Market Mood & Moves
Sentiment-Driven Stock Prediction

Week 1

Mentors: Meet & Sarthak

December 2025

Welcome to the Project

Welcome to Market Mood & Moves! This Week 1 guide establishes the foundational knowledge required for building a sentiment-driven stock prediction system.
The project integrates three key domains: **Data Science**, **Natural Language Processing**, and **Financial Engineering**. Week 1 focuses on understanding core concepts, setting up your development environment, and mastering essential tools.
This guide includes both fundamental topics and some Challenges that introduce industry-standard practices used in production systems.

Contents

1 Project Overview

1.1 Introduction

Welcome to this project! This project builds a system that captures market sentiment from news articles and social media, then uses this information to predict stock price movements.

To understand why we are building this, let's think of the stock market not as a cold spreadsheet, but as a bustling stadium where fans (investors) cheer or boo based on the latest play (news). This section sets the stage by explaining why emotions matter in trading and sketching our project's blueprint.

1.2 The Behavioral Finance Thesis

Imagine the stock market is a giant popularity contest. The traditional theory, known as the **Efficient Market Hypothesis**, says prices are always "correct" because everyone knows everything. But **Behavioral Finance** says, "Hold on!" Humans aren't robots. We get excited, we get scared, and we make mistakes.

The Core Philosophy

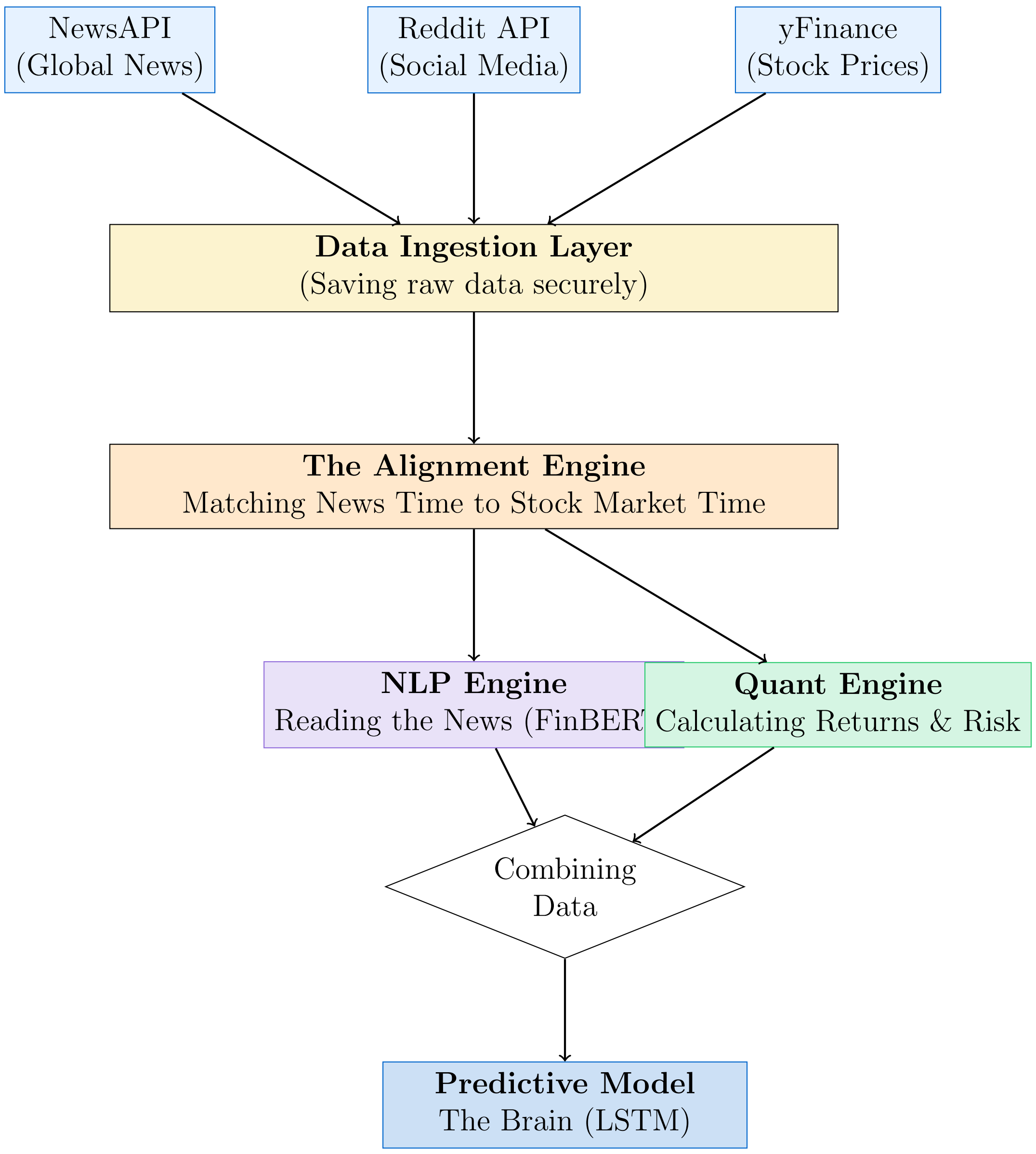
"The market is a voting machine in the short run, but a weighing machine in the long run." – Benjamin Graham.
In the short term, the market is like a voting machine—it counts how many people like or dislike a stock. Our project targets this "voting" phase. By measuring the "mood" on Reddit and in the news, we try to spot when the price doesn't match the mood.

Real-Life Example: GameStop Mania

In 2021, GameStop (GME) stock rocketed 1,500% not because the company was suddenly making huge profits, but because people on Reddit were hyping it up. The "sentiment" (mood) was euphoric, even though the business hadn't changed much. Our tool is designed to spot this kind of excitement early.

1.3 System Architecture

Picture a factory assembly line. News and Reddit chatter enter one side; stock prices enter the other. They move down the line, get synchronized, processed, and finally, a "Buy" or "Sell" signal comes out the end.


NewsAPI (Global News)
Reddit API (Social Media)
yFinance (Stock Prices)
Data Ingestion Layer (Saving raw data securely)
The Alignment Engine Matching News Time to Stock Market Time
NLP Engine Reading the News (FinBERT)
Quant Engine Calculating Returns & Risk
Combining Data
Predictive Model The Brain (LSTM)

1.4 Project Workflow

The detailed steps we will follow to build this engine are:

1. **Data Collection:** Acquire news articles and social media posts about target companies using APIs.
2. **Sentiment Analysis:** Apply Natural Language Processing models (like FinBERT) to extract positive or negative signals.
3. **Data Integration:** Combine sentiment data with historical stock prices, ensuring time-zones match.
4. **Model Training:** Develop neural networks (LSTM) that can learn patterns over time.
5. **Prediction:** Forecast whether the stock will go up or down tomorrow.
6. **Strategy Development:** Create a trading strategy and "backtest" it (test it on past data) to see if it would have made money.

1.5 Learning Objectives

By the end of this project, you will:

- Understand how behavioral finance and human psychology drive market movements.
- Master data collection through Application Programming Interfaces (APIs).
- Apply Natural Language Processing techniques to real-world financial text.
- Implement machine learning models for time series prediction.
- Develop production-ready data pipelines that are robust and scalable.

1.6 Key Resources

- Market Psychology and Sentiment Analysis
- Impact of Behavioral Finance on Markets

2 Python Programming Fundamentals

2.1 Overview

Python serves as the primary language for this project. This section reviews essential programming concepts including loops, conditionals, functions, and data structures.

2.2 Learning Objectives

– Review control flow structures (`for`, `while`, `if/else`)

– Understand data structures (lists, dictionaries, sets, tuples)

– Write modular functions

– Handle file operations and exceptions

2.3 Key Resources

- Python Official Tutorial
- W3Schools Python Course
- Python Basics Video Tutorial

2.4 Practical Exercises

- Write a function to calculate the average of a list of stock prices
- Create a function that classifies stocks as expensive (¿150) or affordable (¡150)
- Implement a loop that computes daily returns over a 10-day period

3 Data Manipulation with Pandas

3.1 Overview

Pandas provides efficient data structures and operations for manipulating numerical tables and time series data. It is essential for data preprocessing and analysis.

3.2 Learning Objectives

– Load and explore CSV files using DataFrames

– Filter rows based on conditions

– Create derived columns (e.g., daily returns)

– Perform aggregations (mean, sum, count)

– Handle missing data

3.3 Key Resources

- 10 Minutes to Pandas
- GeeksforGeeks Pandas Tutorial
- Pandas in 20 Minutes

3.4 Practical Exercises

- Load a stock price CSV file
- Filter for days where sentiment exceeds 0.7
- Create a column for daily returns: $\text{Return}_t = \text{Price}_t - \text{Price}_{t-1}$
- Calculate the weekly average closing price

Note: Refer to the provided Jupyter notebook `pandas_tutorial.ipynb` for executable examples.

4 Numerical Computing with NumPy

4.1 Overview

NumPy provides support for large, multi-dimensional arrays and matrices, along with mathematical functions to operate on these arrays efficiently.

4.2 Learning Objectives

- Create and manipulate n-dimensional arrays
- Perform element-wise operations
- Calculate statistical measures (mean, standard deviation, percentiles)
- Index and slice arrays

4.3 Key Resources

- NumPy for Absolute Beginners
- W3Schools NumPy Tutorial
- NumPy Crash Course

4.4 Practical Exercises

- Create an array of 10 stock prices
- Calculate mean, standard deviation, minimum, and maximum
- Create a 2D array representing prices for 5 days across 3 stocks
- Calculate daily returns using `np.diff()`

Note: Refer to `numpy_tutorial.ipynb` for hands-on examples.

5 Natural Language Processing Fundamentals

5.1 Overview

Natural Language Processing enables computers to understand and process human language. This project uses NLP to extract sentiment from news articles and social media posts.

5.2 Learning Objectives

– Understand NLP applications in finance

– Distinguish between text preprocessing and sentiment analysis

– Explore the NLTK library's capabilities

5.3 Key Resources

- GeeksforGeeks NLP Tutorial
- Natural Language Toolkit Book
- NLP Basics Video

5.4 Setup

```
pip install nltk
python -c "import nltk; nltk.download('all')"
```

6 Text Preprocessing

6.1 Tokenization

Tokenization divides text into individual tokens (typically words or subwords). This is the first step in text preprocessing.

Learning Objectives:

- Understand tokenization methods
- Apply word and sentence tokenization
- Handle punctuation appropriately

Example:

```
from nltk.tokenize import word_tokenize
text = "Apple stock surges 5% on strong earnings!"
tokens = word_tokenize(text)
# Result: ['Apple', 'stock', 'surges', '5', '%', 'on', 'strong', 'earnings', '!']
```

6.2 Stop Words Removal

Stop words are common words (e.g., "the", "is", "and") that typically don't contribute meaningful information for sentiment analysis.

Learning Objectives:

- Identify stop words
- Remove stop words from text
- Understand when to preserve stop words

Example:

```
from nltk.corpus import stopwords
stop_words = set(stopwords.words('english'))
filtered = [w for w in tokens if w.lower() not in stop_words]
```

6.3 Lemmatization

Lemmatization reduces words to their dictionary form (lemma). For example, "running", "runs", and "ran" all reduce to "run".

Learning Objectives:

- Understand lemmatization vs. stemming
- Apply lemmatization to financial text
- Preserve word meaning during normalization

Example:

```
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
lemmatized = [lemmatizer.lemmatize(w, pos='v') for w in words]
```

7 API-Based Data Collection

7.1 NewsAPI Integration

NewsAPI provides programmatic access to news articles from various sources.

Setup Instructions:

1. Register at newsapi.org
2. Obtain API key (free tier: 100 requests/day)
3. Install Python client: `pip install newsapi-python`

Example Usage:

```
from newsapi import NewsApiClient
api = NewsApiClient(api_key='YOUR_API_KEY')
headlines = api.get_top_headlines(
    category='business',
    language='en',
    page_size=10
)
```

Security Notice: Never commit API keys to version control. Use environment variables or configuration files excluded from Git.

7.2 Stock Data via yfinance

yfinance provides access to historical and real-time stock market data from Yahoo Finance.

Installation:

```
pip install yfinance
```

Example Usage:

```
import yfinance as yf
data = yf.download('AAPL', start='2024-01-01', end='2024-12-31')
print(data.head())
```

8 Sentiment Analysis

8.1 Introduction to Sentiment Analysis

Sentiment analysis classifies text as positive, negative, or neutral. In financial contexts, sentiment can indicate market psychology and predict price movements.

8.2 VADER Sentiment Analyzer

VADER (Valence Aware Dictionary and sEntiment Reasoner) is a lexicon and rule-based sentiment analysis tool.

Example:

```
from nltk.sentiment.vader import SentimentIntensityAnalyzer
sia = SentimentIntensityAnalyzer()
score = sia.polarity_scores("Apple stock surges on strong earnings!")
# Returns: {'neg': 0.0, 'neu': 0.508, 'pos': 0.492, 'compound': 0.6369}
```

8.3 FinBERT for Financial Text

FinBERT is a transformer-based model fine-tuned specifically for financial sentiment analysis. It significantly outperforms general-purpose sentiment analyzers on financial text.

Installation:

```
pip install transformers torch
```

Example:

```
from transformers import BertTokenizer, BertForSequenceClassification
import torch

tokenizer = BertTokenizer.from_pretrained('ProsusAI/finbert')
model = BertForSequenceClassification.from_pretrained('ProsusAI/finbert')

inputs = tokenizer("Apple reports record profits", return_tensors='pt')
outputs = model(**inputs)
probs = torch.softmax(outputs.logits, dim=1)
```

Note: Week 1 focuses on using FinBERT as a tool. The underlying transformer architecture will be covered in Week 3.

9 Behavioral Finance Concepts

9.1 Overview

Behavioral finance studies how psychological factors influence financial decisions and market outcomes. Understanding these concepts is crucial for interpreting sentiment data.

9.2 Key Concepts

- **Herd Mentality:** Tendency to follow the crowd, leading to bubbles and crashes
- **Loss Aversion:** People fear losses more than they value equivalent gains
- **Confirmation Bias:** Seeking information that confirms existing beliefs
- **Overconfidence:** Overestimating one's ability to predict market movements

9.3 Resources

- Market Psychology and Sentiment
- Investopedia: Behavioral Finance

10 Quantitative Finance Fundamentals

Trading isn't just guessing directions—it's like driving: You need to factor in gas costs, traffic jams, and if the scenic route's worth the detours. This chapter breaks down those "hidden fees" and scorecards, starting with how trades actually happen, then how to grade your strategy's performance, and finally how to test it without risk.

10.1 Market Microstructure

Market microstructure is the "engine room" of trading—how individual buys and sells nudge prices. Understanding this ensures our simulations feel real, not like a video game on easy mode.

10.1.1 Slippage Modeling

Slippage happens when your trade doesn't hit the "perfect" price you see on the screen. It stems from delays, large orders, or market volatility, making execution prices "slip" away from quoted prices.

The mathematical model for execution price is:

$$P_{execution} = P_{mid} \pm \left(\frac{\text{Spread}}{2} + \sigma_{market} \times \text{Impact Factor} \right) \tag{1}$$

For this project, we will use a conservative estimate:

- **Static Slippage:** We assume a 0.05% penalty on every trade.
- **Dynamic Slippage:** Larger orders incur higher penalties based on average volume.

Real-Life Example: Buying Bitcoin in a Frenzy

During the 2021 crypto boom, a \$50,000 BTC buy might execute at \$50,250 due to slippage—0.5% extra from the rush! Our model adds this "surge fee" to tests, so backtests don't overpromise.

10.1.2 Transaction Costs Analysis (TCA)

These are the fees that eat into profits.

- **Explicit Costs:** Direct fees like broker commissions and exchange fees.
- **Total Cost Formula:** $Cost = \text{Fixed Fee} + (\text{Volume} \times \text{Rate}) + \text{Taxes}$.

Real-Life Example: Day Trading vs. Long Hold

A day trader flipping Apple 10x/day forks over \$50 in fees (erasing tiny wins), while a monthly holder pays once (\$5). Our approach mimics the holder for cost control.

10.1.3 Liquidity & Trade Execution Probability

Liquidity refers to how easily an asset can be bought or sold without affecting its price.

- **Limit Orders:** You set a specific price to buy/sell. It guarantees the price but not the execution (the order might not be filled).
- **Market Orders:** You buy/sell at the current best available price. It guarantees execution but not the price (subject to slippage).

Real-Life Example: Penny Stocks vs. Apple

Snapping 1,000 penny shares? Price jumps 5% from scarcity. Apple? A whisper. We scan volume first—low? Skip or chop orders.

10.2 Performance Metrics

Accuracy isn't enough. We need to evaluate the risk and reward profile of a strategy.

10.2.1 Maximum Drawdown (MDD)

This measures the largest percentage drop from a peak to a trough during a specific period. It quantifies the "worst-case scenario" loss an investor would have experienced.

$$MDD = \min\left(\frac{P_t - \max(P_{0\ldots t})}{\max(P_{0\ldots t})}\right) \tag{2}$$

10.2.2 Compound Annual Growth Rate (CAGR)

The mean annual growth rate of an investment over a specified period of time longer than one year. It smooths out the volatility of periodic returns.

$$CAGR = \left(\frac{P_{final}}{P_{initial}}\right)^{\frac{1}{n}} - 1 \tag{3}$$

10.2.3 Win/Loss Ratio

The ratio of the total number of winning trades to the total number of losing trades. It helps assess the consistency of a strategy.

10.2.4 Profit Factor

The ratio of gross profit to gross loss. A profit factor greater than 1.0 indicates a profitable system.

10.2.5 Alpha

A measure of the active return on an investment, the performance of that investment compared to a suitable market index. It represents the value added by the trader's skill.

10.2.6 Beta

A measure of the volatility—or systematic risk—of a security or portfolio compared to the market as a whole. A beta of 1 indicates the asset moves with the market.

10.2.7 Information Ratio

A measurement of portfolio returns beyond the returns of a benchmark, usually an index, compared to the volatility of those returns.

10.2.8 Sharpe Ratio

The Sharpe Ratio measures risk-adjusted return. It tells us how much excess return we are getting for the extra volatility we endure.

$$\text{Sharpe} = \frac{\mathbb{E}[R_p - R_f]}{\sqrt{\text{Var}[R_p - R_f]}} \tag{4}$$

Real-Life Example: Stocks vs. Savings

Consider two investment options to understand this better:

- **Option 1: The Bank Savings Account.** You get a guaranteed 2% return. It is perfectly safe, but the reward is very low. Since there is no risk, the Sharpe Ratio is near 0.
- **Option 2: The Stock Market (S&P 500).** You might get a 10% return on average, but the price swings wildly (15% volatility). You have to endure a lot of stress for that profit. The Sharpe Ratio is around 0.5.

Our Goal: We aim for a Sharpe Ratio greater than 1.5. This means achieving high returns like the stock market, but with a much smoother, less stressful ride—like upgrading from a bumpy wooden rollercoaster to a modern high-speed train.

10.2.9 Sortino Ratio

Similar to the Sharpe Ratio, but it only penalizes downside volatility (harmful volatility), rather than total volatility.

$$\text{Sortino} = \frac{\mathbb{E}[R_p - R_f]}{\sqrt{\text{Var}[R_p - R_f | R_p - R_f < 0]}} \tag{5}$$

10.2.10 Calmar Ratio

A comparison of the average annual compounded rate of return and the maximum drawdown risk of commodity trading advisors and hedge funds.

$$\text{Calmar} = \frac{\text{CAGR}}{\text{MDD}} \tag{6}$$

Real-Life Example: 2022 Bear Market

Let's look at the 2022 Bear Market to understand why the Calmar Ratio matters.

- **The Market Reality:** In 2022, the S&P 500 dropped by about 25%. That represents a massive Maximum Drawdown (MDD)—a financial nightmare where investors saw a quarter of their portfolio vanish.
- **The Strategy:** Imagine you had a trading strategy that made a modest 10% profit per year (CAGR), but its worst drop was only 15% (MDD).
- **The Result:** The Calmar Ratio would be 10/15 = 0.67. While 0.67 might seem like a small number, in a year where the general market crashed, this strategy acted like a reliable raincoat in a thunderstorm. It kept you relatively dry and profitable while everyone else got soaked.

10.3 Backtesting Methodologies

Backtesting is the process of testing a strategy on historical data.

10.3.1 Walk-Forward Analysis

Instead of testing on the entire dataset at once, we use a rolling window approach. We train on a period, test on the next, and then slide the window forward. This simulates how a model would adapt to changing market conditions over time.

10.3.2 Out-of-Sample (OOS) Testing

We strictly reserve a portion of the data (e.g., the last 6 months) as a "vault." We do not look at or use this data during development. It is used only for the final test to ensure the model hasn't just memorized the past.

Real-Life Example: COVID Crash Backtest

Pre-2020 train: Bull-lover flops in '20 OOS crash. Walk-forward pivots, nabbing rebound vibes.

11 SOME CHALLENGES

Industry-Standard Practices

These challenges introduce production-level techniques used in professional data engineering and quantitative finance. Completing these challenges will significantly enhance the robustness and scalability of your project.
Each challenge includes a dedicated Jupyter notebook with complete implementation examples.

11.1 Challenge 1: Professional Data Storage

The Problem: CSV files are inefficient for large datasets and lose data type information (e.g., timestamps become strings).

Your Mission: Replace `to_csv` with SQLite or Parquet for incremental data storage.

Why This Matters:

- Preserves data types
- Enables efficient querying
- Reduces storage requirements
- Supports concurrent access

Implementation Options:

SQLite:

- Lightweight embedded database
- Supports SQL queries
- Ideal for local development

Parquet:

- Columnar storage format
- Highly compressed
- Industry standard for big data

Resources:

- Pandas to_sql Documentation
- Notebook: `pro_challenge_1_data_storage.ipynb`

11.2 Challenge 2: Named Entity Recognition Filter

The Problem: Searching for "Apple" may return articles about fruit. Searching for "Amazon" may return articles about the rainforest.

Your Mission: Implement NER filtering using spaCy to retain only articles where entities are classified as ORG (Organization).

Implementation Steps:

1. Load spaCy English model: `en_core_web_sm`
2. Process each article to extract named entities
3. Filter articles where company name is tagged as ORG
4. Discard articles where company name is tagged as LOCATION or other types

Expected Outcomes:

- Improved data quality
- Reduced noise in sentiment signals
- Better model performance

Resources:

- spaCy Named Entity Recognition
- NER with spaCy Tutorial
- Notebook: `pro_challenge_2_ner_filter.ipynb`

11.3 Challenge 3: Stationarity Testing

The Problem: Neural networks perform poorly on non-stationary time series (data with trends). Raw stock prices contain trends and are non-stationary.

Your Mission: Use the Augmented Dickey-Fuller (ADF) test to verify stationarity. Transform data until the test passes (p-value ¡ 0.05).

Understanding Stationarity:

A stationary time series has:

- Constant mean over time
- Constant variance over time
- No predictable seasonal patterns

Transformation Methods:

- **Differencing:** $y'_t = y_t - y_{t-1}$
- **Log Returns:** $r_t = \ln(P_t/P_{t-1})$
- **Percentage Returns:** $r_t = (P_t - P_{t-1})/P_{t-1} \times 100$

Implementation:

```
from statsmodels.tsa.stattools import adfuller
result = adfuller(data)
p_value = result[1]
if p_value < 0.05:
    print("Data is stationary")
```

Critical Rule: You cannot proceed to model training in Week 3 unless your data passes the stationarity test.

Resources:

- ADF Test in Python
- ADF Test Tutorial
- Notebook: `pro_challenge_3_stationarity.ipynb`

11.4 Challenge 4: Timezone Alignment

The Problem: Indian stock markets (NSE/BSE) operate 9:15 AM - 3:30 PM IST. News occurs globally 24/7. News published at 8:00 PM IST affects the next trading day, not the current day.

Your Mission: Create a `DataAligner` function that:

1. Converts all news timestamps to IST
2. Maps news after market close to the next trading day
3. Handles weekends (Saturday/Sunday news maps to Monday)

Implementation Strategy:
Use `pandas.tseries.offsets.BusinessDay` for business day calculations:

```
from datetime import time
import pandas as pd

MARKET_CLOSE = time(15, 30)

def map_to_trading_day(timestamp):
    if timestamp.weekday() >= 5:  # Weekend
        return next_monday(timestamp)
    if timestamp.time() > MARKET_CLOSE:
        return timestamp + pd.tseries.offsets.BusinessDay(1)
    return timestamp.date()
```

Why This Matters:

- Prevents data leakage
- Ensures correct causal relationships
- Aligns with actual trading mechanics

Resources:

- Notebook: `pro_challenge_4_timezone_alignment.ipynb`

12 Development Environment Setup

12.1 Initial Setup

Create Project Structure:

```
mkdir sentiment-stock-prediction
cd sentiment-stock-prediction
python -m venv venv
source venv/bin/activate  # On Windows: venv\Scripts\activate
```

12.2 Required Libraries

Create `requirements.txt`:

```
pandas>=1.3.0
numpy>=1.21.0
yfinance>=0.1.70
newsapi-python>=0.2.6
praw>=7.5.0
nltk>=3.6.5
spacy>=3.2.0
transformers>=4.15.0
torch>=1.10.0
statsmodels>=0.13.0
matplotlib>=3.4.3
jupyter>=1.0.0
pyarrow>=6.0.0
```

Install dependencies:

```
pip install -r requirements.txt
python -m spacy download en_core_web_sm
```

12.3 Project Structure

```
sentiment-stock-prediction/
 data/
    raw/
    processed/
    databases/
 notebooks/
 src/
    data_collection/
    preprocessing/
    sentiment/
    models/
 tests/
 requirements.txt
 .gitignore
 README.md
```

12.4 Version Control

Initialize Git repository:

```
git init
```

Create `.gitignore`:

```
.env
api_keys.py
*.db
*.parquet
*.h5
__pycache__/
*.pyc
venv/
.ipynb_checkpoints/
```

Security Best Practice: Never commit API keys, credentials, or sensitive data to version control.

13 Week 1 Completion Checklist

Essential Tasks

Foundational Knowledge:

- ✓ Understand project workflow and objectives
- ✓ Review Python programming fundamentals
- ✓ Complete Pandas and NumPy tutorials

Development Environment:

- ✓ Set up virtual environment
- ✓ Install all required libraries
- ✓ Verify installations
- ✓ Configure version control

Data Collection:

- ✓ Obtain NewsAPI credentials
- ✓ Set up PRAW for Reddit
- ✓ Download sample stock data using yfinance
- ✓ Test all API connections

NLP Fundamentals:

- ✓ Implement tokenization
- ✓ Apply stop word removal
- ✓ Test lemmatization
- ✓ Run VADER sentiment analysis
- ✓ Test FinBERT on sample headlines

Pro Challenges (Recommended):

- ✓ Implement SQLite or Parquet storage
- ✓ Apply NER filtering with spaCy
- ✓ Perform stationarity testing
- ✓ Implement timezone alignment

WiDS 5.0 – Market Mood & Moves — Week 1 Learning Guide
Last Updated: December 2025