Question 1:

Tabularized are the results obtained from the kNN classifiers using the Euclidean distance metric:

k	Training error	Testing Error	Fitting	Variance	Bias
1	0.0000	0.2200	Overfitting	High	Low
3	0.1435	0.2050	Overfitting	High	Low
5	0.1561	0.1700	Fitting	Balanced	Balanced
10	0.1519	0.1700	Fitting	Balanced	Balanced
20	0.1688	0.1700	Fitting	Balanced	Balanced
30	0.1688	0.1600	Fitting	Optimal	Optimal
50	0.1582	0.1900	Underfitting	Low	High
100	0.1962	0.2000	Underfitting	Low	High
150	0.1920	0.1900	Underfitting	Low	High
200	0.2215	0.2050	Underfitting	Low	High

Understandably, the kNN models using **k** = 1 and **k** = 3 generated extremely low training errors, but considerably high test errors. We can confidently flag these models as **Overfitting**, having high variance and low bias. That is, introducing new data points or changing the training data will probably lead to vast changes in the predictions made by these models.

On the other hand, the models with values of **k** = 50, 100, 150 and 200 generated both high training errors and high test errors. These models can be labeled as **Underfitting**, having high bias (towards the particular training data) and low variance to changes in the training dataset.

However, the ideal values of k = 5, 10, 20, 30 led to models that generated consistently low training errors but, more importantly, low test errors as well; with k =30 yielding the **lowest** test error rate.

Question 2:

In Question 1, using **k** = 30 with the Euclidean distance metric yielded the best performance, with the training error rate being **0.1688** and the test error rate being **0.1600**.

Simply replacing the metric with the Manhattan distance, and keeping **k** = 30, we observed a decrease in performance compared to the Euclidean distance metric; the training error rate dropped to **0.1646**, but the test error rate rose to **0.1650**.

Question 3:

Although the Euclidean distance metric generates the **lowest** test error rate of the two metrics, on average (**0.188** v. **0.165**), across all values of **k**, the Manhattan metric yields the **lower** test error rate.

As such, we utilized the Manhattan metric in generating the plots of the capacity ***log***(1/**k**) against the training and test error rates.

Rightmost are the overfitting models with **k** = 1, 3 as discussed in Question 1. Due to the low bias/high variance that models in this region possess, they are also more likely to fit a wider variety of functions; that is, they possess **high capacity**.

Leftmost are the opposite, underfitting models. They possess **low capacity** due to their high bias and low variance; they are unable to generalize or fit well to unseen or new training data.

The models in the middle can be said to be of **optimal capacity**, balanced variance, and balanced bias. While the training error is not necessarily the **lowest**, they **sufficiently** generalize to unseen instances/test data.

Question 4:

Due to the limited training data available, we **chose to not** split the training data into subsets of an actual training set (, validation set), and vaulted test set. The other reason for this decision is because reducing the actual training set to 80% of its original size causes vast changes in the results of the model improvement that we perform, making the results less reliable. At times, the metric changes, at other times, the optimum number of neighbours fluctuates, depending on which subsample of the training data is used for training, and which is vaulted.

Since we already possess an entire 'blind' test set (test.sDAT & test.NC), there's no need to reduce the vital training data we have, especially while using the kNN model whose **parameters** are directly tied to the training data.

Due to the continuous (non-categorical) natures of the features, **x1** and **x2**, methods such as frequency, label, one-hot (OHE) and ordinal encoding are not plausible.

One possible source of performance improvement would be via scaling techniques, since **kNN** is a distance-based learning method. As such, we plotted the distributions of the two attributes, **x1** and **x2**, to get a better overview of the shape of the values, and to assist in picking appropriate scaling techniques.

Plotting the distributions revealed two things:

- the presence of outliers;
- both attributes are normally distributed, having narrow Gaussians

The outliers were removed from each attribute using their respective interquartile ranges. To help determine which scaling technique to settle with, a kNN model, based on the optimum **hyperparameters** (**k**=30, **metric**=Euclidean) from Q1-Q3 was created, and run on different instances of the training set scaled via the **MinMaxScaler**, the **RobustScaler** and the **FunctionTransformer(log1p)**. Using repeated cross-validation on the scaled data, the average model scores were computed and the **FunctionTransformer(log1p)** was selected as the ideal scaler (**0.1688 [Question 1]** v. **0.1065 [unscaled]**).

```
mean unscaled error rate:  0.1073
mean MinMaxScaler() error rate:  0.1069
mean RobustScaler() error rate:  0.1075
mean FunctionTransformer(log1p) error rate:  0.1065
```

Fig. 1 – sample instance of mean training error rates of different scalers

Subsequently, a grid search was performed to further determine the optimum **hyperparameters** to pass into the **kNN** model. The image below shows the ideal hyperparams.

```
Best Params: {'metric': 'euclidean', 'n_neighbors': 13, 'weights': 'uniform'}
```

Fig. 2 – results of hyperparameter grid search

Running the final modified model on the original test set yielded a slight, but noticeable, increase in the test error rate (**0.1600 [Question 1]** v. **0.1650**).

```
using revised model on test.sDAT.csv and test.sNC.csv as "blind" test set...
[1 0 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1
 1 1 1 1 1 1 0 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 1 1 1
 1 1 1 1 1 1 1 0 1 1 1 1 1 0 1 1 1 1 1 1 1 1 0 1 0 1 0 1 0 1 0 0 0 0 0 0 1
 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 1 0 1 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 1
 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 1 0 0 1 0 0 1 0 1 0 1 0 0 0
 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0]
"Blind" test error:  0.165
```

Fig. 3 – running revised model on the original test set