FOREX TRADING PROJECT DOCUMENTATION

TABLE OF CONTENTS

1. INTRODUCTION

This documentation consolidates all project phases and final technical structures for a Random Forest–based automated Forex trading system. It highlights how data was sourced, how features were engineered, and how models were back tested and optimized (Azure ML).

2. PROJECT PHASES

2.1 PHASE 1: DECIDING HOW TO START

1. **Data Source Comparison**
   - Gathered historical data from both yfinance and MT5.
   - Found pricing discrepancies (broker-based differences, time zone alignments).
2. **Standardizing on MT5**
   - Selected **IC Markets** data via MT5 for all historical and future tests.
   - Ensured consistent timeframes and real-time data alignment.

2.2 PHASE 2: UPDATING THE SCRIPT

1. Script Enhancements
   - Implemented ML basic practices (data scaling, model serialization).
   - Leveraged **tsfresh** for feature generation (reducing manual indicator coding).
2. Pipeline Organization
   - Segregated data preprocessing, feature engineering, and modeling steps.
   - Divided **buy** and **sell** logic into separate scripts for clarity and easy maintenance.

2.3 PHASE 3: AZURE ML SETUP

1. **Hyperparameter Tuning Configuration**

- Created Azure ML experiments using random search and Bayesian optimization.
- Logged accuracy, precision, recall, and backtesting metrics in Azure.

2. Systematic Tuning

- Ran multiple parallel trials for each pair's buy/sell scenario.
- Identified optimal parameters for maximum precision.

2.4 PHASE 4: INITIAL BACKTESTING

1. Historical Evaluation

- Backtested first Random Forest iterations on MT5 historical data.
- Reviewed strategy profitability, max drawdown, risk metrics.

2. Performance Tuning

- Adjusted feature sets: RSI periods, EMA windows, rolling stats.
- Tested various risk/reward setups combinations of stoploss takeprofit.

2.5 PHASE 5: UPDATING MODELS

1. **Exploring Algorithms**

Compared LSTM, XGBoost, ADvanced XGBoost vs. Random Forest; found RF more stable in most tests.

Other models showed better results in training and testing across metrics other than precision too but when backtested the results were bad compared to Randomforest models selected.

2. Precision Focus

- Targeted **precision** to reduce false positives (unprofitable trades) under the assumption that it is not easy to find best model that literally solves the market. So Precision if over 5 splits shows positive returns than the models could be reliable.
- Tuned hyperparameters to improve precision without sacrificing trade frequency too drastically.

2.6 PHASE 6: ADDING FEATURES & FURTHER BACKTESTING

1. Additional Indicators for Model training

- Integrated Bollinger Bands, SAR, CCI, advanced rolling stats.
- Explored synergy of combined technical indicators.

2. Comprehensive Backtesting

- Ran extensive tests on “raw” vs. “filtered” backtesting.

3. UPDATED REPOSITORY STRUCTURE

FOREX_TRADING/

```
├── __pycache__/
├── .ipynb_checkpoints/
├── Archive MT5 and yfinance data initial models/
│   └── (Contains older notebooks/scripts for early broker comparisons)
├── AzureML Code and results/
│   └── (Azure ML hyperparameter tuning scripts & final results with more
│       indicator data)
├── Backtesting codebuild test 1/
├── Backtesting codebuild test 2/
├── Backtesting codebuild test 3/
│   └── (Incremental backtesting efforts with various code enhancements)
├── DemoAccountSetup Final Less indicator models/
│   └── (Scripts focusing on fewer indicators for demonstration or
│       simpler usage)
├── DemoScripts testing/
│   └── (Various demonstration scripts for prototyping new approaches)
├── Forex Documentation Last Phase/
│   ├── Backtesting/
│   ├── Models with more indicators/
│   └── Script/
│       └── (Consolidated or final-phase notebooks & reference docs)
├── TA_Lib-0.4.24-cp310-cp310-win_amd64(2).whl
├── LastMEETINGBacktestresults/
│   └── (backtestingcode and results discussed saved here)
├── NEW MODELS codebuilding test/
│   └── (Experimental or updated versions of models, new architecture tests)
├── Old ML_Models less indicators/
│   └── (Older approach with fewer indicators, previously used models)
```

4. AZUREML NEW TRAINING & MAIN SCRIPTS

Save Data for each currency pair with all the basic indicators and before labelling only with buy and sell flag columns added but not labelled.

Upload this csv file on Azure ML data simply to later use it in the scripts

Setup Workspace (ws)

This notebook helps in creating the workspace configuration. This is a onetime setup and the very first step in Azure setup to find the hyperparameters.

label_datafn.py

- Contains the `label_data` function for generating classification labels (win/lose) based on user-defined SL/TP thresholds. This function is imported on train scripts

train_script_sell.py

Contains the script for sell with a custom split of 50 60 70 80 90 percent training data and logs results.

train_script_buy.py

Contains the script for buy with a custom split of 50 60 70 80 90 percent training data and logs results.

mainscript_sellnew.ipynb

Run this notebook to find parameters for sell models

Change the Currency Pair

Set the parameters

Set the number of runs

Run the Job

Save the results to a csv check if all logged parameters and results are all being recorded in the csv

Mainscript_buynew.ipynb

Run this notebook to find parameters for buy models

Change the Currency Pair

Set the parameters

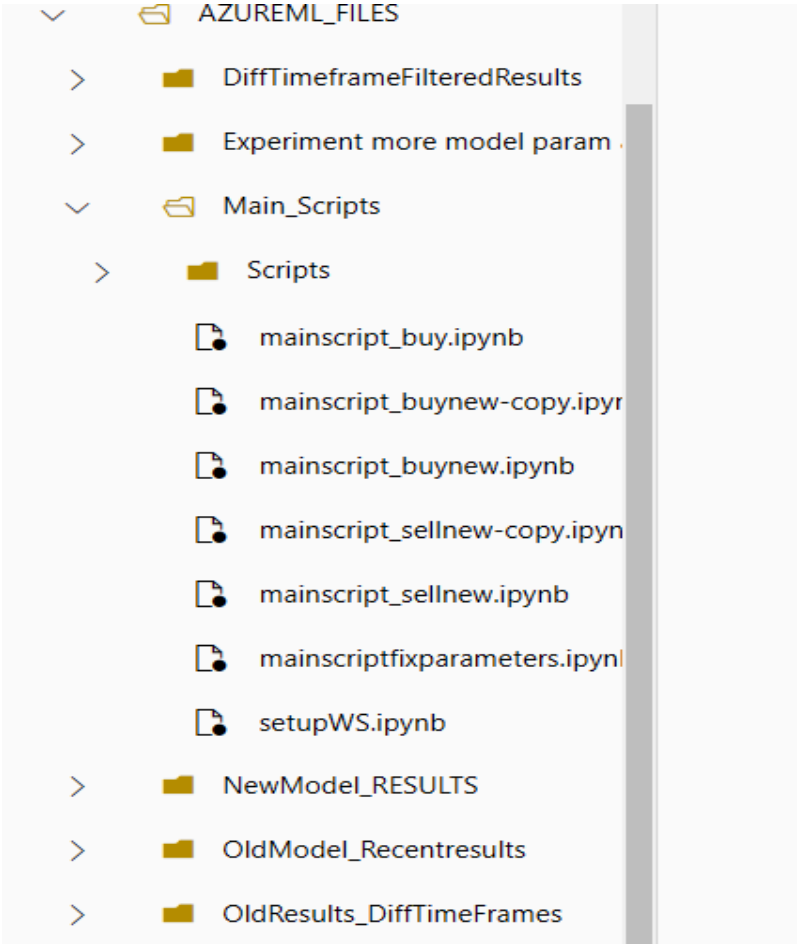
Set the number of runs

Run the Job

Save the results to a csv check if all logged parameters and results are all being recorded in the csv.

We use multiple notebooks (for both buy and sell scenarios) and multiple compute clusters to run hyperparameter tuning jobs in parallel, enabling faster experimentation and efficient resource usage.

Below is filestructure of Azure.



We run these notebooks on simple compute that is the compute main and we run the scripts on different compute clusters from this notebooks setup by adjusting the number of nodes required and computing power and cost

5. TECHNICAL COMPONENTS

Feature Engineering

- Incorporate **technical indicators** (RSI, EMA, MACD, SAR, Bollinger Bands, CCI) and **rolling statistics** (mean, std, lagged values).
- Optionally use **tsfresh** to automatically generate additional features and filter out low-values feature followed by filterselection based on target column.

Data Splits and Time-Series Considerations

Even though we primarily performed **time-series cross-validation** in Azure (which sequentially splits data to maintain chronological fidelity), we use a **static** 80/20 or 90/10 split **locally** for final training and evaluation. This split selection is based on the best-performing hyperparameters/approaches observed from the Azure experiments. By **keeping the chronological sequence in the split** (i.e., the first 80–90% of data for training, the last 10–20% for testing), we still mirror real-world trading conditions—no future data ever "leaks" into the training set. This ensures consistency with the time-series logic applied during Azure ML optimization, while also simplifying the final training/test framework.

Scaling & PCA

- Using **StandardScaler** on numeric features is standard practice for many ML models, especially when combining different types of indicators.
- **PCA** can help to reduce feature dimensionality or address multicollinearity.

Threshold Adjustment

- Adjusting the **probability threshold** (e.g., threshold = 0.6) can significantly affect precision and recall, which is crucial for trading strategies aiming to reduce false positives (unprofitable entries).

Model Evaluation

- In addition to classification metrics, use **backtesting metrics** (e.g., total returns, drawdown, Sharpe ratio) for a more holistic trading performance assessment.
- If using a "break-even" approach, define BreakEvenRatio to reflect the required precision to offset spreads, commissions, or risk-reward constraints.

6. Core Benefits

- **Scalable Tuning**: Efficiently run multiple parameter configurations in parallel.
- **Experiment Tracking**: Store runs, metrics, and logs centrally for easy reference.
- **Model Registry**: Register and version-control final models.

7. TYPICAL WORKFLOW

Create a Hyperparameter Job in Azure ML (via Web UI).

Find Parameters

Evaluate each trial's performance (precision, recall, total trades, true/false positives).

Retrieve Best Parameters, retrain the model locally, and save for algotrading.

Register/Deploy the final model for usage in trading scripts.

8. KEY TAKEAWAYS

- **Data-Driven Approach**
  - Extensive feature engineering (technical indicators, tsfresh).
  - SL/TP-based labeling to directly relate classification outcomes to profit/loss.
- **Robust Environment**
  - **Azure ML** for scalable hyperparameter tuning and experiment tracking.
  - **MT5** for reliable real-time data streams and stable trade execution.
- **Backtesting vs. Live Trading**
  - Backtests guide initial strategy design but cannot fully replicate live market conditions (spreads, slippage).
  - Continuous forward testing in demo/live environments is crucial.
  - Margin = 0.1 is a margin parameter of 1/10 (or 0.1) can imply **10× leverage**, meaning you only need to put down 10% of the total trade value.

- **Continuous Improvement**

  - Markets evolve; periodic retraining and hyperparameter updates are essential maybe in the intervals of 6 months?
  - Potential expansions include adding fundamental/news-based data or trying alternative modeling approaches.

9. FINAL NOTE

By combining **phased development**, **comprehensive feature engineering**, and **Azure ML optimization**—all standardized on MT5 data—this project delivers a **precision-focused Random Forest** pipeline ready for live Forex trading. Going forward, periodic **forward testing**, **performance monitoring**, and potential inclusion of fundamental or sentiment data will be vital for maintaining and improving long-term trading success.