MATH 3190 Homework 5

Focus: Notes 7 Part 2

Due March 16, 2024

Your homework should be completed in R Markdown or Quarto and Knitted to an html or pdf document. You will "turn in" this homework by uploading to your GitHub Math_3190_Assignment repository in the Homework directory.

Problem 1 (20 points)

Suppose $\boldsymbol{x} = (x_1, \ldots, x_n)^T$ are independent and identically distributed random variables with probability density function (pdf) given by

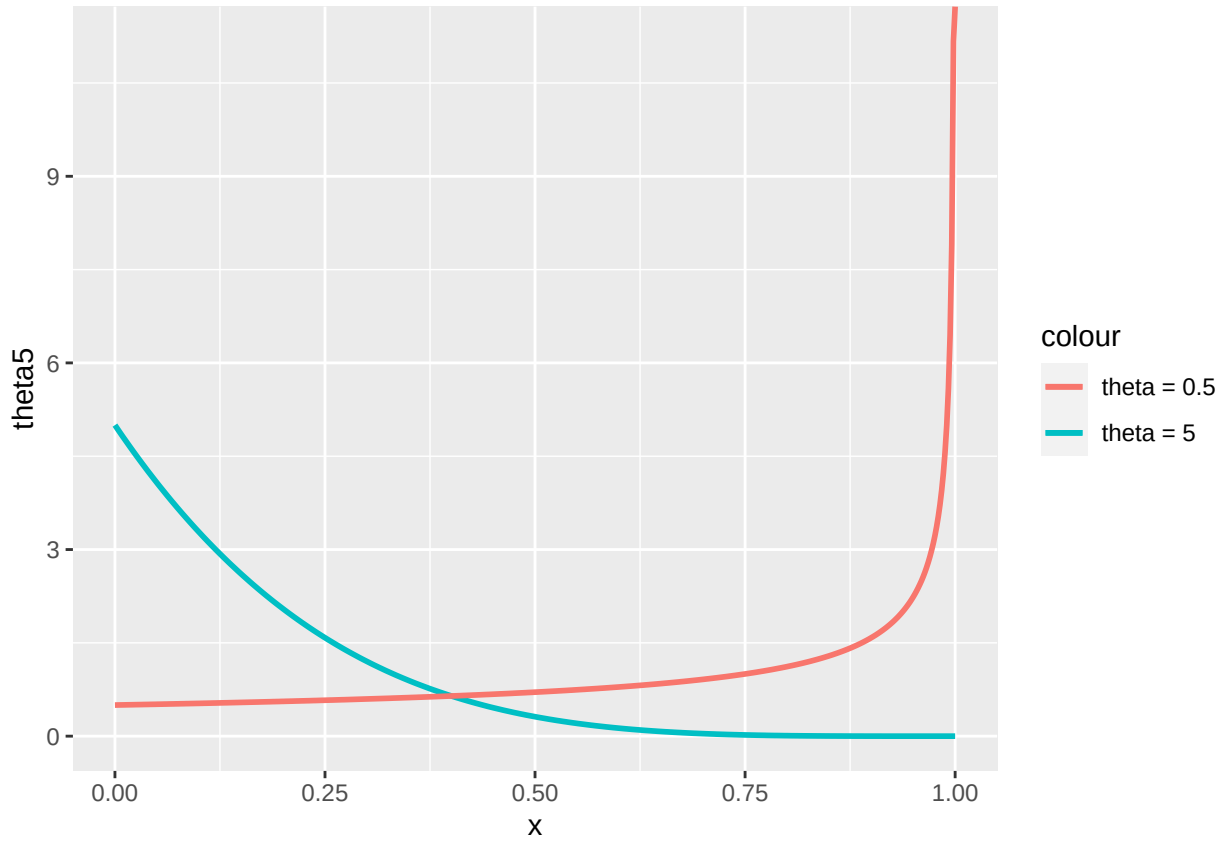
$$f(x_i|\theta) = \theta(1 - x_i)^{\theta - 1}; \;\; 0 \leq x_i \leq 1, \; 0 < \theta < \infty, \; i = 1, \ldots, n$$

Part a (4 points)

Using ggplot, plot the pdf for an individual x_i given $\theta = 0.5$ and then again for $\theta = 5$.

```
pdf <- tibble(x = seq(0, 1, length.out = 500)) |>
  mutate(theta5 = 5*(1 - x)^(5 -1),
         theta0.5 = 0.5*(1 - x)^(0.5 -1))
```

```
ggplot(pdf) +
  geom_line(aes(x, theta5, color = "theta = 5"), lwd = 1) +
  geom_line(aes(x, theta0.5, color = "theta = 0.5"), lwd = 1)
```



Part b (5 points)

Give the likelihood $L(\theta|\boldsymbol{x})$ and log-likelihood $\ell(\theta|\boldsymbol{x})$ functions in this situation.

$L(\theta|\boldsymbol{x}) = (\theta^n \prod_{i=1}^n (1-x_i)^{\theta-1})$

$\ell(\theta|\boldsymbol{x}) = \ell(\theta^n \prod_{i=1}^n (1-x_i)^{\theta-1})$

Part c (4 points)

Find the Maximum Likelihood Estimator (MLE) $\hat{\theta}$ for θ.

$\frac{dy}{d\theta} = \frac{n}{\theta} + \sum_{i=1}^n \log(1-x_i)$

$\frac{n}{\theta} + \sum_{i=1}^n \log(1-xi) = 0$

$\frac{n}{\theta} = -(\sum_{i=1}^n \ell(1-x_i))$

$\hat{\theta} = -(\frac{n}{\sum_{i=1}^n \log(1-x_i)})$

Part d (2 points)

Show that your estimator is in fact a maximum. That is, check the second derivative of the log-likelihood.

$$\frac{d^2y}{d\theta^2} = -\frac{n}{\theta^2}$$

Because the denominator is squared, this derivative will always evaluate to < 0. Therefore, it is a maximum

Part e (1 point)

Find the MLE if the data values are given below. Note, the actual θ value used to generate these data was $\theta = 7.3$.

```
x <- c(0.0194, 0.0053, 0.2488, 0.0456, 0.2059, 0.0992, 0.1168, 0.3705,
       0.2129, 0.018, 0.0464, 0.1401, 0.0759, 0.1588, 0.0334, 0.2931,
       0.0662, 0.2292, 0.1581, 0.3462, 0.035, 0.1086, 0.0793, 0.2095,
       0.1419, 0.1835, 0.1107, 0.0764, 0.1331, 0.042, 0.0911, 0.1608)
```

```
mle <- -(length(x)/sum(log(1 - x)))
print(mle)
```

```
## [1] 6.706184
```

Part f (4 points)

To get an idea of how variable the maximum likelihood estimator is in this case, let's generate many samples of size 32 from this distribution, which is a Beta distribution with parameters 1 and θ. We can do this by using `rbeta(32, 1, 7.3)`.

Generate at least 10,000 samples this way, compute the MLE for each, and then plot a histogram of the values using ggplot. On the plot somewhere, indicate what the standard deviation in those estimates is.

Now repeat this but instead of taking samples of size 32, take samples of size 100. How does this change the histogram and standard deviation of the estimates?

```
to_vec(for (i in 1:10000){

  x <- rbeta(32, 1, 7.3)
  mle <- -(length(x)/sum(log(1 - x)))

}) -> mles32
sd32 <- sd(mles32) |>
  round(3)
```

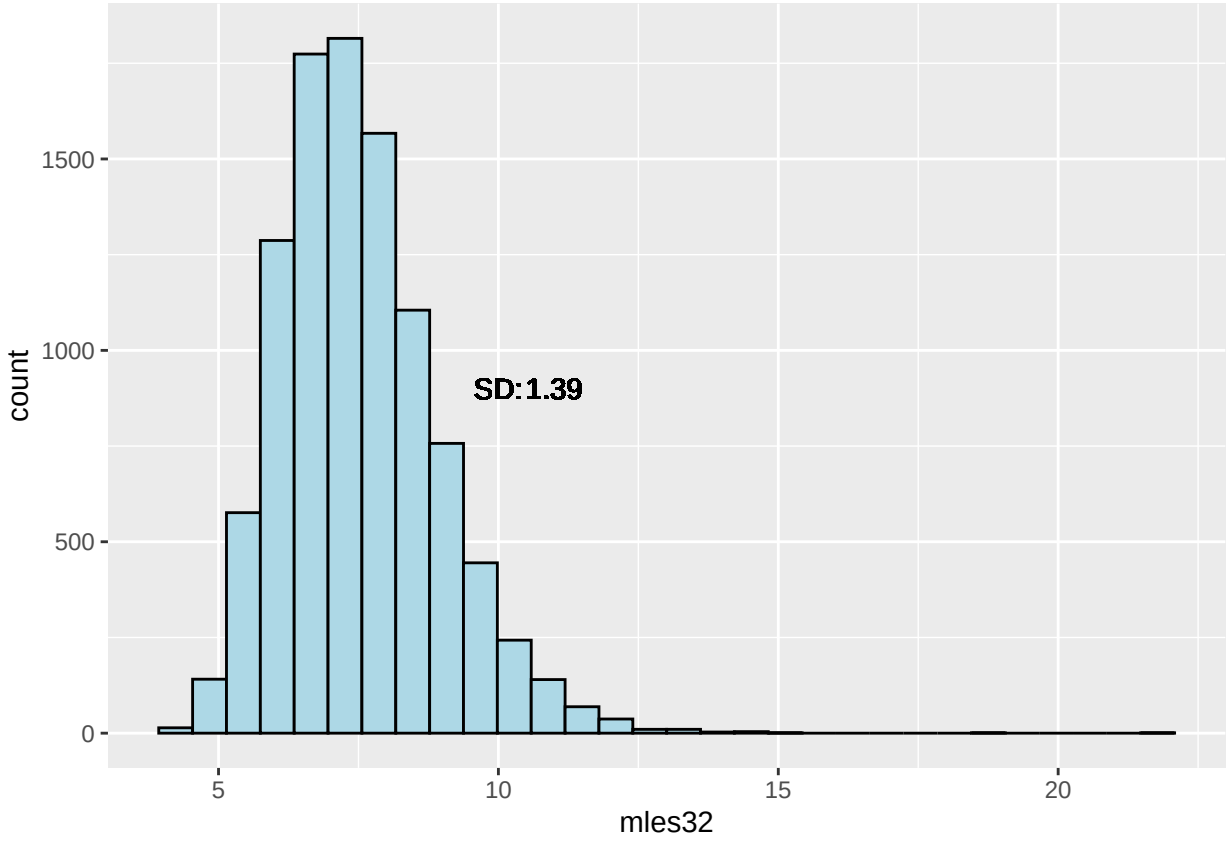
```
to_vec(for (i in 1:10000){

  x <- rbeta(100, 1, 7.3)
  mle <- -(length(x)/sum(log(1 - x)))

}) -> mles100
  sd100 <- sd(mles100) |>
  round(3)
```

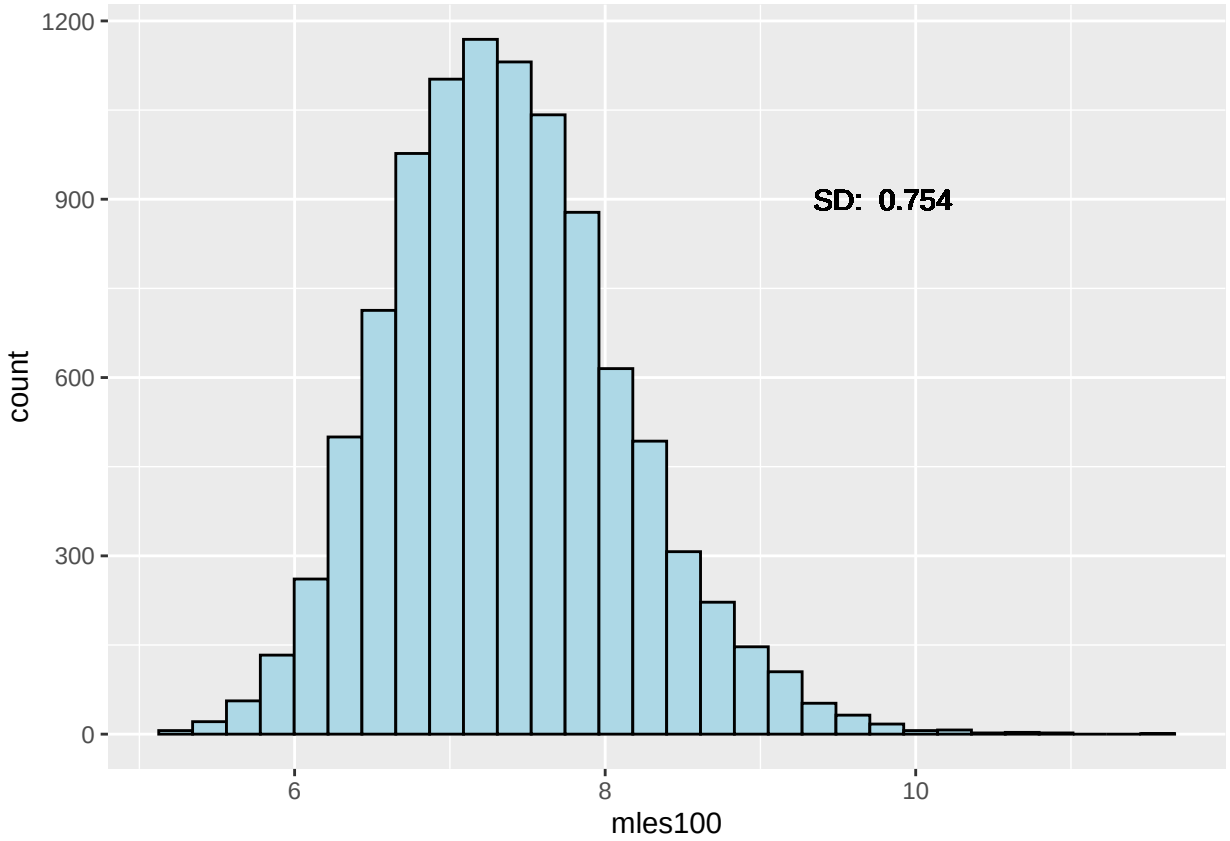
```
ggplot(as.data.frame(mles32), mapping = aes(x = mles32)) +
  geom_histogram(color = "black", fill = "lightblue") +
  geom_text(x = 11, y = 900, label = sd32) +
  geom_text(x = 10, y = 900, label = "SD:")
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```




```
ggplot(as.data.frame(mles100), mapping = aes(x = mles100)) +
  geom_histogram( color = "black", fill = "lightblue") +
  geom_text(x = 10, y = 900, label = sd100) +
  geom_text(x = 9.5, y = 900, label = "SD:")
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



When sample size is increased from 32 to 100, the histogram looks more normal (or at least less skewed), and the standard deviation decreases.

Problem 2 (35 points)

In the `AER` package is the data set "ShipAccidents". You can install that package and load that data using `data(ShipAccidents)`. Type `?ShipAccidents` to read about that data set.

```
data(ShipAccidents)
```

Part a (3 points)

Load in the data, remove the rows for which service is equal to 0, and then fit a Poisson regression model for predicting incidents from all other variables using the ShipAccidents data. Briefly explain why it makes sense to do Poisson regression here.

```
ShipAccidents <- ShipAccidents |>
  filter(service != 0)
```

```
shipMod <- glm(incidents ~., data = ShipAccidents, family = 'poisson')
summary(shipMod)
```

```
##
```

```
## Call:
## glm(formula = incidents ~ ., family = "poisson", data = ShipAccidents)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.8497  -1.4945  -0.4671   0.7131   2.9049
##
## Coefficients:
##                       Estimate Std. Error z value Pr(>|z|)
## (Intercept)          2.657e-01  2.776e-01   0.957 0.338454
## typeB                7.030e-01  2.187e-01   3.214 0.001308 **
## typeC               -1.196e+00  3.275e-01  -3.653 0.000259 ***
## typeD               -8.320e-01  2.877e-01  -2.892 0.003831 **
## typeE               -2.492e-01  2.360e-01  -1.056 0.291027
## construction1965-69  1.048e+00  1.798e-01   5.832 5.47e-09 ***
## construction1970-74  1.446e+00  2.264e-01   6.387 1.70e-10 ***
## construction1975-79  6.788e-01  2.774e-01   2.447 0.014399 *
## operation1975-79     7.031e-01  1.358e-01   5.178 2.24e-07 ***
## service              6.421e-05  8.593e-06   7.473 7.86e-14 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for poisson family taken to be 1)
##
##     Null deviance: 614.54  on 33  degrees of freedom
## Residual deviance:  77.45  on 24  degrees of freedom
## AIC: 195.32
##
## Number of Fisher Scoring iterations: 5
```

It makes sense to do Poisson regression because we want to predict the average number of incidents at certain values of x

Part b (4 points)

Using that model, interpret the slope of "service" and the slope of "construction1970-74" in original (not log) units.

```
exp(0.00006421)
```

```
## [1] 1.000064
```

```
exp(1.446)
```

```
## [1] 4.246096
```

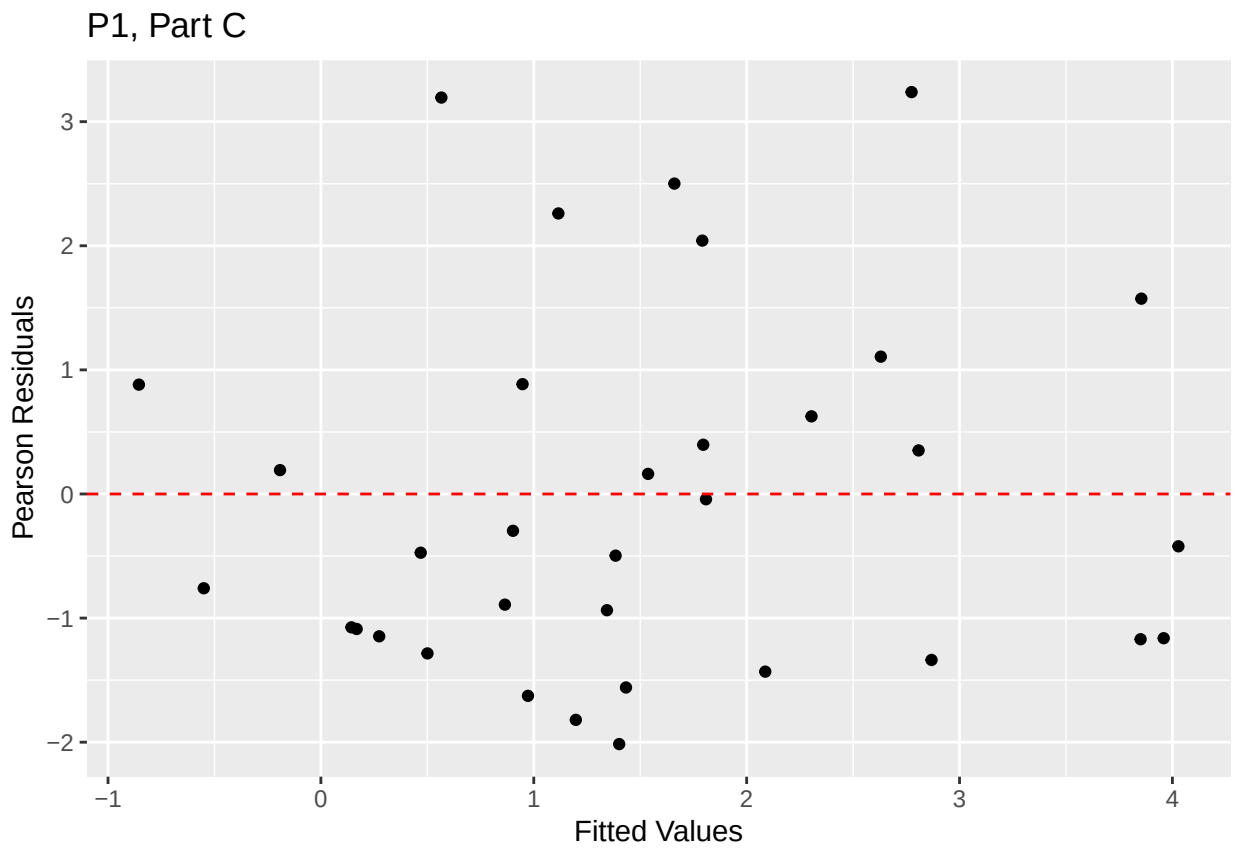
The predicted change in average incidents when months of service increases by one unit is 1.000064, holding all else equal. The predicted change in average incidents when the ship was constructed from 1970-1974 is 4.246, holding all else equal

Part c (5 points)

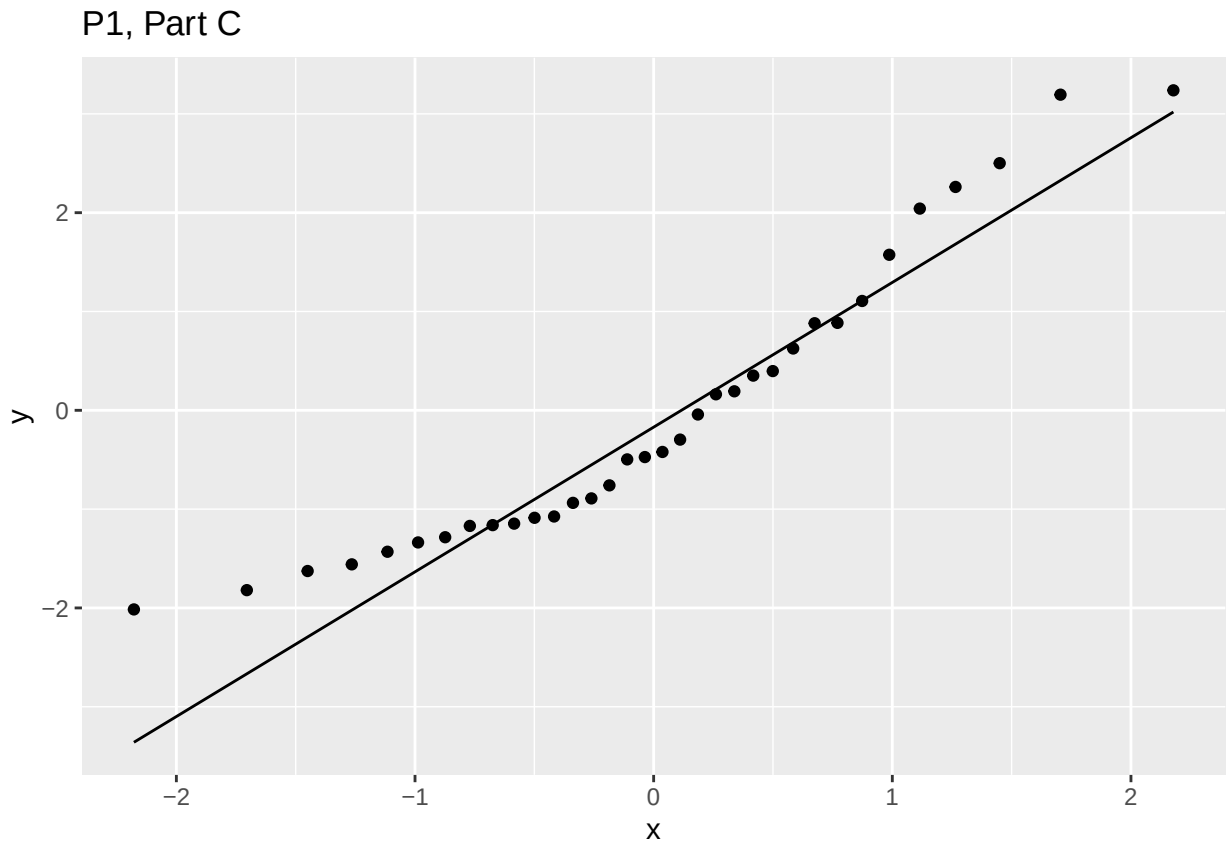
Make a residual plot of the Pearson residuals vs the linear predictors and make a QQ plot of the Pearson residuals. Comment on what these imply about the fit.

```
resids <- data.frame(residuals(shipMod, type = "pearson"))

ggplot(ShipAccidents, aes(x = predict(shipMod),
                          y = residuals(shipMod, type = "pearson"))) +
  geom_point() +
  geom_hline(yintercept = 0, lty = "dashed", color = "red") +
  labs(x = "Fitted Values", y = "Pearson Residuals") +
  ggtitle("P1, Part C")
```




```
ggplot(resids, aes(sample = resids[,1])) +
  geom_qq() +
  geom_qq_line() +
  ggtitle("P1, Part C")
```



The residuals do not look to be very normally distributed. There are only like, two points actually touching the line. The residual plot itself looks okay enough. It does not seem like the fit is great.

Part d (4 points)

Conduct a deviance goodness-of-fit (GOF) test for a lack of fit here. Type out the null and alternative hypotheses, report the test statistic, give the p-value, and provide an interpretation.

H_0: The model is correctly specified H_a: The model is incorrectly specified

```
paste("The test statistic is",shipMod$deviance)
```

[1] "The test statistic is 77.4497670809682"

```
paste("The p-value is", 1-pchisq(shipMod$deviance, shipMod$df.residual))
```

```
## [1] "The p-value is 1.54244734318354e-07"
```

There is very strong evidence that the model is incorrectly specified

Part e (2 points)

Is there evidence that this model has overdispersion? Explain.

```r
pResid <- sum(residuals(shipMod, type = "pearson")^2)

dfResid <- shipMod$df.residual

phi <- pResid/dfResid
print(phi)
```

```
## [1] 2.896271
```

Yes, there is. ϕ is much bigger than 1.25.

Part f (6 points)

Fit a "quasipoisson" model to these data. Using this model, obtain an approximate 95% confidence interval for the mean response for a ship of type "B" with construction between 1965 and 1969, with operation between 1960 and 1974, and with 2000 aggregate months of service. You can construct this interval using a t critical value with 24 degrees of freedom since the quasipoisson family more closely follows a t distribution than a normal one. Interpret the interval.

```r
qMod <- glm(incidents ~., data = ShipAccidents, family = "quasipoisson")
summary(qMod)
```

```
##
## Call:
## glm(formula = incidents ~ ., family = "quasipoisson", data = ShipAccidents)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.8497  -1.4945  -0.4671   0.7131   2.9049
##
## Coefficients:
##                       Estimate Std. Error t value Pr(>|t|)
## (Intercept)          2.657e-01  4.724e-01   0.562 0.579020
## typeB                7.030e-01  3.722e-01   1.889 0.071078 .
## typeC               -1.196e+00  5.574e-01  -2.146 0.042167 *
## typeD               -8.320e-01  4.896e-01  -1.699 0.102218
## typeE               -2.492e-01  4.017e-01  -0.620 0.540827
## construction1965-69  1.048e+00  3.059e-01   3.427 0.002206 **
## construction1970-74  1.446e+00  3.854e-01   3.753 0.000982 ***
## construction1975-79  6.788e-01  4.721e-01   1.438 0.163364
## operation1975-79     7.031e-01  2.311e-01   3.043 0.005604 **
## service              6.421e-05  1.462e-05   4.391 0.000196 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for quasipoisson family taken to be 2.896314)
##
##     Null deviance: 614.54  on 33  degrees of freedom
## Residual deviance:  77.45  on 24  degrees of freedom
## AIC: NA
##
## Number of Fisher Scoring iterations: 5
```

```
pred <- predict(qMod, data.frame(type = "B",
                                 construction = "1965-69",
                                 operation = "1960-74",
                                 service = 2000),
                type = 'response', se.fit = T)
tCrit <- qt((1-0.95)/2, 24)

lBound <- pred$fit + tCrit * pred$se.fit
uBound <- pred$fit - tCrit * pred$se.fit

print(lBound)
```

```
##        1
## 1.303744
```

```
print(uBound)
```

```
##       1
## 15.7899
```

We can say with 95% confidence that a ship of type B, constructed between 1965-1969, operating between 1960-1974, with 2000 months of service will have a mean response of incidents between 1.304 and 15.790

Part g (5 points)

Now construct an approximate 95% prediction interval for the number of incidents for the ship described in part f using the `qpois()` function. Interpret the interval.

```
qpois(0.025, lBound)
```

```
## [1] 0
```

```
qpois(0.975, uBound)
```

```
## [1] 24
```

We can say with 95% confidence that a ship of type B, constructed between 1965-1969, operating between 1960-1974, with 2000 months of service will have a true number of incidents between 0 and 24

Part h (6 points)

Suppose we did not catch the fact that this model is overdispersed. Repeat parts f and g using the model fit with the "poisson" family, not Poisson and using a z interval for the mean response rather than a t interval. Compare your results to parts f and g and comment on what is different.

```r
pred <- predict(shipMod, data.frame(type = "B",
                                    construction = "1965-69",
                                    operation = "1960-74",
                                    service = 2000),
                type = 'response', se.fit = T)
zCrit <- qnorm((1-0.95)/2)

lBound <- pred$fit + zCrit * pred$se.fit
uBound <- pred$fit - zCrit * pred$se.fit

print(lBound)
```

```
##        1
## 4.505159
```

```r
print(uBound)
```

```
##        1
## 12.58849
```

```r
qpois(0.025, lBound)
```

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

```r
qpois(0.975, uBound)
```

```
## [1] 20
```

Both intervals are quite a bit narrower. Also, the interval found using the quasipoisson function includes 0 as the lower bound, while the one found using just the poisson function does not include 0.

Problem 3 (45 points)

In Homework 4, we briefly looked at a small sample from this data set. Now we have the full data set with 20,640 districts. For districts in California from 1990, we want to predict the median house value for the district based on the district location (longitude and latitude), the median house age in the block group, the total number of rooms in the homes, the total number of bedrooms, the population in the district, the number of households, and the median income (in $10,000). These data were obtained from scikit-learn.org and can be found in the Data folder of the Math3190_Sp24 GitHub repo.

Part a (3 points)

Read in the data, assign appropriate column names to the columns and then take logs of all variables except lat and long. Once this is done, split the data into training and testing sets with the testing containing 20% of the values. Set a seed when you do this.

```
housing <- read_csv("cal_housing.data", col_names = F) |>
  rename(longitude = X1,
         latitude = X2,
         age = X3,
         rooms = X4,
         bedrooms = X5,
         population = X6,
         households = X7,
         income = X8,
         value = X9)
```

```
## Rows: 20640 Columns: 9
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## dbl (9): X1, X2, X3, X4, X5, X6, X7, X8, X9
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
for(i in c(3:9)){
  housing[,i] = log(housing[,i])
}
```

```
set.seed(1)
index <- createDataPartition(housing$value, times = 1, p = 0.8, list = F)
hTrain <- housing[index, ]
hTest <- housing[-index, ]
```

Part b (3 points)

Use the following command to view pair-wise scatter plots for the data here. Change `housing_train` to whatever you called the training set and be sure to change `eval = F` to `eval = T` in the option for the code chunk. Note, that `lower` option changes the points to be plotted with periods instead of circles. This speeds up plotting time considerably and makes the plots more readable since there are many data points here.

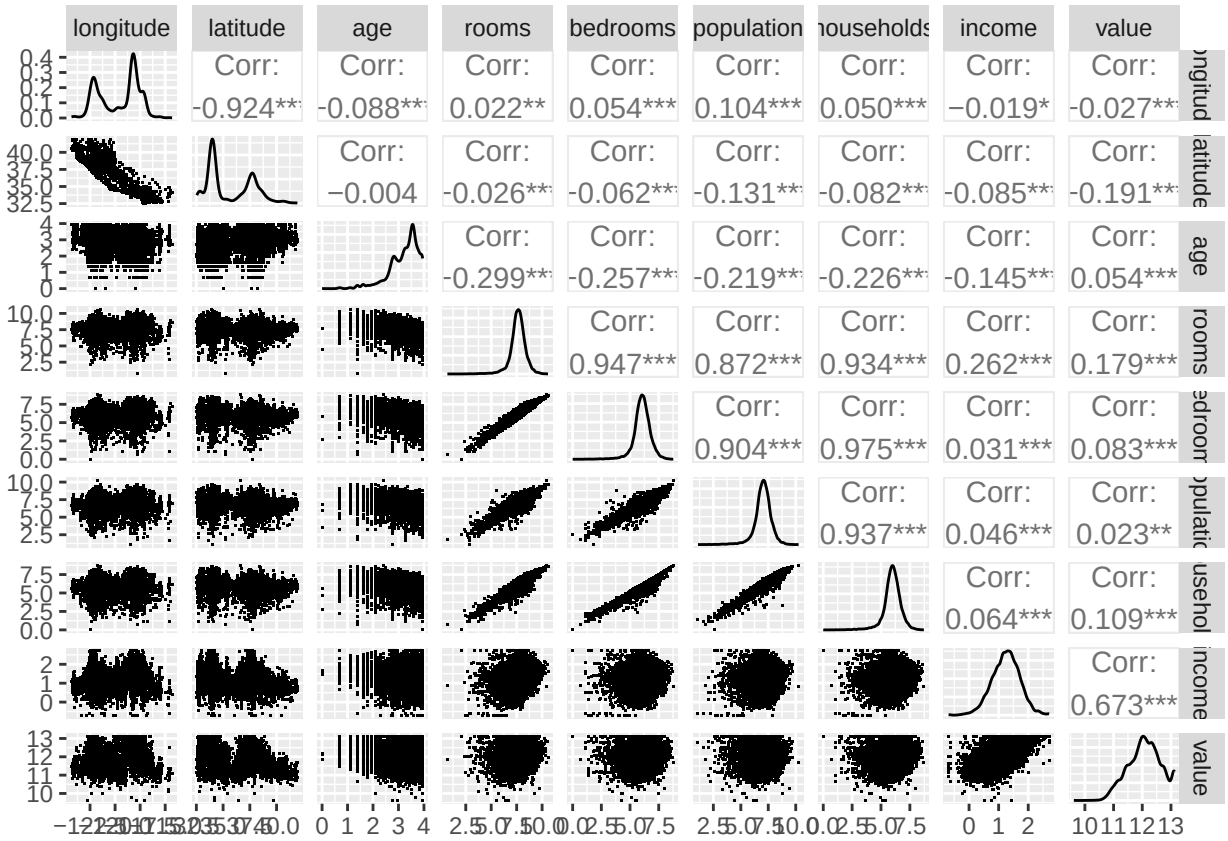
Based on these plots, does multicollinearity appear to be an issue here?

```
library(GGally)
```

```
## Warning: package 'GGally' was built under R version 4.2.3
```

```
## Registered S3 method overwritten by 'GGally':
##   method from
##   +.gg   ggplot2
```

```
ggpairs(hTrain,
        lower = list(continuous = wrap("points", shape = "."))
)
```



Absolutely it does. Basically none of the x variables except for income have a high correlation with house value, but plenty of the x-variables are very highly correlated with each other, such as population and rooms, bedrooms and household size, etc.

Part c (3 points)

Fit a OLS regression model predicting house prices from all other variables. Check its summary output and its VIF values. Comment on the VIFs.

```
houseMod <- lm(value ~ ., data = hTrain)
summary(houseMod)
```

```
##
## Call:
## lm(formula = value ~ ., data = hTrain)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.74698 -0.19330 -0.00339  0.18805  2.90841
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -9.963829   0.343523  -29.00   <2e-16 ***
## longitude   -0.260486   0.003893  -66.92   <2e-16 ***
## latitude    -0.266579   0.003760  -70.91   <2e-16 ***
```

```
## age          0.062630   0.004947   12.66   <2e-16 ***
## rooms       -0.283381   0.016908  -16.76   <2e-16 ***
## bedrooms     0.473827   0.021819   21.72   <2e-16 ***
## population  -0.419288   0.010202  -41.10   <2e-16 ***
## households   0.239919   0.020331   11.80   <2e-16 ***
## income       0.806242   0.008979   89.79   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.3268 on 16504 degrees of freedom
## Multiple R-squared:  0.6713, Adjusted R-squared:  0.6711
## F-statistic:  4213 on 8 and 16504 DF,  p-value: < 2.2e-16
```

```
vif(houseMod)
```

```
##  longitude   latitude        age      rooms   bedrooms population households
##   9.397178   9.948101   1.211564  25.262642  39.977455   8.855912  34.593394
##     income
##   2.760270
```

VIFs are not as bad as I though they might be. Rooms, bedrooms, and households are huge so will need to be dealt with, but longitude, latitude, and population are acceptable (although 5< VIF <10 is indicative of possible multicollinearity issues, it's not always the worst thing) and age and income look great.

Part d (3 points)

Fit a principal component model using the `pcr()` function in the `pls` package. Since the variables are on very different scales, use the `scale = TRUE` option in the function. Then find the VIF values for this model. The `vif()` function from the `car` library won't work here. Instead, take the diagonals of the inverse of the correlation matrix (like we did in the regularization section of Notes 7) for the `scores` output.

```
housePCR <- pcr(value ~ ., data = hTrain, scale = T)


X <- as.matrix(housePCR[["scores"]])

corMat <- cor(X) |>
  solve() |>
  diag() |>
  round(3)
corMat
```

```
## Comp 1 Comp 2 Comp 3 Comp 4 Comp 5 Comp 6 Comp 7 Comp 8
##      1      1      1      1      1      1      1      1
```

Part e (5 points)

Now take the summary of your principal component model. With PCA, the common amount of variation we want to explain in the predictors is 90% or more. How many components are needed to achieve this? For that number of components, how much of the variation in log of home values is explained? How many

components are needed to explain a "good" amount of the variation in the log of home values? Note: the upper bound on the amount of variation in the log of home values here with all 8 components will be equal to the R^2 value of the OLS model.

```
summary(housePCR)
```

```
## Data:    X dimension: 16513 8
##  Y dimension: 16513 1
## Fit method: svdpc
## Number of components considered: 8
## TRAINING: % variance explained
##        1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps  8 comps
## X       48.875   72.768    86.46    97.11    98.63    99.48    99.82   100.00
## value    1.575    2.134    26.09    49.33    49.45    66.94    66.95    67.13
```

```
round(housePCR$projection, 3)
```

```
##            Comp 1 Comp 2 Comp 3 Comp 4 Comp 5 Comp 6 Comp 7 Comp 8
## longitude   0.067 -0.702  0.037 -0.081  0.220  0.591  0.311  0.020
## latitude   -0.073  0.702 -0.008 -0.092  0.148  0.600  0.333  0.041
## age        -0.174  0.009  0.515  0.832  0.068  0.082  0.006  0.012
## rooms       0.488  0.082 -0.085  0.105  0.424  0.236 -0.607 -0.364
## bedrooms    0.492  0.060  0.129  0.002  0.364 -0.213  0.116  0.739
## population  0.478  0.010  0.140  0.045 -0.779  0.314 -0.140  0.153
## households  0.494  0.054  0.124  0.057 -0.001 -0.290  0.604 -0.534
## income      0.078 -0.022 -0.821  0.525 -0.068  0.024  0.163  0.107
```

To explain more than 90% of the variation in the predictors, we need 4 components. For a "good" amount of variation in log of home values, I think 6. 49.45% with 5 components seems good, but a ~16% increase with one additional component doesn't seem too shabby.

Part f (3 points)

Now fit a partial least squares model using the `plsr()` function in the `pls` package. Then find the VIF values for this model like you did in part d.

```
housePLS <- plsr(value~ ., data = hTrain, scale = T)

X <- as.matrix(housePLS[["scores"]])


corMat <- cor(X) |>
  solve() |>
  diag() |>
  round(3)
corMat
```

```
## Comp 1 Comp 2 Comp 3 Comp 4 Comp 5 Comp 6 Comp 7 Comp 8
##      1      1      1      1      1      1      1      1
```

Part g (4 points)

Now take the summary of your partial least squares model. Compare this output to the summary of the PCA model. Explain what the differences are.

How many components are needed to balance a good amount of variation explained in X and in y for the PLS model?

```
summary(housePLS)
```

```
## Data:    X dimension: 16513 8
##  Y dimension: 16513 1
## Fit method: kernelpls
## Number of components considered: 8
## TRAINING: % variance explained
##         1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps  8 comps
## X         25.05    61.19    78.27    91.25    97.97    99.42    99.67   100.00
## value     41.32    51.75    54.87    60.57    66.99    67.05    67.13    67.13
```

```
round(housePLS$projection, 3)
```

```
##            Comp 1 Comp 2 Comp 3 Comp 4 Comp 5 Comp 6 Comp 7 Comp 8
## longitude  -0.037 -0.288 -0.755 -0.646 -1.239 -0.219  0.122 -0.292
## latitude   -0.258 -0.196 -0.252 -1.110 -1.053 -0.146  0.144 -0.305
## age         0.074  0.361  0.438  0.166 -0.612 -0.060  0.022 -0.003
## rooms       0.243 -0.263 -0.103 -0.367 -0.474 -0.645 -0.485  0.455
## bedrooms    0.112 -0.329  0.314  0.448  0.568  0.115  0.863  0.137
## population  0.031 -0.501 -0.241 -0.547 -0.828  0.679  0.083  0.187
## households  0.148 -0.292  0.321  0.432  0.505 -0.169 -0.470 -0.755
## income      0.912  0.869  0.007 -0.129 -0.020  0.128  0.146 -0.119
```

The amount of variation in log of home value explained by the PLS model starts stronger and grows a bit faster than the PCA model. In contrast, the amount of variation explained by X increases slower in the PLS model than in the PCR model. For example, with only 4 components, the PLS model is able to explain 60.57% of the variation in log of home value compared to 49.33% for the PCA model. However, with 4 components, the PLS model only explains 91.25% of variation in X compared to 97.11% for the PCA model.

For the PLS model, I would probably use 6 or 4 components. I can see arguments for both.

Part h (5 points)

In part e, you said how many components were needed for the PCA. Using that number of components, use the PCA and PLS models to find the root MSE of the model when predicting the testing set. Note: the root MSE (or residual standard error, also abbreviated RMSE) is found by taking the square root of the sum of squared residuals divided by the residual degrees of freedom, which is $n-p$. Also, be aware that when you use the `predict()` function on your model, it will give you predictions for each number of components. You can access the predictions for using 3 components, for example, by typing `predict(pca_model, test_set)[,3]`.

Then in part g, you said how many components were needed for the PLS Using that number of components, use the PCA and PLS models to find the root MSE of the model when predicting the testing set.

Compare the results of the predictions in both cases.

```
predPCR6 <- predict(housePCR, hTest)[,,6]
predPLS6 <- predict(housePLS, hTest)[,,6]

predPCR4 <- predict(housePCR, hTest)[,,4]
predPLS4 <- predict(housePLS, hTest)[,,4]

residDF <- nrow(hTest) - ncol(housing) - 1

rmsePCR6 <- sqrt(sum((hTest$value - predPCR6)^2)/residDF)
rmsePLS6 <- sqrt(sum((hTest$value - predPLS6)^2)/residDF)

rmsePCR4 <- sqrt(sum((hTest$value - predPCR4)^2)/residDF)
rmsePLS4 <- sqrt(sum((hTest$value - predPLS4)^2)/residDF)

print(rmsePCR6)
```

```
## [1] 0.3281327
```

```
print(rmsePLS6)
```

```
## [1] 0.3265532
```

```
print(rmsePCR4)
```

```
## [1] 0.4011519
```

```
print(rmsePLS4)
```

```
## [1] 0.3534235
```

The smallest RMSE is for the PLS model with 6 components. In fact, both models with 6 component had smaller RMSEs than the ones with 4 components, which makes sense since 6 components should capture more of the variation in log house value. I'm surprised 6 components wouldn't create overfitting issues.

Part i (10 points)

Using the number of components you said were needed for the PLS model in part g, come up with reasonable surrogates for each of those components. Look at the projection output for this. Make sure to center and scale each variable. You can do this with the `scale()` function.

Then fit a OLS model using those surrogates in the training set. Check the VIF of this model to make sure each one is below 5. If they are not, your surrogates should be changed.

```
housePLS$projection
```

```
##                Comp 1     Comp 2       Comp 3     Comp 4      Comp 5
## longitude  -0.03668363 -0.2882141 -0.755105669 -0.6457674 -1.23923003
## latitude   -0.25828947 -0.1961195 -0.252300063 -1.1097696 -1.05266443
## age         0.07350206  0.3611550  0.438372748  0.1660938 -0.61183757
```

```
## rooms       0.24292285 -0.2626650 -0.102899121 -0.3666026 -0.47430573
## bedrooms    0.11197964 -0.3288953  0.313706279  0.4481235  0.56783386
## population  0.03072851 -0.5010793 -0.240688876 -0.5474728 -0.82767813
## households  0.14752159 -0.2918704  0.320934976  0.4318687  0.50542726
## income      0.91229409  0.8688939  0.006668975 -0.1287986 -0.02003206
##                 Comp 6      Comp 7      Comp 8
## longitude  -0.21916345  0.12205532 -0.29173338
## latitude   -0.14640636  0.14403705 -0.30519483
## age        -0.05978321  0.02215771 -0.00263781
## rooms      -0.64453666 -0.48523330  0.45521533
## bedrooms    0.11508518  0.86273417  0.13688309
## population  0.67902158  0.08296263  0.18654141
## households -0.16851609 -0.47041763 -0.75481883
## income      0.12766711  0.14625431 -0.11938833
```

```
x <- hTrain[,1:8] |>
  scale() |>
  as.matrix()

scaleTrain <- scale(hTrain[,1:8]) |>
  as.data.frame() |>
  mutate(value = hTrain$value)

scaleTest <- scale(hTest[,1:8]) |>
  as.data.frame() |>
  mutate(value = hTest$value)
```

```
sur1 <- (x[,8])
#only one > .30
sur2 <- ((x[,3] + x[,8])/2 - (x[,5] - x[,6]))
#averages of two large positives minus difference between two large negatives
sur3 <- ((x[,3] + x[,5] + x[,7])/3 - x[,1])
#difference between average of positive values > 0.3 and the largest loading
sur4 <- ((x[,1] + x[,2] + x[,6])/3 - (x[,5] + x[,7])/2)
#difference between 3 largest negatives and two largest positives
sur5 <- ((x[,1] + x[,2] + x[,6]))
#average of 3 largest negatives. don't know if it's necessary to include more in here.
sur6 <- (x[,4] - x[,6])
```

```
finalMod <- lm(value ~ sur1 + sur2 + sur3 + sur4 + sur5 + sur6,
               data = scaleTrain)
summary(finalMod)
```

```
##
## Call:
## lm(formula = value ~ sur1 + sur2 + sur3 + sur4 + sur5 + sur6,
##     data = scaleTrain)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.68113 -0.19380 -0.00349  0.18859  2.93430
##
## Coefficients:
```

```
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 12.084578   0.002550 4738.49   <2e-16 ***
## sur1         0.318689   0.004639   68.70   <2e-16 ***
## sur2         0.084508   0.005470   15.45   <2e-16 ***
## sur3        -0.048390   0.002703  -17.90   <2e-16 ***
## sur4        -0.610889   0.006592  -92.68   <2e-16 ***
## sur5        -0.375257   0.003944  -95.15   <2e-16 ***
## sur6        -0.158822   0.008118  -19.56   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.3277 on 16506 degrees of freedom
## Multiple R-squared:  0.6694, Adjusted R-squared:  0.6693
## F-statistic:  5571 on 6 and 16506 DF,  p-value: < 2.2e-16
```

```
vif(finalMod)
```

```
##     sur1     sur2     sur3     sur4     sur5     sur6
## 3.308194 3.098410 1.608756 3.401735 2.624970 2.587625
```

Part j (6 points)

Now using that model you fit in the previous part with the surrogates, find the root MSE for predicting the testing set using that model, and compare it to what you got in part h. Describe some pros and cons of the surrogate model.

Hint: to predict using this model, you will have to create a `newdata` data frame (or tibble) and then redefine your surrogates in that data frame using the scaled testing data set. Make sure the variable names in the `newdata` data frame match the variables names used in the model you definied in part i.

```
x <- scale(hTest[,1:8]) |>
  as.data.frame() |>
  mutate(value = hTest$value)

sur1 <- (x[,8])
#only one > .30
sur2 <- ((x[,3] + x[,8])/2 - (x[,5] - x[,6]))
#averages of two large positives minus difference between two large negatives
sur3 <- ((x[,3] + x[,5] + x[,7])/3 - x[,1])
#difference between average of positive values > 0.3 and the largest loading
sur4 <- ((x[,1] + x[,2] + x[,6])/3 - (x[,5] + x[,7])/2)
#difference between 3 largest negatives and two largest positives
sur5 <- ((x[,1] + x[,2] + x[,6]))
#average of 3 largest negatives. don't know if it's necessary to include more in here.
sur6 <- (x[,4] - x[,6])

scaledPred <- predict(finalMod, scaleTest)
```

```
residDF <- nrow(scaleTest) - 7
rmseScaled <- sqrt(sum((scaleTest$value - scaledPred)^2)/residDF)
rmseScaled
```

```
## [1] 0.3285293
```

Pros: the surrogate model has low VIFs. It also has a low-ish RMSE - at least, barely higher than the PCR and PLS with 6 components. So, a decent amount of predictive power without issues of multicollinearity.

Cons: Very hard to interpret coefficients. Also, if we're going to have hard to interpret coefficients regardless, it might be worth it to just go with the models with the perfect VIFs and lower RMSE?