CCLE Gene Expression

Seema Plaisier

11/13/23

Overview

This R code written in Rmd format is template code meant to be a starting point to analyze gene expression data from the Cancer Cell Line Encyclopedia (CCLE). You have completed a tutorial in R which should help you to understand and run this code yourself to look across various features of the data set. We will start by analyzing the expression of XIST and then you will modify this to analyze different genes. Please go through this code line by line, looking at the variables (double click on the variables in the Environment tab after running a line) as you go. Once you know what the code is doing, you can use this code to help you develop additional code to look at the data in different ways to ask and answer biological questions we come up with as we go.

This code sometimes shows multiple ways to do the same thing just for your learning; feel free to stick with the way you like best when adding to the code. Similarly, you may think of ways that you think would work better than how I did it, feel free to improve the code to your liking.

At minimum, you will need to modify the following things: 1) Change the author in the header to be your name 2) Change the paths to the data to be paths in your directories 3) Change the gene names and cell line features to reflect what you are wanting to study

If you get confused about what you are looking at or get errors that you don’t know how to solve, please ask for help from the instructors and your classmates using Slack.

Import necessary R packages

In this chunk, we will be installing and loading the R packages containing special functions we are using in the main body of the code.

The function ‘require’ returns a true value if a package is already installed and a false if it is not installed. So if a false is returned, not (!) that value will be true, and we will call the function to install the packages. The installation should only take place the first time you run this code (or it won’t if you already have these packages installed in your R environment from running other code before this class). After the installation step, we will use the ‘library’ function to load the packages for use in this code.

```r
# check if the package has been installed
# if not, install it
if(!require(ggplot2)){
    install.packages("ggplot2")
}
```

```
## Loading required package: ggplot2
```

```
## Warning: package ’ggplot2’ was built under R version 4.2.3
```

```r
if(!require(UpSetR)){
    install.packages("UpSetR")
}
```

```
## Loading required package: UpSetR
```

```r
# load packages for use
library(ggplot2)
library(UpSetR)
```

Set options for printing reports

```r
# this will make sure that the code doesn't run off the page when printing a report
knitr::opts_chunk$set(tidy.opts = list(width.cutoff = 50), tidy = TRUE)
```

Set working directory and data directories

Your working directory is where all your output files will be stored by default. You can set it by copying the path of the directory by navigating to the desired directory and copying the 'pwd' (present working directory) (Linux) or from the address bar in explorer windows (Windows or Mac). Make sure you change the slashes to forward slashes '/' if needed.

The 'setwd' function sets your working directory to the path you specify. The 'list.files' and 'list.dirs' functions can be used to print what's inside that directory.

In the code you run in this class, you will need to change the path to the working directory so that your output files get put in your /scratch drive on Sol or to a directory on your computer.

```r
# store the path to the working directory in a
# variable make sure to include the / at the end
# of the directory path if you plan to put it
# together with the path to files inside the
# directory

working_directory = "C:/Users/splaisie/Dropbox (ASU)/GenomicsCURE/CCLE_Fall2023/"
# working_directory PATH WILL NEED TO BE
# CHANGED!!!  initially set to directories in
# splaisie's storage --> change this a path in
# your own directory, such as where you put the
# expression data and annotation data

# set the working directory to where you want
# your output to go
setwd(working_directory)

# store the path to the directory where your data
# files are stored in a variable this makes it so
# you can copy code into different directories
# for different analysis, but always have your
# input data be in the same place make sure to
# include the / at the end of the directory path
# so when we paste it to a file name, the path
# makes sense

data_directory = "C:/Users/splaisie/Dropbox (ASU)/GenomicsCURE/CCLE_Fall2023/"
# data_directory PATH WILL NEED TO BE CHANGED!!!
# initially set to directories in splaisie's
# storage --> change to your own directory where
# you put your expression and annotation data
# files
```

Reading data files

Reading data into variables is a very important task in R. For most scientific problems, you will read a table of data commonly separated by commas or tabs. The 'read.csv' function can be used to read tabular data from files; the delimiter is set to be a comma by default in this function. If you have tab-delimited data, you can set the delimiter to a tab. Running these commands will put variables containing your data in the workspace.

See learning pages for specific instructions on how to download the gene expression matrix file and the annotation information for CCLE cell lines. After the data is loaded, you can see it by clicking on variables in the Environment tab.

The gene expression matrix file contains a table with all the genes as rows and all the cell lines in the CCLE as columns, plus a few extra columns giving important information about the genes like name and IDs in important databases. It is is fairly large and will likely take a few minutes to load.

The sample annotation file contains information about each of the cell lines in the CCLE. This includes tumor tissue type the cell line was derived from, patient information like age and "gender" (reported sex), as well as some information about how the cell line was grown prior to sample collection for RNA sequencing.

```r
# make sure you have changed the path to
# data_directory to your own storage space and
# that you have downloaded the files the names
# below into that directory

# load the CCLE gene expression data
CCLE_data = read.csv(paste0(data_directory, "CCLE_RNAseq_genes_counts_20180929.csv"),
    header = TRUE)

# load the annotation data
annotation_data = read.csv(paste0(data_directory, "Cell_lines_annotations_20181226.txt"),
    header = TRUE, sep = "\t")

# at this point, you should be able to double
# click on the CCLE_data and annotation_data
# variables from the Environment tab and look
# around
```

Exploring the CCLE data set

To get an idea of what is in the sample annotation given our research focus, let's pull out some information from the CCLE sample annotation.

Here we use the ‘levels’ function to show a non-redundant list of values in specific columns of interest in the CCLE. This will tell you what values are there, and show you some if there are values you don’t expect to find so you can handle those appropriately in your code.

We also look at reported sex (“Gender” column) and pathology (“Pathology” column). Pathology means the type of tumor the cell line was isolated from, such as primary (first place the cancer was discovered), metastastis (tumor spread to a different location), benign neoplasia (clump of cells that are not developing into a true tumor), and rarely even a cell line from a normal, non-cancerous tissue.

To pull specific data of interest, we will use subsetting functions with a data frame. We select a specific column in the data frame by asking for which rows have a specific value in another column. So for, `test_matrix$data [test_matrix$tissue == "brain"]`, we are asking to pull out the ‘data’ column of ‘test_matrix’ and filter for the rows where the ‘tissue’ column equals “brain”.

When looking through the ‘annotation_data’ variable, there are plenty of NA values listed, indicating that no value is given for that cell. Here the ‘na.omit’ function is used to remove those NA values.

```
# list values given for Gender in the cell line
# annotation
levels(factor(annotation_data$Gender))
```

```
## [1] ""       "female" "male"   "null"
```

```
# get the cell lines reported as female or male
lines_reported_female = annotation_data$CCLE_ID[annotation_data$Gender ==
    "female"]
length(lines_reported_female)
```

```
## [1] 807
```

```
lines_reported_male = annotation_data$CCLE_ID[annotation_data$Gender ==
    "male"]
length(lines_reported_male)
```

```
## [1] 908
```

```
# get the cell lines reported with different
# pathology (type of tumor/tissue)
levels(factor(annotation_data$Pathology))
```

```
## [1] ""                 "benign_neoplasia" "metastasis"       "normal"
## [5] "primary"
```

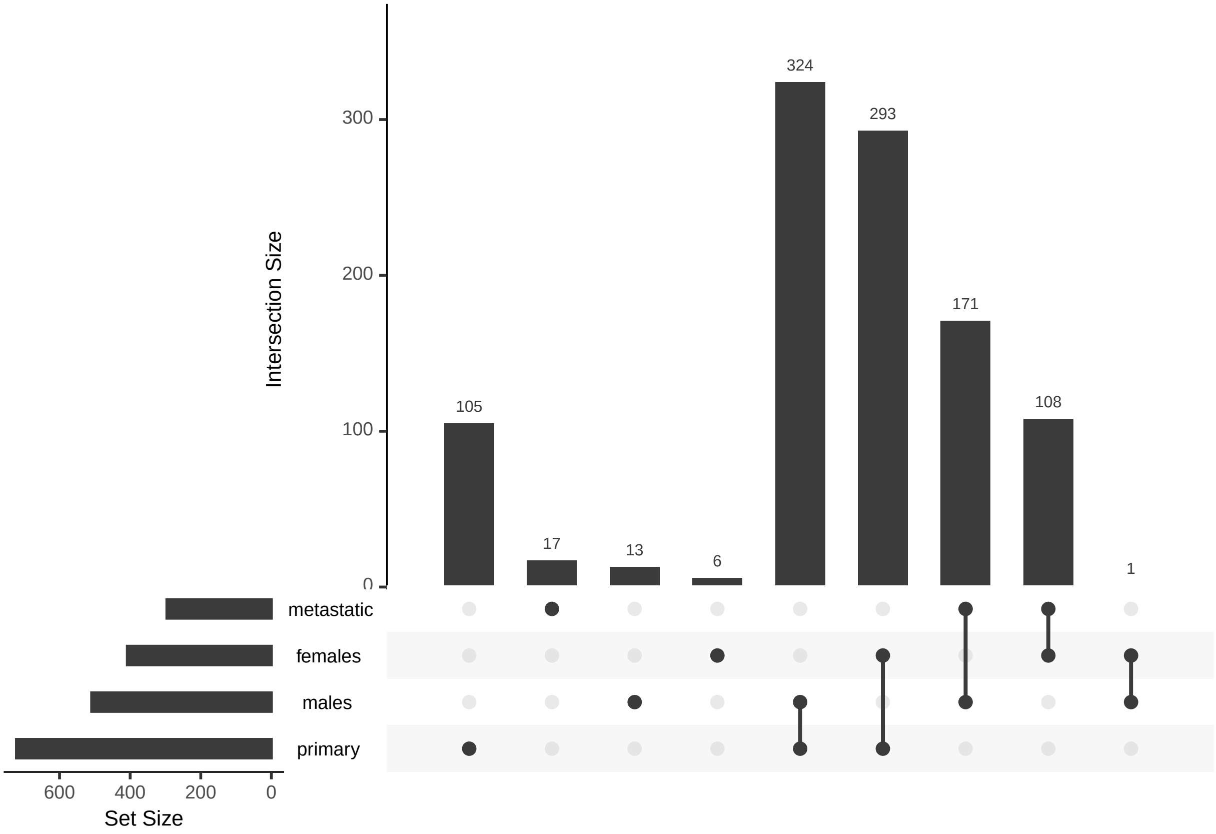
```
lines_reported_benign = na.omit(annotation_data$CCLE_ID[annotation_data$Pathology ==
    "benign_neoplasia"])
lines_reported_primary = na.omit(annotation_data$CCLE_ID[annotation_data$Pathology ==
    "primary"])
lines_reported_metastasis = na.omit(annotation_data$CCLE_ID[annotation_data$Pathology ==
    "metastasis"])
lines_reported_normal = na.omit(annotation_data$CCLE_ID[annotation_data$Pathology ==
    "normal"])

# to see the contents of any of the lists, you
# can just type the name of the variable in the
# Console below
```

Now that we have pulled out lists of cell lines with particular annotations, let's get an idea of how those lists compare to one another.

To do this, we are creating an upset plot. This plot visualizes overlap when you have more than 2 groups to compare.

For more info on the upset plot and related functions, see here: https://r-graph-gallery.com/upset-plot.html https://en.wikipedia.org/wiki/UpSet_Plot#/media/File:UpSet_Intersection_Concept.svg

```
# create a list of lists that we are going to
# compare
list_sex_pathology = list(primary = lines_reported_primary,
    metastatic = lines_reported_metastasis, females = lines_reported_female,
    males = lines_reported_male)

# create an upset plot to compare those lists to
# one another
upset(fromList(list_sex_pathology))
```



Create lookup tables

To associate our cell lines with annotation information, we can created named lists to serve as look up table. In these tables, we will pull all the cells with the annotation we are interested in, and name them with the cell line so we can quickly look it up below.

```
# this code will make a quick lookup table to get
# the reported age of the patient when the cell
# line was started

# pull out the Gender column of the annotation
# data frame
get_reported_sex = annotation_data$Gender

# use the CCLE_ID to label each entry of the
# reported sex this creates a lookup table
names(get_reported_sex) = annotation_data$CCLE_ID

# remove NA values which indicate that no sex was
# reported
get_reported_sex = na.omit(get_reported_sex)

# see what values you got from the reported sex
# in the Gender column it's a good idea to check
# since sometimes there are values you don't
# expect to see
levels(factor(get_reported_sex))
```

```
## [1] ""       "female" "male"   "null"
```

```
# filter for only the entries that are reported
# as 'female' or 'male'
get_reported_sex = get_reported_sex[get_reported_sex ==
    "female" | get_reported_sex == "male"]  # filter

# do same thing with other features that are
# interesting

# create a lookup table for pathology listed as
# 'primary' or 'metastasis'
get_pathology = annotation_data$Pathology
names(get_pathology) = annotation_data$CCLE_ID
get_pathology = na.omit(get_pathology)
levels(factor(get_pathology))
```

```
## [1] ""                 "benign_neoplasia" "metastasis"       "normal"
## [5] "primary"
```

```
get_pathology = get_pathology[get_pathology == "primary" |
    get_pathology == "metastasis"]  # filter

# create a lookup table for age
get_age = annotation_data$Age
names(get_age) = annotation_data$CCLE_ID
get_age = na.omit(get_age)
levels(factor(get_age))
```

```
##  [1] "0.25"        "0.5"         "0.75"        "0.8333"      "1"
##  [6] "1.166666667" "1.6666"      "1.67"        "2"           "2.166666667"
## [11] "3"           "3.5"         "4"           "5"           "6"
## [16] "7"           "8"           "9"           "10"          "11"
## [21] "12"          "13"          "14"          "15"          "16"
## [26] "17"          "18"          "19"          "20"          "21"
## [31] "22"          "23"          "24"          "25"          "26"
## [36] "27"          "28"          "29"          "30"          "31"
## [41] "32"          "33"          "34"          "35"          "36"
## [46] "37"          "38"          "39"          "40"          "41"
## [51] "42"          "43"          "44"          "45"          "46"
## [56] "47"          "48"          "49"          "50"          "51"
## [61] "52"          "53"          "54"          "55"          "56"
## [66] "57"          "58"          "59"          "60"          "61"
## [71] "62"          "63"          "64"          "65"          "66"
## [76] "67"          "68"          "69"          "70"          "71"
## [81] "72"          "73"          "74"          "75"          "76"
## [86] "77"          "78"          "79"          "80"          "81"
## [91] "82"          "84"          "85"          "92"
```

```
# create a lookup table for tumor site (the
# tissue type or organ the tumor was found in)
get_site = annotation_data$Site_Primary
names(get_site) = annotation_data$CCLE_ID
get_site = na.omit(get_site)
levels(factor(get_site))
```

```
##  [1] "autonomic_ganglia"                  "biliary_tract"
##  [3] "bone"                               "breast"
##  [5] "central_nervous_system"             "endometrium"
##  [7] "haematopoietic_and_lymphoid_tissue" "kidney"
##  [9] "large_intestine"                    "liver"
## [11] "lung"                               "oesophagus"
## [13] "ovary"                              "pancreas"
## [15] "pleura"                             "prostate"
## [17] "salivary_gland"                     "skin"
## [19] "small_intestine"                    "soft_tissue"
## [21] "stomach"                            "thyroid"
## [23] "upper_aerodigestive_tract"          "urinary_tract"
```

Gene of Interest

In this chunk, we will set the name of the gene we are interested in analyzing. You will need to make sure that the gene name you choose is in the 'Description' column of the annotation_data variable which contains the gene expression matrix.

Here we use the 'subset' function to pull out the row with the selected gene name in the Description column. We print what we found to make sure that we actually got what we wanted. If you do not see any information in the 'chosen_gene_data' variable, either the gene name is spelled wrong or that gene is not included in the data set under that name. It might be listed using an alternative gene name.

```
# set the gene that you are interested in
chosen_gene = "XIST"
```

```r
# pull out the data for this gene from the full
# data matrix
chosen_gene_data = subset(CCLE_data, Description ==
    chosen_gene)

# print the first few entries just to make sure
# you have data if not, you might have indicated
# a gene name that is not present in the data set
# which could be due to incorrect spelling or
# capitalization
print(chosen_gene_data[1:10])
```

```
##                    Name Description X22RV1_PROSTATE X2313287_STOMACH
## 54376 ENSG00000229807.5        XIST              23               23
##       X253JBV_URINARY_TRACT X253J_URINARY_TRACT X42MGBA_CENTRAL_NERVOUS_SYSTEM
## 54376                    55                  67                             64
##       X5637_URINARY_TRACT X59M_OVARY X639V_URINARY_TRACT
## 54376                  15      13605                  78
```

```r
# remove the first two columns (Name and
# Description) since we are only want to plot
# expression data values
chosen_gene_subset = subset(chosen_gene_data, select = -c(1,
    2))
```

Gene expression in cell lines with different reported sex

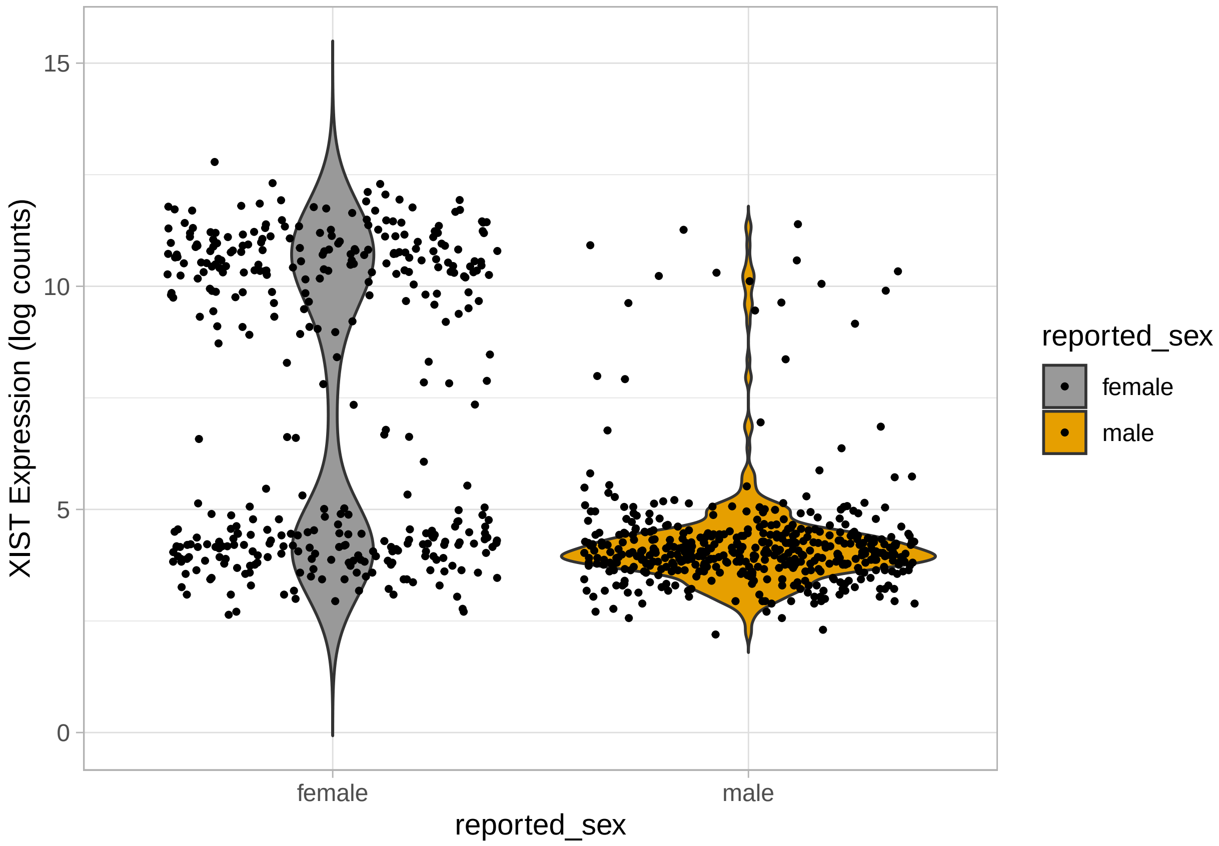
Here we want to look at the expression of the chosen gene in cell lines annotated to have ‘female’ or ‘male’ as their reported sex. We would like to see the range of values we see for the expression of the gene– do all cell lines from patients reported as female have high expression and as male have low? What threshold do we use to even say what is high or low expression?

To plot the expression of the gene, the ggplot2 package will be used. This package is very powerful for easily switching between different types of plots and customizing aesthetic features for making it easier to read.

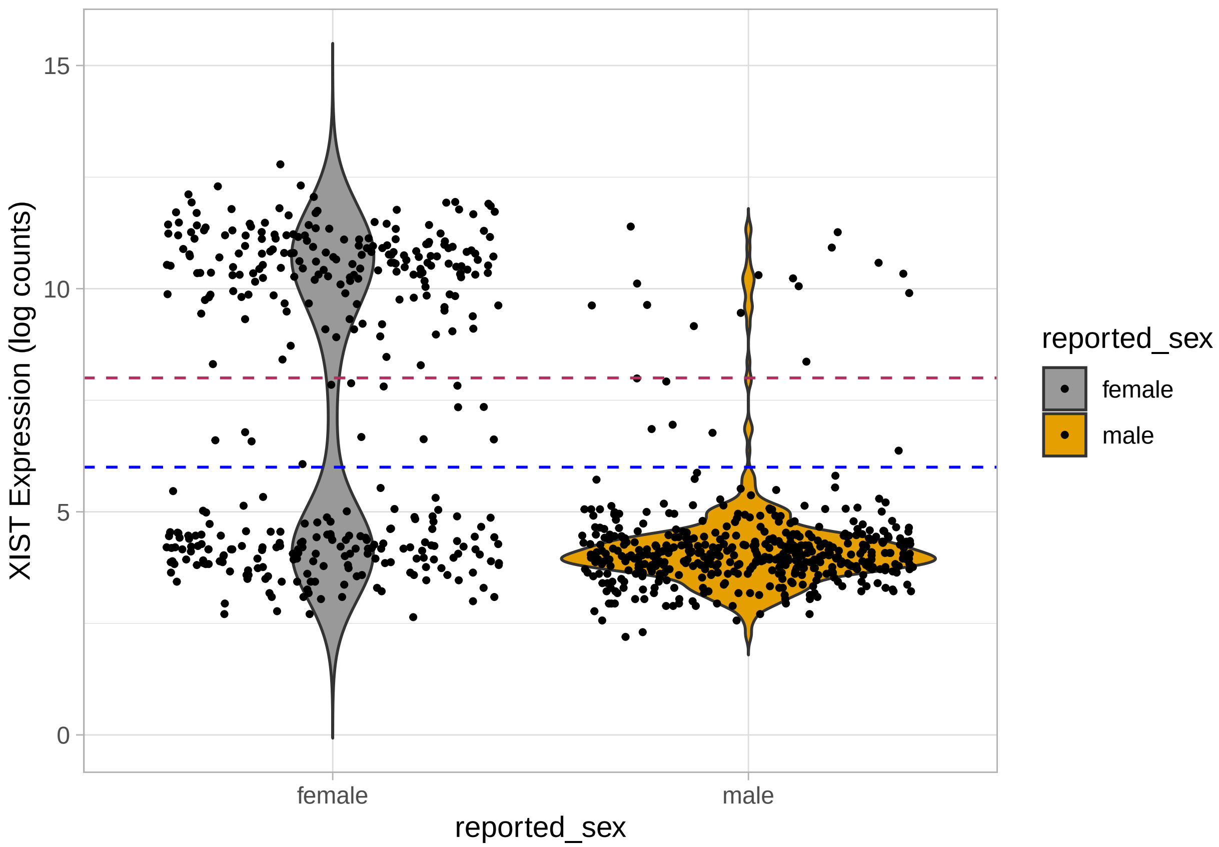
Check out this page for more options for ggplot violin plots: http://www.sthda.com/english/wiki/ggplot2-violin-plot-quick-start-guide-r-software-and-data-visualization

```r
# pull out expression data for cell lines that
# had a reported sex of female or male
chosen_gene_reported_sex = chosen_gene_subset[colnames(chosen_gene_subset) %in%
    names(get_reported_sex)]

# transpose and log transform the data so it's
# easier to work with and put it into a data
# frame
chosen_gene_reported_sex = as.data.frame(t(log(chosen_gene_reported_sex)))

# set the name of the column to be 'expression'
# so we know what to call it when we plot
colnames(chosen_gene_reported_sex) = "expression"

# add a reported sex column
```

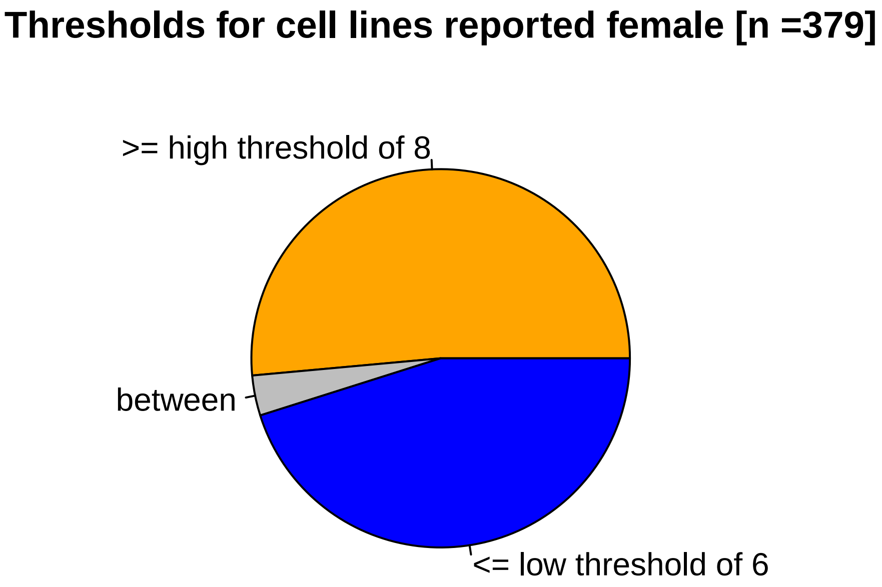
```
chosen_gene_reported_sex$reported_sex = vector(length = nrow(chosen_gene_reported_sex))  # empty placeh
for (cell_line in rownames(chosen_gene_reported_sex)) {
    chosen_gene_reported_sex[cell_line, "reported_sex"] = get_reported_sex[cell_line]
}

# use ggplot to visualize the data as a violin
# jitter plot

ggplot(chosen_gene_reported_sex, aes(x = reported_sex,
    y = chosen_gene_reported_sex$expression, fill = reported_sex)) +
    geom_violin(trim = FALSE) + scale_fill_manual(values = c("#999999",
    "#E69F00")) + geom_jitter(size = 0.75) + ylab(paste0(chosen_gene,
    " Expression (log counts)")) + theme_light()
```



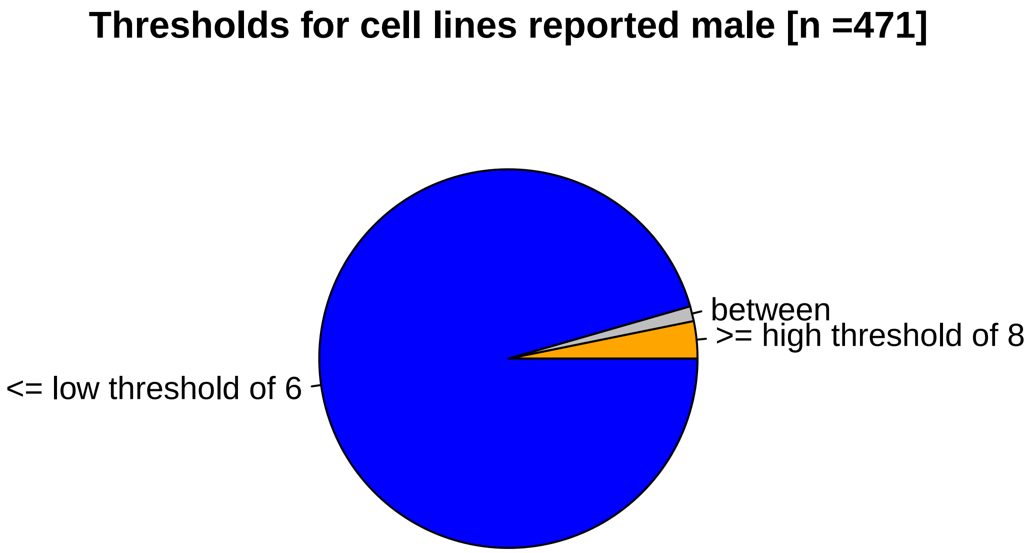
```
# add line to think about thresholds that can be
# used to predict reported sex
high_threshold = 8
low_threshold = 6
ggplot(chosen_gene_reported_sex, aes(x = reported_sex,
    y = chosen_gene_reported_sex$expression, fill = reported_sex)) +
    geom_violin(trim = FALSE) + scale_fill_manual(values = c("#999999",
    "#E69F00")) + geom_jitter(size = 0.75) + ylab(paste0(chosen_gene,
    " Expression (log counts)")) + geom_hline(yintercept = high_threshold,
    linetype = "dashed", color = "maroon") + geom_hline(yintercept = low_threshold,
    linetype = "dashed", color = "blue") + theme_light()
```



```
# count how many cell lines reported female are
# in each region defined by the thresholds chosen
num_females_over_high_threshold = nrow(subset(chosen_gene_reported_sex,
    reported_sex == "female" & expression >= high_threshold))
num_females_between_thresholds = nrow(subset(chosen_gene_reported_sex,
    reported_sex == "female" & expression < high_threshold &
        expression > low_threshold))
num_females_under_low_threshold = nrow(subset(chosen_gene_reported_sex,
    reported_sex == "female" & expression <= low_threshold))

# pie chart to show proportions using the chosen
# threshold can use this to try a few different
# thresholds
female_threshold_nums = c(num_females_over_high_threshold,
    num_females_between_thresholds, num_females_under_low_threshold)
female_threshold_labels = c(paste0(">= high threshold of ",
    high_threshold), "between", paste0("<= low threshold of ",
    low_threshold))
pie(female_threshold_nums, labels = female_threshold_labels,
    main = paste0("Thresholds for cell lines reported female [n =",
        sum(female_threshold_nums), "]"), col = c("orange",
        "grey", "blue"))
```



```r
# now do the same for cell lines reported as male
num_males_over_high_threshold = nrow(subset(chosen_gene_reported_sex,
    reported_sex == "male" & expression >= high_threshold))
num_males_between_thresholds = nrow(subset(chosen_gene_reported_sex,
    reported_sex == "male" & expression < high_threshold &
        expression > low_threshold))
num_males_under_low_threshold = nrow(subset(chosen_gene_reported_sex,
    reported_sex == "male" & expression <= low_threshold))

male_threshold_nums = c(num_males_over_high_threshold,
    num_males_between_thresholds, num_males_under_low_threshold)
male_threshold_labels = c(paste0(">= high threshold of ",
    high_threshold), "between", paste0("<= low threshold of ",
    low_threshold))
pie(male_threshold_nums, labels = male_threshold_labels,
    main = paste0("Thresholds for cell lines reported male [n =",
        sum(male_threshold_nums), "]"), col = c("orange",
        "grey", "blue"))
```



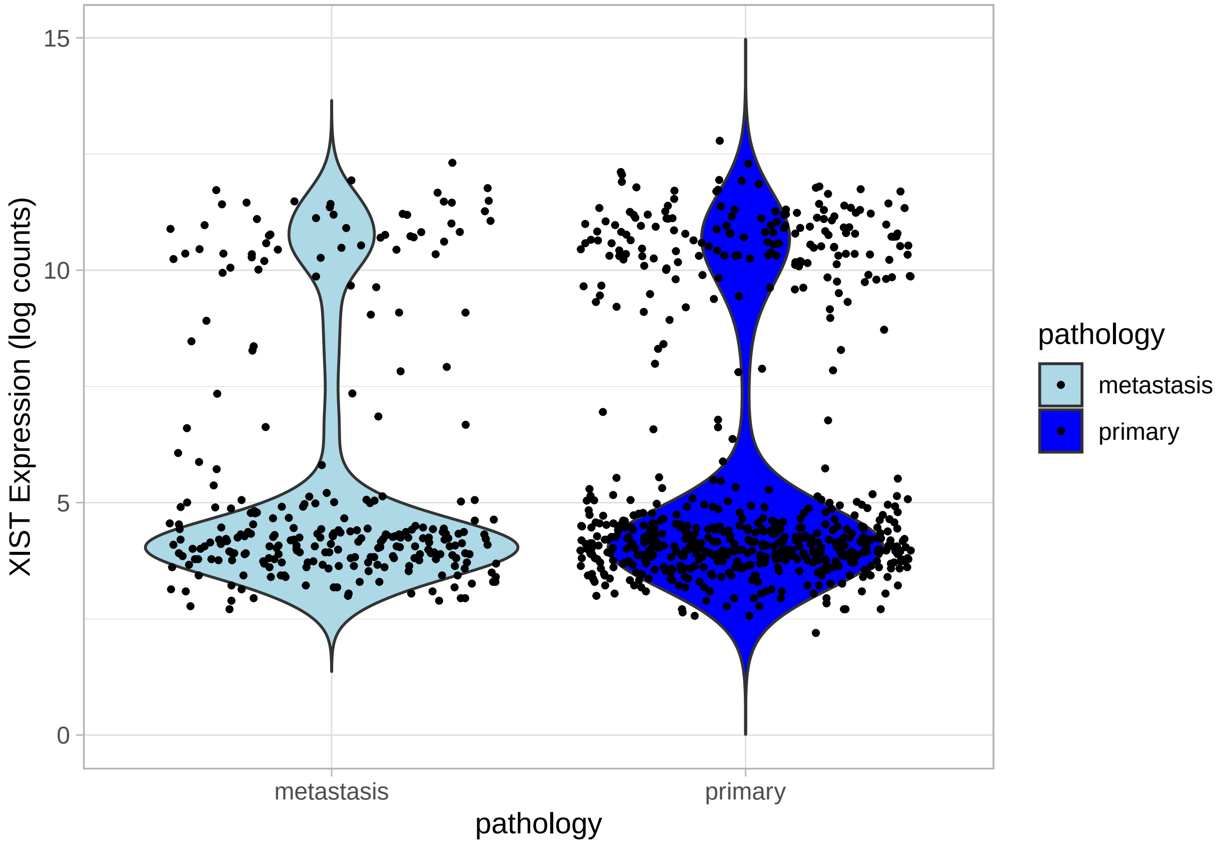
```
# use chosen thresholds to predict sex based on
# expression

# start with all the cell lines that have
# expression of the gene you are using to make a
# prediction make a data frame containing the
# expression
predict_sex = data.frame(t(chosen_gene_subset))
colnames(predict_sex) = "expression"

# add column for predicted sex NOTE: how this
# prediction is done should not be the same for
# all genes add a reported age column
predict_sex$expression_category = vector(length = nrow(predict_sex))  # empty placeholders
for (cell_line in rownames(predict_sex)) {
    if (log(predict_sex[cell_line, "expression"]) >=
        high_threshold) {
        predict_sex[cell_line, "expression_category"] = "high_expression"
    } else if (log(predict_sex[cell_line, "expression"]) <=
        low_threshold) {
        predict_sex[cell_line, "expression_category"] = "male_range"
    } else if (log(predict_sex[cell_line, "expression"]) >
        low_threshold & log(predict_sex[cell_line,
        "expression"]) < high_threshold) {
        predict_sex[cell_line, "expression_category"] = "intermediate_expression"
    }
}

# add gene name to column name
# 'expression_category'
colnames(predict_sex)[colnames(predict_sex) == "expression_category"] = paste0(chosen_gene,
    "_expression_category")

# add in the reported sex if present
predict_sex$reported_sex = vector(length = nrow(predict_sex))  # empty placeholders
for (cell_line in rownames(predict_sex)) {
    predict_sex[cell_line, "reported_sex"] = get_reported_sex[cell_line]
}

# Add the name of the gene you used to predict
# for good housekeeping
expression_index = which(colnames(predict_sex) == "expression")
colnames(predict_sex)[expression_index] = paste0(chosen_gene,
    "_log_expression (thresholds = ", low_threshold,
    ",", high_threshold, ")")

# write to output file so we can look at it later
predicted_sex_output_file = paste0("predicted_sex_",
    chosen_gene, ".csv")
write.csv(predict_sex, file = predicted_sex_output_file)
```

Gene expression in cell lines with different pathology

Here we want to look at the expression of the chosen gene in cell lines according to their pathology. There are a few that have benign and normal listed, but let's focus on primary or metastasis because that is the majority of the data. Does the gene appear to be lower in metastatic tumor cell lines than primary tumor cell lines? What trends to we see in primary tumor cell lines from reported females vs males? etc.

```
# pull out expression data for cell lines that
# had a pathology annoted
chosen_gene_pathology = chosen_gene_subset[colnames(chosen_gene_subset) %in%
    names(get_pathology)]

# transpose and log transform the data so it's
# easier to work with and put it into a data
# frame
chosen_gene_pathology = as.data.frame(t(log(chosen_gene_pathology)))

# set the name of the column to be 'expression'
# so we know what to call it when we plot
colnames(chosen_gene_pathology) = "expression"

# create another column called 'pathology' and
# fill it with the pathology for the cell line
chosen_gene_pathology$pathology = vector(length = nrow(chosen_gene_pathology))  # empty placeholders
for (cell_line in rownames(chosen_gene_pathology)) {
    chosen_gene_pathology[cell_line, "pathology"] = get_pathology[cell_line]
}
```

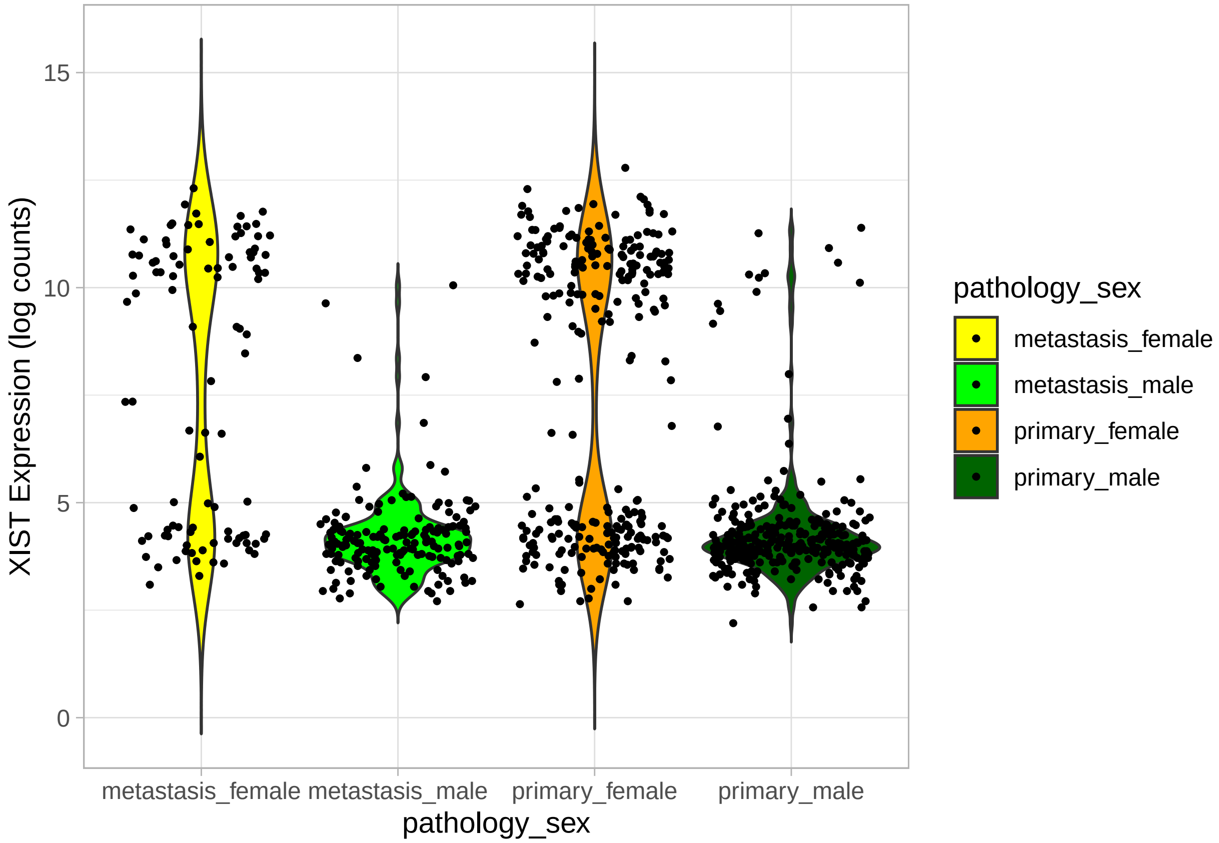
```
# use ggplot to visualize the data as a violin
# jitter plot

ggplot(chosen_gene_pathology, aes(x = pathology, y = chosen_gene_pathology$expression,
    fill = pathology)) + geom_violin(trim = FALSE) +
    scale_fill_manual(values = c("lightblue", "blue")) +
    geom_jitter(size = 0.75) + ylab(paste0(chosen_gene,
    " Expression (log counts)")) + theme_light()
```



```
# create another column called 'reported_sex' and
# fill it with the reported for the cell line
chosen_gene_pathology$reported_sex[rownames(chosen_gene_pathology) %in%
    lines_reported_female] = "female"
chosen_gene_pathology$reported_sex[rownames(chosen_gene_pathology) %in%
    lines_reported_male] = "male"

# thinking about what we saw before, there are
# probably cell lines that have a pathology
# listed but not a reported sex

# confirm this by looking
sum(is.na(chosen_gene_pathology$reported_sex))
```

```
## [1] 101
```

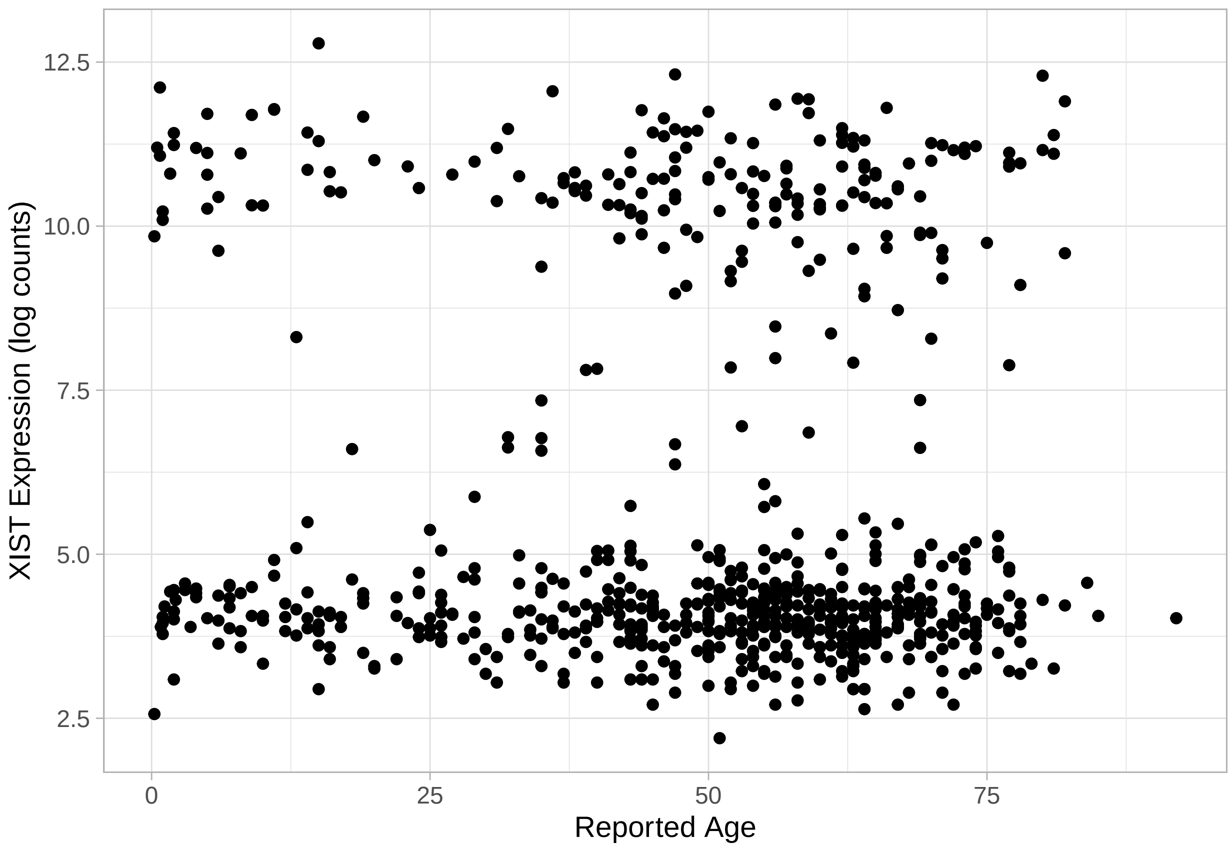
```
# let's make a copy of this dataframe but
# filtered for those with a reported sex
chosen_gene_pathology_sex = subset(chosen_gene_pathology,
    !is.na(chosen_gene_pathology$reported_sex))

# make a new column to group for pathology and
# sex
chosen_gene_pathology_sex$pathology_sex <- paste0(chosen_gene_pathology_sex$pathology,
    "_", chosen_gene_pathology_sex$reported_sex)

# use ggplot to visualize the data as a violin
# jitter plot
ggplot(chosen_gene_pathology_sex, aes(x = pathology_sex,
    y = chosen_gene_pathology_sex$expression, fill = pathology_sex)) +
    geom_violin(trim = FALSE) + scale_fill_manual(values = c("yellow",
    "green", "orange", "darkgreen")) + geom_jitter(size = 0.75) +
    ylab(paste0(chosen_gene, " Expression (log counts)")) +
    theme_light()
```



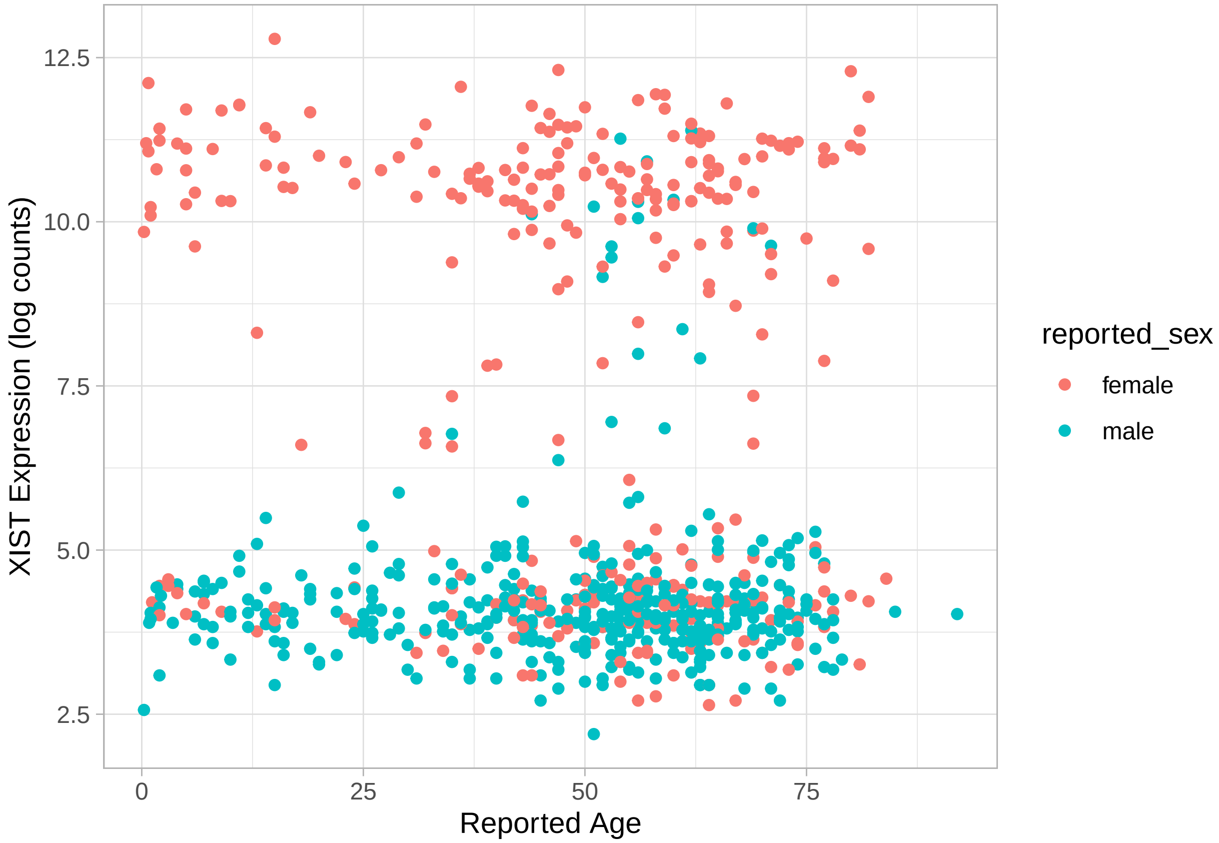
Gene expression in cell lines by age

Knowing that sometimes sex chromosomes can be lost as individuals age, let's take a look at the gene's expression by age, with and without considering the pathology

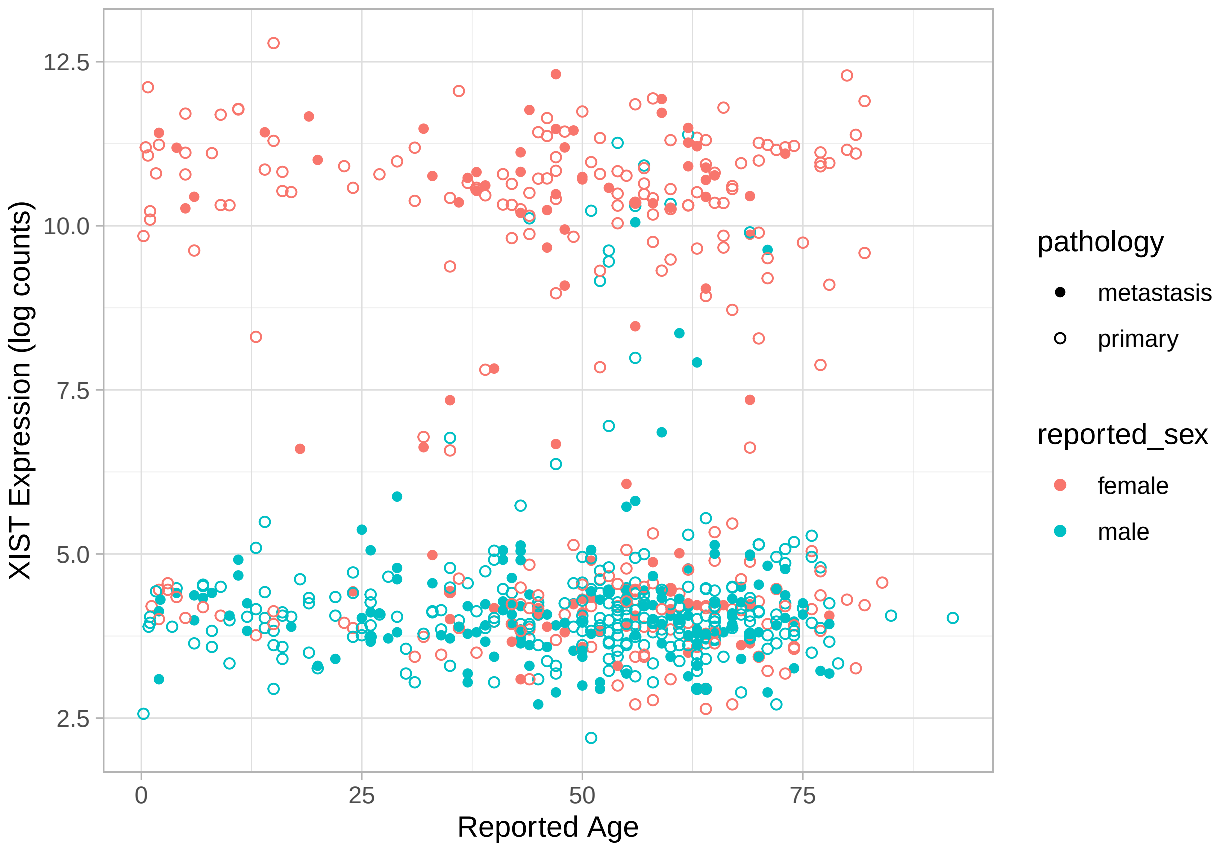
```r
# pull out the expression of our chosen gene for
# those cell lines where age is reported
chosen_gene_reported_age = chosen_gene_subset[colnames(chosen_gene_subset) %in%
    names(get_age)]

# transpose and log transform the data so it's
# easier to work with and put it into a data
# frame
chosen_gene_reported_age = as.data.frame(t(log(chosen_gene_reported_age)))

# set the name of the column to be 'expression'
# so we know what to call it when we plot
colnames(chosen_gene_reported_age) = "expression"


# add a reported age column
chosen_gene_reported_age$age = vector(length = nrow(chosen_gene_reported_age))  # empty placeholders
for (cell_line in rownames(chosen_gene_reported_age)) {
    chosen_gene_reported_age[cell_line, "age"] = get_age[cell_line]
}

# use ggplot to visualize the data as a scatter
# plot
ggplot(chosen_gene_reported_age, aes(x = chosen_gene_reported_age$age,
    y = chosen_gene_reported_age$expression)) + geom_point() +
    xlab("Reported Age") + ylab(paste0(chosen_gene,
    " Expression (log counts)")) + theme_light()
```



```
# consider reported sex

chosen_gene_reported_age$reported_sex = vector(length = nrow(chosen_gene_reported_age))  # empty placeh
for (cell_line in rownames(chosen_gene_reported_age)) {
    chosen_gene_reported_age[cell_line, "reported_sex"] = get_reported_sex[cell_line]
}
# filter empty values
chosen_gene_reported_age = chosen_gene_reported_age[!is.na(chosen_gene_reported_age$reported_sex),
    ]

# use ggplot to visualize the data as a scatter
# plot with reported sex as color
ggplot(chosen_gene_reported_age, aes(x = chosen_gene_reported_age$age,
    y = chosen_gene_reported_age$expression)) + geom_point(aes(color = reported_sex)) +
    xlab("Reported Age") + ylab(paste0(chosen_gene,
    " Expression (log counts)")) + theme_light()
```



```
# consider pathology

chosen_gene_reported_age$pathology = vector(length = nrow(chosen_gene_reported_age))  # empty placehold
for (cell_line in rownames(chosen_gene_reported_age)) {
    chosen_gene_reported_age[cell_line, "pathology"] = get_pathology[cell_line]
}
# filter empty values
chosen_gene_reported_age = chosen_gene_reported_age[!is.na(chosen_gene_reported_age$pathology),
    ]

# use ggplot to visualize the data as a scatter
# plot with reported sex as color
ggplot(chosen_gene_reported_age, aes(x = chosen_gene_reported_age$age,
    y = chosen_gene_reported_age$expression)) + geom_point(aes(color = reported_sex,
    shape = pathology)) + scale_shape_manual(values = c(16,
    1)) + xlab("Reported Age") + ylab(paste0(chosen_gene,
    " Expression (log counts)")) + theme_light()
```

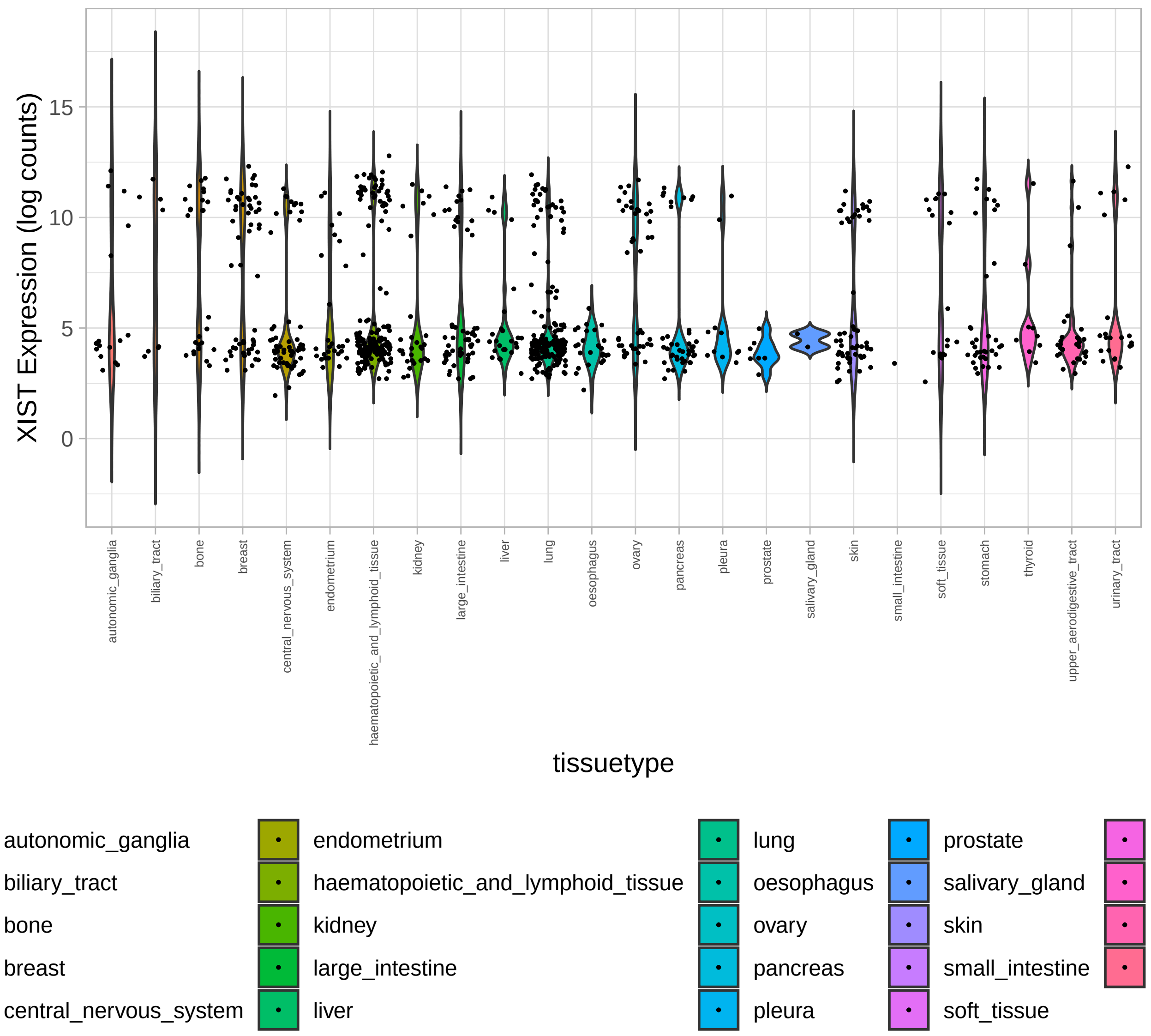


Gene expression in cell lines with tissue type

Each organ in the body has a different gene expression profile before tumors even form, so we know that tumors that form in different organs will have differences in gene expression. Do we see different expression patterns of sex chromosome genes in tumors from different organs (tissues)?

```
# pull out expression data for cell lines that
# had a tissue type annotated
chosen_gene_tissuetype = chosen_gene_subset[colnames(chosen_gene_subset) %in%
    names(get_site)]

# transpose and log transform the data so it's
# easier to work with and put it into a data
# frame
chosen_gene_tissuetype = as.data.frame(t(log(chosen_gene_tissuetype)))

# set the name of the column to be 'expression'
# so we know what to call it when we plot
colnames(chosen_gene_tissuetype) = "expression"

# create another column called 'tissuetype' and
# fill it with tissue type of the tumor the cell
# line came from
chosen_gene_tissuetype$tissuetype = vector(length = nrow(chosen_gene_tissuetype))  # empty placeholders
for (cell_line in rownames(chosen_gene_tissuetype)) {
    chosen_gene_tissuetype[cell_line, "tissuetype"] = get_site[cell_line]
}

# use ggplot to visualize the data as a violin
# jitter plot NOTE: if you see a warning saying
# that groups with less than two data points will
# be dropped, that is fine.  This just means that
# you can make a violin out of 2 or less points

ggplot(chosen_gene_tissuetype, aes(x = tissuetype,
    y = chosen_gene_tissuetype$expression, fill = tissuetype)) +
    geom_violin(trim = FALSE) + geom_jitter(size = 0.25) +
    ylab(paste0(chosen_gene, " Expression (log counts)")) +
    theme_light() + theme(legend.position = "bottom",
    axis.text.x = element_text(size = 5, angle = 90,
        vjust = 0.5, hjust = 1))
```

```
## Warning: Groups with fewer than two data points have been dropped.
```

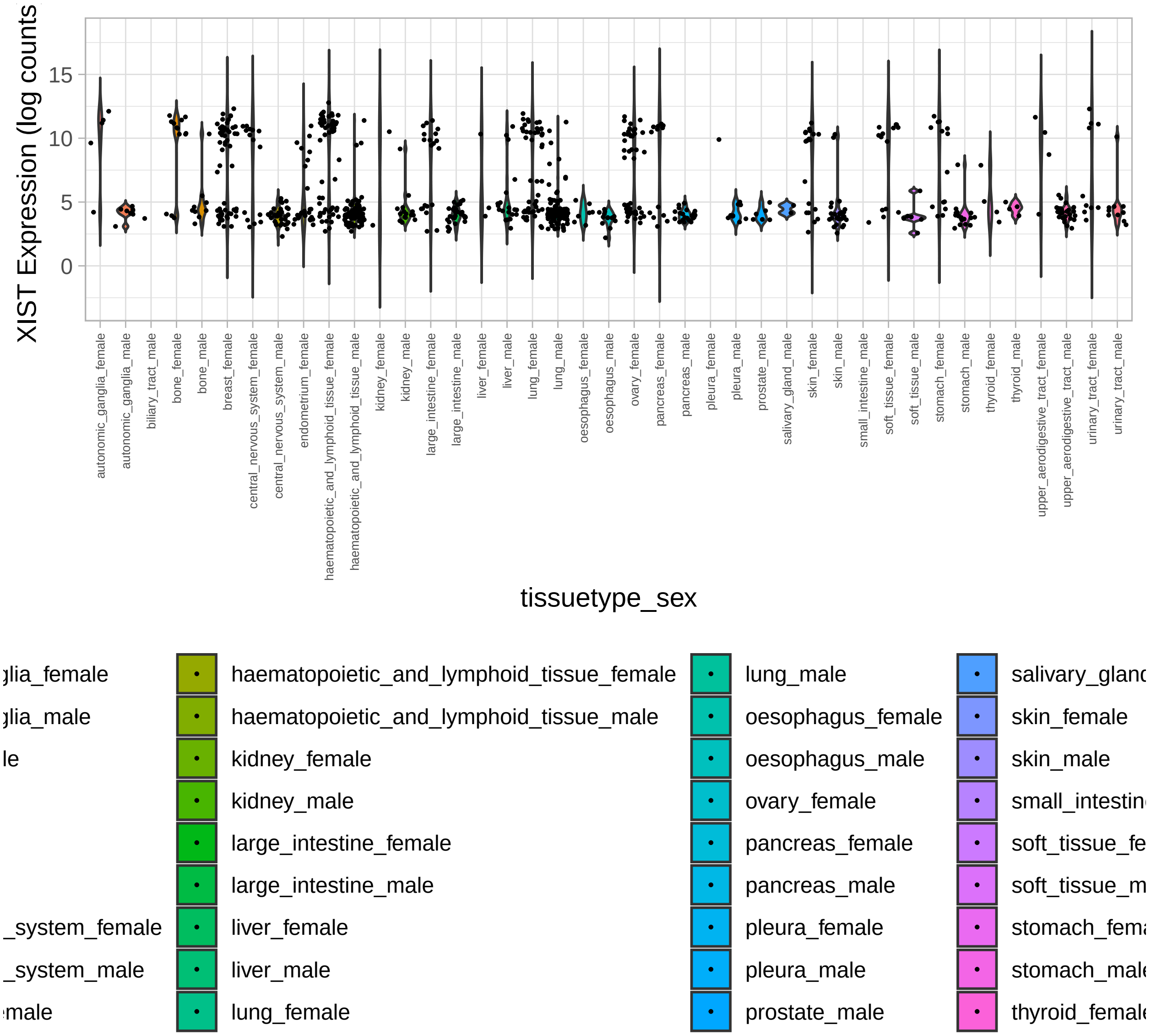


```r
# create another column called 'reported_sex' and
# fill it with the reported for the cell line
chosen_gene_tissuetype$reported_sex[rownames(chosen_gene_tissuetype) %in%
    lines_reported_female] = "female"
chosen_gene_tissuetype$reported_sex[rownames(chosen_gene_tissuetype) %in%
    lines_reported_male] = "male"

# thinking about what we saw before, there are
# probably cell lines that have a tissue type
# listed but not a reported sex

# confirm this by looking
sum(is.na(chosen_gene_tissuetype$reported_sex))
```

```
## [1] 119
```

```r
# let's make a copy of this dataframe but
# filtered for those with a reported sex
chosen_gene_tissuetype_sex = subset(chosen_gene_tissuetype,
    !is.na(chosen_gene_tissuetype$reported_sex))

# make a new column to group for pathology and
# sex
chosen_gene_tissuetype_sex$tissuetype_sex <- paste0(chosen_gene_tissuetype_sex$tissuetype,
    "_", chosen_gene_tissuetype_sex$reported_sex)

# use ggplot to visualize the data as a violin
# jitter plot
ggplot(chosen_gene_tissuetype_sex, aes(x = tissuetype_sex,
    y = chosen_gene_tissuetype_sex$expression, fill = tissuetype_sex)) +
    geom_violin(trim = FALSE) + geom_jitter(size = 0.25) +
    ylab(paste0(chosen_gene, " Expression (log counts)")) +
    theme_light() + theme(legend.position = "bottom",
    axis.text.x = element_text(size = 5, angle = 90,
        vjust = 0.5, hjust = 1))
```

```
## Warning: Groups with fewer than two data points have been dropped.
## Groups with fewer than two data points have been dropped.
## Groups with fewer than two data points have been dropped.
```



What's next?

Now that we have gone through and looked at specific features of the data with specific genes, what questions have come up? Are there any other aspects of the data you think would be interesting to look at? When you do that, how does that data visualization help you to answer your question?

If you need help with any of this, please don't hesitate to reach out by Slack. If you have a question or problem, it is likely that someone else does too, so please don't be shy. Your reaching out helps all of us learn and grow together.

List all the packages used for future reference

It is always a good practice to list out all the packages you are using in your report. It helps ensure reproducibility by you and others that use your code, and thus we report the packages and versions in the Methods section of the manuscript when you publish your work.

```
sessionInfo()
```

```
## R version 4.2.1 (2022-06-23 ucrt)
## Platform: x86_64-w64-mingw32/x64 (64-bit)
## Running under: Windows 10 x64 (build 19044)
##
## Matrix products: default
##
## locale:
## [1] LC_COLLATE=English_United States.utf8
## [2] LC_CTYPE=English_United States.utf8
## [3] LC_MONETARY=English_United States.utf8
## [4] LC_NUMERIC=C
## [5] LC_TIME=English_United States.utf8
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
## [1] UpSetR_1.4.0  ggplot2_3.4.3
##
## loaded via a namespace (and not attached):
##  [1] Rcpp_1.0.11       highr_0.10        pillar_1.9.0      compiler_4.2.1
##  [5] formatR_1.14      plyr_1.8.8        tools_4.2.1       digest_0.6.33
##  [9] evaluate_0.21     lifecycle_1.0.3   tibble_3.2.1      gtable_0.3.3
## [13] pkgconfig_2.0.3   rlang_1.1.1       cli_3.6.1         rstudioapi_0.15.0
## [17] yaml_2.3.7        xfun_0.40         fastmap_1.1.1     gridExtra_2.3
## [21] withr_2.5.0       dplyr_1.1.2       knitr_1.43        generics_0.1.3
## [25] vctrs_0.6.3       grid_4.2.1        tidyselect_1.2.0  glue_1.6.2
## [29] R6_2.5.1          fansi_1.0.4       rmarkdown_2.24    farver_2.1.1
## [33] magrittr_2.0.3    scales_1.2.1      htmltools_0.5.6   colorspace_2.1-0
## [37] labeling_0.4.2    utf8_1.2.3        munsell_0.5.0
```