Trend Prediction

IoT Time Series Forecasting

Presented By:

- Hossam Khairullah
- Mohamed Ashraf El-Melegy
- Mostafa Mohamed Fathy

Supervised By:

- Information Technology Institute (ITI)
- SpimeSenseLabs

Agenda

1. Motivation
2. Dataset
3. Objective & Scope
4. Available Forecasting Techniques
5. Our Model
6. Deployment
7. Live Demo

Motivation

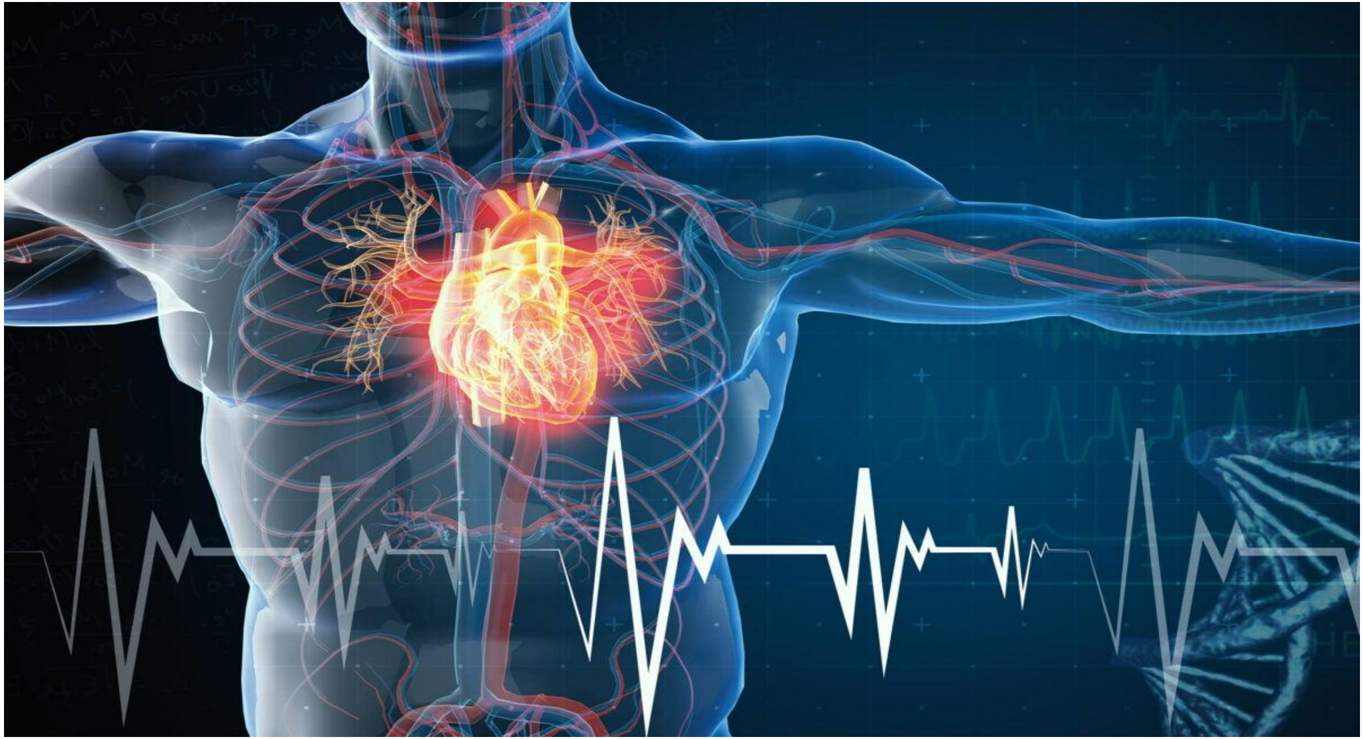
Motivation

Time Series Forecasting has always been a very important area of research in many domains.
For example:

- ➔ Manufacturing
- ➔ Medicine
- ➔ Energy
- ➔ Weather
- ➔ Smart Cities

Motivation: Manufacturing

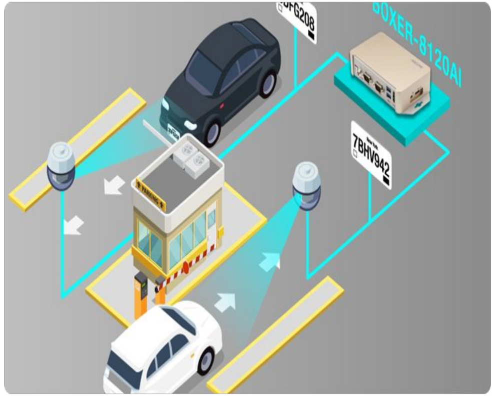
Motivation: Medicine



Motivation: Energy

Motivation: Weather

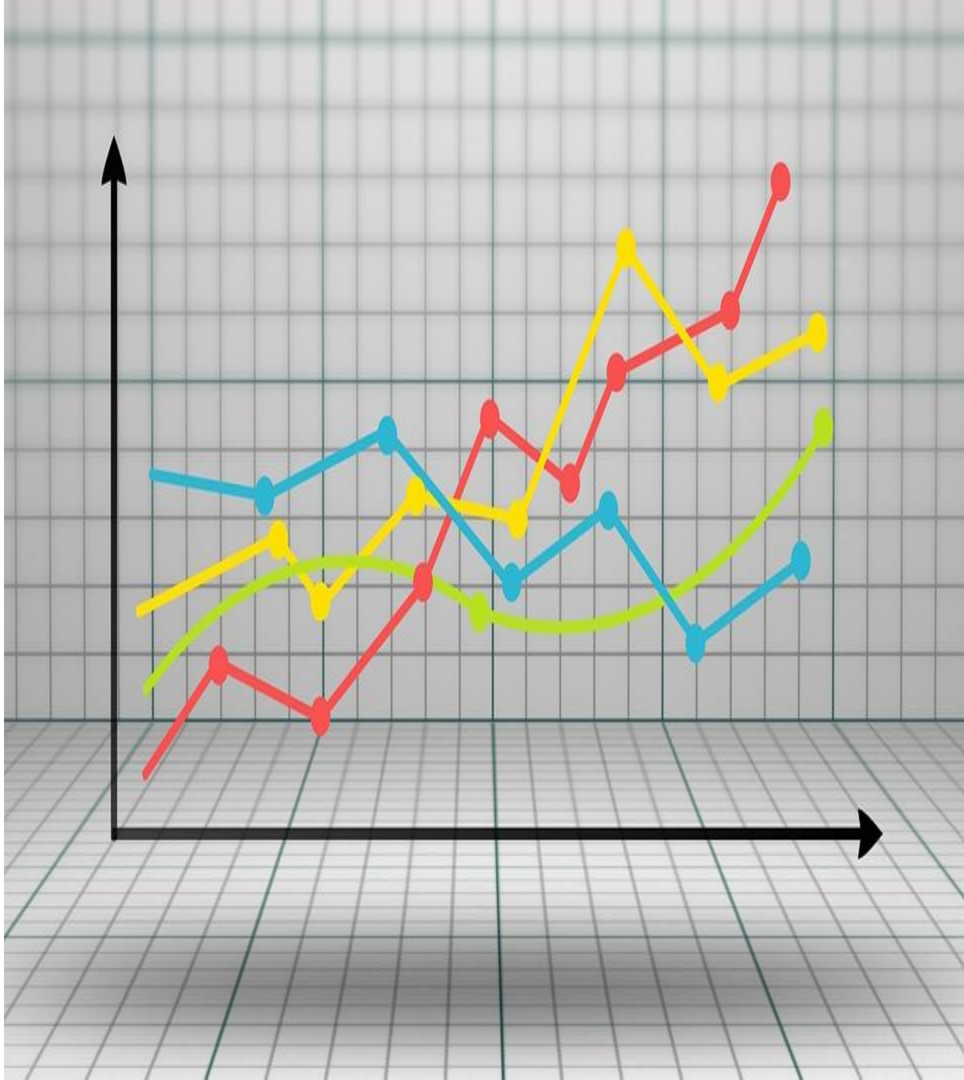
Motivation: Smart Cities



Time Series Analysis

In mathematics, a series of data points indexed (or listed or graphed) in time order. Most commonly, it is a sequence taken at equally spaced points in time. That’s:

Time Series

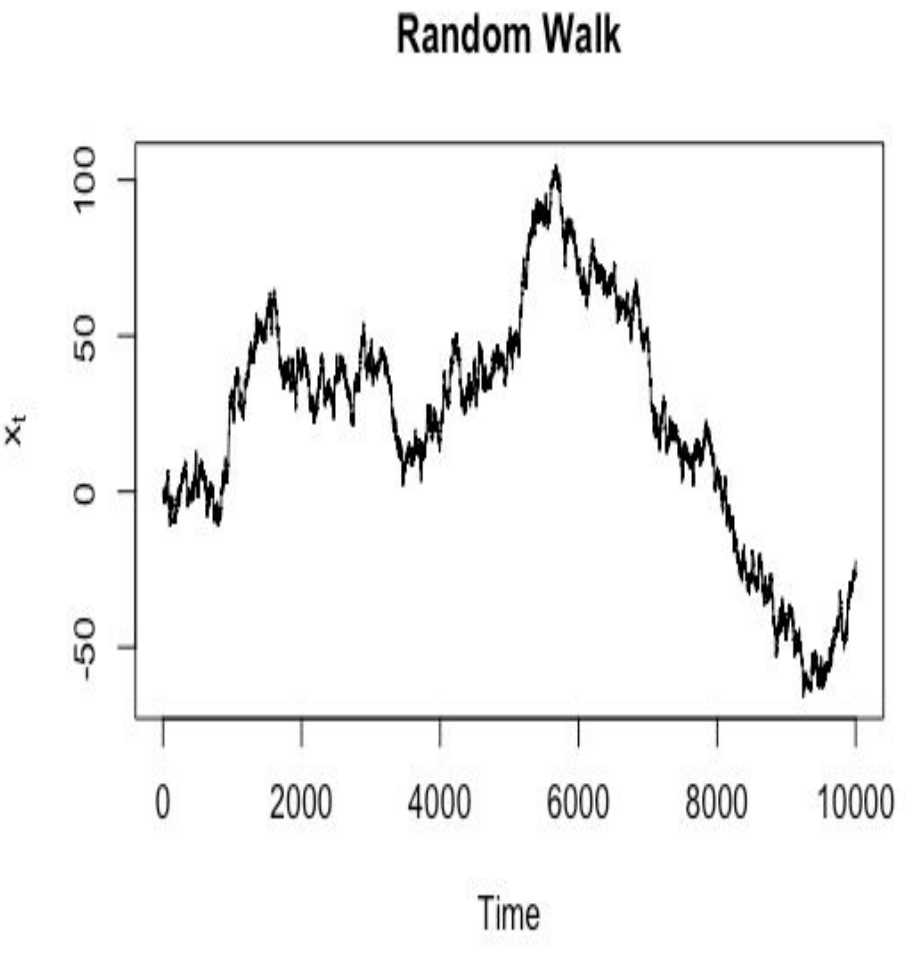


Time Series: Types

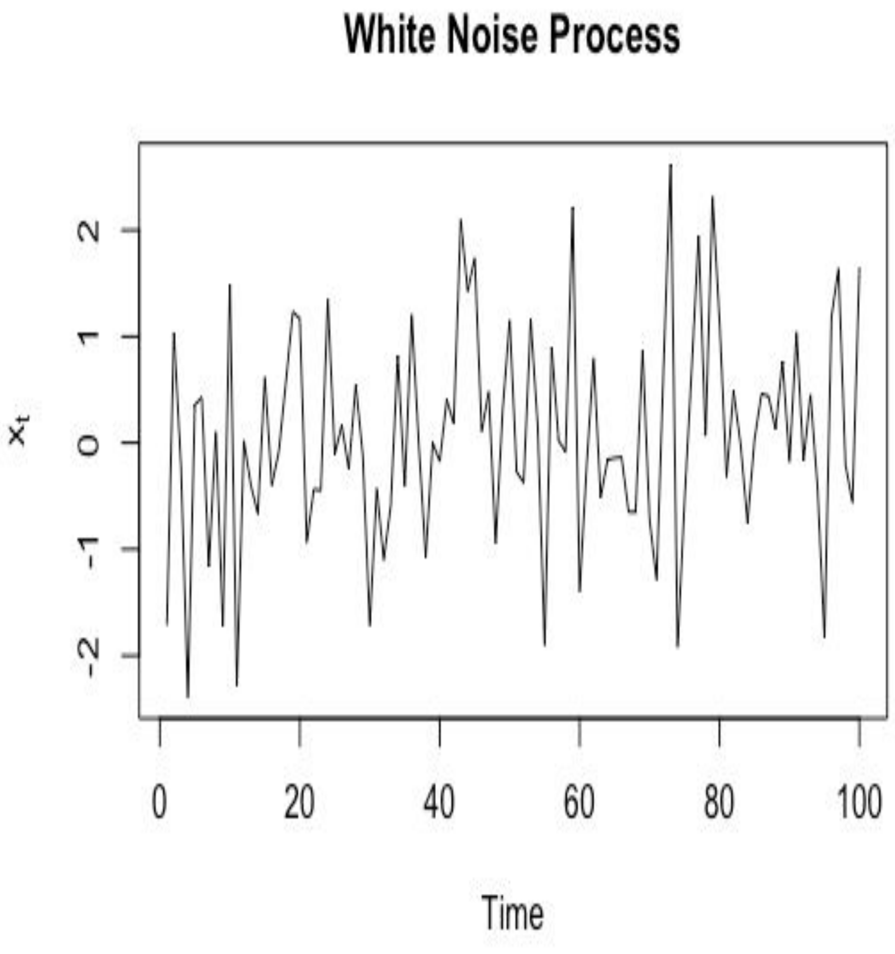
- Non-Stationary
  - Random Walk
  - White Noise
  - Trend Stationary

- Strict Stationary

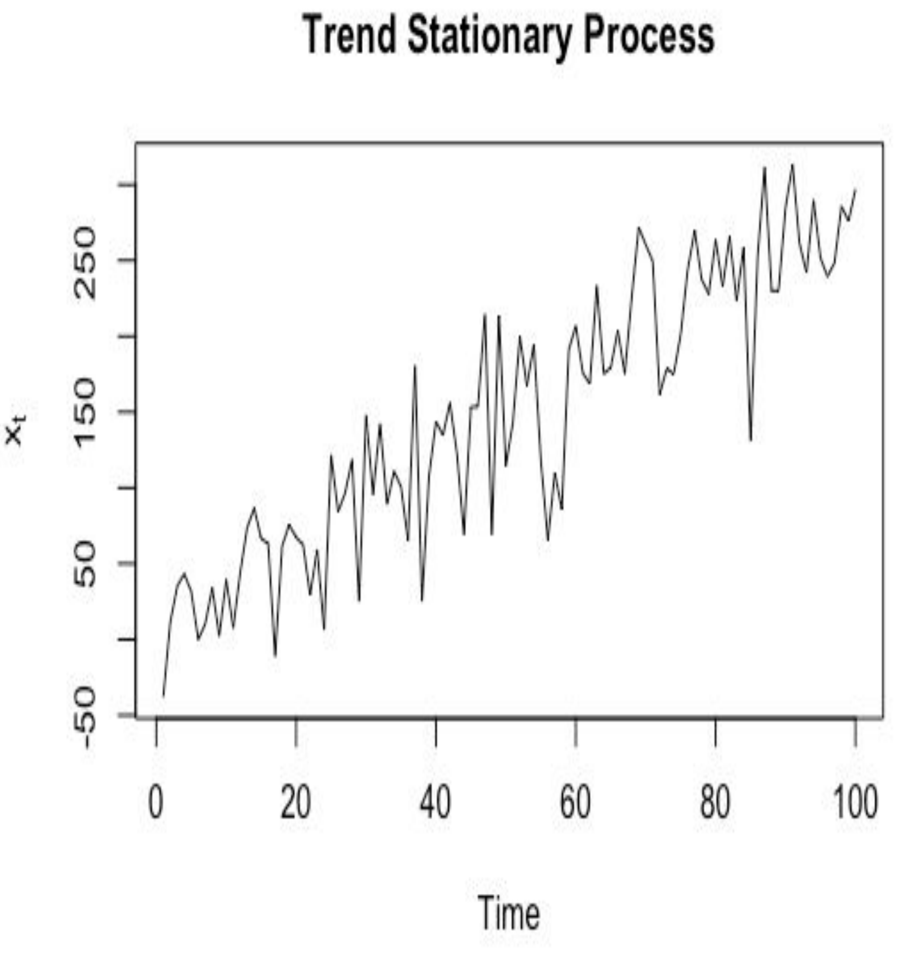
Time Series: Types
Random Walk



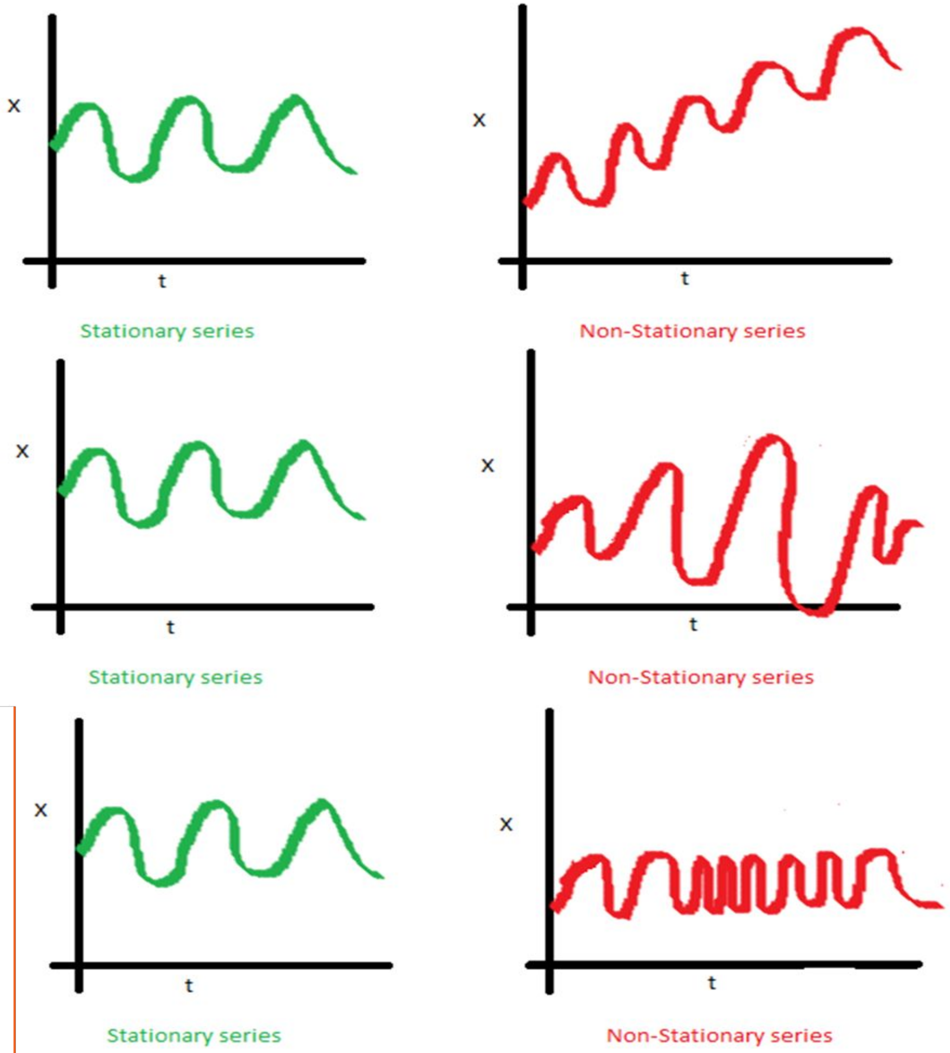
Time Series: Types
White Noise



Time Series: Types
Trend Stationary

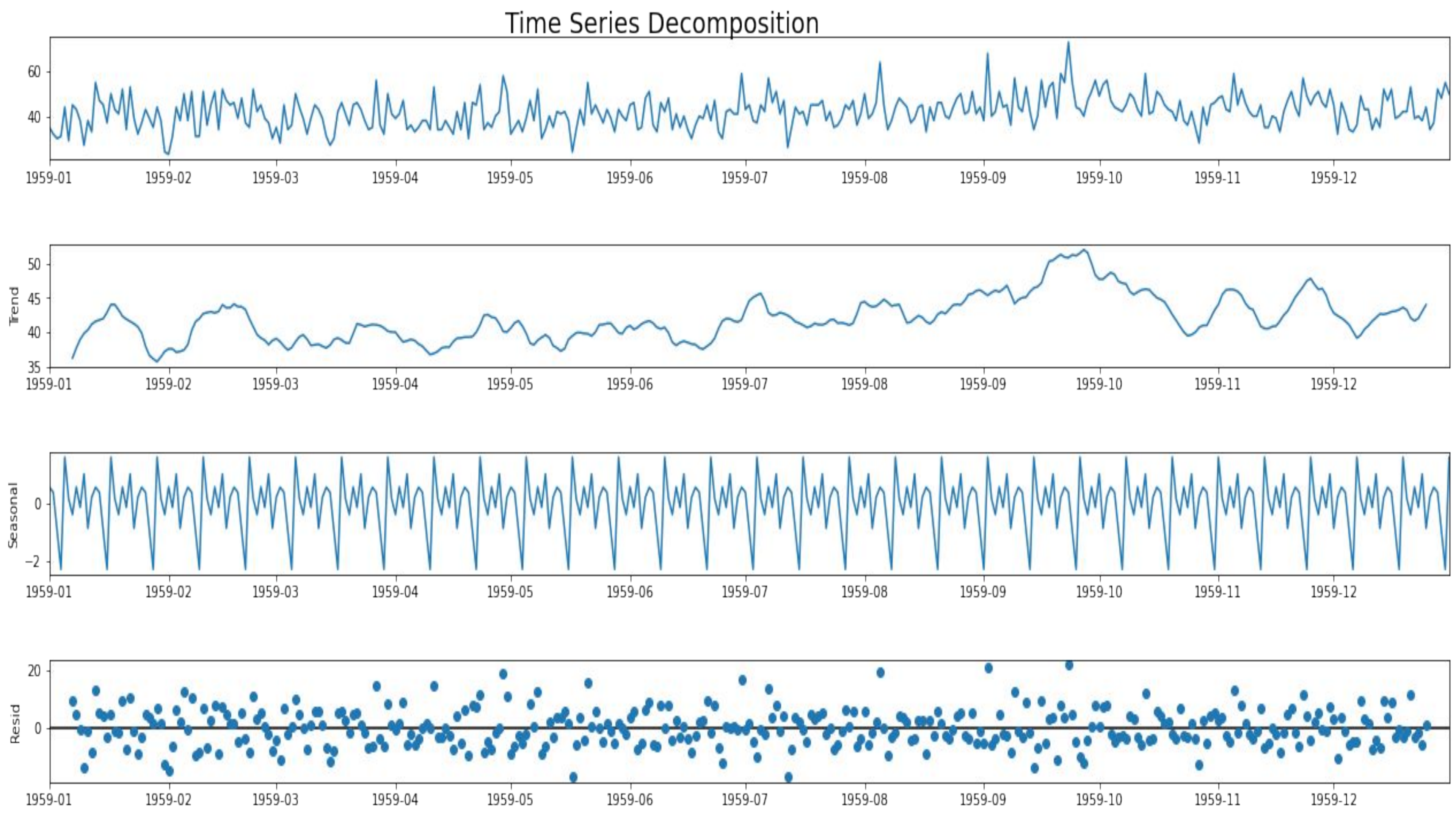


Time Series: Types
Stationary



Time Series: Decomposition

- Long Term Trend
  - Up Trend
  - Downtrend
- Seasonality
  - Based on frequency
  - Annually, Monthly, Daily, etc…
- Noise (Residuals)


Time Series Decomposition
60
40
Trend
50
45
40
35
Seasonal
0
−2
Resid
20
0
1959-01
1959-02
1959-03
1959-04
1959-05
1959-06
1959-07
1959-08
1959-09
1959-10
1959-11
1959-12

Objective & Scope

Objective & Scope

- **Main Objective:**
  - Building a stable Time Series Predictor works for IoT sensor readings in order to forecast the upcoming readings of that sensor.
  - Deployment on Master of Things (MoT).
  - Running of any script in time frame less than 2 seconds.
- **Scope:**
  - Univariate Predictor.
  - Predictor doesn’t forecast long sequence.

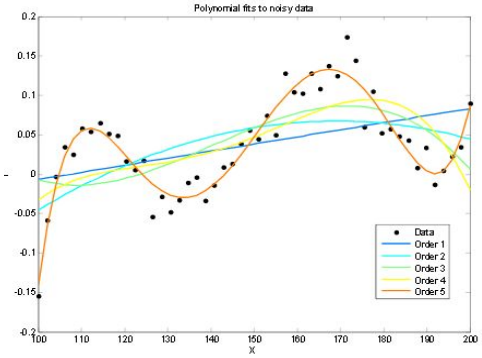
Related Work
Forecasting Methods

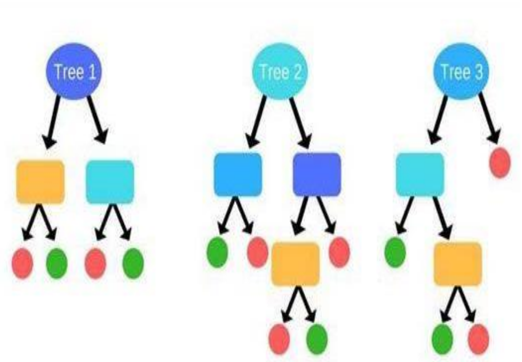
Forecasting: Traditional Methods

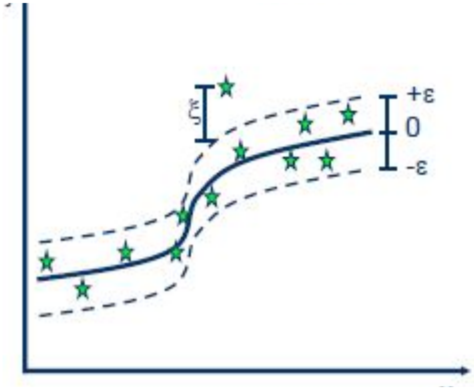
- ➔ Naive Method
- ➔ Simple Average
- ➔ Moving Average
- ➔ Weighted Moving Average
- ➔ Exponential Smoothing

Forecasting: Classic ML

- ➔ Polynomial Regression
- ➔ Random Forests
- ➔ Support Vector Machine



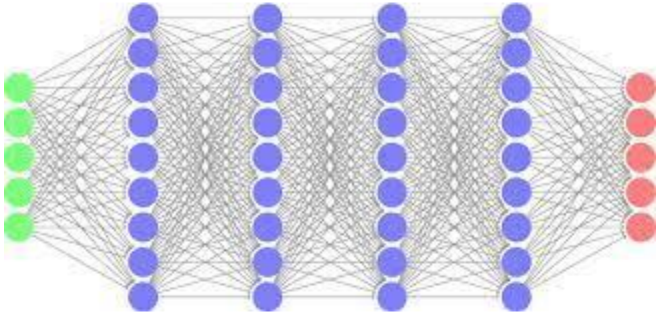


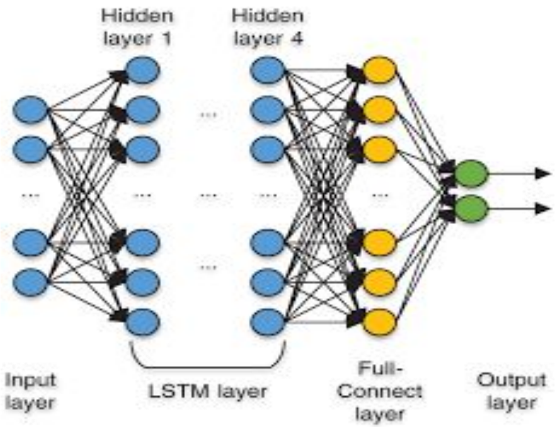


Forecasting: Deep Learning

- ➔ Deep Neural Networks

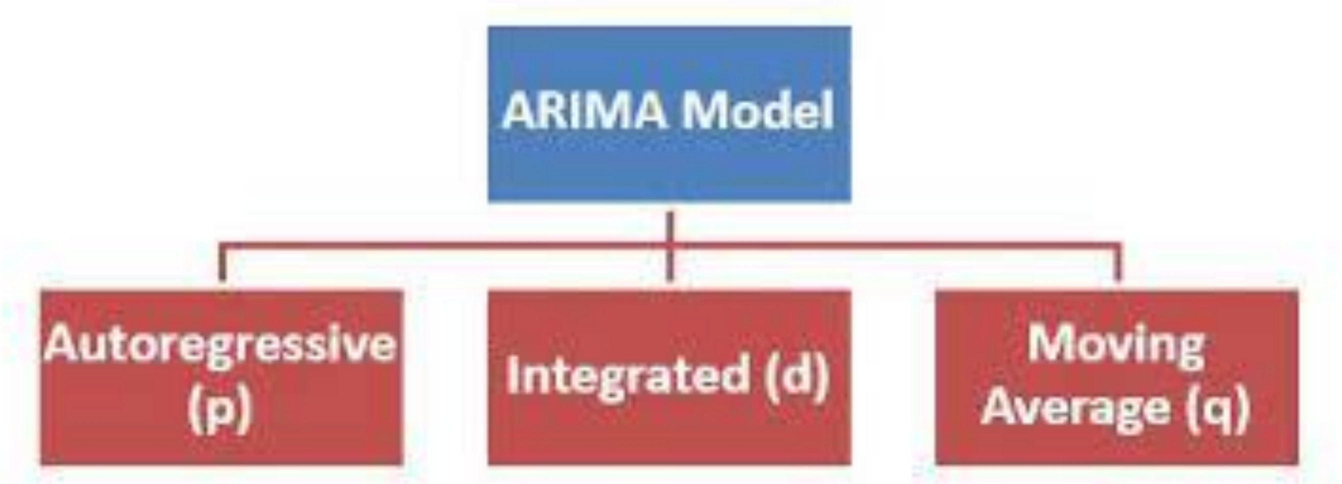
- ➔ Recurrent Neural Networks (RNNs)





Forecasting: ARIMA

AutoRegressive Moving Average



Forecasting: ARIMA

- ARMA
- ARIMA
- SARIMA
- Auto ARIMA

Our ARIMA Model

Our ARIMA Model

- ➔ Overview
- ➔ Mathematically
- ➔ In Code
- ➔ Training Approach
- ➔ Evaluation

Our ARIMA Model: Overview

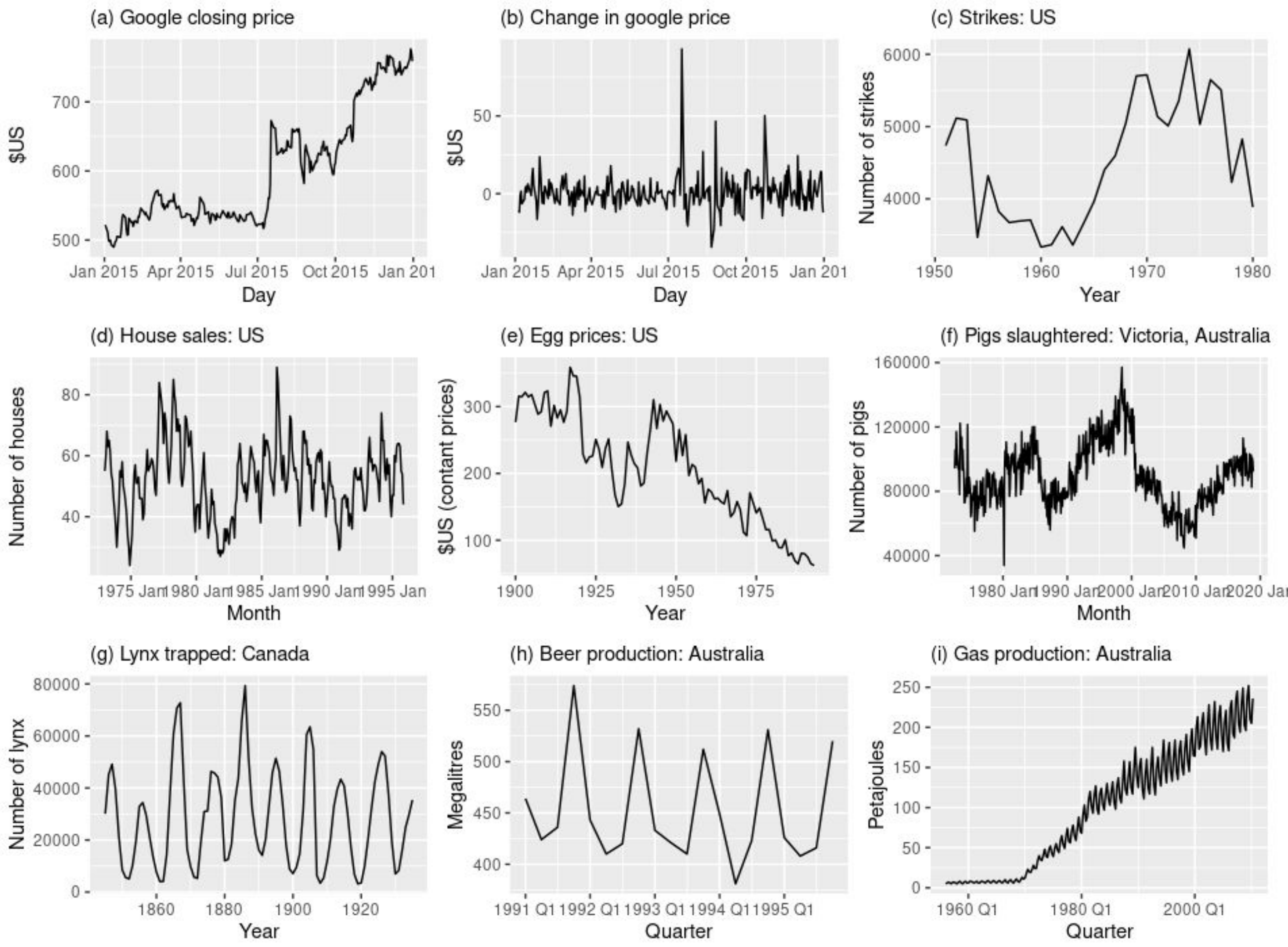
ARIMA(p, d, q)

[Demo Sheet]

- **AR (p):** refers to a model that shows a changing variable that regresses on its own lagged, or prior, values.
- **I (d):** represents the differencing of raw observations to allow for the time series to become stationary.
- **MA (q):** incorporates the dependency between an observation and a residual error.

Our ARIMA Model: Mathematically

- ➔ Stationarisation Techniques
- ➔ Autoregressive
- ➔ Integration
- ➔ Moving Average



https://otexts.com/fpp3/stationarity.html

Our ARIMA Model: Mathematically

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

$y_t - y_{t-1} = \varepsilon_t \rightarrow y_t = y_{t-1} + \varepsilon_t$: random walk

$y_t = c + y_{t-1} + \varepsilon_t$: random walk w/ drift

$y''_t = y'_t - y'_{t-1} = y_t - 2y_{t-1} + y_{t-2}$

$By_t = y_{t-1}; B^2 y_t = y_{t-2} \rightarrow B^d y_t = y_{t-d}$

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

$y''_t = y'_t - y'_{t-1} = y_t - 2y_{t-1} + y_{t-2} = (1 - 2B + B^2)y_t = (1 - B)^2 y_t$

$(1 - B)^d y_t$: d-th order difference

Our ARIMA Model: Mathematically

$$\hat{y}_t = c + \phi_1 y_{t-1} + \phi_2 y_{t-2} + \ldots + \phi_p y_{t-p} + \varepsilon_t$$

$$\hat{y}_t = c + \varepsilon_t + \theta_1 \varepsilon_{t-1} + \theta_2 \varepsilon_{t-2} + \ldots + \theta_q \varepsilon_{t-q}$$

Our ARIMA Model: Mathematically

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

$$(1 - B\phi_1 - \ldots - B^p \phi_p)(1 - B)^d y_t = c + (1 + B\theta_1 + \ldots + B^q \theta_q)\varepsilon_t$$

$$\Phi(B)(y_t' - \mu) = \Theta(B); c = \mu(1 - \phi_1 - \ldots - \phi_p)\varepsilon_t; \mu = \frac{1}{T}\sum_{t=0}^{T} y_t'$$

Our ARIMA Model: In Code

- Pure JS Implementation
  - Pros: has no dependency on any other frameworks.
  - Cons: in absence of any low level optimisation, combined with time limit requirement; data records are kept under 128 & results could be significantly inaccurate.

Our ARIMA Model: In Code

```
module.exports = {
    array, empty, diff, dot, ndim, reshape, shape, sum, transpose, diag, ones,
    zeros, eye, arange, vstack, hstack, NDArray, linalg, linspace, random,
    cumsum, mean, std, prod
}
```

```
class AutoRegressionIntegratedMovingAverage extends GradientDescent {

  constructor(order = [1, 0, 0], KWArgs = { learningRate: 1e-3 }) {
    super(KWArgs.learningRate || 1e-3, KWArgs);
    [this._p, this._d, this._q] = order;
    this._update = function (gradient, m, vt1 = 0) {
      this._W = this._W.add(this.vt(gradient, m, vt1));
    };
  }
```

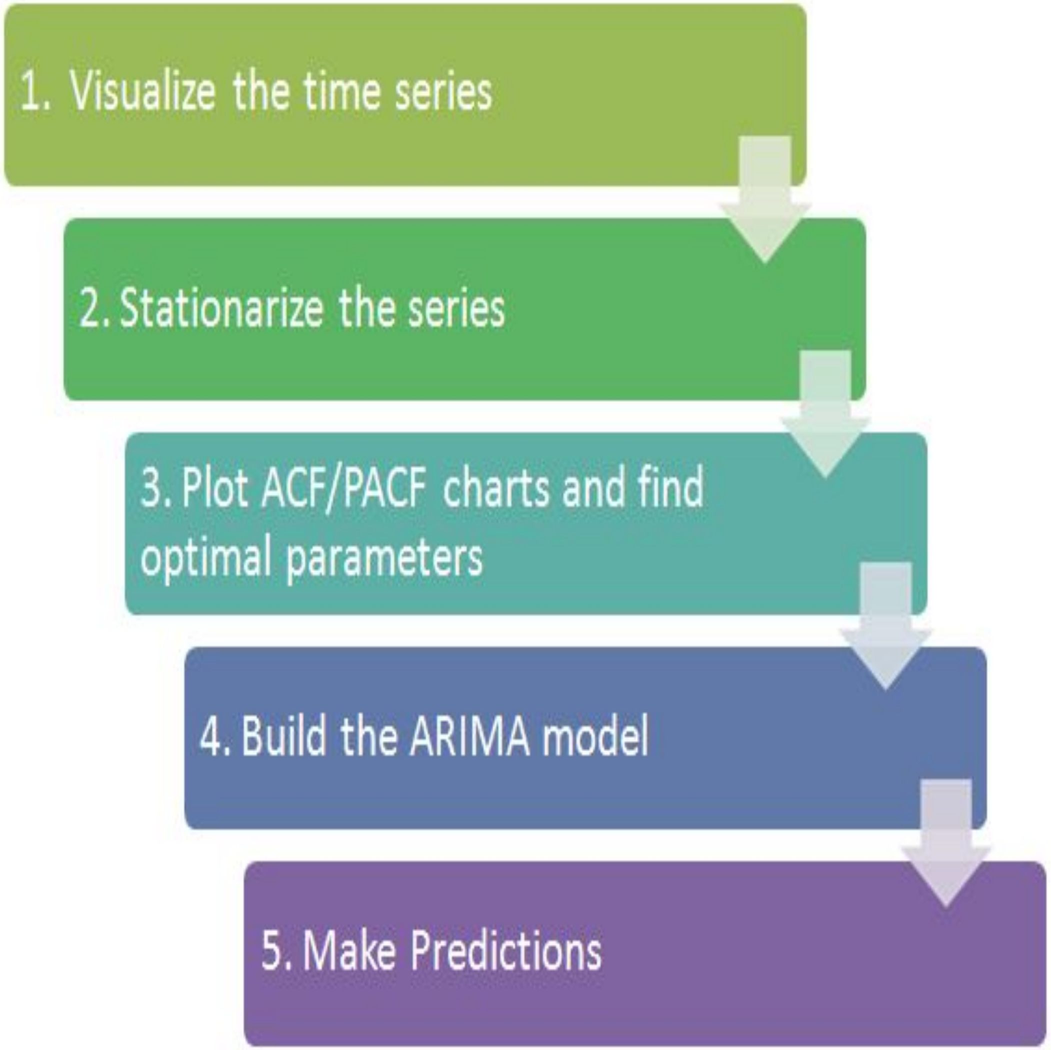
Our ARIMA Model: In Code

➔ **TensorFlow JS Implementation**
- **Pros:** Problem could be implemented as a neural network; Optimisations allow for faster processing (up to x20); which in turn encourages usage of more data records, plus allows for training on several loss functions.
- **Cons:** Still not fully deployed yet, Miss the integration part

Our ARIMA Model: In Code

```
function buildModel(inputShape, optimizer, loss, metrics) {
    // Define input, which has a size of inputShape
    const inputLayer = tf.input({ shape: inputShape });

    // Output dense layer uses linear activation.
    const denseLayer1 = tf.layers.dense({ units: 1 });

    // Obtain the output symbolic tensor by applying the layers on the inputLayer.
    const output = denseLayer1.apply(inputLayer);

    // Create the model based on the inputs.
    const model = tf.model({ inputs: inputLayer, outputs: output });

    model.compile({
        optimizer: optimizer,
        loss: loss,
        metrics: metrics,
    });
```

Framework Of Time Series ARIMA Modeling (Box-Jenkins)



Our ARIMA Model: Training Approach

We need to tune some hyperparameters to help the model adapt the data in an appropriate way, So how to choose correct value for:

- ➔ Epochs
- ➔ Learning Rate
- ➔ p
- ➔ d
- ➔ q

Our ARIMA Model: Training Approach

For tuning (p, d, q) there are some visual and statistical methods such as:

- ➔ Augmented Dickey–Fuller test (ADF)
- ➔ Kwiatkowski-Phillips-Schmidt-Shin test (KPSS)
- ➔ Autocorrelation Function (ACF)
- ➔ Partial Autocorrelation Function (PACF)

Our ARIMA Model: Evaluation Metrics

- ➔ Mean Squared Error (MSE)
- ➔ Root Mean Squared Error (RMSE)
- ➔ Log Likelihood
- ➔ Akaike's Information Criteria (AIC)
- ➔ Bayesian Information Criteria (BIC)

Our ARIMA Model: Evaluation Metrics

Metrics in mathematical forms

Log Likelihood

$$LL = -\frac{1}{2}\left(T\ln(2\pi\sigma^2) + \frac{1}{\sigma^2}\sum_{t=0}^{T}\varepsilon_t^2\right)$$

$$LL = -\frac{n}{2}ln(2\pi\sigma^2) - \frac{1}{2n\sigma^2}\sum_{i=0}^{n}\varepsilon^2$$

Akaike's Information Criteria

$$AIC = 2(p+q+k+1) - 2LL); k = 1 \text{ if } c \neq 0, k = 0 \text{ if } c = 0$$

Corrected AIC

$$AICc = AIC + \frac{2(p+q+k+2)(p+q+k+1)}{T-p-q-k-2}$$

Bayesian Information Criteria

$$BIC = AIC + (p+q+k+1)(ln(T) - 2)$$

Deployment

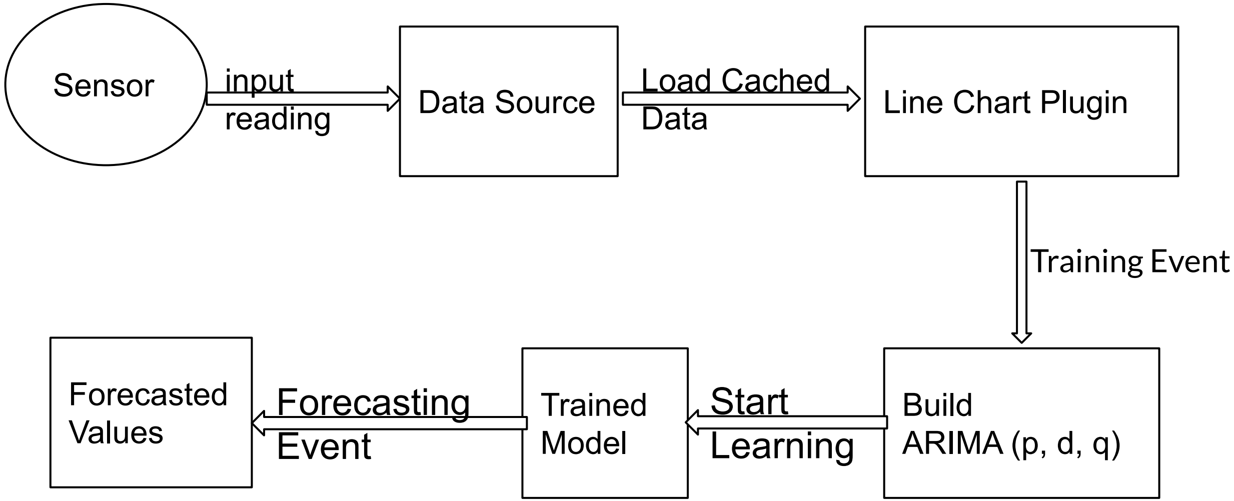
On

MoT

Deployment: Master of Things (MoT)

- ➔ It's an Internet of Things (IoT) platform enables you to create IoT & Machine-to-machine (M2M) applications to serve you in few hours.
- ➔ It'a product of **SpimeSenseLabs.**

Deployment: Architecture



Deployment: Events

- On Click Events
  - Training
  - Forecasting
  - Evaluation

- Custom Events

- Timer Tick Event

Deployment: Events

- ➔ **Onclick Training Event:**
  - Read sensor data.
  - Build the model with required hyperparameters from UI.
  - Start fitting the data.
  - Save the trained model after finishing fitting into JSON object containing all the learned params.

Deployment: Events

- **Onclick Forecasting Event:**
  - Read the number of needed predictionprediction steps.
  - Load the saved trained model.
  - Call forecast method.
  - Show the predictions in UI list.

Deployment: Events

- ➔ **Onclick Evaluate Event:**
  - ◆ Train on 90% of sensor data.
  - ◆ Forecast number of steps equals to 10% of sensor data
  - ◆ Call an evaluation matrix, such as: MSE.

Deployment: Performance

- ➔ Training Action:
  - ◆ Less than 2,000 ms

- ➔ Forecasting Action:
  - ◆ Less than 2,000 ms

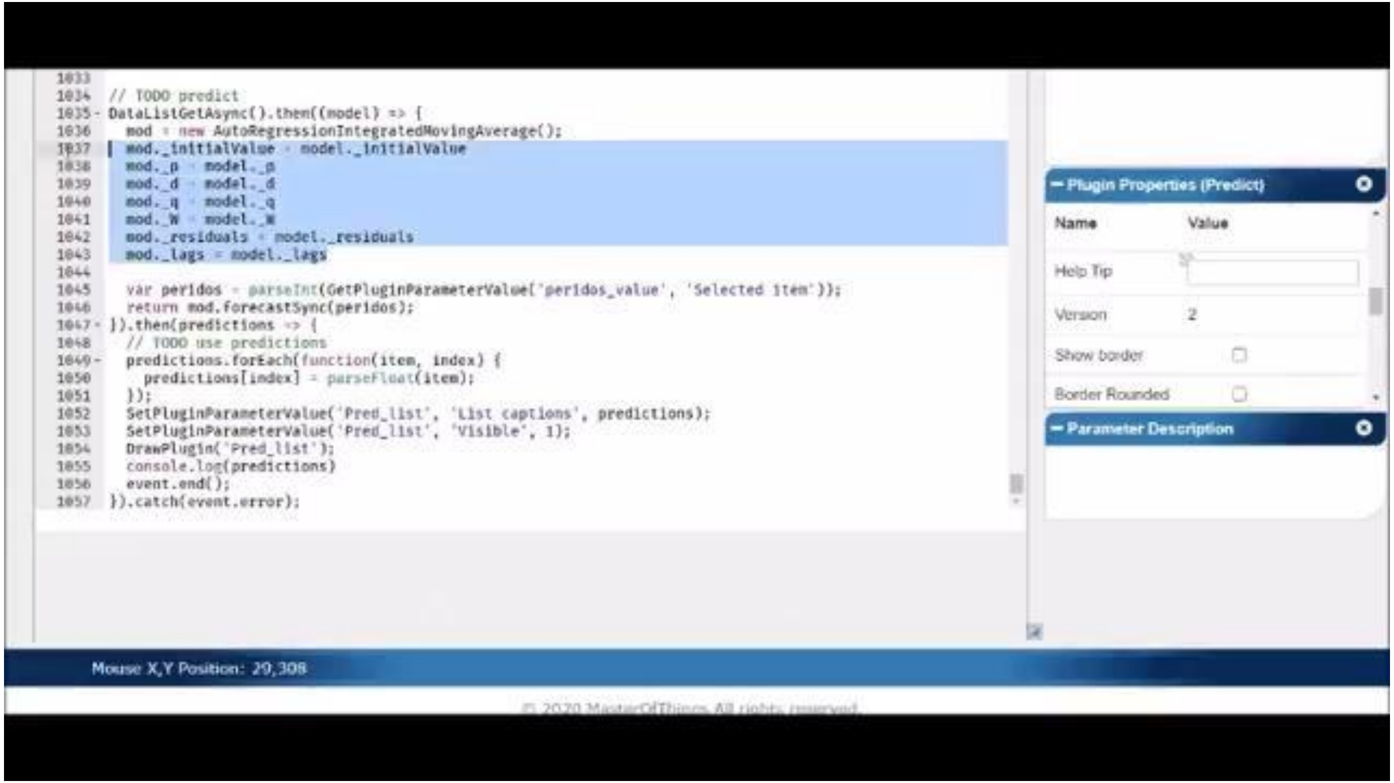
Deployment: Retraining Schedule

Based on readings frequency:

The higher the frequency of coming readings, the more retraining must be done, And vice versa...

Live Demo

Live Demo



// TODO predict
DataListGetAsync().then((model) => {
mod = new AutoRegressionIntegratedMovingAverage();
mod._initialValue = model._initialValue
mod._p = model._p
mod._d = model._d
mod._q = model._q
mod._W = model._W
mod._residuals = model._residuals
mod._lags = model._lags
var peridos = parseInt(GetPluginParameterValue('peridos_value', 'Selected item'));
return mod.forecastSync(peridos);
}).then(predictions => {
// TODO use predictions
predictions.forEach(function(item, index) {
predictions[index] = parseFloat(item);
});
SetPluginParameterValue('Pred_list', 'List captions', predictions);
SetPluginParameterValue('Pred_list', 'Visible', 1);
DrawPlugin('Pred_list');
console.log(predictions)
event.end();
}).catch(event.error);
Plugin Properties (Predict)
Name
Value
Help Tip
Version
2
Show border
Border Rounded
Parameter Description
Mouse X,Y Position: 29,308

Thank You