Time Series

Fátima Rodrigues
Departamento Engenharia Informática
mfc@isep.ipp.pt

Importance of Time Series Analysis

Ample of time series data is being generated from a variety of fields. And hence the study of time series analysis holds a lot of applications

Time series analysis is important in different areas:

- Economics
- Finance
- Healthcare
- Environmental Science
- Sales Forecasting

Why & where Time Series is used?

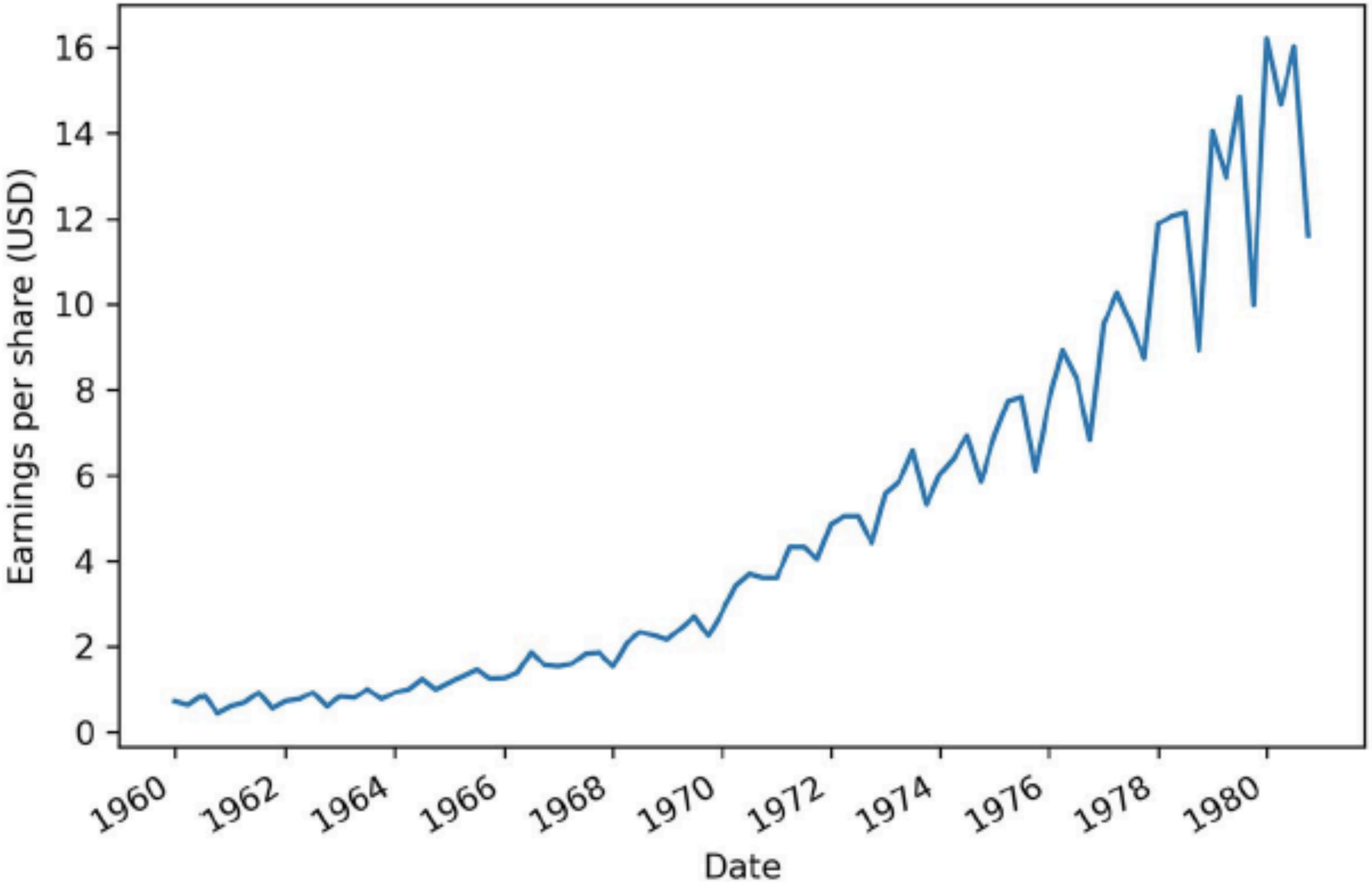
Time series data can be analysed in order to extract meaningful statistics and other characteristics

It's used in at least in the four scenarios:

- Business Forecasting
- Understanding past behaviour
- Plan the future
- Evaluate current accomplishment

Time Series

- Time Series is simple a set of data points ordered in time intervals (usually equal time intervals)



Components of a Time Series

Time series decomposition is a process by which we separate a time series into its components:

- **Trend**

  Represents the slow-moving changes in a time series. It is responsible for making the series gradually increase or decrease over time

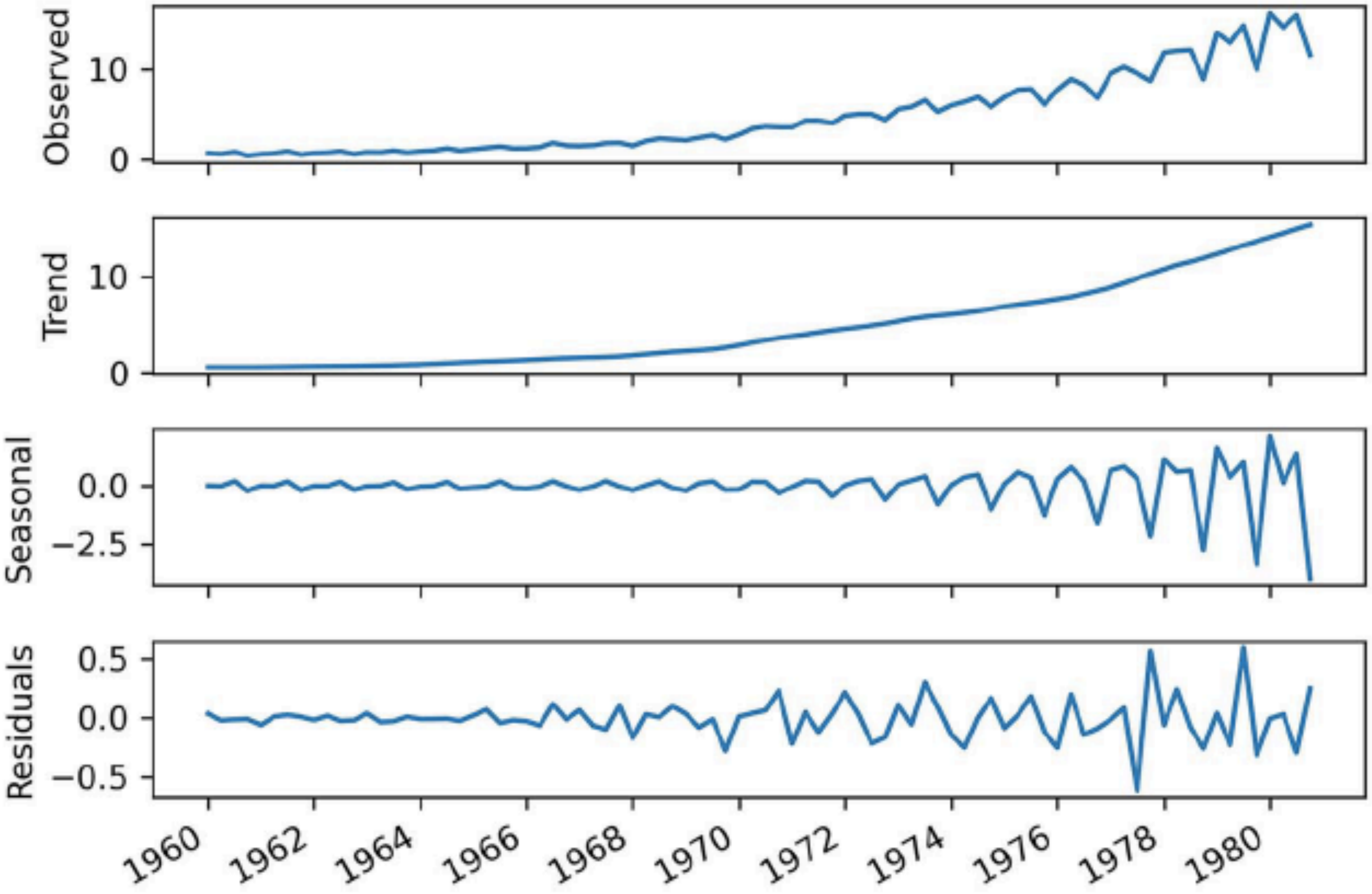
- **Seasonality**

  Represents the seasonal pattern in the series. The cycles occur repeatedly over a fixed period of time

- **Residuals**

  Represent the behaviour that cannot be explained by the trend and seasonality components. They correspond to random errors, also termed white noise

Components of a Time Series



Baseline models

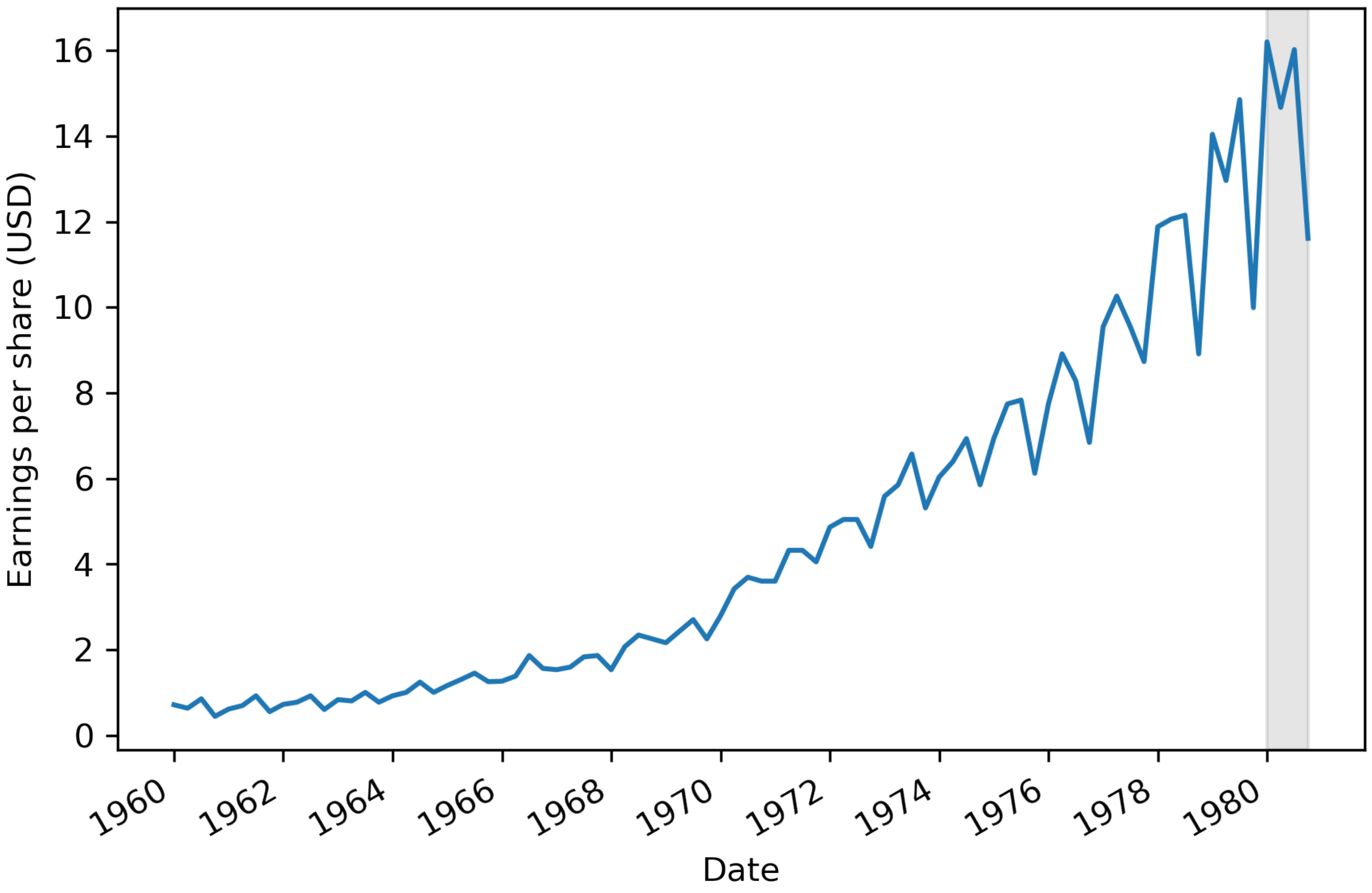
Baseline model

- A baseline model is a trivial solution for forecasting a problem
- It relies on heuristics or simple statistics and is usually the simplest solution
- It does not require model fitting, and it is easy to implement

Examples of baseline models:

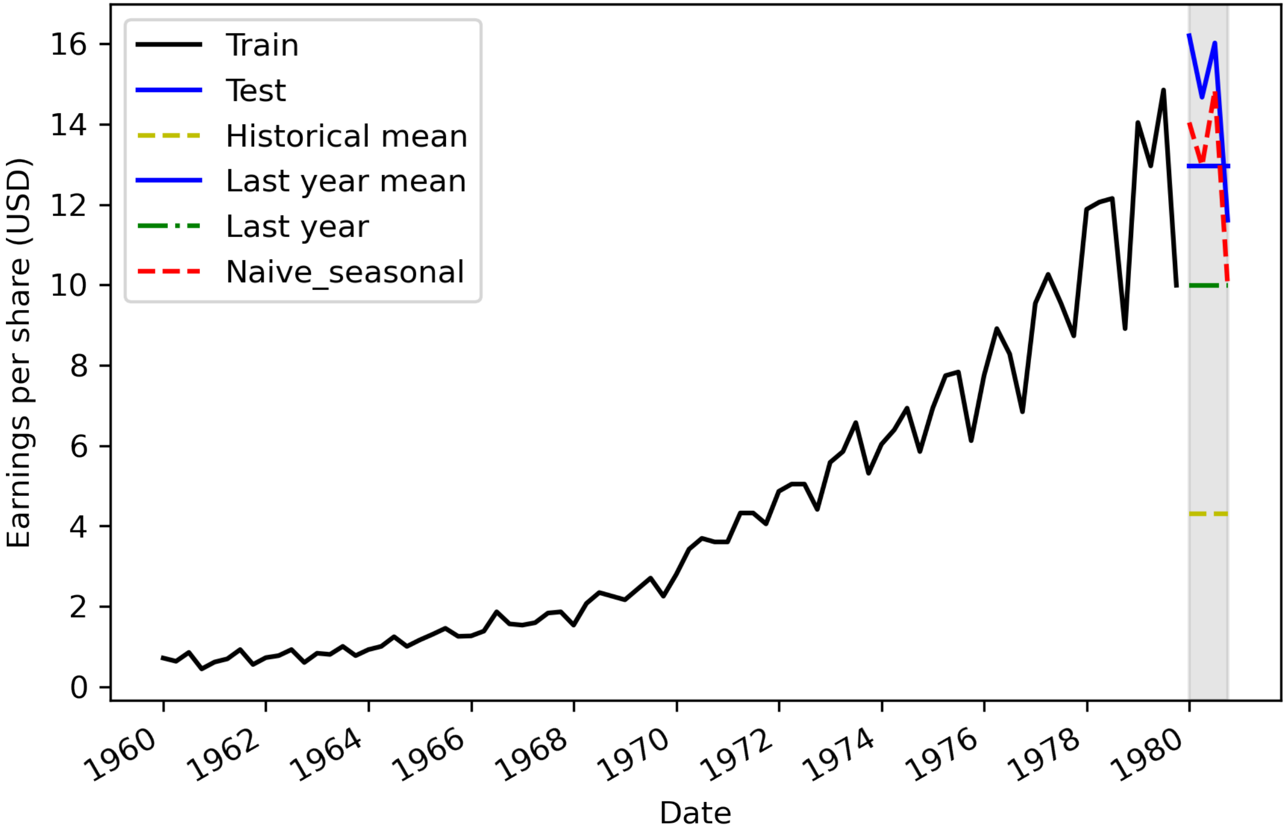
- historical mean
- last year’s mean
- last known value
- naive seasonal forecast

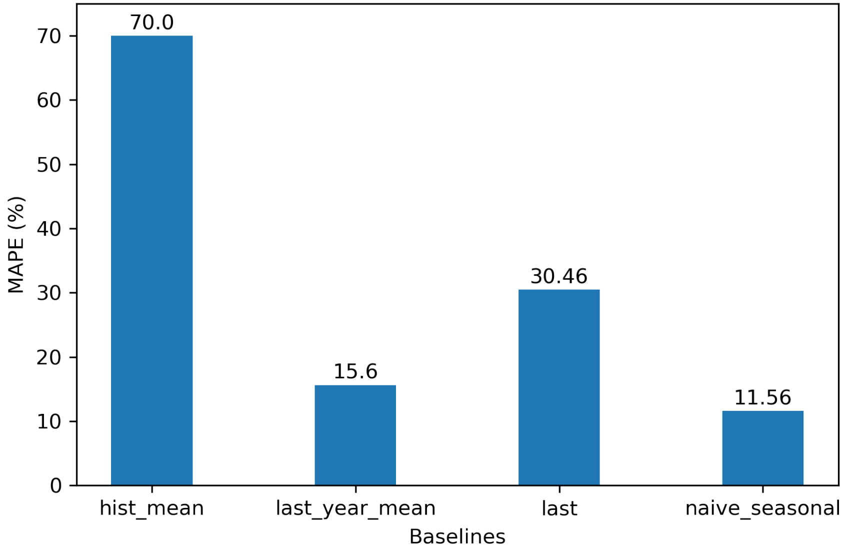
Predict the last four quarters with baseline models



Predict the last four quarters with baseline models

	date	data	hist_mean	last_year_mean	last	naive_seasonal
80	1980-01-01	16.20	4.3085	12.96	9.99	14.04
81	1980-04-01	14.67	4.3085	12.96	9.99	12.96
82	1980-07-02	16.02	4.3085	12.96	9.99	14.85
83	1980-10-01	11.61	4.3085	12.96	9.99	9.99





Random Walk

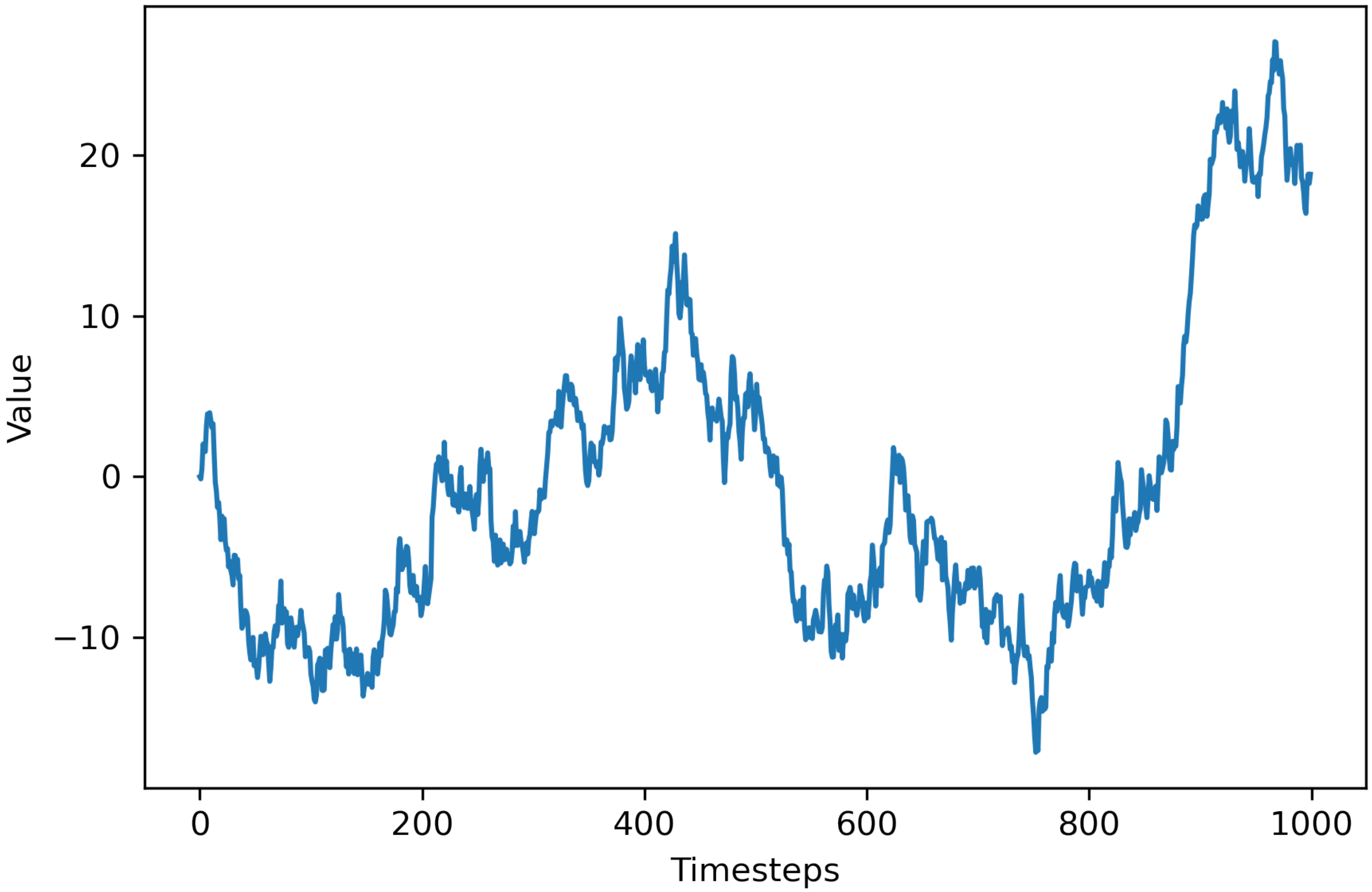
Random Walk

- A random walk is a process in which there is an equal chance of going up or down by a random number
- Random walks often expose long periods where a positive or negative trend can be observed. They are also often accompanied by sudden changes in direction

A random walk can be mathematically express with the following equation

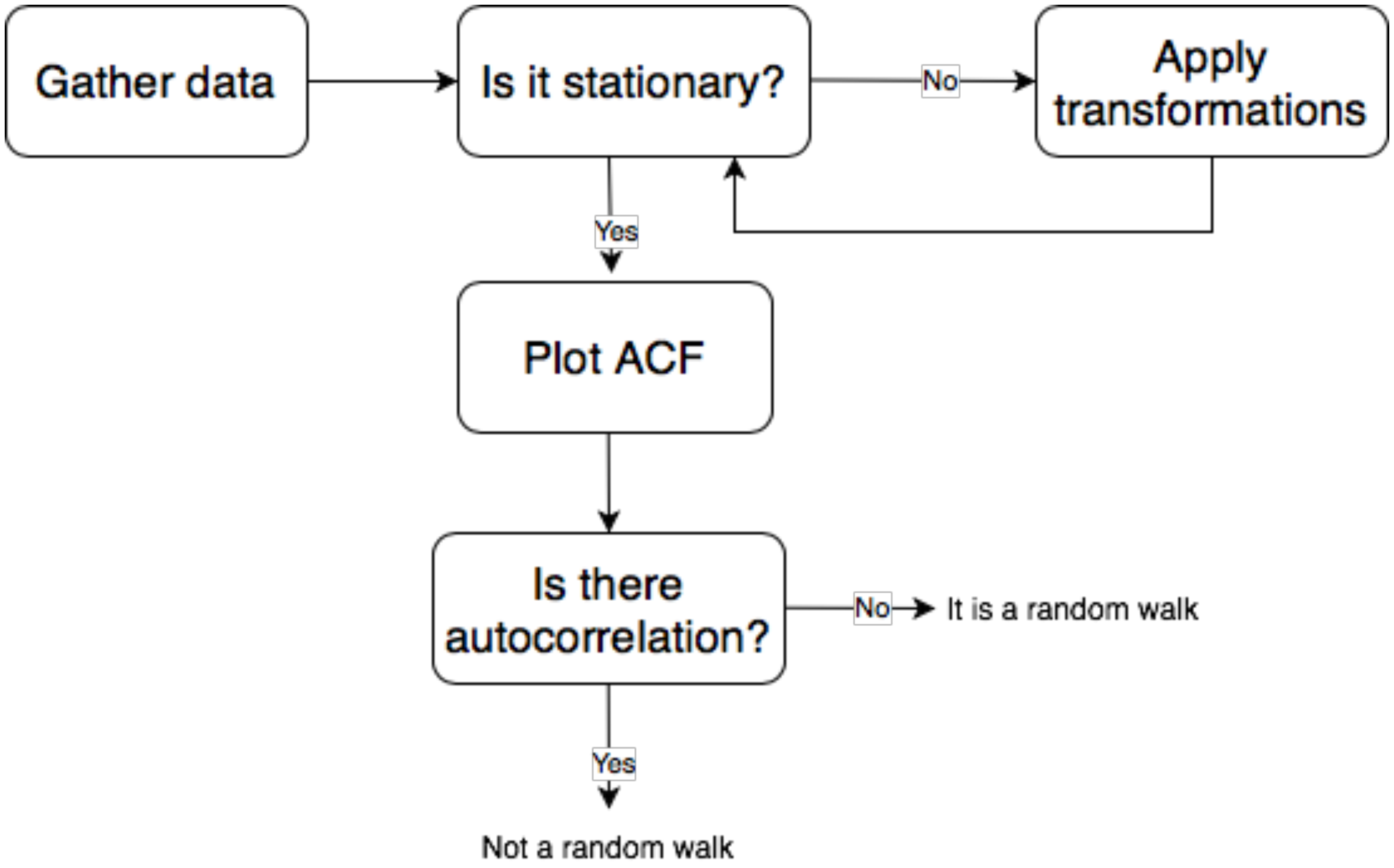
$$y_t = C + y_{t-1} + \epsilon_t$$

Random walk example



Identifying a Random Walk

- A random walk is a series whose first difference is **stationary** and **uncorrelated**



Stationarity

Before applying any statistical model on a time series, the series has to be **stationary**, which means that, over different time periods:

1. it should have constant mean
2. It should have constant variance or standard deviation
3. Autocorrelation do not change on time

How to make the time series stationary

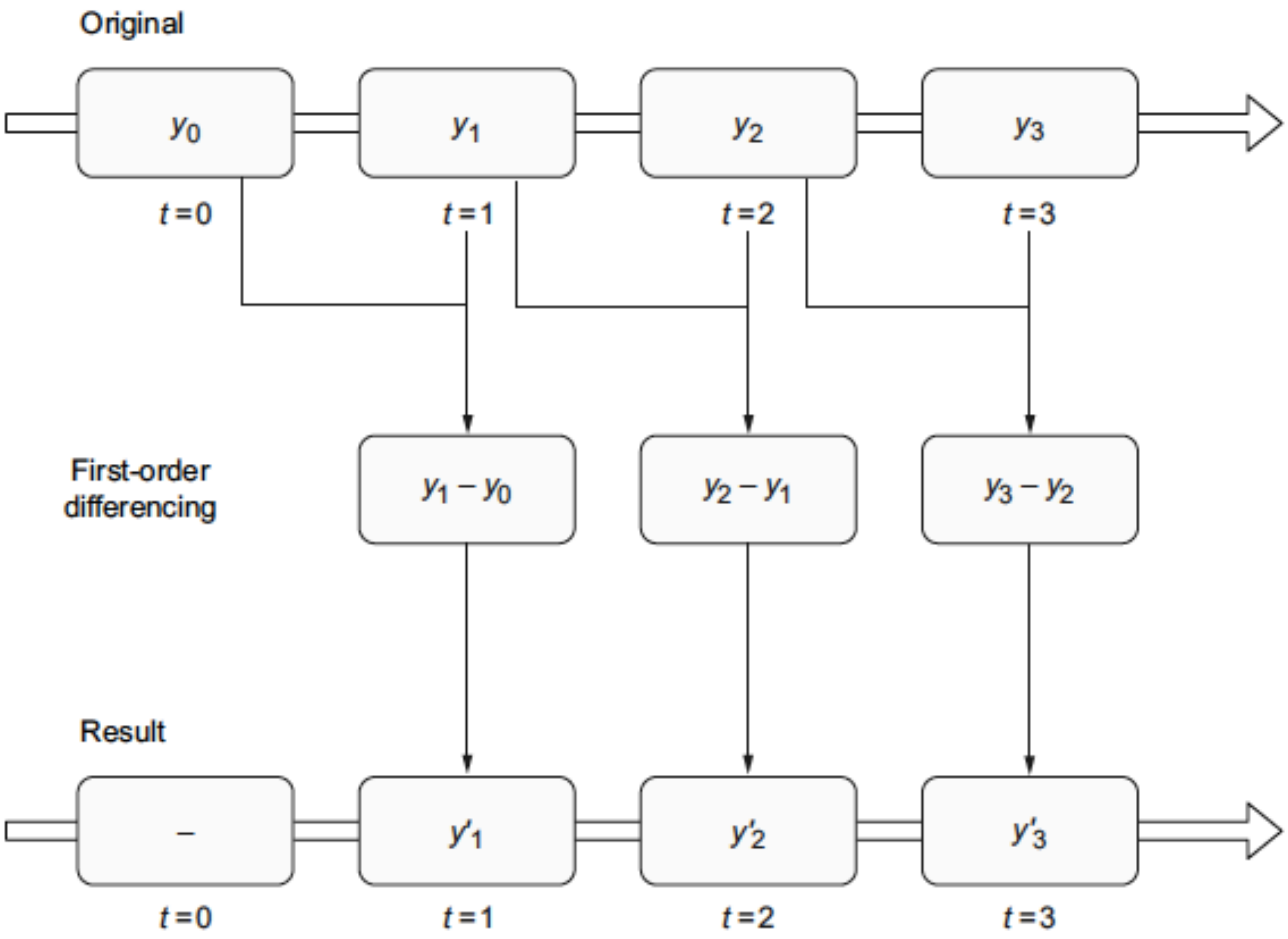
- **Differencing** is a transformation that calculates the change from one timestep to another
- This transformation helps stabilize the mean, which in turn removes or reduces the trend and seasonality effects
- Differencing involves calculating the series of changes from one timestep to another

$$y'_t = y_t - y_{t-1}$$

It is possible to difference a time series many times:

- taking the difference once is applying a **first-order** differencing
- taking it a second time would be a **second-order** differencing

Visualizing a first-order difference



Test to check if a series is stationary

ADCF Test - **Augmented Dickey–Fuller test**

Null hypothesis: says that the time series is non-stationary

The result of this test is the ADF statistic, which is a negative number. The more negative it is, the stronger the rejection of the null hypothesis

If the p-value is less than 0.05, we can also reject the null hypothesis and say the series is stationary

ADCF Test to check if a series is stationary

```python
from statsmodels.tsa.stattools import adfuller

ADF_result = adfuller(random_walk)

print('ADF Statistic:', round(ADF_result[0],3))
print('p-value:', round(ADF_result[1],3))
```

```
ADF Statistic: -0.966
p-value: 0.765
```

Since the series is not stationary a first-order differencing will be applied

```python
diff_random_walk = np.diff(random_walk, n=1)
```

```python
ADF_result = adfuller(diff_random_walk)

print('ADF Statistic:', round(ADF_result[0],3))
print('p-value:', round(ADF_result[1],3))
```

```
ADF Statistic: -31.789
p-value: 0.0
```

Autocorrelation function (ACF)

The autocorrelation function (ACF) measures the linear relationship between lagged values of a time series

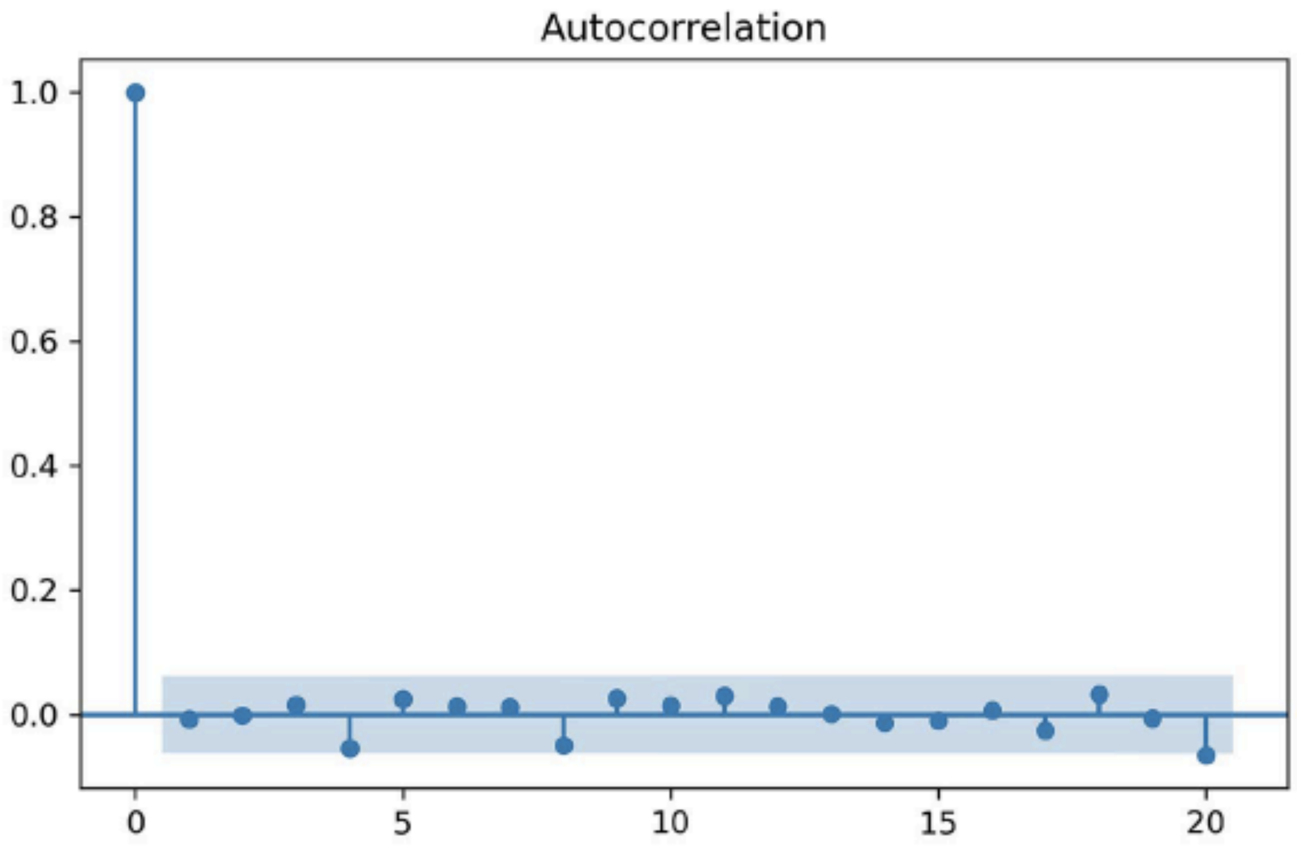
It measures the correlation of the time series with itself

The ACF calculates the autocorrelation coefficient between:

$$y_t \; and \; y_{t-1} : r_1$$
$$y_t \; and \; y_{t-2} : r_2$$
$$\dots$$

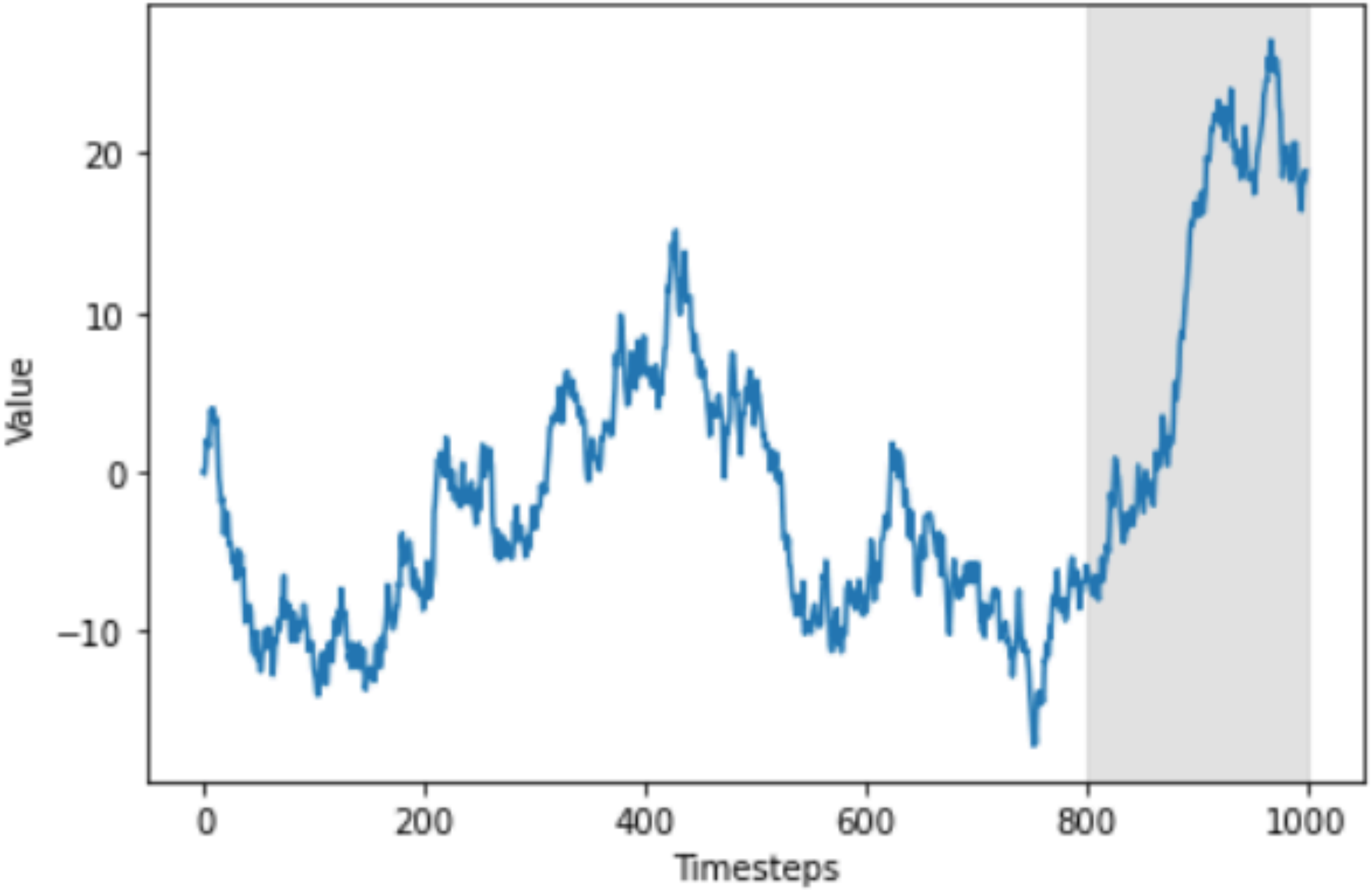
In the ACF plot the coefficient is the dependent variable, while the lag is the independent variable

ACF plot of the random walk



There are no significant coefficients after lag 0, which is a clear indicator of a **random walk** - can be described as **white noise**

Forecasting a random walk



Forecasting a random walk

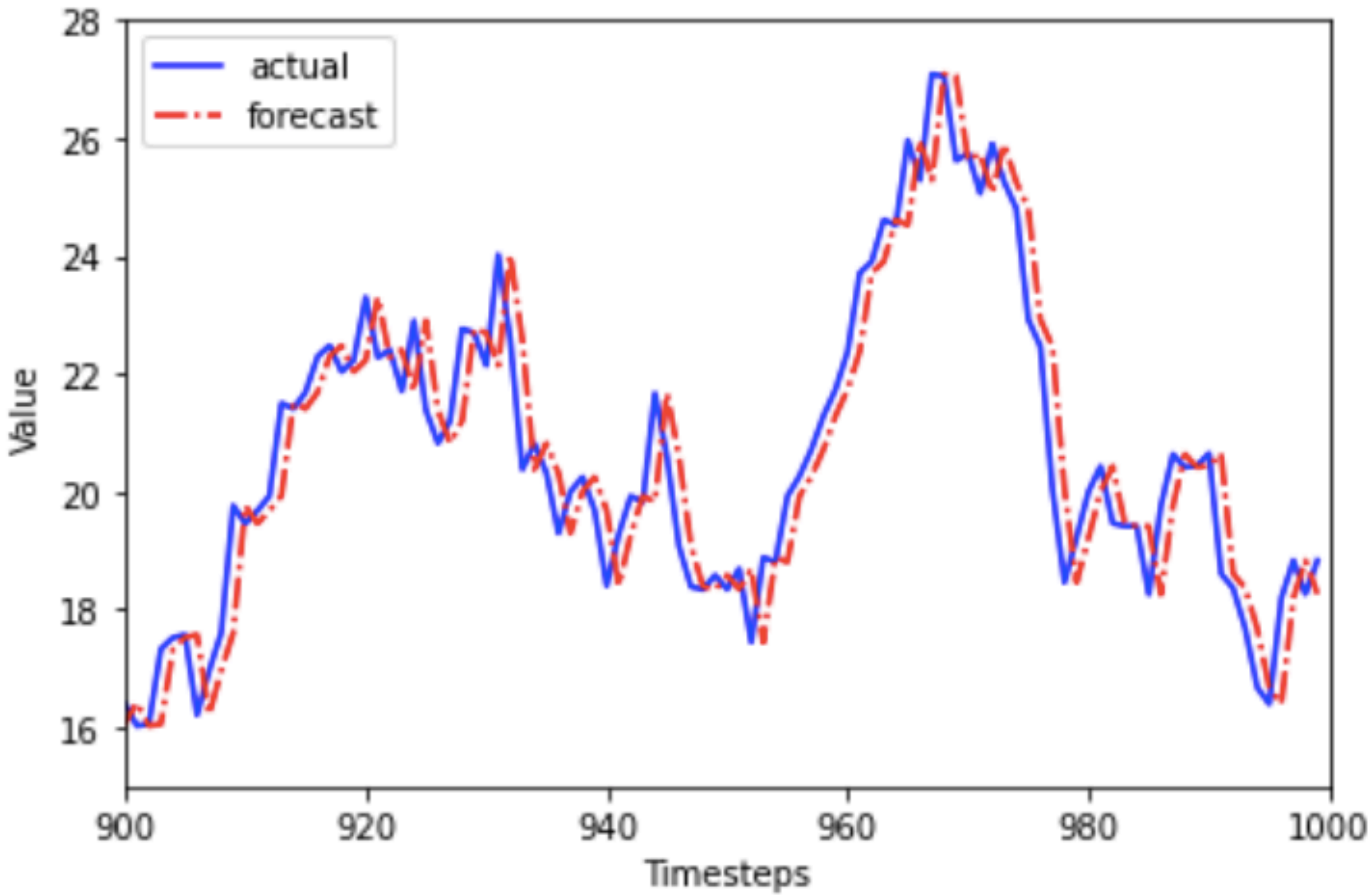
- To predict a random walk, we can only use **naive forecasting** methods or baseline methods

- Forecasting a random walk on a **long horizon** does not make sense — the randomness portion is magnified in a long horizon where many random numbers are added over the course of many timesteps

- In a random walk, it is only possible to forecast the **next timestep**

- The present observed value is used as a forecast for the next timestep. Once a new value is recorded, it will be used as a forecast for the following timestep

Forecasting a random walk




```
In [39]: mse_one_step = mean_squared_error(test['value'], df_shift[800:])

         mse_one_step

Out[39]: 0.9256876651440581
```

Forecasting with statistical models

Statistical models for time series forecasting

- MA(q) models
- AR(p) models
- ARMA(p,q) models
- ARIMA(p,d,q) models for non-stationary time series
- SARIMA(p,d,q)(P,D,Q)$_m$ for seasonal time series
- SARIMAX models to include external variables in the forecast
- VAR(p) model for predicting many time series at once

Stationary time series

Moving Average model
MA(q)

Moving Average process

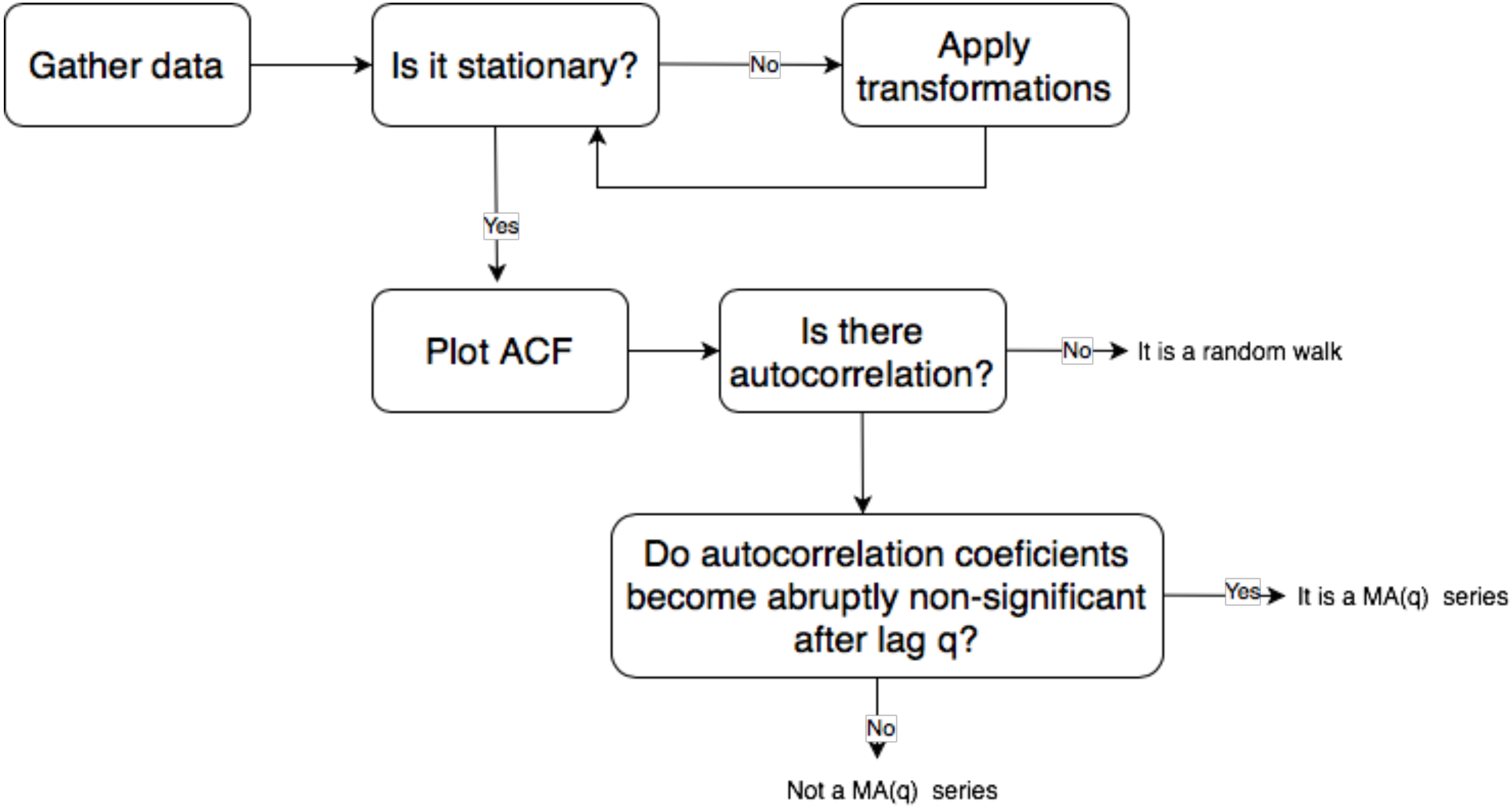
- In a moving average (MA) process, the current value depends linearly on the mean of the series, the current error term, and past error terms
- The moving average model is denoted as MA(q), where q is the order

The general expression of an MA(q) model is

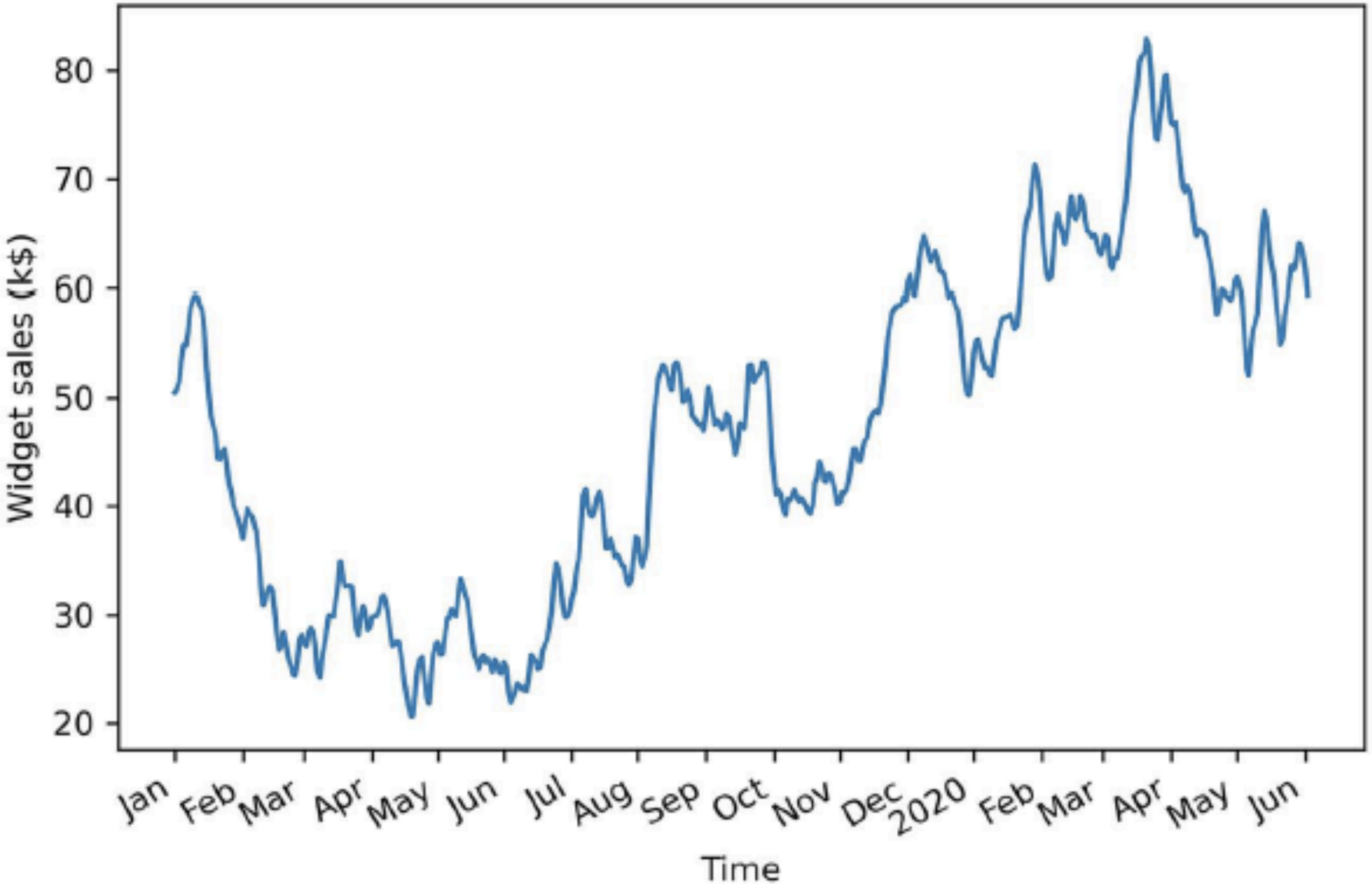
$$y_t = \mu + e_t + \theta_1 e_{t-1} + \theta_2 e_{t-2} + .. + \theta_q e_{t-q}$$

The order q of the moving average model determines the number of past error terms that affect the present value

Identifying a Moving Average series



Moving Average series example



Volume of sales for a Company over 500 days, starting on January 1, 2019

Identifying the order of a Moving Average series

Test for stationarity

```python
# Test for stationarity

from statsmodels.tsa.stattools import adfuller

ADF_result = adfuller(df['widget_sales'])

print(f'ADF Statistic: {ADF_result[0]}')
print(f'p-value: {ADF_result[1]}')
```

```
ADF Statistic: -1.5121662069359012
p-value: 0.5274845352272624
```

```python
# first-order differencing to make it stationary

widget_sales_diff = np.diff(df['widget_sales'], n=1)
```

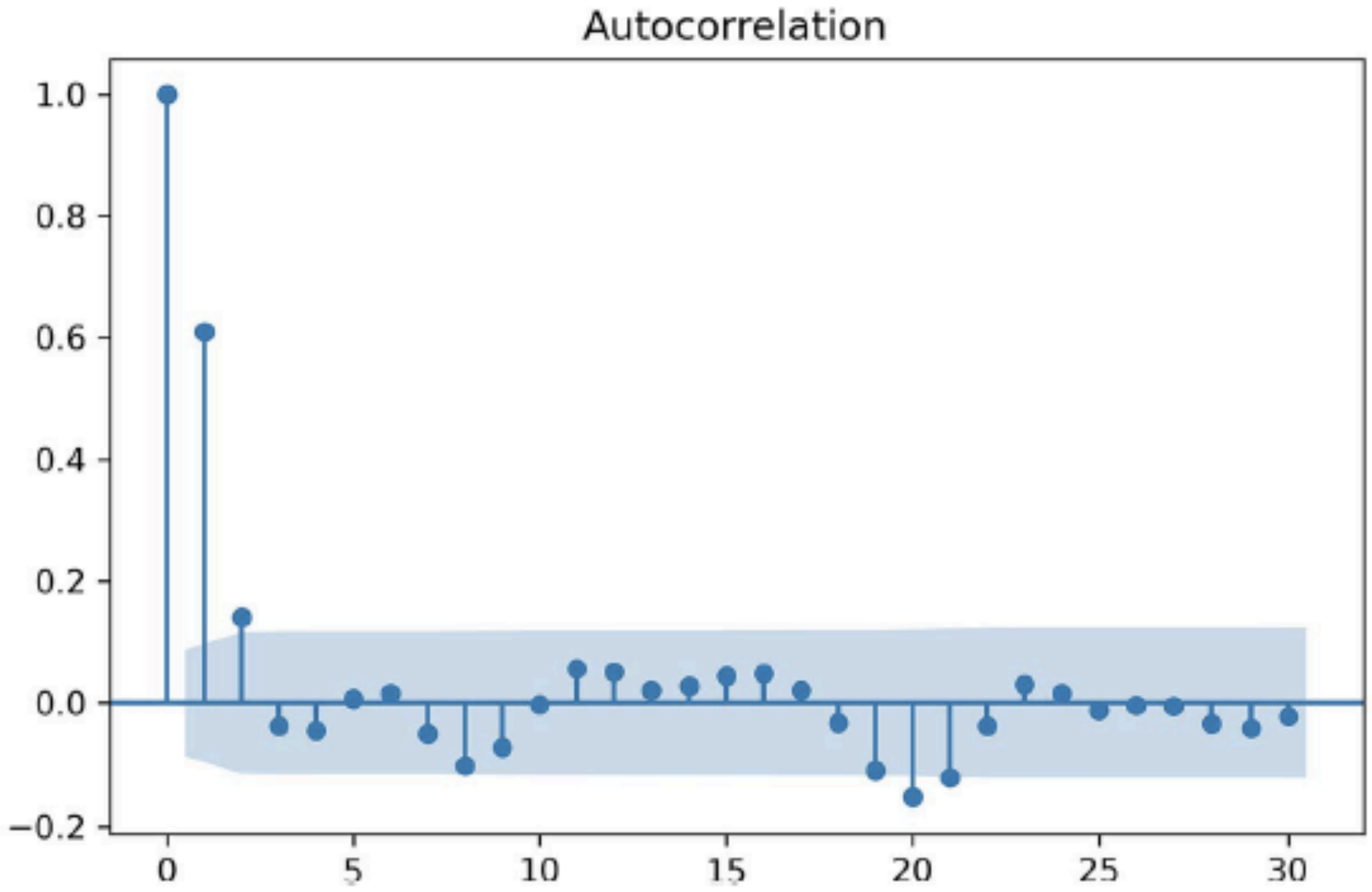
```python
ADF_result = adfuller(widget_sales_diff)

print(f'ADF Statistic: {ADF_result[0]}')
print(f'p-value: {ADF_result[1]}')
```

```
ADF Statistic: -10.576657780341957
p-value: 7.076922818587346e-19
```

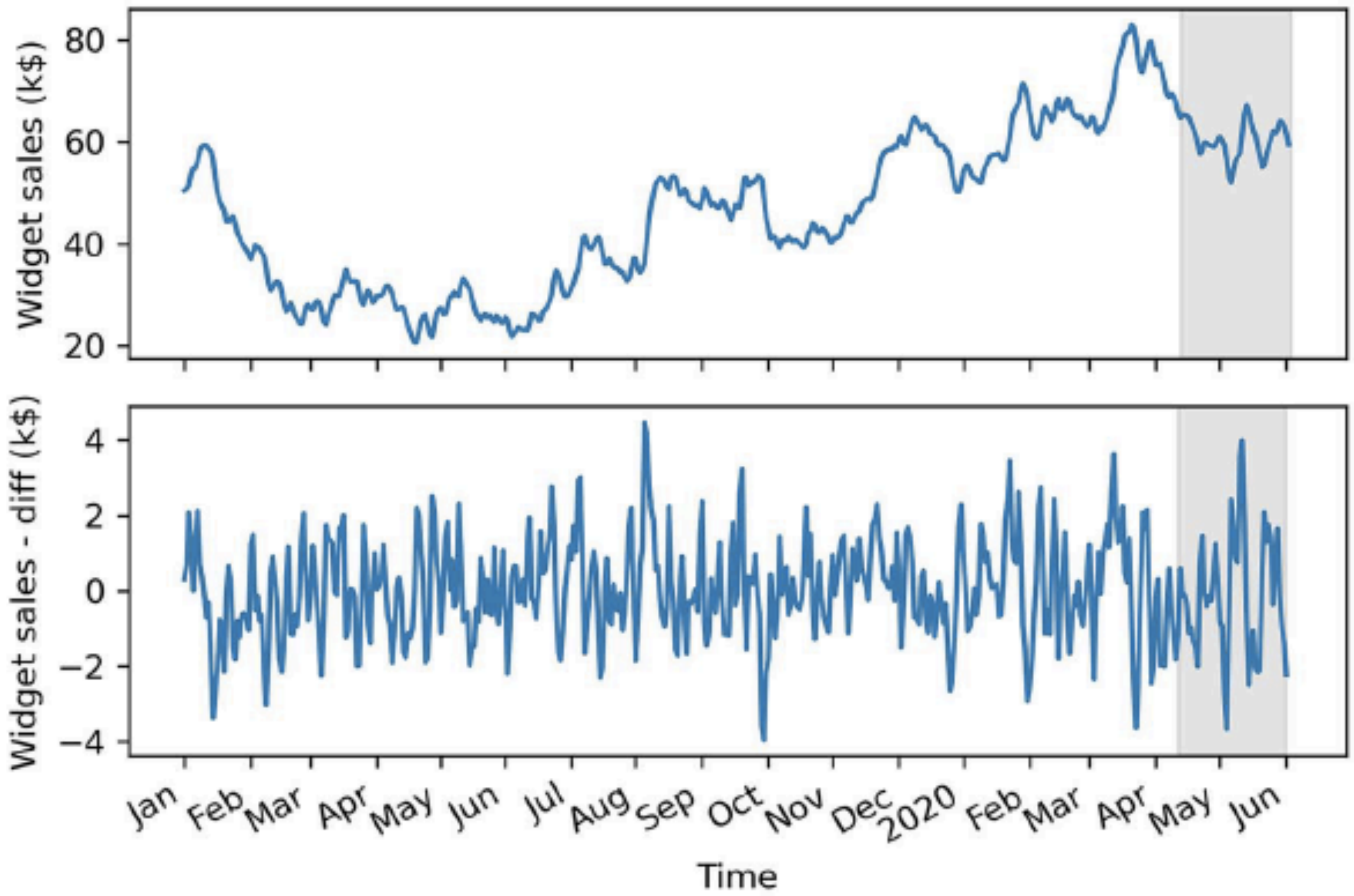
Identifying the order of a Moving Average series

Plot the autocorrelation function - ACF



Stationary moving average series of order 2 - MA(2)

Forecasting a Moving Average series



Forecasting using the MA(q) model

- When using an MA(q) model, forecasting **beyond q steps** into the future will simply return the mean - there are no error terms to estimate beyond q steps

- It is possible to use ***rolling forecasts*** to predict **up to q steps** at a time

- In a dataset with 500 steps, to predict the last 50 steps:
  - First pass: train on the first 449 timesteps to predict timesteps 450 and 451
  - Second pass: train on the first 451 timesteps to predict timesteps 452 and 453
  - …
  - This is repeated until the values at timesteps 498 and 499 are predicted

A function for rolling forecasts on a horizon

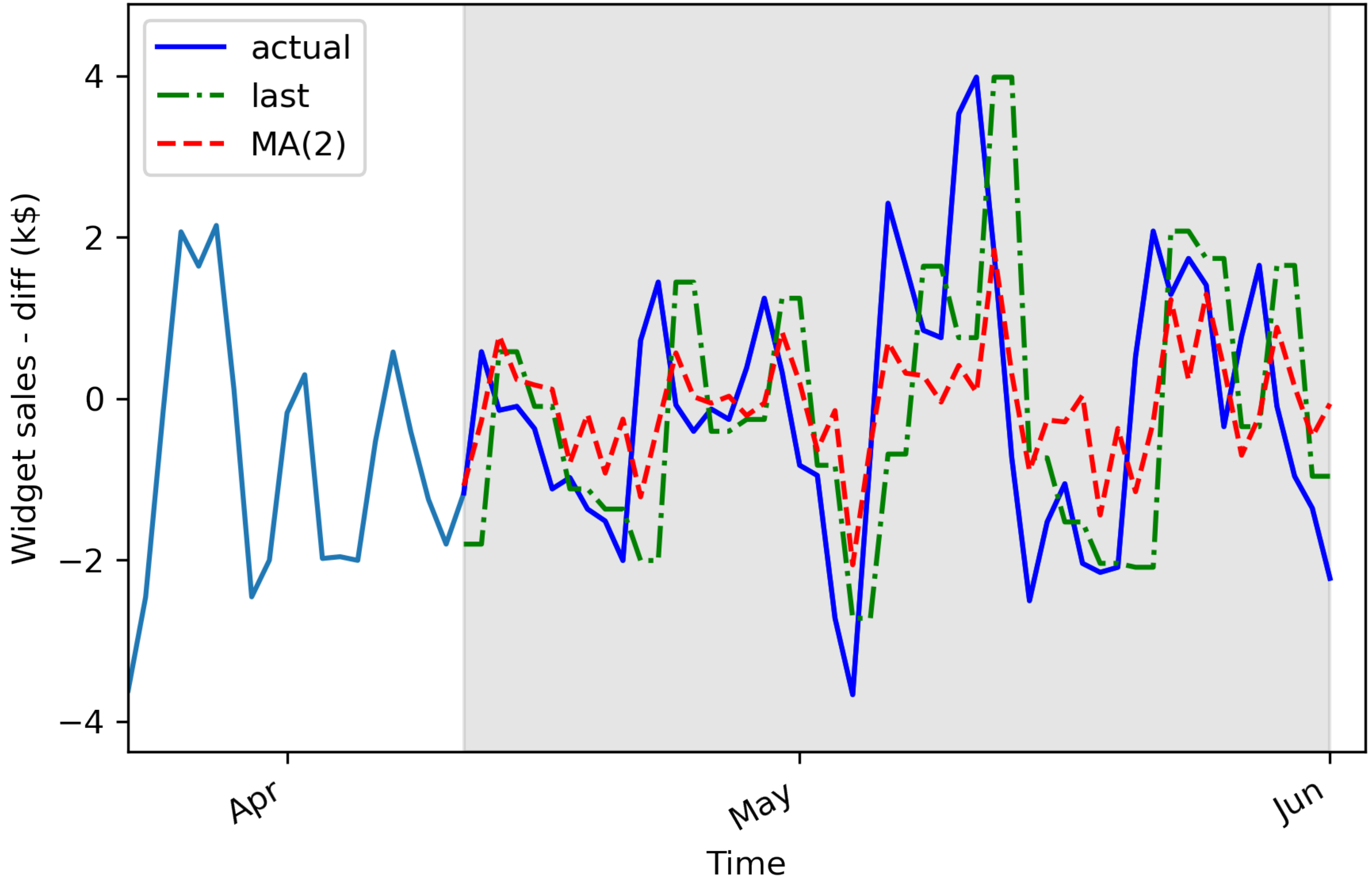
```
from statsmodels.tsa.statespace.sarimax import SARIMAX

def rolling_forecast(df: pd.DataFrame, train_len: int, horizon: int, window: int) -> list:
   total_len = train_len + horizon
   pred_MA = []

   for i in range(train_len, total_len, window):
       model = SARIMAX(df[:i], order=(0,0,2))
       res = model.fit(disp=False)
       predictions = res.get_prediction(0, i + window - 1)
       oos_pred = predictions.predicted_mean.iloc[-window:]
       pred_MA.extend(oos_pred)

    return pred_MA
```

Forecasting the MA(2) series



MSE last value: 3.249
MSE MA(2): 1.948

Differencing to obtain the series to the original scale

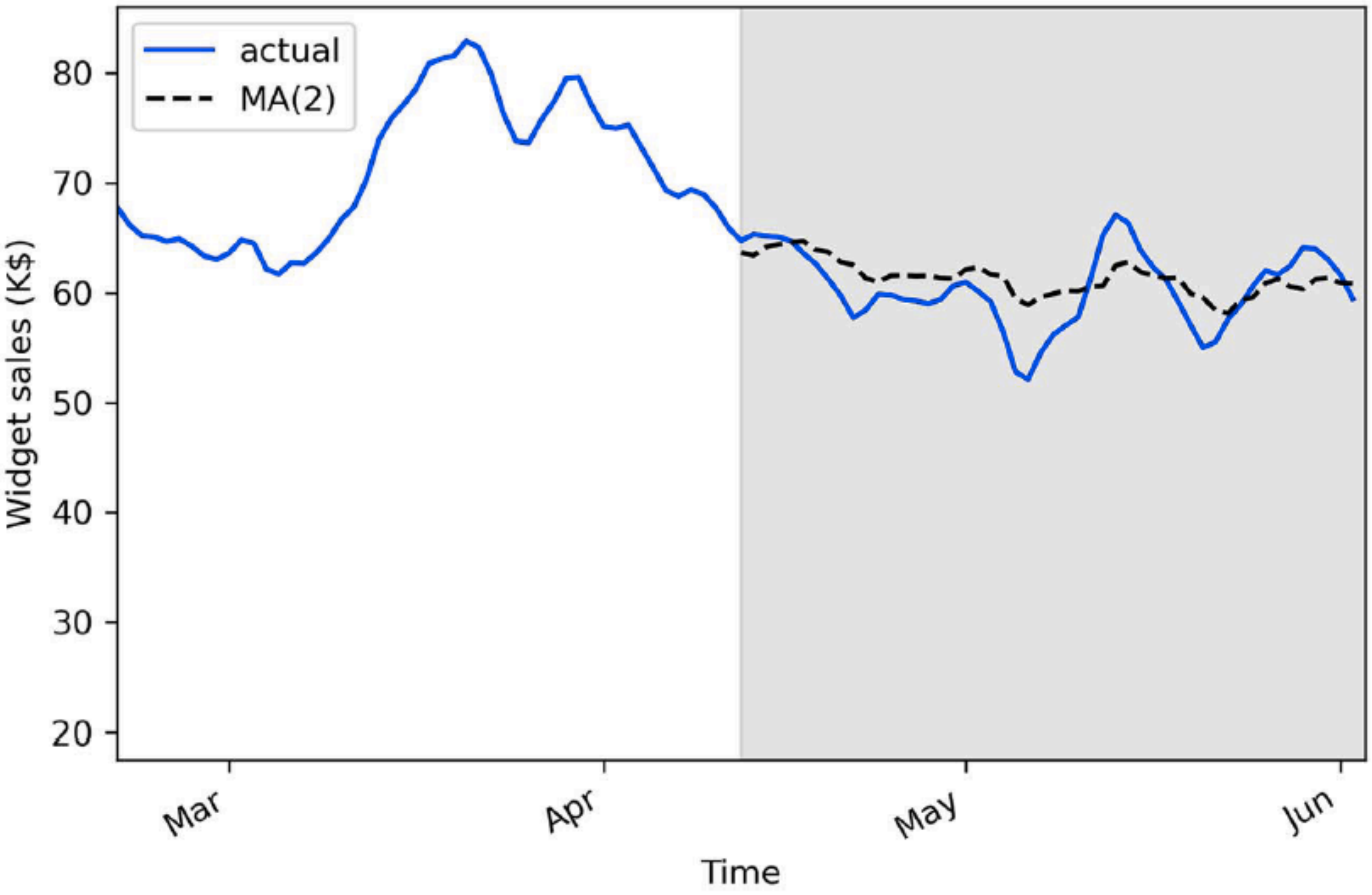
In order to reverse the first-order difference it is necessary to add an initial value y_0 to the first differenced value y_1'

$$y_1 = y_0 + y_1'$$

Then y_2 can be obtained using a cumulative sum of the differenced values

$$y_2 = y_0 + y_1' + y_2'$$

Inverse-transformed MA(2) series



MAE MA(2): 2.32

Autoregressive model AR(p)

Autoregressive process

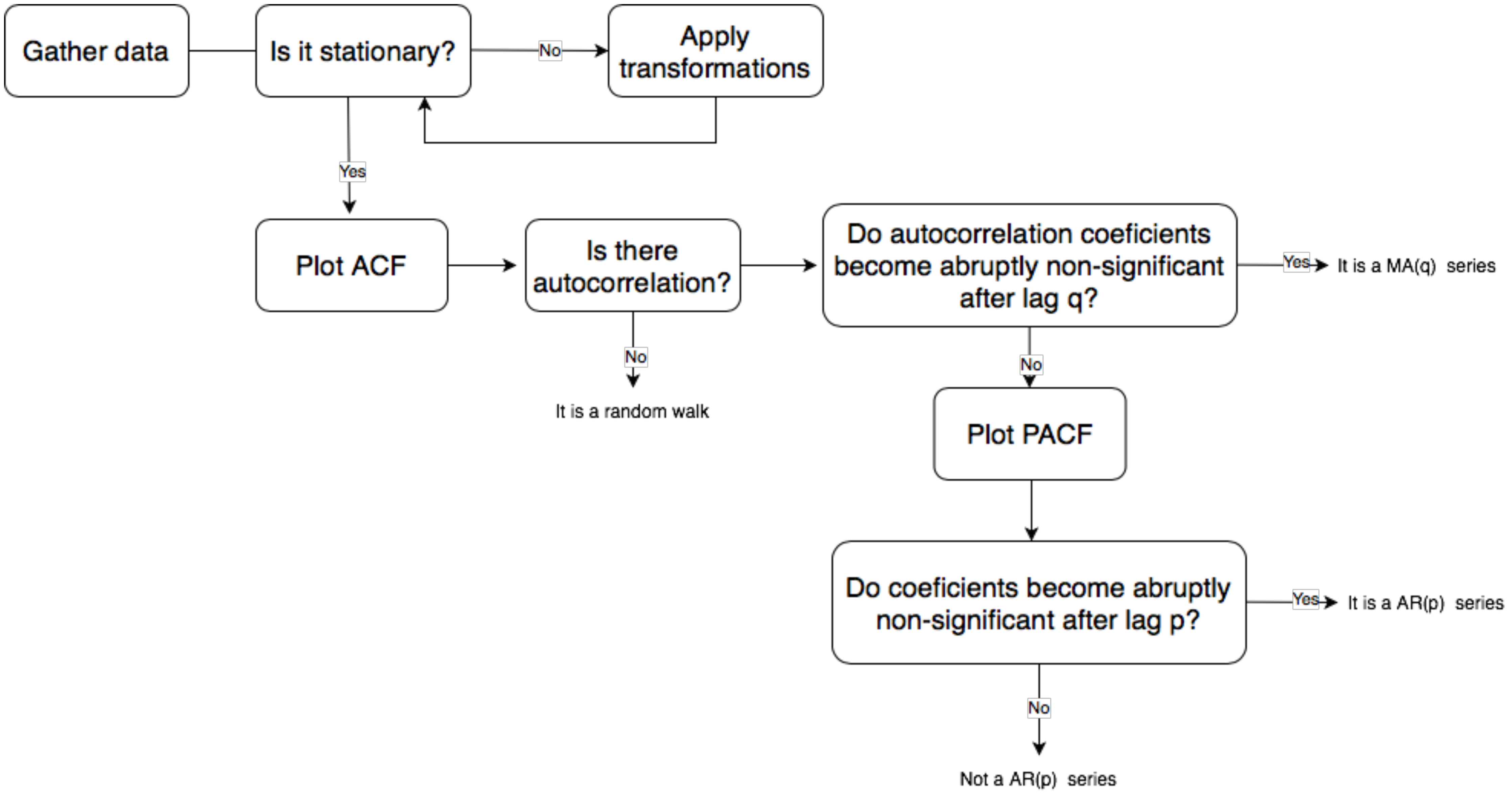
- An autoregressive process is a regression of a variable against itself. In a time series, this means that the present value is linearly dependent on its past values
- The autoregressive process is denoted as AR(p), where p is the order

The general expression of an AR(p) model is

$$y_t = C + \phi_1 y_{t-1} + \phi_2 y_{t-2} + .. + \phi_p y_{t-p} + \epsilon_t$$

Similar to the MA(q) series, the order p of an autoregressive process determines the number of past values that affect the present value.

Identifying a Autoregressive series



Partial Autocorrelation function (PACF)

In a second-order autoregressive series or AR(2)

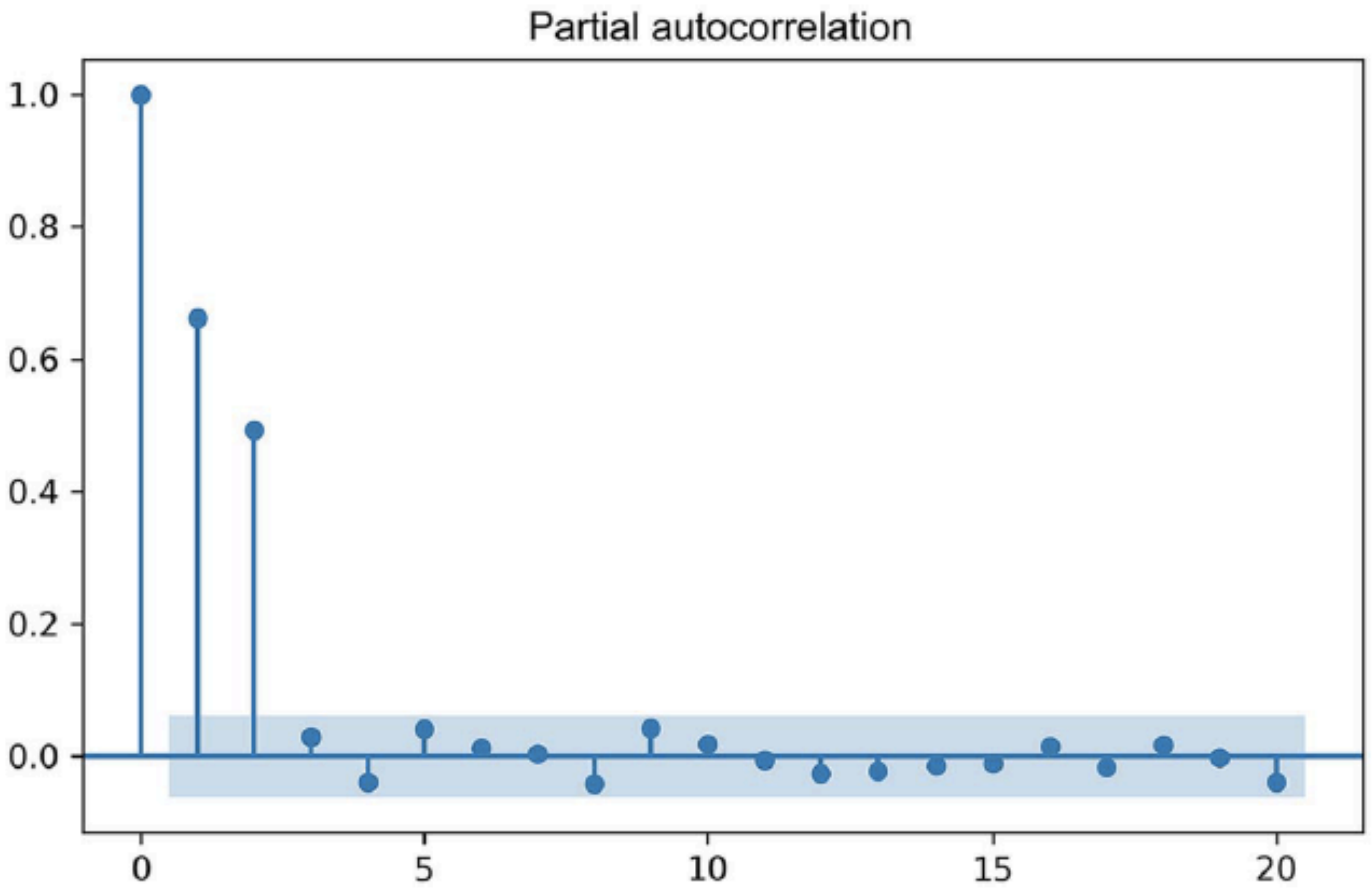
$$y_t = C + \phi_1 y_{t-1} + \phi_2 y_{t-2} + \epsilon_t$$

The autocorrelation between yt and yt-2 using the ACF does not take into account the fact that y_{t-1} has an influence on both y_t and y_{t-2}

To do so, it is necessary to remove the effect of y_{t-1}. Thus, measuring the **partial autocorrelation** between y_t and y_{t-2}

The partial autocorrelation function measures the correlation between lagged values in a time series when the influence of correlated lagged values in between are removed

PACF plot of a AR(2) series



The partial autocorrelation function can be used to determine the order of a stationary AR(p) series - the coefficients will be non-significant after lag p

Autoregressive moving average model
ARMA(p,q)

Autoregressive moving average series

- An autoregressive moving average series is a combination of the autoregressive and the moving average series
- It is denoted as ARMA(p,q), where p is the order of the autoregressive process, and q is the order of the moving average process

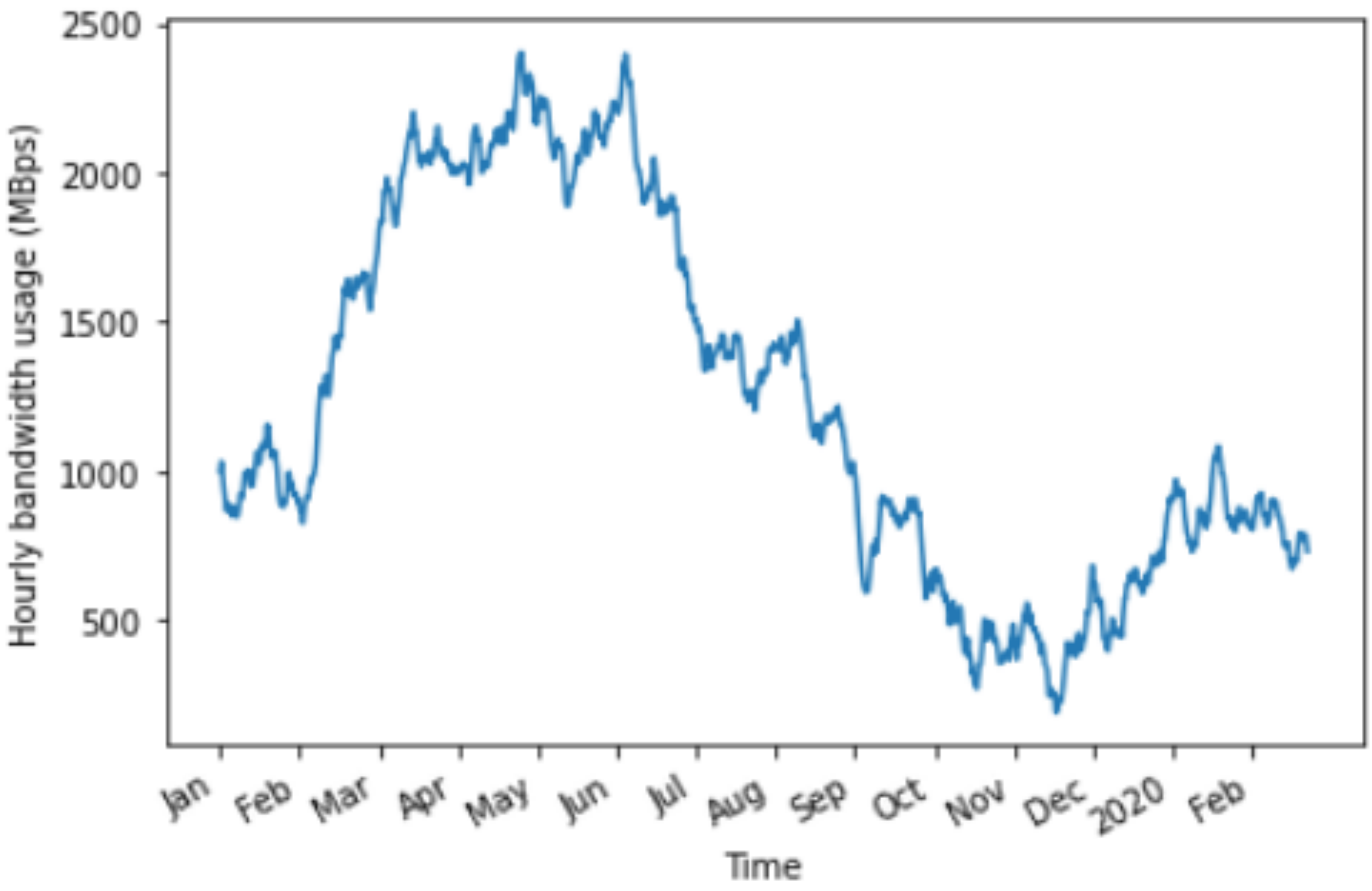
The general equation of the ARMA(p,q)

$$y_t = C + \phi_1 y_{t-1} + \phi_2 y_{t-2} + .. + \phi_p y_{t-p} + \mu + \epsilon_t + \theta_1 e_{t-1} + \theta_2 e_{t-2} + .. + \theta_q e_{t-q}$$

ARMA(0,q) ≈ MA(q), since the order p = 0 cancels the AR(p) portion

ARMA(p,0) ≈ AR(p), since the order q = 0 cancels the MA(q) portion

Autoregressive moving average series example



Bandwidth usage for a large data center in bits per second (bps)

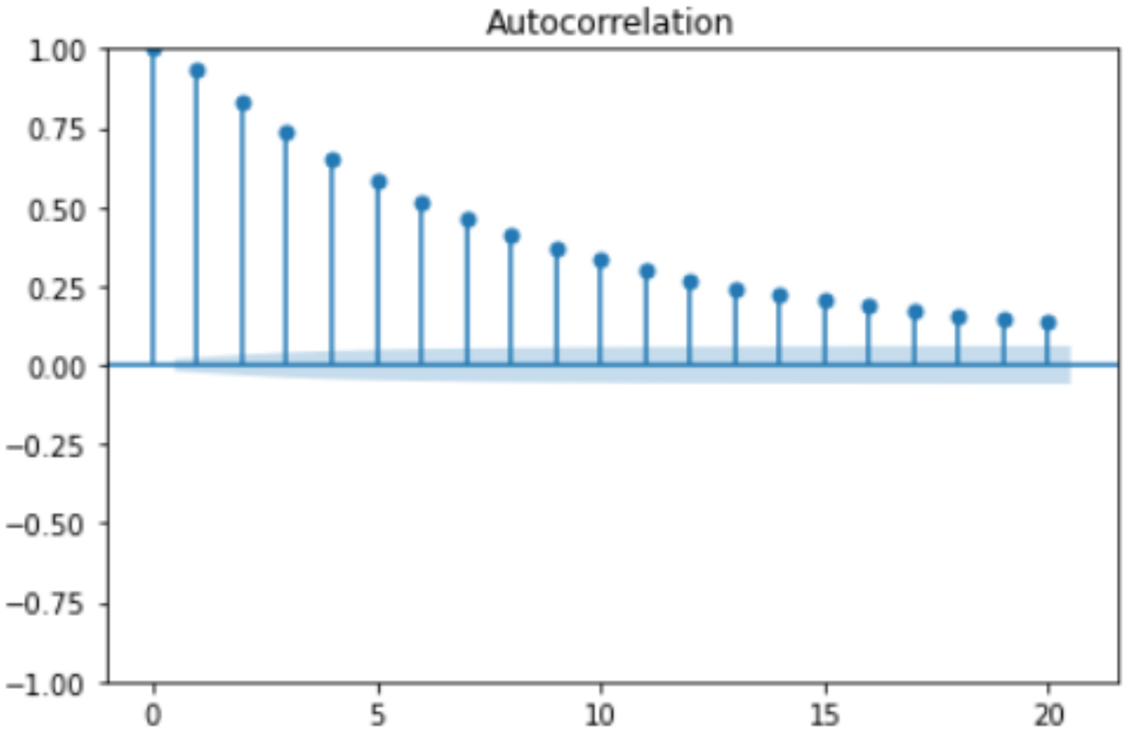
Identifying a stationary ARMA series

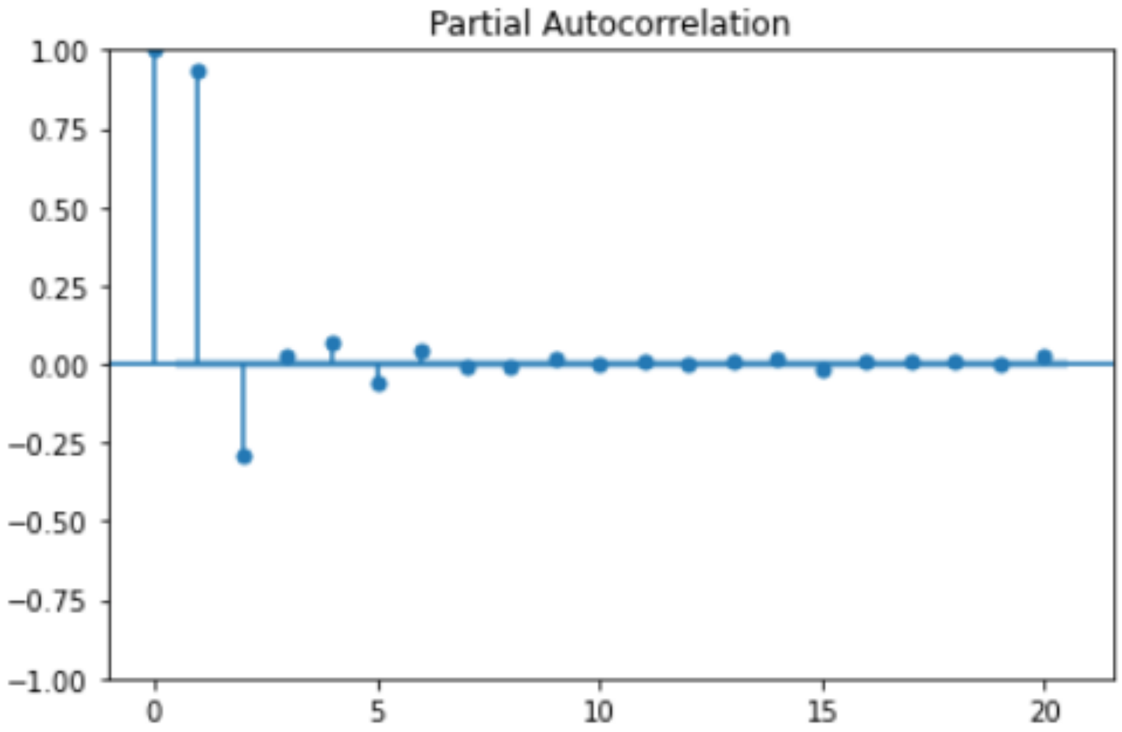
- If the series is a stationary ARMA(p,q) process both the ACF and PACF plots show a decaying or sinusoidal pattern

- The ACF and PACF plots cannot be used to determine the orders q and p of an ARMA(p,q) process

- The solution is to fit many ARMA(p,q) models with various combinations of values for p and q, then choosing a model using the **Akaike information criteria**

ACF and PACF plots





Akaike information criterion (AIC)

- The AIC calculates a model's quality in comparison to other models. It is used for model selection

The AIC is a function of the number of parameters k in a model and the maximum value of the likelihood function $\hat{L}$:

$$AIC = 2k - 2\ln(\hat{L})$$

- The AIC quantifies the relative amount of information lost by the model
- The better the model, the lower the AIC value and the less information is lost

Function to fit several ARMA(p,q) models

```
def optimize_ARMA(data, order_list) -> pd.DataFrame:

    results = []

    for order in order_list:
        try:
            model = SARIMAX(data, order=(order[0], 0, order[1]), simple_differencing=False)
        except:
            continue

        aic = model.aic
        results.append([order, aic])

    result_df = pd.DataFrame(results)
    result_df.columns = ['(p,q)', 'AIC']

    #Sort in ascending order, lower AIC is better
    result_df = result_df.sort_values(by='AIC', ascending=True).reset_index(drop=True)

    return result_df
```

Residual analysis

The residuals of a model are simply the difference between the predicted values and the actual values

If the model has captured all predictive information from a dataset, the residuals of the model are **white noise**; there is only a random fluctuation left that cannot be modelled

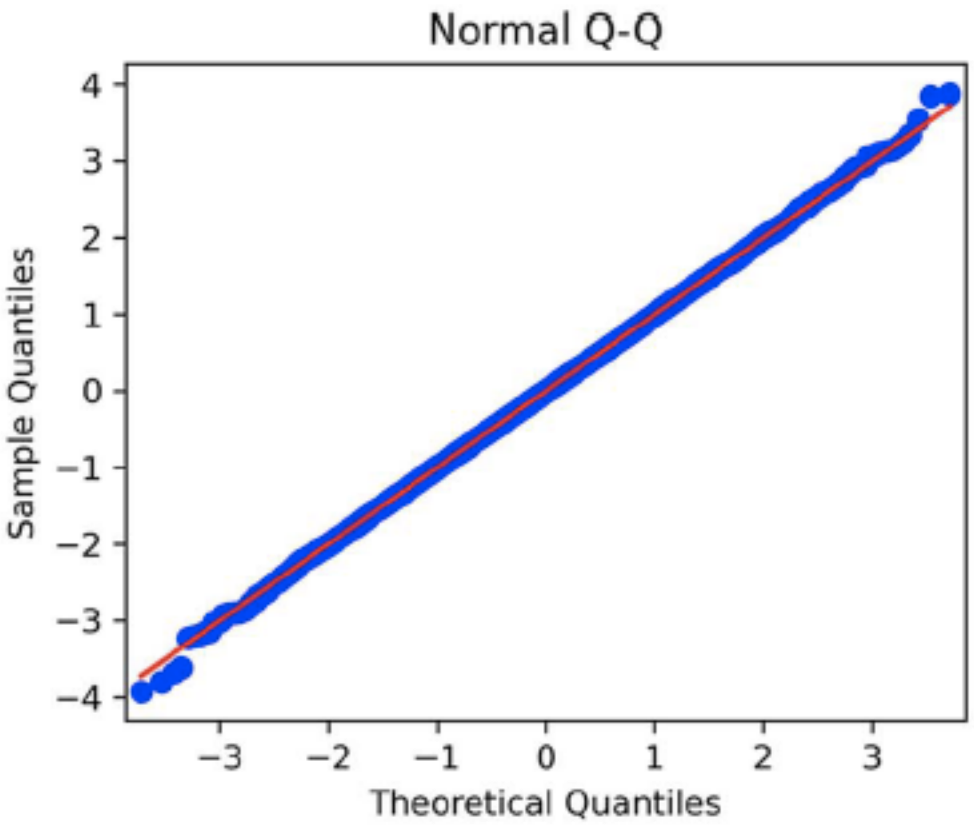
To have a good model for making forecasts, the **residuals** must be **uncorrelated** and have a **normal distribution**

Two residual analysis

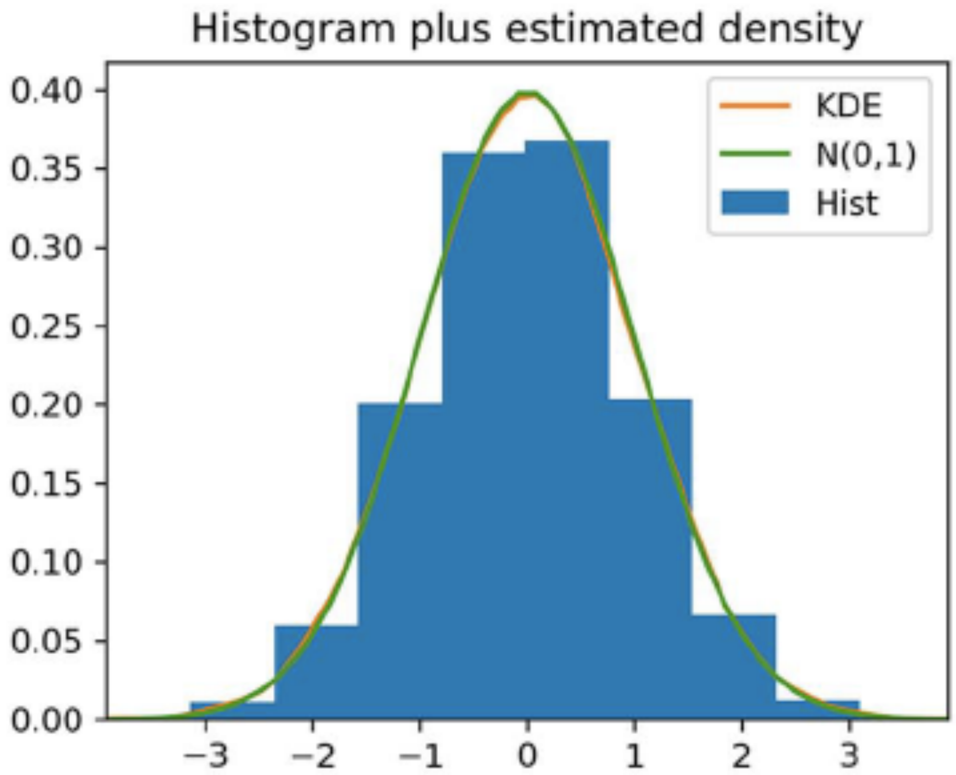
- A qualitative analysis through the study of the ***quantile-quantile plot*** *(*Q-Q plot), for verifying if the model's residuals are normally distributed

- A quantitative analysis applying the ***Ljung-Box test*** to demonstrate that the residuals are uncorrelated

Two residual analysis: quantile-quantile plot



Q-Q plot of the residuals



Histogram of the residuals

Two residual analysis: Ljung-Box test

```
from statsmodels.stats.diagnostic import acorr_ljungbox

# run the Ljung-Box test on the residuals for the first 10 lags

residuals = model_fit.resid

residuals_test = acorr_ljungbox(residuals, np.arange(1, 11, 1))

residuals_test['lb_pvalue'].describe()
```

```
count    10.000000
mean      0.923847
std       0.057180
min       0.811247
25%       0.915579
50%       0.942076
75%       0.961415
max       0.981019
Name: lb_pvalue, dtype: float64
```

All the returned p-values exceed 0.05, the residuals are uncorrelated