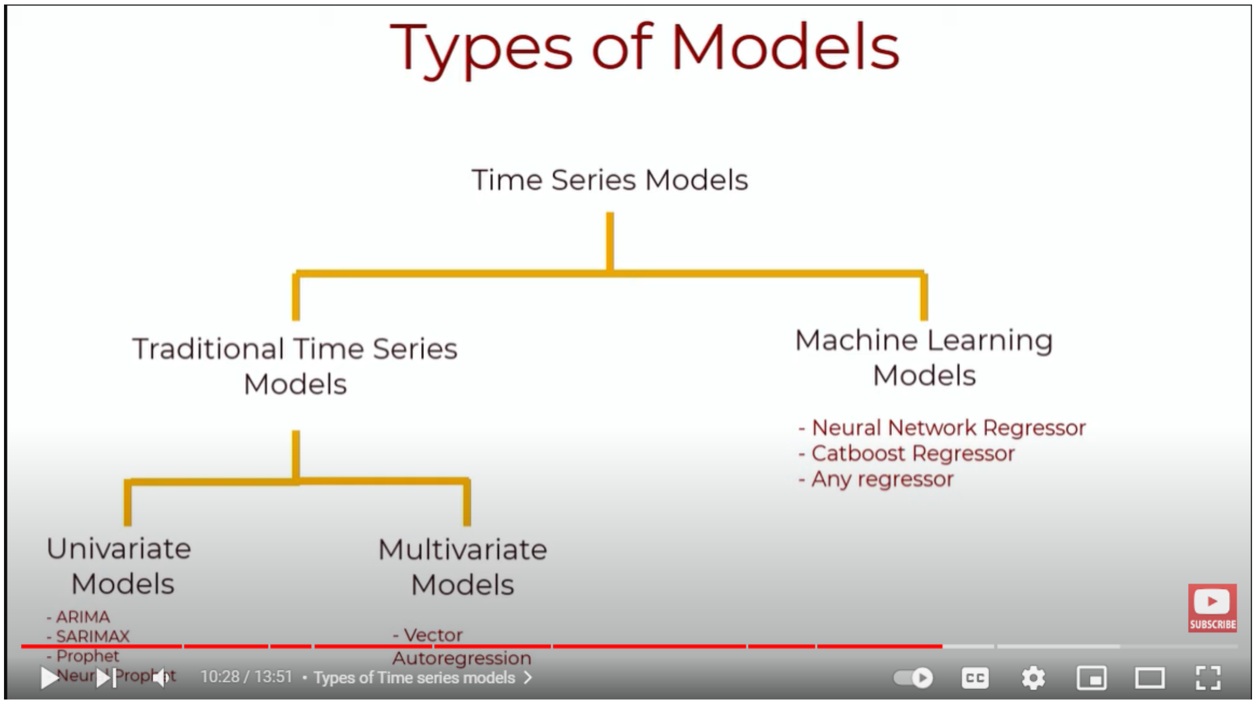
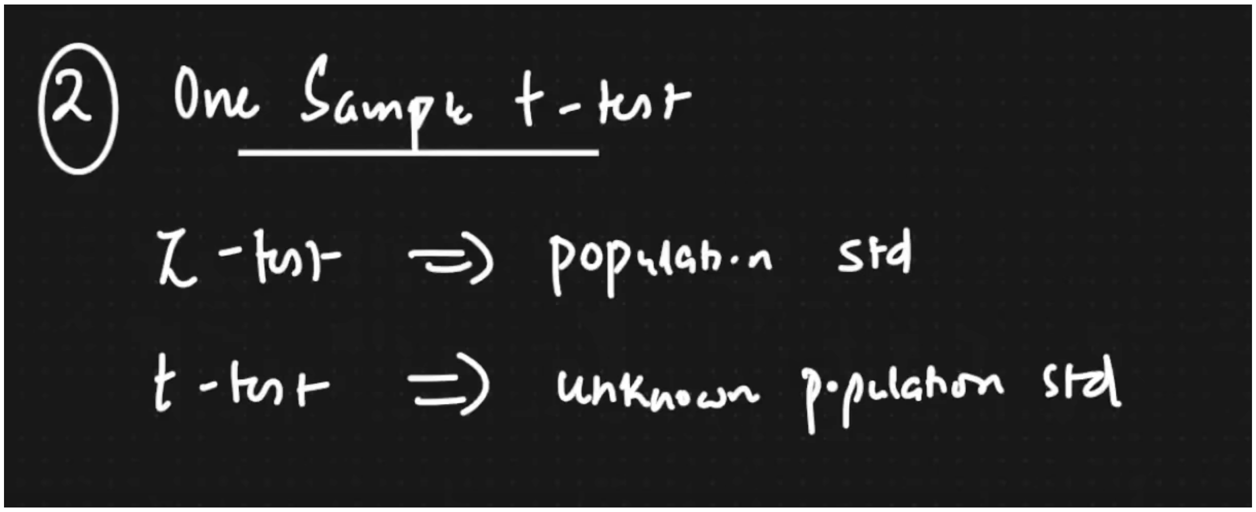
Time series

Traditional Time Series Techniques:



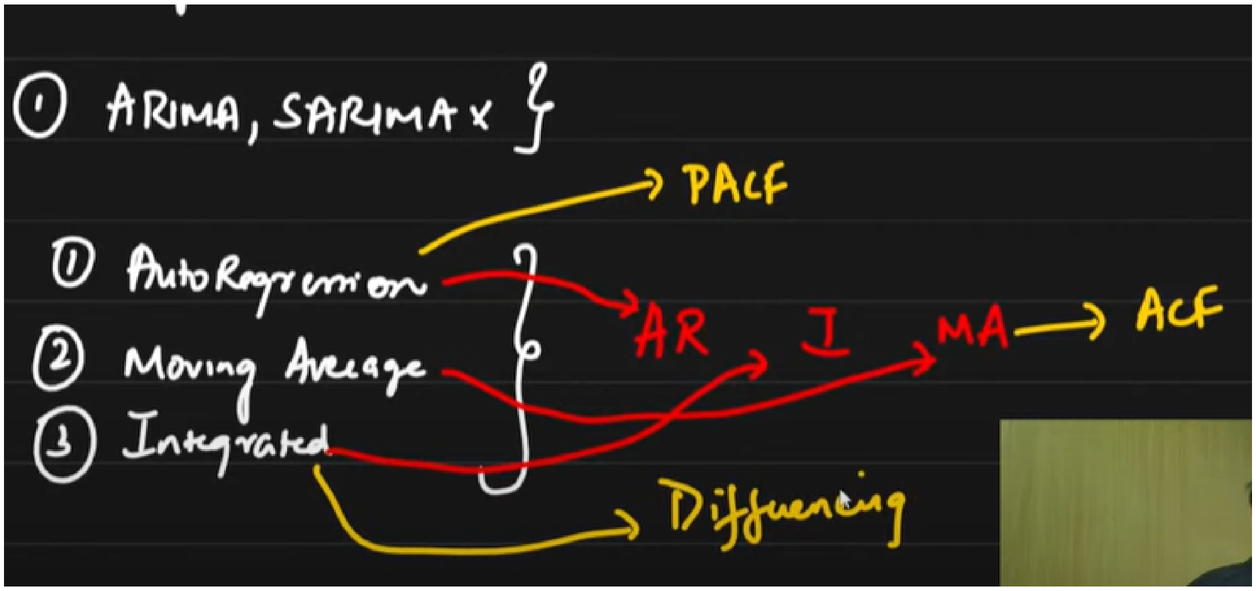


② One Sample t-test

Z-test ⟹ population std

t-test ⟹ unknown population std

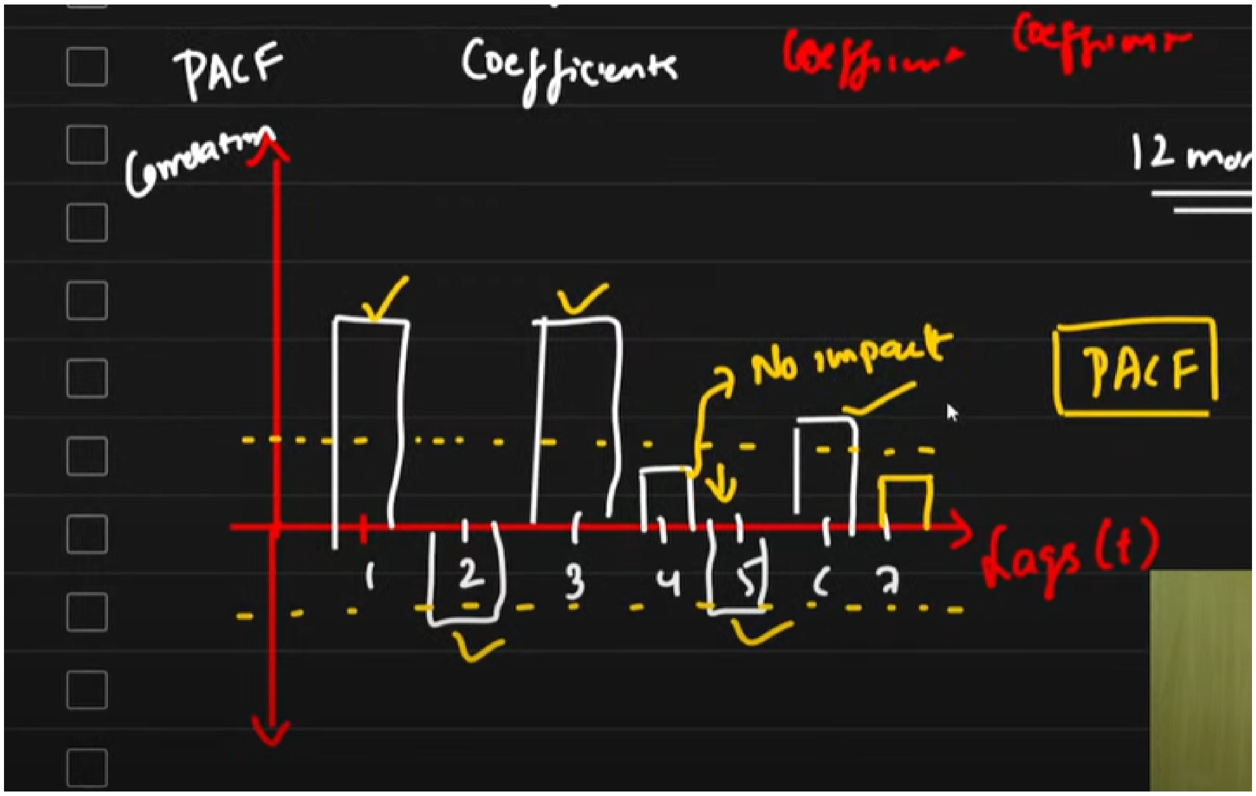
Time series



Auto Regression made by **PACF plot**
Moving Average made by **ACF plot**
Integrated made by Differencing

Time series

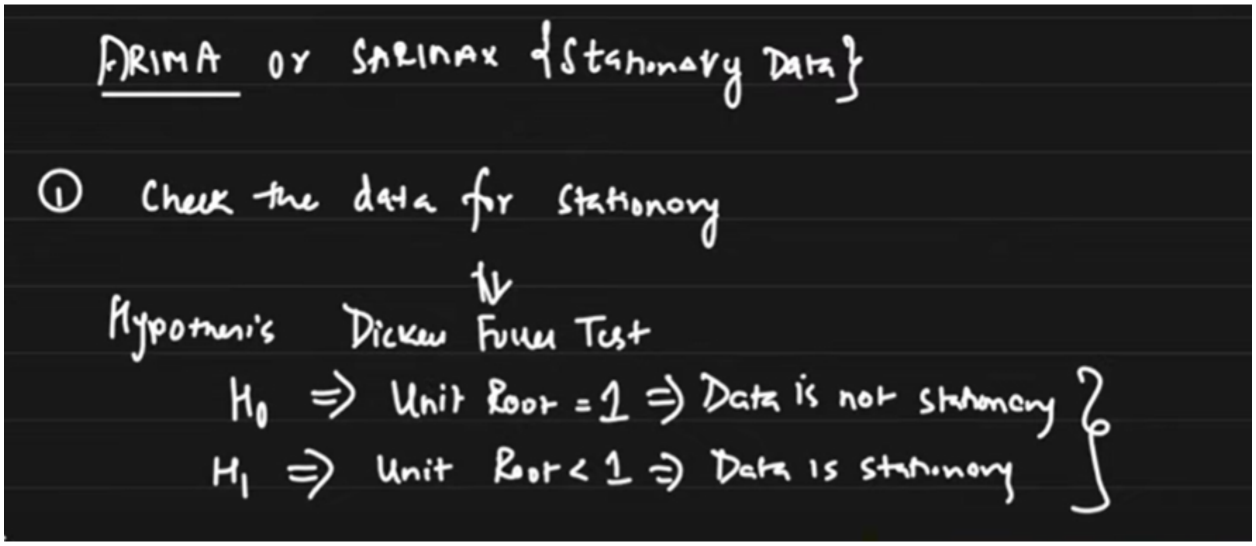
PACF PLOT :



This plot helps us to find out which graph have impact on our sells , for example 4&7 have no impact but 1,2,3,5,6 have impact

PACF in general helps us to check the correlation

ARIMA steps:



ARIMA or SARIMAX {Stationary Data}

① Check the data for Stationary

$\Downarrow$

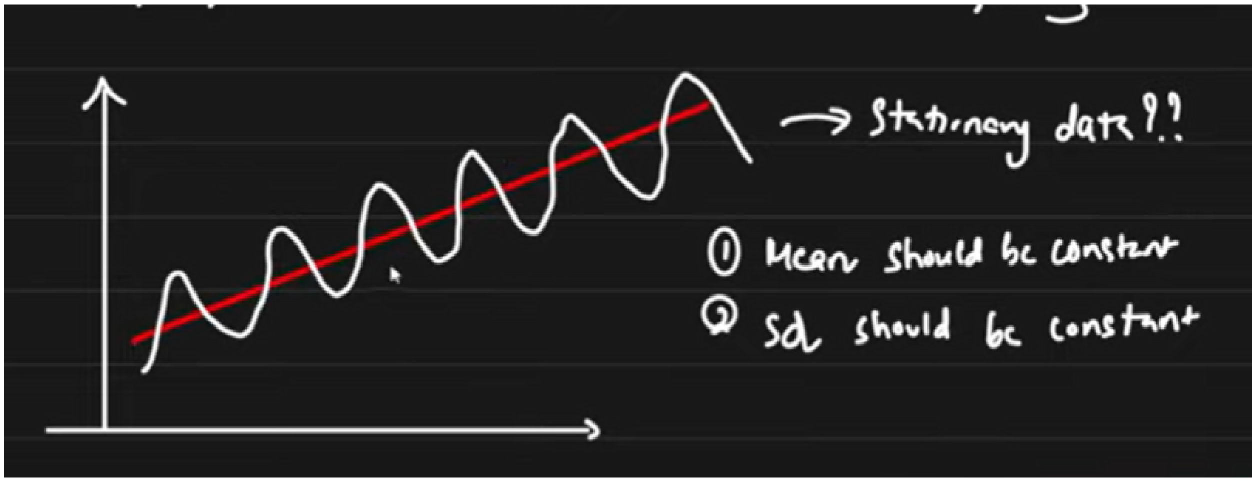
Hypothesis Dickey Fuller Test

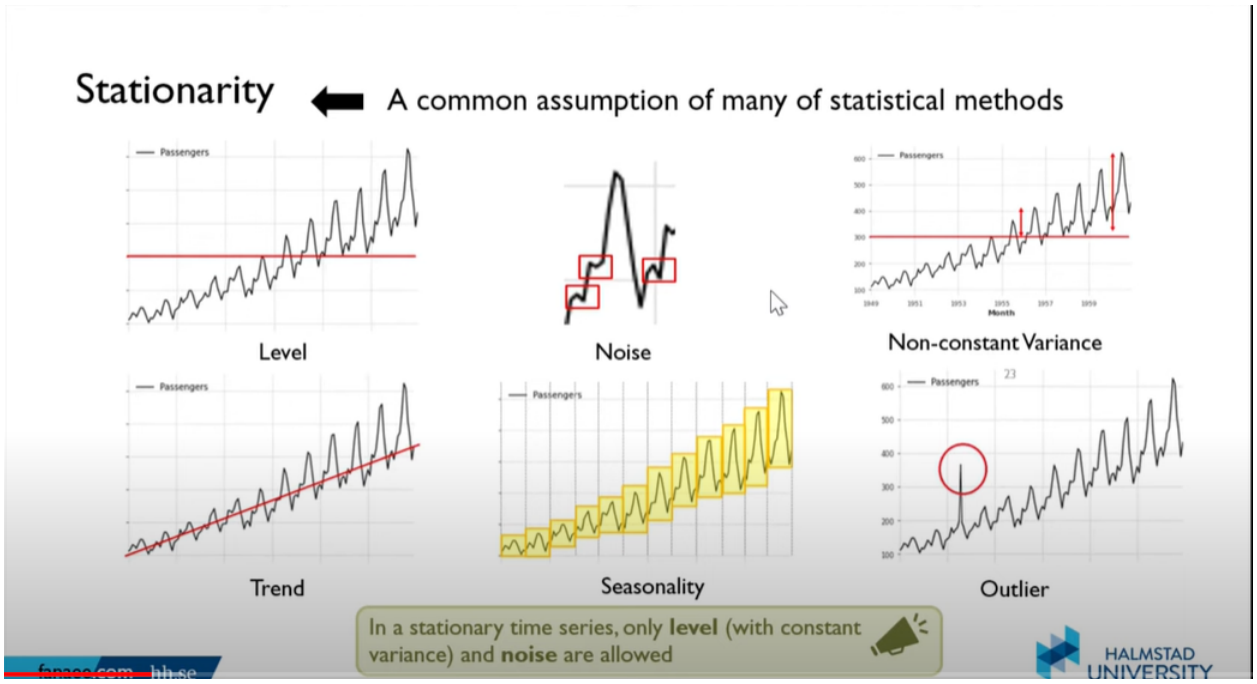
$H_0 \Rightarrow$ Unit Root $= 1 \Rightarrow$ Data is not stationary

$H_1 \Rightarrow$ Unit Root $< 1 \Rightarrow$ Data is stationary

Time series

Check if data is stationary:





Stationarity

A common assumption of many of statistical methods

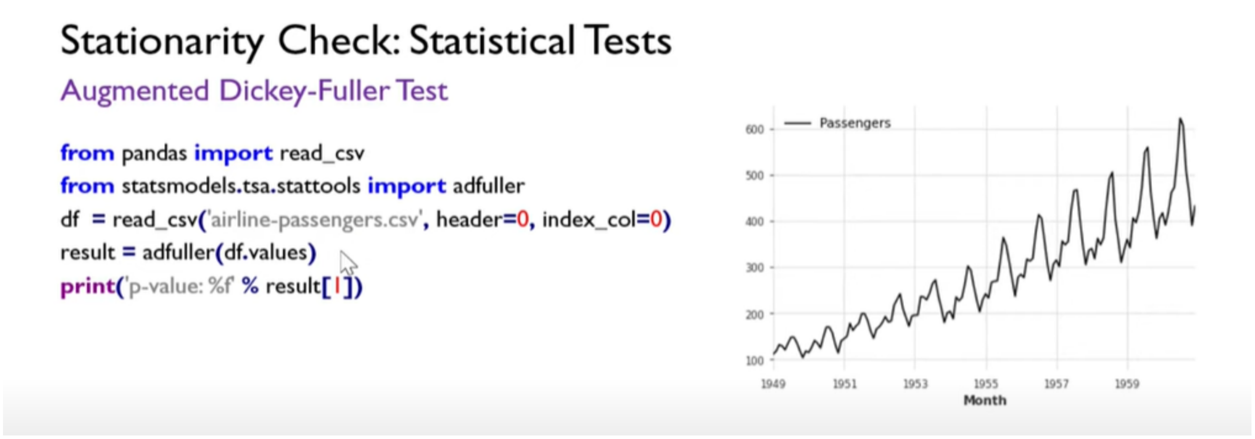
In a stationary time series, only **level** (with constant variance) and **noise** are allowed

HALMSTAD UNIVERSITY

Stationary we are only allowed to have constant variance and noises

Time series

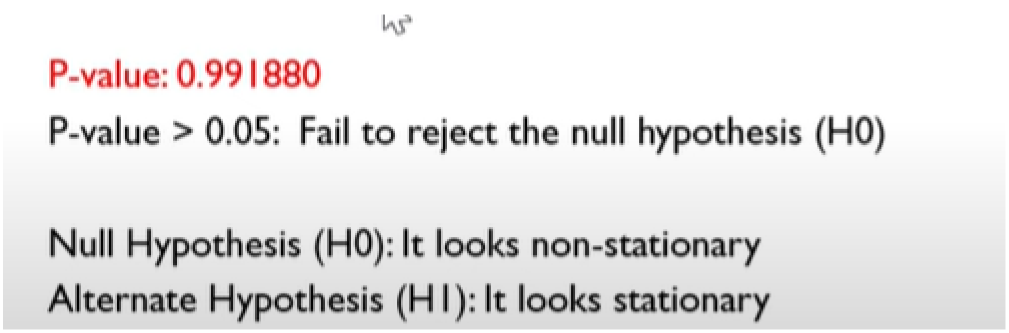
1 / Stationary check :



Stationarity Check: Statistical Tests

Augmented Dickey-Fuller Test

```
from pandas import read_csv
from statsmodels.tsa.stattools import adfuller
df  = read_csv('airline-passengers.csv', header=0, index_col=0)
result = adfuller(df.values)
print('p-value: %f' % result[1])
```



P-value: 0.991880

P-value > 0.05: Fail to reject the null hypothesis (H0)

Null Hypothesis (H0): It looks non-stationary

Alternate Hypothesis (H1): It looks stationary

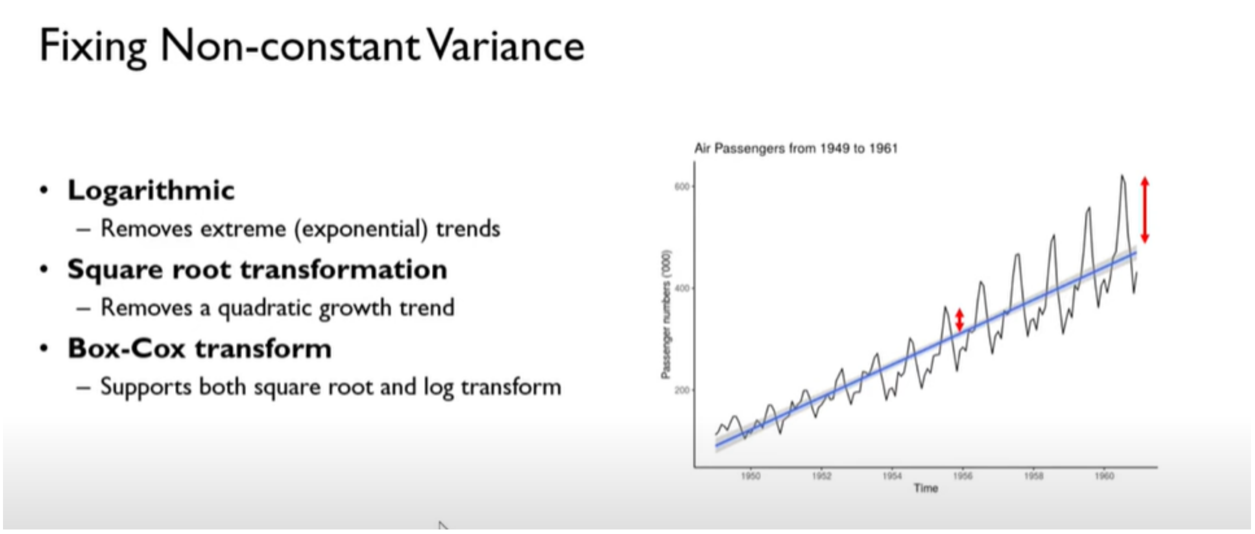
Implementation python :

```
def adf_test(series):
    result=adfuller(series)
    print('ADF Statistics: {}'.format(result[0]))
    print('p- value: {}'.format(result[1]))
    if result[1] <= 0.05:
        print("strong evidence against the null hypothesis, reject the 
null hypothesis. Data has no unit root and is stationary")
    else:
        print("weak evidence against null hypothesis, time series has a 
unit root, indicating it is non-stationary ")
```

```
adf_test(df['Passengers Second Difference'].dropna())
```

Time series

2 / Fixing non-constant Variance :

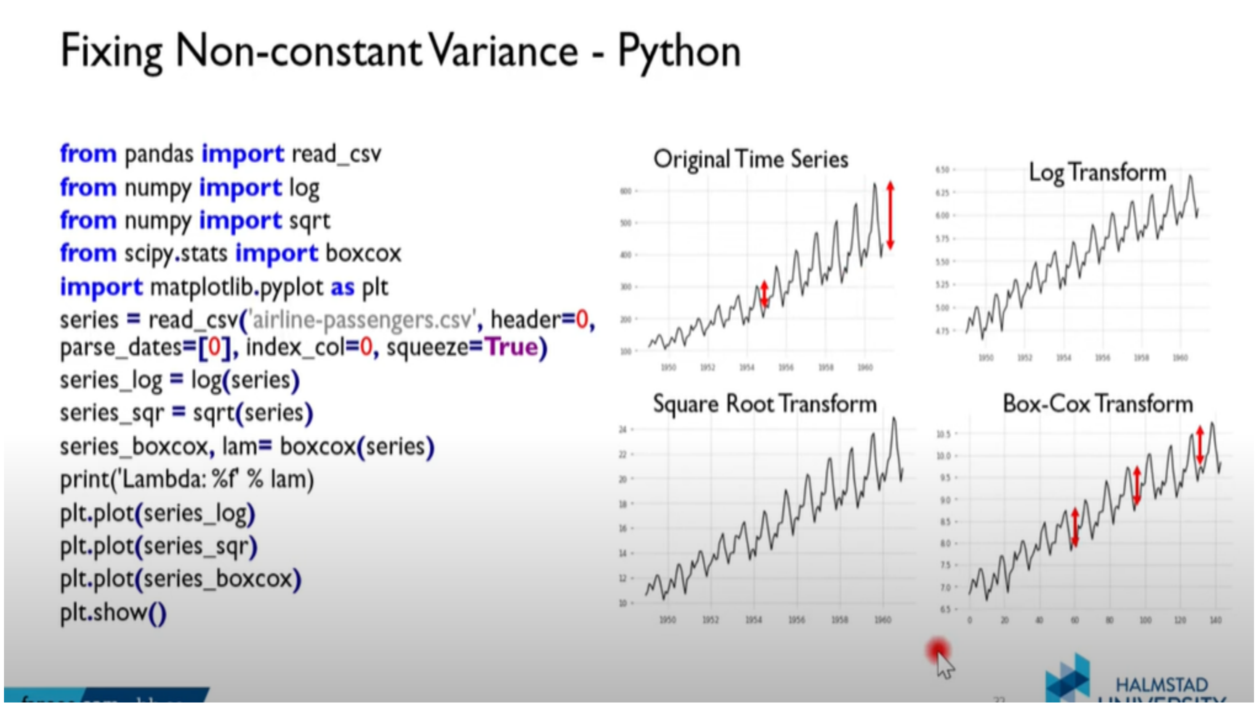


Fixing Non-constant Variance

- **Logarithmic**
  - Removes extreme (exponential) trends
- **Square root transformation**
  - Removes a quadratic growth trend
- **Box-Cox transform**
  - Supports both square root and log transform

Air Passengers from 1949 to 1961
Passenger numbers ('000)
Time
600
400
200
1950
1952
1954
1956
1958
1960

2-1/ Implementation

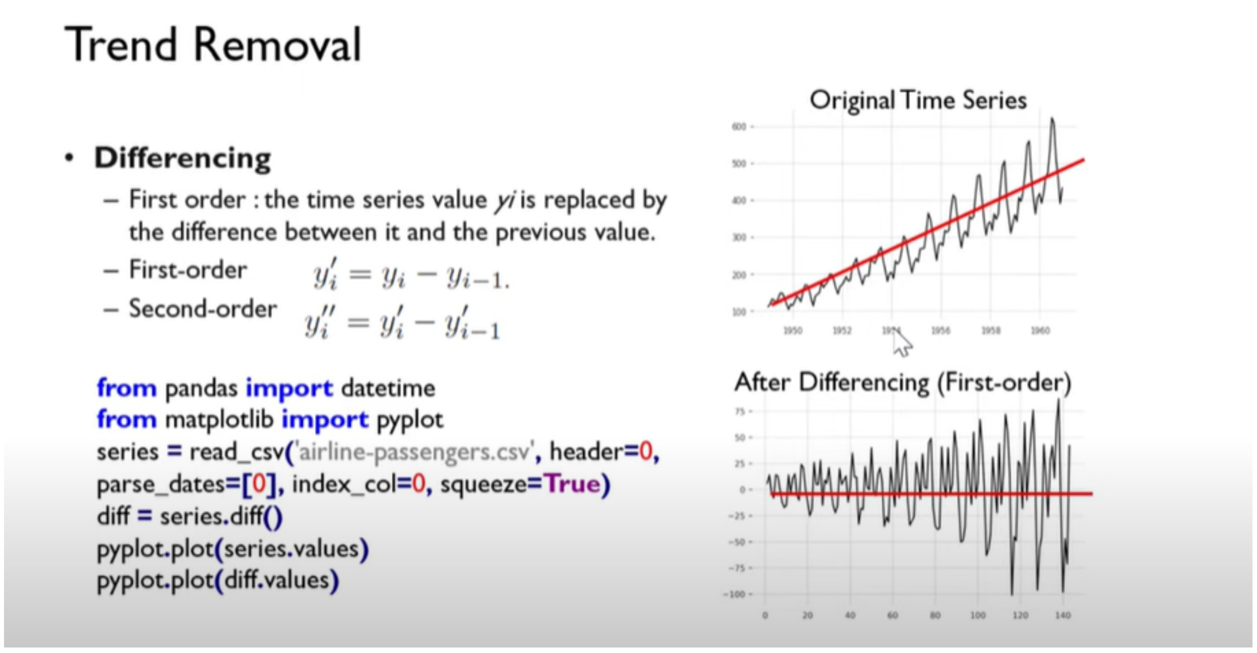


Fixing Non-constant Variance - Python

```
from pandas import read_csv
from numpy import log
from numpy import sqrt
from scipy.stats import boxcox
import matplotlib.pyplot as plt
series = read_csv('airline-passengers.csv', header=0,
parse_dates=[0], index_col=0, squeeze=True)
series_log = log(series)
series_sqr = sqrt(series)
series_boxcox, lam= boxcox(series)
print('Lambda: %f' % lam)
plt.plot(series_log)
plt.plot(series_sqr)
plt.plot(series_boxcox)
plt.show()
```

Time series

3 / Remove trends :



Trend Removal

- **Differencing**
  - First order : the time series value *yi* is replaced by the difference between it and the previous value.
  - First-order $y_i' = y_i - y_{i-1}.$
  - Second-order $y_i'' = y_i' - y_{i-1}'$

```
from pandas import datetime
from matplotlib import pyplot
series = read_csv('airline-passengers.csv', header=0,
parse_dates=[0], index_col=0, squeeze=True)
diff = series.diff()
pyplot.plot(series.values)
pyplot.plot(diff.values)
```

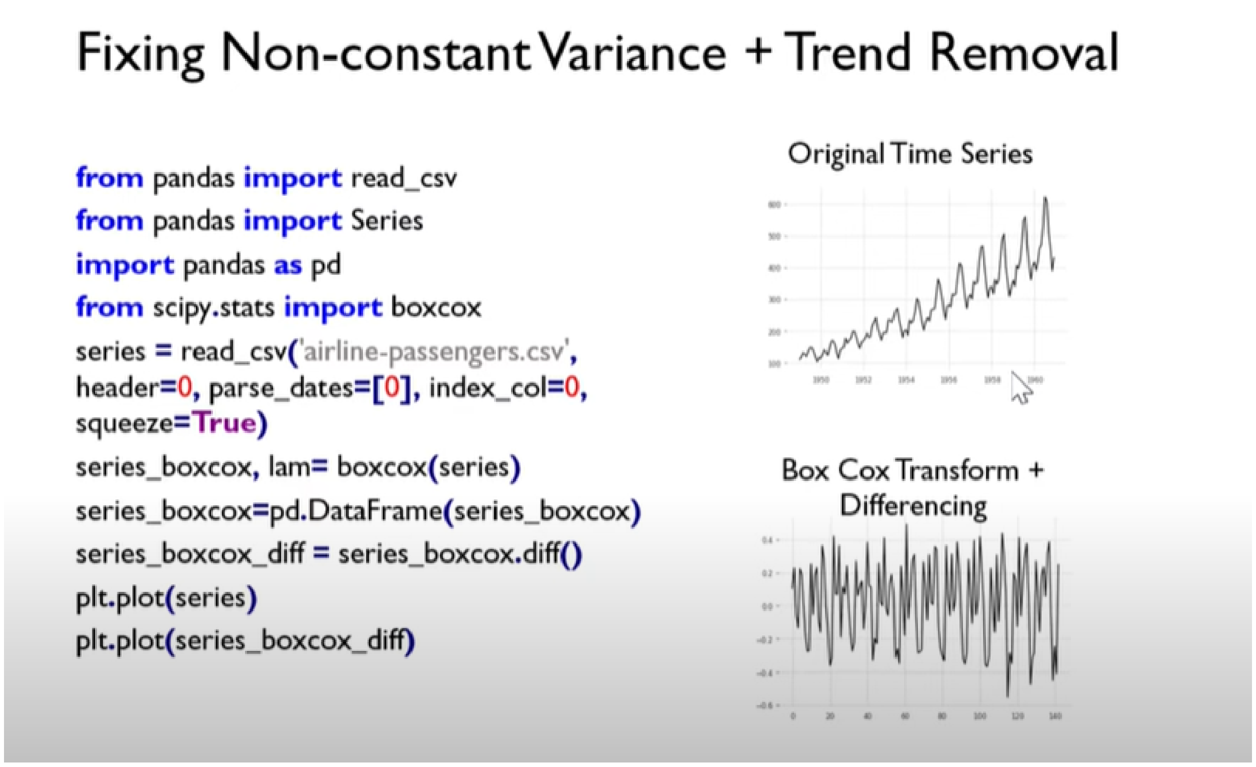
Implementation:

```
## Use Techniques Differencing
df['Passengers First Difference']=df['Thousands of
Passengers']-df['Thousands of Passengers'].shift(1)
```

This is our first shift so we can use multiple shift to remove trends so it called differencing as I in ARIMA model

Time series

We can also combine between fixing non-constant and trend removal



Fixing Non-constant Variance + Trend Removal

```
from pandas import read_csv
from pandas import Series
import pandas as pd
from scipy.stats import boxcox
series = read_csv('airline-passengers.csv',
header=0, parse_dates=[0], index_col=0,
squeeze=True)
series_boxcox, lam= boxcox(series)
series_boxcox=pd.DataFrame(series_boxcox)
series_boxcox_diff = series_boxcox.diff()
plt.plot(series)
plt.plot(series_boxcox_diff)
```

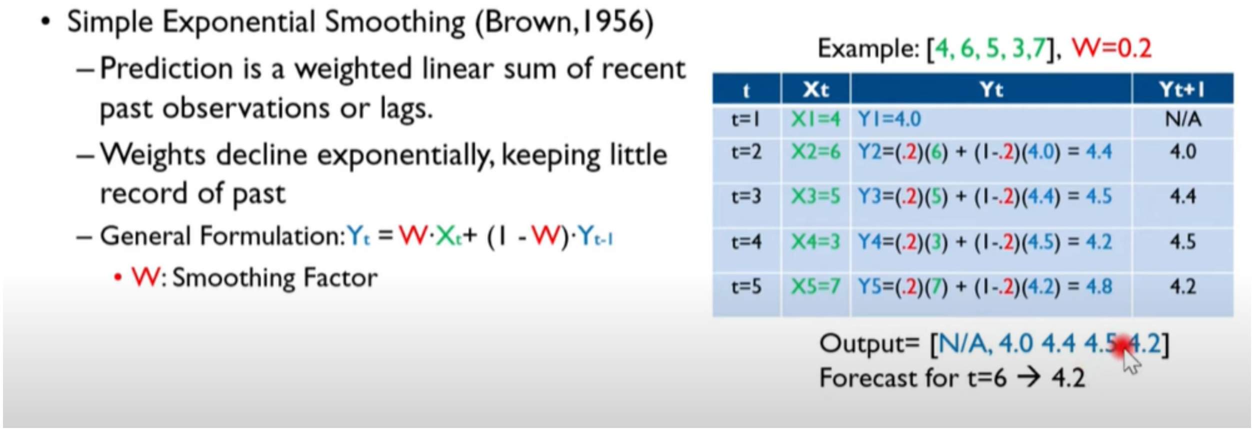
Time series

Time Series Forecasting

- Statistical Models

1 / Exponential smoothing :

Exponential smoothing is a broadly accurate forecasting method for **short-term forecasts**. The technique assigns larger weights to more recent observations while assigning exponentially decreasing weights as the observations get increasingly distant. This method produces slightly unreliable long-term forecasts.



- Simple Exponential Smoothing (Brown, 1956)
  - Prediction is a weighted linear sum of recent past observations or lags.
  - Weights decline exponentially, keeping little record of past
  - General Formulation: $Y_t = W \cdot X_t + (1 - W) \cdot Y_{t-1}$
    - W: Smoothing Factor

Example: [4, 6, 5, 3,7], W=0.2

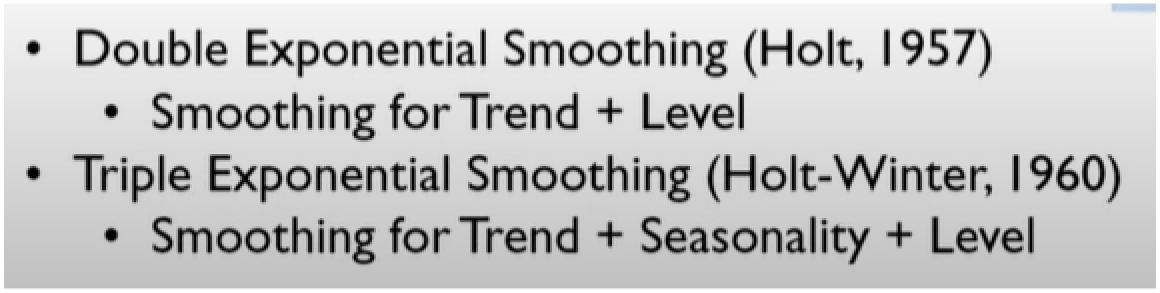
t	Xt	Yt	Yt+1
t=1	X1=4	Y1=4.0	N/A
t=2	X2=6	Y2=(.2)(6) + (1-.2)(4.0) = 4.4	4.0
t=3	X3=5	Y3=(.2)(5) + (1-.2)(4.4) = 4.5	4.4
t=4	X4=3	Y4=(.2)(3) + (1-.2)(4.5) = 4.2	4.5
t=5	X5=7	Y5=(.2)(7) + (1-.2)(4.2) = 4.8	4.2

Output= [N/A, 4.0 4.4 4.5 4.2]

Forecast for t=6 → 4.2

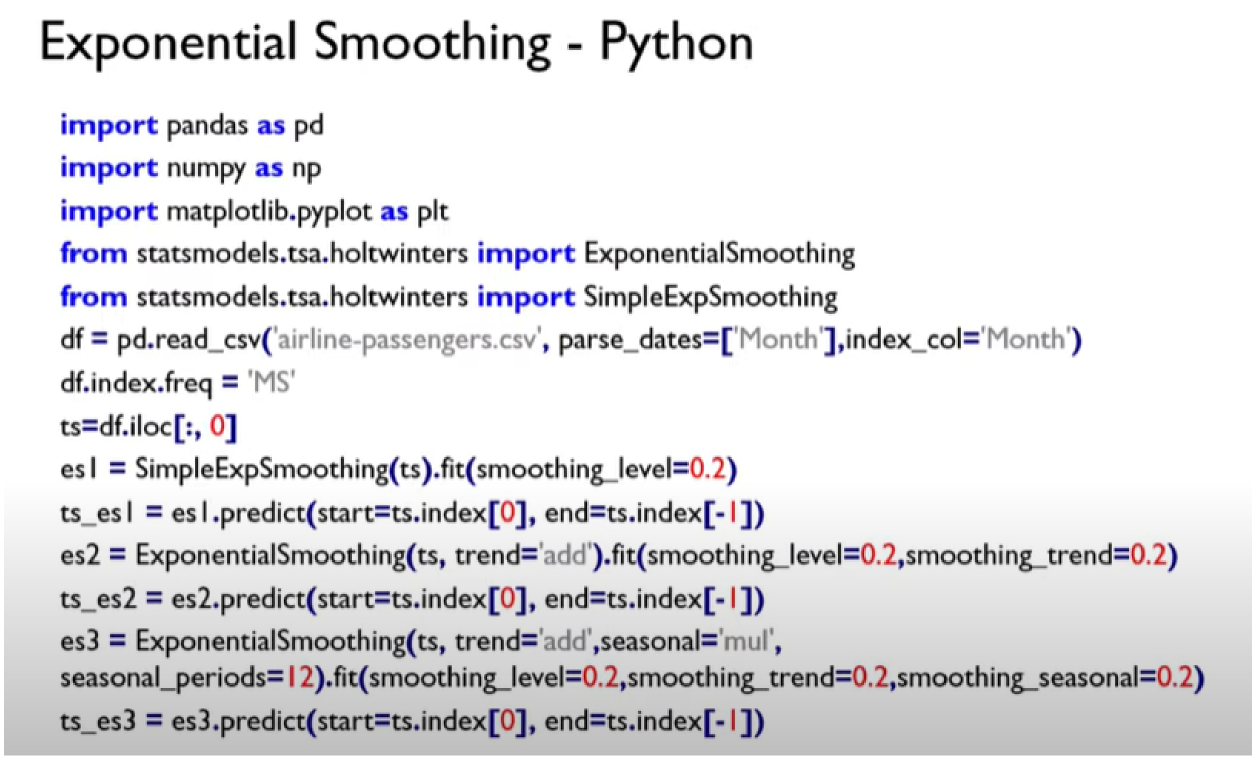
The latest lags have more weight to the old lags so we give more importance to the new observations. As mentioned in the general formulation.

We have also other type of smoothing such as :



- Double Exponential Smoothing (Holt, 1957)
  - Smoothing for Trend + Level
- Triple Exponential Smoothing (Holt-Winter, 1960)
  - Smoothing for Trend + Seasonality + Level

Time series

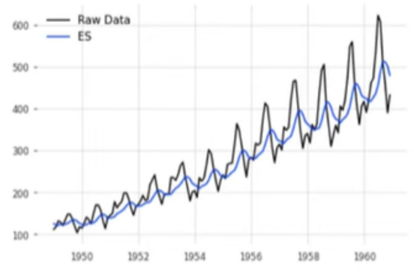


Exponential Smoothing - Python

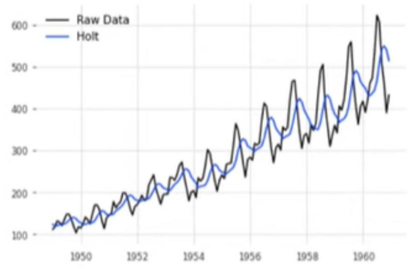
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from statsmodels.tsa.holtwinters import ExponentialSmoothing
from statsmodels.tsa.holtwinters import SimpleExpSmoothing
df = pd.read_csv('airline-passengers.csv', parse_dates=['Month'],index_col='Month')
df.index.freq = 'MS'
ts=df.iloc[:, 0]
es1 = SimpleExpSmoothing(ts).fit(smoothing_level=0.2)
ts_es1 = es1.predict(start=ts.index[0], end=ts.index[-1])
es2 = ExponentialSmoothing(ts, trend='add').fit(smoothing_level=0.2,smoothing_trend=0.2)
ts_es2 = es2.predict(start=ts.index[0], end=ts.index[-1])
es3 = ExponentialSmoothing(ts, trend='add',seasonal='mul',
seasonal_periods=12).fit(smoothing_level=0.2,smoothing_trend=0.2,smoothing_seasonal=0.2)
ts_es3 = es3.predict(start=ts.index[0], end=ts.index[-1])
```

Result :

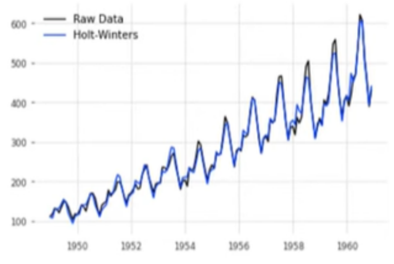
Exponential Smoothing - Python



Simple Exponential Smoothing
α=0.2
Level only



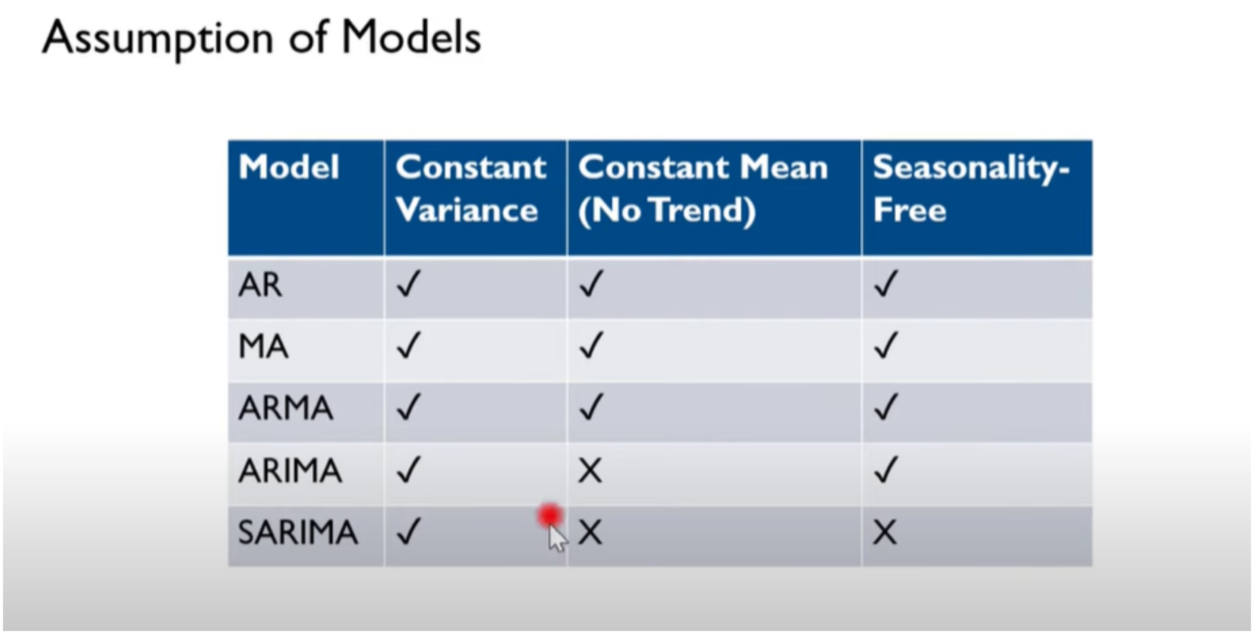
Double Exponential Smoothing
α =0.2, β=0.2
Level+ Trend



Triple Exponential Smoothing
α =0.2, β=0.2, γ=0.2, S=12
Level+ Trend + Seasonality

Here we see that triple exponential smoothing has better and relevant results than simple and double smoothing which is normal.

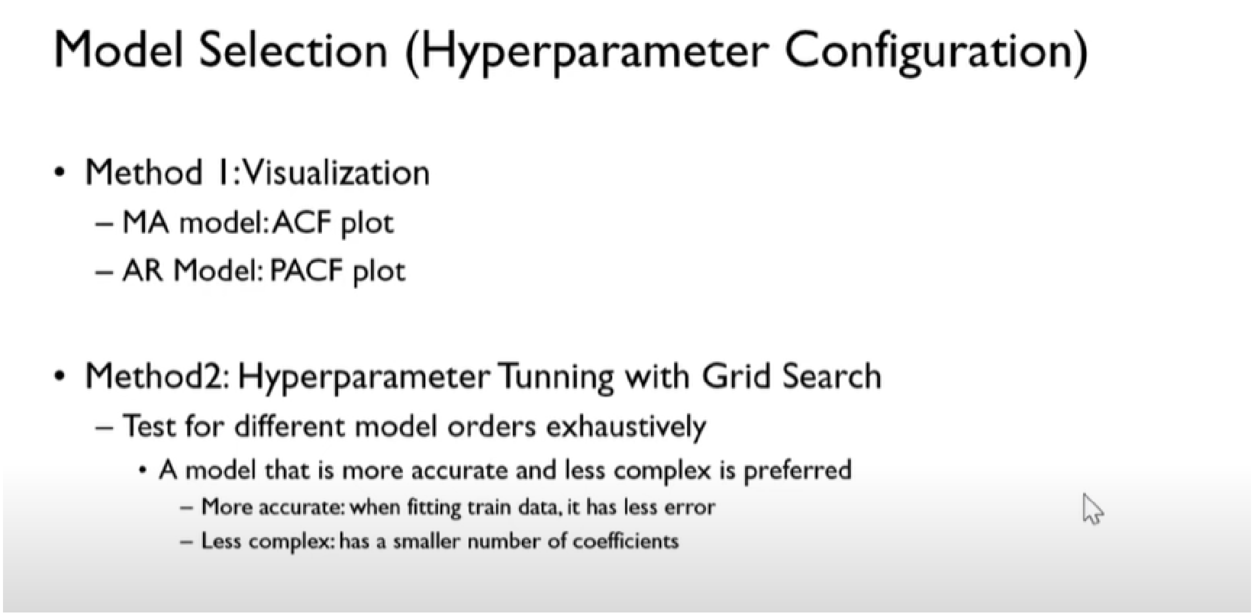
Time series



Assumption of Models

Model	Constant Variance	Constant Mean (No Trend)	Seasonality-Free
AR	✓	✓	✓
MA	✓	✓	✓
ARMA	✓	✓	✓
ARIMA	✓	X	✓
SARIMA	✓	X	X

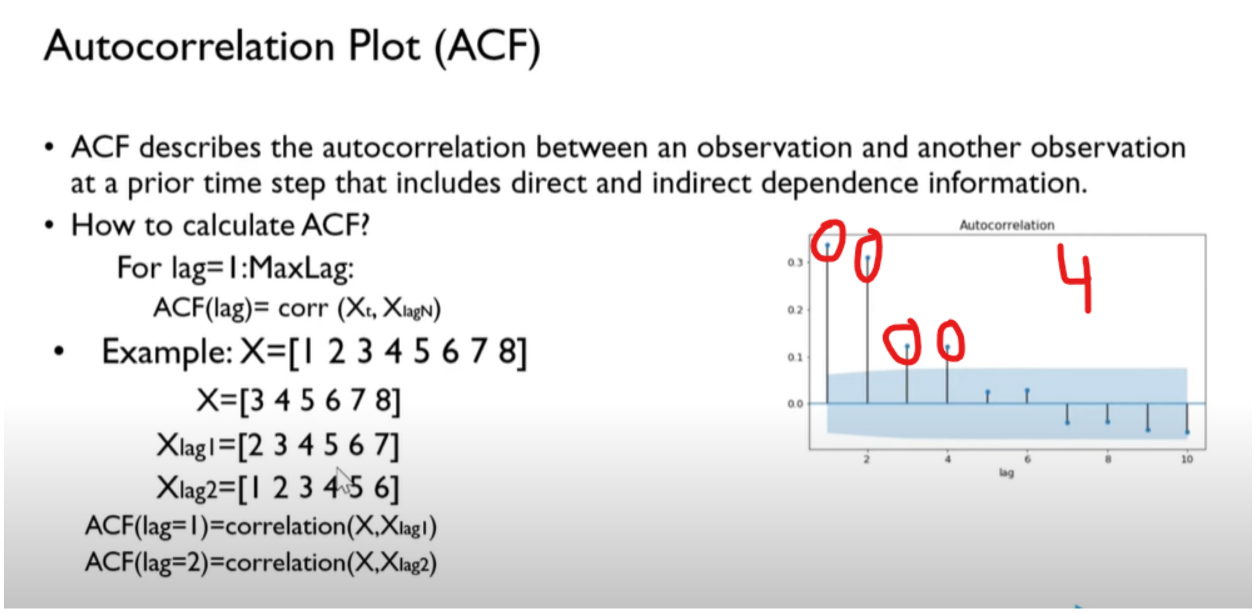
Model selection (Hyperparameter Configuration)



Model Selection (Hyperparameter Configuration)

- Method 1:Visualization
  - MA model: ACF plot
  - AR Model: PACF plot
- Method2: Hyperparameter Tunning with Grid Search
  - Test for different model orders exhaustively
    - A model that is more accurate and less complex is preferred
      - More accurate: when fitting train data, it has less error
      - Less complex: has a smaller number of coefficients

Time series



Autocorrelation Plot (ACF)

- ACF describes the autocorrelation between an observation and another observation at a prior time step that includes direct and indirect dependence information.
- How to calculate ACF?

  For lag=1:MaxLag:

  ACF(lag)= corr (X_t, X_{lagN})

- Example: X=[1 2 3 4 5 6 7 8]

  X=[3 4 5 6 7 8]

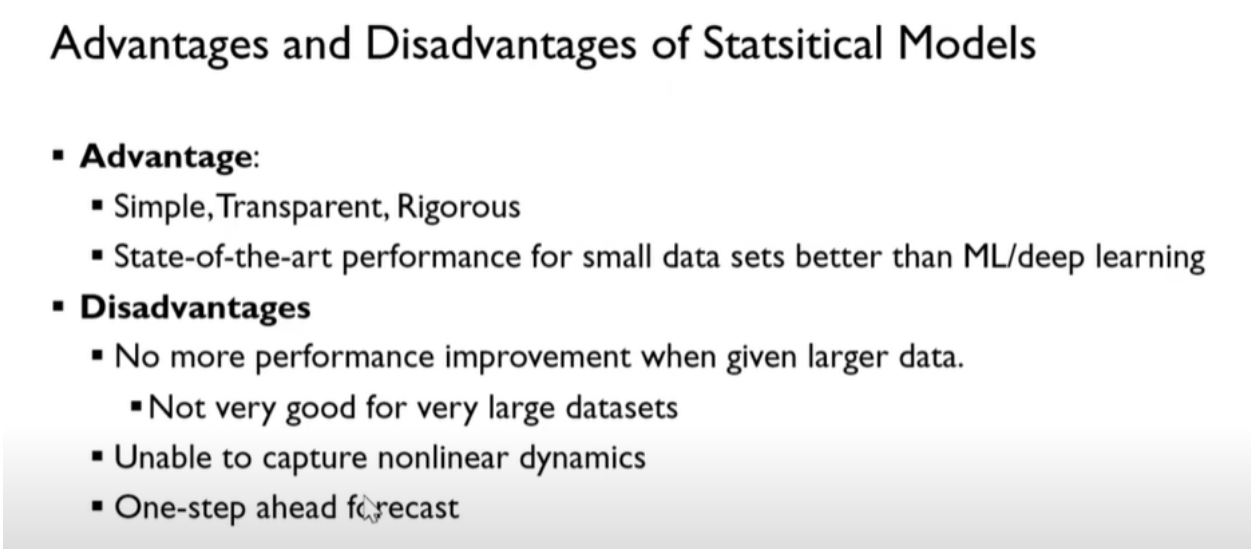
  X_{lag1}=[2 3 4 5 6 7]

  X_{lag2}=[1 2 3 4 5 6]

  ACF(lag=1)=correlation(X,X_{lag1})

  ACF(lag=2)=correlation(X,X_{lag2})

Only 4 of these lags have an impact on our data.



Advantages and Disadvantages of Statsitical Models

- **Advantage**:
  - Simple, Transparent, Rigorous
  - State-of-the-art performance for small data sets better than ML/deep learning
- **Disadvantages**
  - No more performance improvement when given larger data.
    - Not very good for very large datasets
  - Unable to capture nonlinear dynamics
  - One-step ahead forecast

Time series

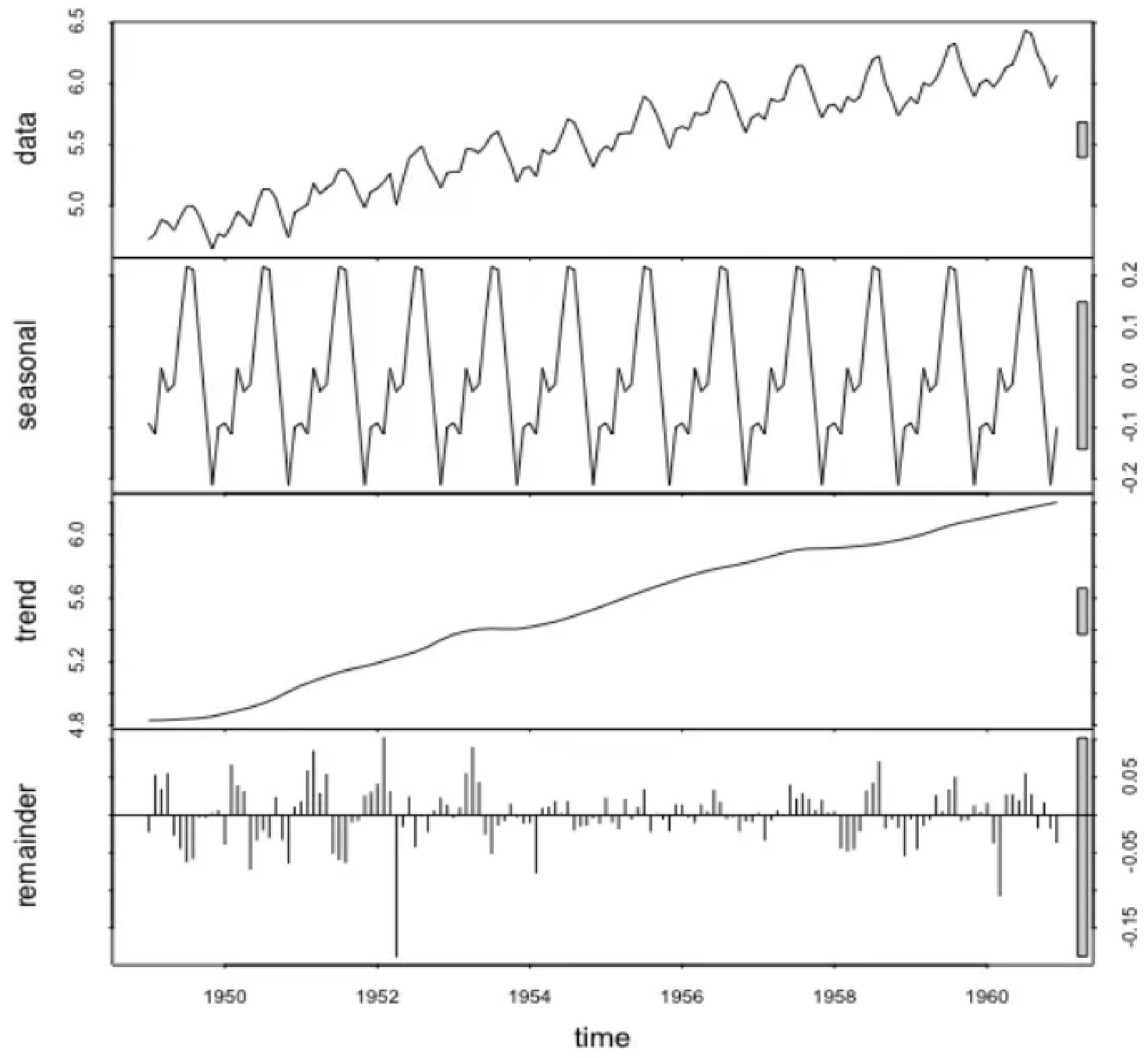
ANOMALY DETECTION :

1 / STL

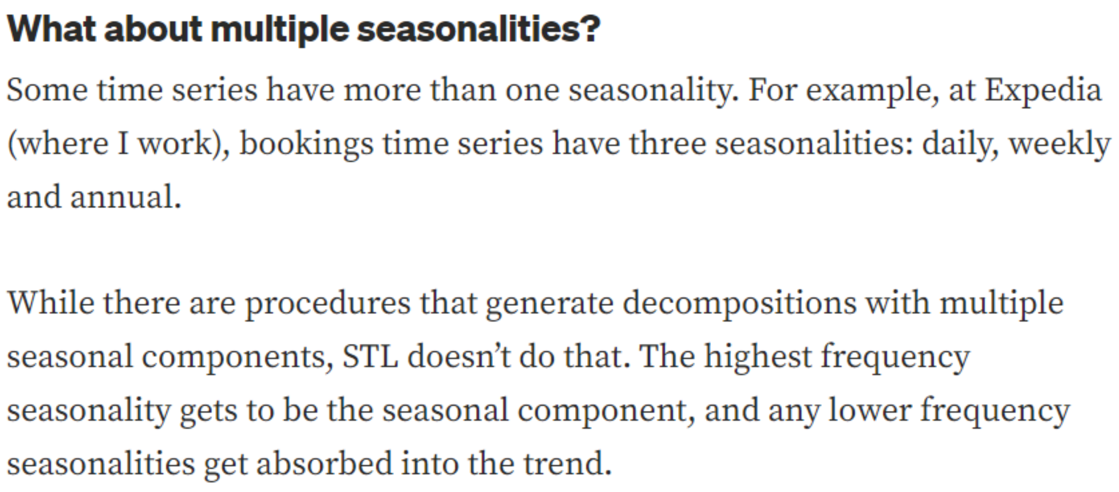
Seasonal-Trend Decomposition using LOESS (STL)

The basic idea is that if you have a time series with a regular pattern to it, you can run the series through the STL algorithm and isolate the regular pattern. Everything left over is by definition "irregular", and anomaly detection amounts to deciding whether the irregularity is sufficiently large.

Time series



The algorithm decomposes the series into three components: seasonal, trend and remainder. The *seasonal* is the periodic component, the *trend* is general up/down behavior, and the *remainder* is what's left over. The seasonal and trend taken together form the "regular" part of the series, and thus the part that we want to discount during anomaly detection.



What about multiple seasonalities?

Some time series have more than one seasonality. For example, at Expedia (where I work), bookings time series have three seasonalities: daily, weekly and annual.

While there are procedures that generate decompositions with multiple seasonal components, STL doesn't do that. The highest frequency seasonality gets to be the seasonal component, and any lower frequency seasonalities get absorbed into the trend.

2 / Isolation Forest