Data Mining 2

Topic 06 — Time Series Mining

Lecture 01 — Introduction to Time Series Mining

Dr Kieran Murphy

Department of Department of Computing and Mathematics,
INSTITUTION.
(kmurphy@wit.ie)

Spring Semester, 2022

RESOURCE OUTLINE LABEL

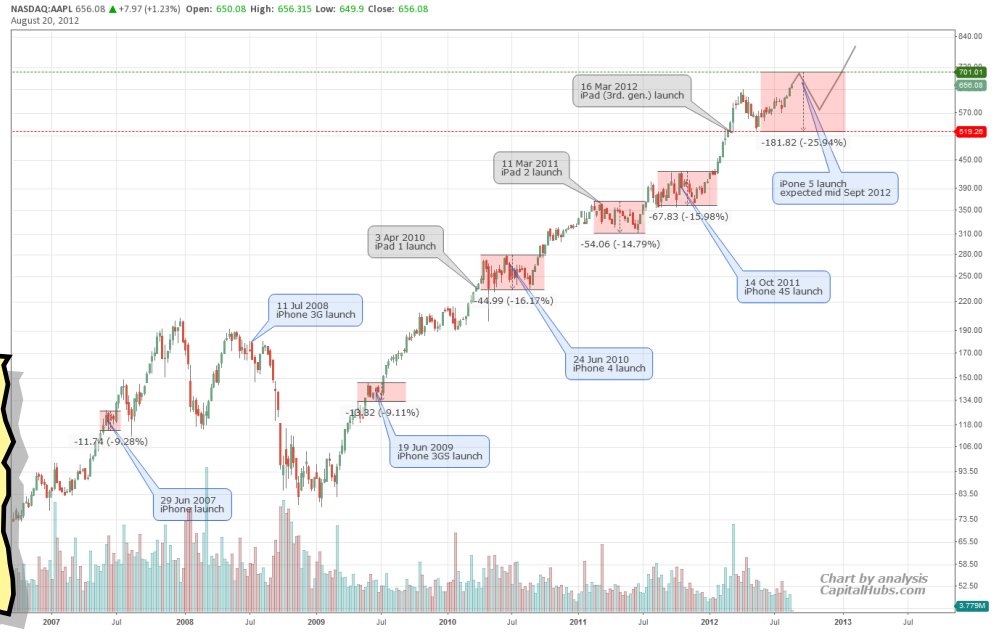
- Components of a time series
- Traditional time series models
- Similar time series matching

Time Series

Definition 1 (Time Series)

A time series consists of a sequence of values or events obtained over repeated measurements of time (weekly, hourly, . . .).

- Many applications —- stock market analysis, economic an sales forecasting, scientific and engineering experiments, medical treatments, etc.
- Given a time series, we are interested in
  - Identify structure (trend/seasonality/lags).
  - Forecasting.
  - Reconcile related time series.
  - Identify correlations

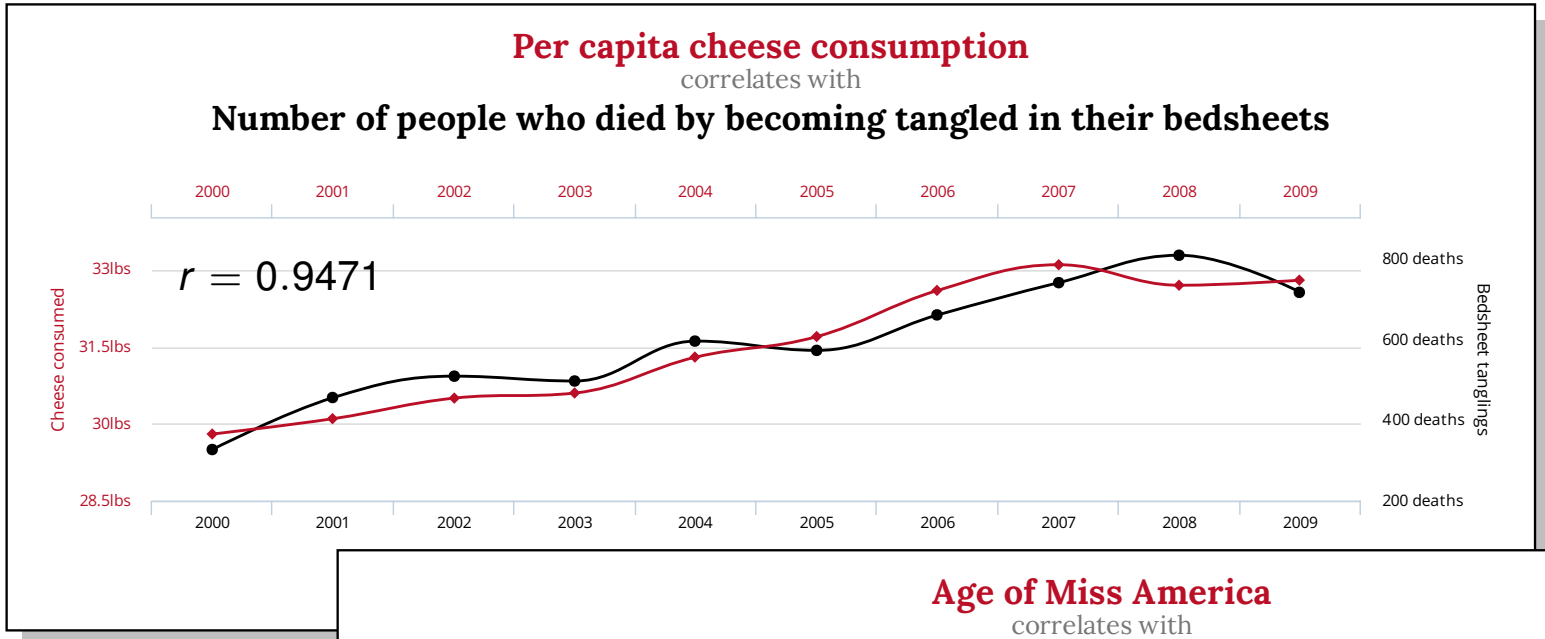


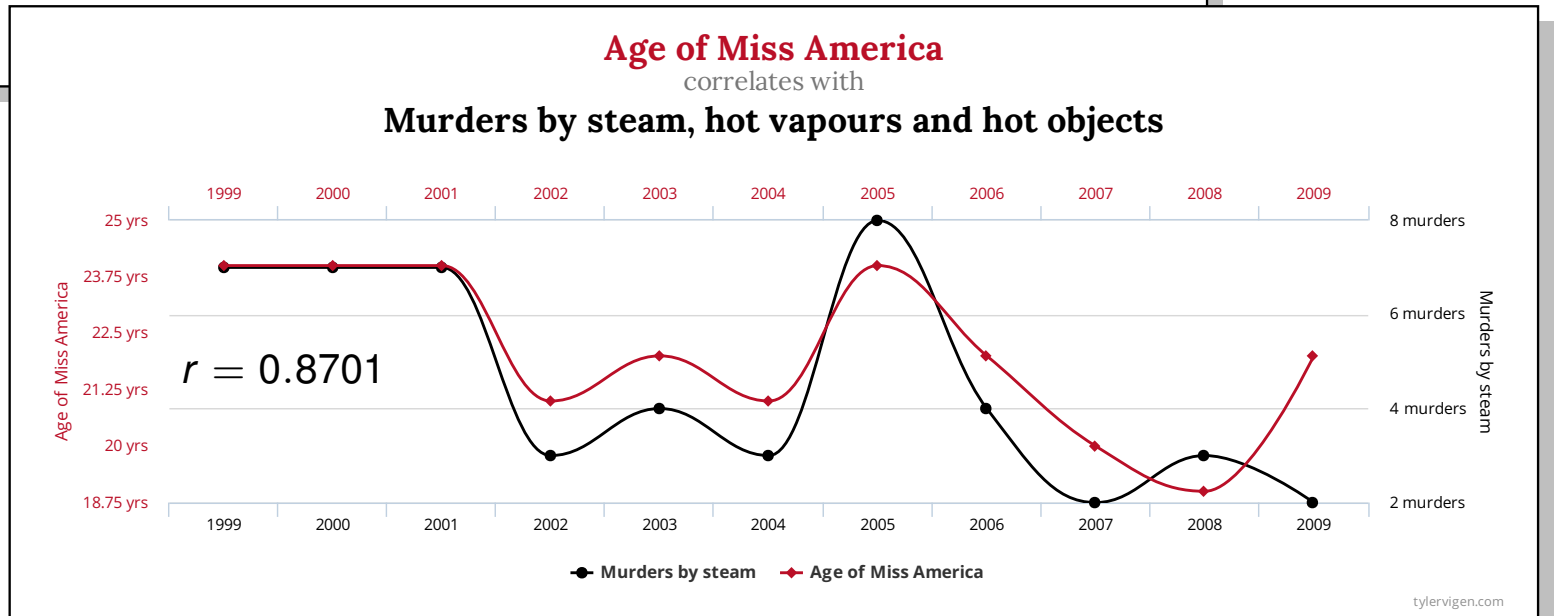
Source: TradingView.com

- The price peaks few days before release date and falls one to three months afterwards.
- The correction after the release date is between 9 and 16%
- The correction is usually zig-zag shaped.

Spurious Correlations (www.tylervigen.com/spurious-correlations)

We always need to take care not to read too much into a high correlation statistic — this is especially true for time series data:

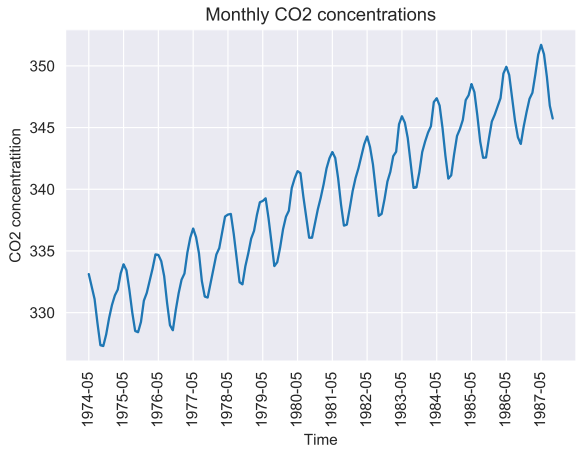




Example Datasets

CO2

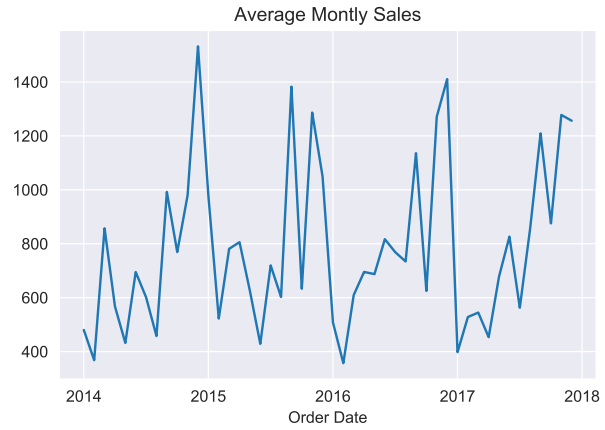
Concentrations in air, by month, from 1974 to 1987.



- Clear upward trend.
- Regular seasonal component.

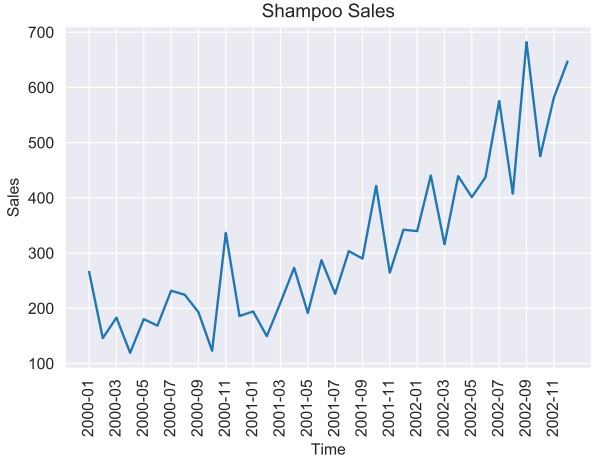
Furniture

Sales, by month — seasonal component and no trend?



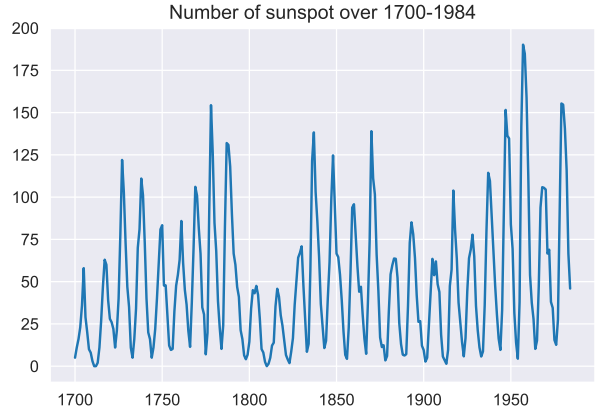
Shampoo

Sales by month — trend but no seasonal?



Sunspots

Frequency by month, over 1700–1984.



- No obvious trend or seasonal effect, but possible lag effect (low value follow low value etc.)
- May need to transform to ensure constant variance.

Notation

Assume that the series X_t runs throughout time, that is $(X_t)_{t=0,\pm1,\pm2,\ldots,}$ but is only observed at times $t = 1, \ldots, n$.

- So we only observe data points $X_1, X_2, \ldots, X_n$.
- Theoretical properties refer to the underlying process $(X_t)_{t\in\mathbb{Z}}$.
- The notations X_t and $X(t)$ are interchangeable, i.e., X_t corresponds to the output of the function X evaluated at point t.

The theory for time series is based on the assumption of second-order stationarity.

- Real-life data are often not stationary, e.g., they exhibit a linear trend over time, or they have a seasonal effect.
- We will discuss identifying/removing trend and seasonal effect later, but first focus on stationary processes.

Main Goals in Time Series

Given a time series, X_t

- We wish to characterise
  - mean or trend

$$\mathbf{E}(X_t) = \mu_t$$

  - Variance or volatility

$$\mathbf{Var}(E_t) = \sigma_t^2$$

  - Autocovariance

$$\text{cov}(X_t, X_{t-\tau}) = \mathbf{E}((X_t - \mu_t)(X_{t-\tau} - \mu_{t-\tau}))$$

- Determine the periodicity or cycles of the observed process.
- Decompose time series into latent process

$$\underbrace{X_t = T_t + S_t + v_t}_{\text{additive}} \qquad \underbrace{X_t = T_t \times S_t + v_t}_{\text{multiplicitive}}$$

where T_t represents the trend, S_t represents the seasonality, and v_t represents the unexplained component.

Weakly Stationarity (Second-Order Stationary)

A process is weakly stationary if

- it has the same mean value, μ, at all time points,
- it has the same variance, γ_0, at all time points,
- the covariance between the values at any two time points, depend only on the difference between the two times, and not on the location of the points along the time axis.

Definition 2 (weakly stationary)

The process, X_t is called weakly stationary or second-order stationary if for all integers t and τ:

$$\mathbf{E}(X_t) = \mu \tag{1}$$

$$\text{cov}(X_{t+\tau}, X_t) = \gamma_\tau < \infty \tag{2}$$

where μ is constant and γ_τ does not depend on t.

Strongly Stationary

A process is strongly stationary if every data point follows the same probability distribution:

Definition 3 (strongly stationary)

The process, X_t, is strictly stationary or strongly stationary if

$$(X_{t_1}, \ldots, X_{t_k}) \quad \text{and} \quad (X_{t_1+\tau}, \ldots, X_{t_k+\tau}) \tag{3}$$

have the same distribution for all sets of time points $t_1, \ldots, t_k$ and all integers τ.

Technical Aside

- Note that finite second moments (finite variance) are not assumed in the definition of strong stationarity, therefore, strong stationarity does not necessarily imply weak stationarity.
- However, if a process has a finite second moment then strongly stationary implies weakly stationary.

Autocorrelation Function (acf)

Given a weakly stationary process, X_t, then the sequence (γ_τ) is called the autocovariance function, and satisfies properties

- $\mathbf{Var}(X_t) = \gamma_0$
- $\gamma_\tau = \gamma_{-\tau}$ (since $\text{cov}(X_{t+\tau}, X_t) = \text{cov}(X_t, X_{t+\tau})$)

Definition 4 (Autocorrelation Function (acf))

The Autocorrelation Function (acf) is defined as

$$\rho_\tau = \text{corr}(X_{t+\tau}, X_t) = \frac{\text{cov}(X_{t+\tau}, X_t)}{\text{cov}(X_t, X_t)} = \frac{\gamma_\tau}{\gamma_0} \tag{4}$$

- The autocorrelation function allows us to assess how a time series relates to its past.
- A plot of the acf is called a correlogram or an autocorrelation plot.
- $-1 < \rho_\tau < 1$ with ρ_τ a measure of the dependency between observations τ units apart.

Partial Autocorrelation Function (pacf)

The partial autocorrelation function (pacf) is related to the acf, in that it is a measure of the relationship between an observation in a time series with observations at prior time steps. But in the case of the pacf, the relationships of intervening observations has been removed.

Definition 5 (Partial Autocorrelation Function (pacf))

$$r_\tau = \text{The partial autocorrelation at lag, } \tau \text{ is the correlation that results after removing the effect of any correlations due to the terms at shorter lags.} \quad (5)$$

- (Aside) the relationships of intervening observations are removed using repeated regression via Levinson-Durbin recursion.
- The derivation/formula are not important but interpreting plots of pacf (and acf) are.

Linear Processes

Q: Assume we have a time series without trends or seasonal effects. How might we construct a linear model for a time series with autocorrelation?

Definition 6 (Linear Processes)

The process, X_t, is called a linear process if it has a representation of the form

$$X_t = \mu + \sum_{\tau=-\infty}^{\infty} c_\tau \cdot \epsilon_{t-\tau} \tag{6}$$

where μ is a common mean, $\{c_\tau\}$ is a sequence of fixed constants, and $\{\epsilon_t\}$ are independent random variables with mean 0 and common variance.

- We assume $\sum c_\tau^2 < \infty$ to ensure that the variance of X_t is finite.
- If $\{\epsilon_t\}$ are identically distributed, then such a porocess is strictly stationary.
- If $c_\tau = 0$ for $\tau < 0$ it is said to be causal, i.e., the process at time t does not depend on the future, as yet unobserved, values of ϵ_t.

Autoregressive Processes (AR)

Assume that a current value of the series is linearly dependent upon its previous value, with some error. Then we could have the linear relationship

Definition 7 (Autoregressive processes (AR))

$$X_t = \alpha X_{t-1} + \epsilon_t$$

where ϵ_t is a white noise* time series

- This model is called an autoregressive (AR) model, since X is regressed on itself.
- Here the lag of the autoregression is 1.
- More generally we could have an autoregressive model of order p, an AR(p) model, defined by

$$X_t = \sum_{k=1}^{p} \alpha_k X_{t-k} + \epsilon_t$$

*The ϵ_t is a sequence of uncorrelated random variables (possibly normally distributed, but not necessarily normal) with mean 0 and common variance.

AR — Python Implementation . . . Setup

Python has function to simulate and model (i.e. fit parameters) for all of the standard stationary models, and functions to generate the acf and pacf. To start

- I use the warning package to suppress messages with statsmodels — don't use this when working with pandas.
- We include the required modules/functions.
- The function param_to_string is to simplify generate of image file names.

Stationary.ipynb In[2]:

```
import warnings
warnings.filterwarnings("ignore")

import statsmodels.api as sm
from statsmodels.graphics.tsaplots import plot_acf
from statsmodels.graphics.tsaplots import plot_pacf

def param_to_string(p):
    return ("%s_" % (len(p)-1) +
            "_".join(["%s" % v for v in -p[1:]]).replace(".",""))
```

AR — Python Implementation . . . Generation

Lets start with an AR(1) with single parameter $\alpha = 0.75$, i.e.,

$$X_t = 0.75X_{t-1} + \epsilon_t$$

First create a list of the parameters values — convention for parameters is to

- Include 1 for zero order term.
- Other terms are multiplied by negative one.
- Support for moving average (more later), but set this to [1] for none.

Stationary.ipynb In[3]:

```
ar = np.array([1, -0.75])
ma = np.array([1])
```

Generate series (using fixed seed)

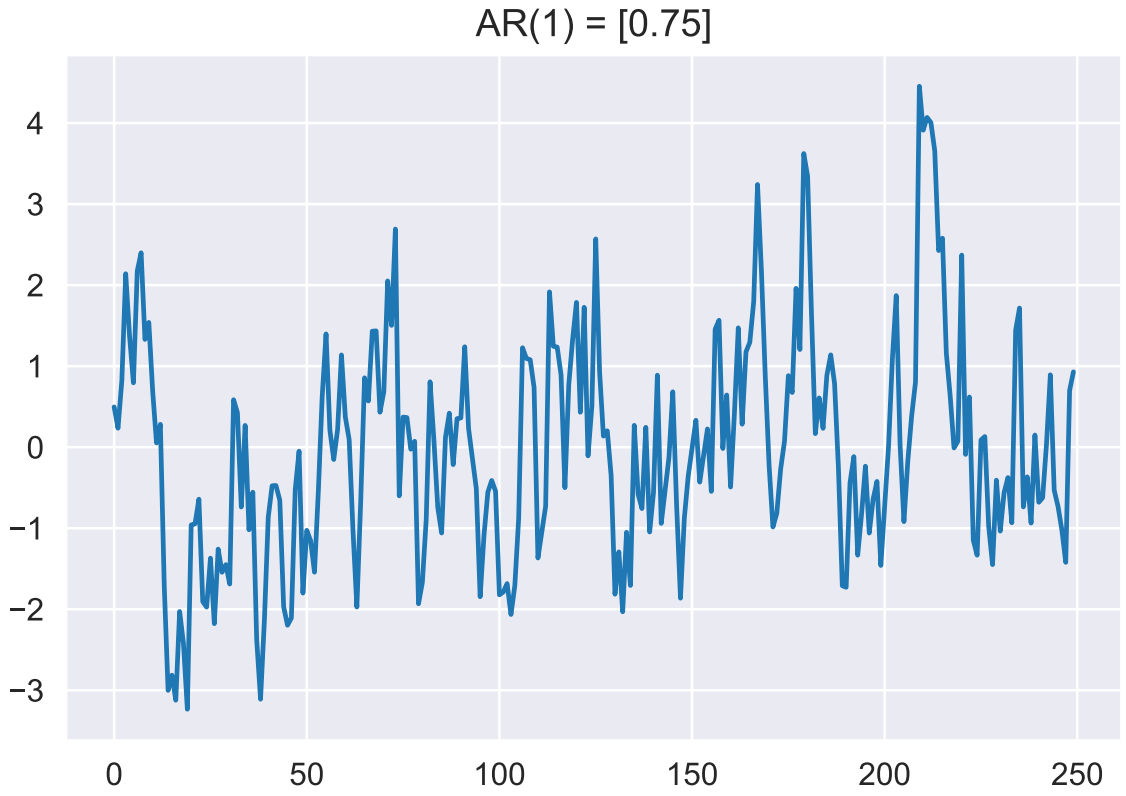
Stationary.ipynb In[4]:

```
np.random.seed(42)
y = sm.tsa.arma_generate_sample(ar, ma, nsample=250, sigma=1)
```

AR — Python Implementation . . . Visualisation

Stationary.ipynb In[5]:

```
plt.plot(y)
plt.title("AR(1) = %s" % -ar[1:])

filename = "../pic/ar_%s.pdf" % param_to_string(ar)
plt.savefig(filename, bbox_inches="tight")
plt.show()
```

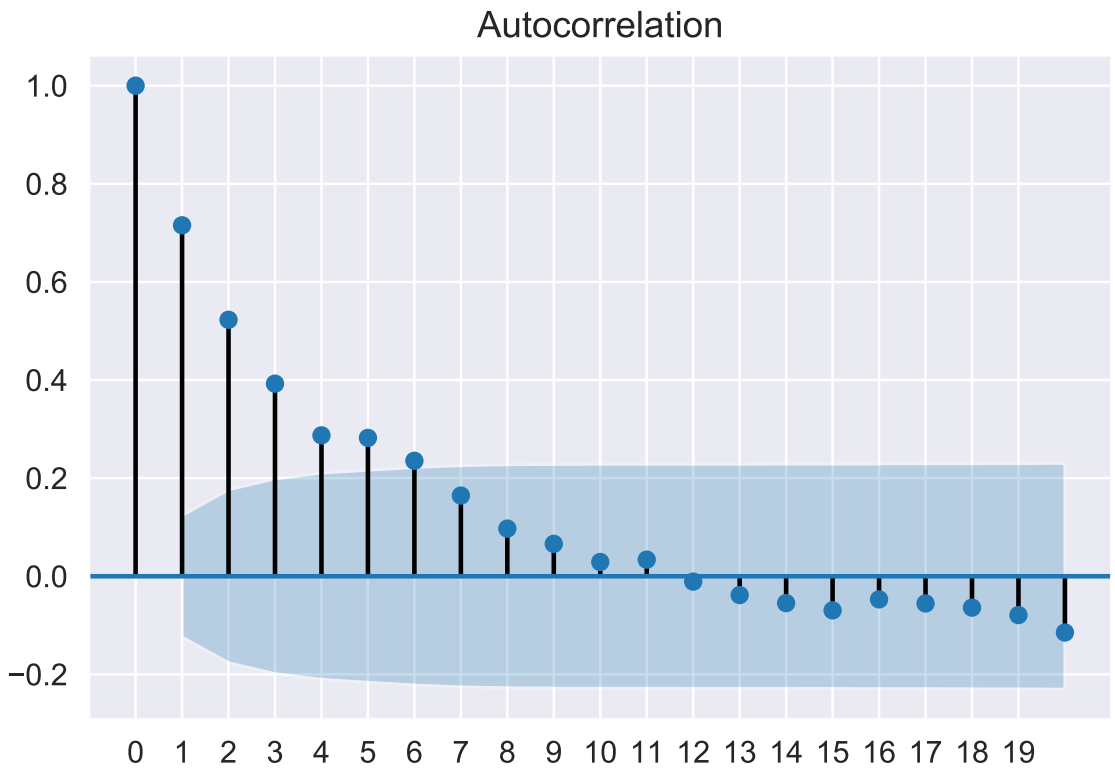


Even a simple model can generate apparently complicated structure.

AR — Python Implementation . . . ACF

Stationary.ipynb In[6]:

```
plot_acf(y, lags=20)
plt.xticks(range(20))

filename = "../pic/ar_%s_acf.pdf" % param_to_string(ar)
plt.savefig(filename, bbox_inches="tight")
plt.show()
```



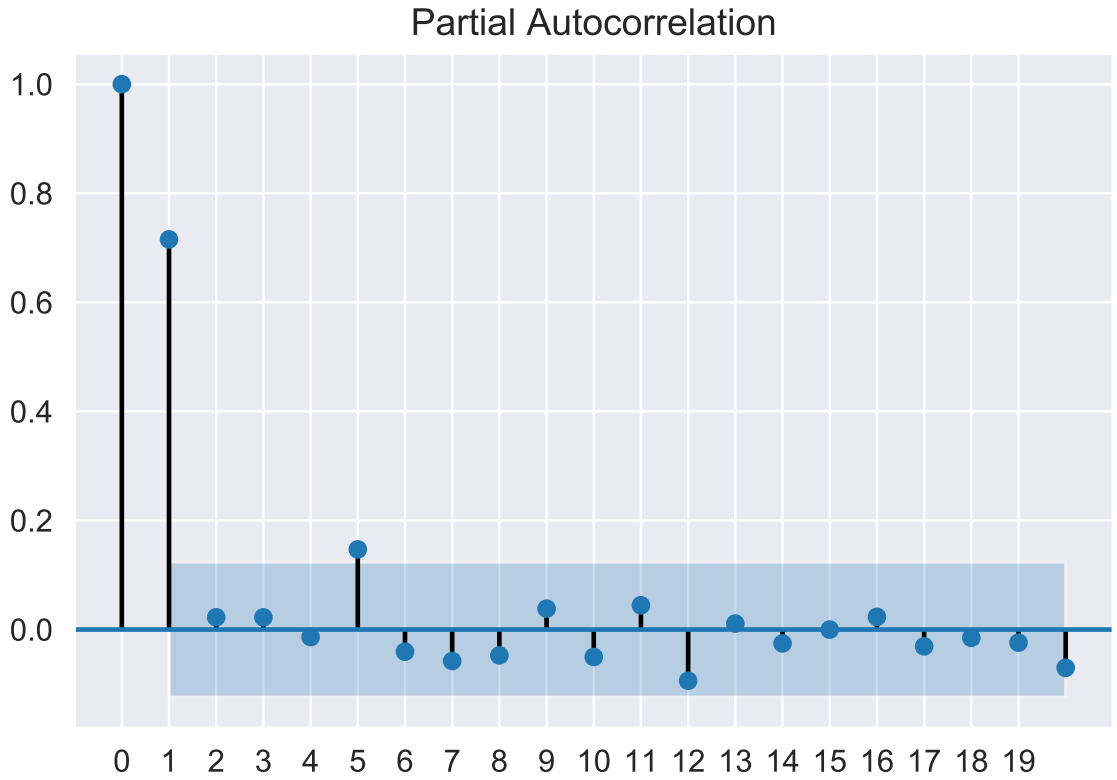
The acf shows an exponential decaying of strength of lag terms.
Shaded area indicates 95% confidence interval of significant lags.

We would expect the ACF for the AR(k) time series to be strong to a lag of k and the inertia of that relationship would carry on to subsequent lag values, trailing off at some point as the effect was weakened.

AR — Python Implementation . . . PACF

Stationary.ipynb In[7]:

```
plot_pacf(y, lags=20)
plt.xticks(range(20))

filename = "../pic/ar_%s_pacf.pdf" % param_to_string(ar)
plt.savefig(filename, bbox_inches="tight")
plt.show()
```



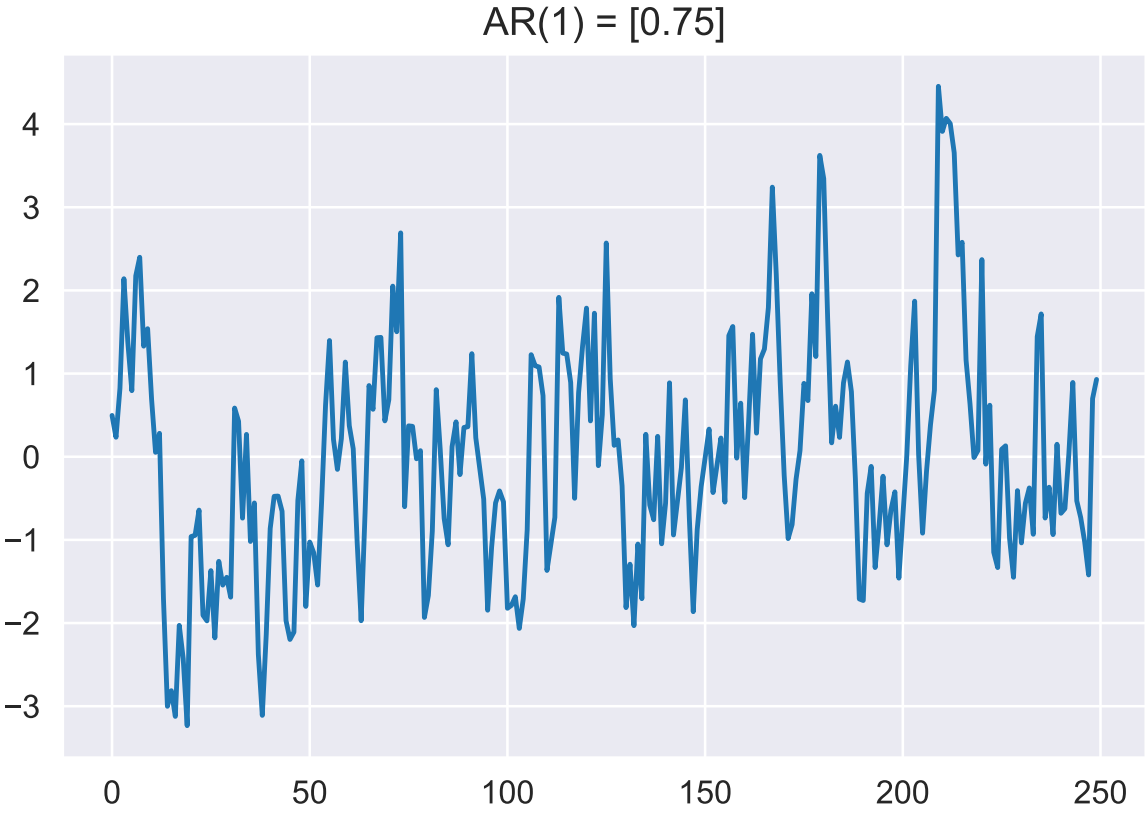
The pacf shows an strong lag at 1. Lag at 5 is due to noise.

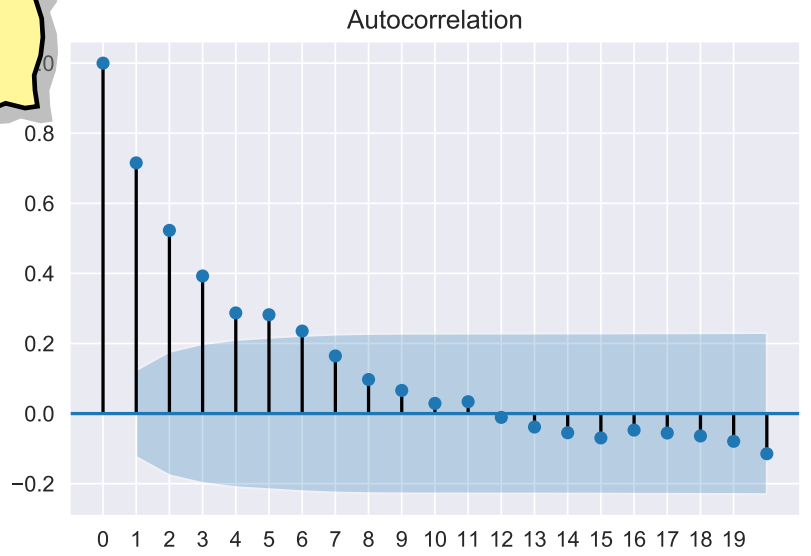
Shaded area indicates 95% confidence interval of significant lags.

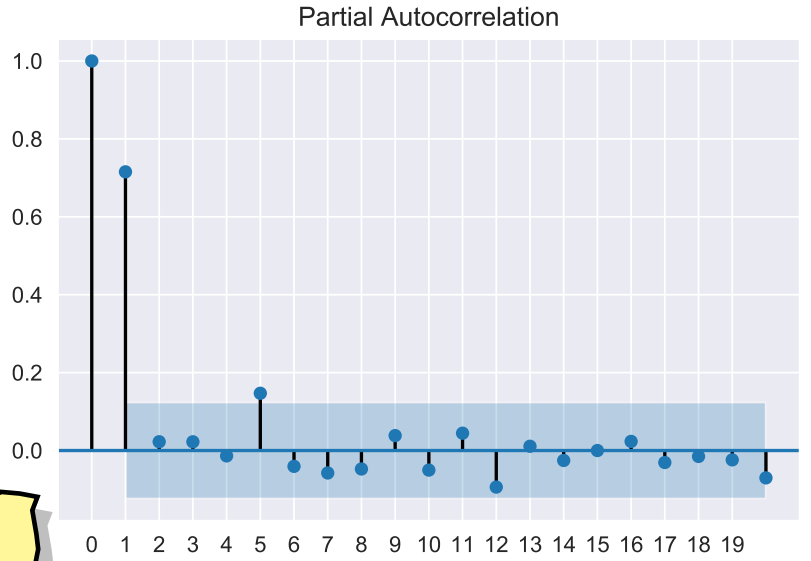
The PACF only describes the direct relationship between an observation and its lag. This would suggest that there would be no correlation for lag values beyond k.

AR — Python Implementation . . . Identification

From the acf it looks like an AR(p), but what is the value for p?







The pacf correctly (if we ignore the lag at 5) that we have an AR(1) model.

AR — Python Implementation . . . Modelling

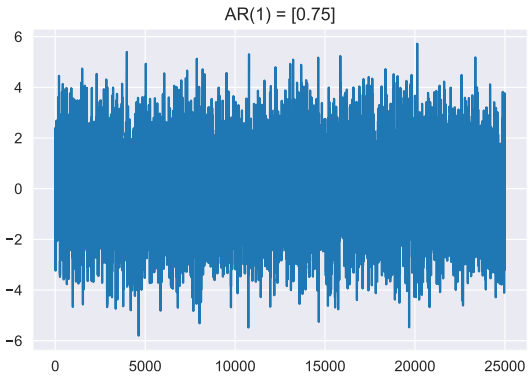
To model we need to supply data and values for the order of the model. Here AR(1) and MA(0).

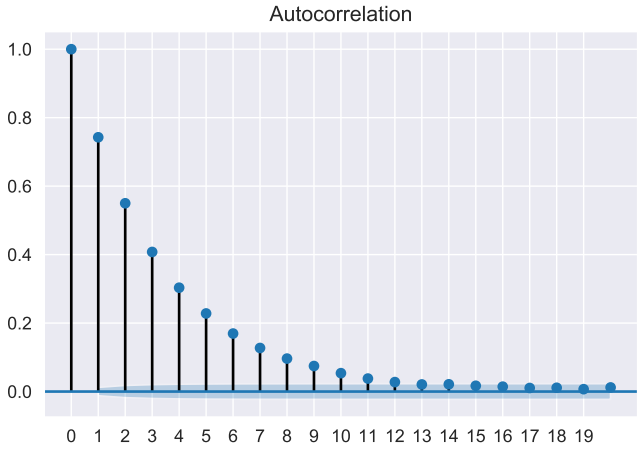
Stationary.ipynb In[8]:

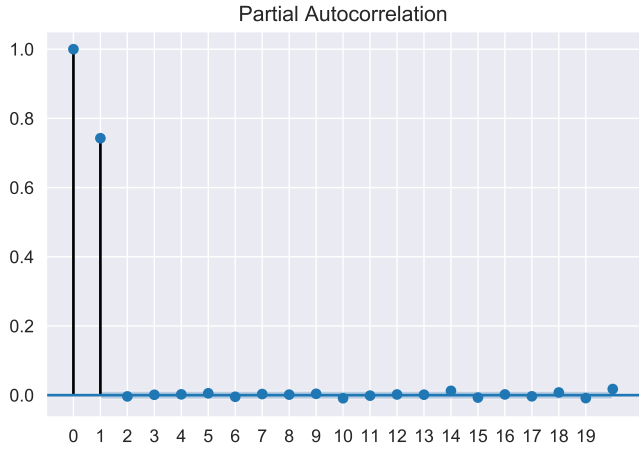
```
model = sm.tsa.ARMA(y, (1, 0)).fit(trend='nc', disp=0)
model.params
```

```
array([0.71435336])
```

Rerunning with 25,000 (instead of 250) points we have








```
array([0.74267818])
```

AR(2) with parameters $[0.75, -0.9]$

Lets start with an AR(2) with parameters $\alpha_1 = 0.75$, $\alpha_2 = -0.9$, i.e.,

$$X_t = 0.75X_{t-1} - 0.9X_{t-2} + \epsilon_t$$

All code is as before . . .

Stationary.ipynb In[15]:

```
ar = np.array([1, -0.75,0.9])
ma = np.array([1])
```

. . . BLAH . . . BLAH . . .

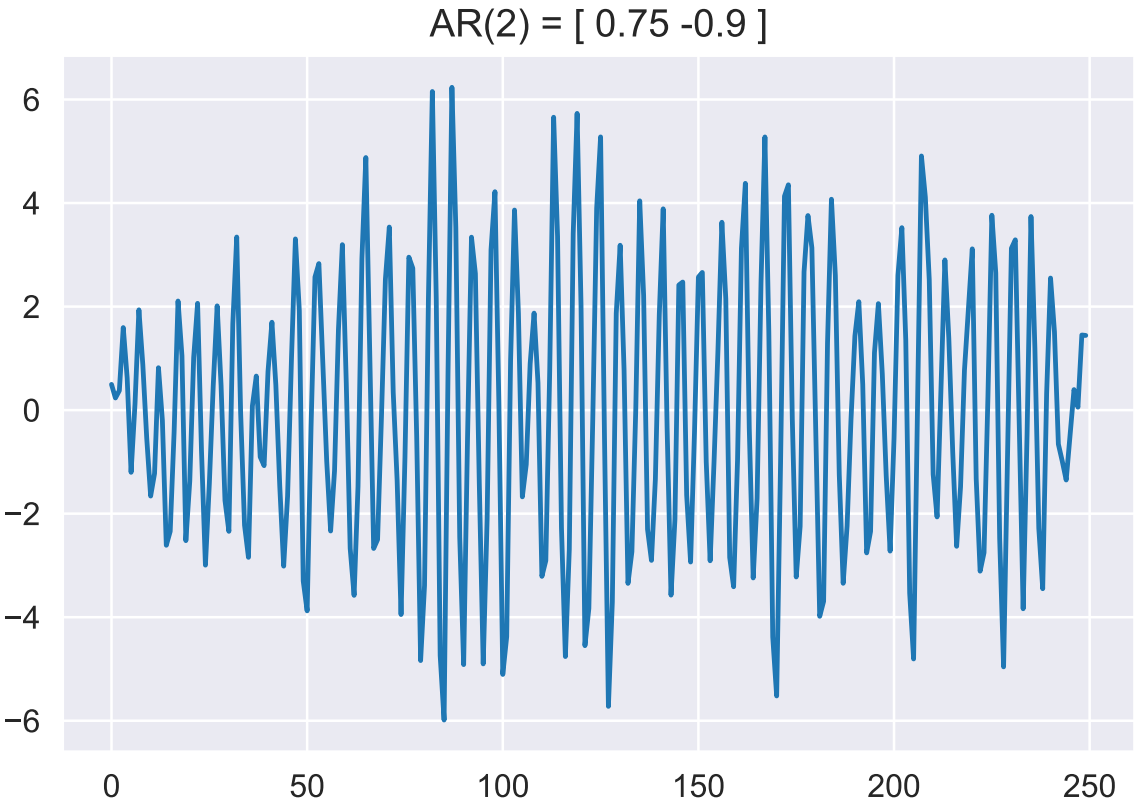
Stationary.ipynb In[20]:

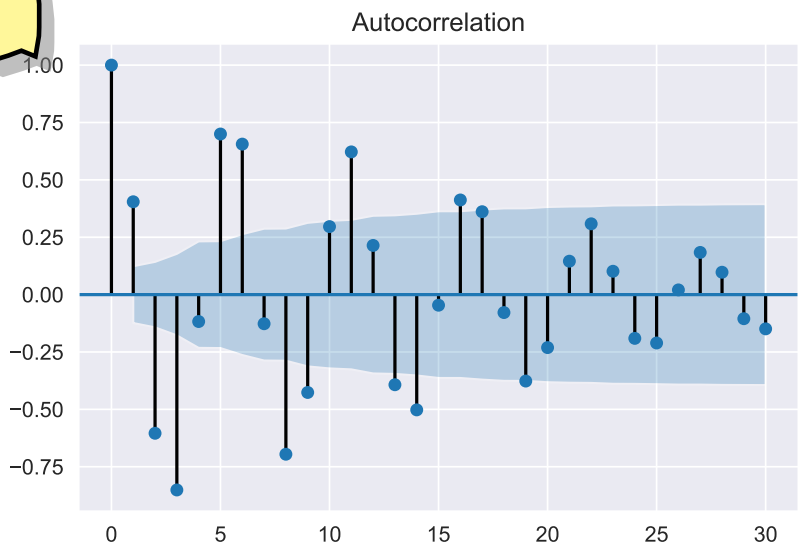
```
model = sm.tsa.ARMA(y, (2, 0)).fit(trend='nc', disp=0)
model.params
```

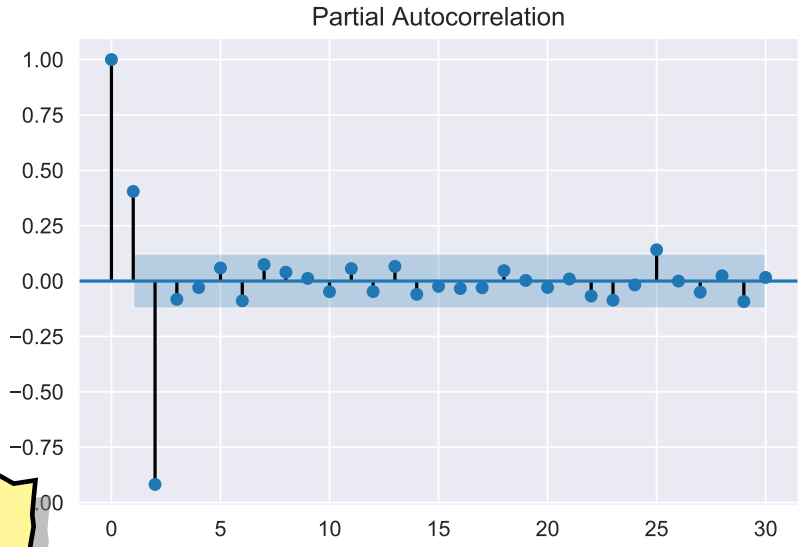
```
array([ 0.77366032, -0.91287896])
```

AR(2) with parameters $[0.75, -0.9]$

From the acf, I see a strong lag effects, but what?







The pacf shows correct structure (if we ignore the lag at 25).

Moving Average (MA)

In a moving average model we assume that the current value of the series is a weighted sum of past white noise terms:

Definition 8 (Moving Average (MA))

$$X_t = \beta\epsilon_{t-1} + \epsilon_t$$

where ϵ_t is a white noise time series

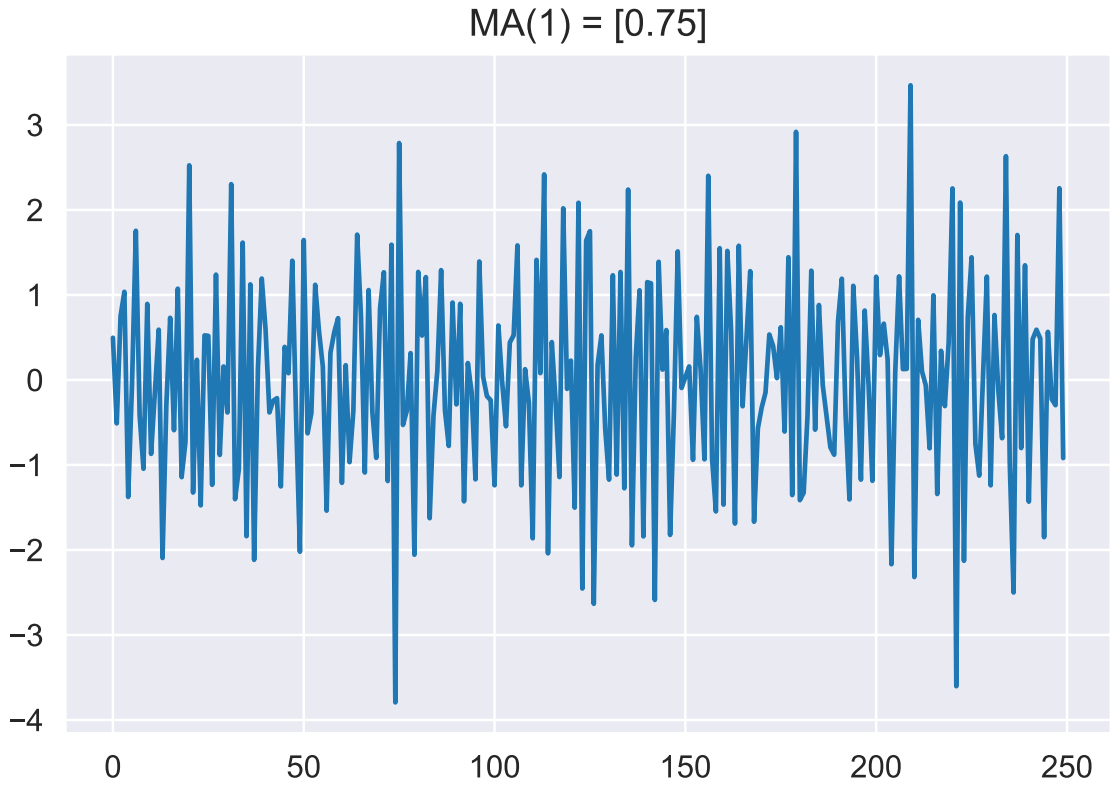
- Here the lag of the moving average is 1.
- We can think of the white noise series as being innovations or shocks: new stochastically uncorrelated information which appears at each time step, which is combined with other innovations (or shocks) to provide the observable series X_t.
- More generally we could have a moving average model of order q, an MA(q) model, defined by

$$X_t = \sum_{k=1}^{q} \beta_k \epsilon_{t-k} + \epsilon_t$$

MA — Python Implementation . . . Visualisation

Stationary.ipynb In[23]:

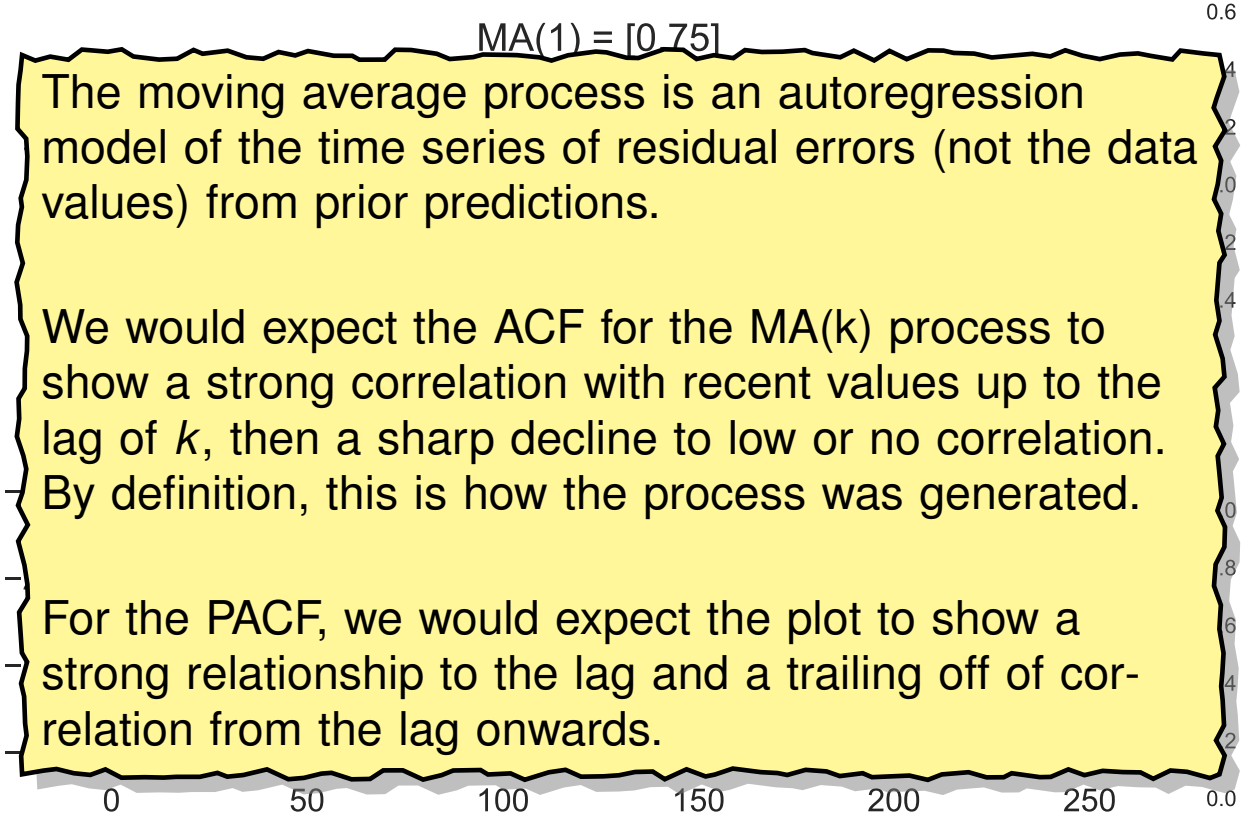
```
plt.plot(y)
plt.title("MA(1) = %s" % -ma[1:])

filename = "../pic/ma_%s.pdf" % param_to_string(ma)
plt.savefig(filename, bbox_inches="tight")
plt.show()
```



Again, looking at the signal is of little use.

MA — Python Implementation . . . Identification

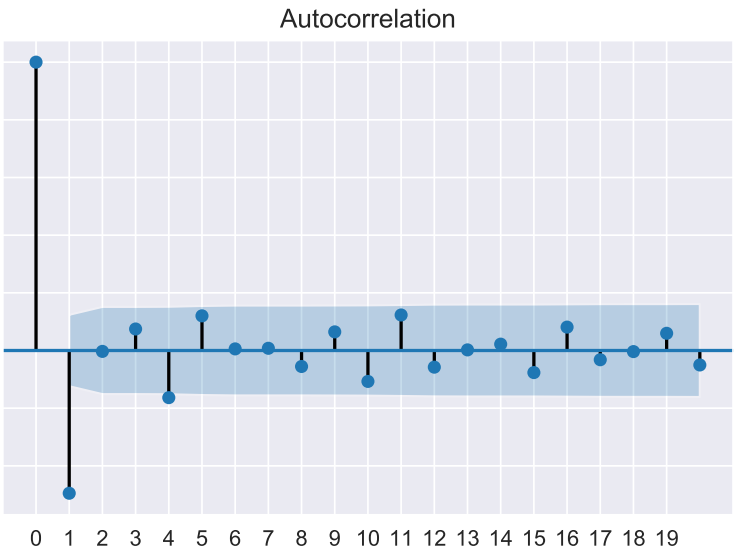
From the acf it looks like an MA(1).

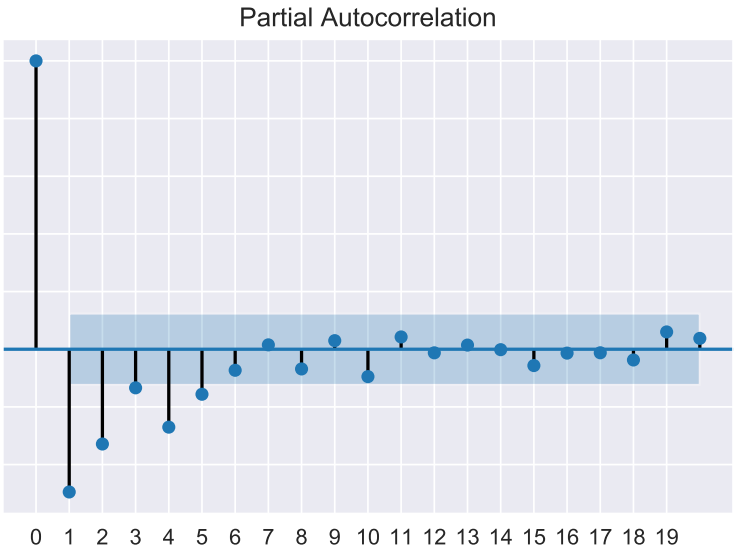


The moving average process is an autoregression model of the time series of residual errors (not the data values) from prior predictions.

We would expect the ACF for the MA(k) process to show a strong correlation with recent values up to the lag of k, then a sharp decline to low or no correlation. By definition, this is how the process was generated.

For the PACF, we would expect the plot to show a strong relationship to the lag and a trailing off of correlation from the lag onwards.





The pacf has slow (in abs) decay of lag strength. so we have a MA(q) model, but what is q?

MA — Python Implementation . . . Modelling

To model we need to supply data and values for the order of the model. Here AR(0) and MA(1).

Stationary.ipynb In[26]:

```
model = sm.tsa.ARMA(y, (0, 1)).fit(trend='nc', disp=0)
model.params
```

```
array([-0.77899406])
```

AR(p) vs MA(q)

- for an MA(q) series,
  - the ACF drops rapidly beyond lag q.
  - the PACF decays slowly.
- for an AR(p) series,
  - the ACF decays slowly.
  - the PACF drops rapidly beyond lag p.

Autoregressive Moving Average Process (ARMA)

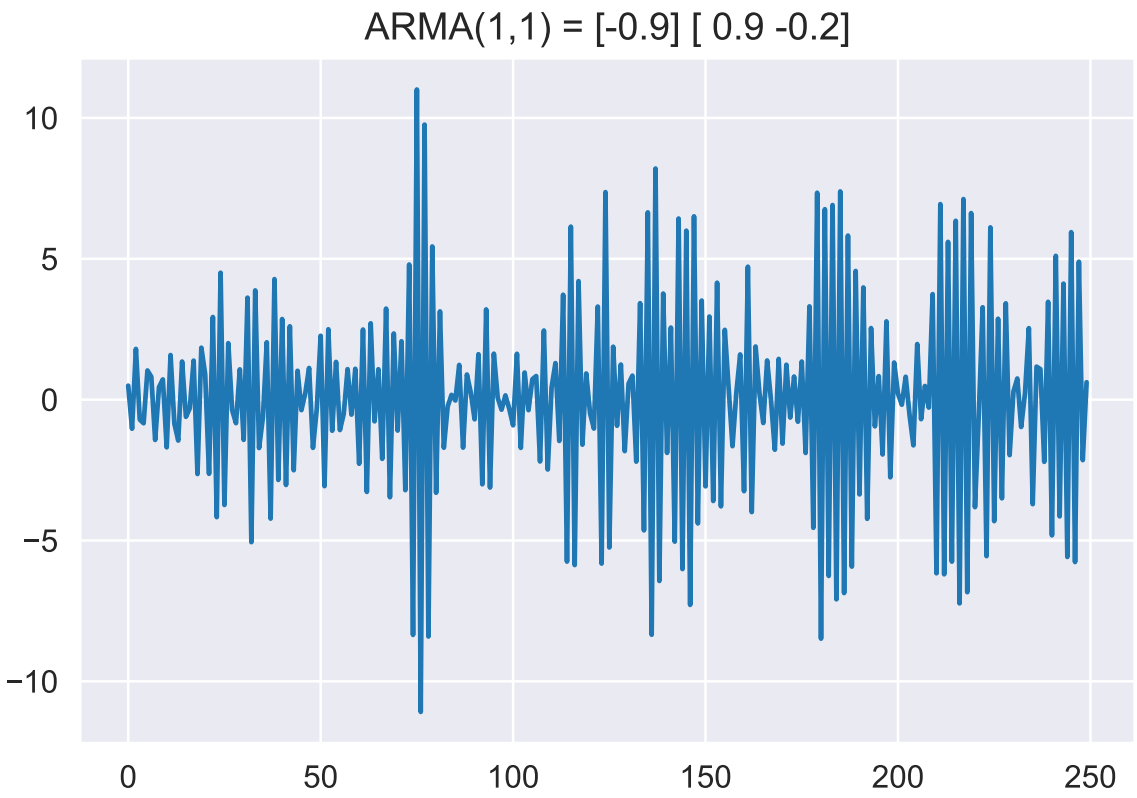
An autoregressive moving average process ARMA(p, q) is a combination of AR(p) and MA(q) models, defined by

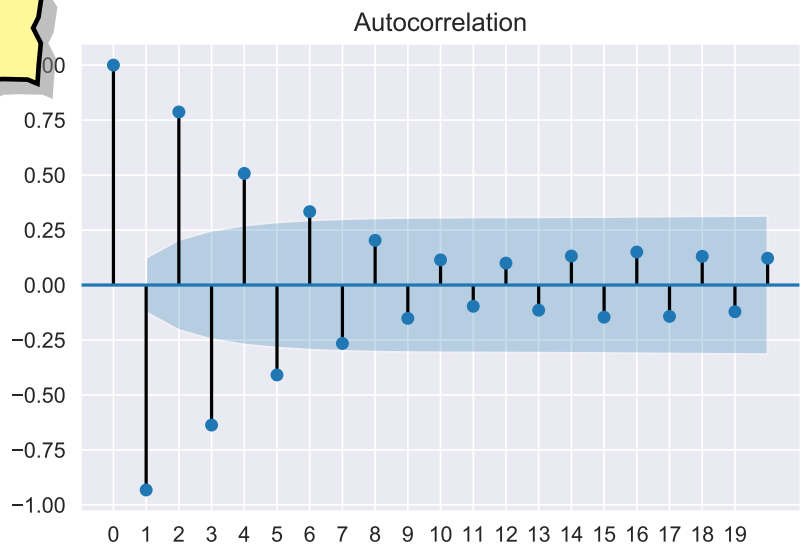
$$X_t = \sum_{k=1}^{p} \alpha_k X_{t-k} + \sum_{k=1}^{q} \beta_k \epsilon_{t-k} + \epsilon_t$$

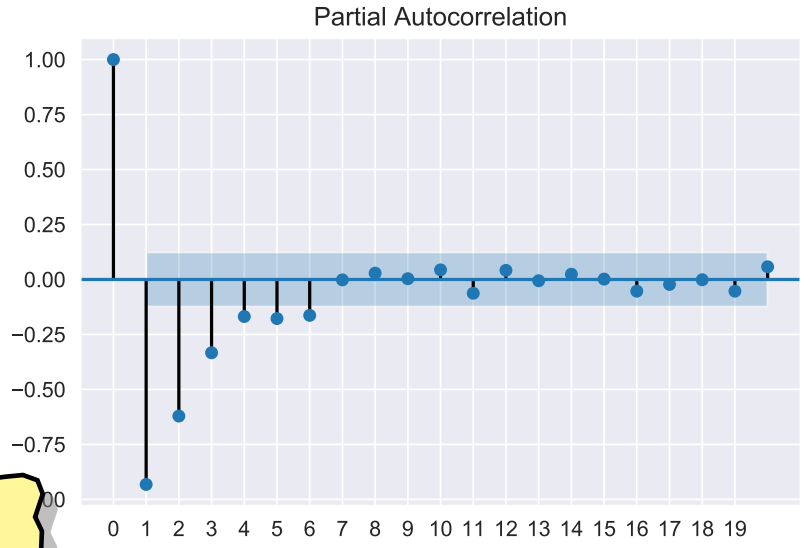
- The value of ARMA processes lies primarily in their ability to approximate a wide range of second-order behaviour using only a small number of parameters.
- However, identifying the order from the acf and pacf becomes more difficult due to influence of both AR and MA components.
- While often replaced by more recent techniques (LSTMs, ANNs, SVMs) the ARIMA models have advantages:
  - ARIMA models are more interpretable, which means that their forecasts can be more intuitively explained.
  - ARIMA predictions naturally produce confidence intervals because they are regressive.

ARMA(1,2) $[-0.9] \times [0.9, -0.2]$

From the acf it looks like an AR(p), but what is the value for p?





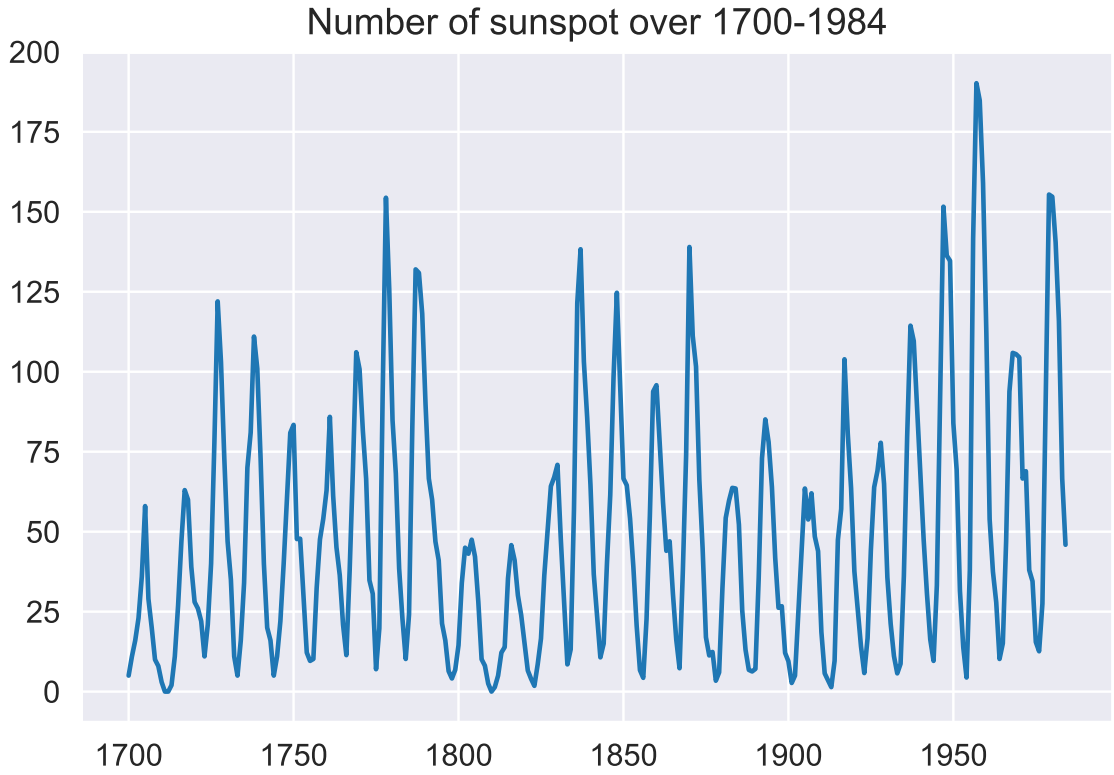


The pacf has slow (in abs) decay of lag strength. so we have a MA(q) model, but what is q?

Sunspots

Two data files:

- sunspot.dat observations by year from 1700 to 1985, used for fitting.
- sunspot2.dat observations by year from 1700 to 1989, last 4 years used for fitting.



Sunspots.ipynb In[2]:

```
data = np.loadtxt("src/sunspot.dat")
year = range(1700,1984+1)
expected = np.loadtxt("src/sunspot2.dat")[-5:]
```

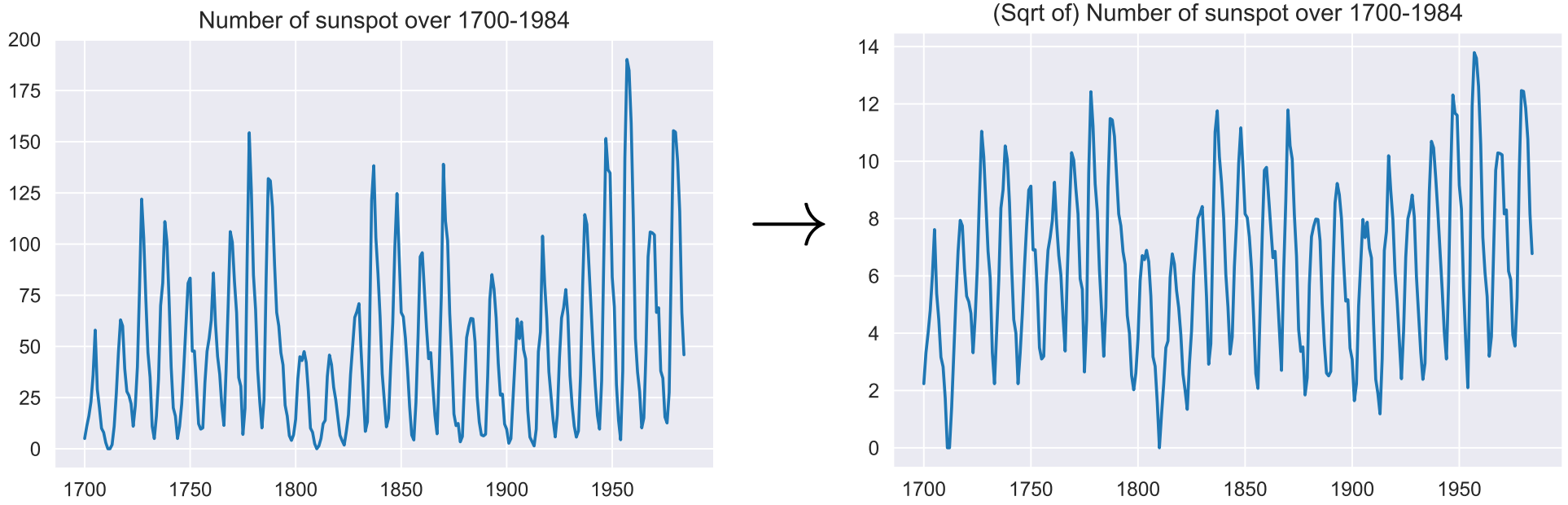
Sunspots — Transformation

Data has no obvious trend and seasonal effect but to ensure constant variance we will work with the square root of the observations.

Sunspots.ipynb In[4]:

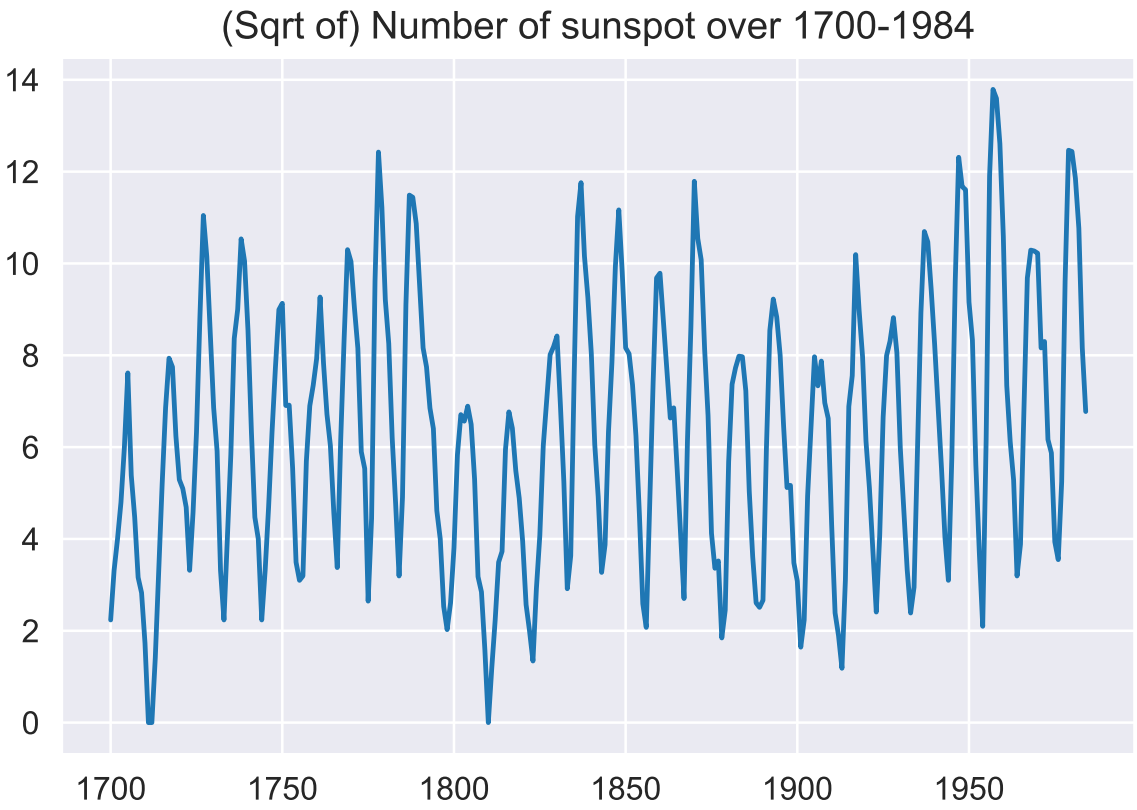
```
y = np.sqrt(data)
```

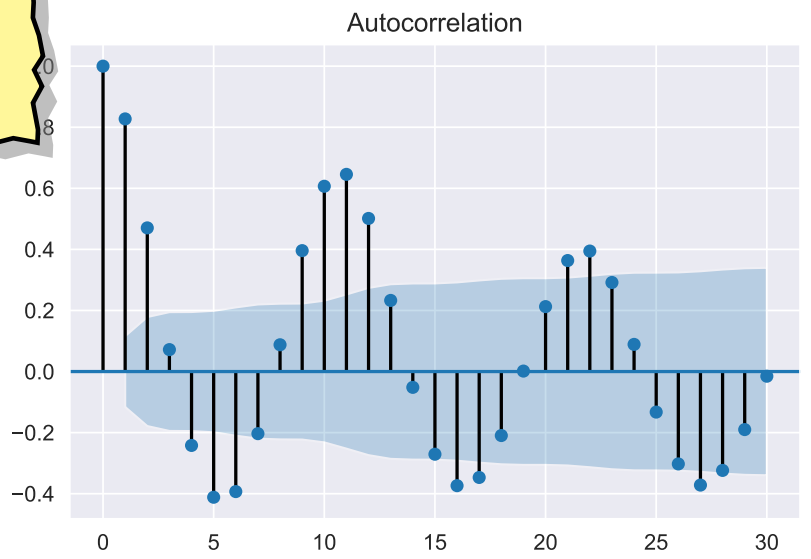
- Build model and predict using *y* and need to square output to compare with actual observations.

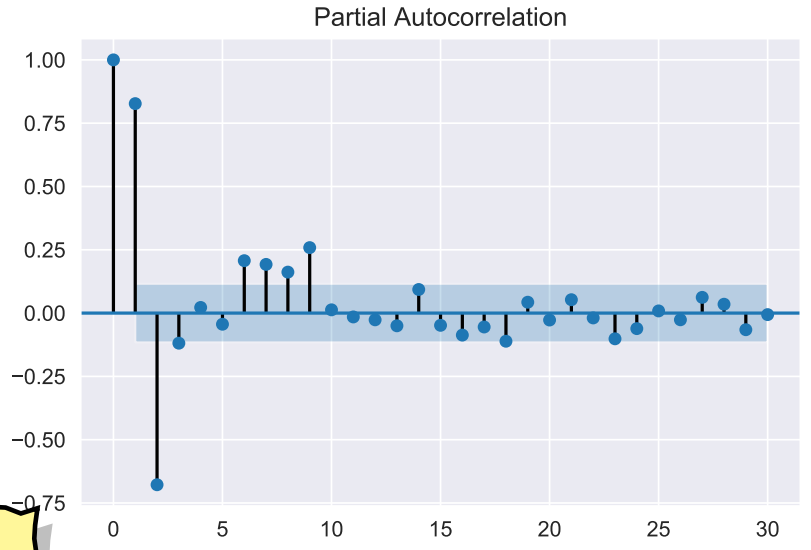


Suspects: ACF and PACF

ACF shows a periodic effect (period 11?) and slow decay AR(P)?
There is a periodic component that we have not dealt with at all.







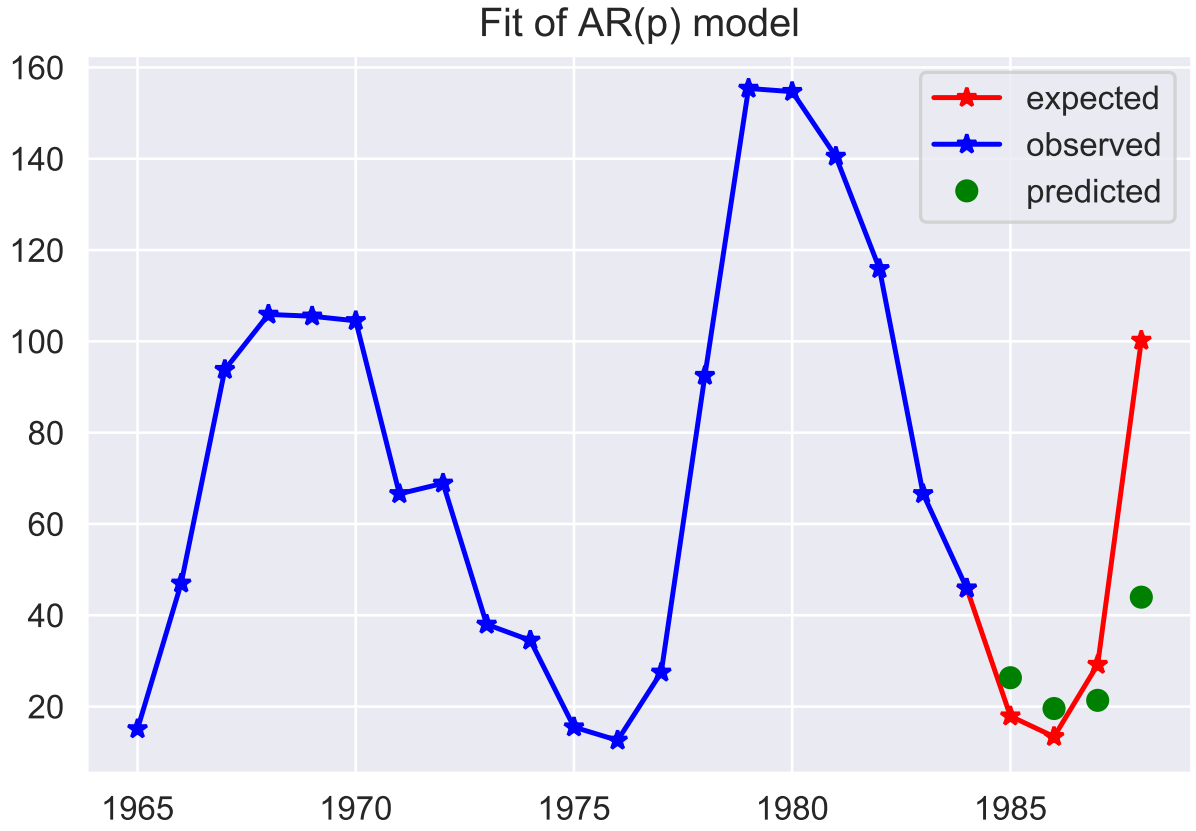
The pacf shows correct rapid cutoff AR(2) or AR(9).

Sunspots — Fit of AR(p)

Sunspots.ipynb In[8]:

```
from statsmodels.tsa.ar_model import AR
model = AR(y).fit()
yhat = model.predict(len(y),len(y)+len(expected)-2)
print(yhat**2)
```

```
[26.3173986  19.57096978 21.36519288 43.96334534]
```



Predictions capture change in direction but not magnitude.

Note AR fits arbitrary order, p (here $p = 16$).

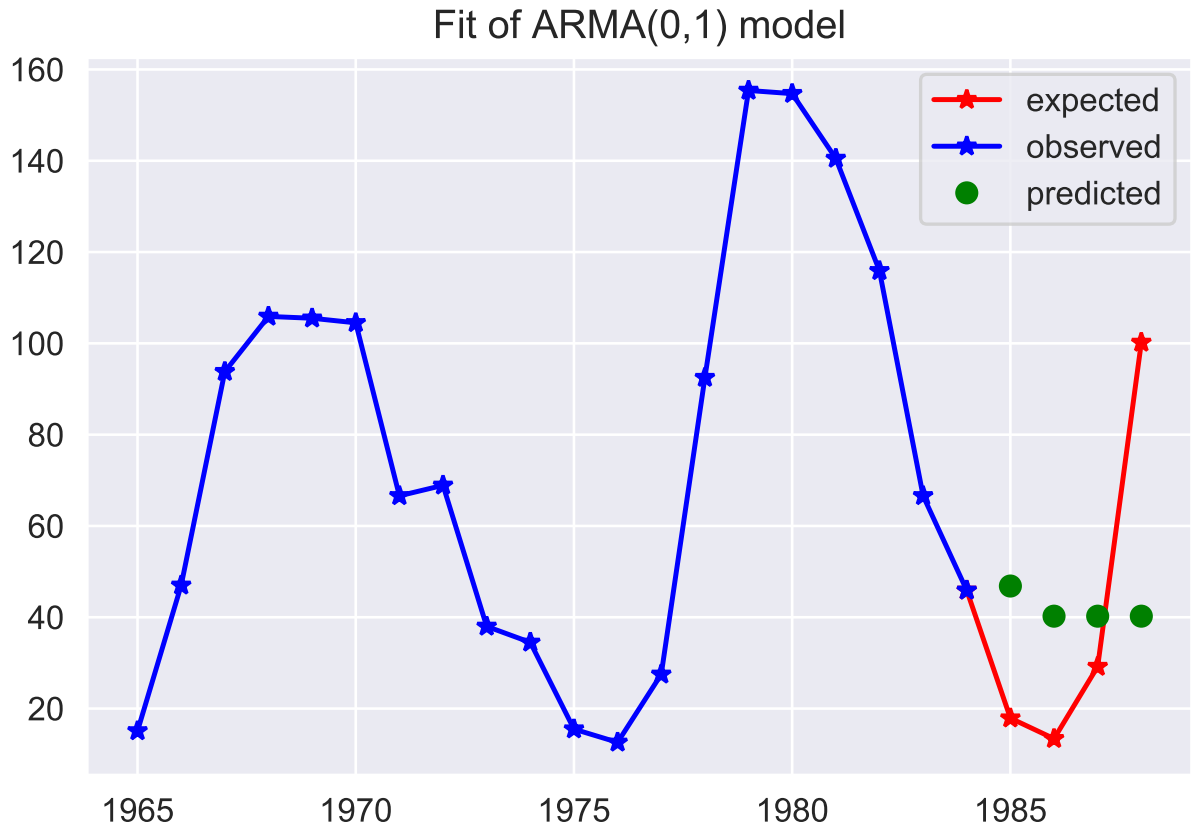
Could also do this using ARMA(16,0).

Sunspots — Fit of ARMA(0,1)

Sunspots.ipynb In[11]:

```
from statsmodels.tsa.arima_model import ARMA
model = ARMA(y, order=(0, 1)).fit(disp=False)
yhat = model.predict(len(y),len(y)+len(expected)-2)
print(yhat**2)
```

```
[46.8412104  40.23836885 40.23836885 40.23836885]
```



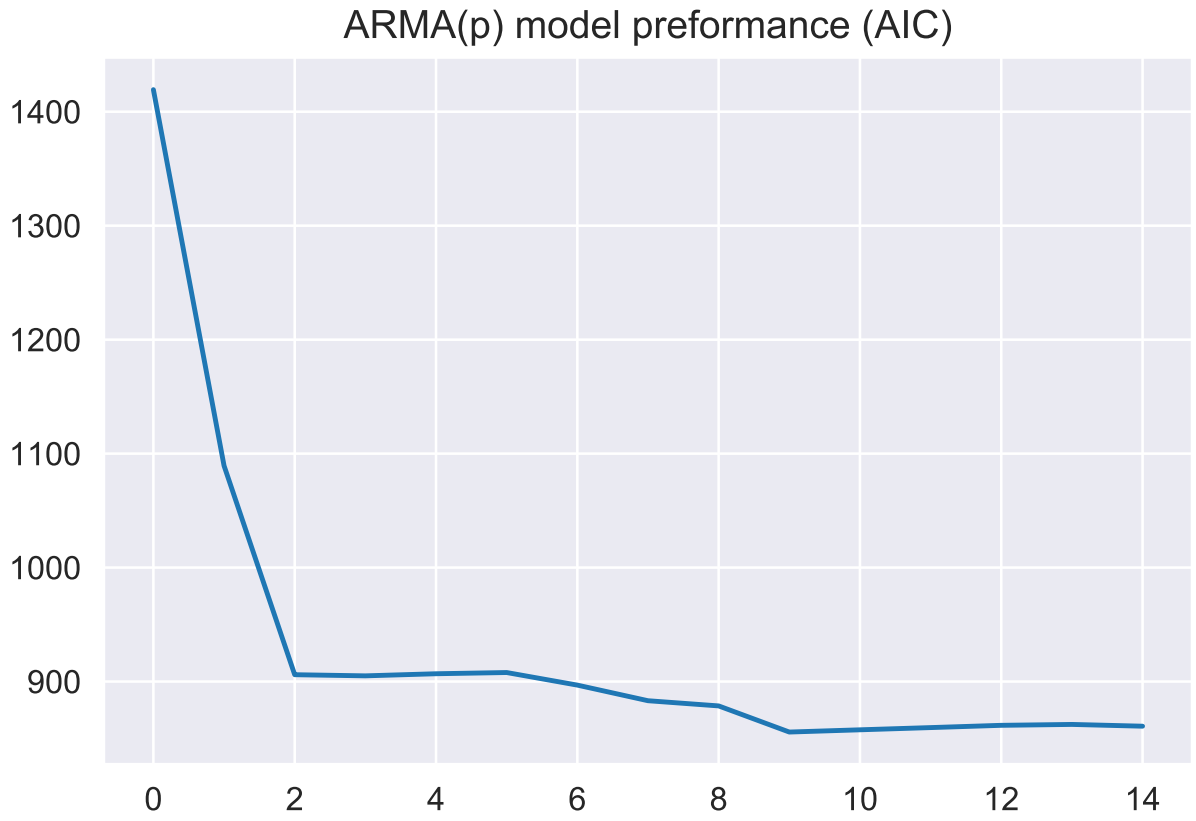
Model is no good here. Why would it be?
Just fitted it to show difference with AR.

Sunspots — Effect of order p in AR model?

Do a parameter sweep over p, using a metric (aic):

Sunspots.ipynb In[15]:

```
aic = [ARMA(y, order=(p,0)).fit(disp=0).aic for p in range(15)]
```



About $p = 9$ seems to be optimal.

Note: *Modeling and Periodicity Analysis of Sunspot Time Series 1700-2015* found AR(11) to be optimal.
DOI: 10.4172/2168-9679.1000385

And need to go back and look at that periodic component also . . .

Sunspots — Quality of ARMA(0,9) Model I

Sunspots.ipynb In[17]:

```
model = ARMA(y, order=(9,0)).fit(disp=0)
print(model.summary())
```

ARMA Model Results

Dep. Variable:	y	No. Observations:	285
Model:	ARMA(9, 0)	Log Likelihood	−416.847
Method:	css-mle	S.D. of innovations	1.037
Date:	Sat, 23 Mar 2019	AIC	855.694
Time:	15:30:00	BIC	895.872
Sample:	0	HQIC	871.800

small is better

	coef	std err	z	P>|z|	[0.025	0.975]
const	6.3296	0.534	11.844	0.000	5.282	7.377
ar.L1.y	1.2208	0.057	21.555	0.000	1.110	1.332
ar.L2.y	−0.4791	0.091	−5.259	0.000	−0.658	−0.301
ar.L3.y	−0.1371	0.095	−1.446	0.149	−0.323	0.049
ar.L4.y	0.2600	0.095	2.726	0.007	0.073	0.447
ar.L5.y	−0.2412	0.096	−2.519	0.012	−0.429	−0.053
ar.L6.y	0.0161	0.096	0.168	0.867	−0.172	0.205
ar.L7.y	0.1754	0.096	1.822	0.070	−0.013	0.364
ar.L8.y	−0.2190	0.093	−2.363	0.019	−0.401	−0.037
ar.L9.y	0.2956	0.058	5.124	0.000	0.183	0.409

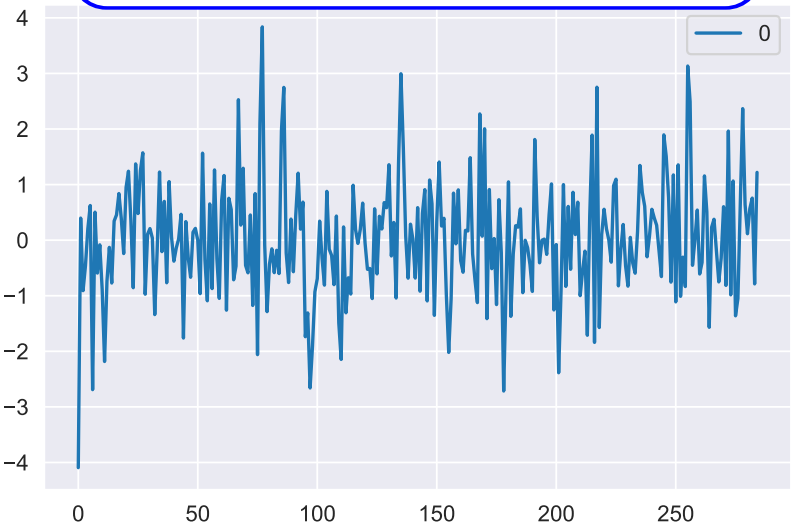
most coefficients are significant
i.e., prob < 0.05,
i.e., zero is not in 95% confidence interval

Sunspots — Quality of ARMA(0,9) Model II

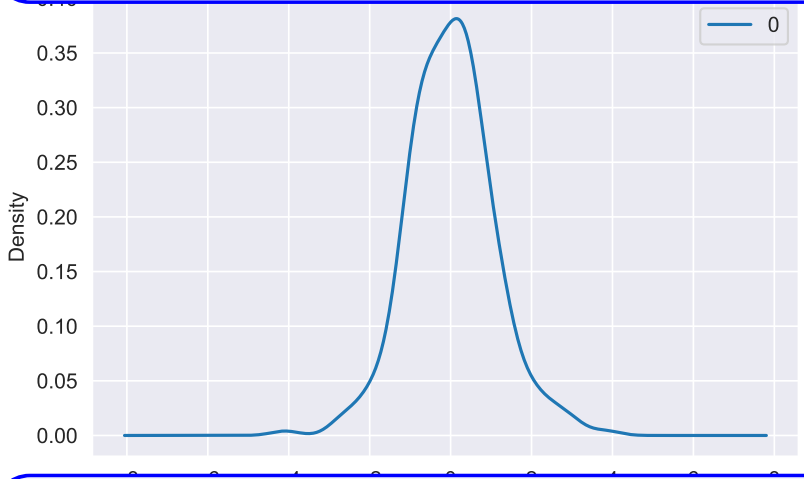
Sunspots.ipynb In[18]:

```
residuals = pd.DataFrame(model.resid)
```

```
residuals.plot()
```




```
residuals.plot(kind='kde')
```



- ✔ No obvious pattern in residual plot.
- ✔ kernel density estimate plot appears near normal with only slight asymmetry.
- ✔ The 5-number summary says same story as kde plot— residuals close to normal, but slight asymmetry is present.

⇒ Model is good but not perfect.

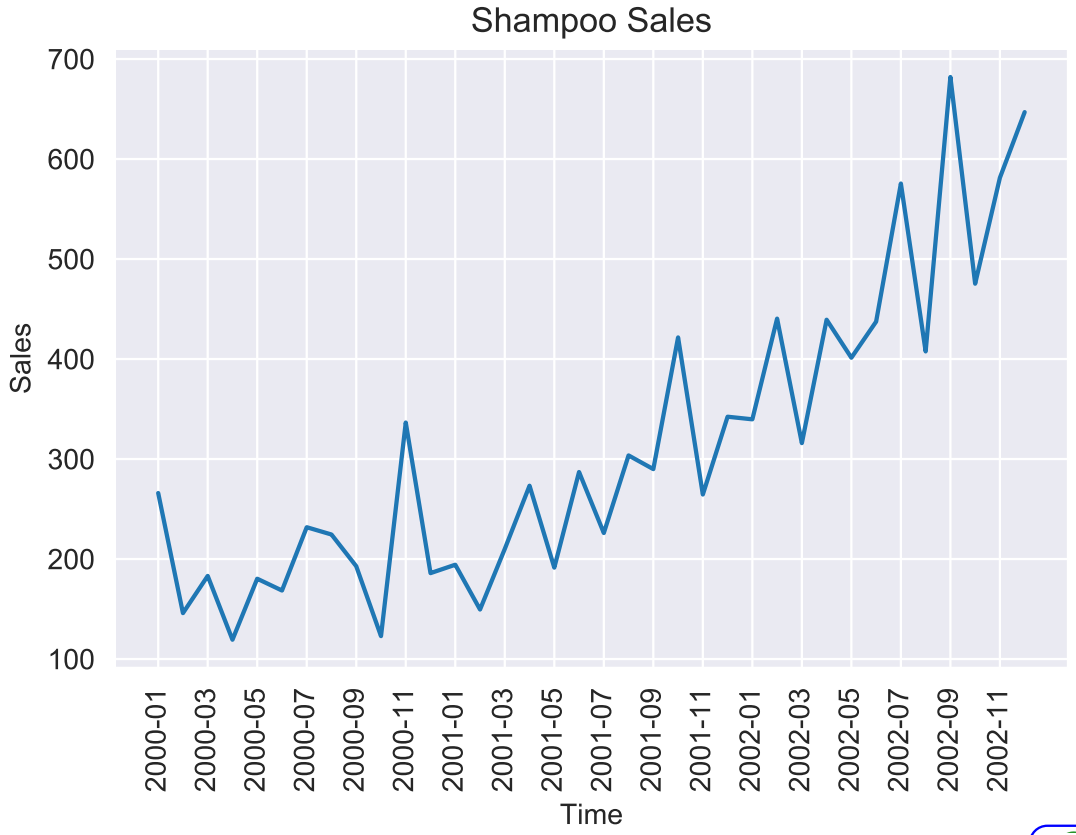
```
print(residuals.describe())
```

count	285.000000
mean	0.012549
std	1.065830
min	−4.093574
25%	−0.692183
50%	0.016957
75%	0.607109
max	3.837647

Shampoo

Monthly observations over three years:

- Upwards (nonlinear?) trend but no seasonal component.
- Series is non-stationary so cannot apply stationary model (ARMA) directly.



Shampoo.ipynb In[5]:

```
t = np.array(df.index).reshape(-1, 1)
y = df.Sales.values
```

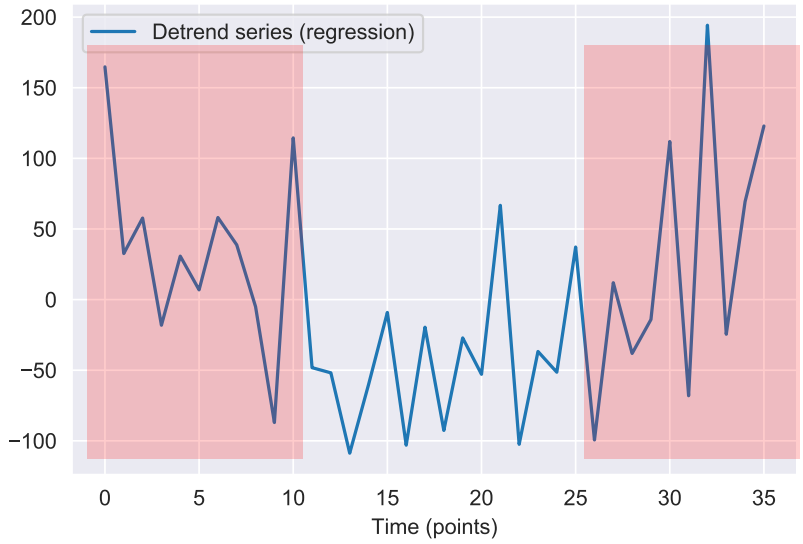
Shampoo — De-trending using Regression

We start by trying a linear regression model ...

Shampoo.ipynb In[6]:

```
from sklearn.linear_model import LinearRegression
trend_model = LinearRegression().fit(t, y)
print('model: %.3f t + %.3f' % (trend_model.coef_[0],
    trend_model.intercept_))
residuals = y - trend_model.predict(t)
```

```
model: 12.079 t + 101.216
```



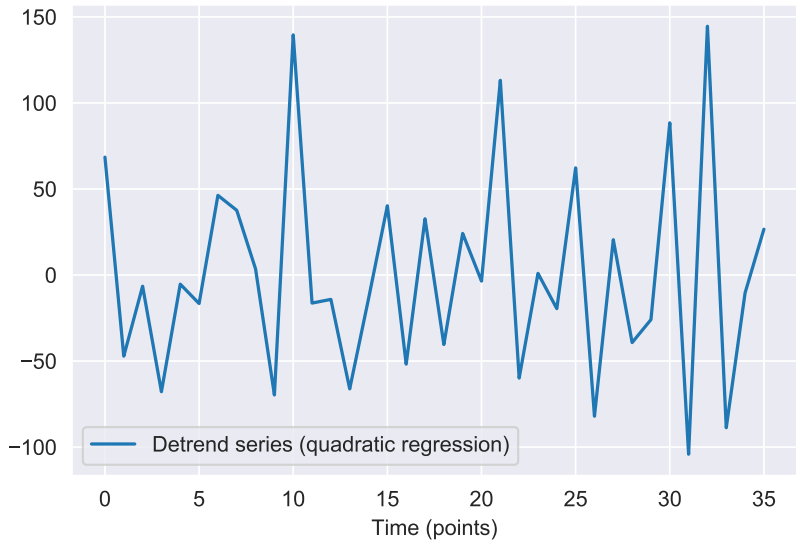
- ✔ On left, trend appears to have been removed.
- ✘ On right, we see a pattern in the residuals (high-low-high) ⇒ polynomial trend?
- ✘ Problem with residuals is not obvious on the left due to large range on vertical axis.

Shampoo — De-trending using Regression (2nd attempt)

To fit a polynomial, we create the polynomial features we want and apply linear regression as normal . . . (note the poor man's scaling in dividing by max)

Shampoo.ipynb In[14]:

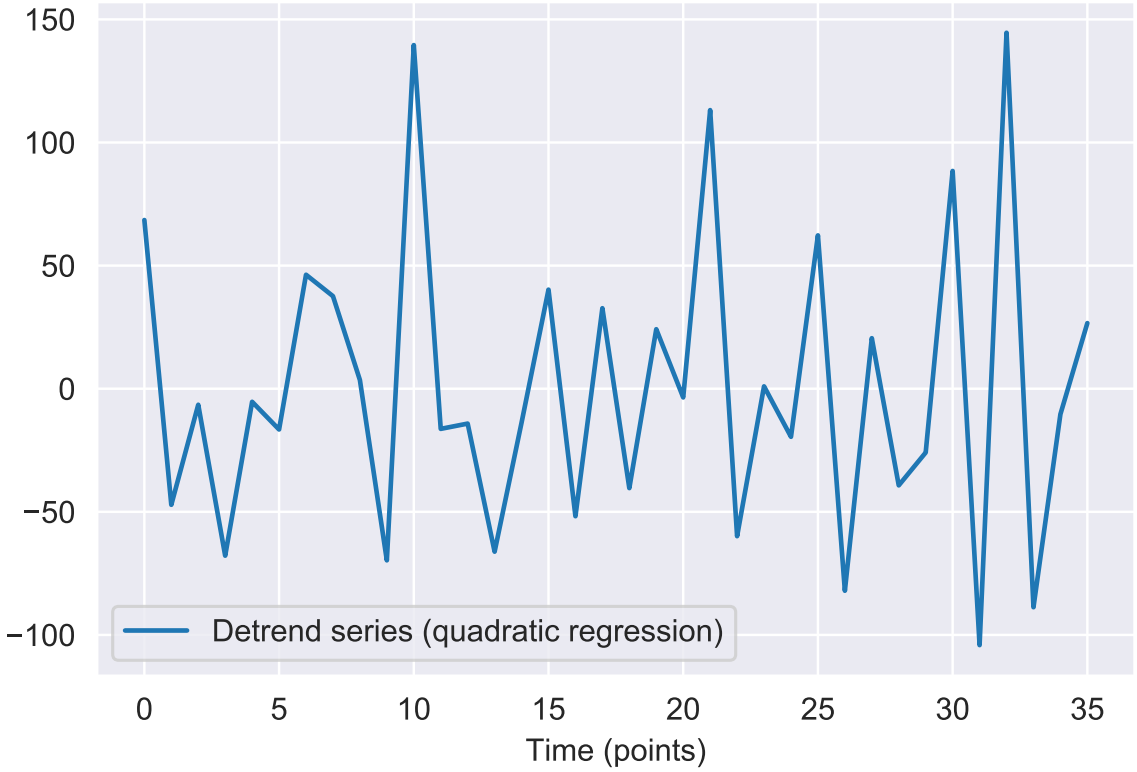
```
from sklearn.preprocessing import PolynomialFeatures
polynomial_features= PolynomialFeatures(degree=2)
t_poly = polynomial_features.fit_transform(t/max(t))
trend_model = LinearRegression().fit(t_poly, y)

residuals = y - trend_model.predict(t_poly)
```

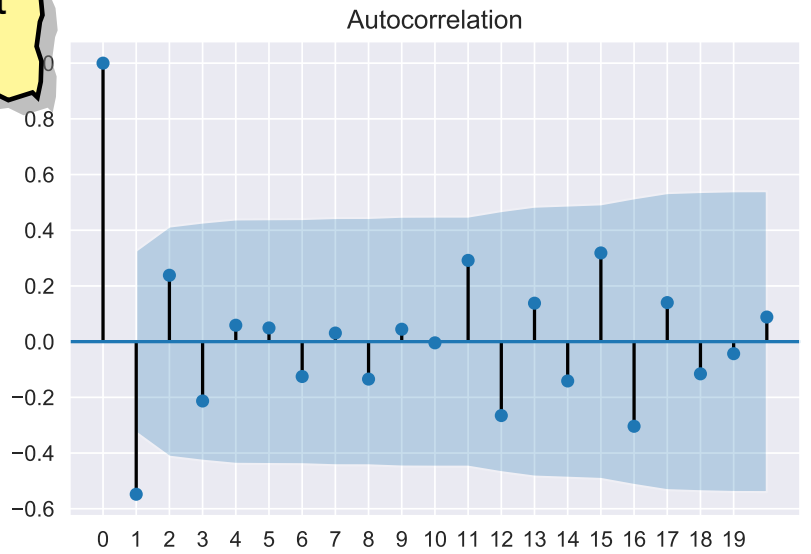


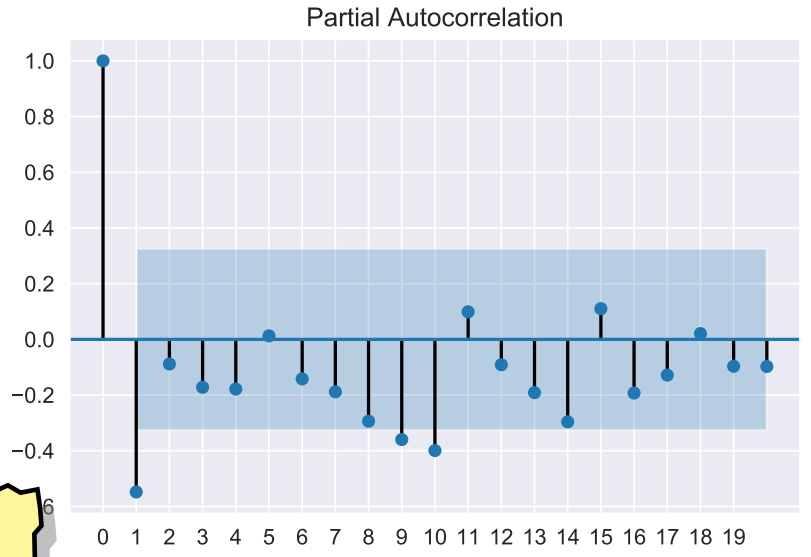
✔ Residuals looks good ⇒ have stationary process.

Shampoo — ACF and PACF

Both ACF and PACF show rapid drop in lags so expect mixture of AR(p) and MA(p).







Expect both p and q to be small,
Do a parameter search. . .

Parameter Sweep of ARMA(p,q) Model

Checking for model ARMA(p,q) with parameters $0 \leq p, q < 4$ we get

Shampoo.ipynb In[19]:

```
p = q = range(0, 4)

opt_aic = np.finfo("float").ma
opt_pq = None

for pq in itertools.product(p,
    try:
        model = ARMA(residuals
        print("%s -> %s" % (pq
        if model.aic<opt_aic:
            opt_aic = model.ai
            opt_pq = pq
    except:
        continue

print("Optimal model: ARMA%s "
```

```
(0, 0) -> 401.2620224547943 6
(0, 1) -> 385.5728256299183
(0, 2) -> 387.1901345734029
(1, 0) -> 390.08471471118935
(1, 1) -> 387.0218300651624
(1, 2) -> 388.7732797725530 5
(1, 3) -> 388.235803422335
(2, 0) -> 391.8043846579636
(2, 1) -> 387.43359609620035
(2, 2) -> 388.94227226708256
(2, 3) -> nan
(3, 0) -> 392.9633469147946
(3, 1) -> 388.35960849425663
(3, 2) -> 385.5349718046033
(3, 3) -> nan
Optimal model: ARMA(3, 2) with AIC=385.5
```

So lets fit a ARMA(3,2) model and see its performance ...

Shampoo — Quality of ARMA(3,2) Model I

Shampoo.ipynb In[20]:

```
model = ARMA(residuals, order=(3,2)).fit(disp=0)
print(model.summary())
```

ARMA Model Results

Dep. Variable:	y	No. Observations:	36
Model:	ARMA(3, 2)	Log Likelihood	−185.767
Method:	css-mle	S.D. of innovations	36.750
Date:	Sun, 24 Mar 2019	AIC	385.535
Time:	15:44:35	BIC	396.620
Sample:	0	HQIC	389.404

	coef	std err	z	P>|z|	[0.025	0.975]
const	0.2518	0.294	0.855	0.399	−0.325	0.829
ar.L1.y	0.8472	0.162	5.228	0.000	0.530	1.165
ar.L2.y	0.1518	0.203	0.747	0.461	−0.246	0.550
ar.L3.y	−0.3983	0.153	−2.599	0.014	−0.699	−0.098
ma.L1.y	−2.0000	0.169	−11.804	0.000	−2.332	−1.668
ma.L2.y	1.0000	0.166	6.025	0.000	0.675	1.325

most coefficients are significant
i.e., prob < 0.05,
i.e., zero is not in 95% confidence interval

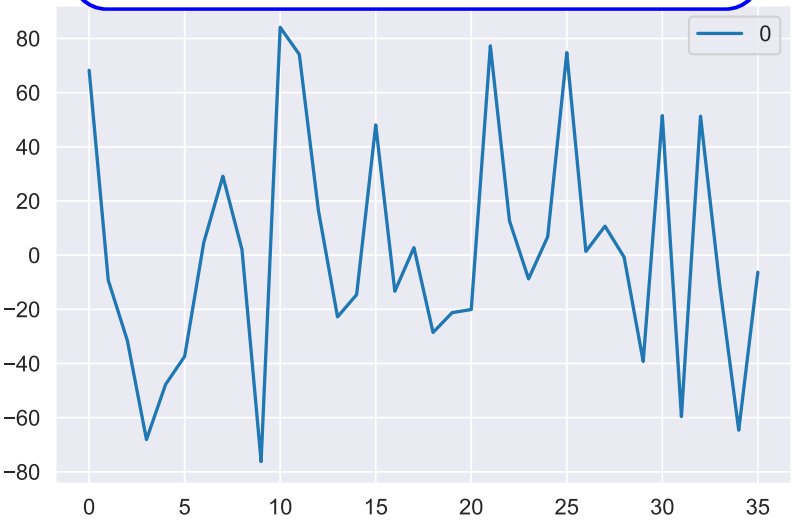
- ✔ Highest order coefficient in AR and MA are significant.
- ✔ AIC of 385 is lower than what we would have got (ARMA(1,2) with AIC=409) if just used linear regression (see notebook).

Shampoo — Quality of ARMA(3,2) Model II

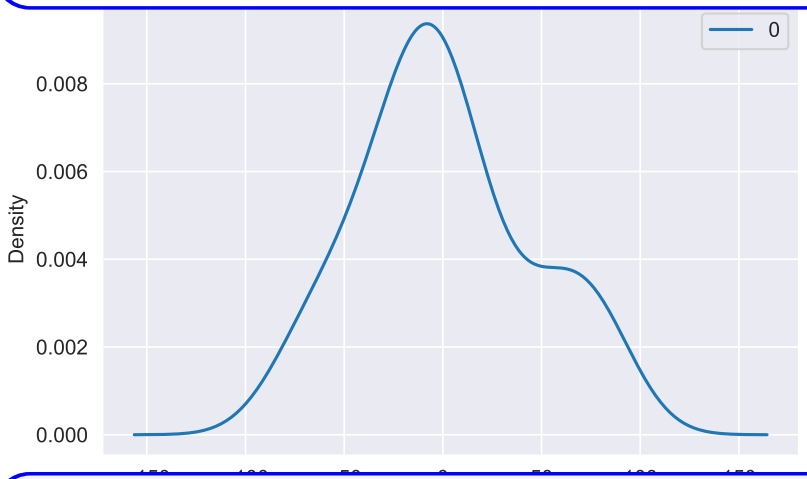
Shampoo.ipynb In[20]:

```
model = ARMA(residuals, order=(3,2)).fit(disp=0)
```

```
residuals.plot()
```




```
residuals.plot(kind='kde')
```




```
print(residuals.describe())
```

count	36.000000
mean	−4.173472
std	64.343005
min	−155.086536
25%	−47.359822
50%	−8.557897
75%	27.487184
max	149.022740

- ✔ No obvious pattern in residual plot.
- ✘ stupid kernel density estimate plot now has a silly ledge on the right.
- ✔ The 5-number summary shows same story as kde plot — distribution is close but is not exactly normal.
- ⇒ Model is best found but is not perfect.

3rd Modelling Attempt — Using Differencing

In lab we talked about removing the trend using differencing. Lets see what happens if we do this . . .

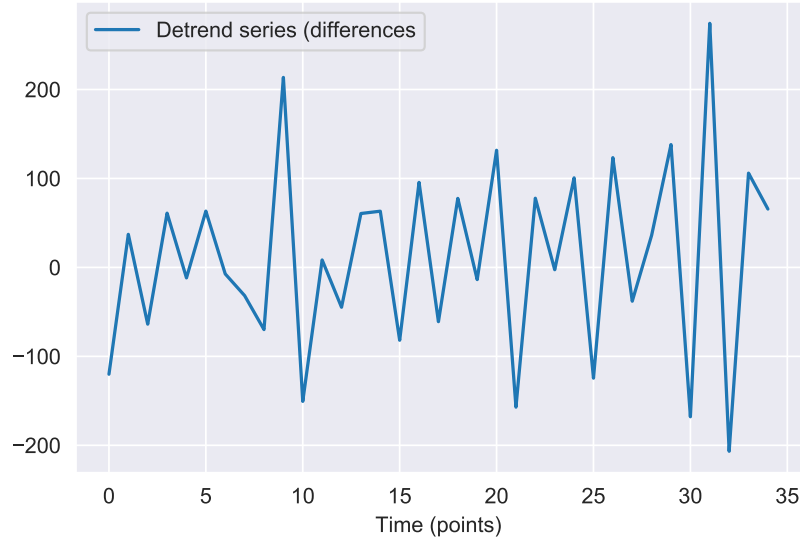
First Difference

The first difference of a sequence x_t is

$$dx_t = x_t - x_{t-1}$$

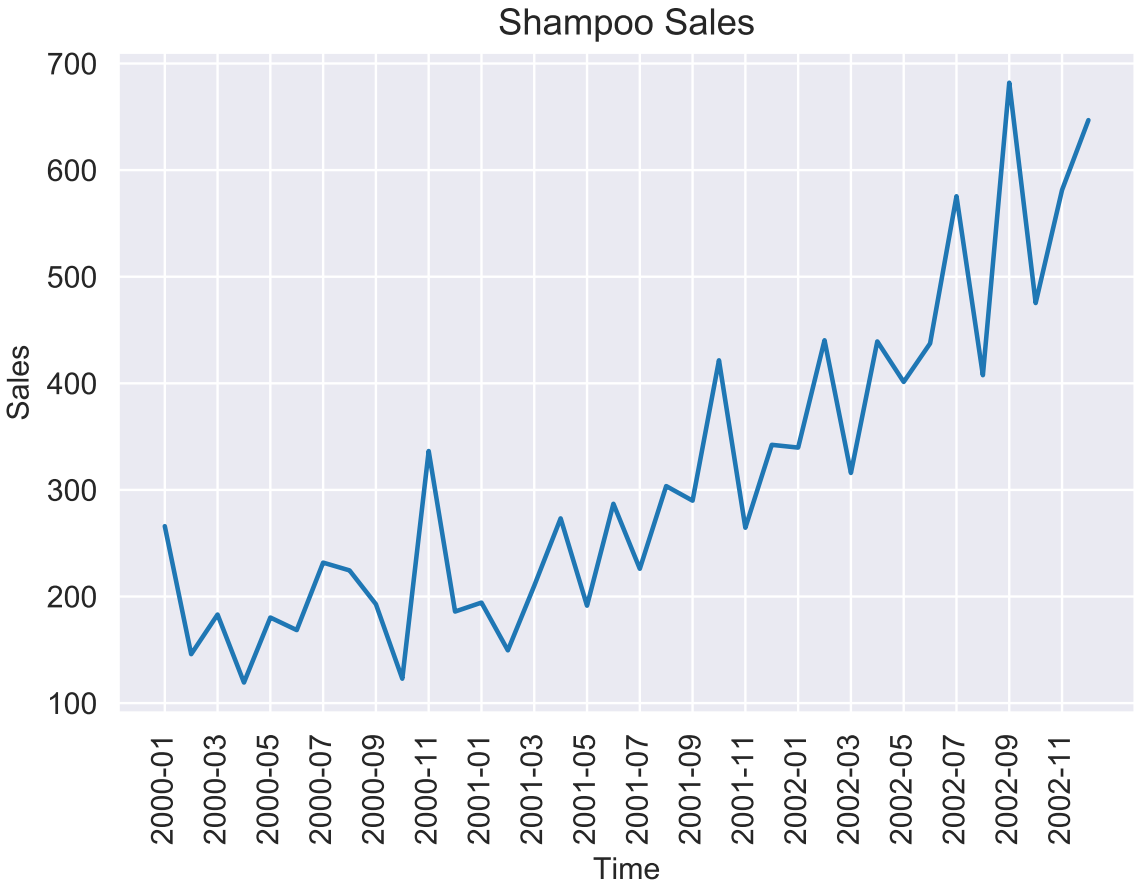
Shampoo.ipynb In[22]:

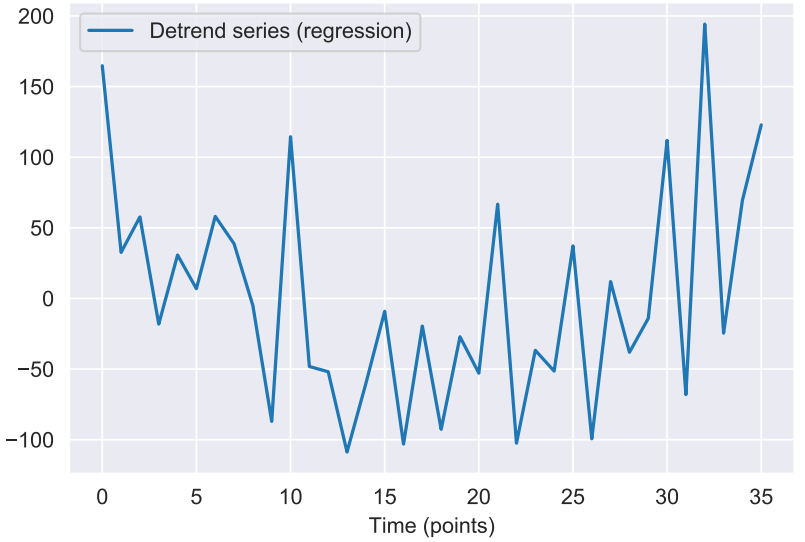
```
dy = np.diff(y)
```

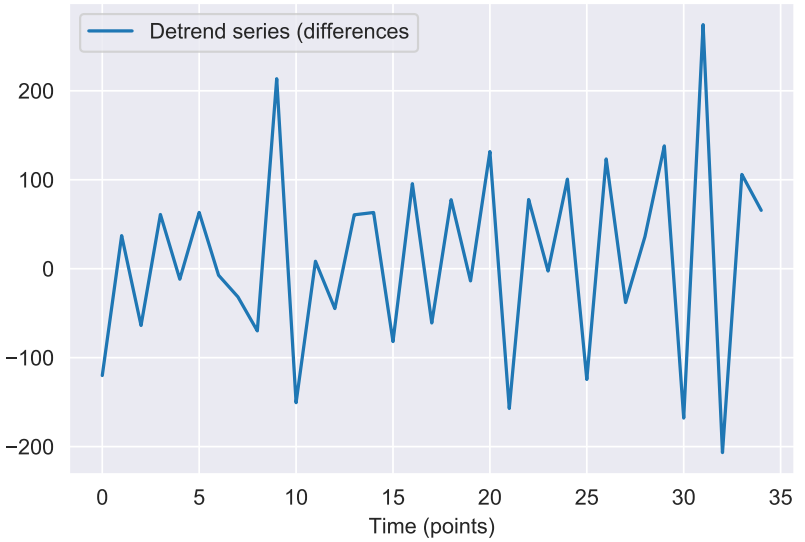


✔ No obvious trends in new sequence ⇒ have stationary process.

Regression vs Differencing



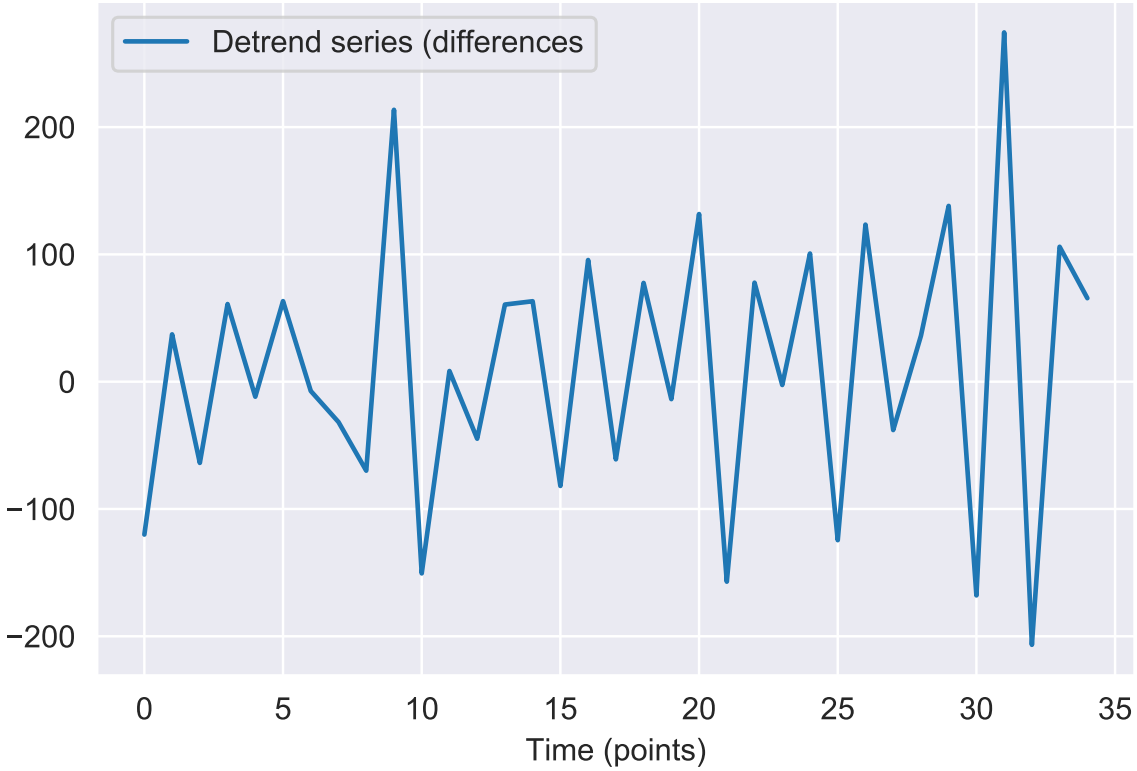


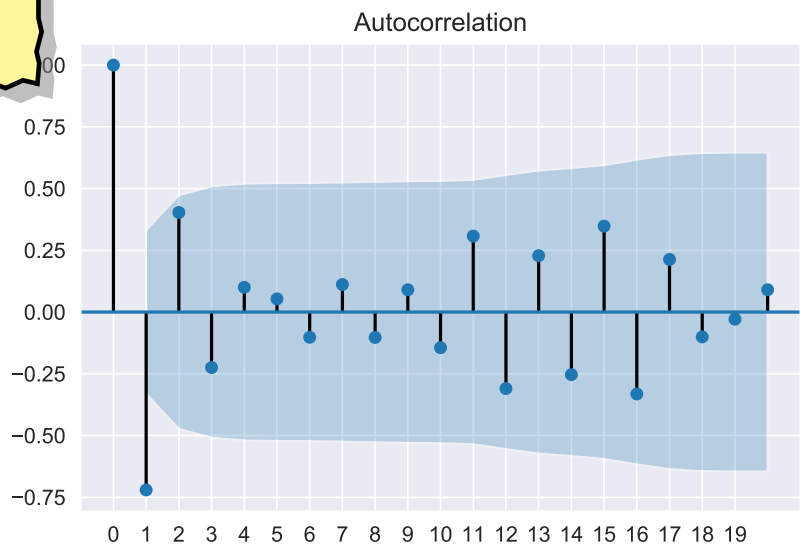


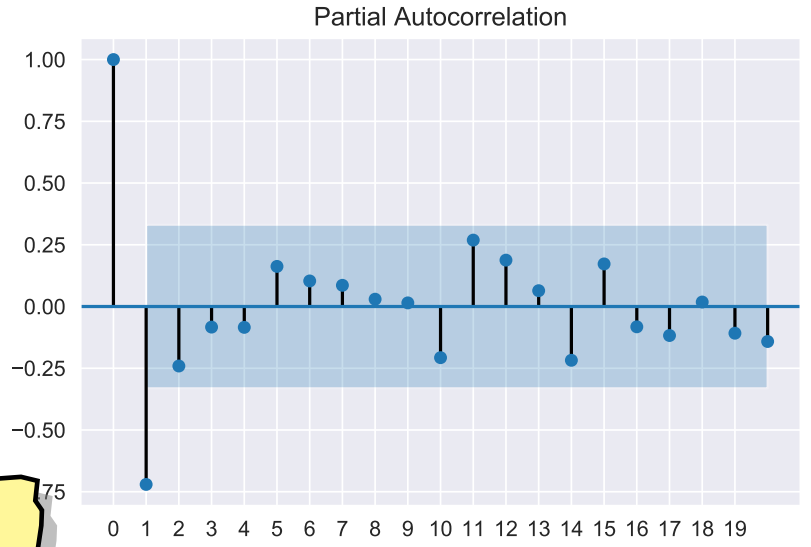
- Resulting de-trended series are not the same (they even don't have the same length!)
- The need for quadratic trend is obvious from the residuals, this is harder to see in the differences (but it is there).

Shampoo — ACF and PACF

Both ACF and PACF show rapid drop in lags so expect mixture of AR(p) and MA(p).







Expect both p and q to be small,
Do a parameter search. . .

Parameter Sweep of ARMA(p,q) Model

Checking for model ARMA(p,q) with parameters $0 \leq p, q < 4$ we get

Shampoo.ipynb In[27]:

```
p = q = range(0, 4)

opt_aic = np.finfo("float").max
opt_pq = None

for pq in itertools.product(p,
    try:
        model = ARMA(dy, order
        print("%s -> %s" % (pq
        if model.aic<opt_aic:
            opt_aic = model.ai
            opt_pq = pq
    except:
        continue

print("Optimal model: ARMA%s "
```

```
(0, 0) -> 430.87311187552797
(0, 1) -> 409.0126622411526
(1, 0) -> 406.0222895923501 5
(1, 1) -> 402.4666524179953
(1, 2) -> 401.5247943798928
(1, 3) -> 403.52417287508956
(2, 0) -> 403.6275641166603
(2, 1) -> 404.28621423318384
(3, 0) -> 404.6924591816878
(3, 1) -> 406.1395443929549
Optimal model: ARMA(1, 2) with AIC=401.5
```

So lets fit a ARMA(1,2) model and see its performance ...

Shampoo — Quality of ARMA(1,2) Model I

Shampoo.ipynb In[28]:

```
model = ARMA(dy, order=(1,2)).fit(disp=0)
print(model.summary())
```

ARMA Model Results

Dep. Variable:	y	No. Observations:	35
Model:	ARMA(1, 2)	Log Likelihood	−195.762
Method:	css-mle	S.D. of innovations	59.410
Date:	Sun, 24 Mar 2019	AIC	401.525
Time:	15:44:37	BIC	409.302
Sample:	0	HQIC	404.209

	coef	std err	z	P>|z|	[0.025	0.975]
const	10.2822	6.239	1.648	0.109	−1.945	22.510
ar.L1.y	0.1019	0.207	0.492	0.626	−0.304	0.508
ma.L1.y	−1.4369	0.477	−3.013	0.005	−2.372	−0.502
ma.L2.y	1.0000	0.653	1.532	0.136	−0.279	2.279

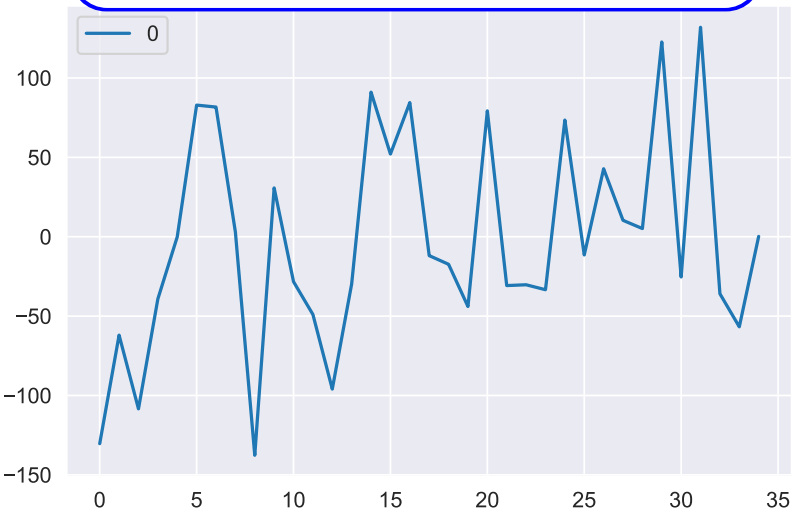
- Model is better (AIC=401) than that with linear regression (AIC=409), but not as good as the quadratic regression model (AIC=385).
- ✘ Highest order coefficients in AR and MA are not significant.

Shampoo — Quality of ARMA(1,2) Model II

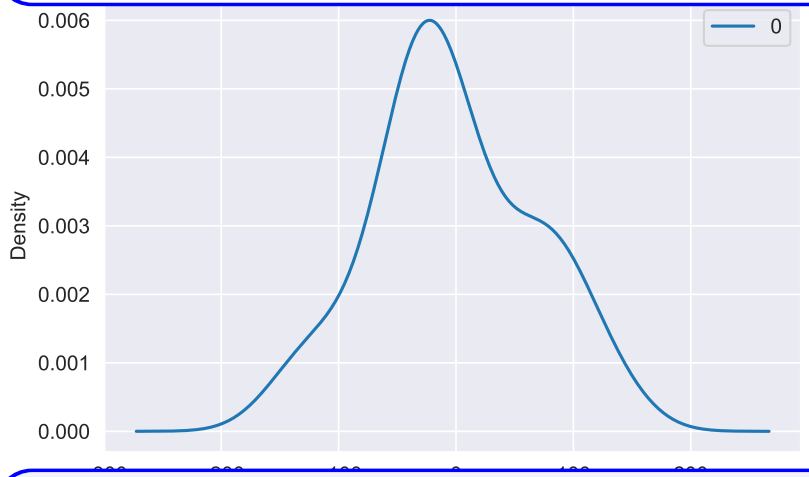
Shampoo.ipynb In[28]:

```
model = ARMA(dy, order=(1, 2)).fit(disp=0)
```

```
residuals.plot()
```




```
residuals.plot(kind='kde')
```



- ✘ upwards trend in residual plot.
- ✘ kernel density estimate plot is not symmetric.
- ✘ The 5-number summary shows same story as kde plot.

⇒ We should have paid more attention and dealt with trend in first differences by apply differences again.

```
print(residuals.describe())
```

count	36.000000
mean	−4.173472
std	64.343005
min	−155.086536
25%	−47.359822
50%	−8.557897
75%	27.487184
max	149.022740

A Model with built in Differencing — ARIMA

An ARIMA model can perform the differencing step as part of the model. So for this dataset we just applied ARMA(1,2) to dy. This is exactly the same as applying

- ARIMA(1,0,2) to dy.
- ARIMA(1,1,2) to y.

We will cover ARIMA in more detail later but for now

ARIMA(p,d,q)
The ARIMA(p,d,q) is the ARMA(p,q) model applied to the d^{th} difference of the original dataset.

- Getting difference is easy, so why yet another model?
  We can search for optimal difference as part of model optimisation.
- De-tending and model fitting can be a single step.

ARMA(1,2) on dy vs ARIMA(1,0,2) on dy vs ARIMA(1,1,2) on y

```
model = ARIMA(dy, order=(1,0,2)).fit(disp=0)
```

Dep. Variable:	y	No. Observations:	35
Model:	ARMA(1, 2)	Log Likelihood	−195.762
Method:	css-mle	S.D. of innovations	59.410
Date:	Sun, 24 Mar 2019	AIC	401.525
Time:	15:44:38	BIC	409.302
Sample:	0	HQIC	404.209

	coef	std err	z	P>|z|	[0.025	0.975]
const	10.2822	6.239	1.648	0.109	−1.945	22.510
ar.L1.y	0.1019	0.207	0.492	0.626	−0.304	0.508
ma.L1.y	−1.4369	0.477	−3.013	0.005	−2.372	−0.502
ma.L2.y	1.0000	0.653	1.532	0.136	−0.279	2.279

```
model = ARIMA(y, order=(1,1,2)).fit(disp=0)
```

Dep. Variable:	D.y	No. Observations:	35
Model:	ARIMA(1, 1, 2)	Log Likelihood	−195.762
Method:	css-mle	S.D. of innovations	59.410
Date:	Sun, 24 Mar 2019	AIC	401.525
Time:	15:44:38	BIC	409.302
Sample:	1	HQIC	404.209

	coef	std err	z	P>|z|	[0.025	0.975]
const	10.2822	6.239	1.648	0.109	−1.945	22.510
ar.L1.D.y	0.1019	0.207	0.492	0.626	−0.304	0.508
ma.L1.D.y	−1.4369	0.477	−3.013	0.005	−2.372	−0.502
ma.L2.D.y	1.0000	0.653	1.532	0.136	−0.279	2.279

Parameter Sweep of ARIMA(p,d,q) Model

Checking for model ARIMA(p,d,q) with parameters $0 \leq p, d, q < 4$ we get

Shampoo.ipynb In[32]:

```
p = d = q = range(0, 4)

opt_aic = np.finfo("float").max
opt_pdq = None

for pdq in itertools.product(p,d,q):
    try:
        model = ARIMA(y, order=pdq).fit(disp=0)
        #print("%s -> %s" % (pdq,model.aic))
        if model.aic<opt_aic:
            opt_aic = model.aic
            opt_pdq = pdq
    except:
        continue

print("Optimal model: ARIMA%s with AIC=%.1f" % (opt_pdq,opt_aic))
```

```
Optimal model: ARIMA(2, 2, 3) with AIC=389.7
```

So ARIMA spotted that we needed to get second difference. Good.

Shampoo — Quality of ARIMA(2,2,3) Model I

Shampoo.ipynb In[33]:

```
model = ARIMA(y, order=(2,2,3)).fit(disp=0)
print(model.summary())
```

ARIMA Model Results

Dep. Variable:	D2.y	No. Observations:	34
Model:	ARIMA(2, 2, 3)	Log Likelihood	−187.842
Method:	css-mle	S.D. of innovations	49.755
Date:	Sun, 24 Mar 2019	AIC	389.684
Time:	15:44:41	BIC	400.368
Sample:	2	HQIC	393.327

	coef	std err	z	P>|z|	[0.025	0.975]
const	0.9216	0.101	9.091	0.000	0.723	1.120
ar.L1.D2.y	−1.4817	0.138	−10.704	0.000	−1.753	−1.210
ar.L2.D2.y	−0.5607	0.139	−4.030	0.000	−0.833	−0.288
ma.L1.D2.y	−0.9857	0.147	−6.695	0.000	−1.274	−0.697
ma.L2.D2.y	−0.9872	0.155	−6.358	0.000	−1.291	−0.683
ma.L3.D2.y	0.9986	0.147	6.794	0.000	0.710	1.287

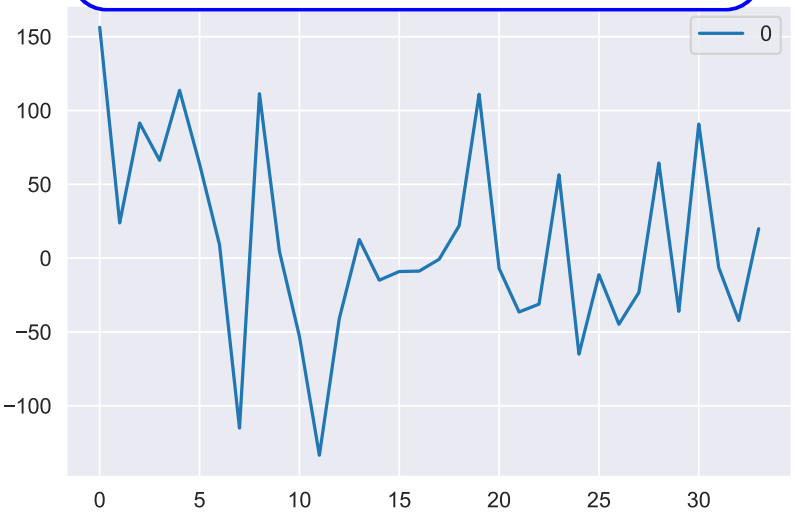
- ✔ Model is nearly as good (AIC=389.7) as best model found to date (quadratic regression + ARMA(3,2) with AIC=385.6).
- ✔ All coefficients are significant.

Shampoo — Quality of ARIMA(2,2,3) Model II

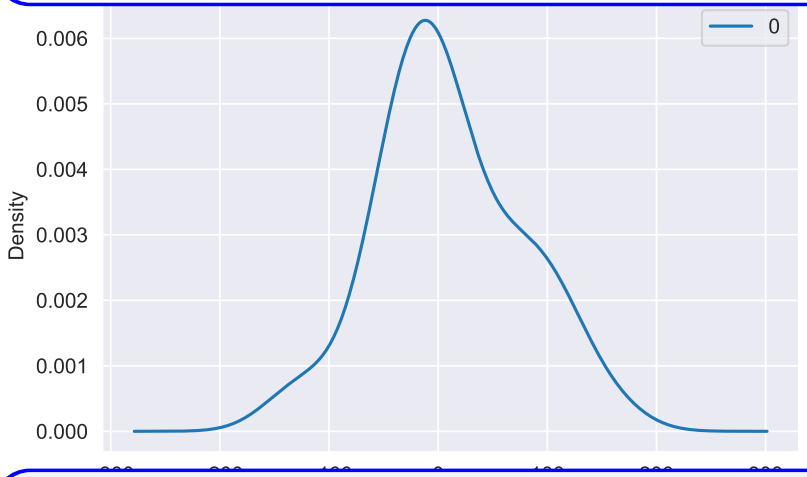
Shampoo.ipynb In[33]:

```
model = ARIMA(y, order=(2, 2, 3)).fit(disp=0)
```

```
residuals.plot()
```




```
residuals.plot(kind='kde')
```




```
print(residuals.describe())
```

count	34.000000
mean	9.954162
std	65.408579
min	−133.519276
25%	−34.811266
50%	−3.560706
75%	61.996983
max	156.378357

- ✔ no obvious pattern in residuals.
- ✔ kernel density estimate plot and 5-number summary is not symmetric but not as bad as in quadratic regression model.

⇒ Probably will go with this model. It is easier and fewer ‘bad smells’.

ARIMA — Integrated ARMA

An ARIMA(p, d, q) is the application of the ARMA(p,q) model to the d^{th} difference of the dataset

$$X_t^{(d)} = \underbrace{\alpha_1 X_{t-1}^{(d)} + \cdots + \alpha_p X_{t-p}^{(d)}}_{\text{AR(p)}} + \underbrace{\beta_1 \epsilon_{t-1} + \cdots + \beta_q \epsilon_{t-q}}_{\text{MA(q)}} + \epsilon_t$$

The model has three parameters/components:

- *p* Autoregressive component, $AR(p)$
  Incorporates the effect of recent past values into the model.
  (Intuition: it is likely to be warm today if it has been warm the past 3 days.)
- *d* Integrated component, $I(d)$.
  The amount of differencing (repeated application of differencing) to apply to the time series.
  (Intuition: it is likely to be same temperature tomorrow if the difference in temperature in the last three days has been very small.)
- *d* Moving Average component, $MA(q)$
  Set the error of our model as a linear combination of the error values observed at previous time points in the past.

SARIMA — Seasonal Integrated ARMA.

The ARIMA model can deal with trends (Shampoo dataset) but not seasonal effects. To handle seasonal effects (CO2 dataset) we extend the ARIMA into a Seasonal Autoregressive Integrated Moving Average (SARIMA).

SARIMA

The seasonal ARIMA model has parameters

SARIMA(p,d,q)(P,D,Q)s

where

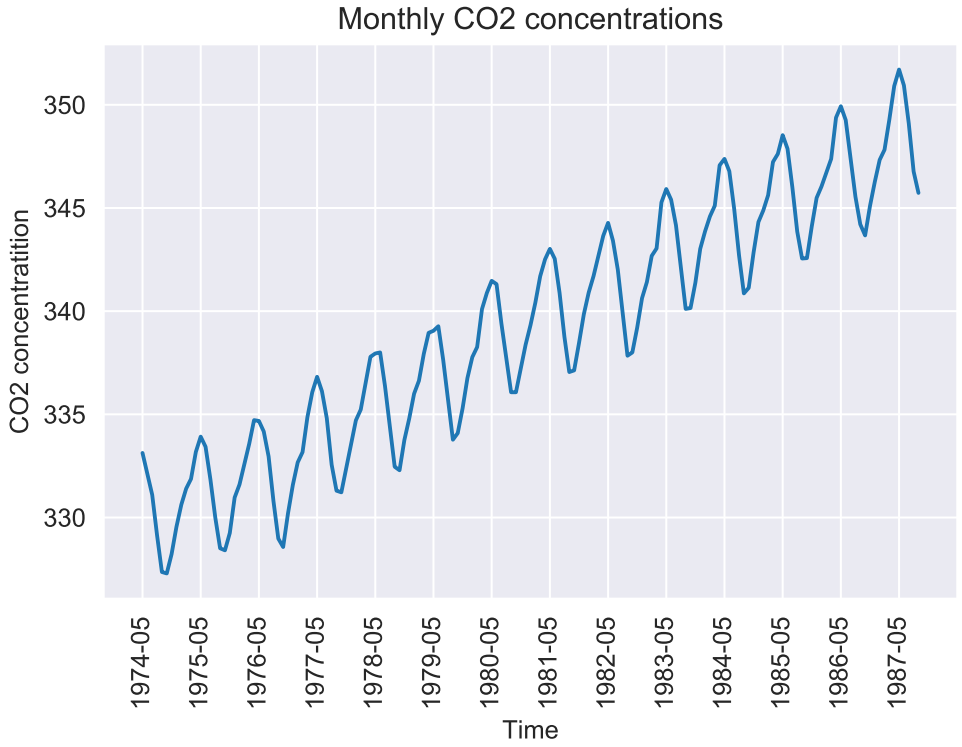
- (p, d, q) are the AR, I and MA parameters as usual.
- (P, D, Q) are the AR, I and MA parameters for the seasonal component with period s.

- Seasonal period, s, is obtained for ACF and PACF, other parameters can be found by grid search.

CO2 Dataset

Monthly observations over 13 years:

- Strong trend and seasonal components.
- Series is non-stationary so cannot apply stationary model (ARMA) directly.



CO2.ipynb In[3]:

```
df = pd.read_excel("src/Monthly_CO2_Concentrations.xlsx")
display(df.shape)
df.isna().sum()
df = df.dropna()
```

Grid Search on SARIMA

- After a sloooow grid search, optimal model[†] is

$$\text{SARIMA}(0,1,1) \times (2,1,3,12)$$

- Fitting model is via

CO2.ipynb In[13]:

```
model = sm.tsa.statespace.SARIMAX(df.CO2.values,
    order=(0,1,1), seasonal_order=(2,1,3,12),
    enforce_stationarity=False,
    enforce_invertibility=False).fit()
```

- Model statistics are generated using

CO2.ipynb In[14]:

```
print(model.summary().tables[0])
print(model.summary().tables[1])
```

[†]see notebook

CO2 — Quality of SARMA(0,1,1)x(2,1,3,12 Model I

Statespace Model Results

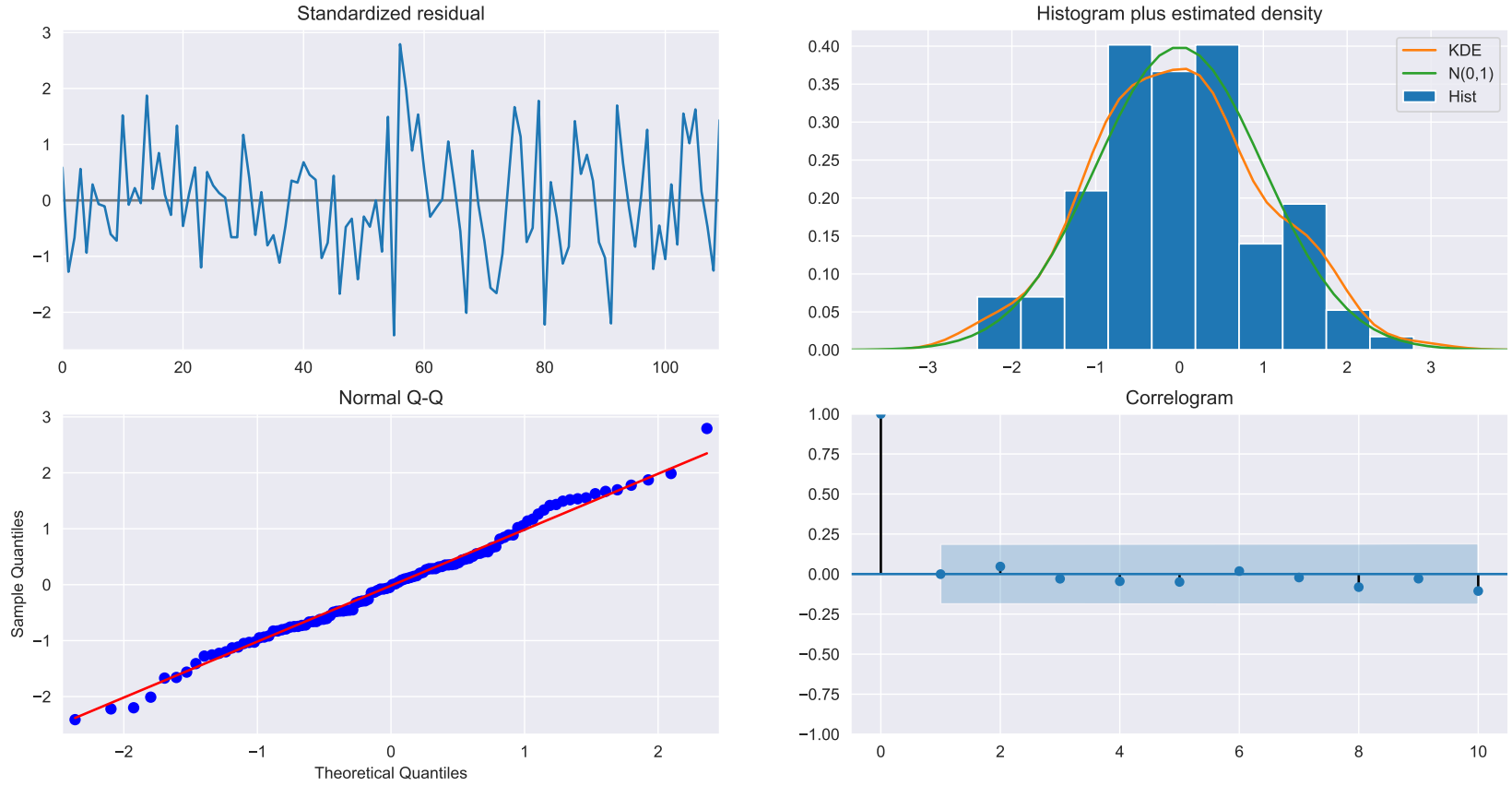
Dep. Variable:	y	No. Observations:	161
Model:	SARIMAX(0, 1, 1)x(2, 1, 3, 12)	Log Likelihood	−23.952
Date:	Mon, 25 Mar 2019	AIC	61.903
Time:	06:54:22	BIC	83.473
Sample:	0	HQIC	70.662
	− 161		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
ma.L1	−0.5625	0.082	−6.827	0.000	−0.724	−0.401
ar.S.L12	−0.7741	0.340	−2.279	0.023	−1.440	−0.108
ar.S.L24	−0.4546	0.245	−1.854	0.064	−0.935	0.026
ma.S.L12	1.3098	0.936	1.399	0.162	−0.525	3.145
ma.S.L24	0.8580	1.297	0.661	0.508	−1.685	3.401
ma.S.L36	−1.9180	1.338	−1.433	0.152	−4.541	0.705
sigma2	0.0143	0.020	0.714	0.475	−0.025	0.054

CO2.ipynb In[15]:

```
results.plot_diagnostics(figsize=(16, 8))
plt.savefig("pic/CO2_SARIMA.pdf", bbox_inches="tight")
plt.show()
```

CO2 — Quality of SARMA(0,1,1) x (2,1,3,12 Model II



- ✔ The KDE follows closely with the standard normal, N(0,1) curve.
- ✔ The qq-plot shows that the ordered distribution of residuals follows the linear trend.
- ✔ No obvious pattern in the residuals
- ✔ The autocorrelation (i.e. correlogram) shows the residuals have low correlation with lagged versions of itself.

All good indications that the residuals are normally distributed and independent.