C1M4_autograded

January 8, 2022

1 Module 4: Autograded Assignment

1.0.1 Outline:

Here are the objectives of this assignment:

1. Review skills in data cleaning and preparation.
2. Create training and test sets for model predictions.
3. Predict on training and test sets.
4. Learn about and visualize Prediction Intervals for linear models.
5. Understand some limitations of predictions with linear models.

Here are some general tips:

1. Read the questions carefully to understand what is being asked.
2. When you feel that your work is completed, feel free to hit the `Validate` button to see your results on the *visible* unit tests. If you have questions about unit testing, please refer to the "Module 0: Introduction" notebook provided as an optional resource for this course. In this assignment, there are hidden unit tests that check your code. You will not recieve any feedback for failed hidden unit tests until the assignment is submitted. **Do not misinterpret the feedback from visible unit tests as all possible tests for a given question–write your code carefully!**
3. Before submitting, we recommend restarting the kernel and running all the cells in order that they appear to make sure that there are no additional bugs in your code.
4. There are 50 points in this assignment.

[2]:
```
# This cell loads the necesary libraries for this assignment
library(testthat)
library(tidyverse)
library(RCurl) #a package that includes the function getURL(), which allows for
 ↪reading data from github.
library(ggplot2)
```

2 Problem 1: Prediction and Octopi (50 points)

Brian has just adopted a baby octopus and wants to know how much it will grow as it gets older. Thankfully, researchers at the University of Florida have provided us with data just for this occassion. The researchers were measuring the number of beak increments per octopi age (in days) and weight (in grams), but we can use the same data to see how their weight was affected by their age.

One thing worth pointing out is that the original data is sorted from youngest to oldest. Later, we will be splitting the data into a training and test sets. To make sure that we don't introduce some systemic error, such as only looking at the youngest octopi, we should randomize the order of the rows. Now let's load in the data!

[3]:
```
# Load in the data
octopus.data = read.table("octopi.dat")

names(octopus.data) = c("weight", "age", "beak_increments", "beak_measured")

# Shuffle the data so it isn't ordered
set.seed(42)
randomize.rows = sample(nrow(octopus.data))
octo.data = octopus.data[randomize.rows, ]
head(octo.data)
```

A data.frame: 6 × 4

	weight <dbl>	age <int>	beak_increments <int>	beak_measured <int>
37	62.4	122	123	1
1	7.6	63	63	2
25	77.1	105	107	1
10	4.4	63	61	1
36	60.0	122	124	2
18	62.9	87	87	2

1. (a) Removing Doubles (5 points) You many notice that our data has a variable named `beak_measured`. Like human jaws, each octopus beak has two parts, so the researchers marked down which they were measuring. For our purposes, that means each age and weight measurement will appear twice in the data. The easiest way to correct this is to remove one of each of those measurements.

Restrict your data to rows where `beak_measured == 1`. Save the reduced data as `octo.data.reduced`.

[8]:
```
octo.data.reduced = NA
# your code here
octo.data.reduced = octo.data %>% filter(beak_measured == 1)
octo.data.reduced
```

A data.frame: 19 × 4

weight	age	beak_increments	beak_measured
<dbl>	<int>	<int>	<int>
62.4	122	123	1
77.1	105	107	1
4.4	63	61	1
7.9	63	67	1
99.1	122	122	1
15.3	87	87	1
9.3	87	85	1
67.9	105	105	1
7.6	63	65	1
9.6	63	66	1
80.2	122	121	1
83.9	122	122	1
46.6	87	87	1
7.5	63	58	1
52.9	105	104	1
49.8	105	103	1
71.9	105	101	1
78.8	122	121	1
68.3	87	84	1

[9]:
```
# Test Cell
if(test_that("Size of cleaned data", {expect_equal(nrow(octo.data.reduced),␣
↪19)})){
    print("Data is the correct number of rows. ")
    print("Make sure this is correct, the rest of the questions depend on these␣
↪data.")
}else{
    print("The reduced data doesn't have the correct number of rows. It should␣
↪have 19.")
}
```

```
[1] "Data is the correct number of rows. "
[1] "Make sure this is correct, the rest of the questions depend on these data."
```

1. (b) Training and Test Sets (5 points) We have our full dataset, but it is often useful to split that into two smaller datasets, one for training the model and the another for testing it. There are many reasons for this, but the main one is that having a test set allows us to see how the model performs with data that it has never seen before.

Split your data into a training and test set and store them in `octo.train` and `octo.test` respectively. The training set should be the first 80% of the rows (rounded down) and the test set should be the remaining 20% of the rows. Keep in mind that the code given above has already shuffled the data.

[15]:

```
octo.train = NA
octo.test = NA
# your code here
n = floor(0.8 * nrow(octo.data.reduced))
octo.train = octo.data.reduced[1:n, ]
octo.test = octo.data.reduced[-(1:n), ]
octo.train
octo.test
```

A data.frame: 15 × 4

	weight <dbl>	age <int>	beak_increments <int>	beak_measured <int>
1	62.4	122	123	1
2	77.1	105	107	1
3	4.4	63	61	1
4	7.9	63	67	1
5	99.1	122	122	1
6	15.3	87	87	1
7	9.3	87	85	1
8	67.9	105	105	1
9	7.6	63	65	1
10	9.6	63	66	1
11	80.2	122	121	1
12	83.9	122	122	1
13	46.6	87	87	1
14	7.5	63	58	1
15	52.9	105	104	1

A data.frame: 4 × 4

	weight <dbl>	age <int>	beak_increments <int>	beak_measured <int>
16	49.8	105	103	1
17	71.9	105	101	1
18	78.8	122	121	1
19	68.3	87	84	1

[16]:

```
# Test Cell
if(test_that("Check train and test sets are correct size",␣
↪{expect_equal(nrow(octo.train), 15)
                                                              ␣
↪expect_equal(nrow(octo.test), 4)})){
    print("The training and test sets are the correct sizes.")
    print("Make sure these contain the correct data! All following problems␣
↪depend on these being correct.")
}else{
    print("Incorrect sizes. Make sure you round down for the size of the␣
↪training set.")
    print("Tip: Use the floor() function.")
}
```

```
[1] "The training and test sets are the correct sizes."
[1] "Make sure these contain the correct data! All following problems depend on
these being correct."
```

1. (c) Predicting on Observed Data (15 points) In order for Brian to know how much his octopus will weigh, we need to fit a linear model to the training data with `weight` as the response and `age` as the predictor. Do this, then compute the predictions (called fitted values) and 95% Prediction Intervals for the **training** data.

Store the predicted (fitted) values in the variable `octo.train.fit`, the lower bounds of the prediction intervals in `octo.train.lower` and the upper bounds in `octo.train.upper`.

[17]:
```
octo.lmod = NA

octo.train.fit = NA
octo.train.upper = NA
octo.train.lower = NA

# your code here
octo.lmod = lm(weight~age, data= octo.train)
summary(octo.lmod)
```

```
Call:
lm(formula = weight ~ age, data = octo.train)

Residuals:
    Min      1Q  Median      3Q     Max
-26.370  -3.731   3.175   6.998  17.921

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) -77.9568    14.1125  -5.524 9.81e-05 ***
age           1.3061     0.1487   8.785 7.91e-07 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 13.62 on 13 degrees of freedom
Multiple R-squared:  0.8558,Adjusted R-squared:  0.8447
F-statistic: 77.18 on 1 and 13 DF,  p-value: 7.91e-07
```

[26]:
```
train_predict = predict(octo.lmod, new = octo.train['age'],␣
 ↪interval='prediction')

octo.train.fit = train_predict[ ,'fit']
octo.train.upper = train_predict[ ,'upr']
octo.train.lower = train_predict[ ,'lwr']
```

```
train_predict
```

A matrix: 15 × 3 of type dbl

	fit	lwr	upr
1	81.38212	49.50584	113.25839
2	59.17915	28.51229	89.84600
3	4.32475	-27.44315	36.09265
4	4.32475	-27.44315	36.09265
5	81.38212	49.50584	113.25839
6	35.67012	5.25049	66.08975
7	35.67012	5.25049	66.08975
8	59.17915	28.51229	89.84600
9	4.32475	-27.44315	36.09265
10	4.32475	-27.44315	36.09265
11	81.38212	49.50584	113.25839
12	81.38212	49.50584	113.25839
13	35.67012	5.25049	66.08975
14	4.32475	-27.44315	36.09265
15	59.17915	28.51229	89.84600

[27]:
```
# Test Cell
if(test_that("Testing number of predictions", {expect_equal(length(octo.train.
 ↪fit), 15)})){
    print("Correct number of predictions.")
    print("Make sure your Prediction Intervals are for 95%.")
}else{
    print("Incorrect number of predictions.")
    print("Make sure you're predicting on the training set.")
}
# This cell has hidden test cases that will run after submission.
```

```
[1] "Correct number of predictions."
[1] "Make sure your Prediction Intervals are for 95%."
```

1. (d) Predicting on Unobserved Data (15 points) Now compute the predictions and 95% prediction intervals for the test set. Store the respected values in `octo.test.fit`, `octo.test.lower` and `octo.test.upper`.

[30]:
```
octo.test.fit = NA
octo.test.lower = NA
octo.test.upper = NA

# # your code here
test_predict = predict(octo.lmod, new = octo.test['age'], interval='prediction')

octo.test.fit = test_predict[ ,'fit']
octo.test.lower = test_predict[ ,'lwr']
octo.test.upper = test_predict[ ,'upr']
```

```
test_predict
```

A matrix: 4 × 3 of type dbl

	fit	lwr	upr
16	59.17915	28.51229	89.84600
17	59.17915	28.51229	89.84600
18	81.38212	49.50584	113.25839
19	35.67012	5.25049	66.08975

[31]:
```
# Test Cell
# This cell has hidden test cases that will run after submission.
```

1. (e) Visualization (5 points) We've calculated our prediction intervals, but that doesn't really help us understand what they are. It can be much more useful to visualize these intervals, to really understand what they mean.

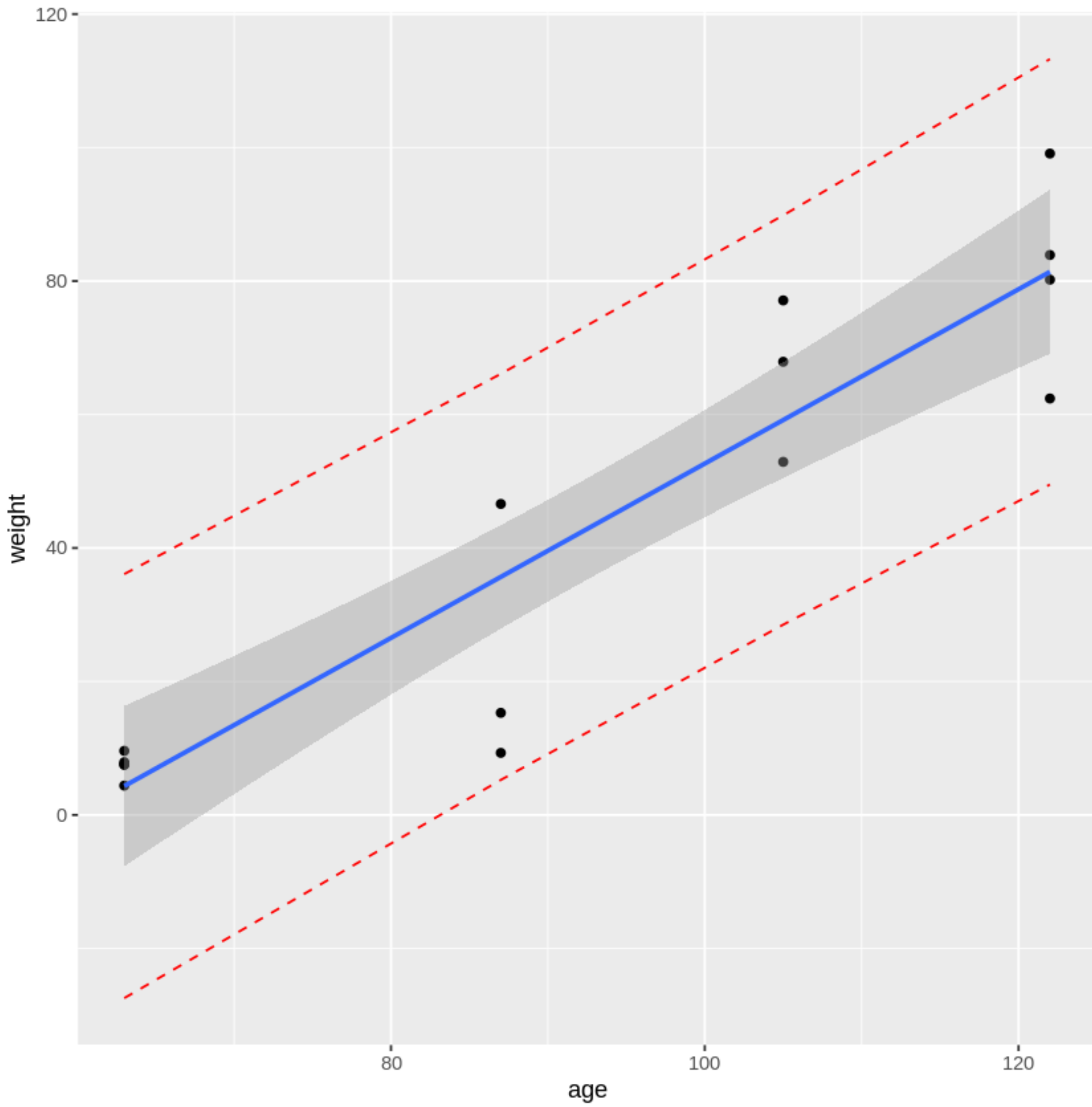
Plot a scatterplot of the data, with `age` on the x-axis and `weight` on the y-axis, with correctly labeled axes. Add a straight line to represent the fit our the linear model `octo.lmod` with a band for the confidence interval. Then add dotted lines for the upper and lower bounds of the prediction intervals. Use ggplot and save your final plot as `octo.plot`.

A site to help with this can be found here.

[36]:
```
# your code here
final_df = cbind(octo.train, train_predict)

octo.plot = ggplot(final_df, aes(age, weight)) +
    geom_point() +
    geom_line(aes(y=lwr), color = "red", linetype = "dashed")+
    geom_line(aes(y=upr), color = "red", linetype = "dashed")+
    geom_smooth(method=lm, se=TRUE)
octo.plot
```

```
`geom_smooth()` using formula 'y ~ x'
```




```
[47]: # Test Cell
      # This cell has hidden test cases that will run after submission.
```

1. (f) How large can an octopus get? (5 points) According to the internet, an octopus of this type can weigh up to 5kg and has a maximum lifespan of 2 years, which we can approximate to 730 days. According to our model, how much would Brian's octopus weigh if it got that old? Store this value in the `brians.old.octopus.weight`variable.

Does this value agrees with the provided weight? Think about potential limitations of our model.

```
[48]: age = data.frame(age = 730)
      brians.old.octopus.weight = NA
```

```
# your code here
new_pred = predict(octo.lmod, new=age, interval="prediction")
brians.old.octopus.weight = new_pred[ ,'fit']
brians.old.octopus.weight
```

875.464796031922

[49]:
```
# Test Cell
# This cell has hidden test cases that will run after submission.
```

[ ]:
```
```