

TIME SERIES ANALYSIS WITH PYTHON

VAMSIKRISHNA ATURU

vamsikrishna@gmail.com

INSTALLATION INSTRUCTIONS

- Please install Conda per 'quick install' instructions:
<http://conda.pydata.org/docs/install/quick.html>
- Make sure you have the following packages installed:
 - pandas
 - numpy
 - Statsmodels
 - scikit-learn
 - scipy
- These would be good to have but are not essential:
 - pytz
 - [hmmlearn](#)

OUTLINE

- Why time series?
- Quick Pandas intro
- Dealing with dates in Pandas
- Reading + manipulating time-stamped data
- Common time series analytical tools
- Prediction
- Classification

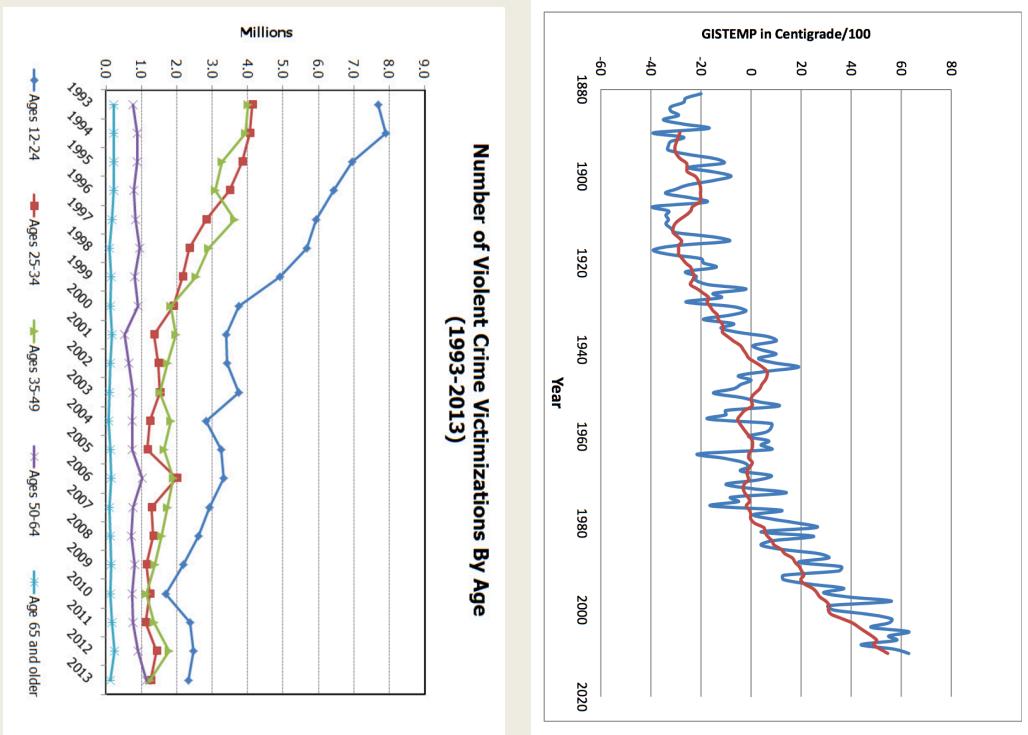
CAVEATS

- Time series analysis is a particularly tricky & controversial field
- I'll give some background as we move ahead, but **you need to read more** when you want to do a real analysis
- Tests for goodness of fit, etc, are particularly error prone in time series analysis
- Whenever I don't specify, but should, assume it's iid normally distributed ('error' terms)

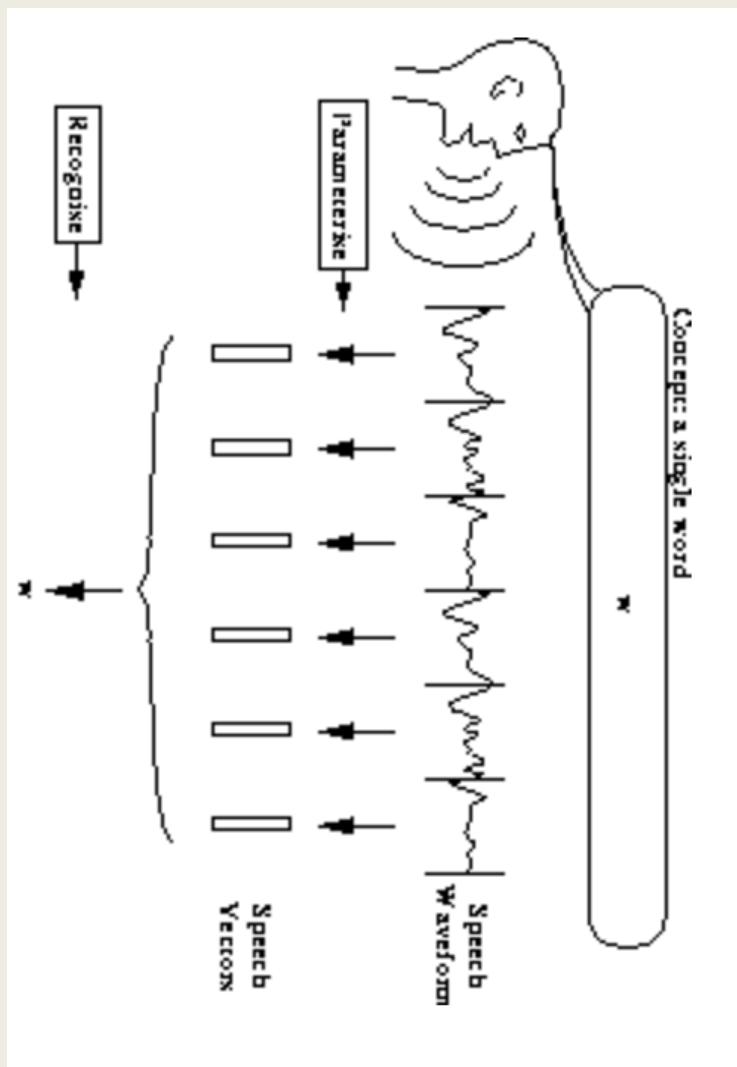
WHAT'S SPECIAL ABOUT
TIME SERIES?

WHERE DO TIME SERIES POP UP?

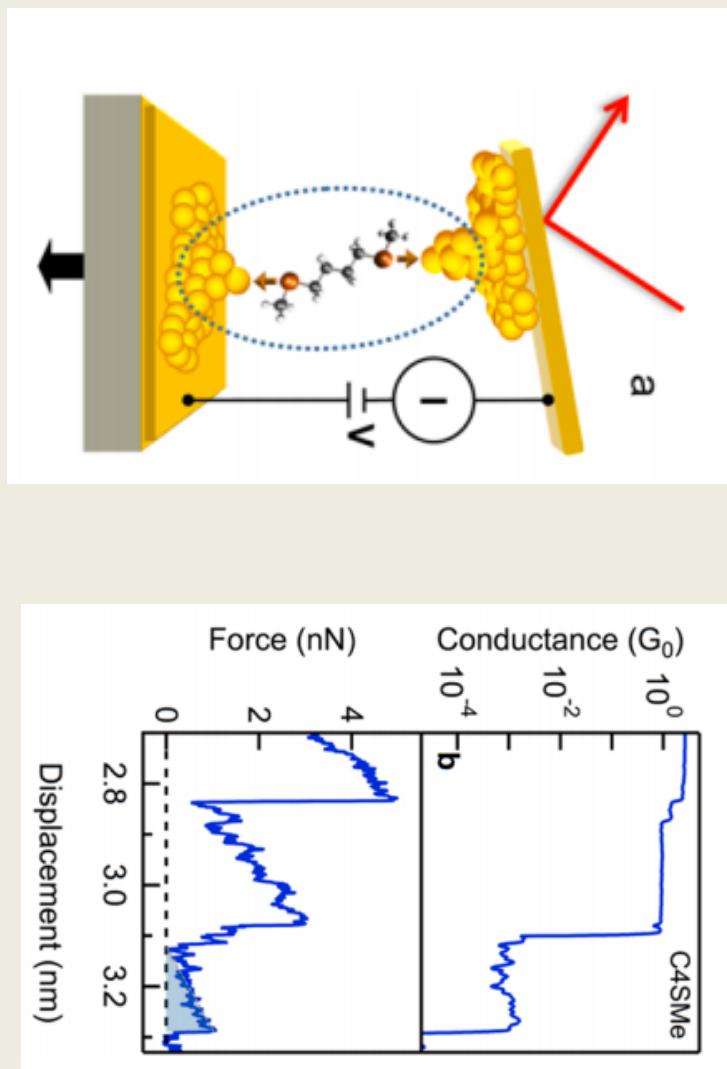
- Many of the most controversial questions arise from time series analysis
- Whenever we want to know the future, we're pretty much stuck with time series analysis
- Ditto for thinking about causality in ‘natural experiments’



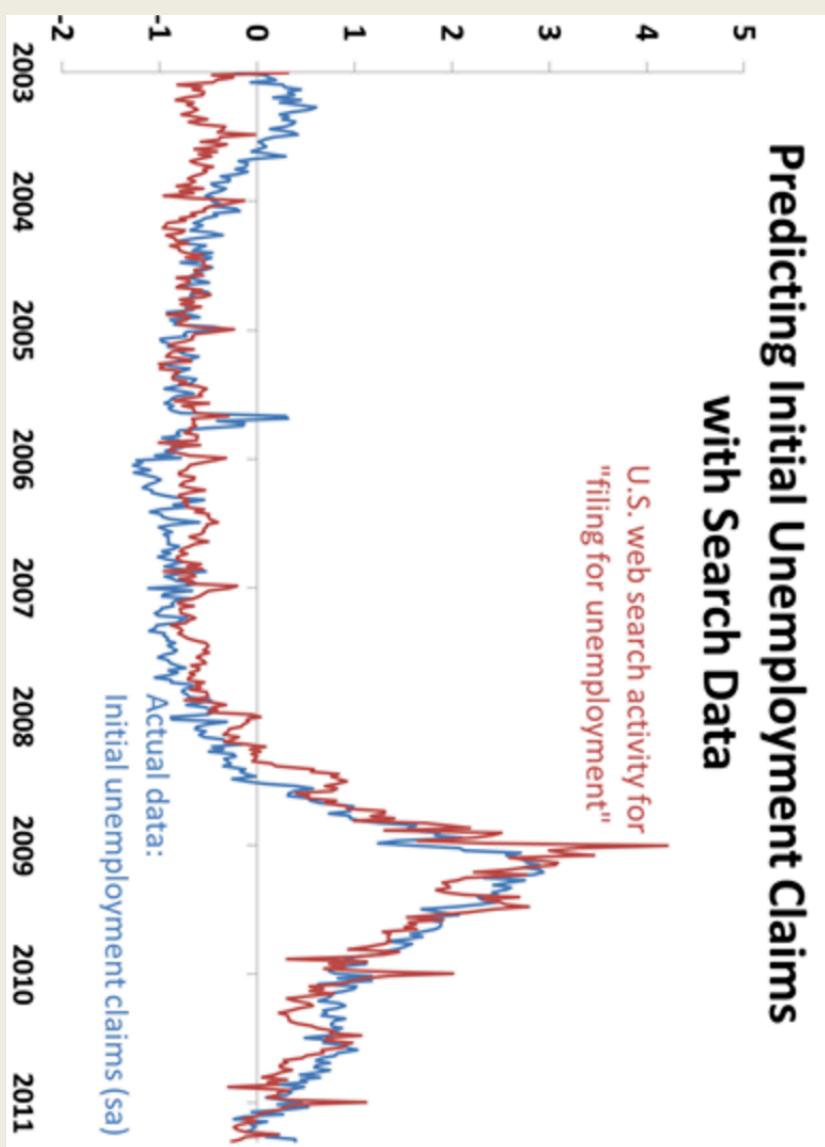
SPEECH RECOGNITION



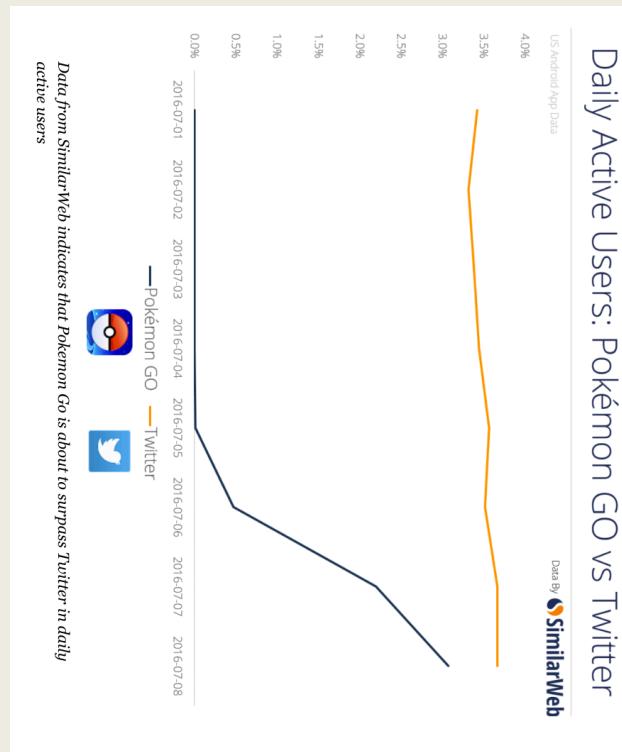
PHYSICS EXPERIMENTS



ECONOMICS, GOVERNMENT, POLICY

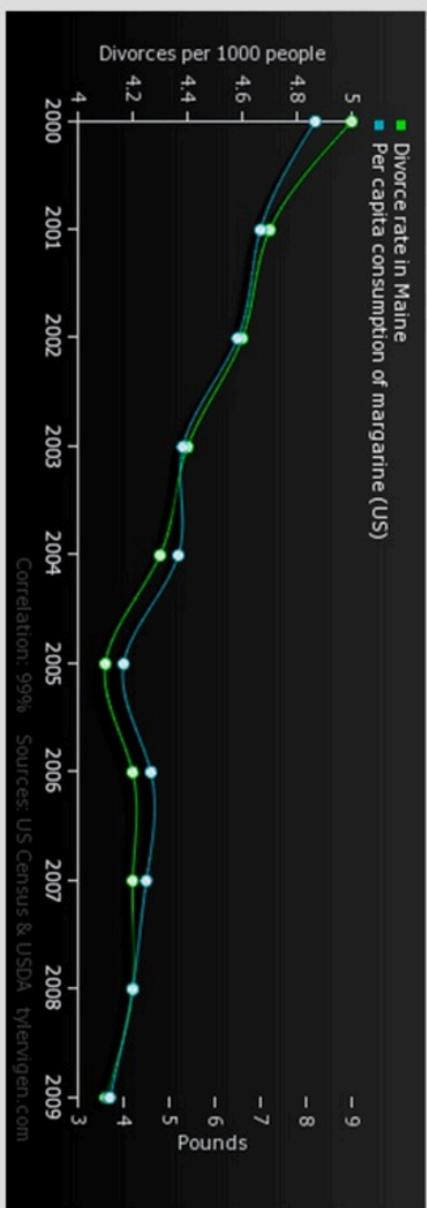


IN THE NEWS



**BE CAREFUL! IT'S ESPECIALLY TRUE
FOR TIME SERIES THAT YOU NEED
TO KNOW SOMETHING ABOUT
YOUR DATA**

Divorce rate in Maine
correlates with
Per capita consumption of margarine (US)



	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009
Divorce rate in Maine	5	4.7	4.6	4.4	4.3	4.1	4.2	4.2	4.2	4.1
Divorces per 1000 people (US Census)	5	4.7	4.6	4.4	4.3	4.1	4.2	4.2	4.2	4.1
Per capita consumption of margarine (US)	8.2	7	6.5	5.3	5.2	4	4.6	4.5	4.2	3.7
Pounds (USDA)										

Correlation: 0.992558

10 MINUTES TO PANDAS

PANDAS USES ‘DATA FRAMES’ TO PUT DATA IN AN EASY-TO-USE FORMAT

- All the convenience of a SQL-like API, but better
- Fast (*in-place*) if you know what you’re doing
- Fast read/write to standard storage formats like csv and databases
- Let’s look at a quick notebook of examples

PANDAS DATA FRAMES ARE BUILT FOR WHAT YOU WANT TO DO ANYWAY

If you're using IPython, tab completion for column names (as well as public attributes) is automatically enabled. Here's a subset of the attributes that will be completed:

```
In [13]: df2.<TAB>
df2.A
df2.abs
df2.add
df2.add_prefix
df2.add_suffix
df2.align
df2.all
df2.any
df2.append
df2.apply
df2.applymap
df2.as_blocks
df2.asfreq
df2.as_matrix
df2.astype
df2.at
df2.at_time
df2.axes
df2.B
df2.between_time
df2.bfill
df2.blocks
df2.bool
df2.D
df2.boxplot
df2.C
df2.clip
df2.clip_lower
df2.clip_upper
df2.columns
df2.combine
df2.combineAdd
df2.combineFirst
df2.combineMult
df2.compound
df2.consolidate
df2.convert_objects
df2.copy
df2.corr
df2.corrwith
df2.count
df2.cov
df2.cummax
df2.cummin
df2.cumprod
df2.cumsum
```

DEALING WITH TIME

PANDAS FUNCTIONALITY

- Generate sequences of fixed-frequency dates and time spans
- Conform or convert time series to a particular frequency
- Compute 'relative' dates based on various non-standard time increments (e.g. 5 business days before the last day of the year) or 'roll' dates backward and forward

PANDAS TIME-RELATED DATA TYPES

Overview

Following table shows the type of time-related classes pandas can handle and how to create them.

Class	Remarks	How to create
Timestamp	Represents a single time stamp	<code>to_datetime, Timestamp</code>
DatetimeIndex	Index of Timestamp	<code>to_datetime, date_range, DatetimeIndex</code>
Period	Represents a single time span	<code>Period</code>
PeriodIndex	Index of Period	<code>period_range, PeriodIndex</code>

DatetimeIndex

One of the main uses for `DatetimeIndex` is as an index for pandas objects. The `DatetimeIndex` class contains many timeseries related optimizations:

- A large range of dates for various offsets are pre-computed and cached under the hood in order to make generating subsequent date ranges very fast (just have to grab a slice)
- Fast shifting using the `shift` and `tshift` method on pandas objects
- Unioning of overlapping DatetimeIndex objects with the same frequency is very fast (important for fast data alignment)
- Quick access to date fields via properties such as `year`, `month`, etc.
- Regularization functions like `snap` and very fast `asof` logic

Time/Date Components

There are several time/date properties that one can access from `Timestamp` or a collection of timestamps like a `DatetimeIndex`.

Property	Description
<code>year</code>	The year of the datetime
<code>month</code>	The month of the datetime
<code>day</code>	The days of the datetime
<code>hour</code>	The hour of the datetime
<code>minute</code>	The minutes of the datetime
<code>second</code>	The seconds of the datetime
<code>microsecond</code>	The microseconds of the datetime
<code>nanosecond</code>	The nanoseconds of the datetime
<code>date</code>	Returns <code>datetime.date</code>
<code>time</code>	Returns <code>datetime.time</code>
<code>dayofyear</code>	The ordinal day of year
<code>weekofyear</code>	The week ordinal of the year
<code>week</code>	The week ordinal of the year
<code>dayofweek</code>	The number of the day of the week with Monday=0, Sunday=6
<code>weekday</code>	The number of the day of the week with Monday=0, Sunday=6
<code>weekday_name</code>	The name of the day in a week (ex: Friday)
<code>quarter</code>	Quarter of the date: Jan-Mar = 1, Apr-Jun = 2, etc.
<code>days_in_month</code>	The number of days in the month of the datetime
<code>is_month_start</code>	Logical indicating if first day of month (defined by frequency)
<code>is_month_end</code>	Logical indicating if last day of month (defined by frequency)
<code>is_quarter_start</code>	Logical indicating if first day of quarter (defined by frequency)
<code>is_quarter_end</code>	Logical indicating if last day of quarter (defined by frequency)
<code>is_year_start</code>	Logical indicating if first day of year (defined by frequency)
<code>is_year_end</code>	Logical indicating if last day of year (defined by frequency)

Offset Aliases

A number of string aliases are given to useful common time series frequencies. We will refer to these aliases as *offset aliases* (referred to as *time rules* prior to v0.8.0).

Alias	Description
B	business day frequency
C	custom business day frequency (experimental)
D	calendar day frequency
W	weekly frequency
M	month end frequency
BM	business month end frequency
CBM	custom business month end frequency
MS	month start frequency
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

DateOffset objects

In the preceding examples, we created DatetimeIndex objects at various frequencies by passing in frequency strings like 'M', 'W', and 'BM' to the `freq` keyword. Under the hood, these frequency strings are being translated into an instance of pandas `DateOffset`, which represents a regular frequency increment. Specific offset logic like "month", "business day", or "one hour" is represented in its various subclasses.

Class name	Description
<code>DateOffset</code>	Generic offset class, defaults to 1 calendar day
<code>BDay</code>	business day (weekday)
<code>CDay</code>	custom business day (experimental)
<code>Week</code>	one week, optionally anchored on a day of the week
<code>WeekOfMonth</code>	the x-th day of the y-th week of each month
<code>LastWeekOfMonth</code>	the x-th day of the last week of each month
<code>MonthEnd</code>	calendar month end
<code>MonthBegin</code>	calendar month begin
<code>BMonthEnd</code>	business month end
<code>BMonthBegin</code>	business month begin
<code>CBMonthEnd</code>	custom business month end
<code>CBMonthBegin</code>	custom business month begin
<code>QuarterEnd</code>	calendar quarter end
<code>QuarterBegin</code>	calendar quarter begin
<code>BQuarterEnd</code>	business quarter end
<code>BQuarterBegin</code>	business quarter begin
<code>FY5253Quarter</code>	retail (aka 52-53 week) quarter
<code>YearEnd</code>	calendar year end
<code>YearBegin</code>	calendar year begin
<code>BYearEnd</code>	business year end
<code>BYearBegin</code>	business year begin
<code>FY5253</code>	retail (aka 52-53 week) year
<code>BusinessHour</code>	business hour
<code>CustomBusinessHour</code>	custom business hour
<code>Hour</code>	one hour
<code>Minute</code>	one minute
<code>Second</code>	one second
<code>Milli</code>	one millisecond
<code>Micro</code>	one microsecond
<code>Nano</code>	one nanosecond

Anchored Offsets

For some frequencies you can specify an anchoring suffix:

Alias	Description
W-SUN	weekly frequency (sundays). Same as 'W'
W-MON	weekly frequency (mondays)
W-TUE	weekly frequency (tuesdays)
W-WED	weekly frequency (wednesdays)
W-THU	weekly frequency (thursdays)
W-FRI	weekly frequency (fridays)
W-SAT	weekly frequency (saturdays)
(B)Q(S)-DEC	quarterly frequency, year ends in December. Same as 'Q'
(B)Q(S)-JAN	quarterly frequency, year ends in January
(B)Q(S)-FEB	quarterly frequency, year ends in February
(B)Q(S)-MAR	quarterly frequency, year ends in March

Holidays / Holiday Calendars

Holidays and calendars provide a simple way to define holiday rules to be used with `CustomBusinessDay` or in other analysis that requires a predefined set of holidays. The `AbstractHolidayCalendar` class provides all the necessary methods to return a list of holidays and only rules need to be defined in a specific holiday calendar class. Further, `start_date` and `end_date` class attributes determine over what date range holidays are generated. These should be overwritten on the `AbstractHolidayCalendar` class to have the range apply to all calendar subclasses. `usfederalHolidayCalendar` is the only calendar that exists and primarily serves as an example for developing other calendars.

For holidays that occur on fixed dates (e.g., US Memorial Day or July 4th) an observance rule determines when that holiday is observed if it falls on a weekend or some other non-observed day. Defined observance rules are:

Rule	Description
<code>nearest_workday</code>	move Saturday to Friday and Sunday to Monday
<code>sunday_to_monday</code>	move Sunday to following Monday
<code>next_monday_or_tuesday</code>	move Saturday to Monday and Sunday/Monday to Tuesday
<code>previous_friday</code>	move Saturday and Sunday to previous Friday"
<code>next_monday</code>	move Saturday and Sunday to following Monday

An example of how holidays and holiday calendars are defined:

pandas.DataFrame.asfreq

`DataFrame.asfreq(freq, method=None, how=None, normalize=False)`

Convert all TimeSeries inside to specified frequency using DateOffset objects. Optionally provide fill method to pad/backfill missing values.

Parameters:

freq : *DateOffset object, or string*

method : {‘backfill’, ‘bfill’, ‘pad’, ‘ffill’, None}

Method to use for filling holes in reindexed Series pad / ffill: propagate last valid observation forward to next valid backfill / bfill: use NEXT valid observation to fill method

how : {‘start’, ‘end’}, default end

For PeriodIndex only, see PeriodIndex.asfreq

normalize : bool, default False

Whether to reset output index to midnight

Returns:

converted : type of caller

pandas.DataFrame.resample

`DataFrame.resample(rule, how=None, axis=0, fill_method=None, closed=None, label=None, convention='start', kind=None, loffset=None, limit=None, base=0)`

Convenience method for frequency conversion and resampling of regular time-series data.

Parameters:

`rule : string`

the offset string or object representing target conversion

`axis : int, optional, default 0`

`closed : {'right', 'left'}`

Which side of bin interval is closed

`label : {'right', 'left'}`

Which bin edge label to label bucket with

`convention : {'start', 'end', 's', 'e'}`

`loffset : timedelta`

Adjust the resampled time labels

`base : int, default 0`

For frequencies that evenly subdivide 1 day, the "origin" of the aggregated intervals.

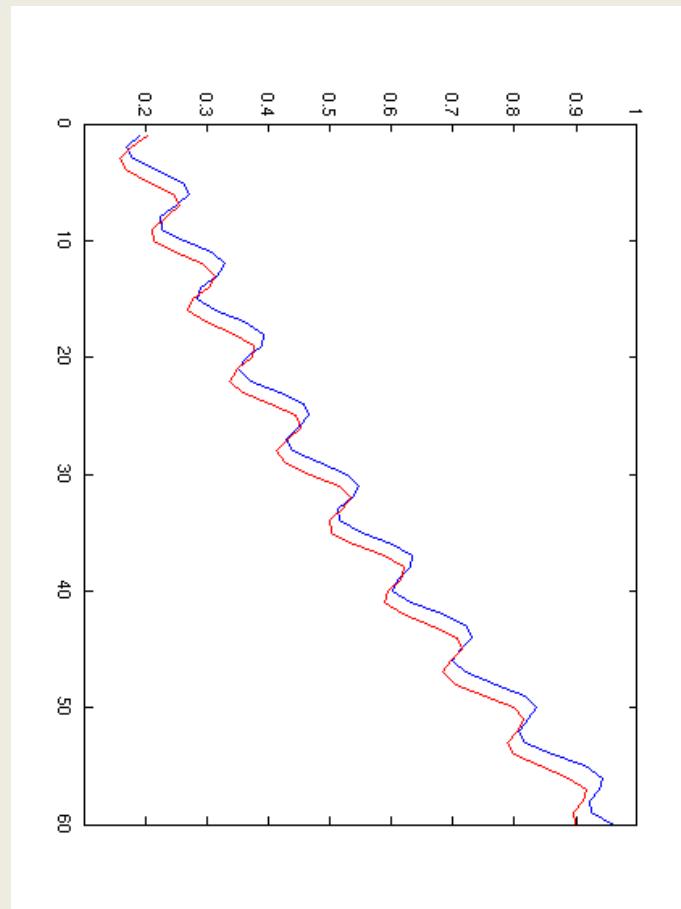
For example, for '5min' frequency, base could range from 0 through 4. Defaults to 0

LAG FUNCTIONS

File Edit View Data Transform Analyze Direct Marketing Graphs Ut

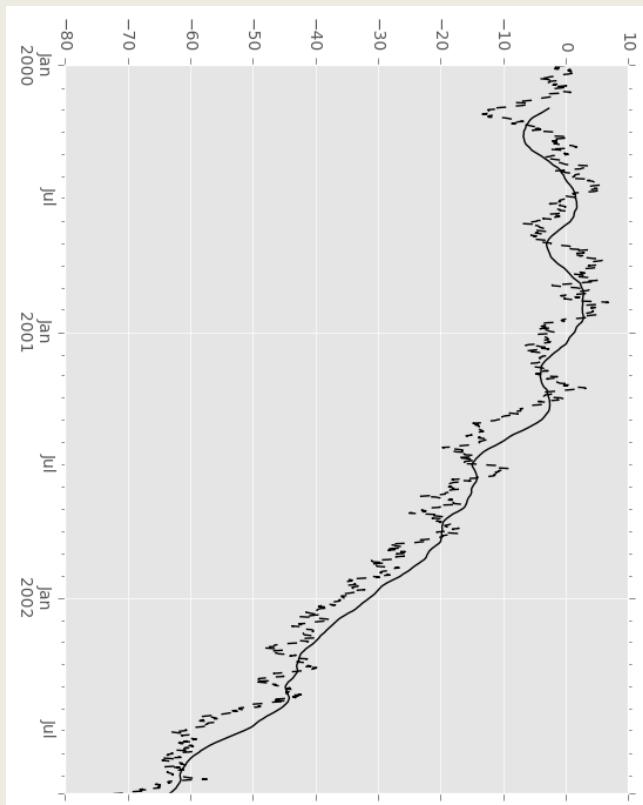
compute previous_id = lag(id).

	id	previous_id	var	var
1	1.00			
2	2.00	1.00		
3	2.00	2.00		
4	3.00	2.00		
5	3.00	3.00		
6	3.00	3.00		
7	4.00	3.00		
8	4.00	4.00		
9	4.00	4.00		
10	4.00	4.00		

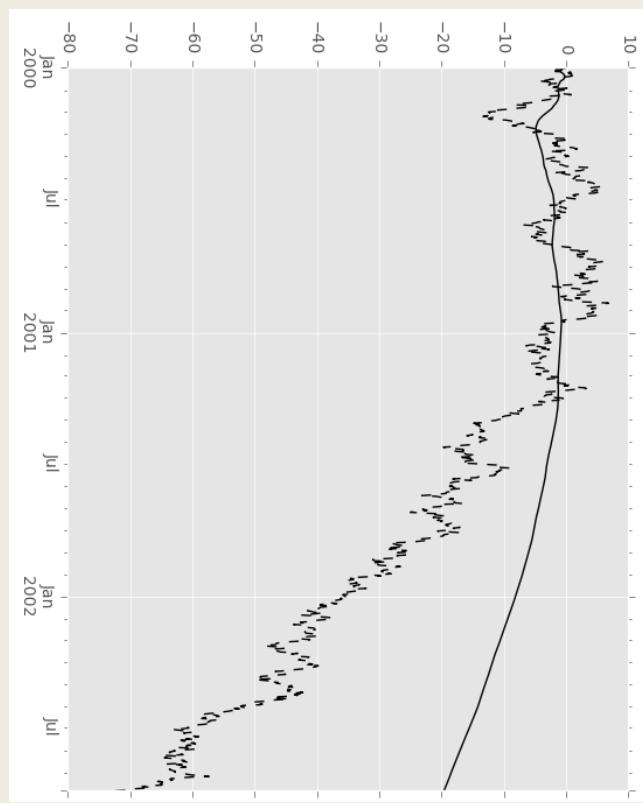


WINDOW FUNCTIONS

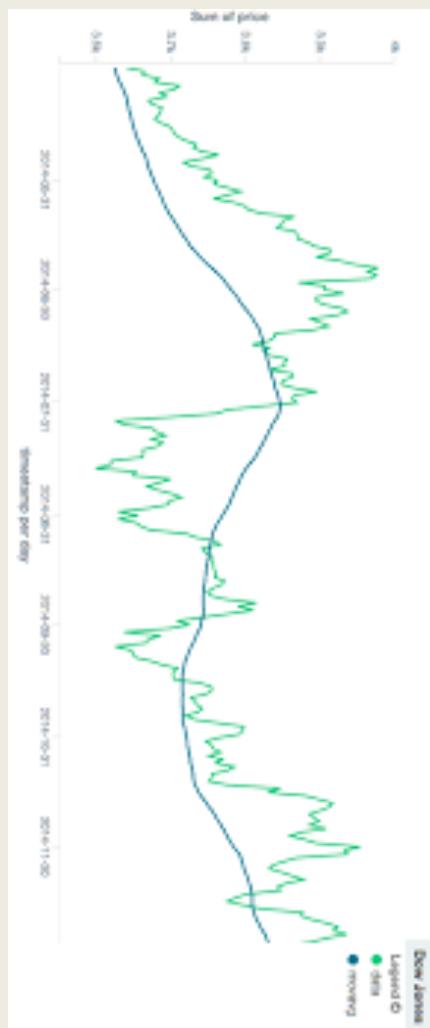
Rolling window



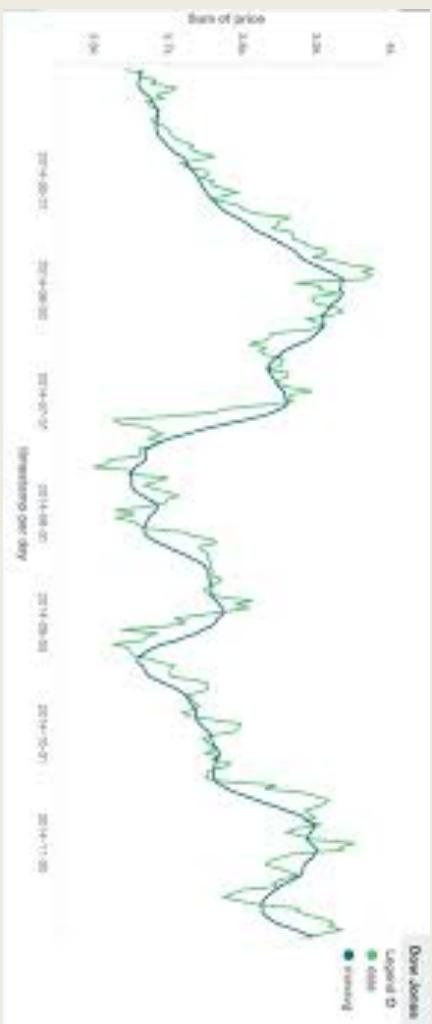
Expanding window



Why use a rolling window function?

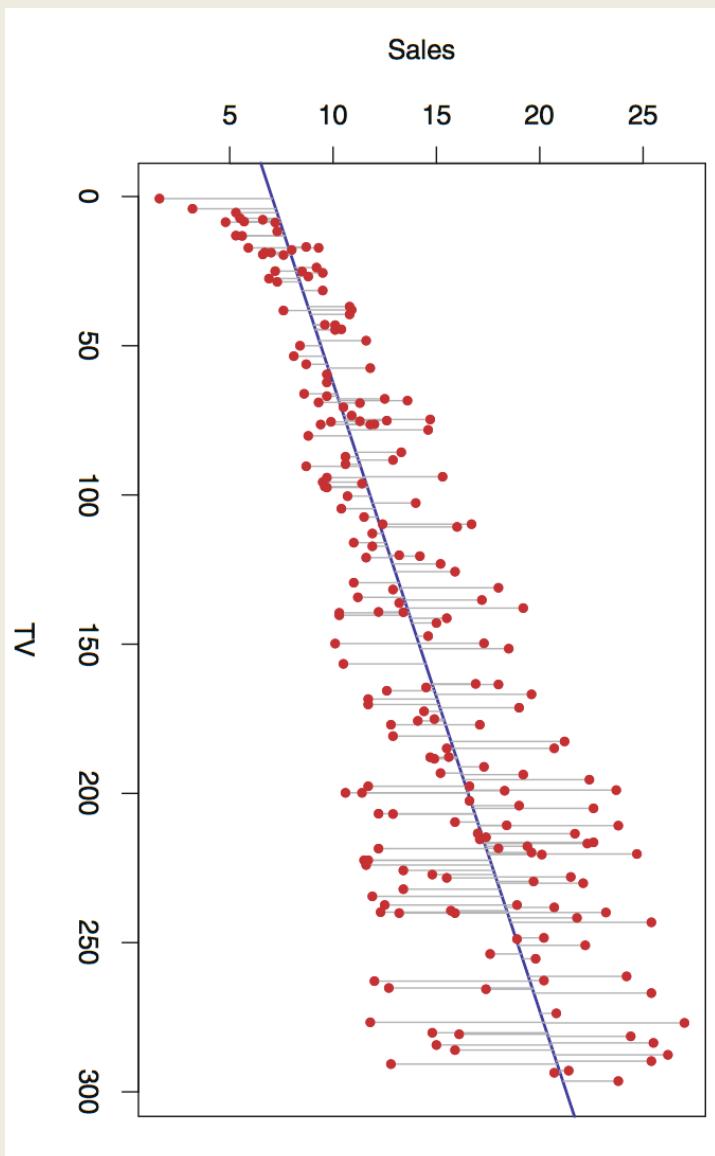


What's a little funky here?



LINEAR REGRESSION

LINEAR REGRESSION INTUITION



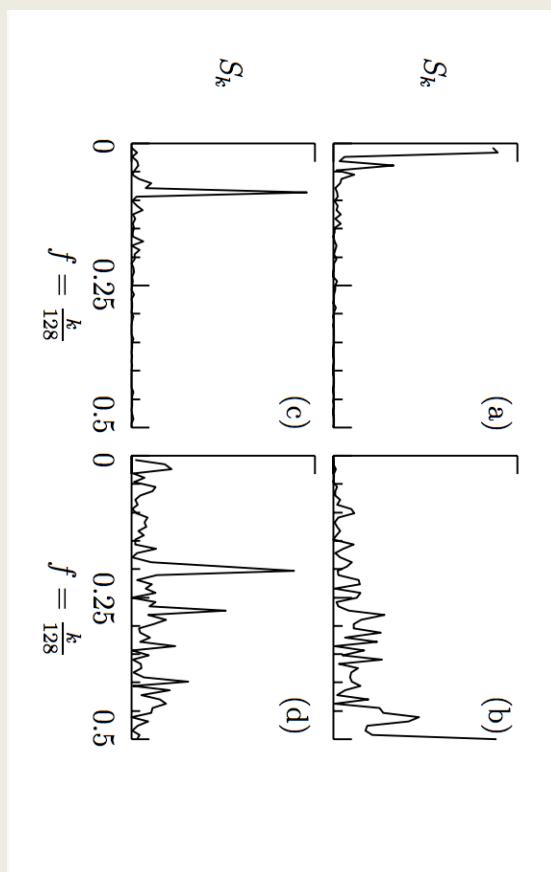
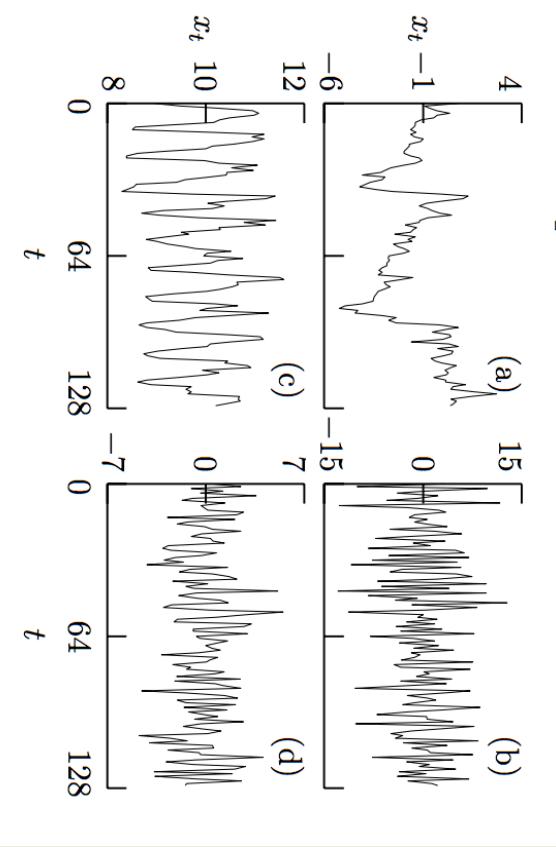
SPECTRAL ANALYSIS

INTUITION

- Decompose a time series into a sum of many many sine or cosine functions
- The coefficients of these functions should have uncorrelated values
- Regression on sinusoids

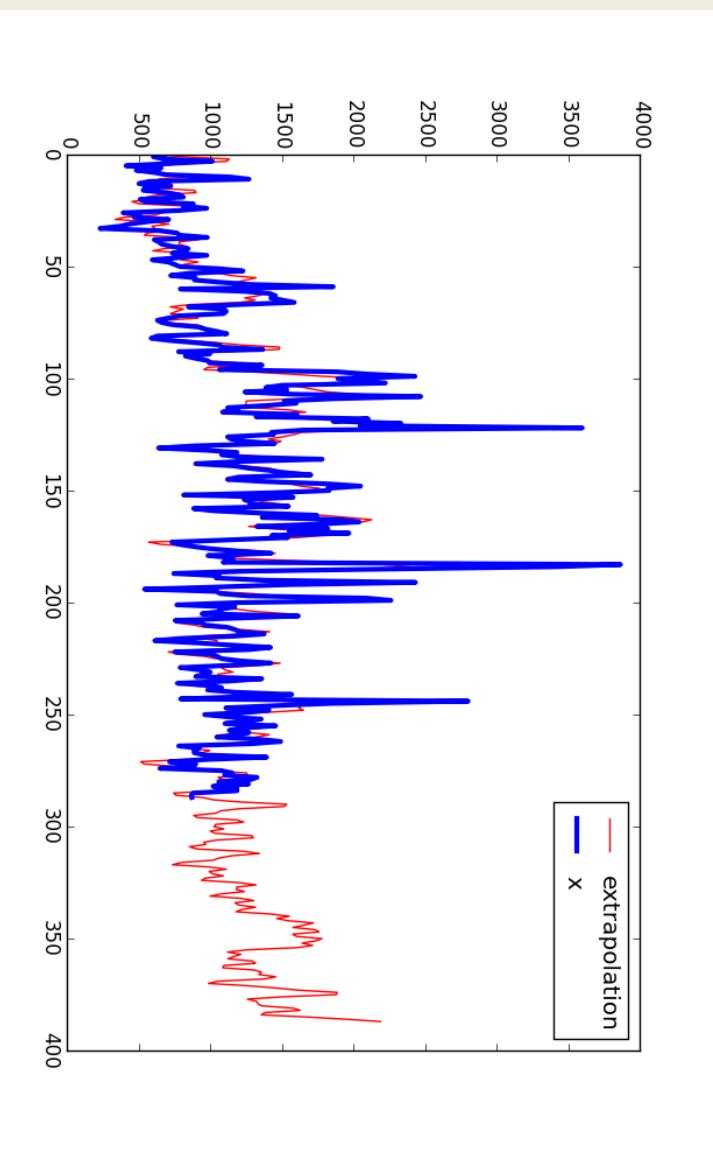


EXAMPLES



1. What are the advantages?
2. When would this provide useful information?
3. When would this *not* provide useful information?

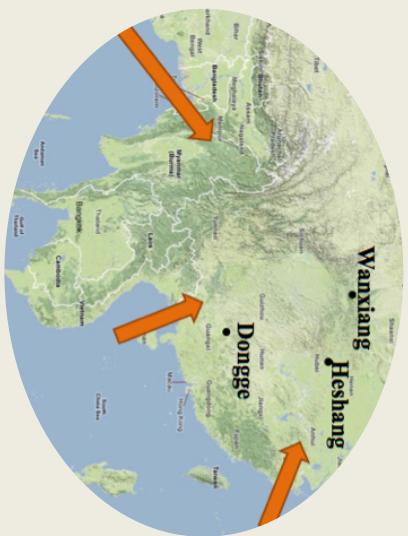
SPECTRA-BASED FIT CAN BE SURPRISINGLY GOOD



SPECTRAL ANALYSIS TURNS UP EVERWHERE



Astronomical data



Paleo-climate proxy data

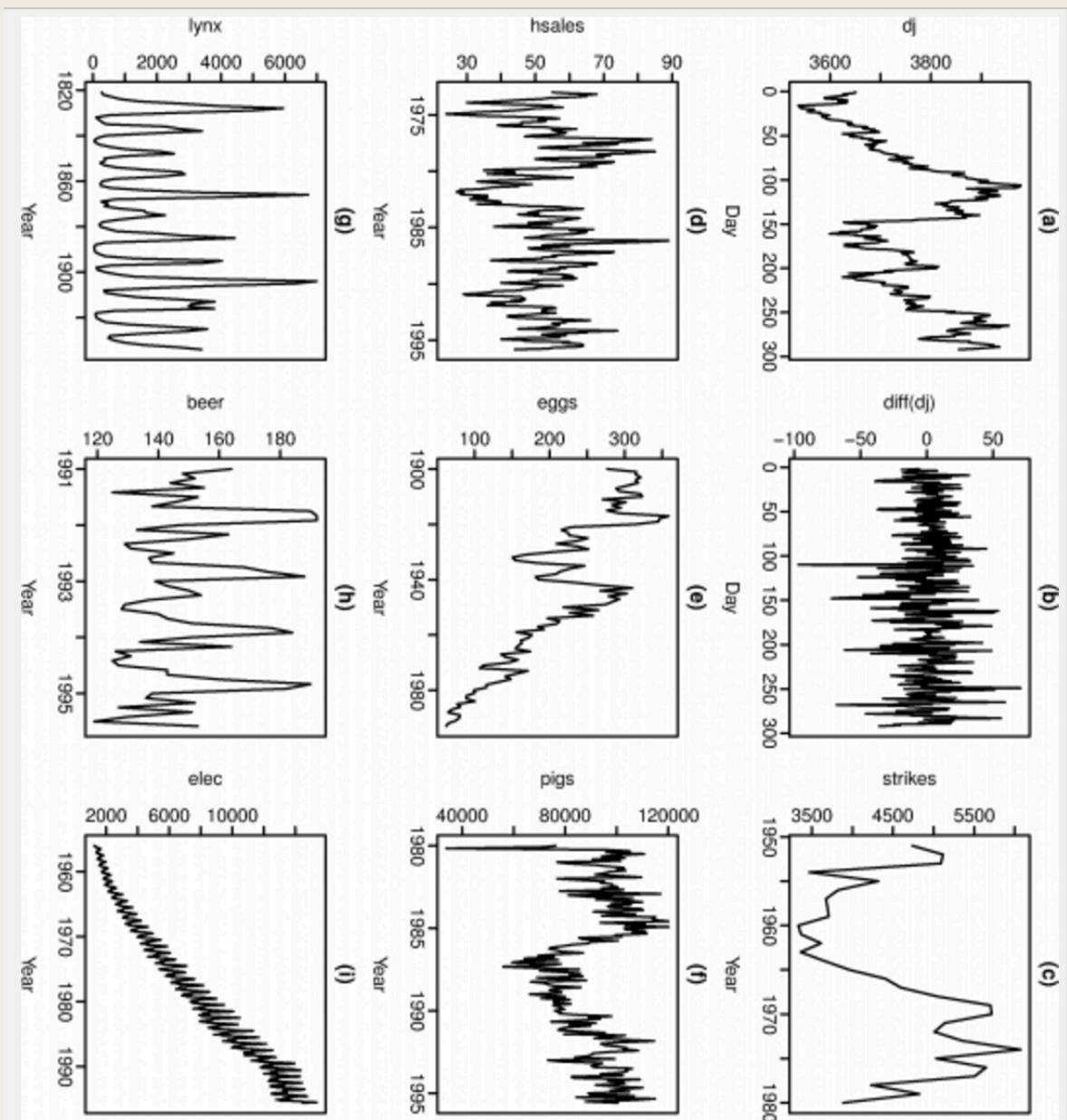


Biology experiments

PRE-PREDICTION MUNGING & STATIONARITY

STATIONARY TIME SERIES

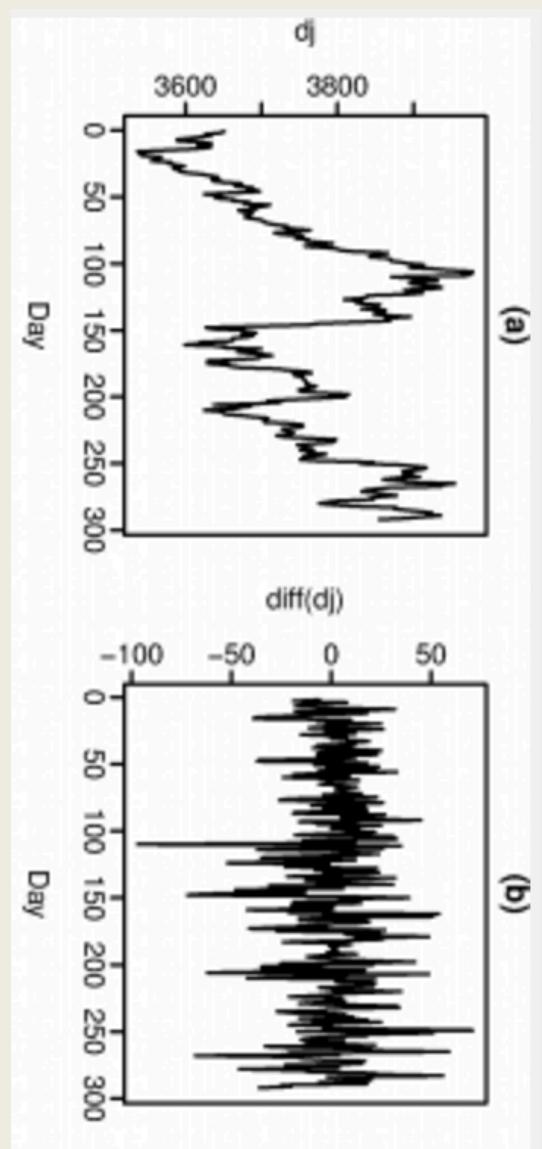
... NOT ALWAYS OBVIOUS



- (a) Dow Jones index on 292 consecutive days; (b) Daily change in Dow Jones index on 292 consecutive days; (c) Annual number of strikes in the US; (d) Monthly sales of new one-family houses sold in the US; (e) Price of a dozen eggs in the US (constant dollars); (f) Monthly total of pigs slaughtered in Victoria, Australia; (g) Annual total of lynx trapped in the McKenzie River district of northwest Canada; (h) Monthly Australian beer production; (i) Monthly Australian electricity production

Stationary: (b) & (g)

DIFFERENCING TO CREATE STATIONARY TIME SERIES

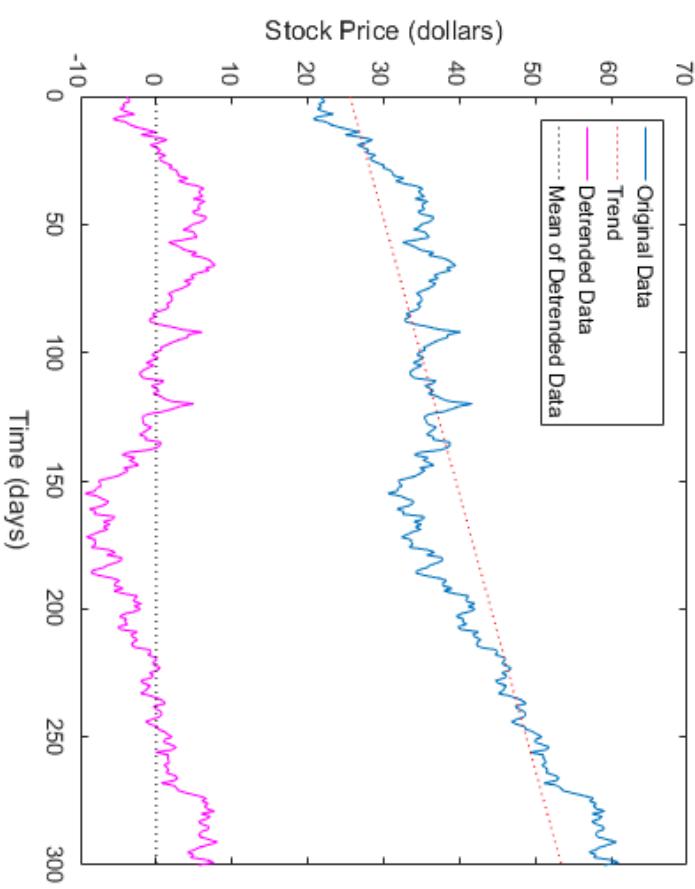
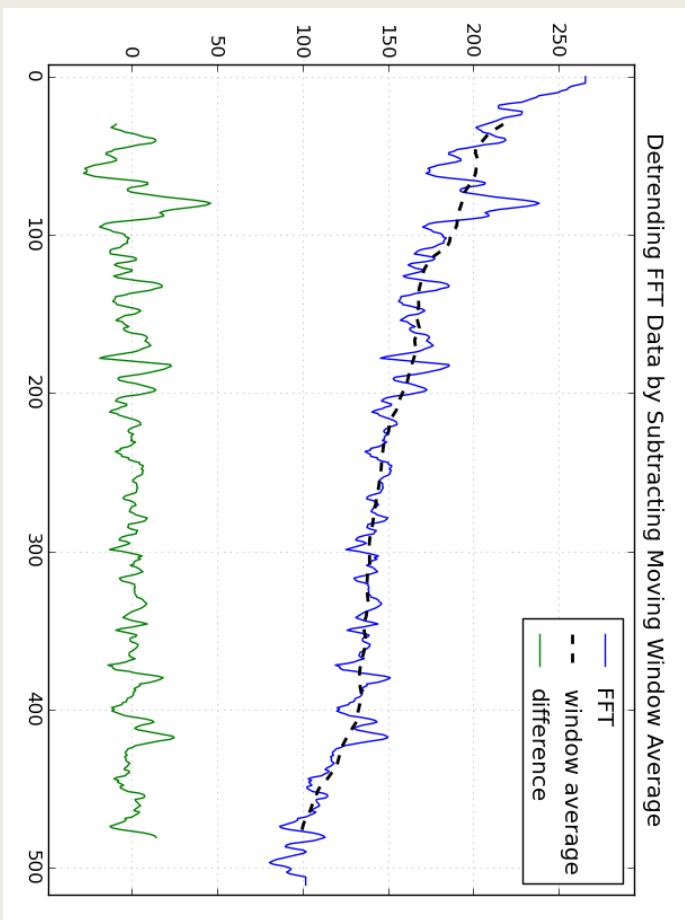


YOU NEED TO REMOVE THE TREND AND SEASONAL ELEMENTS BEFORE FORECASTING

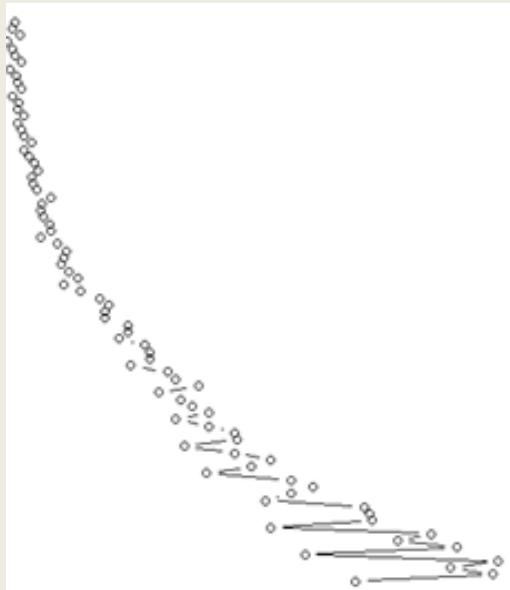
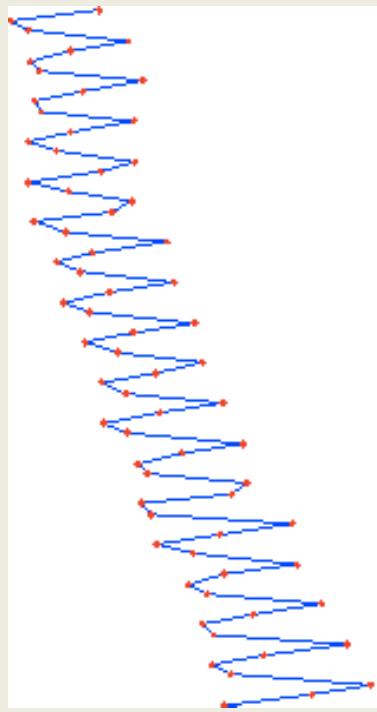
- Most (interesting) data in the real world will show
 - Trends
 - Seasonality
- Most models require data that shows neither of these properties to say something interesting
- In particular, we need a **stationary** time series

DE-TREND YOUR DATA

Use local smoothing or a linear regression



SEASONALITY



REMOVE SEASONALITY

- Simplest: average de-trended values for specific season
- More common: use ‘loess’ method (“locally weighted scatterplot smoothing”)
 - Window of specified width is placed over the data
 - A weighted regression line or curve is fitted to the data, with points closest to center of curve having greatest weight
 - Weighting is reduced on points farthest from regression line/curve and calculation is rerun several times.
- This yields one point on loess curve
- Helps reduce impact of outlier points
- Computationally taxing

DICKEY-FULLER TEST

- Tests the null hypothesis of whether a unit root is present in an autoregressive model
- In plain English, tests whether $\rho = 1$ in

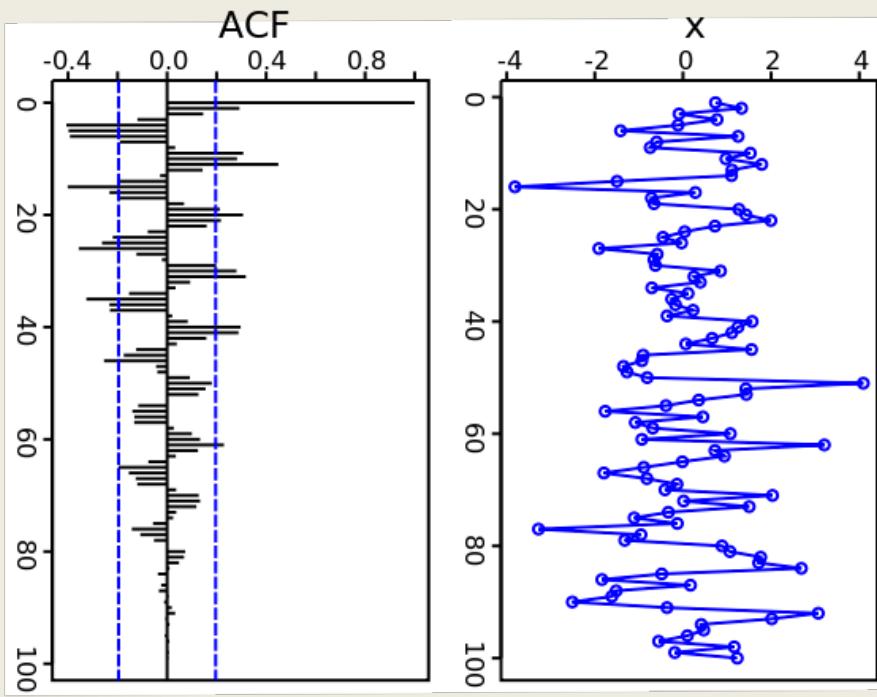
$$y_t = \rho y_{t-1} + u_t$$

- The test gives back several values to help you assess significance with standard p-value reasoning.
- Basic intuition: ρ should not have unit value

SELF-CORRELATION, SELF-
EXPLANATION, AND SELF-
PREDICTION

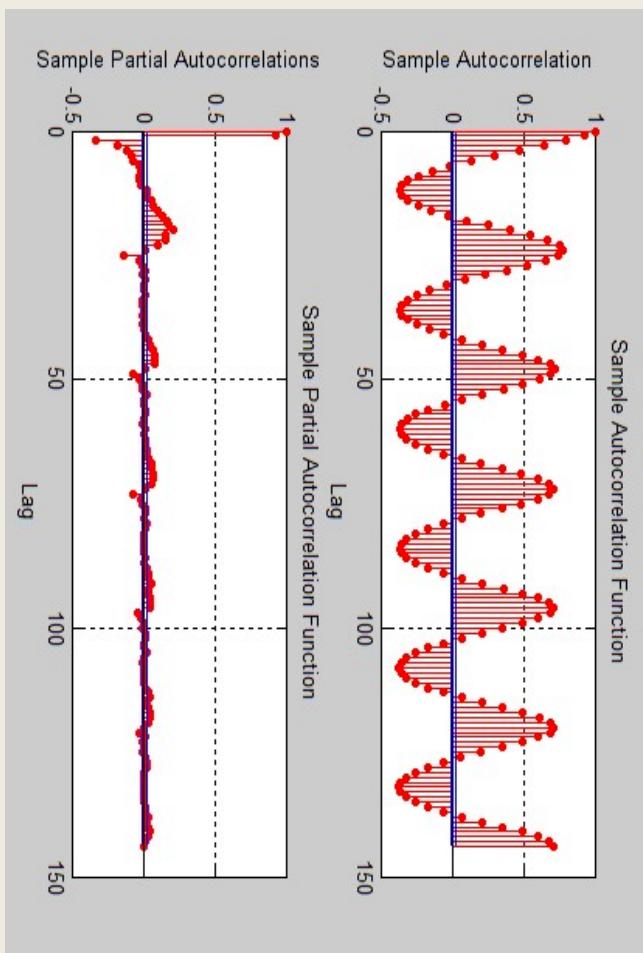
AUTOCORRELATION FUNCTION

- Used to help identify possible structures of time series data
- Gives a sense of how different points in time relate to each other in a way explained by temporal distance



PARTIAL AUTOCORRELATION FUNCTION

- “gives the partial correlation of a time series with its own lagged values, controlling for the values of time series at all shorter lags”
- Why would this be useful?



FORECASTING

MOVING AVERAGE PROCESS (MA)

- Defined as having the form:

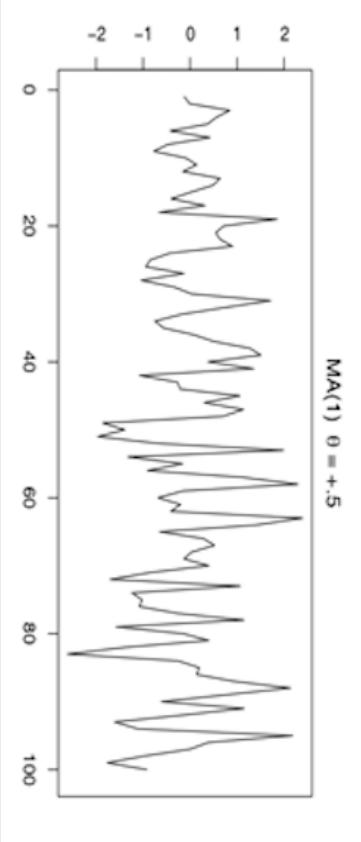
$$X_t = \mu + \varepsilon_t + \theta_1 \varepsilon_{t-1} + \dots + \theta_q \varepsilon_{t-q}$$

- μ is the mean of the series, θ are parameters, θ_q not 0
- This is a stationary process regardless of values of θ

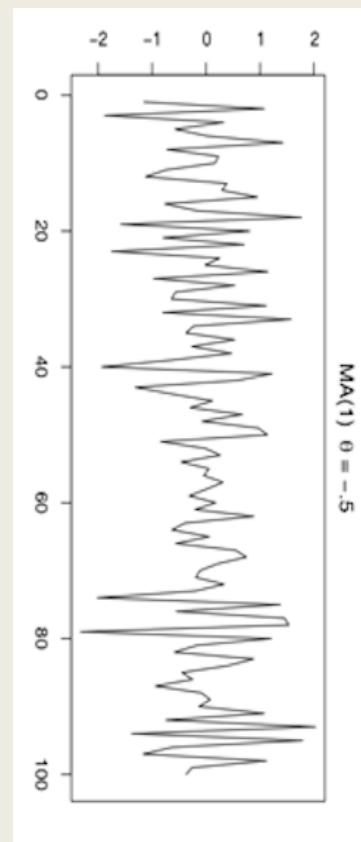
- Consider an MA(1) process (centered at 0):

$$X_t = \varepsilon_t + \theta_1 \varepsilon_{t-1}$$

$$\theta_1 = +.5$$



$$\theta_1 = -.5$$



AUTOREGRESSIVE PROCESS (AR)

- Defined as having the form:

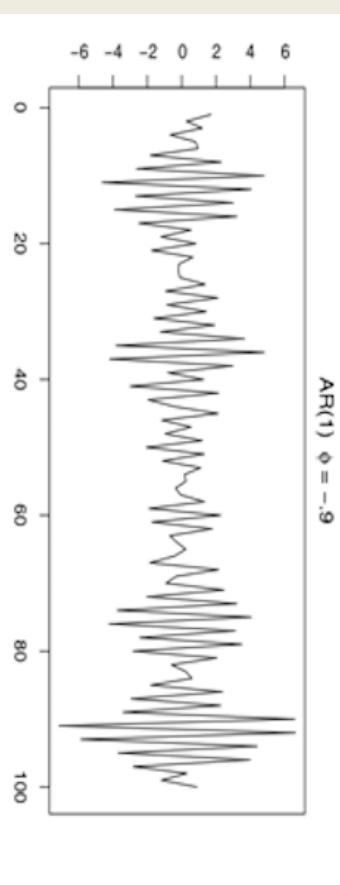
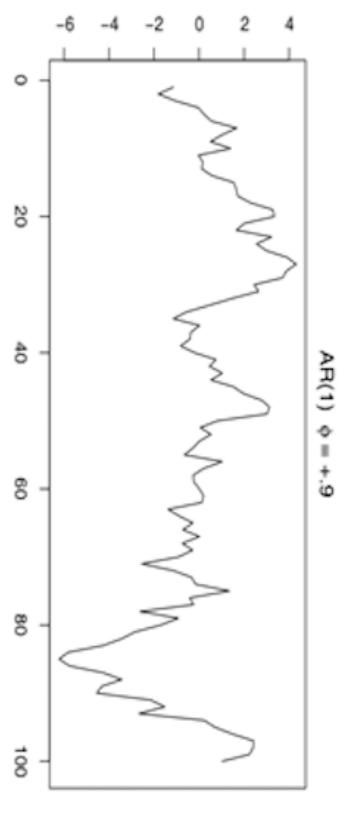
$$X_t = \phi_1 X_{t-1} + \dots + \phi_p X_{t-p} + \varepsilon_t$$

- This is a stationary process if $\text{abs}(\phi) < 1$
- Consider an AR(1) process:

$$X_t = \phi_1 X_{t-1} + \varepsilon_t$$

$$\phi_1 = +.9$$

$$\phi_1 = -.9$$



ARIMA MODEL (A.K.A. BOX-JENKINS)

- AR = autoregressive terms
- I = differencing
- MA = moving average
- Hence specified as (autoregressive terms, differencing terms, moving average terms)

ARIMA MODE: ‘THE MOST GENERAL CLASS OF MODELS FOR FORECASTING A TIME SERIES WHICH CAN BE MADE TO BE STATIONARY

- Statistical properties (mean, variance) constant over time
- ‘its short-term random time patterns always look the same in a statistical sense’
- Autocorrelation function & power spectrum remain constant over time
- Ok to do non-linear transformations to get there
- ARIMA model can be viewed as a combination of signal ad noise
- Extrapolate the signal to obtain forecasts

APPLYING THE APPROPRIATE ARIMA MODEL

- Need to determine what ARIMA model to use
- Use plot of the data, the ACF, and the PACF
- With the plot of the data: look for trend (linear or otherwise) & determine whether to transform data
- Most software will use a maximum likelihood estimation to determine appropriate ARIMA parameters

ARIMA ANALYSIS TURNS UP EVERYWHERE

Time Series Analysis and Forecasting of Temperatures in the Sylhet
Division of Bangladesh

A.H. Nury^{1*}, M. Koch² and M.J.B. Alam³

¹ Department of Civil Engineering, Leading University, Sylhet-3100, Bangladesh

² Department of Geohydraulics and Engineering Hydrology, Kassel University, Kassel, Germany

³ Department of Civil and Environmental Engineering, Shahjalal University, Sylhet, Bangladesh

*e-mail: hasancee@yahoo.com

Application of an Autoregressive Integrated Moving Average Model for Predicting the Incidence of Hemorrhagic Fever with Renal Syndrome

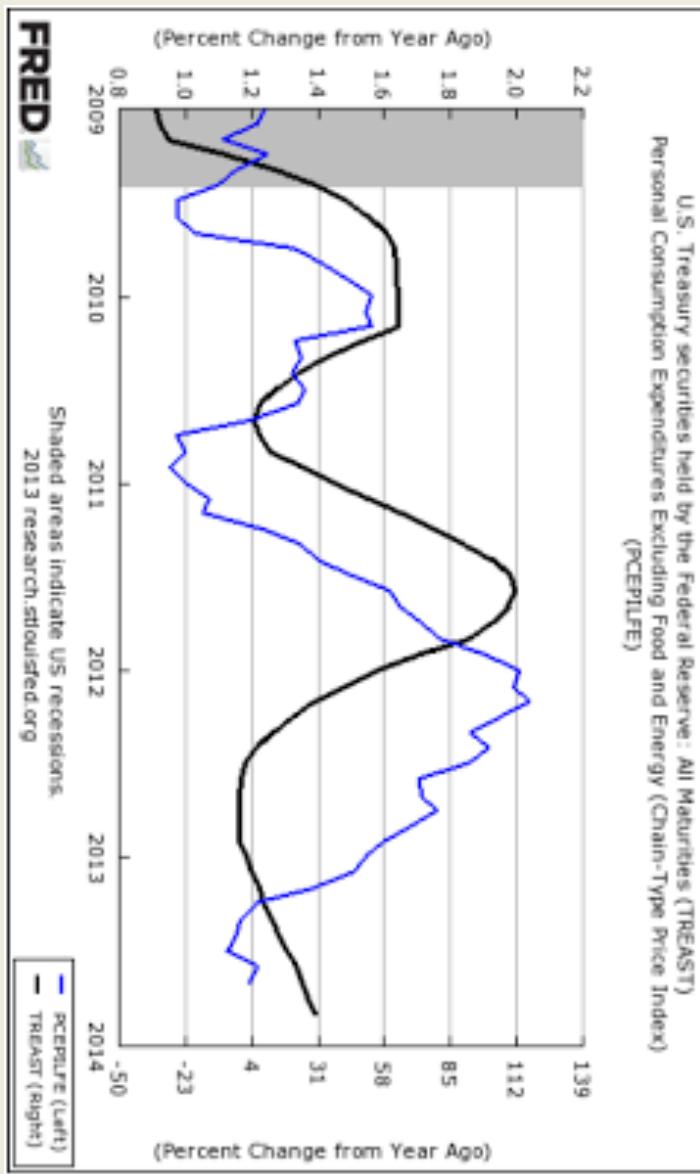
Qi Li*, Na-Na Guo, Zhan-Ying Han, Yan-Bo Zhang, Shun-Xiang Qi, Yong-Gang Xu, Ya-Mei Wei, Xu Han, and Ying-Ying Liu

Forecasting daily maximum surface ozone concentrations in Brunei Darussalam—an ARIMA modeling approach.

Kumar K¹, Yadav AK, Singh MP, Hassan H, Jain VK.

LEARN MORE...

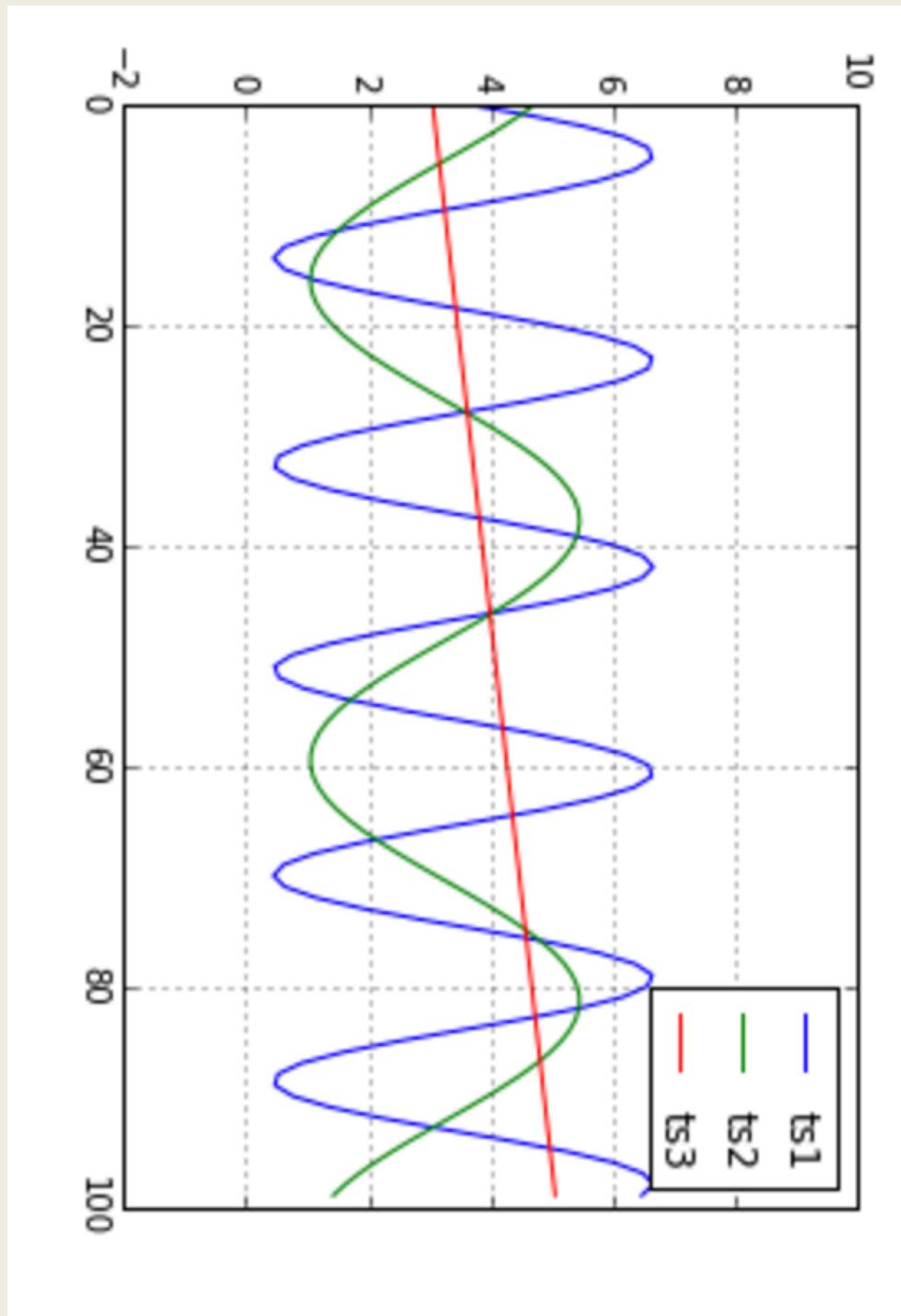
Vector auto-regression works similarly for cases of multivariate time series



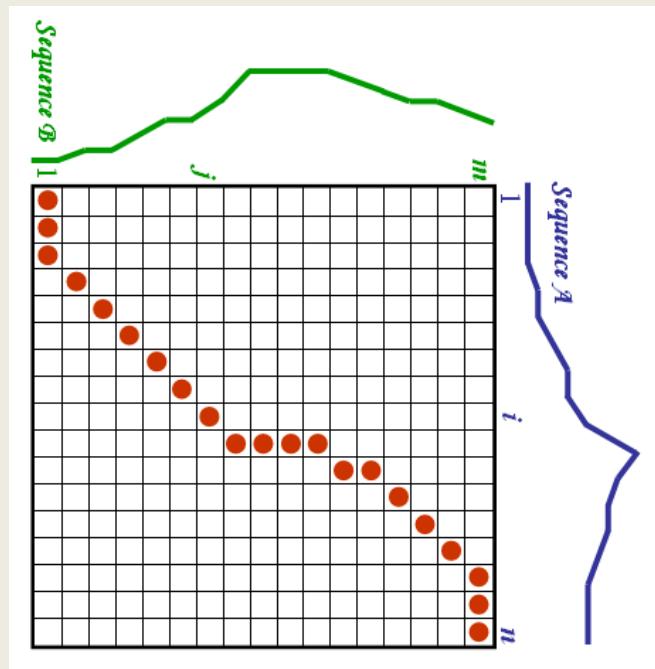
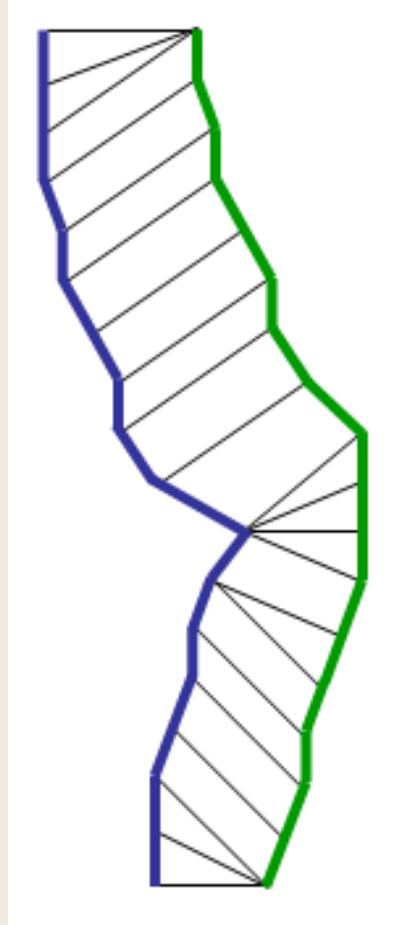
$$\begin{aligned}x_{t,1} &= \alpha_1 + \phi_{11}x_{t-1,1} + \phi_{12}x_{t-1,2} + \phi_{13}x_{t-1,3} + w_{t,1} \\x_{t,2} &= \alpha_2 + \phi_{21}x_{t-1,1} + \phi_{22}x_{t-1,2} + \phi_{23}x_{t-1,3} + w_{t,2} \\x_{t,3} &= \alpha_3 + \phi_{31}x_{t-1,1} + \phi_{32}x_{t-1,2} + \phi_{33}x_{t-1,3} + w_{t,3}\end{aligned}$$

CLUSTERING &
CLASSIFICATION
(YET ANOTHER ROUTE TO
PREDICTION)

NEED TO THINK CAREFULLY ABOUT
DISTANCE METRIC



DYNAMIC TIME WARPING



APPLICATIONS

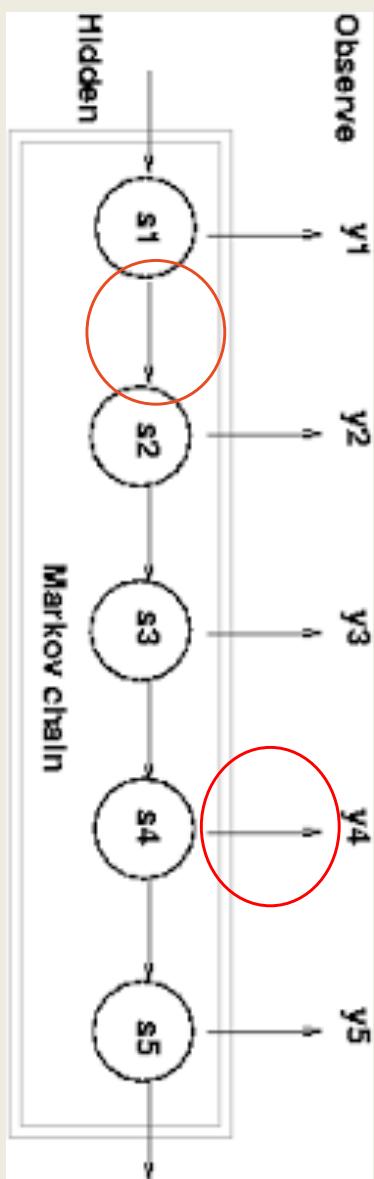
DTW-based clustering

DTW-based nearest neighbor classification

HIDDEN MARKOV MODELS

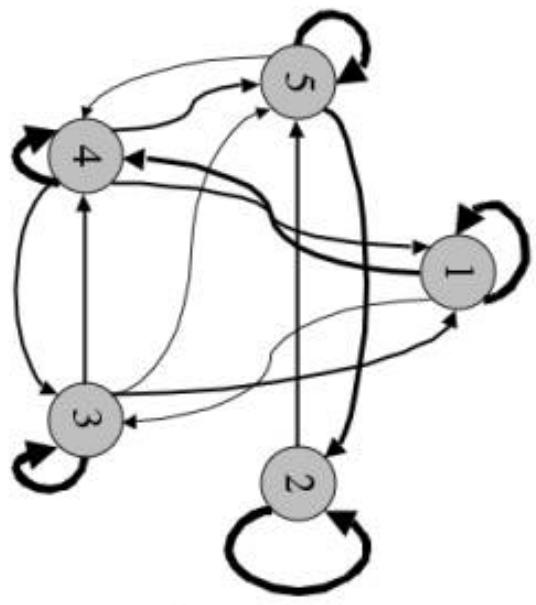
MULTI-STATE TIME-BASED SYSTEMS

Hidden Markov Models are another way of thinking about time series classification.



HOW IT WORKS

A transition matrix specifies the likelihood of transitioning between states.



Start State	Dest. State			
.977	.001	.021		.003
	.997			
.005	.992	.003	.0001	
.008	.004	.982	.006	
	.012	.0006	.988	

GET MORE PRACTICE



DataMarket has been acquired by Qlik®
Read more about this exciting development on DataMarket's blog.

Time Series Data Library

The Time Series Data Library (TSDL) was created by Rob Hyndman, Professor of Statistics at Monash University, Australia.

Results narrowed by:
x Time Series Data Library

You are looking at data from:

▼ Categories:

- Finance 35
- Computing 6
- Utilities 11
- Transport and tourism 13
- Tree-rings 35
- Micro-Economic 22
- Sales 37
- Production 48
- Labour market 24
- Macro-Economic 56
- Miscellaneous 6
- Crime 4
- Chemistry 1
- Hydrology 120

Time Series Data Library
The Time Series Data Library (TSDL) was created by Rob Hyndman, Professor of Statistics at Monash University, Australia.

Search results: 564

Monthly milk production: pounds per cow. Jan 62 – Dec 75
Monthly time series (Jan 1962–Dec 1975). It was last modified on 1 Feb 2014 at 19:52. Categorized as Agriculture.

Annual barley yields per acre in England & Wales 1884 – 1939
Yearly time series (1884–1939). It was last modified on 1 Feb 2014 at 19:52. Categorized as Agriculture.

Monthly total number of pigs slaughtered in Victoria. Jan 1980 – August 1995
Monthly time series (Jan 1980–Aug 1995). It was last modified on 1 Feb 2014 at 19:52. Categorized as Agriculture.



← → ⌂ www.cs.ucr.edu/~eamonn/time_series_data/

UCR Time Series

Classification
Archive