GPR2

Binghong Chen

2024-04-19

We selected the 30 stocks from Dow Jones Indices to be our target stocks.

Data processing

Import data from the excel file of data downloaded from Bloomberg.

```r
set.seed(423)
excel_file = "data.xlsx"
sheets = excel_sheets(excel_file)
data_list = list()
for (sheet in sheets) {
  data_list[[sheet]] = read_excel(excel_file, sheet = sheet, skip = 6)
}
```

Then try to delete the empty columns and the duplicated date columns

```r
for (sheet in names(data_list)) {
  df = data_list[[sheet]][, c(1, seq(2, 59, 3))]

  # Change date column name
  colnames(df)[1] = "date"

  # Change the date column from dttm to date format
  df$date = as.POSIXct(df$date, origin="1980-01-30")
  df$date = as.Date(df$date)

  data_list[[sheet]] = df
}
```

Note that all the values in the data_list is stored as character. We need to change it to numeric.

```r
for (sheet in names(data_list)) {
  df = data_list[[sheet]]

  df = df %>%
    mutate(across(-1, as.numeric))

  data_list[[sheet]] = df
}
```

Then we try to calculate the momentum data from price.

Momentum calculation function. An note for each variable:

`mom1m`: 1-month momentum `mom12m`: 12-month momentum `mom36m`: 36-month momentum `chmom`: change in 6-month momentum `R1M_Usd`: Future return, our dependent variable.

```
mom = function(df) {
  price = df$`Last Price`
  mom1m = c(diff(price, lag = 1), NA)
  chmom = c(diff(diff(price, lag = 6)), rep(NA, 7))
  mom12m = c(diff(price, lag = 12), rep(NA, 12))
  mom36m = c(diff(price, lag = 36), rep(NA, 36))
  R1M_Usd = c(NA, diff(price, lag = 1)) / price
  return(data.frame(mom1m, mom12m, mom36m, chmom, R1M_Usd))
}
```

Calculate `betasq` (beta squared) from beta.

```
btsq = function(df) {
  beta = df$`Overridable Adjusted Beta`
  betasq = beta^2
  return(data.frame(betasq))
}
```

Bid ask spread.

```
bid_ask = function(df) {
  bid = df$`Bid Price`
  ask = df$`Ask Price`
  baspread = df$`Average Bid Ask Spread`
  baspread[is.na(baspread)] = (bid - ask)[is.na(baspread)]
  return(baspread)
}
```

Then apply the change to each spread sheet.

```
for (sheet in names(data_list)) {
  df = data_list[[sheet]]

  # Add momentum and beta-squared to df
  df = cbind(df, mom(df), btsq(df))

  # Change the bid ask
  df$`Average Bid Ask Spread` = bid_ask(df)

  data_list[[sheet]] = df
}
```

Then we remove the columns with all NA value that failed in downloading from Bloomberg.

```
remove_list =
  c("Price Change 1 Month Percent", "Relative Share Price Momentum",
```

```r
    "Research and Development - COGS", "Bid Price", "Ask Price",
    "Current Shares Outstanding...53")

for (sheet in names(data_list)) {
  df = data_list[[sheet]]

  df = select(df, -all_of(remove_list))

  data_list[[sheet]] = df
}
```

Then we add the columns of stock id, and combine all of the sheets into one data frame.

```r
# Remove the " US Equity" from the sheet name
names(data_list) = gsub(" US Equity", "", names(data_list))

# Add stock id
for (sheet in names(data_list)) {
  df = data_list[[sheet]]

  df = df %>%
    mutate(ID = sheet)

  data_list[[sheet]] = df
}

# Get the new data frame
data = bind_rows(data_list)
```

Use a function to remove the characters in column names.

```r
clean_column_names <- function(names) {
  names <- gsub("`", "", names)
  names <- gsub("\\\\", "", names)
  names <- gsub("\\s+", "", names)
  names <- trimws(names)
  names
}

# Apply to column names
colnames(data) <- clean_column_names(colnames(data))
```

```r
# Add market capitalization and covid variable
data = data %>%
  mutate(mcap = data$CurrentSharesOutstanding * data$LastPrice) %>%
  mutate(covid = as.numeric(date >= "2020-01-01")) %>%
  mutate(ID = as.factor(ID))

# Change the factor to dummy variable for regression
data_dum = as.data.frame(model.matrix(~ . - 1, data = data))
```

Then we try to rename the columns based on the factor investing style.

We try to change the data to rectangle by removing all the rows with NA.

```r
data_cleaned = na.omit(data_dum)
```

We found that some of the dummy variable was opt out to be all-0-vector since all of the data from that company include `NA`. So we exclude those variables.

```r
data_cleaned <- data_cleaned[, apply(data_cleaned, 2, function(x) sum(x) != 0)]
```

Normalization

We normalize our data for training efficiency.

```r
# min-max scaling
normalize_minmax <- function(x) {
  return((x - min(x))/(max(x) - min(x)))
}

# Normalize the data except the date column
data_normalized = cbind(data_cleaned[1], apply(data_cleaned[-1], 2, normalize_minmax))

# Mark the max and min return for future calculation
retmax = max(data_cleaned$R1M_Usd)
retmin = min(data_cleaned$R1M_Usd)
```

Training, validation and testing data

We use the last 12 month as test set, and the 12 month before that as validation set.

So training set time range: 2013/08/30 - 2022/03/31, validation set time range: 2022/04/01 - 2023/03/31, Testing set time range: 2023/04/01 - 2024/03/29

```r
not_x_list = c("date", "LastPrice", "R1M_Usd")

tv_break = as.Date("2022-04-01")
vt_break = as.Date("2023-04-01")

# Training set
x_training = data_normalized %>%
  filter(date <= tv_break) %>%
  select(-all_of(not_x_list))

y_training = data_normalized %>%
  filter(date <= tv_break) %>%
  select("R1M_Usd")

training_set = cbind(y_training, x_training)

# Validation set
x_validation = data_normalized %>%
  filter(date > tv_break) %>%
  filter(date <= vt_break) %>%
  select(-all_of(not_x_list))
```

```r
y_validation = data_normalized %>%
  filter(date > tv_break) %>%
  filter(date <= vt_break) %>%
  select("R1M_Usd")

validation_set = cbind(y_validation, x_validation)

# Training & validation
x_tv = data_normalized %>%
  filter(date <= vt_break) %>%
  select(-all_of(not_x_list))

y_tv = data_normalized %>%
  filter(date <= vt_break) %>%
  select("R1M_Usd")

tv_set = cbind(y_tv, x_tv)

# Testing set
x_testing = data_normalized %>%
  filter(date > vt_break) %>%
  select(-all_of(not_x_list))

y_testing = data_normalized %>%
  filter(date > vt_break) %>%
  select("R1M_Usd")

testing_set = cbind(y_testing, x_testing)
```

Model Training

We will try these models:

1. Ordinary Least Squares (OLS)
2. Generalized linear model with group LASSO (GLM)
3. Random Forest (RF)
4. Neural network architectures with one to five layers (NN1, …, NN5)

OLS

Note that the financial data do have multi-collinearity, so we use VIF to do the variable selection.

```r
ols_formula = as.formula("R1M_Usd ~ .")
ols = lm(ols_formula, data = tv_set)

summary(ols)
```

```
##
## Call:
## lm(formula = ols_formula, data = tv_set)
```

```
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.47323 -0.04164  0.00091  0.04147  0.50257
##
## Coefficients: (1 not defined because of singularities)
##                                  Estimate Std. Error t value Pr(>|t|)
## (Intercept)                     0.3541325  0.0589825   6.004 2.19e-09 ***
## `\\`Turnover/TradedValue\\``     0.0664349  0.0423346   1.569 0.116703
## CurrentSharesOutstanding        0.1470138  0.0730857   2.012 0.044373 *
## AverageBidAskSpread            -0.0040666  0.0248714  -0.164 0.870134
## Volatility30Day                -0.1252794  0.0256663  -4.881 1.12e-06 ***
## OverridableAdjustedBeta        -0.0039166  0.0651927  -0.060 0.952099
## `\\`PriceEarningsRatio(P/E)\\`` -0.0067890  0.0398264  -0.170 0.864657
## Revenue                        -0.0447388  0.0528832  -0.846 0.397634
## `\\`Assets-1YearGrowth\\``       0.0577523  0.0422426   1.367 0.171694
## BasicEarningsperShare           0.0720150  0.0520658   1.383 0.166736
## PricetoBookRatio                0.0516417  0.0196601   2.627 0.008672 **
## NetWorkingCapitalInvestment     0.0136361  0.0190016   0.718 0.473049
## FinancialLeverage              -0.0059027  0.0324659  -0.182 0.855745
## GrossProfit                     0.0567930  0.0838233   0.678 0.498128
## mom1m                           0.0668192  0.0468687   1.426 0.154084
## mom12m                         -0.0084407  0.0280797  -0.301 0.763745
## mom36m                         -0.1168615  0.0329456  -3.547 0.000396 ***
## chmom                           0.1568741  0.0405804   3.866 0.000113 ***
## betasq                         -0.0008416  0.0585041  -0.014 0.988523
## IDAAPL                         -0.1066777  0.0376559  -2.833 0.004647 **
## IDAMGN                          0.0324697  0.0324594   1.000 0.317249
## IDAMZN                         -0.0436403  0.0181202  -2.408 0.016093 *
## IDBA                            0.0163619  0.0330728   0.495 0.620837
## IDCAT                           0.0396680  0.0319704   1.241 0.214802
## IDCRM                           0.0289631  0.0328133   0.883 0.377498
## IDCSCO                          0.0176261  0.0218140   0.808 0.419154
## IDCVX                           0.0445510  0.0264602   1.684 0.092359 .
## IDDIS                           0.0346349  0.0283870   1.220 0.222538
## IDDOW                           0.0348040  0.0450883   0.772 0.440239
## IDHD                            0.0149535  0.0299142   0.500 0.617201
## IDHON                           0.0333834  0.0317301   1.052 0.292848
## IDIBM                           0.0518309  0.0302024   1.716 0.086259 .
## IDINTC                          0.0255611  0.0219593   1.164 0.244522
## IDJNJ                           0.0203152  0.0258260   0.787 0.431575
## IDKO                            0.0196745  0.0222025   0.886 0.375625
## IDMCD                           0.0243601  0.0313476   0.777 0.437173
## IDMMM                           0.0477135  0.0322540   1.479 0.139179
## IDMRK                           0.0246727  0.0259894   0.949 0.342538
## IDMSFT                         -0.0339114  0.0174904  -1.939 0.052627 .
## IDNKE                           0.0315249  0.0298916   1.055 0.291687
## IDPG                            0.0240991  0.0255801   0.942 0.346228
## IDUNH                           0.0100332  0.0304483   0.330 0.741792
## IDVZ                            0.0268224  0.0238679   1.124 0.261208
## IDWMT                                  NA         NA      NA       NA
## mcap                           -0.0089742  0.0426796  -0.210 0.833475
## covid                           0.0066140  0.0044069   1.501 0.133524
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.07341 on 2609 degrees of freedom
## Multiple R-squared:  0.04683,    Adjusted R-squared:  0.03075
## F-statistic: 2.913 on 44 and 2609 DF,  p-value: 7.401e-10
```

We can see that the coefficient for `IDWMT` is NA due to perfect collinearity we have while setting up the dummy variable, so we remove it mannually.

```r
formula_text = "R1M_Usd ~ . - IDWMT"
ols_formula = as.formula(formula_text)
ols = lm(ols_formula, data = tv_set)

ols_vif = vif(ols)

while(any(ols_vif > 10)) {
  max_vif_var = names(which.max(ols_vif))

  # Remove the variable with max vif
  formula_text = paste(formula_text, "-", max_vif_var)
  ols_formula = as.formula(formula_text)

  # Fit the model again
  ols = lm(ols_formula, data = tv_set)

  # Calculate the new vif
  ols_vif = vif(ols)
}
```

LASSO

Here we use the 10 fold cross validation to get the best lambda.

```r
# Lambda
Log.Lambda.Seq = seq(-7, 3, by = 0.1)
Lambda.Seq = c(0, exp(Log.Lambda.Seq))

# Training
lasso_fit =
  cv.glmnet(
    x = as.matrix(x_training),
    y = as.vector(y_training$R1M_Usd),
    lambda = Lambda.Seq,
    alpha = 1)

lasso_fit =
  glmnet(
    x = as.matrix(x_training),
    y = as.vector(y_training$R1M_Usd),
    lambda = lasso_fit$lambda.min,
    alpha = 1)
```

Random Forest (RF)

For RF, we use the `caret` package to help the validation process.

```r
# index for training & validation
indx_training = 1:nrow(training_set)
indx_validation = (nrow(training_set)+1):nrow(tv_set)

# Define training control
train_control <- trainControl(
  method = "cv",
  index = list(TrainSet = indx_training),       # training index
  indexOut = list(TestSet = indx_validation),  # validation set
  savePredictions = "final"                    # Save prediction result
)

# Set range of searching for hyper parameters
grid <- expand.grid(
  mtry = c(2, 4, 6, 8, 10),  # nb of predictors
  ntree = c(500, 1000),      # nb of trees
  max_depth = c(10, 15, 20)  # max depth of each single tree
)

# Train the model
rf_fit = train(
  R1M_Usd ~ .,
  data = tv_set,
  method = "rf",
  trControl = train_control
)

# Output
print(rf_fit)
```

```
## Random Forest
##
## 2654 samples
##   45 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 2361
## Resampling results across tuning parameters:
##
##   mtry  RMSE        Rsquared    MAE
##    2    0.07320094  0.02392287  0.05578592
##   23    0.07441947  0.03488542  0.05661979
##   45    0.07465385  0.03713375  0.05661143
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was mtry = 2.
```

Neural Network

In this part, we will build a NN net work from one to five layers.

We use the first layer to have 64 units, and each later layer use units = former / 2.

We write a function to build the structure.

```r
# build_nn build a NN with k layers
build_nn = function(k) {
  model = keras_model_sequential() %>%
    layer_dense(units = 256, activation = 'relu', input_shape = c(45))

  if(k >= 2) {
    for (i in 2:k) {
      model = model %>%
        layer_dense(units = (256/(2^(i-1))), activation = 'relu')
    }
  }

  model = model %>%
    layer_dense(units = 1)

  # Compile
  model %>% compile(
    optimizer = "rmsprop",
    loss = "mse",
    metrics = c('MSE')
  )

  return(model)
}
```

Then try to fit the model by data.

```r
nn_list = list()

for (i in 1:5) {
  model = build_nn(i)
  current_fit = model %>%
    fit(
      x = as.matrix(x_training),
      y = as.matrix(y_training),
      epochs = 50,
      batch_size = 49,
      validation_data = list(as.matrix(x_validation), as.matrix(y_validation)),
      verbose = 0 # Stop printing fitting process
    )

  nn_list[[i]] = model
}
```

Analyze performance by testing set

```r
MSEs = rep(0, 8)
methods = c("OLS", "LASSO", "RF", "NN1", "NN2", "NN3", "NN4", "NN5")
```

OLS

```r
ols_predict = predict(ols, x_testing)
ols_res = y_testing$R1M_Usd - ols_predict
MSEs[1] = mean((ols_res)^2)
```

LASSO

We manually compute the prediction from the LASSO model

```r
lasso_pred = as.vector(predict.glmnet(lasso_fit, newx = as.matrix(x_testing)))

lasso_res <- y_testing$R1M_Usd - lasso_pred
MSEs[2] <- mean((lasso_res)^2, na.rm = TRUE)
```

RF

```r
rf_pred = predict(rf_fit, newdata = x_testing)

rf_res = y_testing$R1M_Usd - rf_pred

MSEs[3] = mean(rf_res^2)
```

Neural Networks

```r
nn_pred = list()
nn_res = list()

for (i in 1:5) {
  nn_pred[[i]] = predict(nn_list[[i]], as.matrix(x_testing))
  nn_res[[i]] = y_testing$R1M_Usd - nn_pred[[i]]
  MSEs[i+3] = mean(nn_res[[i]]^2)
}
```

Output for prediction result

```r
data.frame(method = methods, APSE = MSEs)
```

```
##   method        APSE
## 1    OLS 0.006046081
## 2  LASSO 0.005913862
## 3     RF 0.005909925
## 4    NN1 0.009643970
## 5    NN2 0.010703060
## 6    NN3 0.009313400
## 7    NN4 0.009774450
## 8    NN5 0.016760685
```

The results shows that the performance of Random forests and OLS are the best among the eight models.

Portfolio

For each model, we build a portfolio by using the five best stocks with highest predicted return in that month, and rebalance each month. Then we compare these portfolios with the equal weight portfolio.

We first write a function to pick out the five best stocks by the prediction.

```r
pick_five = function(predictions) {
  ID = filter(na.omit(data), date > vt_break)$ID
  date = filter(na.omit(data), date > vt_break)$date
  df = data.frame(ID = ID,
                  date = date,
                  pred = predictions)

  # Get the five best stocks of each month
  top_five = df %>%
    # arrange(date, desc(pred)) %>%
    group_by(date) %>%
    slice_max(order_by = pred, n = 5) %>%
    ungroup() %>%
    mutate(rank = rep(1:5, 11)) %>%
    select(-all_of("pred"))

  # Use pivot table to show
  pt = top_five %>%
    pivot_wider(names_from = date, values_from = ID)

  return(pt)
}
```

Change the `data$ID` column from factor to character for comparison

```r
data_defactor = data %>%
  mutate(ID = as.character(ID))
```

Then write a function that used the portfolio picked from the `pick_five` function to get a total return over the 11-month testing period

```r
get_return = function(pick5) {
  ndate = dim(pick5)[2]
  profit = 1
  for (i in 2:ndate) {
    r = 0
    for (j in 1:5) {
      related_date = as.Date(colnames(pick5)[i])
      related_ID = (pick5[j, i])[[1,1]]
      value = data_defactor %>%
        filter(date == related_date, ID == related_ID) %>%
        pull(R1M_Usd)
      r = r + 0.2 * value
    }
    profit = profit * (1+r)
  }
  return(profit-1)
}
```

```r
get_return(pick5 = pick_five(lasso_pred))
```

```
## [1] -0.1536415
```

Then we apply this to every method.

```r
pred_list = c(list(ols_predict,
                  lasso_pred,
                  rf_pred),
              nn_pred)
pred_return = rep(0, 8)
for (i in 1:8) {
  pred_return[i] = get_return(pick_five(pred_list[[i]]))
}
```

Also, we calculate the equal weight portfolio's return. We use the 25 stocks that comes from the 30 stock with the 5 stocks with NA removed.

```r
not_include_stocks = c("AXP", "GS", "JPM", "TRV", "V")
ew_data = data %>%
  filter(!ID %in% not_include_stocks) %>%
  filter(date > vt_break) %>%
  group_by(date) %>%
  summarize(return = mean(R1M_Usd)) %>%
  pull(return)
ew_data = na.omit(ew_data) # remove the NA from the last month
ew_return = prod(1+ew_data)
```

Then we print and compare the performance of all the portfolios.

```r
all_methods = c("EW", methods)
all_return = c(ew_return, pred_return)
return_df = data.frame(method = all_methods, return = all_return)
return_df = return_df %>%
  arrange(return)
return_df
```

```
##   method       return
## 1     RF -0.176753339
## 2    NN1 -0.162836901
## 3  LASSO -0.153641523
## 4    OLS -0.107053793
## 5    NN2 -0.092337787
## 6    NN4 -0.068705607
## 7    NN3 -0.067138299
## 8    NN5  0.005909434
## 9     EW  0.850323651
```

We can see that all stocks unperformed the Equal weight portfolio.

However, we can also compare the performance of the five-best-predicted stocks and the five-worst-predicted stocks.

```
pick_five_worst = function(predictions) {
  ID = filter(na.omit(data), date > vt_break)$ID
  date = filter(na.omit(data), date > vt_break)$date
  df = data.frame(ID = ID,
                  date = date,
                  pred = predictions)

  # Get the five best stocks of each month
  top_five = df %>%
    group_by(date) %>%
    slice_min(order_by = pred, n = 5) %>% # change slice_max to slice_min
    ungroup() %>%
    mutate(rank = rep(1:5, 11)) %>%
    select(-all_of("pred"))

  # Use pivot table to show
  pt = top_five %>%
    pivot_wider(names_from = date, values_from = ID)

  return(pt)
}

pred_return_worst = rep(0, 8)
for (i in 1:8) {
  pred_return_worst[i] = get_return(pick_five_worst(pred_list[[i]]))
}

data.frame(method = methods, best = pred_return, worst = pred_return_worst)
```

```
##   method         best      worst
## 1    OLS -0.107053793 -0.2952407
## 2  LASSO -0.153641523 -0.2636148
## 3     RF -0.176753339 -0.2312554
## 4    NN1 -0.162836901 -0.1277849
## 5    NN2 -0.092337787 -0.2391308
## 6    NN3 -0.067138299 -0.2108188
## 7    NN4 -0.068705607 -0.2275748
## 8    NN5  0.005909434 -0.2244395
```

We can see that almost all methods, have had their good predicted portfolio to outperformed the worst predicted portfolio a lot. So it is clear that a long-short strategy here can make profit.

Variable Importance Study

OLS and LASSO

For OLS and LASSO, we evaluate the importance of the variable by the absolute value of the related coefficient. This one is available since the data we used in trainning the model has been normalized.

```r
# OLS
ols_importance =
  data.frame(variable = names(coef(ols)[-1]),
             importance = abs(coef(ols)[-1])) %>%
  arrange(desc(importance)) %>%
  head(5)

# LASSO
lasso_coef = as.matrix(coef(lasso_fit))

lasso_coef_df = data.frame(variable = rownames(lasso_coef),
                           importance = abs(lasso_coef))

lasso_coef_df = lasso_coef_df[-1,]

colnames(lasso_coef_df)[2] = "importance"

lasso_importance = lasso_coef_df %>%
  arrange(desc(importance)) %>%
  head(5)
```

RF

Then we try to compare the variable importance of Random Forest.

```r
rf_importance = rf_fit$finalModel$importance

rf_importance =
  data.frame(variable = rownames(rf_importance),
             importance = as.vector(rf_importance)) %>%
  arrange(desc(importance)) %>%
  head(5)
```

Neural Networks

We use the weights analysis to evaluate the importance of NN models.

```r
get_importance_df = function(model) {
  # Get weight matrix of the first hidden matrix
  weights <- model$layers[[1]]$get_weights()[[1]]
  # Calculate the importance of variables from the weight matrix of
  # input layer to first hidden layer
  importance =  rep(0, 45)
  for (i in 1:45) {
    importance[i] = sum(abs(weights[i,]))
  }

  # Change to data frame
  importance_df =
    data.frame(variable = colnames(x_training),
               importance = importance) %>%
    arrange(desc(importance)) %>%
    head(5)

  return(importance_df)
}
```

```r
nn_importance_list = lapply(nn_list, get_importance_df)
```

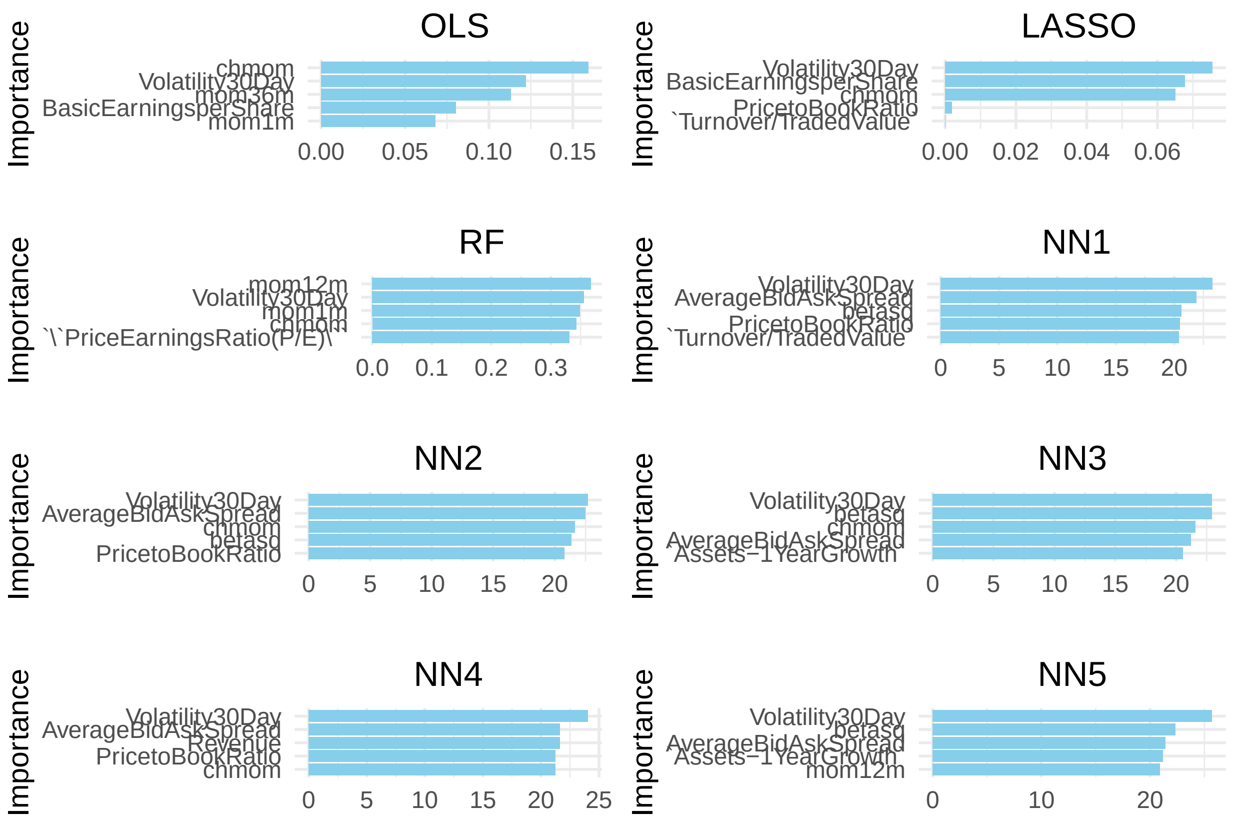
Plot the importance of each model.

```r
all_importance = c(list(ols_importance,
                        lasso_importance,
                        rf_importance),
                   nn_importance_list)

# Get bar variable importance plot
create_bar_plot <- function(importance_data, method_name) {
  ggplot(importance_data, aes(x = reorder(variable, importance), y = importance)) +
    geom_bar(stat = "identity", fill = "skyblue") +
    coord_flip() +  # Get horizontal histograms
    labs(title = method_name, x = "Importance", y = "") +
    theme_minimal() +
    theme(plot.title = element_text(hjust = 0.5))
}

# bar plot for each model
bar_plots <- lapply(seq_along(all_importance), function(i) {
  create_bar_plot(all_importance[[i]], methods[i])
})

grid.arrange(grobs = bar_plots, nrow = 4, ncol = 2)
```



We can see that the among the models, Volatility, momentum, PE/PB ratio, and bid ask spread are often considered as important variable.