



Time Series Analysis For Everyone

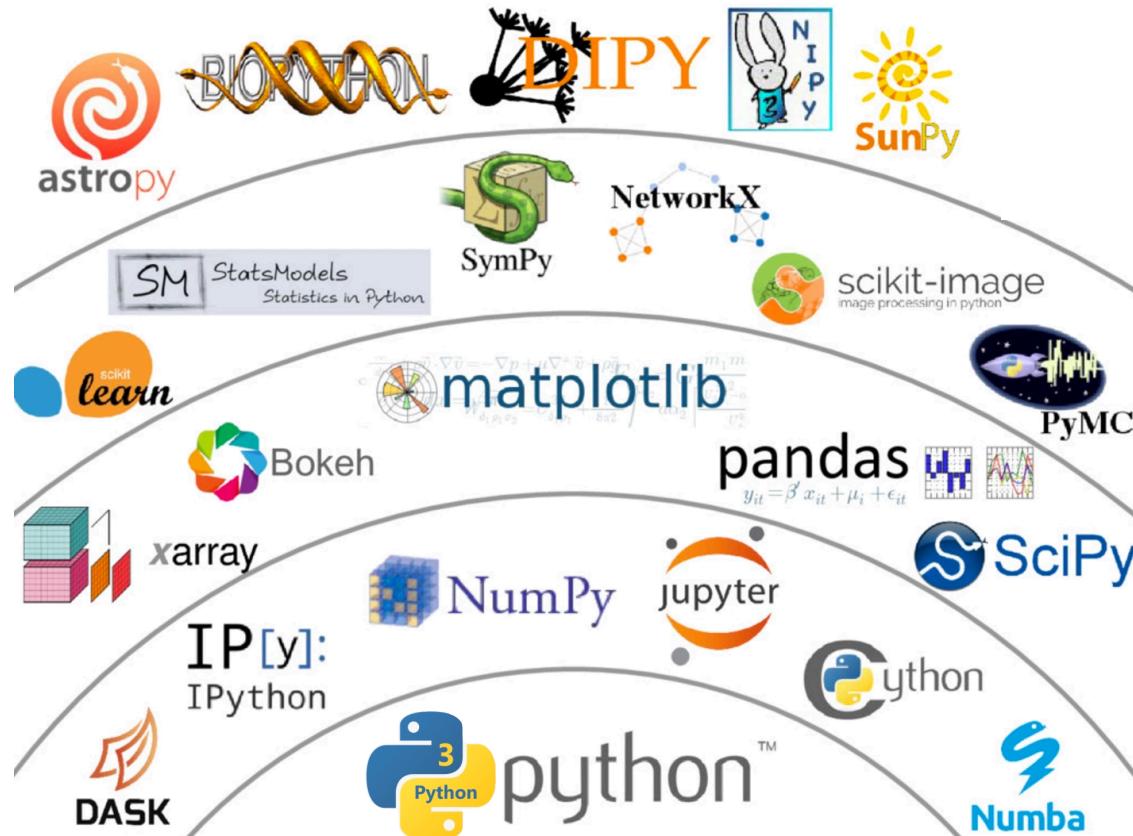
Statistical, and Machine Learning approaches with Python

Bruno Gonçalves

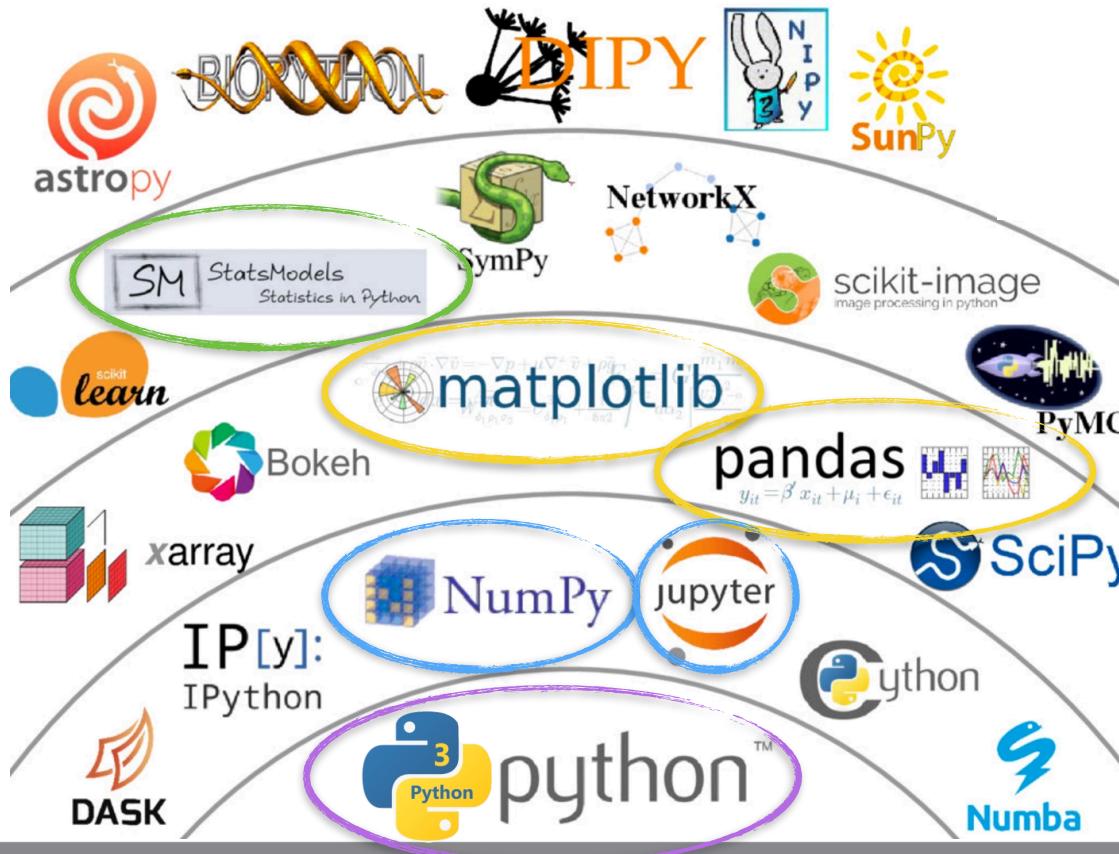
www.data4sci.com/newsletter

https://github.com/DataForScience/Timeseries_LL

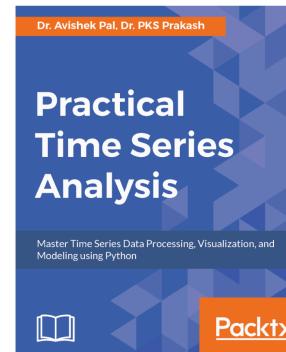
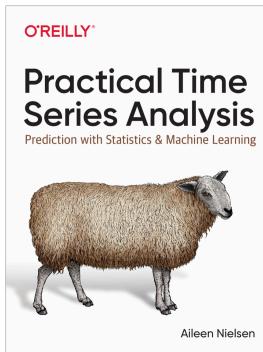
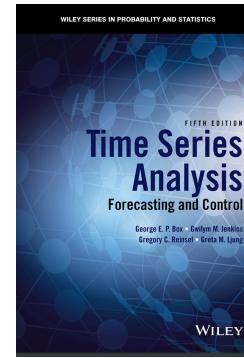
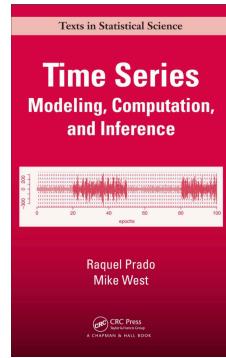
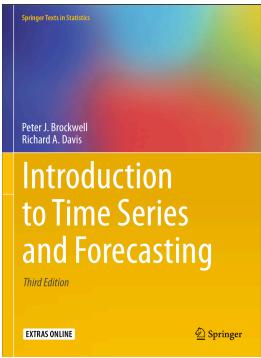
Python Ecosystem



Python Ecosystem



References





Lesson 1 - Pandas for Time Series

1.1 DataFrames and Series

1.2 Subsetting

1.3 Time Series

1.4 DataFrame Manipulations

1.5 Pivot Tables

1.6 Merge and Join



Lesson 1 - Pandas for Time Series

1.1 DataFrames and Series

1.2 Subsetting

1.3 Time Series

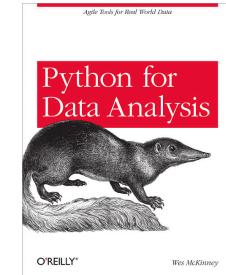
1.4 DataFrame Manipulations

1.5 Pivot Tables

1.6 Merge and Join

pandas

- Created by [Wes McKinney](#) in 2008 while working for [AQR Capital Management](#), currently working at [Two Sigma](#) and [Ursa Labs](#)
- Pandas is specifically designed to handle time series and data frames
- High performance, flexible tool for quantitative analysis on financial data
- The functionality is built on top [numpy](#) and [matplotlib](#) (see next lecture)
- The fundamental data structures are [Series](#) (1-dimensional) and [DataFrame](#) (2-dimensional).
- Each column of a [DataFrame](#) can have a different [dtype](#) but all the elements in a column (or [Series](#)) must be of the same [dtype](#)
- Conventionally imported as
`import pandas as pd`



Series and Data Frames

- A `Series` is conceptually equivalent to a numpy array with some extra information (column and row names, etc)
 - A `DataFrame` can be thought of as the union of several `Series`, with names associated.
 - Similarly, you can think of a `DataFrame` as an Excel sheet and of a `Series` as an individual column
 - A minimal `DataFrame` implementation would be a dict where each element is a list

Series and Data Frames

Series

	id
0	23
1	42
2	12
3	86

+

Series

	Name
0	“Bob”
1	“Karen”
2	“Kate”
3	“Bill”

=

DataFrame

	id	Name
0	23	“Bob”
1	42	“Karen”
2	12	“Kate”
3	86	“Bill”

DataFrame Fields

- `DataFrame`s contain a great deal of extra information in addition to just the data values. In particular, they include:
 - `columns` - column names
 - `index` - row names
 - `dtypes` - dtype associated with each column
 - `shape` - number of rows and columns
 - `ndim` - the number of dimensions
- `DataFrame`s make it easy to add new columns.
- `DataFrame` elements can be accessed and modified in various ways

DataFrame Exploration

- We get information about the contents of a DataFrame using several simple methods:
 - `head()` - display the top 5 rows
 - `tail()` - display the bottom 5 rows
 - `info()` - Print a concise summary of a DataFrame
 - `describe()` - Generate descriptive statistics that summarize the data



Lesson 1 - Pandas for Time Series

1.1 DataFrames and Series

1.2 Subsetting

1.3 Time Series

1.4 DataFrame Manipulations

1.5 Pivot Tables

1.6 Merge and Join

Indexing and Slicing

- pandas supports various ways of indexing the contents of a DataFrame
- [`<column name>`] - select a given column by it's name. column names can also be used as field names
- `.loc[<row name>]` - select a given row by it's (Index) name
- `.iloc[<position>]` - select a given row by it's position in the Index, starting at 0
- `.loc[<row name>, <column name>]` - select an individual element by row and column name
- `.iloc[<row position>, <column position>]` - select an individual element by row and column position (starting at 0)
- `.iloc` also supports ranges and slices similarly to `python` lists or `numpy` arrays

Adding columns and rows

- New columns can be `added` (in place) by simply `assigning` to them
- columns behave like numpy arrays and support mathematical operations
- New rows can be added in two ways:
 - `append()` - Append rows to the end of a DataFrame, returning a new object
 - `pd.concat()` - Concatenate two or more DataFrame objects, adding new rows (`axis=0`) or columns (`axis=1`). A new object is returned

Deleting columns and rows

- Rows and columns can be deleted using the `drop()` function.
- The `drop()` function takes the row/column name as an argument
- The axis argument indicates whether we mean `rows` or `columns`:
- `axis=0` - rows (the default)
- `axis=1` - columns
- In both cases, a new object is returned

Importing and Exporting Data

- pandas has powerful methods to read and write data from multiple sources
 - `pd.read_csv()` - Read a comma-separated values file
 - `sep=''` - Define the separator to use. ',' is the default
 - `header=0` - Row number to use as column names
 - `pd.read_excel()` - Read an Excel file
 - `sheet_name` - The sheet name to load
 - `pd.read_html()` - Read tabular data from a URL (or local html file)
 - `pd.read_pickle()` - Read a Pickle file

Importing and Exporting Data

- Each of these functions accepts a large number of options and parameters controlling its behavior. Use `help(<function name>)` to explore further.
- Each `read_*`() function has a complementary `to_*`() function to write out a `DataFrame` to disk. The `to_*`() functions are members of the `DataFrame` object



Lesson 1 - Pandas for Time Series

1.1 DataFrames and Series

1.2 Subsetting

1.3 Time Series

1.4 DataFrame Manipulations

1.5 Pivot Tables

1.6 Merge and Join

Time series

- pandas was originally developed to handle financial data
- Temporal sequences (time series) are a common type of data encountered in financial applications, so, naturally, pandas has good support for the most common time series operations.
- `pd.to_datetime` - converts a Series or value into a date time timestamp.
 - Format is inferred by looking at the entire Series data
 - Format can also be manually specified using specific arguments:
 - `format` - detailed string specifying the full format
 - `dayfirst=True` - parse 10/11/12 as Nov 10, 2012
 - `yearfirst=True` - parse 10/11/12 as Nov 12, 2010

Time series

- `pd.to_timedelta` - converts a Series into an absolute difference in time
- Dates can also be parse automatically at read time by passing a list of the columns containing dates to the `parse_dates` argument of `pd.read_csv`

Time series

- Sequences of dates/times can be created using `pd.date_range()`, similar to `np.arange()`
 - `start/end` - specify the limits
 - `freq` - specify frequency (step)
 - B - business day frequency
 - D - calendar day frequency
 - W - weekly frequency
 - M - month end frequency
 - H - hourly frequency
 - T - minutely frequency
 - S - secondly frequency

Time series

- By setting the index to be a date time, we turn the `DataFrame` into a `Timeseries`.
- Pandas has several methods that are specialized to this case
- `index.day/index.month/index.year` - return the day, month and year of the time series index value



Lesson 1 - Pandas for Time Series

1.1 DataFrames and Series

1.2 Subsetting

1.3 Time Series

1.4 DataFrame Manipulations

1.5 Pivot Tables

1.6 Merge and Join

DataFrame manipulations

- `pd.to_datetime()` is just one of several methods that allow us to manipulate and transform the format and contents of `Series` and `DataFrame`s.
- other methods include:
 - `map()` - Map values of `Series` according to input correspondence.
 - `func` - `function`, `dict` or `Series` to use for mapping
 - `transform()` - Transform each column/row of the `DataFrame` using a function. Output must have the same shape as the original
 - `axis = 0` - apply function to columns
 - `axis = 1` - apply function to rows

DataFrame manipulations

- `apply()` - Apply a function along an axis of the DataFrame
 - Similar to transform but without the limitation of preserving the shape

groupby

- Sometimes we need to calculate statistics for subsets of our data
- `groupby()` allows us to group data based on a dict or function
- `groupby()` returns a GroupBy object that supports several aggregations functions, including:
 - `max()/min()/mean()/median()`
 - `transform()/apply()`
 - `sum()/cumsum()`
 - `prod()/cumprod()`
 - `quantile()`
- Each of these functions is applied to the contents of each group



Lesson 1 - Pandas for Time Series

1.1 DataFrames and Series

1.2 Subsetting

1.3 Time Series

1.4 DataFrame Manipulations

1.5 Pivot Tables

1.6 Merge and Join

Pivot Tables

- Pandas provides an extremely flexible pivot table implementation
- `pd.pivot_table()` - Creates a spreadsheet-style pivot table as a `DataFrame`.
- Three important arguments:
 - `values` - column to aggregate
 - `index` - Keys to group by on the pivot table index. If an array is passed, it is being used as the same manner as column values.
 - `columns` - Keys to group by on the pivot table column.
 - `aggfunc` - Function to use for aggregation. Defaults to `np.mean()`
- `index`, `columns` and `aggfunc` can also be lists of keys or functions to use.



Lesson 1 - Pandas for Time Series

1.1 DataFrames and Series

1.2 Subsetting

1.3 Time Series

1.4 DataFrame Manipulations

1.5 Pivot Tables

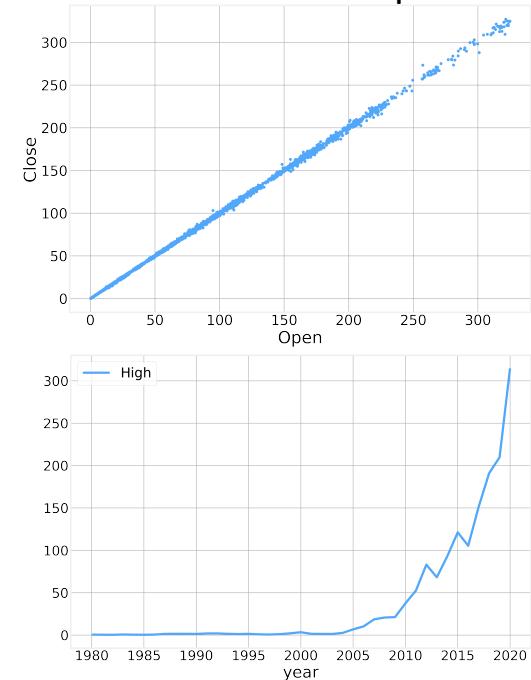
1.6 Merge and Join

merge/join

- `merge()` and `join()` allows us to perform database-style join operation by columns or indexes (rows)
- Some of the most important arguments are common to both methods:
 - `on` - Column(s) to use for joining, otherwise join on index. Can pass an array as the join key if not already contained in the calling DataFrame. Like an Excel VLOOKUP operation
 - `how` - Type of join to perform: `{'left', 'right', 'outer', 'inner'}`
- `merge()` is more sophisticated and flexible. It also allows us to specify:
 - `left_on/right_on` - Field names to join on for each DataFrame
 - `left_index/right_index` - Whether or not to use the left/right index as

plotting

- Finally, pandas also provides a simple interface for basic plotting through the `plot()` function
- By specifying some basic parameters the variables plotted and even the kind of plot can be easily modified:
 - x/y -column name to use for the x/y axis
 - kind - type of plot
 - ‘line’ - line plot (default)
 - ‘bar’/‘barh’ - vertical/horizontal bar plot
 - ‘hist’ - histogram
 - ‘box’ - boxplot
 - ‘pie’ - pie plot
 - ‘scatter’ - scatter plot





Code - Pandas Review

https://github.com/DataForScience/Timeseries_LL



Lesson 2 Visualizing Time Series

2.1 Data Representation

2.2 Gross Domestic Product

2.3 Influenza Mortality

2.4 Sun Activity

2.5 Dow Jones Industrial Average

2.6 Airline Passengers



Lesson 2 Visualizing Time Series

2.1 Data Representation

2.2 Gross Domestic Product

2.3 Influenza Mortality

2.4 Sun Activity

2.5 Dow Jones Industrial Average

2.6 Airline Passengers

Time series

- A set of values measured sequentially in time
- Values are typically (but not always) measured at equal intervals, $x_1, x_2, x_3, x_4, \dots$
- Values can be:
 - continuous
 - discrete or symbolic (words).

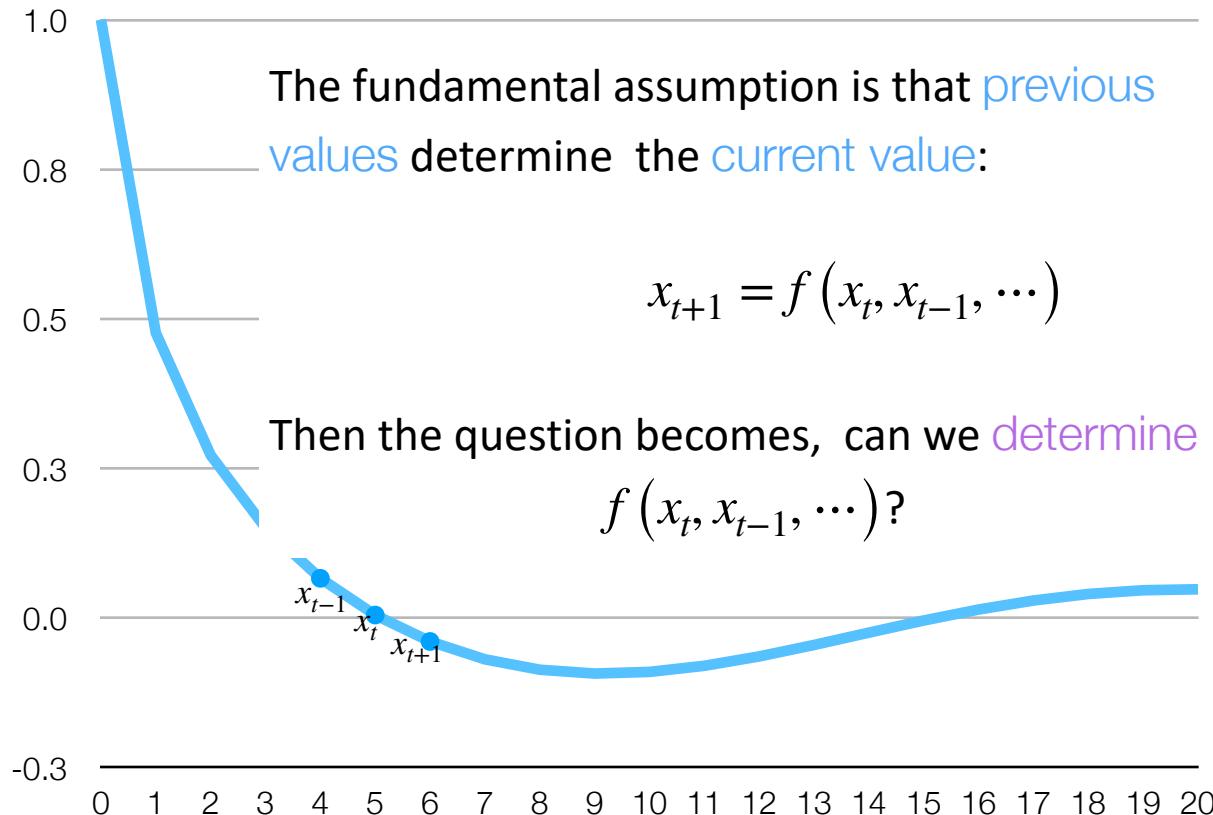
Time series

- Associated with empirical observation of time varying phenomena:
 - Stock market prices (day, hour, minute, tick, etc...)
 - Temperatures (day, minute, second, etc...)
 - Number of patients (week, month, etc...)
 - GDP (quarter, year, etc...)
- Forecasting requires predicting future values based on past behavior

Mathematical Conventions

- The value of the time series at time t is given by x_t
- The values at a given lag l are given by x_{t-l}
- The mean of the overall signal is μ and the corresponding running value is μ_t^w
- The variance of the overall signal is σ and the corresponding running value is σ_t^w
- Running values are calculated over a window of width w

Time series analysis





Lesson 2 Visualizing Time Series

2.1 Data Representation

2.2 Gross Domestic Product

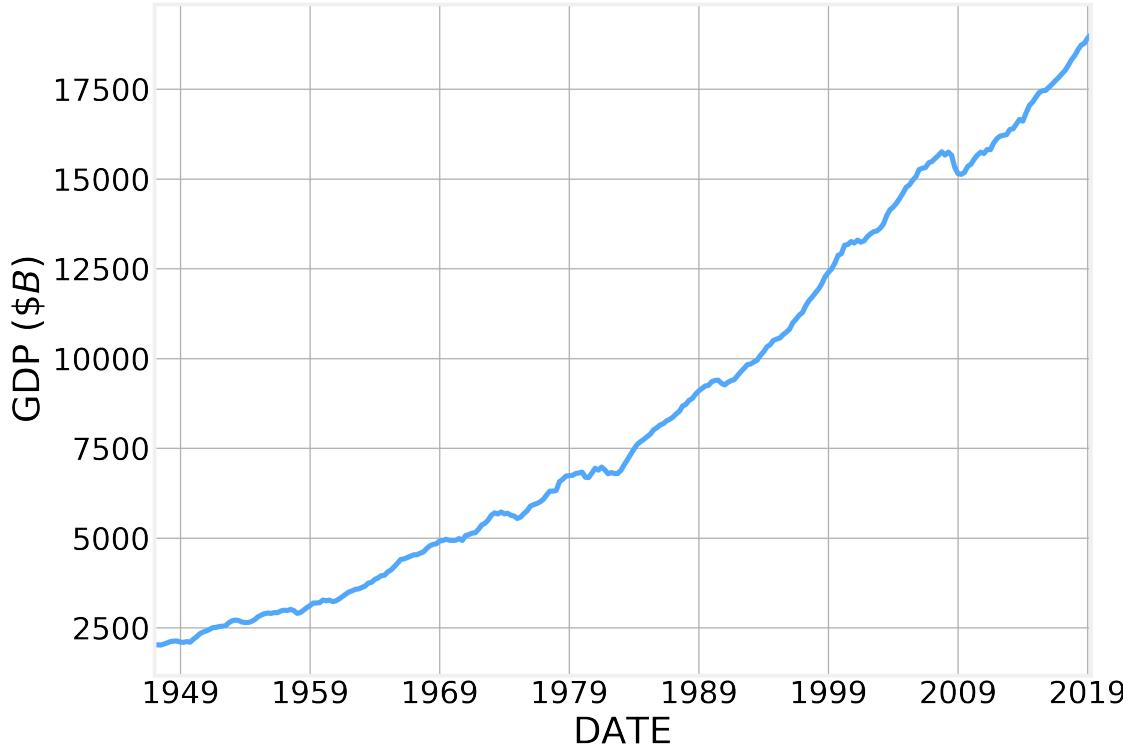
2.3 Influenza Mortality

2.4 Sun Activity

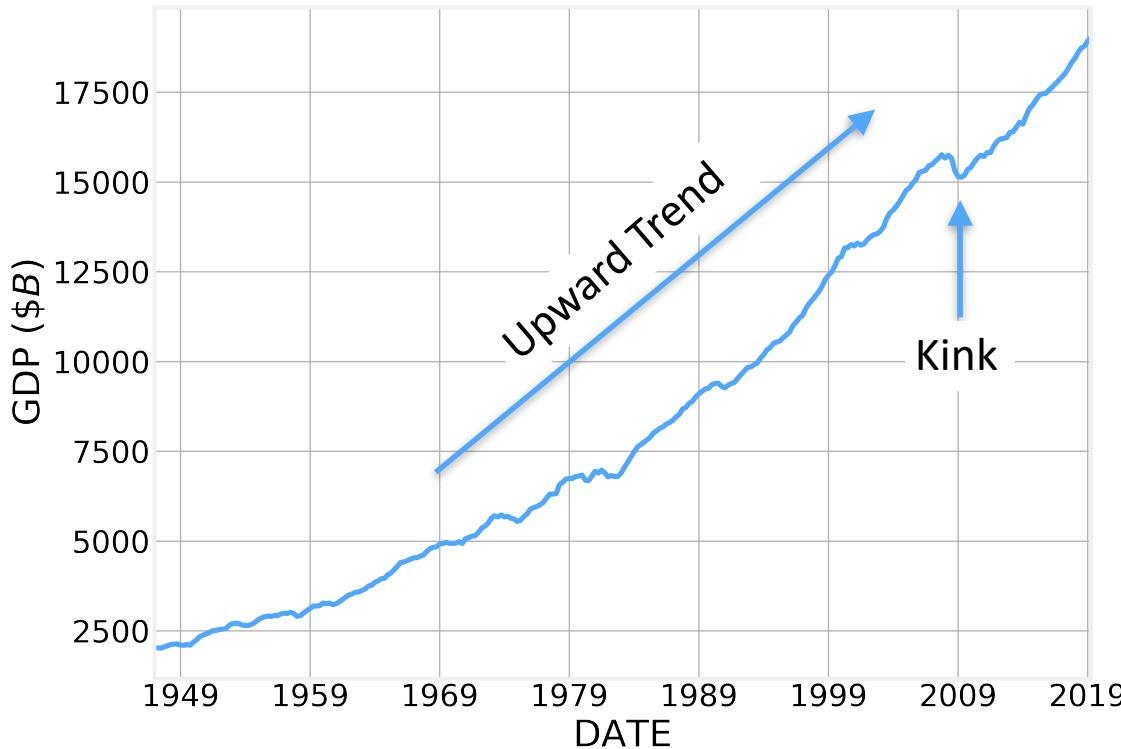
2.5 Dow Jones Industrial Average

2.6 Airline Passengers

Economics - GDP



Economics - GDP





Lesson 2 Visualizing Time Series

2.1 Data Representation

2.2 Gross Domestic Product

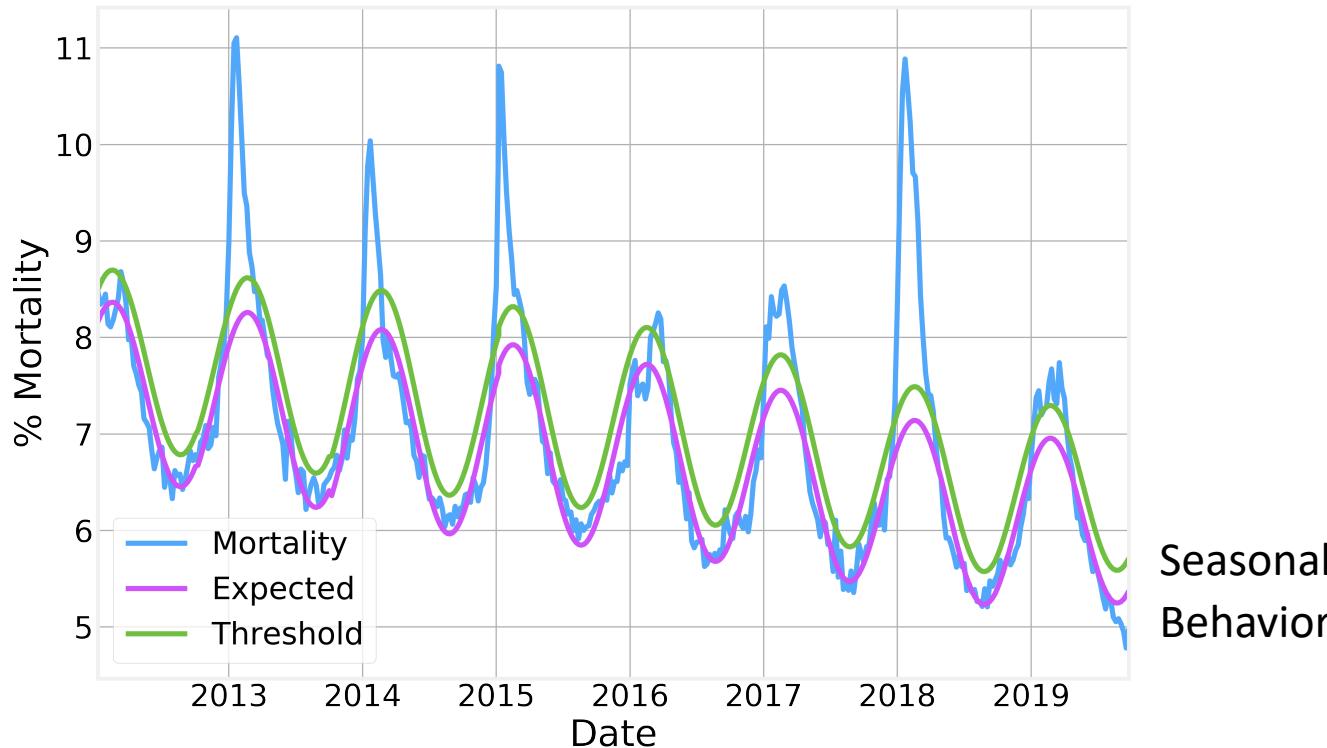
2.3 Influenza Mortality

2.4 Sun Activity

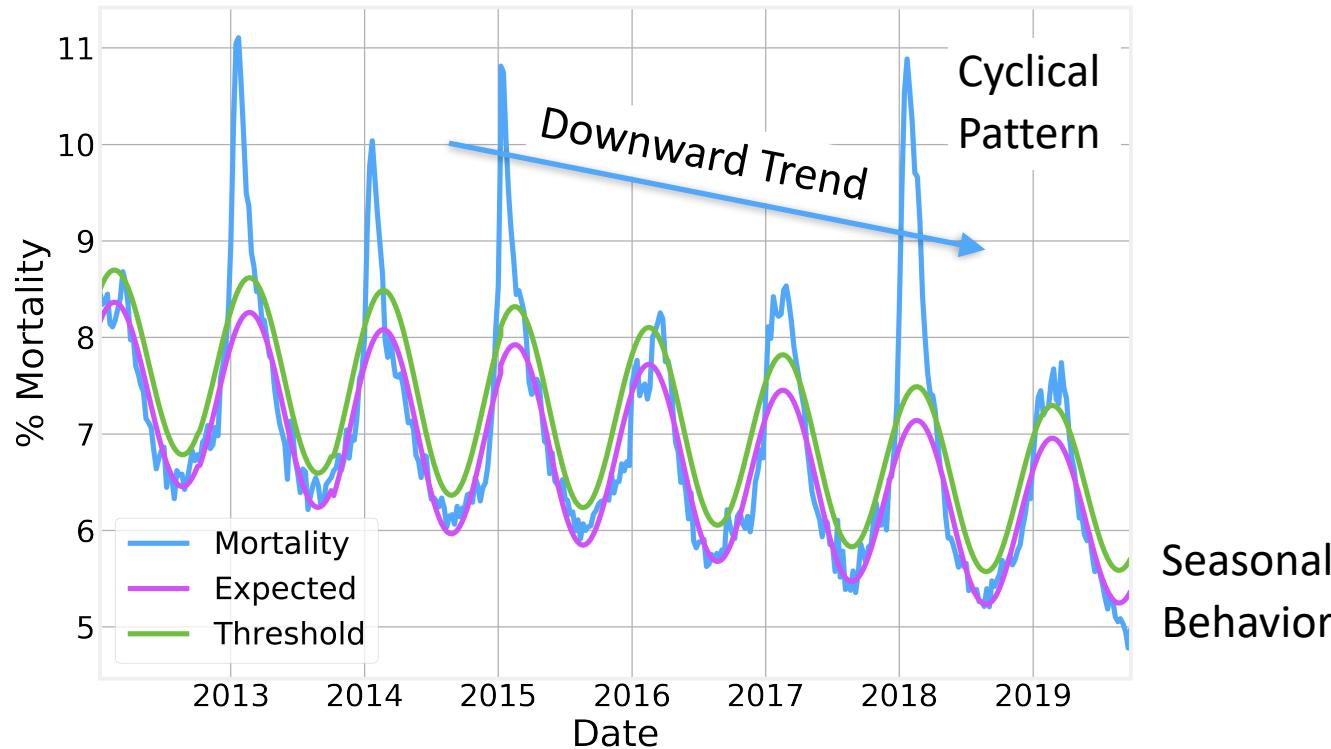
2.5 Dow Jones Industrial Average

2.6 Airline Passengers

Epidemiology - Influenza



Epidemiology - Influenza





Lesson 2 Visualizing Time Series

2.1 Data Representation

2.2 Gross Domestic Product

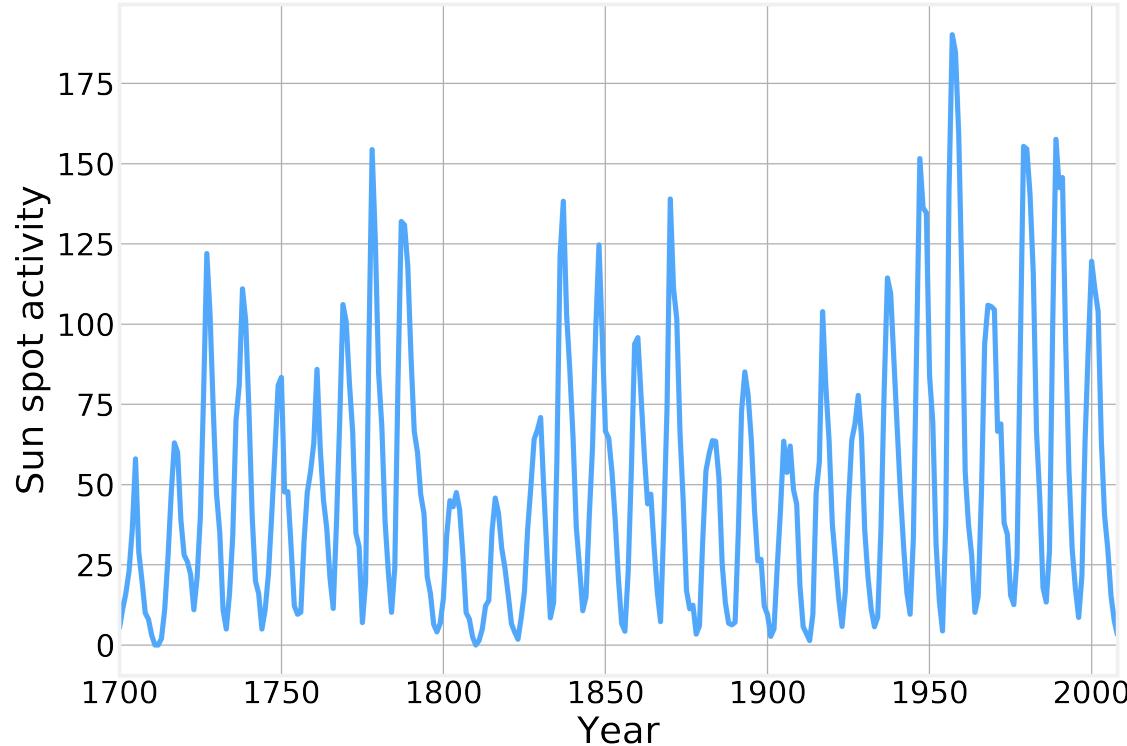
2.3 Influenza Mortality

2.4 Sun Activity

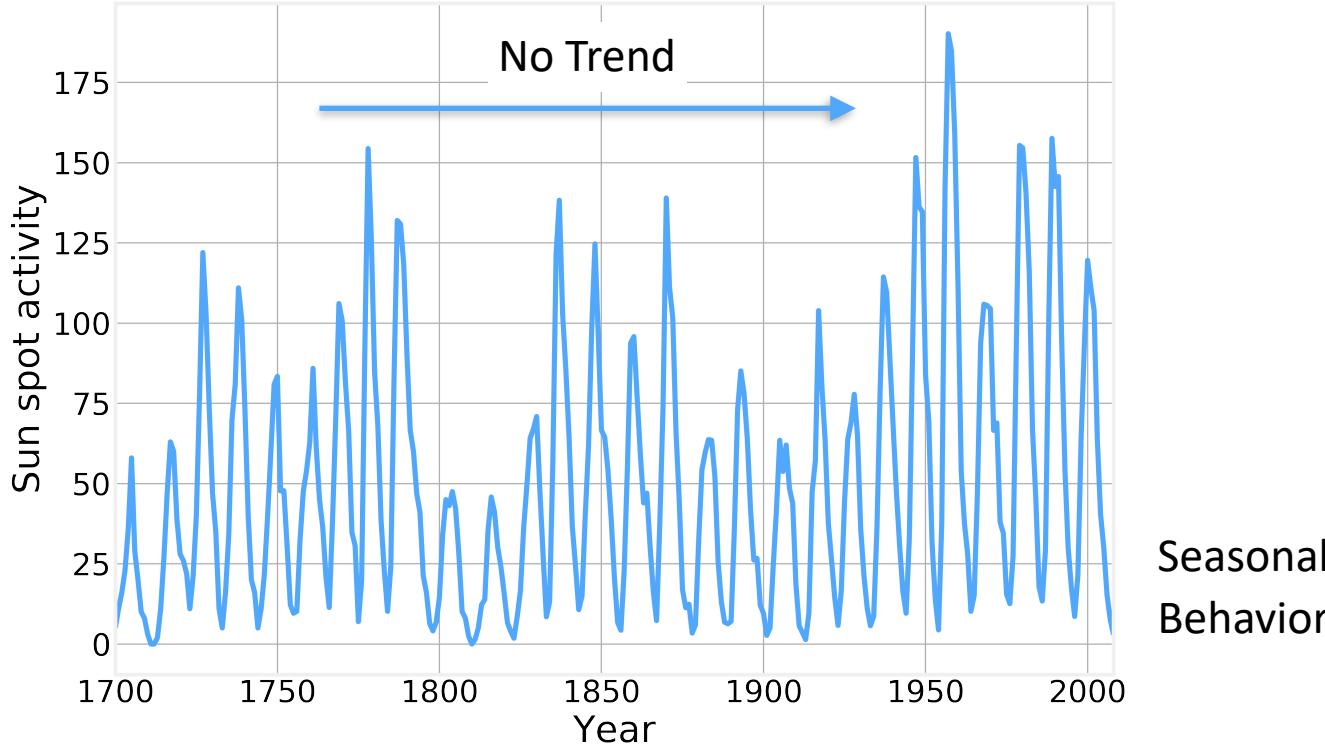
2.5 Dow Jones Industrial Average

2.6 Airline Passengers

Astronomy - Sunspot activity



Astronomy - Sunspot activity





Lesson 2 Visualizing Time Series

2.1 Data Representation

2.2 Gross Domestic Product

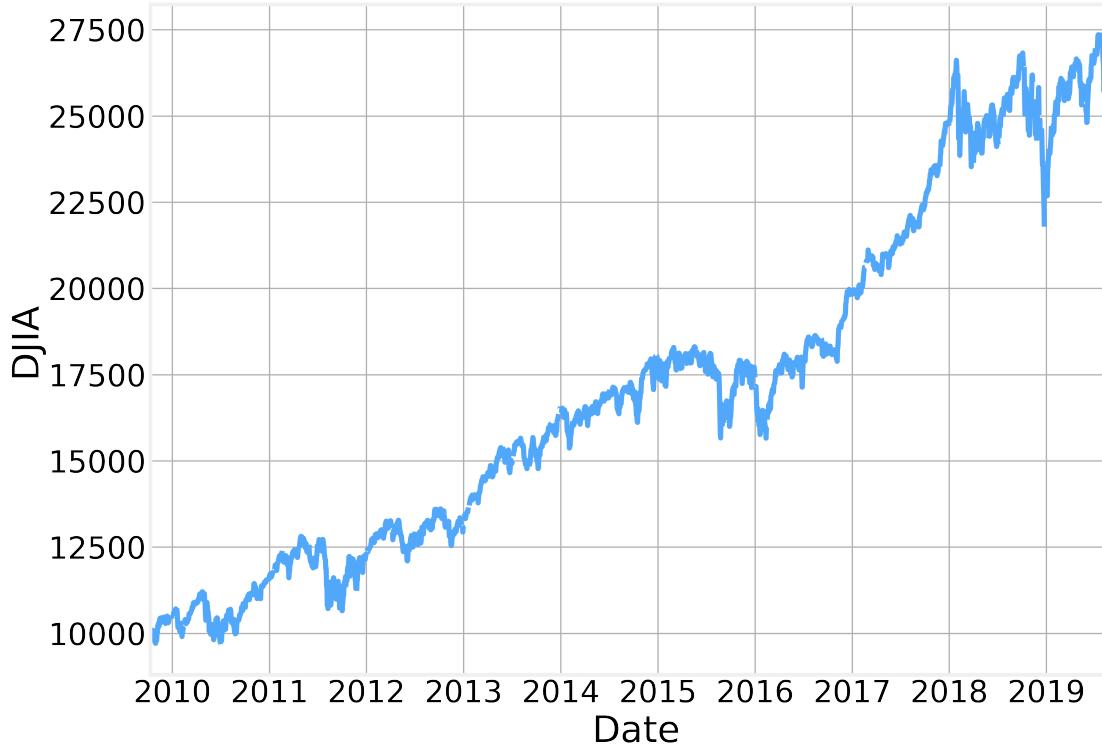
2.3 Influenza Mortality

2.4 Sun Activity

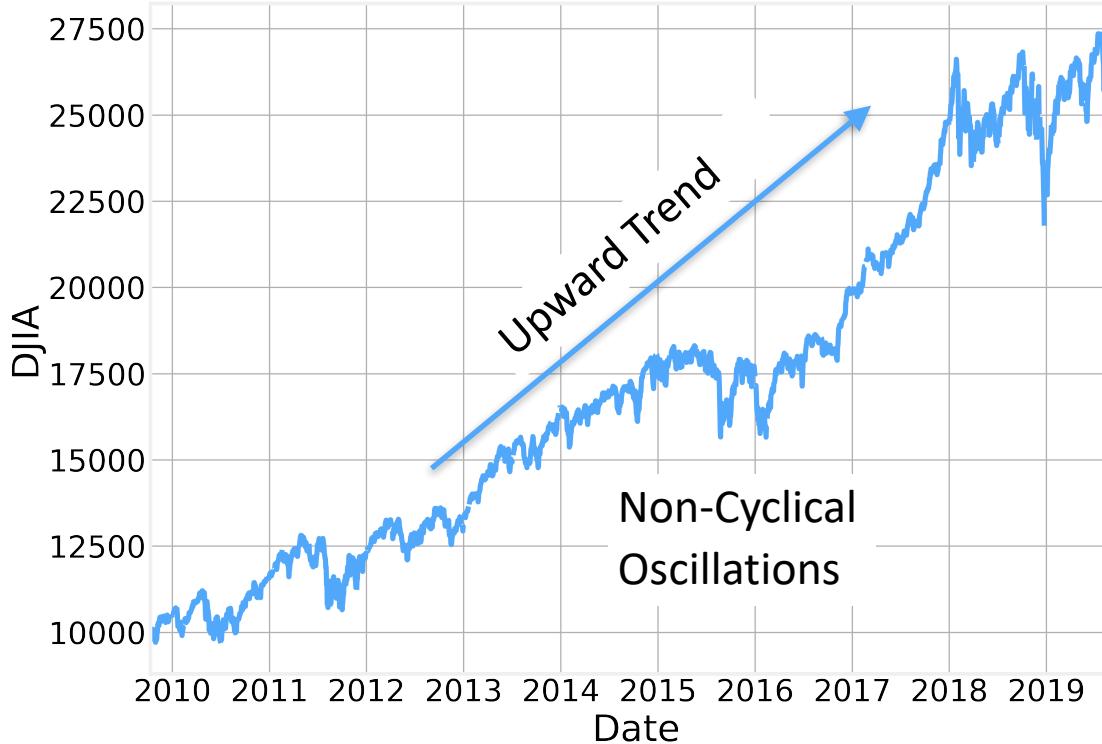
2.5 Dow Jones Industrial Average

2.6 Airline Passengers

Stock Market - DJIA



Stock Market - DJIA





Lesson 2 Visualizing Time Series

2.1 Data Representation

2.2 Gross Domestic Product

2.3 Influenza Mortality

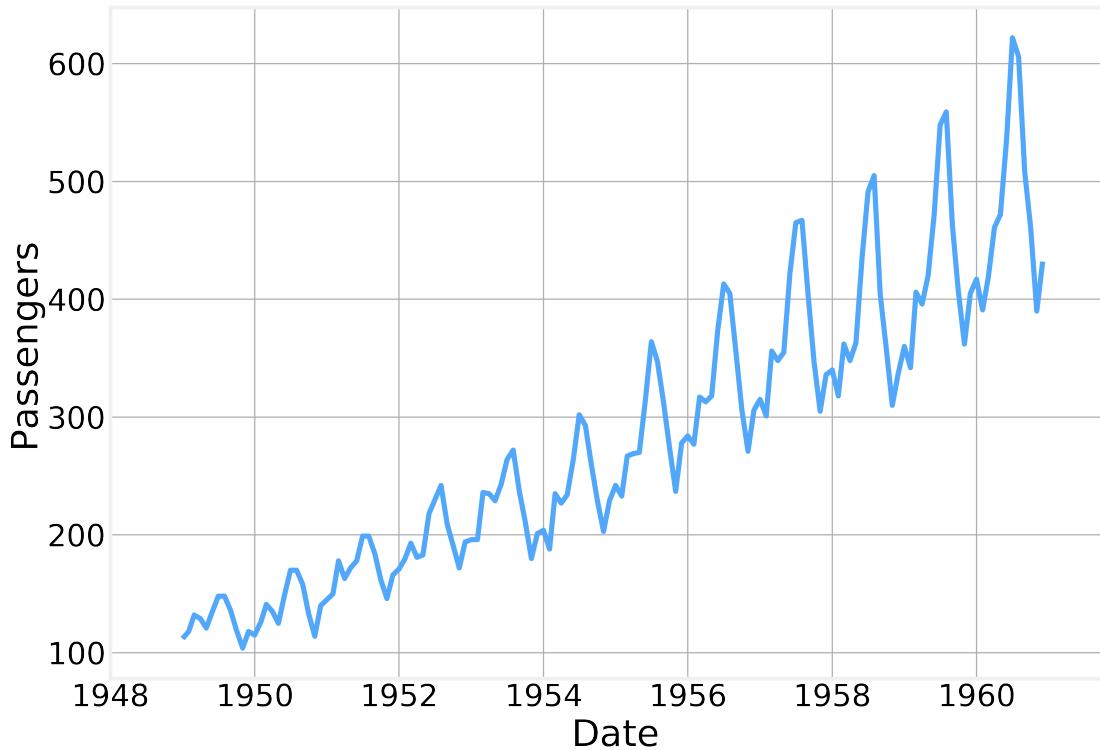
2.4 Sun Activity

2.5 Dow Jones Industrial Average

2.6 Airline Passengers

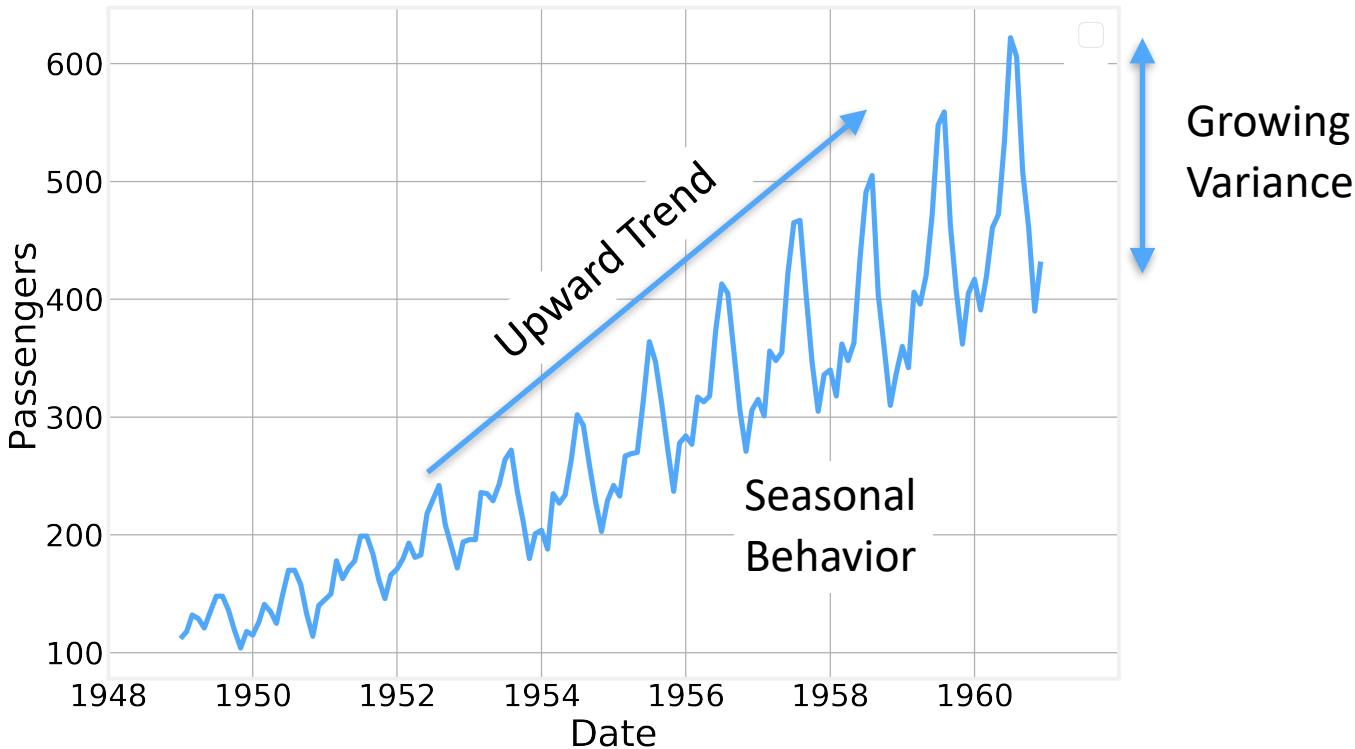
Logistics - Airline Passengers

<https://www.kaggle.com/chirag19/air-passengers>



Logistics - Airline Passengers

<https://www.kaggle.com/chirag19/air-passengers>





Code - Data Exploration

https://github.com/DataForScience/Timeseries_LL



Lesson 3 Stationarity and Trending Behavior

3.1 Non-stationarity

3.2 Trend

3.3 Demo 1

3.4 Seasonality

3.5 Time Series Decomposition

3.6 Demo 2



Lesson 3 Stationarity and Trending Behavior

3.1 Non-stationarity

3.2 Trend

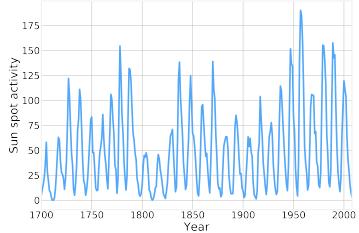
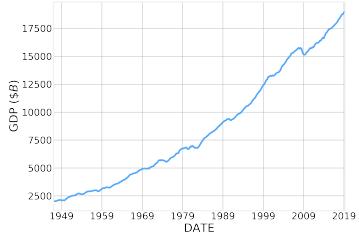
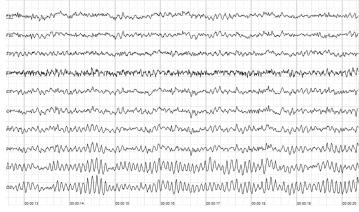
3.3 Demo 1

3.4 Seasonality

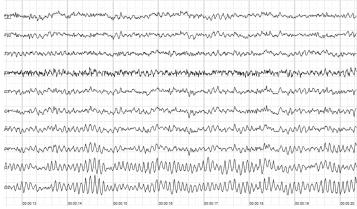
3.5 Time Series Decomposition

3.6 Demo 2

Three fundamental behaviors

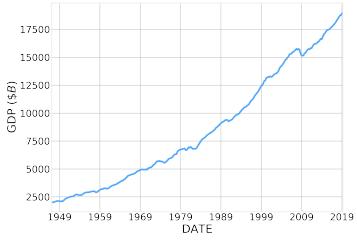


Three fundamental behaviors



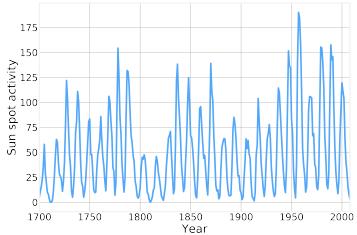
Stationarity

$$\langle x_t \rangle \approx \text{constant}$$



Trend

$$\langle x_t \rangle \approx ct$$



Seasonality

$$x_{t+T} \approx x_t$$

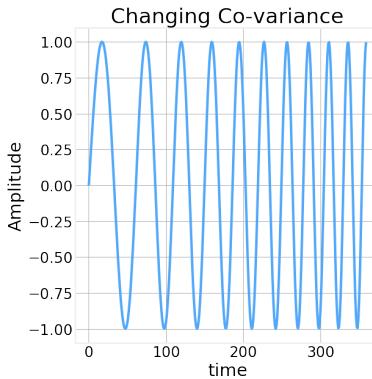
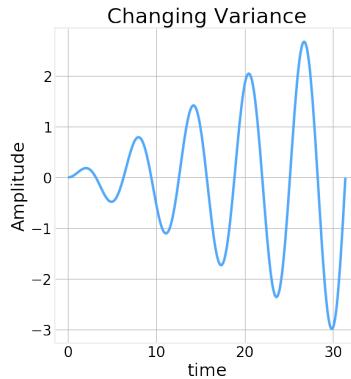
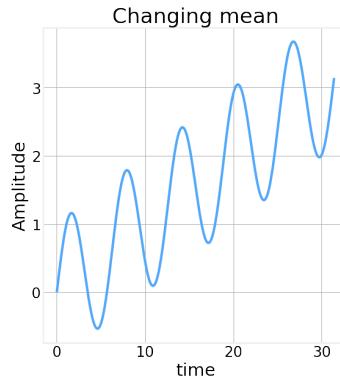
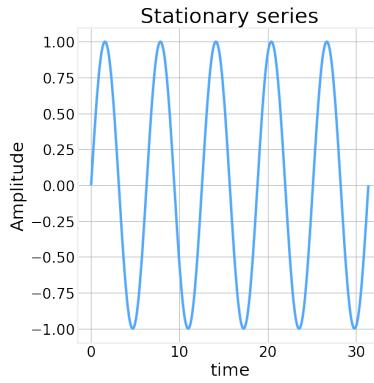
Stationarity

- A time series is said to be stationary if its basic statistical properties are independent of time
- In particular:
 - Mean - Average value stays constant
 - Variance - The width of the curve is bounded
 - Covariance - Correlation between points is independent of time
- Stationary processes are easier to analyze
- Many time series analysis algorithms assume the time series to be

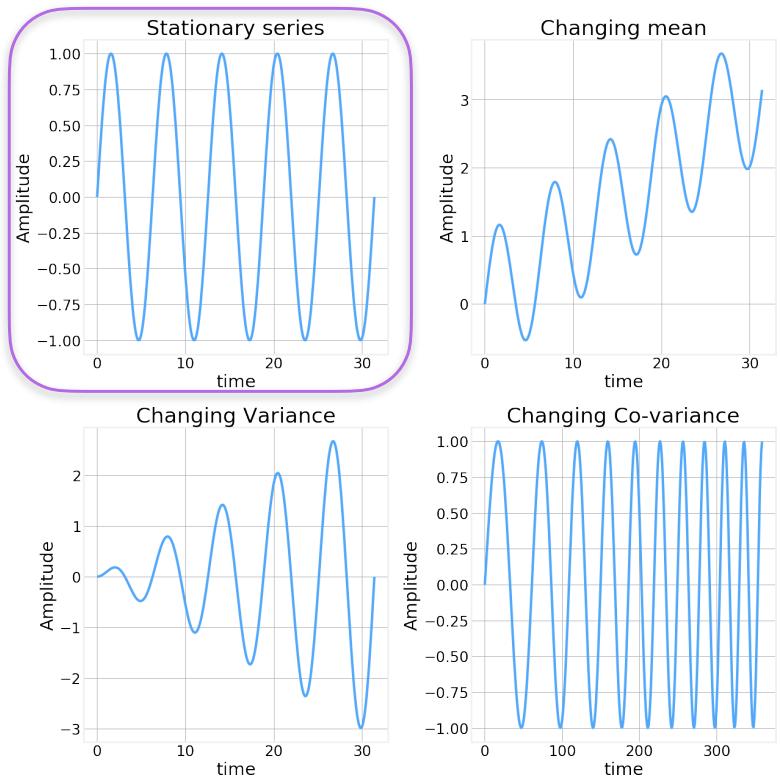
Stationarity

- Several rigorous tests for stationarity have been developed such as the (Augmented) Dickey-Fuller and Hurst Exponent
- Typically, the first step of any analysis is to transform the series to make it stationary

Stationarity



Stationarity





Lesson 3 Stationarity and Trending Behavior

3.1 Non-stationarity

3.2 Trend

3.3 Demo 1

3.4 Seasonality

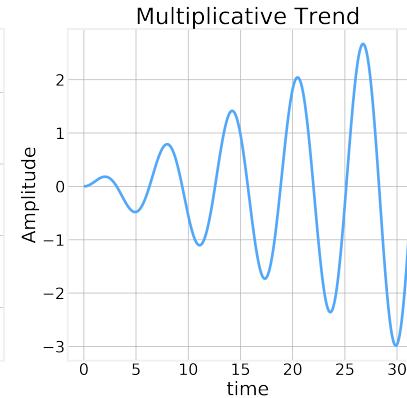
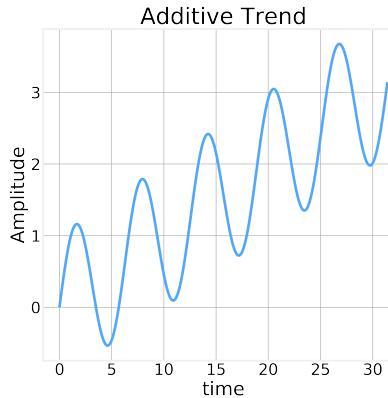
3.5 Time Series Decomposition

3.6 Demo 2

Trend

- Many time series have a clear trend or tendency:
 - Stock market indices tend to go up over time
 - Number of cases of preventable diseases tends to go down over time
 - etc
- Trends can be additive or multiplicative:

$$x = \frac{t}{10} + \sin(t)$$

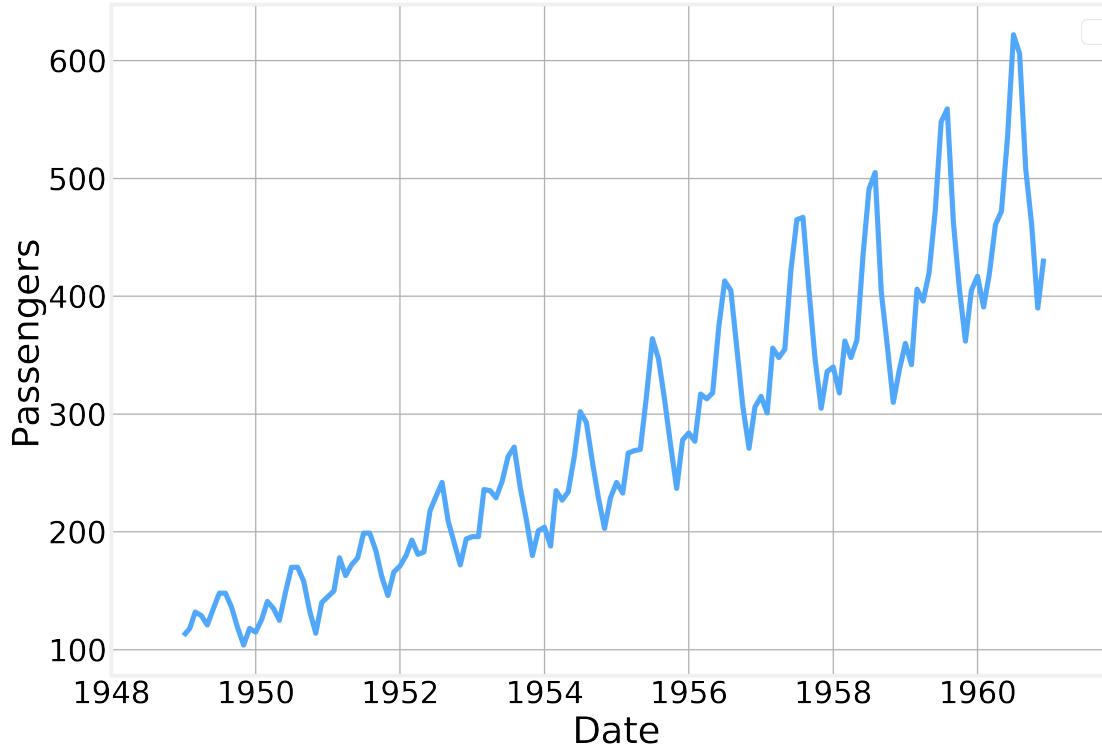


$$x = \frac{t}{10} \sin(t)$$

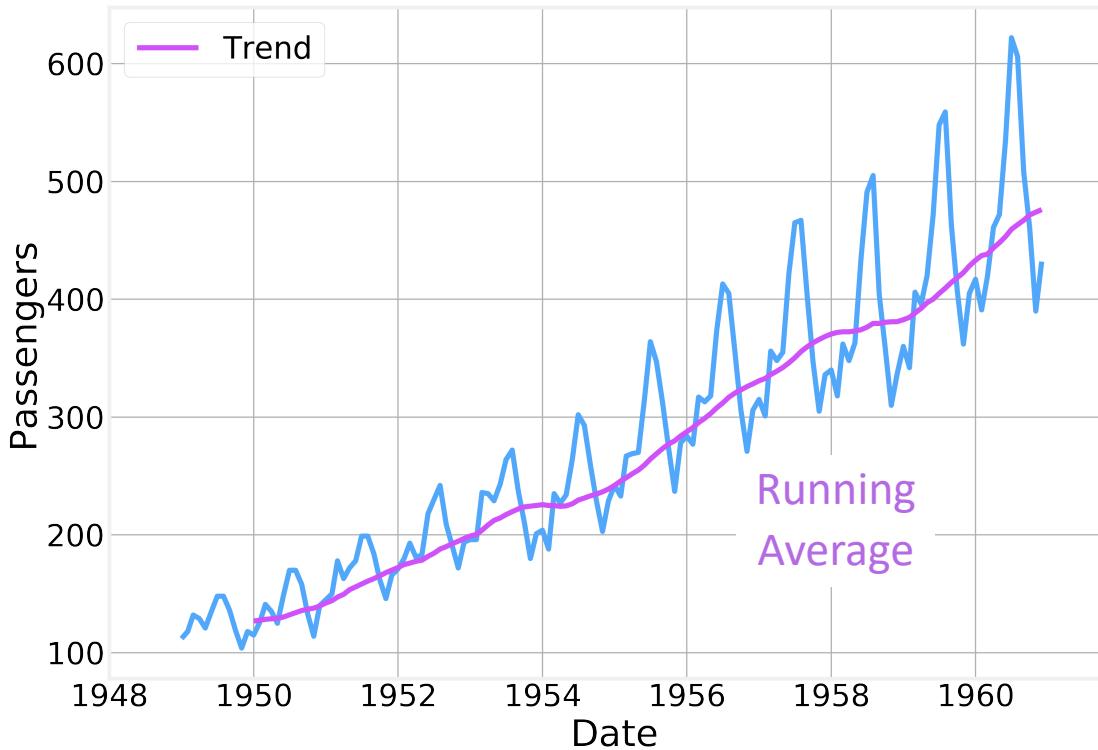
Trend

- Trends can be removed by subtraction or division of the correct values
- One simple way to determine the trend is to calculate a running average over the series

Trend



Trend





Lesson 3 Stationarity and Trending Behavior

3.1 Non-stationarity

3.2 Trend

3.3 Demo 1

3.4 Seasonality

3.5 Time Series Decomposition

3.6 Demo 2



Code - Stationarity and Trends

https://github.com/DataForScience/Timeseries_LL



Lesson 3 Stationarity and Trending Behavior

3.1 Non-stationarity

3.2 Trend

3.3 Demo 1

3.4 Seasonality

3.5 Time Series Decomposition

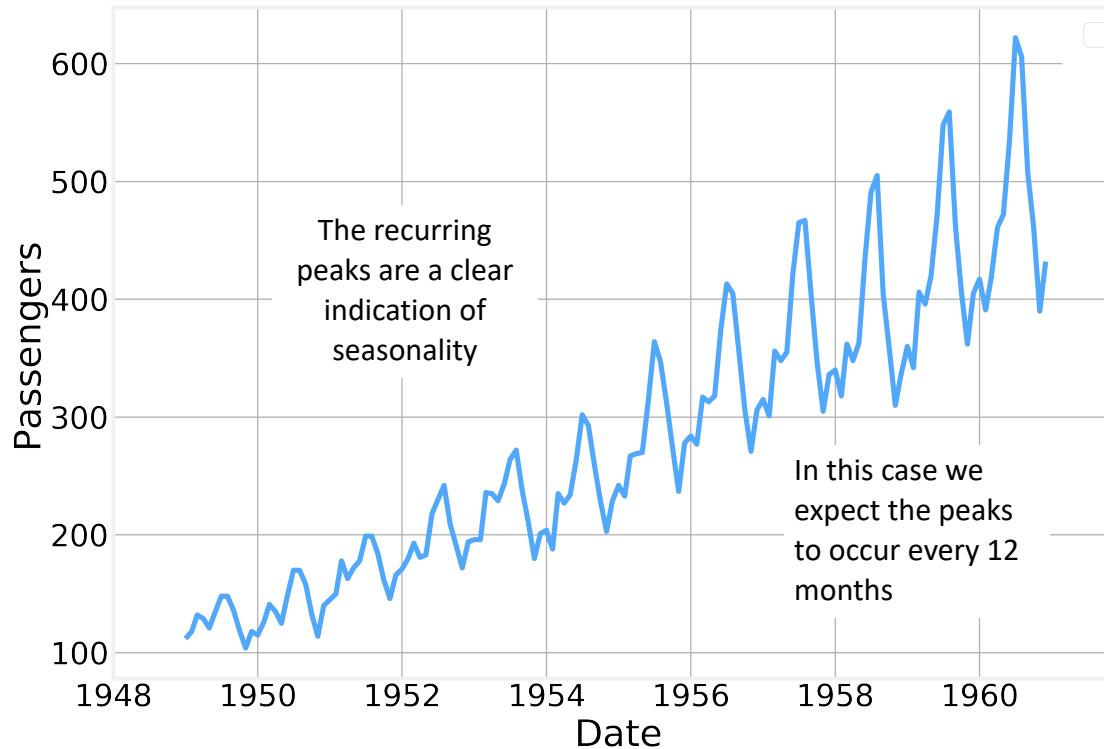
3.6 Demo 2

Seasonality

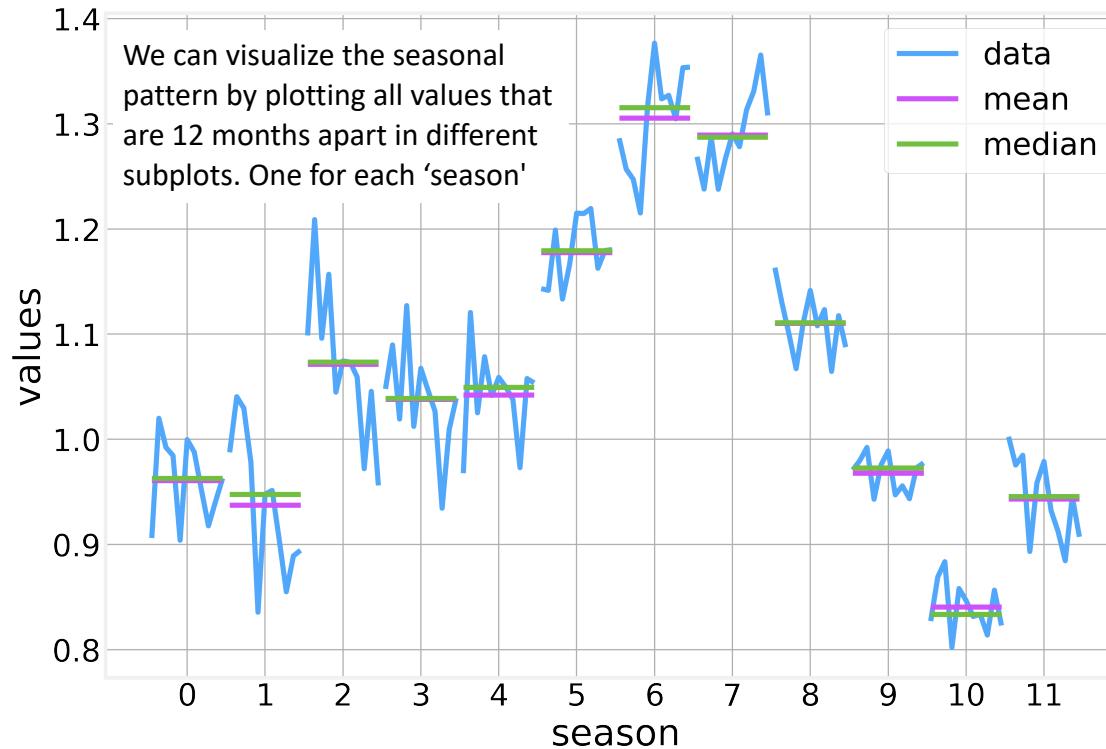
- Many of the phenomena we might be interested in varying in time in a **cyclical** or **seasonal** fashion
 - Ice-cream sales peak in the **summer** and drop in the **winter**
 - Number of cell phone calls made is larger during the day than at night
 - Many types of crime are more frequent at **night** than during the **day**
 - Visits to museums are more frequent in the **weekend** than in **weekdays**
 - The stock market grows during **bull** periods and shrinks during **bear** periods
 - etc
- Understanding the seasonality of a time series provides important information about its long term behavior and is extremely useful in predicting future values

Seasonality

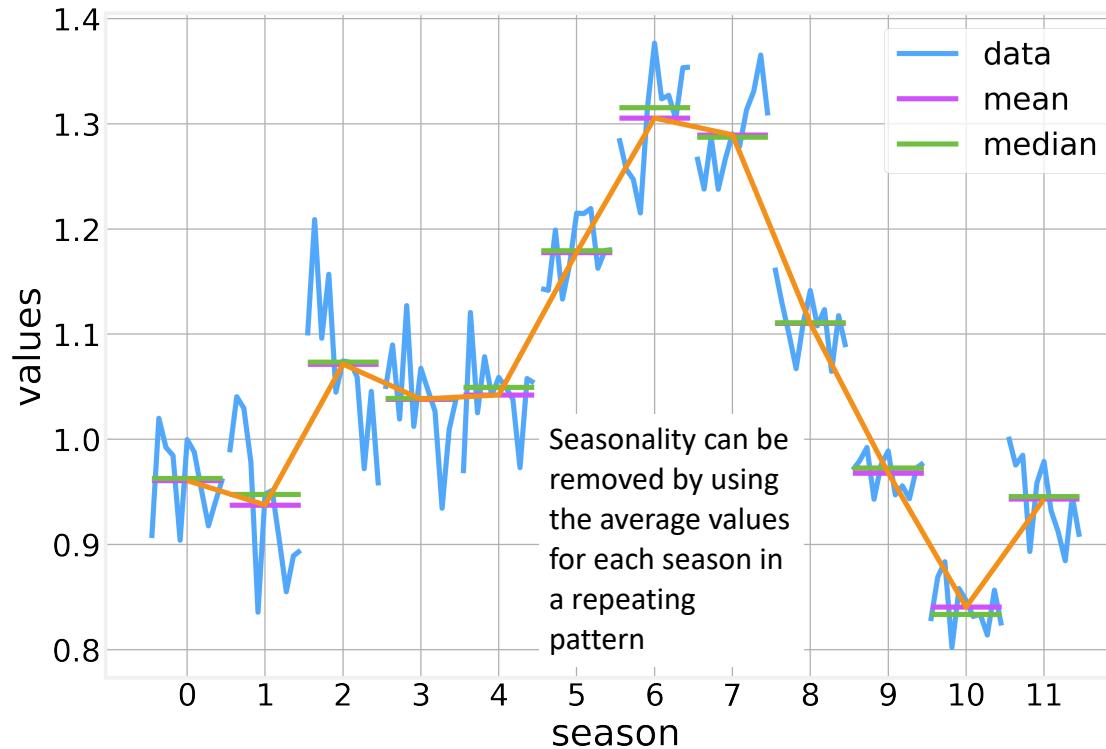
<https://www.kaggle.com/chirag19/air-passengers>



Seasonality



Seasonality





Lesson 3 Stationarity and Trending Behavior

3.1 Non-stationarity

3.2 Trend

3.3 Demo 1

3.4 Seasonality

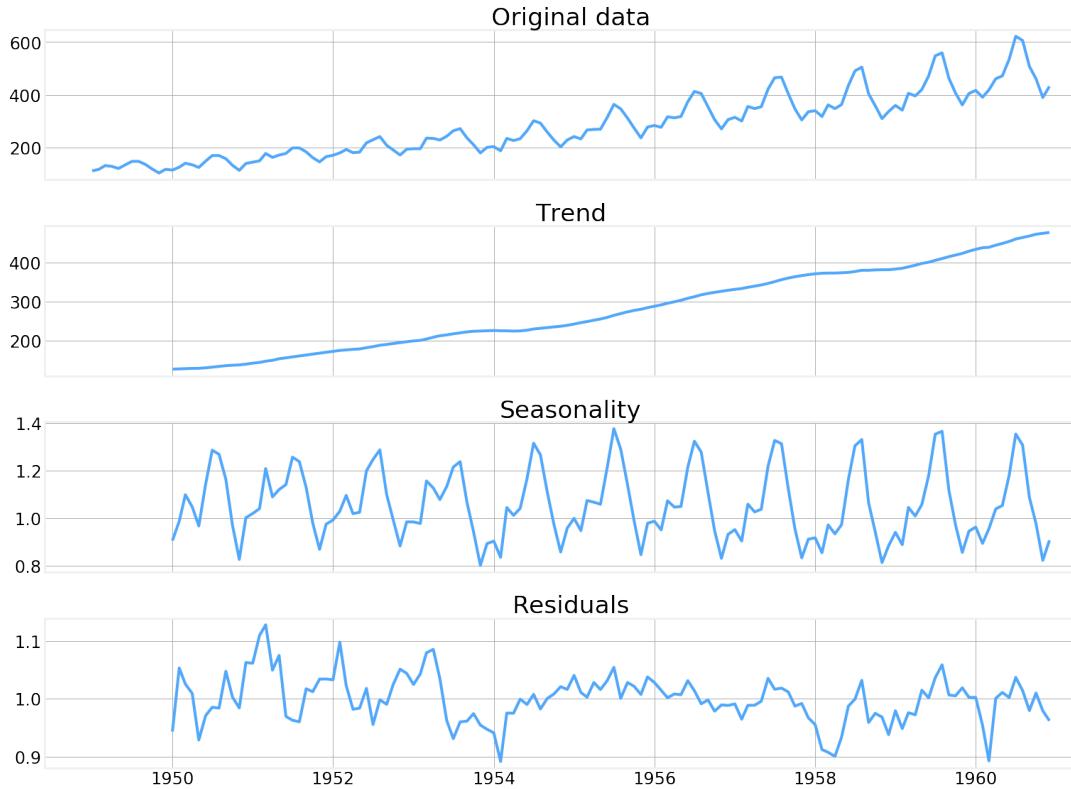
3.5 Time Series Decomposition

3.6 Demo 2

Time series decomposition

- A time series can be decomposed into three components:
 - Trend, T_t
 - Seasonality, S_t
 - Residuals, R_t
- Decompositions can be
 - additive - $x_t = T_t + S_t + R_t$
 - multiplicative - $x_t = T_t \cdot S_t \cdot R_t$
- The residuals are simply what is left of the original signal after we remove the trend and the seasonality

Time series decomposition





Lesson 3 Stationarity and Trending Behavior

3.1 Non-stationarity

3.2 Trend

3.3 Demo 1

3.4 Seasonality

3.5 Time Series Decomposition

3.6 Demo 2



Code - Decomposition

https://github.com/DataForScience/Timeseries_LL



Lesson 4 Transforming Time Series

4.1 Lagged Values

4.2 Differences

4.3 Data Imputation

4.4 Resampling

4.5 Jackknife Estimators

4.6 Bootstrapping



Lesson 4 Transforming Time Series

4.1 Lagged Values

4.2 Differences

4.3 Data Imputation

4.4 Resampling

4.5 Jackknife Estimators

4.6 Bootstrapping

Lagged values

- While analyzing time series, we often refer to values that our time series took 1, 2, 3, etc time steps in the past
- These are known as lagged values and denoted:

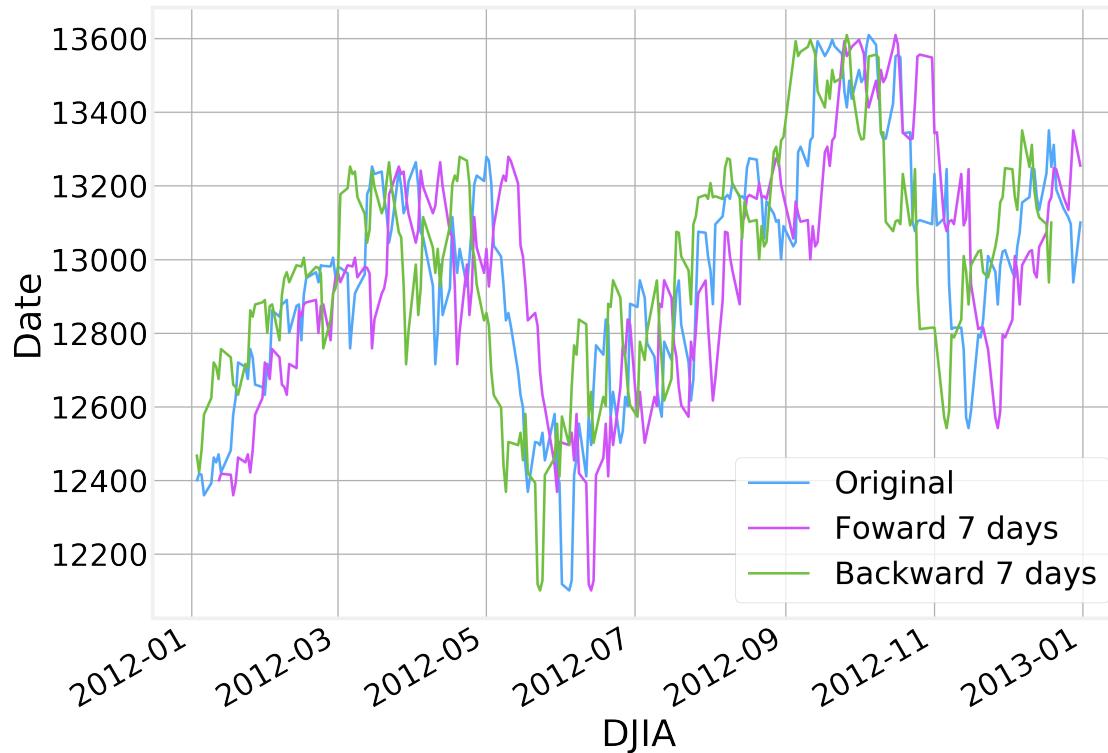
$$x_{t-l}$$

- where l is the value of the lag we are considering.

Lagged values



Lagged values





Lesson 4 Transforming Time Series

4.1 Lagged Values

4.2 Differences

4.3 Data Imputation

4.4 Resampling

4.5 Jackknife Estimators

4.6 Bootstrapping

Differences

- Perhaps the most common use case for lagged values is for the calculation of differences of the form:

$$x_t - x_{t-l}$$

- Where $l \geq 1$ is the value of the lag we are interested in.
- Naturally, higher order differences can also be used, in which case, the difference of the difference is calculated:

$$y_t = x_t - x_{t-l}$$

$$z_t = y_t - y_{t-l} \equiv x_t - 2x_{t-l} + x_{t-2l}$$

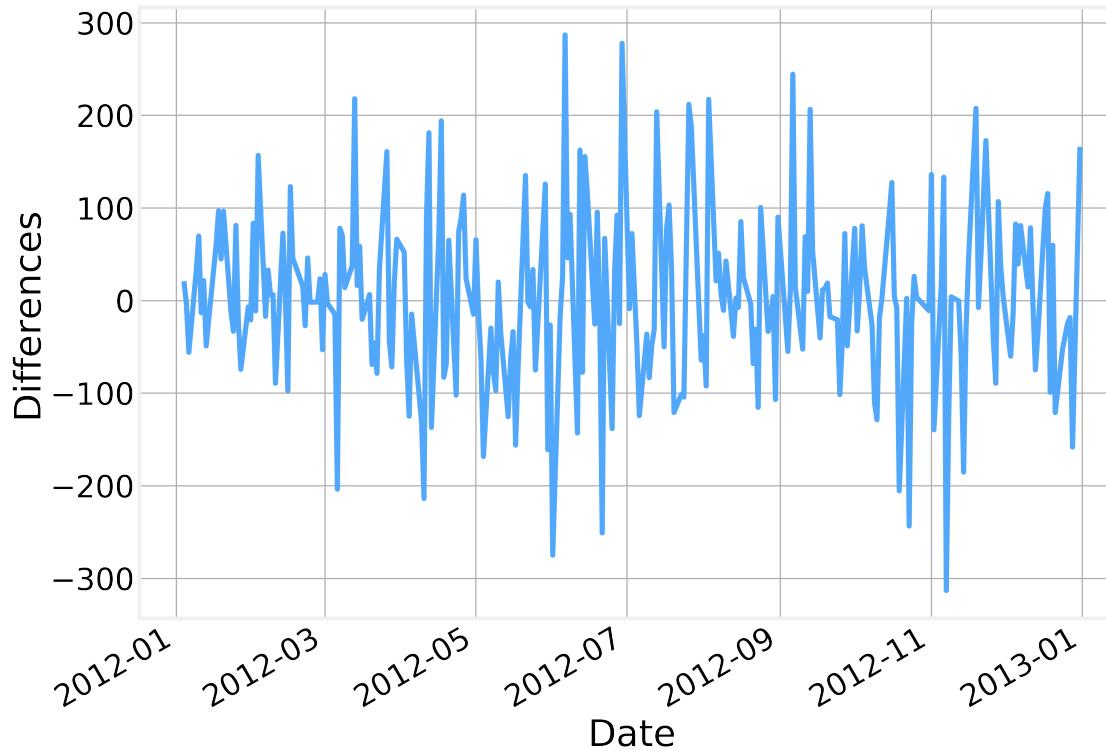
This can be thought of as a discrete version of the usual derivative of a function.

- Differences are also a particularly simple way to **detrend** a time series

Differences



Differences





Lesson 4 Transforming Time Series

4.1 Lagged Values

4.2 Differences

4.3 Data Imputation

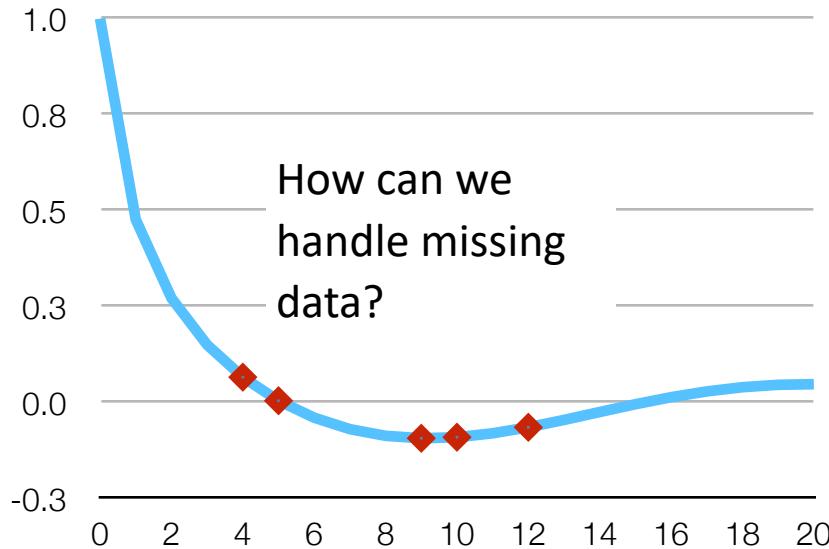
4.4 Resampling

4.5 Jackknife Estimators

4.6 Bootstrapping

Data Imputation

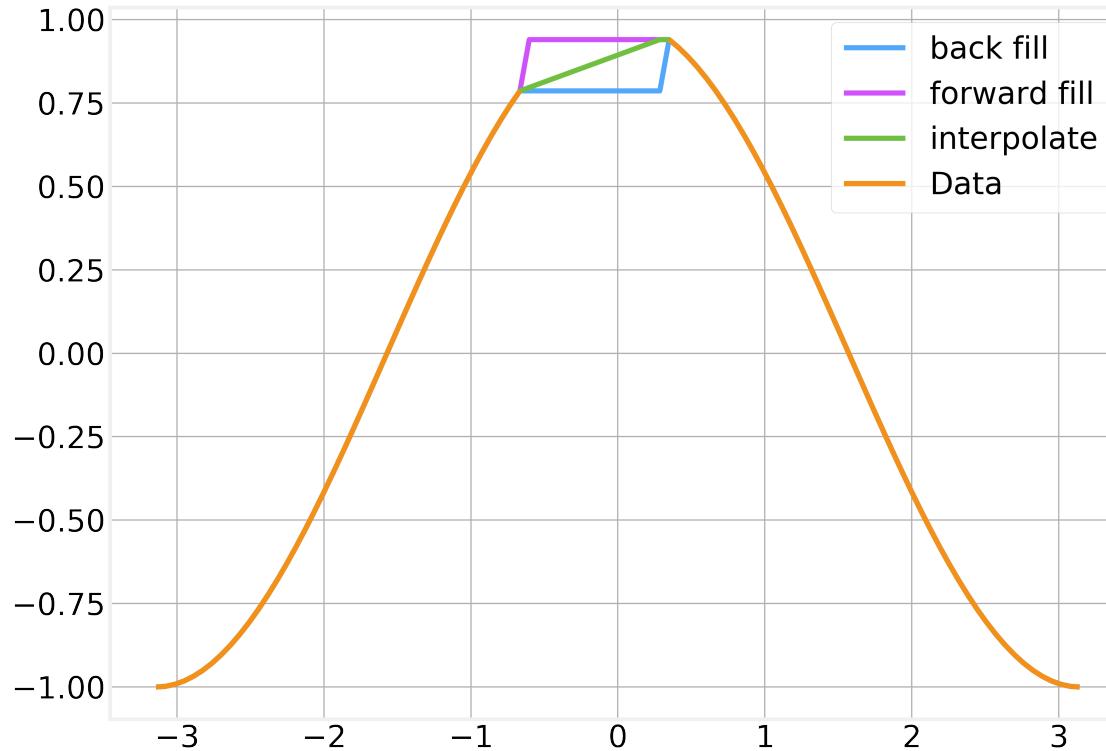
- Sometimes the time series is incomplete.
- Missing data points can be due to data corruption, data collection issues, etc.
- Missing values are represented as `nan`



Data Imputation

- Several techniques have been developed to handle this case:
 - **back fill** - keep the last previous value
 - **forward fill** - keep the next value
 - **interpolate** - add values by interpolating between the previous and the next value
 - **imputation** - add values based on what we expect the missing values to be

Data Imputation





Lesson 4 Transforming Time Series

4.1 Lagged Values

4.2 Differences

4.3 Data Imputation

4.4 Resampling

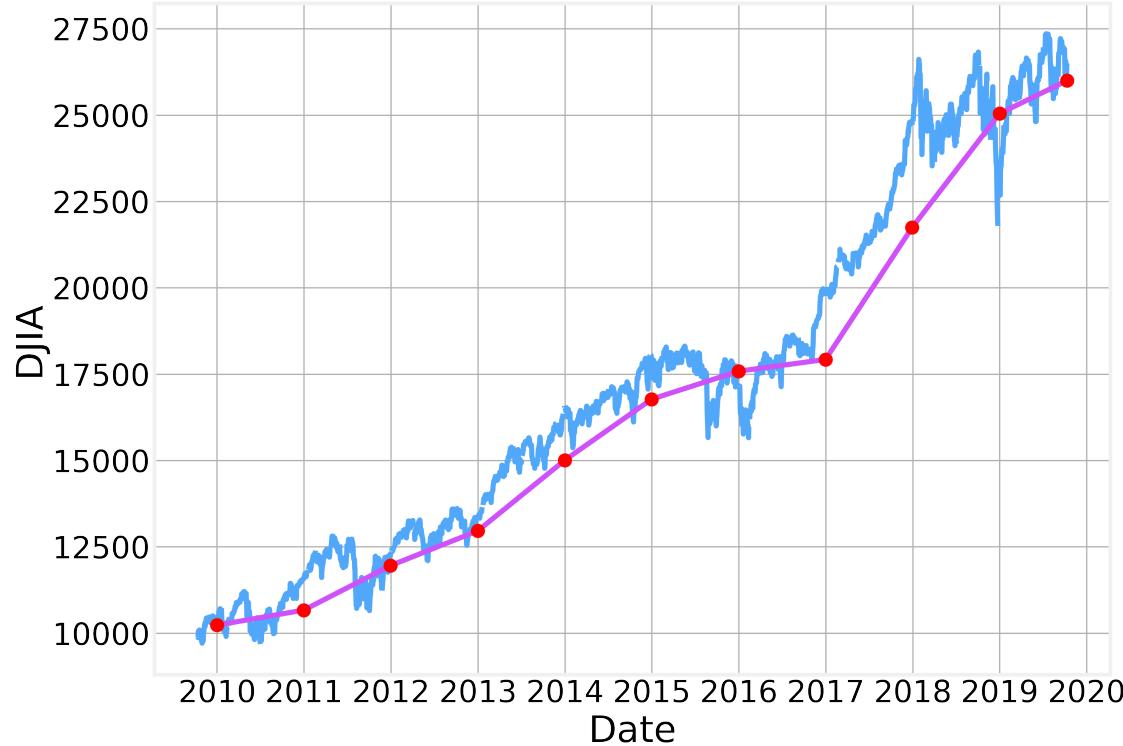
4.5 Jackknife Estimators

4.6 Bootstrapping

Resampling

- Time series typically have an **intrinsic time scale** at which the data was collected: ticks, seconds, days, months, etc
- In many cases, our analysis requires that we **resample** the data to a different time scale
- Resampling to a longer timescale is relatively simple and similar to aggregation:
 - Transforming from daily to weekly frequency requires simply aggregating by week
- Resampling to shorter timescales requires **interpolation or imputation** to make up for the missing values
 - Going from weekly to daily frequency requires **specifying how to**

Resampling





Lesson 4 Transforming Time Series

4.1 Lagged Values

4.2 Differences

4.3 Data Imputation

4.4 Resampling

4.5 Jackknife Estimators

4.6 Bootstrapping

Jackknife Estimation

- Also known as ‘leave one out’ estimation
- Commonly used for mean and variance estimates
- The **Jackknife estimate** of a parameter is the average value of the parameter calculated by omitting each of the values one at a time.
- If μ_i is the mean calculated by omitting the i^{th} value, then the Jackknife estimate of the mean is given by:
$$\hat{\mu} = \frac{1}{N} \sum_{i=1}^N \mu_i$$
- And the **variance** of the estimate is:

$$\hat{\sigma}(\hat{\mu}) = \frac{N-1}{N} \sum_{i=1}^N (\mu_i - \hat{\mu})^2$$



Lesson 4 Transforming Time Series

4.1 Lagged Values

4.2 Differences

4.3 Data Imputation

4.4 Resampling

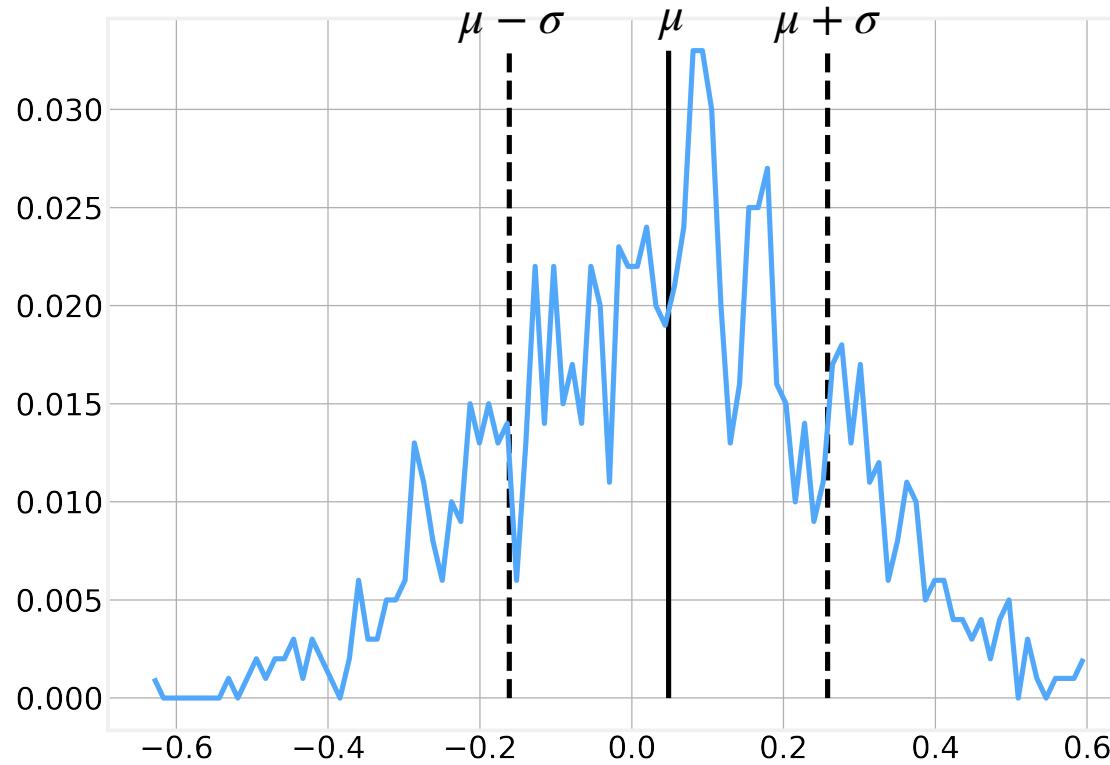
4.5 Jackknife Estimators

4.6 Bootstrapping

Bootstrapping

- Bootstrapping is another alternative to estimate statistical properties such as mean and variance
- Bootstrapping measures the desired property in a large number of samples (with replacement), of the observed dataset.
- Each sample has equal size to the observed dataset.
- From the entire population of samples, the empirical bootstrap distribution of the expected values can be obtained to provide information about the distribution in the total population

Bootstrapping





Code - Transformations

https://github.com/DataForScience/Timeseries_LL



Lesson 5 Running Value Measures

5.1 Windowing

5.2 Running Values

5.3 Bollinger Bands

5.4 Exponential Running Averages

5.5 Forecasting



Lesson 5 Running Value Measures

5.1 Windowing

5.2 Running Values

5.3 Bollinger Bands

5.4 Exponential Running Averages

5.5 Forecasting

Windowing

- When analyzing the temporal behavior of a signal, we often need to evaluate if specific quantities are **time varying** or not
- A common approach is to use **sliding windows** of a given length to evaluate the required values
- So a sliding window of width 6 on a series of length 11 would look like:

$x_0 \ x_1 \ x_2 \ x_3 \ x_4 \ x_5 \ x_6 \ x_7 \ x_8 \ x_9 \ x_{10}$

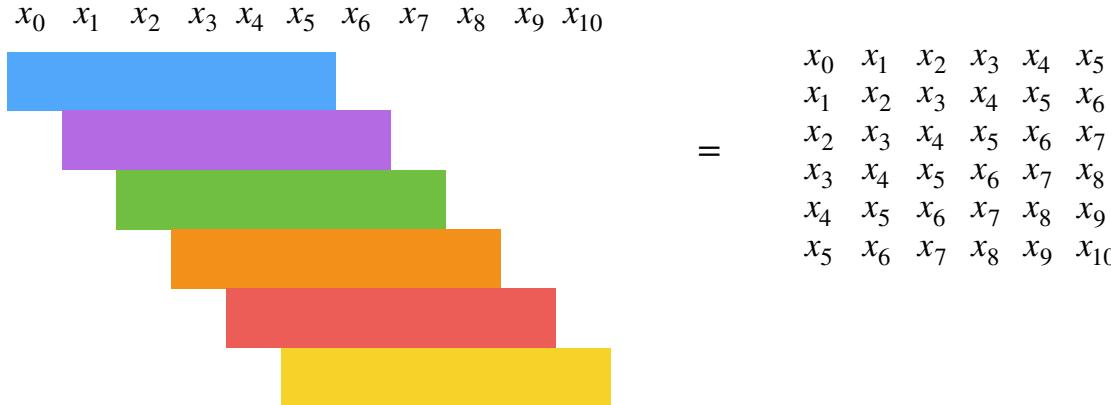


$x_0 \ x_1 \ x_2 \ x_3 \ x_4 \ x_5$
 $x_1 \ x_2 \ x_3 \ x_4 \ x_5 \ x_6$
 $x_2 \ x_3 \ x_4 \ x_5 \ x_6 \ x_7$
 $x_3 \ x_4 \ x_5 \ x_6 \ x_7 \ x_8$
 $x_4 \ x_5 \ x_6 \ x_7 \ x_8 \ x_9$
 $x_5 \ x_6 \ x_7 \ x_8 \ x_9 \ x_{10}$

=

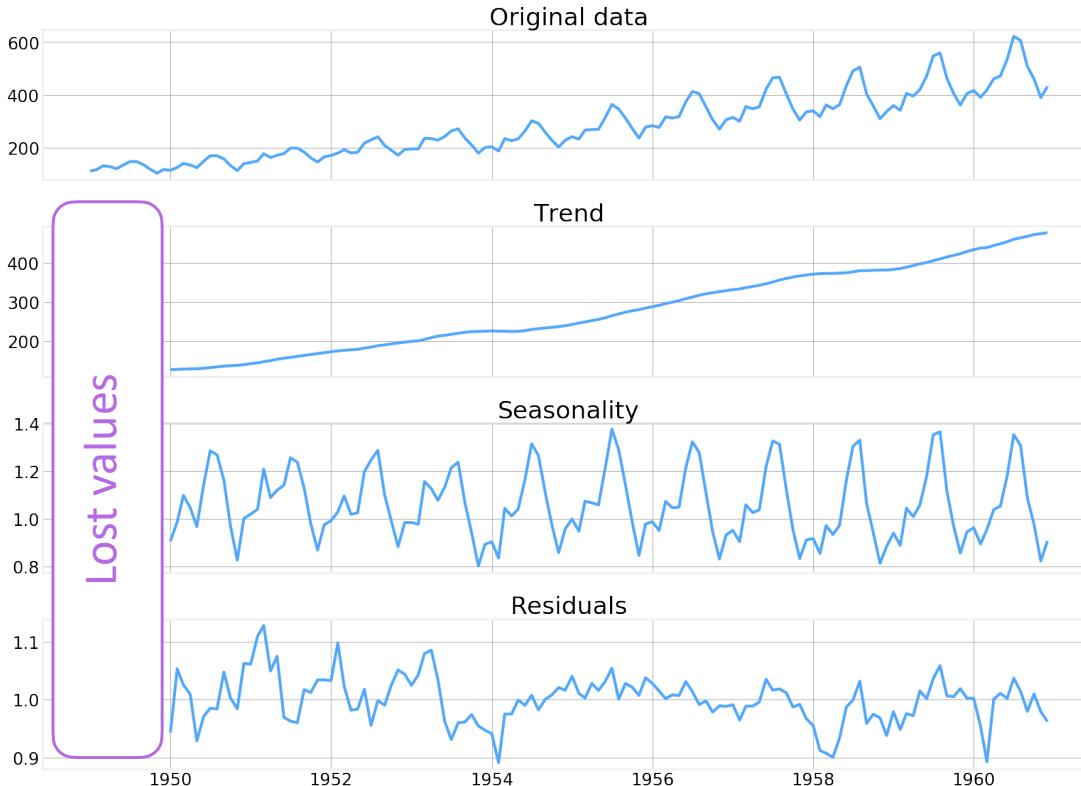
Windowing

- We calculate the metric of interest [within each window](#).
- By using sliding windows we necessarily lose some data points



Windowing

One common approach is to place all “lost values” at the beginning as it avoids “future leaking” when splitting the dataset





Lesson 5 Running Value Measures

5.1 Windowing

5.2 Running Values

5.3 Bollinger Bands

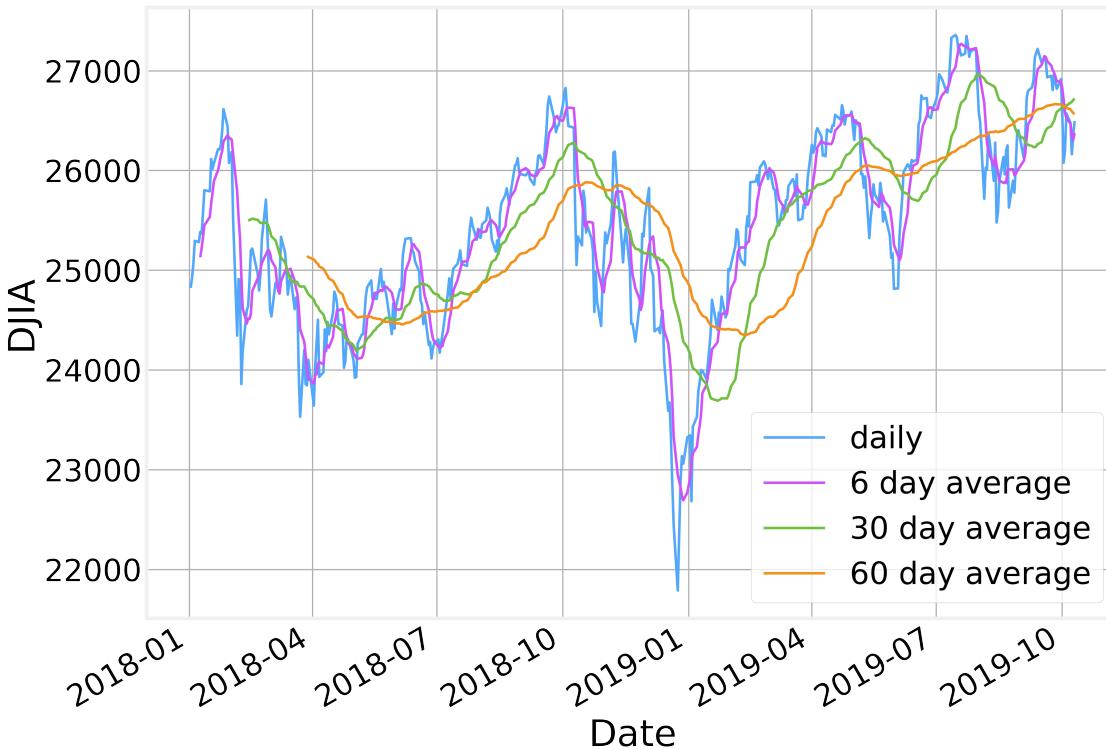
5.4 Exponential Running Averages

5.5 Forecasting

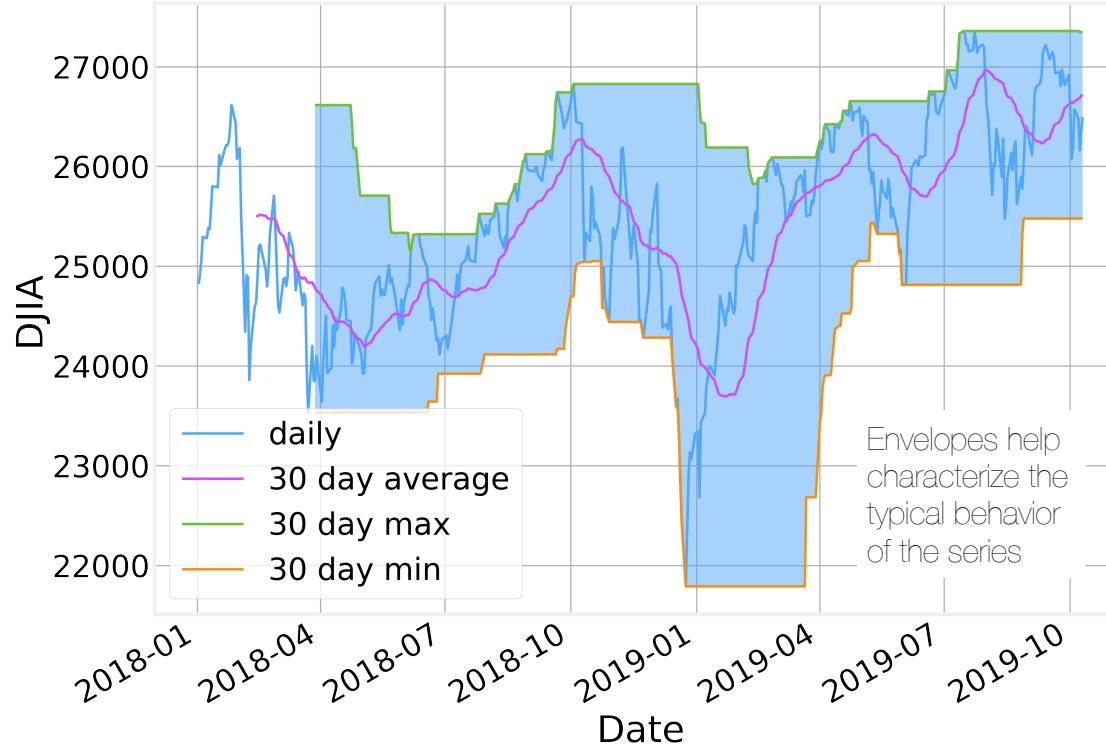
Running Values

- In the first part of the lecture, we already used running averages to detrend a time series
- Other common metrics are:
 - Variance
 - Maximum value
 - Minimum value
 - etc...
- One important detail to note is that while using windowing to calculate running values we “lose” a number of points equal to the width of the window
- Depending on the application we can choose to place the missing values in either or (or even both) extremes of the time interval

Running Values



Envelopes





Lesson 5 Running Value Measures

5.1 Windowing

5.2 Running Values

5.3 Bollinger Bands

5.4 Exponential Running Averages

5.5 Forecasting

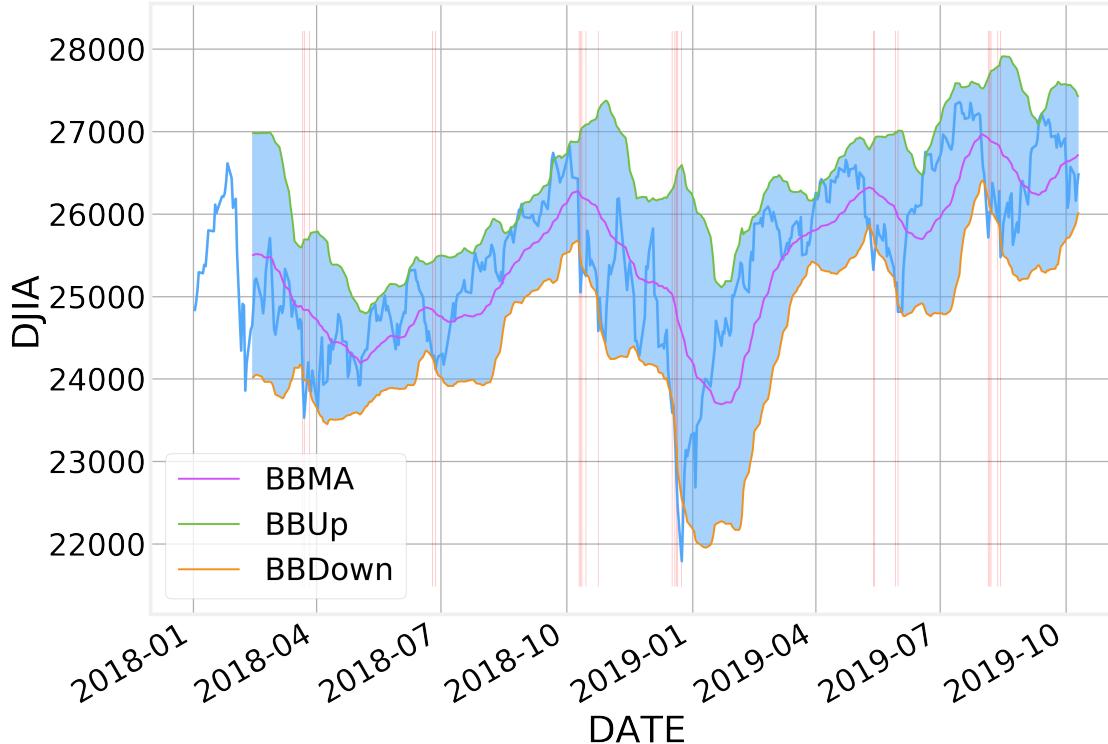
Bollinger Bands

- A common use for application for running values is the calculation of Bollinger Bands.
- Introduced by [John Bollinger](#) in the 1980s as a complement to more traditional time series technical analysis techniques.
- Bollinger Bands are defined by two components:
 - A N period moving average, μ_N
 - The area K standard deviations above and below the moving average $\mu_N \pm K\sigma_N$

Bollinger Bands

- Both μ_N and σ_N are computed on a [running window](#) of size N
- The values of N and K are application specific. For stock trading, $N = 20$ and $K = 2$ are typical values.
- Whenever the time series steps out of the Bollinger Band that's a clear indication of a [change in the temporal behavior](#).

Bollinger Bands





Lesson 5 Running Value Measures

5.1 Windowing

5.2 Running Values

5.3 Bollinger Bands

5.4 Exponential Running Averages

5.5 Forecasting

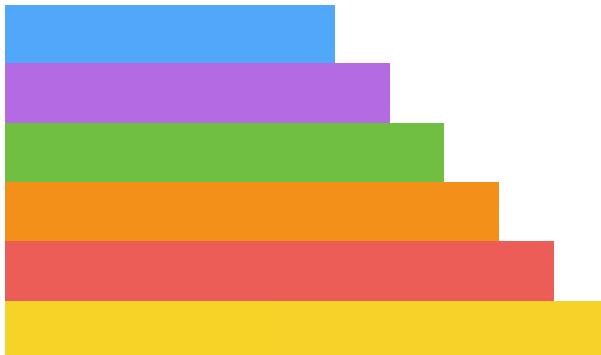
Exponential Running Average

- One alternative to a simple running average is Exponential Smoothing
- The exponentially “smooth” version of a time series is given by:

$$z_t = \alpha x_t + (1 - \alpha) z_{t-1}$$

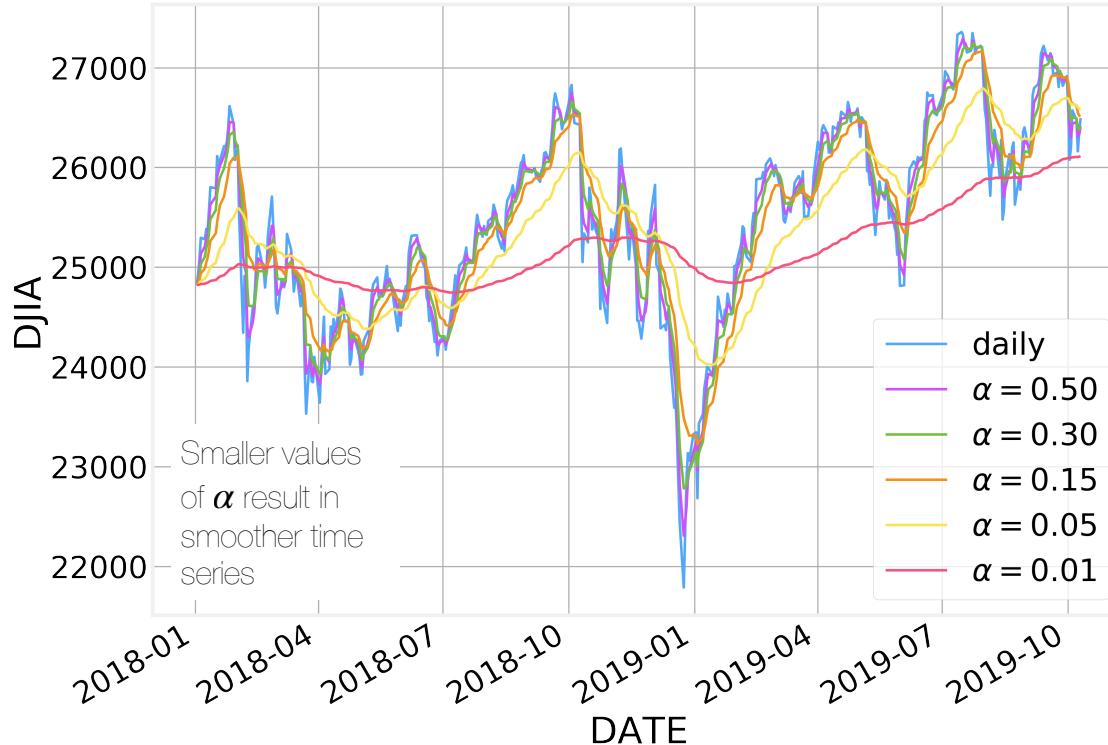
- The smaller the value of the weight α , the less influence each point has on the transformed time series.
- Each point depends implicitly on all previous points

$x_0 \quad x_1 \quad x_2 \quad x_3 \quad x_4 \quad x_5 \quad x_6 \quad x_7 \quad x_8 \quad x_9 \quad x_{10}$



$$\begin{pmatrix} \alpha & & & & & \\ \alpha(1-\alpha)^1 & \alpha & & & & \\ \alpha(1-\alpha)^2 & \alpha(1-\alpha)^1 & \alpha & & & \\ \vdots & \vdots & \vdots & \ddots & & \\ \alpha(1-\alpha)^{n-1} & \alpha(1-\alpha)^{n-2} & \alpha(1-\alpha)^{n-3} & & & \alpha \end{pmatrix}$$

Exponential Running Average





Lesson 5 Running Value Measures

5.1 Windowing

5.2 Running Values

5.3 Bollinger Bands

5.4 Exponential Running Averages

5.5 Forecasting

Forecasting

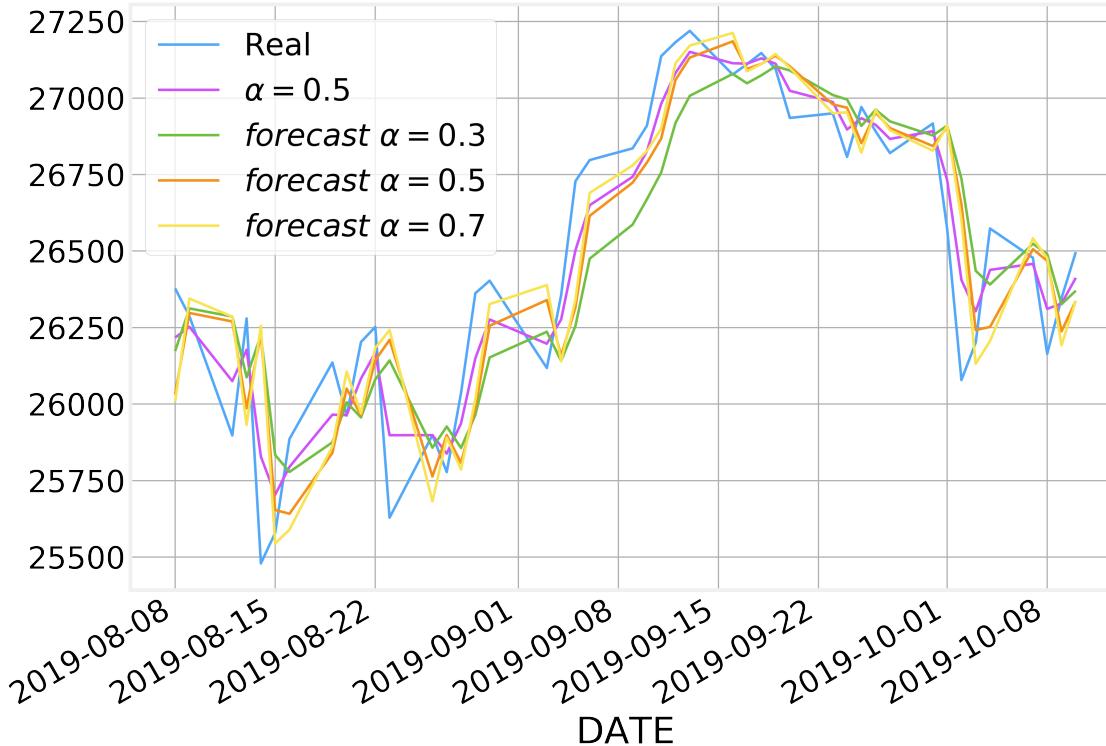
- We can also use the Exponential Moving Averages as a simple forecasting tool.
- The value at time $t + 1$ is given by:

$$z_{t+1} = \alpha x_t + (1 - \alpha) z_t$$

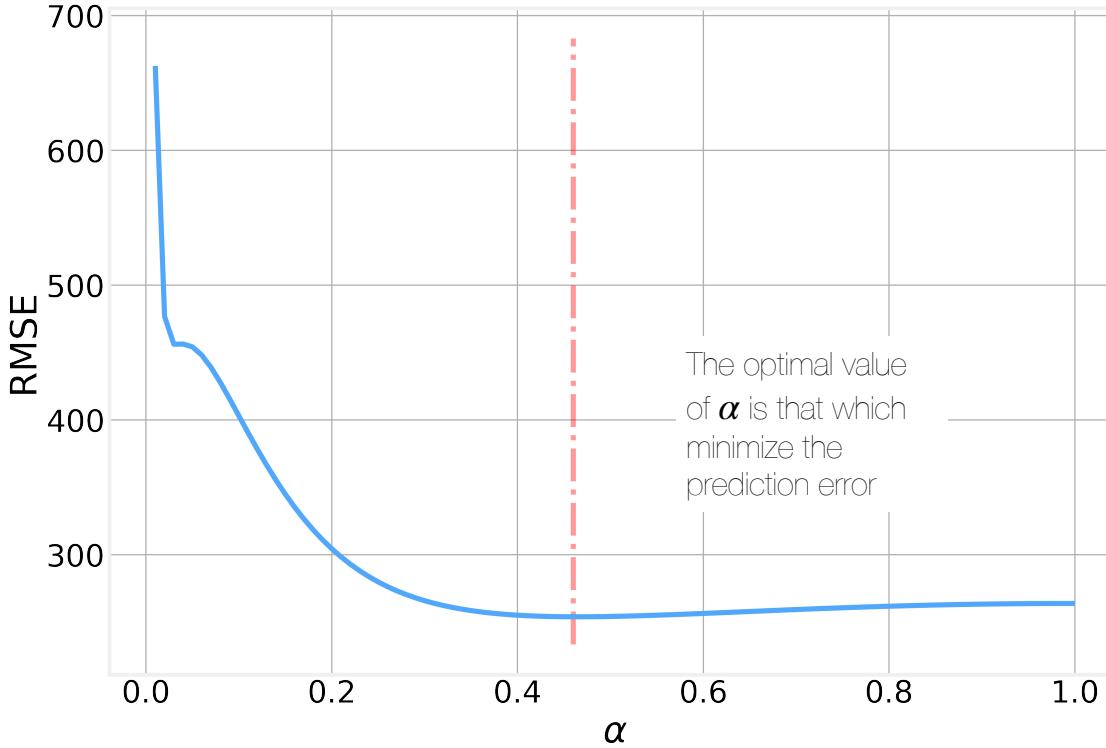
- Which we can consider to be a prediction on the value of x_{t+1} , based on the current value of z_t and some fraction of our current error value $x_t - z_t$:

$$z_{t+1} = z_t + \alpha (x_t - z_t)$$

Forecasting



Forecasting





Code - Running Values

https://github.com/DataForScience/Timeseries_LL



Lesson 6 Fourier Analysis

6.1 Frequency Domain

6.2 Discrete Fourier Transform

6.3 FFT For Filtering

6.4 Forecasting



Lesson 6 Fourier Analysis

6.1 Frequency Domain

6.2 Discrete Fourier Transform

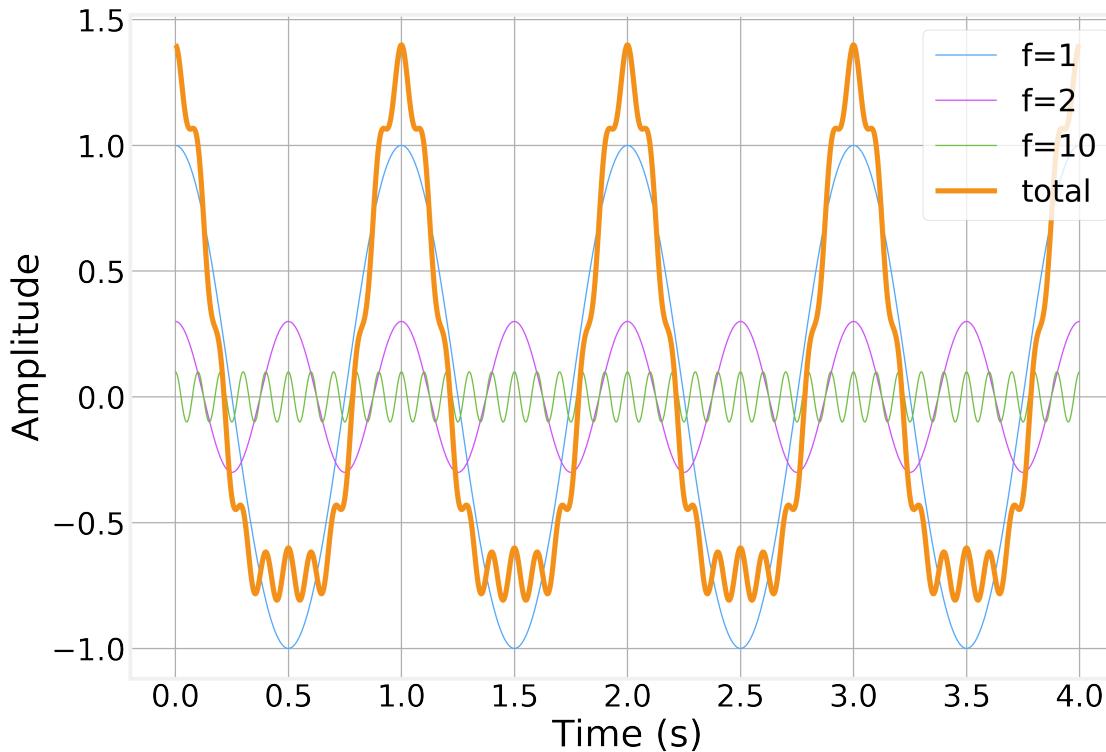
6.3 FFT For Filtering

6.4 Forecasting

Frequency Domain

- So far we have focused on the natural (time based) representation of a time series.
- An alternative representation is based on frequencies and was first introduced by [Jean Fourier](#) in 1807
- Fourier showed that periodic functions can be decomposed as a [sum of trigonometric functions](#)
- Fourier's original result was later extended to [all functions](#)
- The [Discrete Fourier Transforms](#) provides us with an simple and convenient way to move from the [time-domain](#) to the [frequency-domain](#) and back.

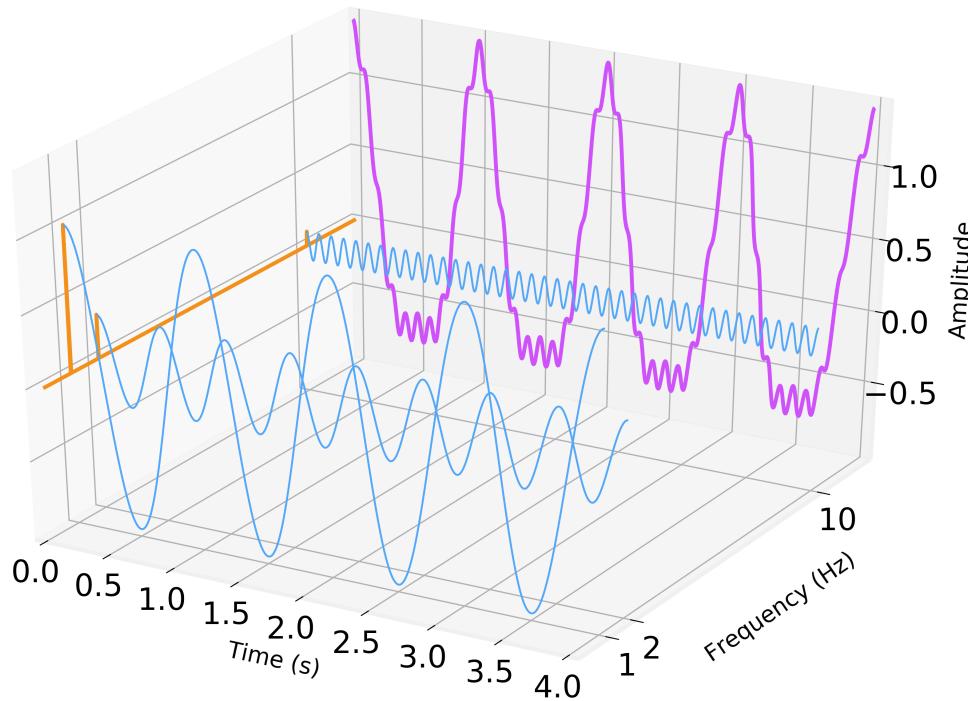
Adding Frequencies



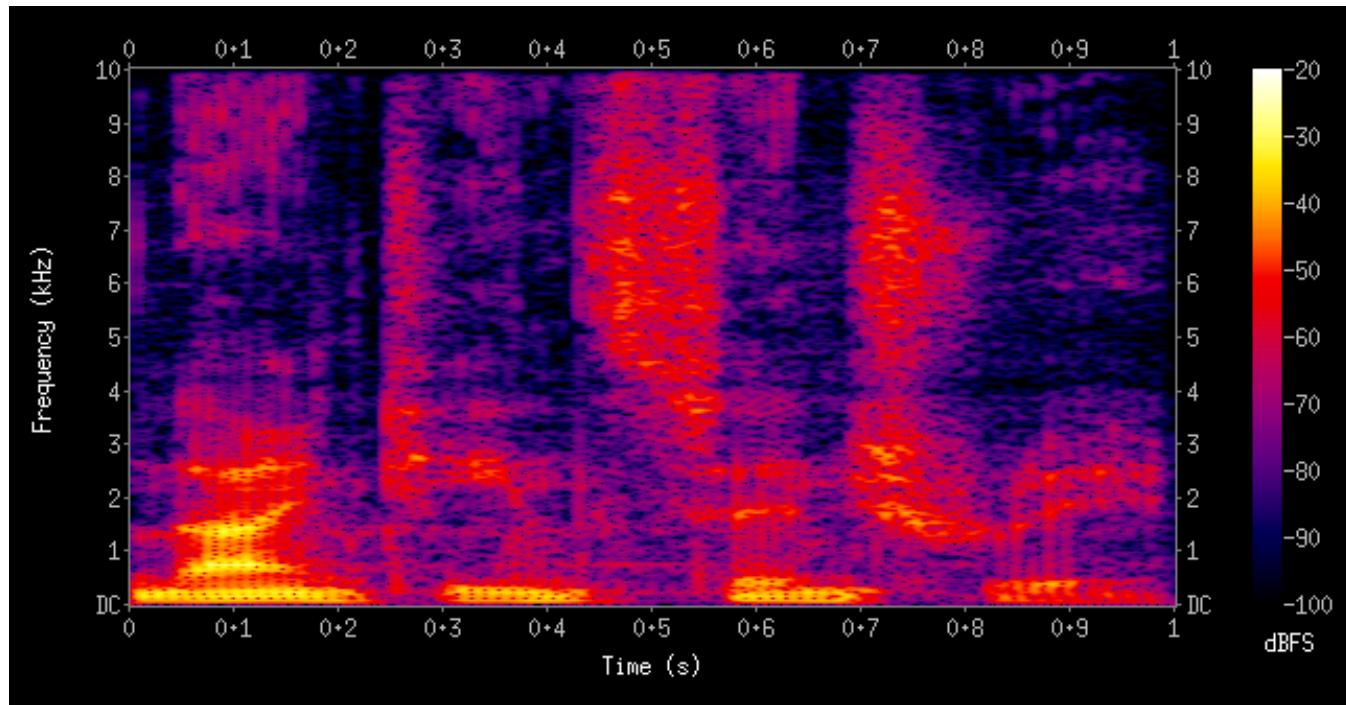
Complex functions can be constructed by adding simple waves of the right frequencies and amplitudes

3D Visualization

Complex functions can be constructed by adding simple waves of the right frequencies and amplitudes



Spectrogram





Lesson 6 Fourier Analysis

6.1 Frequency Domain

6.2 Discrete Fourier Transform

6.3 FFT For Filtering

6.4 Forecasting

(Discrete) Fourier Transform

- The DFT maps a sequence of N values x_n representing a time series $x(t)$ into N complex numbers X_k defined as:

$$X_k = \sum_{n=0}^{N-1} x_n e^{-i \frac{2\pi k n}{N}}$$

- where:

$$e^{i\theta} = \cos(\theta) + i \sin(\theta)$$

(Discrete) Inverse Fourier Transform

- To recover the original values we use the Inverse DFT, defined as:

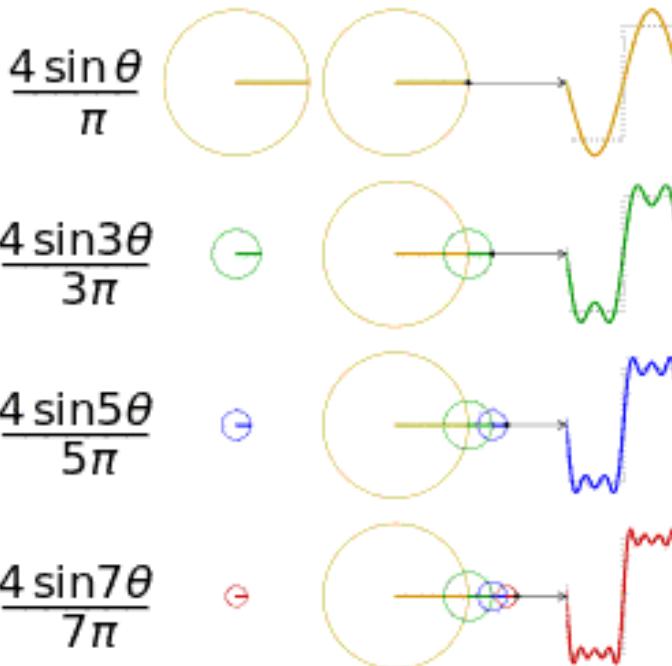
$$x_n = \frac{1}{N} \sum_{k=0}^{N-1} x_k e^{i \frac{2\pi k n}{N}}$$

- The DFT represents the continuous series $x(t)$ as a sum of discrete frequencies:

$$\omega_n = 2\pi \frac{k}{N}$$

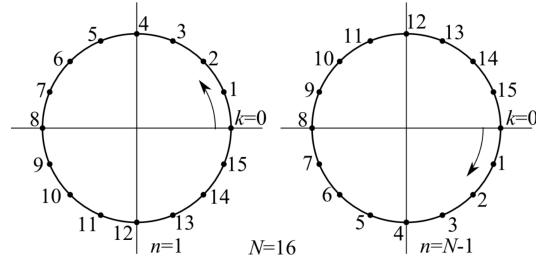
(Discrete) Fourier Transform

https://en.wikipedia.org/wiki/Discrete-time_Fourier_transform



numpy

- Provides practical implementation of the [Fast Fourier Transform](#) an efficient algorithm to compute the [DFT](#) and [IDFT](#). See https://en.wikipedia.org/wiki/Fast_Fourier_transform
- `numpy.fft.fft()`/`numpy.fft.ifft()` - [DFT](#) and [IFT](#)
- `numpy.fft.fftfreq()` - return the list of frequencies
- `numpy.fft.fftshift()`/`numpy.fft.ifftshift()` - Shift the zero-frequency component to the center of the spectrum and back.





Lesson 6 Fourier Analysis

6.1 Frequency Domain

6.2 Discrete Fourier Transform

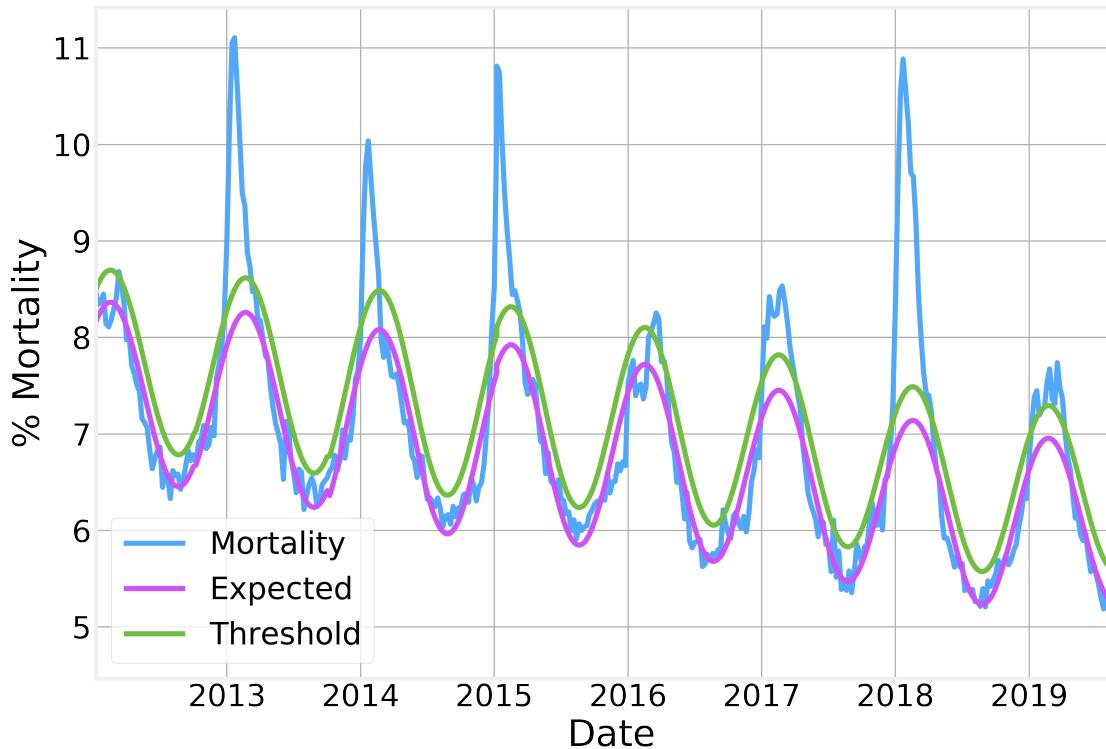
6.3 FFT For Filtering

6.4 Forecasting

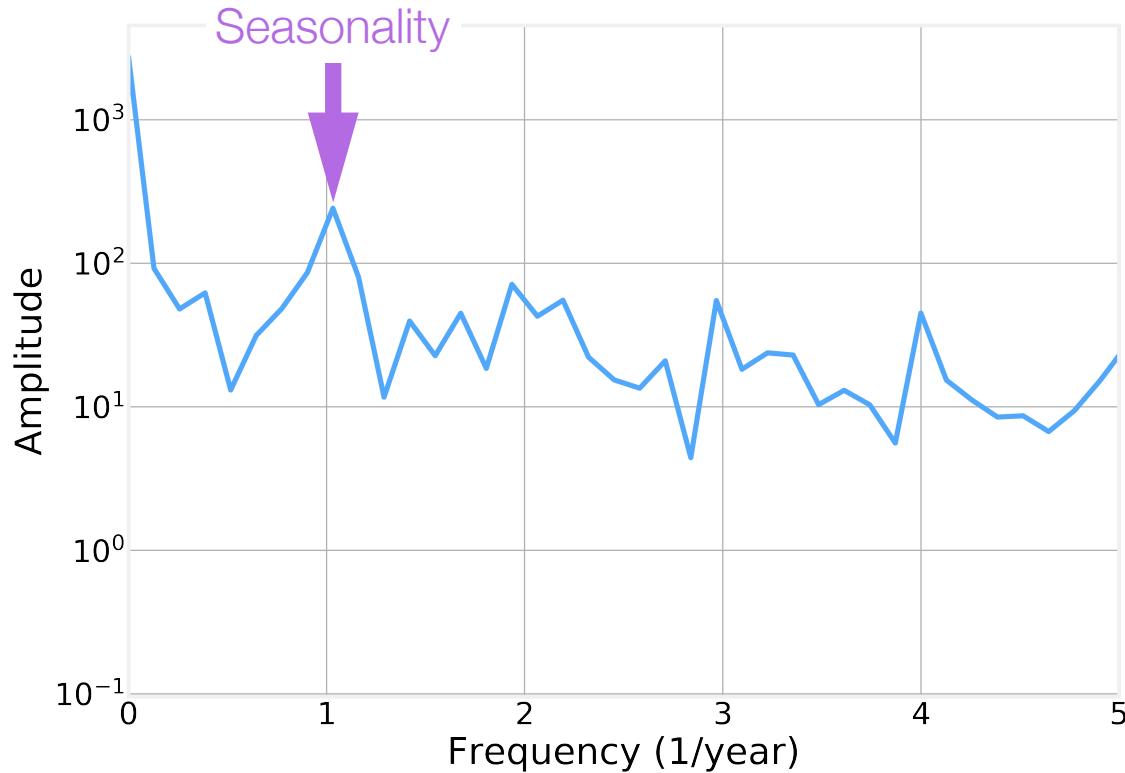
Filtering

- Common applications of Fourier Analysis are:
 - Seasonality - determine the main frequency underlying a time series
 - Filtering - remove higher order frequencies to eliminate noise
 - Processing - Several signal processing operations are simpler to compute in the frequency-space
 - Extrapolation/Forecasting - We can also extend the reconstructed signal to future values, with a couple of small caveats...

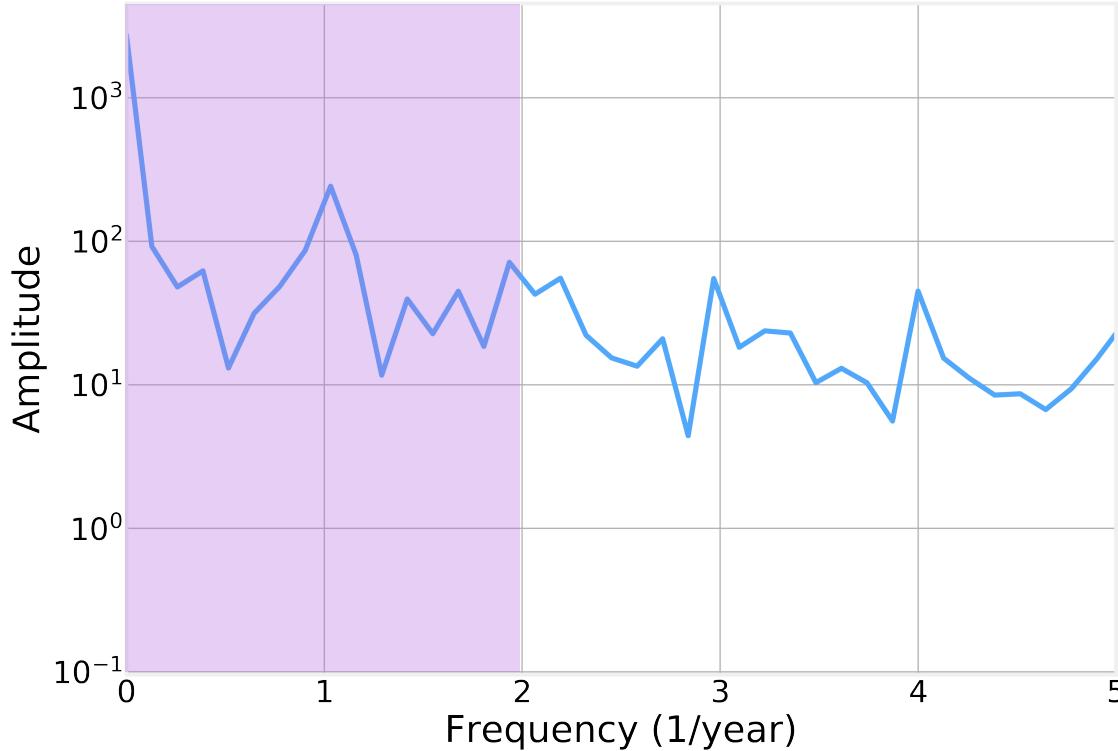
Filtering



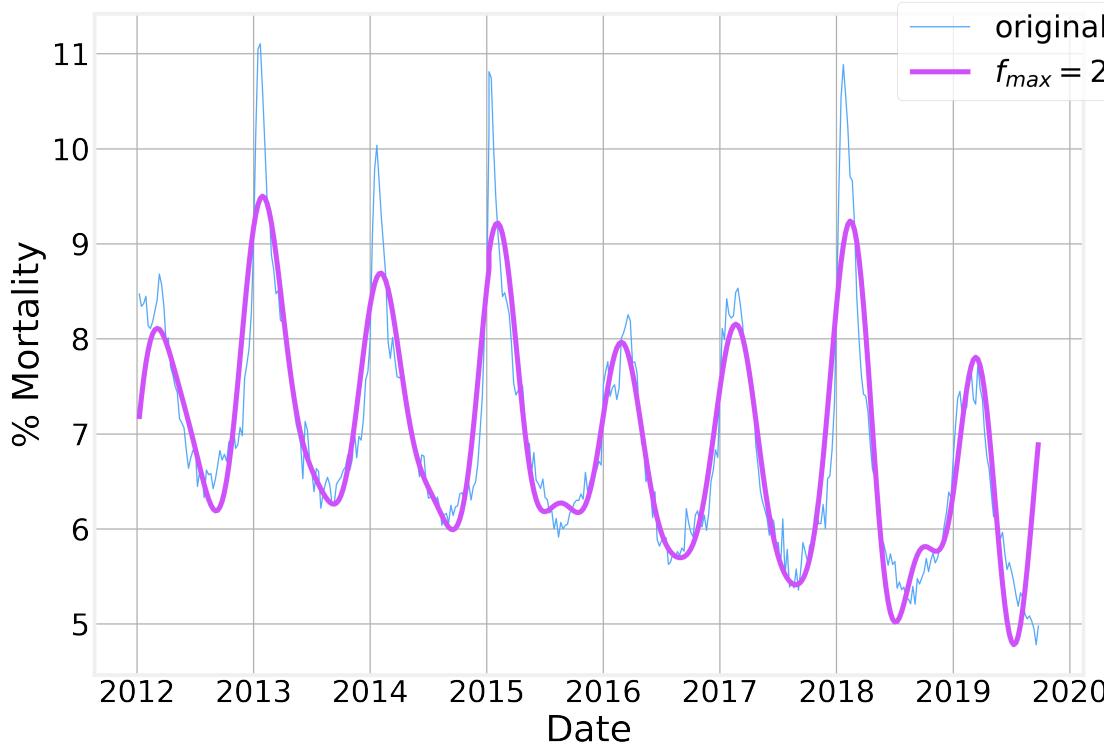
Filtering



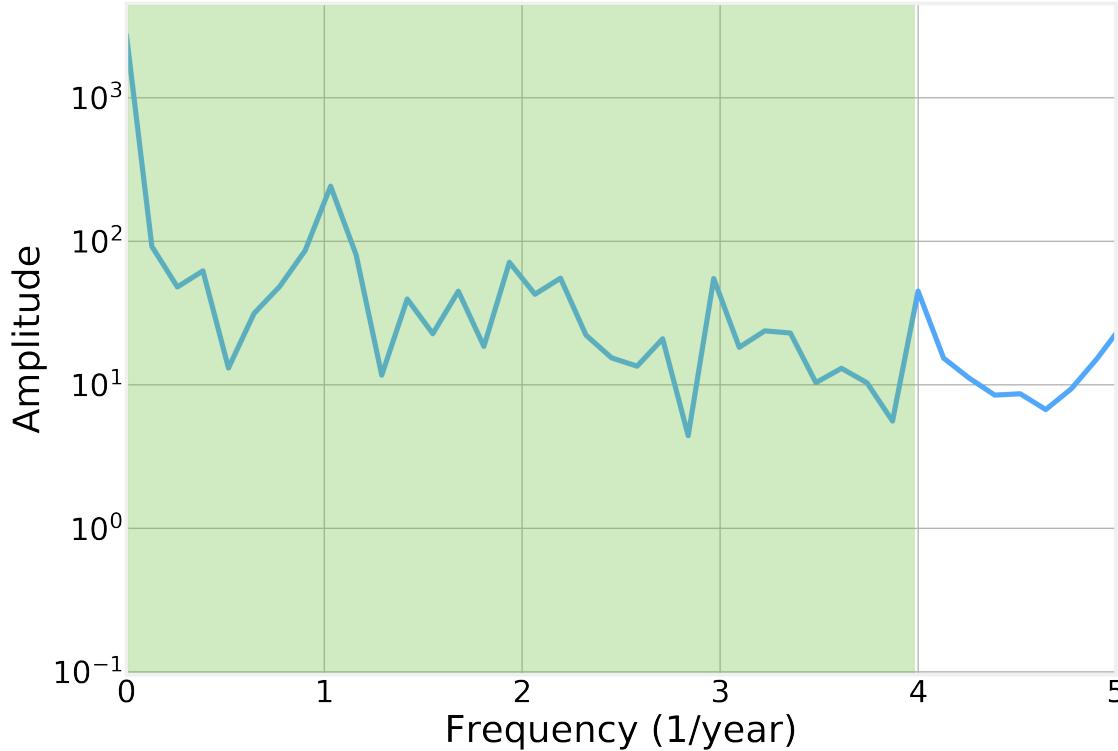
Filtering



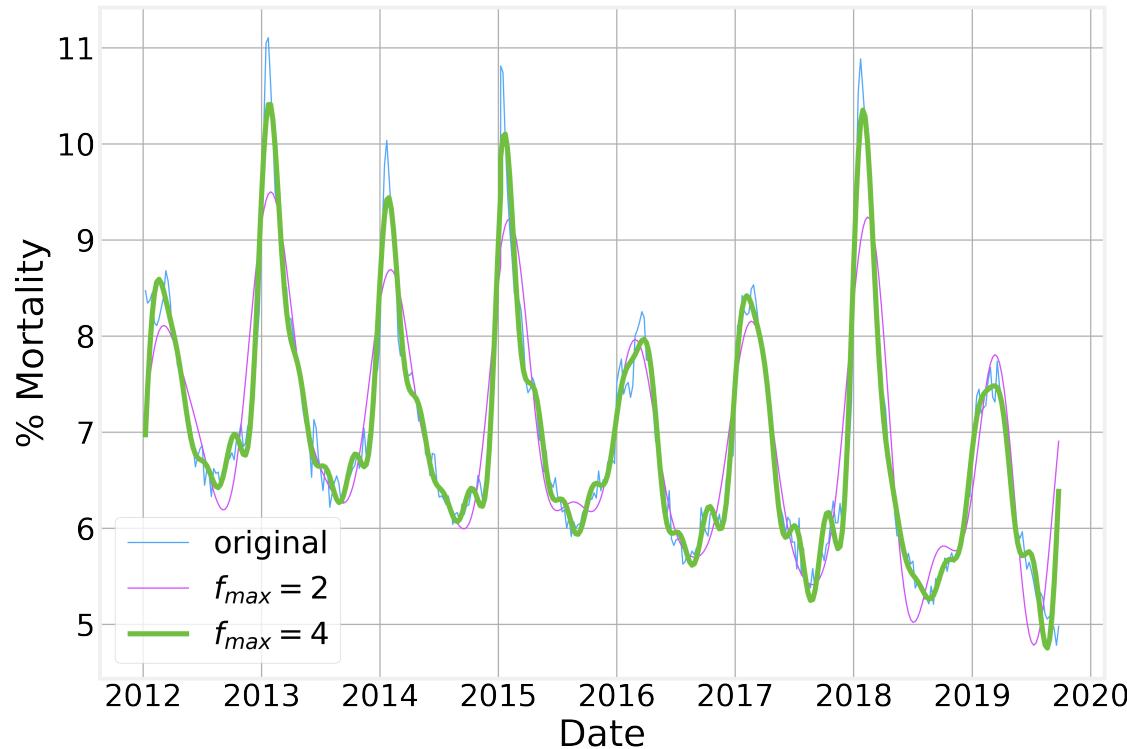
Filtering



Filtering



Filtering





Lesson 6 Fourier Analysis

6.1 Frequency Domain

6.2 Discrete Fourier Transform

6.3 FFT For Filtering

6.4 Forecasting

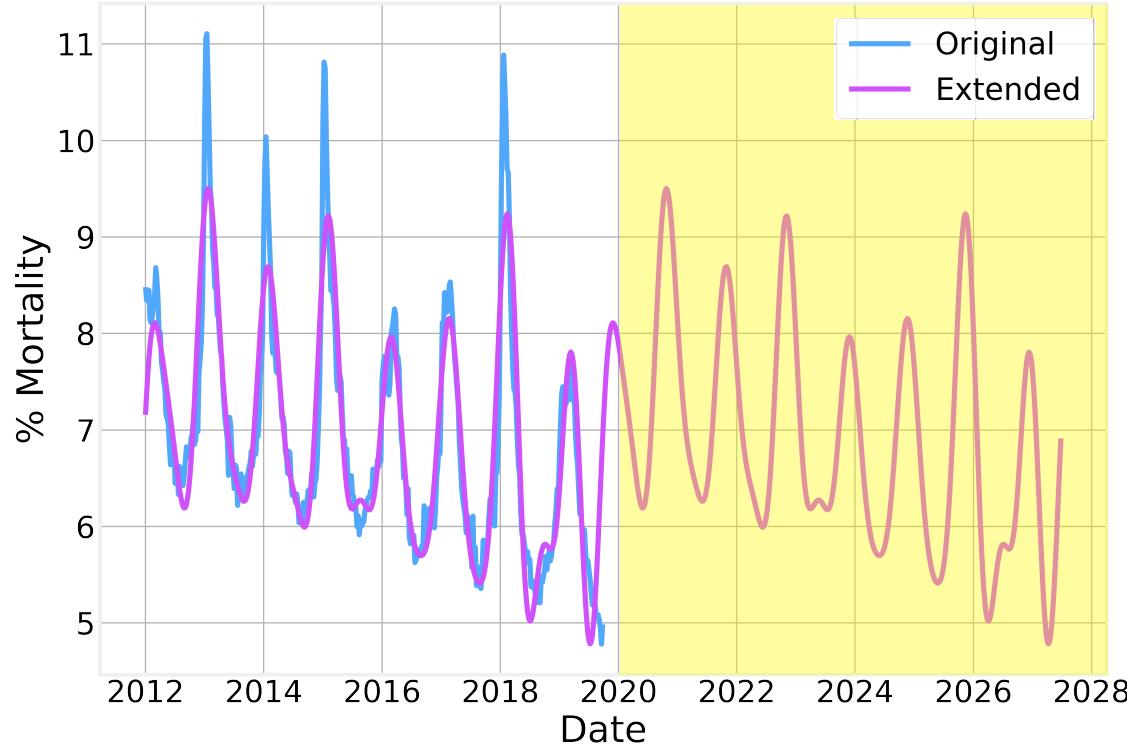
Forecasting

- As we saw above, we can recover the original signal from the FFT values by using:

$$x_n = \frac{1}{N} \sum_{k=0}^{N-1} x_k e^{i \frac{2\pi k n}{N}}$$

- Where n is our time variable.
- There's nothing stopping us from extending the values of n beyond the original domain of the signal
- The resulting extrapolated values correspond to a forecast into the future.

Forecasting





Code - Fourier Analysis

https://github.com/DataForScience/Timeseries_LL



Lesson 7 Time Series Correlations

7.1 Pearson Correlation

7.2 Correlations of Two Time Series

7.3 Auto-Correlation

7.4 Partial Auto-Correlation



Lesson 7 Time Series Correlations

7.1 Pearson Correlation

7.2 Correlations of Two Time Series

7.3 Auto-Correlation

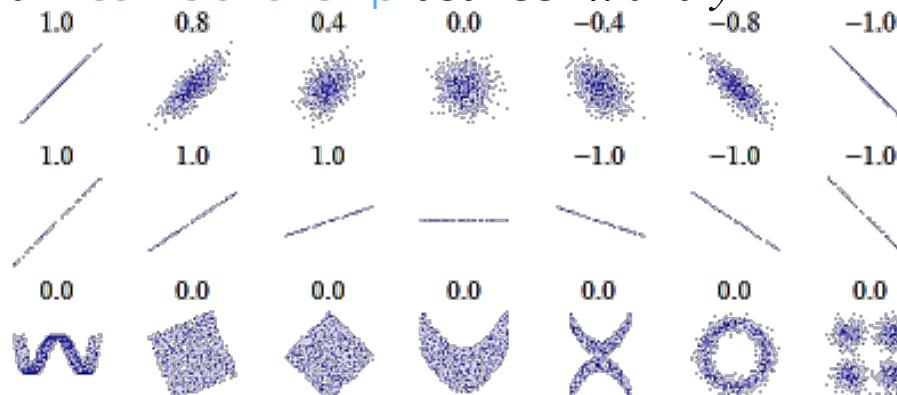
7.4 Partial Auto-Correlation

Correlation

- Many correlation measures have been proposed over the years
- The most well known one is the Pearson Correlation

$$\rho(x, y) = \sum_{i=1}^N \frac{(x_i - \mu_x)(y_i - \mu_y)}{\sigma_x \sigma_y}$$

- Assumes a linear relationship between x and y .





Lesson 7 Time Series Correlations

7.1 Pearson Correlation

7.2 Correlations of Two Time Series

7.3 Auto-Correlation

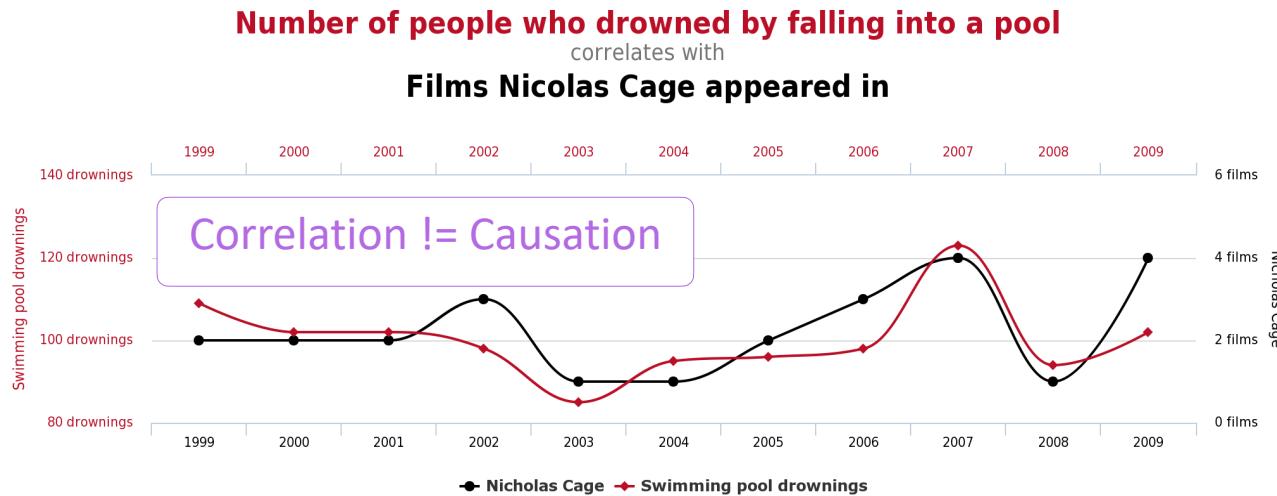
7.4 Partial Auto-Correlation

Correlations of 2 time series

- The correlation of two time series gives you an indication of how similar their behavior is
- Two completely unrelated time series (say, two sequences of random numbers) will have a Pearson correlation coefficient of 0

Correlations of 2 time series

- The correlation of two time series gives you an indication of how similar their behavior is
- Two completely unrelated time series (say, two sequences of random numbers) will have a Pearson correlation coefficient of 0



tylervigen.com

Correlations of 2 time series

- The correlation of two time series gives you an indication of how similar their behavior is
- Two completely unrelated time series (say, two sequences of random numbers) will have a Pearson correlation coefficient of 0
- Correlation != Causation!
- Adding a trend to both series we immediately observe a significant correlation

The Pearson correlation of two trending series is overwhelmed by the trend



Lesson 7 Time Series Correlations

7.1 Pearson Correlation

7.2 Correlations of Two Time Series

7.3 Auto-Correlation

7.4 Partial Auto-Correlation

Auto-correlation

- It follows from the previous slide that a series will have a perfect correlation with itself, but what about lagged versions of itself?
- We define the Auto-correlation function as the Pearson correlation between values of the time series at different lags, as a function of the lag:

$$ACF_x(l) = \rho(x_t, x_{t-l})$$

- By definition, $ACF_x(0) \equiv 1$
- And as the lag l increases the value of the ACF tends to decrease.

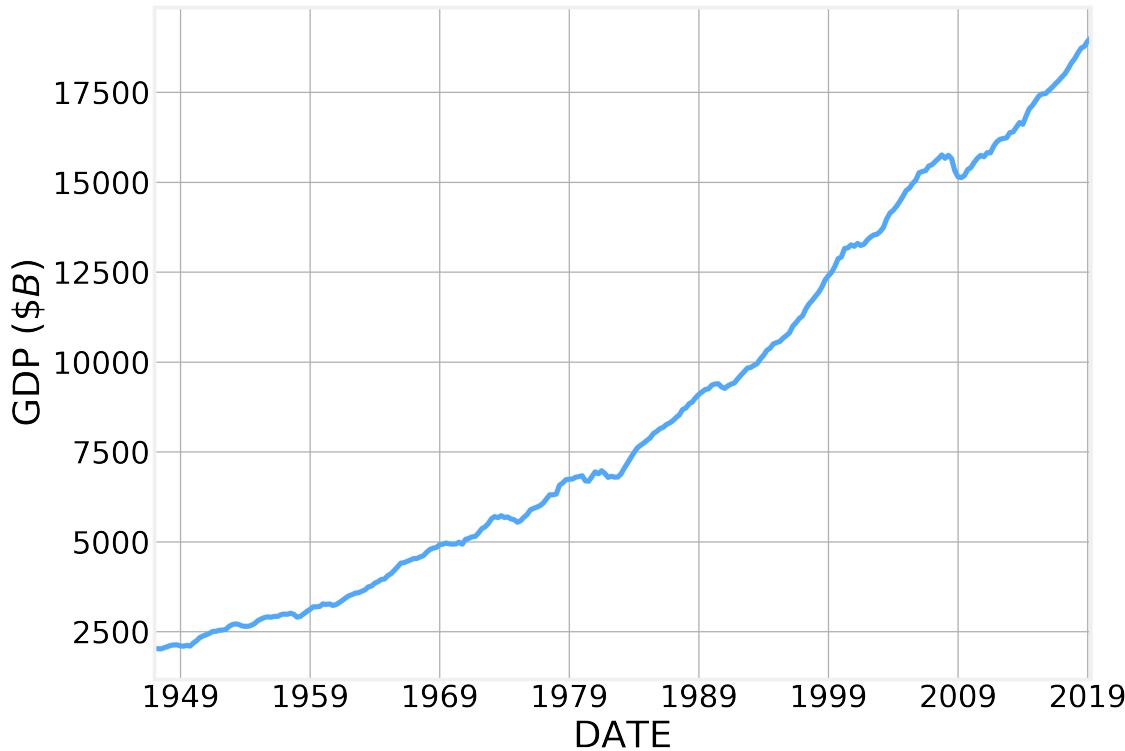
Auto-correlation

- We can calculate the confidence interval for the *ACF* using:

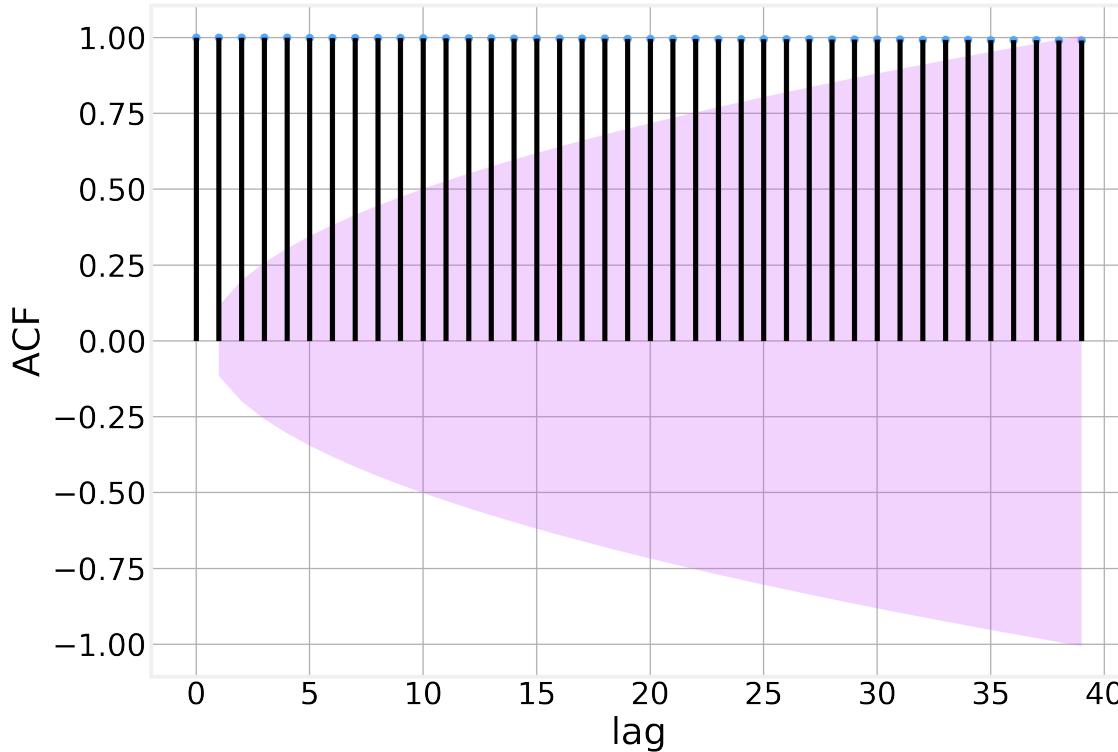
$$CI = \pm z_{1-\alpha/2} \sqrt{\frac{1}{N} \left(1 + 2 \sum_{l=1}^k r_l^2 \right)}$$

- where $z_{1-\alpha/2}$ is the quantile of the normal distribution corresponding to significance level α and r_l are the values of the *ACF* for a specific lag l

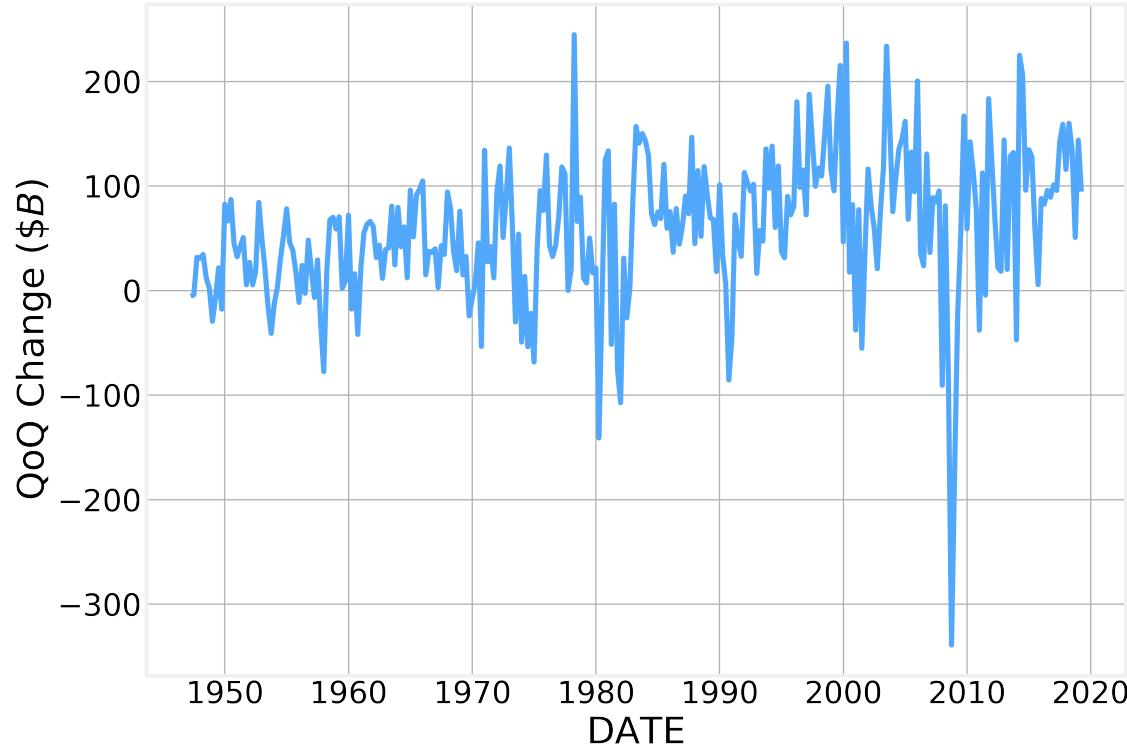
Auto-correlation



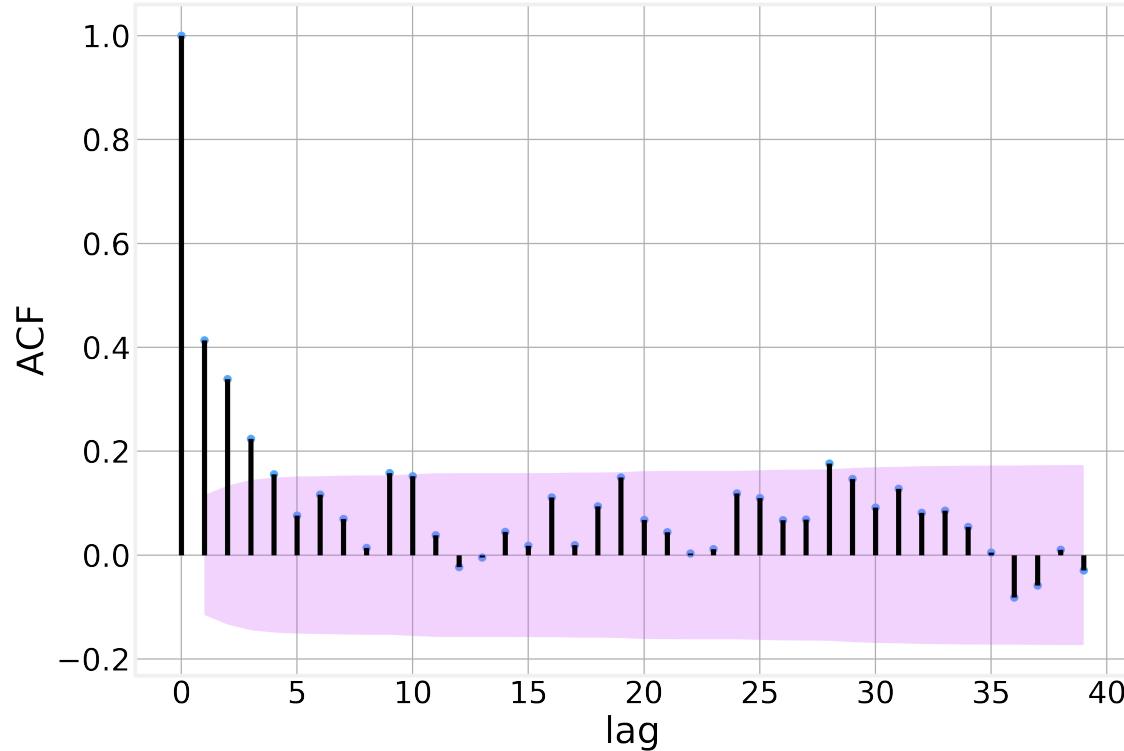
Auto-correlation



Auto-correlation



Auto-correlation





Lesson 7 Time Series Correlations

7.1 Pearson Correlation

7.2 Correlations of Two Time Series

7.3 Auto-Correlation

7.4 Partial Auto-Correlation

Partial Autocorrelation

- One of the disadvantages of the Autocorrelation function is that it still considers the intermediate values
- The Partial Autocorrelation function calculate the correlation function between x_t and x_{t-l} after explaining away all the intermediate values $x_{t-1} \dots x_{t-l+1}$
- Intermediate values are "explained away" by fitting a linear model of the form:

$$\hat{x}_t = f(x_{t-1} \dots x_{t-l+1})$$

$$\hat{x}_{t-l} = f(x_{t-1} \dots x_{t-l+1})$$

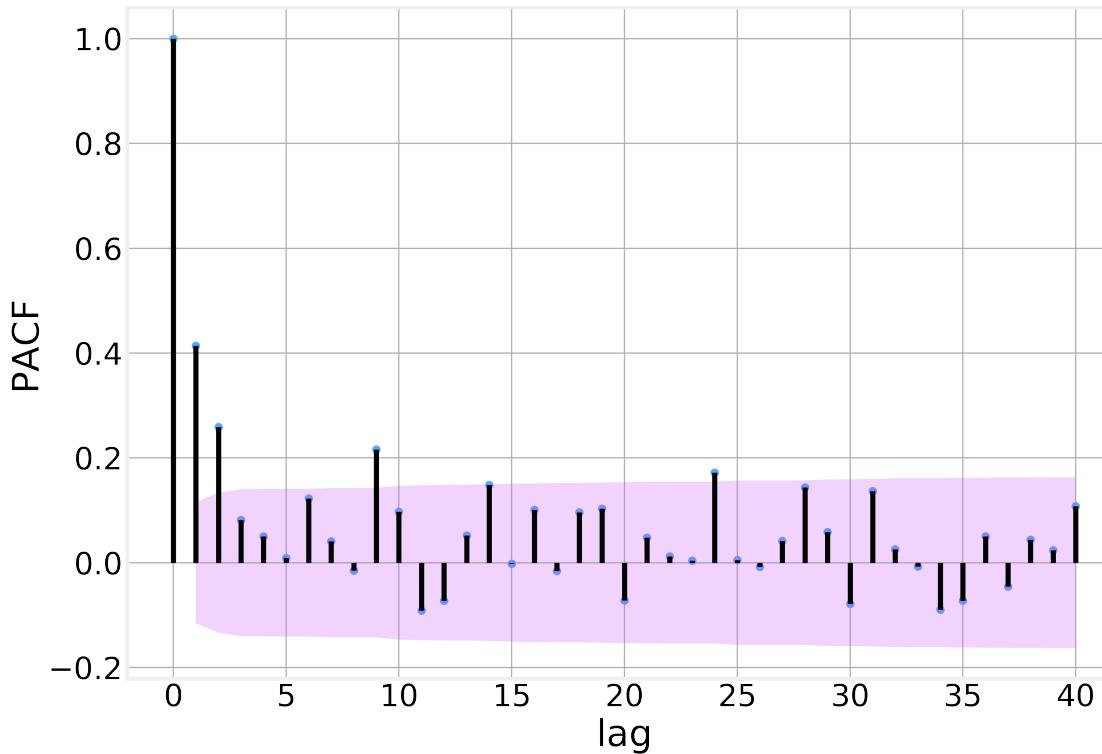
Partial Autocorrelation

- And then calculating the Pearson correlation function between the values and their residuals:

$$PACF_x(l) = \rho(x_t - \hat{x}_t, x_{t-l} - \hat{x}_{t-l})$$

- Confidence intervals can be computed using the same formula used for the ACF

Partial Autocorrelation





Code - Correlations

https://github.com/DataForScience/Timeseries_LL



Lesson 8 Random Walks

8.1 What is a Random Walk?

8.2 White Noise

8.3 Stationary vs Non-Stationary

8.4 Dickey-Fuller Test

8.5 Hurst Exponent



Lesson 8 Random Walks

8.1 What is a Random Walk?

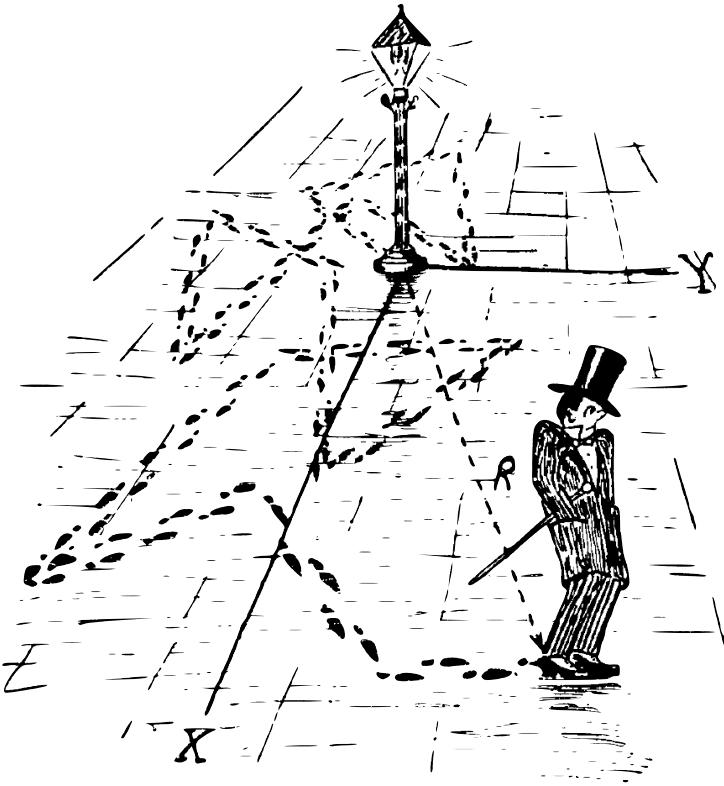
8.2 White Noise

8.3 Stationary vs Non-Stationary

8.4 Dickey-Fuller Test

8.5 Hurst Exponent

Random Walks



- At each step flip a coin
 - Heads: Move right
 - Tails: Move left
- If you start at position 0, do you ever reach position L?
- On average, we expect the position to be always close to 0.
- What if the coin is biased as in the previous example?

Random Walks

- Mathematically, we can describe the **position** of our random walker at time t as:

$$x_t = x_{t-1} + \epsilon_t$$

- Where ϵ_t is the **stochastic value** generated by our coin flip (+1 or -1)
- We can further write:

$$x_t = x_0 + \sum_i \epsilon_i$$

- which shows that the current position is just the **sum** across all **coin flips** in our walk.

Random Walks

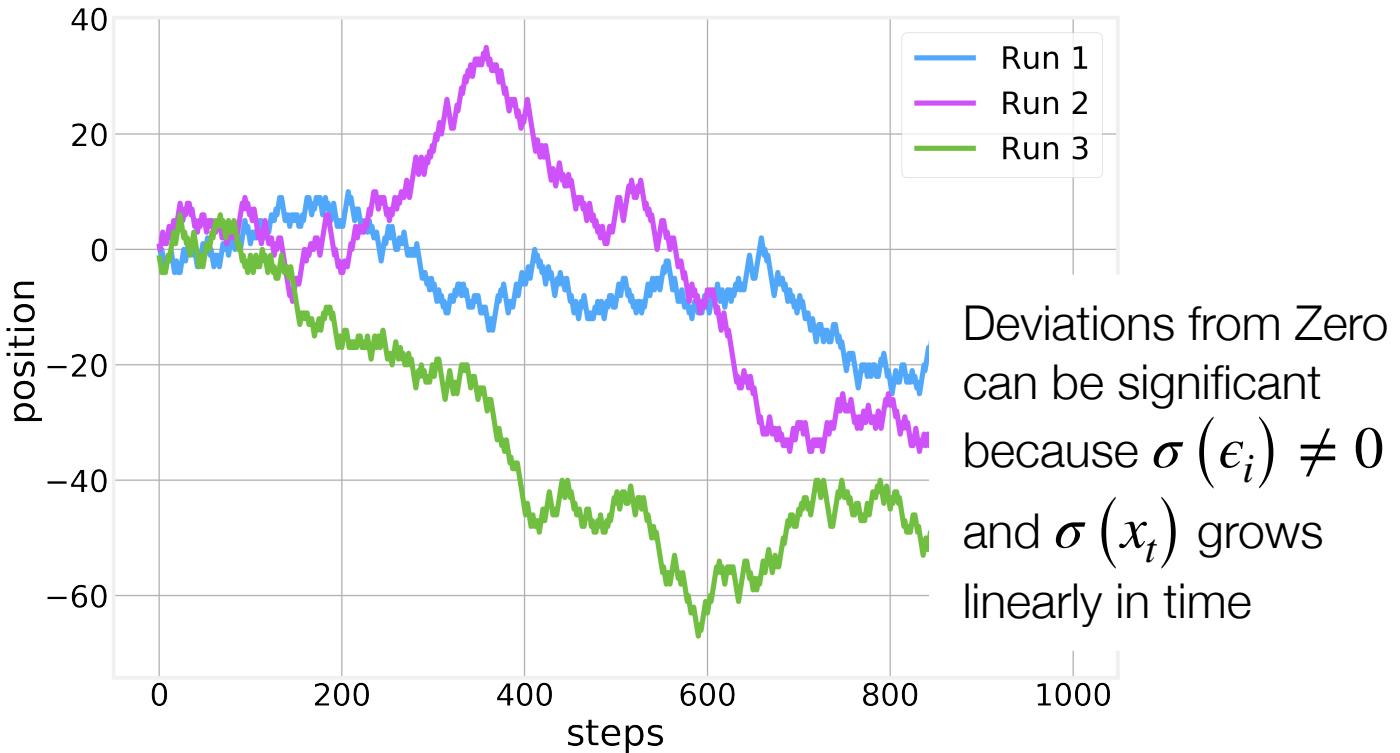
- Naturally, we can treat a random walk as a realization of a time series, but is it stationary?
- The mean position is:

$$\mu = \langle x_t \rangle = \langle x_0 \rangle + \sum_i \langle \epsilon_i \rangle$$

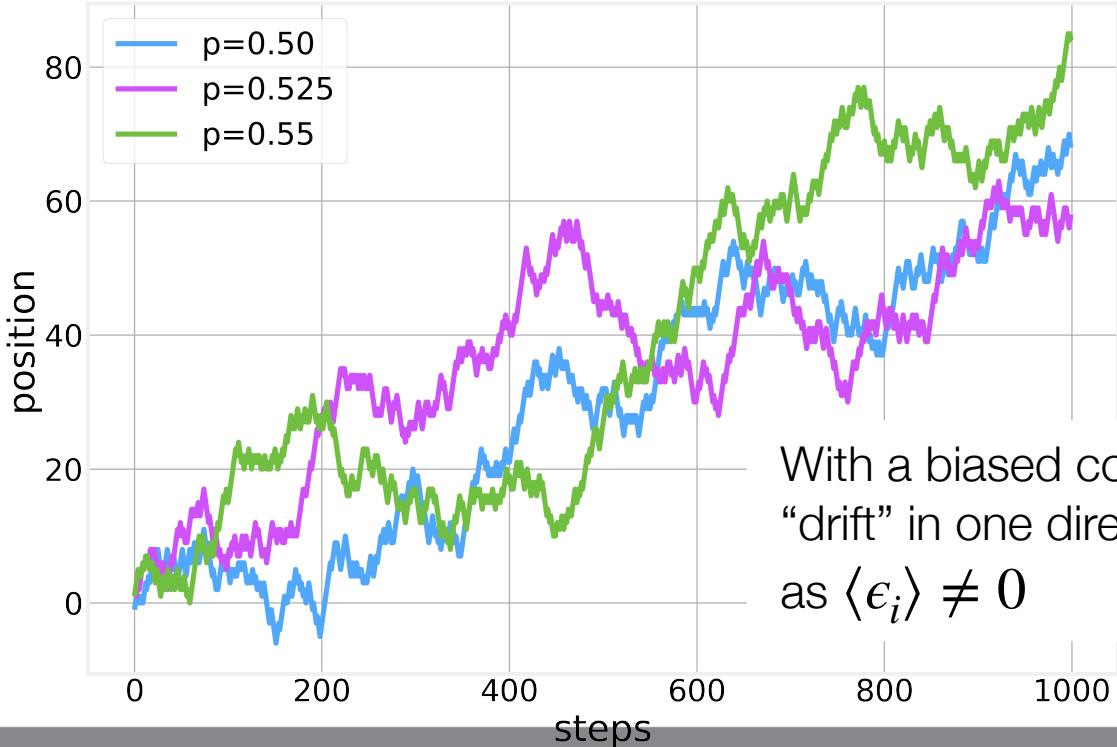
- If the coin is unbiased, $\langle \epsilon_i \rangle = 0$ and the mean is constant. On the other hand, the variance is:
- which is not constant. So even the simple random walk is not a stationary process.

$$\sigma = \sigma(x_t) = \sigma(x_0) + \sum_i \sigma(\epsilon_i) = \sigma(x_0) + t \cdot \sigma(\epsilon)$$

Random Walks



Random Walk with a drift





Lesson 8 Random Walks

8.1 What is a Random Walk?

8.2 White Noise

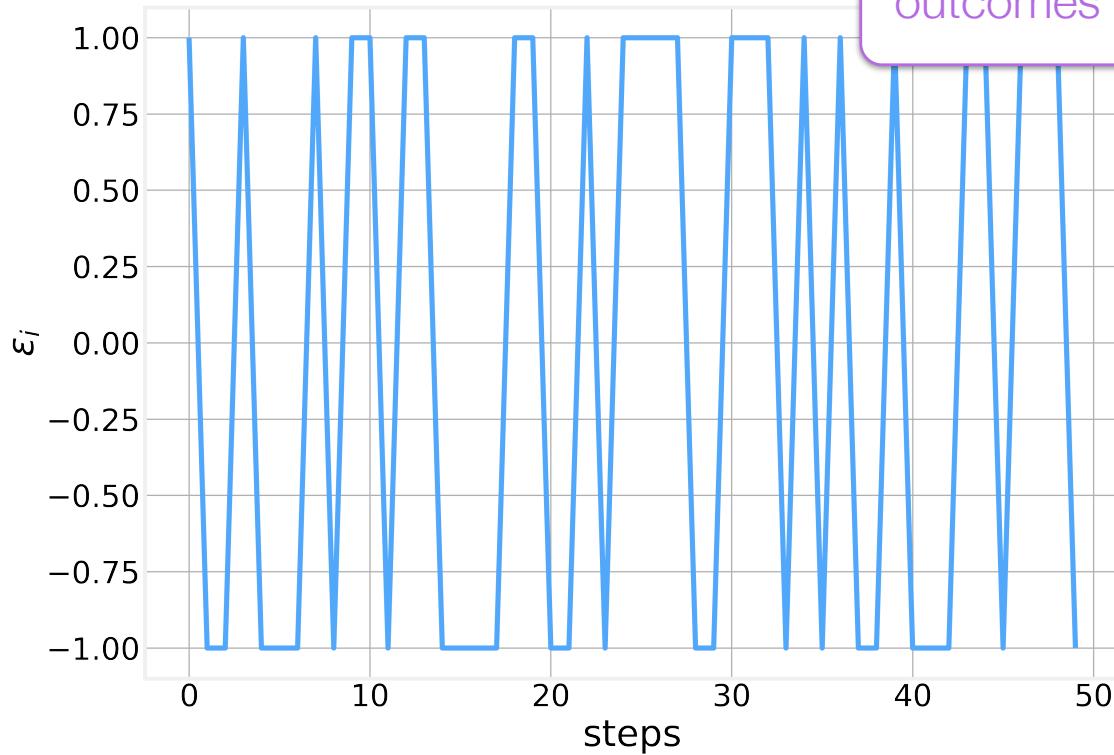
8.3 Stationary vs Non-Stationary

8.4 Dickey-Fuller Test

8.5 Hurst Exponent

White noise

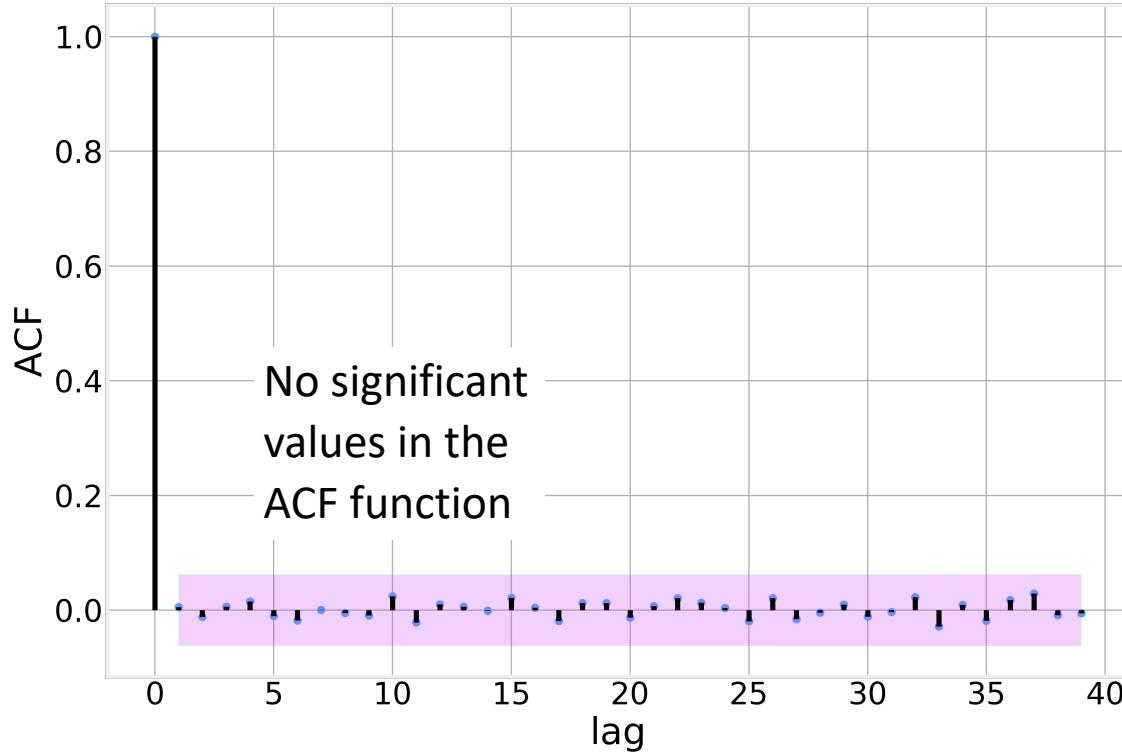
Let's take a deep look at our stochastic variables (the outcomes of the “coin flips”)



White noise

- White Noise is defined as:
 - a signal with equal amplitudes at all frequencies
 - a sequence of uncorrelated random variables (such as coin flips)
- Referred to as “noise” because it contains no information or correlations

White noise





Lesson 8 Random Walks

8.1 What is a Random Walk?

8.2 White Noise

8.3 Stationary vs Non-Stationary

8.4 Dickey-Fuller Test

8.5 Hurst Exponent

Stationary vs Non-Stationary

- We can also describe this process in the same mathematical framework as:

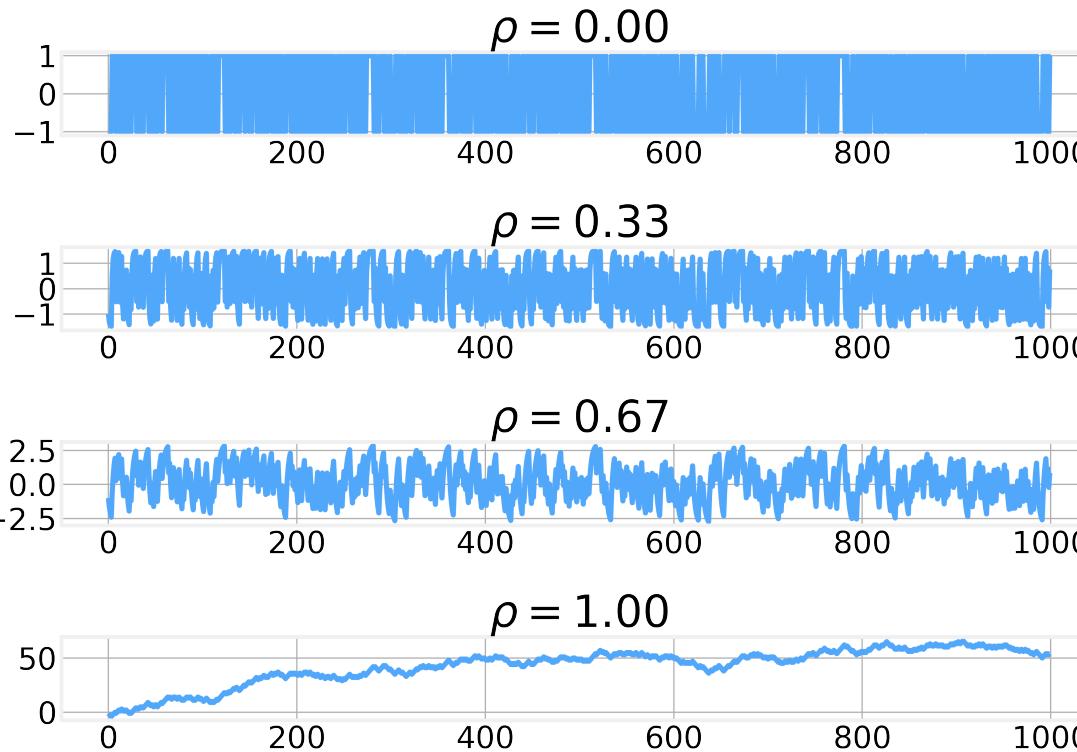
$$x_t = \epsilon_i$$

- which is clearly a stationary process.
- We can combine both expressions into a single one:

$$x_t = \rho x_{t-1} + \epsilon_i$$

- Where ρ gives us a “knob” to interpolate between the two extremes, between a stationary and a non-stationary process.

Stationary vs Non-Stationary





Lesson 8 Random Walks

8.1 What is a Random Walk?

8.2 White Noise

8.3 Stationary vs Non-Stationary

8.4 Dickey-Fuller Test

8.5 Hurst Exponent

Dickey-Fuller Test

- The Dickey-Fuller Test is a test of stationarity inspired by a simple: Can we show that

$$\rho \neq 1$$

- with some degree of certainty?
- Numerically, we can express this as a linear regression fit

$$x_t - x_{t-1} = \gamma x_{t-1} + \epsilon_t$$

- where $\gamma = \rho - 1$
- The slope of the regression is then our expected value for $\rho - 1$.
- If the process is non-stationary then we expect $\gamma \neq 0$.

Dickey-Fuller Test

- From the residuals of γ we compute the Dickey-Fuller statistic:

$$DF = \frac{\hat{\gamma}}{SE(\gamma)}$$

- The value of this statistic is then compared with a critical values table.
- In general, the more negative it is, the more certain we can be that we can **reject the null hypothesis**
- The Dickey-Fuller Test has many variants.
- The most common one is the known as the **Augmented-Dickey-Fuller** test and is able to account for multiple lags, trends, etc.

Critical values for Dickey-Fuller <i>t</i> -distribution.				
	Without trend		With trend	
Sample size	1%	5%	1%	5%
T = 25	-3.75	-3.00	-4.38	-3.60
T = 50	-3.58	-2.93	-4.15	-3.50
T = 100	-3.51	-2.89	-4.04	-3.45
T = 250	-3.46	-2.88	-3.99	-3.43
T = 500	-3.44	-2.87	-3.98	-3.42
T = ∞	-3.43	-2.86	-3.96	-3.41

Source [2]:373



Lesson 8 Random Walks

8.1 What is a Random Walk?

8.2 White Noise

8.3 Stationary vs Non-Stationary

8.4 Dickey-Fuller Test

8.5 Hurst Exponent

Hurst Exponent

- The Hurst Exponent is another metric that allows us to determine whether or not a time series is stationary.
- It measures the “speed of diffusion” defined as:

$$Var(\tau) = \langle |z(t + \tau) - z(t)|^2 \rangle \sim \tau^{2H}$$

- where H is the **Hurst exponent**
 - $H < 0.5$ - mean reverting series
 - $H = 0.5$ - geometric random walk
 - $H > 0.5$ - trending series
- Smaller values indicate stronger levels of **mean reversion**, while larger values



Code - Random Walks

https://github.com/DataForScience/Timeseries_LL



Lesson 9 ARIMA Models

9.1 Moving Average (MA) Models

9.2 Autoregressive (AR) Models

9.3 ARIMA Model

9.4 Fitting ARIMA Models

9.5 statsmodels for ARIMA Models

9.6 Seasonal ARIMA



Lesson 9 ARIMA Models

9.1 Moving Average (MA) Models

9.2 Autoregressive (AR) Models

9.3 ARIMA Model

9.4 Fitting ARIMA Models

9.5 statsmodels for ARIMA Models

9.6 Seasonal ARIMA

Moving Average (MA) Model

- We start our exploration of the ARIMA family of models by considering the [Moving Average](#) model.
- The simplest moving average model can be written as:

$$x_t = \epsilon_t$$

- Which we already saw in our discussion of random walks.
- A more general case can be written as:

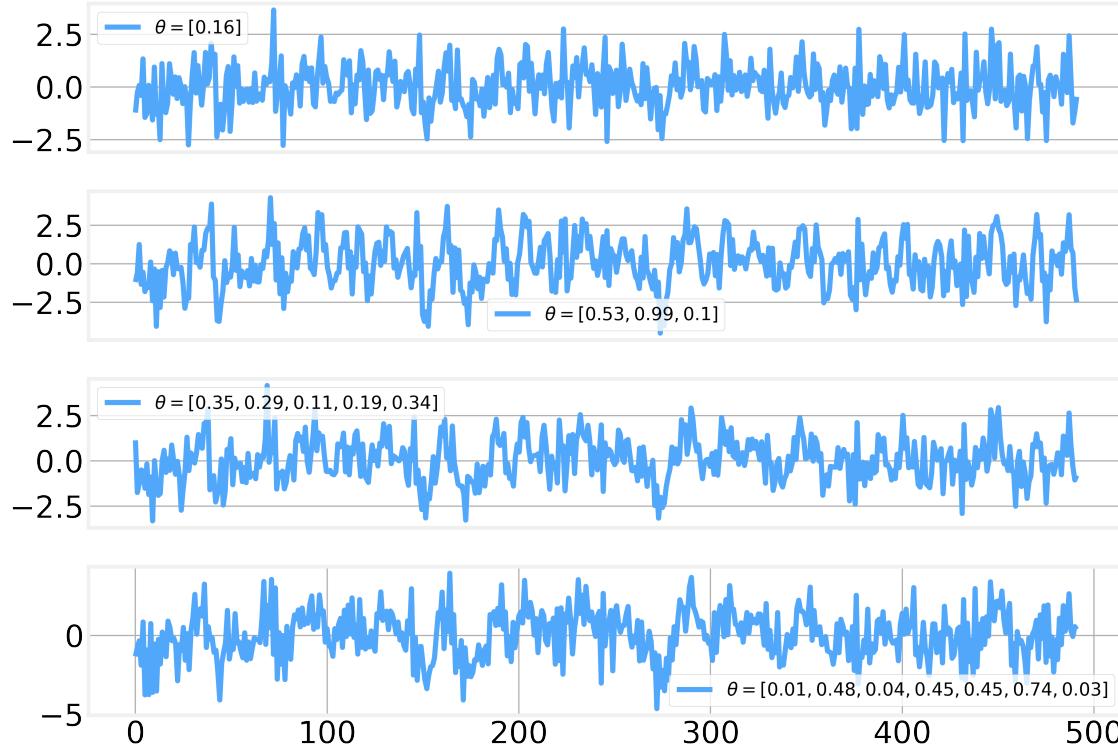
$$x_t = \beta + \sum_{l=0}^q \theta_l \epsilon_{t-l}$$

- where β is a constant offset, θ_l are the weights for the values at lag l and q is the moving window size. $\theta_0 \equiv 1$.

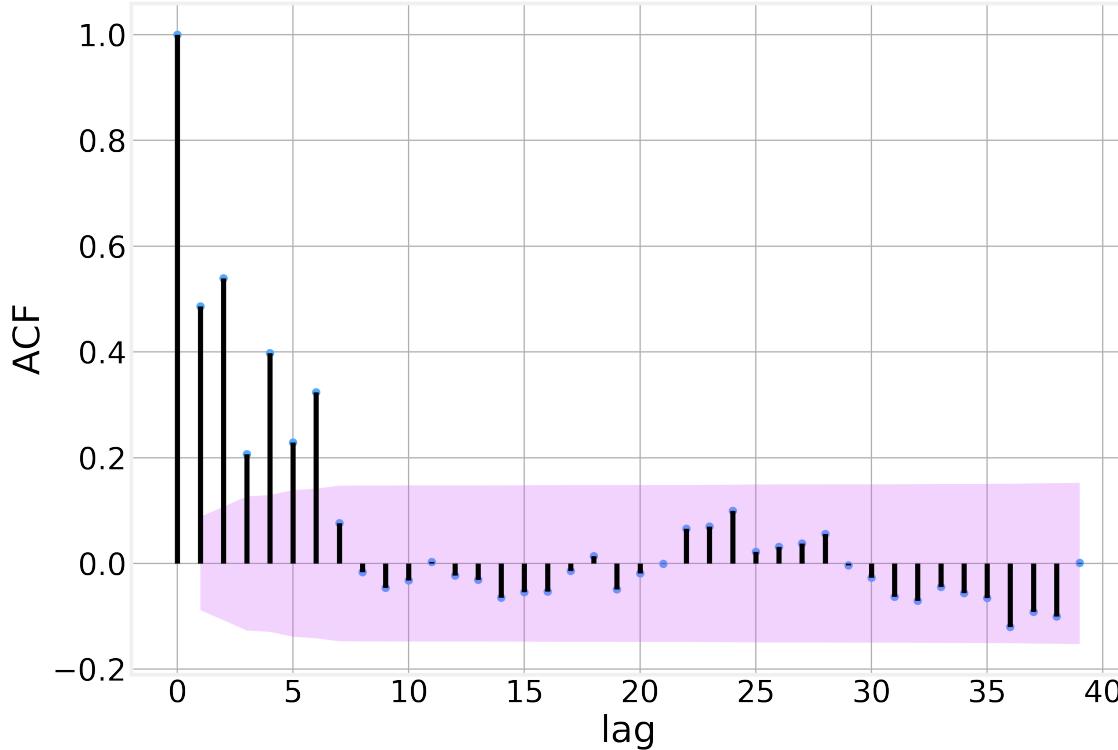
Moving Average (MA) Model

- The ϵ_t values are stochastic variables (often referred to as “errors”) rather than the actual observed values x_t
- The generated x_t is uncorrelated with itself for any lag $l > q$

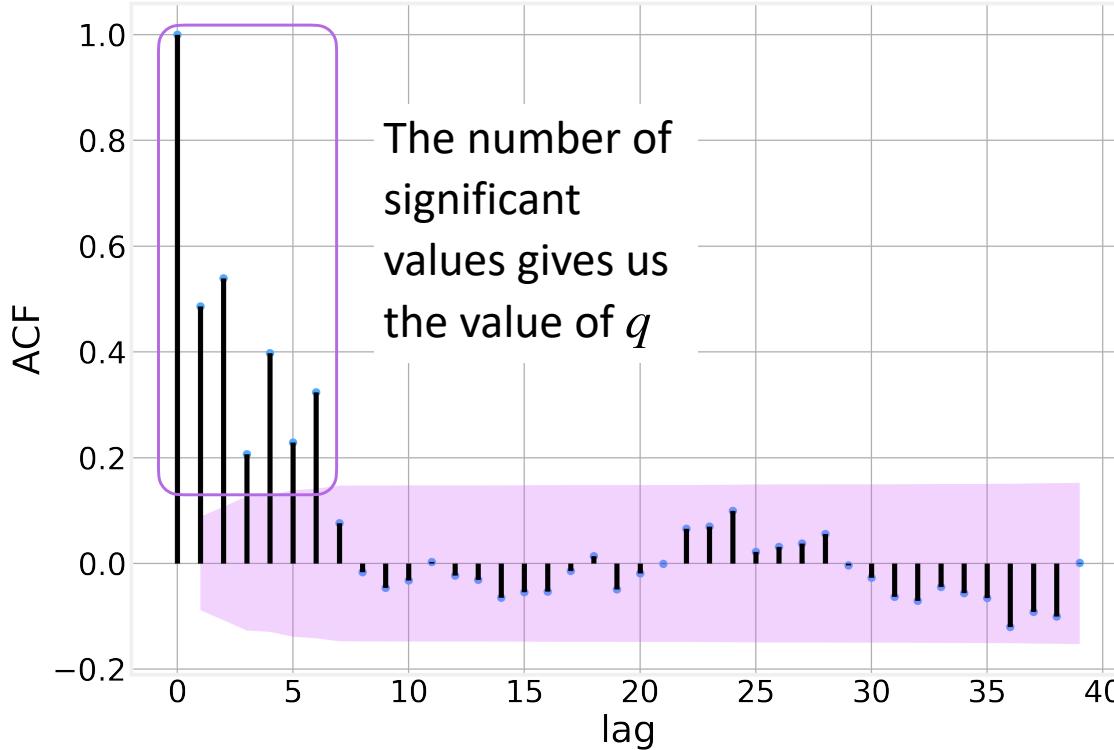
Moving Average (MA) Model



Moving Average (MA) Model



Moving Average (MA) Model





Lesson 9 ARIMA Models

9.1 Moving Average (MA) Models

9.2 Autoregressive (AR) Models

9.3 ARIMA Model

9.4 Fitting ARIMA Models

9.5 statsmodels for ARIMA Models

9.6 Seasonal ARIMA

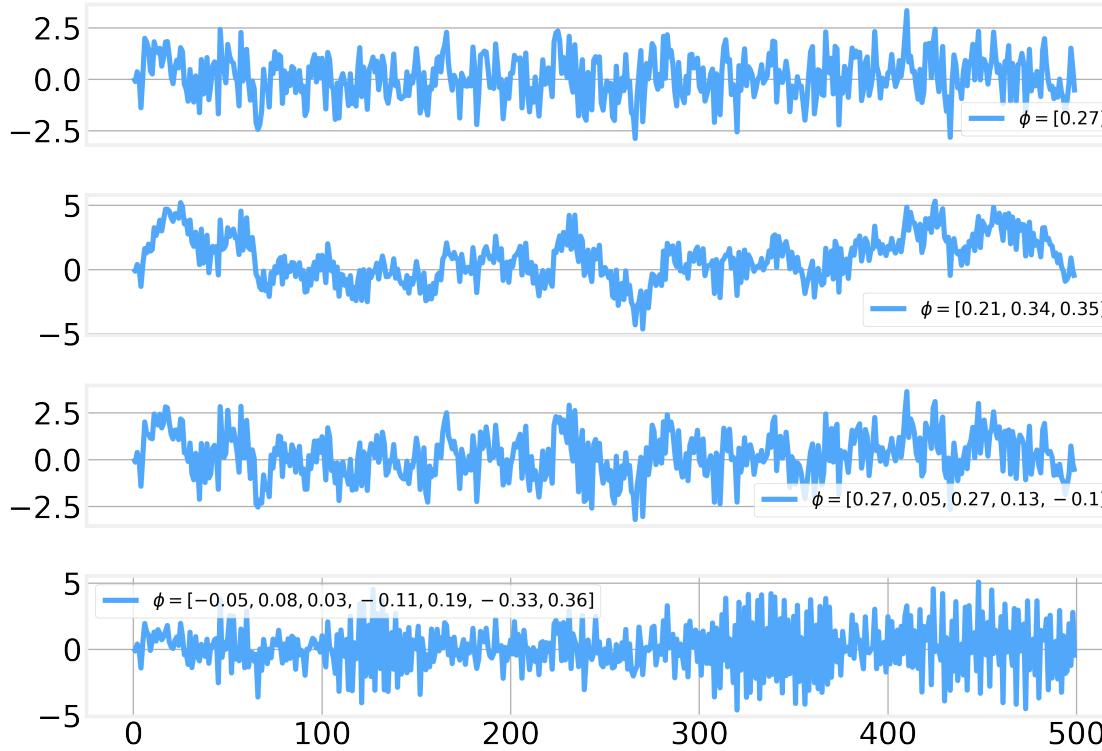
Auto-Regressive (AR) Models

- Auto-Regressive models rely on the fact that in stationary models any deviation from the mean must be compensated. The series must “revert to the mean”.
- AR models can be defined as:

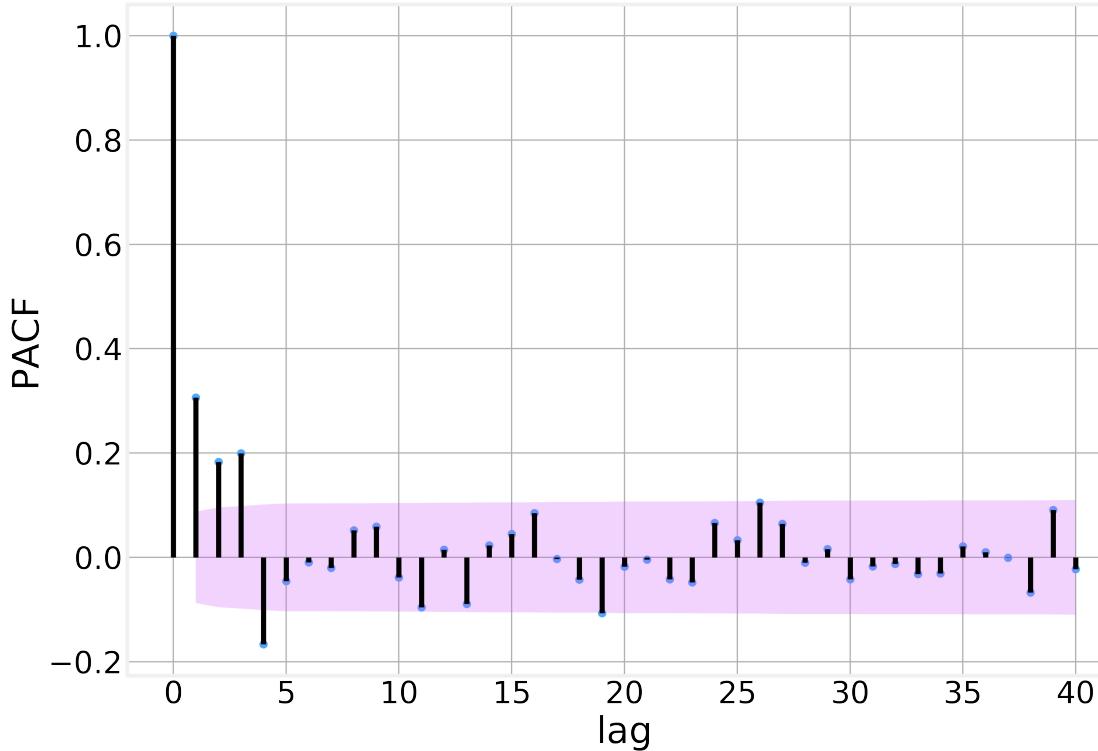
$$x_t = \alpha + \epsilon_t \sum_{l=1}^p \phi_l x_{t-l}$$

- Where the constant c represents the process' average value and the x_{t-l} are the observed values at a given lag l and ϕ_l are the corresponding weights.

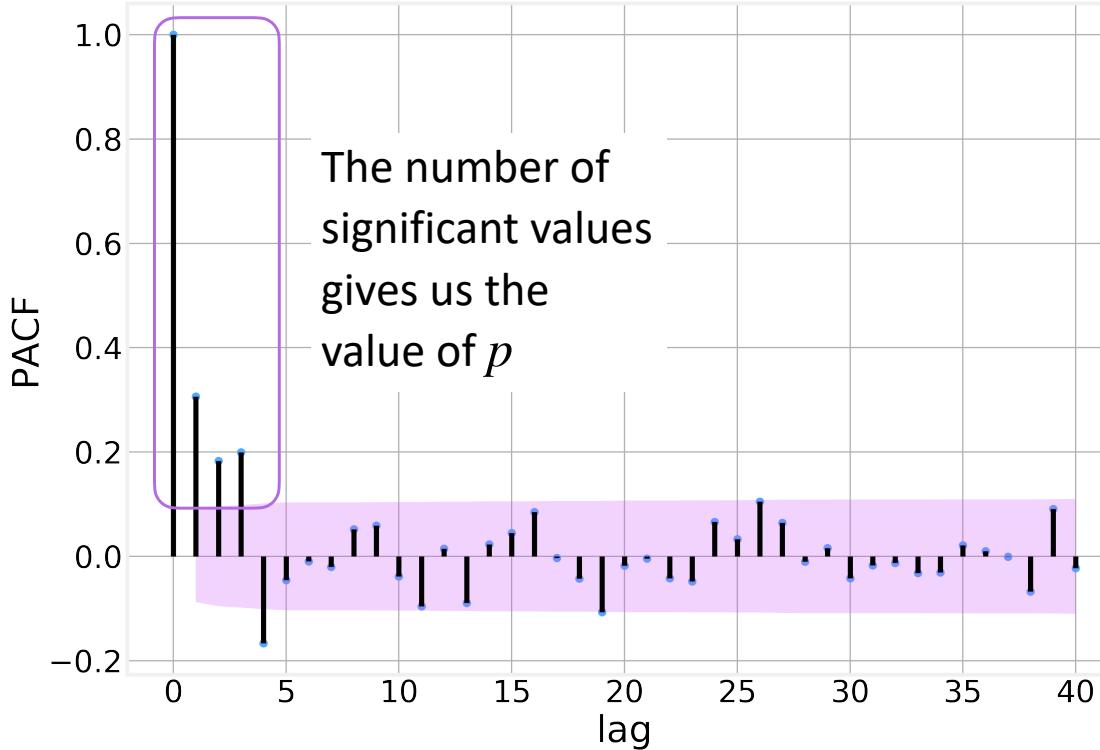
Auto-Regressive (AR) Models



Auto-Regressive (AR) Models



Auto-Regressive (AR) Models





Lesson 9 ARIMA Models

9.1 Moving Average (MA) Models

9.2 Autoregressive (AR) Models

9.3 ARIMA Model

9.4 Fitting ARIMA Models

9.5 statsmodels for ARIMA Models

9.6 Seasonal ARIMA

Integrative (I) “model”

- We already saw that we can take differences to “stationarize” the time series.
- To recover the original values, we must then integrate
- While not a model by itself, it is often an important first step in modeling time series

ARIMA model

- The three classes of models we described above can be integrated into a single model:

Auto

$$x_t = c + \sum_i^p \phi_i x_{t-i} + \epsilon_t$$

Regressive

Integrated

Moving

$$x_t = \mu + \epsilon_t + \sum_i^q \theta_i \epsilon_i$$

Average

ARIMA model

- The complete model can be written as:

$$\hat{x}_t = c + \mu + \sum_i^p \phi_i x_{t-i} + \sum_j^q \theta_j \epsilon_{t-j} + \epsilon_t$$

- where \hat{x}_t is the properly differentiated time series

ARIMA model

- From this simple definition we can easily recover several interesting special cases:
- *ARIMA (0,1,0)* - Random Walk (with or without drift)
 - $x_t - x_{t-1} = c + \epsilon_t$
- *ARIMA (0,0,0)* - White noise (the sequence of stochastic variables)
 - $x_t = \epsilon_t$
- *ARIMA (0,1,1)* - Exponential Smoothing
 - $x_t - x_{t-1} = \epsilon_t + \theta_1 \epsilon_{t-1}$
- *ARIMA (0,2,2)* - Double exponential Smoothing
 - $x_t - 2x_{t-1} + x_{t-2} = \epsilon_t + \theta_1 \epsilon_{t-1} + \theta_2 \epsilon_{t-2}$



Lesson 9 ARIMA Models

9.1 Moving Average (MA) Models

9.2 Autoregressive (AR) Models

9.3 ARIMA Models

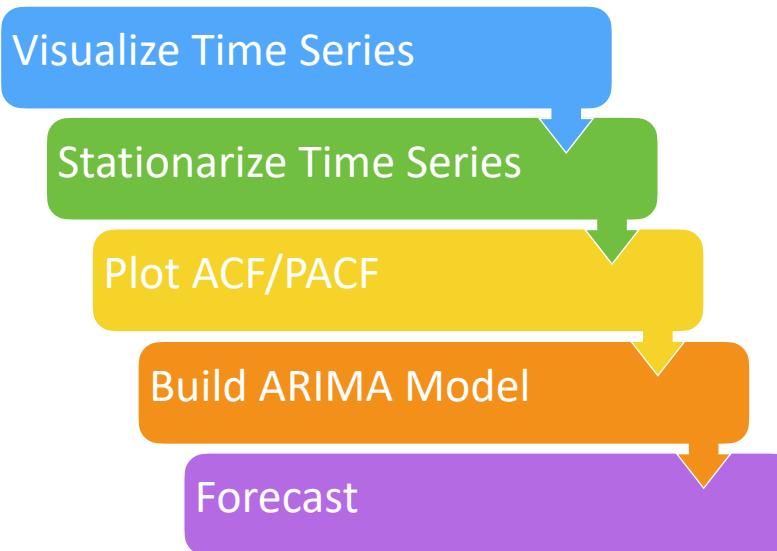
9.4 Fitting ARIMA Models

9.5 statsmodels for ARIMA Models

9.6 Seasonal ARIMA

Fitting ARIMA models

The general procedure to fit an ARIMA model was originally proposal by Box-Jenkins



- d - degree of differencing
- p - number of lag observations included in the model (*PACF*)
- q - size of the moving average window (*ACF*)

Fitting ARIMA models

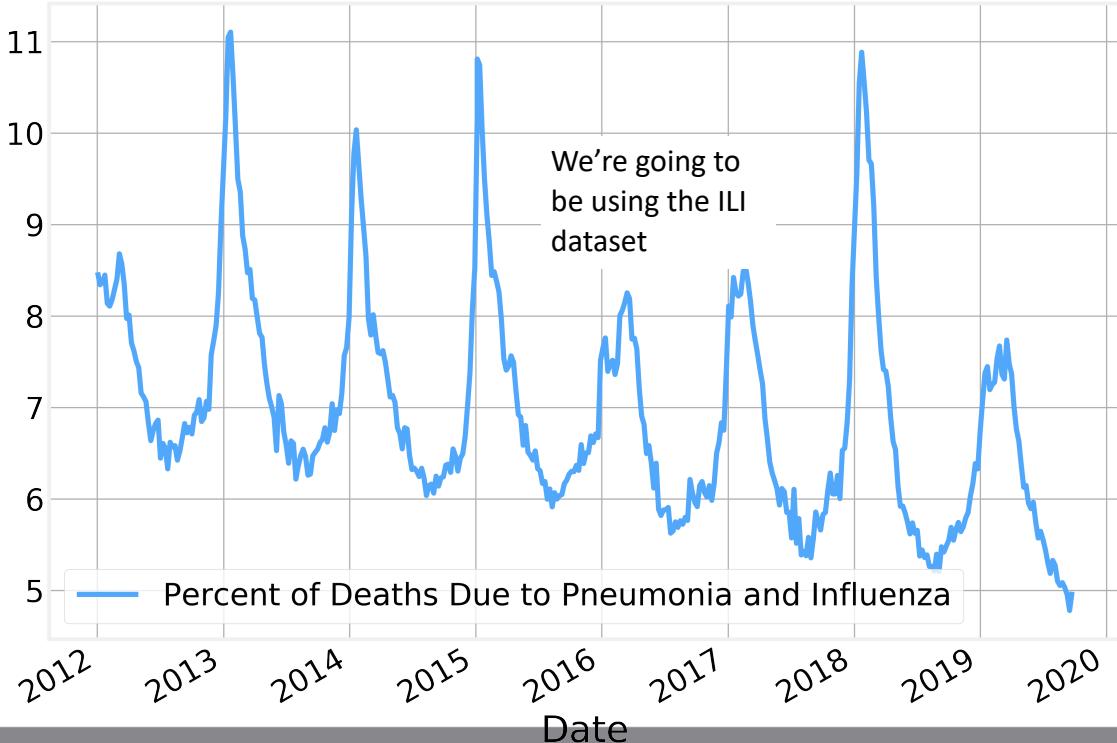
The general procedure to fit an ARIMA model was originally proposal by Box-Jenkins



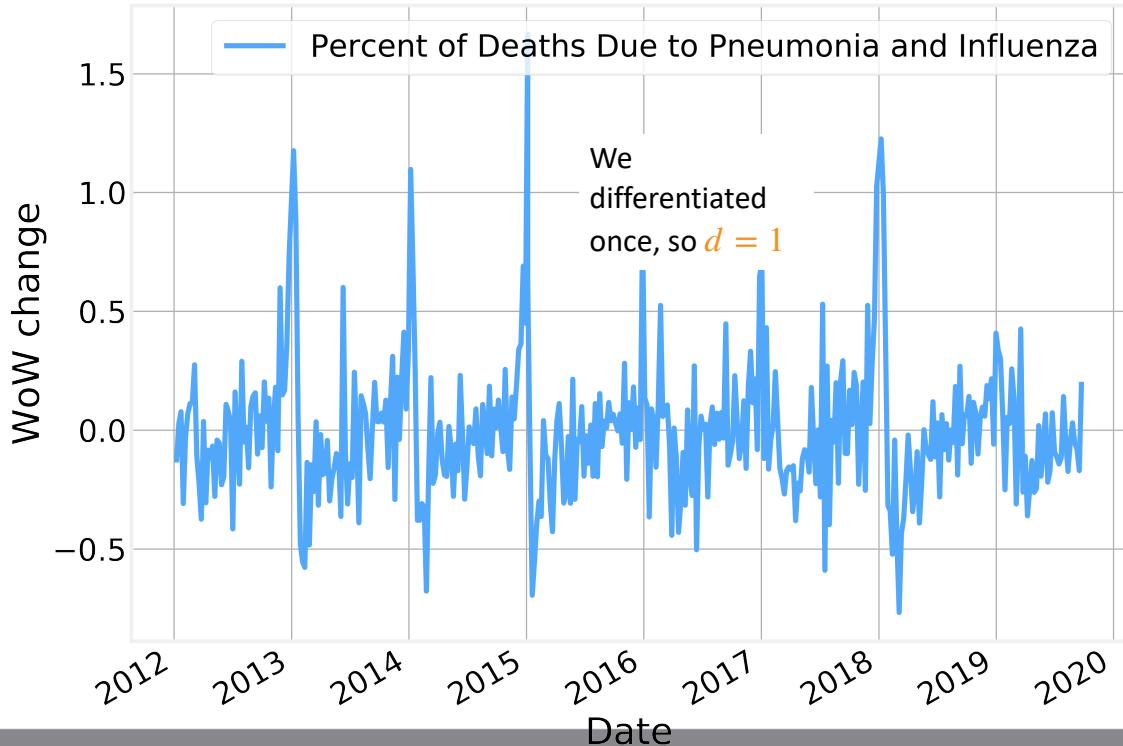
- d - degree of differencing
- p - number of lag observations included in the model (*PACF*)
- q - size of the moving average window (*ACF*)

Let us consider an example

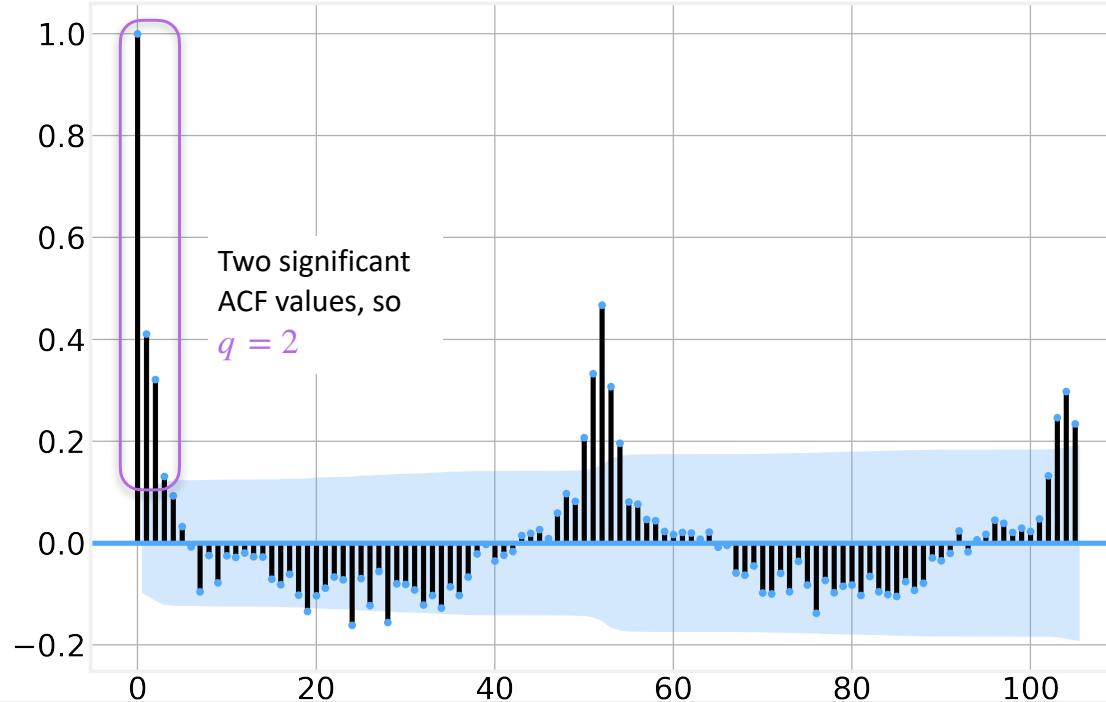
Visualize Time Series

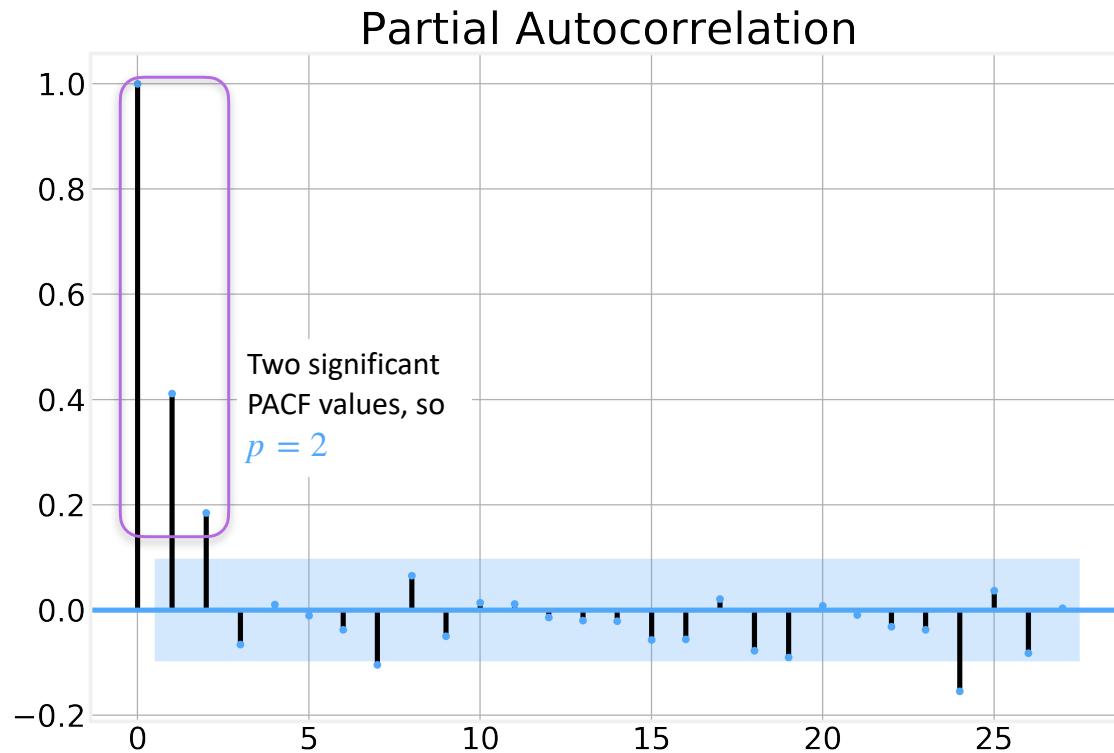


Stationarize Time Series

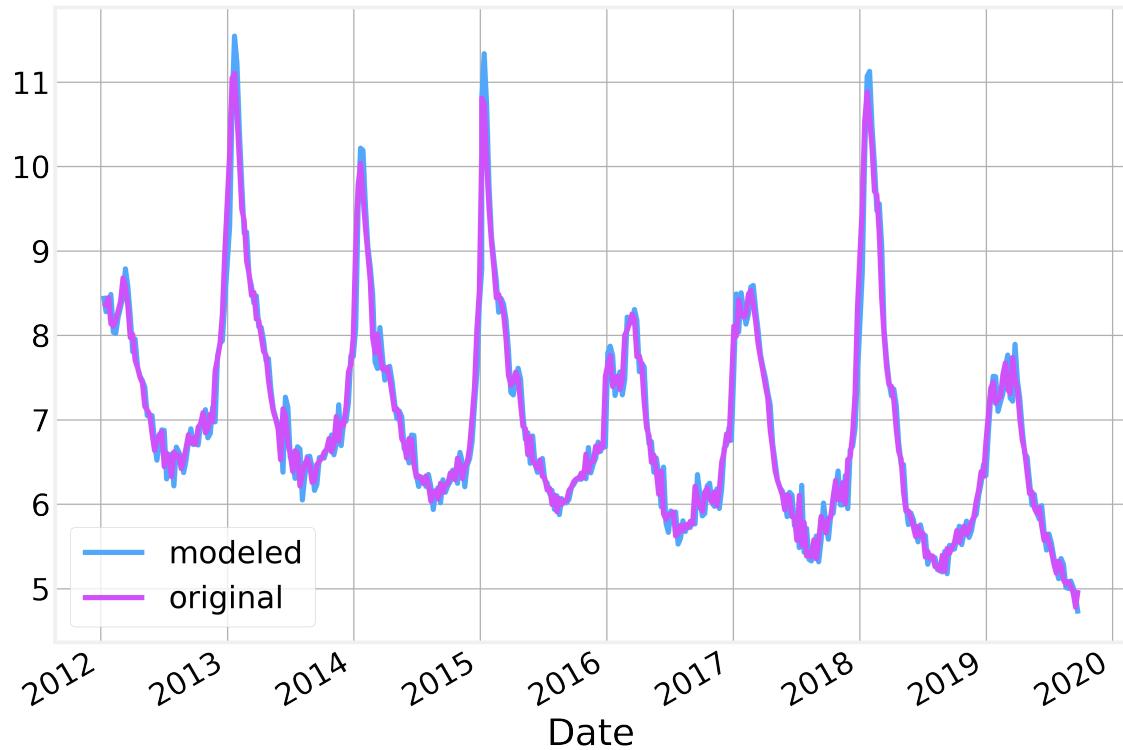


Autocorrelation

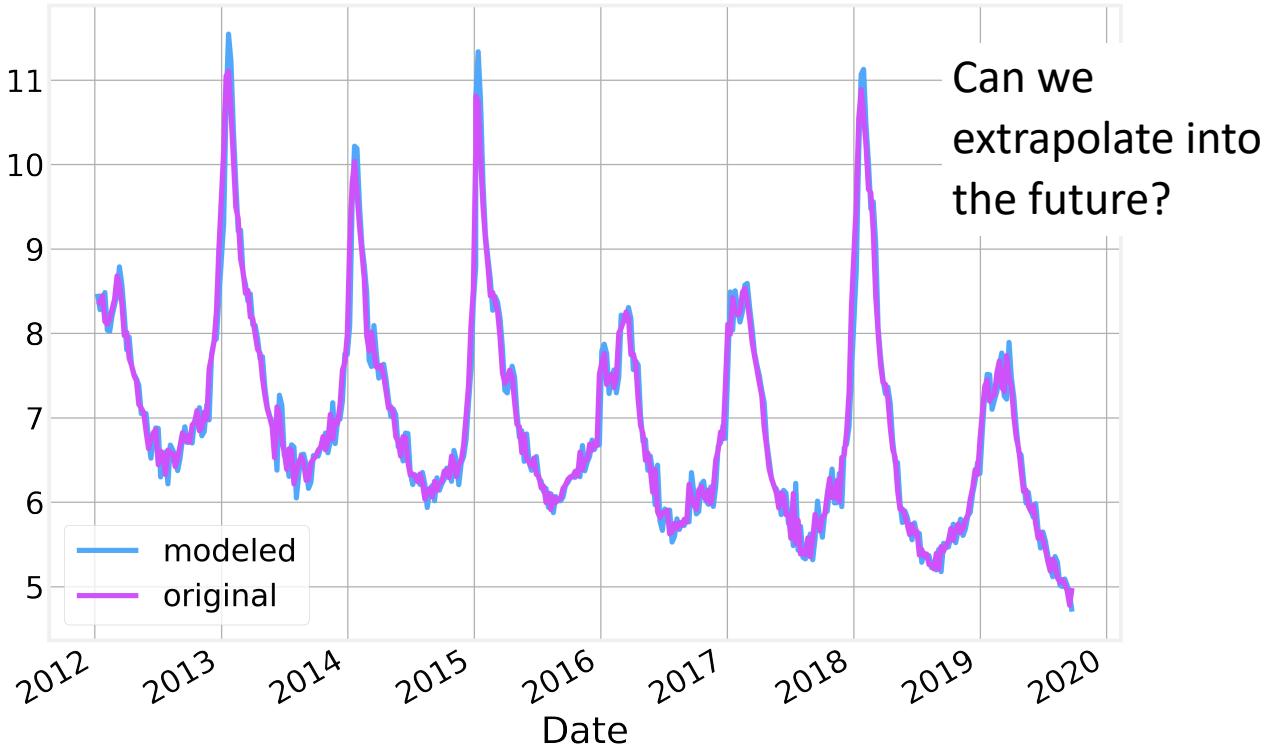




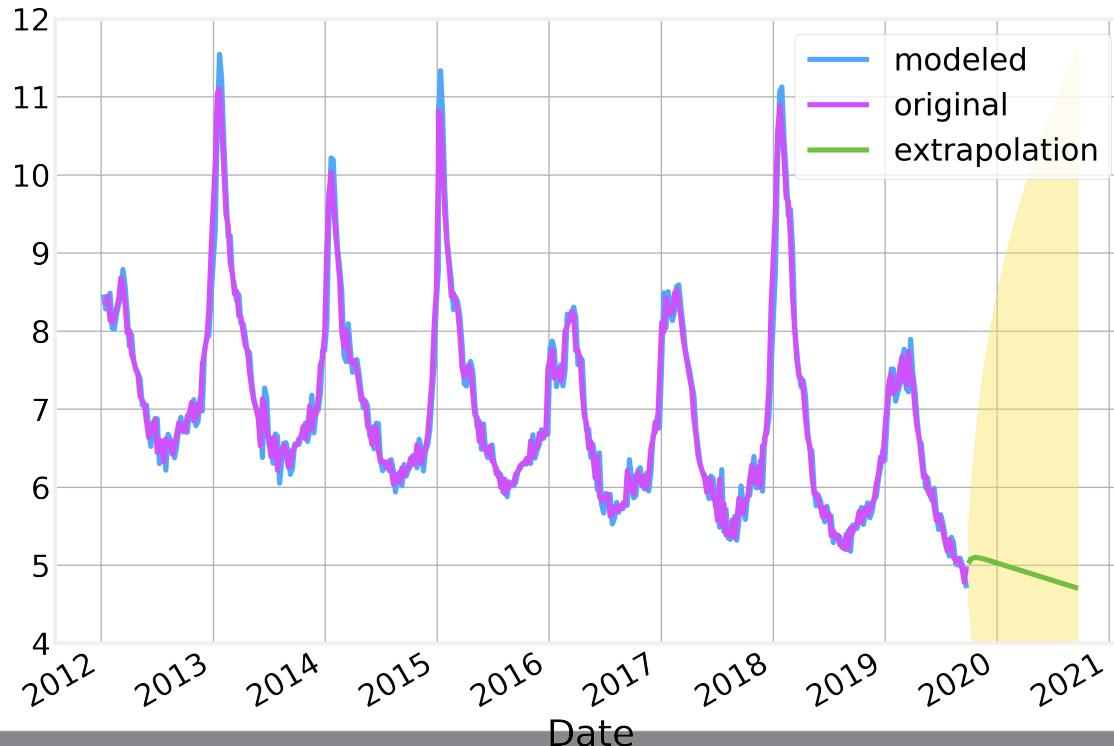
Forecast



Forecast

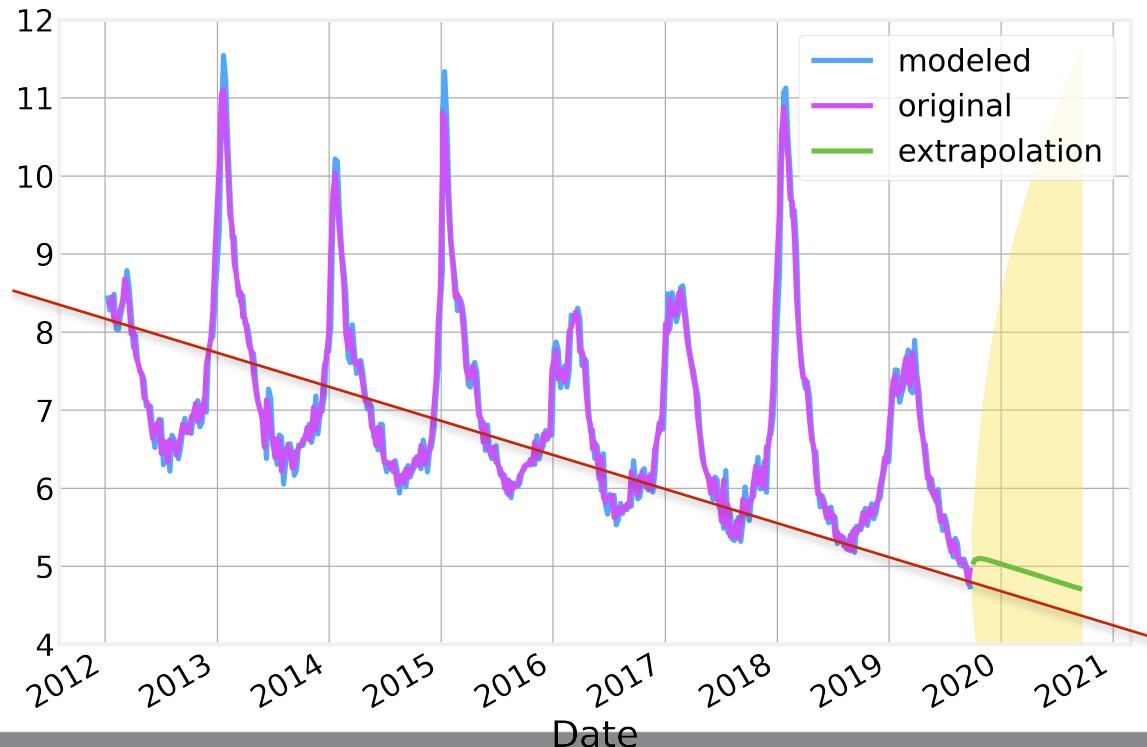


Forecast



After a few time steps the forecast simply regresses to the trend line

Forecast





Lesson 9 ARIMA Models

9.1 Moving Average (MA) Models

9.2 Autoregressive (AR) Models

9.3 ARIMA Models

9.4 Fitting ARIMA Models

9.5 statsmodels for ARIMA Models

9.6 Seasonal ARIMA

- Python module built on top of `numpy` and `scipy`
- Provides classes and functions for:
 - estimations of various statistical models
 - statistical tests
 - data exploration
- Extensive statistics are available for each estimate produced

- Particularly useful for:
 - Linear Regression
 - Generalized Linear Models
 - Survival Analysis
 - Time Series!

- Conventionally imported as:

```
import statsmodels.api as sm
```

- Supports R-style formula notation for statistical models
- For R-style formula notation support, conventionally imported as:

```
import statsmodels.formula.api as smf
```

- And for timeseries:

```
import statsmodels.api as sm
```

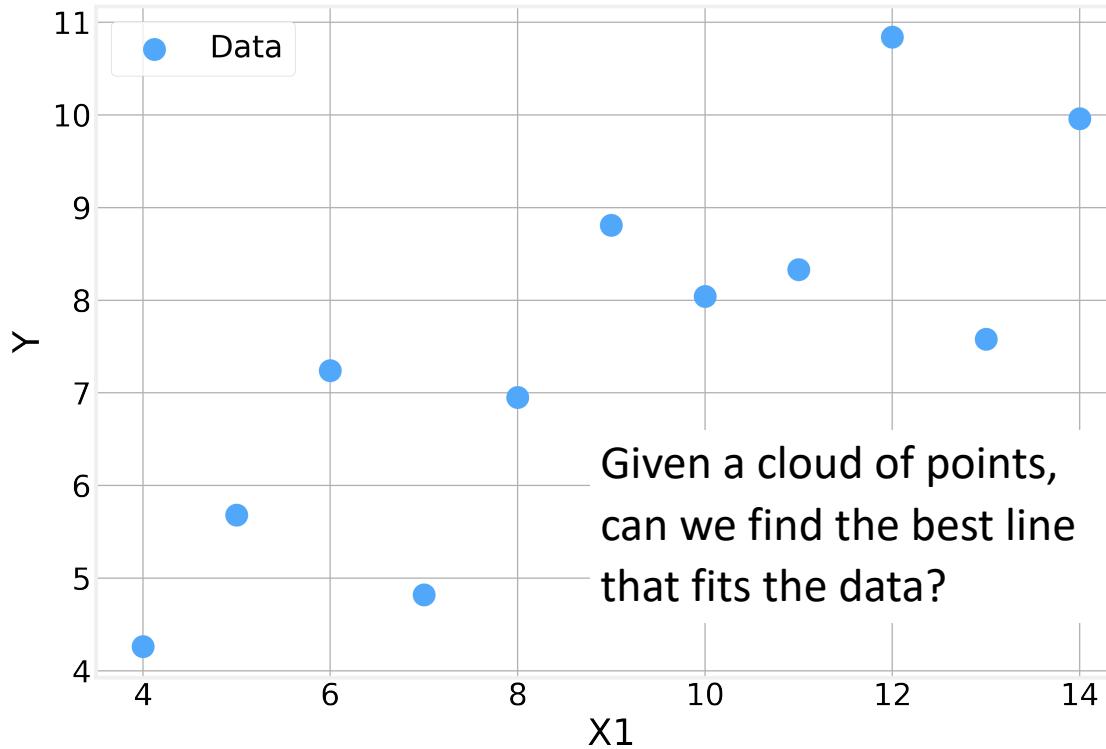
statsmodels.	api.	OLS()	.fit() .predict() .summary()		
		tsa.	seasonal_decompose()		
		stattools.		acf() pacf() adfuller()	
		arima.		ARIMA()	.fit() .predict() .forecast() .plot_predict() .summary()
		statespace.		SARIMAX()	.fit() .predict() .forecast() .summary()
	graphics.	tsa.		plot_acf() plot_pacf()	

The way the library is organized isn't always the most intuitive

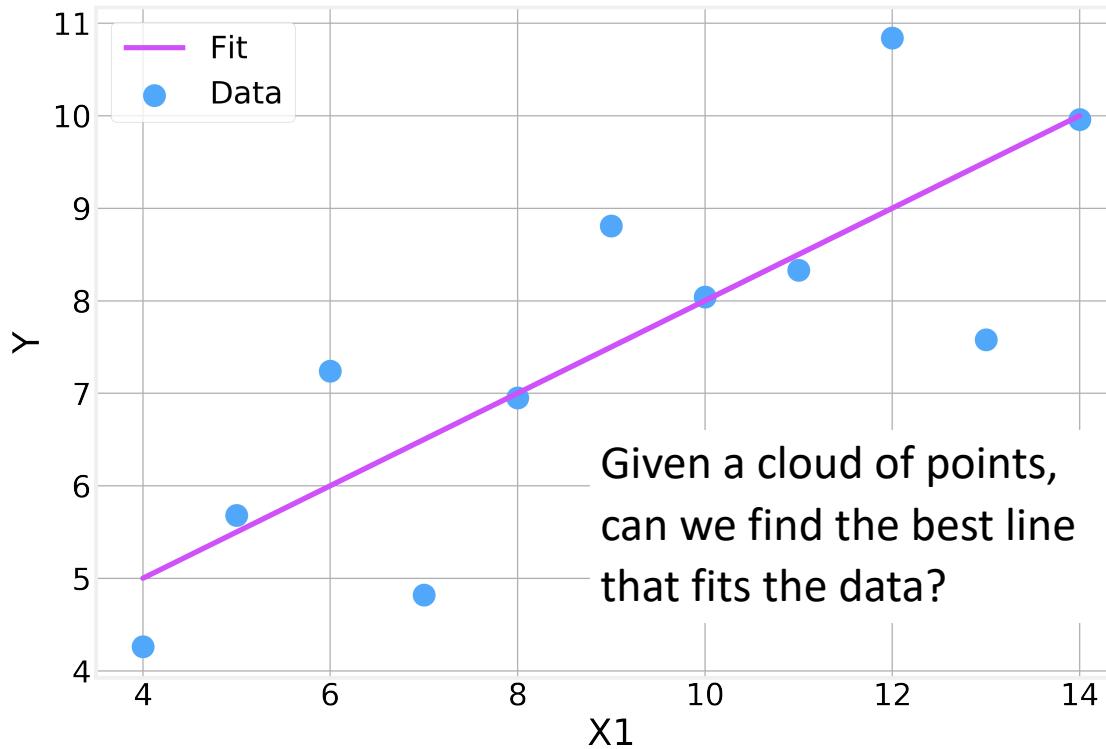
- statsmodels refers to variables as:
 - “**endogenous**” - caused by factors within the system
 - “**exogenous**” - caused by factors outside the system

endog	exog
y	x
y variable	x variable
left hand side (LHS)	right hand side (RHS)
dependent variable	independent variable
regressand	regressors
outcome	design
response variable	explanatory variable

Linear Regression



Linear Regression



Linear Regression

- `statsmodels` returns:
 - a description of the model and fitting procedure,
 - the `coefficients` of the fitted function,
 - estimates of the `error`,
 - `statistical significance`,
 - and a wealth of statistical test results.

OLS Regression Results						
Dep. Variable:	y	R-squared:	0.667			
Model:	OLS	Adj. R-squared:	0.629			
Method:	Least Squares	F-statistic:	17.99			
Date:	Sun, 09 Aug 2020	Prob (F-statistic):	0.00217			
Time:	12:37:16	Log-Likelihood:	-16.841			
No. Observations:	11	AIC:	37.68			
Df Residuals:	9	BIC:	38.48			
Df Model:	1					
Covariance Type:	nonrobust					
	coef	std err	t	P> t	[0.025	0.975]
const	3.0001	1.125	2.667	0.026	0.456	5.544
x1	0.5001	0.118	4.241	0.002	0.233	0.767
Omnibus:	0.082	Durbin-Watson:	3.212			
Prob(Omnibus):	0.960	Jarque-Bera (JB):	0.289			
Skew:	-0.122	Prob(JB):	0.865			
Kurtosis:	2.244	Cond. No.	29.1			

Auto-correlation

<https://www.statsmodels.org/devel/generated/statsmodels.tsa.stattools.acf.html>
https://www.statsmodels.org/devel/generated/statsmodels.graphics.tsaplots.plot_acf.html

- statsmodels provides us with the `acf()` and `plot_acf()` methods in the `tsa.stattools` and the `graphics.tsa` modules, respectively.
- `acf()` calculates the values of the auto-correlation function up to `nlags` (default=40)
- `plot_acf()` calculates the auto-correlation function and the respective confidence interval up to `lags` (default=the number of significant auto-correlation values)
- Both of these methods include various optional arguments to help customize their behavior.

Partial Autocorrelation

<https://www.statsmodels.org/devel/generated/statsmodels.tsa.stattools.pacf.html>

https://www.statsmodels.org/devel/generated/statsmodels.graphics.tsaplots.plot_pacf.html

- `statsmodels` uses a similar structure for the partial auto-correlation function with a `pacf()` and `plot_pacf()` functions provided by the `tsa.statstools` and the `graphics.tsa` modules, respectively.
- `pacf()` calculates the values of the partial auto-correlation function up to `nlags` (default=40)
- `plot_acf()` calculates the auto-correlation function and the respective confidence interval up to `lags` (default=the number of significant auto-correlation values)
- And as with the `acf/plot_acf` duo, both of these methods include various optional arguments to help customize their behavior.

Dickey-Fuller Test

- `statsmodels` provides an implementation of the Augmented Dickey-Fuller test with the function `adftest()` in the `tsa.stattools` module
- `adftest()` is able to automatically handle multiple lags, adjust for trends, etc.
- `adftest()`, when using the parameter `regresults=True`, returns a tuple with 4 elements:
 - `adfstat` - the value of the augmented dickey fuller statistic
 - `p-value` - the p-value (statistical significance) of the result
 - `criticalvalues` - the critical values for the test statistic at the 1%, 5%, and 10% levels
 - `results` - an object containing more information, including the results for the linear regression
- We encourage you to explore the various arguments to this function to fully understand its functionality.

Time series decomposition

- statsmodels provides the `seasonal_decompose()` function in the `tsa.seasonal` module to perform the seasonal decomposition described above.
- The `model` argument allows us to select between “additive” or “multiplicative” decomposition.
- The `two_sided` argument lets us decide how to calculate the moving average. We should set it to `False` to make sure the moving average values are computed only past values.
- It returns an object with `observed`, `seasonal`, `trend`, and `resid` attributes as `pandas` DataFrames. The `plot()` method generates the usual decomposition plot.

ARIMA model

- `statsmodels` provides the `ARIMA` object in the `tsa.arima` module
- This object allows us to fit a specific model (specified by the argument `order=(p, q, d)`) to a given dataset, i.e. find the specific values of θ_j and ϕ_i
- The procedure to fit the model is similar to that of `sklearn`, relying two specific methods:
 - `fit()` - Determine the parameters of the model
 - `predict()` - “Run” the model to determine the predicted values of the “position” x_t
- Plus our friendly `summary()` to summarize the results and `plot_predict()` to compare the original data with the fitted model.



Lesson 9 ARIMA Models

9.1 Moving Average (MA) Models

9.2 Autoregressive (AR) Models

9.3 ARIMA Models

9.4 Fitting ARIMA Models

9.5 statsmodels for ARIMA Models

9.6 Seasonal ARIMA

Seasonal Models

- The problem with our simple ARIMA approach is that our model isn't taking into account the full seasonal structure of the data.
- A simple approach to taking seasonal variations into account is to train the ARIMA model just on the residuals of the time series, after properly decomposing it.
 - We can recover the original signal by “manually” adding the trend and seasonality we obtained in our decomposition
- A more sophisticated approach is to use the SARIMA (Seasonal ARIMA) class of models.

Seasonal Models

- SARIMA explicitly models the seasonal variations as a full fledged ARIMA model at the cost of 4 extra parameters: $SARIMA(p, d, q) \times (P, D, Q, s)$:
 - p, d and q , are defined exactly as above
 - s , is the seasonal periodicity
 - P, D and Q , are the parameters of an ARIMA model trained on the s -lagged time series

SARIMA

- To fit a SARIMA model, we follow a similar procedure to the one described above:



- With one important difference, in the Stationarize step, we differentiate on simple and seasonal lags

SARIMA

- Our procedure is then:



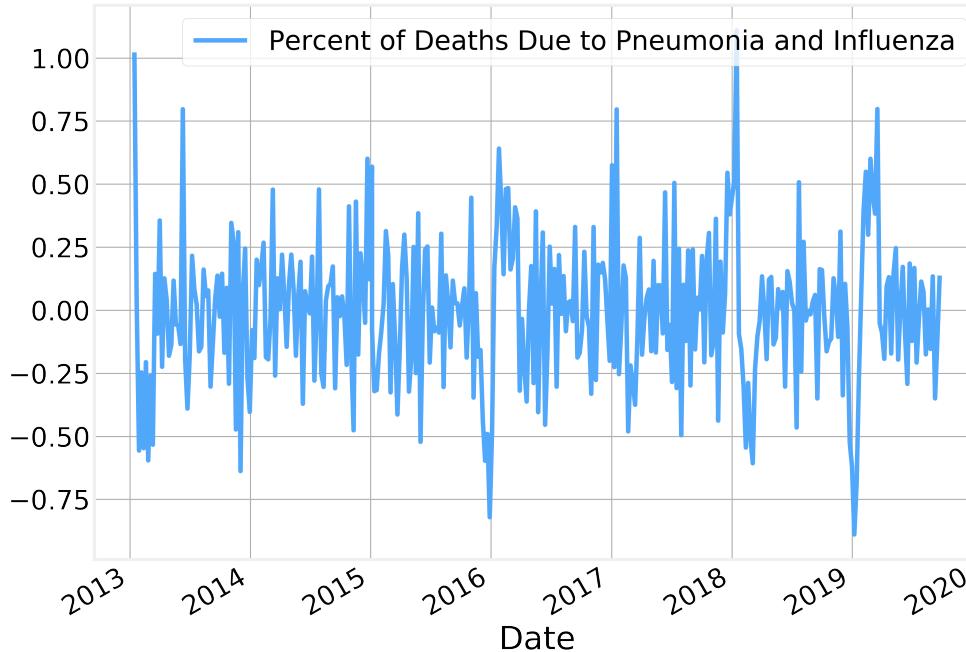
When plotting the ACD/PACF we must look for both **simple** and **seasonal** lags

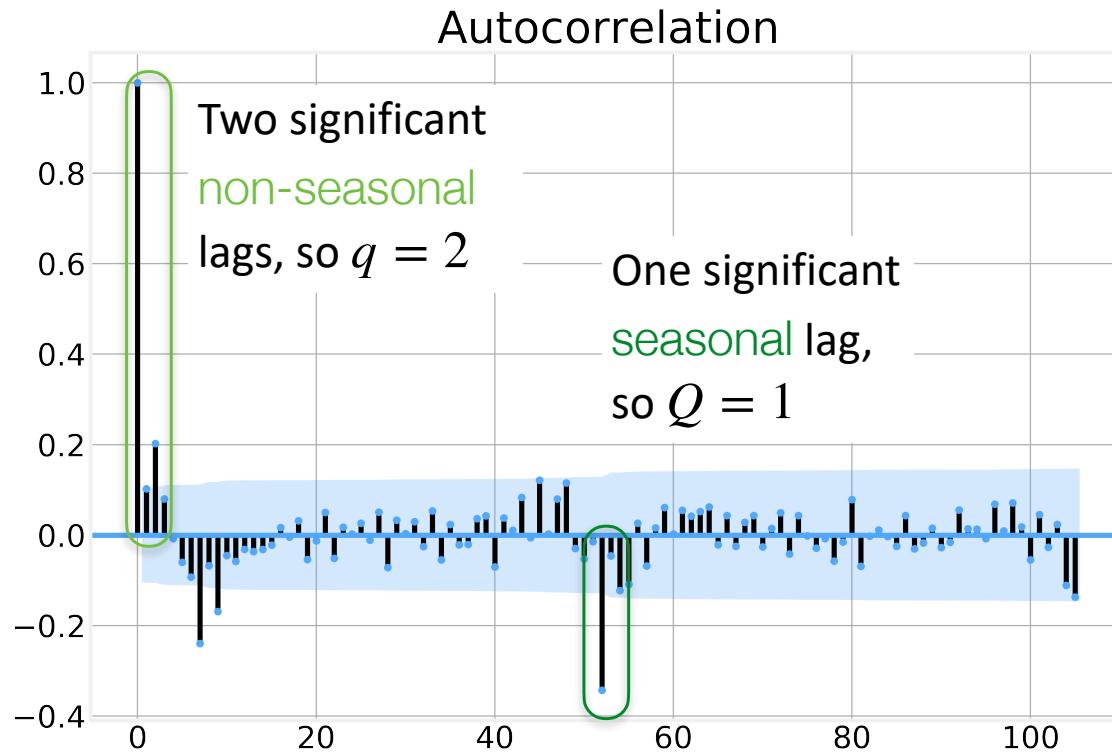
SARIMA

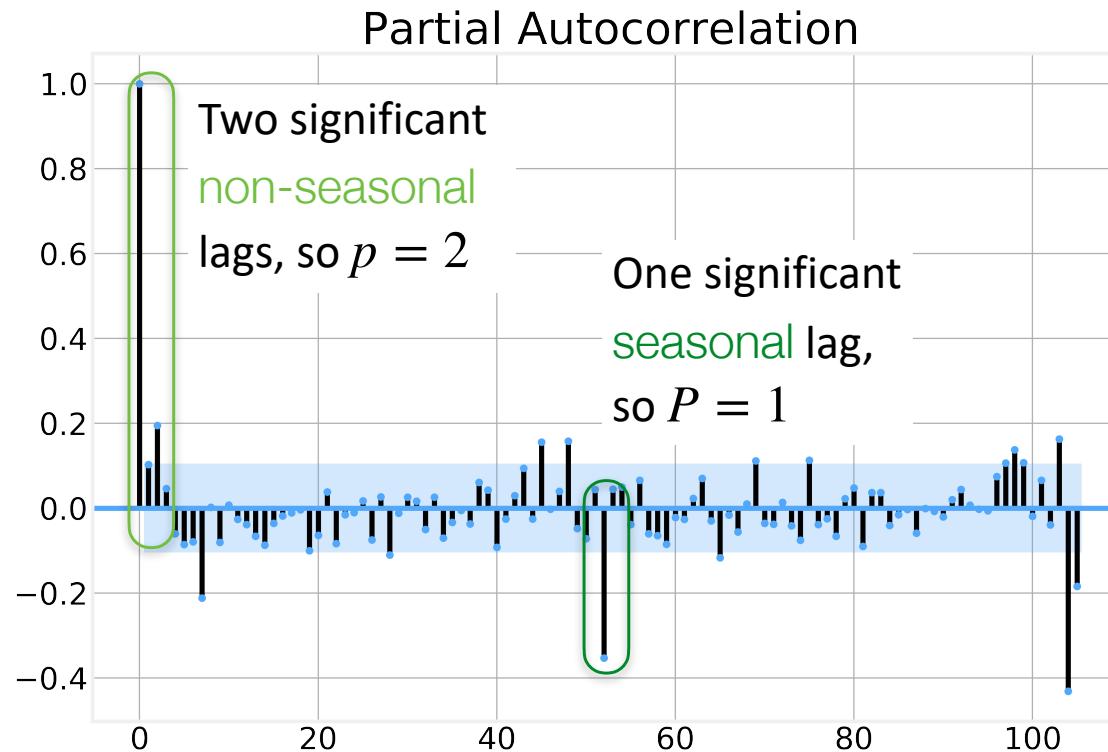
- `statsmodels` provides the `SARIMAX` model in the `tsa.statemodels` package.
- The API is similar to that of the `ARIMA` class of models, but now we must provide both:
 - `order` - (p, d, q) - the usual `ARIMA` parameters
 - `seasonal_order` - (P, D, Q, s) - the parameters specifying the seasonal component
- With this approach, we are now able to perform significantly better forecasts!

SARIMA

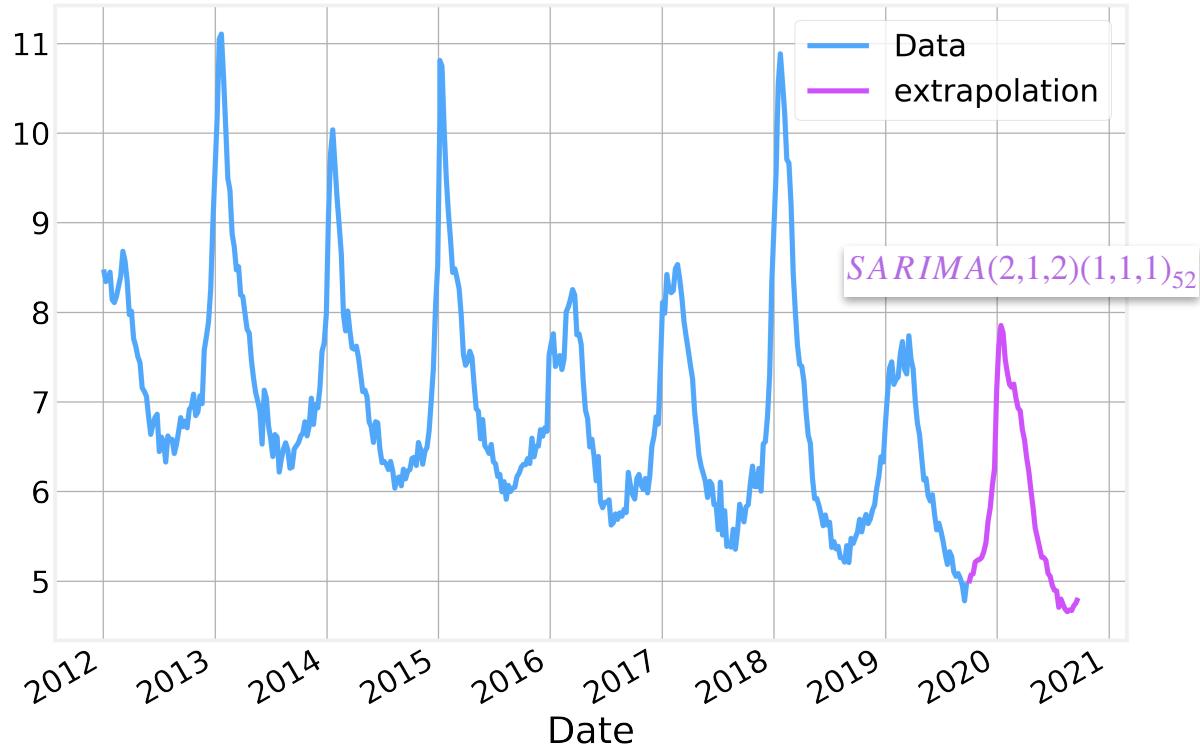
- The first step is to take the seasonal (52) followed but the non-seasonal (1) differences.







Forecast





Code - ARIMA

https://github.com/DataForScience/Timeseries_LL



Lesson 10 ARCH Models

10.1 Heteroscedasticity

10.2 Heteroscedastic Models

10.3 Autoregressive Conditionally
Heteroscedastic (ARCH) Models

10.4 Fitting ARCH Model



Lesson 10 ARCH Models

10.1 Heteroscedasticity

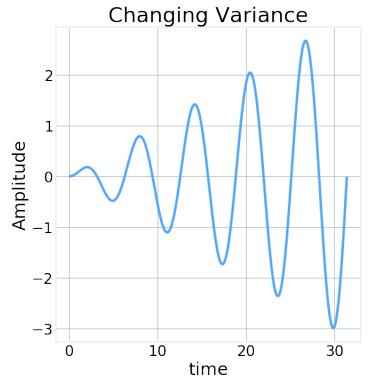
10.2 Heteroscedastic Models

10.3 Autoregressive Conditionally
Heteroscedastic (ARCH) Models

10.4 Fitting ARCH Model

Heteroscedasticity

- Some time series display variance that changes in time:



The temporal behavior of Variance is not required to be monotonous.

- This phenomenon is called Heteroscedasticity
- This kind of time series required a specific class of models from the ones we have considered so far



Lesson 10 ARCH Models

10.1 Heteroscedasticity

10.2 Heteroscedastic Models

10.3 Autoregressive Conditionally
Heteroscedastic (ARCH) Models

10.4 Fitting ARCH Model

Heteroscedastic Models

- ARIMA class models can be written as:

$$x_t = f(x_{t-1}, \dots)$$

- Heteroscedastic Models, referred as ARCH models, can be interpreted as being a simple AR model applied to the variance of a series.

$$\text{Var}(x_t | x_{t-1}) \equiv \sigma_t^2 = f(x_{t-1}, \dots)$$

- In the simplest case of the Autoregressive Model, AR(1) we have:

$$x_t = \mu + \sigma_t \epsilon_t$$

- With

$$\sigma_t^2 = \alpha_0 + \alpha_1 x_{t-1}^2$$

Heteroscedastic Models

- We obtain the order one Autoregressive Conditionally Heteroscedastic Model, ARCH(1) model.
- We further impose the constraints $\alpha_0 > 0$, $\alpha_1 > 0$ and $\alpha_1^2 < 1/3$ to avoid negative variances.



Lesson 10 ARCH Models

10.1 Heteroscedasticity

10.2 Heteroscedastic Models

**10.3 Autoregressive Conditionally
Heteroscedastic (ARCH) Models**

10.4 Fitting ARCH Model

ARCH Model

- The ARCH(1) Model can be written:

$$x_t = \sigma_t \epsilon_t$$

$$\sigma_t^2 = \alpha_0 + \alpha_1 x_{t-1}^2$$

- Where we assume that the stochastic variable ϵ_t is Normally distributed:

$$\epsilon_t \sim \mathcal{N}(0,1)$$

Under these conditions, we have:

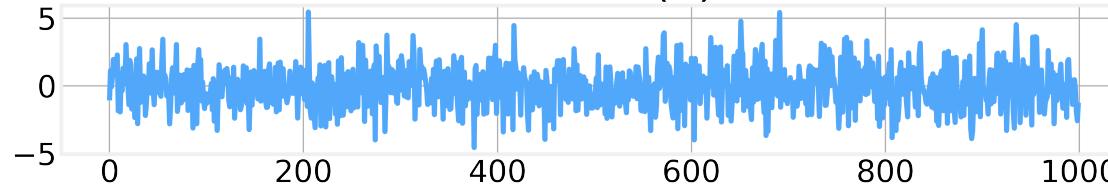
$$x_t^2 = \alpha_0 + \alpha_1 x_{t-1}^2 + \text{error}$$

or, in other words, an AR(1) model for x_t^2 ! But with a significant error term...

- And x_t is just White Noise!
- Let's start by simulating this process in order to understand it better

ARCH Process

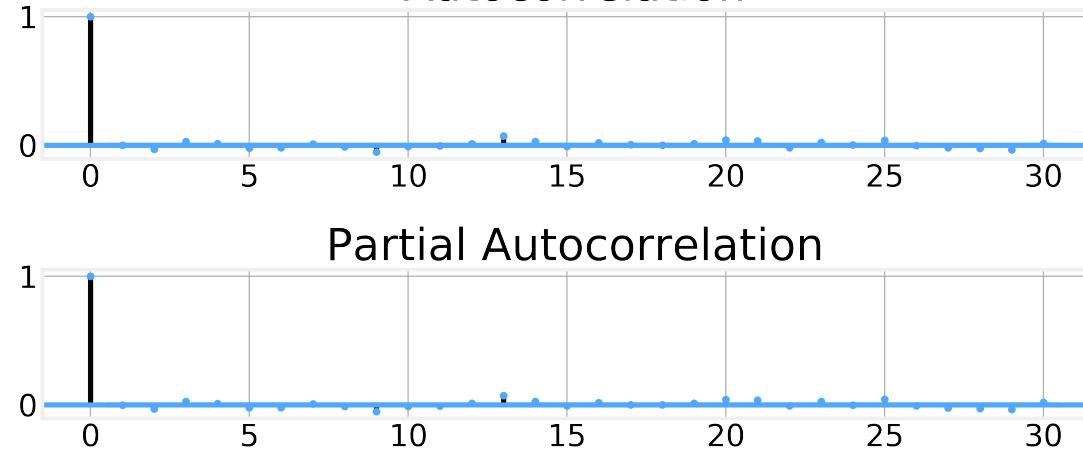
Simulated ARCH(1) Process



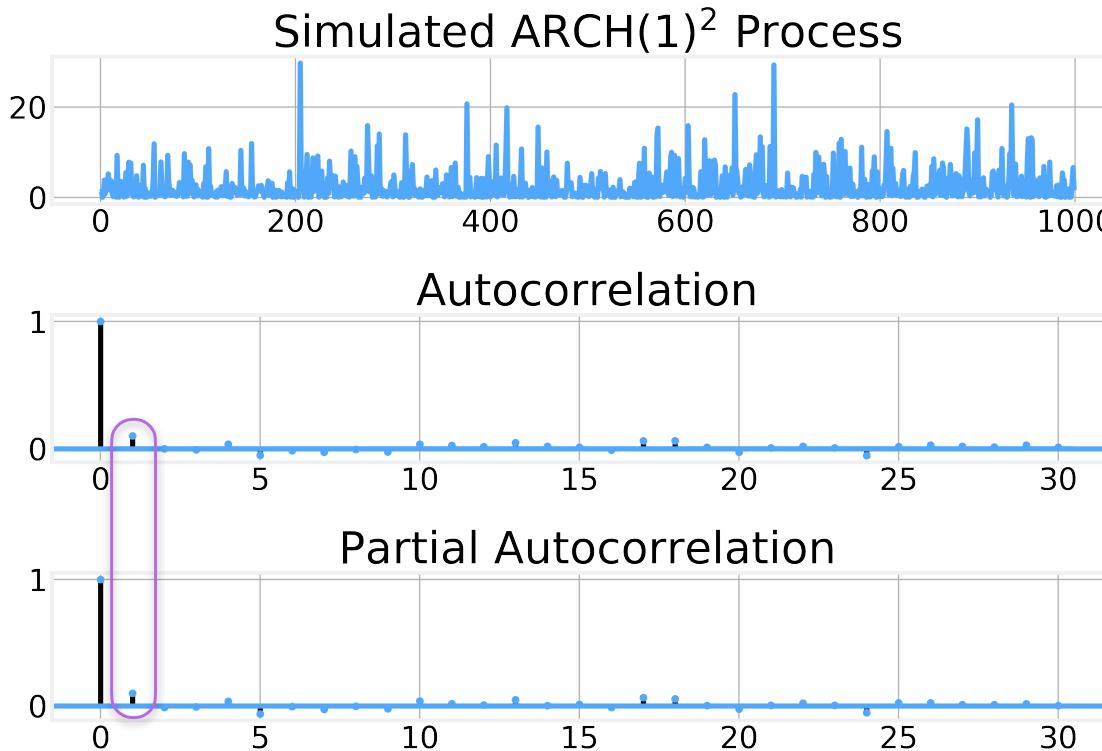
Autocorrelation

Apparently just
White-Noise

Partial Autocorrelation



ARCH Process



GARCH Model

- There are many variants of the ARCH model, such as the [ARCH\(m\)](#):

$$Var(x_t | x_{t-1}, \dots, x_{t-m}) = \sigma_t^2 = \alpha_0 + \sum_{l=1}^m \alpha_l x_{t-l}^2$$

- But perhaps the best known variant is the [Generalized Autoregressive Conditionally Heteroscedastic \(GARCH\) Model](#).
- In [GARCH](#) models, we allow for both a [AR](#) and a [MA](#) components.
- The [GARCH](#) model can be defined as:

$$x_t = \sigma_t \epsilon_t$$

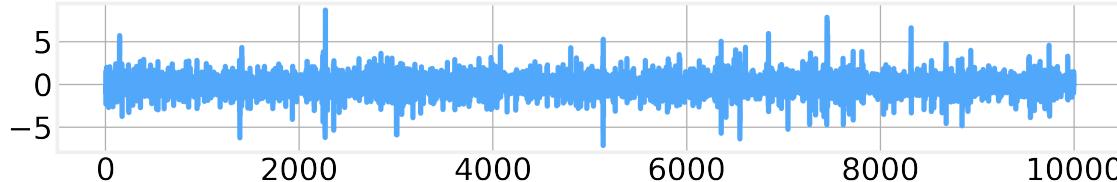
$$\sigma_t^2 = \alpha_0 + \alpha_1 x_{t-1}^2 + \beta_1 \sigma_{t-1}^2$$

where the β term represents the [MA](#) component.

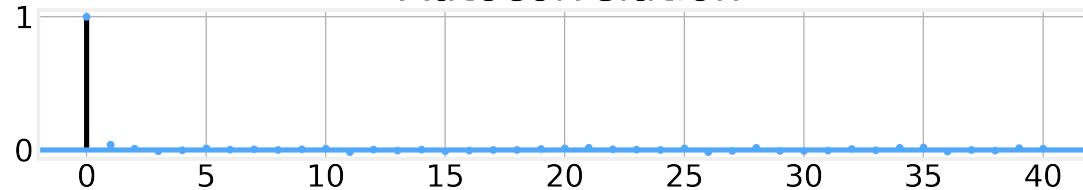
- If we simulate this process, we obtain

GARCH Model

Simulated GARCH(1,1) Process

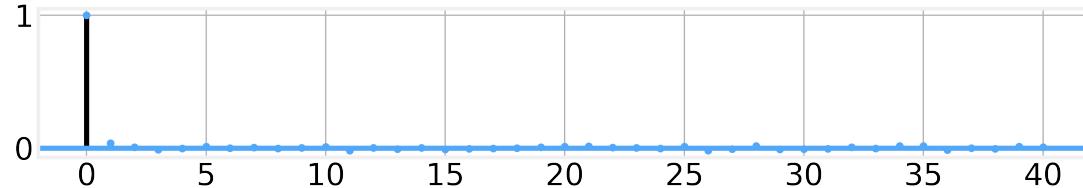


Autocorrelation



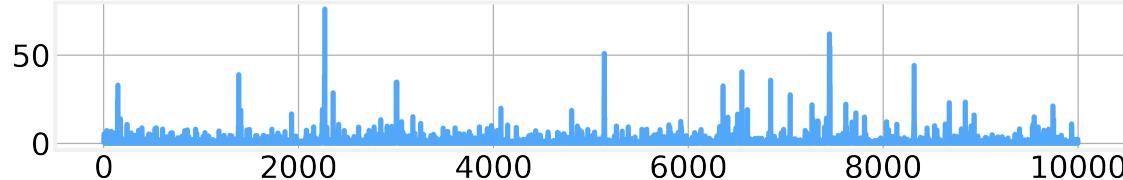
Again just
White-Noise

Partial Autocorrelation

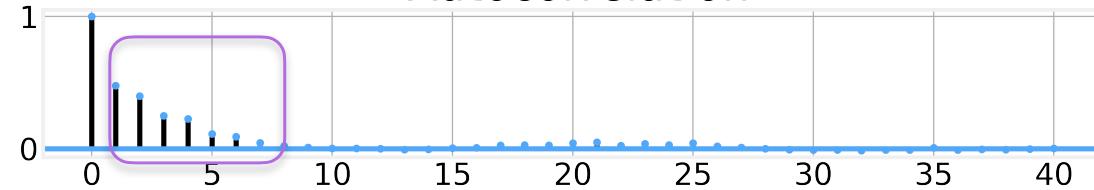


GARCH Model

Simulated GARCH(1,1)² Process

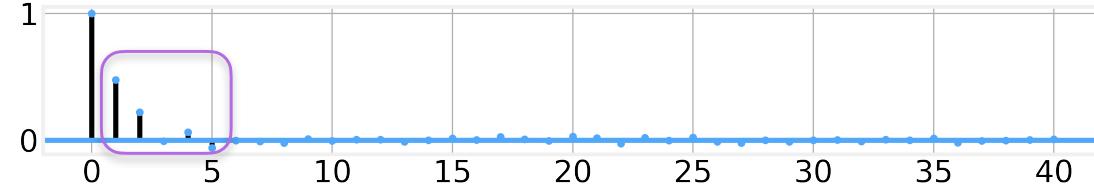


Autocorrelation



Significant
values of ACF
and PACF at
various lags!

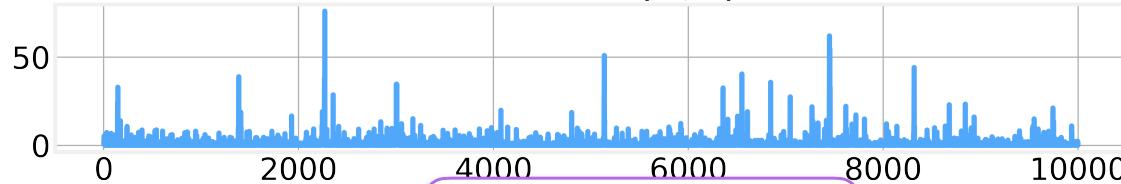
Partial Autocorrelation



GARCH Model

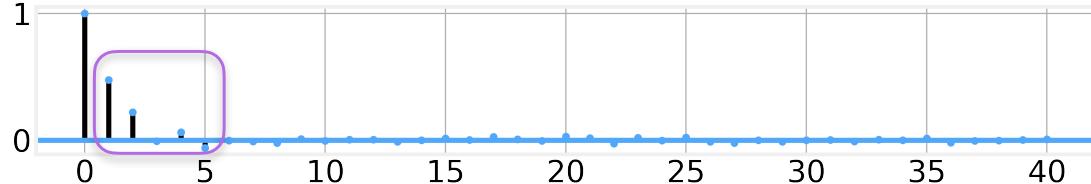
Significant values of ACF and PACF at various lags!

Simulated GARCH(1,1)² Process



No simple way to determine the parameters of the model!

Partial Autocorrelation





Lesson 10 ARCH Models

10.1 Heteroscedasticity

10.2 Heteroscedastic Models

10.3 Autoregressive Conditionally
Heteroscedastic (ARCH) Models

10.4 Fitting ARCH Model

Fitting G/ARCH models

Much more involved than fitting ARIMA class models!

Visualize Time Series

Plot ACF/PACF

Plot Squared ACF/PACF

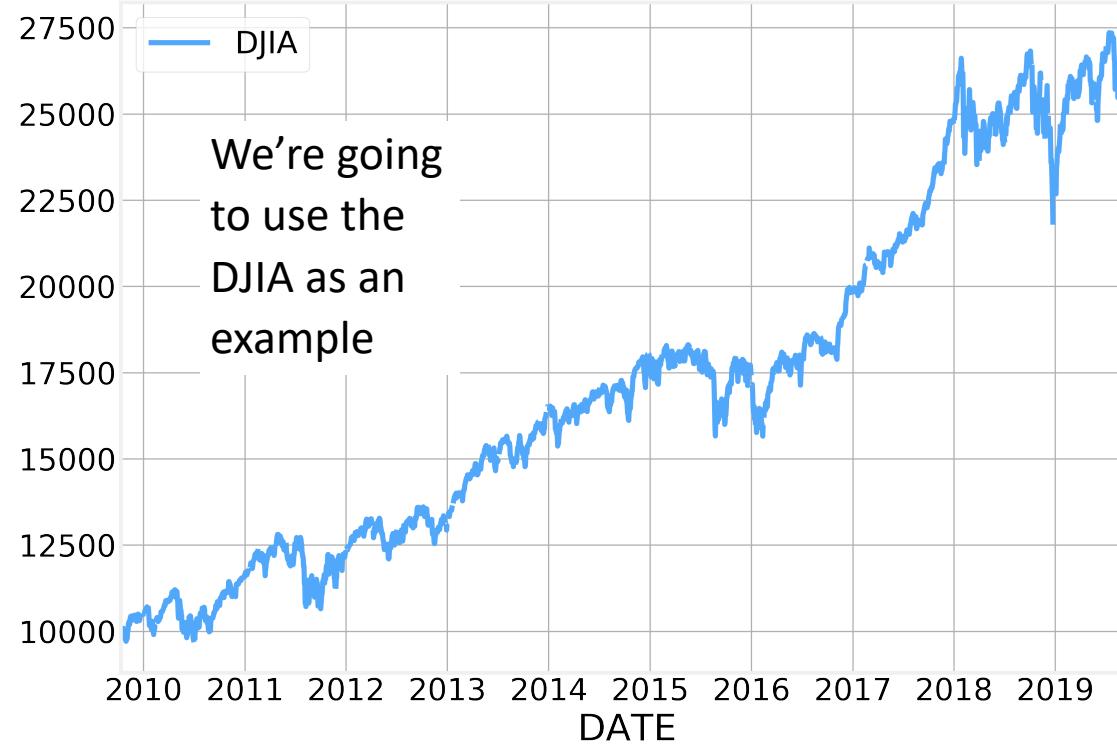
Determine G/ARCH

Build G/ARCH Model

Forecast

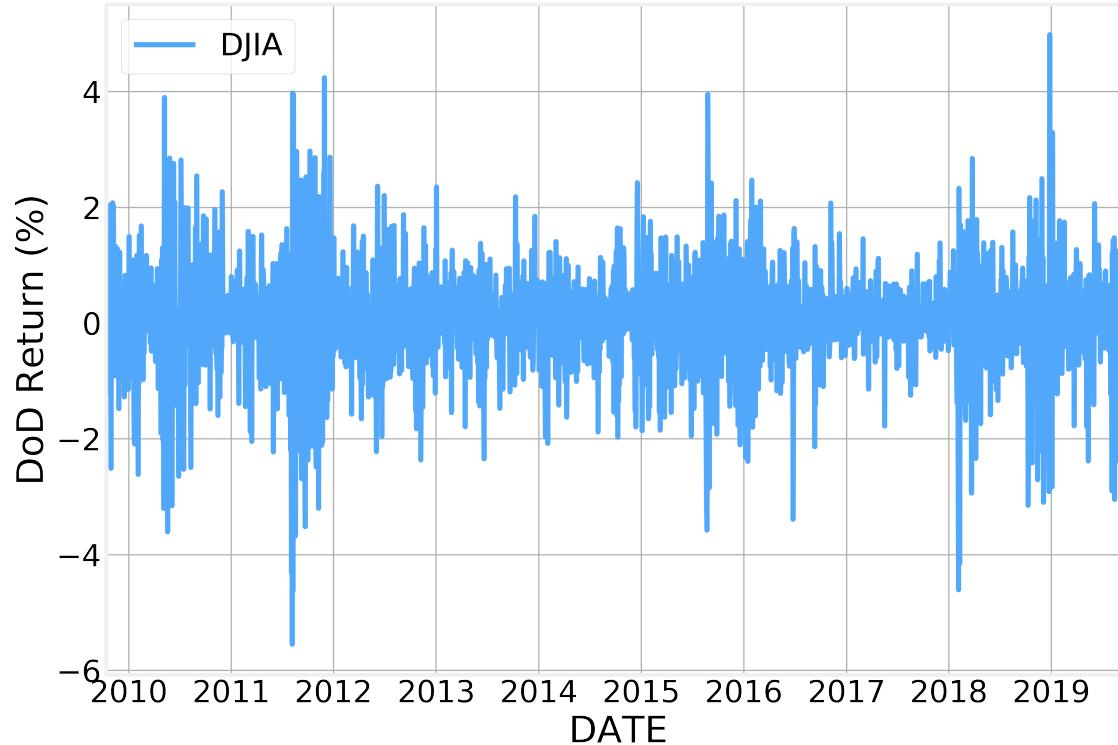
Determine the correct order for model by comparing statistics metrics like **AIC**!

Visualize Time Series

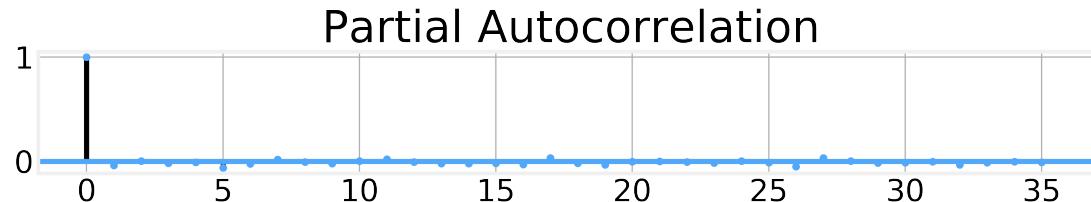
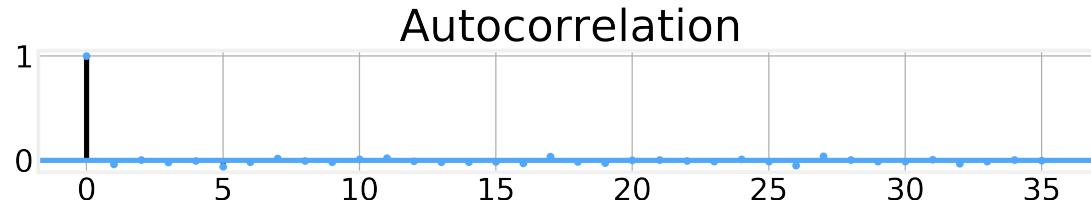
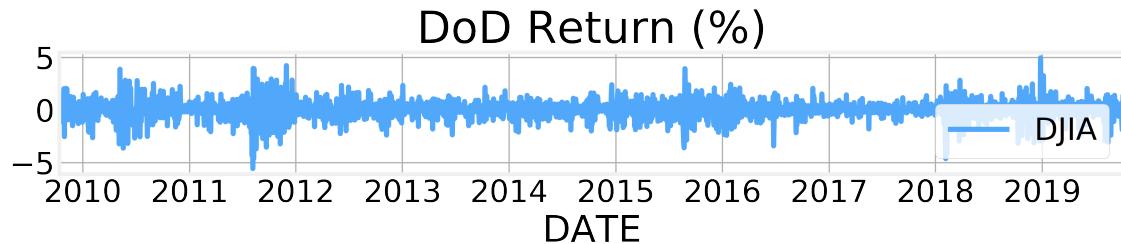


Visualize Time Series

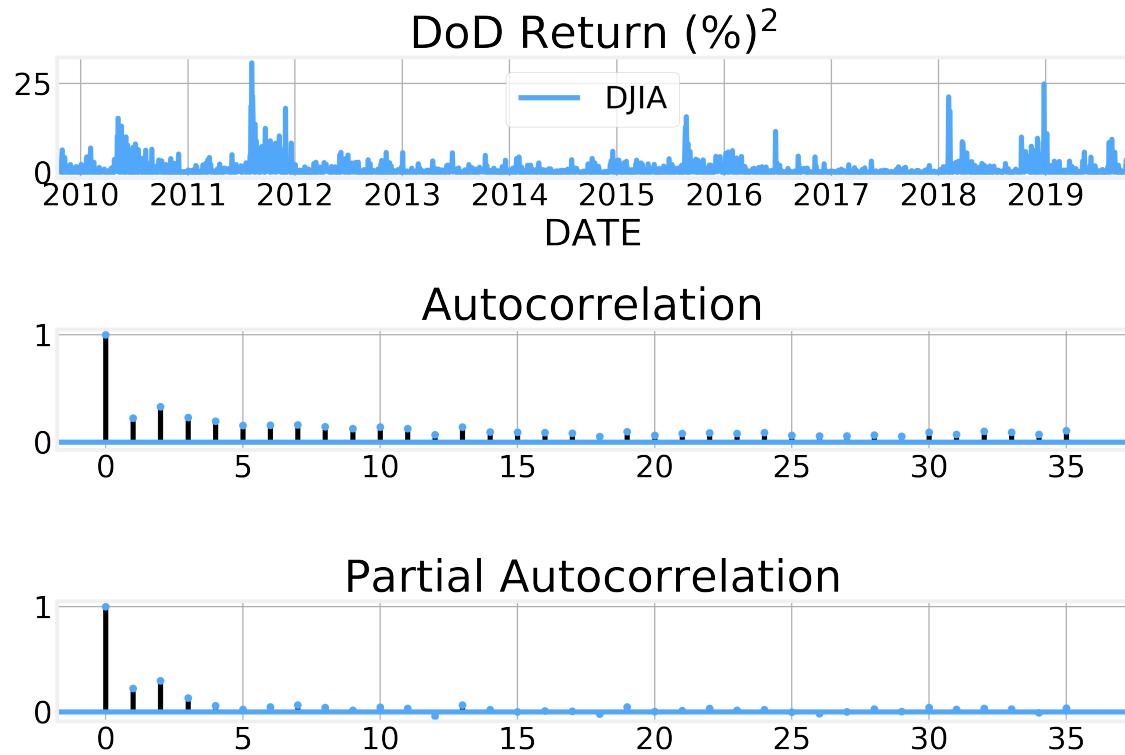
We convert
the daily
values to
returns so that
we have a
“White-noise”
like series



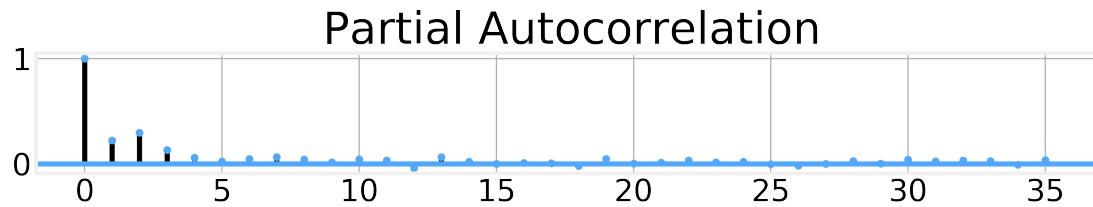
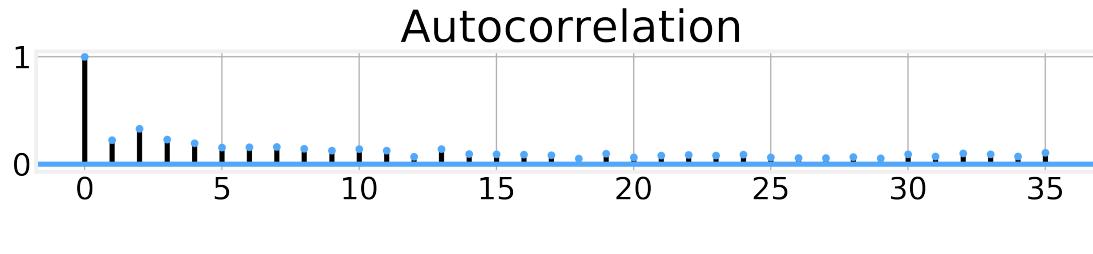
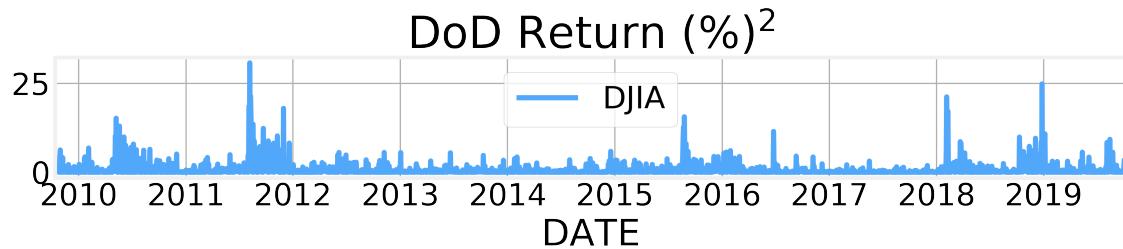
Plot ACF/PACF



Plot Squared ACF/PACF



Determine G/ARCH



GARCH model
since we have
both ACF and
PACF



- To fit and model ARCH and GARCH models, we will use the `arch` library
- `arch` includes a large amount of functionality, but we will focus on the `univariate` sub-package.
- `arch.univariate` provides the `arch_model()` function that allows us to define a wide variety of ARCH or GARCH models.
- While `arch_model()` can take many arguments, there's only a few that are relevant for our purposes:
 - `y` - the (dependent) variable data to train on
 - `vol` - the name of the volatility model, typically “ARCH” or “GARCH”
 - `p, o, q` - the model parameters

- When a `arch_model()` returns a `ARCHModel` object that represents our model. This object provides a great deal of functionality, but we're particularly interested in:
 - `fit()` - fit the parameters of the model (the values of α_i, β_j , etc.)
 - `simulate()` - generate new values from the fitted model.
 - `forecast()` - forecast new values of the data.

And continuing with our example...

Build G/ARCH Model

- We use `arch` to generate a **GARCH(1,1)** model and train it on the DJIA returns dataset.
- `arch` provides us with a nice summary of the model parameters, similarly to `statsmodels`

parameter
coefficients

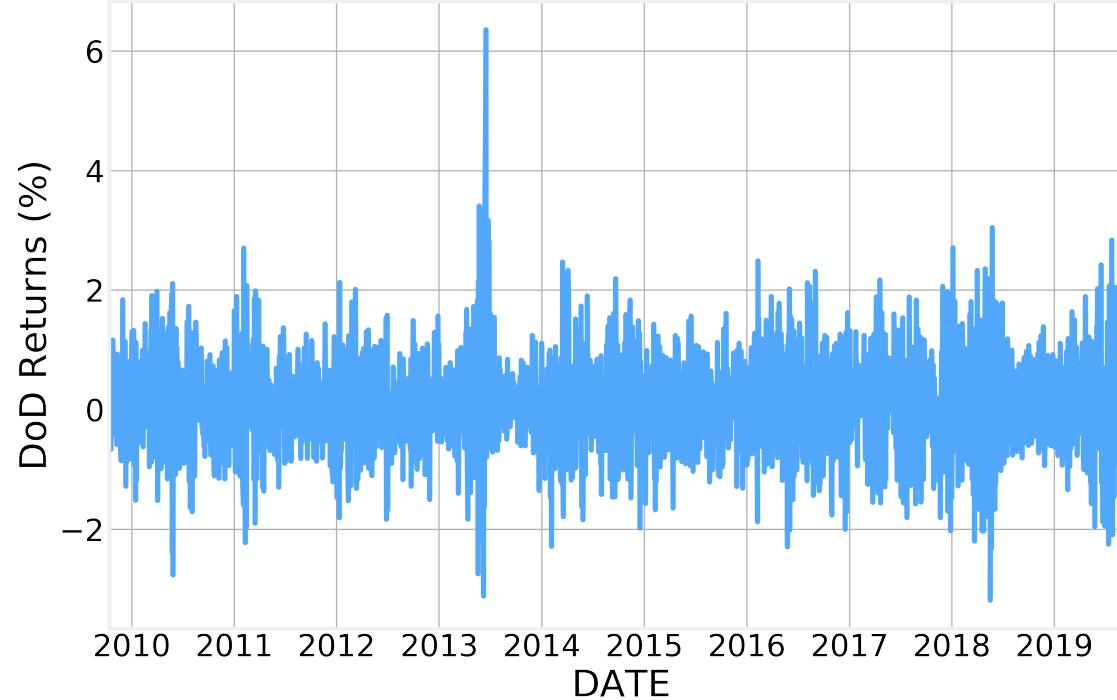
Constant Mean - GARCH Model Results				
Dep. Variable:	DJIA	R-squared:	-0.001	
Mean Model:	Constant Mean	Adj. R-squared:	-0.001	
Vol Model:	GARCH	Log-Likelihood:	-3000.92	
Distribution:	Normal	AIC:	6009.85	
Method:	Maximum Likelihood	BIC:	6033.31	
No. Observations:				
Date:	Mon, Aug 10 2020	Df Residuals:	2604	
Time:	23:46:42	Df Model:	4	

Can use AIC/BIC to compare various models and pick the best (smallest) one

Mean Model					
	coef	std err	t	P> t	95.0% Conf. Int.
mu	0.0744	1.327e-02	5.608	2.043e-08	[4.840e-02, 0.100]

Volatility Model					
	coef	std err	t	P> t	95.0% Conf. Int.
omega	0.0318	6.646e-03	4.780	1.749e-06	[1.875e-02, 4.480e-02]
alpha[1]	0.1587	2.316e-02	6.851	7.310e-12	[0.113, 0.204]
beta[1]	0.8033	2.280e-02	35.232	6.527e-272	[0.759, 0.848]

Simulated GARCH(1,1) Process





Code - ARCH

https://github.com/DataForScience/Timeseries_LL



Machine Learning with Time Series

11.1 Interpolation

11.2 Types of Machine Learning

11.3 Regression and Classification

11.4 Cross-Validation

11.5 Caveats when working with Time Series



Machine Learning with Time Series

11.1 Interpolation

11.2 Types of Machine Learning

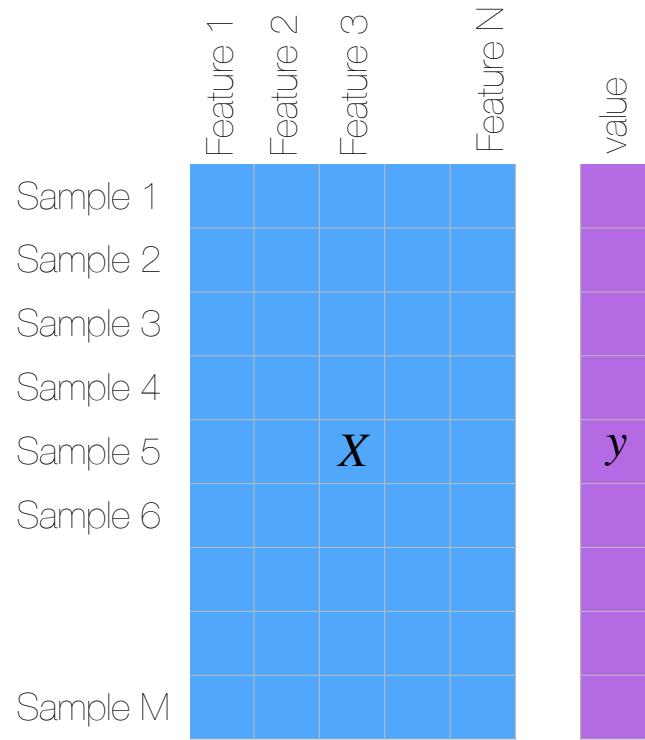
11.3 Regression and Classification

11.4 Cross-Validation

11.5 Caveats when working with Time Series

Data Representations

- ML algorithms typically expect a **table-like representation** of the input data
- In financial applications these are known as ‘Bars’
- In the ideal case, bars (or samples) are **equally spaced** but in practice that’s not always the case
- A typical pre-processing step is that of resampling the time series to make it conform to a uniform sampling frequency



Resampling and Interpolation

<https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.resample.html>
<https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.Series.interpolate.html>

- We often resample time series in order to obtain a representation that is equally spaced in time
- Pandas provides several methods to make our lives easier:
 - `resample()` - frequency conversion and resampling of time series.
 - Same frequency values as `pd.time_range`: ‘B’, ‘D’, ‘W’, ‘M’, etc
 - `interpolate()` - Fill in NaN values by interpolating the time series.
 - Relies on `scipy.interpolate.interp1d()` and supports ‘slinear’, ‘quadratic’, ‘cubic’, ‘polynomial’, etc
 - Conceptually similar to imposing a simple model on the missing values



Machine Learning with Time Series

11.1 Interpolation

[**11.2 Types of Machine Learning**](#)

11.3 Regression and Classification

11.4 Cross-Validation

11.5 Caveats when working with Time Series

3 Types of Machine Learning

- Supervised Learning
 - Predict output given input
 - Training set of known inputs and outputs is provided
- Unsupervised Learning
 - Autonomously learn a good representation of the dataset
 - Find clusters in input
- Reinforcement Learning
 - Learn sequence of actions to maximize payoff
 - Discount factor for delayed rewards



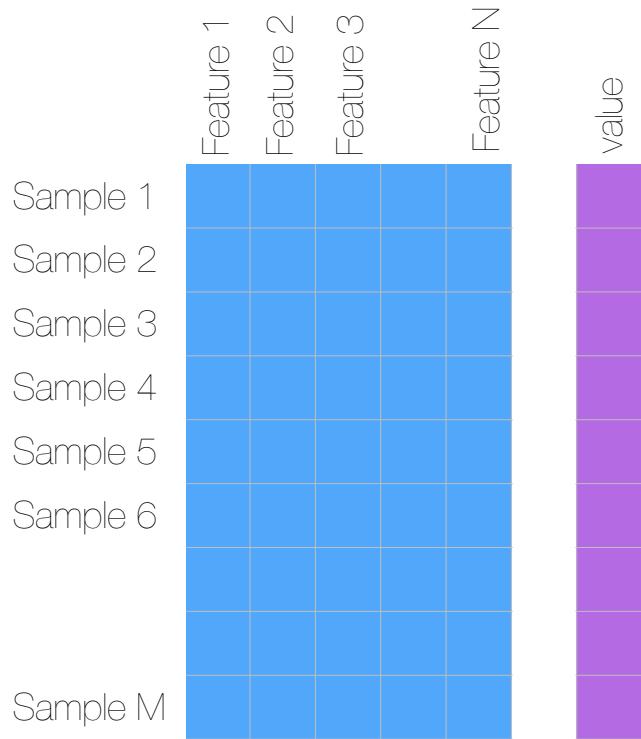
3 Types of Machine Learning

- Supervised Learning
 - Predict output given input
 - Training set of known inputs and outputs is provided
- Unsupervised Learning
 - Autonomously learn a good representation of the dataset
 - Find clusters in input
- Reinforcement Learning
 - Learn sequence of actions to maximize payoff
 - Discount factor for delayed rewards



Machine Learning

- Dataset formatted as an $M \times N$ matrix of M samples and N features
- Supervised learning: Each sample corresponds to a **specific value** of the target variable
- Unsupervised learning: Each sample will be assigned a **cluster label** based on how similar it is to other samples.





Machine Learning with Time Series

11.1 Interpolation

11.2 Types of Machine Learning

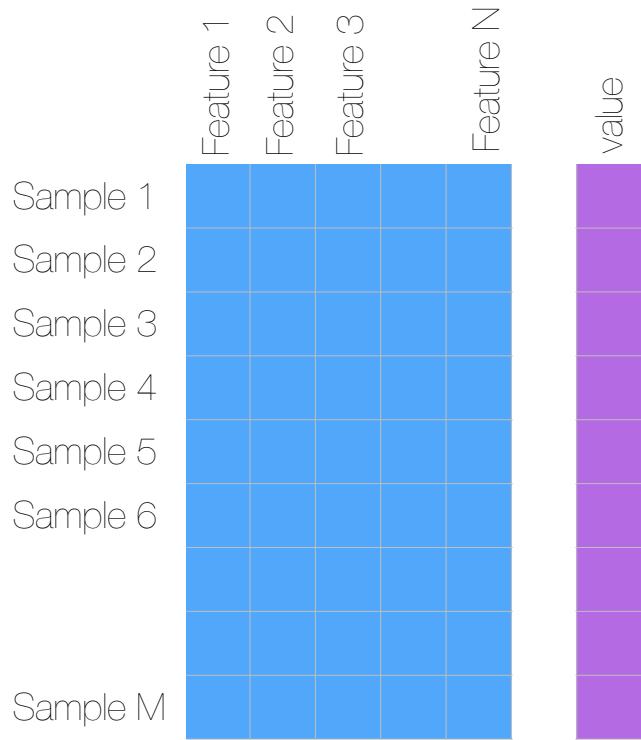
[**11.3 Regression and Classification**](#)

11.4 Cross-Validation

11.5 Caveats when working with Time Series

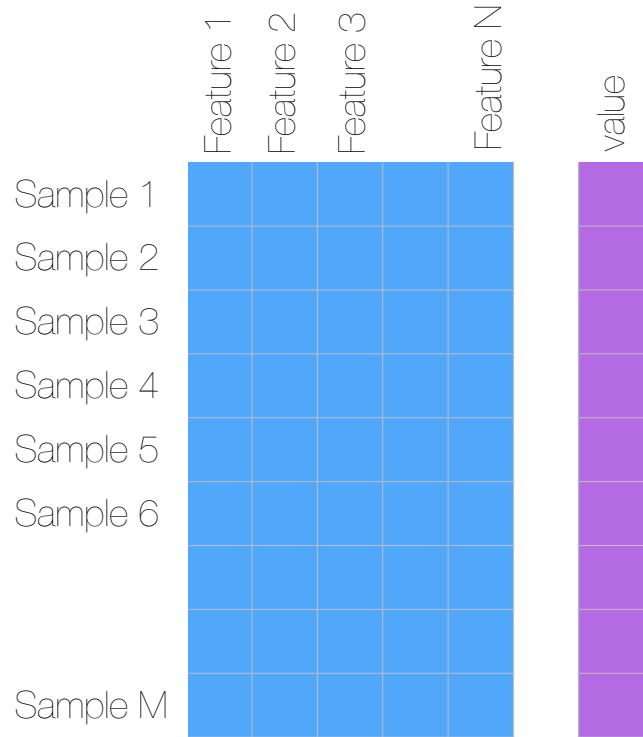
Supervised Learning

- Dataset formatted as an $M \times N$ matrix of M samples and N features
- Each sample corresponds to a specific **value** of the target variable
- Two fundamental types of problems:
 - Regression (continuous output value)
 - Classification (discrete output value)



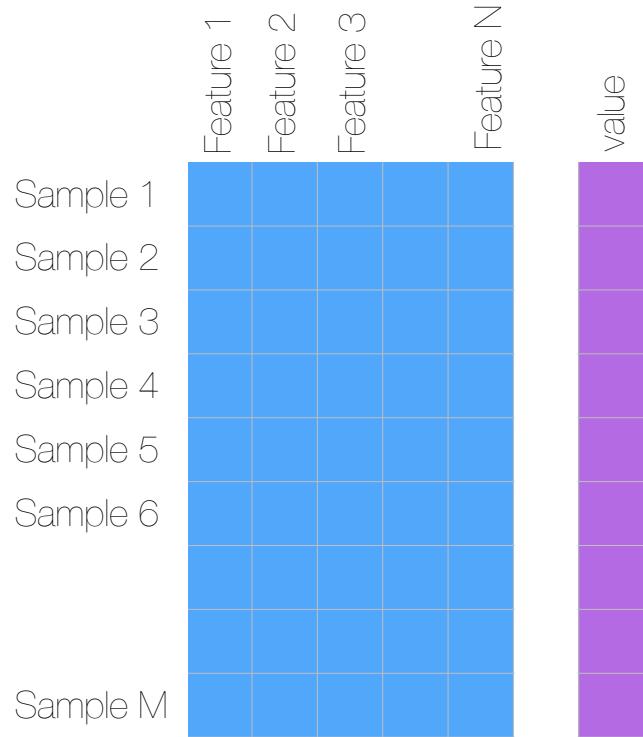
Supervised Learning

- Regression predicts or approximates the value of a function at previously unseen points.
 - Linear Regression
 - Neural Networks
- Classification predicts to which class a previously unseen time series belongs to by learning defining regularities of each class
 - Logistic Regression
 - Neural Networks

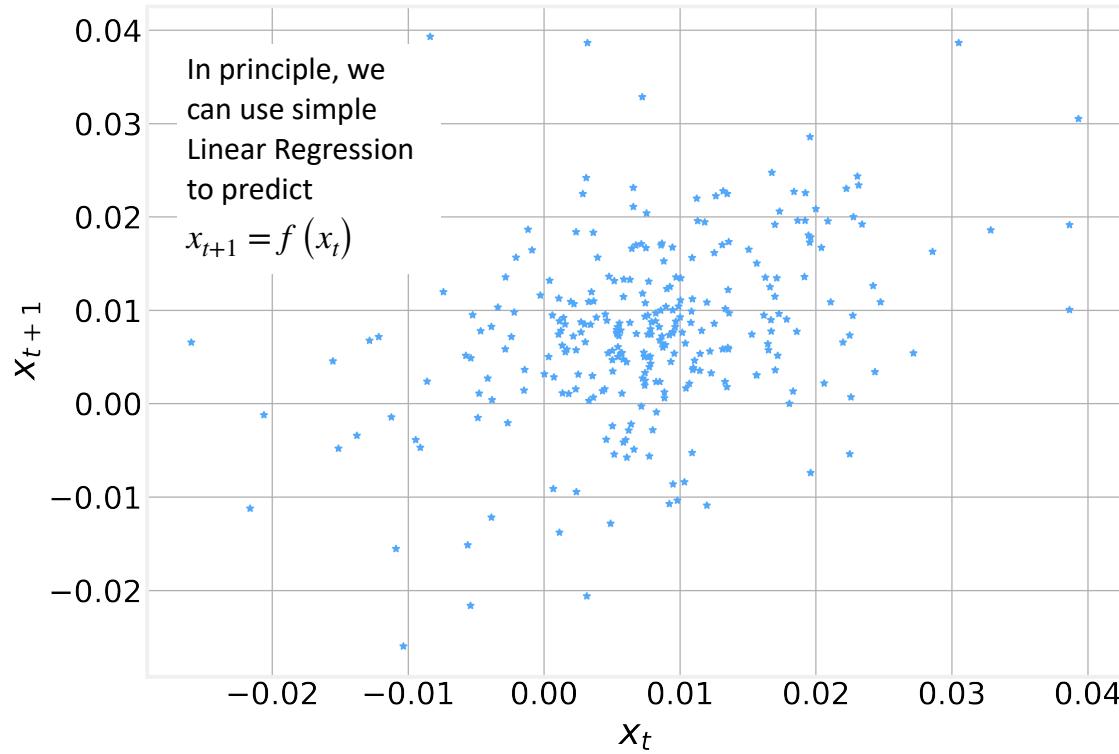


Supervised Learning

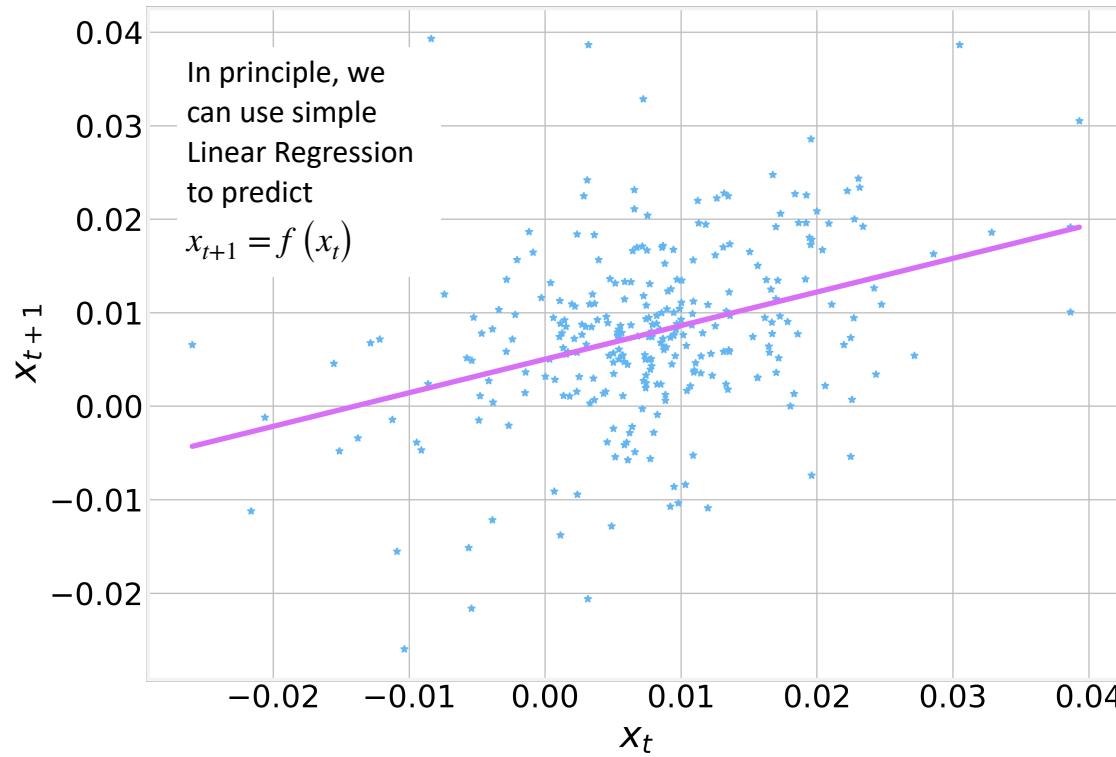
- Regression predicts or approximates the value of a function at previously unseen points.
 - Linear Regression
 - Neural Networks
- Classification predicts to which class a previously unseen time series belongs to by learning defining regularities of each class
 - Logistic Regression
 - Neural Networks



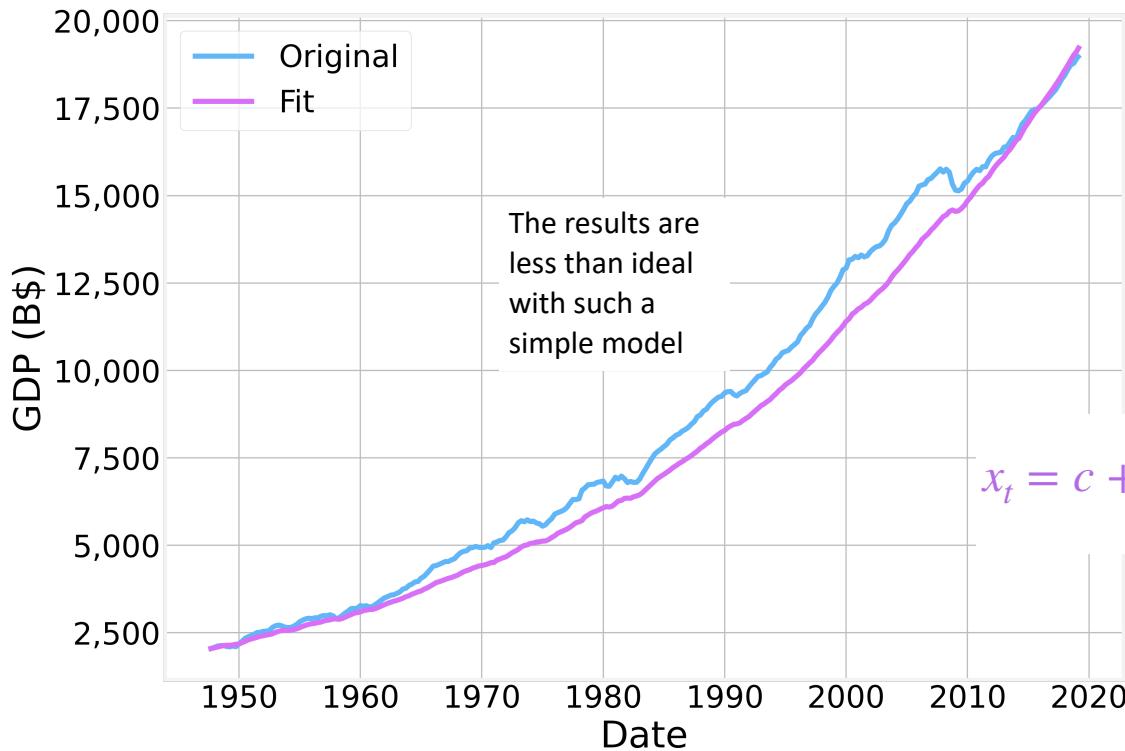
Linear Regression



Linear Regression



Linear Regression



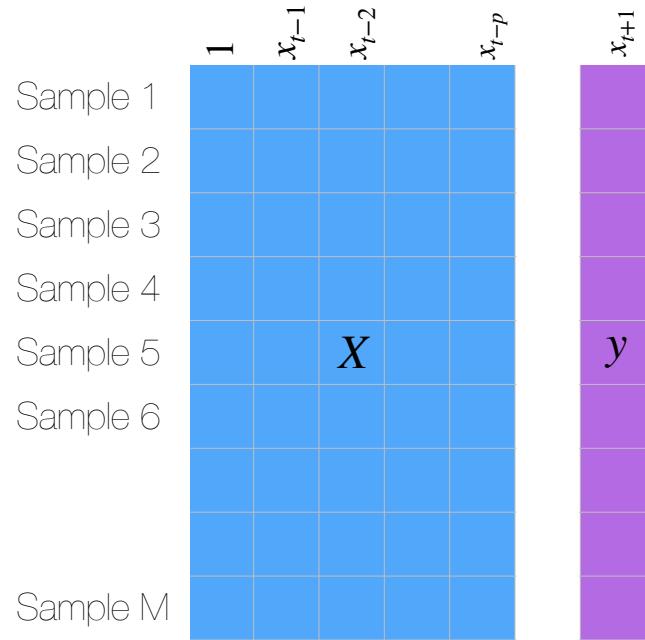
But this is what we've been doing with Auto-Regressive models!

$$x_t = c + \sum_i^p \phi_i x_{t-i} + \epsilon_t$$

Auto-Regressive Model

- Features are just the lagged values of the time series, plus the (constant) intercept value.
- Regression coefficients are the ϕ_i parameters that AR models find for us

$$x_t = c + \sum_i^p \phi_i x_{t-i} + \epsilon_t$$



Moving Average Model

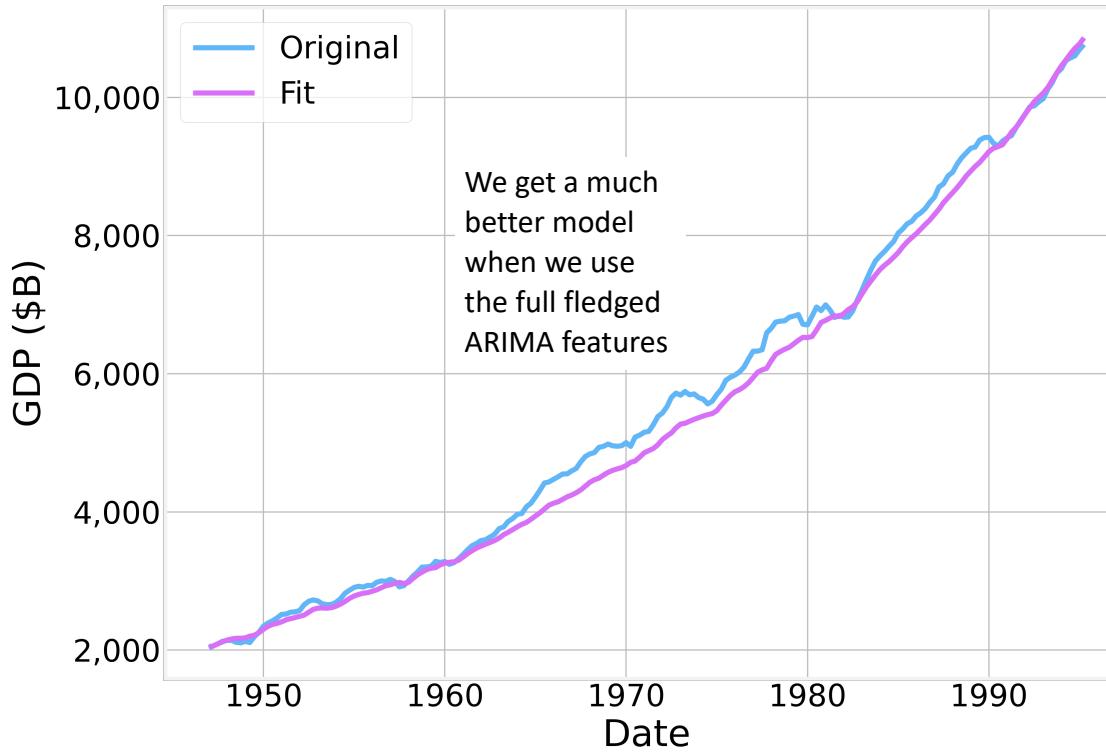
- In the Moving Average model, the features are the stochastic variables, ϵ_t and the regression coefficients are the θ_i parameters
- The values of the stochastic variables, ϵ_i , are simply the residuals of a simple Linear Regression of order p
- We can combine both models into a fully fledged ARIMA(p, d, q) model by simply placing the input matrices for both models side by side (and computing the features after the appropriate number of differences)

$$x_t = \mu + \epsilon_t + \sum_i^q \theta_i \epsilon_i$$

	1	ϵ_{t-1}	ϵ_{t-2}	ϵ_{t-q}	x_{t+1}
Sample 1					
Sample 2					
Sample 3					
Sample 4					
Sample 5					
Sample 6					X
Sample M					

y

ARIMA Model





Machine Learning with Time Series

11.1 Interpolation

11.2 Types of Machine Learning

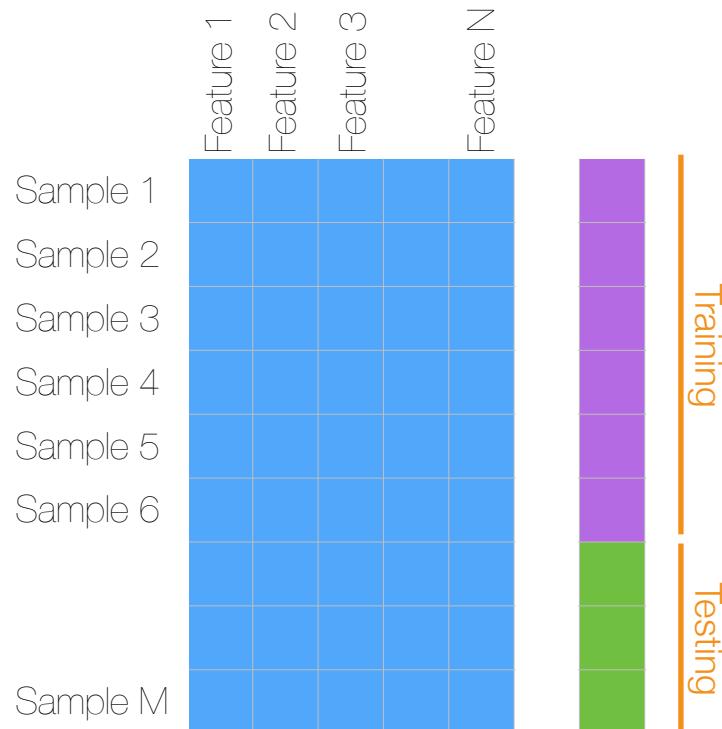
11.3 Regression and Classification

[**11.4 Cross-Validation**](#)

11.5 Caveats when working with Time Series

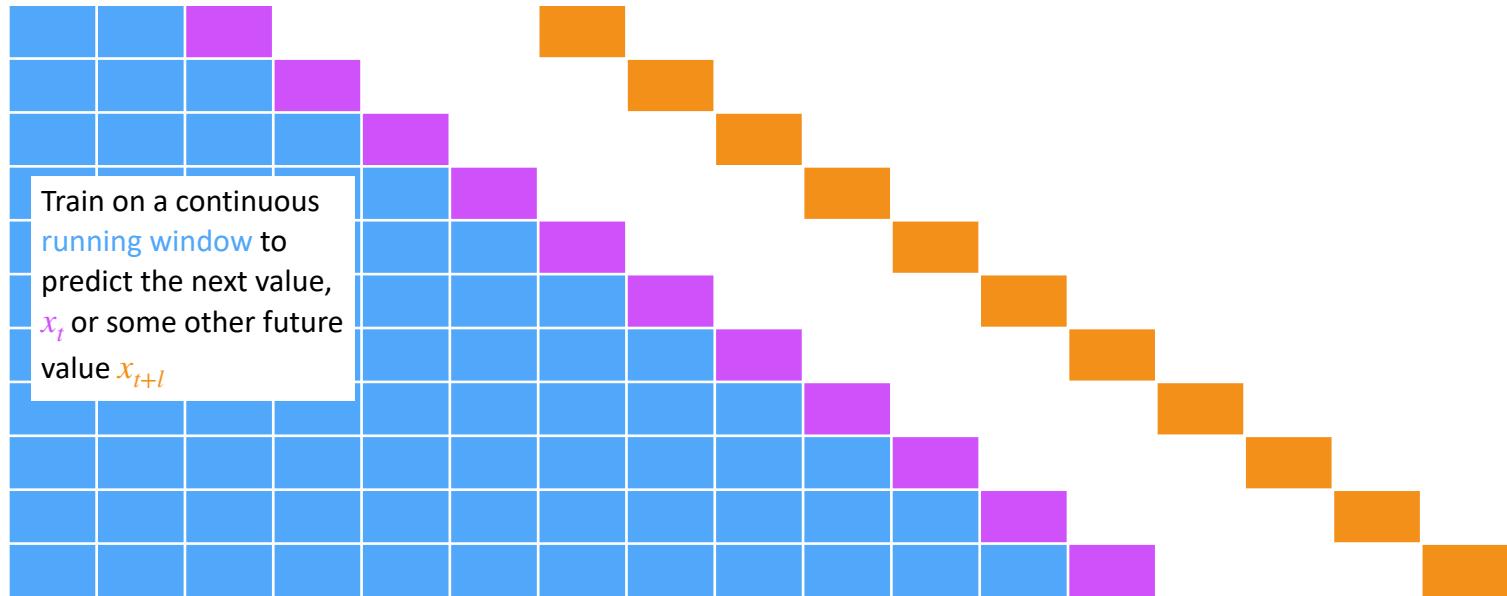
Cross Validation

- After training a model, we must evaluate its results in order to decide how good it is.
- The **train-test** split is the approach we have used so far.
- It has the down side that it provides us only with a single training dataset.
- In Machine Learning, a common alternative is k-fold Cross-Validation that splits the dataset in k equal parts and repeatedly trains the model in k-1



Cross Validation

- Time series require a more sophisticated version of the usual cross validation approach to avoid contaminating the more with future knowledge.



Backtesting

- This Cross-Validation approach is known as ‘Backtesting’ and it has several advantages:
 - Sequential: No future values are ever used.
 - As similar as possible to simply training the model and ‘waiting to see’ how well it does
 - No validation set: All available data is used
 - Can accommodate arbitrary performance measures



Machine Learning with Time Series

11.1 Interpolation

11.2 Types of Machine Learning

11.3 Regression and Classification

11.4 Cross-Validation

11.5 Caveats when working with Time Series

Caveats

- Time series models assume that all information necessary to predict the future can be found in past values. This is **obviously not true**, but often **good enough**
- Always Stationarize and detrend your time series before attempting to model
- Make sure to avoid future leakage
- Check that you are not just memorizing past values
- When it comes to forecasting, simpler is often better. Start with basic models (like ARIMA) before moving on to more sophisticated methods.
- Use Backtesting to validate your models



Code - Machine Learning

https://github.com/DataForScience/Timeseries_LL



Deep Learning Approaches

12.1 Feed Forward Networks (FFN)

12.2 Recurrent Neural Networks (RNN)

12.3 Gated Recurrent Units (GRU)

12.4 Long Short-Term Memory (LSTM)



Deep Learning Approaches

12.1 Feed Forward Networks (FFN)

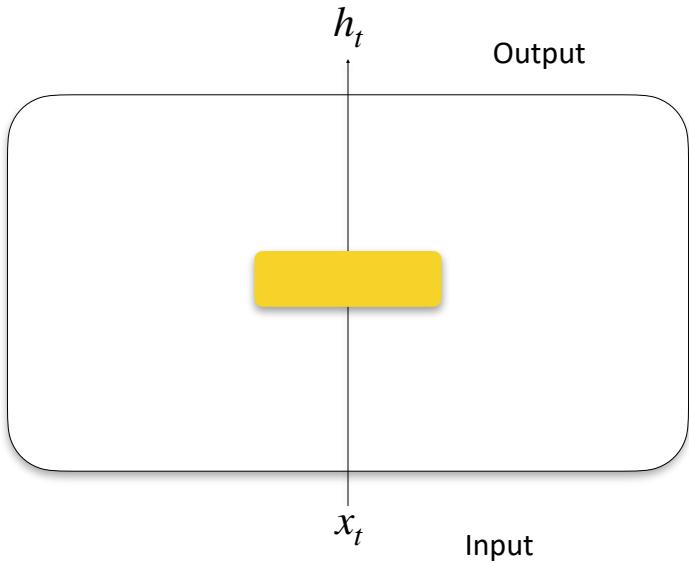
12.2 Recurrent Neural Networks (RNN)

12.3 Gated Recurrent Units (GRU)

12.4 Long Short-Term Memory (LSTM)

Feed Forward Networks

- Every approach we've used so far has followed a similar approach

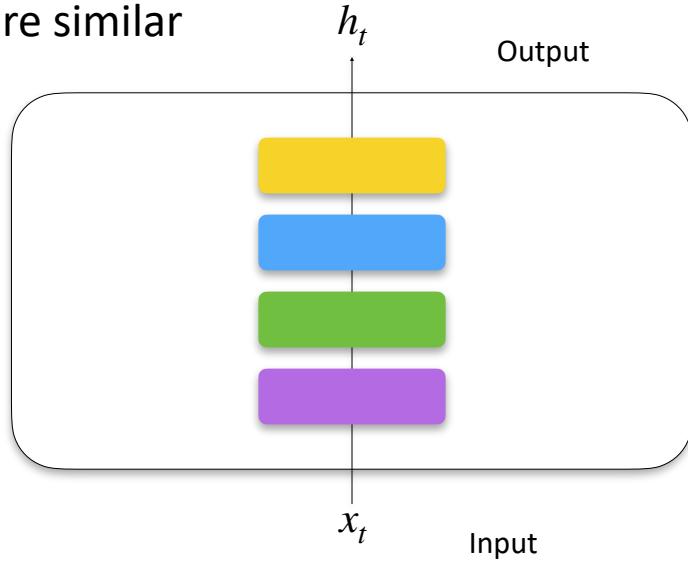


$$h_t = f(x_t)$$

Feed Forward Networks

- Information flowing from the inputs to the outputs
- Classical NNs are similar

Conceptually similar to
[ARIMA](#), [G/ARCH](#), etc



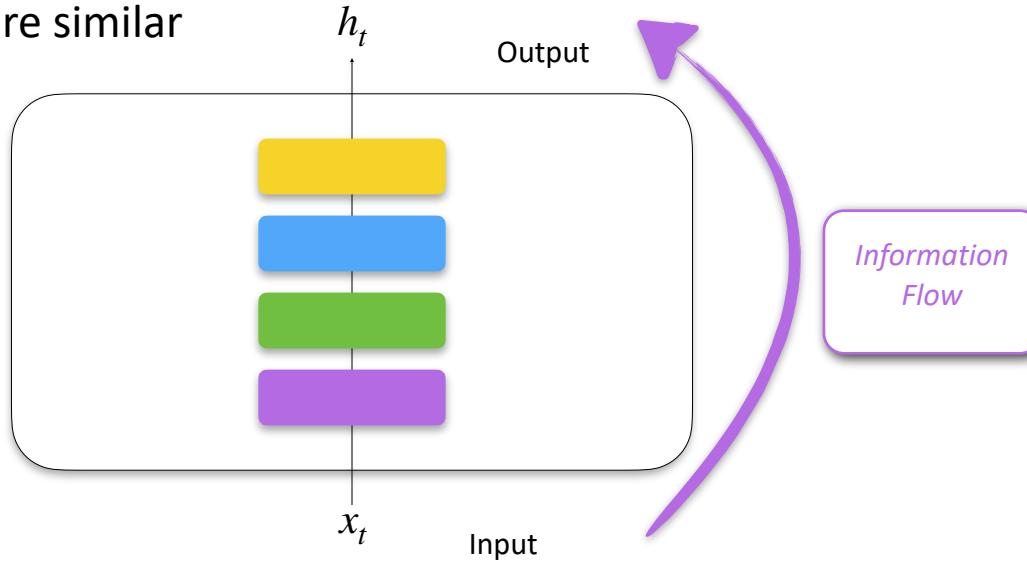
$$h_t = f(x_t)$$

Feed Forward Networks

- Information flowing from the inputs to the outputs
- Classical NNs are similar

Conceptually similar to
ARIMA, G/ARCH, etc

$$h_t = f(x_t)$$





Deep Learning Approaches

12.1 Feed Forward Networks (FFN)

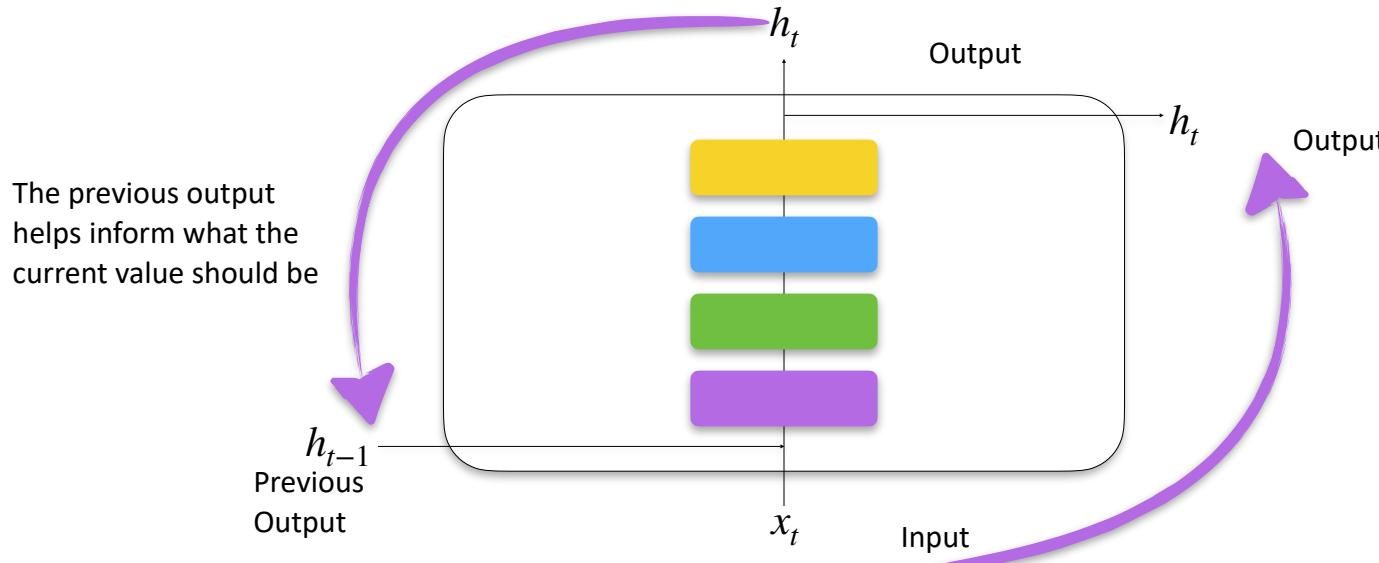
[**12.2 Recurrent Neural Networks \(RNN\)**](#)

12.3 Gated Recurrent Units (GRU)

12.4 Long Short-Term Memory (LSTM)

Recurrent Neural Network (RNN)

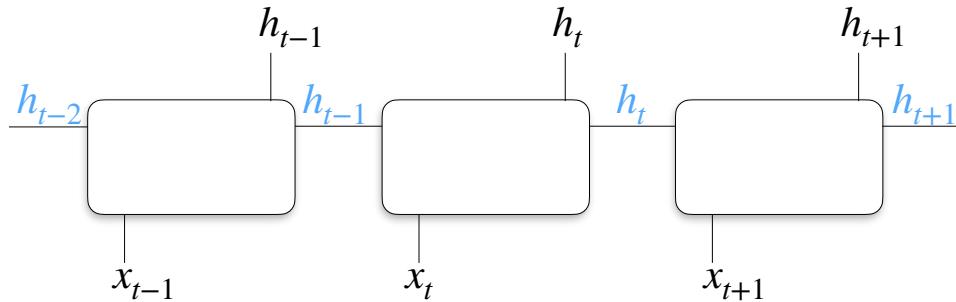
- RNNs follow a different paradigm



$$h_t = f(x_t, h_{t-1})$$

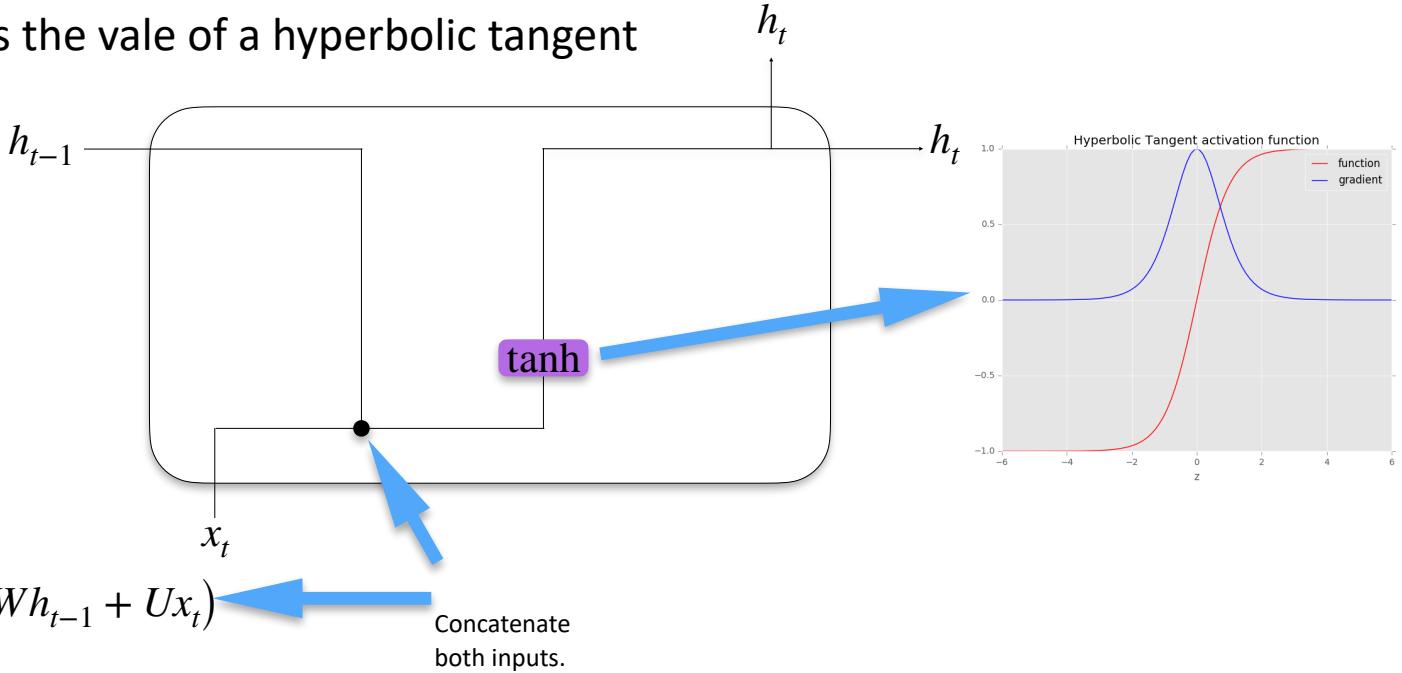
Recurrent Neural Network (RNN)

- Each output depends (implicitly) on all previous outputs.
- Input sequences generate output sequences (**seq2seq**)

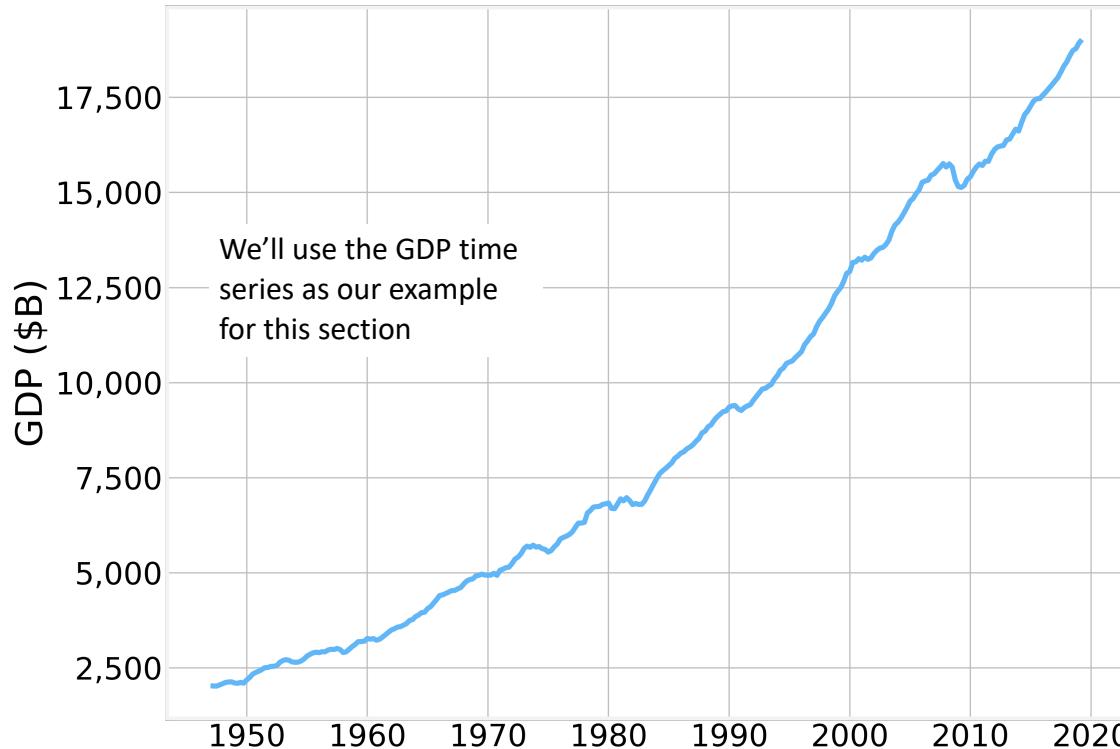


Recurrent Neural Network (RNN)

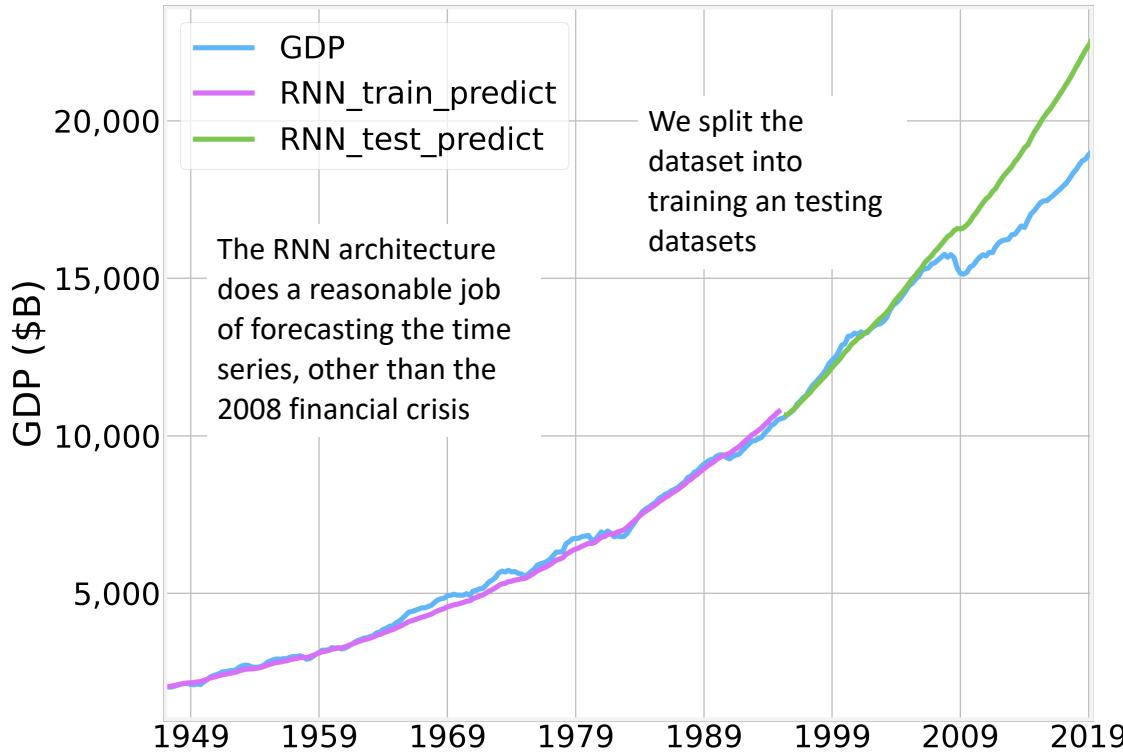
- Simplest possible RNN
- Outputs the value of a hyperbolic tangent



Recurrent Neural Network (RNN)



Recurrent Neural Network (RNN)





Deep Learning Approaches

12.1 Feed Forward Networks (FFN)

12.2 Recurrent Neural Networks (RNN)

12.3 Gated Recurrent Units (GRU)

12.4 Long Short-Term Memory (LSTM)

Gated Recurrent Unit (GRU)

- Introduced in 2014 by K. Cho
- Meant to solve the Vanishing Gradient Problem
- Can be considered as a simplification of LSTMs
- Similar performance to LSTM in some applications, better performance for smaller datasets.

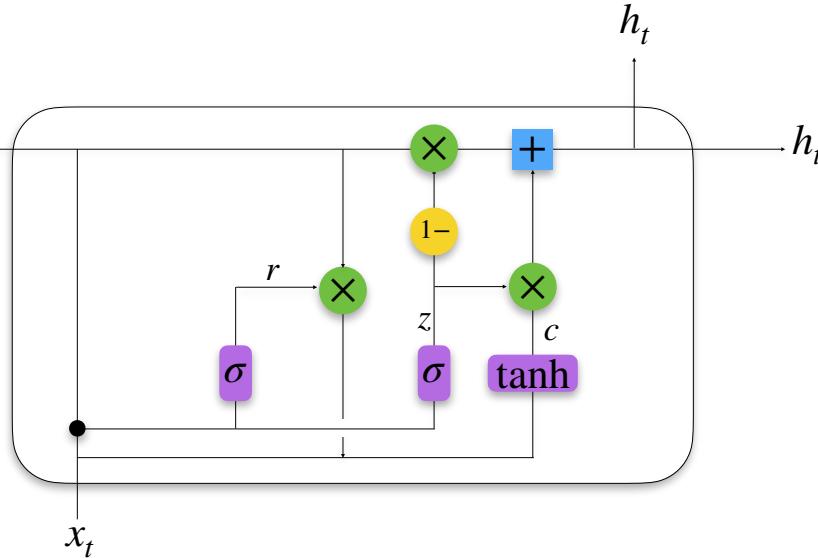
Gated Recurrent Unit (GRU)

$$z = \sigma(W_z h_{t-1} + U_z x_t)$$

$$r = \sigma(W_r h_{t-1} + U_r x_t)$$

$$c = \tanh(W_c (h_{t-1} \otimes r) + U_c x_t)$$

$$h_t = (z \otimes c) + ((1 - z) \otimes h_{t-1})$$



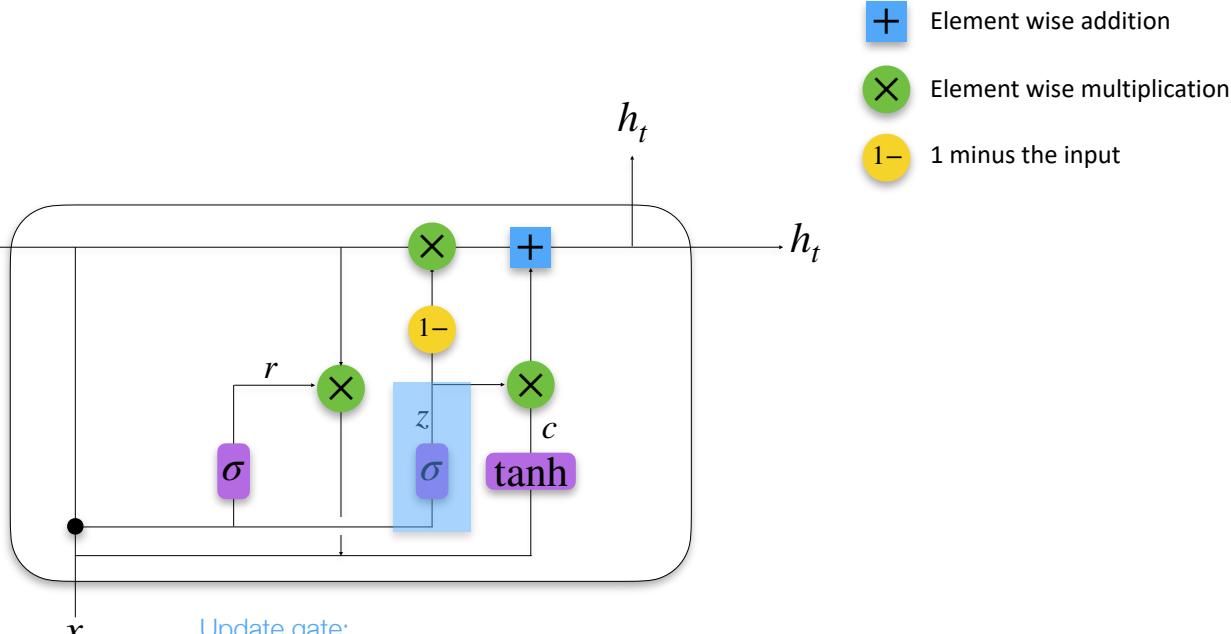
Gated Recurrent Unit (GRU)

$$z = \sigma(W_z h_{t-1} + U_z x_t)$$

$$r = \sigma(W_r h_{t-1} + U_r x_t)$$

$$c = \tanh(W_c (h_{t-1} \otimes r) + U_c x_t)$$

$$h_t = (z \otimes c) + ((1 - z) \otimes h_{t-1})$$



Update gate:
How much of
the previous
state should be
removed?

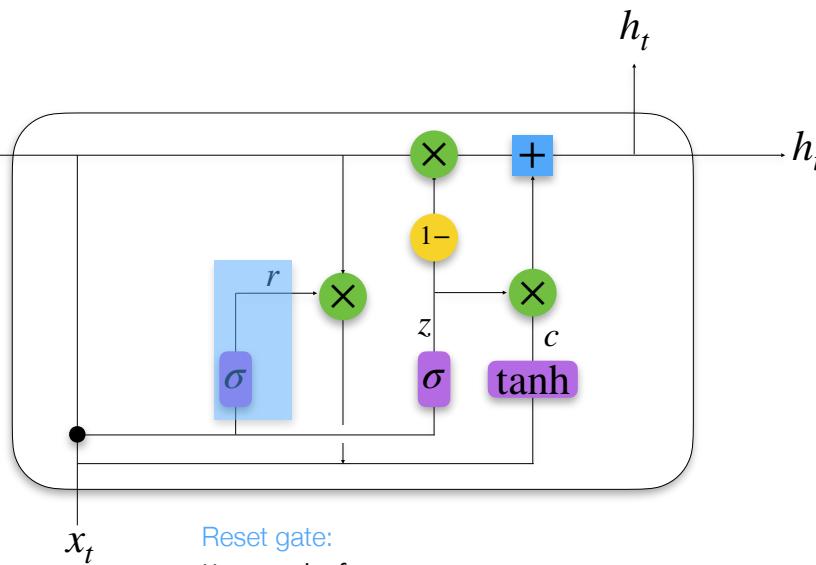
Gated Recurrent Unit (GRU)

$$z = \sigma(W_z h_{t-1} + U_z x_t)$$

$$r = \sigma(W_r h_{t-1} + U_r x_t)$$

$$c = \tanh(W_c (h_{t-1} \otimes r) + U_c x_t)$$

$$h_t = (z \otimes c) + ((1 - z) \otimes h_{t-1})$$



Reset gate:
How much of
the previous
output
should be
removed?

⊕ Element wise addition

⊗ Element wise multiplication

⊖ 1 minus the input

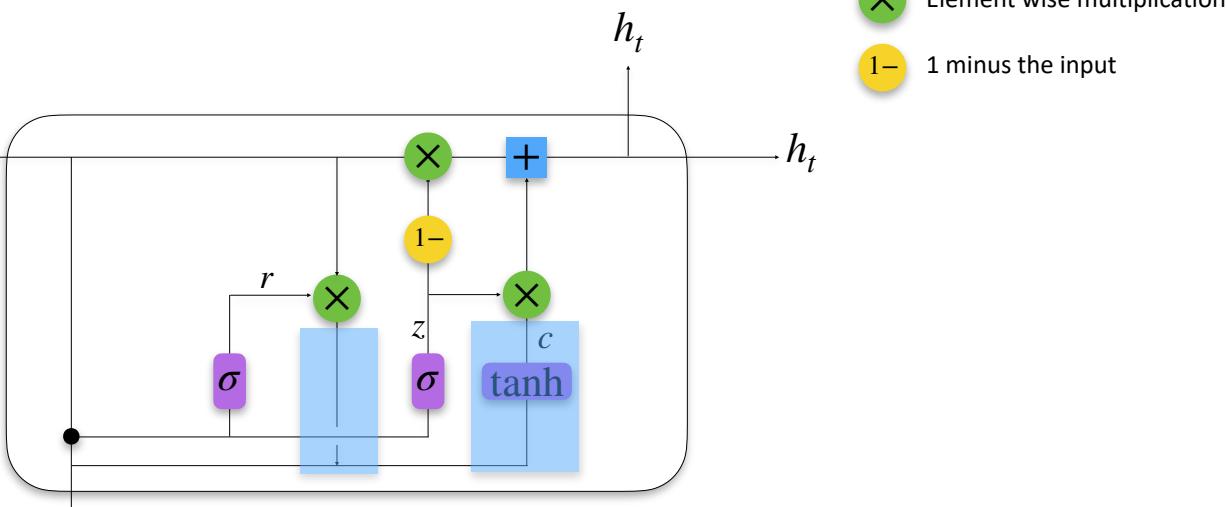
Gated Recurrent Unit (GRU)

$$z = \sigma(W_z h_{t-1} + U_z x_t)$$

$$r = \sigma(W_r h_{t-1} + U_r x_t)$$

$$c = \tanh(W_c (h_{t-1} \otimes r) + U_c x_t)$$

$$h_t = (z \otimes c) + ((1 - z) \otimes h_{t-1})$$



Current
memory:

What
information do
we remember
right now?

⊕ Element wise addition

⊗ Element wise multiplication

⊖ 1 minus the input

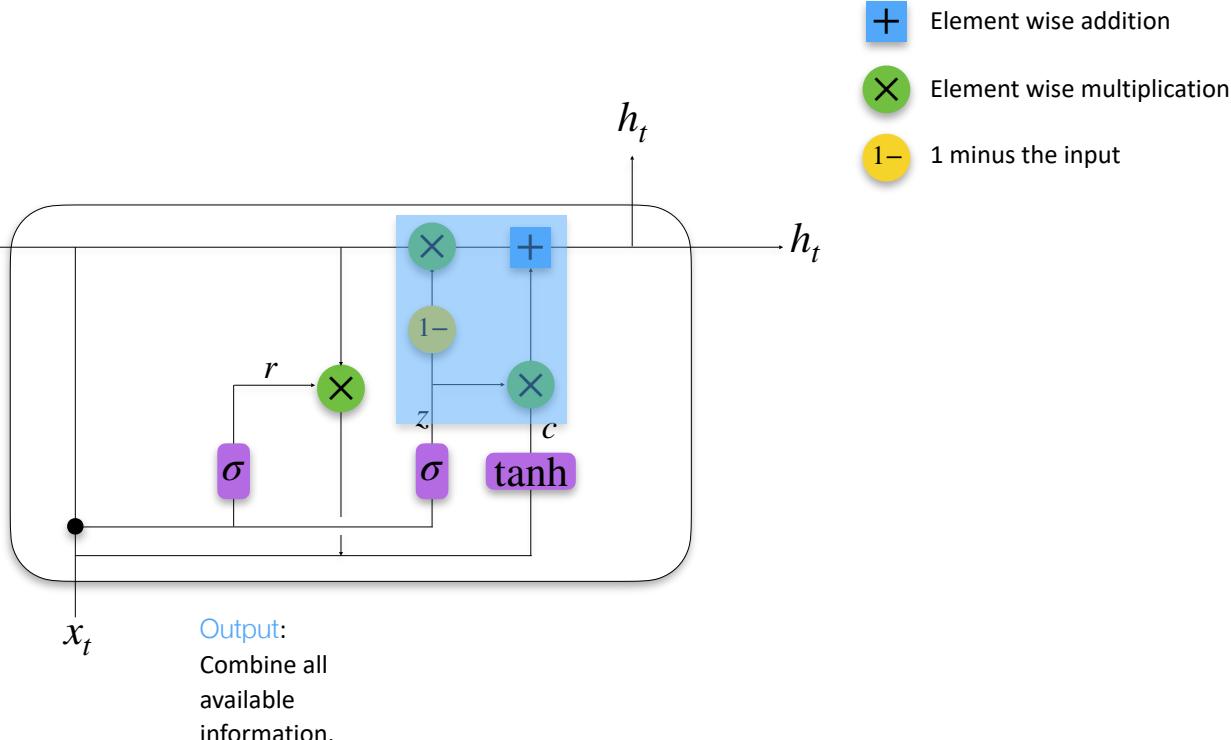
Gated Recurrent Unit (GRU)

$$z = \sigma(W_z h_{t-1} + U_z x_t)$$

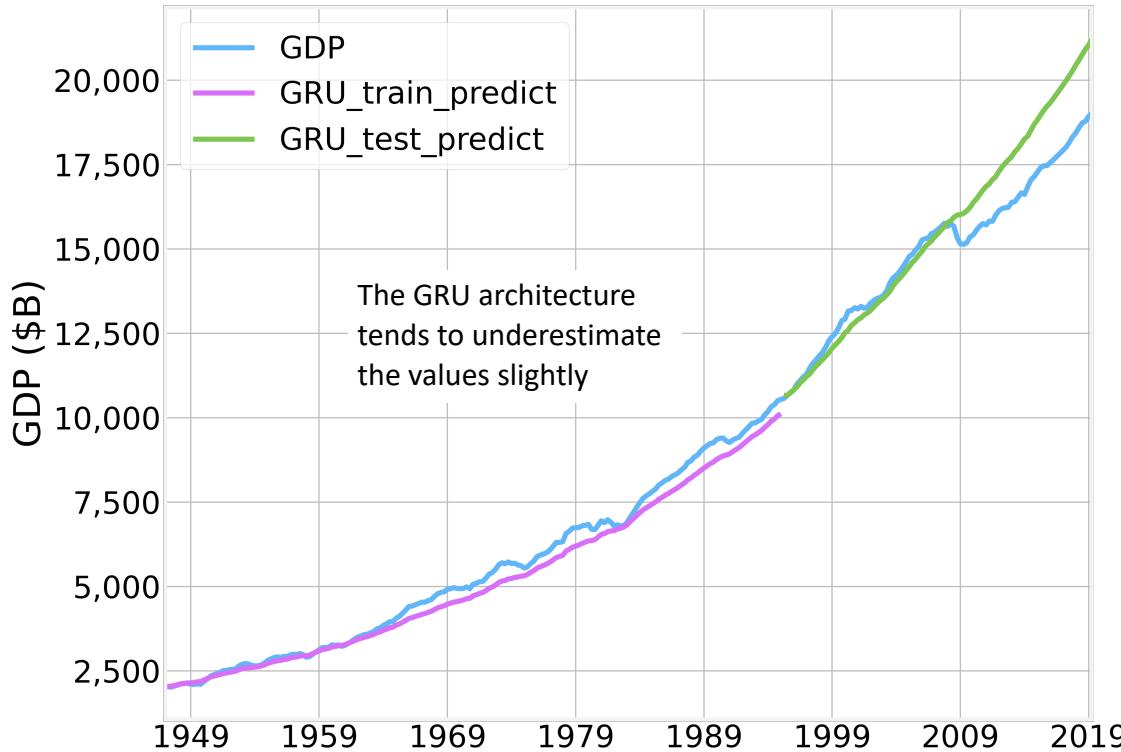
$$r = \sigma(W_r h_{t-1} + U_r x_t)$$

$$c = \tanh(W_c (h_{t-1} \otimes r) + U_c x_t)$$

$$h_t = (z \otimes c) + ((1 - z) \otimes h_{t-1})$$



Gated Recurrent Unit (GRU)





Deep Learning Approaches

12.1 Feed Forward Networks (FFN)

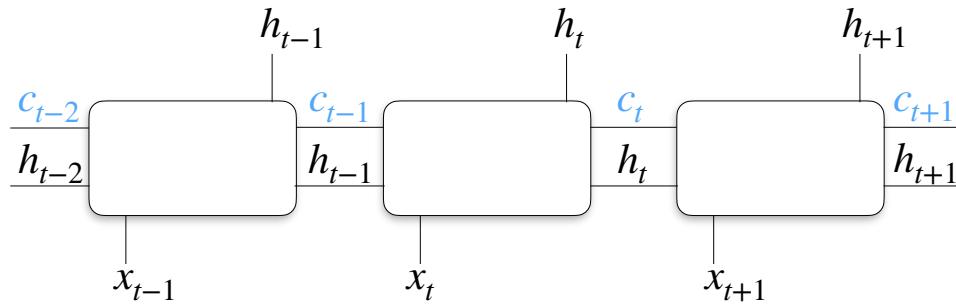
12.2 Recurrent Neural Networks (RNN)

12.3 Gated Recurrent Units (GRU)

12.4 Long Short-Term Memory (LSTM)

Long-Short Term Memory (LSTM)

- What if we want to keep explicit information about previous states (memory)?
- How much information is kept, can be controlled through gates.
- First introduced in 1997 by Hochreiter and Schmidhuber



Long-Short Term Memory (LSTM)

$$f = \sigma(W_f h_{t-1} + U_f x_t)$$

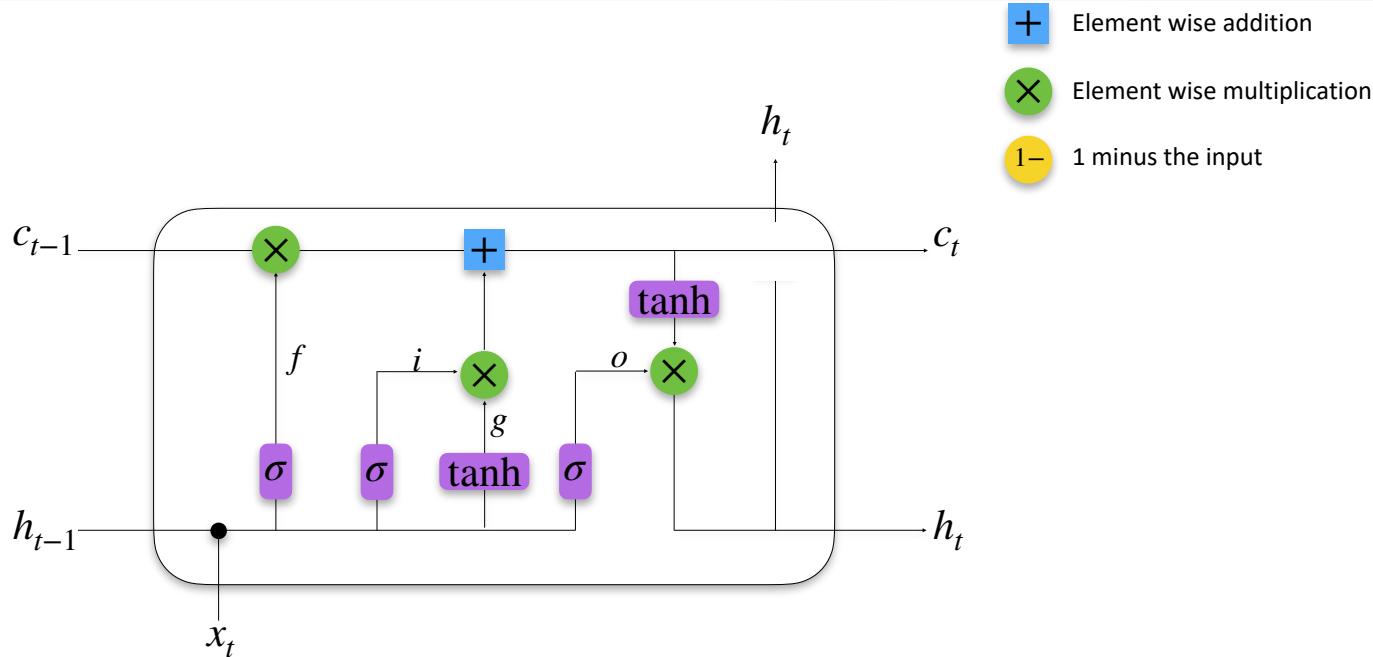
$$i = \sigma(W_i h_{t-1} + U_i x_t)$$

$$o = \sigma(W_o h_{t-1} + U_o x_t)$$

$$g = \tanh(W_g h_{t-1} + U_g x_t)$$

$$c_t = (c_{t-1} \otimes f) + (g \otimes i)$$

$$h_t = \tanh(c_t) \otimes o$$



Long-Short Term Memory (LSTM)

$$f = \sigma(W_f h_{t-1} + U_f x_t)$$

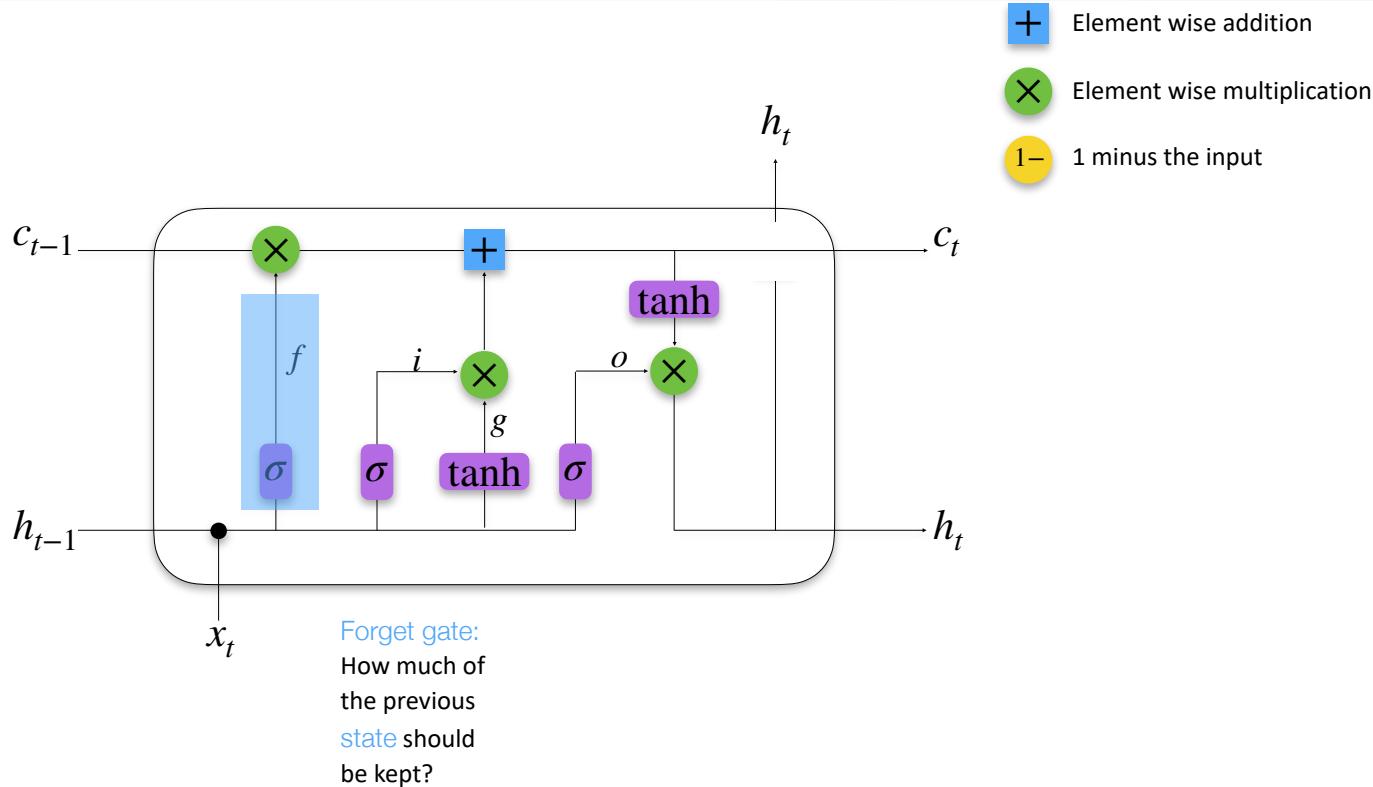
$$i = \sigma(W_i h_{t-1} + U_i x_t)$$

$$o = \sigma(W_o h_{t-1} + U_o x_t)$$

$$g = \tanh(W_g h_{t-1} + U_g x_t)$$

$$c_t = (c_{t-1} \otimes f) + (g \otimes i)$$

$$h_t = \tanh(c_t) \otimes o$$



Long-Short Term Memory (LSTM)

$$f = \sigma(W_f h_{t-1} + U_f x_t)$$

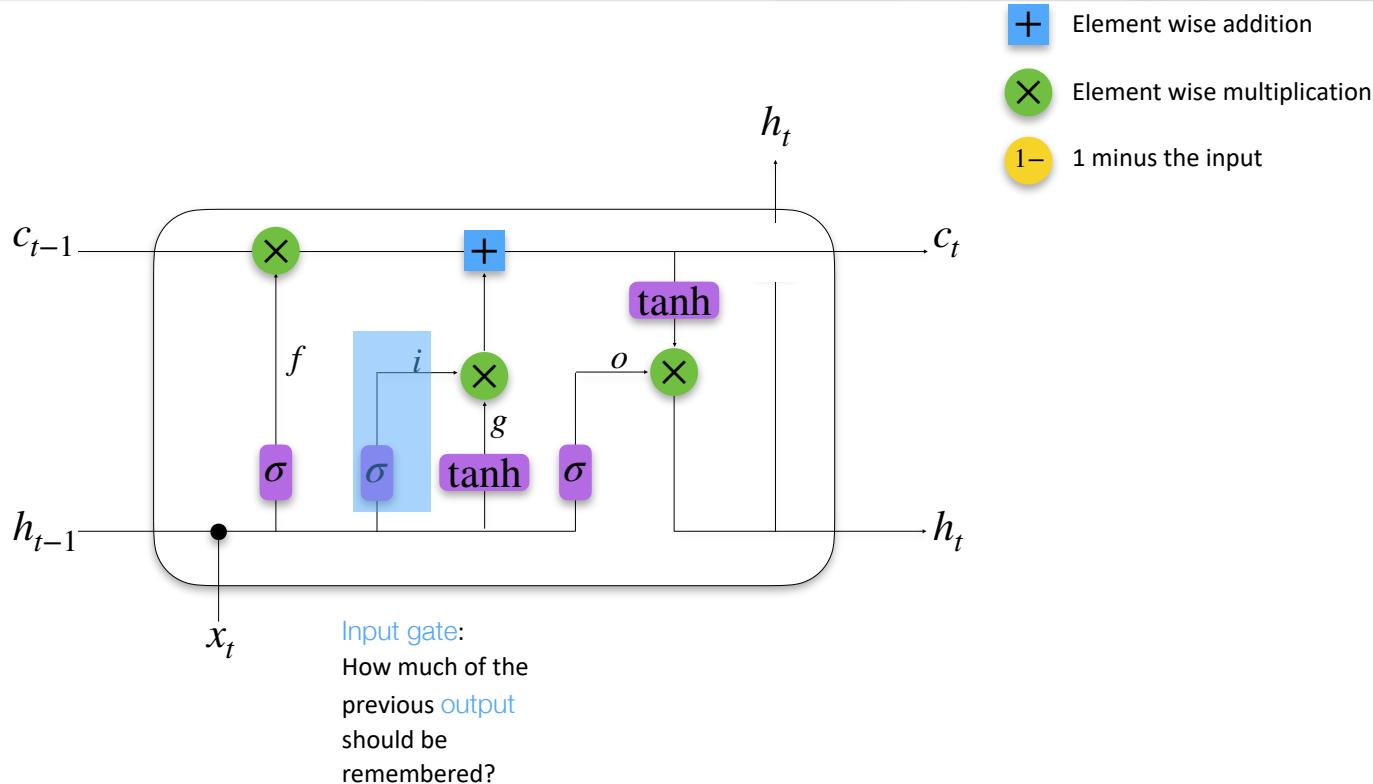
$$i = \sigma(W_i h_{t-1} + U_i x_t)$$

$$o = \sigma(W_o h_{t-1} + U_o x_t)$$

$$g = \tanh(W_g h_{t-1} + U_g x_t)$$

$$c_t = (c_{t-1} \otimes f) + (g \otimes i)$$

$$h_t = \tanh(c_t) \otimes o$$



Long-Short Term Memory (LSTM)

$$f = \sigma(W_f h_{t-1} + U_f x_t)$$

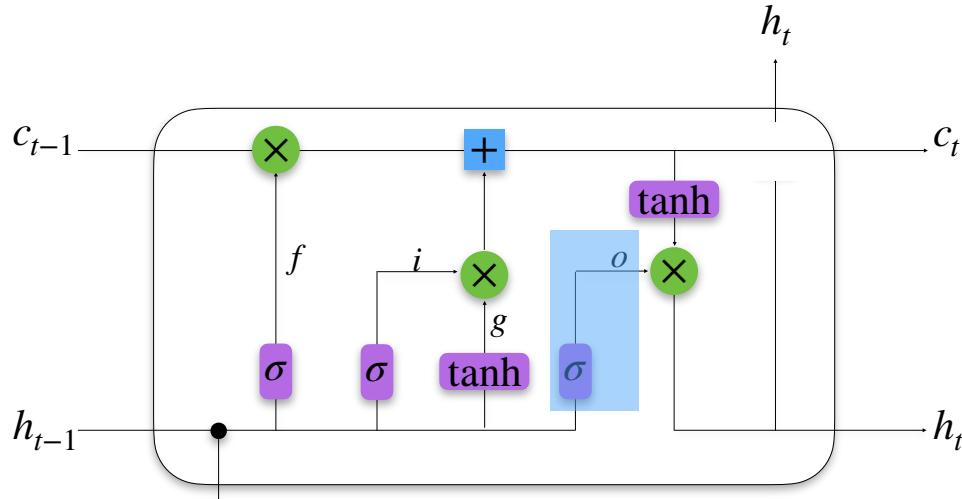
$$i = \sigma(W_i h_{t-1} + U_i x_t)$$

$$o = \sigma(W_o h_{t-1} + U_o x_t)$$

$$g = \tanh(W_g h_{t-1} + U_g x_t)$$

$$c_t = (c_{t-1} \otimes f) + (g \otimes i)$$

$$h_t = \tanh(c_t) \otimes o$$



Output gate:
How much of
the previous
output
should
contribute?

+ Element wise addition

× Element wise multiplication

1- 1 minus the input

All gates use
the same
inputs and
activation
functions,
but different
weights

Long-Short Term Memory (LSTM)

$$f = \sigma(W_f h_{t-1} + U_f x_t)$$

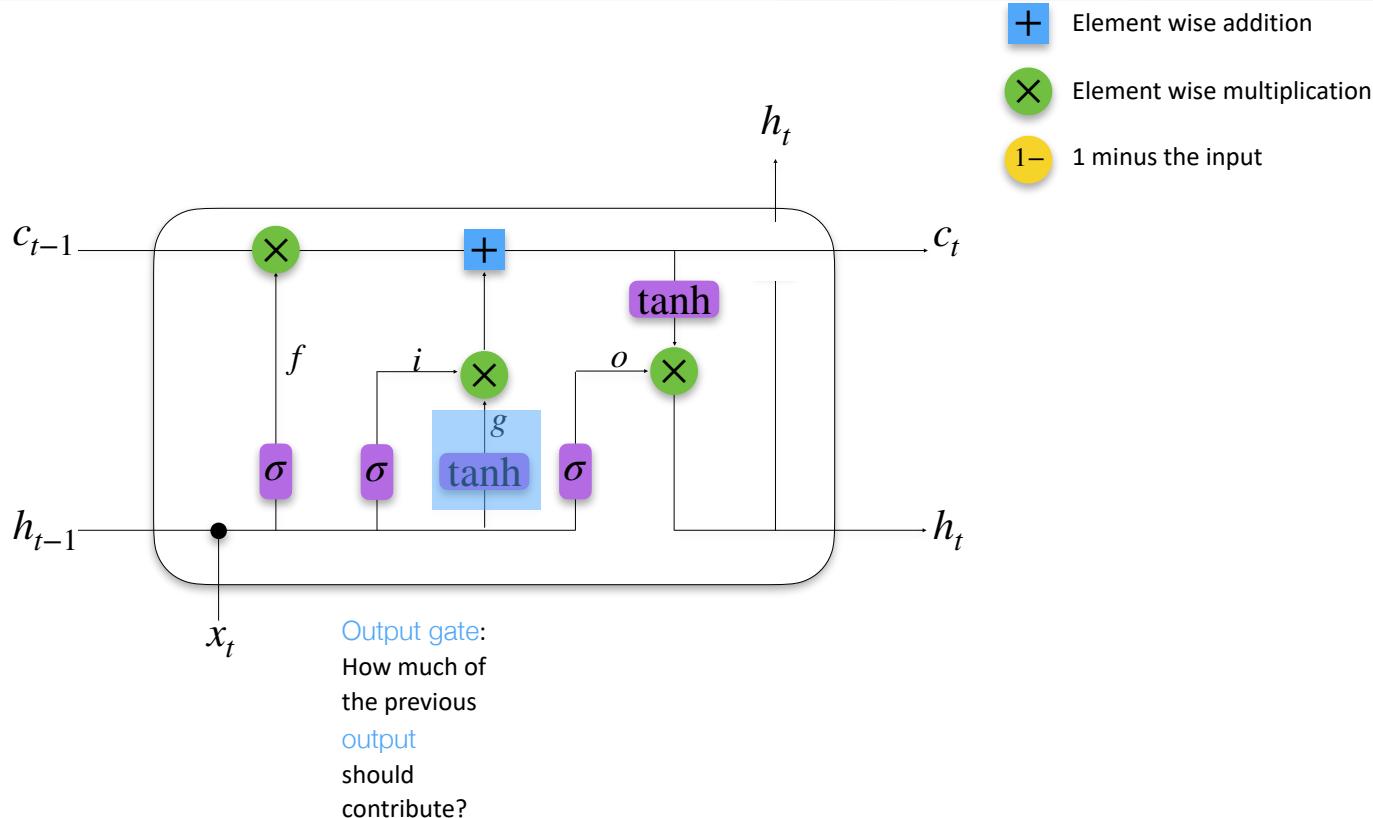
$$i = \sigma(W_i h_{t-1} + U_i x_t)$$

$$o = \sigma(W_o h_{t-1} + U_o x_t)$$

$$g = \tanh(W_g h_{t-1} + U_g x_t)$$

$$c_t = (c_{t-1} \otimes f) + (g \otimes i)$$

$$h_t = \tanh(c_t) \otimes o$$



Long-Short Term Memory (LSTM)

$$f = \sigma(W_f h_{t-1} + U_f x_t)$$

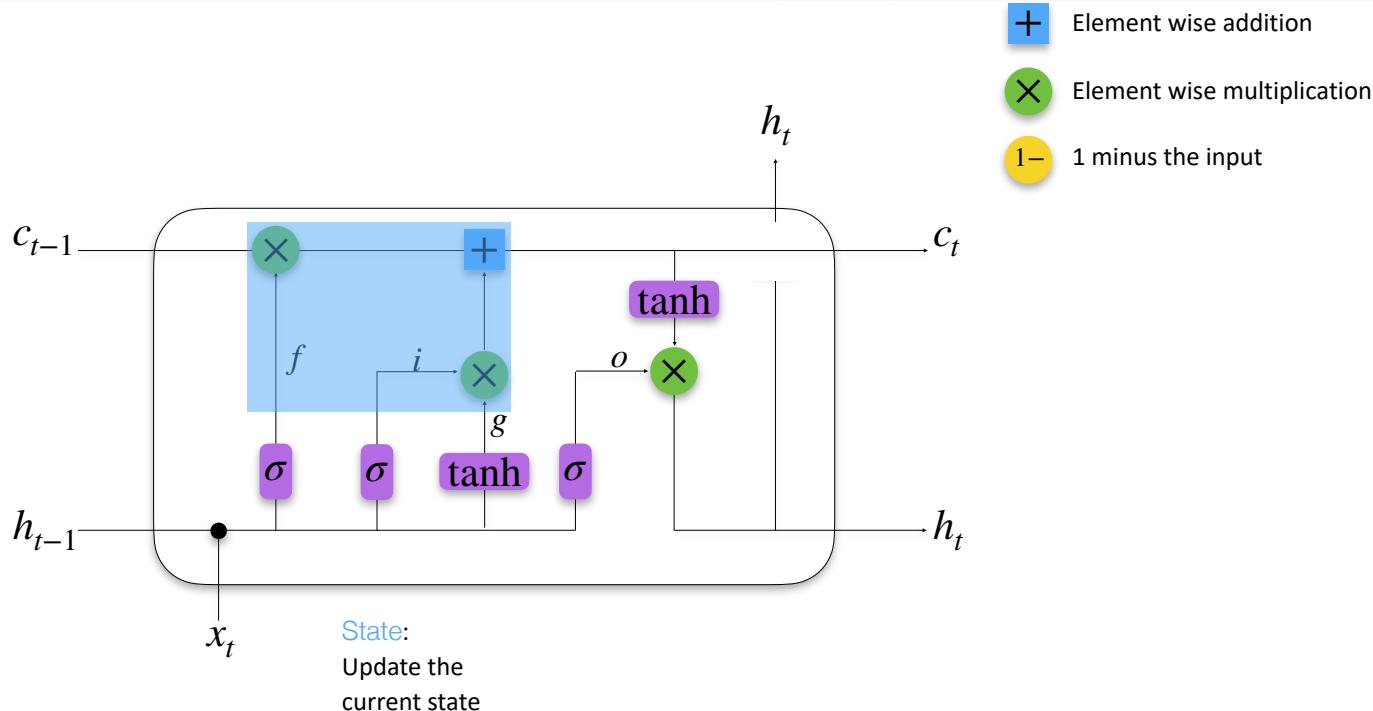
$$i = \sigma(W_i h_{t-1} + U_i x_t)$$

$$o = \sigma(W_o h_{t-1} + U_o x_t)$$

$$g = \tanh(W_g h_{t-1} + U_g x_t)$$

$$c_t = (c_{t-1} \otimes f) + (g \otimes i)$$

$$h_t = \tanh(c_t) \otimes o$$



Long-Short Term Memory (LSTM)

$$f = \sigma(W_f h_{t-1} + U_f x_t)$$

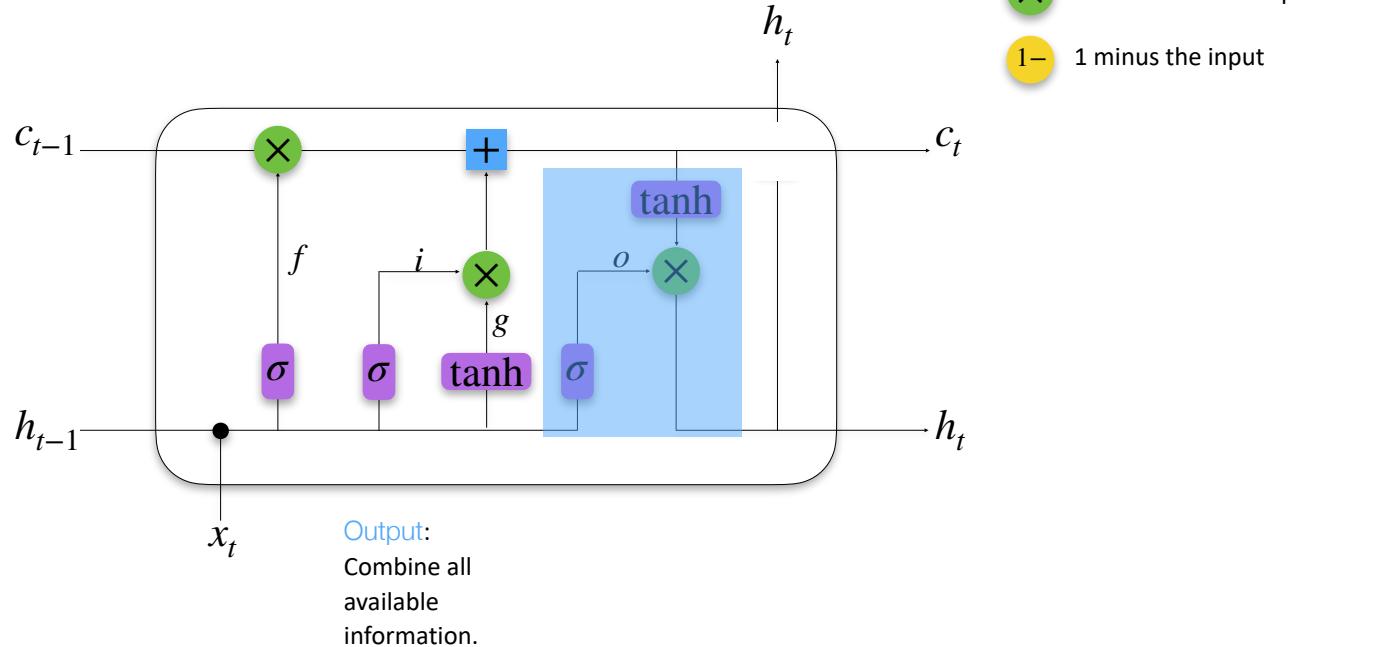
$$i = \sigma(W_i h_{t-1} + U_i x_t)$$

$$o = \sigma(W_o h_{t-1} + U_o x_t)$$

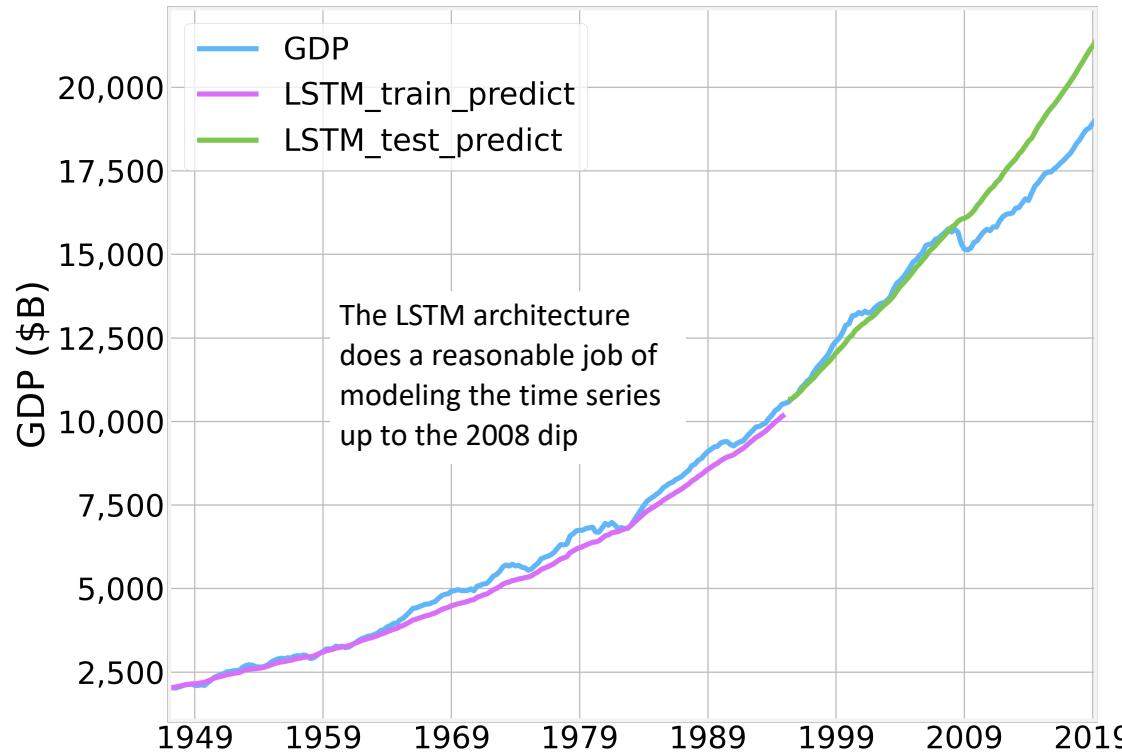
$$g = \tanh(W_g h_{t-1} + U_g x_t)$$

$$c_t = (c_{t-1} \otimes f) + (g \otimes i)$$

$$h_t = \tanh(c_t) \otimes o$$



Long-Short Term Memory (LSTM)





Code - Deep Learning

https://github.com/DataForScience/Timeseries_LL



Thank you!

Bruno Gonçalves

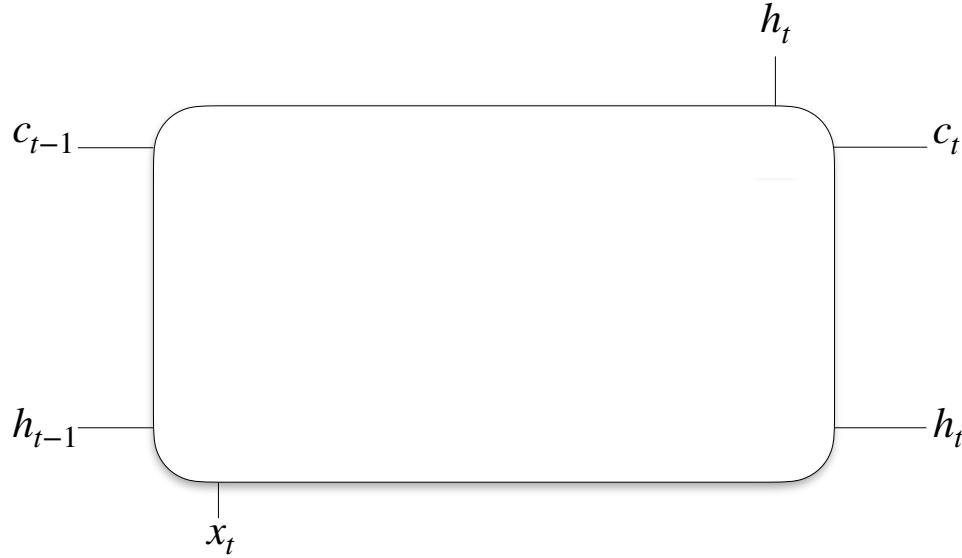
www.data4sci.com/newsletter

https://github.com/DataForScience/Timeseries_LL

Applications

- Language Modeling and Prediction
- Speech Recognition
- Machine Translation
- Part-of-Speech Tagging
- Sentiment Analysis
- Summarization
- Time series forecasting

Neural Networks?



Or legos?



Keras

- Open Source neural network library written in Python
- TensorFlow, Microsoft Cognitive Toolkit or Theano backends
- Enables fast experimentation
- Created and maintained by François Chollet, a Google engineer.
- Implements Layers, Objective/Loss functions, Activation functions, Optimizers, etc...

Keras

- `keras.models.Sequential(layers=None, name=None)`- is the workhorse. You use it to build a model layer by layer. Returns the object that we will use to build the model
- `keras.layers`
 - `Dense(units, activation=None, use_bias=True)` - `None` means linear activation. Other options are, '`tanh`', '`sigmoid`', '`softmax`', '`relu`', etc.
 - `Dropout(rate, seed=None)`
 - `Activation(activation)` - Same as the activation option to `Dense`, can also be used to pass `TensorFlow` or `Theano` operations directly.
 - `SimpleRNN(units, input_shape, activation='tanh', use_bias=True,`

Keras

- `model = Sequential()`
- `model.add(layer)` - Add a layer to the top of the model
- `model.compile(optimizer, loss)` - We have to compile the model before we can use it
 - `optimizer` - ‘adam’, ‘sgd’, ‘rmsprop’, etc...
 - `loss` - ‘mean_squared_error’, ‘categorical_crossentropy’, ‘kullback_leibler_divergence’, etc...
- `model.fit(x=None, y=None, batch_size=None, epochs=1, verbose=1,`