Lab 6

Janine Lim

11:59PM April 15, 2021

#Visualization with the package ggplot2

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.

```
pacman::p_load(carData)
data(GSSvocab)
X = GSSvocab
X = na.omit(GSSvocab)
head(X)
```

```
##        year gender nativeBorn ageGroup educGroup vocab age educ
## 1978.1 1978 female        yes    50-59    12 yrs    10  52   12
## 1978.2 1978 female        yes      60+   <12 yrs     6  74    9
## 1978.3 1978   male        yes    30-39   <12 yrs     4  35   10
## 1978.4 1978 female        yes    50-59    12 yrs     9  50   12
## 1978.5 1978 female        yes    40-49    12 yrs     6  41   12
## 1978.6 1978   male        yes    18-29    12 yrs     6  19   12
```

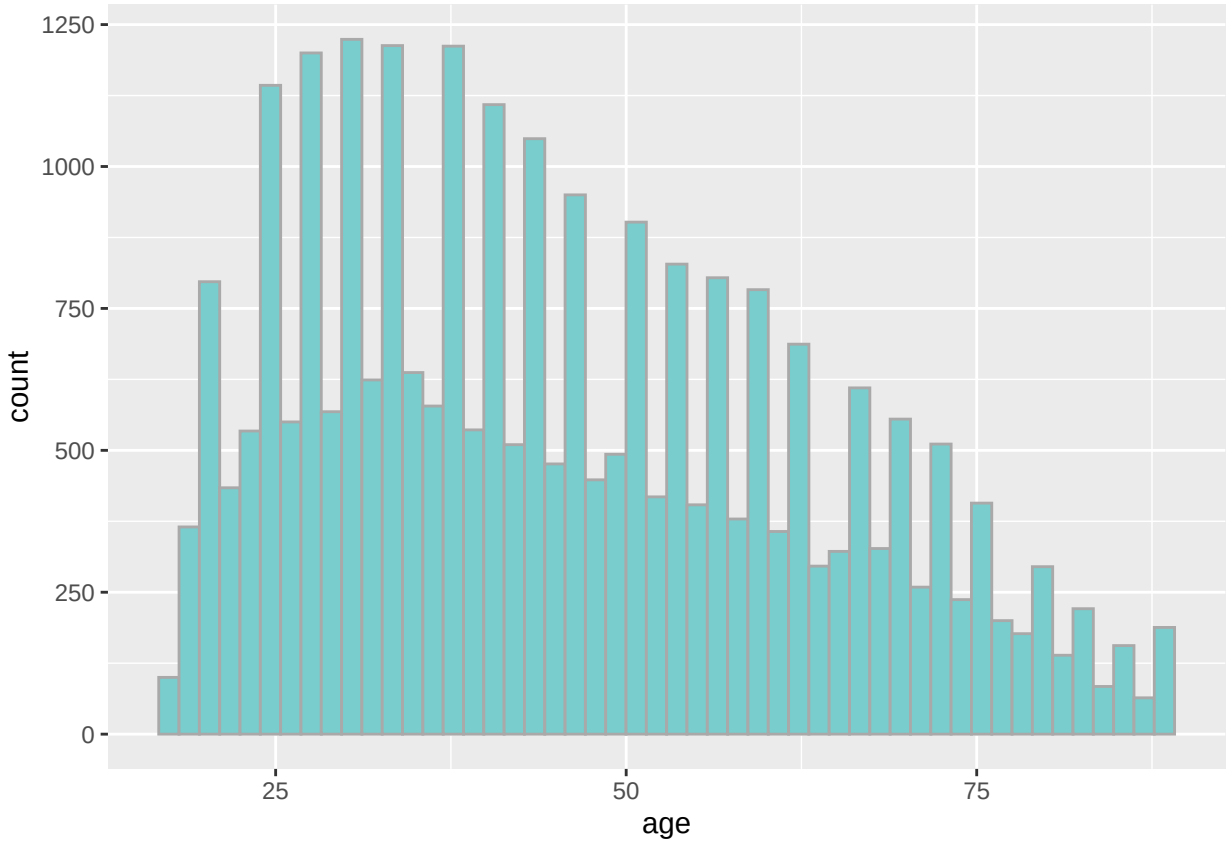
```
?GSSvocab
```

Briefly summarize the documentation on this dataset. What is the data type of each variable? What do you think is the response variable the collectors of this data had in mind?

The dataset has 8 variables with 28, 867 total observations, but now that we removed the NA's, we have 27, 360 observations. The data contains different variables which try to measure for the subject's score on a vocabulary test. The variables are year, which is a factor/categorical variable, gender which a factor/categorical variable, nativeBorn, which is a factor/categorical variable, ageGroup, which is a factor/categorical variable, educGroup, a factor/categorical variable, vocab which is a numeric/continuous variable, age which is a numeric/continuous variable, and educ, a numeric/continuous variable. The response variable would be vocab, the number of words out of 10 correct on a vocabulary test.

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.

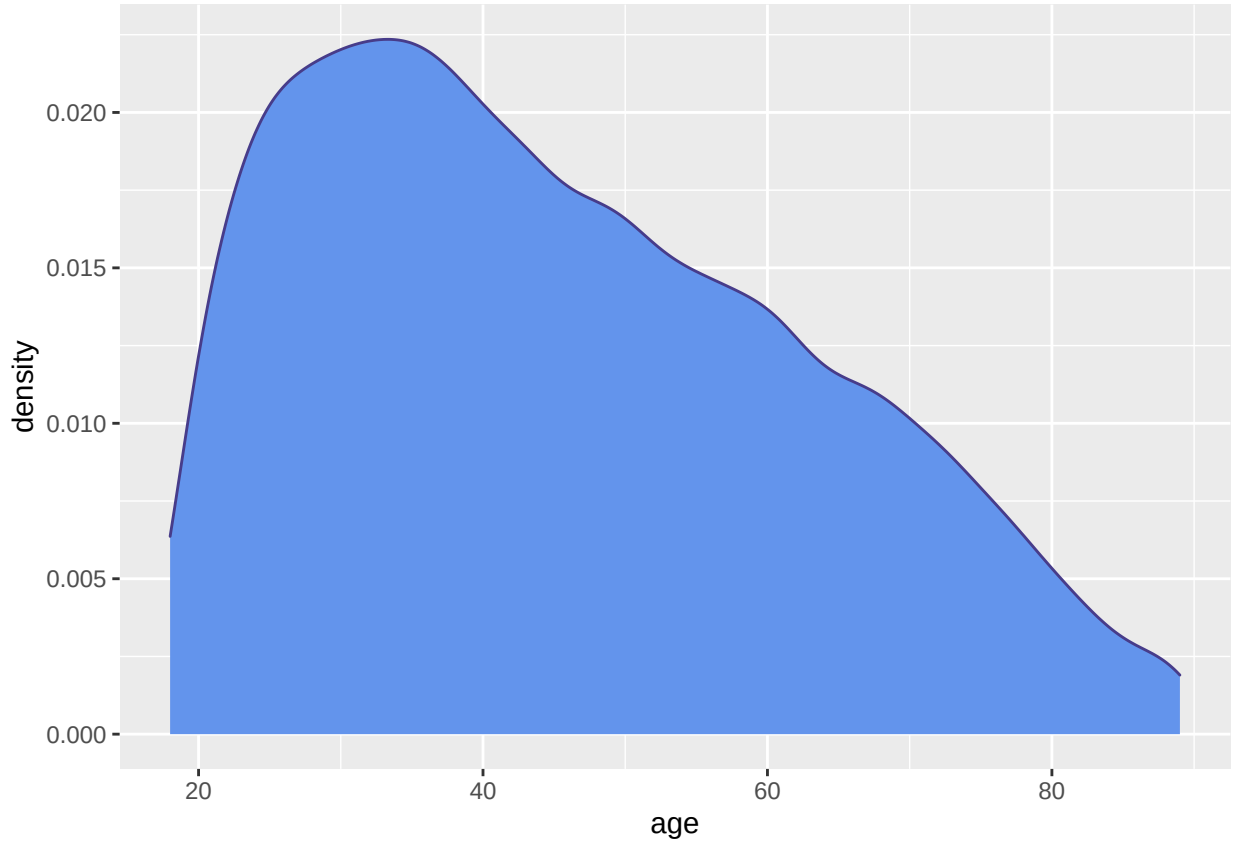
```
pacman::p_load(ggplot2)
ggplot(X) +
  aes(x=age) +
  geom_histogram(bins = 50, fill = "darkslategray3", col = "darkgrey")
```




```
ggsave("plot1.pdf")
```

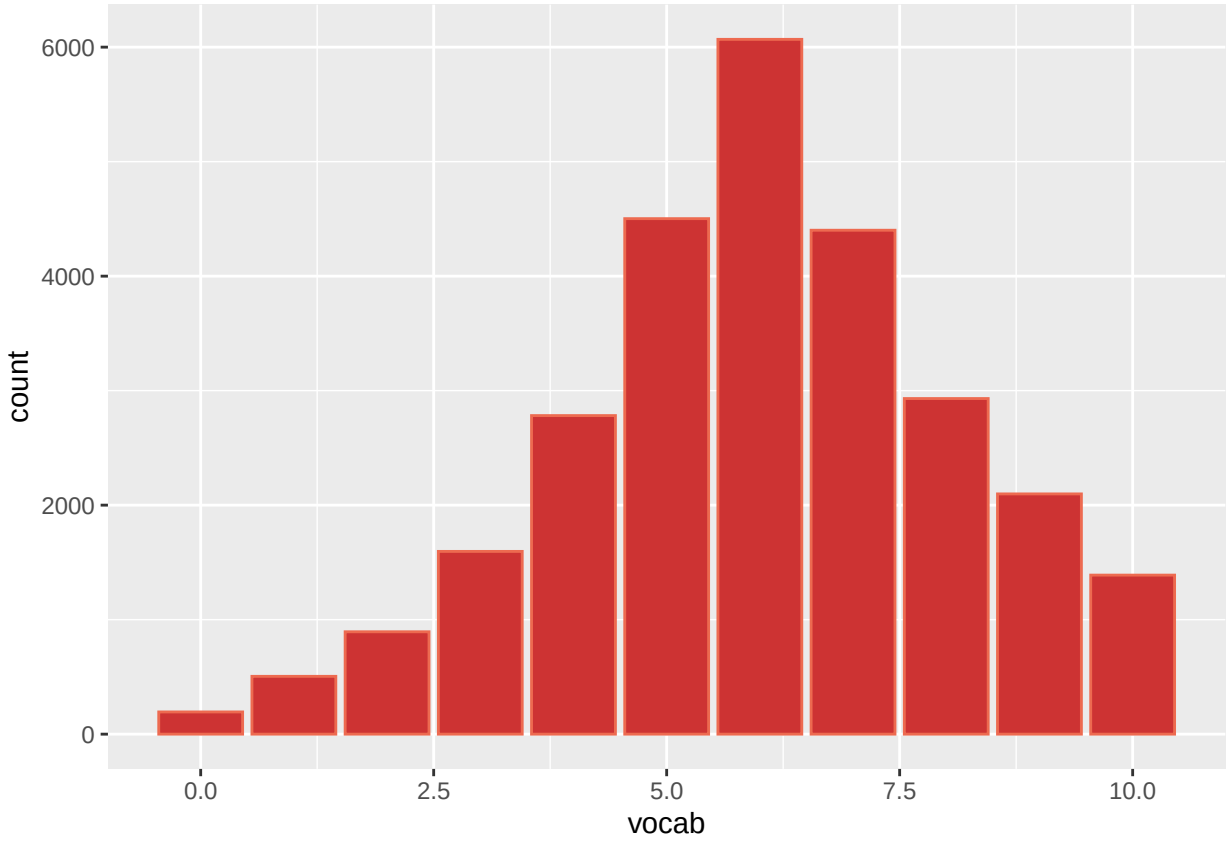
```
## Saving 6.5 x 4.5 in image
```

```
system("open plot1.pdf")

ggplot(X) +
  aes(x = age) +
  geom_density(fill = "cornflowerblue", col = "darkslateblue")
```



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.

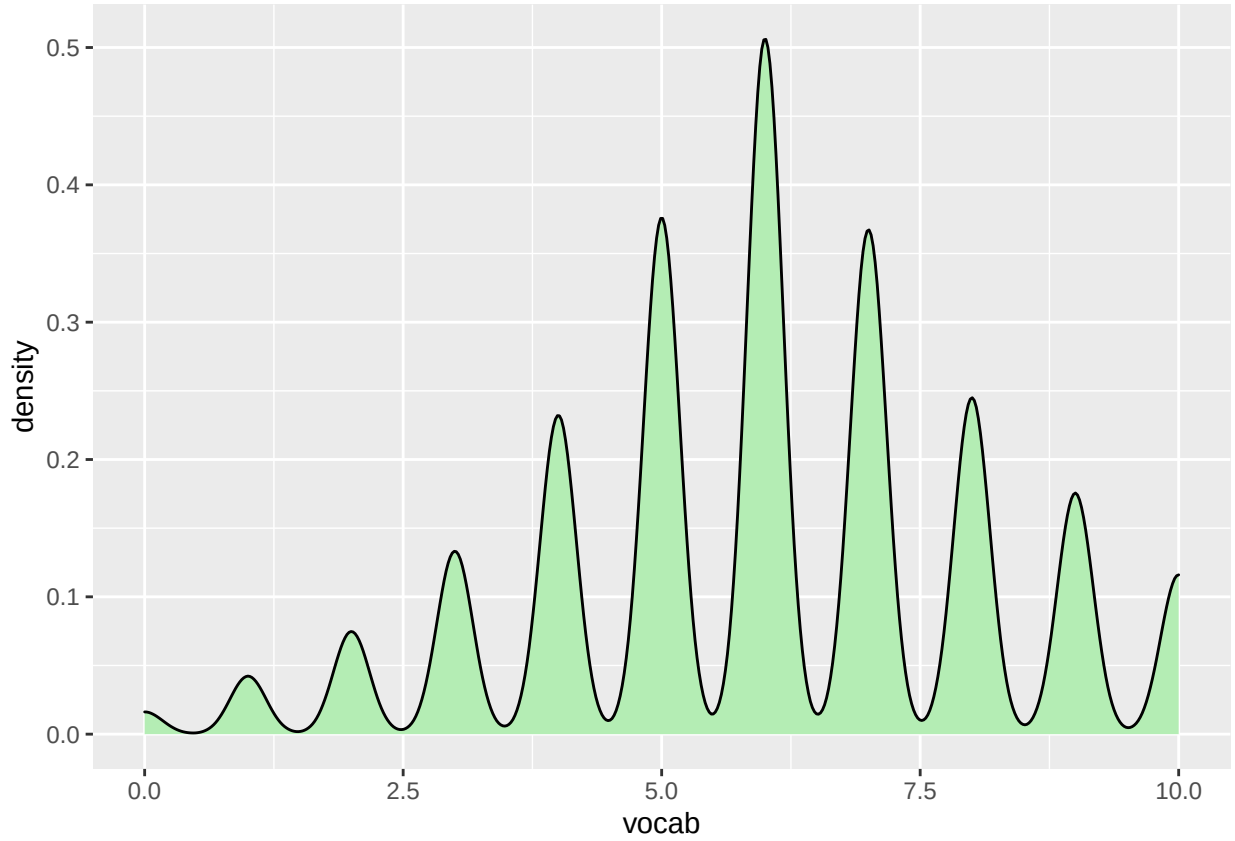
```
ggplot(X) +
  aes(x=vocab) +
  geom_bar(col = "coral2", fill = "brown3")
```




```
ggsave("plot2.pdf")
```

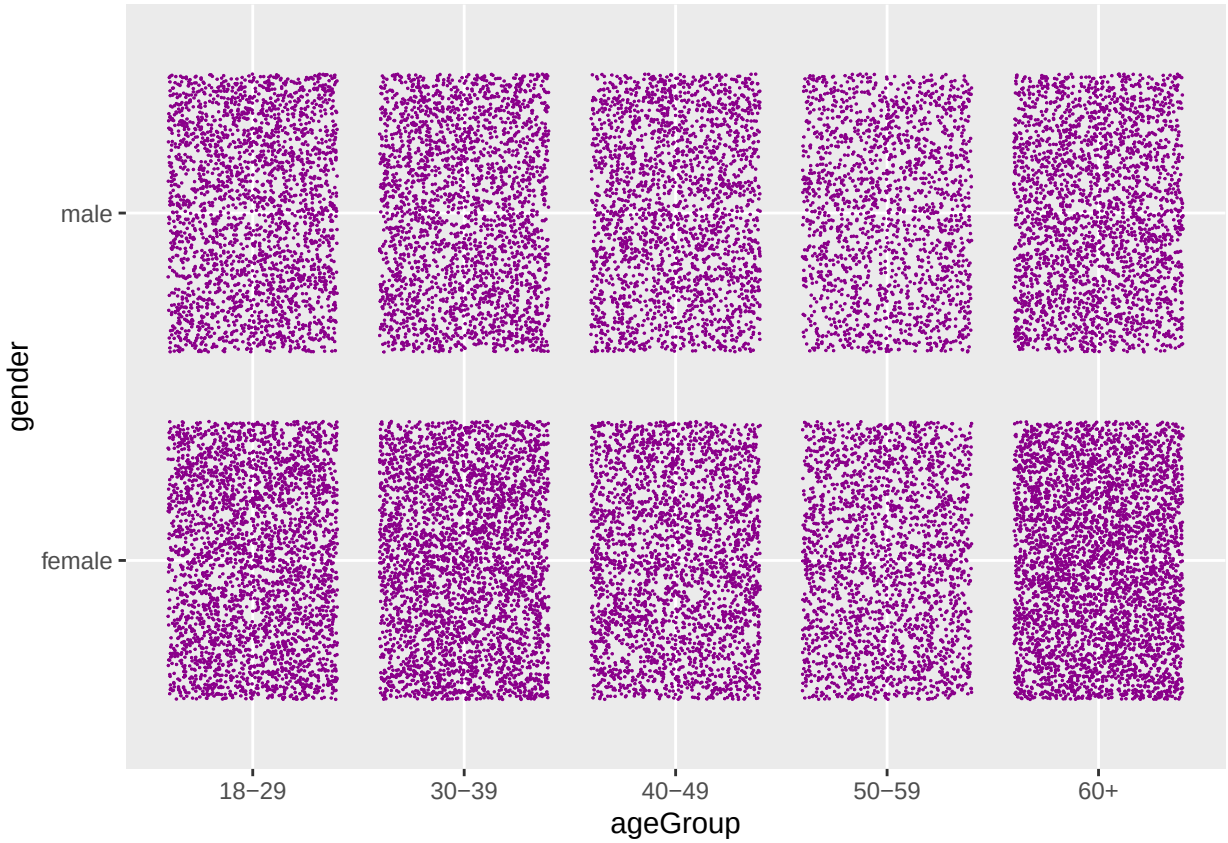
```
## Saving 6.5 x 4.5 in image
```

```
system("open plot2.pdf")

ggplot(X) +
  aes(x=vocab) +
  geom_density(fill = "darkseagreen2")
```



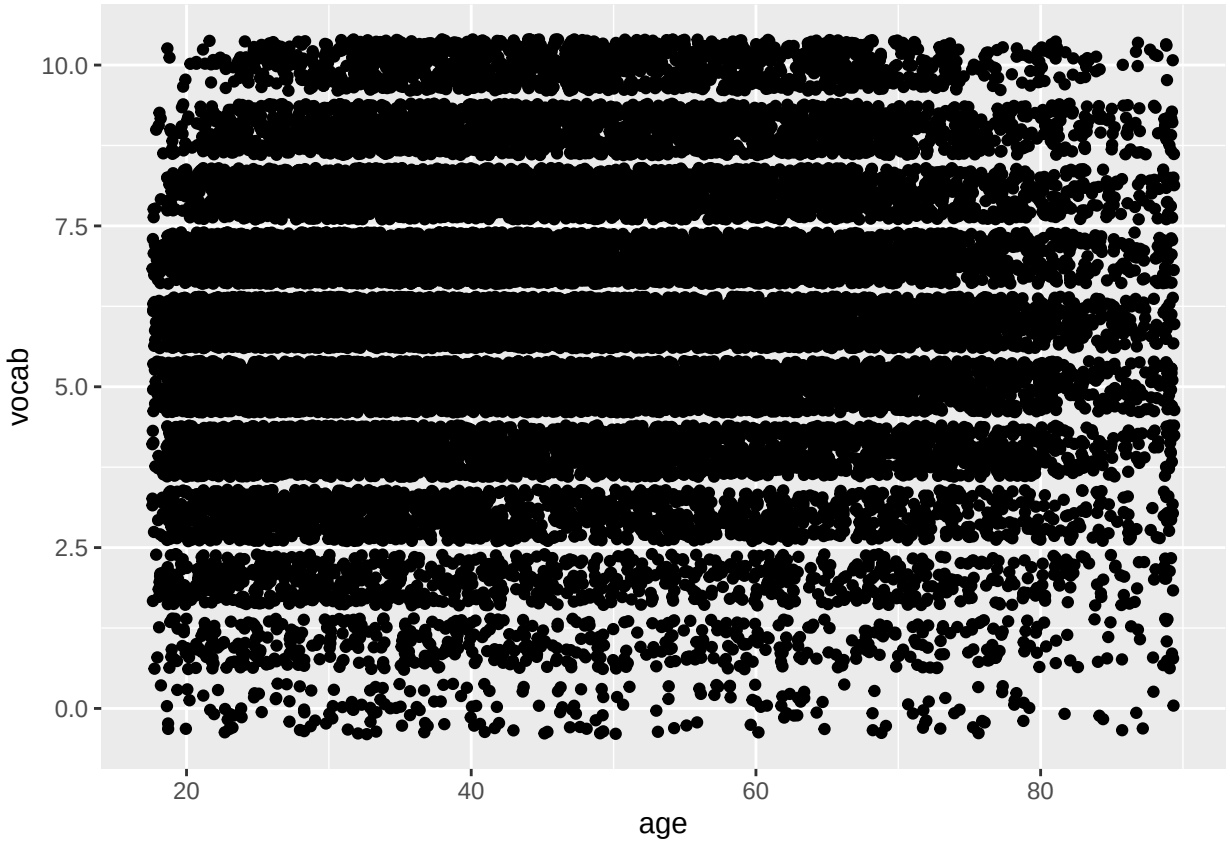
Create the best-looking plot you can to examine the `ageGroup` variable by `gender`. Does there appear to be an association? There are many ways to do this.

```
ggplot(X) +
  aes(x=ageGroup, y = gender) +
  geom_jitter(col = "darkmagenta", size = -.05)
```



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

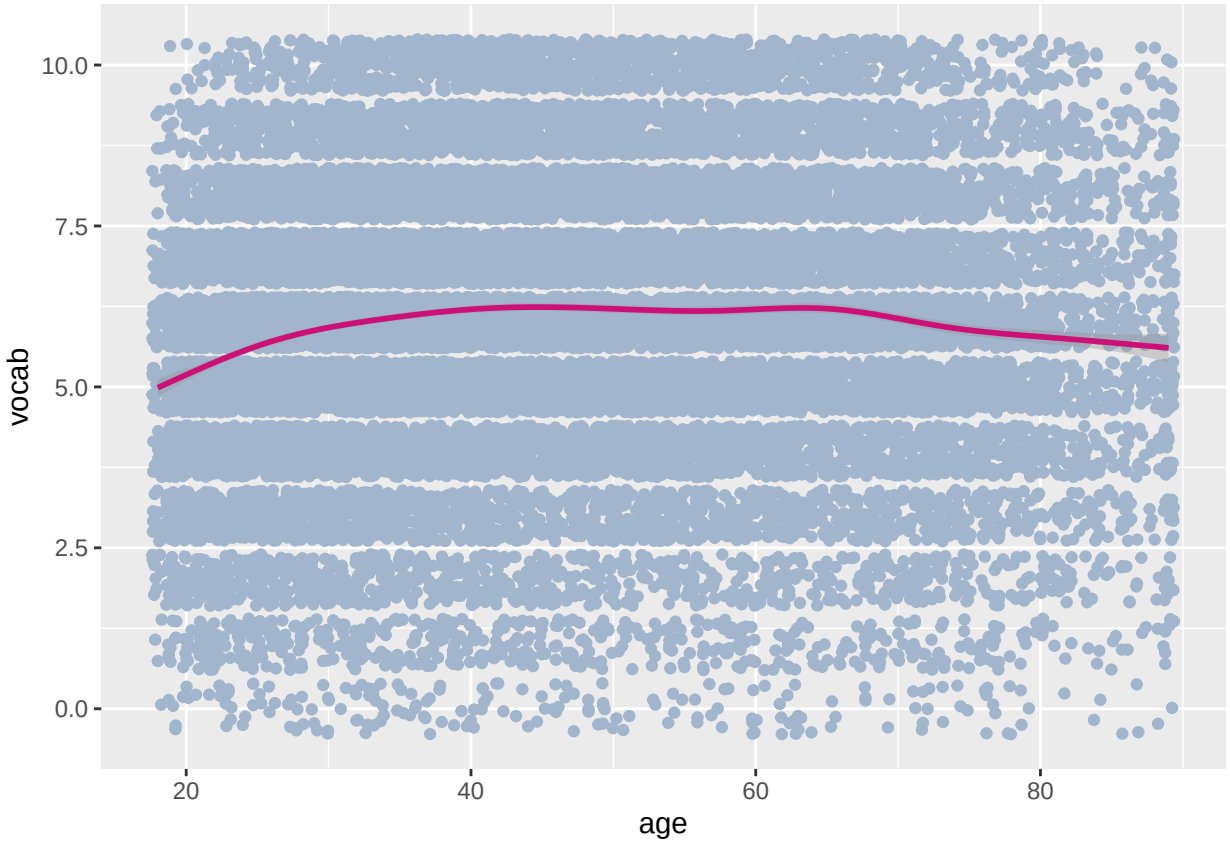
```
ggplot(X) +
  aes(x=age, y = vocab) +
  geom_jitter()
```



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(x=age, y = vocab) +
  geom_jitter(col = "lightsteelblue3") +
  geom_smooth(col = "deeppink3")
```

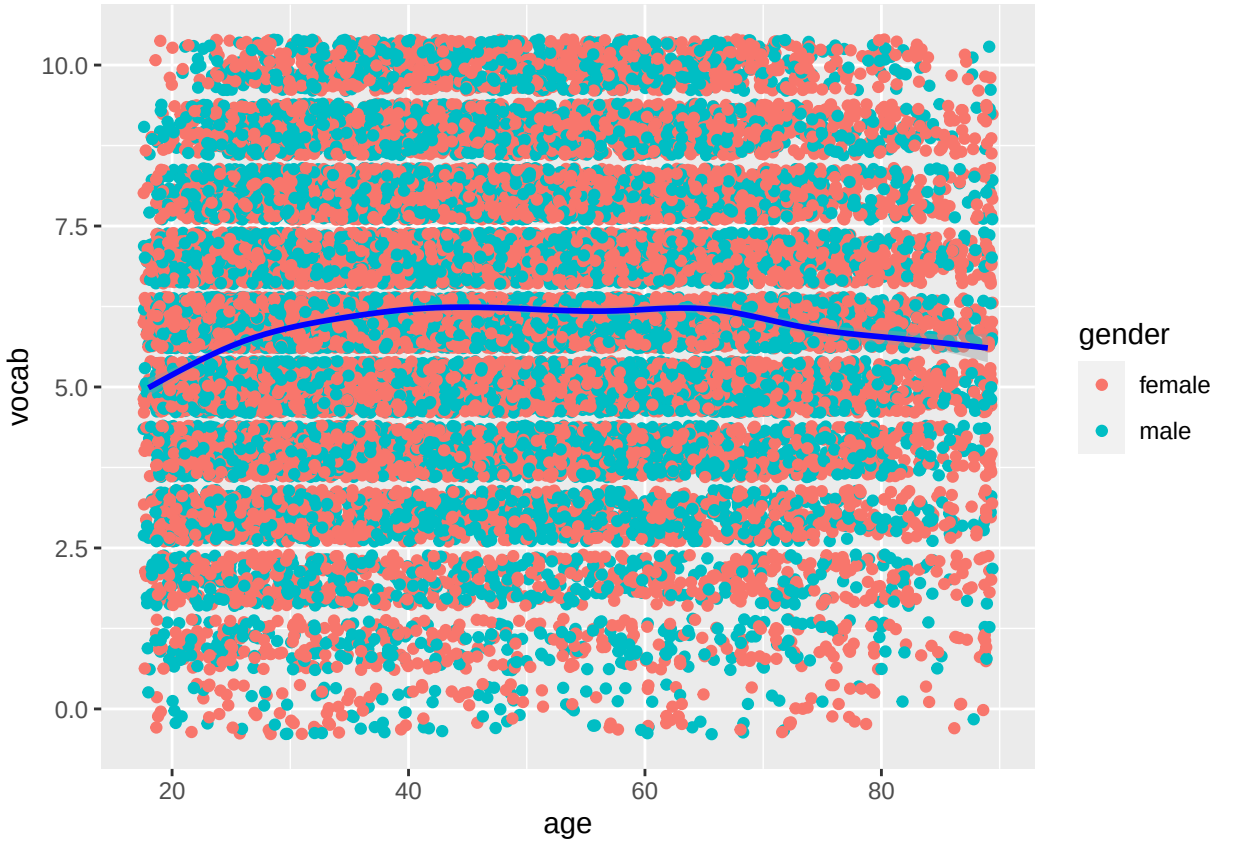
```
## `geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'
```



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

```
ggplot(X) +
  aes(x=age, y = vocab) +
  geom_jitter(aes(col = gender))+
  geom_smooth(col = "blue")
```

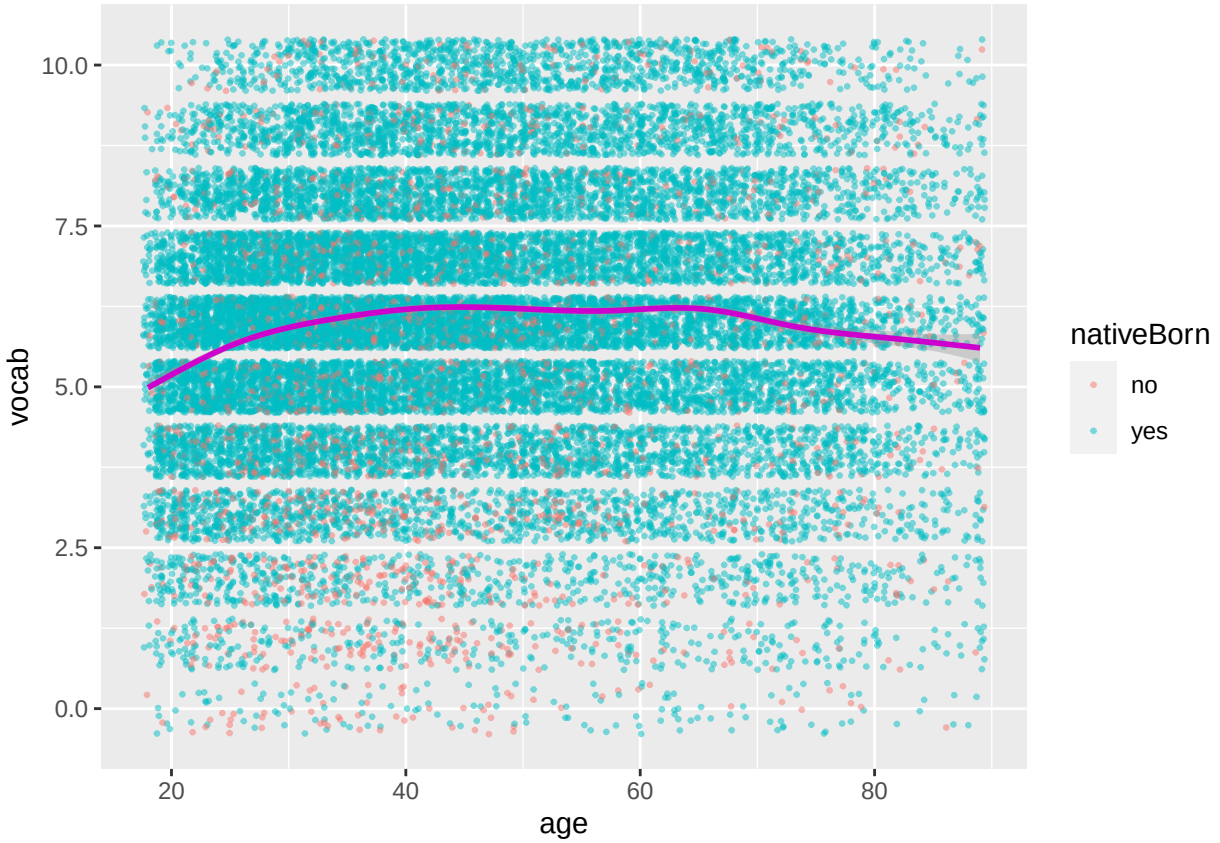
```
## `geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'
```



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

```
ggplot(X) +
  aes(x=age, y = vocab) +
  geom_jitter(aes(col = nativeBorn), size = .5, alpha = 0.5)+
  geom_smooth(col = "magenta3")
```

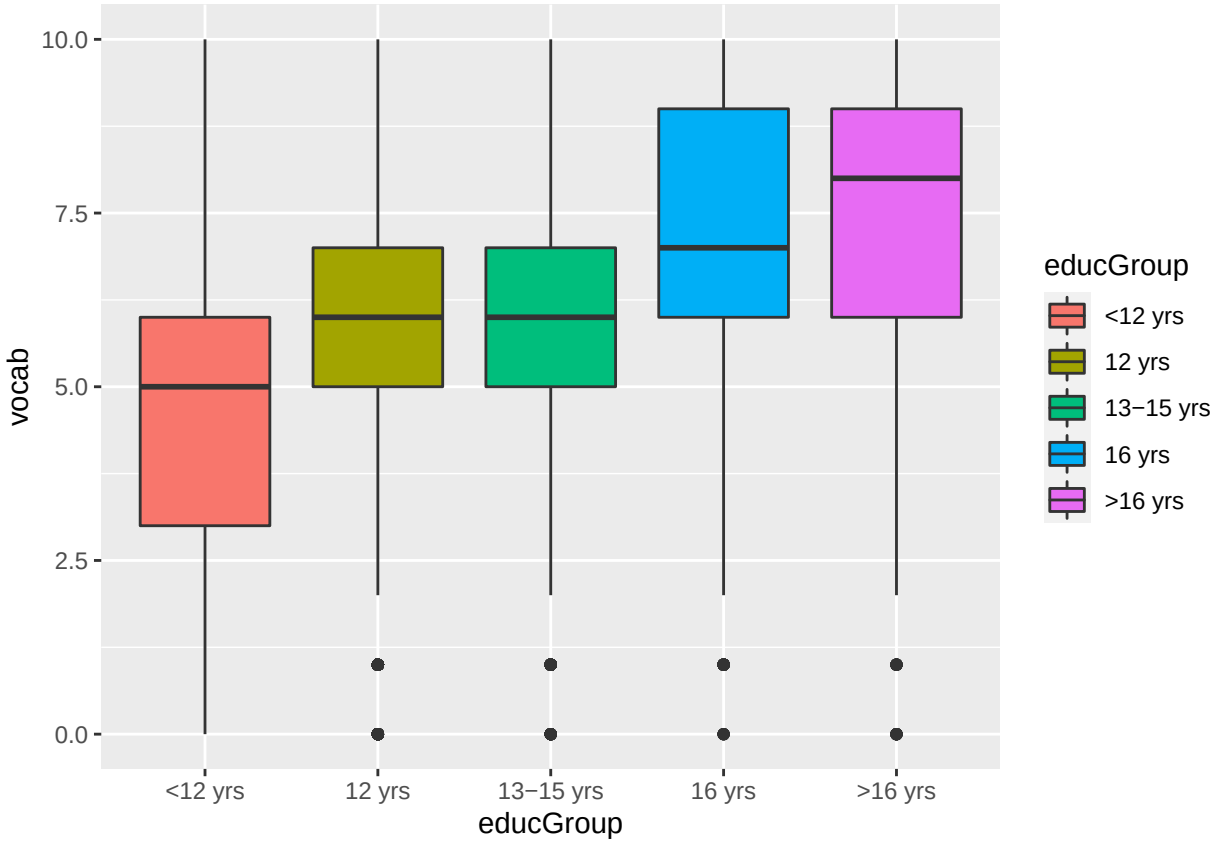
```
## `geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'
```



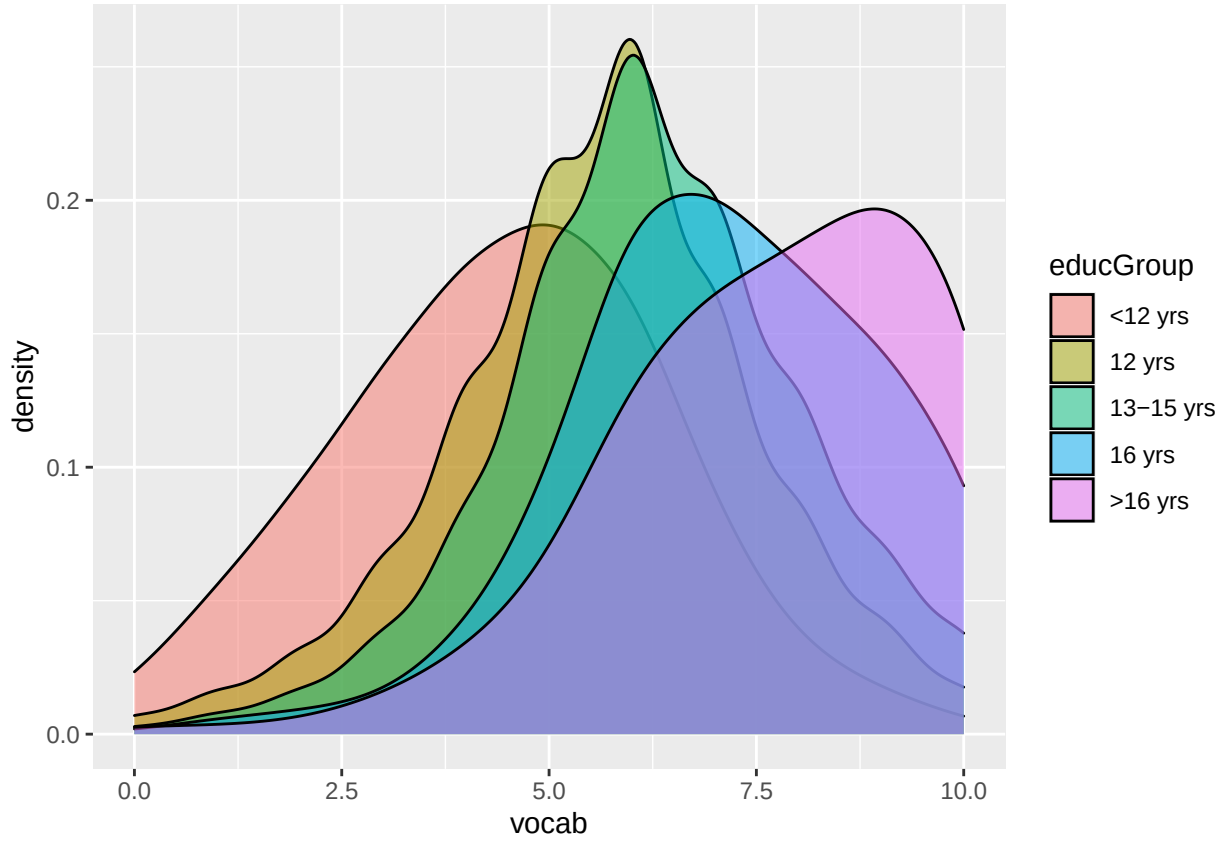
Create two different plots and identify the best-looking plot you can to examine the `vocab` variable by `educGroup`. Does there appear to be an association?

The second plot is the best plot; there appears to be some association where the more years of education results in a higher vocab score.

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

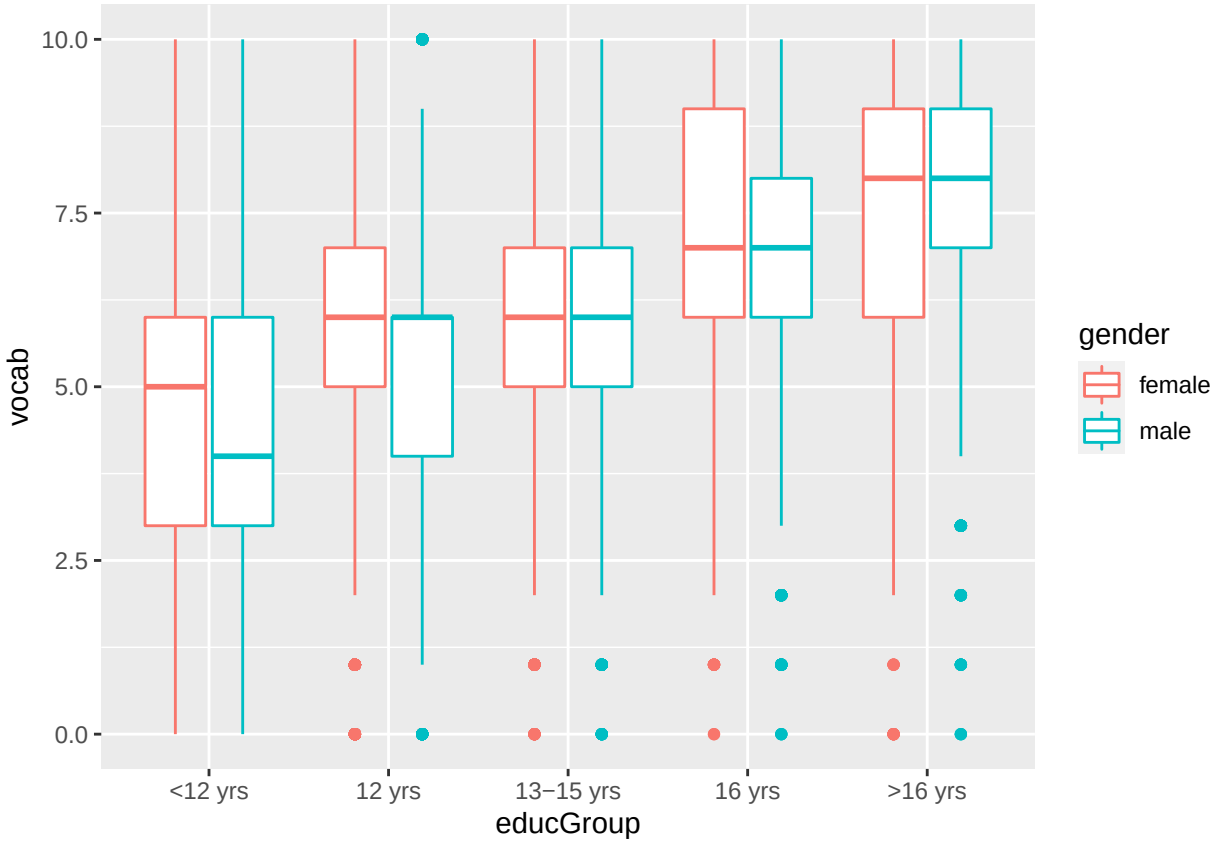



```
ggplot(X) +
  aes(x = vocab) +
  geom_density(aes(fill = educGroup), adjust = 2, alpha = .5)
```



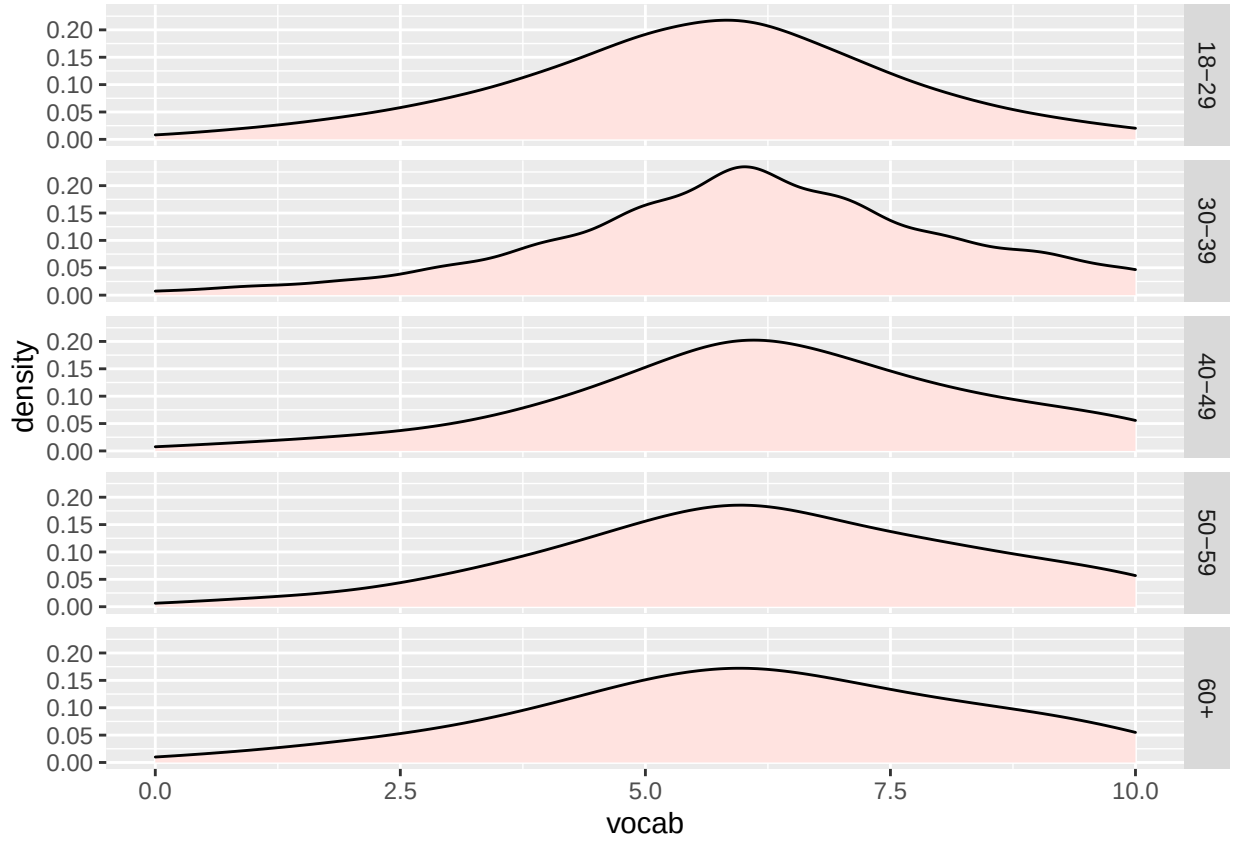
Using the best-looking plot from the previous question, create the best looking overloading with variable `gender`. Does there appear to be an interaction of `gender` and `educGroup`?

```
ggplot(X) +
  aes(x=educGroup, y = vocab) +
  geom_boxplot(aes(col = gender))
```



Using facets, examine the relationship between `vocab` and `ageGroup`. Are we getting dumber?

```
ggplot(X) +
  aes(x = vocab) +
  geom_density(adjust =2, fill = "mistyrose") +
  facet_grid(ageGroup~.)
```



Probability Estimation and Model Selection

Load up the `adult` in the package `ucidata` dataset and remove missingness and the variable `fnlwgt`:

```
pacman::p_load_gh("coatless/ucidata")
data(adult)
adult = na.omit(adult) #kill any observations with missingness
adult$fnlwgt = NULL
```

Cast income to binary where 1 is the `>50K` level.

```
adult$income = ifelse(adult$income == ">50K", 1, 0)
table(adult$income)
```

```
##
##     0     1
## 22653  7508
```

We are going to do some dataset cleanup now. But in every cleanup job, there's always more to clean! So don't expect this cleanup to be perfect.

Firstly, a couple of small things. In variable `marital_status` collapse the levels `Married-AF-spouse` (armed force marriage) and `Married-civ-spouse` (civilian marriage) together into one level called `Married`. Then in variable `education` collapse the levels `1st-4th` and `Preschool` together into a level called `<=4th`.

```
adult$marital_status=as.character(adult$marital_status)
adult$marital_status = ifelse(adult$marital_status == "Married-AF-spouse" | adult$marital_status== "Mar
adult$marital_status = as.factor(adult$marital_status)
table(adult$marital_status)
```

```
##
##              Divorced               Married Married-spouse-absent
##                  4214                 14086                   370
##         Never-married             Separated               Widowed
##                  9725                   939                   827
```

```
adult$education=as.character(adult$education)
adult$education = ifelse(adult$education == "1st-4th" | adult$education== "Preschool", "<=4th", adult$e
adult$education = as.factor(adult$education)
table(adult$education)
```

```
##
##        <=4th          10th          11th          12th      5th-6th      7th-8th
##          196           820          1048           377          288          557
##          9th    Assoc-acdm     Assoc-voc     Bachelors    Doctorate      HS-grad
##          455          1008          1307          5043          375         9840
##      Masters   Prof-school Some-college
##         1627           542          6678
```

Create a model matrix `Xmm` (for this prediction task on just the raw features) and show that it is *not* full rank (i.e. the result of `ncol` is greater than the result of `Matrix::rankMatrix`).

```
Xmm = model.matrix(income ~., adult)
ncol(Xmm)
```

```
## [1] 95
```

```
Matrix::rankMatrix(Xmm)
```

```
## [1] 94
## attr(,"method")
## [1] "tolNorm2"
## attr(,"useGrad")
## [1] FALSE
## attr(,"tol")
## [1] 6.697087e-12
```

```
ncol(Xmm)==Matrix::rankMatrix(Xmm)
```

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

Now tabulate and sort the variable `native_country`.

```
tab = sort(table(adult$native_country))
```

Do you see rare levels in this variable? Explain why this may be a problem.

There are a few rare levels, like Holand-Netherlands, Scotland, Honduras, etc. This may be a problem because it may not make the native country feature useful when there are too many levels since there are so many unique values.

Collapse all levels that have less than 50 observations into a new level called `other`. This is a very common data science trick that will make your life much easier. If you can't hope to model rare levels, just give up and do something practical! I would recommend first casting the variable to type "character" and then do the level reduction and then recasting back to type `factor`. Tabulate and sort the variable `native_country` to make sure you did it right.

```
adult$native_country=as.character(adult$native_country)
adult$native_country = ifelse(adult$native_country %in% names(tab[tab<50]), "other", adult$native_countr
adult$native_country = as.factor(adult$native_country)
sort(table(adult$native_country))
```

```
##
##          Columbia            Poland             Japan         Guatemala
##                56                56                59                63
##           Vietnam Dominican-Republic             China             Italy
##                64                67                68                68
##             South           Jamaica           England              Cuba
##                71                80                86                92
##       El-Salvador             India            Canada       Puerto-Rico
##               100               100               107               109
##           Germany       Philippines             other            Mexico
##               128               188               486               610
##     United-States
##             27503
```

We're still not done getting this data down to full rank. Take a look at the model matrix just for `workclass` and `occupation`. Is it full rank?

This matrix is still not full rank since we see that the number of columns equal 21 but the rank is 20, meaning we have one column that is duplicate information.

```
Xmm_2= model.matrix(income ~ workclass + occupation, adult)
ncol(Xmm_2)
```

```
## [1] 21
```

```
Matrix::rankMatrix(Xmm_2)
```

```
## [1] 20
## attr(,"method")
## [1] "tolNorm2"
## attr(,"useGrad")
## [1] FALSE
## attr(,"tol")
## [1] 6.697087e-12
```

```
ncol(Xmm_2)==Matrix::rankMatrix(Xmm_2)
```

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

These variables are similar and they probably should be interacted anyway eventually. Let's combine them into one factor. Create a character variable named `worktype` that is the result of concatenating `occupation` and `workclass` together with a ":" in between. Use the `paste` function with the `sep` argument (this casts automatically to type `character`). Then tabulate its levels and sort.

```
adult$occupation = as.character(adult$occupation)
adult$workclass = as.character(adult$workclass)
adult$worktype = paste(adult$occupation, adult$workclass, sep = ":")
tab_worktype = sort(table(adult$worktype))
adult$occupation = NULL
adult$workclass = NULL
```

Like the `native_country` exercise, there are a lot of rare levels. Collapse levels with less than 100 observations to type `other` and then cast this variable `worktype` as type `factor`. Recheck the tabulation to ensure you did this correct.

```
adult$worktype=as.character(adult$worktype)
adult$worktype = ifelse(adult$worktype %in% names(tab_worktype[tab_worktype<100]), "other", adult$workty
adult$worktype = as.factor(adult$worktype)
sort(table(adult$worktype))
```

```
##
##        Transport-moving:Local-gov         Protective-serv:State-gov
##                                115                               116
## Transport-moving:Self-emp-not-inc           Other-service:State-gov
##                                118                               123
##            Craft-repair:Local-gov           Priv-house-serv:Private
##                                143                               143
##       Prof-specialty:Self-emp-inc        Prof-specialty:Federal-gov
##                                157                               167
##    Other-service:Self-emp-not-inc       Exec-managerial:Federal-gov
##                                173                               179
##         Exec-managerial:State-gov           Protective-serv:Private
##                                186                               186
##           Other-service:Local-gov         Exec-managerial:Local-gov
##                                189                               212
##            Adm-clerical:State-gov            Adm-clerical:Local-gov
##                                250                               281
##                Sales:Self-emp-inc         Protective-serv:Local-gov
##                                281                               304
##          Adm-clerical:Federal-gov   Prof-specialty:Self-emp-not-inc
##                                316                               365
##            Sales:Self-emp-not-inc  Exec-managerial:Self-emp-not-inc
##                                376                               383
##      Exec-managerial:Self-emp-inc          Prof-specialty:State-gov
##                                385                               403
##  Farming-fishing:Self-emp-not-inc           Farming-fishing:Private
##                                430                               450
```

```
##     Craft-repair:Self-emp-not-inc          Prof-specialty:Local-gov
##                               523                               692
##             Tech-support:Private                             other
##                               723                              1008
##          Transport-moving:Private         Handlers-cleaners:Private
##                              1247                              1255
##         Machine-op-inspct:Private            Prof-specialty:Private
##                              1882                              2254
##           Exec-managerial:Private             Other-service:Private
##                              2647                              2665
##              Adm-clerical:Private                      Sales:Private
##                              2793                              2895
##             Craft-repair:Private
##                              3146
```

To do at home: merge the two variables `relationship` and `marital_status` together in a similar way to what we did here.

```
adult$marital_status = as.character(adult$marital_status)
adult$relationship = as.character(adult$relationship)
adult$relationship_status = paste(adult$marital_status, adult$relationship, sep = ":")
adult$relationship_status = as.factor(adult$relationship_status)
tab_relationship_status = sort(table(adult$relationship_status))
adult$marital_status = NULL
adult$relationship = NULL

adult$relationship_status=as.character(adult$relationship_status)
adult$relationship_status = ifelse(adult$relationship_status %in% names(tab_relationship_status[tab_rela
adult$relationship_status = as.factor(adult$relationship_status)
sort(table(adult$relationship_status))
```

```
##
##            Separated:Other-relative                   Married:Own-child
##                                  53                                  84
##                Separated:Own-child             Divorced:Other-relative
##                                  90                                 103
##              Married:Other-relative     Married-spouse-absent:Unmarried
##                                 119                                 120
##                               other Married-spouse-absent:Not-in-family
##                                 135                                 181
##                 Divorced:Own-child                   Widowed:Unmarried
##                                 308                                 343
##            Separated:Not-in-family                Separated:Unmarried
##                                 383                                 413
##              Widowed:Not-in-family        Never-married:Other-relative
##                                 432                                 548
##            Never-married:Unmarried                         Married:Wife
##                                 801                                1406
##                Divorced:Unmarried              Divorced:Not-in-family
##                                1535                                2268
##            Never-married:Own-child         Never-married:Not-in-family
##                                3929                                4447
##                    Married:Husband
##                               12463
```

We are finally ready to fit some probability estimation models for `income`! In lecture 16 we spoke about model selection using a cross-validation procedure. Let's build this up step by step. First, split the dataset into `Xtrain`, `ytrain`, `Xtest`, `ytest` using K=5.

```
K = 5
test_prop = 1 / K
train_indices = sample(1 : nrow(adult), round((1 - test_prop) * nrow(adult)))
adult_train = adult[train_indices, ]
y_train = adult_train$income
X_train = adult_train
X_train$income = NULL
test_indices = setdiff(1 : nrow(adult), train_indices)
adult_test = adult[test_indices, ]
y_test = adult_test$income
X_test = adult_test
X_test$income = NULL
```

Create the following four models on the training data in a `list` object named `prob_est_mods`: logit, probit, cloglog and cauchit (which we didn't do in class but might as well). For the linear component within the link function, just use the vanilla raw features using the `formula` object `vanilla`. Each model's key in the list is its link function name + "-vanilla". One for loop should do the trick here.

```
link_functions = c("logit", "probit", "cloglog", "cauchit")
vanilla = income ~ .
prob_est_mods = list()

for (link_function in link_functions){
  prob_est_mods[[paste(link_function, "vanilla", sep = "-")]]=glm(formula = vanilla, data = adult_train
}
```

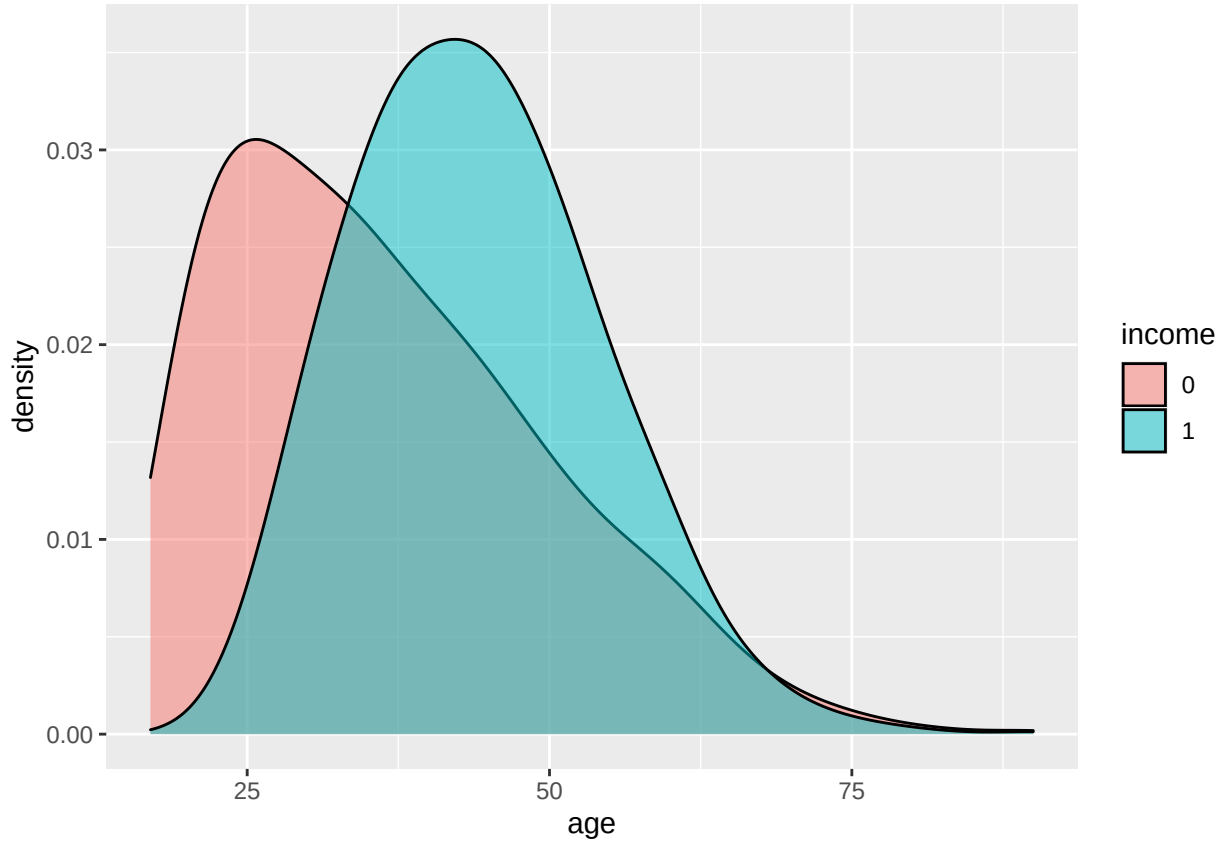
```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

Now let's get fancier. Let's do some variable transforms. Add `log_capital_loss` derived from `capital_loss` and `log_capital_gain` derived from `capital_gain`. Since there are zeroes here, use log_x = log(1 + x) instead of log_x = log(x). That's always a neat trick. Just add them directly to the data frame so they'll be picked up with the `.` inside of a formula.

```
adult$log_capital_loss = log(1 + adult$capital_loss)
adult$log_capital_gain = log(1+adult$capital_gain)
```

Create a density plot that shows the age distribution by `income`.

```
ggplot(adult) +
  aes(x = age) +
  geom_density(aes(fill = as.factor(income)), adjust = 2, alpha = 0.5) + scale_fill_discrete(name = "in
```



What do you see? Is this expected using common sense?

We see that as someone gets older, their income starts to increase since they are probably working full time, and moving up in their careers. Then, it starts to decline at around 55 - 60 years old as they begin to retire.

Now let's fit the same models with all link functions on a formula called `age_interactions` that uses interactions for `age` with all of the variables. Add all these models to the `prob_est_mods` list.

```
K = 5
test_prop = 1 / K
train_indices = sample(1 : nrow(adult), round((1 - test_prop) * nrow(adult)))
adult_train = adult[train_indices, ]
y_train = adult_train$income
X_train = adult_train
X_train$income = NULL
test_indices = setdiff(1 : nrow(adult), train_indices)
adult_test = adult[test_indices, ]
y_test = adult_test$income
X_test = adult_test
X_test$income = NULL


age_interactions = income ~ . *age

for (link_function in link_functions){
  prob_est_mods[[paste(link_function, "age_interactions", sep = "-")]]=glm(formula = age_interactions, 
}
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: algorithm did not converge

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: algorithm did not converge

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

Create a function called `brier_score` that takes in a probability estimation model, a dataframe `X` and its responses `y` and then calculates the brier score.

```
brier_score = function(prob_est_mod, X, y){
  phat = predict(prob_est_mod, X, type = "response")
  mean (- (y-phat)^2)
}
```

Now, calculate the in-sample Brier scores for all models. You can use the function `lapply` to iterate over the list and pass in in the function `brier_score`.

```
lapply(prob_est_mods, brier_score, X_train, y_train)
```

```
## $`logit-vanilla`
## [1] -0.102882
##
## $`probit-vanilla`
## [1] -0.1030133
##
## $`cloglog-vanilla`
## [1] -0.1037922
##
## $`cauchit-vanilla`
## [1] -0.1043954
##
## $`logit-age_interactions`
## [1] -0.09919656
##
## $`probit-age_interactions`
## [1] -0.1005526
##
## $`cloglog-age_interactions`
## [1] -0.09997617
##
## $`cauchit-age_interactions`
## [1] -0.1002172
```

Now, calculate the out-of-sample Brier scores for all models. You can use the function `lapply` to iterate over the list and pass in the function `brier_score`.

```
lapply(prob_est_mods, brier_score, X_test, y_test)
```

```
## $`logit-vanilla`
## [1] -0.1044429
##
## $`probit-vanilla`
## [1] -0.1043848
##
## $`cloglog-vanilla`
## [1] -0.1053128
##
## $`cauchit-vanilla`
## [1] -0.1066176
##
## $`logit-age_interactions`
## [1] -0.1031969
##
## $`probit-age_interactions`
## [1] -0.1041485
##
## $`cloglog-age_interactions`
## [1] -0.1042259
##
## $`cauchit-age_interactions`
## [1] -0.1051838
```

Which model wins in sample and which wins out of sample? Do you expect these results? Explain.

The logit with the interactions on age has the best in-sample and out of sample Brier score, which is exactly what we expect. We usually expect the log model to be better than the other models, and we expected the models with the interactions to do better compared to the "vanilla" models because the interactions add more complexity to the models and can account for more complex relationships between the features.

What is wrong with this model selection procedure? There are a few things wrong.

1) there is only one test subset/ didn't do cross validation so error may have high variability
2)

Run all the models again. This time do three splits: subtrain, select and test. After selecting the best model, provide a true oos Brier score for the winning model.

```
n = nrow(adult)
K = 5
test_indices = sample(1 : n, size = n * 1 / K)
master_train_indices = setdiff(1 : n, test_indices) ##overall train
select_indices = sample(master_train_indices, size = n * 1 / K)
subtrain_indices = setdiff(master_train_indices, select_indices)

adult_train = adult[master_train_indices,]

adult_subtrain = adult[subtrain_indices, ]
y_subtrain = adult_subtrain$income
```

```r
adult_select = adult[select_indices, ]
y_select = adult_select$income
adult_select$income = NULL

adult_test = adult[test_indices, ]
y_test = adult_test$income
adult_test$income = NULL
```

```r
mods = list()

for (link_function in link_functions){
  mods[[paste(link_function, "vanilla", sep = "-")]]=glm(formula = vanilla, data = adult_subtrain, fami
}
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: algorithm did not converge

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: algorithm did not converge

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```r
for (link_function in link_functions){
  mods[[paste(link_function, "age_interactions", sep = "-")]]=glm(formula = age_interactions, data = ad
}
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: algorithm did not converge

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: algorithm did not converge

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: algorithm did not converge
```

```r
briers = lapply(mods, brier_score, adult_select, y_select)
which_final = which.max(briers)
which_final
```

```
## logit-vanilla
##             1
```

```
g_final = glm(income ~., data = adult_train, family = binomial(link = logit))
```

```
## Warning: glm.fit: algorithm did not converge
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
brier_score(g_final, adult_test, y_test)
```

```
## [1] -0.1036743
```