Solution 1:

a) Since the polynomial learner clearly achieves a better fit for the training data and some observations lie rather far from the regression line, which is strongly penalized by $L2$ loss, it will have lower empirical risk than the linear learner.

b) First and foremost, evaluation on the training data is almost never a good idea. Under certain conditions the training error does tell us something about generalization ability, but for flexible learners and/or small training data it is deceptive due to optimistic bias. In this particular situation, we have few training observations and quite some points that look a little extreme. A low training error might be achieved by a learner that fits every quirk in the training data but generalizes poorly to unseen points with only slightly different distribution. Evaluation on separate test data is therefore non-negotiable.

c) We fit the polynomial and linear learner and then compute the squared and absolute differences between their respective predictions and the true target values:

```r
# define train data including outlier
set.seed(123)
x_train <- seq(10, 15, length.out = 50)
y_train <- 10 + 3 * sin(0.15 * pi * x_train) + rnorm(length(x_train), sd = 0.5)
data_train <- data.frame(x = x_train, y = y_train)

# define test data
set.seed(321)
x_test <- seq(10, 15, length.out = 10)
y_test <- 10 + 3 * sin(0.15 * pi * x_test) + rnorm(length(x_test), sd = 0.5)
data_test <- data.frame(x = x_test, y = y_test)

# train learners
polynomial_learner <- lm(y ~ poly(x, 21), data_train)
linear_learner <- lm(y ~ x, data_train)

# predict with both learners
y_polynomial <- predict(polynomial_learner, data_test)
y_lm <- predict(linear_learner, data_test)

# compute errors
abs_differences <- lapply(
  list(y_polynomial, y_lm),
  function(i) abs(data_test$y - i))
errors_mse <- sapply(abs_differences, function(i) mean(i^2))
errors_mae <- sapply(abs_differences, mean)

print(c(errors_mse, errors_mae))

## [1] 0.2731046 0.3031514 0.4140316 0.3827525
```

The picture is inconclusive: based on MSE, we should prefer the complex polynomial model, while MAE tells us to pick the linear one. It is important to understand that the choices of inner and outer loss functions encode our requirements and may have substantial impact on the result. Here, following the MAE assessment would signify preference for a robust model.

However, we must keep in mind that our performance evaluation is based on a single holdout split, which is not advisable in general and particularly deceptive with so little data.

Solution 2:

a) Get the data, define a task and corresponding train-test split, and predict with trained model:

```
# get data
library(mlbench)
data(BostonHousing)
data_pollution <- data.frame(dis = BostonHousing$dis, nox = BostonHousing$nox)
data_pollution <- data_pollution[order(data_pollution$dis), ]
head(data_pollution)

##        dis   nox
## 373 1.1296 0.668
## 375 1.1370 0.668
## 372 1.1691 0.631
## 374 1.1742 0.668
## 407 1.1781 0.659
## 371 1.2024 0.631

# define task and train-test split
library(mlr3)
task <- mlr3::TaskRegr$new("pollution", backend = data_pollution, target = "nox")
train_set = 1:10
test_set = setdiff(seq_len(task$nrow), train_set)

# train linear learner
library(mlr3learners)
learner <- mlr3::lrn("regr.lm")
learner$train(task, row_ids = train_set)

# predict on test data
predictions <- learner$predict(task, row_ids = test_set)
predictions$score(mlr3::msr("regr.mse"))

## regr.mse
## 1.524512
```

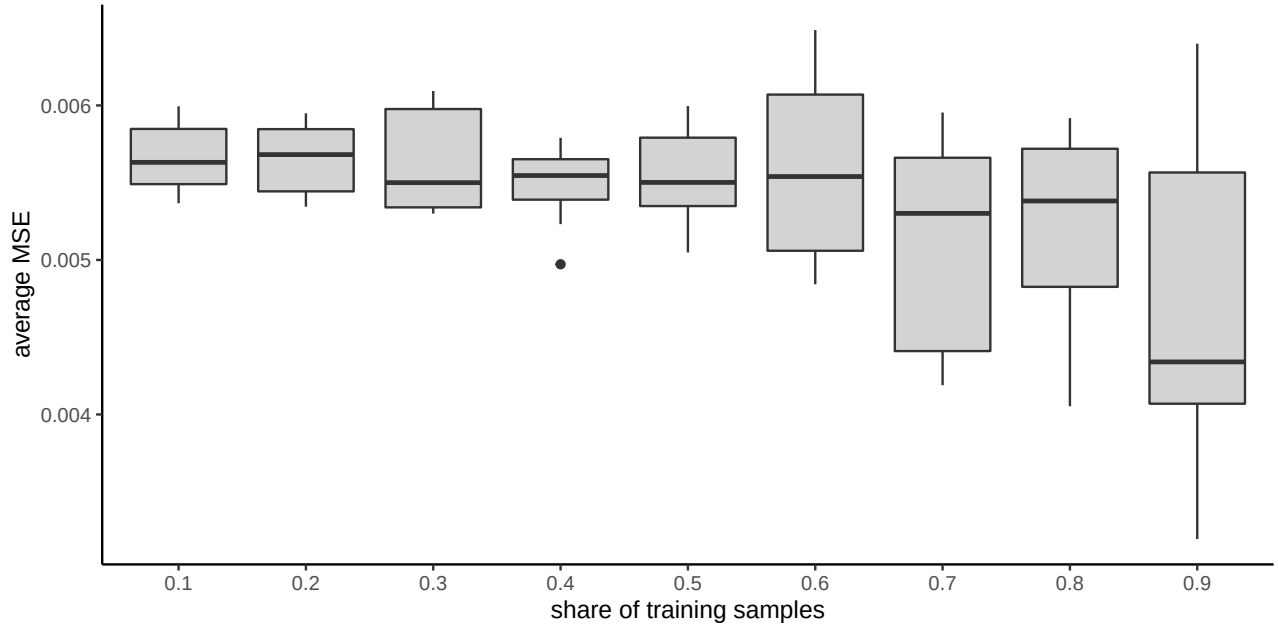
b) We have chosen the first ten observations from a data set that is ordered by feature value, which is not a good idea because this is not a random sample and covers only a particular area of the feature space. Consequently, we obtain a pretty high test MSE (relatively speaking – we will see in the next exercise which error values we can usually expect for this task). Looking at the data, this gives us a steeply declining regression line that does not reflect the overall data situation. Also, a training set of ten points is pretty small and will likely lead to poor generalization.

c) We repeat the above procedure for different train-test splits like so:

```
# define train-test splits
repetitions <- 1:10
split_ratios <- seq(0.1, 0.9, by = 0.1)

# create resampling objects with holdout strategy, using lapply for efficient computation
split_strategies <- lapply(split_ratios, function(i) mlr3::rsmp("holdout", ratio = i))

# train linear learners and predict in one step (remember to set a seed)
set.seed(123)
results <- list()
for (i in repetitions) {
```

```
  results[[i]] <- lapply(split_strategies, function(i) mlr3::resample(task, learner, i))
}
```

```
# compute errors in double loop over repetitions and split ratios
errors <- lapply(
  repetitions,
  function(i) sapply(results[[i]], function(j) j$score()$regr.mse))

# assemble everything in data.frame and convert to long format for plotting
errors_df <- as.data.frame(do.call(cbind, errors))
errors_df$split_ratios <- split_ratios
errors_df_long <- reshape2::melt(errors_df, id.vars = "split_ratios")
names(errors_df_long)[2:3] <- c("repetition", "mse")

# plot errors vs split ratio
library(ggplot2)
ggplot(errors_df_long, aes(x = as.factor(split_ratios), y = mse)) +
  geom_boxplot(fill = "lightgray") +
  theme_classic() +
  labs(x = "share of training samples", y = "average MSE")
```



d) From the experiment in c) we can derive two conclusions:

1) A smaller training set tends to produce higher estimated generalization errors.
2) A larger training set, at the expense of test set size, will cause high variance in the individual generalization error estimates.

e) In order to solve this, we rely mostly on the linearity of the expectation and what is known as "Verschiebungssatz der Varianz" in German (i.e., $\mathsf{Var}(z) = \mathbb{E}(z^2) - \mathbb{E}^2(z)$ for some random variable z). Then:

$$\begin{aligned}
\mathbb{E}_{xy}((\hat{y} - y)^2 \mid \mathbf{x}) &= \mathbb{E}_{xy}(\hat{y}^2 - 2\hat{y}y + y^2 \mid \mathbf{x}) \\
&= \hat{y}^2 - 2 \cdot \hat{y} \cdot \mathbb{E}_{xy}(y \mid \mathbf{x}) + \mathbb{E}_{xy}(y^2 \mid \mathbf{x}) \\
&= \hat{y}^2 - 2 \cdot \hat{y} \cdot \mathbb{E}_{xy}(y \mid \mathbf{x}) + \mathbb{E}^2_{\mathbb{P}_{xy}}(y \mid \mathbf{x}) + \mathsf{Var}_{xy}(y \mid \mathbf{x}) \\
&= (\hat{y} - \mathbb{E}_{xy}(y \mid \mathbf{x}))^2 + \mathsf{Var}_{xy}(y \mid \mathbf{x}).
\end{aligned}$$

f) Now we account for the fact that $\hat{y}$ is a random variable if we compute repeated estimates of the generalization error based on random sampling processes. For this, we start from the expression we have derived in e), only now with the additional expectation around the first component:

$$\begin{aligned}
\mathbb{E}_{xy}((\mathbb{E}_{\hat{y}}((\hat{y} - y)^2) \mid \mathbf{x}) &= \mathbb{E}_{\hat{y}}\left((\hat{y} - \mathbb{E}_{xy}(y \mid \mathbf{x}))^2\right) + \mathsf{Var}_{xy}(y \mid \mathbf{x}) \\
&= \mathbb{E}_{\hat{y}}\left(\hat{y}^2 - 2 \cdot \hat{y} \cdot \mathbb{E}_{xy}(y \mid \mathbf{x}) + \mathbb{E}^2_{xy}(y \mid \mathbf{x})\right) + \mathsf{Var}_{xy}(y \mid \mathbf{x}) \\
&= \mathbb{E}_{\hat{y}}(\hat{y}^2) - 2 \cdot \mathbb{E}_{\hat{y}}(\hat{y}) \cdot \mathbb{E}_{xy}(y \mid \mathbf{x}) + \mathbb{E}^2_{xy}(y \mid \mathbf{x}) + \mathsf{Var}_{xy}(y \mid \mathbf{x}) \\
&= \mathbb{E}^2_{\hat{y}}(\hat{y}) + \mathsf{Var}_{\hat{y}}(\hat{y}) - 2 \cdot \mathbb{E}_{\hat{y}}(\hat{y}) \cdot \mathbb{E}_{xy}(y \mid \mathbf{x}) + \mathbb{E}^2_{xy}(y \mid \mathbf{x}) + \mathsf{Var}_{xy}(y \mid \mathbf{x}) \\
&= (\mathbb{E}_{\hat{y}}(\hat{y}) - \mathbb{E}_{xy}(y \mid \mathbf{x}))^2 + \mathsf{Var}_{\hat{y}}(\hat{y}) + \mathsf{Var}_{xy}(y \mid \mathbf{x}).
\end{aligned}$$

Okay, now what to make of this? The first thing we need to understand is that $\mathbb{E}_{xy}(y \mid \mathbf{x})$ is the best possible prediction we could make: it is the conditional expectation of our target under the true data-generating process, i.e., the value of y we would expect to observe for given $\mathbf{x}$ if we knew the true feature-target relation. Unfortunately, our prediction $\mathbb{E}_{\hat{y}}(\hat{y})$ will in general be different from $\mathbb{E}_{xy}(y \mid \mathbf{x})$ because we do not know the whole truth. Looking at the last line of our derivation, we see that the first term is exactly the MSE btween these two quantities – i.e., the expected square *bias* we make with our prediction.

The second summand $\mathsf{Var}_{\hat{y}}(\hat{y})$ represents, unsurprisingly, the *variance* part of the decomposition.

Now the last term is the variance of the true conditional distribution, which, crucially, does not depend on our prediction. No matter how accurate our model is, this component will be irreducible. It thus marks a lower bound on the generalization error and is also referred to as *Bayes error*.

Summarizing:

$$\mathbb{E}_{xy}((\mathbb{E}_{\hat{y}}((\hat{y} - y)^2) \mid \mathbf{x}) = \left(\underbrace{\mathbb{E}_{\hat{y}}(\hat{y}) - \mathbb{E}_{xy}(y \mid \mathbf{x})}_{\text{bias}}\right)^2 + \underbrace{\mathsf{Var}_{\hat{y}}(\hat{y})}_{\text{variance}} + \underbrace{\mathsf{Var}_{xy}(y \mid \mathbf{x})}_{\text{irreducible error}}.$$

We see again that $\mathbb{E}_{\hat{y}}(\hat{y})$ will, on average, be a better estimator for $\mathbb{E}_{xy}(y \mid \mathbf{x})$ if we have more data for training (provided our learner class is flexible enough), while $\mathsf{Var}_{\hat{y}}(\hat{y})$ tends to be large if we leave little data to test on.