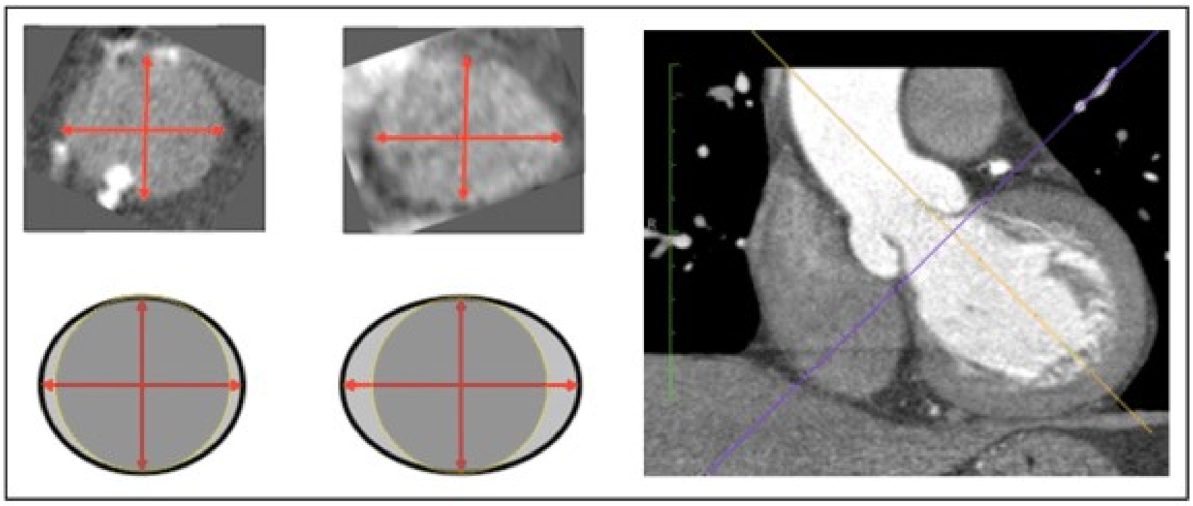
Today we have been assigned the task of identifying boundary conditions for a benchtop durability test of an implantable, artificial heart valve. In other words, we need to identify credible parameters for a physical test such that our test engineers can challenge the device under severe but realistic geometries and loads. To facilitate this task our clinical team has analyzed images and extracted pressure measurements and geometric features for a sample of n=300 patients. The rest of this post explores what we should do with these data.

For the sake of simplicity, assume that the three parameters we care about are the ***ellipticity*** of the vessel cross section, ***curvature*** of the vessel in the vessel region of interest, and the blood ***pressure***. Features such as these are important because they influence both the equilibrium geometry and the magnitude of forces acting on the implantable valve (in other words: the boundary conditions). The image below shows a schematic/example of ellipticity and vessel curvature in the LVOT and aortic valve annulus as observed in CT imaging.[1]



There are two main challenges when working with these data:

- **How do we use our sample to simulate the full population?**
- **How do we use the simulated, full population to identify groups of interest and recommend boundary conditions for the test**

Here is our dataset.[2] It turns out that each of these features can be well described by a **lognormal distribution** and we will assume that this is confirmed via prior domain knowledge. Large pressures, large ellipticities, and small radius of curvature are all bad because they would represent more extreme geometries and/or loads for the implant to resist without migrating or fracturing.

As normal, we're working in R and will lean on the tidyverse packages to accelerate things. I wasn't intending for this post to go as long as it did so I'm also offering this table of contents to show I bear no malice.

```
library(readxl)
library(knitr)
library(DiagrammeR)
library(fitdistrplus)
library(MASS)
library(ggrepel)
library(readxl)
library(ks)
library(broom)
library(ggExtra)
library(GGally)
library(car)
library(rgl)
library(anySim)
library(tidyverse)
```

Start by reading in the data and taking a look at the format.

```
sample_data <- readRDS(file = "sim_anatomy_data.rds")
sample_data
## # A tibble: 300 x 3
##    ellip  curv pressure
##                        
##  1  1.26  4.51     92.7
##  2  1.28  5.02    183.
##  3  1.29  4.03    154.
##  4  1.23  2.14    109.
##  5  1.13  3.67    124.
##  6  1.22  2.37    114.
##  7  1.10  3.06    113.
##  8  1.04  2.31    105.
##  9  1.11  5.31    115.
## 10  1.09  2.04    109.
## # ... with 290 more rows
```

As expected, 300 rows with our 3 features of interest.

Key Point:

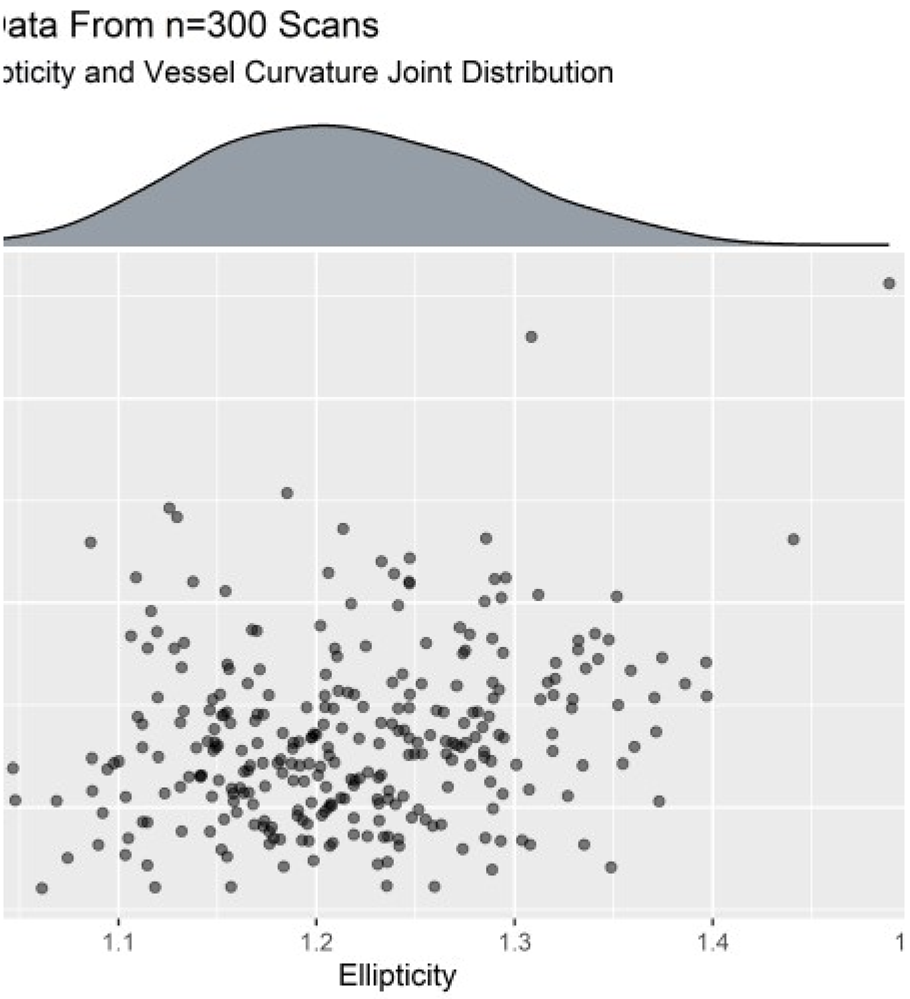
> **Since we are asked to find a worst-case set of values, it might seem reasonable or tempting to extract the maximum value from**

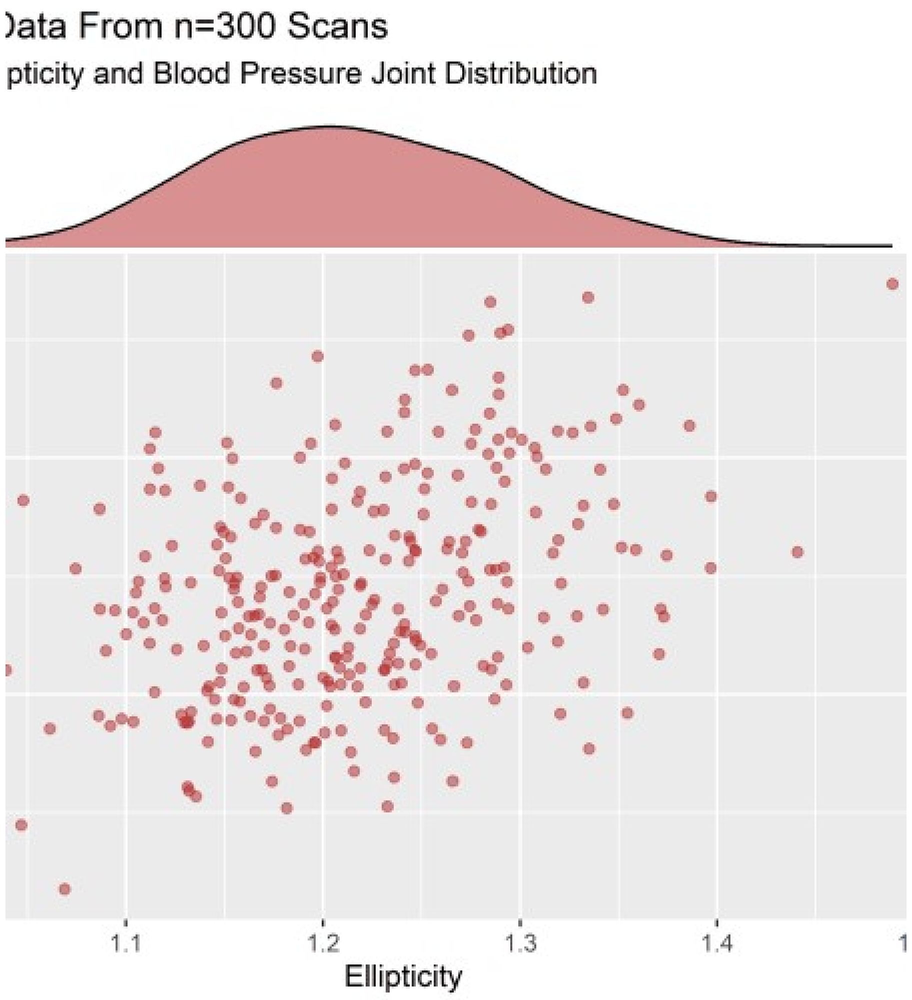
each group (or maybe something like the 95th percentile) and report those values together as a conservative worst-case for ellipticity, curvature, and pressure. Our test engineers would then set up a benchtop test to challenge our prototype devices in those same conditions to see if they survive. The problem with this approach is that each row of data is from a specific patient, so the variables may be correlated. It could be that those severe, 95th percentile values for each variable never occur together in the same patient. If we choose them together, we would over-test the device and over-design the device, potentially setting the program way behind. We must instead look at the data as a joint distribution and investigate correlations and covariance among the variables.

Let’s verify the shape of the distributions for each feature and see if there is any correlation between the variables:

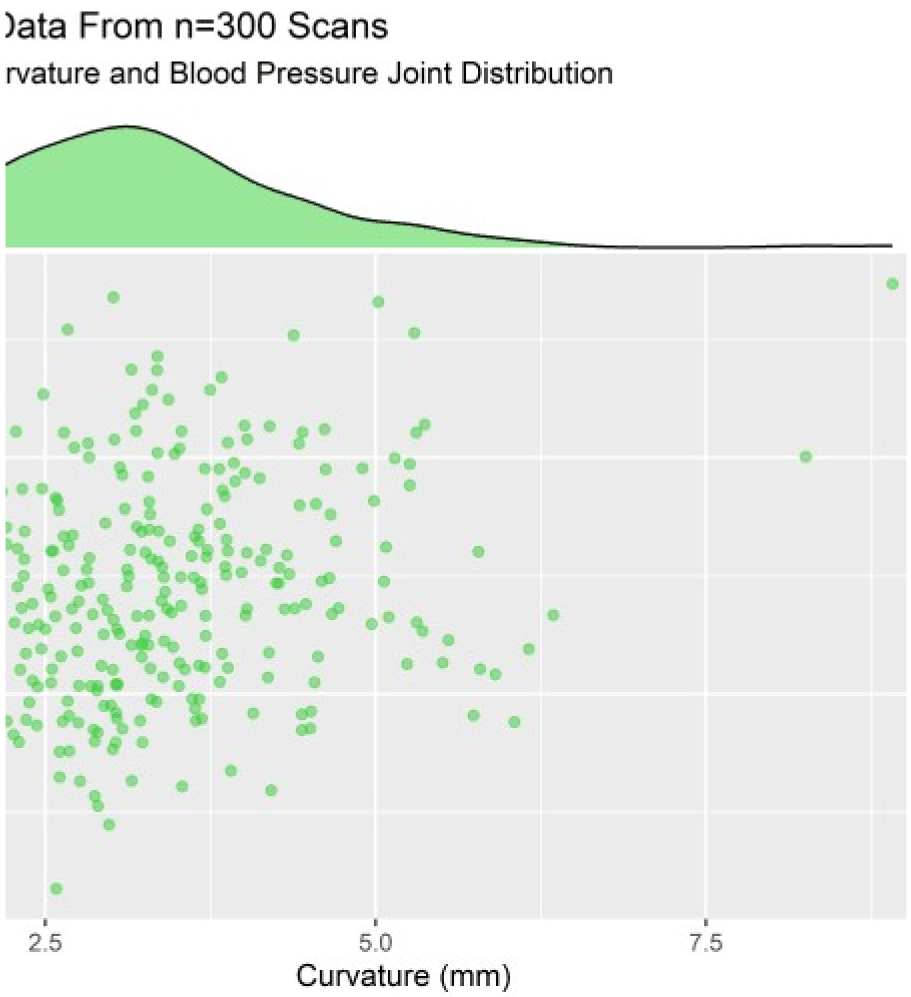
```
ellip_curv_plt <- sample_data %>%
  ggplot(aes(x = ellip, y = curv)) +
  geom_point(alpha = .5) +
  labs(
    title = "Patient Data From n=300 Scans",
    subtitle = "Vessel Ellipticity and Vessel Curvature Joint
Distribution",
    x = "Ellipticity",
    y = "Curvature (mm)"
  )

ellip_pressure_plt <- sample_data %>%
  ggplot(aes(x = ellip, y = pressure)) +
  geom_point(alpha = .5, color = "firebrick") +
  labs(
    title = "Patient Data From n=300 Scans",
    subtitle = "Vessel Ellipticity and Blood Pressure Joint
Distribution",
    x = "Ellipticity",
    y = "Pressure (mm Hg)"
  )

curv_pressure_plt <- sample_data %>%
  ggplot(aes(x = curv, y = pressure)) +
  geom_point(alpha = .5, color = "limegreen") +
  labs(
    title = "Patient Data From n=300 Scans",
    subtitle = "Vessel Curvature and Blood Pressure Joint
Distribution",
    x = "Curvature (mm)",
    y = "Pressure (mm Hg"
  )

ellip_curv_mplt <- ggExtra::ggMarginal(ellip_curv_plt, type =
"density", fill = "#2c3e50", alpha = .5)
```

```
ellip_pressure_mplt <- ggExtra::ggMarginal(ellip_pressure_plt, type =
"density", fill = "firebrick", alpha = .5)
curv_pressure_mplt <- ggExtra::ggMarginal(curv_pressure_plt, type =
"density", fill = "limegreen", alpha = .5)
```





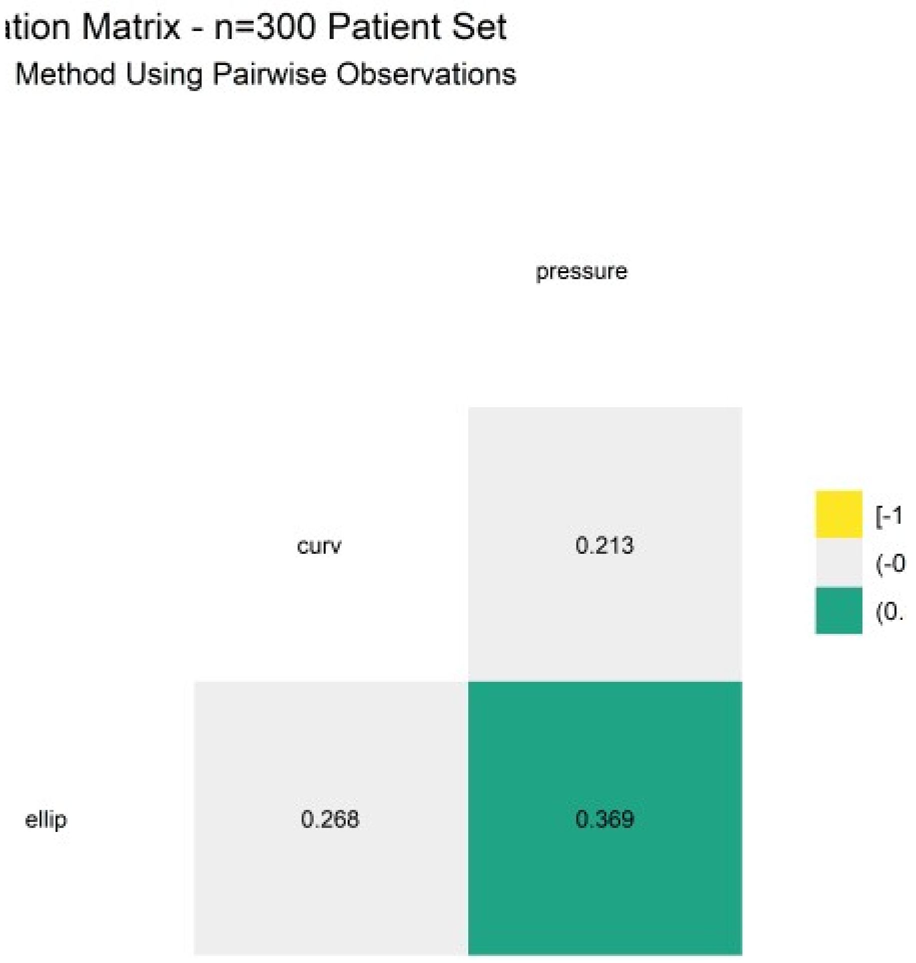
)ata From n=300 Scans
pticity and Blood Pressure Joint Distribution
1.1
1.2
1.3
1.4
1
Ellipticity



Correlations in the Original Dataset

ggcorr() from the GGally package is very convenient for visualizing correlations.

```
sample_data %>% ggcorr(
  high = "#20a486ff",
  low = "#fde725ff",
  label = TRUE,
  hjust = .75,
  size = 3,
  label_size = 3,
  label_round = 3,
  nbreaks = 3
) +
  labs(
    title = "Correlation Matrix - n=300 Patient Set",
    subtitle = "Pearson Method Using Pairwise Observations"
  )
```



We see that there are some positive correlations in this dataset.

To build out the sample into a simulated population we will fit a MLE estimate and use the model to push out a lot of predictions. If each variable was independent the job would be easy - just execute a few rlnorm()s and bind them together. The job is more challenging when the variables are correlated because they must be simulated all at once. I will show 2 approaches in the sections below. Note that the 2nd approach is more efficient but it helped me to walk through the first one to understand the workflow. If you are impatient I would skip to the section on approach 2.

Approach 1 - Manually Transform Everything to Normal

For cases where each variable is normal or can be easily transformed there is a straightforward and relatively simple workflow to generate simulated joint distribution using the **mvrnorm()** function from the MASS package:

In our case, we can easily convert our data from lognormal to normal and will then be able to use the mvrnorm() function to draw from a multivariate normal distribution and then undo the transformation later to recover a simulated population with desired correlations (as shown above). From there we can identify patients of interest, whether they be extreme challenging cases or a central, common group.

Per the workflow above, start by fitting the native data to lognormal distributions using fitdist() and extract the parameters. Storing all the parameters as objects is a bit tedious and I only do it here so we can make a nice summary table of everything at the end.

Step 1 - Fit Distributions to Each Variable

```
ellip_fit <- fitdist(sample_data$ellip, "lnorm")
curv_fit <- fitdist(sample_data$curv, "lnorm")
pressure_fit <- fitdist(sample_data$pressure, "lnorm")

# store lognormal parameters of original data
ellip_meanlog <- ellip_fit$estimate[["meanlog"]]
ellip_sdlog <- ellip_fit$estimate[["sdlog"]]
curv_meanlog <- curv_fit$estimate[["meanlog"]]
curv_sdlog <- curv_fit$estimate[["sdlog"]]
pressure_meanlog <- pressure_fit$estimate[["meanlog"]]
pressure_sdlog <- pressure_fit$estimate[["sdlog"]]

# store correlations in original data
cor_ec <- cor(x = sample_data$ellip, y = sample_data$curv)
cor_ep <- cor(x = sample_data$ellip, y = sample_data$pressure)
cor_cp <- cor(x = sample_data$curv, y = sample_data$pressure)

# store covariances in original data
cov_ellip_curv <- cov(x = sample_data$ellip, y = sample_data$curv)
cov_ellip_ellip <- cov(x = sample_data$ellip, y = sample_data$ellip)
cov_curv_curv <- cov(x = sample_data$curv, y = sample_data$curv)
cov_ellip_pressure <- cov(x = sample_data$ellip, y =
sample_data$pressure)
cov_pressure_pressure <- cov(x = sample_data$pressure, y =
sample_data$pressure)
cov_curv_pressure <- cov(x = sample_data$curv, y =
sample_data$pressure)

# summarize the parameters and reshape a bit
original_data_param_tbl <- tibble(
  ellip_meanlog = ellip_meanlog,
  ellip_sdlog = ellip_sdlog,
  curv_meanlog = curv_meanlog,
  curv_sdlog = curv_sdlog,
  pressure_meanlog = pressure_meanlog,
  pressure_sdlog = pressure_sdlog,
  ellip_curv_correlation = cor_ec,
  ellip_pressure_correlation = cor_ep,
  curv_pressure_correlation = cor_cp,
  ellip_ellip_covariance = cov_ellip_ellip,
  ellip_curv_covariance = cov_ellip_curv,
  curv_curv_covariance = cov_curv_curv,
  ellip_pressure_covariance = cov_ellip_pressure,
  pressure_pressure_covariance = cov_pressure_pressure,
  curv_pressure_covariance = cov_curv_pressure
) %>%
  pivot_longer(cols = everything(), names_to = "feature", values_to =
"value") %>%
  mutate(dataset = "original_data") %>%
  mutate_if(is.character, as_factor)

# View summary table of original data
original_data_param_tbl %>%
  kable(align = "c", digits = 3)
```

feature	value	dataset
ellip_meanlog	0.193	original_data
ellip_sdlog	0.064	original_data
curv_meanlog	1.158	original_data
curv_sdlog	0.309	original_data
pressure_meanlog	4.783	original_data
pressure_sdlog	0.191	original_data
ellip_curv_correlation	0.268	original_data
ellip_pressure_correlation	0.369	original_data
curv_pressure_correlation	0.213	original_data
ellip_ellip_covariance	0.006	original_data
ellip_curv_covariance	0.022	original_data
curv_curv_covariance	1.157	original_data
ellip_pressure_covariance	0.659	original_data
pressure_pressure_covariance	530.683	original_data
curv_pressure_covariance	5.285	original_data

Step 2 - Transform all variables to normal

A simple log operation brings the lognormal variable to normal.

```
# transform original, lognormal data to normal
normal_sample_data <- sample_data %>%
  mutate(
    n_ellip = log(ellip),
    n_curv = log(curv),
    n_pressure = log(pressure)
  )

normal_sample_data %>%
  head() %>%
  kable(align = "c", digits = 3)
```

ellip	curv	pressure	n_ellip	n_curv	n_pressure
1.255	4.506	92.739	0.228	1.505	4.530
1.285	5.019	182.970	0.251	1.613	5.209
1.289	4.027	153.858	0.254	1.393	5.036
1.234	2.139	108.669	0.210	0.760	4.688
1.133	3.673	123.633	0.125	1.301	4.817
1.219	2.373	113.944	0.198	0.864	4.736

Step 3 - Fit normal distributions to each transformed variable

We don't actually have to formally fit normal distributions since it is convenient to obtain the mean and standard deviation at any time using the mean() or sd() functions. But we will extract and store correlations and covariances for the simulation to come.

```
# get correlations of transformed, normal data
ncor_ec <- cor(
  x = normal_sample_data$n_ellip,
  normal_sample_data$n_curv
)
ncor_ep <- cor(
  x = normal_sample_data$n_ellip,
  normal_sample_data$n_pressure
)
ncor_cp <- cor(
  x = normal_sample_data$n_curv,
  normal_sample_data$n_pressure
)

# get covariance of transformed, normal data
n_cov_ellip_curv <- cov(
  x = normal_sample_data$n_ellip,
  y = normal_sample_data$n_curv
)
n_cov_ellip_ellip <- cov(
  x = normal_sample_data$n_ellip,
  y = normal_sample_data$n_ellip
)
n_cov_curv_curv <- cov(
  x = normal_sample_data$n_curv,
  y = normal_sample_data$n_curv
)

n_cov_ellip_pressure <- cov(
  x = normal_sample_data$n_ellip,
  y = normal_sample_data$n_pressure
)
n_cov_pressure_pressure <- cov(
  x = normal_sample_data$n_pressure,
  y = normal_sample_data$n_pressure
)
n_cov_curv_pressure <- cov(
  x = normal_sample_data$n_curv,
  y = normal_sample_data$n_pressure
)
```

Step 4 - Draw joint distribution using mvrnorm() or equivalent function

Time to actually draw the correlated values. I store them here in an object called mult_norm.

```
# draw from multivariate normal with parameters from transformed normal
distributions and correlation
set.seed(0118)

mult_norm <- as_tibble(MASS::mvrnorm(
  10000, c(
    mean(normal_sample_data$n_ellip),
    mean(normal_sample_data$n_curv),
    mean(normal_sample_data$n_pressure)
  ),
  matrix(c(
    n_cov_ellip_ellip,
    n_cov_ellip_curv,
    n_cov_ellip_pressure,
    n_cov_ellip_curv,
    n_cov_curv_curv,
    n_cov_curv_pressure,
    n_cov_ellip_pressure,
    n_cov_curv_pressure,
    n_cov_pressure_pressure
  ), 3, 3)
)) %>%
  rename(
    n_ellip_sim = V1,
    n_curv_sim = V2,
    n_pressure_sim = V3
  )
```

Step 5 - Back-transform simulated data to original distribution

Exponentiating the data brings it back to lognormal.

```
# convert back to lognormal
log_norm <- mult_norm %>%
  mutate(
    ellip_sim = exp(n_ellip_sim),
    curv_sim = exp(n_curv_sim),
    pressure_sim = exp(n_pressure_sim)
  )

log_norm %>%
  head() %>%
  kable(align = "c", digits = 3)
```

n_ellip_sim	n_curv_sim	n_pressure_sim	ellip_sim	curv_sim	pressure_sim
0.254	1.600	5.248	1.290	4.952	190.266
0.233	1.038	5.107	1.262	2.823	165.178
0.236	1.152	4.812	1.266	3.165	123.018
0.313	1.003	5.048	1.368	2.727	155.636
0.224	1.622	5.192	1.251	5.066	179.912
0.197	1.486	4.822	1.218	4.422	124.185

Step 6 - Evaluate parameters and marginal distributions of the back-transfomed data

```
# evaluate the marginal distributions of the simulated data
ellip_sim_fit <- fitdistrplus::fitdist(log_norm$ellip_sim, "lnorm")
curv_sim_fit <- fitdistrplus::fitdist(log_norm$curv_sim, "lnorm")
pressure_sim_fit <- fitdistrplus::fitdist(log_norm$pressure_sim,
"lnorm")
```

Obtain and store the correlation, covariances, and parameters of simulated set:

```
# get correlation and covariances of simulated data
sim_cor_ec <- cor(x = log_norm$ellip_sim, log_norm$curv_sim)
sim_cor_ep <- cor(x = log_norm$ellip_sim, log_norm$pressure_sim)
sim_cor_cp <- cor(x = log_norm$curv_sim, log_norm$pressure_sim)

sim_cov_ellip_curv <- cov(x = log_norm$ellip_sim, y =
log_norm$curv_sim)
sim_cov_ellip_ellip <- cov(x = log_norm$ellip_sim, y =
log_norm$ellip_sim)
sim_cov_curv_curv <- cov(x = log_norm$curv_sim, y = log_norm$curv_sim)

sim_cov_ellip_pressure <- cov(x = log_norm$ellip_sim, y =
log_norm$pressure_sim)
sim_cov_pressure_pressure <- cov(x = log_norm$pressure_sim, y =
log_norm$pressure_sim)
sim_cov_curv_pressure <- cov(x = log_norm$curv_sim, y =
log_norm$pressure_sim)

# store parameters of simulated data
ellip_sim_meanlog <- ellip_sim_fit$estimate[["meanlog"]]
ellip_sim_sdlog <- ellip_sim_fit$estimate[["sdlog"]]
curv_sim_meanlog <- curv_sim_fit$estimate[["meanlog"]]
curv_sim_sdlog <- curv_sim_fit$estimate[["sdlog"]]
pressure_sim_meanlog <- pressure_sim_fit$estimate[["meanlog"]]
pressure_sim_sdlog <- pressure_sim_fit$estimate[["sdlog"]]

# collect parameters from simulated data
sim_data_param_tbl <- tibble(
  ellip_meanlog = ellip_sim_meanlog,
  ellip_sdlog = ellip_sim_sdlog,
  curv_meanlog = curv_sim_meanlog,
  curv_sdlog = curv_sim_sdlog,
  pressure_meanlog = pressure_sim_meanlog,
  pressure_sdlog = pressure_sim_sdlog,

  ellip_curv_correlation = sim_cor_ec,
  ellip_pressure_correlation = sim_cor_ep,
  curv_pressure_correlation = sim_cor_cp,

  ellip_curv_covariance = sim_cov_ellip_curv,
  ellip_ellip_covariance = sim_cov_ellip_ellip,
  curv_curv_covariance = sim_cov_curv_curv,

  ellip_pressure_covariance = sim_cov_ellip_pressure,
  pressure_pressure_covariance = sim_cov_pressure_pressure,
  curv_pressure_covariance = sim_cov_curv_pressure
) %>%
  pivot_longer(cols = everything(), names_to = "feature", values_to =
"value") %>%
  mutate(dataset = "simulated_data") %>%
  mutate_if(is.character, as_factor)

sim_data_param_tbl %>%
  kable(align = "c")
```

feature	value	dataset
ellip_meanlog	0.1932042	simulated_data
ellip_sdlog	0.0630117	simulated_data
curv_meanlog	1.1626798	simulated_data
curv_sdlog	0.3092643	simulated_data
pressure_meanlog	4.7878497	simulated_data
pressure_sdlog	0.1900026	simulated_data
ellip_curv_correlation	0.2505145	simulated_data
ellip_pressure_correlation	0.3644292	simulated_data
curv_pressure_correlation	0.1956149	simulated_data
ellip_curv_covariance	0.0203344	simulated_data

feature	value	dataset
ellip_ellip_covariance	0.0058779	simulated_data
curv_curv_covariance	1.1209300	simulated_data
ellip_pressure_covariance	0.6534943	simulated_data
pressure_pressure_covariance	547.0647415	simulated_data
curv_pressure_covariance	4.8440727	simulated_data

Compare Original Data to Simulated Data

A bit more wrangling let's us compare the feature of the original dataset to the new, simulated population to see if they agree.

```
compare_tbl <- bind_rows(original_data_param_tbl, sim_data_param_tbl)
%>%
  pivot_wider(id_cols = everything(), names_from = dataset)

compare_tbl %>%
  kable(align = "c", digits = 3)
```

feature	original_data	simulated_data
ellip_meanlog	0.193	0.193
ellip_sdlog	0.064	0.063
curv_meanlog	1.158	1.163
curv_sdlog	0.309	0.309
pressure_meanlog	4.783	4.788
pressure_sdlog	0.191	0.190
ellip_curv_correlation	0.268	0.251
ellip_pressure_correlation	0.369	0.364
curv_pressure_correlation	0.213	0.196
ellip_ellip_covariance	0.006	0.006
ellip_curv_covariance	0.022	0.020
curv_curv_covariance	1.157	1.121
ellip_pressure_covariance	0.659	0.653
pressure_pressure_covariance	530.683	547.065
curv_pressure_covariance	5.285	4.844

Everything appears to align well, but this sure took a while. Wouldn't it be nice if there was a faster way than to manually fit and extract values from mvrnorm() ? Fortunately, we're in the R ecosystem, where somebody smart has usually tackled the problem and provided the tools to the community. The tool that I had success with is the AnySim package.[3]

Here's how to perform the simulation in a much more efficient way. Note: this is how I created the original dataset of 300 that we've been working with.

Approach 2 - AnySim

The AnySim workflow:

It may look similar in the flowchart but in practice its way easier than Method 1. Here’s how it works in only a few lines of code:

```
set.seed(13)
# Define the target distribution functions (ICDFs) of each random
variable.

ellip_dist <- "qlnorm"
curv_dist <- "qlnorm"
pressure_dist <- "qlnorm"

# store the 3 ICDFs in a vector
dist_vec <- c(ellip_dist, curv_dist, pressure_dist)

# Define the parameters of the target distribution functions - store
them in a list
ellip_params <- list(meanlog = 0.20, sdlog = .067)
curv_params <- list(meanlog = 1.15, sdlog = 0.3)
pressure_params <- list(meanlog = 4.80, sdlog = 0.2)

# this is a weird way to do it but I'm following along with an example
from AnySim vignette :)
params_list <- list(NULL)
params_list[[1]] <- ellip_params
params_list[[2]] <- curv_params
params_list[[3]] <- pressure_params
```

```
# Define the target correlation matrix.
corr_matrix <- matrix(c(
  1, 0.21, 0.4,
  0.21, 1, .21,
  0.4, 0.21, 1
),
ncol = 3,
nrow = 3,
byrow = T
)
# Estimate the parameters of the auxiliary Gaussian model.
aux_gaussion_param_tbl <- EstCorrRVs(
  R = corr_matrix, dist = dist_vec, params = params_list,
  NatafIntMethod = "GH", NoEval = 9, polydeg = 8
)


# Generate 10000 synthetic realizations of the 3 correlated RVs.
correlated_ln_draws_tbl <- as_tibble(SimCorrRVs(n = 10000, paramsRVs =
aux_gaussion_param_tbl)) %>%
  rename(
    ellip = V1,
    curv = V2,
    pressure = V3
  )

correlated_ln_draws_tbl %>%
  head(10) %>%
  kable(align = "c")
```

ellip	curv	pressure
1.267618	2.158739	133.3040
1.198681	4.437176	127.1982
1.375663	2.649938	160.2052
1.236829	3.621202	141.5427
1.318572	2.541340	129.3427
1.255885	3.236722	146.9361
1.326279	3.387357	167.8579
1.240926	4.254830	145.7207
1.191865	1.853877	87.7240
1.315273	4.144667	90.5485

Let’s see if we were able to produce the desired relationships between the variables:

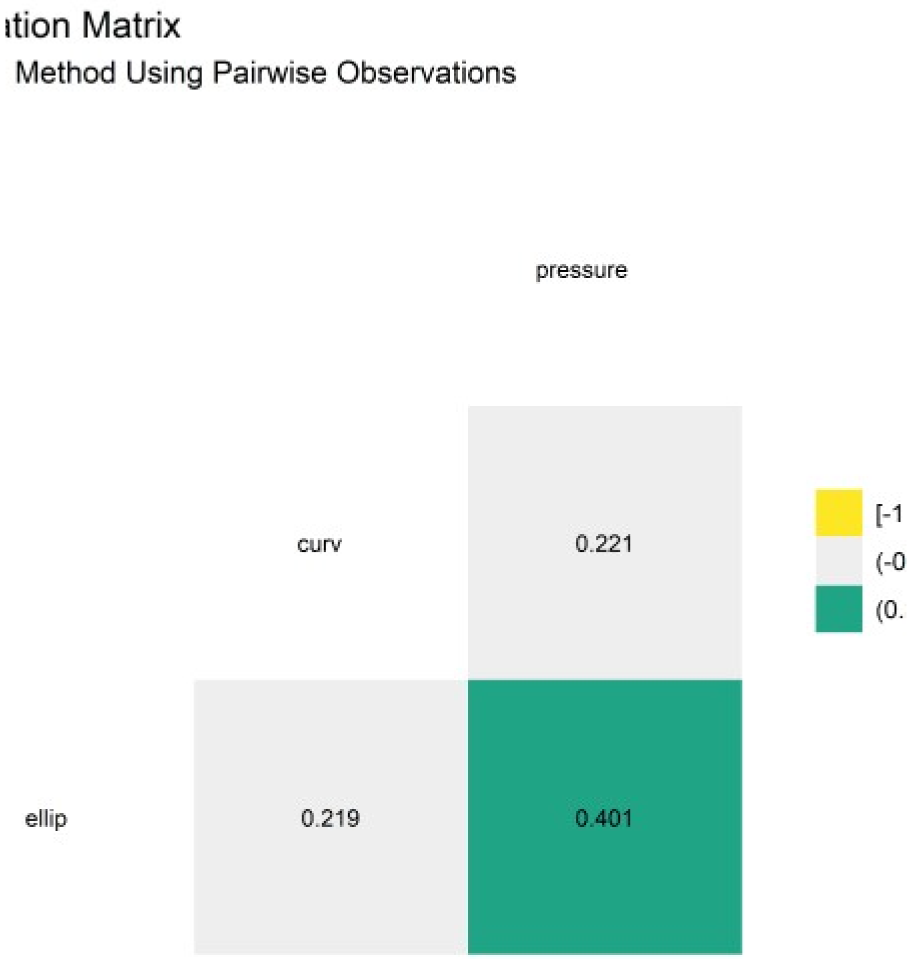
```
# Fit synthetic data
ellip_ln_fit <- tidy(fitdistr(correlated_ln_draws_tbl$ellip, "log-
normal")) %>% mutate(
  var = "ellipticity",
  dataset = "sim_draws"
)
```

```
curv_ln_fit <- tidy(fitdistr(correlated_ln_draws_tbl$curv, "log-
normal")) %>% mutate(
  var = "curvature",
  dataset = "sim_draws"
)
pressure_ln_fit <- tidy(fitdistr(correlated_ln_draws_tbl$pressure,
"log-normal")) %>% mutate(
  var = "pressure",
  dataset = "sim_draws"
)
recovered_params_tbl <- bind_rows(ellip_ln_fit, curv_ln_fit,
pressure_ln_fit)

# Show target values
target_tbl <- tibble(
  term = rep(c("meanlog", "sdlog"), 3),
  estimate = c(.20, .067, 1.15, .3, 4.80, .20),
  var = c("ellipticity", "ellipticity", "curvature", "curvature",
"pressure", "pressure"),
  dataset = "target values"
)

# Compare simulated values to target values
bind_rows(recovered_params_tbl, target_tbl) %>%
  select(-std.error) %>%
  pivot_wider(id_cols = everything(), names_from = dataset, values_from
= estimate) %>%
  mutate_if(is.numeric, round, 2) %>%
  kable(align = rep("c"))
```

term	var	sim_draws	target values
meanlog	ellipticity	0.20	0.20
sdlog	ellipticity	0.07	0.07
meanlog	curvature	1.15	1.15
sdlog	curvature	0.30	0.30
meanlog	pressure	4.80	4.80
sdlog	pressure	0.20	0.20

Excellent agreement. Now check to see if the correlations were preserved:

```
# Check correlations
correlated_ln_draws_tbl %>% ggcorr(
  high = "#20a486ff",
  low = "#fde725ff",
  label = TRUE,
  hjust = .75,
  size = 3,
  label_size = 3,
  label_round = 3,
  nbreaks = 3
) +
  labs(
    title = "Correlation Matrix",
    subtitle = "Pearson Method Using Pairwise Observations"
  )
```



Again, perfect agreement to 2 decimal places. Thank you AnySim!

Now that I've shown 2 ways to preserve the correlation structure in a simulation, we can return to the original question: what are the worst-case (or most common) values of this (simulated) population?[4]

Using the Simulated Population to Define Desired Test Conditions and Groups of Interest

The simulated population is useful because its density properties provide a means to determine how extreme any values of interest are. The trick is that we have to define the boundary of interest based on the question we are trying to answer. Two common questions we have pertain to finding the most common set of patients, or the most extreme patients relative to some point or region of interest. I discuss both below.

Identify a percentage of worst-case patients relative to some value of interest

Consider a 2d test case where we are interested in the joint distribution of 2 variables: curvature and pressure. These would be appropriate for something like a migration test, where the pressure wants to pull the implant out of place a curved configuration is worse than straight (like a pipe elbow).

If we know that the worst-case conditions that are physiologically relevant occur when the curvature is 1 mm (radius-of-curvature) and the pressure is 300 mm Hg (extremely high). How could we identify the worst-case 5% of patients using our simulated population data from above? In this case we don’t need an algorithm, just a bit of geometry.

We can leverage the standard geometric formula for distance between two points:

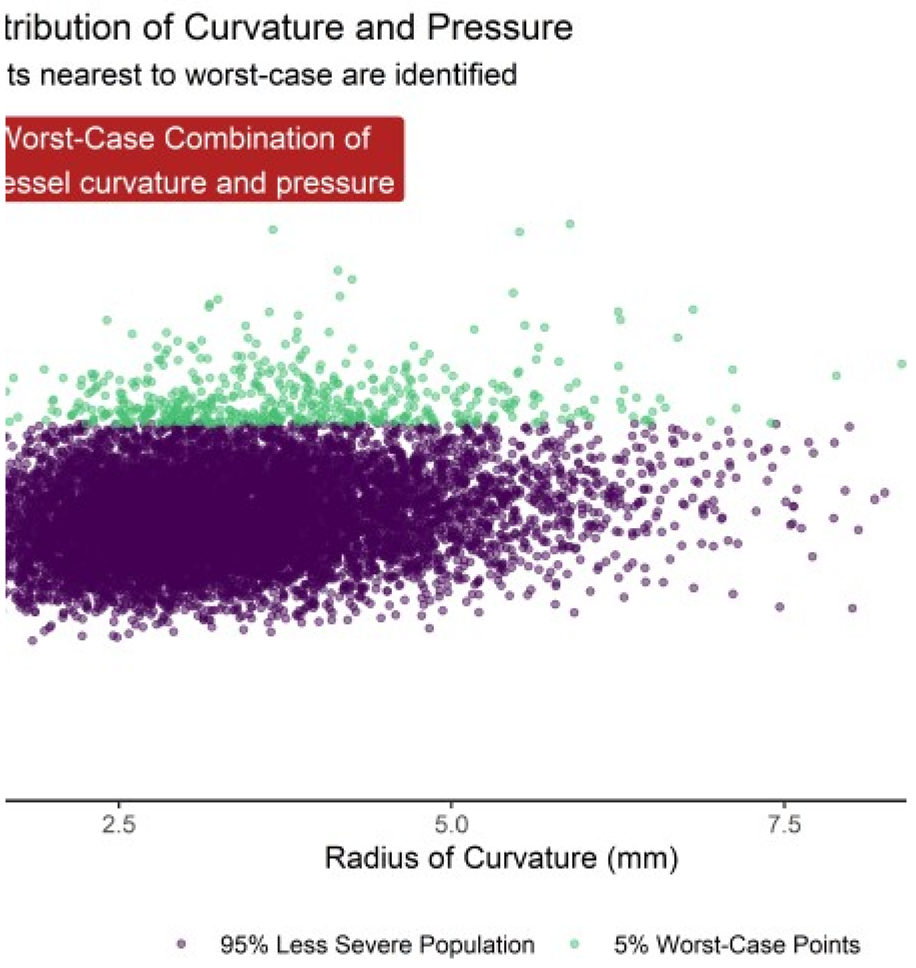
$${\displaystyle d(p,q)={\sqrt {(p_{1}-q_{1})^{2}+(p_{2}-q_{2})^{2}}}}$$

Here’s how to calculate this number for each value in the joint distribution for curvature and pressure:

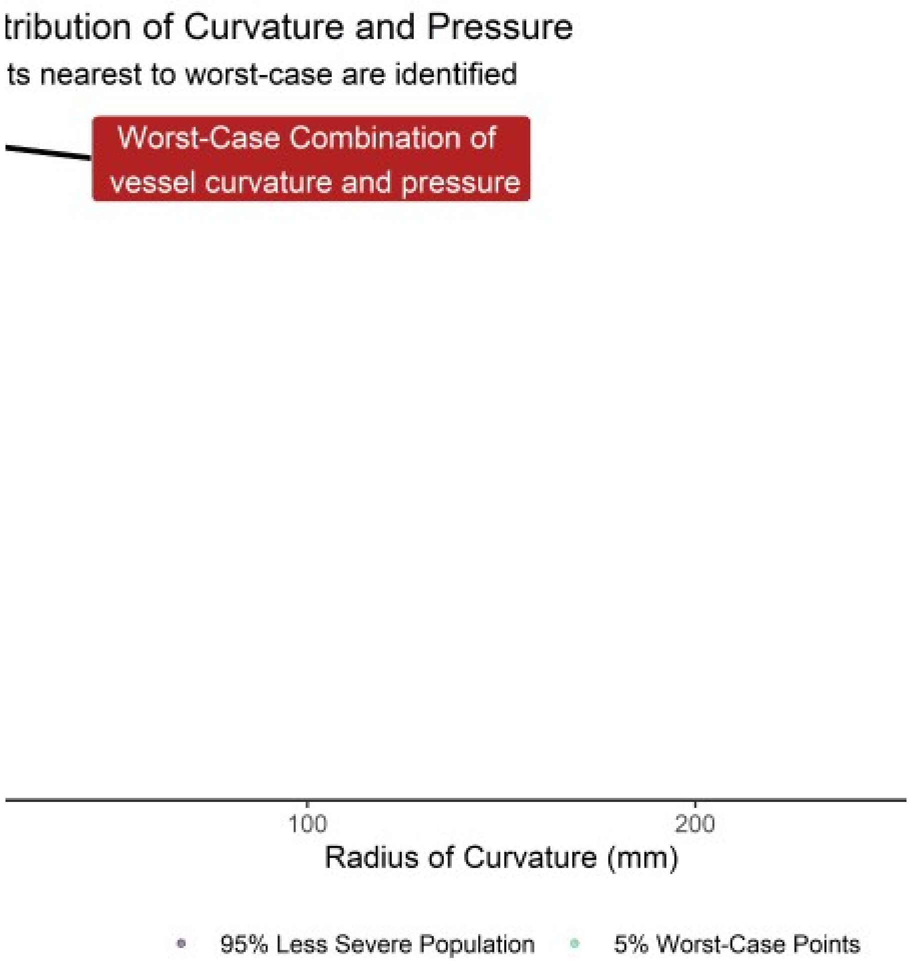
```
# specify theoretical worst-case point in space
theoretical_worst_curv <- 1
theoretical_worst_pressure <- 300

# calculated each point's distance from the theoretical worst-case
d_data <- log_norm %>%
  rowwise() %>%
  mutate(d = ((theoretical_worst_curv - curv_sim)^2 +
(theoretical_worst_pressure - pressure_sim)^2)^.5) %>%
  arrange(desc(d)) %>%
  ungroup()

# make ecdf to map points to percentiles of empirical distribution
d_ECDF_fcn <- ecdf(d_data$d)

# map the ecdf over all the calculated distances to convert them to
percentiles
d_pct_data <- d_data %>%
  mutate(d_pct = map_dbl(d, d_ECDF_fcn)) %>%
  mutate(in_out = case_when(
    d_pct <= .05 ~ "5% Worst-Case Points",
    TRUE ~ "95% Less Severe Population"
  )) %>%
  mutate(in_out = as_factor(in_out))

wc_tbl <- tibble(x = 1, y = 300)
```

Now visualize.

```
# visualize
d_plt <- d_pct_data %>%
  ggplot(aes(x = curv_sim, y = pressure_sim, color = in_out)) +
  geom_point(alpha = .5, size = 1) +
  geom_point(aes(x = 1, y = 300), color = "firebrick") +
  geom_label_repel(
    data = wc_tbl, aes(x, y),
    label = "Worst-Case Combination of \n vessel curvature and
pressure",
    fill = "firebrick",
    color = "white",
    segment.color = "black",
    segment.size = 1,
    #                    min.segment.length = unit(1, "lines"),
    nudge_y = 50,
    nudge_x = 2
  ) +
  labs(
    title = "Joint Distribution of Curvature and Pressure",
    subtitle = "5% of points nearest to worst-case are identified",
    x = "Radius of Curvature (mm)",
    y = "Blood Presure (mm Hg)"
  ) +
  theme_classic() +
  ylim(c(0, 300)) +
  theme(
    legend.position = "bottom",
    legend.title = element_blank()
  ) +
  scale_color_viridis_d(option = "D", end = .7)

d_plt
```



It’s a bit of an illusion since the axes aren’t scaled 1:1. Here’s a look at a scaled version showing the teal points closest to the point of interest.



Neat! We just split the data into the most severe 5% and least severe 95% based on a known point we want to avoid. You could imagine a similar calculation if we had a line that defined a worst-case boundary like we use for the failure threshold in a Goodman Diagram.[5]

In other cases we just want to know the most common values (values in the region of highest probability) based on the multi-dimensional density calculation. In other words, we want to identify contours based on how close the points are to each other, not some arbitrary point in space or line. A kernel density estimate is going to be our tool of choice in all but the most basic cases. The KDE is non-parametric and can accommodate very complex joint distributions and density shapes.[6]

Typically, the default contours on a density plot show an output called "level" but they can be converted to denote upper percentages of highest density regions. This is the method I show below.[7]

Identify probability contours using ks package and kernel density estimation

This first chunk converts the data and generated the KDE. The bandwidth parameters controls the "smoothness" or granularity of the estimate and can be hard to specify in multiple dimensions. Hscv() provides a method of determining a reasonable bandwidth through cross-validation; see documentation in footnotes for more information if interested.

```
# convert simulated data tibble to matrix
d3m <- correlated_ln_draws_tbl %>%
  as.matrix()
```

```
# cross-validated bandwidth for kd (takes a while to calculate)
# hscv1 <- Hscv(correlated_ln_draws_tbl)
# hscv1 %>% write_rds(here::here("hscv1.rds"))

hscv1 <- read_rds(here::here("hscv1.rds"))

# generate kernel density estimate from simulated population
kd_d3m <- ks::kde(d3m, H = hscv1, compute.cont = TRUE)
```

Density percentiles from kde estimate

The KDE estimate provides density percentiles that can be used to generate the contours the define the density regions in multiple dimensions.

```
# see the kde's calculated density thresholds for specified proportions
cont_vals_tbl <- tidy(kd_d3m$cont) %>%
  mutate(n_row = row_number()) %>%
  mutate(probs = 100 - n_row) %>%
  select(probs, x)

reference_grid_probs_tbl <- cont_vals_tbl %>%
  rename(estimate = x)

reference_grid_probs_tbl %>%
  head(10) %>%
  kable(align = rep("c"))
```

probs	estimate
99	0.0332949
98	0.0321061
97	0.0310773
96	0.0303446
95	0.0295937
94	0.0288550
93	0.0282270
92	0.0276995
91	0.0271418
90	0.0266409

```
reference_grid_probs_tbl %>%
  tail(10) %>%
  kable(align = rep("c"))
```

probs	estimate
10	0.0018141
9	0.0016195
8	0.0014430
7	0.0012654

probs	estimate
6	0.0010786
5	0.0008795
4	0.0007005
3	0.0005288
2	0.0003976
1	0.0002657

KDE estimates in the range of the variables

By default the KDE provides density estimates for a grid of points that covers the space of the variables.

```
kd_grid_estimates <- kd_d3m
```

If we want to know the value at each point in the simulated population we use the eval.points argument.

```
mc_estimates <- ks::kde(
  x = d3m, H = hscv1,
  compute.cont = TRUE,
  eval.points = correlated_ln_draws_tbl %>% as.matrix()
)
```

Here are a couple different ways to convert the kde object features into a tibble:

```
mc_est_tbl_10000 <- tibble(estimate = mc_estimates$estimate) %>%
  bind_cols(correlated_ln_draws_tbl)
kd_grid_est_tbl_29k <- broom:::tidy.kde(kd_grid_estimates) %>%
  pivot_wider(names_from = variable, values_from = value) %>%
  rename(ellip = x1, curv = x2, pressure = x3) %>%
  select(-obs)
mc_est_tbl_10000 %>%
  head(10) %>%
  kable(align = "c")
```

estimate	ellip	curv	pressure
0.0140082	1.267618	2.158739	133.3040
0.0118702	1.198681	4.437176	127.1982
0.0026530	1.375663	2.649938	160.2052
0.0194232	1.236829	3.621202	141.5427
0.0122134	1.318572	2.541340	129.3427
0.0167723	1.255885	3.236722	146.9361
0.0055484	1.326279	3.387357	167.8579
0.0112270	1.240926	4.254830	145.7207
0.0082627	1.191865	1.853877	87.7240
0.0019651	1.315273	4.144667	90.5485

```
kd_grid_est_tbl_29k %>%
  head(10) %>%
  kable(align = "c")
```

estimate	ellip	curv	pressure
0	0.8880910	0.1331512	31.02793
0	0.9131427	0.1331512	31.02793
0	0.9381944	0.1331512	31.02793
0	0.9632461	0.1331512	31.02793
0	0.9882978	0.1331512	31.02793
0	1.0133495	0.1331512	31.02793
0	1.0384012	0.1331512	31.02793
0	1.0634528	0.1331512	31.02793
0	1.0885045	0.1331512	31.02793
0	1.1135562	0.1331512	31.02793

Identify the 5% threshold value:

```
# 5% contour line from kd grid based on 10k MC data
percentile_5 <- kd_d3m[["cont"]]["5%"]
```

Verify that 5% (500/10,000) values fall below the threshold:

```
mc_est_tbl_10000 %>% filter(estimate <= percentile_5)
## # A tibble: 500 x 4
##     estimate ellip  curv pressure
##
##  1 0.000243   1.19  6.67    166.
##  2 0.000371   1.37  2.29     81.7
##  3 0.000356   1.22  2.73    200.
##  4 0.000134   1.27  7.91    114.
##  5 0.000560   1.47  3.89    163.
##  6 0.0000732  1.47  4.67    282.
##  7 0.000527   1.29  6.13    185.
##  8 0.000254   1.48  2.44    136.
##  9 0.000644   1.19  6.63    135.
## 10 0.000856   1.43  4.98    156.
## # ... with 490 more rows
```

If we wanted to know the nearest probability contour line for every point we could make a function to do so.

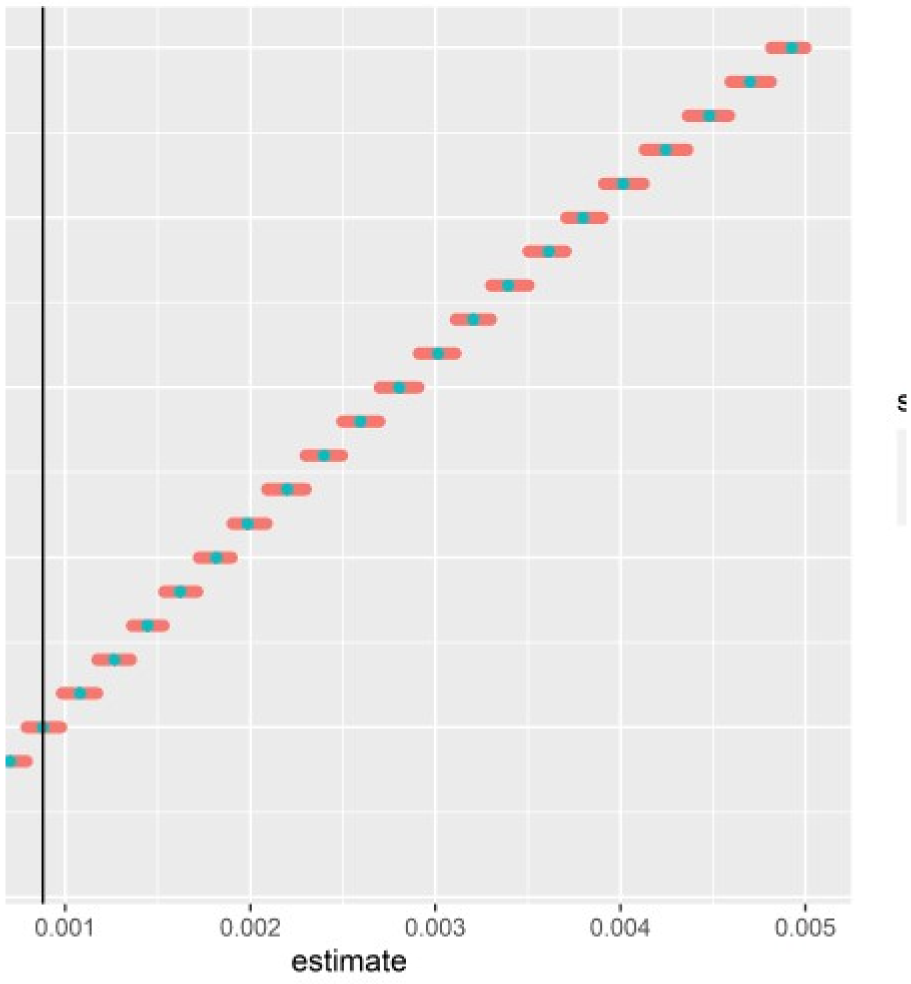
```
get_probs_fcn <- function(value) {
  t <- reference_grid_probs_tbl %>%
    mutate(value = value) %>%
    mutate(dif = abs(estimate - value)) %>%
    filter(dif == min(dif))

  t[[1, 1]]
}
```

Map the function over each value in the dataset.

```
# mc_1_to_99_tbl <- mc_est_tbl_10000 %>%
#   mutate(nearest_prob = map_dbl(estimate, get_probs_fcn))

# mc_1_to_99_tbl %>% write_rds(here::here("mc_1_to_99_tbl.rds"))
mc_1_to_99_tbl <- read_rds(here::here("mc_1_to_99_tbl.rds"))

mc_1_to_99_tbl
## # A tibble: 10,000 x 5
##    estimate ellip  curv pressure nearest_prob
##                                              
##  1  0.0140   1.27  2.16    133.            59
##  2  0.0119   1.20  4.44    127.            52
##  3  0.00265  1.38  2.65    160.            14
##  4  0.0194   1.24  3.62    142.            75
##  5  0.0122   1.32  2.54    129.            53
##  6  0.0168   1.26  3.24    147.            68
##  7  0.00555  1.33  3.39    168.            28
##  8  0.0112   1.24  4.25    146.            50
##  9  0.00826  1.19  1.85     87.7           39
## 10  0.00197  1.32  4.14     90.5           11
## # ... with 9,990 more rows
```

Let’s see what this actually did by comparing the probability contours estimated from the function above vs. the reference contours produced during generation of the kde object:

```
n <- mc_1_to_99_tbl %>%
  select(nearest_prob, estimate) %>%
  mutate(set = as_factor("manual_fit_10k"))

p <- cont_vals_tbl %>%
  mutate(set = as_factor("kde_output")) %>%
  rename(estimate = x, nearest_prob = probs)

bind_rows(n, p) %>%
  filter(estimate < .005 & estimate > 0) %>%
  ggplot(aes(x = estimate, y = nearest_prob)) +
  geom_point(aes(color = set)) +
  geom_vline(xintercept = percentile_5)
```



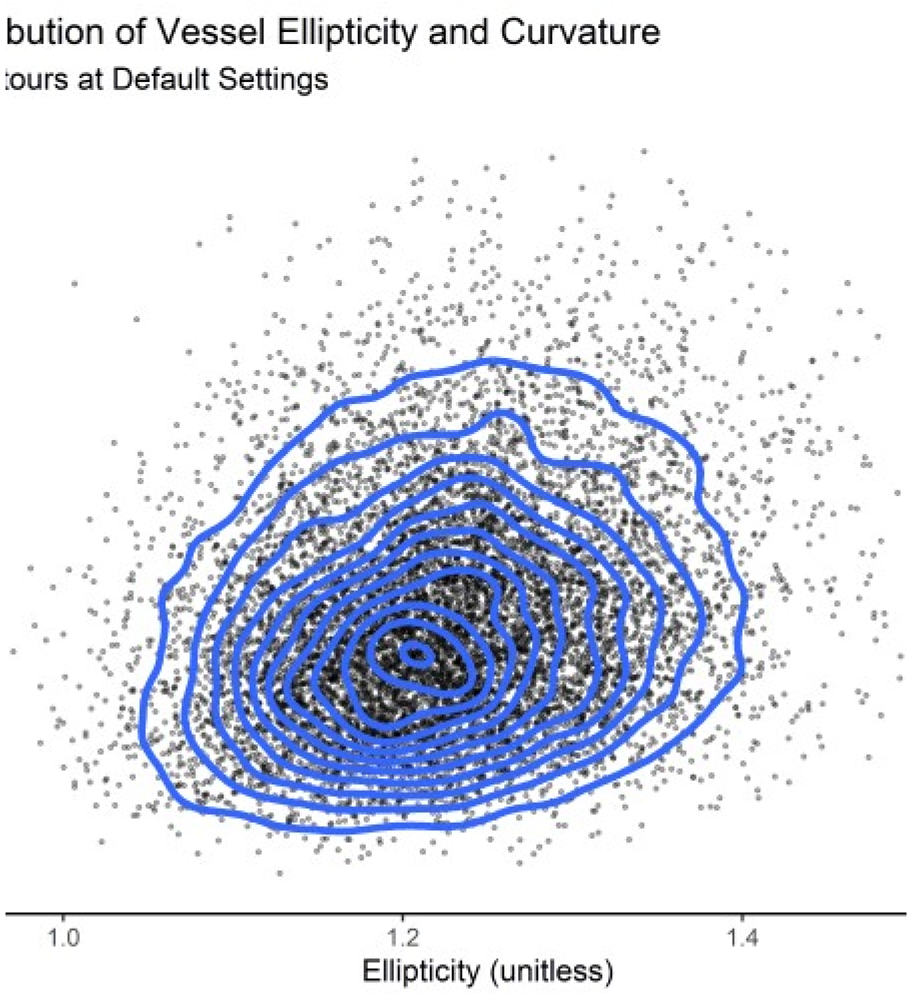
So it appropriately rounded each value in the simulated population to the nearest whole percentile, as desired.

Density Plot with Probability Contours in 2d

Now the fun part - Visualization. Here's how I'd go about creating a 2d density plots. If you don't care about specific probability contours, you can use the built in ggplot method. All you need is the raw data points, not the kde object or estimates. ggplot can do that for you.

Here's ellipticity vs. curvature:

```
cp_plt <- correlated_ln_draws_tbl %>%
  ggplot(aes(x = ellip, y = curv)) +
  geom_point(alpha = .3, size = .5) +
  geom_density2d(size = 1.3) +
  theme_classic() +
  xlim(c(.9, 1.6)) +
  ylim(c(1, 7.5)) +
  labs(
    title = "Joint Distribution of Vessel Ellipticity and Curvature",
    subtitle = "Density Contours at Default Settings",
    x = "Ellipticity (unitless)",
    y = "Radius of Curvature (mm)"
  )

cp_plt
```



This is pretty good. But for specifying specific contours and specifically we'll need a bit more. Here's a solution that I adapted from one I found on Cross Validated.

First, select the 2 variables of interest.

```
d <- correlated_ln_draws_tbl %>% select(ellip, curv)

## density function
kd <- ks::kde(d, compute.cont = TRUE, h = 0.05)
```

Now a a function to extract the points of the contour line from the kde:

```
get_contour <- function(kd_out = kd, prob = "5%") {
  contour_95 <- with(kd_out, contourLines(
    x = eval.points[[1]], y = eval.points[[2]],
    z = estimate, levels = cont[prob]
  )[[1]])
  as_tibble(contour_95) %>%
    mutate(prob = prob)
}
```

Map it over the kd object.

```
dat_out <- map_dfr(c("5%", "20%", "40%", "60%", "80%", "95%"), ~
get_contour(kd, .)) %>%
  group_by(prob) %>%
  mutate(n_val = 1:n()) %>%
  ungroup()

dat_out %>%
  head(10) %>%
  kable(align = "c")
```

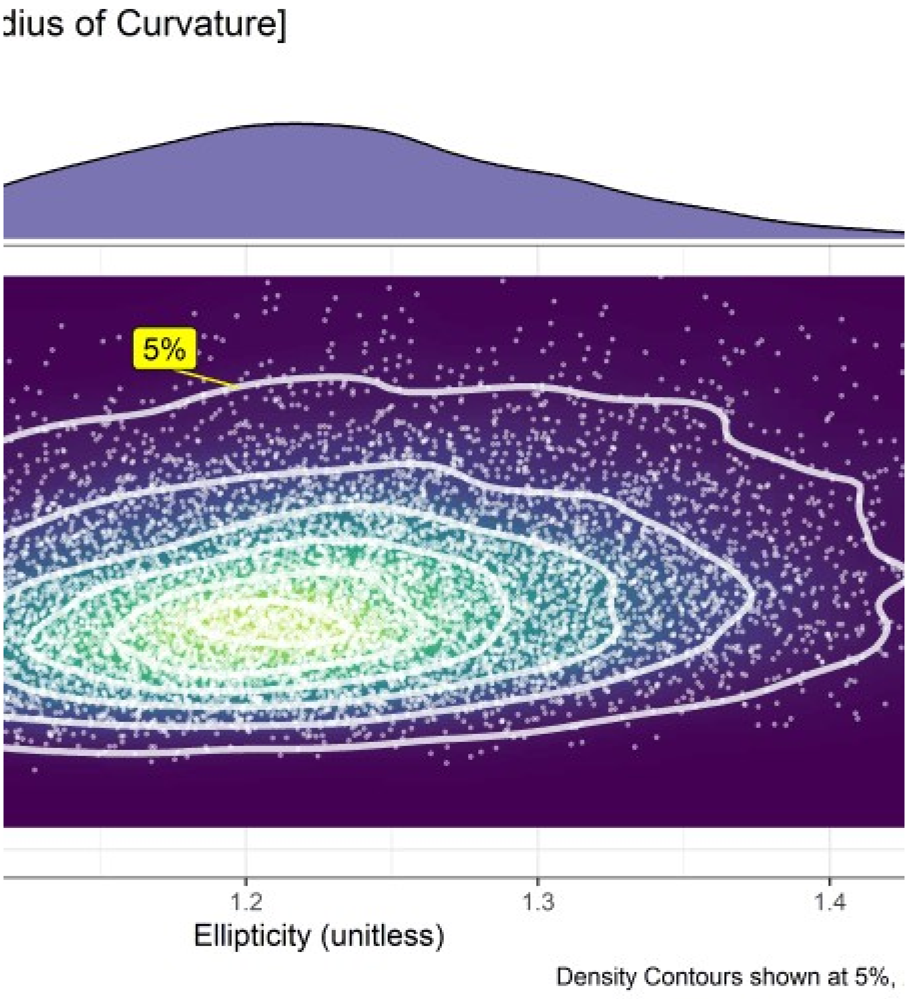
level	x	y	prob	n_val
0.1050379	1.024919	2.053556	5%	1
0.1050379	1.023034	2.112579	5%	2
0.1050379	1.022015	2.176684	5%	3
0.1050379	1.021665	2.240789	5%	4
0.1050379	1.021762	2.304894	5%	5
0.1050379	1.022205	2.368999	5%	6
0.1050379	1.022956	2.433104	5%	7
0.1050379	1.024009	2.497208	5%	8
0.1050379	1.024919	2.540507	5%	9
0.1050379	1.025311	2.561313	5%	10

Clean kde output

```
kd_df <- expand_grid(x = kd$eval.points[[1]], y = kd$eval.points[[2]])
%>%
  mutate(z = c(kd$estimate %>% t()))
```

Now visualize again, this time with probability contours at specified values and the 5% curve labeled with geom_label_repel().

```
label_tbl <- dat_out %>%
  filter(
    prob == "5%",
    n_val == 100
  )

# visualize
ellip_curv_2plt <- ggplot(data = kd_df, aes(x, y)) +
  geom_tile(aes(fill = z)) +
  geom_point(data = d, aes(x = ellip, y = curv), alpha = .4, size = .4,
colour = "white") +
  geom_path(aes(x, y, group = prob),
    data = dat_out %>% filter(prob %in% c("5%", "20%", "40%", "60%",
"80%", "95%")), colour = "white", size = 1.2, alpha = .8
  ) +
  #  geom_text(aes(label = prob), data =
  #              filter(dat_out, (prob %in% c("5%") & n_val==1)), # |
(prob %in% c("90%") & n_val==20)),
  #            colour = "yellow", size = 5)+
  geom_label_repel(
    data = label_tbl, aes(x, y),
    label = label_tbl$prob[1],
    fill = "yellow",
    color = "black",
    segment.color = "yellow",
    #    segment.size = 1,
    min.segment.length = unit(1, "lines"),
    nudge_y = .5,
    nudge_x = -.025
  ) +
  xlim(c(.95, 1.5)) +
  ylim(c(0, 7.5)) +
  labs(
    title = "Joint Distribution [Ellipticity and Radius of Curvature]",
    subtitle = "Simulated Data",
    caption = "Density Contours shown at 5%, 20%, 40%, 60%, 80%, 95%"
  ) +
  scale_fill_viridis_c(end = .9) +
  theme_bw() +
  theme(legend.position = "none") +
  labs(x = "Ellipticity (unitless)", y = "Radius of Curvature (mm)")
ggExtra::ggMarginal(ellip_curv_2plt, type = "density", fill =
"#403891ff", alpha = .7)
```

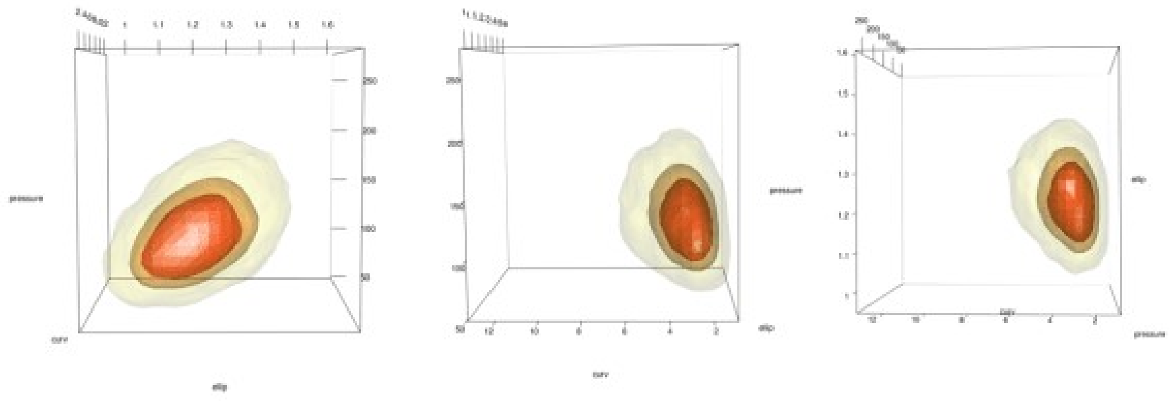


Nice! Now we have a clear delineation of the most common patients and most extreme patients and we can group them as desired. Almost done - the last thing to do is to see how to extend into 3d.

Density Plot with Probability Contours in 3d

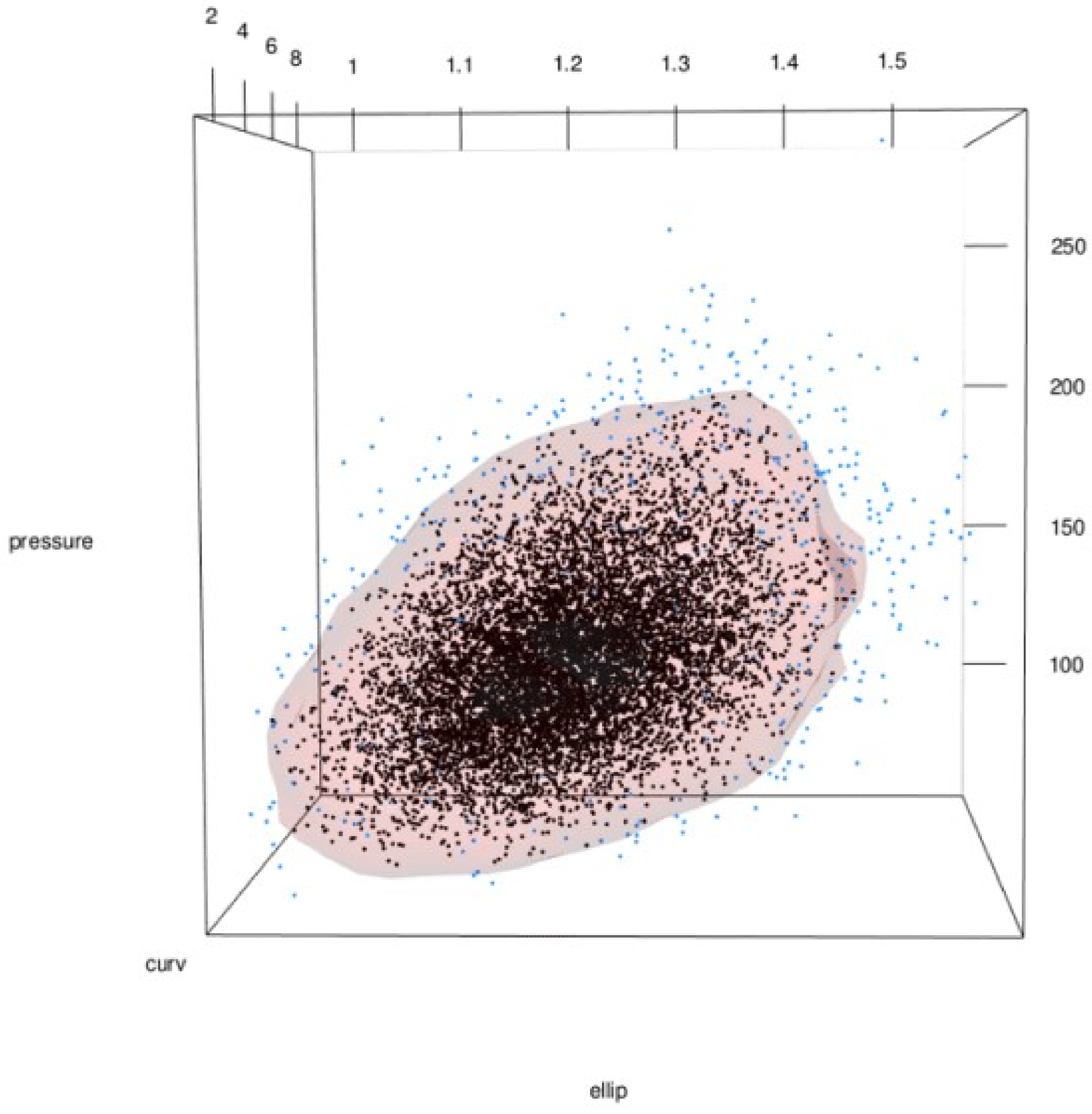
Honestly, this part is pretty easy thanks to a built in plot.kde method. Just use the cont argument to specify with probability contours you want.

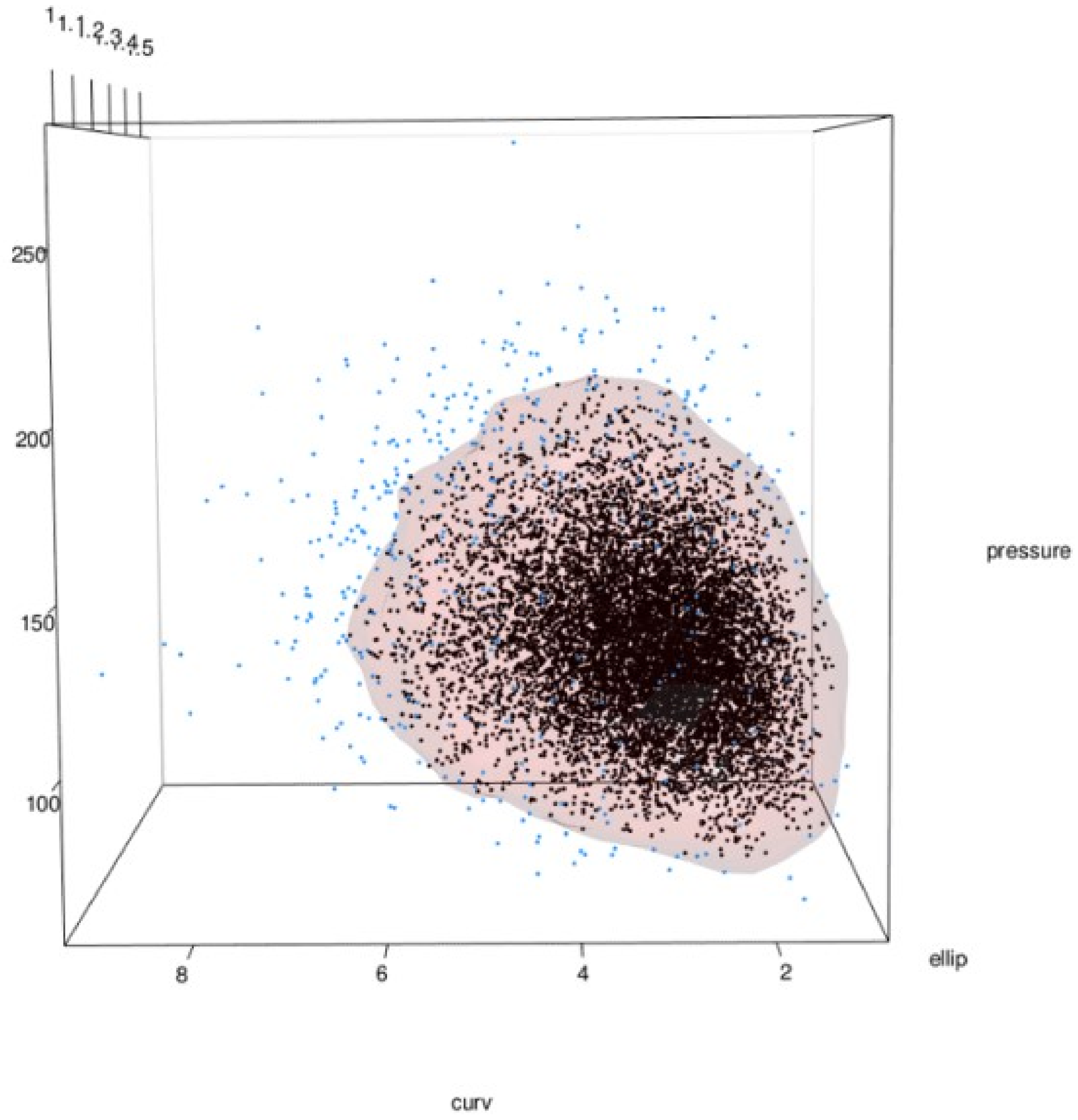
```
plot(x = kd_d3m, cont = c(45, 70, 95), drawpoints = FALSE, col.pt = 1)
```

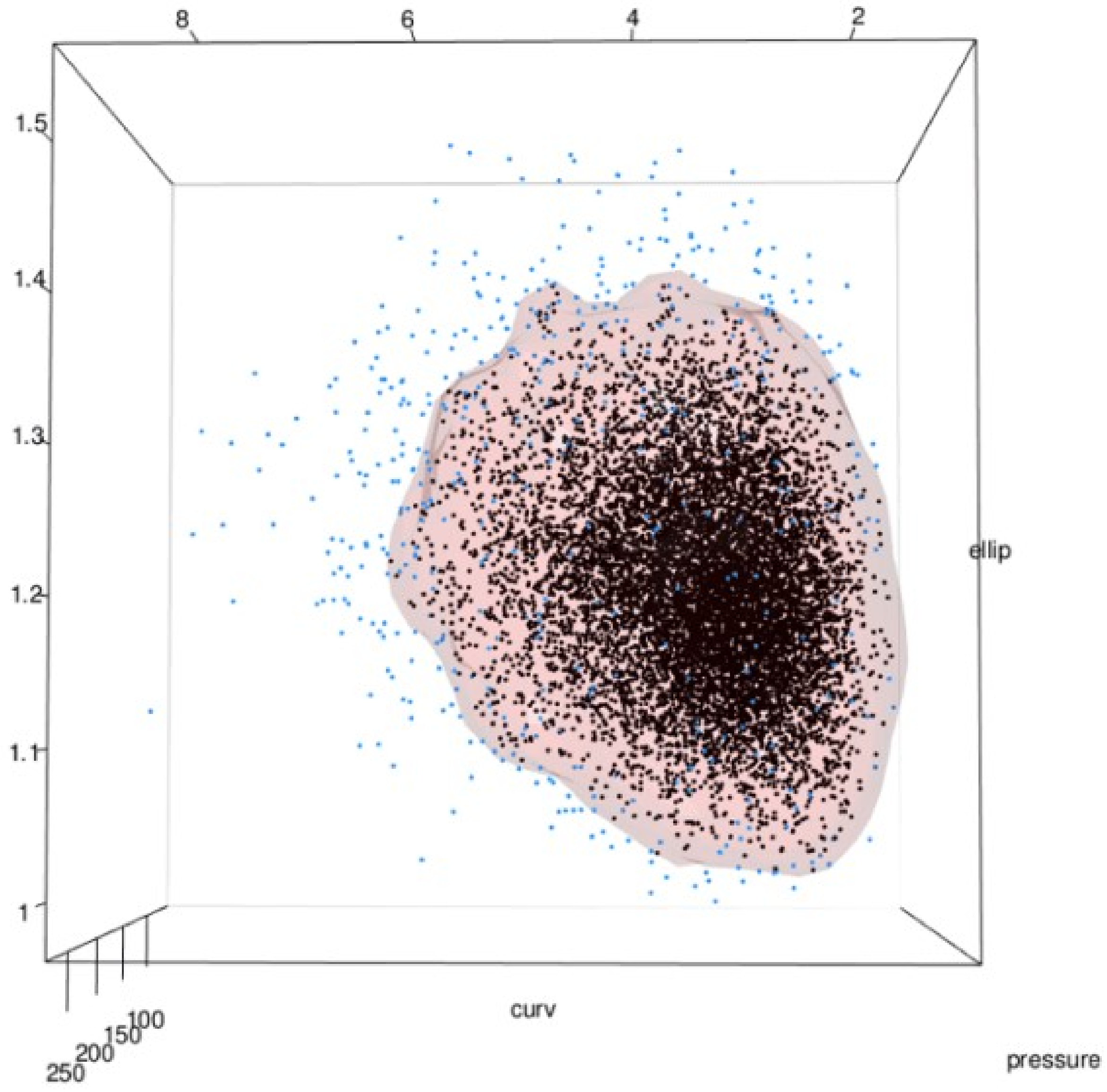


Add points using the points3d function. In this case I add 2 sets, 1 for the 5% most extreme and 1 for the 95% most common.

```
# plot(x = kd_d3m, cont = c(95) ,drawpoints = FALSE, col.pt = 1)
mc_lowest_5_tbl <- mc_1_to_99_tbl %>% filter(estimate < percentile_5)
mc_6_to_100_tbl <- mc_1_to_99_tbl %>% filter(estimate >= percentile_5)

# points3d(x = mc_lowest_5_tbl$ellip, y = mc_lowest_5_tbl$curv, z =
mc_lowest_5_tbl$pressure, color = "dodgerblue",  size = 3, alpha = 1)

# points3d(x = mc_6_to_100_tbl$ellip, y = mc_6_to_100_tbl$curv, z =
mc_6_to_100_tbl$pressure, color = "black",  size = 3, alpha = 1)
```


2
4
6
8
1
1.1
1.2
1.3
1.4
1.5
250
200
150
100
pressure
curv
ellip


1
1.1
1.2
1.3
1.4
1.5
250
200
150
100
pressure
8
6
4
2
ellip
curv



And there we have it! A 3d joint distribution with 95% probability density contour and groups separated by color and confirmed to cover 95/5% of the population.