National College of Ireland

Configuration Manual

MSc Research Project
Data Analytics

Mohit Notani
Student ID: x23269651

School of Computing
National College of Ireland

Supervisor: Christian Horn

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: Mohit Notani

Student ID: x23269651

Programme: MSc in Data Analytics **Year:** 2024-2025

Module: Research Practicum

Lecturer: Christian Horn

Submission Due Date: 15/09/2025

Project Title: LLM-Powered Sentiment Analysis for Stock Price Prediction

Word Count: 1007 **Page Count:** 11

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Mohit Notani

Date: 11/08/2025

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Mohit Notani
Student ID: x23269651

1. Overview

This manual provides a step-by-step guide to set up the environment, obtain data, configure credentials, and execute the notebooks/scripts for a sentiment-augmented stock forecasting pipeline. The project integrates market data (Yahoo Finance), technical indicators, FinBERT-based news sentiment, and Facebook Prophet for next-day close price forecasts.

1.1 Scope

- Local workstation setup (Windows/macOS/Linux)
- Python virtual environment (Conda or venv)
- Dependencies from `requirements.txt`
- CmdStan toolchain for `prophet` via `cmdstanpy`
- Running the notebooks in order and reproducing results
- Troubleshooting common issues

1.2 Project Structure (expected)

```
project-root/
├─ data/
│  ├─ raw/              # downloaded price
│  └─ processed/        # feature tables, merged datasets
├─ notebooks/
│  ├─ 01_data_ingest.ipynb
│  ├─ 02_EDA.ipynb
│  ├─ 03_Stationarity_and_Differencing.ipynb
│  ├─ 04_prophet_baseline_updated.ipynb
│  ├─ 05_fetch_sentiments.ipynb
│  ├─ 06_Finbert.ipynb
│  └─ 07_prophet_tuned.ipynb
├─ merge.py
├─ requirements.txt
└─ README.md
```

2. System Requirements

2.1 Hardware (minimum)

- CPU: 4 cores (8+ recommended)

- RAM: 8–16 GB (≥16 GB recommended for faster NLP inference)
- Disk: 10 GB free (CmdStan build + caches + data/outputs)
- (Optional) NVIDIA GPU for PyTorch/Transformers acceleration

2.2 Operating System

- Windows 10/11, macOS 12+ (Intel/Apple Silicon), or Ubuntu 20.04+/other Linux

2.3 Python

- Python 3.13 recommended

2.4 OS-specific prerequisites

Windows:

- Install Microsoft C++ Build Tools (MSVC). Easiest via Visual Studio Build Tools installer.
- Ensure cl.exe is on PATH (open a fresh Developer Command Prompt or restart terminal).
- Optional: WSL2 + Ubuntu improves build reliability/perf for CmdStan.

macOS (Intel/Apple Silicon):

- Install Xcode Command Line Tools: xcode-select --install
- Homebrew recommended: brew install cmake pkg-config (useful for some optional libs)

Linux (Ubuntu/Debian):

- sudo apt-get update && sudo apt-get install -y build-essential make gcc g++ cmake
- If inside a container/VM, verify you have at least 4 GB RAM available during Stan build.

3. Software Prerequisites

3.1 Package Manager (choose one):

- Python venv + pip (lightweight)

3.2 Build Toolchain for CmdStan (required by Prophet via cmdstanpy):

- C/C++ toolchain must be present . Prophet compiles a small Stan model at first use.

3.3 Python Packages Required:

Item	Version
numpy	2.2.0
pandas	2.2.3
scikit-learn	1.15.1
statsmodels	0.14.47
arch	7.2.0
Prophet	1.1.7

cmdstanpy	1.2.5
holidays	0.56
matplotlip	3.10.5
seaborn	0.13.2
plotly	6.2.0
mplfinance	0.12.10b0
yfinance	0.2.65
ta	0.11.0
transformers	4.55.0
torch	2.6.0
requests	2.32.3
tqdm	4.66.4

4. Environment Setup

4.1 venv + pip

```
python3 -m venv .venv
source .venv/bin/activate        # Windows: .venv\Scriptsctivate
python -m pip install --upgrade pip wheel
pip install -r requirements.txt
python -c "import cmdstanpy as c; c.install_cmdstan()"
```

4.2 Quick Verification

```
# Run inside the activated environment
from prophet import Prophet
from transformers import pipeline
print ("Prophet OK, Transformers OK")
```

If this prints without errors, your core stack is ready.

4.3 Where CmdStan is installed

- cmdstanpy installs under your user cache (e.g., ~/.cmdstan on Unix; %USERPROFILE%/.cmdstan on Windows).
- Use from cmdstanpy import cmdstan_path; print(cmdstan_path()) to see the path.

5. Data Acquisition & Folders

Create folders up front (if not present):
mkdir -p data/raw data/processed outputs/figures outputs/metrics

- Price data: Retrieved via yfinance in 01_data_ingest.ipynb and saved under data/raw/.
- News data (optional): Retrieved via your chosen API (05_fetch_sentiments.ipynb) → data/raw/news_*.csv.
- Processed features: Created in 02_* / 03_* notebooks → data/processed/.

5.1 Data schema & naming conventions

Price CSV (per ticker or merged):

- Columns (typical yfinance): Date, Open, High, Low, Close, Adj Close, Volume, Ticker
- Primary keys: Date + Ticker

News raw CSV

- Columns: published_datetime, date, ticker, title, source, url (provider-dependent)

Daily sentiment CSV (aggregated):

- Columns: date, ticker, sent_pos, sent_neu, sent_neg, n_articles
- Aggregation: mean probabilities per day; keep n_articles for coverage context

Features CSV (stationary):

- Columns include: date, ticker, ret, sma20, sma50, ema20, rsi14, macd, macd_signal, macd_diff, bb_mid, bb_high, bb_low, bb_percB, bb_width, vol30, ...
- Lagged features are suffixed with _lag1 (e.g., ret_lag1) for T+1 forecasting

Joined modeling table:

- Merge on date, ticker: features + (optional) sentiment → used by Prophet notebooks

6. Execution Steps (Notebook Order)

Run the notebooks top-to-bottom unless instructed otherwise. Restart the kernel between notebooks if memory is constrained.

6.1 01_data_ingest.ipynb

- Purpose: Download OHLCV data for target tickers (e.g., AAPL, GOOGL, TSLA) using yfinance.
- Inputs: Ticker list, date range.
- Parameters to edit: TICKERS, START_DATE, END_DATE (top of notebook)

```python
# 2. Download 3 years of history
tickers = ["AAPL", "GOOGL", "TSLA"]
start = "2022-07-01"
end   = "2025-07-01"

raw = {}
for t in tickers:
    df = yf.download(t, start=start, end=end)
    df = df.drop(df.index[1])
    # now this will succeed
    df.to_csv(f"../data/raw/{t}.csv")
    raw[t] = df
df
```

✓ 1.2s Open 'df' in Data Wrangler Python

```
/var/folders/jq/mbgbv6211m9816f1py7zsb0c0000gn/T/ipykernel_25678/3561057797.py:8: FutureWarning: YF.download() has c
  df = yf.download(t, start=start, end=end)
[*********************100%***********************]  1 of 1 completed
/var/folders/jq/mbgbv6211m9816f1py7zsb0c0000gn/T/ipykernel_25678/3561057797.py:8: FutureWarning: YF.download() has c
  df = yf.download(t, start=start, end=end)
[*********************100%***********************]  1 of 1 completed
/var/folders/jq/mbgbv6211m9816f1py7zsb0c0000gn/T/ipykernel_25678/3561057797.py:8: FutureWarning: YF.download() has c
  df = yf.download(t, start=start, end=end)
[*********************100%***********************]  1 of 1 completed
```

Date	# ('Close', 'TSLA')	# ('High', 'TSLA')	# ('Low', 'TSLA')	# ('O
2022-07-01 00:00:00	227.26333618164062	230.22999572753906	222.1199951171875	
2022-07-06 00:00:00	231.73333740234375	234.56333923339844	227.18666076660156	
2022-07-07 00:00:00	244.5433349609375	245.3633270263672	232.2100067138672	
2022-07-08 00:00:00	250.76333618164062	254.97999572753906	241.16000366210938	
2022-07-11 00:00:00	234.3433380126953	253.06333923339844	233.6266632080078	
2022-07-12 00:00:00	233.07000732421875	239.77333068847656	228.3699951171875	
2022-07-13 00:00:00	237.0399932861328	242.05999755859375	225.03334045410156	
2022-07-14 00:00:00	238.31333923339844	238.65333557128906	229.3333282470703	
2022-07-15 00:00:00	240.06666564941406	243.6233367919922	236.88999938964844	
2022-07-18 00:00:00	240.54666137695312	250.51666259765625	239.60333251953125	

750 rows x 5 cols 10 per page Page 1 of 75

6.2 root/merge.py

- Purpose: Utility for merging intermediate datasets or producing final joined tables for modeling.
- Run: python merge.py

```python
def main(argv):
    # Accept file paths via CLI or default to ./data/raw/{AAPL,GOOGL,TSLA}.csv relative to script
    here = Path(__file__).parent
    if len(argv) > 1:
        files = [Path(a) for a in argv[1:]]
    else:
        raw_dir = here / "data" / "raw"
        files = [raw_dir / "AAPL.csv", raw_dir / "GOOGL.csv", raw_dir / "TSLA.csv"]
    frames = [load_custom_csv(f) for f in files]
    merged = pd.concat(frames, axis=0, ignore_index=True).sort_values(["Date", "Ticker"]).reset_ind
    out_csv = (here / "data" / "raw" / "merged_data.csv")
    out_csv.parent.mkdir(parents=True, exist_ok=True)
    merged.to_csv(out_csv, index=False)
    print(f"Wrote {out_csv} with {merged.shape[0]} rows and columns: {list(merged.columns)}")

if __name__ == "__main__":
    raise SystemExit(main(sys.argv))
```

6.3 02_EDA.ipynb

- Purpose: Explore data; compute technical indicators (ta): returns, SMA/EMA, RSI, MACD, Bollinger Bands, volatility.
- Inputs: Price CSVs from data/raw/.
- Outputs: feature tables to data/processed/.
- Notes: Ensure indicators have sufficient lookback (e.g., SMA50 needs ≥50 rows) before modeling.

```python
def add_technical_indicators(group):
    """Given df for one ticker, add SMA, EMA, MACD, RSI, Bollinger Bands."""
    close = group["Close"]
    vol   = group["Volume"]

    # Simple / Exponential MAs
    group["SMA20"] = close.rolling(20).mean()
    group["SMA50"] = close.rolling(50).mean()
    group["EMA20"] = close.ewm(span=20, adjust=False).mean()

    # MACD
    macd = ta.trend.MACD(close, window_slow=26, window_fast=12, window_sign=9)
    group["MACD"]     = macd.macd()
    group["MACD_sig"] = macd.macd_signal()
    group["MACD_diff"]= macd.macd_diff()

    # RSI
    rsi = ta.momentum.RSIIndicator(close, window=14)
    group["RSI14"] = rsi.rsi()

    # Bollinger Bands
    bb = ta.volatility.BollingerBands(close, window=20, window_dev=2)
    group["BB_mid"]   = bb.bollinger_mavg()
    group["BB_high"]  = bb.bollinger_hband()
    group["BB_low"]   = bb.bollinger_lband()
    group["BB_pct"]   = bb.bollinger_pband()    # % within bands
    group["BB_width"] = bb.bollinger_wband()    # width

    group["Roll10Std"] = group["Ret"].rolling(10).std()

    return group
# 5. Apply per-ticker
df = (
    df
    .groupby("Ticker", group_keys=False)
    .apply(lambda grp: add_technical_indicators(grp.copy()))
    .reset_index(drop=True)
)

df.head()
```

6.4 03_Stationarity_and_Differencing.ipynb

- Purpose: ADF/KPSS tests; create stationary features (e.g., returns); save cleaned feature set.

- Inputs: Feature tables from 02.
- Outputs: features_stationary.csv in data/processed/.
- Notes: Record ADF/KPSS p-values in outputs for auditability.

6.5 04_prophet_baseline_updated.ipynb

- Purpose: Baseline Prophet next-day forecasting (technical regressors only).
- Inputs: features_stationary.csv.
- Outputs: Baseline metrics (MAE/RMSE/MAPE/R^2)
- Common gotcha: Prophet expects columns named ds (date) and y (target). Rename accordingly.

```
tickers   = df["Ticker"].unique().tolist()
fcst_dict = {}
test_dict = {}
metrics   = []


n_test = 30  # last 30 rows → test

for sym in tickers:
    df_sym = df[df["Ticker"]==sym].sort_values("ds").reset_index(drop=True)

    train = df_sym.iloc[:-n_test]
    test  = df_sym.iloc[-n_test:].set_index("ds")
    test_dict[sym] = test

    # fit Prophet with all lagged regressors
    m = Prophet()
    for lag in lagged:
        m.add_regressor(lag)
    m.fit(train[["ds","y"] + lagged])

    # for forecasting, just feed in your test ds + lagged values
    future = test.reset_index()[["ds"] + lagged]
    fcst   = m.predict(future).set_index("ds")
    fcst_dict[sym] = fcst

    # compute metrics
    y_true = test["y"]
    y_pred = fcst.loc[test.index,"yhat"]
    mae  = mean_absolute_error(y_true, y_pred)
    rmse = np.sqrt(mean_squared_error(y_true, y_pred))
    mape = np.mean(np.abs((y_true - y_pred)/y_true))*100
    r2   = r2_score(y_true, y_pred)
    metrics.append({
        "Ticker": sym, "MAE": mae, "RMSE": rmse,
        "MAPE (%)": mape, "R²": r2
    })
# 6) display your performance table
perf = pd.DataFrame(metrics).set_index("Ticker")
print(perf)
```

```
02:13:07 - cmdstanpy - INFO - Chain [1] start processing
02:13:07 - cmdstanpy - INFO - Chain [1] done processing
02:13:07 - cmdstanpy - INFO - Chain [1] start processing
02:13:07 - cmdstanpy - INFO - Chain [1] done processing
02:13:07 - cmdstanpy - INFO - Chain [1] start processing
02:13:07 - cmdstanpy - INFO - Chain [1] done processing
              MAE       RMSE  MAPE (%)        R²
Ticker
TSLA    10.263390  14.382916  3.160008  0.310969
AAPL     2.900502   3.553370  1.449798 -0.064736
GOOGL    2.186484   2.762308  1.280864  0.585965
```

6.6 05_fetch_sentiments.ipynb

- Purpose: Pull headlines/news per ticker & date using NEWSAPI_KEY or POLYGON_API_KEY.
- Outputs: news_data_polygon.csv in data/processed/
- Tips: Respect API rate limits; implement basic retry with time.sleep on HTTP 429.
- **This piece of code takes hours to complete as we are using a free public API key.** Can skip this as the data is already saved in the data/processed folder.

```python
BASE_URL   = "https://api.polygon.io/v2/reference/news"
TICKERS    = ["AAPL","GOOGL","TSLA"]
START_DATE = date(2023, 6, 1)
END_DATE   = date(2025, 6, 1)
OUTPUT_CSV = "../data/processed/news_data_polygon.csv"
# ──────────────────────────────────────────────────────────────────────

records = []
delta = timedelta(days=1)

for single_day in tqdm(pd.date_range(START_DATE, END_DATE), desc="Days"):
    day_str = single_day.date().isoformat()  # "YYYY-MM-DD"
    for ticker in TICKERS:
        params = {
            "ticker": ticker,
            "published_utc.gte": day_str,
            "published_utc.lte": day_str,
            "limit": 100,
            "apiKey": API_KEY
        }
        resp = requests.get(BASE_URL, params=params)
        resp.raise_for_status()
        for news in resp.json().get("results", []):
            records.append({
                "ds": pd.to_datetime(news["published_utc"]).normalize(),
                "Ticker": ticker,
                "headline": news.get("title", news.get("description",""))
            })
        time.sleep(12)   # ~5 calls per minute => 12 s pause

# Save to disk
news_df = pd.DataFrame(records)
news_df.to_csv(OUTPUT_CSV, index=False)
print(f"\nFetched {len(news_df)} headlines over {news_df['ds'].nunique()} unique days.")
```

✓ 453m 57.6s Python

```
/Users/mohitnotani/Documents/Thesis/venv/lib/python3.13/site-packages/tqdm/auto.py:21: TqdmWarning: IProgress not fou
  from .autonotebook import tqdm as notebook_tqdm
Days: 100%|██████████| 732/732 [7:33:57<00:00, 37.21s/it]

Fetched 19114 headlines over 729 unique days.
```

6.7 06_Finbert.ipynb

- Purpose: Apply FinBERT to headlines; aggregate daily Positive/Neutral/Negative scores per ticker/date.
- Inputs: news_data_polygon.csv.
- Outputs: features_with_sentiment.csv in data/processed/.
- Performance: Use batch inference (e.g., pipeline(..., batch_size=16, truncation=True)).

```python
tokenizer = AutoTokenizer.from_pretrained("ProsusAI/finbert")
model     = AutoModelForSequenceClassification.from_pretrained("ProsusAI/finbert")
sent_pipe = pipeline(
    "sentiment-analysis",
    model= model,
    tokenizer= tokenizer,
    return_all_scores=True
)

# 3. Define a helper to extract scores
def extract_scores(text):
    scores_list = sent_pipe(text)[0]   # list of dicts: [{"label":"POS","score":...}, …]
    # map labels to lowercase keys
    return {d["label"].lower(): d["score"] for d in scores_list}

# 4. Run sentiment analysis (this may take a while if you have many rows)
sentiments = news_df["Text"].apply(extract_scores).apply(pd.Series)
news_df = pd.concat([news_df, sentiments], axis=1)

# 5. Aggregate to get a daily, per-ticker sentiment summary
daily_sent = (
    news_df
    .groupby(["Date","Ticker"])[["positive","neutral","negative"]]
    .mean()
    .reset_index()
)
daily_sent.head()
```

✓ 4m 25.0s Python

```
Device set to use mps:0
/Users/mohitnotani/Documents/Thesis/venv/lib/python3.13/site-packages/transformers/pipelines/text_classification.py:1
  warnings.warn(
```

	Date	Ticker	# positive	# neutral
0	2023-06-01 00:00:00+00:00	AAPL	0.2329257730394602	0.643316850904
1	2023-06-01 00:00:00+00:00	GOOGL	0.31981538349230376	0.497702577417
2	2023-06-01 00:00:00+00:00	TSLA	0.2929403575447698	0.520991614709
3	2023-06-02 00:00:00+00:00	AAPL	0.17121029094877568	0.651649680115
4	2023-06-02 00:00:00+00:00	GOOGL	0.16642198720946907	0.706851276196

6.8 07_prophet_tuned.ipynb

- Purpose: Grid-search changepoint_prior_scale & seasonality_prior_scale; add sentiment (features_with_sentiment.csv) as regressors; cross-validate and refit; evaluate on holdout.
- Suggested grid: CPS ∈ {0.001, 0.01, 0.05}, SPS ∈ {1, 10, 20}
- Cross-val: Horizon=30d; choose initial/period to yield ≥3 folds.

```
# 4) Hyperparameter grids
cp_scales = [0.001, 0.01, 0.05]
se_scales = [1, 10, 20]

# 4) Prep storage
train_dict, test_dict, fcst_dict = {}, {}, {}
results = []

# 5) Regressor lists
all_tech = tech_cols + lagged

# 6) Container for performance
results = []

# 7) Loop through tickers
for t in tickers:
    # Prepare train/test
    data  = df[df['Ticker']==t].sort_values('ds').reset_index(drop=True)
    train = data.iloc[:-30]
    test  = data.iloc[-30:]
    future = test[['ds'] + all_tech + (sentiment_cols if t=='TSLA' else [])]

    # Grid-search
    best_score, best_params = float('inf'), None
    for cps in cp_scales:
        for sps in se_scales:
            m = Prophet(changepoint_prior_scale=cps, seasonality_prior_scale=sps)
            # add regressors
            for feat in all_tech:
                m.add_regressor(feat)
            if t=='TSLA':
                for s in sentiment_cols:
                    m.add_regressor(s)
            m.fit(train[['ds','y'] + all_tech + (sentiment_cols if t=='TSLA' else [])])

            # CV on training set
            df_cv = cross_validation(m, horizon='30 days', period='15 days', initial='180 days', parallel="proces
            df_p  = performance_metrics(df_cv, rolling_window=0.1)
            mape  = df_p['mape'].mean()

            if mape < best_score:
                best_score, best_params = mape, (cps, sps)
```

```
    # Refit final model with best priors
    cps, sps = best_params
    m_final = Prophet(changepoint_prior_scale=cps, seasonality_prior_scale=sps)
    for feat in all_tech:
        m_final.add_regressor(feat)
    if t=='TSLA':
        for s in sentiment_cols:
            m_final.add_regressor(s)
    m_final.fit(train[['ds','y'] + all_tech + (sentiment_cols if t=='TSLA' else [])])

    # Forecast
    fc = m_final.predict(future)
    # store splits & forecast
    train_dict[t] = train.copy()
    test_dict[t]  = test.set_index('ds')
    fcst_dict[t]  = fc.set_index('ds')
    # Compute metrics
    y_true = test['y'].values
    y_pred = fc['yhat'].values
    mae    = mean_absolute_error(y_true, y_pred)
    rmse   = np.sqrt(mean_squared_error(y_true, y_pred))
    r2     = r2_score(y_true, y_pred)
    results.append({'Ticker':t,'MAE':mae,'RMSE':rmse,'R^2':r2,
                    'CPS':cps,'SPS':sps,'Best_MAPE':best_score})


# 8) Summary DataFrame
perf_df = pd.DataFrame(results).set_index('Ticker')
perf_df
```

✓ 1m 50.9s Python

```
print(perf_df)
```
✓ 0.0s Open 'perf_df' in Data Wrangler Python

Ticker	MAE	RMSE	R^2	CPS	SPS	Best_MAPE
AAPL	2.155552	2.309489	0.550228	0.01	1	0.022899
GOOGL	1.485572	1.823818	0.819509	0.01	10	0.024819
TSLA	7.625784	8.743947	0.745340	0.01	10	0.049152

6.9 Notebook parameters & customization

- Change tickers/dates: edit constants at top; ensure downstream merges still align.
- Add more lags: create _lag2, _lag3 features if experimenting with longer memory.
- Sentiment toggles: include/exclude sent_pos/neu/neg as Prophet regressors per ticker.
- Random seeds: set once (e.g., np.random.seed(42)) for reproducible splits.

6.10 Performance tips

- Close and reopen the kernel between heavy notebooks to free memory.
- Cache intermediate CSV/Parquet files; avoid recomputing indicators unnecessarily.
- On large headline sets, increase batch_size and consider GPU for Transformers.

References

Python 3. Available at: https://docs.python.org/3/.
Prophet. Available at: https://facebook.github.io/prophet/
The Library Source. Available at: https://pandas.pydata.org/.
Scikit-learn. Available at: https://scikit-learn.org/stable/
Finbert. Available at: https://huggingface.co/ProsusAI/finbert