

ECB Tests 6

Contents

Data	2
Main	2
Data Properties	3
Taylor Rule Estimation	6
Without Lag	6
With Lag (interest rate smoothing)	6
Forecasting Model Evaluation	8
Helpers	8
Estimation	12
Spaghetti Plots	16
Plots of FE	17
Variance of FE	21
Absolute Performance: Efficiency & Bias	22
Relative Performance (against benchmark)	24
Actual Forecast Model	29
Helpers	29
Forecasting	29

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

Data

Main

#REDO THE DATA LOADING USING rdbnomics PACKAGE (combines all macro APIs)

--- 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)

```

Data Properties

```

# Interest rate is I(1)
test1 = 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

```

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

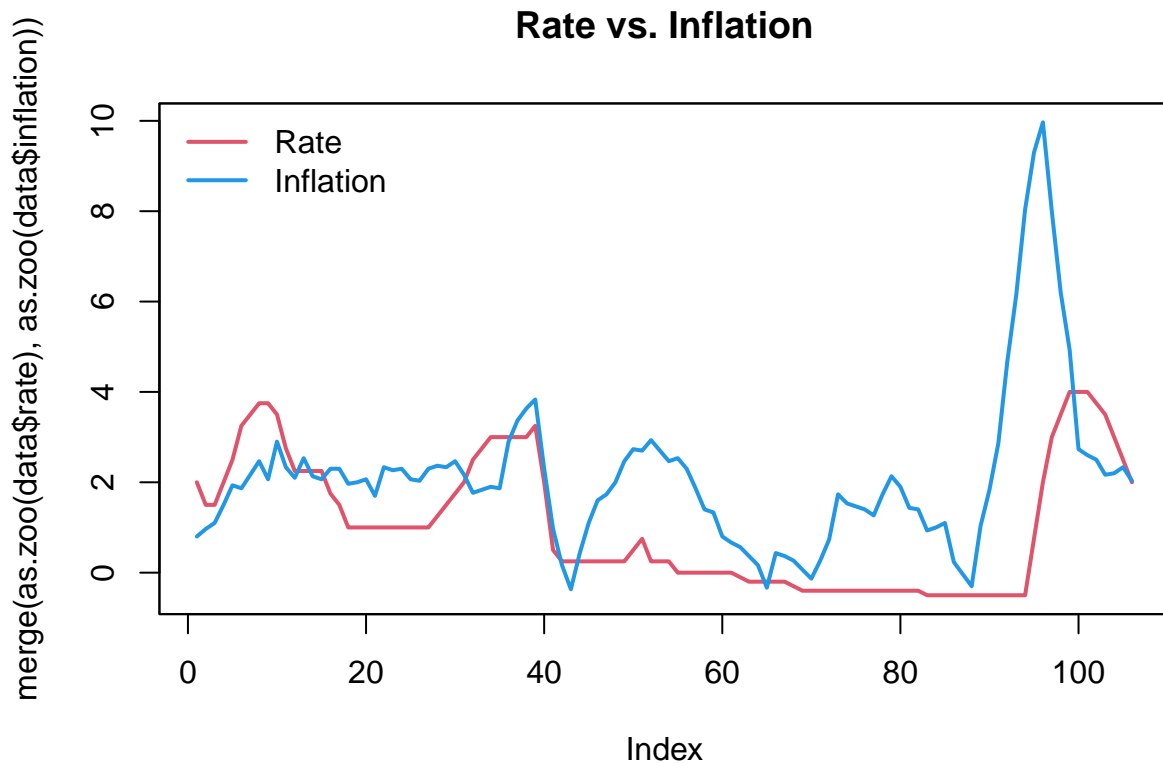
```
##      lag      ADF    p.value
## [1,]  0 -1.125153 0.27478730
## [2,]  1 -2.251445 0.02452223
## [3,]  2 -2.472595 0.01529054
## [4,]  3 -2.263254 0.02402930
## [5,]  4 -1.490401 0.14320596
```

```
# Output gap is I(0), likely from the "gap" part
test3 = 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
```

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

```
# Are interest rates and inflation co-integrated?
plot(merge(as.zoo(data$rate), as.zoo(data$inflation)), plot.type="single", col=c(2, 4), lwd=2, main="Ra
```



```
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 Lag

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

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

export_summs(TR, TRsr, vcov = sandwich::NeweyWest,
             model.names = c("TR", "TR w/ SR"), digits = 4)
```

	TR	TR w/ SR
(Intercept)	1.0305 (0.9157)	-0.4628 (2.5157)
inflation_gap	0.2278 (0.3621)	0.5690 (0.5988)
output_gap	0.1128 (0.2528)	0.0784 (0.4505)
N	106	106
R2	0.1318	0.0965

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

```
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)

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)
```

With Lag (interest rate smoothing)

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

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

export_summs(lTR, lTRsr, vcov = sandwich::NeweyWest,
             model.names = c("TR", "TR w/ SR"), digits = 4)
```

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

```
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)
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)
```

	TR	TR w/ SR
(Intercept)	0.0540 (0.0420)	-0.0444 (0.0637)
rate_lag	0.9371 *** (0.0411)	
inflation_gap	0.0981 * (0.0388)	0.2367 *** (0.0312)
output_gap	0.0166 (0.0206)	-0.0137 (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.

Forecasting Model Evaluation

Helpers

```
#-----
# 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,
```

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

```

                                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-value indicates that the null hypothesis is not rejected.",
    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 (check with Viktor)

#-----
# 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)}

# 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
  training <- data[1:(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 <- pmax(predict(TR_model_1, new_data_base), min(data$rate))
TR_preds_2 <- pmax(predict(TR_model_2, new_data_base), min(data$rate))
BM_preds <- pmax(BMpredicted_rates, min(data$rate)) # Benchmark

# --- 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
  pred_3_h <- pmax(predict(TR_model_3, new_data_3_h), min(data$rate))
  TR_preds_3[h] <- pred_3_h
  pred_4_h <- pmax(predict(TR_model_4, new_data_4_h), min(data$rate))
  TR_preds_4[h] <- pred_4_h

  # 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"

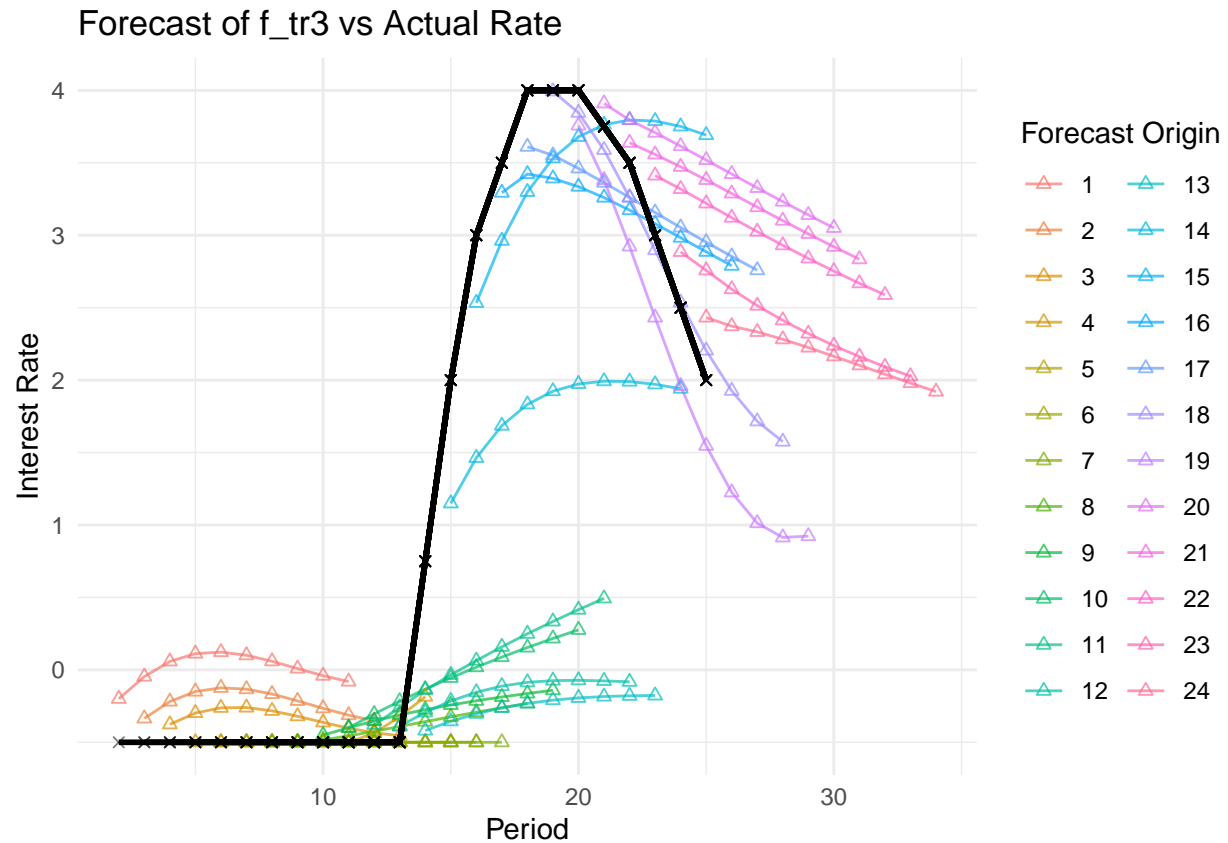
# period = date the forecast was made
# date_of_forecast = the future date we are predicting
df_all$date_of_forecast <- df_all$period + df_all$horizon # is this correct choice?

# 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

```
df_all_3 <- df_all

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

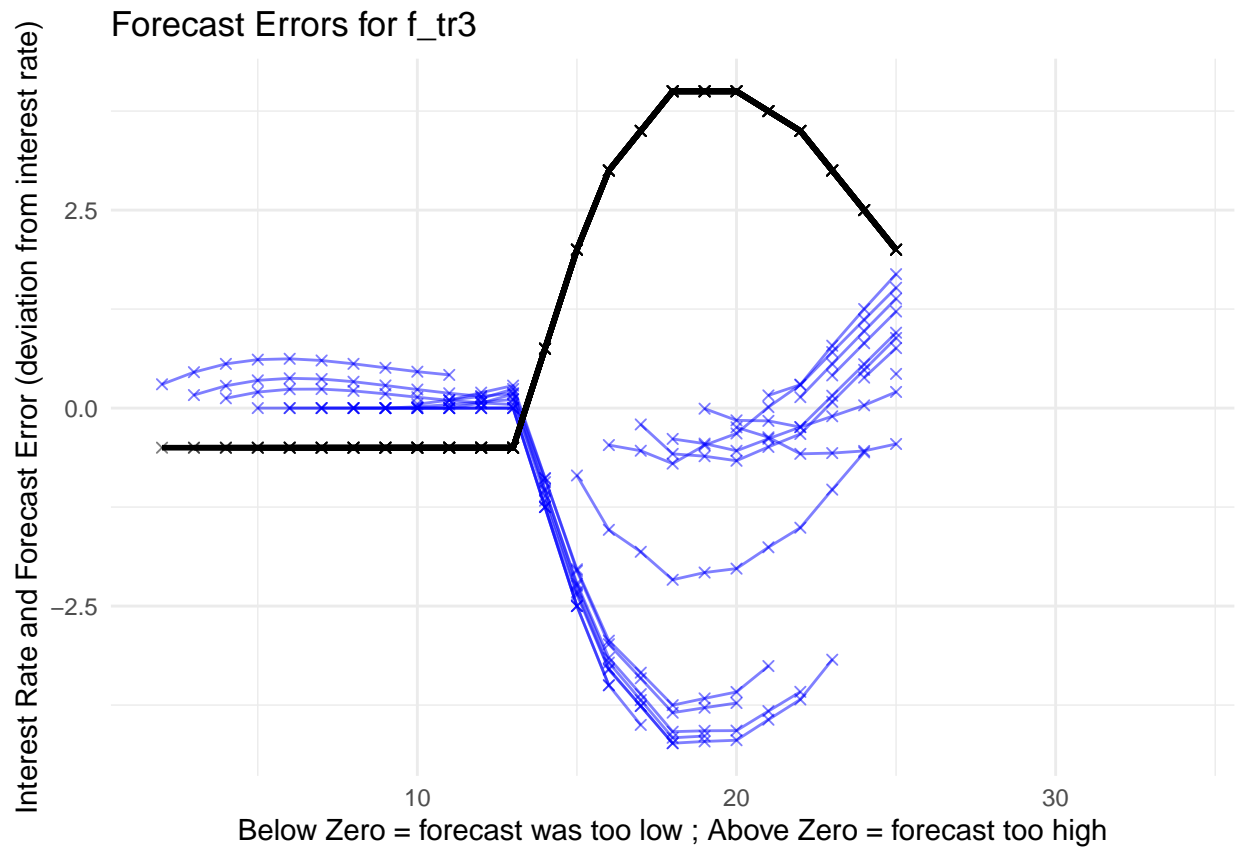
#compute date_of_forecast for x-axis
df_all_3$date_of_forecast <- df_all_3$period + df_all_3$horizon

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()
```



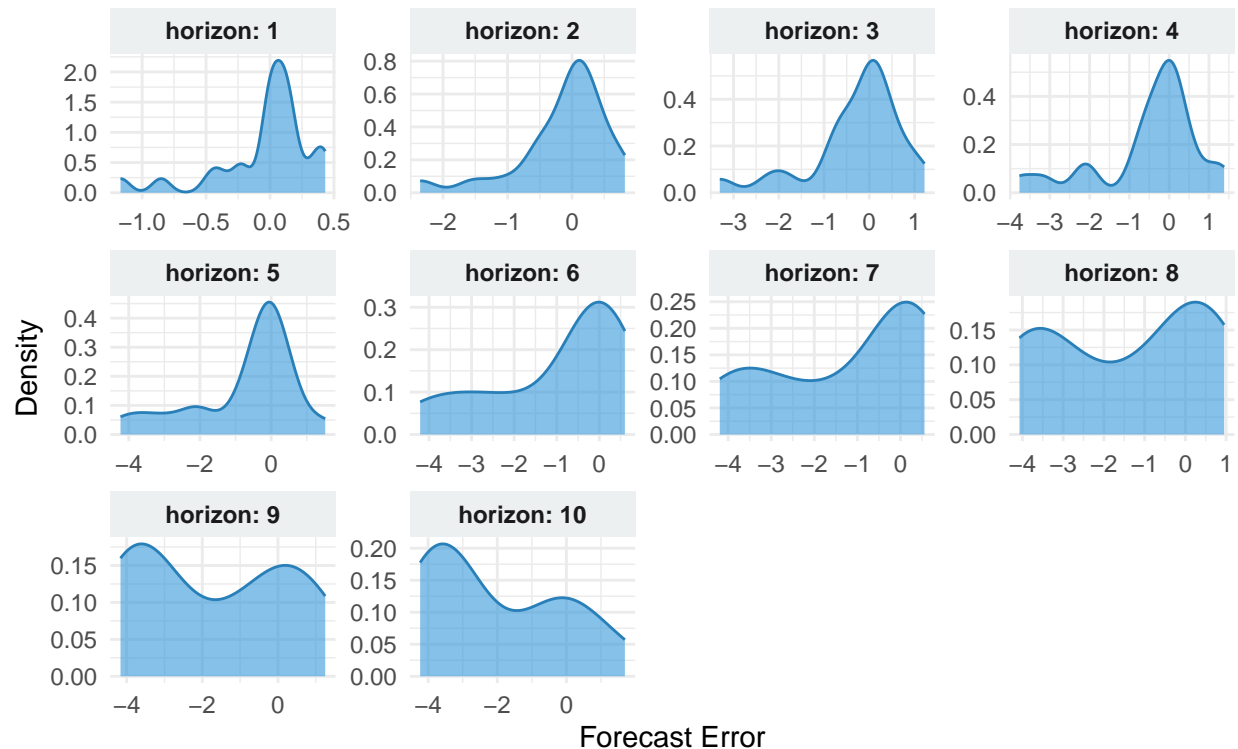
```
## Density of FE
```

```
# 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()`).
```

Density of Forecast Errors by Horizon (Non-Adjusted Scale)

Comparing distribution shapes across 12 horizons

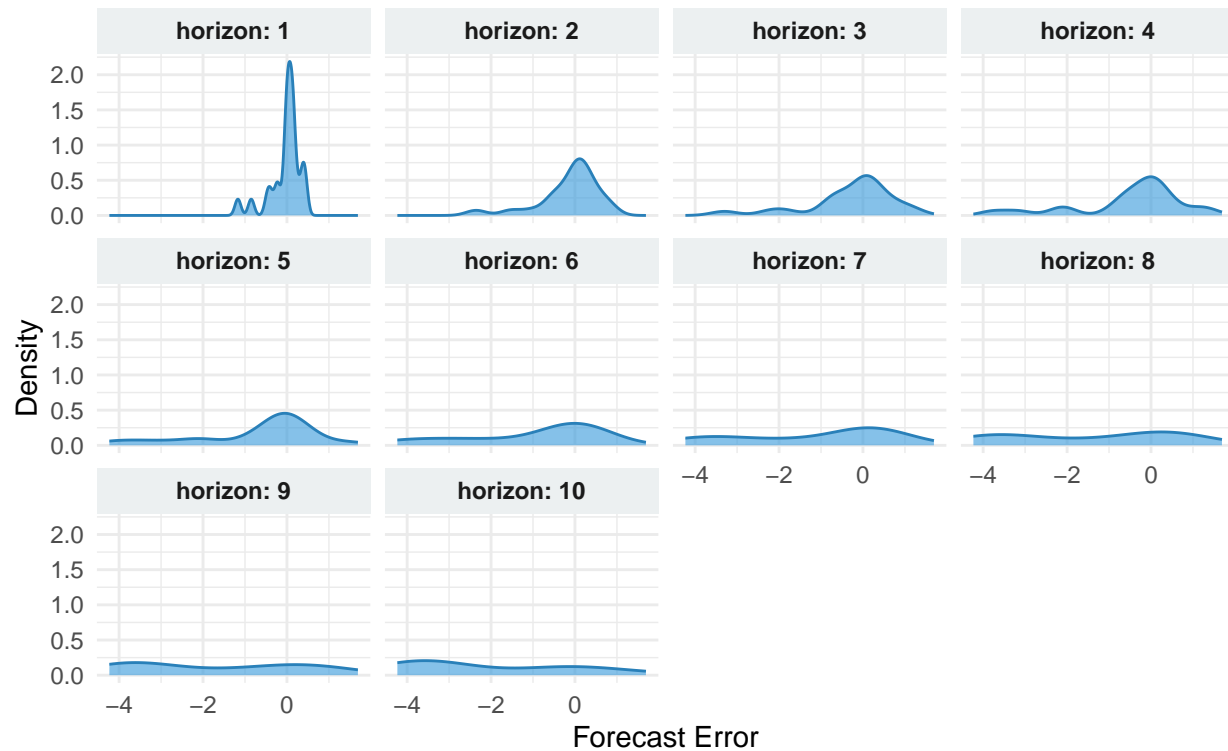


```
# Version 2: Adjusted scales
plot_facet <- ggplot(df_all_3 %>% filter(horizon <= H), aes(x = forecast_error)) +
  geom_density(fill = "#3498db", color = "#2980b9", alpha = 0.6) +
  facet_wrap(~horizon, ncol = 4, labeller = label_both) +
  labs(title = "Density of Forecast Errors by Horizon (Adjusted 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()`).
```

Density of Forecast Errors by Horizon (Adjusted Scale)

Comparing distribution shapes across 12 horizons



```
# 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()

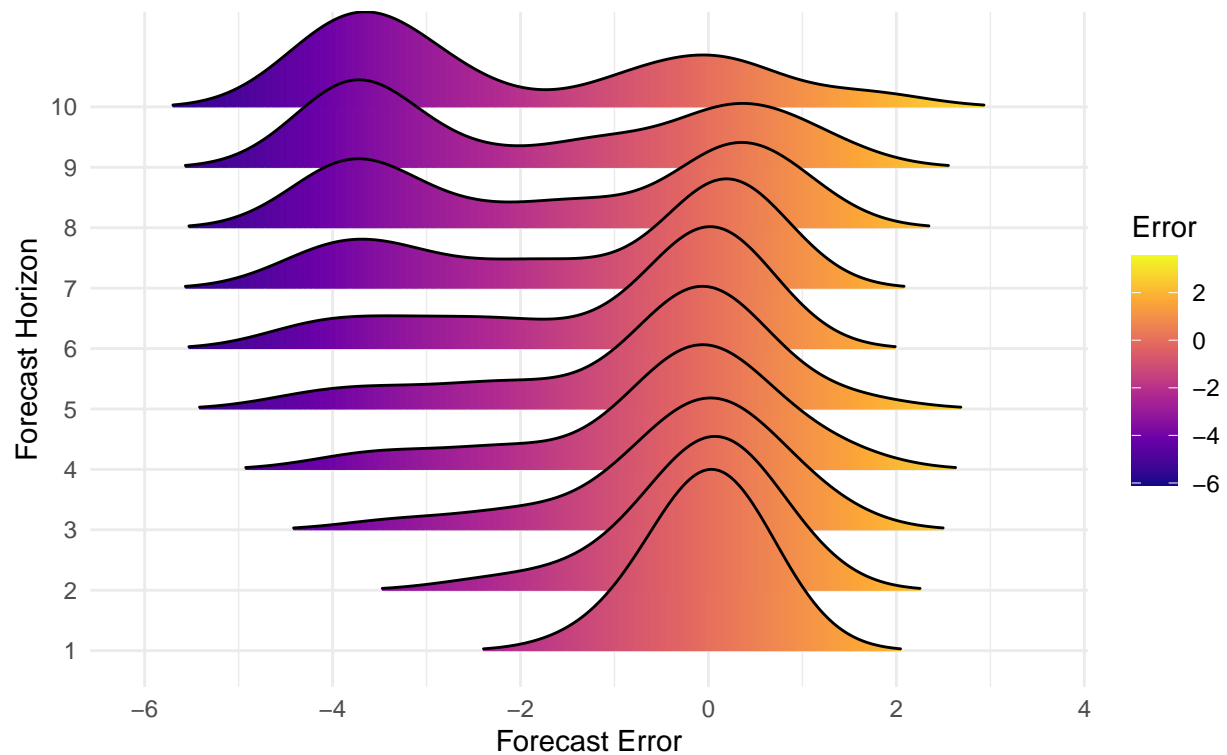
print(plot_ridge)
```

```
## 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 0.621
```

Evolution of Forecast Error Densities

Ridge plot showing widening variance over longer horizons

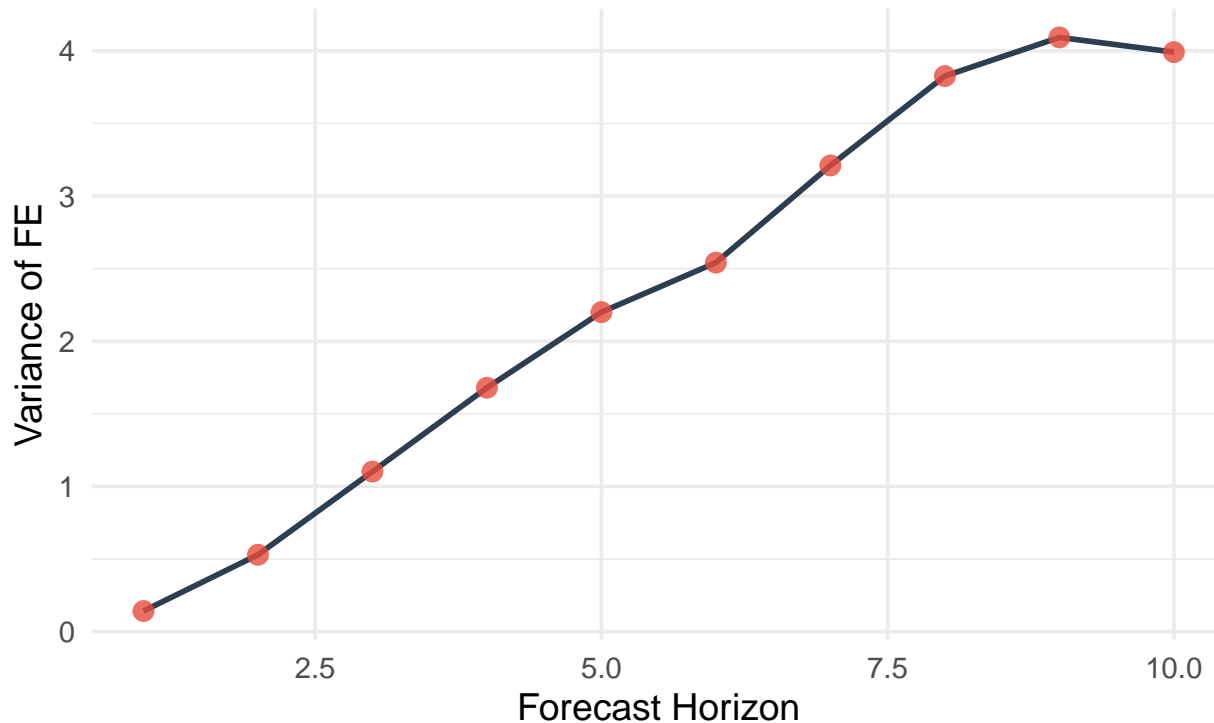


Variance of FE

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

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

Variance of FE by Horizon



Data source: df_all_3

Absolute Performance: Efficiency & Bias

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

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

# MZ Report 2: Shadow Rate, No Lag
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 1: Mincer-Zarnowitz Test: Actual Rate, No Lag

h	Alpha	Beta	pv(Joint)
1	0.8308	0.3870	0.3344
2	0.8883	0.4299	0.8445
3	0.8157	0.6148	0.8468
4	0.6349	0.9893	0.9114
5	0.5547	1.3052	0.5538
6	0.6150	1.4821	0.0831 *
7	0.7550	1.4872	0.0007 ***
8	1.1300	1.1887	0.0000 ***
9	1.6153	0.7628	0.0937 *
10	2.0229	0.4338	0.0461 **

Note:

pv(Joint) is the p-value for the joint hypothesis H_0 : $(\text{Alpha}, \text{Beta}) = (0, 1)$. 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 2: Mincer-Zarnowitz Test: Shadow Rate, No Lag

h	Alpha	Beta	pv(Joint)	
1	1.3655	-0.1584	0.0001	***
2	1.3924	-0.0943	0.0127	**
3	1.3659	0.0666	0.0111	**
4	1.3495	0.3085	0.0557	*
5	1.4272	0.5167	0.2818	
6	1.6067	0.5540	0.3903	
7	1.7856	0.5251	0.2538	
8	1.9180	0.4971	0.0736	*
9	2.0707	0.4321	0.0009	***
10	2.2503	0.3403	0.0000	***

Note:

pv(Joint) is the p-value for the joint hypothesis $H_0: (\text{Alpha}, \text{Beta}) = (0, 1)$. 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)

```
# 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 3: Mincer-Zarnowitz Test: Actual Rate, with Lag

h	Alpha	Beta	pv(Joint)
1	0.0450	0.9916	0.8970
2	0.1846	0.9584	0.8210
3	0.3971	0.9008	0.7833
4	0.6508	0.8429	0.7835
5	0.9245	0.7853	0.7522
6	1.1946	0.7515	0.7090
7	1.4749	0.5792	0.6132
8	1.7624	0.3992	0.2979
9	2.0356	0.2406	0.1236
10	2.2816	0.0865	0.0421 **

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 4: Mincer-Zarnowitz Test: Shadow Rate, with Lag

h	Alpha	Beta	pv(Joint)
1	0.0343	0.9149	0.0615 *
2	0.0628	0.8495	0.2948
3	0.2173	0.7566	0.0488 **
4	0.4621	0.6598	0.0019 ***
5	0.7528	0.5621	0.0001 ***
6	1.0264	0.4793	0.0001 ***
7	1.2330	0.3919	0.0000 ***
8	1.5322	0.2764	0.0000 ***
9	1.8895	0.1622	0.0000 ***
10	2.2269	0.0591	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 5: Mincer-Zarnowitz Test: Benchmark ARIMA

h	Alpha	Beta	pv(Joint)
1	0.0928	0.9668	0.5402
2	0.2858	0.8951	0.3440
3	0.5396	0.7897	0.4567
4	0.8213	0.6816	0.5245
5	1.1148	0.5559	0.5748
6	1.3959	0.4445	0.4233
7	1.6639	0.3159	0.5625
8	1.9040	0.2228	0.1076
9	2.1176	0.0817	0.0000 ***
10	2.2893	-0.0756	0.0113 **

Note:

pv(Joint) is the p-value for the joint hypothesis $H_0: (\text{Alpha}, \text{Beta}) = (0, 1)$. 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

```
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 6: MSFE Comparison, Trained on Actual Rate, No Lag

h	MSFE TR	MSFE BM	Ratio	DM Two-Sided	DM Greater	DM Lesser
1	3.7472	0.1053	35.5873	0.0001 ***	1.0000	0.0000 ***
2	3.7666	0.5505	6.8416	0.0298 **	0.9851	0.0149 **
3	3.5465	1.3757	2.5780	0.2196	0.8902	0.1098
4	3.1477	2.3914	1.3163	0.7202	0.6399	0.3601
5	2.9263	3.6192	0.8086	0.7604	0.3802	0.6198
6	2.9383	4.8317	0.6081	0.3802	0.1901	0.8099
7	3.1641	6.1507	0.5144	0.0524 *	0.0262 **	0.9738
8	3.7850	7.2214	0.5241	0.0000 ***	0.0000 ***	1.0000
9	4.7113	8.4776	0.5557	0.0111 **	0.0056 ***	0.9944
10	5.5903	9.6411	0.5798	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 7: MSFE Comparison, Trained on Shadow Rate, No Lag

h	MSFE TR	MSFE BM	Ratio	DM Two-Sided	DM Greater	DM Lesser
1	7.2945	0.1053	69.2765	0.0002 ***	0.9999	0.0001 ***
2	7.4398	0.5505	13.5135	0.0186 **	0.9907	0.0093 ***
3	6.7284	1.3757	4.8909	0.0717 *	0.9642	0.0358 **
4	5.4524	2.3914	2.2800	0.3595	0.8202	0.1798
5	5.0445	3.6192	1.3938	0.6604	0.6698	0.3302
6	5.5592	4.8317	1.1506	0.7748	0.6126	0.3874
7	6.2884	6.1507	1.0224	0.8947	0.5526	0.4474
8	6.7025	7.2214	0.9281	0.7352	0.3676	0.6324
9	7.3098	8.4776	0.8623	0.4320	0.2160	0.7840
10	8.1174	9.6411	0.8420	0.2677	0.1339	0.8661

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

h	MSFE TR	MSFE BM	Ratio	DM Two-Sided	DM Greater	DM Lesser
1	0.1368	0.1053	1.2994	0.1862	0.9069	0.0931 *
2	0.5241	0.5505	0.9520	0.6187	0.3094	0.6906
3	1.1336	1.3757	0.8240	0.2100	0.1050	0.8950
4	1.8422	2.3914	0.7703	0.1422	0.0711 *	0.9289
5	2.6375	3.6192	0.7288	0.0826 *	0.0413 **	0.9587
6	3.4562	4.8317	0.7153	0.0541 *	0.0270 **	0.9730
7	4.5072	6.1507	0.7328	0.0406 **	0.0203 **	0.9797
8	5.7004	7.2214	0.7894	0.0233 **	0.0116 **	0.9884
9	6.9107	8.4776	0.8152	0.0004 ***	0.0002 ***	0.9998
10	8.0803	9.6411	0.8381	0.0070 ***	0.0035 ***	0.9965

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

h	MSFE TR	MSFE BM	Ratio	DM Two-Sided	DM Greater	DM Lesser
1	0.1504	0.1053	1.4281	0.0498 **	0.9751	0.0249 **
2	0.4212	0.5505	0.7651	0.6666	0.3333	0.6667
3	0.9107	1.3757	0.6620	0.4783	0.2391	0.7609
4	1.6128	2.3914	0.6744	0.4534	0.2267	0.7733
5	2.6493	3.6192	0.7320	0.4444	0.2222	0.7778
6	3.8501	4.8317	0.7968	0.2591	0.1295	0.8705
7	5.2563	6.1507	0.8546	0.6748	0.3374	0.6626
8	7.3856	7.2214	1.0227	0.7535	0.6233	0.3767
9	9.8299	8.4776	1.1595	0.4981	0.7510	0.2490
10	12.3017	9.6411	1.2760	0.4282	0.7859	0.2141

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$TR_Forecast,4),  
    Benchmark_ARIMA_Forecast = round(forecast_list$BM_Forecast,4))  
  
  # Create the table  
  table_output <- kable(  
    forecast_df,  
    format = format,  
    digits = 4,  
    col.names = c("Horizon: Quarter", "Taylor Rule Forecast", "Benchmark 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 ---  
  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 <- pmax(predict(TR_model, new_data_h), min(data$rate))
  TR_preds[h] <- pred_h

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

# --- 5. Compute forecast for BM ---
BM_preds <- pmax(BMpredicted_rates, min(data$rate)) # Benchmark

return(list(TR_Forecast = TR_preds, BM_Forecast = BM_preds))}

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

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

```

Table 10

Horizon: Quarter	Taylor Rule Forecast	Benchmark Forecast
1: 2025 Q3	1.9221	1.6263
2: 2025 Q4	1.8562	1.3672
3: 2026 Q1	1.7761	1.2053
4: 2026 Q2	1.7060	1.1212
5: 2026 Q3	1.6439	1.0953
6: 2026 Q4	1.5882	1.1101
7: 2027 Q1	1.5378	1.1504
8: 2027 Q2	1.4919	1.2043
9: 2027 Q3	1.4498	1.2626
10: 2027 Q4	1.4110	1.3189