NOVEMBER 11, 2025

UTEFA QuantiFi

Launch

Can you beat the market?

QuantiFi challenges you to put your analytical and coding skills to the test in a real-world trading simulation. You'll **build** a data-driven trading strategy, **backtest** its performance, and **compete** against your peers to see who can generate the highest returns.

Whether you're interested in finance, data science, or algorithmic trading, this competition will give you **hands-on experience** with the quantitative methods used by professional traders

Goals

QuantiFi provides the opportunity to

1. **Apply Quantitative Methods to Finance** - Use factor analysis, statistical modeling, and portfolio optimization techniques to develop a trading strategy

2. **Gain Practical Coding Experience** - Implement your strategy in Python and learn how to backtest trading algorithms programmatically

3. **Understand Risk and Performance** - Evaluate your strategy using real performance metrics like returns, volatility, Sharpe ratio, and drawdown

QuantiFi provides the opportunity to

4. **Develop Professional Communication Skills** - Present your methodology and results to industry professionals in a clear, compelling pitch

5. **Bridge Theory and Practice** - See how academic concepts in finance, statistics, and computer science come together in real-world scenarios

6. **Learn by doing** - This isn't just about getting the right answer—it's about understanding the process, testing your ideas, and iterating on your approach.

Introducing our Judge

Robert Grzesiuk

Quantitative Analyst - Credit, OMERS Capital Markets

Indy 2T2 + PEY

<u>Previous Experience:</u>

- Business Systems Analyst, RBC Capital Markets
- Trade Floor Technology Consultant, Scotiabank

Instructions

Three Files Given to You

1. Dataset (.csv)
2. Contestant Template (.py)
3. Backtesting Script (.py)

Dataset (.csv)

- **5** stocks (A - E)
- **252** days (1 trading year) of historical price data
- Factors
  - Macro: Interest Rate (RFR), Economic Growth, Inflation
  - Micro: Volume, Momentum (10d)
  - Suggested: Moving Average, Volatility, and more…

Dataset (.csv)

Use it to:

- **Train** your model
- **Validate** your performance
- **Test** your strategy
- **Calculate** indicators

Pro tip: Use the 70-20-10 rule to split the dataset into training, validation and testing sets.

You can also create your own independent test set.

Note:

The model you submit will be evaluated using a different dataset, hidden to you, like the real world.

Contestant Template (.py)

This is your starting point. It contains three classes you'll work with:

Market: Represents the stock exchange

Has current prices for all 5 stocks

Has the transaction fee (0.5%)

You READ from this class - **do not modify it**

Portfolio: Your trading account

Tracks your cash balance and how many shares you own of each stock

Has buy() and sell() methods for trading

Has helper methods to calculate position values

Context: Your scratch space

Store ANYTHING you need here: price history, indicators, signals, counters, whatever

Contestant Template (.py)

You have to implement ONE function: update_portfolio()

This function is called once per trading day, BEFORE the market prices update. You use it to make your trading decisions.

Important notes:

- A transaction fee of 0.5% is applied to each buy and sell order you place.
- Your portfolio value (= cash + positions) must always be non-negative.
- Your model must run under 60 seconds.

Backtesting Script (.py)

This is the ACTUAL grading script we'll use. We're giving it to you so you can:

- Test your strategy in the exact same environment we'll use for grading
- See detailed performance reports
- Verify you're under the 60-second timeout
- Get transaction summaries

Make sure:

- The backtesting script, your contestant template (which contains your model), and your test dataset are in the same directory.
- The test dataset has only price data for each of the 5 stocks for 252 days and no factors
  - i.e. the first row in your .csv file should be *Day,Stock_A,Stock_B,Stock_C,Stock_D,Stock_E*

Presentations

6 minutes in length, followed by 2 minutes of Q&A.

A good presentation contains:

- Well-explained rationale for your chosen model.
- Why it works well for this task?
- What trade-offs did you consider?
- What challenges you faced, and how did you resolve them?
- Include your back-tested results (graphical visualizations, metrics, etc)
    - Something other than what the backtesting script already does
- Balanced speaker distribution

Important Dates

Tuesday, November 11:

- Launch

Wednesday, November 19 @ 6 pm:

Deadline to submit your modified contestant template (.py)

- Email to utefa@g.skule.ca. Subject: QuantiFi <Team#>
- One submission per team

Thursday, November 20 @ 6 pm:

- Email slide final presentation slide deck to to utefa@g.skule.ca. Subject: QuantiFi Deck <Team#>
- Main event day

Python Basics

Data Structures

Lists - Ordered collections you can modify

```
prices = [100, 102, 98, 105]  # Store historical prices
last_price = prices[-1]       # Access last element: 105
prices.append(107)            # Add new price
```

Use case: Track historical data like past 10 days of prices for momentum calculations

Dictionaries - Store key-value pairs for fast lookups

```
stocks = {"Stock_A": 123.0, "Stock_B": 456.0}
price = stocks["Stock_A"]     # Access by key: 123.0
stocks["Stock_C"] = 234.0     # Add/update entry
```

Use case: The Market and Portfolio classes use dicts to map stock names to prices/shares

Tuples - Immutable ordered collections

```
position = ("Stock_A", 100, 123.0)  # (stock, shares, price)
stock_name = position[0]            # Access by index
```

Use case: Store fixed data like trade records or return multiple values from functions

List Operations

Appending data: *context.price_history.append(current_price)*

Accessing recent values: *context.price_history[-1]* (last element)

Slicing for momentum: *context.price_history[-10:]* (last 10 days)

Classes and Objects

Classes: Blueprints that define how objects store data and behave

```
class Portfolio:
    def __init__(self):
        self.cash = 100000
        self.shares = {"Stock_A": 0, "Stock_B": 0, "Stock_C": 0, "Stock_D": 0, "Stock_E": 0}
```

Objects: Specific instances created from a class

```
my_portfolio = Portfolio()  # Create an object
print(my_portfolio.cash)    # Access attribute: 100000
```

Functions and Parameters

What is a Function? - Reusable block of code that performs a specific task

```
def calculate_total(price, shares):
    return price * shares

total = calculate_total(100, 50)  # Call function: returns 5000
```

Functions and Parameters

Your function to implement: update_portfolio()

```
def update_portfolio(market, portfolio, context, day):
    # Your trading logic goes here
    price = market.stocks["Stock_A"]
    if price < 100 and portfolio.cash > 10000:
        # Buy Stock_A
```

Parameters Explained:

market → Object from the Market class (current prices & fees)

portfolio → Object from the Portfolio class (your cash & shares - modify this to trade!)

context → Object from the Context class (your custom data storage)

day → Integer representing current trading day (0 to 251)

Be careful of…

- Integer vs float division

- You cannot buy fractional shares, so always either:
  - round down using the int(x) typecast, or
  - round to nearest with round(x)

Important Dates

Tuesday, November 11:

- Launch

Wednesday, November 19 @ 6 pm:

Deadline to submit your modified contestant template (.py)

- Email to utefa@g.skule.ca. Subject: QuantiFi <Team#>
- One submission per team

Thursday, November 20 @ 6 pm:

- Email slide final presentation slide deck to to utefa@g.skule.ca. Subject: QuantiFi Deck <Team#>
- Main event day

Questions?