stat448_ass_2

Noah King - 96851177

2024-09-17

Preamble: The rmd file assumes all data is just in the same directory. LaTeX embeddings generated with generative AI. For the purpose of the model in q2, I just copied the output into chatgpt of the previous cell and asked for it to look nice in latex for rmd.

Q1

(40 marks) In the folder for assignment 2 there are 2 files: the RData file Residen and the excel file Residential-Building-Data-Set.xlsx. Residenis a copy of the dataset in the excel file where the variables names have been changed for use in R. The excel file also contains a description of the variables. More information on the dataset can be obtained on the UCI webpage: https://archive.ics.uci.edu/ml/datasets/Residential+Building+Data+Set.

```r
load('STAT448-24S2 Assignment 2 Residen.RData')
housing_data <- Residen
# Remove columns that contain NA values
housing_data <- housing_data[, colSums(is.na(housing_data)) == 0]
# I've just defined a discrete var as having <10 unique vars.....

#rename for readability
```

We also one hot encode the completion quarters. It's arguable whether or not the year should be, we consider it a continuous var

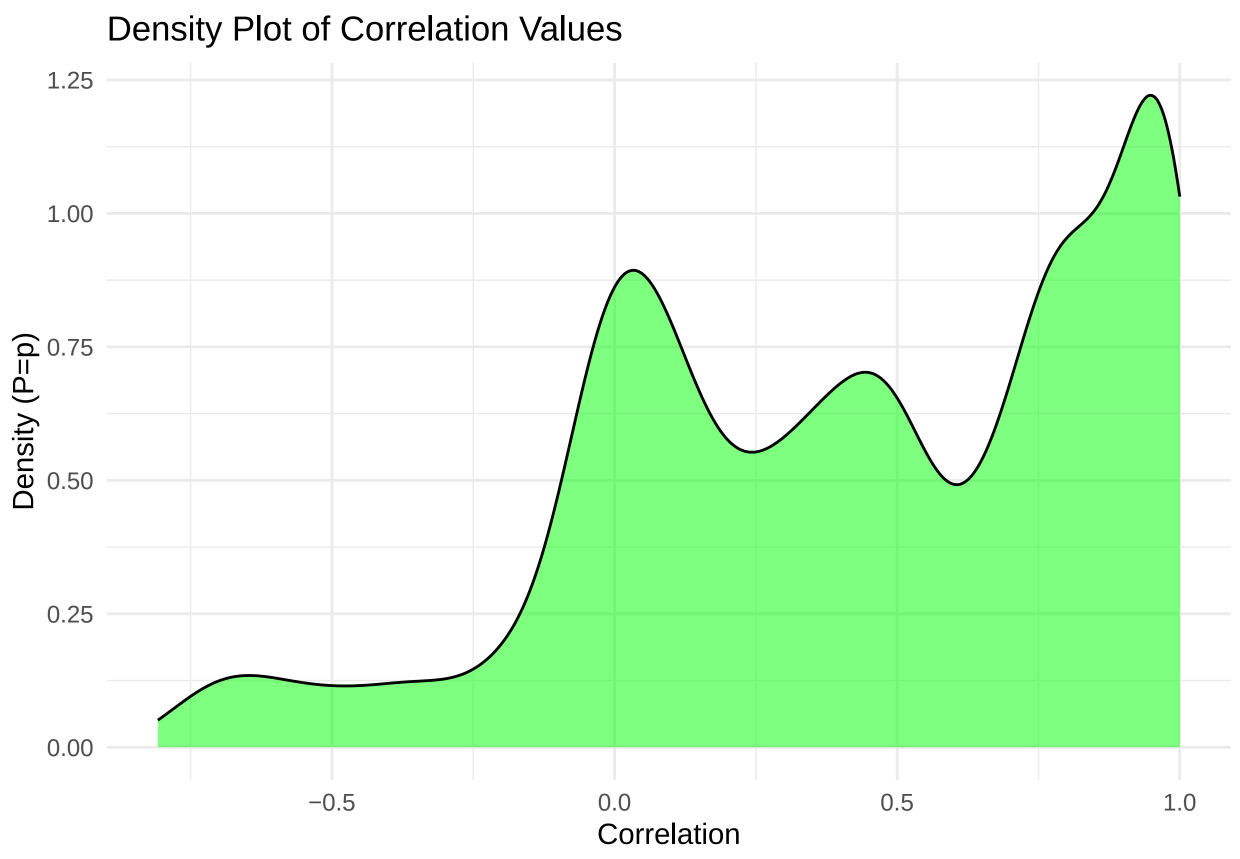
```r
housing_data$`START QUARTER` <- as.factor(housing_data$`START QUARTER`)
housing_data$`COMPLETION QUARTER` <- as.factor(housing_data$`COMPLETION QUARTER`)


start_quarter_dummies <- model.matrix(~ `START QUARTER` - 1, data = housing_data)
end_quarter_dummies <- model.matrix(~ `COMPLETION QUARTER` - 1, data = housing_data)

housing_data <- cbind(housing_data, start_quarter_dummies, end_quarter_dummies)
housing_data$`START QUARTER` <- NULL
housing_data$`COMPLETION QUARTER` <- NULL
```

1. Explore with appropriate graphical tools the correlation between the variables. What are your main observations? Remember to provide evidence to support your observations.

There are 109 variables in our dataset; A useful model is typically built out on linearly independant variables; strong dependance or correlation between these rows can introduce noise, negatively affecting the accuracy of our model, but also potentially introducing redundant compute.

All pairwise combinations of variables are plotted below per their frequency. It's not important (for the plot) which is which, but each combination appears only once.

```r
cor_matrix <- cor(housing_data, use = "complete.obs")

all_pairs <- which(abs(cor_matrix) > 0.0 & abs(cor_matrix) < 1, arr.ind = TRUE)
#we can update this later to check on the actual correlation.




all_pairs <- data.frame(
  Var1 = rownames(cor_matrix)[all_pairs[, 1]],
  Var2 = colnames(cor_matrix)[all_pairs[, 2]],
  Correlation = cor_matrix[all_pairs]
)

#ensure unique pairs only (compute overhead doesn't seem bad from cor_matrix, still <1s compute)
all_pairs <- all_pairs %>%
  filter(Var1 < Var2) %>%
  arrange(desc(abs(Correlation)))

ggplot(all_pairs, aes(x = Correlation)) +
  geom_density(fill = "green", alpha = 0.5) +
  labs(x = "Correlation", y = "Density (P=p)", title = "Density Plot of Correlation Values") +
  theme_minimal()
```



As there are a great number of variables in this data set; we consider the *quick and dirty* measure above. Essentially; all pairs are well correlated; with around 1/3rd of pairs above the 0.85 correlation threshold.

In the interest of applying a the context of the dataset again:

The dataset in question, contains a number of same named columsn in *time lags* as below:

Let's add the column names back in from the raw data:

PROJECT PHYSICAL AND FINANCIAL VARIABLES : V-1 through V-8

ECONOMIC VARIABLES AND INDICES IN TIME LAG 1 : V-9 through V-27 (18 cols)

ECONOMIC VARIABLES AND INDICES IN TIME LAG 2 : V-28 through V-46 (18 cols)

ECONOMIC VARIABLES AND INDICES IN TIME LAG 3 : V-37 through V-65 (18 cols)

ECONOMIC VARIABLES AND INDICES IN TIME LAG 4 : V-66 through V-85 (18 cols)

ECONOMIC VARIABLES AND INDICES IN TIME LAG 5 : V-85 through V-103 (18 cols)

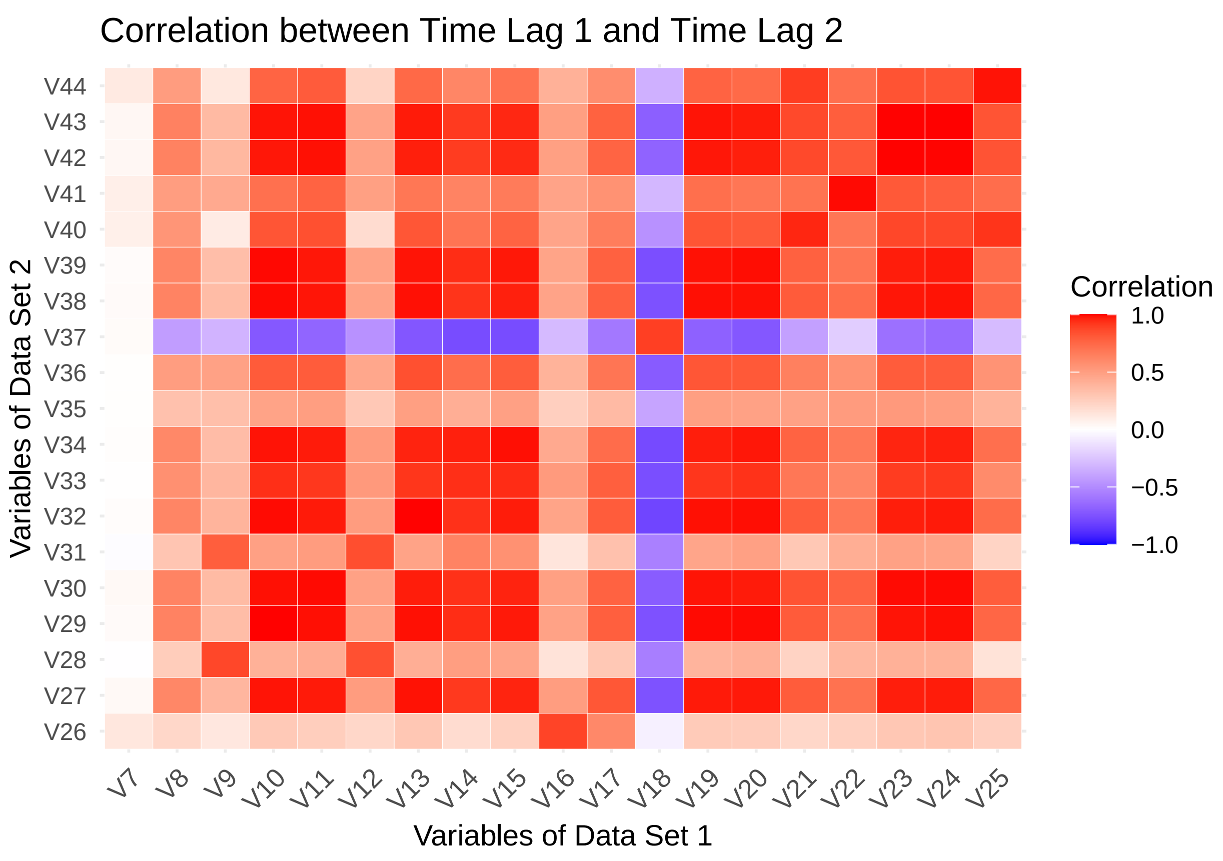
Prediction_vars : V-104 and V-105

Additional Cols for one hot x2 for quarters

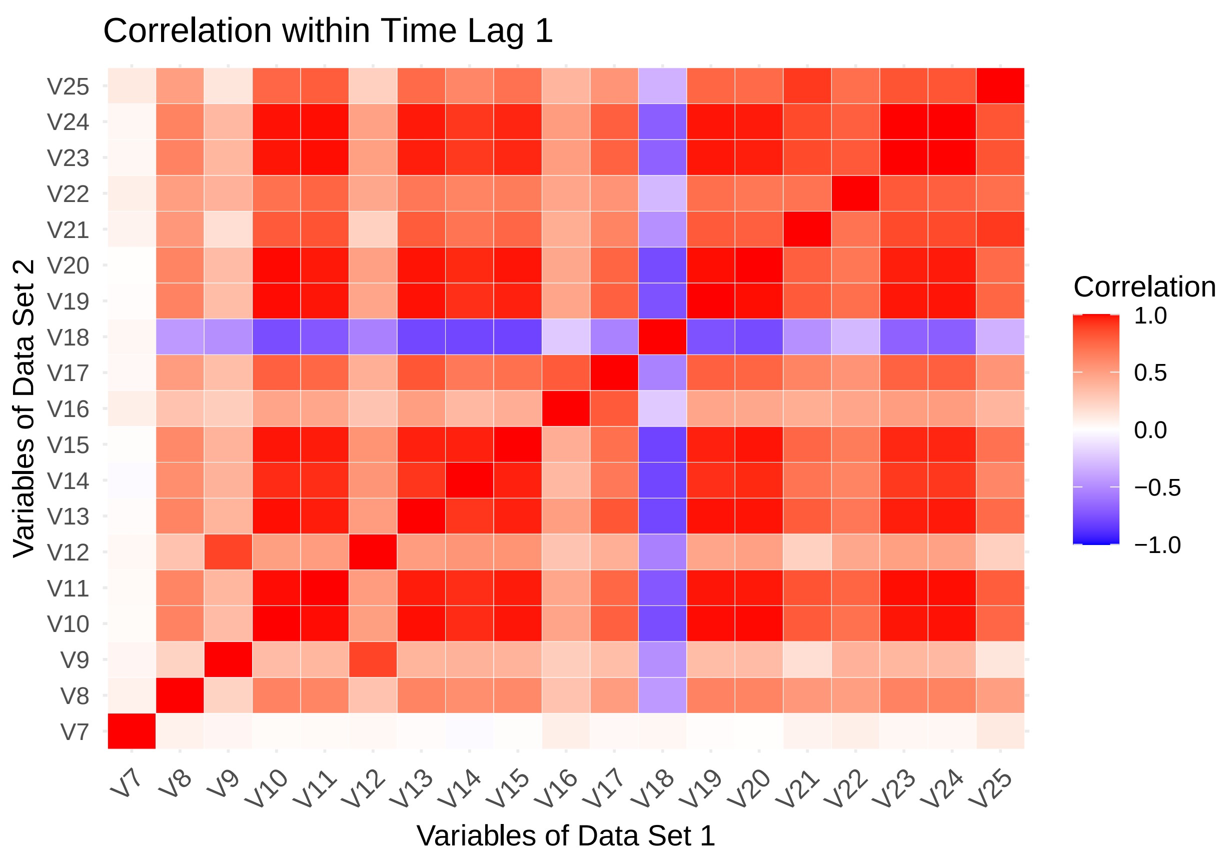
```
library(ggplot2)
library(reshape2)
```

```
##
## Attaching package: 'reshape2'

## The following object is masked from 'package:tidyr':
##
##     smiths
```

```
library(RColorBrewer)
#Purely for looking
time_lag1 <- housing_data[, 9:27]
time_lag2 <- housing_data[, 28:46]
time_lag3 <- housing_data[, 47:65]
time_lag4 <- housing_data[, 66:85]


#handy little function, courtesy of stack overflow...
plot_correlation_heatmap <- function(data1, data2, title) {
  correlation_matrix <- cor(data1, data2, use = "pairwise.complete.obs")
  melted_corr <- melt(correlation_matrix)

  ggplot(data = melted_corr, aes(Var1, Var2, fill = value)) +
    geom_tile(color = "white") +
    scale_fill_gradient2(low = "blue", high = "red", mid = "white",
                         midpoint = 0, limit = c(-1, 1), space = "Lab",
                         name = "Correlation") +
    theme_minimal() +
    theme(axis.text.x = element_text(angle = 45, vjust = 1,
                                     size = 10, hjust = 1)) +
    labs(title = title, x = "Variables of Data Set 1", y = "Variables of Data Set 2")
}
plot_correlation_heatmap(time_lag1, time_lag2, "Correlation between Time Lag 1 and Time Lag 2")
```



```
plot_correlation_heatmap(time_lag1, time_lag1, "Correlation within Time Lag 1")
```



Where each of the timelags represents a 1q delay of the same variables.

It's reasonable to assume (for now) that matching named variables from each timelag would be well correlated.

In fact what we do see is that each of the (ascending) columns are somewhat similarly correlated. There is a number of significant correlations in this example of Time lag 1 and 2; enough to indicate redundancy (i.e. lots of red in our graphic)

But also notably within a randomly selected time lag there is also great colinearity.

For example columns 4 in each lag is are well correlated (there's too much to just read off the graph.)

It holds that our data is greatly redundant.

(note, im sure the intention of removing the column names from the raw data is to *not* do this, but I haven't a clue what else I could use for the number of variables here...)

Let's also consider the *most* correlated variables in the dataset. There are, simply, Lots of them. Noting that there are around 10'000 permittable combinations. Consider the number of variables correlated greater tahn 0.9:

```r
all_pairs <- which(abs(cor_matrix) > 0.9 & abs(cor_matrix) < 1, arr.ind = TRUE)
#we can update this later to check on the actual correlation.
nrow(all_pairs)
```

```
## [1] 2576
```

And further the number of variable correlated < 0.85:

```r
all_pairs <- which(abs(cor_matrix) < 0.85 & abs(cor_matrix) < 1, arr.ind = TRUE)
#we can update this later to check on the actual correlation.
nrow(all_pairs)
```

```
## [1] 10090
```

Essentially, our dataset is awfully colinear, it should be clear from the R code in the chunks above that *all_pairs* can be sorted to determine the most related. All such correlations would result in an unreasonable amount of content for this document; even those well correlated.

So, therefore, we're not going to cull any of these redundant variables now. (In the interest of the assignment handout's wording) however it is usually a prudent move to cull any columns who are colinear to some other column, especially with as high of a correlation as we're seeing here, we're essentially wasting compute and adding noise to the model.

Also, we will keep the quarters etc in as one-hots… (years also included as there's no reason not to)

2.Please fit a linear regression model to explain the "actual sales price" (V104) in terms of the of the other variables excluding the variable "actual construction costs" (V105). Explain the output provided by the summary function.

We're going to let AIC handle cleaning up the highly correlated columns, our model shouldn't be affected by this.

```r
train_indices <- sample(seq_len(nrow(housing_data)), size = 0.8 * nrow(housing_data))
df_train <- housing_data[train_indices, ]
df_test <- housing_data[-train_indices, ]
cat("Training set dimensions:", dim(df_train), "\n")
```

```
## Training set dimensions: 297 115
```

```r
cat("Test set dimensions:", dim(df_test), "\n")
```

```
## Test set dimensions: 75 115
```

```r
# Exclude V105 (actual construction costs) from the model
# Fit the linear regression model, predicting V4 (actual sales price) and excluding V5 (construction co
model <- lm(V104 ~ . - V105, data = df_train)
summary(model)
```

```
##
## Call:
## lm(formula = V104 ~ . - V105, data = df_train)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -858.10  -45.09    0.00   45.45  448.83
##
## Coefficients: (39 not defined because of singularities)
##                              Estimate Std. Error t value Pr(>|t|)
## (Intercept)                 2.521e+04  1.637e+05   0.154 0.877760
## `START YEAR`               -2.337e+02  2.241e+03  -0.104 0.917015
## `COMPLETION YEAR`          -1.030e+02  1.074e+02  -0.959 0.338830
## V1                                -6.683e+00  2.441e+00  -2.738 0.006689 **
## V2                                 4.142e-02  3.216e-02   1.288 0.199132
## V3                                -1.853e-01  9.258e-02  -2.001 0.046571 *
## V4                                 1.460e-01  6.787e-02   2.151 0.032523 *
## V5                                -1.409e+00  3.780e-01  -3.728 0.000245 ***
## V6                                 1.267e-01  6.738e-02   1.881 0.061279 .
## V7                                 6.677e+01  2.667e+01   2.504 0.013012 *
## V8                                 1.219e+00  1.975e-02  61.695  < 2e-16 ***
## V9                                -9.157e-02  4.616e-01  -0.198 0.842941
## V10                                6.391e+01  8.570e+02   0.075 0.940626
## V11                                2.719e+01  1.026e+02   0.265 0.791253
## V12                               -5.126e+01  2.040e+02  -0.251 0.801841
## V13                               -2.101e-02  4.617e-02  -0.455 0.649508
## V14                               -1.016e-01  3.860e-01  -0.263 0.792540
## V15                               -2.331e+01  1.082e+02  -0.215 0.829609
## V16                                1.535e+00  7.194e+00   0.213 0.831176
## V17                                2.987e-02  1.566e-01   0.191 0.848951
## V18                               -4.023e+01  3.282e+02  -0.123 0.902539
## V19                                5.341e-01  2.601e+00   0.205 0.837496
## V20                               -5.977e-01  2.793e+00  -0.214 0.830721
## V21                                2.288e-02  1.686e-01   0.136 0.892150
## V22                               -1.434e-01  9.254e-01  -0.155 0.876978
## V23                                2.035e+01  1.489e+02   0.137 0.891475
## V24                               -2.847e+01  4.088e+02  -0.070 0.944542
## V25                                8.795e-02  7.327e-01   0.120 0.904569
## V26                                1.316e-03  4.575e-02   0.029 0.977079
## V27                               -4.840e-04  1.019e-02  -0.047 0.962166
## V28                               -3.120e-02  3.160e-01  -0.099 0.921439
## V29                               -9.126e+01  8.308e+02  -0.110 0.912627
## V30                                4.317e+01  2.792e+02   0.155 0.877266
## V31                                6.393e+01  6.547e+02   0.098 0.922301
## V32                                1.736e-02  2.532e-02   0.686 0.493715
## V33                               -1.613e-02  2.611e-01  -0.062 0.950784
## V34                               -3.780e+01  1.951e+02  -0.194 0.846541
## V35                                1.694e+00  1.213e+01   0.140 0.889039
## V36                               -2.646e-02  1.000e-01  -0.265 0.791580
## V37                                1.019e+02  2.054e+02   0.496 0.620454
## V38                                1.240e+00  4.960e+00   0.250 0.802885
## V39                               -2.904e-01  1.094e+00  -0.265 0.790908
## V40                               -1.136e-01  5.408e-01  -0.210 0.833896
## V41                               -3.970e-02  7.819e-01  -0.051 0.959556
## V42                                1.355e+02  3.520e+02   0.385 0.700697
## V43                                1.093e+02  6.048e+02   0.181 0.856721
## V44                               -1.090e-01  3.487e-01  -0.313 0.754924
## V45                                5.149e-03  3.366e-02   0.153 0.878555
## V46                               -2.967e-04  1.016e-02  -0.029 0.976720
## V47                                5.010e-02  1.967e-01   0.255 0.799168
## V48                               -6.313e+01  4.084e+02  -0.155 0.877298
## V49                               -4.067e+01  3.662e+02  -0.111 0.911673
## V50                                3.173e+01  3.604e+02   0.088 0.929905
## V51                                2.155e-02  3.266e-02   0.660 0.510020
## V52                                1.850e-01  1.176e+00   0.157 0.875137
## V53                                2.395e+01  1.216e+02   0.197 0.844011
## V54                               -3.107e-02  6.613e+00  -0.005 0.996255
## V55                          -1.066e-03  7.198e-02  -0.015 0.988195
## V56                          -9.724e+01  3.867e+02  -0.251 0.801701
## V57                           1.135e+00  5.439e+00   0.209 0.834918
## V58                          -1.118e+00  5.457e+00  -0.205 0.837892
## V59                          -1.223e-01  6.755e-01  -0.181 0.856446
## V60                          -5.536e-02  3.114e-01  -0.178 0.859047
## V61                          -5.278e+01  3.411e+02  -0.155 0.877185
## V62                          -1.071e+02  3.771e+02  -0.284 0.776666
## V63                           4.839e-02  3.337e-01   0.145 0.884844
## V64                          -4.580e-03  4.038e-02  -0.113 0.909796
## V65                          -7.296e-04  1.675e-03  -0.436 0.663531
## V66                          -1.503e-01  4.723e-01  -0.318 0.750678
## V67                          -2.266e+01  1.596e+02  -0.142 0.887219
## V68                           7.628e+01  5.401e+02   0.141 0.887811
## V69                           7.639e+01  2.029e+02   0.376 0.706979
## V70                          -1.146e-02  2.684e-02  -0.427 0.669878
## V71                                  NA         NA      NA       NA
## V72                                  NA         NA      NA       NA
## V73                                  NA         NA      NA       NA
## V74                                  NA         NA      NA       NA
## V75                                  NA         NA      NA       NA
## V76                                  NA         NA      NA       NA
## V77                                  NA         NA      NA       NA
## V78                                  NA         NA      NA       NA
## V79                                  NA         NA      NA       NA
## V80                                  NA         NA      NA       NA
## V81                                  NA         NA      NA       NA
## V82                                  NA         NA      NA       NA
## V83                                  NA         NA      NA       NA
## V84                                  NA         NA      NA       NA
## V85                                  NA         NA      NA       NA
## V86                                  NA         NA      NA       NA
## V87                                  NA         NA      NA       NA
## V88                                  NA         NA      NA       NA
## V89                                  NA         NA      NA       NA
## V90                                  NA         NA      NA       NA
## V91                                  NA         NA      NA       NA
## V92                                  NA         NA      NA       NA
## V93                                  NA         NA      NA       NA
## V94                                  NA         NA      NA       NA
## V95                                  NA         NA      NA       NA
## V96                                  NA         NA      NA       NA
## V97                                  NA         NA      NA       NA
## V98                                  NA         NA      NA       NA
## V99                                  NA         NA      NA       NA
## V100                                 NA         NA      NA       NA
## V101                                 NA         NA      NA       NA
## V102                                 NA         NA      NA       NA
## V103                                 NA         NA      NA       NA
## `\\`START QUARTER\\`1`               NA         NA      NA       NA
## `\\`START QUARTER\\`2`               NA         NA      NA       NA
## `\\`START QUARTER\\`3`               NA         NA      NA       NA
## `\\`START QUARTER\\`4`               NA         NA      NA       NA
## `\\`COMPLETION QUARTER\\`1`   2.506e+01  7.202e+01   0.348 0.728167
## ‘\\‘COMPLETION QUARTER\\‘2‘ -3.654e+00  4.684e+01  -0.078 0.937893
## ‘\\‘COMPLETION QUARTER\\‘3‘         NA         NA      NA       NA
## ‘\\‘COMPLETION QUARTER\\‘4‘         NA         NA      NA       NA
## ---
## Signif. codes:  0 ’***’ 0.001 ’**’ 0.01 ’*’ 0.05 ’.’ 0.1 ’ ’ 1
##
## Residual standard error: 138.5 on 222 degrees of freedom
## Multiple R-squared:  0.9893, Adjusted R-squared:  0.9858
## F-statistic: 278.2 on 74 and 222 DF,  p-value: < 2.2e-16
```

Firstly, that sure is a lot of *NA*. This is likely due ot the multicolinearity (38 not defined) as many of our predictors are so highly correlated that the model cannot estimate their individual effects on v104 (colinearity - redundant information)

Reading from the model summary:

The model has a residual standard error of 150.8, indicating the average distance that the actual value from the regression line. The Multiple R-squared value of 0.9887 suggests that 98.87% of the variance in the actual sales price (V104) is explained by the predictor variables. The Adjusted R-squared value of 0.9849 accounts for the (large) number of predictors in the model and still indicates a very high explanatory power. The F-statistic of 257.6 with a corresponding p-value $<$ 2.2e-16 indicates that the model is statistically significant, i.e at least some of the predictor variables are reliably associated with variations in the dependent variable.

Also V7,V8,V5,V1 ar ethe greatest contributors to our model.

3. Fit a linear regression model to explain the "actual sales price" (V104) in terms of other variables, excluding the variable "actual construction costs" (V105) using (a) backwards selection and (b) stepwise selection. Compare these two models in terms of outputs, computational time and holdout mean square error. (Hint: summarising your results in a table allows for easy comparison).

As earlier; we consider the entire set of explanatory variables (on the training set)

```r
# Fit the full model using ALL explanatory vars
full_model <- lm(V104 ~ ., data = df_train)

# Backward Selection
b_start_time <- proc.time()
backward_model <- step(full_model, direction = "backward", trace = FALSE)
b_end_time <- proc.time() - b_start_time

# Stepwise Selection & timing just using inbuilts
sw_start_time <- proc.time()
stepwise_model <- step(full_model, direction = "both", trace = FALSE)
sw_end_time <- proc.time() - sw_start_time

# preds
backward_predictions <- predict(backward_model, newdata = df_test)
stepwise_predictions <- predict(stepwise_model, newdata = df_test)

# Compute Mean Squared Error on test set (holdout set per handout)
mse_backward <- mean((df_test$V104 - backward_predictions)^2)
mse_stepwise <- mean((df_test$V104 - stepwise_predictions)^2)


elapsed_time_backward <- b_end_time["elapsed"]
elapsed_time_stepwise <- sw_end_time["elapsed"]
```

```
#pretty output
timing_results <- data.frame(
  Model = c("Backward Selection", "Stepwise Selection"),
  Elapsed_Time_Seconds = c(elapsed_time_backward, elapsed_time_stepwise),
  MSE = c(mse_backward, mse_stepwise),
  Num_Explanatory_Variables = c(length(coef(backward_model)) - 1, length(coef(stepwise_model)) - 1)  #
)
kable(timing_results, caption = "Model Fitting Results: Number of Variables, Processing Time, and MSE")
```

Table 1: Model Fitting Results: Number of Variables, Processing Time, and MSE

Model	Elapsed_Time_Seconds	MSE	Num_Explanatory_Variables
Backward Selection	9.869	55217.2	32
Stepwise Selection	12.451	54760.9	36

Reading from the table, we can see that the stepwise selction yielded an overall better MSE, yet a more complex model, it holds that on different test data either may be better, as they are very close in their MSE on this seed.

The observed differences in elapsed time reflect the more systematic approach of Backward Selection, which only removes variables, however Stepwise Selection alternates between adding and removing cols, resulting in higher computational time.

The number of variables in either does hint slightly toward overfitting; simply due to the large number of correlated vars and how few are <0.85 correlation.

We see later in LASSO and RIDGE approaches to removing predictors….

4. Fit a linear regression model to explain the "actual sales price" (V104) in terms of the other variables, excluding the variable "actual construction costs" (V105) using (a) Ridge regression (b) LASSO regression with an appropriate lambda determined by cross validation. Compare these two models in terms of outputs, computational time and cross validation mean square error. Please give a possible explanation of why one method is better than the other in this case, considering also results from backwards and stepwise selection. (Hint: summarising your results in a table allows for easy comparison).

We will set this up slightly differnetly than above, just for the sake of well fiting glmnet

We simply set alpha = 0 or 1 to be pure lasso and ridge and will just use glmnet

```
# Prepare data matrices
x_train <- model.matrix(V104 ~ . - V105, data = df_train)  # Exclude V105
y_train <- df_train$V104
x_test <- model.matrix(V104 ~ . - V105, data = df_test)
y_test <- df_test$V104

# Ridge
ridge_start_time <- proc.time()
ridge_cv <- cv.glmnet(x_train, y_train, alpha = 0)  # alpha = 0 for Ridge
optimal_lambda_ridge <- ridge_cv$lambda.min
ridge_predictions <- predict(ridge_cv, s = optimal_lambda_ridge, newx = x_test)
ridge_mse <- mean((y_test - ridge_predictions)^2)
ridge_cv_mse <- min(ridge_cv$cvm)  # Minimum CV MSE
```

```
ridge_num_predictors <- sum(coef(ridge_cv, s = optimal_lambda_ridge) != 0) - 1  # Exclude intercept

ridge_end_time <- proc.time() - ridge_start_time

# LASSO
lasso_start_time <- proc.time()
lasso_cv <- cv.glmnet(x_train, y_train, alpha = 1)  # alpha = 1 for LASSO
optimal_lambda_lasso <- lasso_cv$lambda.min
lasso_predictions <- predict(lasso_cv, s = optimal_lambda_lasso, newx = x_test)
lasso_mse <- mean((y_test - lasso_predictions)^2)
lasso_cv_mse <- min(lasso_cv$cvm)  # Minimum CV MSE
lasso_num_predictors <- sum(coef(lasso_cv, s = optimal_lambda_lasso) != 0) - 1  # Exclude intercept

lasso_end_time <- proc.time() - lasso_start_time

# Pretty table
comparison_results <- data.frame(
  Model = c("Ridge", "LASSO"),
  Optimal_Lambda = c(optimal_lambda_ridge, optimal_lambda_lasso),
  CV_MSE = c(ridge_cv_mse, lasso_cv_mse),  # Add CV MSE
  Test_MSE = c(ridge_mse, lasso_mse),
  Elapsed_Time_Seconds = c(ridge_end_time["elapsed"], lasso_end_time["elapsed"]),
  Num_Predictors = c(ridge_num_predictors, lasso_num_predictors)  # Add number of predictors
)

# Display the table
kable(comparison_results, caption = "Comparison of Ridge and LASSO Regression Models")
```

Table 2: Comparison of Ridge and LASSO Regression Models

Model	Optimal_Lambda	CV_MSE	Test_MSE	Elapsed_Time_Seconds	Num_Predictors
Ridge	113.442625	52114.84	95965.86	0.099	113
LASSO	2.444047	30641.25	57689.47	0.049	28

Note: We also include the Optimal lambda value in the table (that reduces CV mse).

Lasso seems to be a much better model choice here. The lower CV MSE and also test MSE indicate that it would be a better fit to the actual data; it also computes significantly faster. We know already that the dataset is hugely colinear; LASSO shrinks the col pool (pool of vars) by *picking* a single member of the correlated 'group'. Having less variables in the model here will contribute to reduced noise and overall better performance on unseen data. The L1 penalty applied shrinks the coefficient of 'pointless' or less impact features towards zero, which helps manage the same idea.

The potential for LASSO to remove (and have removed) so many redundant columns is brilliant for our model and the reduction of noise quantifies this.

NOTE: the order in which LASSO remvoes columsn is (not unknown) implementation dependant; some other implementation than the *lib* we've imported may yield a different combination of columns predicated on their order as LASSO is not fully exhaustive. Our model is, in that context, good enough.

Question 2

(25 marks) Also included in the assignment folder is the file parkinsons.csv, this dataset1 contains information on 42 patients with Parkinson's disease. The outcome of interest is UPDRS, which is the total unified Parkinson's disease rating scale. The first 96 features, X1–X96, have been derived from audio recordings of speech tests (i.e. there has already been a process of feature extraction) while feature X97 is already known to be informative for UPDRS.

Read in the dataset and set up the model matrix X.

Next, randomly split the data into a training set with 30 patients and a test set with 12 patients – remembering to say what your RNG seed was.

Standardize your training and test sets before your analysis: X = scale(X) so that the absolute size of the estimated coefficients will give a measure of the relative importance of the features in the fitted model.

as per usual with R it is prudent to reset the environment before moving on

```r
# Clear workspace and set seed
rm(list = ls())
set.seed(123)

# Load the data
parkinsons_data = read.csv('Parkinsons data.csv')

# Set the matrix X without needing to remove any columns
X <- model.matrix(UPDRS ~ ., parkinsons_data)[,-1]  # This already drops the intercept
Y <- parkinsons_data$UPDRS
head(Y)
```

```
## [1] 34.39833 14.03900 25.72900 16.17750 39.24000 39.05900
```

```r
# Scale X
X <- scale(X)

# Split test & train (30/12)
train <- sample(1:nrow(X), 30)
test <- setdiff(1:nrow(X), train)
train
```

```
##  [1] 31 15 14  3 37 40 25 26 27  5 19 34 36 28 30  9 39  8  7 10 42 32  4 38 17
## [26] 11 24 33 12 23
```

```r
test
```

```
##  [1]  1  2  6 13 16 18 20 21 22 29 35 41
```

1. Confirm that a linear model can fit the training data exactly. Why is this model not going to be useful?

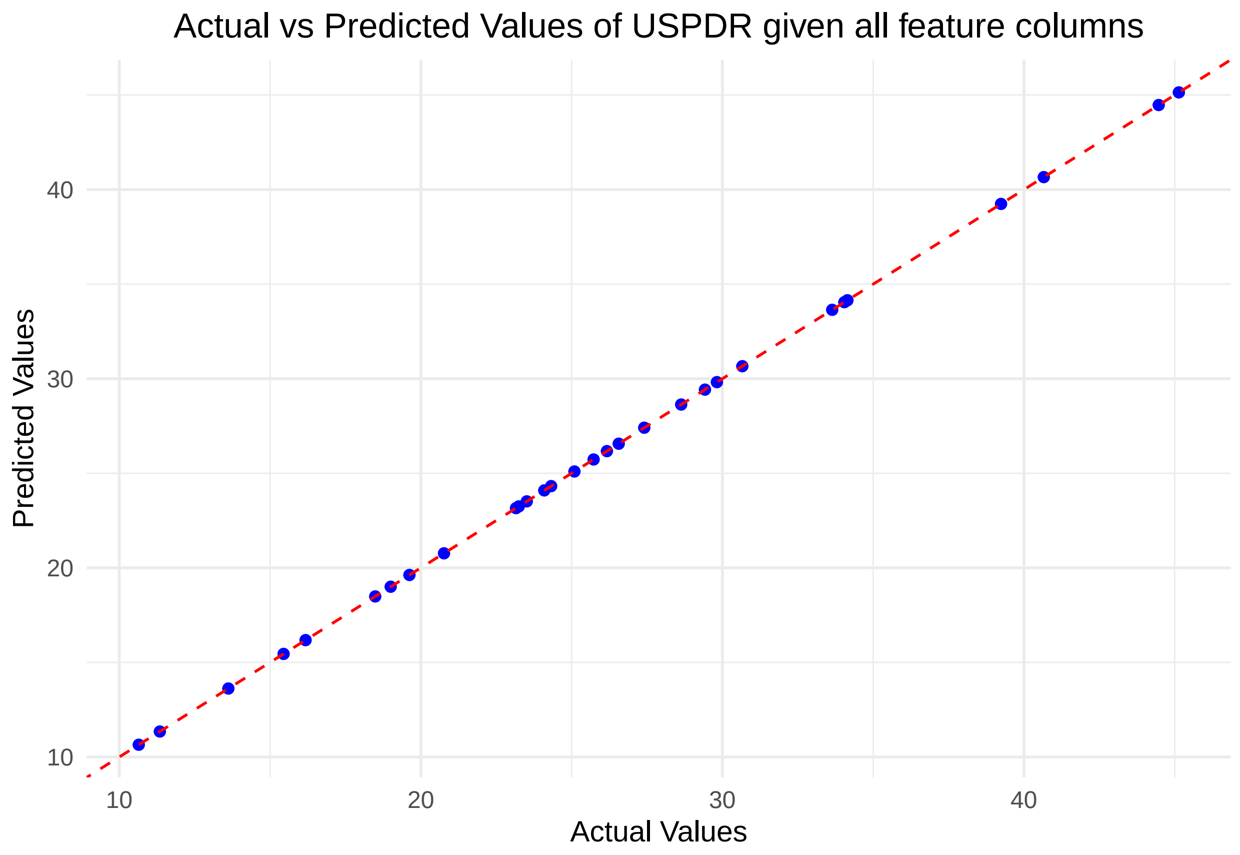
   If the MSE is zero on our test set (that is the distnace between each prediction and it's actual value is zero) then the data is perfectly fit by the model

```r
linear_model <- lm(Y[train] ~ X[train, ])
fitted_values <- predict(linear_model, newdata = data.frame(X = X[train, ]))

actuals <- as.vector((Y[train]))
preds <- as.vector(fitted_values)

fitting_error <- all.equal(actuals, preds)
fitting_error
```

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

```r
results_df <- data.frame(
  Actuals = actuals,
  Predictions = preds
)

# Create the ggplot
ggplot(results_df, aes(x = Actuals, y = Predictions)) +
  geom_point(color = "blue") +  # Points for actuals vs predictions
  geom_abline(slope = 1, intercept = 0, color = "red", linetype = "dashed") +  # 45-degree line
  labs(
    title = "Actual vs Predicted Values of USPDR given all feature columns",
    x = "Actual Values",
    y = "Predicted Values"
  ) +
  theme_minimal() +  # Minimal theme for a clean look+
   scale_color_manual(values = c("Actual Points" = "blue", "Model Prediction" = "red")) + # Specify cus

  theme(
    plot.title = element_text(hjust = 0.5),  # Center the title
    legend.position = "none"  # Remove legend since it's not needed
  )
```

```
## Warning: No shared levels found between ‘names(values)‘ of the manual scale and the
## data’s colour values.
```



In fact we can see above that the MSE is zero (or very close to zero given some floating point calc issues in R)

Now use the LASSO to fit the training data, using leave-one-out cross-validation to find the tuning parameter lambda. (This means nfolds=30. You will also find it convenient to set grid = 10^seq(3,-1,100) and thresh=1e-10). What is the optimal value of lambda and what is the resulting test error?

Firstly, let's define some parameters

```
nfolds <- 30
grid <- 10^seq(3, -1, length = 100)
thresh <- 1e-10
```

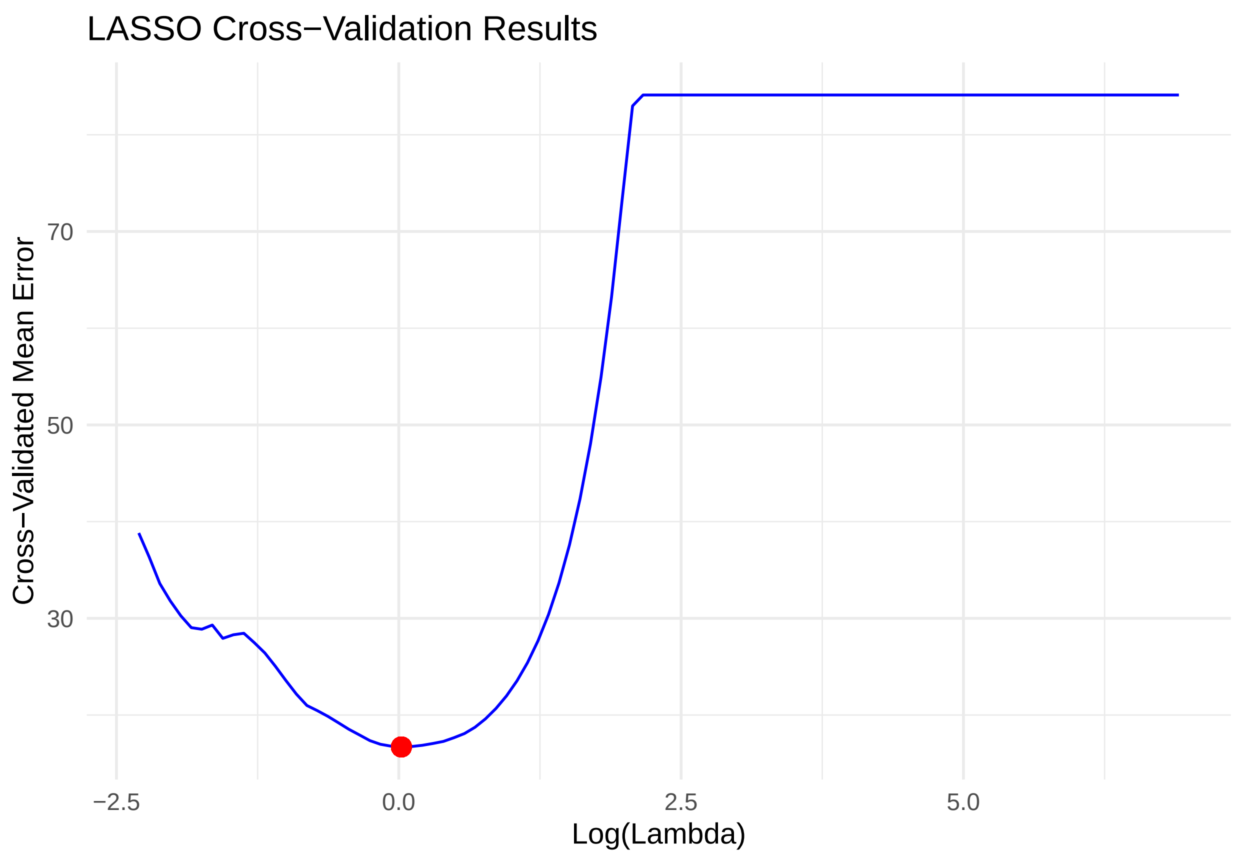
```
set.seed(123)
train_X <- as.matrix(X[train, ])
train_Y <- as.vector(Y[train])

lasso_model <- cv.glmnet(train_X, train_Y, alpha = 1, nfolds = nfolds,
                         lambda = grid, thresh = thresh)
```

```
## Warning: Option grouped=FALSE enforced in cv.glmnet, since < 3 observations per
## fold
```

```
#and graph the data for use
lambda_values <- lasso_model$lambda
cv_means <- lasso_model$cvm  # Mean cross-validated error for each lambda
```

```r
optimal_lambda <- lasso_model$lambda.min
optimal_error <- min(cv_means)

lasso_results <- data.frame(lambda = lambda_values, cv_error = cv_means)
ggplot(lasso_results, aes(x = log(lambda), y = cv_error)) +
  geom_line(color = "blue") +  # Line for cross-validated error
  geom_point(aes(x = log(optimal_lambda), y = optimal_error), color = "red", size = 3) +  # Point for o
  labs(title = "LASSO Cross-Validation Results",
       x = "Log(Lambda)",
       y = "Cross-Validated Mean Error") +
  theme_minimal() +
  theme(legend.position = "none")
```

```
## Warning in geom_point(aes(x = log(optimal_lambda), y = optimal_error), color = "red", : All aestheti
## i Please consider using `annotate()` or provide this layer with data containing
##   a single row.
```



```r
optimal_lambda
```

```
## [1] 1.023531
```

State your final model for the UPDRS. How many features have been selected? What conclusions can you draw?

We extract the optimal model below (best lambda)

```
selected_features <- which(lasso_model$glmnet.fit$beta[, which.min(lasso_model$cvm)] != 0)
selected_features <- rownames(lasso_model$glmnet.fit$beta)[selected_features]
selected_features
```

```
## [1] "X9"  "X82" "X83" "X97"
```

```
beta_coefficients <- coef(lasso_model, s = optimal_lambda)
beta_coefficients_matrix <- as.matrix(beta_coefficients)
non_zero_coefficients <- beta_coefficients_matrix[beta_coefficients_matrix != 0, ]
print(non_zero_coefficients)
```

```
## (Intercept)          X9         X82         X83         X97
## 26.28565266  0.19461853  0.05663504  0.42904877  7.90774073
```

```
predictions_train <- predict(lasso_model, s = optimal_lambda, newx = train_X)

train_MSE <- mean((train_Y - predictions_train)^2)

test_X <- as.matrix(X[test, ])
test_Y <- as.vector(Y[test])
predictions_test <- predict(lasso_model, s = optimal_lambda, newx = test_X)

test_MSE <- mean((test_Y - predictions_test)^2)

SStot <- sum((train_Y - mean(train_Y))^2)
SSres <- sum((train_Y - predictions_train)^2)
r_squared <- 1 - (SSres / SStot)

# Display the results
cat("Training MSE (Model):", train_MSE, "\n")
```

```
## Training MSE (Model): 12.49523
```

```
cat("Test MSE (Model):", test_MSE, "\n")
```

```
## Test MSE (Model): 20.40783
```

```
cat("R-squared (Model):", r_squared, "\n")
```

```
## R-squared (Model): 0.8410249
```

We see above the features that inform the final model (post lasso)

Our model has 4 features.

$$\text{UPDRS} = \beta_0 + \beta_1 X_9 + \beta_2 X_{82} + \beta_3 X_{83} + \beta_4 X_{97}$$

Where:

- $\beta_0 = 26.2857$ (Intercept)

- $\beta_1 = 0.1946$ (Feature X_9)
- $\beta_2 = 0.0566$ (Feature X_{82})
- $\beta_3 = 0.4290$ (Feature X_{83})
- $\beta_4 = 7.9077$ (Feature X_{97})

The LASSO feature selection has indicaed that X9,X82,X83 and X97 are the most significant contributors to the UPDRS variable (post regularization), this implies that the other columns are not significant contributors to the USPRS rating.Specifically; the relationship between these is linear (or, rather comes across as linear enough for LASSO). We should also remember that LASSO isn't exhaustive and that some better combination of features better explains USPDR

Repeat your analysis with a different random split of the training and test sets. Have the same features been selected in your final model? Again, state your final model for the UPDRS. (Hint: summarising your results in a table allows for easy comparison)

```
#Let's redefine the seed
set.seed(324234)

#get a new random train & test...
train <- sample(1:nrow(X), 30)
test <- setdiff(1:nrow(X), train)
train
```

```
##  [1] 29  5 19 34 26 24 41 40 13  3 42 22  8 23 10 35  9  1 32  4 37 11 18 25 33
## [26] 21 30 38 12 16
```

```
test
```

```
##  [1]  2  6  7 14 15 17 20 27 28 31 36 39
```

```
# Fit a linear model using the training data
linear_model_second <- lm(Y[train] ~ X[train, ])
fitted_values_second <- predict(linear_model_second, newdata = data.frame(X = X[train, ]))

# Compare actuals and predictions to validate linear separability
actuals_second <- as.vector((Y[train]))
preds_second <- as.vector(fitted_values_second)
fitting_error_second <- all.equal(actuals_second, preds_second)
fitting_error_second
```

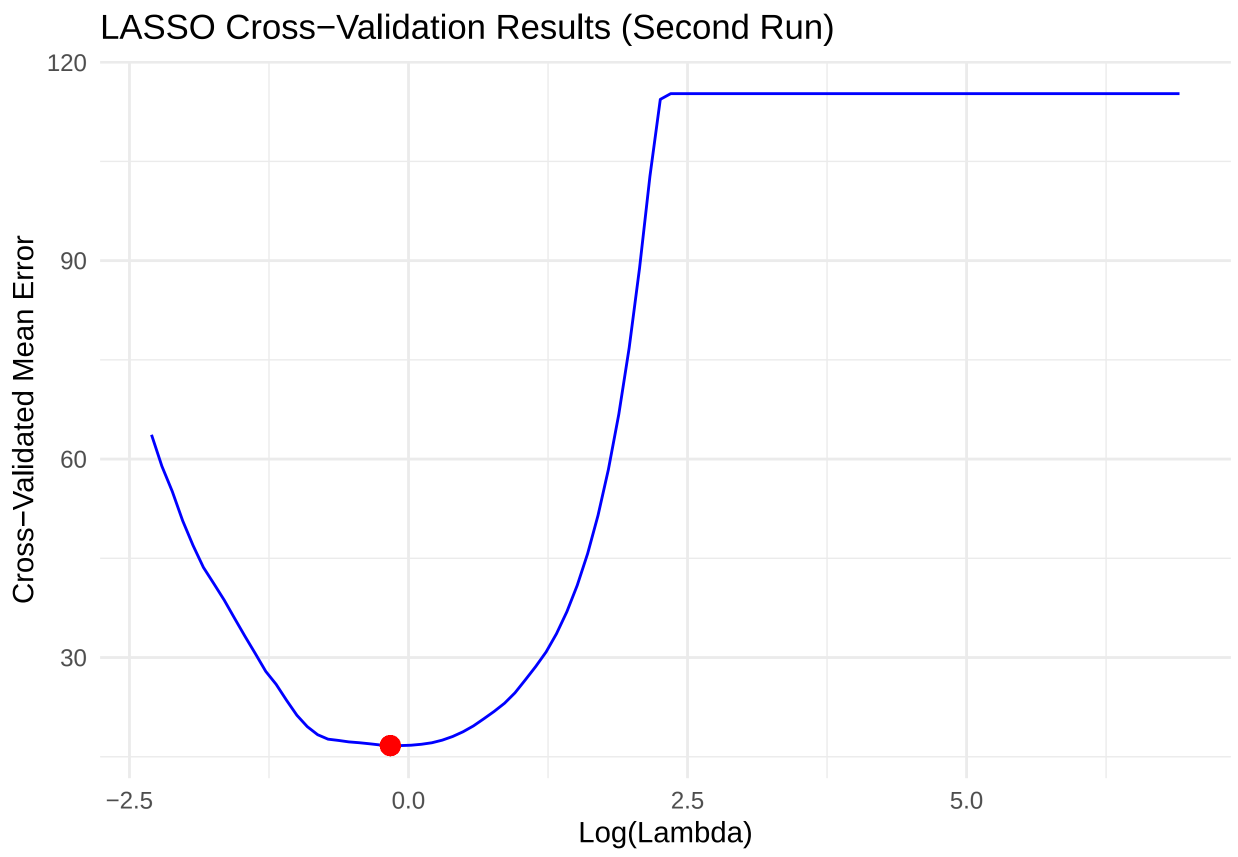
```
## [1] TRUE
```

```
# Prepare the training data for LASSO
train_X_second <- as.matrix(X[train, ])
train_Y_second <- as.vector(Y[train])

# Apply LASSO with cross-validation
lasso_model_second <- cv.glmnet(train_X_second, train_Y_second, alpha = 1, nfolds = nfolds,
                                lambda = grid, thresh = thresh)
```

```
## Warning: Option grouped=FALSE enforced in cv.glmnet, since < 3 observations per
## fold
```

```r
# Plot the LASSO cross-validation results
lambda_values_second <- lasso_model_second$lambda
cv_means_second <- lasso_model_second$cvm  # Mean cross-validated error for each lambda
optimal_lambda_second <- lasso_model_second$lambda.min
optimal_error_second <- min(cv_means_second)

lasso_results_second <- data.frame(lambda = lambda_values_second, cv_error = cv_means_second)
ggplot(lasso_results_second, aes(x = log(lambda), y = cv_error)) +
  geom_line(color = "blue") +  # Line for cross-validated error
  geom_point(aes(x = log(optimal_lambda_second), y = optimal_error_second), color = "red", size = 3) +
  labs(title = "LASSO Cross-Validation Results (Second Run)",
       x = "Log(Lambda)",
       y = "Cross-Validated Mean Error") +
  theme_minimal() +
  theme(legend.position = "none")
```

```
## Warning in geom_point(aes(x = log(optimal_lambda_second), y = optimal_error_second), : All aesthetic
## i Please consider using ‘annotate()‘ or provide this layer with data containing
##   a single row.
```



```r
# Display optimal lambda
optimal_lambda_second
```

```
## [1] 0.8497534
```

And we'll record the Rˆ2 and MSE again (train and test)

```
selected_features_second <- which(lasso_model_second$glmnet.fit$beta[, which.min(lasso_model_second$cvm)
selected_features_second <- rownames(lasso_model_second$glmnet.fit$beta)[selected_features_second]
selected_features_second
```

```
## [1] "X5"  "X11" "X97"
```

```
beta_coefficients_second <- coef(lasso_model_second, s = optimal_lambda_second)
beta_coefficients_matrix_second <- as.matrix(beta_coefficients_second)
non_zero_coefficients_second <- beta_coefficients_matrix_second[beta_coefficients_matrix_second != 0, ]
print(non_zero_coefficients_second)
```

```
## (Intercept)          X5         X11         X97
##   28.041975    3.470596    1.854626    8.714117
```

```
predictions_train_second <- predict(lasso_model_second, s = optimal_lambda_second, newx = train_X_second


train_MSE_second <- mean((train_Y_second - predictions_train_second)^2)






test_X_second <- as.matrix(X[test, ])
test_Y_second <- as.vector(Y[test])
predictions_test_second <- predict(lasso_model_second, s = optimal_lambda_second, newx = test_X_second)


test_MSE_second <- mean((test_Y_second - predictions_test_second)^2)


SStot_second <- sum((train_Y_second - mean(train_Y_second))^2)
SSres_second <- sum((train_Y_second - predictions_train_second)^2)
r_squared_second <- 1 - (SSres_second / SStot_second)

# Display the results
cat("Training MSE (Second Model):", train_MSE_second, "\n")
```

```
## Training MSE (Second Model): 10.49725
```

```
cat("Test MSE (Second Model):", test_MSE_second, "\n")
```

```
## Test MSE (Second Model): 105.3857
```

```
cat("R-squared (Second Model):", r_squared_second, "\n")
```

```
## R-squared (Second Model): 0.9025212
```

$$\text{UPDRS} = \beta_0 + \beta_1 X_5 + \beta_2 X_{11} + \beta_3 X_{97}$$

Where:

- $\beta_0 = 28.04198$ (Intercept)
- $\beta_1 = 3.47060$ (Feature X_5)
- $\beta_2 = 1.85463$ (Feature X_{11})
- $\beta_3 = 8.71412$ (Feature X_{97})

Let's compare the two models

(Note Knittr table doesn't seem to support new line chars so there are truncated variables, however the core data is here)

```r
results_table <- data.frame(
  Random_Split = c("First Split", "Second Split"),
  Selected_Features = I(list(selected_features, selected_features_second)),
  Beta_Coefficients = I(list(non_zero_coefficients, non_zero_coefficients_second)),
  Test_MSE = c(test_MSE, test_MSE_second),
  Training_MSE = c(train_MSE, train_MSE_second),
  R_squared = c(r_squared, r_squared_second),
  Number_of_Features = c(length(selected_features), length(selected_features_second))  # New column
)



# Pretty table
kable(results_table,
      format = "markdown",
      col.names = c("Random Split", "Selected Features", "Beta Coefficients", "Test MSE", "Training MSE
      row.names = FALSE)
```

Random Split	Selected Features	Beta Coefficients	Test MSE	Training MSE	R-squared	Number of Features
First Split	X9, X82,. . . .	26.28565. . . .	20.40783	12.49523	0.8410249	4
Second Split	X5, X11, X97	28.04197. . . .	105.38567	10.49725	0.9025212	3

The Second Split model is superior due to its lower Test MSE (10.50) and higher R-squared value (0.90), indicating better performance and explanatory power compared to the First Split model. The simplicity of the second model (3 vs 4 explanatories) doesn't sufficiently justify the TEST MSE reduction. It's also worth noting that R-squared is much greater in split 2; split 2 better explains the variance in the data. Ultimately the better fit for the data here is liklely modely 2, as we contextualize it with parkisons rates; a lower test MSE is critical in ensuring more accurate predictions for new / changed patients. Therefore, we retread the mdoel as above as the final model:

$$\text{UPDRS} = \beta_0 + \beta_1 X_5 + \beta_2 X_{11} + \beta_3 X_{97}$$

Where:

- $\beta_0 = 28.04198$ (Intercept)

- $\beta_1 = 3.47060$ (Feature X_5)
- $\beta_2 = 1.85463$ (Feature X_{11})
- $\beta_3 = 8.71412$ (Feature X_{97})

Question 3

We wish to predict final year scores given the scores of previous years.

Note; the scale for each year is different; however bounded on each year.

Firstly, set up the data, load in and split etc.

```
# Clear workspace and set seed
rm(list = ls())
set.seed(123)

# Load the data
studydata = read.csv('engstudres.csv')
X <- model.matrix(Final ~., studydata)[,-1]
X <- scale(X)
head(X)
```

```
##          Yr1        Yr2        Yr3
## 1 -1.6849036 -1.4114890 -1.5853685
## 2  1.1963455  0.3862056  0.6231218
## 3 -1.6849036 -1.8775580  1.2797000
## 4 -1.7519094 -0.5792230  0.2948327
## 5 -0.4117935 -0.5126417 -0.8989459
## 6  2.2684381  0.9854371  0.4142106
```

```
y <- studydata$Final
head(y)
```

```
## [1]  93 207 175 125 114 159
```

```
train <- sample(1:nrow(X), 0.8 * nrow(X))  # 80% for training
test <- -train
```

For our Elastic net regression let's define the alpha and lambda parameters for a grid search as follows:

```
grid <- expand.grid(.alpha  = seq(0, 1, by = 0.1),
                    .lambda = 10^seq(5, -2, length = 100))
head(grid, 12)
```

```
##    .alpha   .lambda
## 1     0.0 100000.00
## 2     0.1 100000.00
## 3     0.2 100000.00
## 4     0.3 100000.00
## 5     0.4 100000.00
## 6     0.5 100000.00
## 7     0.6 100000.00
## 8     0.7 100000.00
## 9     0.8 100000.00
## 10    0.9 100000.00
## 11    1.0 100000.00
## 12    0.0  84975.34
```

```
dim(grid)
```

```
## [1] 1100    2
```

Let's train some models

```
control <- caret::trainControl(method = "cv", number = 10)
#10 fold defined here

enet.train <- caret::train(x         = X[train,],
                           y         = y[train],
                           method    = "glmnet",
                           trControl = control,
                           tuneGrid  = grid)
```

```
## Loading required package: lattice
```

```
## Warning in nominalTrainWorkflow(x = x, y = y, wts = weights, info = trainInfo,
## : There were missing values in resampled performance measures.
```

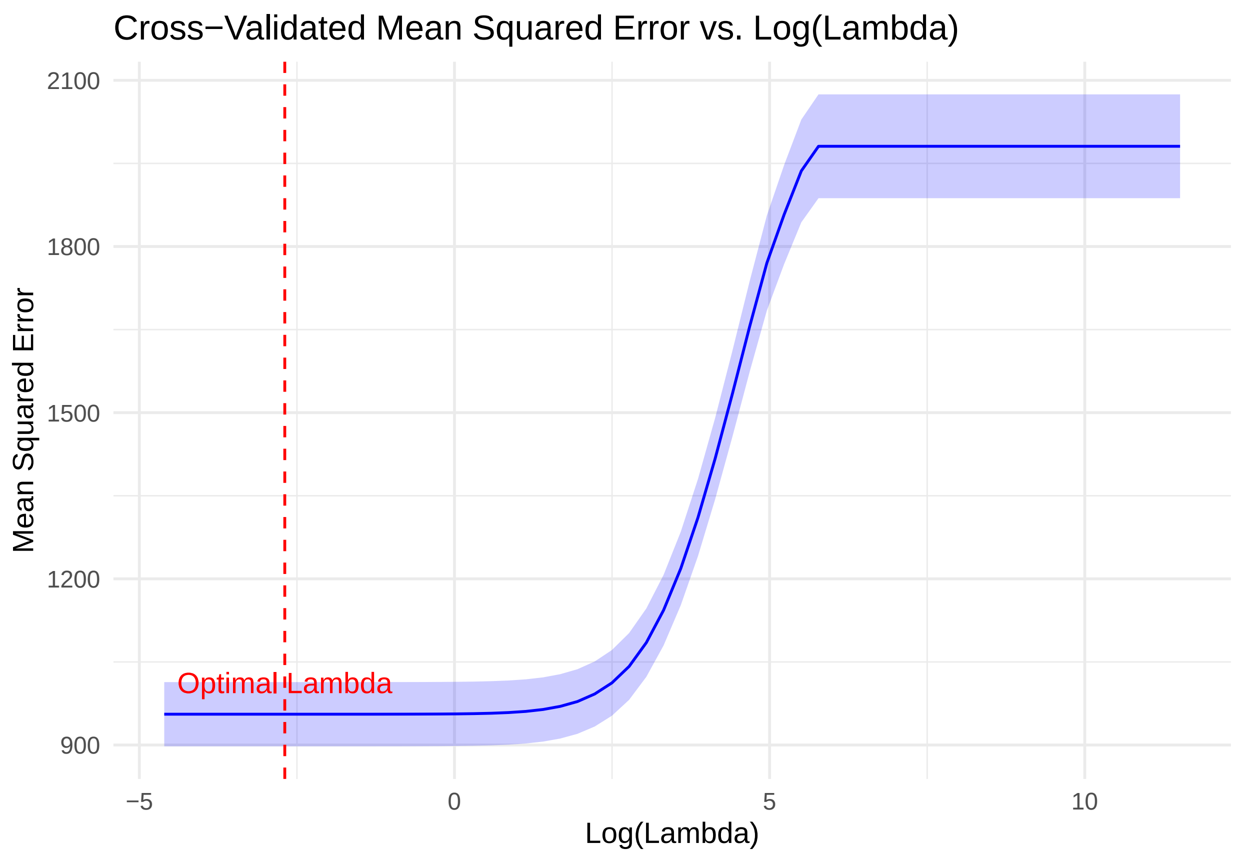
```
best_alpha <- enet.train$bestTune$alpha
best_lambda <- enet.train$bestTune$lambda

print(paste("Best alpha:", best_alpha))
```

```
## [1] "Best alpha: 0.1"
```

```
print(paste("Best lambda:", best_lambda))
```

```
## [1] "Best lambda: 0.259502421139974"
```

```
.lambda <- 10^seq(5, -2, length = 60)




enet.cv <- cv.glmnet(X[train,],
                     y[train],
                     alpha  = enet.train$bestTune$alpha,    # 0.9 ElasticNet
                     lambda = .lambda,
                     nfolds = 10,
                      thresh = 1e-12)




cv_results <- data.frame(
  log_lambda = log(enet.cv$lambda),
  mse = enet.cv$cvm,
  std_err = enet.cv$cvsd
)


ggplot(cv_results, aes(x = log_lambda, y = mse)) +
  geom_line(color = "blue") +
  geom_ribbon(aes(ymin = mse - std_err, ymax = mse + std_err),
              alpha = 0.2, fill = "blue") +  # Confidence interval
  labs(title = "Cross-Validated Mean Squared Error vs. Log(Lambda)",
       x = "Log(Lambda)",
       y = "Mean Squared Error") +
  theme_minimal() +  # A clean theme
  geom_vline(xintercept = log(enet.cv$lambda.min), color = "red", linetype = "dashed") +  # Line for op
  annotate("text", x = log(enet.cv$lambda.min), y = min(cv_results$mse),
           label = "Optimal Lambda", vjust = -1, color = "red")
```



Now we will make some predictions on the test set:

```r
coef_min <- as.vector(coef(enet.cv, s = "lambda.min"))
coef_1se <- as.vector(coef(enet.cv, s = "lambda.1se"))

# Exclude intercept from coefficients
coef_min <- coef_min # Exclude the intercept
coef_1se <- coef_1se # Exclude the intercept

# Extract variable names
variable_names <- colnames(X)

# Compute MSE and RMSE for both models
mse_min <- mean((y[test] - predict(enet.cv, s = "lambda.min", newx = X[test,]))^2)
rmse_min <- sqrt(mse_min)

mse_1se <- mean((y[test] - predict(enet.cv, s = "lambda.1se", newx = X[test,]))^2)
rmse_1se <- sqrt(mse_1se)

mse_1se
```

```
## [1] 869.7771
```

```r
mse_min
```

```
## [1] 828.0422
```

```r
rmse_1se
```

```
## [1] 29.49198
```

```r
rmse_min
```

```
## [1] 28.77572
```

```r
coef_min
```

```
## [1] 148.558598   1.311519  13.309871  29.118097
```

```r
coef_1se
```

```
## [1] 148.69015   0.00000   9.83192  22.44985
```

And we can see the results from above in a table (manually because of challenges with R and potentially sparse matrics)

```r
# These are hardcoded becase R is a challenge sometimes!
model_results <- data.frame(
  Type = c("1se", "min"),
  MSE = c(869.7771, 828.0422),
  RMSE = c(29.49198, 28.77572),
  Intercept = c(148.558598, 148.69015),
  Yr1 = c(1.311519, 0.00000),
  Yr2 = c(13.309871, 9.83192),
  Yr3 = c(29.118097, 22.44985)
)

# Create the table
kable(model_results, format = "markdown", digits = 4, caption = "Model Results Comparison")
```

Table 4: Model Results Comparison

Type	MSE	RMSE	Intercept	Yr1	Yr2	Yr3
1se	869.7771	29.4920	148.5586	1.3115	13.3099	29.1181
min	828.0422	28.7757	148.6901	0.0000	9.8319	22.4499

Given the table above:

The min model has a lower Mean Squared Error (MSE = 828.0422) and Root Mean Squared Error (RMSE = 28.7757) compared to the 1se model (MSE = 869.7771, RMSE = 29.4920). This indicates that the min model provides better overall predictive accuracy regarding students' final-year examination scores based on their previous performance. Notably, the coefficient for Year 1 (Yr1) is zero, suggesting that first year performance is not indicative of final year outcomes (and intuitively this does make sense, noting we scaled the entire dataset which may reduce the influecne of Yr1 as it is bound in 0-100). Both of our interceptes are *about the same* indicating that the basline is ocnsistnet on both models; i.e. all variations are products of the coefficients. As in *min* the Yr3 coefficient is largest, it is the most indicative of final year performance. Year 2, similarly does contribute, but is much lower. In a real world context these can correlate with the relative ease of second year papers, students 'finding their groove' etc.

This is all considered while still noting that our mdoel isn't a great predictor at all. The MSE (and therefore RMSE) is far too high to be useful, i.e. the RMSE puts almost a 30% out-by value on either model.

'Choosing the min model is justified due to its superior predictive accuracy and clearer interpretability of the influence of yearly performances on final examination outcomes.'

Each of the models is as follows:

The regression equations for the models are as follows:

1. **1se Model**:

$$\text{Final Score}_{1se} = 148.5586 + 1.3115 \cdot \text{Yr1} + 13.3099 \cdot \text{Yr2} + 29.1181 \cdot \text{Yr3}$$

2. **Min Model**:

$$\text{Final Score}_{min} = 148.6901 + 0.0000 \cdot \text{Yr1} + 9.8319 \cdot \text{Yr2} + 22.4499 \cdot \text{Yr3}$$

The model of choice is the 'min' model. I.e the model of our pool which overall reduced cross validation error. And indeed it holds that 1se would not be an optimal choice as we do not expect sparsity in our dataset (missing records is the exception not the norm by a huge margin). We won't worry about over fitting as the test MSE is (still very poor); hweover we aren't likely ot be overfitting due ot the large N.