ECB Project

Contents

Preliminaries

Setup

```r
# Clear memory
rm(list=ls())

# Load here package to enable finding script loading other packages
require(here)

# Set directory
getwd()
setwd("...")

# Load packages & helper functions THAT WON'T BE IN THE FINAL MARKDOWN
source(here("helpers/packages.R"))
source(here("helpers/auto_ARIMA_replic.R"))
source(here("helpers/stationarity_tests.R"))
source(here("helpers/roll_TR_plotter.R"))
source(here("helpers/pseudo_outofsample_tables.R"))
source(here("helpers/actual_forecast_helpers.R"))


# Api key for data
fredr_set_key("e0169694a62c1337f1969e3872605eca")

# Dates (to automatically get the latest data from API calls)
start_date <- "1999-01-01"
end_date <- Sys.Date()

# For replication
set.seed(2025)
```

Interactive Option Selection

- Use Hamilton Filter:
  - TRUE: Selects Hamilton method for output gap estimation
  - FALSE: Selects Hodrick-Prescott method for output gap estimation
- Use Inflation Expectations:
  - TRUE: The models used for forecasting will use 12-month ahead inflation expectations from the ECB survey of professional forecasts (average).
  - FALSE: The models used for forecasting will use realised inflation
- Use Formula
  - Formula 1: Actual interest rate regressed on inflation and output gaps

  - Formula 2: Shadow interest rate regressed on inflation and output gaps
  - Formula 3: Actual interest rate regressed on the one-quarter lag of the interest rate and on inflation and output gaps

    - Formula 4: Shadow interest rate regressed on the one-quarter lag of the shadow interest rate and on inflation and output gaps

- Format:
    - html: For outputting in console or knitting to html
    - latex: For knitting to pdf

```
# Related to Analysis
USE_HAMILTON_FILTER <- TRUE
USE_INFLATION_EXPECTATIONS <- FALSE
USE_FORMULA <- "Formula 3"

# Related to Document Output
format <- "html"
format <- "latex"
save_figures <- FALSE #note: format has to be set to latex
```

Data

Sources & Explanations

- FRED Data Source: European Central Bank, ECB Deposit Facility Rate for Euro Area [ECBDFR], retrieved from FRED, Federal Reserve Bank of St. Louis; https://fred.stlouisfed.org/series/ECBDFR. The data is the Deposit Facility and is directly reprinted from the ECB. It's in Percent and not seasonally adjusted. The ECB Monetary Policy is steered through this rate
- Shadow Interest Rate: The Shadow rate was developed by **wuTimeVaryingLowerBound2017** and does quantify a hypothetical removal of the ZLB. The rate does not track 1:1 on the deposit facility in the data before the ZLB, instead being closer to the refinancing rate. (maybe need to adjust)
- GDP: The GDP Data is quarterly real GDP in 2010 Euros in million retrieved from FRED via Eurostat. The data is seasonally adjusted.
- Potential GDP: Either estimated with the Hodrick-Prescott filter or the Hamilton filter, based on the formula below:

$$\text{HP Filter:}\quad \min_{\tau}\left(\sum_{t=1}^{T}(y_t-\tau_t)^2+\lambda\sum_{t=2}^{T-1}\left[(\tau_{t+1}-\tau_t)-(\tau_t-\tau_{t-1})\right]^2\right)$$

$$\text{Hamilton:}\quad y_t=\beta_0+\sum_{j=1}^{p}\beta_j y_{t-h-j+1}+v_t\quad(\text{where } v_t \text{ is the cycle})$$

- Inflation:
  - Realised: The Inflation data is from Eurostat and measures HICP monthly data (annual rate of change). It's the index that the ECB uses for Inflation and is not seasonally adjusted, but the fact that it represents the year-on-year change in prices implies there is no seasonality.
  - Expectations: Expected Inflation is the ECB's survey of professional forecasters. It forecasts the HICP 12 months in advance.

Loading & Preparation Data

```r
# --- 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)) %>%
  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 Q4")

# 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)
```

Options 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
  cat("* CONFIGURATION: Using Hamilton Filter for output gap estimation.")
} else {
  data$output_gap <- data$output_gap_hp
  cat("* CONFIGURATION: Using HP Filter for output gap estimation.") }
```

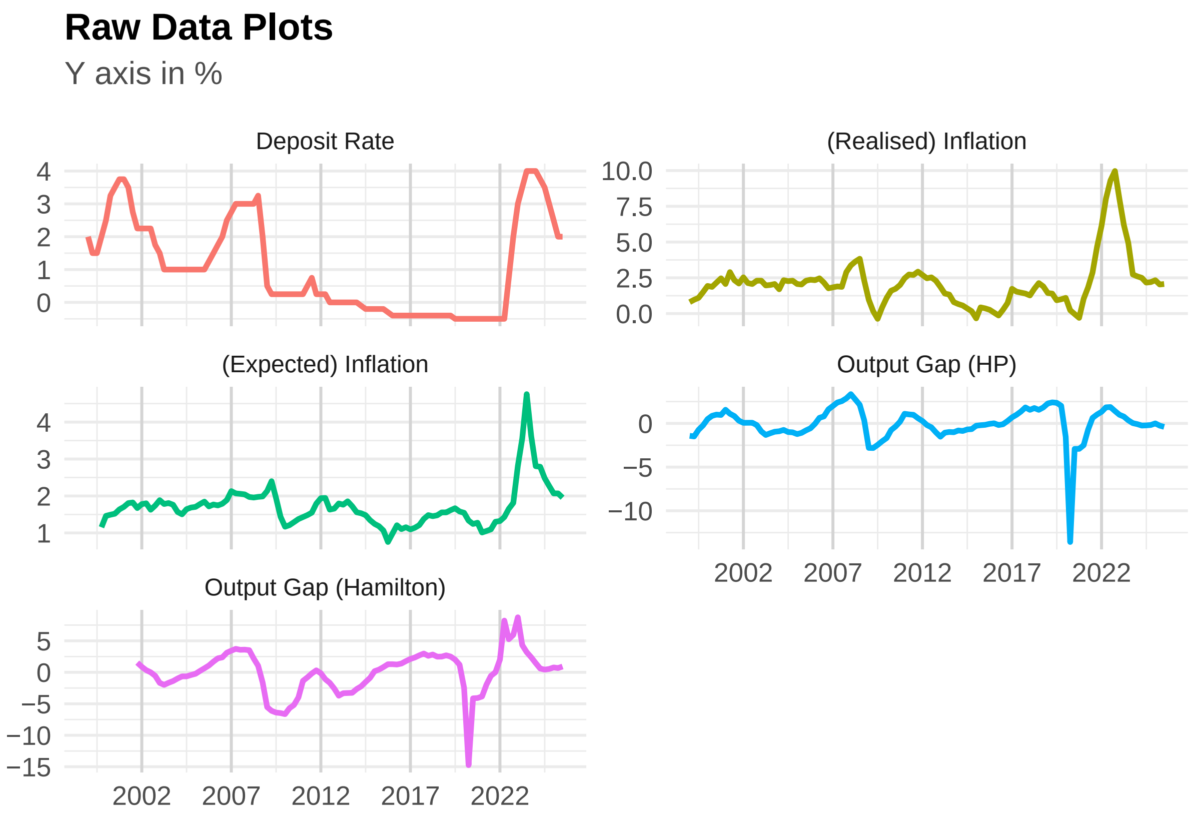
- CONFIGURATION: Using Hamilton Filter for output gap estimation.

```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
  cat("* CONFIGURATION: Using inflation expectations in Taylor Rule forecasting.")
} else {
  data$inflation_gap <- data$realised_inflation_gap
  cat("* CONFIGURATION: Using realised inflation in Taylor Rule forecasting.") }
```

- CONFIGURATION: Using realised inflation in Taylor Rule forecasting.

Raw Data Plots

```
# Data must be in long format for a faceted plot
plot_data <- data %>%
  pivot_longer(cols = c(rate, inflation, exp_inflation, output_gap_hp, output_gap_ham),
               names_to = "series",
               values_to = "value") %>%
  mutate(series = factor(series,
    levels = c("rate", "inflation", "exp_inflation", "output_gap_hp", "output_gap_ham"),
    labels = c("Deposit Rate", "(Realised) Inflation", "(Expected) Inflation", "Output Gap (HP)", "Outpu

raw_plot = ggplot(plot_data, aes(x = date, y = value)) +
  geom_line(aes(color = series), linewidth = 1) +
  facet_wrap(~ series, scales = "free_y", ncol = 2) +
  scale_x_date(date_breaks = "5 years", date_labels = "%Y") +
  labs(title = "Raw Data Plots", subtitle = "Y axis in %", x = "", y = "") +
  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)),
     axis.title = element_text(size = 12),
     axis.text = element_text(size = 10),
     legend.position = "none",
     panel.grid.major.x = element_line(linewidth = 0.525, color = "grey83"))
raw_plot
```



```r
if (save_figures) {
  ggsave(filename = "raw_data_plot.png", path = "figures/", plot = raw_plot)}
rm(plot_data)
```

Data Properties

```r
# -----  Run Tests -----

# Use helper to run stationarity checks for our main variables
test_rate <- check_stationarity(data$rate, "Interest Rate")
test_inflation <- check_stationarity(data$inflation, "Inflation")
test_output_gap <- check_stationarity(na.omit(data$output_gap), "Output Gap")

# Given results for rate and inflation, run tests on 1st diffs
test_rate_diff <- check_stationarity(diff(data$rate), "Interest Rate (1st Diff)")
test_inflation_diff <- check_stationarity(diff(data$inflation), "Inflation (1st Diff)")

# Since rate and inflation are I(1), run a cointegration test
coint_test = check_coint(data$rate, data$inflation,
            var_name1 = "Interest Rate", var_name2 = "Inflation")


# ----- Print Results -----

# Combine stationarity results
all_stationarity_results <- rbind(test_rate,
                                  test_inflation,
                                  test_output_gap,
                                  test_rate_diff,
                                  test_inflation_diff)

# Tables
stat_table <- kable(all_stationarity_results, digits = 4, format = format, booktabs = TRUE,
      caption = "Summary of Stationarity Tests (ADF \\& KPSS)") %>%
      kable_styling(latex_options = "scale_down",
      position = "center") %>%
        column_spec(1, border_right = TRUE) %>%
        column_spec(4, width = "6cm")
stat_table
```

```r
if (save_figures) {
  save_kable(stat_table, "figures/staionarity_results.tex")}

coint_table <- kable(coint_test, digits = 4, format = format, booktabs = TRUE,
      caption = "Cointegration Test Results") %>%
      kable_styling(latex_options = "scale_down",
      position = "center") %>%
        column_spec(1, border_right = TRUE)
coint_table
```

Table 1: Summary of Stationarity Tests (ADF & KPSS)

Variable	ADF p-value	KPSS p-value	Result
Interest Rate	0.4350	0.0359	Non-Stationary I(1): ADF confirms unit root, KPSS rejects stationarity.
Inflation	0.2257	0.1000	Conflicting Results: ADF confirms unit root, while KPSS suggests stationarity.
Output Gap	0.0131	0.1000	Stationary I(0): ADF rejects unit root, KPSS confirms stationarity.
Interest Rate (1st Diff)	0.0100	0.1000	Stationary I(0): ADF rejects unit root, KPSS confirms stationarity.
Inflation (1st Diff)	0.0100	0.1000	Stationary I(0): ADF rejects unit root, KPSS confirms stationarity.

Table 2: Cointegration Test Results

Variables	p-value	Result
Interest Rate & Inflation	0.0585	Not Cointegrated

```r
if (save_figures) {
  save_kable(coint_table, "figures/cointegration_results.tex")}
```

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"
if (save_figures) {
  huxtable::quick_latex(table1, file = "figures/TR_no_lag_no_expectations.tex")}
table1
```

Table 3: No Lag, No Expectations

	TR	TR w/ SR
(Intercept)	0.8460	-0.8296
	(0.8943)	(2.6530)
realised_inflation_gap	0.1946	0.6812
	(0.4318)	(0.6058)
output_gap	0.0839	-0.0528
	(0.1632)	(0.2763)
N	96	96
R2	0.1722	0.1146

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

```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"
if (save_figures) {
  huxtable::quick_latex(table2, file = "figures/TR_no_lag_w_expectations.tex")}
table2
```

Table 4: No Lag, with Inflation Expectations

	TR	TR w/ SR	TR	TR w/ SR
(Intercept)	1.3522 ***	0.3098	1.3470 ***	0.1984
	(0.3169)	(1.5706)	(0.3707)	(1.6114)
exp_inflation_gap	1.7282 ***	3.7705 **	1.7165 ***	3.5220 **
	(0.3556)	(1.3951)	(0.3483)	(1.1795)
output_gap	0.0711	-0.0015	0.0671	-0.0872
	(0.0469)	(0.1872)	(0.0811)	(0.2131)
realised_inflation_gap			0.0146	0.3120
			(0.1278)	(0.3423)
N	96	96	96	96
R2	0.6180	0.3914	0.6182	0.4089

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

Lagged Models

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

```
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"
if (save_figures) {
  huxtable::quick_latex(table3, file = "figures/TR_w_lag_no_expectations.tex")}
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"
if (save_figures) {
  huxtable::quick_latex(table4, file = "figures/TR_w_lag_w_expectations.tex")}
table4
```

Table 5: Interest Rate Lag, No Expectations

	TR	TR w/ SR
(Intercept)	0.0512	-0.0756
	(0.0406)	(0.0592)
rate_lag	0.9195 ***	
	(0.0381)	
realised_inflation_gap	0.0877 *	0.2493 ***
	(0.0348)	(0.0343)
output_gap	0.0219	-0.0113
	(0.0163)	(0.0184)
shadowrate_lag		0.9535 ***
		(0.0173)
N	96	96
R2	0.9597	0.9828

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

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
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)

chow_table <- kable(chow_df, digits = 4, format = format, booktabs = TRUE,
```

Table 6: Interest Rate Lag, with Inflation Expectations

	TR	TR w/ SR	TR	TR w/ SR
(Intercept)	0.1805 *	0.0012	0.1343 ***	-0.0876
	(0.0791)	(0.0852)	(0.0376)	(0.0594)
rate_lag	0.8612 ***		0.8750 ***	
	(0.0436)		(0.0381)	
exp_inflation_gap	0.2381 **	0.1295	0.1530 ***	-0.0548
	(0.0734)	(0.1038)	(0.0434)	(0.0618)
output_gap	0.0449 *	0.0592	0.0234	-0.0106
	(0.0218)	(0.0474)	(0.0165)	(0.0181)
shadowrate_lag		0.9631 ***		0.9586 ***
		(0.0287)		(0.0187)
realised_inflation_gap			0.0768 *	0.2528 ***
			(0.0378)	(0.0346)
N	96	96	96	96
R2	0.9547	0.9714	0.9614	0.9829

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

```r
      caption = "Chow tests for suspected structural breaks") %>%
      kable_styling(latex_options = "scale_down",
      position = "center") %>%
        column_spec(1, border_right = TRUE)
chow_table
```

```r
if (save_figures) {
  save_kable(chow_table, "figures/chow_breaks.tex")}
```

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

Table 7: Chow tests for suspected structural breaks

Event	Date	p-value
ZLB Start	2012 Q3	0.0018
COVID-19 Start	2020 Q1	0.1230

Table 8: Bai-Perron test for multiple breaks

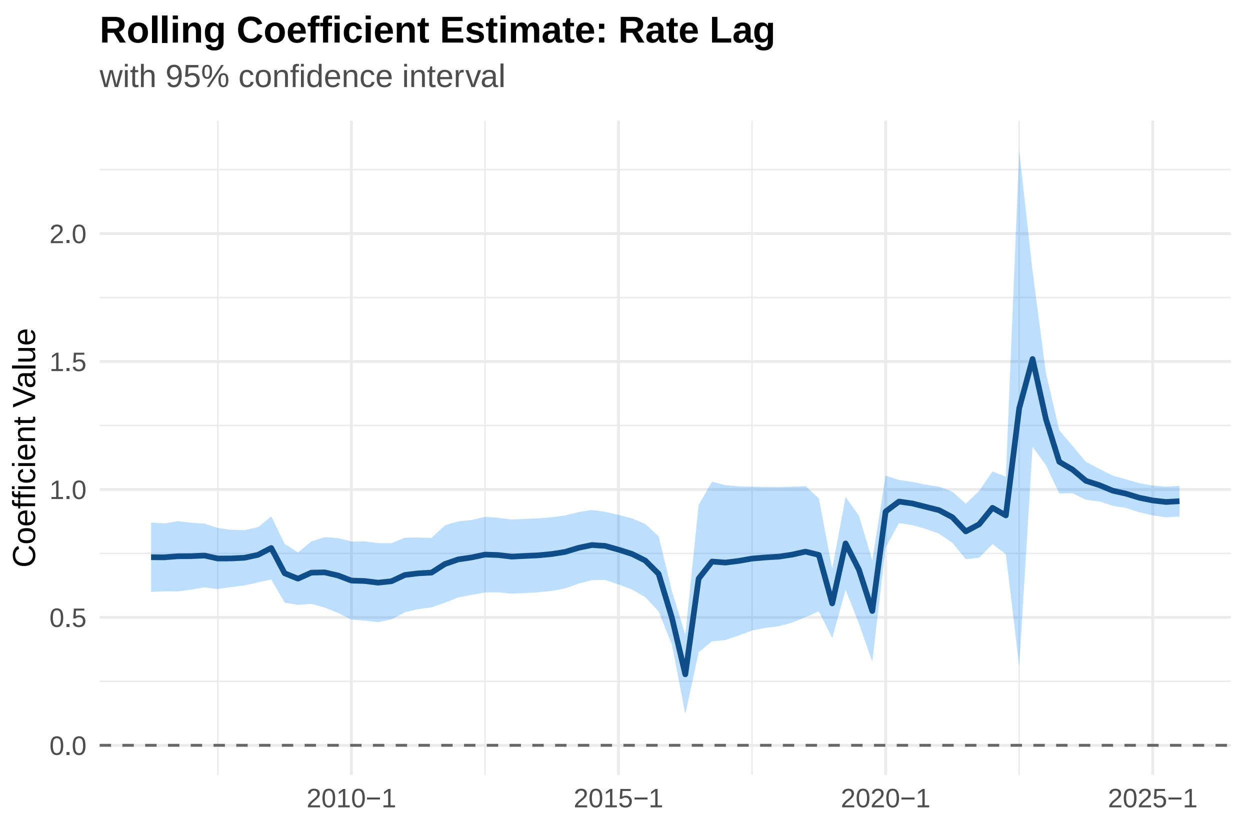
Detected Breaks
2006 Q2
2019 Q2

```r
# 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 Breaks` = "No structural breaks detected",
    check.names = FALSE)
  } else {
  break_obs <- na.omit(BP_test_res$breakpoints[optimal_m, ])
  detected_dates <- data$quarter[break_obs]
  bp_df <- data.frame(
    `Detected Breaks` = as.character(detected_dates),
    check.names = FALSE) }

bp_table <- kable(bp_df, format = format, booktabs=TRUE,
  caption = "Bai-Perron test for multiple breaks") %>%
  kable_styling(latex_options = "scale_down",
      position = "center")
bp_table
```

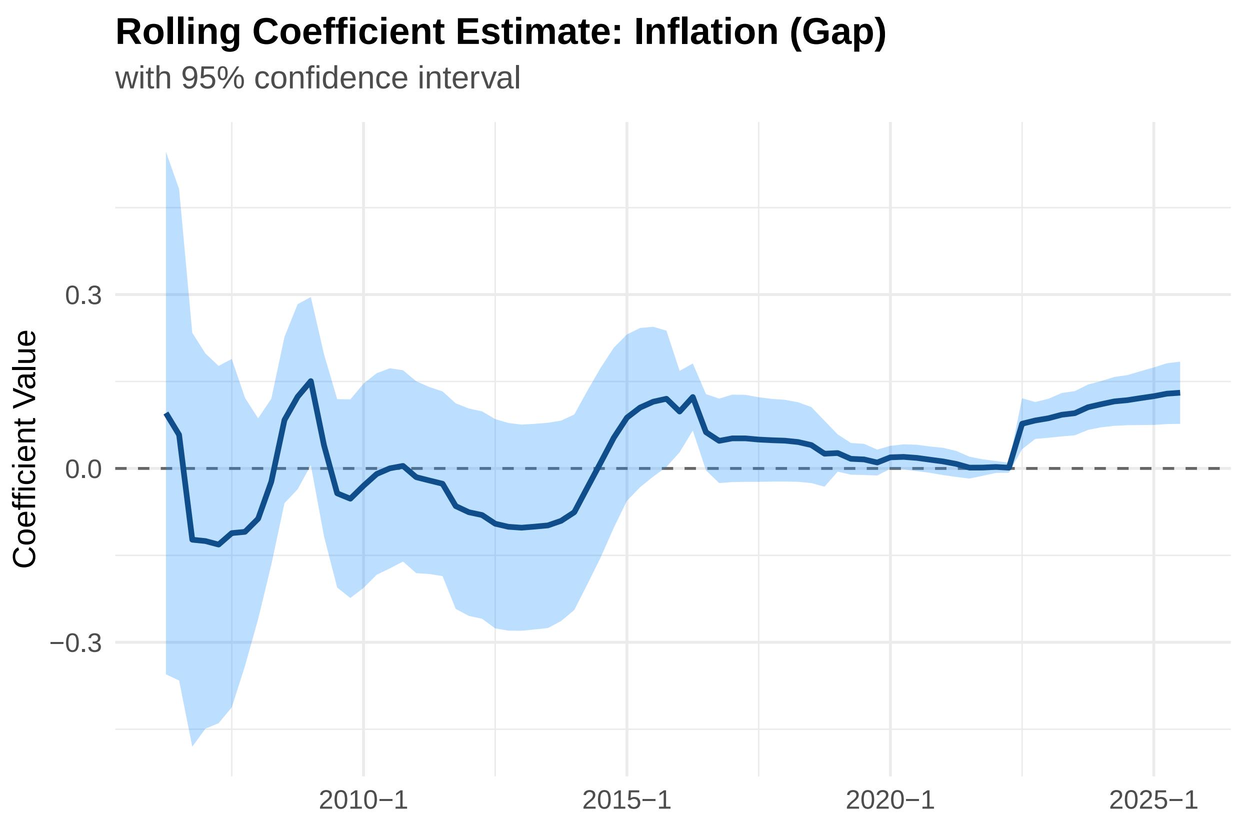
```r
if (save_figures) {
  save_kable(bp_table, "figures/bp_breaks.tex")}
# Note: breaks_obs shows in which row the BP breaks are
```

Rolling Estimation (for structural breaks)

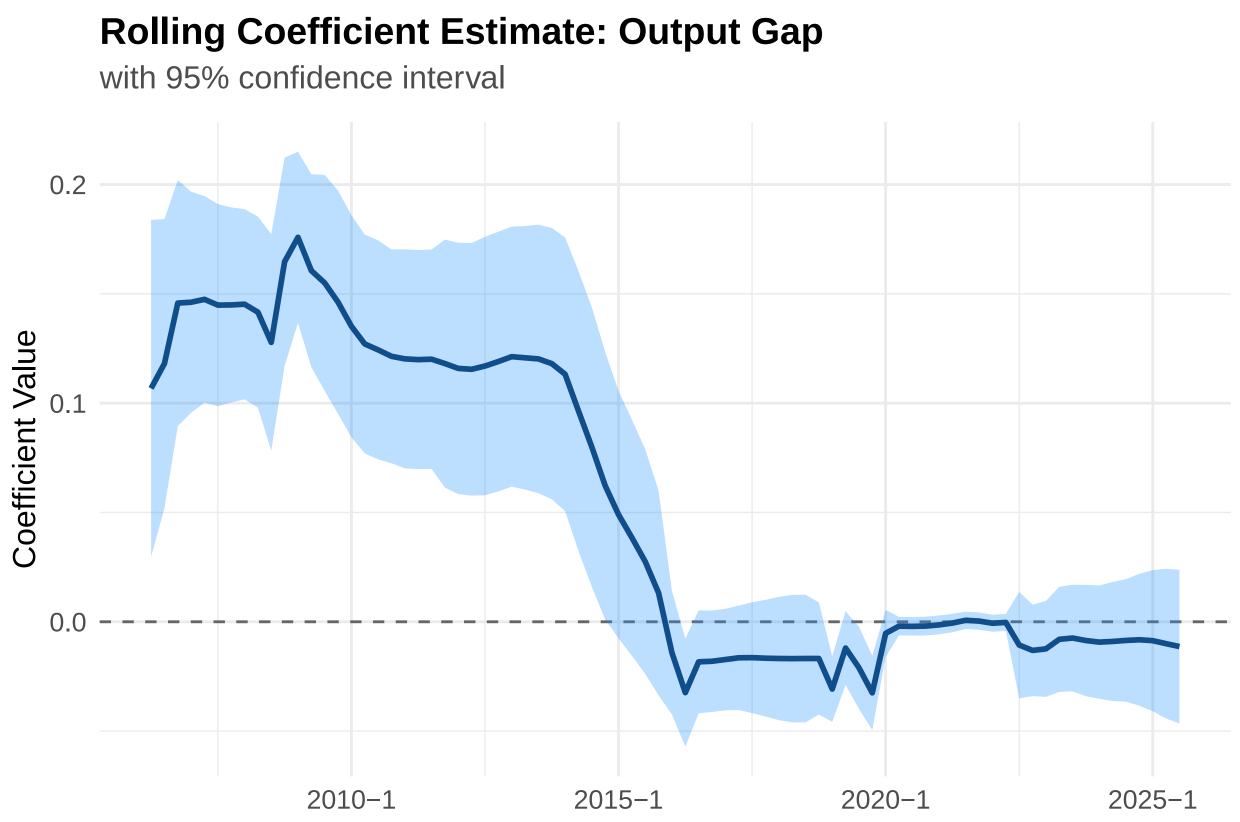
```r
# Set rolling window (in quarters) & Looping Parameter
W = 30
L = nrow(data) - W + 1
formula = rate ~ rate_lag + inflation_gap + output_gap

# 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

Pseudo Out of Sample Estimation

Pseudo-out of sample rolling estimation scheme of direct forecasts for all Taylor Rule formulas and a benchmark ARIMA specification.

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

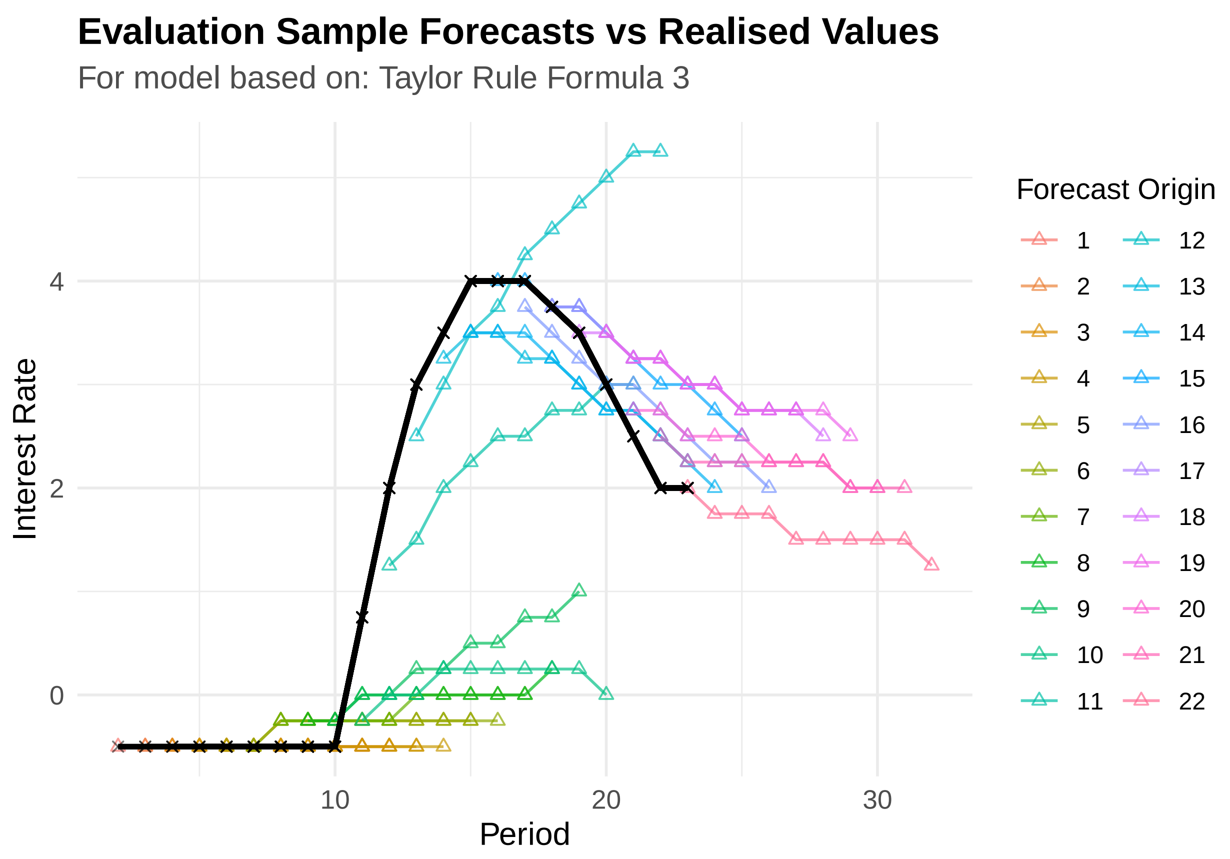
Evaluation sample starts after 2020 Q1.

```
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)

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

```r
FE_BM   <- init_storage_list(H, P) # Benchmark: ARIMA


#-------------------------------------------------------------------
# 3. SETUP & RUN THE PARALLEL BACKTESTING LOOP
#-------------------------------------------------------------------
num_cores <- detectCores() / 4
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)

  # --- 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_TR_FORMULA_1 = TR_preds_1,
       F_TR_FORMULA_2 = TR_preds_2,
       F_TR_FORMULA_3 = TR_preds_3,
       F_TR_FORMULA_4 = 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_TR_FORMULA_1[h]
    f_tr2_val  <- p_results$F_TR_FORMULA_2[h]
    f_tr3_val  <- p_results$F_TR_FORMULA_3[h]
    f_tr4_val  <- p_results$F_TR_FORMULA_4[h]
    f_bm_val   <- p_results$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
  df}

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

Spaghetti Plots

```r
# Select the model to plot based on initial option
if (USE_FORMULA == "Formula 1") {
  model <- "F_TR_FORMULA_1"
  model_formula <- formula_1
  model_name <- "Taylor Rule Formula 1"
} else if (USE_FORMULA == "Formula 2") {
  model <- "F_TR_FORMULA_2"
  model_formula <- formula_2
  model_name <- "Taylor Rule Formula 2"
} else if (USE_FORMULA == "Formula 3") {
  model <- "F_TR_FORMULA_3"
  model_formula <- formula_3
  model_name <- "Taylor Rule Formula 3"
} else if (USE_FORMULA == "Formula 4") {
  model <- "F_TR_FORMULA_4"
  model_formula <- formula_4
  model_name <- "Taylor Rule Formula 4" }

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

# Spaghetti plot with color per period
spag_plot1 <- ggplot(eval_all_models, aes(x = date_of_forecast, y = .data[[model]], group = period, col
  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 = "Evaluation Sample Forecasts vs Realised Values",
       x = "Period",
       y = "Interest Rate",
       color = "Forecast Origin",
    subtitle = paste("For model based on:", model_name)) +
  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)),
      axis.title = element_text(size = 12),
      axis.text = element_text(size = 10))
spag_plot1
```

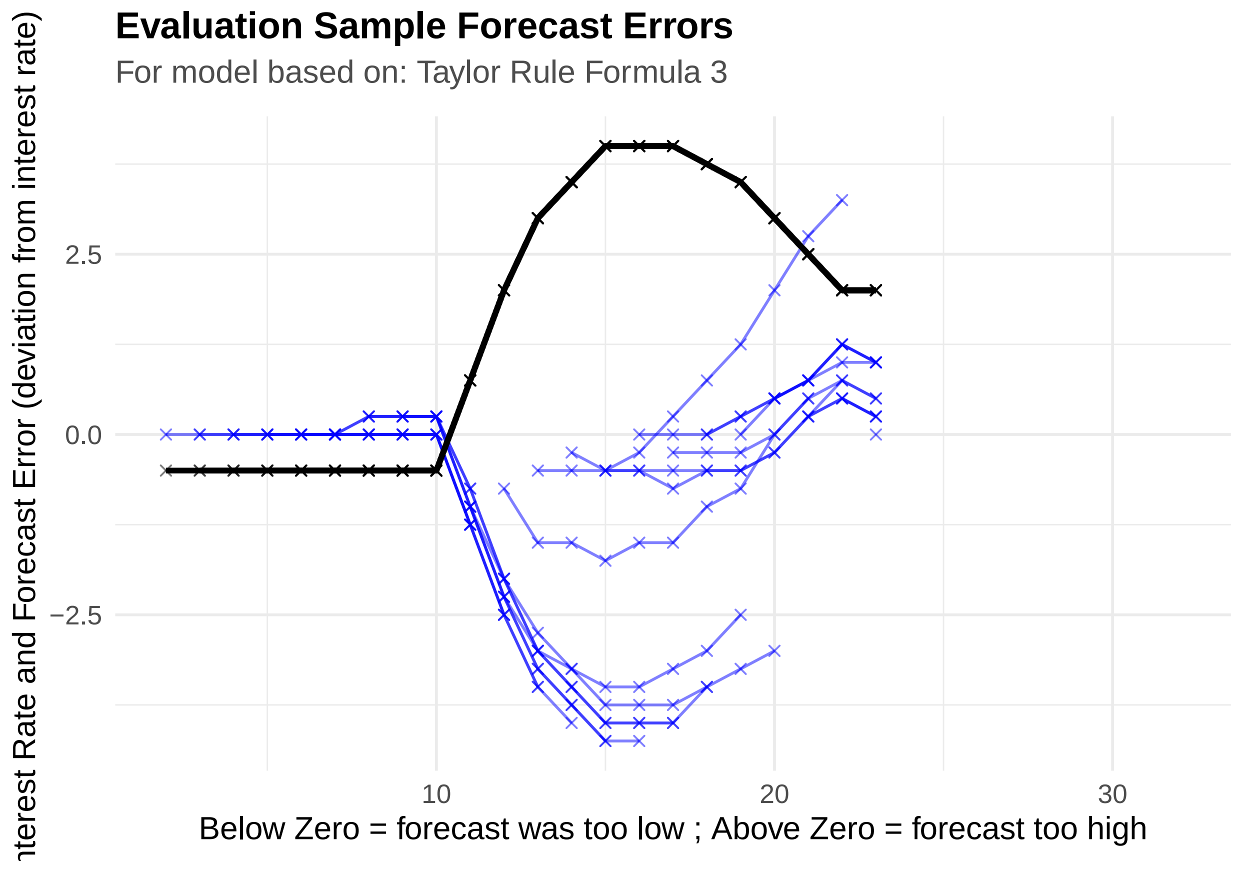


```r
if (save_figures) {
  ggsave(filename = "spaghetti_plot.png", path = "figures/", plot=spag_plot1)}
```

Forecast Errors

Plots of FE

```r
# Compute forecast error
eval_all_models$forecast_error <- eval_all_models[[model]] - eval_all_models$actuals

# Compute date_of_forecast for x-axis
eval_all_models$date_of_forecast <- eval_all_models$period + eval_all_models$horizon

spag_plot2 <- ggplot(eval_all_models, 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) +
  # Actual rates line
  geom_line(aes(y = actuals), color = "black", size = 1) +
  geom_point(aes(y = actuals), color = "black", shape = 4, alpha = 0.5) +
  labs(title = "Evaluation Sample Forecast Errors",
       x = "Below Zero = forecast was too low ; Above Zero = forecast too high",
       y = "Interest Rate and Forecast Error (deviation from interest rate)",
    subtitle = paste("For model based on:", model_name))  +
  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)),
      axis.title = element_text(size = 12),
      axis.text = element_text(size = 10))
spag_plot2
```

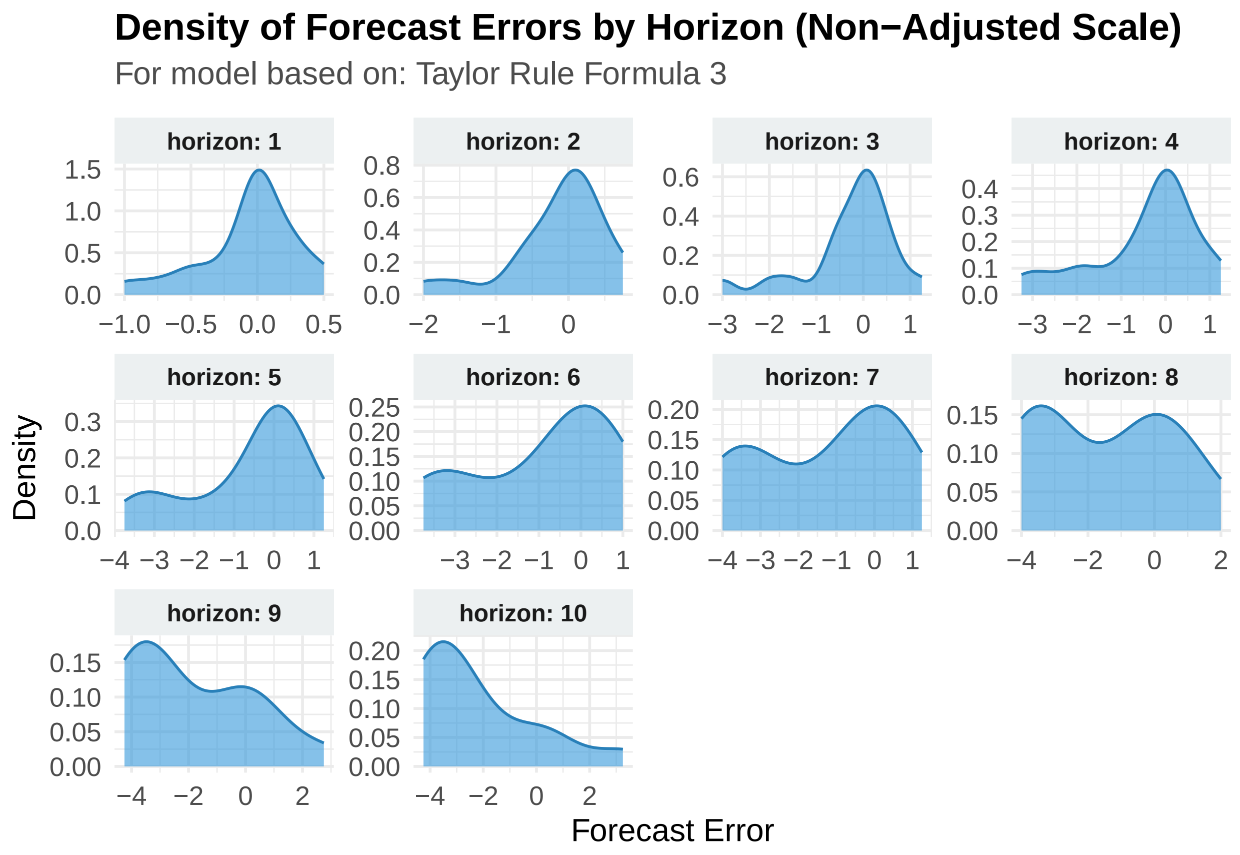


```
if (save_figures) {
  ggsave(filename = "spaghetti_errors_plot.png", path = "figures/", plot=spag_plot2)}
```

Density of FE

```
# Version 1: Non-Adjusted Scales
plot_facet1 <- ggplot(eval_all_models %>% 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 = paste("For model based on:", model_name),
       x = "Forecast Error",
       y = "Density") +
  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)),
        axis.title = element_text(size = 12),
        axis.text = element_text(size = 10),
        strip.background = element_rect(fill = "#ecf0f1", color = NA),
        strip.text = element_text(face = "bold"))
print(plot_facet1)
```

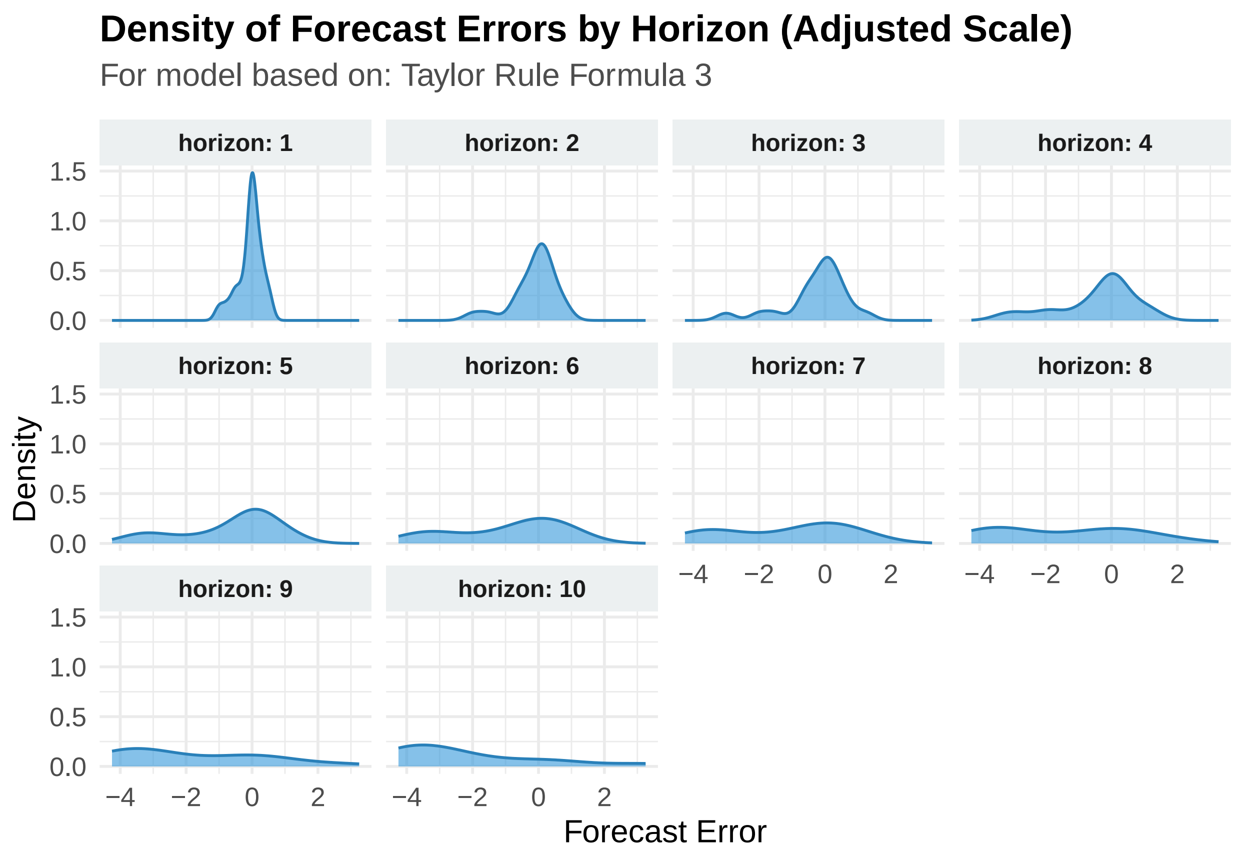
```
## Warning: Removed 45 rows containing non-finite outside the scale range
## (`stat_density()`).
```



```
if (save_figures) {
  ggsave(filename = "density_plot.png", path = "figures/", plot = plot_facet1)}

# Version 2: Adjusted scales
plot_facet2 <- ggplot(eval_all_models %>% 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 = paste("For model based on:", model_name),
       x = "Forecast Error",
       y = "Density") +
  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),
        strip.background = element_rect(fill = "#ecf0f1", color = NA),
        strip.text = element_text(face = "bold"))
print(plot_facet2)
```

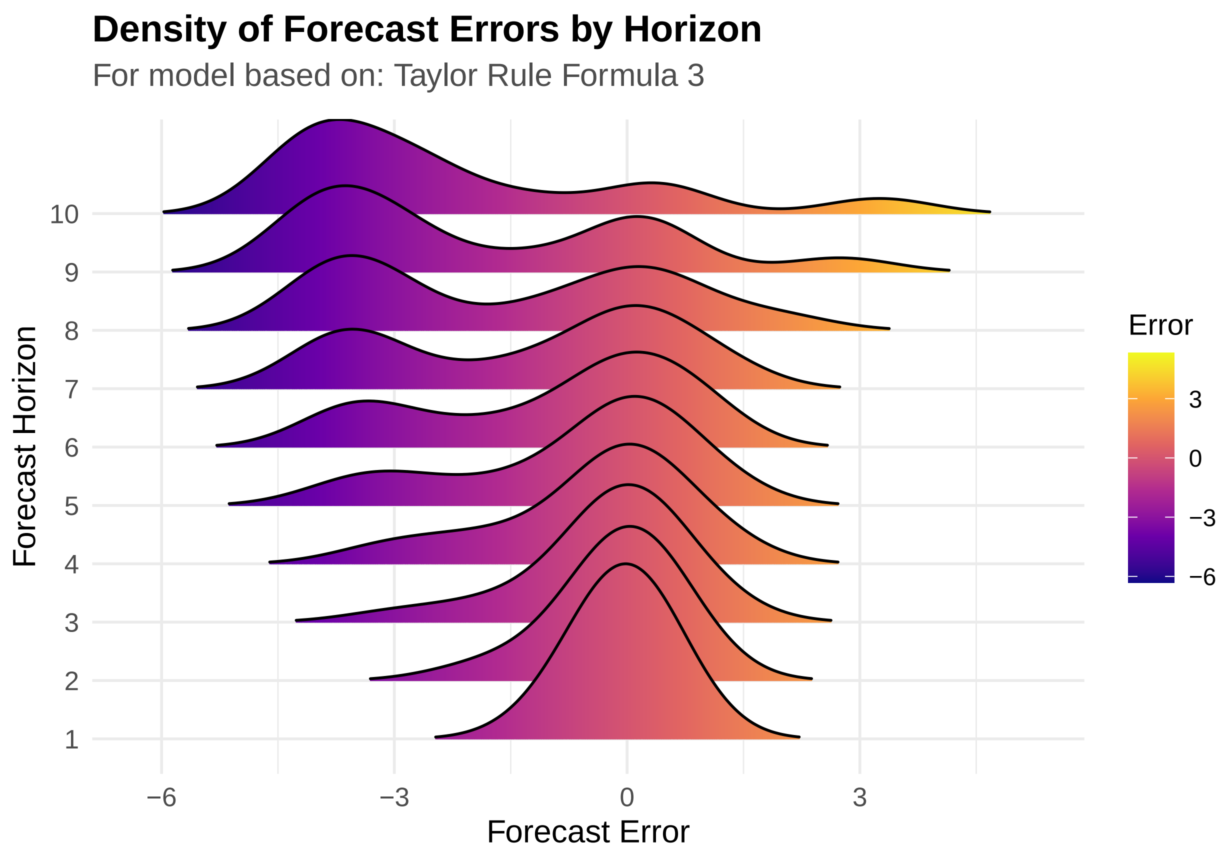
```
## Warning: Removed 45 rows containing non-finite outside the scale range
## (`stat_density()`).
```



```r
if (save_figures) {
  ggsave(filename = "density_scaled_plot.png", path = "figures/", plot = plot_facet2)}

# Version 3: "Ridges"
plot_ridge <- ggplot(eval_all_models %>% 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 = "Density of Forecast Errors by Horizon",
         subtitle = paste("For model based on:", model_name),
         x = "Forecast Error",
         y = "Forecast Horizon") +
    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)),
      axis.title = element_text(size = 12),
      axis.text = element_text(size = 10))
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 0.688
```

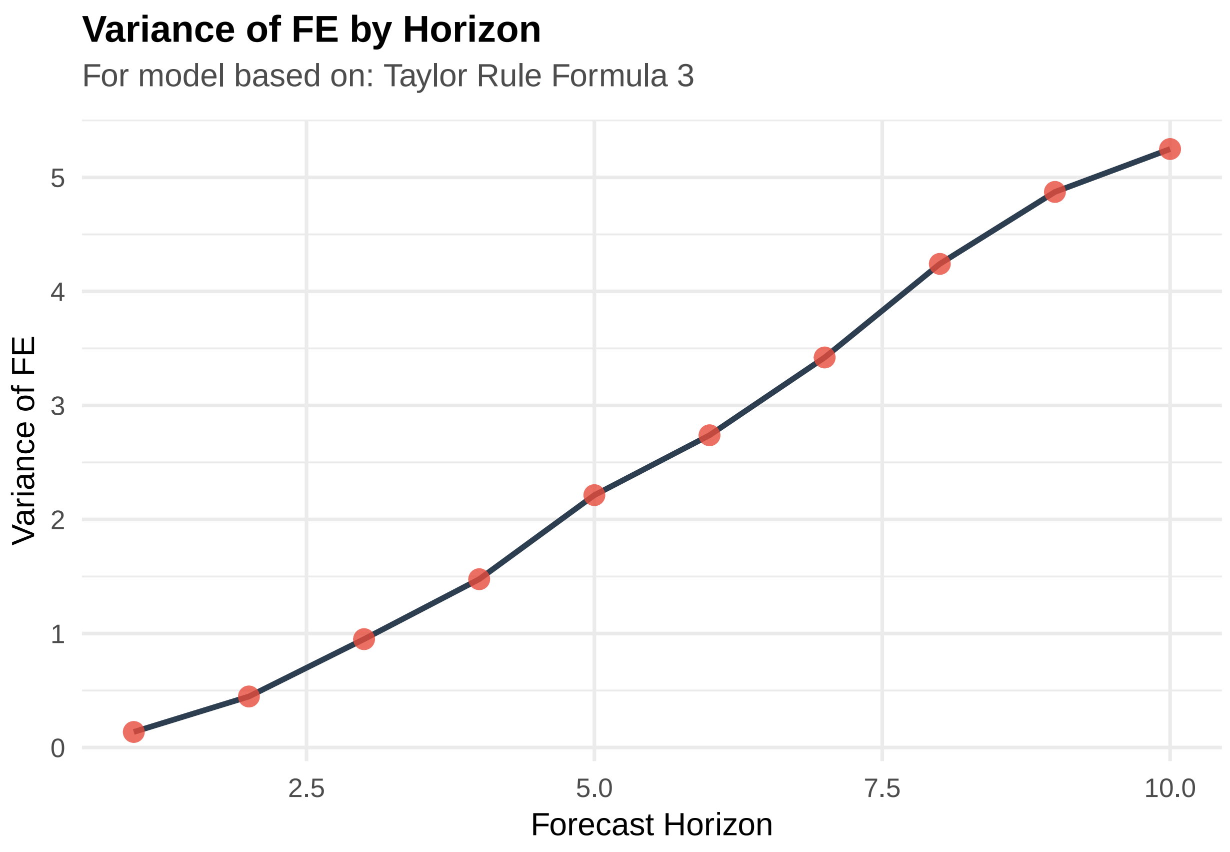


```
if (save_figures) {
  ggsave(filename = "densityridge_plot.png", path = "figures/", plot = plot_ridge)}
```

Variance of FE

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

```r
    var_fe = sd(forecast_error, na.rm=T)^2, n = n() )

fe_var_plot <- 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",
    subtitle = paste("For model based on:", model_name),
    x = "Forecast Horizon",
    y = "Variance of FE") +
  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)),
        axis.title = element_text(size = 12),
        axis.text = element_text(size = 10),
        panel.grid.minor.x = element_blank() )
fe_var_plot
```



```r
if (save_figures) {
  ggsave(filename = "FE_var_plot.png", path = "figures/", plot=fe_var_plot)}
```

Horizon 1 Autocorrelation of FE

Mmodel 1, 2, 3 are autocorrelated at h=1.

```r
#Durbin Watson tests for first autocorrelation at h=1

# Select Forecast Columns
formula_cols <- grep("^F_TR_FORMULA_", names(eval_all_models), value = TRUE)

# Automatically set max horizon for loop
max_h <- max(eval_all_models$horizon)

# Loop for each model
for (e_model_name in formula_cols) {

  # Extract and regress errors
  all_forecast_errors <- eval_all_models[["actuals"]] - eval_all_models[[e_model_name]]
  h1_errors_vector <- all_forecast_errors[eval_all_models$horizon == 1]
  temp_model <- lm(h1_errors_vector ~ 1)

  # Durbin-Watson Test
  dw_test_result <- durbinWatsonTest(temp_model)

  # Print output
  cat(sprintf("\n--- Durbin-Watson Test for Model: %s (h=1) ---\n", e_model_name))
  print(dw_test_result)
  cat(sprintf("DW Statistic: %.4f\n", dw_test_result$statistic))}
```

— Durbin-Watson Test for Model: F_TR_FORMULA_1 (h=1) — lag Autocorrelation D-W Statistic p-value 1 0.9175895 0.1455016 0 Alternative hypothesis: rho != 0

— Durbin-Watson Test for Model: F_TR_FORMULA_2 (h=1) — lag Autocorrelation D-W Statistic p-value 1 0.8986753 0.1798937 0 Alternative hypothesis: rho != 0

— Durbin-Watson Test for Model: F_TR_FORMULA_3 (h=1) — lag Autocorrelation D-W Statistic p-value 1 0.4973872 1.002973 0.022 Alternative hypothesis: rho != 0

— Durbin-Watson Test for Model: F_TR_FORMULA_4 (h=1) — lag Autocorrelation D-W Statistic p-value 1 -0.07327455 2.115385 0.804 Alternative hypothesis: rho != 0

Overall Autocorrelation of FE

We don't reject H0 -> h+1 are uncorrelated 1,2, are autocorrelated, 3 and 4 arent

```r
#Ljung Box Test, with Columns & max lags set in Durbin Watson code chunk

# Loop through each TR
for (e_model_name in formula_cols) {

    cat(sprintf("--- Checking Errors for Model: %s ---\n", e_model_name))

    # Calculate the errors of the model
    all_errors <- eval_all_models[["actuals"]] - eval_all_models[[e_model_name]]

    # Loop over all forecast horizons to test for autocorrelation
    for (h in 1:max_h){

        # select errors for each horizon
        h_errors <- all_errors[eval_all_models$horizon == h]

        # max lag according to slides around sqrt T
        T_errors <- length(h_errors)
        max_lag <- 4 #round(sqrt(T_errors)) # use 4 or 5 chose 4 because 1 year has 4 quarters

        # Ljung-Box Test
        lb_test_result <- Box.test(h_errors,
                                   lag = max_lag,
                                   type = "Ljung-Box")

        #Print Results
        cat(sprintf("Horizon h=%d (N=%d, Lags=%d): Q=%.2f, p-value=%.3f\n",
                    h, T_errors, max_lag, lb_test_result$statistic, lb_test_result$p.value))}
        cat("\n")}
```

— Checking Errors for Model: F_TR_FORMULA_1 — Horizon h=1 (N=22, Lags=4): Q=47.79, p-value=0.000 Horizon h=2 (N=22, Lags=4): Q=45.55, p-value=0.000 Horizon h=3 (N=22, Lags=4): Q=44.69, p-value=0.000 Horizon h=4 (N=22, Lags=4): Q=40.67, p-value=0.000 Horizon h=5 (N=22, Lags=4): Q=27.72, p-value=0.000 Horizon h=6 (N=22, Lags=4): Q=16.42, p-value=0.003 Horizon h=7 (N=22, Lags=4): Q=10.25, p-value=0.036 Horizon h=8 (N=22, Lags=4): Q=9.62, p-value=0.047 Horizon h=9 (N=22, Lags=4): Q=7.04, p-value=0.134 Horizon h=10 (N=22, Lags=4): Q=8.80, p-value=0.066

— Checking Errors for Model: F_TR_FORMULA_2 — Horizon h=1 (N=22, Lags=4): Q=43.55, p-value=0.000 Horizon h=2 (N=22, Lags=4): Q=31.45, p-value=0.000 Horizon h=3 (N=22, Lags=4): Q=18.25, p-value=0.001 Horizon h=4 (N=22, Lags=4): Q=5.79, p-value=0.215 Horizon h=5 (N=22, Lags=4): Q=2.22, p-value=0.695 Horizon h=6 (N=22, Lags=4): Q=1.52, p-value=0.824 Horizon h=7 (N=22, Lags=4): Q=1.55, p-value=0.819 Horizon h=8 (N=22, Lags=4): Q=1.09, p-value=0.896 Horizon h=9 (N=22, Lags=4): Q=0.95, p-value=0.918 Horizon h=10 (N=22, Lags=4): Q=0.74, p-value=0.947

— Checking Errors for Model: F_TR_FORMULA_3 — Horizon h=1 (N=22, Lags=4): Q=8.50, p-value=0.075 Horizon h=2 (N=22, Lags=4): Q=15.23, p-value=0.004 Horizon h=3 (N=22, Lags=4): Q=15.87, p-value=0.003 Horizon h=4 (N=22, Lags=4): Q=18.15, p-value=0.001 Horizon h=5 (N=22, Lags=4): Q=17.36, p-value=0.002 Horizon h=6 (N=22, Lags=4): Q=16.86, p-value=0.002 Horizon h=7 (N=22, Lags=4): Q=16.02, p-value=0.003 Horizon h=8 (N=22, Lags=4): Q=14.72, p-value=0.005 Horizon h=9 (N=22, Lags=4): Q=12.30, p-value=0.015 Horizon h=10 (N=22, Lags=4): Q=8.83, p-value=0.065

— Checking Errors for Model: F_TR_FORMULA_4 — Horizon h=1 (N=22, Lags=4): Q=1.82, p-value=0.770 Horizon h=2 (N=22, Lags=4): Q=9.02, p-value=0.061 Horizon h=3 (N=22, Lags=4): Q=11.89, p-value=0.018 Horizon h=4 (N=22, Lags=4): Q=12.75, p-value=0.013 Horizon h=5 (N=22, Lags=4): Q=11.97, p-value=0.018 Horizon h=6 (N=22, Lags=4): Q=11.28, p-value=0.024 Horizon h=7 (N=22, Lags=4): Q=11.98, p-value=0.017 Horizon h=8 (N=22, Lags=4): Q=12.30, p-value=0.015 Horizon h=9 (N=22, Lags=4): Q=12.41, p-value=0.015 Horizon h=10 (N=22, Lags=4): Q=11.81, p-value=0.019

Errors Normally Distributed

Run the Jarque Bera Test with helpers

```r
# in final markdown, input just the relevant code, i.e. the tests made
# who cares about code to create tables, that stays in helpers
source(here("helpers/pseudo_outofsample_tests.R"))
```

```r
# Run with helper functions
# missing benchmark
jb_results <- generate_all_jb_reports(
  formula_cols =  formula_cols,
  eval_all_models = eval_all_models,
  max_h = max_h
)

jb_results
```

Table 9: Jarque–Bera Test Results

horizon	F_TR_FORMULA_1	F_TR_FORMULA_2	F_TR_FORMULA_3	F_TR_FORMULA_4
1	1.3147 (0.518)	1.4536 (0.483)	3.0503 (0.218)	17.8432 (0.000 ***)
2	1.5416 (0.463)	1.3227 (0.516)	8 (0.018 **)	2.9683 (0.227)
3	1.7299 (0.421)	1.6514 (0.438)	7.5554 (0.023 **)	2.5318 (0.282)
4	1.4723 (0.479)	30.1444 (0.000 ***)	2.75 (0.253)	2.2488 (0.325)
5	1.4886 (0.475)	58.129 (0.000 ***)	2.3399 (0.310)	2.194 (0.334)
6	1.5736 (0.455)	67.3941 (0.000 ***)	1.8789 (0.391)	2.4866 (0.288)
7	0.9188 (0.632)	64.089 (0.000 ***)	1.6653 (0.435)	3.159 (0.206)
8	0.3858 (0.825)	56.1628 (0.000 ***)	1.3626 (0.506)	2.9565 (0.228)
9	0.3242 (0.850)	42.1115 (0.000 ***)	1.3005 (0.522)	3.2125 (0.201)
10	1.5089 (0.470)	30.9638 (0.000 ***)	3.1801 (0.204)	3.0005 (0.223)

Absolute Performance: Efficiency & Bias

```r
# in final markdown, input just the relevant code, i.e. the tests made
# who cares about code to create tables, that stays in helpers
source(here("helpers/pseudo_outofsample_tests.R"))
```

```r
# Call MZ-test helper function 4 times.
#  Note: These reports is wrapped in trycatch as it sometimes fails
#         If it does fail, simply decrease R in order to have more
#          observations, removing potential multicolinearity.

# MZ Report 1: Actual Rate, No Lag
mz_report_1 <- tryCatch({
  generate_absolute_performance_report(
    F_model = F_TR_1,
    Actual_values = Actuals,
    H = H,
    model_caption = "Mincer-Zarnowitz Test: Actual Rate, No Lag",
    format = format)}, error = function(e) {
  message("Error generating MZ Report (Actual Rate, No Lag): ", e$message)
  message("Skipping this report and continuing...")
  return(NULL)})

# MZ Report 2: Shadow Rate, No Lag (
mz_report_2 <- tryCatch({
  generate_absolute_performance_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)
  message("Skipping this report and continuing...")
  return(NULL)})

# MZ Report 3: Actual Rate, with Lag
mz_report_3 <- tryCatch({
  generate_absolute_performance_report(
    F_model = F_TR_3,
    Actual_values = Actuals,
    H = H,
    model_caption = "Mincer-Zarnowitz Test: Actual Rate, with Lag",
    format = format)}, error = function(e) {
  message("Error generating MZ Report (Actual Rate, with Lag): ", e$message)
  message("Skipping this report and continuing...")
  return(NULL)})

# MZ Report 4: Shadow Rate, with Lag
mz_report_4 <- tryCatch({
  generate_absolute_performance_report(
    F_model = F_TR_4,
    Actual_values = Actuals,
    H = H,
    model_caption = "Mincer-Zarnowitz Test: Shadow Rate, with Lag",
    format = format)}, error = function(e) {
  message("Error generating MZ Report (Shadow Rate, with Lag): ", e$message)
  message("Skipping this report and continuing...")
  return(NULL)})

# MZ Report 5: Benchmark
mz_report_BM <- tryCatch({
  generate_absolute_performance_report(
    F_model = F_BM,
    Actual_values = Actuals,
    H = H,
    model_caption = "Mincer-Zarnowitz Test: Benchmark ARIMA",
    format = format)}, error = function(e) {
  message("Error generating MZ Report (Benchmark ARIMA): ", e$message)
  message("Skipping this report and continuing...")
  return(NULL)})

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

[[1]]

[[2]]

[[3]]

[[4]]

Table 10: Mincer-Zarnowitz Test: Actual Rate, No Lag

h	Alpha	Beta	pv(Joint)
1	1.2076	0.3116	0.184
2	1.2531	0.3795	0.756
3	0.9920	0.8033	0.857
4	0.8109	1.2440	0.218
5	0.8868	1.4210	0.007 ***
6	1.0360	1.4710	0.000 ***
7	1.2366	1.3372	0.000 ***
8	1.5904	1.0953	0.000 ***
9	2.1531	0.6397	0.000 ***
10	2.8582	0.0750	0.004 ***

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: Shadow Rate, No Lag

h	Alpha	Beta	pv(Joint)
1	1.8200	-0.3633	0.000 ***
2	1.8598	-0.2299	0.000 ***
3	1.7793	-0.0457	0.000 ***
4	1.8299	0.0095	0.000 ***
5	1.9916	-0.0130	0.000 ***
6	2.1898	-0.0409	0.000 ***
7	2.4185	-0.0665	0.000 ***
8	2.6798	-0.0894	0.000 ***
9	2.8483	-0.0617	0.000 ***
10	3.0489	-0.0389	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 12: Mincer-Zarnowitz Test: Actual Rate, with Lag

h	Alpha	Beta	pv(Joint)
1	0.0547	1.0015	0.793
2	0.2077	0.9484	0.793
3	0.4454	0.8750	0.731
4	0.7365	0.7927	0.655
5	1.1190	0.6535	0.492
6	1.4765	0.5449	0.372
7	1.8810	0.3823	0.108
8	2.2834	0.1895	0.023 **
9	2.6784	0.0002	0.001 ***
10	3.0516	-0.1807	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 13: Mincer-Zarnowitz Test: Shadow Rate, with Lag

h	Alpha	Beta	pv(Joint)
1	0.0681	0.9143	0.087 *
2	0.1930	0.7986	0.186
3	0.4392	0.6389	0.007 ***
4	0.7626	0.4854	0.000 ***
5	1.1300	0.3526	0.000 ***
6	1.4534	0.2523	0.000 ***
7	1.8018	0.1659	0.000 ***
8	2.1810	0.0875	0.000 ***
9	2.6095	0.0196	0.000 ***
10	3.0870	-0.0420	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 14: Mincer-Zarnowitz Test: Benchmark ARIMA

h	Alpha	Beta	pv(Joint)
1	0.1487	0.9447	0.382
2	0.3943	0.8503	0.193
3	0.7096	0.7125	0.197
4	1.0656	0.5675	0.290
5	1.4463	0.4072	0.146
6	1.8131	0.2622	0.144
7	2.1639	0.1192	0.006 ***
8	2.4781	-0.0143	0.001 ***
9	2.7540	-0.1508	0.000 ***
10	2.9915	-0.2735	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

[[5]]

Relative Performance (against benchmark)

```r
# in final markdown, input just the relevant code, i.e. the tests made
# who cares about code to create tables, that stays in helpers
source(here("helpers/pseudo_outofsample_tests.R"))
```

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

# Report 1: Actual Rate, No Lag
report_1 <- generate_relative_performance_report(
  FE_TR_model = FE_TR_1,
  FE_BM_model = FE_BM,
  H = H,
  model_caption = "MSFE Comparison, Trained on Actual Rate, No Lag",
  format = format)

# Report 2: Shadow Rate, No Lag
report_2 <- generate_relative_performance_report(
  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_relative_performance_report(
```

Table 15: MSFE Comparison, Trained on Actual Rate, No Lag

h	MSFE TR	MSFE BM	Ratio	DM Two-Sided	DM Greater	DM Lesser
1	3.9517	0.1591	24.8393	0.000 ***	1.000	0.000 ***
2	3.9732	0.6458	6.1521	0.032 **	0.984	0.016 **
3	3.4469	1.6062	2.1459	0.386	0.807	0.193
4	3.1086	2.8224	1.1014	0.915	0.542	0.458
5	3.0938	4.3368	0.7134	0.603	0.302	0.698
6	3.2794	5.8787	0.5578	0.124	0.062 *	0.938
7	3.5234	7.4570	0.4725	0.000 ***	0.000 ***	1.000
8	4.0667	9.0333	0.4502	0.001 ***	0.000 ***	1.000
9	4.8839	10.1786	0.4798	0.000 ***	0.000 ***	1.000
10	5.8798	11.5721	0.5081	0.000 ***	0.000 ***	1.000

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.

```r
  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_relative_performance_report(
  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]]

[[2]]

[[3]]

[[4]]
```

Table 16: MSFE Comparison, Trained on Shadow Rate, No Lag

h	MSFE TR	MSFE BM	Ratio	DM Two-Sided	DM Greater	DM Lesser
1	8.7557	0.1591	55.0357	0.000 ***	1.000	0.000 ***
2	9.4762	0.6458	14.6728	0.002 ***	0.999	0.001 ***
3	9.4969	1.6062	5.9125	0.030 **	0.985	0.015 **
4	12.0493	2.8224	4.2692	0.209	0.895	0.105
5	16.4410	4.3368	3.7910	0.313	0.844	0.156
6	22.9412	5.8787	3.9024	0.363	0.819	0.182
7	30.3125	7.4570	4.0650	0.384	0.808	0.192
8	40.6792	9.0333	4.5032	0.336	0.832	0.168
9	48.6339	10.1786	4.7781	0.256	0.872	0.128
10	59.1154	11.5721	5.1084	0.144	0.928	0.072 *

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 17: MSFE Comparison, Trained on Actual Rate, with Lag

h	MSFE TR	MSFE BM	Ratio	DM Two-Sided	DM Greater	DM Lesser
1	0.1335	0.1591	0.8393	0.491	0.246	0.754
2	0.4435	0.6458	0.6866	0.196	0.098 *	0.902
3	0.9719	1.6062	0.6051	0.136	0.068 *	0.932
4	1.5987	2.8224	0.5664	0.104	0.052 *	0.948
5	2.5347	4.3368	0.5845	0.037 **	0.018 **	0.982
6	3.4632	5.8787	0.5891	0.008 ***	0.004 ***	0.996
7	4.7305	7.4570	0.6344	0.009 ***	0.004 ***	0.996
8	6.2083	9.0333	0.6873	0.002 ***	0.001 ***	0.999
9	7.9732	10.1786	0.7833	0.013 **	0.006 ***	0.994
10	9.7356	11.5721	0.8413	0.080 *	0.040 **	0.960

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 18: MSFE Comparison, Trained on Shadow Rate, with Lag

h	MSFE TR	MSFE BM	Ratio	DM Two-Sided	DM Greater	DM Lesser
1	0.1591	0.1591	1.0000	1.000	0.500	0.500
2	0.5536	0.6458	0.8571	0.715	0.357	0.643
3	1.5969	1.6062	0.9942	0.946	0.473	0.527
4	3.5822	2.8224	1.2692	0.371	0.815	0.185
5	6.7917	4.3368	1.5661	0.230	0.885	0.115
6	11.0846	5.8787	1.8856	0.076 *	0.962	0.038 **
7	17.4023	7.4570	2.3337	0.079 *	0.960	0.040 **
8	26.3667	9.0333	2.9188	0.100	0.950	0.050 *
9	37.3973	10.1786	3.6741	0.096 *	0.952	0.048 **
10	51.5048	11.5721	4.4508	0.056 *	0.972	0.028 **

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

Forecasting

```r
# Function to make our actual forecast, including forecasts of our inputs,
#  and also outputting the ARIMA coefficients and models used
our_predict <- function(data,formula,H){

  # --- 1. Fit inputs and benchmark models  ---
   inflation_arma <- my.auto.arima(data$inflation_gap, max.p=4, max.q=4, d=1, var_name="Inflation Gap")
   outputgap_arma <- my.auto.arima(data$output_gap, max.p=4, max.q=4, d=0, var_name="Output Gap")
   interest_arma <- my.auto.arima(data$rate, max.p=4, max.q=4, d=1, var_name="Interest Rate") # Benchma

  # --- 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],
        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,
              inflation_gap_arma_coef = inflation_arma$coef,
              output_gap_arma_coef = outputgap_arma$coef,
```

Table 19: For model based on: Taylor Rule Formula 3

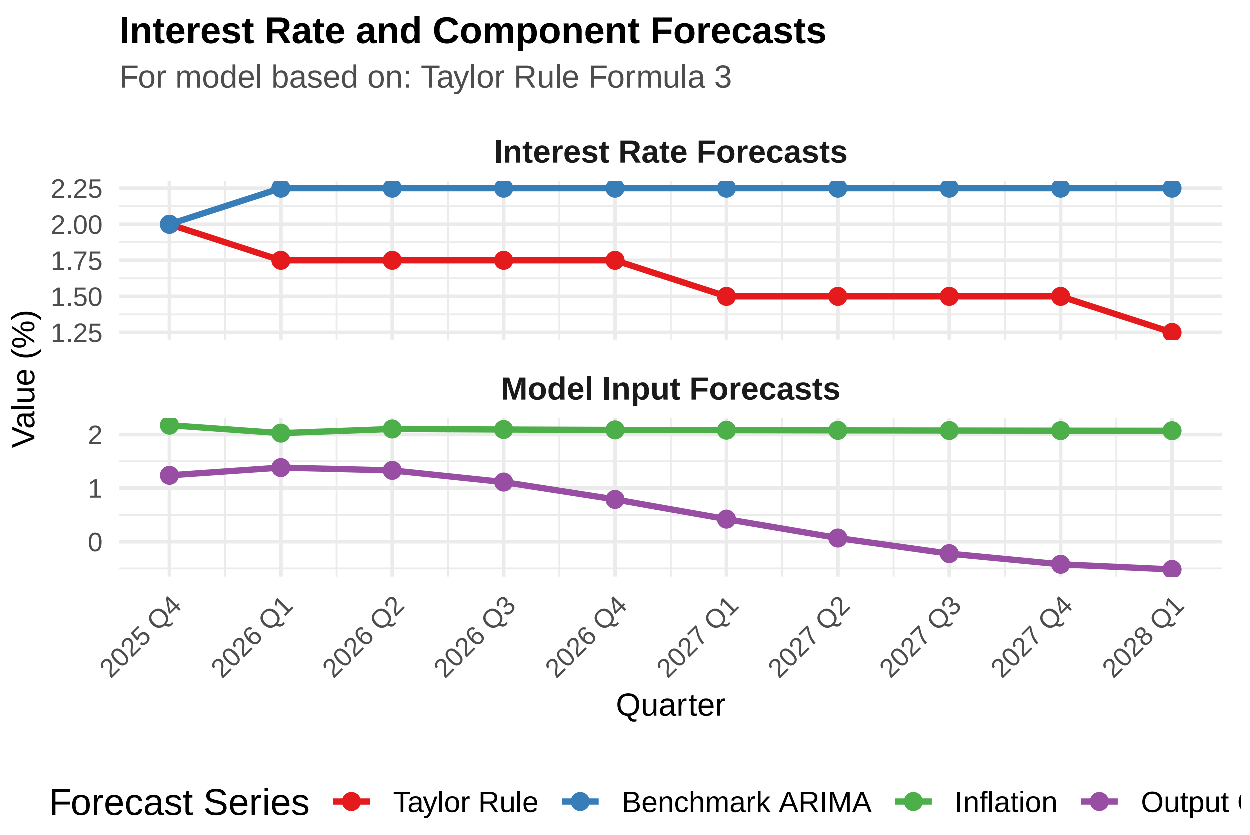
Horizon: Quarter	Taylor Rule Forecast	Benchmark Forecast	Inflation Forecast	Output Gap Forecast
1: 2025 Q4	2.00	2.00	2.17	1.24
2: 2026 Q1	1.75	2.25	2.03	1.38
3: 2026 Q2	1.75	2.25	2.10	1.33
4: 2026 Q3	1.75	2.25	2.09	1.11
5: 2026 Q4	1.75	2.25	2.09	0.79
6: 2027 Q1	1.50	2.25	2.08	0.42
7: 2027 Q2	1.50	2.25	2.08	0.07
8: 2027 Q3	1.50	2.25	2.07	-0.22
9: 2027 Q4	1.50	2.25	2.07	-0.42
10: 2028 Q1	1.25	2.25	2.07	-0.52

```
                benchmark_arma_coef = interest_arma$coef,
                inflation_gap_arma_var = inflation_arma$sigma2,
                output_gap_arma_var = outputgap_arma$sigma2,
                benchmark_arma_var = interest_arma$sigma2))}


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

[1] "This model is fitted to the variable 'Inflation Gap' specified as ARIMA(1, 1, 4)." [1] "This model is fitted to the variable 'Output Gap' specified as ARIMA(2, 0, 3)." [1] "This model is fitted to the variable 'Interest Rate' specified as ARIMA(2, 1, 0)."

```
display_forecasts(final_forecasts,
                  caption = paste("For model based on:", model_name),
                  format = format)
```

```
final_fc_plot1 <- plot_forecasts(final_forecasts)
final_fc_plot1
```



```r
if (save_figures) {
  ggsave(filename = "actual_forecasts_plot.png", path = "figures/", plot=final_fc_plot1)
}
```

Prediction Intervals

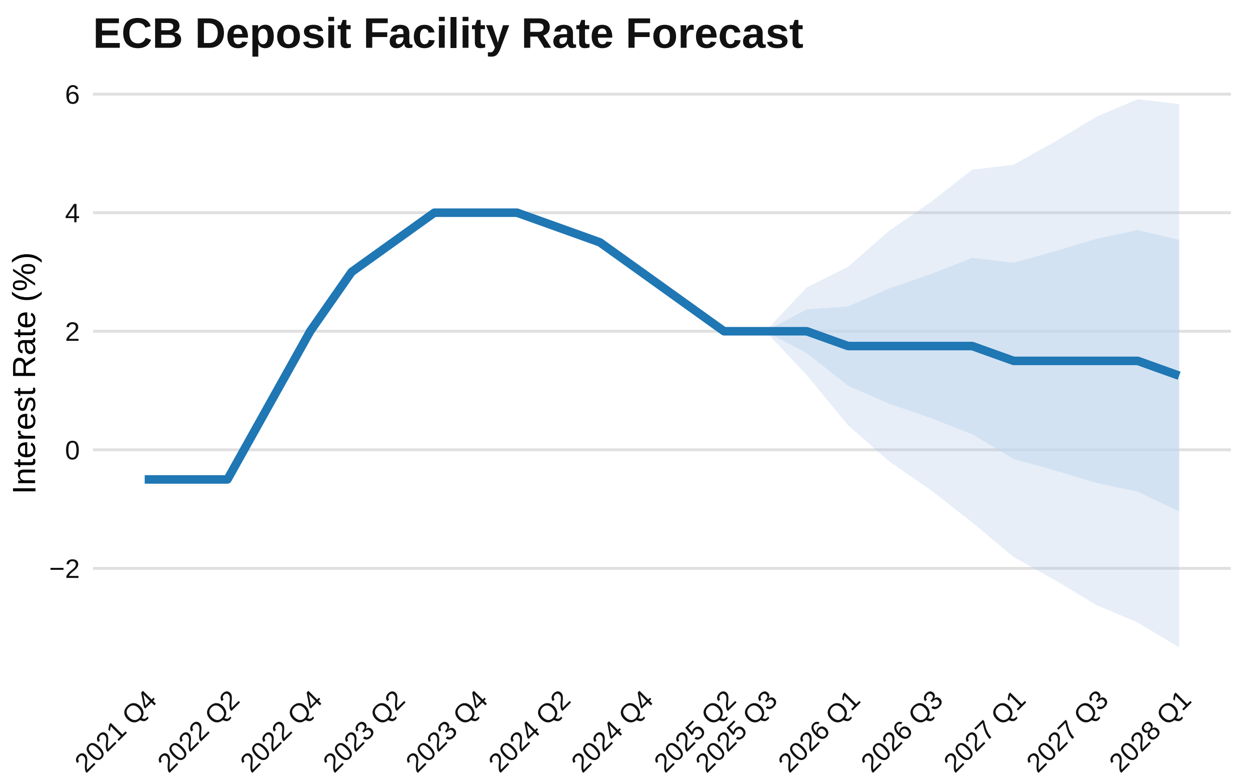
Prepare Data

```r
# Preparing data with point forecasts and variance
prediction <- var_by_horizon
final_interval <- final_forecasts[c(1,2,3,4)]
prediction$sd_fe <- sqrt(prediction$var_fe)

# Computing confidence intervals
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)
```

Prediction interval Plot

```r
# Get last 4 years of actual data -> show trend over past
last_obs <- data %>%
  filter(quarter > max(quarter) - 4) %>%  # last 4 years
  select(quarter, rate) %>%
  rename(Value = rate)

# Prepare forecast data
H <- nrow(final_interval)
forecast_quarters_yearqtr <- seq(from = last(data$quarter) + 0.25,
                                 by = 0.25,
                                 length.out = H)

# This df is for the forecast part to make the band
forecast_df <- final_interval %>%
  mutate(
    quarter = forecast_quarters_yearqtr,
    quarter_numeric = as.numeric(quarter),
    Value = TR_Forecast
  ) %>%
  select(quarter, quarter_numeric, Value, lower_1_sd, upper_1_sd, lower_2_sd, upper_2_sd)

# Add a row where lowest quarter -0.25 and add for all lower and upper sd = 0 and add last(data) as the

# Add a row at the start
forecast_df <- forecast_df %>%
  bind_rows(
    tibble(
      quarter = forecast_df$quarter[1] - 0.25,
      quarter_numeric = as.numeric(forecast_df$quarter[1] - 0.25),
      Value = data %>% filter(quarter == forecast_df$quarter[1] - 0.25) %>% pull(rate),
      lower_1_sd = Value,
      upper_1_sd = Value,
      lower_2_sd = Value,
      upper_2_sd = Value
    )
  ) %>%
  arrange(quarter_numeric)


# Combine last actuals and forecast for plotting
plot_df <- bind_rows(
  last_obs %>% mutate(lower_1_sd = NA, upper_1_sd = NA, lower_2_sd = NA, upper_2_sd = NA),
  forecast_df
)

# Numeric and label for x-axis
plot_df <- plot_df %>%
  mutate(
    quarter_numeric = as.numeric(quarter),
    quarter_label = as.character(quarter)
  )
```

```r
# Plot
# Color palette
line_color <- "#1f77b4"       # Strong blue for forecast line
ribbon_outer <- "#aec7e8"     # Light transparent blue for 2 SD
ribbon_inner <- "#c6dbef"     # Lighter for 1 SD

# Plot
final_fc_plot2 <- ggplot() +

  # Prediction intervals (forecast only)
  geom_ribbon(data = forecast_df,
              aes(x = quarter_numeric, ymin = lower_2_sd, ymax = upper_2_sd),
              fill = ribbon_outer, alpha = 0.3) +
  geom_ribbon(data = forecast_df,
              aes(x = quarter_numeric, ymin = lower_1_sd, ymax = upper_1_sd),
              fill = ribbon_inner, alpha = 0.6) +

  # Line for actual + forecast
  geom_line(data = plot_df,
            aes(x = quarter_numeric, y = Value),
            color = line_color, size = 1.5) +

  # X-axis formatting: only show every 2 quarters to avoid clutter
  scale_x_continuous(breaks = plot_df$quarter_numeric[seq(1, nrow(plot_df), by = 2)],
                     labels = plot_df$quarter_label[seq(1, nrow(plot_df), by = 2)]) +

  # Labels
  labs(title = "ECB Deposit Facility Rate Forecast",
       #subtitle = paste("Model:", model_name),
       y = "Interest Rate (%)",
       x = "",
       caption = "Shaded areas: ±1 S.D. (dark) / ±2 S.D. (light)") +

  # Theme settings
  theme_minimal() +
  theme(
    plot.title = element_text(face = "bold", size = 16, color = "#111111"),
    plot.subtitle = element_text(size = 12, color = "#333333"),
    axis.text = element_text(size = 10, color = "#111111"),
    axis.title = element_text(size = 12),
    panel.grid.major.y = element_line(color = "#e0e0e0"),
    panel.grid.major.x = element_blank(),
    panel.grid.minor = element_blank(),
    plot.caption = element_text(size = 9, color = "#555555", hjust = 0),
    axis.text.x = element_text(angle = 45, hjust = 1)
  )

final_fc_plot2
```



Shaded areas: ±1 S.D. (dark) / ±2 S.D. (light)

```r
if (save_figures) {
ggsave(
  filename = "forecast_plot.png",
  path = "figures/",
  plot = final_fc_plot2,
  width = 10,          # in inches
  height = 6,          # in inches
  dpi = 300            # high-quality for print
)
}
```