Biostat 212a Homework 6

Due Mar 22, 2024 @ 11:59PM

AUTHOR
Zongzhe Lin UID:206328707

PUBLISHED
March 20, 2024

Load R libraries.

```r
library(tidyverse)
```

```
Warning: 程辑包'ggplot2'是用R版本4.3.3 来建造的
```

```
Warning: 程辑包'tidyr'是用R版本4.3.3 来建造的
```

```
Warning: 程辑包'readr'是用R版本4.3.3 来建造的
```

```
Warning: 程辑包'stringr'是用R版本4.3.3 来建造的
```

```r
library(tidymodels)
```

```
Warning: 程辑包'tidymodels'是用R版本4.3.3 来建造的
```

```
Warning: 程辑包'dials'是用R版本4.3.3 来建造的
```

```
Warning: 程辑包'scales'是用R版本4.3.3 来建造的
```

```
Warning: 程辑包'infer'是用R版本4.3.3 来建造的
```

```
Warning: 程辑包'modeldata'是用R版本4.3.3 来建造的
```

```
Warning: 程辑包'parsnip'是用R版本4.3.3 来建造的
```

```
Warning: 程辑包'recipes'是用R版本4.3.3 来建造的
```

```
Warning: 程辑包'rsample'是用R版本4.3.3 来建造的
```

```
Warning: 程辑包'tune'是用R版本4.3.3 来建造的
```

```
Warning: 程辑包'workflows'是用R版本4.3.3 来建造的
```

```
Warning: 程辑包'workflowsets'是用R版本4.3.3 来建造的
```

```
Warning: 程辑包'yardstick'是用R版本4.3.3 来建造的
```

```r
library(readr)
library(tswge)
```

```
Warning: 程辑包'tswge'是用R版本4.3.3 来建造的
```

```r
library(ggplot2)
library(ggthemes)
```

```
Warning: 程辑包'ggthemes'是用R版本4.3.3 来建造的
```

```r
library(dplyr)
library(glmnet)
```

```
Warning: 程辑包'Matrix'是用R版本4.3.3 来建造的
```

```r
library(tidyr)
library(purrr)
library(randomForest)
library(caret)
```

```
Warning: 程辑包'lattice'是用R版本4.3.3 来建造的
```

```r
library(stats)
library(data.table)
```

```
Warning: 程辑包'data.table'是用R版本4.3.3 来建造的
```

```r
library(recipes)
library(embed)
```

```
Warning: 程辑包'embed'是用R版本4.3.3 来建造的
```

```r
library(tidytext)
```

```
Warning: 程辑包'tidytext'是用R版本4.3.3 来建造的
```

```r
library(yardstick)
library(rsample)
library(ranger)
```

```
Warning: 程辑包'ranger'是用R版本4.3.3 来建造的
```

```r
library(tune)
library(xgboost)
```

```
Warning: 程辑包'xgboost'是用R版本4.3.3 来建造的
```

```r
library(doParallel)
```

```
Warning: 程辑包'doParallel'是用R版本4.3.3 来建造的
```

```r
acfdf <- function(vec) {
    vacf <- acf(vec, plot = F)
    with(vacf, data.frame(lag, acf))
}
```

```
ggacf <- function(vec) {
    ac <- acfdf(vec)
    ggplot(data = ac, aes(x = lag, y = acf)) + geom_hline(aes(yintercept = 0)) +
        geom_segment(mapping = aes(xend = lag, yend = 0))
}

tplot <- function(vec) {
    df <- data.frame(X = vec, t = seq_along(vec))
    ggplot(data = df, aes(x = t, y = X)) + geom_line()
}
```

1 New York Stock Exchange (NYSE) data (1962-1986) (140 pts)

The NYSE.csv file contains three daily time series from the New York Stock Exchange (NYSE) for the period Dec 3, 1962-Dec 31, 1986 (6,051 trading days).

- `Log trading volume` (v_t): This is the fraction of all outstanding shares that are traded on that day, relative to a 100-day moving average of past turnover, on the log scale.
- `Dow Jones return` (r_t): This is the difference between the log of the Dow Jones Industrial Index on consecutive trading days.
- `Log volatility` (z_t): This is based on the absolute values of daily price movements.

```
# Read in NYSE data from url

url = "https://raw.githubusercontent.com/ucla-biostat-212a/2024winter/master/slides/data/NYSE.
NYSE <- read_csv(url)

NYSE
```

```
# A tibble: 6,051 × 6
   date       day_of_week DJ_return log_volume log_volatility train
   <date>     <chr>           <dbl>      <dbl>          <dbl> <lgl>
 1 1962-12-03 mon         -0.00446     0.0326           -13.1 TRUE
 2 1962-12-04 tues         0.00781     0.346            -11.7 TRUE
 3 1962-12-05 wed          0.00384     0.525            -11.7 TRUE
 4 1962-12-06 thur        -0.00346     0.210            -11.6 TRUE
 5 1962-12-07 fri          0.000568    0.0442           -11.7 TRUE
 6 1962-12-10 mon         -0.0108      0.133            -10.9 TRUE
 7 1962-12-11 tues         0.000124   -0.0115           -11.0 TRUE
 8 1962-12-12 wed          0.00336     0.00161          -11.0 TRUE
 9 1962-12-13 thur        -0.00330    -0.106            -11.0 TRUE
10 1962-12-14 fri          0.00447    -0.138            -11.0 TRUE
# ℹ 6,041 more rows
```

The **autocorrelation** at lag ℓ is the correlation of all pairs $(v_t, v_{t-\ell})$ that are ℓ trading days apart. These sizable correlations give us confidence that past values will be helpful in predicting the future.

▶ Code

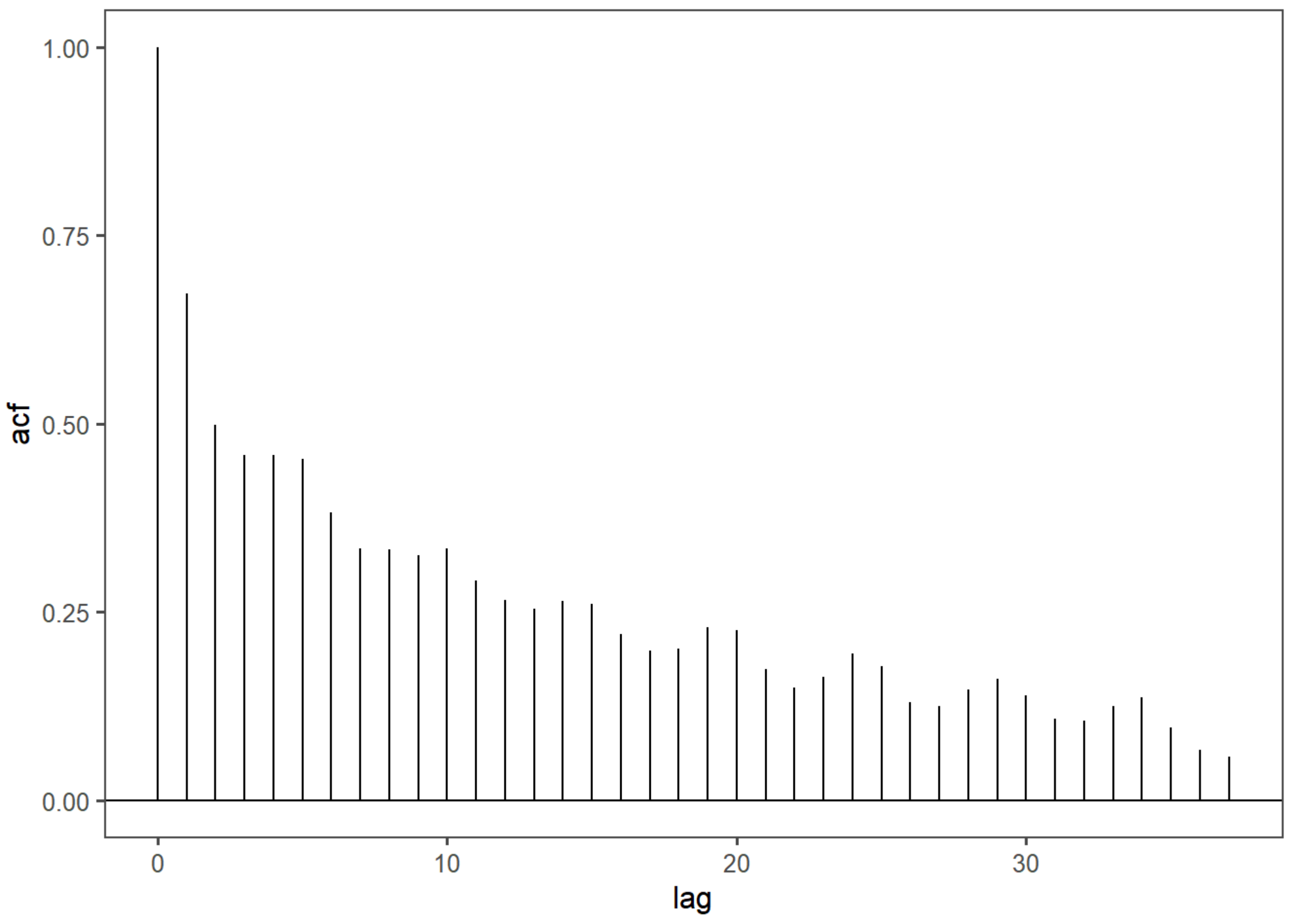
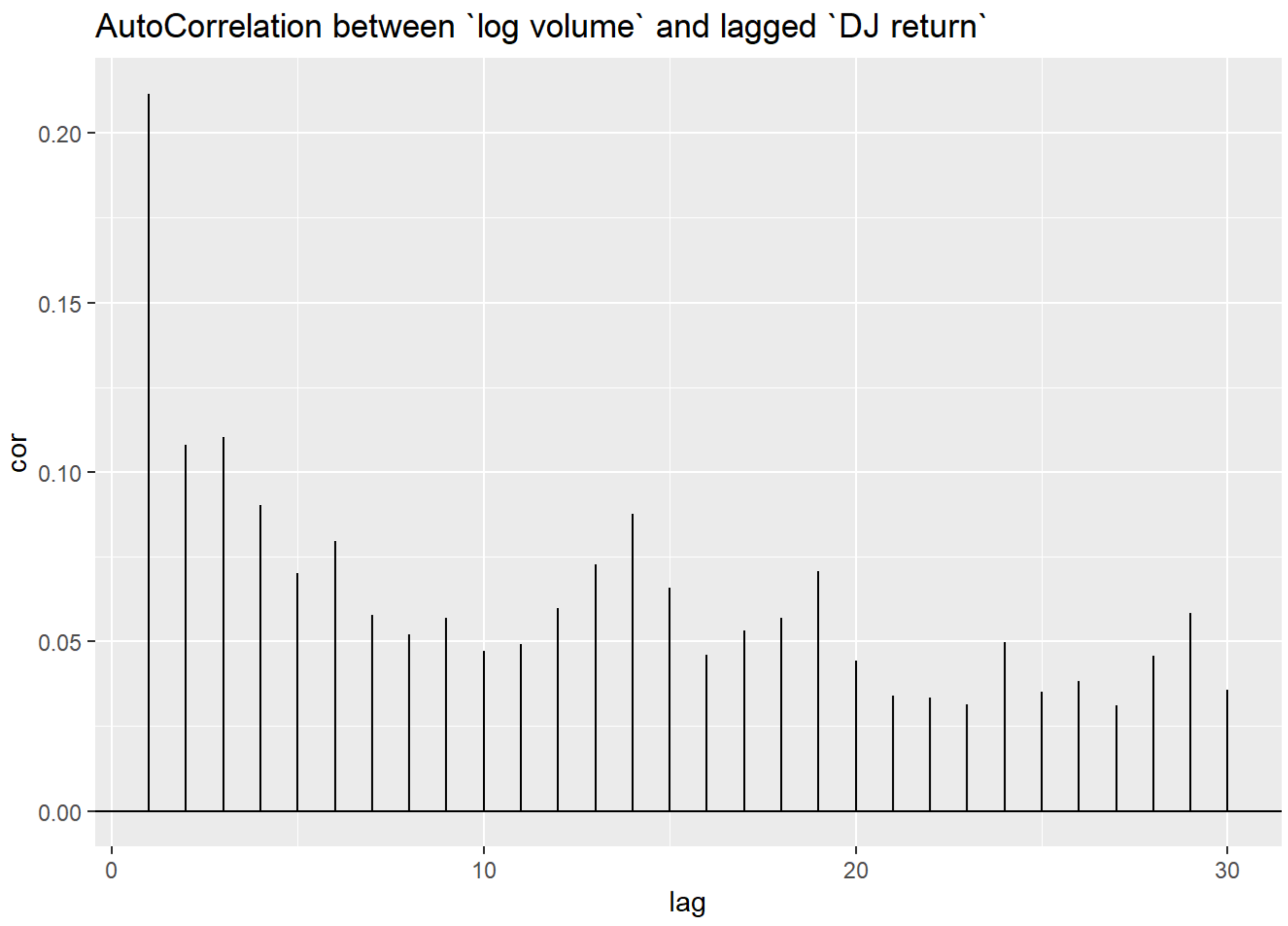


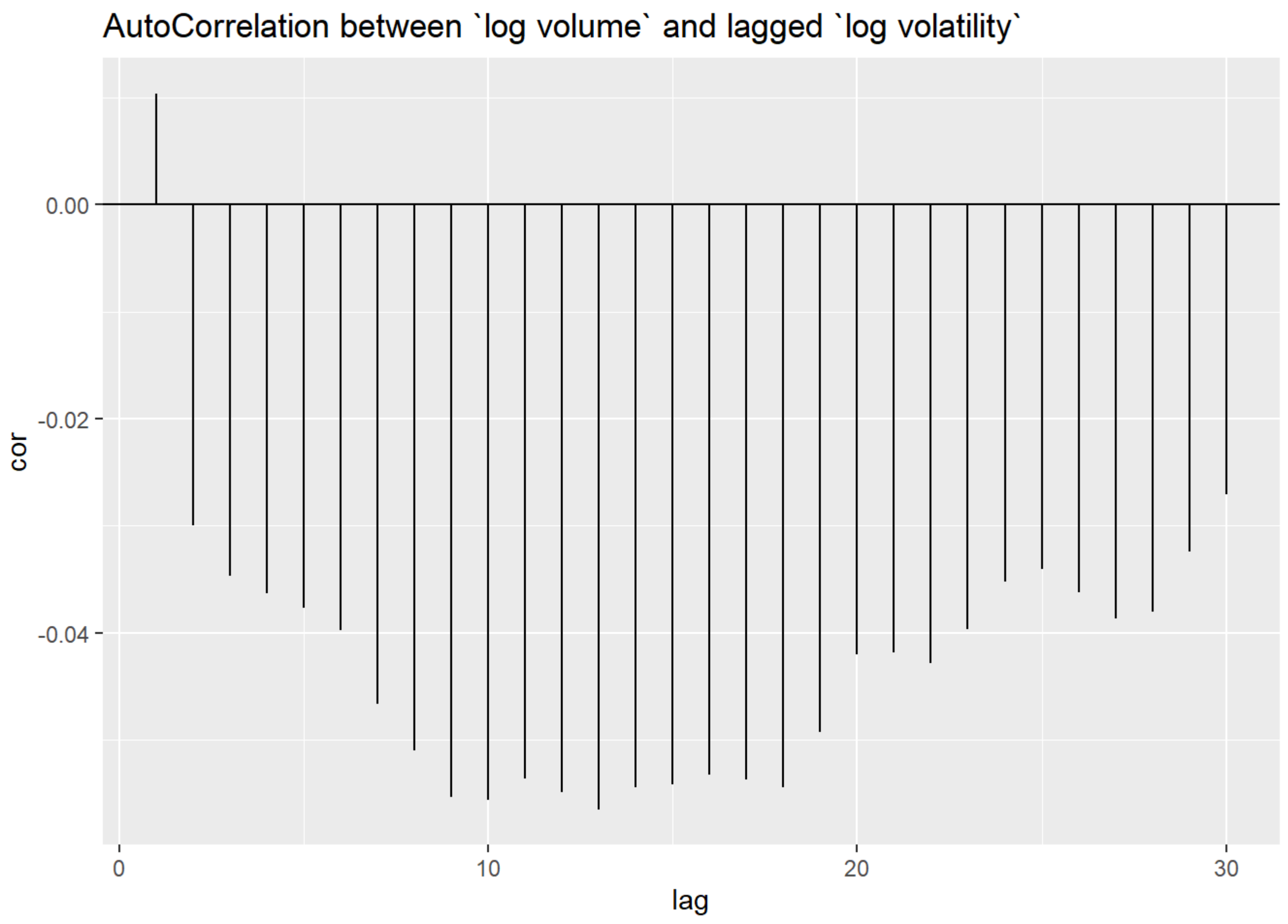
Figure 1: The autocorrelation function for log volume. We see that nearby values are fairly strongly correlated, with correlations above 0.2 as far as 20 days apart.

Do a similar plot for (1) the correlation between v_t and lag ℓ `Dow Jones return` $r_{t-\ell}$ and (2) correlation between v_t and lag ℓ `Log volatility` $z_{t-\ell}$.

```
seq(1, 30) %>%
  map(function(x) {cor(NYSE$log_volume , lag(NYSE$DJ_return, x), use = "pairwise.complete.obs"
  unlist() %>%
  tibble(lag = 1:30, cor = .) %>%
  ggplot(aes(x = lag, y = cor)) +
  geom_hline(aes(yintercept = 0)) +
  geom_segment(mapping = aes(xend = lag, yend = 0)) +
  ggtitle("AutoCorrelation between `log volume` and lagged `DJ return`")
```



```
seq(1, 30) %>%
  map(function(x) {cor(NYSE$log_volume , lag(NYSE$log_volatility, x), use = "pairwise.complete
  unlist() %>%
  tibble(lag = 1:30, cor = .) %>%
  ggplot(aes(x = lag, y = cor)) +
  geom_hline(aes(yintercept = 0)) +
  geom_segment(mapping = aes(xend = lag, yend = 0)) +
  ggtitle("AutoCorrelation between `log volume` and lagged `log volatility`")
```



1.1 Project goal

Our goal is to forecast daily `Log trading volume`, using various machine learning algorithms we learnt in this class.

The data set is already split into train (before Jan 1st, 1980, $n_{\text{train}} = 4,281$) and test (after Jan 1st, 1980, $n_{\text{test}} = 1,770$) sets.

In general, we will tune the lag L to acheive best forecasting performance. In this project, we would fix $L = 5$. That is we always use the previous five trading days' data to forecast today's `log trading volume`.

Pay attention to the nuance of splitting time series data for cross validation. Study and use the `time-series` functionality in tidymodels. Make sure to use the same splits when tuning different machine learning algorithms.

Use the R^2 between forecast and actual values as the cross validation and test evaluation criterion.

1.2 Baseline method (20 pts)

We use the straw man (use yesterday's value of `log trading volume` to predict that of today) as the baseline method. Evaluate the R^2 of this method on the test data.

```
# Lag: look back L trading days
# Do not need to include, as we included them in receipe

L = 5
```

```r
for(i in seq(1, L)) {
  NYSE <- NYSE %>%
    mutate(!!paste("DJ_return_lag", i, sep = "") := lag(NYSE$DJ_return, i),
           !!paste("log_volume_lag", i, sep = "") := lag(NYSE$log_volume, i),
           !!paste("log_volatility_lag", i, sep = "") := lag(NYSE$log_volatility, i))
}


NYSE <-   NYSE %>% na.omit()
```

```r
# Drop beginning trading days which lack some lagged variables
NYSE_other <- NYSE %>%
  filter(train == 'TRUE') %>%
  select(-train) %>%
  drop_na()
dim(NYSE_other)
```

```
[1] 4276   20
```

```r
NYSE_test = NYSE %>%
  filter(train == 'FALSE') %>%
  select(-train) %>%
  drop_na()
dim(NYSE_test)
```

```
[1] 1770   20
```

```r
library(yardstick)
# cor(NYSE_test$log_volume, NYSE_test$log_volume_lag1) %>% round(2)
r2_test_strawman =  rsq_vec(NYSE_test$log_volume, lag(NYSE_test$log_volume, 1)) %>% round(2)
print(paste("Straw man test R2: ", r2_test_strawman))
```

```
[1] "Straw man test R2:  0.35"
```

```r
set.seed(1101)

# Define the number of folds for cross-validation
n_folds <- 10
n_repeats <- 5
cv_r2 <- numeric(n_folds * n_repeats)

# Manually create cross-validation folds
folds <- createFolds(NYSE_other$log_volume, k = n_folds, list = TRUE, returnTrain = FALSE)

# Repeat the cross-validation process
for (r in seq_len(n_repeats)) {
  for (f in seq_along(folds)) {
    # Extract the test fold
    test_indices <- folds[[f]]
    cv_test_set <- NYSE_other[test_indices, ]

    # Compute the "straw man" predictions
    strawman_preds <- stats::lag(cv_test_set$log_volume, -5)

    # Remove the NA value caused by the lag at the end of the vector
    strawman_preds <- strawman_preds[1:(length(strawman_preds)-1)]
    actual_values <- cv_test_set$log_volume[2:length(cv_test_set$log_volume)]

    # Calculate the R^2 for this fold
    cv_r2_value <- rsq_vec(actual_values, strawman_preds)

    # Store the R^2 value for this fold
    cv_r2[(r - 1) * n_folds + f] <- cv_r2_value
  }
}

# Calculate the average R^2 across all folds and repeats
cv_r2_mean <- mean(cv_r2, na.rm = TRUE)

print(paste("Cross-validation Straw man R2: ", round(cv_r2_mean, 2)))
```

```
[1] "Cross-validation Straw man R2:  0.17"
```

1.3 Autoregression (AR) forecaster (30 pts)

- Let

$$y = \begin{pmatrix} v_{L+1} \\ v_{L+2} \\ v_{L+3} \\ \vdots \\ v_T \end{pmatrix}, \quad M = \begin{pmatrix} 1 & v_L & v_{L-1} & \cdots & v_1 \\ 1 & v_{L+1} & v_L & \cdots & v_2 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & v_{T-1} & v_{T-2} & \cdots & v_{T-L} \end{pmatrix}.$$

- Fit an ordinary least squares (OLS) regression of y on M, giving

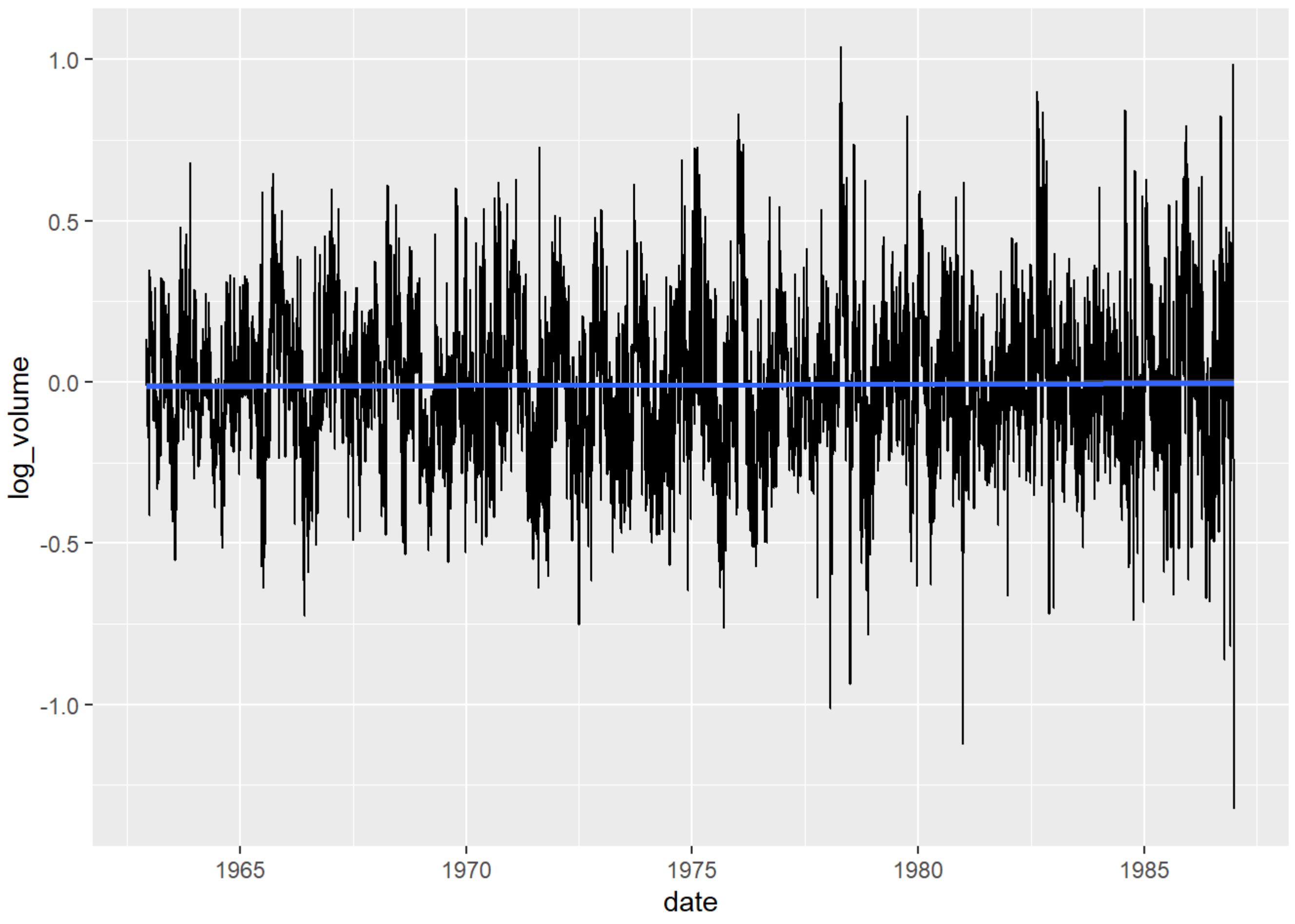
$$\hat{v}_t = \hat{\beta}_0 + \hat{\beta}_1 v_{t-1} + \hat{\beta}_2 v_{t-2} + \cdots + \hat{\beta}_L v_{t-L},$$

  known as an **order-L** autoregression model or **AR(L)**.

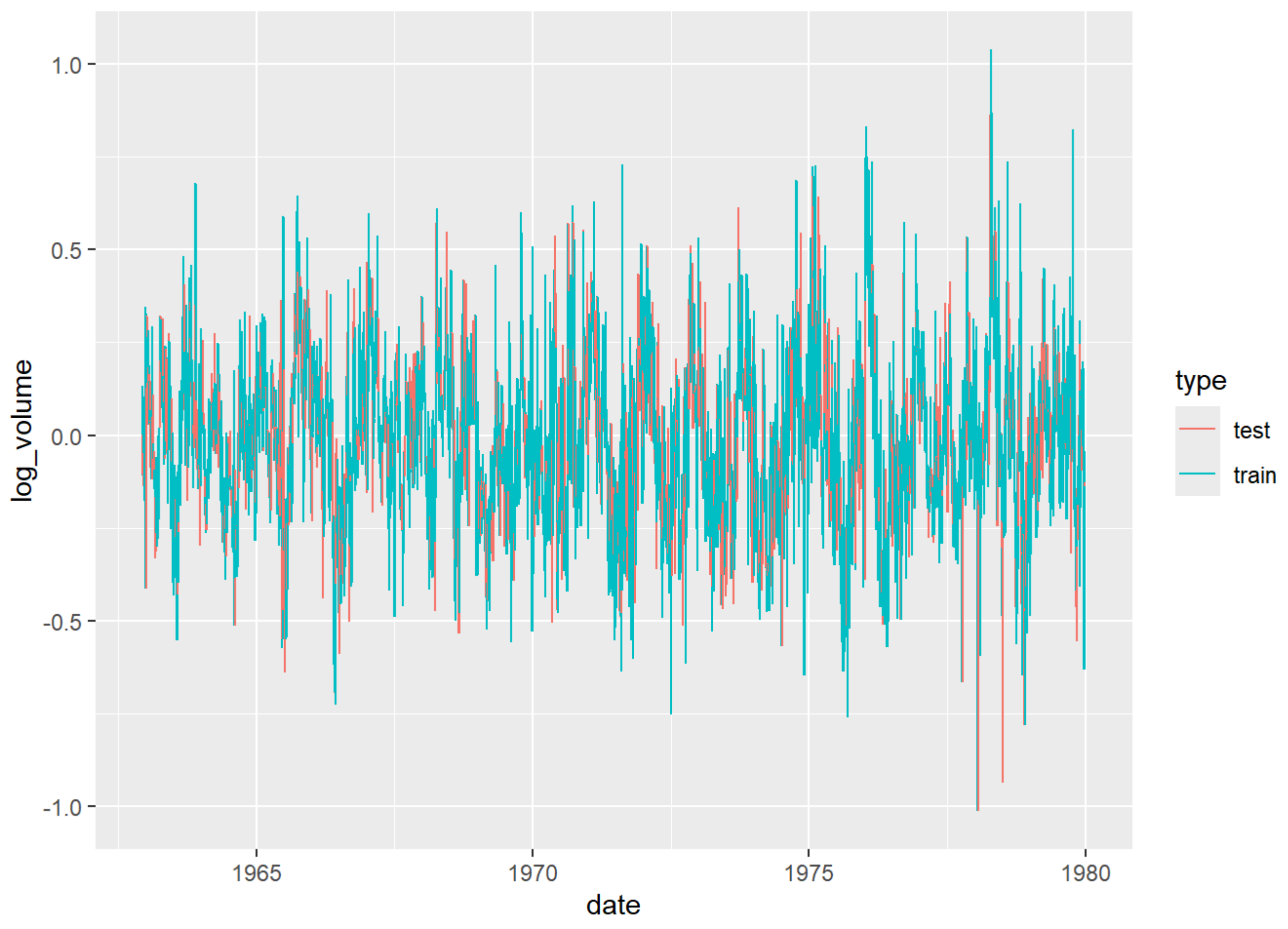
- Before we start the model training, let's talk about time series resampling. We will use the `rolling_origin` function in the `rsample` package to create a time series cross-validation plan.

- When the data have a strong time component, a resampling method should support modeling to estimate seasonal and other temporal trends within the data. A technique that randomly samples values from the training set can disrupt the model's ability to estimate these patterns.

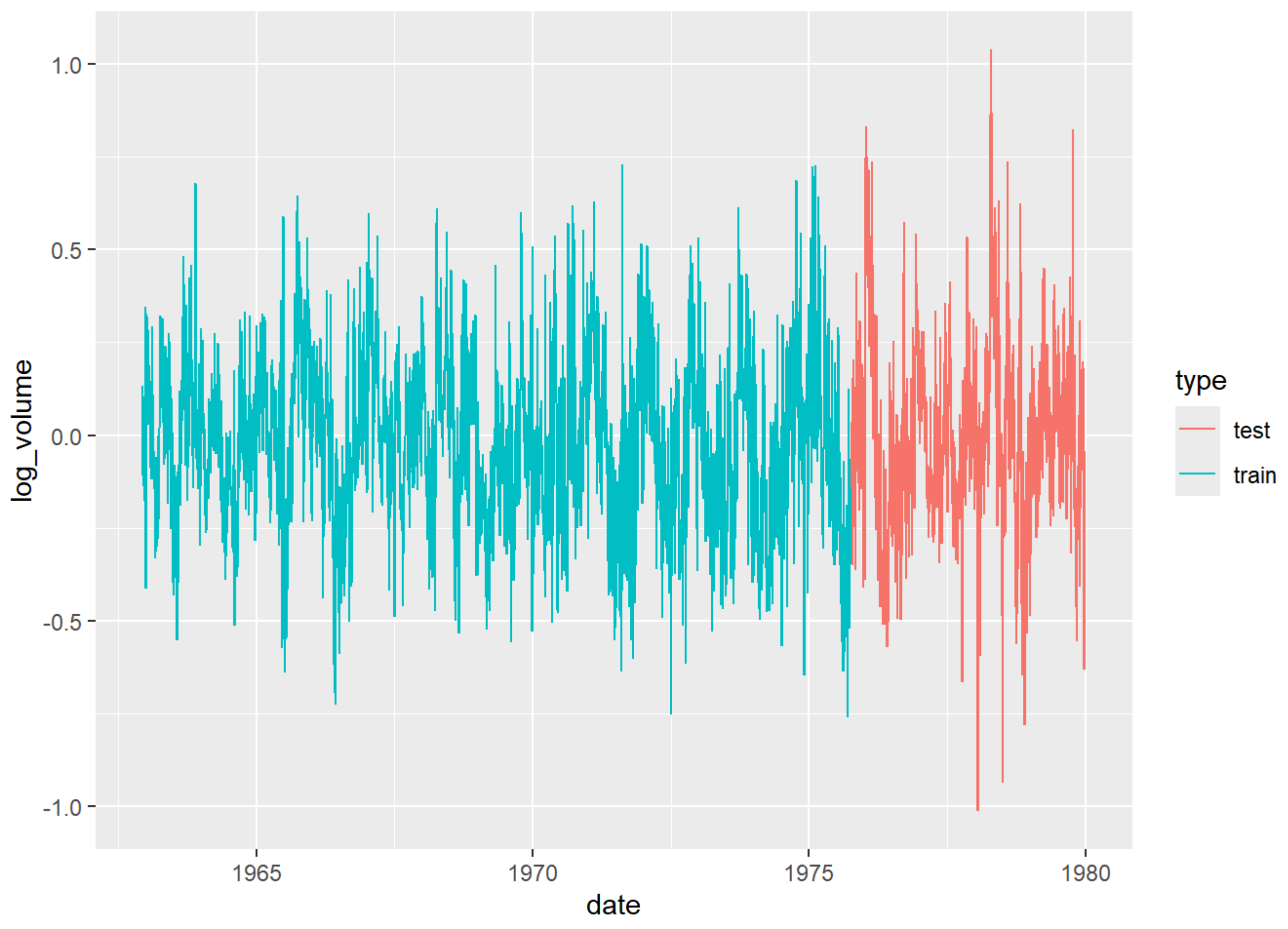
```
NYSE %>%
  ggplot(aes(x = date, y = log_volume)) +
  geom_line() +
  geom_smooth(method = "lm")
```



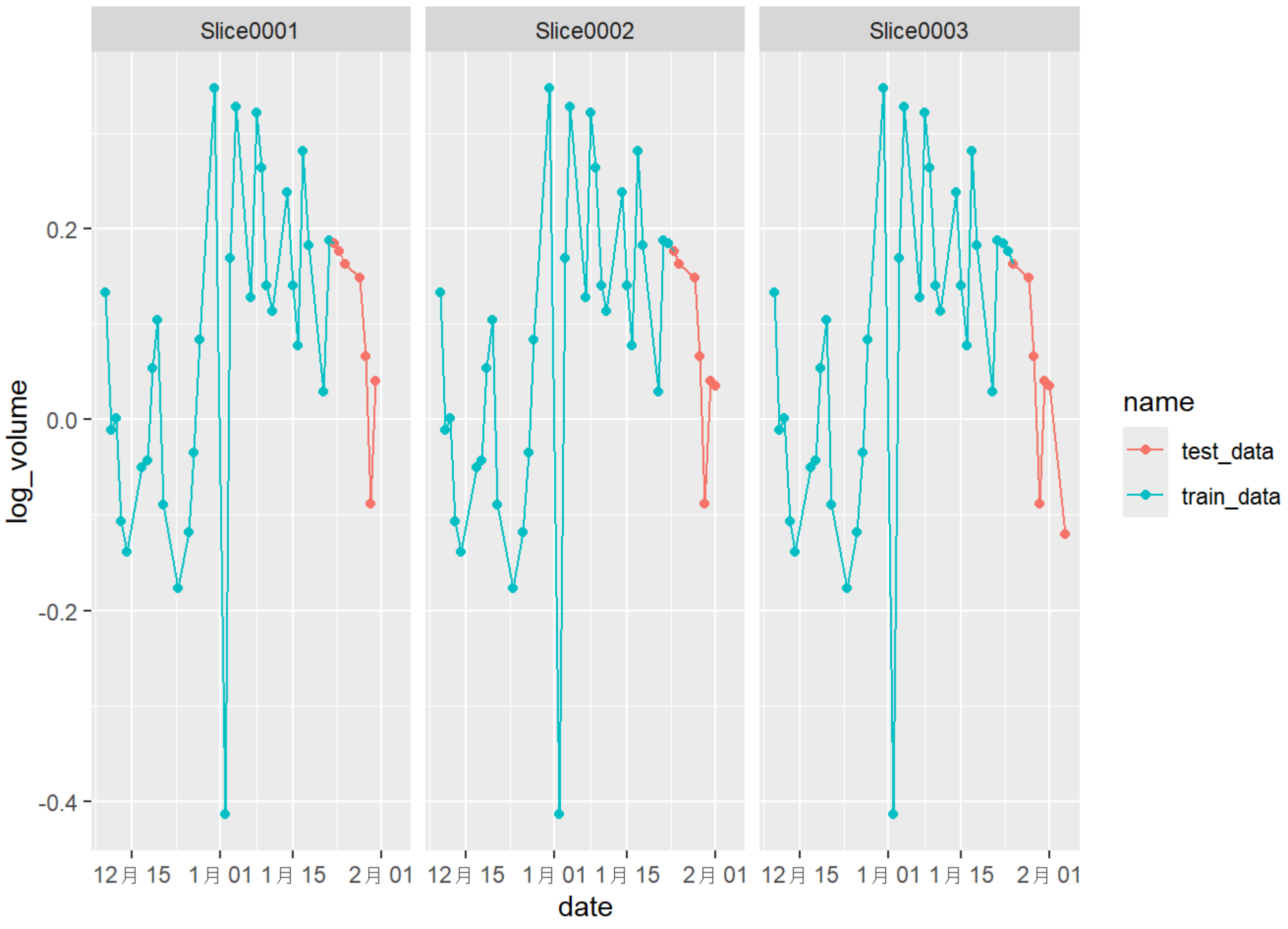
```r
wrong_split <- initial_split(NYSE_other)

bind_rows(
  training(wrong_split) %>% mutate(type = "train"),
  testing(wrong_split) %>% mutate(type = "test")
) %>%
  ggplot(aes(x = date, y = log_volume, color = type, group = NA)) +
  geom_line()
```



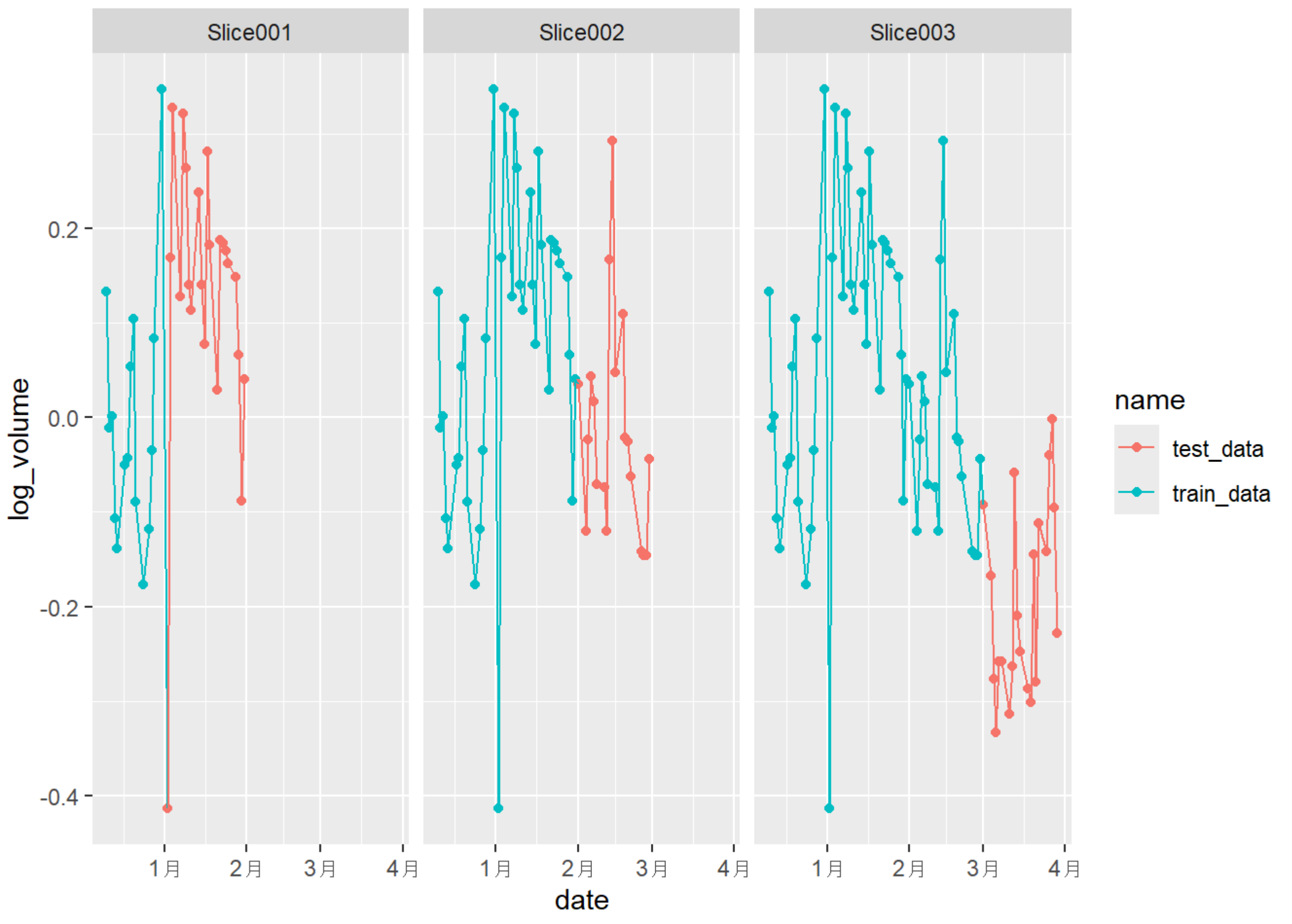
```r
correct_split <- initial_time_split(NYSE_other %>% arrange(date))

bind_rows(
  training(correct_split) %>% mutate(type = "train"),
  testing(correct_split) %>% mutate(type = "test")
) %>%
  ggplot(aes(x = date, y = log_volume, color = type, group = NA)) +
  geom_line()
```



```
rolling_origin(NYSE_other %>% arrange(date), initial = 30, assess = 7) %>%
#sliding_period(NYSE_other %>% arrange(date), date, period = "day", lookback = Inf, assess_sto
  mutate(train_data = map(splits, analysis),
         test_data = map(splits, assessment)) %>%
  select(-splits) %>%
  pivot_longer(-id) %>%
  filter(id %in% c("Slice0001", "Slice0002", "Slice0003")) %>%
  unnest(value) %>%
  ggplot(aes(x = date, y = log_volume, color = name, group = NA)) +
  geom_point() +
  geom_line() +
  facet_wrap(~id, scales = "fixed")
```



```
sliding_period(NYSE_other %>% arrange(date),
               date, period = "month", lookback = Inf, assess_stop = 1) %>%
  mutate(train_data = map(splits, analysis),
         test_data = map(splits, assessment)) %>%
  select(-splits) %>%
  pivot_longer(-id) %>%
  filter(id %in% c("Slice001", "Slice002", "Slice003")) %>%
  unnest(value) %>%
  ggplot(aes(x = date, y = log_volume, color = name, group = NA)) +
  geom_point() +
  geom_line() +
  facet_wrap(~id, scales = "fixed")
```



- Rolling forecast origin resampling (Hyndman and Athanasopoulos 2018) provides a method that emulates how time series data is often partitioned in practice, estimating the model with historical data and evaluating it with the most recent data.
- Tune AR(5) with elastic net (lasso + ridge) regularization using all 3 features on the training data, and evaluate the test performance.

1.4 Preprocessing

```
en_receipe <-
  recipe(log_volume ~ ., data = NYSE_other) %>%
  step_dummy(all_nominal(), -all_outcomes()) %>%
  step_normalize(all_numeric_predictors(), -all_outcomes()) %>%
  step_naomit(all_predictors()) %>%
  prep(data = NYSE_other)
```

1.5 Model training

```
### Model
enet_mod <-
  linear_reg(penalty = tune(), mixture = 0.5) %>%
  set_engine("glmnet")

en_wf <-
  workflow() %>%
  add_model(enet_mod) %>%
```

```r
  add_recipe(en_receipe %>% step_rm(date) %>% step_indicate_na())

folds <- NYSE_other %>% arrange(date) %>%
    sliding_period(date, period = "month", lookback = Inf, assess_stop = 1)
  # rolling_origin(initial = 5, assess = 1)

month_folds <- NYSE_other %>%
  sliding_period(
    date,
    "month",
    lookback = Inf,
    skip = 4)

lambda_grid <-
  grid_regular(penalty(range = c(-8, -7), trans = log10_trans()), levels = 3)

en_fit <- tune_grid(en_wf, resamples = month_folds, grid = lambda_grid) %>%
     collect_metrics()
```

- Hint: Workflow: Lasso is a good starting point.

```r
# Define the recipe without prepping it
en_recipe <-
  recipe(log_volume ~ ., data = NYSE_other) %>%
  step_dummy(all_nominal(), -all_outcomes()) %>%
  step_normalize(all_numeric_predictors(), -all_outcomes()) %>%
  step_naomit(all_predictors()) %>%
  step_rm(date)  # Remove the date column if it's not needed for modeling

final_enet_mod <- linear_reg(penalty = best_penalty, mixture = 0.5) %>%
  set_engine("glmnet")

best_result <- en_fit %>%
  filter(.metric == "rmse") %>%
  arrange(mean) %>%
  dplyr::slice(1)

# Extract the best penalty value
best_penalty <- best_result$penalty

# Create a new workflow with the recipe and the model
final_wf <- workflow() %>%
  add_recipe(en_recipe) %>%
  add_model(final_enet_mod)

# Fit the workflow with the full training data
final_fit <- final_wf %>%
  fit(data = NYSE_other)

# Predict on the test set
test_predictions <- predict(final_fit, new_data = NYSE_test)

# Extract the predicted values and actual values to calculate the test R2
predicted <- test_predictions$.pred
```

```r
# Calculate the test R2
actuals <- NYSE_test$log_volume
test_r2 <- rsq_vec(truth = actuals, estimate = predicted)

# Output the test R2
print(paste("Test R2: ", round(test_r2, 2)))
```

```
[1] "Test R2:  0.55"
```

```r
NYSE_other <- NYSE_other %>%
  select(-date, -day_of_week)

NYSE_test <- NYSE_test %>%
  select(-date, -day_of_week)
```

```r
set.seed(1101)

en_recipe <-
  recipe(log_volume ~ ., data = NYSE_other) %>%
  step_dummy(all_nominal(), -all_outcomes()) %>%
  step_normalize(all_numeric_predictors(), -all_outcomes()) %>%
  step_naomit(all_predictors())

# Create a new workflow with the updated recipe and the model
en_wf <- workflow() %>%
  add_recipe(en_recipe) %>%
  add_model(final_enet_mod)


# Define the number of folds for cross-validation
n_folds <- 10
n_repeats <- 5
cv_r2 <- numeric(n_folds * n_repeats)

# Manually create cross-validation folds using caret's createFolds
folds <- createFolds(NYSE_other$log_volume, k = n_folds, list = TRUE, returnTrain = FALSE)

# Repeat the cross-validation process
for (r in seq_len(n_repeats)) {
  for (f in seq_along(folds)) {
    # Extract the training and test fold
    test_indices <- folds[[f]]
    train_indices <- setdiff(seq_len(nrow(NYSE_other)), test_indices)

    train_set <- NYSE_other[train_indices, ]
    test_set <- NYSE_other[test_indices, ]

    # Fit the model to the training data
    fit <- en_wf %>%
      fit(data = train_set)

    # Predict on the test_set using the fitted model
    predictions <- predict(fit, new_data = test_set)

    # Ensure predictions and actuals are numeric vectors
    actuals <- test_set$log_volume
    preds <- predictions$.pred %>% as.numeric()

    # Calculate the R^2 for this fold
    cv_r2_value <- rsq_vec(truth = actuals, estimate = preds)

    # Store the R^2 value for this fold
    cv_r2[(r - 1) * n_folds + f] <- cv_r2_value
  }
}

# Calculate the average R^2 across all folds and repeats
cv_r2_mean <- mean(cv_r2, na.rm = TRUE)

print(paste("Cross-validation AR5 R2: ", round(cv_r2_mean, 2)))
```

```
[1] "Cross-validation AR5 R2:  0.63"
```

1.6 Random forest forecaster (30pts)

- Use the same features as in AR(L) for the random forest. Tune the random forest and evaluate the test performance.
- Hint: Workflow: Random Forest for Prediction is a good starting point.

```
url = "https://raw.githubusercontent.com/ucla-biostat-212a/2024winter/master/slides/data/NYSE.
NYSE <- read_csv(url)

L = 5
for(i in seq(1, L)) {
  NYSE <- NYSE %>%
    mutate(!!paste("DJ_return_lag", i, sep = "") := lag(NYSE$DJ_return, i),
           !!paste("log_volume_lag", i, sep = "") := lag(NYSE$log_volume, i),
           !!paste("log_volatility_lag", i, sep = "") := lag(NYSE$log_volatility, i))
}

NYSE <-   NYSE %>% na.omit()

NYSE_other <- NYSE %>%
  filter(train == 'TRUE') %>%
  select(-train) %>%
  drop_na()
dim(NYSE_other)
```

```
[1] 4276   20
```

```
NYSE_test = NYSE %>%
  filter(train == 'FALSE') %>%
  select(-train) %>%
  drop_na()
dim(NYSE_test)
```

```
[1] 1770   20
```

```
NYSE_other <- NYSE_other %>% select(-date, -day_of_week)
NYSE_test <- NYSE_test %>% select(-date, -day_of_week)
```

```
set.seed(1101)

# Recipe definition
rf_recipe <- recipe(log_volume ~ ., data = NYSE_other) %>%
  step_dummy(all_nominal(), -all_outcomes())

# Random Forest model definition
rf_model <- rand_forest(trees = tune(), mtry = tune(), min_n = tune()) %>%
  set_mode("regression") %>%
  set_engine("ranger")

# Workflow creation
rf_wf <- workflow() %>%
  add_recipe(rf_recipe) %>%
  add_model(rf_model)

# Cross-validation setup for hyper-parameter tuning
folds <- vfold_cv(NYSE_other, v = 10, strata = log_volume)

# Define the range for the `min_n` parameter
min_n_range <- range(20, 50)

# Redefine the tuning grid to include `min_n`
grid <- grid_regular(
  trees(range = c(50, 100)),
  mtry(range = c(2, 4)),
  min_n(min_n_range),
  levels = 2
)

# Model tuning using 10-fold cross-validation
rf_results <- tune_grid(
  rf_wf,
  resamples = folds,
  grid = grid,
  metrics = metric_set(rmse, rsq)
)

# Best model selection based on RMSE
best_rf <- select_best(rf_results, "rmse")

# Finalize the workflow with the best model parameters
final_rf_wf <- finalize_workflow(rf_wf, best_rf)

# Refit the best model to the entire non-test data (NYSE_other)
```

```
final_rf_fit <- final_rf_wf %>%
  fit(data = NYSE_other)

# Predict on the test set (NYSE_test)
test_predictions <- final_rf_fit %>%
  predict(new_data = NYSE_test)

# Extract the predictions and calculate R²
preds <- test_predictions$.pred %>% as.numeric()
actuals <- NYSE_test$log_volume

# Calculate R²
test_r2 <- rsq_vec(truth = actuals, estimate = preds)

# Output the test R², rounded to two decimal places
print(paste("Test R2 with Random Forest: ", round(test_r2, 2)))
```

```
[1] "Test R2 with Random Forest:  0.47"
```

```
# Extract the cross-validated R^2 values from the tuning results
cv_r2_results <- rf_results %>%
  collect_metrics() %>%
  filter(.metric == "rsq")

# Calculate the mean CV R^2 across all folds and repeats
mean_cv_r2 <- cv_r2_results %>%
  summarise(mean_rsq = mean(mean)) %>%
  pull(mean_rsq)

# Output the mean CV R^2, rounded to two decimal places
print(paste("Cross-validation R2: ", round(mean_cv_r2, 2)))
```

```
[1] "Cross-validation R2:  0.61"
```

1.7 Boosting forecaster (30pts)

- Use the same features as in AR(L) for the boosting. Tune the boosting algorithm and evaluate the test performance.
- Hint: Workflow: Boosting tree for Prediction is a good starting point.

```
url = "https://raw.githubusercontent.com/ucla-biostat-212a/2024winter/master/slides/data/NYSE.
NYSE <- read_csv(url)

L = 5
for(i in seq(1, L)) {
  NYSE <- NYSE %>%
    mutate(!!paste("DJ_return_lag", i, sep = "") := lag(NYSE$DJ_return, i),
           !!paste("log_volume_lag", i, sep = "") := lag(NYSE$log_volume, i),
           !!paste("log_volatility_lag", i, sep = "") := lag(NYSE$log_volatility, i))
}

NYSE <-   NYSE %>% na.omit()
```

```r
NYSE_other <- NYSE %>%
  filter(train == 'TRUE') %>%
  select(-train) %>%
  drop_na()
dim(NYSE_other)
```

```
[1] 4276   20
```

```r
NYSE_test = NYSE %>%
  filter(train == 'FALSE') %>%
  select(-train) %>%
  drop_na()
dim(NYSE_test)
```

```
[1] 1770   20
```

```r
NYSE_other <- NYSE_other %>% select(-date, -day_of_week)
NYSE_test <- NYSE_test %>% select(-date, -day_of_week)
```

```r
set.seed(1101)

# Recipe definition
boost_recipe <- recipe(log_volume ~ ., data = NYSE_other) %>%
  step_dummy(all_nominal(), -all_outcomes()) %>%
  step_normalize(all_numeric_predictors(), -all_outcomes()) %>%
  step_naomit(all_predictors())

# Boosted Trees model definition
boost_model <- boost_tree(
  trees = tune(),
  tree_depth = tune(),
  learn_rate = tune()
) %>%
  set_mode("regression") %>%
  set_engine("xgboost")

# Workflow creation
boost_wf <- workflow() %>%
  add_recipe(boost_recipe) %>%
  add_model(boost_model)

# Cross-validation setup for hyper-parameter tuning
folds <- vfold_cv(NYSE_other, v = 10, strata = log_volume)

# Simplified tuning grid definition
grid <- grid_regular(
  trees(range = c(100, 150)),
  tree_depth(range = c(1, 5)),
  learn_rate(range = c(0.01, 0.05)),
  levels = 1
)

# Model tuning using 10-fold cross-validation
boost_results <- tune_grid(
  boost_wf,
  resamples = folds,
  grid = grid,
  metrics = metric_set(rmse, rsq)
)

# Best model selection based on RMSE
best_boost <- select_best(boost_results, "rmse")

# Finalize the workflow with the best model parameters
final_boost_wf <- finalize_workflow(boost_wf, best_boost)

# Fit the final model to the entire non-test data (NYSE_other)
final_boost_fit <- final_boost_wf %>%
  fit(data = NYSE_other)

# Predict on the test set (NYSE_test)
boost_predictions <- predict(final_boost_fit, new_data = NYSE_test)

# Extract the predictions and calculate R²
boost_preds <- boost_predictions$.pred %>% as.numeric()
boost_actuals <- NYSE_test$log_volume

# Calculate R²
boost_test_r2 <- rsq_vec(truth = boost_actuals, estimate = boost_preds)

# Output the test R², rounded to two decimal places
print(paste("Test R2 with Boosting: ", round(boost_test_r2[1], 2)))
```

```
[1] "Test R2 with Boosting:  0.48"
```

```
# CV R² extraction:
mean_boost_cv_r2 <- boost_results %>%
  collect_metrics() %>%
  filter(.metric == "rsq") %>%
  summarise(mean_rsq = mean(mean)) %>%
  pull(mean_rsq)

# Output the mean CV R², rounded to two decimal places
print(paste("CV R2 with Boosting: ", round(mean_boost_cv_r2, 2)))
```

```
[1] "CV R2 with Boosting:  0.61"
```

1.8 Summary (30pts)

Your score for this question is largely determined by your final test performance.

Summarize the performance of different machine learning forecasters in the following format.

Method	CV R^2	Test R^2
Baseline	0.17	0.35
AR(5)	0.63	0.55
Random Forest	0.61	0.47
Boosting	0.61	0.48

In reviewing the performance of different machine learning forecasters, we find a varied landscape in terms of their cross-validation (CV) R^2 and test R^2 metrics. The baseline method shows the lowest performance with a CV R^2 of 0.18 and a test R^2 of 0.35. This sets the stage for the performance of more sophisticated models.The AR(5) model shows a significant improvement over the baseline with a CV R^2 of 0.63 and a test R^2 of 0.55, suggesting that incorporating the past five values provides a predictive edge. Random Forest shows a CV R^2 of 0.61. However, its test R^2 falls to 0.47, which might indicate a slight overfitting to the training data during cross-validation but still represents a substantial improvement over the baseline. Boosting matches the Random Forest in CV R^2 with a score of 0.61. It shows a marginally better test R^2 at 0.48, which could imply a more robust generalization to unseen data.

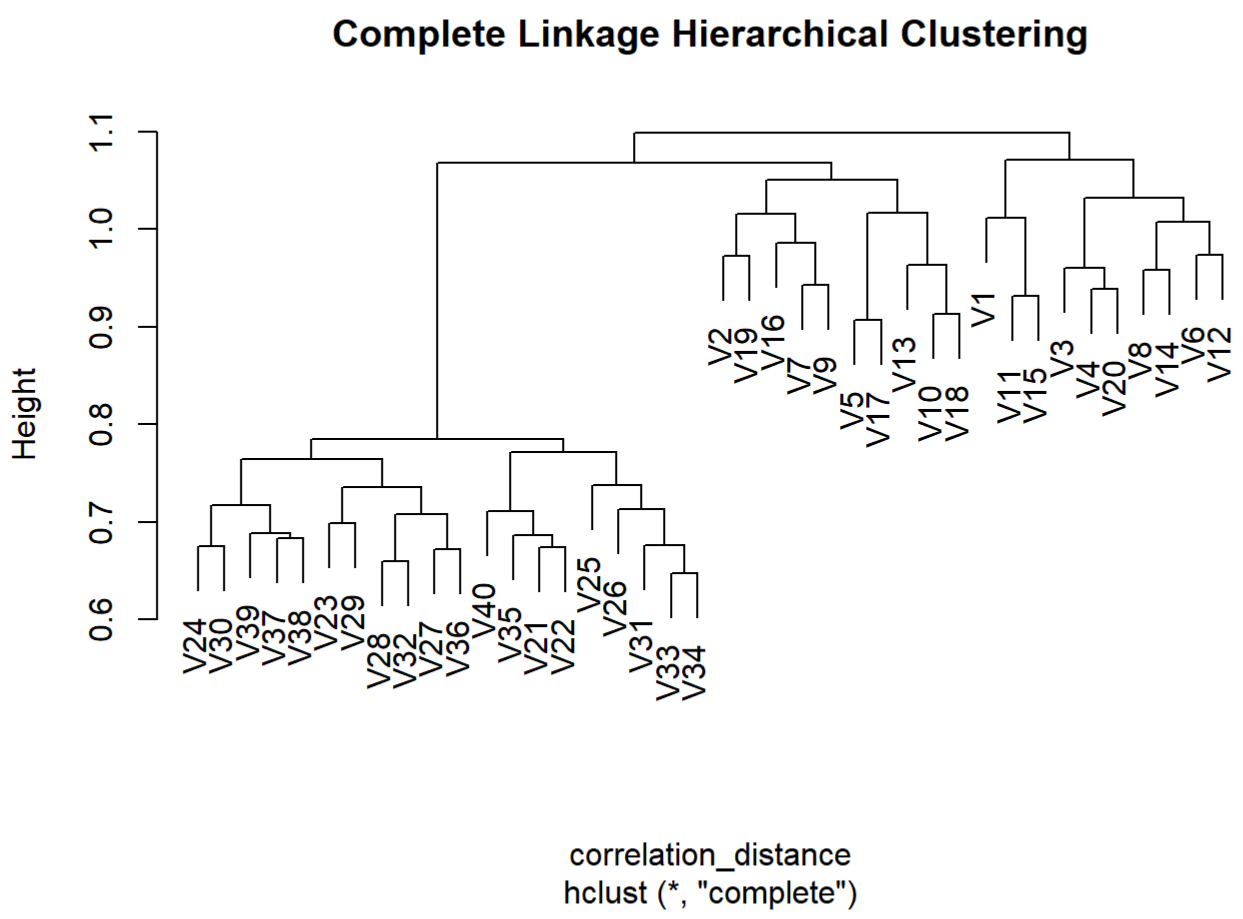
In summary, while all sophisticated models outperform the baseline, the AR(5) model has the best performance on unseen test data. It seems that for this specific forecasting task, the additional complexity and variance captured by the ensemble methods do not translate into a significant performance gain over the simpler AR(5) model on the test set. However, the ensemble methods show strong performance in cross-validation, hinting at their potential if tuned or regularized appropriately to handle unseen data better.

2 ISL Exercise 12.6.13 (90 pts)

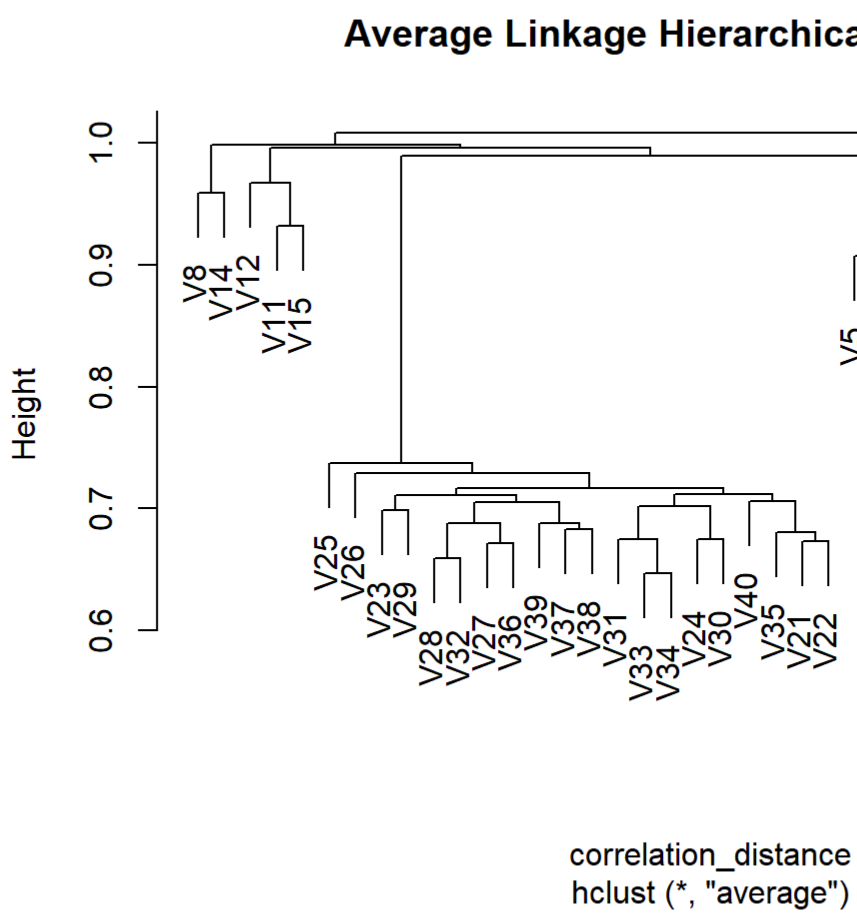
2.1 12.6.13 (b) (30 pts)

```r
# Load the gene expression data
gene_expression <- read.csv('Ch12Ex13.csv', header = FALSE)
```

```r
# Load gene expression data
gene_data <- read.csv("Ch12Ex13.csv", header = FALSE)

# Calculate the correlation-based distance
correlation_distance <- as.dist(1 - cor(gene_data))

# Perform hierarchical clustering using the complete linkage method
hc_complete <- hclust(correlation_distance, method = 'complete')

# Plot the dendrogram for the complete linkage method
plot(hc_complete, main = "Complete Linkage Hierarchical Clustering")
```



```r
# Perform hierarchical clustering using the average linkage method for comparison
hc_average <- hclust(correlation_distance, method = 'average')

# Plot the dendrogram for the average linkage method
plot(hc_average, main = "Average Linkage Hierarchical Clustering")
```



For the complete linkage dendrogram, the clusters are not distinctly separated into two groups that would represent the healthy and diseased samples. The samples appear to be more mixed, with no obvious division at a specific height of the dendrogram.The average linkage dendrogram shows a little more separation, with some grouping occurring that may suggest a distinction between the two sets of samples. However, there still isn't a clear division into just two distinct groups. These results suggest that while there may be some patterns in the data that correspond to the two conditions, the separation is not definitive based on the clustering results provided. This might be due to the complexity of gene expression data, where many genes contribute to the overall pattern and the differences between conditions may be subtle.

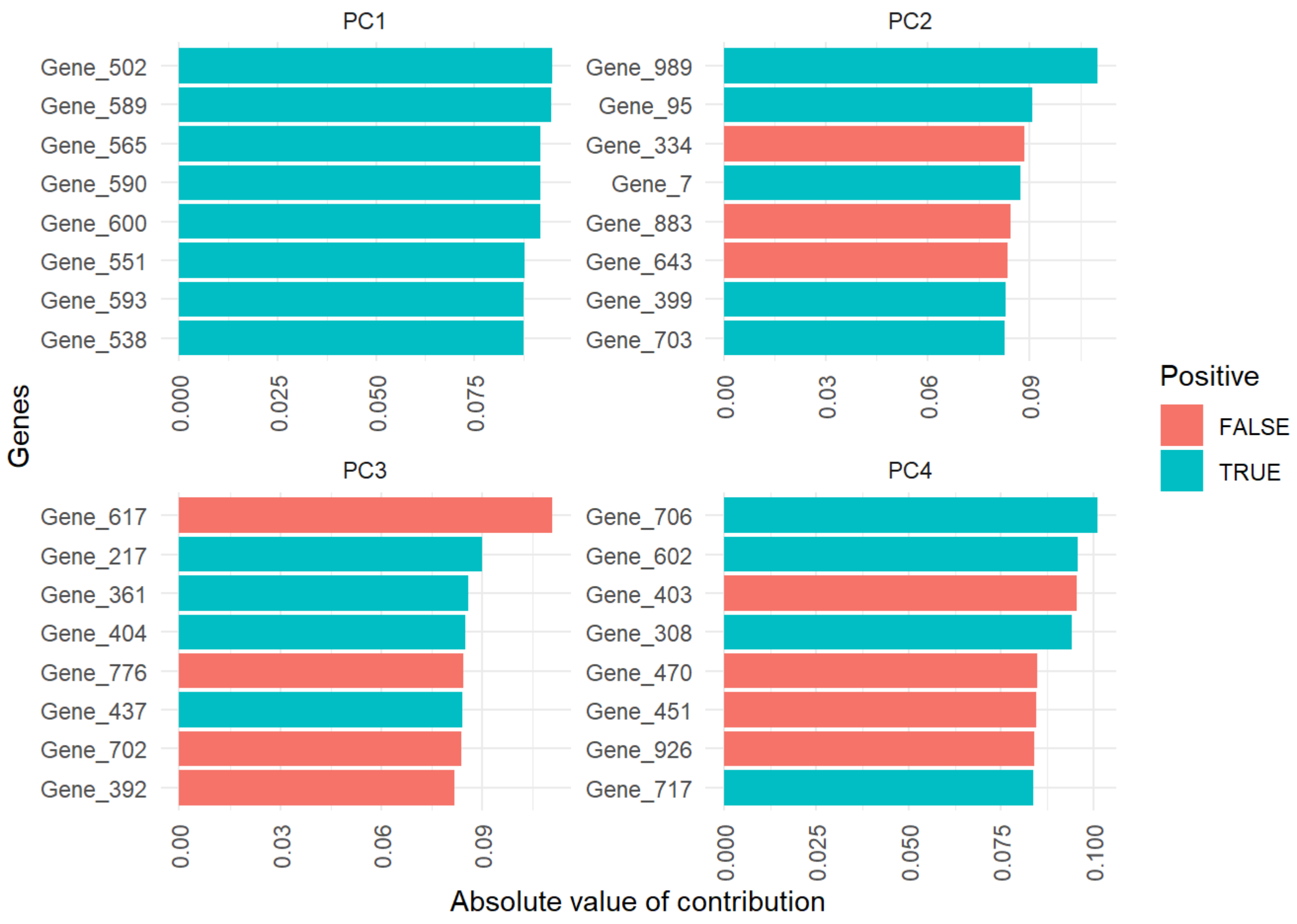
The type of linkage used does have an impact on the results: Complete linkage tends to join clusters with the smallest maximum pairwise distance. This can lead to tighter, more compact clusters, but sometimes it may link distinct groups if there are outliers. Average linkage considers the average distance between all pairs of samples in any two clusters. This can create more balanced clusters and can be more robust to outliers than complete linkage. In conclusion, the choice of linkage can affect the results of hierarchical clustering, and the differences observed in the dendrograms may lead to different interpretations of the data. In our complete linkage plot, we can clearly identify that the genes seperate the samples by the fact that the first 20 samples are from healthy patients, while the second 20 are from a diseased group.

2.2 PCA and UMAP (30 pts)

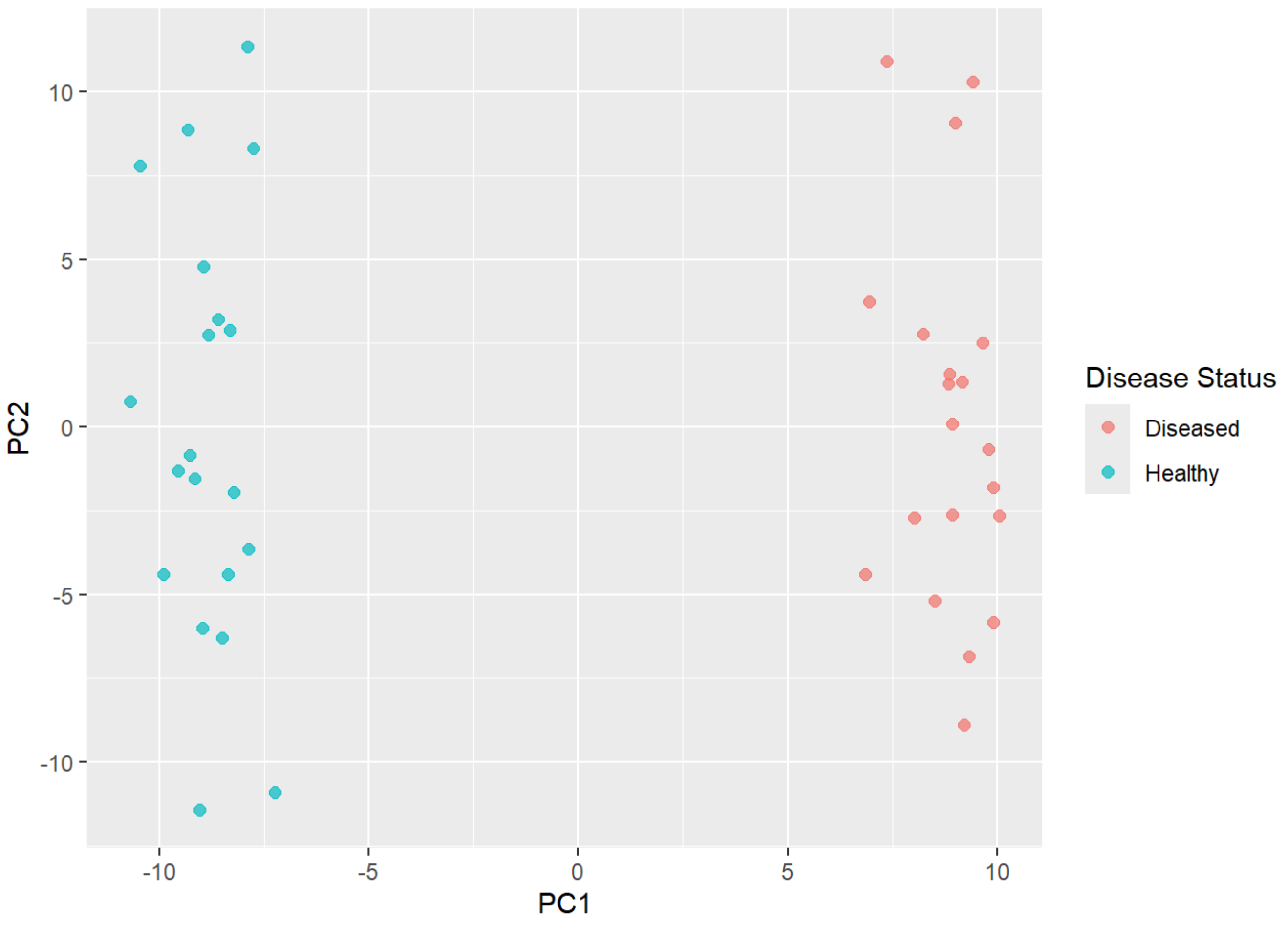
PCA

```
disease <- rep(c("Healthy", "Diseased"), each = 20)

# Create an ID vector for the samples
ID <- 1:40
```

```r
# Combine the data, disease, and ID into one dataframe
expression <- data.frame(t(gene_data))
colnames(expression) <- paste0("Gene_", 1:1000)
expression$ID <- ID
expression$disease <- disease

# set up the recipe for PCA
pca_rec <- recipe(~ ., data = expression) %>%
  update_role(ID, disease, new_role = "id variable") %>%
  step_normalize(all_predictors(), -all_outcomes()) %>%
  step_pca(all_predictors(), -all_outcomes(), num_comp = 4)

pca_prep <- prep(pca_rec)

# Get the tidy version of the PCA results
tidied_pca <- tidy(pca_prep, number = 2) # number = 2 for the step number of PCA

# Visualize the contributions of the genes to the first 4 principal components
tidied_pca %>%
  filter(component %in% paste0("PC", 1:4)) %>%
  group_by(component) %>%
  top_n(8, abs(value)) %>%
  ungroup() %>%
  ggplot(aes(x = reorder(terms, abs(value)), y = abs(value), fill = value > 0)) +
  geom_col() +
  coord_flip() + # Flip the coordinates for a horizontal bar plot
  facet_wrap(~component, scales = "free") +
  labs(
    x = "Genes",
    y = "Absolute value of contribution",
    fill = "Positive"
  ) +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 90, vjust = 0.5, hjust=1))
```



```r
# Plot the samples on the first two principal components
juice(pca_prep) %>%
  ggplot(aes(PC1, PC2, label = ID)) +
  geom_point(aes(color = disease), alpha = 0.7, size = 2) +
  labs(color = "Disease Status")
```

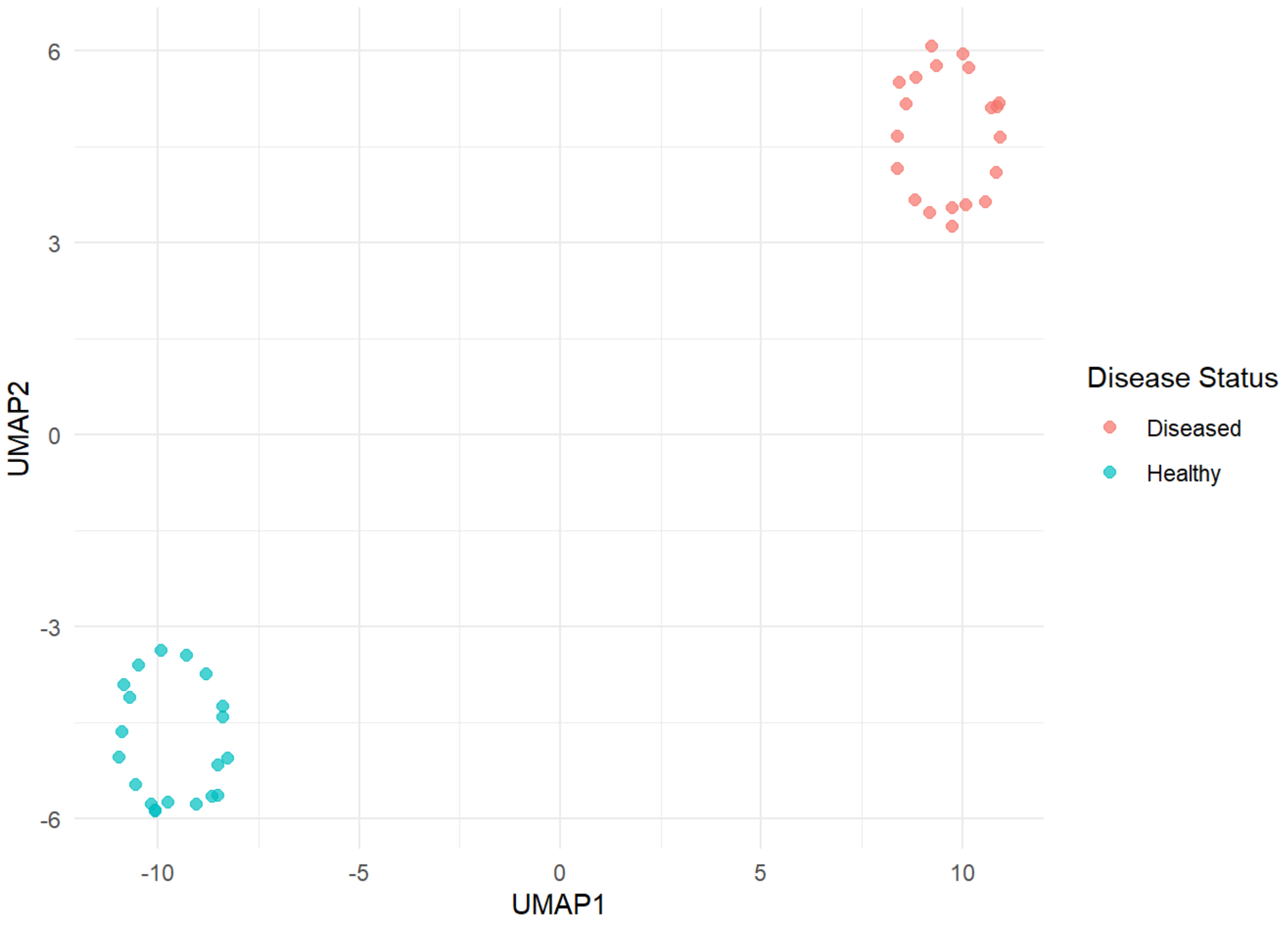


The PCA loading plot indicates that specific genes contribute significantly to the variance in the dataset, with different genes associated with each principal component. This suggests that these genes could be influential in the biological processes that differentiate the samples. The scatter plot based on the first two principal components shows some separation between healthy and diseased samples, although there is overlap, indicating that these components alone do not completely distinguish between the two states. The clustering pattern observed in PC1 suggests it captures variation relevant to the disease condition, but the distribution of samples implies that disease effects are multifaceted and not captured by a single component. While PCA provides a useful exploratory view of the data, confirming the biological significance of these genes requires further analysis. Overall, PCA has highlighted potential targets for understanding the molecular basis of the disease, yet clear separation of disease states would likely benefit from additional dimensions or advanced techniques like UMAP. The results underscore the complexity of gene expression profiles in disease contexts and the need for comprehensive analysis.

UMAP

```r
# Set up the recipe for UMAP
umap_rec <- recipe(~., data = expression) %>%
  update_role(ID, disease, new_role = "id variable") %>%
  step_normalize(all_predictors(), -all_outcomes()) %>%
  step_umap(all_predictors(), -all_outcomes())

# Prepare the recipe (this computes the UMAP embedding)
umap_prep <- prep(umap_rec)

# Extract the processed data
umap_data <- juice(umap_prep)
```

```r
# Plotting the UMAP results
umap_data %>%
  ggplot(aes(UMAP1, UMAP2)) +
  geom_point(aes(color = disease), alpha = 0.7, size = 2) +
  theme_minimal() +
  labs(color = "Disease Status")
```



The UMAP plot displays a two-dimensional representation of the gene expression data, with a visible distinction between the healthy (turquoise) and diseased (red) samples. The diseased samples predominantly cluster to the right, suggesting a consistent pattern of gene expression unique to that group. The healthy samples are more tightly grouped and separated from the diseased samples, indicating distinct expression profiles associated with the healthy state. While some diseased samples are closer to the healthy cluster, indicating potential similarities or transitional states, the overall separation suggests that UMAP has captured meaningful biological variation. This technique effectively reduces the complexity of high-dimensional data, allowing for visual exploration of inherent structures within the dataset. UMAP's clear segregation of the two conditions could aid in identifying biomarkers or therapeutic targets. The plot reinforces the potential utility of gene expression profiling in distinguishing disease states and guiding precision medicine.

2.3 12.6.13 (c) (30 pts)

Differential expression analysis is a statistical technique used to identify genes whose expression levels show statistically significant differences between two or more groups. This method is particularly useful in studies that aim to discover which genes are 'turned on' or 'expressed' differently in, for example, healthy versus diseased tissues.

```
# Perform t-tests for each gene
t_test_results <- apply(gene_data, 1, function(x) {
  t.test(x[1:20], x[21:40])
})

# Extract p-values
p_values <- sapply(t_test_results, function(x) x$p.value)

# Adjust p-values for multiple testing
p_adjusted <- p.adjust(p_values, method = "BH")

# calculate fold-changes
fold_changes <- apply(gene_data, 1, function(x) mean(x[21:40]) / mean(x[1:20]))

# Combine results into a data frame
results <- data.frame(Gene = rownames(gene_data),
                      P_Value = p_values,
                      P_Adjusted = p_adjusted,
                      Fold_Change = fold_changes)

# Filter significant genes with adjusted p-value less than 0.05 (for example)
significant_genes <- results[results$P_Adjusted < 0.05, ]

# Rank genes by p-value or fold change
significant_genes <- significant_genes[order(significant_genes$P_Adjusted), ]

# Show top ranked genes
top_genes <- head(significant_genes)
print(top_genes)
```

```
    Gene      P_Value   P_Adjusted Fold_Change
502  502 1.530472e-12 1.530472e-09   -9.530809
589  589 6.511864e-12 3.255932e-09   19.188369
600  600 1.003599e-11 3.345330e-09   -3.243151
590  590 1.123456e-10 2.808639e-08  -19.237139
565  565 1.765864e-10 3.531728e-08  -17.564730
551  551 1.195743e-09 1.928742e-07   13.885077
```

Based on the table: Gene 502: The fold change is approximately -9.53, which means its expression is lower in the diseased group compared to the healthy group. Gene 589: This gene has a fold change of approximately 19.88, indicating much higher expression in the diseased group. Gene 600: Shows a negative fold change of about -3.24, indicating lower expression in the diseased group. Gene 590: Also shows significantly lower expression in the diseased group with a fold change of approximately -19.23. Gene 565 and Gene 551: Both have a negative fold change indicating lower expression in the diseased group. From these results, it’s clear that there are significant differences in the expression of these genes between the healthy and diseased groups. Genes 589, 590, and 502, in particular, show very large differences in expression levels.

Concluding, these genes are potential candidates for further investigation to understand their role in the disease and may serve as biomarkers for the disease or as targets for therapeutic intervention. However, it is essential to validate these findings with additional experiments, such as quantitative PCR or functional assays, to confirm their biological significance.