Question 1.
I decided to use the kknn function to find a good classifier because I found the library to be more understandable and easier to use:

(a) Using cross-validation for the k-nearest-neighbors model…

Please see attached question-1a.R for code

Cross-validation is used to better estimate model quality and allow us to choose a model more effectively. First, I split the credit card dataset into a training (80% of the data) and a testing (remaining 20%) using the random method.

In this example, I built three models using 10-fold cross validation. The three models have number of neighbors (k) set to 4, 12, and 15. Each k value is passed to a train.model function I created that trains the model using cv.kknn and returns its predictions on the training/validation datasets.

The results of each of these models is passed to eval.model - a function I created to calculate predictive accuracy. The three k values produced the following accuracies:

k = 4 was 79.92352% accurate
k = 12 was 85.277725% accurate
k = 15 was 86.61568% accurate

Based on these CV results, k = 15, was chosen as the highest performing model. The model was re-trained using the entire training dataset and evaluated using the test dataset. From this, the k = 15 model produced an accuracy of 83.96947%.

(b) Splitting the data into training, validation, and test data sets…

Please see attached question-1b.R for code

I split the credit card dataset into a training (70% of the data), validation (15% of the data), and test (remaining 15%) using the random method.

In this example, I built three models using various number of neighbors. The three models have number of neighbors (k) set to 4, 12, and 15. Each k value is passed to a train.model function I created that trains the model using kknn() and returns its predictions on the validation dataset.

The results of each of these models is passed to eval.model - a function I created to calculate predictive accuracy. The three k values produced the following accuracies:

k = 4 was 79.59184% accurate
k = 12 was 85.71429% accurate
k = 15 was 86.73469% accurate

Based on these CV results, k = 15, was chosen as the highest performing model. The model was re-trained using the training dataset and evaluated against the test dataset. From this, the k = 15 model produced an accuracy of 81.63265%.

Question 2.
My daughter has a wide variety of books. Every night we pick out three to read before bed. It would be useful to apply a clustering model to group these books for storing and picking from for our nightly readings. Five attributes that could be used for grouping would be: word count, reading level, pages, publisher, and subject matter.

Question 3.
Please see attached question-3.R for code

There are four parameters that can be adjusted when applying the kmeans algorithm:

1. **centers → the number of clusters**
    a. Since we're using the iris dataset we can confidently set this at three; we know there are three different species.
2. **iter.max → the maximum number of iterations allowed**
    a. We should set this high enough to allow the algorithm to converge on an optimal solution, but low enough that it doesn't take an excess amount of time to run. Given the small dataset size of 150 observations, this parameter won't have an impact and will be fixed at 100,000.
3. **nstart → number of random sets to be chosen**
    a. This will be one of the parameters I'll adjust to find an optimal application of kmeans. I looped through a range of 1:100 for this parameter.
4. **Algorithm → the specific kmeans algorithm to use**
    a. This is the other parameter I adjusted to find an optimal application of kmeans. The possible values are Hartigan-Wong, Lloyd, and MacQueen (Forgy isn't used since it's an alternative name for Lloy).

I created three functions in my R script to iterate on the kmeans algorithm. The iris.cluster function takes an nstart and algorithm value and applies kmeans to the iris dataset; returning an accuracy value. Rename.cluster replaces the cluster value (1,2,3) with the corresponding Species value that was most common for the cluster value. Algo.cluster is passed an algorithm value and loops through iris.cluster; passing the algorithm value and nstart values from 1:100.

Results:

kmeans applied to Iris dataset

nstart	Hartigan	Lloyd	MacQueen
1	89.3%	89.3%	89.3%
2	89.3%	88.7%	89.3%
3	89.3%	88.7%	88.7%
4	89.3%	89.3%	89.3%
5	89.3%	89.3%	89.3%
6	89.3%	89.3%	89.3%
7	89.3%	88.7%	89.3%
8	89.3%	89.3%	89.3%
9	89.3%	89.3%	89.3%
10	89.3%	89.3%	89.3%
...	89.3%	89.3%	89.3%
100	89.3%	89.3%	89.3%

There was very little accuracy variation between the algorithms and nstarts. For this reason, I would use the Hartigan algorithm and default nstart value; since it maintained a consistent 89.3% accuracy regardless of nstart value.