MTH 325, Lab #6, Spring 2023 Name _____

Instructions: Follow along with the tutorial portion of the lab. Replicate the code examples in R on your own, along with the demonstration. Then use those examples as a model to answer the questions/perform the tasks that follow. Copy and paste the results of your code to answer questions where directed. Submit your response file and the code used (both for the tutorial and part two). Your code file and your lab response file should each include your name inside. Be sure to follow the write-up directions in the Lab Directions file.

We are going to examine a number of different methods in R to help you with multivariable models, non-linear models and some additional regression methods we've looked at in lecture. We are going to start with 3D scatterplots.

3D graphs can be difficult to interpret and even misleading, particularly when they are static. Many of the 3D graphing programs will produce interactive graphs that you can rotate to overcome some of these drawbacks. These kinds of graphs can be useful in certain contexts, but generally speaking should be used sparingly.

Two packages we are going to look at are rgl and plotly. Plotly is a commercial package that we can use freely in academia in class, but if you publish graphs from their package, there are additional rules required that go beyond just citing the package. Plotly is pretty easy to use, but you should have a backup plan. We will start with the rgl package. It's graphs can be customized, but the default settings look a lot like base-R plotting. (If you are using a mac, you will also need to install quartz https://www.xquartz.org/). The plot will pop up in a new window.

```
library(rgl)
plot3d(mtcars$wt, mtcars$disp, mtcars$mpg, type = "s", size = 0.75, lit = FALSE)
```

We can improve the look of the axis titles.

```
plot3d(mtcars$wt, mtcars$disp, mtcars$mpg,
       xlab = "Weight", ylab = "Displacement", zlab = "MPG",
       size = .75, type = "s", lit = FALSE)
```

The first reference at the bottom also describes how to add lines to the base of the plot to help with the 3D perspective when the interactive features of the graph are immobilized.

Let's look at plotly. For this example, we are also going to use the mtcars dataset, but we'll have to do some cleaning first to get things set up for some of the features.

```
library(plotly)

mtcars$am[which(mtcars$am == 0)] <- 'Automatic'
mtcars$am[which(mtcars$am == 1)] <- 'Manual'
mtcars$am <- as.factor(mtcars$am)
```

In plotly, you can create a plot, and then add modifications to it on subsequent lines, and then plot it only when you are done modifying it.

```
fig <- plot_ly(mtcars, x = ~wt, y = ~hp, z = ~qsec, color = ~am,
               colors = c('#BF382A', '#0C4B8E'))
fig <- fig %>% add_markers()
fig <- fig %>% layout(scene = list(xaxis = list(title = 'Weight'),
                                   yaxis = list(title = 'Gross horsepower'),
                                   zaxis = list(title = '1/4 mile time')))

fig
```

In this graph, we are plotting weight, horsepower and qsec, and then color-coding the resulting values by am so that we can see four variables on the graph. The %>% is a piping feature which is very common in some data cleaning packages in the tidyverse (a family of packages including the package tidy). The third command adds the axis titles.

In this example the color coding is of a categorical variable, but we can also color code with a continuous variable like mpg.

```
fig <- plot_ly(mtcars, x = ~wt, y = ~hp, z = ~qsec,
               marker = list(color = ~mpg,
                             colorscale = c('#FFE1A1', '#683531'),
                             showscale = TRUE))
fig <- fig %>% add_markers()
fig <- fig %>% layout(scene = list(xaxis = list(title = 'Weight'),
                                   yaxis = list(title = 'Gross horsepower'),
                                   zaxis = list(title = '1/4 mile time')),
                      annotations = list(
                        x = 1.13,
                        y = 1.05,
                        text = 'Miles/(US) gallon',
                        xref = 'paper',
                        yref = 'paper',
                        showarrow = FALSE
                      ))
fig
```

Our first fig creates the basic plot, and with the colorscale we set the range of possible values in hexcode. In the third command, in addition to the axis titles, we also add annotations for the scale (legend).

In plotly, you can also add a 5th variable to the plot with a bubble plot, by using the size of the markers to indicate another continuous variable. Plotly also allows you to make a movie to watch the changes over time. However, we'll leave that aside for now. These are not the only plotting packages you can use.

Let's next consider variable transformations. You may recall from examples in class that in the mtcars dataset, mpg and disp appear to be related, but not in a linear fashion.

```
ggplot(data=mtcars,aes(x=disp,y=mpg))+geom_point()
```

We can try to improve the linearity by applying a log transformation.

```
mtcars$log_disp<-log(mtcars$disp)
ggplot(data=mtcars,aes(x=log_disp,y=mpg))+geom_point()
```

We can confirm our suspicion that the correlation has improved by comparing the correlation values.

```
cor(mtcars$mpg, mtcars$disp,  method = "pearson")
cor(mtcars$mpg, mtcars$log_disp,  method = "pearson")
```

We can apply other transformations the same way. Functions like sqrt() will apply term-by-term in a vector just like log (which is the natural log by default).

One note is that you need to think about the mathematics of the function you are applying. You can't apply either log or square root to any negative values (you could try a cube root instead). Logs also can't be applied to any data that contains a 0 value. In this situation, add a very small value like 0.0001 to every value in the dataset. This won't really change any other values, but it will allow the log function to operate on the values that were previously zero. The size of this adjustment should be smaller than any other non-zero value in the dataset.

Sometimes, when transforming variables, depending on the method used, you may run into issues with two columns in your dataset that have the same name. A resource on changing the names of columns is included in the references in case you run into this issue.

You can apply these transformations before applying the model, or directly inside the linear model function. In this first example, we are modeling the mpg and disp variables directly. The second two models are the same. One applies the log transformation in the model formula, and the third one applies the previously transformed variable.

```
model1<-glm(data=mtcars,formula=mpg~disp)
summary(model1)

model2<-glm(data=mtcars,formula=mpg~log(disp))
summary(model2)

model3<-glm(data=mtcars,formula=mpg~log_disp)
summary(model3)
```

If we want to include an interaction term in a linear model, we can do it one of two ways.

```
model4<-glm(data=mtcars,formula=mpg~disp*wt)
summary(model4)

model5<-glm(data=mtcars,formula=mpg~disp+wt+disp:wt)
summary(model5)
```

In the first method, we are just specifying the interaction, and the main effects will be automatically included. We saw this in the ANOVA models. We can also specify an interaction specifically in the second case. This method is useful if we have several main effect variables and we don't wish to include all the possible interaction terms, just some of them.

If we want to use polynomial terms in regression models, we need to use the I() function (this is a capital "i") to specify the power terms. Or, you can do the transformation first and make the power terms an extra variable as we did before with the log transformation.

```
model6<-glm(mpg ~ gear + I(gear^2), data = mtcars)
summary(model6)
```

A full quadratic model would include both the squared terms, the interaction term and the linear terms.

```
model7<-glm(mpg ~ (disp*gear)^2+I(gear^2)+I(disp^2), data = mtcars)
summary(model7)
```

Although, as our model becomes more complex, with additional variables, fewer of them are likely to be significant and will have to be trimmed.

To pair back our model, we can use penalized regression methods like ridge regression or LASSO regression. For this, we'll need the glmnet package.

To use the glmnet package, we need to set up the data to put it into a form the package likes.

```
library(glmnet)

x <- model.matrix(mpg~., mtcars)[,-1]
y<-mtcars$mpg
```

We cross validation is a method of running different subsets of the data through the model to select the best results. Here, it is built in to the glmnet package, and we can use it to select the best parameter for our penalty. Setting alpha=0 is for ridge regression. If we set this parameter to 1, we get LASSO regression, and a value between 0 and 1 is an elastic net.

```
cv <- cv.glmnet(x, y, alpha = 0)
# Display the best lambda value
cv$lambda.min
```

Then we can run the model with the selected λ and then extract the coefficients from the model. Note that with ridge regression, variables are not eliminated, but it does drive less important coefficients to be closer to zero.

```
model8 <- glmnet(x, y, alpha = 0, lambda = cv$lambda.min)
coef(model8)
```

We can compare the results with LASSO regression. You'll need to rerun the cross-validation to obtain a new value of lambda with alpha=1.

```
cv <- cv.glmnet(x, y, alpha = 1)
# Display the best lambda value
cv$lambda.min

model9 <- glmnet(x, y, alpha = 1, lambda = cv$lambda.min)
coef(model9)
```

Compare the results of the coefficients. Any that are displayed as . were removed from the model (the coefficients are zero). Are the eliminated ones in LASSO the same as the smallest coefficients in ridge?

Tasks:

1. Use the data in **325lab6data1.xlsx**. Create scatterplots of the data, including at least one 3D plot. Test the data to apply the log, square root and square transformations of miles and age to predict maintenance expense. Describe your findings. Did any of the transformations produce better results than the untransformed variables? What final model did you settle on?

2. Use the data in **325lab6data2.xlsx**. Use the remaining variables to predict the number of weekly riders. Perform whatever variables transforms that is necessary, then perform ridge and LASSO regression. Describe the resulting models and compare predictions based on both models (choose values for the predictor variables inside the range of the original data).

Resources:

1. https://r-graphics.org/recipe-miscgraph-3d-scatter
2. https://plotly.com/r/3d-scatter-plots/
3. https://cran.r-project.org/web/packages/scatterplot3d/vignettes/s3d.pdf
4. https://www.statology.org/transform-data-in-r/
5. http://www.sthda.com/english/wiki/correlation-test-between-two-variables-in-r
6. https://sparkbyexamples.com/r-programming/change-column-names-of-the-r-dataframe/
7. https://cran.r-project.org/web/packages/interactions/vignettes/interactions.html
8. https://www.geeksforgeeks.org/polynomial-regression-in-r-programming/
9. http://www.sthda.com/english/articles/37-model-selection-essentials-in-r/153-penalized-regression-essentials-ridge-lasso-elastic-net/