

Tracy_Project_I.r

tracywang

2025-10-02

```
#Code for downloading state level data from FRED
library(quantmod)

## Loading required package: xts

## Loading required package: zoo

##
## Attaching package: 'zoo'

## The following objects are masked from 'package:base':
##
##   as.Date, as.Date.numeric

## Loading required package: TTR

## Registered S3 method overwritten by 'quantmod':
##   method      from
##   as.zoo.data.frame zoo

library(lubridate)

##
## Attaching package: 'lubridate'

## The following objects are masked from 'package:base':
##
##   date, intersect, setdiff, union

series <- "PAUR"
end_date <- lubridate::ymd("2025-06-30")
start_date <- end_date %m-% years(30)

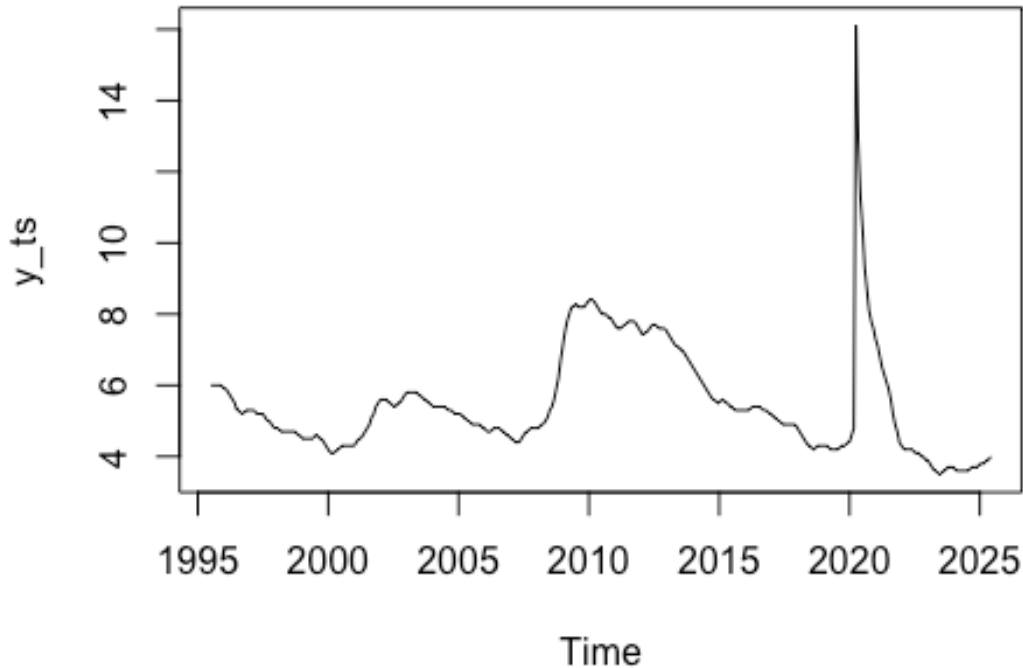
suppressWarnings(
  getSymbols(Symbols = series, src = "FRED",
    from = start_date, to = end_date, auto.assign = TRUE)
)

## [1] "PAUR"

x <- get(series); colnames(x) <- "value"

t0 <- as.Date(head(index(x),1))
ts_start <- c(year(t0), month(t0))
y_ts <- ts(as.numeric(x$value), start = ts_start, frequency = 12)
```

```
plot(y_ts)
```



```
# =====  
# BOX-JENKINS: STEP 1 – IDENTIFICATION  
# Purpose: Determine (p, d, q) and, if seasonal, (P, D, Q)[m]  
# Input: y_ts (monthly ts built in your preceding code)  
# =====  
  
needed <- c("forecast", "tseries", "urca")  
to_install <- setdiff(needed, rownames(installed.packages()))  
if (length(to_install)) install.packages(to_install, repos = "https://cloud.r-project.org")  
library(forecast) # ACF/PACF, seasonplot, ndiffs, nsdiffs  
library(tseries) # kpss.test(), adf.test()  
  
# Ensure numeric, finite series  
y_ts <- stats::ts(as.numeric(y_ts), start = start(y_ts), frequency = frequency(y_ts))  
m <- frequency(y_ts) # 12 for monthly  
  
# =====  
# 1) STATIONARITY CHECK
```

```

# Visual inspection + statistical tests (ADF, KPSS)
# =====
par(mfrow = c(2,2))
plot(y_ts, main = "Level series y_t (visual stationarity check)", ylab = "Percent", xlab = "")
Acf(y_ts, lag.max = 72, main = "ACF (level)")
Pacf(y_ts, lag.max = 72, main = "PACF (level)")

# Quick preview of first difference and seasonal difference (if m>1)
if (m > 1) {
  plot(diff(y_ts, lag = m), main = expression("Preview: seasonal difference " * nabla[m] * " y_t"), ylab = "", xlab = "")
} else {
  plot(diff(y_ts), main = expression("Preview: first difference " * nabla * " y_t"), ylab = "", xlab = "")
}

```

Packages for Identification (tests, plots)-1.png)

```

par(mfrow = c(1,1))

#####Stationarity testting

cat("\n--- Stationarity tests on LEVEL ---\n")

##
## --- Stationarity tests on LEVEL ---

# KPSS (H0: stationarity). Small p-value -> evidence AGAINST stationarity (suggest differencing)
print(kpss.test(y_ts, null = "Level"))

##
## KPSS Test for Level Stationarity
##
## data: y_ts
## KPSS Level = 0.5119, Truncation lag parameter = 5, p-value = 0.03899

# ADF test using R, H0: unit root (nonstationary)
print(adf.test(y_ts))

##
## Augmented Dickey-Fuller Test
##
## data: y_ts
## Dickey-Fuller = -2.7191, Lag order = 7, p-value = 0.2734
## alternative hypothesis: stationary

# NOTE:
# Combine visual + tests:
# • If KPSS rejects stationarity and ADF fails to reject unit root -> i

```

```

nclude difference.
# • Prefer the SMALLEST differencing that yields stationarity to avoid
  overdifferencing.

# =====
# 2) AUTOCORRELATION & PARTIAL AUTOCORRELATION ANALYSIS (LEVEL)
#   Use ACF/PACF to sense AR vs MA structure, but final read is on the
#   stationarized (differenced) series in Section 4.
# =====
par(mfrow = c(1,2))
Acf(y_ts, lag.max = 72, main = "ACF (level): AR/MA cues") #repeat PACF
and ACF
Pacf(y_ts, lag.max = 72, main = "PACF (level): AR/MA cues")

```

Packages for Identification (tests, plots)-2.png

```

par(mfrow = c(1,1))

# NOTE:
# • AR(p): PACF tends to “cut off” near p; ACF tails off.
# • MA(q): ACF tends to “cut off” near q; PACF tails off.
# • Mixed ARMA: both ACF & PACF often tail off.

# =====
# 3) SEASONALITY CHECK
#   Visual inspection + seasonal decomposition (STL)
# =====

# STL decomposition (This R function is more robust than the decompose()
# function,
#but it only allows for additive decomposition. Additive is reasonable
# for rates)
stl_fit <- try(stl(y_ts, s.window = "periodic"), silent = TRUE)
if (!inherits(stl_fit, "try-error")) plot(stl_fit, main = "STL decompos
ition (Trend/Seasonal/Remainder)")

```

Packages for Identification (tests, plots)-3.png

```

if (m > 1) {
  # Month/season means pattern also visible via monthplot (base)
  # If this plot looks like a unit root for each month, include a seaso
  nal difference D=1
  suppressWarnings(monthplot(y_ts, main = "Monthplot: within-year patte
  rn", ylab = "Percent"))
}

```

Packages for Identification (tests, plots)-4.png

```

# NOTE:
# Clear repeating within-year pattern ⇒ include seasonal terms in ARIM
A:
# ARIMA(p,d,q)(P,D,Q)[m].
# For NSA unemployment rate, D often = 1; for SA rate, D often = 0.

# =====
# 4) DIFFERENCING ORDER SELECTION
# Choose seasonal D first, then nonseasonal d. Build working series
w_t.
# =====
D_hat <- 0 #Try your option here based on seasonal plots above
y_seas <- if (D_hat > 0) diff(y_ts, lag = m, differences = D_hat) else
y_ts

d_hat <- 1 #Try your option here based on stationarity tests above
w_ts <- if (d_hat > 0) diff(y_seas, differences = d_hat) else y_seas
cat(sprintf("\nTrial differencing orders: D = %d (seasonal), d = %d (no
nseasonal)\n", D_hat, d_hat))

##
## Trial differencing orders: D = 0 (seasonal), d = 1 (nonseasonal)

# Re-check stationarity on working series
cat("\n--- Stationarity tests on WORKING SERIES w_t ---\n")

##
## --- Stationarity tests on WORKING SERIES w_t ---

print(kpss.test(w_ts, null = "Level"))

## Warning in kpss.test(w_ts, null = "Level"): p-value greater than pri
nted
## p-value

##
## KPSS Test for Level Stationarity
##
## data: w_ts
## KPSS Level = 0.031539, Truncation lag parameter = 5, p-value = 0.1

print(adf.test(w_ts))

## Warning in adf.test(w_ts): p-value smaller than printed p-value

##
## Augmented Dickey-Fuller Test
##
## data: w_ts
## Dickey-Fuller = -7.6778, Lag order = 7, p-value = 0.01
## alternative hypothesis: stationary

```

```

# ACF/PACF of w_t to guide p, q (and seasonal P, Q via spikes at multiples of m)
par(mfrow = c(1,2))
Acf(w_ts, lag.max = 72, main = expression(paste("ACF of working series", w[t])))
Pacf(w_ts, lag.max = 72, main = expression(paste("PACF of working series", w[t])))

```

Packages for Identification (tests, plots)-5.png

```

par(mfrow = c(1,1))

# Simple candidate ranges to carry into estimation step. Choose based on above ACF/PACF
p_candidates <- 0:3; q_candidates <- 0 # Set max value based on ACF/PACF
P_candidates <- 0 # Max is one if Seasonality present in ACF/PACF, zero otherwise
Q_candidates <- 0 # Max is one if Seasonality present in ACF/PACF, zero otherwise

cat("\n--- Identification summary ---\n")

##
## --- Identification summary ---

cat(sprintf("Period m = %d | D = %d | d = %d\n", m, D_hat, d_hat))

## Period m = 12 | D = 0 | d = 1

cat("Use ACF/PACF of w_t to bound:\n")

## Use ACF/PACF of w_t to bound:

cat("  Nonseasonal: p in {0,1,2,3}, q in {0,1,2,3}\n")

##  Nonseasonal: p in {0,1,2,3}, q in {0,1,2,3}

if (m > 1) cat("  Seasonal:    P in {0,1,2},    Q in {0,1,2}\n")

##  Seasonal:    P in {0,1,2},    Q in {0,1,2}

cat("Proceed to ESTIMATION with a small grid using these bounds.\n")

## Proceed to ESTIMATION with a small grid using these bounds.

# Object to pass forward
identification <- list(
  frequency = m,
  d = d_hat, D = D_hat,
  w_ts = w_ts, y_seas = y_seas,
  p_candidates = p_candidates,
  q_candidates = q_candidates,

```

```

P_candidates = P_candidates,
Q_candidates = Q_candidates
)
# Inspect structure
str(identification)

## List of 9
## $ frequency      : num 12
## $ d              : num 1
## $ D              : num 0
## $ w_ts           : Time-Series [1:359] from 1996 to 2025: 0 0 0 0 0 ...
## $ y_seas         : Time-Series [1:360] from 1996 to 2025: 6 6 6 6 6 6
5.9 5.9 5.8 5.7 ...
## $ p_candidates: int [1:4] 0 1 2 3
## $ q_candidates: num 0
## $ P_candidates: num 0
## $ Q_candidates: num 0

# =====
# BOX-JENKINS: STEP 2 – ESTIMATION
# Purpose: Fit candidate ARIMA models using orders suggested by Identif
ication,
#         compare by information criteria (AICc/BIC), and select a fin
al model.
# Inputs expected:
#   - y_ts : original monthly series
#   - identification list from Step 1 with:
#     d, D, frequency=m, w_ts, y_seas,
#     p_candidates, q_candidates, P_candidates, Q_candidates
# =====
#####Initial Code to Run

needed <- c("forecast")
to_install <- setdiff(needed, rownames(installed.packages()))
if (length(to_install)) install.packages(to_install, repos = "https://c
loud.r-project.org")
library(forecast)

# Safe AICc getter (avoids "could not find function 'AICc'" errors)
get_AICc <- function(fit) {
  if ("AICc" %in% ls("package:forecast")) return(forecast::AICc(fit))
  if (exists("AICc")) return(AICc(fit))
  # Fallback formula: AICc = AIC + 2k(k+1)/(n - k - 1), k = df of LogLi
k (incl. sigma^2)
  k <- attr(logLik(fit), "df")
  n <- fit$nobs
  AIC(fit) + (2 * k * (k + 1)) / (n - k - 1)
}

#Define coefficient table function

```

```

coef_table <- function(fit) {
  se <- sqrt(diag(fit$var.coef))
  est <- fit$coef
  tval <- est / se
  pval <- 2 * pt(-abs(tval), df = length(fit$residuals) - length(est))
  data.frame(Estimate = est, Std.Error = se, t = tval, `Pr(>|t|)` = pva
1, check.names = FALSE)
}

m <- frequency(y_ts)
if (!exists("identification")) {
  # Fallback: recompute minimal pieces if Step 1 object is missing
  D_hat <- if (m > 1) forecast::nsdiffs(y_ts, m = m) else 0
  y_seas <- if (D_hat > 0) diff(y_ts, lag = m, differences = D_hat) else
e y_ts
  d_hat <- forecast::ndiffs(y_seas)
  identification <- list(
    frequency = m, d = d_hat, D = D_hat,
    w_ts = if (d_hat > 0) diff(y_seas, differences = d_hat) else y_seas,
    y_seas = y_seas,
    p_candidates = 0:3, q_candidates = 0:3,
    P_candidates = if (m > 1) 0:2 else 0:0,
    Q_candidates = if (m > 1) 0:2 else 0:0
  )
}

d_hat <- identification$d
D_hat <- identification$D
p_grid <- identification$p_candidates
q_grid <- identification$q_candidates
P_grid <- identification$P_candidates
Q_grid <- identification$Q_candidates

# Choose information criterion for selection:
criterion <- "AICc" # options: "AICc" or "BIC"

# Include mean / drift rules:
include_mean <- (d_hat + D_hat) == 0
include_drift <- (d_hat + D_hat) > 0 # allow deterministic drift when
differenced

# =====
# (1) MODEL SELECTION
# Build a small, interpretable grid over (p,q,P,Q), fit each candid
ate,
# and rank by chosen information criterion (AICc default).
# =====
cand_fits <- list()

```

```

scoreboard <- data.frame()

### This function estimates the values of the coefficients for all values in the grid
for (p in p_grid) for (q in q_grid) {
  for (P in P_grid) for (Q in Q_grid) {
    # Skip the totally empty model unless a mean/drift is present
    if ((p + q + P + Q) == 0 && !(include_mean || include_drift)) next
    fit <- try(
      Arima(y_ts,
        order = c(p, d_hat, q),
        seasonal = if (m > 1) list(order = c(P, D_hat, Q), period =
m) else NULL,
        include.mean = include_mean,
        include.drift = include_drift,
        method = "ML"),
      silent = TRUE
    )
    if (inherits(fit, "try-error")) next

    # Scores
    aicc <- get_AICc(fit)
    bic <- BIC(fit)

    cand_fits[[length(cand_fits) + 1]] <- fit
    scoreboard <- rbind(scoreboard, data.frame(
      idx = length(cand_fits),
      p = p, d = d_hat, q = q, P = P, D = D_hat, Q = Q, m = m,
      AICc = aicc, BIC = bic,
      stringsAsFactors = FALSE
    ))
  }
}

stopifnot(nrow(scoreboard) > 0)

# Rank by the default criterion (AICc unless you changed `criterion`) and set label
ranking_table <- scoreboard[order(scoreboard[[criterion]]), ]
best_row <- ranking_table[1, , drop = FALSE]
final_fit <- cand_fits[[best_row$idx]]
selection_label <- criterion # will be updated if an override is used

cat("\n=== MODEL SELECTION (top 10 by ", selection_label, ") ===\n", sep = "")

##
## === MODEL SELECTION (top 10 by AICc) ===

```

```

print(head(ranking_table, 10))

##   idx p d q P D Q m   AICc   BIC
## 3   3 2 1 0 0 0 12 695.2163 710.6366
## 2   2 1 1 0 0 0 12 695.3782 706.9605
## 4   4 3 1 0 0 0 12 696.5329 715.7795
## 1   1 0 1 0 0 0 12 701.9637 709.6967

cat("\nChosen model (", selection_label, "):\n", sep = "")

##
## Chosen model (AICc):

print(best_row)

##   idx p d q P D Q m   AICc   BIC
## 3   3 2 1 0 0 0 12 695.2163 710.6366

# NOTE:
# We deliberately use a SMALL grid bound by Identification to avoid overfitting.
# Selection principle: minimize information criterion (AICc default) to balance fit vs complexity.

# =====
# (2) PARAMETER ESTIMATION
#   Once the model is specified, estimate AR/MA (and seasonal) parameters
#   via Maximum Likelihood (using the Arima() function). Report estimates & standard errors.
# =====
cat("\n=== PARAMETER ESTIMATION: Coefficient table (t-stats & p-values)
    ==\n")

##
## == PARAMETER ESTIMATION: Coefficient table (t-stats & p-values) ==

print(round(coef_table(final_fit), 4))

##      Estimate Std.Error      t Pr(>|t|)
## ar1      -0.1659      0.0526 -3.1562  0.0017
## ar2      -0.0781      0.0525 -1.4880  0.1376
## drift    -0.0056      0.0267 -0.2106  0.8333

cat("\nModel summary (including logLik, AIC/AICc/BIC):\n")

##
## Model summary (including logLik, AIC/AICc/BIC):

print(final_fit)

## Series: y_ts
## ARIMA(2,1,0) with drift

```

```

##
## Coefficients:
##          ar1          ar2          drift
##      -0.1659  -0.0781  -0.0056
## s.e.    0.0526   0.0525   0.0267
##
## sigma^2 = 0.4003: log likelihood = -343.55
## AIC=695.1   AICc=695.22   BIC=710.64

cat("\nInformation criteria for the chosen model:\n")

##
## Information criteria for the chosen model:

cat(sprintf("AIC = %.3f | AICc = %.3f | BIC = %.3f\n", AIC(final_fit),
  get_AICc(final_fit), BIC(final_fit)))

## AIC = 695.103 | AICc = 695.216 | BIC = 710.637

# NOTE:
# ML finds  $\vartheta = (\phi, \vartheta, \Phi, \theta, \text{drift/mean}, \sigma^2)$  maximizing Log-Likelihood.
# Standard errors come from the observed information matrix (inverse Hessian).

# =====
# (3) MODEL FITTING
# Use the estimated model to obtain fitted values; assess in-sample
# fit
# (plots + simple accuracy metrics).
# =====
# One-step-ahead fitted values (in-sample)
fitted_vals <- fitted(final_fit)
resids <- residuals(final_fit)

# Accuracy metrics (in-sample)
rmse <- sqrt(mean(resids^2, na.rm = TRUE))
mae <- mean(abs(resids), na.rm = TRUE)
mape <- mean(abs(resids / y_ts), na.rm = TRUE) * 100

cat("\n=== MODEL FITTING: In-sample accuracy ===\n")

##
## === MODEL FITTING: In-sample accuracy ===

cat(sprintf("RMSE = %.4f | MAE = %.4f | MAPE = %.2f%%\n", rmse, mae, mape))

## RMSE = 0.6291 | MAE = 0.1257 | MAPE = 1.75%

# Plot actual vs fitted (one-step ahead estimate)
op <- par(no.readonly = TRUE)

```

```

par(mfrow = c(2,1))
plot(y_ts, main = "Actual vs Fitted (one-step ahead)", ylab = "Percent",
     xlab = "")
lines(fitted_vals, col = "blue")
legend("topleft", lty = c(1,1), col = c("black", "blue"), bty = "n", c
      ("Actual", "Fitted"))

plot(resids, main = "Residuals from chosen model", ylab = "", xlab = "")
abline(h = 0, lty = 2)

```

Pull Identification outputs (with safe fallbacks)-1.png

```

par(op)

# NOTE:
# "Model fitting" here refers to applying the estimated parameters to p
# roduce
# fitted values and inspecting basic fit metrics. Formal residual diagn
# ostics
# (residual analysis, Ljung-Box test) belong to Step 3 (Diagnostics).

#=====
# BOX-JENKINS: STEP 3 – DIAGNOSTICS
# Purpose: Validate that residuals from the chosen ARIMA are white nois
# e:
#         no autocorrelation, mean  $\sim 0$ , roughly constant variance.
# Inputs expected:
#   - final_fit : the chosen Arima() model from Step 2
#   - y_ts      : original series (for plots/scale)
# =====

needed <- c("forecast", "tseries")
to_install <- setdiff(needed, rownames(installed.packages()))
if (length(to_install)) install.packages(to_install, repos = "https://c
loud.r-project.org")
library(forecast)
library(tseries)

# Safe access to residuals/fitted
stopifnot(exists("final_fit"))
resids <- residuals(final_fit)
fits   <- fitted(final_fit)

# helper: count ARMA-type parameters for Ljung-Box df (exclude mean/dri
# ft/sigma)
arma_df <- {
  nm <- names(coef(final_fit))
  if (is.null(nm)) 0L else sum(grepl("ar|ma", nm, ignore.case = TRUE) &
    !grepl("mean|drift|intercept|sigma", n

```

```

m, ignore.case = TRUE))
}

# =====
# (1) RESIDUAL ANALYSIS
#   Visual inspection: no pattern over time, roughly constant variance,
#   residual ACF/PACF  $\sim 0$ 
# =====
op <- par(no.readonly = TRUE)
par(mfrow = c(2,2))
plot(resids, main = "Residuals over time", ylab = "", xlab = "")
abline(h = 0, lty = 2)

hist(resids, breaks = "FD", main = "Histogram of residuals", xlab = "Residual")
lines(density(na.omit(resids)), lwd = 2)

Acf(resids, lag.max = 72, main = "Residual ACF")
Pacf(resids, lag.max = 72, main = "Residual PACF")

```

Packages & helpers-1.png)

```

par(op)

# NOTE:
# • White-noise residuals show no visible trend/seasonal pattern; ACF/PACF spikes lie within bands.

# =====
# (2) LJUNG-BOX TEST
#   H0: no autocorrelation up to lag L (residuals are white noise).
#   We test multiple horizons (e.g., 12, 24, 36) and adjust df by ARM A params.
# =====
LB_lags <- c(12, 24, 36)
cat("\n=== Ljung-Box tests (H0: no residual autocorrelation) ===\n")

##
## === Ljung-Box tests (H0: no residual autocorrelation) ===

for (L in LB_lags) {
  lb <- Box.test(resids, lag = L, type = "Ljung-Box", fitdf = arma_df)
  cat(sprintf("Lag %2d: X^2=%.3f, df=%d, p=%.4f\n", L, lb$statistic, lb$parameter, lb$p.value))
}

## Lag 12: X^2=3.681, df=10, p=0.9606
## Lag 24: X^2=4.958, df=22, p=0.9999
## Lag 36: X^2=5.365, df=34, p=1.0000

```

```
cat("\n--- Combined check (forecast::checkresiduals) ---\n")
```

```
##
```

```
## --- Combined check (forecast::checkresiduals) ---
```

```
# Produces residual plot + ACF + LB test in one view
```

```
suppressWarnings(checkresiduals(final_fit))
```

Packages & helpers-2.png)

```
##
```

```
## Ljung-Box test
```

```
##
```

```
## data: Residuals from ARIMA(2,1,0) with drift
```

```
## Q* = 4.9579, df = 22, p-value = 0.9999
```

```
##
```

```
## Model df: 2. Total lags used: 24
```

```
# NOTE:
```

```
# • Prefer p-values > 0.05 (no evidence of residual autocorrelation).
```

```
# • If p < 0.05 at seasonal multiples (e.g., 12, 24), consider seasonal  
AR/MA adjustments.
```

```
#####
```

```
# ARIMA Forecasting in R (Post Box-Jenkins)
```

```
# Series: PAUR (SA monthly PA unemployment rate)
```

```
# Window: 1995-07 to 2025-06
```

```
#####
```

```
needed <- c("quantmod", "forecast", "lubridate", "zoo")
```

```
to_install <- setdiff(needed, rownames(installed.packages()))
```

```
if (length(to_install)) install.packages(to_install, repos = "https://cloud.r-project.org")
```

```
library(quantmod) # getSymbols() from FRED
```

```
library(forecast) # Arima(), forecast(), autoplot(), accuracy()
```

```
library(lubridate) # date handling
```

```
library(zoo) # as.yearmon(), plotting helpers
```

```
getSymbols("PAUR", src = "FRED", auto.assign = TRUE, warnings = FALSE)
```

```
## [1] "PAUR"
```

```
start_date <- ymd("1995-07-01")
```

```
end_date <- ymd("2025-06-30")
```

```
# xts subset by date
```

```
un_sa_xts <- window(PAUR, start = start_date, end = end_date)
```

```
# Convert to a monthly 'ts' object with frequency m = 12
```



```

## Nov 2025      3.943249 2.4377054 5.448793 1.64071865 6.245780
## Dec 2025      3.937512 2.3074705 5.567553 1.44457879 6.430444
## Jan 2026      3.931786 2.1863082 5.677263 1.26230832 6.601263
## Feb 2026      3.926152 2.0726170 5.779686 1.09141514 6.760888
## Mar 2026      3.920509 1.9648498 5.876169 0.92958633 6.911432
## Apr 2026      3.914860 1.8621313 5.967588 0.77548243 7.054237
## May 2026      3.909208 1.7637952 6.054621 0.62808221 7.190334
## Jun 2026      3.903558 1.6693052 6.137811 0.48656321 7.320553
## Jul 2026      3.897908 1.5782157 6.217600 0.35024490 7.445571
## Aug 2026      3.892258 1.4901631 6.294353 0.21857097 7.565945
## Sep 2026      3.886608 1.4048448 6.368371 0.09107886 7.682137
## Oct 2026      3.880958 1.3220056 6.439910 -0.03262172 7.794537
## Nov 2026      3.875307 1.2414276 6.509187 -0.15286414 7.903479
## Dec 2026      3.869657 1.1629230 6.576392 -0.26993563 8.009250
## Jan 2027      3.864007 1.0863286 6.641686 -0.38408571 8.112100
## Feb 2027      3.858357 1.0115015 6.705213 -0.49553278 8.212247
## Mar 2027      3.852707 0.9383161 6.767098 -0.60446932 8.309883
## Apr 2027      3.847057 0.8666605 6.827453 -0.71106601 8.405180
## May 2027      3.841407 0.7964354 6.886378 -0.81547501 8.498289
## Jun 2027      3.835757 0.7275517 6.943962 -0.91783268 8.589346

# Convert the ts index of fc$mean into "yearmon" and then a YYYY-MM Date
f_dates <- as.yearmon(time(fc$mean))           # "2025 Jul", "2025 Aug", ...
f_dates <- as.Date(f_dates)                   # becomes first day of each month
forecast_tbl <- data.frame(
  date = f_dates,
  mean = as.numeric(fc$mean),
  lo80 = as.numeric(fc$lower[, "80%"]),
  hi80 = as.numeric(fc$upper[, "80%"]),
  lo95 = as.numeric(fc$lower[, "95%"]),
  hi95 = as.numeric(fc$upper[, "95%"])
)
head(forecast_tbl, 6)

##      date      mean    lo80    hi80    lo95    hi95
## 1 2025-07-01 3.971186 3.160104 4.782268 2.730744 5.211629
## 2 2025-08-01 3.960160 2.905819 5.014501 2.347685 5.572635
## 3 2025-09-01 3.952628 2.725695 5.179561 2.076196 5.829060
## 4 2025-10-01 3.948805 2.578419 5.319190 1.852981 6.044628
## 5 2025-11-01 3.943249 2.437705 5.448793 1.640719 6.245780
## 6 2025-12-01 3.937512 2.307470 5.567553 1.444579 6.430444

# You can write to CSV if desired:
# write.csv(forecast_tbl, "forecast_PAUR_ARIMA.csv", row.names = FALSE)

autoplot(fc) +
  ggplot2::labs(title = "ARIMA Forecast for Pennsylvania Unemployment (SA)",

```

```

        subtitle = sprintf("Model: (%d,%d,%d)(%d,%d,%d)[%d], h
= %d months",
                                p,d,q,P,D,Q,m,H),
        x = "", y = "Percent")

```

Plot: full sample + forecast and Pls-1.png)

```

N_years <- 10 # show last 10 years of history for context
start_zoom <- c(end(time(y_ts)) - (N_years - 1), 1) # rough; or use win
dow()

autoplot(window(y_ts, start = time(y_ts)[length(y_ts) - N_years*m + 1]))
+
  autolayer(fc$mean, series = "Forecast") +
  autolayer(fc$lower[, "95%"], series = "95% PI (lower)") +
  autolayer(fc$upper[, "95%"], series = "95% PI (upper)") +
  ggplot2::labs(title = "Zoomed: Recent History and ARIMA Forecast (SA)
",
                x = "", y = "Percent")

```

Zoomed plot: last N years + forecast-1.png)

```

# If you want to *verify* forecast skill out-of-sample, reserve the las
t k months
# as a test set (here, k = 18 for example) and compare forecasts to act
uals.

```

```

k <- 18
y_train <- head(y_ts, length(y_ts) - k)
y_test <- tail(y_ts, k)

fit_bt <- Arima(
  y = y_train,
  order = c(p, d, q),
  seasonal = list(order = c(P, D, Q), period = m),
  include.mean = include_mean,
  include.drift = include_drift,
  method = "ML"
)

fc_bt <- forecast(fit_bt, h = k)
print(accuracy(fc_bt, y_test)) # MAE, RMSE, MAPE, etc.

```

```

##              ME      RMSE      MAE      MPE      MAPE
## MASE
## Training set 2.227008e-05 0.6453432 0.1302104 -0.2420419 1.787542 0.
1551804
## Test set    8.258384e-02 0.1736688 0.1327195  2.0940015 3.486659 0.
1581706
##              ACF1 Theil's U

```

```
## Training set -0.004061638      NA
## Test set      0.790081851  3.216152

autoplot(fc_bt) +
  autolayer(y_test, series = "Actual (Holdout)") +
  ggplot2::labs(title = "Holdout Backtest: ARIMA Forecast vs Actual",
                x = "", y = "Percent")
```

(Optional) Backtesting snippet (rolling-origin or holdout)-1.png

```
#####
# End of script
#####
```