MATH 3190 Homework 4

Focus: Notes 7 Part 1

Due March 9, 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 (55 points)

Concrete is the most important material in civil engineering. The concrete compressive strength is an important attribute of the concrete and our goal is to predict the concrete compressive strength (in MPa) from the following variables:

- Cement (in kg/m^3)
- Blast furnace slag (in kg/m^3)
- Fly ash (in kg/m^3)
- Water (in kg/m^3)
- Superplasticizer (in kg/m^3)
- Coarse aggregate (in kg/m^3)
- Fine aggregate (in kg/m^3)
- Age of concrete (in days)

This dataset came from the UCI ML Repository and can be found on the Math3190_Sp24 GitHub page along with a Readme file.

Part a (3 points)

Read in the dataset. The data file is a .xls file, so you'll need to either convert it to a .csv or, preferably, use the `readxl` package to read in the Excel file. Once it is read in, change the names of the variables so they are shorter yet still descriptive.

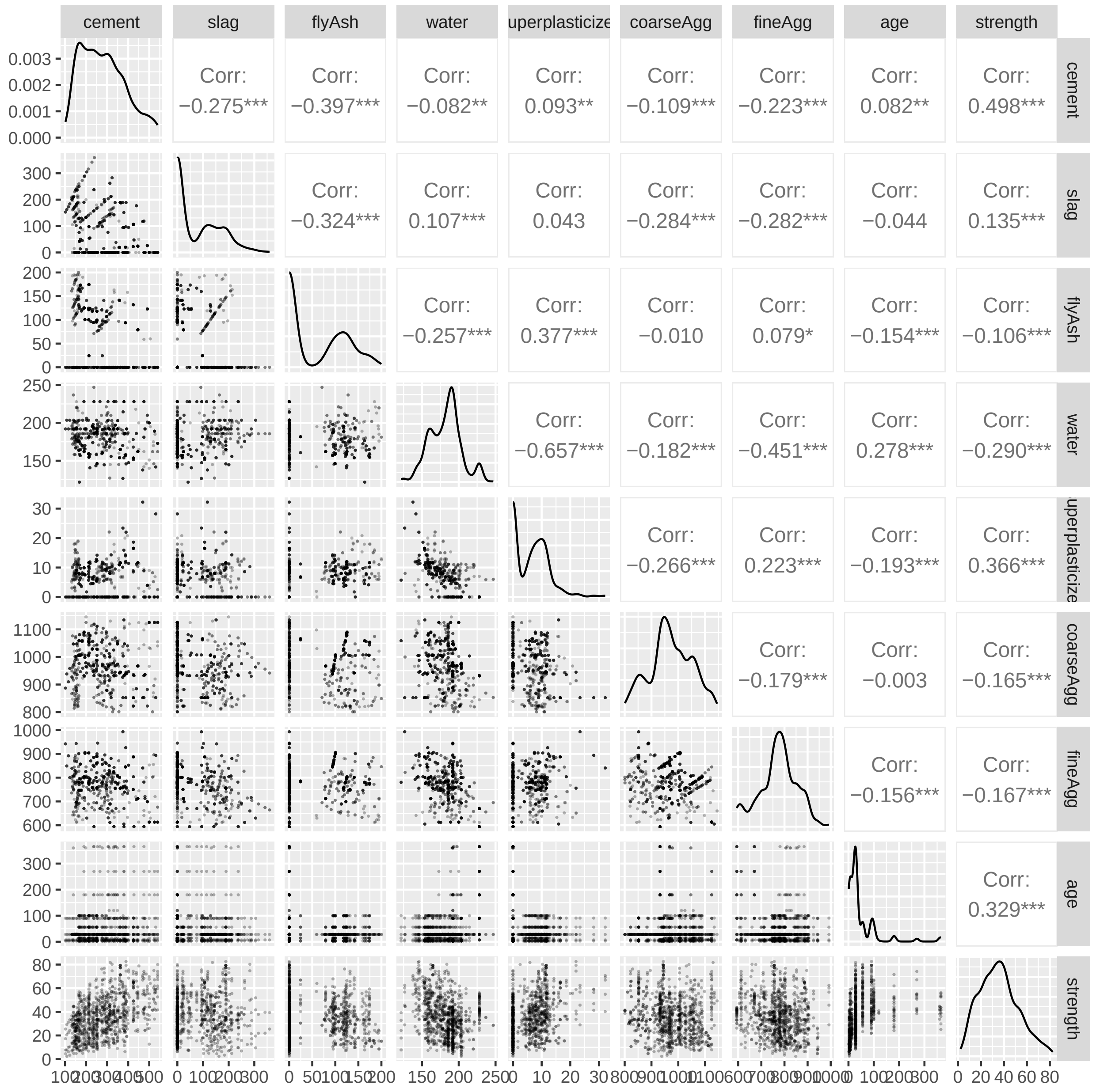
```r
conc <- readxl::read_xls("Concrete_Data.xls") |>
  rename("cement" = "Cement (component 1)(kg in a m^3 mixture)",
         "slag" = "Blast Furnace Slag (component 2)(kg in a m^3 mixture)",
         "flyAsh" = "Fly Ash (component 3)(kg in a m^3 mixture)",
         "water" = "Water  (component 4)(kg in a m^3 mixture)",
         "superplasticizer" = "Superplasticizer (component 5)(kg in a m^3 mixture)",
         "coarseAgg" = "Coarse Aggregate  (component 6)(kg in a m^3 mixture)",
         "fineAgg" = "Fine Aggregate (component 7)(kg in a m^3 mixture)",
         "age" = "Age (day)",
         "strength" = "Concrete compressive strength(MPa, megapascals)"
         )
attach(conc)
```

Part b (5 points)

In the `GGally` library is the function `ggpairs`, which makes a nice scatterplot matrix in the `ggplot2` style. Create this scatterplot matrix for all of the variables in the dataset (they should all be plotted together in one plot).

Comment on the scatterplot matrix. Which variables seem to have a (at least somewhat) linear relationship with compressive strength? Does it seem like multicollinearity will be an issue here? Does it seem like the transformation of at least one variable is appropriate? There should be one (fairly) obvious variable that needs to be transformed.

```r
ggpairs(conc, lower = list(continuous = wrap("points", alpha = 0.3,
                                              size = 0.1),
                                              combo = wrap("dot", alpha = 0.4, size = 0.2)))
```



Cement amount and strength seem to have some semblance of a linear relationship. If I squint super hard, superplasticizer and strength also look somewhat promising. Water and superplasticizer are more correlated with each other than they are with strength, so there could be some issues with multicollinearity. Slag and flyash also look problematic. Age should definitely be logged.

Part c (8 points)

Fit a linear model (with the `lm()` function) predicting compressive strength using all other variables and include any transformations you thought were appropriate in part b.

Using `ggplot()`, make a QQ plot of the raw residuals. Include the QQ line as well. Comment on whether the residuals appear to be approximately normal.

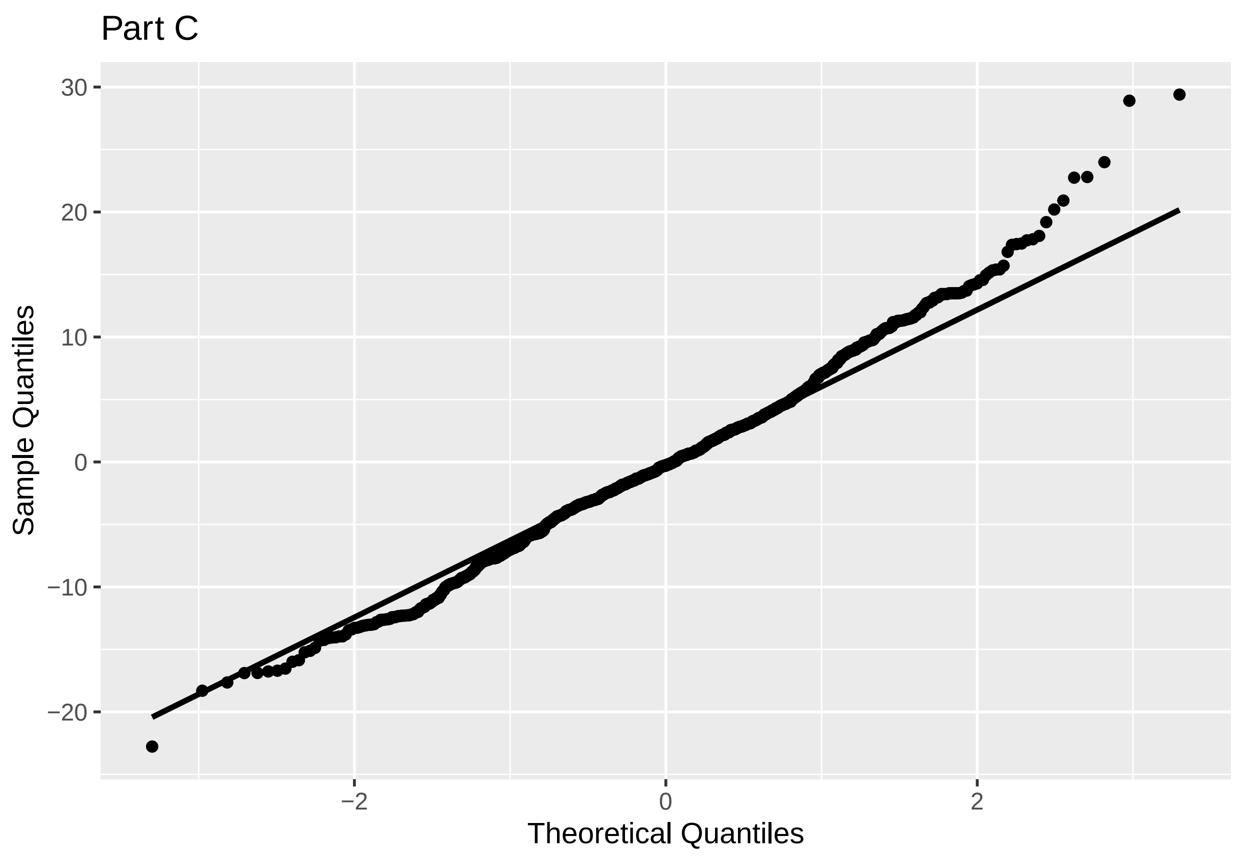
Using `ggplot()`, make a plot of the jackknife residuals (obtained using the `rstudent()` function) on the y-axis and the fitted values of the model on the x-axis. Comment on whether this residual plot looks good. If it does not, indicate what the problem(s) is (are).

```r
conc$age <- log(conc$age)
```

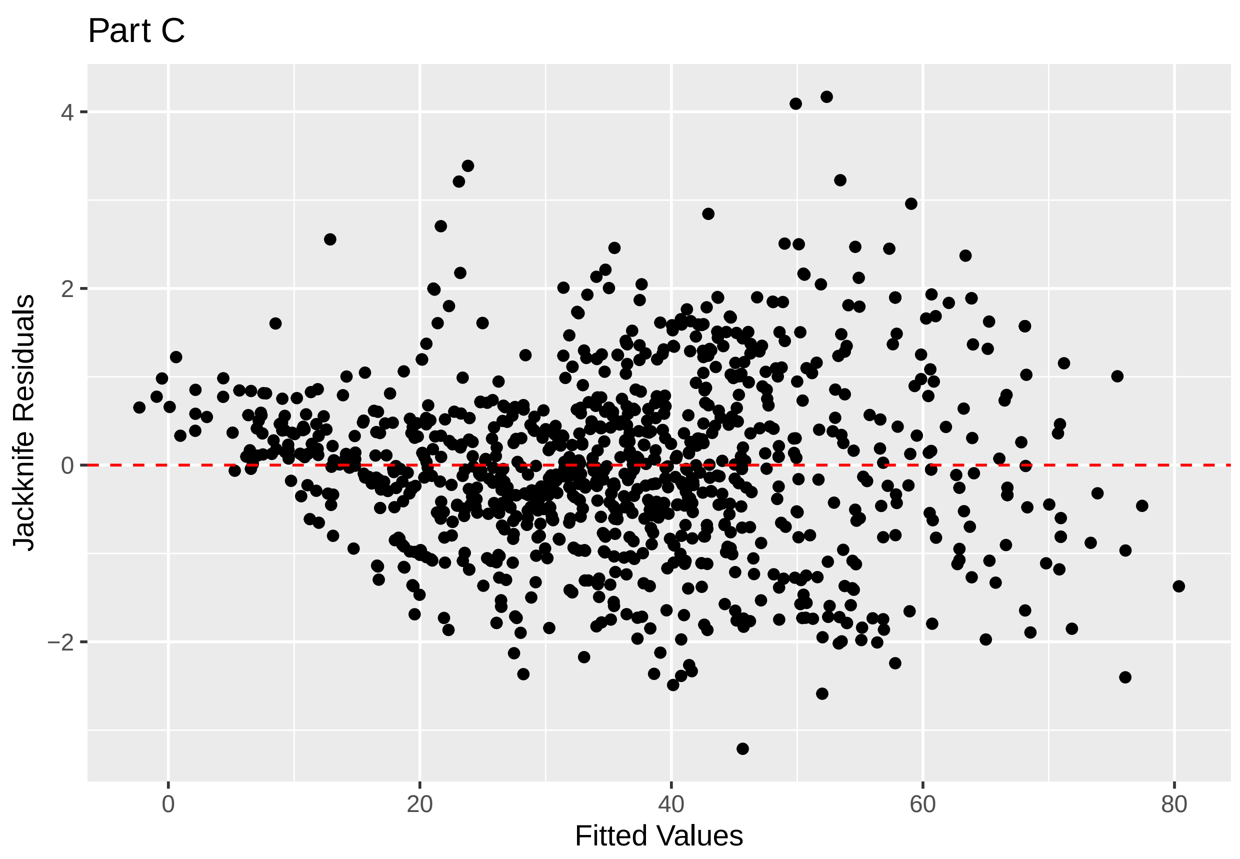
```r
mod1 <- lm(strength ~ ., data = conc)
summary(mod1)
```

```
##
## Call:
## lm(formula = strength ~ ., data = conc)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -22.7746  -4.2769  -0.2939   4.0230  29.3969
##
## Coefficients:
##                    Estimate Std. Error t value Pr(>|t|)
## (Intercept)      -77.370586  18.332821  -4.220 2.66e-05 ***
## cement             0.134840   0.005850  23.051  < 2e-16 ***
## slag               0.115926   0.006965  16.644  < 2e-16 ***
## flyAsh             0.094397   0.008626  10.943  < 2e-16 ***
## water             -0.129181   0.027566  -4.686 3.16e-06 ***
## superplasticizer   0.117387   0.064390   1.823   0.0686 .
## coarseAgg          0.029625   0.006463   4.583 5.14e-06 ***
## fineAgg            0.035952   0.007367   4.880 1.23e-06 ***
## age                8.746983   0.191779  45.610  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 7.145 on 1021 degrees of freedom
## Multiple R-squared:  0.8185, Adjusted R-squared:  0.8171
## F-statistic: 575.5 on 8 and 1021 DF,  p-value: < 2.2e-16
```

```r
resids <- data.frame(residuals(mod1))

ggplot(resids, aes(sample = residuals.mod1.)) +
  geom_qq() +
  geom_qq_line(lwd = 1) +
  labs(x = "Theoretical Quantiles",
       y = "Sample Quantiles") +
  ggtitle("Part C")
```



```r
ggplot(conc, aes(x = fitted(mod1), y = rstudent(mod1))) +
  geom_point() +
  geom_hline(yintercept = 0, lty = "dashed", color = "red") +
  labs(x = "Fitted Values",
       y = "Jackknife Residuals") +
  ggtitle("Part C")
```



This looks pretty rough. The residuals don't seem to be approximately normal although I know I tend to be a bit picky about this. The residual plot looks somewhat weird too. I feel like for fitted values above 40 or so it looks okay, but from 0 - 40 there's a definite fan shape.

Part d (6 points)

Let's do some model selection to determine if any variables should be dropped from the model.

First, use the `step()` function on the model you fit in the previous part. This will select the variables using AIC.

Second, use `step()` with the option `k` equal to the log of the sample size. This will select the variables using BIC.

Third, use the `cv.glmnet()` function in the `glmnet` library (set a seed first) to fit a LASSO for variable selection. Note, the `model.matrix()` function will be helpful here to get a matrix to input for the `x` argument in the `cv.glmnet()` function. Use the "lambda.1se" value to select the variables and using that λ value, fit the LASSO model using `glmnet()`.

Finally, compare the variables that were selected by the three methods.

```r
step(mod1)
```

```
## Start:  AIC=4059.84
## strength ~ cement + slag + flyAsh + water + superplasticizer +
##     coarseAgg + fineAgg + age
##
```

```
##                    Df Sum of Sq    RSS    AIC
## <none>                           52127 4059.8
## - superplasticizer  1       170  52296 4061.2
## - coarseAgg         1      1073  53199 4078.8
## - water             1      1121  53248 4079.8
## - fineAgg           1      1216  53343 4081.6
## - flyAsh            1      6114  58241 4172.1
## - slag              1     14143  66269 4305.1
## - cement            1     27129  79256 4489.4
## - age               1    106207 158333 5202.2

##
## Call:
## lm(formula = strength ~ cement + slag + flyAsh + water + superplasticizer +
##     coarseAgg + fineAgg + age, data = conc)
##
## Coefficients:
##      (Intercept)            cement              slag            flyAsh
##        -77.37059           0.13484           0.11593           0.09440
##            water  superplasticizer         coarseAgg           fineAgg
##         -0.12918           0.11739           0.02962           0.03595
##              age
##          8.74698
```

```
step(mod1, k = log(nrow(conc)))
```

```
## Start:  AIC=4104.28
## strength ~ cement + slag + flyAsh + water + superplasticizer +
##     coarseAgg + fineAgg + age
##
##                    Df Sum of Sq    RSS    AIC
## - superplasticizer  1       170  52296 4100.7
## <none>                           52127 4104.3
## - coarseAgg         1      1073  53199 4118.3
## - water             1      1121  53248 4119.3
## - fineAgg           1      1216  53343 4121.1
## - flyAsh            1      6114  58241 4211.6
## - slag              1     14143  66269 4344.6
## - cement            1     27129  79256 4528.9
## - age               1    106207 158333 5241.7
##
## Step:  AIC=4100.69
## strength ~ cement + slag + flyAsh + water + coarseAgg + fineAgg +
##     age
##
##             Df Sum of Sq    RSS    AIC
## <none>                    52296 4100.7
## - coarseAgg  1       906  53202 4111.4
## - fineAgg    1      1107  53404 4115.3
## - water      1      2256  54553 4137.3
## - flyAsh     1      6474  58771 4214.0
## - slag       1     14128  66425 4340.1
## - cement     1     27115  79411 4524.0
## - age        1    107618 159915 5245.0
```

```
##
## Call:
## lm(formula = strength ~ cement + slag + flyAsh + water + coarseAgg +
##     fineAgg + age, data = conc)
##
## Coefficients:
## (Intercept)       cement         slag       flyAsh        water    coarseAgg
##   -66.44178      0.13480      0.11587      0.09637     -0.15569      0.02553
##     fineAgg          age
##     0.03390      8.77553
```

```
set.seed(1)

X <- model.matrix(mod1)[,-1]
y <- conc$strength

lasso <- cv.glmnet(X, y)
coef(lasso)
```

```
## 9 x 1 sparse Matrix of class "dgCMatrix"
##                           s1
## (Intercept)      13.35741085
## cement            0.09951322
## slag              0.07361064
## flyAsh            0.04255259
## water            -0.22313727
## superplasticizer  0.14168885
## coarseAgg         .
## fineAgg           .
## age               8.33112904
```

```
lasso2 <- glmnet(X, y, lambda = lasso$lambda.1se)
lasso2
```

```
##
## Call:  glmnet(x = X, y = y, lambda = lasso$lambda.1se)
##
##   Df  %Dev Lambda
## 1  6 81.03  0.295
```

** The model selected based on AIC included all the variables. The model selected based on BIC included all the variables except for superplasticizer. The model fit using the LASSO selected all the variables except coarse aggregate and fine aggregate.**

Part e (4 points)

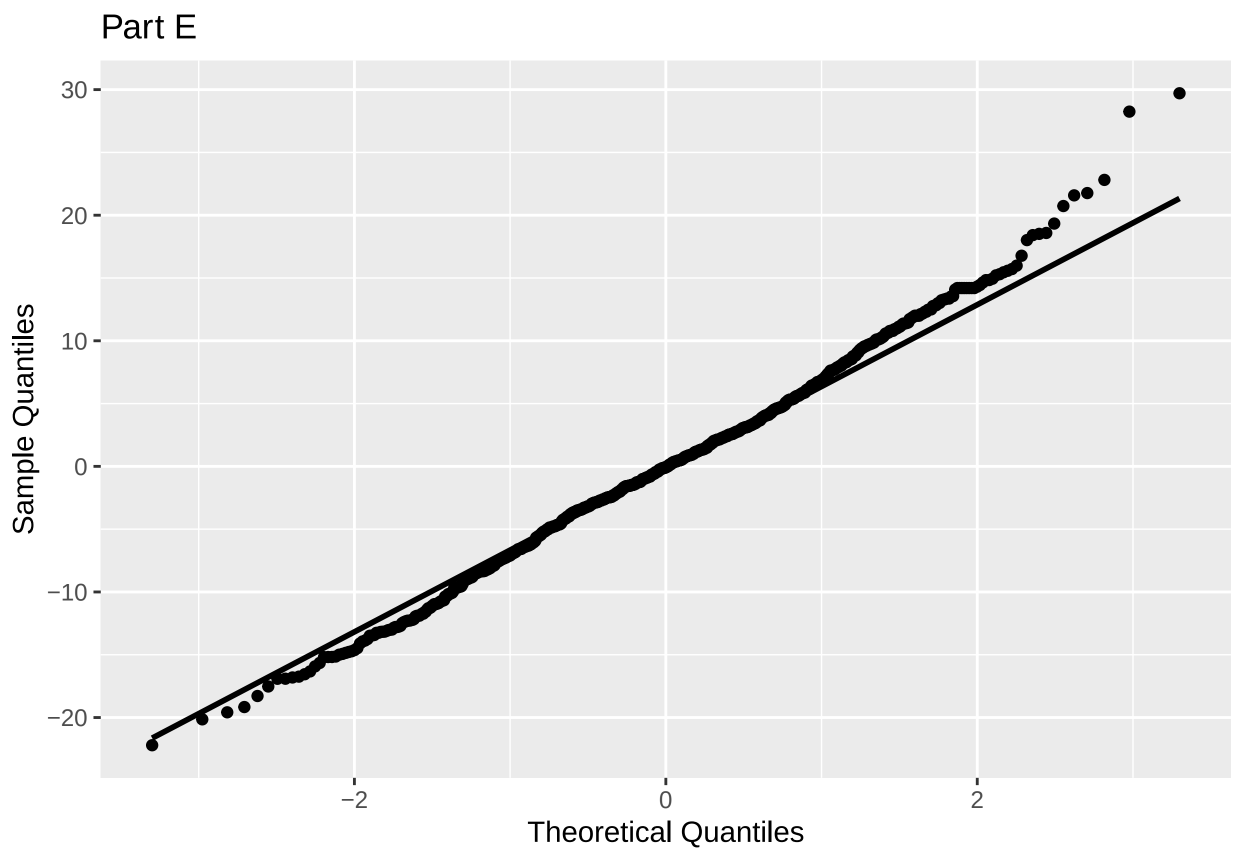
Using the variables selected by the method that reduced the number of variables the most, fit a new ordinary least squares (OLS) model for predicting compressive strength using the `lm()` function.

Using `ggplot()`, make a QQ plot of the raw residuals. Include the QQ line as well. Comment on whether the residuals appear to be approximately normal and whether this plot looks better than the QQ plot in part c.

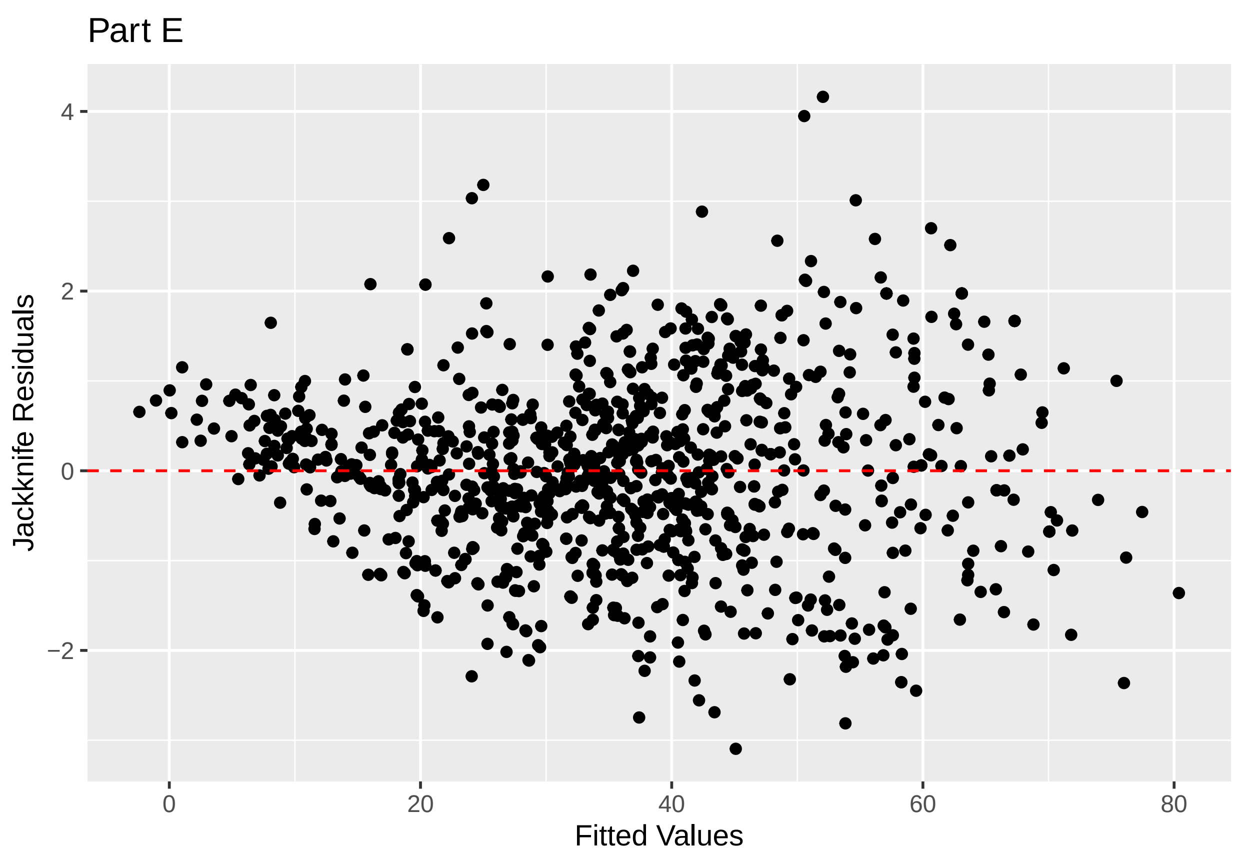
Using `ggplot()`, make a plot of the jackknife residuals (obtained using the `rstudent()` function) on the y-axis and the fitted values of the model on the x-axis. Comment on whether this residual plot looks good and whether this plot looks better than the residual plot in part c. If it does not, indicate what the problem(s) is (are).

```r
newMod <- lm(strength ~ cement + slag + flyAsh + water +
               superplasticizer + age, data = conc)
```

```r
resids <- data.frame(residuals(newMod))

ggplot(resids, aes(sample = residuals.newMod.)) +
  geom_qq() +
  geom_qq_line(lwd = 1) +
  labs(x = "Theoretical Quantiles",
       y = "Sample Quantiles") +
  ggtitle("Part E")
```



```r
ggplot(conc, aes(x = fitted(newMod), y = rstudent(newMod))) +
  geom_point() +
  geom_hline(yintercept = 0, lty = "dashed", color = "red")+
  labs(x = "Fitted Values",
       y = "Jackknife Residuals") +
  ggtitle("Part E")
```



I still wouldn't say that the residuals look normal, but I do think they look marginally better than they did in part C. The residual plot still has that slight fan shape going on, especially with smaller fitted values.

Part f (10 points)

Since the residual plot still does not look good, let's try to use weighted least squares. Following slides 12-14 of Notes 7, create a vector of weights and then fit a model using weighted least squares.

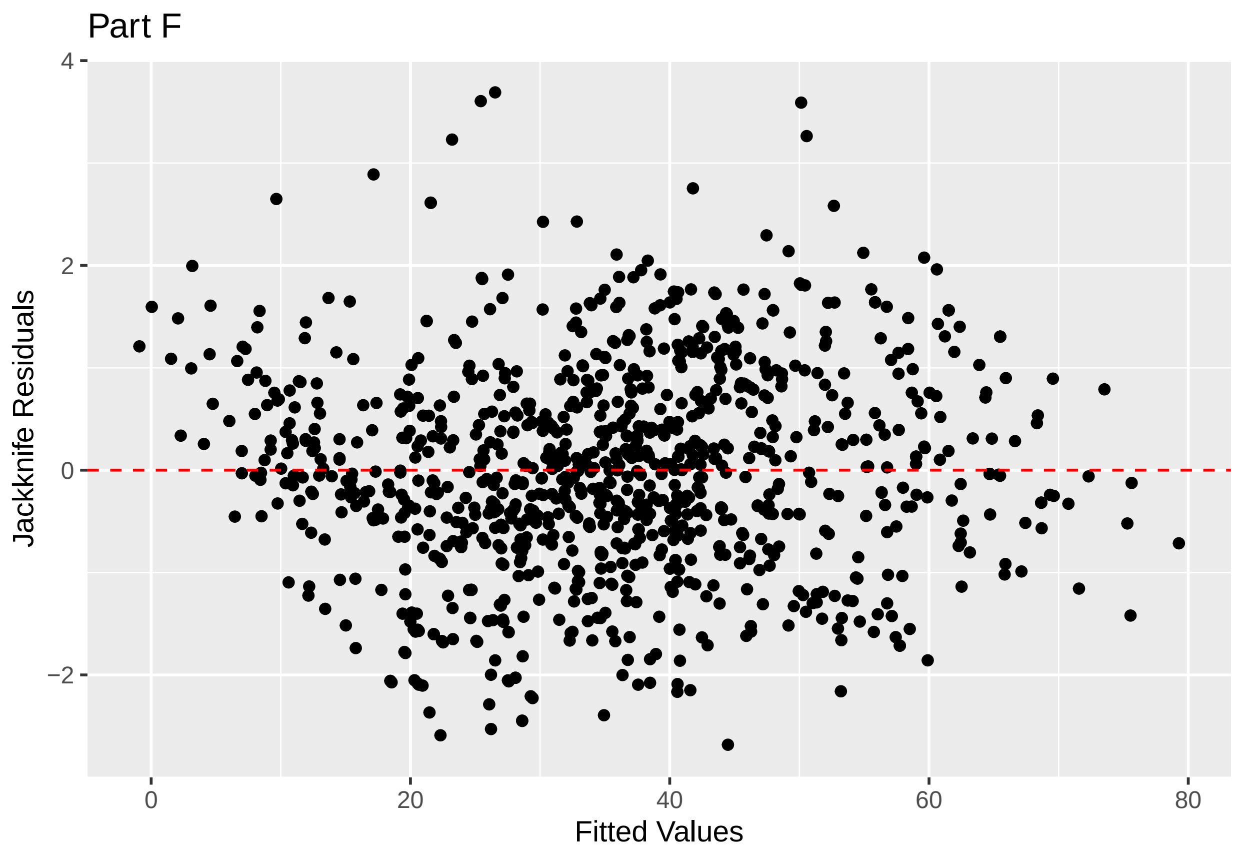
Using `ggplot()`, make a plot of the jackknife residuals (obtained using the `rstudent()` function) on the y-axis and the fitted values of the weighted model on the x-axis. Comment on whether this residual plot looks good and whether this plot looks better than the residual plot in part e.

```
newMod2 <- lm(strength ~ cement + slag + flyAsh + water +
                superplasticizer + age, data = conc)

tempDf <- data.frame(
  residuals <- abs(newMod2$residuals),
  fit <- newMod2$fitted.values
)

eMod <- lm(residuals ~ fit, data = tempDf)

w <- 1/eMod$fitted.values^2

wlsMod <- lm(strength ~ cement + slag + flyAsh + water +
```

```r
                superplasticizer + age, data = conc, weights = w)
summary(wlsMod)
```

```
##
## Call:
## lm(formula = strength ~ cement + slag + flyAsh + water + superplasticizer +
##     age, data = conc, weights = w)
##
## Weighted Residuals:
##     Min      1Q  Median      3Q     Max
## -3.3670 -0.7834 -0.0461  0.8287  4.6120
##
## Coefficients:
##                   Estimate Std. Error t value Pr(>|t|)
## (Intercept)       7.664843   2.650101   2.892  0.00391 **
## cement            0.106612   0.002792  38.179  < 2e-16 ***
## slag              0.077462   0.003142  24.650  < 2e-16 ***
## flyAsh            0.053431   0.004981  10.727  < 2e-16 ***
## water            -0.204880   0.013655 -15.004  < 2e-16 ***
## superplasticizer  0.118963   0.060696   1.960  0.05027 .
## age               8.187942   0.165391  49.507  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.263 on 1023 degrees of freedom
## Multiple R-squared:  0.8387, Adjusted R-squared:  0.8378
## F-statistic: 886.6 on 6 and 1023 DF,  p-value: < 2.2e-16
```

```r
ggplot(conc, aes(x = fitted(wlsMod), y = rstudent(wlsMod))) +
  geom_point() +
  geom_hline(yintercept = 0, lty = "dashed", color = "red") +
  labs(x = "Fitted Values",
       y = "Jackknife Residuals") +
  ggtitle("Part F")
```



This residual plot looks a lot better than the one from part E. Still a bit funky maybe, but I don't see an obvious pattern.

Part g (9 points)

Using the unweighted model from part e and the weighted model from part f, find and report both a confidence interval for the mean compressive strength and a prediction interval for the specific compressive strength for concrete that has a cement value of 300, a blast furnace slag of 90, a fly ash of 50, a water value of 200, a superplasticizer of 2.5, a coarse aggregate of 900, a fine aggregate of 600, and an age of 300 days. Note: some of those variables will not be used since you reduced the number of variables earlier. You can use the `predict()` function here and remember that you will need to find the specific weight for the given predictor variable values for the weighted intervals.

Comment on how these intervals differ. Does the change make sense given the value of the fitted value?

```
pred <- data.frame(cement = 300, slag = 90, flyAsh = 50, water = 200,
                   superplasticizer = 2.5, age = log(300))
```

Confidence and Prediction intervals for unweighted model

```
uwConf <- predict(newMod, pred, interval = "confidence")
uwPred <- predict(newMod, pred, interval = "prediction")

print(uwConf)
```

```
##        fit      lwr      upr
## 1 56.29408 55.17605 57.41212
```

```
print(uwPred)
```

```
##        fit      lwr      upr
## 1 56.29408 42.07413 70.51404
```

Confidence and Prediction Intervals for weighted model

```
wConf <- predict(wlsMod, pred, interval = "confidence")

newFit <- predict(wlsMod, pred)
eModInt <- eMod[["coefficients"]][["(Intercept)"]]
eModCoef <- eMod[["coefficients"]][["fit"]]

newWeight <- 1/(eModInt + eModCoef*newFit)^2

wPred <- predict(wlsMod, pred, interval = "prediction", weight
                 = newWeight)

print(wConf)
```

```
##        fit      lwr      upr
## 1 55.31516 54.19356 56.43677
```

```
print(wPred)
```

```
##        fit      lwr      upr
## 1 55.31516 37.12486 73.50547
```

The weighted prediction interval is quite a bit wider than the OLS prediction interval, which seems to track with what we saw in class. The confidence intervals for both also seem to be similar widths, which makes sense to me.

Part h (10 points)

Write a function called `predict_weighted` that takes three inputs: the **unweighted** model, the data frame containing the information about the value(s) of the predictor variables we are using to predict, and the interval type (either "confidence" or "prediction"). This function should return the predicted value and the interval bounds for the specified interval type for weighted least squares. So, this function should compute the weights, obtain the weighted least squares model, find the specific weight for the new value, and then get the prediction and the interval. This function should work for any model you give it, not just for this exact situation of this problem. So, you should not reference any data sets or variables specific to this concrete problem in the function.

Test this new function for the confidence and prediction intervals you made in part g.

```r
predict_weighted <- function(model, predictors, type){
  tempDf <- data.frame(
  residuals <- abs(model$residuals),
  fit <- model$fitted.values
)

eMod <- lm(residuals ~ fit, data = tempDf)

w <- 1/eMod$fitted.values^2

wlsMod <- lm(formula(model), data = model.frame(model), weights = w)

newFit <- predict(model, predictors)
eModInt <- eMod[["coefficients"]][["(Intercept)"]]
eModCoef <- eMod[["coefficients"]][["fit"]]

newWeight <- 1/(eModInt + eModCoef*newFit)^2

if (type ==  "confidence"){
  return(predict(wlsMod, predictors, interval = "confidence"))
} else if (type == "prediction"){
  return(predict(wlsMod, predictors, interval = "prediction",
                 weights = newWeight))
}

}
```

```r
print(predict_weighted(newMod, pred, "confidence"))
```

```
##        fit      lwr      upr
## 1 55.31516 54.19356 56.43677
```

```r
print(predict_weighted(newMod, pred, "prediction"))
```

```
##        fit      lwr      upr
## 1 55.31516 36.90817 73.72215
```

Problem 2 (29 points)

An automobile consulting company wants to understand the factors on which the pricing of cars depends. The dataset `car_price_prediction.csv` in the GitHub data folder has information on the sales of 4340 vehicles.

Part a (3 points)

Read in the data file and take the log of all numeric variables. Then fit a linear model for predicting the log of selling price using all other variables except “name”.

```r
car <- read_csv("car_price_prediction.csv")
```

```
## Rows: 4340 Columns: 8
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (5): name, fuel, seller_type, transmission, owner
## dbl (3): year, selling_price, km_driven
##
## 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.
```

```r
noNameCar <- car |>
  select(!c("name"))

noNameCar$selling_price <- log(car$selling_price)
noNameCar$km_driven <- log(car$km_driven)
noNameCar$year <- log(car$year)
```

```r
carMod <- lm(selling_price ~ ., data = noNameCar)
summary(carMod)
```

```
##
## Call:
## lm(formula = selling_price ~ ., data = noNameCar)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.81328 -0.30394  0.00052  0.28819  2.32756
##
## Coefficients:
##                               Estimate Std. Error t value Pr(>|t|)
## (Intercept)                 -1.686e+03  3.345e+01 -50.406  < 2e-16 ***
## year                         2.234e+02  4.389e+00  50.896  < 2e-16 ***
## km_driven                   -3.601e-02  1.010e-02  -3.566 0.000366 ***
## fuelDiesel                   5.842e-01  7.526e-02   7.763 1.03e-14 ***
## fuelElectric                 1.943e-01  4.775e-01   0.407 0.684097
## fuelLPG                     -5.132e-02  1.233e-01  -0.416 0.677337
## fuelPetrol                   6.553e-02  7.539e-02   0.869 0.384754
## seller_typeIndividual       -1.555e-01  1.817e-02  -8.555  < 2e-16 ***
## seller_typeTrustmark Dealer  3.089e-01  4.910e-02   6.292 3.45e-10 ***
## transmissionManual          -8.001e-01  2.432e-02 -32.897  < 2e-16 ***
## ownerFourth & Above Owner   -1.418e-01  5.502e-02  -2.577 0.010003 *
## ownerSecond Owner           -4.521e-02  1.840e-02  -2.458 0.014024 *
## ownerTest Drive Car          7.418e-02  1.192e-01   0.622 0.533702
## ownerThird Owner            -1.156e-01  3.058e-02  -3.780 0.000159 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.4705 on 4326 degrees of freedom
## Multiple R-squared:  0.6866, Adjusted R-squared:  0.6857
## F-statistic: 729.1 on 13 and 4326 DF,  p-value: < 2.2e-16
```

Part b (4 points)

Now fit some LASSO models for predicting log price using all but the “name” variable. Try the following values for the regularization parameter: $\lambda = 0, 0.01, 0.1$, and 1 and comment on how the coefficients of the model change.

Note: when $\lambda = 0$, the LASSO coefficients should equal the OLS model coefficients. However, they will actually be a bit off here. That is because the `glmnet()` function has a `thresh` argument that sets a threshold for convergence. It is, by default, set to `1e-07`. To make the parameters match, you can change that to `thresh = 1e-14` instead. This is not necessary, though.

```
X <- model.matrix(carMod)[,-1]
y <- noNameCar$selling_price
lam <- c(0, 0.01, 0.1, 1)
```

```
lassoFit <- glmnet(X, y, lambda = lam, alpha = 1, thresh = 1e-14)
lassoFit[["beta"]]
```

```
## 13 x 4 sparse Matrix of class "dgCMatrix"
##                              s0           s1            s2           s3
## year                          . 208.01313377  2.259669e+02 223.40186674
## km_driven                     .   .          -2.835369e-02  -0.03600606
## fuelDiesel                    .   0.32180675  4.968308e-01   0.58420358
## fuelElectric                  .   .           .              0.19428669
## fuelLPG                       .   .          -9.679051e-04  -0.05132901
## fuelPetrol                    .   .           .              0.06552814
## seller_typeIndividual         .  -0.04696683 -1.549690e-01  -0.15549188
## seller_typeTrustmark Dealer   .   .           2.561571e-01   0.30890345
## transmissionManual            .  -0.56617862 -7.769334e-01  -0.80007713
## ownerFourth & Above Owner     .   .          -6.286074e-02  -0.14178073
## ownerSecond Owner             .   .          -1.927708e-02  -0.04520994
## ownerTest Drive Car           .   .           .              0.07417652
## ownerThird Owner              .   .          -6.959216e-02  -0.11560008
```

The only variables included for all 3 λs (not including 1 because that got rid of them all) are year, fuelDiesel, individual seller, and manual transmission. $\lambda = 0.01$ only removes fuelElectric, fuelPetrol, and owner Test Drive car. The coefficient values did decrease as λ increased, as expected.

Part c (4 points)

Now fit some ridge regression models for predicting log price using all but name. Try the following values for the regularization parameter: $\lambda = 0, 0.01, 0.1$, and 1 and comment on how the coefficients of the model change.

```
ridgeFit <- glmnet(X, y, lambda = lam, alpha = 0, thresh = 1e-14)
ridgeFit[["beta"]]
```

```
## 13 x 4 sparse Matrix of class "dgCMatrix"
##                                        s0           s1           s2           s3
## year                         101.51834774 193.02605799 219.51641534 223.40187429
## km_driven                     -0.07230191  -0.05914219  -0.03939605  -0.03600604
## fuelDiesel                     0.19711870   0.30294038   0.45600681   0.58420362
## fuelElectric                  -0.06727554  -0.09128584   0.06263870   0.19428674
## fuelLPG                       -0.25507627  -0.32229429  -0.17992831  -0.05132895
## fuelPetrol                    -0.18192230  -0.21278783  -0.06329965   0.06552820
## seller_typeIndividual         -0.13492707  -0.15380520  -0.15489444  -0.15549188
## seller_typeTrustmark Dealer    0.22295319   0.30193505   0.30938351   0.30890345
## transmissionManual            -0.41997426  -0.73263339  -0.79379346  -0.80007713
## ownerFourth & Above Owner     -0.24992755  -0.21624680  -0.15523006  -0.14178072
## ownerSecond Owner             -0.10657414  -0.07937217  -0.05052655  -0.04520994
## ownerTest Drive Car            0.13642446   0.06709446   0.07286190   0.07417655
## ownerThird Owner              -0.17062851  -0.15652287  -0.12156697  -0.11560008
```

None of the variables were removed for any value of λ for the ridge regression. They did continue to decrease as λ increased though, as expected.

Part d (4 points)

Now fit some elastic net regression models for predicting log price using all but name. Try the following values for the regularization parameter: $\lambda = 0, 0.01, 0.1$, and 1 for $\alpha = 1/3$ and then comment on how the coefficients of the model change for each α in parts b, c, & d.

```r
elasFit <- glmnet(X, y, lambda = lam, alpha = 1/3, thresh = 1e-14)
elasFit[["beta"]]
```

```
## 13 x 4 sparse Matrix of class "dgCMatrix"
##                                     s0           s1           s2           s3
## year                         66.60137 207.61975247 221.72245636 223.40187442
## km_driven                     .        -0.02483014  -0.03587151  -0.03600604
## fuelDiesel                    .         0.32064671   0.50039360   0.58420362
## fuelElectric                  .         .            .            0.19428674
## fuelLPG                       .         .           -0.08997721  -0.05132895
## fuelPetrol                    .        -0.11815598  -0.01045258   0.06552820
## seller_typeIndividual         .        -0.14374686  -0.15535518  -0.15549188
## seller_typeTrustmark Dealer   .         0.14396005   0.29097346   0.30890345
## transmissionManual            .        -0.68179330  -0.78789688  -0.80007713
## ownerFourth & Above Owner     .         .           -0.12356871  -0.14178072
## ownerSecond Owner             .         .           -0.03985438  -0.04520994
## ownerTest Drive Car           .         .            0.02832558   0.07417655
## ownerThird Owner              .        -0.01702776  -0.10417919  -0.11560007
```

Interestingly, year never decreases to 0 with $\alpha = 1/3$. Less variables decreased to 0 at each λ compared to the LASSO model, but more than the ridge model.

Part e (4 points)

Use the `cv.glmnet()` to find the "optimal" λ obtained by lambda 1.se value for LASSO, ridge, and elastic net and then fit models using the `glmnet()` function for all three using their respective "best" λ. **Set a seed** before running the `cv.glmnet()` each time.

Compare each model's variables and coefficients.

Finding optimal λ for LASSO

```r
set.seed(1)
opt1 <- cv.glmnet(x = X, y = y, alpha = 1)$lambda.1se

lassOpt <- glmnet(X, y, lambda = opt1, alpha = 1 ,
                  thresh = 1e-14)
lassOpt[["beta"]]
```

```
## 13 x 1 sparse Matrix of class "dgCMatrix"
##                                       s0
## year                        228.23506033
## km_driven                    -0.01286447
## fuelDiesel                    0.46024393
## fuelElectric                  .
## fuelLPG                       .
## fuelPetrol                    .
## seller_typeIndividual        -0.14842226
## seller_typeTrustmark Dealer   0.17962169
## transmissionManual           -0.74286315
## ownerFourth & Above Owner     .
## ownerSecond Owner             .
## ownerTest Drive Car           .
## ownerThird Owner             -0.01471026
```

Finding optimal λ for Ridge

```r
set.seed(1)
opt2 <- cv.glmnet(x = X, y = y, alpha = 0)$lambda.1se

ridgeOpt <- glmnet(X, y, lambda = opt2 , alpha = 0, thresh = 1e-14)
ridgeOpt[["beta"]]
```

```
## 13 x 1 sparse Matrix of class "dgCMatrix"
##                                       s0
## year                        187.70272044
## km_driven                    -0.06233777
## fuelDiesel                    0.29481693
## fuelElectric                 -0.09713117
## fuelLPG                      -0.32640636
## fuelPetrol                   -0.21854699
## seller_typeIndividual        -0.15374782
## seller_typeTrustmark Dealer   0.29955787
## transmissionManual           -0.71840760
## ownerFourth & Above Owner    -0.22569518
## ownerSecond Owner            -0.08416574
## ownerTest Drive Car           0.06802304
## ownerThird Owner             -0.16214917
```

Finding optimal λ for Elastic Net

```r
set.seed(1)
opt3 <- cv.glmnet(x = X, y = y, alpha = 1/3)$lambda.1se
```

```r
elasOpt <- glmnet(X, y, lambda = opt3, alpha = 1/3, thresh = 1e-14)
elasOpt[["beta"]]
```

```
## 13 x 1 sparse Matrix of class "dgCMatrix"
##                                      s0
## year                       215.61618885
## km_driven                   -0.03142027
## fuelDiesel                   0.38579061
## fuelElectric                 .
## fuelLPG                     -0.02384564
## fuelPetrol                  -0.09169299
## seller_typeIndividual       -0.15175018
## seller_typeTrustmark Dealer  0.22145044
## transmissionManual          -0.73803243
## ownerFourth & Above Owner   -0.04568816
## ownerSecond Owner           -0.01852343
## ownerTest Drive Car          .
## ownerThird Owner            -0.06047097
```

Similar to the models we fit in parts b, c, and d, ridge reduced the amount of variables by the least and LASSO the most. Year continues to have the greatest coefficient in every model, although it is quite a bit less in the Ridge model. Elastic net only removes fuelElectric and owner Test Drive Car. LASSO removes fuelElectric, fuelLPG, fuelPetrol, ownerFourth & Above Owner, Second owner, and ownerTestDrive car.

Part f (10 points)

Use bootstrapping with at least 1000 samples to estimate the standard errors of the coefficients of the ridge regression model. For each bootstrap sample, run the `cv.glmnet()` function to find the best λ and fit a model using that optimized λ. Use the "lambda.1se" for this. Then compare these bootstrapped standard errors to the standard errors for the OLS model you fit in part a. Are they larger or smaller? Does this make sense?

```r
set.seed(1)
samps <- 1000
betas <- matrix(rep(0, samps * (ncol(X) + 1)), ncol = ncol(X) + 1)

for (i in 1:samps) {
  index <- sample(1:nrow(X), nrow(X), replace = T)
  mod <- cv.glmnet(X[index, ], y[index], alpha = 0, thresh = 1e-14)
  betas[i, ] <- coef(mod, s = "lambda.1se")[1:(ncol(X) + 1)]
}
```

```r
for (i in 1:(ncol(X) + 1)) {
  print(paste("Standard Error for Coefficient", i, ":", sd(betas[, i])))
}
```

```
## [1] "Standard Error for Coefficient 1 : 44.0853397852412"
## [1] "Standard Error for Coefficient 2 : 5.78896789354893"
## [1] "Standard Error for Coefficient 3 : 0.0095435635181005"
## [1] "Standard Error for Coefficient 4 : 0.0104997799138997"
## [1] "Standard Error for Coefficient 5 : 0.0545098780723518"
## [1] "Standard Error for Coefficient 6 : 0.0513786644785666"
## [1] "Standard Error for Coefficient 7 : 0.00991892145309998"
## [1] "Standard Error for Coefficient 8 : 0.0140059532717391"
## [1] "Standard Error for Coefficient 9 : 0.031103692165781"
## [1] "Standard Error for Coefficient 10 : 0.0306720203328968"
## [1] "Standard Error for Coefficient 11 : 0.0489550116958988"
## [1] "Standard Error for Coefficient 12 : 0.0156490313513333"
## [1] "Standard Error for Coefficient 13 : 0.0680598415517056"
## [1] "Standard Error for Coefficient 14 : 0.0275627689416036"
```

```
summary(carMod)
```

```
##
## Call:
## lm(formula = selling_price ~ ., data = noNameCar)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.81328 -0.30394  0.00052  0.28819  2.32756
##
## Coefficients:
##                               Estimate Std. Error t value Pr(>|t|)
## (Intercept)                 -1.686e+03  3.345e+01 -50.406  < 2e-16 ***
## year                         2.234e+02  4.389e+00  50.896  < 2e-16 ***
## km_driven                   -3.601e-02  1.010e-02  -3.566 0.000366 ***
## fuelDiesel                   5.842e-01  7.526e-02   7.763 1.03e-14 ***
## fuelElectric                 1.943e-01  4.775e-01   0.407 0.684097
## fuelLPG                     -5.132e-02  1.233e-01  -0.416 0.677337
## fuelPetrol                   6.553e-02  7.539e-02   0.869 0.384754
## seller_typeIndividual       -1.555e-01  1.817e-02  -8.555  < 2e-16 ***
## seller_typeTrustmark Dealer  3.089e-01  4.910e-02   6.292 3.45e-10 ***
## transmissionManual          -8.001e-01  2.432e-02 -32.897  < 2e-16 ***
## ownerFourth & Above Owner   -1.418e-01  5.502e-02  -2.577 0.010003 *
## ownerSecond Owner           -4.521e-02  1.840e-02  -2.458 0.014024 *
## ownerTest Drive Car          7.418e-02  1.192e-01   0.622 0.533702
## ownerThird Owner            -1.156e-01  3.058e-02  -3.780 0.000159 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.4705 on 4326 degrees of freedom
## Multiple R-squared:  0.6866, Adjusted R-squared:  0.6857
## F-statistic: 729.1 on 13 and 4326 DF,  p-value: < 2.2e-16
```

The OLS standard errors for the intercept, year, and transmissionManual are less than the bootstrapped standard errors. Those coefficients are also the largest in the OLS model. I think it kind of makes sense since manual transmission and year were pretty significant (never decreased to 0) in the other models. I honestly don't totally know if this makes sense, but it seems to follow the bootstrapping example from class.

Problem 3 (16 points)

Sixty districts in California in 1990 were randomly selected. 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). The data are already in the .Rmd file.

Part a (2 points)

Fit a linear regression model predicting house value from the other variables.

```r
houseMod <- lm(house_value ~ ., data = housing_train)
```

Part b (4 points)

Fit a ridge regression using `cv.glmnet()` to choose the optimal λ. Set the seed before using`cv.glmnet()`.

```r
X <- model.matrix(houseMod)[,-1]
y <- housing_train$house_value
houseRidge <- cv.glmnet(X, y, alpha = 0, thresh = 1e-14)
ridgeFit <- glmnet(X, y, alpha = 0, thresh = 1e-14,
                   lambda = houseRidge$lambda.1se)

coef(ridgeFit)
```

```
## 8 x 1 sparse Matrix of class "dgCMatrix"
##                        s0
## (Intercept) -284483.12543
## longitude    -11598.56594
## latitude     -26018.74488
## age             -60.84117
## rooms            35.43205
## bedrooms         20.74396
## population      -72.98972
## households       72.64514
```

Part c (6 points)

Compute the sum of squared errors for both the OLS model and the ridge regression model for the testing set. Remember to use the exact models you fit to the training sets. The `predict()` functions will be useful here. Compare the results and comment on why the ridge regression performs better here. Slides 26-29 of Notes 7 may be helpful.

```r
testMod <- model.matrix(house_value ~ ., data = housing_test)[,-1]
yTest <- housing_test$house_value

ols <- predict(houseMod, newdata = housing_test)
ridge <- predict(ridgeFit, newx = testMod)

olsSSE <- sum((yTest - ols)^2)
```

```r
ridgeSSE <- sum((yTest - ridge)^2)

print(ridgeSSE)
```

```
## [1] 103503559972
```

```r
print(olsSSE)
```

```
## [1] 111268808003
```

It makes sense because the Ridge model specifically focuses on reducing overfitting. Ridge regression also penalizes sum of square weights, which if I understand it correctly, would mean that by design it'll produce models with a smaller SSE.

Part d (4 points)

Use the `train()` function in the `caret()` package to find the optimal α, λ pair in this situation (using the training set and 5-fold cross validation). For λ, search for values between 0 and 10000 by 100. For α, search for values between 0 and 1 by 0.05. Then fit a model using the optimal α and λ using `glmnet()` and compare the sum of squared errors using this optimized model to the two models in part c. Set a seed before running the `train()` function.

```r
set.seed(1)
grid <- expand.grid(
  alpha = seq(0, 1, by = 0.05),
  lambda = seq(0, 10000, by = 100)
)

trained <- train(
  house_value ~ .,
  data = housing_train,
  method = "glmnet",
  tuneGrid = grid,
  trControl = trainControl(method = "cv", number = 5))

trained$bestTune
```

```
##     alpha lambda
## 127  0.05   2500
```

```r
trained <- glmnet(X, y, alpha = 0.05, lambda = 2500)
predicted <- predict(trained, testMod)

print(sum((yTest - predicted)^2))
```

```
## [1] 85101782862
```

With an $\alpha = 0.05$ and $\lambda = 2500$, this model performs a bit better than the two models in part c, based on the SSE being 11 digits long instead of 12. Although the Ridge and LASSO aren't perfectly balanced, it seems like it has aspects of both which makes it perform well.