2550 Problem Set 2

Alyson Singleton

10/4/2019

Part A

Use the data in the baseseg.csv file on kidney disease to construct a good fitting model for GFR as a function of the following potential predictors measured at baseline.

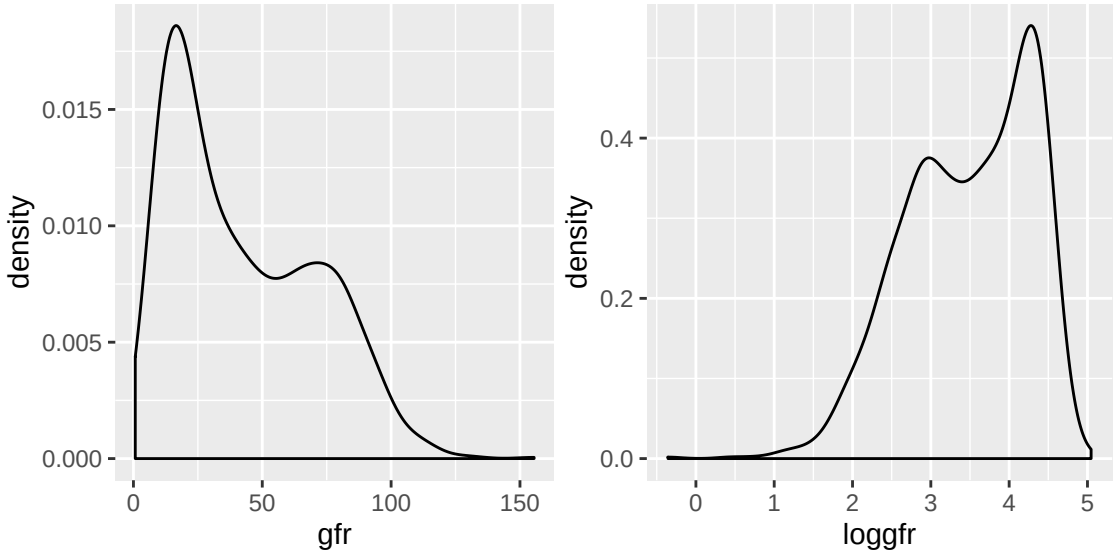
Introduction

GFR stands for glomerular filtration rate and is considered the best measure of kidney function. We therefore use it as our outcome measure in this analysis to explore possible predictor of kidney disease. Normal kidney function is represented by a GFR of 90 or above [https://www.kidney.org/atoz/content/gfr]. Our potential predictors are serum creatinine (bascre), systolic blood pressure (sbase), diastolic blood pressure (dbase), urine protein (baseu), age, sex (1 if male, 0 if female) and race (specifically, black or non-black). The website cited also references blood creatinine, age, body size and gender as variables a doctor might use to "calculate" a patient's kidney function.

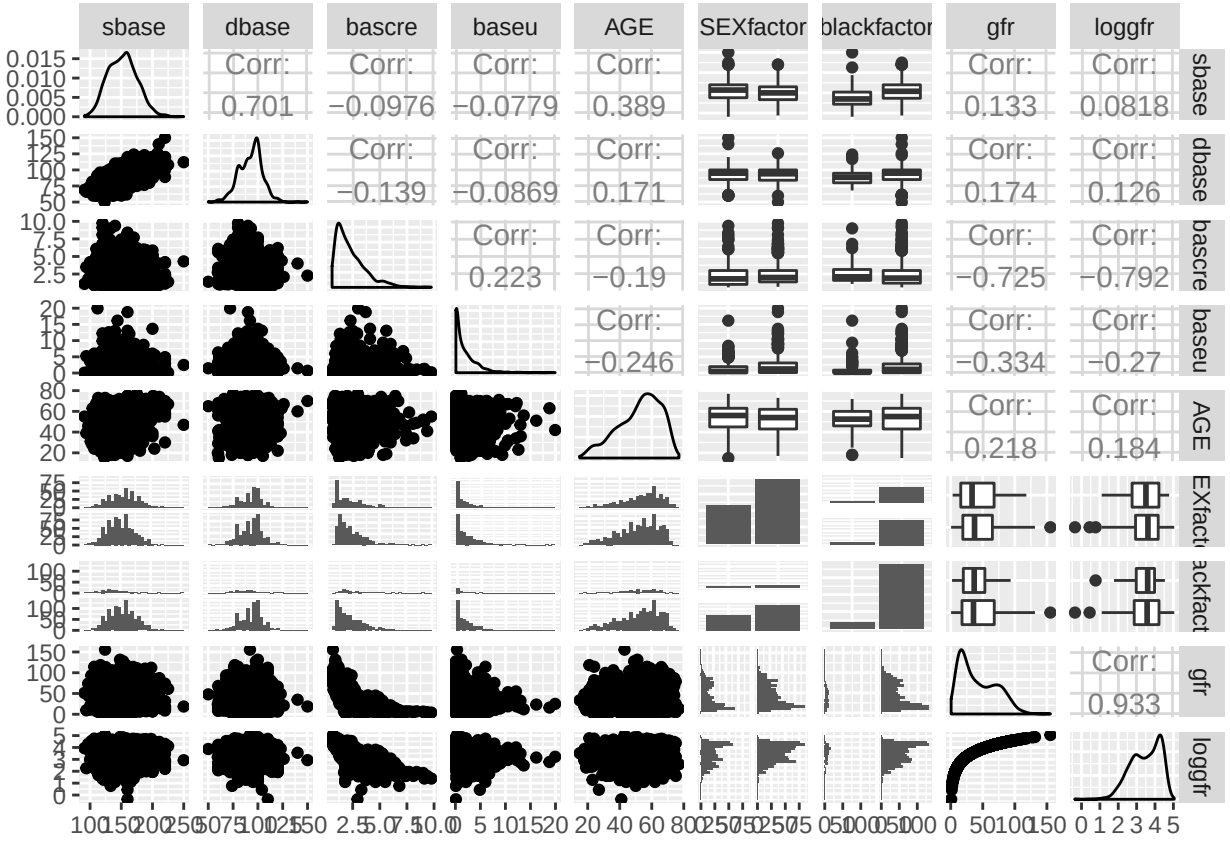
Data Exploration

First I look to explore the data set. The first thing that I notice is that there are missing values. Specifically there are 611 observations where gfr, our outcome, is encoded as NA. This acounts for about 1/3 of the observations in our data set. I decided to remove the observations where gfr is missing since observations without an outcome will not contribute to building a useful model. I recognize that there are other ways to deal with missing data that help you prevent bias and allow you to use the other data included in the row with the missing value. I, however, do not know how to employ these techniques and I opted for explicitly naming their removal and conditioning my results upon this decision.

Next I began to investigate the distributions of the variables and their relationships to each other. I first noticed that the outcome variable, gfr, is skewed positively, to the right. This indicated to me to use a log transform so that the outcome takes more the shape of a bell curve. See basic plots below:



Next, I stumbled upon a function called "ggpairs" from the package "GGally" that allows for an efficient survey of the other relationships. I have printed the output from this function below, including both GFR and the log transform of GFR.



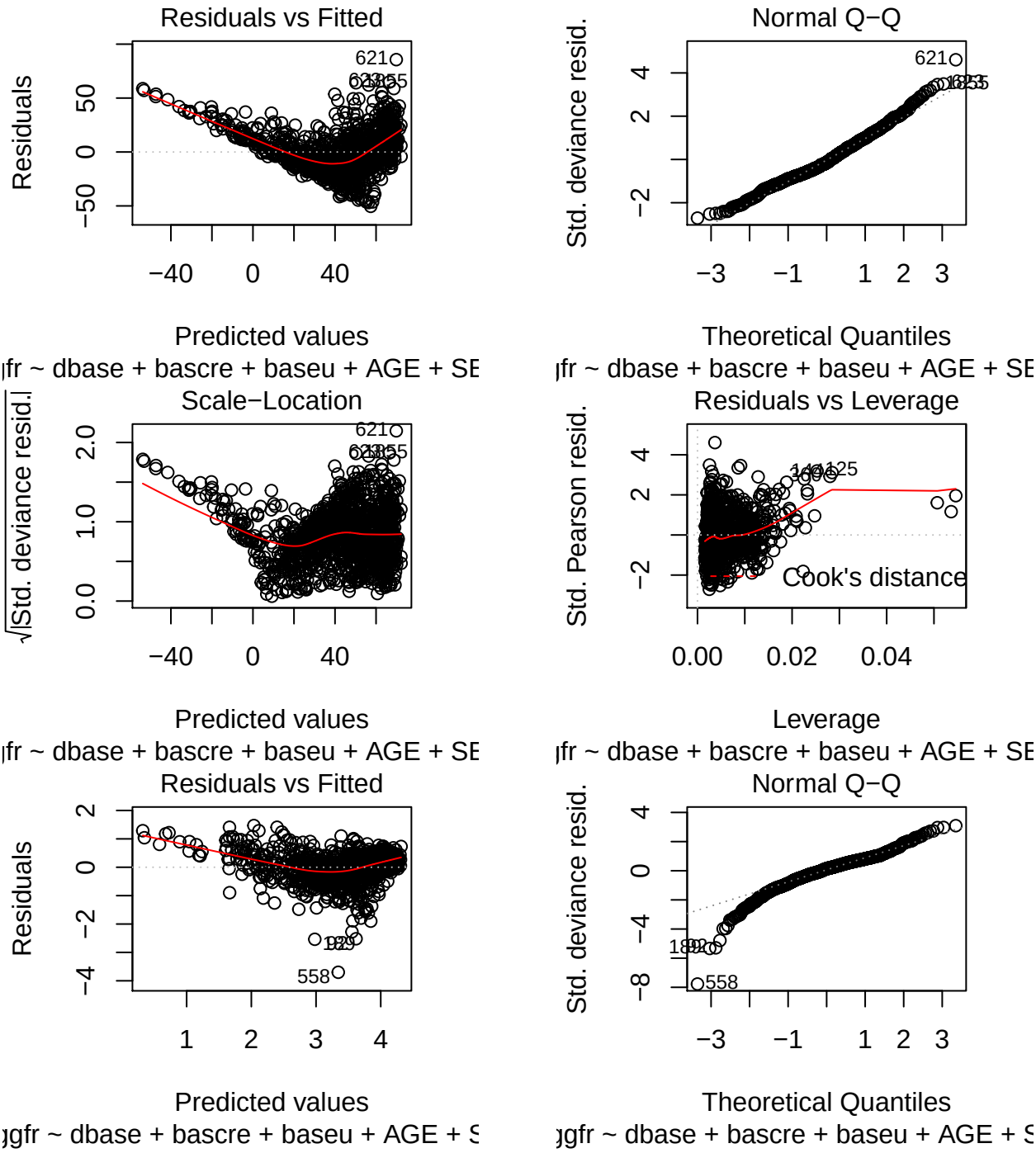
My main take aways from these plots were that sbase and dbase are highly correlated, that the log transform additionally assists in bascre taking a more linear shape, and that I do not see immediate needs for interaction terms. Lastly, I get an idea from the correlations of bascre with log(GFR) that it might be a relevant predictor.

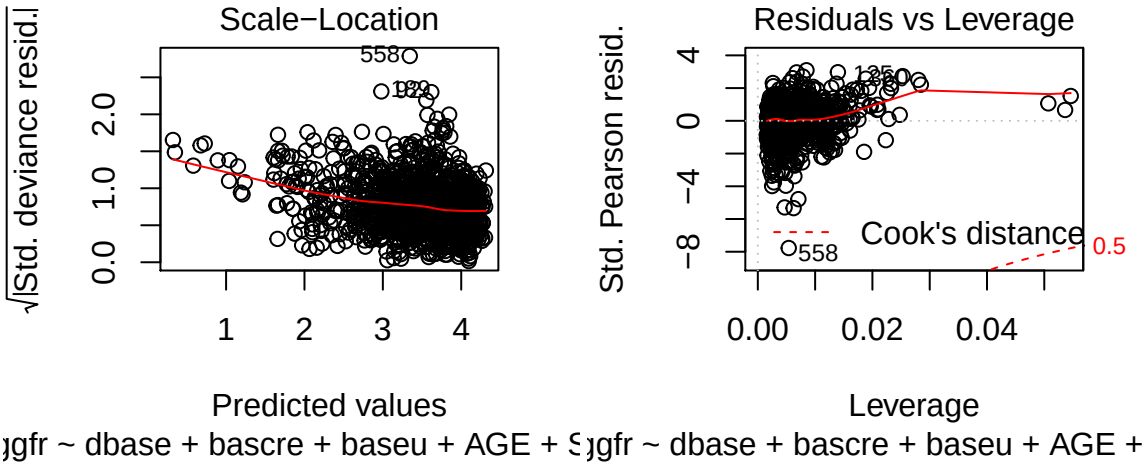
Model Construction and Testing

Now I begin testing models and making edits / removals / additions one by one, testing their performance as I go. I used AIC, BIC, ANOVA, diagnostic plots, and VIF as model tests. I begin with all seven variables and original GFR outcome. Here are my decisions and my rationale :

(1) Sbase is not significant in this initial model and analysis of variance inflation factors encourages its removal (sbase and dbase both >5 until removal of sbase).

(2) Choose to use log(GFR) as outcome. Transformation reduces AIC from 10858 to 1705.7. This change also improves the output on the diagnostic plots. Specifically, the residuals v fitted values plot for GFR indicates there is non-linear relationship between predictor variables and the outcome variable. This plot improves with the log transform. Additionally, the Spread-Location plot improves with the transform, leading me to believe the assumption of homoscedasticity is more appropriate when using the transform. See these diagnostic plots below.


Residuals vs Fitted
Residuals
621
−50
0
50
−40
0
40
Predicted values
lfr ~ dbase + bascre + baseu + AGE + SE
Normal Q−Q
Std. deviance resid.
621
−2
2
4
−3
−1
1
2
3
Theoretical Quantiles
lfr ~ dbase + bascre + baseu + AGE + SE
Scale−Location
√|Std. deviance resid.|
621
0.0
1.0
2.0
−40
0
40
Predicted values
lfr ~ dbase + bascre + baseu + AGE + SE
Residuals vs Leverage
Std. Pearson resid.
Cook's distance
−2
2
4
0.00
0.02
0.04
Leverage
lfr ~ dbase + bascre + baseu + AGE + SE
Residuals vs Fitted
Residuals
558
−4
−2
0
2
1
2
3
4
Predicted values
gfr ~ dbase + bascre + baseu + AGE + S
Normal Q−Q
Std. deviance resid.
558
−8
−4
0
4
−3
−1
1
2
3
Theoretical Quantiles
gfr ~ dbase + bascre + baseu + AGE + S



(3) As there weren't many visual clues of interaction terms to include, I decided to use my brief research to hypothesize a few interaction terms and investigate if they were useful. I decided to test race and sex's interaction with all other variables. These seemed like the most likely to interact with other variables. When doing this, the interactions of baseu with black and baseu with SEX produced significant results. I tried including these interaction terms in my model and they remained significant. They also reduced AIC and ANOVA revealed that this updated model was significantly different than the original.

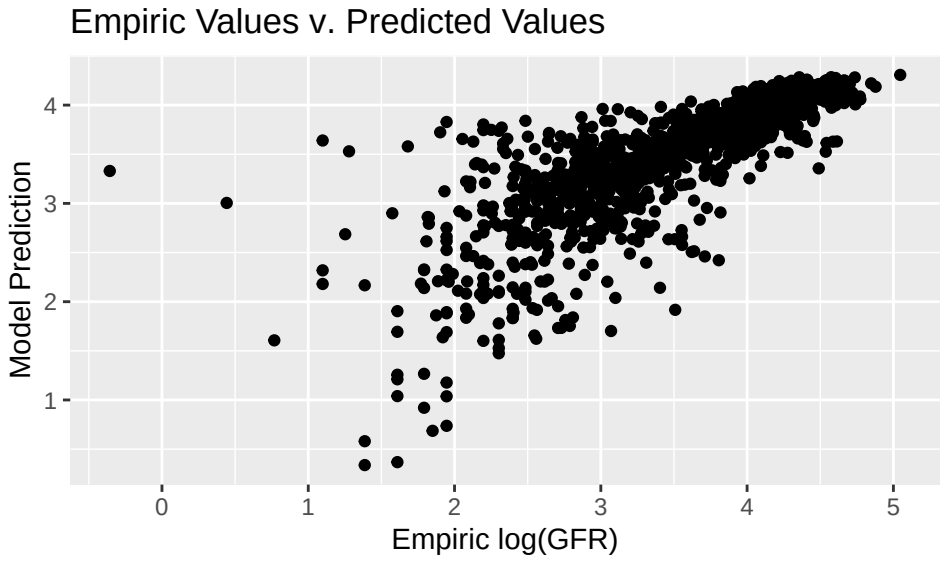
Final Model Decision

Therefore, my final model proposal is as follows :

log(gfr) ~ dbase + bascre + (baseu * SEX) + (baseu * black) + AGE

Model Output:

	Estimate	Std. Error	T-value	P-Value
Intercept	4.4448779	0.1301299	34.157	2e-16
dbase	0.0004369	0.0012002	0.364	0.715889
bascre	-0.4381677	0.0100053	-43.794	2e-16
baseu	-0.0868579	0.0141462	-6.14	1.11e-09
Sex	0.1052609	0.0356762	2.95	0.003233
Black/Non Black	-0.0284902	0.0533463	-0.534	0.593396
Age	0.0005805	0.0010857	0.535	0.592948
Int, baseu:Sex	0.0539947	0.0151636	3.561	0.000384
Int baseu:Black/Non Black	0.0650867	0.0243217	2.676	0.007547



Interpretation of Model

Due to their significant p-values in my model, I have determined that serum creatinine (bascre), urine protein (baseu), and sex are predictive factors of gfr. I would interpret the results of this model by saying that the average GFR when all predictive variables are zero is eˆ4.44 or 84.77. An example of how to interpret a coefficient follows, eˆ-0.44 (or 0.645) is the average change in GFR for an increase in serum creatinine (bascre) by one when all other variables are zero. Recall that we must be sure to exponentiate the coefficients to correctly adjust for the transformation of GFR to log(GFR). The interpretation of the urine protein (baseu) and sex follow this format. Lastly, the method for interpreting the interaction terms follows. eˆ0.0539947 (or 1.055) is the difference in rates of association of GFR and urine protein (baseu) between males and females. The same format applies for the baseu:Black/Non Black interaction term.

Part B

This exercise will allow you to explore the phenomenon of overfitting in linear regression through simulation.

Question 1

Part A

Simulate a random vector y of length 100 in which each element follows a normal distribution with mean 10 and variance 4. Then simulate a random vector x of length 100 with mean 3 and variance 1.

```
library(arm)
y <- rnorm (100, mean=10, sd=4)
x <- rnorm (100, mean=3, sd=1)
```

Part B

Regress y on x and save the p-value for the slope (HINT: use the summary.lm function or summary(lmobject) and use the coef element of the summary object if using R).

```r
# regress y on x
lm1 <- lm(y ~ x)
# find and store p-value for slope as pval.slope
hold <- summary(lm1)[4]
pval.slope <- hold$coefficients[2,4]
```

Part C

What do you think this p-value should be?

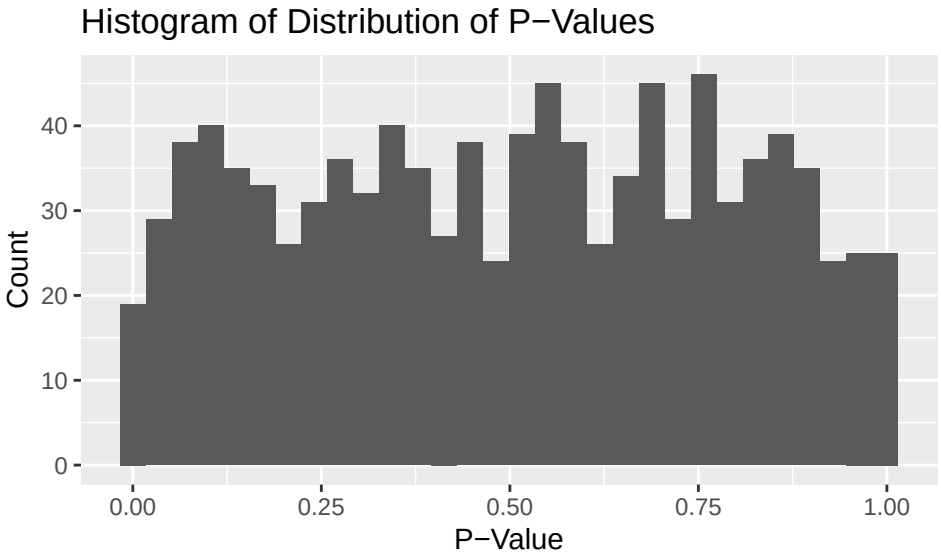
The p-value should be a random sample from the uniform distribution from [0,1] since X and Y are independent. This means they have no relationship except when they coincidentally align by random chance. Therefore the p-value is simply a random pull from the uniform distribution from [0,1]. See demonstration of distribution of p-values in Part D below once we run 1,000 of these simulations.

Part D

Now replicate this procedure 1000 times to develop a simulation.

```r
# 1,000 simulations
sims <- 1000
# initialize pvals to hold the 1,000 p-values
pvals <- vector("list", length(1000))
# for loop to run determined number of simulations
for (i in 1:sims){
  # sample y and x
  y <- rnorm (100, mean=10, sd=4)
  x <- rnorm (100, mean=3, sd=1)
  # build linear model for specific sample
  lm2 <- lm (y ~ x)
  # find and store p-value for slope as pval.slope
  hold <- summary(lm2)[4]
  pval.slope <- hold$coefficients[2,4]
  # store in corresponding index of vector/list
  pvals[i] <- pval.slope
}

# build plot for distribution of p-values demonstration
pvals.df <- as.data.frame(as.numeric(pvals))
ggplot(pvals.df) +
  geom_histogram(aes(pvals.df[,1])) +
  labs(x ="P-Value", y = "Count", title="Histogram of Distribution of P-Values")
```



Part E

Calculate the proportion of times the p-value is less than 0.05. How does this match with your intuition? [HINT: Should the slope be related to the outcome? What type of error are you simulating?]

The slope is currently not related to the outcome as X and Y are independent as previously discussed. This again means that the p-value will take on a uniform distribution, and therefore we would expect it to be at or below 0.05 about 5% of the time. This is a simulation of Type I error: we conclude there is indeed a relationship 5% of the time when there is actually not (a false positive).

I did not set a seed, so the above simulations will be different every time, but I ran it a few times and had proportions of 0.36, 0.42, and 0.38.

Part F

Now simulate y and x together so that the conditional mean of y given x is 10+x and the conditional variance is 1. Repeat b-e above for this distribution. How does your answer differ from the first simulation? What are you simulating now? [HINT: Consider if the slope is actually related to the outcome]

Now the slope is directly related to the outcome, since Y is dependent on X, so we expect see the p-value below 0.05 100% of the time (indicating that X and Y are related in each of the simulations). And this is indeed what we see!

Question 2

Now we will investigate overfitting which occurs when you are making too complex a model. First we will simulate some data in which y and multiple x's are unrelated.

Part A

Extend the function you have written to carry out exercise 1 so that you now generate the same y vector as in 1a (i.e. 100 draws from N(10,4)) but you now generate 100 draws from a 3-variate multivariate normal X matrix with means 1, 2, 3 and covariance matrix equal to the identity matrix of rank 3 (i.e., each X variable has variance 1 and they are uncorrelated). [HiNT: use the mvrnorm function in the MASS library to generate multivariate normal draws].

See code below.

Part B

Regress y on the X matrix and save the p-values from this regression. In R if you specify X as a matrix in the call to the lm() function, it will automatically use the formula y~x[,1]+x[,2]+... i.e. it will use the columns of the matrix as the predictors.

See code below.

```
library(MASS)
sims <- 1000
sigma <- diag(3)
pvalsfirst <- vector("list", length(1000))
pvalssecond <- vector("list", length(1000))
pvalsthird <- vector("list", length(1000))

for (i in 1:sims){
  x <- mvrnorm (100, mu=c(1,2,3), sigma)
  y <- rnorm (100, mean=10, sd=4)
  lm3 <- lm (y ~ as.matrix(x))
  hold <- summary(lm3)[4]

  pvalsfirst[i] <- hold$coefficients[2,4]
  pvalssecond[i] <- hold$coefficients[3,4]
  pvalsthird[i] <- hold$coefficients[4,4]
}
```

Part C

Determine how many of the p-values for each variable are significant at the 0.05 level.

I again did not set a seed, but one go of a set of simulations produced 40 significant p-values for variable 1, 38 significant p-values for variable 2, and 47 significant p-values for variable 3.

Part D

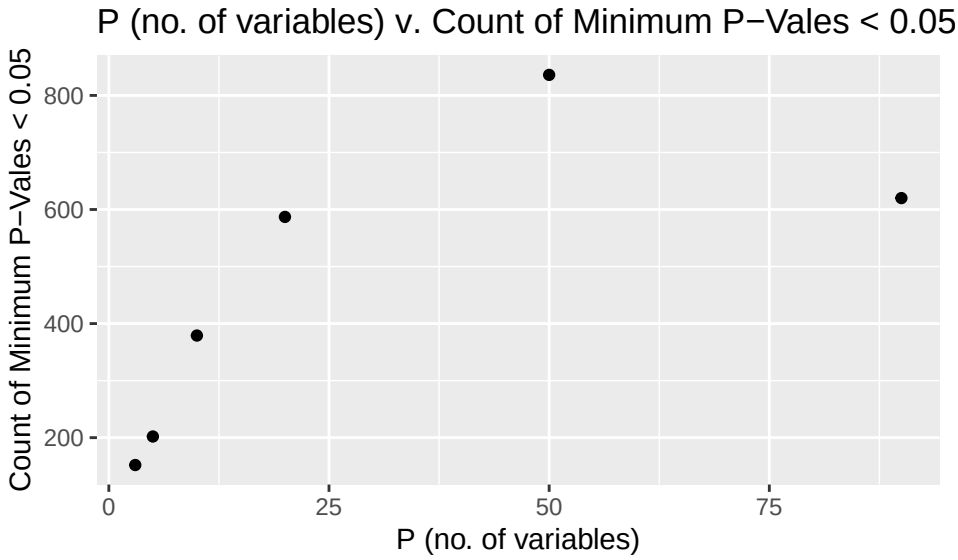
Compute the minimum p-value for each regression and calculate how many times the minimum p-value is less than 0.05. Why is this answer different from those in question c?

I counted from the set of p-values mentioned directly above in Part C, and the minimum p-value is less that 0.05 112 times. As you add in more variables, each will randomly have a non-zero relationship with the outcome 5% of the time, and these will be assorted randomly across the simulations. Therefore they will not necessary line up across the three variables, but we only need one, the minimum, to be at or below 0.05 to consider the relationship non-zero. We can think of these as "bonus" counts. Specifically, the number of simulations with *at least* one significant p-value will increase to about 1-(0.95^3), which is about 0.1426 and is near the count I produced.

Part E

Now repeat this exercise changing the number of predictors P so that you do it for P = 3, 5, 10, 20, 50, 90. Compute the minimum p-value in each simulated regression and plot the number of times you find at least one significant variable for each P. What pattern do you see? How do you explain this?

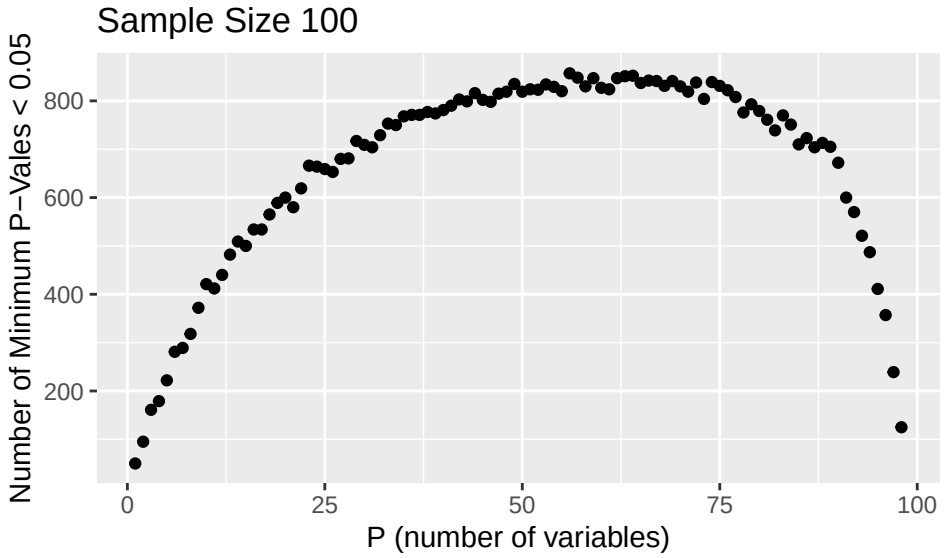
At first the count increases drastically, as adding more variables increases the chances of seeing at least one significant p-value. However, once we hit 90, we see a decrease in count of significant p-values. As you increase P, random noise causes the likelihood of collinearity between P's to increase. Continue reading for further explanation.



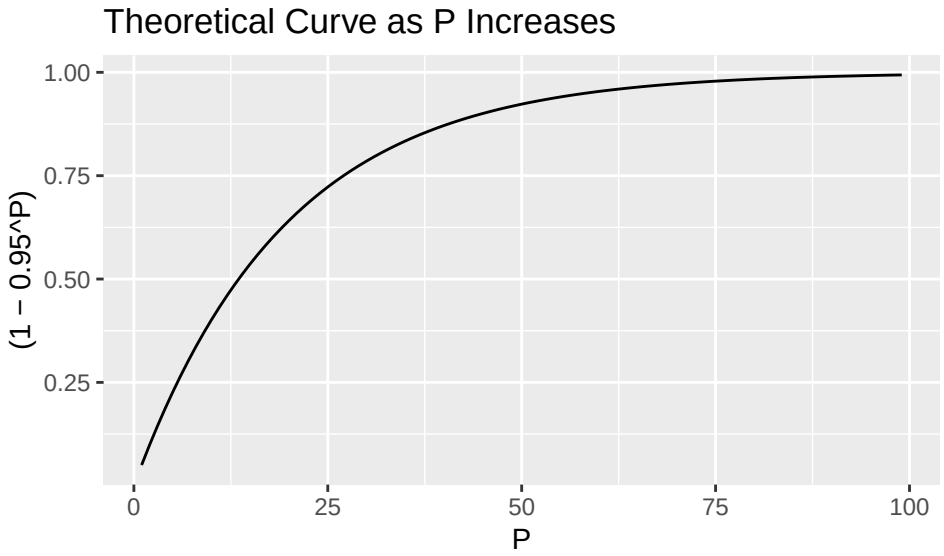
Part F

Set P equal to every number from 1 to 99 (i.e. you try models with all possible number of predictors) and run more simulations to reduce simulation error.

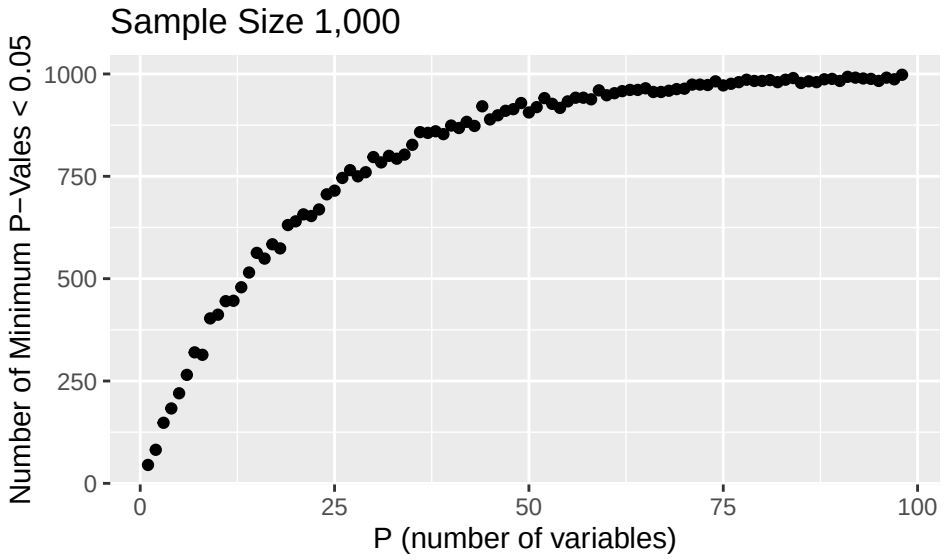
Below we see more clearly the pattern described in Part E.



Now let us graph the theoretical curve that we might expect to see without any colinearity. We graph P on the x-axis and (1-(0.95^P)) on the y-axis.



You can lower probability of seeing random colinearity by taking more samples. More samples reduces the chance of seeing colinearity, even as you increase the number of variables. See below for a graph where I took 1000 examples instead of 100. If we were able to take infinite samples, we would be able to produce the values of the theoretical curve above.



Code Appendix

```
knitr::opts_chunk$set(echo = TRUE)
setwd("~/Desktop/masters/PHP2550/")
library(dplyr)
library(tidyr)
library(ggplot2)

#read in data set
baseseg <- read.csv("baseseg.csv")
#reduce to variables of interest
```

```r
baseseg.red <- baseseg %>%
  select(1,4:7,22:23,26, 40)

# Missing Values Investigation
no.missing.gfr <- sum(is.na(baseseg.red$gfr))
# There are 611 observations where gfr is encoded as NA. This is about 1/3 of the total data set. We ha

baseseg.red.no.na <- na.omit(baseseg.red)

# change discrete variables into factors for graphing ease
baseseg.red.no.na$SEXfactor = as.factor(ifelse(baseseg.red.no.na$SEX==1, "male", "female"))

baseseg.red.no.na$blackfactor = as.factor(ifelse(baseseg.red.no.na$black==1, "black", "non-black"))
# Add transformed outcome
baseseg.red.no.na$loggfr <- log(baseseg.red.no.na$gfr)

ggplot(baseseg.red.no.na) + geom_density(aes(x=gfr))

ggplot(baseseg.red.no.na) + geom_density(aes(x=loggfr))
# Which variables to include, linear relationships? or transformations necessary?
library(GGally)
ggpairs(baseseg.red.no.na, columns=c("sbase","dbase","bascre","baseu","AGE", "SEXfactor", "blackfactor"
    diag=list(continuous="density", discrete="bar"), axisLabels="show")
# - interaction term only seems viable for race and gender, otherwise neither seem to have any ~visual~

# want to decide which variables to include, which require a transfomation (either on predictors or out
# AIC, BIC, ANOVA
# multicolinearity (vif)

#first try for baseline
modela <- glm(gfr ~ sbase + dbase + bascre + baseu + AGE + SEX + black, family = gaussian, baseseg.red.n
#summary(modela)
#vif(modela)
#plot(modela)

#remove sbase and vif, AIC goes down
modela <- glm(gfr ~ dbase + bascre + baseu + AGE + SEX + black, family = gaussian, baseseg.red.no.na)
#summary(modela)
plot(modela)

#transform output to see if that improves diagnostic plots, also *really* improves AIC, helps with resi
model.log.out <- glm(loggfr ~ dbase + bascre + baseu + AGE + SEX + black, family = gaussian, baseseg.red
#summary(model.log.out)
plot(model.log.out)

model.big.int<- glm(loggfr ~ (dbase + bascre + baseu + AGE) * SEX * black, family = gaussian, baseseg.re
#summary(model.big.int)
#plot(model.big.int)

model.few.int<- glm(loggfr ~ dbase + bascre + baseu*SEX + baseu*black + AGE, family = gaussian, baseseg
#summary(model.few.int)
#plot(model.big.int)
```

```r
#anova(model.few.int, model.log.out, test="LRT")

# tells us that there is indeed a significant difference between the two models
#anova(model.log.out.int, model.log.out, test="LRT")

#final model:

model.final<- glm(loggfr ~ dbase + bascre + baseu*SEX + baseu*black + AGE, family = gaussian, baseseg.re
store <- summary(model.final)
#plot(model.big.int)

model.output.df <- as.data.frame(rbind(c(4.4448779, 0.1301299, 34.157, "2e-16"), c(0.0004369,  0.001200
colnames(model.output.df) <- c("Estimate", "Std. Error", "T-value", "P-Value")
rownames(model.output.df) <- c("Intercept", "dbase", "bascre", "baseu", "Sex", "Black/Non Black", "Age"

library(tidyverse)
library(knitr)
library(kableExtra)

#build table
kable(model.output.df) %>%
  kable_styling(bootstrap_options = "striped", full_width = F, position = "center")
baseseg.red.no.na$predictions <- predict(model.final)
ggplot(baseseg.red.no.na) +
  geom_point(aes(x=loggfr, y=predictions)) +
  labs(title="Empiric Values v. Predicted Values", x="Empiric log(GFR)", y="Model Prediction")

library(arm)
y <- rnorm (100, mean=10, sd=4)
x <- rnorm (100, mean=3, sd=1)
# regress y on x
lm1 <- lm(y ~ x)
# find and store p-value for slope as pval.slope
hold <- summary(lm1)[4]
pval.slope <- hold$coefficients[2,4]
# 1,000 simulations
sims <- 1000
# initialize pvals to hold the 1,000 p-values
pvals <- vector("list", length(1000))
# for loop to run determined number of simulations
for (i in 1:sims){
  # sample y and x
  y <- rnorm (100, mean=10, sd=4)
  x <- rnorm (100, mean=3, sd=1)
  # build linear model for specific sample
  lm2 <- lm (y ~ x)
  # find and store p-value for slope as pval.slope
  hold <- summary(lm2)[4]
  pval.slope <- hold$coefficients[2,4]
  # store in corresponding index of vector/list
  pvals[i] <- pval.slope
}
```

```r
# build plot for distribution of p-values demonstration
pvals.df <- as.data.frame(as.numeric(pvals))
ggplot(pvals.df) +
  geom_histogram(aes(pvals.df[,1])) +
  labs(x ="P-Value", y = "Count", title="Histogram of Distribution of P-Values")
index.for.count <- 0
for (i in 1:length(pvals)) {
  if (pvals[i]<0.05) {
  index.for.count <- index.for.count +1
  }
}
x <- rnorm (100, mean=3, sd=1)
y <- rnorm (100, mean=10+x, sd=1)

lm3 <- lm(y ~ x)
#summary(lm3)
hold2 <- summary(lm3)[4]
pval.slope2 <- hold$coefficients[2,4]

sims <- 1000
pvals2 <- vector("list", length(1000))
for (i in 1:sims){
  x <- rnorm (100, mean=3, sd=1)
  y <- rnorm (100, mean=10+x, sd=1)
  lm2 <- lm (y ~ x)
  hold <- summary(lm2)[4]
  pval.slope <- hold$coefficients[2,4]
  pvals2[i] <- pval.slope
}

index.for.count2 <- 0
for (i in 1:length(pvals2)) {
  if (pvals2[i]<0.05) {
  index.for.count2 <- index.for.count2 +1
  }
}
library(MASS)
sims <- 1000
sigma <- diag(3)
pvalsfirst <- vector("list", length(1000))
pvalssecond <- vector("list", length(1000))
pvalsthird <- vector("list", length(1000))

for (i in 1:sims){
  x <- mvrnorm (100, mu=c(1,2,3), sigma)
  y <- rnorm (100, mean=10, sd=4)
  lm3 <- lm (y ~ as.matrix(x))
  hold <- summary(lm3)[4]

  pvalsfirst[i] <- hold$coefficients[2,4]
  pvalssecond[i] <- hold$coefficients[3,4]
  pvalsthird[i] <- hold$coefficients[4,4]
}
```

```
# count for all variables
count.first <- 0
count.second <- 0
count.third <- 0
for (i in 1:length(pvalsfirst)) {
  if (pvalsfirst[i]<0.05) {
  count.first <- count.first +1
  }
  if (pvalssecond[i]<0.05) {
  count.second <- count.second +1
  }
  if (pvalsthird[i]<0.05) {
  count.third <- count.third +1
  }
}
all.p.vals <- cbind(pvalsfirst, pvalssecond, pvalsthird)
all.p.vals <- data.frame(all.p.vals)

pvalsmin <- vector("list", length(1000))
for (i in 1:nrow(all.p.vals)) {
  pvalsmin[i] <- min(unname(unlist(all.p.vals[i,])))
}

count.pval.mins <- 0
for (i in 1:length(pvalsmin)) {
  if (pvalsmin[i]<0.05) {
  count.pval.mins <- count.pval.mins +1
  }
}
library(MASS)
sims <- 1000
p = c(3, 5, 10, 20, 50, 90)

pvalsminperp <- vector("list", length(p))

for (pi in 1:length(p)) {
  #get all pvals for one regression w P variables and store in pvalstotal
  pvalstotal <- matrix(NA, nrow=sims, ncol=p[pi])
  for (i in 1:sims){
    sigma = diag(p[pi])
    meu = c(1:p[pi])
    x <- mvrnorm (100, mu=meu, sigma)
    y <- rnorm (100, mean=10, sd=4)
    lm4 <- lm (y ~ as.matrix(x))
    hold <- summary(lm4)[4]
    #print(hold)
    for (ii in 1:p[pi]){
      pvalstotal[i,ii] <- hold$coefficients[(ii+1),4]
    }

  }
  #print(pvalstotal)
  # build vector of minimum pval per regression
```

```
  pvalsmin <- vector("list", length(sims))

  for (i in 1:nrow(pvalstotal)) {
  pvalsmin[i] <- min(unname(unlist(pvalstotal[i,])))
  }
  #print(pvalsmin)

  count.pval.mins <- 0
  for (i in 1:length(pvalsmin)) {
    if (pvalsmin[i]<0.05) {
    count.pval.mins <- count.pval.mins +1
    }
  }
  #print(count.pval.mins)
  pvalsminperp[pi] <- count.pval.mins
}

minpvalsperpplot <- cbind(pvalsminperp, c(3, 5, 10, 20, 50, 90))
minpvalsperpplotdf <- data.frame(minpvalsperpplot)
colnames(minpvalsperpplotdf) <- c("pvalsminperp", "p")

library(ggplot2)
ggplot(minpvalsperpplotdf) +
  labs(title="P (no. of variables) v. Count of Minimum P-Vales < 0.05", x= "P (no. of variables)", y =
  geom_point(aes(x=as.numeric(minpvalsperpplotdf$p), y=as.numeric(minpvalsperpplotdf$pvalsminperp)))
library(MASS)
sims <- 1000
p = c(1:98)

pvalsminperp <- vector("list", length(p))

for (pi in 1:length(p)) {
  #get all pvals for one regression w P variables and store in pvalstotal
  pvalstotal <- matrix(NA, nrow=sims, ncol=p[pi])
  for (i in 1:sims){
    sigma = diag(p[pi])
    meu = c(1:p[pi])
    x <- mvrnorm (100, mu=meu, sigma)
    y <- rnorm (100, mean=10, sd=4)
    lm4 <- lm (y ~ as.matrix(x))
    hold <- summary(lm4)[4]
    #print(hold)
    for (ii in 1:p[pi]){
      pvalstotal[i,ii] <- hold$coefficients[(ii+1),4]
    }
  }
  # build vector of minimum pval per regression
  pvalsmin <- vector("list", length(sims))

  for (i in 1:nrow(pvalstotal)) {
  pvalsmin[i] <- min(unname(unlist(pvalstotal[i,])))
  }
  count.pval.mins <- 0
```

```
  for (i in 1:length(pvalsmin)) {
    if (pvalsmin[i]<=0.05) {
    count.pval.mins <- count.pval.mins +1
    }
  }
  pvalsminperp[pi] <- count.pval.mins
}

minpvalsperpplot <- cbind(pvalsminperp, c(1:98))
minpvalsperpplotdf <- data.frame(minpvalsperpplot)
colnames(minpvalsperpplotdf) <- c("pvalsminperp", "p")

library(ggplot2)
ggplot(minpvalsperpplotdf) +
  labs(title="Sample Size 100", x= "P (number of variables)", y = "Number of Minimum P-Vales < 0.05") +
  geom_point(aes(x=as.numeric(minpvalsperpplotdf$p), y=as.numeric(minpvalsperpplotdf$pvalsminperp)))
#theoretical curve
P = c(1:99)
ggplot(data=NULL) + geom_line(aes(x=P, y= (1-0.95^P))) +
  labs(title="Theoretical Curve as P Increases")
sims <- 1000
p = c(1:98)

pvalsminperp2 <- vector("list", length(p))

for (pi in 1:length(p)) {
  #get all pvals for one regression w P variables and store in pvalstotal
  pvalstotal <- matrix(NA, nrow=sims, ncol=p[pi])
  for (i in 1:sims){
    sigma = diag(p[pi])
    meu = c(1:p[pi])
    x <- mvrnorm (1000, mu=meu, sigma)
    y <- rnorm (1000, mean=10, sd=4)
    lm4 <- lm (y ~ as.matrix(x))
    hold <- summary(lm4)[4]
    #print(hold)
    for (ii in 1:p[pi]){
      pvalstotal[i,ii] <- hold$coefficients[(ii+1),4]
    }
  }
  # build vector of minimum pval per regression
  pvalsmin <- vector("list", length(sims))

  for (i in 1:nrow(pvalstotal)) {
  pvalsmin[i] <- min(unname(unlist(pvalstotal[i,])))
  }
  count.pval.mins <- 0
  for (i in 1:length(pvalsmin)) {
    if (pvalsmin[i]<=0.05) {
    count.pval.mins <- count.pval.mins +1
    }
  }
  pvalsminperp2[pi] <- count.pval.mins
```

```
}

minpvalsperpplot2 <- cbind(pvalsminperp2, c(1:98))
minpvalsperpplotdf2 <- data.frame(minpvalsperpplot2)
colnames(minpvalsperpplotdf2) <- c("pvalsminperp", "p")

library(ggplot2)
ggplot(minpvalsperpplotdf2) +
  labs(title="Sample Size 1,000", x= "P (number of variables)", y = "Number of Minimum P-Vales < 0.05") +
  geom_point(aes(x=as.numeric(minpvalsperpplotdf2$p), y=as.numeric(minpvalsperpplotdf2$pvalsminperp)))
```