Lab 7

Tyron Samaroo

7:09PM April 8, 2020

Run three OLS models on the boston housing data using all available features:

(1) where the response is medv,
(2) where the response is the log base 10 of medv and
(3) where the response is the square root of medv.

Compare the two models on oos se of the residuals. Use K = 5 to create a training-test split. Which model is better? Answer: The normal model with response medv did better.

```
set.seed(404)
Boston = MASS::Boston
x = Boston[,2:13]
y = Boston[,13]
K = 5
train_test_split = (K - 1) / K # even splits
num_train_points = round(train_test_split * nrow(Boston))  #Takes 80% of 506 to nearest hold number
index_of_train = sample(x = 1:nrow(Boston), size = num_train_points, replace = FALSE) # 405 points
index_of_test = setdiff(1:nrow(Boston), index_of_train) # the other 101 points

#check
pacman::p_load(testthat)
expect_equal(nrow(Boston), length(index_of_train) + length(index_of_test))

x_train = x[index_of_train, ]
x_test = x[index_of_test, ]

y_train = y[index_of_train]
y_test = y[index_of_test]


Dtrain = data.frame(x= x_train, y = y_train) # Dtrain gets y and x

model1 = lm(y ~ ., Dtrain)
model2 = lm(log10(y) ~ . ,Dtrain)
model3 = lm(sqrt(y) ~ . ,Dtrain)

y_hat_g1 = predict(model1, data.frame(x = x_test))
oos_se1 = sd(y_test - y_hat_g1)


y_hat_g2 = predict(model2, data.frame(x = x_test))
oos_se2 = sd(y_test - y_hat_g2)
```

```
y_hat_g3 = predict(model3, data.frame(x = x_test))
oos_se3 = sd(y_test - y_hat_g3)
```

```
oos_se1
```

```
## [1] 5.053323e-15
```

```
oos_se2
```

```
## [1] 6.100887
```

```
oos_se3
```

```
## [1] 5.461207
```

When evaluating the models out of sample, did you ever extrapolate? Which predictions specifically in your test set were extrapolations? How "bad" were the exrapolations?

We didn't extrapolate strictly speaking since in this my example we were not out of range. In reality should make sure you aren't predicting on a range outside of what was used to build the model.

```
max(y_test)
```

```
## [1] 34.41
```

```
max(y_train)
```

```
## [1] 37.97
```

```
min(y_test)
```

```
## [1] 1.92
```

```
min(y_train)
```

```
## [1] 1.73
```

Regardless of the model that came out better, let's consider the response to be raw medv i.e. without taking a transformation. Run a model that includes all squared features (except `chas` which is binary). Does this model do better than vanilla OLS from question 1?

Yes this model do much better than the vanilla OLD from question as it is a more complex model including polynomials.

```
colnames(Boston)
```

```
##  [1] "crim"    "zn"      "indus"   "chas"    "nox"     "rm"      "age"
##  [8] "dis"     "rad"     "tax"     "ptratio" "black"   "lstat"   "medv"
```

```
squared_model_feature = lm(medv ~ poly(crim,2) + poly(zn,2)+ poly(indus,2) + chas + poly(nox,2) + poly(
                           poly(dis,2) + poly(rad,2) + poly(tax,2) + poly(ptratio,2) + poly(black,2)

summary(squared_model_feature)$sigma
```

```
## [1] 3.875402
```

```
summary(squared_model_feature)$r.sq
```

```
## [1] 0.8312355
```

Run a model that includes all polynomail functions of degree 3 of all features (except `chas` which is binary). Does this model do better than the degree 2 polynomial function of the previous question?

This model does better than the degree 2 polynomial function from previous question

```
cubed_model_feature = lm(medv ~ poly(crim,3) + poly(zn,3)+ poly(indus,3) + chas + poly(nox,3) + poly(rm
                           poly(dis,3) + poly(rad,3) + poly(tax,3) + poly(ptratio,3) + poly(black,3) +
summary(cubed_model_feature)$sigma
```

```
## [1] 3.757808
```

```
summary(cubed_model_feature)$r.sq
```

```
## [1] 0.8452889
```

Use polynomial regression to perfectly fitting the following data:

```
n = 10
set.seed(1984)
x = runif(n, 0, 10)
y = 5 + 2 * x + rnorm(n)


mod = lm(y ~ poly(x,9, raw = TRUE))
summary(mod)$sigma
```

```
## [1] NaN
```

```
summary(mod)$r.squared
```

```
## [1] 1
```

```
summary(mod)
```

```
##
## Call:
## lm(formula = y ~ poly(x, 9, raw = TRUE))
##
## Residuals:
## ALL 10 residuals are 0: no residual degrees of freedom!
##
## Coefficients:
##                             Estimate Std. Error t value Pr(>|t|)
## (Intercept)                1.431e+02         NA      NA       NA
## poly(x, 9, raw = TRUE)1 -6.527e+02         NA      NA       NA
## poly(x, 9, raw = TRUE)2  8.690e+02         NA      NA       NA
## poly(x, 9, raw = TRUE)3 -5.551e+02         NA      NA       NA
## poly(x, 9, raw = TRUE)4  2.027e+02         NA      NA       NA
## poly(x, 9, raw = TRUE)5 -4.537e+01         NA      NA       NA
## poly(x, 9, raw = TRUE)6  6.352e+00         NA      NA       NA
## poly(x, 9, raw = TRUE)7 -5.431e-01         NA      NA       NA
## poly(x, 9, raw = TRUE)8  2.597e-02         NA      NA       NA
## poly(x, 9, raw = TRUE)9 -5.331e-04         NA      NA       NA
##
## Residual standard error: NaN on 0 degrees of freedom
## Multiple R-squared:      1,  Adjusted R-squared:    NaN
## F-statistic:   NaN on 9 and 0 DF,  p-value: NA
```

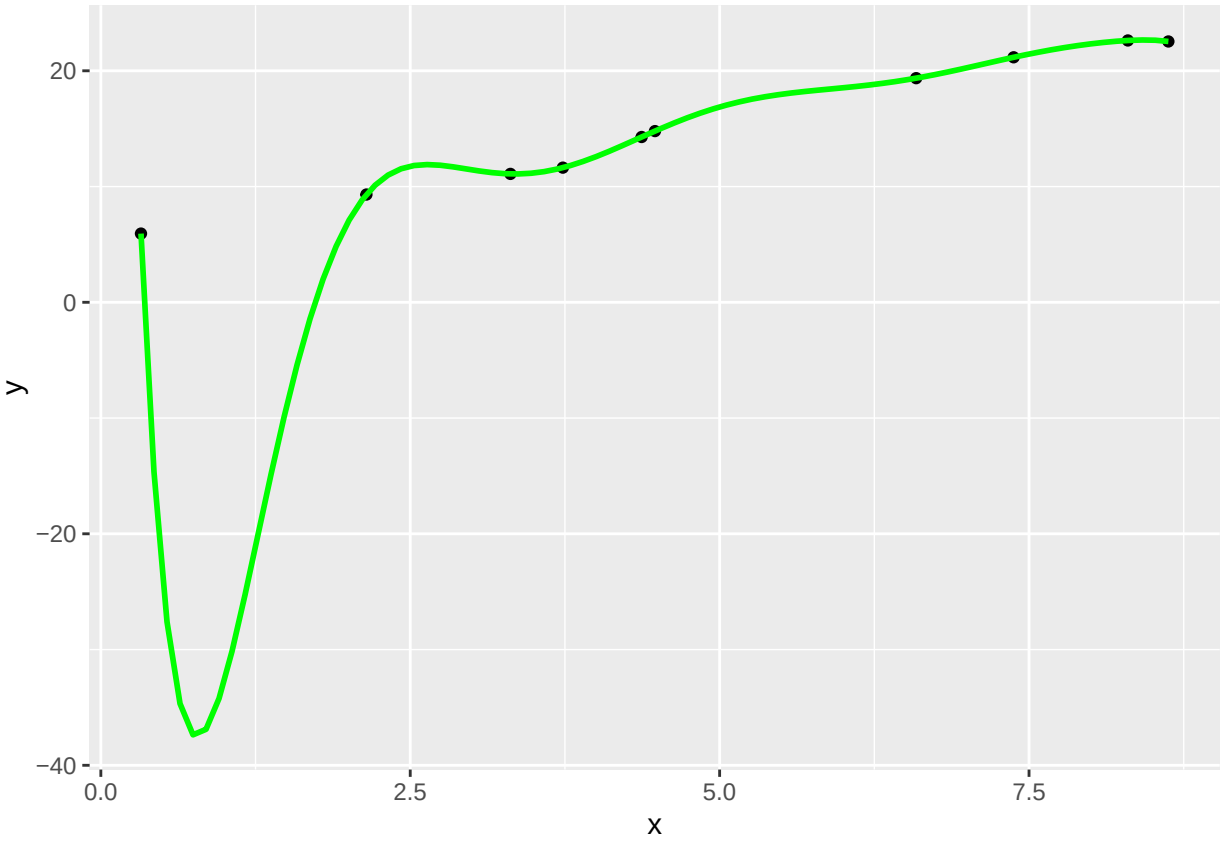
Illustrate Runge's phenomenon in this model by scatterplotting the data with $g(x)$ overlaid in green.

```
pacman::p_load(ggplot2)

  ggplot(,aes(x= x, y=y)) + geom_point() +  geom_smooth(formula = y ~ poly(x,9, raw = TRUE),method = "lm
```

```
## Warning in qt((1 - level)/2, df): NaNs produced
```

```
## Warning in max(ids, na.rm = TRUE): no non-missing arguments to max; returning -
## Inf
```



For the rest of this assignment, I highly recommend using the ggplot cheat sheet as a reference resource. You will see questions that say "Create the best-looking plot". Among other things you may choose to do, remember to label the axes using real English, provide a title, subtitle. You may want to pick a theme and color scheme that you like and keep that constant throughout this lab. The default is fine if you are running short of time.

Load up the `GSSvocab` dataset in package `carData` as `X` and drop all observations with missing measurements. Briefly summarize the documentation on this dataset. What is the data type of each variable? What is the response variable?

```
pacman::p_load(carData,skimr)

X = GSSvocab

# There is 1785 NA's
table(is.na(X))
```

```
##
##  FALSE   TRUE
## 229151   1785
```

```r
X = na.omit(X)

# No more NA's
table(is.na(X))
```

```
##
##  FALSE
## 218880
```

```r
#X

# There are 5 factor variables year,gender,nativeBorn, ageGroup, educGroup and 3 numeric vocav,age,educ
skim(X)
```

Table 1: Data summary

Name	X
Number of rows	27360
Number of columns	8
Column type frequency:	
factor	5
numeric	3
Group variables	None

Variable type: factor

skim_variable	n_missing	complete_rate	ordered	n_unique	top_counts
year	0	1	FALSE	20	201: 1856, 199: 1855, 199: 1842, 198: 1714
gender	0	1	FALSE	2	fem: 15512, mal: 11848
nativeBorn	0	1	FALSE	2	yes: 25018, no: 2342
ageGroup	0	1	FALSE	5	60+: 6497, 30-: 6024, 18-: 5691, 40-: 5035
educGroup	0	1	FALSE	5	12 : 8259, 13-: 6942, <12: 5264, 16 : 3814

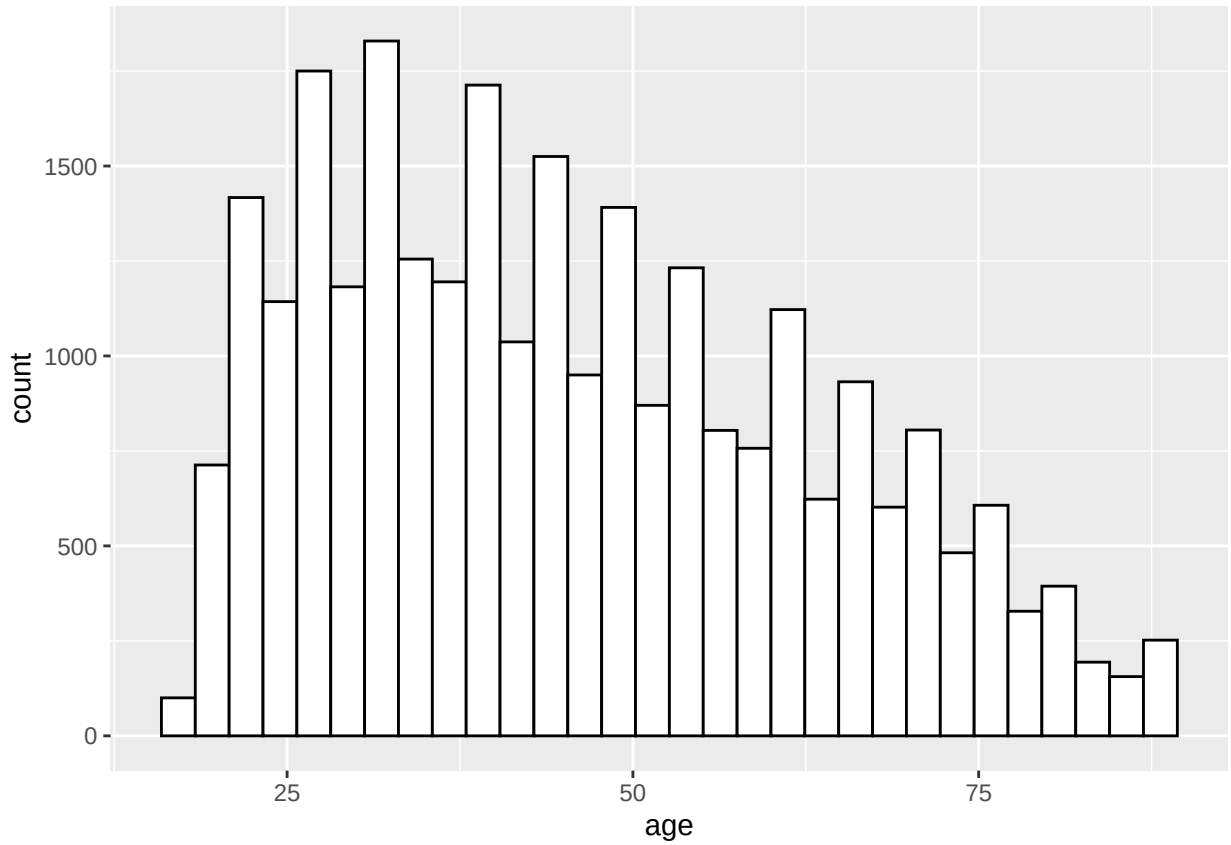
Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100	hist
vocab	0	1	6.00	2.10	0	5	6	7	10	
age	0	1	45.75	17.39	18	31	43	59	89	
educ	0	1	13.16	3.01	0	12	13	16	20	

Create two different plots and identify the best-looking plot you can to examine the `age` variable. Save the best looking plot as an appropriately-named PDF.

```r
ggplot(X, aes(x=age)) + geom_histogram(color="black", fill="white")
```

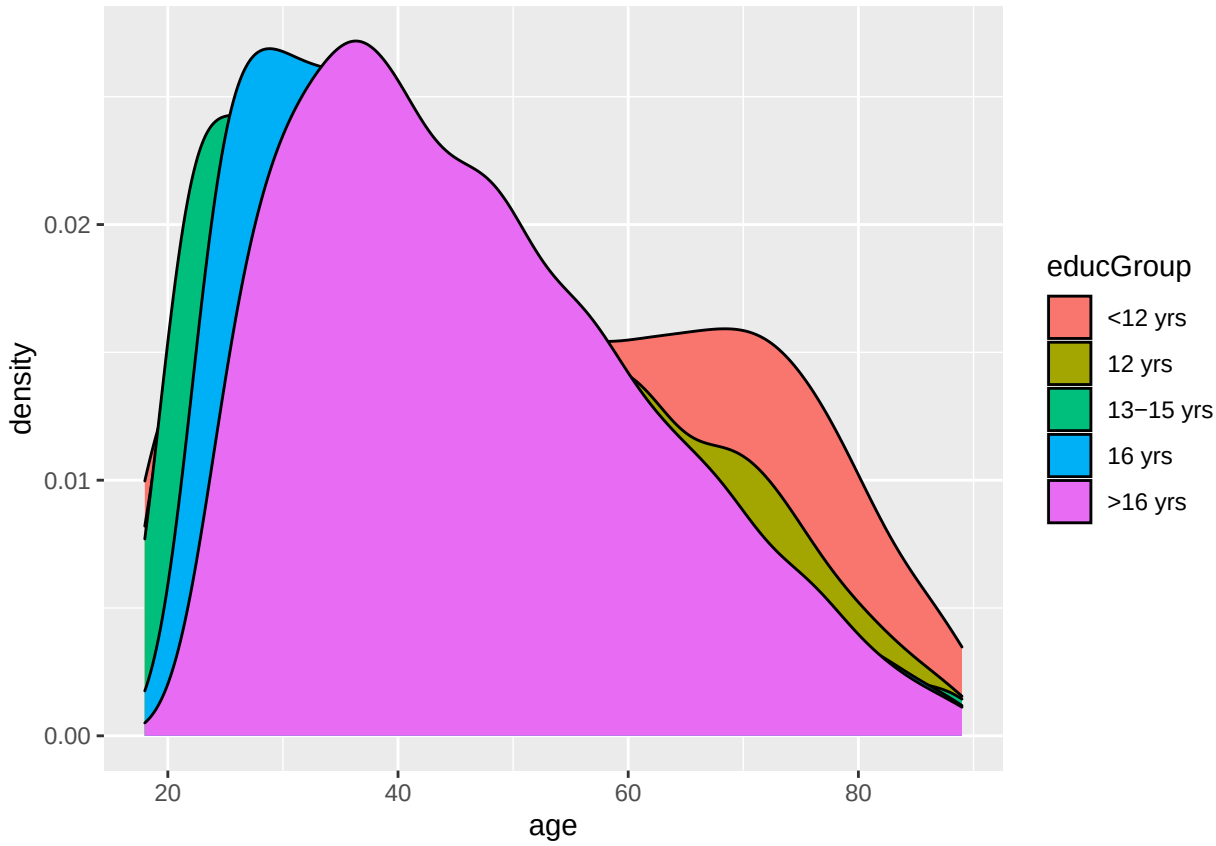
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```




```
ggsave("age_histogram.pdf")
```

```
## Saving 6.5 x 4.5 in image
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

```
ggplot(X, aes(age, fill = educGroup)) + geom_density()
```



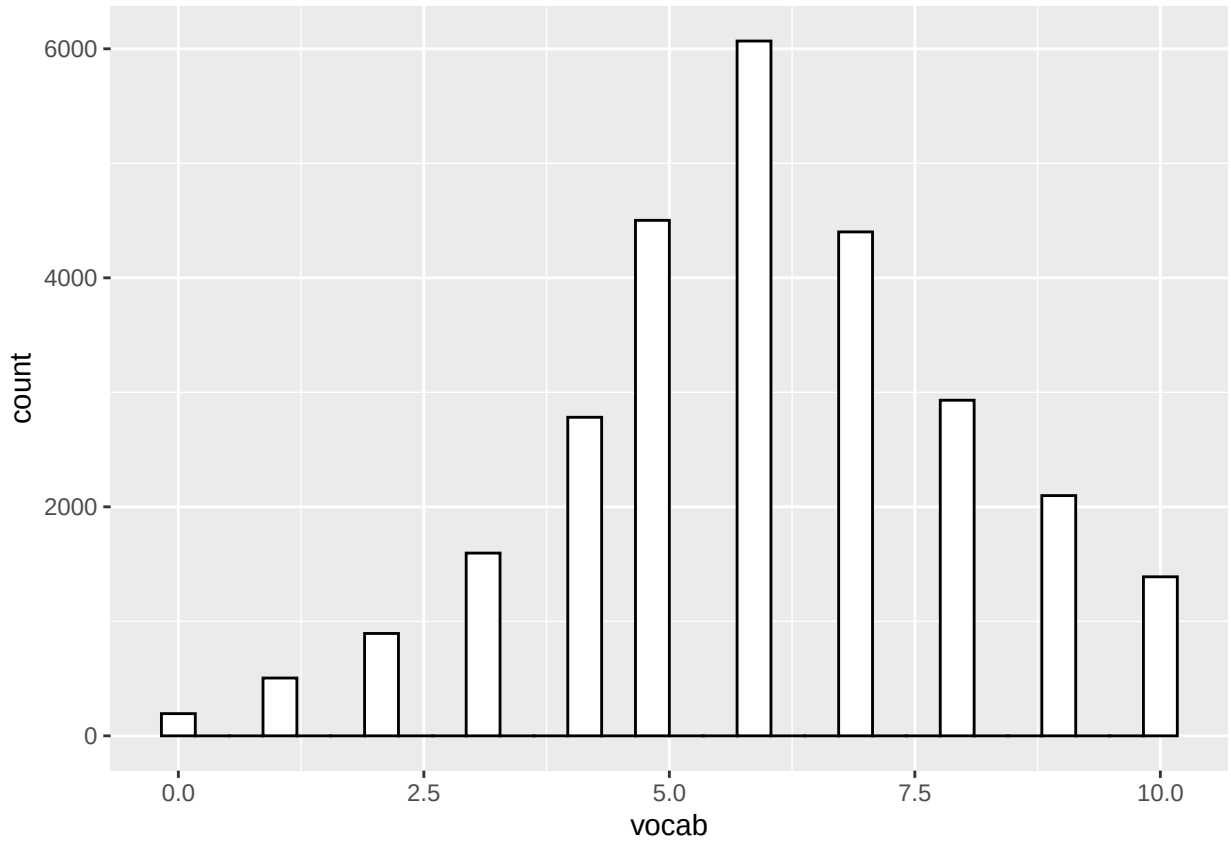

```
ggsave("age_density.pdf")
```

```
## Saving 6.5 x 4.5 in image
```

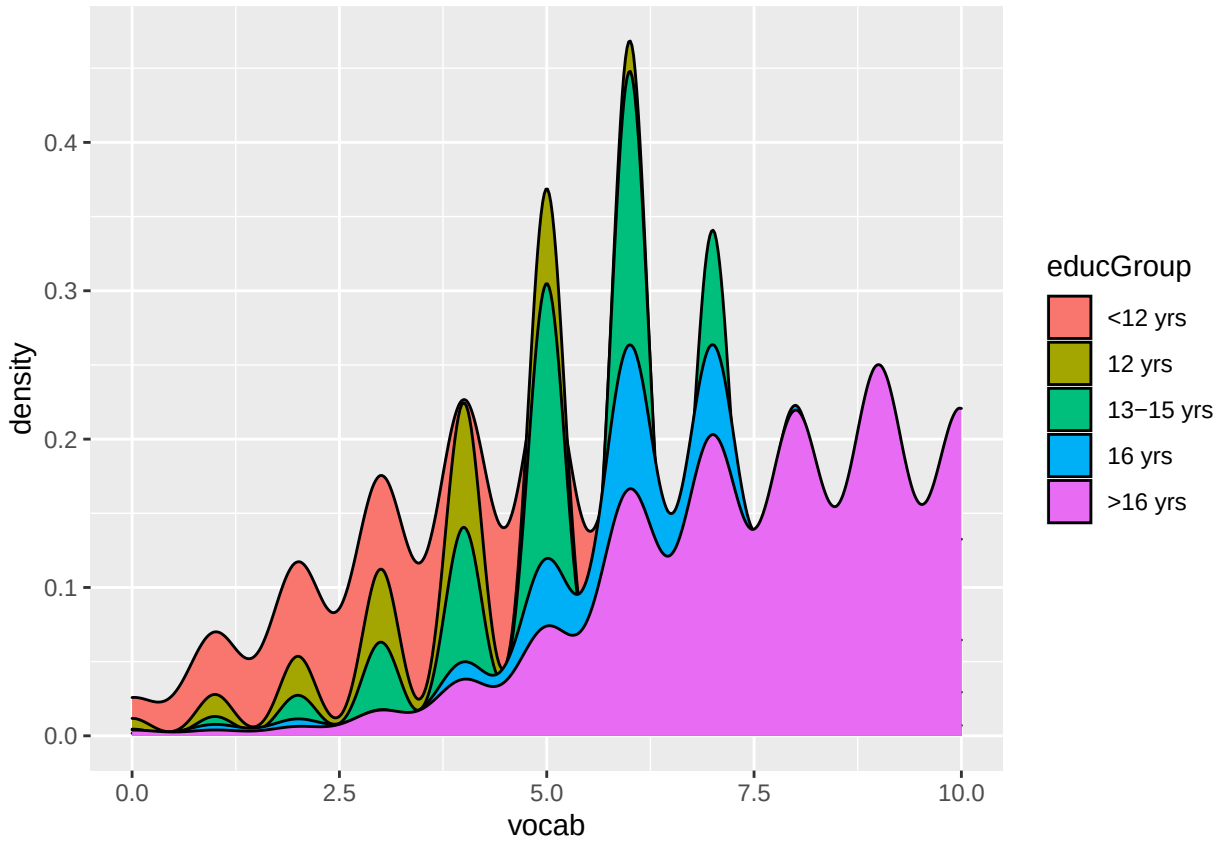
Create two different plots and identify the best looking plot you can to examine the `vocab` variable. Save the best looking plot as an appropriately-named PDF.

```
ggplot(X, aes(x=vocab)) + geom_histogram(color="black", fill="white")
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```




```
ggplot(X, aes(x=vocab,fill = educGroup)) + geom_density()
```

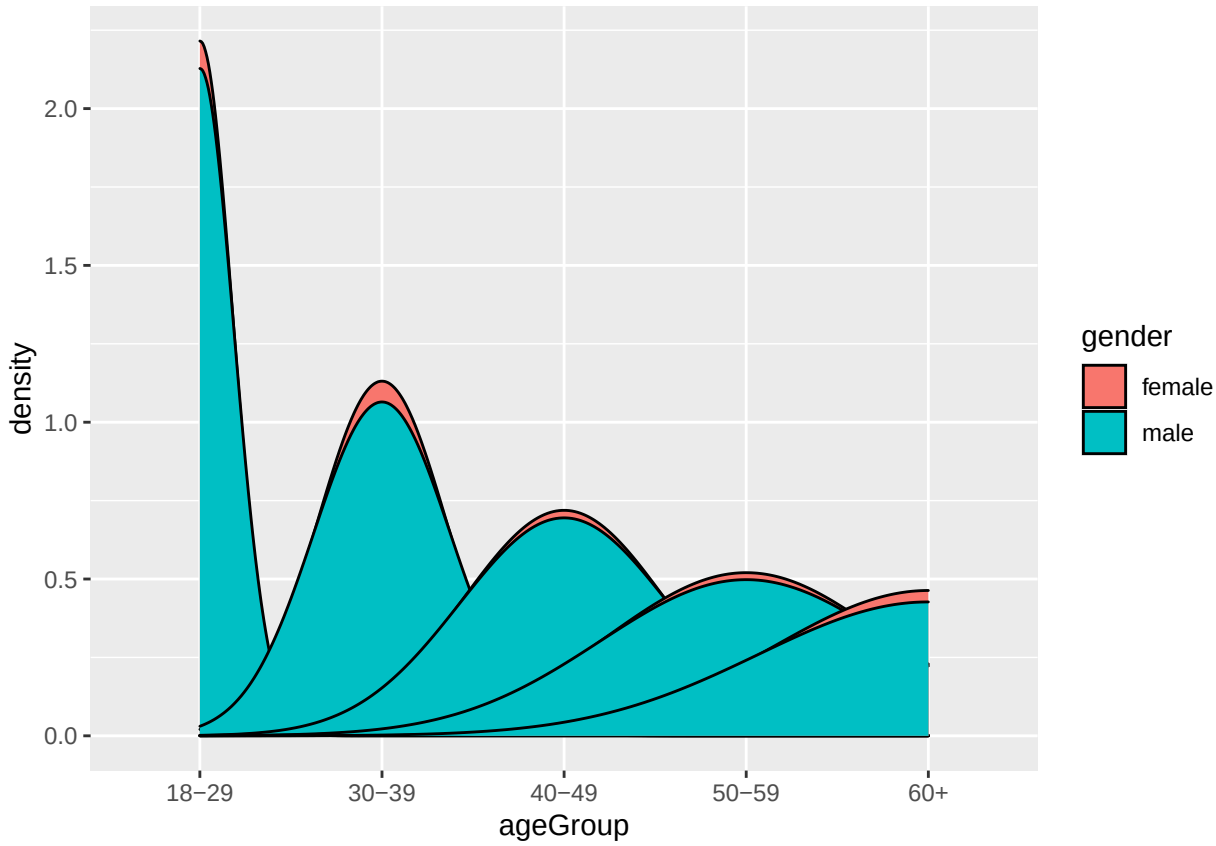



```
ggsave("vocab_density.pdf")
```

```
## Saving 6.5 x 4.5 in image
```

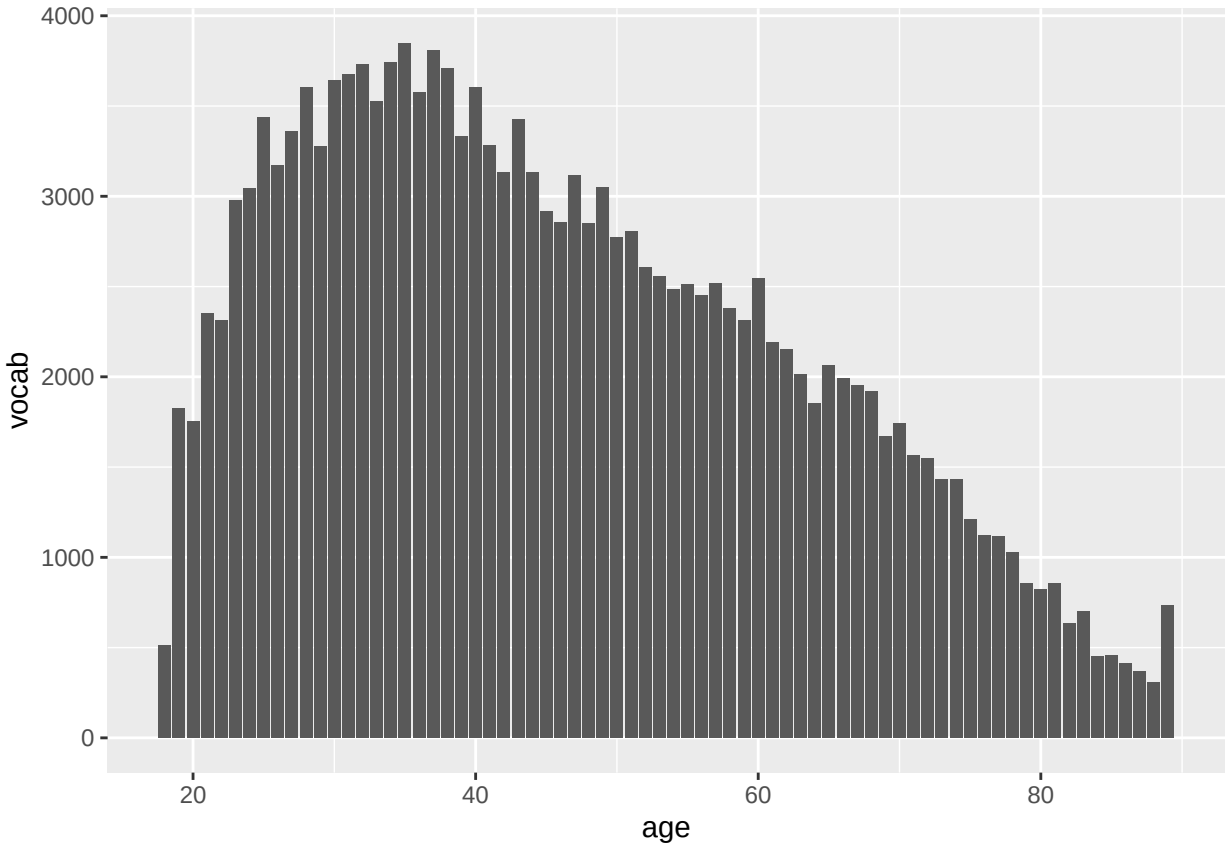
Create the best-looking plot you can to examine the `ageGroup` variable by `gender`. Does there appear to be an association? There are many ways to do this.

```
ggplot(X, aes(x=ageGroup,fill = gender)) + geom_density()
```



Create the best-looking plot you can to examine the `vocab` variable by `age`. Does there appear to be an association?

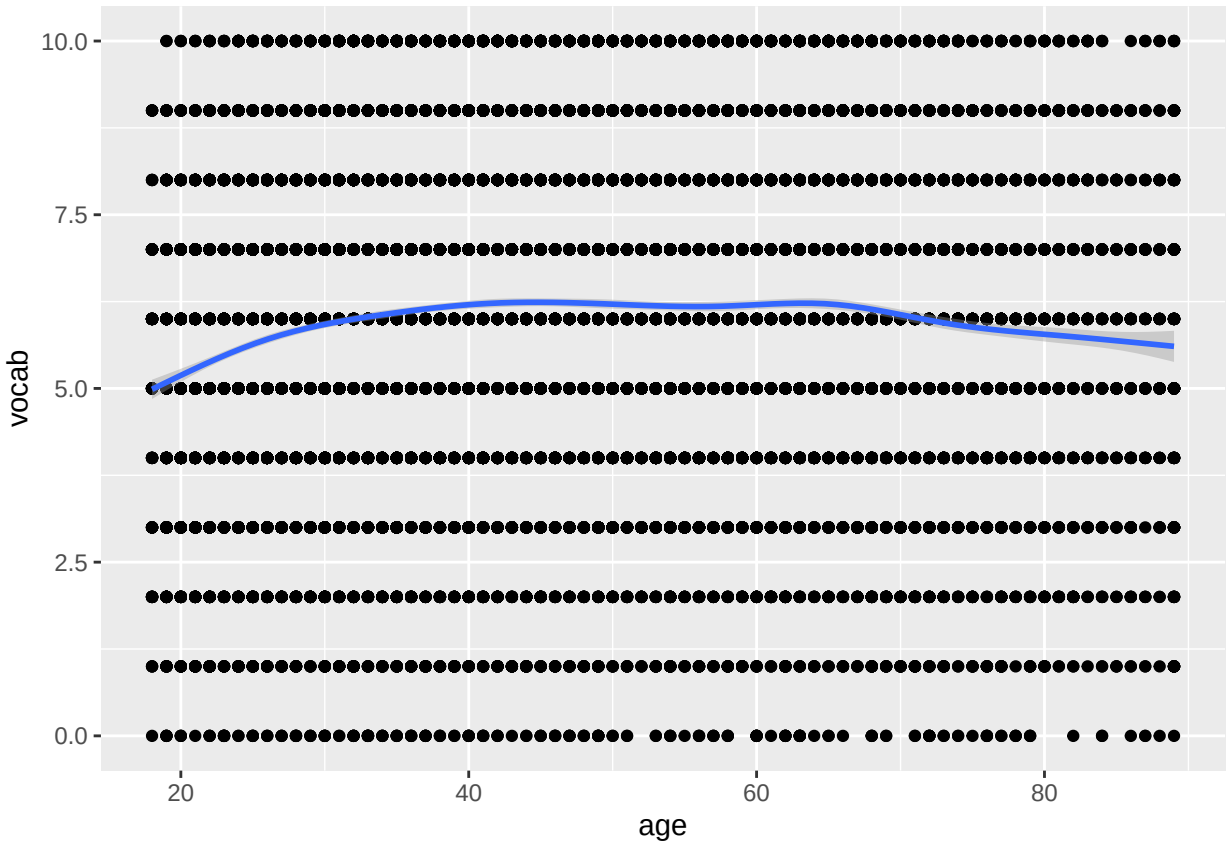
```
# As age increased vocab decreases
ggplot(X, aes(x=age,y = vocab)) + geom_bar(stat="identity")
```



Add an estimate of $f(x)$ using the smoothing geometry to the previous plot. Does there appear to be an association now?

```
ggplot(X, aes(age,vocab)) + geom_point() + geom_smooth()
```

```
## `geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'
```

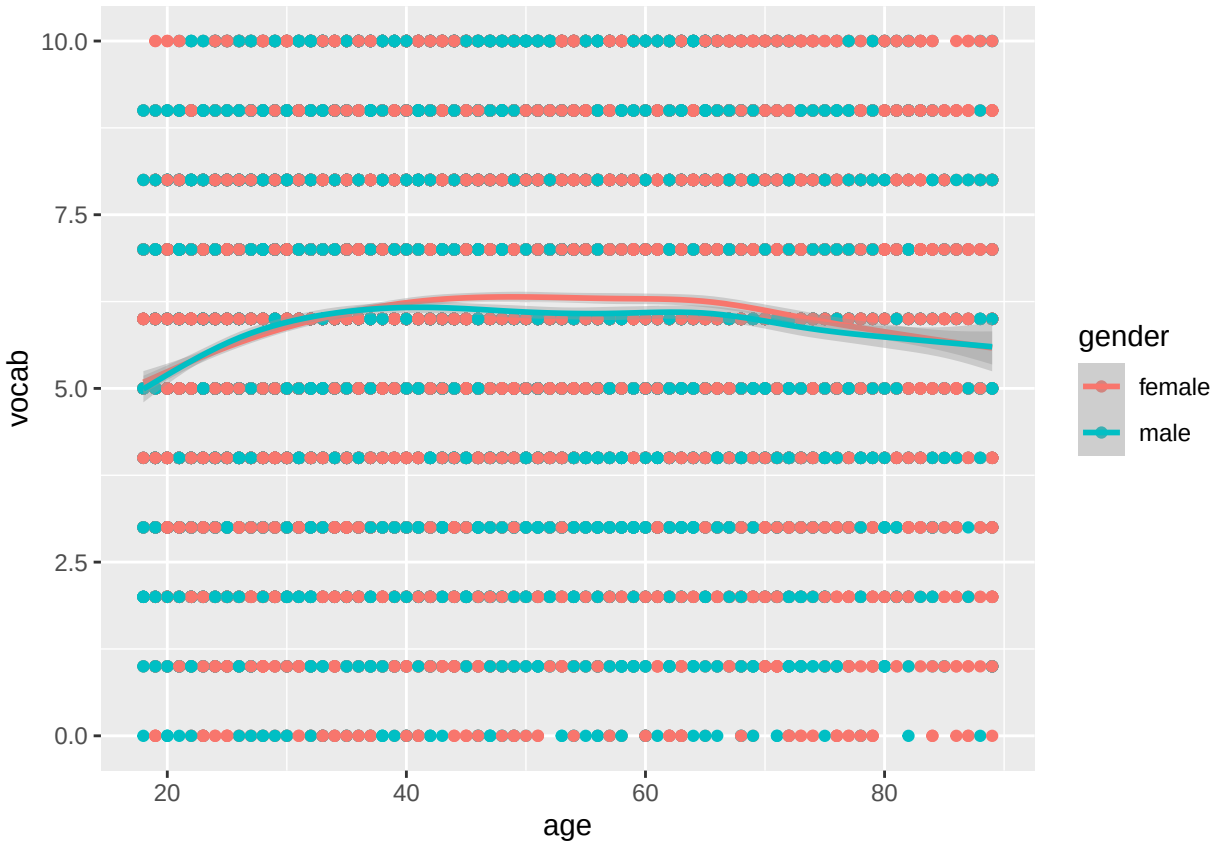


Using the plot from the previous question, create the best looking overloading with variable `gender`. Does there appear to be an interaction of `gender` and `age`?

```
above = ggplot(X, aes(age,vocab)) + geom_point() + geom_smooth()

above + aes(col=gender)
```

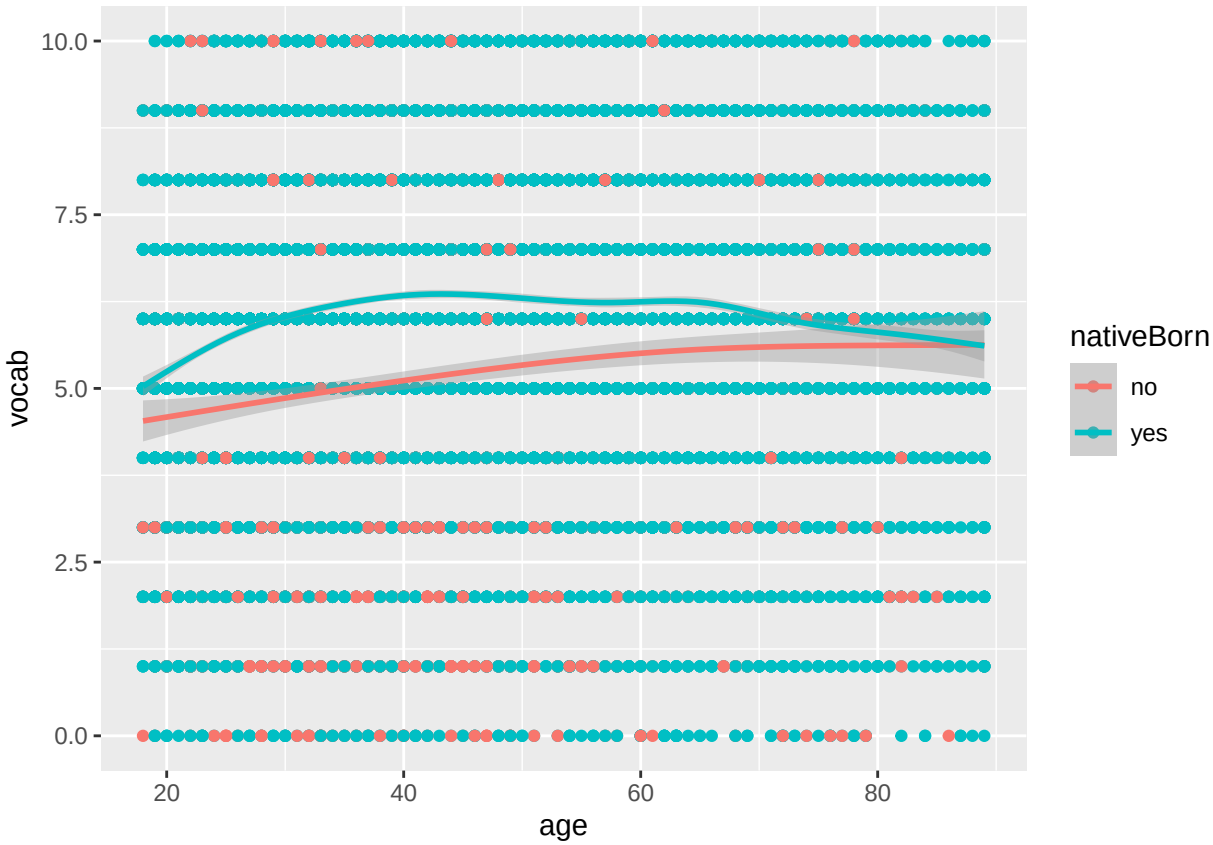
```
## `geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'
```



Using the plot from the previous question, create the best looking overloading with variable `nativeBorn`. Does there appear to be an interaction of `nativeBorn` and `age`?

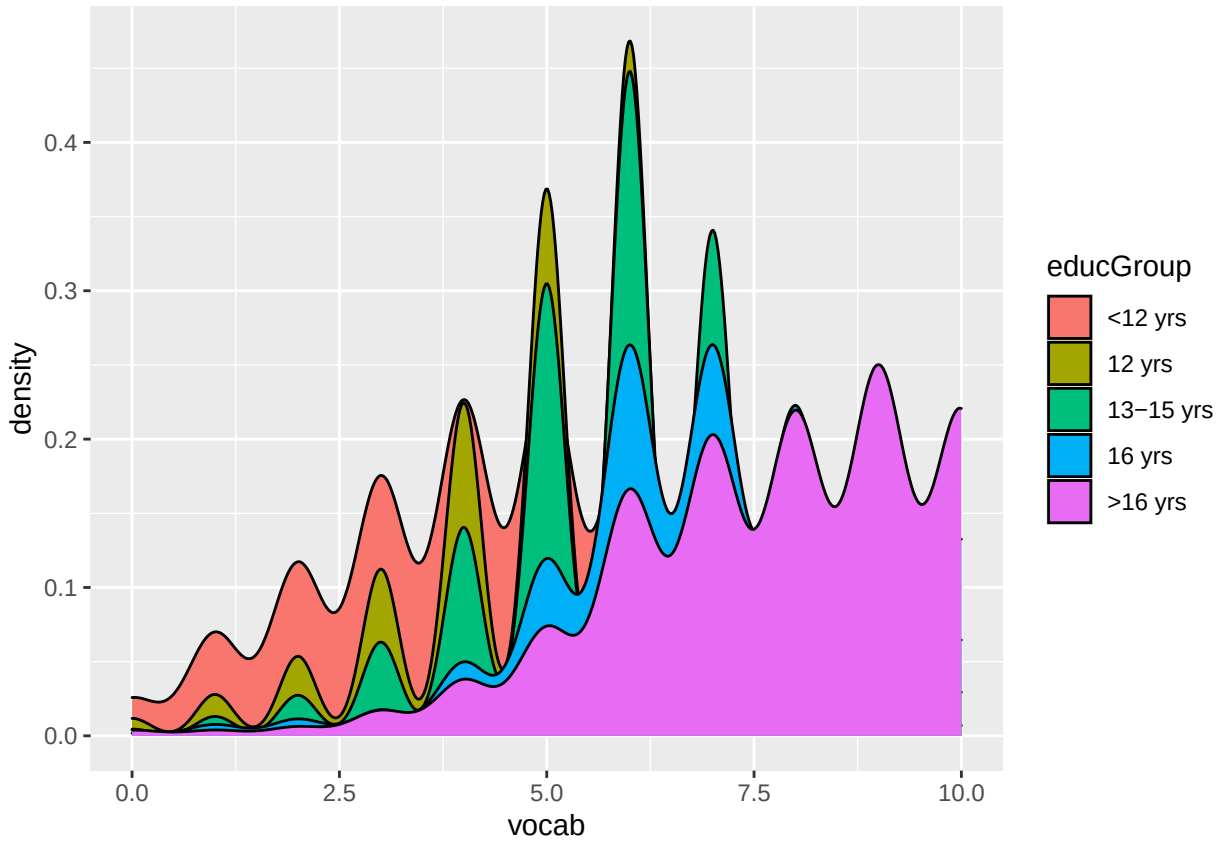
```
above + aes(col=nativeBorn)
```

```
## `geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'
```



Create two different plots and identify the best-looking plot you can to examine the `vocab` variable by `educGroup`. Does there appear to be an association?

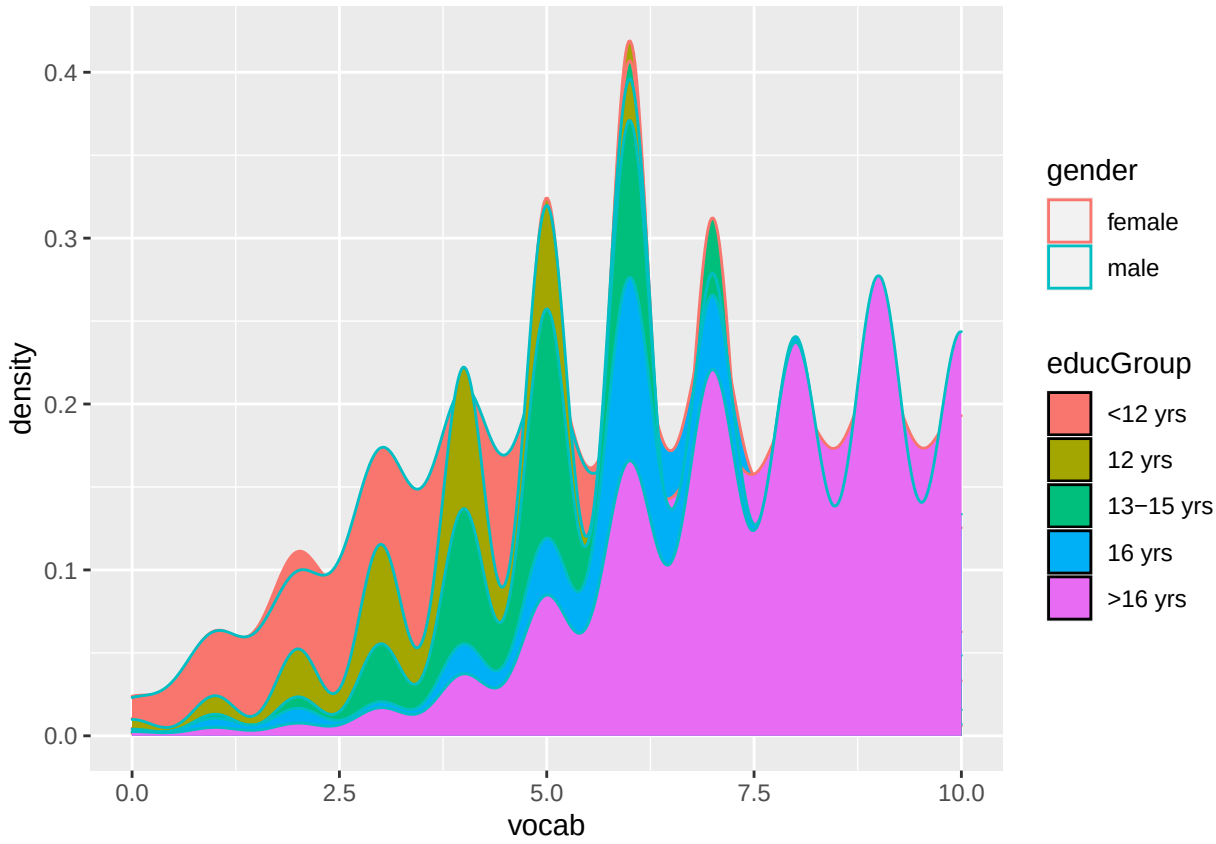
```
ggplot(X, aes(x=vocab,fill = educGroup)) + geom_density()
```



Using the best-looking plot from the previous question, create the best looking overloading with variable `gender`. Does there appear to be an interaction of `gender` and `educGroup`?

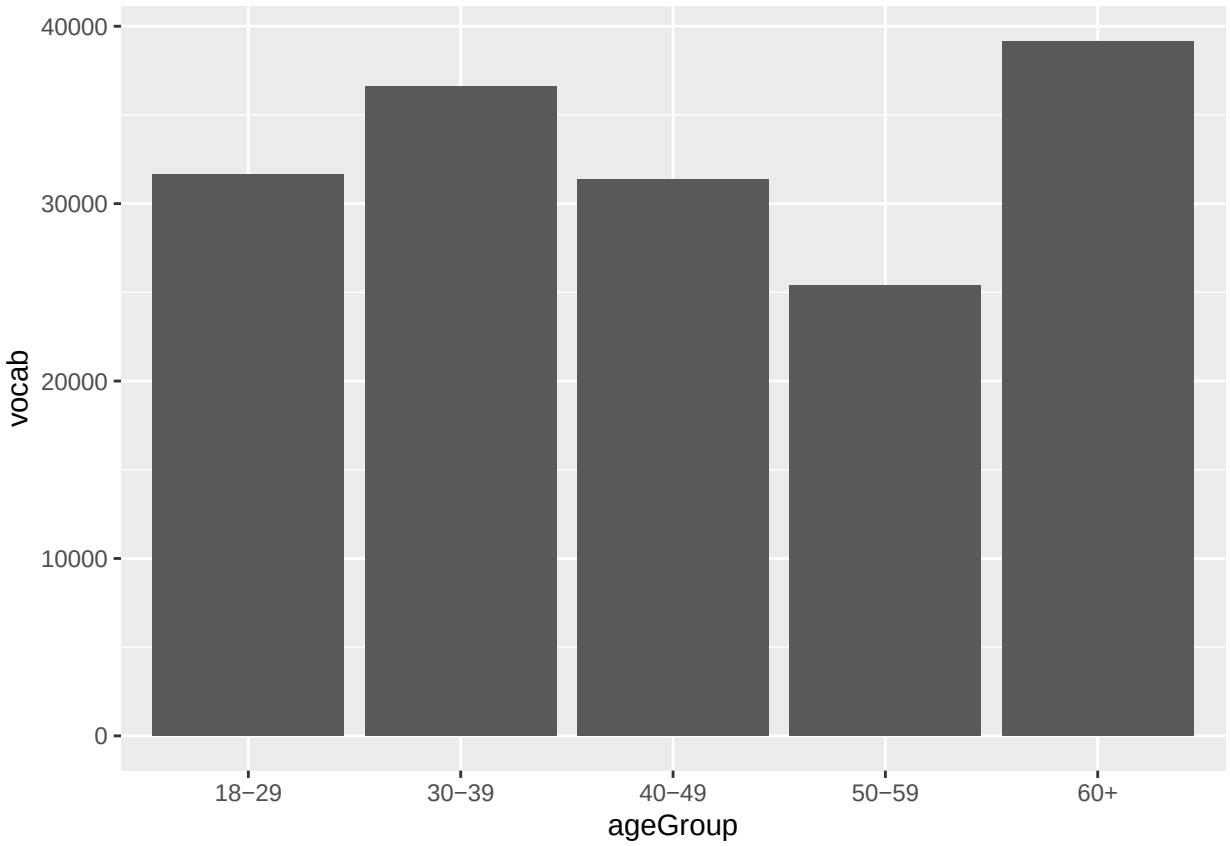
```
above_1 = ggplot(X, aes(x=vocab,fill = educGroup)) + geom_density()

above_1 + aes(col=gender)
```



Using facets, examine the relationship between `vocab` and `ageGroup`. You can drop year level `(Other)`. Are we getting dumber?

```
ggplot(X, aes(x=ageGroup,y = vocab)) + geom_bar(stat="identity")
```




```
# Staying about the same
```

We will now be getting some experience with speeding up R code using C++ via the `Rcpp` package.

First, clear the workspace and load the `Rcpp` package.

```
#TO-DO
```

Create a variable `n` to be 10 and a vaiable `Nvec` to be 100 initially. Create a random vector via `rnorm` `Nvec` times and load it into a `Nvec` x `n` dimensional matrix.

```
#TO-DO
```

Write a function `all_angles` that measures the angle between each of the pairs of vectors. You should measure the vector on a scale of 0 to 180 degrees with negative angles coerced to be positive.

```
#TO-DO
```

Plot the density of these angles.

```
#TO-DO
```

Write an Rcpp function `all_angles_cpp` that does the same thing. Use an IDE if ou want, but write it below in-line.

```
#TO-DO
```

Test the time difference between these functions for `n = 1000` and `Nvec = 100, 500, 1000, 5000`. Store the results in a matrix.

```
#TO-DO
```

Plot the divergence of performance (in log seconds) over n using a line geometry. Use two different colors for the R and CPP functions. Make sure there's a color legend on your plot.

```
#TO-DO
```

Let `Nvec = 10000` and vary `n` to be 10, 100, 1000. Plot the density of angles for all three values of `n` on one plot using color to signify `n`. Make sure you have a color legend. This is not easy.

```
#TO-DO
```

Write an R function `nth_fibonnaci` that finds the nth Fibonnaci number via recursion but allows you to specify the starting number. For instance, if the sequency started at 1, you get the familiar 1, 1, 2, 3, 5, etc. But if it started at 0.01, you would get 0.01, 0.01, 0.02, 0.03, 0.05, etc.

```
#TO-DO
```

Write an Rcpp function `nth_fibonnaci_cpp` that does the same thing. Use an IDE if ou want, but write it below in-line.

```
#TO-DO
```

Time the difference in these functions for n = 100, 200, ..., 1500 while starting the sequence at the smallest possible floating point value in R. Store the results in a matrix.

```
#TO-DO
```

Plot the divergence of performance (in log seconds) over n using a line geometry. Use two different colors for the R and CPP functions. Make sure there's a color legend on your plot.

```
#TO-DO
```