ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

# Python for IoT Data Analytics

## Time-series Forecasting

**EIT4SEMM PHD Program, July 2023**


Prof. Marco Di Felice

Department of Computer Science and Engineering, University of Bologna

marco.difelice3@unibo.it

# IoT Sensor Data Forecasting

For ease of disposition, we distinguish between:

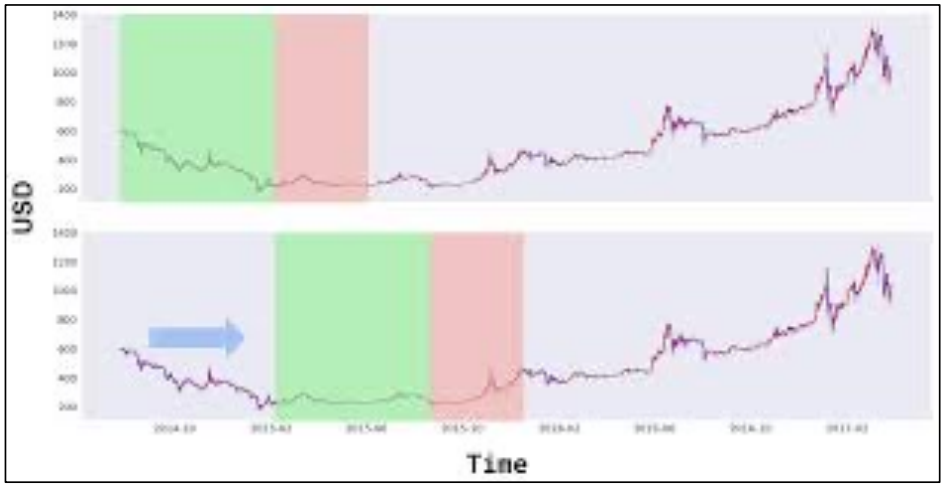
✧ **Time-series Processing** → Techniques and tools to manipulate the time-series (e.g. data aggregation).

✧**Time-series Analytics** → Techniques and tools to extract information from the time-series (e.g. forecasting).

NOTE. In several use-cases, processing and analytics are combined together (e.g. first aggregate/filter the time-series, then apply learning techniques).

# IoT Sensor Data Forecasting

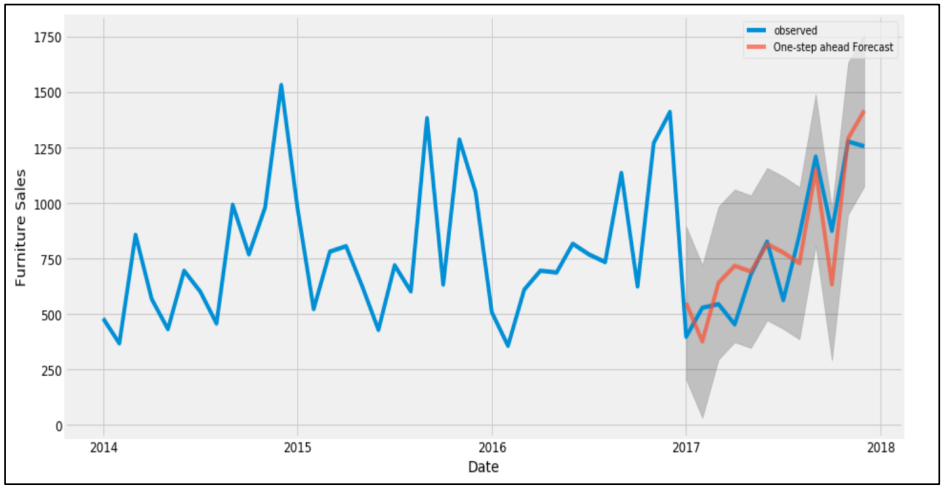
Source: https://www.mdpi.com/1911-8074/12/1/17/htm




TIME-SERIES CLASSIFICATION

- Assign a label to a time-series or to a subset

Source: https://towardsdatascience.com/an-end-to-end-project-on-time-series-analysis-and-forecasting-with-python-4835e6bf050b




TIME-SERIES FORECASTING

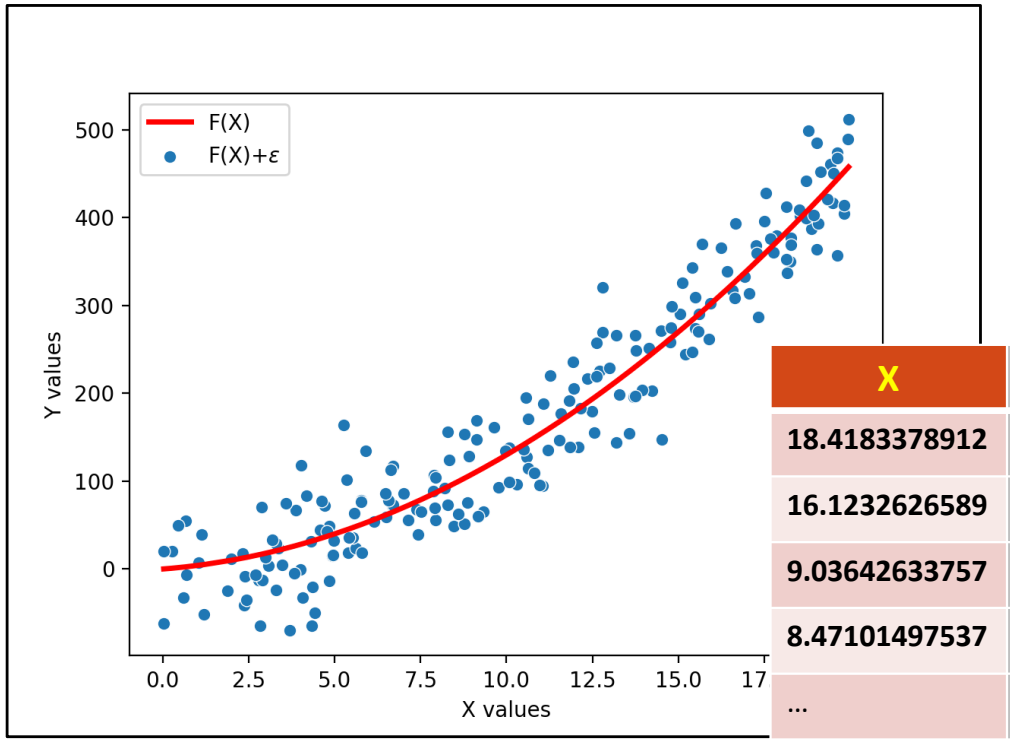
- Predict future values based on previously observed values.

# IoT Sensor Data Forecasting

**PREDICTION/FORECASTING**

- Given a quantitative response $y$ and $p$ different variables $(x_1, x_2, \ldots, x_p)$
- Find the relationship between $y$ and $\boldsymbol{x} = (x_1, x_2, \ldots, x_p)$: $y = f(\boldsymbol{x}) + \epsilon$
- $\epsilon$ is a random error term, which is independent of $\boldsymbol{x}$ and has mean zero.
- The function $f$ that connects the input variable to the output variable is **unknown**
- Learn the function $\hat{y} = \hat{f}(x)$, where $\hat{f}$ represents our estimate for $f$, and $\hat{y}$ represents the resulting prediction for $y$

# IoT Sensor Data Forecasting




$$Y = F(X) + \epsilon$$

$$Y = X^2 + 3X + \mathcal{N}(0,40)$$

DATASET

| X | F(X) | $\epsilon$ | Y=F(X) +$\epsilon$ |
|---|---|---|---|
| **18.4183378912** | 394.490184350 | 37.5662180915 | **356.923966259** |
| **16.1232626589** | 308.329386744 | -2.87868760592 | **305.450699138** |
| **9.03642633757** | 108.766279967 | 51.0748597503 | **159.841139717** |
| **8.47101497537** | 97.1711396392 | -42.7643064745 | **54.40683316466** |
| … | … | … | … |

# IoT Sensor Data Forecasting

**CLASSIFICATION**

- Given a **qualitative** response *y* and *p* different variables ($x_1, x_2, \ldots, x_p$)
- The variable *y* can take values in one of *K* different **classes** or **categories**
- Like in prediction problems, the function *f* that connects the input variable to the output variables is **unknown**
- Learn the function $\hat{y} = \hat{f}(x)$ with $\hat{f} : X \rightarrow K$ **minimizing the error rate**:

$$\frac{1}{n}\sum_{1}^{n} I_i \qquad I_i = \begin{cases} 1 \; if \; (yi \neq \hat{y}_i) \\ 0 \; if \; (yi = \hat{y}_i) \end{cases}$$

# IoT Sensor Data Forecasting

## Time series Components

- **Secular Trend**
  - Smooth long-term direction of a time series
- **Seasonal**
  - Patterns of change in a time series within a period. These patterns tend to repeat themselves each period.
- **Irregular Variations**
  - Further classified as episodic or residual, cannot be predicted

# IoT Sensor Data Forecasting

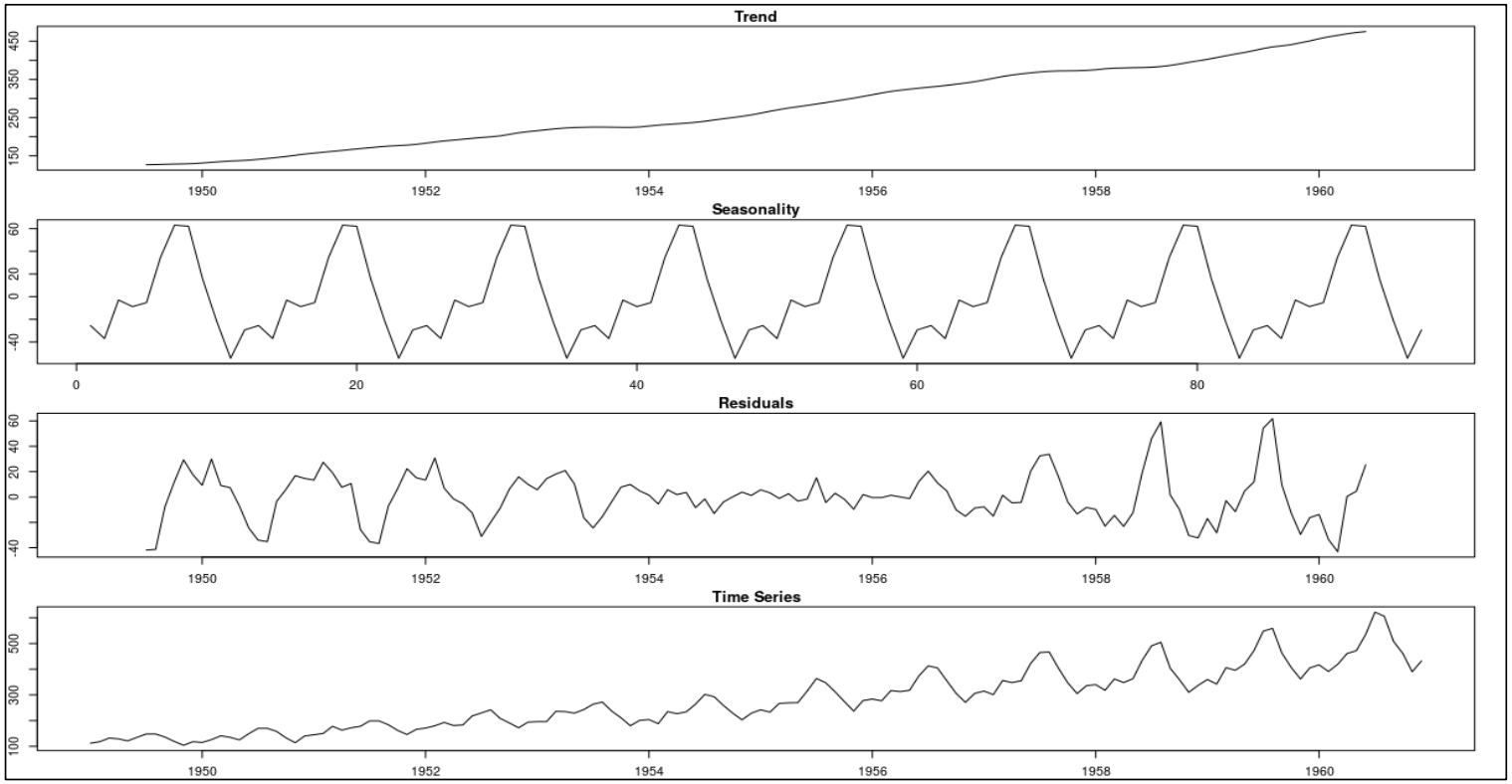
## Single-variate analysis

| Date | Soil Moisture |
|---|---|
| 2023-07-20 08:00:00 | 34.12 |
| 2023-07-20 09:00:00 | 38.12 |
| 2023-07-20 10:00:00 | 37.12 |
| 2023-07-20 11:00:00 | 39.12 |
| 2023-07-20 12:00:00 | 39.45 |
| 2023-07-20 13:00:00 | 60.12 |

## Multi-variate analysis

| Date | Soil Moisture | Precipitation |
|---|---|---|
| 2023-07-20 08:00:00 | 34.12 | 0 |
| 2023-07-20 09:00:00 | 38.12 | 0 |
| 2023-07-20 10:00:00 | 37.12 | 0 |
| 2023-07-20 11:00:00 | 39.12 | 0 |
| 2023-07-20 12:00:00 | 39.45 | 0 |
| 2023-07-20 13:00:00 | 60.12 | 2 |

# IoT Sensor Data Forecasting




TREND

SEASONALITY

RESIDUALS

TIME-SERIES

# IoT Sensor Data Forecasting

ADDITIVE DECOMPOSITION

Observed Series= Trend + Seasonal + Irregular

$$O_t = T_t + S_t + I_t$$

MULTIPLICATIVE DECOMPOSITION

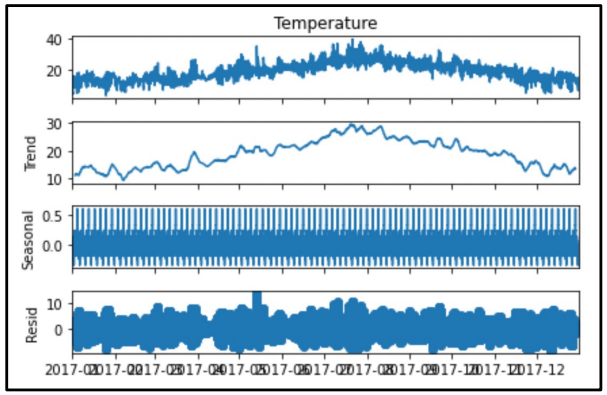
Observed Series= Trend * Seasonal * Irregular

$$O_t = T_t * S_t * I_t$$

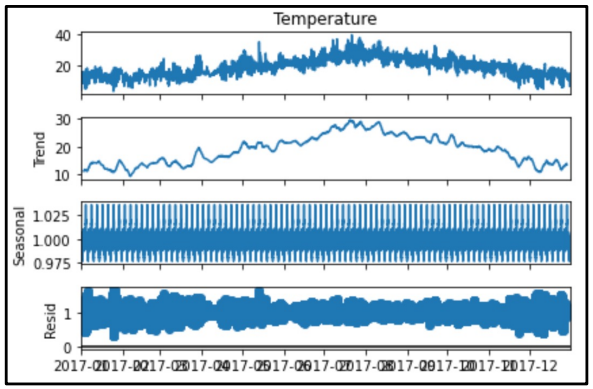
# IoT Sensor Data Forecasting

```
from statsmodels.tsa.seasonal import seasonal_decompose
result = seasonal_decompose(df["Temperature"],
                            model="multiplicative",
                            period = 60*10)
result.plot()
```

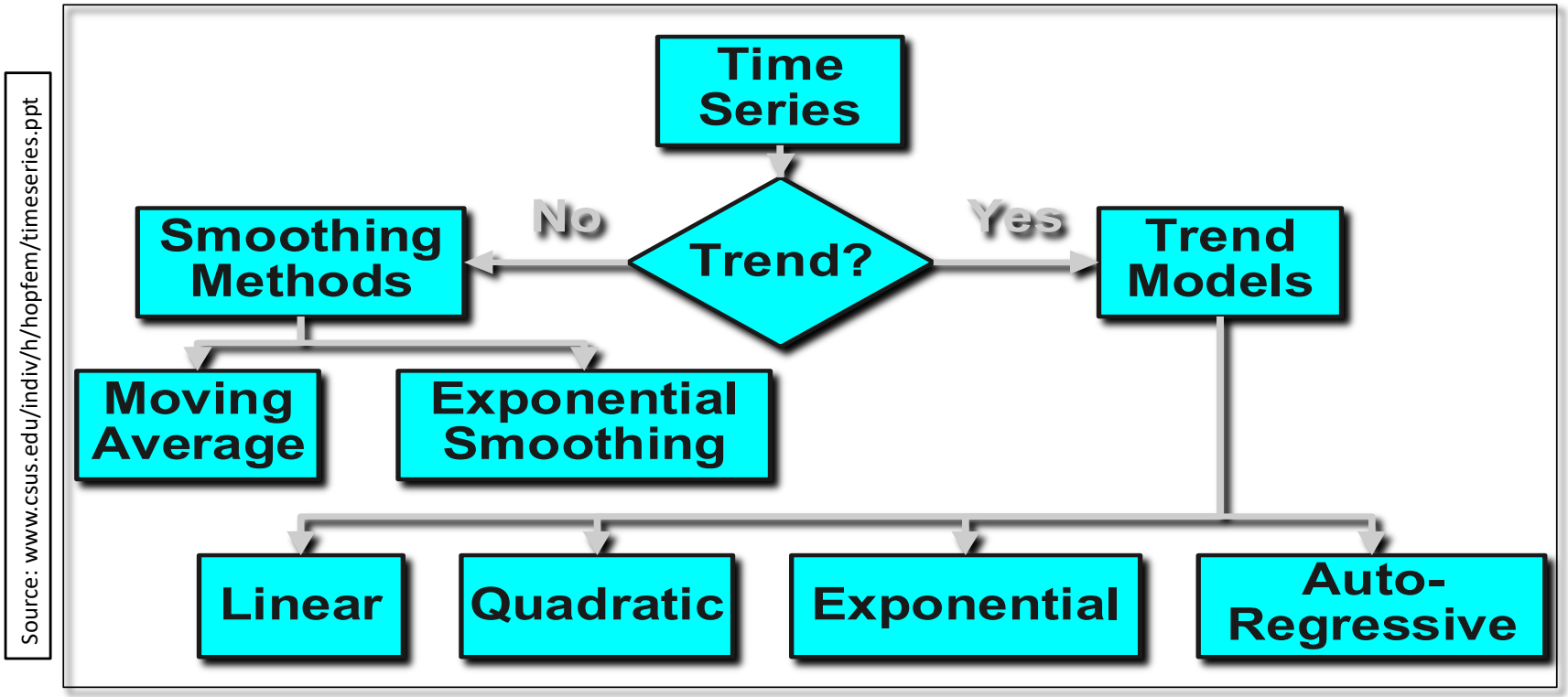
ADDITIVE




MULTIPLICATIVE



# IoT Sensor Data Forecasting



# IoT Sensor Data Forecasting

(SIMPLE) SMOOTHING METHODS

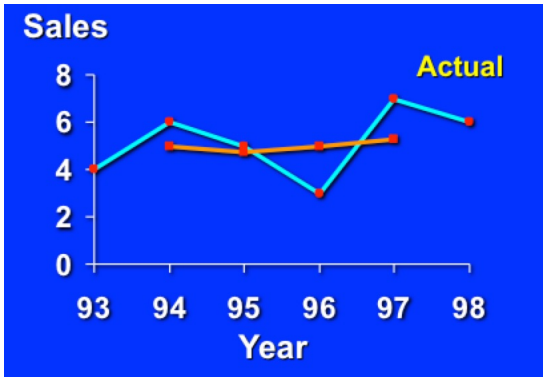
Source: www.csus.edu/indiv/h/hopfem/timeseries.ppt

# IoT Sensor Data Forecasting

## ✧ Moving Average Method

- ❑ Elementary **forecasting** solution
- ❑ The *n*-period forecast is simply the average of the observations in the most recent *n* periods:

$$A_t = (x_t + x_{t-1} + \ldots + x_{t-n+1}) / n$$



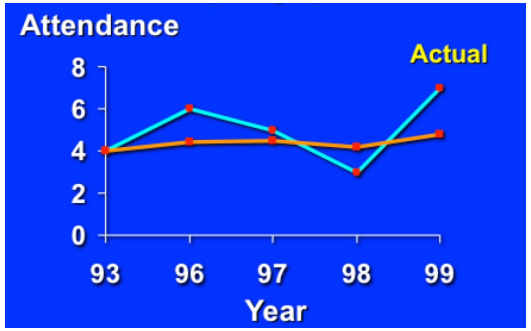
Source: www.csus.edu/indiv/h/hopfem/timeseries.ppt

# IoT Sensor Data Forecasting

## ✧ Exponential Smoothing

- ❑ Form of weighted moving average
- ❑ Use a smoothing costant (W), range between 0 and 1
- ❑ Record only **actual value** and **past prediction**

$$S_t = \alpha x_t + (1 - \alpha) S_{t-1}$$



# IoT Sensor Data Forecasting

LINEAR REGRESSION METHODS

Source: R.B. Clough – UNH M. E. Henrie - UAA

# IoT Sensor Data Forecasting

- **Straightforward approach** for predicting a quantitative response
- *Parametric* method, *explainable* modeling
- Assume a *linear relationship* between *X* and *Y*:

LINEAR REGRESSION

$$\hat{y} = \hat{f}(\boldsymbol{x}) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_p x_p$$

$$Estimate\ \hat{f}(\boldsymbol{x}) \rightarrow Estimate\ \beta_0, \beta_1, \beta_2 \ldots \beta_p$$

Use the **training data** to fit/train the model …

Source: R.B. Clough – UNH M. E. Henrie - UAA

# IoT Sensor Data Forecasting

## ❑ Simple Linear Regression ($p$=1)

Dataset composed of *n* observation pairs:

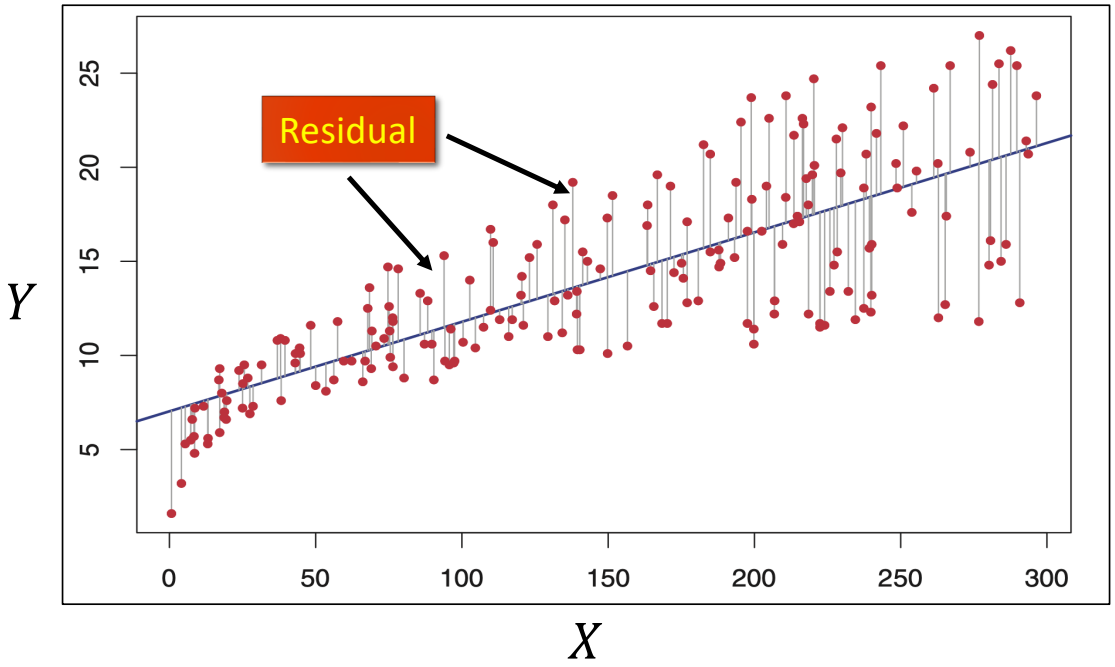
$$(x_1, y_1), (x_2, y_2), \dots (x_n, y_n)$$

Regression problem:

$$\hat{y} = \hat{f}(\boldsymbol{x}) = \beta_0 + \beta_1 x$$

# IoT Sensor Data Forecasting

## ❑ Simple Linear Regression ($p$=1)




Real $Y$ values:

$$y_1, y_2, y_3, \dots, y_n$$

Predicted $\hat{Y} = \hat{f}(x)$ values:

$$\hat{y}_1, \hat{y}_2, \hat{y}_3, \dots, \hat{y}_n$$

Residual of the i-th prediction:

$$e_i = y_i - \hat{y}_i$$

Residual Sum of Squares (**RSS**):

$$\sum_{1}^{n} e_i^2 = \sum_{1}^{n} (y_i - \hat{y}_i)^2$$

# IoT Sensor Data Forecasting

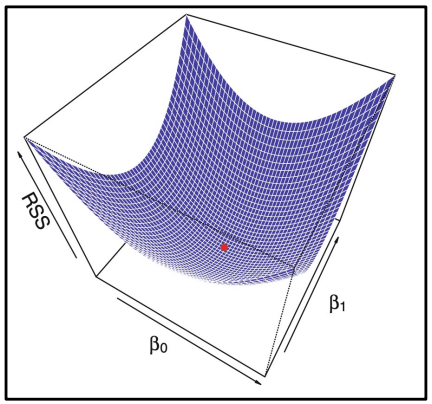
## ❑ Simple Linear Regression ($p$=1)

**LINEAR LEAST SQUARE** METHODS

Compute $\beta_0$, $\beta_1$ values such that the **RSS function is minimized**

$$argmin_{\beta_0,\,\beta_1} RSS = \sum_{1}^{n} (y_i - \hat{y}_i)^2$$

$$= \sum_{1}^{n} (y_i - \beta_0 - \beta_1 x_i)^2$$



# IoT Sensor Data Forecasting

- **Simple Linear Regression** ($p$=1)

**LINEAR LEAST SQUARE** METHODS

$$RSS = \sum_{1}^{n} (y_i - \beta_0 - \beta_1 x_i)^2$$

$$\begin{cases} \dfrac{\partial RSS}{\partial \beta_0} = -2 \sum_{1}^{n} (y_i - \beta_0 - \beta_1 x_i) = 0 \\ \dfrac{\partial RSS}{\partial \beta_1} = -2 x_i \sum_{1}^{n} (y_i - \beta_0 - \beta_1 x_i) = 0 \end{cases}$$

# IoT Sensor Data Forecasting

## Linear Regression

$$\hat{y} = \hat{f}(\boldsymbol{x}) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_p x_p$$

Rewriting the system of linear Equations with **matrices**:

$$Y = X\beta$$

$$Y = \begin{pmatrix} y_1 \\ \dots \\ y_n \end{pmatrix} \quad X = \begin{pmatrix} 1 & x_{1,1} & \dots & x_{1,p} \\ 1 & x_{1,\dots} & \dots & \dots \\ 1 & x_{1,1} & \dots & x_{n,p} \end{pmatrix} \quad \beta = \begin{pmatrix} \beta_0 \\ \dots \\ \beta_p \end{pmatrix}$$

# IoT Sensor Data Forecasting

## Linear Regression

$$\hat{y} = \hat{f}(\boldsymbol{x}) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_p x_p$$

**LINEAR LEAST SQUARE** METHODS

Compute $\beta$ vector such that the **RSS function is minimized**

$$argmin_\beta \, RSS = \|Y - \beta X\|^2$$

$$argmin_\beta \, RSS = (Y - \beta X)^T (Y - \beta X)$$

In the Equation above, $\beta$ is a quadratic function with $p$+1 parameters

# IoT Sensor Data Forecasting

## Linear Regression

$$\hat{y} = \hat{f}(\boldsymbol{x}) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_p x_p$$

**LINEAR LEAST SQUARE** METHODS

$$argmin_\beta \, RSS = (Y - \beta X)^T (Y - \beta X)$$

Differentiating with respect to $\beta$ we get:

$$\frac{\partial RSS}{\partial \beta} = -2\mathrm{XT}(\mathrm{Y} - \mathrm{X}\beta) = 0$$

$$\beta = (X^T X)^{-1} X^T Y$$

# IoT Sensor Data Forecasting

Generalizing the linear regression model …

- **Polynomial Regression**

$$\hat{y} = \hat{f}(\boldsymbol{x}) = \beta_0 + \beta_1 x_1 + \beta_2 x^2{}_2 + \cdots + \beta_p x^d{}_p$$

- **Basis Function**

$$\hat{y} = \hat{f}(\boldsymbol{x}) = \beta_0 + \beta_1 \mathrm{b}_1(x_1) + \beta_2 \mathrm{b}_2(x_1) + \cdots + \beta_p \mathrm{b}_p(x_p)$$

Functions $\mathrm{b}_n(x_n)$ are fixed and known in advance

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

# IoT Sensor Data Forecasting

## ❑ How to assess the accuracy of the model?

Total **variance** of data:

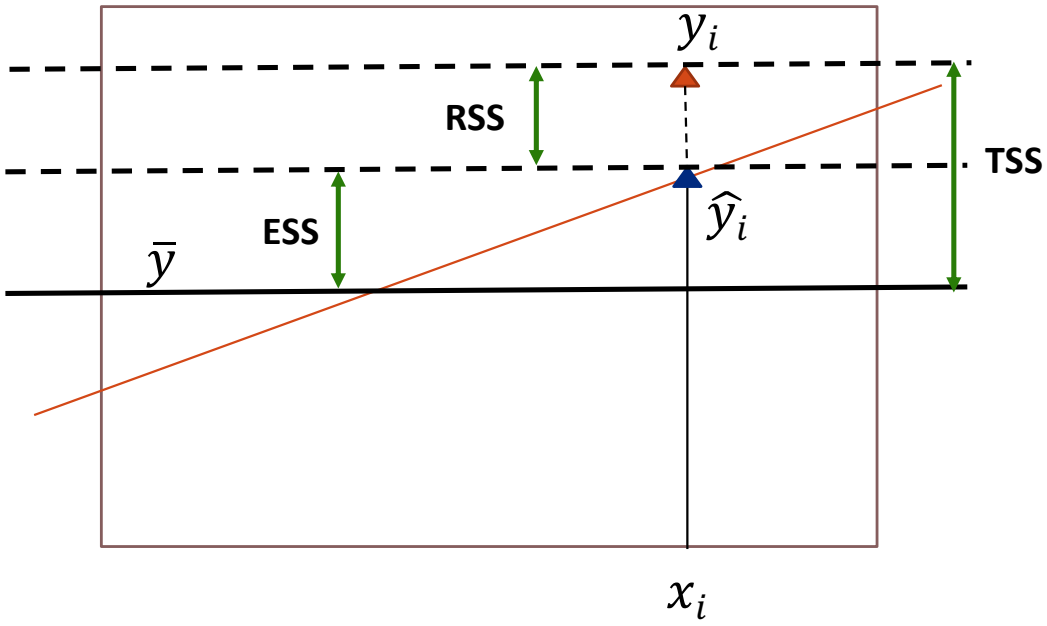
$$\underbrace{\sum_{1}^{n}(y_i - \bar{y})^2}_{\text{TSS}} = \underbrace{\sum_{1}^{n}(\hat{y}_i - \bar{y})^2}_{\text{ESS}} + \underbrace{\sum_{1}^{n}(y_i - \hat{y}_i)^2}_{\text{RSS}}$$

- **TSS** (Total Sum of Squares) → total variance in the response data
- **ESS** (Explained Sum of Squares) → variance explained by the model
- **RSS** (Residual Sum of Squares) → variance still present in the model

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

# IoT Sensor Data Forecasting

- How to assess the accuracy of the model?




**TSS** $\sum_{1}^{n}(y_i - \bar{y})^2 =$

**RSS** $\sum_{1}^{n}(y_i - \hat{y}_i)^2 +$

**ESS** $\sum_{1}^{n}(\hat{y}_i - \bar{y})^2$

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

# IoT Sensor Data Forecasting

## ❑ How to assess the accuracy of the model?

$$R^2 = \frac{ESS}{TSS} = 1 - \frac{RSS}{TSS}$$

$R^2$ statistic:

- Lies between 0 and 1 → *Relative lack of fit*
- Measures the proportion of variability in Y that can be explained by the regression
- **Close to 1** → The model fits well the data
- **Close to 0** → Linear model is wrong or inherent error $\varepsilon$ is high
- Acceptable value depends on the application scenario (e.g. marketing, biology, etc)
- It is a measure of the linear relationship between X and Y
  (indeed, $R^2 = Corr(X, Y)^2$ for the simple case)

# IoT Sensor Data Forecasting

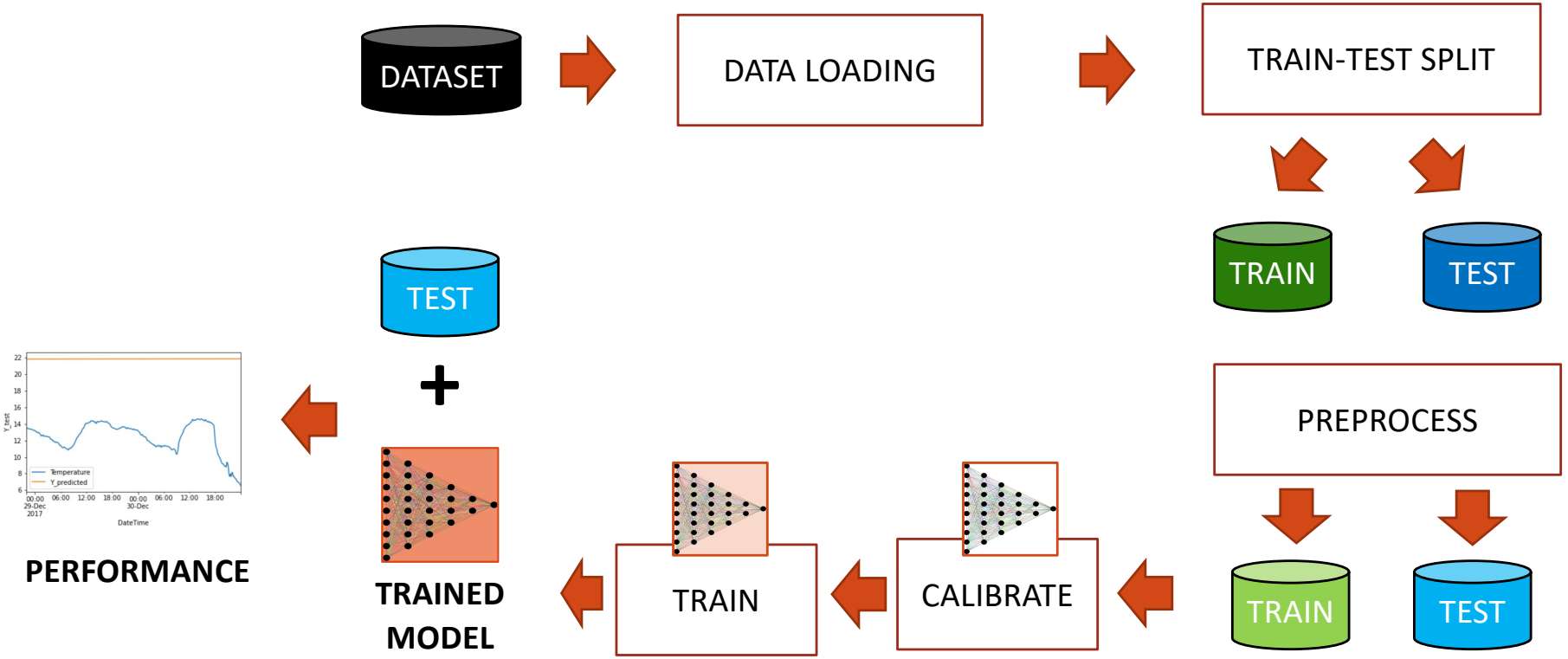
STEP 1: LOAD THE DATASET

```
df = pd.read_csv("../tetuan.csv",  parse_dates = ["DateTime"], index_col = "DateTime")
df["Time"] = np.arange(len(df.index))

X = pd.DataFrame(df["Time"])
y = pd.DataFrame(df["Temperature"])
```

STEP 2: SPLIT THE DATASET

```
TOTAL_SAMPLES = len(df["Time"])
TEST_SIZE = 300

X_train = X.iloc[:TOTAL_SAMPLES - TEST_SIZE]
X_test = X.iloc[TOTAL_SAMPLES - TEST_SIZE:]
y_train = y.iloc[:TOTAL_SAMPLES - TEST_SIZE]
y_test = y.iloc[TOTAL_SAMPLES - TEST_SIZE:]
```

# IoT Sensor Data Forecasting



# IoT Sensor Data Forecasting

STEP 3: TRAIN THE MODEL

```
lreg = LinearRegression()
lreg.fit(X_train, y_train)
```

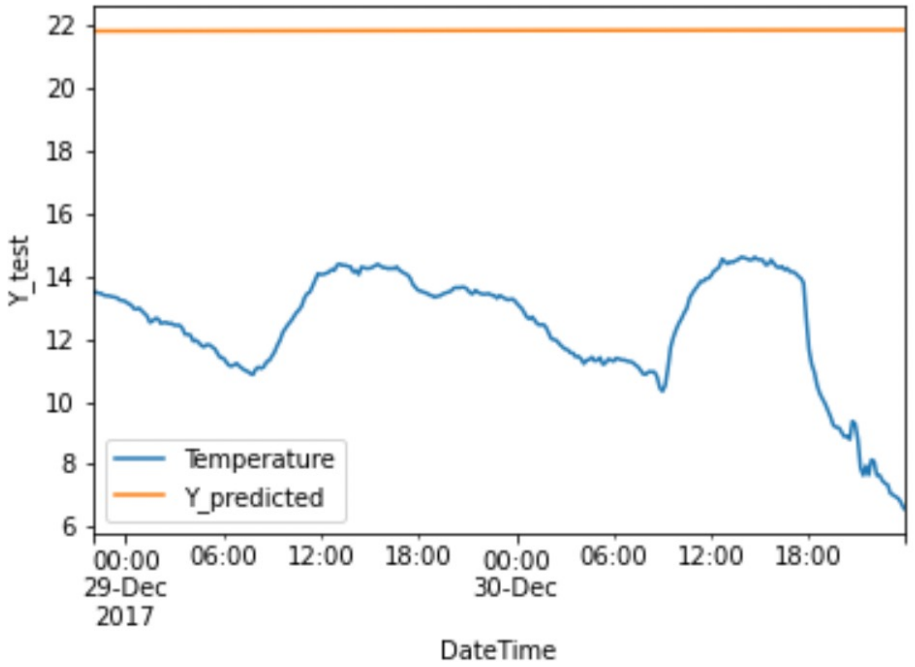
STEP 4: PREDICT THE VALUES OF THE TEST

```
y_pred = lreg.predict(X_test)
```

STEP 5: COMPUTE THE STATS

```
mse=mean_squared_error(y_test, y_pred)
mae=mean_absolute_error(y_test, y_pred)
rsquare = r2_score(y_test, y_pred)
```

# IoT Sensor Data Forecasting




```
Beta_0:
 [[0.00011495]]
Beta_1:
 [15.8517469]
Mean Square Error: 93.22180989659512
Mean Absolute Error: 9.478330181662253
```

# IoT Sensor Data Forecasting

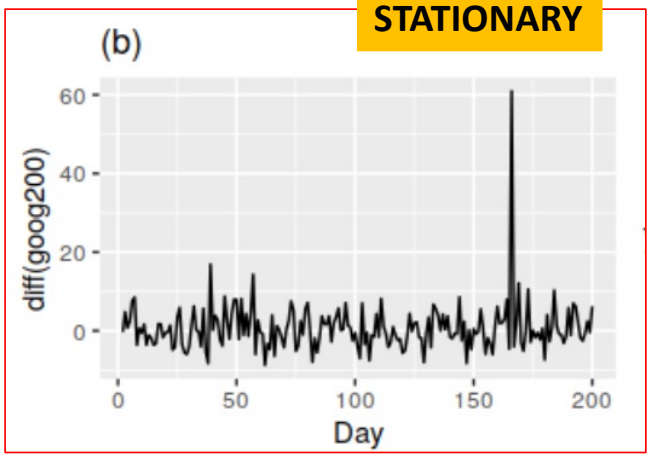
AUTO-REGRESSION METHODS

# IoT Sensor Data Forecasting

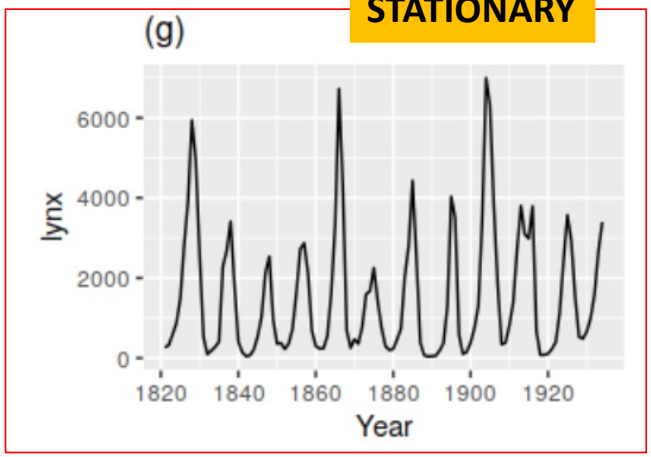
✧ **Time-series (Weak) Stationary** **Source**: https://otexts.com/fpp2/stationarity.html

✓ **Informal definition** → mean and variance of the time-series do not change over time.

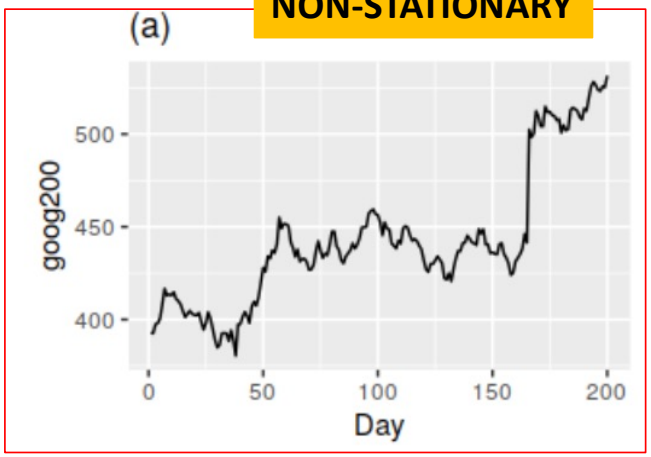
**STATIONARY**




**STATIONARY**



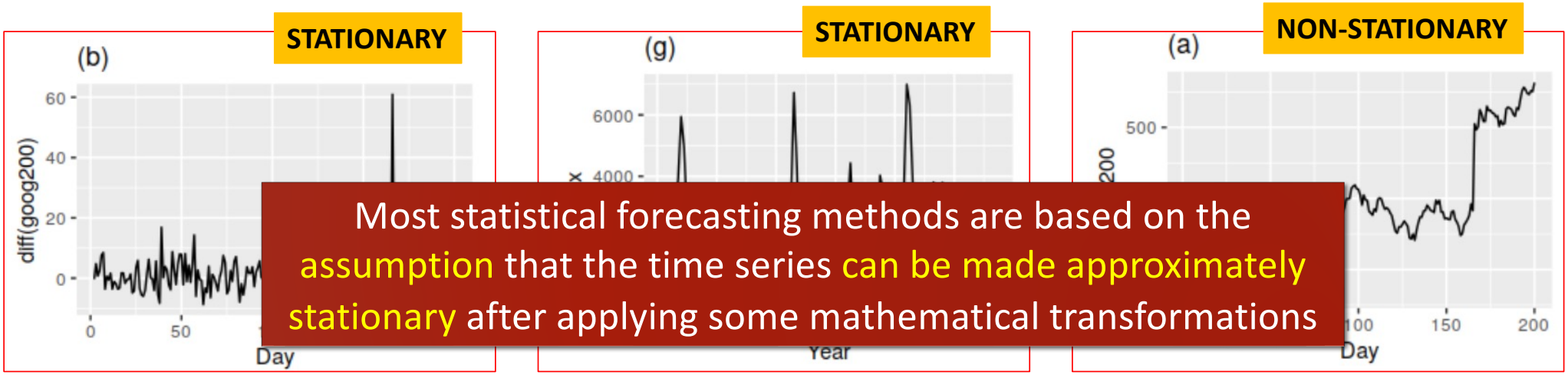

**NON-STATIONARY**



# IoT Sensor Data Forecasting

✧ **Time-series (Weak) Stationary** **Source**: https://otexts.com/fpp2/stationarity.html

✓ **Informal definition** → mean and variance of the time-series do not change over time.




Most statistical forecasting methods are based on the assumption that the time series can be made approximately stationary after applying some mathematical transformations

# IoT Sensor Data Forecasting

**STATIONARY TIME-SERIES**

✧ **Lag** → delay between two points/timeseries

- $X_t$ and $X_{t+k}$ samples have lag $k$
- $\{X_0, X_1, \dots, X_t\}$ and $\{X_k, X_{k+1}, \dots, X_{k+t}\}$ are $k$-lagged time-series

## ✧ Autocorrelation Function (ACF)

- ✧ Generally speaking, autocorrelation is the correlation of a signal with a delayed copy of itself (Wikipedia)
- ✧ It measures the degree of similarity between a given time series, and a lagged version of itself.

$$\rho_{X,Y} = \operatorname{corr}(X,Y) = \frac{\operatorname{cov}(X,Y)}{\sigma_X \sigma_Y} = \frac{\mathrm{E}[(X-\mu_X)(Y-\mu_Y)]}{\sigma_X \sigma_Y}$$

Correlation function

$$r_k = \frac{\sum_{t=k+1}^{n} (y_t - \bar{y})(y_{t-k} - \bar{y})}{\sum_{t=1}^{n} (y_t - \bar{y})^2}$$

Autocorrelation

# IoT Sensor Data Forecasting

## ✧ Moving Average (MA) Model

**STATIONARY TIME-SERIES**

❑ Q-th order moving average model:

$$Y_t = \mu + \varepsilon_t + \alpha_1 \cdot \varepsilon_{t-1} + \alpha_2 \cdot \varepsilon_{t-2} + \cdots + \alpha_q \cdot \varepsilon_{t-q}$$

- Weighted moving average of the past few forecast errors
- $\mu$ is the mean of the series
- $\varepsilon_i$ are error terms (white noise terms, zero mean, iid)
- $q$ is the order of MA model. How to choose it? ACF!

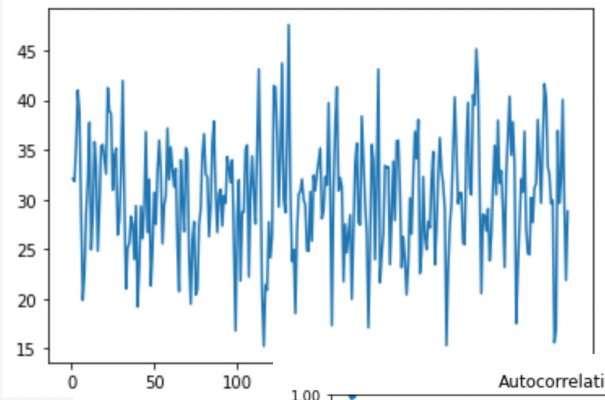
RULE. The autocorrelation function (ACF) of an MA(q) process is zero at lag q + 1 and greater.

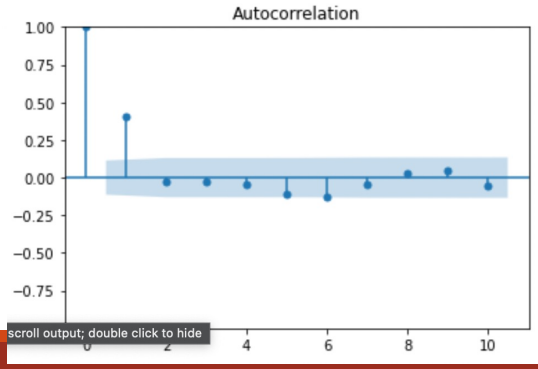
# IoT Sensor Data Forecasting

MA(1) process

STATIONARY TIME-SERIES

```
import numpy as np
from statsmodels.graphics import tsaplots

NUM_VALUES = 300
VARIANCE = 5
THETA = 0.5
MEAN = 30

e = np.random.normal(0, VARIANCE, NUM_VALUES)
y = []
x = range(1, NUM_VALUES)
for i in range(1, NUM_VALUES):
    val = MEAN + THETA * e[i-1]+ e[i]
    y.append(val)
plt.plot(x,y)
fig = tsaplots.plot_acf(y, lags = 10)
```



# IoT Sensor Data Forecasting

## ✧ Auto Regression (AR) Model

**STATIONARY TIME-SERIES**

- ❑ P-th order auto-regression model:

$$Y_t = c + \phi_1 Y_{t-1} + \phi_2 Y_{t-2} + \cdots + \phi_p Y_{t-p} + \varepsilon_t$$

  - Value of the series at time t depends from past $p$ values
  - $\varepsilon_i$ are white noise terms (zero mean, iid)
  - $p$ is the order of MA model. How to choose it?
    - ACF is not informative (too many parameters)
    - Use Partial Autocorrelation Fuction (**PACF**)

# IoT Sensor Data Forecasting

## ✧ Partial Autocorrelation Fuction (PACF)

**STATIONARY TIME-SERIES**

- ❑ Partial correlation of the time-series with itself at that lag given all the information between the points in time.
- ❑ IDEA: PACF between $X_t$ and $X_{t+k}$ → correlation between the two points removing the amount of correlation explained by $X_{t+1}, X_{t+2}, \dots, X_{t+k-1}$

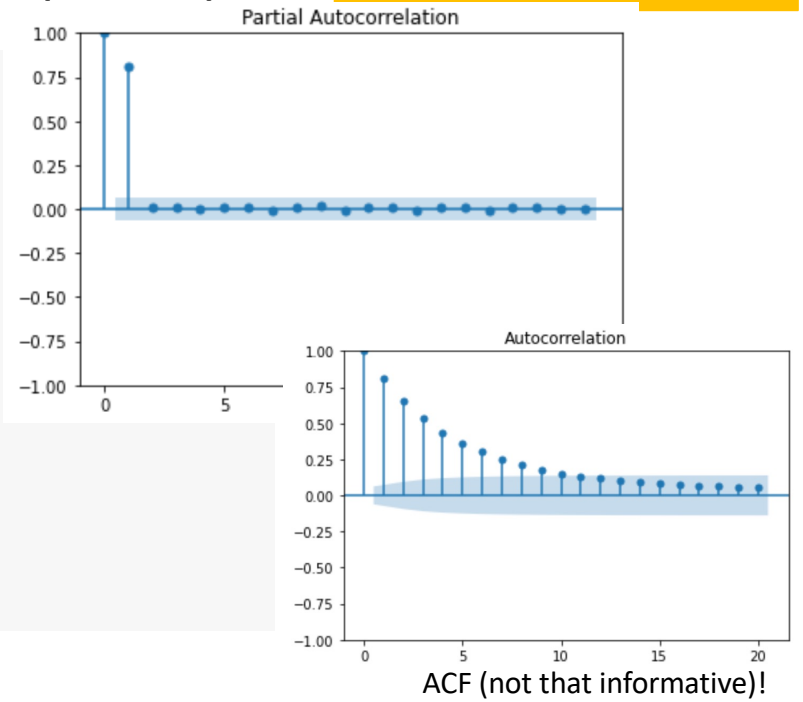
- ❑ HOW to choose the $p$ order for AR process?
  - ○ The partial autocorrelation of an AR($p$) process is zero at lag $p$+1 and greater.

# IoT Sensor Data Forecasting

## ✧ Partial Autocorrelation Fuction (PACF)

**STATIONARY TIME-SERIES**

```
1  from statsmodels.graphics import tsaplots
2  import matplotlib.pyplot as plt
3  import numpy as np
4  import math
5  import random
6
7
8  pval = 100
9
10 for i in range(0,100):
11     cval = 0.8 * pval + random.random()
12     x.append(cval)
13     pval = cval
14
15 fig = tsaplots.plot_pacf(x, lags=20)
16 plt.show()
```




ACF (not that informative)!

# IoT Sensor Data Forecasting

## ✧ ARMA(p,q) Model

**STATIONARY TIME-SERIES**

- Combination of $p$ order Auto Regressive model and $q$ order Moving Average model

$$Y_t = b + \varepsilon_t + \alpha_1 \cdot \varepsilon_{t-1} + \alpha_2 \cdot \varepsilon_{t-2} + \cdots + \alpha_q \cdot \varepsilon_{t-q} +$$
$$\phi_1 \cdot Y_{t-1} + \phi_2 \cdot Y_{t-2} + \cdots + \phi_{\mathrm{t-p}} \cdot Y_{t-p}$$

- Estimate $p$ → Use **ACF** Plot
- Estimate $q$ → Use **PACF** Plot

# IoT Sensor Data Forecasting

## ✧ How to decide whether the time-series is stationary?

- **Dickey-Fuller** (ADF) test
  - The test posits a null hypothesis that a unit root is present in a time series.
  - Depending on the result of the test, the null hypothesis can be rejected for a specified significance level (series is stationary) or accepted (series is not stationary)

INTUITION: AR(1) Model

$$Y_t = c + \phi \cdot Y_{t-1} + \varepsilon_t$$

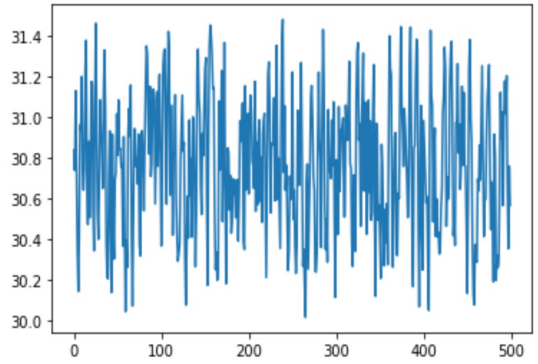
Null Hypothesis is true → unit root is present → $\phi$=1 → mean not constant → Series not stationary!

# IoT Sensor Data Forecasting

## ✧ How to decide whether the time-series is stationary?

### ❑ Dickey-Fuller (ADF) test

```
1  from statsmodels.tsa.stattools import adfuller
2  import math
3
4  ALPHA = 0.5
5  MEAN = 30
6  NUM_SAMPLES = 500
7  perror = 0
8  y = []
9  x = [ i for i in range(0,NUM_SAMPLES) ]
10 for i in x:
11     cerror = random.random()
12     cval = cerror + MEAN + perror * ALPHA
13     y.append(cval)
14     perror = cerror
15
16 result = adfuller(y)
17 print('ADF Statistic: %f' % result[0])
18 print('p-value: %f' % result[1])
```



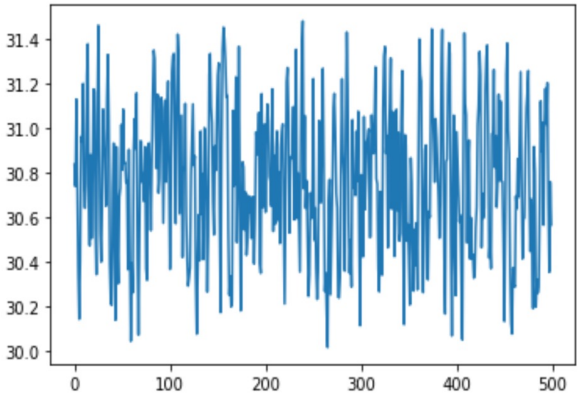

$p$ – value is lower than confidence threshold (0.05) → Hypothesis rejected → **Stationary** Series ! (it is AR(1) …)

```
ADF Statistic: -9.016135
p-value: 0.000000
```

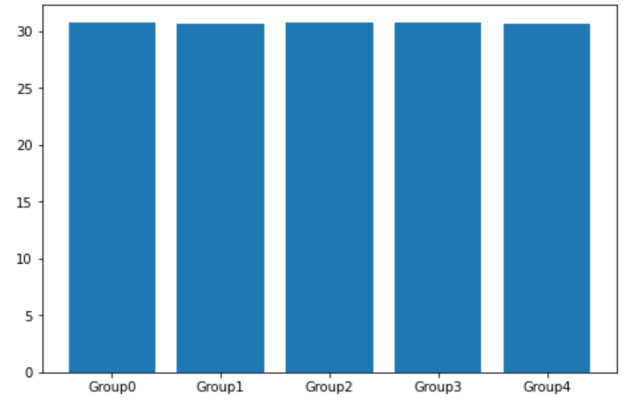
# IoT Sensor Data Forecasting

## ✧ How to decide whether the time-series is stationary?

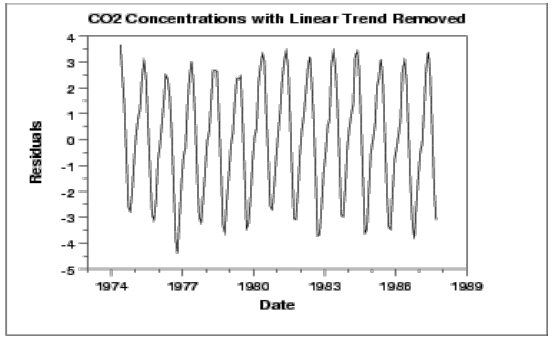
- **Visual** check
  - Split the time series into two (or more) partitions and compare the mean  variance of each group.

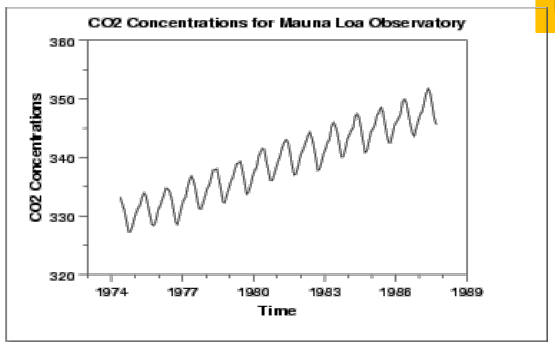



Mean per group (group size: 100 samples)



# IoT Sensor Data Forecasting




**NON-STATIONARY TIME-SERIES**

Source: https://www.itl.nist.gov/div898/handbook/pmc/section4/pmc442.htm

- Transforming a *non-stationary* time-series into a *stationary* time-series

**Differencing** technique → Stabilise the **mean** of a time series by removing changes over time

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

**FIRST ORDER** DIFFERENCING

$$\begin{aligned} y''_t &= y'_t - y'_{t-1} \\ &= (y_t - y_{t-1}) - (y_{t-1} - y_{t-2}) \\ &= y_t - 2y_{t-1} + y_{t-2}. \end{aligned}$$

**SECOND ORDER** DIFFERENCING

# IoT Sensor Data Forecasting

**NON-STATIONARY TIME-SERIES**

- **ARIMA(p,q,d)** *Autoregressive Integrated Moving Average Model*
  - p → number of **autoregressive** terms (AR order)
  - d → number of **nonseasonal differences** (differencing order)
  - q → number of **moving-average** terms (MA order)

$$y'_t = c + \phi_1 y'_{t-1} + \cdots + \phi_p y'_{t-p} + \theta_1 \varepsilon_{t-1} + \cdots + \theta_q \varepsilon_{t-q} + \varepsilon_t$$

$y'_t$ is the d-differenced value at time t

# IoT Sensor Data Forecasting

```python
#Step 1: Load the dataset

import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

df = pd.read_csv('airlines_data.csv')
```

**PUTTING ALL TOGETHER …**

LOAD THE DATASET
AND TRAIN-TEST SPLIT …

```python
#Step 2: Test-train split
from datetime import datetime

df['Month']=pd.to_datetime(df['Month'],infer_datetime_format=True)
df = df.set_index('Month',inplace=False)

nrows = (len(df.values))
splitPoint = int(nrows * 0.80)

train = df['Passengers'][:splitPoint]
test = df['Passengers'][splitPoint:]

```

# IoT Sensor Data Forecasting

```
1  #Step 3: Check whether the time-series is stationary through the Dickey-Fuller test
2
3  from statsmodels.tsa.stattools import adfuller
4
5  result = adfuller(train)
6  print('ADF Statistic: %f' % result[0])
7  print('p-value: %f' % result[1])
```

IS IT STATIONARY? … NO!

```
ADF Statistic: -0.356889
p-value: 0.917052
```

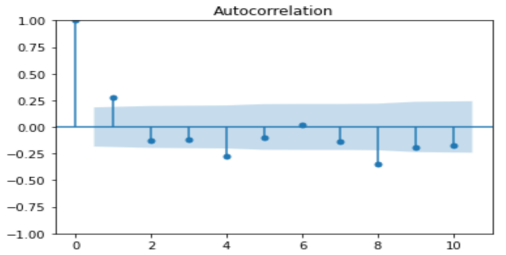
```
1  #Step 4: Find the differencing parameter (d)
2  from statsmodels.graphics import tsaplots
3
4  trainNew = train.diff().dropna()
5
6  result = adfuller(trainNew)
7  print('ADF Statistic: %f' % result[0])
8  print('p-value: %f' % result[1])
```

REPEAT THE TEST AFTER 1ST-ORDER DIFFERENCING
IS IT STATIONARY? YES (threshold=0.1)

```
ADF Statistic: -2.539635
p-value: 0.106126
```

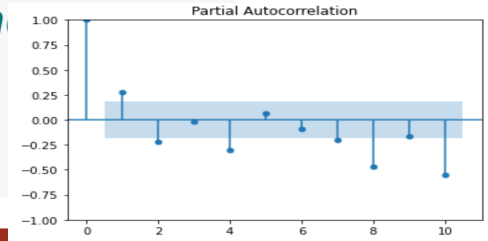
# IoT Sensor Data Forecasting

```
1 #Step 4: Find the order of the MA model through the
2
3 fig = tsaplots.plot_acf(trainNew, lags=10)
4 plt.show()
```




FIND p ORDER FOR THE MA MODEL
COMPUTE ACF → p = 1

```
1 #Step 5: Find the order of the AR model through th
2
3 fig = tsaplots.plot_pacf(trainNew, lags=10)
4 plt.show()
```




FIND q ORDER FOR THE AR MODEL
COMPUTE PACF → q = 1

# IoT Sensor Data Forecasting

```
#Step 6: Apply ARIMA model

from statsmodels.tsa.arima.model import ARIMA

history = [x for x in train]
predictions = list()

for t in range(len(test)):
    model = ARIMA(history, order=(1,1,1))
    model_fit = model.fit()
    output = model_fit.forecast()
    yest = output[0]
    predictions.append(yest)
    obs = test[t]
    history.append(obs)
    print('predicted=%f, expected=%f' % (yest, obs))
```

APPLY ARIMA(p,q,d)

FORECAST ALL SAMPLES IN THE TEST DATASET

# IoT Sensor Data Forecasting

## ✧ Mean Absolute Error (MAE)

- ❑ Measures the total error in forecast (without sign)

$$\mathbf{MAD} = \frac{\sum |\mathbf{actual} - \mathbf{forecast}|}{\mathbf{n}}$$

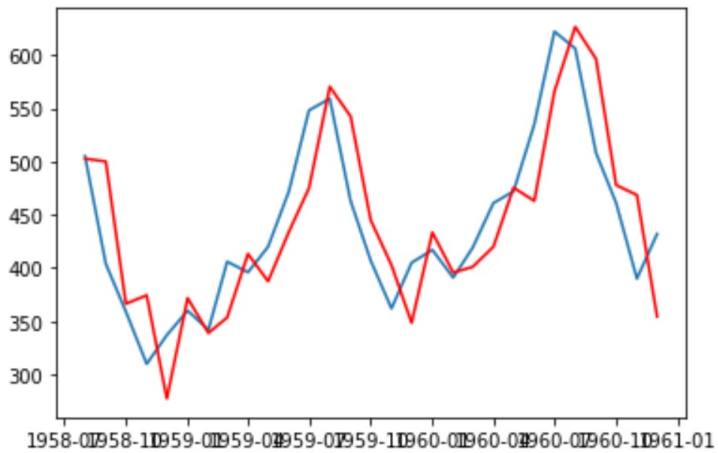
## ✧ Mean Square Error (MSE)

- ❑ Similar to MAD, but gives higher penalty to larger errors

$$\mathbf{MSE} = \frac{\sum (\mathbf{actual} - \mathbf{forecast})^2}{\mathbf{n}}$$

# IoT Sensor Data Forecasting

```
1  import math
2  from sklearn.metrics import mean_squared_error
3
4  rmse = math.sqrt(mean_squared_error(test, predictions))
5  print('Test RMSE: %.3f' % rmse)
6
7  df2 = pd.DataFrame(predictions)
8  df2.set_index(test.index,inplace=True)
9
10 plt.plot(test)
11 plt.plot(df2, color='red')
12 plt.show()
13
```



# IoT Sensor Data Forecasting

BEYOND ARIMA 1/2

- Python `module pmdarima` allows to compute the optimal ARIMA parameters (p, d, and q values). HOW?
- **Grid search** technique (testing parameters in an range)

```
from pmdarima.arima import auto_arima

model = auto_arima(trainset, start_p=1, start_q=1, test='adf', max_p=5,
max_q=5, m=1, d=1, seasonal=False, start_P=0, D=None, trace=True,
error_action='ignore', suppress_warnings=True, stepwise=True)

prediction, confint = model.predict(n_periods=TEST_SIZE,
return_conf_int=True)
```

# IoT Sensor Data Forecasting

BEYOND ARIMA 2/2

- HOW to handle with the seasonality?
- **SARIMA (Seasonal ARIMA)**

  `SARIMA(p,d,q)(P,D,Q)m`

Four additional components:

- P: **Seasonal** autoregressive order.
- D: **Seasonal** difference order.
- Q: **Seasonal** moving average order.
- m: The number of time steps for a single seasonal period.

# IoT Sensor Data Forecasting

BEYOND ARIMA 2/2

- HOW to handle with the seasonality?
- **SARIMA (Seasonal ARIMA)**

  `SARIMA(p,d,q)(P,D,Q)m`

```
from pmdarima.arima import auto_arima

arima_model_auto = auto_arima(y_train, seasonal=True, m=12, trace=True)
```

# IoT Sensor Data Forecasting

PROPHET LIBRARY FOR TIME-SERIES FORECASTING

https://facebook.github.io/prophet/

CONNECTING PROPHET TO INFLUXDB

https://www.influxdata.com/blog/forecasting-with-fb-prophet-and-influxdb/

- Additive Regression Model
- It works best with time-series with strong **seasonal** effects and several seasons of data
- It estimates yearly, weakly and daily seasonality plus the holiday effect
- **Automatic** method (no pre-preprocessing, no parameter tuning)

# IoT Sensor Data Forecasting

```
from datetime import datetime

df['Month'] = pd.DatetimeIndex(df['Month'])
df = df.rename(columns={'Month': 'ds', 'Passengers': 'y'})
df.set_index('ds')
```

```
1 from fbprophet import Prophet
2
3 model = Prophet(interval_width=0.95)
4 model.fit(df)
5 future_dates = model.make_future_dataframe(periods=36, freq='MS')
6 forecast = model.predict(future_dates)
7 model.plot(forecast, uncertainty=True)
8 model.plot_components(forecast)
```

# IoT Sensor Data Forecasting