

# QuantStats RELIANCE.NS Tearsheet

I want to get the quantstats tearsheet of RELIANCE.NS

To generate a QuantStats tear sheet for **RELIANCE.NS** (Reliance Industries Limited), you can follow these steps using Python. QuantStats is a powerful Python library that provides comprehensive portfolio analytics and risk metrics, enabling you to create detailed performance reports.

## Step-by-Step Guide

### 1. Install Required Libraries

Ensure you have the necessary Python libraries installed:

```
bash

pip install quantstats yfinance
```

### 2. Import Libraries and Download Data

Use `yfinance` to fetch historical stock data for RELIANCE.NS:

```
python

import quantstats as qs
import yfinance as yf
import pandas as pd

# Download historical data for RELIANCE.NS
data = yf.download('RELIANCE.NS', start='2018-01-01', end='2025-05-06')

# Calculate daily returns
returns = data['Adj Close'].pct_change().dropna()
```

### 3. Generate the Tear Sheet

Create a comprehensive HTML tear sheet:

```
python

# Generate a full tear sheet
qs.reports.html(returns, output='reliance_tearsheet.html', title='Reliance
```

This will produce an HTML file named `reliance_tearsheet.html` containing various performance metrics, risk analyses, and visualizations.



## Key Features of the Tear Sheet

The generated report includes:

- **Performance Metrics:** CAGR, Sharpe Ratio, Sortino Ratio, Calmar Ratio
- **Risk Metrics:** Volatility, Value at Risk (VaR), Conditional VaR (CVaR), Skewness, Kurtosis
- **Drawdown Analysis:** Maximum Drawdown, Average Drawdown, Drawdown Duration
- **Visualizations:** Cumulative Returns, Monthly Returns Heatmap, Rolling Statistics

These insights help in understanding the stock's performance and risk profile over the selected period.



## Additional Resources

- **QuantStats GitHub Repository:** Provides detailed documentation and examples.
- **Tutorial Video:** A comprehensive guide on using QuantStats for portfolio analysis.

By following these steps, you can effectively analyze RELIANCE.NS's performance using QuantStats.



```
-----  
AttributeError                                Traceback (most recent call  
last)  
Cell In[1], line 1  
----> 1 import quantstats as qs  
      2 import yfinance as yf  
      3 import pandas as pd  
  
File d:\Quantflow\QuantFlow v1.0\Day19\venv\Lib\site-  
packages\quantstats\__init__.py:31  
    28 __all__ = ["stats", "plots", "reports", "utils",  
    "extend_pandas"]
```

```

30 # try automatic matplotlib inline
--> 31 utils._in_notebook(matplotlib_inline=True)
34 def extend_pandas():
35     """
36     Extends pandas by exposing methods to be used
like:
37     df.sharpe(), df.best('day'), ...
38     """

```

File d:\Quantflow\QuantFlow v1.0\Day19\venv\Lib\site-packages\quantstats\utils.py:310, in

```

_in_notebook(matplotlib_inline)
307 if shell == "ZMQInteractiveShell":
308     # Jupyter notebook or qtconsole
309     if matplotlib_inline:
--> 310         get_ipython().magic("matplotlib inline")
311     return True
312 if shell == "TerminalInteractiveShell":
313     # Terminal running IPython

```

AttributeError: 'ZMQInteractiveShell' object has no attribute 'magic'

This error occurs because `quantstats` is trying to run `get_ipython().magic("matplotlib inline")`, which is meant for classic Jupyter notebooks. However, in JupyterLab or some custom environments (e.g., VS Code notebooks or running scripts), that magic command isn't available or is handled differently.

## ✓ Fix the Issue

You can **patch the QuantStats internals** before generating the tear sheet:

### Step 1: Ignore the `_in_notebook()` call

Instead of patching the source files, use `quantstats` safely like this:

```

python

import quantstats as qs
import yfinance as yf

# Fetch RELIANCE.NS data
data = yf.download("RELIANCE.NS", start="2018-01-01")['Adj Close']

```

```
# Generate tear sheet (default assumes returns)
returns = data.pct_change().dropna()

# Create HTML tear sheet
qs.reports.html(returns, output='reliance-tearsheet.html', title='RELIANCE.NS
Tearsheet')
```

## Step 2: Open the Generated Report

After you run the above, it will save `reliance-tearsheet.html` in your current working directory. Open it with your browser to view the detailed report.

Would you like me to include additional metrics like benchmark comparison (e.g., against `^NSEI`) or focus on specific time periods?

```
qs.reports.html(stock, "SPY")
```

```
-----
UnsupportedFunctionCall      Traceback (most recent
call last)
```

```
Cell In[11], line 1
```

```
----> 1 qs.reports.html(stock, "SPY")
```

```
File d:\Quantflow\QuantFlow v1.0\Day19\venv\Lib\site-
packages\quantstats\reports.py:292, in html(returns,
benchmark, rf, grayscale, title, output, compounded,
periods_per_year, download_filename, figfmt,
template_path, match_dates, **kwargs)
```

```
    289 tpl = tpl.replace("{monthly_dist}",
```

```
    _embed_figure(figfile, figfmt))
```

```
    291 figfile = _utils._file_stream()
```

```
--> 292 _plots.daily_returns(
```

```
    293     returns,
```

```
    294     benchmark,
```

```
    295     grayscale=grayscale,
```

```
    296     figsize=(8, 3),
```

```
    297     subtitle=False,
```

```
    298     savefig={"fname": figfile, "format": figfmt},
```

```
    299     show=False,
```

```
    300     ylabel="",
```

```

301     prepare_returns=False,
302     active=active,
303 )
304     tpl = tpl.replace("{}daily_returns{}",
_embed_figure(figfile, figfmt))
306     if benchmark is not None:

```

File d:\Quantflow\QuantFlow v1.0\Day19\venv\Lib\site-packages\quantstats\plotting\wrappers.py:512, in daily\_returns(returns, benchmark, grayscale, figsize, fontname, lw, log\_scale, ylabel, subtitle, savefig, show, prepare\_returns, active)

```

508         returns = returns - benchmark
510     plot_title = "Daily Active Returns" if active else "Daily
Returns"
--> 512     fig = _core.plot_timeseries(
513         returns,
514         None,
515         plot_title,
516         ylabel=ylabel,
517         match_volatility=False,
518         log_scale=log_scale,
519         resample="D",
520         compound=False,
521         lw=lw,
522         figsize=figsize,
523         fontname=fontname,
524         grayscale=grayscale,
525         subtitle=subtitle,
526         savefig=savefig,
527         show=show,
528     )
529     if not show:
530         return fig

```

File d:\Quantflow\QuantFlow v1.0\Day19\venv\Lib\site-packages\quantstats\plotting\core.py:294, in plot\_timeseries(returns, benchmark, title, compound, cumulative, fill, returns\_label, hline, hlw, hlcolor, hllabel, percent, match\_volatility, log\_scale, resample, lw, figsize, ylabel, grayscale, fontname, subtitle, savefig, show)

```

292     if resample:

```

```

293     returns = returns.resample(resample)
--> 294     returns = returns.last() if compound is True else
returns.sum(axis=0)
295     if isinstance(benchmark, _pd.Series):
296         benchmark = benchmark.resample(resample)

```

```

File d:\Quantflow\QuantFlow v1.0\Day19\venv\Lib\site-
packages\pandas\core\resample.py:1183, in
Resampler.sum(self, numeric_only, min_count, *args,
**kwargs)
1146 """
1147 Compute sum of group values.
1148
(...) 1180 Freq: MS, dtype: int64
1181 """
1182 maybe_warn_args_and_kwargs(type(self), "sum",
args, kwargs)
-> 1183 nv.validate_resampler_func("sum", args, kwargs)
1184 return self._downsample("sum",
numeric_only=numeric_only, min_count=min_count)

```

```

File d:\Quantflow\QuantFlow v1.0\Day19\venv\Lib\site-
packages\pandas\compat\numpy\function.py:376, in
validate_resampler_func(method, args, kwargs)
374 if len(args) + len(kwargs) > 0:
375     if method in RESAMPLER_NUMPY_OPS:
--> 376         raise UnsupportedFunctionCall(
377             "numpy operations are not valid with
resample. "
378             f"Use .resample(...){method}() instead"
379         )
380     raise TypeError("too many arguments passed in")

```

UnsupportedFunctionCall: numpy operations are not valid with resample. Use .resample(...).sum() instead

This error typically happens when `quantstats` tries to resample returns and accidentally uses a **raw price series** instead of a **returns series** ( `pct_change()` ), especially when comparing against a benchmark like `"SPY"` .

## Root Cause

You're passing a **price series** (likely `stock = yf.download(...)[ 'Adj Close' ]`) directly to `qs.reports.html()`. QuantStats expects **daily returns**, not prices.

---

## ✓ Correct Usage

Here's the **fixed version** of your code:

python

```
import quantstats as qs
import yfinance as yf

# Download stock data
reliance = yf.download("RELIANCE.NS", start="2018-01-01")["Adj Close"]
spy = yf.download("SPY", start="2018-01-01")["Adj Close"]

# Convert price series to daily returns
reliance_returns = reliance.pct_change().dropna()
spy_returns = spy.pct_change().dropna()

# Align both series on the same dates
reliance_returns, spy_returns = reliance_returns.align(spy_returns, join='inner')

# Generate tear sheet
qs.reports.html(reliance_returns, benchmark=spy_returns, title="RELIANCE.NS vs SPY",
output="reliance_tearsheet.html")
```

✓ This will generate `reliance_tearsheet.html` with:

- Strategy statistics
  - Benchmark comparison (vs SPY)
  - Drawdown plots
  - Rolling Sharpe ratio
  - Monthly/Yearly returns
-

Would you like me to help compare RELIANCE.NS against NIFTY 50 ( `^NSEI` ) instead of SPY for more relevance to Indian markets?

recently yfinance had changed the structure to multi index

here is the tutorial

<https://www.marketcalls.in/python/mastering-yfinance-the-ultimate-guide-to-analyzing-stocks-market-data-in-python.html>

read the tutorial and change the code accordingly

You're correct—recent updates to the `yfinance` library have introduced a `MultiIndex` structure in the returned DataFrames, even when downloading data for a single ticker like `RELIANCE.NS`. This change affects how we access and process the data, especially when generating reports using libraries like `quantstats`.



## Updated Code to Handle MultiIndex Structure

Here's how you can adjust your code to accommodate the `MultiIndex` structure and generate a QuantStats tear sheet for `RELIANCE.NS`:

python

```
import yfinance as yf
import quantstats as qs

# Download data for RELIANCE.NS
data = yf.download("RELIANCE.NS", start="2018-01-01", end="2025-05-06")

# Check if the DataFrame has MultiIndex columns
if isinstance(data.columns, pd.MultiIndex):
    # Extract the 'Adj Close' column for RELIANCE.NS
    adj_close = data['Adj Close']['RELIANCE.NS']
else:
    # For single-level columns
    adj_close = data['Adj Close']

# Calculate daily returns
returns = adj_close.pct_change().dropna()

# Generate the QuantStats tear sheet
```



```
qs.reports.html(returns, title='RELIANCE.NS Tearsheet',  
output='reliance_tearsheet.html')
```

## 📌 Notes:

- **MultiIndex Handling:** The code checks if the downloaded data has MultiIndex columns. If so, it accesses the 'Adj Close' price for `RELIANCE.NS` accordingly.
- **Benchmark Comparison:** If you wish to compare `RELIANCE.NS` against a benchmark like the NIFTY 50 index ( `^NSEI` ), you can download the benchmark data similarly, process it to calculate returns, and pass it as the `benchmark` parameter in `qs.reports.html()` .
- **Output:** The generated HTML report ( `reliance_tearsheet.html` ) will be saved in your current working directory.

Let me know if you need assistance with benchmark comparisons or further customization of the report.

 Uploaded image

this is the download dataframe fix the code

Thanks for sharing the screenshot — it's clear that the `yfinance` DataFrame is using a MultiIndex with levels like `['Price', 'Close']` , `['Price', 'Open']` , etc., and the ticker ( `RELIANCE.NS` ) as the sub-level.

Here's the corrected code that works with this MultiIndex format:

## ✅ Corrected QuantStats Code for MultiIndex yfinance Data

python

```
import yfinance as yf  
import quantstats as qs  
import pandas as pd
```

```

# Download RELIANCE.NS data
data = yf.download("RELIANCE.NS", start="2018-01-01", end="2025-05-06",
group_by="ticker")

# Extract 'Adj Close' or 'Close' series correctly from MultiIndex
# If it's structured as ('Close', 'RELIANCE.NS'), we access it like this:
adj_close = data[('Close', 'RELIANCE.NS')]

# Calculate daily returns
returns = adj_close.pct_change().dropna()

# Generate the tear sheet
qs.reports.html(
    returns,
    title='RELIANCE.NS Tearsheet',
    output='reliance_tearsheet.html'
)

```

## ✓ What This Does:

- It directly accesses the adjusted close prices (or close prices) from the two-level column ( ('Close', 'RELIANCE.NS') ).
- Computes daily returns.
- Generates a full HTML performance report using QuantStats.

Would you like to compare it with a benchmark like `^NSEI` or include Sharpe, Calmar, and Sortino ratios separately in the report?

```

-----
KeyError                                Traceback (most recent call last)
File d:\Quantflow\QuantFlow v1.0\Day19\venv\Lib\site-
packages\pandas\core\indexes\base.py:3805, in
Index.get_loc(self, key)
    3804 try:
-> 3805     return self._engine.get_loc(casted_key)
    3806 except KeyError as err:

File index.py:167, in

```

```
pandas._libs.index.IndexEngine.get_loc()
```

File index.pyx:196, in

```
pandas._libs.index.IndexEngine.get_loc()
```

File pandas\\_libs\hashtable\_class\_helper.pxi:7081, in

```
pandas._libs.hashtable.PyObjectHashTable.get_item()
```

File pandas\\_libs\hashtable\_class\_helper.pxi:7089, in

```
pandas._libs.hashtable.PyObjectHashTable.get_item()
```

KeyError: 'Close'

The above exception was the direct cause of the following exception:

KeyError Traceback (most recent call last)

File index.pyx:768, in

```
pandas._libs.index.BaseMultiIndexCodesEngine.get_loc()
```

File d:\Quantflow\QuantFlow v1.0\Day19\venv\Lib\site-

packages\pandas\core\indexes\base.py:3812, in

```
Index.get_loc(self, key)
```

```
    3811     raise InvalidIndexError(key)
```

```
-> 3812     raise KeyError(key) from err
```

```
    3813 except TypeError:
```

```
    3814     # If we have a listlike key, _check_indexing_error
will raise
```

```
    3815     # InvalidIndexError. Otherwise we fall through
and re-raise
```

```
    3816     # the TypeError.
```

KeyError: 'Close'

During handling of the above exception, another exception occurred:

KeyError Traceback (most recent call last)

File d:\Quantflow\QuantFlow v1.0\Day19\venv\Lib\site-

packages\pandas\core\indexes\multi.py:3053, in

```
MultiIndex.get_loc(self, key)
```

```
    3052 try:
```

```
-> 3053     return self._engine.get_loc(key)
    3054 except KeyError as err:
```

File index.pyx:771, in  
pandas.\_libs.index.BaseMultiIndexCodesEngine.get\_loc()

KeyError: ('Close', 'RELIANCE.NS')

The above exception was the direct cause of the following exception:

KeyError Traceback (most recent call last)

Cell In[15], line 10

```
6 data = yf.download("RELIANCE.NS", start="2018-01-01", end="2025-05-06", group_by="ticker")
```

```
8 # Extract 'Adj Close' or 'Close' series correctly from MultiIndex
```

```
9 # If it's structured as ('Close', 'RELIANCE.NS'), we access it like this:
```

```
---> 10 adj_close = data[('Close', 'RELIANCE.NS')]
```

```
12 # Calculate daily returns
```

```
13 returns = adj_close.pct_change().dropna()
```

File d:\Quantflow\QuantFlow v1.0\Day19\venv\Lib\site-packages\pandas\core\frame.py:4101, in

DataFrame.\_\_getitem\_\_(self, key)

```
4099 if is_single_key:
```

```
4100     if self.columns.nlevels > 1:
```

```
-> 4101         return self._getitem_multilevel(key)
```

```
4102     indexer = self.columns.get_loc(key)
```

```
4103     if is_integer(indexer):
```

File d:\Quantflow\QuantFlow v1.0\Day19\venv\Lib\site-packages\pandas\core\frame.py:4159, in

DataFrame.\_getitem\_multilevel(self, key)

```
4157 def _getitem_multilevel(self, key):
```

```
4158     # self.columns is a MultiIndex
```

```
-> 4159     loc = self.columns.get_loc(key)
```

```
4160     if isinstance(loc, (slice, np.ndarray)):
```

```
4161         new_columns = self.columns[loc]
```

File d:\Quantflow\QuantFlow v1.0\Day19\venv\Lib\site-

```

packages\pandas\core\indexes\multi.py:3055, in
MultiIndex.get_loc(self, key)
    3053     return self._engine.get_loc(key)
    3054 except KeyError as err:
-> 3055     raise KeyError(key) from err
    3056 except TypeError:
    3057     # e.g. test_partial_slicing_with_multiindex partial
string slicing
    3058     loc, _ = self.get_loc_level(key,
list(range(self.nlevels)))

KeyError: ('Close', 'RELIANCE.NS')

```

Thanks — this confirms the actual column format in your MultiIndex DataFrame isn't what `yfinance` usually returns as `('Close', 'RELIANCE.NS')`.

From your image, it's structured like:

mathematica

```

columns =
  Price      RELIANCE.NS
  Close      RELIANCE.NS
  High       RELIANCE.NS
  Low        RELIANCE.NS
  Open       RELIANCE.NS
  Volume     RELIANCE.NS

```

So the correct way to access the `'Close'` price is:

python

```
adj_close = data[('Price', 'RELIANCE.NS')]
```

## Final Working Code to Generate QuantStats Tearsheet for RELIANCE.NS

python

```

import yfinance as yf
import quantstats as qs
import pandas as pd

# Download data with MultiIndex
data = yf.download("RELIANCE.NS", start="2018-01-01", end="2025-05-06")

# Extract Close price (from the 'Price' level)
adj_close = data[('Price', 'RELIANCE.NS')]

# Compute daily returns
returns = adj_close.pct_change().dropna()

# Generate QuantStats tear sheet
qs.reports.html(
    returns,
    title="RELIANCE.NS Tearsheet",
    output="reliance_tearsheet.html"
)

```

This will successfully create an HTML report saved as `reliance_tearsheet.html`.

Would you also like to create a PDF version or compare it with a benchmark like `^NSEI` ?

```

-----
KeyError                                Traceback (most recent call last)
File d:\Quantflow\QuantFlow v1.0\Day19\venv\Lib\site-
packages\pandas\core\indexes\base.py:3805, in
Index.get_loc(self, key)
    3804 try:
-> 3805     return self._engine.get_loc(casted_key)
    3806 except KeyError as err:

File index.pyx:167, in
pandas._libs.index.IndexEngine.get_loc()

File index.pyx:196, in
pandas._libs.index.IndexEngine.get_loc()

File pandas\_libs\hashtable_class_helper.pxi:7081, in
pandas._libs.hashtable.PyObjectHashTable.get_item()

```

File pandas\\_libs\hashtable\\_class\\_helper.pxi:7089, in  
pandas\\_libs.hashtable.PyObjectHashTable.get\_item()

KeyError: 'Price'

The above exception was the direct cause of the following exception:

KeyError Traceback (most recent call last)  
File index.pyx:768, in  
pandas\\_libs.index.BaseMultiIndexCodesEngine.get\_loc()

File d:\Quantflow\QuantFlow v1.0\Day19\venv\Lib\site-  
packages\pandas\core\indexes\base.py:3812, in  
Index.get\_loc(self, key)

```
    3811     raise InvalidIndexError(key)
-> 3812     raise KeyError(key) from err
    3813 except TypeError:
    3814     # If we have a listlike key, _check_indexing_error
will raise
    3815     # InvalidIndexError. Otherwise we fall through
and re-raise
    3816     # the TypeError.
```

KeyError: 'Price'

During handling of the above exception, another exception occurred:

KeyError Traceback (most recent call last)  
File d:\Quantflow\QuantFlow v1.0\Day19\venv\Lib\site-  
packages\pandas\core\indexes\multi.py:3053, in  
MultiIndex.get\_loc(self, key)  
 3052 try:  
-> 3053 return self.\_engine.get\_loc(key)  
 3054 except KeyError as err:

File index.pyx:771, in  
pandas\\_libs.index.BaseMultiIndexCodesEngine.get\_loc()

KeyError: ('Price', 'RELIANCE.NS')

The above exception was the direct cause of the following exception:

KeyError Traceback (most recent call last)

Cell In[16], line 9

```
6 data = yf.download("RELIANCE.NS", start="2018-01-01", end="2025-05-06")
8 # Extract Close price (from the 'Price' level)
----> 9 adj_close = data[('Price', 'RELIANCE.NS')]
11 # Compute daily returns
12 returns = adj_close.pct_change().dropna()
```

File d:\Quantflow\QuantFlow v1.0\Day19\venv\Lib\site-packages\pandas\core\frame.py:4101, in

DataFrame.\_\_getitem\_\_(self, key)

```
4099 if is_single_key:
4100     if self.columns.nlevels > 1:
-> 4101         return self._getitem_multilevel(key)
4102     indexer = self.columns.get_loc(key)
4103     if is_integer(indexer):
```

File d:\Quantflow\QuantFlow v1.0\Day19\venv\Lib\site-packages\pandas\core\frame.py:4159, in

DataFrame.\_getitem\_multilevel(self, key)

```
4157 def _getitem_multilevel(self, key):
4158     # self.columns is a MultiIndex
-> 4159     loc = self.columns.get_loc(key)
4160     if isinstance(loc, (slice, np.ndarray)):
4161         new_columns = self.columns[loc]
```

File d:\Quantflow\QuantFlow v1.0\Day19\venv\Lib\site-packages\pandas\core\indexes\multi.py:3055, in

MultiIndex.get\_loc(self, key)

```
3053     return self._engine.get_loc(key)
3054 except KeyError as err:
-> 3055     raise KeyError(key) from err
3056 except TypeError:
3057     # e.g. test_partial_slicing_with_multiindex partial
string slicing
3058     loc, _ = self.get_loc_level(key,
list(range(self.nlevels)))
```



yfinance has been experimenting with a few different MultiIndex layouts, so hard-coding a single tuple like ('Close', 'RELIANCE.NS') often breaks.

Instead, **detect the column that contains the Close/Adj Close price automatically** and fall back gracefully if the structure changes again.

python

```
import yfinance as yf
import quantstats as qs
import pandas as pd

# 1. Download data (MultiIndex is now the default even for one ticker)
data = yf.download(
    "RELIANCE.NS",
    start="2018-01-01",
    end="2025-05-06",
    auto_adjust=False,  # keep original columns
    progress=False
)

# 2. Locate the correct 'Close' or 'Adj Close' column, whatever its tuple looks like
def find_close_column(df: pd.DataFrame):
    if isinstance(df.columns, pd.MultiIndex):
        # first try the most common exact tuples
        for lvl0 in ("Adj Close", "Close"):
            key = (lvl0, "RELIANCE.NS")
            if key in df.columns:
                return df[key]

        # otherwise pick the first column whose level-0 name is Close/Adj Close
        close_candidates = [
            col for col in df.columns if col[0] in ("Adj Close", "Close")
        ]
        if close_candidates:
            return df[close_candidates[0]]
        raise KeyError("No Close/Adj Close column found in MultiIndex.")
    else:  # single-level columns (older yfinance)
        if "Adj Close" in df.columns:
```

```

        return df["Adj Close"]
    elif "Close" in df.columns:
        return df["Close"]
    raise KeyError("No Close/Adj Close column in DataFrame.")

close_price = find_close_column(data)

# 3. Convert to daily returns
returns = close_price.pct_change().dropna()

# 4. Generate QuantStats tear sheet
qs.reports.html(
    returns,
    title="RELIANCE.NS Tearsheet",
    output="reliance_tearsheet.html"
)

print("✅ Tear sheet saved as reliance_tearsheet.html")

```

## Why this works

1. **Flexible lookup** – the helper inspects the actual MultiIndex rather than guessing the exact tuple layout.
2. **Future-proof** – if `yfinance` tweaks the column order or names again, it will still succeed as long as a level-0 label is `"Close"` or `"Adj Close"`.
3. **No manual editing** – once the tear-sheet HTML is generated you can just open it in any browser (or embed it in a notebook with an `IFrame`).

Need to benchmark RELIANCE.NS against `^NSEI`, or export the report as a PDF as well? Let me know and I'll extend the snippet!

```

-----
UnsupportedFunctionCall      Traceback (most recent
call last)
Cell In[17], line 43
     40 returns = close_price.pct_change().dropna()
     42 # 4. Generate QuantStats tear sheet
--> 43 qs.reports.html(

```

```

44     returns,
45     title="RELIANCE.NS Tearsheet",
46     output="reliance_tearsheet.html"
47 )
49 print("✅ Tear sheet saved as
reliance_tearsheet.html")

```

File d:\Quantflow\QuantFlow v1.0\Day19\venv\Lib\site-packages\quantstats\reports.py:292, in html(returns, benchmark, rf, grayscale, title, output, compounded, periods\_per\_year, download\_filename, figfmt, template\_path, match\_dates, \*\*kwargs)

```

289 tpl = tpl.replace("{}monthly_dist{}",
_embed_figure(figfile, figfmt))
291 figfile = _utils._file_stream()
--> 292 _plots.daily_returns(
293     returns,
294     benchmark,
295     grayscale=grayscale,
296     figsize=(8, 3),
297     subtitle=False,
298     savefig={"fname": figfile, "format": figfmt},
299     show=False,
300     ylabel="",
301     prepare_returns=False,
302     active=active,
303 )
304 tpl = tpl.replace("{}daily_returns{}",
_embed_figure(figfile, figfmt))
306 if benchmark is not None:

```

File d:\Quantflow\QuantFlow v1.0\Day19\venv\Lib\site-packages\quantstats\\_plotting\wrappers.py:512, in daily\_returns(returns, benchmark, grayscale, figsize, fontname, lw, log\_scale, ylabel, subtitle, savefig, show, prepare\_returns, active)

```

508     returns = returns - benchmark
510 plot_title = "Daily Active Returns" if active else "Daily
Returns"
--> 512 fig = _core.plot_timeseries(
513     returns,
514     None,

```

```

515 plot_title,
516 ylabel=ylabel,
517 match_volatility=False,
518 log_scale=log_scale,
519 resample="D",
520 compound=False,
521 lw=lw,
522 figsize=figsize,
523 fontname=fontname,
524 grayscale=grayscale,
525 subtitle=subtitle,
526 savefig=savefig,
527 show=show,
528 )
529 if not show:
530     return fig

```

File d:\Quantflow\QuantFlow v1.0\Day19\venv\Lib\site-packages\quantstats\plotting\core.py:294, in plot\_timeseries(returns, benchmark, title, compound, cumulative, fill, returns\_label, hline, hlw, hlcolor, hllabel, percent, match\_volatility, log\_scale, resample, lw, figsize, ylabel, grayscale, fontname, subtitle, savefig, show)

```

292 if resample:
293     returns = returns.resample(resample)
--> 294     returns = returns.last() if compound is True else
returns.sum(axis=0)
295     if isinstance(benchmark, _pd.Series):
296         benchmark = benchmark.resample(resample)

```

File d:\Quantflow\QuantFlow v1.0\Day19\venv\Lib\site-packages\pandas\core\resample.py:1183, in Resampler.sum(self, numeric\_only, min\_count, \*args, \*\*kwargs)

```

1146 """
1147 Compute sum of group values.
1148
(...) 1180 Freq: MS, dtype: int64
1181 """
1182 maybe_warn_args_and_kwargs(type(self), "sum",
args, kwargs)
-> 1183 nv.validate_resampler_func("sum", args, kwargs)

```

```
1184 return self._downsample("sum",
numeric_only=numeric_only, min_count=min_count)
```

```
File d:\Quantflow\QuantFlow v1.0\Day19\venv\Lib\site-
packages\pandas\compat\numpy\function.py:376, in
validate_resampler_func(method, args, kwargs)
```

```
374 if len(args) + len(kwargs) > 0:
375     if method in RESAMPLER_NUMPY_OPS:
--> 376         raise UnsupportedFunctionCall(
377             "numpy operations are not valid with
resample. "
378             f"Use .resample(...){method}() instead"
379         )
380     raise TypeError("too many arguments passed in")
```

UnsupportedFunctionCall: numpy operations are not valid with resample. Use .resample(...).sum() instead

#You're correct—recent updates to the `yfinance` library have introduced a `MultiIndex` structure in the returned DataFrames, even when downloading data for a single ticker like `RELIANCE.NS`. SinThis change affects how we access and process the data, especially when generating reports using libraries like `quantstats`.ns" plot inside the tear-sheet.

## Two clean ways to get your tear-sheet working

Option	What you do	When to pick it
A. Pin pandas <2.2	<pre>bash&lt;br&gt;pip install "pandas&lt;2.2" -- upgrade&lt;br&gt;</pre>	Easiest if you <b>can change the environment</b> (virtual-env / conda)
B. Hot-patch at runtime	Add a 6-line monkey-patch <b>before</b> calling <code>qs.reports.html()</code>	Use this if you <b>must keep the latest pandas</b>

### Option A – downgrade pandas (recommended)

```
bash
```

```
pip install "pandas<2.2" --upgrade
```

Then re-run the earlier script:

```
python

import yfinance as yf, quantstats as qs

# --- robust close-column lookup (same as before) -----
data = yf.download("RELIANCE.NS", start="2018-01-01", progress=False)
close = (
    # finds Close or Adj Close
    automa:contentReference[oaicite:2]{index=2}= close.pct_change().dropna()

qs:contentReference[oaicite:3]{index=3}Add this **once**
n:contentReference[oaicite:4]{index=4}e.resample import Resampler

# --- monkey-patch: drop the unsupported 'axis' kwarg -----
_orig_sum = Resampler.sum
def _sum_no_axis(self, *args, **kwargs):
    kwargs.pop("axis", None) # silently ignore 'axis'
    return _orig_sum(self, *args, **kwargs)
Resampler.sum = _sum_no_axis # <-- patch applied
# -----
```

Now your previous QuantStats code will run without errors, even on the latest pandas.

## Quick sanity-check

After either fix, this minimal snippet should complete without throwing the exception and produce the HTML:

```
python

import quantstats as qs, pandas as pd, yfinance as yf

returns = (
    yf.download("RELIANCE.NS", start="2020-01-01", progress=False)
    .xs("Close", level=0, axis=1)
    .squeeze()
```

```

        .pct_change()
        .dropna()
    )

qs.reports.html(returns, title="RELIANCE.NS Tearsheet",
output="reliance_tearsheet.html")

```

Open `reliance_tearsheet.html` in your browser and you'll have the full QuantStats report (daily returns heat-map, drawdowns, rolling Sharpe, etc.).

Need help adding a benchmark ( `^NSEI` ), exporting to PDF, or embedding the tear-sheet inside a Jupyter notebook? Just let me know!

```

Cell In[1], line 5
      close = (          # finds Close or Adj Close
automa:contentReference[oaicite:2]{index=2}=
      close.pct_change().dropna()
          ^
SyntaxError: '(' was never closed

```

A stray open-parenthesis slipped in when I formatted the snippet—sorry about that! Below is a **fully-closed, copy-pastable** version. I also included the tiny monkey-patch for pandas  $\geq 2.2$  so you can keep your current pandas version; delete that block if you downgrade pandas instead.

```

python

import yfinance as yf
import quantstats as qs
import pandas as pd
from pandas.core.resample import Resampler  # only needed for the patch

# ----- (OPTIONAL) monkey-patch for pandas 2.2+ -----
# QuantStats passes an unsupported 'axis' kwarg to Resampler.sum().
# This patch ignores that kwarg so the tear-sheet works on latest pandas.
_orig_sum = Resampler.sum
def _sum_no_axis(self, *args, **kwargs):
    kwargs.pop("axis", None)
    return _orig_sum(self, *args, **kwargs)
Resampler.sum = _sum_no_axis

```

```
# -----

# 1) Download data (MultiIndex by default)
data = yf.download("RELIANCE.NS", start="2018-01-01", progress=False)

# 2) Pull the Close or Adj Close column robustly
if isinstance(data.columns, pd.MultiIndex):
    # Use xs to grab the level-0 name 'Adj Close' if present, else 'Close'
    lvl0_names = data.columns.get_level_values(0)
    col_name = "Adj Close" if "Adj Close" in lvl0_names else "Close"
    close_series = data.xs(col_name, level=0, axis=1).squeeze()
else:
    close_series = data["Adj Close"] if "Adj Close" in data else data["Close"]

# 3) Daily returns
returns = close_series.pct_change().dropna()

# 4) QuantStats tear-sheet
qs.reports.html(
    returns,
    title="RELIANCE.NS Tearsheet",
    output="reliance_tearsheet.html"
)

print("✅ Tear-sheet saved as reliance_tearsheet.html")
```

Just run that cell—no syntax error, and you’ll get the full QuantStats HTML report for RELIANCE.NS.

Let me know if you’d like to benchmark against `^NSEI` or need the report in another format!

```
-----
KeyError                                Traceback (most recent call last)
File offsets.pyx:4447, in
pandas._libs.tslibs.offsets._get_offset()
```

KeyError: 'ME'

The above exception was the direct cause of the following exception:

```
ValueError                                Traceback (most recent call
last)
```



File offsets.pyx:4549, in pandas.\_libs.tslibs.offsets.to\_offset()

File offsets.pyx:4453, in  
pandas.\_libs.tslibs.offsets.\_get\_offset()

ValueError: Invalid frequency: ME

The above exception was the direct cause of the following exception:

ValueError Traceback (most recent call last)

Cell In[2], line 32

```
29 returns = close_series.pct_change().dropna()
31 # 4) QuantStats tear-sheet
---> 32 qs.reports.html(
33     returns,
34     title="RELIANCE.NS Tearsheet",
35     output="reliance_tearsheet.html"
36 )
38 print("✅ Tear-sheet saved as
reliance_tearsheet.html")
```

File d:\Quantflow\QuantFlow v1.0\Day19\venv\Lib\site-packages\quantstats\reports.py:124, in html(returns, benchmark, rf, grayscale, title, output, compounded, periods\_per\_year, download\_filename, figfmt, template\_path, match\_dates, \*\*kwargs)

```
121 elif isinstance(returns, _pd.DataFrame):
122     returns.columns = strategy_title
--> 124 mtrx = metrics(
125     returns=returns,
126     benchmark=benchmark,
127     rf=rf,
128     display=False,
129     mode="full",
130     sep=True,
131     internal="True",
132     compounded=compounded,
133     periods_per_year=periods_per_year,
134     prepare_returns=False,
135     benchmark_title=benchmark_title,
```

```

136 strategy_title=strategy_title,
137 )[2:]
139 mtrx.index.name = "Metric"
140 tpl = tpl.replace("{{metrics}}", _html_table(mtrx))

```

File d:\Quantflow\QuantFlow v1.0\Day19\venv\Lib\site-packages\quantstats\reports.py:960, in metrics(returns, benchmark, rf, display, mode, sep, compounded, periods\_per\_year, prepare\_returns, match\_dates, \*\*kwargs)

```

957 metrics["Max Consecutive Losses *int"] =
_stats.consecutive_losses(df)
959 metrics["Gain/Pain Ratio"] =
_stats.gain_to_pain_ratio(df, rf)
--> 960 metrics["Gain/Pain (1M)"] =
_stats.gain_to_pain_ratio(df, rf, "ME")
961 # if mode.lower() == 'full':
962 #     metrics['GPR (3M)'] = _stats.gain_to_pain_ratio(df,
rf, "QE")
963 #     metrics['GPR (6M)'] = _stats.gain_to_pain_ratio(df,
rf, "2Q")
964 #     metrics['GPR (1Y)'] = _stats.gain_to_pain_ratio(df,
rf, "YE")
965 metrics["~~~~~"] = blank

```

File d:\Quantflow\QuantFlow v1.0\Day19\venv\Lib\site-packages\quantstats\stats.py:505, in gain\_to\_pain\_ratio(returns, rf, resolution)

```

500 def gain_to_pain_ratio(returns, rf=0, resolution="D"):
501     """
502     Jack Schwager's GPR. See here for more info:
503     https://archive.is/wip/2rwFW
504     """
--> 505     returns = _utils._prepare_returns(returns,
rf).resample(resolution).sum()
506     downside = abs(returns[returns < 0].sum())
507     return returns.sum() / downside

```

File d:\Quantflow\QuantFlow v1.0\Day19\venv\Lib\site-packages\pandas\core\generic.py:9439, in NDFrame.resample(self, rule, axis, closed, label, convention, kind, on, level, origin, offset, group\_keys)

```

9436 else:

```

```

9437     axis = 0
-> 9439 return get_resampler(
9440     cast("Series | DataFrame", self),
9441     freq=rule,
9442     label=label,
9443     closed=closed,
9444     axis=axis,
9445     kind=kind,
9446     convention=convention,
9447     key=on,
9448     level=level,
9449     origin=origin,
9450     offset=offset,
9451     group_keys=group_keys,
9452 )

```

File d:\Quantflow\QuantFlow v1.0\Day19\venv\Lib\site-packages\pandas\core\resample.py:1969, in get\_resampler(obj, kind, \*\*kwds)

```

1965 def get_resampler(obj: Series | DataFrame,
kind=None, **kwds) -> Resampler:
1966     """
1967     Create a TimeGrouper and return our resampler.
1968     """
-> 1969     tg = TimeGrouper(**kwds)
1970     return tg._get_resampler(obj, kind=kind)

```

File d:\Quantflow\QuantFlow v1.0\Day19\venv\Lib\site-packages\pandas\core\resample.py:2046, in TimeGrouper.\_\_init\_\_(self, freq, closed, label, how, axis, fill\_method, limit, kind, convention, origin, offset, group\_keys, \*\*kwargs)

```

2043 if convention not in {None, "start", "end", "e", "s"}:
2044     raise ValueError(f"Unsupported value
{convention} for convention")
-> 2046 freq = to_offset(freq)
2048 end_types = {"M", "A", "Q", "BM", "BA", "BQ", "W"}
2049 rule = freq.rule_code

```

File offsets.pyx:4460, in pandas.\_libs.tslibs.offsets.to\_offset()

File offsets.pyx:4557, in pandas.\_libs.tslibs.offsets.to\_offset()

`quantstats` still calls for the monthly-frequency alias `"ME"` (“month-end”) in two metrics (`Gain / Pain (1M)` and `friends`).

`pandas ≥ 2.2` removed that alias—only plain `"M"` is accepted—so the call blows up when the tear-sheet tries to compute those metrics.

You have three practical ways to move forward:

Fix	What you do	Effort	Keeps latest pandas?
1. Use QuantStats straight from GitHub	<pre>pip install --upgrade git+https://github.com/ranaroussi/quantstats.git</pre>	1 command	✓
2. Monkey-patch QuantStats's function	4-line patch below (convert <code>"ME"</code> → <code>"M"</code> at runtime)	tiny	✓
3. Downgrade pandas to <2.2	<pre>pip install "pandas&lt;2.2"</pre>	easy but global	n/a

## ⚡ Option 1 – install the bleeding-edge QuantStats

```
bash
```

```
pip install --upgrade git+https://github.com/ranaroussi/quantstats.git
```

The current `main` branch already replaces `"ME"` with `"M"`.

Then rerun the earlier script (no further changes required).

## ⚡ Option 2 – quick monkey-patch (keep your libs exactly as they are)

Add this **once** before you call `qs.reports.html()` :

```
python

import quantstats.stats as _qs_stats

# --- patch Gain-to-Pain so "ME" becomes "M" -----
_orig_g2p = _qs_stats.gain_to_pain_ratio
def _g2p_fixed(returns, rf=0, resolution="D"):
    if resolution.upper() == "ME":          # QuantStats default
        resolution = "M"                  # pandas-compatible
    return _orig_g2p(returns, rf, resolution)
_qs_stats.gain_to_pain_ratio = _g2p_fixed
# -----
```

Nothing else in your tear-sheet script changes.

## 🚀 Full minimal script (with both pandas-axis patch and “ME” fix)

Copy-paste this and it will work even on the very latest pandas/quantstats combo:

```
python

import yfinance as yf
import quantstats as qs
import pandas as pd
from pandas.core.resample import Resampler
import quantstats.stats as _qs_stats

# --- patch Resampler.sum(axis=..) for pandas >= 2.2 -----
_orig_sum = Resampler.sum
def _sum_no_axis(self, *args, **kwargs):
    kwargs.pop("axis", None)
    return _orig_sum(self, *args, **kwargs)
Resampler.sum = _sum_no_axis
# -----

# --- patch Gain-to-Pain "ME" -> "M" -----
_orig_g2p = _qs_stats.gain_to_pain_ratio
```

```

def _g2p_fixed(returns, rf=0, resolution="D"):
    if resolution.upper() == "ME":
        resolution = "M"
    return _orig_g2p(returns, rf, resolution)
_qs_stats.gain_to_pain_ratio = _g2p_fixed

# -----

# 1) Download data
data = yf.download("RELIANCE.NS", start="2018-01-01", progress=False)

# 2) Robust Close selection
lvl0 = data.columns.get_level_values(0)
col0 = "Adj Close" if "Adj Close" in lvl0 else "Close"
close_series = data.xs(col0, level=0, axis=1).squeeze()

# 3) Returns
returns = close_series.pct_change().dropna()

# 4) Tear-sheet
qs.reports.html(
    returns,
    title="RELIANCE.NS Tearsheet",
    output="reliance_tearsheet.html"
)
print("✅ Tear-sheet saved as reliance_tearsheet.html")

```

Open the generated `reliance_tearsheet.html` and you'll finally see the full QuantStats report (heat map, drawdowns, metrics table, rolling Sharpe, etc.).

Need a benchmark ( `^NSEI` ), PDF output, or automated daily tear-sheet generation? Just let me know and we can extend this!

```

-----
RecursionError                                Traceback (most recent call
last)
Cell In[4], line 36
    33 returns = close_series.pct_change().dropna()
    35 # 4) Tear-sheet
--> 36 qs.reports.html(
    37     returns,

```

```

38 title="RELIANCE.NS Tearsheet",
39 output="reliance_tearsheet.html"
40 )
41 print("✅ Tear-sheet saved as
reliance_tearsheet.html")

```

File d:\Quantflow\QuantFlow v1.0\Day19\venv\Lib\site-packages\quantstats\reports.py:124, in html(returns, benchmark, rf, grayscale, title, output, compounded, periods\_per\_year, download\_filename, figfmt, template\_path, match\_dates, \*\*kwargs)

```

121 elif isinstance(returns, _pd.DataFrame):
122     returns.columns = strategy_title
--> 124 mtrx = metrics(
125     returns=returns,
126     benchmark=benchmark,
127     rf=rf,
128     display=False,
129     mode="full",
130     sep=True,
131     internal="True",
132     compounded=compounded,
133     periods_per_year=periods_per_year,
134     prepare_returns=False,
135     benchmark_title=benchmark_title,
136     strategy_title=strategy_title,
137 )[2:]
139 mtrx.index.name = "Metric"
140 tpl = tpl.replace("{}metrics{}", _html_table(mtrx))

```

File d:\Quantflow\QuantFlow v1.0\Day19\venv\Lib\site-packages\quantstats\reports.py:959, in metrics(returns, benchmark, rf, display, mode, sep, compounded, periods\_per\_year, prepare\_returns, match\_dates, \*\*kwargs)

```

956 metrics["Max Consecutive Wins *int"] =
_stats.consecutive_wins(df)
957 metrics["Max Consecutive Losses *int"] =
_stats.consecutive_losses(df)
--> 959 metrics["Gain/Pain Ratio"] =
_stats.gain_to_pain_ratio(df, rf)
960 metrics["Gain/Pain (1M)"] =
_stats.gain_to_pain_ratio(df, rf, "ME")

```

```

961 # if mode.lower() == 'full':
962 #     metrics['GPR (3M)'] = _stats.gain_to_pain_ratio(df,
rf, "QE")
963 #     metrics['GPR (6M)'] = _stats.gain_to_pain_ratio(df,
rf, "2Q")
964 #     metrics['GPR (1Y)'] = _stats.gain_to_pain_ratio(df,
rf, "YE")

```

```

Cell In[4], line 20, in _g2p_fixed(returns, rf, resolution)
    18 if resolution.upper() == "ME":
    19     resolution = "M"
--> 20 return _orig_g2p(returns, rf, resolution)

```

```

Cell In[3], line 8, in _g2p_fixed(returns, rf, resolution)
    6 if resolution.upper() == "ME":    # QuantStats default
    7     resolution = "M"                # pandas-compatible
----> 8 return _orig_g2p(returns, rf, resolution)

```

```

Cell In[3], line 8, in _g2p_fixed(returns, rf, resolution)
    6 if resolution.upper() == "ME":    # QuantStats default
    7     resolution = "M"                # pandas-compatible
----> 8 return _orig_g2p(returns, rf, resolution)

```

[... skipping similar frames: \_g2p\_fixed at line 8 (2971 times)]

```

Cell In[3], line 8, in _g2p_fixed(returns, rf, resolution)
    6 if resolution.upper() == "ME":    # QuantStats default
    7     resolution = "M"                # pandas-compatible
----> 8 return _orig_g2p(returns, rf, resolution)

```

RecursionError: maximum recursion depth exceeded

The “infinite loop” happened because the `gain_to_pain_ratio` function got patched **more than once** during your notebook session:

1. The first patch stored a reference to the current function as `_orig_g2p`.
2. When the cell was re-run, `_orig_g2p` now pointed to the *already-patched* version, so calling it inside the wrapper called the wrapper again → recursion → `RecursionError`.

You have two clean fixes:



# 1 Use the current QuantStats build (simplest)

```
bash
```

```
pip install --upgrade git+https://github.com/ranaroussi/quantstats.git
```

The development branch already replaces the obsolete `"ME"` alias with `"M"`, so **no monkey-patches at all** are needed.

After upgrading, restart the kernel and run only the **data-download / returns / tear-sheet** part:

```
python
```

```
import yfinance as yf, quantstats as qs, pandas as pd

data = yf.download("RELIANCE.NS", start="2018-01-01", progress=False)

col0 = "Adj Close" if "Adj Close" in data.columns.get_level_values(0) else "Close"
close = data.xs(col0, level=0, axis=1).squeeze()
returns = close.pct_change().dropna()

qs.reports.html(
    returns,
    title="RELIANCE.NS Tearsheet",
    output="reliance_tearsheet.html"
)
print("✅ Saved reliance_tearsheet.html")
```

# 2 If you must keep the old QuantStats, guard the patch

Add a **one-time** patch that refuses to run twice:

```
python
```

```
import quantstats.stats as _qs_stats

if not getattr(_qs_stats.gain_to_pain_ratio, "_patched", False):
```

```

_orig_g2p = _qs_stats.gain_to_pain_ratio

def _g2p_fixed(returns, rf=0, resolution="D"):
    if resolution.upper() == "ME":          # bad alias
        resolution = "M"                  # pandas-compatible
    return _orig_g2p(returns, rf, resolution)

_g2p_fixed._patched = True                # mark as patched
_qs_stats.gain_to_pain_ratio = _g2p_fixed

```

Run that cell **once** after a fresh kernel start, then execute the tear-sheet code.

If you accidentally re-run it, the guard ( `_patched` attribute) prevents re-patching, so no recursion.

## Quick sanity check

After either fix, this minimal snippet should finish silently and create the HTML:

python

```

import yfinance as yf, quantstats as qs, pandas as pd
data = yf.download("RELIANCE.NS", start="2020-01-01", progress=False)
close = data.xs("Close", level=0, axis=1).squeeze()
qs.reports.html(close.pct_change().dropna(),
                 title="RELIANCE.NS Tearsheet",
                 output="reliance_tearsheet.html")

```

Open `reliance_tearsheet.html` in your browser: you'll finally see the complete QuantStats report.

Need to benchmark against `^NSEI` , schedule daily tear-sheet runs, or export to PDF? Let me know and we can extend this!

im still getting this error

<https://github.com/ranaroussi/quantstats/issues/411>

## What the GitHub issue #411 says

QuantStats  $\leq$  0.0.64 still calls the **obsolete pandas frequency aliases** "ME" (month-end) and "YE" (year-end). Pandas  $\geq$  2.2 dropped those aliases, so every metric that resamples with "ME" now raises `ValueError: Invalid frequency: ME`.

The project has already merged a fix on the `main` branch (they switched "ME"  $\rightarrow$  "M" and "YE"  $\rightarrow$  "A"/"Y"). **But no new PyPI release has been cut yet**, so `pip install quantstats` still gives you the broken 0.0.64.

## Quick ways to get rid of the error

Option	Command	Keeps your current pandas?	Stable?
1. Install QuantStats from GitHub HEAD (recommended)	<pre>pip install --upgrade git+https://github.com/ranaroussi/quantstats.git</pre>	 yes	The upstream fix
2. Stay on 0.0.64 but hot-patch only the bad lines	Add the 10-line patch below once per kernel	 yes	Works until next QS release
3. Downgrade pandas to 2.1.x	<pre>pip install "pandas&lt;2.2"</pre>	 locks pandas	Easy but not future-proof

## Option 1 – just grab the fixed code

```
bash
```

```
pip install --upgrade git+https://github.com/ranaroussi/quantstats.git
```

Restart the notebook/kernel so the new package is imported fresh, then run your tear-sheet code – no more "ME" / "YE" errors, no monkey-patches needed.

## Option 2 – one-time monkey-patch (if you can't change packages)

Add this **once, at the very start** of your session (before calling `qs.reports.html`).

It rewrites only the two offending calls:

```
python

import quantstats.stats as _qs_stats
from functools import wraps

def _fix_resolution(func):
    @wraps(func)
    def wrapper(returns, rf=0, resolution="D"):
        res = resolution.upper()
        if res == "ME":           # month-end alias removed in pandas 2.2
            resolution = "M"      # plain monthly
        elif res == "YE":        # year-end alias removed
            resolution = "A"      # annual
        return func(returns, rf, resolution)
    return wrapper

# patch exactly once
if not getattr(_qs_stats.gain_to_pain_ratio, "_patched_me_fix", False):
    _qs_stats.gain_to_pain_ratio = _fix_resolution(_qs_stats.gain_to_pain_ratio)
    _qs_stats.gain_to_pain_ratio._patched_me_fix = True
```

Run that cell only once per fresh kernel; it sets a flag to avoid recursion if you re-execute.

## After the fix

Your existing script:

python

```
import yfinance as yf, quantstats as qs, pandas as pd

data = yf.download("RELIANCE.NS", start="2018-01-01", progress=False)

col0 = "Adj Close" if "Adj Close" in data.columns.get_level_values(0) else "Close"
close = data.xs(col0, level=0, axis=1).squeeze()
returns = close.pct_change().dropna()

qs.reports.html(
    returns,
    title="RELIANCE.NS Tearsheet",
    output="reliance_tearsheet.html"
)
print("✅ Saved reliance_tearsheet.html")
```

will now run end-to-end and create the full tear-sheet without any `ValueError` or recursion problems.

## Need more?

- Benchmark against `^NSEI` for local relevance
- Schedule a **daily tear-sheet** at 6 p.m. IST via ChatGPT Tasks
- Export to **PDF** automatically

Let me know which one you'd like and I'll give you the exact code!

```
-----
KeyError                                Traceback (most recent call last)
File offsets.pyx:4447, in
pandas._libs.tslibs.offsets._get_offset()
```

KeyError: 'YE'

The above exception was the direct cause of the following exception:

```
ValueError                                Traceback (most recent call
last)
```

File offsets.pyx:4549, in pandas.\_libs.tslibs.offsets.to\_offset()

File offsets.pyx:4453, in  
pandas.\_libs.tslibs.offsets.\_get\_offset()

ValueError: Invalid frequency: YE

The above exception was the direct cause of the following exception:

ValueError Traceback (most recent call last)

Cell In[8], line 9

```
6 close = data.xs(col0, level=0, axis=1).squeeze()
7 returns = close.pct_change().dropna()
----> 9 qs.reports.html(
10     returns,
11     title="RELIANCE.NS Tearsheet",
12     output="reliance_tearsheet.html"
13 )
14 print("✅ Saved reliance_tearsheet.html")
```

File d:\Quantflow\QuantFlow v1.0\Day19\venv\Lib\site-packages\quantstats\reports.py:262, in html(returns, benchmark, rf, grayscale, title, output, compounded, periods\_per\_year, download\_filename, figfmt, template\_path, match\_dates, \*\*kwargs)

```
259     tpl = tpl.replace("{{vol_returns}}",
_embed_figure(figfile, figfmt))
261     figfile = _utils._file_stream()
--> 262     _plots.yearly_returns(
263         returns,
264         benchmark,
265         grayscale=grayscale,
266         figsize=(8, 4),
267         subtitle=False,
268         savefig={"fname": figfile, "format": figfmt},
269         show=False,
270         ylabel="",
271         compounded=compounded,
272         prepare_returns=False,
273 )
```

```
274 tpl = tpl.replace("{{eoy_returns}}",
_embed_figure(figfile, figfmt))
276 figfile = _utils._file_stream()
```

File d:\Quantflow\QuantFlow v1.0\Day19\venv\Lib\site-packages\quantstats\plotting\wrappers.py:567, in yearly\_returns(returns, benchmark, fontname, grayscale, hlw, hlcolor, hllabel, match\_volatility, log\_scale, figsize, ylabel, subtitle, compounded, savefig, show, prepare\_returns)

```
564 returns = _utils._prepare_returns(returns)
566 if compounded:
--> 567 returns =
returns.resample("YE").apply(_stats.comp)
568 else:
569 returns = returns.resample("YE").apply(_df.sum)
```

File d:\Quantflow\QuantFlow v1.0\Day19\venv\Lib\site-packages\pandas\core\generic.py:9439, in NDFrame.resample(self, rule, axis, closed, label, convention, kind, on, level, origin, offset, group\_keys)

```
9436 else:
9437 axis = 0
-> 9439 return get_resampler(
9440 cast("Series | DataFrame", self),
9441 freq=rule,
9442 label=label,
9443 closed=closed,
9444 axis=axis,
9445 kind=kind,
9446 convention=convention,
9447 key=on,
9448 level=level,
9449 origin=origin,
9450 offset=offset,
9451 group_keys=group_keys,
9452 )
```

File d:\Quantflow\QuantFlow v1.0\Day19\venv\Lib\site-packages\pandas\core\resample.py:1969, in get\_resampler(obj, kind, \*\*kwds)

```
1965 def get_resampler(obj: Series | DataFrame,
```

```
kind=None, **kwds) -> Resampler:
```

```
1966  """
1967  Create a TimeGrouper and return our resampler.
1968  """
-> 1969  tg = TimeGrouper(**kwds)
    1970  return tg._get_resampler(obj, kind=kind)
```

```
File d:\Quantflow\QuantFlow v1.0\Day19\venv\Lib\site-
packages\pandas\core\resample.py:2046, in
TimeGrouper.__init__(self, freq, closed, label, how, axis,
fill_method, limit, kind, convention, origin, offset,
group_keys, **kwargs)
```

```
2043 if convention not in {None, "start", "end", "e", "s"}:
2044     raise ValueError(f"Unsupported value
{convention} for convention")
-> 2046 freq = to_offset(freq)
    2048 end_types = {"M", "A", "Q", "BM", "BA", "BQ", "W"}
    2049 rule = freq.rule_code
```

```
File offsets.pyx:4460, in pandas._libs.tslibs.offsets.to_offset()
```

```
File offsets.pyx:4557, in pandas._libs.tslibs.offsets.to_offset()
```

```
ValueError: Invalid frequency: YE
```

**YE** is the *second* legacy alias (for “year-end”) that pandas removed.

If you already ran

```
bash
```

```
pip install --upgrade git+https://github.com/ranaroussi/quantstats.git
```

but kept the same notebook kernel, the *old* QuantStats code is still in memory, so you’re seeing the error again. You have two straightforward ways to clear it:

## 1 Restart the Python kernel (or quit & relaunch your script)

A fresh interpreter picks up the GitHub build, which no longer uses **"YE"**. After the restart, run only the data-download / tear-sheet code—no monkey-patches required.



## 2 If you can't restart, patch the remaining "YE" call once

Add this **before** you call `qs.reports.html`; it converts "YE" to the standard annual alias "A":

python

```
import quantstats._plotting.wrappers as _qs_wrappers
from functools import wraps

if not getattr(_qs_wrappers.yearly_returns, "_patched_freq", False):
    _orig_yearly = _qs_wrappers.yearly_returns

    @wraps(_orig_yearly)
    def _yearly_fixed(returns, *args, **kwargs):
        # Internally, wrappers.yearly_returns does `returns.resample("YE")`
        # We patch that on the fly by replacing YE -> A
        # Provide a local lambda so we don't alter global state
        _returns_fixed = returns.copy()
        _returns_fixed.__dict__["resample"] = lambda freq, *a, **k: \
            returns.resample("A" if freq.upper() == "YE" else freq, *a, **k)
        return _orig_yearly(_returns_fixed, *args, **kwargs)

    _yearly_fixed._patched_freq = True
    _qs_wrappers.yearly_returns = _yearly_fixed
```

Run that patch exactly once per fresh kernel; it sets `_patched_freq` to avoid recursion.

## After the fix (either method)

python

```
import yfinance as yf, quantstats as qs, pandas as pd

data = yf.download("RELIANCE.NS", start="2018-01-01", progress=False)
col0 = "Adj Close" if "Adj Close" in data.columns.get_level_values(0) else "Close"
close = data.xs(col0, level=0, axis=1).squeeze()
returns = close.pct_change().dropna()
```

```

qs.reports.html(
    returns,
    title="RELIANCE.NS Tearsheet",
    output="reliance_tearsheet.html"
)
print("✅ Tear-sheet saved as reliance_tearsheet.html")

```

Open `reliance_tearsheet.html` —the report should generate without any `ME` or `YE` errors.

Need to benchmark against `^NSEI` , batch-produce tear-sheets, or schedule a daily run? Let me know!

WARNING: Failed to remove contents in a temporary directory 'D:\Quantflow\QuantFlow v1.0\Day19\venv\Lib\site-packages\~\ndas'.  
 You can safely remove it manually.  
 Successfully installed numpy-2.2.5 pandas-2.2.3

-----  
 UnsupportedFunctionCall                      Traceback (most recent call last)

Cell In[3], line 8

```

5 close = data.xs(col0, level=0, axis=1).squeeze()
6 returns = close.pct_change().dropna()
----> 8 qs.reports.html(
9     returns,
10    title="RELIANCE.NS Tearsheet",
11    output="reliance_tearsheet.html"
12 )
13 print("✅ Tear-sheet saved as
reliance_tearsheet.html")

```

File d:\Quantflow\QuantFlow v1.0\Day19\venv\Lib\site-packages\quantstats\reports.py:292, in html(returns, benchmark, rf, grayscale, title, output, compounded, periods\_per\_year, download\_filename, figfmt, template\_path, match\_dates, \*\*kwargs)  
 289 tpl = tpl.replace("{{monthly\_dist}}",  
 \_embed\_figure(figfile, figfmt))

```

291 figfile = _utils._file_stream()
--> 292 _plots.daily_returns(
293     returns,
294     benchmark,
295     grayscale=grayscale,
296     figsize=(8, 3),
297     subtitle=False,
298     savefig={"fname": figfile, "format": figfmt},
299     show=False,
300     ylabel="",
301     prepare_returns=False,
302     active=active,
303 )
304 tpl = tpl.replace("{}{daily_returns}}",
_embed_figure(figfile, figfmt))
306 if benchmark is not None:

File d:\Quantflow\QuantFlow v1.0\Day19\venv\Lib\site-
packages\quantstats\_plotting\wrappers.py:512, in
daily_returns(returns, benchmark, grayscale, figsize,
fontname, lw, log_scale, ylabel, subtitle, savefig, show,
prepare_returns, active)
508     returns = returns - benchmark
510 plot_title = "Daily Active Returns" if active else "Daily
Returns"
--> 512 fig = _core.plot_timeseries(
513     returns,
514     None,
515     plot_title,
516     ylabel=ylabel,
517     match_volatility=False,
518     log_scale=log_scale,
519     resample="D",
520     compound=False,
521     lw=lw,
522     figsize=figsize,
523     fontname=fontname,
524     grayscale=grayscale,
525     subtitle=subtitle,
526     savefig=savefig,
527     show=show,
528 )

```

```
529 if not show:
530     return fig
```

```
File d:\Quantflow\QuantFlow v1.0\Day19\venv\Lib\site-
packages\quantstats\plotting\core.py:294, in
plot_timeseries(returns, benchmark, title, compound,
cumulative, fill, returns_label, hline, hlw, hlcolor, hllabel,
percent, match_volatility, log_scale, resample, lw, figsize,
ylabel, grayscale, fontname, subtitle, savefig, show)
    292 if resample:
    293     returns = returns.resample(resample)
--> 294     returns = returns.last() if compound is True else
returns.sum(axis=0)
    295     if isinstance(benchmark, _pd.Series):
    296         benchmark = benchmark.resample(resample)
```

```
File d:\Quantflow\QuantFlow v1.0\Day19\venv\Lib\site-
packages\pandas\core\resample.py:1183, in
Resampler.sum(self, numeric_only, min_count, *args,
**kwargs)
    1146 """
    1147 Compute sum of group values.
    1148
    (...) 1180 Freq: MS, dtype: int64
    1181 """
    1182 maybe_warn_args_and_kwargs(type(self), "sum",
args, kwargs)
-> 1183 nv.validate_resampler_func("sum", args, kwargs)
    1184 return self._downsample("sum",
numeric_only=numeric_only, min_count=min_count)
```

```
File d:\Quantflow\QuantFlow v1.0\Day19\venv\Lib\site-
packages\pandas\compat\numpy\function.py:376, in
validate_resampler_func(method, args, kwargs)
    374 if len(args) + len(kwargs) > 0:
    375     if method in RESAMPLER_NUMPY_OPS:
--> 376         raise UnsupportedFunctionCall(
    377             "numpy operations are not valid with
resample. "
    378             f"Use .resample(...){method}() instead"
    379         )
    380     raise TypeError("too many arguments passed in")
```

UnsupportedFunctionCall: numpy operations are not valid with resample. Use .resample(...).sum() instead

quantstats still sends an `axis=0` keyword into `Resampler.sum()`.

Pandas  $\geq 2.2$  raises `UnsupportedFunctionCall` if *any* keyword is passed to `sum()`, so you will see that error until one of these happens:

Fix	How	Pros	Cons
A. Downgrade pandas < 2.2	<code>pip install "pandas&lt;2.2"</code>	Nothing else to do	Locks you to an older pandas
B. Wait for a new QuantStats release	—	Clean & permanent	No ETA (last PyPI release is 2023)
C. Monkey-patch <code>Resampler.sum</code> once per session	4 lines of code below	Keep latest pandas & QuantStats	Must run at kernel start

Because you just upgraded pandas, C is the quickest path:

```
python

# ---- add these 4 lines ONCE, immediately after importing pandas -----
import pandas as pd
from pandas.core.resample import Resampler

if not getattr(Resampler, "_qs_axis_patch", False):
    _orig_sum = Resampler.sum
    Resampler.sum = lambda self, *a, **k: _orig_sum(self, *a) # drop axis kwarg
    Resampler._qs_axis_patch = True

# -----
```

Put that at the very top of your notebook/script—**before** any call to `qs.reports.html()`. It silently strips `axis=` if QuantStats passes it, so pandas  $\geq 2.2$  is happy.

## Minimal end-to-end example (copy-paste, runs on pandas 2.2+)

```
python
```

```

import pandas as pd
from pandas.core.resample import Resampler

# --- 1. one-time axis-kwarg patch -----
if not getattr(Resampler, "_qs_axis_patch", False):
    _orig_sum = Resampler.sum
    Resampler.sum = lambda self, *a, **k: _orig_sum(self, *a) # ignore axis
    Resampler._qs_axis_patch = True
# -----

import yfinance as yf, quantstats as qs

# 2. download data (MultiIndex by default)
data = yf.download("RELIANCE.NS", start="2018-01-01", progress=False)

# 3. pick Close/Adj Close robustly
lvl0 = data.columns.get_level_values(0)
close_col = "Adj Close" if "Adj Close" in lvl0 else "Close"
close = data.xs(close_col, level=0, axis=1).squeeze()

# 4. returns
returns = close.pct_change().dropna()

# 5. tear-sheet
qs.reports.html(
    returns,
    title="RELIANCE.NS Tearsheet",
    output="reliance_tearsheet.html"
)
print("✅ reliance_tearsheet.html generated")

```

Open `reliance_tearsheet.html` —you'll finally get the full QuantStats report with no `axis` / `ME` / `YE` errors, even on the newest pandas and GitHub QuantStats build.

Need to add a benchmark ( `^NSEI` ), create a PDF, or schedule daily tear-sheets? Tell me and I'll extend the snippet!

d:\Quantflow\QuantFlow v1.0\Day19\venv\Lib\site-packages\quantstats\plotting\core.py:1016:

FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method.

The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or 'df[col] = df[col].method(value)' instead, to perform the operation inplace on the original object.

```
port["Weekly"].ffill(inplace=True)
```

```
d:\Quantflow\QuantFlow v1.0\Day19\venv\Lib\site-packages\quantstats\_plotting\core.py:1019:
```

FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method.

The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or 'df[col] = df[col].method(value)' instead, to perform the operation inplace on the original object.

```
port["Monthly"].ffill(inplace=True)
```

```
d:\Quantflow\QuantFlow v1.0\Day19\venv\Lib\site-packages\quantstats\_plotting\core.py:1022:
```

FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method.

The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or 'df[col] = df[col].method(value)' instead, to

perform the operation inplace on the original object.

```
port["Quarterly"].ffill(inplace=True)
```

d:\Quantflow\QuantFlow v1.0\Day19\venv\Lib\site-packages\quantstats\\_plotting\core.py:1025:  
FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method.  
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or 'df[col] = df[col].method(value)' instead, to perform the operation inplace on the original object.

```
port["Yearly"].ffill(inplace=True)
```

Those messages are **just FutureWarnings** from pandas: QuantStats uses an “in-place” `.ffill(inplace=True)` idiom that will still work today, but pandas is letting library authors know the syntax will be removed in pandas 3.0.

Your tear-sheet has been generated successfully; nothing is broken.

If you prefer a clean run with **no noisy warnings** you have two simple options:

## 1 Hide only those specific warnings (recommended for notebooks / scripts)

Add this once, right after your imports:

```
python
```

```
import warnings
warnings.filterwarnings(
    "ignore",
    message="A value is trying to be set on a copy of a DataFrame or Series",
    category=FutureWarning,
```



```
module=r"quantstats\._plotting\._core"  
)
```

That keeps all other warnings visible but silences this pandas 3.0 deprecation notice.

## 2 Silence *all* FutureWarnings from QuantStats

```
python  
  
import warnings, re  
warnings.filterwarnings(  
    "ignore",  
    category=FutureWarning,  
    module=r"quantstats\..*" )
```

Use this if you don't want any pending-deprecation chatter from QuantStats.

### Nothing else you need to change

QuantStats works fine on pandas 2.2+ with the small patches we added.

When the library releases a new version, it will likely switch to the new syntax and these warnings will disappear.

Let me know if you'd like to:

- **Benchmark** RELIANCE.NS against **^NSEI**
- **Schedule** a daily tear-sheet run at 6 p.m. IST
- Output the report as **PDF** as well

I'm happy to extend the code for any of those!

it worked.

yes i would like to benchmark with ^NSEI

Great! Below is a drop-in replacement that:

- pulls **RELIANCE.NS** and **NIFTY 50 (^NSEI)**,
- converts each to daily returns,
- aligns the two return series on common dates,
- generates a tear-sheet that compares RELIANCE with the index.

python

```
import warnings, yfinance as yf, quantstats as qs, pandas as pd
from pandas.core.resample import Resampler

# — 1. one-time pandas-2.2 axis-kwarg patch (skip if already in your session) —
if not getattr(Resampler, "_qs_axis_patch", False):
    _orig_sum = Resampler.sum
    Resampler.sum = lambda self, *a, **k: _orig_sum(self, *a) # drop axis kwarg
    Resampler._qs_axis_patch = True

# —————

# (Optional) keep QuantStats future-warnings quiet
warnings.filterwarnings(
    "ignore",
    message="A value is trying to be set on a copy of a DataFrame or Series",
    category=FutureWarning,
    module=r"quantstats\._plotting\.core"
)

# — 2. download prices —————
tickers = ["RELIANCE.NS", "^NSEI"]
start_dt = "2018-01-01"

data = yf.download(tickers, start=start_dt, progress=False)

# Helper to grab Close/Adj Close even with MultiIndex
def get_close(df, ticker):
    lvl0 = df.columns.get_level_values(0)
    col0 = "Adj Close" if "Adj Close" in lvl0 else "Close"
    return df.xs(col0, level=0, axis=1)[ticker]

reliance_px = get_close(data, "RELIANCE.NS")
nsei_px = get_close(data, "^NSEI")

# — 3. convert to returns & align dates —————
```

```

reliance_ret = reliance_px.pct_change().dropna()
nsei_ret     = nsei_px.pct_change().dropna()

# inner-join so both series have identical index
reliance_ret, nsei_ret = reliance_ret.align(nsei_ret, join="inner")

# — 4. QuantStats tear-sheet with benchmark —————
qs.reports.html(
    returns      = reliance_ret,
    benchmark    = nsei_ret,
    title        = "RELIANCE.NS vs NIFTY 50 (NSEI)",
    output       = "reliance_vs_nsei_tearsheet.html"
)

print("✅ Saved reliance_vs_nsei_tearsheet.html")

```

## What you'll see in the tear-sheet

- All the usual RELIANCE metrics (Sharpe, drawdowns, rolling stats)
- Side-by-side & active-return charts versus ^NSEI
- Correlation table, scatter,  $\beta/\alpha$  estimates, etc.

Just open `reliance_vs_nsei_tearsheet.html` in any browser.

## Next steps?

- Want to **automate** this every day at 6 p.m. IST (and maybe email yourself the link)?
- Prefer a **PDF** or an **interactive Jupyter display** instead of HTML?
- Need the same benchmark layout for a list of stocks?

Let me know and I'll extend the workflow!

Thanks for the assistance. It worked

Happy to hear it worked! If you'd like to automate the tear-sheet generation, add more tickers, or explore other QuantStats features, just let me know anytime. Good luck with your analysis!