https://t.me/SchoolofAiItalia

School of Ai Italia

Serie Temporali

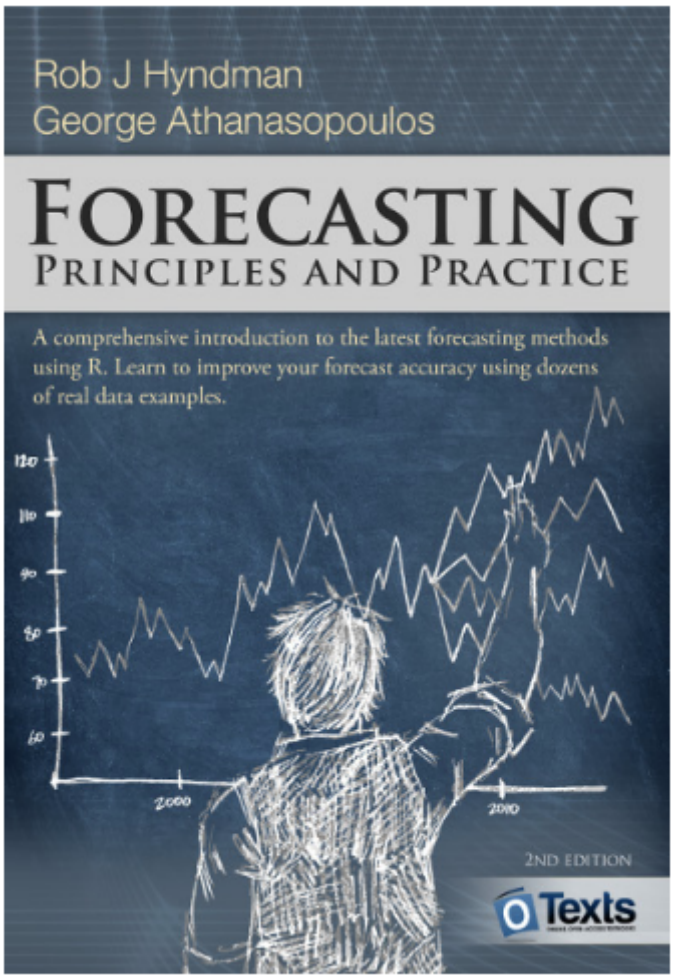
Parte 1...

Come vedremo le serie temporali sono al limite tra il puro Machine Learning e il Modelling in Machine Learning ed avendo una storia lunga quanto tutta la matematica moderna si portano dietro parecchia nomenclatura

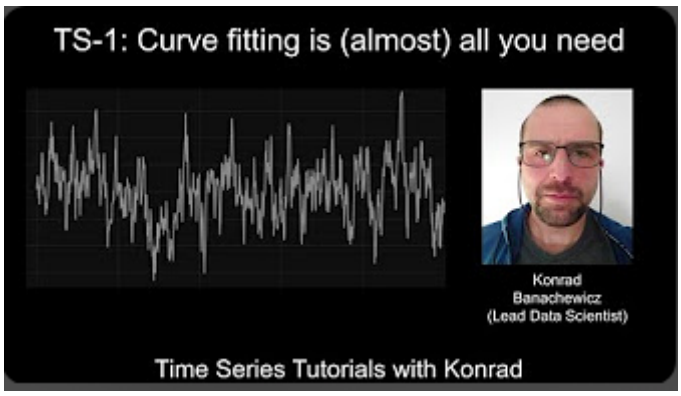
---

Bibliografia essenziale

- https://www.kaggle.com/learn/time-series
    - Linear Regression with Time Series
    - Trend
    - Seasonality
    - Time Series as Features
    - Hybrid Models
    - Forecasting With Machine Learning
- Testo consigliato Hyndman https://otexts.com/fpp2/



- Lezioni di Konrad

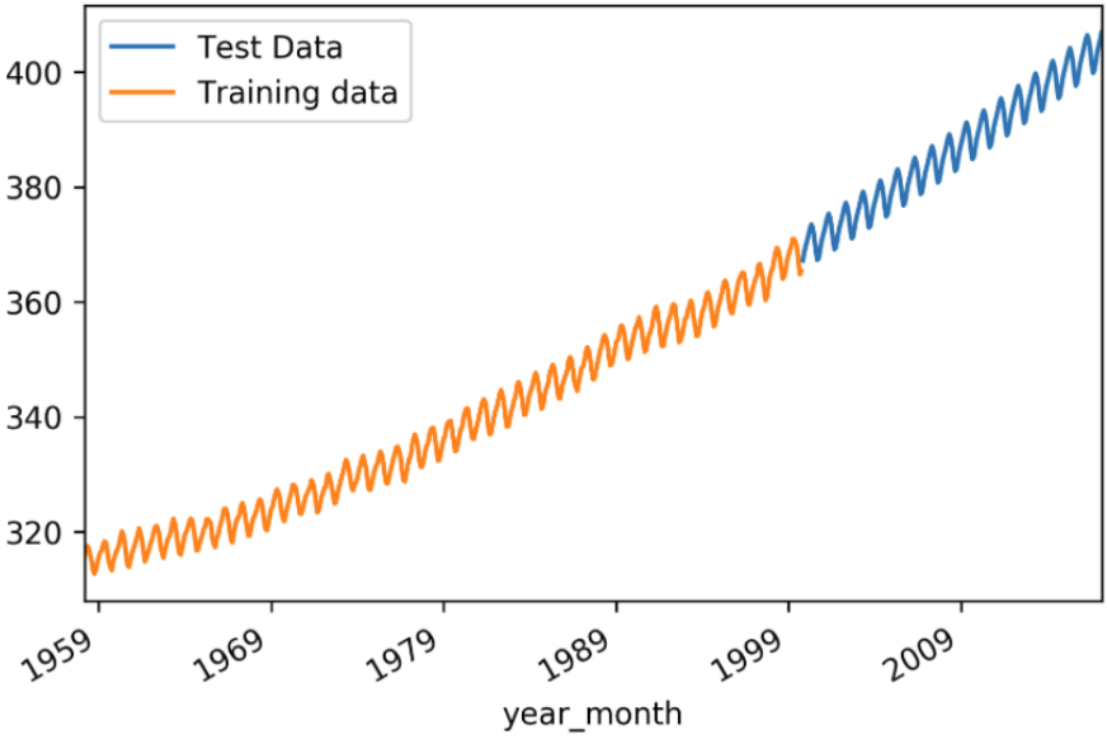


https://www.youtube.com/playlist?list=PL98nY_tJQXZmT9ZB59T0lsx0ZzzLrYdX4

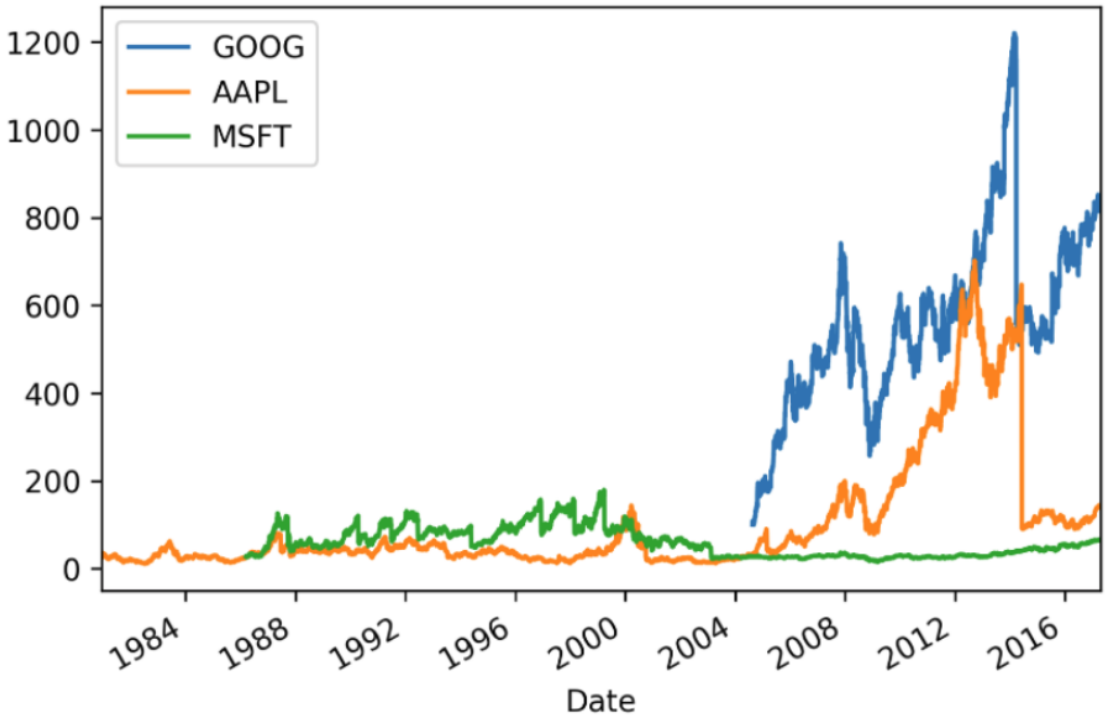
Materiale preso da https://amueller.github.io/ e altre fonti

Tasks

1d Forecasting



Nd Forecasting



Feature-based forecasting (expanatory variables)

date	Appliances	lights	T1	RH_1	T2	RH_2	T3	RH_3	T4	RH_4	...	T9	RH_9	T_out	Press_n
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

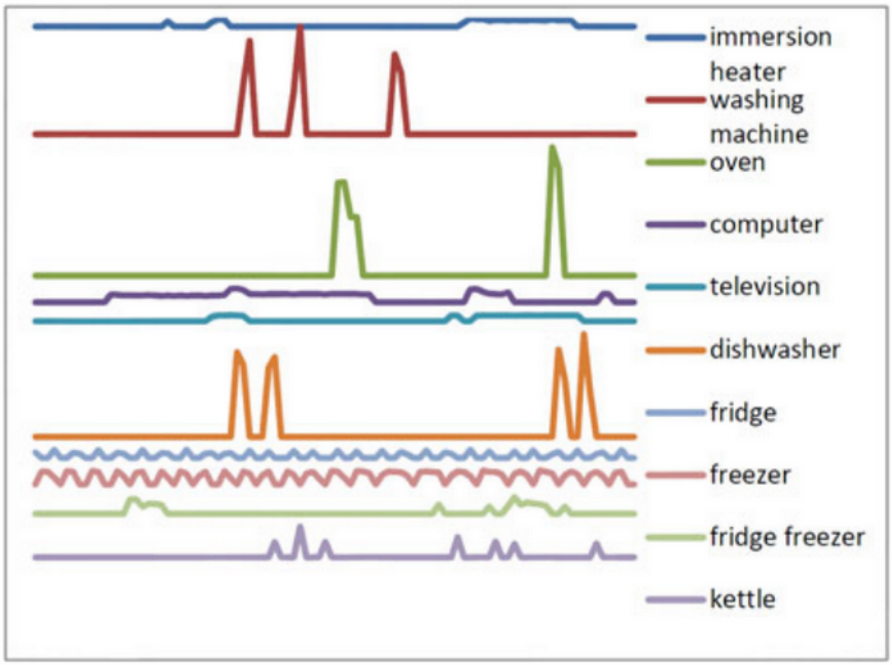


Fig. 1. Examples of daily profiles for the ten devices considered

##

Applicazioni

- Forecasting 1d
  - Forecasting nd (eg... Vector Models)
- Forecasting a uno step
  - Forecasting multistep

- Tabular forecasting
- Time Series Classification
- L’ Anomaly detection
- Times Series for finance
- Survival Analysis

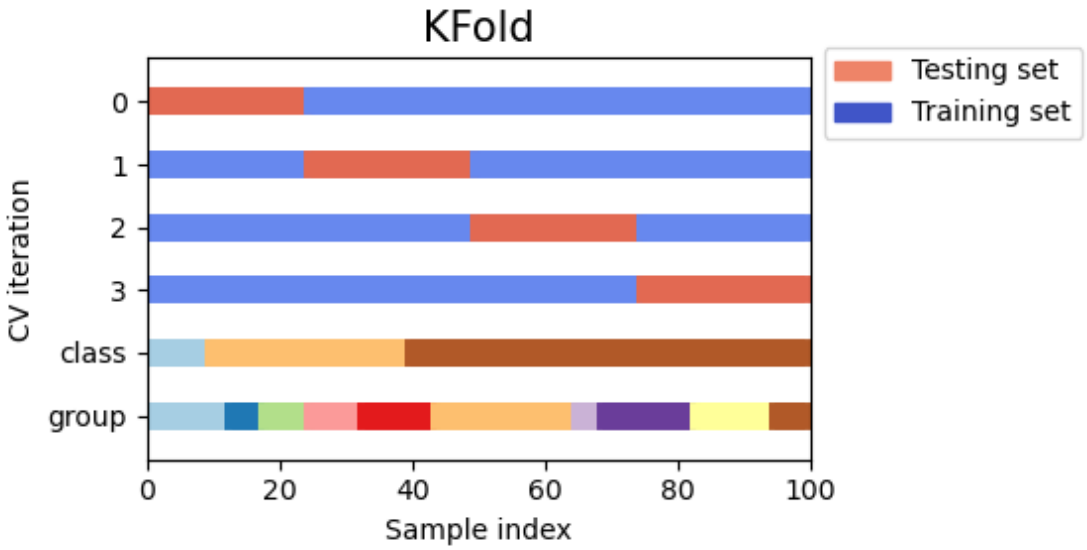
Cosa c’è di differente?

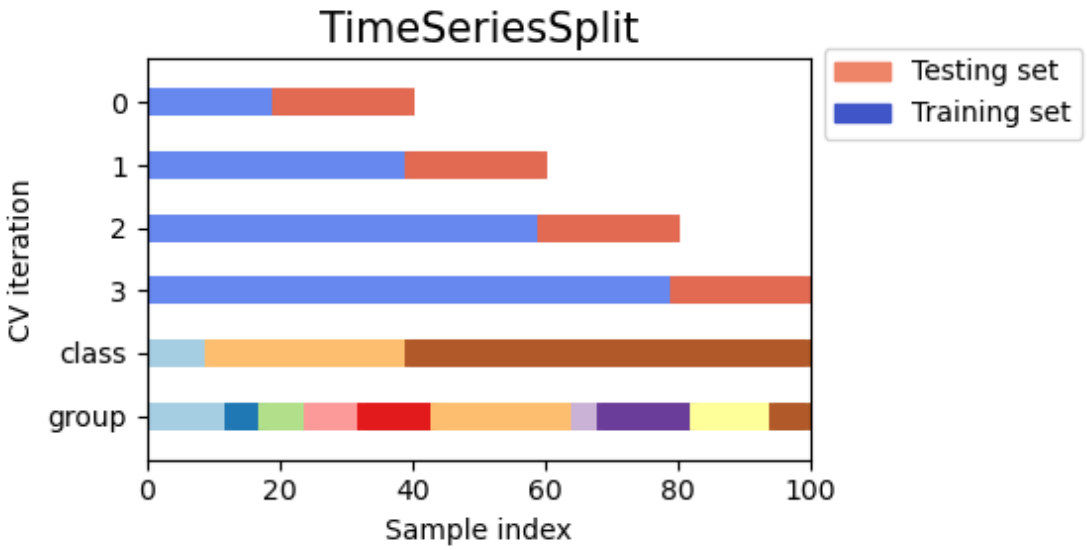
- La previsione non è IID
- La lunghezza delle features è variabile

NON IID -----> Cambia completamente lo schema di validazione

- perché dobbiamo preservare la causalità
- non possiamo fare in modo che ci sia inormation leaking tra trainset e validation set

  (la questione della scelta del validation set è in realtà ancora abbondantemente aperta ...)





Lunghezza features variabile ---> Memoria o State Space

Due approcci complementari:

- estrarre features su una **finestra** a dimensione variabile o opportunamente lunga
- usare una “memoria” o “latent state” che si aggiorni via via che è quello che fanno ad esempio le reti neurali ricorsive, il filtro di Kalman, la media mobile e così via ...

Preliminaries

Measurement frequencies

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

Equally spaced:

$$t_{i+1} - t_i = const$$

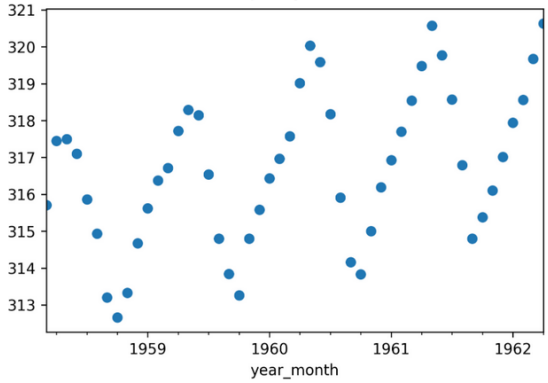
Unevenly spaced:

$$t_{i+1} - t_i \quad varies$$

aka "Point processes" in statistics

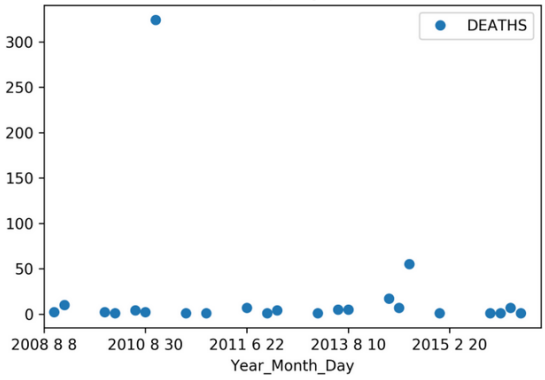
Examples

Evenly spaced



CO2 Measurements

Unevenly spaced



Volcanic Eruption Deaths

Data reading / munging with Pandas

MAUNA LOA

https://gml.noaa.gov/ccgg/trends/data.html

https://gml.noaa.gov/ccgg/trends/history.html

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

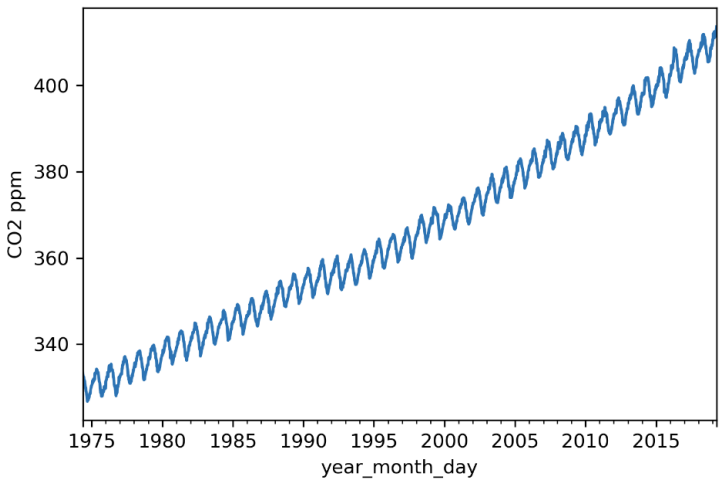
Time Series Index

```
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_month_day	year_decimal	co2	days	1 yr ago	10 yr ago	since 1800
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

https://amueller.github.io/COMS4995-s20/slides/aml-21-time-series/#16

Mauna Loa CO2 Dataset




```
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
```

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")
resampled_co2
```

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

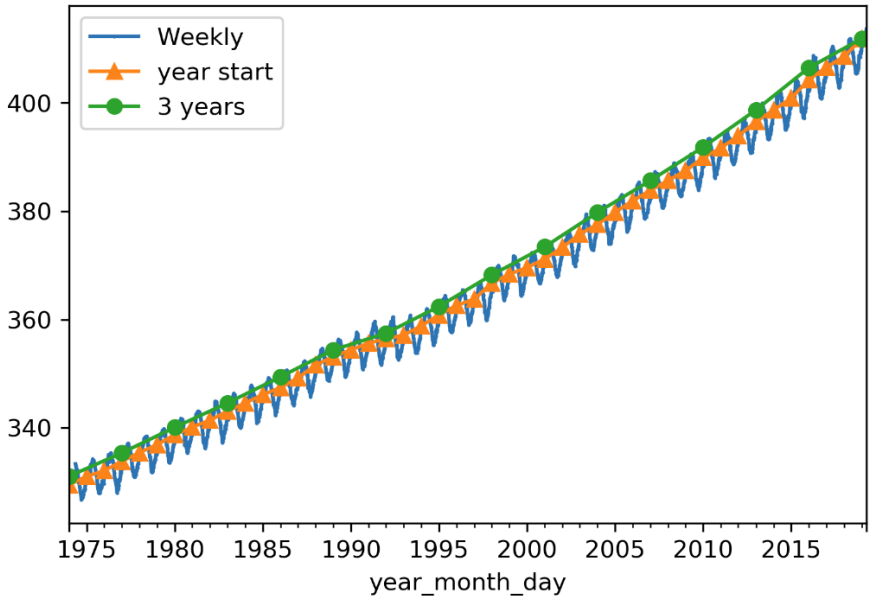
```
resampled_co2.mean().head()
```

```
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")
```

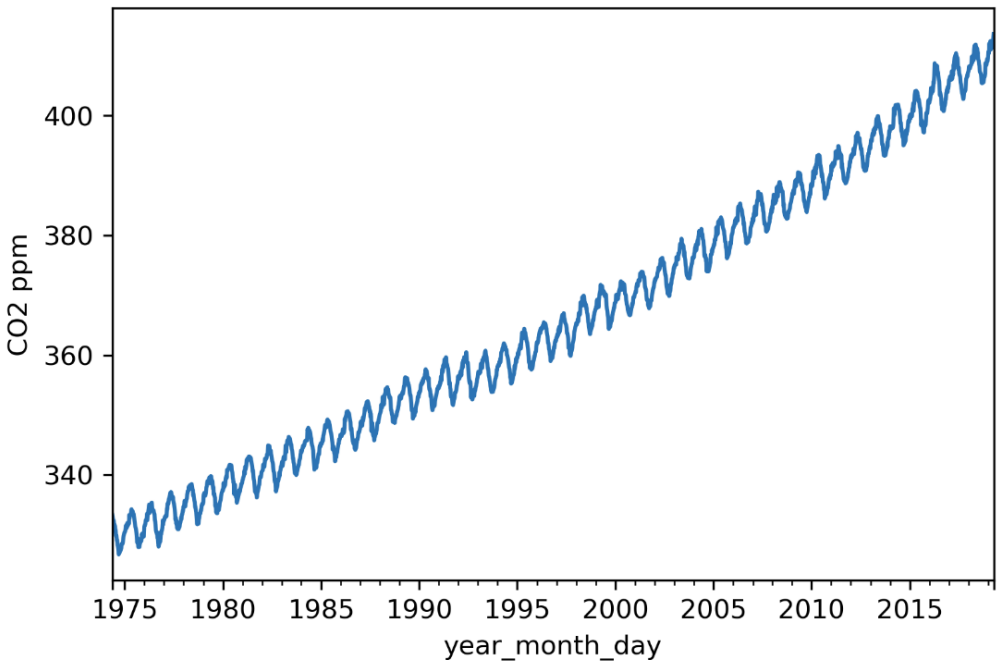


Terminologie

1d Forecasting Basics

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

Autocorrelation

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

```
ppm = maunaloa.co2
ppm.autocorr()
```

```
0.9997
```

```
ppm.autocorr(lag=26)
```

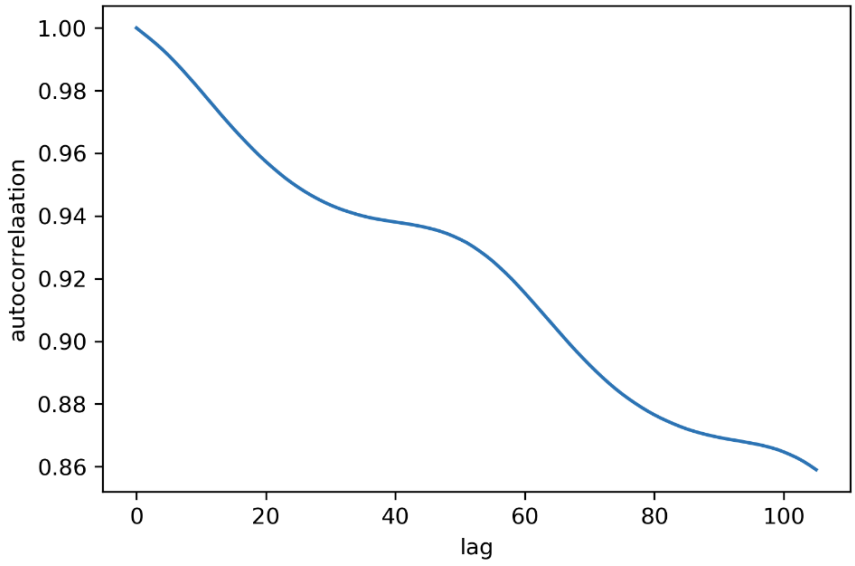
```
0.9826
```

```
ppm.autocorr(lag=52)
```

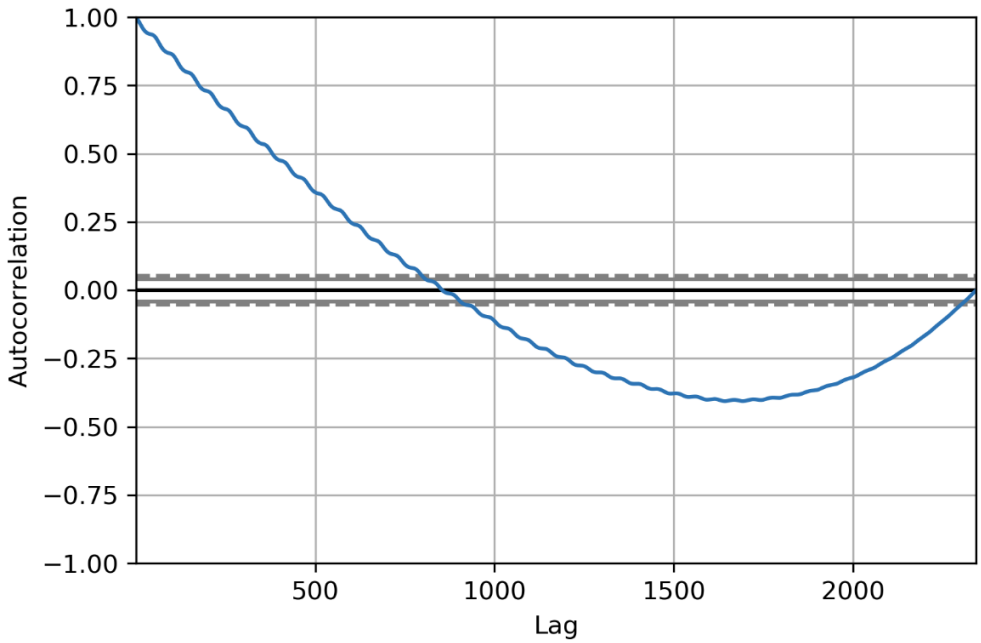
```
0.9994
```

Autocorrelation function

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



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



in qualche modo è una misura sia della capacità predittiva delle passate realizzazioni che della memoria del fenomeno che stiamo “modellando” o “prevedendo”

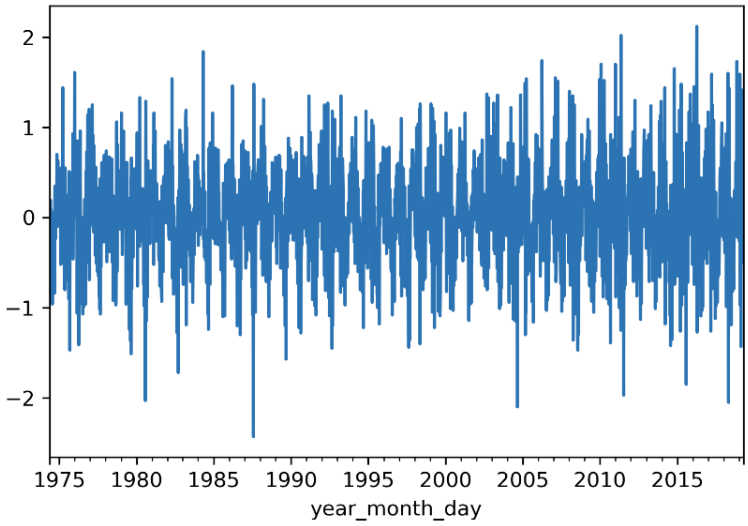
Spesso più utile dell’autocorrelazione è l’autocorrelazione parziale…

Nel modello ARIMA con la partial si determina p, con la correlation si determina q

De-trending

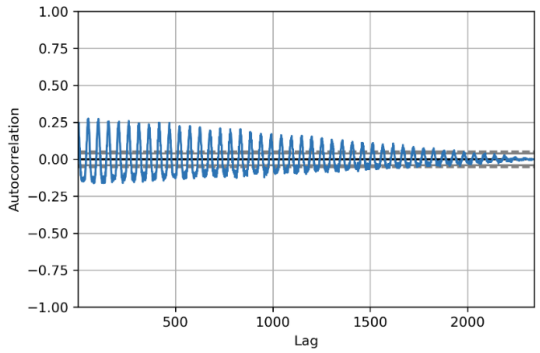
- Model trend
- Or differencing (compute new series).... $\hat{x}_t = x_t - x_{t-1}$

```
ppm.diff().plot()
```

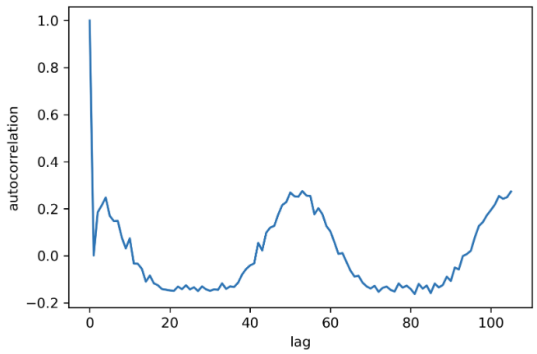


Autocorrelation of differenced series

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

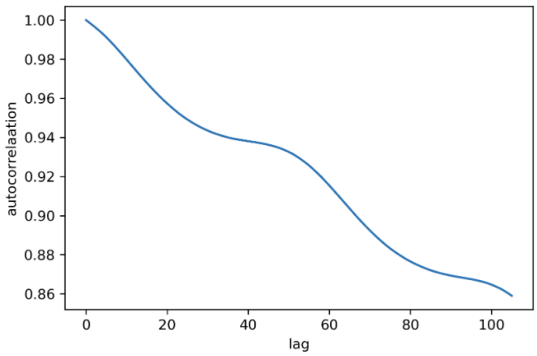



```
from statsmodels.tsa.stattools import acf
autocorrelation = acf(ppm.diff()[1:])
plt.plot(autocorrelation)
```

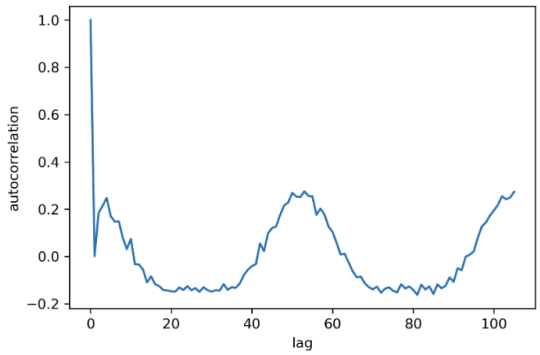


Original vs differenced

Autocorrelation on original data

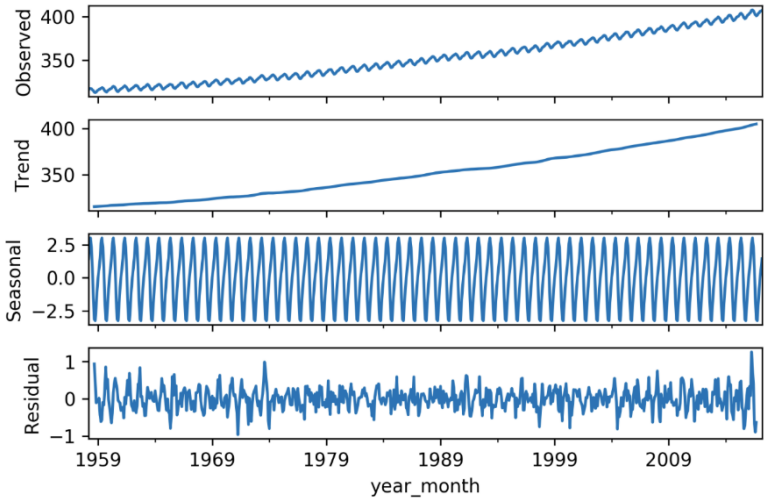


Autocorrelation on differenced data



Seasonal model for CO2

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



Modelli

- Modelli autoregressivi
- Modelli di curvefitting (dipendenti esplicitamente dal tempo)

Il rischio di modello che esplicitamente dipende dal tempo è l'OVERFIT!

Autoregressive (linear) model

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

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

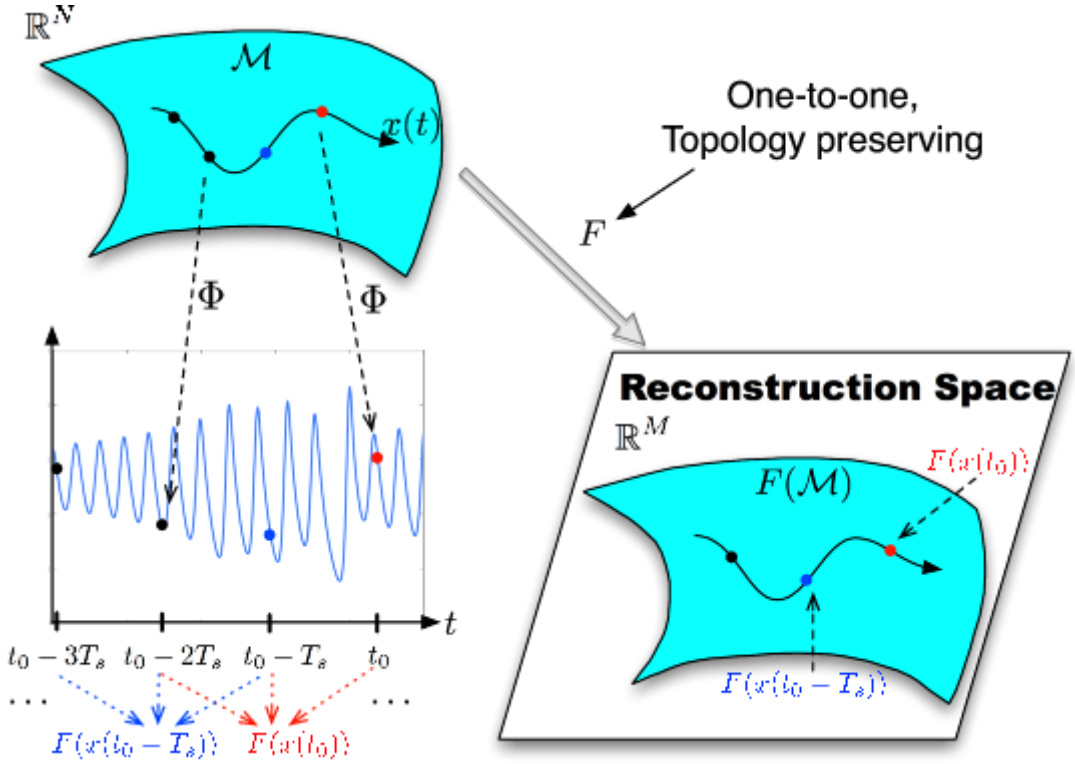
- learn c_i (i.e. using least squares)

https://www.kaggle.com/code/ryanholbrook/time-series-as-features

In fisica un'equazione differenziale i cui coefficienti non dipendono esplicitamente dal tempo si dice stazionaria. Dato che in fisica le leggi fisiche non cambiano nel tempo (non siamo su scale cosmologiche), le equazioni importanti in fisica sono tutte stazionarie e per loro vale il Teorema di Takens da una varietà di evoluzioni che non dipende dal tempo

Teorema di Takens

https://cnx.org/contents/k57_M8Tw@2/Takens-Embedding-Theorem



Insomma i modelli di Machine Learning possono agevolmente imparare il fenomeno ammesso che il fenomeno non cambi nel tempo attraverso i lag

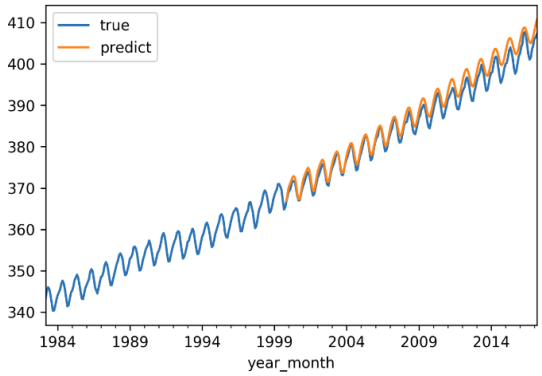
AR model with statsmodels

replaced by statsmodels.tsa.AutoReg now!

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

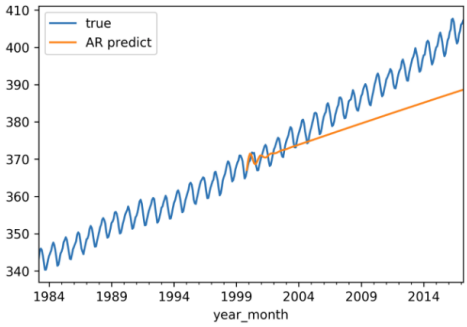
```
const              -2.018779
L1.interpolated     0.931758
L2.interpolated    -0.240629
L3.interpolated    -0.099048
L4.interpolated    -0.076488
L5.interpolated     0.164472
L6.interpolated    -0.000531
L7.interpolated    -0.023481
L8.interpolated    -0.125134
L9.interpolated     0.100288
L10.interpolated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

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

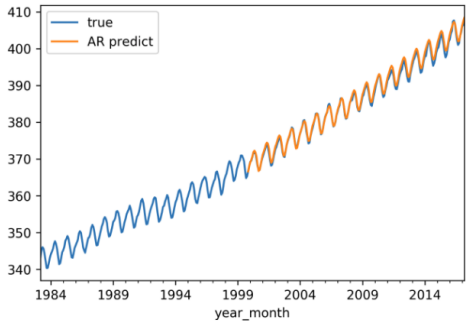


Impact of order

```
ar6 = ar_model.AR(ppm[:500])
res = ar6.fit(maxlag=6)
ar_pred = res.predict(ppm.index[500],
                      ppm.index[-1])
```




```
ar25 = ar_model.AR(ppm[:500])
res = ar25.fit(maxlag=25)
ar_pred = res.predict(ppm.index[500],
                      ppm.index[-1])
```



aumentare le lagged features porta all'overfit!

I modelli lineari con lag sono in grado di modellare agilmente anche fenomeni non lineari

I modelli lineari con features polinomiali sono invece addirittura universali cioè possono fittare ogni tipo di funzione

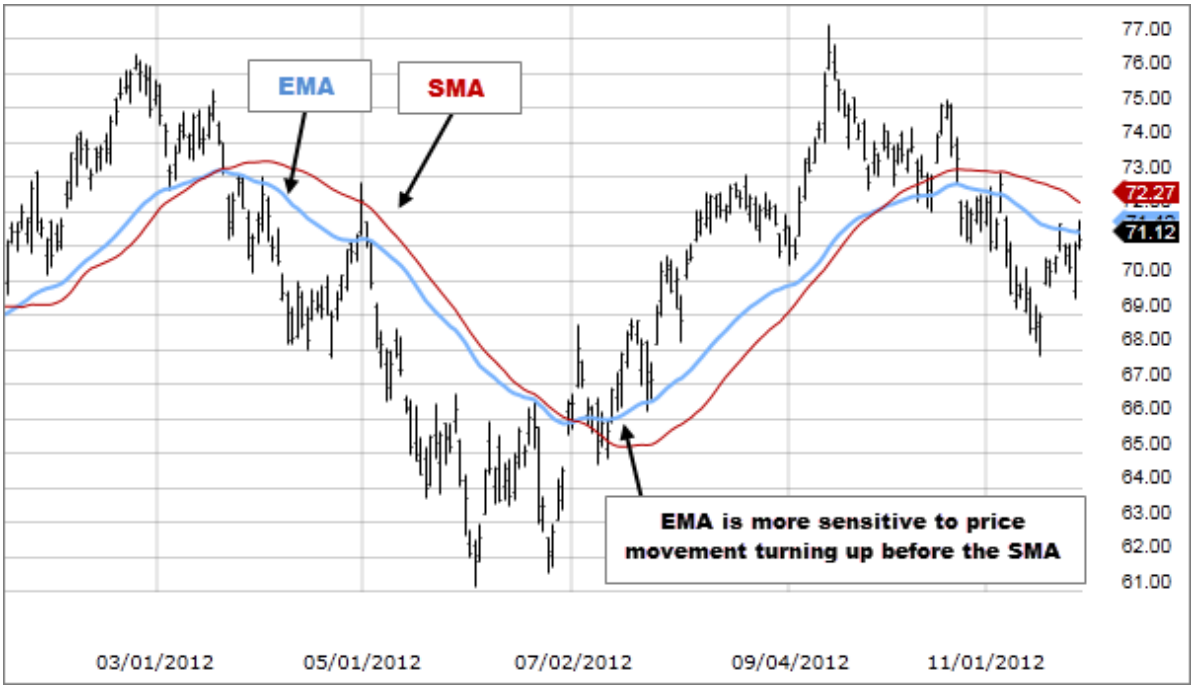
Exponential Smoothing (MA)

$$s_0 = x_0$$
$$s_t = \alpha x_t + (1-\alpha)s_{t-1}, \quad t > 0$$

where α is the *smoothing factor*, and $0 < \alpha < 1$.

$$s_t = \alpha x_t + (1-\alpha)s_{t-1} = s_{t-1} + \alpha(x_t - s_{t-1}).$$

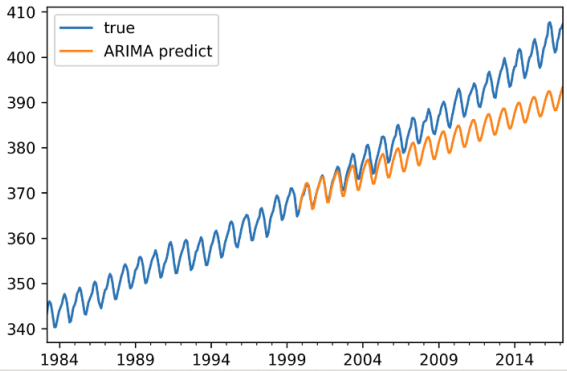
la formula di aggiornamento è analoga a quella del gradient descent



- Double exponential (Holt linear) P + Trend
- Triple exponential (Holt Winters) P + Trend + Seasonal

Autoregressive integrated moving average (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")
```



- Equation for an ARMA(p,q) model:

$$y_t = C + \phi_1 y_{t-1} + \phi_2 y_{t-2} + \ldots + \phi_p y_{t-p} + \varepsilon_t - \theta_1 \varepsilon_{t-1} - \theta_2 \varepsilon_{t-2} - \ldots - \theta_q \varepsilon_{t-q}$$

ricetta di Box e Jenkins per l'ARIMA(p,i,q)

- applicare una trasformazione tipo logaritmo alla funzione o la Box Cox
- differenziare la serie i volte per renderla più stazionaria
- prendere p termini laggati
- e q termini media esponenziale degli errori commessi dal modello nei q passi successivi
- usare una Linear Regression

Arima è una regressione lineare quindi ha bisogno di tutte le cure per i modelli lineari

- No outlier <----> differenziazione
- errore gaussiano <-----> trasformazione della serie per rendere gli errori gaussiani (esempio con un logaritmo)

Arima è un modello di machine learning di tipo stacking!!!

- come features regressori ha i lag che sono essi stessi crude stime del valore attuale
- come features ha gli errori ai passi precedenti che è l'output di altri modelli

Arima è time-independent quindi la previsione ritiene che il fenomeno sottostante non sia cambiato nel tempo (model shift)

- le prime estensioni create sono ad esempio doppio arima con un altro modello che decideva come passare da un ARIMA all' altro

L'Hyndman sostiene che uno dei modelli migliori che si possano realizzare è un ARIMAX con features di Fourier

- X sta per esogeno cioè esterna al modello
- Il modello ARIMAX di Hyndman è un'ARIMA che aggiunge nella regressione lineare funzioni di Fourier cioè seni e coseni del tempo a diverse frequenze per tener conto della variabilità nel tempo del modello

Arima usa delle crude differenze per semplificare la serie temporale

- si può far meglio con fractional derivatives
- decomposizioni o algoritmi di riduzione dimensionale vari

Per esempio posso usare l'ICA per decomporre nlag in n componenti indipendenti usare Arima (o exp mean o … o…) e poi riassemblare le componenti

---

Modelli di fit nel tempo

1d with sklearn

```
ppm.shape
```

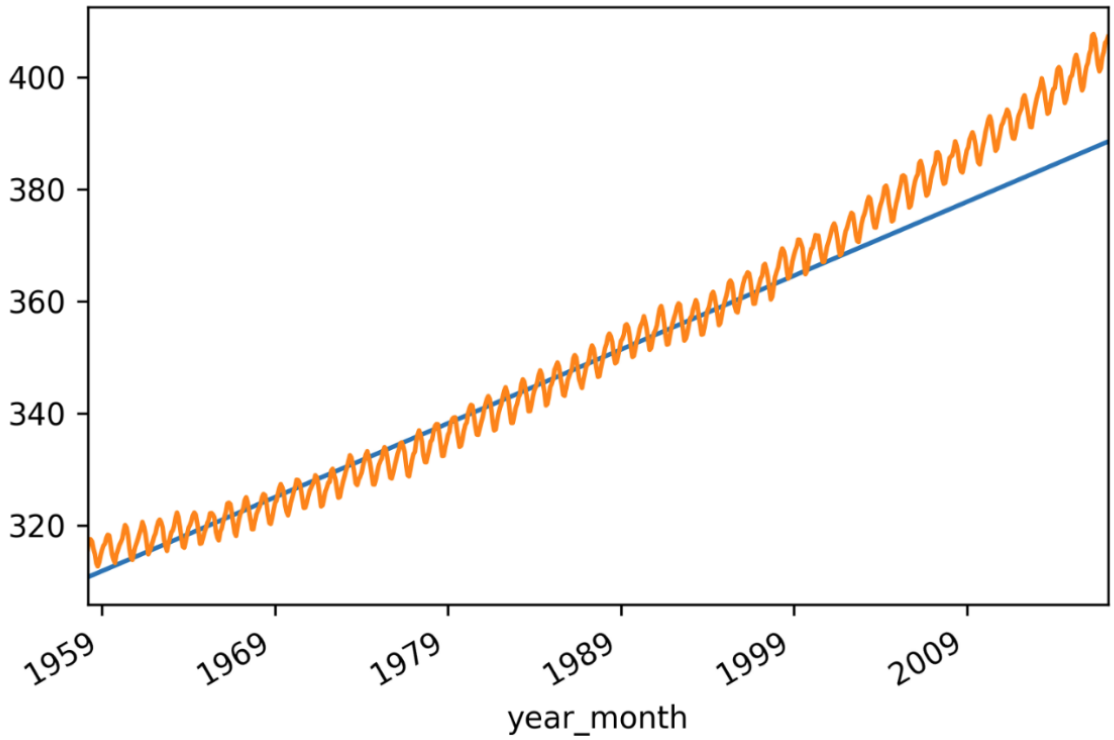
```
(709,)
```

```
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)
```

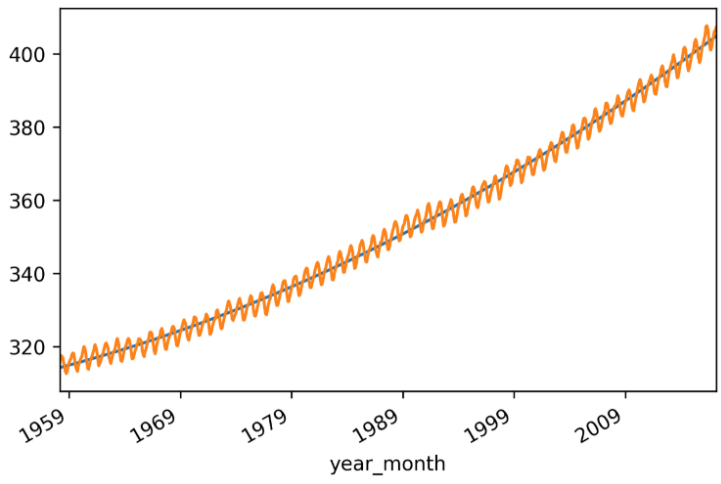
```
from sklearn.linear_model import LinearRegression
lr = LinearRegression().fit(X_train, train)
lr_pred = lr.predict(X_test)
plt.plot(ppm.index, lr.predict(X))
ppm.plot()
```

Linear model for trend



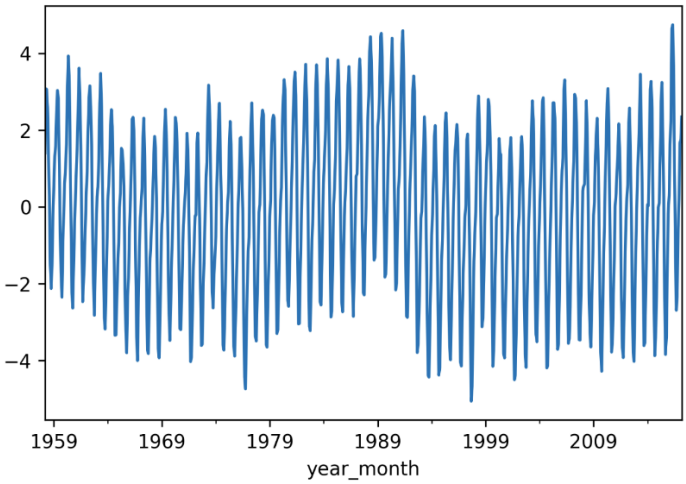
Quadratic Model

```
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()
```



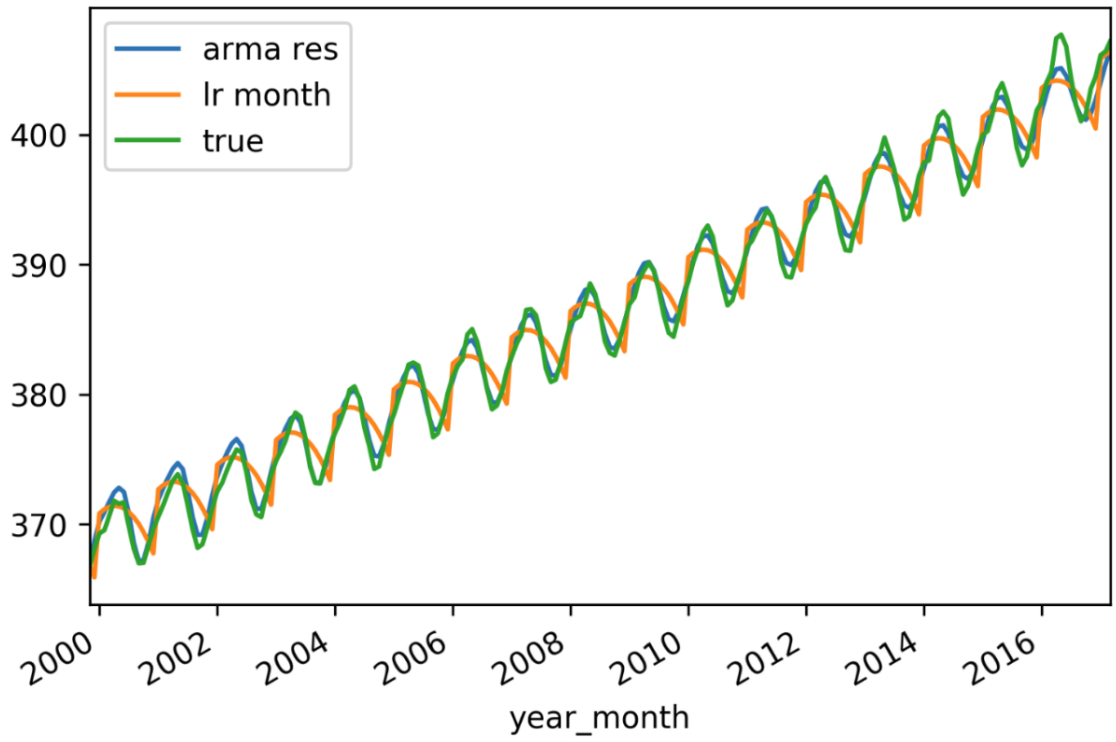
Si può fare Machine Lerning con le serie temporali usando solo anche solo la Regressione Lineare

Just Linear Regression

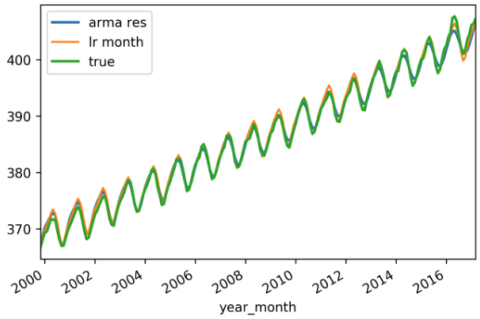
```
X_month = pd.concat([X, pd.DataFrame({'month': X.index.month},index=X.index)],
                    axis=1)
X_month.head()
```

year_month	year_month	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)
```



```
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)) #0.505
mean_squared_error(ppm[500:], pred_arma_res) #0.577
```

Time-related feature engineering

https://scikit-learn.org/stable/auto_examples/applications/plot_cyclical_feature_engineering.html

https://scikit-lego.netlify.app/preprocessing.html

Immaginate allora come questo “preprocessing” possa aiutare i GBM o le reti neurali - https://arxiv.org/abs/1907.05321

Time2Vec: Learning a Vector Representation of Time

Seyed Mehran Kazemi, Rishab Goel*, Sepehr Eghbali*, Janahan Ramanan, Jaspreet Sahota, Sanjay Thakur, Stella Wu, Cathal Smyth, Pascal Poupart, Marcus Brubaker
Borealis AI

Abstract

Time is an important feature in many applications involving events that occur synchronously and/or asynchronously. To effectively consume time information, recent studies have focused on designing new architectures. In this paper, we take an orthogonal but complementary approach by providing a model-agnostic vector representation for time, called *Time2Vec*, that can be easily imported into many existing and future architectures and improve their performances. We show on a range of models and problems that replacing the notion of time with its Time2Vec representation improves the performance of the final model.

1 Introduction

In building machine learning models, “time” is often an important feature. Examples include predicting daily sales for a company based on the date (and other available features), predicting the time for a patient’s next health event based on their medical history, and predicting the song a person is interested in listening to based on their listening history. The input for problems involving time can be considered as a sequence where, rather than being identically and independently distributed (*iid*), there exists a dependence across time (and/or space) among the data points. The sequence can be either synchronous, *i.e.* sampled at regular intervals, or asynchronous, *i.e.* sampled at different points in time. In both cases, time may be an important feature. For predicting daily sales, for instance, it may be useful to know if it is a holiday or not. For predicting the time for a patient’s next encounter, it is important to know the (asynchronous) times of their previous visits.

Recurrent neural networks (RNNs) [23, 9] have achieved impressive results on a range of sequence modeling problems. Most RNN models do not treat time itself as a feature, typically assuming that inputs are synchronous. When time is known to be a relevant feature, it is often fed in as yet another input dimension [10, 14, 35]. In practice, RNNs often fail at effectively making use of time as a feature. To help the RNN make better use of time, several researchers design hand-crafted features of time that suit their specific problem and feed those features into the RNN [10, 3, 29]. Hand-crafting features, however, can be expensive and requires domain expertise about the problem.

Il rischio di modello che esplicitamente dipende dal tempo è l'OVERFIT!

Il “teorema” di decomposizione

X(time) = Trend(time) + Seasonality(time) + Cycle(time) + ϵ(time)

Trend è una dipendenza sul lungo periodo (tipicamente immaginata come una tendenza lineare)

Seasonality è una periodicità di cui ci è nota la causa (ad esempio sui giorni del calendario)

Cycle è una periodicità anche approssimata di cui non sono note le cause

ϵ**psilon** è l’errore il residuo non spiegato rispetto al modello

T + S + C = Signal

“Teorema” è tra virgolette perché è ovvio usando il teorema di Fourier che ogni funzione temporale può essere decomposta in questo modo ma in realtà si tratta solo di un mindset di framework interpretativo o modellistico infatti in questo modo possiamo esplicitare le cause

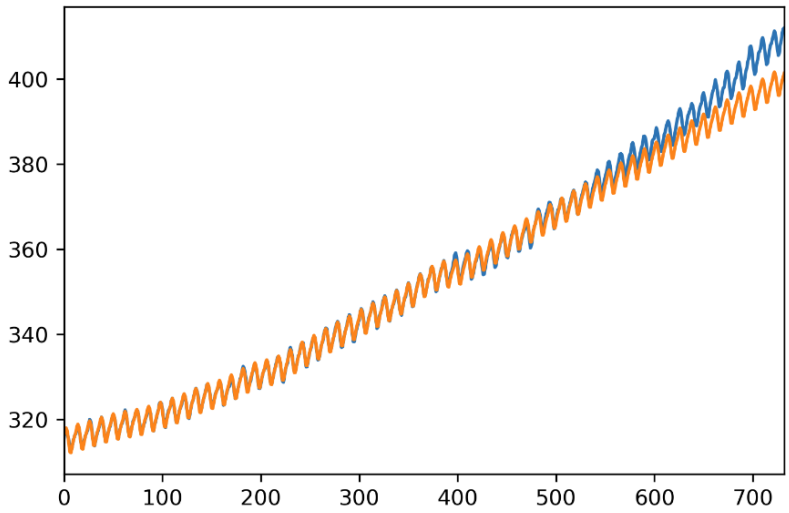
---

Seasonal Forecasting with FBProphet

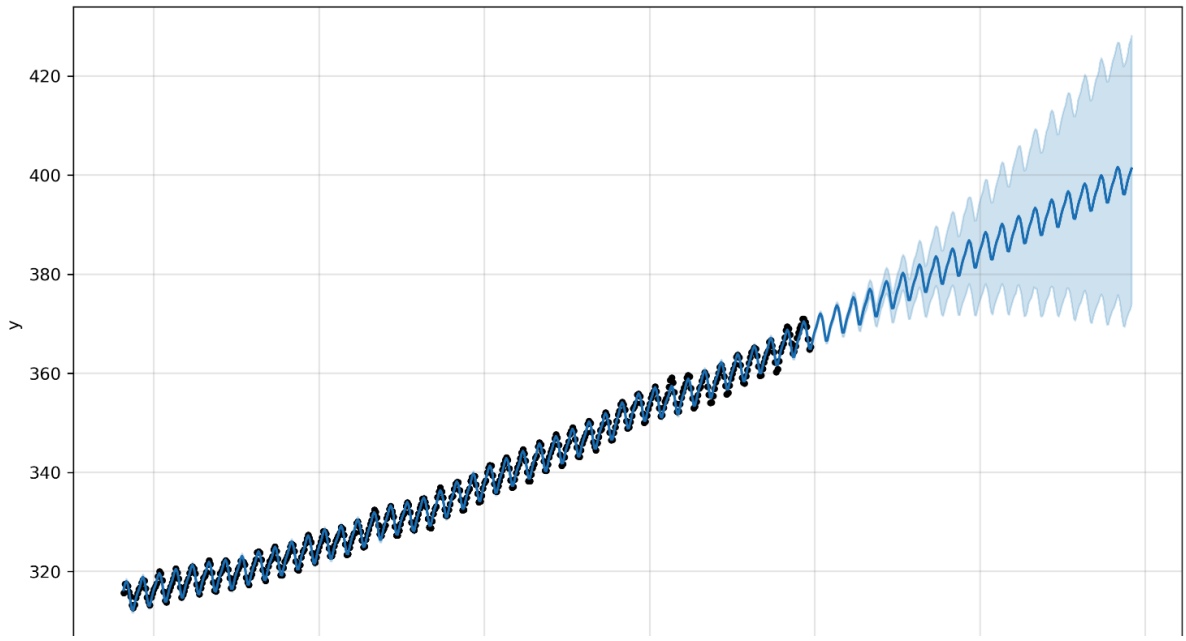
- A piecewise linear or logistic growth curve trend. Prophet automatically detects changes in trends by selecting changepoints from the data.
- A yearly seasonal component modeled using Fourier series.
- A weekly seasonal component using dummy variables.
- A user-provided list of important holidays.

https://research.fb.com/prophet-forecasting-at-scale/

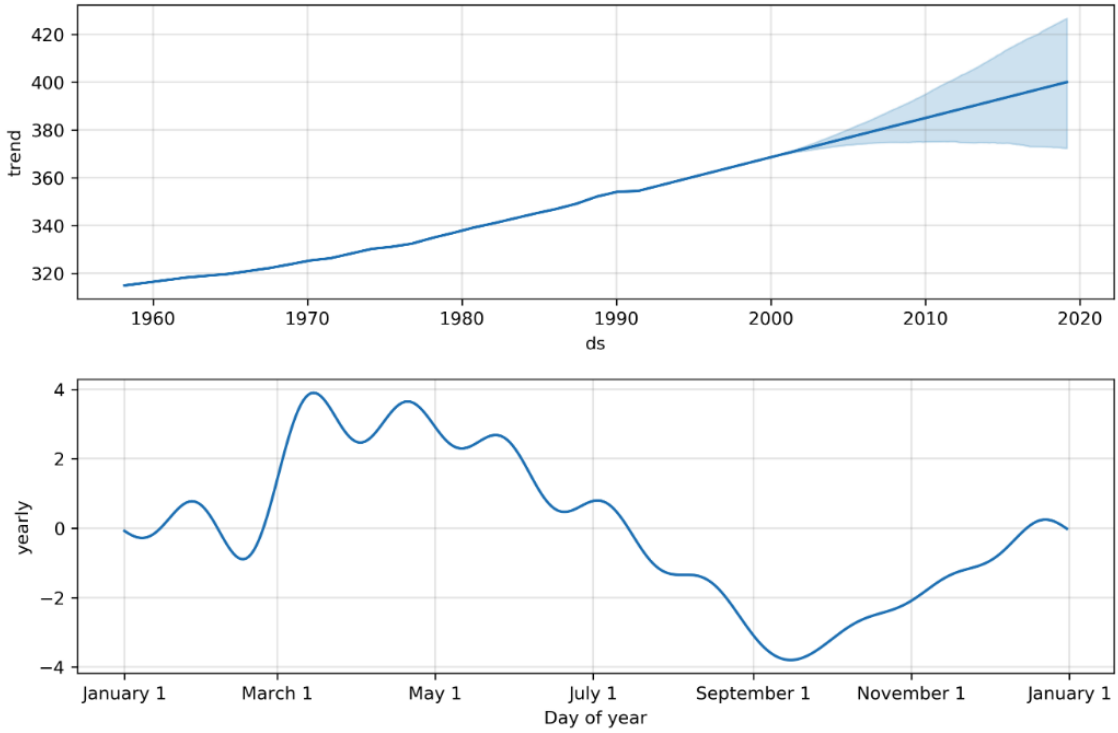
```
from fbprophet import Prophet
m = Prophet()
m.fit(fb_ppm[:500])
forecast = m.predict(fb_ppm)
```



Uncertainty prediction



Seasonal Components



Prophet è notorio per fare “schifo”

Dimostrazione che Prophet fa schifo

https://www.sarem-seitz.com/facebook-prophet-covid-and-why-i-dont-trust-the-prophet/

Dimostrazione che Prophet non fa schifo

MA è molto usato perché facilmente spiegabile e c’è un effetto di simmetria (i.e. FEEDBACK) interessante nelle Serie Temporali

Stocks Prediction Simmetria POSITIVA

Covid Prediction Simmetria NEGATIVA

in più può sempre essere sempre usato nei modelli ibridi

Modelli ibridi

https://towardsdatascience.com/boost-your-time-series-forecasts-combining-gradient-boosting-models-with-prophet-features-8e738234ffd

Other libraries

- sktime
  - Time series classification
  - Forecasting
- tslearn
  - Classification
  - Regression
  - Clustering
  - Very active development

- DARTS

Qual è il modello migliore? Ovviamente nessuno ma...

Le M Competition

https://forecasters.org/resources/time-series-data/

M3 - Theta model (**LR** + **ES**)

M4 - ES-RNN Model - Uber

(in seguito altri modelli di puro Deep Learning hanno raggiunto la medesima performance)

M5 - lightgbm

Where to go from here

- Gaussian Processes
- Recurrent Neural Nets / LSTMs
  - N-BEATS
  - Temporal Transformers

Inviare il lavoro a

https://forms.gle/qohbTnMGH4GEj9kE7

Esercizio per avere l'attestato di frequenza del corso

- Scaricare dati Mauna Loa
- split train/test 80/20

- Provare metodi tre/quattro metodi di previsione a vostra scelta, con e senza preprocessing o inventatene di vostri

  alcuni consigli:

  - ARIMA, Linear Regression sui lagged (**AR**), Linear regression su una qualche codifica del tempo (**LR**)
  - Random Forest (**RF**), KNN, Reti neurali
  - Prophet
  - **GBM** https://towardsdatascience.com/time-series-forecasting-with-machine-learning-b3072a5b44ba
  - provate a usare **Prophet + GBM** https://towardsdatascience.com/boost-your-time-series-forecasts-combining-gradient-boosting-models-with-prophet-features-8e738234ffd
  - fare ensemble facendo medie aritmetiche dei modelli
  - fare ensemble (boosting) facendo modelli Ibridi addestrando un modello sulla serie e uno sui residui di quello precedente $y - \hat{y_1}$
    - quindi la previsione è la somma delle due previsioni (tipo detrend) $\hat{y_2} + \hat{y_1}$
  - fare ensemble (boosting) addestrando il secondo modello su $y - 1/2 * \hat{y_1}$ (*tipo Theta Model*).
    - La previsione sarà metà della prima previsione più la seconda previsione $\hat{y_2} + \frac{1}{2}\hat{y_1}$. Funziona molto bene fare un AR+RF
  - fare ensemble aggiungendo le previsioni del primo modello tra le feature del secondo (*Chaining*)