XGBoost Regressor

> screening and walk-forward optimisation for per minute data of the NIFTY 500

```
!pip install -q tqdm backtesting joblib
```

the TVDataFeed API

```
!pip install -q --upgrade --no-cache-dir git+https://github.com/rongardF/tvdatafeed.git
```

```
  Preparing metadata (setup.py) ... done
```

```
import pandas as pd
import numpy as np
import joblib
from tqdm import tqdm
from tvDatafeed import TvDatafeed, Interval
from backtesting import Backtest
import warnings
from xgboost import XGBRegressor
```

```
/usr/local/lib/python3.12/dist-packages/backtesting/_plotting.py:55: UserWarning: Jupyter Notebook detec
  warnings.warn('Jupyter Notebook detected. '
```

```
warnings.filterwarnings("ignore")
```

Screening

```
nifty_500 = pd.read_csv("/content/500List.csv").Symbol.to_list()
```

The Screening Function

> the idea is to find the companies suitable for a regression based predictive model (XGBRegressor)

```
tv = TvDatafeed()
```

```
WARNING:tvDatafeed.main:you are using nologin method, data you access may be limited
```

Deriving the globals

```
import pandas as pd
import numpy as np
import os

def fetch_daily(symbol):
    """
    fetches per day OHLCV data for a
    NSE symbol via TradingView and returns a dataframe
    """
    try:
      df = tv.get_hist(
          symbol=symbol,
          exchange='NSE',
          interval=Interval.in_daily,
          n_bars=1000
      )
      df = df[:int(0.7*len(df)) + 1]
      df.index = df.index.tz_localize("UTC").tz_convert("Asia/Kolkata")
      df.columns = [c.capitalize() for c in df.columns]

      if "Symbol" in df.columns:
```

```
            df.drop(columns=["Symbol"], inplace=True)

        df["ret"] = df["Close"].pct_change()
        return df.dropna()
      except:
        print(f"{symbol} not in TvDataFeed's database for the timeframe")

def compute_metrics(symbols):
      """
      computes: volatility, autocorrelation with 1-lag, median volume
      and stability
      """
      rows = []

      for sym in symbols:
          try:
              df = fetch_daily(sym)
              if len(df) < 700:
                  continue

              ret = df['ret']
              if len(ret) < 100:
                  continue

              vol = ret.std()
              ac1 = ret.autocorr(lag=1)

              # stability: CV of rolling vol
              rv = ret.rolling(200).std()
              stab = (rv.std() / rv.mean()) if rv.mean() > 0 else np.nan

              rows.append({
                  'symbol': sym,
                  'vol': vol,
                  'ac1': ac1,
                  'stab': stab,
              })


          except:
              print(f"couldn't compute metrics for {sym}")
              continue

      return pd.DataFrame(rows).dropna()

def derive_globals(metrics_df):
      """
      derives the percentiles based on which
      screening is done.
      """

      globals_out = {
          'VOL_LOW'  : metrics_df['vol'].quantile(0.40),
          'VOL_HIGH' : metrics_df['vol'].quantile(0.90),
          'STAB_MIN' : metrics_df['stab'].quantile(0.40),
          'AC_MIN'   : metrics_df['ac1'].quantile(0.60),
      }

      return globals_out


metrics_df = compute_metrics(nifty_500)
print("metrics_df:")
print(metrics_df.head())

globals_dict = derive_globals(metrics_df)
print("\nDerived Globals:")
for k, v in globals_dict.items():
    print(f"{k}: {v}")
```

```
ERROR:tvDatafeed.main:Connection timed out
ERROR:tvDatafeed.main:no data, please check the exchange and symbol
BAJAJ-AUTO not in TvDataFeed's database for the timeframe
couldn't compute metrics for BAJAJ-AUTO
ERROR:tvDatafeed.main:Connection timed out
```

```
ERROR:tvDatafeed.main:no data, please check the exchange and symbol
DUMMYSKFIN not in TvDataFeed's database for the timeframe
couldn't compute metrics for DUMMYSKFIN
ERROR:tvDatafeed.main:Connection timed out
ERROR:tvDatafeed.main:no data, please check the exchange and symbol
NAM-INDIA not in TvDataFeed's database for the timeframe
couldn't compute metrics for NAM-INDIA
metrics_df:
    symbol       vol       ac1      stab
0   360ONE  0.021382  0.033883  0.124928
1  3MINDIA  0.018275 -0.036718  0.119262
2      ABB  0.021096 -0.003184  0.125486
3      ACC  0.019766  0.030034  0.096149
4   AIAENG  0.017881  0.025922  0.103126

Derived Globals:
VOL_LOW: 0.020851031583696635
VOL_HIGH: 0.03146073828075781
STAB_MIN: 0.1278231933817813
AC_MIN: 0.0162186963271431
```

Master Screener

```
import numpy as np
import pandas as pd
from statsmodels.tsa.stattools import acf

def acf_strength(r, G):
    ac1 = r.autocorr(lag=1)
    return ac1 >= G["AC_MIN"]

def volatility_pass(std, G):
    return (G['VOL_LOW'] <= std <= G['VOL_HIGH'])

def stability_value(r, window=200):
    if len(r) < window * 2:
        return None
    rv = pd.Series(r).rolling(window).var().dropna()
    if len(rv) == 0:
        return None
    cv = rv.std() / rv.mean()
    return cv

def screen(symbol, df, G):
    try:
      r = df["ret"]
      if len(r) < 200:
          return None

      # volatility filter
      vol = r.std()
      if not volatility_pass(vol, G):
          return None

      # autocorrelation
      if not acf_strength(r, G):
          return None

      # stability
      stab_val = stability_value(r)
      if stab_val is None or stab_val > (1 / G['STAB_MIN']):
          return None

      # combine into score
      score = (
          (vol - G['VOL_LOW']) / (G['VOL_HIGH'] - G['VOL_LOW']) +
          (G['STAB_MIN'] / stab_val)
      )

      return (symbol, score)
    except:
      print("couldn't get the returns column")
```

```
    results = []

    for sym in tqdm(nifty_500):
        try:
            df = fetch_daily(sym)
            if len(df) < 700: continue
        except Exception as e:
            print(f"Error fetching {sym}: {e}")
            continue

        out = screen(sym, df, globals_dict)
        if out:
            print(f"Selected: {out}")
            results.append(out)

    ranked = sorted(results, key=lambda x: x[1], reverse=True)

    for sym, score in ranked[:10]:
        print(sym, score)
```

```
[('SCHNEIDER', np.float64(1.7146693589179844)),
 ('SAMMAANCAP', np.float64(1.6961317145393235)),
 ('IRB', np.float64(1.570547188403514)),
 ('INTELLECT', np.float64(1.5470799856630337)),
 ('LODHA', np.float64(1.524082438602976)),
 ('PGEL', np.float64(1.3994762971559245)),
 ('ZENTEC', np.float64(1.3847449447501279)),
 ('OLECTRA', np.float64(1.372713435058598)),
 ('TEJASNET', np.float64(1.3652006881918395)),
 ('GVT&D', np.float64(1.3646740891554108))]
```

```
    tickers = [x[0] for x in ranked[:20]]
    tickers
```

```
    ['SCHNEIDER',
     'SAMMAANCAP',
     'IRB',
     'INTELLECT',
     'LODHA',
     'PGEL',
     'ZENTEC',
     'OLECTRA',
     'TEJASNET',
     'GVT&D',
     'HSCL',
     'SAREGAMA',
     'GODFRYPHLP',
     'SCI',
     'FORCEMOT',
     'BDL',
     'POLICYBZR',
     'BSE',
     'BEML',
     'KEC']
```

Evaluation Function

```
    def evaluate_model(y_true, y_pred):
        """
        Compact evaluation suite for next-day return prediction.
        """
        import numpy as np
        from sklearn.metrics import (
            mean_absolute_error,
            mean_squared_error,
            r2_score
        )

        # core errors
```

```
    mae = mean_absolute_error(y_true, y_pred)
    mse = mean_squared_error(y_true, y_pred)
    rmse = np.sqrt(mse)
    r2 = r2_score(y_true, y_pred)

    # signal metrics
    corr = np.corrcoef(y_true, y_pred)[0, 1]
    true_dir = np.sign(y_true)
    pred_dir = np.sign(y_pred)
    da = (true_dir == pred_dir).mean()

    # tails
    thr = np.percentile(np.abs(y_true), 90)
    mask = np.abs(y_true) >= thr
    tail = (np.sign(y_true[mask]) == np.sign(y_pred[mask])).mean() if mask.any() else None

    # naive trading value
    strat = pred_dir * y_true
    sharpe = strat.mean() / strat.std() if strat.std() > 0 else None

    return {
        "MAE": mae,
        "MSE": mse,
        "RMSE": rmse,
        "R2": r2,
        "Correlation": corr,
        "Directional_Accuracy": da,
        "Tail_Hit_Rate": tail,
        "Naive_Strategy_Sharpe": sharpe
    }
```

Training, Testing, and Evaluating Separate Regressor per Ticker

```
def create_xgbregressor(symbol: str):
    import yfinance as yf
    import numpy as np
    import joblib
    from xgboost import XGBRegressor

    # fetch
    data = tv.get_hist(
          symbol=symbol,
          exchange='NSE',
          interval=Interval.in_daily,
          n_bars=1000
      )
    data.index = data.index.tz_localize('UTC')
    data.index = data.index.tz_convert('Asia/Kolkata')
    data.columns = [col.capitalize() for col in data.columns]

    # target: tomorrow's percent change
    data['change_tomorrow'] = data['Close'].pct_change().shift(-1)
    data = data.dropna()

    target = 'change_tomorrow'
    y = data[target]
    X = data.drop(columns=[target, 'Symbol'])

    # time-ordered split
    split = int(len(X) * 0.7) + 1
    X_train, X_test = X.iloc[:split], X.iloc[split:]
    y_train, y_test = y.iloc[:split], y.iloc[split:]

    model = XGBRegressor(
        min_child_weight=10,
        objective='reg:squarederror'
    )
    model.fit(X_train, y_train)

    # evaluation
    preds = model.predict(X_test)
    metrics = evaluate_model(y_test.values, preds)
```

```
    print(f"{symbol}: {metrics}")

    joblib.dump(model, f'/content/xgb/xgboost_model_{symbol.replace(".NS", "")}.joblib')
```

```
from tqdm import tqdm

for symbol in tickers:
  try:
    create_xgbregressor(f"{symbol}")
  except:
    continue
```

```
SCHNEIDER: {'MAE': 0.027224401165801507, 'MSE': 0.0011467513947927728, 'RMSE': np.float64(0.033863717970
SAMMAANCAP: {'MAE': 0.02733719753800366, 'MSE': 0.0012387626006803729, 'RMSE': np.float64(0.035196059448
IRB: {'MAE': 0.029438448335259187, 'MSE': 0.0014474045849039814, 'RMSE': np.float64(0.038044770795787185
INTELLECT: {'MAE': 0.025347998830751315, 'MSE': 0.0012414149113632996, 'RMSE': np.float64(0.035233718386
LODHA: {'MAE': 0.02171864887349911, 'MSE': 0.0008144450653653901, 'RMSE': np.float64(0.02853848393600105
PGEL: {'MAE': 0.03191374319679569, 'MSE': 0.0017882356964824676, 'RMSE': np.float64(0.0422875359471614),
ZENTEC: {'MAE': 0.029761767602969783, 'MSE': 0.0015465771587133519, 'RMSE': np.float64(0.039326545217109
OLECTRA: {'MAE': 0.024803741709467263, 'MSE': 0.0011058955420497794, 'RMSE': np.float64(0.03325500777401
TEJASNET: {'MAE': 0.025538334796837023, 'MSE': 0.0012471634301758413, 'RMSE': np.float64(0.0353152011204
GVT&D: {'MAE': 0.025408697101666256, 'MSE': 0.001103372958129024, 'RMSE': np.float64(0.03321705824014258
HSCL: {'MAE': 0.025954953681815017, 'MSE': 0.0010517158859750966, 'RMSE': np.float64(0.0324301693793772 3
SAREGAMA: {'MAE': 0.02146606354760206, 'MSE': 0.0009249043947311768, 'RMSE': np.float64(0.03041224086993
GODFRYPHLP: {'MAE': 0.0348822572574128, 'MSE': 0.0020222031273603204, 'RMSE': np.float64(0.0449689128994
SCI: {'MAE': 0.02694367997195792, 'MSE': 0.0012972084027279963, 'RMSE': np.float64(0.03601677946080127),
FORCEMOT: {'MAE': 0.028186947646920632, 'MSE': 0.00154026510042068, 'RMSE': np.float64(0.039246211287469
BDL: {'MAE': 0.02487984779122893, 'MSE': 0.0010142823520432323, 'RMSE': np.float64(0.031847799799094946)
POLICYBZR: {'MAE': 0.022230582431099575, 'MSE': 0.0008318002930864935, 'RMSE': np.float64(0.028840948200
BSE: {'MAE': 0.030607853954265303, 'MSE': 0.0016749057591493575, 'RMSE': np.float64(0.04092561250793148)
BEML: {'MAE': 0.026696400499348596, 'MSE': 0.0012407897401949658, 'RMSE': np.float64(0.03522484549568622
KEC: {'MAE': 0.024159492731879447, 'MSE': 0.0010420185977490408, 'RMSE': np.float64(0.032280312850854476
```

Testing Inference

```
model = joblib.load("/content/xgb/xgboost_model_POLICYBZR.joblib")
```

```
data = tv.get_hist(
          symbol='POLICYBZR',
          exchange='NSE',
          interval=Interval.in_daily,
          n_bars=100
      )
data.index = data.index.tz_localize('UTC')
data.drop(columns=['symbol'], inplace=True)
data.index = data.index.tz_convert('Asia/Kolkata')
len(data)
```

```
100
```

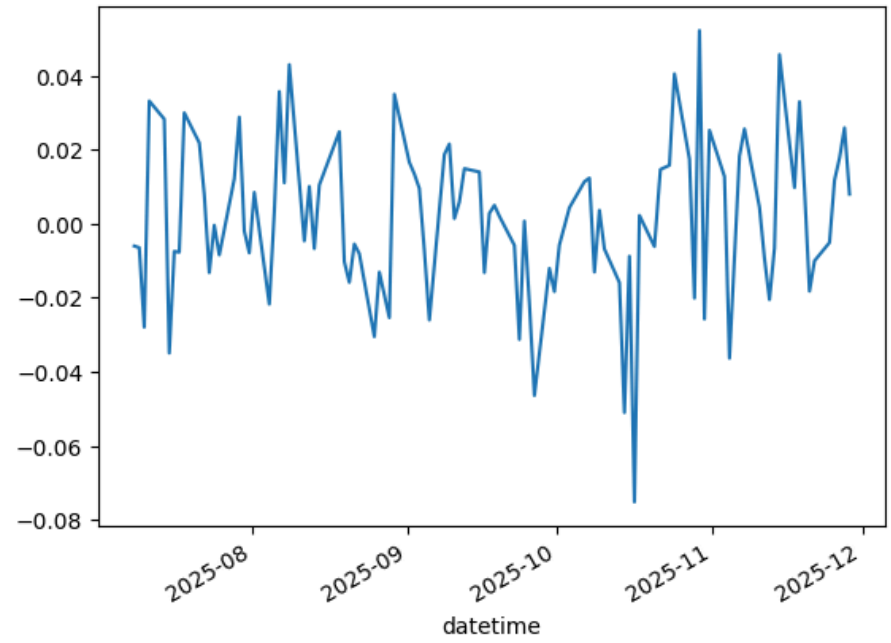
```
data['change tomorrow'] = data['close'].pct_change().shift(-1)
data.dropna(inplace=True)

target = 'change tomorrow'
y = data[target]
x = data.drop(columns=[target])

# Capitalize column names to match the loaded model's expected feature names
x.columns = [col.capitalize() for col in x.columns]
```

```
y_pred = model.predict(x)
```

```
error = y - y_pred
```

```
error.plot()
```

```
<Axes: xlabel='datetime'>
```




```
error2 = error ** 2
MSE = error2.mean()
RMSE = MSE ** 0.5
```

```
RMSE
```

```
np.float64(0.021511320445964648)
```

Backtesting

```
from backtesting import Strategy

class MlStrategy(Strategy):
  def init(self):
    self.model = model

  def next(self):
    x_today = self.data.df.iloc[[-1]]
    y_tomorrow = self.model.predict(x_today)

    if y_tomorrow > RMSE:
      self.buy()
    elif y_tomorrow < -RMSE:
      self.sell()
    else:
      pass
```

```
from backtesting import Backtest

bt = Backtest(
    x, MlStrategy, cash=1_00_000, commission=0.0002,
    exclusive_orders=True, trade_on_close=True
)

results = bt.run()
results
```

	0
Start	2025-07-08 09:15:00+05:30
End	2025-11-28 09:15:00+05:30
Duration	143 days 00:00:00
Exposure Time [%]	14.141414
Equity Final [$]	108266.17966
Equity Peak [$]	110142.37966
Commissions [$]	80.06654
Return [%]	8.26618
Buy & Hold Return [%]	-1.903786
Return (Ann.) [%]	22.405205
Volatility (Ann.) [%]	26.890805
CAGR [%]	15.022959
Sharpe Ratio	0.833192
Sortino Ratio	1.723546
Calmar Ratio	3.073013
Alpha [%]	9.24301
Beta	0.513099
Max. Drawdown [%]	-7.290957
Avg. Drawdown [%]	-4.427859
Max. Drawdown Duration	21 days 00:00:00
Avg. Drawdown Duration	12 days 00:00:00
# Trades	2
Win Rate [%]	50.0
Best Trade [%]	1.837867
Worst Trade [%]	-1.069877
Avg. Trade [%]	0.373466
Max. Trade Duration	18 days 00:00:00
Avg. Trade Duration	10 days 00:00:00
Profit Factor	1.71783
Expectancy [%]	0.383995
SQN	0.255443

Start coding or generate with AI.

_strategy	MlStrategy
_equity_curve	Equity Drawd...
_trades	Size EntryBar ExitBar EntryPrice ExitPr...

dtype: object


Equity
110%
105%
100%
Peak (110%)
Final (108%)
Max Drawdown (-7.3%)
Max Dd Dur. (21 days)
Profit / Loss
1%
0%
-1%
OHLC
Trades (2)
1950
1900
1850
1800
1750
1700
1650
Created with Backtesting.py: http://kernc.github.io/backtesting.py
Volume
5 m
0
07/2025
08/2025
09/2025
10/2025
11/2025