

Final_Project_Report

November 29, 2025

1 Final Project Report: Stock Market Prediction Pipeline

Workshop in Data Science — Team 003 (Itay, Moran, Shaked)

This notebook serves as the primary research report and documentation for our stock prediction pipeline. It demonstrates the methodology, design decisions, and performance evaluation of our models.

Note: Heavy computational logic and reusable components are implemented in the `src/` directory to maintain a clean and narrative-driven notebook.

```
[ ]: # --- Global Imports ---
import sys
import platform
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# --- Project Modules ---
from src.data.loader import fetch_sample_data
from src.evaluation.analysis import check_stationarity, plot_price_vs_returns
from src.models.baselines import NaiveBaseline, RandomBaseline
from src.evaluation.metrics import evaluate_regression, print_eval
from src.evaluation.plots import set_style
from src.features.preprocessing import LogReturnTransformer

# --- Configuration ---
%matplotlib inline
set_style() # Apply consistent academic plotting style

print(f"Python Version: {platform.python_version()}")
print("Environment initialized successfully.")
```

1.1 Pipeline Overview

The project follows a structured data science lifecycle:

1. Business Understanding & Problem Formulation (Current Stage)
2. Data Access & Ingestion
3. Feature Engineering (Transformers)

4. Preprocessing & Windowing
5. Modeling (Baselines -> ML -> DL)
6. Evaluation & Backtesting
7. Explainability

1.2 1. Business Understanding & Problem Formulation

1.2.1 1.1 Objective

The primary objective is to develop a machine learning pipeline capable of predicting short-term movements in the stock market, specifically focusing on **Apple Inc. (AAPL)**. Unlike traditional forecasting which often targets raw prices, our goal is to predict the **Logarithmic Returns** of the next trading day ($t + 1$).

1.2.2 1.2 Target Variable Selection: The Stationarity Challenge

Financial time series are notoriously **non-stationary**, meaning their statistical properties (mean, variance) change over time. Standard regression models (e.g., Linear Regression) rely on the assumption of stationarity to generalize effectively.

To address this, we transform our target variable from Raw Price (P_t) to Log>Returns (Y_t):

$$Y_t = \ln(P_{t+1}) - \ln(P_t) = \ln\left(\frac{P_{t+1}}{P_t}\right)$$

Why Log>Returns?

1. **Stationarity:** They exhibit more stable statistical properties than raw prices.
2. **Additivity:** Log-returns are time-additive, simplifying multi-period analysis.
3. **Normality:** They often approximate a normal distribution better than simple percentage returns.

```
[ ]: # --- Empirical Stationarity Analysis ---

# 1. Load Sample Data (AAPL)
df_research = fetch_sample_data("AAPL", period="2y")

# 2. Compute Target (Log Returns) using our Transformer
log_transformer = LogReturnTransformer()
df_research = log_transformer.transform(df_research)
df_research.dropna(inplace=True)

# 3. Visual & Statistical Comparison
plot_price_vs_returns(df_research, 'Log_Returns')

# 4. Statistical Proof (ADF Test)
check_stationarity(df_research['Close'], "Raw Close Price")
check_stationarity(df_research['Log_Returns'], "Log Returns")
```

1.2.3 1.3 Evaluation Strategy

We employ a dual-metric evaluation strategy to balance statistical rigor with financial reality.

1. Statistical Metric: MSE (Mean Squared Error)

$$MSE = \frac{1}{N} \sum_{i=1}^N (y_i - \hat{y}_i)^2$$

Rationale: Penalizes large errors heavily, which is crucial in finance where a single large prediction error can be catastrophic.

2. Financial Metric: Sharpe Ratio

$$Sharpe = \frac{E[R_p] - R_f}{\sigma_p} \times \sqrt{252}$$

Rationale: Measures risk-adjusted return. A model with low MSE might still lose money if it misses significant market moves or mispredicts direction. The Sharpe ratio ensures the strategy justifies the risk taken.

3. Trading Metric: Directional Accuracy (DA)

$$DA = \frac{1}{N} \sum_{i=1}^N \mathbb{1}_{\text{sign}(y_i) == \text{sign}(\hat{y}_i)}$$

Rationale: In many trading strategies, correctly predicting the *direction* (Long/Short) is more critical than the exact magnitude.

1.2.4 1.4 Validation Strategy: Strict Walk-Forward

To prevent **Look-Ahead Bias**, we strictly use `TimeSeriesSplit` (expanding window) and avoid random shuffling. Future data must never leak into the training set.

1.3 2. Baseline Performance

Before building complex models, we establish simple baselines to serve as performance benchmarks. Any advanced model must significantly outperform these to be considered valuable.

1. **Naive Baseline (Zero Strategy):** Predicts no change ($y_{t+1} = 0$). This is a strong baseline for log-returns due to the Martingale property of efficient markets.
2. **Random Baseline:** Predicts random values drawn from a normal distribution matching the training data's μ and σ .

```
[ ]: # --- Baseline Evaluation ---

# 1. Define Baselines
naive_model = NaiveBaseline(strategy="zero")
random_model = RandomBaseline(seed=42)

# 2. Fit/Predict (Using the research data for demonstration)
# Note: In the full pipeline, this will be done via proper Train/Test splits.
y_true = df_research['Log>Returns']
random_model.fit(y_true)
```

```

y_pred_naive = naive_model.predict(df_research)
y_pred_random = random_model.predict(df_research)

# 3. Evaluate & Report
metrics_naive = evaluate_regression(y_true, y_pred_naive)
metrics_random = evaluate_regression(y_true, y_pred_random)

print_eval(metrics_naive, "Naive Baseline (Zero)")
print_eval(metrics_random, "Random Baseline")

```

1.4 3. Future Work (Stages 3-8)

The following sections outline the roadmap for the remainder of the project. Implementations will be added iteratively.

1.4.1 3. Data Access

- Implementation of robust data contracts and caching mechanisms.

1.4.2 4. Transformers

- Calculation of Technical Indicators (RSI, MACD, Bollinger Bands).

1.4.3 5. Preprocessing

- Time alignment, windowing (lookback sequences), and normalization.

1.4.4 6. Modeling

- Development of Machine Learning (XGBoost, LightGBM) and Deep Learning (LSTM, Transformer) models.

1.4.5 7. Evaluation

- Comprehensive backtesting and error analysis.

1.4.6 8. Explainability

- SHAP value analysis to understand feature importance.