Pluralsight Model Building Take-Home

William Rinauto

2022-03-23

```
library(tidyverse)
library(h2o)
library(vtable)
library(scales)
library(ggcorrplot)
library(kableExtra)
library(rpart)
library(rpart.plot)
```

```
quick_kable <- function(x) {
  kable(x) %>%
    kable_styling(bootstrap_options = "bordered",
                  full_width = FALSE)
}

training_dat <- read.csv('recruiting_zeta-disease_training-data_take-home-challenge - 2021_zeta-disease_training-data_take-h
ome-challenge.csv')
predict_these <- read.csv('recruiting_zeta-disease_prediction-data_take-home-challenge - 2021-01-21_zeta-disease_prediction-
data_take-home-challenge.csv')

#Check for duplicate rows.
#Even though there is no person identifier in this data, I am going to assume that if
#data values are the same for each data field in two or more rows, then that is duplicate data
#(meaning the same person has been included twice in the data).
#I think this is a fair assumption because it is highly improbable that two individuals would match
#across all of these features.

#How many rows will be dropped:
nrow(training_dat) - nrow(distinct(training_dat))
```

```
[1] 5
```

```
#Check to make sure that a person hasn't been recorded twice with two different values for zeta_disease
#If this matches return value of above line of code, then I'll know they haven't

nrow(training_dat %>% select(-zeta_disease)) - nrow(distinct(training_dat %>% select(-zeta_disease)))
```

```
[1] 5
```

```
#Drop the 5 duplicate rows:
training_dat <- distinct(training_dat)

#What proportion of training data is zeta positive?
mean(training_dat$zeta_disease)
```

```
[1] 0.3496855
```

```
#Define function to count NA values in each column of some dataframe
na_count <- function(df) sapply(df, function(c) sum(is.na(c)))

#A return value of true here indicates that there is no missing data in training data
all(na_count(training_dat) == 0)
```

```
[1] TRUE
```

```
#A return value of true here indicates that there is no missing data in testing data
all(na_count(predict_these) == 0)
```

```
[1] FALSE
```

```
#Returns False so I will investigate further:
na_count(predict_these)
```

```
          age            weight               bmi     blood_pressure
            0                 0                 0                  0
  insulin_test  liver_stress_test cardio_stress_test      years_smoking
            0                 0                 0                  0
  zeta_disease
           20
```

```
#In this case, I can see that the only "missing" data is zeta_disease, which makes sense because it hasn't been
#predicted yet



#Summary statistics table to get a quick sense of data distributions and also sanity check
#for outliers in mins and maximums (negative weight or age for example would be an indicator of bad data)




sumtable(training_dat)
```

Summary Statistics

Variable	N	Mean	Std. Dev.	Min	Pctl. 25	Pctl. 75	Max
age	795	30.636	12.861	18	21	38	109
weight	795	172.376	31.687	94	150	192	308
bmi	795	32.231	8.565	0	27.3	36.6	86.1
blood_pressure	795	69.567	19.922	0	62	80	157
insulin_test	795	85.906	126.687	0	0	130	1077
liver_stress_test	795	0.544	0.348	0.141	0.308	0.7	3.481
cardio_stress_test	795	43.067	30.495	0	0	62	214
years_smoking	795	4.057	4.177	0	1	6	40
zeta_disease	795	0.35	0.477	0	0	1	1

```
#Also want to see summary table of the testing data
sumtable(predict_these)
```

Summary Statistics

Variable	N	Mean	Std. Dev.	Min	Pctl. 25	Pctl. 75	Max
age	20	34.75	11.511	19	26.25	44.25	60
weight	20	178.8	27.935	120	153.25	197.75	216
bmi	20	34.48	6.629	25.8	30.25	37.6	50.7
blood_pressure	20	78.5	14.006	59	69.75	89.25	108
insulin_test	20	145.05	75.964	50	76.25	167.75	362
liver_stress_test	20	1.57	0.23	1.25	1.412	1.738	2.051

Variable	N	Mean	Std. Dev.	Min	Pctl. 25	Pctl. 75	Max
cardio_stress_test	20	61.95	9.703	43	55.75	68	83
years_smoking	20	6.05	3.471	2	3	7.5	13
zeta_disease	0						
… No	0	NaN%					
… Yes	0	NaN%					

```
#A few things stick out in the summary table for the training data.
#There are 5 fields (ignoring zeta_disease) that have minimum values of 0.
#Two of these stick out to me, because I would think that there is something problematic about having a 0 value
#here: bmi and blood_pressure

#Let's investigate further


#How often does 0 occur by data field?

#Define function to count % of 0's by field
zero_count <- function(df) sapply(df, function(c) percent(mean(c == 0)))
zero_count(training_dat)
```

```
          age             weight                bmi     blood_pressure 
         "0%"               "0%"               "1%"               "4%" 
  insulin_test  liver_stress_test cardio_stress_test      years_smoking 
        "47%"               "0%"              "29%"              "14%" 
  zeta_disease 
        "65%" 
```

```
#I can see that 1% of the data has bmi = 0, and 4% of the data has blood_pressure = 0
#(Not as important, but we should probably understand why only 14% of people in this dataset don't smoke. Is this a
#non-representative sample, or is something going on on mars that makes people more inclined to take up smoking?)

#At this point, I am going to make two assumptions moving forward with this project:

#ASSUMPTION 1: A person's BMI cannot be 0. According to the data dictionary, bmi = weight/height. Since weight
#definitely can't be 0 (and a weight of 0 would have shown up in the data, assuming that it is the
#same instance of weight measured here as was used to calculate bmi), bmi can't be 0.

#ASSUMPTION 2: A person's blood_pressure CANNOT be 0. I believe this means that the heart has stopped. I'm
#assuming no dead people had data collected here

#Assumptions 1 and 2 require me to take action to clean the data:

#Some possible options for each field:
#1. Mean imputation
#2. Group mean imputation
#3. KNN imputation
#4. Delete bad data

#To keep things simple, I will use mean imputation and replace all 0 values for blood_pressure and bmi with the
#mean of all non-zero values for each
non_zer_bp <- training_dat[training_dat$blood_pressure > 0,]
avg_bp <- mean(non_zer_bp$blood_pressure)

non_zer_bmi <- training_dat[training_dat$bmi > 0,]
avg_bmi <- mean(non_zer_bmi$bmi)

replacements_vals <- tibble(blood_pressure = avg_bp,
                            bmi = avg_bmi)

#This will be used in "production" environment to clean data before
#creating predictions
write.csv(replacements_vals, 'replacement_vals.csv',row.names =F)

training_dat <- training_dat %>%
  #Replace values of 0 with non-zero means
  mutate(blood_pressure = ifelse(blood_pressure == 0, avg_bp, blood_pressure),
         bmi = ifelse(bmi == 0, avg_bmi, bmi))



#see that min blood_pressure and bmi are no longer 0
min(training_dat$blood_pressure)
```

```
[1] 24
```

```
min(training_dat$bmi)
```

```
[1] 18.2
```

```
#One more sanity check: is years smoking ever greater than age? If it is, that's another indicator of bad data
#In this case, I will use greater than/equal to in my logical statement rather than strict inequality,
#therefore implying that it is not irrational for a person to have begun smoking
#on mars at 0 years-old. I will not presume to understand mars culture.

#A return value of false here indicates a logical contradiction in the data
all(training_dat$age >= training_dat$years_smoking)
```

```
[1] FALSE
```

```
#Returns false so I will investigate further:
smoker_anomaly_training <- filter(training_dat, years_smoking > age)
head(smoker_anomaly_training) %>% quick_kable()
```

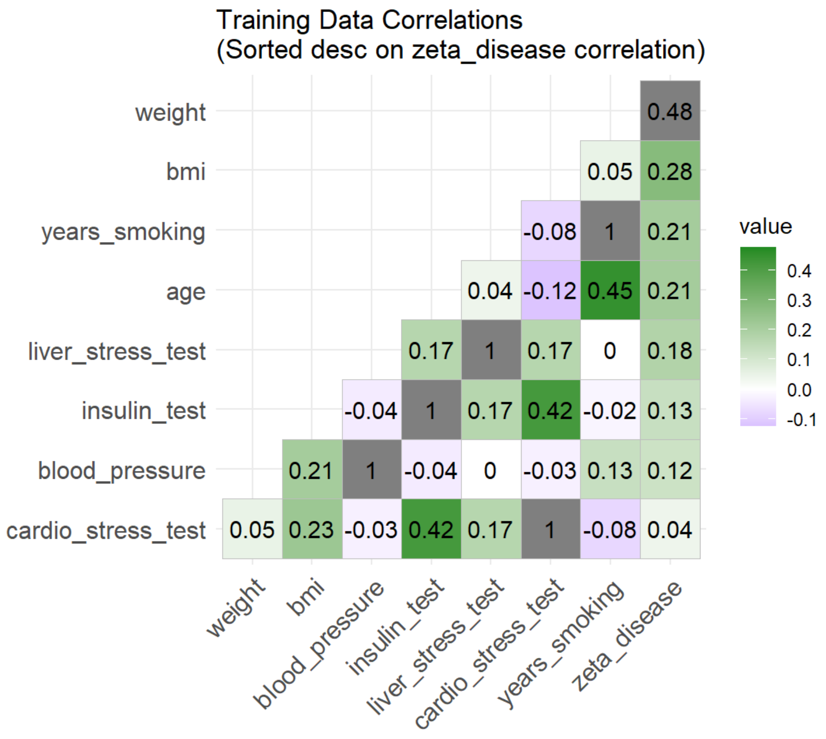
age	weight	bmi	blood_pressure	insulin_test	liver_stress_test	cardio_stress_test	years_smoking	zeta_disease
19	153	19.4	80	82	0.5538	41	22	0
19	158	25.3	62	278	0.9438	40	38	0

```
#In this case, we can see that two individuals labeled as 19 years old have supposedly been smoking for longer than they've
 been alive
#At this point, I have to clean up the data. I can do so by selecting from the four options listed above.

#This situation is a little different than the blood pressure and bmi situations, though, in that I am
#finding a contradiction in the data by comparing two
#data fields to each other, rather than making a logical assumption about a single data field in isolation.
#The implication is that I don't know
#which of these two data fields is the incorrect one for these two observations, age or years_smoking. Therefore, a decision
 to replace bad data values by imputation would
#be arbitrary: I have no way of saying that age must be replaced and years_smoking kept, or vice versa.
#Therefore, I will delete these rows of data entirely.
#Since this is only two rows of data that account for 0.25% of total observations in this dataset,
#it should be fairly inconsequential to remove them.

#Delete contradictory data:
training_dat <- filter(training_dat, age >= years_smoking)


#Summary table after cleaning up data:
sumtable(training_dat)
```

Summary Statistics

Variable	N	Mean	Std. Dev.	Min	Pctl. 25	Pctl. 75	Max
age	793	30.666	12.864	18	21	38	109
weight	793	172.419	31.715	94	150	192	308
bmi	793	32.709	7.656	18.2	27.6	36.6	86.1
blood_pressure	793	72.776	13.189	24	64	80	157
insulin_test	793	85.668	126.663	0	0	130	1077
liver_stress_test	793	0.543	0.348	0.141	0.308	0.7	3.481
cardio_stress_test	793	43.073	30.534	0	0	62	214
years_smoking	793	3.991	3.953	0	1	6	40
zeta_disease	793	0.351	0.477	0	0	1	1

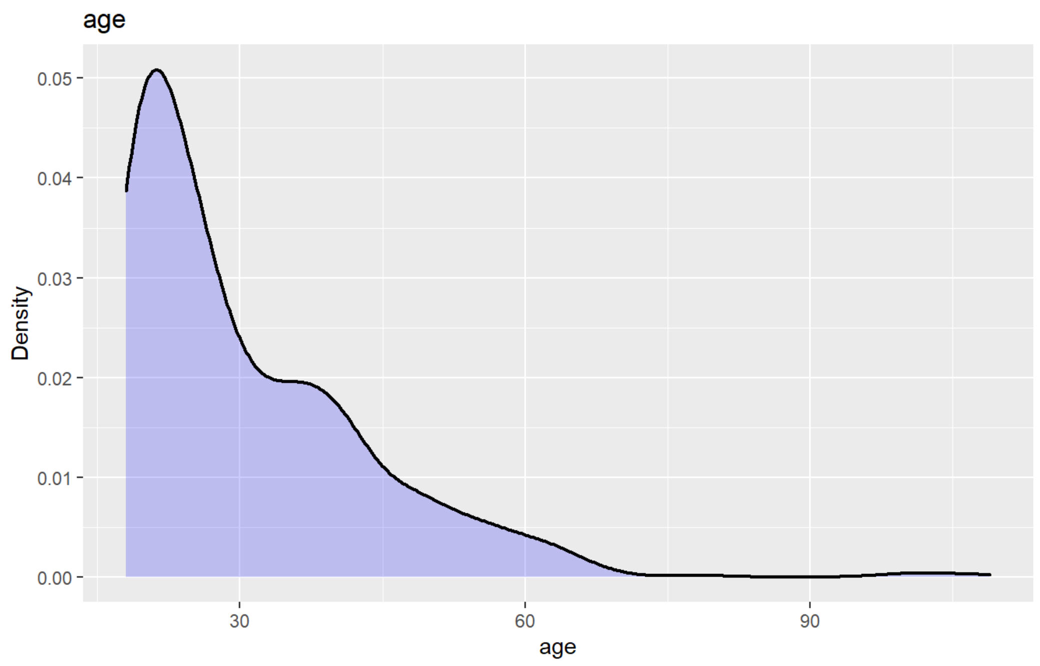
```
#Now that I am done cleaning the training data, I am going to
#search for relationships in the data

cor_dat <- training_dat
# names(cor_dat) <- knitr:::escape_latex(names(cor_dat))

cors <- cor(cor_dat)
cors <- cors[,order(cors['zeta_disease',], decreasing = F)]
cor_plot <- ggcorrplot(cors,
                       # colors = c(min(cors), 0, max(cors[cors!=1])),
                       type = 'lower',
                       lab = T) +
  scale_fill_gradient2(limit = c(min(cors),max(cors[cors!=1])), low = "blue", high =  "forestgreen", mid = "white", midpoint
= 0) +
  ggtitle('Training Data Correlations\n(Sorted desc on zeta_disease correlation)')

print(cor_plot)
```

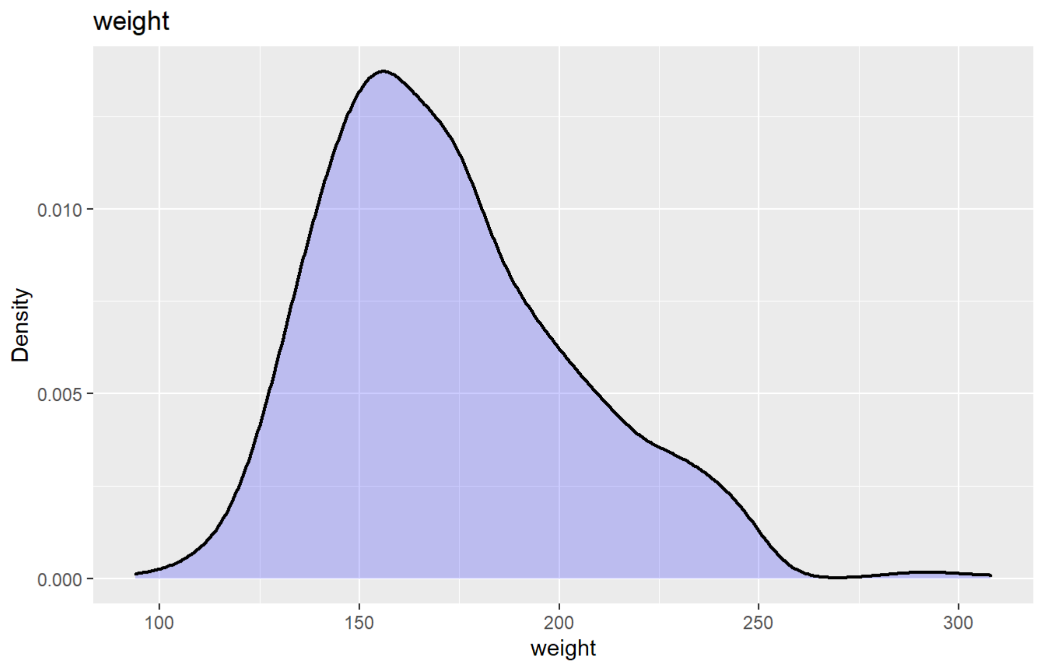


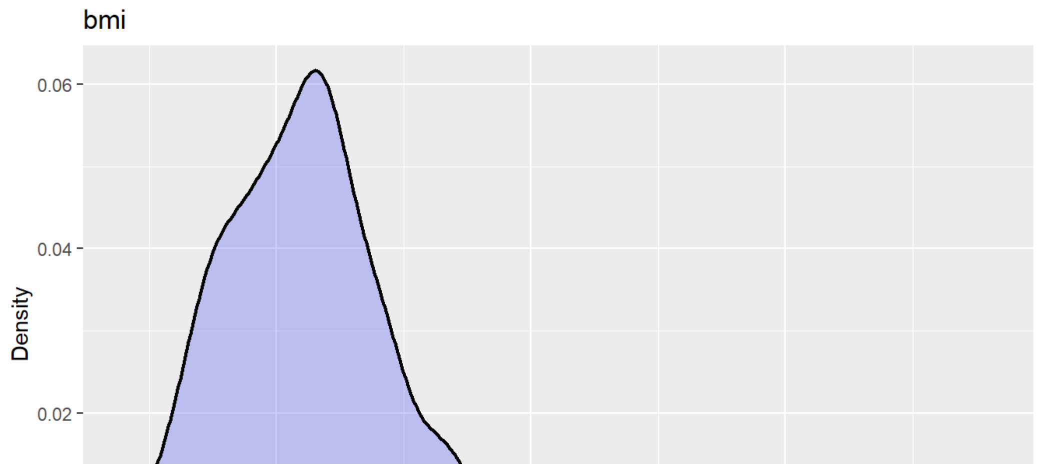
```
#Based on correlations alone, weight immediately sticks out as potentially predictive. Cardio stress test seems
#like it will be the least predictive


#Another angle I'd like to see on the data is average of each candidate predictor based on whether or not the person is
#zeta positive

#This shows an average profile of a zeta positive individual compared to a non zeta positive individual

zeta_means <- training_dat %>%
  group_by(zeta_disease) %>%
  summarise_all(mean) %>%
  left_join(training_dat %>% count(zeta_disease,name = 'Count')) %>%
  arrange(desc(zeta_disease))


#Chop off some decimal places
rnd <- function(x) round(x,2)

#mutate_all is a quick way to apply a function to every column in a dataframe
zeta_means <- mutate_all(zeta_means, rnd)

kable(zeta_means) %>%
  kable_styling(bootstrap_options = "bordered",
                full_width = FALSE)
```

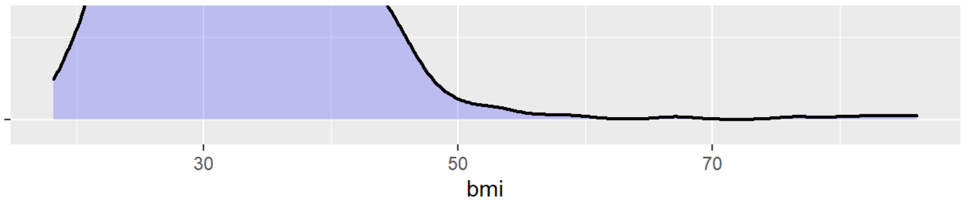
zeta_disease	age	weight	bmi	blood_pressure	insulin_test	liver_stress_test	cardio_stress_test	years_smoking	Count
1	34.40	192.92	35.62	74.98	107.56	0.63	44.57	5.14	278
0	28.65	161.35	31.14	71.59	73.85	0.50	42.26	3.37	515

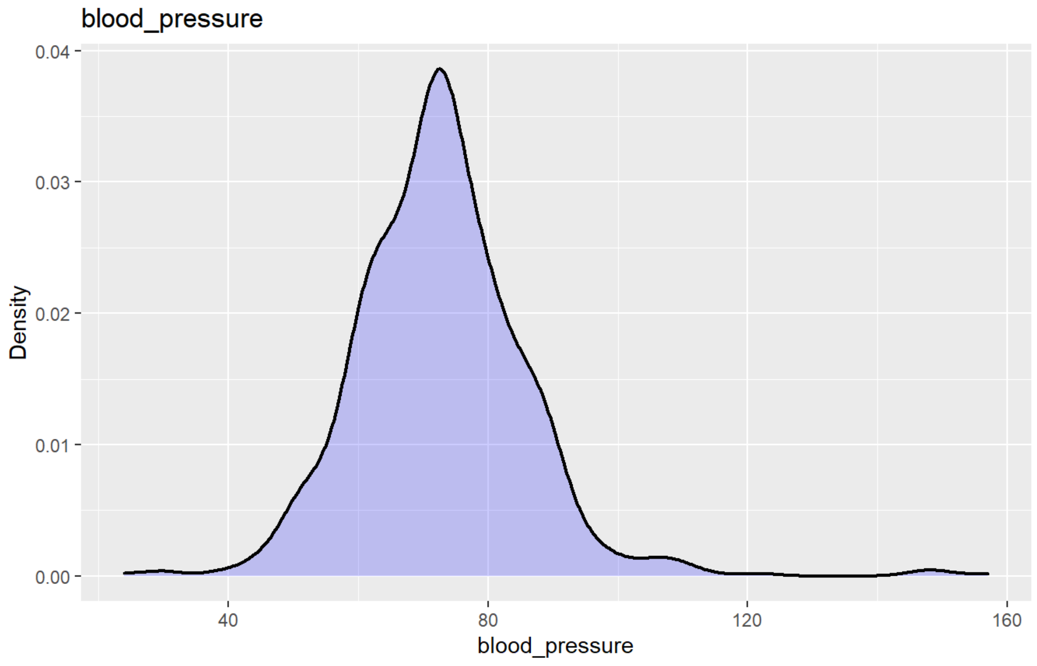
```
#Want to also see how various candidate predictors are distributed

#Create function that takes dataframe and column name as input, and outputs density plot
plot_dense <- function(dat, col) {

  ggplot(dat, aes_string(x = col)) +
    geom_density(lwd = .8, fill = 'blue',alpha = .2) +
    ylab('Density') +
    ggtitle(c) +
    xlab(paste0(c, "\n\n\n"))


}

cols <- names(training_dat)[!names(training_dat) == 'zeta_disease']

for(c in cols) {

  p <- plot_dense(training_dat, c)
  print(p)
}
```

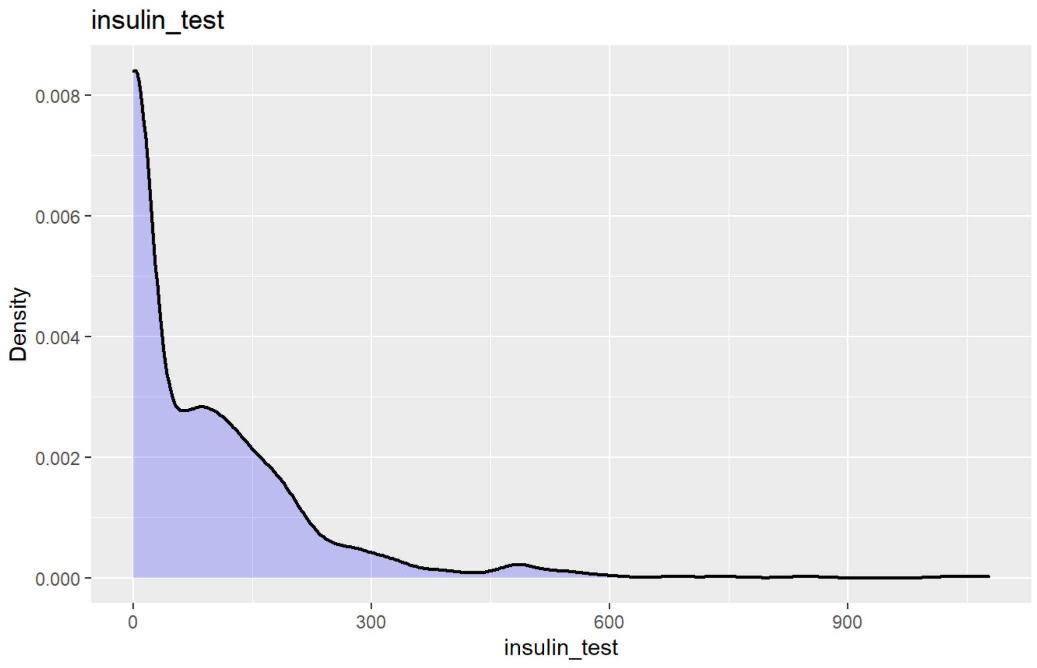

age
0.05
0.04
0.03
0.02
0.01
0.00
Density
30
60
90
age

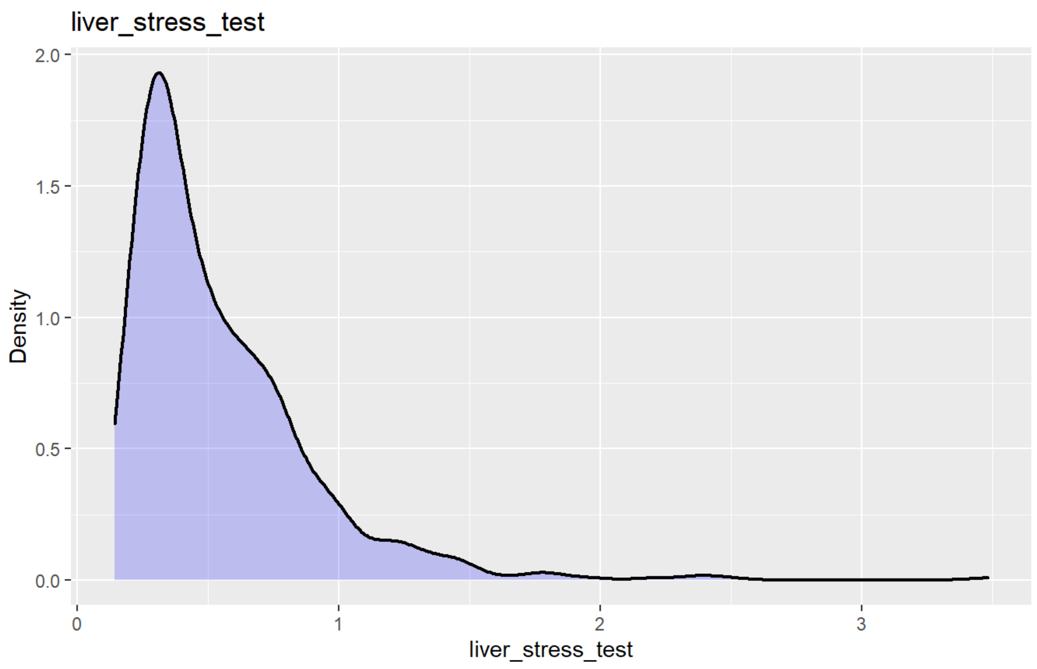

weight
0.010
0.005
0.000
Density
100
150
200
250
300
weight

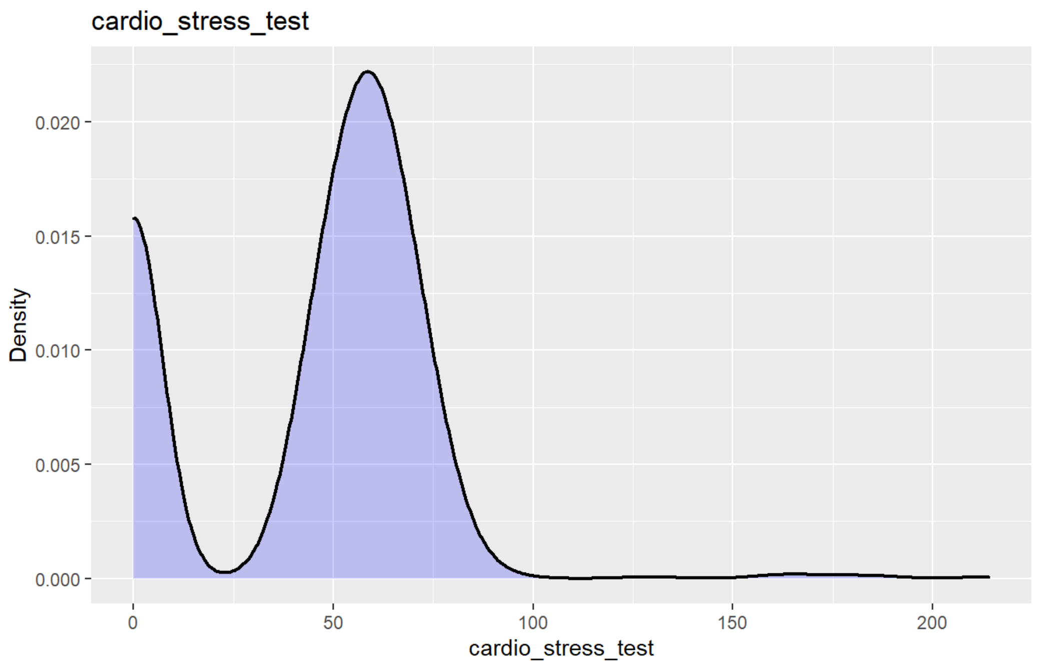

bmi
0.06
0.04
0.02
Density

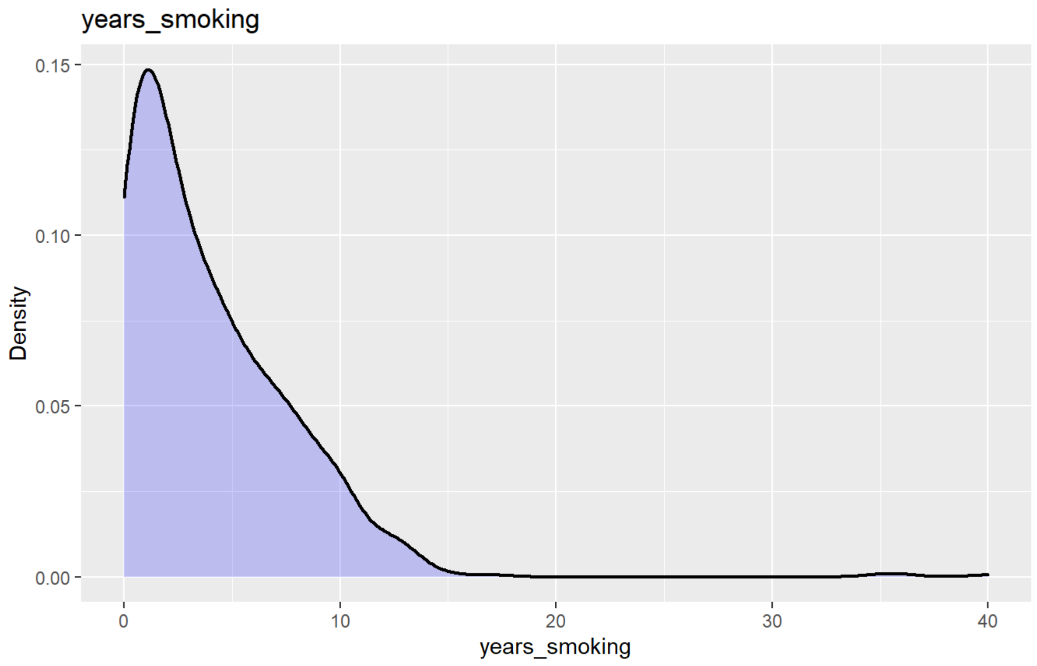

0.00
30
50
70
bmi


blood_pressure
Density
0.04
0.03
0.02
0.01
0.00
40
80
120
160
blood_pressure

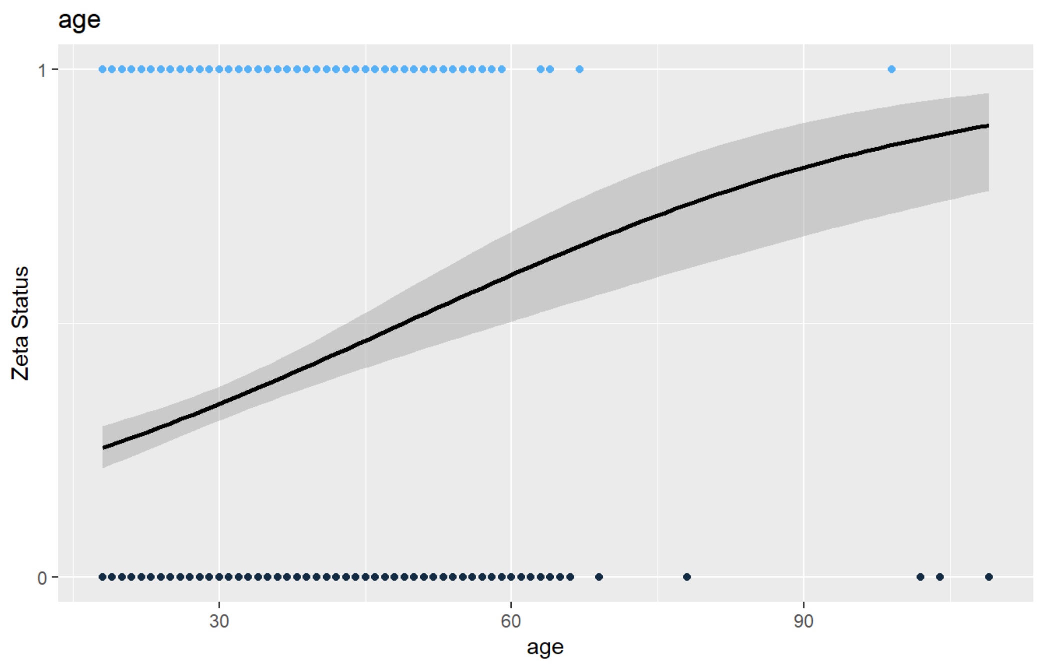

insulin_test
Density
0.008
0.006
0.004
0.002
0.000
0
300
600
900
insulin_test

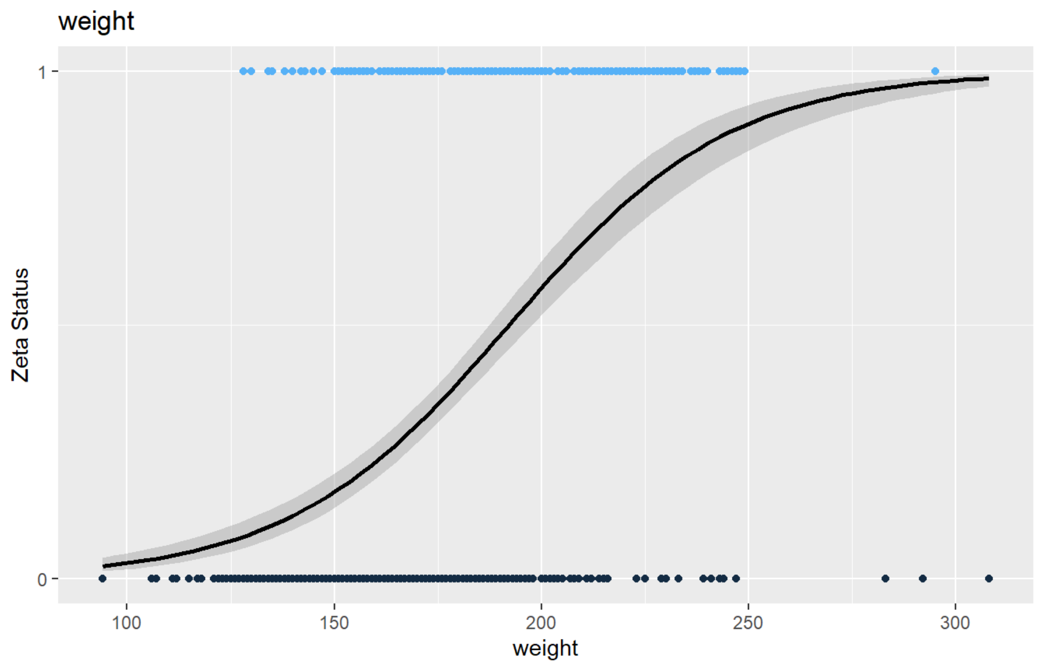

liver_stress_test
Density
2.0
1.5
1.0
0.5
0.0
0
1
2
3
liver_stress_test

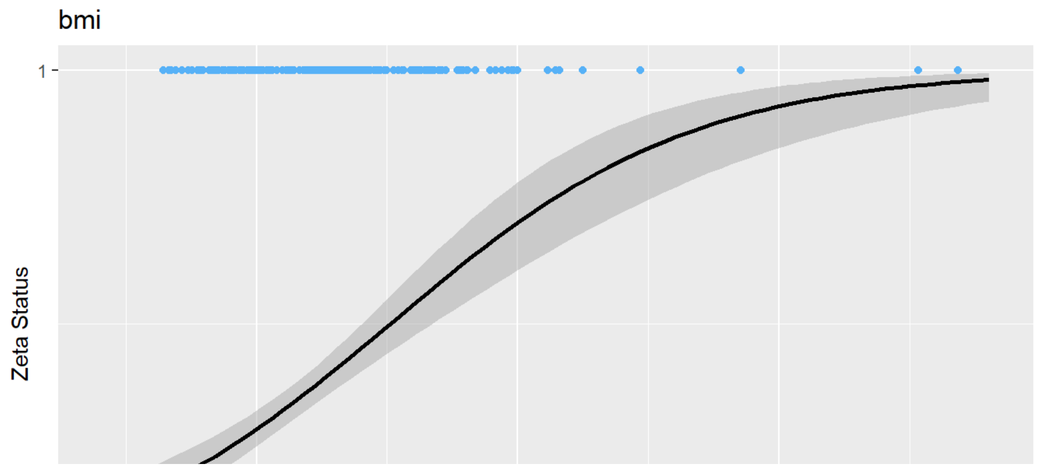

cardio_stress_test
Density
0.020
0.015
0.010
0.005
0.000
0
50
100
150
200
cardio_stress_test

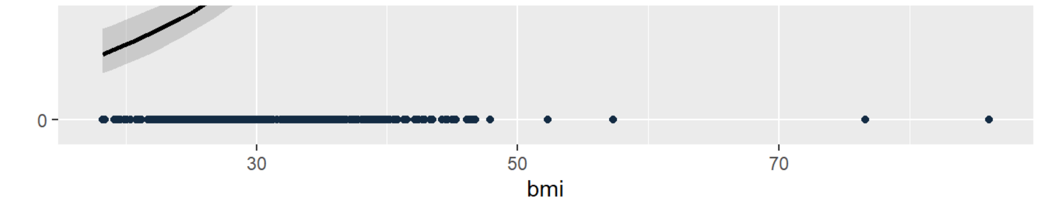


```
#Also interested in seeing each variable plotted against zeta_disease in a scatterplot
#I will add a smoothing line to give some indicacator of the relationship between each
#variable and zeta_disease.

#Create scatterplot function
plot_scatter <- function(dat1, dat2, col) {

  ggplot(dat1, aes_string(x = col, y = 'zeta_disease', color = 'zeta_disease')) +
    stat_smooth(method="glm", color="black", se=T,
                method.args = list(family=binomial)) +     geom_point() +
    # geom_vline(data = dat2, aes_string(xintercept = col, color = 'zeta_disease'), lwd = 1) +
    xlab(paste0(c, "\n\n\n")) +
    ggtitle(c) +
    scale_y_continuous(breaks = c(0,1)) +
    theme(legend.position = 'none') +
    ylab('Zeta Status')



}



for(c in cols) {

  p <- plot_scatter(training_dat, zeta_means, c)
  print(p)
}
```

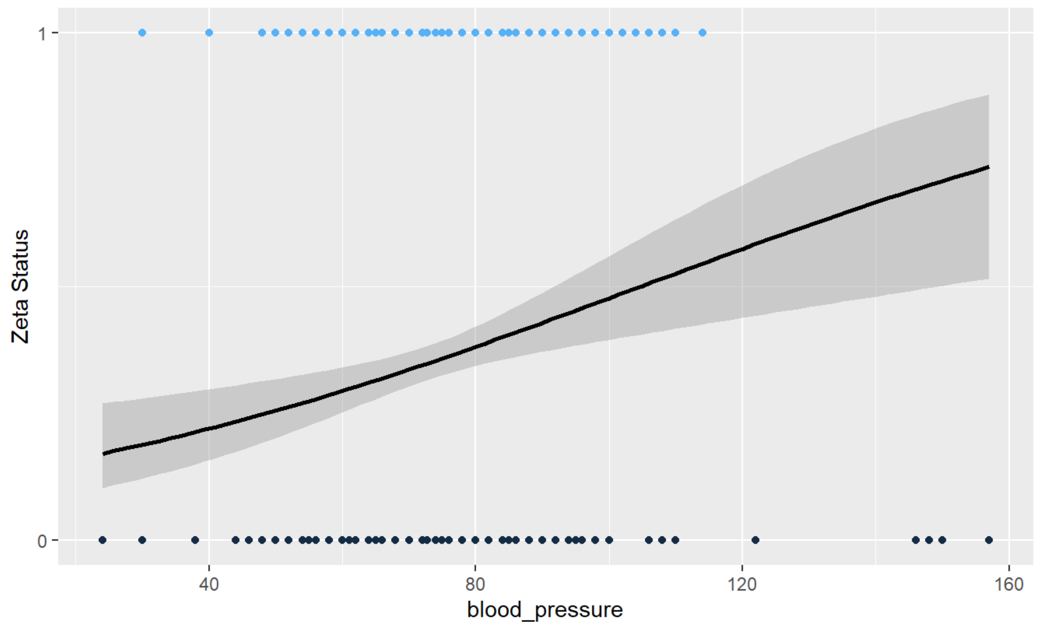

age
Zeta Status
1
0
30
60
90
age


weight
Zeta Status
1
0
100
150
200
250
300
weight

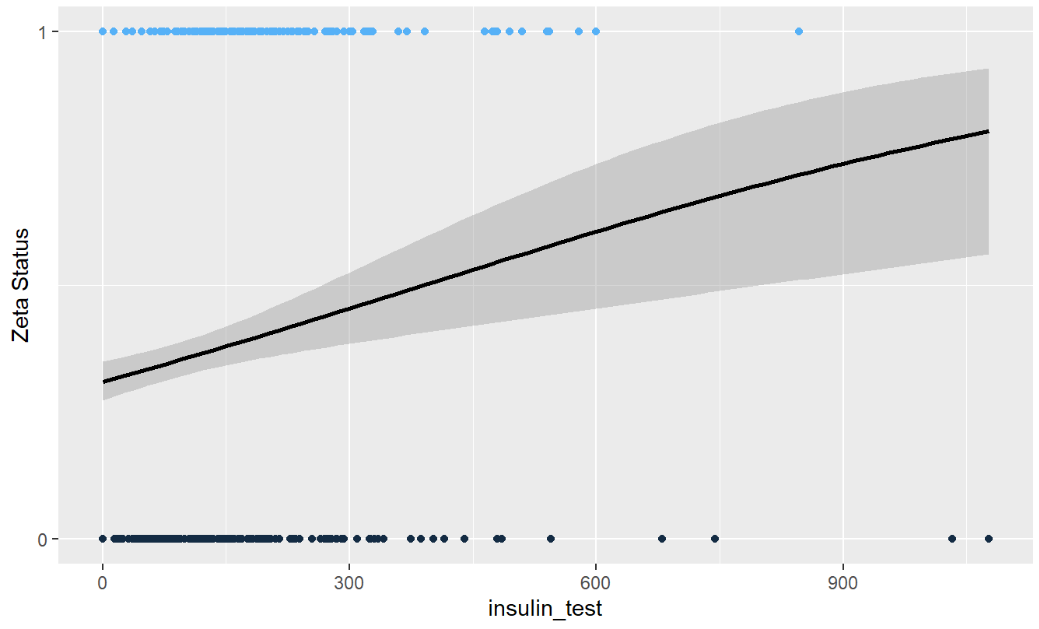

bmi
Zeta Status
1



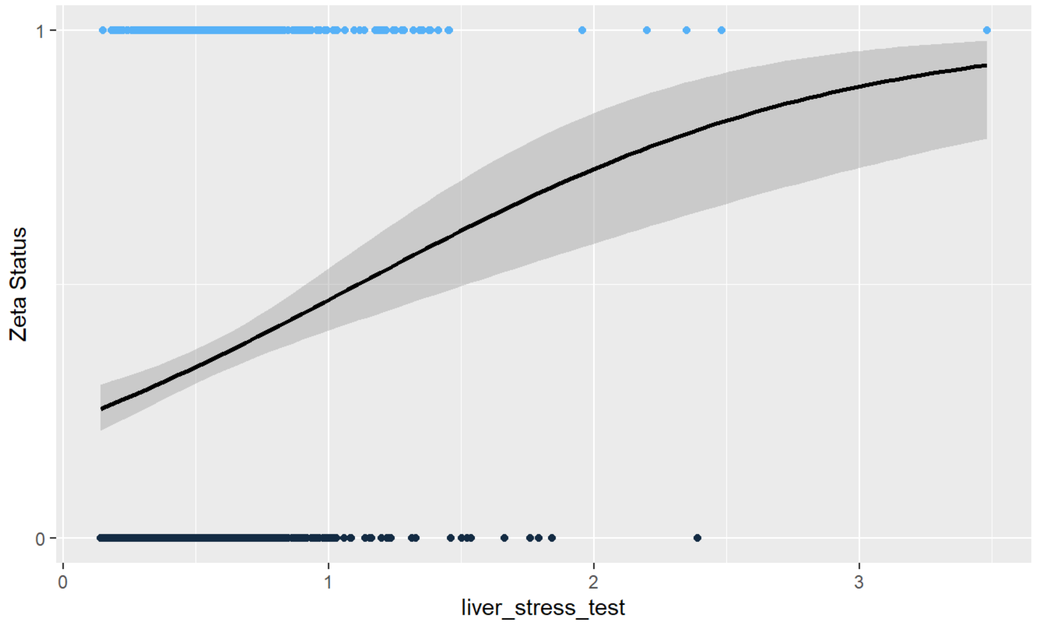
blood_pressure



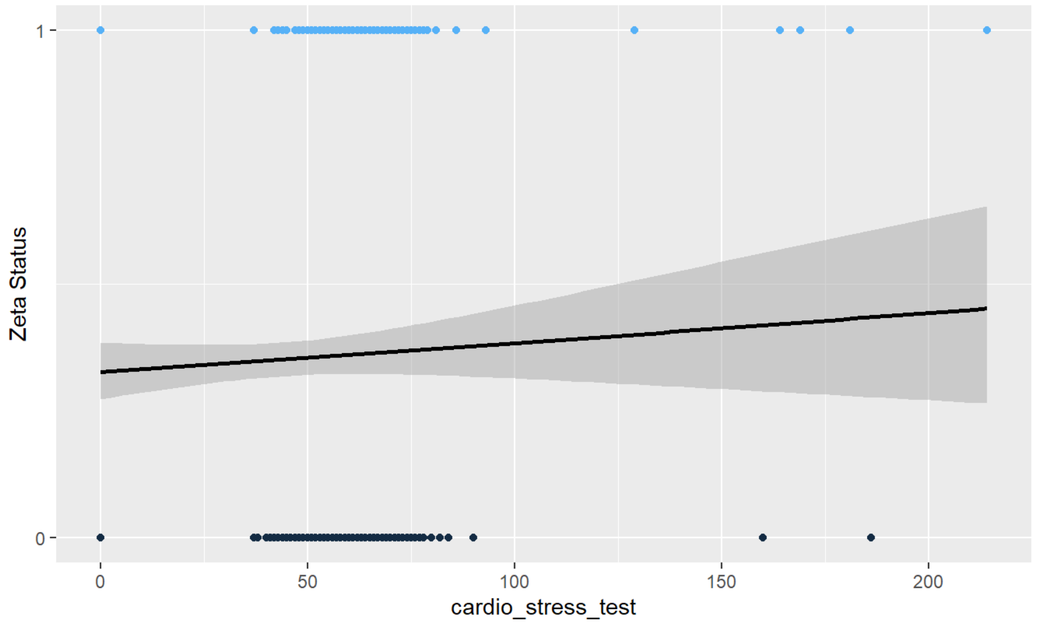
insulin_test

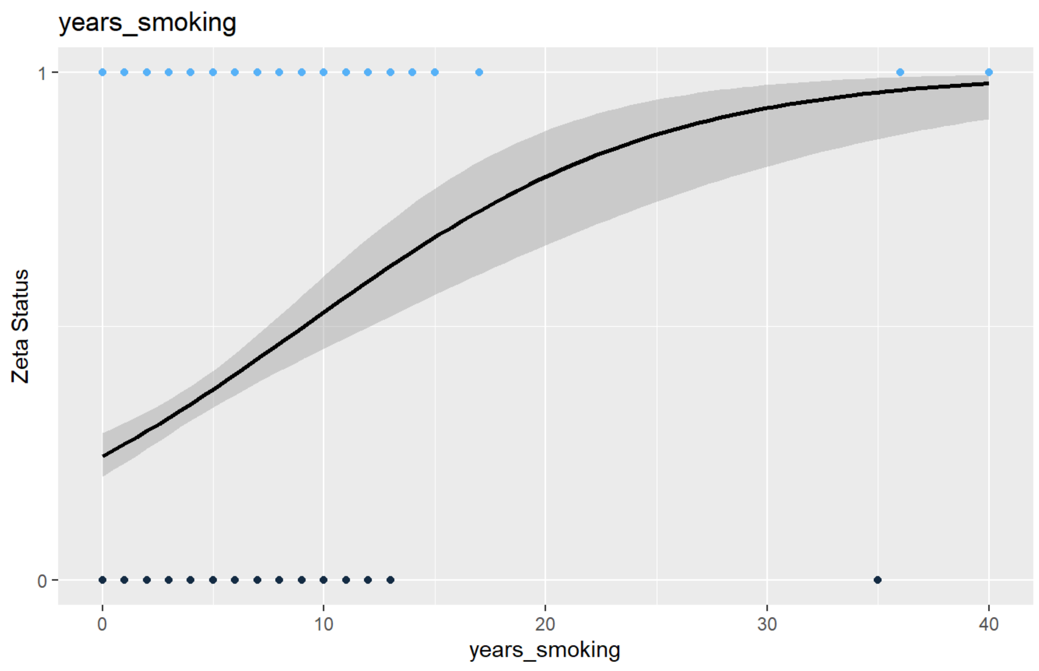


liver_stress_test

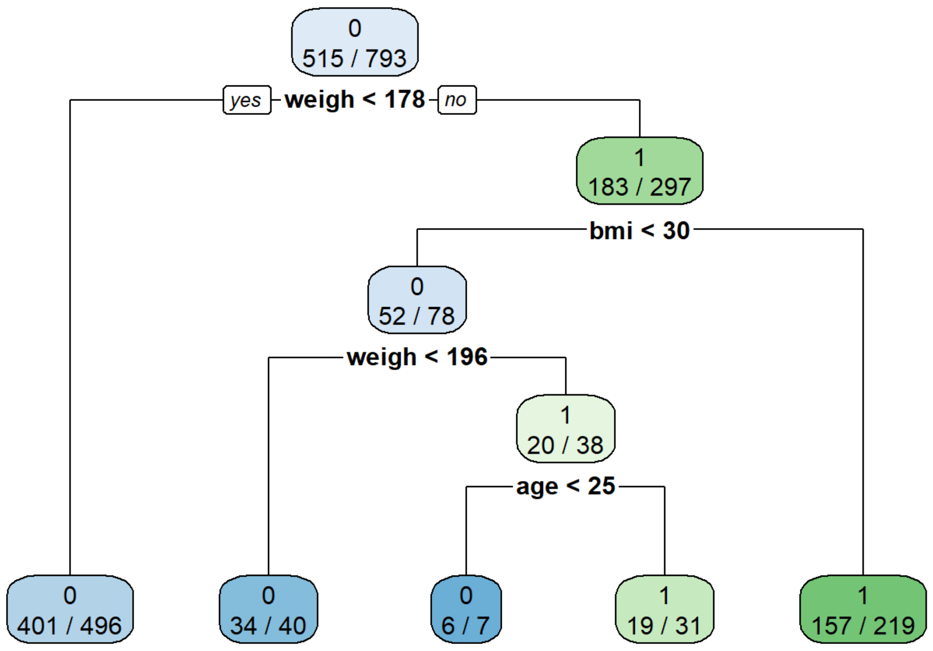


cardio_stress_test





```
#Tree------


#Another thing I want to see to get a very simple idea of how these features influence the outcome and interact with
#one another is a decision tree

#Indicate in training data that zeta_disease is categorical
training_dat <- mutate(training_dat, zeta_disease = as.factor(zeta_disease))

#I am going to shorten the variable names just so the plot prints tidier
tree_dat <- training_dat

shorten <- function(x) substr(x, 1,5)
names(tree_dat) <- sapply(names(tree_dat), shorten)

simple_tree <- rpart(zeta_ ~ ., data = tree_dat,
                     control = rpart.control(maxdepth = 4))

#Forcing the tree to make only 4 splits at most indicates that weight,
#BMI, and age will likely be important in a final model predicting zeta_disease

rpart.plot(simple_tree, cex = 1, extra = 2)
```



```
#What I'm seeing in the tree corroborates some of what I was seeing in the correlation table and plot:
#weight as the root node and the field with the highest importance measure is consistent with it having
#the highest correlation with the target variable. cardio_stress_test seems to be inconsequential based on both
#correlation with the target as well as importance in the tree
```

```
#Model----

#Now that I have cleaned the data and explored some of the relationships, I will
#build a few different classification models and select the one that is most successful


#The first thing I'm going to do is drop cardio_stress_test from the training data because it seems to be unimportant based
#on the preliminary analysis

training_dat <- select(training_dat, -cardio_stress_test)

#I also want to add a few features to the data. I could have
#added these in the data exploration step, but I wanted to avoid cluttering
#the rmarkdown document too much

# square <- function(x) x^2
#
# feature_funs <- list(sqrt = sqrt, square = square)


#mutate_at to target every variable that is not zeta_disease
#training_dat <- mutate_at(training_dat, vars(!matches('zeta_disease')), feature_funs)


#Actually, after creating features this way and running the models, I am seeing that they add nothing
#to the accuracy of any of the models, so I will comment out the prior couple lines of code

#Initialize h2o
h2o.init(nthreads = -1)
```

H2O is not running yet, starting it now…

Note: In case of errors look at the following log files: C:43bc35e779de/h2o_willi_started_from_r.out C:43bc45133464/h2o_willi_started_from_r.err

Starting H2O JVM and connecting: Connection successful!

R is connected to the H2O cluster: H2O cluster uptime: 1 seconds 948 milliseconds H2O cluster timezone: America/New_York H2O data parsing timezone: UTC H2O cluster version: 3.36.0.3 H2O cluster version age: 1 month and 7 days
H2O cluster name: H2O_started_from_R_willi_zew062 H2O cluster total nodes: 1 H2O cluster total memory: 15.93 GB H2O cluster total cores: 16 H2O cluster allowed cores: 16 H2O cluster healthy: TRUE H2O Connection ip: localhost H2O Connection port: 54321 H2O Connection proxy: NA H2O Internal Security: FALSE R Version: R version 4.1.3 (2022-03-10)

```
#Hide progrses bar because it looks bad in rendered document
h2o.no_progress()

#Convert target variable to categorical to avoid accidentally predicting continuous outcome:
training_dat <- mutate(training_dat, zeta_disease = as.factor(zeta_disease))

#Convert training data to h2o object:
training_dat_h2o  <- as.h2o(training_dat)


#this line looks pointless but I had deleted a few lines of code
#where previously it made sense to have this here. Now I'm keeping it to avoid
#changing code below
train_h2o <- training_dat_h2o


#Want to run predictions on final test data as I go along
testing_dat_h2o <- predict_these %>%
  select(-zeta_disease) %>%
  as.h2o()

#character vector of candidate regressors:
candidate_regressors <- names(training_dat)[names(training_dat) != 'zeta_disease']

#target_variable
target <- 'zeta_disease'

#See all data types to make sure nothing accidentally ended up as categorical somehow
#(zeta_disease should show as factor here)
sapply(training_dat, class)
```

```
          age            weight               bmi    blood_pressure
    "integer"         "integer"         "numeric"         "numeric"
 insulin_test liver_stress_test     years_smoking      zeta_disease
    "integer"         "numeric"         "integer"          "factor"
```

```
#For each model, I will use a grid search method to tune hyperparameters

#First test GLM

#It's probably overkill to use elastic net regression but
#if alpha = lamda = 0 is optimal, the grid search will indicate this
hyper_params_glm <- list(
  #Controls distribution between ridge and lasso componenets of penalty
  #https://docs.h2o.ai/h2o/latest-stable/h2o-docs/data-science/algo-params/alpha.html
  alpha = seq(from = 0, to = 1, by = 0.001),

  #Amount of regularization (large value here means coefficients shrink closer to 0
  #https://docs.h2o.ai/h2o/latest-stable/h2o-docs/data-science/algo-params/lambda.html
  lambda = c(.00001, .0001, .001, .01, .1, .5, 1)
)

#Number of models to be tested for in absence of RandomDiscrete strategy:
sapply(hyper_params_glm, length) %>% prod()
```

```
[1] 7007
```

```
#Controls how grid search is run (will use same search_criteria for every model)
search_criteria <- list(
  #RandomDiscrete grid search samples from parameter space
  #https://docs.h2o.ai/h2o/latest-stable/h2o-docs/grid-search.html
  strategy = "RandomDiscrete",
  max_runtime_secs = 30,
  max_models = 200,
  stopping_metric = "AUC",
  stopping_tolerance = 0.00001,
  stopping_rounds = 5,
  seed = 123
)

glm_models <- h2o.grid(algorithm = "glm",
                       grid_id = "regression",
                       x = candidate_regressors,
                       y = target,
                       training_frame = train_h2o,
                       nfolds = 10,
                       family = "binomial",
                       hyper_params = hyper_params_glm,
                       search_criteria = search_criteria,
                       seed = 123)

#Since h2o.predict will use f1 under the hood to determine classification
#threshold, I will select the model with the highest cv f1 score
glm_sorted <- h2o.getGrid(grid_id = "regression", sort_by = "f1", decreasing = TRUE)

#Top model when sorted descending on f1
glm_best <- h2o.getModel(glm_sorted@model_ids[[1]])


#Useful stackoverflow thread discussing h2o.performance object:
# https://stackoverflow.com/questions/43699454/how-to-understand-the-metrics-of-h2omodelmetrics-object-through-h2o-performan
ce

#xval = T below means I am pulling performance data based on cross validation
#testing datasets, not training data. Will do this throughout
glm_performance <- h2o.performance(glm_best, xval = T)
glm_f1 <- h2o.F1(glm_performance) %>%
  as.data.frame() %>%
  arrange(desc(f1))

#Store f1 value of best performing glm model
glm_f1 <- glm_f1[1,2]

#Appy predictions to test data. Will take a look at this later
final_predictions_glm <- h2o.predict(glm_best, testing_dat_h2o) %>%
  as.data.frame()
imp <- h2o.varimp(glm_best) %>%
  as.data.frame() %>%
  quick_kable()

#Next try a random forest. This tree based model will be better if there are
#interactions among variables

hyper_params_forest <- list(
  #number of trees
  #https://docs.h2o.ai/h2o/latest-stable/h2o-docs/data-science/algo-params/ntrees.html
  ntrees = 10000,

  #how deep tree can go:
  #https://docs.h2o.ai/h2o/latest-stable/h2o-docs/data-science/algo-params/max_depth.html
  max_depth = 12:25,

  #How much data must be in each bucket to make a split:
  #https://docs.h2o.ai/h2o/latest-stable/h2o-docs/data-science/algo-params/min_rows.html
  min_rows = seq(1,101, 5),

  #Row sampling rate:
  #https://docs.h2o.ai/h2o/latest-stable/h2o-docs/data-science/algo-params/sample_rate.html
  sample_rate = seq(.1, 1, by = .1),

  #Number of columns to sample at each node
  #https://docs.h2o.ai/h2o/latest-stable/h2o-docs/data-science/algo-params/mtries.html
  mtries = c(-1,1:7)
)


forest_models <- h2o.grid(algorithm = "randomForest",
                          grid_id = "forest",
                          x = candidate_regressors,
                          y = target,
                          training_frame = train_h2o,
                          nfolds = 10,
                          hyper_params = hyper_params_forest,
                          search_criteria = search_criteria,
                          seed = 123)

forest_sorted <- h2o.getGrid(grid_id = "forest", sort_by = "f1", decreasing = TRUE)

#Grab top performing model based on f1
forest_best <- h2o.getModel(forest_sorted@model_ids[[1]])

forest_perf <- h2o.performance(forest_best, xval = T)
forest_f1 <- h2o.F1(forest_perf) %>%
  as.data.frame() %>%
  arrange(desc(f1))

#grab f1 statistic
forest_f1 <- forest_f1[1,2]

#Apply predictions to test data
final_predictions_forest <- h2o.predict(forest_best, testing_dat_h2o) %>%
  as.data.frame()


#Last I will try gradient boosting:
hyper_params_gbm <- list(ntrees = 10000,
                         max_depth = 5:15,
                         min_rows = c(15, 20,30, 50,100),

                         #GBM learn rate (how much to adjust predicted residuals based on new tree)
                         #https://docs.h2o.ai/h2o/latest-stable/h2o-docs/data-science/algo-params/learn_rate.html
                         learn_rate = c(0.001,0.01,0.1, .3, .5),

                         #Change in learn rate each round
                         #https://docs.h2o.ai/h2o/latest-stable/h2o-docs/data-science/algo-params/learn_rate_annealing.html
                         learn_rate_annealing = c(0.99,0.999,1),
                         sample_rate = seq(.2, 1, by = 1),
                         col_sample_rate = seq(.1, 1, by = 1)

)


models_gbm <- h2o.grid(algorithm = "gbm", grid_id = "gbm",
                       x = candidate_regressors,
                       y = target,
                       training_frame = train_h2o,
                       nfolds = 10,
                       hyper_params = hyper_params_gbm,
                       search_criteria = search_criteria,
                       seed = 123)
```

```
gbm_sorted <- h2o.getGrid(grid_id = "gbm", sort_by = "f1", decreasing = TRUE)

#Best gbm model based on f1 stat
gbm_best <- h2o.getModel(gbm_sorted@model_ids[[1]])

gbm_perf <- h2o.performance(gbm_best, xval = T)
gbm_f1 <- h2o.F1(gbm_perf) %>%
  as.data.frame() %>%
  arrange(desc(f1))

#grab best gbm f1 statistic
gbm_f1 <- gbm_f1[1,2]

#Final predictions.. will take a look later
final_predictions_gbm <- h2o.predict(gbm_best, testing_dat_h2o) %>%
  as.data.frame()

models_perf_metric <- list(glm = glm_f1, rf = forest_f1, gbm = gbm_f1)
#f1 score of each model:
models_perf_metric
```

```
$glm [1] 0.6872111

$rf [1] 0.6985173

$gbm [1] 0.6953938
```

```
#See how each model predicts on test data
dat <- bind_cols(list(final_predictions_glm, final_predictions_forest, final_predictions_gbm)) %>%
  select(contains('predict'))
names(dat) <- c('glm','forest','gbm')
head(dat, 20) %>%
  quick_kable()
```

glm	forest	gbm
1	0	0
1	1	1
0	0	0
1	1	1
1	1	1
0	1	0
0	0	0
0	0	0
1	1	1
1	0	0
1	1	1
1	1	1
1	0	0
1	1	1
1	1	1
1	1	1

glm	forest	gbm
1	1	1
1	1	1
1	1	1
1	1	1

```
#proportion of zeta positive predictions by model on test data:
convert_fct_numeric <- function(x) mean(as.numeric(as.character(x)))
lapply(dat, convert_fct_numeric) %>%
  bind_rows() %>%
  quick_kable()
```

glm	forest	gbm
0.8	0.7	0.65

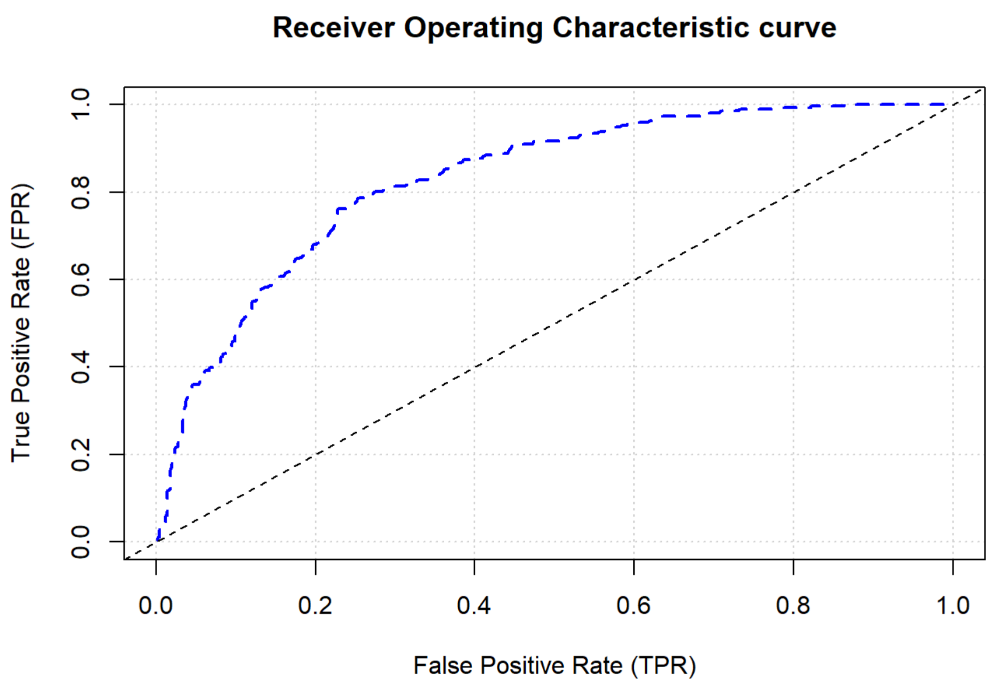
```
#Based on f1, random forest just barely squeaks out as the winner,
#so I will go with that

#Save model to be ingested by python
h2o.saveModel(forest_best, getwd(), filename = 'model', force = T)
```

[1] “C:\Users\willi\Desktop\Pluralsight Test\model”

```
#Take a look at confusion matrix. By default,
#this will be based on training data

#Note that h2o switches 0 and 1 from their conventional positions
#(0/1 instead of 1/0)
h2o.confusionMatrix(forest_best) %>%
  as.data.frame() %>%
  quick_kable()
```

	0	1	Error	Rate
0	394	121	0.2349515	=121/515
1	63	215	0.2266187	=63/278
Totals	457	336	0.2320303	=184/793

```
#ROC Curve for GBM model based on cross validation
plot(h2o.performance(forest_best, xval = T) ,type='roc')
```



```
#Since random forest performed the best, that is what I will
#use in my "production" python environment.
```