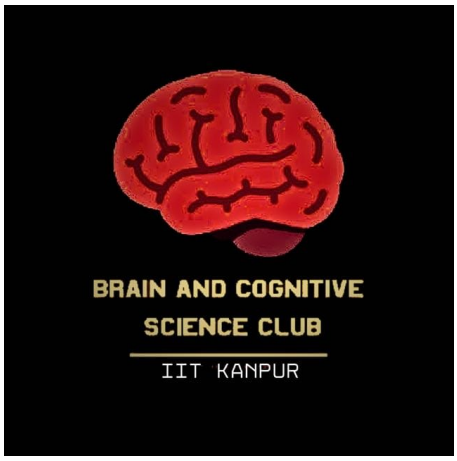

Freshers Amateur Project - Brain and Cognitive Science Club

Gradient Gains

Mentors

Aaryan Agarwal, Atharv Dubey, Rishitesh

Contents

1 Project Introduction

1.1 Problem Statement

The objective of this project is to build a practical forecasting system that predicts the **next closing price** of an asset using historical daily market data. The solution is deployed as an interactive Streamlit application so that users can select a ticker, run the model, and view both forecast and model-quality metrics in real time.

1.2 Project Goals

The main goals are:

- Support multiple asset classes (US equities, Indian equities, and crypto) from one interface.
- Build a robust short-horizon predictive pipeline using engineered time-series features.
- Report transparent evaluation metrics (RMSE and MAPE) on holdout data.
- Deliver an end-to-end user-facing product with clear visualization and forecast output.

1.3 Models Tried (From Comparative Experiments)

Before settling on the final ensemble strategy, multiple model families were tested:

- **Linear Regression**: baseline model, fastest but consistently highest error.
- **Feed-Forward Neural Network**: improved over linear baseline, but less stable across tickers.
- **RNN/LSTM**: better sequence modeling than plain neural nets, still weaker than tree ensembles in this setup.
- **XGBoost**: strong standalone tree-boosting performance.
- **Random Forest**: strong and stable nonlinear baseline.
- **XGBoost + Random Forest Ensemble**: best overall MAPE in the comparison snapshot.

In the current deployed code (`code_fin (1).py`), the ensemble is implemented as `xgb.XGBRegressor` + `RandomForestRegressor` inside a weighted `VotingRegressor`, preserving the same design idea: combine strong tree-based learners.

STOCK NAME	Linear Reg.	Neural Net	RNN/LSTM	XGBoost	Random Forest	XGB + RF (Final)
HDFCBANK.NS	9.235%	2.257%	2.511%	0.751%	0.871%	0.691%
BHARTIARTL.NS	8.624%	3.670%	3.296%	0.761%	0.834%	0.706%
AAPL	8.750%	2.448%	4.459%	1.041%	1.227%	0.926%
GOOGL	6.089%	4.766%	2.690%	1.234%	1.673%	1.094%
RELIANCE.NS	5.216%	5.487%	4.382%	1.370%	1.479%	1.164%
TCS.NS	5.912%	5.583%	3.452%	1.600%	1.852%	1.347%
AMZN	9.038%	4.951%	3.242%	1.681%	2.202%	1.439%
TATASTEEL.NS	6.374%	3.772%	3.578%	1.646%	2.154%	1.486%
NVDA	8.632%	4.416%	3.175%	1.957%	2.154%	1.519%
MSFT	8.436%	5.039%	3.052%	2.309%	2.214%	1.793%
TSLA	8.607%	5.788%	3.649%	2.024%	2.218%	1.861%
BTC-USD	12.023%	10.236%	7.090%	3.162%	4.359%	2.838%

Figure 1: Cross-model MAPE comparison used during model selection.

2 Core Project Workflow (Main Implementation)

2.1 Data Collection

Data is collected using `yfinance` with daily frequency (`interval = "1d"`) for roughly the last three years per selected ticker.

- Start date: `today - 3*365 days`
- End date: current day
- Fields used: Date, Open, High, Low, Close, Volume

The app also supports quick ticker selection through sidebar presets for:

- US giants: AAPL, MSFT, GOOGL, AMZN, META, NFLX
- US chips and auto: NVDA, AMD, TSLA
- Indian market (NSE suffix): RELIANCE.NS, TCS.NS, INFY.NS, HDFCBANK.NS, ICICIBANK.NS, SBIN.NS, TATAMOTORS.NS, ADANIENT.NS, BHARTIARTL.NS
- Crypto: BTC-USD, ETH-USD, SOL-USD

2.2 Data Preprocessing

The preprocessing steps in `code_fin (1).py` are:

- Flatten `MultiIndex` columns (when returned by `yfinance`).
- Reset index so date becomes an explicit column.
- Standardize all column names to lowercase.
- Use a working copy (`work_df`) to avoid fragmentation and side effects.

2.3 EDA and Data Understanding

The project includes practical EDA through the application itself:

- Time-windowed price history (1M, 3M, 6M, 1Y, 5Y) for trend inspection.
- Two-row visualization: price with EMA overlays plus RSI(14) panel.
- Historical line plot vs forecast line for immediate visual sanity checks.
- Data table output for predicted prices and business-day dates.

This EDA layer helps verify trend continuity, volatility regimes, and short-horizon forecast behavior before using model outputs.

2.4 Feature Engineering

2.4.1 Lag Features

For each of `open`, `high`, `low`, `close`, and `volume`, lag features are generated from lag 0 to lag 20:

$$x_{t-i}, \quad i \in \{0, 1, \ldots, 20\}.$$

2.4.2 Derived Features

The engineered features include:

- **Log-return features**: $\log(\text{close}_t/\text{close}_{t-i})$ for $i \in \{1, 5, 10, 20\}$.
- **Relative moving-average ratios**: $\text{close}_t/\text{MA}_i$ for $i \in \{5, 10, 20\}$.
- **Short-term volatility proxy**: standard deviation over recent close lags (`vol5`).

2.5 Target Construction

The pipeline predicts one-step-ahead log return:

$$y_t = \log\left(\frac{P_{t+1}}{P_t}\right),$$

where P_t = `close_lag_0` and P_{t+1} is generated using a one-step forward shift. Predicted price is recovered as:

$$\hat{P}_{t+1} = P_t \cdot e^{\hat{y}_t}.$$

2.6 Train/Test Processing

After dropping NaN rows induced by lagging and shifting:

- Last 60 observations are reserved as test set.
- Earlier observations are used as training set.

This gives a realistic recent holdout window for validation of short-term forecasting quality.

2.7 Model Architecture

The deployed model in `code_fin (1).py` is a weighted voting ensemble:

- `XGBRegressor` ($n\_estimators = 300$, $learning\_rate = 0.03$, $max\_depth = 5$, $subsample = 0.8$, $colsample\_bytree = 0.8$)
- `RandomForestRegressor` ($n\_estimators = 400$, $max\_depth = 10$, $min\_samples\_leaf = 5$)
- `VotingRegressor` weights: $[2, 1]$

2.8 Evaluation Metrics

Model quality is measured on holdout data:

- **RMSE**

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

- **MAPE**

$$\text{MAPE} = \frac{100}{n}\sum_{i=1}^{n}\left|\frac{y_i - \hat{y}_i}{y_i}\right|$$

In this project, y_i and $\hat{y}_i$ above are computed on actual and predicted **next-close prices**.

2.9 Recursive Forecasting Logic

For a 3-day horizon, prediction is generated recursively:

- Predict day $t + 1$ from latest available row.
- Append predicted value as synthetic OHLC row.
- Recompute features and predict next step.
- Skip weekends while generating future dates.

2.10 Trading Signal Layer (RSI)

The app computes EMA(50), EMA(200), and RSI(14) from close prices and displays a simple signal:

- BUY if RSI < 40
- SELL if RSI > 60
- HOLD otherwise

3 Results and Comparative Performance

3.1 Model Comparison Insights

The model-comparison image shows a clear pattern:

- Linear and basic neural baselines have significantly higher MAPE.
- Tree-based methods (XGBoost / Random Forest) reduce error strongly.
- Ensemble setup gives the best and most consistent final MAPE.

3.2 Final Ranking Snapshot (XGB + RF)

Table 1: Final MAPE ranking from the comparison snapshot.

Stock	**MAPE (%)**
HDFCBANK.NS	0.691
BHARTIARTL.NS	0.706
AAPL	0.926
GOOGL	1.094
RELIANCE.NS	1.164
TCS.NS	1.347
AMZN	1.439
TATASTEEL.NS	1.486
NVDA	1.519
MSFT	1.793
TSLA	1.861
BTC-USD	2.838

4 Final Product Output (Streamlit)

4.1 Dashboard View

The final interface combines configuration, metric cards, and forecast visualization in one page:

- Sidebar controls for ticker and timeframe
- Performance block: `Model Performance (Backtest)` with RMSE, MAPE, Train Data, and Test Data
- Price block: `Price Prediction` with current price, next-close prediction, and RSI signal
- Historical vs 3-day forecast line chart with EMA overlays
- RSI subplot with 40/60 guide lines
- Currency-aware display ($ for global tickers, for `.NS` tickers)

Figure 2: Main Streamlit dashboard output.

4.2 Forecast Plot Example

Figure 3: Focused view of historical trend with 3-day dotted forecast.

4.3 Forecast Table Output

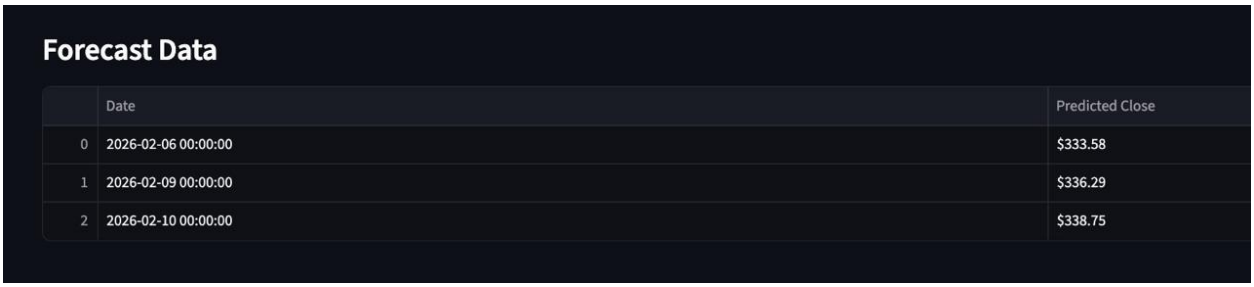

Figure 4: Forecasted close values with generated future business dates.

5 Limitations and Future Enhancements

5.1 Current Limitations

- Fixed 60-day holdout split; not a full walk-forward backtest.
- Recursive multi-step forecasting can accumulate error.
- No macroeconomic or sentiment signals are included in current features.

5.2 Possible Improvements

- Include exogenous variables (indices, rates, sector signals, news sentiment).
- Ensemble using LSTMs, XgBoost can also be tried.
- Introduction of model explainability using SHAP score.

6 Conclusion

This project delivers a complete ML product pipeline: **data collection → feature engineering → training and validation → multi-step forecasting → interactive deployment**. Comparative experiments indicate that ensemble tree methods are most reliable for this task, and the Streamlit application converts the modeling pipeline into an accessible decision-support tool.

Appendix: Run Instructions

```
pip install streamlit yfinance pandas numpy scikit-learn plotly xgboost
streamlit run "code_fin (1).py"
```