STAT448 Assignment 2

Thomas Waldin 17775654

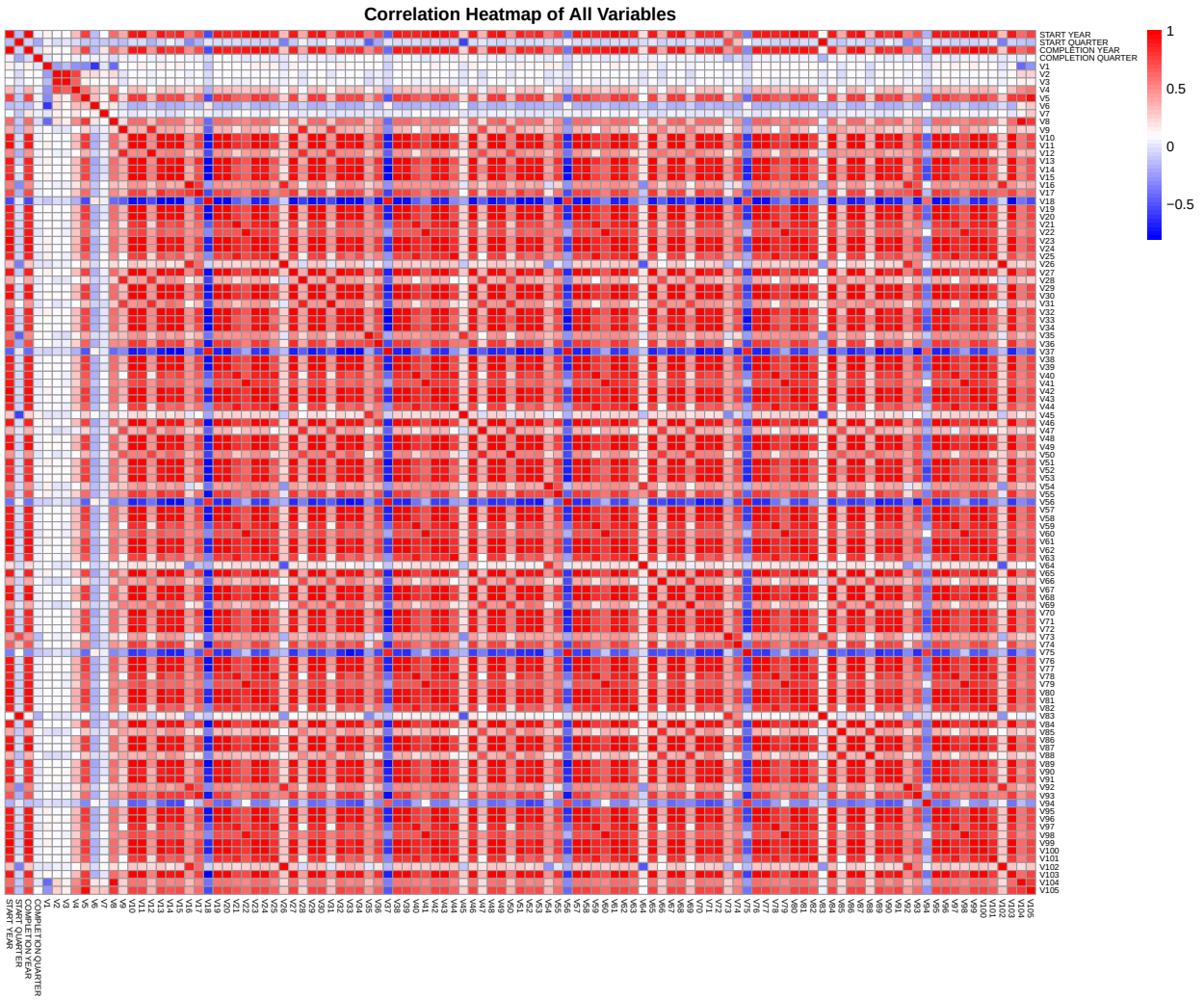
2025-04-14

Part 1

A dataset that includes a large number of variables related to residential building construction has been used.

a). The correlation between variables was explored by the following correlation heatmap.

```
load('Residen.RData')
```



Although the heatmap is cluttered, it demonstrates substantial multicollinearity among variables. This

will likely cause issues when fitting a linear regression, as it violates the key assumption that variables are independent.

b). A linear regression model was fitted to explain the 'actual sales price' (V104) in terms of all other variables excluding 'actual construction costs' (V105).

```
set.seed(0)

# Create model excluding V105
model = lm(V104 ~ ., data = Residen[, !names(Residen) %in% c("V105")])

# View summary
summary(model)
```

```
##
## Call:
## lm(formula = V104 ~ ., data = Residen[, !names(Residen) %in%
##     c("V105")])
##
## 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 p-values show that 'COMPLETION YEAR' and 'COMPLETION QUARTER' are highly significant ($p < 0.001$). V2, V3, and V8 are also significant and V5 may be significant. The p-values of all other variables are high, suggesting they would be unlikely to improve predictive performance.

32 of the variables were dropped from the model 'because of singularities' indicating perfect linear dependence on other variables. This suggest severe multicollinearity and reinforces the findings from the correlation heatmap.

The RSE is the average size of prediction errors, which is approximately 10% of the mean actual sales price.

```
cat("RSE as % of mean V104:", 148.6 / mean(Residen$V104) * 100, "%")
```

```
## RSE as % of mean V104: 10.71043 %
```

The R-squared demonstrates that 98.8% of the variance in V104 is explained by the model. The adjusted R-squared is at a slightly lower 98.5% as is penalises the number of predictor variables.The F-statistic and p-value suggest that the model is statistically significant.

c). A linear regression model was generated using a backwards selection and a step-wise selection.

```
set.seed(0)
```

```r
# Prepare data
data = Residen[, !(names(Residen) %in% c("V105"))]
train_index = sample(seq_len(nrow(data)), size = 0.8 * nrow(data))
train_data = data[train_index, ]
test_data = data[-train_index, ]
n = nrow(data)

# Backwards selection
full_model = glm(V104 ~ ., data = train_data)
start_back = Sys.time()
back_model = step(full_model, direction = "backward", trace = 0,)
end_back = Sys.time()
time_back = end_back - start_back

# Holdout MSE (on test data)
pred_back = predict(back_model, newdata = test_data)
mse_back = mean((test_data$V104 - pred_back)^2)

# Cross-Validation MSE (10-fold)
cv_mse_back = mean(cv.glm(data = train_data, glmfit = back_model, K = 10)$delta[1])

# Step wise selection
null_model = glm(V104 ~ 1, data = train_data)
start_step = Sys.time()
step_model = step(null_model, scope = list(lower=~1, upper=formula(full_model)), direction="both", trac
end_step = Sys.time()
time_step = end_step - start_step

# Holdout MSE
pred_step = predict(step_model, newdata = test_data)
mse_step = mean((test_data$V104 - pred_step)^2)

# Cross-Validation MSE
cv_mse_step = mean(cv.glm(data = train_data, glmfit = step_model, K = 10)$delta[1])

### == OUTPUT COMPARISON == ###
cat("== BACKWARD SELECTION ==\n")
cat("Time taken:", time_back, "\n")
cat("Holdout MSE:", mse_back, "\n")
cat("Cross-validated MSE:", cv_mse_back, "\n")
cat("Number of predictors:", length(coef(back_model)) - 1, "\n\n")

cat("== STEPWISE SELECTION ==\n")
cat("Time taken:", time_step, "\n")
cat("Holdout MSE:", mse_step, "\n")
cat("Cross-validated MSE:", cv_mse_step, "\n")
cat("Number of predictors:", length(coef(step_model)) - 1, "\n\n")
```

```
## == BACKWARD SELECTION ==
## Time taken: 18.37484
## Holdout MSE: 122953.4
## Cross-validated MSE: 12442.28
## Number of predictors: 52
```

```
##
## == STEPWISE SELECTION ==
## Time taken: 2.351429
## Holdout MSE: 106596.3
## Cross-validated MSE: 9822.408
## Number of predictors: 18
```

The holdout MSE for the model with step-wise selection was lower than the model with backward selection ($107000 vs. $123000). The cross-validated MSE scores were also lower for the step-wise model ($9800 vs. $12400). The step-wise model was also much faster to train and it used far fewer features (18 vs. 52). The step-wise is the better model, as the MSE is lower, it is more parsimonious, and has a lesser time cost. However, both models appear to be overfitting, as the holdout MSE is significantly larger than the cross-validated MSE for each model.

d). A linear regression model was generated using Ridge regression and LASSO regression. Note that glmnet standardises predictors by default.

```r
set.seed(0)

# Prepare data
y = Residen$V104
X = model.matrix(V104 ~ . - V105, data = Residen)[, -1]  # Remove intercept

# Ridge
start_ridge = Sys.time()
cv_ridge = cv.glmnet(X, y, alpha = 0)
end_ridge = Sys.time()
time_ridge = end_ridge - start_ridge

lambda_ridge = cv_ridge$lambda.min
model_ridge = glmnet(X, y, alpha = 0, lambda = lambda_ridge)
mse_ridge = min(cv_ridge$cvm)

# LASSO
start_lasso = Sys.time()
cv_lasso = cv.glmnet(X, y, alpha = 1)
end_lasso = Sys.time()
time_lasso = end_lasso - start_lasso

lambda_lasso = cv_lasso$lambda.min
model_lasso = glmnet(X, y, alpha = 1, lambda = lambda_lasso)
mse_lasso = min(cv_lasso$cvm)

# Output comparison
cat("== Ridge Regression ==\n")
cat("Best lambda:", lambda_ridge, "\n")
cat("Cross-validated MSE:", mse_ridge, "\n")
cat("Time taken:", time_ridge, "\n")
cat("Number of non-zero coefficients:", sum(coef(model_ridge) != 0), "\n\n")

cat("== LASSO Regression ==\n")
cat("Best lambda:", lambda_lasso, "\n")
cat("Cross-validated MSE:", mse_lasso, "\n")
cat("Time taken:", time_lasso, "\n")
cat("Number of non-zero coefficients:", sum(coef(model_lasso) != 0), "\n")
```

```
## == Ridge Regression ==
## Best lambda: 117.6074
## Cross-validated MSE: 57418.26
## Time taken: 0.217613
## Number of non-zero coefficients: 108
##
## == LASSO Regression ==
## Best lambda: 2.103583
## Cross-validated MSE: 29339.59
## Time taken: 0.09884787
## Number of non-zero coefficients: 33
```

The best λ for Ridge regression was 117.6 which is very large, indicating strong L2 regularisation and strong shrinkage of all coefficients. The best λ for LASSO regression was 1.9, suggesting moderate L1 regularisation and many coefficients have become zero. This is evident by the number of non-zero coefficients, 108 for Ridge and 34 for LASSO. The time taken for Ridge was about 2.5 times longer than LASSO. The cross-validated MSE for the LASSO regression was about half of that of the Ridge Regression ($29500 vs. $58200)

e). The LASSO regression is better in this case. It is more parsimonious, has a lower time cost, while also having a lower cross-validated MSE. Comparing the cross-validated MSE to the backward and step-wise selecting models, the Ridge and LASSO initially appear to have a greater error, but this is due to these models sacrificing accuracy to reduce overfitting. The cross-validated MSE is a better representation of the models ability to generalise in this case. LASSO regression also typically handles multicollinearity far better than the other models considered, so it makes sense that it would be the best performing in this case.

Part 2

A different dataset contains information on 42 patients with Parkinson's disease. The target variable is the total unified Parkinson's disease rating scale (UPDRS).

The dataset was read and split into a training set with 30 patients and a testing set with the remaining 12. The preparation of the data also incuded scaling.

```
data = read.csv("parkinsons.csv")
set.seed(0)

# Prepare data
y = data$UPDRS
X = as.matrix(data[, setdiff(names(data), "UPDRS")])

n = nrow(data)
train_idx = sample(seq_len(n), size = 30)
test_idx = setdiff(seq_len(n), train_idx)

X_train = X[train_idx, ]
X_test = X[test_idx, ]
y_train = y[train_idx]
y_test = y[test_idx]

# Scale data
X_train_scaled = scale(X_train)
train_center = attr(X_train_scaled, "scaled:center")
```

```
train_scale = attr(X_train_scaled, "scaled:scale")
X_test_scaled = scale(X_test, center = train_center, scale = train_scale)
```

a). It was confirmed that a linear model can fit the training data exactly.

```
model <- lm(y_train ~ X_train_scaled)

# Check residuals
residuals <- resid(model)
max(abs(residuals))  # Should be 0 (or very close)
```

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

```
# Check R-squared
summary(model)$r.squared  # Should be 1 for perfect fit
```

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

This model will not be useful. There are more predictor variables than observations (patients) so there will always be a linear combination of features that fit the observations perfectly (system of linear equations is undetermined, allowing infinite solutions). This model is overfit, and therefore has no predicting power and is not generalisable.

b). LASSO Regression was used to fit the training data, using leave-one-out cross-validation to find λ.

```
# Define grid of lambda values
grid = 10^seq(3, -1, length = 100)

# LASSO with LOOCV (nfolds = number of training samples)
lasso_cv <- cv.glmnet(
  x = X_train,
  y = y_train,
  alpha = 1,
  lambda = grid,
  nfolds = 30,
  thresh = 1e-10
)
```

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

```
# Best lambda
best_lambda = lasso_cv$lambda.min
cat("Optimal lambda:", best_lambda, "\n")

# Fit model with best lambda
lasso_model = glmnet(X_train, y_train, alpha = 1, lambda = best_lambda)

# Predict on test set
pred_test = predict(lasso_model, s = best_lambda, newx = X_test)
```

```r
# Compute test MSE
test_mse = mean((y_test - pred_test)^2)
cat("Test MSE:", test_mse, "\n")

null_mse = mean((y_test - mean(y_train))^2)
cat("Null MSE (mean predictor):", null_mse)
```

```
## Optimal lambda: 0.5336699
## Test MSE: 30.99368
## Null MSE (mean predictor): 154.297
```

The optimal value of the tuning parameter λ was computed to be 0.534 and the resulting test error was 31.0, which is significantly better than a null model (154, for context).

c). Inspecting the coefficients shows which variables were used.

```r
coef(lasso_model)
```

```
## 99 x 1 sparse Matrix of class "dgCMatrix"
##                    s0
## (Intercept) 1.515994
## X           .
## X1          .
## X2          .
## X3          .
## X4          .
## X5          .
## X6          .
## X7          .
## X8          .
## X9          .
## X10         .
## X11         .
## X12         .
## X13         .
## X14         .
## X15         .
## X16         .
## X17         .
## X18         .
## X19         .
## X20         .
## X21         .
## X22         .
## X23         .
## X24         .
## X25         .
## X26         .
## X27         .
## X28         .
## X29         .
## X30         .
## X31         .
```

```
## X32         .
## X33         .
## X34         .
## X35         .
## X36         .
## X37         .
## X38         .
## X39         .
## X40         .
## X41         .
## X42         .
## X43         .
## X44         .
## X45         .
## X46         .
## X47         .
## X48         .
## X49         .
## X50         .
## X51         .
## X52         .
## X53         .
## X54         .
## X55         .
## X56         .
## X57         .
## X58         .
## X59         .
## X60         .
## X61         .
## X62         .
## X63         .
## X64         .
## X65         .
## X66         .
## X67         .
## X68         .
## X69         .
## X70         .
## X71         .
## X72         .
## X73         .
## X74         .
## X75         .
## X76         .
## X77         .
## X78         .
## X79         .
## X80         .
## X81         .
## X82         .
## X83        6.652645
## X84         .
## X85         .
```

```
## X86         .
## X87         .
## X88         .
## X89         .
## X90         .
## X91         .
## X92         .
## X93         .
## X94         .
## X95         .
## X96         .
## X97         1.066585
```

The final model is a LASSO regression where the tuning parameter λ was found using 'leave one out cross validation'. Two variables X83 and X97 were selected out of 97 total, meaning many variables were shrunk to zero. An MSE of approximately 31 indicates the model should perform reasonably on unseen data.

Feature selection and regularisation were successfully completed by the LASSO regression. The selection of only two variables suggests many are not informative in predicting UPDRS in this dataset. X97 was known to be informative, so its inclusion makes sense. Interestingly, the coefficient for X83 was over 6 times larger than the coefficient for X97, suggesting it may have a stronger linear relationship with UPDRS.

d). The analysis was repeated with a different seed set when creating the data, generating a different random split of the training and test sets.

```
# Repeated with a different seed set
set.seed(1)

# Prepare data
y = data$UPDRS
X = as.matrix(data[, setdiff(names(data), "UPDRS")])

n = nrow(data)
train_idx = sample(seq_len(n), size = 30)
test_idx = setdiff(seq_len(n), train_idx)

X_train = X[train_idx, ]
X_test = X[test_idx, ]
y_train = y[train_idx]
y_test = y[test_idx]

# Scale data
X_train_scaled = scale(X_train)
train_center = attr(X_train_scaled, "scaled:center")
train_scale = attr(X_train_scaled, "scaled:scale")
X_test_scaled = scale(X_test, center = train_center, scale = train_scale)

# Define grid of lambda values
grid <- 10^seq(3, -1, length = 100)

# LASSO with LOOCV (nfolds = number of training samples)
lasso_cv <- cv.glmnet(
  x = X_train,
  y = y_train,
  alpha = 1,
```

```
  lambda = grid,
  nfolds = 30,
  thresh = 1e-10
)
```

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

```
# Best lambda
best_lambda <- lasso_cv$lambda.min
cat("Optimal lambda:", best_lambda, "\n")

# Fit model with best lambda
lasso_model <- glmnet(X_train, y_train, alpha = 1, lambda = best_lambda)

# Predict on test set
pred_test <- predict(lasso_model, s = best_lambda, newx = X_test)

# Compute test MSE
test_mse <- mean((y_test - pred_test)^2)
cat("Test MSE:", test_mse, "\n")
```

```
## Optimal lambda: 0.5857021
## Test MSE: 43.25471
```

With a different seed, different values of λ and MSE have been generated. The variables selected with this seed were examined.

```
coef(lasso_model)
```

```
## 99 x 1 sparse Matrix of class "dgCMatrix"
##                         s0
## (Intercept) -6.413691e-01
## X            3.290184e-02
## X1           .
## X2           .
## X3           .
## X4           .
## X5           3.352422e+01
## X6           .
## X7           .
## X8           .
## X9           .
## X10          .
## X11          7.283568e+04
## X12          .
## X13          .
## X14          .
## X15          .
## X16          .
## X17          .
## X18          .
```

```
## X19          .
## X20          .
## X21          .
## X22          .
## X23          .
## X24          .
## X25          .
## X26          .
## X27          .
## X28          .
## X29          .
## X30          .
## X31          .
## X32          .
## X33          .
## X34          .
## X35          .
## X36          .
## X37          .
## X38          .
## X39          .
## X40          .
## X41          .
## X42          .
## X43          .
## X44          .
## X45          .
## X46          .
## X47          .
## X48          .
## X49          .
## X50          .
## X51          .
## X52          .
## X53          .
## X54          .
## X55          .
## X56          .
## X57          .
## X58          .
## X59          .
## X60          .
## X61          .
## X62          .
## X63          .
## X64          .
## X65          .
## X66          .
## X67          .
## X68          .
## X69          .
## X70          .
## X71          .
## X72          .
```

```
## X73          .
## X74          .
## X75          .
## X76          .
## X77          .
## X78          .
## X79          .
## X80          .
## X81          .
## X82          .
## X83          .
## X84          .
## X85          .
## X86          .
## X87          .
## X88          .
## X89          .
## X90          .
## X91          7.463780e-01
## X92          .
## X93          .
## X94          .
## X95          8.350295e+00
## X96          .
## X97          1.137869e+00
```

X97 was selected again, but X83 was not. In addition X, X5, X11, and X95 were selected. This suggests that X97 is a robust predictor (which is known) but perhaps the other features may not be, if different features are chosen each time. Due to the small sample size its possible that the other features may be noise. Training a number of different models could be a good next step and see what features are consistently predictors.

Part 3

A dataset containing 2000 observations of weather station data including thirteen features (listed in output) was loaded and split randomly using a 80/20 split.

```
data = read.csv("Weather_Station_data_v1.csv")
set.seed(0)
names(data)
```

```
##  [1] "ALTITUDE"               "MEAN_ANNUAL_AIR_TEMP"   "MEAN_MONTHLY_MAX_TEMP"
##  [4] "MEAN_MONTHLY_MIN_TEMP"  "MEAN_ANNUAL_WIND_SPEED" "MEAN_CLOUD_COVER"
##  [7] "MEAN_ANNUAL_SUNSHINE"   "MEAN_ANNUAL_RAINFALL"   "MAX_MONTHLY_WIND_SPEED"
## [10] "MAX_AIR_TEMP"           "MAX_WIND_SPEED"         "MAX_RAINFALL"
## [13] "MIN_AIR_TEMP"           "MEAN_RANGE_AIR_TEMP"
```

```
# Prepare data
y = data$MEAN_ANNUAL_RAINFALL
X = as.matrix(data[, names(data) != "MEAN_ANNUAL_RAINFALL"])

n = nrow(data)
train_idx = sample(seq_len(n), size = n*0.8)
```

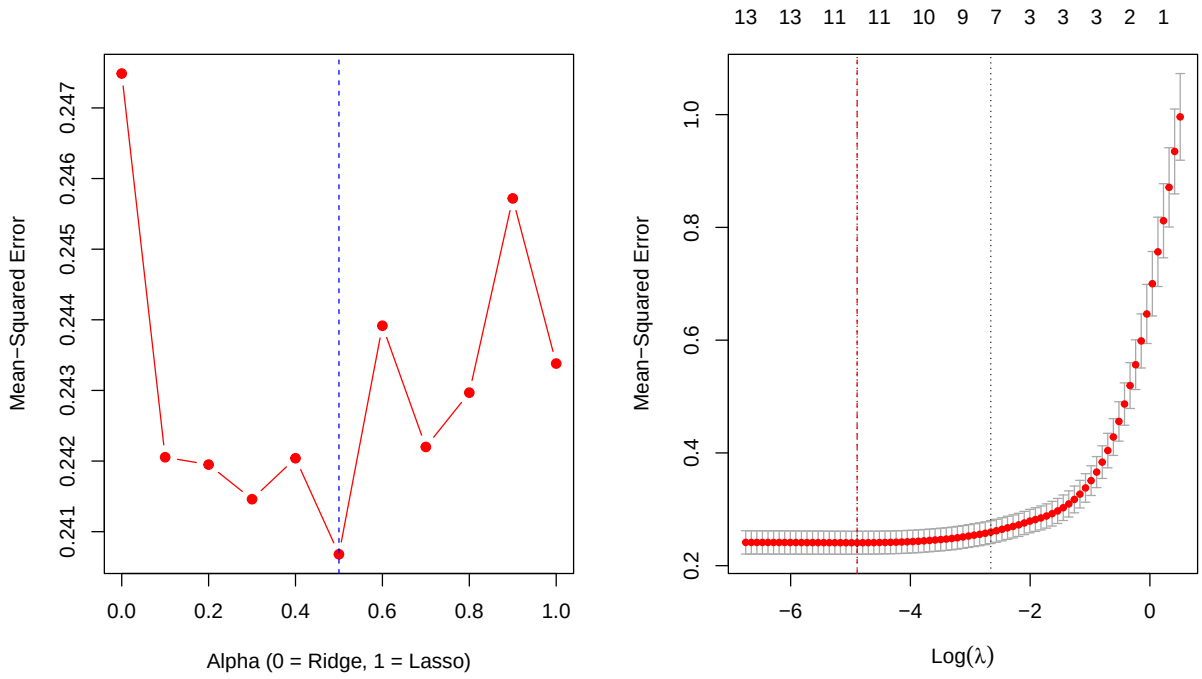
```
test_idx = setdiff(seq_len(n), train_idx)

X_train = X[train_idx, ]
X_test = X[test_idx, ]
y_train = y[train_idx]
y_test = y[test_idx]

# Scale data
X_train_scaled = scale(X_train)
train_center = attr(X_train_scaled, "scaled:center")
train_scale = attr(X_train_scaled, "scaled:scale")
X_test_scaled = scale(X_test, center = train_center, scale = train_scale)
y_train_scaled = scale(y_train)
y_mean = attr(y_train_scaled, "scaled:center")
y_sd = attr(y_train_scaled, "scaled:scale")
```

An ElasticNet model was used to predict the mean annual rainfall using 10-fold cross-validation to optimise values for α and λ.

```
library(glmnet)
set.seed(0)

# Search over alpha values
alphas = seq(0, 1, by = 0.1)
cv_errors = c()
models = list()

for (a in alphas) {
  cv = cv.glmnet(X_train_scaled, y_train_scaled, alpha = a, nfolds = 10)
  cv_errors = c(cv_errors, min(cv$cvm))
  models[[as.character(a)]] = cv
}

# Find best alpha and corresponding lambda
best_alpha = alphas[which.min(cv_errors)]
best_model = models[[as.character(best_alpha)]]
best_lambda = best_model$lambda.min

# Report results
cat("Best alpha:", best_alpha, "\n")
cat("Best lambda:", best_lambda, "\n")
```

```
## Best alpha: 0.5
## Best lambda: 0.007524363
```

The cross-validation results for the model were plotted below.



As can be seen in the left plot, the lowest CV MSE occurs at $\alpha = 0.5$, suggesting a mix of LASSO and Ridge regularization works best. The right plot shows the coefficient shrinkage as λ increases, with vertical lines at the minimum λ, and at the maximum λ within one standard error.

Predictions were made on the test set using both lambda.min and lambda.1se. The coefficients of each model, the MSE and RMSE of each, and the number of predictors used in each model are shown below.

```
# Fit final models for BOTH lambda.min and lambda.1se
final_model_min = glmnet(X_train_scaled, y_train_scaled,
                         alpha=best_alpha, lambda=best_model$lambda.min)
final_model_1se = glmnet(X_train_scaled, y_train_scaled,
                         alpha=best_alpha, lambda=best_model$lambda.1se)

# Predictions (unscaled)
pred_min = predict(final_model_min, newx=X_test_scaled) * y_sd + y_mean
pred_1se = predict(final_model_1se, newx=X_test_scaled) * y_sd + y_mean

# Calculate errors
mse_min = mean((y_test - pred_min)^2)
mse_1se = mean((y_test - pred_1se)^2)
rmse_min = sqrt(mse_min)
rmse_1se = sqrt(mse_1se)

# Count non-zero predictors
num_pred_min = sum(coef(final_model_min) != 0) - 1  # Exclude intercept
num_pred_1se = sum(coef(final_model_1se) != 0) - 1

# Print comprehensive results
cat("\n=== lambda.min ===\n")
cat("MSE:", mse_min, "\n")
cat("RMSE:", rmse_min, "\n")
```

```r
cat("Number of predictors used:", num_pred_min, "\n\n")

cat("=== lambda.1se ===\n")
cat("MSE:", mse_1se, "\n")
cat("RMSE:", rmse_1se, "\n")
cat("Number of predictors used:", num_pred_1se, "\n\n")

# Show coefficients comparison
cat("Coefficients (lambda.min):\n")
coef(final_model_min)
cat("\nCoefficients (lambda.1se):\n")
coef(final_model_1se)
```

```
##
## === lambda.min ===
## MSE: 13629.45
## RMSE: 116.7452
## Number of predictors used: 11
##
## === lambda.1se ===
## MSE: 14901.59
## RMSE: 122.0721
## Number of predictors used: 7
##
## Coefficients (lambda.min):
## 14 x 1 sparse Matrix of class "dgCMatrix"
##                                      s0
## (Intercept)            -3.186123e-16
## ALTITUDE                1.628213e-01
## MEAN_ANNUAL_AIR_TEMP   -4.969996e-02
## MEAN_MONTHLY_MAX_TEMP   .
## MEAN_MONTHLY_MIN_TEMP  -7.853089e-02
## MEAN_ANNUAL_WIND_SPEED -4.824807e-02
## MEAN_CLOUD_COVER        3.313915e-02
## MEAN_ANNUAL_SUNSHINE   -1.470526e-02
## MAX_MONTHLY_WIND_SPEED -4.744404e-02
## MAX_AIR_TEMP           -2.074625e-01
## MAX_WIND_SPEED          .
## MAX_RAINFALL            6.492224e-01
## MIN_AIR_TEMP            2.617869e-01
## MEAN_RANGE_AIR_TEMP     9.395625e-02
##
## Coefficients (lambda.1se):
## 14 x 1 sparse Matrix of class "dgCMatrix"
##                                      s0
## (Intercept)            -3.430990e-16
## ALTITUDE                1.756886e-01
## MEAN_ANNUAL_AIR_TEMP    .
## MEAN_MONTHLY_MAX_TEMP   .
## MEAN_MONTHLY_MIN_TEMP   .
## MEAN_ANNUAL_WIND_SPEED -2.263262e-02
## MEAN_CLOUD_COVER        6.965909e-03
## MEAN_ANNUAL_SUNSHINE    .
```

```
## MAX_MONTHLY_WIND_SPEED -6.093245e-03
## MAX_AIR_TEMP           -1.488026e-01
## MAX_WIND_SPEED          .
## MAX_RAINFALL            6.148917e-01
## MIN_AIR_TEMP            6.668271e-02
## MEAN_RANGE_AIR_TEMP     .
```

The lambda.min value is the value of λ that minimises the cross validation MSE, which typically leads to the best accuracy in predictions. This model uses 11 predictors, which more than the 7 used by the other model. The increased complexity may risk overfitting the training data.

The lambda.1se value is the largest value of λ within one standard error of the lambda.min value. This model is simpler, with fewer predictors, likely with higher bias but lower variance. While the accuracy of this model is lower, it is likely to generalise better to unseen data.

In this case the simpler model with lambda.1se is recommended. There are only 2000 observations so having a less complex model that is less likely to overfit is preferable in this case. The slight increase in RMSE (122 vs. 116) is likely worth it to reduce the complexity from 11 variables to 7.