Lab 1

Benjamin Agardh

September 17, 2025

1 Introduction

This lab considers data for bike rentals in the city of London. It explores the correlation between the number of rentals and the humidity through linear regression. This includes transforming the response variable using a Box-Cox-transformation, comparing linear models on the form $\beta_0 + \beta_1 x^\gamma$ with the $pMSE$ index, creating approximate prediction intervals using a simulation-based approach, and numerically validating the theoretical concept of a confidence interval.

2 Assignment 1(i)

2.1 Problem

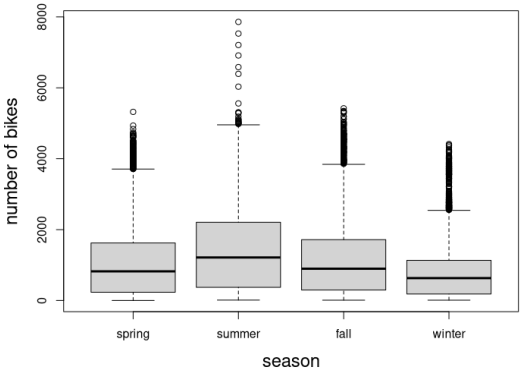
Before we begin drawing any conclusions from it we want to visualize and understand the dataset provided. The first task is to generate boxplots for the number of bikes shared vs. the season and the hour of the day.

2.2 Theory and implementation

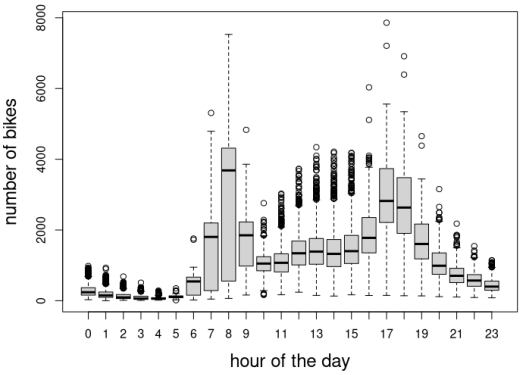
- **The data:** The dataset contains hourly data from January 2015 until January 2017. It includes 12 variables, and a description of the ones relevant to this lab is given here:
  - `timestamp`: day of the year and hour
  - `cnt`: the number of new bike shares in the considered timestamp
  - `hum`: humidity in percentage
  - `season`: meteorological seasons: 0-spring ; 1-summer; 2-fall; 3-winter.
- **Formatting:** The data was formatted and variables redefined to make it easier to work with (see A.1). Most notably, the `season` variable was redefined to contain the labels ("spring", "summer", "fall", "winter"), and a `hrs` variable containing the hour of the day was created from the `timestamp`.
- **Plotting:** To generate boxplots, the R function `boxplot()` was used to plot the `cnt` variable versus `season` and `hrs` (see A.2).

2.3 Results and discussion

- Plots of the number of shared bikes vs. season and hour of the day are shown in Figure 1.
- By studying the boxplots of the bike sharing dataset we conclude that the bike sharing is busiest during summer (see 1a) and in the mornings and the afternoons (ca 7-9 am and 17-18 pm, see 1b). This makes sense as people probably are more likely to go biking when the weather is warm (which it usually is during summer) and when they go to or from work.



(a) The number of shared bikes vs. season.



(b) The number of shared bikes vs. hour of the day.

Figure 1: The number of shared bikes (`cnt`) vs. season (`season`) or the hour of the day (`hrs`), for the entire dataset.

3 Assignment 1(ii)

3.1 Problem

For this task we will study only a subset of the data by excluding data between hours [7am, 9am] and between [17.00, 18.00], and only including data from spring. On this dataset we want to apply linear regression to study how the number of shared bikes correlate to the humidity. To reduce noise we first apply a Box-Cox transformation to the response variable.

3.2 Theory and implementation

- **Filtering data:** To exclude data from "rush hours" and then to only retain data from spring we use the R function `subset(x, subset)`. It takes the data we want to subset as the first argument and a filter as the second. For example, we use

  ```
  norush = subset(bikesharing, !(hrs>=7 & hrs<=9) & !(hrs>=17 & hrs<=18))
  data = subset(norush, season == "spring")
  ```

  to exclude data between hours [7am, 9am] and between [17.00, 18.00], and then to only retain data from spring (see A.3). The dataset denoted `data` will be used throughout the remaining tasks, and will always refer to the data filtered in this way.

- **Linear regression:** To understand how a variable y correlates to another variable x, we can use *linear regression* to fit a linear model

  $$E(y|x) = f(x)$$

  to our dataset. We call the variable that is to be estimated the *response variable* y, and the variable that it depends on the *covariate* x. A common choice for f is to use a polynomial, for example $f(x) = \beta_0 + \beta_1 x$, where β_0 and β_1 are parameters that need to be estimated when fitting the model. For this, we often use ordinary least squares (OLS).

  When we have fitted the model to our data, we can use it to predict values for our response variable given a value for x. We often denote the predictions $\hat{y} := E(y|x)$. We note that an observed value y_i is given by $y_i = \hat{y}_i + e_i$, where e is the *residual* that is present due to for example measurement errors or due to assuming an incorrect model.

In R, we use the `lm(y~x)` function to fit a simple linear model $\beta_0 + \beta_1 x$ to the covariate `x` and the response variable `y`.

- **Box-Cox transformation:** To obtain a model using linear regression that gives the best possible predictions, we want the *residuals* $e_i = y_i - \hat{y}_i$ to be approximately symmetric around 0. The so-called *Box-Cox transformation* introduces a way to transform the response variable y so that the residuals might become more symmetrically distributed around 0, by calculating

$$y(\lambda) = \begin{cases} \frac{y^\lambda - 1}{\lambda} & \text{if } \lambda \neq 0 \\ \log y & \text{if } \lambda = 0 \end{cases}$$

and using it as the new response variable when applying linear regression. $\lambda > 0$ is a parameter of the transformation that has to be estimated, usually through a maximum likelihood method.

- **Implementation:** For this task, a Box-Cox transformation is performed using

```
m = lm(data$cnt+1 ~ data$hum)
library(MASS) # Include required library
bc = boxcox(m)
```

The best approximation for the parameter λ is then extracted using `bc$x[which(bc$y == max(bc$y))]`, and a figure of the log-likelihood plot for λ is included. We then apply linear regression to the transformed variable with

```
m = lm((data$cnt^lambda-1)/lambda ~ data$hum)
```

Note that we're adding 1 to the response variable `cnt` when estimating λ. This is to make sure it is > 0, or the `boxcox()` method throws an error.

See A.3 for the entire code.

3.3 Results and discussion

- The plot of the number of shared bikes (`cnt`) versus the humidity (`hum`) for the entire dataset is shown in Figure 2a, and for data from spring and outside of "rush hours" in Figure 2b. We note that both datasets contain a lot of noise, but from just looking at the plots it seems like a trend would be easier to find in the filtered data than in the whole dataset.

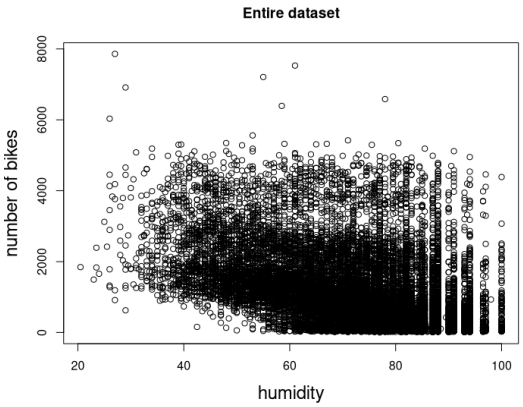
- For the Box-Cox approximation, the optimal prediction for λ is $\hat{\lambda} \approx 0.343$. This means we're using the transformed response

$$y(\hat{\lambda}) = \frac{y^{\hat{\lambda}} - 1}{\hat{\lambda}} \tag{1}$$

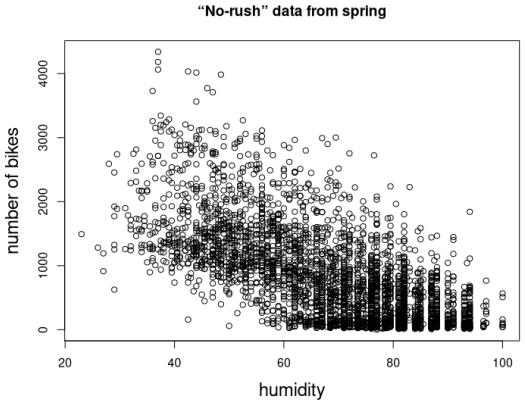
where y is the original untransformed response, and $\hat{\lambda} \approx 0.343$. The log-likelihood plot for λ is presented in Figure 3.

- The transformed response data $y(\hat{\lambda})$ calulated using (1) plotted versus humidity is shown in Figure 4a, and the residuals $e = y(\hat{\lambda}) - \hat{y}(\hat{\lambda})$ are shown in Figure 4b.

  We note that the transformed data visually follows the trend line quite well, and the residuals are somewhat evenly distributed about the horizon line $y = 0$. This gives us a hunch that the model might be quite good, but a proper mathematical evaluation will be done in the coming tasks.



(a) The entire dataset.



(b) Data from spring outside of "rush hours" (7-9am or 17-18).

Figure 2: The number of shared bikes against humidity for two subsets of the data.

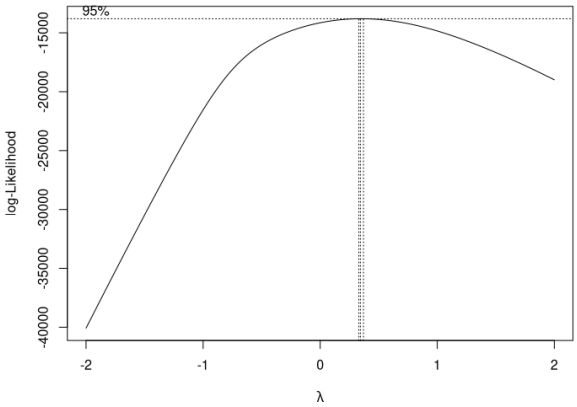
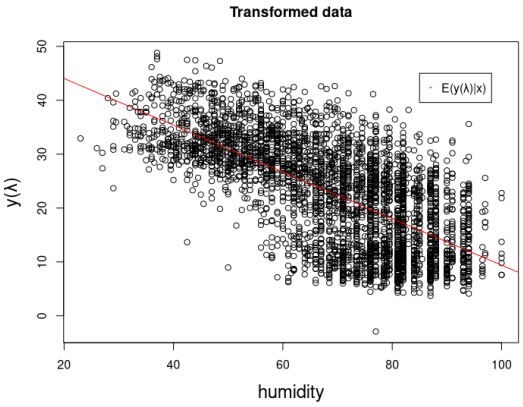
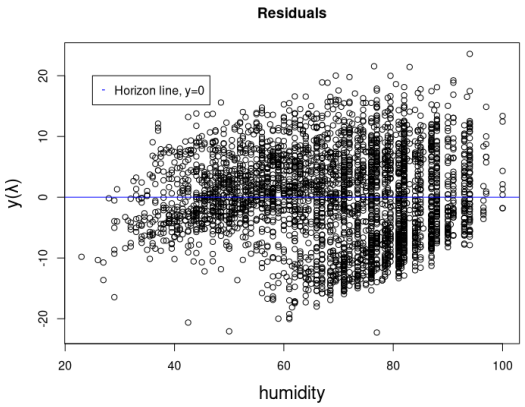


Figure 3: Log-likelihood for λ from the Box-Cox transformation of the response variable `cnt` with the covariate `hum`.



(a) The transformed response variable $y(\hat{\lambda})$ against humidity (`hum`), and the simple linear model fitted to the data.



(b) Residuals from the model fitted to the transformed response variable $y(\hat{\lambda})$ vs. the humidity (`hum`).

Figure 4: The data after applying a Box-Cox transformation with parameter $\hat{\lambda} \approx 0.343$ to the response variable `cnt`.

4 Assignment 1(iii)

4.1 Problem

First, we use the model from the previous task to simulate new responses for `hum` = 40. We perform $B = 2000$ such simulations, and untransform the simulated responses by reverting the Box-Cox transformation. We then calculate a 95% prediction interval for the untransformed response y.

4.2 Theory and implementation

- **Residual standard error:** The *residual standard error* of a linear model is given by

$$s := \sqrt{\frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{n-q}} \tag{2}$$

  where q is the number of parameters in our model; in our case 2, and n is the length of the data set that the model was trained on. In R, we obtain s by using `summary(m)$sigma`, where `m` is the linear model obtained from `lm()`.

- **Simulating new data:** To simulate new responses from our model we will use

$$y(\hat{\lambda})_{new} = \hat{\beta}_0 + \hat{\beta}_1 x + \epsilon \tag{3}$$

  where $\hat{\beta}_0$ and $\hat{\beta}_1$ are the estimated parameters of the model, x is the covariate (`hum`) which in this case = 40, and ϵ is a simulated error term introduced to represent measurement errors or other real-world variability. We assume that ϵ is distributed according to $N(0, s^2)$. For the given `hum` value $x = 40$ we simulate $B = 2000$ values for ϵ. In R, we do this using `rnorm(n,0,s)`. We then plug ϵ into (3) to obtain our new data.

- **"Untransformation":** To revert the Box-Cox transformation, we use

$$(1) \Rightarrow \left(\hat{\lambda} \cdot y(\hat{\lambda}) + 1\right)^{1/\hat{\lambda}} = y \tag{4}$$

  since $\hat{\lambda} \neq 0$.

- **Prediction intervals:** We want to construct so called *prediction intervals* for the new responses. A prediction interval with confidence level $1-\alpha$ means that if we simulate new data infinitely many times, the simulated data will be in the given interval $100(1-\alpha)\%$ of the times. While a closed formula for such intervals exists, in this task we use a simulation-based approach. From the simulated data we simply take the $\frac{\alpha}{2}$- and $(1-\frac{\alpha}{2})$-quantiles, which will give us a decent approximation for the prediction interval. For this task, we let $\alpha = 0.05$ to obtain a 95% prediction interval.

4.3 Results and discussion

- The residual standard error for the model from the previous task was $s \approx 7.33$.

- With $B = 2000$ simulated responses we obtained the approximate prediction interval $[475, 4608]$ with confidence level 0.95. This means that if we (hypothetically) obtain an infinite amount of new data, approximately 95% of the new data will be in the interval $[475, 4608]$. The simulated data is visualized in Figure 5.

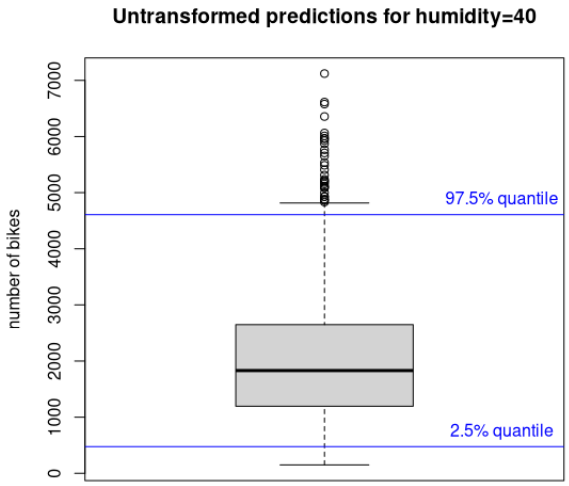


Figure 5: The distribution of our $B = 2000$ new untransformed responses simulated from our linear model for `hum` $= 40$.

5 Assignment 1(iv)

5.1 Problem

In this task we will consider transforming the covariate `hum` as well. Using the Box-Cox-transformed response variable $y(\hat{\lambda})$ from previous tasks, we will fit a linear model $E(y(\lambda)|x) = \beta_0 + \beta_1 x^\gamma$ for γ in the range $[0.5, 2]$ to find the "best" model.

To evaluate the performance of a model we should calculate its $\sqrt{pMSE}$ and plot it against γ to find an estimate for the best value.

5.2 Theory and implementation

- **Splitting the dataset:** We want to split out dataset (`data`) into training- and test-data. By training our model on one dataset and evaluating it on another, we reduce the risk for *overfitting*, meaning the model would be too adjusted to the training data and would not generalize well to new data. In this task, we split the dataset 80-20, meaning 80% of the data is used for training and 20% for evaulation.

  In R, we randomly choose 80% of the indices of the data

  ```
  N = nrow(data) # Size of the entire data set
  n = floor(0.8*N) # Size of the training data
  indices = sample.int(N, n) # Get n values from the sequence 1...N randomly
  ```

  and then split the dataset accordingly

  ```
  y.train = data$cnt[indices]
  x.train = data$hum[indices]
  y.test = data$cnt[-indices]
  x.test = data$hum[-indices]
  ```

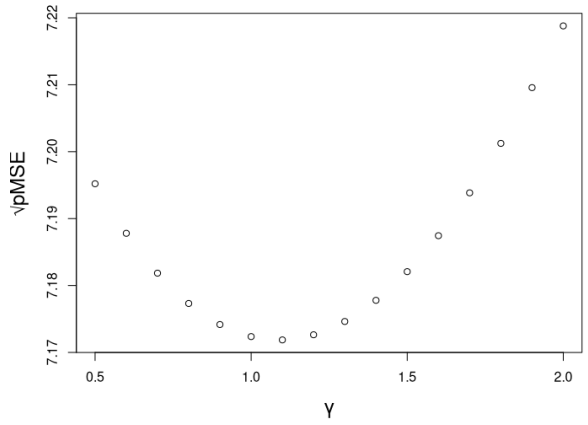


Figure 6: The $\sqrt{pMSE}$ for the untransformed responses from the fitted linear model $\hat{\beta}_0 + \hat{\beta}_1 x^\gamma$ for different values of γ.

- **pMSE:** The *prediction mean-squared-error* for a model is defined as

$$pMSE = \frac{1}{m}\sum_{i=1}^{m}(y_i^{test} - \hat{y}_i)^2 \tag{5}$$

  where y_i^{test} are the actual responses from our test data set (`y.test`), and $\hat{y}_i$ are responses predicted from our test covariates (`x.test`) using a model fitted to the training data.

  The $pMSE$ index is a measure of how well a given model handles previously unseen data, where a lower index is better.

- **Implementation:** To calculate the $\sqrt{pMSE}$ for a linear model $E(y(\lambda)|x) = \beta_0 + \beta_1 x^\gamma$ for a given γ, we first fit a model to our training data. Note that both the response variable and the covariate is transformed.

```
m = lm((y.train^lambda-1)/lambda ~ I(x.train^gamma))
```

  Note that we have to use `^` inside the `I()` method for R to interpret our equation correctly.

  We then use the model to predict responses for our test data `x.test`, using the R `predict()` method. We untransform the responses using (4), and then calculate the $pMSE$ using (5) (see A.5).

- To find the γ value that gives the "best" model, we calculate $\sqrt{pMSE}$ for each $\gamma \in [0.5, 0.6, ..., 1.9, 2.0]$ and plot them to approximately find which γ gives the lowest $\sqrt{pMSE}$.

5.3 Results and discussion

- Figure 6 shows the $\sqrt{pMSE}$ for different values of γ in the range $[0.5, 2.0]$. By looking at the figure, we conclude that the lowest $pMSE$ is obtained when $\gamma \approx 1.1$, with $\sqrt{pMSE} \approx 7.172$.

- We note, however, that the $\sqrt{pMSE}$ varies very little around $\gamma = 1.1$, so all values in the range $[0.9, 1.3]$ should be considered. We should probably use another method such as Occam's razor to decide which model is actually most appropriate, but that is beyond the scope of this assignment.

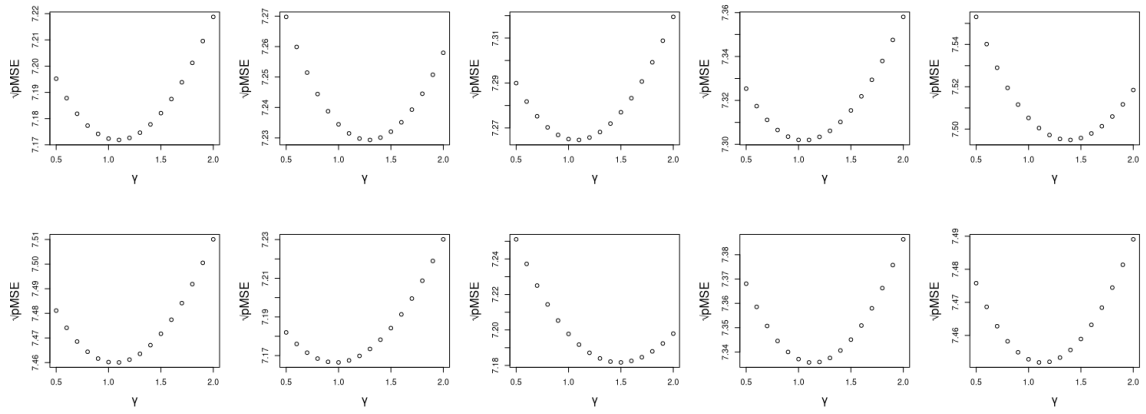


Figure 7: The $\sqrt{pMSE}$ for the untransformed responses from the fitted linear model $\hat{\beta}_0 + \hat{\beta}_1 x^\gamma$ for different values of γ, simulated 10 times.

6 Assignment 1(v)

6.1 Problem

This task consists of repeating what we did in task (iv) independently 10 times to see if we consistently get the same optimal γ,

6.2 Theory and implementation

- **Implementation:** We begin by splitting the data set into training- and test-data. We then repeat what we did in task (iv) with calculating $\sqrt{pMSE}$ for different values of γ and plotting 10 times (see A.6).

6.3 Results and discussion

- Figure 7 shows the $\sqrt{pMSE}$ for different values of γ in the range $[0.5, 2.0]$ for each repetition. By looking at the figure, we note that the optimal γ value varies between 1.0 and 1.5, but the median seems to be around 1.1. Just like in task (iv), however, the variation is quite small, so any $\gamma \in [1.0, 1.5]$ could be a good choice.

7 Assignment 1(vi)

7.1 Problem

Consider the simple linear model with a transformed response from the first tasks: $E(y(\hat{\lambda})|x) = \hat{\beta}_0 + \hat{\beta}_1 x$. Let us assume that the estimated parameters $(\hat{\beta}_0, \hat{\beta}_1)$ are the true parameter values that generated our data, and denote them (β_0^*, β_1^*).

We want to verify that if we obtain new data and construct confidence intervals of level $1-\alpha$ for the parameter estimates obtained from that data, those intervals will contain the **true** parameters β_0^* and β_1^* with a $1-\alpha$ probability.

7.2 Theory and implementation

- **Confidence intervals:** Assume we have datasets $\mathcal{D}_1, \mathcal{D}_2, ...$ generated from

$$y = \beta_0^* + \beta_1^* x + \epsilon. \tag{6}$$

We can obtain parameter estimates $(\hat{\beta}_0^{(i)}, \hat{\beta}_1^{(i)})$ by fitting a linear model $E(y|x) = \beta_0 + \beta_1 x$ to $\mathcal{D}_i$.

If we then construct *confidence intervals* of confidence level $(1-\alpha)$ for each $\hat{\beta}_0^{(i)}$ then $100(1-\alpha)\%$ of these will contain the true parameter β_0^*, and similarily for $\hat{\beta}_1^{(i)}$. This is the definition of confidence intervals, and what we want to validate in this task.

- **In practice:** To validate this, we create the datasets $\mathcal{D}_1, \mathcal{D}_2, ..., \mathcal{D}_B$ by simulating $\epsilon \sim N(0, s^2)$ for each x in our dataset and plugging them into (6), and then repeating B times.

  Then, for each dataset $\mathcal{D}_i$ we fit a simple linear model to the data. We construct confidence intervals of level $(1-\alpha)$ for the estimated parameters $(\hat{\beta}_0^{(i)}, \hat{\beta}_1^{(i)})$ and check if they contain the true parameters (β_0^*, β_1^*).

  Note that we construct confidence intervals for the **transformed** response $y(\hat{\lambda})$, but that doesn't matter as we're only interested in what **fraction** of the intervals contain the true parameters, not the intervals themselves.

- **Implementation:** In R, we use `rnorm(length(data$hum),0,s)` to simulate values for ϵ. We plug these into (6) and fit a linear model. Confidence intervals are constructed for $\alpha = 0.2$ using `confint(m, level = 1-alpha)`, where `m` is our model. Lastly, we check if these intervals contain the true parameters. This procedure is repeated $B = 2000$ times, and the fraction of the intervals that do is calculated (see A.7).

7.3 Results and discussion

- The proportion of confidence intervals for $\hat{\beta}_0^{(i)}$ that contained the true parameter β_0^* was 0.8105, and for $\hat{\beta}_1^{(i)}$ it was 0.8075. These are very close to the expected proportion $1-\alpha = 0.8$, so the theory behind confidence intervals seems to hold.

Statement on AI usage

Large language models were used to fact-check certain information and to help rephrase a few sentences, in cases where my English ability was not sufficient. It was **not** used to produce any content of any kind, and was always checked against the lecture notes.

A Appendix - code

A.1 Data preparation

```
## Prepare data
# Load dataset
setwd("~/Chalmers/tms150/lab1")
bikesharing = read.csv("bikesharing.csv", header=TRUE)
bikesharing = data.frame(bikesharing)

# Format the 'timestamp' variable as a date
library(chron) # load chron
bikesharing$timestamp = as.chron(bikesharing$timestamp) # convert to date
bikesharing$hrs = hours(bikesharing$timestamp) # create a "hours" variable
bikesharing$mnths = months(bikesharing$timestamp) # create a "months"
    variable

# Redefine categorical variables
bikesharing$weather_code = factor(bikesharing$weather_code,levels=c
    (1,2,3,4,7,10,26,94), labels=c("clear","semiclear","brkclouds","cloudy
    ","lghtrain","thunderstorm","snow","freezing"))
bikesharing$is_holiday = factor(bikesharing$is_holiday, levels=c(0,1),
    labels=c(FALSE,TRUE))
bikesharing$is_weekend = factor(bikesharing$is_weekend, levels=c(0,1),
    labels=c(FALSE,TRUE))
bikesharing$season = factor(bikesharing$season, levels=c(0,1,2,3), labels=c
    ("spring","summer","fall","winter"))
```

A.2 Assignment 1(i)

```
# Plot count vs. season
boxplot(bikesharing$cnt~bikesharing$season, xlab="season", ylab="number of
    bikes", cex.lab=1.5)

# Plot count vs. hour of the day
boxplot(bikesharing$cnt~bikesharing$hrs, xlab="hour of the day", ylab="
    number of bikes", cex.lab=1.5)
```

A.3 Assignment 1(ii)

```
# Extract the relevant data
norush = subset(bikesharing, !(hrs>=7 & hrs<=9) & !(hrs>=17 & hrs<=18)) #
    Exclude data from "rush hours"
data = subset(norush, season == "spring") # Only data from spring and
    excluding "rush hours"

# Plot
plot(bikesharing$hum, bikesharing$cnt, xlab="humidity", ylab="number of
    bikes", cex.lab=1.5, main="Entire dataset") # Plot entire dataset
plot(data$hum, data$cnt, xlab="humidity", ylab="number of bikes", cex.lab
    =1.5, main="'No-rush' data from spring") # Plot the filtered data

# Apply BoxCox transformation
m = lm(data$cnt+1 ~ data$hum) # We add 1 here to since the response
    variable must be positive
library(MASS) # Include required library
```

```
bc = boxcox(m) # Perform transformation

# Find the best lambda value
id_max <- which(bc$y==max(bc$y)) # The index of the best lambda
lambda <- bc$x[id_max] # Optimal lambda value

# Transform the response variable
cnt.trans = (data$cnt^lambda - 1)/lambda

# Apply linear regression
m = lm(cnt.trans ~ data$hum) # Fit the model to the transformed response

# Plot the transformed response
plot(data$hum, cnt.trans, xlab="humidity", ylab="y(lambda)", main="
    Transformed data", cex.lab=1.5)
abline(m, col="red") # Fitted line
legend(85, 45, "E(y|x)", col=c("red"), pch="-")

# Plot residuals
plot(data$hum, m$residuals, xlab="humidity", ylab="y(lambda)", main="
    Residuals", cex.lab=1.5)
abline(h=0, col="blue") # horizon, y=0
legend(25, 20, "Horizon line, y=0", col=c("blue"), pch="-")
```

A.4 Assignment 1(iii)

```
set.seed(321) # For reproducibility

B = 2000 # Number of simulated data points
hum = 40 # The humidity we want to simulate responses for

# Extract coefficients
b0.hat = m$coefficients[1]
b1.hat = m$coefficients[2]
s = summary(m)$sigma

# Simulate responses
eps = rnorm(B,0,s)
y.trans = b0.hat + b1.hat * hum + eps # Transformed predictions
y = (y.trans*lambda + 1)^(1/lambda) # Untransformed predictions

# Calculate prediction interval
alpha = 0.05
q.low = quantile(y, alpha/2) # Lower quantile, 2.5%
q.high = quantile(y, 1-alpha/2) # Upper quantile, 97.5%

# Plot the predictions along with the quantiles
boxplot(y, ylab="number of bikes", main="Untransformed predictions for
    humidity=40")
abline(h=q.high, col="blue") # Upper quantile
text(1.4, q.high, "97.5% quantile", pos=3, col="blue")
abline(h=q.low, col="blue") # Lower quantile
text(1.4, q.low, "2.5% quantile", pos=3, col="blue")
```

A.5 Assignment 1(iv)

```
set.seed(321) # For reproducibility

# Split the data into training- and test-sets
N = nrow(data) # Size of the entire data set
n = floor(0.8*N) # Size of the training data
indices = sample.int(N, n) # Get n values from the sequence 1...N randomly
# Split the data set
y.train = data$cnt[indices]
x.train = data$hum[indices]
y.test = data$cnt[-indices]
x.test = data$hum[-indices]

gammas = seq(0.5, 2.0, 0.1) # We try different gamma values in [0.5, 2]
pMSE.sqrt = array(dim=length(gammas)) # will be filled, we just want the
    same length

# Function for calculating the pMSE, using x^gamma model
pMSE = function(gamma) {
  # Fit a x^gamma model to the BoxCox-transformed response
  z = x.train
  m = lm((y.train^lambda-1)/lambda ~ I(z^gammas[i]))
  # Predict y and calculate pMSE
  y.hat = predict(m, newdata=data.frame(z = x.test)) # Transformed
      predictions
  y.test.trans = (y.test^lambda-1)/lambda # Transformed test data
  return (sum((y.test.trans - y.hat)^2) /(N-n))
}

# For each value of gamma, we calculate the sqrt(pMSE) and save it
for (i in 1:length(gammas)) {
  pMSE.sqrt[i] = sqrt(pMSE(gammas[i]))
}

# Plot the pMSE values
plot(gammas, pMSE.sqrt, xlab="gamma", ylab="sqrt(pMSE)", cex.lab=1.5)
```

A.6 Assignment 1(v)

```
set.seed(321) # For reproducibility

K = 10 # Repeat the experiment 10 times
gammas = seq(0.5, 2.0, 0.1) # We try different gamma values in [0.5, 2]
pMSE.sqrt = array(dim=c(K, length(gammas))) # will be filled, we just want
    the same length

# For each value of gamma, we calculate the sqrt(pMSE) and save it for
    later
for (k in 1:K) {
  indices = sample.int(N, n) # Get n values from the sequence 1...N
      randomly
  # Split the data set
  y.train = data$cnt[indices]
  x.train = data$hum[indices]
  y.test = data$cnt[-indices]
  x.test = data$hum[-indices]

  for (i in 1:length(gammas)) {
```

```
    pMSE.sqrt[k,i] = sqrt(pMSE(gammas[i]))
  }
}

# Plot
par(mfrow = c(2,5))
gammas.min = array(dim=K)
for (k in 1:K) {
  plot(gammas, pMSE.sqrt[k,], xlab="gamma", ylab="sqrt(pMSE)", cex.lab=1.5)
  gammas.min[k] = gammas[which.min(pMSE.sqrt[k,])]
}
```

A.7 Assignment 1(vi)

```
set.seed(321) # For reproducibility

# Start with the model from earlier
m = lm((data$cnt^lambda-1)/lambda ~ data$hum)
b0.star = m$coefficients[1] # original B_0
b1.star = m$coefficients[2] # origianal B_1
s = summary(m)$sigma

B = 2000 # Number of iterations
alpha = 0.2

coeffs = array(dim=c(B,2)) # The parameter estimates for each iteration.
b0.included = 0 # The number of confidence intervals for b0 that contains
    the "actual" value b0.star
b1.included = 0 # The number of confidence intervals for b1 that contains
    the "actual" value b1.star
for (i in 1:B) {
  # Generate predictions
  eps = rnorm(length(data$hum),0,s)
  y.new = b0.star + b1.star * data$hum + eps # Transformed predictions

  # Fit a new model
  m = lm(y.new ~ data$hum)
  # Extract coefficients
  coeffs[i,] = m$coefficients
  # Generate confidence intervals
  # Note that these are for the transformed response variable, but it doesn
      't matter as we're only interested in the
  # proportion of the intervals that contain the actual value, not the
      confidence intervals themselves.
  interval = confint(m, level = 1-alpha)
  if (interval[1,1] <= b0.star & b0.star <= interval[1,2]) {
    b0.included = b0.included + 1
  }
  if (interval[2,1] <= b1.star & b1.star <= interval[2,2]) {
    b1.included = b1.included + 1
  }
}

b0.prop = b0.included / B # The proportion of confidence intervals for b0
    that contains the "actual" value b0.star  = 0.8105
b1.prop = b1.included / B # The proportion of confidence intervals for b1
    that contains the "actual" value b1.star  = 0.8075
```