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

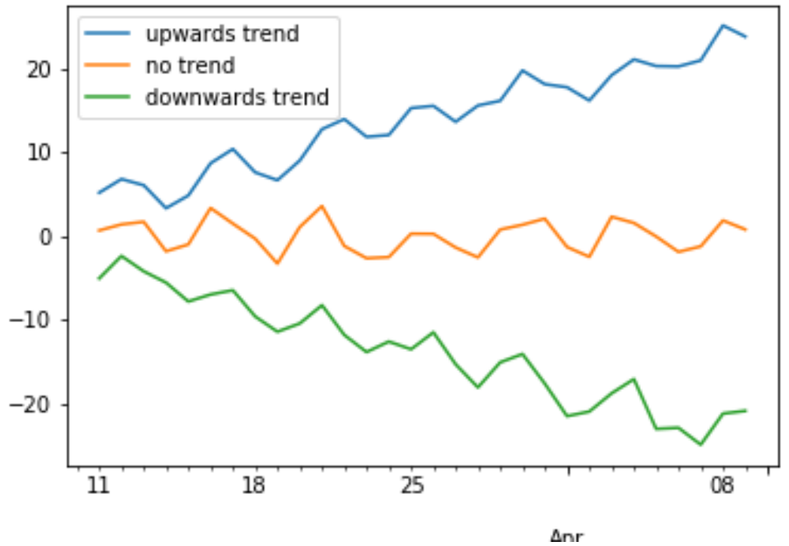
Loading and plotting

```
import pandas as pd
import matplotlib as plt

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

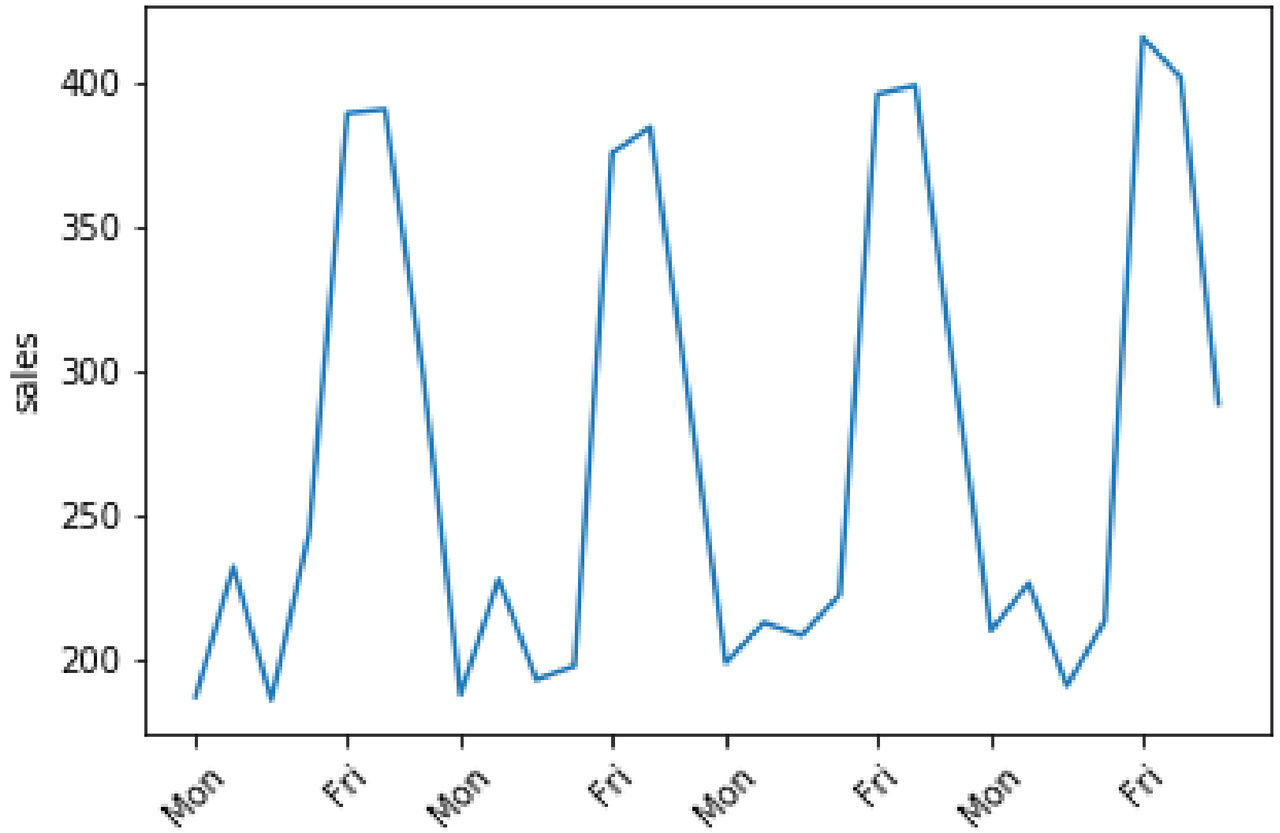
```
date              values
2019-03-11    5.734193
2019-03-12    6.288708
2019-03-13    5.205788
2019-03-14    3.176578
```

Trend

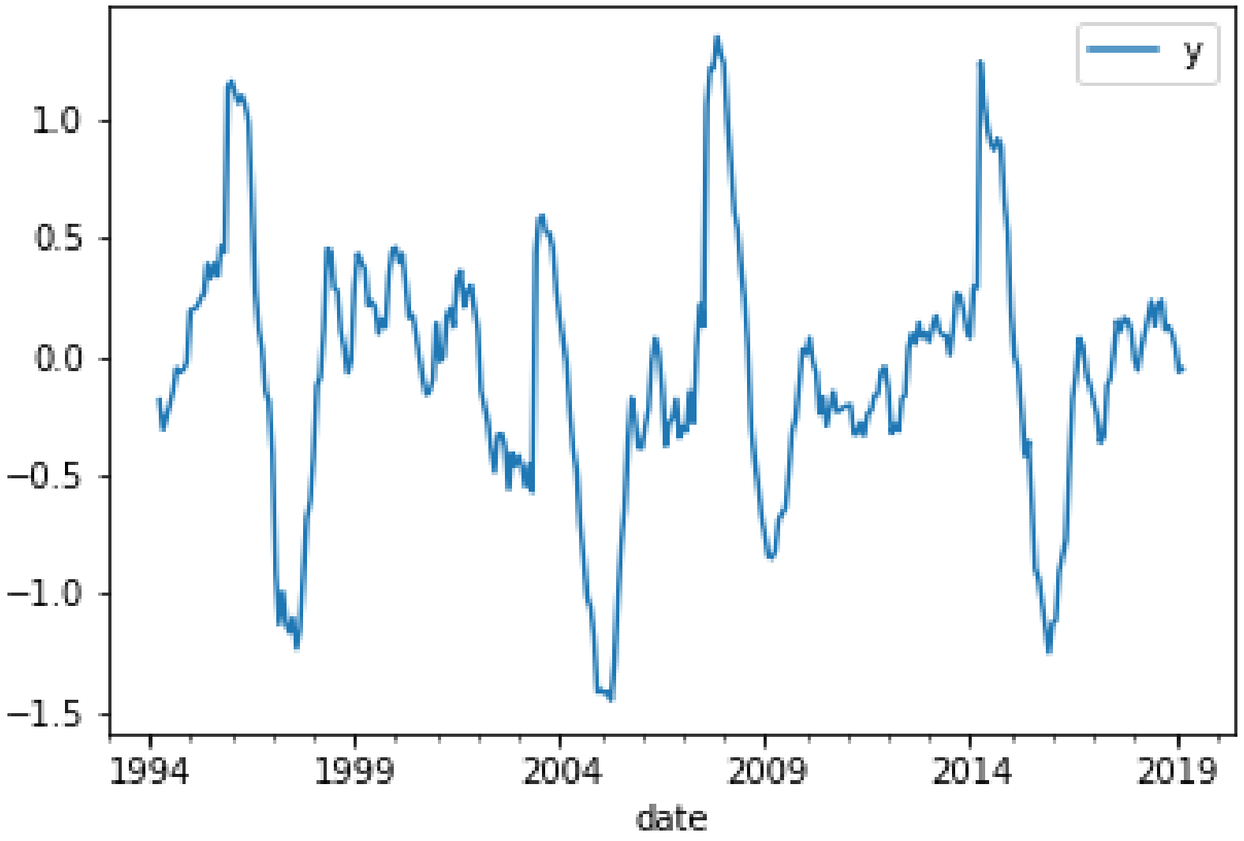
```
fig, ax = plt.subplots()
df.plot(ax=ax)
plt.show()
```



Seasonality



Cyclicality



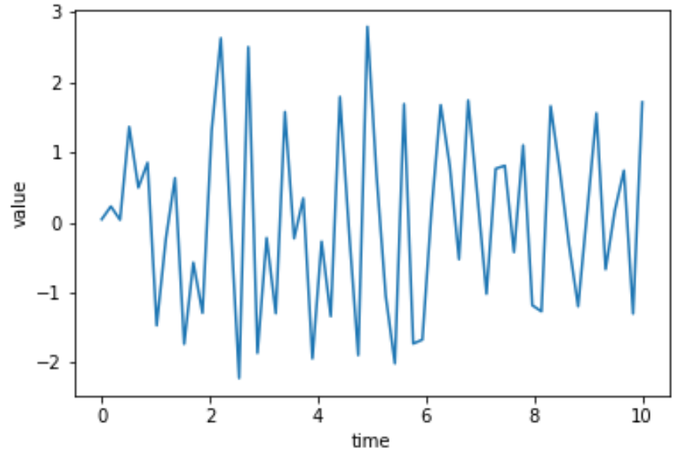
White noise

White noise series has uncorrelated values

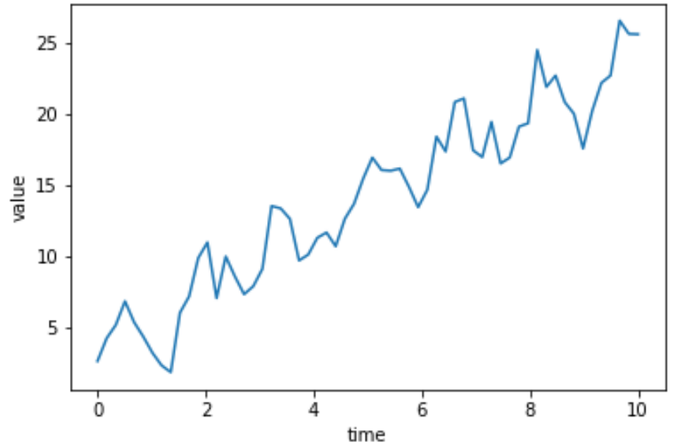
- Heads, heads, heads, tails, heads, tails, ...
- 0.1, -0.3, 0.8, 0.4, -0.5, 0.9, ...

Stationarity

Stationary



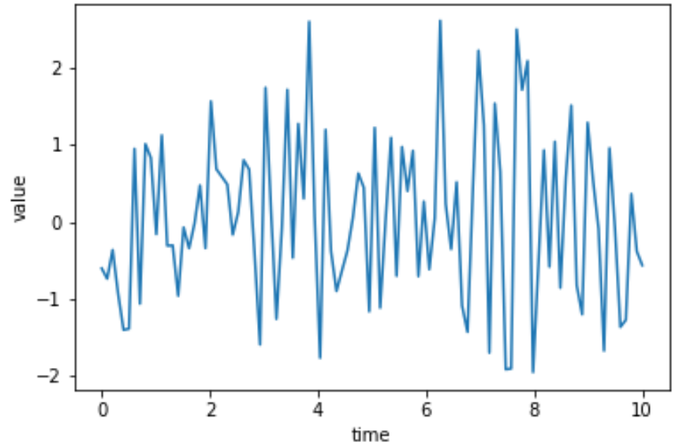
Not stationary



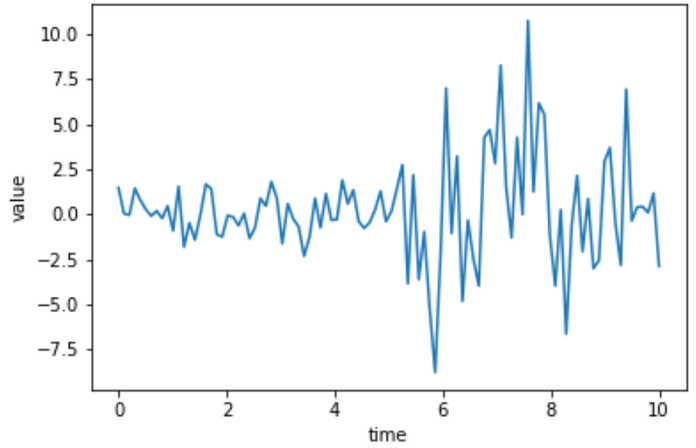
- Trend stationary: Trend is zero

Stationarity

Stationary



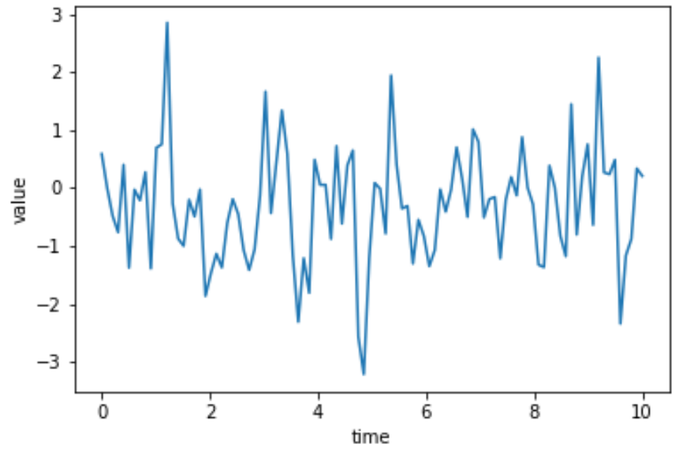
Not stationary



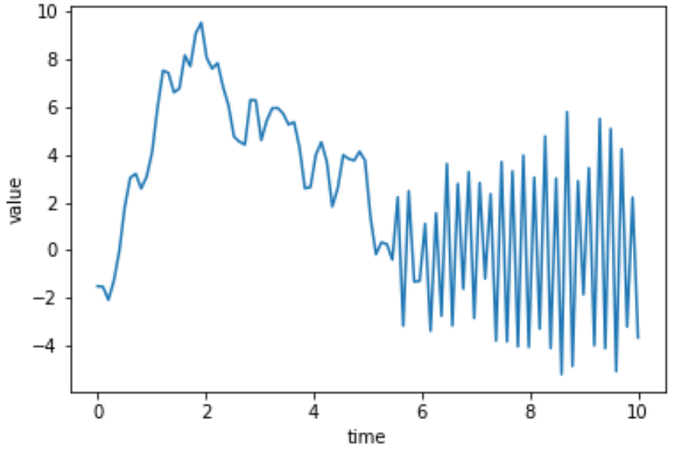
- Trend stationary: Trend is zero
- Variance is constant

Stationarity

Stationary



Not stationary



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

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

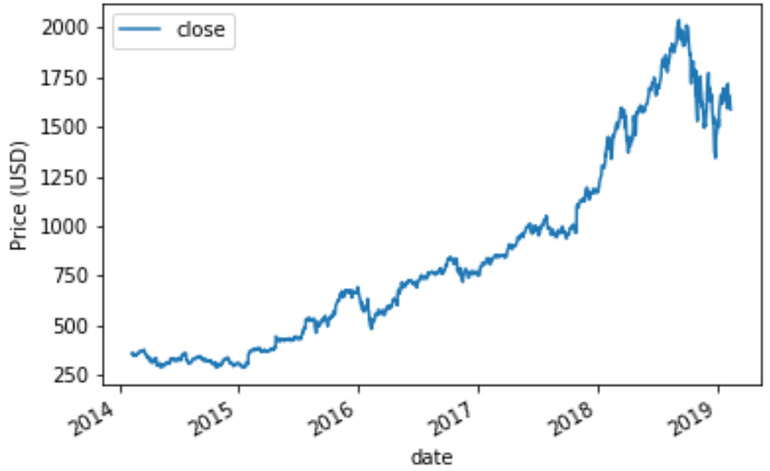
Overview

- 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




```
from statsmodels.tsa.stattools import adfuller


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

Interpreting the test result

```
print(results)
```

```
(-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 $\rightarrow$ 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)
```

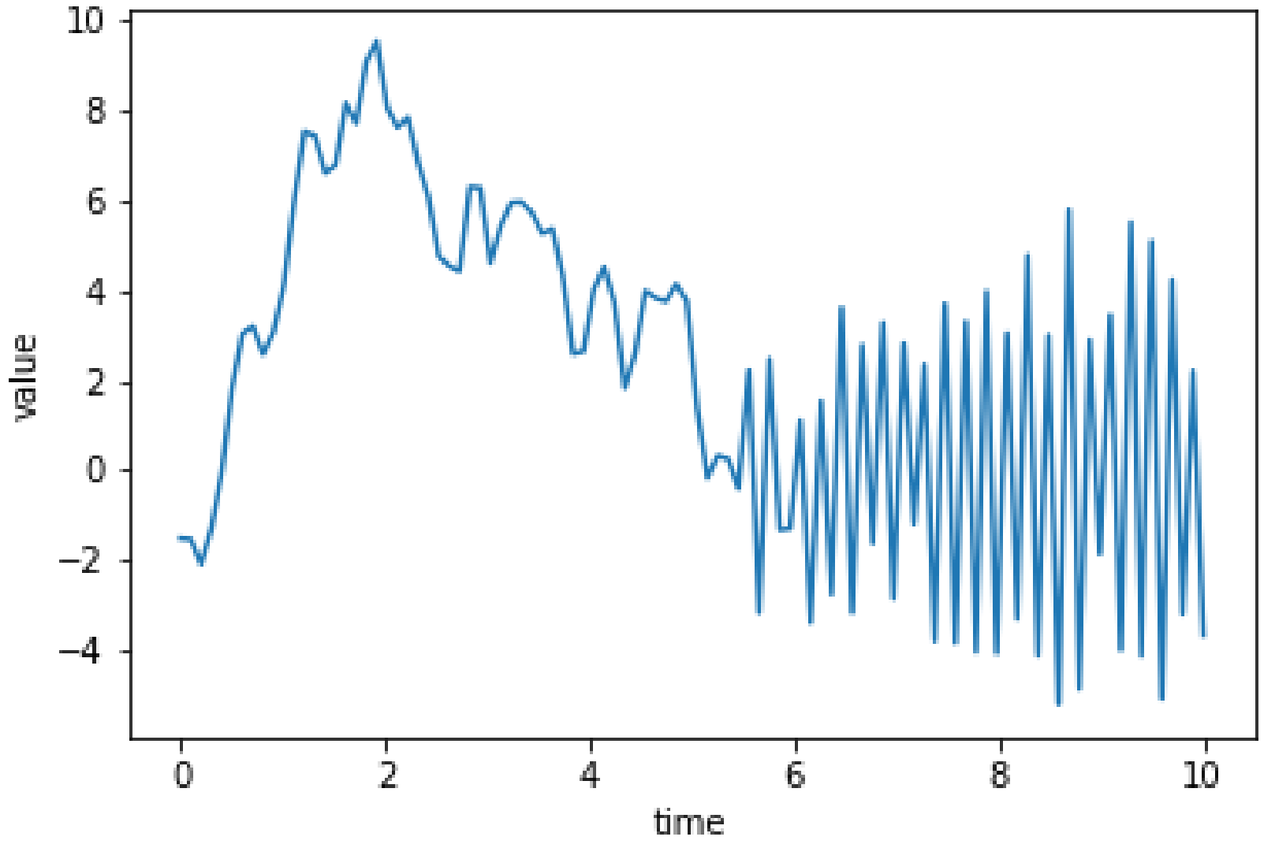
- 0th element is test statistic (-1.34)
  - More negative means more likely to be stationary
- 1st element is p-value: (0.60)
  - If p-value is small $\rightarrow$ reject null hypothesis. Reject non-stationary.
- 4th element is the critical test statistics

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

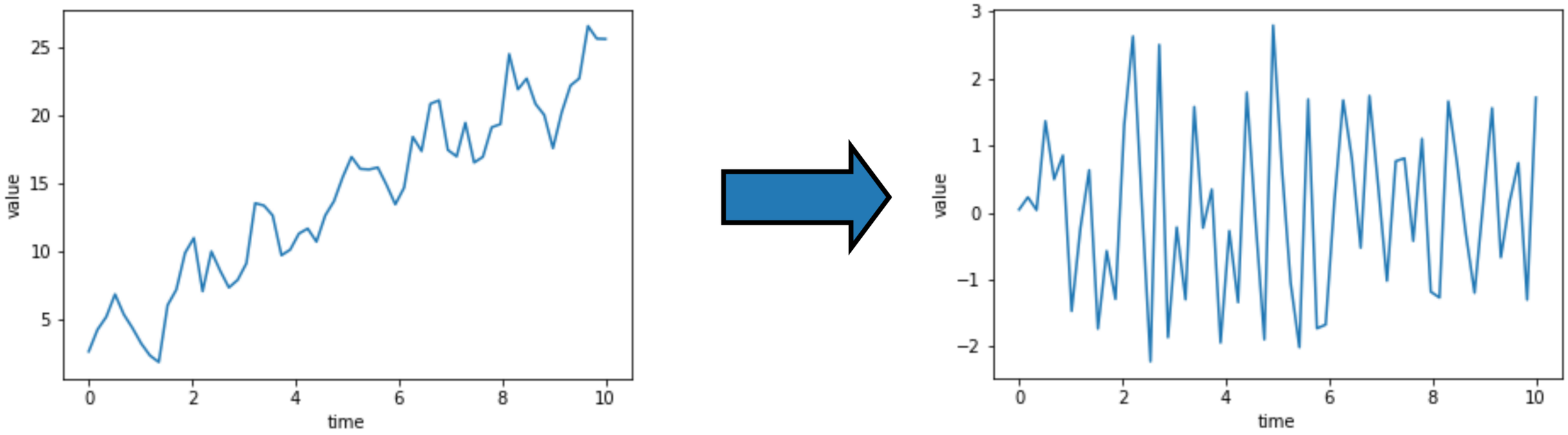
The value of plotting

- Plotting time series can stop you making wrong assumptions

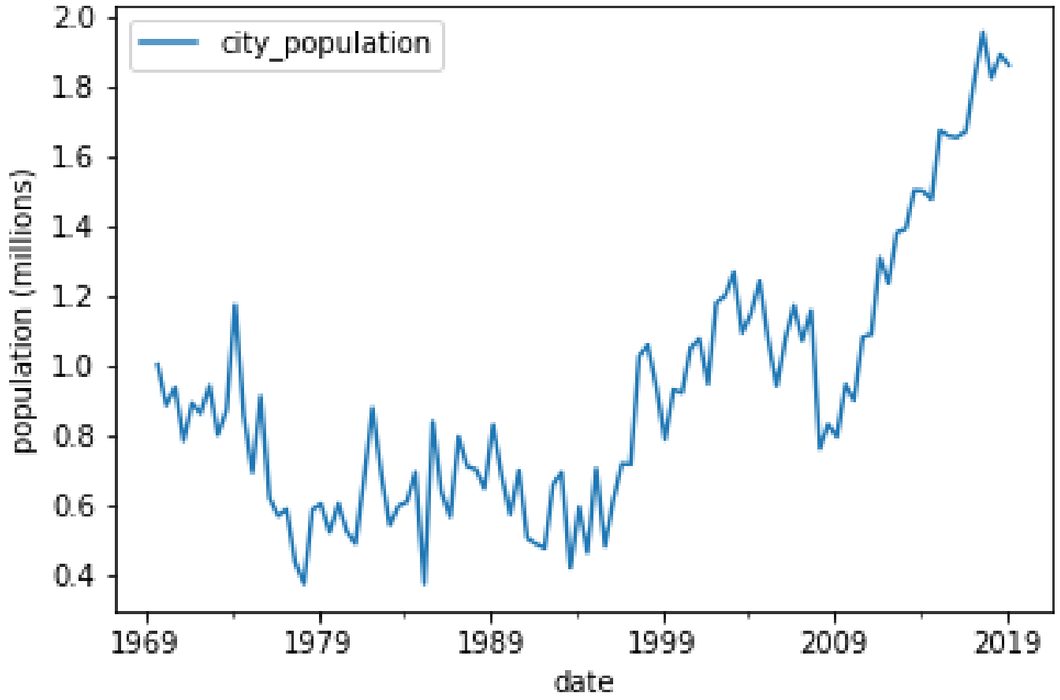
The value of plotting



Making a time series stationary



Taking the difference



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

Taking the difference

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

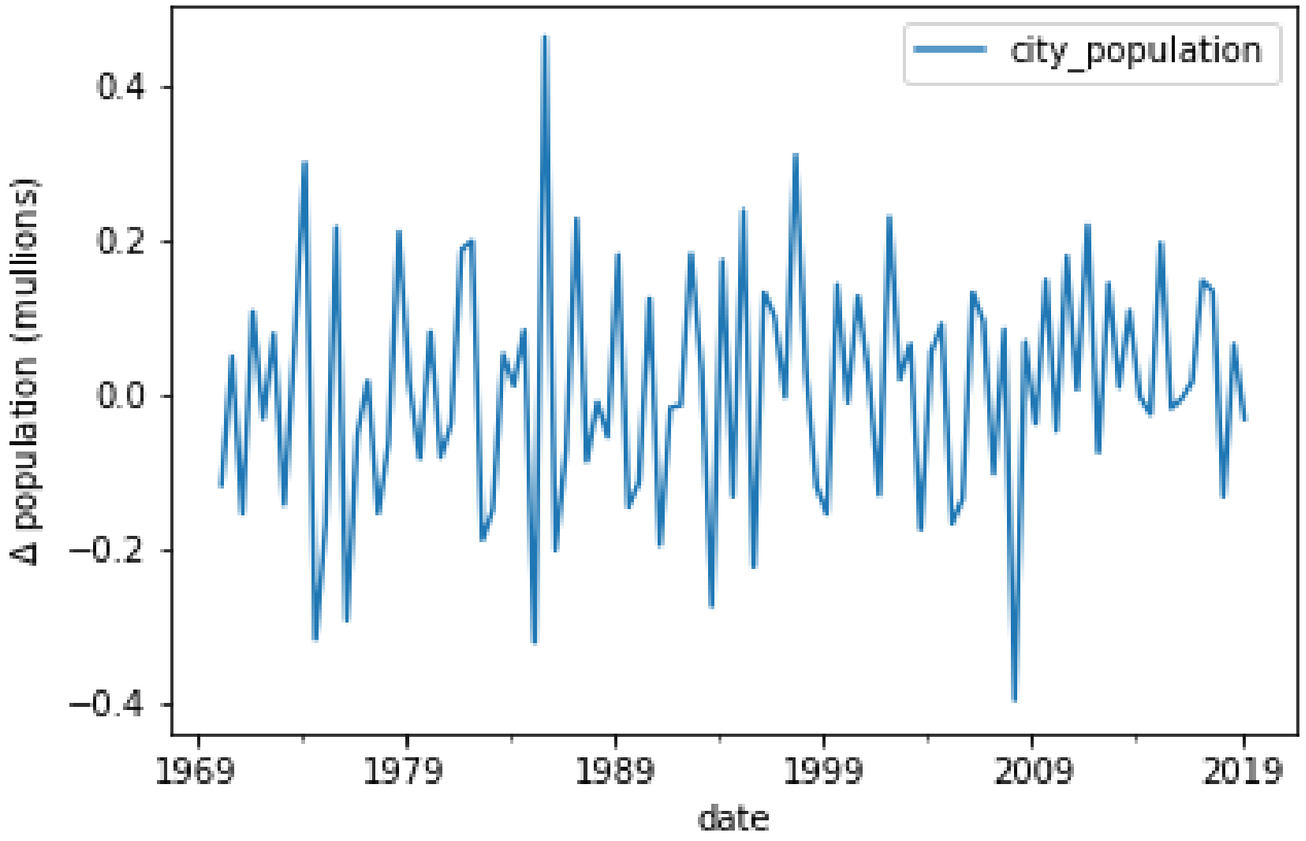
```
            city_population
date
1969-09-30              NaN
1970-03-31        -0.116156
1970-09-30         0.050850
1971-03-31        -0.153261
1971-09-30         0.108389
```

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

Intro to AR, MA and ARMA models

ARIMA MODELS IN PYTHON

James Fulton
Climate informatics researcher

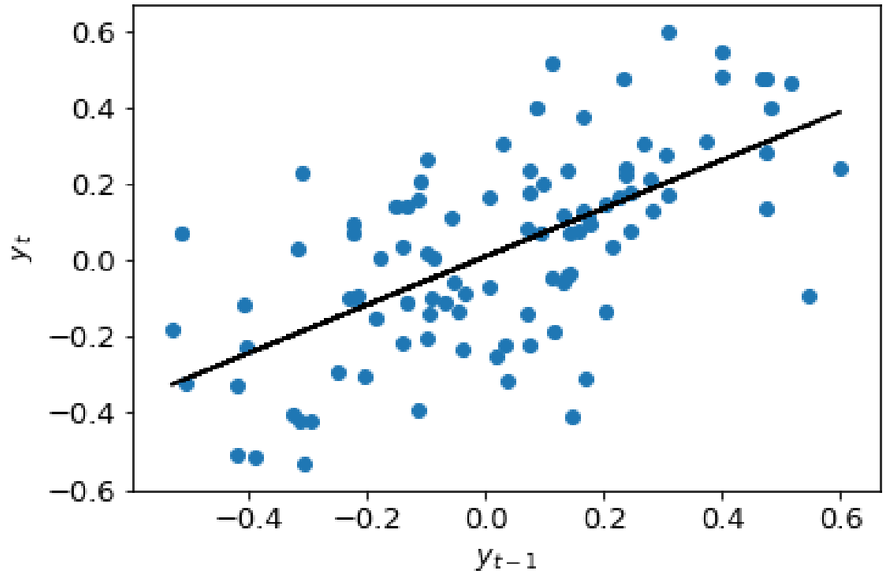
datacamp

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 :

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

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

Creating ARMA data

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

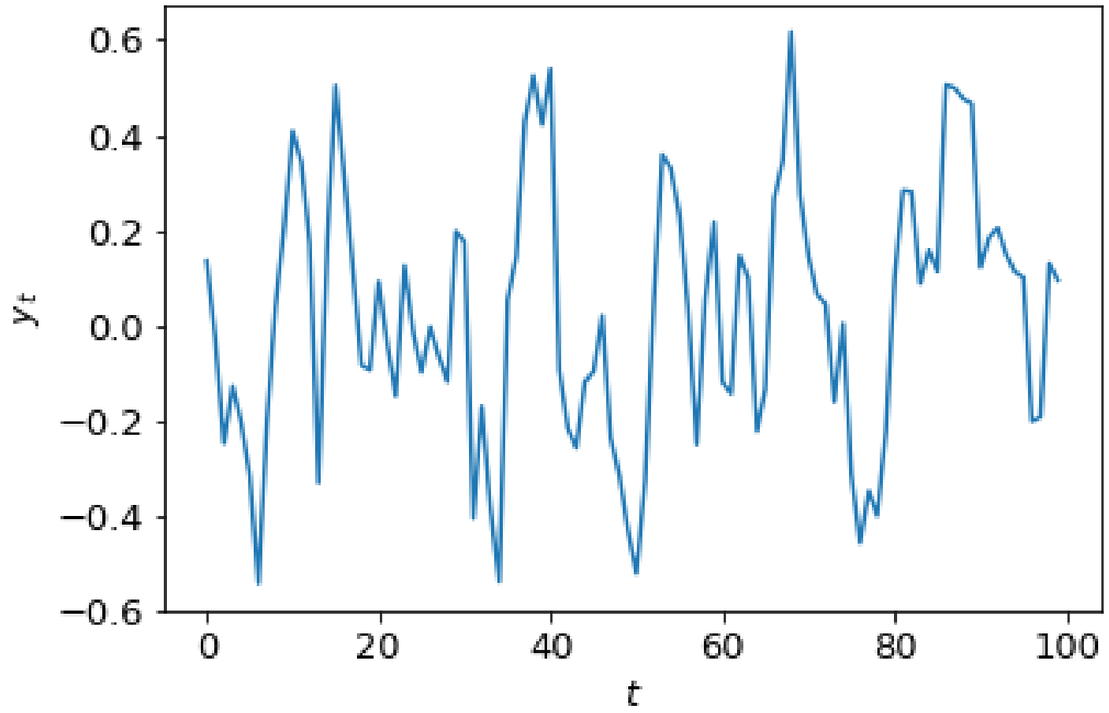
Creating ARMA data

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

```
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)
```

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

Fitting time series models

ARIMA MODELS IN PYTHON

James Fulton
Climate informatics researcher

datacamp

Creating a model

```
from statsmodels.tsa.arima_model import ARMA
```

```
model = ARMA(timeseries, order=(p,q))
```

Creating AR and MA models

```
ar_model = ARMA(timeseries, order=(p,0))
```

```
ma_model = ARMA(timeseries, order=(0,q))
```

Fitting the model and fit summary

```
model = ARMA(timeseries, order=(2,1))
results = model.fit()
```

```
print(results.summary())
```

Fit summary

```
                              ARMA Model Results
==============================================================================
Dep. Variable:                      y   No. Observations:                 1000
Model:                     ARMA(2, 1)   Log Likelihood                 148.580
Method:                       css-mle   S.D. of innovations              0.208
Date:                Thu, 25 Apr 2019   AIC                           -287.159
Time:                        22:57:00   BIC                           -262.621
Sample:                             0   HQIC                          -277.833

==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const         -0.0017      0.012     -0.147      0.883      -0.025       0.021
ar.L1.y        0.5253      0.054      9.807      0.000       0.420       0.630
ar.L2.y       -0.2909      0.042     -6.850      0.000      -0.374      -0.208
ma.L1.y        0.3679      0.052      7.100      0.000       0.266       0.469
                                    Roots
=============================================================================
                  Real          Imaginary           Modulus         Frequency
-----------------------------------------------------------------------------
AR.1            0.9029           -1.6194j            1.8541           -0.1690
AR.2            0.9029           +1.6194j            1.8541            0.1690
MA.1           -2.7184           +0.0000j            2.7184            0.5000
-----------------------------------------------------------------------------
```

Fit summary

```
                              ARMA Model Results
==============================================================================
Dep. Variable:                      y   No. Observations:                 1000
Model:                     ARMA(2, 1)   Log Likelihood                 148.580
Method:                       css-mle   S.D. of innovations              0.208
Date:                Thu, 25 Apr 2019   AIC                           -287.159
Time:                        22:57:00   BIC                           -262.621
Sample:                             0   HQIC                          -277.833
```

Fit summary

```
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const         -0.0017      0.012     -0.147      0.883      -0.025       0.021
ar.L1.y        0.5253      0.054      9.807      0.000       0.420       0.630
ar.L2.y       -0.2909      0.042     -6.850      0.000      -0.374      -0.208
ma.L1.y        0.3679      0.052      7.100      0.000       0.266       0.469
```

Introduction to ARMAX models

- Exogenous ARMA
- Use external variables as well as time series
- ARMAX = ARMA + linear regression

ARMAX equation

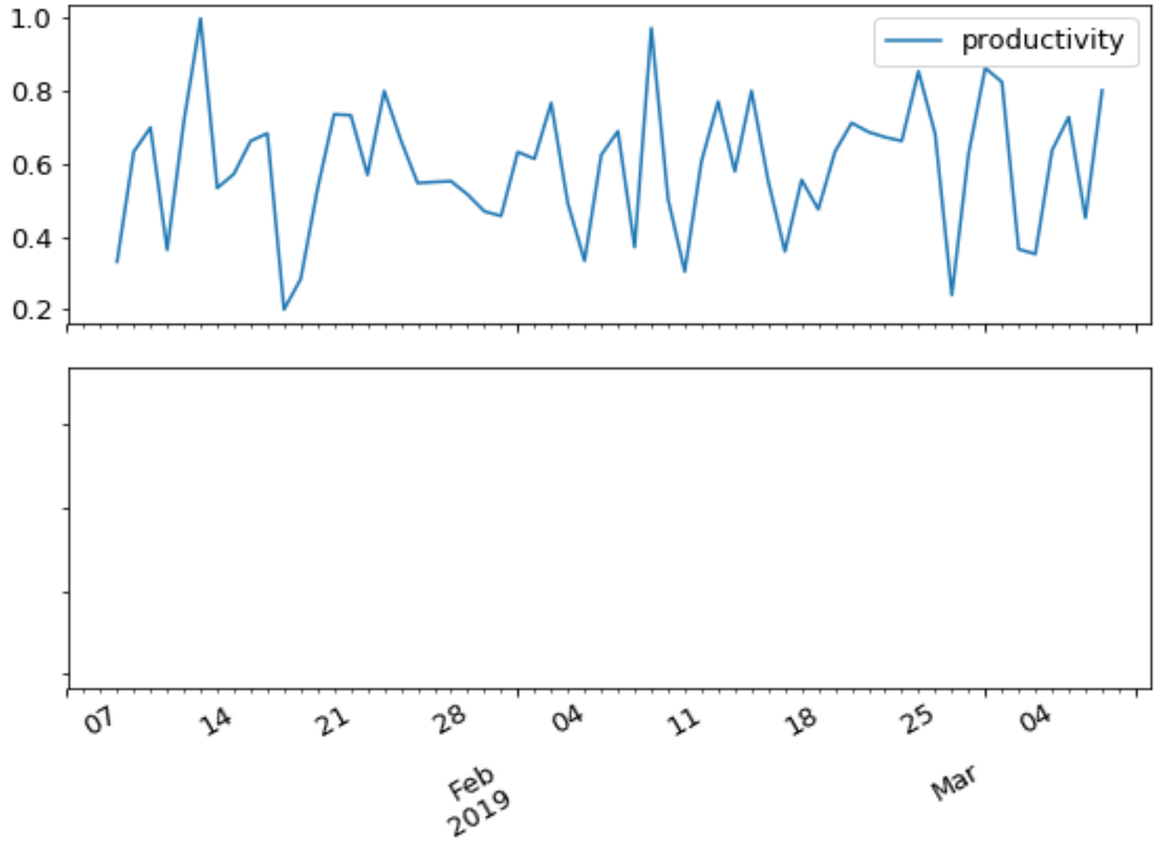
ARMA(1,1) model :

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

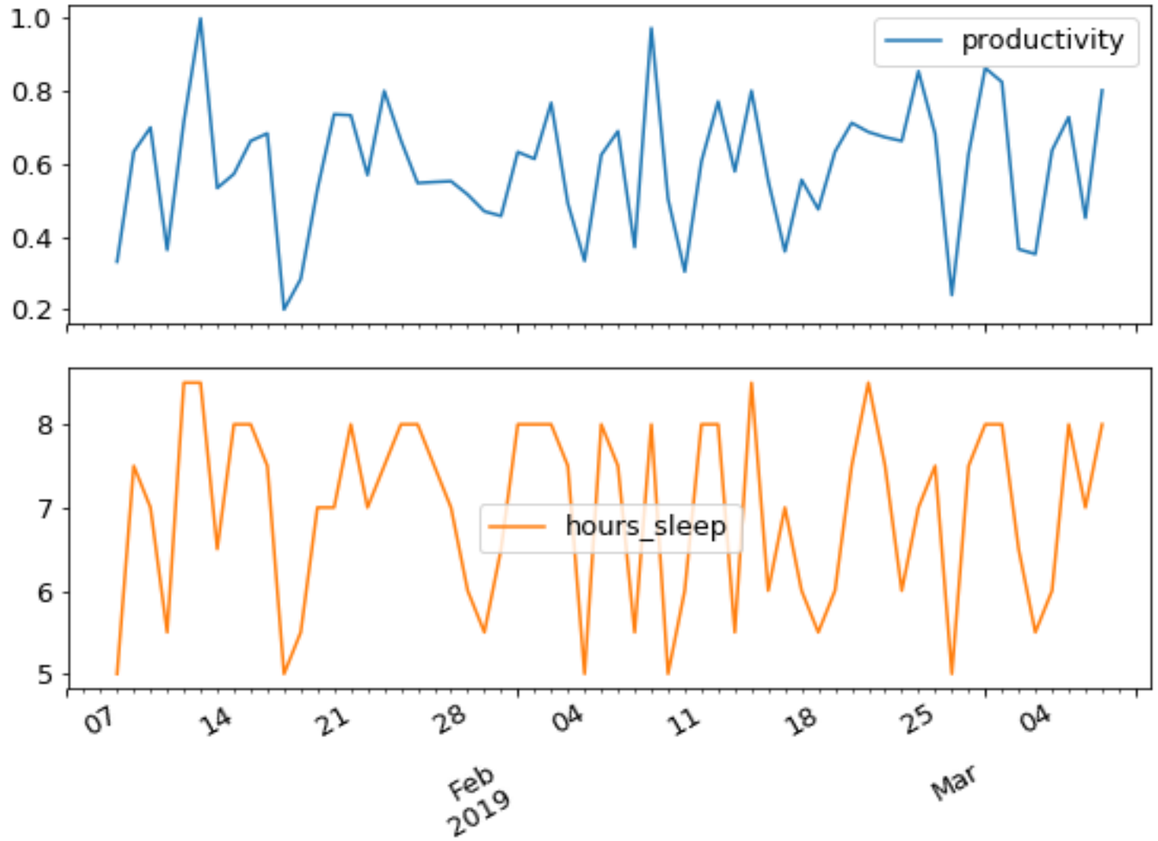
ARMAX(1,1) model :

$$y_t = x_1 z_t + a_1 y_{t-1} + m_1 \epsilon_{t-1} + \epsilon_t$$

ARMAX example



ARMAX example



Fitting ARMAX

```
# Instantiate the model
model = ARMA(df['productivity'], order=(2,1), exog=df['hours_sleep'])


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

ARMAX summary

```
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const         -0.1936      0.092     -2.098      0.041      -0.375      -0.013
x1             0.1131      0.013      8.602      0.000       0.087       0.139
ar.L1.y        0.1917      0.252      0.760      0.450      -0.302       0.686
ar.L2.y       -0.3740      0.121     -3.079      0.003      -0.612      -0.136
ma.L1.y       -0.0740      0.259     -0.286      0.776      -0.581       0.433
```

Let's practice!

ARIMA MODELS IN PYTHON

Forecasting

ARIMA MODELS IN PYTHON

James Fulton
Climate informatics researcher

datacamp

Predicting the next value

Take an AR(1) model

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

Predict next value

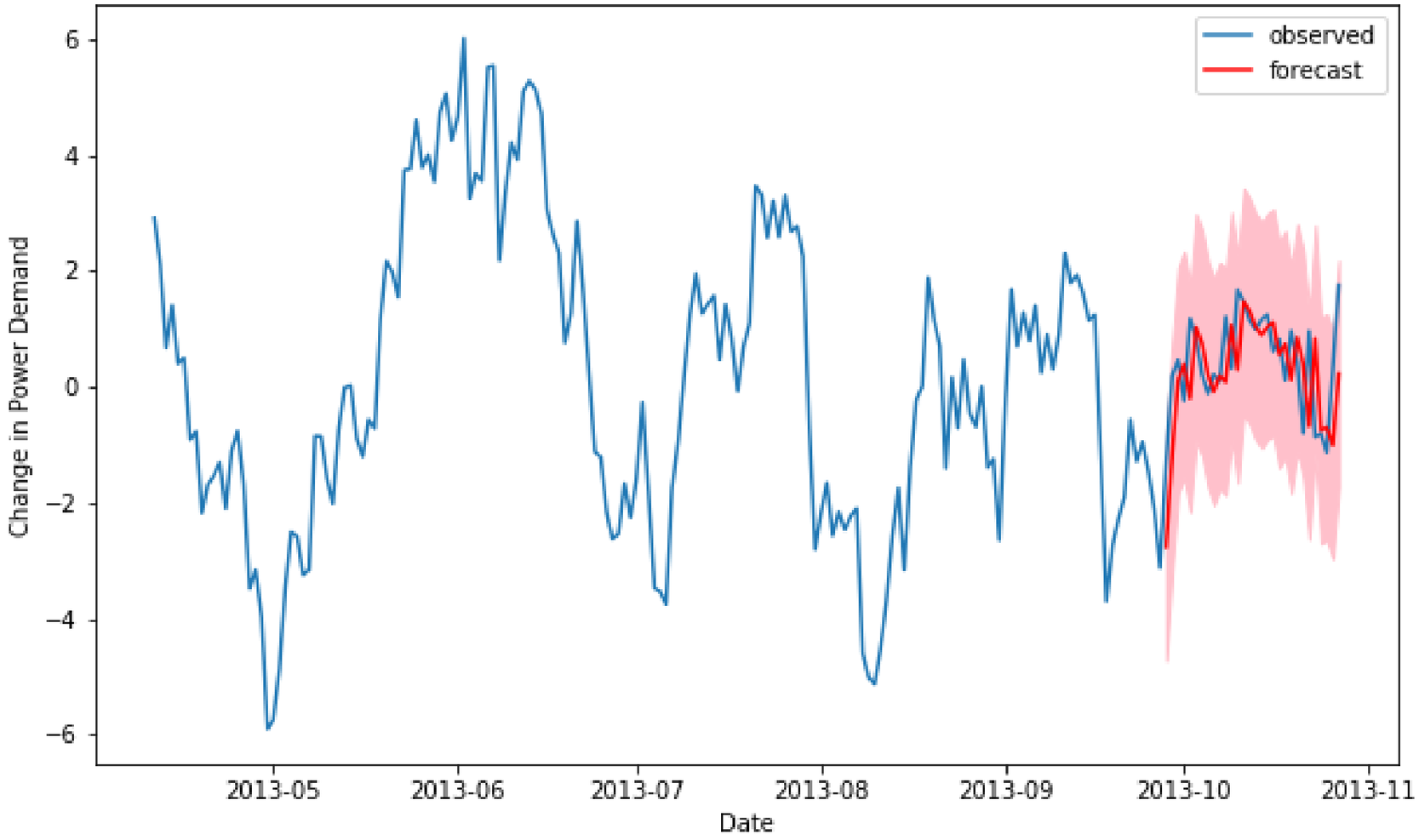
$$y_t = 0.6 \text{ x } 10 + \epsilon_t$$

$$y_t = 6.0 + \epsilon_t$$

Uncertainty on prediction

$$5.0 < y_t < 7.0$$

One-step-ahead predictions



Statsmodels SARIMAX class

```
from statsmodels.tsa.statespace.sarimax import SARIMAX


# Just an ARMA(p,q) model
model = SARIMAX(df, order=(p,0,q))
```

Statsmodels SARIMAX class

```
from statsmodels.tsa.statespace.sarimax import SARIMAX


# An ARMA(p,q) + constant model
model = SARIMAX(df, order=(p,0,q), trend='c')
```

Making one-step-ahead predictions

```
# Make predictions for last 25 values
results = model.fit()
# Make in-sample prediction
forecast = results.get_prediction(start=-25)
```

Making one-step-ahead predictions

```
# Make predictions for last 25 values
results = model.fit()
# Make in-sample prediction
forecast = results.get_prediction(start=-25)
# forecast mean
mean_forecast = forecast.predicted_mean
```

Predicted mean is a pandas series

```
2013-10-28    1.519368
2013-10-29    1.351082
2013-10-30    1.218016
```

Confidence intervals

```
# Get confidence intervals of forecasts
confidence_intervals = forecast.conf_int()
```

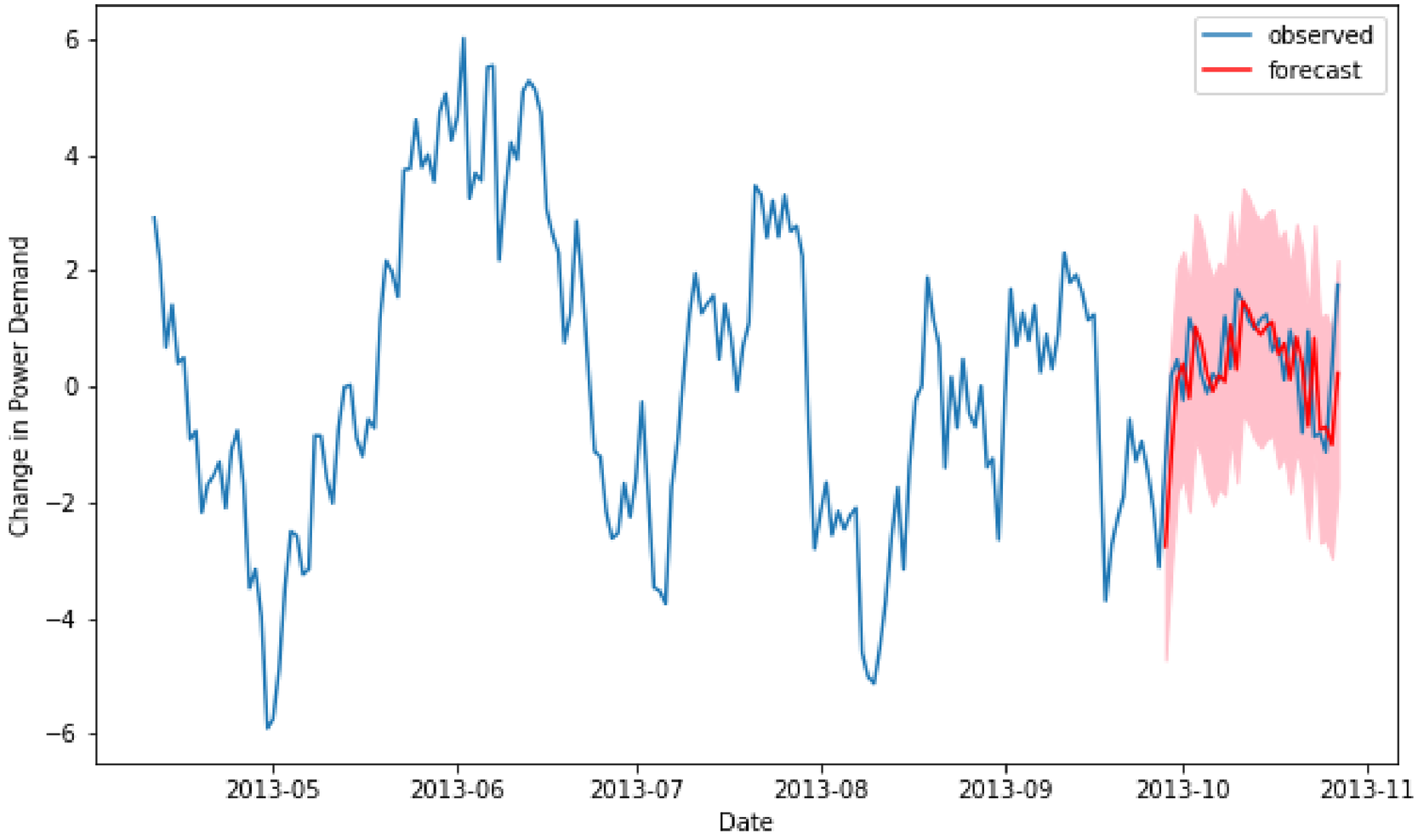
Confidence interval method returns `pandas` DataFrame

```
                            lower y                       upper y
2013-09-28                -4.720471                     -0.815384
2013-09-29                -5.069875                      0.112505
2013-09-30                -5.232837                      0.766300
2013-10-01                -5.305814                      1.282935
2013-10-02                -5.326956                      1.703974
```

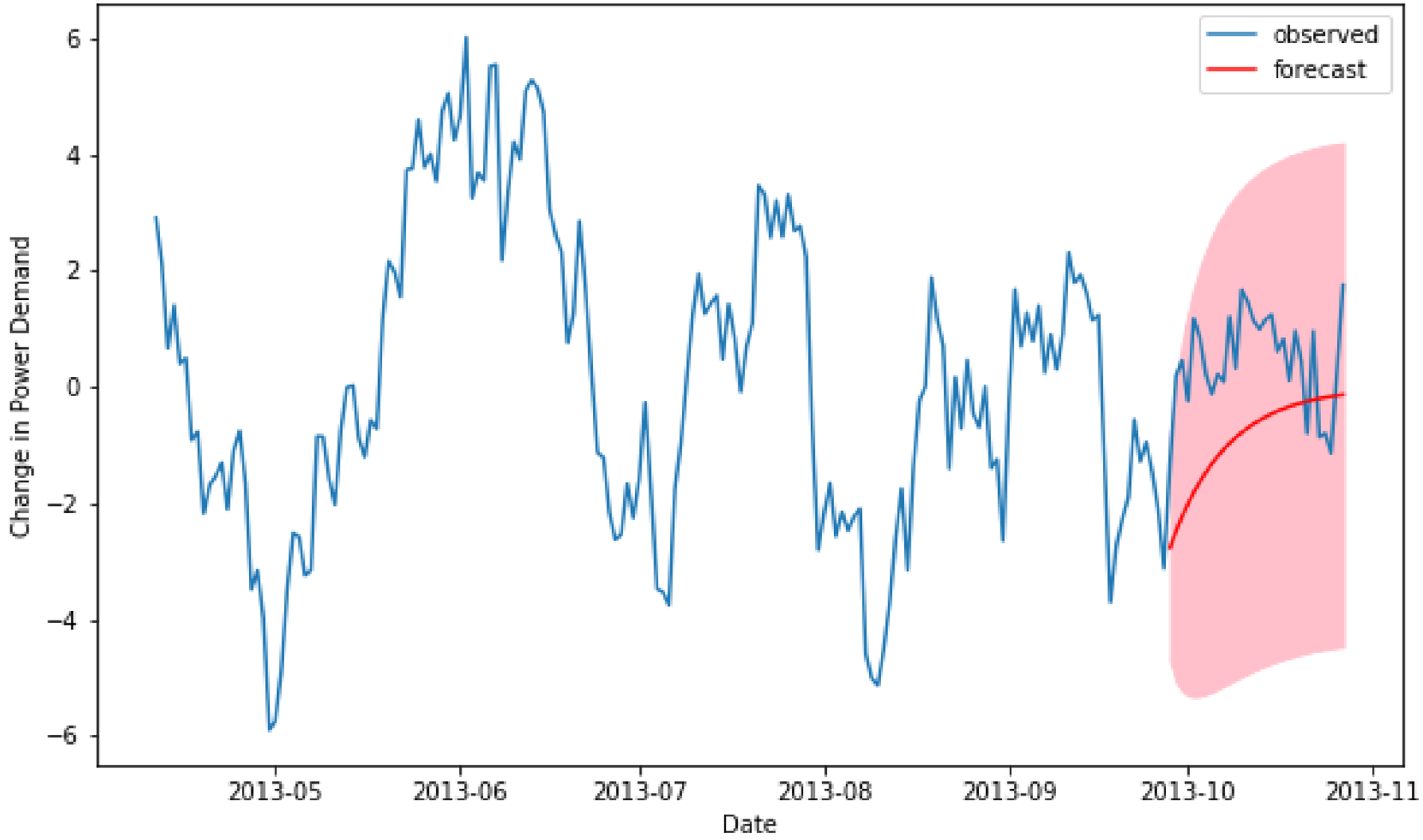
Plotting predictions

```
plt.figure()

# Plot prediction
plt.plot(dates,
         mean_forecast.values,
         color='red',
         label='forecast')
# Shade uncertainty area
plt.fill_between(dates, lower_limits, upper_limits, color='pink')

plt.show()
```

Plotting predictions



Dynamic predictions



Making dynamic predictions

```
results = model.fit()
forecast = results.get_prediction(start=-25, dynamic=True)
```

```
# forecast mean
mean_forecast = forecast.predicted_mean

# Get confidence intervals of forecasts
confidence_intervals = forecast.conf_int()
```

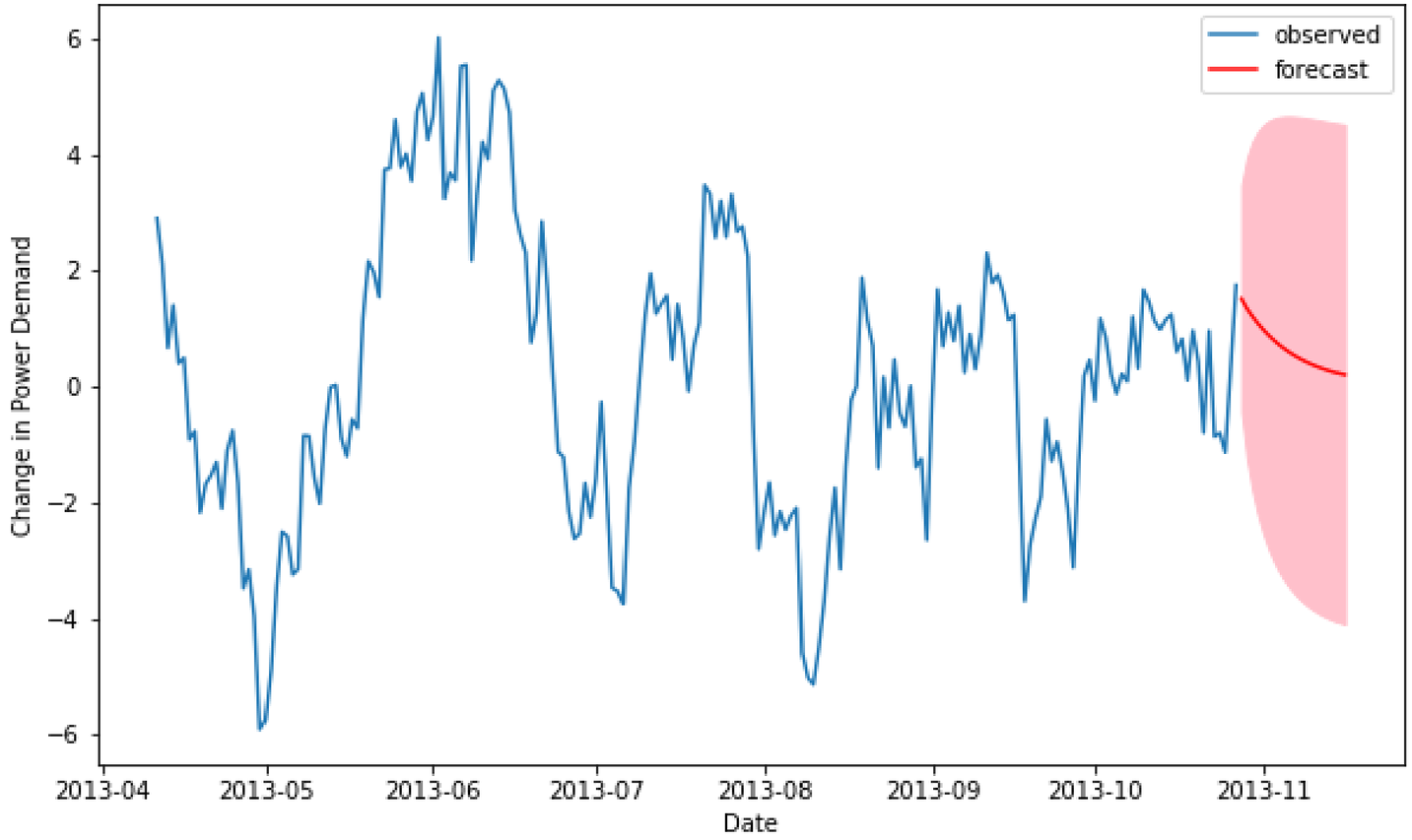
Forecasting out of sample

```
forecast = results.get_forecast(steps=20)
```

```
# forecast mean
mean_forecast = forecast.predicted_mean

# Get confidence intervals of forecasts
confidence_intervals = forecast.conf_int()
```

Forecasting out of sample

```
forecast = results.get_forecast(steps=20)
```



Let's practice!

ARIMA MODELS IN PYTHON

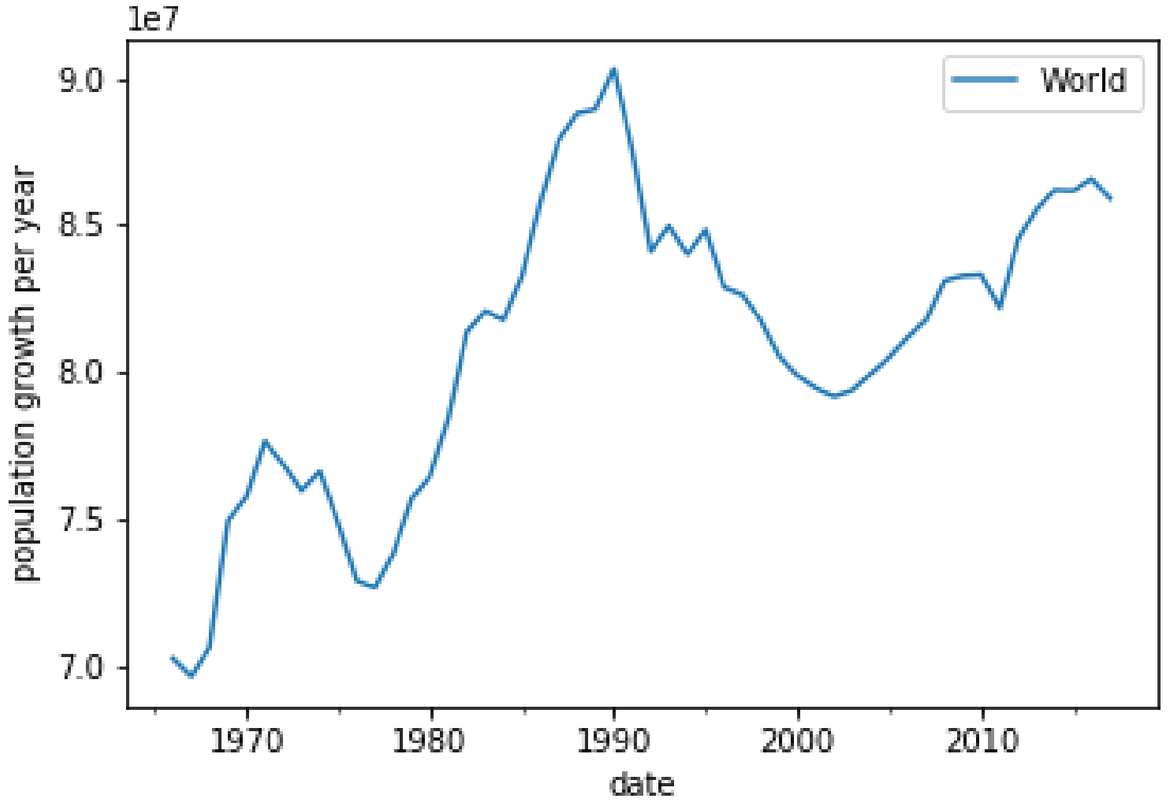
Introduction to ARIMA models

ARIMA MODELS IN PYTHON

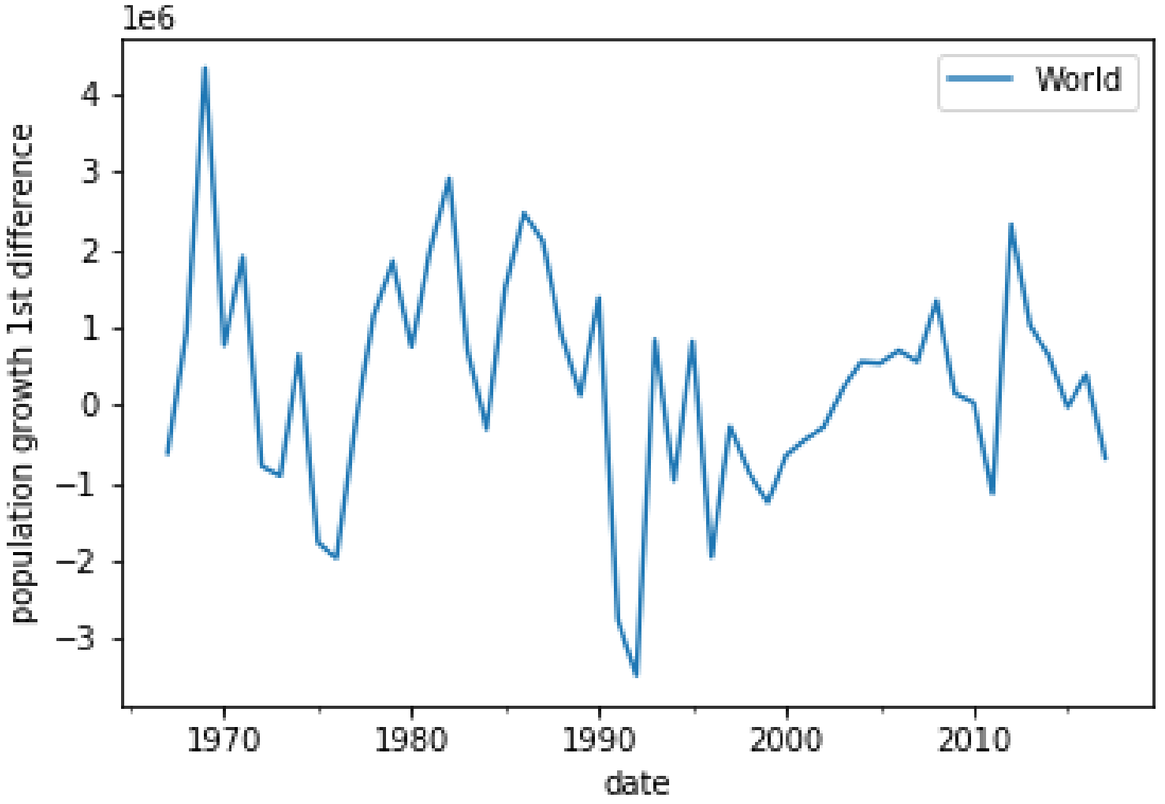
James Fulton
Climate informatics researcher

datacamp

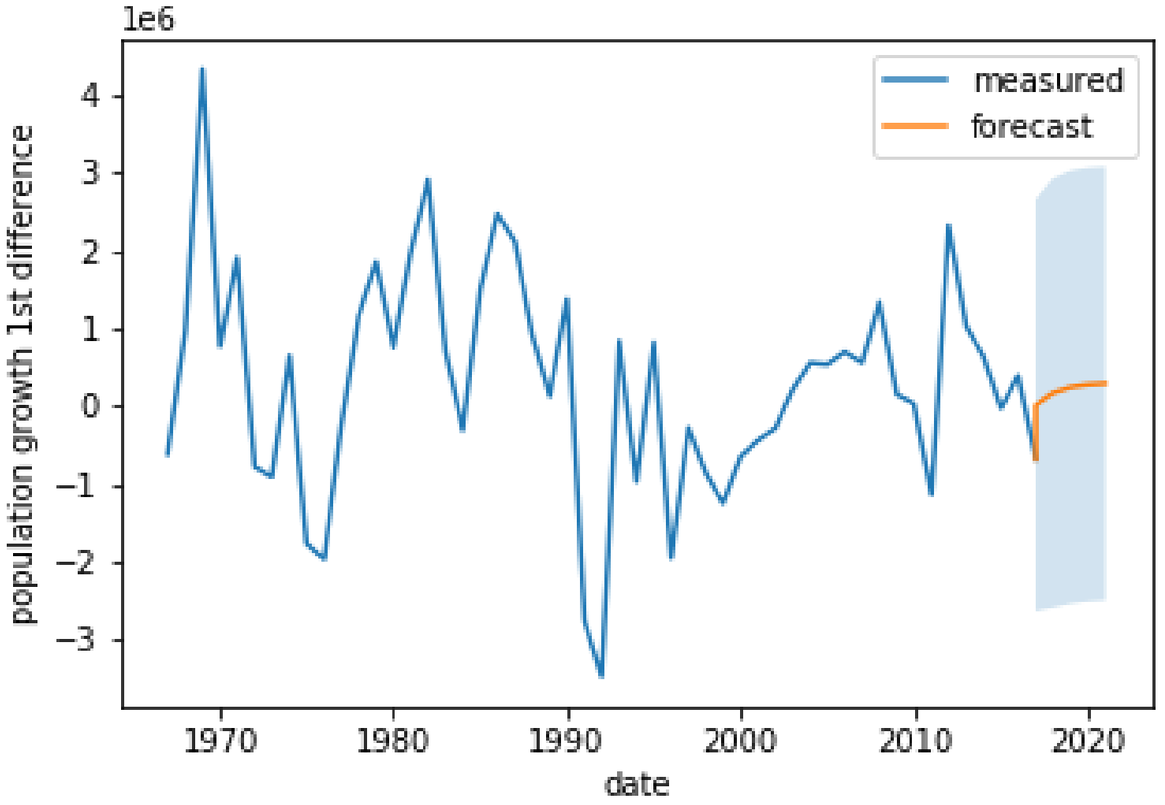
Non-stationary time series recap



Non-stationary time series recap



Forecast of differenced time series



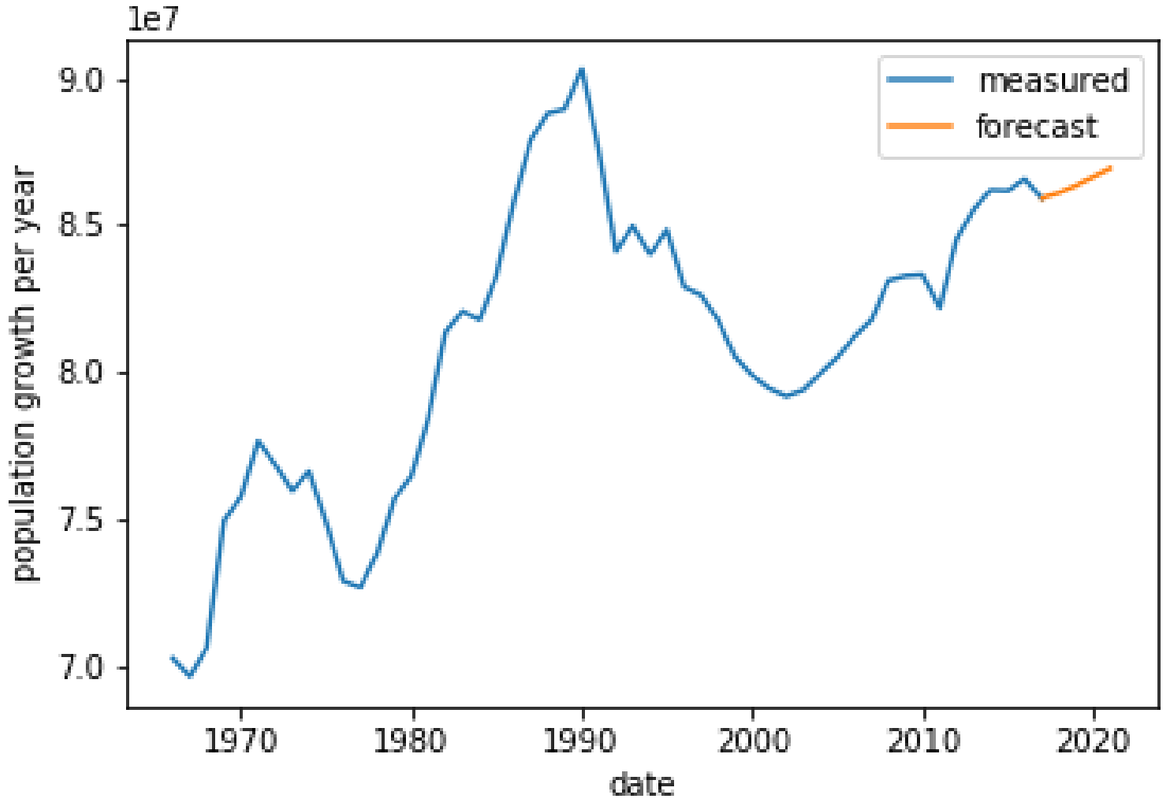
Reconstructing original time series after differencing

```
diff_forecast = results.get_forecast(steps=10).predicted_mean
from numpy import cumsum
mean_forecast = cumsum(diff_forecast)
```

Reconstructing original time series after differencing

```
diff_forecast = results.get_forecast(steps=10).predicted_mean
from numpy import cumsum
mean_forecast = cumsum(diff_forecast) + df.iloc[-1,0]
```

Reconstructing original time series after differencing



The ARIMA model

- Take the difference
- Fit ARMA model
- Integrate forecast

Can we avoid doing so much work?

Yes!

ARIMA - Autoregressive Integrated Moving Average

Using the ARIMA model

```
from statsmodels.tsa.statespace.sarimax import SARIMAX
model = SARIMAX(df, order =(p,d,q))
```

- p - number of autoregressive lags
- d - order of differencing
- q - number of moving average lags

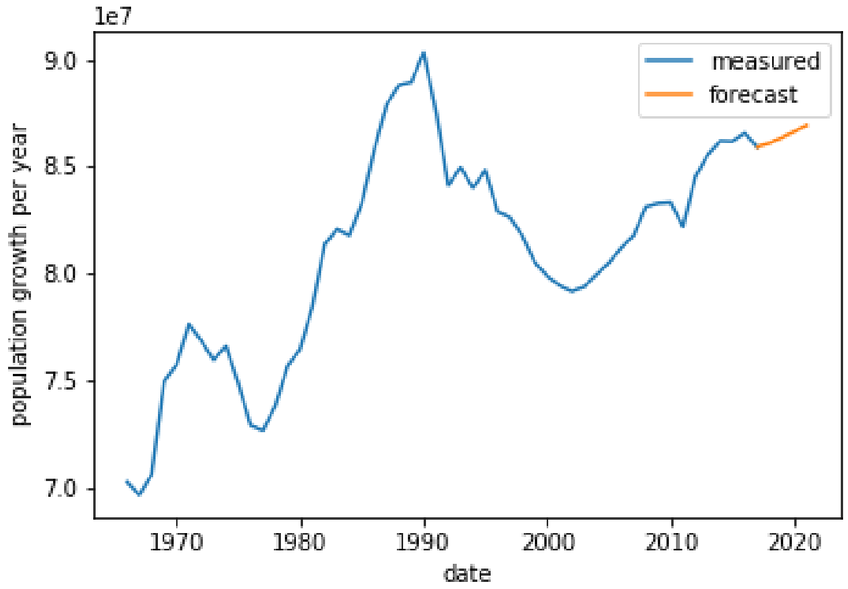
ARMA$(p, 0, q)$ = ARMA(p, q)

Using the ARIMA model

```
# Create model
model = SARIMAX(df, order=(2,1,1))
# Fit model
model.fit()
# Make forecast
mean_forecast = results.get_forecast(steps=10).predicted_mean
```

Using the ARIMA model

```
# Make forecast
mean_forecast = results.get_forecast(steps=steps).predicted_mean
```



Picking the difference order

```
adf = adfuller(df.iloc[:,0])
print('ADF Statistic:', adf[0])
print('p-value:', adf[1])
```

```
ADF Statistic: -2.674
p-value: 0.0784
```

```
adf = adfuller(df.diff().dropna().iloc[:,0])
print('ADF Statistic:', adf[0])
print('p-value:', adf[1])
```

```
ADF Statistic: -4.978
p-value: 2.44e-05
```

Picking the difference order

```
model = SARIMAX(df, order=(p,1,q))
```

Let's practice!

ARIMA MODELS IN PYTHON

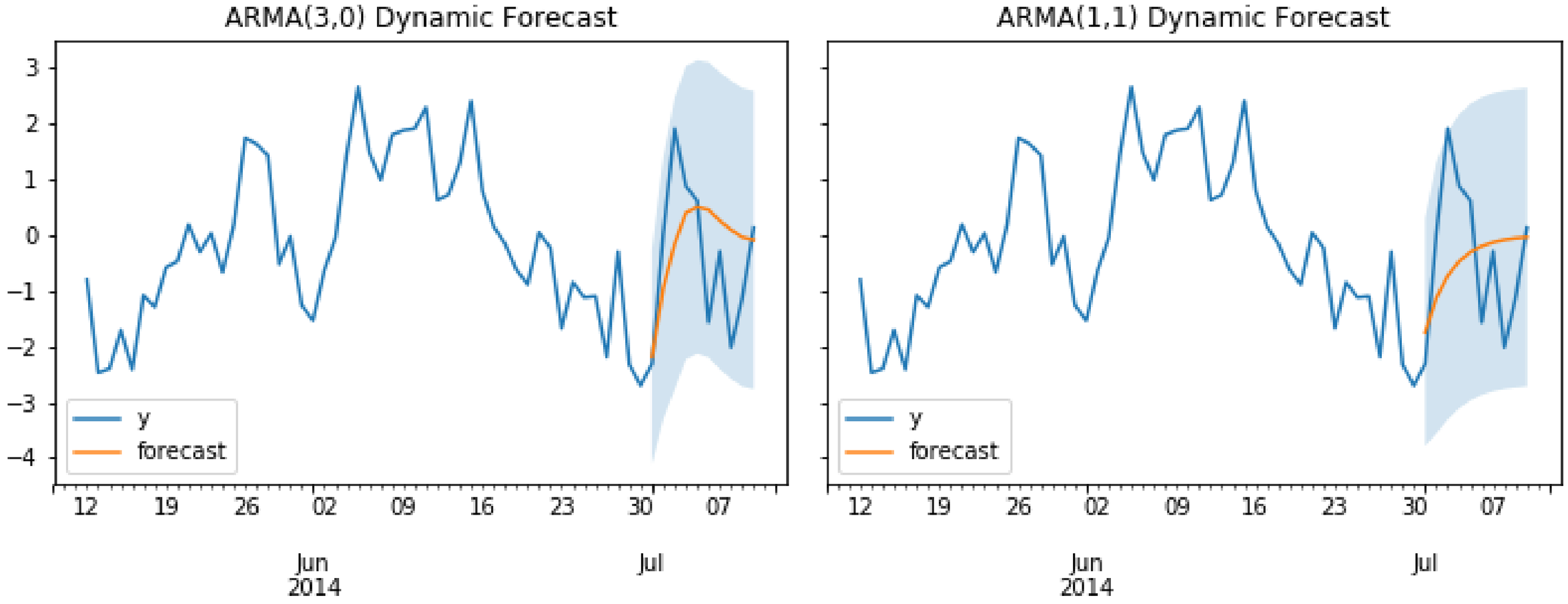
Intro to ACF and PACF

ARIMA MODELS IN PYTHON

James Fulton
Climate informatics researcher

datacamp

Motivation


ARMA(3,0) Dynamic Forecast
3
2
1
0
−1
−2
−3
−4
y
forecast
12
19
26
02
09
16
23
30
07
Jun
2014
Jul
ARMA(1,1) Dynamic Forecast
y
forecast
12
19
26
02
09
16
23
30
07
Jun
2014
Jul

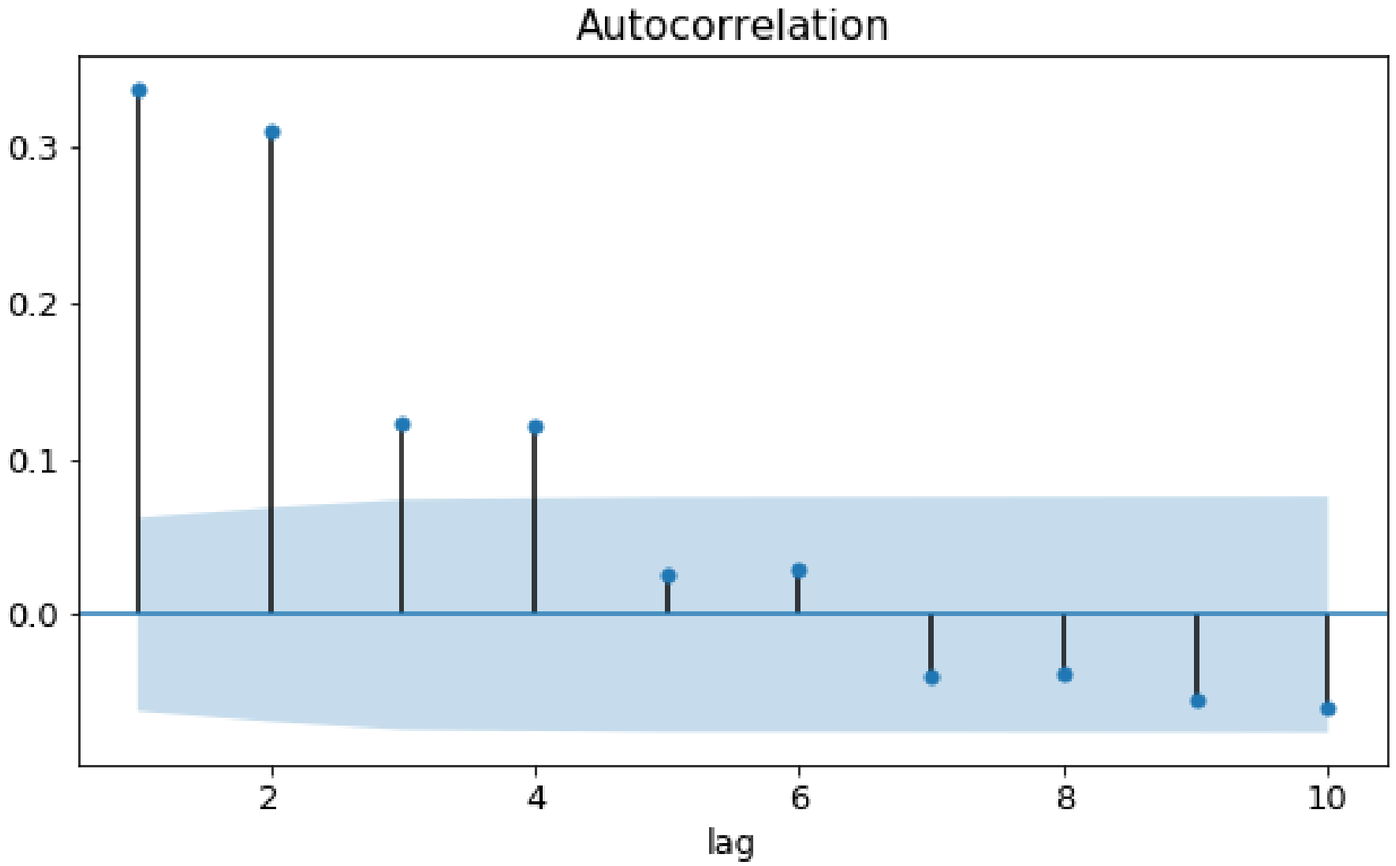
ACF and PACF

- ACF - Autocorrelation Function
- PACF - Partial autocorrelation function

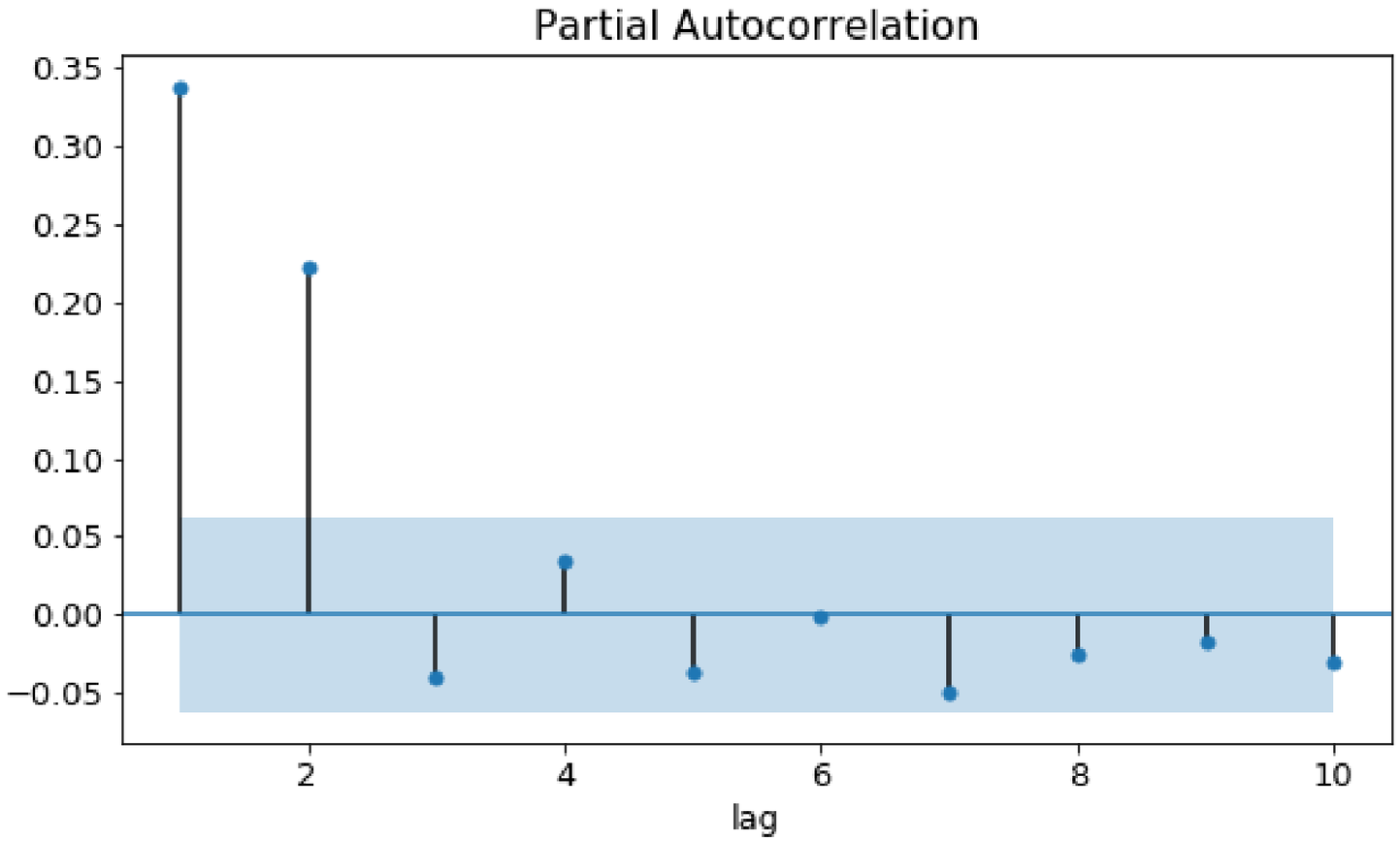
What is the ACF

- lag-1 autocorrelation $\rightarrow corr(y_t, y_{t-1})$
- lag-2 autocorrelation $\rightarrow corr(y_t, y_{t-2})$
- ...
- lag-n autocorrelation $\rightarrow corr(y_t, y_{t-n})$

What is the ACF



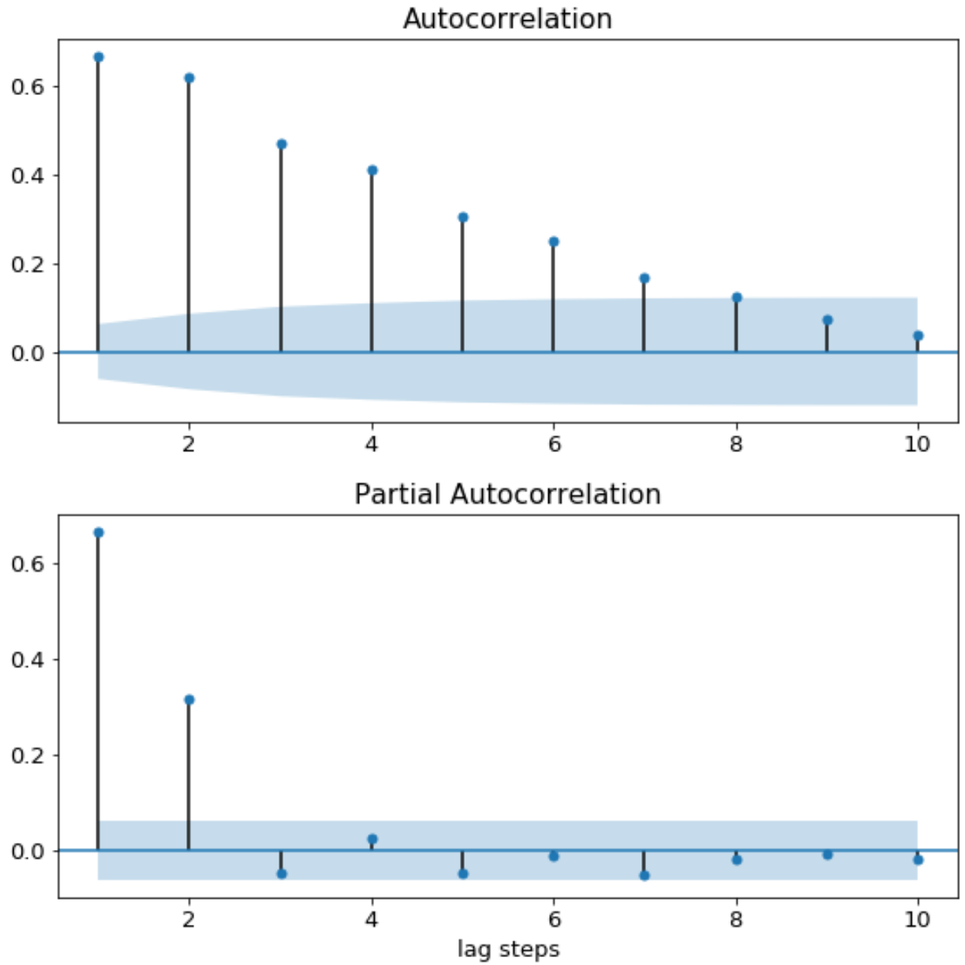
What is the PACF



Using ACF and PACF to choose model order

	AR(p)
ACF	Tails off
PACF	Cuts off after lag p

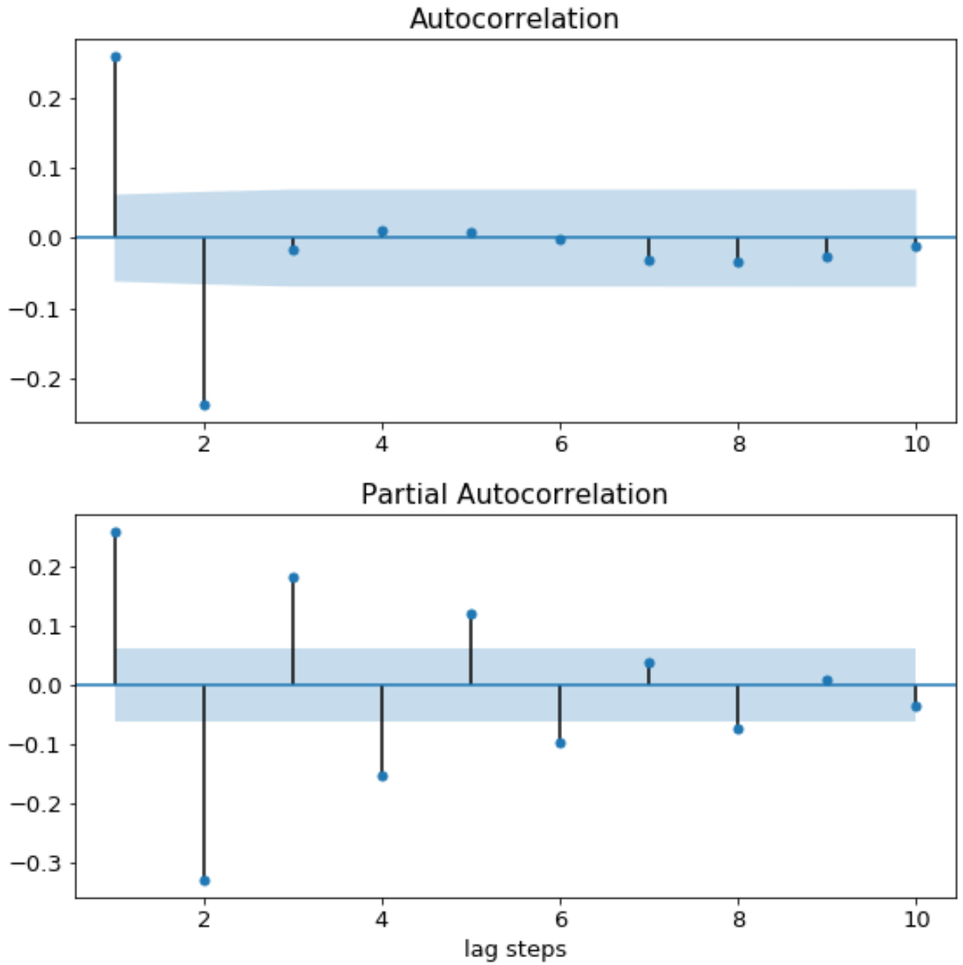
- AR(2) model $\rightarrow$



Using ACF and PACF to choose model order

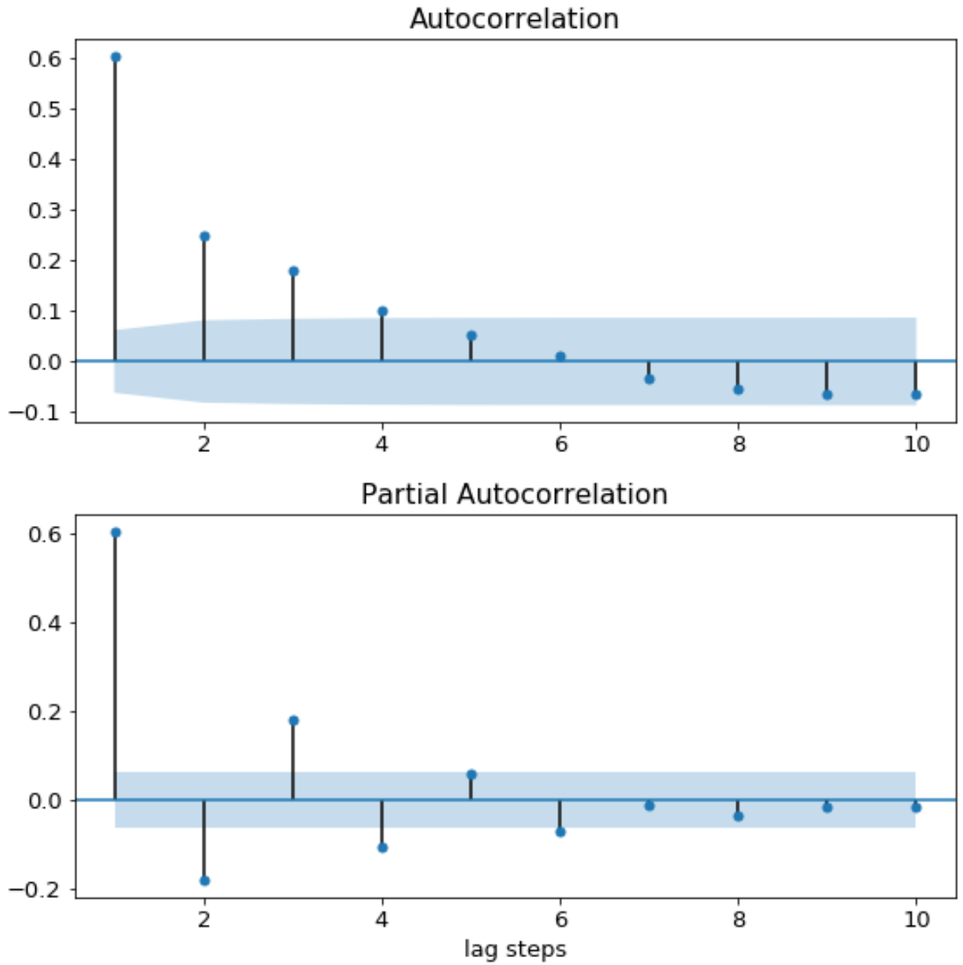
	MA(q)
ACF	Cuts off after lag q
PACF	Tails off

- MA(2) model $\rightarrow$



Using ACF and PACF to choose model order

	ARMA(p,q)
ACF	Tails off
PACF	Tails off



Using ACF and PACF to choose model order

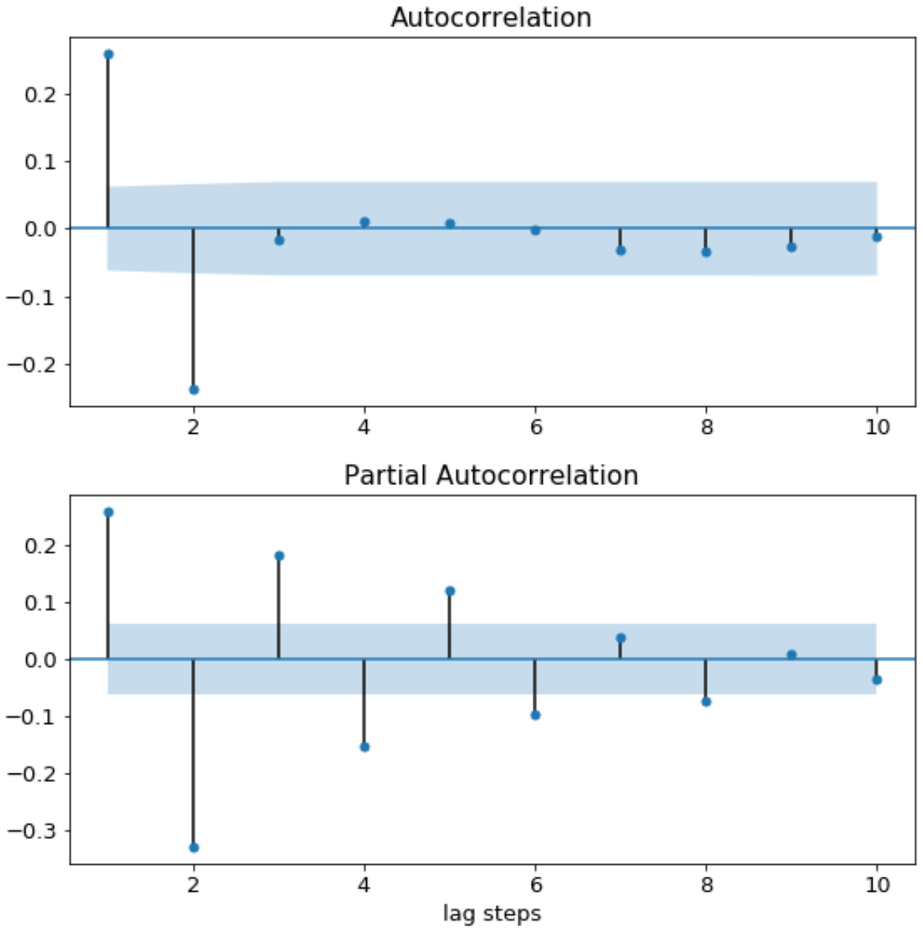
	AR(p)	MA(q)	ARMA(p,q)
ACF	Tails off	Cuts off after lag q	Tails off
PACF	Cuts off after lag p	Tails off	Tails off

Implementation in Python

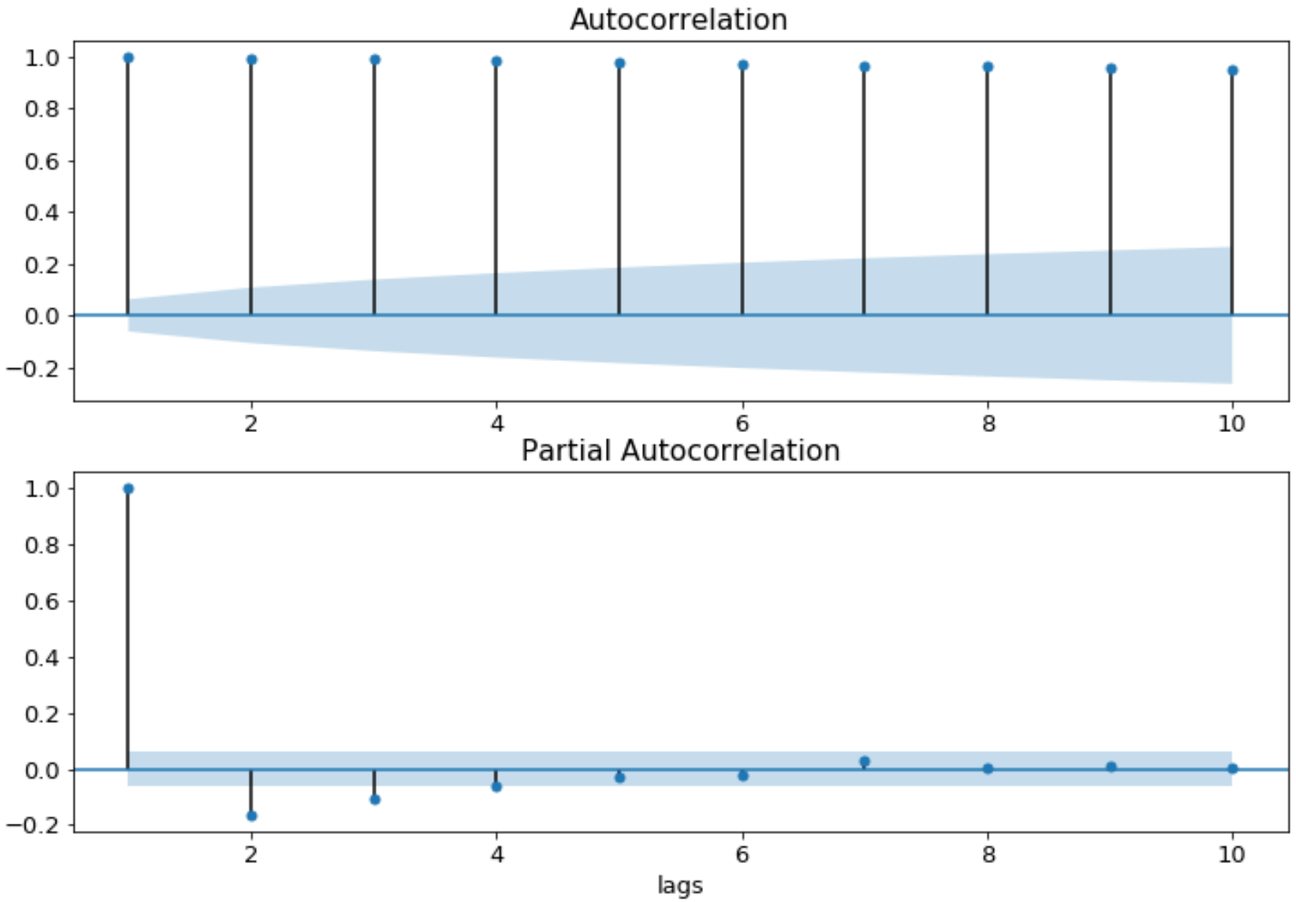
```
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
```

```
# Create figure
fig, (ax1, ax2) = plt.subplots(2,1, figsize=(8,8))
# Make ACF plot
plot_acf(df, lags=10, zero=False, ax=ax1)
# Make PACF plot
plot_pacf(df, lags=10, zero=False, ax=ax2)


plt.show()
```

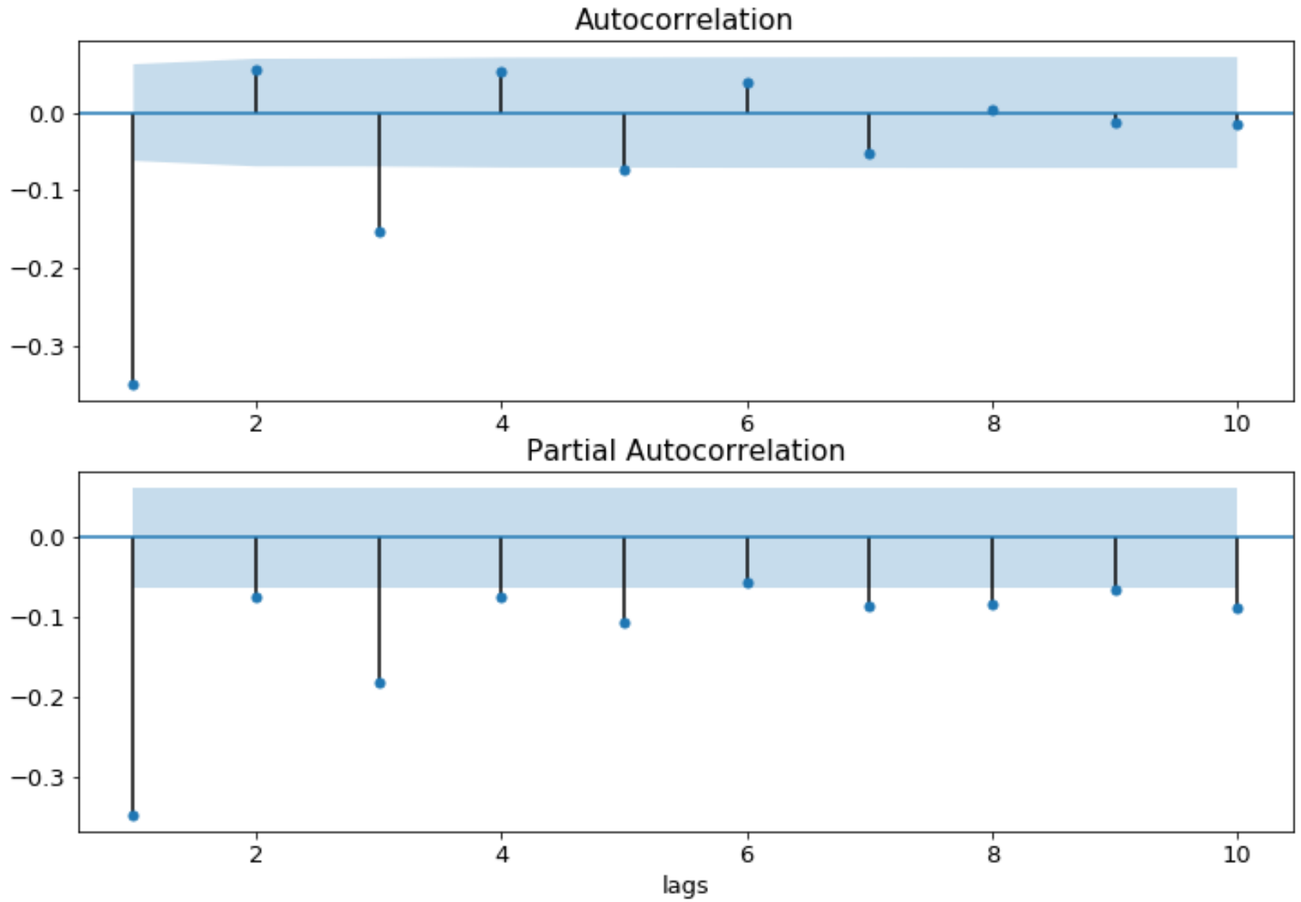
Implementation in Python



Over/under differencing and ACF and PACF



Over/under differencing and ACF and PACF



Let's practice!

ARIMA MODELS IN PYTHON

AIC and BIC

ARIMA MODELS IN PYTHON

James Fulton
Climate informatics researcher

datacamp

AIC - Akaike information criterion

- Lower AIC indicates a better model
- AIC likes to choose simple models with lower order

BIC - Bayesian information criterion

- Very similar to AIC
- Lower BIC indicates a better model
- BIC likes to choose simple models with lower order

AIC vs BIC

- BIC favors simpler models than AIC
- AIC is better at choosing predictive models
- BIC is better at choosing good explanatory model

AIC and BIC in statsmodels

```
# Create model
model = SARIMAX(df, order=(1,0,1))
# Fit model
results = model.fit()
# Print fit summary
print(results.summary())
```

```
                              Statespace Model Results
==============================================================================
Dep. Variable:                           y   No. Observations:                 1000
Model:                 SARIMAX(2, 0, 0)   Log Likelihood               -1399.704
Date:                  Fri, 10 May 2019   AIC                           2805.407
Time:                          01:06:11   BIC                           2820.131
Sample:                      01-01-2013   HQIC                          2811.003
                           - 09-27-2015
Covariance Type:                     opg
```

AIC and BIC in statsmodels

```
# Create model
model = SARIMAX(df, order=(1,0,1))
# Fit model
results = model.fit()
# Print AIC and BIC
print('AIC:', results.aic)
print('BIC:', results.bic)
```

```
AIC: 2806.36
BIC: 2821.09
```

Searching over AIC and BIC

```
# Loop over AR order
for p in range(3):
    # Loop over MA order
    for q in range(3):
        # Fit model
        model = SARIMAX(df, order=(p,0,q))
        results = model.fit()
        # print the model order and the AIC/BIC values
        print(p, q, results.aic, results.bic)
```

```
0 0 2900.13 2905.04
0 1 2828.70 2838.52
0 2 2806.69 2821.42
1 0 2810.25 2820.06
1 1 2806.37 2821.09
1 2 2807.52 2827.15
...
```

Searching over AIC and BIC

```
order_aic_bic =[]
# Loop over AR order
for p in range(3):
    # Loop over MA order
    for q in range(3):
        # Fit model
        model = SARIMAX(df, order=(p,0,q))
        results = model.fit()
        # Add order and scores to list
        order_aic_bic.append((p, q, results.aic, results.bic))
```

```
# Make DataFrame of model order and AIC/BIC scores
order_df = pd.DataFrame(order_aic_bic, columns=['p','q', 'aic', 'bic'])
```

Searching over AIC and BIC

```
# Sort by AIC
print(order_df.sort_values('aic'))
```

```
   p  q      aic      bic
7  2  1  2804.54  2824.17
6  2  0  2805.41  2820.13
4  1  1  2806.37  2821.09
2  0  2  2806.69  2821.42
...
```

```
# Sort by BIC
print(order_df.sort_values('bic'))
```

```
   p  q      aic      bic
3  1  0  2810.25  2820.06
6  2  0  2805.41  2820.13
4  1  1  2806.37  2821.09
2  0  2  2806.69  2821.42
...
```

Non-stationary model orders

```
# Fit model
model = SARIMAX(df, order=(2,0,1))
results = model.fit()
```

```
ValueError: Non-stationary starting autoregressive parameters
found with `enforce_stationarity` set to True.
```

When certain orders don't work

```
# Loop over AR order
for p in range(3):
    # Loop over MA order
    for q in range(3):

        # Fit model
        model = SARIMAX(df, order=(p,0,q))
        results = model.fit()

        # Print the model order and the AIC/BIC values
        print(p, q, results.aic, results.bic)
```

When certain orders don't work

```
# Loop over AR order
for p in range(3):
    # Loop over MA order
    for q in range(3):
        try:
            # Fit model
            model = SARIMAX(df, order=(p,0,q))
            results = model.fit()

            # Print the model order and the AIC/BIC values
            print(p, q, results.aic, results.bic)
        except:
            # Print AIC and BIC as None when fails
            print(p, q, None, None)
```

Let's practice!

ARIMA MODELS IN PYTHON

Model diagnostics

ARIMA MODELS IN PYTHON

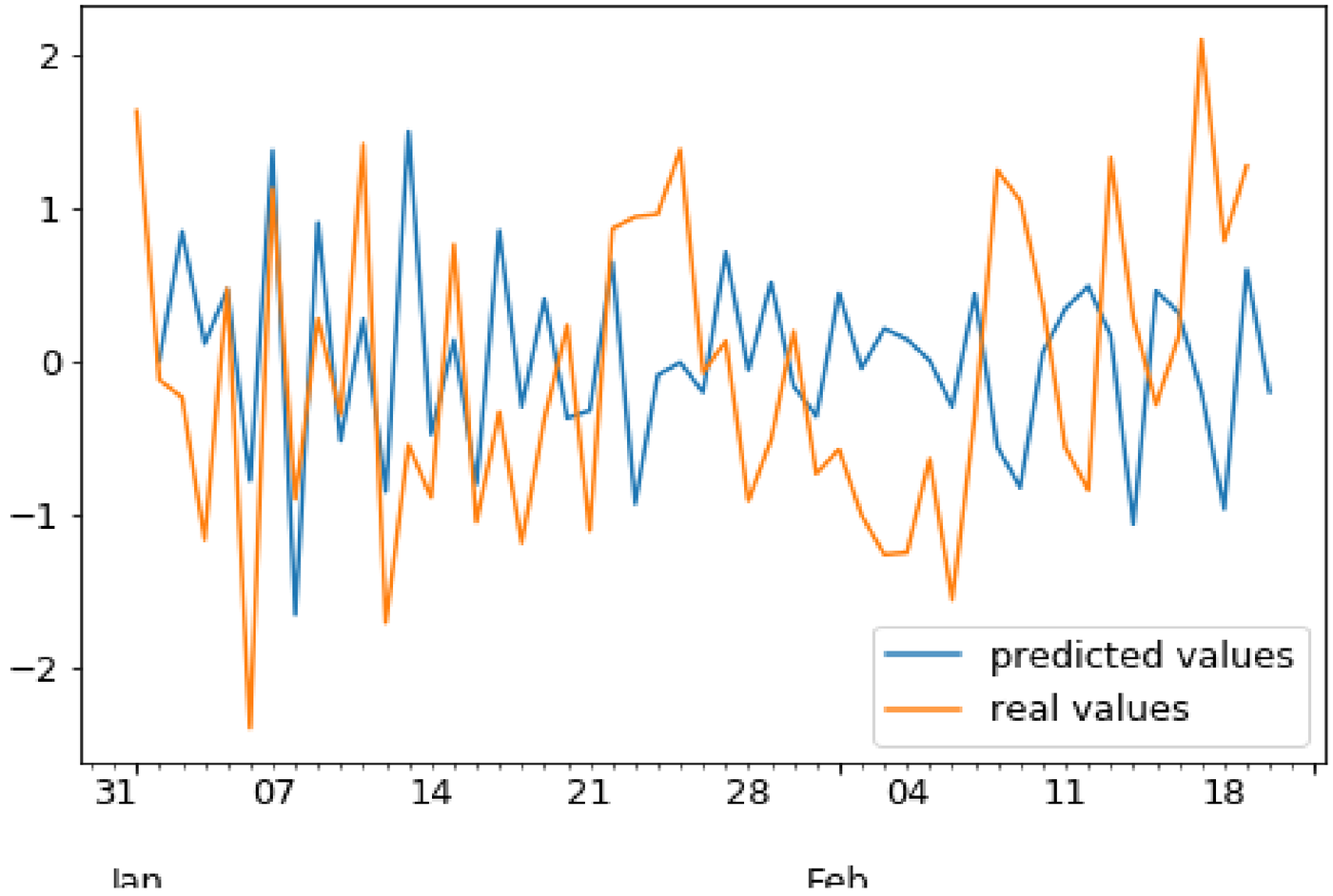
James Fulton
Climate informatics researcher

datacamp

Introduction to model diagnostics

- How good is the final model?

Residuals


2
1
0
−1
−2
31
07
14
21
28
04
11
18
Jan
Feb
predicted values
real values

Residuals

```
# Fit model
model = SARIMAX(df, order=(p,d,q))
results = model.fit()
# Assign residuals to variable
residuals = results.resid
```

```
2013-01-23    1.013129
2013-01-24    0.114055
2013-01-25    0.430698
2013-01-26   -1.247046
2013-01-27   -0.499565
       ...         ...
```

Mean absolute error

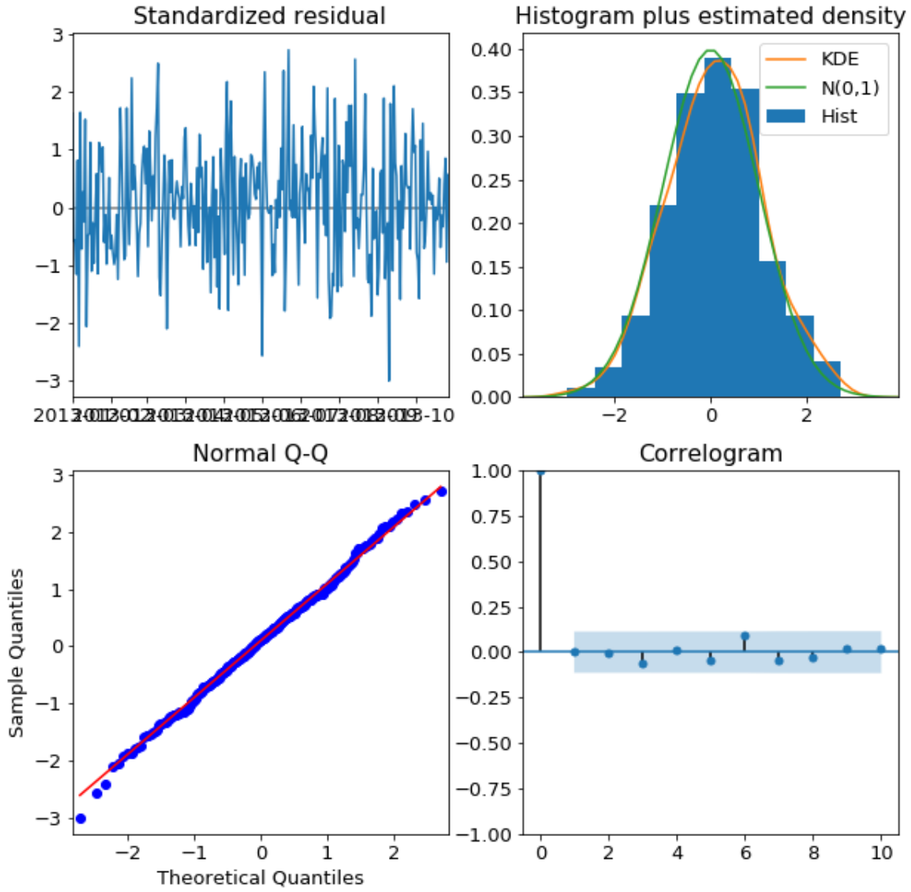
How far our the predictions from the real values?

```
mae = np.mean(np.abs(residuals))
```

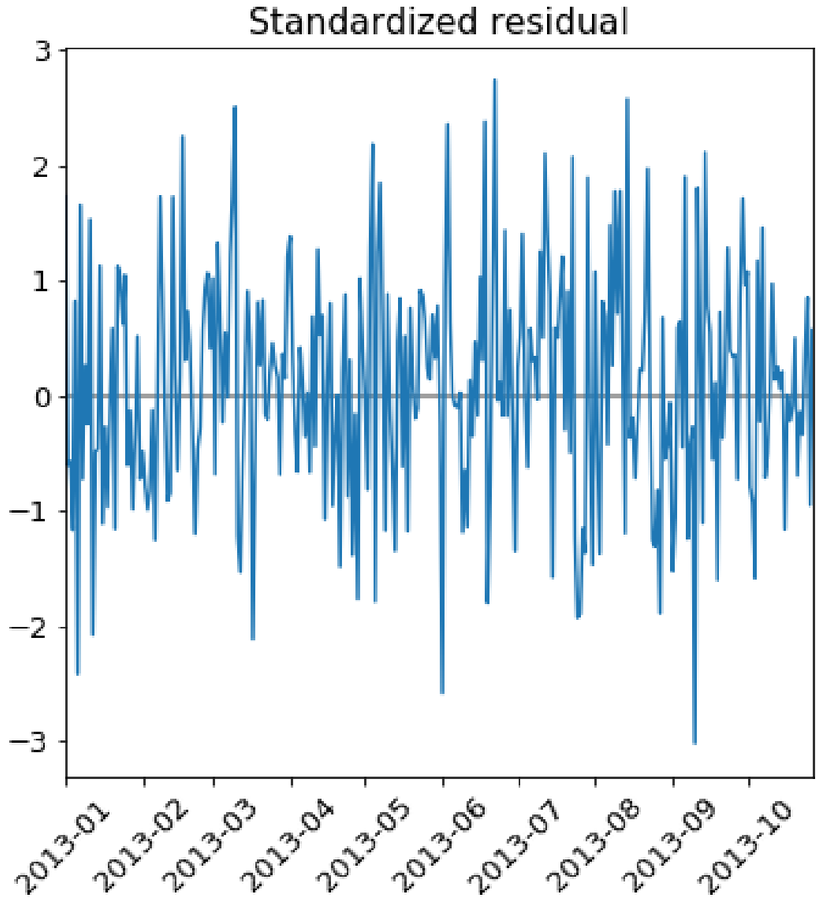
Plot diagnostics

If the model fits well the residuals will be white Gaussian noise

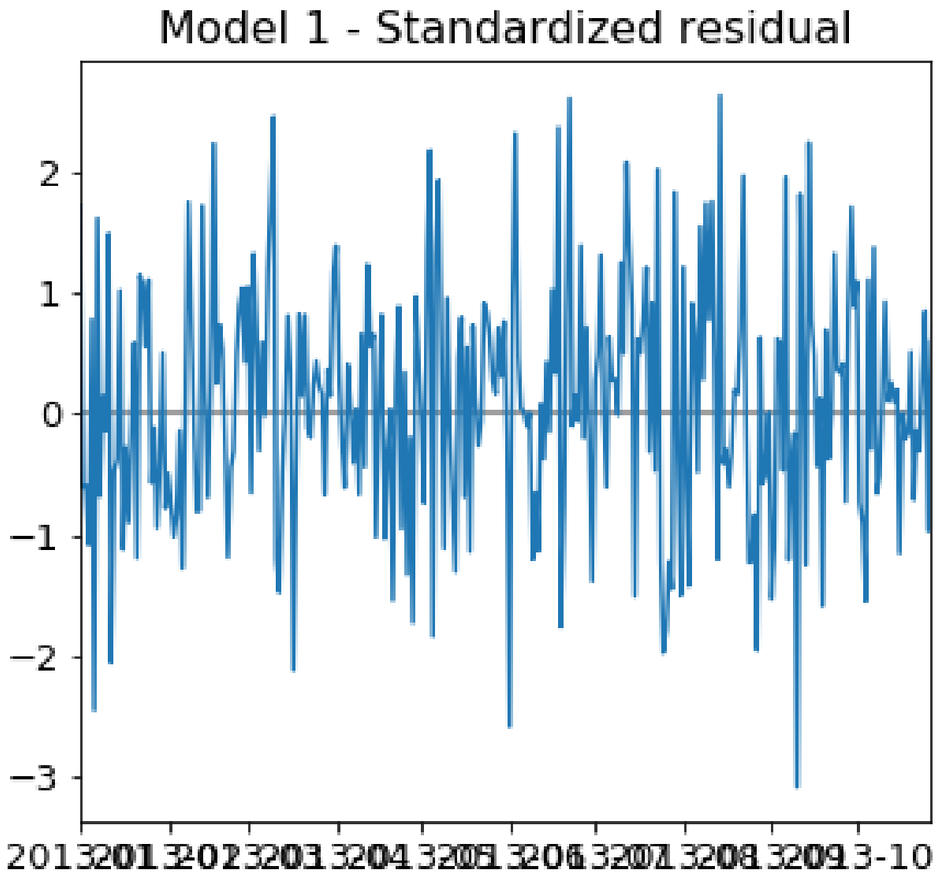
```
# Create the 4 diagostics plots
results.plot_diagnostics()
plt.show()
```

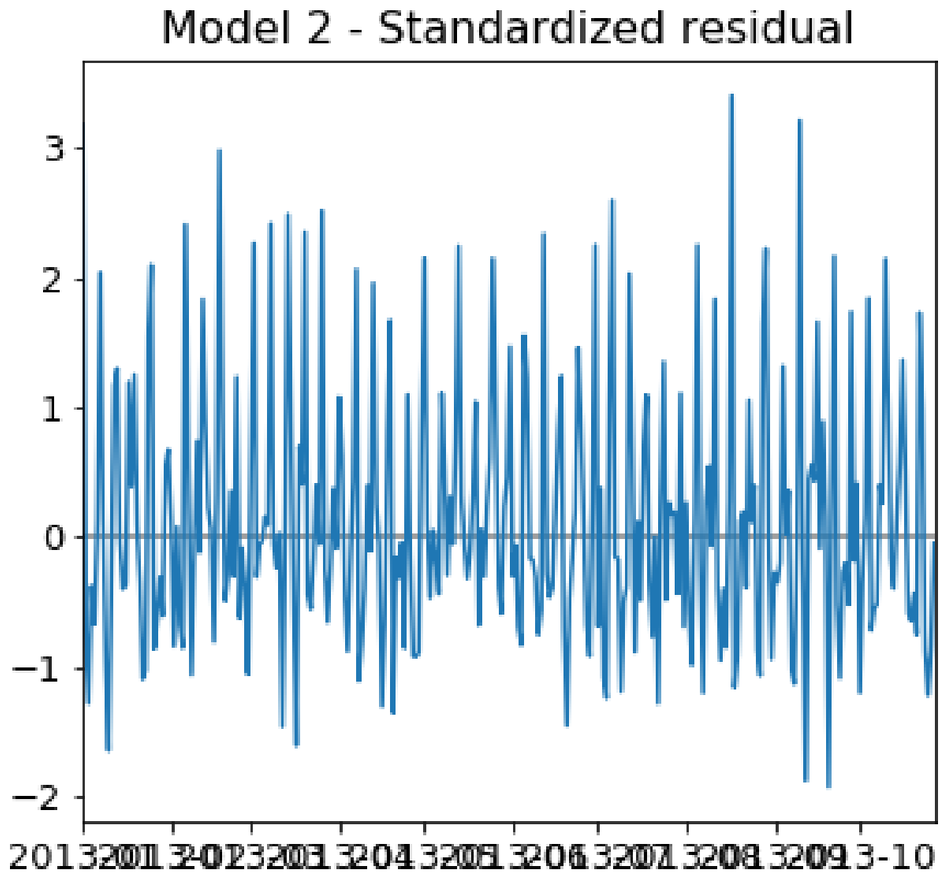


Residuals plot

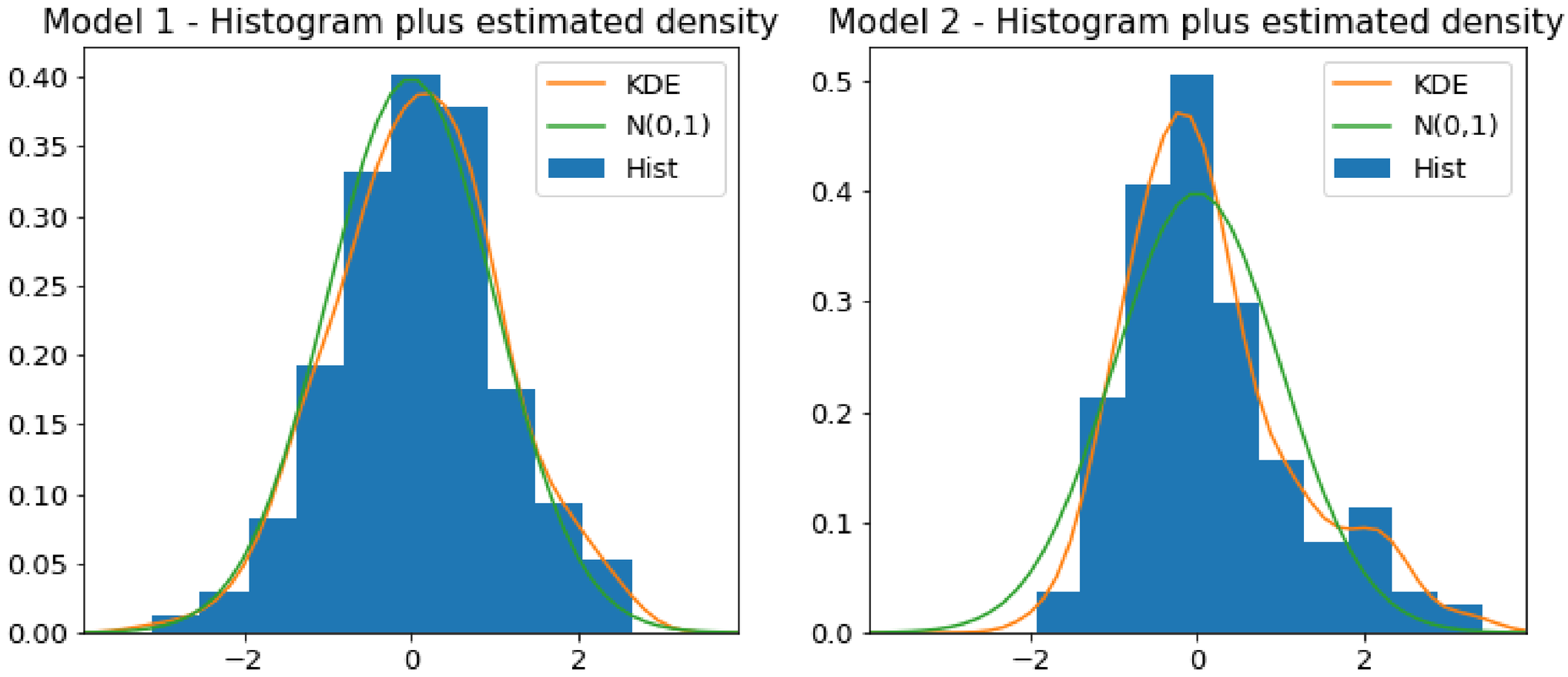


Residuals plot

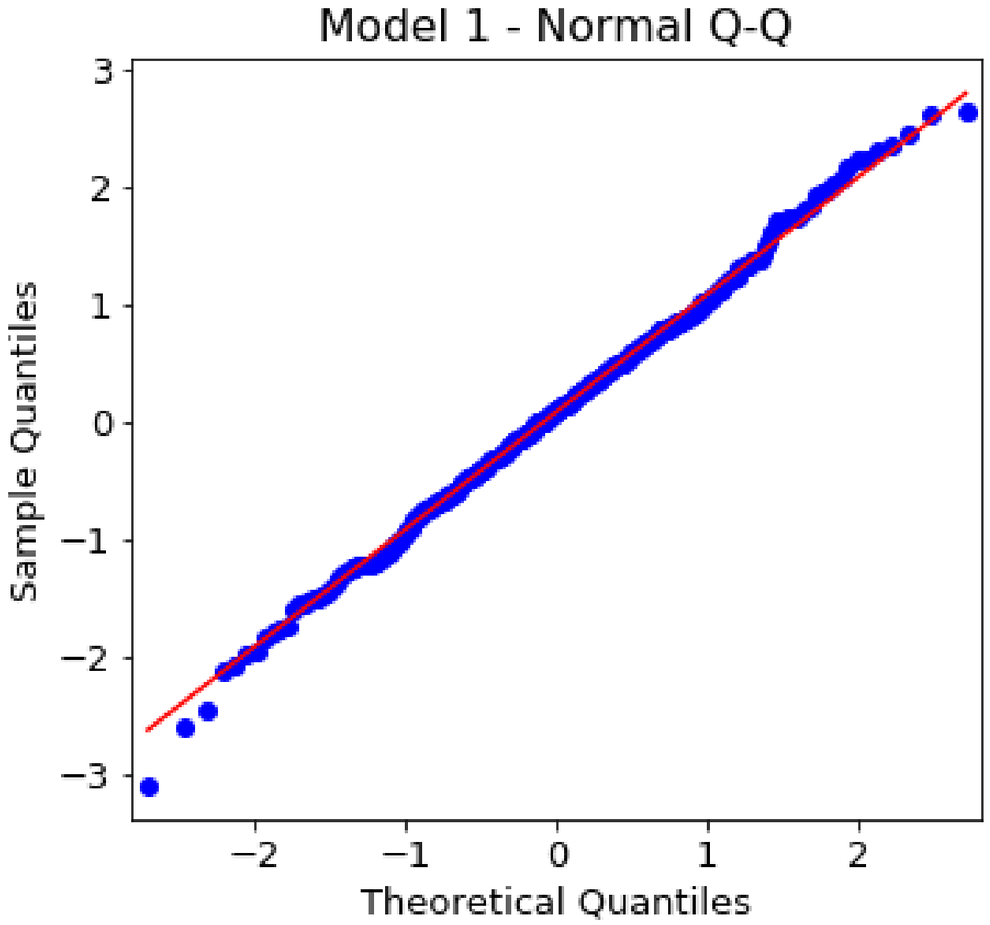


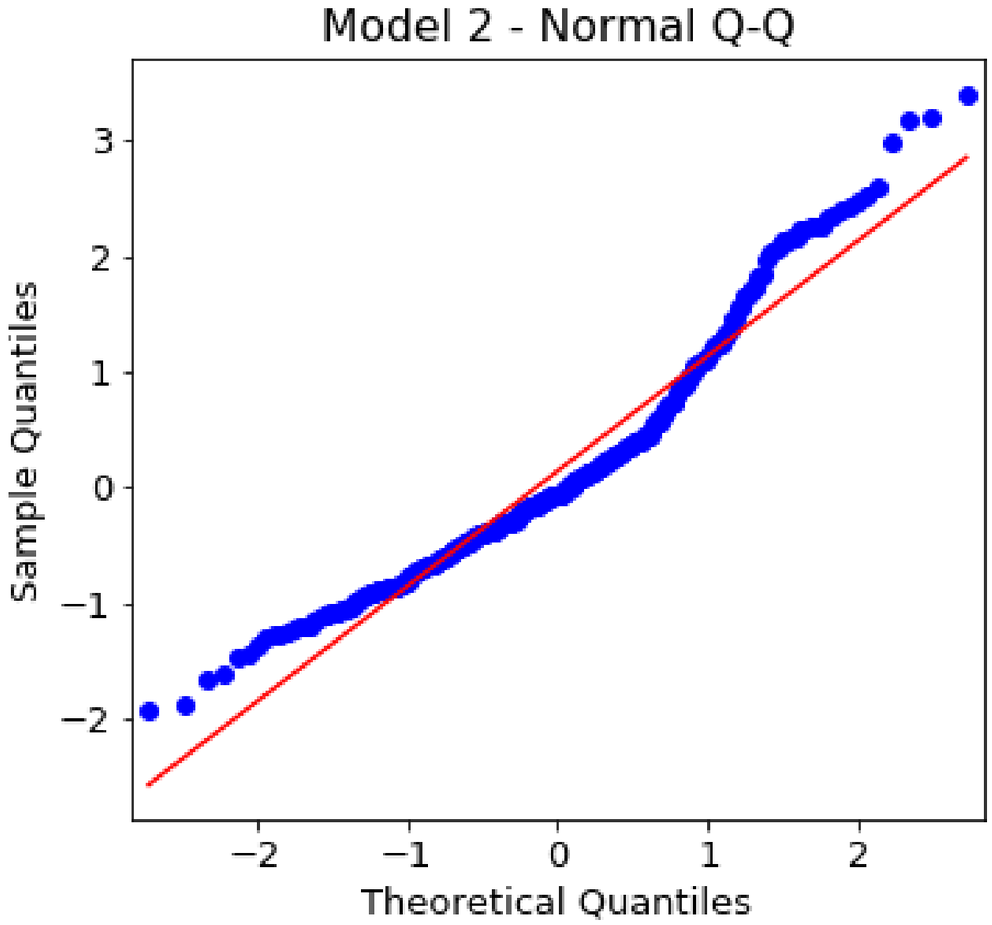


Histogram plus estimated density

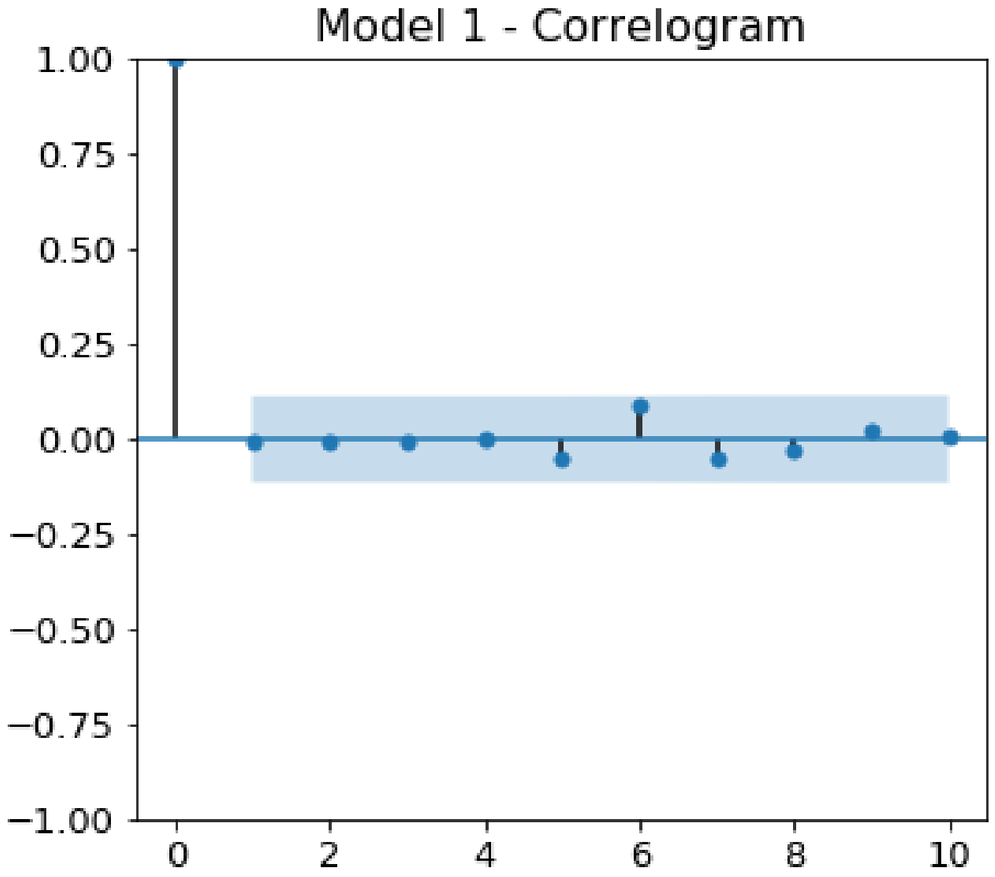


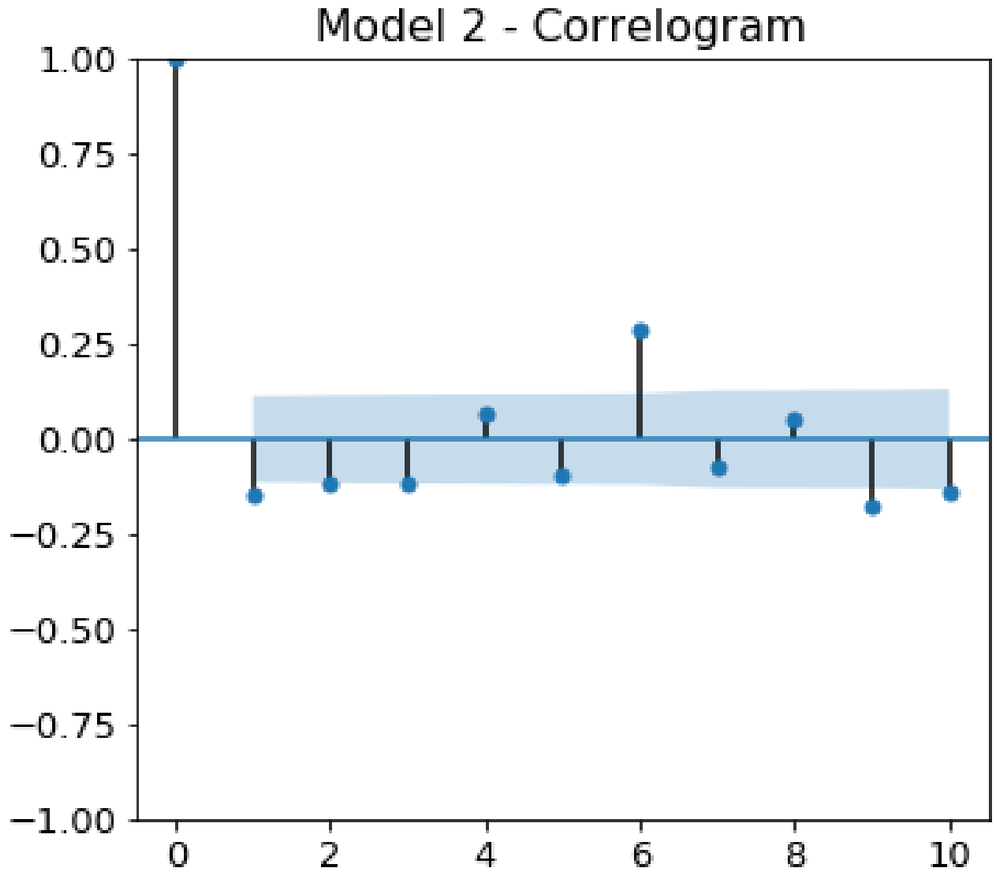
Normal Q-Q


Model 1 - Normal Q-Q
Sample Quantiles
Theoretical Quantiles
3
2
1
0
−1
−2
−3
−2
−1
0
1
2


Model 2 - Normal Q-Q
Sample Quantiles
Theoretical Quantiles
3
2
1
0
−1
−2
−2
−1
0
1
2

Correlogram





Summary statistics

```
print(results.summary())
```

```
...
==============================================================================
Ljung-Box (Q):                       32.10   Jarque-Bera (JB):                 0.02
Prob(Q):                              0.81   Prob(JB):                         0.99
Heteroskedasticity (H):               1.28   Skew:                            -0.02
Prob(H) (two-sided):                  0.21   Kurtosis:                         2.98
==============================================================================
```

- `Prob(Q)` - p-value for null hypothesis that residuals are uncorrelated
- `Prob(JB)` - p-value for null hypothesis that residuals are normal

Let's practice!

ARIMA MODELS IN PYTHON

Box-Jenkins method

ARIMA MODELS IN PYTHON

James Fulton
Climate informatics researcher

datacamp

The Box-Jenkins method

From raw data $\rightarrow$ production model

- identification
- estimation
- model diagnostics

Identification

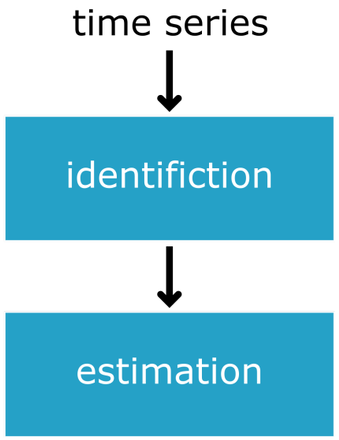
- Is the time series stationary?
- What differencing will make it stationary?
- What transforms will make it stationary?
- What values of p and q are most promising?

Identification tools

- Plot the time series
  - `df.plot()`
- Use augmented Dicky-Fuller test
  - `adfuller()`
- Use transforms and/or differencing
  - `df.diff()`, `np.log()`, `np.sqrt()`
- Plot ACF/PACF
  - `plot_acf()`, `plot_pacf()`

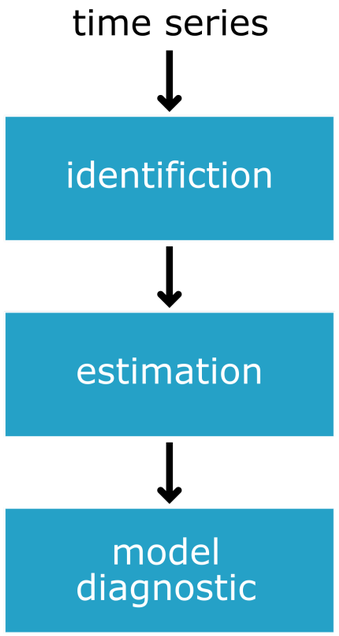
Estimation

- Use the data to train the model coefficients
- Done for us using `model.fit()`
- Choose between models using AIC and BIC
  - `results.aic`, `results.bic`

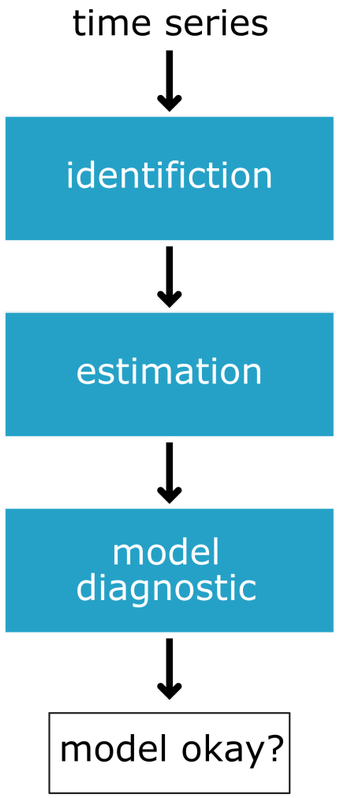


Model diagnostics

- Are the residuals uncorrelated
- Are residuals normally distributed
  - `results.plot_diagnostics()`
  - `results.summary()`

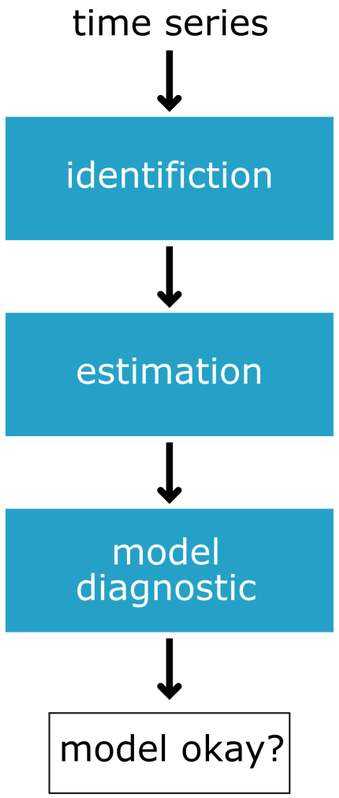


Decision


time series
identifiction
estimation
model diagnostic
model okay?

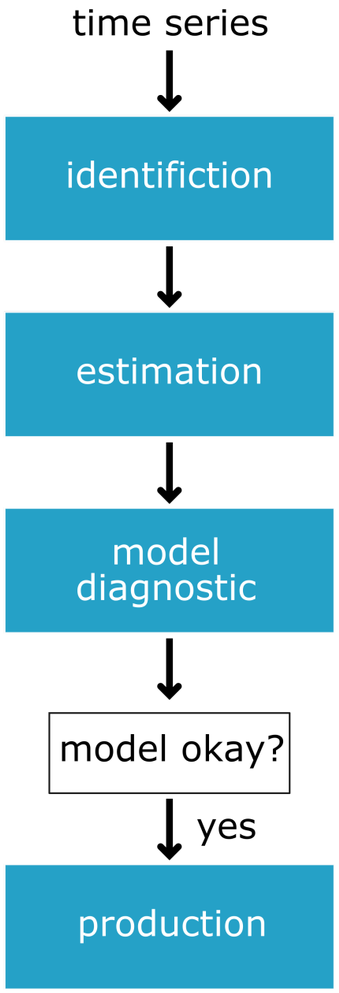
Repeat

- We go through the process again with more information
- Find a better model

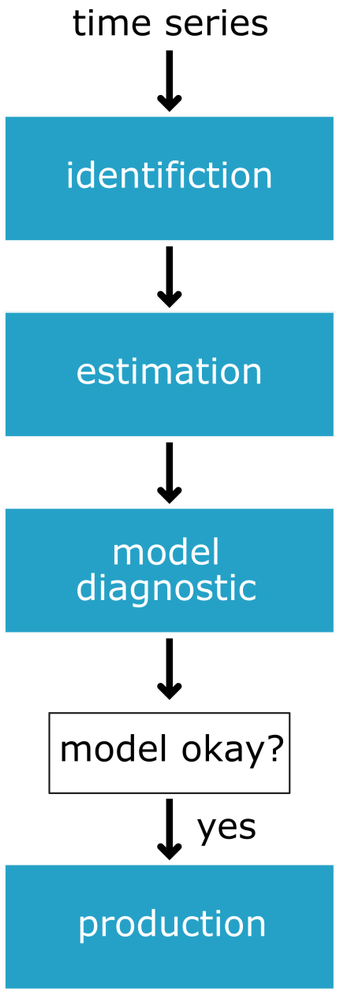


Production

- Ready to make forecasts
  - `results.get_forecast()`



Box-Jenkins


time series
identifiction
estimation
model diagnostic
model okay?
yes
production

Let's practice!

ARIMA MODELS IN PYTHON

Seasonal time series

ARIMA MODELS IN PYTHON

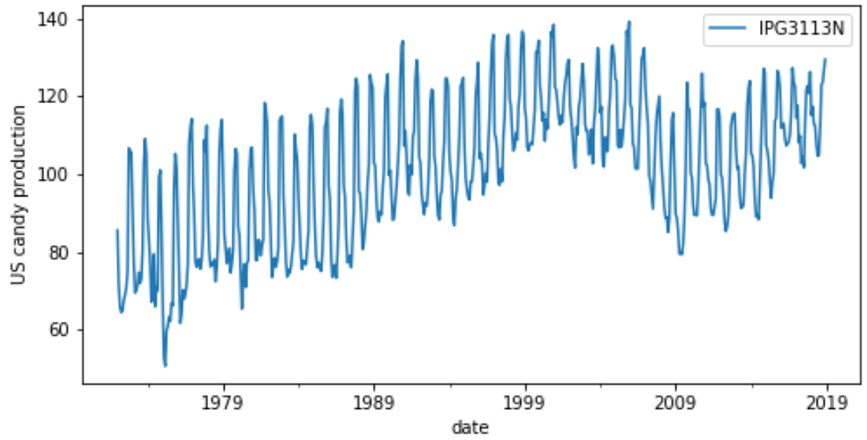
James Fulton
Climate informatics researcher

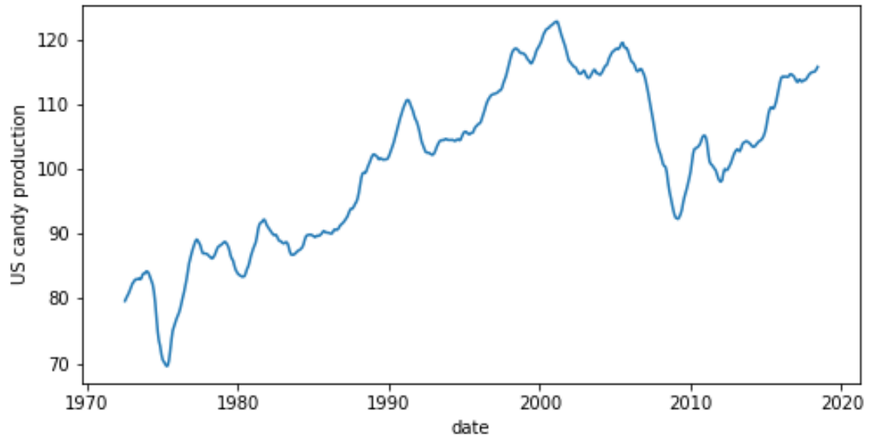
datacamp

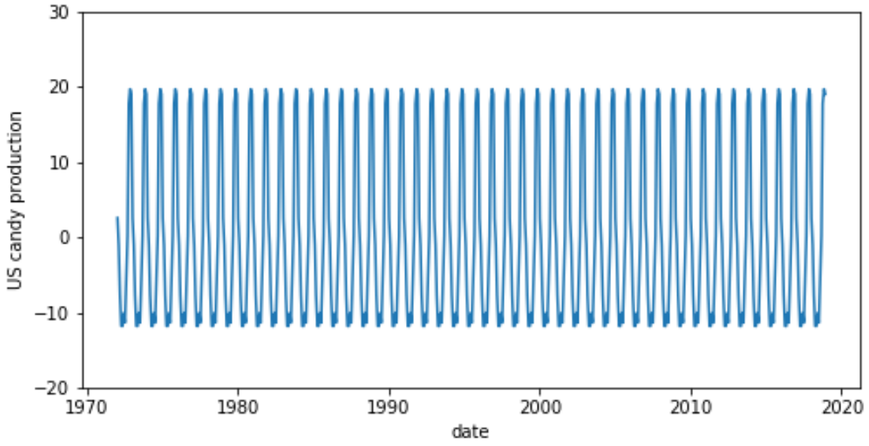
Seasonal data

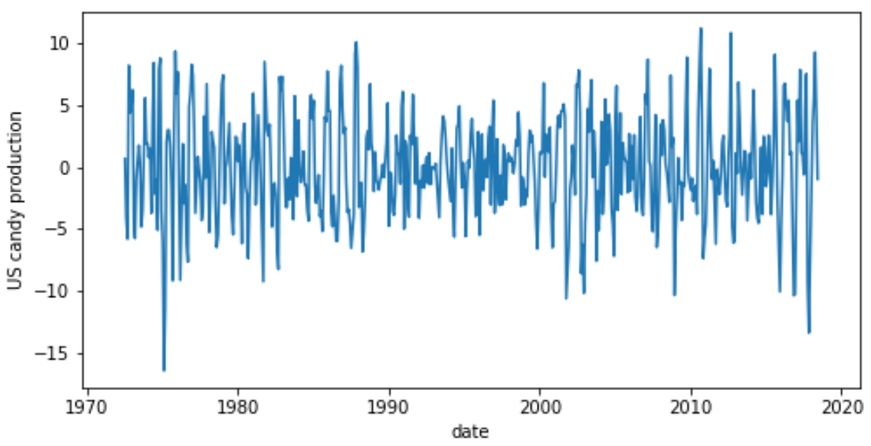
- Has predictable and repeated patterns
- Repeats after any amount of time

Seasonal decomposition

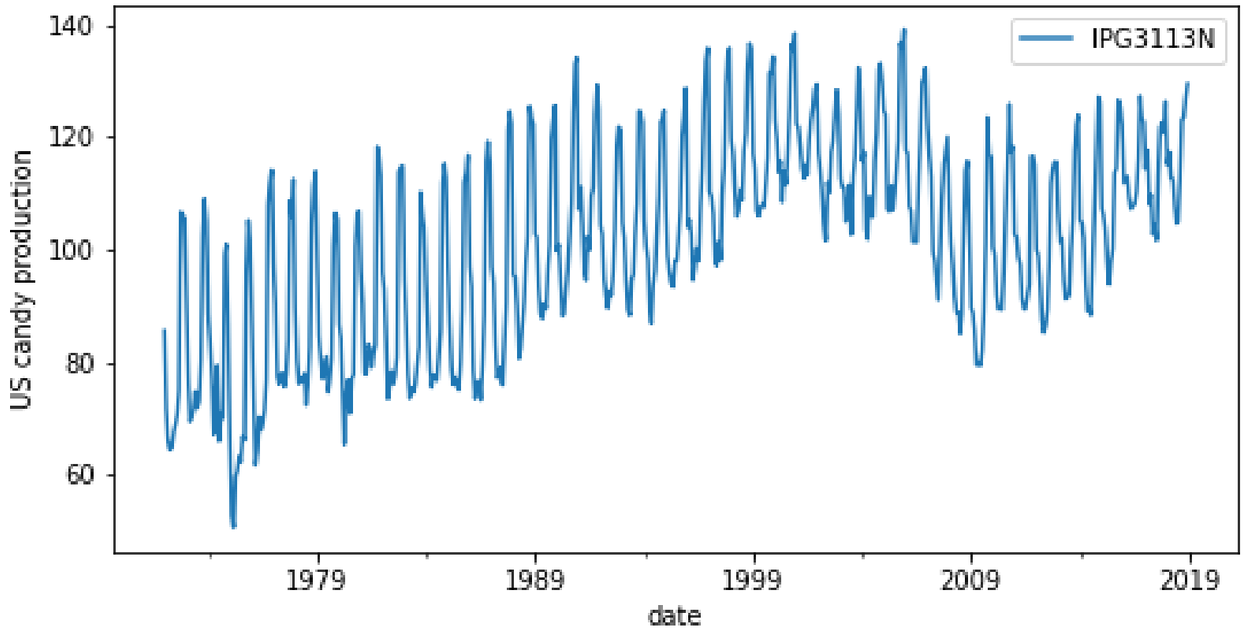








Seasonal decomposition



time series = trend + seasonal + residual

Seasonal decomposition using statsmodels

```
# Import
from statsmodels.tsa.seasonal import seasonal_decompose
```

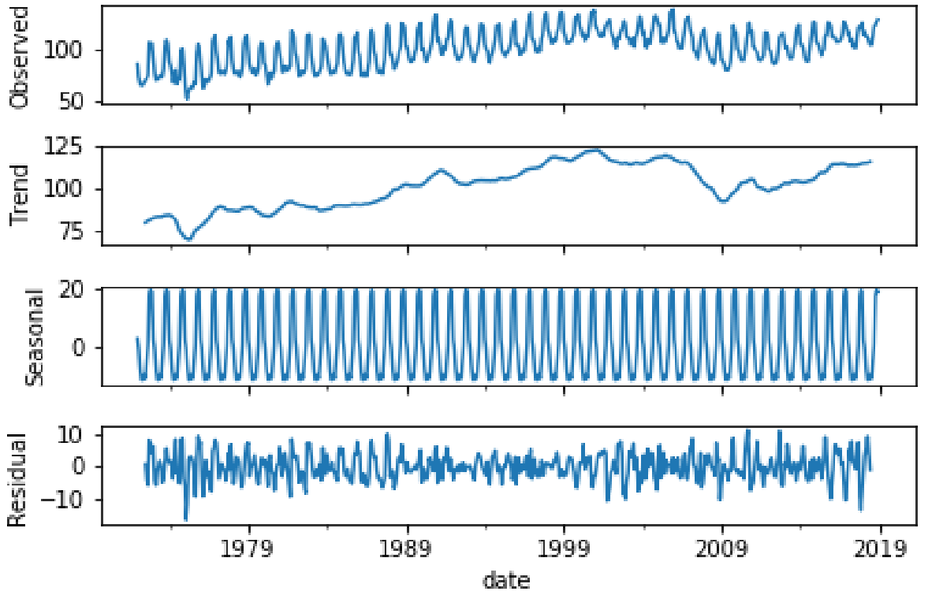
```
# Decompose data
decomp_results = seasonal_decompose(df['IPG3113N'], freq=12)
```

```
type(decomp_results)
```

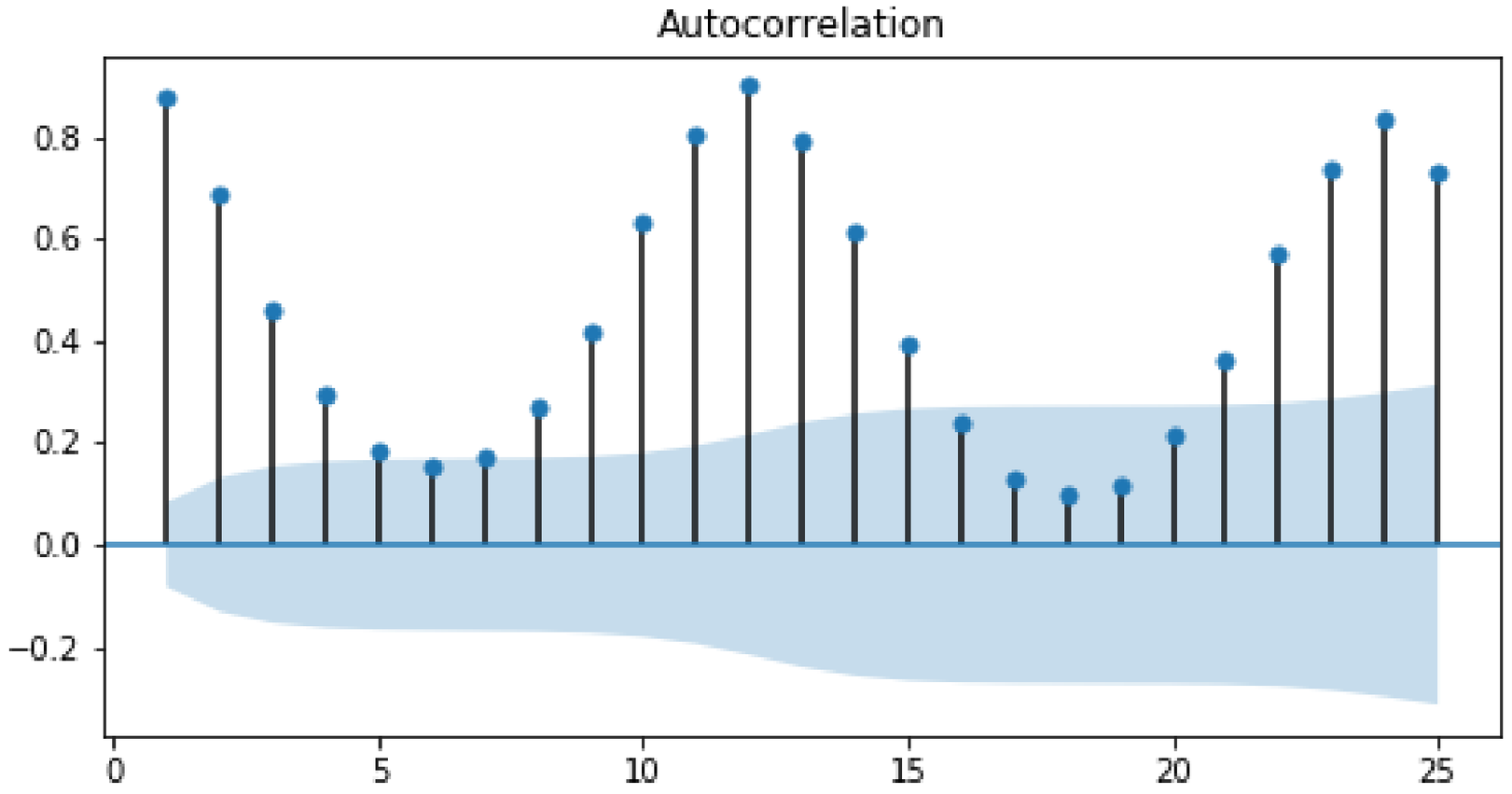
```
statsmodels.tsa.seasonal.DecomposeResult
```

Seasonal decomposition using statsmodels

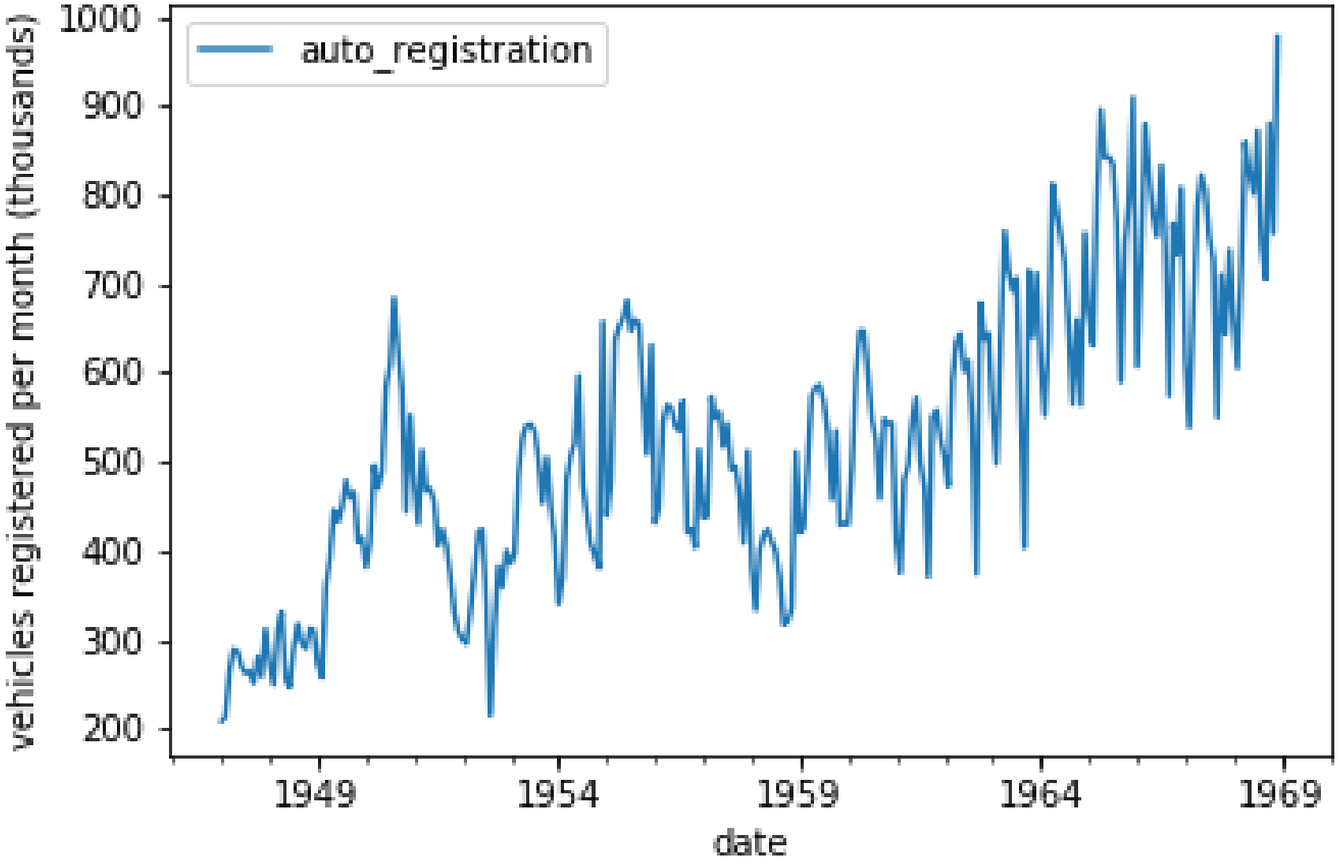
```
# Plot decomposed data
decomp_results.plot()
plt.show()
```



Finding seasonal period using ACF

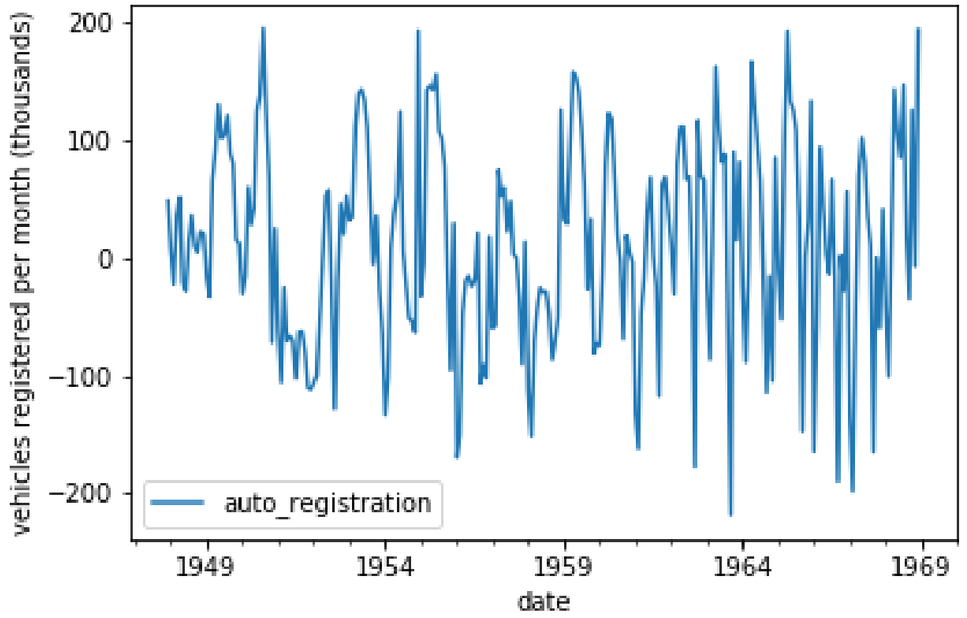


Identifying seasonal data using ACF



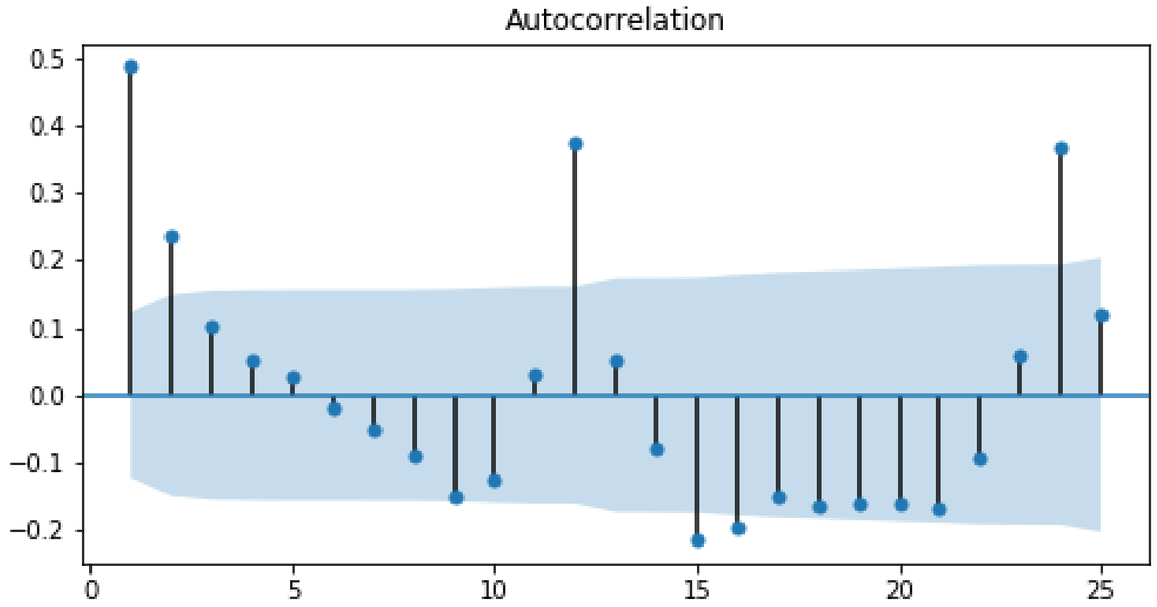
Detrending time series

```
# Subtract long rolling average over N steps
df = df - df.rolling(N).mean()
# Drop NaN values
df = df.dropna()
```

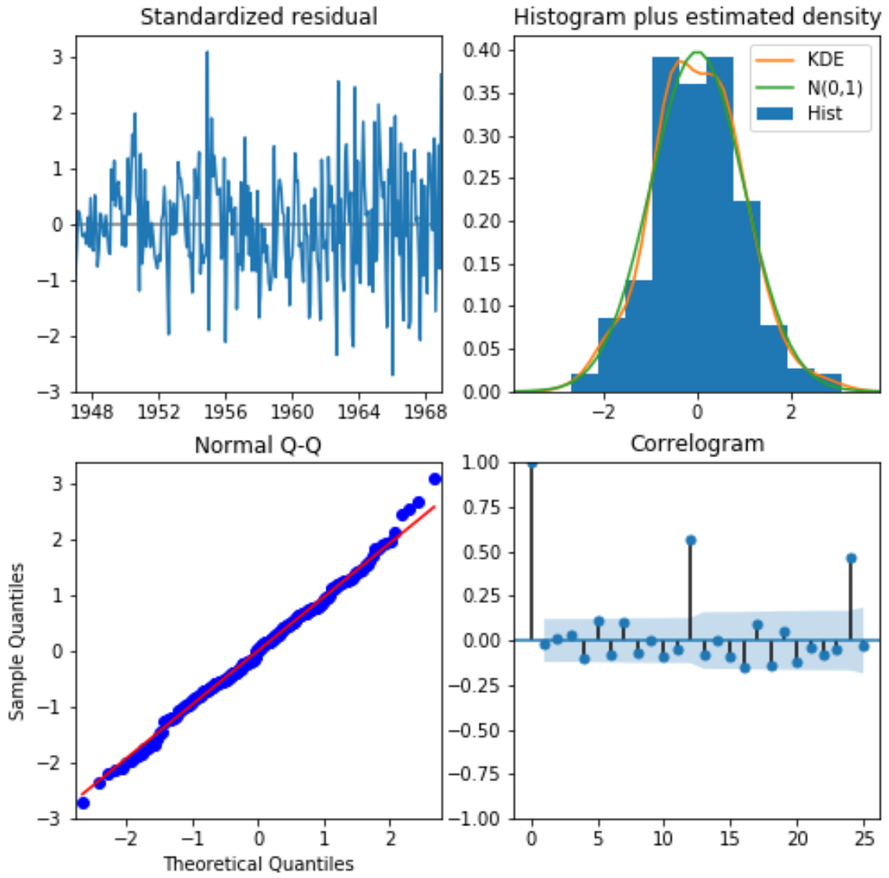


Identifying seasonal data using ACF

```
# Create figure
fig, ax = plt.subplots(1,1, figsize=(8,4))

# Plot ACF
plot_acf(df.dropna(), ax=ax, lags=25, zero=False)
plt.show()
```



ARIMA models and seasonal data



Let's practice!

ARIMA MODELS IN PYTHON

SARIMA models

ARIMA MODELS IN PYTHON

James Fulton
Climate informatics researcher

datacamp

The SARIMA model

Seasonal ARIMA = SARIMA

- Non-seasonal orders
  - p: autoregressive order
  - d: differencing order
  - q: moving average order

SARIMA(p,d,q)(P,D,Q)$_S$

- Seasonal Orders
  - P: seasonal autoregressive order
  - D: seasonal differencing order
  - Q: seasonal moving average order
  - S: number of time steps per cycle

The SARIMA model

ARIMA(2,0,1) model :

$$y_t = a_1 y_{t-1} + a_2 y_{t-2} + m_1 \epsilon_{t-1} + \epsilon_t$$

SARIMA(0,0,0)(2,0,1)$_7$ model:

$$y_t = a_7 y_{t-7} + a_{14} y_{t-14} + m_7 \epsilon_{t-7} + \epsilon_t$$

Fitting a SARIMA model

```
# Imports
from statsmodels.tsa.statespace.sarimax import SARIMAX
# Instantiate model
model = SARIMAX(df, order=(p,d,q), seasonal_order=(P,D,Q,S))
# Fit model
results = model.fit()
```

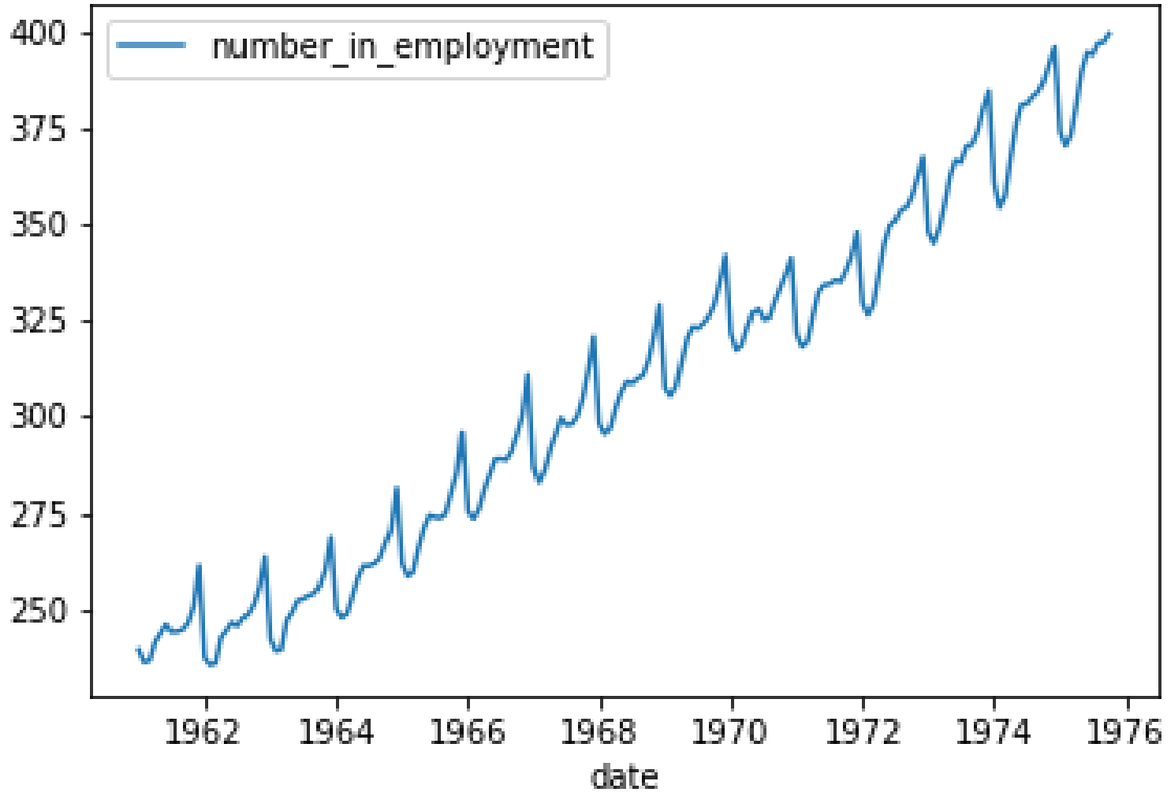
Seasonal differencing

Subtract the time series value of one season ago

$$\Delta y_t = y_t - y_{t-S}$$

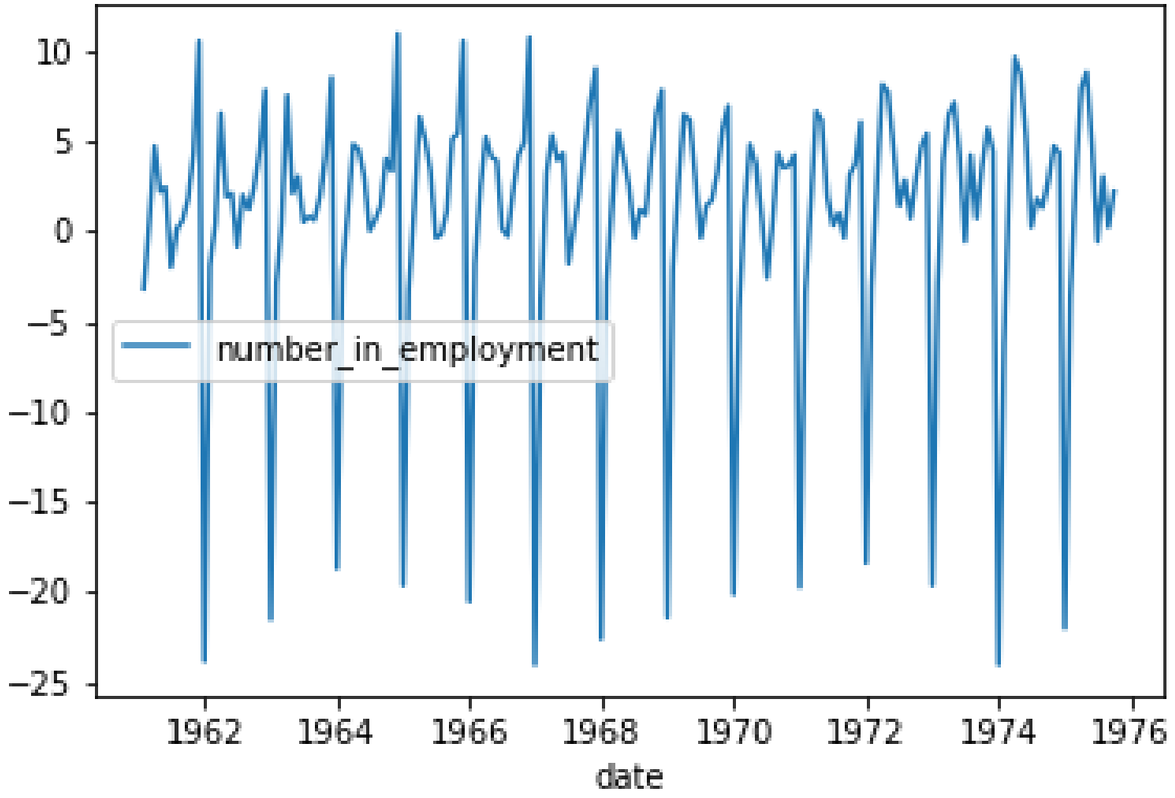
```
# Take the seasonal difference
df_diff = df.diff(S)
```

Differencing for SARIMA models



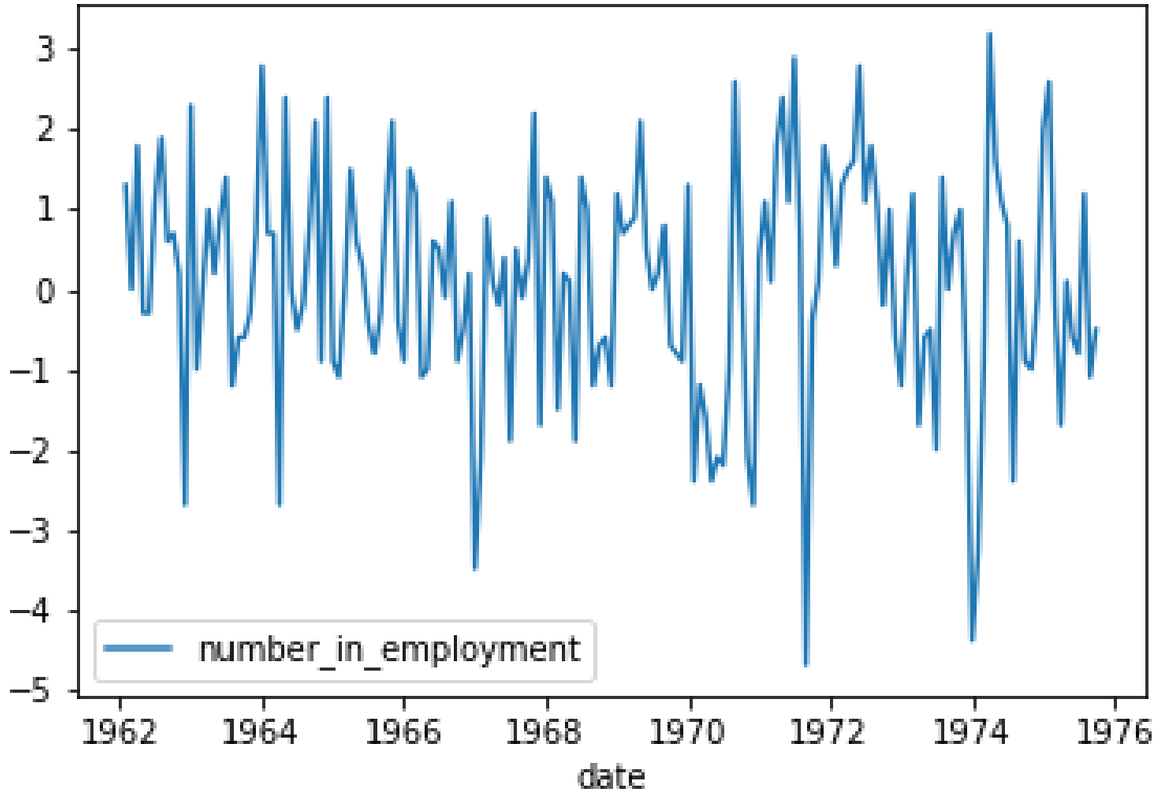
Time series

Differencing for SARIMA models



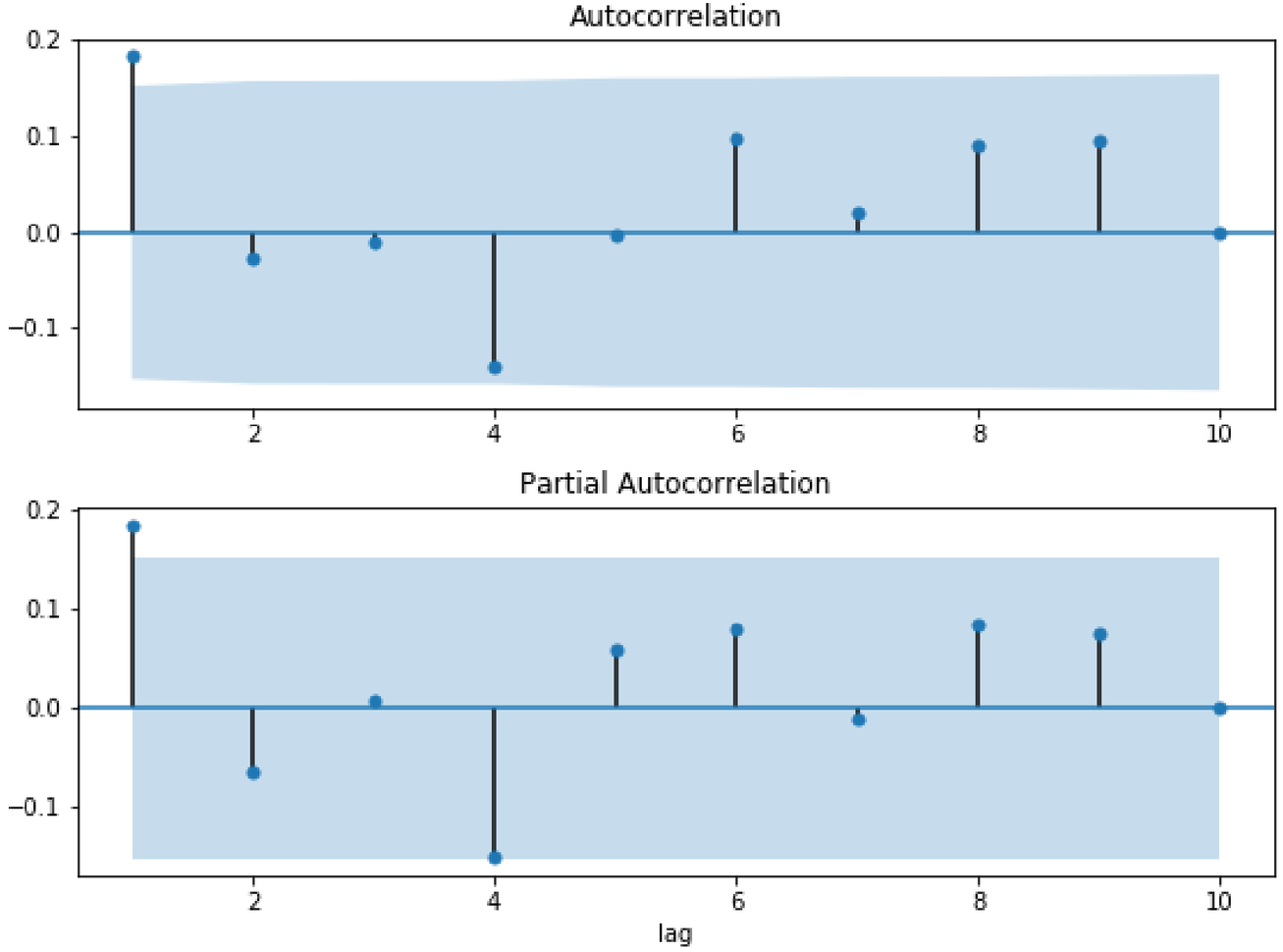
First difference of time series

Differencing for SARIMA models

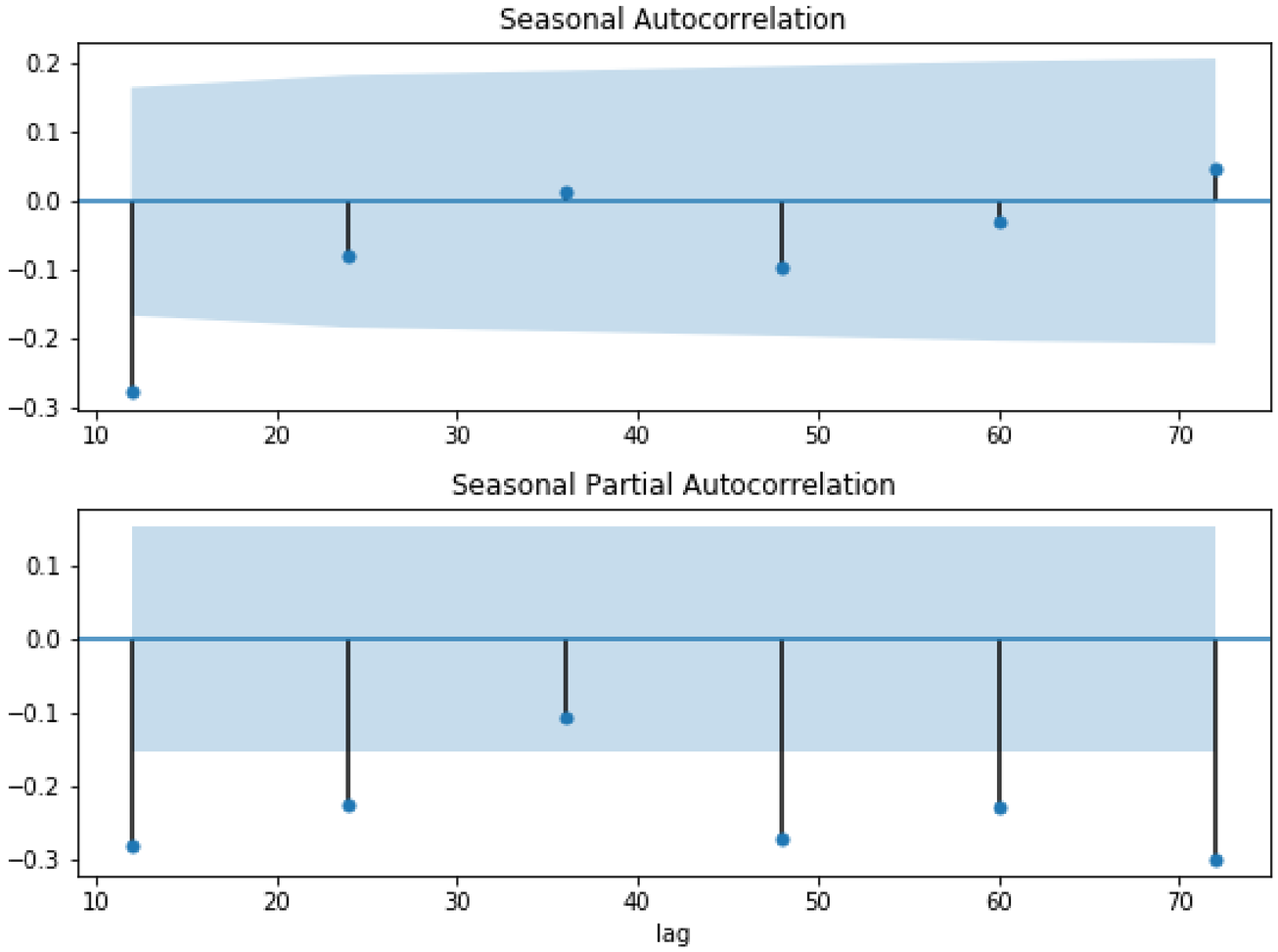


First difference and first seasonal difference of ime series

Finding p and q



Finding P and Q



Plotting seasonal ACF and PACF

```
# Create figure
fig, (ax1, ax2) = plt.subplots(2,1)

# Plot seasonal ACF
plot_acf(df_diff,  lags=[12,24,36,48,60,72], ax=ax1)

# Plot seasonal PACF
plot_pacf(df_diff, lags=[12,24,36,48,60,72], ax=ax2)

plt.show()
```

Let's practice!

ARIMA MODELS IN PYTHON

Automation and saving

ARIMA MODELS IN PYTHON

James Fulton
Climate informatics researcher

datacamp

Searching over model orders

```
import pmdarima as pm
```

```
results = pm.auto_arima(df)
```

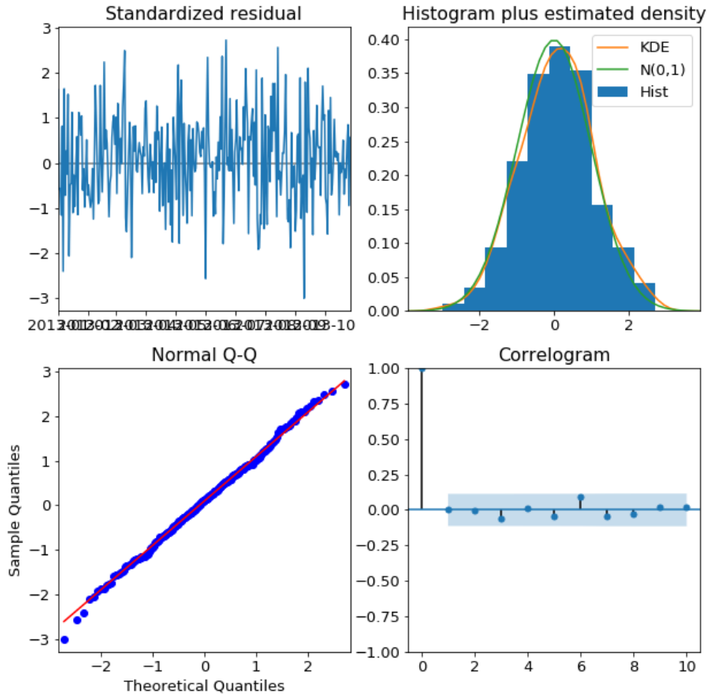
```
Fit ARIMA: order=(2, 0, 2) seasonal_order=(1, 1, 1, 12); AIC=nan, BIC=nan, Fit time=nan seconds
Fit ARIMA: order=(0, 0, 0) seasonal_order=(0, 1, 0, 12); AIC=2648.467, BIC=2656.490, Fit time=0.062 seconds
Fit ARIMA: order=(1, 0, 0) seasonal_order=(1, 1, 0, 12); AIC=2279.986, BIC=2296.031, Fit time=1.171 seconds

...

Fit ARIMA: order=(3, 0, 3) seasonal_order=(1, 1, 1, 12); AIC=2173.508, BIC=2213.621, Fit time=12.487 seconds
Fit ARIMA: order=(3, 0, 3) seasonal_order=(0, 1, 0, 12); AIC=2297.305, BIC=2329.395, Fit time=2.087 seconds
Total fit time: 245.812 seconds
```

pymarima results

```
print(results.summary())
```

```
                           Statespace Model Results
==============================================================================
Dep. Variable:            real values   No. Observations:                  300
Model:                SARIMAX(2, 0, 0)   Log Likelihood                -408.078
Date:                Tue, 28 May 2019   AIC                            822.156
Time:                        15:53:07   BIC                            833.267
Sample:                    01-01-2013   HQIC                           826.603
                         - 10-27-2013
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          0.2189      0.054      4.072      0.000       0.114       0.324
ar.L2          0.1960      0.054      3.626      0.000       0.090       0.302
sigma2         0.8888      0.073     12.160      0.000       0.746       1.032
===================================================================================
Ljung-Box (Q):                       32.10   Jarque-Bera (JB):                 0.02
Prob(Q):                              0.81   Prob(JB):                         0.99
Heteroskedasticity (H):               1.28   Skew:                            -0.02
Prob(H) (two-sided):                  0.21   Kurtosis:                         2.98
===================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
```

```
results.plot_diagnostics()
```



Non-seasonal search parameters

Non-seasonal search parameters

```
results = pm.auto_arima( df,              # data
                         d=0,             # non-seasonal difference order
                         start_p=1,       # initial guess for p
                         start_q=1,       # initial guess for q
                         max_p=3,         # max value of p to test
                         max_q=3,         # max value of q to test
                        )
```

[1] https://www.alkaline [2] ml.com/pmdarima/modules/generated/pmdarima.arima.auto_arima.html

Seasonal search parameters

```
results = pm.auto_arima( df,                # data
                         ... ,             # non-seasonal arguments
                         seasonal=True,    # is the time series seasonal
                         m=7,              # the seasonal period
                         D=1,              # seasonal difference order
                         start_P=1,        # initial guess for P
                         start_Q=1,        # initial guess for Q
                         max_P=2,          # max value of P to test
                         max_Q=2,          # max value of Q to test
                        )
```

Other parameters

```
results = pm.auto_arima( df,                          # data
                         ... ,                        # model order parameters
                         information_criterion='aic', # used to select best model
                         trace=True,                  # print results whilst training
                         error_action='ignore',       # ignore orders that don't work
                         stepwise=True,               # apply intelligent order search
                       )
```

Saving model objects

```
# Import
import joblib
```

```
# Select a filepath
filepath ='localpath/great_model.pkl'

# Save model to filepath
joblib.dump(model_results_object, filepath)
```

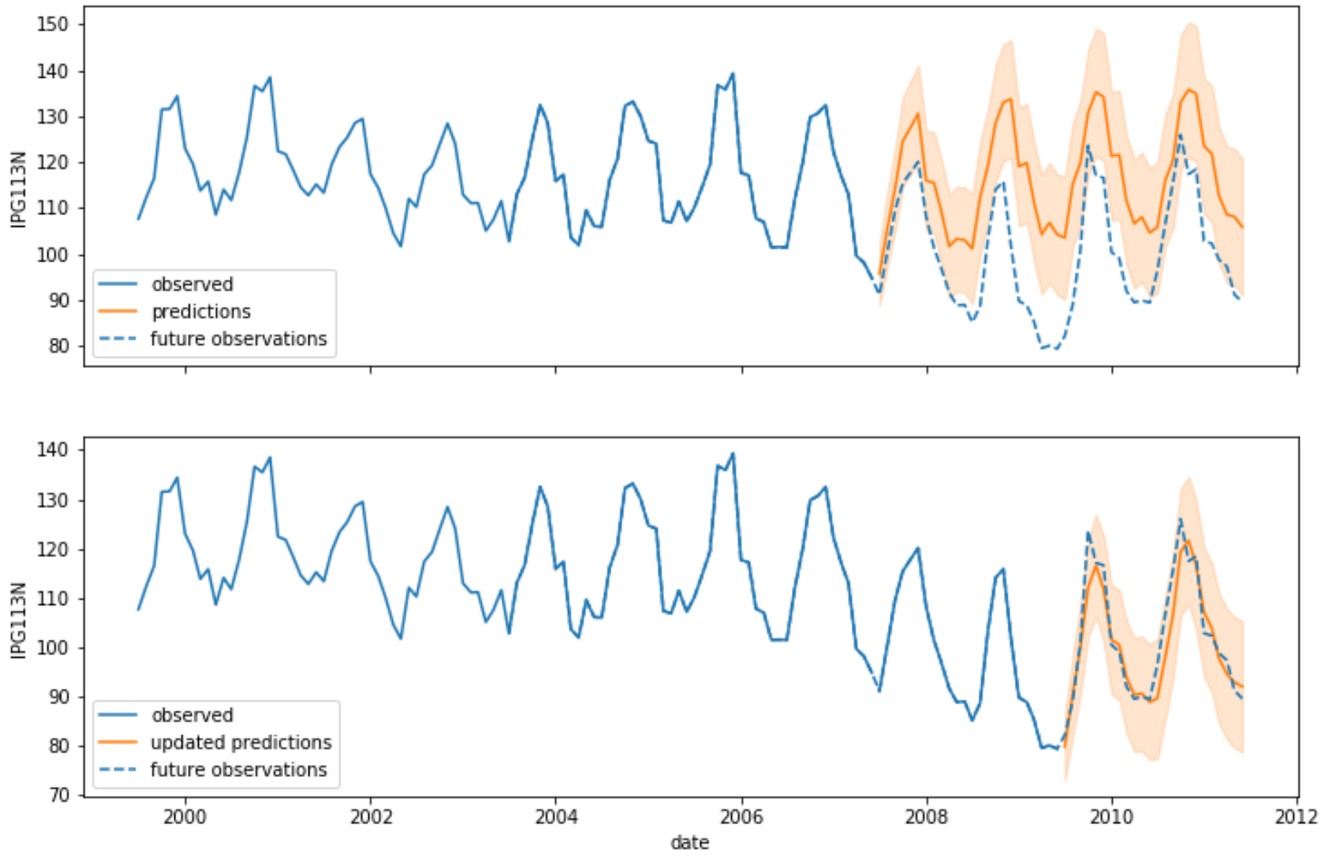
Saving model objects

```
# Select a filepath
filepath ='localpath/great_model.pkl'


# Load model object from filepath
model_results_object = joblib.load(filepath)
```

Updating model

```
# Add new observations and update parameters
model_results_object.update(df_new)
```

Update comparison



Let's practice!

ARIMA MODELS IN PYTHON

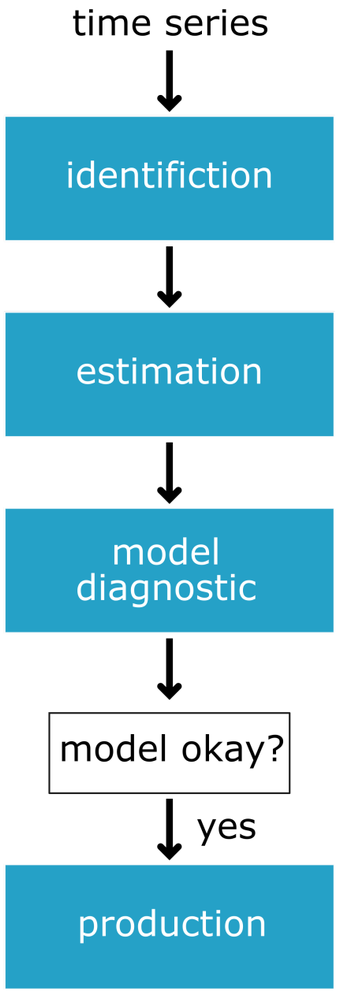
SARIMA and Box-Jenkins

ARIMA MODELS IN PYTHON

James Fulton
Climate informatics researcher

datacamp

Box-Jenkins

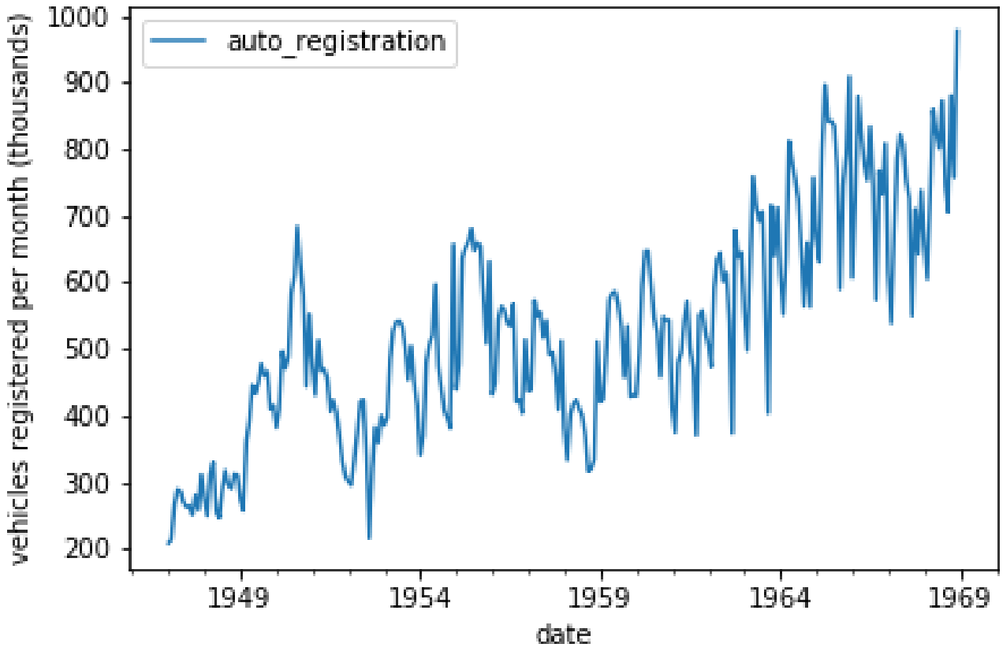

time series
identifiction
estimation
model diagnostic
model okay?
yes
production

Box-Jenkins with seasonal data

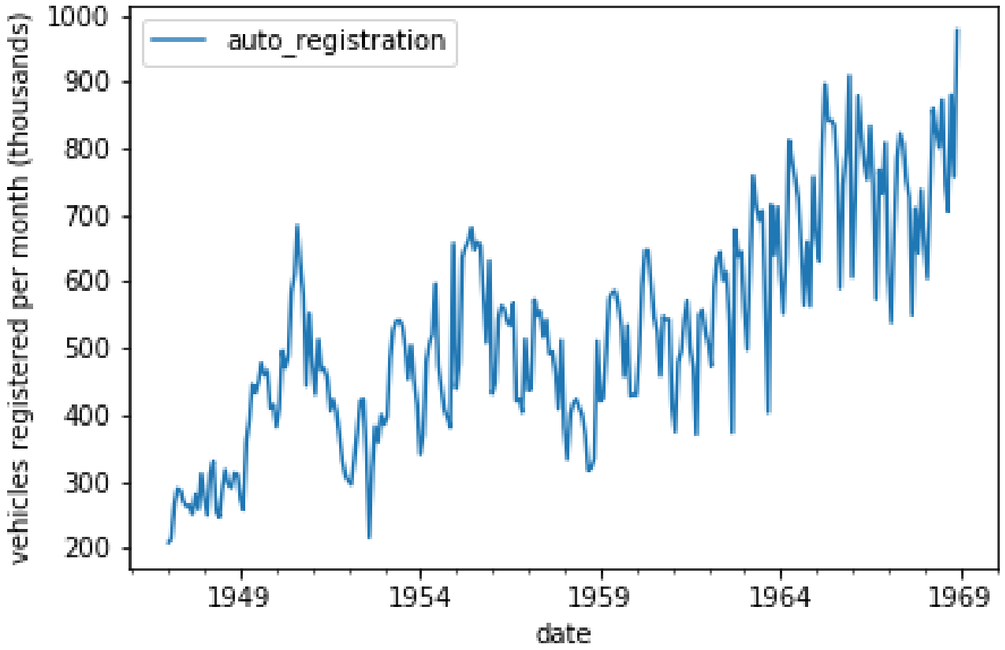
- Determine if time series is seasonal
- Find seasonal period
- Find transforms to make data stationary
  - Seasonal and non-seasonal differencing
  - Other transforms

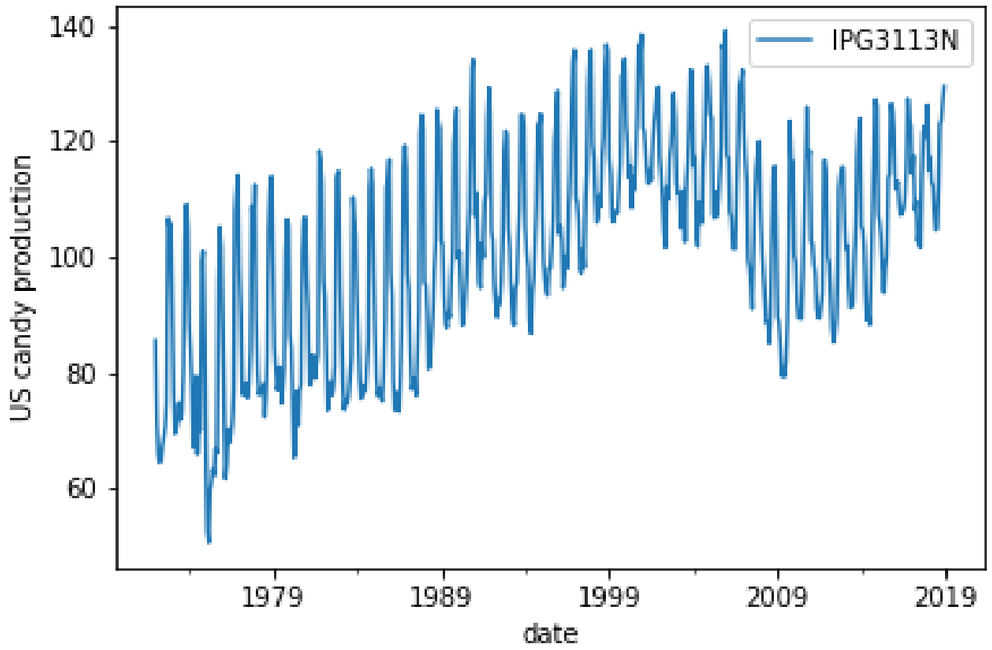
Mixed differencing

- `D` should be 0 or 1
- `d + D` should be 0-2



Weak vs strong seasonality

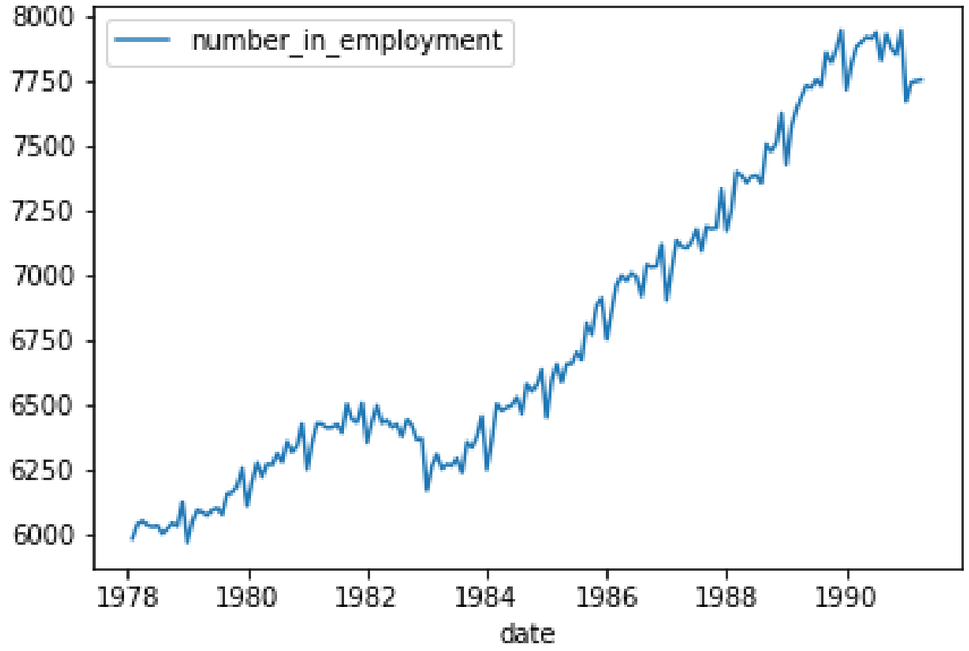


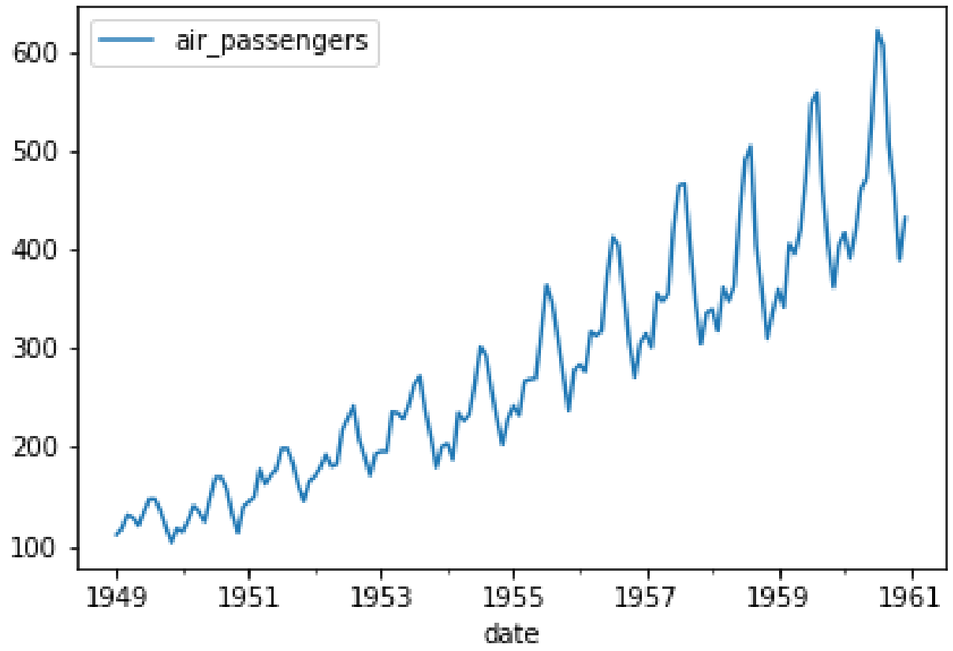


- Weak seasonal pattern
- Use seasonal differencing if necessary

- Strong seasonal pattern
- Always use seasonal differencing

Additive vs multiplicative seasonality

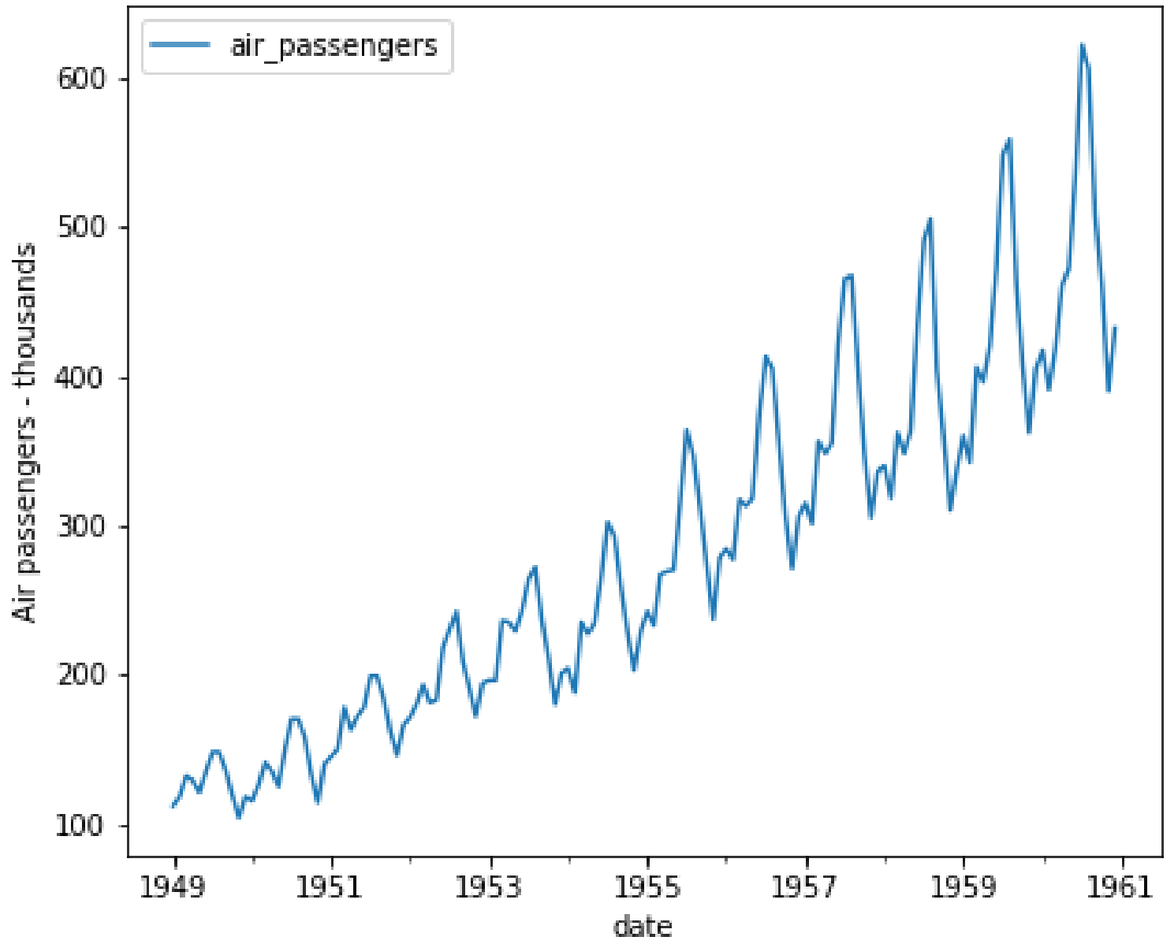


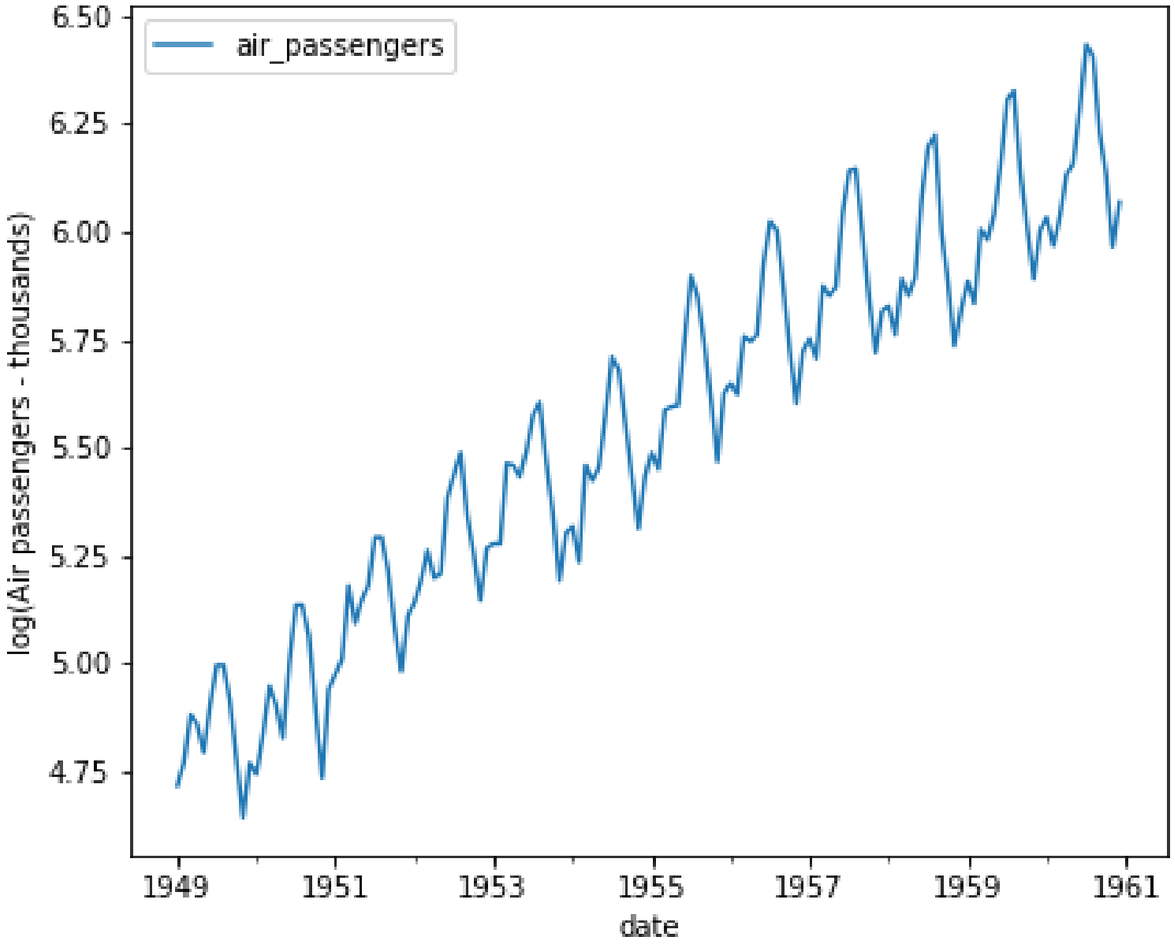


- Additive series = trend + season
- Proceed as usual with differencing

- multiplicative series = trend x season
- Apply log transform first - `np.log`

Multiplicative to additive seasonality





Let's practice!

ARIMA MODELS IN PYTHON

Congratulations!

ARIMA MODELS IN PYTHON

James Fulton
Climate informatics researcher

datacamp

The SARIMAX model

S - seasonal

AR - autoregressive

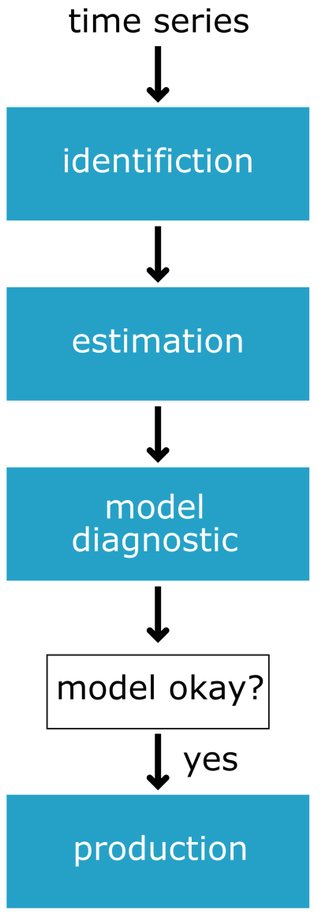
I - integrated

MA - moving average

X - exogenous

Time series modeling framework

- Test for stationarity and seasonality
- Find promising model orders
- Fit models and narrow selection with AIC/BIC
- Perform model diagnostics tests
- Make forecasts
- Save and update models



Further steps

- Fit data created using `arma_generate_sample()`
- Tackle real world data! Either your own or **examples from statsmodels**

Further steps

- Fit data created using `arma_generate_sample()`
- Tackle real world data! Either your own or **examples from statsmodels**
- More time series courses here

[1] https://www.statsmodels.org/stable/datasets/index.html

Good luck!

ARIMA MODELS IN PYTHON