NIFTY Market Data Engine

API Documentation

Version 3.0

Data Pipeline Team
Mathematical Finance Group Project

January 2026

Purpose. This document is the complete technical reference for the `NiftyMarketData` Python class. It is intended for all teams that consume options market data: Pricing (Team 2b), Hedging, and Volatility Surface. Data producers on the Data Pipeline Team should also consult this document to ensure new datasets follow the required schema.

Contents

Overview

The NIFTY Market Data Engine provides a clean, professional query interface over historical NIFTY 50 European options data. Raw data is stored as thousands of CSV files across monthly folders on a shared Google Drive. This engine abstracts that entirely: downstream teams write a single Python query and receive a fully standardised, spot-merged `pandas` DataFrame.

Why This Engine is Needed

Pricing, Hedging, and Volatility Surface teams require market data as direct input to:

- Black–Scholes theoretical pricing
- Implied volatility computation $\sigma(K, T)$
- Greeks computation: Δ, Γ, $\mathcal{V}$, Θ
- Volatility surface construction $\sigma(K, T)$

Without this engine, each team would need to locate files manually, handle timestamp reconstruction, and merge spot data themselves — a slow, error-prone process. The engine encapsulates all of that.

Multi-Year Support

Version 3.0 extends the engine to support multiple calendar years (2024, 2025, 2026 and beyond). The correct year folder is determined automatically from the trade date in each query.

Dataset Structure

Folder Layout

All data resides under a single root directory (`base_dir`), which is the shared Google Drive folder. The internal structure must follow this layout exactly:

```
base_dir/
    2024/
        2024JAN/          <- option CSV files for January 2024
        2024FEB/
        ...
        2024DEC/
        2024Nifty/        <- spot index CSV files
    2025/
        2025JAN/
        ...
        2025Nifty/
    2026/
        ...
```

Option File Naming Convention

Each option file follows the pattern:

```
NIFTY-{EXPIRY}-{TRADEDATE}.csv
```

Example: `NIFTY-01FEB24-01JAN24.csv`

Component	Description
`EXPIRY`	Expiry date in `DDMMMYY` format, e.g. `01FEB24`
`TRADEDATE`	The date trading occurred, e.g. `01JAN24`

Raw File Columns

Each option CSV contains intraday one-minute OHLC bars:

Column	Description
`datetime`	Intraday time (HH:MM:SS)
`strike_price`	Strike price (integer)
`right`	Option type: `CE` (Call) or `PE` (Put)
`open`	Opening price of 1-minute bar
`high`	High price
`low`	Low price
`close`	Closing price
`volume`	Contracts traded in this minute
`open_interest`	Open interest

Spot File Naming Convention

```
Nifty-{YEAR}{MONTH}.csv
```

Example: `Nifty-2024JAN.csv`

Spot files contain columns `datetime` and `close`, where `close` is the NIFTY 50 spot index level.

Key Assumptions

All consumers of this API must be aware of the following assumptions that affect data quality and pricing calculations.

1. **No Bid/Ask Data.** This dataset does not contain bid or ask quotes. Option market price is proxied by the close price of each one-minute bar:

$$P_{\text{market}} = P_{\text{close}}$$

2. **Zero-Volume Rows.** A large fraction of rows have `volume` $= 0$. These represent stale quotes where no trade occurred in that minute. For implied volatility calculations, apply `min_volume` $\geq$ `10` to ensure only actively traded contracts are used.

3. **Spot Price Matching.** Spot prices are merged using nearest-timestamp matching (one-minute resolution):

$$S_t \approx S_{\hat{t}}, \quad \hat{t} = \arg\min_{\tau} |t - \tau|$$

4. **Trading Hours.** NIFTY trades from 09:15 to 15:30 IST. Time window queries outside these hours will return empty results.
5. **European Options.** All NIFTY index options are European-style. This is assumed in all downstream pricing models.

Installation and Setup

Requirements

```
pip install pandas
```

Python 3.8 or higher is required.

File Layout

Place the engine file inside an `api/` folder at your project root:

```
your_project/
    api/
        __init__.py        <- empty file, required
        marketdatav2.py    <- the engine
    your_notebook.ipynb
```

The `__init__.py` file may be empty; it is required to make `api` a Python package.

Import and Initialise

```python
from api.marketdatav2 import NiftyMarketData

BASE_DIR = r"G:/SharedDrive/NiftyHistorical"
md = NiftyMarketData(base_dir=BASE_DIR)
```

Date Format Reference

Parameter	Format	Examples
`expiry`	`DDMMMYY`	`01FEB24`, `27MAR25`
`trade_date`	`DDMMMYY`	`01JAN24`, `15JAN25`
`start`, `end`	`YYYY-MM-DD HH:MM`	`2024-01-01 10:00`
`timestamp`	`YYYY-MM-DD HH:MM`	`2024-01-01 10:00`
`snapshot_time`	`HH:MM`	`10:00`, `09:30`

Complete API Reference

`query_options()` — Core Query

The primary method for all option data access. All filters are optional; omitting them returns the full unfiltered dataset for that expiry and date.

```
df = md.query_options(
    expiry="01FEB24",             # required
    trade_date="01JAN24",         # required
    strikes=[21500, 21700],       # optional: list of int
    option_type="C",              # optional: "C" or "P"
    start="2024-01-01 10:00",     # optional
    end="2024-01-01 11:00",       # optional
    min_volume=10,                # optional: int >= 0
    raise_if_empty=False          # optional: bool
)
```

Output columns:

Column	Description
`timestamp`	Full datetime of the record
`expiry_date`	Expiry date (Python `date` object)
`days_to_expiry`	Calendar days until expiry from trade date
`strike`	Strike price (integer)
`option_type`	`"C"` (Call) or `"P"` (Put)
`open_price`	Open price of the 1-minute bar
`high_price`	High price
`low_price`	Low price
`close_price`	Close price
`market_price`	Pricing proxy $= P_{\text{close}}$
`volume`	Contracts traded in that minute
`open_interest`	Open interest
`spot_price`	NIFTY 50 spot level (auto-merged)

`list_expiries()` — Discovery

```
expiries = md.list_expiries(trade_date="01JAN24")
# Returns: ['01FEB24', '04JAN24', '11JAN24', '25JAN24', ...]
```

Returns all expiry files that exist on disk for the given trade date. Use before `query_options()` to confirm an expiry is available.

`list_strikes()` — Discovery

```
strikes = md.list_strikes(expiry="01FEB24",
   trade_date="01JAN24")
# Returns: [19850, 20000, 21000, 21100, ..., 23500]
```

Returns all strike prices that appear in a given expiry/date file.

`list_trading_days()` — Discovery

```python
days = md.list_trading_days(year=2024, month="JAN")
# Returns: [’01JAN24’, ’02JAN24’, ’03JAN24’, ...]
```

Returns all trading days for which option files exist in a given month.

`get_atm_strikes()` — ATM Grid

Generates a symmetric grid of strikes centred on the at-the-money level.

$$K_{\text{ATM}} = \left\lfloor \frac{S_0}{\Delta} \right\rceil \cdot \Delta$$

where S_0 is the opening spot price and Δ is the step size.

```python
atm, grid = md.get_atm_strikes(
    expiry="01FEB24",
    trade_date="01JAN24",
    n_strikes=10,    # strikes each side of ATM
    step=100         # spacing between strikes
)
print(atm)     # 21700
print(grid)    # [21200, 21300, ..., 21700, ..., 22200]
```

Total grid size $= 2 \times$ `n_strikes` $+ 1$.

`surface_snapshot()` — Volatility Surface

The primary deliverable for Team 2b. Constructs the complete (K, T) input grid required for implied volatility surface fitting at a single point in time.

```python
surface = md.surface_snapshot(
    trade_date="01JAN24",
    timestamp="2024-01-01 10:00",
    n_expiries=8,        # max expiries to include
    n_strikes=10,        # strikes each side of ATM per expiry
    step=100,
    option_type="C",     # "C", "P", or None
    min_volume=0
)
```

Use for Black–Scholes IV fitting:

```python
S     = surface["spot_price"]
K     = surface["strike"]
T     = surface["days_to_expiry"] / 365.0   # years
C_mkt = surface["market_price"]

# Solve: C_BS(S, K, T, r, sigma) = C_mkt  for sigma
```

`query_time_series()` — Multi-Day Evolution

Queries the same expiry across multiple trade dates. Useful for studying time decay and rolling IV across a trading week or month.

```
df_ts = md.query_time_series(
    expiry="01FEB24",
    trade_dates=["01JAN24", "02JAN24", "03JAN24"],
    strikes=[21700],
    option_type="C",
    snapshot_time="10:00",   # optional: fix the intraday time
    min_volume=0
)
```

The output contains an additional `trade_date` column prepended.

`clear_spot_cache()` and `cache_status()`

```
# Inspect what is currently cached
print(md.cache_status())
# {’2024JAN’: 375, ’2024FEB’: 391}

# Clear cache (free memory or after dataset update)
md.clear_spot_cache()
```

Error Handling

The engine raises typed exceptions with descriptive messages:

Exception	When Raised
`FileNotAvailable`	Required CSV file not found on disk
`NoDataReturned`	Query matched 0 rows (`raise_if_empty=True`)
`InvalidParameter`	Bad value for `option_type`, dates, or `strikes`
`MarketDataError`	Base class; all engine errors inherit from this

Example:

```
from api.marketdatav2 import FileNotAvailable,
   InvalidParameter, NoDataReturned

try:
    df = md.query_options(expiry="01FEB24",
       trade_date="01JAN24",
                            min_volume=999999,
                               raise_if_empty=True)
except NoDataReturned as e:
    print(e)   # prints actionable guidance
```

Typical Workflow: Pricing Team

The following is the recommended end-to-end workflow for Team 2b:

```
from api.marketdatav2 import NiftyMarketData

md = NiftyMarketData(base_dir=BASE_DIR)

# Step 1     Confirm available expiries
expiries = md.list_expiries("01JAN24")

# Step 2     Get surface snapshot
surface = md.surface_snapshot(
    trade_date="01JAN24",
    timestamp="2024-01-01 10:00",
    n_expiries=8,
    n_strikes=10,
    option_type="C",
    min_volume=0
)

# Step 3     Extract inputs
S     = surface["spot_price"].values
K     = surface["strike"].values
T     = surface["days_to_expiry"].values / 365.0
C_mkt = surface["market_price"].values

# Step 4     Implied volatility solve
# sigma = your_iv_solver(S, K, T, r, C_mkt, "C")
```

Summary of All Methods

Method	Purpose
`query_options(...)`	Core data access with all filters
`list_expiries(trade_date)`	Discover expiries on a date
`list_strikes(expiry, date)`	Discover strikes in a file
`list_trading_days(y, m)`	Discover trading days in a month
`get_atm_strikes(...)`	Build ATM-centred strike grid
`query_time_series(...)`	Multi-day evolution query
`surface_snapshot(...)`	Full vol-surface input grid
`clear_spot_cache()`	Free memory / reset cache
`cache_status()`	Inspect cached months

Contact and Maintenance

This engine is maintained by the **Data Pipeline Team**. For the following situations, contact the Data Pipeline Team rather than modifying files directly:

- A `FileNotAvailable` error for a date that should have data.
- New data for 2025 or 2026 not yet loaded.
- Schema changes in the raw CSV format.
- Questions about risk-free rate or dividend yield integration.

Do not modify the raw CSV files or folder structure. The engine depends on the naming conventions being exact.

Data Pipeline Team · Mathematical Finance Group Project · Version 3.0 · January 2026