# Défis en Intelligence Artificielle

# Défi 3 : L'IA pour l'analyse et la prévision de séries temporelles

Souhaib BEN TAIEB

UMONS
Université de Mons

Faculté
des Sciences

December 5, 2019

# Who are we?

- Lecturer
  - Souhaib BEN TAIEB
- Assistants
  - Bastien VANDERPLAETSE
  - Nico SALAMONE

# Schedule

- Week 1: 5 December 2019 (Room Mirzakhani and 404PC)
  - Introduction to statistical time series analysis and forecasting
- Week 2: 12 December 2019 (Room 404PC)
  - From time series forecasting to supervised learning (regression)
- Week 3: 19 December 2019 (Room 404PC)
  - Deep learning for time series forecasting

# Ressources and tools

- DataCamp:
  - `www.datacamp.com/enterprise/university-certificate-in-artificial-intellignece-hands-on-ai/leaderboard`
- Kaggle Kernels
  - `www.kaggle.com/docs/kernels`
- Other tools: Google Colab, etc.

# Assessment

- Two assignments: **20%**
- One project (group of two students): **80%**

| Task | Due Date | Value |
|---|---|---|
| Assignment 1 | 11 December 11:55pm | 10% |
| Assignment 2 | 18 December 11:55pm | 10% |
| Project | | 80% |
| → Predictive accuracy on Kaggle | 8 January 11:55pm | 30% |
| → Report | 22 January 11:55pm | 50% |

- Web Traffic Time Series Forecasting: Forecast future traffic to Wikipedia pages - `https://www.kaggle.com/c/web-traffic-time-series-forecasting`
- Project report
  - Discuss the three different forecasting approaches: (1) time series models, (2) from forecasting to supervised learning and (3) deep neural networks
  - More details next time

# Communication

- Moodle
  - `https://moodle.umons.ac.be/course/view.php?id=2666`
  - Forum for asking questions, etc.
  - Assignment submissions
- Course webpage (GitHub)
  - `https://github.com/bsouhaib/Hands-On-AI-2019`
- **No email please — use the forum**

# Contents

# Part I

# Time Series Data

# Time series data

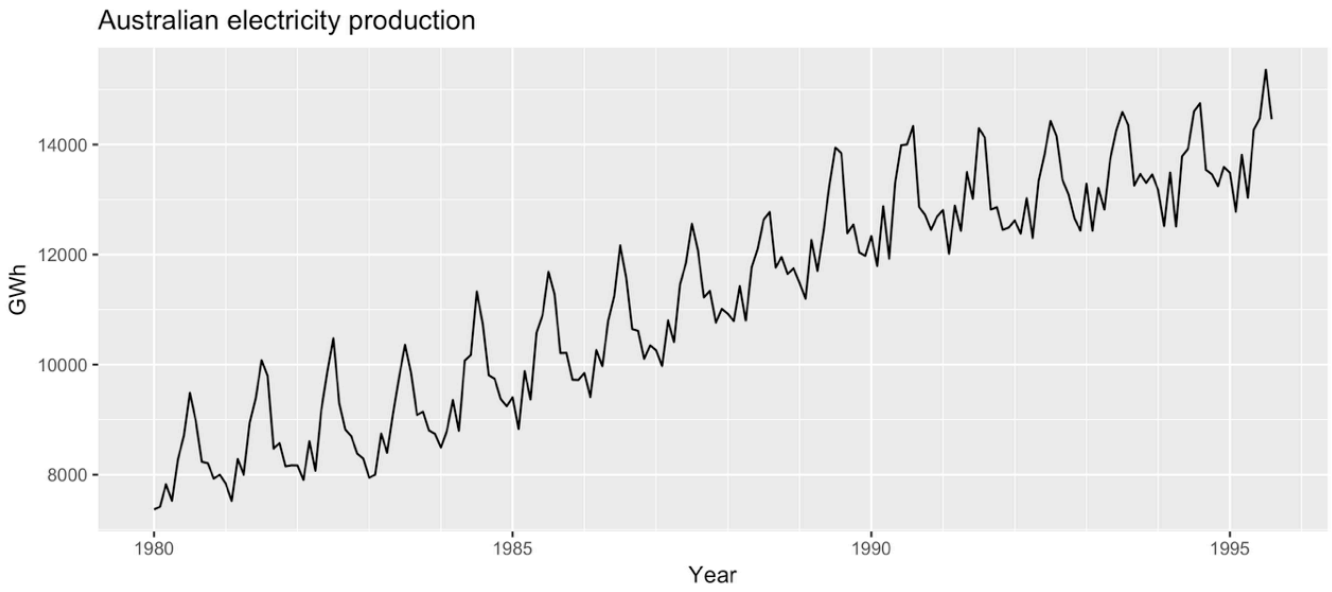
- Series of data observed over time
- Eg.: Daily IBM stock prices, monthly rainfall in London,...


**Forecasting is estimating how the sequence of observations will continue into the future.**

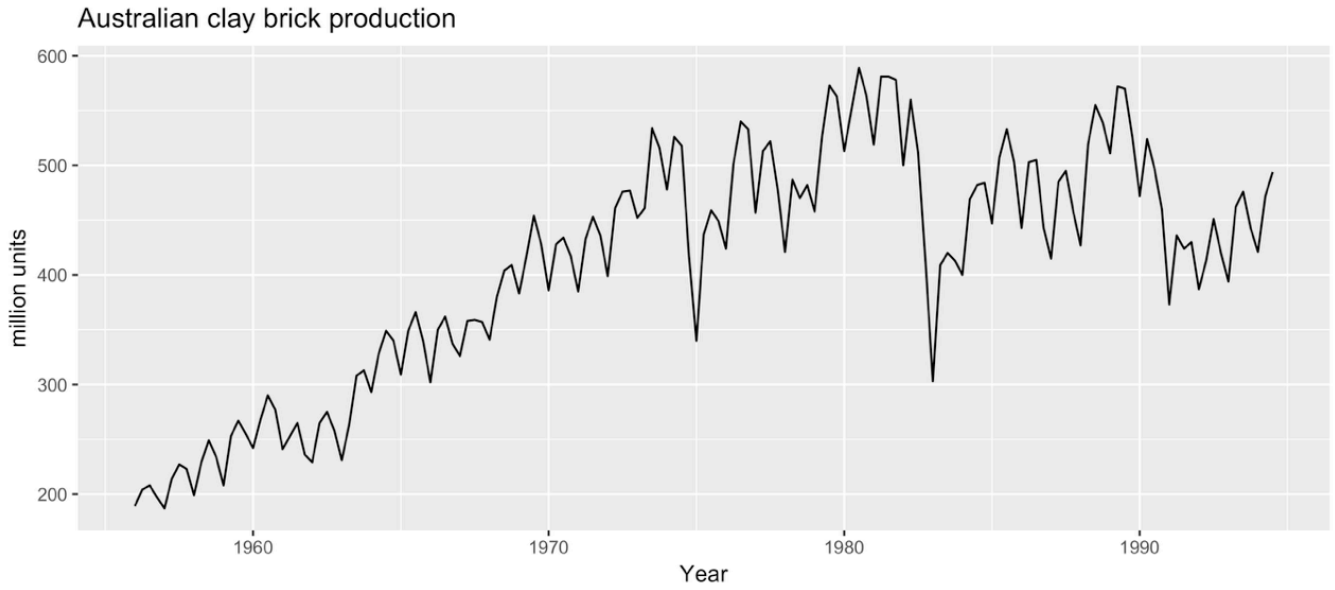
# Time series patterns

| Pattern | Description |
|---|---|
| **Trend** | A pattern exists involving a long-term increase OR decrease in the data |
| **Seasonal** | A periodic pattern exists due to the calendar (e.g. the quarter, month, or day of the week) |
| **Cyclic** | A pattern exists where the data exhibits rises and falls that are *not of fixed period* (duration usually of at least 2 years) |

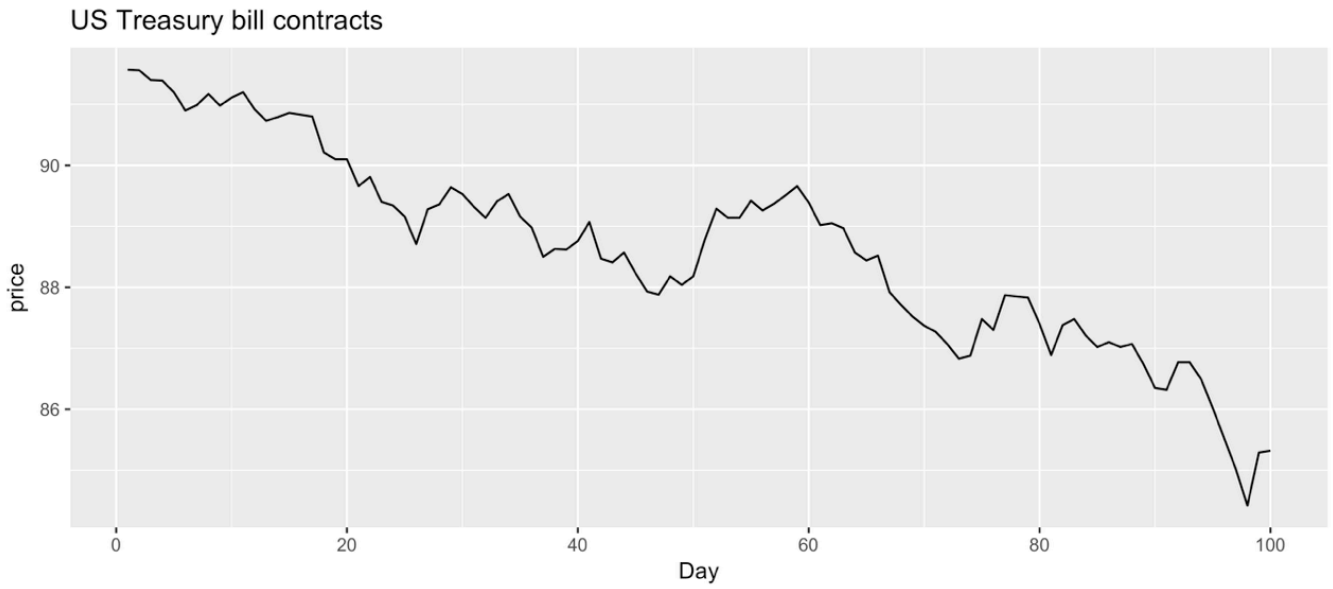
# Examples of time series patterns



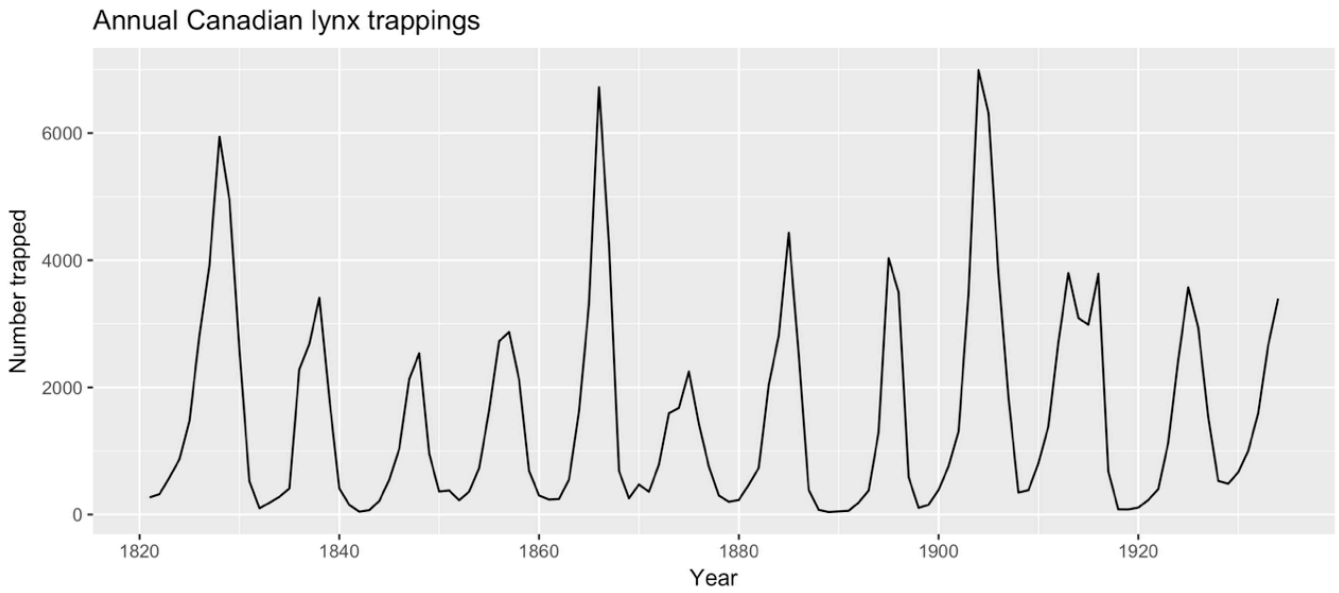
# Examples of time series patterns



# Examples of time series patterns



# Examples of time series patterns



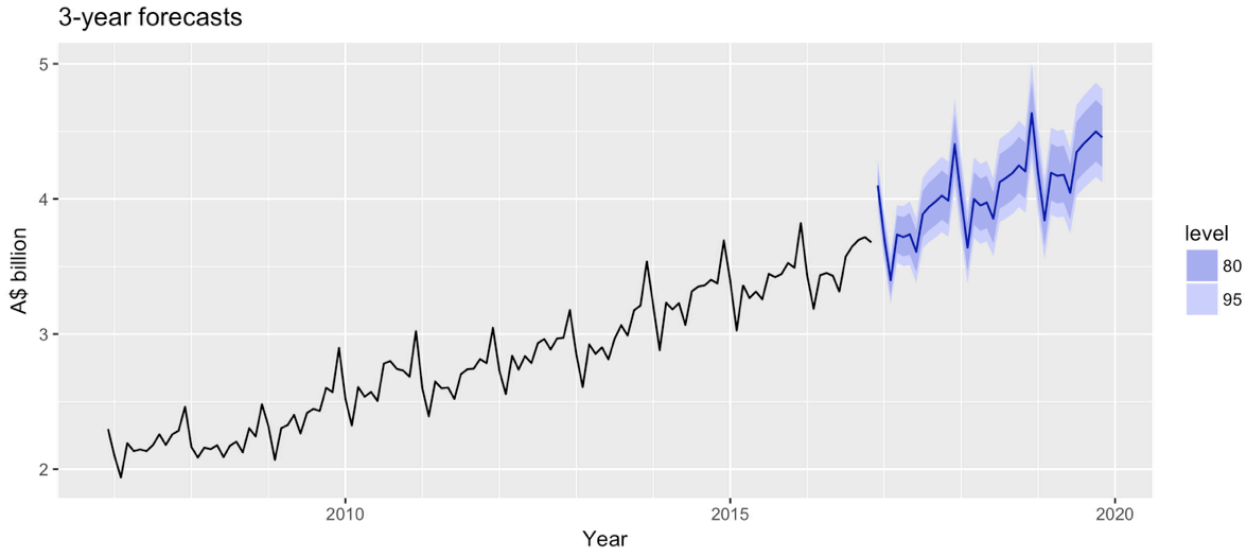
# Seasonal or cyclic?

**Differences between seasonal and cyclic patterns:**

- Seasonal pattern constant length vs. cyclic pattern variable length
- Average length of cycle longer than length of seasonal pattern
- Magnitude of cycle more variable than magnitude of seasonal pattern

**The timing of peaks and troughs is predictable with seasonal data, but unpredictable in the long term with cyclic data.**

# Forecasts of monthly Australian expenditure on eating out



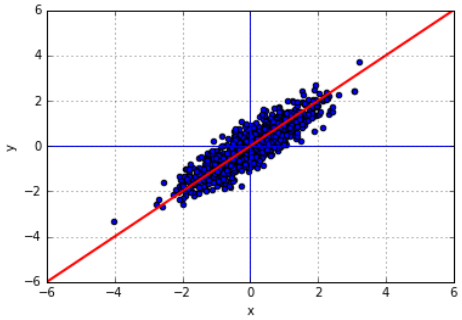

- What forecasting methods are available that take account of trend, seasonality and other features of the data?
- How to measure the accuracy of your forecasts?
- How to choose a good forecasting model?

# Part II

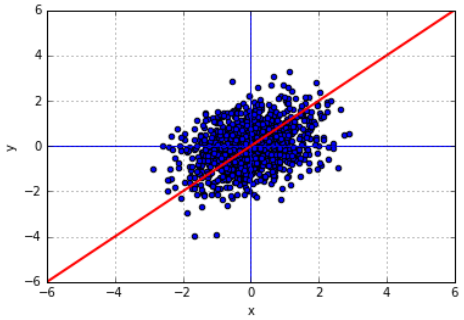
# Autocorrelation

# Correlation

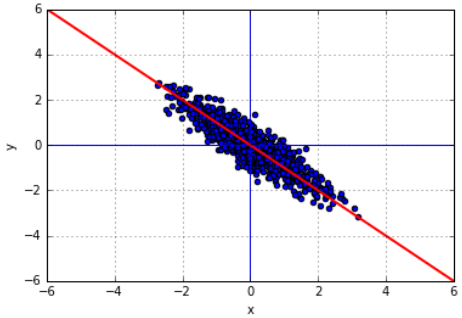
- Correlation = 0.9



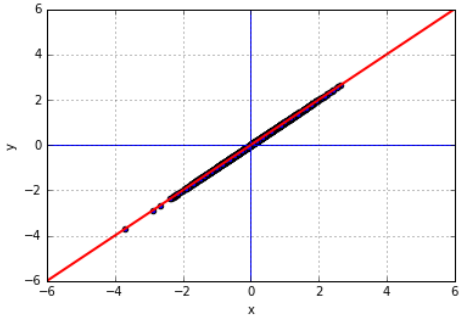

- Correlation = 0.4




- Correlation = -0.9




- Corelation = 1.0



# What is Autocorrelation?

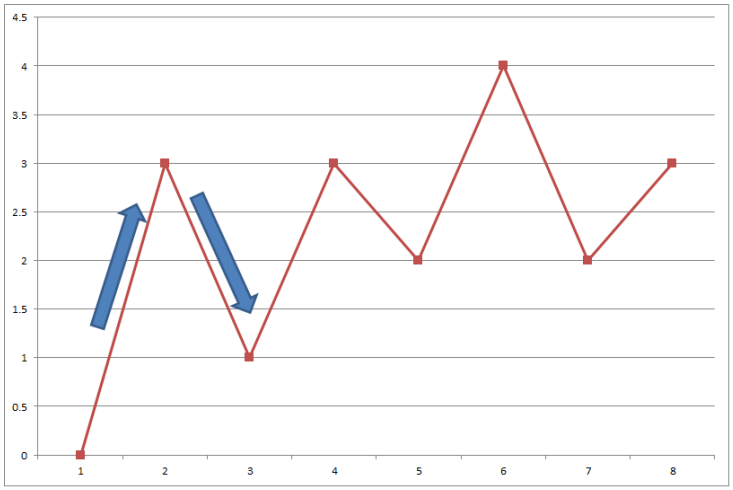
- Correlation of a time series with a lagged copy of itself

| Series | Lagged Series |
|---|---|
| 5 | |
| 10 | 5 |
| 15 | 10 |
| 20 | 15 |
| 25 | 20 |
| ⋮ | ⋮ |

- **Lag-one** autocorrelation
- Also called **serial correlation**

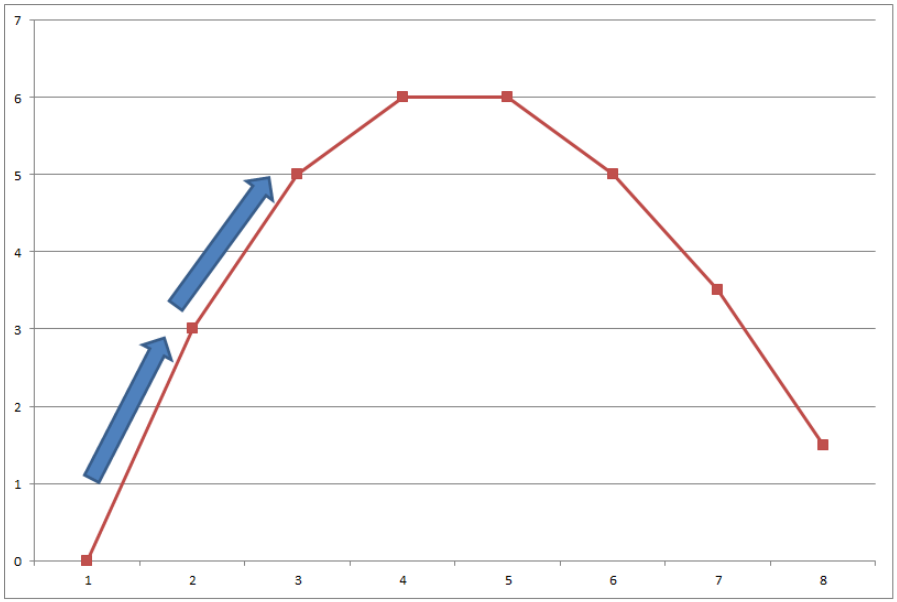
# Interpretation of Autocorrelation

- **Mean Reversion** - Negative autocorrelation



# Interpretation of Autocorrelation

- **Momentum**, or **Trend Following** - Positive autocorrelation



# Autocorrelation at lag $k$

Given a time series $y_1, y_2, \ldots, y_T$, the (sample) autocorrelation at lag $k$ is given by

$$r_k = \frac{\sum_{t=k+1}^{T} (y_t - \bar{y})(y_{t-k} - \bar{y})}{\sum_{t=1}^{T} (y_t - \bar{y})^2},$$

where

$$\bar{y} = \frac{1}{T} \sum_{t=1}^{T} y_t,$$

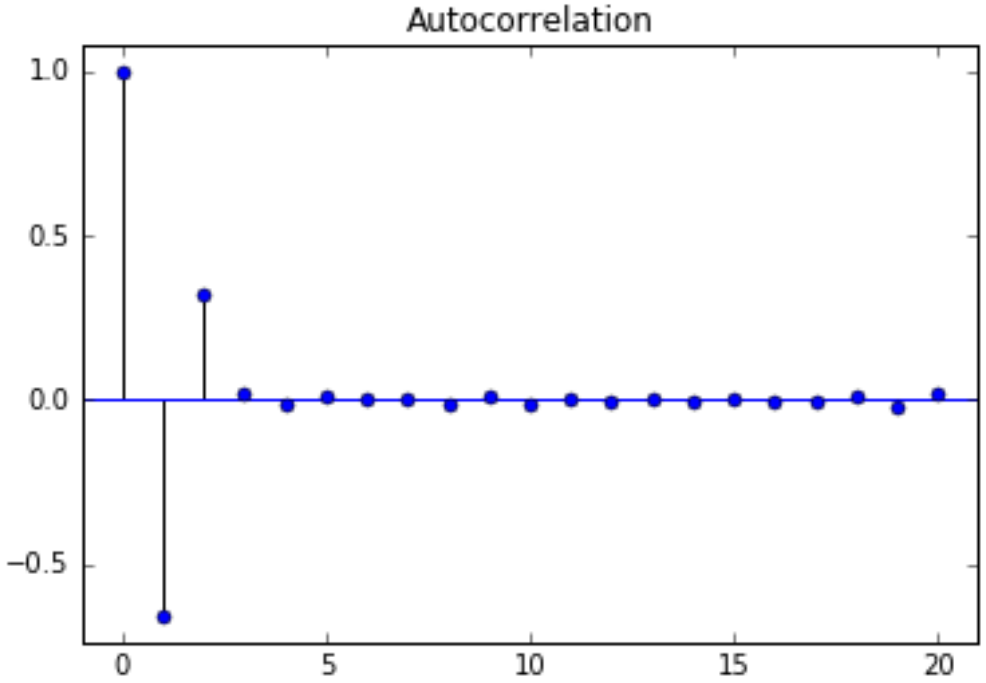
and $0 \leq k < T$.

# Autocorrelation Function

- Autocorrelation Function (ACF): The autocorrelation as a function of the lag
- Equals one at lag-zero
- Interesting information beyond lag-one

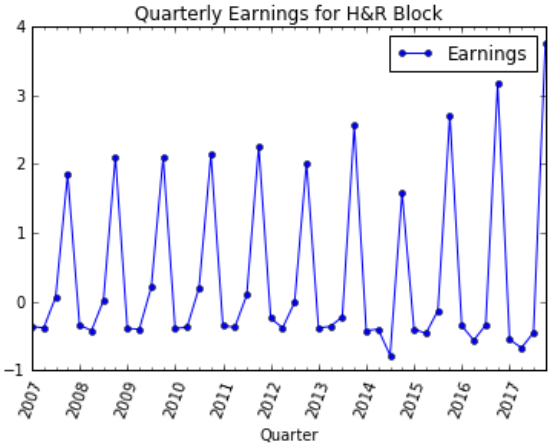
# ACF Example 1: Simple Autocorrelation Function

- Can use last two values in series for forecasting

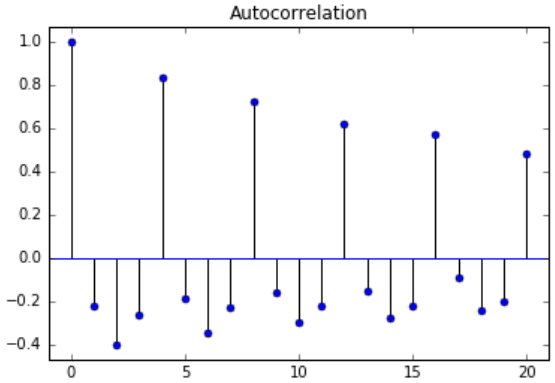


# ACF Example 2: Seasonal Earnings

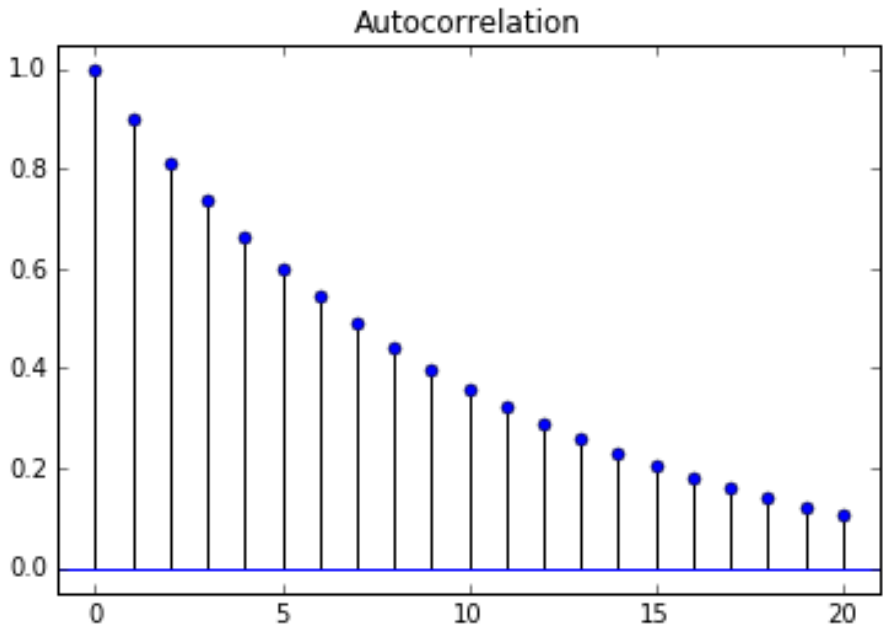
- Earnings for H&R Block




- ACF for H&R Block



# ACF Example 3: Useful for Model Selection




- Model selection

# Plot ACF in Python

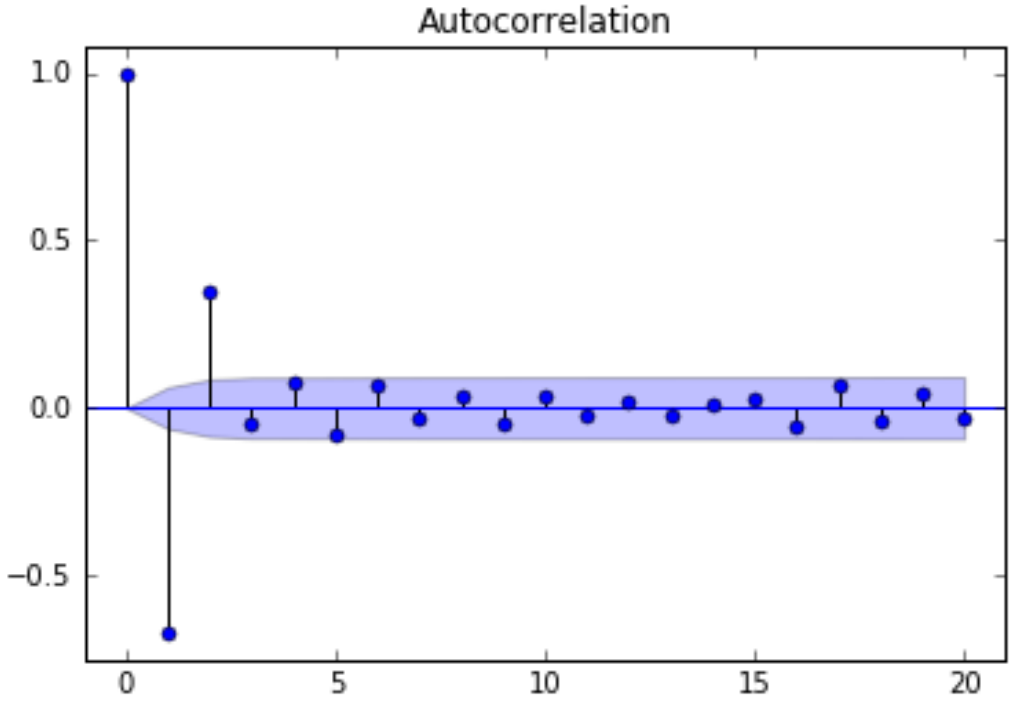
- Import module:

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

- Plot the ACF:

```
plot_acf(x, lags= 20, alpha=0.05)
```

# Confidence Interval of ACF



# Confidence Interval of ACF

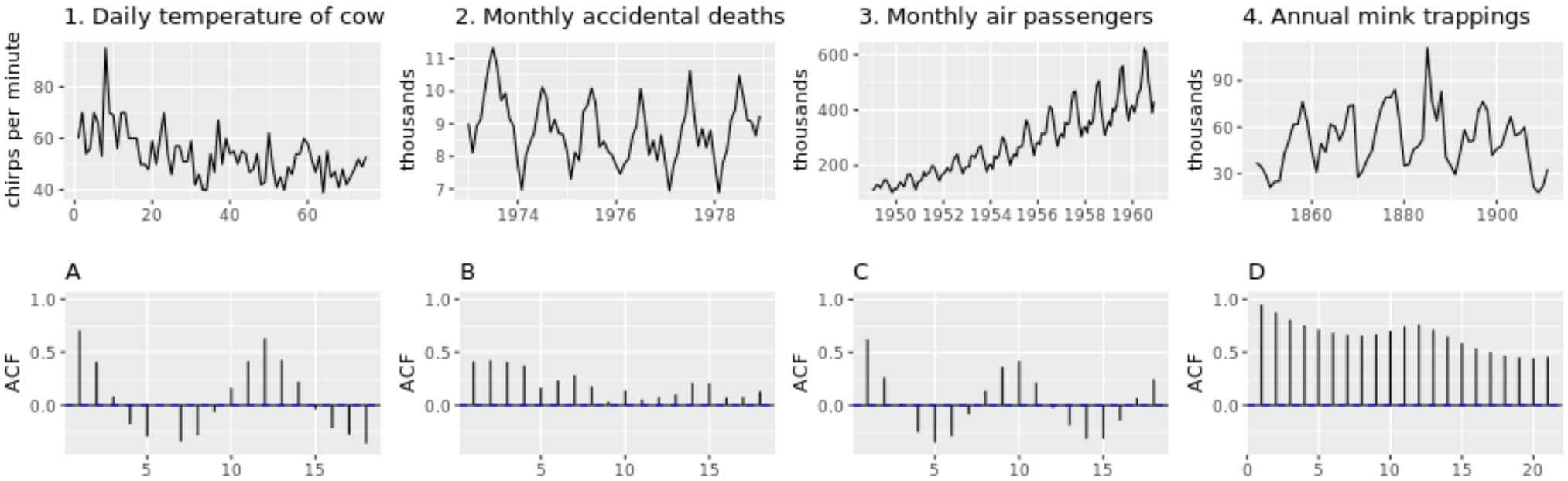
- Argument `alpha` sets the width of confidence interval
- Example: `alpha=0.05`
  - 5% chance that if true autocorrelation is zero, it will fall outside blue band
- Confidence bands are wider if:
  - Alpha lower
  - Fewer observations
- Under some simplifying assumptions, 95% confidence bands are $\pm 2/\sqrt{N}$
- If you want no bands on plot, set `alpha=1`

# ACF Values Instead of Plot

```
from statsmodels.tsa.stattools import acf
print(acf(x))
```

```
[ 1.         -0.6765505   0.34989905 -0.01629415 -0.0250701
 -0.03186545  0.01399904 -0.03518128  0.02063168 -0.0262064
...
  0.07191516 -0.12211912  0.14514481 -0.09644228  0.0521588
```

# Match the ACF to the time series




Match the ACF plots shown (A-D) to their corresponding time plots (1-4).

- Use the process of elimination - pair the easy ones first, then see what is left.
- Trends induce positive correlations in the early lags.
- Seasonality will induce peaks at the seasonal lags.
- Cyclicity induces peaks at the average cycle length.

## Part III

# White Noise & Random Walk

# What is White Noise?

- White Noise is a series with:
  - Constant mean
  - Constant variance
  - Zero autocorrelations at all lags
- Special Case: if data has normal distribution, then *Gaussian White Noise*

***—> White Noise is a series that is purely random***

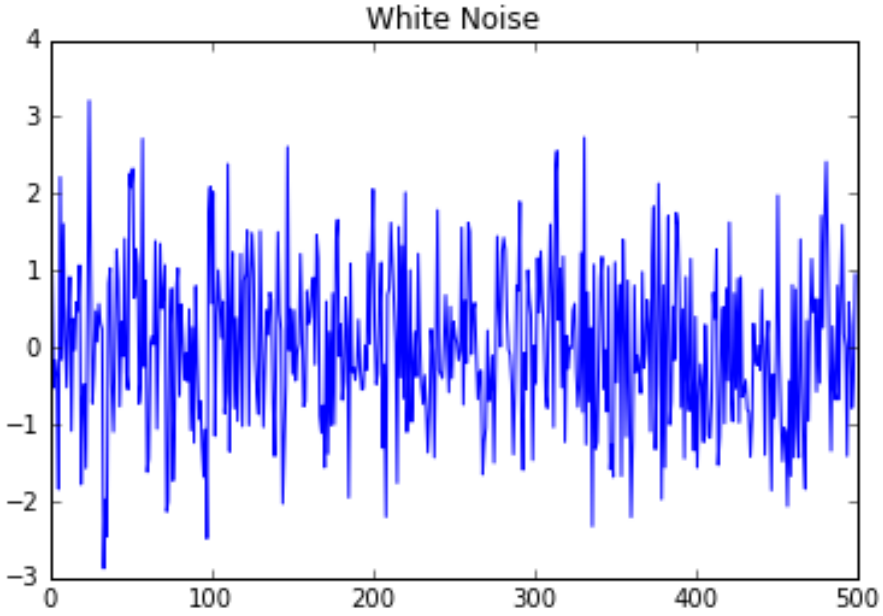
# Simulating White Noise

- It's very easy to generate white noise

```
import numpy as np
noise = np.random.normal(loc=0, scale=1, size=500)
```

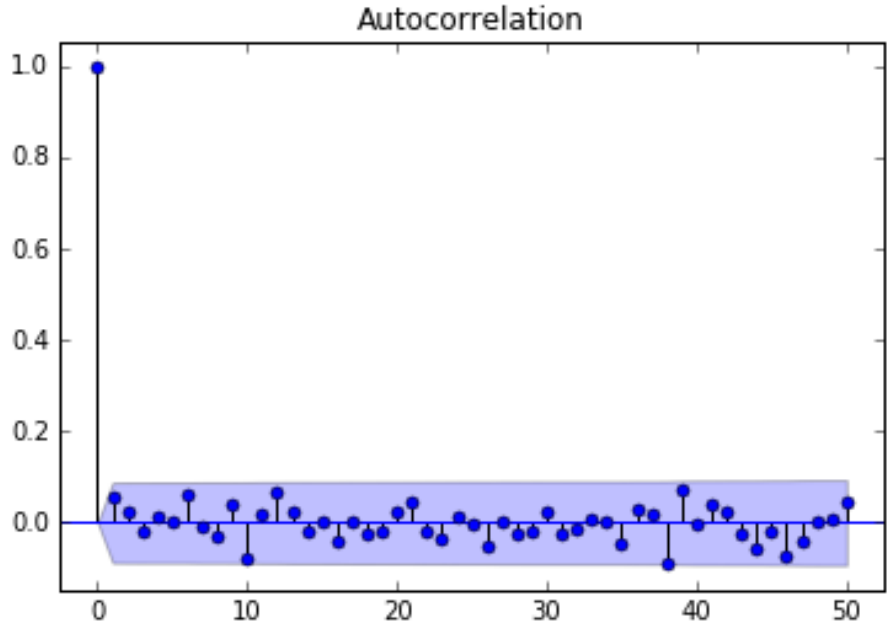
# What Does White Noise Look Like?

```
plt.plot(noise)
```



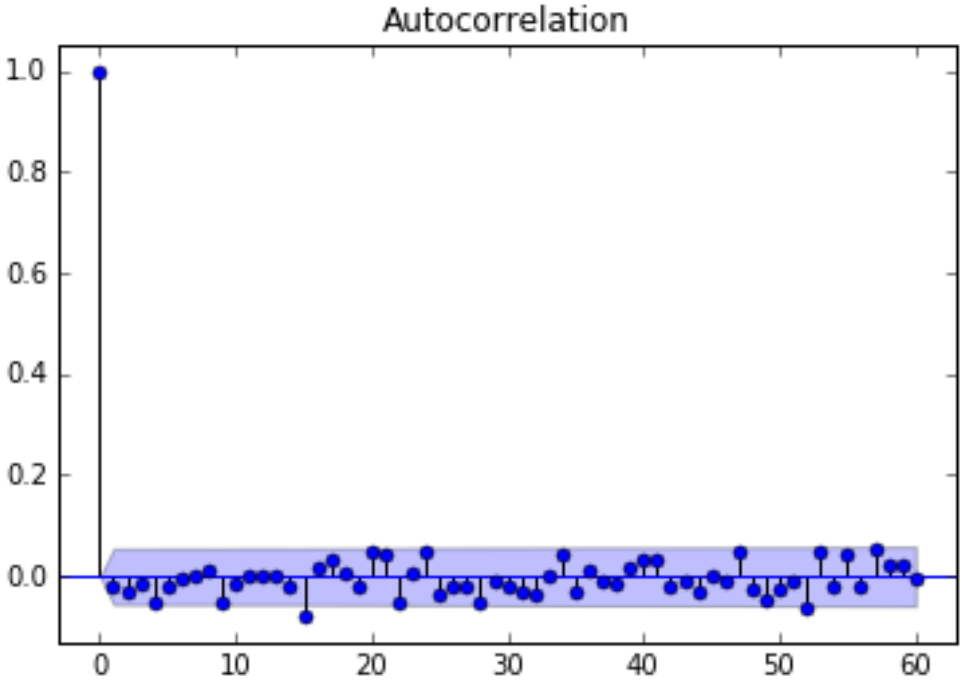
# Autocorrelation of White Noise

```
plot_acf(noise, lags=50)
```



# Stock Market Returns: Close to White Noise

- Autocorrelation Function for the S&P500

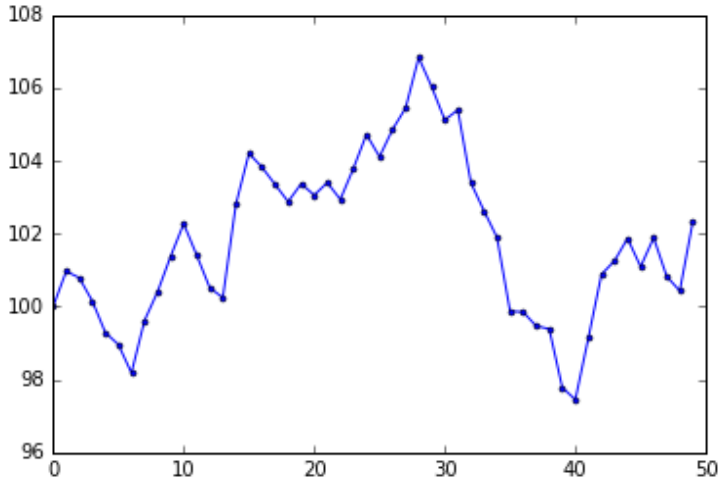


# What is a Random Walk?

- Today's Price = Yesterday's Price + Noise

$$P_t = P_{t-1} + \epsilon_t$$

- Plot of simulated data



# What is a Random Walk?

- Today's Price = Yesterday's Price + Noise

$$P_t = P_{t-1} + \epsilon_t$$

  - Change in price is white noise

$$P_t - P_{t-1} = \epsilon_t$$

  - Can't forecast a random walk
  - Best forecast for tomorrow's price is today's price

# Part IV

# Stationarity

# What is Stationarity?

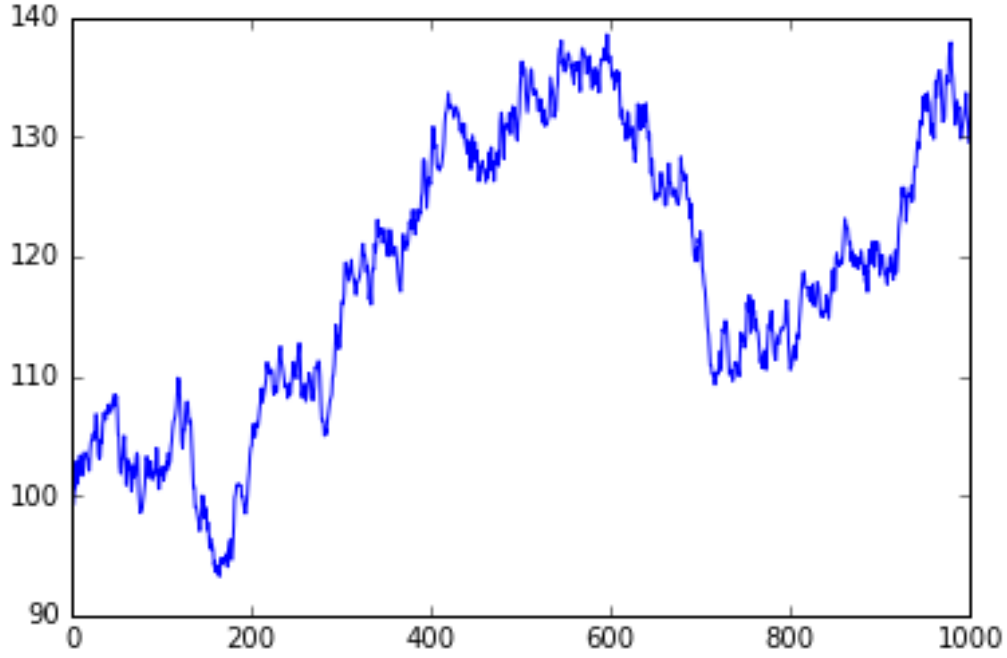
- **Strong stationarity**: entire distribution of data is time-invariant
- **Weak stationarity**: mean, variance and autocorrelation are time- invariant (i.e., for autocorrelation, corr$\left(X_t, X_{t-\tau}\right)$ is only a function of $\tau$)

# Why Do We Care?

- If parameters vary with time, too many parameters to estimate
- Can only estimate a parsimonious model with a few parameters

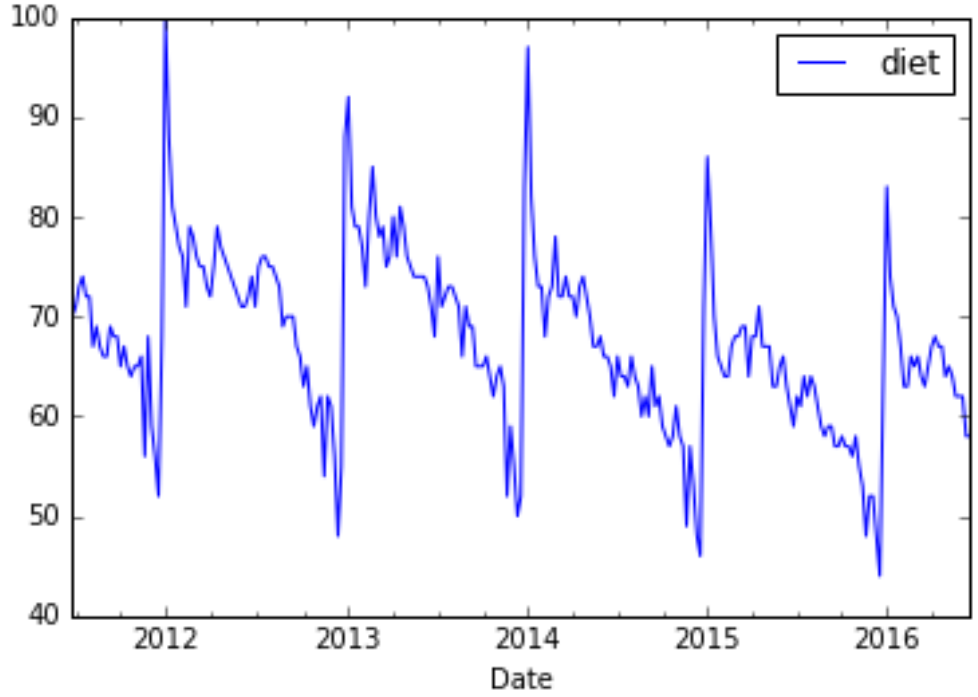
# Examples of Nonstationary Series

- Random Walk



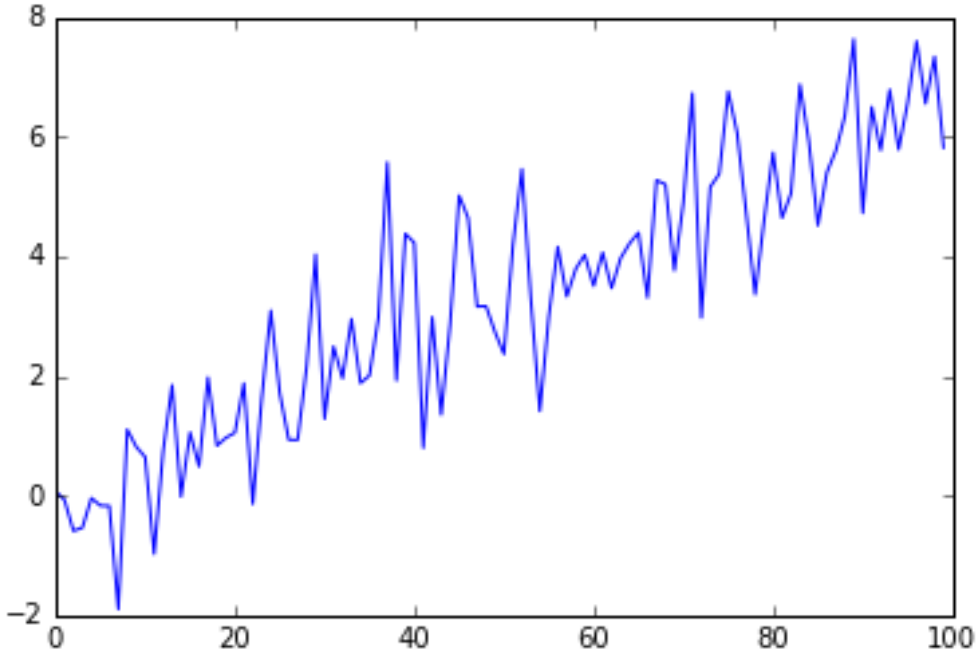
# Examples of Nonstationary Series

- Seasonality in series

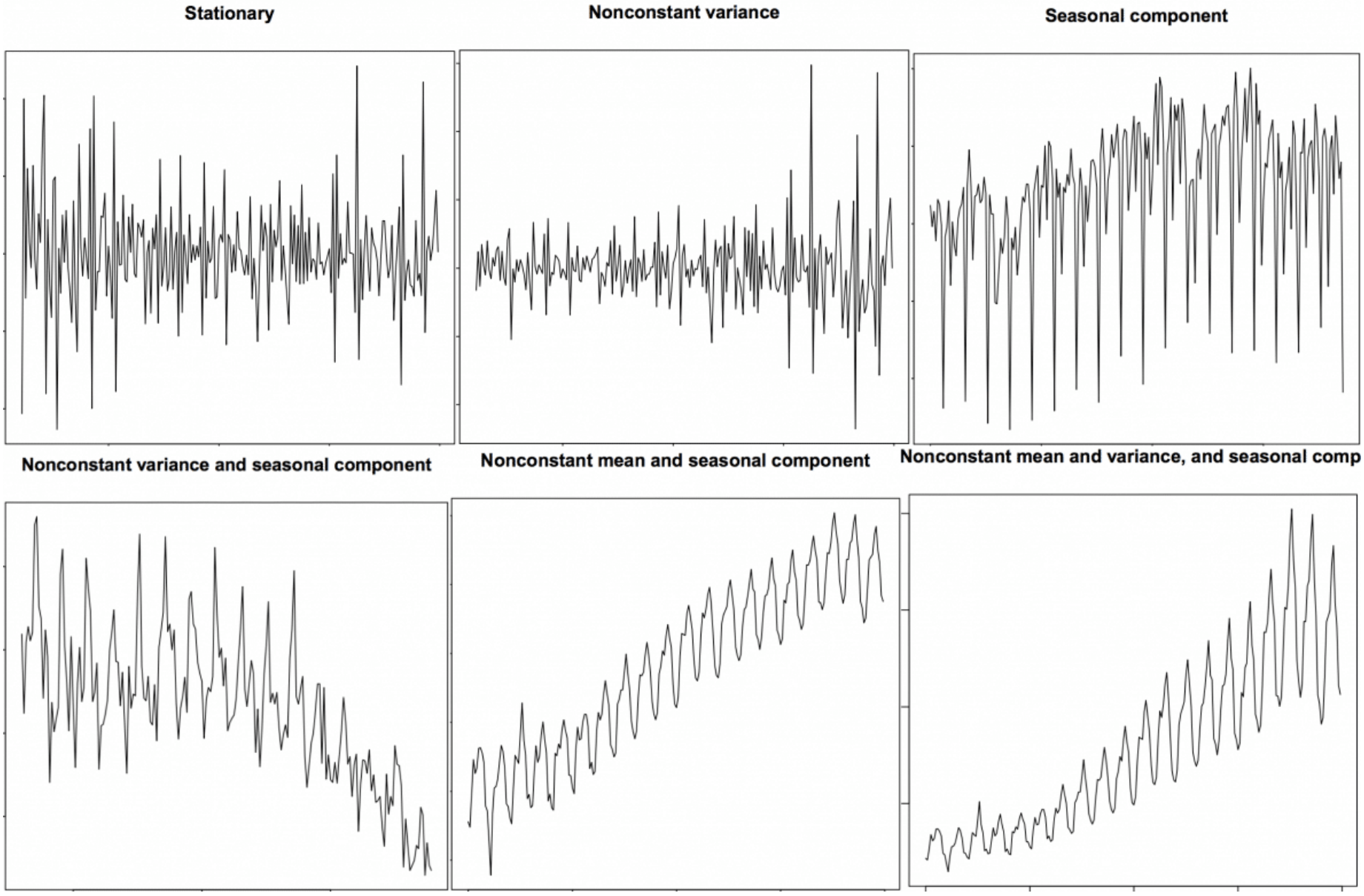


# Examples of Nonstationary Series

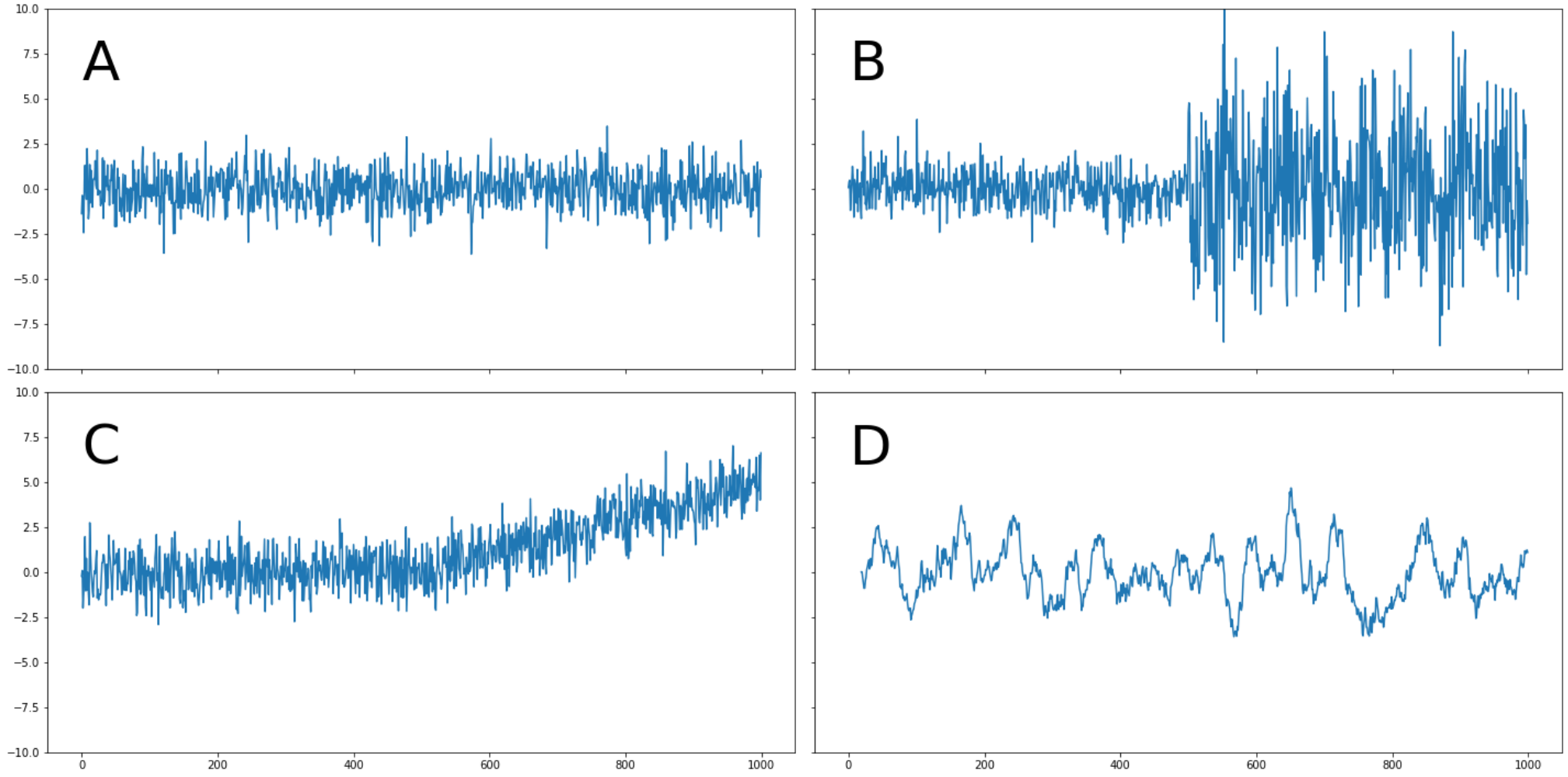
- Change in Mean or Standard Deviation over time



# Stationarity



# Stationarity

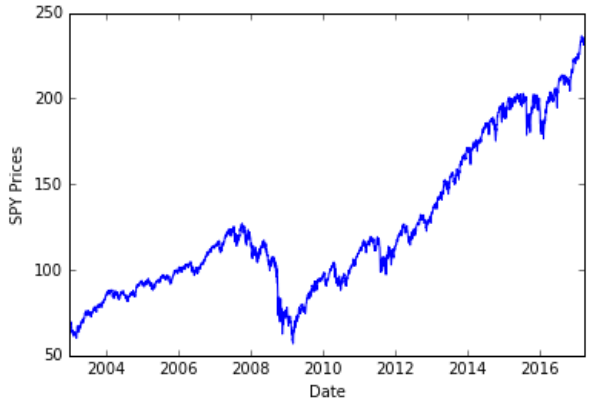



Which of the following time series do you think are not stationary?

# Transforming Nonstationary Series Into Stationary Series

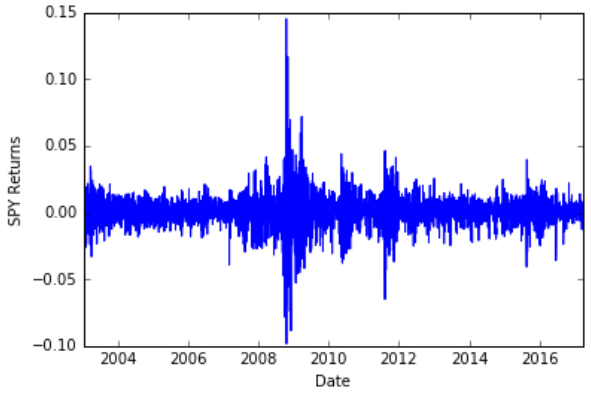
- Random Walk

```
plot.plot(SPY)
```




- First difference

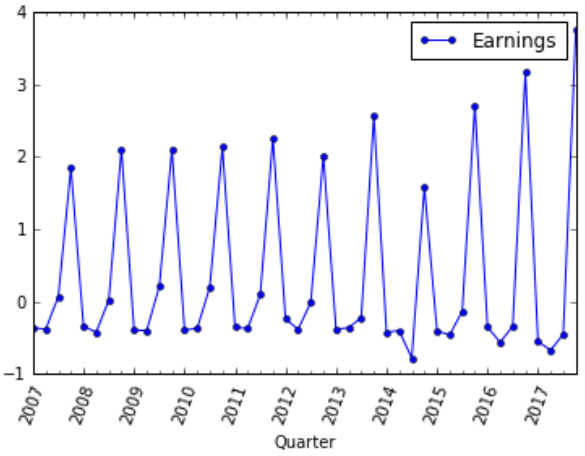
```
plot.plot(SPY.diff())
```



# Transforming Nonstationary Series Into Stationary Series

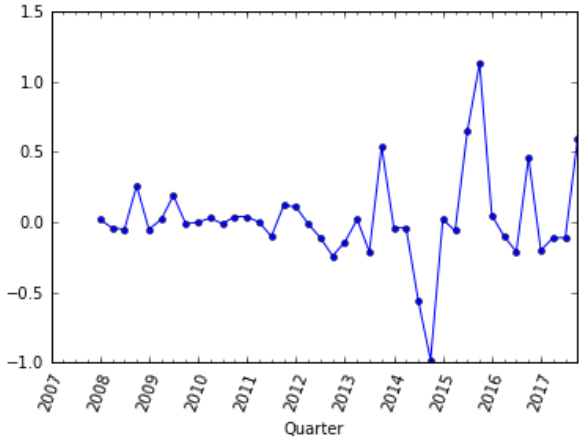
- Seasonality

```
plot.plot(HRB)
```




- Seasonal difference

```
plot.plot(HRB.diff(4))
```



# Part V

# Transformations

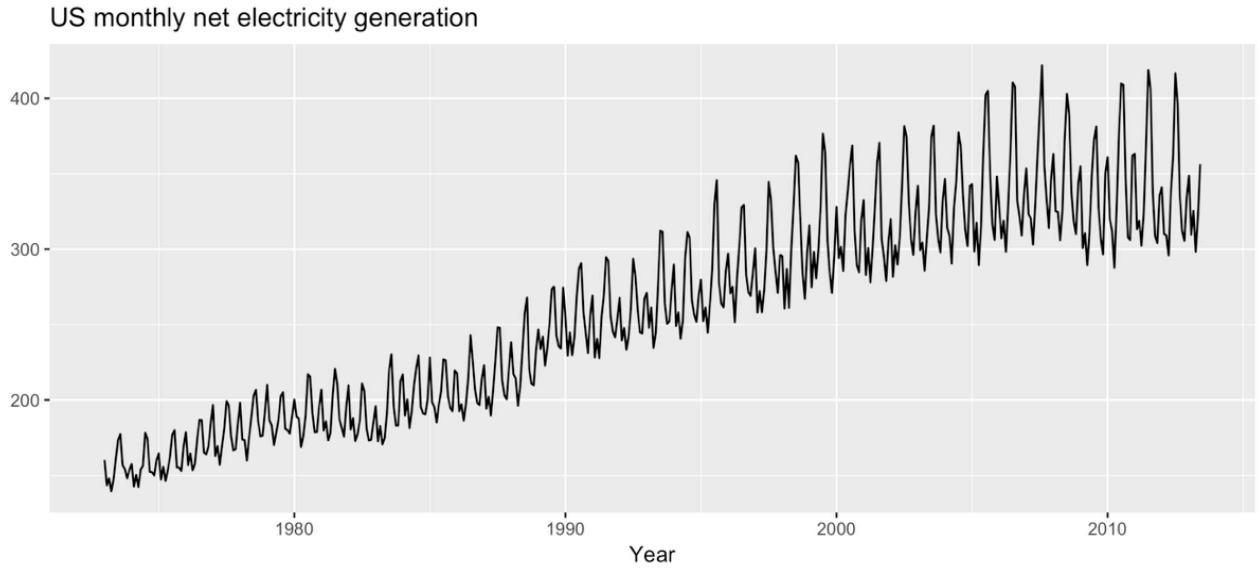
# Variance stabilization

- If the data show increasing variation as the level of the series increases, then a **transformation** can be useful
- $y_1, \dots, y_n$: original observations, $w_1, \dots, w_n$: transformed observations

| Mathematical transformations for stabilizing variation | | |
|---|---|---|
| Square Root | $w_t = \sqrt{y_t}$ | $\downarrow$ |
| Cube Root | $w_t = \sqrt[3]{y_t}$ | Increasing |
| Logarithm | $w_t = \log(y_t)$ | strength |
| Inverse | $w_t = -1/y_t$ | $\downarrow$ |

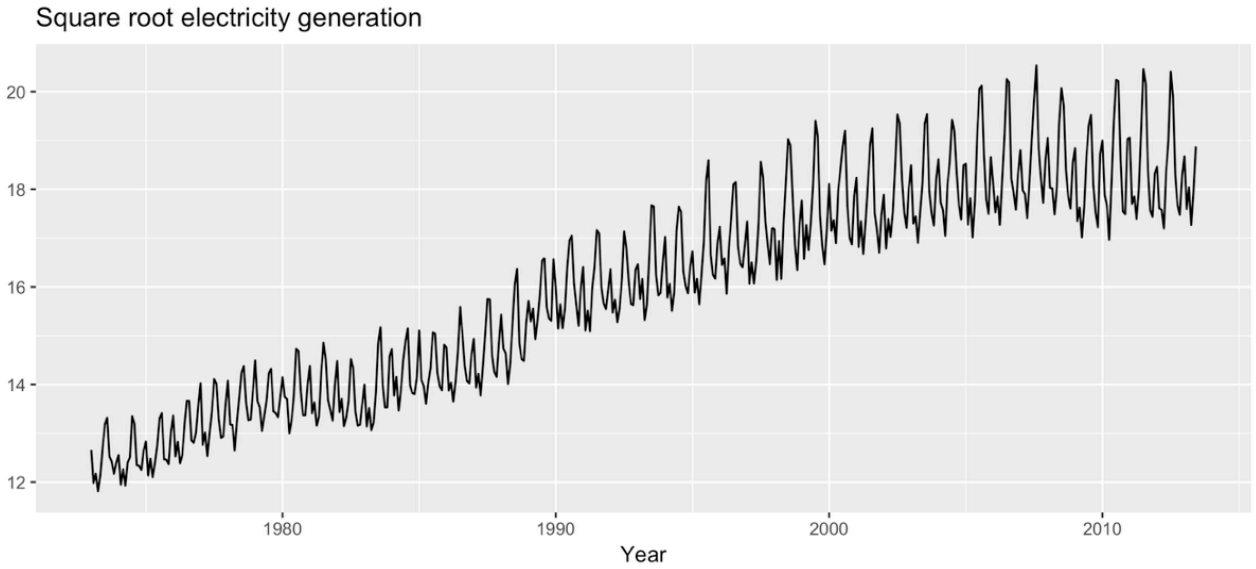
# Variance stabilization

```
> plot.plot(usmelec)
```



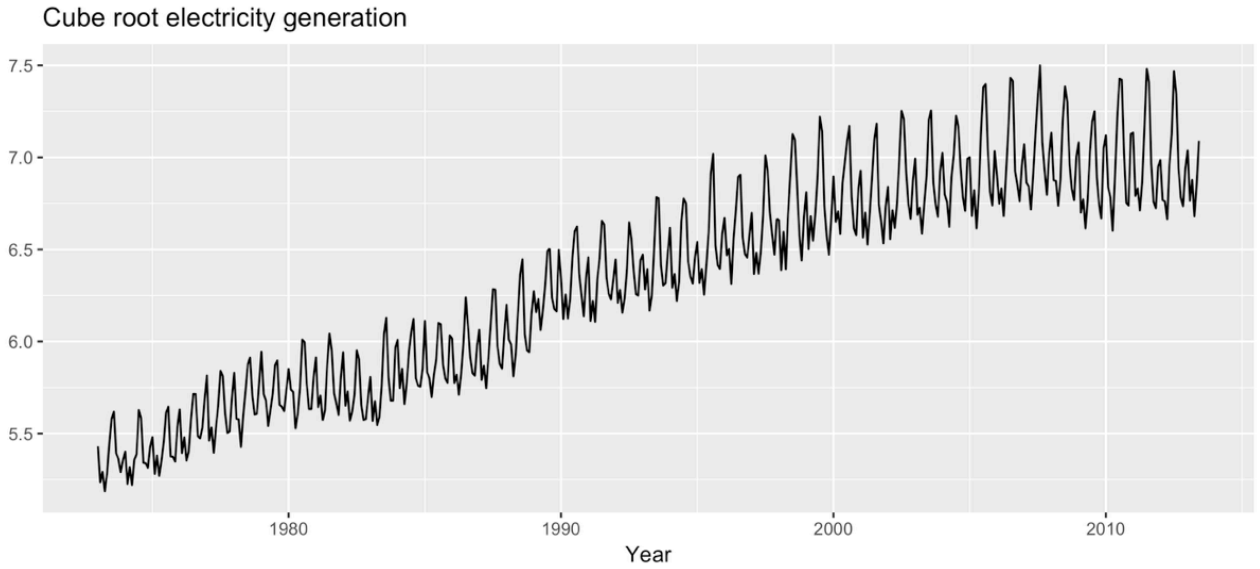
# Variance stabilization

```
> plot.plot(usmelec ** 0.5)
```



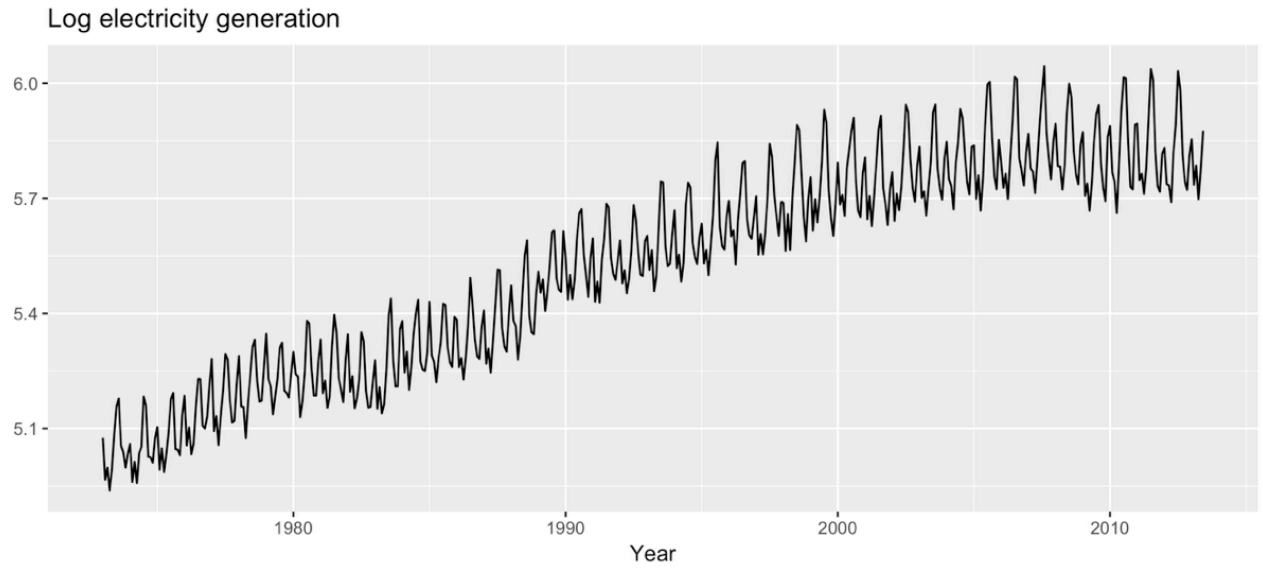
# Variance stabilization

```
> plot.plot(usmelec ** 0.333)
```



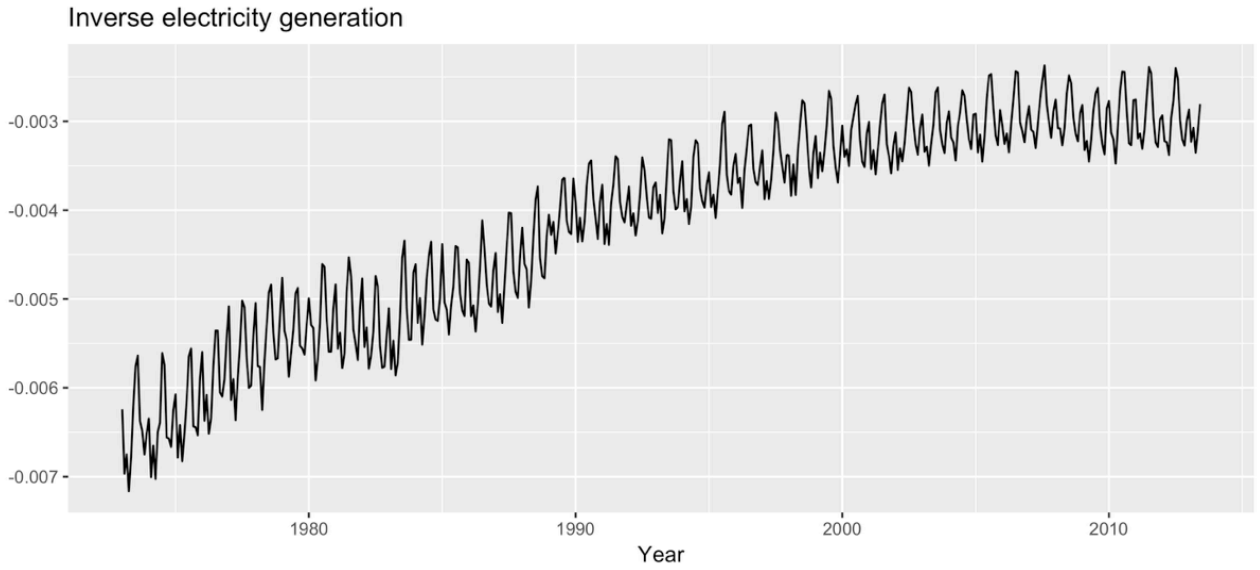
# Variance stabilization

```
> plot.plot(np.log(usmelec))
```



# Variance stabilization

```
> plot.plot(-1 / usmelec)
```



# Box-Cox transformations

- Each of these transformations is close to a member of the family of Box-Cox transformations

$$w_t = \begin{cases} \log(y_t) & \lambda = 0 \\ (y_t^{\lambda} - 1)/\lambda & \lambda \neq 0 \end{cases}$$

- $\lambda = 1$ : No substantive transformation
- $\lambda = \frac{1}{2}$ : Square root plus linear transformation
- $\lambda = \frac{1}{3}$ : Cube root plus linear transformation
- $\lambda = 0$ : Natural logarithm transformation
- $\lambda = -1$ : Inverse transformation

> See scipy.stats.boxcox

# Part VI

# Statistical tests

# Statistical tests

- Serial correlation tests
  - **Ljung-Box test** considers the first $h$ autocorrelation values together.
  - A significant test (small p-value) indicates the data are probably not white noise.
  - See *statsmodels.stats.diagnostic.acorr_ljungbox*
- Unit root tests (stationarity tests)
  - Statistical tests to determine the required order of differencing
  - **Augmented Dickey–Fuller** test: null hypothesis is that the data are non-stationary and non-seasonal.
  - A significant test (small p-value) indicates the data are probably stationary.
  - They do not detect nonstationarity of the seasonal kind. Other tests available for seasonal data.
  - See *statsmodels.tsa.stattools.adfuller*

# Part VII

# Time Series Decomposition

# Time series decomposition

- Three types of time series patterns: trend, seasonality and cycles.
- When we decompose a time series into components, we usually combine the trend and cycle into a single **trend-cycle** component (sometimes called the trend for simplicity).
- Thus we think of a time series as comprising three components: a **trend-cycle** component, a **seasonal** component, and a **remainder** component (containing anything else in the time series)

# Time series decomposition

If we assume an additive decomposition, then we can write

$$y_t = S_t + T_t + R_t,$$

where $y_t$ is the data, $S_t$ is the seasonal component, $T_t$ is the trend-cycle component, and $R_t$ is the remainder component, all at period $t$. Alternatively, a multiplicative decomposition would be written as

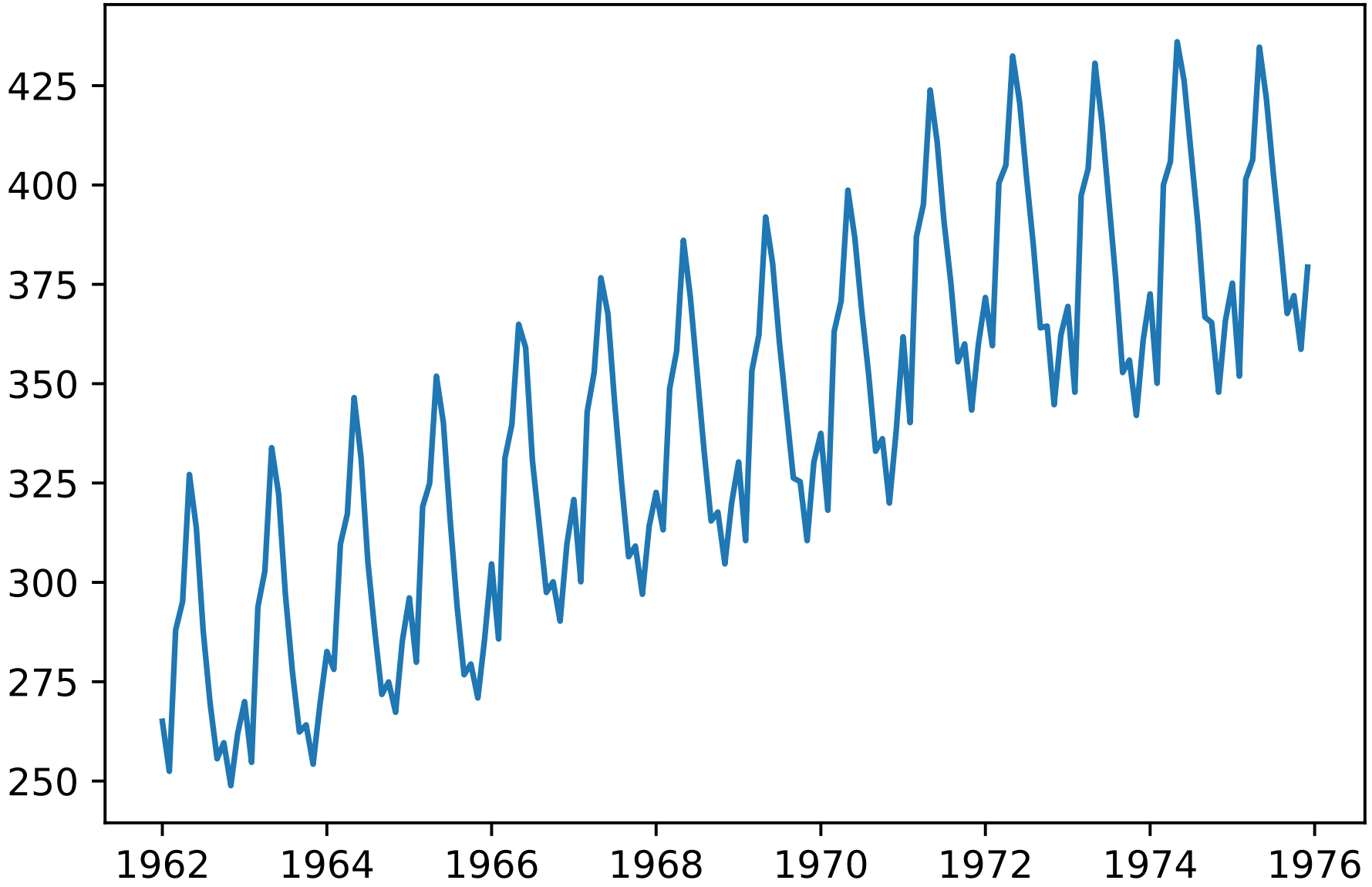
$$y_t = S_t \times T_t \times R_t.$$

Note that

$$y_t = S_t \times T_t \times R_t \quad \text{is equivalent to} \quad \log y_t = \log S_t + \log T_t + \log R_t.$$

In other words, when a log transformation has been used, an additive decomposition is equivalent to using a multiplicative decomposition.

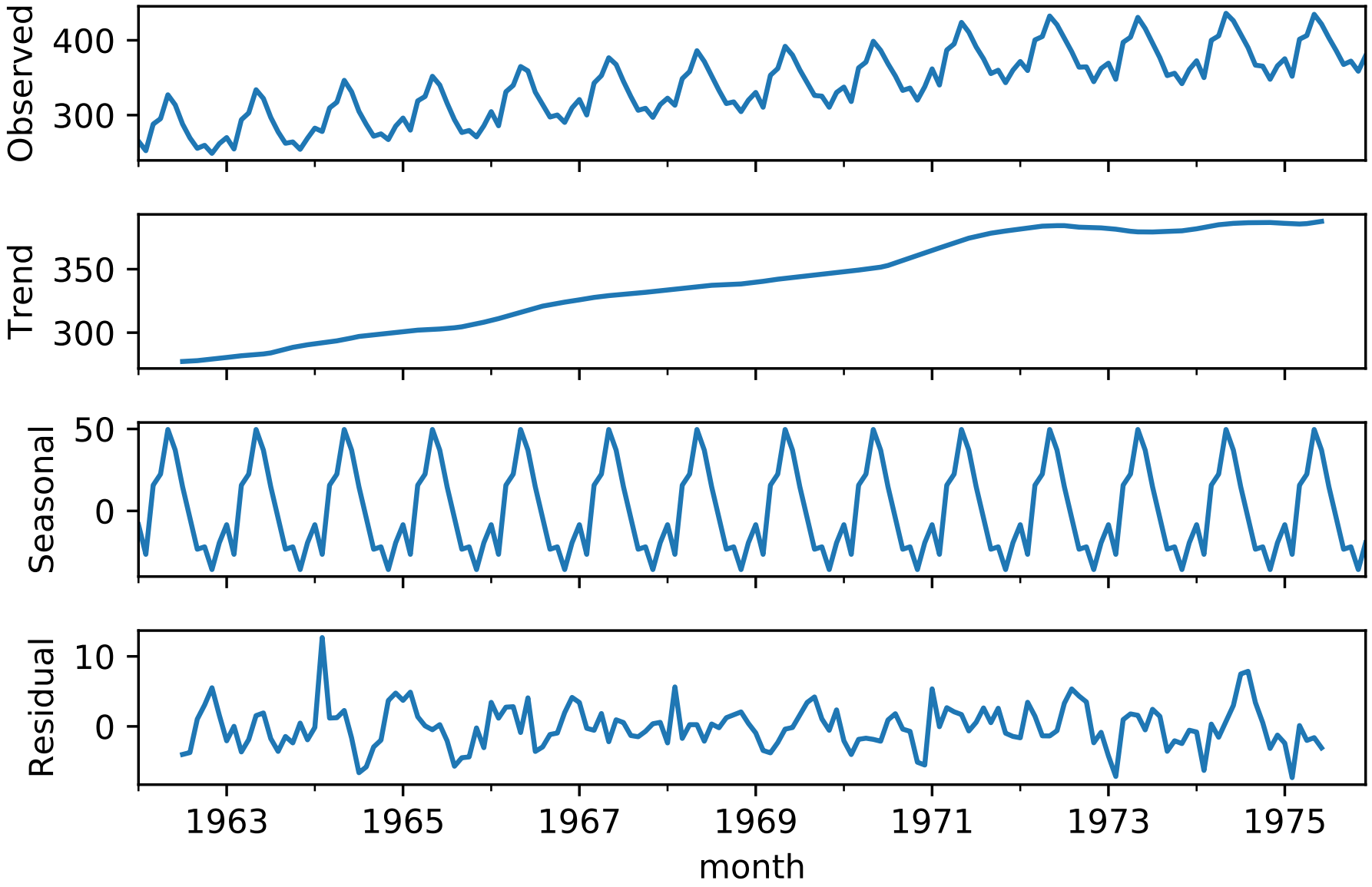
# Time series decomposition - example



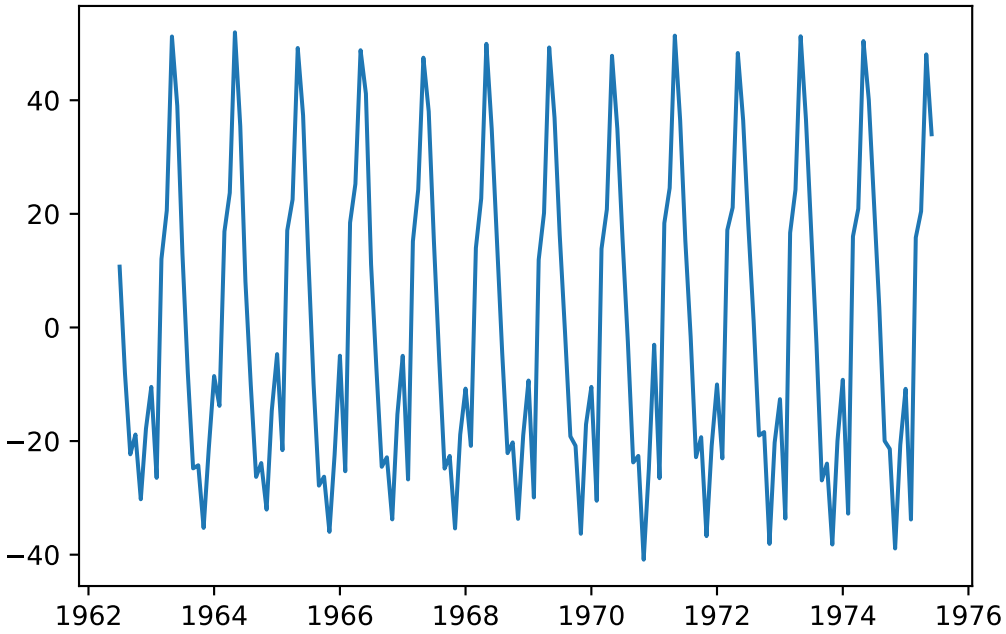
# Time series decomposition - example

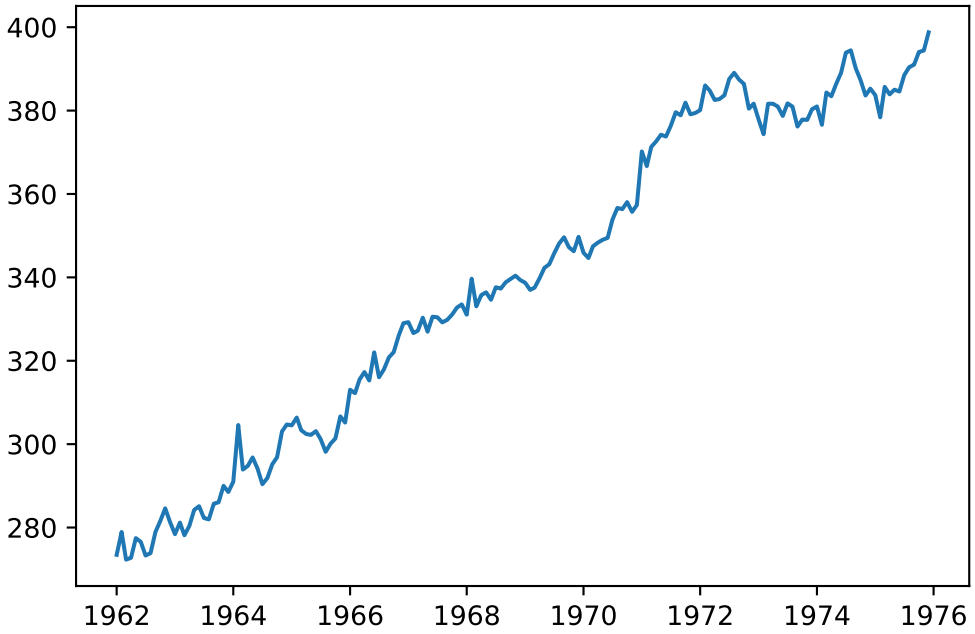
```
1  import pandas as pd
2  import matplotlib.pyplot as plt
3  import statsmodels.api as sm
4
5  data = pd.read_csv("./monthly_milk.csv")
6  data = data.set_index("month")
7  data.index = pd.to_datetime(data.index, format = "%Y-%m-%d")
8
9  decomposition = sm.tsa.seasonal_decompose(data)
10 decomposition.plot()
11
12 # seasonally adjusted data
13 deseas = decomposition.observed - decomposition.seasonal
14 deseas.plot()
15
16 # detrended data
17 detrended = decomposition.observed - decomposition.trend
18 detrended.plot()
```

# Time series decomposition - example



# Time series decomposition - example

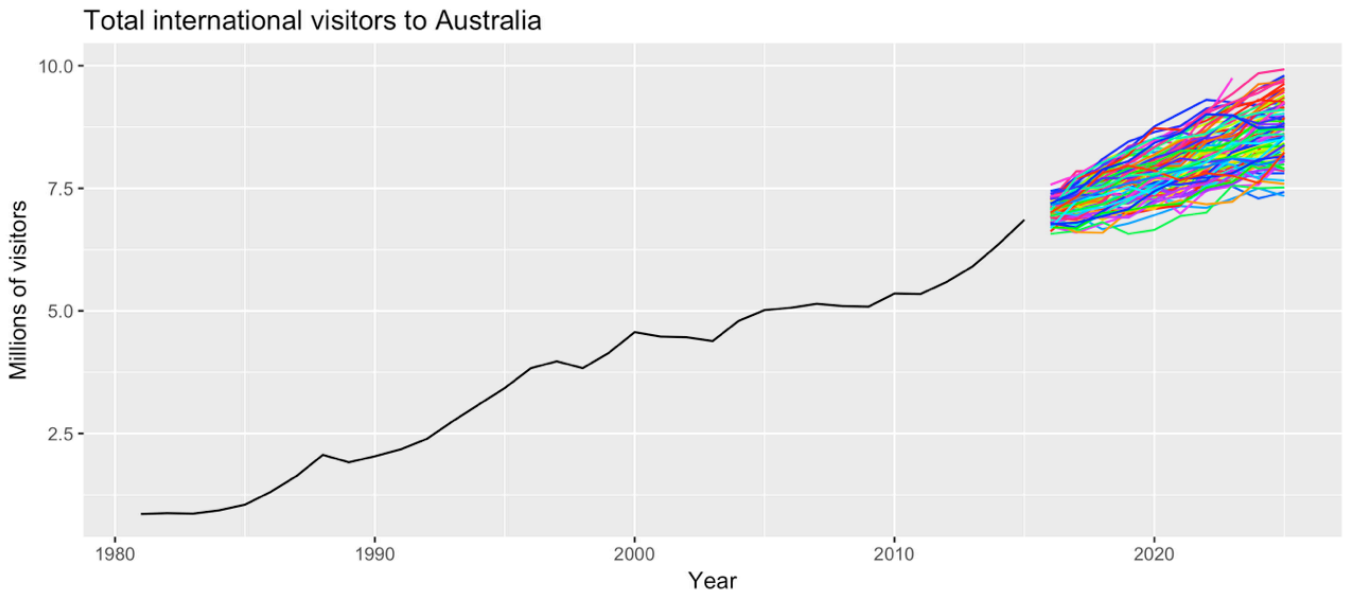


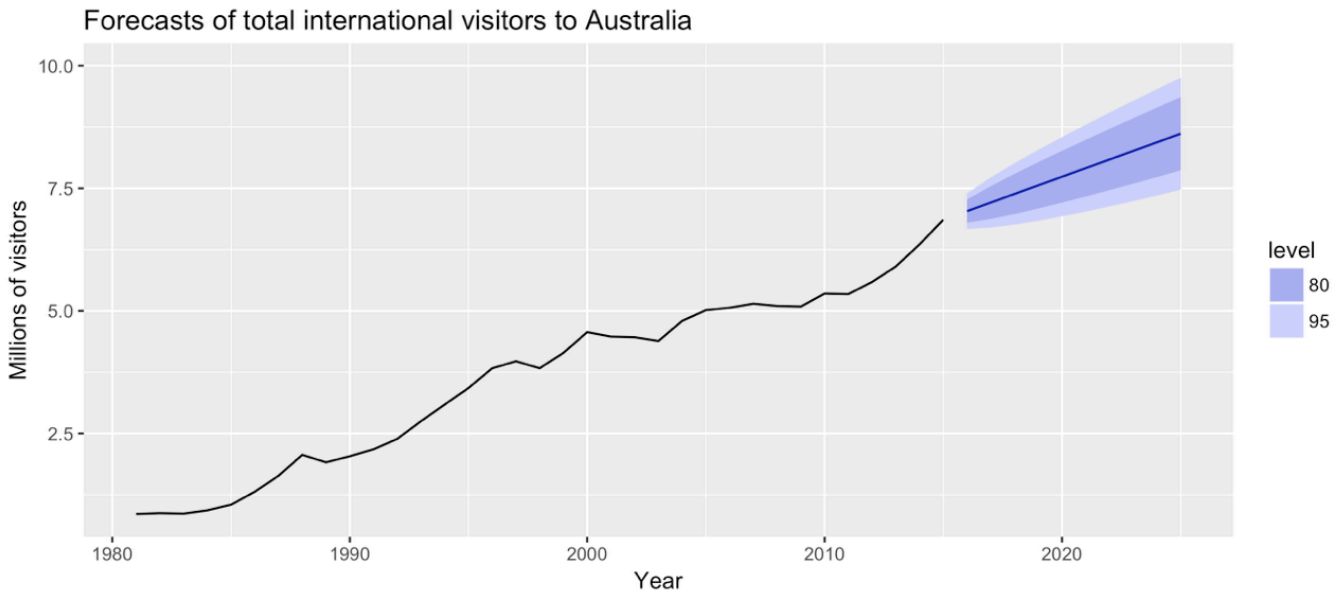
# Part VIII

# Forecasting Methods

# Sample futures



# Forecast intervals




- The 80% forecast intervals should contain 80% of the future observations
- The 95% forecast intervals should contain 95% of the future observations

# Measures of forecast accuracy

| Definitions | Observation $y_t$ | Forecast $\hat{y}_t$ | Forecast error $e_t = y_t - \hat{y}_t$ |
|---|---|---|---|

| Accuracy measure | Calculation |
|---|---|
| Mean Absolute Error | $MAE = average(\lvert e_t \rvert)$ |
| Mean Squared Error | $MSE = average(e_t^2)$ |
| Mean Absolute Percentage Error | $MAPE = 100 \times average(\lvert \frac{e_t}{y_t} \rvert)$ |
| Mean Absolute Scaled Error | $MASE = MAE/Q$ |

* Where Q is a scaling constant.

# Some simple forecasting methods

- **Average method**:

$$\hat{y}_{T+h|T} = \bar{y} = (y_1 + \cdots + y_T)/T.$$

The notation $\hat{y}_{T+h|T}$ is a short-hand for the estimate of $y_{T+h}$ based on $y_1, \ldots, y_T$

- **Naïve method**:

$$\hat{y}_{T+h|T} = y_T.$$

Because a naïve forecast is optimal when data follow a random walk, these are also called random walk forecasts.

- **Seasonal naïve method**:

$$\hat{y}_{T+h|T} = y_{T+h-m(k+1)},$$

where $m$ is the seasonal period, and $k$ is the integer part of $(h-1)/m$ (i.e., the number of complete years in the forecast period prior to time $T+h$)

# Some simple forecasting methods

- **Drift method (a variation on the naïve method)**:

$$\hat{y}_{T+h|T} = y_T + \frac{h}{T-1}\sum_{t=2}^{T}(y_t - y_{t-1}) = y_T + h\left(\frac{y_T - y_1}{T-1}\right).$$

  We allow the forecasts to increase or decrease over time, where the amount of change over time (called the drift) is set to be the average change seen in the historical data. This is equivalent to drawing a line between the first and last observations, and extrapolating it into the future.

- **Forecasting with decomposition**:

  `https://otexts.com/fpp2/forecasting-decomposition.html`

# Intro to AR, MA and ARMA models

FORECASTING USING ARIMA MODELS IN PYTHON

**James Fulton**
Climate informatics researcher

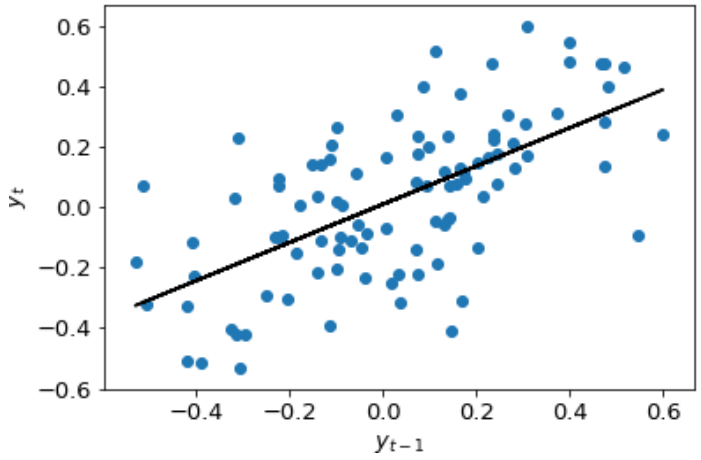
DataCamp

# 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 :

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

# 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

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

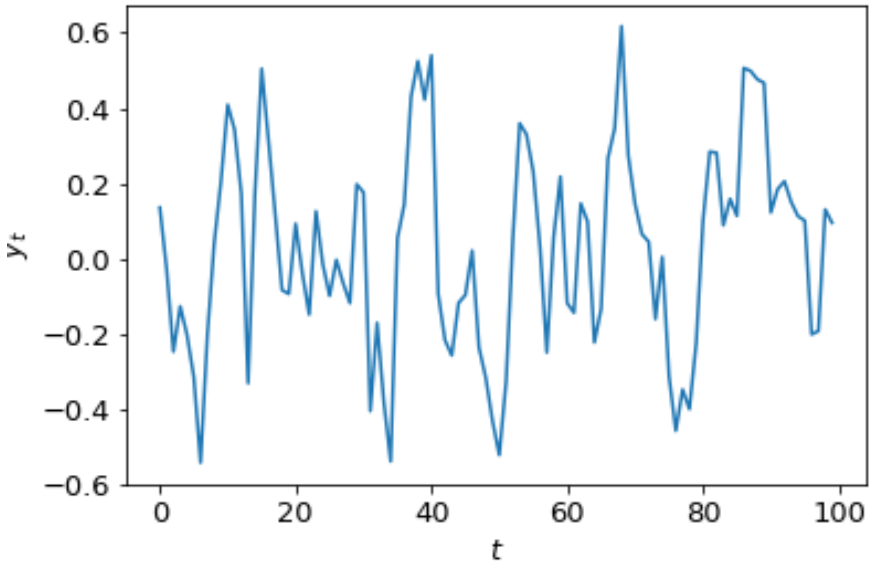
# Creating ARMA data

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

```
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

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

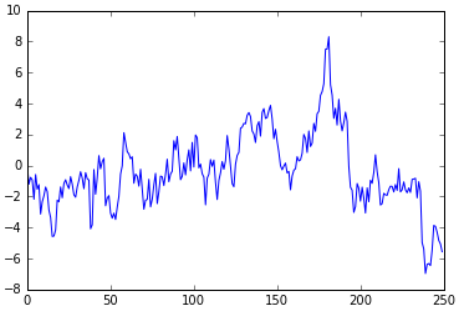


# Fitting and ARMA model

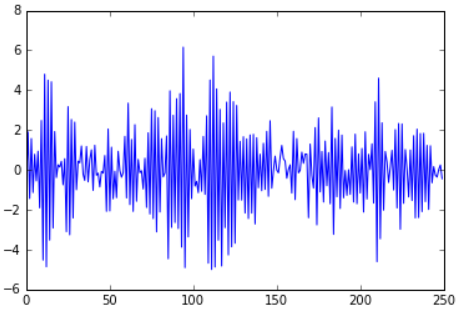
```
from statsmodels.tsa.arima_model import ARMA

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

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

# Comparison of AR(1) Time Series

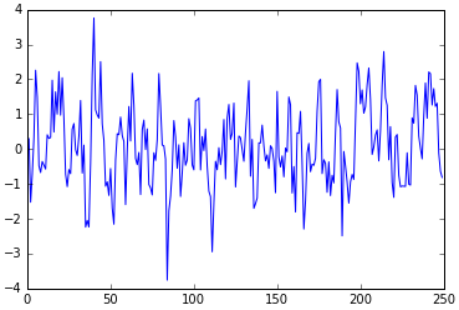
- $\phi = 0.9$



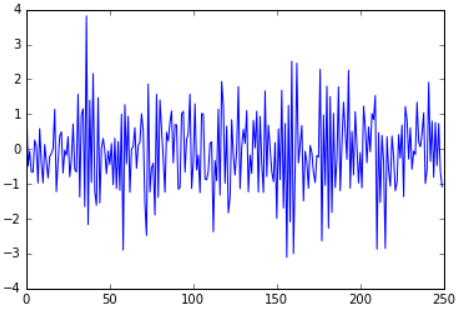

- $\phi = -0.9$




- $\phi = 0.5$

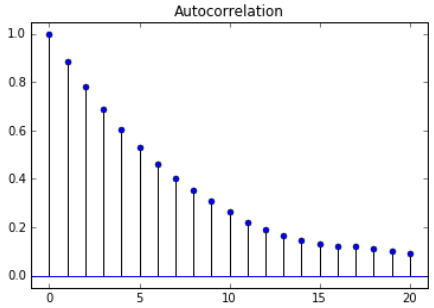



- $\phi = -0.5$

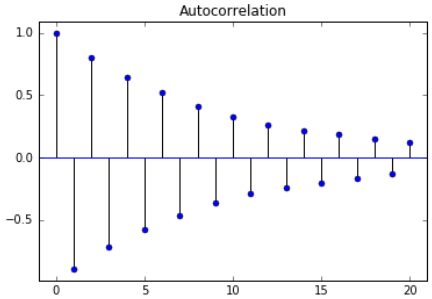


# Comparison of AR(1) Autocorrelation Functions

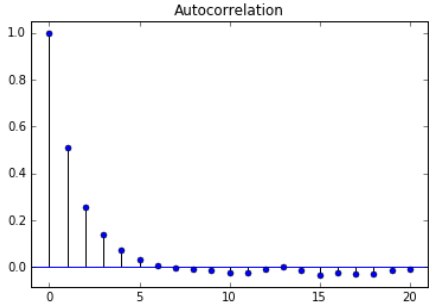
- $\phi = 0.9$



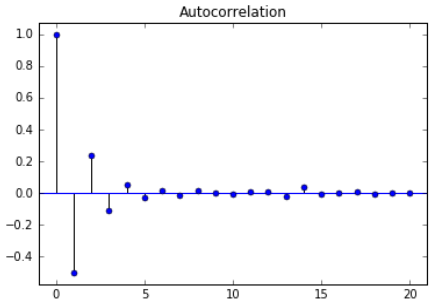

- $\phi = -0.9$




- $\phi = 0.5$




- $\phi = -0.5$



# Identifying the Order of an AR Model

- The order of an AR(p) model will usually be unknown
- Two techniques to determine order
  - Partial Autocorrelation Function
  - Information criteria

# Partial Autocorrelation Function (PACF)

$$R_t = \phi_{0,1} + \phi_{1,1} R_{t-1} + \epsilon_{1t}$$

$$R_t = \phi_{0,2} + \phi_{1,2} R_{t-1} + \phi_{2,2} R_{t-2} + \epsilon_{2t}$$

$$R_t = \phi_{0,3} + \phi_{1,3} R_{t-1} + \phi_{2,3} R_{t-2} + \phi_{3,3} R_{t-3} + \epsilon_{3t}$$

$$R_t = \phi_{0,4} + \phi_{1,4} R_{t-1} + \phi_{2,4} R_{t-2} + \phi_{3,4} R_{t-3} + \phi_{4,4} R_{t-4} + \epsilon_{4t}$$

$$\vdots$$

# Plot PACF in Python

- Same as ACF, but use `plot_pacf` instead of `plt_acf`
- Import module

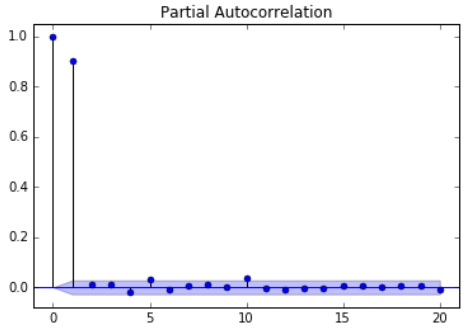
```
from statsmodels.graphics.tsaplots import plot_pacf
```

- Plot the PACF

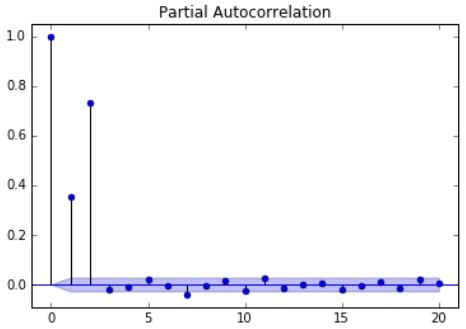
```
plot_pacf(x, lags= 20, alpha=0.05)
```

# Comparison of PACF for Different AR Models

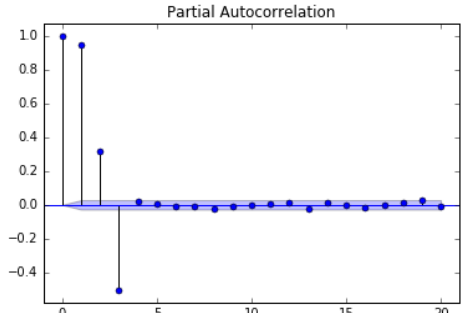
- AR(1)



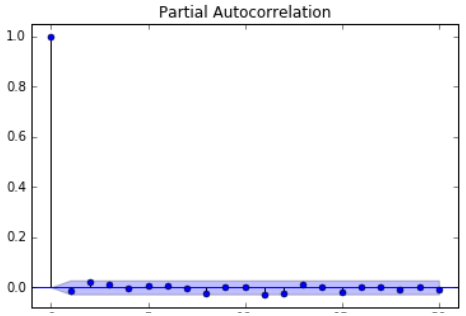

- AR(2)




- AR(3)

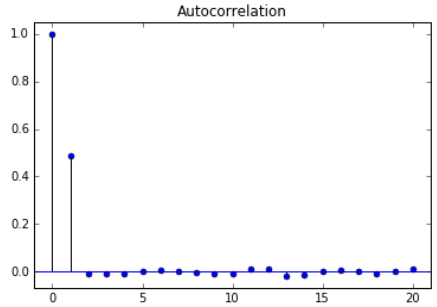



- White Noise

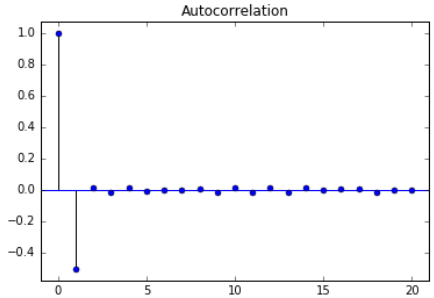


# Comparison of MA(1) Autocorrelation Functions

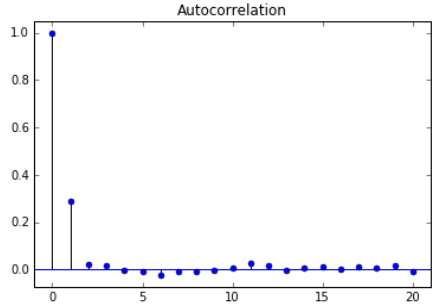
- $\theta = 0.9$



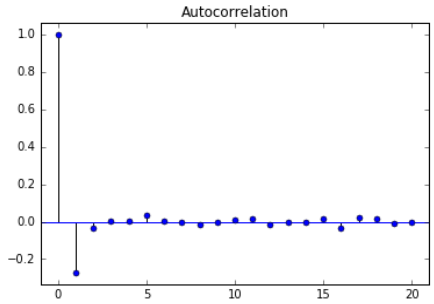

- $\theta = -0.9$




- $\theta = 0.5$




- $\theta = -0.5$



# Fitting time series models

FORECASTING USING 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

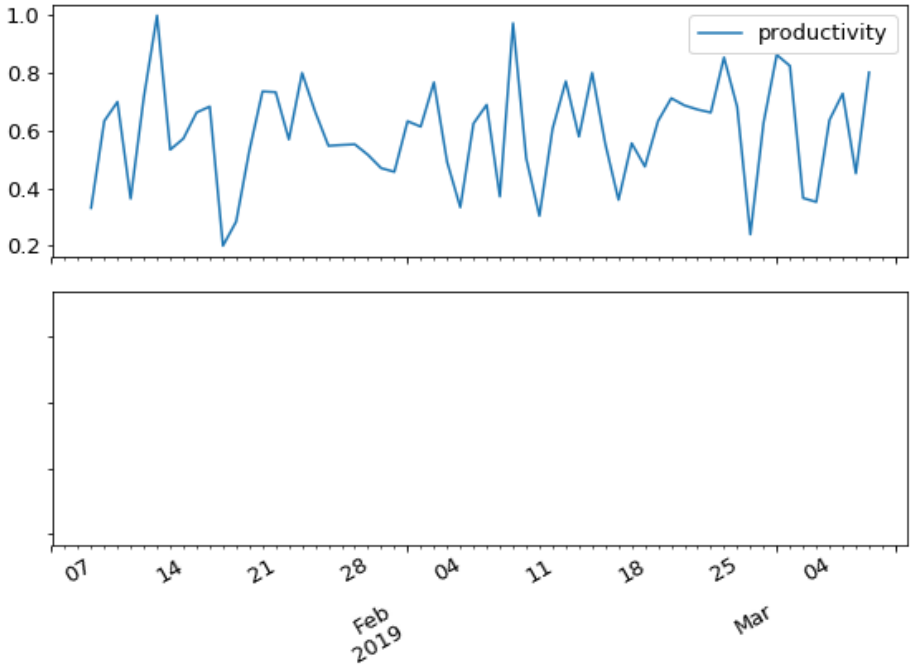
ARMA(1,1) model :

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

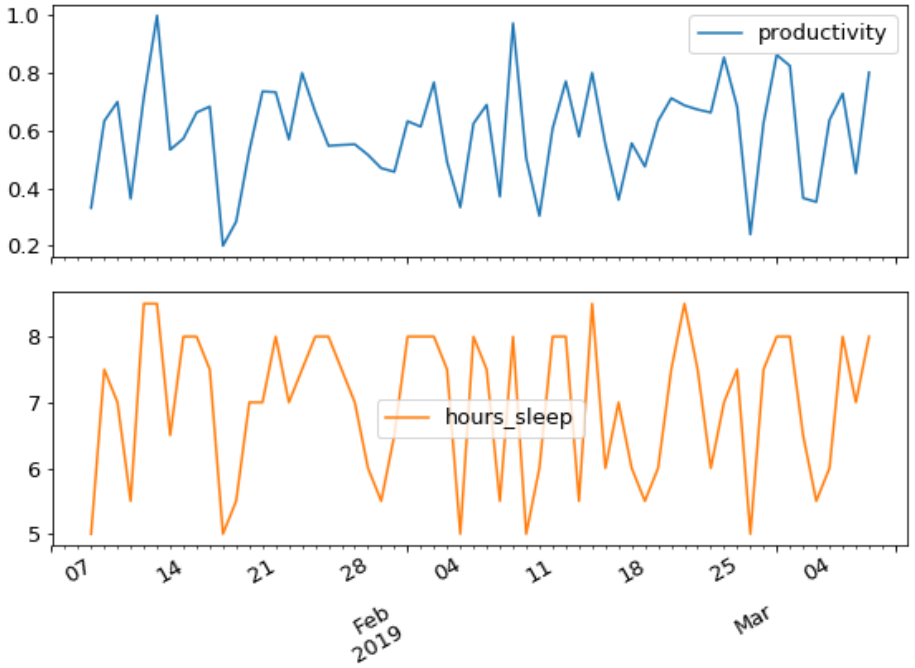
ARMAX(1,1) model :

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

# 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!

FORECASTING USING ARIMA MODELS IN PYTHON

# Forecasting

## FORECASTING USING ARIMA MODELS IN PYTHON

**James Fulton**
Climate informatics researcher

# Predicting the next value

Take an AR(1) model

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

Predict next value

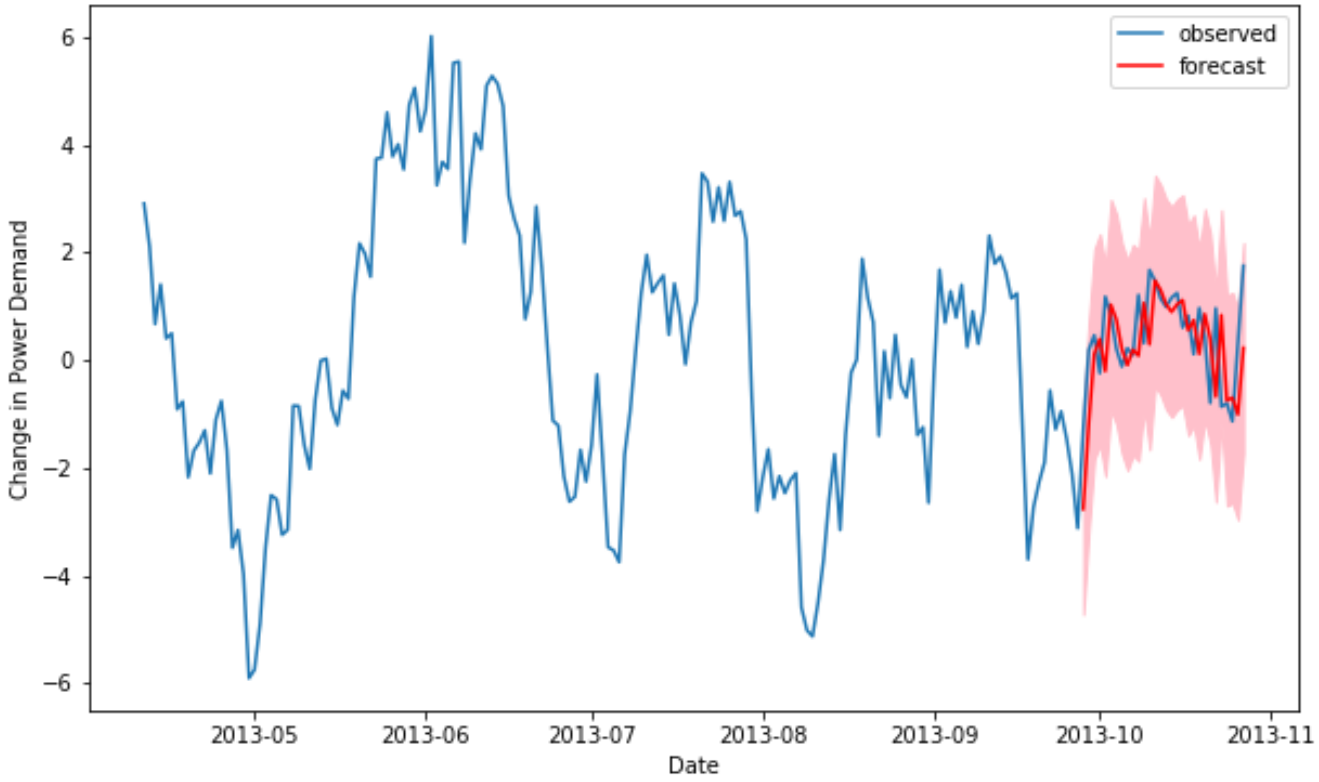
$$y_t = 0.6 \text{ x } 10 + \epsilon_t$$

$$y_t = 6.0 + \epsilon_t$$

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()
```

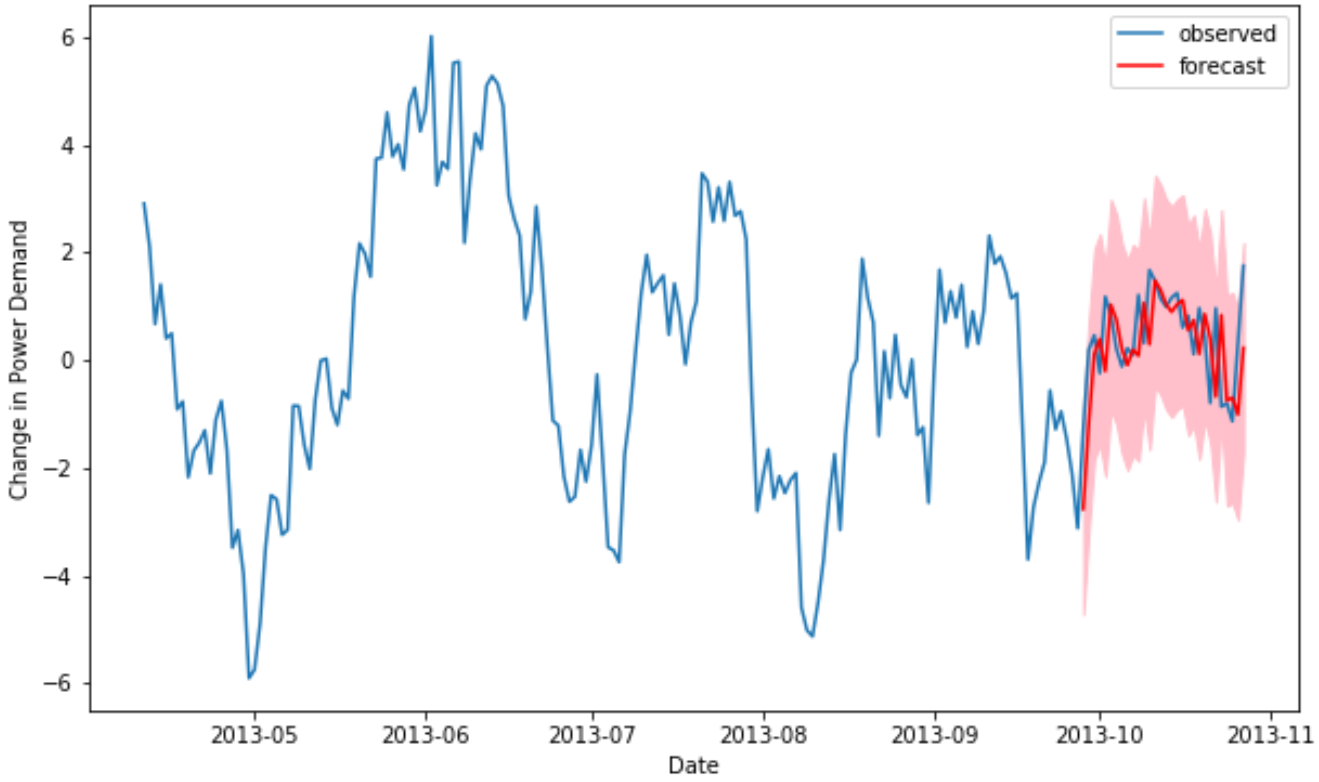
Confidence interval method returns `pandas` DataFrame

```
                            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
```

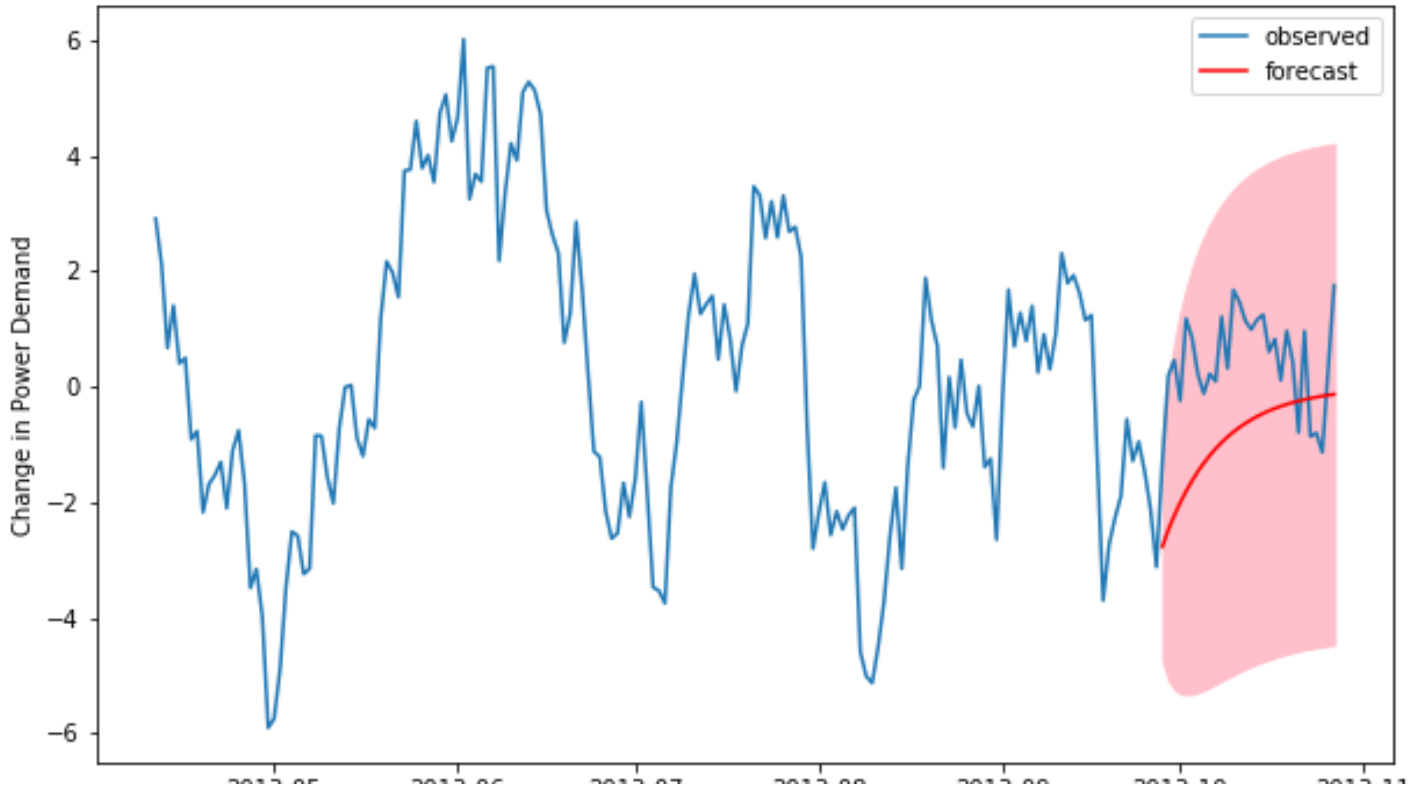
# Plotting predictions

```
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()
```

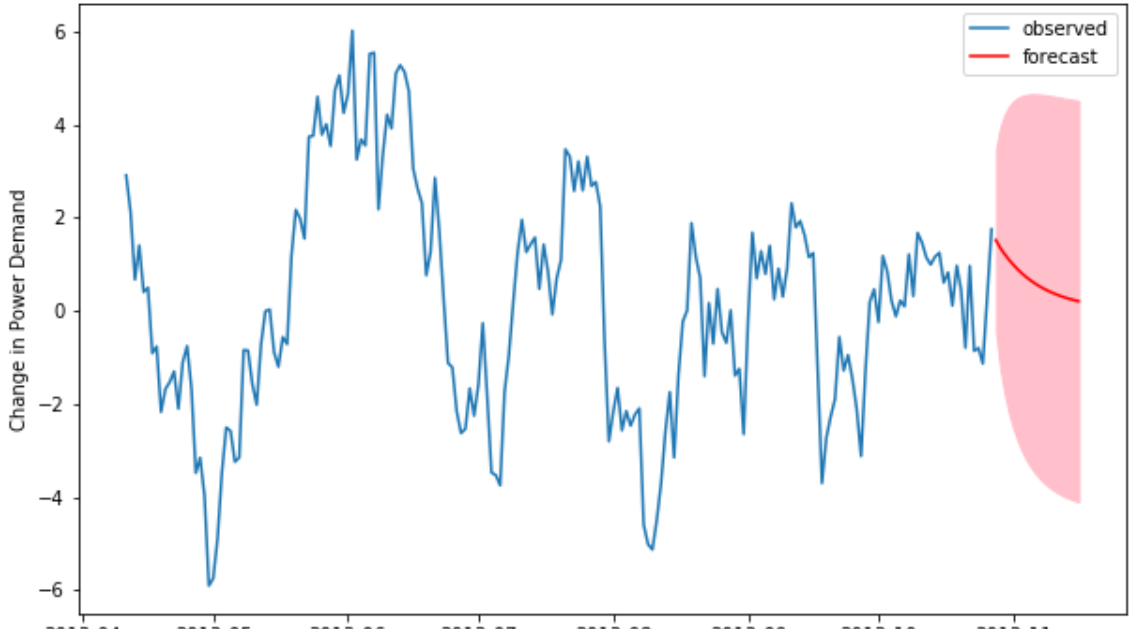
# Forecasting out of sample

```
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)
```



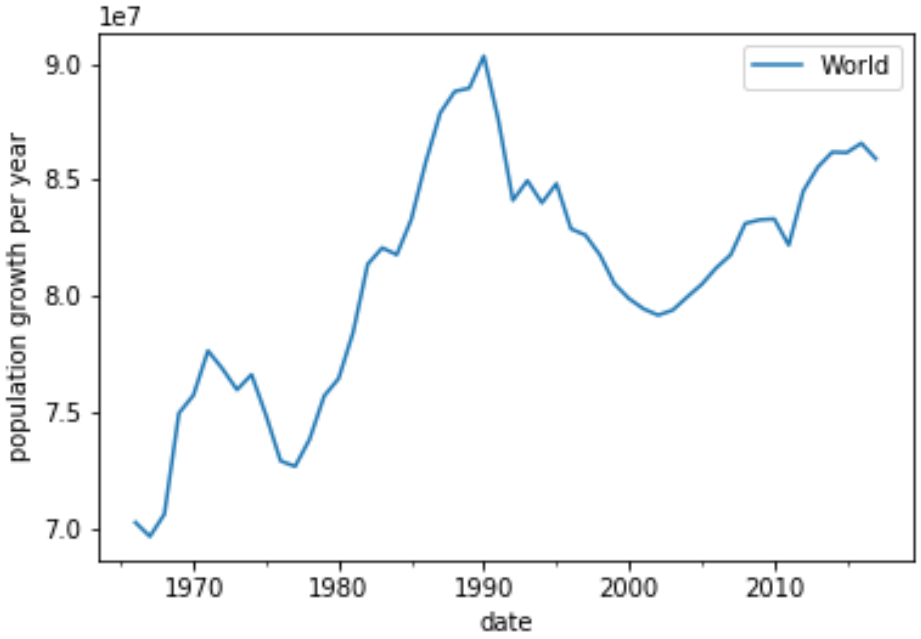
# Introduction to ARIMA models

FORECASTING USING ARIMA MODELS IN PYTHON

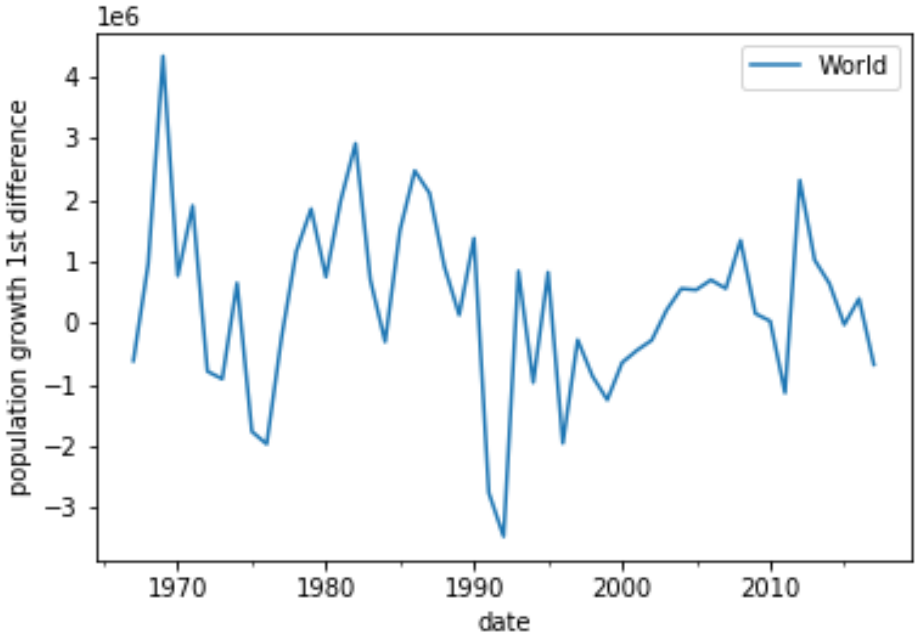
**James Fulton**
Climate informatics researcher

DataCamp

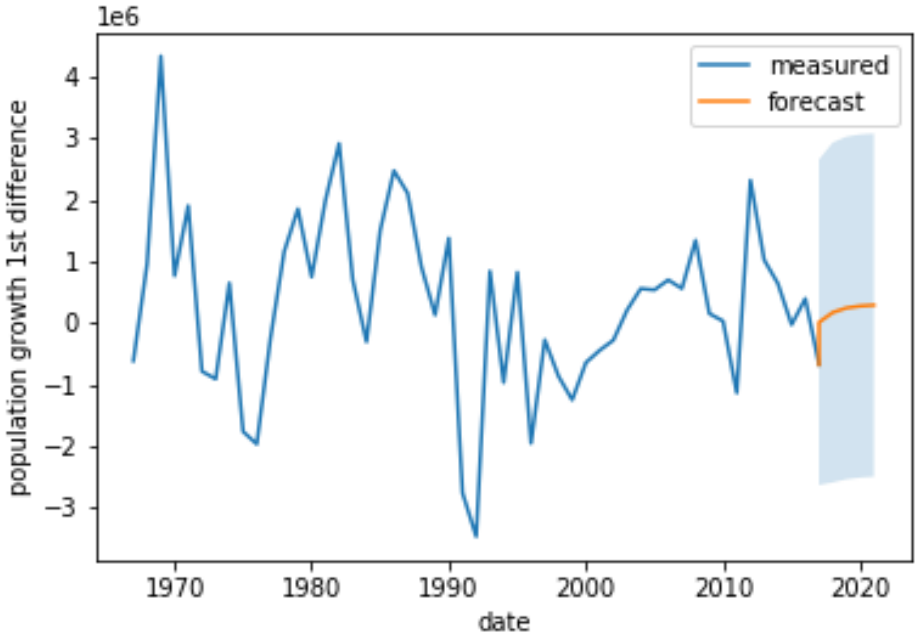
# Non-stationary time series recap



# Non-stationary time series recap



# Forecast of differenced time series



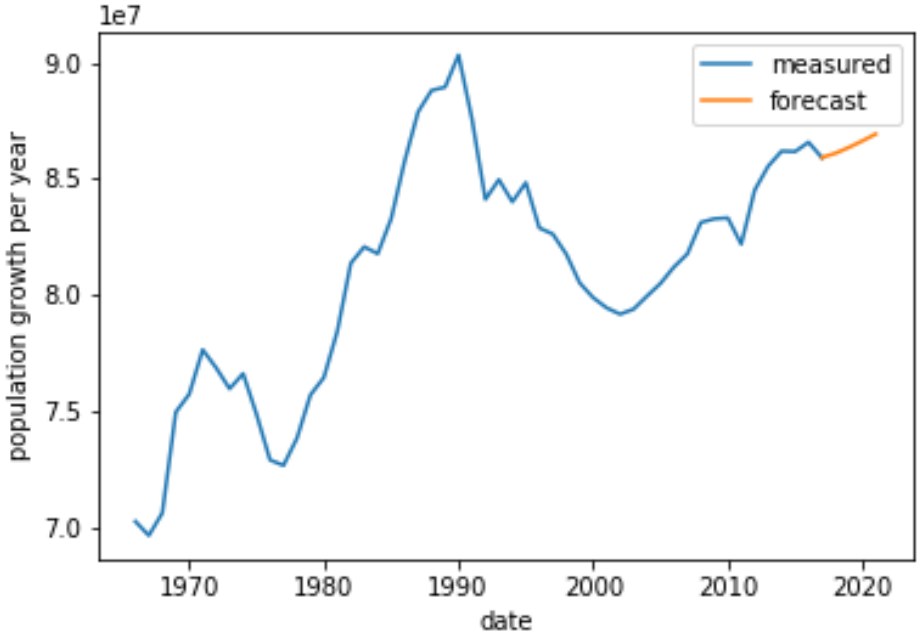
# Reconstructing original time series after differencing

```
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

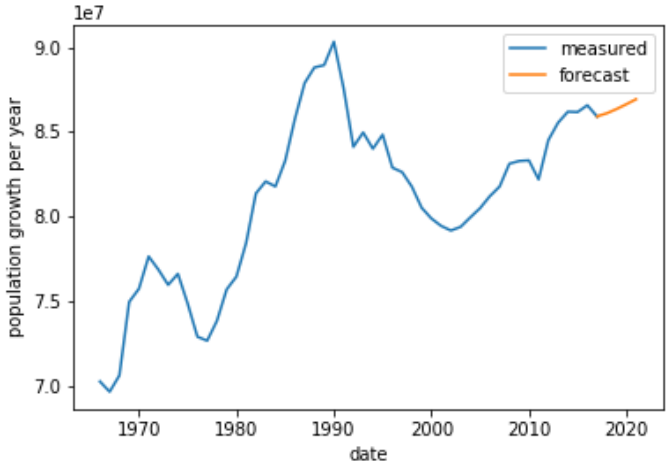
ARMA$(p, 0, q)$ = ARMA$(p, q)$

# 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))
```

# AIC and BIC

FORECASTING USING 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)
```

# Model diagnostics

FORECASTING USING ARIMA MODELS IN PYTHON

**James Fulton**
Climate informatics researcher

# Fitted values and residuals

A *fitted value* is the forecast of an observation using all previous observations

- That is, they are one-step forecasts
- Often not true forecasts since parameters are estimated on all data

A *residual* is the difference between an observation and its fitted value

- That is, they are one-step forecast errors

# Residuals should look like white noise

**Essential assumptions**

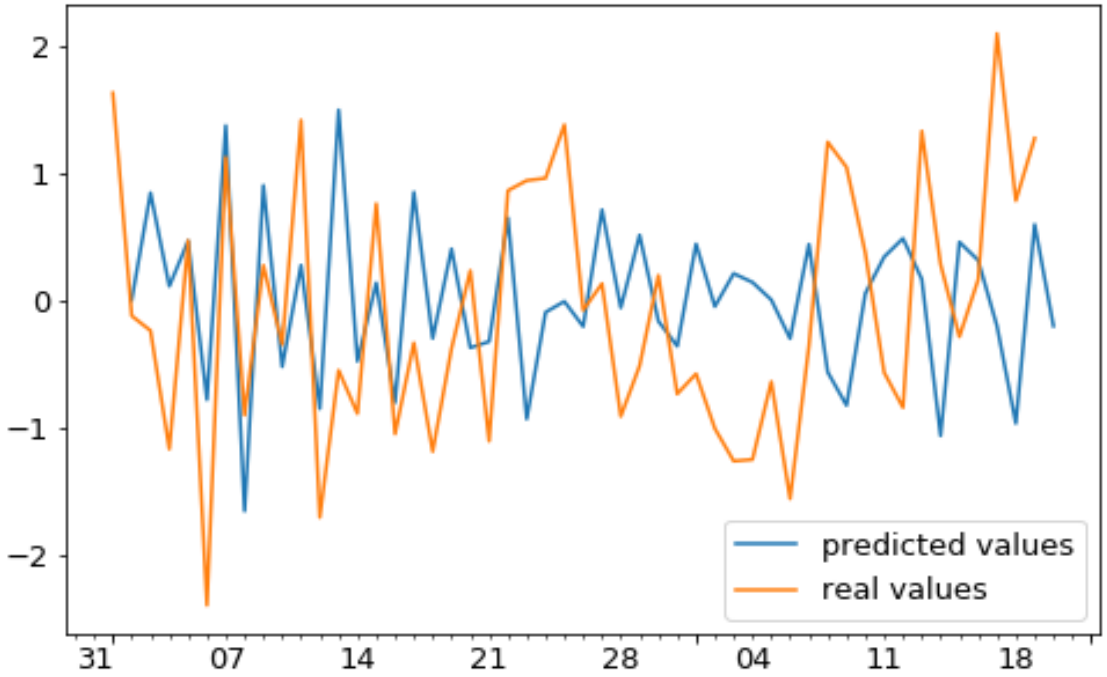
- They should be uncorrelated
- They should have mean zero

**Useful properties (for computing prediction intervals)**

- They should have constant variance
- They should be normally distributed

We can test these assumptions using the `checkresiduals()` function.

# Residuals



# 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

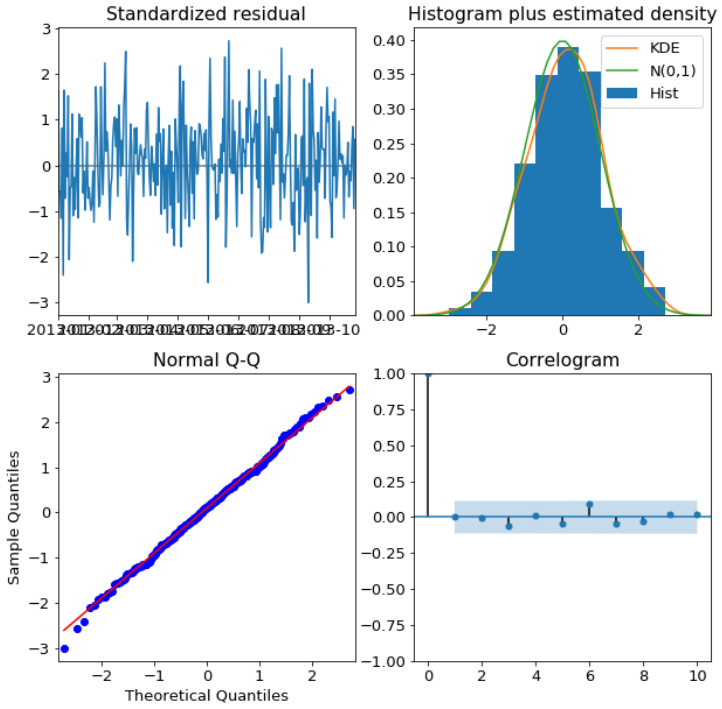
How far our the predictions from the real values?

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

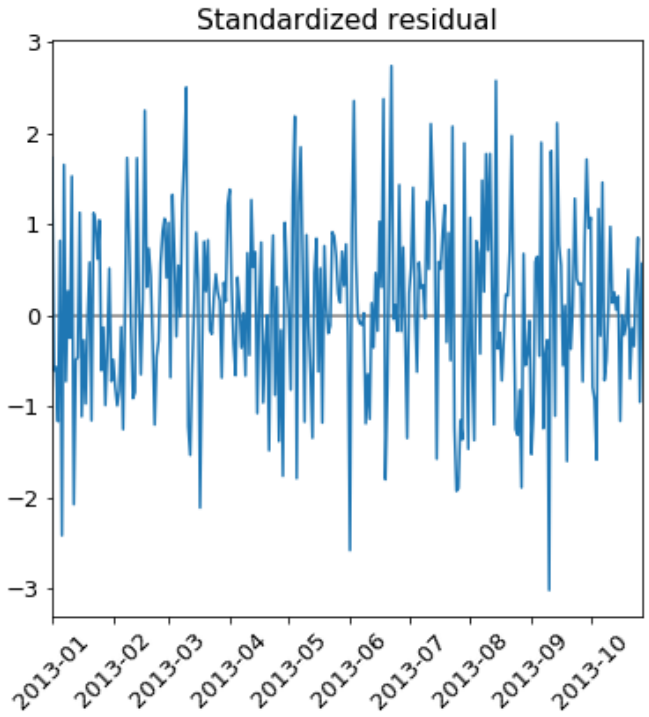
# Plot diagnostics

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

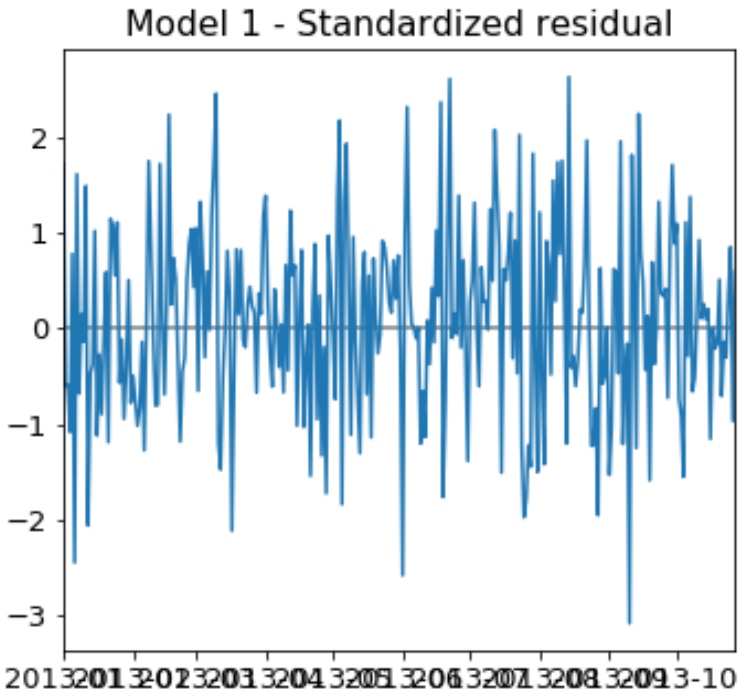
```
# Create the 4 diagostics plots
results.plot_diagnostics()
plt.show()
```

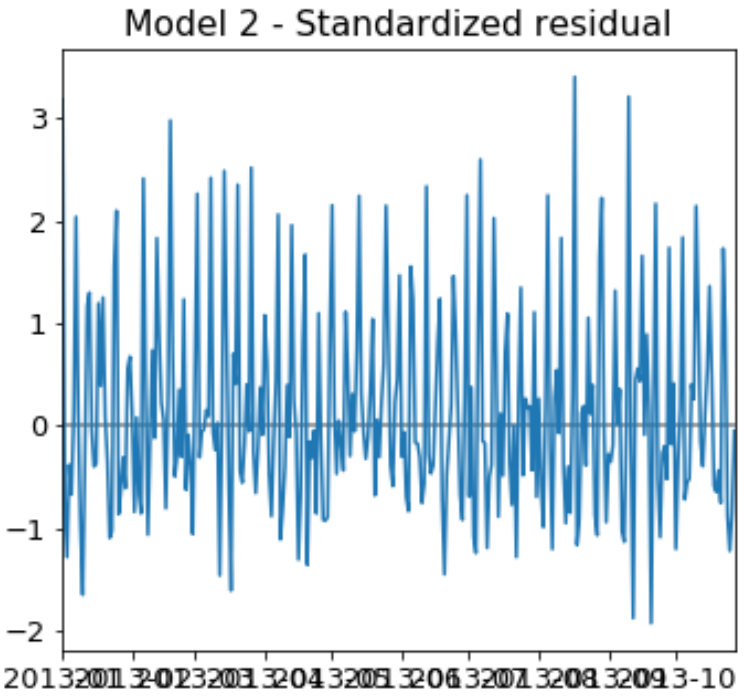


# Residuals plot

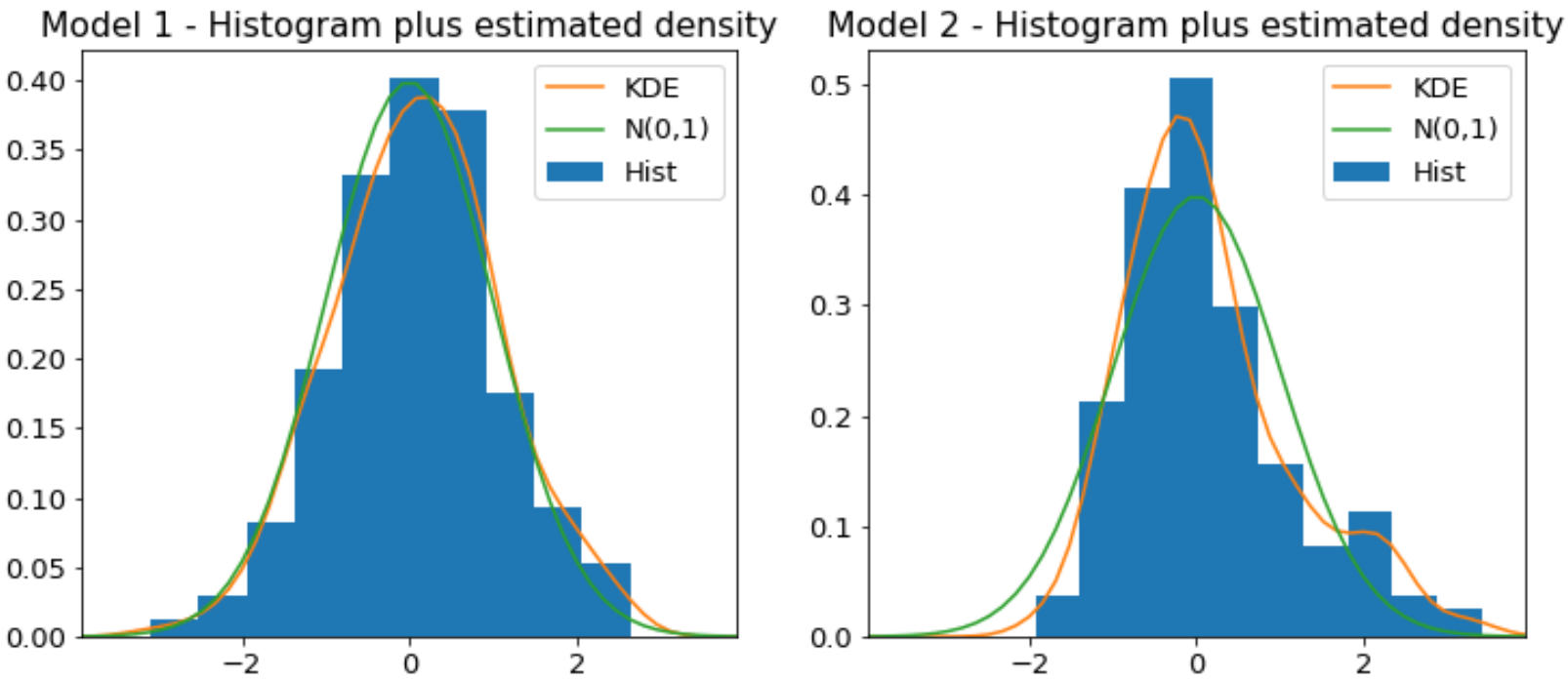


# Residuals plot

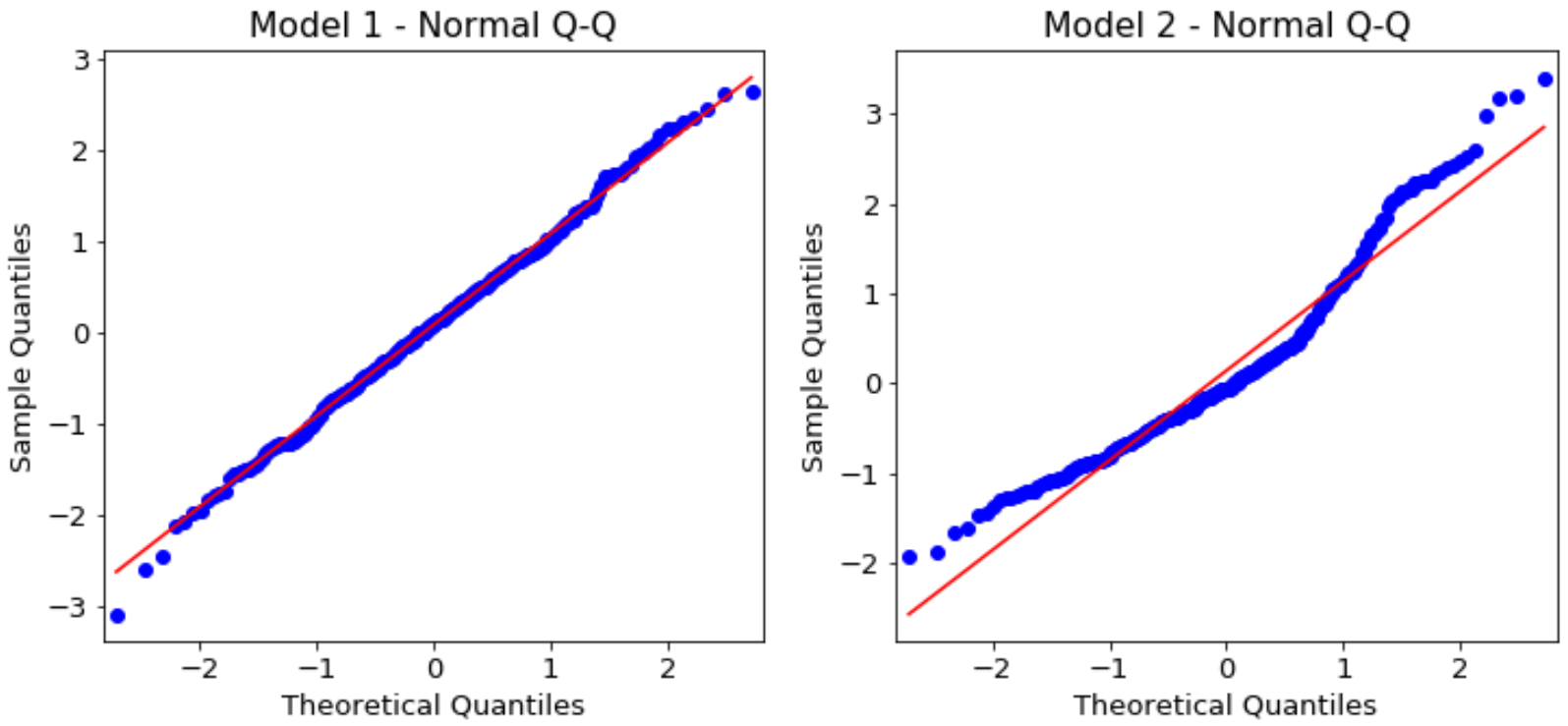


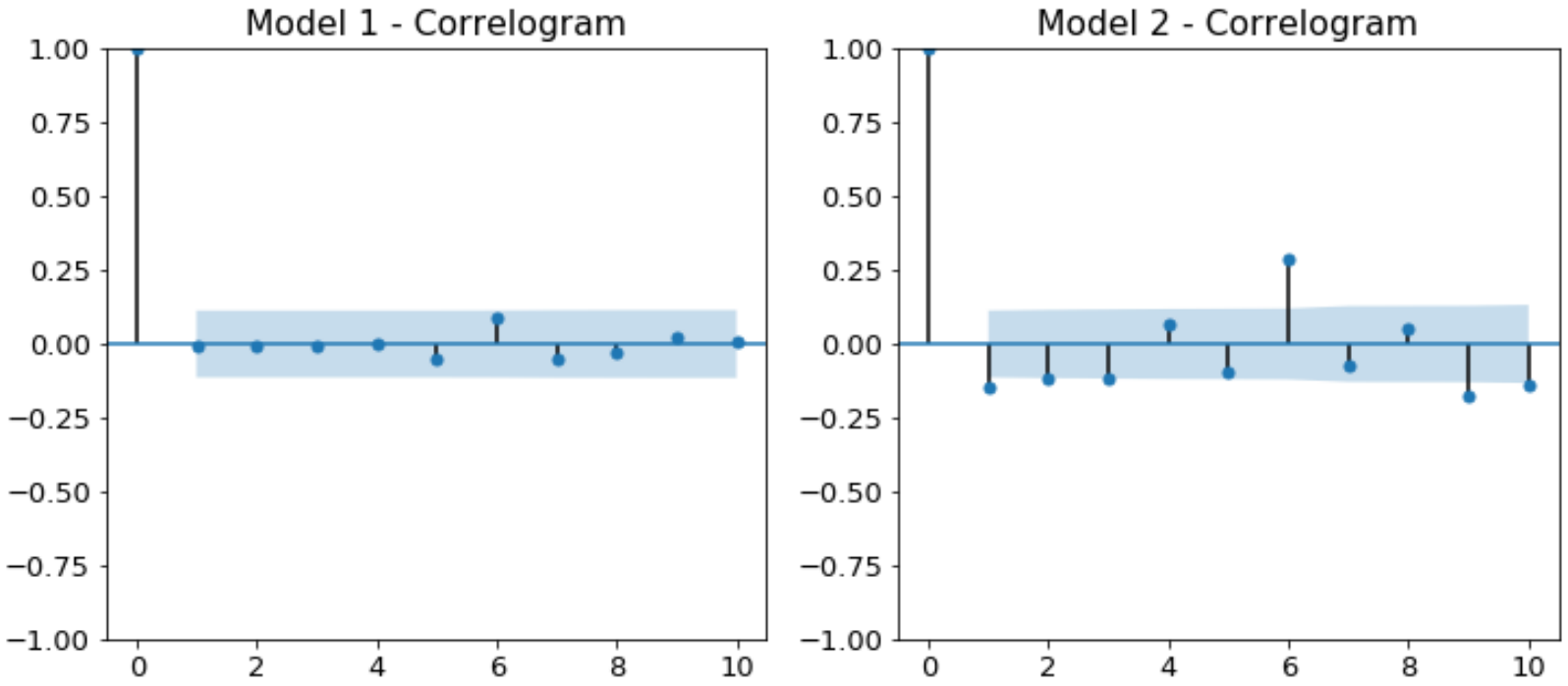
# Histogram plus estimated density



# 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

# Forecast errors

*Forecast "error"* = the difference between observed value and its forecast in the test set.

≠ residuals

- which are errors on the **training set** (vs. **test set**)
- which are based on **one-step** forecasts (vs. **multi-step**)

Compute accuracy using forecast errors on test data

# Measures of forecast accuracy

| Definitions | Observation $y_t$ | Forecast $\hat{y}_t$ | Forecast error $e_t = y_t - \hat{y}_t$ |
|---|---|---|---|

| Accuracy measure | Calculation |
|---|---|
| Mean Absolute Error | $MAE = average(\lvert e_t \rvert)$ |
| Mean Squared Error | $MSE = average(e_t^2)$ |
| Mean Absolute Percentage Error | $MAPE = 100 \times average(\lvert \frac{e_t}{y_t} \rvert)$ |
| Mean Absolute Scaled Error | $MASE = MAE/Q$ |

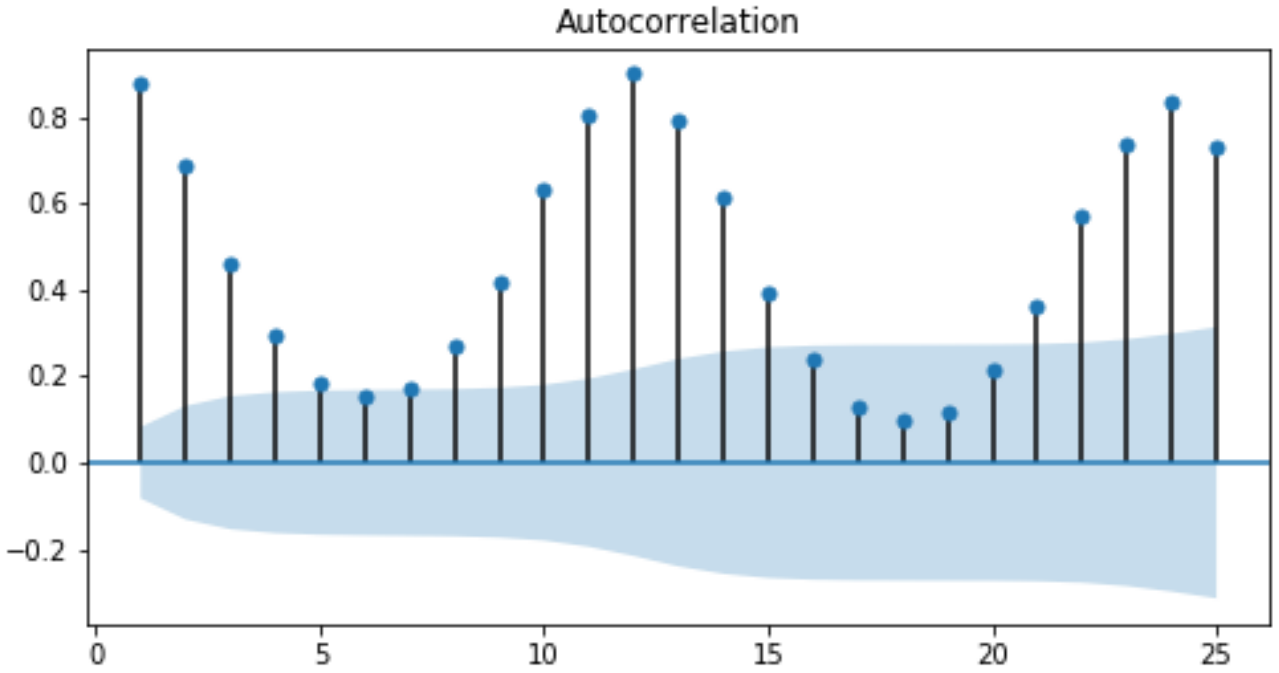
* Where Q is a scaling constant.

# Seasonal time series

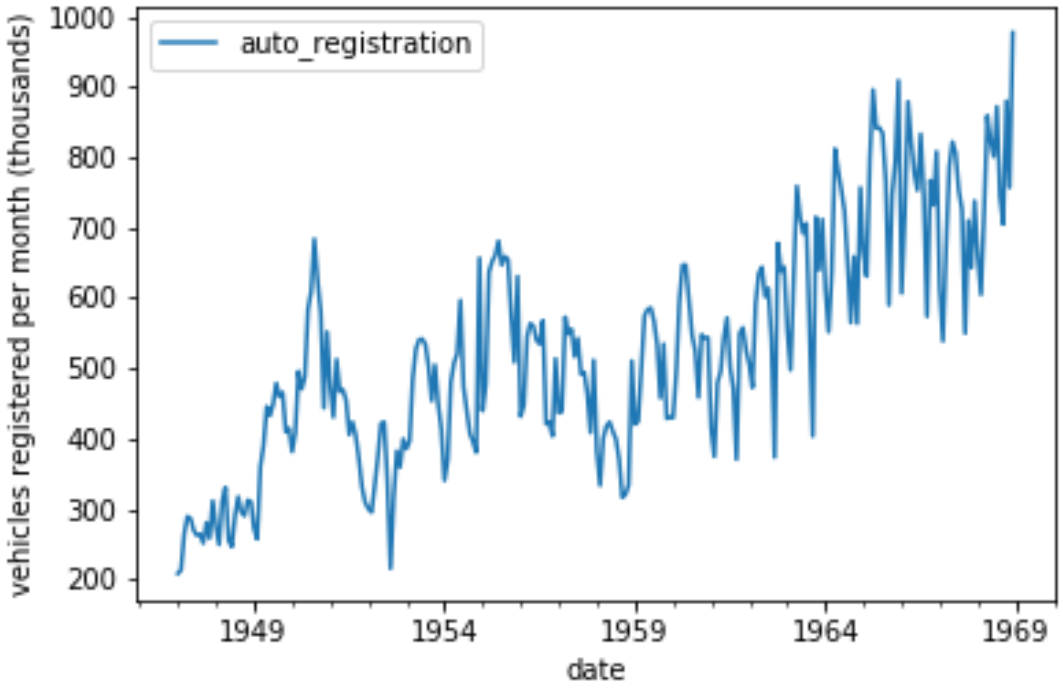
## FORECASTING USING ARIMA MODELS IN PYTHON

**James Fulton**
Climate informatics researcher

# Finding seasonal period using ACF

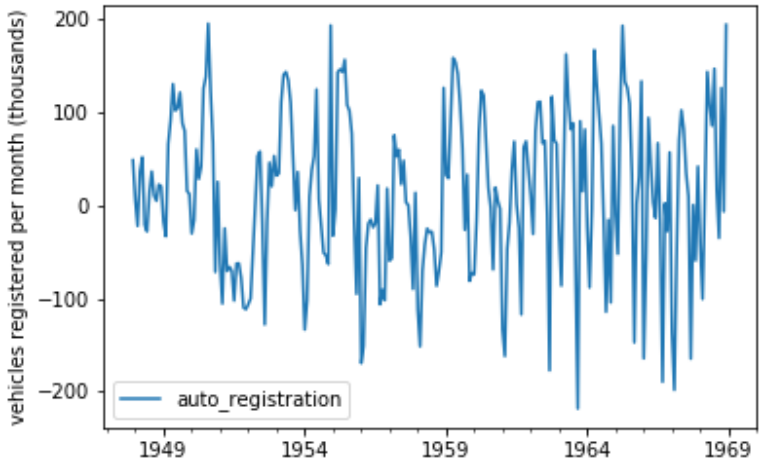


# Identifying seasonal data using ACF



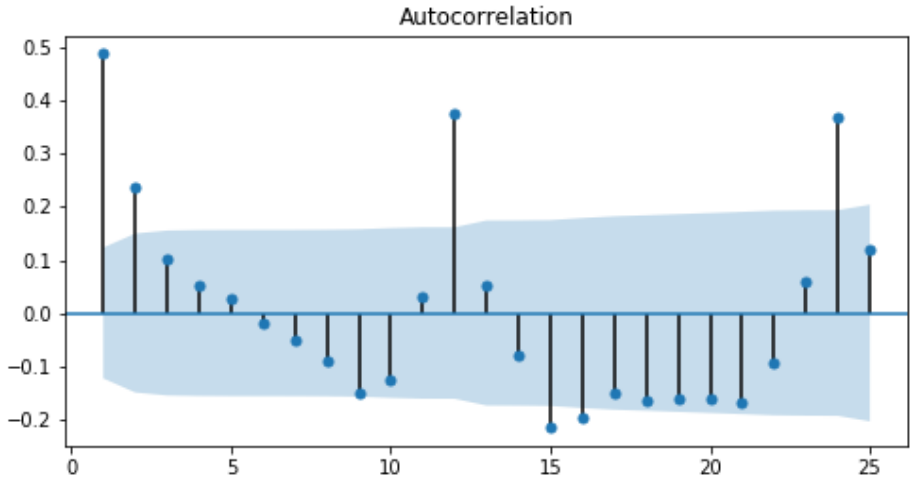
# Detrending time series

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

# Drop NaN values
df = df.dropna()
```

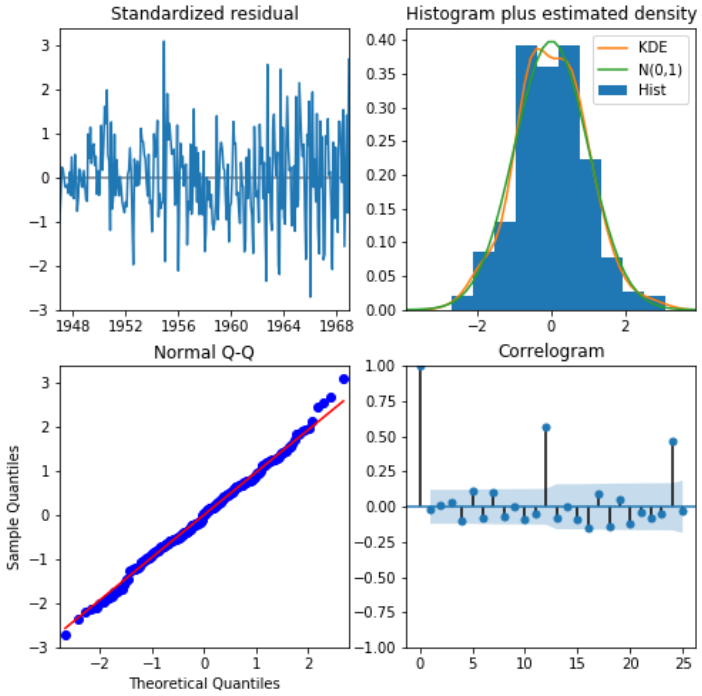


# Identifying seasonal data using ACF

```
# 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



# SARIMA models

FORECASTING USING ARIMA MODELS IN PYTHON

**James Fulton**
Climate informatics researcher

# 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()
```

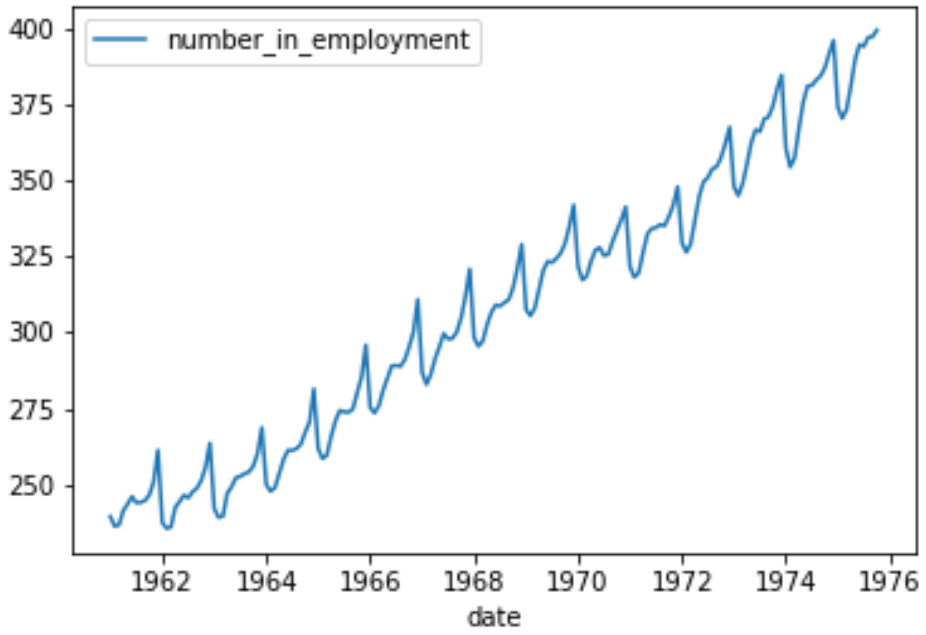
# Seasonal differencing

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

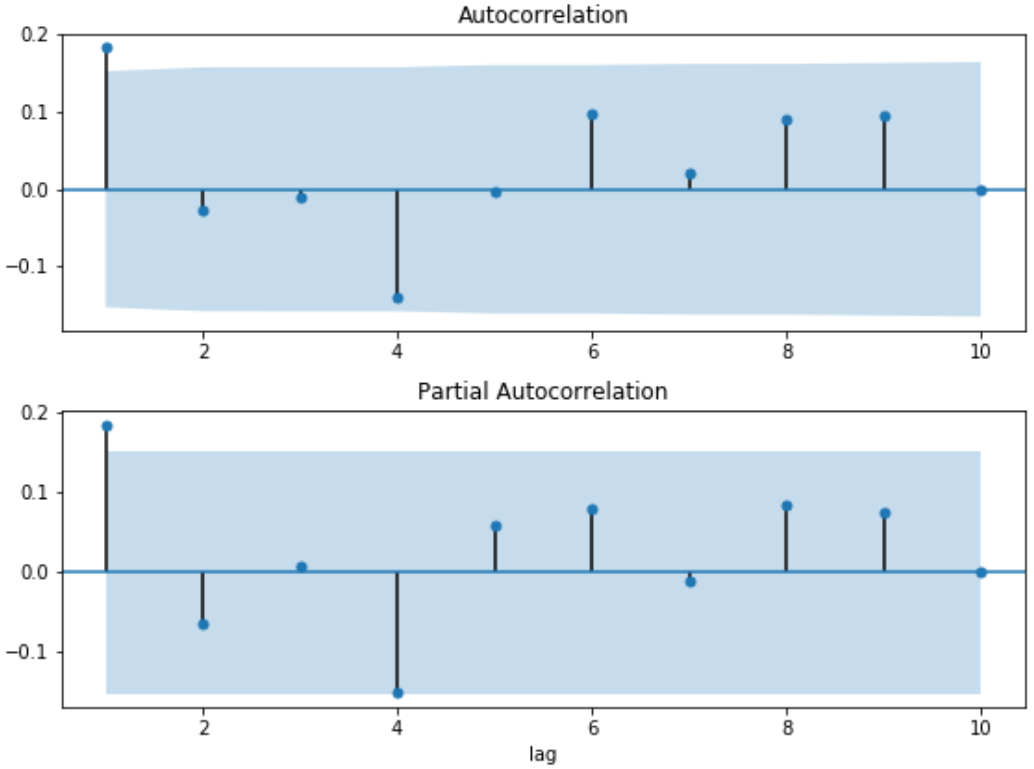
# Differencing for SARIMA models


First difference of time series

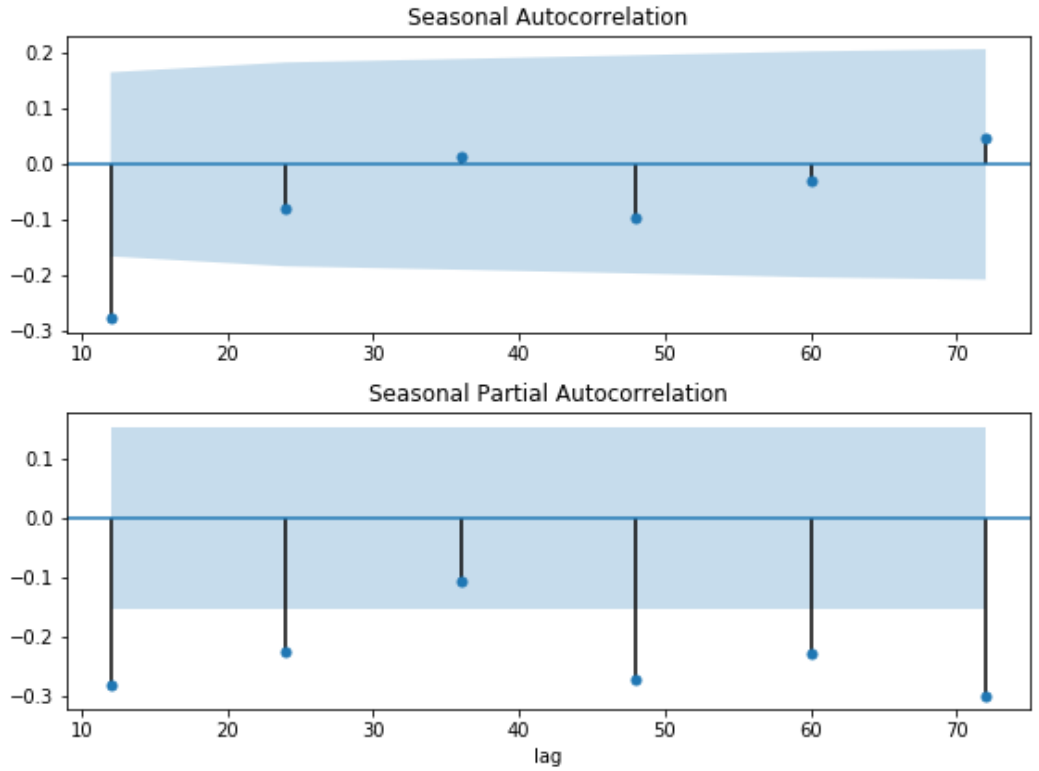
# Differencing for SARIMA models


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()
```

# Automation and saving

FORECASTING USING ARIMA MODELS IN PYTHON

**James Fulton**
Climate informatics researcher

DataCamp

# Searching over model orders

```
import pmdarima as pm
```

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

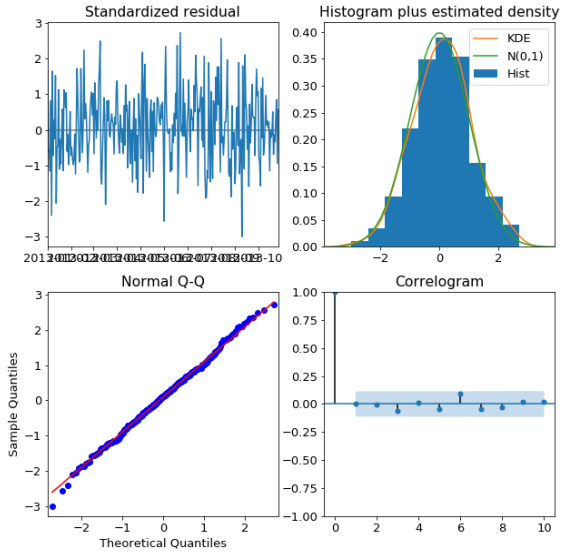
```
Fit ARIMA: order=(2, 0, 2) seasonal_order=(1, 1, 1, 12); AIC=nan, BIC=nan, Fit time=nan seconds
Fit ARIMA: order=(0, 0, 0) seasonal_order=(0, 1, 0, 12); AIC=2648.467, BIC=2656.490, Fit time=0.062 s
Fit ARIMA: order=(1, 0, 0) seasonal_order=(1, 1, 0, 12); AIC=2279.986, BIC=2296.031, Fit time=1.171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

...

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

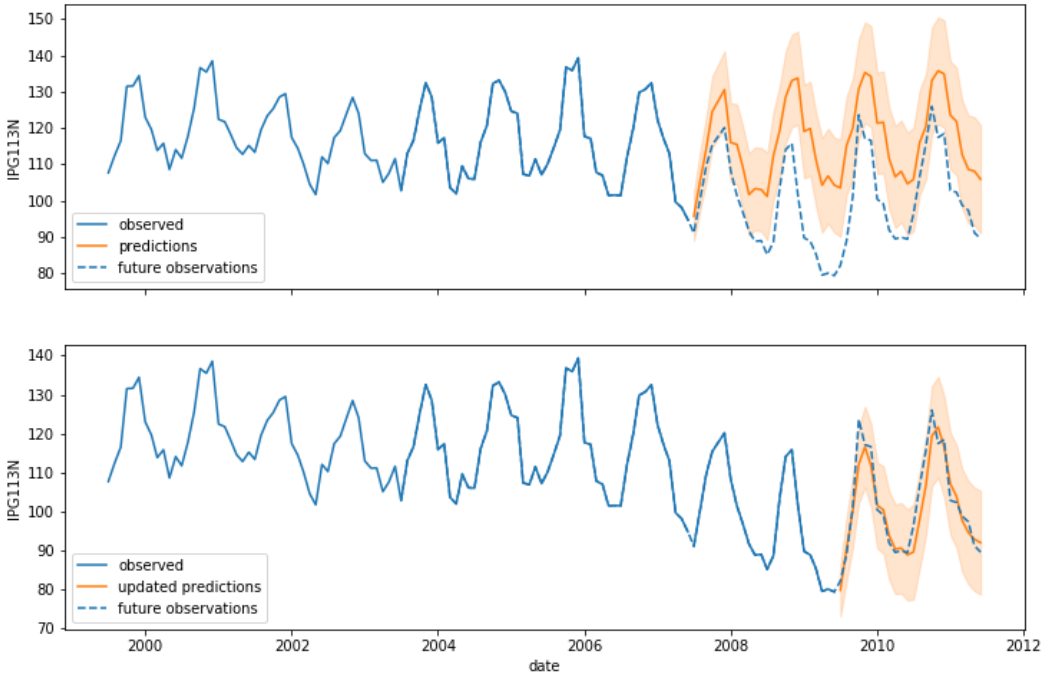
# Saving model objects

```
# 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



# The SARIMAX model

**S** - seasonal

**AR** - autoregressive

**I** - integrated

**MA** - moving average

**X** - exogenous