Course Project: Hull Tactical – Market Prediction Competition

1. Project Overview

1.1 Background of the Kaggle Competition

The Hull Tactical – Market Prediction challenge on Kaggle invites participants to **predict the daily excess returns of the S&P 500 index and design a trading strategy that dynamically adjusts the portfolio's exposure to the index**. Excess returns are the returns beyond the risk-free rate (e.g., Treasury bill rates), and *market exposure* refers to how much of your portfolio is invested in the S&P 500. The goal is to test whether sophisticated data science techniques can uncover patterns in financial markets, challenging the Efficient Market Hypothesis (EMH), which posits that it is impossible to consistently beat the market because prices already reflect all available information.

The competition description notes that although conventional wisdom discourages investors from trying to "time the market," machine-learning models combined with macro-economic indicators, market dynamics, interest rates, volatility and sentiment data may reveal repeatable signals. Participants are asked to blend these diverse features into a robust predictive model.

1.2 What You Must Predict

- **Target –** For each trading day in the test set, you must output a weight between 0 and 2. This value represents the fraction of your portfolio invested in the S&P 500 on that day. A weight of 0 means holding only cash (no exposure), 1 means fully invested, and 2 means the portfolio is 200 % long the index. Some leverage is allowed up to a maximum weight of 2.
- **Objective –** Your goal is to maximize risk-adjusted returns rather than just raw returns. The evaluation metric penalizes strategies whose volatility is substantially higher than that of the market or that fail to outperform the market's return. The winning strategy must therefore generate superior returns while keeping risk close to the underlying index.

1.3 Data Description

https://www.kaggle.com/competitions/hull-tactical-market-prediction/data

Competition phases and data updates

The competition proceeds in two distinct phases. In the model training phase the organizers provide a test set of roughly six months of historical market data. Because these prices are already public, leaderboard scores during this phase are not meaningful. After submissions close the contest enters a forecasting phase in which a new test set is collected; the scored portion of this set is expected to be about the same size as in the training phase. During forecasting, the evaluation API streams test data from the start of the public set through to the end of the private set. This stream includes trading days before the submission deadline that are not scored, and the first date_id served remains constant throughout the competition.

Files

Key files include:

- **train.csv** – Historic market data covering decades; early observations contain many missing values. Each row represents a trading day and includes hundreds of features grouped into categories such as M* (market dynamics/technical indicators), E* (macro-economic indicators), I* (interest rate variables), P* (price/valuation metrics), V* (volatility measures), S* (sentiment variables), MOM* (momentum indicators) and D* (dummy/binary flags). The training set also provides the target variable forward_returns (the return from buying the S&P 500 and selling it a day later), risk_free_rate (federal funds rate) and market_forward_excess_returns (forward returns relative to expectations).
- **test.csv** – A mock test set that mirrors the structure of the unseen private test set. The version used for the public leaderboard is simply a copy of the last 180 date IDs from the training data, so public leaderboard scores are not meaningful. Each row contains the same feature columns as train.csv and additional fields: is_scored flags whether the row contributes to the evaluation metric (only true for the first 180 rows during model training), lagged_forward_returns (previous day's return from buying and selling the S&P 500), lagged_risk_free_rate (risk-free rate lagged by one day) and lagged_market_forward_excess_returns (lagged forward excess returns).
- **kaggle_evaluation/** – A directory containing Python modules and protocol files used by the evaluation API. The demo submission on the competition page shows how to use these files to score your predictions. The evaluation API may be updated during the model-training phase, and once the competition ends the organizers plan to periodically publish market data on their website for participants' own analysis.

These files collectively provide the features and targets needed for training, testing and evaluating your models. Remember that only the *scored* portion of the test data counts toward your public leaderboard position, and the final ranking will be determined on a hidden test set.

2. Competition Requirements & Evaluation

2.1 Evaluation Metric

https://www.kaggle.com/code/metric/hull-competition-sharpe

This competition uses a **volatility-adjusted Sharpe ratio** as its evaluation metric. To make the scoring logic easier to follow, the organizers have provided a Python implementation. Below we break the code into logical pieces and pair each snippet with a plain-English explanation so that you can understand how each line contributes to the final score.

1. **Define constants and custom exception**

```
MIN_INVESTMENT = 0
 MAX_INVESTMENT = 2

 class ParticipantVisibleError(Exception):
   pass
```

These constants set the minimum and maximum allowable portfolio weight. A weight of 0 means no exposure to the market, while 2 represents 200 % long the index (using leverage). The custom exception `ParticipantVisibleError` is raised whenever a submission violates basic rules so that participants can see clear error messages.

2. **Define the scoring function and its documentation**

```
def score(solution: pd.DataFrame, submission: pd.DataFrame, row_id_column_name: str) ->
float:
   """
   Calculates a custom evaluation metric (volatility-adjusted Sharpe ratio).
   This metric penalizes strategies that take on significantly more volatility
   than the underlying market.

   Returns:
     float: The calculated adjusted Sharpe ratio.
   """
```

The `score` function accepts two pandas DataFrames: `solution` contains the ground-truth values (`forward_returns` and `risk_free_rate`) and `submission` contains your predicted weights (in a column named `prediction`). The docstring clarifies that the metric is a modified Sharpe ratio that penalizes strategies with high volatility.

3. **Ensure predictions are numeric**

```
if not pandas.api.types.is_numeric_dtype(submission['prediction']):
   raise ParticipantVisibleError('Predictions must be numeric')
```

Before any calculations, the scoring code checks that your `prediction` column is numeric. Submissions with non-numeric values (e.g., strings or missing values) immediately fail. This guard prevents invalid portfolio weights from causing obscure errors later.

4. **Assign positions and enforce bounds**

```
solution = solution
solution['position'] = submission['prediction']

if solution['position'].max() > MAX_INVESTMENT:
  raise ParticipantVisibleError(f'Position of {solution["position"].max()} exceeds maximum
of {MAX_INVESTMENT}')
if solution['position'].min() < MIN_INVESTMENT:
  raise ParticipantVisibleError(f'Position of {solution["position"].min()} below minimum
of {MIN_INVESTMENT}')
```

Your predicted weights are copied into a new `position` column on the `solution` dataframe. The next two checks verify that the largest position does not exceed **2** and the smallest does not fall below **0**. If either limit is violated, the function raises an error and your submission receives no score.

5. **Compute strategy returns**

```
solution['strategy_returns'] = solution['risk_free_rate'] * (1 - solution['position']) \
   + solution['position'] * solution['forward_returns']
```

Each day's **strategy return** is a blend of the risk-free rate and the index's forward return. When your position (weight) is less than 1, part of your capital stays in the risk-free asset; when it exceeds 1, you borrow at the risk-free rate to lever up your exposure to the S&P 500. This line computes the resulting daily return for your strategy.

6. **Compute the strategy's Sharpe ratio**

```
strategy_excess_returns = solution['strategy_returns'] - solution['risk_free_rate']
strategy_excess_cumulative = (1 + strategy_excess_returns).prod()
strategy_mean_excess_return = strategy_excess_cumulative ** (1 / len(solution)) - 1
strategy_std = solution['strategy_returns'].std()

trading_days_per_yr = 252
if strategy_std == 0:
  raise ParticipantVisibleError('Division by zero, strategy std is zero')
sharpe = strategy_mean_excess_return / strategy_std * np.sqrt(trading_days_per_yr)
strategy_volatility = float(strategy_std * np.sqrt(trading_days_per_yr) * 100)
```

The function next calculates your strategy's excess returns by subtracting the risk-free rate. Multiplying all `(1 + excess_return)` terms yields a cumulative growth factor; taking the `len(solution)`-th root then subtracting 1 gives the geometric mean excess return. The standard deviation of daily returns (`strategy_std`) measures volatility. Both the mean return and standard deviation are annualized using $\sqrt{252}$ trading days to produce a raw Sharpe ratio (`sharpe`) and the annualized volatility in percent (`strategy_volatility`). If there is no variation (`strategy_std == 0`), an error is raised to avoid dividing by zero.

7. **Compute the market benchmark**

```
market_excess_returns = solution['forward_returns'] - solution['risk_free_rate']
market_excess_cumulative = (1 + market_excess_returns).prod()
market_mean_excess_return = market_excess_cumulative ** (1 / len(solution)) - 1
market_std = solution['forward_returns'].std()
market_volatility = float(market_std * np.sqrt(trading_days_per_yr) * 100)

if market_volatility == 0:
   raise ParticipantVisibleError('Division by zero, market std is zero')
```

To benchmark your performance, the same steps are applied to the S&P 500's forward returns. The market mean excess return and market volatility provide a baseline for comparison. If the market's volatility is zero, the function raises an error.

8. **Compute volatility and return penalties**

```
# Volatility penalty
excess_vol = max(0, strategy_volatility / market_volatility - 1.2) if market_volatility >
0 else 0
vol_penalty = 1 + excess_vol

# Return penalty
return_gap = max(
  0,
  (market_mean_excess_return - strategy_mean_excess_return) * 100 * trading_days_per_yr,
)
return_penalty = 1 + (return_gap ** 2) / 100
```

Two penalty factors adjust the raw Sharpe ratio:

- The volatility penalty measures how much more volatile your strategy is than the market. It divides `strategy_volatility` by `market_volatility`, subtracts 1.2 (a 20 % tolerance) and retains only positive excess volatility.
- The return penalty penalizes strategies whose average excess return is below the market's. It computes the difference, scales it by `100 * trading_days_per_yr` and squares the result to amplify large gaps. If your strategy outperforms the market, the return gap is zero and the penalty factor remains 1.

9. **Compute the adjusted Sharpe ratio**

```
adjusted_sharpe = sharpe / (vol_penalty * return_penalty)
return min(float(adjusted_sharpe), 1_000_000)
```

The final score is obtained by dividing the raw Sharpe ratio by both penalty factors. This adjusted Sharpe ratio rewards strategies with high risk-adjusted returns and penalizes those that are excessively volatile or that underperform the market. To prevent pathological outputs, the score is capped at 1,000,000.

In summary, this evaluation function rewards strategies that generate high excess returns relative to the risk-free rate while maintaining volatility close to the market and not underperforming the market's mean excess return. When preparing your Kaggle submission, ensure your weight predictions remain within [0, 2]; otherwise the evaluation script will raise an error and assign a score of zero.

2.2 Code Requirements (Kaggle)

http://kaggle.com/docs/competitions#notebooks-only-FAQ

Kaggle imposes strict submission rules:

- **Kaggle Notebooks only** – You must submit via a Kaggle Notebook. Local scripts or external servers are not accepted.
- **Internet disabled** – External internet access is disabled. You can use only packages provided by Kaggle and any external data you upload yourself.

- **External data** – Freely and publicly available external datasets and pretrained models are allowed, as long as they are included in your Notebook or are publicly accessible on Kaggle.
- **Evaluation API** – Use the provided kaggle_evaluation module to calculate your score. The example Notebook on the competition page demonstrates proper usage. https://www.kaggle.com/code/sohier/hull-tactical-market-prediction-demo-submission

3. Detailed Kaggle Submission Guide

3.1 Register and Join the Competition

1. Create a Kaggle account at kaggle.com if you don't already have one.
2. Navigate to the *Hull Tactical – Market Prediction* competition page and click "Join Competition." Read and accept the competition rules. Kaggle's rules prohibit participating under multiple accounts.
3. After joining, go to the Data tab to explore the dataset and either download `train.csv` and `test.csv` or access them directly in a Kaggle Notebook.

3.2 Create Your Notebook & Build a Model

1. On the competition page, click "New Notebook" to create a Kaggle Notebook. Click "Add Input" and add Hull Tactical – Market Prediction Competition. The dataset is mounted at `/kaggle/input/hull-tactical-market-prediction/`.
2. Perform exploratory data analysis. Examine how various features relate to `forward_returns` and other derived targets.
3. Develop a model that predicts the optimal `weight` for each date in test.csv.
4. Use the Kaggle Evaluation API by importing kaggle_evaluation from the provided `kaggle_evaluation/` directory. Call the evaluation function to compute the modified Sharpe ratio on a validation set.

3.3 Prepare and Submit Your Notebook

1. Ensure your Notebook outputs predictions for **all rows** in test.csv. Create a DataFrame with columns date_id and weight, clipping weight values to the [0, 2] range.
2. At the end of the Notebook, call the Evaluation API to compute your score. Once satisfied, you can remove extra validation logic to reduce runtime.
3. Commit the Notebook. Kaggle will execute it. If the run completes successfully, a "Submit to Competition" button will appear.
4. Click "Submit to Competition." Kaggle will compute a Public Score, which is published on the leaderboard.
5. You can submit multiple times (subject to Kaggle's daily submission limit, typically 5).

3.4 Troubleshooting

https://www.kaggle.com/code-competition-debugging

https://www.kaggle.com/docs/competitions#notebooks-only-FAQ

4. Course Project Submission Requirements

For this course project, you will use your participation in the Hull Tactical competition as a practical project. You must submit:

1. **Code –** In addition to the **Kaggle Notebook (.ipynb)** used to generate your final predictions, you must provide the **complete code for to reproduce your project**, such as any scripts or functions used to preprocess the data, train the model, and tune hyperparameters. The code should be complete, well-documented, and allow someone else to reproduce your results from start to finish.
2. **Report –** A comprehensive PDF report explaining your approach. Include:
    - Introduction – Reframe the problem and its significance.
    - Data exploration & feature engineering – Describe how you handled financial time-series data, any feature transformations, missing-value handling, and external data used.
    - Model selection & training – Explain which algorithms were tried, why you chose the final model, and how you managed volatility and overfitting.
    - Evaluation – Show how you used the Evaluation API, present validation results, and discuss how your strategy balances return and risk. Include your Kaggle Public Score as reference.
    - Conclusion – Summarize what worked, challenges faced, and possible improvements.
    - Citations – Attribute any external resources.

    Note: The report is **limited to a maximum of 8 pages**(excluding references). Ensure that your writing is concise and covers all necessary aspects of your approach within this limit.

    Formatting requirement: Please typeset your report using the attached conference LaTeX template (ICLR-style, “iclr2026.zip” template). Follow the template’s sectioning, fonts, margins, and figure/table styles. Include author names and team info (no anonymization). **Compile to PDF for submission.**
3. **Kaggle score –** A screenshot or link to your public leaderboard entry, with the numeric score included. You do not need to wait for the forecasting phase or submit a score from the final test set. Only the current score from the public leaderboard during the training phase is required.

4.1 Deadlines & Team Rules

- **Deadline – 9 December 2025 at 23:59**. Late submission: 0-48 hours -> 50%, more hours -> 0%.
- **Submission format –** Upload a single zip file to BB containing your code, report, and Kaggle score evidence.
- **Team size –** You may work individually or in teams of up to **five members**. **Cross-class teams are allowed.** If you form a team, the final zip file could be uploaded by one of you, but ensure all members are listed in your submitted report.

- **Grading criteria are identical regardless of team size.** All members of the **same team receive the same score**.
- **Academic integrity –** High-level collaboration is permitted, but copying code or results without attribution is prohibited. Cite any public Kaggle notebooks or external sources that influence your work.

4.2 Assessment

Your course project will be assessed based on the quality of your **code** and **report**. In this course, we **prioritize completeness and the correct application of course concepts to a real financial setting**. Strong leaderboard performance or novelty is not required for a high grade. Even if your final model performs modestly, you can still earn a high score provided your reasoning is clear, your design choices are justified, your workload is substantive, and your work is reproducible.

The Kaggle leaderboard ranking will **not** affect your final grade for the course project. However, the **top three teams in the class**, based on their performance in the Kaggle competition, will receive prizes as recognition of their efforts.

If any student has a particularly strong idea or wants to push for a top Kaggle ranking, feel free to contact the professors for additional guidance or support in refining your model and strategy for the competition.