ECB Tests 6

Contents

```
#explain what his script does and why and how to run code in readme file
```

Data

Main

```
# --- 1. ECB Deposit Facility Rate & Shadow Rate ---
ecb_rate_daily <- fredr(series_id = "ECBDFR", observation_start = as.Date(start_date))
ecb_rate_q <- ecb_rate_daily %>%
  mutate(quarter = as.yearqtr(date)) %>%
  group_by(quarter) %>%
  summarise(rate = last(value)) %>%
  mutate(date = as.Date(quarter))
# Wu-Xia Shadow Rate
shadow_rate_daily = as.data.frame(readMat("data/shadowrate_ECB.mat"))
colnames(shadow_rate_daily) <- c("DATE", "shadowrate")
shadow_rate_daily$DATE <- as.Date(paste0(shadow_rate_daily$DATE, "01"), format="%Y%m%d")
shadow_rate_daily$quarter <- as.yearqtr(as.Date(shadow_rate_daily$DATE))
shadow_rate_daily$month <- as.yearmon(as.Date(shadow_rate_daily$DATE))
quarterly_shadow = aggregate(shadowrate ~ quarter, data=shadow_rate_daily, FUN=mean, na.rm=T)
monthly_shadow = aggregate(shadowrate ~ month, data=shadow_rate_daily, FUN=mean, na.rm=T)

# --- 2. HICP Inflation (Euro Area) ---
inflation_data <- get_eurostat("prc_hicp_manr", filters = list(geo = "EA", coicop = "CP00"), type = "la
inflation_q <- inflation_data %>%
  filter(time >= start_date) %>%
  dplyr::select(date = time, inflation = values) %>%
  mutate(quarter = as.yearqtr(date)) %>%
  group_by(quarter) %>%
  summarise(inflation = mean(inflation, na.rm = TRUE)) %>%
  mutate(date = as.Date(quarter))

#inflation expectations
inflation_exp <- rdb(ids = "ECB/SPF/M.U2.HICP.POINT.P12M.Q.AVG")
#inflation_exp <- rdb(ids = "ECB/SPF/M.U2.HICP.POINT.P24M.Q.AVG")
inflation_exp_q <- inflation_exp %>%
  mutate(quarter = as.yearqtr(period)) %>%
  group_by(quarter) %>%
  summarise(exp_inflation = last(original_value)) %>%
  mutate(date = as.Date(quarter))

#P12M : 12-month ahead forecasts
inflation_q$exp_inflation = c(rep(NA,3),as.numeric(inflation_exp_q$exp_inflation),NA)
#P24M : 24-month ahead forecasts
#inflation_q$exp_inflation = c(rep(NA,7),as.numeric(inflation_exp_q$exp_inflation[1:101]))

# --- 3. Real GDP and Estimated Output Gap ---
# a) Real GDP for the Euro Area. The series ID is CLVMNACSCAB1GQE_A.
gdp_q <- fredr(
  series_id = "CLVMEURSCAB1GQEA19",
  observation_start = as.Date(start_date)) %>%
```

```r
  mutate(quarter = as.yearqtr(date)) %>%
  dplyr::select(quarter, real_gdp = value) %>%
  mutate(log_real_gdp = log(real_gdp))

# b) Estimate Potential GDP (the trend) using the HP Filter on the log of real GDP.
# The lambda value of 1600 is standard for quarterly data.
hp_gdp <- hpfilter(gdp_q$log_real_gdp, freq = 1600)
gdp_q$potential_gdp_log <- as.numeric(hp_gdp$trend)
ham_gdp_cycle <- filter_hamilton(gdp_q$log_real_gdp, p = 4, horizon = 8)
gdp_q$potential_gdp_log_ham <- gdp_q$log_real_gdp - ham_gdp_cycle

# Combine all data into a single data frame
data <- ecb_rate_q %>%
  dplyr::select(quarter, rate) %>%
  left_join(inflation_q, by = "quarter") %>%
  left_join(gdp_q, by = "quarter") %>%
  left_join(quarterly_shadow, by = "quarter")

# Create model variables
data <- data %>%
  mutate(
    realised_inflation_gap = inflation - 2.0,
    exp_inflation_gap = exp_inflation -2.0,
    output_gap_hp = 100 * (log_real_gdp - potential_gdp_log),
    output_gap_ham = 100 * (log_real_gdp - potential_gdp_log_ham),
    rate_lag = lag(rate, 1),
    shadowrate = case_when(
      quarter < "2012 Q3" | quarter >= "2022 Q3" ~ rate,
      TRUE ~ shadowrate),
    shadowrate_lag = lag(shadowrate, 1))

# Remove last row since no output data
data = subset(data, quarter < "2025 Q3")

# Clean environment
rm(gdp_q, hp_gdp, ecb_rate_daily, ecb_rate_q, inflation_data, inflation_q,
   inflation_exp, inflation_exp_q, monthly_shadow, quarterly_shadow, shadow_rate_daily)
```

Data Configuration

```r
# Choices in setup chunk

# --------- 1. Filter selection for output gap estimation ----------

# TRUE  = Use Hamilton Filter (newer, arguably more robust)
# FALSE = Use HP Filter (classic approach)

# Applying selection
if (USE_HAMILTON_FILTER) {
    data$output_gap <- data$output_gap_ham
  message(">> CONFIGURATION: Using Hamilton Filter for output gap estimation.")
```

```r
} else {
  data$output_gap <- data$output_gap_hp
  message(">> CONFIGURATION: Using HP Filter for output gap estimation.") }
```

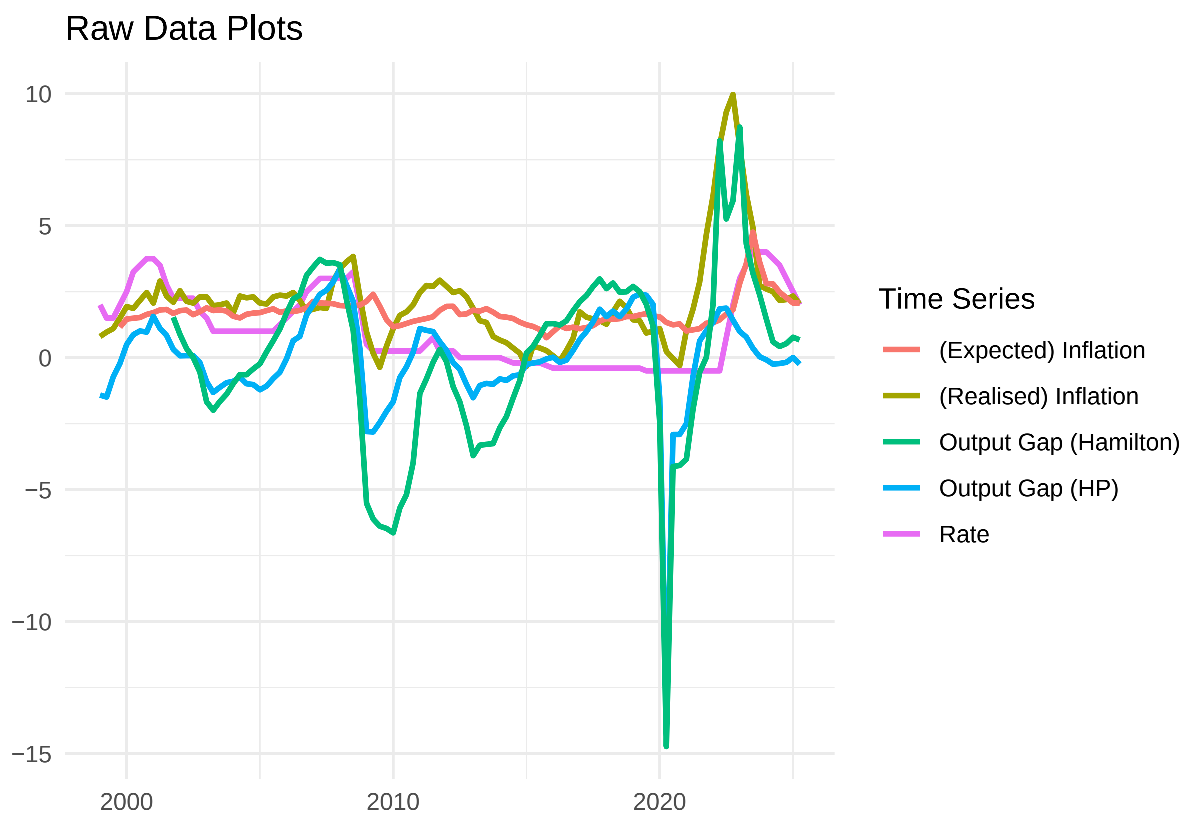
```
## >> CONFIGURATION: Using Hamilton Filter for output gap estimation.
```

```r
# --------- 2. Inflation expectations choice ----------

# TRUE  = Use inflation expectations from ECB survey of professional forecasts
# FALSE = Use realised inflation

# Applying selection
if (USE_INFLATION_EXPECTATIONS) {
    data$inflation_gap <- data$exp_inflation_gap
  message(">> CONFIGURATION: Using inflation expectations in Taylor Rule forecasting.")
} else {
  data$inflation_gap <- data$realised_inflation_gap
  message(">> CONFIGURATION: Using realised inflation in Taylor Rule forecasting.") }
```

```
## >> CONFIGURATION: Using realised inflation in Taylor Rule forecasting.
```

Raw Data Plots

```r
ggplot(data, aes(x = date, color = series)) +
  geom_line(aes(y = rate, color = "Rate"), linewidth = 1) +
  geom_line(aes(y = inflation, color = "(Realised) Inflation"), linewidth = 1) +
  geom_line(aes(y = exp_inflation, color = "(Expected) Inflation"), linewidth = 1) +
  geom_line(aes(y = output_gap_hp, color = "Output Gap (HP)"), linewidth = 1) +
  geom_line(aes(y = output_gap_ham, color = "Output Gap (Hamilton)"), linewidth = 1) +
  labs(title = "Raw Data Plots", x = "", y = "", color = "Time Series") +
  theme_minimal()
```



Data Properties

```r
# Interest rate is I(1)
test1 = aTSA::adf.test(data$rate, output=F)
test1$type1
```

```
##      lag       ADF    p.value
## [1,]   0 -1.030614 0.30884518
## [2,]   1 -2.163114 0.03162811
## [3,]   2 -1.951938 0.04983296
## [4,]   3 -2.234322 0.02548950
## [5,]   4 -2.293507 0.02276643
```

```r
# Inflation is I(1)
test2 = aTSA::adf.test(data$inflation, output=F)
test2$type1
```

```
##      lag       ADF    p.value
## [1,]   0 -1.125153 0.27478730
## [2,]   1 -2.251445 0.02452223
## [3,]   2 -2.472595 0.01529054
## [4,]   3 -2.263254 0.02402930
## [5,]   4 -1.490401 0.14320596
```

```r
# Output gap is I(0), likely from the "gap" part
test3 = aTSA::adf.test(data$output_gap, output=F)
test3$type1
```

```
##      lag       ADF p.value
## [1,]   0 -3.180481    0.01
## [2,]   1 -2.862680    0.01
## [3,]   2 -2.846986    0.01
## [4,]   3 -3.604893    0.01
```

```r
# Cleanup
rm(test1,test2,test3)

# Are interest rates and inflation co-integrated?
aTSA::coint.test(data$rate, data$inflation)
```

```
## Response: data$rate
## Input: data$inflation
## Number of inputs: 1
## Model: y ~ X + 1
## -------------------------------
## Engle-Granger Cointegration Test
## alternative: cointegrated
##
## Type 1: no trend
##     lag      EG p.value
##  4.0000 -2.9904  0.0433
## -----
##  Type 2: linear trend
##     lag      EG p.value
##   4.000  -0.778   0.100
## -----
##  Type 3: quadratic trend
##     lag      EG p.value
##   4.000  -0.566   0.100
## -----------
## Note: p.value = 0.01 means p.value <= 0.01
##     : p.value = 0.10 means p.value >= 0.10
```

Taylor Rule Estimation

Without Lags

$$i_t = \pi^* + \beta(\pi_t - \pi^*) + \gamma(y_t - \bar{y_t})$$

```r
TR <- lm(rate ~ realised_inflation_gap + output_gap, data = data)
TRsr <- lm(shadowrate ~ realised_inflation_gap + output_gap, data = data)

table1 <- export_summs(TR, TRsr, vcov = sandwich::NeweyWest,
          model.names = c("TR", "TR w/ SR"), digits = 4)
huxtable::caption(table1) <- "No Lag, No Expectations"
table1
```

Table 1: No Lag, No Expectations

	TR	TR w/ SR
(Intercept)	0.8345	-0.8601
	(0.9055)	(2.6782)
realised_inflation_gap	0.1963	0.6857
	(0.4289)	(0.6062)
output_gap	0.0824	-0.0567
	(0.1628)	(0.2758)
N	95	95
R2	0.1726	0.1159

*** $p < 0.001$; ** $p < 0.01$; * $p < 0.05$.

```r
TR_e <- lm(rate ~ exp_inflation_gap + output_gap, data = data)
TRsr_e <- lm(shadowrate ~ exp_inflation_gap + output_gap, data = data)
TR_ie <- lm(rate ~ realised_inflation_gap + exp_inflation_gap + output_gap, data = data)
TRsr_ie <- lm(shadowrate ~ realised_inflation_gap + exp_inflation_gap + output_gap, data = data)

table2 <- export_summs(TR_e, TRsr_e, TR_ie, TRsr_ie, vcov = sandwich::NeweyWest,
          model.names = c("TR", "TR w/ SR", "TR", "TR w/ SR"), digits = 4)
huxtable::caption(table2) <- "No Lag, with Inflation Expectations"
table2
```

Lagged Models

$$i_t = \pi^* + \phi i_{t-1} + \beta(\pi_t - \pi^*) + \gamma(y_t - \bar{y_t})$$

Table 2: No Lag, with Inflation Expectations

	TR	TR w/ SR	TR	TR w/ SR
(Intercept)	1.3440 ***	0.2867	1.3381 ***	0.1715
	(0.3179)	(1.5603)	(0.3714)	(1.5918)
exp_inflation_gap	1.7238 ***	3.7581 **	1.7107 ***	3.5044 **
	(0.3523)	(1.3756)	(0.3466)	(1.1674)
output_gap	0.0707	-0.0026	0.0662	-0.0898
	(0.0468)	(0.1871)	(0.0810)	(0.2133)
realised_inflation_gap			0.0163	0.3171
			(0.1277)	(0.3433)
N	95	95	95	95
R2	0.6177	0.3904	0.6180	0.4086

*** $p < 0.001$; ** $p < 0.01$; * $p < 0.05$.

```
lTR <- lm(rate ~ rate_lag + realised_inflation_gap + output_gap, data = data)
lTRsr <- lm(shadowrate ~ shadowrate_lag + realised_inflation_gap + output_gap, data = data)

table3 <- export_summs(lTR, lTRsr, vcov = sandwich::NeweyWest,
          model.names = c("TR", "TR w/ SR"), digits = 4)
huxtable::caption(table3) <- "Interest Rate Lag, No Expectations"
table3
```

```
lTR_e <- lm(rate ~ rate_lag + exp_inflation_gap + output_gap, data = data)
lTRsr_e <- lm(shadowrate ~ shadowrate_lag + exp_inflation_gap + output_gap, data = data)
lTR_ie <- lm(rate ~ rate_lag + realised_inflation_gap + exp_inflation_gap + output_gap, data = data)
lTRsr_ie <- lm(shadowrate ~ shadowrate_lag + realised_inflation_gap + exp_inflation_gap + output_gap, da

table4 <- export_summs(lTR_e, lTRsr_e, lTR_ie,lTRsr_ie, vcov = sandwich::NeweyWest,
          model.names = c("TR", "TR w/ SR", "TR", "TR w/ SR"), digits = 4)
huxtable::caption(table4) <- "Interest Rate Lag, with Inflation Expectations"
table4
```

Checking for structural breaks

```
# Formula to test for breaks
break_formula = rate ~ rate_lag + inflation_gap + output_gap

# Suspected break: Start of ZLB in 2012 Q3
breakpoint1_obs = 55 #R = 55 in evaluation chunk
```

Table 3: Interest Rate Lag, No Expectations

	TR	TR w/ SR
(Intercept)	0.0507	-0.0777
	(0.0410)	(0.0618)
rate_lag	0.9189 ***	
	(0.0390)	
realised_inflation_gap	0.0879 *	0.2498 ***
	(0.0350)	(0.0344)
output_gap	0.0218	-0.0116
	(0.0163)	(0.0185)
shadowrate_lag		0.9530 ***
		(0.0176)
N	95	95
R2	0.9594	0.9828

*** $p < 0.001$; ** $p < 0.01$; * $p < 0.05$.

```r
breakpoint1_date <- data$quarter[breakpoint1_obs]

# Suspected break: Covid
breakpoint2_obs = 85 #R = 85 in evaluation chunk
breakpoint2_date <- data$quarter[breakpoint2_obs]

# Chow test (rejecting the null means there are structural breaks)
chow_test1 <- sctest(break_formula, type = "Chow", point = breakpoint1_obs, data = data)
chow_test2 <- sctest(break_formula, type = "Chow", point = breakpoint2_obs, data = data)


# Table with Chow results (for suspected breaks)
chow_df <- data.frame(
  Event = c("ZLB Start", "COVID-19 Start"),
  Date = c(as.character(breakpoint1_date), as.character(breakpoint2_date)),
  `p-value` = c(chow_test1$p.value, chow_test2$p.value),check.names = FALSE)

kable(chow_df,
      digits = 4, format = format,
      caption = "Chow tests for suspected structural breaks") %>%
  kable_styling(bootstrap_options = "striped", full_width = FALSE)
```

Table 4: Interest Rate Lag, with Inflation Expectations

	TR	TR w/ SR	TR	TR w/ SR
(Intercept)	0.1804 *	0.0009	0.1341 ***	-0.0896
	(0.0796)	(0.0869)	(0.0378)	(0.0622)
rate_lag	0.8606 ***		0.8742 ***	
	(0.0438)		(0.0392)	
exp_inflation_gap	0.2387 **	0.1296	0.1536 ***	-0.0546
	(0.0733)	(0.1039)	(0.0435)	(0.0622)
output_gap	0.0449 *	0.0592	0.0233	-0.0108
	(0.0218)	(0.0474)	(0.0165)	(0.0182)
shadowrate_lag		0.9630 ***		0.9581 ***
		(0.0287)		(0.0190)
realised_inflation_gap			0.0770 *	0.2532 ***
			(0.0378)	(0.0347)
N	95	95	95	95
R2	0.9544	0.9712	0.9611	0.9828

*** $p < 0.001$; ** $p < 0.01$; * $p < 0.05$.

Table 5: Chow tests for suspected structural breaks

Event	Date	p-value
ZLB Start	2012 Q3	0.0020
COVID-19 Start	2020 Q1	0.0365

```r
# Estimate Bai-Perron test & output results
BP_test = breakpoints(break_formula, data = data)
BP_test_res = summary(BP_test)

# Optimal values (modular!)
bic_values <- BP_test_res$RSS[2, ]
optimal_m <- as.numeric(names(bic_values)[which.min(bic_values)])

# Make a table out of results (also modular!)
if (optimal_m == 0) {
  bp_df <- data.frame(
    `Detected Break Dates` = "No structural breaks detected",
    check.names = FALSE)
  } else {
  break_obs <- na.omit(BP_test_res$breakpoints[optimal_m, ])
```

Table 6: Bai-Perron test for multiple breaks

Detected Break Dates
2006 Q2
2019 Q1

```
  detected_dates <- data$quarter[break_obs]
  bp_df <- data.frame(
    `Detected Break Dates` = as.character(detected_dates),
    check.names = FALSE) }

kable(bp_df, format = format,
  caption = "Bai-Perron test for multiple breaks") %>%
  kable_styling(bootstrap_options = "striped", full_width = FALSE, position = "left")
```

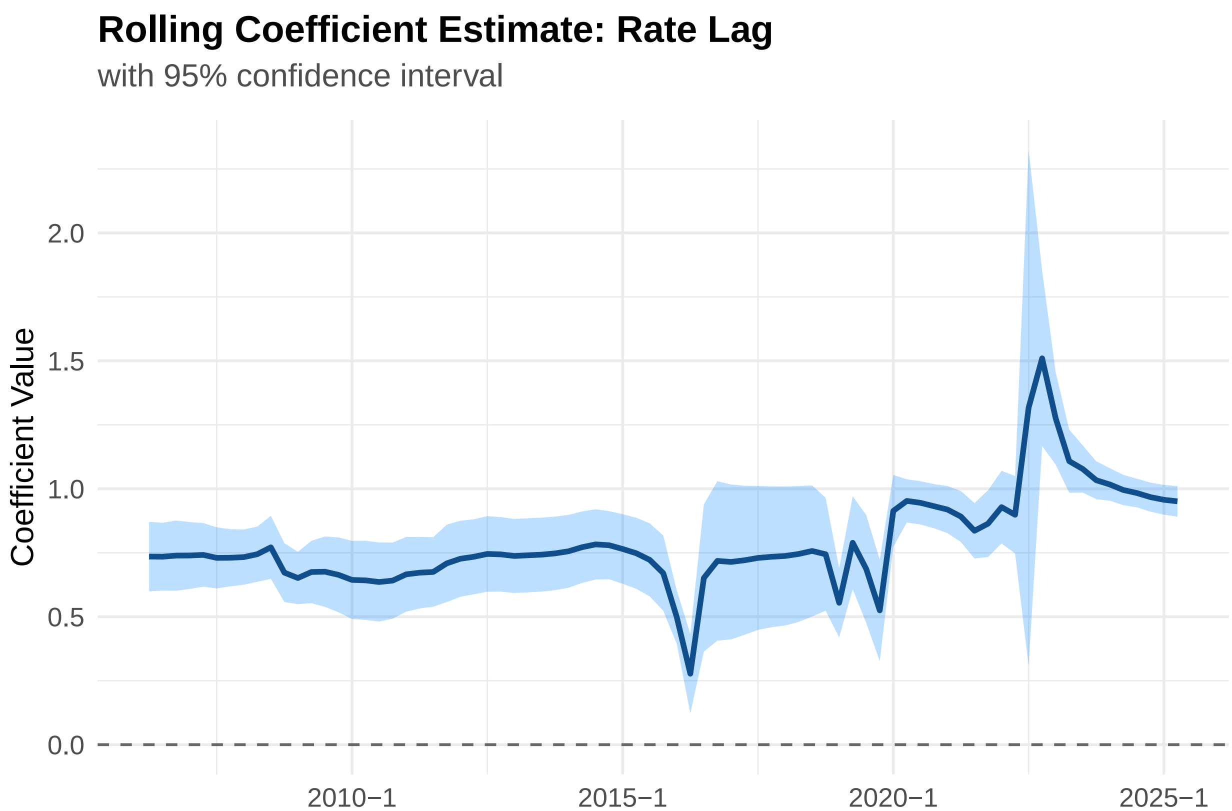
```
# Note: breaks_obs shows in which row the BP breaks are
```

Rolling Estimation (for structural breaks)

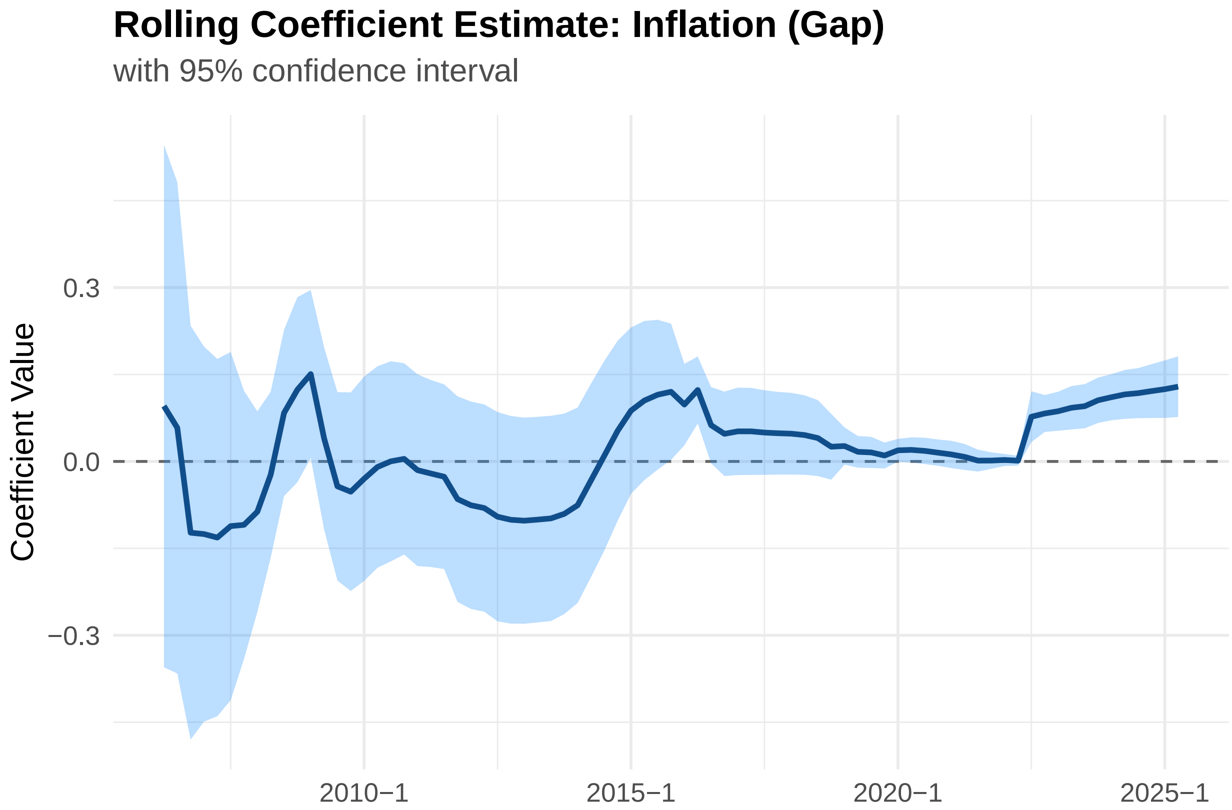
```
# This function automatically plots the output from the rolling estimation loop
plot_rolling_coefs <- function(data, var_name, var_name_title=var_name) {

  # 1. Dynamically create the column names for CIs
  lower_col <- paste0(var_name, "_lower")
  upper_col <- paste0(var_name, "_upper")

  # 2. Create the plot, using the .data[[]] pronoun to find the
  # columns based on the variable names (modularity)

  plot <- ggplot(data, aes(x = date)) +

    # Add a dashed line at y=0 for reference
    geom_hline(yintercept = 0, linetype = "dashed", color = "grey40", linewidth = 0.5) +

    # Add the 95% confidence interval ribbon
    geom_ribbon(aes(ymin = .data[[lower_col]], ymax = .data[[upper_col]]),
                fill = "dodgerblue", alpha = 0.3) +

    # Add the coefficient estimate line
    geom_line(aes(y = .data[[var_name]]),
              color = "dodgerblue4", linewidth = 1) +

    # Aesthetic
    labs(title = paste("Rolling Coefficient Estimate:", var_name_title),
        subtitle = "with 95% confidence interval",
      x = "",
      y = "Coefficient Value") +
    theme_minimal() +
    theme(plot.title = element_text(face = "bold", size = 14, margin = margin(b=5)),
          plot.subtitle = element_text(size = 12, color = "grey30", margin = margin(b=10)),
```

```r
        axis.title = element_text(size = 12),
        axis.text = element_text(size = 10))
    return(plot) }
```

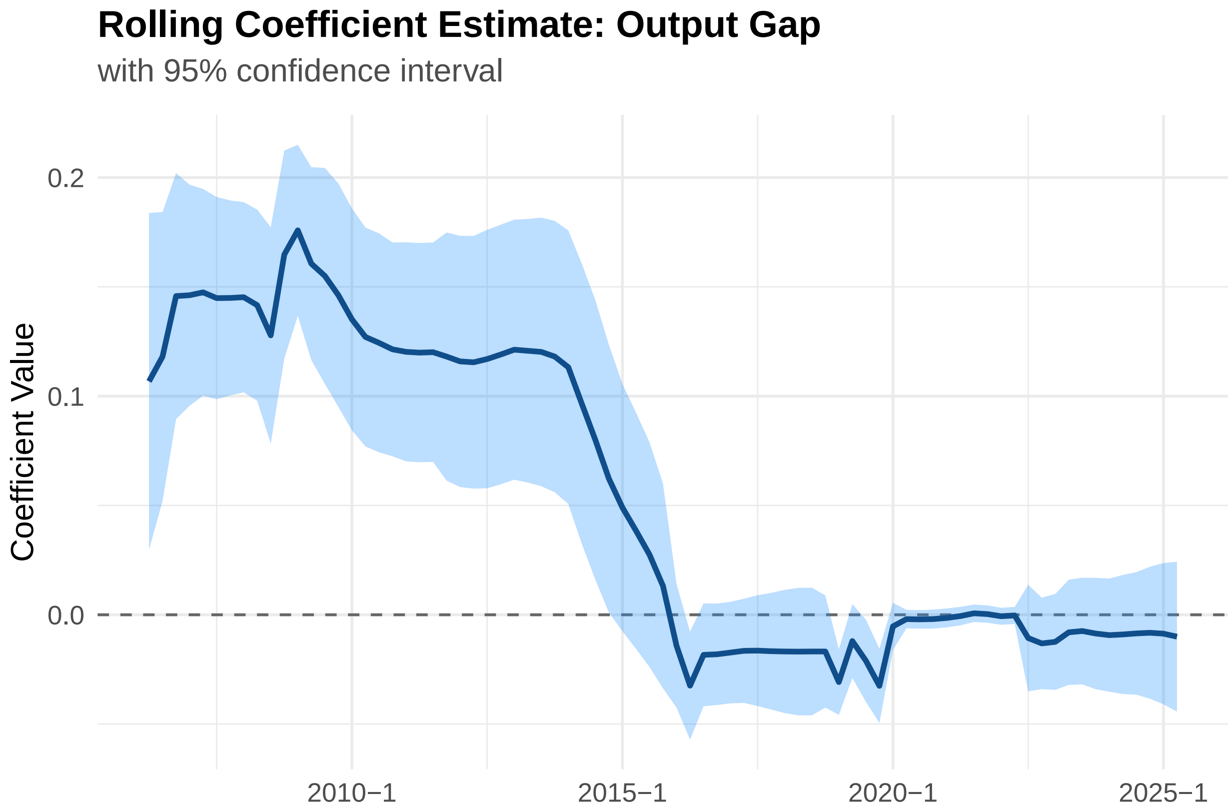
```r
# Set rolling window (in quarters) & Looping Parameter
W = 30
L = nrow(data) - W + 1
formula = rate ~ rate_lag + inflation_gap + output_gap

# Preparation of result data, dates, var names, and confidence intervals
var_names <- attr(terms(formula), "term.labels")
window_end_dates <- data$quarter[W:nrow(data)] # First window [1:W] ends at data$date[W]
TR_roll <- data.frame(date = window_end_dates)
TR_roll[var_names] <- NA
lower_col_names <- paste0(var_names, "_lower")
upper_col_names <- paste0(var_names, "_upper")

# Looped estimation of TR, outputs coefficients and CIs
for (l in 1:L) {
  # 1. Define splits (with rolling scheme)
  rolled_data <- data[l:(W + l - 1), ]

  # 2. Estimate TR on split data, using whatever formula is desired
  TR_estimate <- lm(formula, data = rolled_data)

  # 3. Pull out coefficients & compute confidence intervals
  all_coefs <- coef(TR_estimate)
  all_cis <- confint(TR_estimate)

  TR_roll[l, var_names] <- all_coefs[var_names]
  TR_roll[l, lower_col_names] <- all_cis[var_names, 1]
  TR_roll[l, upper_col_names] <- all_cis[var_names, 2]   }

# Plotting (note: this part is not modular, obviously)
plot_rolling_coefs(TR_roll, "rate_lag", var_name_title="Rate Lag")
```




```
plot_rolling_coefs(TR_roll, "inflation_gap", var_name_title="Inflation (Gap)")
```




```r
plot_rolling_coefs(TR_roll, "output_gap", var_name_title="Output Gap")
```


Rolling Coefficient Estimate: Output Gap
with 95% confidence interval
Coefficient Value
0.2
0.1
0.0
2010−1
2015−1
2020−1
2025−1

Forecasting Model Evaluation

Helpers

```r
#--------------------------------------------------------------------
# Helper function for adding p-value significance stars
#--------------------------------------------------------------------

  format_p_values_with_stars <- function(p) {
    stars <- case_when(
      p < 0.01 ~ "***",
      p < 0.05 ~ "**",
      p < 0.10 ~ "*",
      TRUE     ~ "")
    paste0(format(round(p, 4), nsmall = 3), " ", stars)}

#--------------------------------------------------------------------
# HELPER FUNCTION FOR MINCER-ZARNOWITZ REPORTING
#--------------------------------------------------------------------
# This function runs the Mincer-Zarnowitz regression (Actuals ~ Forecasts)
# for each horizon h and tests the joint null hypothesis H0: (alpha, beta) = (0, 1).
#--------------------------------------------------------------------

generate_mincer_zarnowitz_report <- function(F_model,
                                             Actual_values,
                                             H,
                                             model_caption,
                                             format = "html") {
  # Pre-allocate storage for results
  mz_results <- data.frame(
    Horizon = 1:H,
    Alpha = numeric(H),
    Beta = numeric(H),
    P_Value_Joint_Test = numeric(H))

  for (h in 1:H) {
    # 1. Create a clean data frame for this horizon
    #    This pairs the forecasts and actual values and removes any NAs,
    #    ensuring they remain perfectly aligned.
    df_h <- data.frame(
      actuals = Actual_values[[h]],
      forecasts = F_model[[h]] ) %>%
      na.omit()

    # Check if we have enough data to run the regression (at least 2 obs)
    if (nrow(df_h) > 2) {
      # 2. Run MZ regression
      mz_reg <- lm(actuals ~ forecasts, data = df_h)

      # 3. Get coefficients
      coeffs <- summary(mz_reg)$coefficients
      mz_results$Alpha[h] <- coeffs[1, 1]
      mz_results$Beta[h]  <- coeffs[2, 1]
```

```
    # Using NW errors as seen in class, with lag selection h-1
    v_matrix <-
      if (h == 1) {
        # h=1: No autocorrelation, use standard "White" (HC) errors
        sandwich::vcovHC(mz_reg, type = "HC3")
      } else {
        # h>1: Use Newey-West, manually setting lag = h-1
        sandwich::NeweyWest(mz_reg, lag = h - 1)}

    # 4. Test Joint Hypothesis H0: Alpha = 0 AND Beta = 1 and store pvalues
    test_joint <- linearHypothesis(mz_reg,
                       c("(Intercept) = 0", "forecasts = 1"), vcov. = v_matrix)
    mz_results$P_Value_Joint_Test[h] <- test_joint$"Pr(>F)"[2]

  } else {
    # Not enough data to run regression for this horizon
    mz_results$Alpha[h] <- NA_real_
    mz_results$Beta[h]  <- NA_real_
    mz_results$P_Value_Joint_Test[h] <- NA_real_ } }

# Format the results for the table
mz_results <- mz_results %>%
  mutate(Alpha = round(Alpha, 4),
         Beta = round(Beta, 4),
         P_Value_Joint_Test = format_p_values_with_stars(P_Value_Joint_Test))

# Create the table
table_output <- kable(
  mz_results,
  format = format,
  booktabs = TRUE,
  caption = model_caption,
  digits = 4,
  col.names = c("h", "Alpha", "Beta", "pv(Joint)"),
  escape = FALSE ) %>%
  kable_styling(
    latex_options = c("striped", "scale_down"),
    position = "center") %>%
  column_spec(1, bold = TRUE, border_right = TRUE) %>%
  column_spec(4, monospace = TRUE) %>%
  footnote(
    general = "pv(Joint) is the p-value for the joint hypothesis H_0: (Alpha, Beta) = (0, 1). A high p
    symbol = c(
      "Signif. codes:  '***' 0.01,  '**' 0.05,  '*' 0.1"),
    general_title = "Note:",
    symbol_title = "",
    footnote_as_chunk = TRUE,
    threeparttable = TRUE)
return(table_output) }



#--------------------------------------------------------------------
```

```r
# HELPER FUNCTION FOR REPORTING DM tests
#-----------------------------------------------------------------

# This function creates the DM tests and kable output
generate_report_table <- function(FE_TR_model, FE_BM_model, H, model_caption, format = "html") { MSFE_TR
  MSFE_BM = numeric(H)

  # Calculate MSFEs
  for (h in 1:H) {
    # Ensure errors are cleaned of NAs
    fe1 <- na.omit(FE_TR_model[[h]])
    fe2 <- na.omit(FE_BM_model[[h]])

    MSFE_TR[h] = mean((fe1)^2)
    MSFE_BM[h] = mean((fe2)^2)}

  # Run DM Tests
  DMpvalues = matrix(, nrow = H, ncol = 3)
  colnames(DMpvalues) <- c("DM_Two_Sided", "DM_Greater", "DM_Lesser")
  for (h in 1:H){
    # Note: dm.test needs the *full* (un-omitted) error vectors
    # to align them properly, hence using the original list inputs
    x1 = dm.test(e1 = FE_BM_model[[h]], e2 = FE_TR_model[[h]], h = h)
    x2 = dm.test(e1 = FE_BM_model[[h]], e2 = FE_TR_model[[h]], h = h, alternative = "greater")
    x3 = dm.test(e1 = FE_BM_model[[h]], e2 = FE_TR_model[[h]], h = h, alternative = "less")
    DMpvalues[h, 1] = round(x1$p.value, digits = 4)
    DMpvalues[h, 2] = round(x2$p.value, digits = 4)
    DMpvalues[h, 3] = round(x3$p.value, digits = 4)}

  # Create final table data
  forecast_comparison <- data.frame(
    Horizon = 1:H,
    MSFE_TR = MSFE_TR,
    MSFE_BM = MSFE_BM) %>%
    mutate(Ratio_TR_vs_BM = MSFE_TR / MSFE_BM)

  forecast_comparison <- bind_cols(forecast_comparison, as.data.frame(DMpvalues))

  final_data_formatted <- forecast_comparison %>%
    mutate(across(starts_with("DM_"), format_p_values_with_stars))

  # Create the kable table
  table_output <- kable(
    final_data_formatted,
    format = format,
    booktabs = TRUE,
    caption = model_caption,
    digits = 4,
    col.names = c("h", "MSFE TR", "MSFE BM", "Ratio", "DM Two-Sided", "DM Greater", "DM Lesser"),
    escape = FALSE) %>%
    kable_styling(
      latex_options = c("striped", "scale_down"),
      position = "center") %>%
```

```r
    column_spec(1, bold = TRUE, border_right = TRUE) %>%
    column_spec(5:7, monospace = TRUE) %>%
    footnote(
      general = "TR refers to the forecast made with an estimated Taylor Rule. BM refers to a benchmark
      symbol = c(
        "'DM Greater' tests if the TR model is significantly more accurate than the BM model.",
        "'DM Lesser' tests if the TR model is significantly less accurate than the BM model."),
      general_title = "Note:",
      symbol_title = "DM Test Alternative Hypotheses (H_A):",
      footnote_as_chunk = TRUE,
      threeparttable = TRUE)

  return(table_output)}
```

Estimation

```r
#parameters
R = 85 # Chow: Structural breaks at R=55 and R=85
cat("Evaluation sample starts after", as.character(data$quarter[R]))
```

Evaluation sample starts after 2020 Q1

```r
P = nrow(data) - R #but will effectively be: P = T-h-R
H = 10 #number of different horizons (takes 10 to go until 2027 Q4)

#note: we are doing a recursive estimation scheme for out-of-sample tests
#note: we are doing direct forecasts

#------------------------------------------------------------------
# 1. DEFINE THE TAYLOR RULE (TR) MODEL FORMULAS
#------------------------------------------------------------------

# TR specifications (using either current inflation or inflation expectations
#                    according to configuration, same with HP vs Hamilton)
formula_1 <- rate ~ inflation_gap + output_gap
formula_2 <- shadowrate ~ inflation_gap + output_gap
formula_3 <- rate ~ rate_lag + inflation_gap + output_gap
formula_4 <- shadowrate ~ shadowrate_lag + inflation_gap + output_gap

#------------------------------------------------------------------
# 2. PRE-ALLOCATE STORAGE FOR ALL RESULTS
#------------------------------------------------------------------

# We need 4 lists for the TR models, 1 list for the shared benchmark
init_storage_list <- function(H, P) {
  storage <- vector("list", length = H)
  for (h in 1:H) {
    storage[[h]] <- rep(NA_real_, P)}
  return(storage)}

# Storage for realised values
```

```r
Actuals <- init_storage_list(H, P)

# Storage for Forecasts
F_TR_1 <- init_storage_list(H, P) # Model 1: shadowrate, no lag
F_TR_2 <- init_storage_list(H, P) # Model 2: rate, no lag
F_TR_3 <- init_storage_list(H, P) # Model 3: shadowrate, with lag
F_TR_4 <- init_storage_list(H, P) # Model 4: rate, with lag
F_BM   <- init_storage_list(H, P) # Benchmark: ARIMA

# Storage for Forecast Errors
FE_TR_1 <- init_storage_list(H, P) # Model 1: shadowrate, no lag
FE_TR_2 <- init_storage_list(H, P) # Model 2: rate, no lag
FE_TR_3 <- init_storage_list(H, P) # Model 3: shadowrate, with lag
FE_TR_4 <- init_storage_list(H, P) # Model 4: rate, with lag
FE_BM   <- init_storage_list(H, P) # Benchmark: ARIMA


#-------------------------------------------------------------------
# 3. SETUP & RUN THE PARALLEL BACKTESTING LOOP
#-------------------------------------------------------------------
num_cores <- detectCores() - 1
cl <- makeCluster(num_cores)
registerDoParallel(cl)

# .export sends read-only objects to each core
# .packages loads libraries on each core
worker_results <- foreach(
  p = P:1,
  .packages = c("forecast", "stats", "dplyr"),
  .export = c("data", "H", "formula_1", "formula_2", "formula_3", "formula_4")
) %dopar% {

  # 1. Define splits (with rolling scheme)
  training <- data[(1 + nrow(data) - R - p):(nrow(data) - p), ]
  testing <- data[(nrow(data) - (p - 1)):nrow(data), ]

  # --- 2. Fit common models only once ---
  # note: d=1 for interest and inflation as non-stationary
  inflation_arma <- my.auto.arima(training$inflation_gap, max.p=4, max.q=4, d=1)
  outputgap_arma <- my.auto.arima(training$output_gap, max.p=4, max.q=4, d=0)
  interest_arma <- my.auto.arima(training$rate, max.p=4, max.q=4, d=1) # Benchmark

  # --- 3. Get common forecasts only once (all H horizons) ---
  inflation_forecasts <- my.forecast(inflation_arma, h = H)
  outputgap_forecasts <- my.forecast(outputgap_arma, h = H)
  BMpredicted_rates <- my.forecast(interest_arma, h = H)

  # --- 4. Fit the 4 TR models ---
  TR_model_1 <- lm(formula_1, data = training)
  TR_model_2 <- lm(formula_2, data = training)
  TR_model_3 <- lm(formula_3, data = training)
  TR_model_4 <- lm(formula_4, data = training)
```

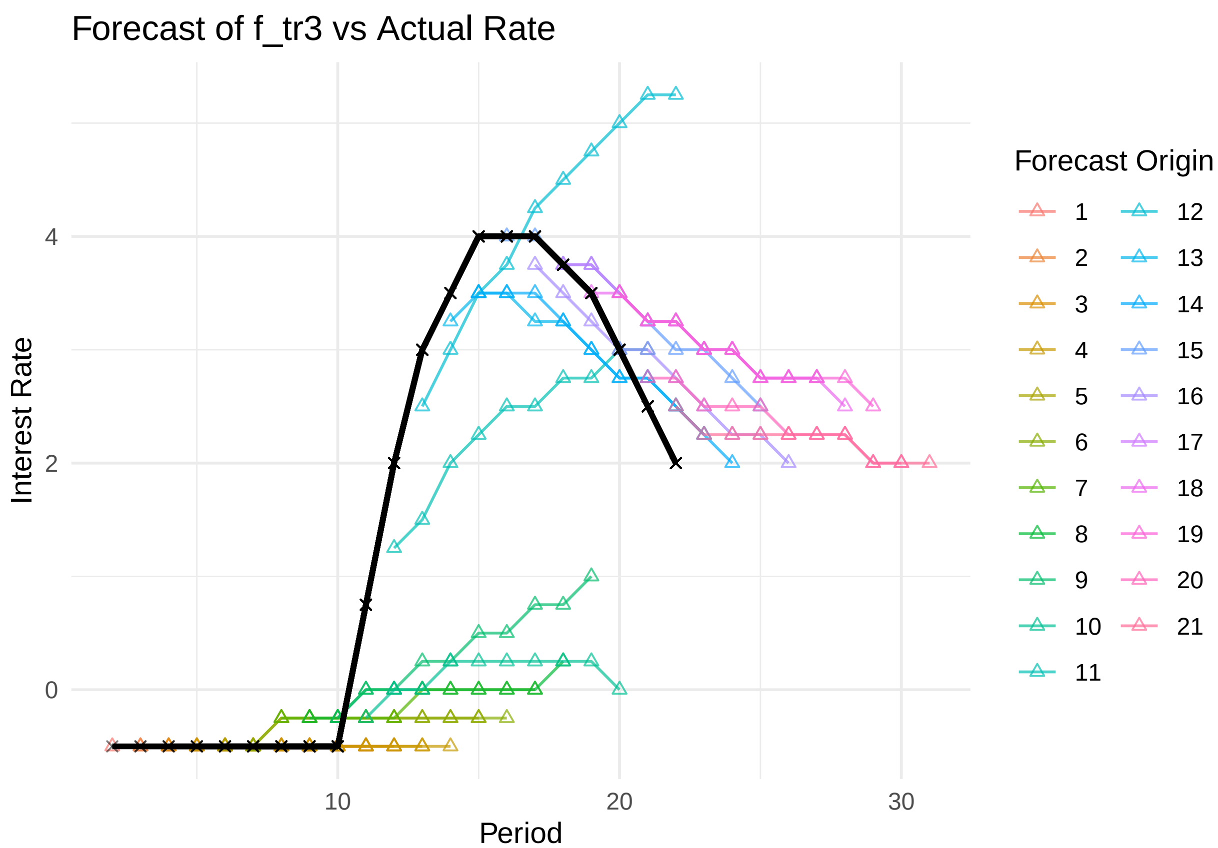
```r
# --- 5. Build forecast input data & get forecasts for non-lagged models ---
# These are direct forecasts
new_data_base <- data.frame(
  inflation_gap = inflation_forecasts,
  output_gap = outputgap_forecasts)

TR_preds_1 <- round(pmax(predict(TR_model_1, new_data_base), min(data$rate)) / 0.25) * 0.25
TR_preds_2 <- round(pmax(predict(TR_model_2, new_data_base), min(data$rate)) / 0.25) * 0.25
BM_preds <- round(pmax(BMpredicted_rates, min(data$rate)) / 0.25) * 0.25

# --- 6. Get forecasts for lagged models via iteration ---
# We must loop 1 step at a time, feeding forecasts back in.

# a) Pre-allocate storage for H forecasts
TR_preds_3 <- numeric(H)
TR_preds_4 <- numeric(H)

# b) Get the last known lag from the training set (lag for h=1 forecast)
current_rate_lag  <- last(training$rate)
current_shadowrate_lag <- last(training$shadowrate)

# Loop for iterative forecasting
for (h in 1:H) {
  # --- Prepare dataset for predictions ---
  new_data_3_h <- data.frame(
    inflation_gap = inflation_forecasts[h],
    output_gap = outputgap_forecasts[h],
    rate_lag = current_rate_lag)
  new_data_4_h <- data.frame(
    inflation_gap = inflation_forecasts[h],
    output_gap = outputgap_forecasts[h],
    shadowrate_lag = current_shadowrate_lag )

  # Get the forecast values (keep for lag, and then round for actual prediction)
  pred_3_h <- predict(TR_model_3, new_data_3_h)
  TR_preds_3[h] <- round(pmax(pred_3_h, min(data$rate)) / 0.25) * 0.25
  pred_4_h <- predict(TR_model_4, new_data_4_h)
  TR_preds_4[h] <- round(pmax(pred_4_h, min(data$rate)) / 0.25) * 0.25

  # Update lag for h+1
  current_shadowrate_lag <- pred_4_h
  current_rate_lag <- pred_3_h }

# --- 7. Get actual values in evaluation sample  ---
actual_rates <- testing$rate[1:H]

# --- 8. Return all FORECASTS and ACTUALS from the worker ---
list(f_tr1 = TR_preds_1,
     f_tr2 = TR_preds_2,
     f_tr3 = TR_preds_3,
     f_tr4 = TR_preds_4,
     f_bm  = BM_preds,
     actuals = actual_rates) }
```

```
# --- Stop the Cluster ---
stopCluster(cl)
rm(cl)

#------------------------------------------------------------------
# 4. UNPACK PARALLEL RESULTS INTO STORAGE LISTS
#------------------------------------------------------------------

# 'worker_results' is a list of P lists. We need to re-organize it.
for (i in 1:P) {
  # i=1 corresponds to p=P, i=2 to p=P-1, ... i=P to p=1
  # This 'storage_index' matches the loop order
  storage_index <- i
  p_results <- worker_results[[i]]

  for (h in 1:H) {
    # Get the raw values for this h
    actual_val <- p_results$actuals[h]
    f_tr1_val  <- p_results$f_tr1[h]
    f_tr2_val  <- p_results$f_tr2[h]
    f_tr3_val  <- p_results$f_tr3[h]
    f_tr4_val  <- p_results$f_tr4[h]
    f_bm_val   <- p_results$f_bm[h]

    # Store Actuals (for MZ)
    Actuals[[h]][storage_index] <- actual_val

    # Store Forecasts (for MZ)
    F_TR_1[[h]][storage_index] <- f_tr1_val
    F_TR_2[[h]][storage_index] <- f_tr2_val
    F_TR_3[[h]][storage_index] <- f_tr3_val
    F_TR_4[[h]][storage_index] <- f_tr4_val
    F_BM[[h]][storage_index]   <- f_bm_val

    # Calculate and Store Errors (for MSFE/DM)
    FE_TR_1[[h]][storage_index] <- f_tr1_val - actual_val
    FE_TR_2[[h]][storage_index] <- f_tr2_val - actual_val
    FE_TR_3[[h]][storage_index] <- f_tr3_val - actual_val
    FE_TR_4[[h]][storage_index] <- f_tr4_val - actual_val
    FE_BM[[h]][storage_index]   <- f_bm_val  - actual_val } }

#------------------------------------------------------------------
# 5. RENDER RESULTS MORE INTUITIVE FOR FURTHER ANALYSIS
#------------------------------------------------------------------

# Convert the forecast lists (F_TR_x) into single dataframes
forecast_to_df <- function(forecast_list, period) {
  # Convert each element to numeric (benchmark is ts object, which is bad)
  numeric_list <- lapply(forecast_list, function(x) as.numeric(x)) #just make each list inside numeric
  df <- as.data.frame(numeric_list)
  # Add period and horizon
  df$period <- period #first list is all horizon 1 forecasts, gives this to all observations
  df$horizon <- 1:nrow(df) #counts rows and gives each the horizon corresponding to it
```

```r
  df}

# Apply to all forecasting models
df_all <- do.call(rbind, lapply(seq_along(worker_results), function(i) {
  forecast_to_df(worker_results[[i]], period = i) }))
df_all[] <- lapply(df_all, function(x) as.numeric(x))
```

Spaghetti Plots

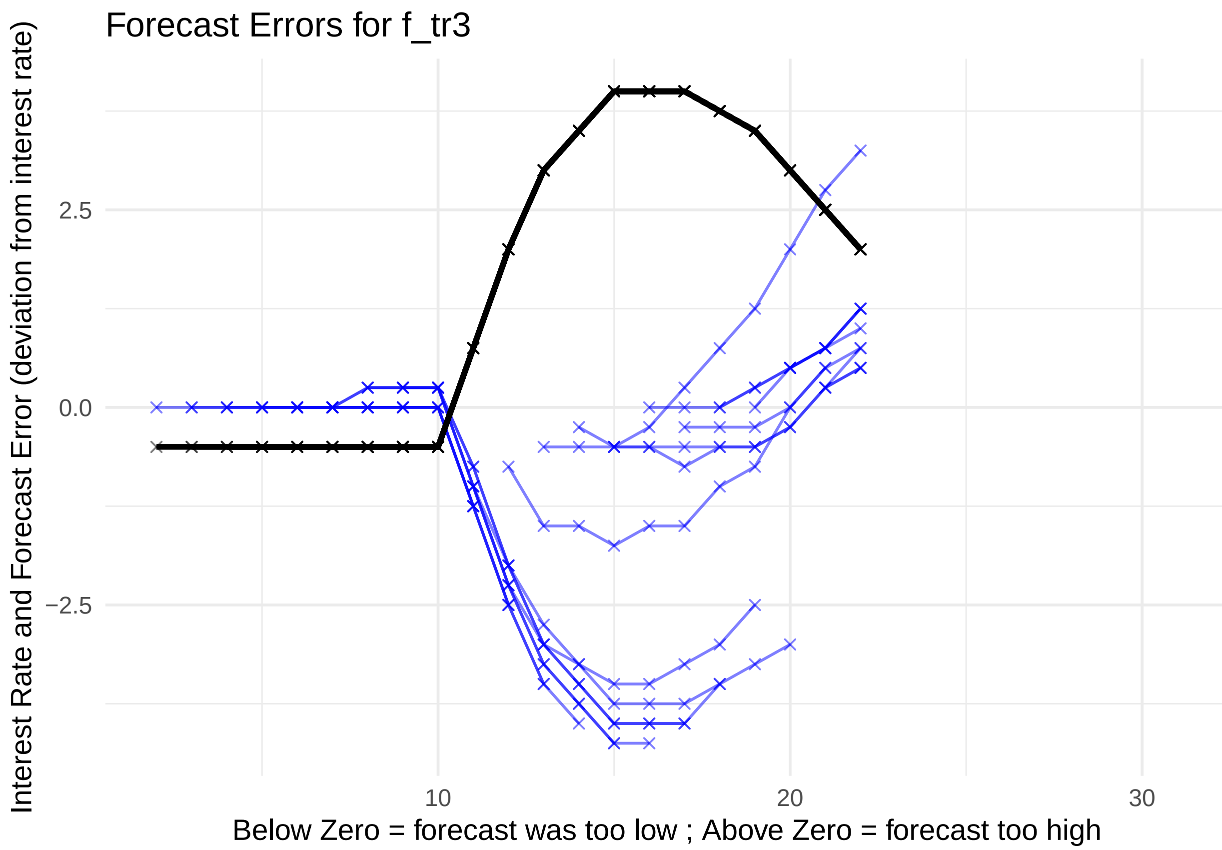
```r
# Select the model to plot
model <- "f_tr3"

# period = date the forecast was made
# date_of_forecast = the future date we are predicting
df_all$date_of_forecast <- df_all$period + df_all$horizon

# Spaghetti plot with color per period
ggplot(df_all, aes(x = date_of_forecast, y = .data[[model]], group = period, color = factor(period))) +
  geom_line(alpha = 0.7) +     # forecast lines
  geom_point(shape = 2, alpha = 0.7) +

  # Actuals as black baseline
  geom_line(aes(y = actuals), color = "black", size = 1) +
  geom_point(aes(y = actuals), color = "black", shape = 4, alpha = 0.5) +

  labs(title = paste("Forecast of", model, "vs Actual Rate"),
       x = "Period",
       y = "Interest Rate",
       color = "Forecast Origin") +
  theme_minimal()
```



Plots of FE

```
df_all_3 <- df_all

# Compute forecast error
df_all_3$forecast_error <- df_all_3[[model]] - df_all_3$actuals

#compute date_of_forecast for x-axis
df_all_3$date_of_forecast <- df_all_3$period + df_all_3$horizon

ggplot(df_all_3, aes(x = date_of_forecast, group = period)) +
  # Forecast error lines
  geom_line(aes(y = forecast_error), color = "blue", alpha = 0.5) +
  geom_point(aes(y = forecast_error), color = "blue", alpha = 0.5, shape = 4) +

  # Actuals line
  geom_line(aes(y = actuals), color = "black", size = 1) +
  geom_point(aes(y = actuals), color = "black", shape = 4, alpha = 0.5) +

  labs(
    title = paste("Forecast Errors for", model),
    x = "Below Zero = forecast was too low ; Above Zero = forecast too high",
    y = "Interest Rate and Forecast Error (deviation from interest rate)"
```

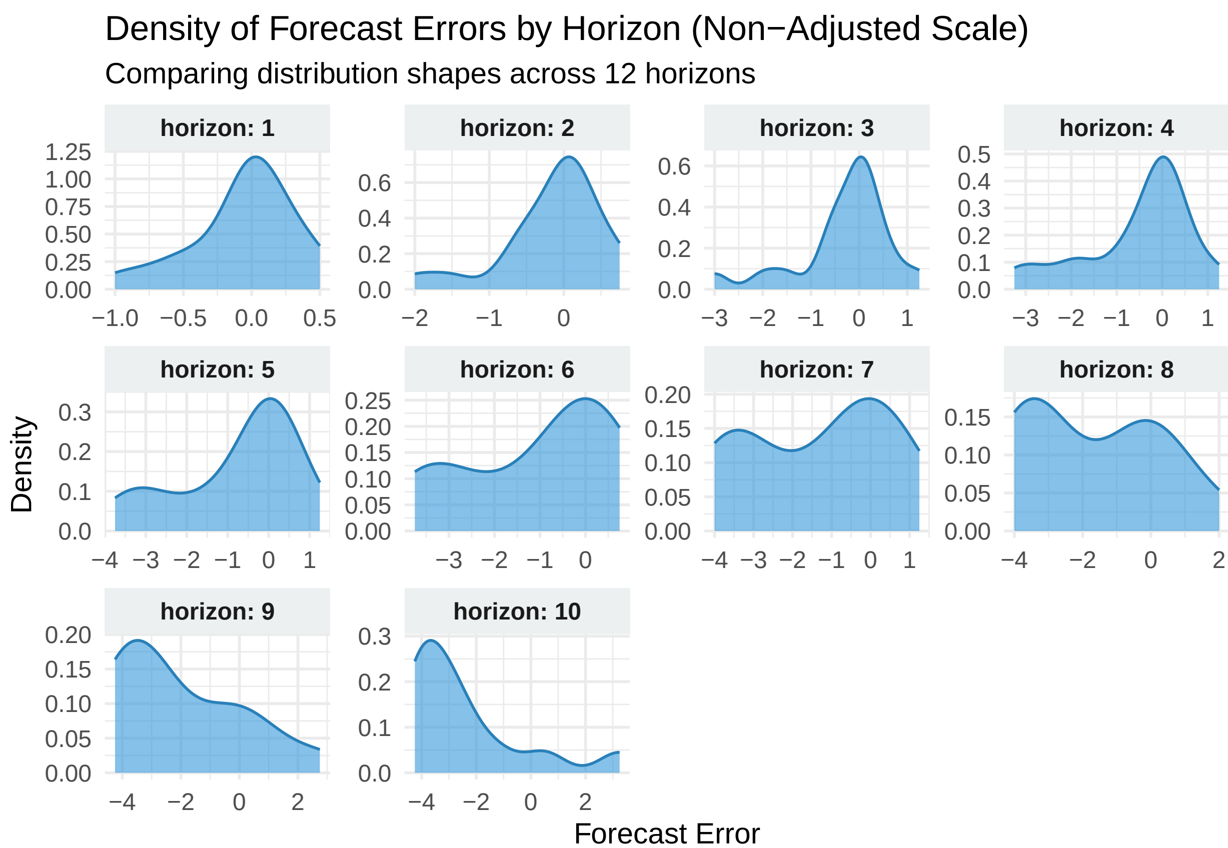
```r
  ) +
  theme_minimal()
```



Density of FE

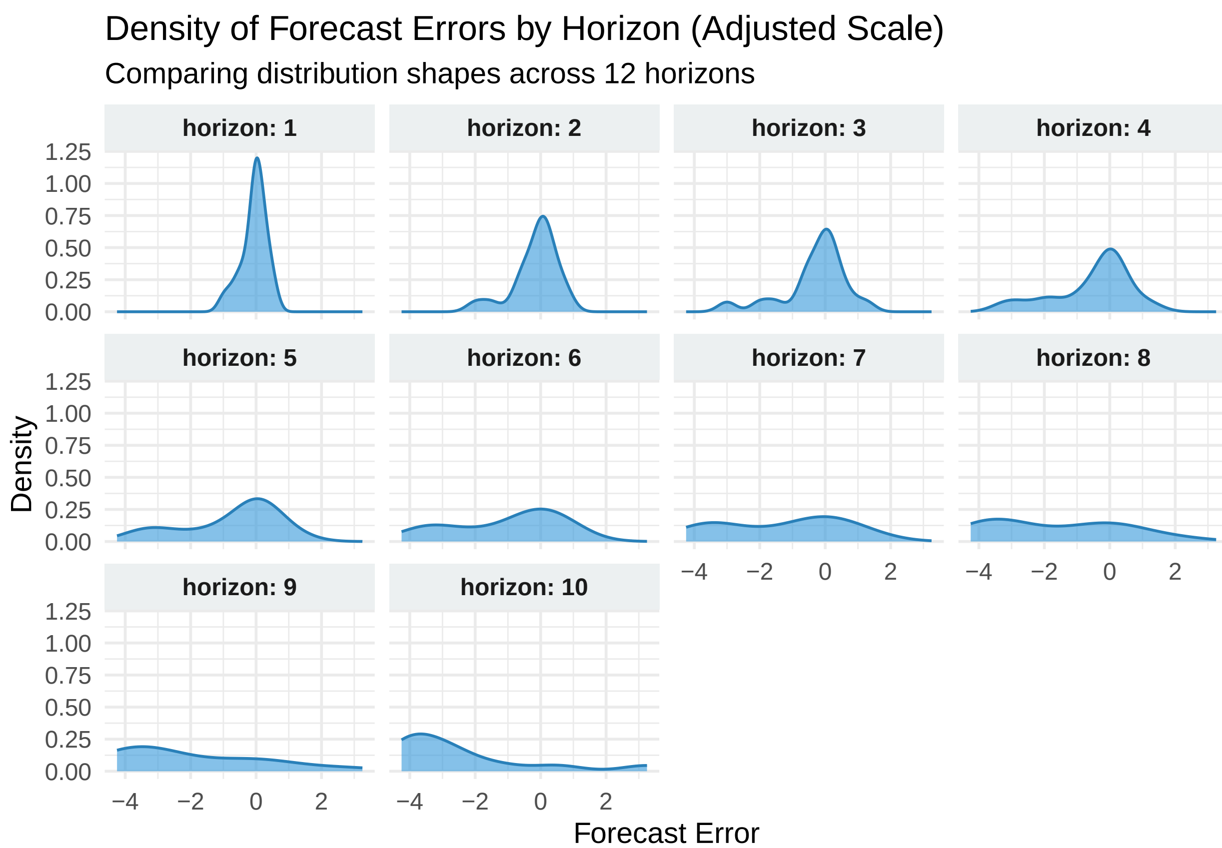
```r
# Version 1: Non-Adjusted Scales
plot_facet <- ggplot(df_all_3 %>% filter(horizon <= H), aes(x = forecast_error)) +
  geom_density(fill = "#3498db", color = "#2980b9", alpha = 0.6) +
  facet_wrap(~horizon, ncol = 4, labeller = label_both, scales = "free") +
  labs(
    title = "Density of Forecast Errors by Horizon (Non-Adjusted Scale)",
    subtitle = "Comparing distribution shapes across 12 horizons",
    x = "Forecast Error",
    y = "Density") +
  theme_minimal() +
  theme(
    strip.background = element_rect(fill = "#ecf0f1", color = NA), # Nice gray boxes for labels
    strip.text = element_text(face = "bold"))
print(plot_facet)
```

```
## Warning: Removed 45 rows containing non-finite outside the scale range
## (`stat_density()`).
```




```r
# Version 2: Adjusted scales
plot_facet <- ggplot(df_all_3 %>% filter(horizon <= H), aes(x = forecast_error)) +
  geom_density(fill = "#3498db", color = "#2980b9", alpha = 0.6) +
  facet_wrap(~horizon, ncol = 4, labeller = label_both) +
  labs(title = "Density of Forecast Errors by Horizon (Adjusted Scale)",
    subtitle = "Comparing distribution shapes across 12 horizons",
    x = "Forecast Error",
    y = "Density") +
  theme_minimal() +
  theme(strip.background = element_rect(fill = "#ecf0f1", color = NA),
    strip.text = element_text(face = "bold"))
print(plot_facet)
```

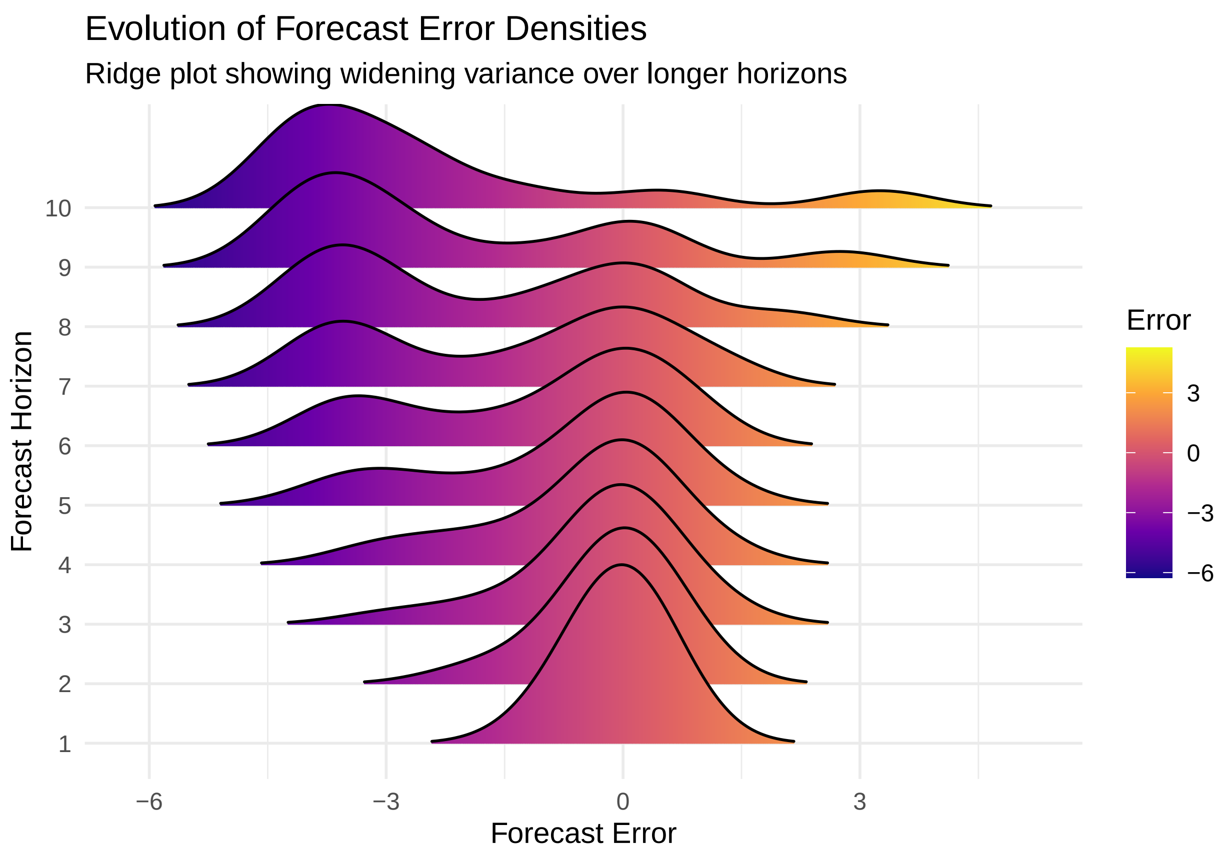
```
## Warning: Removed 45 rows containing non-finite outside the scale range
## (`stat_density()`).
```




```r
# Version 3: "Ridges"
plot_ridge <- ggplot(df_all_3 %>% filter(horizon <= H),
                       aes(x = forecast_error, y = as.factor(horizon), fill = stat(x))) +
    geom_density_ridges_gradient(scale = 3, rel_min_height = 0.01) +
    scale_fill_viridis_c(name = "Error", option = "C") +
    labs(title = "Evolution of Forecast Error Densities",
      subtitle = "Ridge plot showing widening variance over longer horizons",
      x = "Forecast Error",
      y = "Forecast Horizon") +
    theme_minimal()

print(plot_ridge)
```

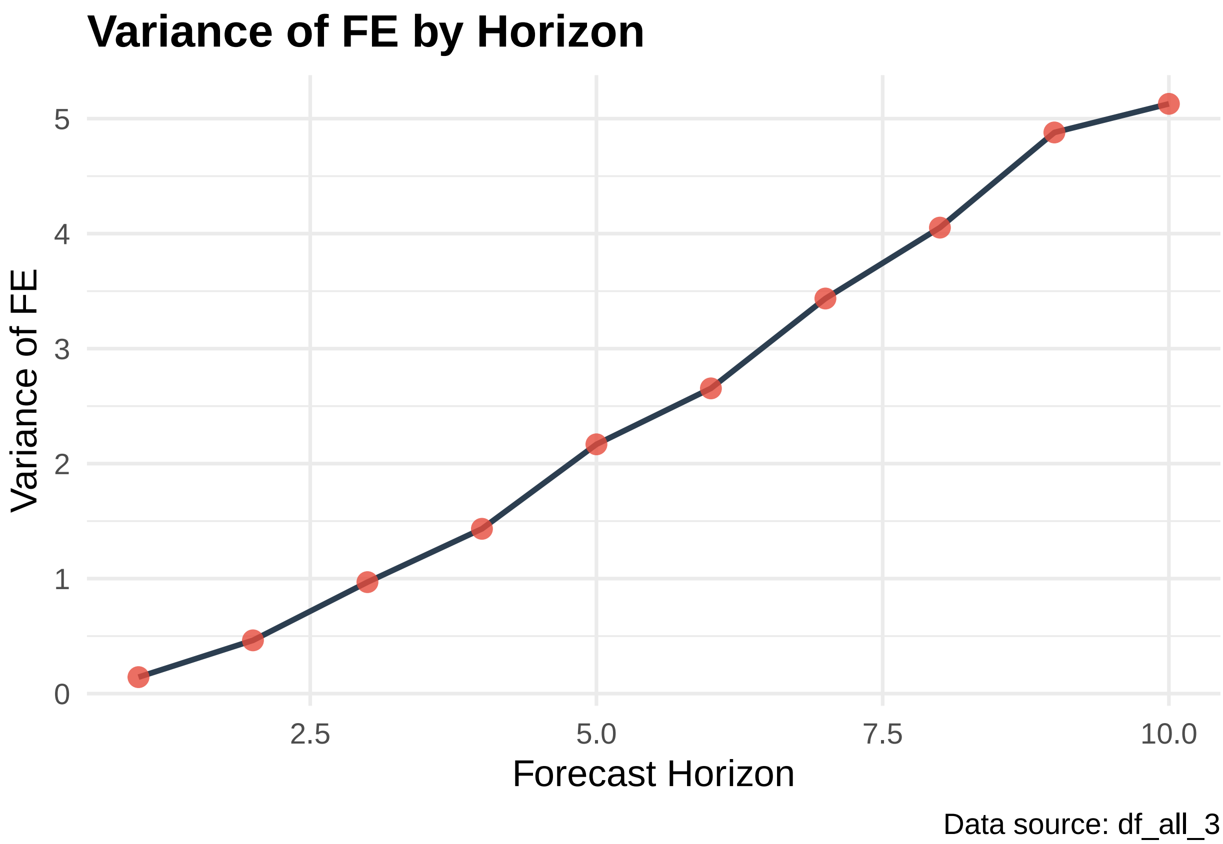
```
## Warning: `stat(x)` was deprecated in ggplot2 3.4.0.
## i Please use `after_stat(x)` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```

```
## Picking joint bandwidth of 0.664
```



Variance of FE

```
# This gives us the mean forecast error for the h step ahead forecast
var_by_horizon <- df_all_3 %>%
  group_by(horizon) %>%
  summarize(
    mean_fe = mean(forecast_error, na.rm=T),
    var_fe = sd(forecast_error, na.rm=T)^2, n = n() )

ggplot(var_by_horizon, aes(x = horizon, y = var_fe)) +
  geom_line(color = "#2c3e50", size = 1) +
  geom_point(color = "#e74c3c", size = 3, alpha = 0.8) +
  theme_minimal(base_size = 14) +
  scale_y_continuous(labels = scales::comma) +
  labs(title = "Variance of FE by Horizon",
    x = "Forecast Horizon",
    y = "Variance of FE",
    caption = "Data source: df_all_3") +
  theme(plot.title = element_text(face = "bold"),
        plot.subtitle = element_text(color = "gray50"),
        panel.grid.minor.x = element_blank() )
```



Absolute Performance: Efficiency & Bias

```
# Call MZ-test helper function 4 times.

# MZ Report 1: Actual Rate, No Lag
mz_report_1 <- generate_mincer_zarnowitz_report(
  F_model = F_TR_1,
  Actual_values = Actuals,
  H = H,
  model_caption = "Mincer-Zarnowitz Test: Actual Rate, No Lag",
  format = format)

# MZ Report 2: Shadow Rate, No Lag (
#  Note: This report is wrapped in trycatch as it sometimes fails
#         If it does fail, simply decrease R in order to have more
#          observations, thus no problem in multicolinearity.
mz_report_2 <- tryCatch({
  generate_mincer_zarnowitz_report(
    F_model = F_TR_2,
    Actual_values = Actuals,
    H = H,
    model_caption = "Mincer-Zarnowitz Test: Shadow Rate, No Lag",
    format = format)}, error = function(e) {
  message("  Error generating MZ Report (Shadow Rate, No Lag): ", e$message)
```

```r
  message(">> Skipping this report and continuing...")
  return(NULL)})

# MZ Report 3: Actual Rate, with Lag
mz_report_3 <- generate_mincer_zarnowitz_report(
  F_model = F_TR_3,
  Actual_values = Actuals,
  H = H,
  model_caption = "Mincer-Zarnowitz Test: Actual Rate, with Lag",
  format = format)

# MZ Report 4: Shadow Rate, with Lag
mz_report_4 <- generate_mincer_zarnowitz_report(
  F_model = F_TR_4,
  Actual_values = Actuals,
  H = H,
  model_caption = "Mincer-Zarnowitz Test: Shadow Rate, with Lag",
  format = format)

# MZ Report 5: Benchmark
mz_report_BM <- generate_mincer_zarnowitz_report(
  F_model = F_BM,
  Actual_values = Actuals,
  H = H,
  model_caption = "Mincer-Zarnowitz Test: Benchmark ARIMA",
  format = format)

list(
  mz_report_1,
  mz_report_2,
  mz_report_3,
  mz_report_4,
  mz_report_BM)
```

[[1]]

[[2]]

[[3]]

[[4]]

[[5]]

Relative Performance (against benchmark)

```r
# Call DM-test helper function 4 times.

# Report 1: Actual Rate, No Lag
report_1 <- generate_report_table(
  FE_TR_model = FE_TR_1,
  FE_BM_model = FE_BM,
  H = H,
```

Table 7: Mincer-Zarnowitz Test: Actual Rate, No Lag

h	Alpha	Beta	pv(Joint)
1	1.1720	0.3204	0.2184
2	1.2241	0.3856	0.7727
3	0.9667	0.8076	0.8783
4	0.7954	1.2455	0.2386
5	0.8566	1.4292	0.0103 **
6	1.0184	1.4755	0.0000 ***
7	1.2284	1.3395	0.0000 ***
8	1.6036	1.0913	0.0000 ***
9	2.2021	0.6248	0.0001 ***
10	2.9629	0.0414	0.0030 ***

Note:
pv(Joint) is the p-value for the joint hypothesis H_0: (Alpha, Beta) = (0, 1). A high p-value means we fail to reject the null hypothesis of an unbiased, efficient forecast.
* Signif. codes: '***' 0.01, '**' 0.05, '*' 0.1

Table 8: Mincer-Zarnowitz Test: Shadow Rate, No Lag

h	Alpha	Beta	pv(Joint)
1	1.8201	-0.3634	0.000 ***
2	1.8580	-0.2293	0.000 ***
3	1.7647	-0.0424	0.000 ***
4	1.8174	0.0116	0.000 ***
5	1.9914	-0.0130	0.000 ***
6	2.2061	-0.0425	0.000 ***
7	2.4557	-0.0697	0.000 ***
8	2.7436	-0.0941	0.000 ***
9	2.9320	-0.0671	0.000 ***
10	3.1602	-0.0452	0.000 ***

Note:
pv(Joint) is the p-value for the joint hypothesis H_0: (Alpha, Beta) = (0, 1). A high p-value means we fail to reject the null hypothesis of an unbiased, efficient forecast.
* Signif. codes: '***' 0.01, '**' 0.05, '*' 0.1

Table 9: Mincer-Zarnowitz Test: Actual Rate, with Lag

h	Alpha	Beta	pv(Joint)
1	0.0568	1.0019	0.7940
2	0.2189	0.9525	0.7692
3	0.4630	0.8861	0.7036
4	0.7537	0.8270	0.6691
5	1.1378	0.6906	0.5505
6	1.4947	0.5933	0.4541
7	1.9075	0.4131	0.1574
8	2.3049	0.2375	0.0148 **
9	2.7110	0.0277	0.0010 ***
10	3.0873	-0.1496	0.0003 ***

Note:
pv(Joint) is the p-value for the joint hypothesis H_0: (Alpha, Beta) = (0, 1). A high p-value means we fail to reject the null hypothesis of an unbiased, efficient forecast.
[*] Signif. codes: '***' 0.01, '**' 0.05, '*' 0.1

Table 10: Mincer-Zarnowitz Test: Shadow Rate, with Lag

h	Alpha	Beta	pv(Joint)
1	0.0533	0.9137	0.0771 *
2	0.1926	0.7985	0.1967
3	0.4409	0.6390	0.0084 ***
4	0.7725	0.4865	0.0000 ***
5	1.1435	0.3537	0.0000 ***
6	1.4715	0.2535	0.0000 ***
7	1.8170	0.1656	0.0000 ***
8	2.2158	0.0884	0.0000 ***
9	2.6636	0.0189	0.0000 ***
10	3.1629	-0.0418	0.0000 ***

Note:
pv(Joint) is the p-value for the joint hypothesis H_0: (Alpha, Beta) = (0, 1). A high p-value means we fail to reject the null hypothesis of an unbiased, efficient forecast.
[*] Signif. codes: '***' 0.01, '**' 0.05, '*' 0.1

Table 11: Mincer-Zarnowitz Test: Benchmark ARIMA

h	Alpha	Beta	pv(Joint)
1	0.1403	0.9439	0.4025
2	0.3980	0.8509	0.1991
3	0.7262	0.7193	0.2696
4	1.0868	0.5906	0.3892
5	1.4665	0.4339	0.2697
6	1.8291	0.3006	0.1350
7	2.1804	0.1495	0.3461
8	2.4846	0.0308	0.0078 ***
9	2.7628	-0.1282	0.0000 ***
10	2.9597	-0.3873	0.0000 ***

Note:
pv(Joint) is the p-value for the joint hypothesis H_0: (Alpha, Beta) = (0, 1). A high p-value means we fail to reject the null hypothesis of an unbiased, efficient forecast.
[*] Signif. codes: '***' 0.01, '**' 0.05, '*' 0.1

```r
  model_caption = "MSFE Comparison, Trained on Actual Rate, No Lag",
  format = format)

# Report 2: Shadow Rate, No Lag
report_2 <- generate_report_table(
  FE_TR_model = FE_TR_2,
  FE_BM_model = FE_BM,
  H = H,
  model_caption = "MSFE Comparison, Trained on Shadow Rate, No Lag",
  format = format)

# Report 3: Actual Rate, with Lag
report_3 <- generate_report_table(
  FE_TR_model = FE_TR_3,
  FE_BM_model = FE_BM,
  H = H,
  model_caption = "MSFE Comparison, Trained on Actual Rate, with Lag",
  format = format)

# Report 4: Shadow Rate, with Lag
report_4 <- generate_report_table(
  FE_TR_model = FE_TR_4,
  FE_BM_model = FE_BM,
  H = H,
  model_caption = "MSFE Comparison, Trained on Shadow Rate, with Lag",
  format = format)

list(report_1, report_2, report_3, report_4)
```

[[1]]

Table 12: MSFE Comparison, Trained on Actual Rate, No Lag

h	MSFE TR	MSFE BM	Ratio	DM Two-Sided	DM Greater	DM Lesser
1	4.0655	0.1637	24.8364	0.0001 ***	0.9999	0.0001 ***
2	4.0938	0.6781	6.0369	0.0355 **	0.9823	0.0177 **
3	3.5461	1.6776	2.1137	0.4047	0.7977	0.2023
4	3.1944	2.8924	1.1044	0.9159	0.5421	0.4579
5	3.1434	4.4596	0.7049	0.5990	0.2995	0.7005
6	3.3438	5.9961	0.5577	0.1229	0.0614 *	0.9386
7	3.6083	7.7500	0.4656	0.0120 **	0.0060 ***	0.9940
8	4.1964	9.0357	0.4644	0.0020 ***	0.0010 ***	0.9990
9	5.0865	10.7260	0.4742	0.0002 ***	0.0001 ***	0.9999
10	6.1823	12.0156	0.5145	0.0000 ***	0.0000 ***	1.0000

Note: TR refers to the forecast made with an estimated Taylor Rule. BM refers to a benchmark of the interest rate using an ARIMA model. Ratio < 1 indicates that the TR model has lower MSFE. *DM Test Alternative Hypotheses (H_A):*
[*] 'DM Greater' tests if the TR model is significantly more accurate than the BM model.
[†] 'DM Lesser' tests if the TR model is significantly less accurate than the BM model.

[[2]]

[[3]]

[[4]]

Table 13: MSFE Comparison, Trained on Shadow Rate, No Lag

h	MSFE TR	MSFE BM	Ratio	DM Two-Sided	DM Greater	DM Lesser
1	8.8750	0.1637	54.2182	0.0001 ***	1.0000	0.0000 ***
2	9.6375	0.6781	14.2120	0.0024 ***	0.9988	0.0012 ***
3	9.6678	1.6776	5.7627	0.0371 **	0.9815	0.0185 **
4	12.3715	2.8924	4.2773	0.2245	0.8878	0.1122
5	17.0404	4.4596	3.8211	0.3261	0.8369	0.1631
6	23.9844	5.9961	4.0000	0.3694	0.8153	0.1847
7	31.9167	7.7500	4.1183	0.3853	0.8073	0.1927
8	43.1384	9.0357	4.7742	0.3166	0.8417	0.1583
9	51.8942	10.7260	4.8382	0.2251	0.8875	0.1125
10	63.5208	12.0156	5.2865	0.0521 *	0.9739	0.0261 **

Note: TR refers to the forecast made with an estimated Taylor Rule. BM refers to a benchmark of the interest rate using an ARIMA model. Ratio < 1 indicates that the TR model has lower MSFE. *DM Test Alternative Hypotheses (H_A):*
[*] 'DM Greater' tests if the TR model is significantly more accurate than the BM model.
[†] 'DM Lesser' tests if the TR model is significantly less accurate than the BM model.

Table 14: MSFE Comparison, Trained on Actual Rate, with Lag

h	MSFE TR	MSFE BM	Ratio	DM Two-Sided	DM Greater	DM Lesser
1	0.1399	0.1637	0.8545	0.5402	0.2701	0.7299
2	0.4625	0.6781	0.6820	0.1832	0.0916 *	0.9084
3	1.0099	1.6776	0.6020	0.1328	0.0664 *	0.9336
4	1.6319	2.8924	0.5642	0.1117	0.0559 *	0.9441
5	2.6250	4.4596	0.5886	0.0449 **	0.0224 **	0.9776
6	3.6172	5.9961	0.6033	0.0143 **	0.0072 ***	0.9928
7	5.0292	7.7500	0.6489	0.0143 **	0.0072 ***	0.9928
8	6.5804	9.0357	0.7283	0.0000 ***	0.0000 ***	1.0000
9	8.5817	10.7260	0.8001	0.0220 **	0.0110 **	0.9890
10	10.5417	12.0156	0.8773	0.1656	0.0828 *	0.9172

Note: TR refers to the forecast made with an estimated Taylor Rule. BM refers to a benchmark of the interest rate using an ARIMA model. Ratio < 1 indicates that the TR model has lower MSFE.
DM Test Alternative Hypotheses (H_A): [*] 'DM Greater' tests if the TR model is significantly more accurate than the BM model.
[†] 'DM Lesser' tests if the TR model is significantly less accurate than the BM model.

Table 15: MSFE Comparison, Trained on Shadow Rate, with Lag

h	MSFE TR	MSFE BM	Ratio	DM Two-Sided	DM Greater	DM Lesser
1	0.1637	0.1637	1.0000	1.0000	0.5000	0.5000
2	0.5781	0.6781	0.8525	0.7065	0.3533	0.6467
3	1.6678	1.6776	0.9941	0.9464	0.4732	0.5268
4	3.7257	2.8924	1.2881	0.3371	0.8315	0.1685
5	7.0993	4.4596	1.5919	0.1935	0.9033	0.0967 *
6	11.6367	5.9961	1.9407	0.0913 *	0.9544	0.0456 **
7	18.5458	7.7500	2.3930	0.1470	0.9265	0.0735 *
8	27.9643	9.0357	3.0949	0.1615	0.9193	0.0807 *
9	40.1971	10.7260	3.7476	0.1525	0.9238	0.0762 *
10	55.4635	12.0156	4.6160	0.1044	0.9478	0.0522 *

Note: TR refers to the forecast made with an estimated Taylor Rule. BM refers to a benchmark of the interest rate using an ARIMA model. Ratio $<$ 1 indicates that the TR model has lower MSFE.

DM Test Alternative Hypotheses (H_A): [*] 'DM Greater' tests if the TR model is significantly more accurate than the BM model.

[†] 'DM Lesser' tests if the TR model is significantly less accurate than the BM model.

Actual Forecast Model

Helpers

```r
#----------------------------------------------------------------------
# Helper function for displaying our final forecast results (table)
#----------------------------------------------------------------------

display_forecasts <- function(forecast_list,
                              caption = "Interest Rate Forecasts",
                              format = "html") {

  # Determine the number of horizons and corresponding quarters
  H <- length(forecast_list$TR_Forecast)

  forecast_quarters <- seq(from = last(data$quarter) + 0.25,
                           by = 0.25,
                           length.out = H)

  horizon_quarter_label <- paste0(1:H, ": ", as.character(forecast_quarters))

  # Create a data frame for display
  forecast_df <- data.frame(
    Horizon_Quarter = horizon_quarter_label,
    Taylor_Rule_Forecast = round(forecast_list$TR_Forecast,2),
    Benchmark_ARIMA_Forecast = round(forecast_list$BM_Forecast,2),
    Inflation_Gap_Forecast = round(forecast_list$Inflation_Forecast,2),
    Output_Gap_Forecast = round(forecast_list$OutputGap_Forecast,2))

  # Create the table
  table_output <- kable(
    forecast_df,
    format = format,
    digits = 4,
    col.names = c("Horizon: Quarter", "Taylor Rule Forecast", "Benchmark Forecast",
                  "Inflation Forecast", "Output Gap Forecast"),
    caption = caption,
    booktabs = TRUE) %>%
  kable_styling(
    latex_options = "striped",
    position = "center") %>%
  column_spec(1, bold = TRUE, border_right = TRUE)
    return(table_output) }



#----------------------------------------------------------------------
# Helper function for plotting our final forecast results
#----------------------------------------------------------------------

plot_forecasts <- function(forecast_list,
                           title = "Interest Rate and Component Forecasts") {
```

```r
# Create the data frame for plotting w/ numerical quarters
H <- length(forecast_list$TR_Forecast)
forecast_quarters_yearqtr <- seq(from = last(data$quarter) + 0.25,
                                 by = 0.25,
                                 length.out = H)

# b) numeric version (for plotting)
forecast_quarters_numeric <- as.numeric(forecast_quarters_yearqtr)

# c) character version (for labels)
forecast_quarters_labels <- as.character(forecast_quarters_yearqtr)

forecast_df <- data.frame(
  Quarter = forecast_quarters_numeric,
  "Taylor Rule" = forecast_list$TR_Forecast,
  "Benchmark ARIMA" = forecast_list$BM_Forecast,
  "Inflation" = forecast_list$Inflation_Forecast,
  "Output Gap" = forecast_list$OutputGap_Forecast,
  check.names = FALSE )

# Long format for ggplot
forecast_long <- forecast_df %>%
  pivot_longer(cols = -Quarter,
    names_to = "Forecast_Type",
    values_to = "Value") %>%
  mutate(Plot_Group = case_when(
         Forecast_Type %in% c("Taylor Rule", "Benchmark ARIMA") ~ "Interest Rate Forecasts",
         Forecast_Type %in% c("Inflation", "Output Gap") ~ "Model Input Forecasts"),
    Plot_Group = factor(Plot_Group, levels = c("Interest Rate Forecasts", "Model Input Forecasts")),
    Forecast_Type = factor(Forecast_Type, levels = c("Taylor Rule", "Benchmark ARIMA", "Inflation", "(

# Actual plot
plot <- ggplot(forecast_long, aes(x = Quarter, y = Value, color = Forecast_Type)) +
  geom_line(linewidth = 1.1) +
  geom_point(size = 2.5) +
  facet_wrap(~ Plot_Group, ncol = 1, scales = "free_y") +
  # Prettyness
  labs(title = title,
       x = "Quarter",
       y = "Value (%)",
       color = "Forecast Series") +
  scale_x_continuous(breaks = forecast_quarters_numeric,
                     labels = forecast_quarters_labels) +
  theme_minimal(base_size = 14) +
  theme(legend.position = "bottom",
        plot.title = element_text(face = "bold", size = 14),
        plot.subtitle = element_text(size = 12),
        strip.text = element_text(face = "bold", size = 12),
        axis.text.x = element_text(angle = 45, hjust = 1)  ) +
        scale_color_brewer(palette = "Set1")
return(plot) }
```

Forecasting

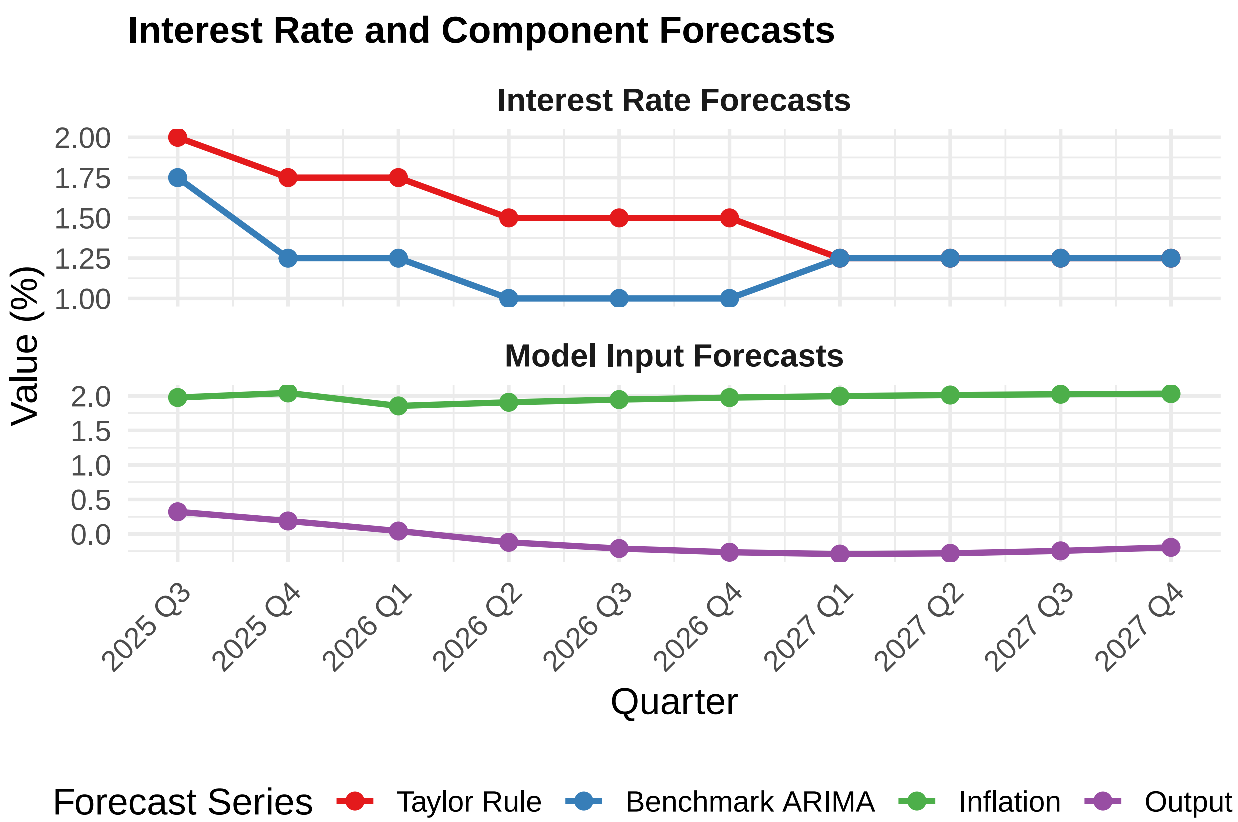
```r
# Formula 4 seems to work best
our_predict <- function(data,formula,H){

  # --- 1. Fit inputs and benchmark models  ---
   inflation_arma <- my.auto.arima(data$inflation_gap, max.p=4, max.q=4, d=1)
   outputgap_arma <- my.auto.arima(data$output_gap, max.p=4, max.q=4, d=0)
   interest_arma <- my.auto.arima(data$rate, max.p=4, max.q=4, d=1) # Benchmark

  # --- 2. Get forecasts of inputs (all H horizons) ---
   inflation_forecasts <- my.forecast(inflation_arma, h = H)
   outputgap_forecasts <- my.forecast(outputgap_arma, h = H)
   BMpredicted_rates <- my.forecast(interest_arma, h = H)

  # --- 3. Fit TR model
   TR_model <- lm(formula, data = data)

  # --- 4. Build forecast input data frame (iteratively for lags) ---

   # Allocate storage for full horizon
   TR_preds <- numeric(H)

   # Get last known lags (starting point for lagged models)
   current_shadowrate_lag <- last(data$shadowrate)
   current_rate_lag <- last(data$rate)

  for (h in 1:H) {
    new_data_h <- data.frame(
       inflation_gap = inflation_forecasts[h],
       #exp_inflation_gap = exp_inflation_forecasts[h],
       output_gap = outputgap_forecasts[h],
       shadowrate_lag = current_shadowrate_lag,
       rate_lag = current_rate_lag)

    # Get forecasted values
    pred_h <- predict(TR_model, new_data_h)
    TR_preds[h] <- round(pmax(pred_h, min(data$rate)) / 0.25) * 0.25

    # Update the lag for h+1
    current_rate_lag <- pred_h }

   # --- 5. Compute forecast for BM ---
    BM_preds <- round(pmax(BMpredicted_rates, min(data$rate)) / 0.25) * 0.25

  return(list(TR_Forecast = TR_preds,
              BM_Forecast = BM_preds,
              Inflation_Forecast = inflation_forecasts + 2, #to add back target
              OutputGap_Forecast = outputgap_forecasts  ))}


final_forecasts <- our_predict(data = data, formula = formula_3, H = H)
```

Table 16

Horizon: Quarter	Taylor Rule Forecast	Benchmark Forecast	Inflation Forecast	Output Gap Forecast
1: 2025 Q3	2.00	1.75	1.98	0.32
2: 2025 Q4	1.75	1.25	2.04	0.19
3: 2026 Q1	1.75	1.25	1.85	0.04
4: 2026 Q2	1.50	1.00	1.91	-0.12
5: 2026 Q3	1.50	1.00	1.95	-0.21
6: 2026 Q4	1.50	1.00	1.98	-0.26
7: 2027 Q1	1.25	1.25	2.00	-0.29
8: 2027 Q2	1.25	1.25	2.01	-0.28
9: 2027 Q3	1.25	1.25	2.02	-0.24
10: 2027 Q4	1.25	1.25	2.03	-0.19

```r
display_forecasts(final_forecasts,
                  caption = "",
                  format = format)
```

```r
plot_forecasts(final_forecasts)
```



Prediction Intervals

```
prediction <- var_by_horizon
final_interval <- final_forecasts
prediction$sd_fe <- sqrt(prediction$var_fe)

final_interval$sd <- prediction$sd_fe
final_interval$upper_1_sd <- final_interval$sd + final_interval$TR_Forecast
final_interval$lower_1_sd <- final_interval$sd*(-1) + final_interval$TR_Forecast

final_interval$upper_2_sd <- final_interval$sd*2 + final_interval$TR_Forecast
final_interval$lower_2_sd <- final_interval$sd*2*(-1) + final_interval$TR_Forecast

final_interval <- as.data.frame(final_interval)
```

```
ggplot(final_interval, aes(x = seq_len(nrow(final_interval)), y = TR_Forecast, group = 1)) +
  geom_ribbon(aes(ymin = lower_1_sd, ymax = upper_1_sd), fill = "lightgrey", alpha = 0.6) + # intervall
  geom_ribbon(aes(ymin = lower_2_sd, ymax = upper_2_sd), fill = "lightgrey", alpha = 0.2) + # intervall
  geom_line(color = "#1f78b4", size = 1) +  # line
  geom_point(color = "#e31a1c", size = 3) + # points
  labs(title = "Prelinimary Forecast plot", x = "h step ahead") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```