MIS4321
Computational Finance and Algorithmic Trading

Fall 2025

Lecture #4

Evaluating Trading Strategies and Backtesting

Evaluating Trading Strategies and Backtesting

Backtesting a trading strategy involves **testing a strategy on historical market data** to evaluate its effectiveness.

To backtest a trading strategy**, the common steps:**

1. **Define the trading strategy:** You need to have a clear understanding of the strategy you want to test. This includes the entry and exit rules, the position sizing, and risk management.

2. **Gather historical data:** You need historical market data **to test your trading strategy**. This data can be obtained from various sources such as online brokers, data vendors, or financial websites.

3. **Develop a trading algorithm:** You need to **code the trading strategy** into an algorithm that can automatically execute trades based on the historical data.

Evaluating Trading Strategies and Backtesting

4. **Run the backtest:** Run the backtesting algorithm on the historical data to **simulate the trades and generate performance metrics such as profit and loss,** win rate, and drawdown.

5. **Analyze the results:** Analyze the performance metrics to evaluate the effectiveness of the trading strategy. You can also **use the results to optimize the strategy** by adjusting the parameters or rules.

It's important to note that **backtesting** has limitations and **may not accurately reflect the future performance** of a trading strategy. Therefore, it's important to **use backtesting as a tool** for evaluating a strategy, **but not as the only factor in making trading decisions**.

Evaluating Trading Strategies and Backtesting

Backtesting is a **key phase to get statistics** showing how effective (or not) the trading strategy is.

Some important **statistics**:

Profit and loss (**P and L**): The money made by the strategy **without transaction fees.**

Net profit and loss (**net P and L**): The money made by the strategy **with transaction fees**.

Investment: The capital invested.

Number of trades: The number of trades placed during a trading session.

Building a Backtester

Backtesting helps us to generate results **showing risk and profitability** before risking any capital loss.

If the backtesting **returns good results** (high profits with reasonable risk), it will encourage getting this strategy to **go alive.**

If the results are not satisfactory, **backtesters can help to find issues.**

Trading strategies define rules for entry and exit into a portfolio of assets.

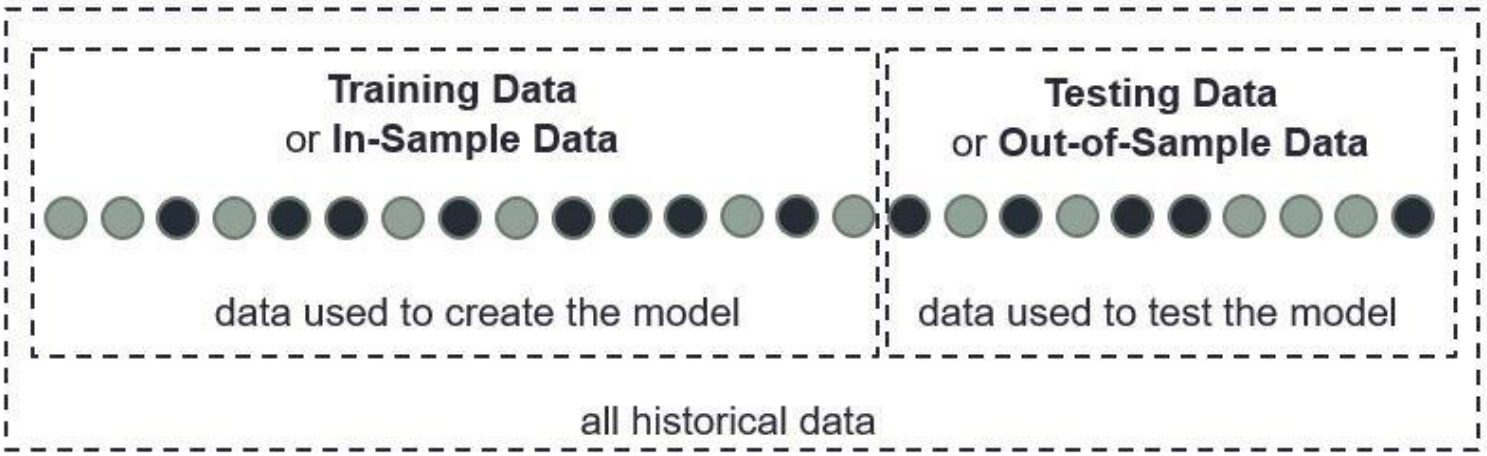
We will make assumptions based on the experience. But **if the model is not close enough to reality**, the trading strategies will end up not performing as well, which will **result in financial losses.**

Building a Backtester

In-sample versus out-of-sample data: When building a statistical model, we use cross-validation to avoid overfitting.

Cross- validation imposes a **division of data into two or three different sets**. One set will be used to create your model, while the other sets will be used to validate the model's accuracy.

It is important to set aside between 70% and 80% to build the model. When the trading model is built, the performance of this model will be tested on the out-of-sample data (20-30% of data).



Building a Backtester

Paper trading (forward testing): Paper trading (also known as forward performance testing) is **the final step of the testing phase**.

The trading strategy is **implemented on the real-time environment** of our system and **fake orders are generated**.

After a day of trading, we will **have the logs of all the orders** and compare them to what they were supposed to be.

This phase is a way to do **a last test of the trading strategy** before investing real money (**practice in a stress-free environment**).

The performance obtained by paper trading **is not directly correlated to the market.** It is difficult to **ensure that an order can be fulfilled, or not,** and at what price.

Building a Backtester

Naive data storage

One of the most common ways to store data is to use flat file on the hard disk. The problem with this approach is that **the hard disk will need to traverse a vast area to get to the part of a file** corresponding to the data you would like to use for your backtesting.

Databases

Financial data is time series data, and most databases do not handle time series data in the most efficient way. **The biggest challenge** associated with storing time series data **is scalability**. An important data stream comes rapidly.

HDF5 file

The **Hierarchical Data Format** (**HDF**) is a file format designed to store and manage large amounts of data. The trading world rapidly adopted this format, in particular, **High-Frequency Trading** (**HFT**) firms, hedge funds, and investment banks. These financial firms rely on **gigantic amounts of data** for backtesting, trading, and any other kinds of analysis.

The Assumptions in Backtesting

The fundamental assumption is that any methodology that functioned in the past is probably going to function in the future. **Any strategies that performed ineffectively in the past are probably going to perform inadequately in the future.**

It is impossible to obtain a model corresponding to reality; a backtester will just be a model of reality.

There are **rules to be followed** in order **to be as close as possible to the real market:**

Training/testing data: As with any models, you should not test your model with the data you use to create this model. You need to **validate the model on unseen data to limit overfitting** and improve the accuracy of your model.

The Assumptions in Backtesting

Survivorship-bias free data: Such datasets **only include historical data for stocks that are still actively trading.** For a long-term position strategy, **it is important to use the survivorship-bias free data** (that include also **delisted stocks**). This will **prevent you from focusing on winners alone without considering the losers.**

Look-ahead data: When you build a strategy, you **should not look ahead to make a trading decision.** This type of bias occurs **when a simulation relies on** data or **information that was not yet available** or known during the time period being studied. It generally **leads to inaccurate results** from a study or simulation.

Market change regime: Modeling **stock distribution parameters are not constant** in time because the market changes regime.

The Assumptions in Backtesting

Transaction costs: It is important to consider the transaction costs of your trading. It is **very easy to forget this cost** and results in loss of money in the real market.

Data quality/source: Since there are **many financial data sources**, **data composition differs a lot**. For instance, when you use OHLC (Open-high-low-close) data from Google Finance, it is an aggregation of many exchange feeds. It will be difficult to obtain the same highs and lows with your trading system. Indeed, in order **to have a match between your model and reality, the data you use must be as close as possible to the one you will use.**

Money constraint: Always consider that the **amount of money you trade is not infinite.** Additionally, if you use a credit/margin account, you will be limited by the position you take.

The assumptions in Backtesting

Average daily volume (ADV): The average number of shares traded over a day for a given ticker. **The quantity of shares** you choose to trade will be based on this number so as **to avoid any impact on the market**.

Benchmark testing: To test the performance of your trading strategy, you will **compare against another type of strategy** or just against the return of some indexes. **If you trade futures, do not test against the S&P 500.** If you **trade in airlines**, you should **check whether the airline industry as a whole performs** better than your model.

Initial condition assumption: You **should not depend on the day you start your backtesting or the month (**the initial condition is not always the same.)

The Assumptions in Backtesting

Fill ratio of orders: One of the first assumption you need to consider is the fill ratio. If you **trade with a high-frequency trading strategy, you may have 95% of the orders rejected.** If you trade when there are important news on the market (such as FED announcements), you may have most of your orders rejected. Therefore, you will need to **give a lot of thoughts on the fill ratio of your backtester**.

Order execution time: It is important to create an assumption regarding **when your order will be executed** (or maybe it will not be executed). This assumption will **add a condition to the backtester**. We may get additional data.

Latency assumptions: Each commponent of a trading system causes some latency such as generation, execution and acknowledge time for an order.

The Assumptions in Backtesting

Psychology: When we trade for real, **there is always a way to override (reverse) what the algorithm is doing.** After the backtest, a **trading strategy can have a large dropdown** but, then, this strategy can bring in high profit if we maintain a given position. **For a computer, there are no problems with taking that risk** but, for a human, it is more difficult. Therefore, psychology can play a large role in the performance of a strategy.

Market behave: On top of the prior rules, we will need to **assume how we expect the market to behave.**

When you present a trading strategy to anyone, **it is important to specify what these assumptions are.**
The list of assumptions can be pretty long but they are important **to explain how likely your trading strategy will perform** on the real market.

Take a break..

The type of Backtesting

For-loop backtest systems

The for-loop backtester is a very **simple infrastructure**.

It **reads price updates line by line** and calculates more metrics out of those prices (such as the moving average at the close).

It then **makes a decision on the trading direction**. The profit and loss is calculated and displayed at the end of backtester.

The design is very simple and **can quickly discern (recognize) whether a trading idea is feasible (or not).**

An algorithm to **picture how this kind of backtester works** is:

```
for each tick coming to the system (price update):
    create_metric_out_of_prices()
    buy_sell_or_hold_something()
    next_price()
```

The type of Backtesting

Advantages of For-loop backtest systems

- The for-loop backtester is **very simple** to comprehend (understand).
- It can be **easily implemented in any programming language**.
- The main functionality of this type of backtester is to read a file and calculate new metrics **based on price alone.**
- Complexity and the need for **calculating power are very low.** Therefore, **execution does not take too long** and it is quick to obtain results regarding the performance of the trading strategies.

The type of Backtesting

Disadvantages of For-loop backtest systems

- **The main weakness** of the for-loop backtester **is accuracy** in relation to the real market.

- It **neglects** transactions costs, transaction time, the bid and offer price, and volume.

- The likelihood of making a mistake by **reading a value ahead of time** is pretty high (look-ahead bias).

The code of a for-loop backtester is easy to write, thus we could still use this type of backtester **to eliminate low-performance strategies**.

*If a strategy does not perform well with for-loop backtesters, this means that it will perform even worse on more realistic backtesters.

The type of Backtesting

Event-driven backtest systems

An event-driven backtester uses almost all the components of the trading system (such as the order manager system, the position manager, and the risk manager).

The components will read the input one after the other and will then generate events if needed (or triggered).

Important events we encountered when we coded the trading system were the following:

Tick events – When we read a new line of market data
Book events – When the top of the book is modified
Signal events – When it is possible to go long or short
Order events – When orders are sent to the market
Market response events – When the market response comes to the trading system

The type of Backtesting

Event-driven backtest systems

The market simulator component **must have the same market assumptions.** We can add the following parameters to the market simulator (backtester):

- Latency to send an acknowledgement
- Latency to send an order
- An order filling condition
- A volatility filling condition
- A market making estimate

The type of Backtesting

Advantages of Event-driven backtest systems

- Because we use all the components, we will have a result that **more closely corresponds to reality.**
- **Look-ahead bias elimination**—since we receive events, we cannot look at the data ahead.
- **Code encapsulation**—because we use objects for the different parts of the trading system, **we can just change the behavior of our trading system by changing the objects.**
- We can insert a **position/risk management system** and check whether we do not go against the limit.

The type of Backtesting

Disadvantages of Event-driven backtest systems

Event-based system is **difficult to code.**

Indeed, if there are threads in the trading system, we **need to make this thread deterministic.** For instance, let's assume the trading system **takes care of timing out if an order doesn't get a response within 5 seconds**. The best practice to code this functionality would be to **have a thread counting 5 seconds and then timing out.**

If we use the thread in backtesting, **the time shouldn't be the real time** because when we read the tick, the time will be the simulated time.

It requires a lot of handling, such as log management, unit testing, and version control. **The execution of this system can be very slow.**

Value of Time in Backtesting

Evaluating the value of time in Backtesting

When we use the time in production, we can get the time from the **computer's clock**.
For instance, we can stamp a book event coming to the trading strategy by just getting the time from the function ***now*** coming from the **datetime** module in Python.

By way of another example, suppose **we place an order**. Because **it is unsure whether the market will respond to this order**, we will use a timeout system. This timeout system will call a function after a given period of time if no acknowledgement has been received by the trading system from the market.
To accomplish this operation**, we usually spawn a thread counting the number of seconds up to the timeout time.** When counting, if the state of the order has not changed to acknowledge the order, this thread will call a **callback function**, **onTimeOut**.
Because **we cannot use the real-time clock of the machine** to count to the timeout time, we will need to **use a simulated clock during the whole process**.

Next week

- **Financial Data Storage**

Thank you for your participation ☺