Learning Deep Broadband Network@HOME

Hongjoo LEE

europython
9-16 JULY 2017 Rimini

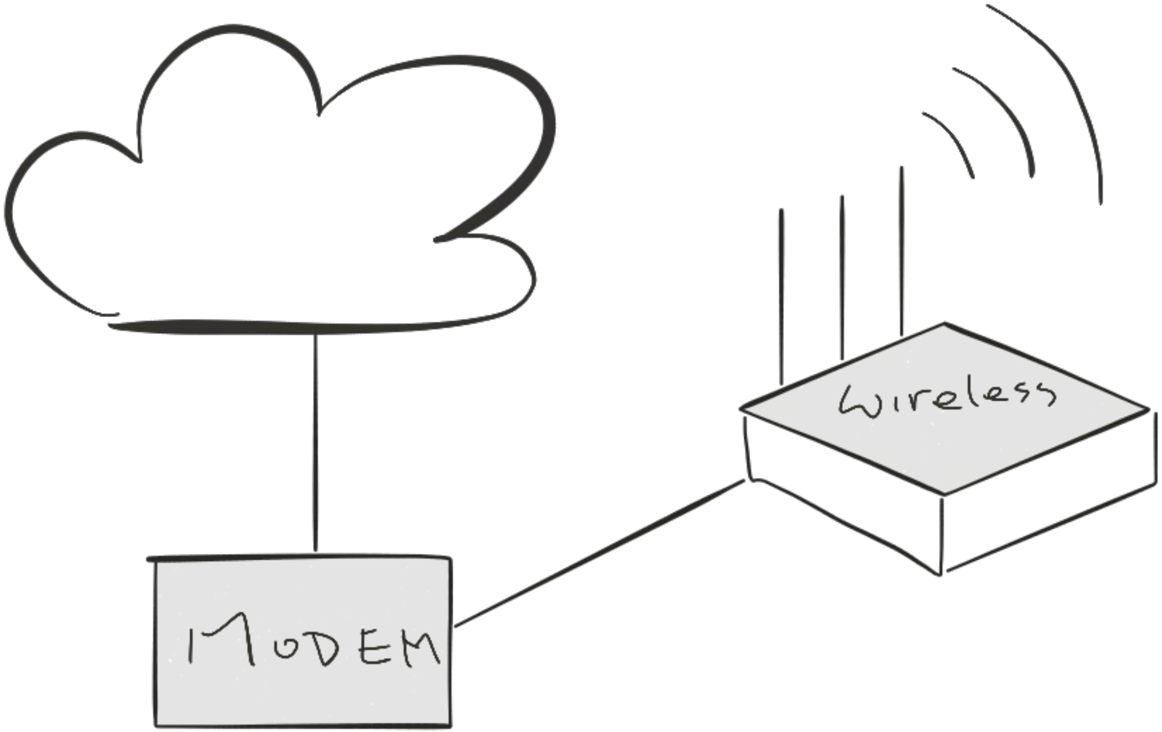
Who am I?

- Machine Learning Engineer
  - Fraud Detection System
  - Software Defect Prediction
- Software Engineer
  - Email Services (40+ mil. users)
  - High traffic server (IPC, network, concurrent programming)
- MPhil, HKUST
  - Major : Software Engineering based on ML tech
  - Research interests : ML, NLP, IR

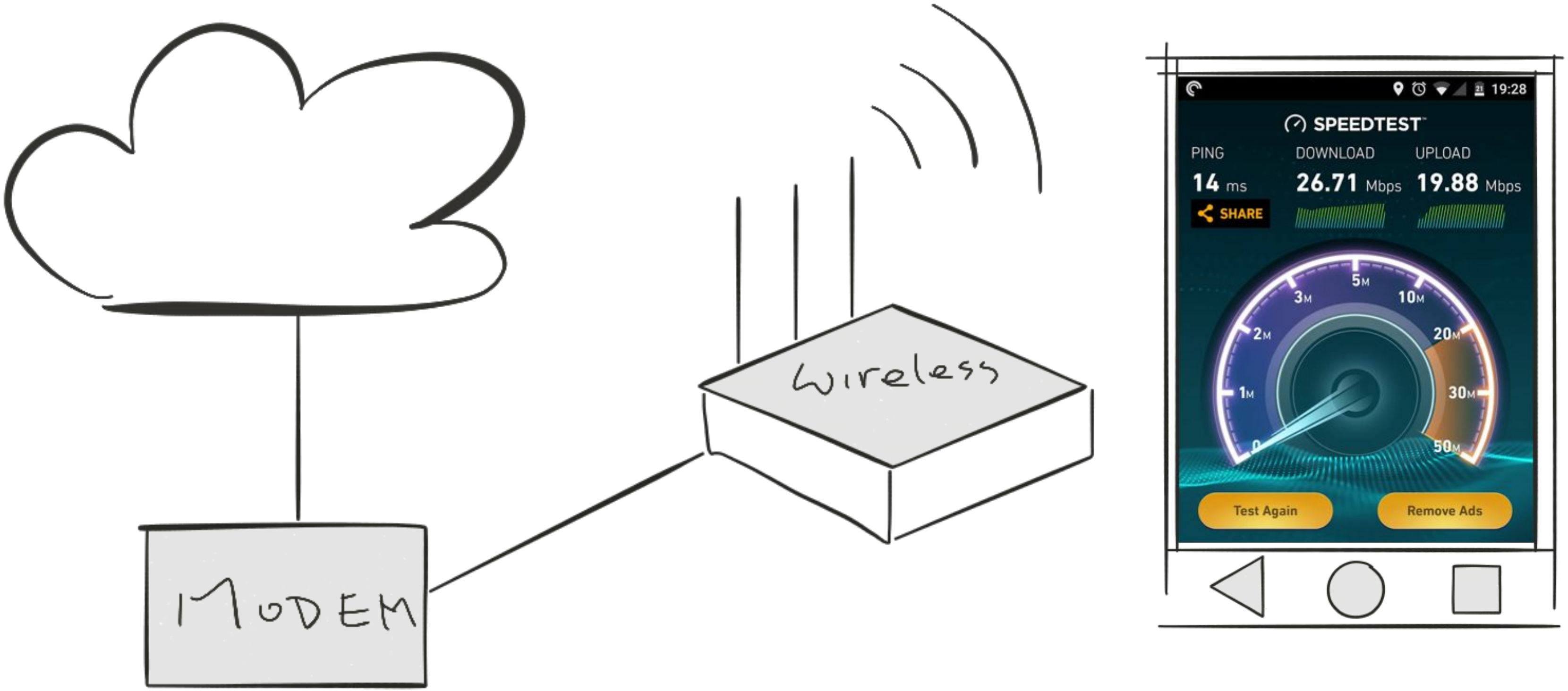
Outline

Data Collection	Time series Analysis	Forecast Modeling	Anomaly Detection
			Naive approach
Logging SpeedTest Data preparation Handling time series	Seasonal Trend Decomposition	Rolling Forecast	Basic approaches
	Stationarity Autoregression, Moving Average Autocorrelation	ARIMA	Multivariate Gaussian
		LSTM	

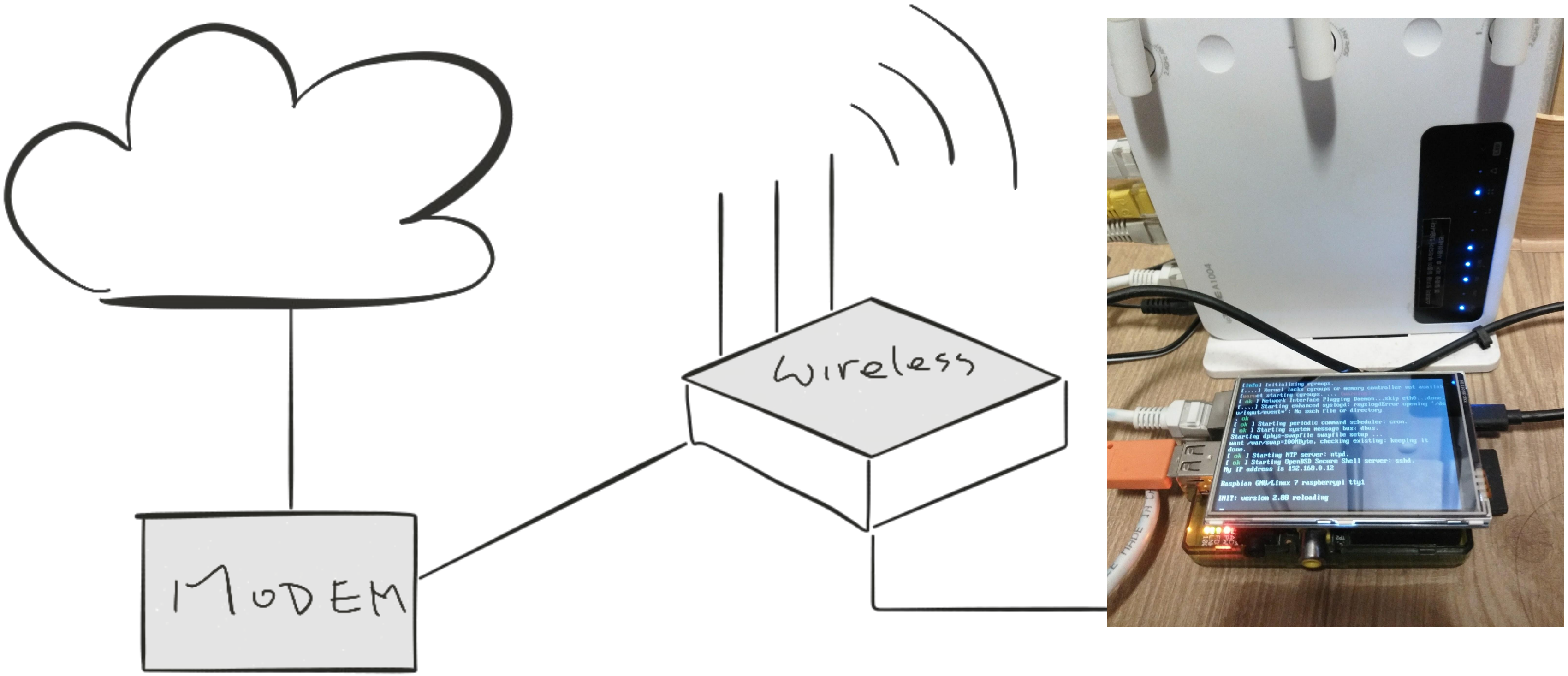
Home Network



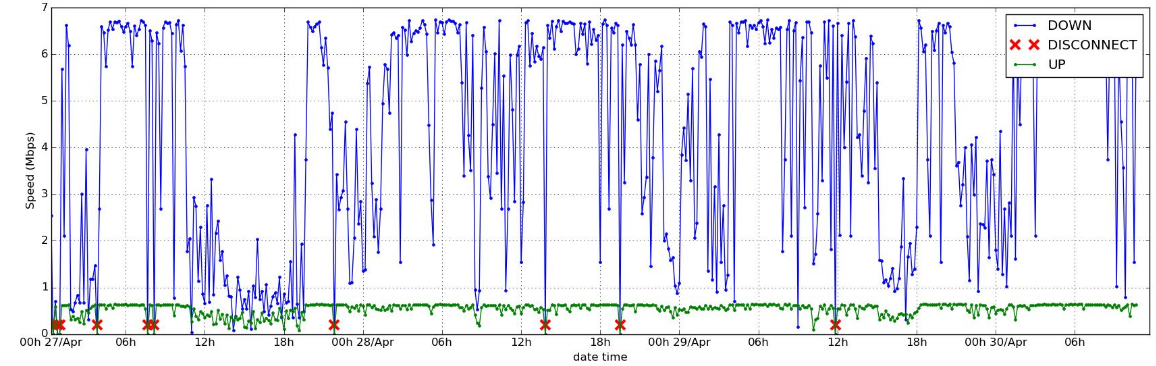
Home Network

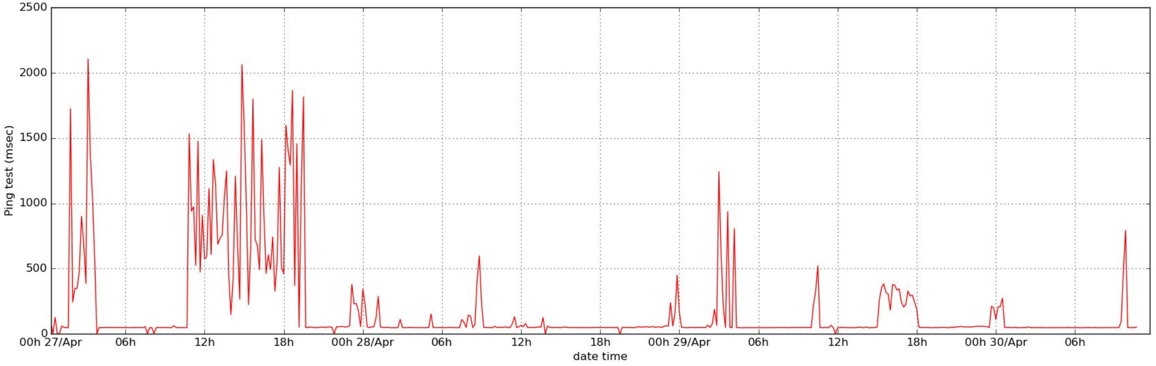


Home Network



Anomaly Detection (Naive approach in 2015)



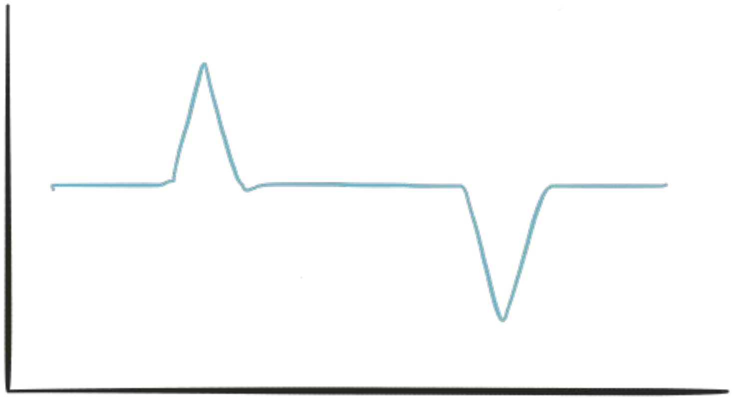


Problem definition

- Detect abnormal states of Home Network
- Anomaly detection for time series
  - Finding outlier data points relative to some usual signal

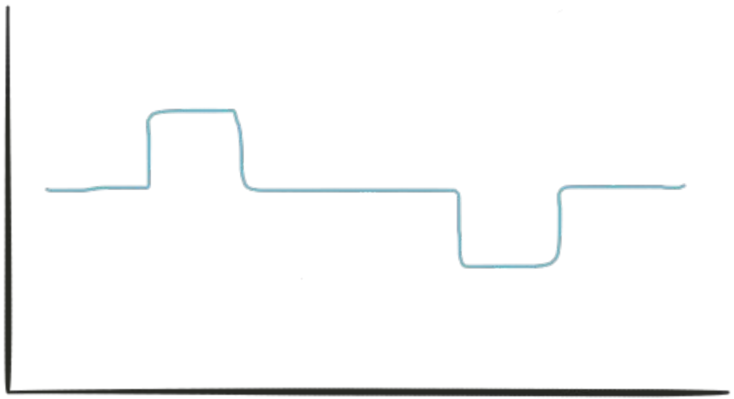
Types of anomalies in time series

- Additive outliers



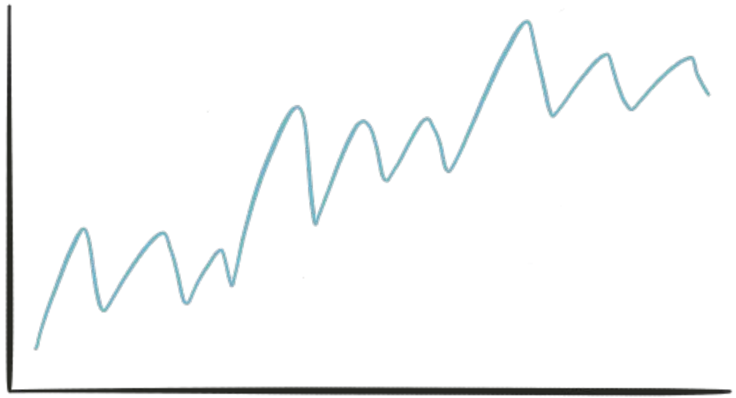
Types of anomalies in time series

- Temporal changes



Types of anomalies in time series

- Level shift



Outline

Data Collection	Time series Analysis	Forecast Modeling	Anomaly Detection
			Naive approach
Logging SpeedTest Data preparation Handling time series	Seasonal Trend Decomposition	Rolling Forecast	Basic approaches
	Stationarity Autoregression, Moving Average Autocorrelation	ARIMA	Multivariate Gaussian
		LSTM	

Logging Data

- Speedtest-cli

```
$ speedtest-cli --simple
Ping: 35.811 ms
Download: 68.08 Mbit/s
Upload: 19.43 Mbit/s
$ crontab -l
*/5 * * * * echo ‘>>> ‘$(date) >> $LOGFILE; speedtest-cli --simple >> $LOGFILE
2>&1
```

- Every 5 minutes for 3 Month. ⇒ 20k observations.

Logging Data

- Log output

```
$ more $LOGFILE
>>> Thu Apr 13 10:35:01 KST 2017
Ping: 42.978 ms
Download: 47.61 Mbit/s
Upload: 18.97 Mbit/s
>>> Thu Apr 13 10:40:01 KST 2017
Ping: 103.57 ms
Download: 33.11 Mbit/s
Upload: 18.95 Mbit/s
>>> Thu Apr 13 10:45:01 KST 2017
Ping: 47.668 ms
Download: 54.14 Mbit/s
Upload: 4.01 Mbit/s
```

Data preparation

- Parse data

```
class SpeedTest(object):
    def __init__(self, string):
        self.__string = string
        self.__pos = 0
        self.datetime = None# for DatetimeIndex
        self.ping = None          # ping test in ms
        self.download = None# down speed in Mbit/sec
        self.upload = None        # up speed in Mbit/sec

    def __iter__(self):
        return self

    def next(self):
        …
```

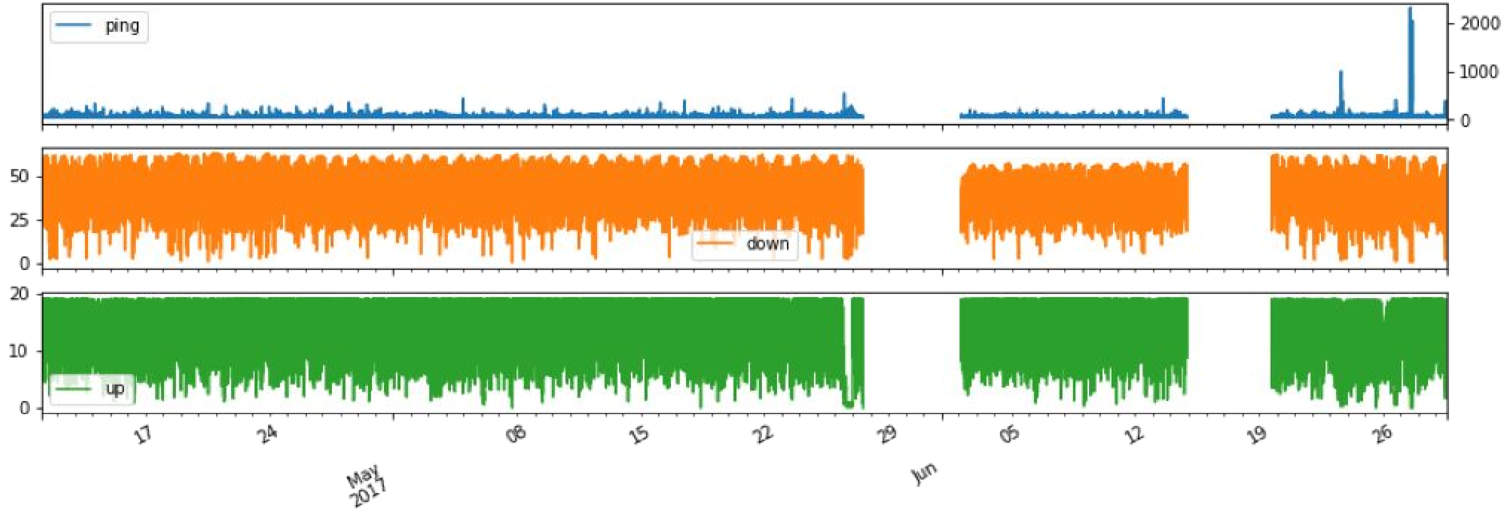
Data preparation

- Build panda DataFrame

```
speedtests = [st for st in SpeedTests(logstring)]

dt_index = pd.date_range(
                   speedtests[0].datetime.replace(second=0, microsecond=0),
                   periods=len(speedtests), freq='5min')

df = pd.DataFrame(index=dt_index,
              data=([st.ping, st.download, st.upload] for st in speedtests),
              columns=['ping','down','up'])
```

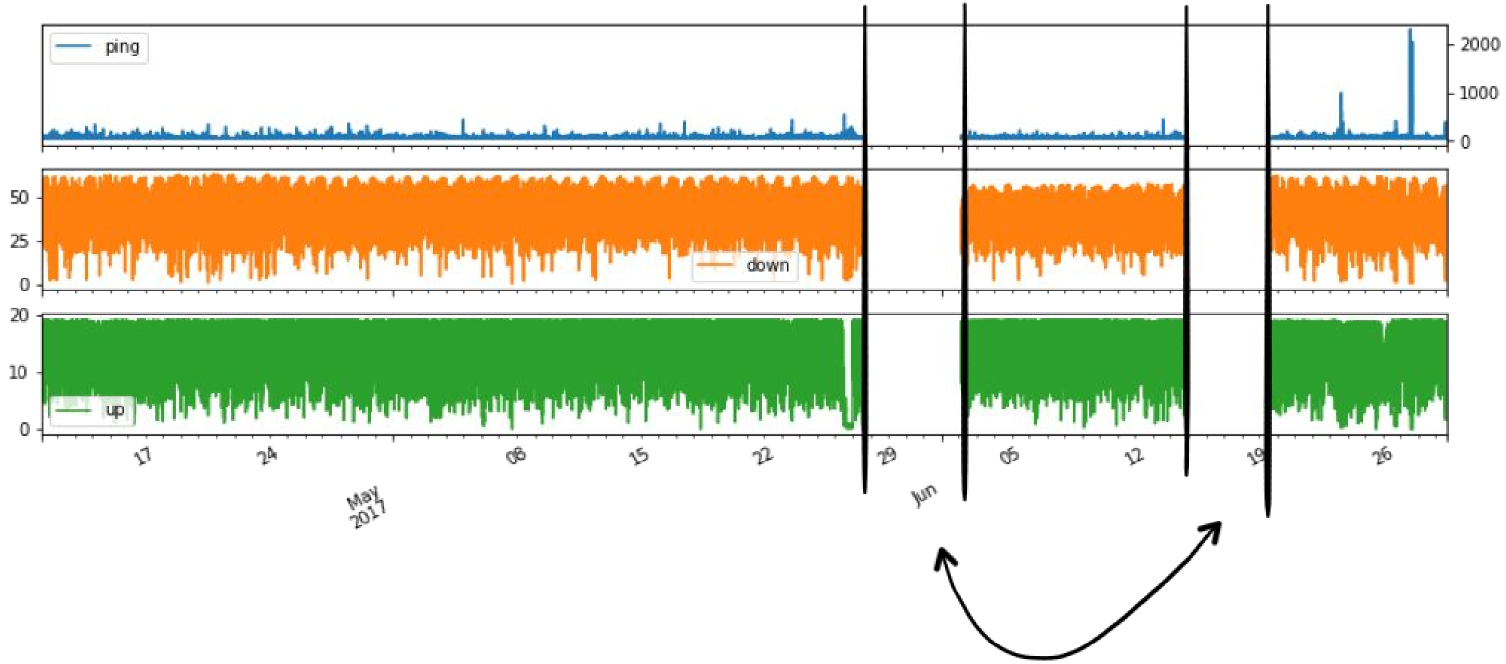
Data preparation

- Plot raw data



Data preparation

- Structural breaks
  - Accidental missings for a long period



Data preparation

- Handling missing data
  - Only a few occasional cases

In [147]:
```
df[df.ping.isnull()]
```

Out[147]:

	ping	down	up
2017-04-15 14:55:00	NaN	NaN	NaN
2017-04-16 07:50:00	NaN	NaN	NaN
2017-04-16 08:15:00	NaN	NaN	NaN
2017-04-19 17:20:00	NaN	NaN	NaN
2017-04-19 22:20:00	NaN	NaN	NaN
2017-04-20 00:00:00	NaN	NaN	NaN

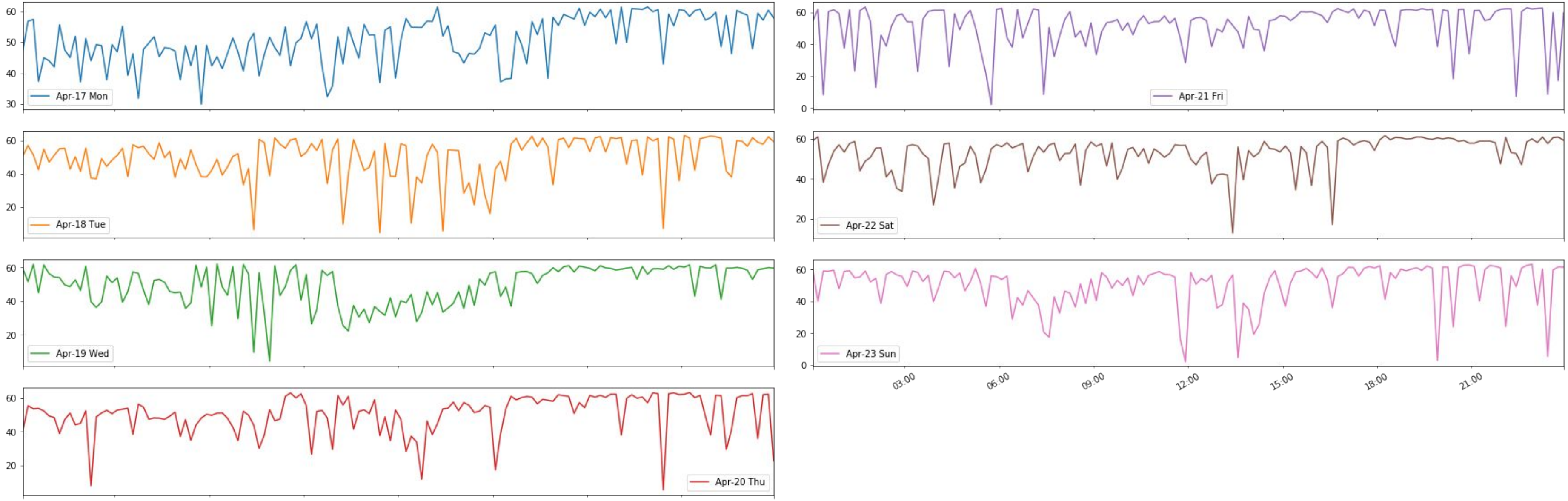
In [148]:
```
df = df.fillna(method='pad', limit=1)
```

Handling time series

- By DatetimeIndex
  - df[‘2017-04’:’2017-06’]
  - df[‘2017-04’:]
  - df[‘2017-04-01 00:00:00’:]
  - df[df.index.weekday_name == ‘Monday’]
  - df[df.index.minute == 0]
- By TimeGrouper
  - df.groupby(pd.TimeGrouper(‘D’))
  - df.groupby(pd.TimeGrouper(‘M’))

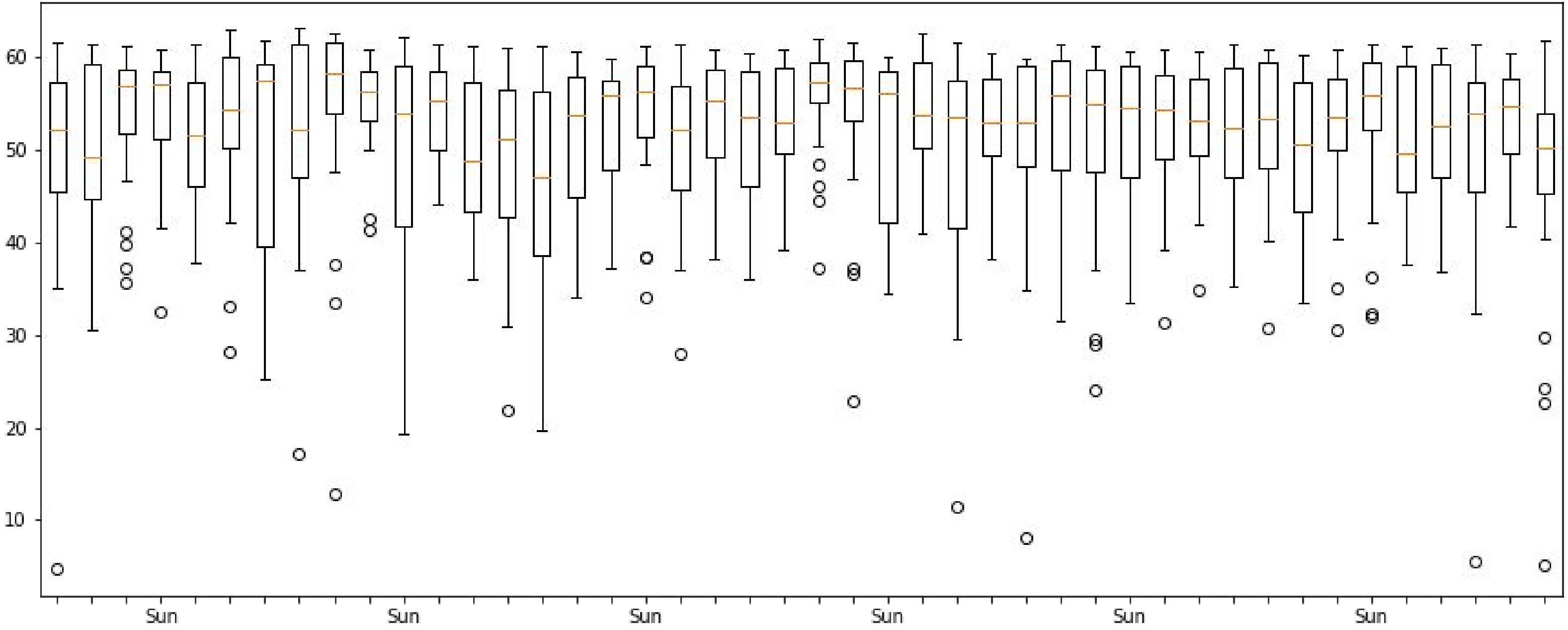
Patterns in time series

- Is there a pattern in 24 hours?



Patterns in time series

- Is there a daily pattern?



Components of Time series data

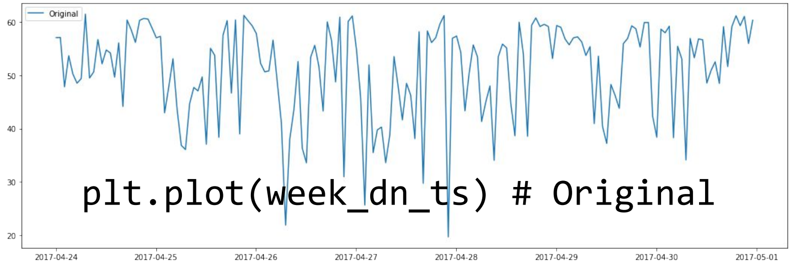
- Trend :The increasing or decreasing direction in the series.
- Seasonality : The repeating in a period in the series.
- Noise : The random variation in the series.

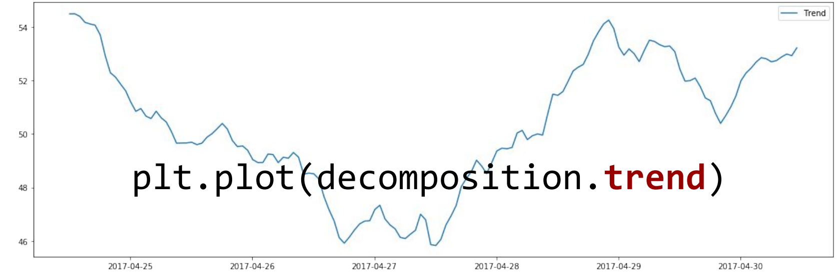
Components of Time series data

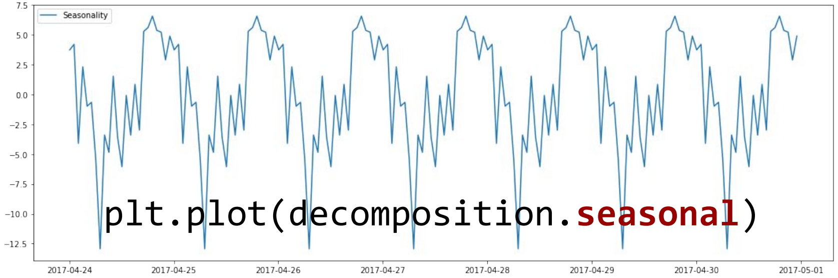
- A time series is a combination of these components.
  - $y_t = T_t + S_t + N_t$ (additive model)
  - $y_t = T_t \times S_t \times N_t$ (multiplicative model)

Seasonal Trend Decomposition

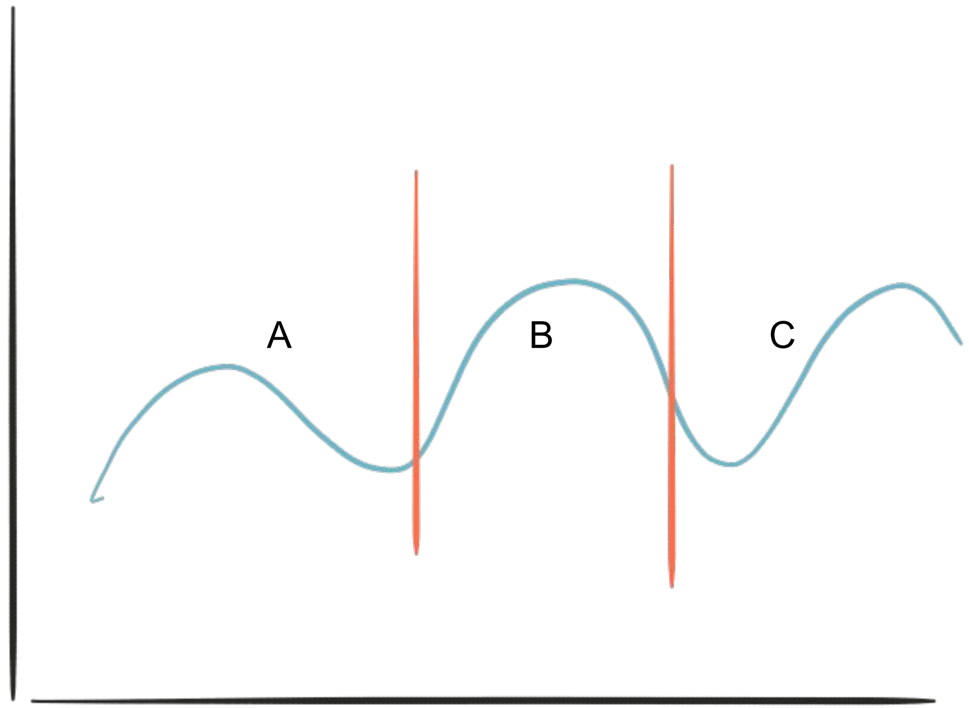
```
from statsmodels.tsa.seasonal import seasonal_decompose
decomposition = seasonal_decompose(week_dn_ts)
```







Rolling Forecast

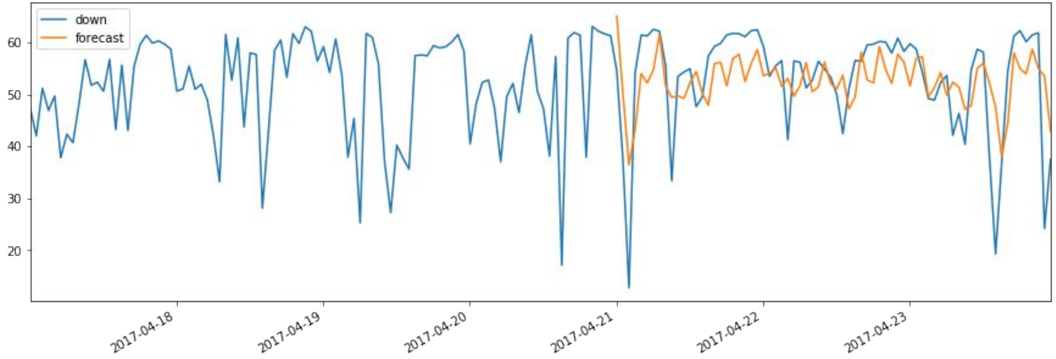


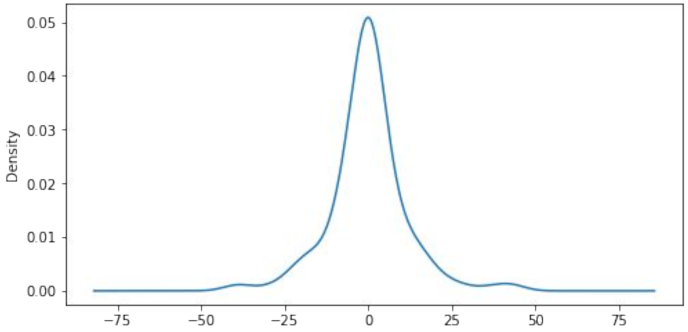
Rolling Forecast

```
from statsmodels.tsa.arima_model import ARIMA

forecasts = list()
history = [x for x in train_X]
for t in range(len(test_X)):          # for each new observation
    model = ARIMA(history, order=order)   # update the model
    y_hat = model.fit().forecast()        # forecast one step ahead
    forecasts.append(y_hat)           # store predictions
    history.append(test_X[t])             # keep history updated
```

Residuals ~ N(μ, σ^2)

```
residuals = [test[t] - forecasts[t] for t in range(len(test_X))]
residuals = pd.DataFrame(residuals)
residuals.plot(kind='kde')
```



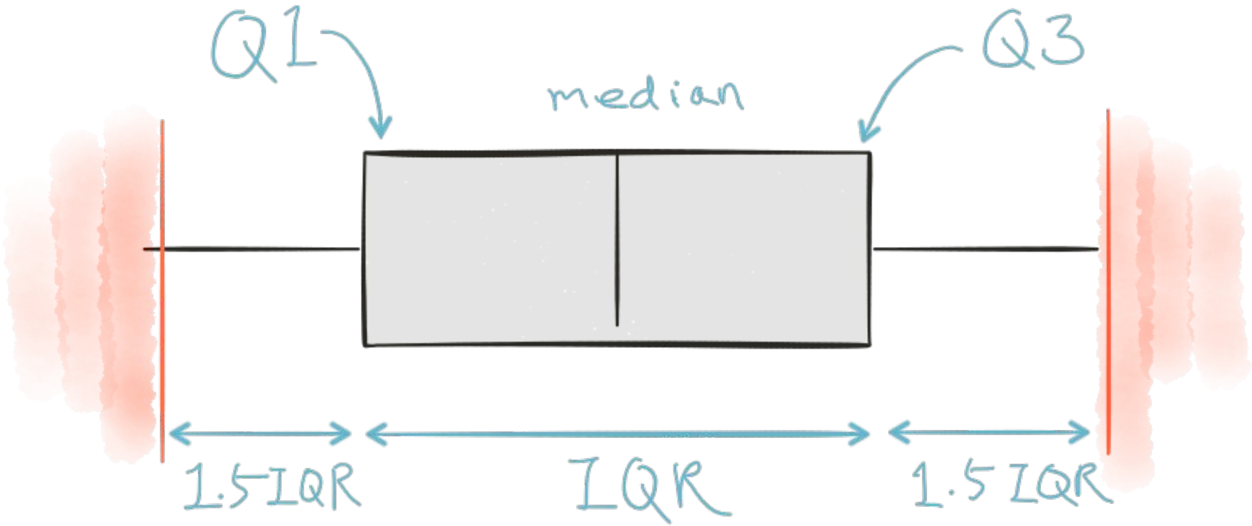


Anomaly Detection (Basic approach)

- IQR (Inter Quartile Range)
- 2-5 Standard Deviation
- MAD (Median Absolute Deviation)

Anomaly Detection (Naive approach)

- Inter Quartile Range



Anomaly Detection (Naive approach)

- Inter Quartile Range
  - NumPy

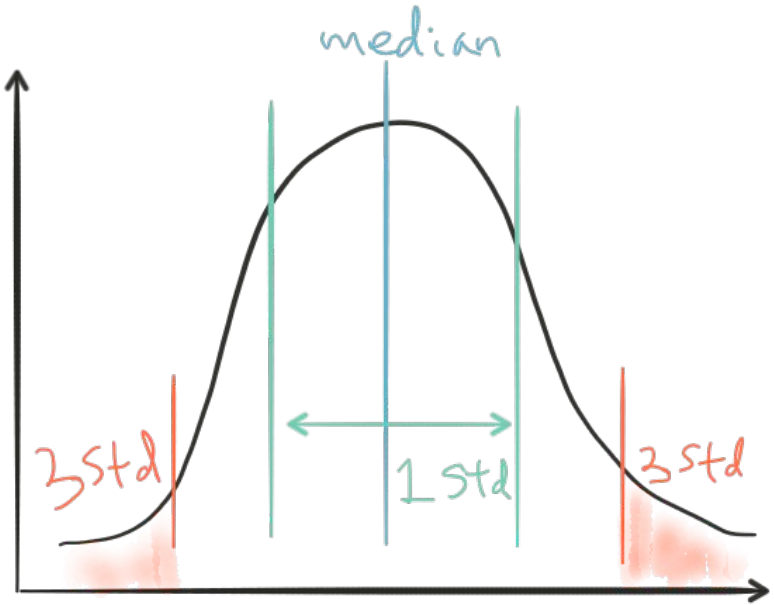
```
q1, q3 = np.percentile(col, [25, 75])
iqr = q3 - q1
np.where((col < q1 - 1.5*iqr) | (col > q3 + 1.5*iqr))
```

  - Pandas

```
q1 = df['col'].quantile(.25)
q3 = df['col'].quantile(.75)
iqr = q3 - q1
df.loc[~df['col'].between(q1-1.5*iqr, q3+1.5*iqr),'col']
```

Anomaly Detection (Naive approach)

- 2-5 Standard Deviation



Anomaly Detection (Naive approach)

- 2-5 Standard Deviation
  - NumPy

```
std = np.std(col)
med = np.median(col)
np.where((col < med - 3*std) | (col < med + 3*std))
```

  - Pandas

```
std = pd[‘col’].std()
med = pd[‘col’].median()
df.loc[~df[‘col’].between(med - 3*std, med + 3*std), 0]
```

Anomaly Detection (Naive approach)

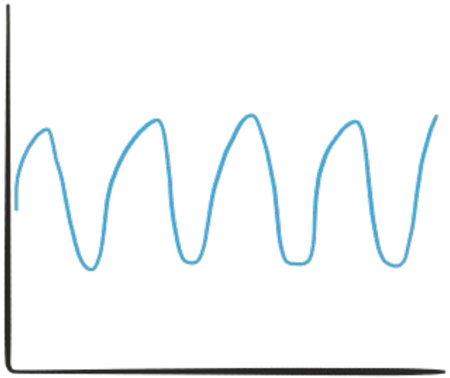
- MAD (Median Absolute Deviation)
  - MAD = median(| X_i - median(X)|)
  - “Detecting outliers: Do not use standard deviation around the mean, use absolute deviation around the median” - Christopher Leys (2013)

Outline

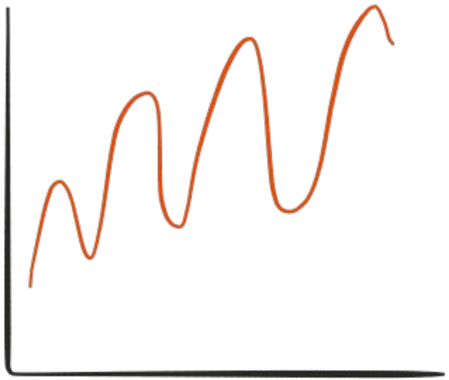
Data Collection	Time series Analysis	Forecast Modeling	Anomaly Detection
			Naive approach
Logging SpeedTest Data preparation Handling time series	Seasonal Trend Decomposition	Rolling Forecast	Basic approaches
	Stationarity Autoregression, Moving Average Autocorrelation	ARIMA	Multivariate Gaussian
		LSTM	

Stationary Series Criterion

- The **<u>mean</u>**, variance and covariance of the series are time invariant.



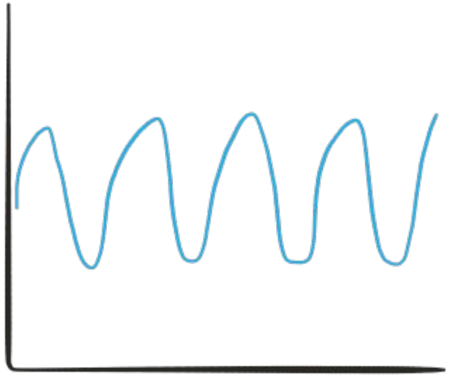
stationary



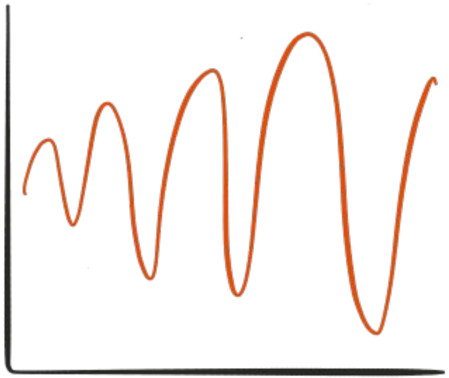
non-stationary

Stationary Series Criterion

- The mean, **<u>variance</u>** and covariance of the series are time invariant.



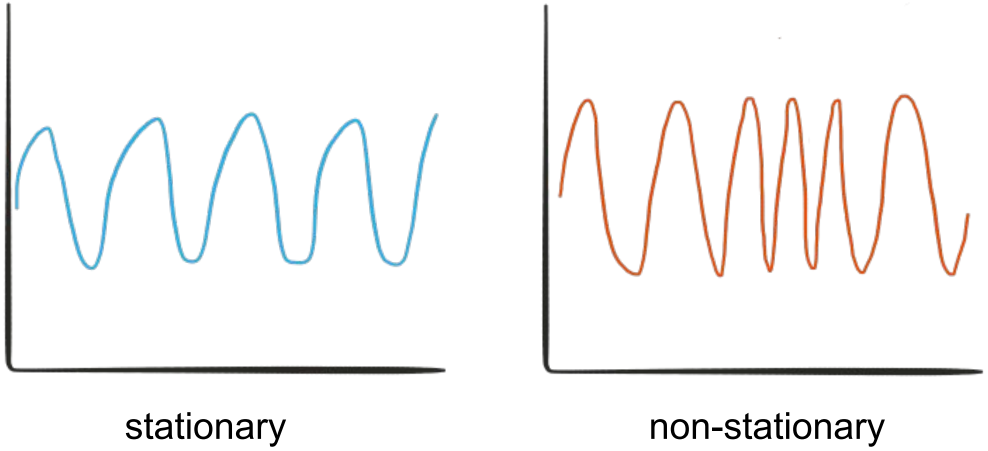
stationary



non-stationary

Stationary Series Criterion

- The mean, variance and **covariance** of the series are time invariant.



Test Stationarity

In [181]:
```
from statsmodels.tsa.stattools import adfuller

dftest = adfuller(weekly_dn_ts, autolag='AIC')

print('''Test Statistic : {:.4f}'
Critical Value (1%) : {:.4f}
Critical Value (5%) : {:.4f}
Critical Value (10%) : {:.4f}'''.format(dftest[0],
                                        dftest[4]['1%'], dftest[4]['5%'], dftest[4]['10%']))
```

```
Test Statistic : -4.0462'
Critical Value (1%) : -3.4716
Critical Value (5%) : -2.8797
Critical Value (10%) : -2.5764
```

Differencing

- A non-stationary series can be made stationary after differencing.
- Instead of modelling the level, we model the change
- Instead of forecasting the level, we forecast the change
- $I(d) = y_t - y_{t-d}$
- AR + I + MA

Autoregression (AR)

- Autoregression means developing a linear model that uses observations at previous time steps to predict observations at future time step.
- Because the regression model uses data from the same input variable at previous time steps, it is referred to as an autoregression

Moving Average (MA)

- MA models look similar to the AR component, but it's dealing with different values.
- The model account for the possibility of a relationship between a variable and the residuals from previous periods.

ARIMA(p, d, q)

- Autoregressive Integrated Moving Average
  - AR : A model that uses dependent relationship between an observation and some number of lagged observations.
  - I : The use of differencing of raw observations in order to make the time series stationary.
  - MA : A model that uses the dependency between an observation and a residual error from a MA model.
- parameters of ARIMA model
  - p : The number of lag observations included in the model
  - d : the degree of differencing, the number of times that raw observations are differenced
  - q : The size of moving average window.

Identification of ARIMA

- Autocorrelation function(ACF) : measured by a simple correlation between current observation Y_t and the observation p lags from the current one Y_{t-p}.
- Partial Autocorrelation Function (PACF) : measured by the degree of association between Y_t and Y_{t-p} when the effects at other intermediate time lags between Y_t and Y_{t-p} are removed.
- Inference from ACF and PACF : theoretical ACFs and PACFs are available for various values of the lags of AR and MA components. Therefore, plotting ACFs and PACFs versus lags and comparing leads to the selection of the appropriate parameter p and q for ARIMA model

Identification of ARIMA (easy case)

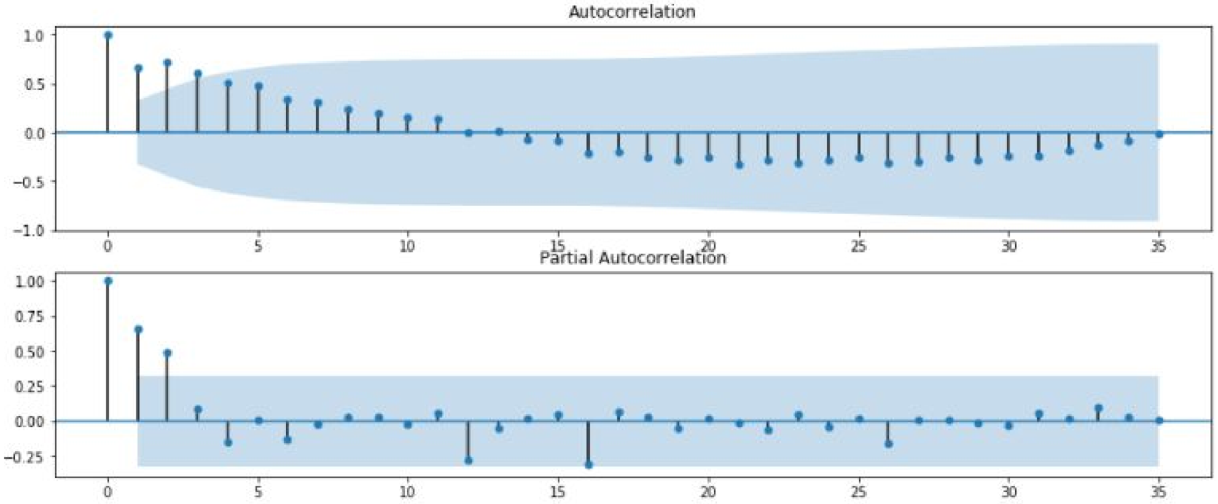
- General characteristics of theoretical ACFs and PACFs

model	ACF	PACF
AR(p)	Tail off; Spikes decay towards zero	Spikes cutoff to zero after lag p
MA(q)	Spikes cutoff to zero after lag q	Tails off; Spikes decay towards zero
ARMA(p,q)	Tails off; Spikes decay towards zero	Tails off; Spikes decay towards zero

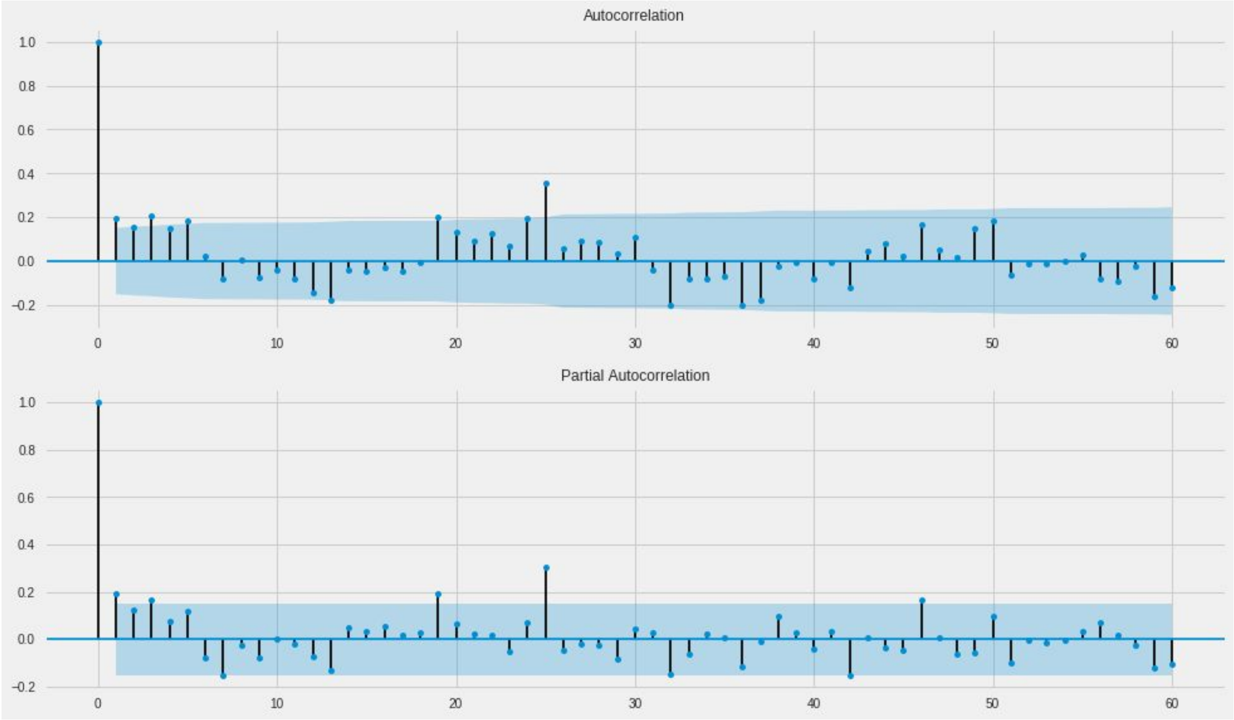
- Reference :
  - http://people.duke.edu/~rnau/411arim3.htm
  - Prof. Robert Nau

Identification of ARIMA (easy case)

In [25]:
```
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf

fig = plt.figure(figsize=(15,6))
ax1 = fig.add_subplot(211)
plot_acf(series, ax=ax1)
ax2 = fig.add_subplot(212)
plot_pacf(series, ax=ax2)
plt.show()
```

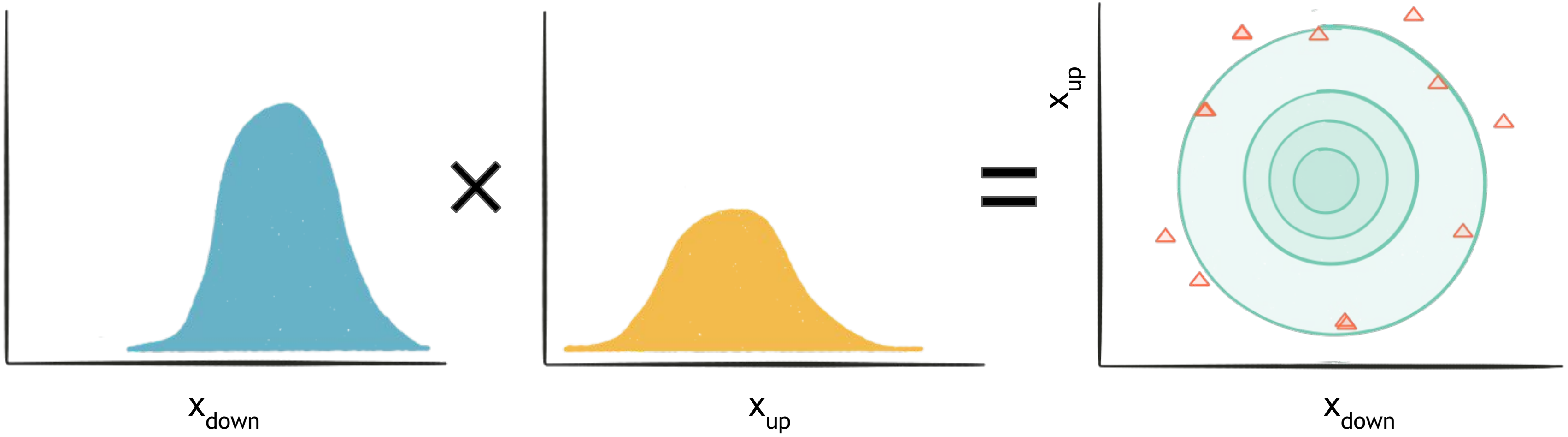


Identification of ARIMA (complicated)



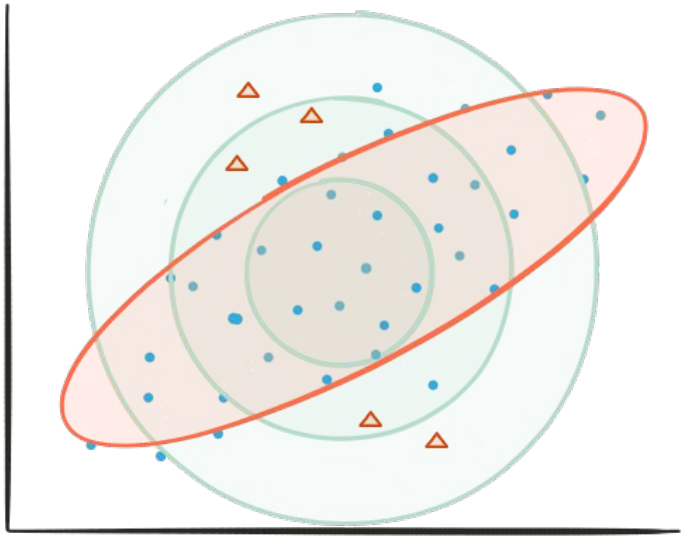
Anomaly Detection (Parameter Estimation)

$x_{dn} \sim N(\mu_{dn}\ ,\ \sigma_{dn}^2)$ $\qquad$ $x_{up} \sim N(\mu_{up}\ ,\ \sigma_{up}^2)$



$$P(x) = P(x_{dn} \mid \mu_{dn}\ ,\ \sigma_{dn}^2) \times P(x_{up} \mid \mu_{up}\ ,\ \sigma_{up}^2),\ y = \begin{cases} 1 & if\ P(x_{test}) < \varepsilon\ (anomaly) \\ 0 & if\ P(x_{test}) \geq \varepsilon\ (normal) \end{cases}$$

Anomaly Detection (Multivariate Gaussian Distribution)



$$\mu = \frac{1}{m}\sum_{i=1}^{m} x(i)$$

$$\Sigma = \frac{1}{m}\sum_{i=1}^{m} (x(i)-\mu)\cdot(x(i)-\mu)^T$$

$$p(x) = \frac{1}{(2\pi)^{\frac{\pi}{2}} |\Sigma|^{\frac{1}{2}}} exp\left(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right) \ , \ y = \begin{cases} 1 & if \ \ P(x_{test}) < \varepsilon \ \ (anomaly) \\ 0 & if \ \ P(x_{test}) \geq \varepsilon \ \ (normal) \end{cases}$$

Anomaly Detection (Multivariate Gaussian)

```
import numpy as np
from scipy.stats import multivariate_normal

def estimate_gaussian(dataset):
     mu = np.mean(dataset, axis=0)
     sigma = np.cov(dataset.T)
     return mu, sigma

def multivariate_gaussian(dataset, mu, sigma):
     p = multivariate_normal(mean=mu, cov=sigma)
     return p.pdf(dataset)

mu, sigma = estimate_gaussian(train_X)
p = multivariate_gaussian(train_X, mu, sigma)
anomalies = np.where(p < ep)                    # ep : threshold
```

$$\mu = \frac{1}{m}\sum_{i=1}^{m} x(i)$$

$$\Sigma = \frac{1}{m}\sum_{i=1}^{m} (x(i)-\mu)\cdot(x(i)-\mu)^T$$

Anomaly Detection (Multivariate Gaussian)

```
import numpy as np
from scipy.stats import multivariate_normal

def estimate_gaussian(dataset):
     mu = np.mean(dataset, axis=0)
     sigma = np.cov(dataset.T)
     return mu, sigma

def multivariate_gaussian(dataset, mu, sigma):
     p = multivariate_normal(mean=mu, cov=sigma)
     return p.pdf(dataset)

mu, sigma = estimate_gaussian(train_X)
p = multivariate_gaussian(train_X, mu, sigma)
anomalies = np.where(p < ep)                    # ep : threshold
```

$$p(x) = \frac{1}{(2\pi)^{\frac{\pi}{2}} |\Sigma|^{\frac{1}{2}}} exp\left(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right)$$

Anomaly Detection (Multivariate Gaussian)

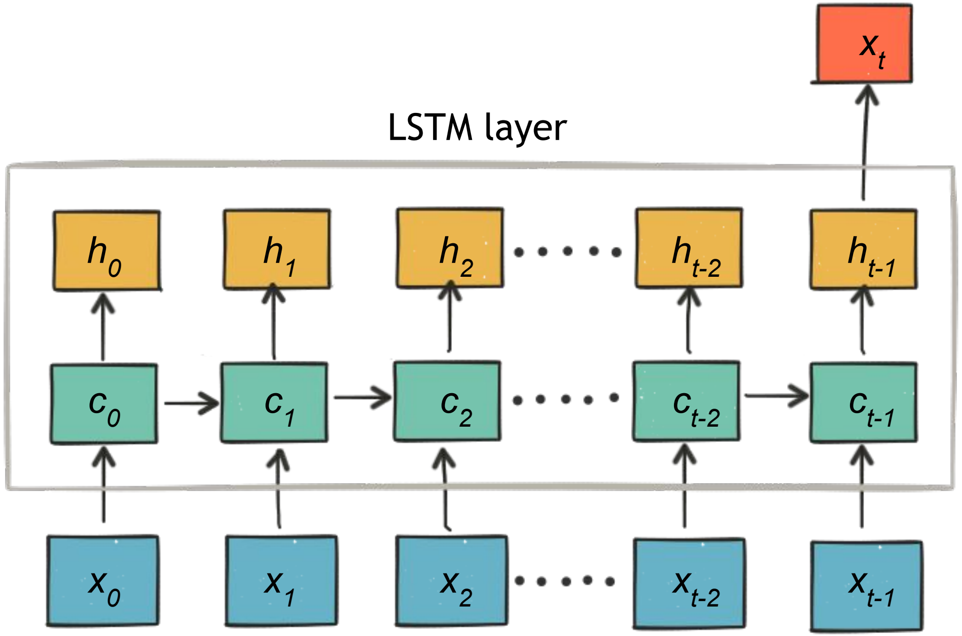
```
import numpy as np
from scipy.stats import multivariate_normal

def estimate_gaussian(dataset):
     mu = np.mean(dataset, axis=0)
     sigma = np.cov(dataset.T)
     return mu, sigma

def multivariate_gaussian(dataset, mu, sigma):
     p = multivariate_normal(mean=mu, cov=sigma)
     return p.pdf(dataset)

mu, sigma = estimate_gaussian(train_X)
p = multivariate_gaussian(train_X, mu, sigma)
anomalies = np.where(p < ep)                    # ep : threshold
```

$$y = \begin{cases} 1 & \text{if } P(x_{test}) < \varepsilon \ (anomaly) \\ 0 & \text{if } P(x_{test}) \geq \varepsilon \ (normal) \end{cases}$$

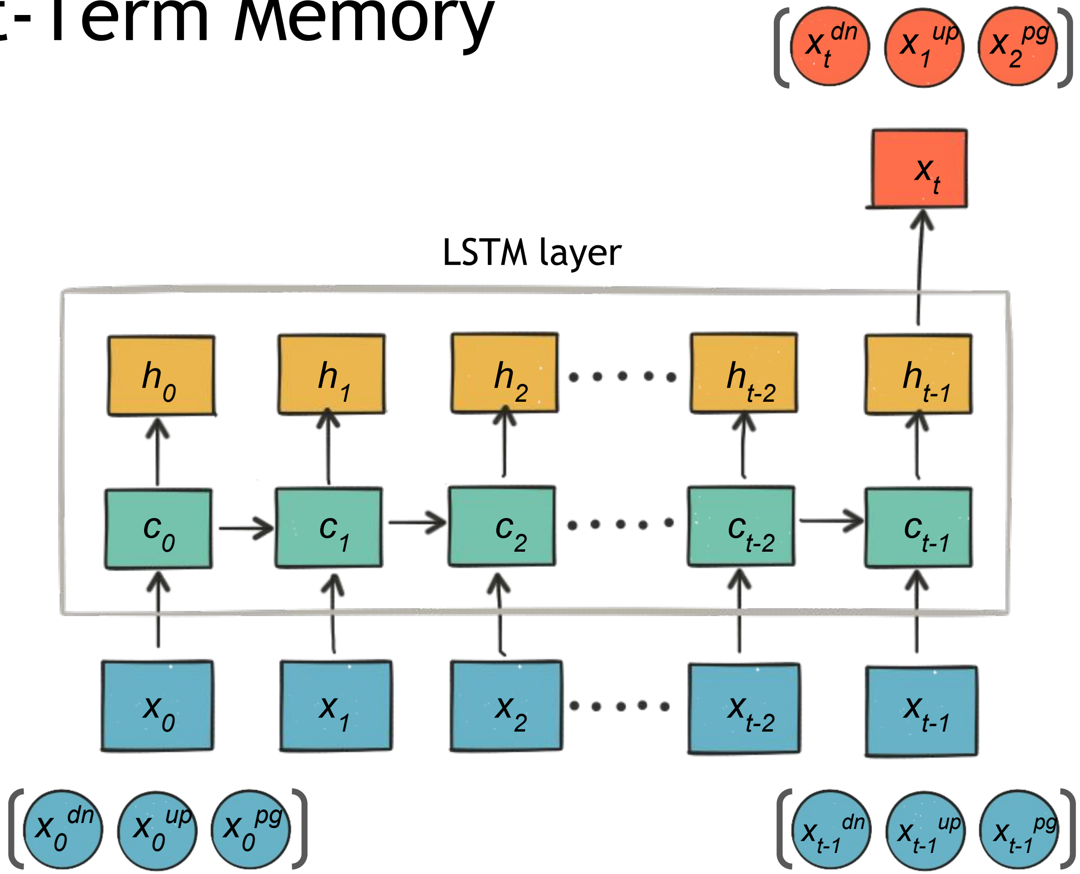
Outline

Data Collection	Time series Analysis	Forecast Modeling	Anomaly Detection
			Naive approach
Logging SpeedTest Data preparation Handling time series	Seasonal Trend Decomposition	Rolling Forecast	Basic approaches
	Stationarity Autoregression, Moving Average Autocorrelation	ARIMA	Multivariate Gaussian
		LSTM	

Long Short-Term Memory



Long Short-Term Memory



Long Short-Term Memory

```
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from sklearn.metrics import mean_squared_error

model = Sequential()
model.add(LSTM(num_neurons, stateful=True, return_sequences=True,
                   batch_input_shape=(batch_size, timesteps, input_dimension))
model.add(LSTM(num_neurons, stateful=True,
                   batch_input_shape=(batch_size, timesteps, input_dimension))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='adam')
for i in range(num_epoch):
     model.fit(train_X, y, epochs=1, batch_size=batch_size, shuffle=False)
     model.reset_states()
```

Long Short-Term Memory

- Will allow to model sophisticated and seasonal dependencies in time series
- Very helpful with multiple time series
- On going research, requires a lot of work to build model for time series

Summary

- Be prepared before calling engineers for service failures
- Pythonista has all the powerful tools
  - **pandas** is great for handling time series
  - **statsmodels** for analyzing and modeling time series
  - **sklearn** is such a multi-tool in data science
  - **keras** is good to start deep learning
- Pythonista needs to understand a few concepts before using the tools
  - Stationarity in time series
  - Autoregressive and Moving Average
  - Means of forecasting, anomaly detection
- Deep Learning for forecasting time series
  - still on-going research
- Do try this at home

Contacts

lee.hongjoo@yandex.com

linkedin.com/in/hongjoo-lee