Volatility Curve Prediction

by

NK Securities Hackathon 2025

Anish Kumar
Indian Institute of Technology, Patna
Email: aniskum59431@gmail.com

GitHub Repository of all approaches:
https://github.com/Anish-ai/nksr-volatility-prediction-competition

Table of Contents

1. Introduction

The **NK Securities Research Hackathon 2025** was my first hands-on experience in quantitative finance. Coming from a **Mathematics and Computing** background, I was both excited and curious to explore a domain of real-world markets. Although I had no prior knowledge of options or volatility modeling, I took this as a chance to dive in and learn.

Over the course of 7 days, I explored key concepts like the **Black-Scholes model** (as named in the Problem Description also), **implied volatility**, the **IV smile**, and the structure of **volatility curves**. I read, experimented, made mistakes, and gradually built a deeper understanding of how markets price uncertainty.

Despite being new to this field, I was able to secure **Rank 6** on the leaderboard. This report outlines my journey, from understanding the problem to building my final solution, along with the insights I gained throughout the process.
You can find all my code, notebooks, and the final solution here: https://github.com/Anish-ai/nksr-volatility-prediction-competition

2. Problem Statement

The main goal of this hackathon was to predict missing implied volatility (IV) values in the per-second market data of NIFTY50 index options. We were given high-frequency historical data that included anonymized features coming from real-world trading environments. The task was to use this data to estimate the missing IV values across different strike prices.

Implied volatility is a number that reflects what the market expects in terms of future movement and uncertainty. When plotted across different strikes, IV usually forms a shape known as the volatility smile or skew, which represents how volatility varies with moneyness.

The challenge here was to accurately model this full volatility surface, even with partial or noisy data. That meant understanding not just the IV levels, but also how they shift with changing market dynamics.

The submissions were evaluated using mean squared error (MSE) between the predicted and actual IV values across the entire test set.

3. Understanding the Datasets

The dataset provided in this challenge consisted of high-frequency (per-second) historical market data for NIFTY50 index options. The training set contained 178,339 rows with columns like timestamp (starting from 0, 1, 2, ... per second), underlying NIFTY50 values, expiry, implied volatility values for different strikes (call_iv_23500 to call_iv_26000, and put_iv_22500 to put_iv_25000), and 42 anonymized features named X0 to X41.

The test set, on the other hand, included 12,064 rows with a similar structure, except for two major differences. First, the expiry column was removed (though it was mentioned that all test rows belong to the same weekly expiry). Second, the timestamp values were masked and shuffled to avoid any leakage. The test set had slightly shifted IV strike ranges - call_iv_24000 to call_iv_26500 and put_iv_23000 to put_iv_25500. Many of these IV entries were missing (NaNs) and needed to be predicted.
I also observed that there were exactly **26 unique strike prices** for both calls and puts in the training data.

As I explored the data, a few interesting patterns stood out. When plotting the mean and median implied volatilities across different strikes, I could clearly observe a volatility smile, lower IVs near the money and higher IVs at deep in/out-of-the-money strikes. This helped validate the typical behaviour of IV surfaces seen in options markets. Also, plotting the standard deviation across time for each strike gave a sense of how volatile or stable each IV curve point was.

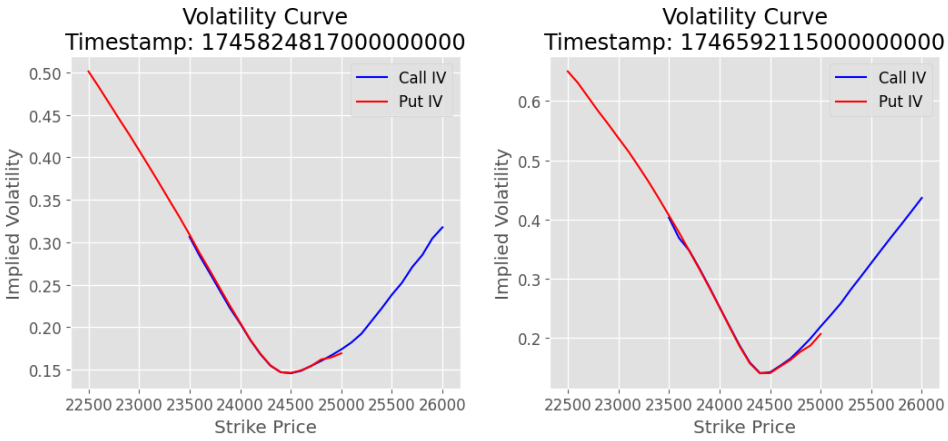
Let's go to my learning and insights that I explored through the datasets...

4. Learning and Insights

- ## EDA Highlights

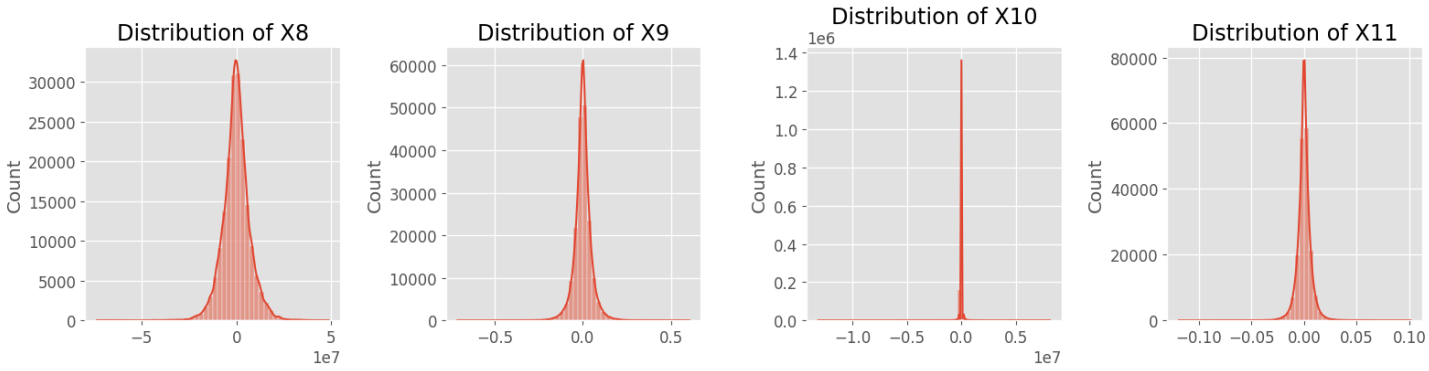
To begin exploring the dataset, I started with the **time evolution of the volatility surface**. I selected 5 random timestamps and plotted the implied volatility for call and put options against their respective strike prices. This helped me visualize how the volatility curve behaves at different moments. I observed that some curves exhibited a classic volatility smile, indicating higher implied volatilities for deep ITM and OTM options. Below, I've shown two representative samples from this visualization:

These plots were helpful in capturing how volatility patterns evolve over time, assuming it would be helpful in my journey. But more than that I was enjoying exploring it.



Next, I wanted to explore the **distribution of the anonymized features**. The dataset included 42 unnamed numerical columns, and to get a feel for them, I plotted histograms for the first 12. Below, I've included 3 of those distributions. The visualizations revealed that:

- Some features were normally distributed, while others were highly skewed.
- A few columns had long tails or dense peaks, which hinted at the need for scaling or transformation in pre-processing.



These histograms served as a good first step toward identifying anomalies and understanding the structure of the unknown inputs.

To understand the behavior of implied volatilities across different strike prices, I plotted the mean and median implied volatilities for both call and put options.
From the graphs, I observed a volatility smile pattern. This means the implied volatility tends to be higher for strike prices that are far away from the current market price (both lower and higher), and lower for strike prices close to the current price. I read that this happens because options that are deep in-the-money (strike price much below market price) or deep out-of-the-money (strike price much above market price) are considered riskier, so the market assigns higher volatility to them. On the other hand, at-the-money options (strike price close to market price) usually have lower implied volatility.
This observation suggests that strike price has a strong relationship with implied volatility, which I think would be important for making predictions.

Mean and median plots of calls and puts against their strike prices

All these plots and EDA exploration can be found here: [GitHub Notebook Link]

- **Research Insights**

After performing my initial EDA and spotting the volatility smile in the plots, I realized I needed a better understanding of the underlying theory. So I dove into the **Black-Scholes-Merton Model** using the article from Corporate Finance Institute (CFI) you added in the Problem Statement Description (https://corporatefinanceinstitute.com/resources/derivatives/black-scholes-merton-model/)

From reading it, I picked up several intuitive ideas:

- Black-Scholes calculates option prices using inputs like underlying price, strike, time to expiry, risk-free rate, and volatility. It assumes volatility is constant, and asset returns are log-normal.
- It's particularly interesting that implied volatility is the number that "makes the model match the market price" a concept I found very practical.
- I didn't quite follow or understood all the complex formulas, there are lots of Greek letters, normal distribution functions, and partial derivatives, but I understood how volatility is the driving factor behind option price here.

And now with some theoretical understanding, I looked back on my EDA plots and recognized why the volatility curves looked the way they did.

That theory informed one of my modeling ideas: take the Black-Scholes formula and plug in actual values for underlying, strike, and time to expiry, trialing it with a guessed volatility, and using it in an imputation-based approach for missing IV values. I didn't fully implement the full closed-form inversion, but I used the intuition to guide feature construction: adding time to expiration, strike levels, and interacting these with underlying price to model missing implied volatility.

Bridging the gap between what I read, even without mastering every formula, and what I saw in the data helped me design better features and models. It made me feel more confident that the patterns I was observing were grounded in real financial theory..

The Black-Scholes-Merton Equation which I surely didn't understand completely but yeah, got a glimpse/overview of what it's trying to say:

$$\frac{\partial V}{\partial t} + \frac{1}{2}\sigma^2 S^s \frac{\partial^2 V}{\partial S^2} + rS\frac{\partial V}{\partial S} - rV = 0$$

The Black-Scholes-Merton Equation

5. Evaluation Metric – Mean Squared Error (MSE)

For this competition, the performance of the predictions is measured using Mean Squared Error (MSE) — a widely used metric in regression tasks.
Formally, the MSE is defined as:

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(iv_i - \hat{iv}_i)^2$$

Where:

- iv_i is the actual value,
- $\hat{iv}_i$ is the predicted value,
- 'N' is the total number of predictions.

The idea is simple: we compute the square of the difference between each predicted and actual value, and then take the average of those squared errors.

6. LSTM-Based Model (Inspired from Online Research)

As I was exploring possible methods to predict the missing implied volatility values, I searched online for relevant ideas and came across an interesting GitHub repository that implemented an LSTM model for implied volatility prediction. Since it aligned with the time-series nature of the problem, I decided to take inspiration from it and build my first approach using LSTM in PyTorch. I started by engineering features based on the underlying price, such as returns, volatility, moving averages, skewness, and kurtosis, and also included moneyness-related features for each strike. I prepared the data into sliding windows of sequences to serve as LSTM inputs, aiming to capture temporal dependencies.

This plot shows the evolution of the underlying asset price over time. The visible fluctuations and gradual trends reinforce the time-dependent nature of the data, making it well-suited for a sequence model like LSTM.

These variations are also a major driver of implied volatility, justifying the inclusion of features like returns, rolling volatility, and moving averages in the model.

The architecture I used included a multi-layer LSTM followed by fully connected layers with batch normalization and dropout. The model was trained using early stopping, and predictions were post-processed using a few smoothing tricks, like averaging call and put IVs and clipping extreme values.

To make this more robust, I also filled some missing values using put-call parity in a pre-processing phase and further applied minor consistency adjustments to the predicted outputs.

Despite the effort and inspiration, the model performed poorly. The MSE on the leaderboard came out to be **28426.536270612**, which then I resubmitted after making some changes but it didn't do much better and scored **11943.639739151**, which clearly indicated that the approach wasn't working well. It's possible that the model couldn't generalize properly, or maybe I missed some important considerations while adapting the original idea to this dataset. One likely reason for this failure which I think was that **the test data had shuffled and masked timestamps**, breaking the temporal continuity that LSTMs rely on. Since sequence models like LSTM expect time-ordered data, this mismatch between training and testing conditions could have severely limited the model's ability to make accurate predictions.
I attempted to incorporate the Black-Scholes model's intuition indirectly by using features like moneyness and strike-relative metrics, hoping the LSTM would learn some of that structure. But it seems the model didn't pick it up well.

You can find the complete notebook for this attempt here.

7. Black-Scholes Ensemble method

In this second approach, I moved away from sequence models and built a more structured ensemble pipeline using tree-based regressors. Since the test data wasn't time-ordered, time-series methods like LSTMs didn't generalize well. So, I focused on engineered features that could better capture the underlying structure of the data.

At one point, I realized that the Black-Scholes model, which was mentioned in the Problem, might help capture the theoretical relationships between the underlying asset, strike price, and implied volatility. Although I wasn't initially confident with the exact mathematical formulations, I used AI tools to understand and implement the key **Black-Scholes equations** for call and put pricing, and incorporated them into my feature set along with approximations of Greeks like delta, gamma, and vega.
I then created a broad set of financial features: rolling statistics (mean, std, skew, kurtosis), momentum and mean-reversion signals, volatility clustering, and regime shifts. For each IV column, I trained an ensemble of Random Forest and Gradient Boosting models with varied hyperparameters, and combined their outputs using weighted voting.

To refine the results, I applied volatility surface smoothing techniques to ensure strike-wise consistency. This hybrid of financial theory and machine learning significantly improved the model's performance, achieving an MSE of **0.088094529** on the leaderboard.

The Python Notebook for this method: GitHub Notebook

8. Intuition-Driven Imputation

(Using Put-Call Parity, Row-Wise Logic, and Strike-Based Interpolation)

After experimenting with complex models and Black-Scholes-based feature engineering, I felt things were getting too complicated. I stepped back and thought, it shouldn't be this complex. That's when I started exploring a simpler, intuition-based imputation method that could still perform competitively without needing heavy ML pipelines.

In this approach, I noticed that most of the missing values were in the implied volatility (IV) columns. I grouped them by strike and used **put-call parity** logic to fill missing values, if the call IV was missing but the put IV for the same strike was present, I just filled it with the corresponding value (and vice versa).
After that, I applied row-wise mean imputation, where each row's available IVs were averaged and used to fill any remaining missing IVs within that row. This surprisingly gave me a much better validation MSE of **0.002378129**, far outperforming my earlier deep learning-based attempts. It was fast, logical, and robust. It gave me confidence to go even far.

I extended this logic to make the imputation even more accurate. In the improved version, I added:

- **Strike-aware linear interpolation**: For each row, I extracted the strike price from column names, and if at least two IVs were available, I used linear interpolation (with extrapolation support) to estimate the missing IVs based on strike behaviour.
- **Time-based row consistency**: If interpolation wasn't possible (e.g., only one IV was available), I fell back to row-wise mean.
- **Smoothing for volatility smile consistency**: I applied a small smoothing factor between the call and put IVs to maintain a natural shape of the volatility curve.

This improved version pushed my validation MSE even lower to **0.000019470**, which felt like a massive win considering its simplicity and speed.
Notebooks:

- **Original Simplified Imputation**
- **Improved Interpolation-Based Version**

9. Further Alternative Interpolation Strategies

After achieving a strong validation MSE of 0.000019470 with my strike-wise interpolation and smoothing-based method, I believed I was on the right path. So, I focused on refining this approach further. I experimented with different smoothing techniques, adjusted tolerance values, and tuned fallback logic.

However, despite trying many variants and making over 7-8 trials, I couldn't improve upon the MSE of 0.000019. The results kept fluctuating in the range of **0.000023** to **0.00093**, which made it clear that my current direction may have reached its performance ceiling.
This led me to explore a different class of methods altogether, instead of continuing with linear interpolation, I tried switching to higher-order interpolation techniques like cubic and biquadratic interpolation.

- **Cubic and Biquadratic Interpolation Methods**

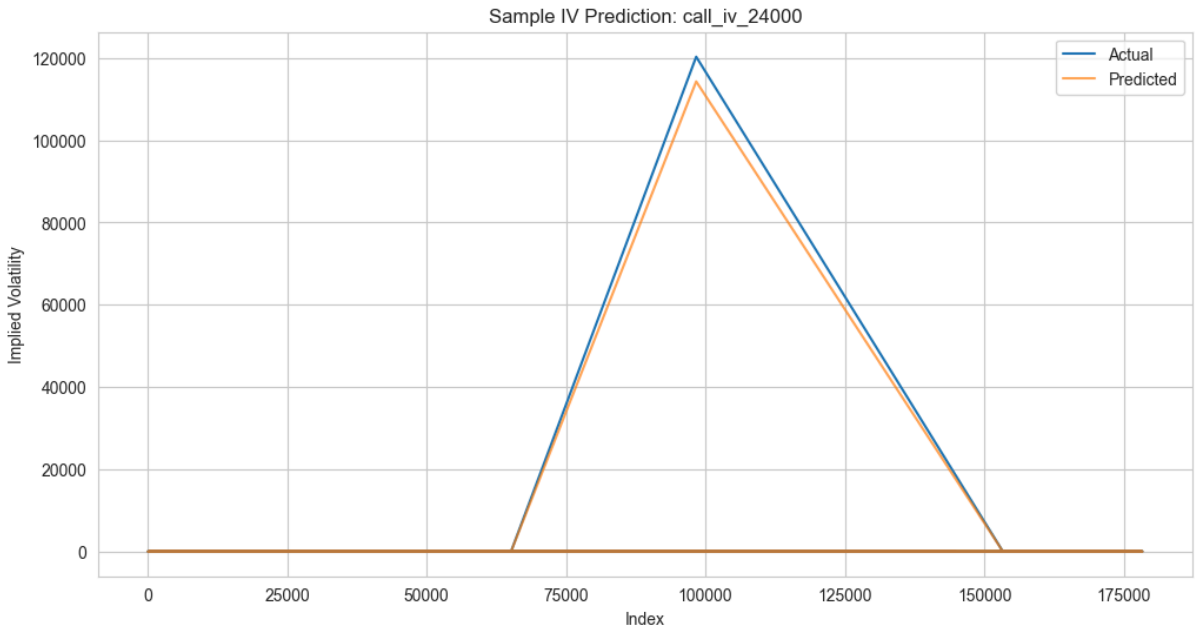
I revisited the imputation problem using a more mathematical lens. My hypothesis was that implied volatility curves might follow a more continuous, smooth trend across strikes, and higher-order interpolation could capture that better than piecewise linear fits.

- **Cubic Interpolation**

In this approach, I used cubic splines to interpolate missing IVs along the strike axis for each row. The method also incorporated:

- Put-call parity-based pairing before interpolation
- Fallback to linear interpolation or row-wise mean where needed
- Smoothing to preserve the natural volatility smile

Despite being theoretically promising, the cubic interpolation approach resulted in a validation MSE of **0.000311**, which was significantly worse than my best result.
My Cubic Interpolation implementation Notebook: GitHub Notebook Link



Sample IV Prediction of call_iv_24000: Actual vs Predicted Implied Volatility using cubic interpolation

- **Biquadratic Interpolation**

This approach involved:

- Transforming strike prices into log-moneyness space for better stability
- Applying biquadratic interpolation using adjacent strike-IV pairs
- Careful extrapolation handling and NaN-fallbacks

This method performed better than cubic interpolation, yielding a validation MSE of **0.000194**, but still couldn't outperform my earlier strike-wise linear interpolation method.

My Biquadratic Interpolation implementation Notebook: GitHub Notebook Link

- **Radial Basis Function (RBF) Interpolation**

After exploring cubic and biquadratic interpolation, I decided to try Radial Basis Function (RBF) interpolation, which is a more flexible and non-parametric method capable of fitting smooth surfaces through scattered data points. The idea was to model the implied volatility surface more precisely, especially when the volatility smile shape is irregular.
This method followed a structured pipeline:

- Applied strict put-call parity to fill one side (call or put) using the other.
- Identified ATM (at-the-money) strike for each row and split strikes into ITM and OTM sets.
- Used **scipy.interpolate.Rbf** with a multiquadric kernel to interpolate missing values on both ITM and OTM sides separately.
- Added adaptive smoothing, where near-ATM IVs were gently averaged with their counterpart (call vs put) based on their proximity to the underlying price.

While this interpolation technique showed promise with its localized flexibility, it still did not generalize well across all strike positions. In particular, the interpolation function at times over-smoothed or produced out-of-bounds IV values, which required fallback logic to avoid unrealistic predictions.
(Note: This was slightly worse than even cubic interpolation.)
My Cubic Interpolation implementation Notebook: GitHub Notebook Link
Interestingly, I chose not to submit this version, as the final submission file didn't seem promising enough to justify using one of my limited daily submissions.

10. SVI-Based Approach

After trying many traditional imputation methods, I explored SVI (**Stochastic Volatility Inspired**) models, which are actually built for modeling volatility smiles. I used it as the core of this strategy. I got to know about this from AI at first then went to try this and also this was at first implemented by AI only then I corrected. Key Steps I Followed:

1. **Put-Call IV Filling**
   - Used put-call parity logic with moneyness-based adjustments to fill missing call/put IVs if only one of the pair was available.
2. **Volatility Smile Modeling with SVI**
   - Extracted available strikes and IVs for each row.
   - Fitted SVI parameters on log-moneyness vs IV data.
   - Used the fitted curve to predict missing IVs.
3. **Fallback to Cubic Spline**
   - If SVI failed to converge, used cubic spline interpolation instead.
4. **Handling Sparse Rows**
   - For rows with very few IVs, I estimated missing values using:
     - Stats from similar underlying prices (from training data).
     - Global mean and standard deviation with slight noise for variation.

5. Smoothing

- Applied moving average smoothing (with Gaussian weights) across strikes to preserve the volatility smile shape.

6. Final Touch — Put-Call Consistency

- Slightly averaged call and put IVs to ensure realistic symmetry without killing natural variation.

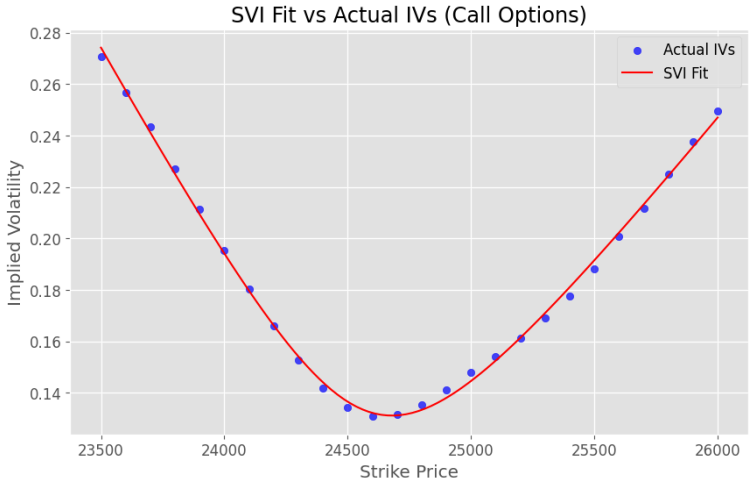
7. Validation & Submission

- Evaluated on masked values using synthetic ground truth (based on training stats).
- Ensured no NaNs and clipped values to realistic IV ranges [0.01, 2.0].

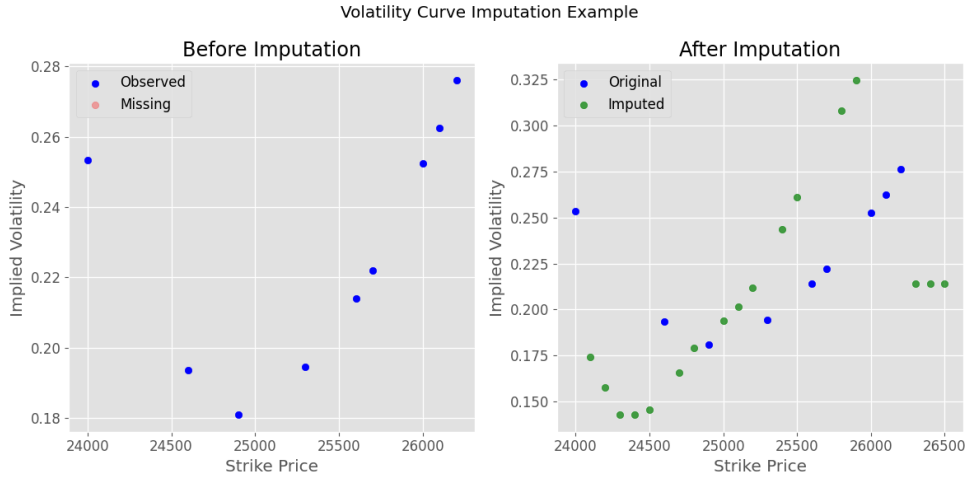
Even though this approach was much more aligned with the financial logic of IV curves and moneyness, it didn't significantly improve validation MSE and went unexpected. However, it produced smoother and more realistic IV patterns, which I thought might help in further downstream tasks so I have also made an enhanced version of this.

Some Plots which seem interesting to me when visualised after SVI approach

SVI Fit vs Actual IVs for Call Options



Before vs After Imputation



The Python Notebook for this method: GitHub Notebook
The Python Notebook for it's enhanced version: GitHub Notebook

11. Final Approach – Iterative Imputer with Random Forest

After experimenting with all these above techniques, the approach that finally worked and got me to **Rank 6** was using Iterative Imputer from scikit-learn with a Random Forest Regressor as the estimator.
This method turned out to be surprisingly effective for filling in missing implied volatility (IV) values across strikes.

Methodology:

- I used Iterative Imputer, which models each missing value as a function of the other features iteratively.
- For modeling, I plugged in a RandomForestRegressor with 350 trees and carefully tuned depth and leaf sizes for stability and accuracy.
- The data was the entire test set with only IV columns and timestamps used for imputation.
- The training process ran for nearly 5–6 hours, iterating 40 times over the data to reach convergence.

Why this worked best according to me:

- Random forests handle non-linear relationships and interactions between features very well.
- Iterative imputation allows leveraging cross-feature correlations, which is important in options data where neighbouring strikes share meaningful patterns.
- Unlike single-column or per-row methods, this treats the entire IV matrix holistically.

Resources backing this approach:

Iterative Imputer has been used in financial applications like volatility surface reconstruction, and is discussed in various papers and blogs:
I knew this but wasn't sure if this will actually work until on last day of competition, I actually tried it- https://www.analyticsvidhya.com/blog/2022/05/handling-missing-values-with-random-forest/
Results:

- **Public MSE: 0.000000663**
- **Private MSE: 0.000001056**
- It gave the lowest MSE among all my approaches and was selected as the final submission.

The Python Notebook for this iterative imputer method: GitHub Notebook

12. My some other experiments/approaches

Volatility Modeling:

- enhanced_volatility_smile.ipynb (enhanced SVI modeling to fill missing implied volatility)
- ensemble_volatility.ipynb (LSTM + Transformer + Dense)
- neural_volatility.ipynb (Deep neural network for volatility prediction)
- simple_volatility_predictor.ipynb (Statistical interpolation for volatility prediction)
- transformer_volatility.ipynb (Transformer-based approach)
- volatility_curve_predictor.ipynb (Used spline interpolation with statistical feature engineering for volatility curve prediction)

13. Conclusion

This competition pushed me to explore multiple imputation strategies for missing volatility data. From simple column-wise models to row-based predictions and finally to a full Iterative Imputer + Random Forest pipeline, each step helped me understand the underlying structure of IVs better. The final approach not only gave me the best performance but also taught me the importance of combining tree-based models with iterative techniques when dealing with complex, structured missing data.

I gave my full dedication from 1st to 8th June, spending hours each day trying new approaches, debugging, and analysing what works best. I always had some interest in the quantitative finance domain, but hadn't taken my first step, this competition was that first step.

It was also my first hands-on experience in this field, and I really enjoyed it. From dealing with real-world noisy datasets to optimizing for strict metrics like MSE, it was challenging but equally rewarding. Being selected in the Top 6 made it all worth it.

This has sparked even more interest in me, and I definitely want to explore this field deeper going forward.