

See discussions, stats, and author profiles for this publication at: <https://www.researchgate.net/publication/324889271>


# An Introduction to Time Series Forecasting with Python

**Presentation** · April 2018  
DOI: 10.13140/RG.2.2.18053.86249

CITATIONS  
0

READS  
14,331

1 author:



**Andrii Gakhov**  
V. N. Karazin Kharkiv National University

17 PUBLICATIONS 3 CITATIONS

SEE PROFILE

Some of the authors of this publication are also working on these related projects:

- Project

Probabilistic Data Structures and Algorithms for Big Data Applications. [View project](#)
- Project

Ph.D. in Mathematical Modelling and Numerical Methods [View project](#)

# An Introduction to Time Series Forecasting with Python

Andrii Gakhov, ferret go GmbH  
[www.gakhov.com](http://www.gakhov.com)

PyCon UA, Kharkiv, April 28–29, 2018

What is a Time Series?

A sequential set of data points  
measured over time.

$$x(t), t = 0, 1, 2, 3, \dots$$

$t$  – a variable that represents the elapsed time.

# Examples

- **hourly** number of page views for a website
- **daily** air temperature in a city
- **monthly** average number of reported bugs
- **quarterly** income for a company
- **annual** population data in a country

A Time Series may contain information  
about general tendency in data,  
seasonal effects, occasional events, and so on.

# Time Series Decomposition

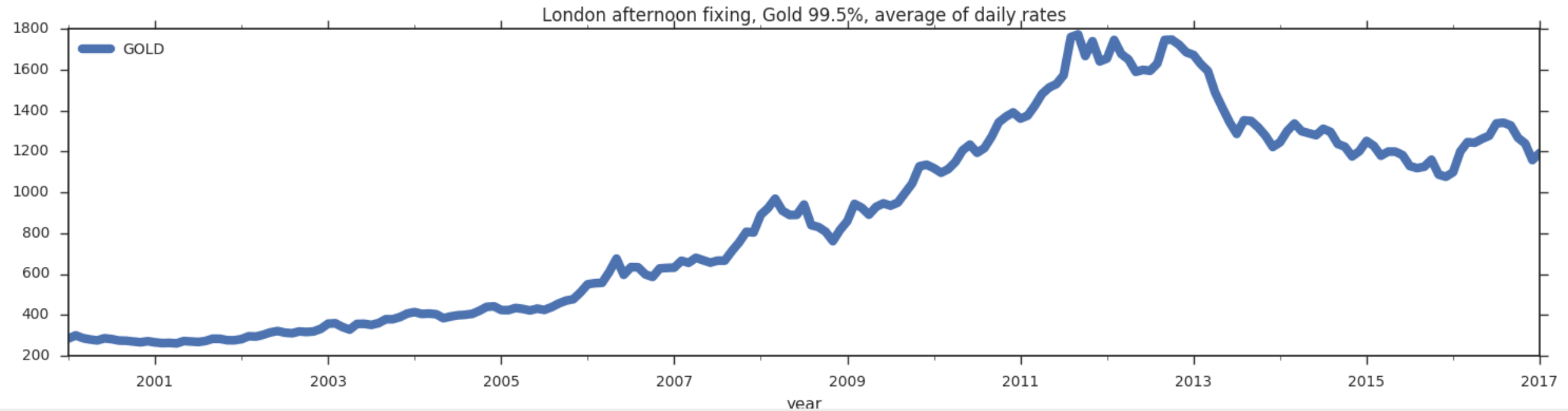
# Trend

The general (long-term, non-periodic)  
tendency of a time series

⚠ Often modeled using the logistic growth model, or can be extracted from data using Moving Average technique



# Example



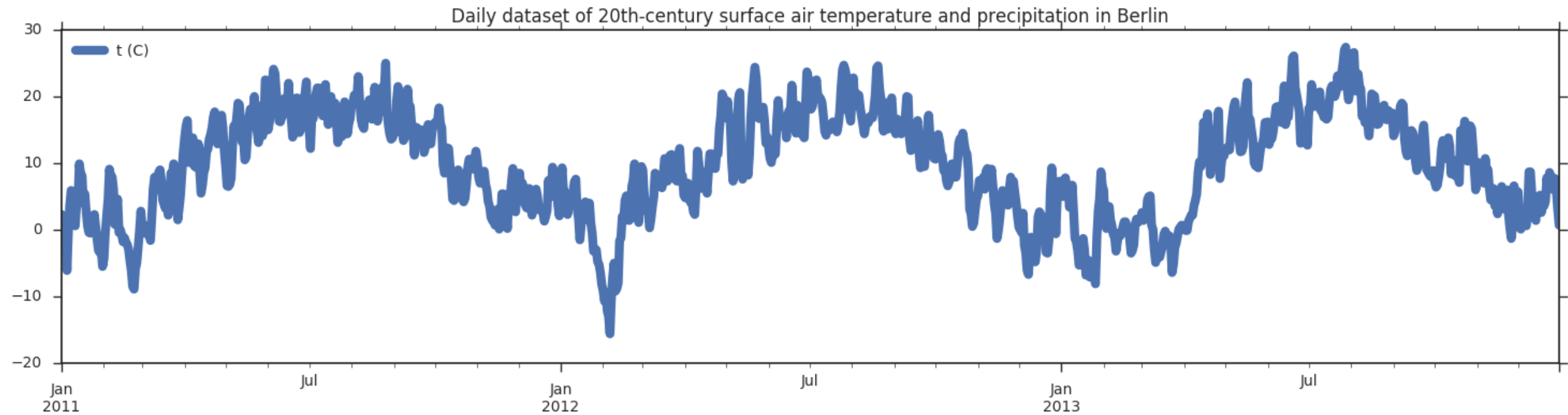
The WoldBank Dataset, London afternoon fixing, Gold 99.5%, average of daily rates

# Seasonal

The periodic fluctuations  
caused by regular influences

⚠ Can be modeled using Fourier series

# Example

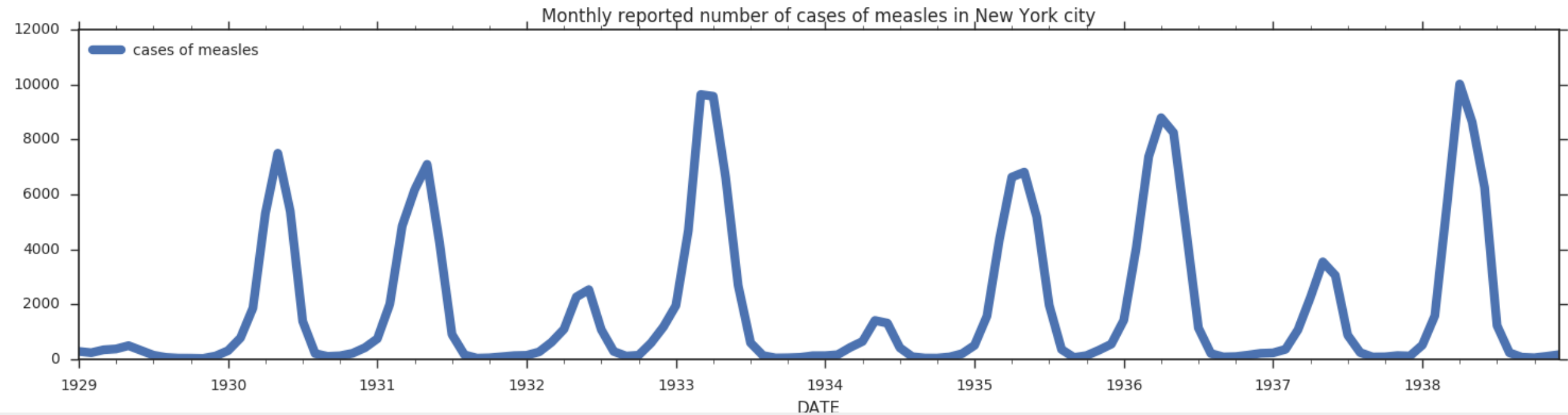


European Climate Assessment & Dataset, Air temperature in Berlin

# Cyclical

The medium-term fluctuations caused  
by cyclically occurred non-periodic influences

# Example

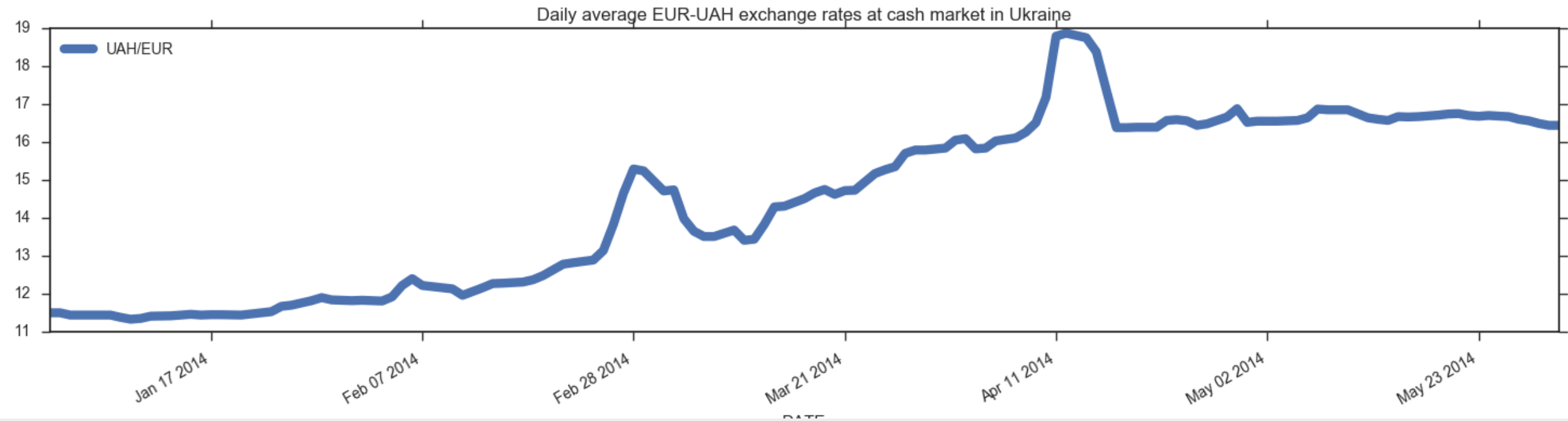


London W., Yorke J.A., The monthly reported number of cases of measles, New York city

# Irregular

Random variations caused  
by unpredictable influences

# Example



valuta.today, Daily average EUR-UAH exchange rates in the cash market, Ukraine

# Time Series Decomposition Model

There are 2 main models that generally used  
to model the effects of these components

⚠ The choice of the decomposition model is called **test for log-level specification**.



# Additive model

Assumes that the four components are *independent*

$$Y(t) = T(t) + S(t) + C(t) + I(t)$$

# When to choose the Additive model

- The behaviors of the components are independent,  
i.e. an increase in the trend-cycle will not cause an increase in the magnitude of seasonal
- The difference of the trend and the raw data  
is roughly constant in similar periods of time  
irrespectively of the tendency of the trend
- The pattern of seasonal variation  
is roughly stable over the year  
i.e. the seasonal movements are the approximately same from year to year

<https://ec.europa.eu/eurostat/sa-elearning/additive-decomposition-0>

# Multiplicative model

Assumes that the components are *dependent*

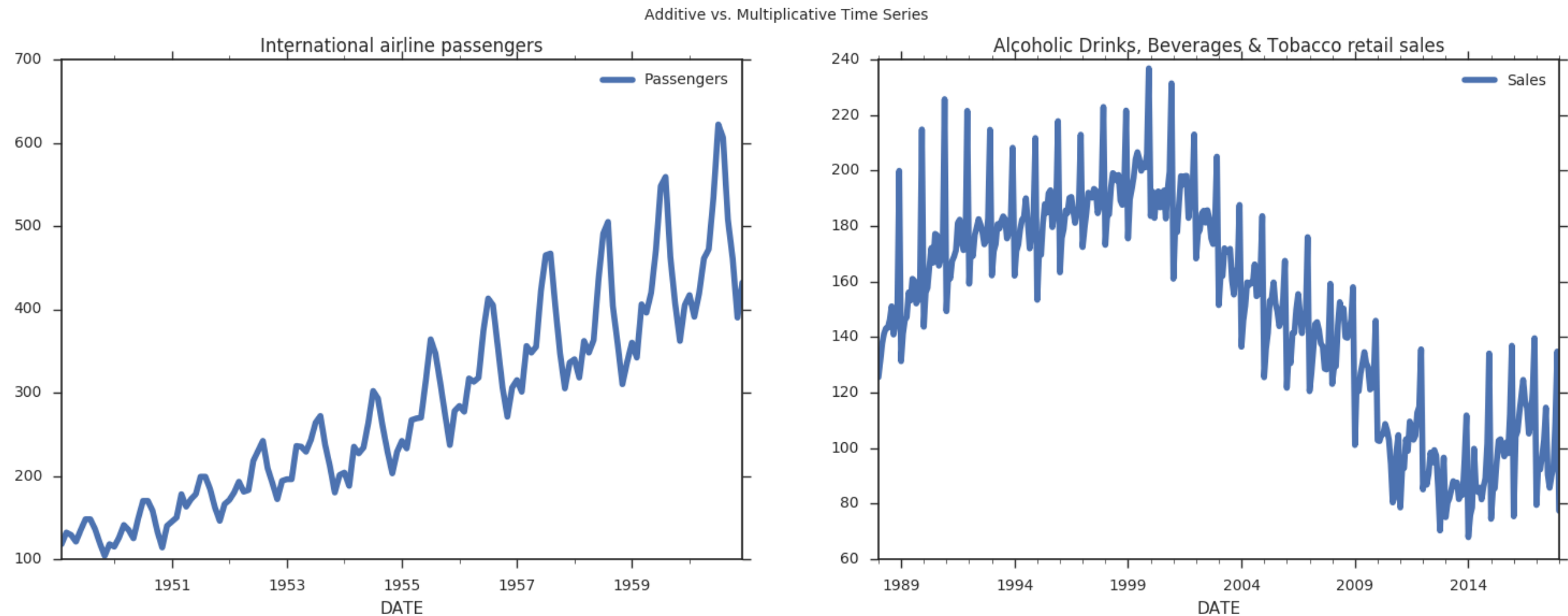
$$Y(t) = T(t) \cdot S(t) \cdot C(t) \cdot I(t)$$

By taking *logs* it is possible to present the multiplicative model in an additive form. This is called *log-additive* model.

# When to choose the Multiplicative model

- Components on the graph jump in magnitude over time
- The seasonal and irregular fluctuations change in a specific manner after the trend  
i.e. the amplitude of the seasonality increase (decrease) with an increasing (decreasing) trend

# Additive vs. Multiplicative



**Additive:** Alcoholic Drinks, Beverages & Tobacco retail sales, UK; **Multiplicative:** International airline passengers

# Demo

---

Meet the dataset

Total number of visits to the UK by overseas residents

Source: Office for National Statistics

<https://github.com/gakhov/pycon-ua-2018/blob/master/look-into-the-data.ipynb>

# The Roadmap

Data – Decomposition Model –  
– Forecasting Model – Training – Prediction

# How To Measure The Forecast Performance



# The Mean Forecast Error (MFE)

$$\frac{1}{n} \sum_{t=1}^n (x_t - f_t)$$

A good forecast has the MFE close to zero

---

- Provides the direction of the error
- $MFE = 0$  doesn't guarantee that the forecast contains no errors (positives can cancel negative errors)
- Sensitive to observations scale and data transformations
- Doesn't penalize extreme errors

# The Mean Squared Error (MSE)

$$\frac{1}{n} \sum_{t=1}^n (x_t - f_t)^2$$

```
from sklearn.metrics import mean_squared_error
```

A good forecast has the MSE close to zero

---

- Shows the overall idea of the error
- Emphasizes the fact that the total forecast error is much affected by large individual errors
- Sensitive to observations scale and data transformations
- Penalizes extreme errors

# The Normalized Mean Squared Error (NMSE)

$$\frac{1}{\sigma^2 n} \sum_{t=1}^n (x_t - f_t)^2$$

The smaller the NMSE, the better the forecast

---

- It's balanced error measure
- Very effective in judging forecast accuracy of the model

# The Coefficient of Determination ( $R^2$ )

$$R^2 = 1 - \frac{\sum (x_t - f_t)^2}{\sum (x_t - \bar{x})^2}$$

```
from sklearn.metrics import r2_score
```

A good forecast has the  $R^2$  close to one

---

- Estimates the variability of the forecast around its mean
- $R^2 \leq 1$
- Doesn't indicate whether the model is adequate

## Theil's U

$$U = \frac{\sqrt{\frac{1}{n} \sum_{t=1}^n (x_t - f_t)^2}}{\sqrt{\frac{1}{n} \sum f_t^2} \sqrt{\frac{1}{n} \sum x_t^2}}$$

A good forecast has the U close to zero

---

- A normalized measure of total forecast error
- $0 \leq U \leq 1$
- $U = 0$  means the perfect fit

# Models

There are many different approaches,  
but here we focus on the top 3 families.

# Stochastic Models

## with Python

# ARMA Model

Assumes that data are  
a realization of a stationary process

Augmented Dickey-Fuller (ADF) unit root test,  
Kwiatkowski-Phillips-Schmidt-Shin (KPSS) test



## AR – autoregression process (long memory)

Have a relatively long memory, because the current observation is correlated with all previous ones, although with decreasing coefficients

## MA – moving average process (short memory)

Have a short-term memory, since it's just a function of a finite, and generally small, number of its past observations.

---

ARMA is to combine these properties and represent model in a simplified form whose first  $q$  coefficients can be any, whereas the following ones decay according to simple rules.

$$ARMA(p, q)$$

- $p$  – the lag order (AR)  
the number of lag observations to include
- $q$  – the order of moving average (MA)  
the size of the moving average window

# ARIMA

## Autoregressive Integrated Moving Average

A generalization of an ARMA for  
Integrated (Difference Stationary) Time Series

Non-stationary time series whose differences of some order are stationary.

---

- Non-stationary time series are differenced until stationarity is achieved (Box–Jenkins method)
- Doesn't support seasonal component modeling

$$ARIMA(p, d, q)$$

- $p$  – the lag order (AR)  
the number of lag observations to include
- $d$  – the degree of differencing (I)  
the number of times that the raw observations were differenced
- $q$  – the order of moving average (MA)  
the size of the moving average window

The data has to be prepared by a degree of differencing to make time series stationary, i.e. by removing trend and seasonal structures that negatively affect the regression model.

ARMA corresponds to  $d=0$ .

# SARIMA

## Seasonal Autoregressive Integrated Moving Average

A modification of ARIMA to support a seasonal time series.

$$ARIMA(p, d, q) \times (P, D, Q)_s$$

- $(p, d, q)$  are the non-seasonal parameters
- $(P, D, Q)$  the seasonal component of the time series.
- $s$  is the seasonal periodicity  
(4 for quarterly periods, 12 for yearly periods, etc.)

- Check whether there is any evidence of a trend or seasonal effects
- Stationarization by taking differences
- Seasonal component research

# Python Tools

---

## Statsmodels

A high-level module to work with statistical models,  
conducting statistical tests and data exploration.

<https://www.statsmodels.org/>

```
from statsmodels.tsa.statespace import sarimax

model = sarimax.SARIMAX(
    X_train,
    trend='n',
    order=(1,1,1),
    seasonal_order=(1,1,0,12),
    enforce_stationarity=True,
    enforce_invertibility=True)
results = model.fit()

steps = X_train.shape[0]

forecast = results.get_forecast(steps=steps)
forecast_ci = forecast.conf_int()

yhat_test = forecast.predicted_mean.values
```



# Demo

---

Build a Seasonal AutoRegressive Integrated Moving Average model  
to predict the number of visits to the UK by overseas residents

<https://github.com/gakhov/pycon-ua-2018/blob/master/stochastic-models.ipynb>

# Python Tools

---

## Prophet

A tool for producing forecasts for time series data that has multiple seasonality with linear or non-linear growth.

<https://github.com/facebook/prophet>

# Prophet Forecasting Model

Based on ARIMA and utilizes a Bayesian-based curve fitting method

$$X(t) = T(t) + S(t) + H(t) + \varepsilon_t$$

$T(t)$  – trend,  $S(t)$  – seasonal effects,  $H(t)$  – holidays (potentially irregular),  $\varepsilon_t$  – unpredicted effects.

The seasonality is modeled by the Fourier Series.

```
from fbprophet import Prophet
```

```
m = Prophet()  
m.fit(df)
```

```
forecast = m.predict(df)  
m.plot(forecast)
```

```
m.plot_components(forecast);
```

# Demo

---

Build a Prophet Forecasting Model  
to predict the number of visits to the UK by overseas residents

<https://github.com/gakhov/pycon-ua-2018/blob/master/prophet.ipynb>

# Artificial Neural Networks

## with Python

# ANN can be seen as a computational graph

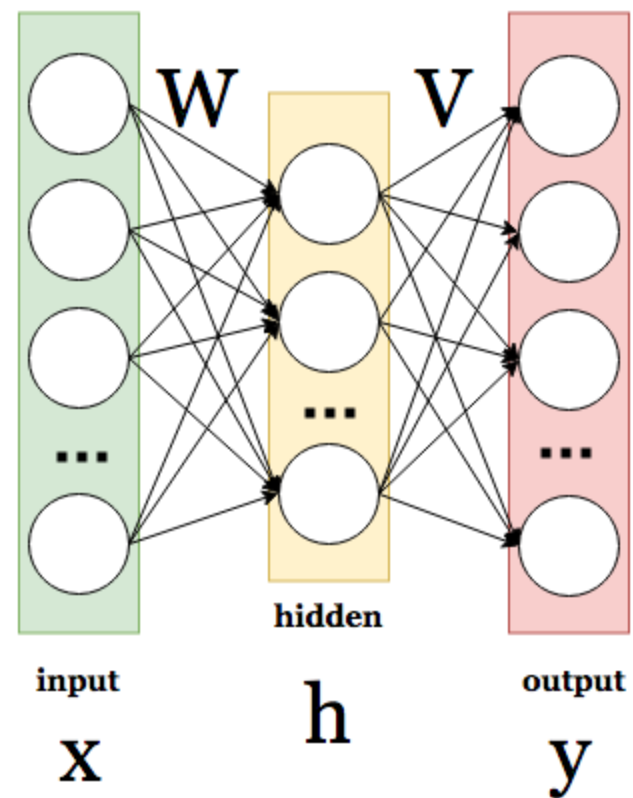
- **nodes** are computing units
- **directed edges** transmit numeric information

Each computing unit (neuron) is capable of evaluating a single primitive function (activation function) of its input.

At practice, units are organized in layers.

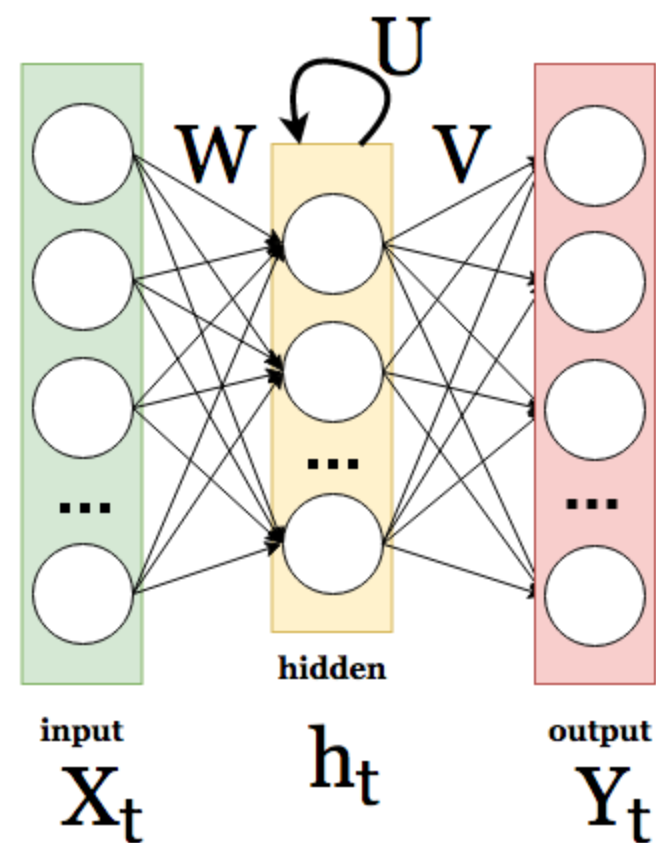
# Popular ANNs types

FeedForward Neural Network



$$h = g(Wx)$$
$$y = f(Vh)$$

Recurrent Neural Network



$$h_t = g(Wx_t + Uh_{t-1})$$
$$y_t = f(Vh_t)$$



# Recurrent Neural Network

Similar to Feedforward networks, but allows a recurrent hidden state with activation depended on the previous cycles.

---

## Long Short-Term Memory (LSTM) architecture

Unlike the traditional recurrent unit, which overwrites its content each timestep, the LSTM unit can decide whether to keep the existing memory via introduced additional convolution steps (gates)

Read More: <https://www.slideshare.net/gakhov/recurrent-neural-networks-part-1-theory>

The main problem while working with ANNs  
is the choice of the network's parameters,  
such as the number of layers and number of neurons.

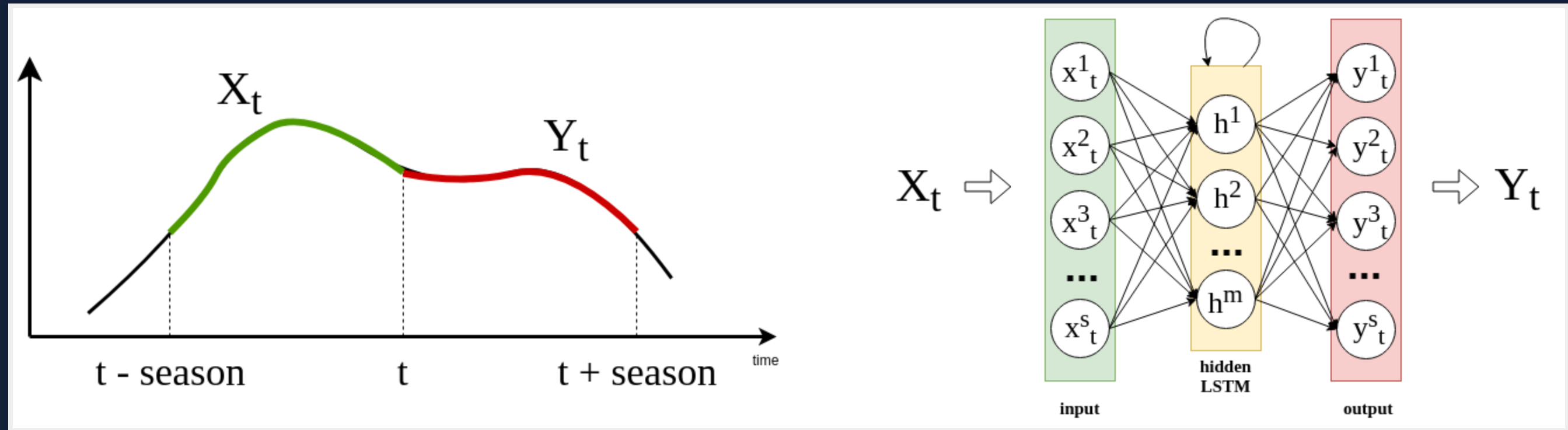
# SANN

## Seasonal Artificial Neural Networks

- only 1 hidden layer
- **number of input and output neurons**  
can be taken as number of observations in a season
- **number of hidden nodes**  
should be determined experimentally (a few)

\* as recommended by C. Hamzaçebi.

# How to build a model



We want to teach SANN to learn next season value from the previous season values

Using sliding-window technique transform time series to a number of s-dimensional vectors

# Python Tools

---

## Keras

A high-level neural networks API,  
capable of running on top of TensorFlow, CNTK, or Theano.

<https://keras.io/>

```
from keras.layers import InputLayer, Dense, LSTM
from keras.models import Sequential
from keras.optimizers import SGD

model = Sequential()
model.add(InputLayer(input_shape=(1, seasons), name="input"))
model.add(
    LSTM(4, name="hidden", activation='sigmoid',
        use_bias = True, bias_initializer='ones')
)
model.add(
    Dense(seasons, name="output", activation='linear',
        use_bias = True, bias_initializer='ones')
)

model.compile(loss='mean_squared_error',
              optimizer=SGD(lr=0.05, decay=1e-6, momentum=0.9),
              metrics=["mae", "mse"])
history = model.fit(X_train, y_train, epochs=100, batch_size=1,
                    validation_data=(X_val, y_val))

yhat_test = model.predict(X_test[:,seasons])
```

# Demo

---

Build a Seasonal Artificial Neural Network with LSTM layer  
to predict the number of visits to the UK by overseas residents

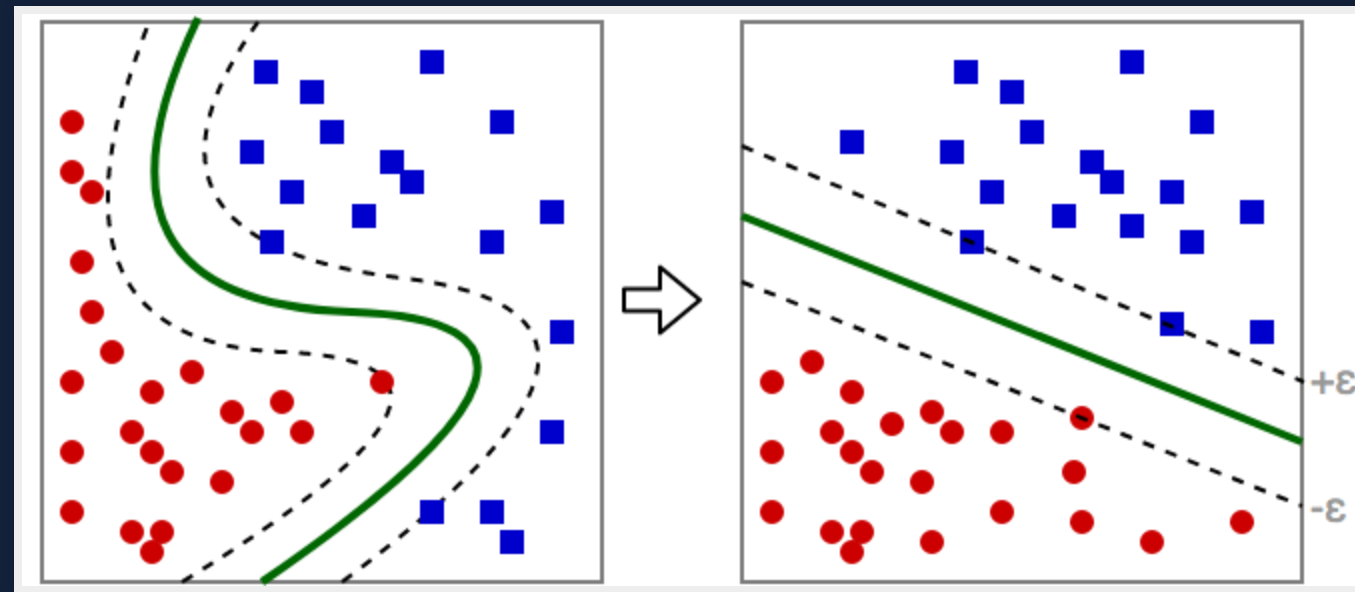
<https://github.com/gakhov/pycon-ua-2018/blob/master/artificial-neural-networks.ipynb>

# Support Vector Machines

## with Python



A technique that maps data  
into a high-dimensional feature space  
where it can be linearly separable in the best way.



Support vectors are "critical" observations that fully specify the decision function

# Support Vector Machine Regression

Performs a linear regression  
in the high-dimension feature space  
with a special  $\epsilon$ -insensitive loss function.

# Main SVM meta-parameters

---

## $\epsilon$ -tube for no penalty

Controls the width of the  $\epsilon$ -insensitive zone, used to fit the training data.

Bigger  $\epsilon$ -values result in more 'flat' estimates (fewer support vectors).

## Penalty parameter $C$

Controls the amount up to which deviations larger than  $\epsilon$  is tolerated.

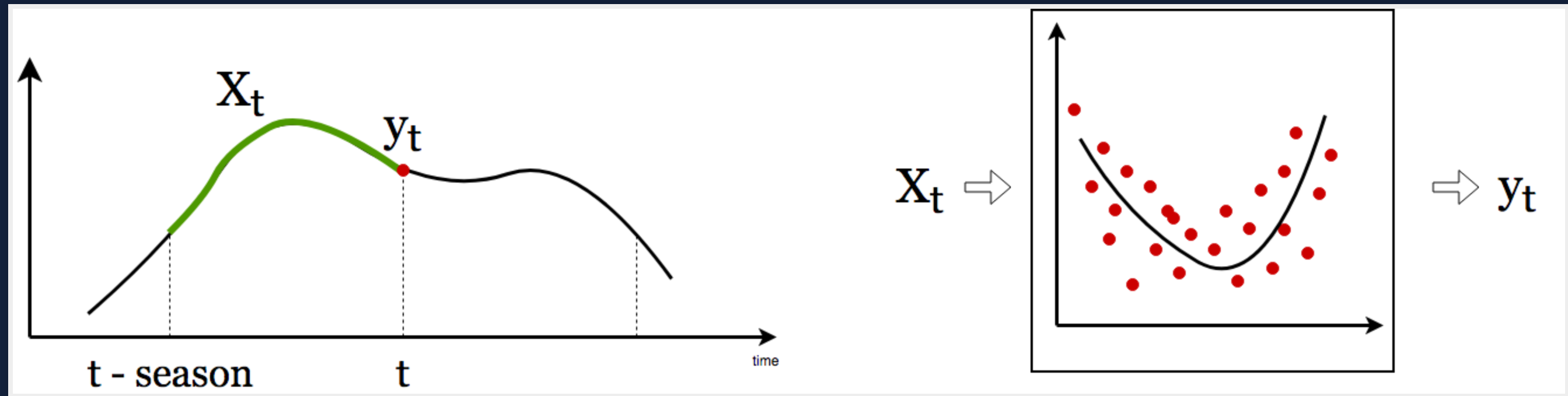
Prevents overfitting and in a trade-off with the model complexity.

## Kernel function type

Induces the high-dimensional feature space.

Most popular are linear, polynomial and Gaussian Radial Basis Function (RBF) kernels.

# How to build a model



We want to train SVM Regressor to learn value at time  $t$  from the previous season values

Using sliding-window technique transform time series to a number of  $s$ -dimensional vectors

# Python Tools

---

## Scikit-learn

A simple and efficient tool for machine learning,  
data mining and data analysis.

<http://scikit-learn.org/>

```
from sklearn import svm
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import ShuffleSplit

C_range = np.logspace(-2, 10, 5)
gamma_range = np.logspace(-9, 3, 5)
param_grid = {
    "gamma": gamma_range,
    "C": C_range,
    "epsilon": 0.1
}

cv = ShuffleSplit(n_splits=3, test_size=0.25, random_state=73)

grid = GridSearchCV(
    svm.SVR(kernel="rbf", max_iter=100),
    param_grid=param_grid,
    cv=cv)

grid.fit(X_train, y_train)
model = grid.best_estimator_

yhat_test = model.predict(X_test)
```

# Demo

---

Build a Seasonal Support Vector Regressor  
to predict the number of visits to the UK by overseas residents

<https://github.com/gakhov/pycon-ua-2018/blob/master/support-vector-machines.ipynb>

# Final Notes

- Always visualize the data
- Think which decomposition model better describes you data
- Select a proper forecasting model
- Divide the raw data into 2 parts (training / test sets).  
Small part of the training set reserve for validation



# Final Notes

- The observations only from the training set shall be used for the model construction
- Use the test set to verify how accurate the constructed model performs
- Many models expect normalized, scaled data as input  
(check Box-Cox Transformation, log-scaling, etc.)

# Thank you

These slides can be found at  
<https://github.com/gakhov/pycon-ua-2018>

Andrii Gakhov, ferret go GmbH  
[www.gakhov.com](http://www.gakhov.com)

PyCon UA, Kharkiv, April 28-29, 2018