Simulation Code

Art Tay

Data Generation

Simulation Scheme

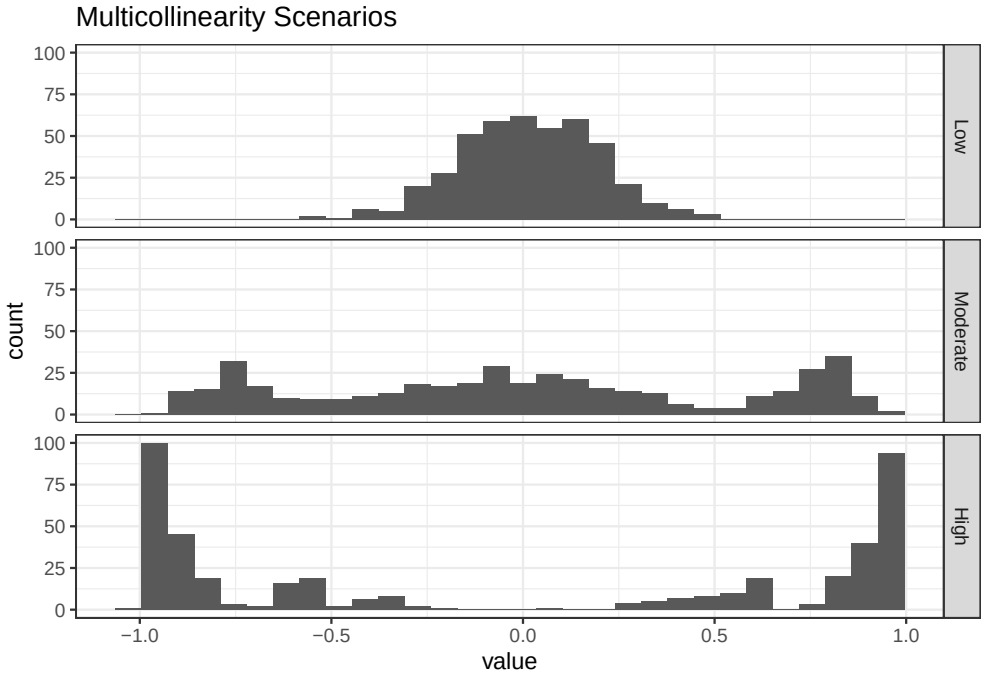
We assume that our data generating model follows and the standard assumptions of MLR.

$$Y = [\vec{1}, X]\beta + \epsilon, \quad \epsilon \sim MVN(0, \sigma^2 I)$$

We will simulation potential predictor values by sampling from $X \sim MVN(\vec{0}, \Sigma_X)$, where

$$\Sigma_X = \begin{bmatrix} 1 & \rho_{12} & \cdots & \rho_{1p} \\ \rho_{21} & 1 & \cdots & \rho_{2p} \\ \vdots & \cdots & \ddots & \vdots \\ \rho_{n1} & \cdots & \cdots & 1 \end{bmatrix}$$

This structure follow from the assumption that the predictor space needs to be centered and scaled to apply PCA. The multicollinearity of the predictor space will be simulated to follow a numerically shaped correlation matrices:



We will also simulate the β parameters to evaluate the usefulness of regularization.

True Parameters		
Large Negative	Small	Large Positive
$\beta_{1...k} \sim U[-1, -0.5]$	$\beta_{k+1...j} \sim U[-0.3, 0.3]$	$\beta_{j+1...p} \sim U[0.5, 1]$

Furthermore to avoid an possibility of error associated with zero-intercept models, we will assume $\beta_0 = 1$.

We will draw 100 sample data set from the above regression model with 30 predictor and 100 observations. This will be the training data. We will also draw 50 sample data set of the size 20 to be test data. Between each sample the correlation matrix will be fixed as will the βs. Will be assume that there are 10 large negative βs, 10 small βs, and 10 large positive βs.

Functions

```
extract_upper_tri <- function(x){
    #' Extract Upper Triangle
    #' @description Extracts the upper triangular portion,
    #' excluding the diagonal, of a given matrix.
    #' @param x A given matrix.
    #' @return A vector containing the value of the upper
    #' triangle.
    #'

    # Creates a empty vector return object.
    ret <- c()

    # Loops through the upper triangle of the matrix
    # appending the values to the return object.
    for(i in 1:(ncol(x)-1)){
        for(j in (i + 1):ncol(x)){
            ret <- append(ret, x[i, j])
        }
    }

    return(ret)
}
```

```
gen_cor_mat <- function(shape = 2, n, maxit = 100){
    #' Shaped Correlation Matrix Random Generator
    #' @description Generates positive definite correlation
    #' matrices of size n x n based on a shape parameter.
    #' @param shape A real number. 0-1 creates normally distributed
    #' correlations centered around zero. 1.7 creates a roughly
    #' bimodal distribution at -0.5 and 0.5. 2 Creates a uniform spread.
    #' Anything greater creates a bimodal distribution around -1 and 1.
    #' Although values can be negative, only the absolute value of the shape
    #' parameter matters. No perfect correlation are usually created.
    #' @param n The size of the desired matrix
    #' @param maxit The max iteration to attempt to find a positive definite
    #' matrix.
```

```
    #' @return An n x n positive definite correlation matrix.

    # Algorithm Reference:
    # https://stats.stackexchange.com/questions/124538/how-to-generate-a-large-full-rank-random-correla
    for(i in 1:maxit){
        A <- t(replicate(n, rnorm(n) + rnorm(1) * shape))
        A <- A %*% t(A)
        D_half <- diag(diag(A)^(-0.5))
        C <- D_half %*% A %*% D_half

        # Round off to ensure symmetry.
        C <- round(C, 4)

        if(is.positive.definite(C)){
            return(C)
        }
    }

    stop("No positive definite matrix found.", call. = F)
}
```

```
gen_betas <- function(bln, bs, blp){
    #' Generate Sized Betas
    #' @description Creates a vector of betas of varying
    #' size from a uniform distribution.
    #' @param bln The desired number of large negative betas.
    #' @param bs The desired number of small betas.
    #' @param blp The desired number of large positive betas.
    #' @return A vector of betas.

    # Generate betas of desired size.
    betas <- c(runif(n = bln, min = -1, max = -1/2),
               runif(n = bs, min = -0.2, max = 0.2),
               runif(n = blp, min = 1/2, max = 1))

    return(betas)
}
```

```
sim_data <- function(n = 100, cor_mat, betas, bint = 1){
    #' Simulate Data for MLR Method Testing
    #' @description Generic function for generating a single sample
    #' from the scheme described above.
    #' @param n The desired sample size.
    #' @param cor_mat The correlation matrix of the predictor space.
    #' @param betas A vector of predictor betas.
    #' @param bint The desired intercept beta, defaults to 1.
    #' @return An n by p + 1 dataframe with the response Y followed by
    #' the p X predictor values.

    # Exit if the beta vector does not match the length of
    #the correlation martix.
    if(nrow(cor_mat) != length(betas)) {
        stop("Predictor space dimension does not match parameter space.",
```

```
            call. = F)
    }

    # Generate a sample of predictor values.
    X <- mvrnorm(n = n, mu = rep(0, nrow(cor_mat)), Sigma = cor_mat)
    X <- cbind(rep(1, n), X) # Dummy column for the intercept.

    # Generate random errors with unit variance.
    epsilon <- mvrnorm(n = 1, mu = rep(0, n), Sigma = diag(1, n, n))

    betas <- c(bint, betas) # Add intercept beta.

    # Calculate the response based on the model assumptions.
    Y <- X %*% as.matrix(betas) + epsilon

    ret <- cbind(Y, X[, -1]) # Removes the intercept column.

    # Creates predictor column names.
    pred_names <- sapply(seq(from = 1, to = nrow(cor_mat), by = 1),
        FUN = function(x){
            paste("X", x, sep = "")
        })

    # Renames the columns of the simulated data.
    colnames(ret) <- c("Y", pred_names)

    return(as.data.frame(ret))
}
```

Sampling

```
# Generate correlation matrices under different shape parameters.
set.seed(123)
low_cor_mat <- gen_cor_mat(shape = 0, n = 30)
med_cor_mat <- gen_cor_mat(shape = 2, n = 30)
high_cor_mat <- gen_cor_mat(shape = 6, n = 30)

# Extract correlation values from matrices.
low_cor <- extract_upper_tri(low_cor_mat)
med_cor <- extract_upper_tri(med_cor_mat)
high_cor <- extract_upper_tri(high_cor_mat)

# Plot side-by-side histograms
cor_scenarios_plot <- cbind(Low = low_cor, Moderate = med_cor, High = high_cor)
cor_scenarios_plot <- as.data.frame(cor_scenarios_plot)
cor_scenarios_plot %<>%
    pivot_longer(cols = everything()) %>%
    mutate(name = factor(name, levels = c("Low", "Moderate", "High")))

cor_scenarios_plot <- cor_scenarios_plot %>%
    ggplot(aes(value)) +
    geom_histogram() +
```

```
    facet_grid(name~.) +
    ggtitle("Multicollinearity Scenarios") +
    theme_bw()

save(cor_scenarios_plot, file = 'Figures/cor_scenarios_plot.rds')
```

```
set.seed(123)
betas <- gen_betas(bln = 10, bs = 10, blp = 10)
parameter_dist_plot <- betas %>% as.data.frame() %>%
    ggplot(aes(x = betas)) +
    geom_histogram() +
    ggtitle("True Parameter Values") +
    theme_bw()

save(parameter_dist_plot, file = "Figures/parameter_dist_plot.rds")
```

Training Data

```
set.seed(123)
train_low <- replicate(100, sim_data(n = 100, cor_mat = low_cor_mat, betas))
train_med <- replicate(100, sim_data(n = 100, cor_mat = med_cor_mat, betas))
train_high <- replicate(100, sim_data(n = 100, cor_mat = high_cor_mat, betas))
```

Testing Data

```
set.seed(123)
test_low <- replicate(50, sim_data(n = 20, cor_mat = low_cor_mat, betas))
test_med <- replicate(50, sim_data(n = 20, cor_mat = med_cor_mat, betas))
test_high <- replicate(50, sim_data(n = 20, cor_mat = high_cor_mat, betas))
```

Metric Functions

```
train_rmse <- function(train_data, model_list){
    #'
    #' @param train_data A array of training data sets where
    #' each column represents a different dataset.
    #' @param model_list A list of fit workflow objects.
    #' @return A vector of RMSE values associate with each
    #' model.

    ret <- sapply(1:ncol(train_data),
        FUN = function(i) {
            # Extract training data for the ith model.
            data <- as.data.frame(train_data[, i])
            # Calculate training RMSE
            res <- (data[1] - predict(model_list[[i]], new_data = data[-1]))
            sqrt(sum(res^2) / nrow(data))
```

```
        }
    )

    return(ret)
}
```

```
test_rmse <- function(test_data, model_list){
    #'
    #' @param test_data A array of testing data sets where
    #' each column represents a different dataset.
    #' @param model_list A list of fit workflow objects.
    #' @return A matrix of RMSE values with each row
    #' representing a dataset and each column a model.

    ret <- future_sapply(model_list,
        # Loops through the models.
        FUN = function(x){
            apply(test_data, MARGIN = 2,
                # Loops through the test sets.
                FUN = function(y){
                    y <- as.data.frame(y)
                    # Calculates RMSE
                    res <- y[1] - predict(x, new_data = y[, -1])
                    sqrt(sum(res^2) / nrow(y))
                }
            )
        },
        future.packages = c('tidymodels')
    )
}
```

```
count_pred_lm <- function(model_list){
    #'
    #' @param model_list A list of lm class models.
    #' @return A vector of the number of predictors
    #' used in the model.

    ret <- sapply(model_list,
        FUN = function(x){
            length(x$coefficients) - 1 # don't count intercept.
        }
    )

    return(ret)
}
```

```
count_sig_param_lm <- function(model_list, alpha = 0.5){
    #'
    #' @param model_list A list of lm class models.
    #' @param alpha The desired Type I error rate.
    #' @return A vector of the number of significant parameters
    #' in the model.
```

```
    ret <- sapply(model_list,
        FUN = function(x){
            x %<>% extract_fit_engine()
            sum(summary(x)$coefficients[, 4] <= alpha)
        }
    )

    return(ret)
}
```

```
non_zero_param_glmnet <- function(model_list){
    #'
    #' @param model_list A list of Workflow objects
    #' with glmnet as the engine.
    #' @return A vector of the number of non-zero parameters
    #' used in each model.

    ret <- sapply(model_list,
        FUN = function(x){
            fit <- extract_fit_parsnip(x) %>% tidy()
            sum(fit$estimate > 0)
        }
    )

    return(ret)
}
```

Modeling Functions

```
fit_model_list_lm <- function(train_data, model_wflow){
    #'
    #' @param train_data A array of training data, where each
    #' column represents a different sample.
    #' @param model_wflow A workflow object of a lm engined model.
    model_list <-  apply(train_data, MARGIN = 2,
        # Loops through training samples.
        FUN = function(x){
            x <- as.data.frame(x)
            # Applies basic fit that works with lm.
            model_wflow %>% fit(data = x)
        }
    )

    return(model_list)
}
```

```
fit_model_list_glmnet <- function(train_data, model_wflow){
    #'
    #' @param train_data
    #' @param model_wflow
    #' @return
```

```
    # define a grid of lambdas to check.
    param_grid <- grid_regular(penalty(), levels = 10)

    ret <- future_lapply(1:ncol(train_data),
        # Loop through all training samples.
        FUN = function(i){
            #browser()
            train_sample <- as.data.frame(train_data[, i])

            # Define 10-Fold CV
            resamples <- train_sample %>% vfold_cv(v = 10)

            # Tune model on grid.
            fit <- model_wflow %>%
                tune_grid(resamples = resamples, grid = param_grid,
                          metrics = metric_set(rmse))

            # Select best tuned model.
            fit <- finalize_workflow(model_wflow, select_best(fit))

            # Fit best model to data.
            fit %>% fit(data = train_sample)
        },
        future.packages = c('tidymodels'), future.seed = T
    )

    return(ret)
}
```

Workflow Definitions

OLS

```
# Define the model engine.
lm_mod <- linear_reg(engine = "lm")

# Package into Workflow.
ols_wflow <- workflow() %>%
    add_model(lm_mod) %>%
    add_formula(Y ~ .)
```

OLS Regularized

```
# Define model engine.
lasso_spec <- linear_reg(penalty = tune(), mixture = 1) %>%
    set_engine('glmnet')

# Package into Workflow.
ols_reg_wflow <- workflow() %>% add_model(lasso_spec) %>%
    add_formula(Y ~ .)
```

PCA

```
# Build a model framework in tidymodels

# Extract a arb. sample; just for structure.
dummy_dataset <- as.data.frame(train_high[, 1])

# Define the recipe.
pca_recipe <- recipe(Y ~ ., data = dummy_dataset) %>%
    step_pca(all_numeric_predictors(), num_comp = 30) # changed from threshold = 1

# Define the model engine.
lm_mod <- linear_reg(engine = "lm")

# Put it all in a workflow.
pca_wflow <- workflow() %>%
    add_model(lm_mod) %>%
    add_recipe(pca_recipe)
```

PCA-Cut

```
# Build a model framework in tidymodels

# Extract a arb. sample; just for structure.
dummy_dataset <- as.data.frame(train_high[, 1])

# Define the recipe.
pca_cut_recipe <- recipe(Y ~ ., data = dummy_dataset) %>%
    step_pca(all_numeric_predictors(), threshold = 0.75)

# Define the model engine.
lm_mod <- linear_reg(engine = "lm")

# Put it all in a workflow.
pca_cut_wflow <- workflow() %>%
    add_model(lm_mod) %>%
    add_recipe(pca_cut_recipe)
```

PCA Regularized

```
# Build a model framework in tidymodels

# Extract a arb. sample; just for structure.
dummy_dataset <- as.data.frame(train_high[, 1])

# Define the recipe.
pca_reg_recipe <- recipe(Y ~ ., data = dummy_dataset) %>%
    step_pca(all_numeric_predictors(), num_comp = 30)
```

```
# Define model engine.
lasso_spec <- linear_reg(penalty = tune(), mixture = 1) %>%
    set_engine('glmnet')

# Package into Workflow
pca_reg_wflow <- workflow() %>%
    add_model(lasso_spec) %>%
    add_recipe(pca_reg_recipe)
```

PLS

```
# Build a model framework in tidymodels

# Extract a arb. sample; just for structure.
dummy_dataset <- as.data.frame(train_high[, 1])

# Define the recipe.
pls_recipe <- recipe(Y ~ ., data = dummy_dataset) %>%
    step_pls(all_numeric_predictors(), outcome = 'Y', num_comp = 30)

# Define the model engine.
lm_mod <- linear_reg(engine = "lm")

# Put it all in a workflow.
pls_wflow <- workflow() %>%
    add_model(lm_mod) %>%
    add_recipe(pls_recipe)
```

PLS Regularized

```
# Build a model framework in tidymodels

# Extract a arb. sample; just for structure.
dummy_dataset <- as.data.frame(train_high[, 1])

# Define the recipe.
pls_reg_recipe <- recipe(Y ~ ., data = dummy_dataset) %>%
    step_pls(all_numeric_predictors(), outcome = 'Y', num_comp = 30)

# Define the model engine.
lasso_spec <- linear_reg(penalty = tune(), mixture = 1) %>%
    set_engine('glmnet')

# Put it all in a workflow.
pls_reg_wflow <- workflow() %>%
    add_model(lasso_spec) %>%
    add_recipe(pls_reg_recipe)
```

Model Fitting and Metric Calculations

OLS

```
set.seed(123)
# Low correlation case
ols_fit_low <- fit_model_list_lm(train_low, ols_wflow)
# Moderate correlation case
ols_fit_med <- fit_model_list_lm(train_med, ols_wflow)
# High correlation case
ols_fit_high <- fit_model_list_lm(train_high, ols_wflow)
```

```
# Low correlation case
ols_low_train_rmse <- train_rmse(train_low, ols_fit_low)
ols_low_test_rmse <- test_rmse(test_low, ols_fit_low)
ols_low_sig_param <- count_sig_param_lm(ols_fit_low)

# Moderate correlation case
ols_med_train_rmse <- train_rmse(train_med, ols_fit_med)
ols_med_test_rmse <- test_rmse(test_med, ols_fit_med)
ols_med_sig_param <- count_sig_param_lm(ols_fit_med)

# High correlation case
ols_high_train_rmse <- train_rmse(train_high, ols_fit_high)
ols_high_test_rmse <- test_rmse(test_high, ols_fit_high)
ols_high_sig_param <- count_sig_param_lm(ols_fit_high)
```

OLS Regularized

```
set.seed(123)
# Low correlation case
ols_reg_fit_low <- fit_model_list_glmnet(train_low, ols_reg_wflow)
# Moderate correlation case
ols_reg_fit_med <- fit_model_list_glmnet(train_med, ols_reg_wflow)
# High correlation case
ols_reg_fit_high <- fit_model_list_glmnet(train_high, ols_reg_wflow)
```

```
# Low correlation case
ols_reg_low_train_rmse <- train_rmse(train_low, ols_reg_fit_low)
ols_reg_low_test_rmse <- test_rmse(test_low, ols_reg_fit_low)
ols_reg_low_sig_param <- non_zero_param_glmnet(ols_reg_fit_low)

# Moderate correlation case
ols_reg_med_train_rmse <- train_rmse(train_med, ols_reg_fit_med)
ols_reg_med_test_rmse <- test_rmse(test_med, ols_reg_fit_med)
ols_reg_med_sig_param <- non_zero_param_glmnet(ols_reg_fit_med)

# High correlation case
ols_reg_high_train_rmse <- train_rmse(train_high, ols_reg_fit_high)
```

```r
ols_reg_high_test_rmse <- test_rmse(test_high, ols_reg_fit_high)
ols_reg_high_sig_param <- non_zero_param_glmnet(ols_reg_fit_high)
```

PCA

```r
set.seed(123)
# Low correlation case
pca_fit_low <- fit_model_list_lm(train_low, pca_wflow)
# Moderate correlation case
pca_fit_med <- fit_model_list_lm(train_med, pca_wflow)
# High correlation case
pca_fit_high <- fit_model_list_lm(train_high, pca_wflow)
```

```r
# Low correlation case
pca_low_train_rmse <- train_rmse(train_low, pca_fit_low)
pca_low_test_rmse <- test_rmse(test_low, pca_fit_low)
pca_low_sig_param <- count_sig_param_lm(pca_fit_low)

# Moderate correlation case
pca_med_train_rmse <- train_rmse(train_med, pca_fit_med)
pca_med_test_rmse <- test_rmse(test_med, pca_fit_med)
pca_med_sig_param <- count_sig_param_lm(pca_fit_med)

# High correlation case
pca_high_train_rmse <- train_rmse(train_high, pca_fit_high)
pca_high_test_rmse <- test_rmse(test_high, pca_fit_high)
pca_high_sig_param <- count_sig_param_lm(pca_fit_high)
```

PCA - Cutoff Rule

```r
set.seed(123)
# Low correlation case
pca_cut_fit_low <- fit_model_list_lm(train_low, pca_cut_wflow)
# Moderate correlation case
pca_cut_fit_med <- fit_model_list_lm(train_med, pca_cut_wflow)
# High correlation case
pca_cut_fit_high <- fit_model_list_lm(train_high, pca_cut_wflow)
```

```r
# Low correlation case
pca_cut_low_train_rmse <- train_rmse(train_low, pca_cut_fit_low)
pca_cut_low_test_rmse <- test_rmse(test_low, pca_cut_fit_low)
pca_cut_low_sig_param <- count_sig_param_lm(pca_cut_fit_low)

# Moderate correlation case
pca_cut_med_train_rmse <- train_rmse(train_med, pca_cut_fit_med)
pca_cut_med_test_rmse <- test_rmse(test_med, pca_cut_fit_med)
pca_cut_med_sig_param <- count_sig_param_lm(pca_cut_fit_med)
```

```r
# High correlation case
pca_cut_high_train_rmse <- train_rmse(train_high, pca_cut_fit_high)
pca_cut_high_test_rmse <- test_rmse(test_high, pca_cut_fit_high)
pca_cut_high_sig_param <- count_sig_param_lm(pca_cut_fit_high)
```

PCA Regularized

```r
set.seed(123)
# Low correlation case
pca_reg_fit_low <- fit_model_list_glmnet(train_low, pca_reg_wflow)
# Moderate correlation case
pca_reg_fit_med <- fit_model_list_glmnet(train_med, pca_reg_wflow)
# High correlation case
pca_reg_fit_high <- fit_model_list_glmnet(train_high, pca_reg_wflow)
```

```r
# Low correlation case
pca_reg_low_train_rmse <- train_rmse(train_low, pca_reg_fit_low)
pca_reg_low_test_rmse <- test_rmse(test_low, pca_reg_fit_low)
pca_reg_low_sig_param <- non_zero_param_glmnet(pca_reg_fit_low)

# Moderate correlation case
pca_reg_med_train_rmse <- train_rmse(train_med, pca_reg_fit_med)
pca_reg_med_test_rmse <- test_rmse(test_med, pca_reg_fit_med)
pca_reg_med_sig_param <- non_zero_param_glmnet(pca_reg_fit_med)

# High correlation case
pca_reg_high_train_rmse <- train_rmse(train_high, pca_reg_fit_high)
pca_reg_high_test_rmse <- test_rmse(test_high, pca_reg_fit_high)
pca_reg_high_sig_param <- non_zero_param_glmnet(pca_reg_fit_high)
```

PLS

```r
set.seed(123)
# Low correlation case
pls_fit_low <- fit_model_list_lm(train_low, pls_wflow)
# Moderate correlation case
pls_fit_med <- fit_model_list_lm(train_med, pls_wflow)
# High correlation case
pls_fit_high <- fit_model_list_lm(train_high, pls_wflow)
```

```r
# Low correlation case
pls_low_train_rmse <- train_rmse(train_low, pls_fit_low)
pls_low_test_rmse <- test_rmse(test_low, pls_fit_low)
pls_low_sig_param <- count_sig_param_lm(pls_fit_low)

# Moderate correlation case
pls_med_train_rmse <- train_rmse(train_med, pls_fit_med)
pls_med_test_rmse <- test_rmse(test_med, pls_fit_med)
```

```
pls_med_sig_param <- count_sig_param_lm(pls_fit_med)

# High correlation case
pls_high_train_rmse <- train_rmse(train_high, pls_fit_high)
pls_high_test_rmse <- test_rmse(test_high, pls_fit_high)
pls_high_sig_param <- count_sig_param_lm(pls_fit_high)
```

PLS Regularized

```
set.seed(123)
# Low correlation case
pls_reg_fit_low <- fit_model_list_glmnet(train_low, pls_reg_wflow)
# Moderate correlation case
pls_reg_fit_med <- fit_model_list_glmnet(train_med, pls_reg_wflow)
# High correlation case
pls_reg_fit_high <- fit_model_list_glmnet(train_high, pls_reg_wflow)
```

```
# Low correlation case
pls_reg_low_train_rmse <- train_rmse(train_low, pls_reg_fit_low)
pls_reg_low_test_rmse <- test_rmse(test_low, pls_reg_fit_low)
pls_reg_low_sig_param <- non_zero_param_glmnet(pls_reg_fit_low)

# Moderate correlation case
pls_reg_med_train_rmse <- train_rmse(train_med, pls_reg_fit_med)
pls_reg_med_test_rmse <- test_rmse(test_med, pls_reg_fit_med)
pls_reg_med_sig_param <- non_zero_param_glmnet(pls_reg_fit_med)

# High correlation case
pls_reg_high_train_rmse <- train_rmse(train_high, pls_reg_fit_high)
pls_reg_high_test_rmse <- test_rmse(test_high, pls_reg_fit_high)
pls_reg_high_sig_param <- non_zero_param_glmnet(pls_reg_fit_high)
```

Tables

```
t_conf_asString <- function(x){
    test <- try(round(t.test(x, conf.level = 0.99)$conf.int, 2))
    if('try-error' %in% class(test)){
        return(mean(x))
    }
    else{
        return(paste("(", test[1], ", ", test[2], ")", sep = ""))
    }
}

#t_conf_asString(rnorm(10))
#t_conf_asString(rep(10, 10))
```

Low Table

```
low_results <- list(ols_low_train_rmse, ols_low_test_rmse, ols_low_sig_param,
                    ols_reg_low_train_rmse, ols_reg_low_test_rmse, ols_reg_low_sig_param,
                    pca_low_train_rmse, pca_low_test_rmse, pca_low_sig_param,
                    pca_cut_low_train_rmse, pca_cut_low_test_rmse, pca_cut_low_sig_param,
                    pca_reg_low_train_rmse, pca_reg_low_test_rmse, pca_reg_low_sig_param,
                    pls_low_train_rmse, pls_low_test_rmse, pls_low_sig_param,
                    pls_reg_low_train_rmse, pls_reg_low_test_rmse, pls_reg_low_sig_param)

table_low <- sapply(low_results, FUN = t_conf_asString)

col_label <- rep(c("Training RMSE", "Test RMSE", "Parameters"), 7)

Model <- c(rep("OLS", 3), rep("LASSO", 3),
    rep("PCA", 3), rep("PCA + Cutoff", 3), rep("PCA + LASSO", 3),
    rep("PLS", 3), rep("PLS + LASSO", 3))


table_low <- cbind(table_low, col_label, Model) %>% as.data.frame()

table_low %<>% pivot_wider(names_from = col_label, values_from = table_low)

table_low
```

```
## # A tibble: 7 x 4
##   Model        `Training RMSE` `Test RMSE`  Parameters
##   <chr>        <chr>           <chr>        <chr>
## 1 OLS          (0.81, 0.85)    (1.19, 1.2)  (19.49, 21.57)
## 2 LASSO        (0.82, 0.86)    (1.17, 1.18) (15.13, 15.81)
## 3 PCA          (0.81, 0.85)    (1.19, 1.2)  (25.91, 26.93)
## 4 PCA + Cutoff (1.88, 1.99)    (2.16, 2.19) (9.55, 10.07)
## 5 PCA + LASSO  (0.84, 0.89)    (1.19, 1.2)  (13.97, 15.55)
## 6 PLS          (0.81, 0.85)    (1.19, 1.2)  (13.15, 14.43)
## 7 PLS + LASSO  (0.85, 0.89)    (1.17, 1.18) (14, 17.74)
```

```
save(table_low, file = "Figures/table_low.rds")
```

Moderate Table

```
med_results <- list(ols_med_train_rmse, ols_med_test_rmse, ols_med_sig_param,
                    ols_reg_med_train_rmse, ols_reg_med_test_rmse, ols_reg_med_sig_param,
                    pca_med_train_rmse, pca_med_test_rmse, pca_med_sig_param,
                    pca_cut_med_train_rmse, pca_cut_med_test_rmse, pca_cut_med_sig_param,
                    pca_reg_med_train_rmse, pca_reg_med_test_rmse, pca_reg_med_sig_param,
                    pls_med_train_rmse, pls_med_test_rmse, pls_med_sig_param,
                    pls_reg_med_train_rmse, pls_reg_med_test_rmse, pls_reg_med_sig_param)

table_med <- sapply(med_results, FUN = t_conf_asString)
```

```
col_label <- rep(c("Training RMSE", "Test RMSE", "Parameters"), 7)

Model <- c(rep("OLS", 3), rep("LASSO", 3),
    rep("PCA", 3), rep("PCA + Cutoff", 3), rep("PCA + LASSO", 3),
    rep("PLS", 3), rep("PLS + LASSO", 3))


table_med <- cbind(table_med, col_label, Model) %>% as.data.frame()

table_med %<>% pivot_wider(names_from = col_label, values_from = table_med)

table_med
```

```
## # A tibble: 7 x 4
##   Model        `Training RMSE` `Test RMSE`  Parameters
##   <chr>        <chr>           <chr>        <chr>
## 1 OLS          (0.8, 0.83)     (1.18, 1.19) (14.99, 19.69)
## 2 LASSO        (0.83, 0.88)    (1.13, 1.14) (12.86, 14.08)
## 3 PCA          (0.8, 0.83)     (1.18, 1.19) (24.51, 25.53)
## 4 PCA + Cutoff (1.77, 1.86)    (1.91, 1.94) (4.9, 5.32)
## 5 PCA + LASSO  (0.85, 0.9)     (1.15, 1.16) (12.83, 14.37)
## 6 PLS          (0.8, 0.83)     (1.18, 1.19) (15.79, 17.13)
## 7 PLS + LASSO  (0.85, 0.89)    (1.12, 1.13) (15.16, 17.66)
```

```
save(table_med, file = "Figures/table_med.rds")
```

High Table

```
high_results <- list(ols_high_train_rmse, ols_high_test_rmse, ols_high_sig_param,
                     ols_reg_high_train_rmse, ols_reg_high_test_rmse, ols_reg_high_sig_param,
                     pca_high_train_rmse, pca_high_test_rmse, pca_high_sig_param,
                     pca_cut_high_train_rmse, pca_cut_high_test_rmse, pca_cut_high_sig_param,
                     pca_reg_high_train_rmse, pca_reg_high_test_rmse, pca_reg_high_sig_param,
                     pls_high_train_rmse, pls_high_test_rmse, pls_high_sig_param,
                     pls_reg_high_train_rmse, pls_reg_high_test_rmse, pls_reg_high_sig_param)

table_high <- sapply(high_results, FUN = t_conf_asString)
```

```
## Error in t.test.default(x, conf.level = 0.99) :
##   data are essentially constant
```

```
col_label <- rep(c("Training RMSE", "Test RMSE", "Parameters"), 7)

Model <- c(rep("OLS", 3), rep("LASSO", 3),
    rep("PCA", 3), rep("PCA + Cutoff", 3), rep("PCA + LASSO", 3),
    rep("PLS", 3), rep("PLS + LASSO", 3))


table_high <- cbind(table_high, col_label, Model) %>% as.data.frame()
```

```
table_high %<>% pivot_wider(names_from = col_label, values_from = table_high)

table_high
```

```
## # A tibble: 7 x 4
##   Model        `Training RMSE` `Test RMSE`  Parameters
##   <chr>        <chr>           <chr>        <chr>
## 1 OLS          (0.8, 0.84)     (1.23, 1.25) (16.35, 19.11)
## 2 LASSO        (0.86, 0.91)    (1.15, 1.17) (9.27, 10.59)
## 3 PCA          (0.8, 0.84)     (1.23, 1.25) (20.8, 22.1)
## 4 PCA + Cutoff (1.7, 1.77)     (1.76, 1.78) 2
## 5 PCA + LASSO  (0.88, 0.91)    (1.16, 1.18) (10.04, 11.44)
## 6 PLS          (0.8, 0.84)     (1.23, 1.25) (15.26, 17.14)
## 7 PLS + LASSO  (0.87, 0.9)     (1.14, 1.15) (14.16, 15.76)
```

```
save(table_high, file = "Figures/table_high.rds")
```

Result Plot

Test RMSE

```
low_test_rmse_df <- list(ols_low_test_rmse, ols_reg_low_test_rmse,
    pca_low_test_rmse, pca_cut_low_test_rmse, pca_reg_low_test_rmse,
    pls_low_test_rmse, pls_reg_low_test_rmse)

low_test_rmse_df <- as.data.frame(sapply(low_test_rmse_df, FUN = c))

colnames(low_test_rmse_df) <- c("OLS", "LASSO", "PCA", "PCA + Cutoff",
    "PCA + LASSO", "PLS", "PLS + LASSO")

low_test_rmse_df %<>% pivot_longer(cols = everything(), names_to = "Models",
    values_to = "Test_RMSE") %>%
    group_by(Models) %>%
    summarise(
        Mean = mean(Test_RMSE),
    ) %>%
    mutate(Multicollinearity = "Low")
```

```
med_test_rmse_df <- list(ols_med_test_rmse, ols_reg_med_test_rmse,
    pca_med_test_rmse, pca_cut_med_test_rmse, pca_reg_med_test_rmse,
    pls_med_test_rmse, pls_reg_med_test_rmse)

med_test_rmse_df <- as.data.frame(sapply(med_test_rmse_df, FUN = c))

colnames(med_test_rmse_df) <- c("OLS", "LASSO", "PCA", "PCA + Cutoff",
    "PCA + LASSO", "PLS", "PLS + LASSO")

med_test_rmse_df %<>% pivot_longer(cols = everything(), names_to = "Models",
    values_to = "Test_RMSE") %>%
    group_by(Models) %>%
```

```
    summarise(
        Mean = mean(Test_RMSE),
    ) %>%
    mutate(Multicollinearity = "Moderate")
```

```
high_test_rmse_df <- list(ols_high_test_rmse, ols_reg_high_test_rmse,

    pca_high_test_rmse, pca_cut_high_test_rmse, pca_reg_high_test_rmse,
    pls_high_test_rmse, pls_reg_high_test_rmse)

high_test_rmse_df <- as.data.frame(sapply(high_test_rmse_df, FUN = c))

colnames(high_test_rmse_df) <- c("OLS", "LASSO", "PCA", "PCA + Cutoff",
    "PCA + LASSO", "PLS", "PLS + LASSO")

high_test_rmse_df %<>% pivot_longer(cols = everything(), names_to = "Models",
    values_to = "Test_RMSE") %>%
    group_by(Models) %>%
    summarise(
        Mean = mean(Test_RMSE),
    ) %>%
    mutate(Multicollinearity = "High")
```

```
comparison_test_plot <- as.data.frame(rbind(low_test_rmse_df, med_test_rmse_df,
    high_test_rmse_df)) %>%
    mutate(Multicollinearity = factor(Multicollinearity,
        levels = c("Low", "Moderate", "High")))

comparison_test_plot <- comparison_test_plot %>%
    ggplot(aes(x = Models, y = Mean,
        fill = Multicollinearity)) +
        geom_bar(stat = 'identity', position = 'dodge') +
        scale_x_discrete(guide = guide_axis(n.dodge = 2)) +
        xlab("Model") + ylab("Mean Test RMSE") +
        theme_bw()

save(comparison_test_plot, file = 'Figures/comparison_test_plot.rds')
```

Parameters

```
low_sig_param_df <- list(ols_low_sig_param, ols_reg_low_sig_param,
    pca_low_sig_param, pca_cut_low_sig_param, pca_reg_low_sig_param,
    pls_low_sig_param, pls_reg_low_sig_param)

low_sig_param_df <- as.data.frame(sapply(low_sig_param_df, FUN = c))

colnames(low_sig_param_df) <- c("OLS", "LASSO", "PCA", "PCA + Cutoff",
    "PCA + LASSO", "PLS", "PLS + LASSO")

low_sig_param_df %<>% pivot_longer(cols = everything(), names_to = "Models",
    values_to = "Sig_PARAM") %>%
```

```
    group_by(Models) %>%
    summarise(
        Mean = mean(Sig_PARAM),
    ) %>%
    mutate(Multicollinearity = "Low")
```

```
med_sig_param_df <- list(ols_med_sig_param, ols_reg_med_sig_param,
    pca_med_sig_param, pca_cut_med_sig_param, pca_reg_med_sig_param,
    pls_med_sig_param, pls_reg_med_sig_param)

med_sig_param_df <- as.data.frame(sapply(med_sig_param_df, FUN = c))

colnames(med_sig_param_df) <- c("OLS", "LASSO", "PCA", "PCA + Cutoff",
    "PCA + LASSO", "PLS", "PLS + LASSO")

med_sig_param_df %<>% pivot_longer(cols = everything(), names_to = "Models",
    values_to = "Sig_PARAM") %>%
    group_by(Models) %>%
    summarise(
        Mean = mean(Sig_PARAM),
    ) %>%
    mutate(Multicollinearity = "Moderate")
```

```
high_sig_param_df <- list(ols_high_sig_param, ols_reg_high_sig_param,

    pca_high_sig_param, pca_cut_high_sig_param, pca_reg_high_sig_param,
    pls_high_sig_param, pls_reg_high_sig_param)

high_sig_param_df <- as.data.frame(sapply(high_sig_param_df, FUN = c))

colnames(high_sig_param_df) <- c("OLS", "LASSO", "PCA", "PCA + Cutoff",
    "PCA + LASSO", "PLS", "PLS + LASSO")

high_sig_param_df %<>% pivot_longer(cols = everything(), names_to = "Models",
    values_to = "Sig_PARAM") %>%
    group_by(Models) %>%
    summarise(
        Mean = mean(Sig_PARAM),
    ) %>%
    mutate(Multicollinearity = "High")
```

```
comparison_param_plot <- as.data.frame(rbind(low_sig_param_df, med_sig_param_df,
    high_sig_param_df)) %>%
    mutate(Multicollinearity = factor(Multicollinearity,
        levels = c("Low", "Moderate", "High")))

comparison_param_plot <- comparison_param_plot %>%
    ggplot(aes(x = Models, y = Mean,
        fill = Multicollinearity)) +
        geom_bar(stat = 'identity', position = 'dodge') +
        scale_x_discrete(guide = guide_axis(n.dodge = 2)) +
        xlab("Model") + ylab("Parameters") +
        theme_bw()
```

```r
save(comparison_param_plot, file = 'Figures/comparison_param_plot.rds')
```