Introduction

PRECOG RECRUITMENT TASKS 2026

Tasks are centered around four central themes

- NLP
- Computer Vision
- Graphs
- Quant

Applicants must choose **any one** theme and complete the task listed under that theme.

You are encouraged to draw upon existing literature, blogs, and other credible resources to inform your design decisions. While doing so, you are expected to clearly justify the choices and assumptions you make.

You are not required to follow the architectures or methods suggested in the task description, if any. Feel free to design and experiment with your own approaches, modify existing ideas, combine techniques, or explore entirely new directions.

Importantly, all experiments are valuable, including those that do not yield positive results. Any approach you attempt, successful or not should be documented and discussed in your report or presentation.

We **strongly recommend reviewing relevant prior work and appropriately citing the literature** wherever it informs your methodology, analysis, or implementation.

For some tasks there are bonus tasks - to demonstrate that you can go the extra mile (which is an important characteristic of being in our group)! We encourage you to try those bonus tasks.

To overcome compute constraints:

- You can use Google Colab, Kaggle for compute intensive jobs.
- Take a subset of data if the dataset is huge - ex. 25% / 50% / 75% of train data

You must attempt this task on your own. **Please make sure you cite any external resources that you may use.** We will also check your submission for plagiarism, including ChatGPT :)

Submission:

- Put all your code/notebooks in a GitHub repository. Maintain a README.md explaining your codebase, the directory structure, commands to run your project, the dependency libraries used and the approach you followed.
- A **Python notebook is mandatory** and must clearly log all experiments and results. Submissions consisting only of Python scripts **without accountable, logged results** will not be considered. All outcomes must be transparent, reproducible, and traceable within the notebook.
- The link to the GitHub repository will be asked for during the interview.

- Presentation / Report:
    - Programming Task : Document the process and compile a presentation / report **summarizing your findings, methodologies, and insights** gained from the analysis in a presentation/report. Your report must contain your methodology, findings, results etc. On the first page, It must detail exactly what parts of the task you did, and what parts of the task you did not do and why. The report would be one of the main documents that would help us take your application forward for the next stages.

Evaluation:

- You will be evaluated based on how you approach the problem, and not so much on the performance measures like accuracy etc. Although, we would expect you to beat random performance.
- How you present your code and findings. You should justify all the choices you make regarding data, model, and hyperparameters that you use. You should also be able to demonstrate theoretical understanding of the approaches used.
- Across all the tasks, your experiments will give you some quantitative measures to indicate the efficacy of your approach (Accuracy, Recall, MAE, MSE etc) - but dig deeper to analyze the predictions. Can you come up with any hypothesis for why your approach fails/succeeds? Can this analysis help you improve on your approach? Creativity in this analysis is what we are looking for!! - SURPRISE US!!.
- How do you handle and sample from large real-world datasets, and solve tasks with whatever resources you have access to..

For any doubts regarding this task please directly email to the address provided for each task, add the following email in cc - george.rahul@research.iiit.ac.in **and** debangan.mishra@students.iiit.ac.in. If you don't get a reply within 24 hrs feel free to drop a reminder.

Tasks

- The CV Task can be found in this tab CV.
- The NLP Task can be found in this tab NLP.
- The Graphs Task can be found in this tab Graphs.
- The Quant Task can be found in this tab Quant.

“If you wait until you’re ready, you’ll be waiting the rest of your life.” - Lemony Snicket

CV

The Lazy Artist

"The eye sees only what the mind is prepared to comprehend." — *Henri Bergson*

Modern Convolutional Neural Networks (CNNs) are incredibly powerful, but they are also incredibly lazy. They will cheat whenever possible. If you train a model to classify "Wolves" vs "Dogs," and all your wolf pictures have snow in the background, the model might not learn what a wolf looks like. It may instead learn to detect snow.

In this task, you are not just an engineer; you are a **psychologist for neural networks**. You will deliberately traumatize a model with a biased dataset, diagnose its "mental block" using interpretability tools, and then invent a cure to force it to learn properly.

Task 0 - The Biased Canvas

"Illusion is the first of all pleasures." — *Voltaire*

We need a dataset that lies. Since downloading real "biased" datasets is heavy, you will synthetically generate one using **MNIST** (or Fashion-MNIST).

You must write a script to create a **Colored-MNIST** dataset with a specific **Spurious Correlation**:

1. **The Signal (Digit):** The standard MNIST digits (0-9).
2. **The Noise (Color):** You will assign a specific color to specific digits, but **not 100% of the time**.

The Rules of the Bias:

- **The "Easy" Set (Train):**
  - 95% of all `0`s are rendered in **Red**.
  - 95% of all `1`s are rendered in **Green**.
  - ... (Assign a dominant color to each digit).
  - The remaining 5% are colored randomly (Counter-examples).
- **The "Hard" Set (Test):**
  - The correlation is inverted or randomized. `0`s are never Red. `1`s are never Green.

The Twist: The background shouldn't just be a solid flat color (too easy). The color should be applied to the *foreground stroke* or a *background texture*.

Task 1 - The Cheater

"All models are wrong, some are useful." — *George Box*

Train a standard CNN (ResNet-18 or a simple 3-layer CNN) on your **Easy (Train)** set.

1. **Training:** It should achieve extremely high accuracy (>95%) on the Easy Train set and the Easy Validation set. The model will think, "Oh, Red = 0. Easy."
2. **The Trap:** Evaluate this model on the **Hard (Test)** set.
3. **The Result:** The accuracy should plummet (likely < 20%).

Analysis:

- Show the Confusion Matrix.
- Prove that the model is looking at color. Feed it a Red `1`. Does it predict `0`?

Task 2- The Prober

"All models are wrong, some are useful." — *George Box*
Can you now see what the neurons in the trained CNN are seeing? Start with an optimizable image tensor and optimize it to maximize the raw activations / channelwise mean activation / neuron's spatial activation mean etc. (your choice, feel free to explore and see what works best or gives the most interesting results) and try to see concretely what a neuron "sees".

You can use OpenAI Microscope as a reference and simplify their procedure. You can explore a wide range of neurons and try to evaluate if they focus more on color, sub-parts of a digit or more. While you are at it, do take some time to explore polysemanticity of neurons. NOTE: This is a very open ended task, and there are no concrete metrics. Explore multiple neurons.

Task 3 - The Interrogation

"To see is to know."

You know the model is cheating. Now you must prove it mathematically. You cannot use libraries like `pytorch-gradcam`. You must implement **Grad-CAM (Gradient-weighted Class Activation Mapping)** from scratch.

- **The Math:** Hook into the final convolutional layer of your Task 1 model. Calculate the gradients of the target class score with respect to the feature maps.
- **The Visual:** Overlay the Heatmap on the original image.
- **The "Aha!" Moment:**
    - Take a "biased" image (e.g., Red `0`). Does the heatmap focus on the shape of the zero, or does it "smear" across the colored pixels?
    - Take a "conflicting" image (e.g., Green `0`). Where does the model look?

Task 4 - The Intervention

"Constraint inspires creativity."

Now, retrain the model to ignore the color and focus on the shape, **without** converting the image to grayscale and **without** changing the dataset (you still have the 95% bias).
You must implement a custom training strategy. Get creative here, and implement at least 2 methods that you think might work out here. Start with a hypothesis about what feels intuitive and then see if they work out. Some methods you might want to consider are – adding a color penalty or saliency guides.

Evaluation: Your new model must achieve respectable accuracy (>70%) on the **Hard Test Set** (the one with swapped colors), despite being trained on the biased set.

Task 5: The Invisible Cloak

"Reality is merely an illusion, albeit a very persistent one." — *Einstein*

Take your **robust** model from Task 3. Can you perform a **Targeted Adversarial Attack**?

- Take an image of a 7.
- Optimize a noise pattern (perturbation) such that the model predicts a 3 with >90% confidence.
- **The Twist:** The perturbation must be invisible to the human eye (Constraint: Max pixel change epsilon < 0.05).
- **The Question:** Is your "robust" model (which ignores color) harder to fool than your "lazy" model from Task 1? Quantify the difference in the required noise magnitude.

Task 6: The Decomposition

Try training a Sparse Autoencoders (SAEs) to decompose the intermediate hidden states into an overcomplete representation and see if you are finding any meaningful features or decompositions. You may have to play around with some manual labelling. Explore!

Can you then make interventions by dialing up or down these intermediate features? Attempt this on the original model which learns color features instead of shapes, and try to see if color features are even present in the hidden states. This will again involve trial and error, and no concrete outcomes are expected. We are just interested to see what you discover.

NLP

The Ghost in the Machine

"Le style, c'est l'homme même" (The style is the man himself). — Georges-Louis Leclerc

Task 0: The Library of Babel

You will build a dataset where the primary variable is **authorship**, not topic.

1. **Class 1:** Download novels from two authors in Project Gutenberg (e.g., Dickens, Not Shakespeare, etc.). Make sure to clean up your data before using it anywhere.
2. **Topic Extraction:** Identify 5-10 core topics from the book (e.g., "The Ethics of Science," "The Loneliness of the Arctic").
3. **Class 2:** Use the Gemini API to generate 500 paragraphs (100-200 words each) on those same topics.
4. **Class 3:** Use the Gemini API to generate 500 paragraphs on those same topics, specifically prompted to mimic your chosen author's unique style.

Task 1: The Fingerprint

Before training, you must prove that these classes are mathematically distinct. Perform the following analyses:

1. **Lexical Richness:**
    - **Type-Token Ratio (TTR):** (Unique words / Total words).
    - **Hapax Legomena:** Count the number of words that appear only *once* in a 5,000-word sample. (Higher in humans, usually).
2. **Syntactic Complexity (SpaCy/NLTK):**
    - **POS Distribution:** Calculate the ratio of Adjectives to Nouns. Does the AI "over-describe" compared to the Human?
    - **Dependency Tree Depth:** Use SpaCy to calculate the average depth of the sentence parse trees. Longer, more nested branches indicate higher complexity.
3. **Punctuation Density:** Create a heatmap of punctuation usage (semicolons, em-dashes, exclamation marks).
4. **Readability Indices:** Calculate the Flesch-Kincaid Grade Level.

Task 2: The Multi-Tiered Detective

Build three detectors to separate AI generated text from Human written ones. **Note:** If your models fail to reach high accuracy, *this is still a valid research finding if you are able to do good analysis.* Document why.

- **Tier A (The Statistician):** An XGBoost/Random Forest model using *only* the numerical features from Task 1.
- **Tier B (The Semanticist):** A Feedforward NN using averaged pre-trained embeddings (GloVe/FastText).
- **Tier C (The Transformer):** Fine-tune a `distilbert-base-uncased` or `roberta-base` using LoRA.

Negative Results: If your models cannot distinguish between the Classes, find out why instead. (e.g., 50/50 accuracy).

Task 3: The Smoking Gun

We need to know *why* the model thinks a text is AI generated.

- **Saliency Mapping:** Use a library like `SHAP` or `Captum` to highlight the words in an "Imposter" paragraph that most strongly signaled "AI" to your Tier C model.
- **The Findings:** Does the model pick up on specific "AI-isms" (e.g., words like "tapestry," "delve," "testament") or is it looking at the rhythm of the sentence?
- **Error Analysis:** Find 3 samples where the Human was labeled as AI. Was the author being particularly repetitive? Was the AI being particularly brilliant?

Task 4: The Turing Test

1. **The Super-Imposter:** Can you "evolve" a paragraph that bypasses your best detector? You will implement a **Genetic Algorithm (GA)** to optimize a piece of AI-generated text until your classifier labels it as "Human."

   The GA Workflow:

   1. **Initial Population:** Generate 10 "Imposter" paragraphs using Gemini.
   2. **Fitness Function:** The "Human" probability score from your model.
   3. **Selection:** Keep the top 3 paragraphs that look "most human" to the model.
   4. **Mutation (LLM-as-Mutator):** For the next generation, prompt Gemini to "perturb" the winners:
      - *"Rewrite this paragraph to change the rhythm of the sentences while keeping the vocabulary."*
      - *"Introduce a subtle grammatical inconsistency or a rare archaic word."*
   5. **Iteration:** Run this for 5-10 generations.
   6. **The Goal:** Can you reach a >90% "Human" confidence score for a machine-written paragraph?
2. **The Personal Test:** Take your Statement of Purpose (SOP) or a recent essay you wrote. Run it through your detector.
   - If it says you are AI, why? Try to "humanize" your own writing manually to lower the AI score.

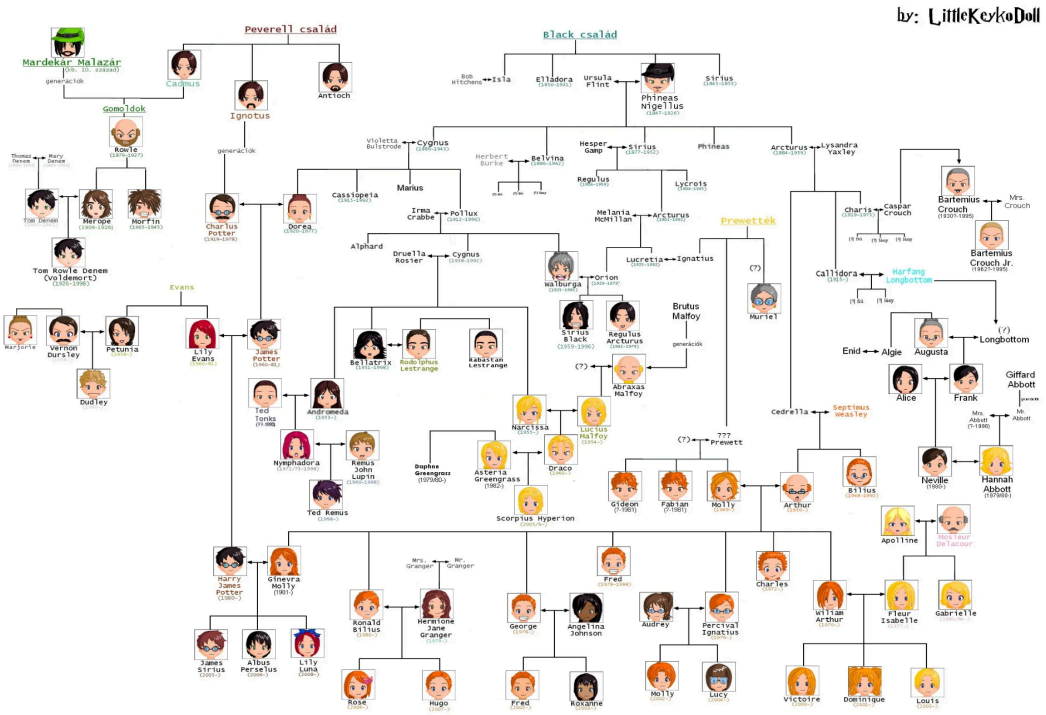
- If it says you are Human, try to rewrite a paragraph *manually* to sound like an LLM (overly helpful, structured, repetitive). Can you fool your own machine?

Graphs

Meta F-A-M-I-L-Y

“I solemnly swear that I am up to no good” ~ The Marauders

Graphs are all around us, and nearly everything can be represented as a graph. One particularly interesting application of graphs are family trees. Think of how huge a family can get even if you just account for a few generations of family, let aside all generations of it and graphs hold the power of representing this huge amount of data in a subtle, compact and beautiful way.



For this task, we'll be working with **MetaFam** - a synthetic family *Knowledge Graph (*KGs are cool, go look them up*)* containing many people connected through various family relationship types. To give a brief introduction, a KG is a structured representation of entities and their relationships, stored as (head, relation, tail) triples. For example, (Alice, motherOf, Bob) represents that Alice is the mother of Bob. It basically gives more textual meaning to a graph.

Go ahead and download the dataset from here.
Note: For all tasks, you are supposed to only use *train.txt,* unless stated otherwise.

Done? Now for the task…

Task 1: Dataset Exploration

“Makes a diff’rence, havin’ a decent family” ~Rubeus

Your first task is to load the dataset, explore its structure, and understand what you're working with. A few questions to get you started: How many people are there? How many relationship types? What's the distribution of different relationships?
(The more you work with your data, the more insights you get and more points as well 🙂)

1. Compute and report relevant statistics about the graph. Think about what metrics make sense for a family knowledge graph.
2. Create at least one meaningful visualization of the graph (or a subgraph, preferred). Make it informative (use gpt to make it pretty)

With this information, we now ask: *Can we use tools and concepts from Graph Theory to uncover interesting patterns in family structures, identify 'important' family members, and understand the hierarchical nature of family relationships*?

Some properties to explore include different types of centrality, density, diameter, clustering coefficient, and path lengths. Feel free to use properties that feel well connected to the task at hand. This task is focused on exploratory data analysis, and you are, at the very least, **expected to show plots/metrics to support your findings**. However, more emphasis will be placed on the **qualitative** insights that you come up with. Few tasks which are interesting are finding and defining *important* nodes in the graph, identifying generations in a family.

Task 2: Family Clusters

“We are only as strong as we are united, as weak as we are divided” ~Albus

Community detection or clustering is an important analysis for graphs. In the study of complex networks, a network is said to have community structure if the nodes of the network can be easily grouped into disjoint sets of nodes such that each set of nodes is densely connected internally, and sparsely between different communities.

1. Implement any two algorithms/ML methods for community detection on the graph (No need to do it from scratch).
2. Objectively assess the quality of your community detection through metrics like Modularity.
3. Analysis Questions:
    - Do the detected communities correspond to actual family units (e.g., nuclear families, extended families)?
    - How many generations typically exist within a single community?
    - Are there any "bridge" individuals who connect different family clusters?

4. In real genealogy, people often want to find their *closest* relatives. Based on your community analysis, propose a metric/method to rank how *related* two people are beyond just counting hops.

Task 3: Rule Mining

“Happiness can be found even in the darkest of times, if only one remembers to turn on the light” ~Albus

One of the most fascinating aspects of knowledge graphs is that they encode **logical rules** that can be discovered automatically. In a family graph, many relationships can be *inferred* from others through compositional reasoning. For example if (X, motherOf, Y) Λ (Y, fatherOf, Z) $\Rightarrow$ (X, grandmotherOf, Z).

1. Rule Discovery: Discover at least **5 logical rules** that hold in the MetaFam knowledge graph. These can be:
    - Horn clauses: relation1(X,Y) $\wedge$ relation2(Y,Z) $\rightarrow$ relation3(X,Z)
    - Inverse rules: relation1(X,Y) $\rightarrow$ relation2(Y,X)
    - More complex patterns involving 3+ relations

2. Validation: For each rule you discover:
    - Report its confidence (what fraction of the time does the rule hold?)
    - Report its support (how many instances of this rule exist in the data?)
    - Provide concrete examples from the dataset

Task 4: Link prediction!

“Things we lose have a way of coming back to us in the end, if not always in the way we expect” ~Luna

In real-world KGs, not all relationships are explicitly stated. Can we predict missing family relationships using machine learning? This is called KG Completion or Link Prediction - a fundamental task in graph machine learning!

Use the *train.txt* as your training data and *test.txt* as your test set to evaluate link prediction performance.

Train a **KG Embedding method** (TransE, DistMult, RotatE, CompleX, etc) and a **GNN based approach** (R-GCN, CompGCN, etc). Try to include both, but **don’t worry if you’re only able to do one of them**! Note that for GNN-based approaches, you'll need to make a scoring function for edge prediction. Python notebook with examples are provided in resource 🙂
Report standard link prediction metrics like MRR, Hits@1, Hits@10 on the test data. Plot the loss after training.

Resources:

1. Networkx python library
2. Community Detection
3. Graph machine learning
4. Knowledge Graph Embeddings: Simplistic and Powerful Representations.ipynb

5. Link prediction with RGCN

"Mischief Managed"

Quant

Congrats Congrats Money Money

"In this business it's easy to confuse luck with brains." - Jim Simons

Objective

Develop an end-to-end algorithmic trading pipeline for a universe of **N anonymized stocks**. Your goal is to transform raw price data into a trading system that maximizes risk-adjusted returns.

We are testing your ability to handle three critical stages of quantitative research: **Data Engineering**, **Strategy Formulation**, **and Simulation**.

General Instructions/Tips:

- There are FOUR TOTAL parts to this task.
    a. Feature Engineering & Data Cleaning
    b. Model Training & Strategy Formulation
    c. Backtesting & Performance Analysis
    d. Statistical Arbitrage Overlay
- We will expect at least one Python notebook as your deliverable for submission detailing every part of the three stage pipeline with comments and explanations of the approach taken.
- You may use Python scripts and multiple notebooks to modularize and clean up your code. We recommend this approach (ideally, three notebooks, supplementary scripts, and organized outputs).
- Do make visuals to illustrate your learnings and hypotheses wherever possible. It helps both you and us have a better time understanding it.
- Learn math. Nothing about this task is impossible to learn/implement with the knowledge a student from IIIT has acquired up until their 2nd year, first sem. It may seem daunting, but it's easier than it looks.
- This task is open-ended. Not everything will have a defined, objectively correct end goal. This is reflective of real quant research as well. Feel free to explore well beyond the scope of just this task's specifications.
- **KEEP LOOKING BACK AT THIS PAGE FOR POSSIBLE UPDATES**

Data

- **File:** daily_prices.csv, **T** years of OHLCV data for **N** tickers. (final file name may differ)
    - **https://www.kaggle.com/datasets/iamspace/precog-quant-task-2026**
- **Split:** The training, testing, and validations splits are entirely up to your creative process. However, keep in mind that we want to see your model perform out of sample for as long of a time frame as possible.
- **Note:** The years of data and number of stocks in the file that will be provided is open to change. This shouldn't affect how you approach the problem.

Part 1: Feature Engineering & Data Cleaning

Raw market data is messy and noisy. Show us how you handle it. Good data = good features, and good features = good model. (this prior statement has no mathematical backing and is not subject to any guarantees)

- **Cleaning:** Assess data quality. Handle missing values, outliers, or potential anomalies in the provided dataset.
- **Feature Extraction:** Generate features that capture market dynamics (e.g., momentum, volatility, volume patterns, or statistical factors). Get fancy with it. Think. What would actually be useful for a model to know?

Note: The quality of your inputs defines the ceiling of your performance. Simple raw prices are rarely sufficient for high-quality predictions. Too many features isn't a good thing either. Quant research is fast, so your models must be faster. *And remember, you are dealing with a universe here; not a single asset.*

Deliverable:

- A Python notebook showing your cleaning logic and feature generation code.

Part 2: Model Training & Strategy Formulation

Develop a predictive engine that translates your features from Part 1 into actionable trading signals for the universe.

- **Prediction:** Build a model (or models) to predict asset performance or rank attractiveness.
    - You have two broad choices for how you choose to define your prediction target:
        - Classification (whether the stock goes up, down, or stays the same)
        - Regression (predict, say, 1 day forward return, or 5 day forward return)
    - Regardless of what you choose, keep in mind that you will eventually have to convert this prediction into a viable, tradeable signal that your backtester is able to interpret.
- **Strategy Logic:** Define how these predictions translate into portfolio decisions.

Hint A: Financial data is incredibly noisy. Relying on a single signal source or a single naive model architecture can often lead to instability.

Hint B: Markets evolve. A relationship that held true in Year 1 may not exist in Year 3. Consider how your methodology ensures relevance as market dynamics shift over time.

Deliverable(s):

- A notebook detailing your modeling approach and the logic used to generate predictions.

Part 3: Backtesting & Performance Analysis

Simulate your strategy over the **2-Year Test Set**. Your backtester must realistically model the execution of your strategy.

- **Simulation Constraints:**
    - **Initial Capital:** $1,000,000. (this isn't necessary, you can use whatever)
    - **Transaction Costs:** Deduct **0.10% (10 bps)** per trade. (play around with it, we also want to see performance in absence of transaction costs)
    - **Universe:** You may trade any subset of the **N** stocks.
- **Metrics:** Report the following metrics (Note that all these metrics have multiple interpretations. Your choice of interpretation is also open to you, but you will be expected to justify your choice in the context of your pipeline):
    - **Sharpe Ratio** (annualized)
    - **Maximum Drawdown** (and average)
    - **Portfolio Turnover**
    - **Return** (how much money we made 💸)

Deliverable(s):

- The backtesting code. We want to see out-of-sample performance on at least two years worth of data.
- A cumulative PnL plot (Strategy vs. Equal-Weight Benchmark). How much did your strategy make over a benchmark? (either the market or an equal-weight buy-and-hold portfolio of the same assets)
- A brief analysis: Did the strategy survive transaction costs? When and why did it fail?

Notes:

- We will be judging you based on the above metrics, but we will also be judging the period over which those metrics were delivered. A lower Sharpe that remains stable over a long time frame is much more valuable than a ridiculously high Sharpe over one year. Always remember, you cannot see into the future.

Part 4: Statistical Arbitrage Overlay

While the main pipeline focuses on broad alpha, specific opportunities often exist in relative value.

- Oftentimes, there exist assets that exhibit correlated or *cointegrated* movement: they mode together. This movement may be instantaneous, or it may occur after a certain time lag.
- Your goal will be to find examples of such co-moving assets and explain the rationale behind their discovery.
- Do **not** stick to pure correlation. Use it as a baseline, but get creative with it. I won't provide any guidance or hints here as it would only serve to bias your research.
- Which assets seem correlated? Do they lie in the same sector? Is their customer base in the same country? Does sentiment play a role? Which timeframes do they appear to move together in? Which assets leads the relationship, and which asset lags in it?
- This is a very open ended question, and you may not even see statistically significant results. Do not be discouraged by this. It's all part of the quant research process. Make visuals, present ideas. We want it all.

Deliverable(s):

- **Analysis:** Present visuals and analysis of identified pairs or asset groups whose movements appear to be tethered together. Mathematically and empirically justify your approach and results.
- **Implementation Idea:** Demonstrate how you would incorporate this relative-value signal into your main portfolio structure. You don't need to code this up explicitly, but we would love to see mathematically backed ideas.

Something to consider before you dive in (feel free to skip)

- Quant research is messy. From start to finish, many things are abstracted away, and at times, the mathematical formulations used by researchers to try and make sense of things seem to only drive you further away from the truth. This is completely natural! Take it in stride as you proceed with this project.
- Don't be discouraged by the lack of results, and always be wary of results that seem too good.
- Remember that we cannot see into the future, and however elite your model is, it can't either.
- AI is your best friend that sometimes can't be trusted. Use it for your research, use it to learn, but always, always, validate what it's telling you.
- **Have fun with it!** Explore the data, make your visualizations, see how things move. An intuitive understanding only serves to supplement a purely mathematical one.
- The data is very naively anonymized. It's actually really easy to reverse engineer and figure out which asset's data it actually is. Feel free to try and come up with a mapping! It won't help you with this task regardless.

Doubts Document Link:

- https://docs.google.com/document/d/1ybowfluIkde2ggIqVEkyZ84t4IpuDXBiX0Wk5ZOZs3E/edit?usp=sharing