Exercise 1:

We want to predict the scored home runs (variable `runs_scored`) from a number of characteristics of US baseball games. You can find the corresponding data on Moodle.

The corresponding task is created (adapt your path if necessary) as follows:

```
library(data.table)
data_baseball <- fread("baseball.csv")
data_baseball$team <- as.factor(data_baseball$team)
data_baseball$league <- as.factor(data_baseball$league)
```

We want to use a k-NN algorithm. However, we are not sure what number of neighbors k yields the best result. This is why we want to use tuning to determine the best value for k. We assume that it might be somewhere in the range from 1 to 100.

Furthermore, we want to use **random search** to search our defined search space. In addition, we only want to carry out the tuning a maximum of 80 times.

1) Think about what random search actually means with regard to the search space and a suitable stopping criterion.

We still need to define which resampling method is supposed to be used during tuning. We want to use 5-fold CV and compute the MSE in each iteration to estimate the generalization error of the respective candidate.

2) Implement the resampling procedure for n-fold CV as an auxiliary function that returns the train and test indices for each fold. The user should be able to feed in the data indices (`idx`), the number of `folds`, and a random `seed`.

```
resample_cv <- function(idx, folds, seed = 123) {
    # shuffle indices
    ...
    # prepare objects needed to store indices
    ...
    # CV iterations
    for (i in seq(folds)) {
        ...
    }
    # return
    ...
}
```

In order to make this easier, let's break the code down to smaller building blocks:

- First, we shuffle the input indices (`idx`) to avoid any unintended sorting:

```
# shuffle indices
...
```

- We will take this randomly sorted list of indices and cut it into non-overlapping slices representing the test sets – the rest makes up the training set for the respective fold. For this, we will move along the indices from a start index for one interval length (= number of test indices in the fold). Take the largest integer number possible as `interval_length` (for the sake of simplicity, you can ignore a potential rest

resulting from division here). Moreover, we create an empty list object of length `folds` in which to store the indices of each fold.

```
# prepare objects needed to store indices
start_index <- 1
interval_length <- ...
idx_list <- ...
```

– Now we can loop through the folds and find the train and test indices for each iteration. Your code should find the test indices for this fold, set the remainder as training indices, and store everything in `idx_list`. At the end of each iteration, remember to move the `start_index`.

```
# CV iterations
for (i in seq(folds)) {
    # Define test and train indices
    test_idx <- ...
    train_idx <- ...
    idx_list[[i]]$test <- test_idx
    idx_list[[i]]$train <- train_idx
    # Move start index
    start_index <- ...
}
```

– In the last step, return the list object of indices:

```
# return
...
```

3) Perform the random search with the above specifications, storing the results for each candidate configuration. Use the following set-up:

```
# Define task and learner
task <- TaskRegr$new("bb", backend = data_baseball, target = "runs_scored")
lrn_knn = lrn("regr.kknn")

# Define tuning settings
search_space <- ...
max_evals <- ...
folds <- ...
resampling_idx <- resample_cv(task$row_ids, folds)

# Get configuration candidates
set.seed(123)
k_candidates <- ...
```

And perform the tuning procedure as follows:

```
library(mlr3)
library(mlr3learners)
# Define archive to store results
tuning_archive <- data.table(
    "iteration" = seq_along(k_candidates), "k" = k_candidates, "ge" = 0
)
```

```
# Perform tuning
for (i in tuning_archive$iteration) {
    # Set current configuration
    lrn_knn$param_set$values <- list(k = tuning_archive[iteration == i, k])
    # Estimate GE via 5-CV
```

```
    ge_est <- 0
    for (j in 1:folds) {
        # Train on training data in j-th fold
        ...
        # Predict on test data in j-th fold
        predictions <- ...
        # Accumulate estimated GE
        ge_est <- ge_est + (1 / folds) * predictions$score()
    }
    tuning_archive[iteration == i]$ge <- ge_est

}
```

4) In the last step, plot the estimated generalization error for different values of k:

```
library(ggplot2)
ggplot(tuning_archive, aes(x = k, y = ge)) +
    geom_line() +
    geom_point(
        tuning_archive[which.min(tuning_archive$ge)],
        mapping = aes(x = k, y = ge),
        col = "blue",
        size = 5
)
```