W4995 Applied Machine Learning

Time Series and Forecasting

04/24/17

Andreas Müller

What's different?

- Not iid

Equally spaced & not equally spaced

$$(x_{t_1}, y_{t_1}), ...(x_{t_m}, y_{t_m}) \quad t_1 < t_2 < ... < t_m$$

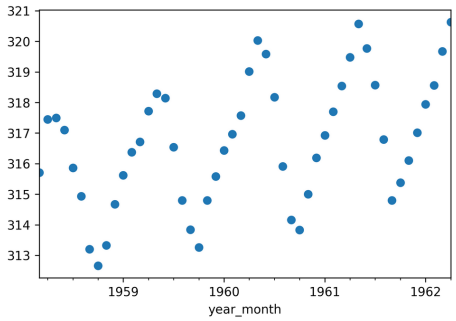
Equally spaced:

$$t_{i+1} - t_i = \text{const}$$ 1 second, 1 day, 1 year

Unevenly spaced:

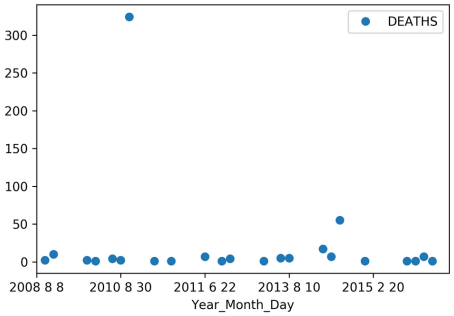
$$t_{i+1} - t_i \quad \text{varies}$$ Event time series (earthquake, trade, ...)

Evenly spaced



CO2 Measurements

Unevenly spaced

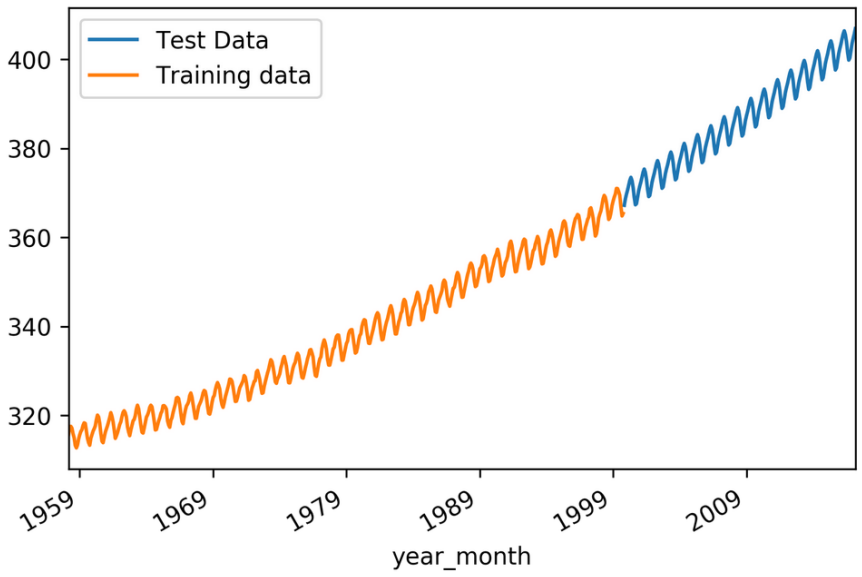


Volcanic eruption deaths

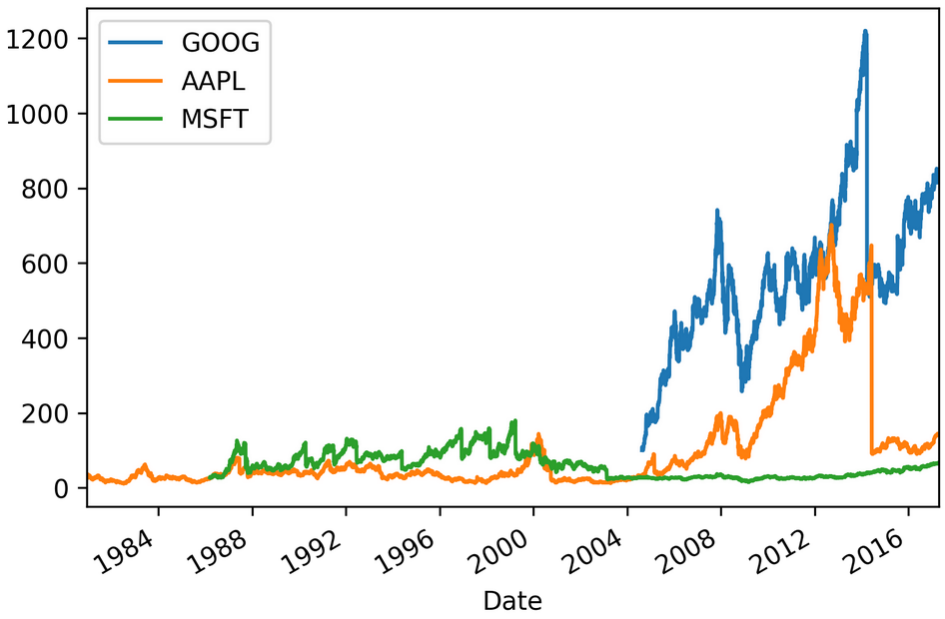
Today: Only evenly spaced.
For uneven: sometimes aggregates over fixed periods make sense.

Tasks

1d Forecasting



Nd forecasting



Feature-based forecasting (explanatory variables)

	Appliances	lights	T1	RH_1	T2	RH_2	T3	RH_3	T4	RH_4	...	T9	RH_9	T_out	Press
date															
2016-01-11 17:00:00	60	30	19.89	47.596667	19.2	44.790000	19.79	44.730000	19.000000	45.566667	...	17.033333	45.53	6.600000	733.5
2016-01-11 17:10:00	60	30	19.89	46.693333	19.2	44.722500	19.79	44.790000	19.000000	45.992500	...	17.066667	45.56	6.483333	733.6
2016-01-11 17:20:00	50	30	19.89	46.300000	19.2	44.626667	19.79	44.933333	18.926667	45.890000	...	17.000000	45.50	6.366667	733.7
2016-01-11 17:30:00	50	40	19.89	46.066667	19.2	44.590000	19.79	45.000000	18.890000	45.723333	...	17.000000	45.40	6.250000	733.8
2016-01-11 17:40:00	60	40	19.89	46.333333	19.2	44.530000	19.79	45.000000	18.890000	45.530000	...	17.000000	45.40	6.133333	733.9

Time Series Classification

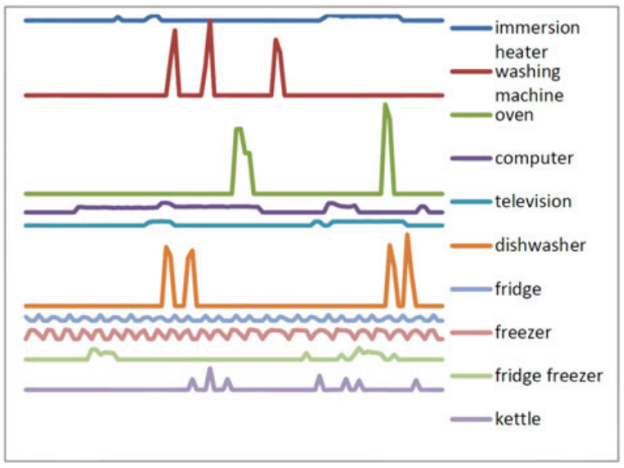


Fig. 1. Examples of daily profiles for the ten devices considered

From: Classification of Household Devices byElectricity Usage Profiles

Tasks

- 1d time-series forecasting
  - Number of customers (naive)
- Multivariate time-series – forecast all
  - Stock market
- Multivariate time-series – forecast one
  - Number of customers (with features)
- Time-series classification (1d or nd)
  - Activity from smart phone sensors

Data reading / munging with Pandas

Parse Dates

```
url = "ftp://aftp.cmdl.noaa.gov/products/trends/co2/co2_weekly_mlo.txt"
names = ["year", "month", "day", "year_decimal", "co2", "days", "1 yr ago", "10 yr ago", "since 1800"]
maunaloa = pd.read_csv(url, skiprows=49, header=None, delim_whitespace=True,
                       names=names, na_values=[-999.99])
```

```
maunaloa.head()
```

	year	month	day	year_decimal	co2	days	1 yr ago	10 yr ago	since 1800
0	1974	5	19	1974.3795	333.34	6	NaN	NaN	50.36
1	1974	5	26	1974.3986	332.95	6	NaN	NaN	50.06
2	1974	6	2	1974.4178	332.32	5	NaN	NaN	49.57
3	1974	6	9	1974.4370	332.18	7	NaN	NaN	49.63
4	1974	6	16	1974.4562	332.37	7	NaN	NaN	50.07

Parse dates

```
maunaloa = pd.read_csv(url, skiprows=49, header=None, delim_whitespace=True,
                       names=names, parse_dates=[[0, 1, 2]], na_values=[-999.99])
```

```
maunaloa.head()
```

	year_month_day	year_decimal	co2	days	1 yr ago	10 yr ago	since 1800
0	1974-05-19	1974.3795	333.34	6	NaN	NaN	50.36
1	1974-05-26	1974.3986	332.95	6	NaN	NaN	50.06
2	1974-06-02	1974.4178	332.32	5	NaN	NaN	49.57
3	1974-06-09	1974.4370	332.18	7	NaN	NaN	49.63
4	1974-06-16	1974.4562	332.37	7	NaN	NaN	50.07

Multiple columns combined to a single date!

```
maunaloa.year_month_day.head()
```

```
0   1974-05-19
1   1974-05-26
2   1974-06-02
3   1974-06-09
4   1974-06-16
Name: year month day, dtype: datetime64[ns]
```

Time Series Index

```
maunaloa = pd.read_csv(url, skiprows=49, header=None, delim_whitespace=True,
                       names=names, parse_dates=[[0, 1, 2]], na_values=[-999.99],
                       index_col="year_month_day")
```

```
maunaloa.head()
```

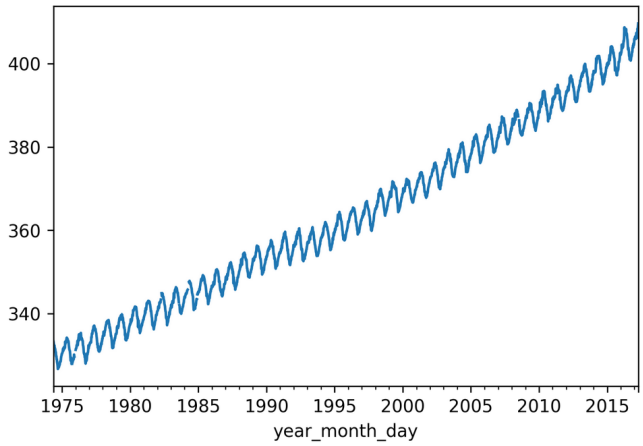
	year_decimal	co2	days	1 yr ago	10 yr ago	since 1800
year_month_day						
1974-05-19	1974.3795	333.34	6	NaN	NaN	50.36
1974-05-26	1974.3986	332.95	6	NaN	NaN	50.06
1974-06-02	1974.4178	332.32	5	NaN	NaN	49.57
1974-06-09	1974.4370	332.18	7	NaN	NaN	49.63
1974-06-16	1974.4562	332.37	7	NaN	NaN	50.07

Can also use set_index if we already read the data it.

```
maunaloa.co2.head()
```

```
year_month_day
1974-05-19    333.34
1974-05-26    332.95
1974-06-02    332.32
1974-06-09    332.18
1974-06-16    332.37
Name: co2, dtype: float64
```

```
maunaloa.co2.plot()
```



Backfill and Forward Fill

- Simple “imputation” method

```
maunaloa.co2.isnull().sum()
```

```
20
```

```
maunaloa.fillna(method="ffill", inplace=True)  # or bfill
maunaloa.co2.isnull().sum()
```

```
0
```

- If we resample later, we could also just drop

Resampling

```
# resampling is lazy
resampled_co2 = maunaloa.co2.resample("MS")
print(resampled_co2)
resampled_co2.mean().head()
```

```
DatetimeIndexResampler [freq=<MonthBegin>, axis=0, closed=left, label=left, convention=start, base=0]

year_month_day
1974-05-01    333.1450
1974-06-01    332.0280
1974-07-01    330.7125
1974-08-01    329.0725
1974-09-01    327.3240
Freq: MS, Name: co2, dtype: float64
```

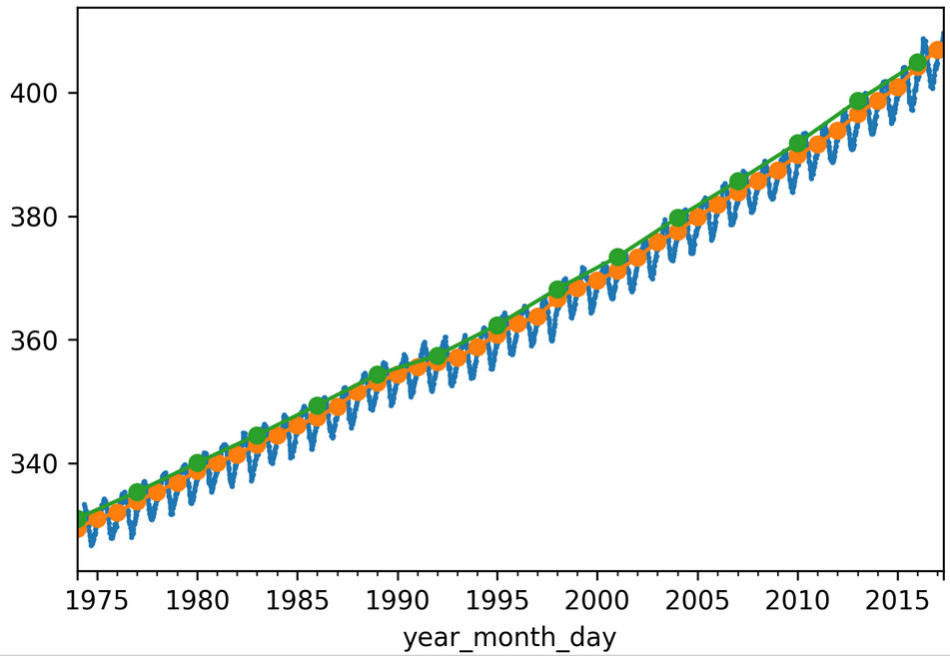
Alias	Description
B	business day frequency
C	custom business day frequency (experimental)
D	calendar day frequency
W	weekly frequency
M	month end frequency
SM	semi-month end frequency (15th and end of month)
BM	business month end frequency
CBM	custom business month end frequency
MS	month start frequency
SMS	semi-month start frequency (1st and 15th)
BMS	business month start frequency
CBMS	custom business month start frequency
Q	quarter end frequency

BQ	business quarter endfrequency
QS	quarter start frequency
BQS	business quarter start frequency
A	year end frequency
BA	business year end frequency
AS	year start frequency
BAS	business year start frequency
BH	business hour frequency
H	hourly frequency
T, min	minutely frequency
S	secondly frequency
L, ms	milliseconds
U, us	microseconds
N	nanoseconds

http://pandas.pydata.org/pandas-docs/stable/timeseries.html#timeseries-offset-aliases

```
maunaloa.co2.resample("W").mean().plot(marker="o", markersize=1)
maunaloa.co2.resample("AS").mean().plot(marker="o")
maunaloa.co2.resample("3AS").mean().plot(marker="o")
```

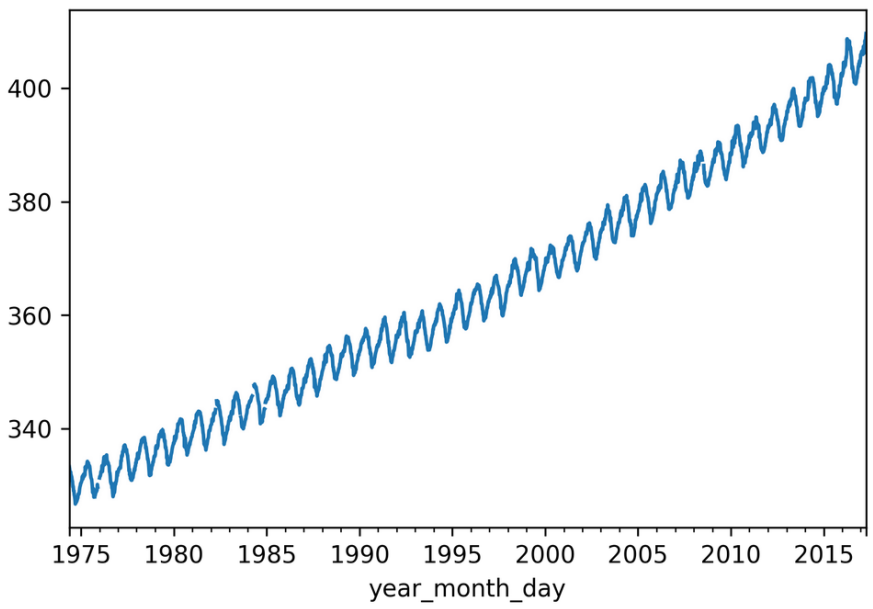
```
<matplotlib.axes. subplots.AxesSubplot at 0x7f240ea8c828>
```



Basic Analysis

Stationarity

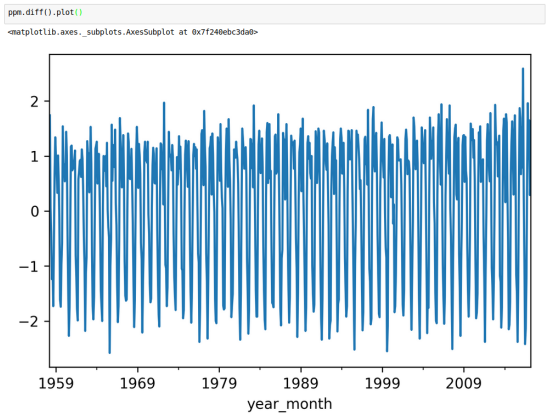
- Required for some classical statistics methods
- Mean independent of time
- Variance independent of time
- Covariance of two observations k steps apart independent of time



Mean changes → not stationary
“there is a trend”

De-trending

- Model trend
- Or: differencing (compute new series $x_{t+1} - x_t$)



Autocorrelation

$$R(s,t) = \frac{\mathrm{E}[(X_t - \mu_t)(X_s - \mu_s)]}{\sigma_t \sigma_s}$$

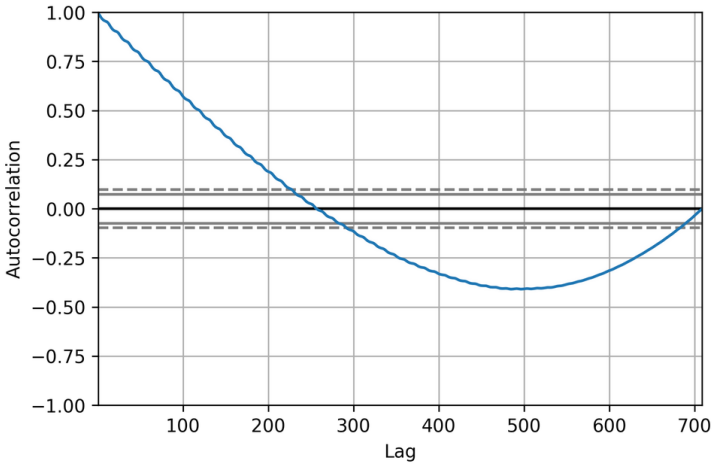
```
ppm.autocorr()
```

```
0.99892081520225573
```

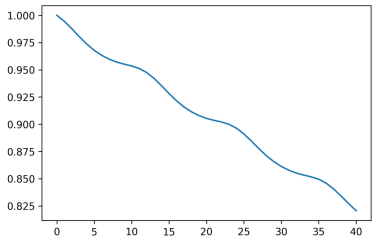
```
ppm.autocorr(lag=12)
```

```
0.99973333237787498
```

```
from pandas.tools.plotting import autocorrelation_plot
autocorrelation_plot(ppm)
```

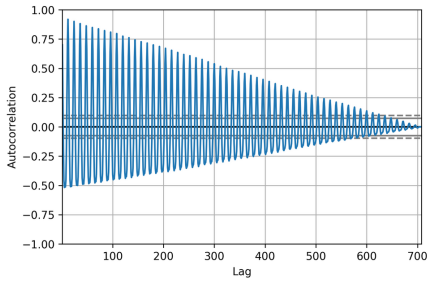


```
from statsmodels.tsa.stattools import acf
autocorrelation = acf(ppm)
plt.plot(autocorrelation)
```



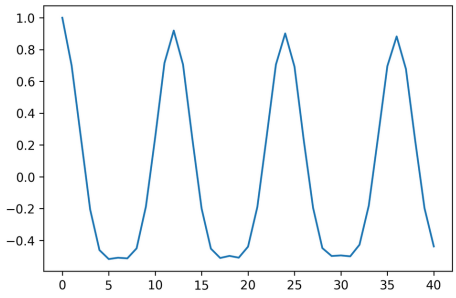

```
autocorrelation_plot(ppm.diff()[1:])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f2406473588>
```



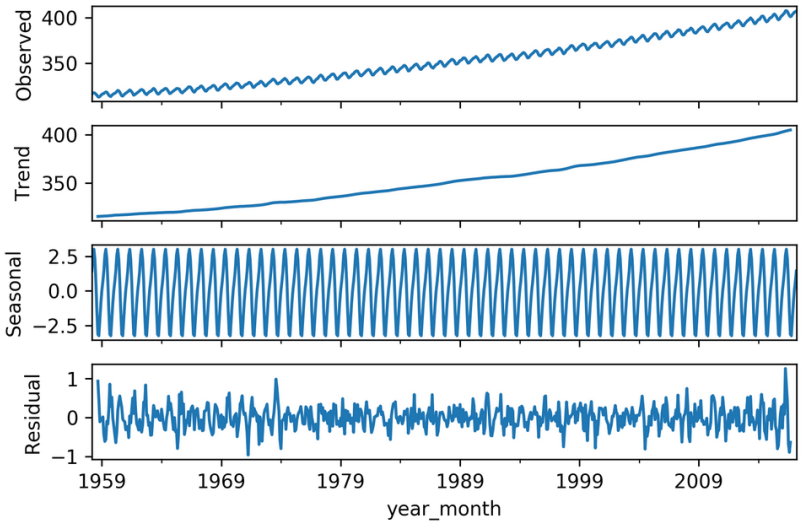

```
from statsmodels.tsa.stattools import acf
autocorrelation = acf(ppm.diff()[1:])
plt.plot(autocorrelation)
```

```
[<matplotlib.lines.Line2D at 0x7f240628fda0>]
```



Seasonal model for co2

```
from statsmodels.tsa.seasonal import seasonal_decompose
decomposition = seasonal_decompose(ppm, model='additive')
fig = decomposition.plot()
#fig.set_figheight(6)
fig.tight_layout()
```



Naive model using averages (so no parametric form)

Autoregressive (linear) model

Order k AR model (can include trend and offset):

$$x_{t+k} = c_0 x_t + c_1 x_{t+1} + ... + c_{k-1} x_t$$

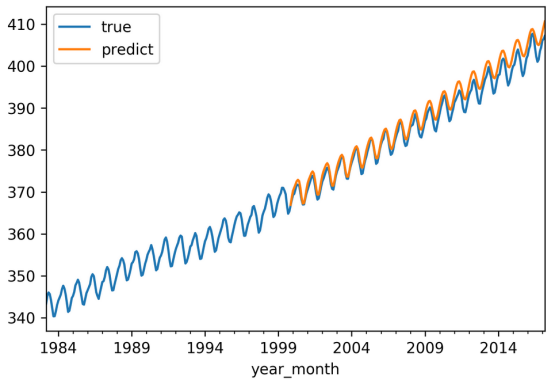
learn c_i (i.e. using least squares)

AR model with statsmodels

```
from statsmodels.tsa import ar_model
ar = ar_model.AR(ppm[:500])
res = ar.fit(maxlag=12)
res.params
```

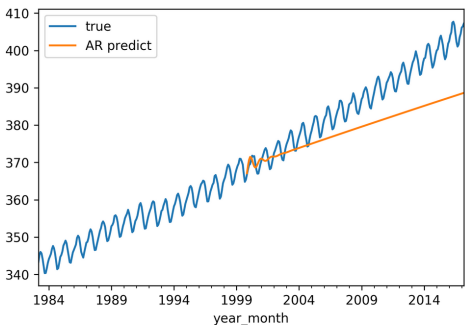
```
const              -2.018779
L1.interpolated     0.931758
L2.interpolated    -0.240629
L3.interpolated    -0.099048
L4.interpolated    -0.076488
L5.interpolated     0.164472
L6.interpolated    -0.000531
L7.interpolated    -0.023481
L8.interpolated    -0.125134
L9.interpolated     0.100288
L10.interpolated   -0.021266
L11.interpolated    0.414740
L12.interpolated   -0.017268
dtype: float64
```

```
res.predict(ppm.index[500], ppm.index[-1]).plot(label="predict")
ppm[300:].plot(label="true")
```

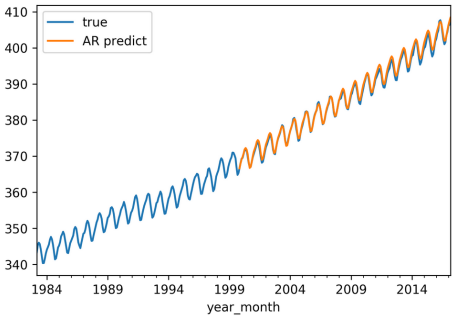


Impact of order

```
ar6 = ar_model.AR(ppm[:500])
res = ar6.fit(maxlag=6)
ar_pred = res.predict(ppm.index[500], ppm.index[-1])
ar_pred.plot(label="AR predict")
ppm[300:].plot(label="true")
plt.legend(loc="best")
```

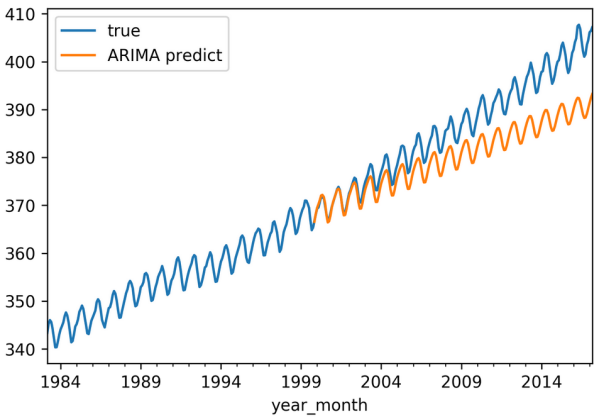



```
ar25 = ar_model.AR(ppm[:500])
res = ar25.fit(maxlag=25)
ar_pred = res.predict(ppm.index[500], ppm.index[-1])
ar_pred.plot(label="AR predict")
ppm[300:].plot(label="true")
plt.legend(loc="best")
```



ARIMA

```
from statsmodels import tsa

arima_model = tsa.arima_model.ARIMA(ppm[:500], order=(12, 1, 0))
res = arima_model.fit()
arima_pred = res.predict(ppm.index[500], ppm.index[-1], typ="levels")
ppm[300:].plot(label="true")
arima_pred.plot(label="ARIMA predict")
plt.legend(loc="best")
```



More complicated process model.

Internally uses differencing.

Also check out SARIMAX (state space model)

1d with sklearn

```
ppm.shape
```

```
(709,)
```

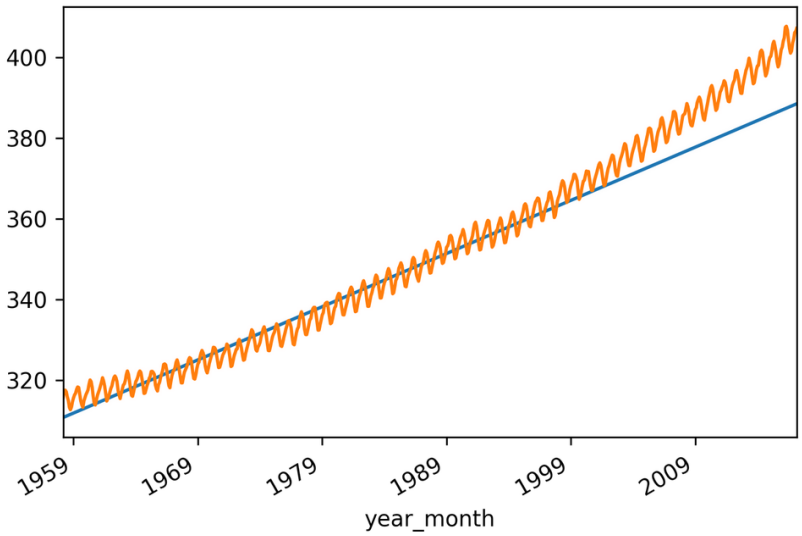
```
train = ppm[:500]
test = ppm[500:]
```

```
X = ppm.index.to_series().apply(lambda x: x.toordinal())
X = pd.DataFrame(X)

X_train, X_test = X.iloc[:500, :], X.iloc[500:, :]
X_train.shape
```

```
(500, 1)
```

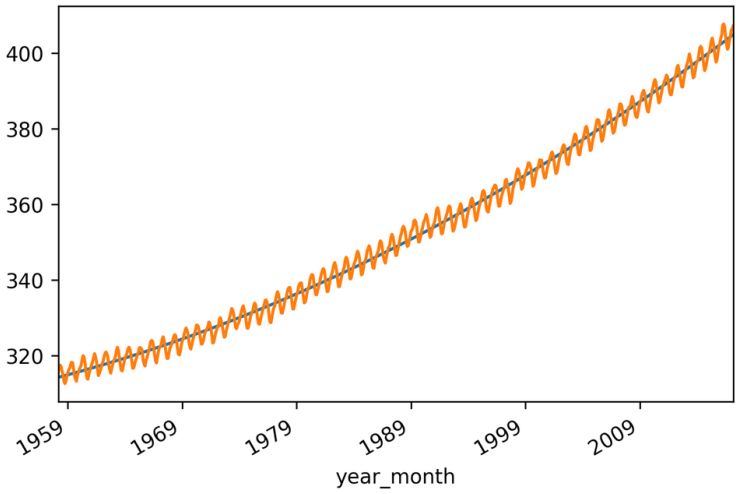
```
from sklearn.linear_model import LinearRegression
lr = LinearRegression().fit(X_train, train)
lr_pred = lr.predict(X_test)
plt.plot(ppm.index, lr.predict(X))
ppm.plot()
```

Linear model for trend



Quadratic Model

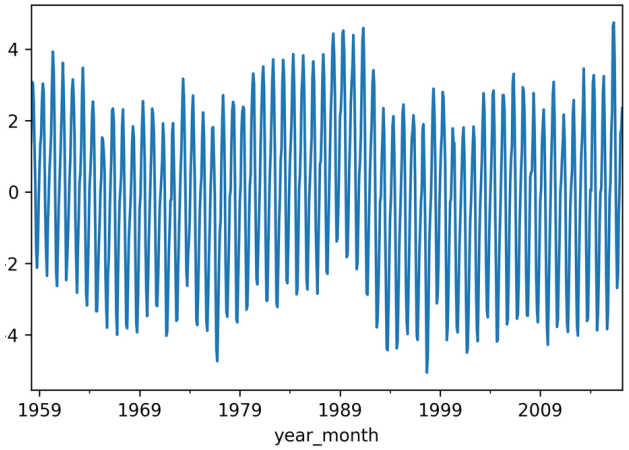
```
from sklearn.preprocessing import PolynomialFeatures
lr_poly = make_pipeline(PolynomialFeatures(include_bias=False), LinearRegression())
lr_poly.fit(X_train, train)

plt.plot(ppm.index, lr_poly.predict(X))
ppm.plot()
```



Detrending with trend model

```
y_res = ppm - lr_poly.predict(X)
```

```
y_res.plot()
```



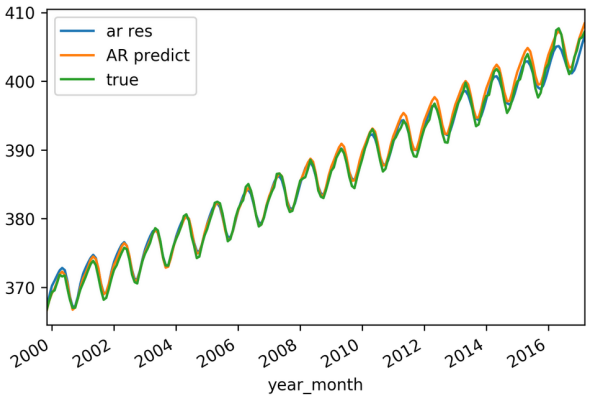
AR model on residuals

```
from statsmodels.tsa import ar_model
ar_model_res = ar_model.AR(y_res[:500])
res_res = ar_model_res.fit(maxlag=12)
```

```
ar_pred_res = res_res.predict(ppm.index[500], ppm.index[-1])
pred_ar_res = ar_pred_res + lr_poly.predict(X_test)
```

```
res_res.params
```

```
const              -0.008039
L1.interpolated     0.922508
L2.interpolated    -0.243015
L3.interpolated    -0.101173
L4.interpolated    -0.077538
L5.interpolated     0.158073
L6.interpolated    -0.006164
L7.interpolated    -0.023890
L8.interpolated    -0.126381
L9.interpolated     0.092664
L10.interpolated   -0.025640
L11.interpolated    0.414180
L12.interpolated   -0.018078
dtype: float64
```



Just Linear Regression

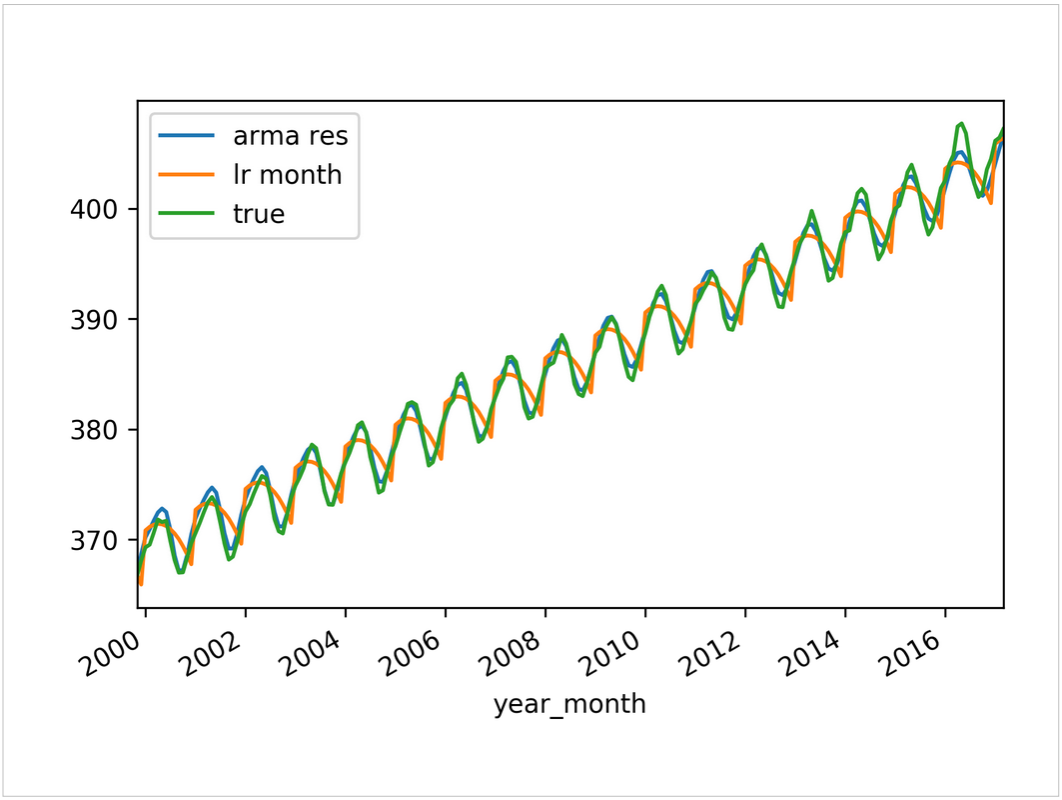
```
X_month = pd.concat([X, pd.DataFrame({'month': X.index.month}, index=X.index)], axis=1)
```

```
X_month.head()
```

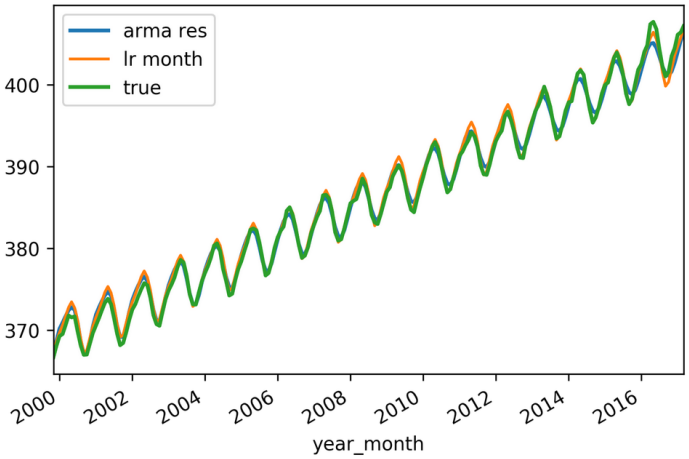
	year_month	month
year_month		
1958-03-01	714839	3
1958-04-01	714870	4
1958-05-01	714900	5
1958-06-01	714931	6
1958-07-01	714961	7

```
X_train_month = X_month[:500]
X_test_month = X_month[500:]
lr_poly_month = make_pipeline(PolynomialFeatures(include_bias=False), LinearRegression())
lr_poly_month.fit(X_train_month, train)
X_test_month.shape
```

```
(209, 2)
```


arma res
lr month
true
400
390
380
370
2000
2002
2004
2006
2008
2010
2012
2014
2016
year_month

```
from sklearn.preprocessing import OneHotEncoder
lr_poly_month_ohe = make_pipeline(OneHotEncoder(categorical_features=[1], sparse=False),
                                  PolynomialFeatures(include_bias=False), LinearRegression())
lr_poly_month_ohe.fit(X_train_month, train)
```




```
from sklearn.metrics import mean_squared_error
mean_squared_error(ppm[500:], lr_poly_month_ohe.predict(X_test_month))
```

```
0.50487819894708341
```

```
mean_squared_error(ppm[500:], pred_arma_res)
```

```
0.57735548378852652
```

General Autoregressors

- https://github.com/scikit-learn/scikit-learn/pull/6029
- Trees work well (as always)
- Can take explanatory variables

Multivariate Timeseries

energy data from https://archive.ics.uci.edu/ml/datasets/Appliances+energy+prediction

```
energy = pd.read_csv("energydata_complete.csv", parse_dates=['date'], index_col="date")
```

```
energy.head()
```

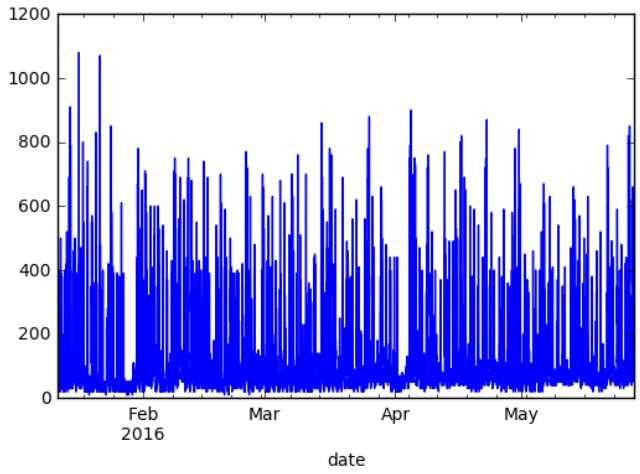
	Appliances	lights	T1	RH_1	T2	RH_2	T3	RH_3	T4	RH_4	...
date											
2016-01-11 17:00:00	60	30	19.89	47.596667	19.2	44.790000	19.79	44.730000	19.000000	45.566667	...
2016-01-11 17:10:00	60	30	19.89	46.693333	19.2	44.722500	19.79	44.790000	19.000000	45.992500	...
2016-01-11 17:20:00	50	30	19.89	46.300000	19.2	44.626667	19.79	44.933333	18.926667	45.890000	...
2016-01-11 17:30:00	50	40	19.89	46.066667	19.2	44.590000	19.79	45.000000	18.890000	45.723333	...
2016-01-11 17:40:00	60	40	19.89	46.333333	19.2	44.530000	19.79	45.000000	18.890000	45.530000	...

5 rows × 28 columns

Exploration

```
energy.Appliances.plot()
```

```
<matplotlib.axes. subplots.AxesSubplot at 0x7f3acfe626d8>
```



Weekly trends

```
week = energy.Appliances.groupby([energy.index.hour, energy.index.dayofweek]).mean()
```

```
week.head(30)
```

```
0  0      53.684211
   1      52.719298
   2      50.964912
   3      50.175439
   4      50.438596
   5      54.824561
   6      56.228070
   7      71.403509
   8      94.035088
   9      96.052632
   10    151.842105
   11    178.421053
   12    154.561404
   13    177.982456
   14    151.228070
   15    166.052632
   16    144.736842
   17    190.166667
   18    189.083333
   19    144.833333
   20    148.916667
   21    100.000000
   22     76.166667
   23     63.333333
1  0      66.083333
   1      56.250000
   2      49.083333
   3      47.083333
   4      48.833333
   5      59.833333
Name: Appliances, dtype: float64
```