C1M1_autograded

December 12, 2021

1 Module 1 - Autograded Assignment

1.0.1 Outline:

Here are the objectives of this assignment:

1. Familiarize yourself with the basics of R and Jupyter Notebooks.
2. Visualize data using scatterplots, histograms and density plots.
3. Center, Scale and Standardize predictors so they end up with similar distributions.
4. Fit a best fit line to data and plot the results.
5. Observe the effects of standardized data on best fit lines.
6. Reinforce our understanding of the "linear" part of linear regression 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 total points in this assignment.

[4]:
```
# This cell loads the necesary libraries for this assignment
library(testthat)
library(tidyverse)
```

```
Error in get(genname, envir = envir) : object 'testthat_print' not found

  Attaching packages                                    tidyverse
1.3.0

 ggplot2 3.3.0      purrr   0.3.4
 tibble  3.0.1      dplyr   0.8.5
 tidyr   1.0.2      stringr 1.4.0
 readr   1.3.1      forcats 0.5.0
```

```
  Conflicts
tidyverse_conflicts()
  dplyr::filter()  masks stats::filter()
  purrr::is_null() masks
testthat::is_null()
  dplyr::lag()     masks stats::lag()
  dplyr::matches() masks
tidyr::matches(), testthat::matches()
```

2 Problem 1: The Basics of Standardizing Data

Welcome to your first problem on your first autograded assignment! Don't worry, they aren't that bad. Just know that, with respect to the autograder, there are three types of cells:

- Read-Only Cells: You can run these cells, but can't change any of the code or markdown inside of them. These will be used for us to give you questions or code that you will need.
- Autograded Answer Cells: This is where you will write your code to answer the prompts. Be sure to answer the prompts with the specified methods and variables. Otherwise, you won't receive credit. If needed, you can always create additional cells to do other work.
- Autograded Test Cells: These cells test your code and are usually located just after an Autograded Answer cell. For the most part, you can't interact with these cells (including delete them, sorry!). Note that there can be both visible and hidden tests in any autograded test cell. Make sure your code passes all the visible tests. You will only get feedback on the hidden tests once you've submitted the assignment.

Now on to the actual problem! We will be analyzing a dataset of how much a person's age and income affects their opinion of turtles. That's right, some people don't like turtles, and in Problem 2, we'll investigate this important issue! But, before that, we'll first learn to *standardize* the predictor variables. By the end of this problem, we will have standardized both the `age` and `income` predictors in order to begin addressing the "crisis".

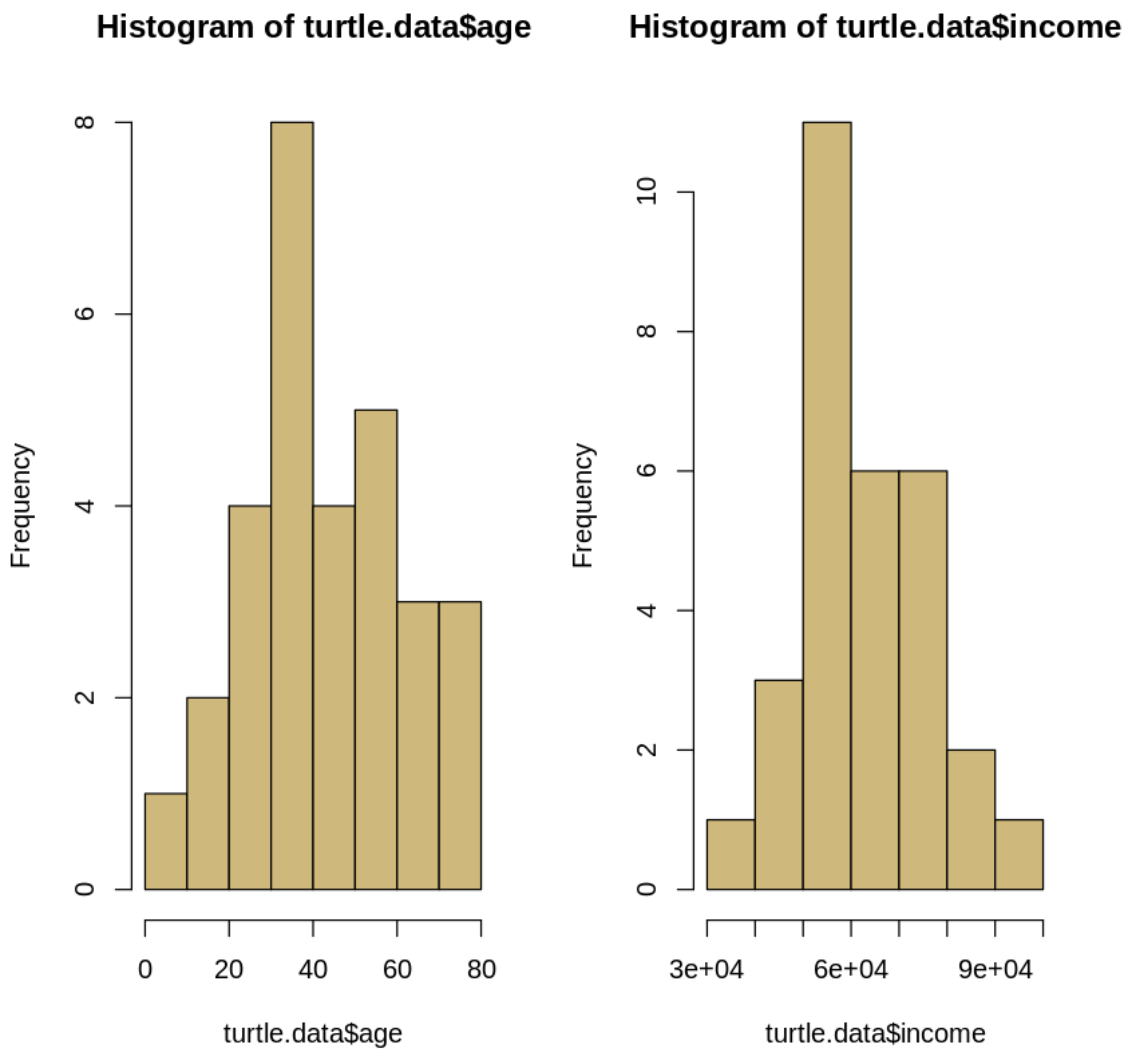
In the code cells below, we load in the data and visualize those two features with some histograms and a scatter plot. Think about what information we can gather about the two predictors just from these plots.

[5]:
```
# Load the data
turtle.data = read.csv("turtle.csv")
head(turtle.data)
```

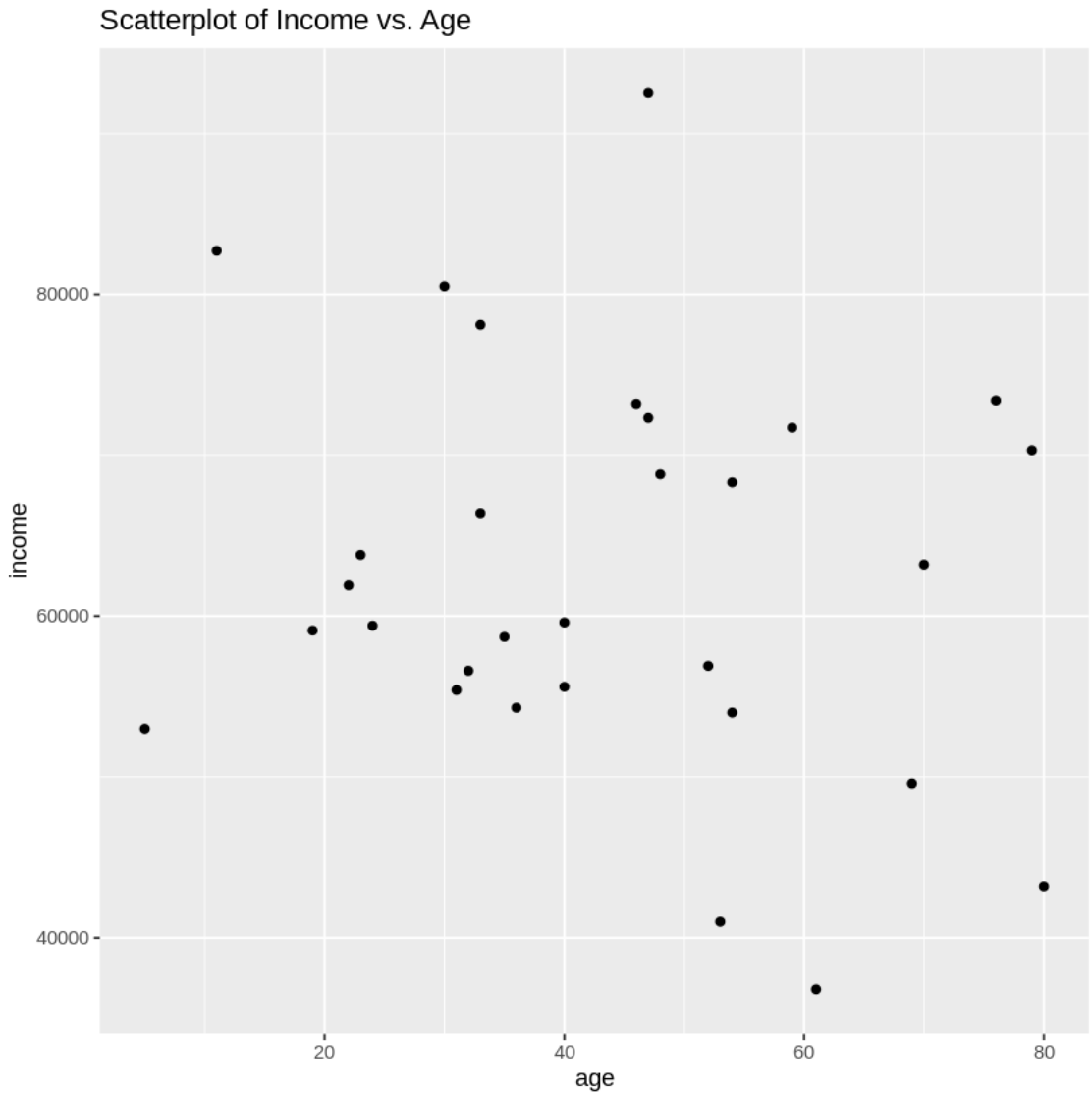
A data.frame: 6 × 3

	age <int>	income <int>	turtle_rating <int>
1	33	66400	1
2	40	55600	-2
3	76	73400	-1
4	46	73200	-4
5	47	72300	-4
6	79	70300	1

[6]:
```
# Histograms of the age and income features
par(mfrow = c(1, 2))
hist(turtle.data$age, col="#CFB87C")
hist(turtle.data$income, col="#CFB87C")
```



[7]:
```
g = ggplot(turtle.data, aes(x=age, y=income)) +
    # This function adds the points for the scatterplot
    geom_point() +
    # This function adds the title (and potentially other labels)
    labs(title="Scatterplot of Income vs. Age")
g
```



1. (a) Centering the data (9 points) Well, that plot looks nice. But notice the scale of the axes. The x-axis has a range of about $[10, 90]$ and the y-axis has a range of $[20,000, 90,000]$. These ranges raise two issues with respect to the interpretation of a regression model:

1. Recall that the intercept parameter is interpreted as the mean of the response when each predictor is zero. But in this case, zero is not a meaningful value for the `age` predictor.

2. Recall that a slope parameter in multiple linear regression is interpreted as the average change in the response for a *one-unit increase* in the value of the corresponding predictor, holding other predictors constant. But, for this problem, do we really care about how a "one dollar increase in (yearly) salary" impacts turtle rating? Probably not. It would be more convenient to interpret the change turtle rating with respect to bigger jumps in income, say, one thousand dollar increases.

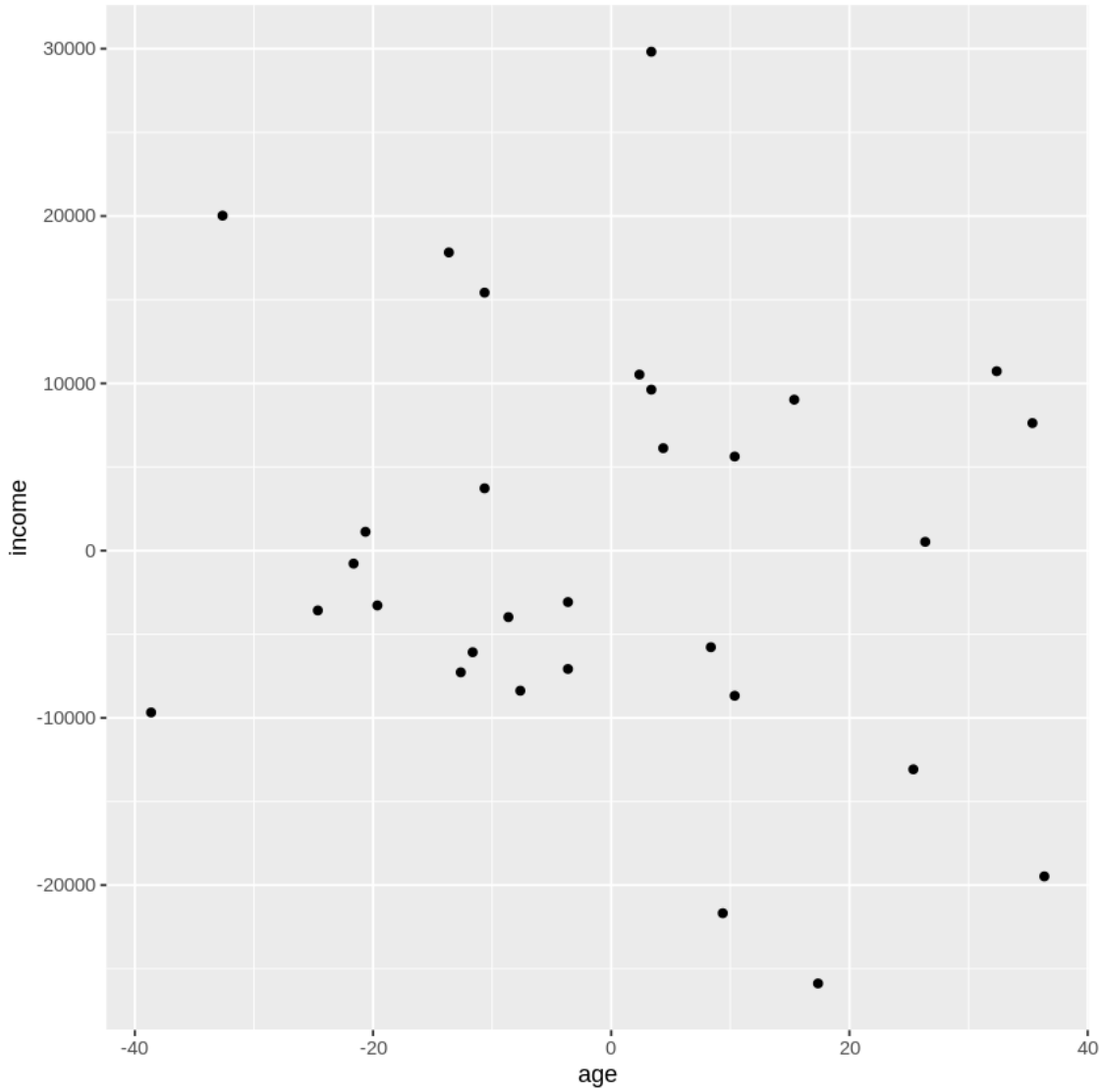
To better assess and interpret the affect of the predictors on `turtle_rating`, we can changes our scales! Let's start by centering our data, which requires setting the mean of each predictor equal to 0 (note how that changes the interpretation of the intercept parameter!). To do this, we can subtract the mean value of each predictor from each value. So, for instance:

$$x_{1,age}^{centered} = x_{1,age} - \bar{x}_{age}$$

Center the `age` and `income` data and save the updated vectors into `age.centered` and `income.centered` respectively. Then recreate the plot above with the centered data, which should be stored in `g.centered`. Make sure to use ggplot!

Tip for plotting with ggplot: The `ggplot()` function requires the data to be in a `data.frame` object. It might be useful to save your centered vectors into a data.frame with `df.centered = data.frame(age=age.centered, income=income.centered)`.

[10]:
```
age.centered = NA
income.centered = NA

g.centered = NA # Look at the code cell above for the basic ggplot() syntax.

# your code here
age.centered = turtle.data$age - mean(turtle.data$age)
income.centered = turtle.data$income - mean(turtle.data$income)
df.centered = data.frame(age=age.centered, income=income.centered)
g.centered = ggplot(df.centered, aes(x=age, y=income)) + geom_point()
g.centered
```



[11]:
```
# Test Cell

if(test_that("Checking the class of the plot", {expect_is(g.centered,␣
 ↪"ggplot")})){
    print("Good job.")
    print("Make sure your answers are correct, there are hidden tests and you␣
 ↪won't recieve feedback on them until you submit your assignment!")
}else{
    print("Make sure to use ggplot for your plotting.")
    print("It's annoying at first, but it's a very powerful visualization tool␣
 ↪once you get the hang of it.")
}
```

```
# Note: Each question may have some hidden tests!
# Make sure you're confident in your answers!
```

```
[1] "Good job."
[1] "Make sure your answers are correct, there are hidden tests and you won't
recieve feedback on them until you submit your assignment!"
```

1. (b) Scaling the Data (9 points) Our plot shows that both `age` and `income` are centered at 0, which is good, since it will fix our intercept interpretation. But the range of the predictors is still very different, which means that our slope interpretations will still be awkward. How do we address that? Well, we know that the "spread" of a variable is represented by its standard deviation. Therefore, if we divide the centered predictors by their standard deviation, the new "standardized" `age` and `income` variables will both have the same spread. In math terms:

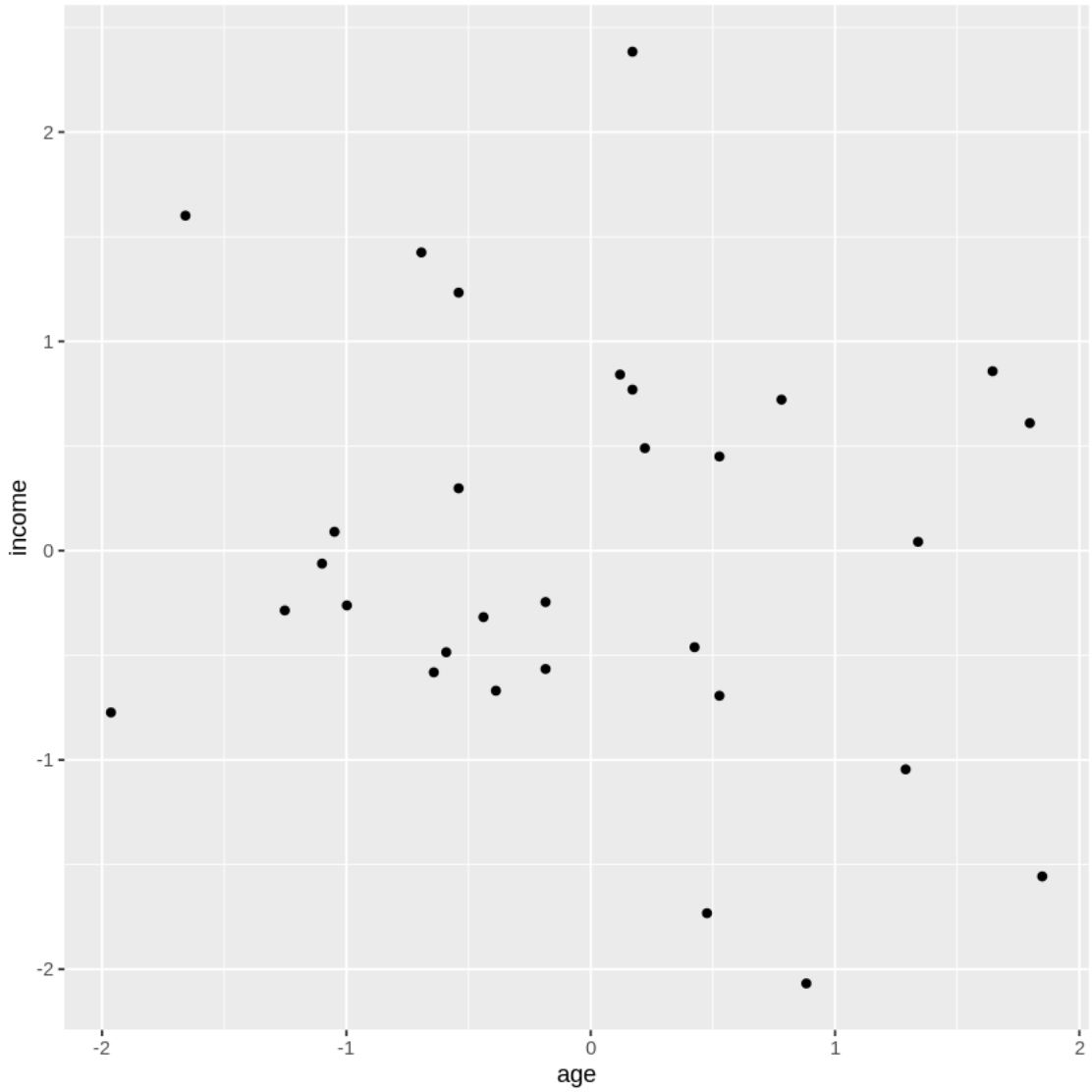
$$x_{i,age}^{scaled} = \frac{x_{i,age}}{std.dev(x_{age})}$$

Note that the process of both centering and scaling data is called standardizing.

$$x_{i,age}^{standardized} = \frac{x_{i,age}^{centered}}{std.dev(x_{age})} = \frac{x_{i,age} - \bar{x}_{age}}{std.dev(x_{age})}$$

Standardize both the `age` and `income` predictors, and save the standardized vectors into `age.stand` and `income.stand` respectively. Then plot our scaled data using ggplot. Save your plot as `g.stand`.

[14]:
```
age.stand = NA
income.stand = NA
g.stand = NA

# your code here
age.stand = age.centered / sd(turtle.data$age)
income.stand = income.centered / sd(turtle.data$income)
df.stand = data.frame(age = age.stand, income = income.stand)
g.stand = ggplot(df.stand, aes(x=age, y=income)) + geom_point()
g.stand
```



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

Note that this should fix the interpretation of the slope parameters: if we were to use the standardized variables in a regression, a "one-unit increase" would be a "one-standard deviation increase"!

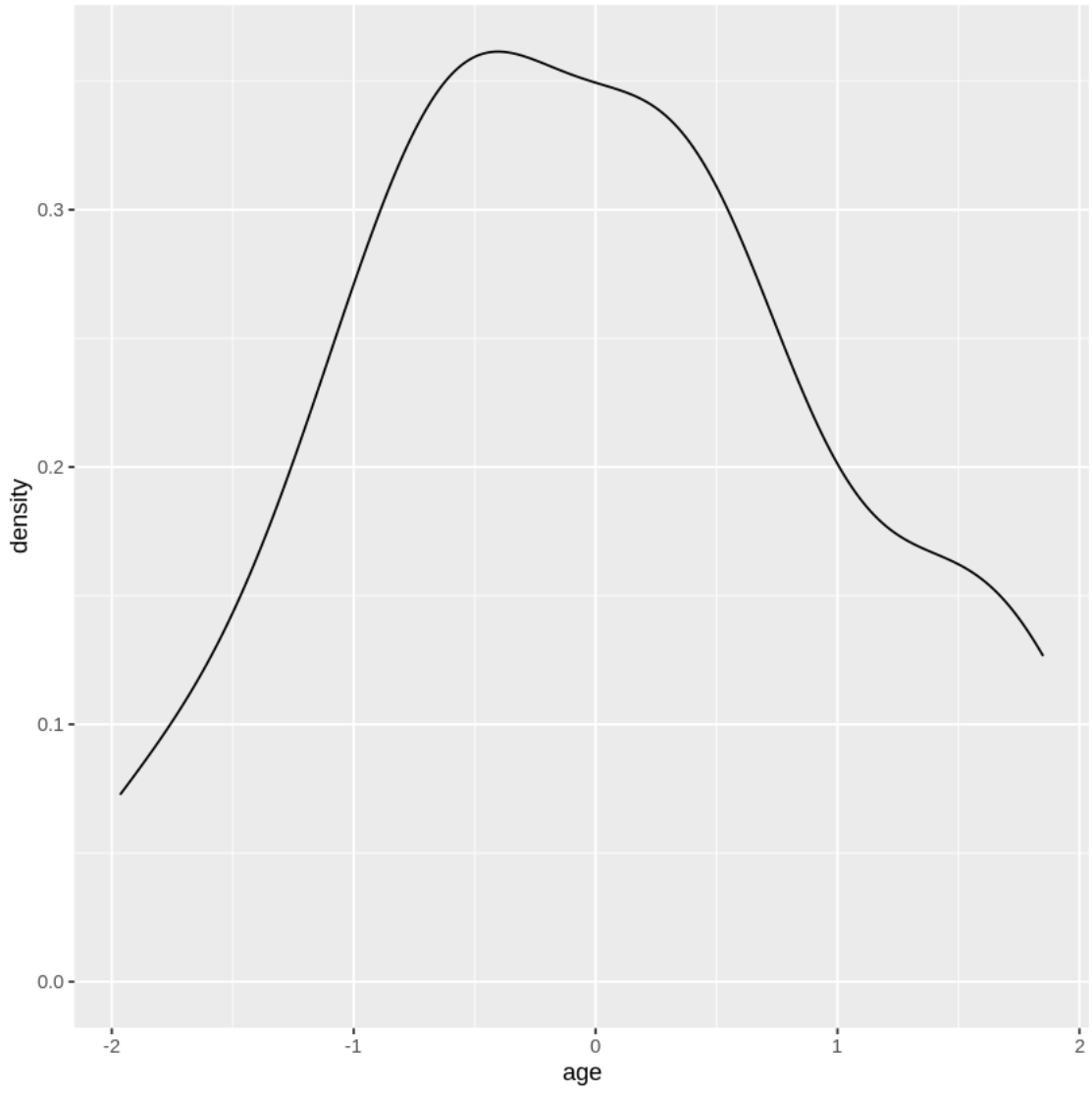
1. (c) Wait what? So we just did a bunch of math, and now people can have a negative age and income? Of course not! Instead, we've just changed the units; the new units measure income but are centered at zero, with a "standard" spread. In addition, the order of the data is still preserved. Let's visualize the resulting predictors!

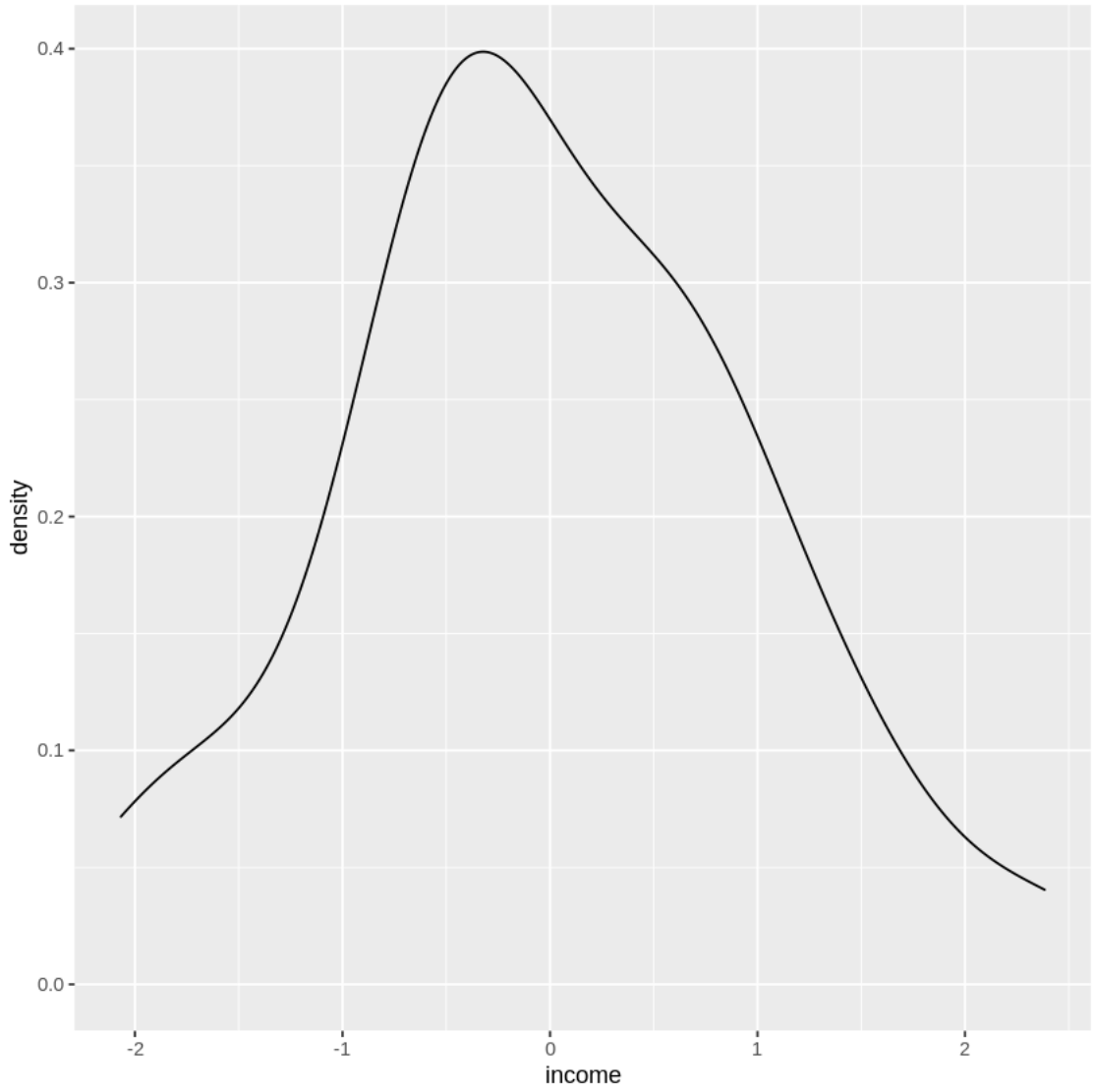
Recall at the beginning of this problem, there were some histograms for `age` and `income`. Using ggplot, create density plots (using `geom_density()`) for the standardized versions of those two

predictors. Overlay the standard normal curve onto both of these plots. Save these plots as `g.age` and `g.income`.

What do you notice?

[19]:
```
g.age = NA
g.income = NA

# your code here
g.age = ggplot(df.stand, aes(x=age)) + geom_density()
g.income = ggplot(df.stand, aes(x=income)) + geom_density()

g.age
g.income
```





3 Problem 2: Using Our Standardized Data

Now that we have standardized versions of the predictors, let's see how they are related to the response, `turtle_rating`.

In problem 1, we made a claim that standardizing the data could affect our best fit line. Let's see if that claim is true.

2. (a) Slope of the Original Data (10 points) Using the original data (`turtle.data`), create a scatterplot with ggplot named `g.turtle` that has `turtle_rating` on the y-axis and `income` on the x-axis. Then add the best fit line by adding the layer `geom_smooth(method="lm", se=FALSE,`

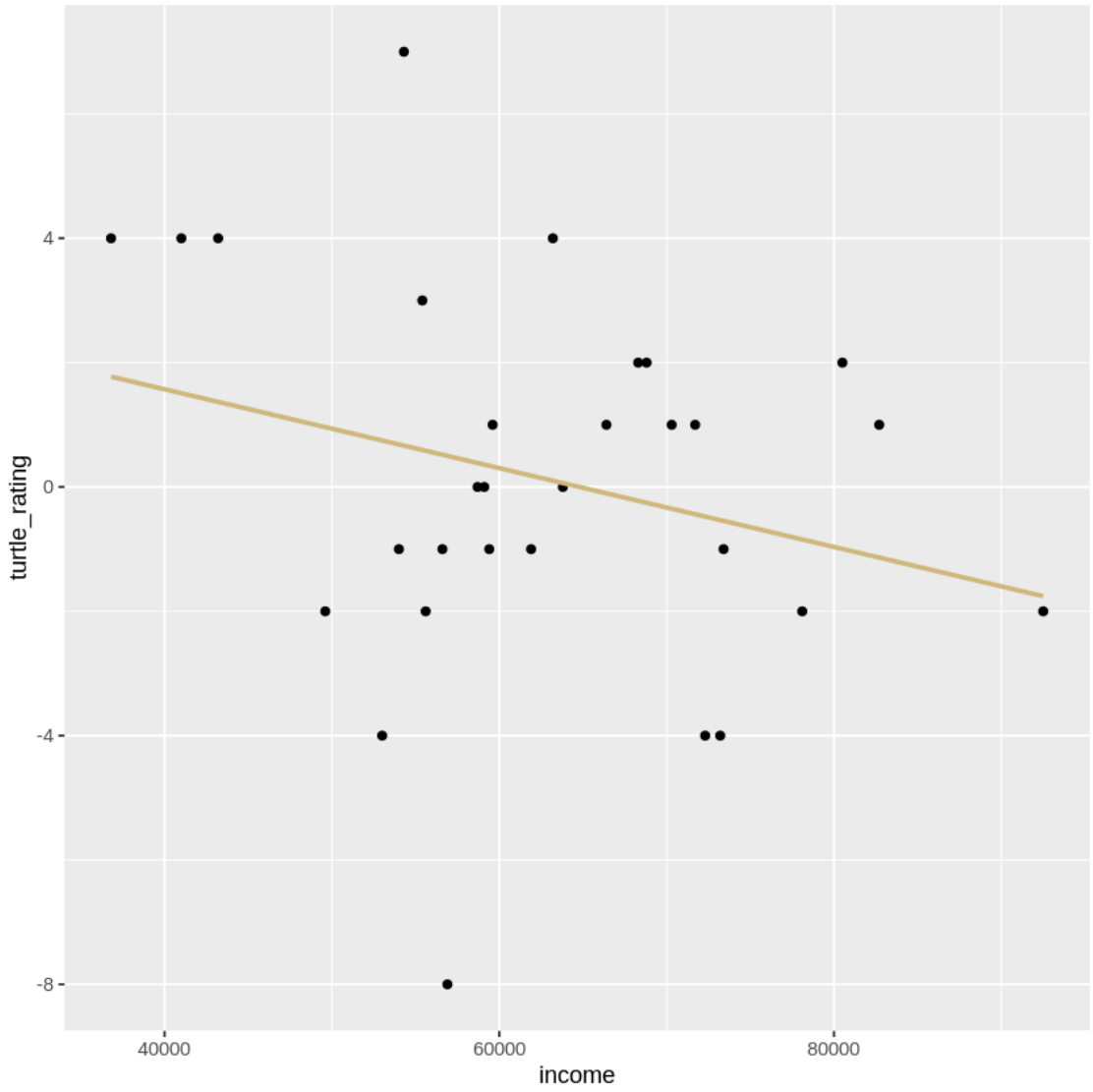
color="#CFB87C").

To find the exact slope of our best fit line, we can use the `lm()` function. We will be learning more about this function in future modules, but for now, all we need to know is that it calculates the best fit line. Use the form `lm(response.name~predictor.name, data=dataset.name)$coefficients[2]` to get the slope of our line. Store that value in `turtle.slope`.

[23]:
```
g.turtle = NA
g.turtle = NA

# your code here
g.turtle = ggplot(turtle.data, aes(x=income, y=turtle_rating)) + geom_point() +
           geom_smooth(method="lm", se=FALSE, color= "#CFB87C")

# your code here
turtle.slope = lm(turtle_rating~income, data=turtle.data)$coefficients[2]

g.turtle
turtle.slope
```

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

income: -6.33941858878002e-05



[24]:
```
# Test Cell
if(!test_that("Checking class of plot", {expect_is(g.turtle, "ggplot")})){
    print("Make sure your plots are made in ggplot.")
}
# This cell has hidden test cases that will run after submission.
```

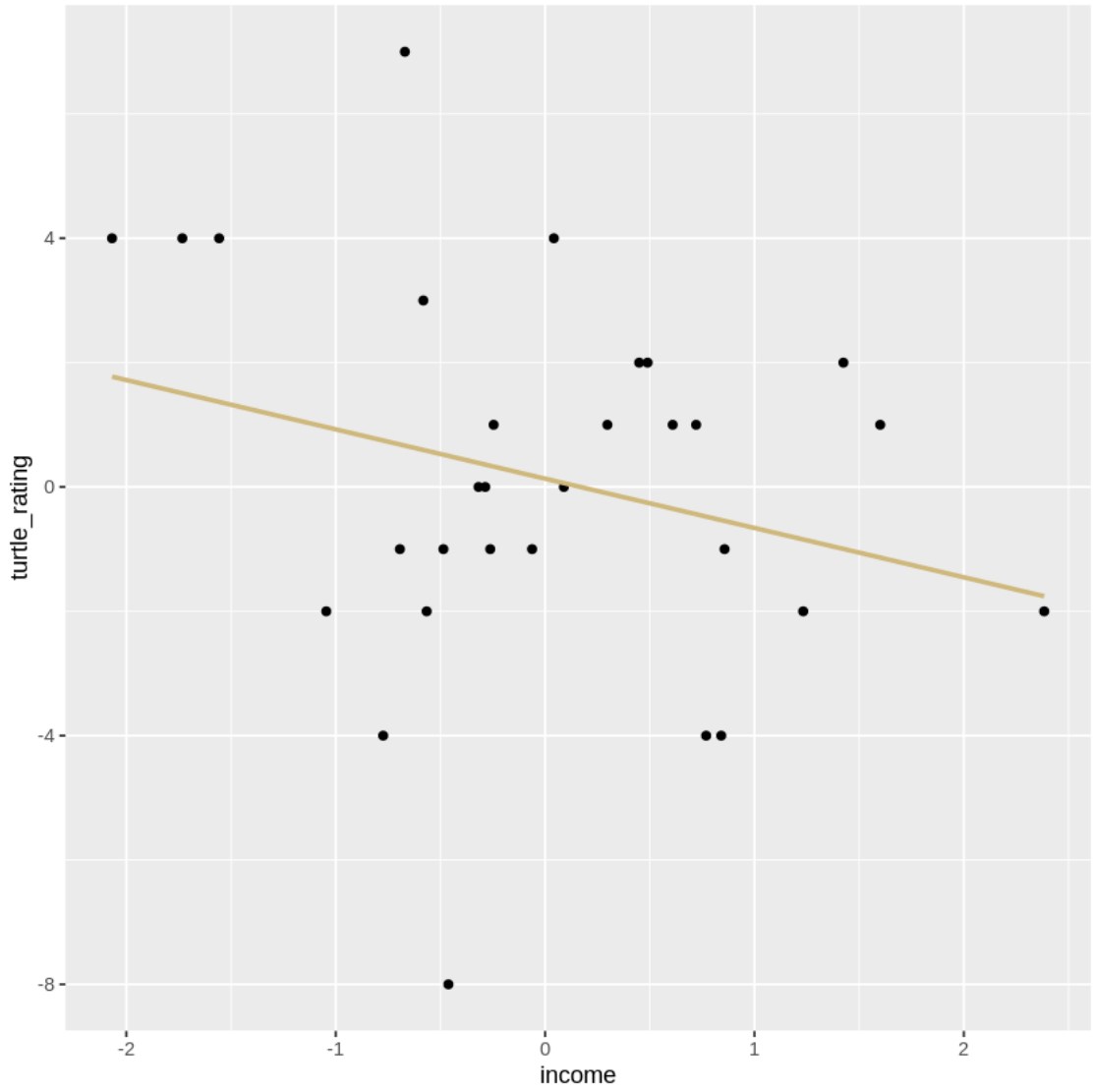
2. (b) Slope of the Standardized Data (10 points) Repeat the above with the standardized income data. Store the slope in `turtle.slope.stand` and the plot in `g.turtle.stand`.

Take note of whether the slope has changed with the use of the standardized predictors!

[28]:
```
g.turtle.stand = NA
turtle.slope.stand = NA

# your code here
turt_rat = turtle.data$turtle_rating
df.stand2 = data.frame(income = income.stand, turtle_rating = turt_rat)
g.turtle.stand = ggplot(df.stand2, aes(x=income, y=turtle_rating)) +␣
 ↪geom_point() +
           geom_smooth(method="lm", se=FALSE, color= "#CFB87C")

# your code here
turtle.slope.stand = lm(turtle_rating~income, data=df.stand2)$coefficients[2]

g.turtle.stand
turtle.slope.stand
```

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

income: -0.793188009334631



[ ]:
```
# Test Cell
test_that("Check class of plot", {expect_is(g.turtle.stand, "ggplot")})
# This cell has hidden test cases that will run after submission.
```

4 Problem 3: What Counts As A Linear Model? (12 points)

When thinking about linear regression models, the "linear" term can lead to confusion. To make sure you have a strong grasp of what is and is not considered linear, please look at the following models. In the code cell below, answer `TRUE` if the corresponding model is a "linear regression model" and `FALSE` if it is not.

1. $y_i = \beta_0$

2. $y_i = \beta_0 + \beta_1 x_{i,1} + \beta_2^2 x_{i,2}^2$
3. $y_i = \beta_0 + \beta_1^{x_i}$
4. $y_i = \beta_0 + \log(\beta_1 x_i)$
5. $y_i = \beta_0 \log(\beta_1 x_i)$
6. $y_i = \beta_0 + \beta_1 \sin(x_{i,1}) + \beta_2 e^{x_{i2}} + \beta_3 \log(x_{i,3})$

[33]:
```
# Answer each question by replacing each NA with the corresponding boolean of␣
 ↪TRUE or FALSE.
prob.2.1 = TRUE

prob.2.2 = TRUE

prob.2.3 = FALSE

prob.2.4 = TRUE

prob.2.5 = FALSE

prob.2.6 = TRUE

# your code here
```

[ ]:
```
# Test Cell

# Make sure the type of the answers is boolean
if(test_that("Check answer types", {expect_is(prob.2.1, "logical")})){
    print("Answers are booleans")
}else{
    print("Answers aren't booleans! Make sure they are, or else they will be␣
 ↪incorrect.")
}
# This cell has hidden test cases that will run after submission.
```

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

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

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

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