STAT-448

Assignment 2

Rujal Shrestha - 29954403

08 September, 2025

Question 1

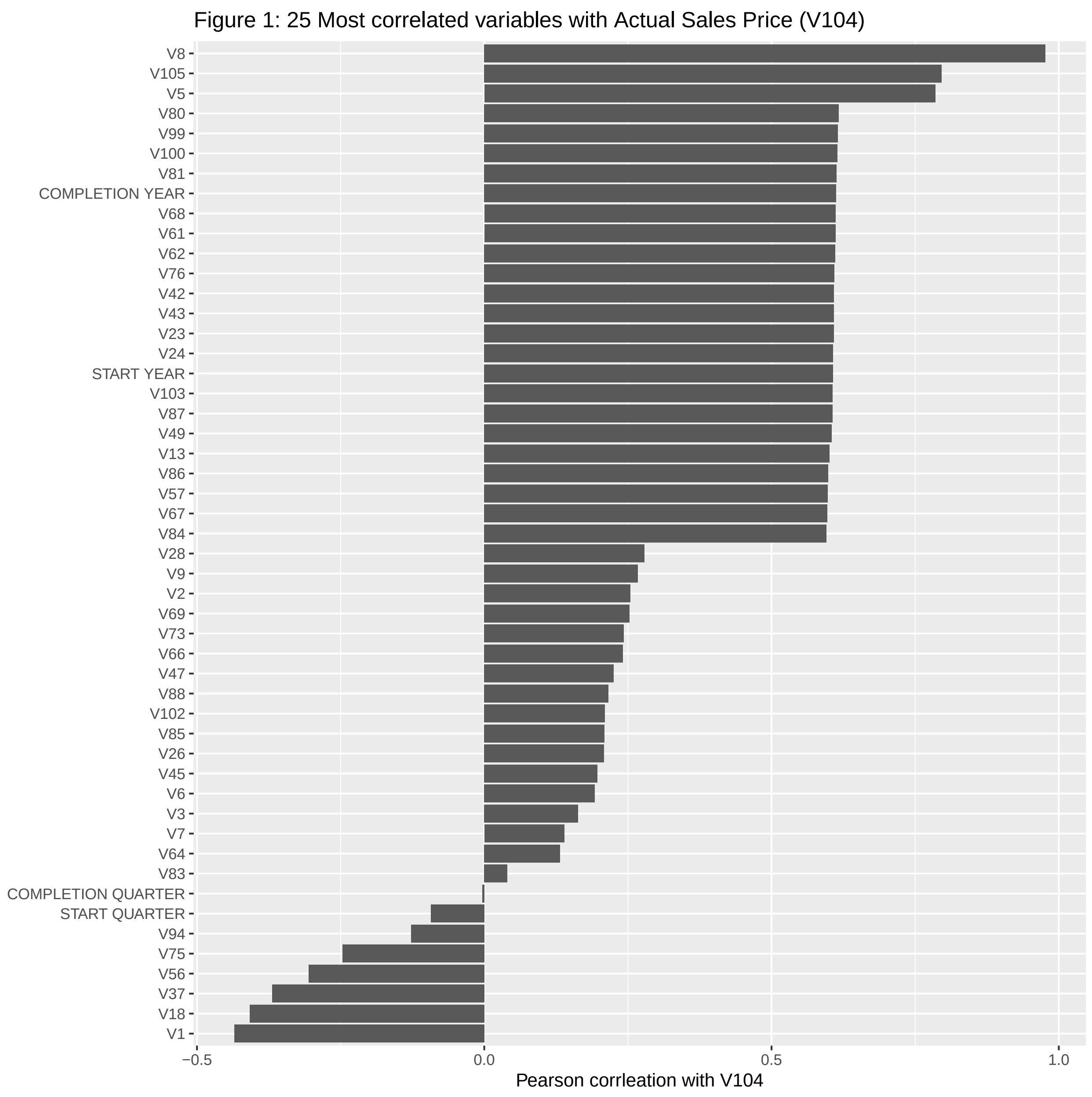
Question 1(a)

```r
# creating new environment and loading data into it
# this avoids polluting the global environment when loading Rdata
myenv <- new.env()
load("./dataset/Residen.RData", envir = myenv)
```

```r
df <- myenv[["Residen"]]
```

```r
correlation <- cor(df)
most_correlated <- sort(correlation[, "V104"], decreasing = TRUE)

# only select the most extreme 25 correlated variables on both ends
top_k <- 25

extreme_correlated <- tibble(
  var = names(most_correlated),
  r = as.numeric(most_correlated)
) |>
  filter(var != "V104") |>
  slice(c(1:top_k, (n() - top_k + 1):n())) |>
  mutate(var = fct_reorder(var, r))
```

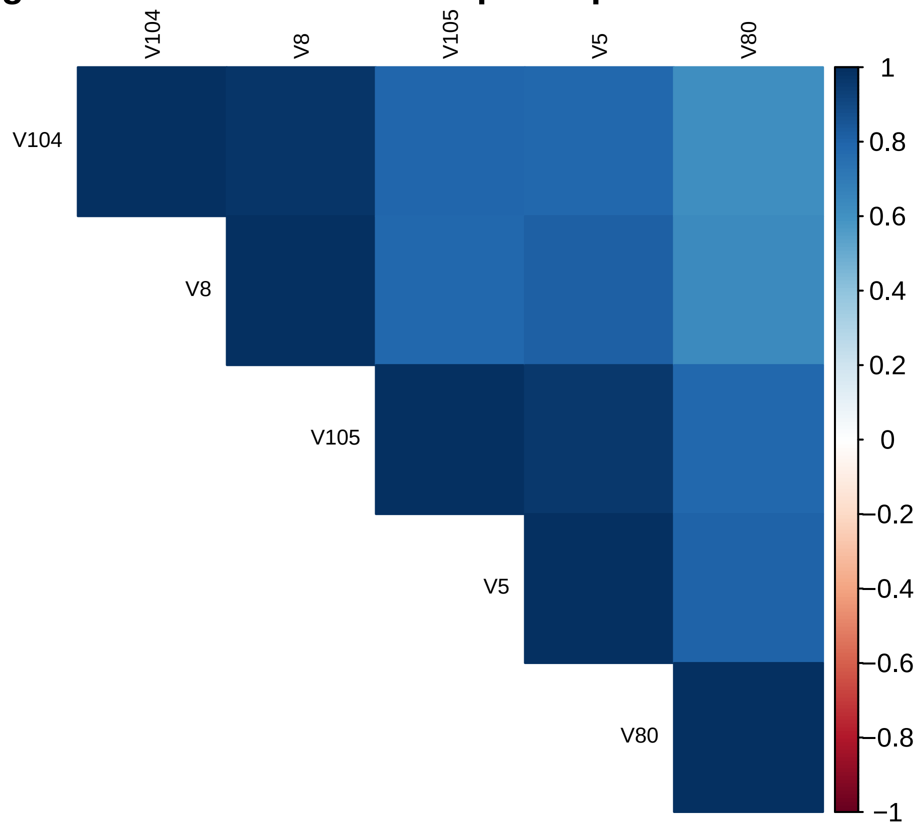
```r
extreme_correlated |>
  ggplot(aes(r, var)) +
  geom_col() +
  labs(
    title = "Figure 1: 25 Most correlated variables with Actual Sales Price (V104)",
    x = "Pearson corrleation with V104",
    y = NULL
  )
```



The *Figure 1* displays the 50 most extreme correlating variables with the actual sales price. We can see that variables V-8(prior price of unit), V105 (Construction output), V-5 (estimated constrution cost) are some variables that correlated positively, which indicates that increase in these values have a direct positive impact on the actual sales prices. Furthermore, V1, V18, V37 correlated negatively, that is, lowers the actual sales price.

```r
top_vars <- names(head(most_correlated, 5))

corrplot(cor(df[, top_vars]),
  method = "color", type = "upper",
  tl.cex = 0.7, tl.col = "black", mar = c(0, 0, 1, 0),
  title = "Figure 2: Correlation heatmap of top 5 variables vs V104"
)
```

Figure 2: Correlation heatmap of top 5 variables vs V104



The *Figure 2* represents a correlation heatmap of the top 5 variables positively correlated with V104. We can see a very strong correlation with V8 and moderately good correlation with V105, V5 and v80.

Question 1(b)

```
model <- lm(data = df, as.formula("V104 ~ . -V105"))

summary <- summary(model)

summary
```

```
##
## Call:
## lm(formula = as.formula("V104 ~ . -V105"), data = df)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -901.15  -52.29   -1.50   45.71  645.31
##
## Coefficients: (32 not defined because of singularities)
##                         Estimate Std. Error t value Pr(>|t|)
## (Intercept)            1.004e+05  1.999e+05   0.502 0.615781
## `START YEAR`          -1.614e+03  3.388e+03  -0.476 0.634175
## `START QUARTER`       -4.928e+02  2.457e+03  -0.201 0.841139
## `COMPLETION YEAR`     1.521e+02  1.689e+01   9.007  < 2e-16 ***
## `COMPLETION QUARTER`  5.984e+01  8.881e+00   6.738 8.36e-11 ***
## V1                   -4.779e+00  2.225e+00  -2.148 0.032529 *
## V2                    6.630e-02  2.195e-02   3.020 0.002746 **
## V3                   -2.331e-01  6.451e-02  -3.612 0.000356 ***
## V4                    6.442e-03  3.570e-02   0.180 0.856905
## V5                   -6.548e-01  3.342e-01  -1.960 0.050975 .
## V6                    8.764e-02  6.322e-02   1.386 0.166727
## V7                           NA         NA      NA       NA
## V8                    1.203e+00  1.681e-02  71.579  < 2e-16 ***
## V9                    2.016e-01  1.103e+00   0.183 0.855159
## V10                   1.373e+02  9.675e+02   0.142 0.887245
## V11                   1.048e+01  7.426e+01   0.141 0.887830
## V12                  -1.626e+02  6.837e+02  -0.238 0.812200
## V13                   1.329e-04  1.125e-01   0.001 0.999058
## V14                   1.799e-01  4.339e-01   0.415 0.678643
## V15                  -2.265e+01  5.008e+01  -0.452 0.651433
## V16                   8.696e-01  2.753e+00   0.316 0.752315
## V17                  -4.247e-03  1.860e-01  -0.023 0.981801
## V18                   2.328e+02  1.235e+03   0.188 0.850672
## V19                  -2.007e-01  4.647e+00  -0.043 0.965587
## V20                  -3.910e-01  1.187e+00  -0.329 0.742147
## V21                   6.422e-02  3.565e-01   0.180 0.857174
## V22                   3.682e-03  4.524e-01   0.008 0.993511
## V23                   6.413e+00  6.128e+02   0.010 0.991658
## V24                   4.270e+01  2.347e+02   0.182 0.855747
## V25                   5.267e-02  1.033e-01   0.510 0.610369
## V26                  -4.675e-04  4.268e-01  -0.001 0.999127
## V27                   9.455e-04  9.654e-03   0.098 0.922046
## V28                   5.915e-02  1.795e-01   0.330 0.741930
## V29                  -1.144e+02  8.626e+02  -0.133 0.894615
## V30                  -7.247e+00  1.452e+02  -0.050 0.960240
## V31                  -9.085e+01  3.036e+02  -0.299 0.764953
## V32                  -1.780e-03  8.287e-02  -0.021 0.982879
## V33                  -5.666e-02  3.707e-01  -0.153 0.878630
## V34                  -9.607e+00  1.077e+02  -0.089 0.929008
## V35                   9.323e-01  1.976e+00   0.472 0.637449
## V36                   1.490e-02  8.094e-02   0.184 0.854034
## V37                   7.421e+01  6.343e+02   0.117 0.906942
## V38                  -4.162e-01  6.246e+00  -0.067 0.946913
## V39                   6.458e-01  2.013e+00   0.321 0.748563
## V40                  -8.375e-02  1.877e-01  -0.446 0.655784
## V41                   1.615e-01  5.671e-01   0.285 0.776052
## V42                   1.890e+02  6.380e+02   0.296 0.767239
## V43                  -1.891e+02  8.982e+02  -0.211 0.833373
## V44                  -1.202e-02  5.531e-01  -0.022 0.982673
## V45                  -1.184e-02  3.554e-01  -0.033 0.973441
## V46                  -1.038e-03  8.216e-03  -0.126 0.899575
## V47                   1.040e-01  2.590e-01   0.402 0.688186
## V48                   1.119e+02  1.334e+03   0.084 0.933224
## V49                  -3.006e+01  4.980e+01  -0.604 0.546571
## V50                  -1.746e+02  4.440e+02  -0.393 0.694395
## V51                   6.526e-03  1.409e-01   0.046 0.963102
## V52                  -7.985e-02  4.933e-01  -0.162 0.871519
## V53                    6.439e+00  1.296e+02   0.050 0.960422
## V54                    2.405e+00  4.657e+00   0.516 0.605918
## V55                   -1.698e-02  1.631e-01  -0.104 0.917133
## V56                    3.089e+01  2.974e+02   0.104 0.917333
## V57                   -2.722e-01  2.233e+00  -0.122 0.903072
## V58                    3.539e-01  3.079e+00   0.115 0.908564
## V59                   -9.660e-02  5.540e-01  -0.174 0.861684
## V60                   -2.140e-01  1.307e+00  -0.164 0.870070
## V61                    4.198e+01  1.636e+02   0.257 0.797647
## V62                    2.040e+02  1.364e+03   0.150 0.881170
## V63                   -1.273e-01  8.479e-01  -0.150 0.880716
## V64                   -2.178e-02  3.772e-01  -0.058 0.953994
## V65                   -9.490e-04  8.072e-03  -0.118 0.906483
## V66                    6.260e-02  6.990e-01   0.090 0.928700
## V67                   -2.018e+02  7.758e+02  -0.260 0.794924
## V68                    6.947e+01  7.047e+01   0.986 0.324982
## V69                    1.248e+02  1.397e+02   0.894 0.372099
## V70                   -9.328e-03  4.437e-02  -0.210 0.833629
## V71                   -1.346e-01  4.248e-01  -0.317 0.751583
## V72                   -1.492e+01  2.247e+02  -0.066 0.947118
## 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
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
```

```
## Residual standard error: 148.6 on 296 degrees of freedom
## Multiple R-squared:  0.9879, Adjusted R-squared:  0.9848
## F-statistic: 321.9 on 75 and 296 DF,  p-value: < 2.2e-16
```

The following points can be extracted from the summary:

1. The median (-1.50) is close to zero, which indicates that there is no bias as seen from the distribution of the residuals. The range is shows large deviations, with majority residing between 50 units on either side
2. The most significant variables are completion year, completion quarter, V8, etc and 32 variables are shows as NA because of singularities.
3. The model explains 99% of the variation in the prices, which is an indication of good or overfitted model. The adjusted R-squared 0.885 indicates a very explanation of variation after penalising predictors. Overall, the regression model fits the data extremely well with 32 coefficients dropped due to coliniearity. It seems that we can further progress into regularization methods to further improve the model as it seems the model is overfitted to the data.

Question 1(c)

```r
set.seed(42)

n <- nrow(df)

training_indexes <- sample(seq_len(n), size = floor(0.8 * n))

train <- df[training_indexes, ]
test <- df[-training_indexes, ]

y_test <- test$V104
```

Training and testing backwards selection

```r
time_backwards <- system.time({
  backwards_model <- stepAIC(
    lm(as.formula("V104 ~ . - V105"), data = train),
    direction = "backward",
    trace = FALSE
  )
})

backwards_predictions <- predict(backwards_model, newdata = test)
backwards_mse <- mse(y_test, backwards_predictions)
```

Training and testing stepwise selection

```r
time_stepwise <- system.time({
  stepwise_model <- stepAIC(
    lm(V104 ~ 1, data = train),
    scope = list(lower = ~1, upper = ~ . - V105),
    direction = "both",
    trace = FALSE
  )
})

stepwise_predictions <- predict(stepwise_model, newdata = test)
stepwise_mse <- mse(y_test, stepwise_predictions)
```

Now, we compare the above quantities in a new table

```
output_table <- tibble(
  Method = c("Backward", "Stepwise"),
  Num_predictors = c(
    length(coef(backwards_model)) - 1,
    length(coef(stepwise_model)) - 1
  ),
  Mse = c(backwards_mse, stepwise_mse),
  Time = c(time_backwards[3], time_stepwise[3])
)

kable(
  output_table,
  digits = 3,
  col.names = c("Method", "Number of Predictors", "MSE", "Time"),
  caption = "Comparison of backwards vs stepwise selection"
)
```

Table 1: Comparison of backwards vs stepwise selection

Method	Number of Predictors	MSE	Time
Backward	38	29233.43	5.704
Stepwise	0	2295678.16	0.000

The above *Table 1* compares outputs, computational time and mean square error for test set of the two selection methods. The backward selection method ended up with a more complex model with 38 predictors and a reasonably low error of around 29,233 all within the span of 5.1 seconds, which is a moderate amount.

The stepwise selection on the other hand ended up with an extremely high error with over 2,295,678. It took nearly 0 seconds to compute, but this is reasonable considering that this model did not add any other predictors other than the intercept.

Overall, the backward selection outperformed the stepwise selection and better explained the variation in actual sale prices. This might be due to the presence of strong multicollinearity.

Question 1(d)

```
# exclude the intercept column as it is automatically added by glmnet
x_train <- model.matrix(V104 ~ . - V105, data = train)[, -1]
y_train <- train$V104
```

```r
x_test <- model.matrix(V104 ~ . - V105, data = test)[, -1]

time_ridge <- system.time({
  model_ridge <- cv.glmnet(x_train, y_train, alpha = 0, nfolds = 10)
})

time_lasso <- system.time({
  model_lasso <- cv.glmnet(x_train, y_train, alpha = 1, nfolds = 10)
})

predictions_ridge <- predict(model_ridge, newx = x_test, s = "lambda.min")
predictions_lasso <- predict(model_lasso, newx = x_test, s = "lambda.min")

mse_ridge <- mse(y_test, predictions_ridge)
mse_lasso <- mse(y_test, predictions_lasso)

nz_ridge <- sum(coef(model_ridge, s = "lambda.min")[-1] != 0)
nz_lasso <- sum(coef(model_lasso, s = "lambda.min")[-1] != 0)

ridge_vs_lasso_table <- tibble(
  Model = c("Ridge", "LASSO"),
  lambda_min = c(model_ridge$lambda.min, model_lasso$lambda.min),
  mse = c(mse_ridge, mse_lasso),
  non_zero_predictors = c(nz_ridge, nz_lasso),
  time = c(time_ridge[3], time_lasso[3])
)

kable(ridge_vs_lasso_table, caption = "Ridge vs Lasso comparison")
```

Table 2: Ridge vs Lasso comparison

Model	lambda_min	mse	non_zero_predictors	time
Ridge	108.358549	114093.89	107	0.065
LASSO	2.334514	28362.21	29	0.031

From the *table 2*, it can be seen that the LASSO regression achieved a lower MSE on test data compared to ridge while reducing the number of predictors down to 29 only. LASSO regression not only provided better predictive accuracy, but also variable selection and in turn producing a more parsimonious model.

Question 1(e)

From the outputs obtained above, LASSO performed best with the lowest error, minimal number of coefficients, obtaining a parsimonious model by performing variable selection. It was able to lower the predictors down to 29, removing any noise and irrelevant variables. This also got rid of multicollinear variables which would've otherwise abrupted the predictability of the model as well as increase the computation time.

Among the selection methods, though backward selection method produced a somewhat acceptable model with low error and 38 predictors, this model heavily relised on hypothesis testing and AIC, which can be unstable when multicollinearity is present. The stepwise selection model performed the worse with no predictors selected, this also showcased the limitation of stepwise predictors, where the model couldn't improve further from the null model.

From the regularization method, ridge couldn't reduce the precictors as much compared to LASSO, so due to the existance of irrelevant variables, a lot of noise was introduced, which resulted in a very high mean squared error of 113,093.

Question 2

```r
df <- read.csv("dataset/parkinsons.csv")

set.seed(42)

training_indexes <- sample(seq_len(nrow(df)), size = 30)

train <- df[training_indexes, ]
test <- df[-training_indexes, ]

predictors <- paste0("X", 1:97)

x_train <- scale(as.matrix(train[, predictors]))

# Here, we are using the same mean and standard deviations from the training dataset to scale the test
train_center <- attr(x_train, "scaled:center")
train_scale <- attr(x_train, "scaled:scale")

x_test <- scale(
  as.matrix(test[, predictors]),
  center = train_center,
  scale = train_scale
)

y_train <- train$UPDRS
y_test <- test$UPDRS
```

The sampling was done with seed 42. Why 42? Cause it is the answer to life, the universe and everything.

Question 2(a)

```r
model <- lm(y_train ~ x_train)
y_predicted <- predict(model)
mse_trained <- mse(y_train, y_predicted)

coefs <- coef(model)
print(paste0("Non-zero coeffs: ", sum(coefs != 0 & !is.na(coefs))))
```

```
## [1] "Non-zero coeffs: 30"
```

```r
mse_trained
```

```
## [1] 6.808988e-22
```

The training MSE is essentially zero, which confirms that the model fit the training data perfectly. This is because there are very high number of predictors (97) compared to the number of observations (30), which is overparameterized with 30 coefficients and fits the training data perfectly. This is a classic case of overfitting where the model will have no bias with a very high variance. Due to which the model will not perform well with another set of data.

Question 2(b)

```
set.seed(42)
grid <- 10^seq(3, -1, length.out = 100)
model_lasso <- cv.glmnet(
  x_train,
  y_train,
  alpha = 1, # lasso regularization
  nfolds = 30,
  lambda = grid,
  thresh = 1e-10,
  standardize = FALSE
)
```

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

```
print(paste0("minimum lambda: ", model_lasso$lambda.min))
```

```
## [1] "minimum lambda: 0.642807311728432"
```

```
predicted_test <- predict(model_lasso, newx = x_test, s = "lambda.min")
test_mse <- mse(y_test, predicted_test)

print(paste0("test mse: ", test_mse))
```

```
## [1] "test mse: 21.123482464396"
```

```
coefs <- coef(model_lasso, s = "lambda.min")
```

```
nz_indexes <- which(coefs != 0)
nz_predictors <- setdiff(rownames(coefs)[nz_indexes], "(Intercept)")

print(paste0("Non-zero predictors: ", paste(nz_predictors, collapse = ", ")))
```

```
## [1] "Non-zero predictors: X9, X83, X86, X97"
```

The optimal value of lambda is 0.643 and the resulting test MSE was 21.12.

Question 2(c)

The LASSO model tuned with the specified settings was able to retain only 4 features (vs 30 using OLS) while shrinking other irrelevant ones to zero. This model is more parsimonious and generalized while yielding a very acceptable mse of 21.12 (compared to the ~0 MSE using OLS due to overfitting). Furthermore, it is also validated that the model was able to retain the X97 feature, which was stated to be informative. Overall, the LASSO improves predictive performance, enhances interpretability, a more sparse model with lesser variance and a more appropriate method for this setting.

Question 3

```r
getwd()
```

```
## [1] "/Users/rujalshrestha/Projects/mac-migration/stat-448/assignment-2"
```

```r
weather <- read.csv("dataset/Weather_Station_data_v1.csv", header = TRUE)

set.seed(42)
train_idx <- sample(seq_len(nrow(weather)), size = floor(0.8 * nrow(weather)))

train <- weather[train_idx, ]
test <- weather[-train_idx, ]

x_train <- model.matrix(MEAN_ANNUAL_RAINFALL ~ ., data = train)[, -1]
x_test <- model.matrix(MEAN_ANNUAL_RAINFALL ~ ., data = test)[, -1]

y_train <- train$MEAN_ANNUAL_RAINFALL
y_test <- test$MEAN_ANNUAL_RAINFALL
```

```r
alphas <- seq(0.1, 0.9, by = 0.1)

cv_list <- lapply(alphas, function(a) {
  set.seed(42)
  cv.glmnet(x_train, y_train, alpha = a, nfolds = 10)
})

cv_errors <- sapply(cv_list, function(cv) min(cv$cvm))

best_error_idx <- which.min(cv_errors)

best_alpha <- alphas[best_error_idx]
best_model <- cv_list[[best_error_idx]]

print(paste0("Best alpha: ", best_alpha))
```

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

```r
print(paste0("Lambda MSE: ", best_model$lambda.min))
```

```
## [1] "Lambda MSE: 4.7159811840903"
```

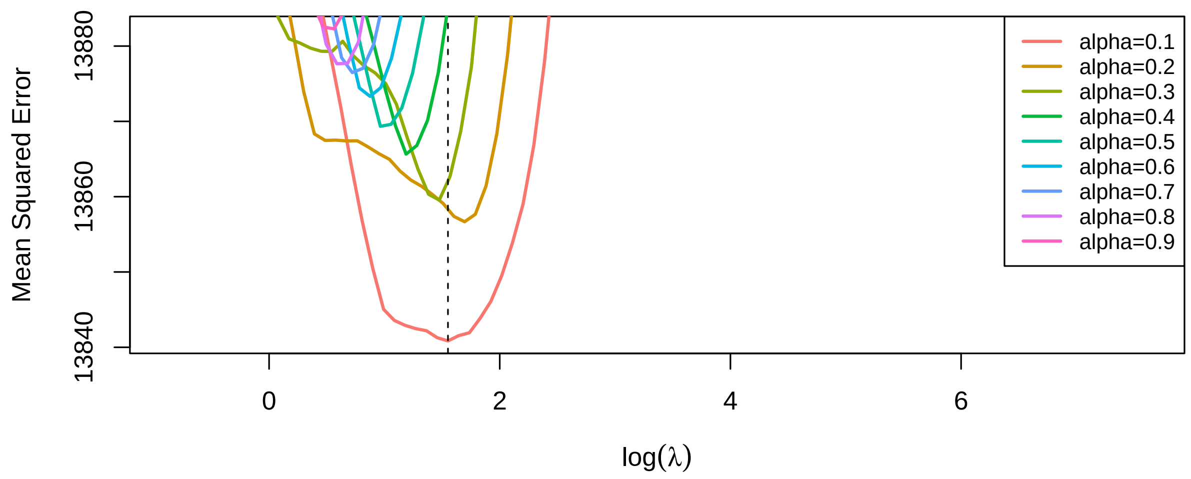
```r
print(paste0("Lambda 1SE: ", best_model$lambda.1se))
```

```
## [1] "Lambda 1SE: 33.2703149548259"
```

Here, we were able to obtain the best model with alpha of 0.1. The corresponding minimum mse was associated with lambda value of 4.72 and the generalized lambda 1SE of 33.27.

```r
colors <- scales::hue_pal()(length(alphas))

plot(
  NULL,
  xlab = expression(log(lambda)),
  ylab = "Mean Squared Error",
  xlim = range(log(cv_list[[1]]$lambda)),
  ylim = range(cv_errors)
)

for (i in seq_along(alphas)) {
  with(cv_list[[i]], lines(log(lambda), cvm, col = colors[i], lwd = 2))
}

legend(
  "topright",
  legend = paste0(
    "alpha=",
    alphas
  ),
  col = colors,
  lwd = 2,
  cex = .8
)

title("Figure 3: Elastic Net 10-fold across alpha and lambda")

abline(v = log(best_model$lambda.min), lty = 2)
```

Figure 3: Elastic Net 10−fold across alpha and lambda



The *figure 3* displays characteristics of all the values of alpha in the cross validation. The red line (associated with alpha = 0.1) stoops lowest, portraying the lowest MSE at lambda = 4.72, (or log(4.72) = ~1.55 as shown in the graph).

Now, comparing models using **lambda.min** and **lambda.1se**.

```r
# for lambda.min

model_min <- glmnet(
  x_train,
  y_train,
  alpha = best_alpha,
  lambda = best_model$lambda.min
)

model_1se <- glmnet(
  x_train,
  y_train,
  alpha = best_alpha,
  lambda = best_model$lambda.1se
)

predictions_min <- drop(predict(model_min, newx = x_test))
predictions_1se <- drop(predict(model_1se, newx = x_test))

mse_min <- mse(y_test, predictions_min)
mse_1se <- mse(y_test, predictions_1se)

rmse_min <- sqrt(mse_min)
rmse_1se <- sqrt(mse_1se)

coefs_min <- coef(model_min)
coefs_1se <- coef(model_1se)
```

```
nz_min <- sum(coefs_min[-1] != 0)
nz_1se <- sum(coefs_1se[-1] != 0)

min_vs_1se_table <- tibble(
  Model = c("lambda.min", "lambda.1se"),
  alpha = best_alpha,
  lambda = c(best_model$lambda.min, best_model$lambda.1se),
  mse = c(mse_min, mse_1se),
  rmse = c(rmse_min, rmse_1se),
  predictors = c(nz_min, nz_1se)
)

kable(min_vs_1se_table, caption = "lambda.min vs lambda.1se")
```

Table 3: lambda.min vs lambda.1se

Model	alpha	lambda	mse	rmse	predictors
lambda.min	0.1	4.715981	12302.7	110.9175	13
lambda.1se	0.1	33.270315	13417.4	115.8335	11

The *Table 3* illustrates the lambda value, RMSE/MSE and number of predictors for both models. The **lambda.min** model has the best accuracy, but is more complex with two additional coefficients. On the other hand, the **lambda.1se** model fits a sparser model with only 11 precitors and a slightly inaccurate model.

```
get_nz_coefs <- function(model) {
  cf <- coef(model)
  nz <- which(cf != 0)
  kable(tibble(
    Predictor = rownames(cf)[nz],
    Coefficient = as.numeric(cf[nz])
  ) %>% filter(Predictor != "(Intercept)"))
}

nz_coefs_min <- get_nz_coefs(model_min)
nz_coefs_1se <- get_nz_coefs(model_1se)

nz_coefs_min
```

Predictor	Coefficient
ALTITUDE	0.1473201
MEAN_ANNUAL_AIR_TEMP	-22.5649422
MEAN_MONTHLY_MAX_TEMP	7.6829885
MEAN_MONTHLY_MIN_TEMP	-6.6513083
MEAN_ANNUAL_WIND_SPEED	-14.3799298
MEAN_CLOUD_COVER	3.1456437
MEAN_ANNUAL_SUNSHINE	-0.0143899
MAX_MONTHLY_WIND_SPEED	-12.0405616
MAX_AIR_TEMP	-25.8083487
MAX_WIND_SPEED	-0.4075071

Predictor	Coefficient
MAX_RAINFALL	20.6201529
MIN_AIR_TEMP	22.3598153
MEAN_RANGE_AIR_TEMP	13.8186032

```
nz_coefs_1se
```

Predictor	Coefficient
ALTITUDE	0.1648370
MEAN_ANNUAL_AIR_TEMP	-12.0389073
MEAN_MONTHLY_MIN_TEMP	-2.9104012
MEAN_ANNUAL_WIND_SPEED	-11.2199879
MEAN_CLOUD_COVER	1.8512273
MEAN_ANNUAL_SUNSHINE	-0.0098279
MAX_MONTHLY_WIND_SPEED	-5.7822744
MAX_AIR_TEMP	-17.8155396
MAX_WIND_SPEED	-0.1284573
MAX_RAINFALL	18.0826590
MIN_AIR_TEMP	11.5337501

The above tables lists the coefficients of both models. The 1se, being a more sparse model, sacrificed two coefficients and a small accuracy for a more parsimonious and generalized model that will perform better than lambda.min model on unseen data as well.

Overall, both the models provide comparable results with the **lambda.min** achieveing a marginally more accurate model. I would choose the **lambda.1se** model as it is simpler, more stable and less prone to overfitting with only 11 predictors (instead of 13), all while retaining a decently comparable accuracy (losing only 5 units of RMSE). Furthermore, despite having a worse accuracy, the model will perform better on unseen data better than **lambda.min**, which is comparatively overfitted to the dataset on hand.

LLM usage acknowledgement There have been instances where guidance from ChatGPT was used, particularly to help understand data wrangling and model training steps. There were multiple concepts and R specific code blocks, issues that was best learned with the help of ChatGPT. That being said, all the code, comments and answers have been hand-typed after understanding and comprehensing in personal format.