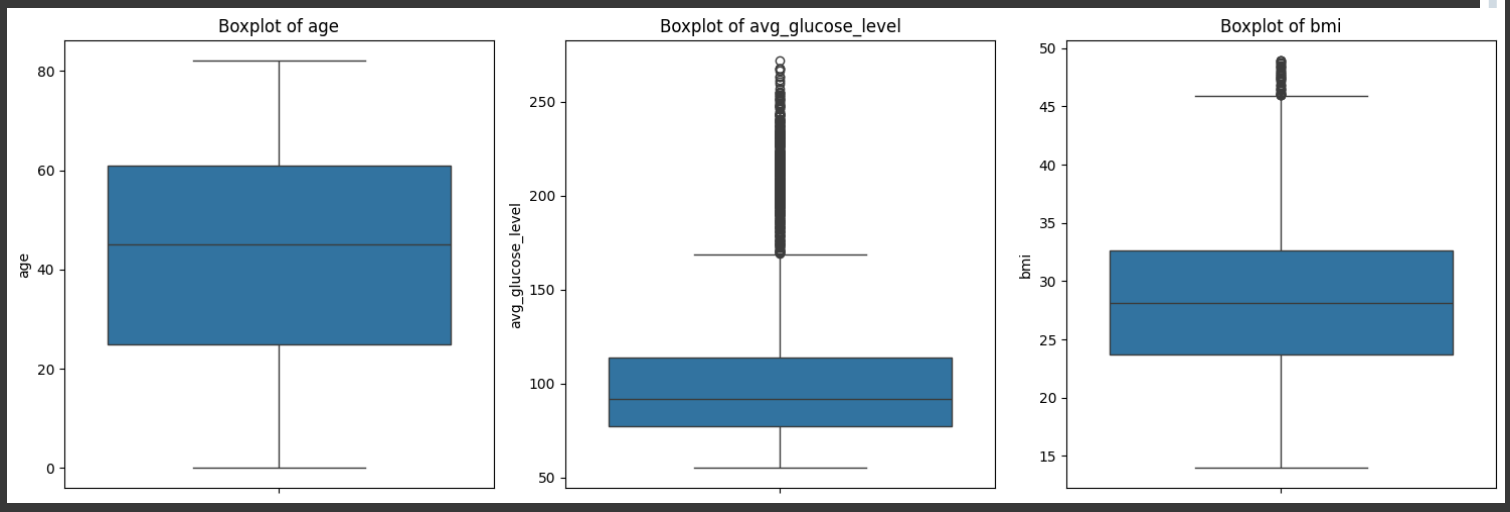
	work_type	Residence_type	avg_glucose_level	bmi	\
count	4981	4981	4981.000000	4981.000000	
unique	4	2	NaN	NaN	
top	Private	Urban	NaN	NaN	
freq	2860	2532	NaN	NaN	
mean	NaN	NaN	105.943562	28.498173	
std	NaN	NaN	45.075373	6.790464	
min	NaN	NaN	55.120000	14.000000	
25%	NaN	NaN	77.230000	23.700000	
50%	NaN	NaN	91.850000	28.100000	
75%	NaN	NaN	113.860000	32.600000	
max	NaN	NaN	271.740000	48.900000	

	smoking_status	stroke
count	4981	4981.000000
unique	4	NaN
top	never smoked	NaN
freq	1838	NaN
mean	NaN	0.049789
std	NaN	0.217531
min	NaN	0.000000
25%	NaN	0.000000
50%	NaN	0.000000
75%	NaN	0.000000
max	NaN	1.000000

4. Data Visualizations

4.1 Box Plots for Numeric Variables

```
def plot_boxplots(df):
    import matplotlib.pyplot as plt
    import seaborn as sns

    numeric_cols = ['age', 'avg_glucose_level', 'bmi']
    plt.figure(figsize=(15, 5))
    for i, col in enumerate(numeric_cols):
        plt.subplot(1, 3, i+1)
        sns.boxplot(data=df, y=col)
        plt.title(f'Boxplot of {col}')
    plt.tight_layout()

plot_boxplots(df)
```



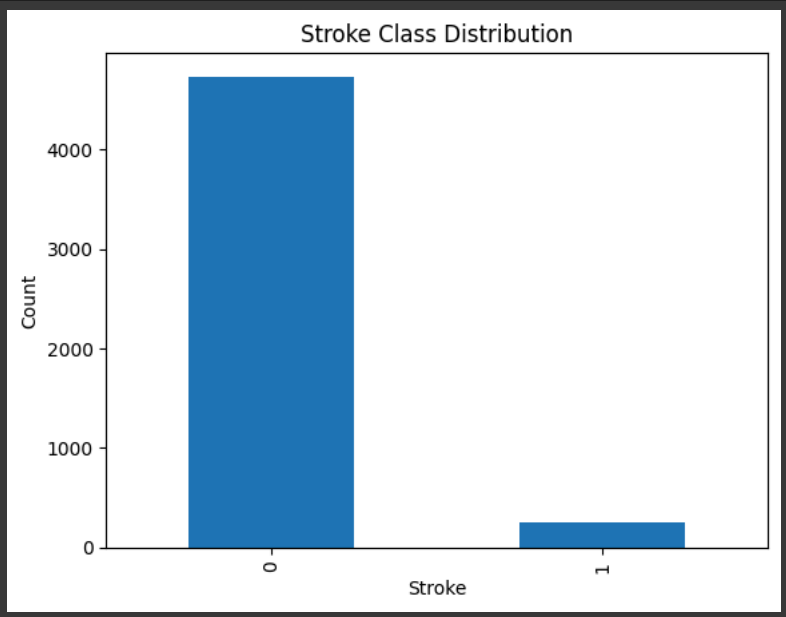
4.2 Correlation Heatmap

```
def plot_correlation_heatmap(df):
    import matplotlib.pyplot as plt
```

```
    import seaborn as sns

    plt.figure(figsize=(10, 6))
    correlation = df.corr(numeric_only=True)
    sns.heatmap(correlation, annot=True, cmap='coolwarm')
    plt.title("Correlation Heatmap")
    plt.show()

plot_correlation_heatmap(df)
```

4.3 Stroke Class Distribution

```
def plot_stroke_distribution(df):
    import matplotlib.pyplot as plt

    stroke_counts = df['stroke'].value_counts()
    stroke_counts.plot(kind='bar', title='Stroke Class Distribution')
    plt.xlabel('Stroke')
    plt.ylabel('Count')
    plt.show()

plot_stroke_distribution(df)
```



5. Preprocessing

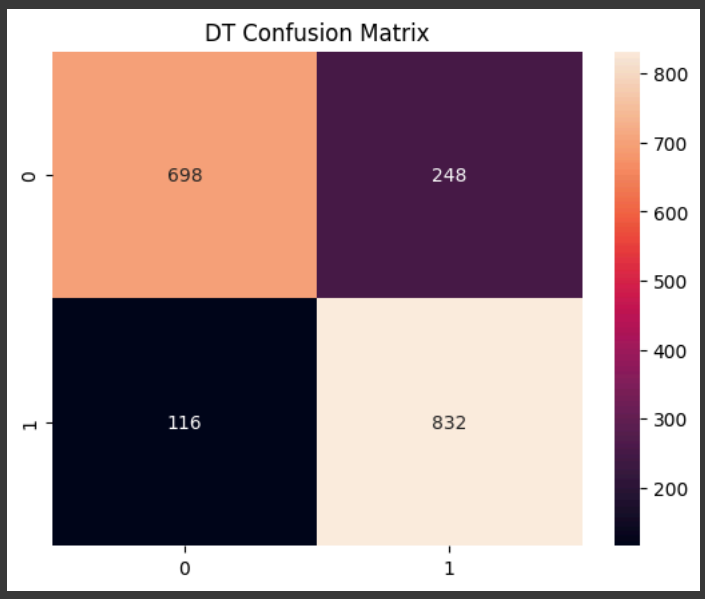
```
def preprocess_data(df):
    from sklearn.model_selection import train_test_split
    from sklearn.preprocessing import LabelEncoder, StandardScaler
    from imblearn.over_sampling import SMOTE
    import pandas as pd # Import pandas here

    if 'id' in df.columns:
        df.drop('id', axis=1, inplace=True)

    df['bmi'] = df['bmi'].fillna(df['bmi'].median())

    binary_columns = ['gender', 'ever_married', 'Residence_type']
    for column in binary_columns:
        if df[column].dtype == 'object':
            encoder = LabelEncoder()
            df[column] = encoder.fit_transform(df[column])

    df = pd.get_dummies(df, columns=['work_type', 'smoking_status'])

    X = df.drop('stroke', axis=1)
    y = df['stroke']

    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X)

    smote = SMOTE(random_state=42)
    X_resampled, y_resampled = smote.fit_resample(X_scaled, y)

    X_train, X_test, y_train, y_test = train_test_split(
        X_resampled, y_resampled, test_size=0.2, random_state=42
    )

    return X, y, X_resampled, y_resampled, X_train, X_test, y_train, y_test

X, y, X_resampled, y_resampled, X_train, X_test, y_train, y_test = preprocess_data(df)

print("Original dataset shape:", X.shape)
print("Resampled dataset shape:", X_resampled.shape)
print("Training set shape:", X_train.shape)
print("Test set shape:", X_test.shape)
```

```
Original dataset shape: (4981, 16)
Resampled dataset shape: (9466, 16)
Training set shape: (7572, 16)
Test set shape: (1894, 16)
```

6. Decision Tree Classifier

```
def decision_tree_classifier(X_train, X_test, y_train, y_test):
    from sklearn.tree import DecisionTreeClassifier
    from sklearn.metrics import accuracy_score, f1_score, classification_report, confusion_matrix
    import seaborn as sns
    import matplotlib.pyplot as plt

    model = DecisionTreeClassifier(max_depth=5, random_state=42)
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)

    print("Decision Tree Classifier")
    print("Accuracy:", accuracy_score(y_test, y_pred))
    print("F1 Score:", f1_score(y_test, y_pred))
    print("Classification Report:\n", classification_report(y_test, y_pred))

    matrix = confusion_matrix(y_test, y_pred)
    sns.heatmap(matrix, annot=True, fmt='d')
    plt.title("DT Confusion Matrix")
    plt.show()

    return y_pred

y_pred_dt = decision_tree_classifier(X_train, X_test, y_train, y_test)
```

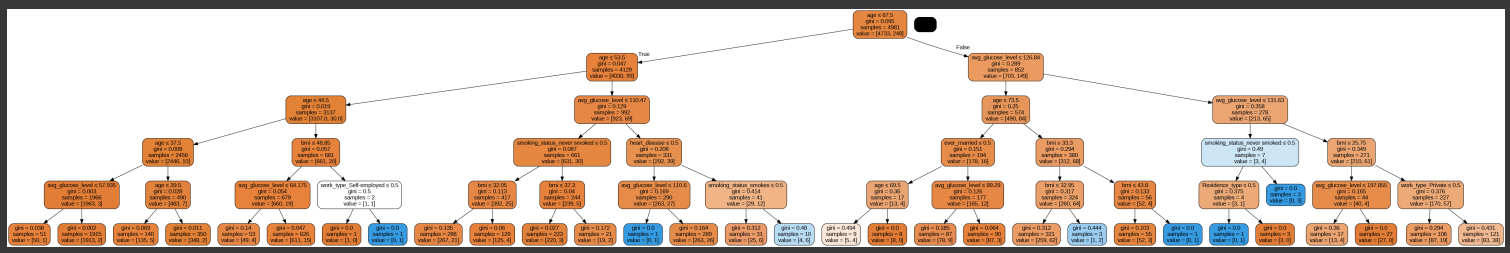
```
Decision Tree Classifier
Accuracy: 0.8078141499472017
F1 Score: 0.8205128205128205
Classification Report:
              precision    recall  f1-score   support

           0       0.86      0.74      0.79       946
           1       0.77      0.88      0.82       948

    accuracy                           0.81      1894
   macro avg       0.81      0.81      0.81      1894
weighted avg       0.81      0.81      0.81      1894
```




```
!apt-get install -y graphviz
!pip install graphviz
```

```
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
graphviz is already the newest version (2.42.2-6ubuntu0.1).
0 upgraded, 0 newly installed, 0 to remove and 35 not upgraded.
Requirement already satisfied: graphviz in /usr/local/lib/python3.11/dist-packages (0.20.3)
```

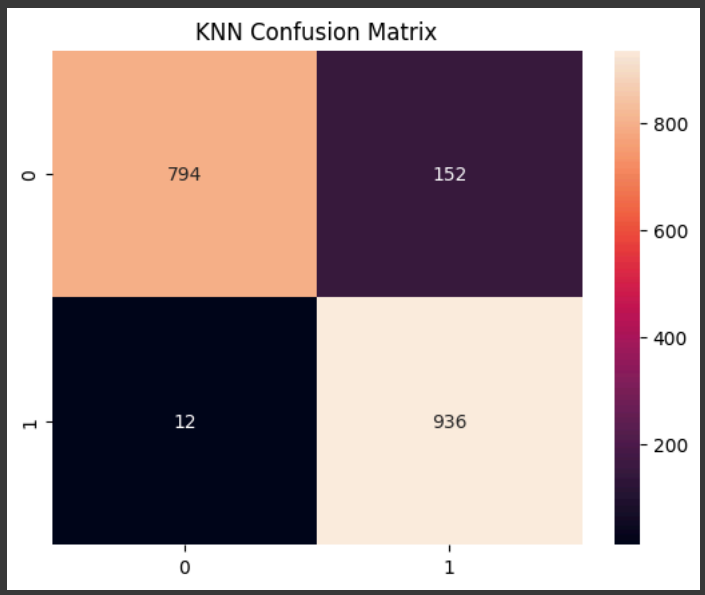
```
#  Preprocess data
df.drop(columns=["id"], inplace=True, errors='ignore')
df.dropna(inplace=True)
df = pd.get_dummies(df)
X, y = df.drop("stroke", axis=1), df["stroke"]

#  Train the model
from sklearn.tree import DecisionTreeClassifier
clf_tree = DecisionTreeClassifier(max_depth=5, random_state=42)
clf_tree.fit(X, y)

# Export to PNG using pydotplus
import pydotplus
from sklearn.tree import export_graphviz

def tree_graph_to_png(tree, feature_names, png_file_to_save):
    """
    Generate a PNG image of the decision tree.
    Requires pydotplus and Graphviz installed.
    """
    dot_data = export_graphviz(
        tree, feature_names=feature_names,
        filled=True, rounded=True, special_characters=True,
        out_file=None
    )
    graph = pydotplus.graph_from_dot_data(dot_data)
    graph.write_png(png_file_to_save)

# Save the tree to a PNG file
tree_graph_to_png(
    tree=clf_tree,
    feature_names=X.columns.tolist(),
    png_file_to_save="stroke_decision_tree.png"
)

# Step 6: Display PNG
from IPython.display import Image
Image(filename="stroke_decision_tree.png")
```



7. K-Nearest Neighbors Classifier

```
def knn_classifier(X_train, X_test, y_train, y_test):
    from sklearn.neighbors import KNeighborsClassifier
    from sklearn.metrics import accuracy_score, f1_score, classification_report, confusion_matrix
    import seaborn as sns
    import matplotlib.pyplot as plt

    model = KNeighborsClassifier(n_neighbors=5)
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)

    print("K-Nearest Neighbors Classifier")
    print("Accuracy:", accuracy_score(y_test, y_pred))
    print("F1 Score:", f1_score(y_test, y_pred))
    print("Classification Report:\n", classification_report(y_test, y_pred))

    matrix = confusion_matrix(y_test, y_pred)
    sns.heatmap(matrix, annot=True, fmt='d')
    plt.title("KNN Confusion Matrix")
    plt.show()

    return y_pred

y_pred_knn = knn_classifier(X_train, X_test, y_train, y_test)
```

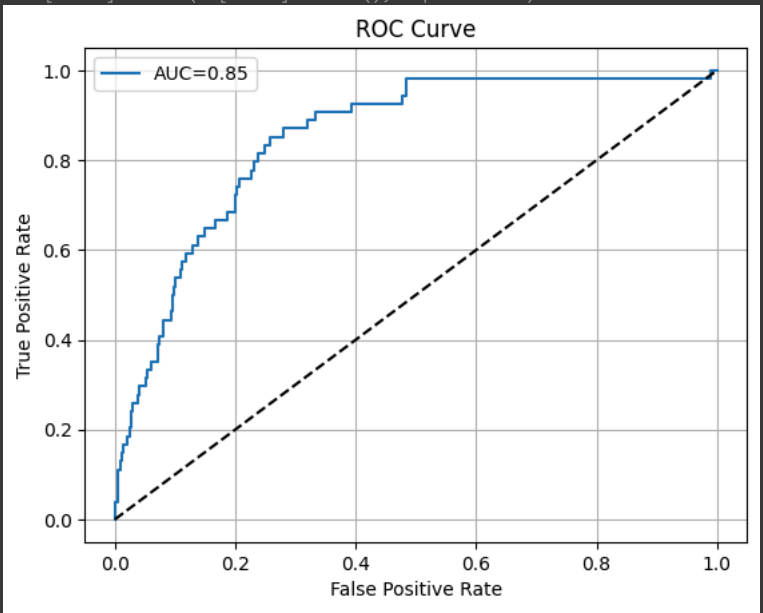
```
K-Nearest Neighbors Classifier
Accuracy: 0.9134107708553326
F1 Score: 0.9194499017681729
Classification Report:
              precision    recall  f1-score   support

           0       0.99      0.84      0.91       946
           1       0.86      0.99      0.92       948

    accuracy                           0.91      1894
   macro avg       0.92      0.91      0.91      1894
weighted avg       0.92      0.91      0.91      1894
```



ROC Curve

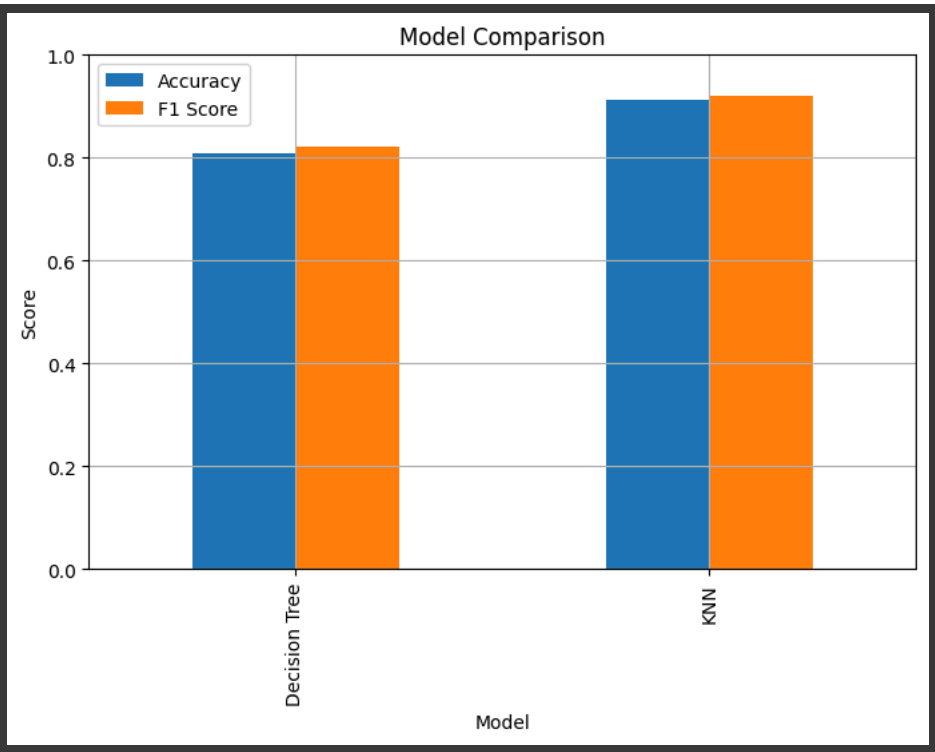
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

df = pd.read_csv(install_and_download_dataset() + "/brain_stroke.csv")
df.drop('id', axis=1, inplace=True, errors='ignore')
df['bmi'].fillna(df['bmi'].median(), inplace=True)
for col in ['gender', 'ever_married', 'Residence_type']:
    df[col] = LabelEncoder().fit_transform(df[col])
df = pd.get_dummies(df, columns=['work_type', 'smoking_status'])
X = StandardScaler().fit_transform(df.drop('stroke', axis=1))
y = df['stroke']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

m = LogisticRegression(max_iter=1000).fit(X_train, y_train)
fpr, tpr, _ = roc_curve(y_test, m.predict_proba(X_test)[:, 1])
plt.plot(fpr, tpr, label=f"AUC={auc(fpr, tpr):.2f}"); plt.plot([0,1],[0,1],'k--')
plt.xlabel("False Positive Rate"); plt.ylabel("True Positive Rate")
plt.title("ROC Curve"); plt.legend(); plt.grid(); plt.show()
```

```
<ipython-input-19-2ae6f036df00>:10: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value

  df['bmi'].fillna(df['bmi'].median(), inplace=True)
```



8. Model Comparison

```
def compare_models(y_test, y_pred_dt, y_pred_knn):
    from sklearn.metrics import accuracy_score, f1_score
    import pandas as pd
    import matplotlib.pyplot as plt

    accuracy_dt = accuracy_score(y_test, y_pred_dt)
    f1_dt = f1_score(y_test, y_pred_dt)

    accuracy_knn = accuracy_score(y_test, y_pred_knn)
    f1_knn = f1_score(y_test, y_pred_knn)

    results_df = pd.DataFrame({
        'Model': ['Decision Tree', 'KNN'],
        'Accuracy': [accuracy_dt, accuracy_knn],
        'F1 Score': [f1_dt, f1_knn]
    })

    results_df.set_index('Model').plot(kind='bar', figsize=(8, 5), legend=True, title="Model Comparison")
    plt.ylabel("Score")
    plt.ylim(0, 1)
    plt.grid(True)
    plt.show()

compare_models(y_test, y_pred_dt, y_pred_knn)
```


Model Comparison
Accuracy
F1 Score
1.0
0.8
0.6
0.4
0.2
0.0
Score
Decision Tree
KNN
Model