14/02/2025

**Python for Data Professionals in Finance**

This comprehensive learning path is designed for data professionals in the finance industry who want to leverage the power of Python. Learn best practices for performing calculations; ingesting and preparing financial data; integrating Python with Excel for data manipulation, analysis, and reporting; creating dynamic financial models; and using advanced machine learning models to enhance your trading strategies.

* **Explore**, analyze, and visualize financial data with Python.
* **Unlock** the power of Python functionality within Excel.
* **Build** dynamic financial models for trading strategies.

Course 1 : **Getting Started with Python for Finance**

**What you should know**

* **Basic Python Knowledge**: You should have a basic understanding of Python, including concepts like variables, loops, and functions.
* **Experience with Jupyter Notebooks**: Familiarity with Jupyter Notebooks will be helpful as it allows you to create and share code, visualizations, and other content in a single document.
* **Excel Formulas**: Knowledge of basic Excel formulas (like sum, average, and if) will be beneficial, even though Excel is not directly used in this course.

***Loading Data & Visualizations***

**Loading data**

* **Importing Libraries**: The video demonstrates how to import essential libraries like matplotlib, NumPy, Pandas, and Y Finance, and the use of aliasing for convenience.
* **Loading Data**: The instructor shows how to use the Y Finance library to download historical data for the S&P 500 and Apple from 2010 to 2019, resulting in a Pandas DataFrame.
* **Inspecting Data**: The video covers inspecting the DataFrame, understanding its structure, and using the pipe method to chain operations for better readability and maintainability of the code.

These steps are crucial for preparing financial data for analysis using Python.

**raw.pipe? – pulls the documentation for pipe in Jupyter notebook. Whenever we put a ? after a method or function Jupyter will pull a documentation for it.**

15/02/2025

**Line plots**

* **Creating Line Plots**: The video demonstrates how to create line plots using the Pandas library, which is built on top of Matplotlib.
* **Plotting Data**: It explains how to plot data with the date index on the X-axis and each column as its own line.
* **Customizing Plots**: The video shows how to customize plots, such as changing the figure size and plotting specific columns like the Close or Volume columns.

**Chaining operations in Pandas is important for several reasons:**

* **Readability**: Chaining makes the code easier to read and understand, as it flows like a recipe. Each step is clear and sequential.
* **Efficiency**: It allows for more concise code, reducing the need for intermediate variables and making the code more efficient.
* **Maintainability**: Chained operations are easier to maintain and debug, as each step of the data manipulation process is clearly defined.

By chaining, you can perform complex data transformations in a streamlined and organized manner, which is particularly useful when working with financial data in your field.

Bottom of Form

**Resampling data**

* **Resampling Data**: The video explains how to change the granularity of data using the Pandas library, such as converting daily data to monthly or quarterly data.
* **Using Pandas**: It demonstrates how to use the resample method in Pandas with different offset aliases (e.g., 'M' for month, 'Q' for quarter) to aggregate data.
* **Aggregation Methods**: The video shows how to apply aggregation methods like mean to compute average values for the resampled data.
* **Plotting Resampled Data**: Once resampled, the data can be easily plotted to visualize trends over different time periods.

The steps taught in the video:

1. **Load Data**: Load your data into a Pandas DataFrame, ensuring it includes a date column.
2. **Select Columns**: Use the iloc method to select the columns you need.
3. **Fix Columns**: Apply the fixed\_columns method to collapse hierarchical columns.
4. **Resample Data**: Use the resample method with an offset alias (e.g., 'M' for month) to change the data granularity.
5. **Aggregate Data**: Apply an aggregation method, such as mean, to compute the average values for the resampled data.
6. **Plot Data**: Plot the resampled data using the plot method to visualize the results.

**Offset aliases in Pandas are strings that specify the frequency for resampling data.**

* **'M'**: Represents month. Aggregates data at the monthly level.
* **'2M'**: Represents two months. Aggregates data at the two monthly level.
* **'Q'**: Represents quarter. Aggregates data at the quarterly level.

For example, using 'M' will resample the data to monthly values, while 'Q' will resample it to quarterly values. These aliases help you easily change the granularity of your data.

Bottom of Form

**Granularity refers to the level of detail in the data.** Specifically, it describes the frequency at which data points are recorded. For example:

* **Daily Granularity**: Data points are recorded every day.
* **Monthly Granularity**: Data points are aggregated and recorded every month.
* **Quarterly Granularity**: Data points are aggregated and recorded every quarter.

The video demonstrates how to change the granularity of data using the Pandas library, allowing you to resample daily data to monthly or quarterly data for different levels of analysis.

16/02/2025

**Candlestick plot from scratch:**

* The video explains how to create a candlestick plot using Pandas and Matplotlib.
* It demonstrates how to resample data at different intervals (daily, monthly, etc.) and aggregate values using the ag method.
* The instructor shows how to use the pipe method to pass a custom plotting function for creating the candlestick plot.
* The plot displays vertical lines (wicks) for high and low values and colored bars (candles) for open and close values, with red indicating a decrease and green indicating an increase.

**What is a candlestick plot?**

A candlestick plot is a type of financial chart used to represent the price movements of an asset over a specific period. Each "candlestick" shows four key pieces of information: the opening price, closing price, highest price, and lowest price for that period.

* The "body" of the candlestick represents the range between the opening and closing prices.
* If the closing price is higher than the opening price, the body is typically colored green (indicating a price increase).
* If the closing price is lower than the opening price, the body is colored red (indicating a price decrease).
* The "wicks" or "shadows" are the lines extending above and below the body, representing the highest and lowest prices during the period.

This type of plot helps visualize the price trends and volatility of financial instruments.

**What is use of pipe method?**

The pipe method in Pandas is used to apply a function to a DataFrame or Series in a clean and readable way. In the context of creating a candlestick plot, the pipe method is used to pass the DataFrame to a custom plotting function. This helps streamline the code and makes it easier to follow.  
  
In the video, the pipe method is used to:

* Pass the DataFrame to the plot\_candle function.
* The plot\_candle function then uses Matplotlib to create the candlestick plot.

This method allows for chaining operations and maintaining a clear and concise workflow.

**Explain the candle plot function in the video**

* **Purpose**: The plot\_candle function is used to create a candlestick plot using Matplotlib, as Pandas alone cannot create this type of plot.
* **Function Input**: It takes a DataFrame and a Matplotlib axes object (ax) as inputs.
* **Steps Inside the Function**:
  1. **Vertical Lines (Wicks)**:
     + Plots vertical lines for each date, representing the high and low prices.
     + The Y-minimum value is set to the low price, and the Y-maximum value is set to the high price.
     + These lines are drawn in black.
  2. **Red Candles (Price Decrease)**:
     + Filters the DataFrame to find periods where the opening price is greater than the closing price.
     + Plots vertical lines for these periods, with the Y-minimum value set to the closing price and the Y-maximum value set to the opening price.
     + These lines are drawn in red and are slightly wider to represent the body of the candlestick.
  3. **Green Candles (Price Increase)**:
     + Filters the DataFrame to find periods where the opening price is less than the closing price.
     + Plots vertical lines for these periods, with the Y-minimum value set to the opening price and the Y-maximum value set to the closing price.
     + These lines are drawn in green and are slightly wider to represent the body of the candlestick.
  4. **Axes Formatting**:
     + Sets the major locator to mark the months on the x-axis.
     + Formats the dates to display as "Month-Year".
     + Adds a minor locator for smaller ticks.
* **Output**: The function returns the DataFrame, and the plot is displayed in Jupyter Notebook.

This function effectively creates a visual representation of the price movements over time, making it easier to analyze trends and patterns in the data.

**Difference between iloc and loc in Pandas:**

* iloc:
  + Stands for "integer-location".
  + Selects data based on the numerical position (index) of the rows and columns.
  + Example: df.iloc[0, 1] selects the first row and second column.
* loc:
  + Stands for "label-location".
  + Selects data based on the labels or names of the rows and columns.
  + Example: df.loc['2021-01-01', 'Close'] selects the row with the date '2021-01-01' and the column 'Close'.

In the video, iloc is used to select columns by their position, while loc is used to select rows based on date ranges.

Bottom of Form

***Calculations***

**Returns**

**How do you calculate returns using Pandas?**

To calculate returns using Pandas, you can use the pct\_change method. Here's a step-by-step process:

1. **Load your data** into a DataFrame. For example, let's assume you have a DataFrame called AAPL with stock data.
2. **Calculate the returns** using the pct\_change method on the 'Close' column:  
   python  
   returns = AAPL['Close'].pct\_change()

The pct\_change method calculates the percentage change between the current and previous values, which represents the returns. For example, if the closing price on January 5th is higher than on January 4th, the return will be positive.  
  
You can also check the documentation for pct\_change within Jupyter by typing:  
AAPL.pct\_change?  
  
This will provide detailed information and examples on how to use the method.

**What are returns?**

In the context of finance, "returns" refer to the profit or loss generated by an investment over a specific period. It is usually expressed as a percentage.  
  
In the video, returns are calculated using the pct\_change method in Pandas, which computes the percentage change between the current and previous values in a series. For example, if the closing price of a stock increases from one day to the next, the return for that day will be positive, indicating a profit. Conversely, if the closing price decreases, the return will be negative, indicating a loss.

**What is percentage change?**

Percentage change is a way to measure the relative change between two values over time. In the context of the video on "Returns," it is used to calculate the returns of a stock.  
  
Here's how it works:

* **Formula**: (Current Value - Previous Value) / Previous Value \* 100
* **Example**: If the closing price of a stock was $100 yesterday and $105 today, the percentage change (return) would be:
  + (105 - 100) / 100 \* 100 = 5%

In Pandas, you can use the pct\_change method to easily calculate this for a series of values, such as stock prices. This method computes the percentage change between the current and previous values in the series.

**How is percent change explained in the video?**

In the "Returns" video, percent change is explained as follows:

* The instructor uses the pct\_change method in Pandas to calculate returns.
* This method computes the percentage change between the current value and the previous value in a series.
* For example, if the closing price on January 5th is slightly higher than on January 4th, the percent change will be positive, indicating a small growth.
* Conversely, if the closing price on January 6th is lower than on January 5th, the percent change will be negative, indicating a loss.
* The instructor emphasizes the usefulness of the pct\_change method and recommends checking the documentation within Jupyter for more details on how it works.

This method helps in easily calculating and understanding the returns on financial data.

**Plotting returns**

**What is a histogram in pandas?**

A histogram in Pandas is a type of plot that shows the distribution of a dataset. It divides the data into bins (intervals) and counts the number of data points that fall into each bin. This helps visualize the frequency of different ranges of values.  
  
In the "Plotting returns" video, the instructor uses a histogram to better understand the distribution of stock returns. Instead of plotting the returns directly, the instructor calls the hist method on the data to create a histogram, which provides a clearer picture of whether the returns are mostly positive or negative.  
  
Here's a basic example of creating a histogram in Pandas:  
python  
data['returns'].hist(bins=30)  
  
This code will create a histogram with 30 bins, showing the distribution of the 'returns' data.

**What are bins in histogram?**

In a histogram, "bins" are intervals that divide the entire range of data into segments. Each bin represents a range of values, and the histogram shows how many data points fall into each bin.  
  
In the "Plotting returns" video, the instructor explains that:

* By default, Pandas creates 10 bins.
* You can adjust the number of bins to change the granularity of the histogram. For example, using 30 bins can provide a more detailed view of the data distribution.

Adjusting the number of bins can help you better understand the distribution of your data, such as whether returns are mostly positive or negative.

**What is bar plot and difference between bar plot and histogram?**

A **bar plot** is a type of chart that represents data with rectangular bars. Each bar's height or length corresponds to the value it represents. In the "Plotting returns" video, a bar plot is used to show whether returns are positive or negative over a specific period.  
  
**Differences between a bar plot and a histogram**:

* **Bar Plot**:
  + Displays individual data points as bars.
  + Each bar represents a specific category or value.
  + Used to compare different categories or values.
* **Histogram**:
  + Displays the distribution of a dataset.
  + Divides data into bins (intervals) and shows the frequency of data points in each bin.
  + Used to understand the distribution and frequency of data.

In the video, the instructor uses a histogram to show the distribution of returns and a bar plot to visualize individual returns over time.

Bottom of Form

**Cumulative returns**

* **Calculation of Cumulative Returns**:
  + **Simple Method**: Subtract the original price from the current price, then divide by the original price.
  + **Complex Method**: Use pct\_change to calculate percentage change, add 1, then use cumprod to get the cumulative product, and subtract 1.
* **Refactoring into Functions**:
  + The video demonstrates how to refactor the cumulative returns calculation into a function for reusability.
  + It introduces lambda functions as a way to create anonymous functions for concise code.
* **Visualization**:
  + The video shows how to plot cumulative returns using pandas and matplotlib.

This video helps you understand different ways to calculate and visualize cumulative returns using pandas, and how to refactor your code for better reusability.

**What is cumulative return?**

Cumulative return measures the total amount an investment has gained or lost over a period of time. It's calculated by taking the current price, subtracting the original price, and then dividing by the original price. This gives you a percentage that shows the overall return on the investment. For example, if an investment's price has increased from $10 to $20, the cumulative return would be 100%.

**How to calculate cumulative returns in pandas?**

To calculate cumulative returns in pandas, you can follow these steps:

1. **Subtract the original price from the current price:**  
   python  
   cumulative\_returns = close.sub(close.iloc[0])
2. **Divide by the original price:**  
   python  
   cumulative\_returns = cumulative\_returns.div(close.iloc[0])
3. **Plot the cumulative returns:**  
   python  
   cumulative\_returns.plot()

Alternatively, you can use the pct\_change and cumprod methods:

1. **Calculate percentage change and add 1:**  
   python  
   cumulative\_returns = close.pct\_change().add(1)
2. **Calculate the cumulative product and subtract 1:**  
   python  
   cumulative\_returns = cumulative\_returns.cumprod().sub(1)
3. **Plot the cumulative returns:**  
   python  
   cumulative\_returns.plot()

Both methods will give you the cumulative returns over time. You can choose the one that best fits your needs.

**What is the easy and the complex way of calculating cumulative returns?**

To calculate cumulative returns in pandas, the video explains two methods:  
  
**Simple Method:**

1. Subtract the original price from the current price.  
   python  
   cumulative\_returns = close.sub(close.iloc[0])
2. Divide by the original price.  
   python  
   cumulative\_returns = cumulative\_returns.div(close.iloc[0])

**Complex Method:**

1. Calculate the percentage change and add 1.  
   python  
   cumulative\_returns = close.pct\_change().add(1)
2. Calculate the cumulative product and subtract 1.  
   python  
   cumulative\_returns = cumulative\_returns.cumprod().sub(1)

Both methods will give you the cumulative returns over time. The simple method is more straightforward, while the complex method might be more efficient for larger datasets.

**What is lambda function and compare with normal function in the video.**

A **lambda function** in Python is a small, anonymous function defined using the lambda keyword. It can take any number of arguments but can only have one expression. Here's a comparison with normal functions as discussed in the video:  
  
**Normal Function:**

* Defined using the def keyword.
* Can have multiple lines of code.
* Has a name and can be reused.

Example from the video:  
def get\_returns(data\_frame):  
 return calc\_cumulative\_returns(data\_frame['close'])  
  
**Lambda Function:**

* Defined using the lambda keyword.
* Typically used for short, simple operations.
* Does not have a name (anonymous).

Example from the video:   
lambda df: calc\_cumulative\_returns(df['close'])  
  
In the video, the lambda function is used to simplify the code by creating an anonymous function that can be passed directly into methods like assign. This makes the code more concise and readable when performing operations like adding new columns to a DataFrame.

**Break down of lambda function step by step:**

1. **Lambda Function**:
   * A lambda function is a small, anonymous function defined using the lambda keyword. It can have any number of arguments but only one expression.
   * Syntax: lambda arguments: expression
2. **df**:
   * df is the argument for the lambda function. It represents a DataFrame, which is a data structure in pandas used to store tabular data.
3. **calc\_cumulative\_returns(df['close'])**:
   * This is the expression part of the lambda function. It calls the calc\_cumulative\_returns function, passing df['close'] as an argument.
   * df['close'] accesses the 'close' column of the DataFrame df.

Putting it all together:

* lambda df: calc\_cumulative\_returns(df['close']) defines a lambda function that takes a DataFrame df as input and returns the result of calc\_cumulative\_returns(df['close']).

In simpler terms, this lambda function:

* Takes a DataFrame as input.
* Extracts the 'close' column from the DataFrame.
* Passes this 'close' column to the calc\_cumulative\_returns function to calculate cumulative returns.

This allows you to use the lambda function in pandas operations like apply or pipe to process DataFrames efficiently.

Bottom of Form

**The complex method for calculating cumulative returns involves several mathematical steps:**

1. **Percentage Change**:
   * Calculate the percentage change between consecutive prices.
   * Formula: ((\text{current price} - \text{previous price}) / \text{previous price})
   * In pandas: close.pct\_change()
2. **Add 1**:
   * Add 1 to each percentage change to prepare for cumulative product calculation.
   * Formula: (\text{percentage change} + 1)
   * In pandas: close.pct\_change().add(1)
3. **Cumulative Product**:
   * Calculate the cumulative product of the values obtained in the previous step.
   * Formula: (\prod\_{i=1}^{n} (1 + \text{percentage change}\_i))
   * In pandas: close.pct\_change().add(1).cumprod()
4. **Subtract 1**:
   * Subtract 1 from the cumulative product to get the cumulative returns.
   * Formula: (\text{cumulative product} - 1)
   * In pandas: close.pct\_change().add(1).cumprod().sub(1)

This method leverages the power of pandas to perform efficient calculations on large datasets. It ensures that the cumulative returns are accurately computed by considering the compounded effect of returns over time.

Bottom of Form

17/02/2025

**Volatility**

* **Volatility Calculation**: The video explains how to calculate volatility using standard deviation in pandas, focusing on the percent change of the closing prices.
* **Rolling Windows**: It demonstrates how to apply rolling windows to calculate the standard deviation over a specified period (e.g., 30 days) to measure volatility.
* **Resampling Data**: The video also covers how to resample data (e.g., every 15 days) and calculate the standard deviation for each resampled period to understand volatility trends.

These concepts are crucial for performing more advanced financial analysis using Python.

**What is volatility?**

Volatility refers to the measure of how much the price of a financial instrument, like a stock, fluctuates over time. In the video, volatility is calculated using the standard deviation of the percent change in closing prices.  
  
Key points:

* **Standard Deviation**: This statistical measure indicates how much the prices vary from the average.
* **Rolling Windows**: By applying a rolling window (e.g., 30 days), you can calculate the standard deviation for each period, giving you a moving measure of volatility.
* **Resampling**: You can also resample data (e.g., every 15 days) to calculate the standard deviation for each resampled period, which helps in understanding volatility trends over different time frames.

This helps in assessing the risk and stability of financial instruments.

**What is rolling ? what is grouping? difference between rolling and grouping**

**Rolling**:

* Rolling refers to applying calculations over a moving window of a specified size. For example, a 30-day rolling window calculates the standard deviation for each 30-day period, moving one day at a time.
* This method is useful for getting a continuous measure of volatility over time.

**Grouping**:

* Grouping involves aggregating data based on specific intervals or categories. For example, resampling data every 15 days and then calculating the standard deviation for each 15-day period.
* This method provides a segmented view of volatility over defined intervals.

**Difference**:

* **Rolling**: Provides a continuous, overlapping measure by moving the window across the data.
* **Grouping**: Provides a segmented, non-overlapping measure by dividing the data into distinct intervals.

Both methods help in analyzing volatility but offer different perspectives on the data.

**How to calculate volatility in pandas?**

To calculate volatility in pandas, you can follow these steps as explained in the "Volatility" video:

1. **Calculate Percent Change**:
   * Create a new column for percent change in closing prices.  
     python  
     df['percent\_change'] = df['close'].pct\_change()
2. **Calculate Standard Deviation**:
   * Use the standard deviation of the percent change to measure volatility.  
     python  
     volatility = df['percent\_change'].std()
3. **Apply Rolling Window**:
   * Calculate the rolling standard deviation over a specified window (e.g., 30 days).  
     python  
     df['rolling\_volatility'] = df['close'].rolling(window=30).std()
4. **Plotting Volatility**:
   * Visualize the rolling volatility.  
     python  
     df[['rolling\_volatility']].plot()

These steps will help you measure and visualize the volatility of financial data using pandas.

Bottom of Form

17/03/2025

***Rolling Windows***

**Creating moving averages**

**What is a simple moving average?**

A simple moving average (SMA) is a calculation that helps smooth out data by creating a constantly updated average price.

* **Definition**: An SMA is the average of a selected range of prices, usually closing prices, by the number of periods in that range.
* **Calculation**: For example, a 3-day SMA would add up the closing prices of the last three days and divide by three.
* **Purpose**: It helps to identify trends by smoothing out fluctuations in the data.

The instructor demonstrates creating a 3-day moving average using a rolling window in Pandas, which calculates the mean of the specified number of periods. This helps in analyzing financial data more effectively.

**What is the lambda function and why is df\_ required in the video?**

A **lambda function** in Python is a small, anonymous function defined with the lambda keyword. It's used for creating small, one-time, and inline functions without needing to formally define a function using def.  
  
In the video "Creating moving averages," the instructor uses a lambda function to create a new column for the moving average. Here's why df\_ is required:

* **Lambda Function**: The lambda function is used to apply a function to each row of the DataFrame. In this case, it calculates the moving average using the columns S1 and S2.
* **df\_ Parameter**: df\_ represents the DataFrame being passed to the lambda function. It's necessary because the lambda function needs to reference the DataFrame to access its columns and perform calculations.

For example, the lambda function is defined as:  
lambda df\_: df\_.loc[:, ['close', 'S1', 'S2']].mean(axis=1)  
  
This means it takes the df\_ DataFrame, selects the columns close, S1, and S2, and then calculates the mean across these columns for each row.  
  
This approach helps in dynamically creating new columns based on existing data in the DataFrame.

**Difference between inbuilt and user created rolling window function in the video**

1. **Inbuilt Rolling Window Function**:
   * **Usage**: Utilizes Pandas' built-in rolling() method.
   * **Example**: df['close'].rolling(window=3).mean()
   * **Behavior**: Automatically handles missing values and only calculates the mean when there are enough data points (e.g., a 3-day window will only start calculating after the first 3 days).
   * **Advantage**: Simpler and more efficient as it leverages Pandas' optimized functions.
2. **User-Created Rolling Window Function**:
   * **Usage**: Manually creates new columns and calculates the mean using a lambda function.
   * **Example**:  
     python  
     df['S1'] = df['close'].shift(1)  
     df['S2'] = df['close'].shift(2)  
     df['MA3'] = df.apply(lambda df\_: df\_[['close', 'S1', 'S2']].mean(), axis=1)
   * **Behavior**: Provides more control over the calculation process, allowing for custom aggregations and handling of missing values.
   * **Advantage**: Flexibility to customize the rolling window calculation beyond the built-in methods.

Both methods achieve the same goal of calculating a moving average, but the built-in function is generally preferred for its simplicity and efficiency, while the user-created function offers more customization options.

**Plotting moving averages**

**What is golden cross and why do we need plot of moving averages of 50 and 200 days?**

The **Golden Cross** is a bullish signal in financial analysis that occurs when a short-term moving average (like the 50-day) crosses above a long-term moving average (like the 200-day). Here's why it's significant:

* **Indicator of Trend Reversal**: The Golden Cross suggests a potential shift from a downtrend to an uptrend, indicating a buying opportunity.
* **Market Sentiment**: It reflects positive market sentiment and increased buying pressure.

**Why plot 50 and 200-day moving averages?**

* **Short-term vs. Long-term Trends**: The 50-day moving average captures short-term trends, while the 200-day moving average represents long-term trends.
* **Smoothing Data**: These moving averages help smooth out price data, making it easier to identify trends and potential reversals.

***Technical Analysis***

**OBV**

In the video, the instructor demonstrates how to plot these moving averages to visualize the Golden Cross and analyze market trends effectively.

**Bottom of Form**

**Top of Form**

**Bottom of Form**