ECB Tests 6

Contents

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

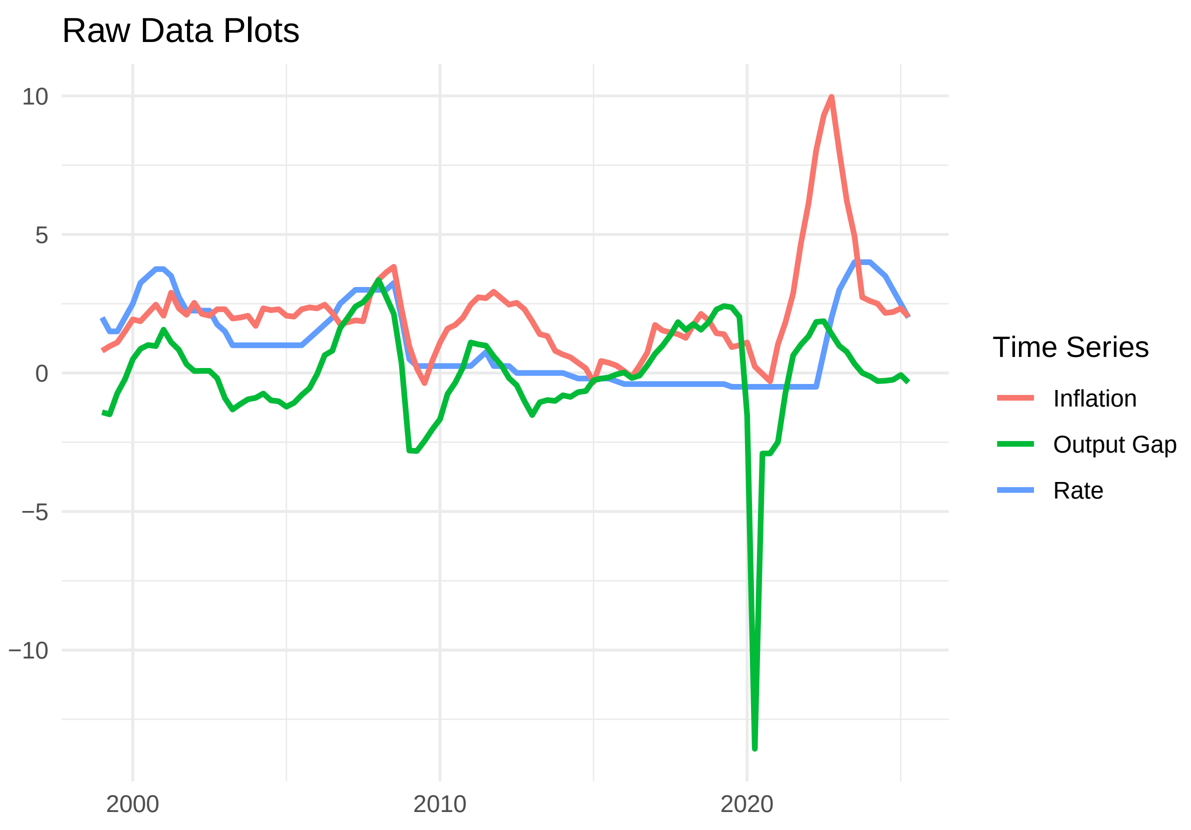
Data

Main

```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) %>%
  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
inflation_q$exp_inflation = c(rep(NA,3),as.numeric(inflation_exp_q$exp_inflation),NA)
#P24M
#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)) %>%
  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)


# Combine all data into a single data frame
data <- ecb_rate_q %>%
  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(
    inflation_gap = inflation - 2.0,
    exp_inflation_gap = exp_inflation -2.0,
    output_gap = 100 * (log_real_gdp - potential_gdp_log),
    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 = 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)
```

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 = "Inflation"), linewidth = 1) +
  geom_line(aes(y = output_gap, color = "Output Gap"), 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 -5.116880    0.01
## [2,]   1 -4.442376    0.01
## [3,]   2 -4.181371    0.01
## [4,]   3 -4.526270    0.01
## [5,]   4 -4.714331    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

$$\begin{aligned} i_t &= \pi^* + \gamma(y_t - \bar{y_t}) + \beta(\pi_t - \pi^*) \\ &= (1-\beta)\pi^* + \gamma(y_t - \bar{y_t}) + \beta\pi_t \end{aligned}$$

```r
TR <- lm(rate ~ inflation_gap + output_gap, data = data)
TRsr <- lm(shadowrate ~ 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)	1.0305	-0.4628
	(0.9157)	(2.5157)
inflation_gap	0.2278	0.5690
	(0.3621)	(0.5988)
output_gap	0.1128	0.0784
	(0.2528)	(0.4505)
N	106	106
R2	0.1318	0.0965

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

```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 ~ inflation_gap + exp_inflation_gap + output_gap, data = data)
TRsr_ie <- lm(shadowrate ~ 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

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

Table 2: No Lag, with Inflation Expectations

	TR	TR w/ SR	TR	TR w/ SR
(Intercept)	1.5234 ***	0.5540	1.5077 **	0.4698
	(0.4184)	(1.3237)	(0.5141)	(1.3453)
exp_inflation_gap	1.6903 ***	3.5180 **	1.6518 ***	3.3113 **
	(0.3236)	(1.3225)	(0.3848)	(1.0299)
output_gap	0.1549	0.2052	0.1440	0.1463
	(0.1396)	(0.4275)	(0.1667)	(0.4195)
inflation_gap			0.0352	0.1892
			(0.1418)	(0.3221)
N	103	103	103	103
R2	0.4832	0.3407	0.4845	0.3478

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

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

```r
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 + inflation_gap + exp_inflation_gap + output_gap, data = data)
lTRsr_ie <- lm(shadowrate ~ shadowrate_lag + inflation_gap + exp_inflation_gap + output_gap, data = data

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

```r
# Start of ZLB in 2012 Q3
breakpoint1 <- 55

# xxx
breakpoint2 <- 85
```

Table 3: Interest Rate Lag, No Expectations

	TR	TR w/ SR
(Intercept)	0.0540	-0.0444
	(0.0420)	(0.0637)
rate_lag	0.9371 ***	
	(0.0411)	
inflation_gap	0.0981 *	0.2367 ***
	(0.0388)	(0.0312)
output_gap	0.0166	-0.0137
	(0.0206)	(0.0206)
shadowrate_lag		0.9604 ***
		(0.0192)
N	105	105
R2	0.9551	0.9822

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

```r
# Chow test (rejecting the null means there are structural breaks)
chow_test1 <- sctest(rate ~ rate_lag + inflation_gap + output_gap, type = "Chow", point = breakpoint1, 

chow_test2 <- sctest(rate ~ rate_lag + inflation_gap + output_gap, type = "Chow", point = breakpoint2, 

chow_df <- data.frame(
  "2012 Q3" = round(chow_test1$p.value, 4),
  "2020 Q1" = round(chow_test2$p.value, 4),
  check.names = FALSE)

kable(chow_df, digits = 4, format = format)%>%
  kable_styling(bootstrap_options = "striped", full_width = FALSE)
```

Rolling Estimation (for structural breaks)

```r
# 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")
```

Table 4: Interest Rate Lag, with Inflation Expectations

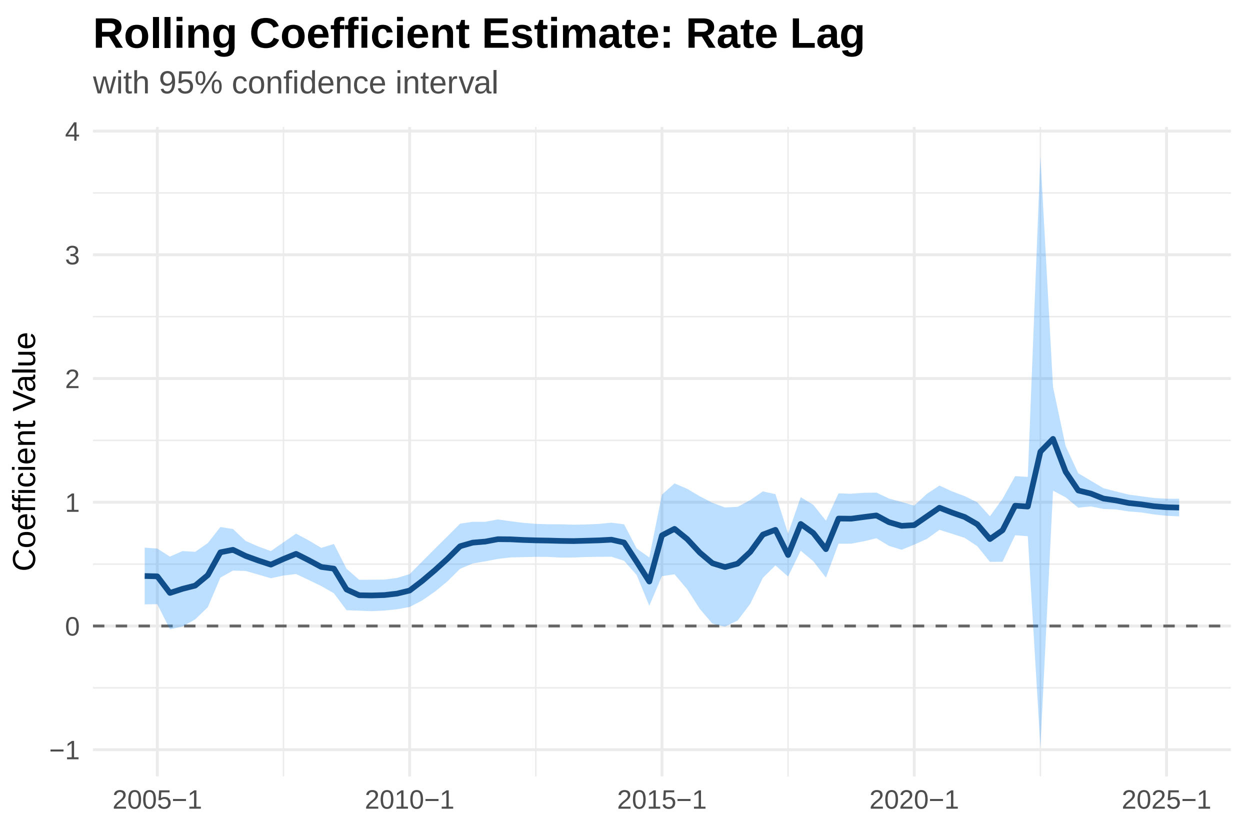
	TR	TR w/ SR	TR	TR w/ SR
(Intercept)	0.1321	0.0311	0.0708	-0.0827
	(0.1227)	(0.1166)	(0.0576)	(0.0494)
rate_lag	0.9189 ***		0.9314 ***	
	(0.0598)		(0.0522)	
exp_inflation_gap	0.1553	0.1507	0.0306	-0.1384 *
	(0.1435)	(0.1259)	(0.0928)	(0.0685)
output_gap	0.0479	0.0613	0.0168	-0.0172
	(0.0321)	(0.0488)	(0.0216)	(0.0192)
shadowrate_lag		0.9669 ***		0.9715 ***
		(0.0366)		(0.0182)
inflation_gap			0.0950 *	0.2500 ***
			(0.0401)	(0.0269)
N	103	103	103	103
R2	0.9455	0.9702	0.9556	0.9825

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

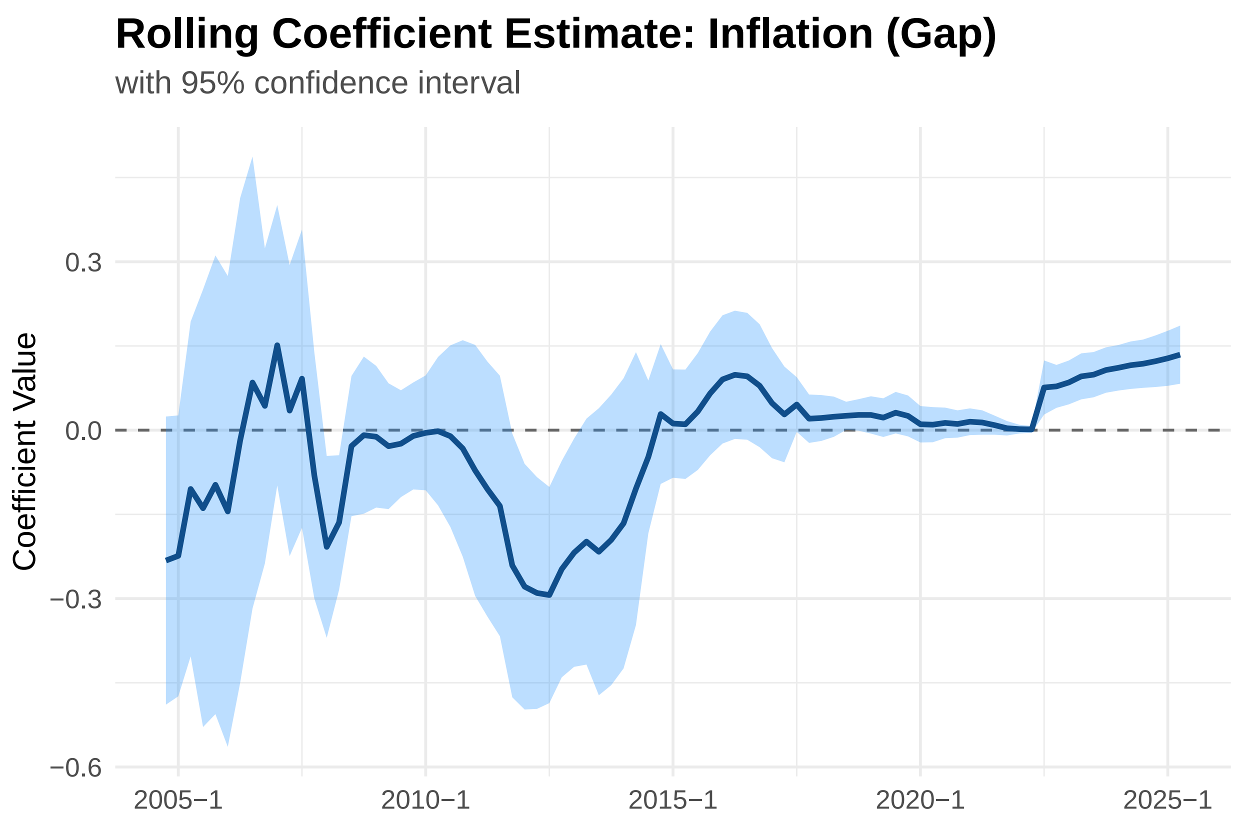
	2012 Q3	2020 Q1
F	1e-04	0.0053

```r
# 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 = 16, 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))
    return(plot) }
```

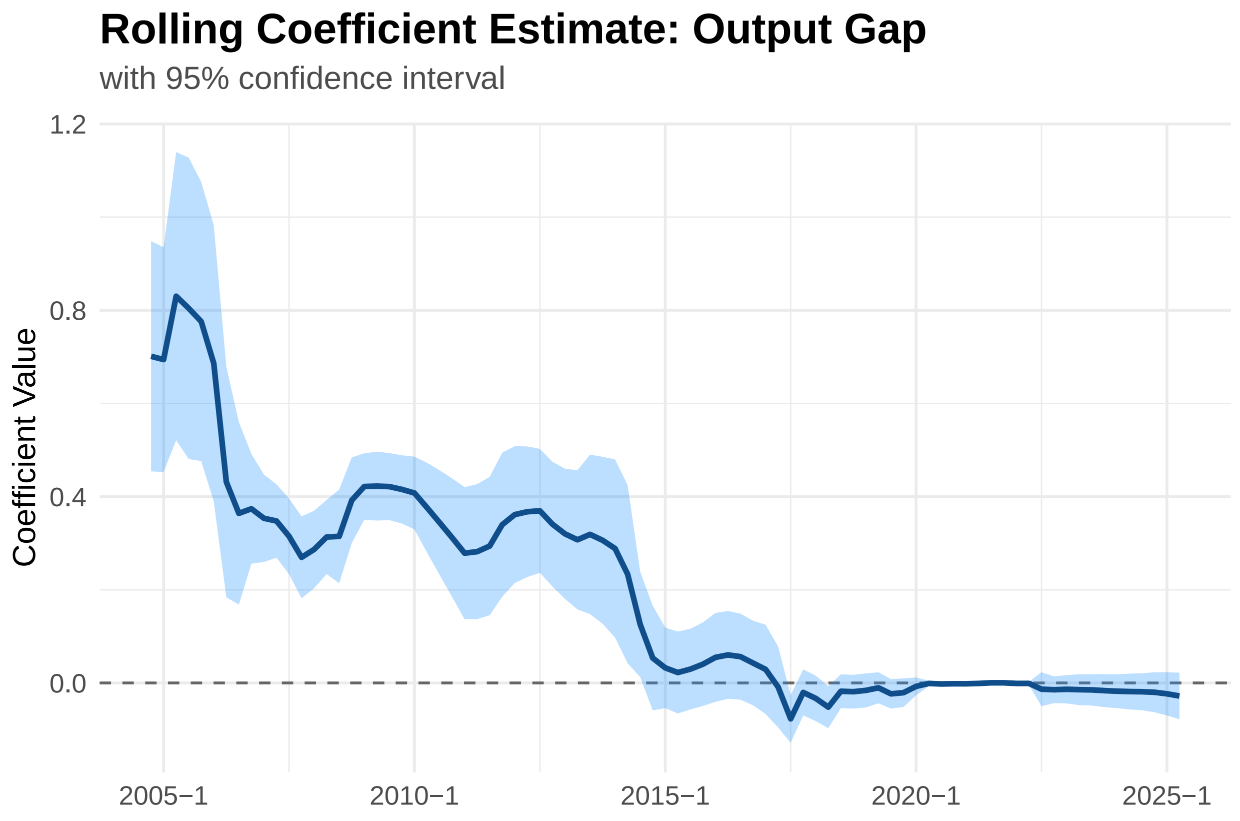
```
# Set rolling window (in quarters) & Looping Parameter
W = 24
L = nrow(data) - W + 1
formula = rate ~ rate_lag + inflation_gap + output_gap
#formula = shadowrate ~ shadowrate_lag + inflation_gap + exp_inflation_gap + output_gap

# 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 ICss
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)")
```



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


Rolling Coefficient Estimate: Output Gap
with 95% confidence interval
Coefficient Value
1.2
0.8
0.4
0.0
2005−1
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) }



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

```
# 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") %>%
    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)}
```

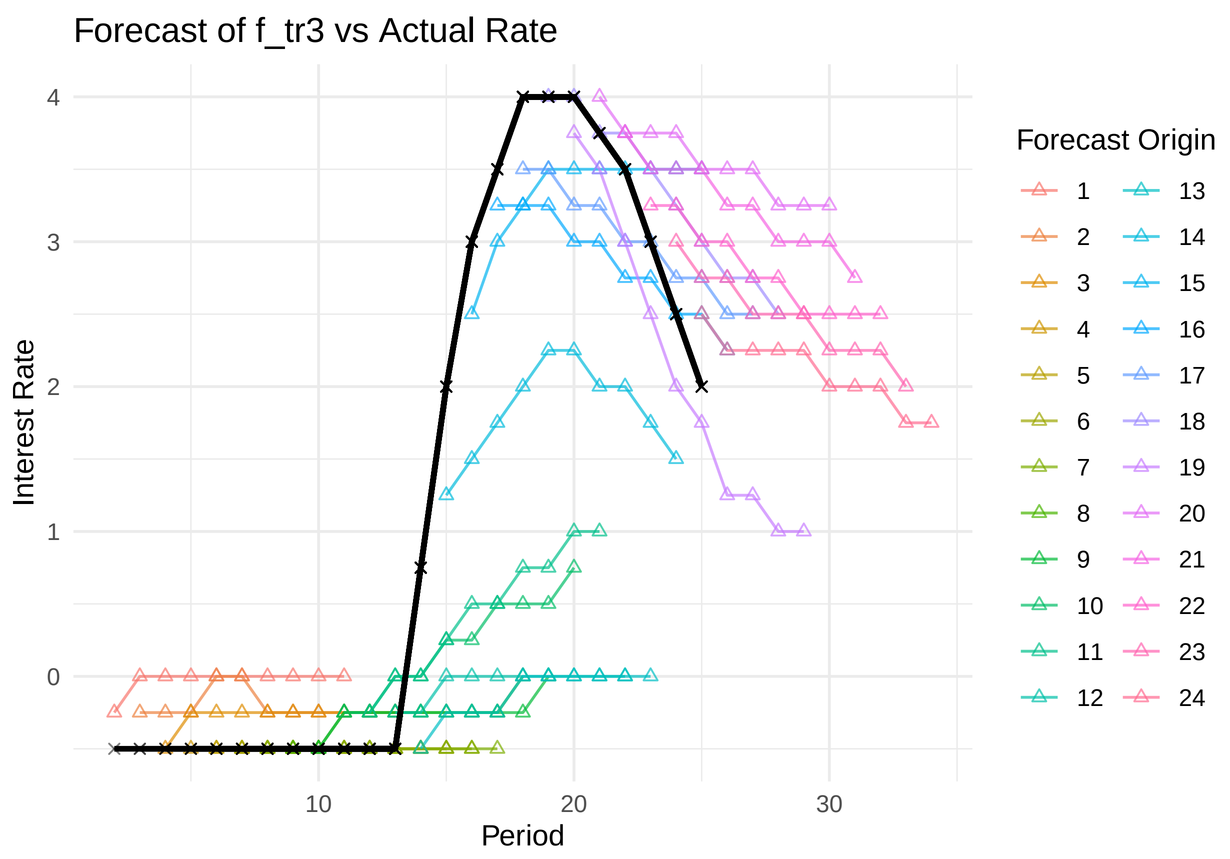
Estimation

```
#parameters
R = 54-28-8 #note: start of ZLB at R=54
R = 54+28
P = nrow(data) - R #but will effectively be: P = T-h-R
H = 10 #number of different horizons

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

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

# TR based on current inflation
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

# TR based on current inflation expectations of inflation in 12 months
#formula_1 <- rate ~ exp_inflation_gap + output_gap
#formula_2 <- shadowrate ~ exp_inflation_gap + output_gap
#formula_3 <- rate ~ rate_lag + exp_inflation_gap + output_gap
#formula_4 <- shadowrate ~ shadowrate_lag + exp_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)}
```

```r
# 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
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 ---
  inflation_arma <- auto.arima(training$inflation_gap, max.p=4, max.q=4, max.d=1)
  #exp_inflation_arma <- auto.arima(training$exp_inflation_gap, max.p=4, max.q=4, max.d=1)
  outputgap_arma <- auto.arima(training$output_gap, max.p=4, max.q=4, max.d=1)
  interest_arma <- auto.arima(training$rate, max.p=4, max.q=4, max.d=1) # Benchmark

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

  # --- 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,
    #exp_inflation_gap = exp_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],
      #exp_inflation_gap = exp_inflation_forecasts[h],
      output_gap = outputgap_forecasts[h],
      rate_lag = current_rate_lag)
    new_data_4_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 )

    # 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
  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))
```

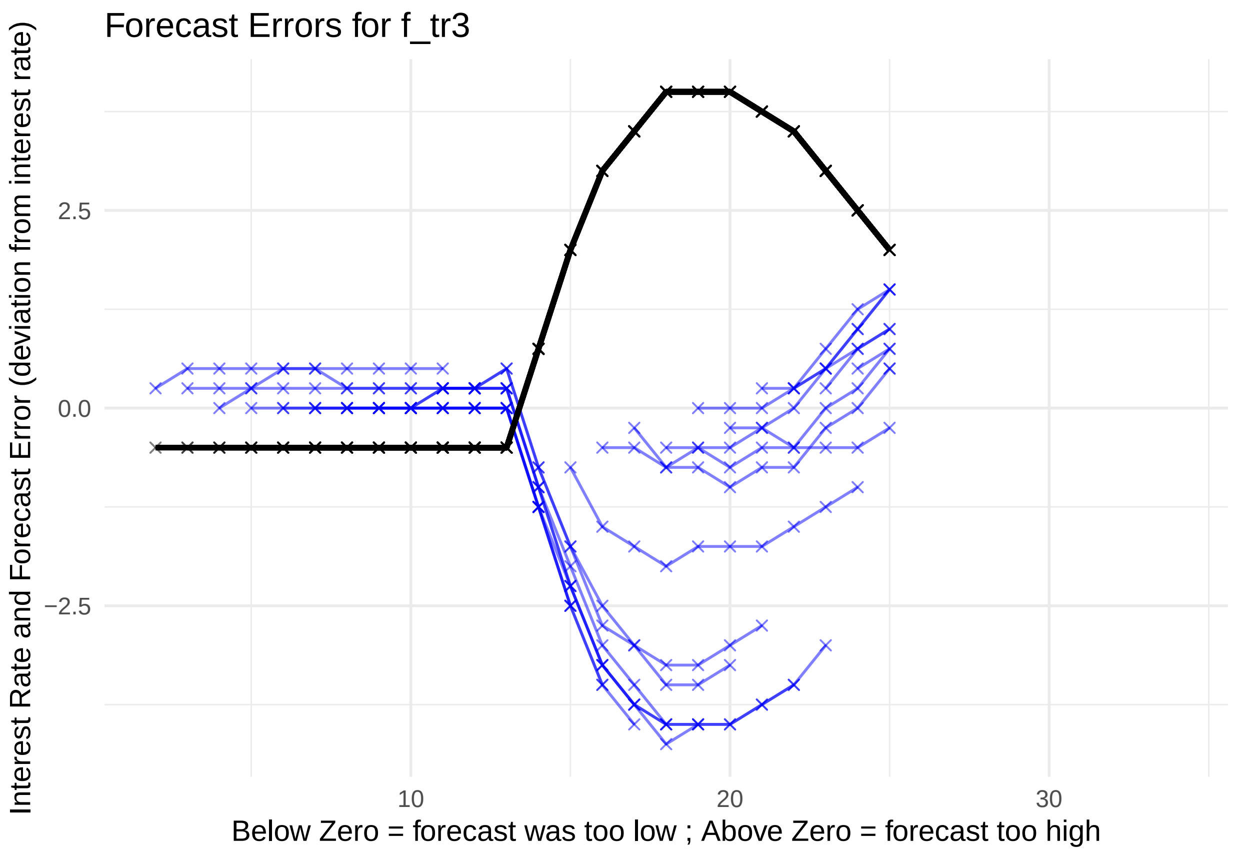
Spaghetti Plots

```
# 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 # is this correct choice?

# 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

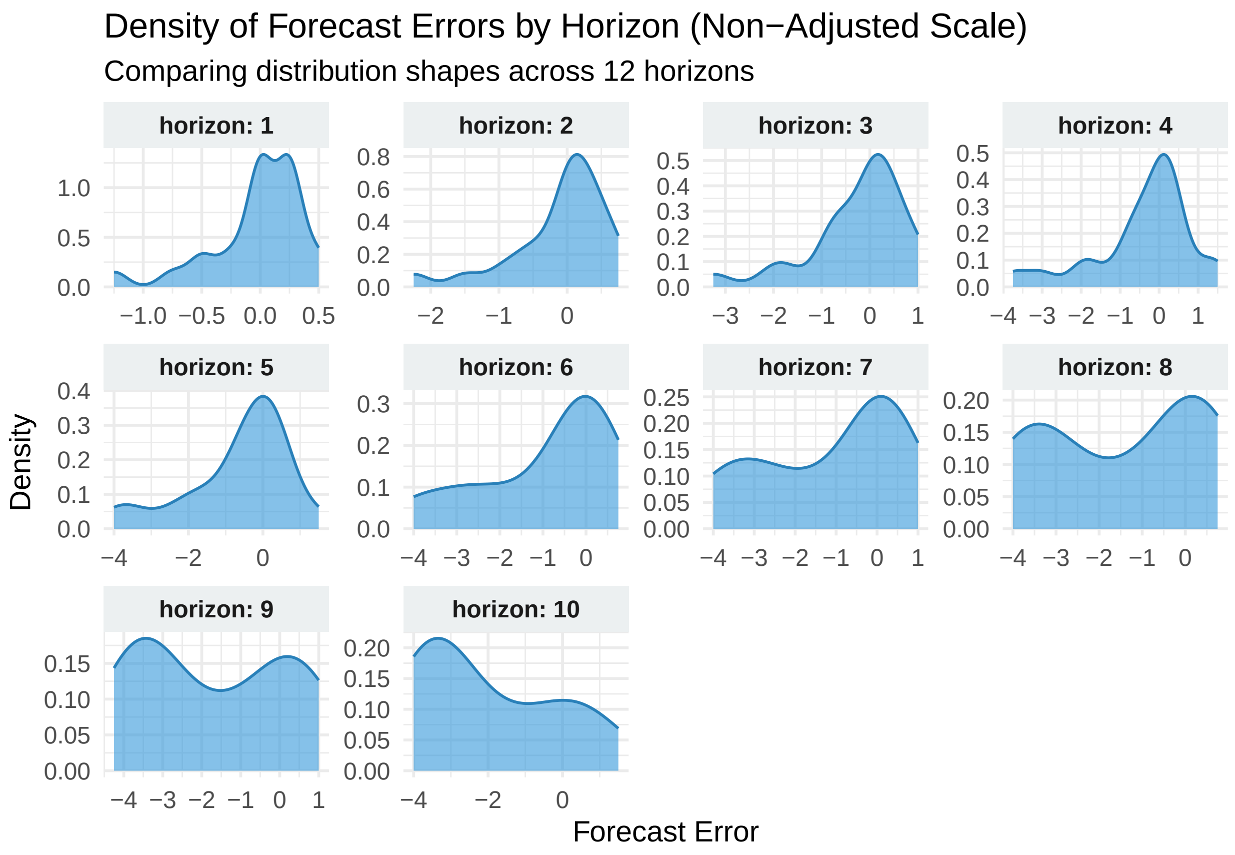
```r
df_all_3 <- df_all

# 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)"
  ) +
  theme_minimal()
```



Density of FE

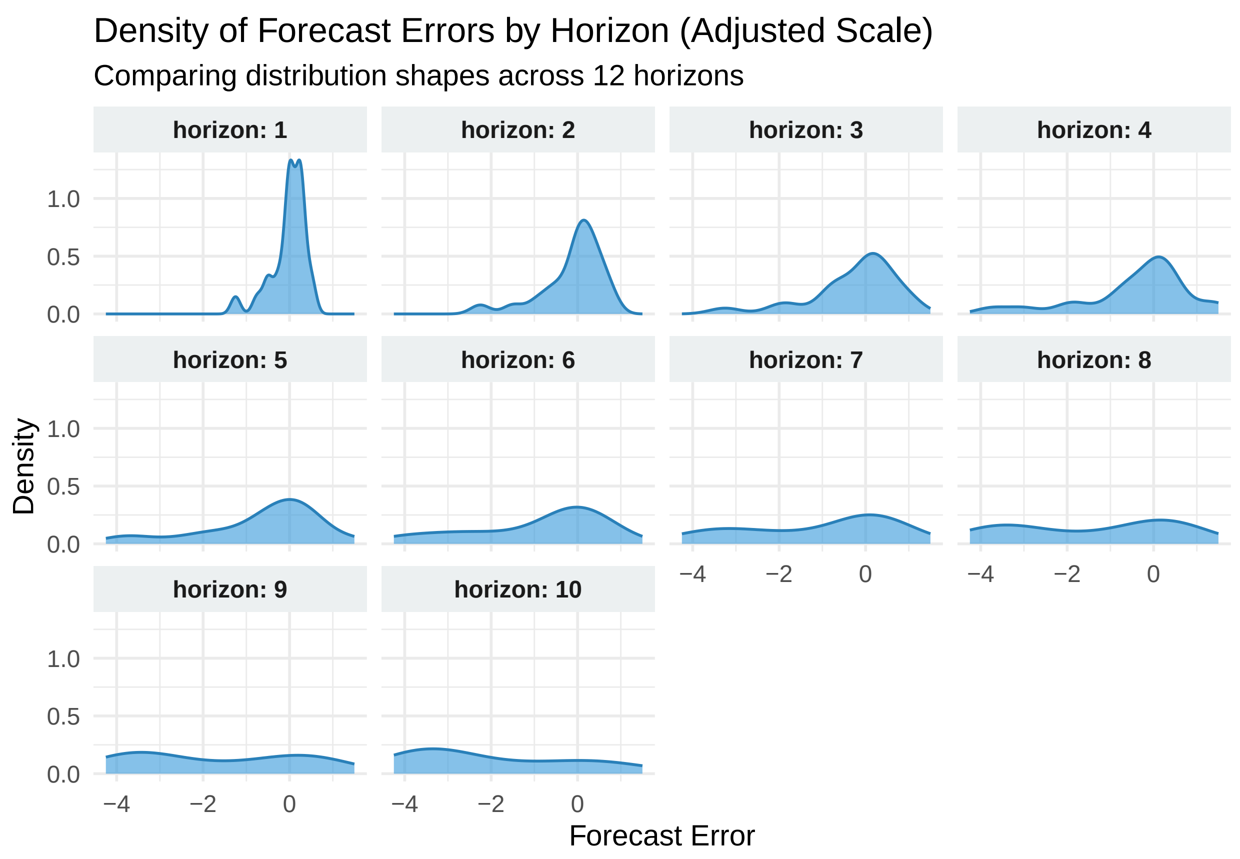
```
# Version 1: Non-Adjusted Scales
plot_facet <- ggplot(df_all_3 %>% filter(horizon <= H), aes(x = forecast_error)) +
  geom_density(fill = "#3498db", color = "#2980b9", alpha = 0.6) +
  facet_wrap(~horizon, ncol = 4, labeller = label_both, scales = "free") +
  labs(
    title = "Density of Forecast 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)
```

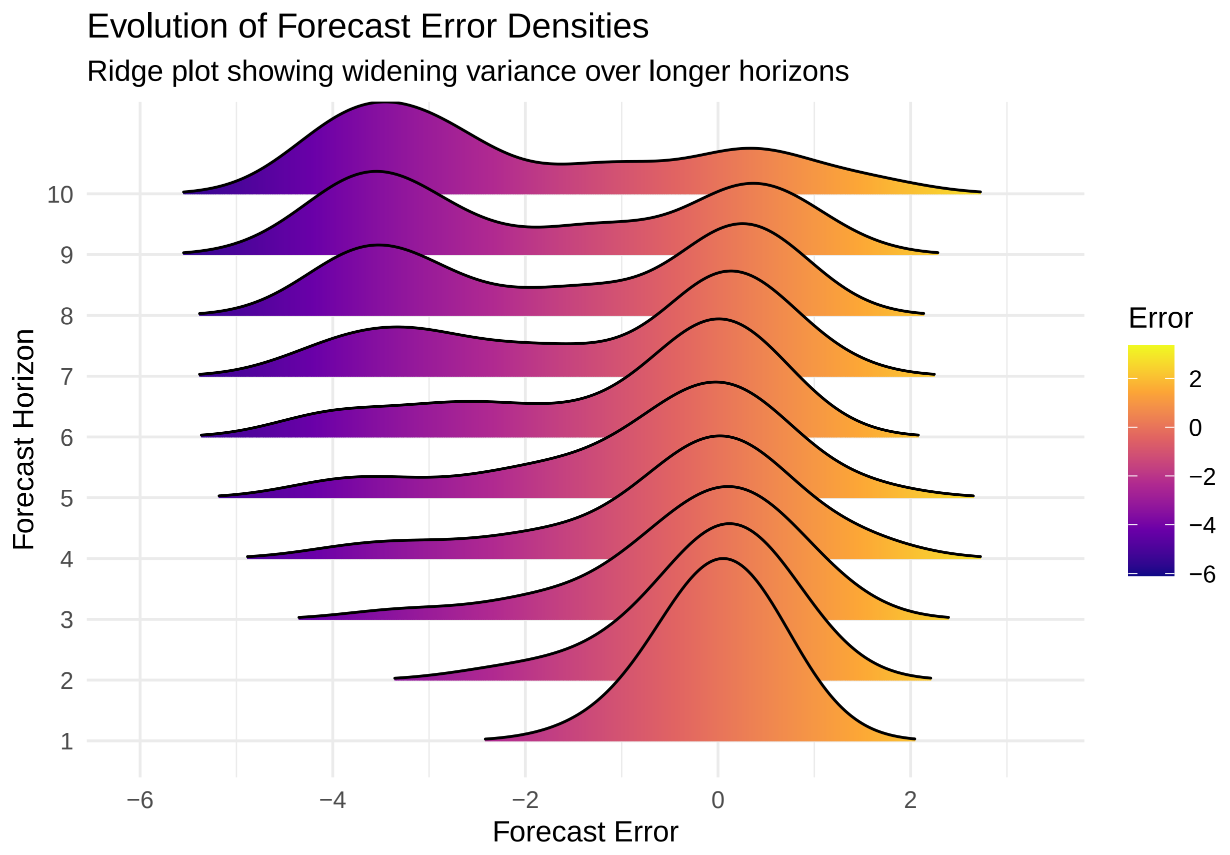
```
## Warning: Removed 45 rows containing non-finite outside the scale range
## (`stat_density()`).
```



```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)
```

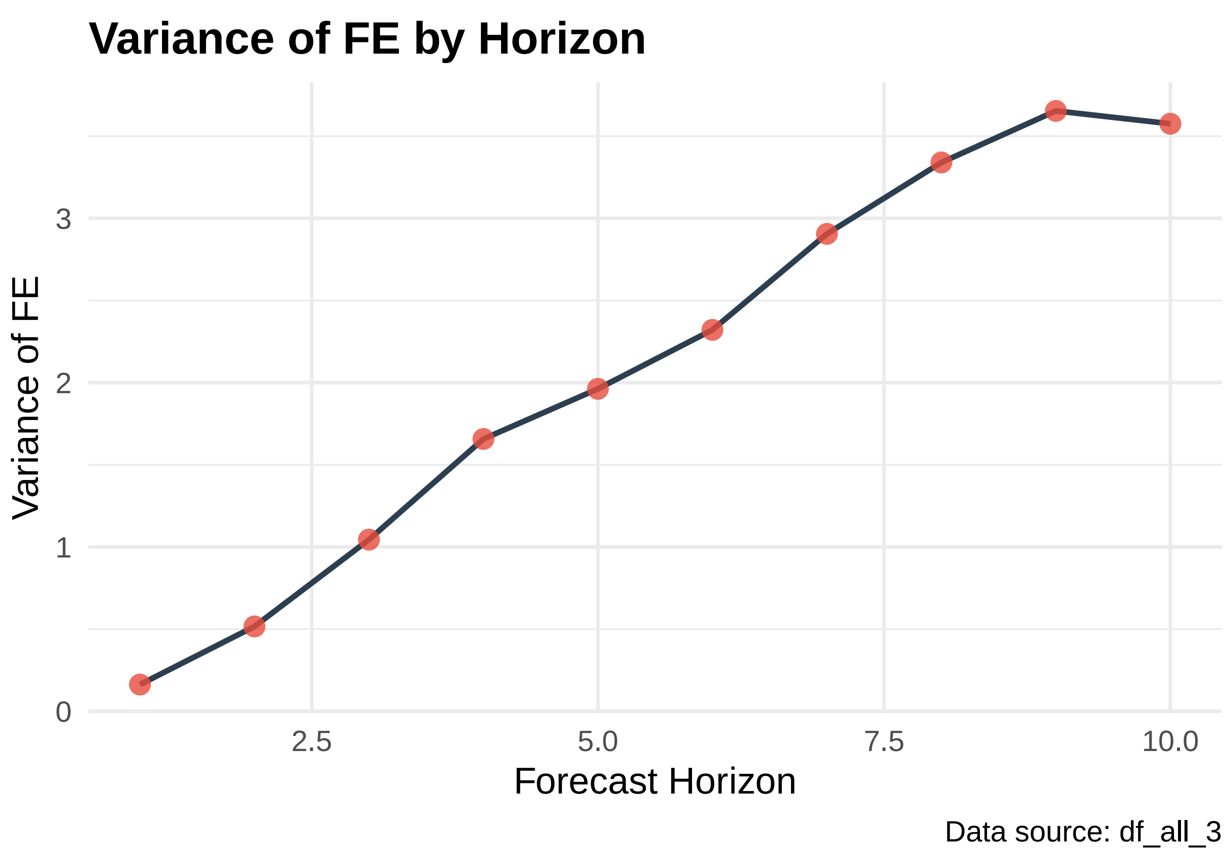
```
## Warning: `stat(x)` was deprecated in ggplot2 3.4.0.
## i Please use `after_stat(x)` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```

```
## Picking joint bandwidth of 0.611
```



Variance of FE

```r
# 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

```r
# 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
mz_report_2 <- 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)

# MZ Report 3: Actual Rate, with Lag
mz_report_3 <- generate_mincer_zarnowitz_report(
  F_model = F_TR_3,
  Actual_values = Actuals,
```

Table 5: Mincer-Zarnowitz Test: Actual Rate, No Lag

h	Alpha	Beta	pv(Joint)
1	1.0512	0.2219	0.2226
2	1.1581	0.1956	0.7090
3	1.1232	0.3446	0.6866
4	0.8490	0.8545	0.9113
5	0.7444	1.2223	0.5018
6	0.8941	1.2663	0.0807 *
7	0.9535	1.3771	0.0003 ***
8	1.2814	1.1158	0.0000 ***
9	1.6787	0.7325	0.3333
10	2.0251	0.4582	0.1465

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

```
  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]]

Table 6: Mincer-Zarnowitz Test: Shadow Rate, No Lag

h	Alpha	Beta	pv(Joint)
1	1.3812	-0.2422	0.0001 ***
2	1.4354	-0.1829	0.0030 ***
3	1.4142	-0.0093	0.0042 ***
4	1.3914	0.3145	0.1917
5	1.4927	0.5364	0.5661
6	1.6769	0.5108	0.3788
7	1.8265	0.4932	0.0225 **
8	1.9438	0.4562	0.0022 ***
9	2.0931	0.4083	0.0000 ***
10	2.2622	0.3241	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

[[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,
  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,
```

Table 7: Mincer-Zarnowitz Test: Actual Rate, with Lag

h	Alpha	Beta	pv(Joint)
1	0.0318	0.9910	0.9547
2	0.1412	0.9733	0.8887
3	0.3420	0.9297	0.8389
4	0.5939	0.8457	0.7941
5	0.8083	0.8445	0.7716
6	1.1034	0.8094	0.7547
7	1.3835	0.6610	0.6467
8	1.6946	0.5214	0.5102
9	1.9901	0.3454	0.2565
10	2.2406	0.2096	0.1378

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, with Lag

h	Alpha	Beta	pv(Joint)
1	0.0361	0.9104	0.0731 *
2	0.1746	0.8087	0.3006
3	0.3989	0.6838	0.0211 **
4	0.6619	0.5632	0.0001 ***
5	0.9611	0.4563	0.0000 ***
6	1.2182	0.3705	0.0000 ***
7	1.4356	0.2922	0.0000 ***
8	1.7190	0.1990	0.0000 ***
9	2.0085	0.1050	0.0000 ***
10	2.2865	0.0155	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 9: Mincer-Zarnowitz Test: Benchmark ARIMA

h	Alpha	Beta	pv(Joint)
1	0.1195	0.9777	0.4059
2	0.3259	0.9110	0.5332
3	0.5703	0.7937	0.4930
4	0.8485	0.6840	0.5591
5	1.1376	0.5605	0.5944
6	1.4127	0.4527	0.4491
7	1.6755	0.3248	0.5928
8	1.9108	0.2390	0.2074
9	2.1203	0.1002	0.0021 ***
10	2.2923	-0.0423	0.0012 ***

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
  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]]

[[2]]

[[3]]

[[4]]

Table 10: MSFE Comparison, Trained on Actual Rate, No Lag

h	MSFE TR	MSFE BM	Ratio	DM Two-Sided	DM Greater	DM Lesser
1	4.0208	0.1276	31.5102	0.0001 ***	1.0000	0.0000 ***
2	4.2092	0.5842	7.2047	0.0253 **	0.9873	0.0127 **
3	4.1222	1.4460	2.8507	0.1877	0.9062	0.0938 *
4	3.6667	2.4643	1.4879	0.6016	0.6992	0.3008
5	3.5187	3.6750	0.9575	0.9464	0.4732	0.5268
6	3.6743	4.8618	0.7558	0.5916	0.2958	0.7042
7	3.8889	6.1632	0.6310	0.0724 *	0.0362 **	0.9638
8	4.3897	7.2169	0.6083	0.0530 *	0.0265 **	0.9735
9	5.0508	8.4648	0.5967	0.0194 **	0.0097 ***	0.9903
10	5.6583	9.6125	0.5886	0.0050 ***	0.0025 ***	0.9975

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 11: MSFE Comparison, Trained on Shadow Rate, No Lag

h	MSFE TR	MSFE BM	Ratio	DM Two-Sided	DM Greater	DM Lesser
1	7.1667	0.1276	56.1633	0.0001 ***	0.9999	0.0001 ***
2	7.4049	0.5842	12.6744	0.0196 **	0.9902	0.0098 ***
3	6.8381	1.4460	4.7289	0.0844 *	0.9578	0.0422 **
4	5.5982	2.4643	2.2717	0.3519	0.8241	0.1759
5	5.3375	3.6750	1.4524	0.6063	0.6968	0.3032
6	6.1151	4.8618	1.2578	0.6459	0.6771	0.3229
7	6.6910	6.1632	1.0856	0.6954	0.6523	0.3477
8	7.0441	7.2169	0.9761	0.9157	0.4578	0.5422
9	7.5898	8.4648	0.8966	0.5766	0.2883	0.7117
10	8.2792	9.6125	0.8613	0.3493	0.1747	0.8253

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

h	MSFE TR	MSFE BM	Ratio	DM Two-Sided	DM Greater	DM Lesser
1	0.1562	0.1276	1.2245	0.2781	0.8610	0.1390
2	0.5054	0.5842	0.8651	0.3822	0.1911	0.8089
3	1.0653	1.4460	0.7367	0.1776	0.0888 *	0.9112
4	1.7619	2.4643	0.7150	0.1832	0.0916 *	0.9084
5	2.3031	3.6750	0.6267	0.1193	0.0596 *	0.9404
6	3.1217	4.8618	0.6421	0.0723 *	0.0361 **	0.9639
7	4.0729	6.1632	0.6608	0.0388 **	0.0194 **	0.9806
8	5.2206	7.2169	0.7234	0.0261 **	0.0131 **	0.9869
9	6.4336	8.4648	0.7600	0.0028 ***	0.0014 ***	0.9986
10	7.4042	9.6125	0.7703	0.0076 ***	0.0038 ***	0.9962

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

h	MSFE TR	MSFE BM	Ratio	DM Two-Sided	DM Greater	DM Lesser
1	0.1719	0.1276	1.3469	0.2501	0.8749	0.1251
2	0.5734	0.5842	0.9814	0.9427	0.4713	0.5287
3	1.4034	1.4460	0.9705	0.7480	0.3740	0.6260
4	2.6339	2.4643	1.0688	0.8525	0.5738	0.4262
5	4.2750	3.6750	1.1633	0.2793	0.8603	0.1397
6	6.0099	4.8618	1.2361	0.3185	0.8407	0.1593
7	7.4792	6.1632	1.2135	0.2384	0.8808	0.1192
8	9.4412	7.2169	1.3082	0.2964	0.8518	0.1482
9	11.3398	8.4648	1.3396	0.3356	0.8322	0.1678
10	12.9917	9.6125	1.3515	0.3558	0.8221	0.1779

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

```
#------------------------------------------------------------------
# Helper function for displaying our final forecast results
#------------------------------------------------------------------

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) }
```

Forecasting

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

  # --- 1. Fit inputs and benchmark models  ---
```

```r
   inflation_arma <- auto.arima(data$inflation_gap, max.p=4, max.q=4, max.d=1)
   #exp_inflation_arma <- auto.arima(data$exp_inflation_gap, max.p=4, max.q=4, max.d=1)
   outputgap_arma <- auto.arima(data$output_gap, max.p=4, max.q=4, max.d=1)
   interest_arma <- auto.arima(data$rate, max.p=4, max.q=4, max.d=1) # Benchmark

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

  # --- 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)

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

Table 14

Horizon: Quarter	Taylor Rule Forecast	Benchmark Forecast	Inflation Forecast	Output Gap Forecast
1: 2025 Q3	2.00	1.75	1.97	-0.20
2: 2025 Q4	1.75	1.25	2.03	-0.12
3: 2026 Q1	1.75	1.25	1.83	-0.07
4: 2026 Q2	1.75	1.00	1.88	-0.04
5: 2026 Q3	1.75	1.00	1.91	-0.03
6: 2026 Q4	1.50	1.00	1.94	-0.02
7: 2027 Q1	1.50	1.25	1.96	-0.01
8: 2027 Q2	1.50	1.25	1.97	-0.01
9: 2027 Q3	1.50	1.25	1.98	0.00
10: 2027 Q4	1.50	1.25	1.98	0.00