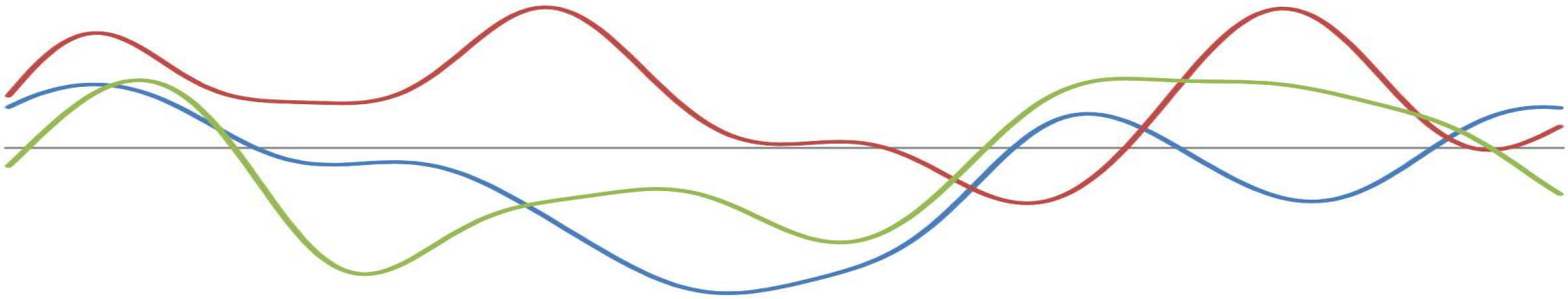
UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Campus de Terrassa

Advanced Methodologies for Time Series Analysis
From Preprocessing to Deep Learning for Event Detection and Variable Prediction

Carlos Cano Domingo
Computer Science - Universitat politécnica de catalunya
Dual Technology - Barcelona Supercomputing Center



Index

1. Presentation
2. Time-Series.
3. Forecasting in R
4. LSTM with pytorch Lightning.
5. AutoEncoder with PyTorch Lightning
6. Mlflow & RayTune
7. Transformers with DARTS

Slide 2

1 Carlos, 17/04/2023

A bit of Myself

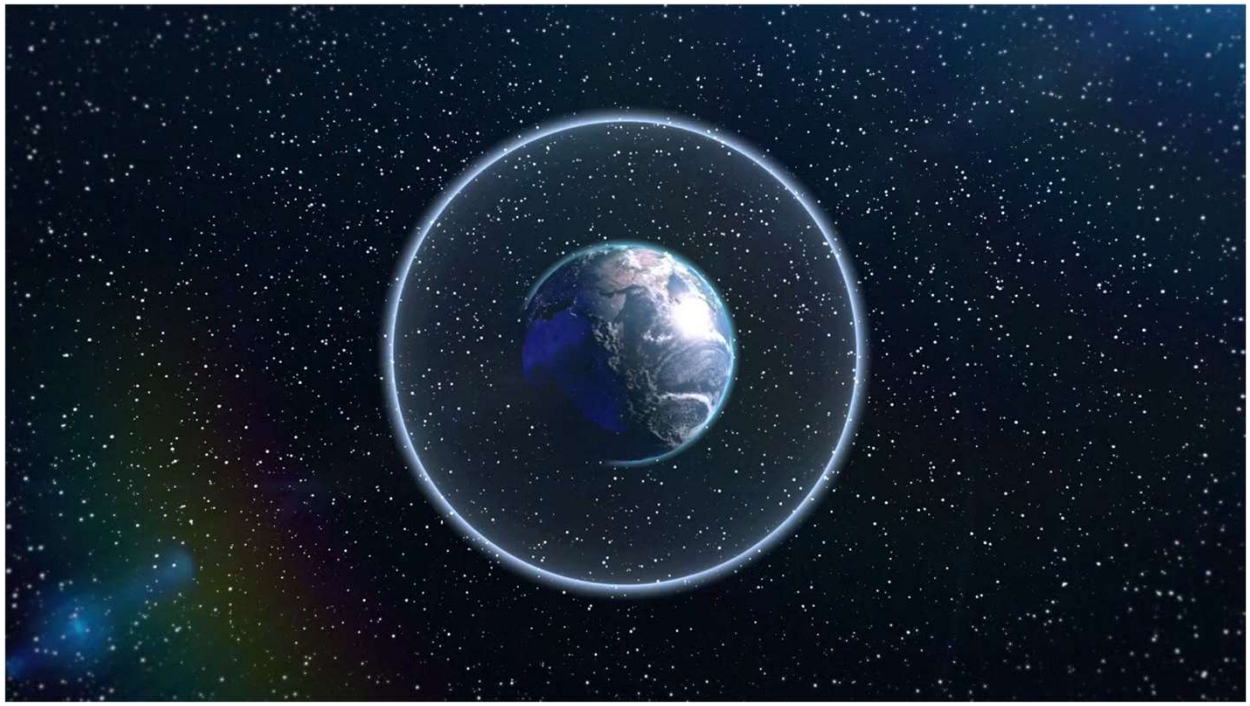
UNIVERSIDAD DE MÁLAGA

UNIVERSITATEA DIN CRAIOVA

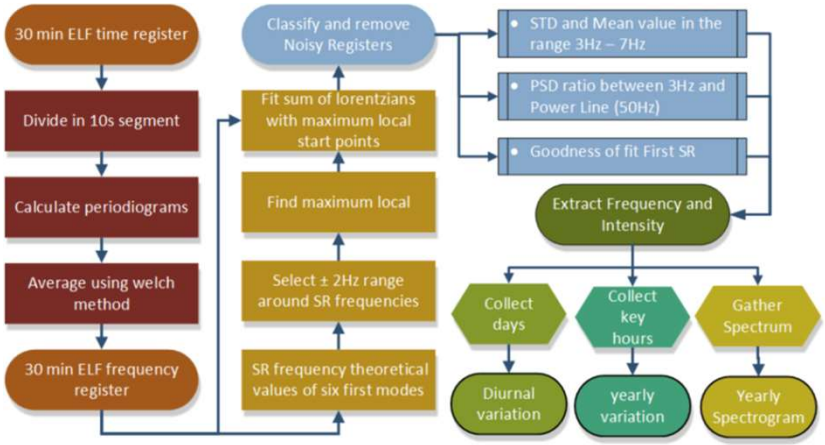
MANCHESTER 1824
The University of Manchester

UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH

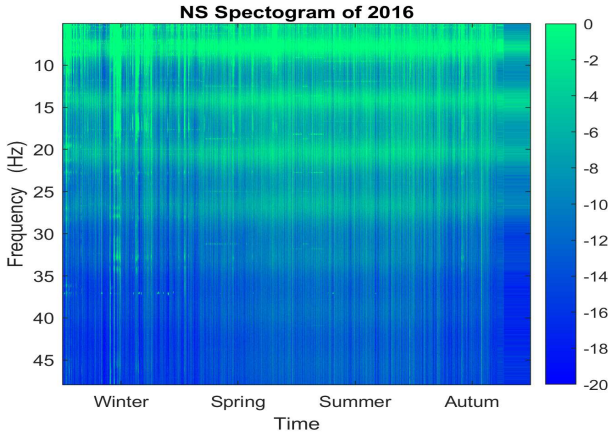
Barcelona Supercomputing Center
Centro Nacional de Supercomputación

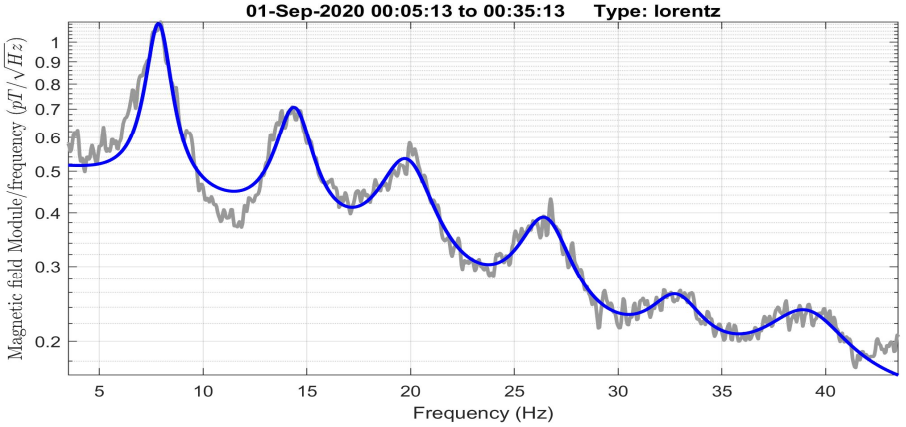


Classical processing signal applied to SR

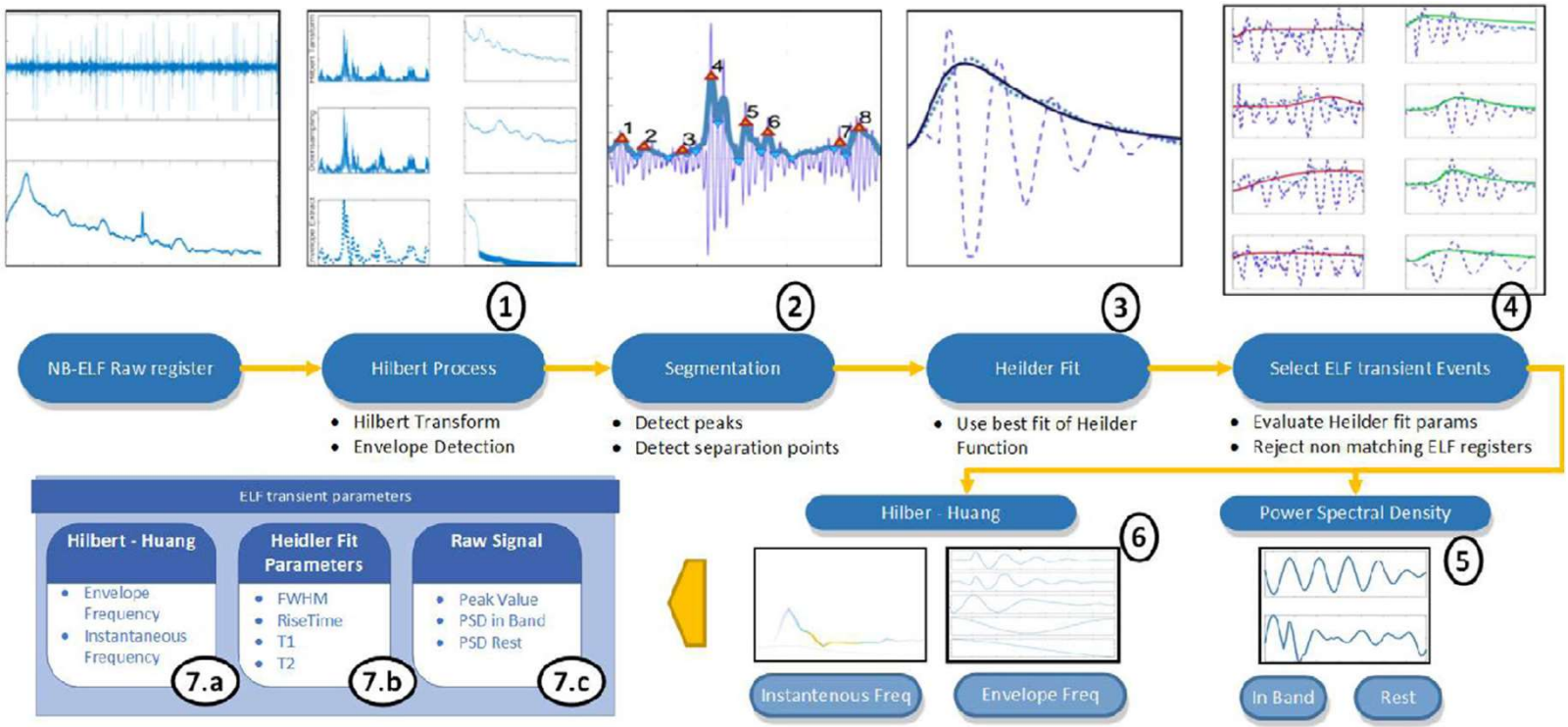


$$L(t) = \sum_{i=1}^{i=6} \left[\frac{Amplitude_i}{1 + \left(\frac{(t - Center_i)}{Width_i} \right)^2} \right] + k.$$

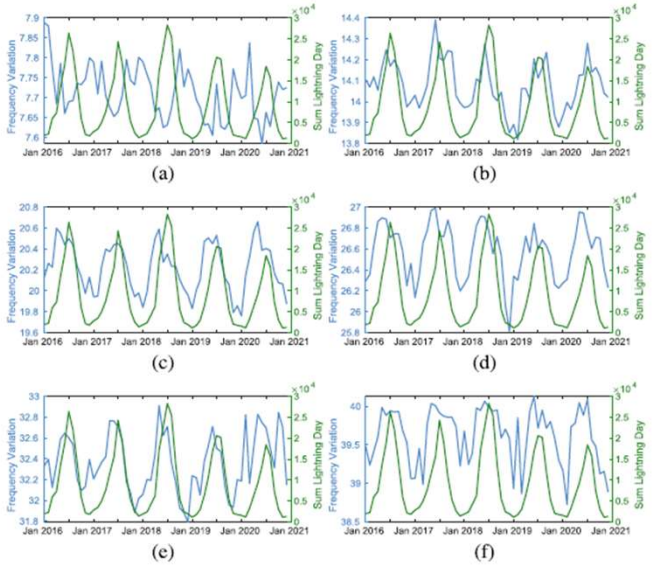


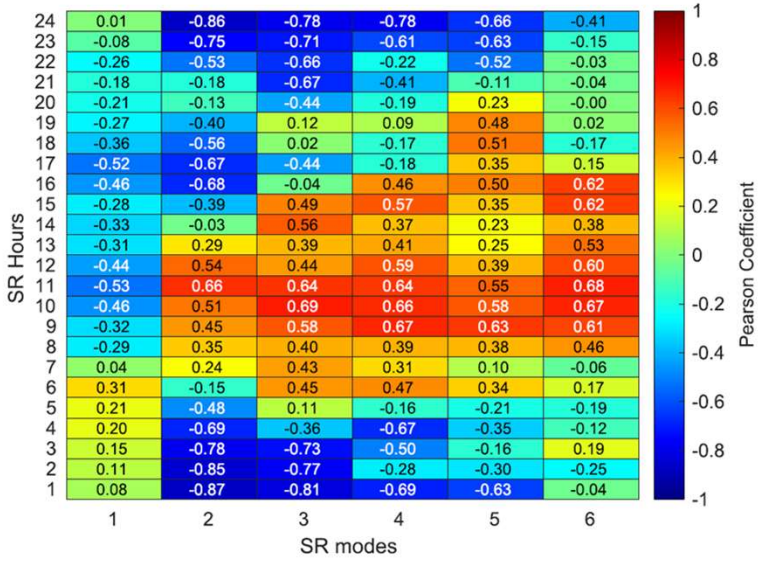


Classical processing signal applied to SR



Classical processing signal applied to SR

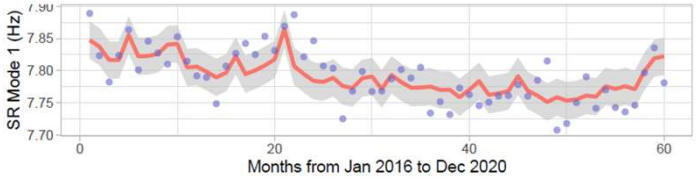


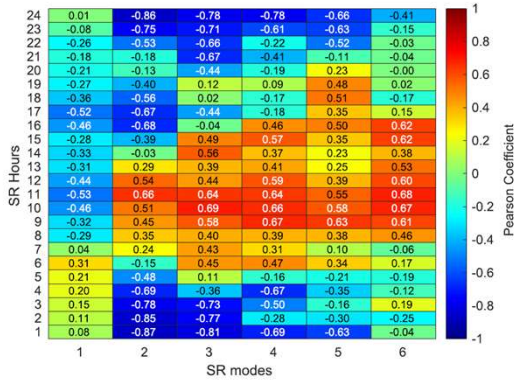


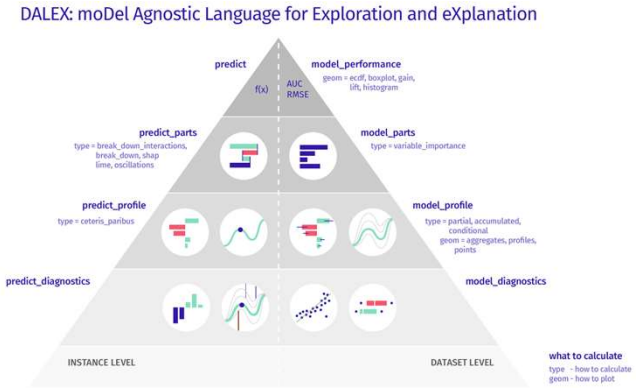
Classical processing signal applied to SR

1. Signal Processing: **MATLAB**
2. ML Models, Visualization & Statistical Analysis: **R**

- Complex description of the interaction between External Variables and SR modes for each hour.
- DALEX: moDel Agnostic Language for Exploration and eXplanation.
- External data as predictors, SR as the output.
- A composition of six machine Learning methods:
    1. Analytical methods: RIDGE linear regression and Multivariate Adaptive Regression Splines (MARS).
    2. Black-Box methods: Artificial Neural Network (ANN) and Support Vector Machines (SVM).
    3. Ensemble methods: Random Forests (RF) and Gradient Boosting Machines (GBM).
- Extract a ML explanatory metrics of each model.

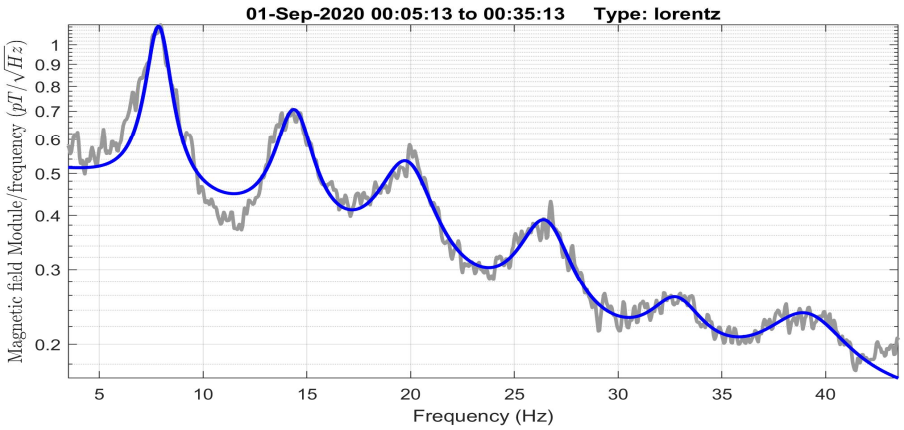


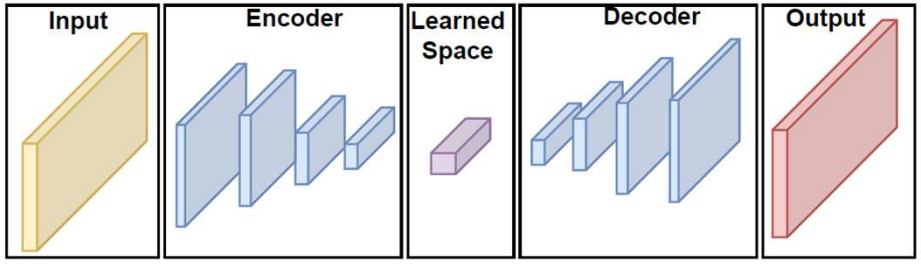


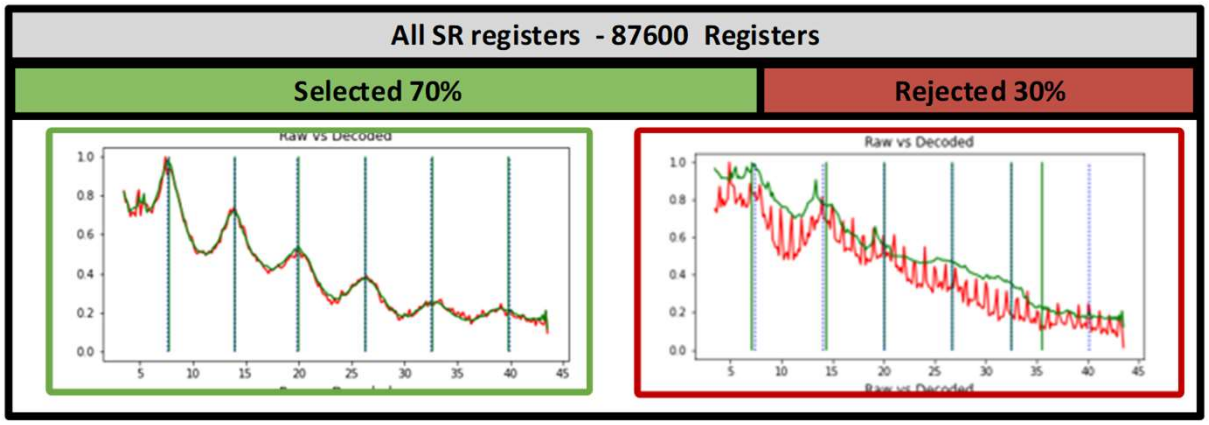


AutoEncoder for Signal processing

1. Signal Processing: **MATLAB**
2. DL Models: Python (Pytorch)
3. Visualization: **R**

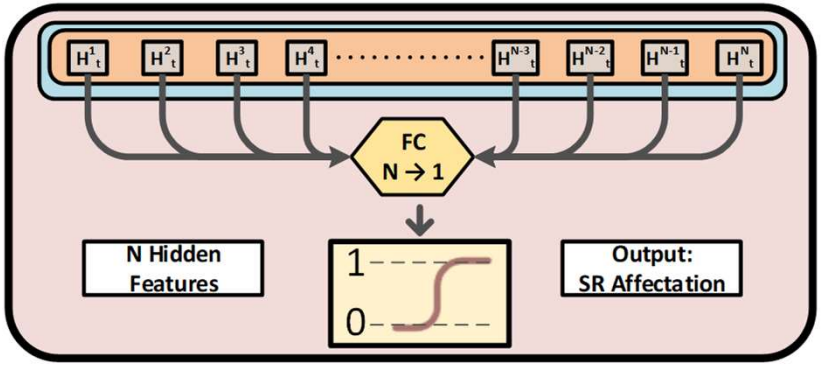


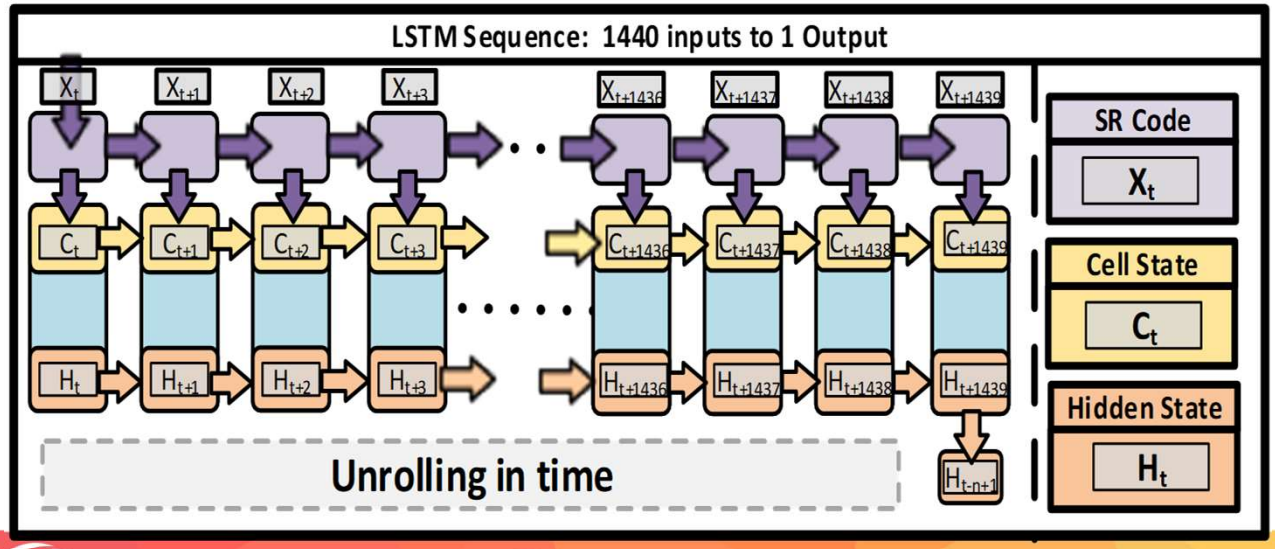




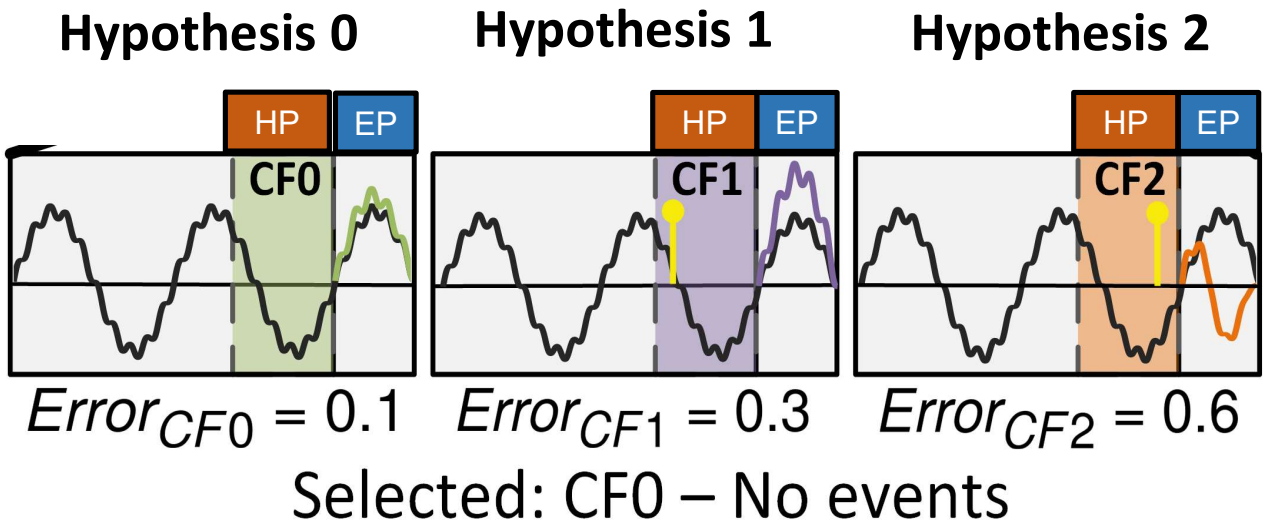
Deep Learning to EarthQuake detection

1. Signal Processing: **MATLAB**
2. DL Models: Python (Pytorch)
3. Visualization: **R**





1. Signal Processing: **MATLAB**
2. DL Models: Python (Dart)
3. Visualization: **R**
4. Hyperparameter Tunning**: R(irace)**



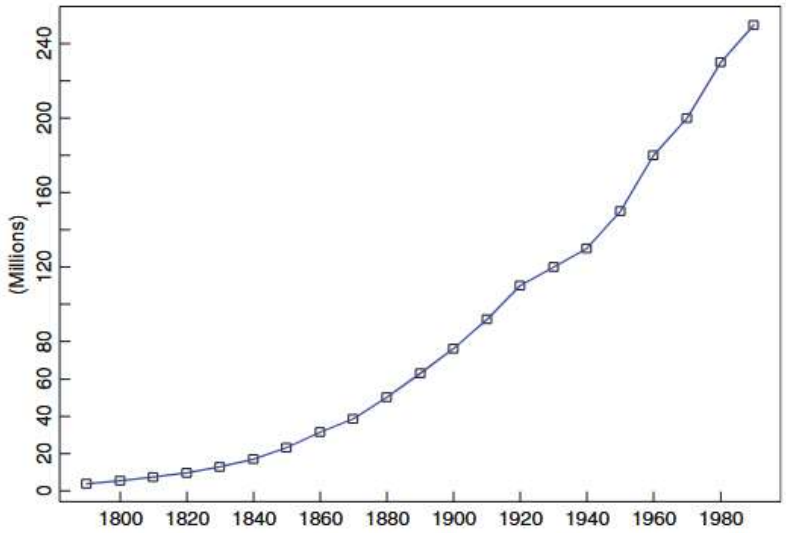
Event period **(EP)**: How long will **remain the effect** of the event.
Hypothesis period **(HP)**: How long takes the event to **make effect**.

2 TimeSeries

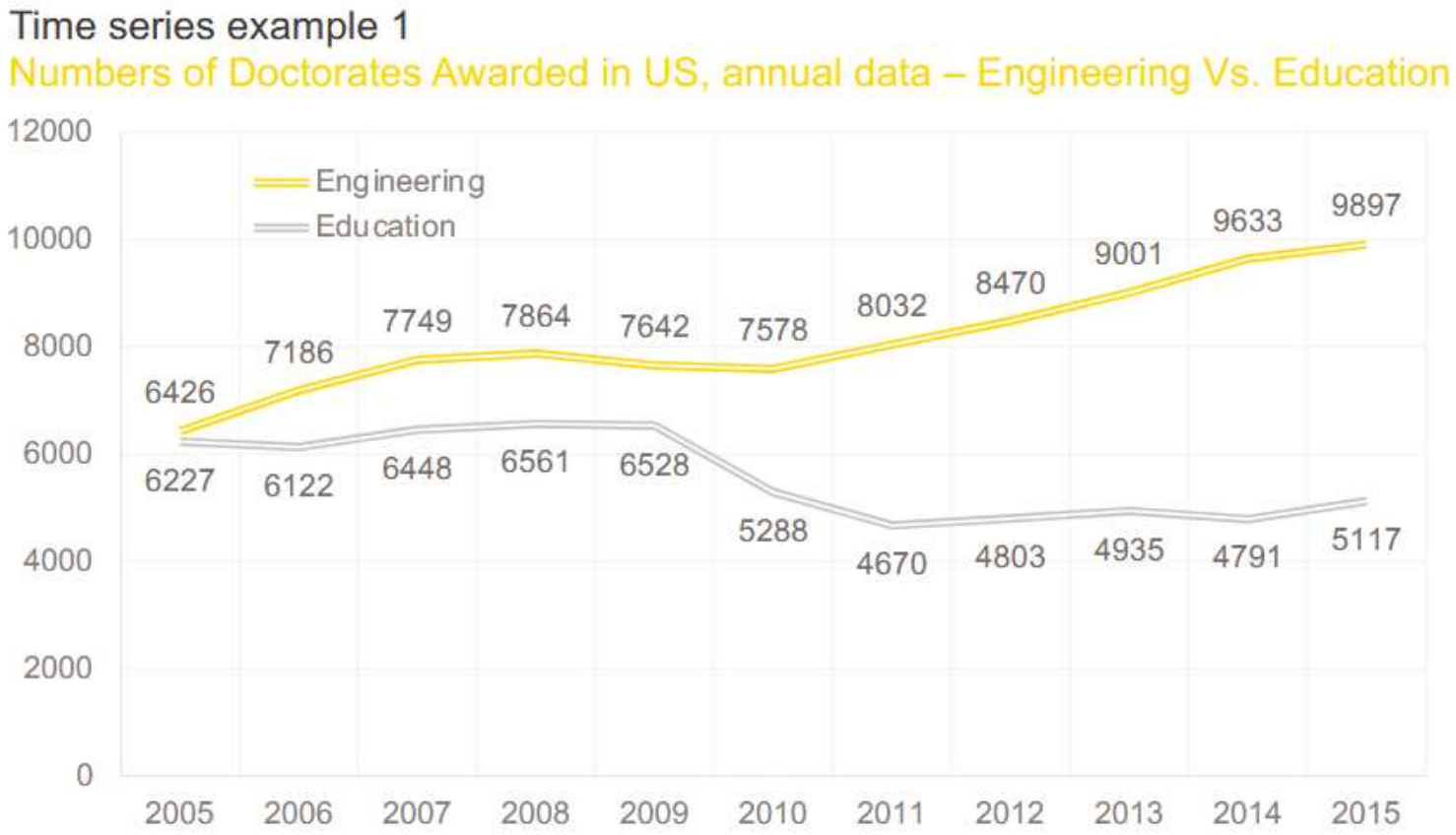
TimeSeries

A time series is a sequence of data points collected at regular time intervals.
Past observation often influence future ones.

1. **Temporal order**: Each observation is associated with a specific point in time and the order of these observations is critical.
2. **Regular Intervals**: Data is typically recorder at regular intervals, such as daily monthly or annually
3. **Pattern over the time**: Time series often exhibit identifiable patterns.



Population of the U.S.A., 1790–1990

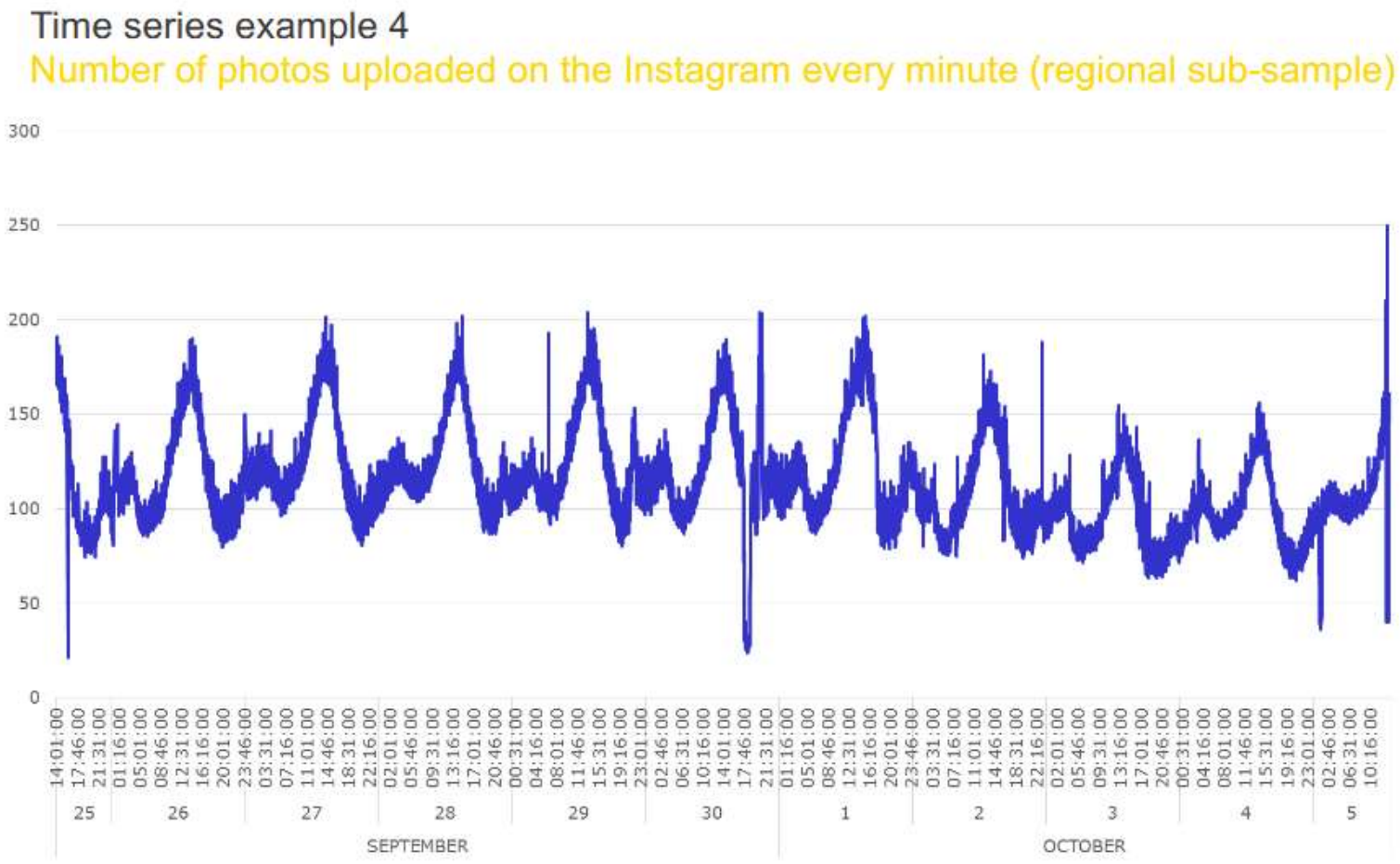


Pictures from KNIME AG.

Pictures from KNIME AG.

Pictures from KNIME AG.

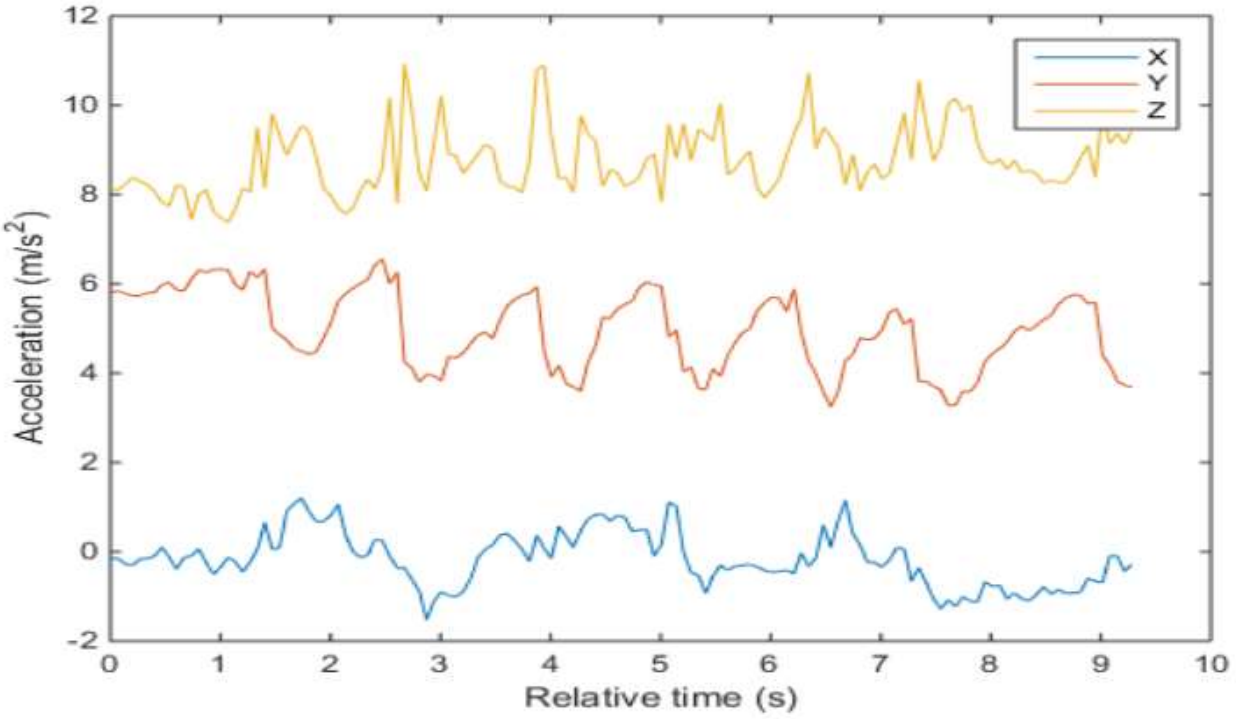
TimeSeries



Pictures from KNIME AG.

Time series example 5

Acceleration detected by a smartphone sensors during a workout session (10 seconds)



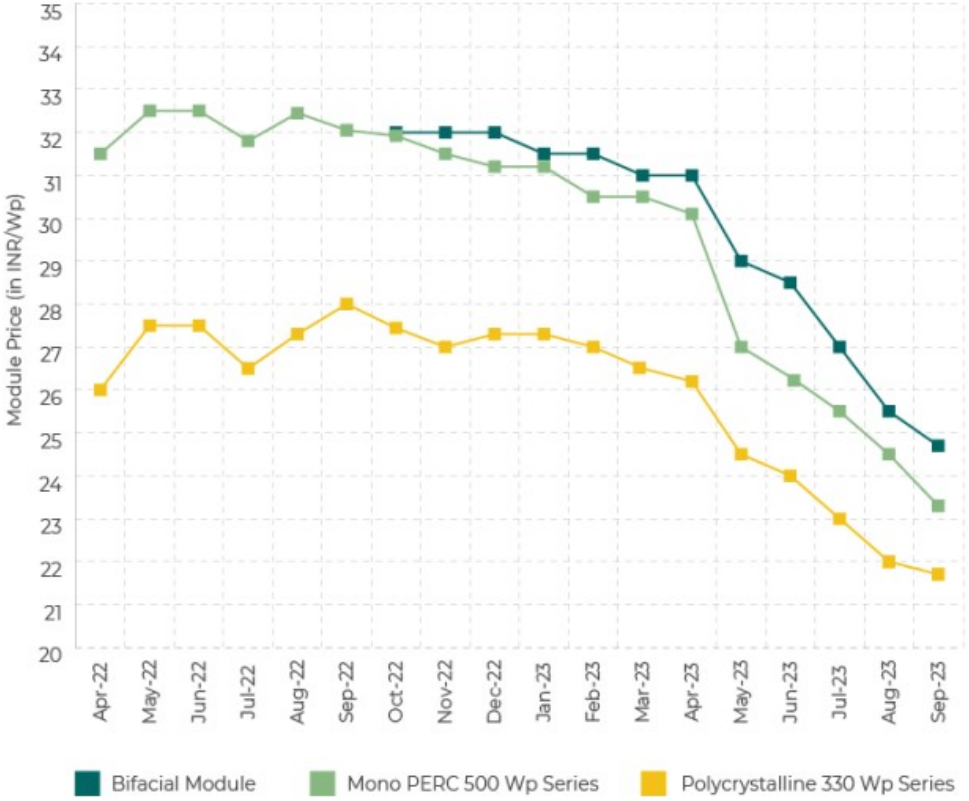
At a glance

Milliseconds basis data

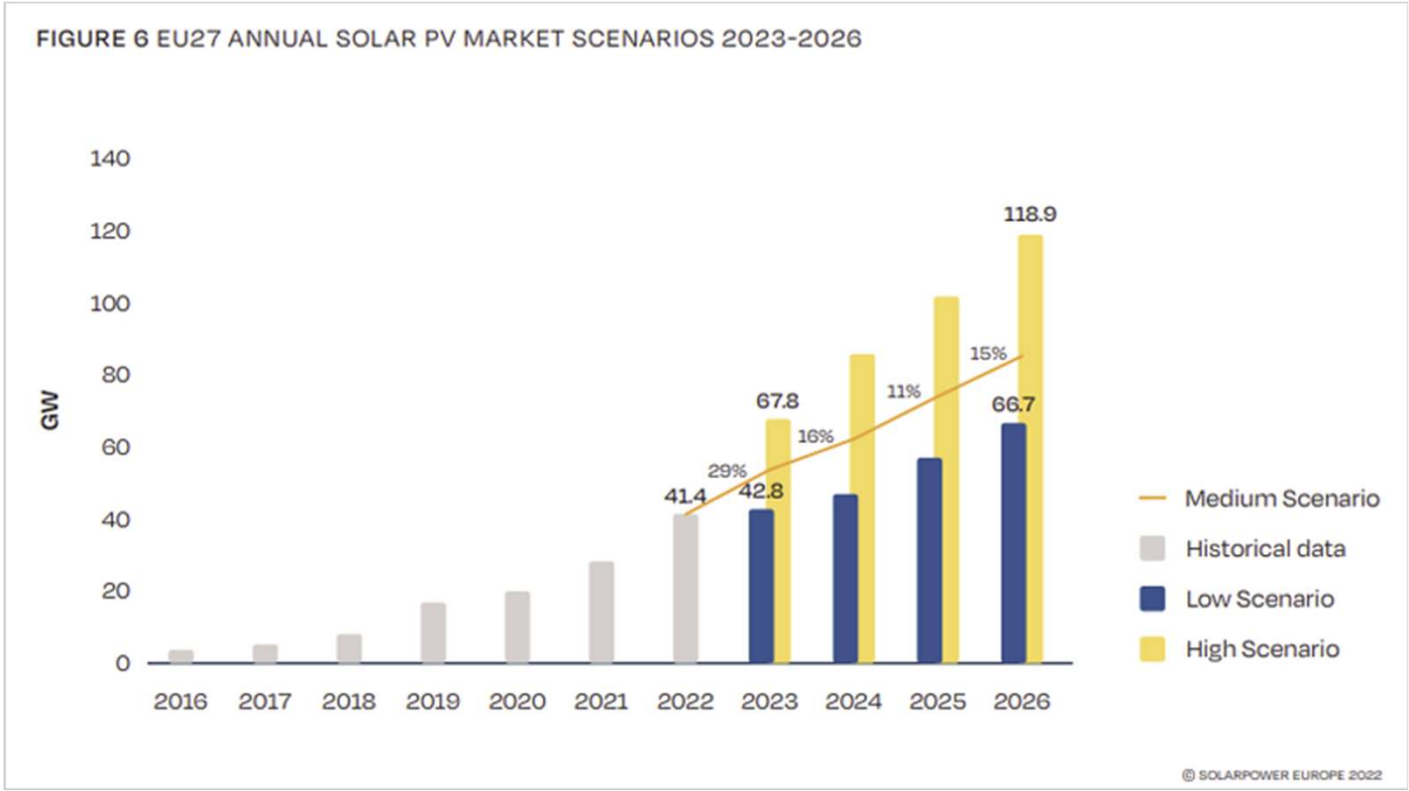
Each sensor has its own dynamics

Pictures from KNIME AG.

TimeSeries



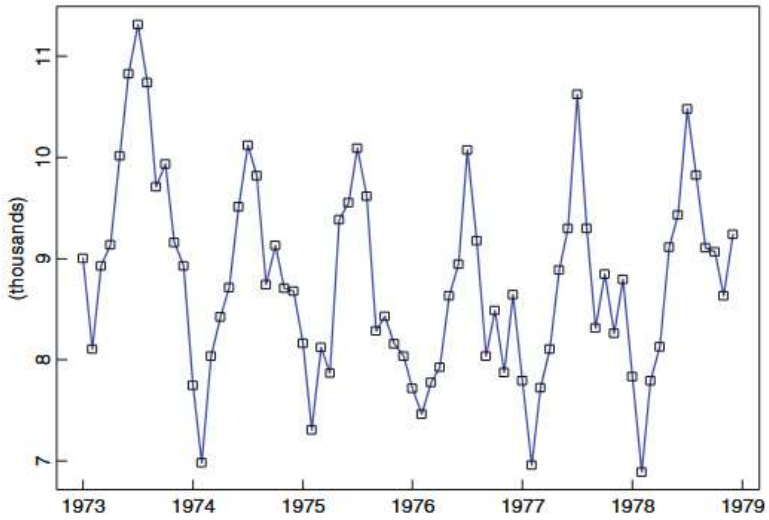
Pictures from KNIME AG.



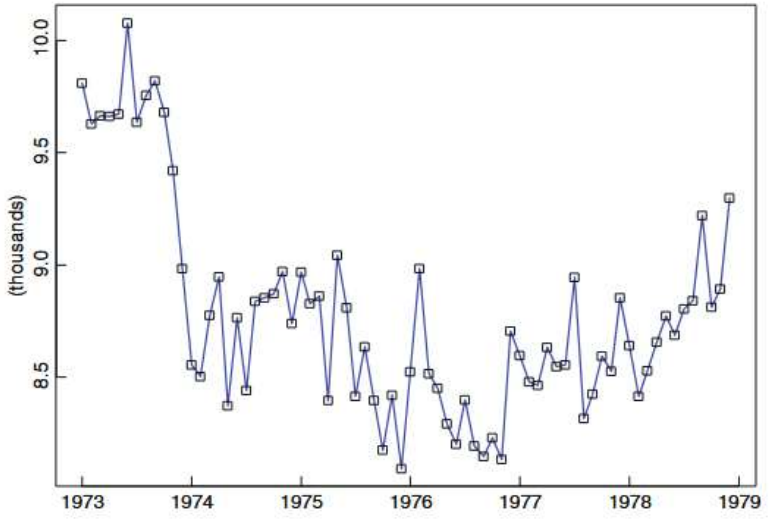
Pictures from KNIME AG.

TimeSeries

Time **series analysis seeks** to find the change in data **patterns** over different time periods.
This provides a better picture of how the **data trends** move, allowing a user to develop a **model based** on these observations that captures most of the trends of the data.



The monthly accidental deaths data, 1973–1978



Deseasonalized - The monthly accidental deaths data, 1973–1978

Pictures from https://unit8co.github.io

TimeSeries

Uses of Time Series

- **Forecasting**: Predict **future values** based on historical data: Predicting future sales, stock price, energy demand.
- **Pattern identification**: Helps to **identify recurrent pattern** like trend and seasonality which are useful for making informed decision.
- **Monitoring and control**: Used in industries like engineering to monitor systems in real item and **detect anomalies** or failures.
- **Causal Analysis**: Can be used to understand what factors drive changes in a variable over time.

KNIME AG.

Typical Pattern

- **Trend**: Long/term upward or downward movement in the data.
- **Seasonality**: A repeating patterh that occurs at regular intervals such us daily, monthly or annually.
- **Cyclical pattern**: Simular to seasonality but without fixed period.
- **Noise**: Don’t follow a discernible pattern.

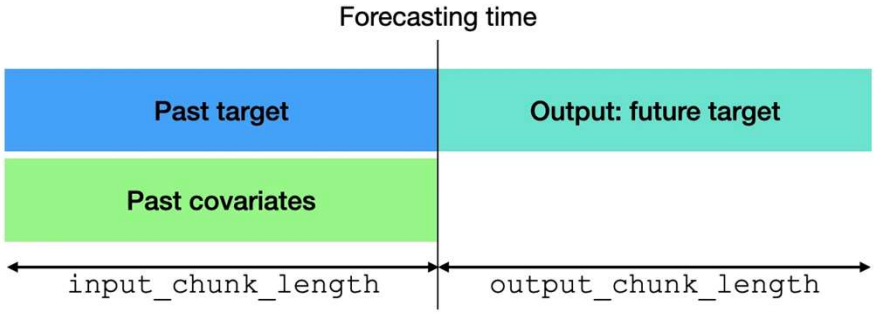
KNIME AG.

Forecasting Techniques:

- **Classical Statistical Models**: Traditional techniques include ARIMA (AutoRegresive Integrated Moving Average), SARIMA, Exponential Smoothing…
- **Deep Learning Approaches**: NeuralNetwork such as LSTM and Transformers. These models can capture more complex pattern such as long-term dependencies and non linear relationship.
- **Autoencoder**: Used for dimensionaly reduction and anomaly detection. These models learn an efficient representation of the input data.
- **Hybrid models**: Combining statistical method with machine learning approaches.

KNIME AG.

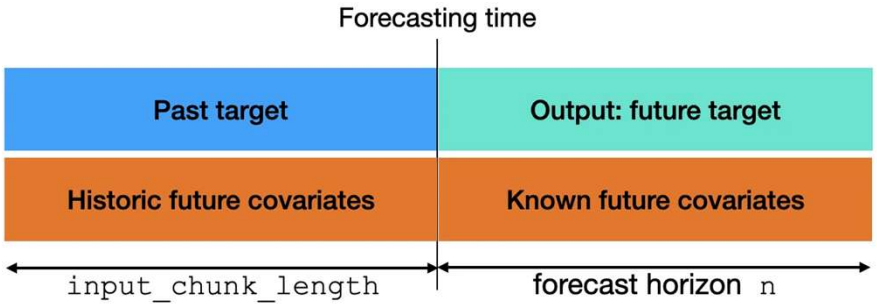
Only historical exogenous data



Inputs
- Past targets
- Past covariates

Outputs
- Future targets

With future exogenous data



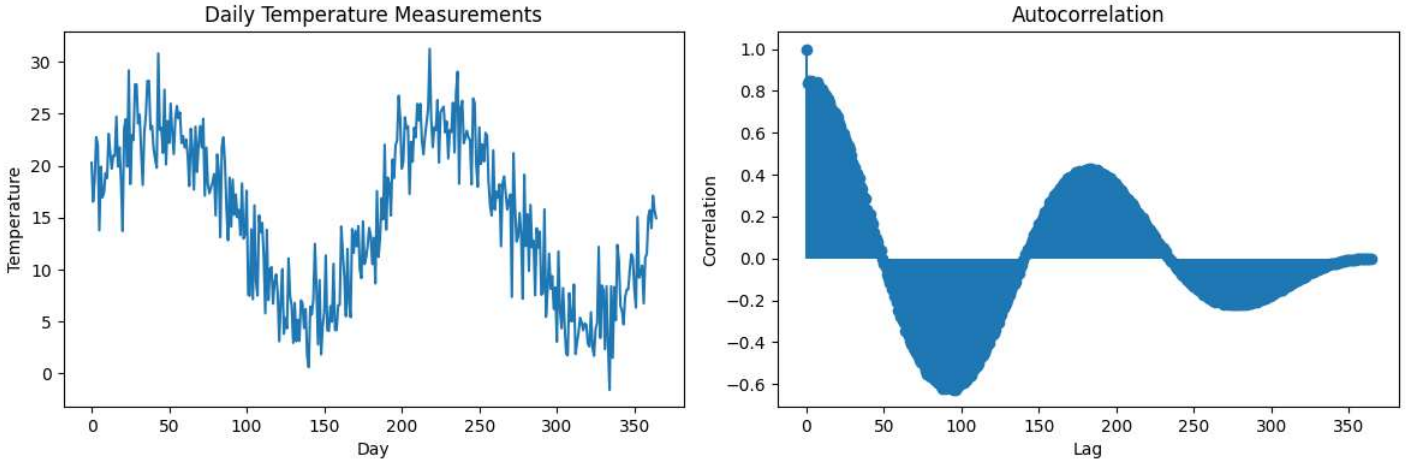
Inputs
- Past targets
- Historic future covariates
- Future covariates

Outputs
- Future targets

Pictures from https://unit8co.github.io

1. Autocorrelation:

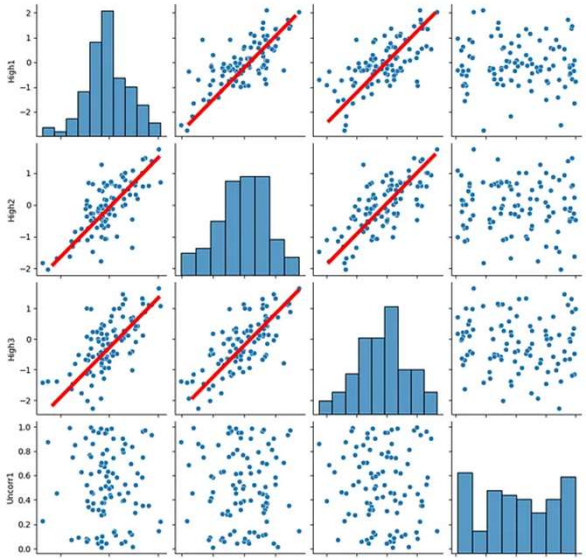
- **Description**: This occurs when the values in a time series are correlated with their own past values. This can make modelling difficult with traditional methods that assume independence between observations.
- **Problem**: It can lead to incorrect conclusions if not addressed properly. Residuals of a poorly fitted model may show autocorrelation, indicating that the model doesn't capture all the information present in the time series.



Picture from https://www.scicoding.com

2. Collinearity (Multicollinearity):

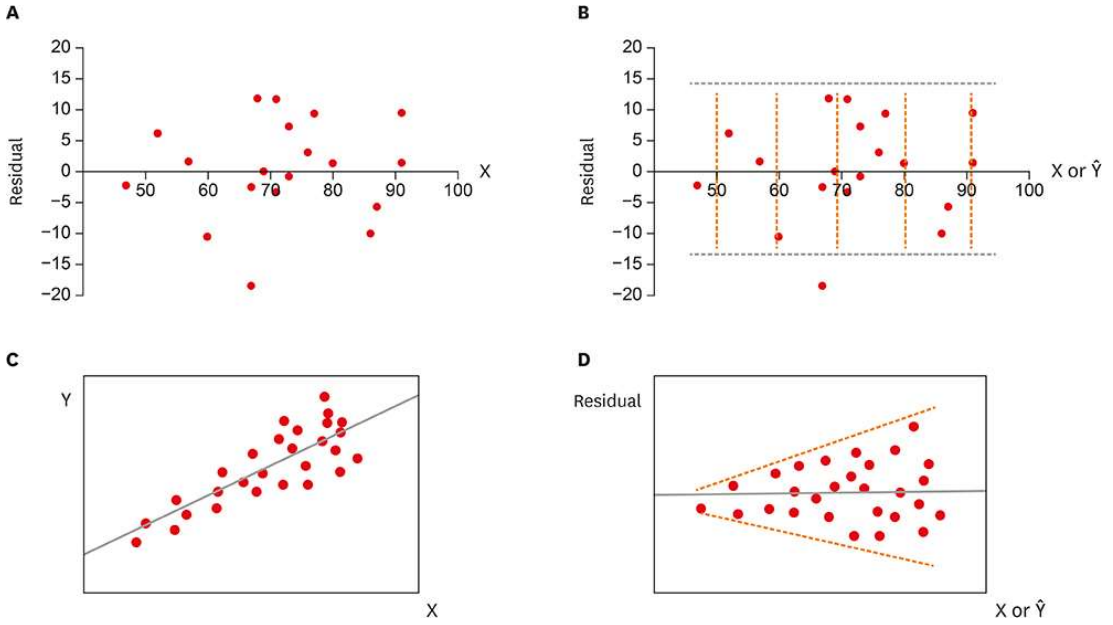
- **Description**: Occurs when two or more explanatory variables in a model are highly correlated with each other. In time series, this can happen if different variables have similar trends over time.
- **Problem**: Multicollinearity makes it hard to precisely estimate the coefficients of the variables in models, potentially leading to unstable predictions. It can also make it difficult to interpret the effects of variables.



Picture from https://medium.com

3. Heteroscedasticity:

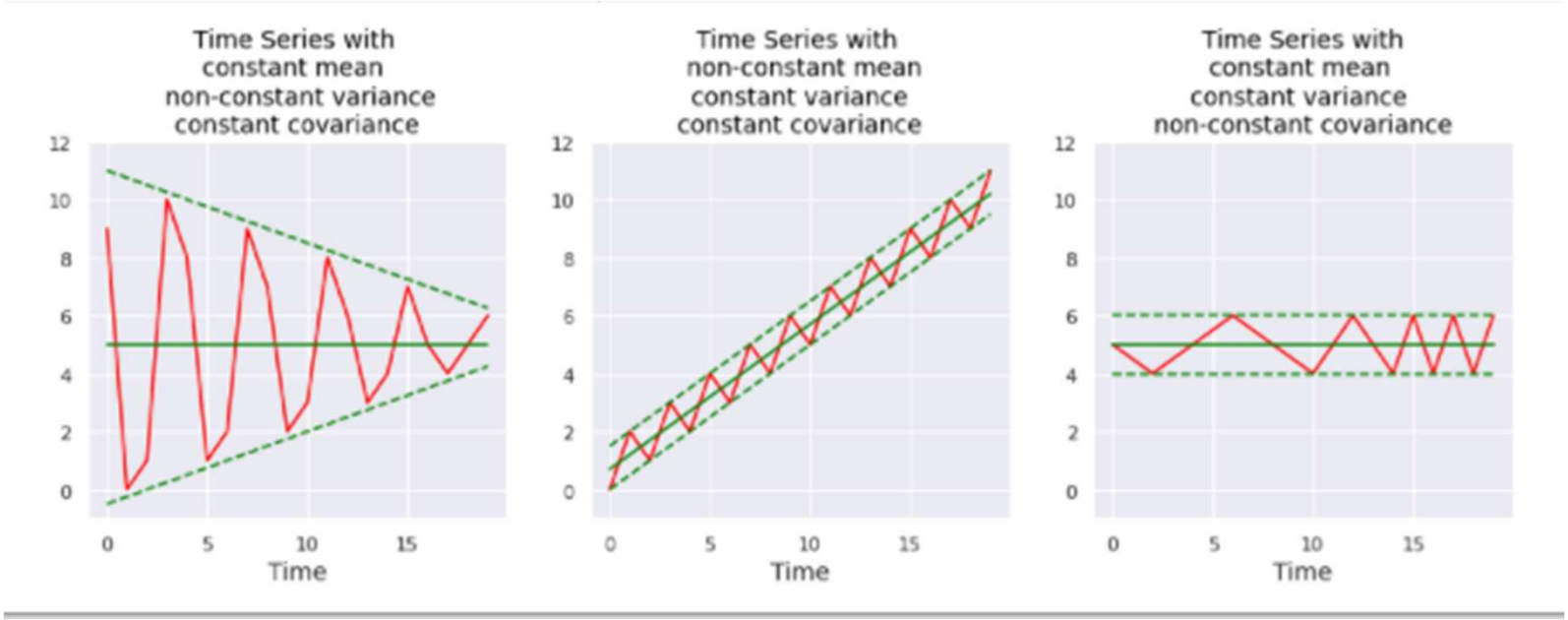
- **Description**: Refers to the presence of non-constant variance in the residuals of a model. In other words, the prediction errors have varying spread over time.
- **Problem**: When heteroscedasticity is present, it can violate the assumptions of many statistical models like linear regression, leading to incorrect standard errors and thus wrong inferences.



Picture from www.rde.ac

4. Stationarity:

- **Description**: A non-stationary time series has statistical properties such as mean and variance that change over time.
- **Problem**: Many time series analysis methods, like ARIMA, assume that the series is stationary. If the series is non-stationary, the model results may not be reliable.



Picture from www.rde.ac

3 Forecasting with R

Forecasting in R

Forecasting is about **predicting the future** as accurately as possible, given all of the information available, including **historical data** and knowledge of **any future events** that might impact the forecasts.

Goals are what you would like to have happen. Goals should be linked to **forecasts and plans**, but this does not always occur. Too often, goals are set without any plan for how to achieve them, and no forecasts for whether they are realistic.

Planning is a response to forecasts and goals. Planning involves determining the appropriate actions that are required to make your forecasts match your goals.

Forecasting should be an integral part of the decision-making activities of management, as it can play an **important role** in many areas of a company. Modern organisations require short-term, medium-term and long-term forecasts, depending on the specific application.

Forecasting in R

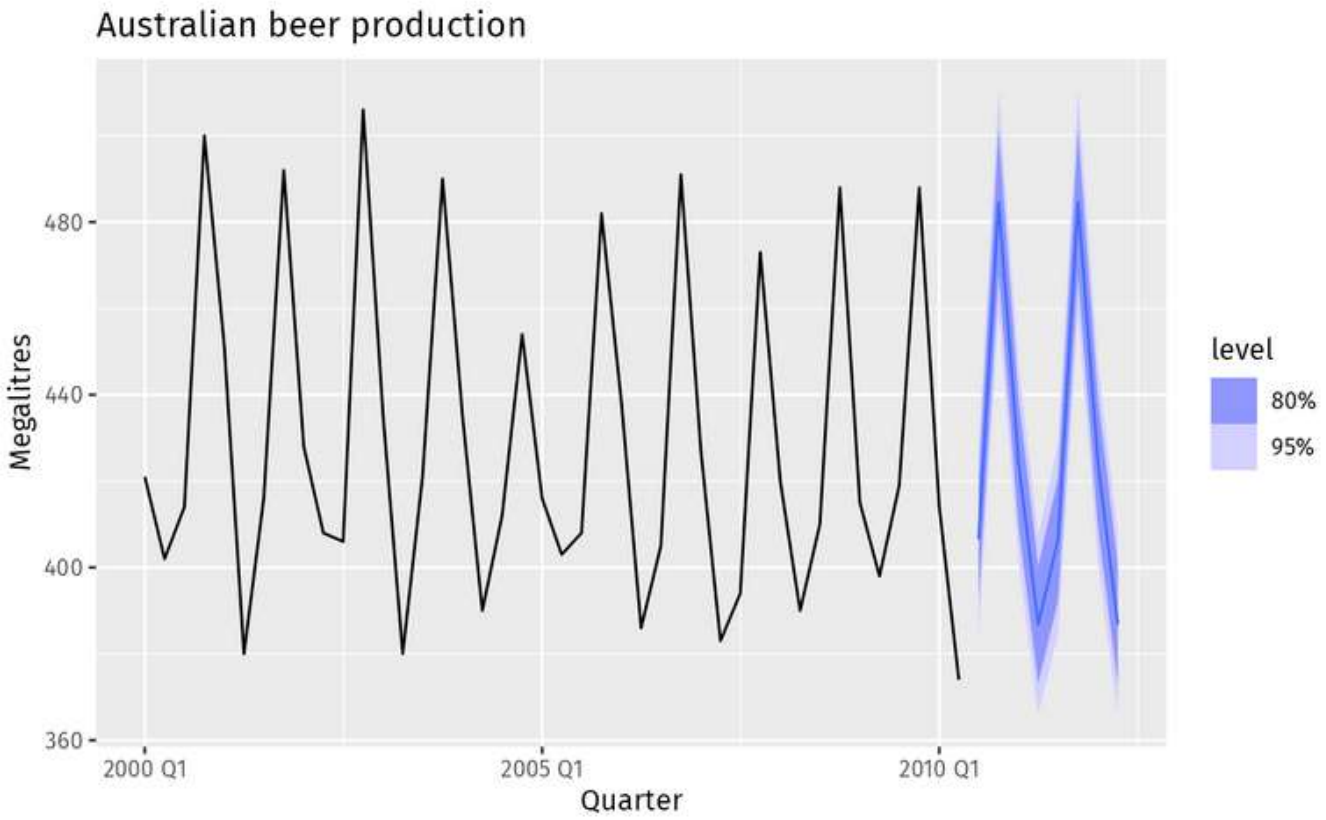
Short-term forecasts are needed for the scheduling of personnel, production and transportation. As part of the scheduling process, forecasts of demand are often also required.

Medium-term forecasts are needed to determine future resource requirements, in order to purchase raw materials, hire personnel, or buy machinery and equipment.

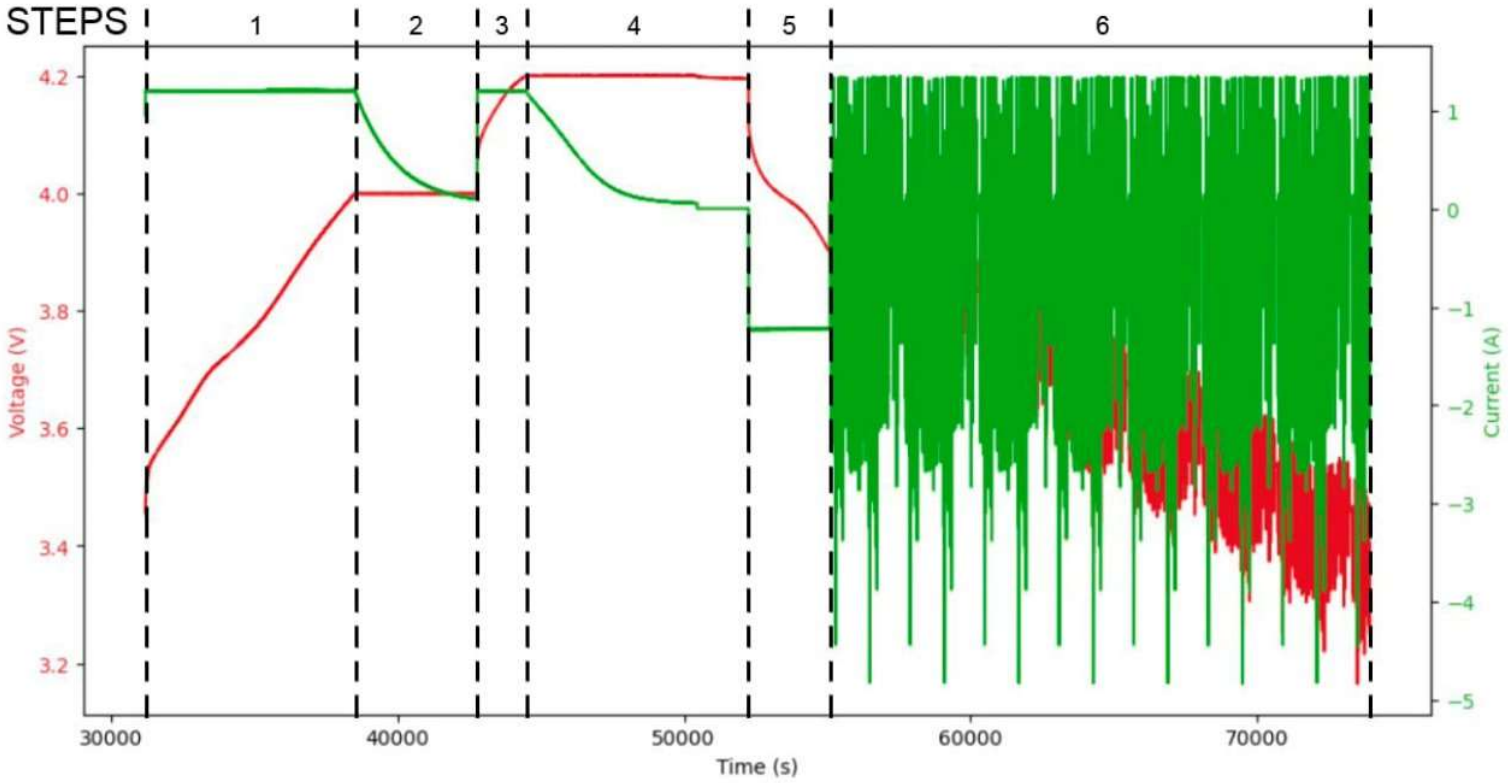
Long-term forecasts are used in strategic planning. Such decisions must take account of market opportunities, environmental factors and internal resources.

Examples of time series data include:

- Annual Google profits
- Quarterly sales results for Amazon
- Monthly rainfall
- Weekly retail sales
- Daily IBM stock prices
- Hourly electricity demand
- 5-minute freeway traffic counts
- Time-stamped stock transaction data

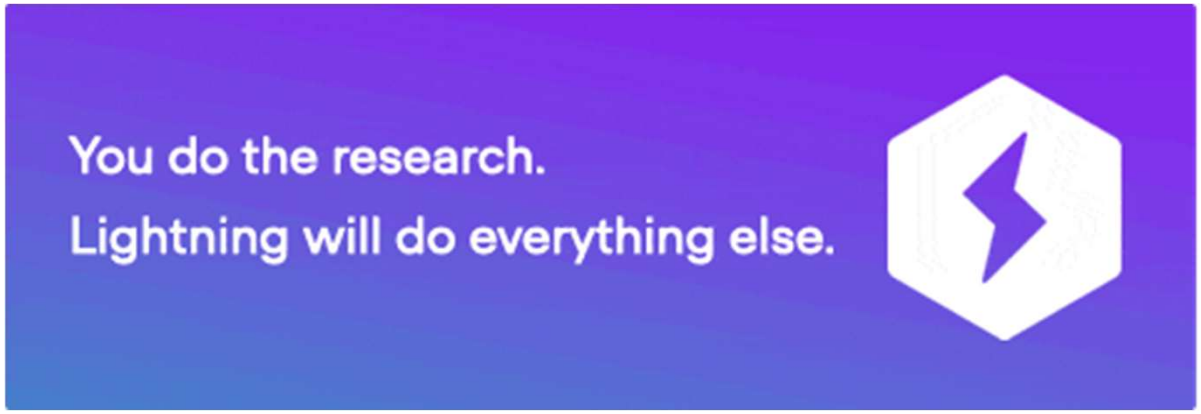


Forecasting in R



4 LSTM Prediction with Pytorch Lightning

Pytorch Lightning



Advantages

1. Organized Code Structure. (LightningModule & LightningModule)
2. Automatic Hardware Management (DataParallel & DistributedDataParallel)
3. Integration with External Tools (mlflow & RayTune)

Pytorch Lightning

PYTORCH

```
# models
encoder = nn.Sequential(nn.Linear(28 * 28, 64), nn.ReLU(), nn.Linear(64, 3))
decoder = nn.Sequential(nn.Linear(3, 64), nn.ReLU(), nn.Linear(64, 28 * 28))

encoder.cuda(0)
decoder.cuda(0)

# download on rank 0 only
if global_rank == 0:
    mnist_train = MNIST(os.getcwd(), train=True, download=True)

# split dataset
transform=transforms.Compose([transforms.ToTensor(),
                              transforms.Normalize(0.5, 0.5)])
mnist_train = MNIST(os.getcwd(), train=True, download=True, transform=transform)

# train (55,000 images), val split (5,000 images)
mnist_train, mnist_val = random_split(mnist_train, [55000, 5000])

# The dataloaders handle shuffling, batching, etc...
mnist_train = DataLoader(mnist_train, batch_size=64)
mnist_val = DataLoader(mnist_val, batch_size=64)

 # optimizer
 params = [encoder.parameters(), decoder.parameters()]
 optimizer = torch.optim.Adam(params, lr=1e-3)

# TRAIN LOOP
model.train()
num_epochs = 1
for epoch in range(num_epochs):
  for train_batch in mnist_train:
    x, y = train_batch
    x = x.cuda(0)
    x = x.view(x.size(0), -1)
    z = encoder(x)
    x_hat = decoder(z)
    loss = F.mse_loss(x_hat, x)
    print('train loss: ', loss.item())

    loss.backward()
    optimizer.step()
    optimizer.zero_grad()

# EVAL LOOP
model.eval()
with torch.no_grad():
  val_loss = []
  for val_batch in mnist_val:
    x, y = val_batch
    x = x.cuda(0)
    x = x.view(x.size(0), -1)
    z = encoder(x)
    x_hat = decoder(z)
    loss = F.mse_loss(x_hat, x)
    val_loss.append(loss)
  val_loss = torch.mean(torch.tensor(val_loss))
  model.train()
```

PYTORCH LIGHTNING

Turn PyTorch into Lightning

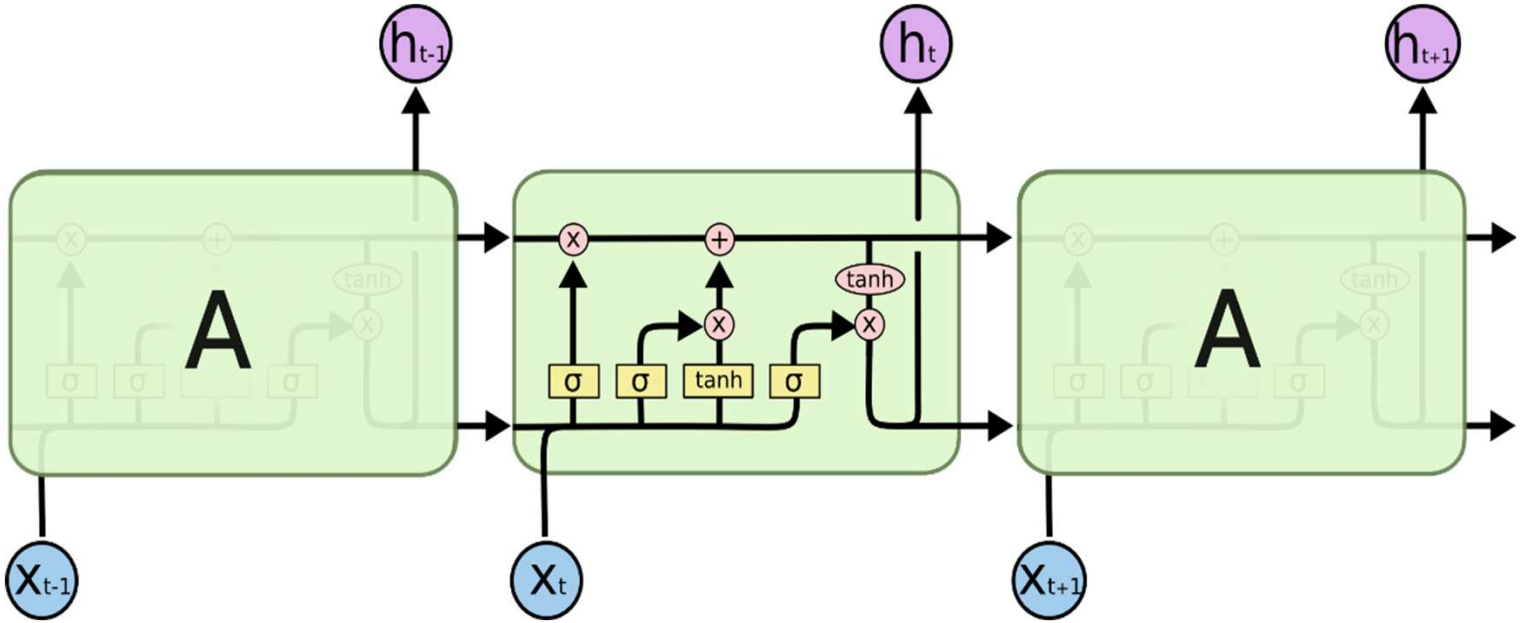
Lightning is just plain PyTorch

Pytorch Lightning

PyTorch

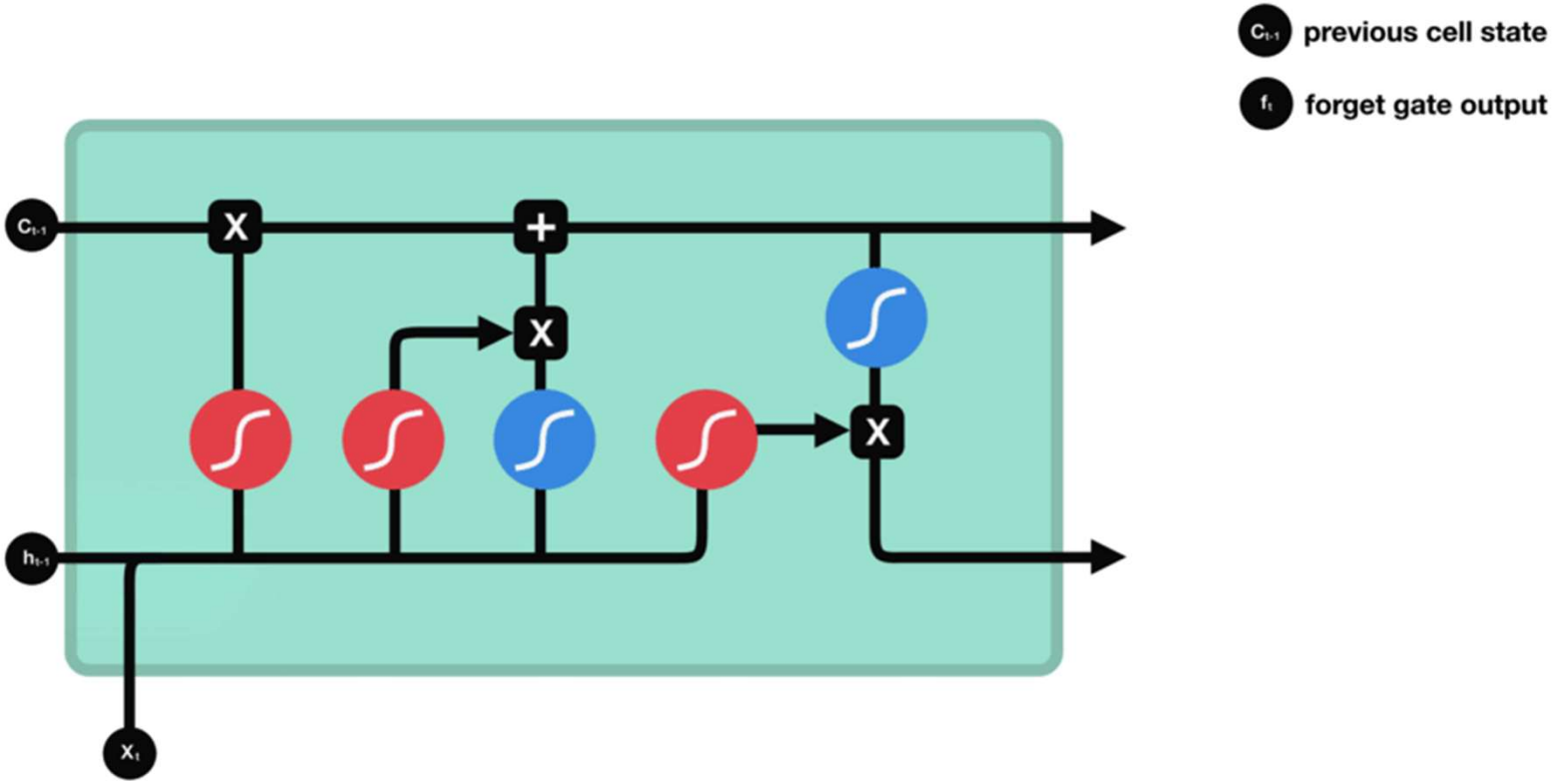
```
class MNISTClassifier(nn.Module):

  def __init__(self):
      self.layer_1 = torch.nn.Linear(28 * 28, 128)
      self.layer_2 = torch.nn.Linear(128, 10)

  def forward(self, x):
    x = x.view(x.size(0), -1)
    x = self.layer_1(x)
    x = F.relu(x)
    x = self.layer_2(x)
    return x

# download data
if global_rank == 0:
  mnist_train = MNIST(os,getcwd(), train=True, download=True)
  mnist_test = MNIST(os.getcwd(), train=False, download=True)

dist.barrier()

# transforms
transform=transforms.Compose([transforms.ToTensor(),
                              transforms.Normalize((0.1307,), (0.3081,))])
mnist_train = MNIST(os.getcwd(), train=True, transform=transform)
mnist_test = MNIST(os.getcwd(), train=False, transform=transform)

# split dataset
mnist_train, mnist_val = random_split(mnist_train, [55000, 5000])
mnist_test = MNIST(os.getcwd(), train=False, download=True)

# build dataloaders
mnist_train = DataLoader(mnist_train, batch_size=64)
mnist_val = DataLoader(mnist_val, batch_size=64)
mnist_test = DataLoader(mnist_test, batch_size=64)
pytorch_model = MNISTClassifier()
optimizer = torch.optim.Adam(pytorch_model.parameters(), lr=1e-3)

def cross_entropy_loss(logits, labels):
  return F.nll_loss(logits, labels)

num_epochs = 1
for epoch in range(num_epochs):
  for train_batch in mnist_train:
    x, y = train_batch
    logits = pytorch_model(x)
    loss = cross_entropy_loss(logits, y)
    print('train loss: ', loss.item())
    loss.backward()
    optimizer.step()
    optimizer.zero_grad()


  with torch.no_grad():
    val_loss = []
    for val_batch in mnist_val:
      x, y = val_batch
      logits = pytorch_model(x)
      val_loss.append(cross_entropy_loss(logits, y).item())
```

LSTM with pytorch Lightning



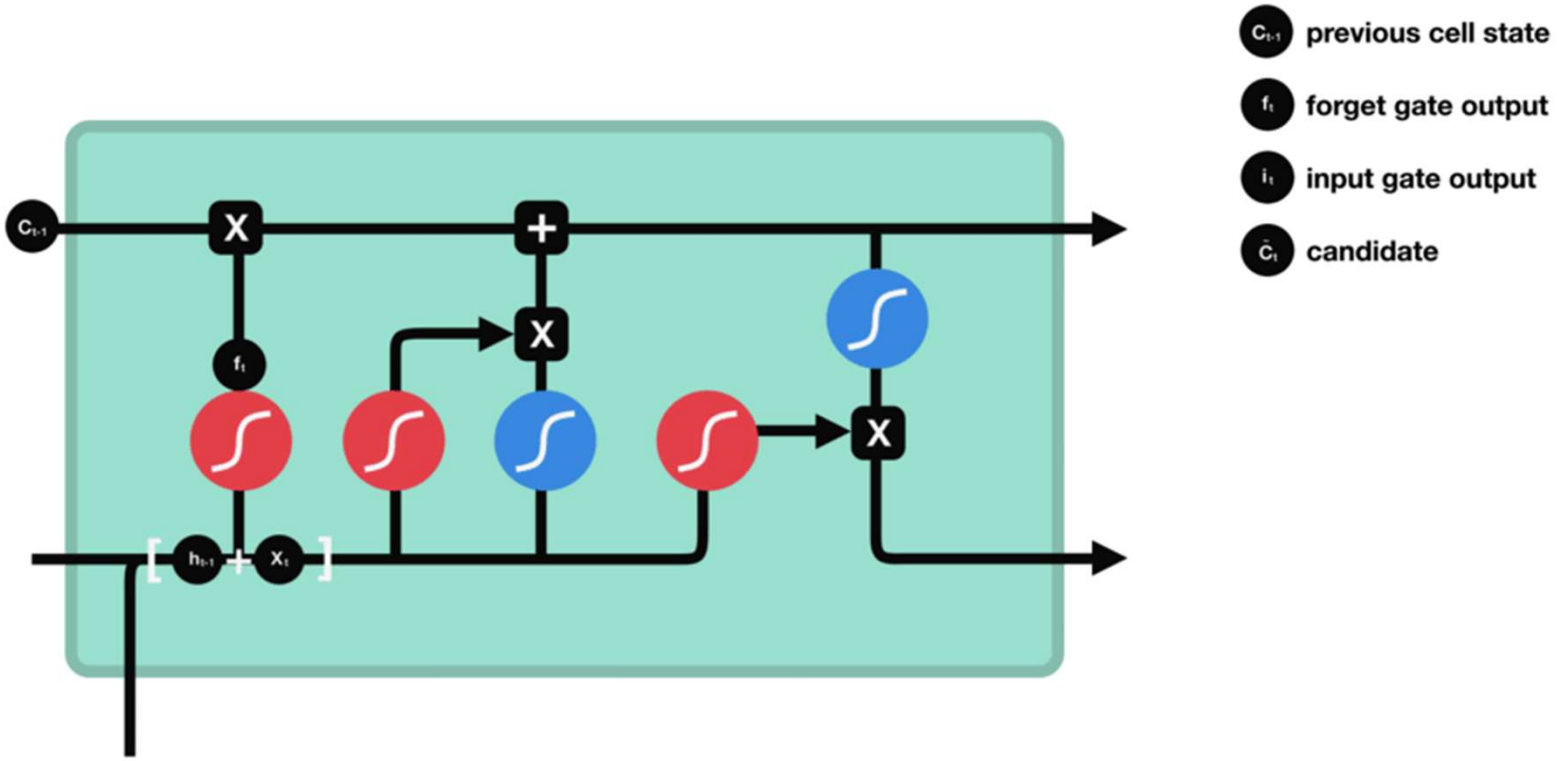
https://colah.github.io/

LSTM with pytorch Lightning



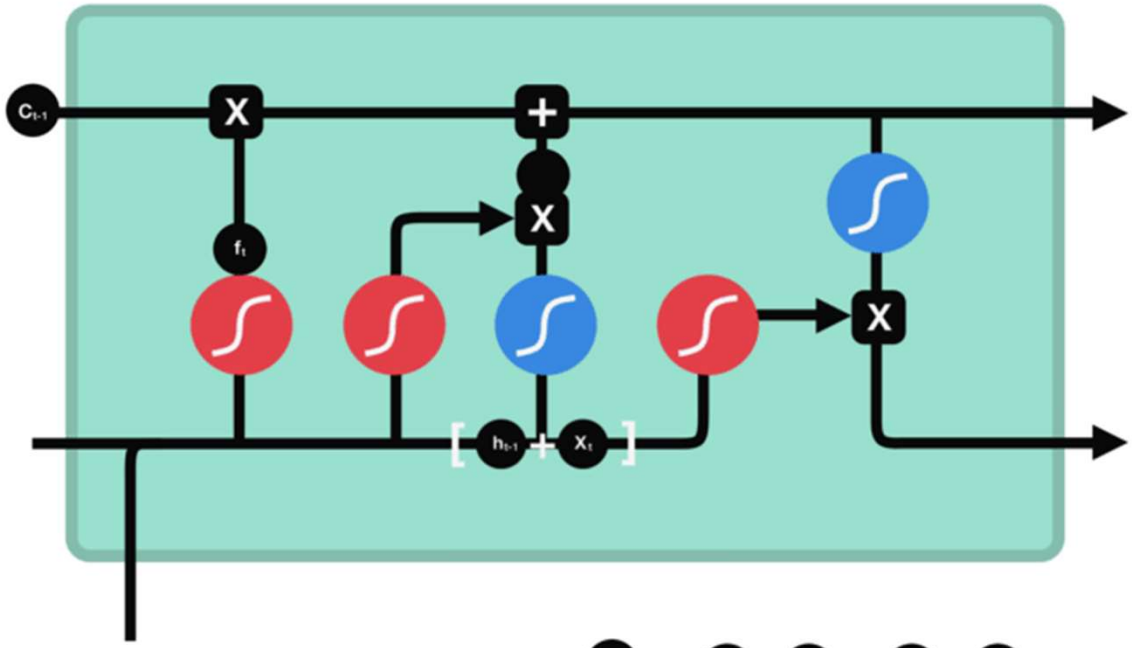
https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21

LSTM with pytorch Lightning



https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21

LSTM with pytorch Lightning



C_{t-1} previous cell state

f_t forget gate output

i_t input gate output

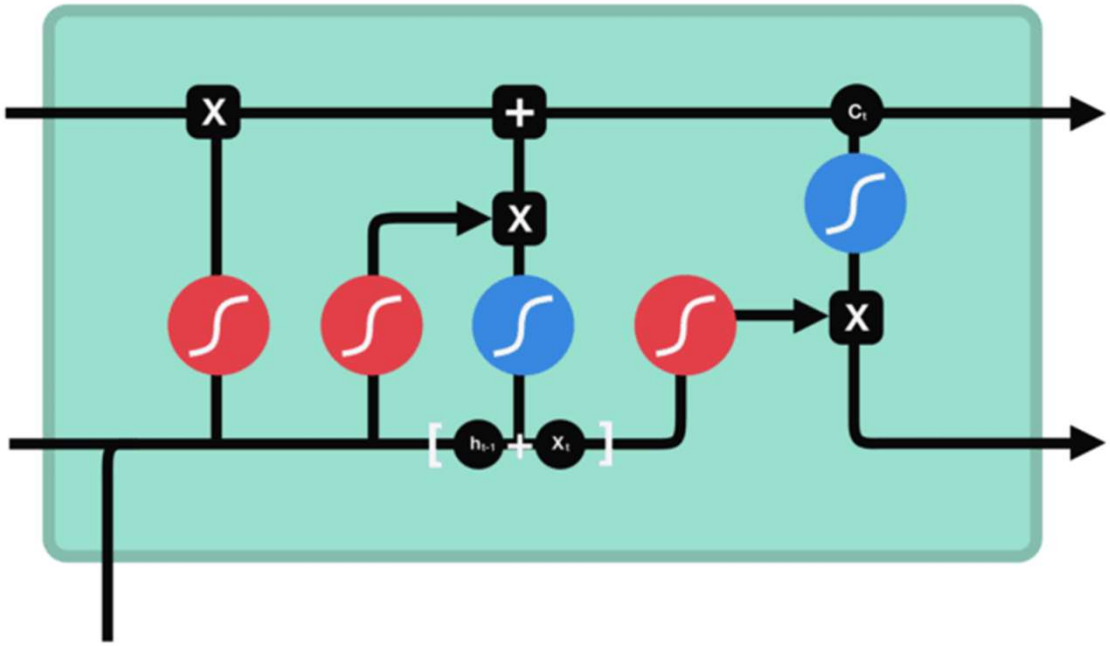
$\tilde{C}_t$ candidate

C_t new cell state

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21

LSTM with pytorch Lightning



C_{t-1} previous cell state

f_t forget gate output

i_t input gate output

$\tilde{C}_t$ candidate

C_t new cell state

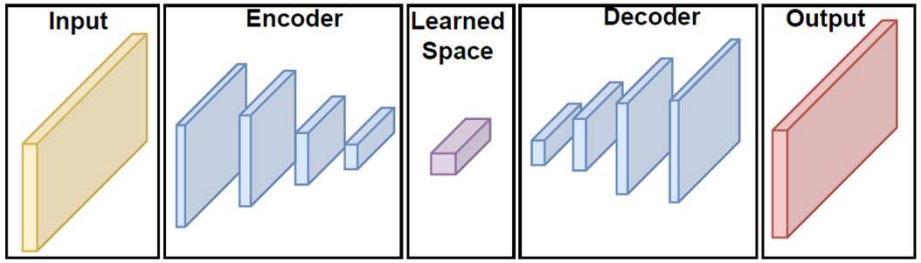
o_t output gate output

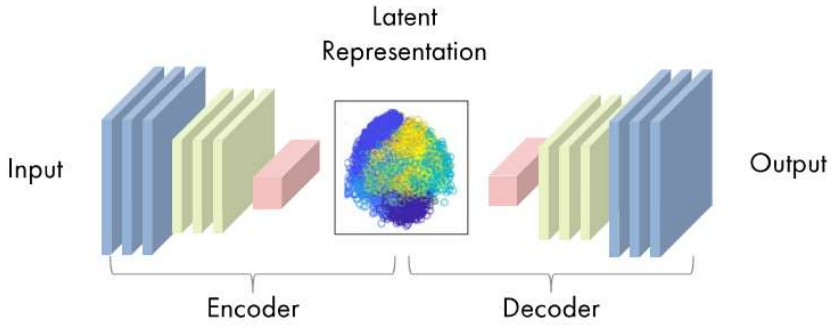
h_t hidden state

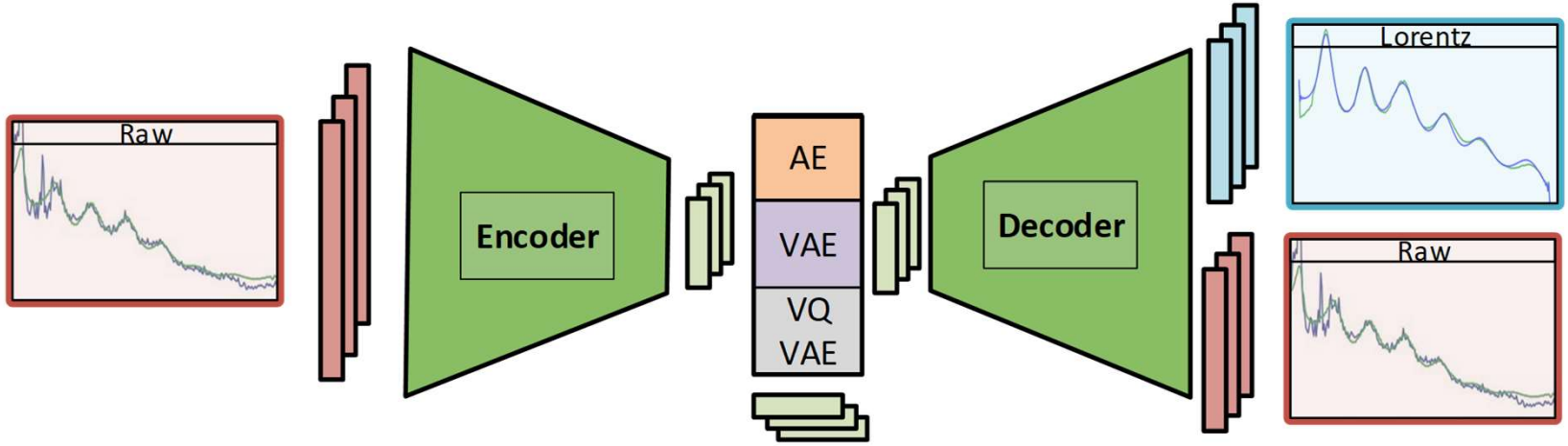
https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21

5 AutoEncoders with pytorch Lightning

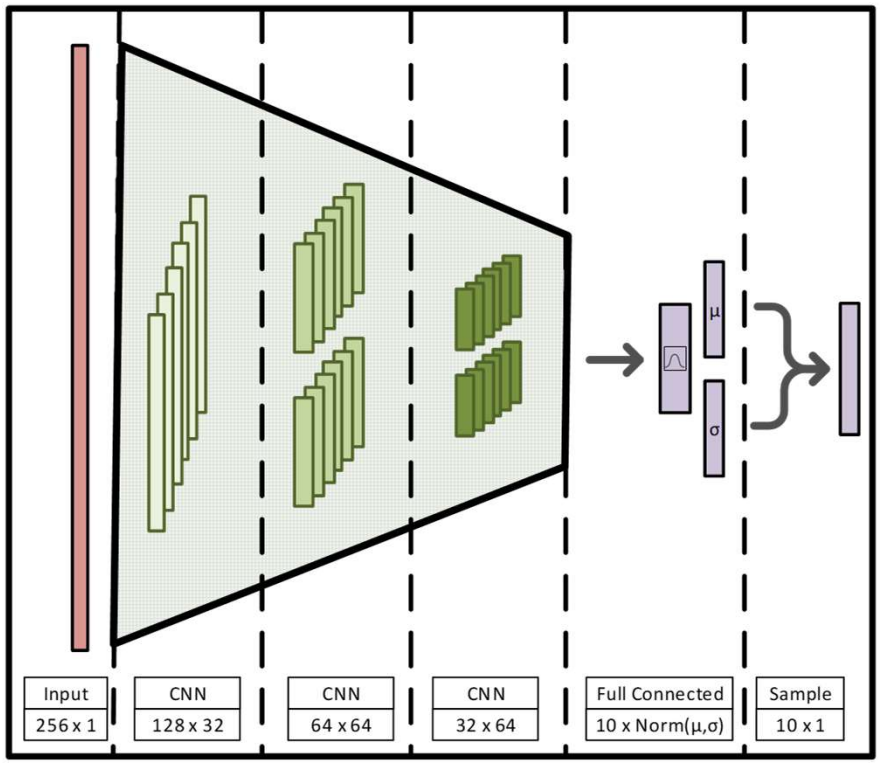
AutoEncoders with pytorch Lightning

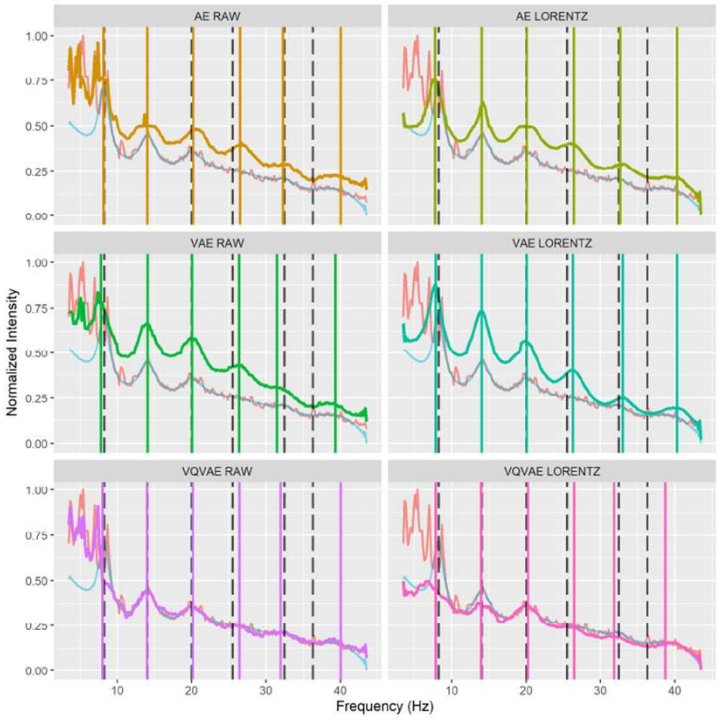






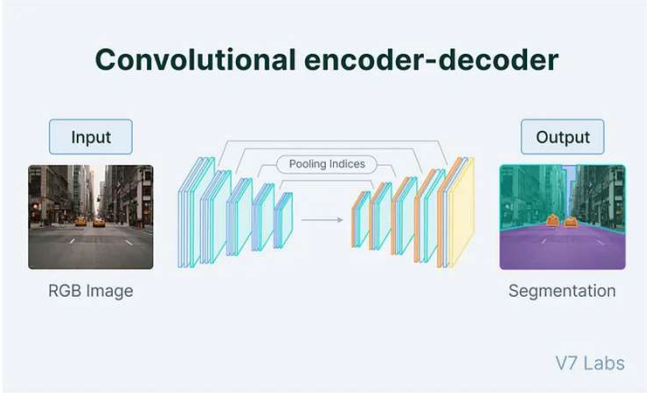
AutoEncoders with pytorch Lightning

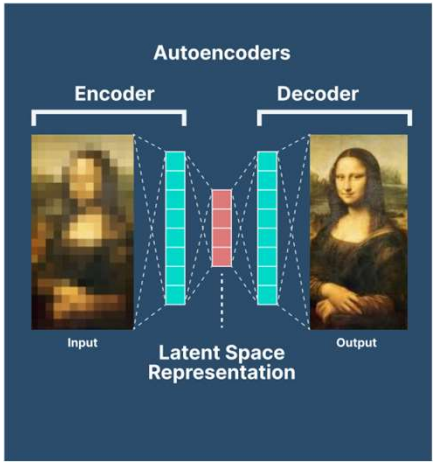


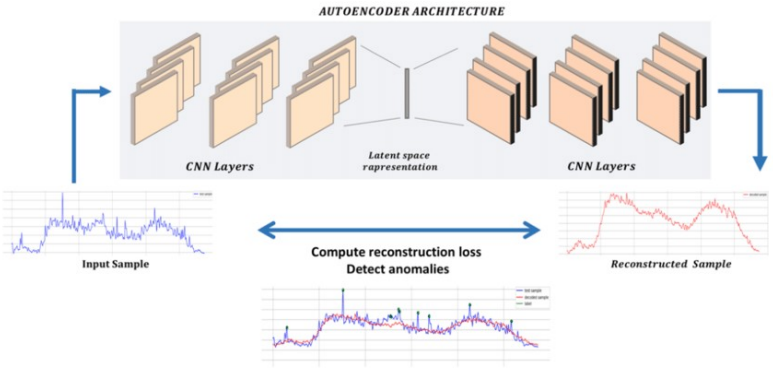


AutoEncoders with pytorch Lightning

- Time Series segmentation.
- Time Series Classification
- Denoising Signals
- Detect anomalies

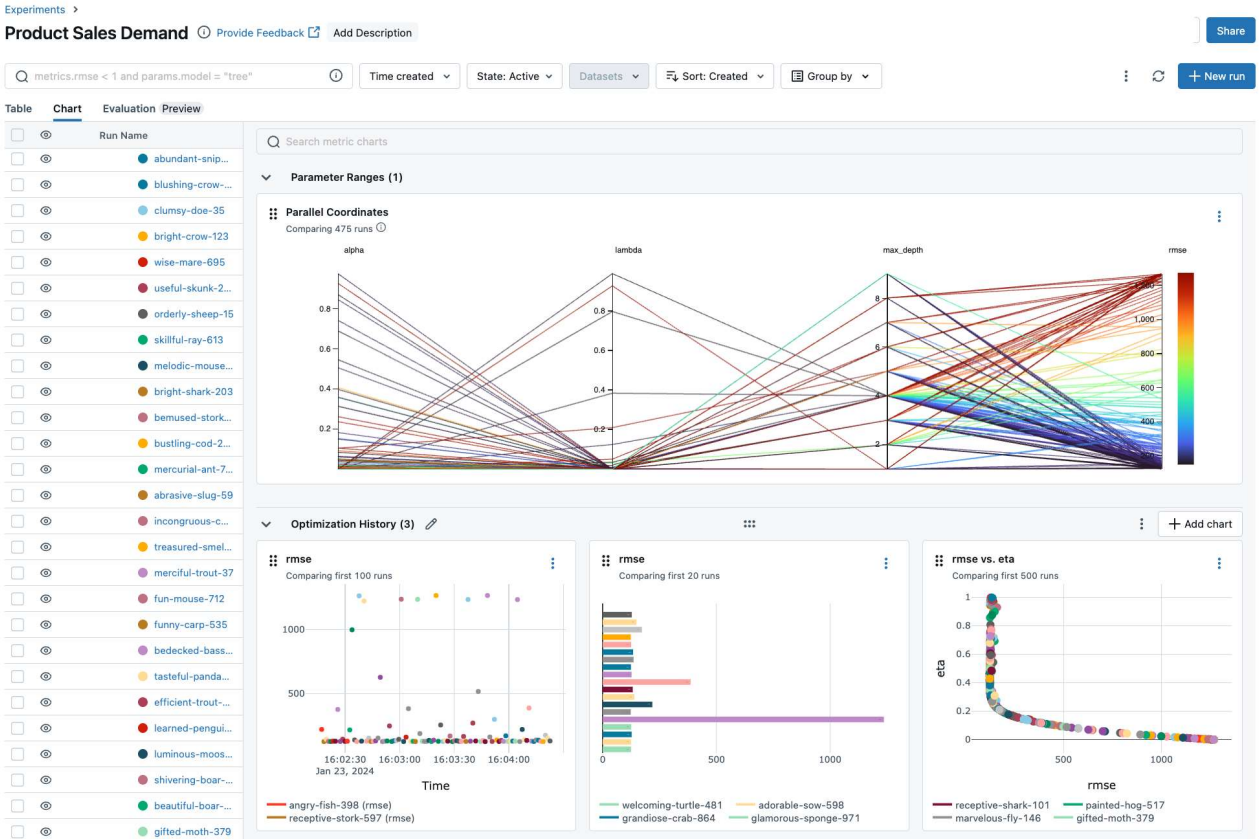




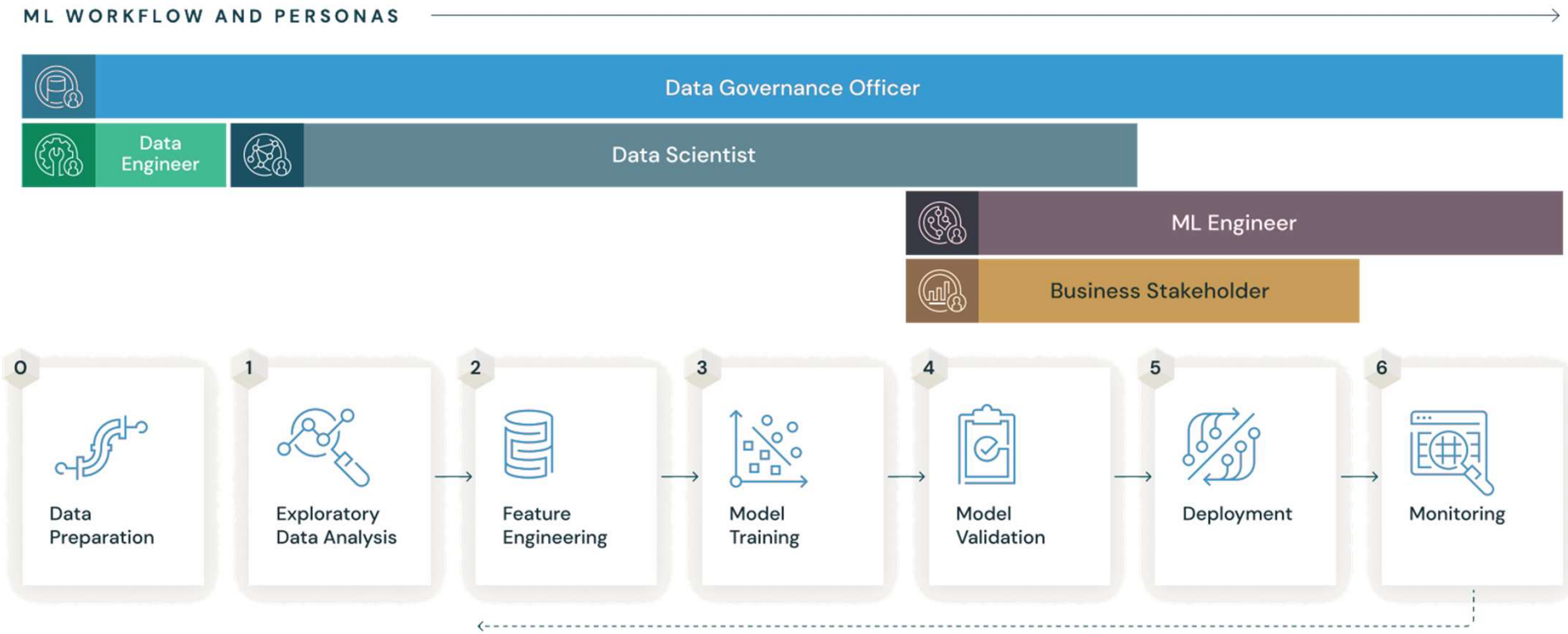


6 MLFlow & RayTune

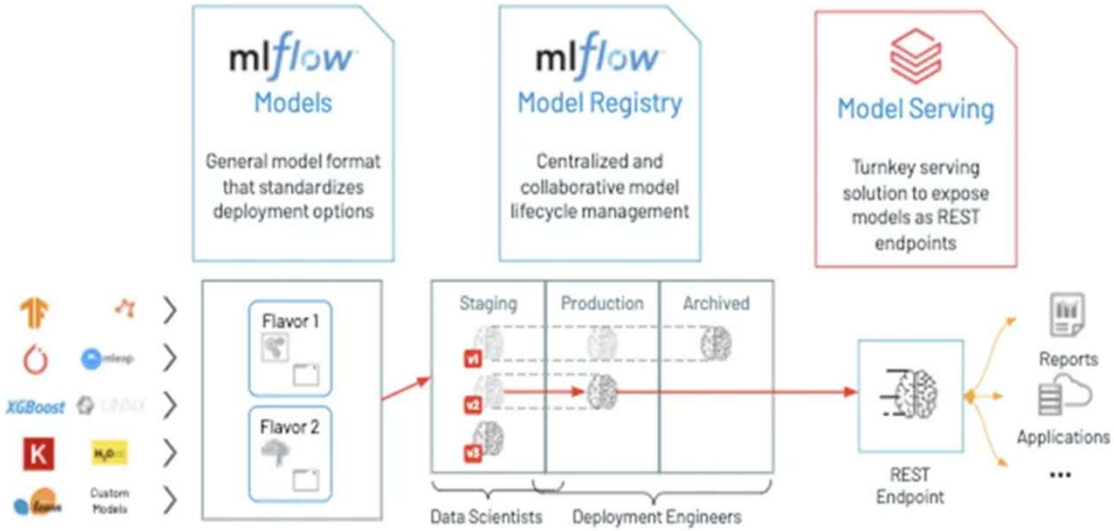
mlflow & RayTune

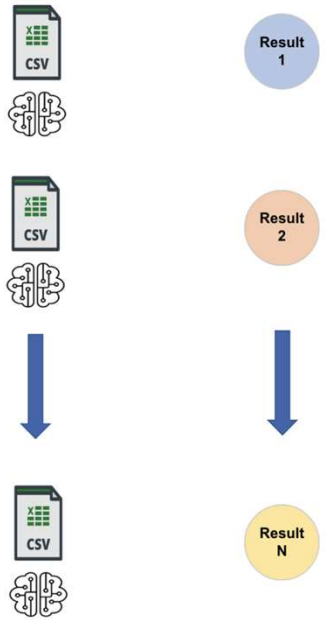


mlflow & RayTune

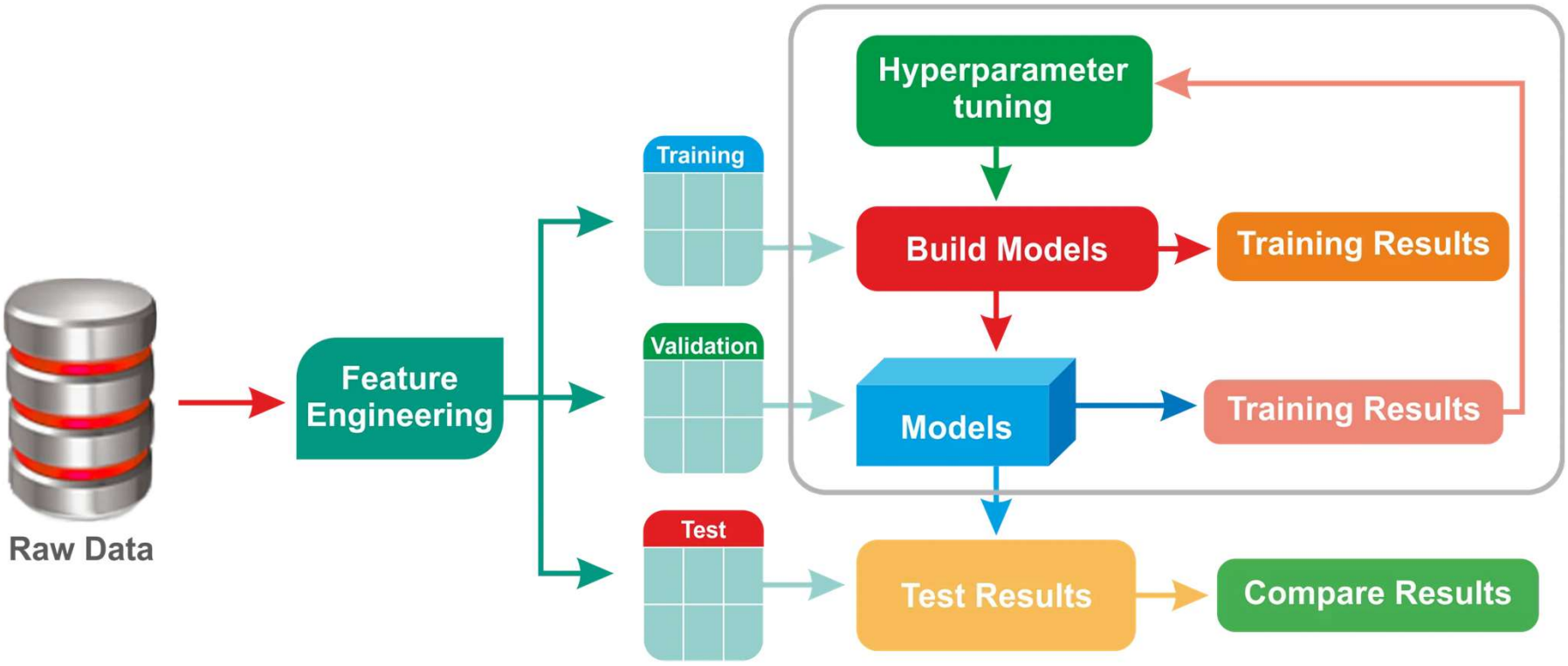


mlflow & RayTune

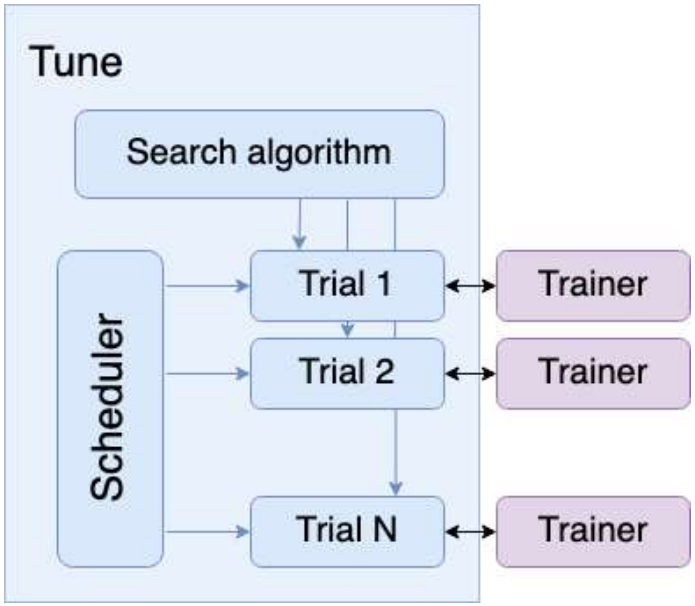


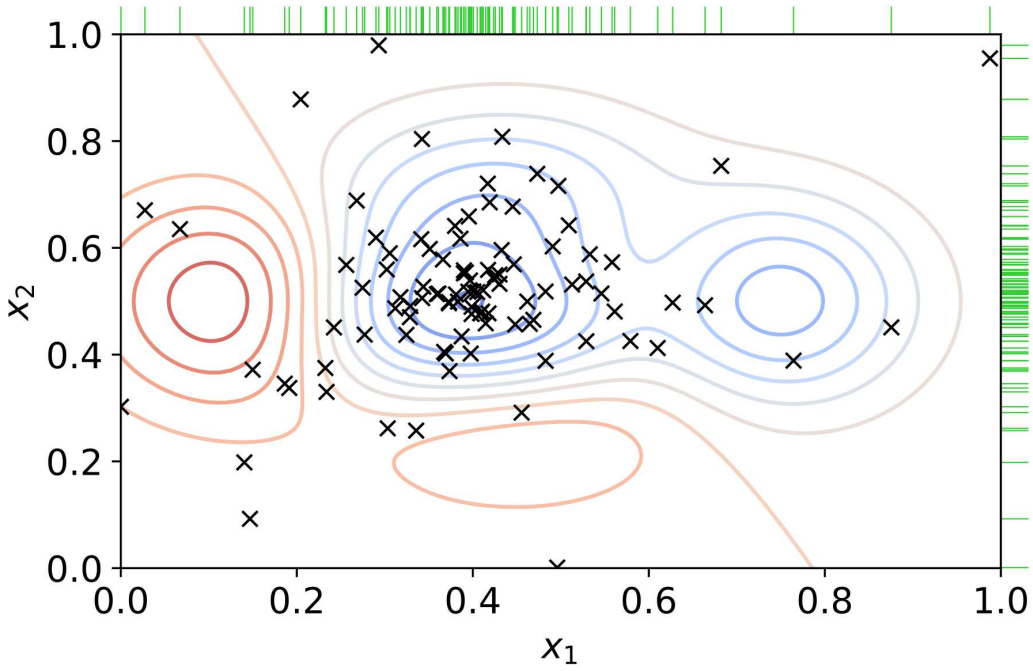


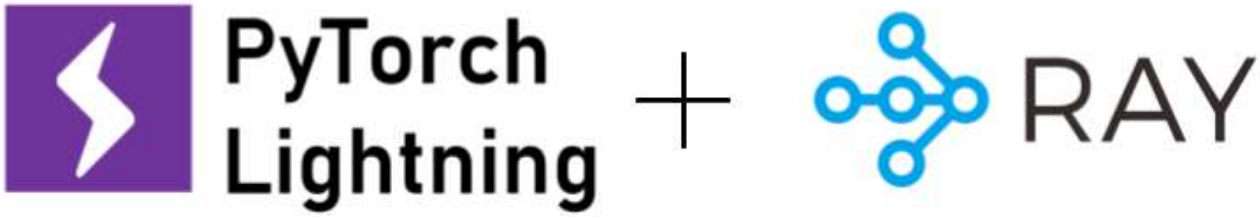
mlflow & RayTune



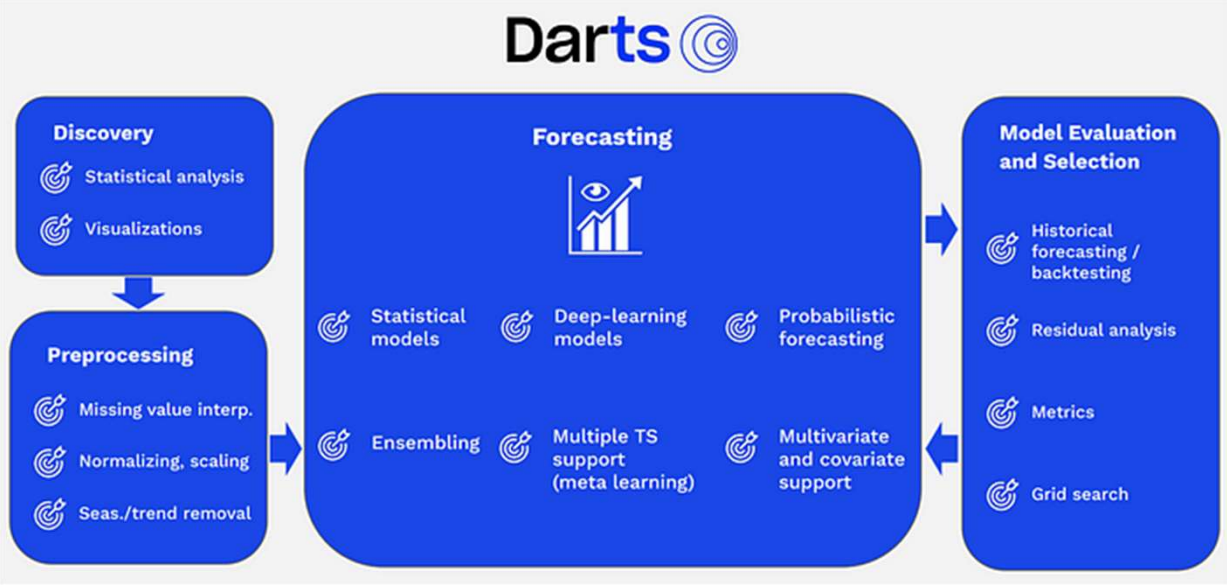
mlflow & RayTune

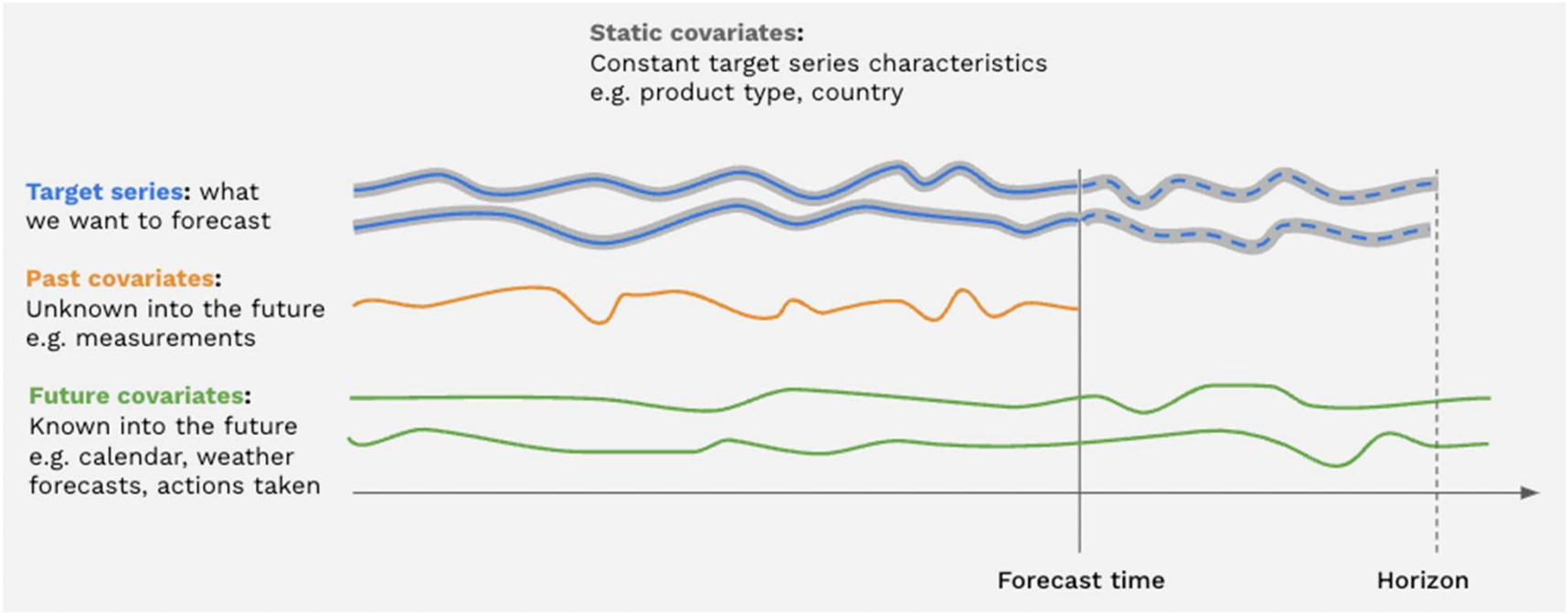




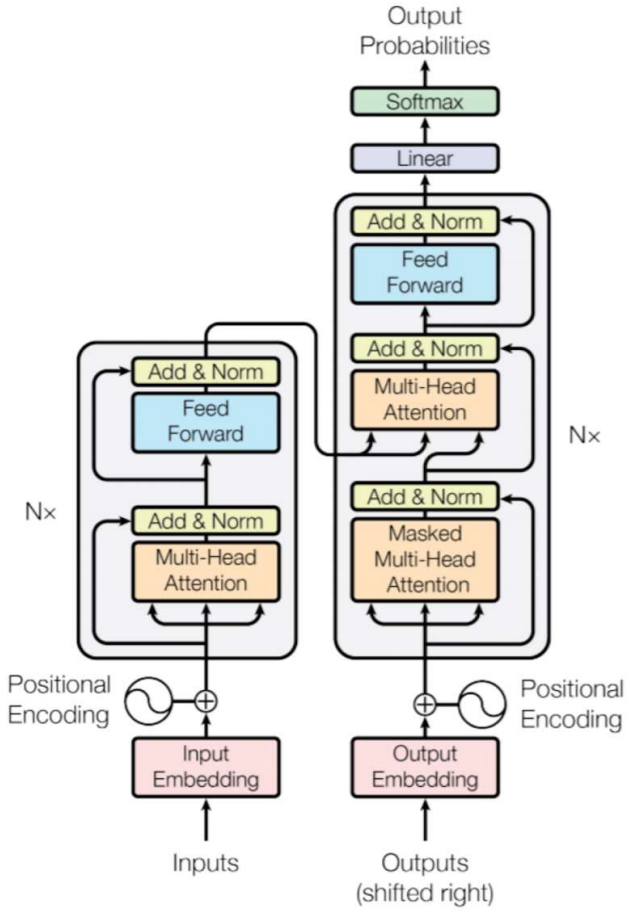

PyTorch
Lightning
+
RAY

7 Transformers with DARTS

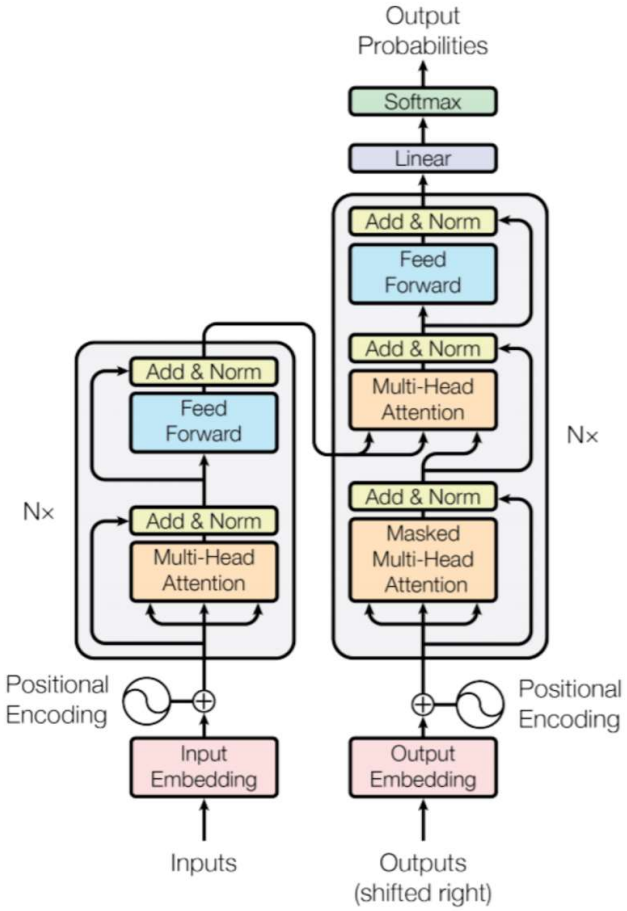

Darts
Discovery
Statistical analysis
Visualizations
Preprocessing
Missing value interp.
Normalizing, scaling
Seas./trend removal
Forecasting
Statistical models
Deep-learning models
Probabilistic forecasting
Ensembling
Multiple TS support (meta learning)
Multivariate and covariate support
Model Evaluation and Selection
Historical forecasting / backtesting
Residual analysis
Metrics
Grid search

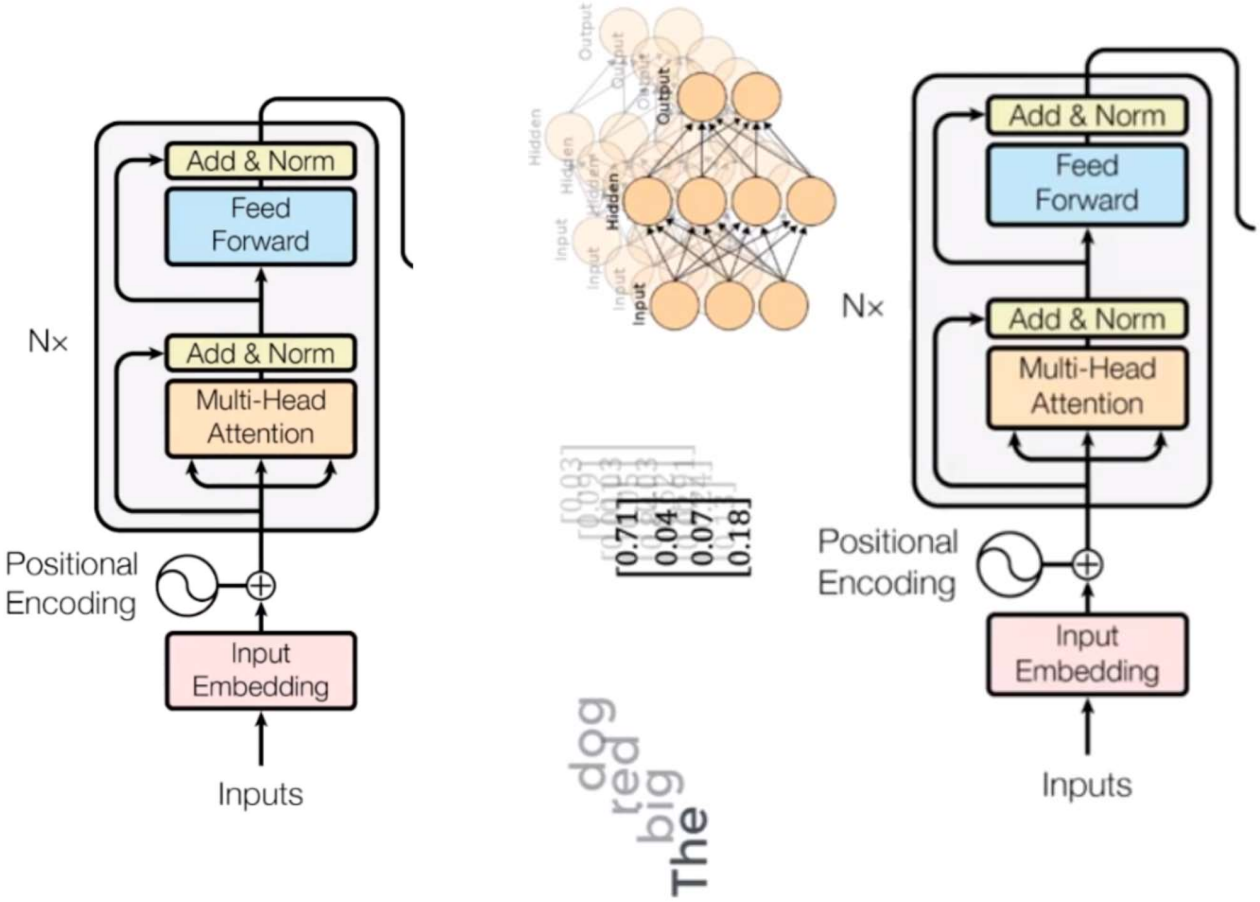

Static covariates:
Constant target series characteristics
e.g. product type, country
Target series: what we want to forecast
Past covariates:
Unknown into the future
e.g. measurements
Future covariates:
Known into the future
e.g. calendar, weather forecasts, actions taken
Forecast time
Horizon

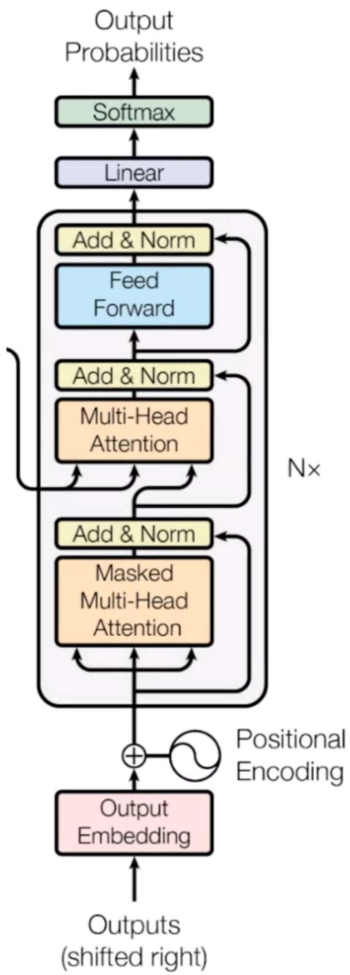
•**Attention Mechanism**: Unlike traditional RNNs (Recurrent Neural Networks) or LSTMs, transformers use **self-attention**. This allows them to capture **long-range dependencies** in time series data more effectively without the vanishing gradient problem.
•**Scalability**: Transformers process data in **parallel** instead of sequentially, which significantly increases the speed and efficiency when dealing with large time series datasets.
•**Versatility**: They can handle **irregularly sampled** data and incorporate **exogenous variables** (like covariates) easily, making them a flexible solution for time series problems.



- **Input Representation**: Time series data is first **embedded** (similar to how words are embedded in NLP tasks) and combined with **positional encodings** to maintain the order of the sequence.
- **Self-Attention**: The key component is the **multi-head attention mechanism**, where each time step attends to all others. This allows the model to learn both short-term and long-term dependencies in the data.
- **Decoder**: For forecasting tasks, the **masked multi-head attention** is used in the decoder to ensure that the predictions are made in an autoregressive fashion (i.e., only depending on past information).




Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Nx
Positional Encoding
Input Embedding
Inputs
Output
Hidden
Input
[0.71]
[0.04]
[0.07]
[0.18]
dog
red
big
The
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Nx
Positional Encoding
Input Embedding
Inputs


Output Probabilities
Softmax
Linear
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Add & Norm
Masked Multi-Head Attention
Nx
Positional Encoding
Output Embedding
Outputs (shifted right)