# Assignment No. 4

**Implement K-Nearest Neighbors algorithm on diabetes.csv dataset. Compute confusion matrix, accuracy, error rate, precision and recall on the given dataset.**

### Step 1: Download the dataset from Kaggle

1. **Dataset link:** Diabetes Dataset
2. Go to the Kaggle dataset page.
3. Download the diabetes.csv file.
4. Save the dataset in the directory where you will run the Jupyter notebook.

---

### Step 2: Open Jupyter Notebook

1. **Open Jupyter Notebook:**
   - Launch **Jupyter Notebook**.
   - Navigate to the directory where you saved the diabetes.csv file.
   - Create a new Python notebook.

---

### Step 3: Import necessary libraries

```
import pandas as pd

import numpy as np

from sklearn.model_selection import train_test_split

from sklearn.svm import SVC

from sklearn.neighbors import KNeighborsClassifier

from sklearn import metrics
```

- **Explanation**: These libraries will help load the dataset, split the data, and apply the KNN classifier.

---

### Step 4: Load and inspect the dataset

```
# Load the dataset

diabetes_data = pd.read_csv('diabetes.csv')

# Display the first 5 rows of the dataframe

diabetes_data.head()
```

- **Explanation**: The dataset is loaded into a pandas DataFrame diabetes_data, and head() displays the first 5 rows for inspection.

```
diabetes_data.shape
```

- **Explanation**: This outputs the shape of the dataset, i.e., the number of rows and columns.

```
diabetes_data.describe()
```

- **Explanation**: This provides summary statistics of the numerical columns in the dataset.

---

### Step 5: Define features (X) and target (Y)

```
# Drop the 'Outcome' column from the feature set
X = diabetes_data.drop(columns='Outcome', axis=1)


# Display the first 5 rows of X
X.head()
```

- **Explanation**: X contains all the features except the 'Outcome' column, which represents the target.

```
# Define the target variable
Y = diabetes_data['Outcome']
```

- **Explanation**: Y contains the target variable, 'Outcome', which we are trying to predict (whether the patient has diabetes or not).

---

### Step 6: Split the data into training and testing sets

```
x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size=0.2, random_state=1)
```

- **Explanation**: The dataset is split into training and testing sets using train_test_split(), with 80% of the data used for training and 20% for testing. random_state=1 ensures the split is consistent every time the code is run.

---

### Step 7: Train and apply the KNN classifier

```
# Initialize and train a KNN classifier with 7 neighbors
KN = KNeighborsClassifier
knn = KN(n_neighbors=7)
```

```
knn.fit(x_train, y_train)


# Make predictions on the test set
y_pred = knn.predict(x_test)
print("Prediction: \n")
print(y_pred)
```

- **Explanation**:
  - The KNN classifier is initialized with 7 neighbors (n_neighbors=7), meaning it will consider the 7 nearest neighbors to classify each test data point.
  - The model is trained on x_train and y_train, and predictions are made on x_test.

---

**Step 8: Evaluate the model performance**

**1. Confusion Matrix**

```
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score


# Compute the confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:\n", conf_matrix)
```

- **Explanation**:
  - The confusion matrix is computed using confusion_matrix(). It shows how well the model predicted the classes:
    - **True Negatives (TN)**: Number of cases correctly classified as not having diabetes.
    - **False Positives (FP)**: Number of cases incorrectly classified as having diabetes.
    - **False Negatives (FN)**: Number of cases incorrectly classified as not having diabetes.
    - **True Positives (TP)**: Number of cases correctly classified as having diabetes.

**2. Accuracy**

```
# Compute accuracy
```

```
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

### 3. Error Rate

```
# Compute error rate
error_rate = 1 - accuracy
print("Error Rate:", error_rate)
```

### 4. Precision

```
# Compute precision
precision = precision_score(y_test, y_pred)
print("Precision:", precision)
```

### 5. Recall

```
# Compute recall
recall = recall_score(y_test, y_pred)
print("Recall:", recall)
```