Analysis and Forecast of Polish Inflation Rate

Erste Group CEE Research Interview Task

Aleksandar Gradev

28/05/2025

Contents

Task Overview

The task is to analyze the historical dynamics of Polish inflation and forecast the Harmonized Index of Consumer Prices (HICP) for the first four months of 2025. We will try to understand inflation patterns and build a model that can provide reliable estimates.

My approach begins with a detailed diagnosis of the time series, examining key properties such as stationarity, trend, seasonality and autocorrelation. I then build and compare three forecasting models: a basic AR(1) model, an auto-selected univariate ARIMA model and a multivariate ARIMAX that includes macroeconomic variables such as unemployment, oil prices, exchange rates, wages and bond yields.

The objective is to create a forecast that is not only statistically sound but also economically meaningful.

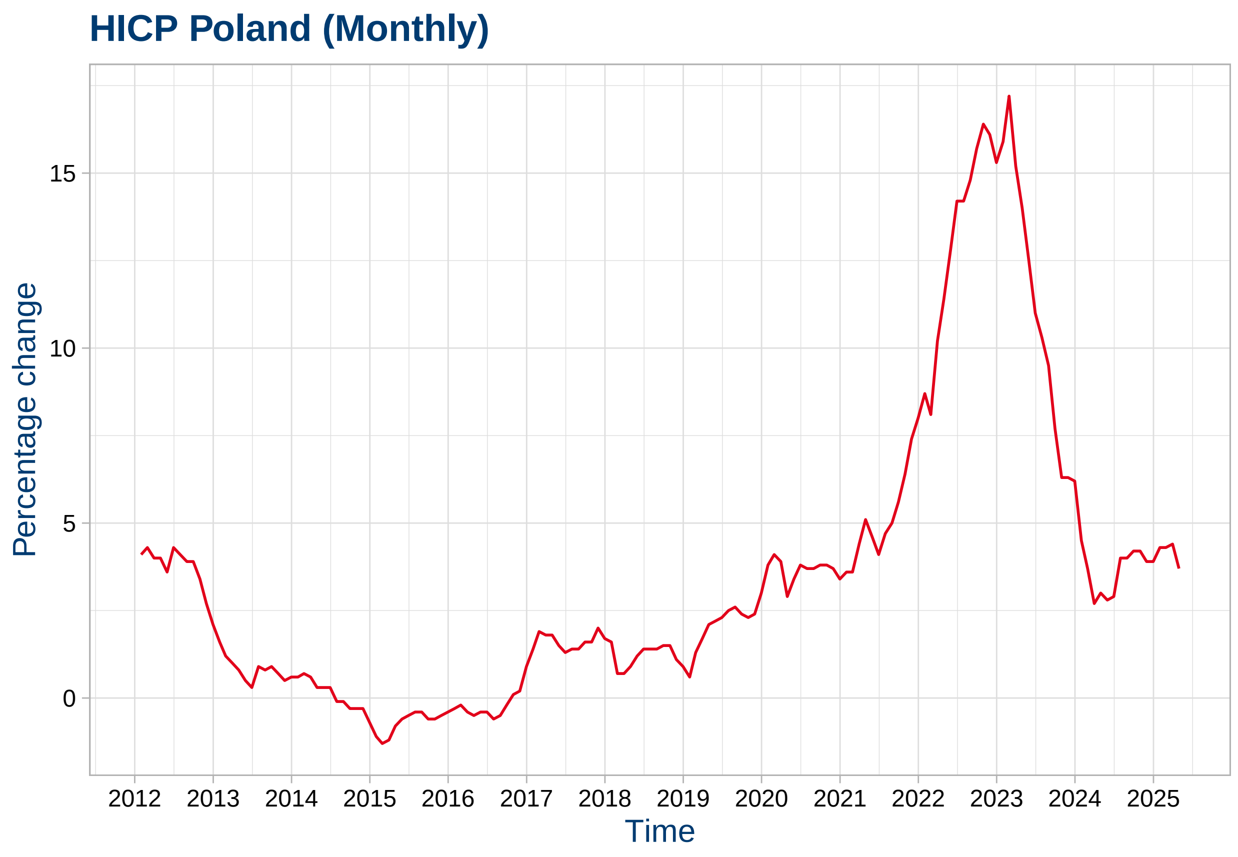
Loading and Pre-processing of Data

I start by loading the data. This data set was downloaded on the 21st of May 2025 from the Eurostat's webpage and can be accessed here. This data set includes monthly information for Percentage Change in Poland's Overall HICP (Harmonized Index of Consumer Prices). In other word, the data represent polish year-on-year inflation rate. It spans from January 1997 to April 2025 and captures its inflation before and after joining the European Union.

```r
# Load libraries
library(lubridate)
library(ggplot2)
library(tseries)
library(forecast)
library(quantmod)
library(dplyr)
library(tidyr)

# Load data
data <- read.csv("ECB Data Portal_20250521164956.csv")
```

In order to work with the data more eficiently, I will store it in a separate data frame. Then I will create a subset with data between Jan 2012 and Apr 2025 and plot it.

```r
# Create a df to work with
data_df <- data.frame(
  hicp_index = data$HICP...Overall.index..ICP.M.PL.N.000000.4.ANR.,
  date = as.Date(data$DATE)
)

poland_df <- subset(data_df, subset = data_df$date > as.Date("2012-01-01") &
                      data_df$date <  as.Date("2025-05-01"))

ggplot(poland_df, aes(x = date, y = hicp_index)) +
  geom_line(color = "#E2001A") +
  labs(
    title = "HICP Poland (Monthly)",
    x = "Time",
    y = "Percentage change"
  ) +
  theme_light() +
  scale_x_date(
    date_breaks = "1 year",
    date_labels = "%Y"
  ) +
  theme(
    plot.title = element_text(color = "#003B71", size = 14, face = "bold"),
    axis.title.x = element_text(color = "#003B71", size = 12),
    axis.title.y = element_text(color = "#003B71", size = 12),
    axis.text = element_text(color = "black")
  )
```



Time Series Diagnosis

A time series has 3 characteristics - stationarity, trend & seasonality and autocorrelation. To diagnose the time series we need to look at each of those characteristics and understand them.

Data Split

First, we need to split the data set into 2 parts - training data and testing data. The training data set will include data until the end of 2024 and the rest will be part of the testing data set. I will create both a data frame and time series objects for the 2 parts of our data.

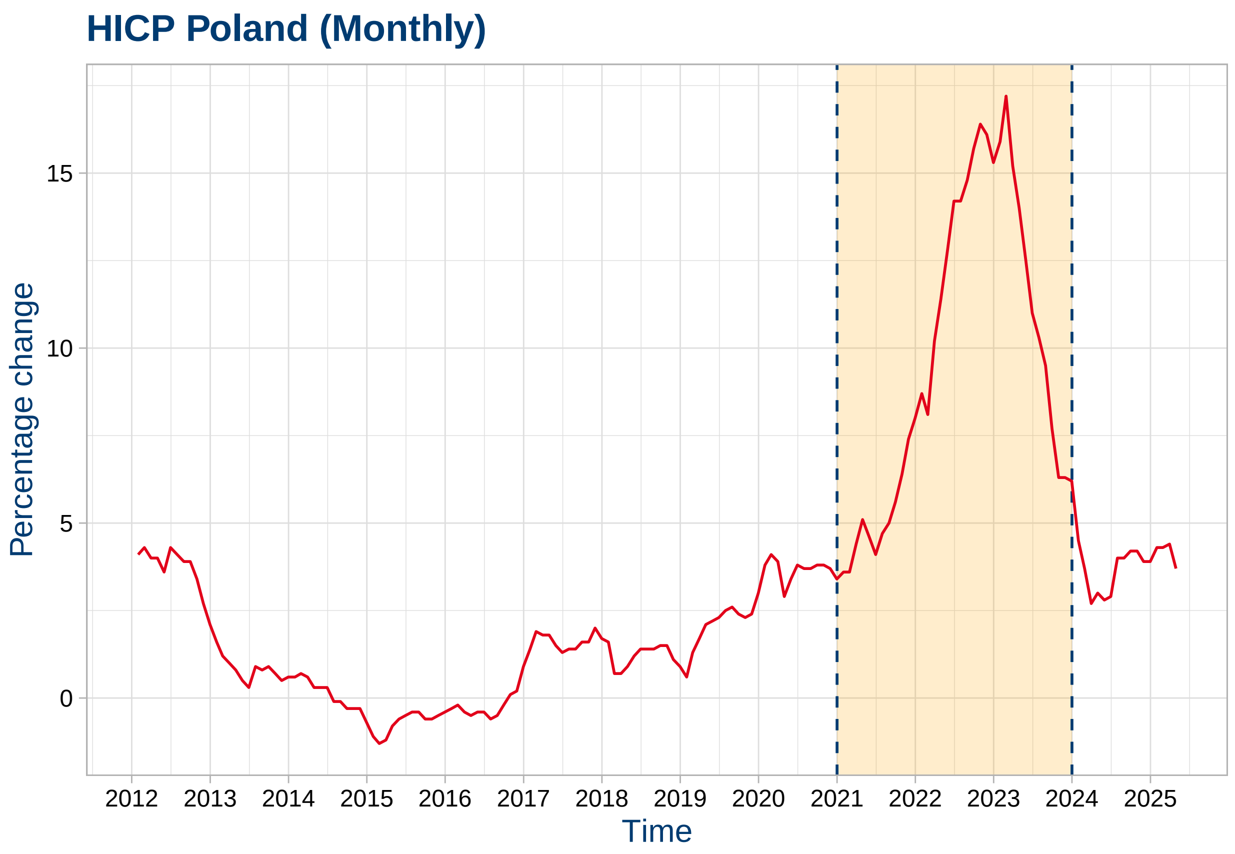
```r
# Training data
train_df <- subset(data_df, subset = data_df$date >= as.Date("2012-01-01") &
                      data_df$date <=  as.Date("2024-12-31"))

# Create time series object
train_ts <- ts(train_df$hicp_index, start = c(2012, 1),
                frequency = 12)

# Test data
test_df <- subset(data_df, subset = data_df$date > as.Date("2024-12-31"))

# Create time series object
test_ts <- ts(test_df$hicp_index, start = c(2025, 1),
               frequency = 12)
```

Stationarity

Stationarity is an characteristic of a time series data. A stationary time series is a process whose statistical properties remain constant over time. A stationary time series does not exhibit trends, seasonality or changing volatility. In other words, it has no seasonality and its mean and variance are constant. Stationarity is probably the most important characteristic as many of the time series models assume stationarity.

Let’s take a look at a plot of the training data set.

```r
ggplot(poland_df, aes(x = date, y = hicp_index)) +
  # Shaded region for trend
  annotate("rect",
           xmin = as.Date("2021-01-01"),
           xmax = as.Date("2024-01-01"),
           ymin = -Inf, ymax = Inf,
           alpha = 0.2, fill = "orange") +
  # Dashed lines for start and end of trend
  geom_vline(xintercept = as.Date("2021-01-01"), linetype = "dashed", color = "#003B71") +
  geom_vline(xintercept = as.Date("2024-01-01"), linetype = "dashed", color = "#003B71") +
  geom_line(color = "#E2001A") +
  labs(
    title = "HICP Poland (Monthly)",
    x = "Time",
    y = "Percentage change"
  ) +
  theme_light() +
  scale_x_date(
    date_breaks = "1 year",
    date_labels = "%Y"
  ) +
  theme(
    plot.title = element_text(color = "#003B71", size = 14, face = "bold"),
    axis.title.x = element_text(color = "#003B71", size = 12),
    axis.title.y = element_text(color = "#003B71", size = 12),
    axis.text = element_text(color = "black")
  )
```



When looking at the plot, it is hard to conclude whether this time series is stationary. Between 2021 and 2024 we can observe 2 really clear trends - first there is a sharp increase in inflation due to COVID measures and afterwards there is a fast decline. This cannot visually prove that the mean and the variance are constant for the whole time series.

One can use the Augmented Dickey-Fuller Test to statistically prove if the time series is stationary. The ADF test checks whether a time series has a unit root, which would indicate non-stationarity. It does this by estimating a regression where the current value of the series is explained by its lagged values and then it tests if the coefficient on the lagged level term is significantly different from zero. The null hypothesis of the test is that the time series has a unit root (i.e. it is non-stationary), while the alternative hypothesis is that the series is stationary. We select a significance level of 5% for the test.

```
# ADF test
# alpha=0.05
# H_0 = there is unit root (non-stationary)
```

```r
# H_A = there is no unit root (stationary)
adf.test(train_ts)
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  train_ts
## Dickey-Fuller = -3.5707, Lag order = 5, p-value = 0.03827
## alternative hypothesis: stationary
```

The ADF test produced a test statistic of -3.5707 with a p-value of 0.03827, which is below the 0.05 significance level. This means that the test detected enough mean reversion in the series. Therefore, we reject the null hypothesis and conclude that the time series is stationary.

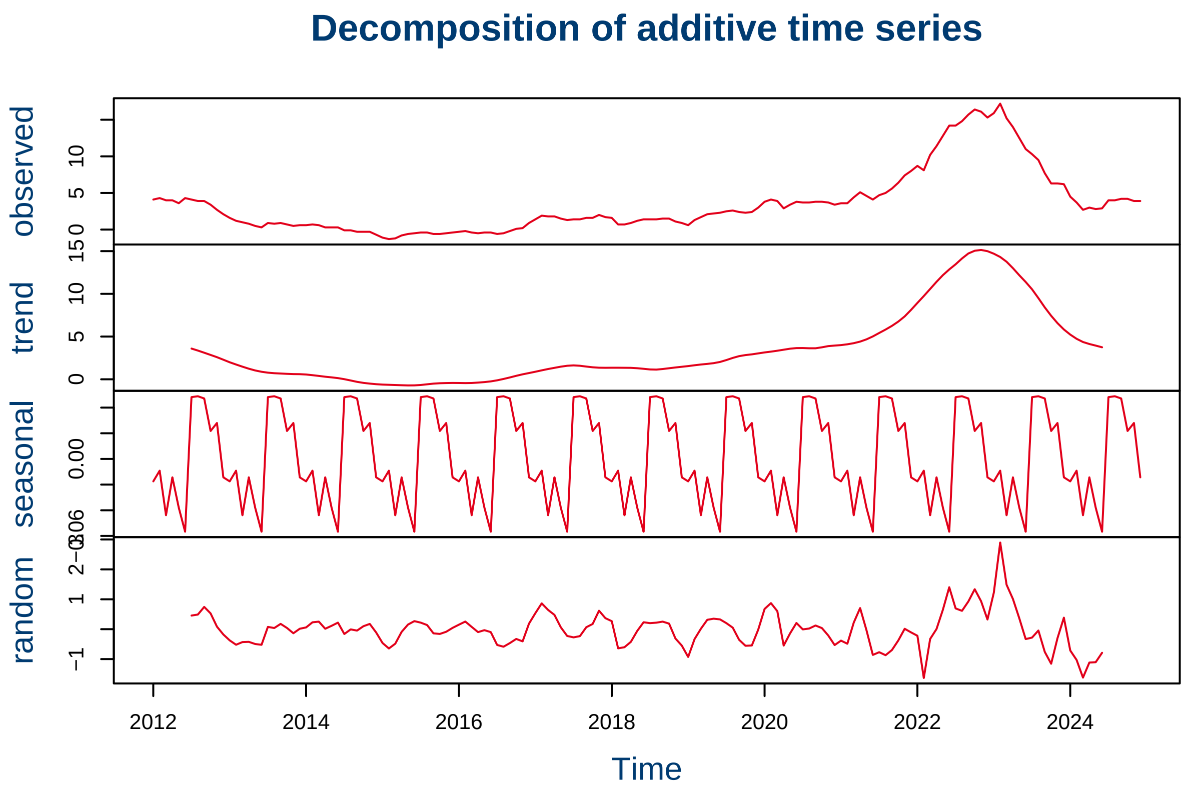
Seasonality and Trend

Seasonality is a characteristic that represents a repeating pattern within a year. These patterns are often driven by external factors. If a time series has seasonality, it is not stationary. That is why it is important to include it into our diagnosis.

Trend, on the other hand, refers to the long-term direction in the data over time. It indicates persistent increase or decrease in the series that is not caused by random variation or seasonal effects.

A time series can be decomposed into its 3 ingredients - Trend, Seasonality and Noise (randomness). Let's take a look at the decomposition plot of the time series.

```r
# Decomposition plot
par(col.main = "#003B71", col.lab = "#003B71")

plot(decompose(train_ts), col = "#E2001A")
```



From 2012 to 2020 the trend is relatively flat with small decline around 2013 and 2017. From early 2021 to late 2022 there's a sharp upward movement when we started to feel the consequences of COVID's monetary and fiscal policies. After 2023 the trend declines noticeably, returning to the pre-COVID baseline inflation of around 3-4%.

This does not represent a constant or linear trend. It reflects structural macroeconomic shocks. Since the COVID period is not a long-term deterministic trend, we won't difference the data.

From the seasonal panel, we can see that the range of its variation is approximately 0.08 percentage points. This is small in absolute terms, especially when compared to the trend panel. The seasonality is statistically detectable, its pattern is regular and stable, but its impact is economically minor. This suggests that most of the variation is explained by trend and shocks and not seasonality. We can still use the seasonal ARIMA model and if the SARIMA doesn't find strong seasonal structure, it will just drop it automatically.

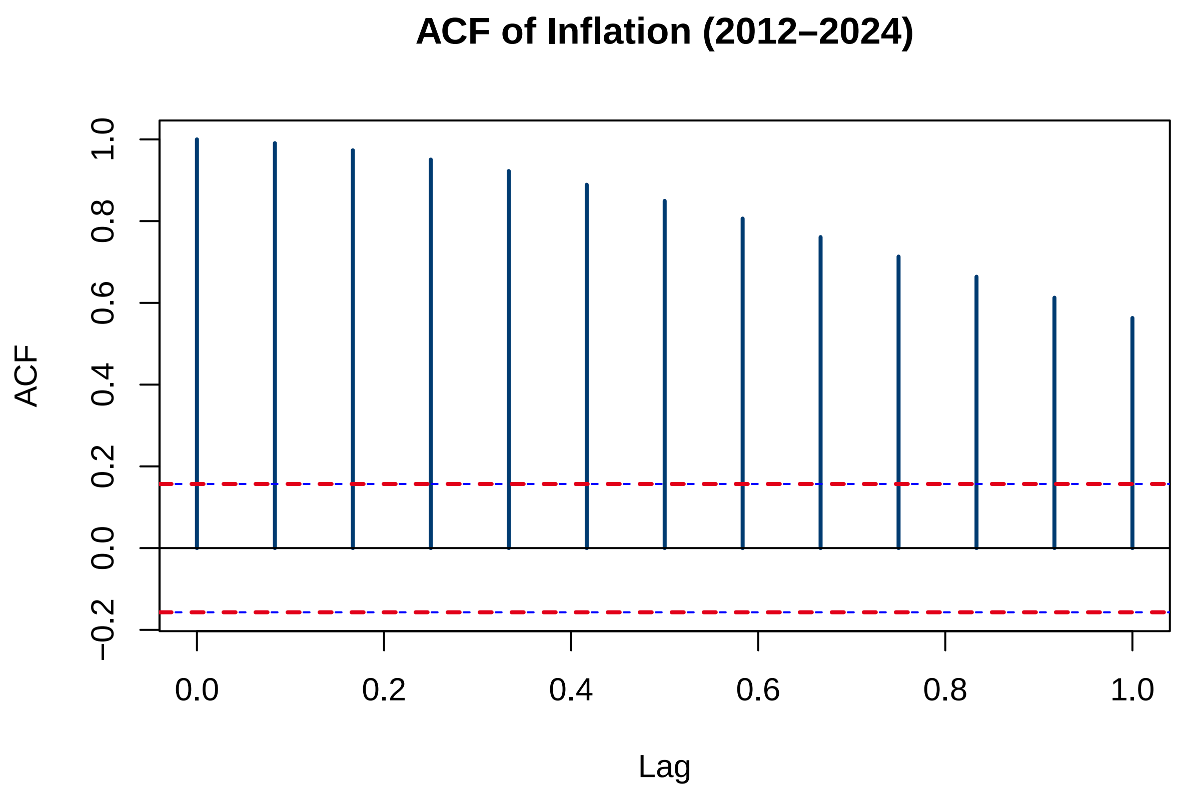
Noise refers to what's left in the time series after removing trend and seasonality. It was relatively stable before 2020 but became noticeably more volatile during 2021–2023. This suggests that external shocks or structural factors likely played a significant role during that period.

Autocorrelation

Autocorrelation measures how the current value of a time series correlates with past values, allowing for more accurate predictions based on recent trends.

Let's take a look at the ACF plot. On the x-axis we can see the time steps (lags) going back in time. On the y-axis we can see the correlation of each time step with the current time.

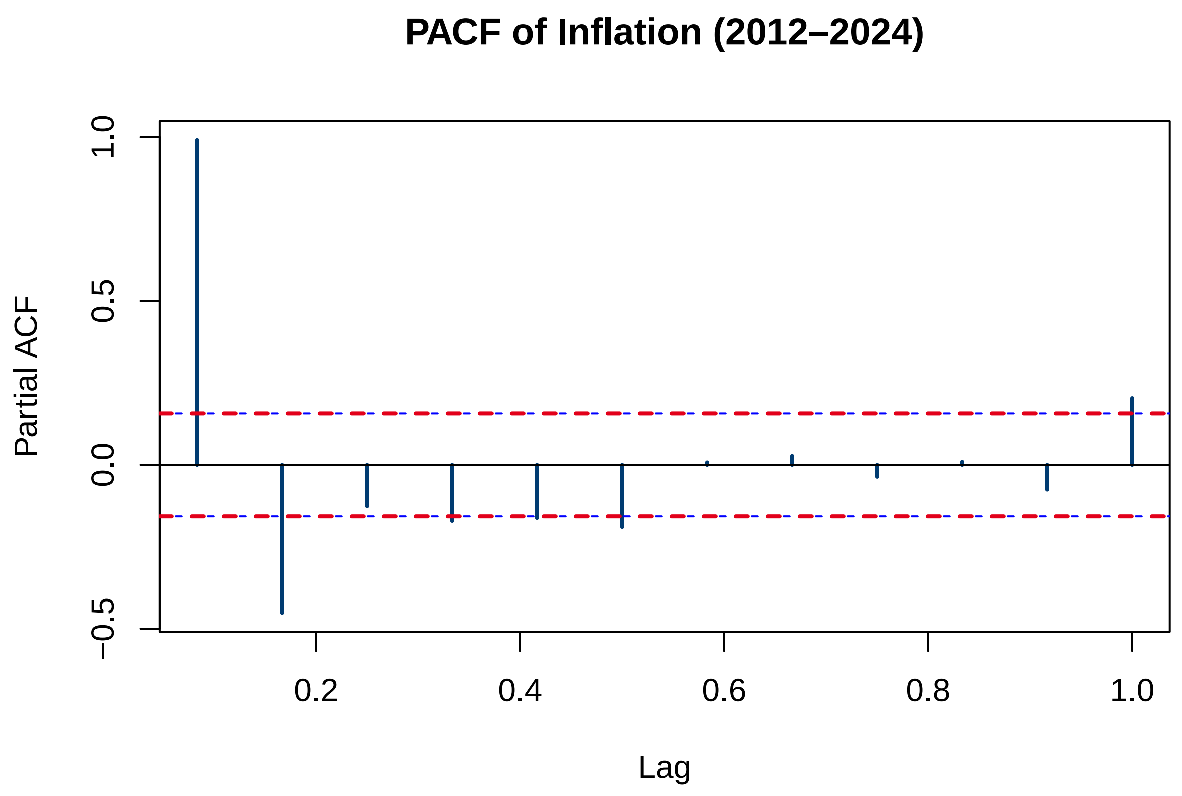
```r
# ACF Plot
acf(train_ts,
    lag.max = 12, lwd = 2,col = "#003B71",
    main = "ACF of Inflation (2012-2024)")

abline(h = c(1.96, -1.96) / sqrt(length(train_ts)),
       col = "#E2001A", lty = 2, lwd = 2)
```



The ACF plot shows significant autocorrelation. At lag=1 we have a value really close to 1. With increasing the lags, the bars slowly and smoothly become shorter. No lags fall outside the blue confidence bounds suggesting statistical significance.

The PACF (partial ACF) is an alternative. Instead of showing all autocorrelations, it shows only the unique correlation at each time step and removes the indirect effects. This helps identify the true relationship between each lag and the present time.

```r
# PACF Plot
pacf(train_ts,
    lag.max = 12, lwd = 2,col = "#003B71",
    main = "PACF of Inflation (2012-2024)")

abline(h = c(1.96, -1.96) / sqrt(length(train_ts)),
       col = "#E2001A", lty = 2, lwd = 2)
```


PACF of Inflation (2012–2024)
Partial ACF
Lag
−0.5
0.0
0.5
1.0
0.2
0.4
0.6
0.8
1.0

The PACF plot shows very strong positive partial autocorrelation at lag=1 but after lag=2 the partial autocorrelation drops quickly. This suggests that the first lag alone explains most of the autocorrelation and higher lags add little value.

Univariate Model Selection and Forecast

Univariate time series model uses only one variable (the target variable) and its variation over time to make future predictions. Based on the time series diagnostics, I will try 2 approaches:

AR(1)/ARIMA(1,0,0) model

I choose this model, because of the PACF results suggesting parameter p=1, the stationarity proven with the ADF test suggesting parameter d=0 and the slow decline of ACF suggesting parameter q=0.

```r
# AR(1) model
model_ar1 <- arima(train_ts, order = c(1, 0, 0))
summary(model_ar1)
```

```
##
## Call:
## arima(x = train_ts, order = c(1, 0, 0))
##
## Coefficients:
##          ar1  intercept
##       0.9866     3.7379
## s.e.  0.0094     2.4845
##
## sigma^2 estimated as 0.3352:  log likelihood = -137.92,  aic = 281.84
##
## Training set error measures:
##                         ME      RMSE       MAE      MPE     MAPE     MASE
## Training set -0.004282424 0.5789923 0.3912852 2.596835 22.93119 1.000812
##                   ACF1
## Training set 0.4529115
```

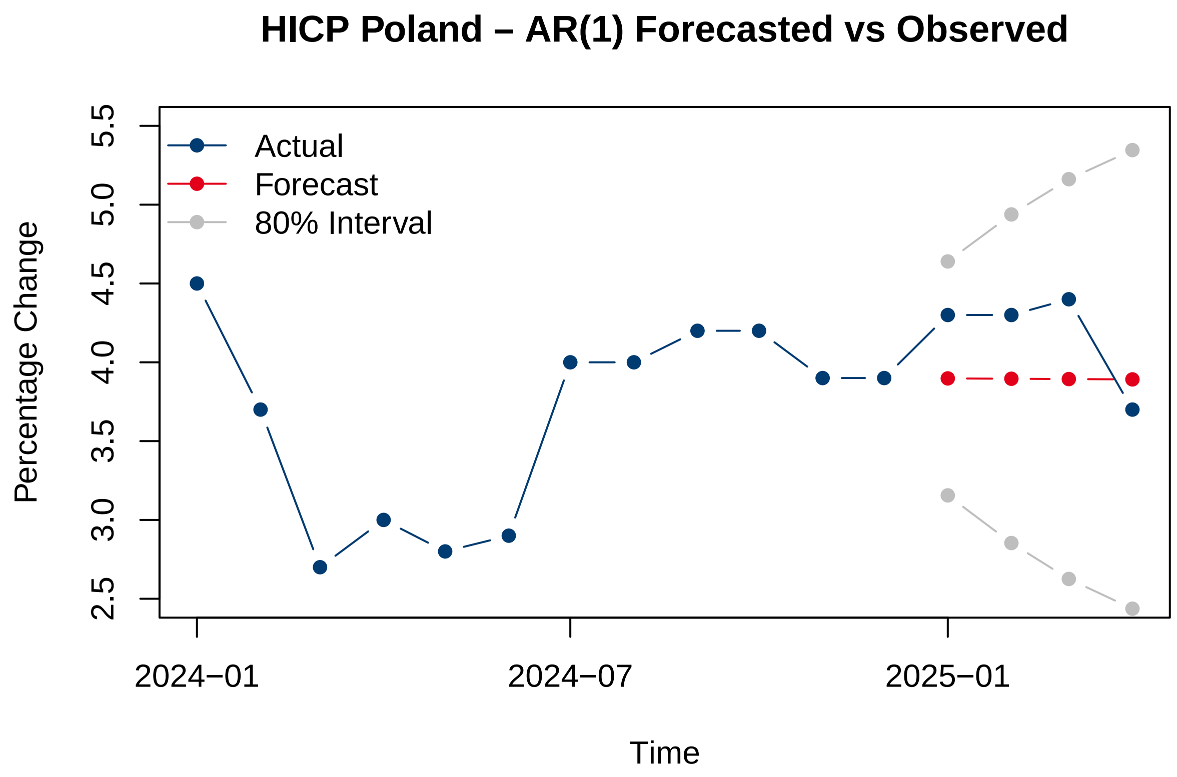
The AR(1) coefficient has very low standard error which suggest strong autocorrelation. The coefficient itself is very close to 1 indicating the time series evolve slowly and retain memory of the past.

The model's residual variance (σ^2) is approximately 0.34 which suggests moderate level of unexplained variance. Its AIC is 281.84.

Let's forecast using the AR(1) model.

```r
# Generate forecasted values for 4 periods
ar1_forecast <- forecast(model_ar1, h = 4)

# Create a df to compare numerically
ar1_forecast_df <- data.frame(
  Observed = test_ts,
  Forecast = ar1_forecast$mean
)
rownames(ar1_forecast_df) <- format(as.Date(time(ar1_forecast$mean)),"%Y %b")
```

```r
ar1_forecast_df

# Big Plot

# Define date sequences for easier plotting
forecast_dates <- seq(as.Date("2025-01-01"), by = "month", length.out = 4)
more_dates <- seq(as.Date("2024-01-01"), by = "month", length.out = 12)
all_dates <- c(more_dates, forecast_dates)

# Plot actual values (2024 and 2025)
plot(all_dates,
     poland_df$hicp_index[poland_df$date >= "2024-01-01"],
     type = "b", col = "#003B71", pch = 16,
     ylim = c(2.5, 5.5),
     xlim = c(as.Date("2024-01-01"), as.Date("2025-04-01")),
     main = "HICP Poland - AR(1) Forecasted vs Observed",
     xlab = "Time", ylab = "Percentage Change")

# Add forecast intervals
lines(forecast_dates, ar1_forecast$lower[,1], col = "gray", type = "b", pch = 16)
lines(forecast_dates, ar1_forecast$upper[,1], col = "gray", type = "b", pch = 16)

# Add AR(1) forecast
lines(forecast_dates, ar1_forecast$mean, col = "#E2001A", type = "b", pch = 16)

# Add legend
legend("topleft", legend = c("Actual", "Forecast", "80% Interval"),
       col = c("#003B71", "#E2001A", "gray"), lty = 1, pch = 16, bty = "n")
```



```
##          Observed Forecast
## 2025 Jan      4.3 3.897829
## 2025 Feb      4.3 3.895688
## 2025 Mar      4.4 3.893575
## 2025 Apr      3.7 3.891490
```

As shown in the plot, the model forecasts a relatively flat inflation trend in early 2025, while the actual data shows a decline in April. The forecast remains within the 80% confidence interval but underestimates the observed values in March and April, indicating that it may lack responsiveness to structural changes or external shocks.

Auto ARIMA

My second approach would be to take advantage of R's prebuilt function *auto.arima()* from the *forecast* library. This function selects automatically the best ARIMA by testing different combinations of p, d and q to minimize the AIC.

The *auto.arima()* function can also estimate seasonal ARIMA model and I will use this function because of the seasonality observed from the decomposition plot. If no seasonality is found, the function will default to a non-seasonal ARIMA model.

```
# SARIMA model
model_sarima <- auto.arima(train_ts, seasonal = T)
summary(model_sarima)
```

```
## Series: train_ts
## ARIMA(1,1,0)(0,0,1)[12]
##
## Coefficients:
##          ar1     sma1
##       0.5082  -0.7709
## s.e.  0.0713   0.0853
##
## sigma^2 = 0.1907:  log likelihood = -96.08
## AIC=198.16   AICc=198.32   BIC=207.29
##
## Training set error measures:
##                      ME      RMSE       MAE      MPE     MAPE     MASE
## Training set 0.03951055 0.4325144 0.2790079 3.335904 18.13826 0.101355
##                    ACF1
## Training set -0.1537011
```

The model selected by minimizing the AIC value is **ARIMA(1,1,0)(0,0,1)[12]**. The model includes one autoregressive term and no moving average.

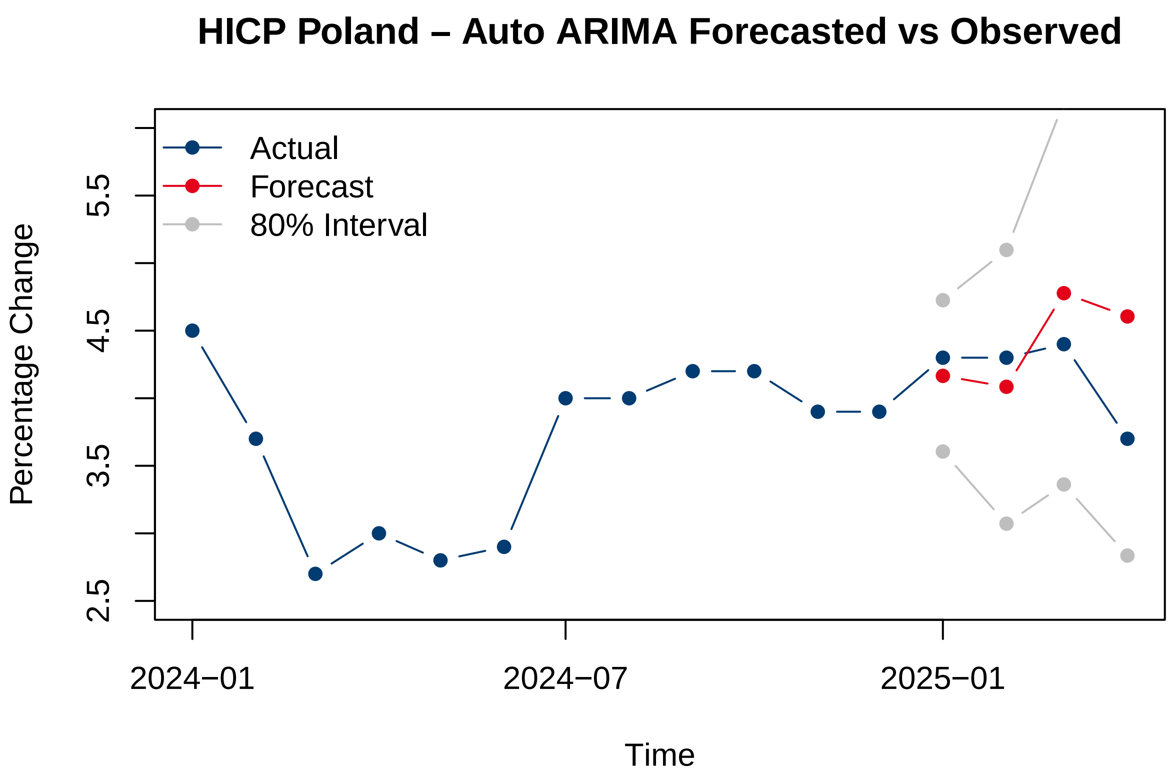
It is interesting that the **d** parameter is set to 1 by the algorithm. This means that the data was differenced once to achieve stationarity which contradicts the results of the ADF test.

The model's residual variance (σ^2) is approximately 0.19 which is lower than the result of the AR(1) model. The AIC value is 198.16 which is significantly lower than the AR(1) model. This implies that it achieved much better model fit than the other model.

Let's forecast using the ARIMA(1,1,0)(0,0,1)[12] model.

```r
# Generate forecasted values for 4 periods
sarima_forecast <- forecast(model_sarima, h = 4)

# Create a df to compare numerically
sarima_forecast_df <- data.frame(
  Observed = test_ts,
  Forecast = sarima_forecast$mean
)
rownames(sarima_forecast_df) <- format(as.Date(time(sarima_forecast$mean)),"%Y %b")

sarima_forecast_df

# Plot actual values (2024 + 2025)
plot(all_dates,
     poland_df$hicp_index[poland_df$date >= "2024-01-01"],
     type = "b", col = "#003B71", pch = 16,
     ylim = c(2.5, 6),
     xlim = c(as.Date("2024-01-01"), as.Date("2025-04-01")),
     main = "HICP Poland - Auto ARIMA Forecasted vs Observed",
     xlab = "Time", ylab = "Percentage Change")

# Add forecast intervals (gray)
lines(forecast_dates, sarima_forecast$lower[,1],
      col = "gray", type = "b", pch = 16)
lines(forecast_dates, sarima_forecast$upper[,1],
      col = "gray", type = "b", pch = 16)
```

```r
# Add forecast mean (red)
lines(forecast_dates, sarima_forecast$mean, col = "#E2001A",
      type = "b", pch = 16)

# Add legend
legend("topleft", legend = c("Actual", "Forecast", "80% Interval"),
       col = c("#003B71", "#E2001A", "gray"), lty = 1, pch = 16, bty = "n")
```



```
##          Observed Forecast
## 2025 Jan      4.3 4.165496
## 2025 Feb      4.3 4.084557
## 2025 Mar      4.4 4.777414
## 2025 Apr      3.7 4.605298
```

Visually, the model tracks January and February quite well, but begins to overestimate inflation in March and April. This may be due to the model interpreting short-term shocks as long-term trends, which is a limitation of univariate models without external context.

Expanding the Model

In order to improve the model's predictive power, one can use more predictor variables. This will provide the model with more context about the underlying patterns and external influences affecting the target variable. As a result, one tries to make the forecasts more accurate and robust across different conditions.

Unemployment

I chose the unemployment rate as an external variable for the extended model based on the well-known Phillips curve relationship that suggests an inverse link between unemployment and inflation. When unemployment is low, labor shortages put upward pressure on wages, which in turn leads to cost-push inflation. On the other hand, higher unemployment reduces inflationary pressure due to weaker demand. The monthly unemployment rate data from Eurostat can be found here.

Inflation tends to respond to labor market conditions in near real time in Poland. However, I tested different lagged versions of the unemployment rate and found that the 1-month lag performs the best.

```r
# Load unemployment data
unemployment_data <- read.csv("unemployment_eurostat.csv")

# Create a df to work with
unemployment_full_df <- data.frame(
  date = as.Date(unemployment_data$DATE),
  unemployment_perc = unemployment_data$X.LFSI.M.PL.S.UNEHRT.TOTAL0.15_74.T.
)

# Create a subset for broader period to lag
unemployment_df <- subset(unemployment_full_df,
                  subset = unemployment_full_df$date > as.Date("2011-09-01") &
                           unemployment_full_df$date <  as.Date("2025-05-01"))

# Generate the time element for april 2025
new_date <- unemployment_df$date[nrow(unemployment_df)] %m+% months(1)

# Append the time element as new row
unemployment_df <- rbind(unemployment_df, data.frame(
  date = new_date,
  unemployment_perc = NA
))

# Create a lagged column
unemployment_df$unemployment_perc_lag1 <- dplyr::lag(unemployment_df$unemployment_perc,1)

# Create a final subset for the period
unemployment_df <- subset(unemployment_df,
                          subset = unemployment_df$date > as.Date("2012-01-01") &
                            unemployment_df$date <  as.Date("2025-05-01"))


# Merge the unemployment df with inflation df
poland_df$year_month <- format(poland_df$date, "%Y-%m")
unemployment_df$year_month <- format(unemployment_df$date, "%Y-%m")
poland_df$date <- NULL
unemployment_df$date <- NULL

# Create a new df to store all other variables later
poland_full_df <- merge(poland_df, unemployment_df[,-1], by ="year_month")
```

Oil Price

I will include Brent crude oil prices as an external variable because energy costs are a major driver of consumer price inflation, particularly through transport and production inputs. Big changes in oil prices often lead to immediate effects on inflation. The data is from the Federal Reserve Economic Data (FRED), it provides daily oil spot prices in USD and can be found here.

To be able to measure the effect of oil price changes on the polish economy, I will convert the oil prices in PLN. For exchange rate, I will download the data for the USD/PLN currency pair from Yahoo Finance with the help of the quantmod library. I will then convert it to monthly data so it matches our main data set.

```r
# Load oil price data
oil_data <- read.csv("DCOILBRENTEU.csv")
colnames(oil_data) <- c("date", "oil_price")

# Create a subset for broader period to lag
oil_df <- subset(oil_data, subset = oil_data$date >= as.Date("2011-01-01") &
                   oil_data$date <  as.Date("2025-05-01"))

# Download USD FX rate
getSymbols("USDPLN=X", src = "yahoo",from = oil_df$date[1])
```

```
## [1] "USDPLN=X"
```

```r
usdpln_df <- data.frame(
  date = index(`USDPLN=X`),
  close = Cl(`USDPLN=X`)
)

# Merge both by date
oil_df <- merge(oil_df, usdpln_df, by="date")

# Calculate daily oil prices in PLN
oil_df$oil_price_pln <- oil_df$oil_price * oil_df$USDPLN.X.Close
oil_df$year_month <- format(as.Date(oil_df$date), "%Y-%m")

# Calculate average price by months
oil_pln_monthly_df <- oil_df %>%
  group_by(year_month) %>%
  summarize(oil_price_pln = mean(oil_price_pln, na.rm = TRUE)) %>%
  ungroup()

# Lag the data
oil_pln_monthly_df$oil_price_pln_lag9 <- dplyr::lag(oil_pln_monthly_df$oil_price_pln, 9)

# Create a final subset for the period
oil_pln_monthly_df <- subset(oil_pln_monthly_df,
                             subset = as.Date(paste0(oil_pln_monthly_df$year_month,"-01")) >= as.Date("
                               as.Date(paste0(oil_pln_monthly_df$year_month,"-01")) <  as.Date("2025-05

#usd_pln_monthly_df <- oil_df %>%
#  group_by(year_month) %>%
#  summarize(usd_pln_close = mean(USDPLN.X.Close, na.rm = TRUE)) %>%
#  ungroup()
# Merge the data frame with the main df
poland_full_df <- merge(poland_full_df, oil_pln_monthly_df[,-2], by="year_month")
```

EUR/PLN and USD/PLN Exchange Rates

As Poland is a relatively small open economy with significant import dependence, exchange rate fluctuations directly affect domestic prices. A depreciation of the PLN versus the USD or EUR leads to higher import prices and pushes up inflation, especially for dollar-denominated goods like fuel and agricultural commodities or euro-denominated goods like cars and industrial machinery, coming mostly from Germany and France.

The data for both currency pairs is downloaded directly from Yahoo Finance with a function from the quantmod library. I decided to use 12-month log difference of both currency pairs to capture long-run effects and make volatility smoother.

```r
# Download USD/PLN exchange rate for broader period
getSymbols("USDPLN=X", src = "yahoo",from = "2011-01-01")
```

```
## [1] "USDPLN=X"
```

```r
# Pre-processing
usdpln_df <- data.frame(
  date = index(`USDPLN=X`),
  close = Cl(`USDPLN=X`)
)

usdpln_df$year_month <- format(as.Date(usdpln_df$date), "%Y-%m")
colnames(usdpln_df)[2] <- "usd_pln_close"

# Calculate average rate by months
usd_pln_monthly_df <- usdpln_df %>%
  group_by(year_month) %>%
  summarize(usd_pln_close = mean(usd_pln_close, na.rm = TRUE)) %>%
  ungroup()

# Calculate YoY rate
usd_pln_monthly_df$usd_pln_yoy[13:nrow(usd_pln_monthly_df)] <- diff(log(usd_pln_monthly_df$usd_pln_clos

# Fix the df
usd_pln_monthly_df <- na.omit(usd_pln_monthly_df)
usd_pln_monthly_df <- usd_pln_monthly_df[-nrow(usd_pln_monthly_df),]
usd_pln_monthly_df[,2] <- NULL

# Download EUR/PLN exchange rate for broader period
getSymbols("EURPLN=X", src = "yahoo",from = "2010-09-01")
```

```
## [1] "EURPLN=X"
```

```r
# Pre-processing
eurpln_df <- data.frame(
  date = index(`EURPLN=X`),
  close = Cl(`EURPLN=X`)
```

```r
)

eurpln_df$year_month <- format(as.Date(eurpln_df$date), "%Y-%m")
colnames(eurpln_df)[2] <- "eur_pln_close"

# Calculate average rate by months
eur_pln_monthly_df <- eurpln_df %>%
  group_by(year_month) %>%
  summarize(eur_pln_close = mean(eur_pln_close, na.rm = TRUE)) %>%
  ungroup()

# Calculate YoY rate
eur_pln_monthly_df$eur_pln_yoy[13:nrow(eur_pln_monthly_df)] <- diff(log(eur_pln_monthly_df$eur_pln_clos

# Lag the YoY rate
eur_pln_monthly_df$eur_pln_yoy_lag1 <- dplyr::lag(eur_pln_monthly_df$eur_pln_yoy, 1)

# Fix the df
eur_pln_monthly_df <- na.omit(eur_pln_monthly_df)
eur_pln_monthly_df <- eur_pln_monthly_df[-1,]
eur_pln_monthly_df <- eur_pln_monthly_df[-nrow(eur_pln_monthly_df),]

eur_pln_monthly_df <- subset(eur_pln_monthly_df,
          subset = as.Date(paste0(eur_pln_monthly_df$year_month,"-01")) >= as.Date("2012-01-01") &
          as.Date(paste0(eur_pln_monthly_df$year_month,"-01")) <  as.Date("2025-05-01"))

eur_pln_monthly_df[,2] <- NULL

# Merge the data frames with the main df
poland_full_df <- merge(poland_full_df, usd_pln_monthly_df, by="year_month")
poland_full_df <- merge(poland_full_df, eur_pln_monthly_df, by="year_month")
```

Global Food Price Index

I included the FAO Global Food Price Index as an external variable because food prices are a key driver of inflation, especially in Poland where food has a significant weight in the consumer basket. Increases in global food prices often lead to higher domestic food inflation through import costs and supply chain pressures. This variable captures external reasons for inflation linked to global agricultural markets, wars or trade restrictions. The data is monthly, it is from the Food and Agriculture Organization and can be found here.

```r
# Load food price index data
gfpi_data <- read.csv("food_price_indices_data_may25.csv")

# Pre-processing
colnames(gfpi_data) <- gfpi_data[2,]
gfpi_data <- gfpi_data[-(1:3),]
gfpi_data <- gfpi_data[,1:2]
gfpi_data$Date <- as.Date(paste0(gfpi_data$Date, "-01"))

# Subset for the period
gfpi_df <- subset(gfpi_data, subset = gfpi_data$Date >= as.Date("2012-01-01") &
                    gfpi_data$Date <  as.Date("2025-05-01"))
```

```r
gfpi_df$year_month <- format(gfpi_df$Date, "%Y-%m")
gfpi_df$Date <- NULL
colnames(gfpi_df)[1] <- "global_food_price_index"

# Merge into the full df
poland_full_df <- merge(poland_full_df, gfpi_df, by="year_month")
poland_full_df$global_food_price_index <- as.numeric(poland_full_df$global_food_price_index)
```

Average nominal gross wage

I decided to use the average nominal gross wage as wages are a key driver of inflation. As wages increase, households have more disposable income, increasing consumption and placing upward pressure on prices. At the same time, rising wages raise business costs. The data is from Statistics Poland (GUS), section "Wages and salaries and social benefits".

I decided to use 2-month lagged data as the effect of wage changes on inflation is not immediate. I tested different lagged versions of the average wage data and found that the 2-month lag performs the best.

```r
# Load wages data
wages_data <- read.csv("Wages and salaries and social benefits_monthly.csv", sep=";")

# Data pre-processing
wages_data <- wages_data[1:2,-(1:99)]

wages_ts <- ts(as.numeric(wages_data[2,]), start = c(2008, 1), frequency = 12)

wages_df <- data.frame(
  year_month = as.yearmon(time(wages_ts)),
  avg_wage = wages_ts
)
wages_df <- subset(wages_df,
                    subset = as.Date(wages_df$year_month) >= as.Date("2011-06-01") &
                      as.Date(wages_df$year_month) <  as.Date("2025-05-01"))

# Generate the time element for april 2025
new_date <- as.Date(wages_df$year_month[nrow(wages_df)]) %m+% months(1)

# Append new row for april 2025
wages_df <- rbind(wages_df, data.frame(
  year_month = as.yearmon(new_date),
  avg_wage = NA
))

wages_df$year_month <- format(as.Date(wages_df$year_month, format = "%b %Y"),
                              "%Y-%m")

# Lag the data
wages_df$avg_wage_lag2 <- dplyr::lag(wages_df$avg_wage, 2)

# Fix the period
wages_df <- subset(wages_df,
       subset = as.Date(paste0(wages_df$year_month,"-01")) >= as.Date("2012-01-01"))
```

```r
# Merge data frames
poland_full_df <- merge(poland_full_df, wages_df[,-2], by="year_month")
```

Long-Term Government Bond Yield

As a last explanatory variable I included 10-year government bond yield for Poland because it shows what investors think will happen with inflation and interest rates in the future. It reflects market expectations and overall economic confidence. When the yield goes up, it usually means investors expect higher inflation ahead. The data is from FRED and can be found here

I use a 6-month lag because changes in bond yields don't affect inflation right away. It takes time for higher borrowing costs or changing expectations to influence consumer prices.

```r
# Load bond yield data
bond_yield_data <- read.csv("monthly.csv")

# Data Pre-processing
colnames(bond_yield_data) <- c("date", "bond_yield")
bond_yield_data$date <- as.Date(bond_yield_data$date)

bond_yield_df <- subset(bond_yield_data,
                           subset = bond_yield_data$date >= as.Date("2011-03-01") &
                             bond_yield_data$date <  as.Date("2025-05-01"))

# Generate the time element for april 2025
new_date <- bond_yield_df$date[nrow(bond_yield_df)] %m+% months(1)

# Create a new row for april 2025
bond_yield_df <- rbind(bond_yield_df, data.frame(
  date = as.Date(new_date),
  bond_yield = NA
))

# Lag the data
bond_yield_df$bond_yield_lag6 <- dplyr::lag(bond_yield_df$bond_yield, 6)

# Fix the period
bond_yield_df <- subset(bond_yield_df,
                           subset = bond_yield_df$date >= as.Date("2012-01-01") &
                             bond_yield_df$date <  as.Date("2025-05-01"))

# Merge data frames
bond_yield_df$year_month <- format(bond_yield_df$date, "%Y-%m")
bond_yield_df$date <- NULL

poland_full_df <- merge(poland_full_df, bond_yield_df[,-1], by="year_month")
```

Forecasting

Before training the new model, we need to split the full data frame into training set (2012/01-2024/12) and testing set(2025/01-2025/05).

```r
# Training data
full_train_df <- subset(poland_full_df,
                        as.Date(paste0(year_month, "-01")) >= as.Date("2012-01-01") &
                          as.Date(paste0(year_month, "-01")) <= as.Date("2024-12-31")
)

# Create training time series object
full_train_ts <- ts(full_train_df$hicp_index, start = c(2012, 1),
                    frequency = 12)

# Create external variables df
ext_vars_full_train_df <- full_train_df[, -c(1,2)]
rownames(ext_vars_full_train_df) <- full_train_df$year_month

# Test data
full_test_df <- subset(poland_full_df,
                       subset = as.Date(paste0(year_month, "-01")) > as.Date("2024-12-31"))

# Create test time series object
full_test_ts <- ts(full_test_df$hicp_index, start = c(2025, 1),
                   frequency = 12)

# Create external variables df
ext_vars_full_test_df <- full_test_df[, -c(1,2)]
rownames(ext_vars_full_test_df) <- full_test_df$year_month
```

Now I will train the model on the training set. I am going to use the *auto.arima()* estimation function again as it selects automatically the best ARIMA by testing different combinations of p, d and q to minimize the AIC.

```r
full_model_arimax <- auto.arima(full_train_ts,
                                xreg = data.matrix(ext_vars_full_train_df))
summary(full_model_arimax)
```

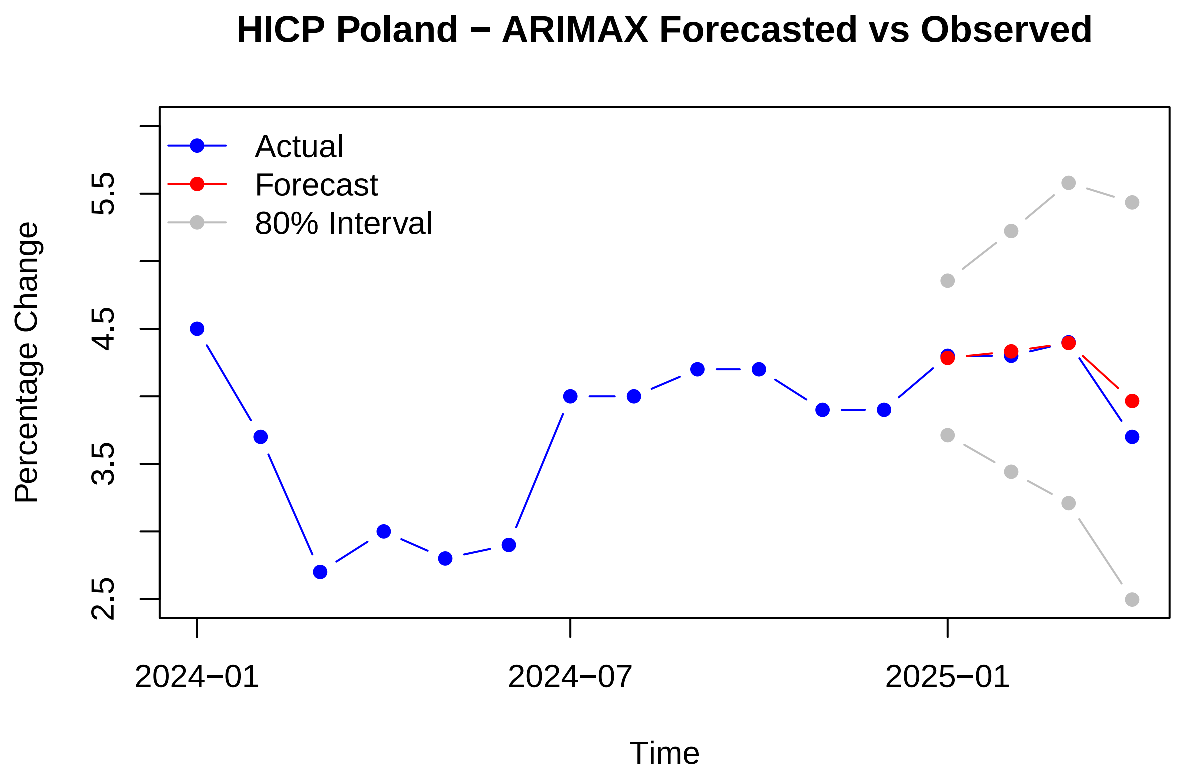
```
## Series: full_train_ts
## Regression with ARIMA(2,0,1)(1,0,0)[12] errors
##
## Coefficients:
##          ar1      ar2      ma1     sar1  unemployment_perc_lag1
##       1.9224  -0.9293  -0.7274  -0.4273                 -0.3755
## s.e.  0.0438   0.0432   0.0872   0.0800                  0.2414
##       oil_price_pln_lag9  usd_pln_yoy  eur_pln_yoy  eur_pln_yoy_lag1
##                   0.0009       3.4995      -3.7519            2.1943
## s.e.              0.0013       1.8055       3.1143            2.1143
##       global_food_price_index  avg_wage_lag2  bond_yield_lag6
##                        0.0452         0e+00          -0.0428
## s.e.                   0.0113         1e-04           0.1118
##
```

```
## sigma^2 = 0.199:  log likelihood = -92.54
## AIC=211.08   AICc=213.64   BIC=250.73
##
## Training set error measures:
##                       ME      RMSE       MAE       MPE     MAPE     MASE
## Training set 0.005184756 0.4286257 0.3061531 0.3851896 19.84501 0.111216
##                    ACF1
## Training set 0.04359644
```

The model selected by minimizing the AIC value is ARIMA(2,1,0)(1,0,0)[12]. The model has 2 autoregressive terms, meaning that it uses the last two values of the time series to predict the current one. It has no differences meaning the data is stationary and it has 1 moving average term. There is also a seasonal AR term with cycle of 12.

Compared to the univariate model trained earlier (ARIMA(1,1,0)(0,0,1)[12]), this model has a slightly higher residual variance (0.199 compared to 0.1907). Its AIC value is also higher than the simple model, suggesting this model has worse in-sample fit. However, this doesn't necessarily mean it will outperform out-of-sample, so let's see how it copes with forecasting.

```r
# Generate forecasted values for 4 periods
full_model_arimax_forecast <- forecast(full_model_arimax,
                                       xreg = data.matrix(ext_vars_full_test_df), h = 4)

# Create a df to compare numerically
full_model_arimax_forecast_df <- data.frame(
  Observed = full_test_ts,
  Forecast = full_model_arimax_forecast$mean
)
rownames(full_model_arimax_forecast_df) <- format(
  as.Date(time(full_model_arimax_forecast$mean)),"%Y %b")

full_model_arimax_forecast_df

# Big Plot
plot(c(more_dates,forecast_dates),
     poland_full_df$hicp_index[as.Date(paste0(poland_full_df$year_month,"-01")) >= "2024-01-01"],
     type = "b", col="blue",
     ylim = c(2.5, 6), pch=16,
     xlim = c(as.Date("2024-01-01"), as.Date("2025-04-01")),
     main = "HICP Poland - ARIMAX Forecasted vs Observed",
     xlab = "Time", ylab = "Percentage Change")

# Add 2025 forecast intervals (gray)
lines(forecast_dates, full_model_arimax_forecast$lower[,1],
      col = "gray", type = "b", pch = 16)
lines(forecast_dates, full_model_arimax_forecast$upper[,1],
      col = "gray", type = "b", pch = 16)

# Add 2025 forecast mean (red)
lines(forecast_dates, full_model_arimax_forecast$mean,
      col = "red", type = "b", pch = 16)

legend("topleft", legend = c("Actual", "Forecast", "80% Interval"),
       col = c("blue", "red", "gray"), lty = 1, pch = 16, bty = "n")
```



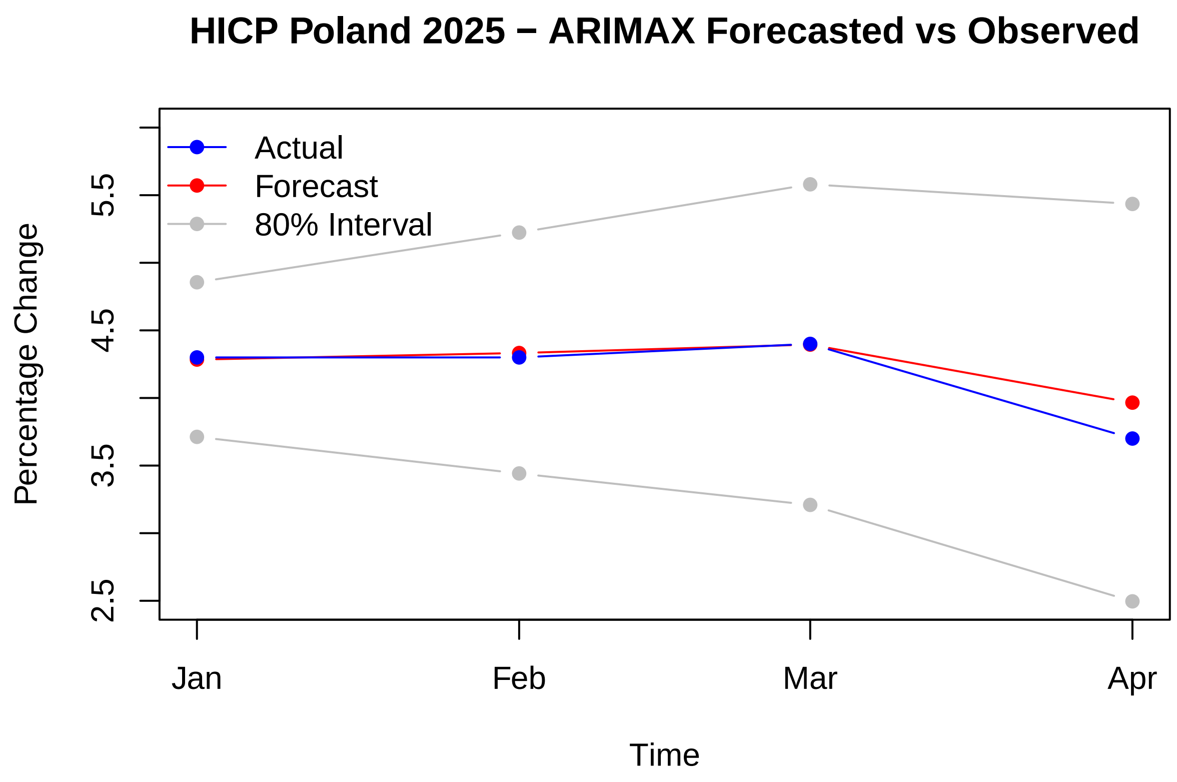
```
##          Observed Forecast
## 2025 Jan      4.3 4.284120
## 2025 Feb      4.3 4.332600
## 2025 Mar      4.4 4.394824
## 2025 Apr      3.7 3.965498
```

As seen in the chart, the ARIMAX model tracks the actual inflation path very closely. It slightly overpredicts April 2025, but overall it aligns much better with observed data compared to the simpler models. The ability to incorporate macroeconomic signals gives this model an edge in adapting to structural changes and explaining the underlying drivers of inflation.

Let's take a closer look at the forecast period only.

```r
# Plot only forecast period
forecast_dates <- seq(as.Date("2025-01-01"), by = "month", length.out = 4)

plot(forecast_dates, full_model_arimax_forecast$mean,
     type="b", ylim = c(2.5,6), col="red", pch = 16,
     main="HICP Poland 2025 - ARIMAX Forecasted vs Observed",
     xlab = "Time", ylab="Percentage Change")
lines(forecast_dates,full_model_arimax_forecast$lower[,1],
      type="b", pch=16, col="gray")
lines(forecast_dates, full_model_arimax_forecast$upper[,1],
      type="b", pch=16, col="gray")
lines(forecast_dates, full_test_ts, type="b",
      col = "blue", pch=16)
```

```r
legend("topleft", legend = c("Actual", "Forecast", "80% Interval"),
       col = c("blue", "red", "gray"), lty = 1, pch = 16, bty = "n")
```



Although the ARIMAX model has a slightly higher AIC than the univariate Auto ARIMA, this is expected due to its greater complexity. The AIC formula penalizes model's that are too complex (include many variables) and this is the case with our ARIMAX model's AIC value.

However, AIC evaluates in-sample fit, not forecasting performance. ARIMAX incorporates real-world economic drivers. This allows it to anticipate structural changes and respond to external shocks that univariate models cannot detect.

As a result, the ARIMAX model generalizes better and produced more accurate out-of-sample forecasts for 2025. This highlights the strength of models with more context.

Final Words

This project demonstrated how time series modeling can produce accurate and meaningful inflation forecasts. The ARIMAX model, despite being more complex, outperformed simpler alternatives by incorporating external variables that reflect external drivers of inflation.

While the AR(1) and Auto ARIMA models were useful for benchmarking and fast forecasting, they lacked responsiveness to big macroeconomic changes. The ARIMAX model generalized better and offered better forecasts.

The ARIMAX model performed well, but there is always room for improvement. One extension could be exploring Bayesian regression techniques to handle potential multicollinearity and reduce overfitting from multiple predictors. One could also think "outside of the box" and incorporate indicators that do not seem related at first sight, but could have explanatory power over inflation.

Appendix

Used literature for this project:

The Complete Guide to Time Series Models - Marco Peixeiro

How to Select a Model For Your Time Series Prediction Task - Joos Korstanje

Ritvikmath Youtube Channel

“Time Series in R” Track at DataCamp