# Time Series Analysis with Statsmodels

PIERIAN DATA

# Python for Time Series

- Now that we know the necessary foundational tools to work with Time Series data, it is time to move on to learning about the main tool for time series forecasting: Statsmodels library!

# Python for Time Series

- Statsmodels is a Python module that provides classes and functions for the estimation of many different statistical models, as well as for conducting statistical tests, and statistical data exploration.

# Python for Time Series

- Section Goals
    - Introduction to Statsmodels
    - ETS Decomposition
    - Moving Averages
    - Holt Winters Methods
    - Statsmodels Time Series Exercises

# Introduction to Statsmodels

# Python for Time Series

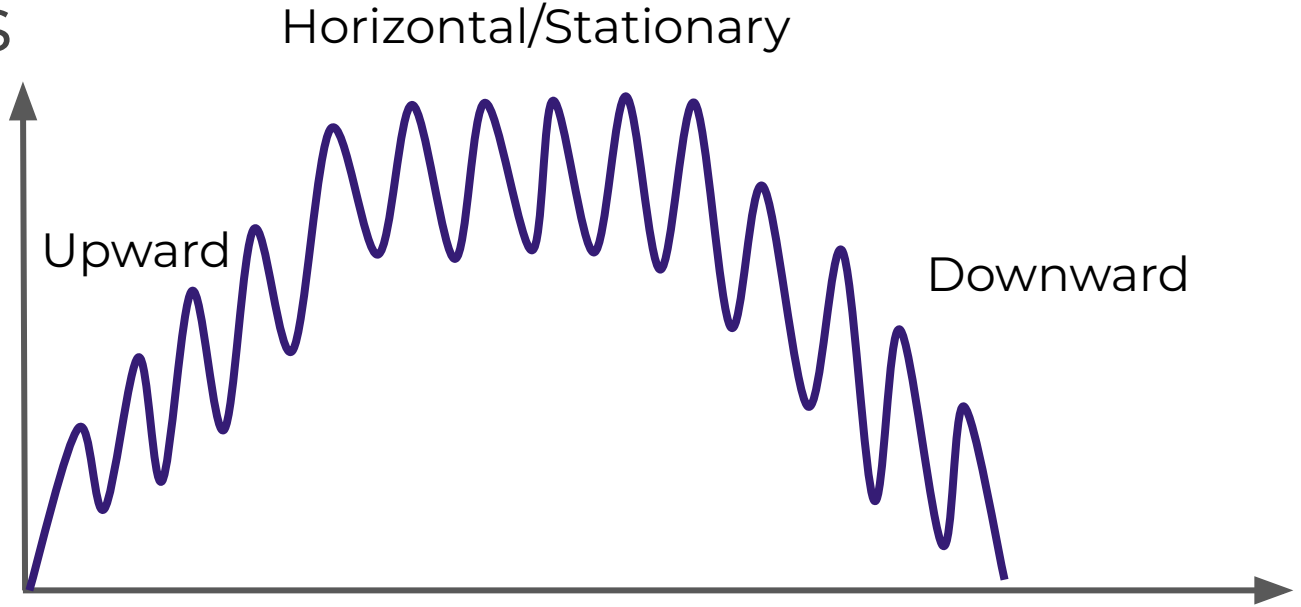
- Let’s learn how to call a function test from Statsmodels.
- Statsmodels has lots of useful statistical tests built in.
- We will learn about the Hodrick-Prescott filter.

# Python for Time Series

- Let’s begin discussing some important Time series concepts.
- Time series data has particular properties, let’s take a look at some plots and discuss some important terms!

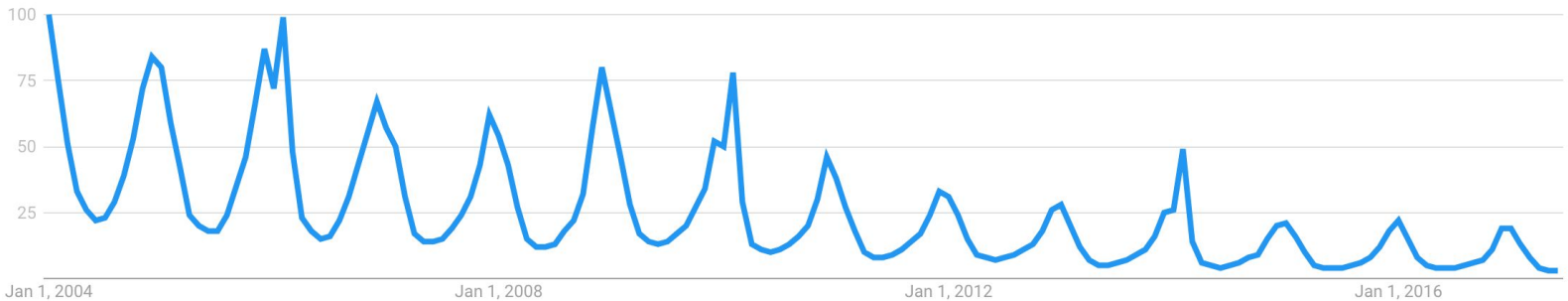
# Python for Time Series

- Trends



# Python for Time Series

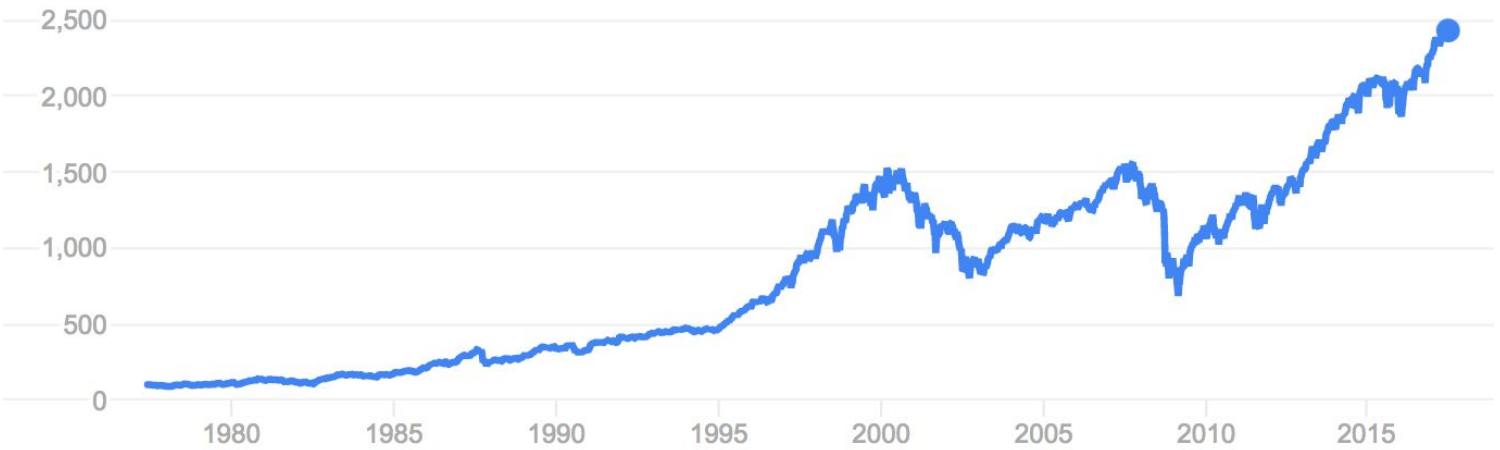
- Seasonality - Repeating trends




Google Trends - “Snowboarding”

# Python for Time Series

- Cyclical - Trends with no set repetition.




SP500

# Python for Time Series

- The Hodrick-Prescott filter separates a time-series  y_t  into a trend component τ_t  and a cyclical component  c_t

$$y_t = \tau_t + c_t$$

# Python for Time Series

- The components are determined by minimizing the following quadratic loss function, where $\lambda$ is a smoothing parameter:

$$\min_{\tau_t} \sum_{t=1}^{T} c_t^2 + \lambda \sum_{t=1}^{T} [(\tau_t - \tau_{t-1}) - (\tau_{t-1} - \tau_{t-2})]^2$$

# Python for Time Series

- The $\lambda$ value above handles variations in the growth rate of the trend component.

$$\min_{\tau_t} \sum_{t=1}^{T} c_t^2 + \lambda \sum_{t=1}^{T} [(\tau_t - \tau_{t-1}) - (\tau_{t-1} - \tau_{t-2})]^2$$

# Python for Time Series

- When analyzing quarterly data, the default lambda value of 1600 is recommended. Use 6.25 for annual data, and 129,600 for monthly data.

$$\min_{\tau_t} \sum_{t=1}^{T} c_t^2 + \lambda \sum_{t=1}^{T} [(\tau_t - \tau_{t-1}) - (\tau_{t-1} - \tau_{t-2})]^2$$

# Python for Time Series

- Let’s explore how to use this basic functionality with Statsmodels.
- Also make sure you have activated our provided environment before beginning!

# Time Series Basics

# Python for Time Series

- Now that we understand some of the very basics, let’s begin to learn about the most popular library in Python for handling time series data, statsmodels!

# Statsmodels

# Python for Time Series

- The most popular library in Python for dealing with Time Series data is the statsmodels library.
- It is heavily inspired by the R statistical programming language.

# Python for Time Series

- Statsmodels is a Python module that allows users to explore data, estimate statistical models, and perform statistical tests.

# Python for Time Series

- An extensive list of descriptive statistics, statistical tests, plotting functions, and result statistics are available for different types of data and each estimator.

# Python for Time Series

- Statsmodels is already included in the provided environment file.
- To manually install you can use:
    - conda install statsmodels

# Python for Time Series

- Let’s explore the documentation and then run through a simple demonstration of what we can use statsmodels for in relation to time series data.

# ETS Models

# Python for Time Series

- ETS Models (Error-Trend-Seasonality)
    - Exponential Smoothing
    - Trend Methods Models
    - ETS Decomposition
- We'll work with several of these with the python statsmodels library!

# Python for Time Series

- Statsmodels provides a seasonal decomposition tool we can use to separate out the different components.
- We've already seen a simplistic example of this in the Introduction to Statsmodels section with the Hodrick-Prescott filter.

# Python for Time Series

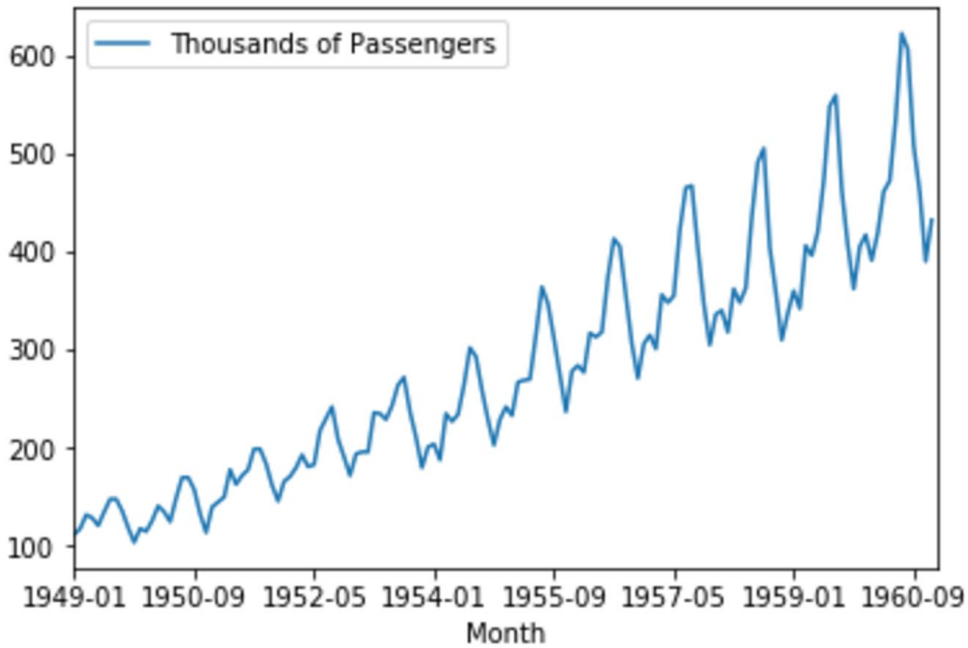
- ETS (Error-Trend-Seasonality) Models will take each of those terms for “smoothing” and may add them, multiply them, or even just leave some of them out.
- Based off these key factors, we can try to create a model to fit our data.

# Python for Time Series

- Time Series Decomposition with ETS (Error-Trend-Seasonality).
- Visualizing the data based off its ETS is a good way to build an understanding of its behaviour.

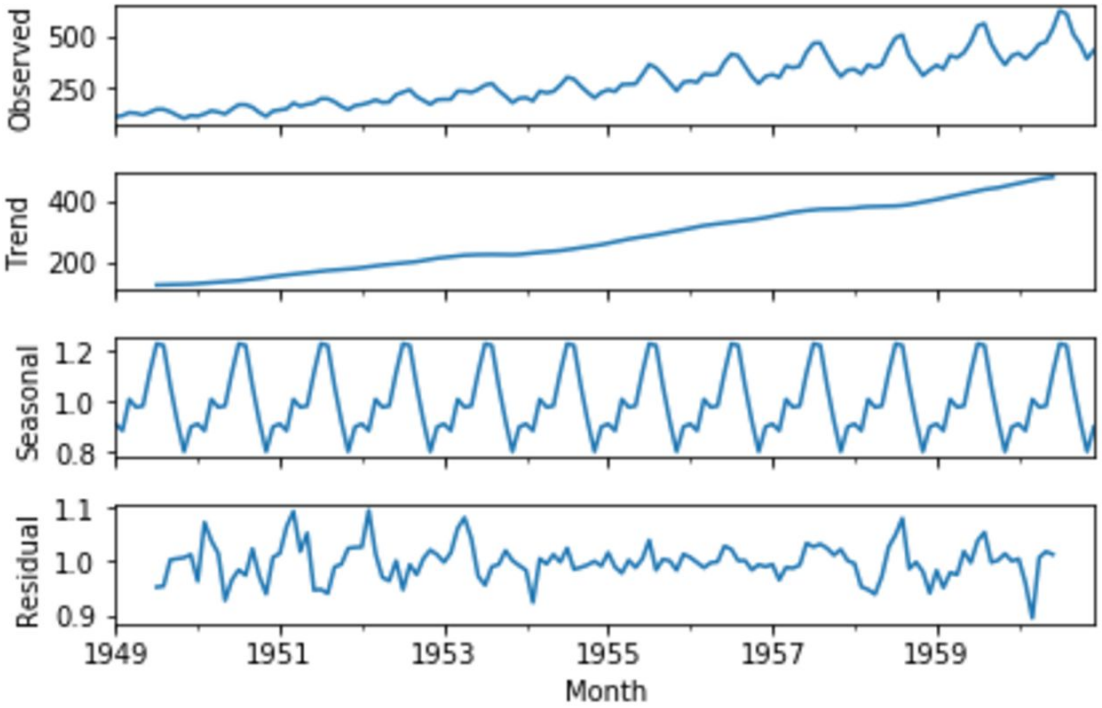
# Python for Time Series

- ETS Decomposition - Airline Passengers



# Python for Time Series

- ETS Decomposition - Airline Passengers




PIERIAN DATA

# Python for Time Series

- We apply an additive model when it seems that the trend is more linear and the seasonality and trend components seem to be constant over time (e.g. every year we add 10,000 passengers).

# Python for Time Series

- A multiplicative model is more appropriate when we are increasing (or decreasing) at a non-linear rate (e.g. each year we double the amount of passengers).

# Python for Time Series

- Let’s explore how to perform an ETS decomposition with Statsmodels!

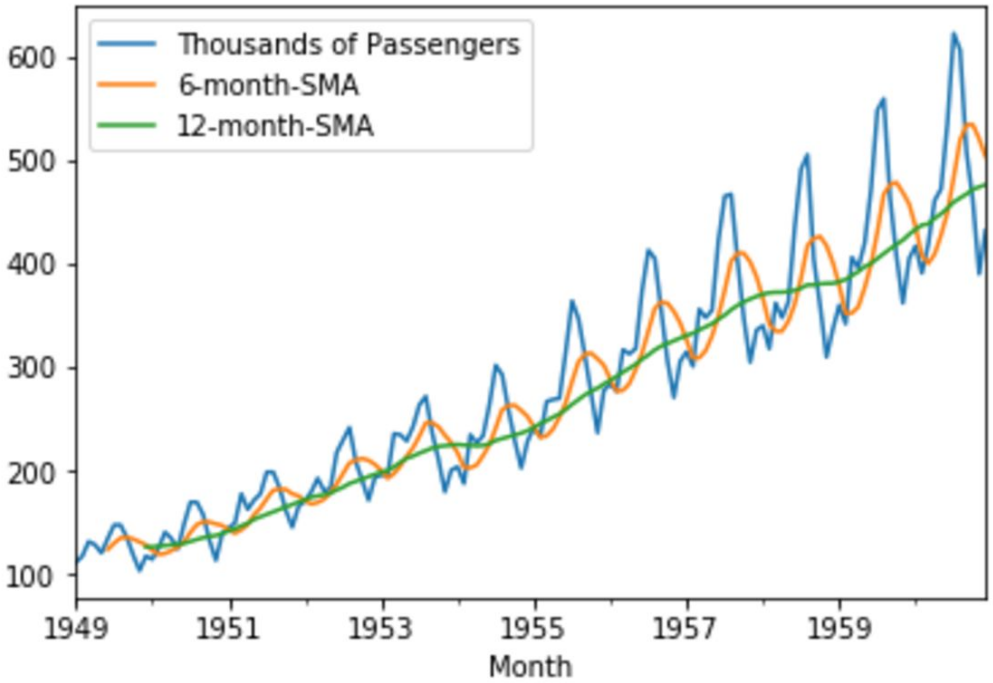
# EWMA Models

# Python for Time Series

- We’ve previously seen how calculating simple moving averages can allow us to create a simple model that describes some trend level behavior of a time series, for example...

# Python for Time Series

- SMA - Simple Moving Averages



# Python for Time Series

- We could theoretically attempt to use these simple moving averages to build a generalized model for the real world time series we're analyzing.
- Later on, we'll discover much more sophisticated models for this.

# Python for Time Series

- We could expand off the idea of an SMA (simple moving average) by utilizing a EWMA (Exponentially weighted moving average).
- As issue with SMA is that the entire model will be constrained to the same window size.

# Python for Time Series

- It would be nice if we could have more recent data be **weighted** more than older data.
- We do this by implementing a EWMA instead of SMA.

# Python for Time Series

- EWMA- Exponentially Weighted Moving Averages
- Basic SMA has some "weaknesses".
    - Smaller windows will lead to more noise, rather than signal

# Python for Time Series

- EWMA- Exponentially Weighted Moving Averages
- Basic SMA has some "weaknesses".
    - It will always lag by the size of the window

# Python for Time Series

- EWMA- Exponentially Weighted Moving Averages
- Basic SMA has some "weaknesses".
    - It will never reach to full peak or valley of the data due to the averaging.

# Python for Time Series

- EWMA- Exponentially Weighted Moving Averages
- Basic SMA has some "weaknesses".
    - Does not really inform you about possible future behaviour, all it really does is describe trends in your data.

# Python for Time Series

- EWMA- Exponentially Weighted Moving Averages
- Basic SMA has some "weaknesses".
    - Extreme historical values can skew your SMA significantly

# Python for Time Series

- EWMA- Exponentially Weighted Moving Averages
- Basic SMA has some "weaknesses".
    - To help fix some of these issues, we can use an EWMA (Exponentially-weighted moving average).

# Python for Time Series

- EWMA will allow us to reduce the lag effect from SMA and it will put more weight on values that occurred more recently (by applying more weight to the more recent values, thus the name).

# Python for Time Series

- The amount of weight applied to the most recent values will depend on the actual parameters used in the EWMA and the number of periods given a window size.

# Python for Time Series

- Let’s code out an example of using pandas to create EWMA in the next lecture!

# Holt Winters Method

# Python for Time Series

- Previously with  Exponentially Weighted Moving Averages (EWMA) we applied Simple Exponential Smoothing using just one smoothing factor **α** (alpha).
- This failed to account for other contributing factors like trend and seasonality.

# Python for Time Series

- Note: We won't use Holt-Winters for forecasting just yet, that will come in the next section of the course!
- For now, we'll focus on fitting the Holt-Winters model to an underlying time series data set.

# Python for Time Series

- Holt (1957) and Winters (1960) extended Holt's method to capture seasonality.
- The Holt-Winters seasonal method comprises of the forecast equation and three smoothing equations.

# Python for Time Series

- One for the level $\ell$_t , one for the trend b_t , and one for the seasonal component s_t , with corresponding smoothing parameters α , β and γ .

# Python for Time Series

- There are two variations to this method that differ in the nature of the seasonal component.

# Python for Time Series

- The additive method is preferred when the seasonal variations are roughly constant through the series, while the multiplicative method is preferred when the seasonal variations are changing proportional to the level of the series.

# Python for Time Series

- Let’s explore the Holt-Winters methods , which allow us to add on double and triple exponential smoothing.

# Python for Time Series

- Single Exponential Smoothing

$$y_0 = x_0$$
$$y_t = (1 - \alpha) y_{t-1} + \alpha x_t$$

# Python for Time Series

- In Double Exponential Smoothing (aka Holt's Method) we introduce a new smoothing factor **β** (beta) that addresses trend:

# Python for Time Series

- Double Exponential Smoothing

$$
\begin{aligned}
l_t &= (1-\alpha)l_{t-1} + \alpha x_t, \\
b_t &= (1-\beta)b_{t-1} + \beta(l_t - l_{t-1}) \\
y_t &= l_t + b_t \\
\hat{y}_{t+h} &= l_t + hb_t
\end{aligned}
$$

# Python for Time Series

- In Double Exponential Smoothing (aka Holt's Method) we introduce a new smoothing factor **β** (beta) that addresses trend:

$$l_t = (1-\alpha)l_{t-1} + \alpha x_t, \qquad \text{level}$$

$$b_t = (1-\beta)b_{t-1} + \beta(l_t - l_{t-1}) \qquad \text{trend}$$

$$y_t = l_t + b_t \qquad \text{fitted model}$$

$$\hat{y}_{t+h} = l_t + hb_t \qquad \text{forecasting model } (h = \#\text{ periods into the future})$$

# Python for Time Series

- Because we haven't yet considered seasonal fluctuations, the forecasting model is simply a straight sloped line extending from the most recent data point.

# Python for Time Series

- With Triple Exponential Smoothing (aka the Holt-Winters Method) we introduce a smoothing factor (gamma) that addresses seasonality:

# Python for Time Series

- With Triple Exponential Smoothing (aka the Holt-Winters Method) we introduce a smoothing factor (gamma) that addresses seasonality:

$$
\begin{aligned}
l_t &= (1-\alpha)l_{t-1} + \alpha x_t, \\
b_t &= (1-\beta)b_{t-1} + \beta(l_t - l_{t-1}) \\
c_t &= (1-\gamma)c_{t-L} + \gamma(x_t - l_{t-1} - b_{t-1}) \\
y_t &= (l_t + b_t)c_t \\
\hat{y}_{t+m} &= (l_t + mb_t)c_{t-L+1+(m-1) \bmod L}
\end{aligned}
$$

# Python for Time Series

- Here L represents the number of divisions per cycle. In our case looking at monthly data that displays a repeating pattern each year, we would use L=12

$$l_t = (1-\alpha)l_{t-1} + \alpha x_t, \quad \text{level}$$

$$b_t = (1-\beta)b_{t-1} + \beta(l_t - l_{t-1}) \quad \text{trend}$$

$$c_t = (1-\gamma)c_{t-L} + \gamma(x_t - l_{t-1} - b_{t-1}) \quad \text{seasonal}$$

$$y_t = (l_t + b_t)c_t \quad \text{fitted model}$$

$$\hat{y}_{t+m} = (l_t + mb_t)c_{t-L+1+(m-1) \bmod L} \quad \text{forecasting model } (m = \#\text{ periods into the future})$$

# Holt Winters Methods Code Along

PIERIAN DATA

# Statsmodels Time Series Exercises

# Statsmodels Time Series Exercises

## SOLUTIONS

PIERIAN DATA