```
import numpy as np

import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import (

    accuracy_score, confusion_matrix, precision_score,

    recall_score, f1_score, mean_squared_error

)

from collections import Counter


# Step 1: Load dataset

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

columns = ['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness',

           'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome']

data = pd.read_csv(url, header=None, names=columns)

print(data.head())


# Step 2: Replace 0 with NaN for certain columns, then fill with mean

cols_with_zero = ['Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI']

data[cols_with_zero] = data[cols_with_zero].replace(0, np.nan)

data.fillna(data.mean(), inplace=True)


# Step 3: Split into features and labels

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

y = data['Outcome']


# Step 4: Normalize features

scaler = StandardScaler()

X_scaled = scaler.fit_transform(X)
```

```
# Step 5: Split into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y, test_size=0.3, random_state=42, stratify=y)

# Step 6: Define KNN (Euclidean and Manhattan)
def euclidean_distance(a, b):
    return np.sqrt(np.sum((a - b) ** 2))

def knn_predict(X_train, y_train, X_test, k=5, distance_metric='euclidean'):
    predictions = []
    for test_point in X_test:
        distances = []
        for i in range(len(X_train)):
            if distance_metric == 'euclidean':
                dist = euclidean_distance(test_point, X_train[i])
            elif distance_metric == 'manhattan':
                dist = np.sum(np.abs(test_point - X_train[i]))
            else:
                raise ValueError("Unsupported distance metric")
            distances.append((dist, y_train.iloc[i]))
        distances.sort(key=lambda x: x[0])
        neighbors = [distances[i][1] for i in range(k)]
        most_common = Counter(neighbors).most_common(1)
        predictions.append(most_common[0][0])
    return np.array(predictions)

# Step 7: Evaluation Function
def evaluate_model(y_test, y_pred, k, metric_name):
    acc = accuracy_score(y_test, y_pred)
    prec = precision_score(y_test, y_pred, zero_division=0)
    rec = recall_score(y_test, y_pred, zero_division=0)
```

```python
    f1 = f1_score(y_test, y_pred, zero_division=0)
    mse = mean_squared_error(y_test, y_pred)
    rmse = np.sqrt(mse)
    cm = confusion_matrix(y_test, y_pred, labels=[0, 1])
    print(f"\nKNN Evaluation - K={k}, Distance: {metric_name}")
    print("Accuracy:", round(acc, 4))
    print("Precision:", round(prec, 4))
    print("Recall:", round(rec, 4))
    print("F1 Score:", round(f1, 4))
    print("MSE:", round(mse, 4))
    print("RMSE:", round(rmse, 4))
    print("Confusion Matrix:\n", cm)

# Step 8: Run for different k values and distances
for k in [3, 5, 7]:
    # Euclidean
    pred_euc = knn_predict(X_train, y_train, X_test, k, distance_metric='euclidean')
    evaluate_model(y_test, pred_euc, k, "Euclidean")
    # Manhattan
    pred_man = knn_predict(X_train, y_train, X_test, k, distance_metric='manhattan')
    evaluate_model(y_test, pred_man, k, "Manhattan")
```