```python
import pandas as pd
```

```python
# 1. Consider the toy dataset below which shows if 4 subjects have diabetes or
not, along with two diagnostic measurements. (Note: do NOT write any code for
this problem. The answers are to be computed by hand.)
```

```python
# a. The "Class" variable is HasDiabetes, which indicates whether a subject has
diabetes (Yes or No).
# b. Normalize the Preg and Glucose values by scaling the minimum-maximum range
of each column to 0-1. Fill in the empty columns in the table.
'''
1. Preg.Norm
  Row 1: (2 - 1) / (3 - 1) = 0.5
  Row 2: (3 - 1) / (3 - 1) = 1.0
  Row 3: (2 - 1) / (3 - 1) = 0.5
  Row 4: (1 - 1) / (3 - 1) = 0.0
  Row 5: (2 - 1) / (3 - 1) = 0.5

2. Glucose.Norm
  Row 1: (157 - 77) / (174 - 77) = 0.8247
  Row 2: (174 - 77) / (174 - 77) = 1.0
  Row 3: (105 - 77) / (174 - 77) = 0.2887
  Row 4: (77 - 77) / (174 - 77) = 0.0
  Row 5: (94 - 77) / (174 - 77) = 0.1752
'''
# c. Predict whether a subject with Preg=2, Glucose=94 will have diabetes using
the 1-NN algorithm and
# i. Using Euclidean distance on the original variables
'''
distance = √((Preg - 2)^2 + (Glucose - 94)^2)

Row 1: √((2 - 2)^2 + (157 - 94)^2) = √(0 + 63^2) = 63.0
Row 2: √((3 - 2)^2 + (174 - 94)^2) = √(1 + 80^2)   80.0063
Row 3: √((2 - 2)^2 + (105 - 94)^2) = √(0 + 11^2) = 11.0
Row 4: √((1 - 2)^2 + (77 - 94)^2) = √(1 + 17^2)   17.0294

Nearest Neighbor: Row 3
Prediction: Yes (HasDiabetes = Yes for Row 3)
'''
# ii. Using Euclidean distance on the normalized variables
'''
Target (Preg.Norm = 0.5, Glucose.Norm = 0.1752)

Row 1: √((0.5 - 0.5)^2 + (0.8247 - 0.1752)^2)   0.6495
Row 2: √((1.0 - 0.5)^2 + (1.0 - 0.1752)^2)   0.9351
Row 3: √((0.5 - 0.5)^2 + (0.2887 - 0.1752)^2)   0.1135
```

```
Row 4: √((0.0 - 0.5)^2 + (0.0 - 0.1752)^2)   0.5293

Nearest Neighbor: Row 3
Prediction: Yes (HasDiabetes = Yes for Row 3)
'''
```

'\nTarget (Preg.Norm = 0.5, Glucose.Norm = 0.1752)\n\nRow 1: √((0.5 - 0.5)^2 + (0.8247 - 0.1752)^2)   0.6495\nRow 2: √((1.0 - 0.5)^2 + (1.0 - 0.1752)^2) 0.9351\nRow 3: √((0.5 - 0.5)^2 + (0.2887 - 0.1752)^2)   0.1135\nRow 4: √((0.0 - 0.5)^2 + (0.0 - 0.1752)^2)   0.5293\n\nNearest Neighbor: Row 3\nPrediction: Yes (HasDiabetes = Yes for Row 3)\n'

```
# 2. The pima-indians-diabetes-resampled.csv file on Canvas contains records indicating whether the subjects have diabetes or not, along with certain diagnostic measurements. All subjects are of Pima Indian heritage and this dataset is called the Pima Indian Diabetes Database. The goal is to see if it is possible to predict if a subject has diabetes given some of the diagnostic measurements. (Note: this problem is an extension of the classwork assignment; Python code from the class is also posted on Canvas.)
# a. Read the data file
data = pd.read_csv('pima-indians-diabetes-resampled.csv')
# b. What does "Preg" represent in the dataset? (2-3 sentences. Search for the Pima Indian Diabetes Database online. Its background and the ethics issues it raises are also important.
# ANSWER: "Preg" represents the number of times a subject has been pregnant. In the context of the Pima Indian Diabetes Database, it serves as one of the diagnostic features used to predict diabetes. This dataset, collected from a specific population, raises ethical questions about the representation and health privacy of Indigenous communities.
# c. 0 values in the Glucose column indicate missing values. Remove rows which contain missing values in the Glucose column. You should have 763 rows. [code]
data['Glucose'].replace(0, pd.NA, inplace=True)
data = data.dropna(subset=['Glucose'])
print(data)
```

```
     Preg  Glucose  BP  Skin  Insulin   BMI  Pedigree  Age  HasDiabetes
0       2      157  74    35      440  39.4     0.134   30            0
1       7      159  64     0        0  27.4     0.294   40            0
2       7       83  78    26       71  29.3     0.767   36            0
3       0      124  56    13      105  21.8     0.452   21            0
4       5       99  54    28       83  34.0     0.499   30            0
..    ...      ...  ..   ...      ...   ...       ...  ...          ...
763     7       81  78    40       48  46.7     0.261   42            0
764     6      125  78    31        0  27.6     0.565   49            1
765     3      130  78    23       79  28.4     0.323   34            1
766     1      116  78    29      180  36.1     0.496   25            0
767     3      173  78    39      185  33.8     0.970   31            1
```

```
[763 rows x 9 columns]

FutureWarning: A value is trying to be set on a copy of a DataFrame or Series
through chained assignment using an inplace method.
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 behaves as
a copy.

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)
instead, to perform the operation inplace on the original object.


  data['Glucose'].replace(0, pd.NA, inplace=True)
```

```
# d. Create three new columns/variables which are the normalized versions of
Preg, Pedigree, and Glucose columns, scaling the minimum-maximum range of each
column to 0-1 (you can use the code developed in class). [code]
data['Preg.Norm'] = (data['Preg'] - data['Preg'].min()) / (data['Preg'].max() -
data['Preg'].min())
data['Pedigree.Norm'] = (data['Pedigree'] - data['Pedigree'].min()) /
(data['Pedigree'].max() - data['Pedigree'].min())
data['Glucose.Norm'] = (data['Glucose'] - data['Glucose'].min()) /
(data['Glucose'].max() - data['Glucose'].min())
print(data)
```

```
     Preg Glucose  BP  Skin  Insulin   BMI  Pedigree  Age  HasDiabetes  \
0       2     157  74    35      440  39.4     0.134   30            0
1       7     159  64     0        0  27.4     0.294   40            0
2       7      83  78    26       71  29.3     0.767   36            0
3       0     124  56    13      105  21.8     0.452   21            0
4       5      99  54    28       83  34.0     0.499   30            0
..    ...     ...  ..   ...      ...   ...       ...  ...          ...
763     7      81  78    40       48  46.7     0.261   42            0
764     6     125  78    31        0  27.6     0.565   49            1
765     3     130  78    23       79  28.4     0.323   34            1
766     1     116  78    29      180  36.1     0.496   25            0
767     3     173  78    39      185  33.8     0.970   31            1

     Preg.Norm  Pedigree.Norm Glucose.Norm
0     0.117647       0.023911     0.729032
1     0.411765       0.092229     0.741935
2     0.411765       0.294193     0.251613
3     0.000000       0.159693     0.516129
4     0.294118       0.179761     0.354839
..         ...            ...          ...
763   0.411765       0.078138      0.23871
```

```
764    0.352941        0.207942     0.522581
765    0.176471        0.104611     0.554839
766    0.058824        0.178480     0.464516
767    0.176471        0.380871     0.832258

[763 rows x 12 columns]
```

```
# e. Split the dataset into train and test datasets with the first 500 rows for
training, and the remaining rows for test. Do NOT randomly sample the data
(though resampling is usually done, this hw problem does not use this step for
ease of grading).
train_data = data.iloc[:500]
test_data = data.iloc[500:]
print(train_data)
print(test_data)
```

```
     Preg  Glucose  BP  Skin  Insulin   BMI  Pedigree  Age  HasDiabetes  \
0       2      157  74    35      440  39.4     0.134   30            0
1       7      159  64     0        0  27.4     0.294   40            0
2       7       83  78    26       71  29.3     0.767   36            0
3       0      124  56    13      105  21.8     0.452   21            0
4       5       99  54    28       83  34.0     0.499   30            0
..    ...      ...  ..   ...      ...   ...       ...  ...          ...
500     8      120  86     0        0  28.4     0.259   22            1
501     2       99   0     0        0  22.2     0.108   23            0
502     8      196  76    29      280  37.5     0.605   57            1
503     2       83  65    28       66  36.8     0.629   24            0
504     1       88  78    29       76  32.0     0.365   29            0

     Preg.Norm  Pedigree.Norm Glucose.Norm
0     0.117647       0.023911     0.729032
1     0.411765       0.092229     0.741935
2     0.411765       0.294193     0.251613
3     0.000000       0.159693     0.516129
4     0.294118       0.179761     0.354839
..         ...            ...          ...
500   0.470588       0.077284     0.490323
501   0.117647       0.012810     0.354839
502   0.470588       0.225021     0.980645
503   0.117647       0.235269     0.251613
504   0.058824       0.122545     0.283871

[500 rows x 12 columns]
     Preg  Glucose  BP  Skin  Insulin   BMI  Pedigree  Age  HasDiabetes  \
505     0      100  88    60      110  46.8     0.962   31            0
506     2       81  72    15       76  30.1     0.547   25            0
507     6      102  90    39        0  35.7     0.674   28            0
```

```
508     9     156  86     0       0  24.8     0.230   53             1
509     6     129  90     7     326  19.6     0.582   60             0
..    ...     ...  ..   ...     ...   ...       ...  ...           ...
763     7      81  78    40      48  46.7     0.261   42             0
764     6     125  78    31       0  27.6     0.565   49             1
765     3     130  78    23      79  28.4     0.323   34             1
766     1     116  78    29     180  36.1     0.496   25             0
767     3     173  78    39     185  33.8     0.970   31             1

     Preg.Norm  Pedigree.Norm Glucose.Norm
505   0.000000       0.377455      0.36129
506   0.117647       0.200256      0.23871
507   0.352941       0.254483     0.374194
508   0.529412       0.064902     0.722581
509   0.352941       0.215201     0.548387
..         ...            ...          ...
763   0.411765       0.078138      0.23871
764   0.352941       0.207942     0.522581
765   0.176471       0.104611     0.554839
766   0.058824       0.178480     0.464516
767   0.176471       0.380871     0.832258

[263 rows x 12 columns]
```

```python
# f. Train and test a k-nearest neighbor classifier with the dataset. Consider
only the normalized Preg and Pedigree columns. Set k=1. What is the error rate
(number of misclassifications)? [code, error rate]
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score


knn_1 = KNeighborsClassifier(n_neighbors=1)
knn_1.fit(train_data[['Preg.Norm', 'Pedigree.Norm']], train_data['HasDiabetes'])


predictions_1 = knn_1.predict(test_data[['Preg.Norm', 'Pedigree.Norm']])
error_rate_1 = 1 - accuracy_score(test_data['HasDiabetes'], predictions_1)

print(error_rate_1)
```

```
0.40684410646387836
```

```python
# g. Repeat part (f) but consider the normalized Preg, Pedigree, and Glucose
columns. Set k=1. What is the error rate? Will the error rate always decrease
with a larger number of features? Why or why not: answer in 2-3 sentences? [code,
error rate, answer]
```

```
knn_1_with_glucose = KNeighborsClassifier(n_neighbors=1)
knn_1_with_glucose.fit(train_data[['Preg.Norm', 'Pedigree.Norm',
'Glucose.Norm']], train_data['HasDiabetes'])


predictions_1_with_glucose = knn_1_with_glucose.predict(test_data[['Preg.Norm',
'Pedigree.Norm', 'Glucose.Norm']])
error_rate_1_with_glucose = 1 - accuracy_score(test_data['HasDiabetes'],
predictions_1_with_glucose)

print(error_rate_1_with_glucose)

# ANSWER: No, adding more features does not always reduce the error rate. Extra
features can introduce noise and increase the model's complexity, potentially
leading to overfitting. It's essential to select features that are truly
informative.
```

```
0.31939163498098855
```

```
# h. Repeat part (g) but set k=5. What is the error rate? [code, error rate]

knn_5 = KNeighborsClassifier(n_neighbors=5)
knn_5.fit(train_data[['Preg.Norm', 'Pedigree.Norm', 'Glucose.Norm']],
train_data['HasDiabetes'])


predictions_5 = knn_5.predict(test_data[['Preg.Norm', 'Pedigree.Norm',
'Glucose.Norm']])
error_rate_5 = 1 - accuracy_score(test_data['HasDiabetes'], predictions_5)

print(error_rate_5)
```

```
0.2395437262357415
```

```
# i. Repeat part (h) but set k=11. What is the error rate?  [code, error rate]

knn_11 = KNeighborsClassifier(n_neighbors=11)
knn_11.fit(train_data[['Preg.Norm', 'Pedigree.Norm', 'Glucose.Norm']],
train_data['HasDiabetes'])


predictions_11 = knn_11.predict(test_data[['Preg.Norm', 'Pedigree.Norm',
'Glucose.Norm']])
error_rate_11 = 1 - accuracy_score(test_data['HasDiabetes'], predictions_11)

print(error_rate_11)
```

```
0.22053231939163498
```

```
# j. Considering your observations from (g)-(i), which is the best value for k?
[answer]
# ANSWER: The best value for k is the one with the lowest error rate, observed
from the results of parts (g)-(i).
```