Introduction to time series and stationarity

ARIMA MODELS IN PYTHON

James Fulton
Climate informatics researcher

DataCamp

Motivation

Time series are everywhere

- Science
- Technology
- Business
- Finance
- Policy

Course content

You will learn

- Structure of ARIMA models
- How to fit ARIMA model
- How to optimize the model
- How to make forecasts
- How to calculate uncertainty in predictions

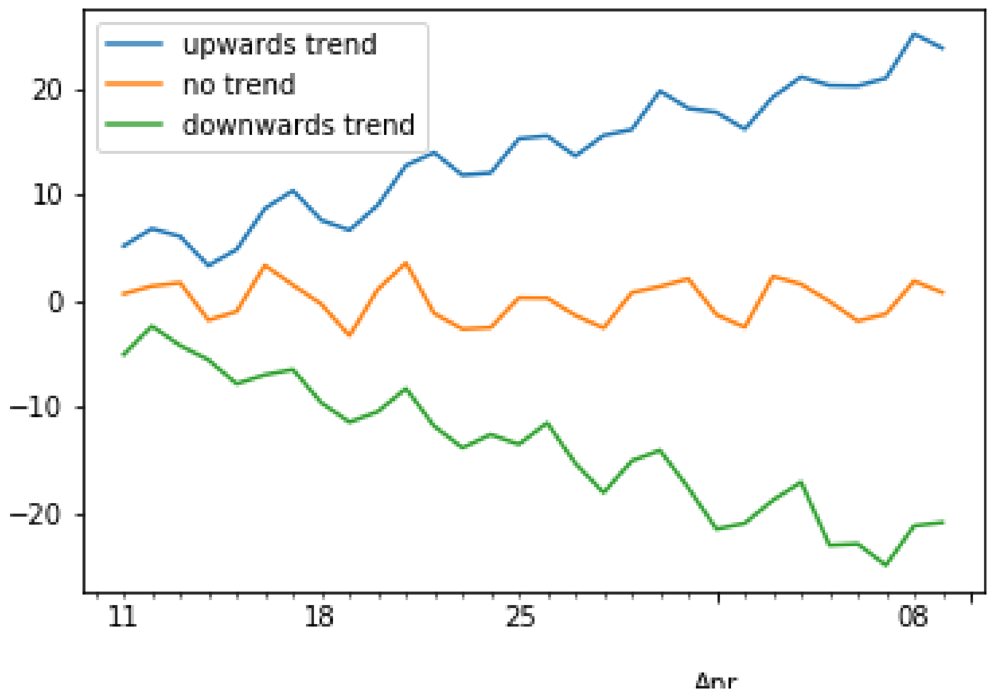
Loading and plotting

```python
import pandas as pd
import matplotlib as plt

df = pd.read_csv('time_series.csv', index_col='date', parse_dates=True)
```

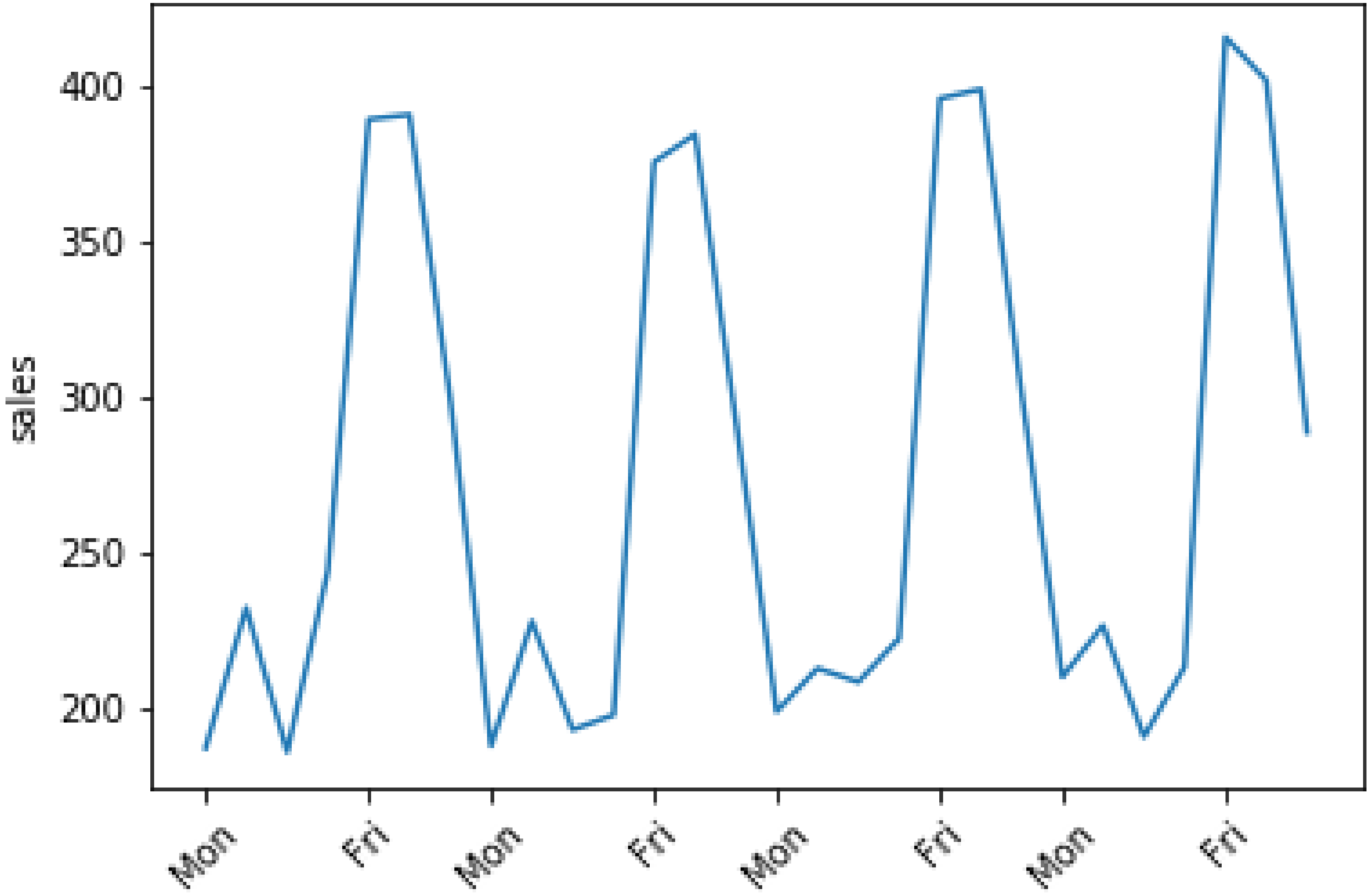
```
date                values
2019-03-11        5.734193
2019-03-12        6.288708
2019-03-13        5.205788
2019-03-14        3.176578
```

Trend

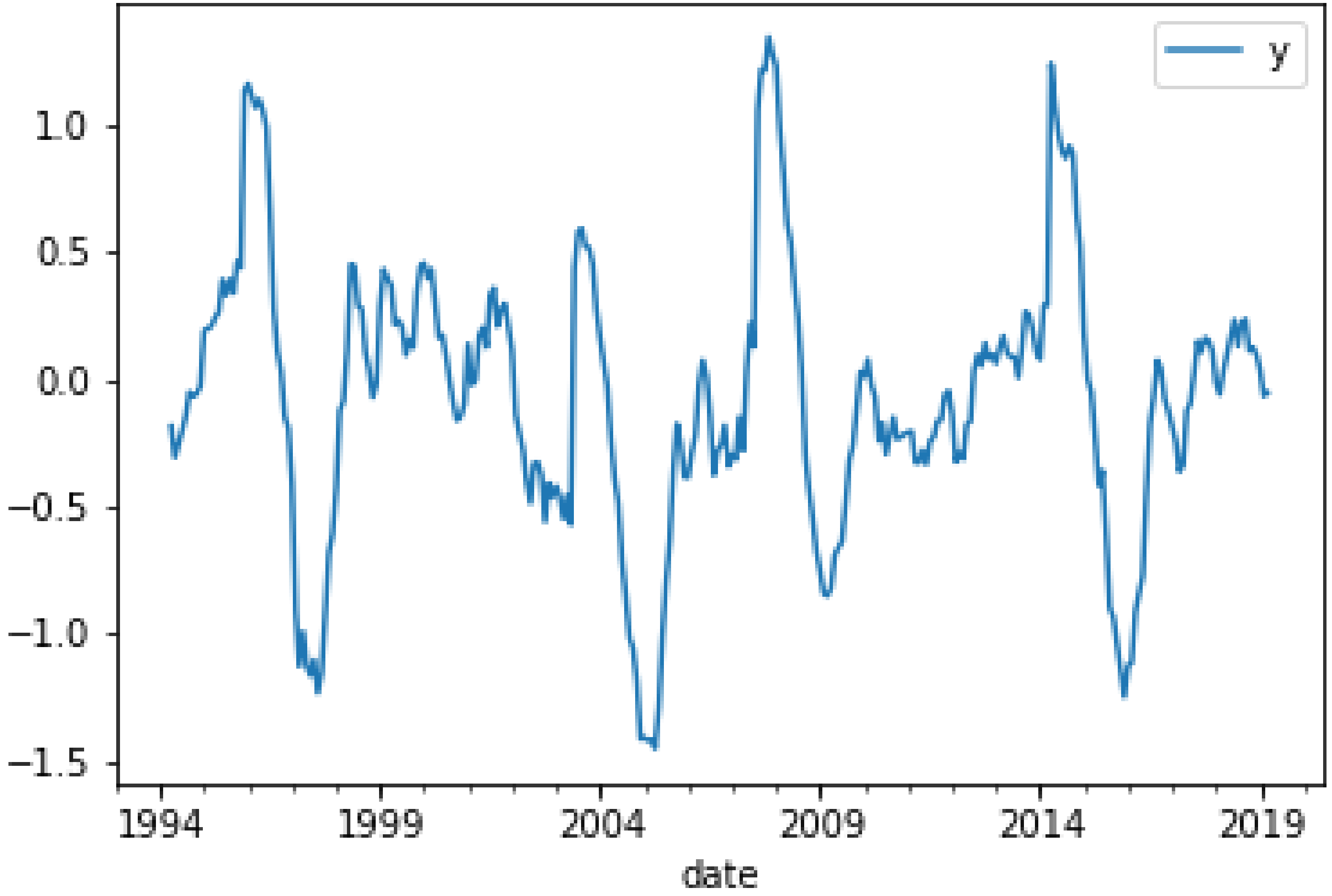
```
fig, ax = plt.subplots()
df.plot(ax=ax)
plt.show()
```



Seasonality



Cyclicality



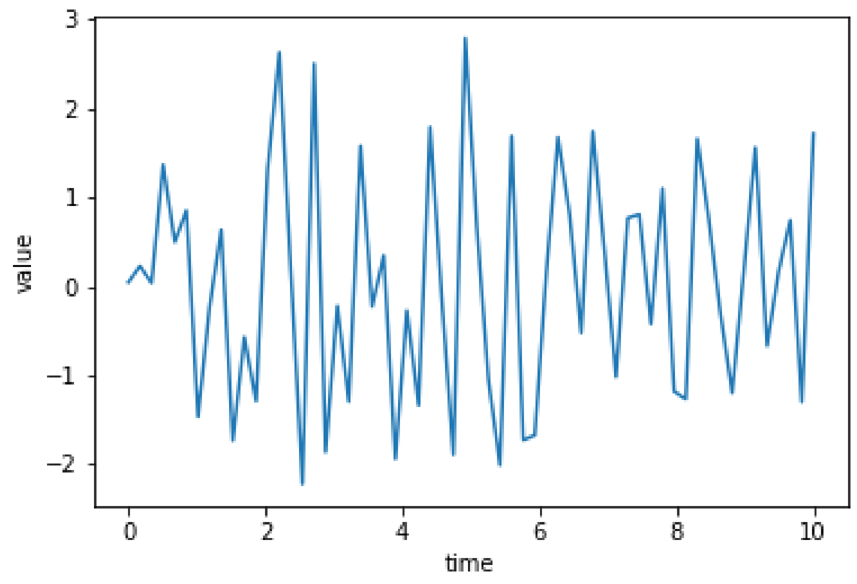
White noise

White noise series has uncorrelated values

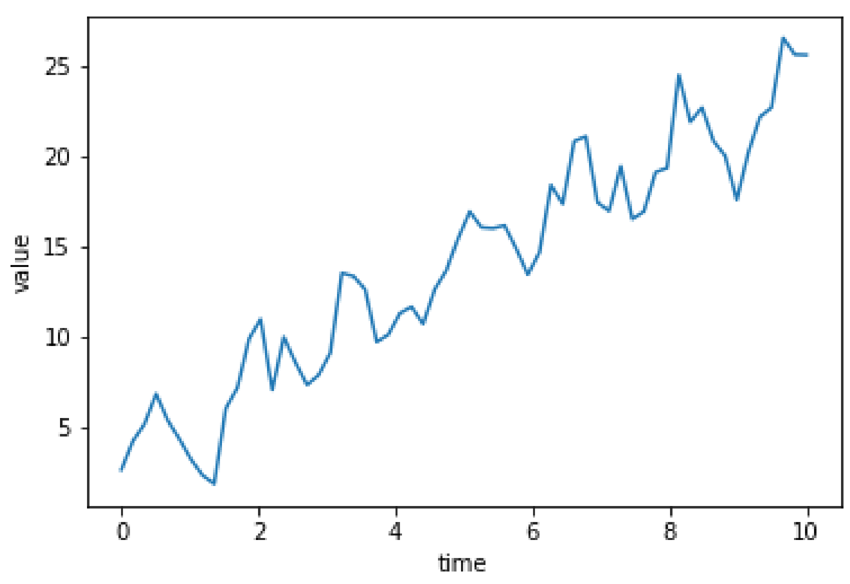
- Heads, heads, heads, tails, heads, tails, ...
- 0.1, -0.3, 0.8, 0.4, -0.5, 0.9, ...

Stationarity

Stationary



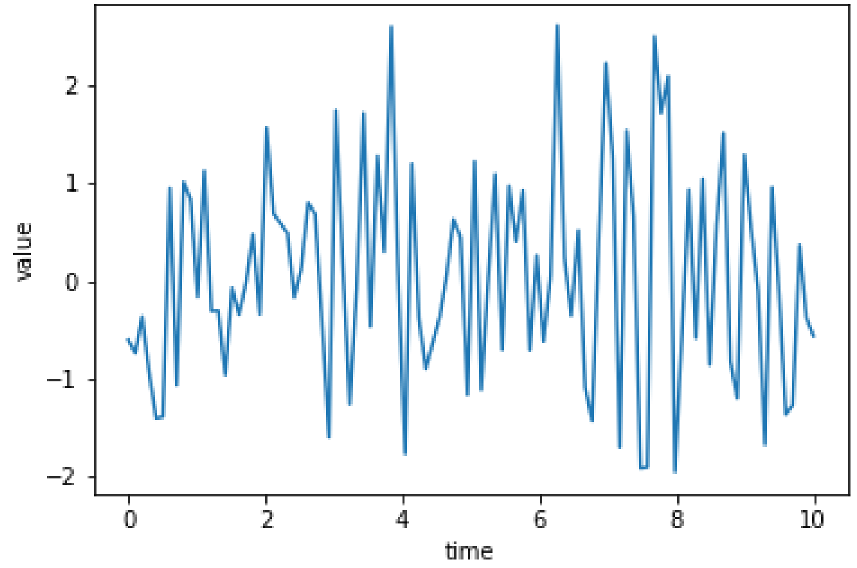
Not stationary



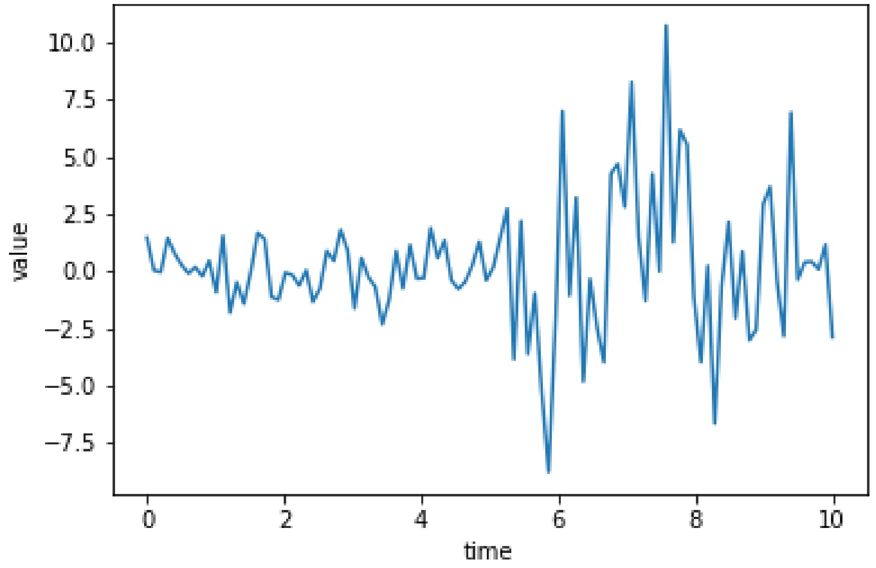
- Trend stationary: Trend is zero

Stationarity

Stationary



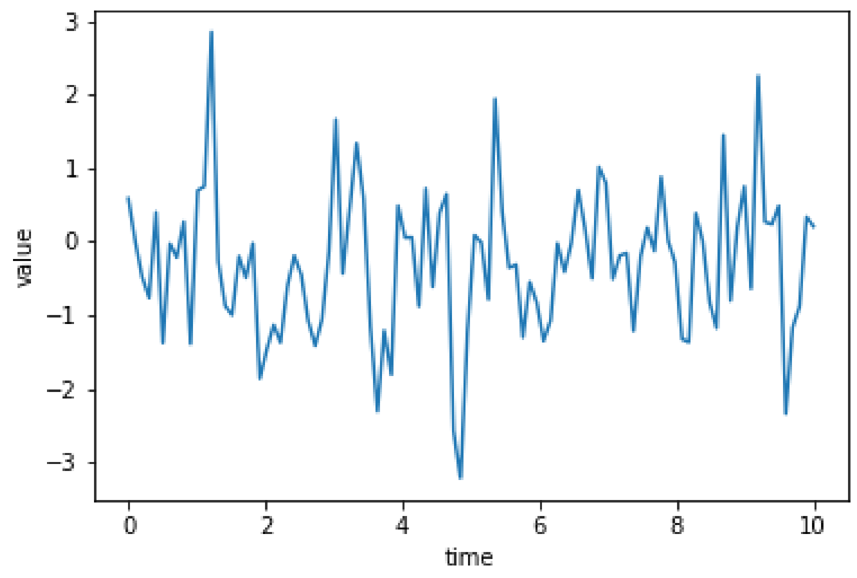
Not stationary



- Trend stationary: Trend is zero
- Variance is constant

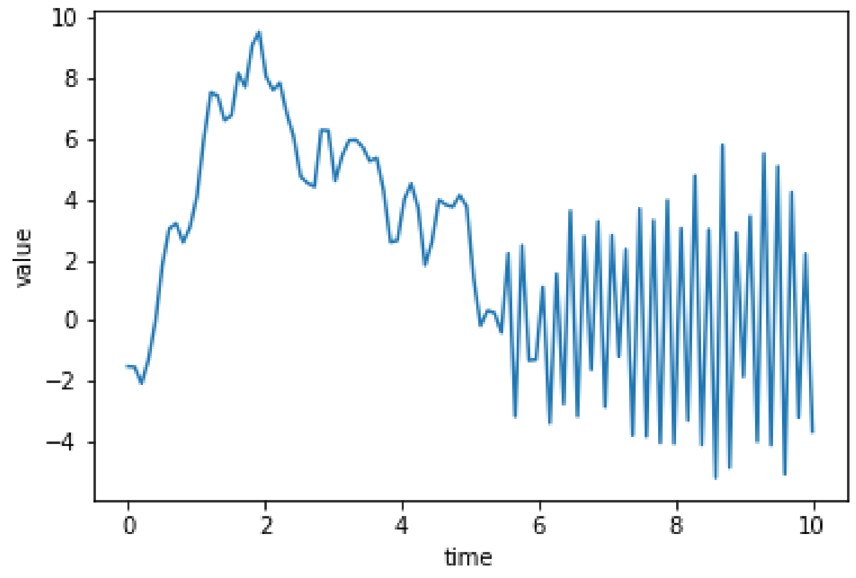
Stationarity

Stationary



- Trend stationary: Trend is zero
- Variance is constant
- Autocorrelation is constant

Not stationary



Train-test split

```
# Train data - all data up to the end of 2018
df_train = df.loc[:'2018']

# Test data - all data from 2019 onwards
df_test = df.loc['2019':]
```

Let's Practice!

ARIMA MODELS IN PYTHON

DataCamp

Making time series stationary

ARIMA MODELS IN PYTHON

James Fulton
Climate informatics researcher

DataCamp

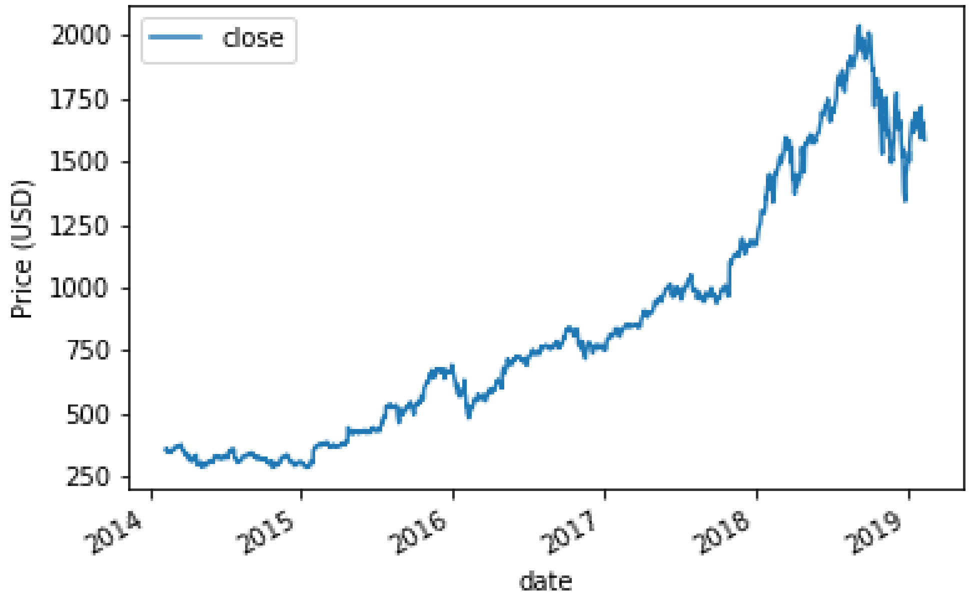
Overview

- Statistical tests for stationarity
- Making a dataset stationary

The augmented Dicky-Fuller test

- Tests for trend non-stationarity
- Null hypothesis is time series is non-stationary

Applying the adfuller test




```python
from statsmodels.tsa.stattools import adfuller

results = adfuller(df['close'])
```

Interpreting the test result

```
print(results)
```

The result is a tuple.

```
(-1.34, 0.60, 23, 1235, {'1%': -3.435, '5%': -2.913, '10%': -2.568}, 10782.87)
```

- 0th element is test statistic (-1.34)
  - More negative means more likely to be stationary
- 1st element is p-value: (0.60)
  - If p-value is small → reject null hypothesis. Reject non-stationary.
- 4th element is the critical test statistics

Interpreting the test result

```
print(results)
```

```
(-1.34, 0.60, 23, 1235, {'1%': -3.435, '5%': -2.863, '10%': -2.568}, 10782.87)
```

- 0th element is test statistic (-1.34)
  - More negative means more likely to be stationary
- 1st element is p-value: (0.60)
  - If p-value is small → reject null hypothesis. Reject non-stationary.
- 4th element is the critical test statistics

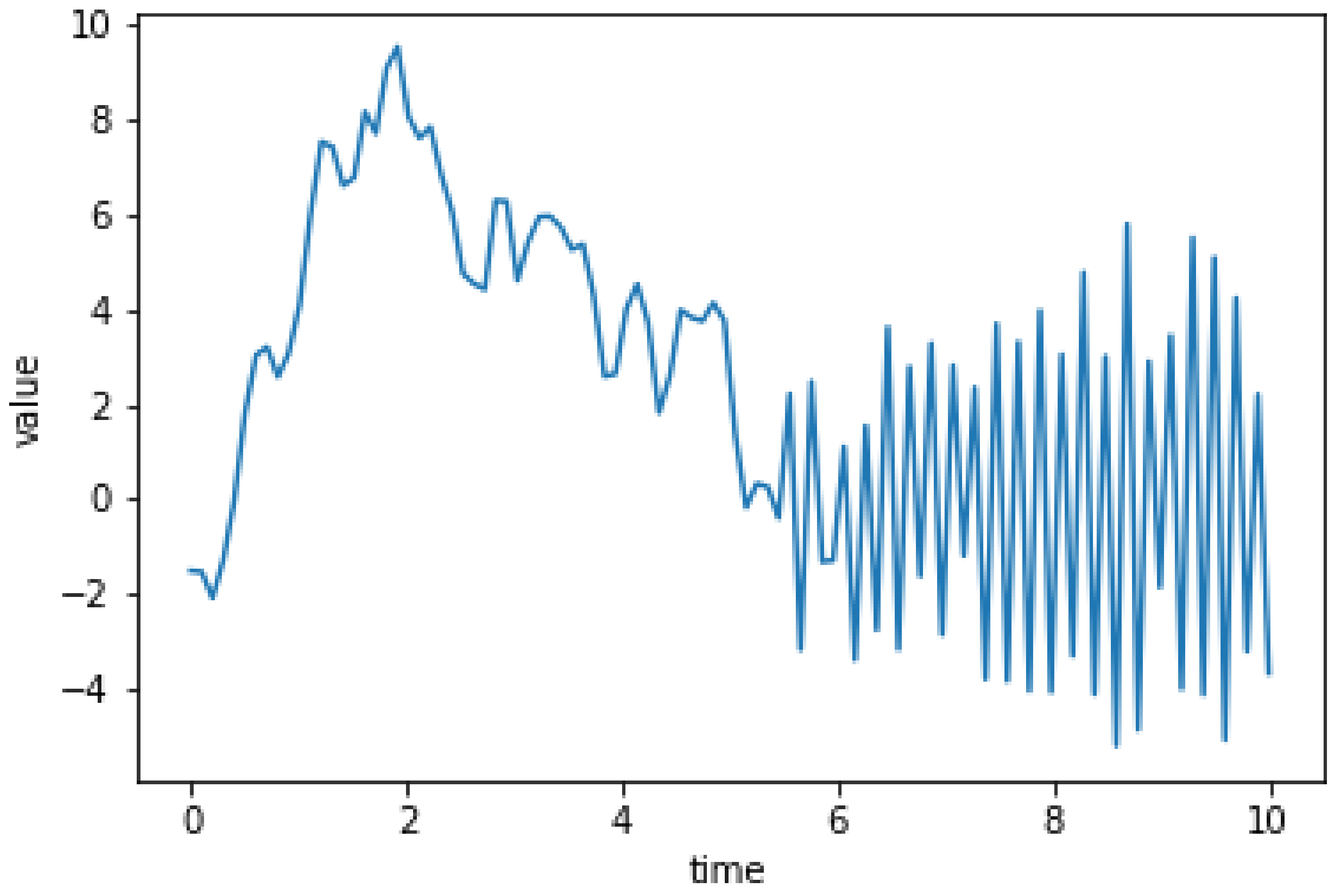
[1] https://www.statsmodels.org/dev/generated/statsmodels.tsa.stattools.adfuller.html

The value of plotting

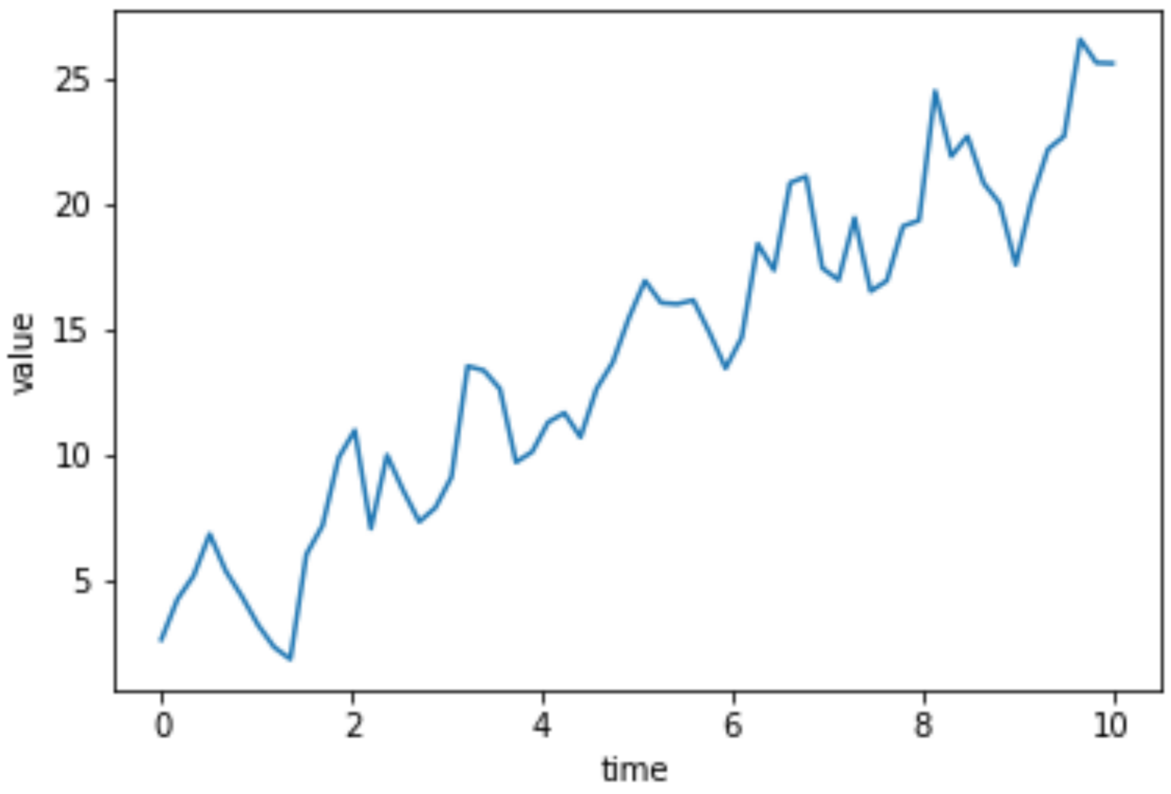
- Plotting time series can stop you making wrong assumptions

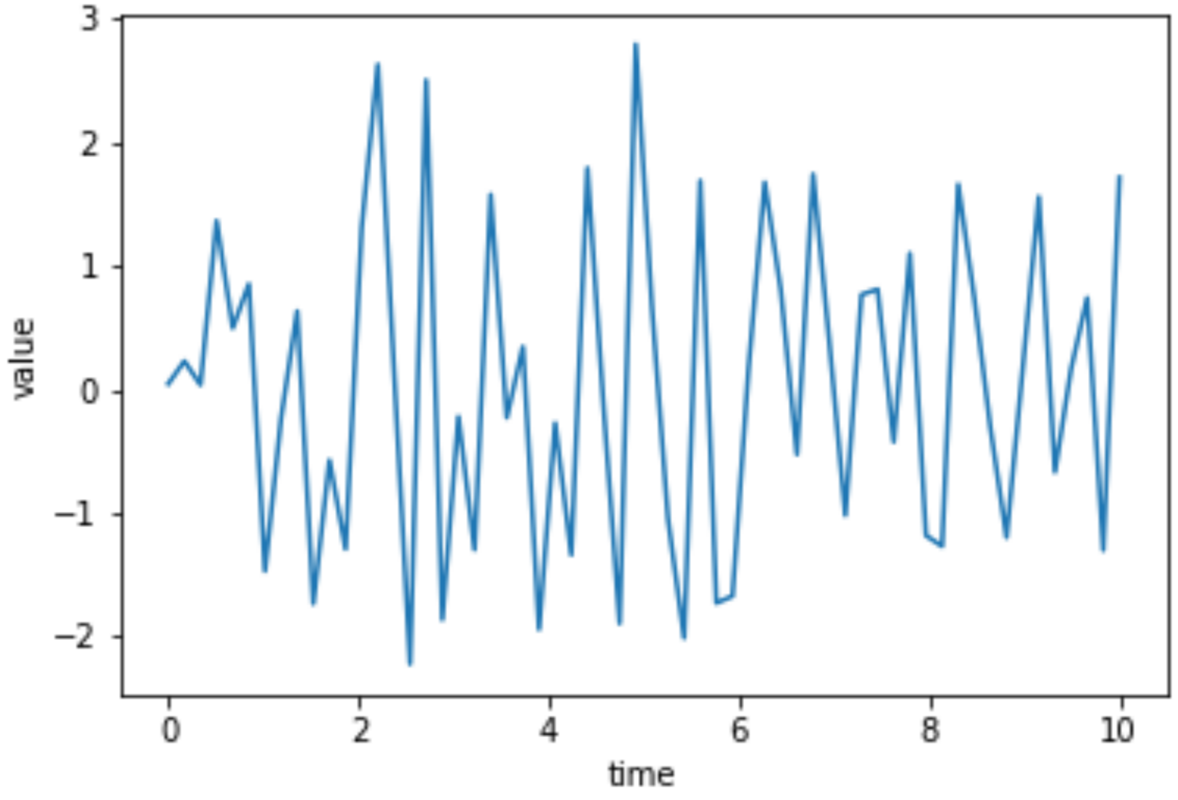
The value of plotting

The Dicky-Fuller only tests for trend stationarity.
In this example, although the time series behavior clearly changes, and is non-stationary, it passes the Dicky-Fuller test.

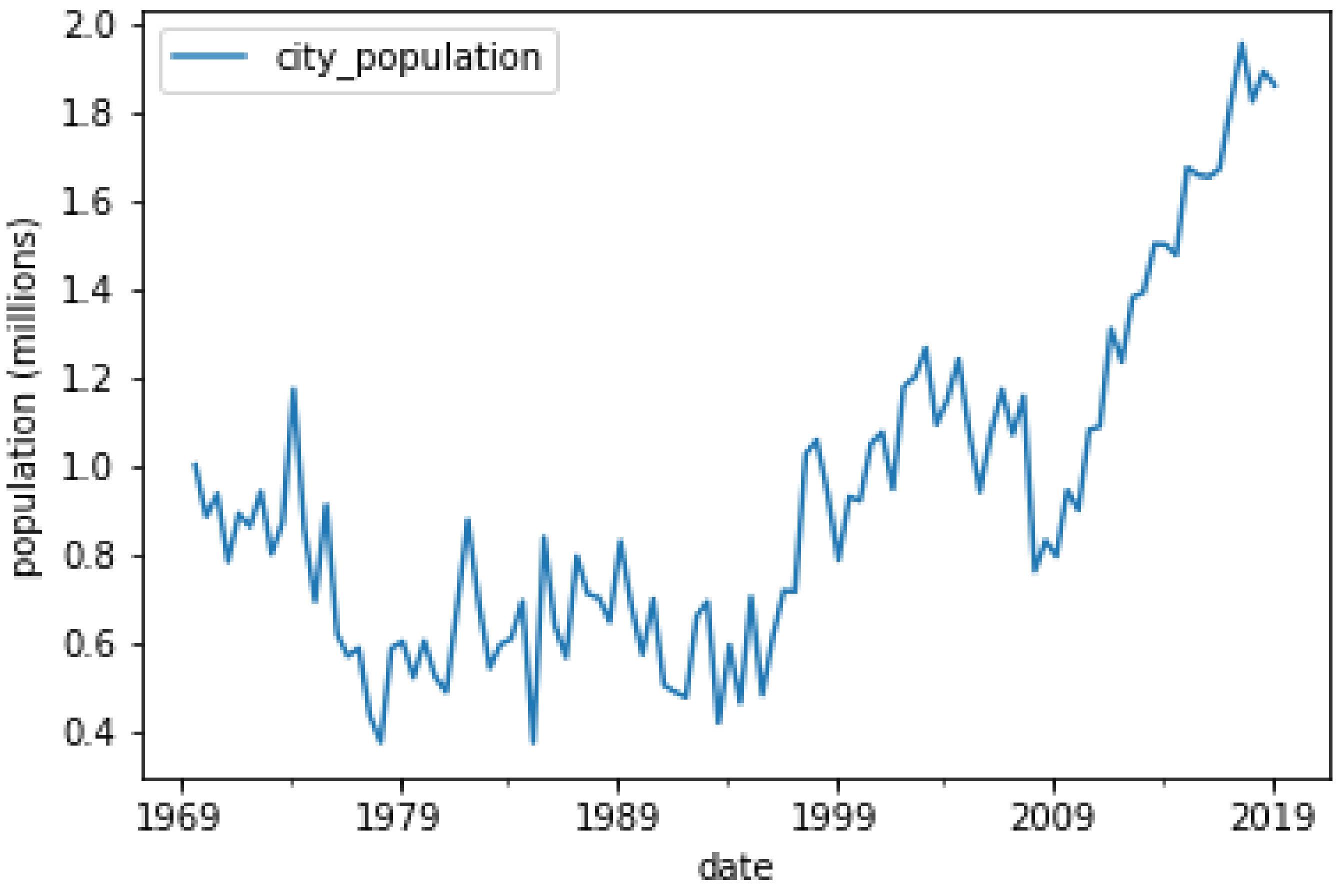


Making a time series stationary





Taking the difference



Difference: $\Delta y_t = y_t - y_{t-1}$

Taking the difference

```
df_stationary = df.diff()
```

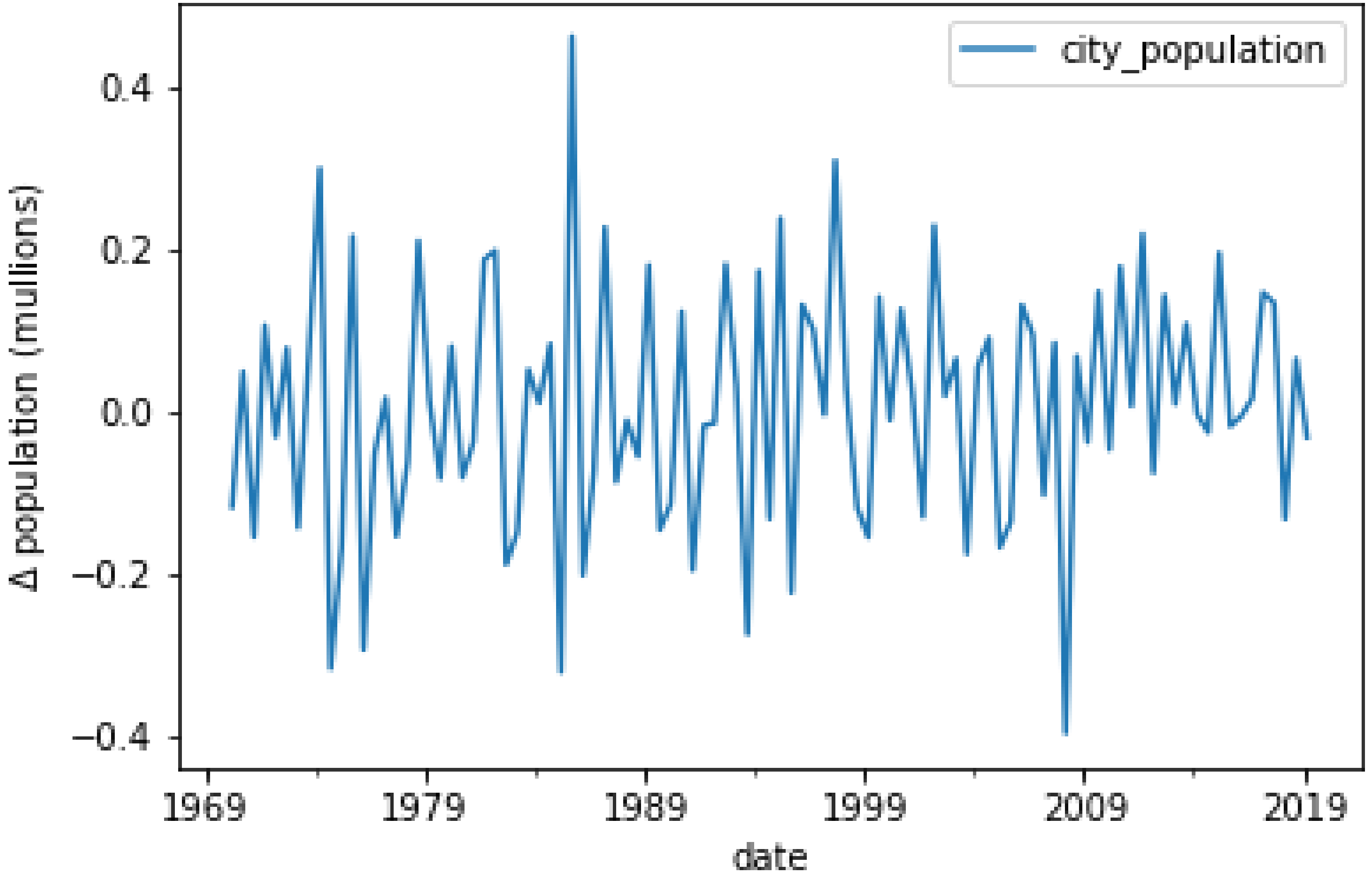
```
            city_population
date
1969-09-30              NaN
1970-03-31        -0.116156
1970-09-30         0.050850
1971-03-31        -0.153261
1971-09-30         0.108389
```

Taking the difference

```
df_stationary = df.diff().dropna()
```

```
            city_population
date
1970-03-31        -0.116156
1970-09-30         0.050850
1971-03-31        -0.153261
1971-09-30         0.108389
1972-03-31        -0.029569
```

Taking the difference



Other transforms

Examples of other transforms

- Take the log
  - `np.log(df)`
- Take the square root
  - `np.sqrt(df)`
- Take the proportional change
  - `df.shift(1)/df`

Let's practice!

ARIMA MODELS IN PYTHON

DataCamp

Intro to AR, MA and ARMA models

ARIMA MODELS IN PYTHON

James Fulton
Climate informatics researcher

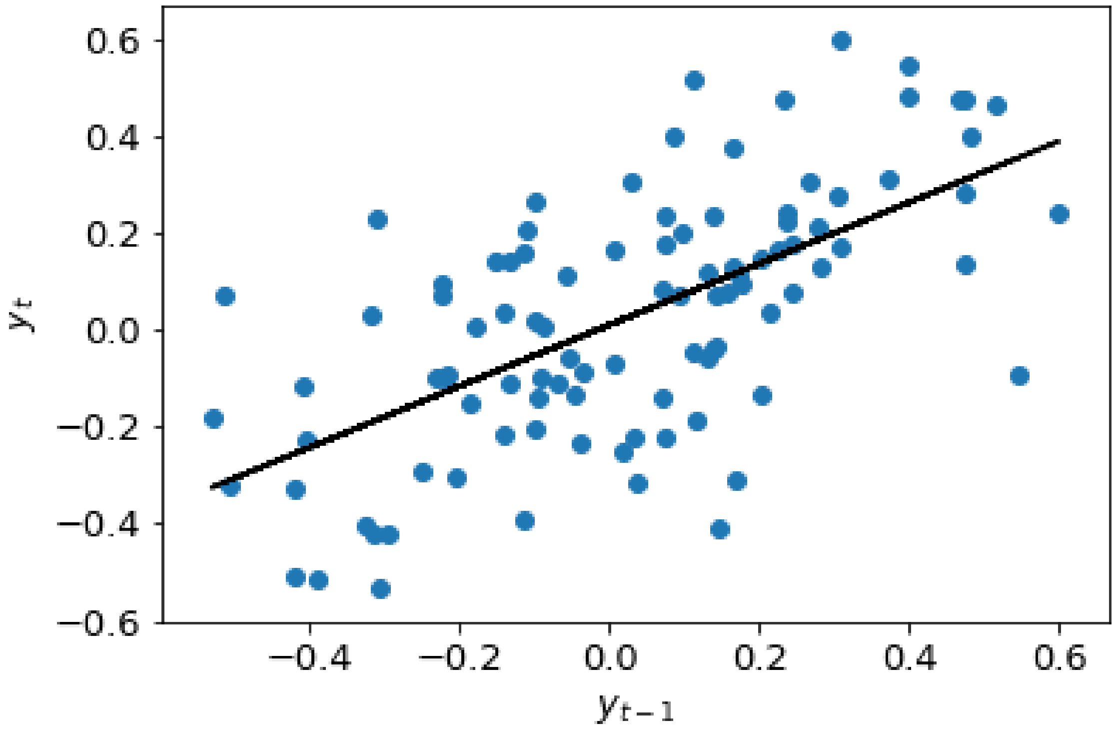
DataCamp

AR models

Autoregressive (AR) model

AR(1) model :

$$y_t = a_1 y_{t-1} + \epsilon_t$$



AR models

Autoregressive (AR) model

AR(1) model :

$$y_t = a_1 y_{t-1} + \epsilon_t$$

AR(2) model :

$$y_t = a_1 y_{t-1} + a_2 y_{t-2} + \epsilon_t$$

AR(p) model :

$$y_t = a_1 y_{t-1} + a_2 y_{t-2} + ... + a_p y_{t-p} + \epsilon_t$$

MA models

Moving average (MA) model

MA(1) model :

$$y_t = m_1 \epsilon_{t-1} + \epsilon_t$$ a shock term

MA(2) model :

$$y_t = m_1 \epsilon_{t-1} + m_2 \epsilon_{t-2} + \epsilon_t$$

MA(q) model :

$$y_t = m_1 \epsilon_{t-1} + m_2 \epsilon_{t-2} + ... + m_q \epsilon_{t-q} + \epsilon_t$$

ARMA models

Autoregressive moving-average (ARMA) model

- ARMA = AR + MA

ARMA(1,1) model :

$$y_t = a_1 y_{t-1} + m_1 \epsilon_{t-1} + \epsilon_t$$

ARMA(p, q)

- p is order of AR part
- q is order of MA part

The p tells us the order of the autoregressive part of the model and the q tells us the order of the moving-average part.

Creating ARMA data

$$y_t = a_1 y_{t-1} + m_1 \epsilon_{t-1} + \epsilon_t$$

Creating ARMA data

$$y_t = 0.5y_{t-1} + 0.2\epsilon_{t-1} + \epsilon_t$$

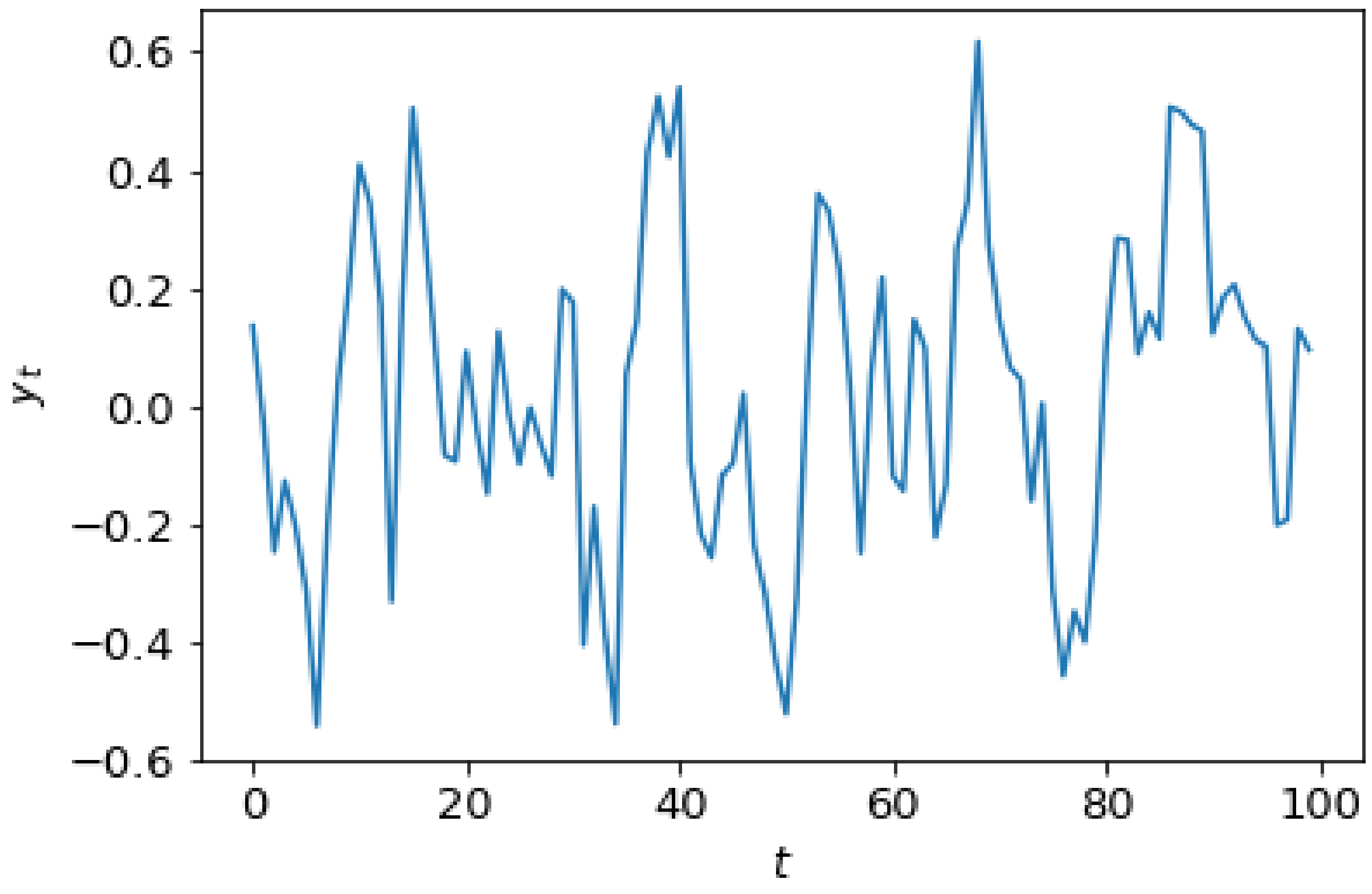
```
from statsmodels.tsa.arima_process import arma_generate_sample

ar_coefs = [1, -0.5]
ma_coefs = [1, 0.2]

y = arma_generate_sample(ar_coefs, ma_coefs, nsample=100, sigma=0.5)
```

You can use the arma_generate_sample() function available in your workspace to generate time series using different AR and MA coefficients.

Remember for any model ARMA(p,q):
The list ar_coefs has the form [1, -a_1, -a_2, ..., -a_p].
The list ma_coefs has the form [1, m_1, m_2, ..., m_q],
where a_i are the lag-i AR coefficients and m_j are the lag-j MA coefficients.

Creating ARMA data

$$y_t = 0.5y_{t-1} + 0.2\epsilon_{t-1} + \epsilon_t$$



Fitting and ARMA model

```
from statsmodels.tsa.arima_model import ARMA

# Instantiate model object
model = ARMA(y, order=(1,1))

# Fit model
results = model.fit()
```

Let's practice!

ARIMA MODELS IN PYTHON