Problem - 1: Perform a classification task with knn from scratch.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

print("\n")
print("1.  Load the Dataset")
print("\n")

data = pd.read_csv("/content/drive/MyDrive/Untitled folder/diabetes_.csv")

print(data.head())
print(data.info())
print(data.describe())
print(data.isnull().sum())


print("\n")
print("2. Handle Missing Data")
print("\n")

for column in data.columns:
    if data[column].isnull().sum() > 0:
        data[column].fillna(data[column].mean(), inplace=True)

print(data.isnull().sum())


print("\n")
print("3. Feature Engineering")
print("\n")

X = data.drop(columns=["Outcome"]).values
y = data["Outcome"].values


def train_test_split_scratch(X, y, test_size=0.3, random_seed=42):
    np.random.seed(random_seed)
    indices = np.arange(len(X))
    np.random.shuffle(indices)

    split = int(len(X) * test_size)
    test_idx = indices[:split]
    train_idx = indices[split:]

    return X[train_idx], X[test_idx], y[train_idx], y[test_idx]


X_train, X_test, y_train, y_test = train_test_split_scratch(X, y)

print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)


print("\n")
print("4. Implement KNN")
print("\n")


def euclidean_distance(p1, p2):
    if p1.shape != p2.shape:
        raise ValueError("Dimension mismatch")
    return np.sqrt(np.sum((p1 - p2) ** 2))


def knn_predict_single(query, X_train, y_train, k):
    distances = []

    for i in range(len(X_train)):
        dist = euclidean_distance(query, X_train[i])
        distances.append(dist)
```

```
    nearest_indices = np.argsort(distances)[:k]
    nearest_labels = y_train[nearest_indices]

    prediction = np.bincount(nearest_labels).argmax()
    return prediction


def knn_predict(X_test, X_train, y_train, k):
    predictions = []
    for x in X_test:
        predictions.append(knn_predict_single(x, X_train, y_train, k))
    return np.array(predictions)


def compute_accuracy(y_true, y_pred):
    return (np.sum(y_true == y_pred) / len(y_true)) * 100


predictions = knn_predict(X_test, X_train, y_train, k=3)
accuracy = compute_accuracy(y_test, predictions)

print(f"Accuracy of kNN (k=3) on Unscaled Data: {accuracy:.2f}%")
```

```
count   768.000000  768.000000     768.000000     768.000000  768.000000
mean      3.845052  120.894531      69.105469      20.536458   79.799479
std       3.369578   31.972618      19.355807      15.952218  115.244002
min       0.000000    0.000000       0.000000       0.000000    0.000000
25%       1.000000   99.000000      62.000000       0.000000    0.000000
50%       3.000000  117.000000      72.000000      23.000000   30.500000
75%       6.000000  140.250000      80.000000      32.000000  127.250000
max      17.000000  199.000000     122.000000      99.000000  846.000000

              BMI  DiabetesPedigreeFunction         Age     Outcome
count  768.000000                768.000000  768.000000  768.000000
mean    31.992578                  0.471876   33.240885    0.348958
std      7.884160                  0.331329   11.760232    0.476951
min      0.000000                  0.078000   21.000000    0.000000
25%     27.300000                  0.243750   24.000000    0.000000
50%     32.000000                  0.372500   29.000000    0.000000
75%     36.600000                  0.626250   41.000000    1.000000
max     67.100000                  2.420000   81.000000    1.000000
Pregnancies                 0
Glucose                     0
BloodPressure               0
```

```
Accuracy of kNN (k=3) on Unscaled Data: 67.20%
```

Problem - 2 - Experimentation:

```
def min_max_scaling(X):
    X_min = X.min(axis=0)
    X_max = X.max(axis=0)
    return (X - X_min) / (X_max - X_min)


X_scaled = min_max_scaling(X)


X_train, X_test, y_train, y_test = train_test_split_scratch(X, y)

pred_unscaled = knn_predict(X_test, X_train, y_train, k=3)
accuracy_unscaled = compute_accuracy(y_test, pred_unscaled)



X_train_s, X_test_s, y_train_s, y_test_s = train_test_split_scratch(X_scaled, y)

pred_scaled = knn_predict(X_test_s, X_train_s, y_train_s, k=3)
accuracy_scaled = compute_accuracy(y_test_s, pred_scaled)

print(f"Accuracy on Unscaled Data (k=3): {accuracy_unscaled:.2f}%")
print(f"Accuracy on Scaled Data   (k=3): {accuracy_scaled:.2f}%")
```

```
Accuracy on Unscaled Data (k=3): 67.39%
Accuracy on Scaled Data   (k=3): 69.13%
```

Problem - 3 - Experimentation with k:

```
import time


def evaluate_knn_k_values(X_train, y_train, X_test, y_test, k_values):
    accuracies = []
    times = []

    for k in k_values:
        start_time = time.time()

        predictions = knn_predict(X_test, X_train, y_train, k)

        end_time = time.time()

        accuracy = compute_accuracy(y_test, predictions)
        time_taken = end_time - start_time

        accuracies.append(accuracy)
        times.append(time_taken)

        print(f"k={k} | Accuracy={accuracy:.2f}% | Time={time_taken:.6f}s")

    return accuracies, times



k_values = range(1, 16)

print("----- Original Dataset Results -----")
acc_original, time_original = evaluate_knn_k_values(
    X_train, y_train, X_test, y_test, k_values
)



print("\n----- Scaled Dataset Results -----")
acc_scaled, time_scaled = evaluate_knn_k_values(
    X_train_s, y_train_s, X_test_s, y_test_s, k_values
)
```

```
import matplotlib.pyplot as plt

plt.figure()
plt.plot(k_values, acc_original, marker='o', label="Original Data")
plt.plot(k_values, acc_scaled, marker='o', label="Scaled Data")
plt.xlabel("k (Number of Neighbors)")
plt.ylabel("Accuracy (%)")
plt.title("k vs Accuracy")
plt.legend()
plt.grid()
plt.show()



plt.figure()
plt.plot(k_values, time_original, marker='o', label="Original Data")
plt.plot(k_values, time_scaled, marker='o', label="Scaled Data")
plt.xlabel("k (Number of Neighbors)")
plt.ylabel("Prediction Time (seconds)")
plt.title("k vs Prediction Time")
plt.legend()
plt.grid()
plt.show()



best_k_original = k_values[acc_original.index(max(acc_original))]
best_k_scaled = k_values[acc_scaled.index(max(acc_scaled))]

print("----- Optimal k Values -----")
print(f"Best k (Original Data): {best_k_original}")
print(f"Best k (Scaled Data): {best_k_scaled}")
```

```
----- Original Dataset Results -----
k=1 | Accuracy=68.70% | Time=0.939347s
k=2 | Accuracy=72.61% | Time=1.522459s
k=3 | Accuracy=67.39% | Time=1.515273s
k=4 | Accuracy=72.17% | Time=1.300493s
k=5 | Accuracy=68.70% | Time=0.908821s
k=6 | Accuracy=70.00% | Time=0.947219s
k=7 | Accuracy=69.13% | Time=0.911161s
k=8 | Accuracy=71.30% | Time=0.862743s
k=9 | Accuracy=70.00% | Time=0.916247s
k=10 | Accuracy=71.74% | Time=2.041885s
k=11 | Accuracy=73.04% | Time=0.984937s
k=12 | Accuracy=73.91% | Time=0.872694s
k=13 | Accuracy=74.35% | Time=0.910316s
k=14 | Accuracy=72.61% | Time=1.294807s
k=15 | Accuracy=73.91% | Time=1.554614s

----- Scaled Dataset Results -----
k=1 | Accuracy=67.39% | Time=1.528401s
k=2 | Accuracy=67.83% | Time=0.966769s
k=3 | Accuracy=69.13% | Time=0.888378s
k=4 | Accuracy=69.57% | Time=0.908580s
k=5 | Accuracy=67.83% | Time=0.881157s
k=6 | Accuracy=66.52% | Time=0.885844s
k=7 | Accuracy=71.30% | Time=0.877103s
k=8 | Accuracy=69.57% | Time=0.894743s
k=9 | Accuracy=72.17% | Time=0.900736s
k=10 | Accuracy=70.43% | Time=0.898081s
k=11 | Accuracy=70.00% | Time=0.919598s
k=12 | Accuracy=68.70% | Time=0.912148s
k=13 | Accuracy=69.13% | Time=1.404951s
k=14 | Accuracy=68.70% | Time=1.497566s
k=15 | Accuracy=70.87% | Time=1.508878s
```