STAT 443 Report

Chow Sheng Liang

2025-07-26

Part 1

A fundamental part of forecasting is understanding how a variable relates to its own past, this is because most real-world processes are not memoryless. For example, daily temperature tends to evolve smoothly over time, electricity demand shows strong daily and weekly cycles, and stock market volatility clusters over consecutive days.

Stationarity is a core concept that provides the theoretical framework to model such temporal relationships. Formally we define a time series as :

$$\{X_t\}_{t\in\mathbb{Z}} \quad \text{or} \quad \{X_1, X_2, \ldots, X_T\}$$

where each X_t is a random variable representing the value of a process at time t. The model we are trying to approximate here would differ slightly from linear regression:

$$\hat{f}(\{X_1, \cdots, X_t\}) \approx f(\{X_1, \cdots, X_t\}) = X_{t+1}$$

Stationarity is satisfied by the following 3 conditions:

1. $E(|X_t|^2) < \infty \quad$ for all $t \in T$
2. $E(X_t) = \mu \quad$ for all $t \in T$
3. $Cov(X_r, X_s) = Cov(X_{r+t}, X_{s+t}) \quad$ for all $\; r, s, r+t, s+t \in T$

These conditions combine to ensure that the statistical properties of a process remain **consistent over time**, allowing us to estimate future values based on past patterns without worrying about structural drift. *(Theoretically, stationarity by itself is not enough, we would also need causality that ensures a variable is **only influenced by its past**)*

Just as linear regression builds on linearity, time series analysis builds on stationarity. Under this assumption, we can fit structured models like AR, MA, or $ARMA$, which linearly relate a variable to its past values, much like how linear models relate outcomes to transformed inputs.

In practice, as you may have noticed with linear regression, we often do not observe data that adhere to the linearity/normality assumption and we typically transform y or X. In the same spirit, we transform a time series with differencing and lag transformations:

- Backward operator: $BX_t = X_{t-1}$
- d^{th} order differencing: $Y_t := \nabla^d X_t = (1-B)^d X_t$
- Seasonal differencing: $\nabla^s X_t := (1-B^s)X_t$

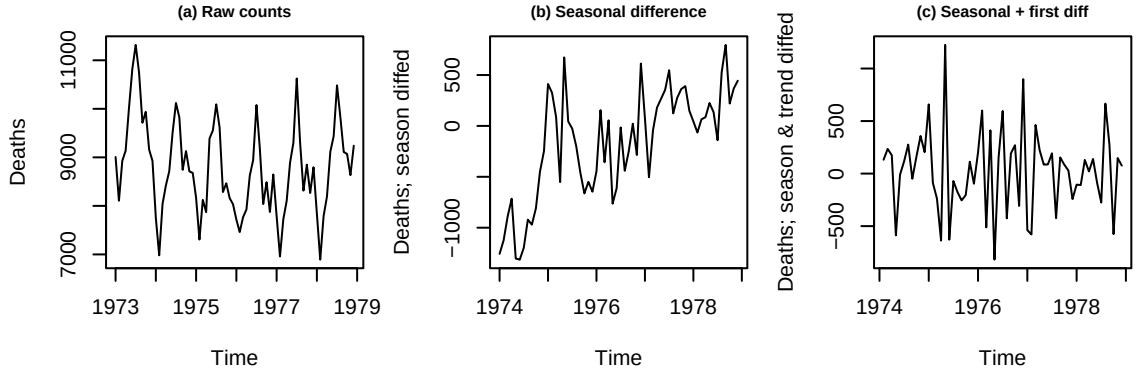

Figure 1: Transformation of US accidental-death counts

So in summary, just as linearity is crucial for linear regression, stationarity is a core assumption in forecasting as it restricts the function space so we can meaningfully fit and extrapolate a process' temporal relationships independent of external changes.

Part 2

```r
project_data <- readRDS("projectdata.rds")
pred_num_ahead <- 100
conf_int_mult <- 1.96

# Model fitting
ar1_model <- Arima(project_data, order = c(1, 0, 0))
ar1_prediction <- predict(ar1_model, n.ahead = pred_num_ahead)

arima_model <- Arima(project_data, order = c(0, 1, 0))
arima_prediction <- predict(arima_model, n.ahead = pred_num_ahead)

# Get prediction time range
freq <- frequency(project_data)
last_time <- time(project_data)[length(project_data)]
future_time <- seq(from = last_time + 1/freq, by = 1/freq, length.out = pred_num_ahead)

# Construct vectors for CI
ar1_upper <- ar1_prediction$pred + conf_int_mult * ar1_prediction$se
ar1_lower <- ar1_prediction$pred - conf_int_mult * ar1_prediction$se

arima_upper <- arima_prediction$pred + conf_int_mult * arima_prediction$se
arima_lower <- arima_prediction$pred - conf_int_mult * arima_prediction$se

data_mean = mean(project_data)
data_var_pos = data_mean + conf_int_mult * sd(project_data)
data_var_neg = data_mean - conf_int_mult * sd(project_data)

# Layout
par(mfrow = c(1, 2), mar = c(4, 4, 2, 2))

# AR(1)
plot(project_data, xlim = c(start(project_data)[1], future_time[pred_num_ahead]),
     ylim = c(-20, 10), main = "(a)", xlab = "Time", ylab = "Value")
lines(future_time, ar1_prediction$pred, col = "red")
lines(future_time, ar1_upper, col = "blue")
lines(future_time, ar1_lower, col = "blue")
abline(h = data_mean, col = "red", lty = 2)
abline(h = data_var_pos, col = "blue", lty = 2)
abline(h = data_var_neg, col = "blue", lty = 2)

# ARIMA(0,1,0)
plot(project_data, xlim = c(start(project_data)[1], future_time[pred_num_ahead]),
     ylim = c(-20, 10), main = "(b)", xlab = "Time", ylab = "Value")
lines(future_time, arima_prediction$pred, col = "red")
lines(future_time, arima_upper, col = "blue")
lines(future_time, arima_lower, col = "blue")
abline(h = data_mean, col = "red", lty = 2)
abline(h = data_var_pos, col = "blue", lty = 2)
abline(h = data_var_neg, col = "blue", lty = 2)
```

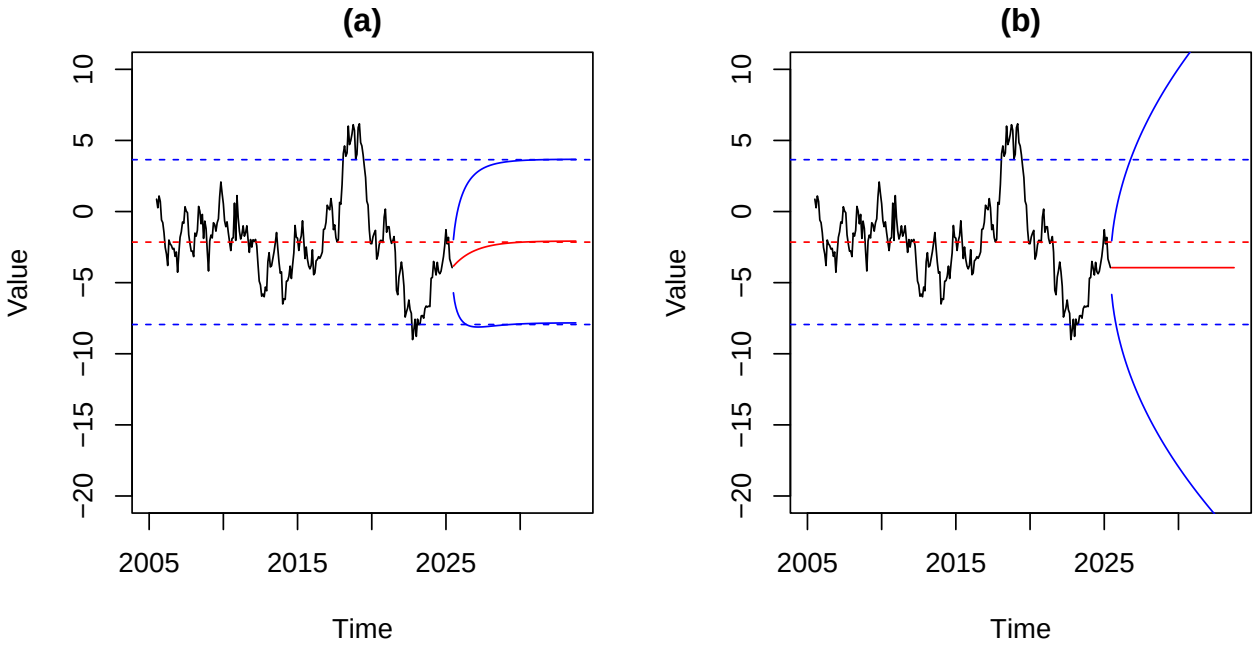

Figure 2: Panel (a) shows long term forecasts using an AR(1) model. Panel (b) shows long term forecasts using an ARIMA(0, 1, 0) model.

Through the Box-Jenkins methodology, we would first examine the time-series plot, ACF, and PACF plots to visually check for any reason to be suspicious of our stationary assumption. We can also numerically test for this using the Phillips-Perron test, where the null hypothesis is of a unit root → non-stationarity; which we hope to reject.

If we see a clear trend from the time-series plot or if the Phillips-Perron test suggests non-stationarity we may consider differencing it and re-evaluating. If we see a seasonal trend in the ACF(i.e. ACF spikes at regular periods) we may consider differencing with a lag that matches the period and re-evaluating.

Once the process is stationary, Box-Jenkins requires us to identify the model by fitting an appropriate order and validating with a criterion, then reiterating until the one with the best metric is found. However this entire process can be much easily done with auto.arima() through numerical evaluation. We can optionally provide auto.arima() with a ceiling for its search to optimize speed, auto.arima() will then automate the identification process by iteratively minimizing AIC across the possible (p,d,q)(P,D,Q) combinations with options to include/exclude the seasonal components as well as options to exhaustively search all permutations or do a stepwise search.

We will then visually validate this model by looking at it residual plot and residual ACF plot which we should expect to see evidence of a white noise process. If the residuals suggests otherwise, we should go back to step 1 and re-evaluate our model selection, and maybe even consider completely different models like ARCH, GARCH or simpler linear models like Holt-Winters.

The goal of this simulation study is to assess the performance of 2 tools used in time series modeling:

The first tool is the auto.arima() function, which automatically selects ARIMA models based on information criteria such as AIC. It tries combinations of autoreggressive (AR), differencing(I), and moving average (MA) components to minimize prediction error.

The second tool is the PP.test() function, which is a statistical test which tests for a unit root (non-stationary) against a stationary alternative.

We simulate data from a stationary AR(1) , which is a simple autoreggressive process where the current value depends on its immediate past value. In our case our model is:

$$X_t = 0.95X_{t-1} + \epsilon_t, \epsilon_t \sim \mathcal{N}(0, 0.5^2)$$

With $\phi = 0.95$, this model retains a strong memory of past values, meaning the series evolves slowly and appears to drift over short periods, even though it is theoretically stationary. We can show this model is stationary if it satisfies that we have a finite second moment, a constant mean, and we have a time-invariant auto-covariance function. By definition, and for h > 0:

$$\gamma(h) = Cov(X_t, X_{t-h}) = Cov(\phi X_{t-1} + Z_t, X_{t-h}) = \phi Cov(X_{t-1}, X_{t-h}) + Cov(Z_t, X_{t-h}) = \phi\gamma(h-1) + 0$$

and iterating this argument gives us an auto-covariance function $\gamma(h) = \phi^h\gamma(0)$. From this we can see if we have a finite second moment/variance if we evaluate $\gamma(0) = Cov(X_t, X_t) = Var(X_t)$ so:

$$\gamma(0) = Cov(\phi X_{t-1} + Z_t, \phi X_{t-1} + Z_t) = \phi^2\gamma(0) + \sigma^2$$

and rearranging gives us:

$$Var(X_t) = \mathbb{E}(|X_t|^2) = \gamma(0) = \frac{\sigma^2}{1-\phi^2}$$

So we have seen that it satisfies the second and third condition for finite second moment and a time-invariant auto-covariance function. Now we will show it has a constant mean:

$$\mathbb{E}(X_t) = \mathbb{E}(\phi X_{t-1}) + \mathbb{E}(Z_t) = \phi\,\mathbb{E}(X_{t-1}) + 0 = \phi^t\,\mathbb{E}(X_0)$$

and this would imply for every X_t the expected value is 0.

Although the AR(1) process with $\phi = 0.95$ is theoretically stationary, its values change slowly over time and can appear to drift or follow a trend in one direction over short intervals of time. This is because the autocorrelation decays very slowly, making the time series look similar to a non-stationary process.

This slow decay of correlation can mislead both visual inspection and statistical tests. For example, the Phillips-Perron test may not have enough evidence to correctly reject the null hypothesis of a unit root in these cases.

This gives motivation towards the logic and design of our experiment. We will compare the performance of the Phillips-Perron test across a high number of realizations of simulated data with a large length, and then because we know the AR(1) can appear non-stationary in shorter time intervals, we will test across another 1000 realizations of simulated data to emphasize the importance of dataset size in the result of applying these tools (PP test and auto arima).

```r
# Set seed for reproducibility
set.seed(443)

# Simulation function
simulate_experiment <- function(n_sim, series_length, phi = 0.95, sigma = 0.5) {
  pp_rejections <- logical(n_sim)
  arima_orders <- character(n_sim)

  for (i in 1:n_sim) {
    # Simulate AR(1) series
    e <- rnorm(series_length, mean = 0, sd = sigma)
    x <- numeric(series_length)
    x[1] <- rnorm(1, 0, sigma)
    for (t in 2:series_length) {
      x[t] <- phi * x[t - 1] + e[t]
    }

    # Phillips-Perron test
    pp_result <- suppressWarnings(pp.test(x))
    pp_rejections[i] <- pp_result$p.value < 0.05

    # auto.arima model
    fit <- auto.arima(x, ic = "aic", stationary = FALSE, seasonal = FALSE)
    ord <- arimaorder(fit)
    arima_orders[i] <- paste0("ARIMA(", ord[1], ",", ord[2], ",", ord[3], ")")
  }

  # Return results
  list(
    pp_reject_count = sum(pp_rejections),
    pp_reject_rate = mean(pp_rejections),
    arima_freq = sort(table(arima_orders), decreasing = TRUE)
  )
}

# Run experiments
results_long <- simulate_experiment(n_sim = 1000, series_length = 1000)
results_short <- simulate_experiment(n_sim = 1000, series_length = 100)

# Print results
cat("=== LONG SERIES (n = 1000) ===\n")
cat("PP Test Rejections:", results_long$pp_reject_count, "\n")
print(results_long$arima_freq)

cat("\n=== SHORT SERIES (n = 100) ===\n")
cat("PP Test Rejections:", results_short$pp_reject_count, "\n")
print(results_short$arima_freq)
```

To assess the performance of the PP-test and the auto arima function, we evaluated the performance of the functions in identifying stationarity and model structure from the known AR(1) process:

$$X_t = 0.95X_{t-1} + \epsilon_t, \epsilon_t \sim \mathcal{N}(0, 0.5^2)$$

As we have shown the process is stationary in theory but exhibits non-stationary properties in short intervals due to the high value of $\phi = 0.95$. We compared results for: A long series with 1000 realizations of length 1000 and a shorter length series with 1000 realizations of length 100.

The auto.arima() function was allowed to select from the full class of ARIMA models, including those that apply differencing. Although we know our simulated data is stationary, in real-world applications the true model is unknown. As such, we set stationary = FALSE to reflect a more realistic setting where the algorithm must determine whether differencing is needed.

The PP test examines the null hypothesis that the series has a unit root (i.e., is non-stationary). Therefore, rejecting the null supports the conclusion that the series is stationary.

In the long series (length = 1000), the PP test correctly concluded the data was stationary in all 1000 simulations.

In the shorter series (length = 100), the PP test only correctly rejected the null hypothesis in only 119 out of 1000 simulations. In the remaining 881 cases, the test failed to detect stationarity, indicating that short series may mislead formal statistical tests, such as the PP test in this case.

The auto.arima() funciton selects ARIMA models based on AIC. Ideally it should frequently select the ARIMA(1,0,0) model which is the true model.

In the long series, the most frequent model selections were ARIMA(0,1,0): 369 times, ARIMA(1,0,0) 243 times, and the third most frequent ARIMA(0, 1, 1) only 40 times.

In the shorter series the ARIMA(0, 1, 0) model was selected 676 times and the ARIMA(1,0,0) was only selected 158 times

These results show that both the Phillips-Perron test can struggle with processes with high autocorrelation ($\phi \approx 1$), especially when sample sizes are small. We see this as the shorter length simulations only had the PP test correctly reject the hypothesis/ detect stationarity around 12% of the time. These results conclude that the PP-test has low power to detect stationarity in short samples, particularly when autocorrelation is high, leading to many false conclusions of non-stationarity within time series.

The auto.arima() tends to favour differenced or unnecessarily complex models when the process has a high autocorrelation (phi close to 1, in this case 0.95) even in the samples with longer length as the most selected model was the ARIMA(0,1,0). This shows model selection via AIC can be unstable in small samples, often choosing unnecessarily complex ARIMA models that do not reflect the true generating process.

Overall, the results demonstrate the importance of dataset length and the limitations of relying solely on automated statistical tests. When working with time series that exhibit strong autocorrelation, analysts must be cautious as short-term behavior may mimic non-stationarity and lead to incorrect model identification or hypothesis test results. Visual inspection, diagnostic tools, and theoretical understanding should be used together to avoid these misunderstandings.