Module Code & Module Title

CU6051NI Artificial Intelligence

75% Individual Coursework

Submission: Proposal

Academic Semester: Autumn Semester 2025

Credit: 15 credit semester long module

Student Name: Amiks Karki

London Met ID: 23049352

College ID: NP01CP4A230172

Assignment Due Date: 07/01/2026.

Assignment Submission Date: 07/01/2026

Submitted To: Mr. Binod Bhattarai

GitHub Link	*https://github.com/AmiksKarki/Stock-Price-Prediction*

Table of Contents

Table of Figures

Table of Tables

1 Title of the Project

Evaluating Deep Learning Architectures (LSTM, CNN, and Transformers) for NEPSE Stock Price Direction Prediction on Aggregated Company Data.

2 Introduction

Machine learning has become a fundamental part of the contemporary artificial intelligence as computational systems are capable of recognizing patterns in sets of data and producing predictions not tied to a collection of personally defined rules. The methods are used very popularly in tasks of analysis including classification, regression, clustering, and forecasting of sequential data. This has been facilitated by the growing access to large-scale data sets coupled with the increase of computational power that have allowed machine learning models to reveal concealed behaviors in complex data making it highly useful in many sectors such as healthcare, finance, transportation, manufacturing and digital commerce (Hapuarachchi, 2025).

Financial markets are very dynamic in behavior with a great deal of uncertainty, noise and strong non-linear relationships especially in stock exchanges. The dynamics of the stock prices are determined by the interplay of the economic indicators, political events, market sentiment and investor psychology, which made it very difficult to predict. Therefore, classification of stock prices has remained as one of the most difficult problems in machine learning particularly where highly sensitive financial time-series data is involved. Still, predictive models that are predictable with high accuracy could be of great value to investors, financial entities, and regulators as they enable sound decision-making (Seetharaman, 2021).

This research focuses on building a conceptual AI solution for predicting NEPSE (Nepal Stock Exchange) stock prices using three popular deep learning models: Long Short-Term Memory (LSTM) networks (Wenjie Lu, 2020), 1D Convolutional Neural Networks (CNN), and Transformer architectures. These models represent three different approaches to sequence modelling and allow a comparative analysis of forecasting performance.

NEPSE is an emerging market with limited research done on data-driven forecasting. This makes it an ideal problem domain, as the development of intelligent predictive models could contribute towards improved decision-making in the Nepali financial system.

2.1 Key Deliverables:

- Comprehensive dataset preparation pipeline for NEPSE stock data
- Implementation of three deep learning architectures optimized for time-series classification
- Rigorous performance evaluation using multiple metrics
- Comparative analysis identifying strengths and limitations of each approach
- Evidence-based recommendations

3 Background and literature review

3.1 NEPSE Stock Market Domain

The Nepal Stock Exchange (NEPSE) represents a dynamic financial marketplace where shares of different companies including commercial banks, insurance firms, hydropower projects, telecommunications, and manufacturing industries are traded.

NEPSE lists a range of commercial banks, insurance companies, hydropower projects, telecom operators, and manufacturing industries. Typical historical price datasets represent market behaviour through multiple indicators, for example:

- Open price
- High price
- Low price
- Close price
- Percentage change
- Traded quantity
- Traded amount

The availability of structured CSV datasets for NEPSE companies allows experimentation with AI algorithms for prediction.

3.2 Time-Series Forecasting in AI

In time-series analysis, future data points are projected by learning temporal relationships from historical ordered data. Stock prices are a typical example, where the order of data points is crucial. Traditional statistical models have limitations in capturing complex non-linear patterns, motivating the use of deep learning.

3.3 Deep Learning Approaches for Forecasting Financial Market Data

Deep learning models excel at feature extraction, non-linear pattern learning, and sequence modelling. The following architectures are widely used for financial forecasting:

3.3.1 Long Short-Term Memory (LSTM) Networks

Long Short-Term Memory networks extend the recurrent neural network framework by introducing memory cells and gating mechanisms that regulate information flow, allowing the model to retain relevant information over long temporal sequences.
Strengths:

- Good at learning temporal relationships
- Handles vanishing gradients
- Widely used for stock prediction

3.3.2 Convolutional Neural Networks (1D CNN) for Time-Series

Though commonly used in image processing, 1D CNNs can detect spatial/temporal patterns in sequential data.

Strengths:

- Captures local patterns
- Faster to train than LSTM
- Effective for short-term prediction windows

3.3.3 Transformer Models

Transformers use attention-based mechanisms rather than recurrent structures, which helps the model identify influential time steps and understand long-range patterns in sequential data.

Strengths:

- State-of-the-art performance in many time-series tasks
- Parallel training
- Works well with multi-series datasets

3.4 Review of Existing Literature

Research in global stock forecasting has used LSTM, GRU, CNN, and Transformer-based models.

3.4.1 Study 1: Forecasting NEPSE Index Prices Using Deep Learning Techniques

- **Key Finding:** The **LSTM model architecture provided the best performance** in terms of standard assessment metrics like Root Mean Square Error (RMSE) and Mean Absolute Percentage Error (MAPE) compared to GRU and CNN, highlighting its strength in processing the non-linear, complex, and volatile NEPSE time series data. (Nawa Raj Pokhrel, 2022)

3.4.2 Study 2: Financial Market Prediction Employing Long Short-Term Memory Networks

- **Key Findings:** LSTM significantly outperformed traditional ML (Random Forests, DNNs) by capturing long-term dependencies. Preprocessing (normalization, sequence formation) was critical. (Thomas Fischer, 2018)

3.4.3 Study 3: Comparative Analysis of RNN, LSTM, BiLSTM, GRU, and Transformer Models for Financial Forecasting

- **Key Findings:** Transformer exhibited superior accuracy and more efficient convergence compared to all RNN variants (LSTM, GRU), highlighting the effectiveness of the self-attention mechanism. (T.O. Kehinde, 2023)

3.4.4 Study 4: Stock Prediction Based on Transformer Model

- **Key Findings:** Combining the **Transformer with LASSO regularization** significantly improved robustness and achieved low MAPE, demonstrating the necessity of **feature engineering** for accurate NEPSE forecasting. (Ekanta Rai, 2025)

3.4.5 Study 5: A Deep Neural Network-Based Stock Trading System Incorporating Evolutionary Optimized Technical Analysis Parameters

- **Key Findings:** CNN performed better for high-frequency trading (speed). LSTM gave more stable predictions for longer investment horizons. (Omer Berat Sezera, 2017)

These studies show:

- LSTM captures sequential memory effectively
- CNN is efficient for pattern extraction
- Transformers often outperform both due to attention mechanisms

In Nepal, limited academic literature exists for computational stock prediction, highlighting a gap where AI-based forecasting can provide value.

3.5 Problem Statement

Stock price prediction on emerging markets such as NEPSE encounters great difficulty in predicting due to complexity of the data, volatility in the markets, availability of systematic comparative study using advanced deep learning architecture. Furthermore, there is little work on the application of state-of-the-art models such as Transformers to NEPSE data, leaving questions to whether which algorithms work best in this market context.

The main problem this paper resolves is the following problem:

How do LSTM, CNN, and Transformer models compare in predicting next-day stock price direction (UP/DOWN) for companies listed on NEPSE?

Change from coursework 1: The courserowk 1 focused on predicting next-day closing prices (regression problem). The implementation shifted to predicting price direction (classification problem). This change was strategic for several reasons:

- Practical Trading Relevance – Trading decisions depend on price direction rather than exact price values.
- Noise Robustness – Directional prediction is less sensitive to market noise and

volatility.
- Clear Evaluation – Classification metrics provide more interpretable and decision oriented performance assessment.
- Both approaches were implemented and evaluated; however, directional classification consistently produced more stable and reliable results.

By solving this problem, the research will give evidence-based suggestions of choosing suitable deep learning architectures for stock price forecasting for emerging markets to contribute to better decision-making.

3.6 Research Questions / Objectives

3.6.1 Research Questions

1. Which deep learning architecture (LSTM, CNN or Transformer) has the most accurate prediction accuracy of NEPSE Stock Price Forecasting?
2. How are the preprocessing steps such as data normalization, sequence generation and feature encoding affecting the performance of each model?
3. What are the computation efficiency differences between the LSTM, the CNN, and the Transformer model in terms of time of training and the amount of resources needed?
4. Which are the most suitable evaluation metrics in predicting financial time series data in real-world scenarios?

3.6.2 Research Objectives

1. Develop a unified multi-company dataset by incorporating historical stock data from NEPSE-listed companies.
2. Design and implement preprocessing pipelines such as data cleaning, feature scaling and sequence generation suitable for deep learning models.
3. Develop three architectures for deep learning: LSTM, CNN and Transformer which are optimized for time series regression using the same input data to compare the three architectures fairly.

4. Train all the three models with the same hyperparameters, training protocols and validation strategies.
5. Evaluate model performance using several metrics and visualize comparison results.
6. Issue evidence-based suggestions for algorithm choice and provide evidence for selecting sensitive, efficient, and implementation feasible for algorithms.

4 Solution

4.1 Proposed Solution Overview

The solution implements a supervised learning framework where historical stock data is used to predict future directional movement. The approach involves collecting 15+ years of NEPSE trading data, engineering 18 technical indicators from raw price and volume data, creating binary classification targets (UP/DOWN) based on next-day price changes, generating 60-day sliding window sequences for temporal context, training three distinct deep learning architectures independently, and comparing their performance on unseen test data.

4.1.1 Proposed AI Algorithms

Algorithm	Architecture Type	Reason for Selection
LSTM	Recurrent Neural Network (Deep Learning)	LSTM networks are designed for sequential data and effectively capture long-term temporal dependencies using memory cells and gating mechanisms. (Sepp Hochreiter, 1997) They overcome the vanishing gradient problem of traditional RNNs and have demonstrated strong performance in financial time-series forecasting by learning complex historical price patterns over extended periods. (Thomas Fischer, 2018)
CNN (1D)	Convolutional Neural Network (Deep Learning)	CNNs with one-dimensional convolutions efficiently extract local temporal patterns from time-series data. They are effective in identifying short-term trends and sudden market movements while offering faster training and computational efficiency compared to recurrent models (Avinash, 2021).
Transformer	Attention-based	Transformers utilize self-attention mechanisms to model

	Neural Network (Deep Learning)	both short-term and long-term dependencies without relying on recurrence. Their ability to process sequences in parallel improves training efficiency, and recent studies show superior performance in time-series forecasting tasks compared to traditional architectures (Rogerio Pereira dos Santos, 2025)

Table 1: Proposed Algorithms

4.1.2 Dataset Description

- Dataset Source: Historical stock price data collected from Nepal Stock Exchange (NEPSE) official records made publicly available via GitHub repository. The repository is continuously updated using automated web-scraping techniques to reflect daily market activity.
- Type: Time-series financial dataset containing daily stock trading information.
- Instances: Each instance represents one trading day for a listed stock. The dataset spans multiple years with new records appended on each trading day.
- Features: The dataset used in this study includes the following variables:
    - published_date: The date corresponding to each trading session
    - open: The stock's price at the start of the trading day
    - high: The maximum price reached during the trading session
    - low: The minimum price observed within the session
    - close: The price at market close
    - per_change: The percentage change in the closing price relative to the previous trading day
    - traded_quantity: Total number of shares traded
    - traded_amount: Total monetary value of shares traded
    - company_id: Company identifier
- Target Variable: The study focuses on forecasting the directional accuracy (0=DOWN, 1=UP)
- Missing Values: The dataset contains occasional missing values which are

handled during the preprocessing stage using appropriate techniques.

4.2 Pseudocode

4.2.1 Dataset Creation and Sequence Generation

```
BEGIN
    // Load and combine data
    FOR each commercial bank CSV file:
        Load data
        Add company identifier
    END FOR
    Merge all into single dataset
    Sort by date (chronological)
    // Create target variable
    FOR each company:
        FOR each row:
            IF next_day_close > today_close THEN
                target = 1  // UP
            ELSE
                target = 0  // DOWN
            END IF
        END FOR
    END FOR
    // Engineer features
    Calculate technical indicators (MA, RSI, MACD, etc.)
    Total features = 18
    // Split data temporally
    train_data = records before 2024-01-01
    test_data = records from 2024-01-01 onward
    // Encode and scale
    Encode company_id to numeric (0-12)
    Fit StandardScaler on train_data
    Transform both train and test
    // Create sequences
    FOR each company:
        FOR each valid 60-day window:
            Input = 60 days of features
            Target = day 61 direction
        END FOR
    END FOR
    // Create DataLoaders
    Create PyTorch datasets
    Create batched loaders (batch_size=64)
    RETURN train_loader, test_loader
END
```

4.2.2 LSTM Model

```
BEGIN
    // Architecture
    INITIALIZE:
        Company_Embeddings
        BiLSTM(3 layers, 128 units, dropout=0.3)
        LayerNorm()
        Attention_Mechanism()
        Classifier(hidden=128, output=2)
    FORWARD(sequence, company_id):
        // Process sequence
        lstm_out = BiLSTM(sequence)
        lstm_out = LayerNorm(lstm_out)
        // Attention
        attn_weights = Attention(lstm_out)
        context = Sum(attn_weights × lstm_out)
        // Combine and classify
        company_emb = Embedding(company_id)
        combined = Concatenate(context, company_emb)
        output = Classifier(combined)
        probabilities = Softmax(output)
        RETURN probabilities
    // Training
    TRAIN:
        optimizer = AdamW(lr=0.001)
        loss_fn = CrossEntropyLoss(class_weights)
```

```
best_accuracy = 0
patience = 0
FOR epoch = 1 TO 50:
   // Train
   FOR batch in train_loader:
      predictions = Forward(batch)
      loss = loss_fn(predictions, targets)
      Backpropagate(loss)
      Update_weights()
   END FOR
   // Validate
   accuracy = Evaluate(test_loader)
   // Early stopping
   IF accuracy > best_accuracy THEN
      best_accuracy = accuracy
      Save_model()
      patience = 0
   ELSE
      patience += 1
      IF patience >= 15 THEN
         STOP
      END IF
   END IF
END FOR
Load best model
```

```
    RETURN moel
END
```

4.2.3 CNN Model

```
BEGIN
    // Architecture
    INITIALIZE:
        Company_Embedding
        Conv1D_Block1(kernel=3, filters=128)
        Conv1D_Block2(kernel=5, filters=128)
        Conv1D_Block3(kernel=7, filters=128)
        BatchNorm_Layers()
        GlobalMaxPool()
        Classifier(hidden=128, output=2)
    FORWARD(sequence, company_id):
        // Transpose for convolution
        x = Transpose(sequence)  // [batch, features, time]
        // Multi-scale convolutions
        x = Conv1D_3(x)
        x = BatchNorm(x) → ReLU → Dropout
        x = Conv1D_5(x)
        x = BatchNorm(x) → ReLU → Dropout
        x = Conv1D_7(x)
        x = BatchNorm(x) → ReLU → Dropout
        // Pool and classify
        x = GlobalMaxPool(x)
        company_emb = Embedding(company_id)
        combined = Concatenate(x, company_emb)
        output = Classifier(combined)
        probabilities = Softmax(output)
        RETURN probabilities
    // Training (same structure as LSTM)
    TRAIN:
        optimizer = AdamW(lr=0.001)
        loss_fn = CrossEntropyLoss(class_weights)
        Train with early stopping (patience=15)
        RETURN best model
END
```

4.2.4 Transformer Model

```
BEGIN
  // Architecture
  INITIALIZE:
    Company_Embedding(13 → 16)
    Input_Projection(18 → 128)
    Positional_Encoding(learnable)
    Transformer_Encoder(3 layers, 8 heads)
    Classifier(hidden=128, output=2)
  FORWARD(sequence, company_id):
    // Project and add position
    x = Input_Projection(sequence)
    x = x + Positional_Encoding
    x = Dropout(x)
    // Transformer layers
    FOR layer = 1 TO 3:
      attn_out = MultiHeadAttention(x)
      x = LayerNorm(x + attn_out)
      ff_out = FeedForward(x)
      x = LayerNorm(x + ff_out)
    END FOR
    // Aggregate and classify
    x = MeanPool(x)  // Average over time
    company_emb = Embedding(company_id)
    combined = Concatenate(x, company_emb)
    output = Classifier(combined)
    probabilities = Softmax(output)
    RETURN probabilities
  // Training
  TRAIN:
    optimizer = AdamW(lr=0.0005)  // Lower LR
    loss_fn = CrossEntropyLoss(class_weights)
    Train with early stopping (patience=15)
    RETURN best model
END
```

4.2.5 Model Evaluation

```
BEGIN Evaluation
    // Evaluate single model
    FUNCTION Evaluate(model, test_loader):
        model.eval()
        all_predictions = []
        all_targets = []
        FOR batch in test_loader:
            predictions = model.forward(batch)
            predicted_class = ArgMax(predictions)
            Append predictions and targets
        END FOR
        // Calculate metrics
        accuracy = Correct / Total
        precision = TP / (TP + FP)
        recall = TP / (TP + FN)
        f1 = 2 × (precision × recall) / (precision + recall)
        RETURN metrics
    // Compare all models
    lstm_results = Evaluate(lstm_model, test_loader)
    cnn_results = Evaluate(cnn_model, test_loader)
    transformer_results = Evaluate(transformer_model, test_loader)
    // Identify best
    best_model = model with highest accuracy
    // Visualize
    Plot_comparison_charts()
    Plot_confusion_matrices()
    RETURN comparison_results
END Evaluation
```

4.3 Diagrammatic Representation

4.3.1 Overall System Flowchart

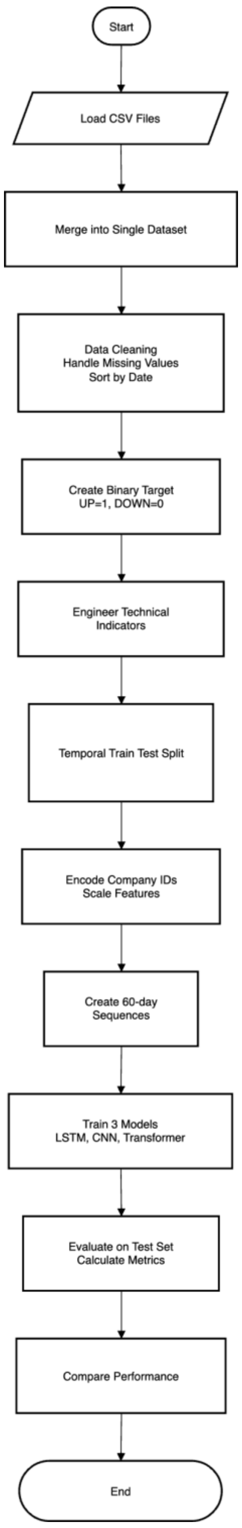


Figure 1: Overall System Flowchart

4.3.2 Data preprocessing flowchart

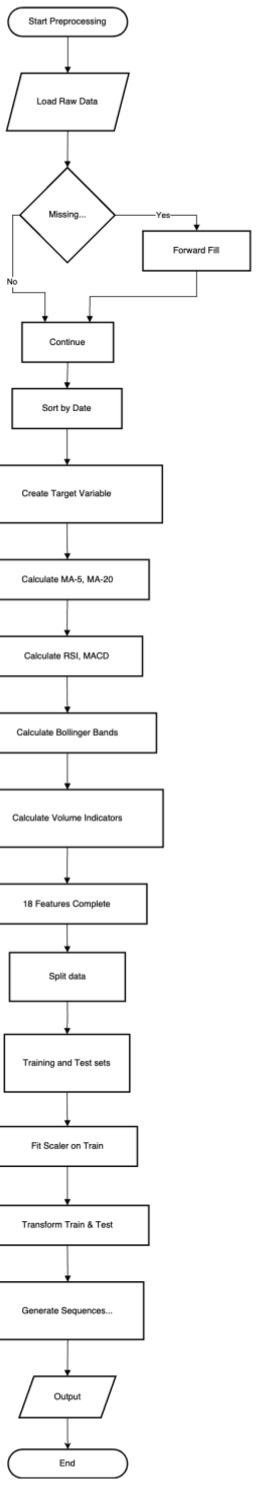


Figure 2: Data Preprocessing

4.3.3 LSTM Model Implementation Flowchart

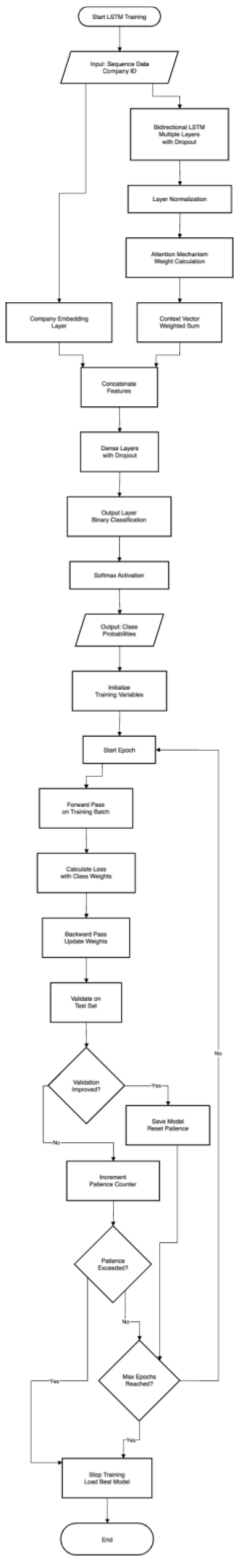

Start LSTM Training
Input: Sequence Data Company ID
Bidirectional LSTM Multiple Layers with Dropout
Layer Normalization
Attention Mechanism Weight Calculation
Company Embedding Layer
Context Vector Weighted Sum
Concatenate Features
Dense Layers with Dropout
Output Layer Binary Classification
Softmax Activation
Output: Class Probabilities
Initialize Training Variables
Start Epoch
Forward Pass on Training Batch
Calculate Loss with Class Weights
Backward Pass Update Weights
Validate on Test Set
Validation Improved?
Yes
Save Model Reset Patience
No
Increment Patience Counter
Patience Exceeded?
No
Yes
Max Epochs Reached?
No
Yes
Stop Training Load Best Model
End

Figure 3: LSTM Implementation

4.3.4 CNN Model Implementation Flowchart

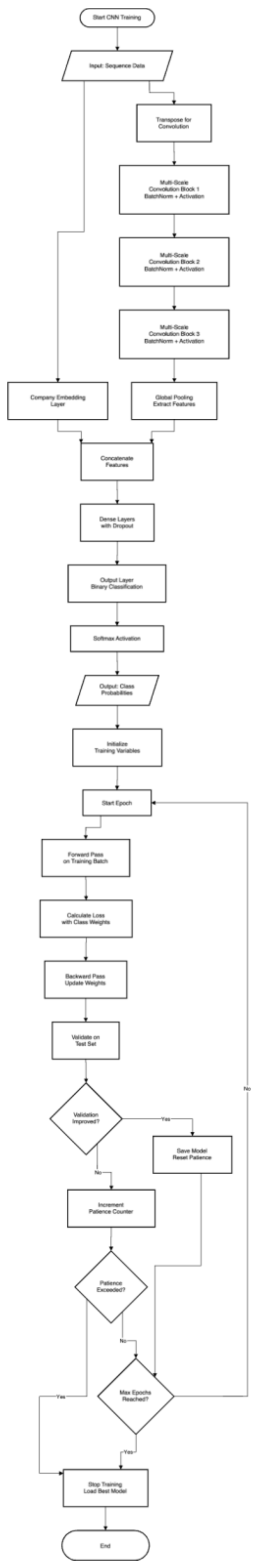

Start CNN Training
Input: Sequence Data
Transpose for Convolution
Multi-Scale Convolution Block 1 BatchNorm + Activation
Multi-Scale Convolution Block 2 BatchNorm + Activation
Multi-Scale Convolution Block 3 BatchNorm + Activation
Company Embedding Layer
Global Pooling Extract Features
Concatenate Features
Dense Layers with Dropout
Output Layer Binary Classification
Softmax Activation
Output: Class Probabilities
Initialize Training Variables
Start Epoch
Forward Pass on Training Batch
Calculate Loss with Class Weights
Backward Pass Update Weights
Validate on Test Set
Validation Improved?
Yes
Save Model Reset Patience
No
Increment Patience Counter
Patience Exceeded?
No
Yes
Max Epochs Reached?
No
Yes
Stop Training Load Best Model
End

Figure 4: CNN Implementation

4.3.5 Transformer Model Implementation Flowchart

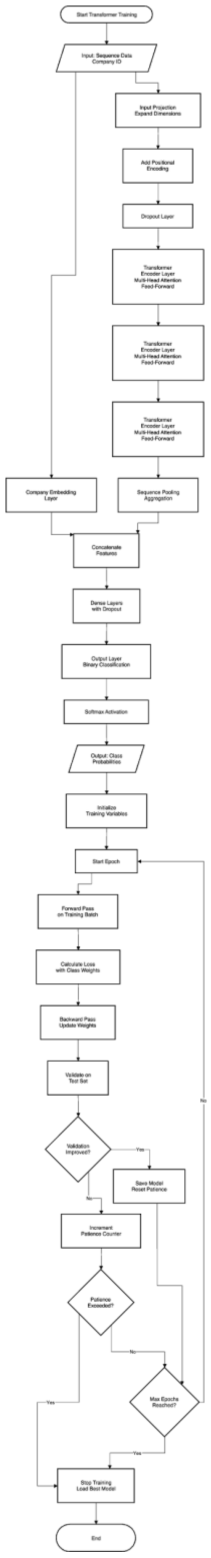

Start Transformer Training
Input: Sequence Data Company ID
Input Projection Expand Dimensions
Add Positional Encoding
Dropout Layer
Transformer Encoder Layer Multi-Head Attention Feed-Forward
Transformer Encoder Layer Multi-Head Attention Feed-Forward
Transformer Encoder Layer Multi-Head Attention Feed-Forward
Company Embedding Layer
Sequence Pooling Aggregation
Concatenate Features
Dense Layers with Dropout
Output Layer Binary Classification
Softmax Activation
Output: Class Probabilities
Initialize Training Variables
Start Epoch
Forward Pass on Training Batch
Calculate Loss with Class Weights
Backward Pass Update Weights
Validate on Test Set
Validation Improved?
Yes
Save Model Reset Patience
No
Increment Patience Counter
Patience Exceeded?
No
Yes
Max Epochs Reached?
No
Yes
Stop Training Load Best Model
End

Figure 5: Transformer Implementation

4.4 Explanation of the development process

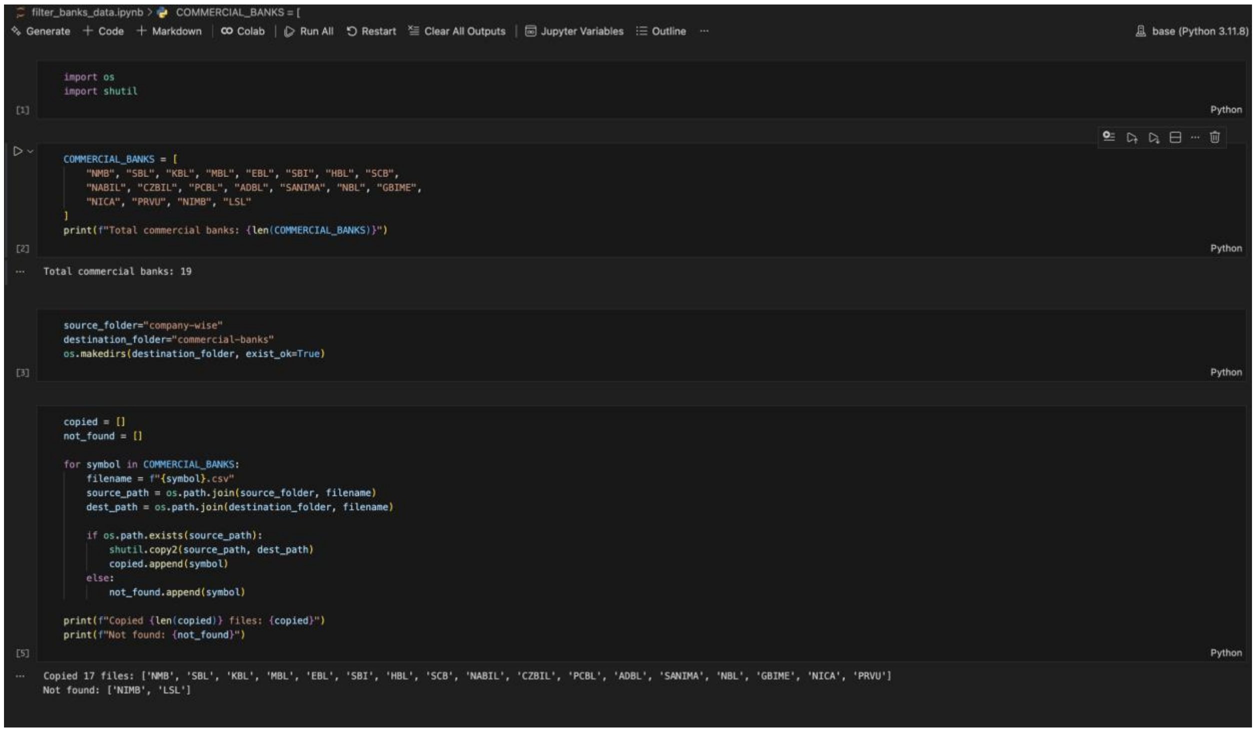
4.4.1 Development Workflow

4.4.1.1 Setup and Exploration

- Configured Google Colab environment
- Loaded and inspected 17 commercial bank datasets
- Analyzed data quality, completeness, and distribution
- Created initial exploratory visualizations

4.4.1.2 Data Preprocessing

- Merged datasets from 13 banks (38,138 records)
- Handled missing values through forward-fill
- Engineered binary target: target = 1 if close(t+1) > close(t) else 0
- Calculated 18 technical indicators (MA, RSI, MACD, Bollinger Bands, ATR, Volume ratios)
- Implemented temporal split: Train (before 2024-01-01), Test (2024+ onward)
- Encoded company IDs (0-12)
- Standardized features using StandardScaler (fit on train only)
- Generated 60-day sliding window sequences
- Created PyTorch DataLoaders (batch size: 64)



```
import os
import shutil
```

```
COMMERCIAL_BANKS = [
    "NMB", "SBL", "KBL", "MBL", "EBL", "SBI", "HBL", "SCB",
    "NABIL", "CZBIL", "PCBL", "ADBL", "SANIMA", "NBL", "GBIME",
    "NICA", "PRVU", "NIMB", "LSL"
]
print(f"Total commercial banks: {len(COMMERCIAL_BANKS)}")
```

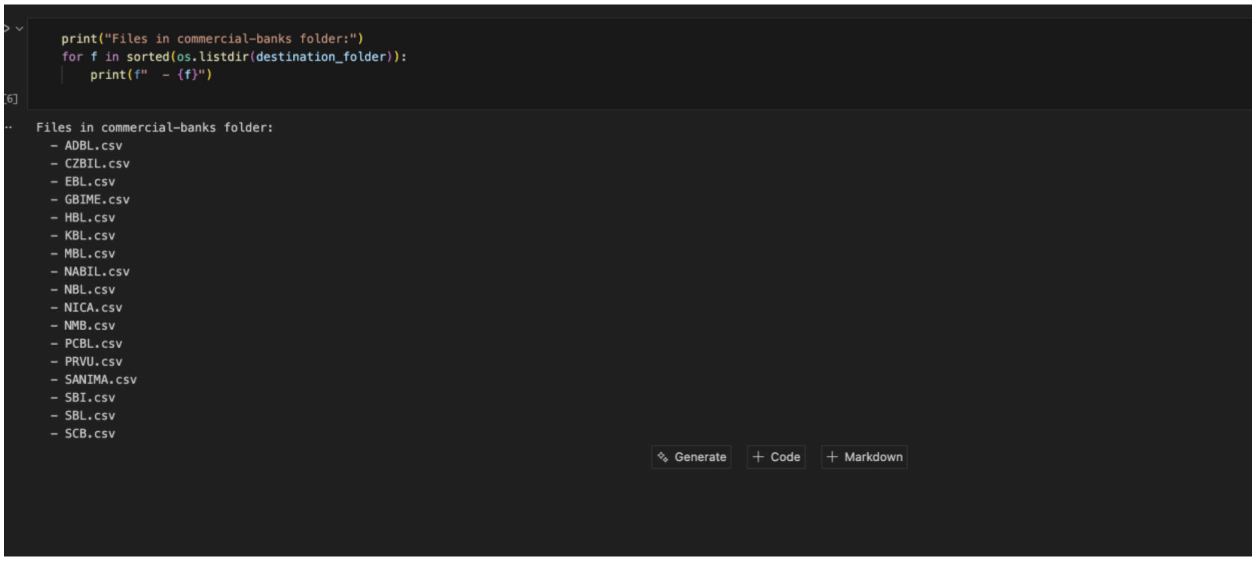
```
Total commercial banks: 19
```

```
source_folder="company-wise"
destination_folder="commercial-banks"
os.makedirs(destination_folder, exist_ok=True)
```

```
copied = []
not_found = []

for symbol in COMMERCIAL_BANKS:
    filename = f"{symbol}.csv"
    source_path = os.path.join(source_folder, filename)
    dest_path = os.path.join(destination_folder, filename)

    if os.path.exists(source_path):
        shutil.copy2(source_path, dest_path)
        copied.append(symbol)
    else:
        not_found.append(symbol)

print(f"Copied {len(copied)} files: {copied}")
print(f"Not found: {not_found}")
```

```
Copied 17 files: ['NMB', 'SBL', 'KBL', 'MBL', 'EBL', 'SBI', 'HBL', 'SCB', 'NABIL', 'CZBIL', 'PCBL', 'ADBL', 'SANIMA', 'NBL', 'GBIME', 'NICA', 'PRVU']
Not found: ['NIMB', 'LSL']
```

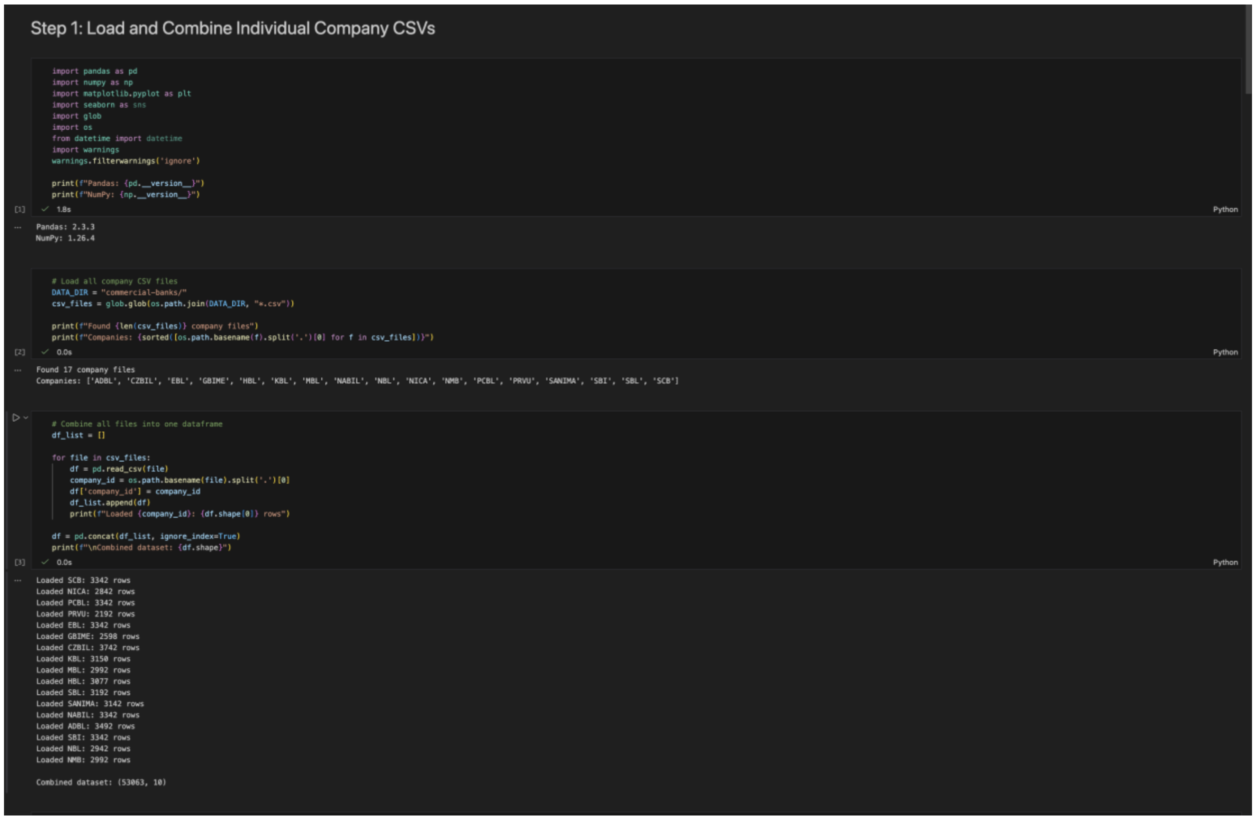
Figure 6: Filtering commercial banks data



```
print("Files in commercial-banks folder:")
for f in sorted(os.listdir(destination_folder)):
    print(f"  - {f}")
```

```
Files in commercial-banks folder:
  - ADBL.csv
  - CZBIL.csv
  - EBL.csv
  - GBIME.csv
  - HBL.csv
  - KBL.csv
  - MBL.csv
  - NABIL.csv
  - NBL.csv
  - NICA.csv
  - NMB.csv
  - PCBL.csv
  - PRVU.csv
  - SANIMA.csv
  - SBI.csv
  - SBL.csv
  - SCB.csv
```

Figure 7: Saving the commercial banks data in the respective folder



Step 1: Load and Combine Individual Company CSVs

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import glob
import os
from datetime import datetime
import warnings
warnings.filterwarnings('ignore')

print(f"Pandas: {pd.__version__}")
print(f"NumPy: {np.__version__}")
```

```
Pandas: 2.3.3
NumPy: 1.26.4
```

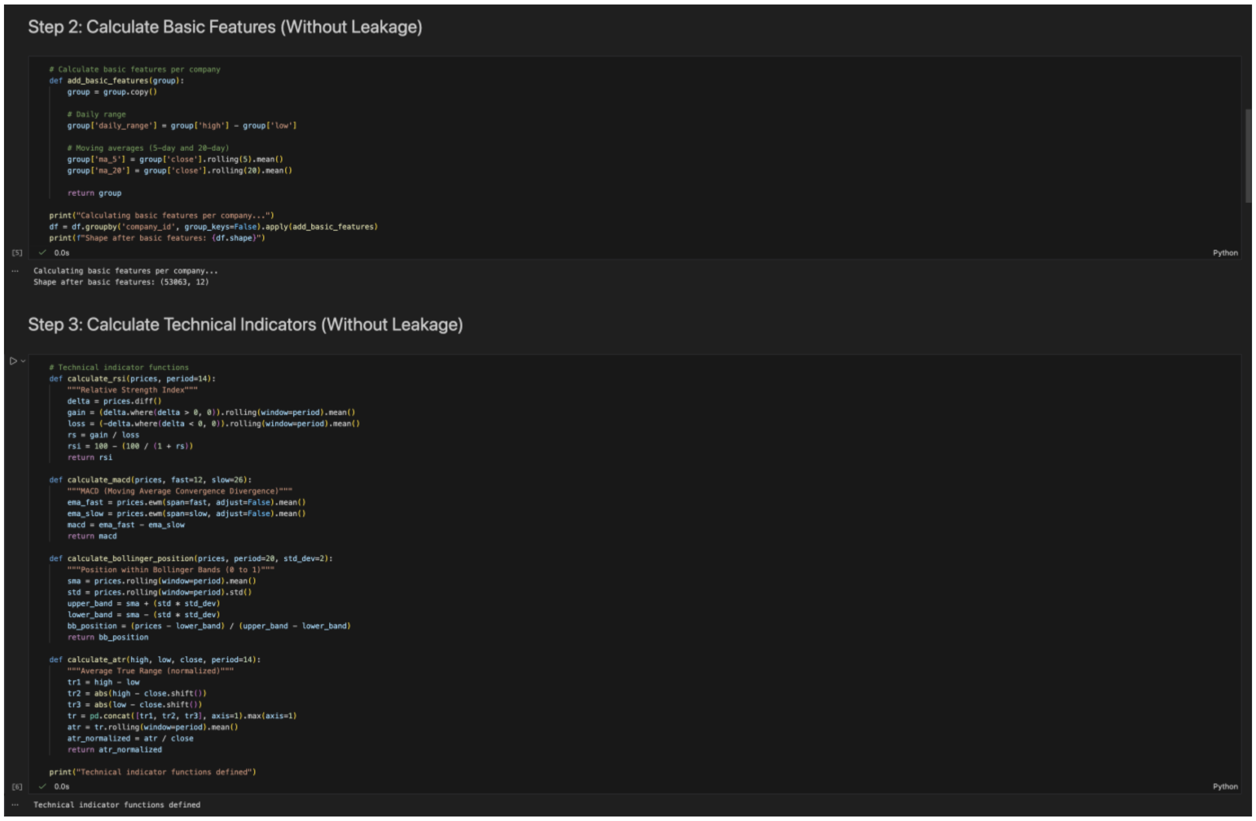
```python
# Load all company CSV files
DATA_DIR = "commercial-banks/"
csv_files = glob.glob(os.path.join(DATA_DIR, "*.csv"))

print(f"Found {len(csv_files)} company files")
print(f"Companies: {sorted([os.path.basename(f).split('.')[0] for f in csv_files])}")
```

```
Found 17 company files
Companies: ['ADBL', 'CZBIL', 'EBL', 'GBIME', 'HBL', 'KBL', 'MBL', 'NABIL', 'NBL', 'NICA', 'NMB', 'PCBL', 'PRVU', 'SANIMA', 'SBI', 'SBL', 'SCB']
```

```python
# Combine all files into one dataframe
df_list = []

for file in csv_files:
    df = pd.read_csv(file)
    company_id = os.path.basename(file).split('.')[0]
    df['company_id'] = company_id
    df_list.append(df)
    print(f"Loaded {company_id}: {df.shape[0]} rows")

df = pd.concat(df_list, ignore_index=True)
print(f"\nCombined dataset: {df.shape}")
```

```
Loaded SCB: 3342 rows
Loaded NICA: 2842 rows
Loaded PCBL: 3342 rows
Loaded PRVU: 2192 rows
Loaded EBL: 3342 rows
Loaded GBIME: 2598 rows
Loaded CZBIL: 3742 rows
Loaded KBL: 3150 rows
Loaded MBL: 2992 rows
Loaded HBL: 3077 rows
Loaded SBL: 3192 rows
Loaded SANIMA: 3142 rows
Loaded NABIL: 3342 rows
Loaded ADBL: 3492 rows
Loaded SBI: 3342 rows
Loaded NBL: 2942 rows
Loaded NMB: 2992 rows

Combined dataset: (53063, 10)
```

Figure 8: Combining individual banks dataset into single one



Step 2: Calculate Basic Features (Without Leakage)

```python
# Calculate basic features per company
def add_basic_features(group):
    group = group.copy()

    # Daily range
    group['daily_range'] = group['high'] - group['low']

    # Moving averages (5-day and 20-day)
    group['ma_5'] = group['close'].rolling(5).mean()
    group['ma_20'] = group['close'].rolling(20).mean()

    return group

print("Calculating basic features per company...")
df = df.groupby('company_id', group_keys=False).apply(add_basic_features)
print(f"Shape after basic features: {df.shape}")
```

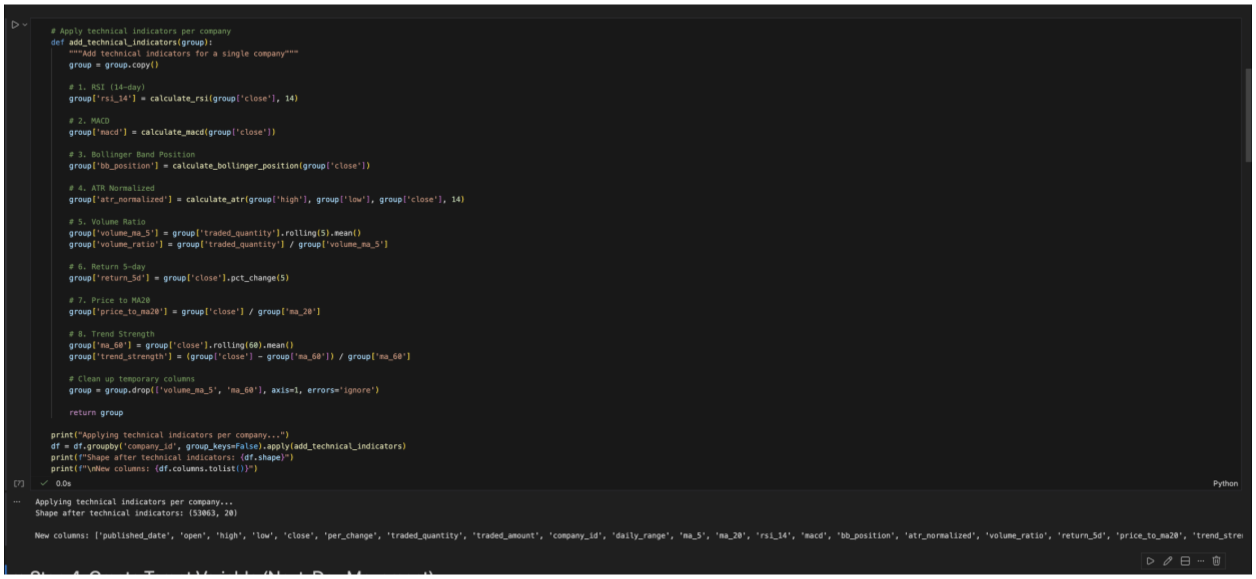
```
Calculating basic features per company...
Shape after basic features: (53063, 12)
```

Step 3: Calculate Technical Indicators (Without Leakage)

```python
# Technical indicator functions
def calculate_rsi(prices, period=14):
    """Relative Strength Index"""
    delta = prices.diff()
    gain = (delta.where(delta > 0, 0)).rolling(window=period).mean()
    loss = (-delta.where(delta < 0, 0)).rolling(window=period).mean()
    rs = gain / loss
    rsi = 100 - (100 / (1 + rs))
    return rsi

def calculate_macd(prices, fast=12, slow=26):
    """MACD (Moving Average Convergence Divergence)"""
    ema_fast = prices.ewm(span=fast, adjust=False).mean()
    ema_slow = prices.ewm(span=slow, adjust=False).mean()
    macd = ema_fast - ema_slow
    return macd

def calculate_bollinger_position(prices, period=20, std_dev=2):
    """Position within Bollinger Bands (0 to 1)"""
    sma = prices.rolling(window=period).mean()
    std = prices.rolling(window=period).std()
    upper_band = sma + (std * std_dev)
    lower_band = sma - (std * std_dev)
    bb_position = (prices - lower_band) / (upper_band - lower_band)
    return bb_position

def calculate_atr(high, low, close, period=14):
    """Average True Range (normalized)"""
    tr1 = high - low
    tr2 = abs(high - close.shift())
    tr3 = abs(low - close.shift())
    tr = pd.concat([tr1, tr2, tr3], axis=1).max(axis=1)
    atr = tr.rolling(window=period).mean()
    atr_normalized = atr / close
    return atr_normalized

print("Technical indicator functions defined")
```

```
Technical indicator functions defined
```

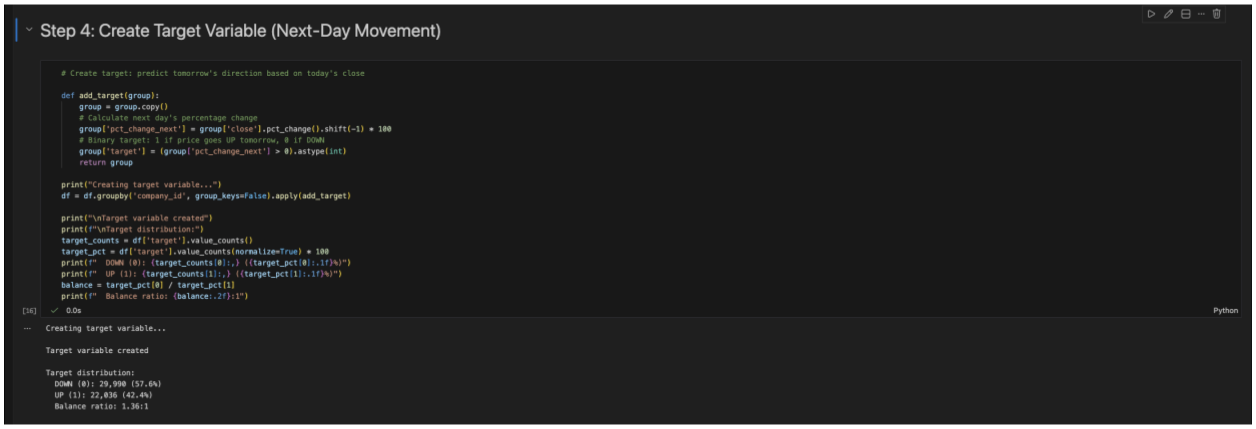
Figure 9: Calculating basic features and defining Technical Indicators Columns



```python
# Apply technical indicators per company
def add_technical_indicators(group):
    """Add technical indicators for a single company"""
    group = group.copy()

    # 1. RSI (14-day)
    group['rsi_14'] = calculate_rsi(group['close'], 14)

    # 2. MACD
    group['macd'] = calculate_macd(group['close'])

    # 3. Bollinger Band Position
    group['bb_position'] = calculate_bollinger_position(group['close'])

    # 4. ATR Normalized
    group['atr_normalized'] = calculate_atr(group['high'], group['low'], group['close'], 14)

    # 5. Volume Ratio
    group['volume_ma_5'] = group['traded_quantity'].rolling(5).mean()
    group['volume_ratio'] = group['traded_quantity'] / group['volume_ma_5']

    # 6. Return 5-day
    group['return_5d'] = group['close'].pct_change(5)

    # 7. Price to MA20
    group['price_to_ma20'] = group['close'] / group['ma_20']

    # 8. Trend Strength
    group['ma_60'] = group['close'].rolling(60).mean()
    group['trend_strength'] = (group['close'] - group['ma_60']) / group['ma_60']

    # Clean up temporary columns
    group = group.drop(['volume_ma_5', 'ma_60'], axis=1, errors='ignore')

    return group

print("Applying technical indicators per company...")
df = df.groupby('company_id', group_keys=False).apply(add_technical_indicators)
print(f"Shape after technical indicators: {df.shape}")
print(f"\nNew columns: {df.columns.tolist()}")
```

```
Applying technical indicators per company...
Shape after technical indicators: (53063, 20)

New columns: ['published_date', 'open', 'high', 'low', 'close', 'per_change', 'traded_quantity', 'traded_amount', 'company_id', 'daily_range', 'ma_5', 'ma_20', 'rsi_14', 'macd', 'bb_position', 'atr_normalized', 'volume_ratio', 'return_5d', 'price_to_ma20', 'trend_stre
```

Figure 10: Adding Technical Indicators Columns

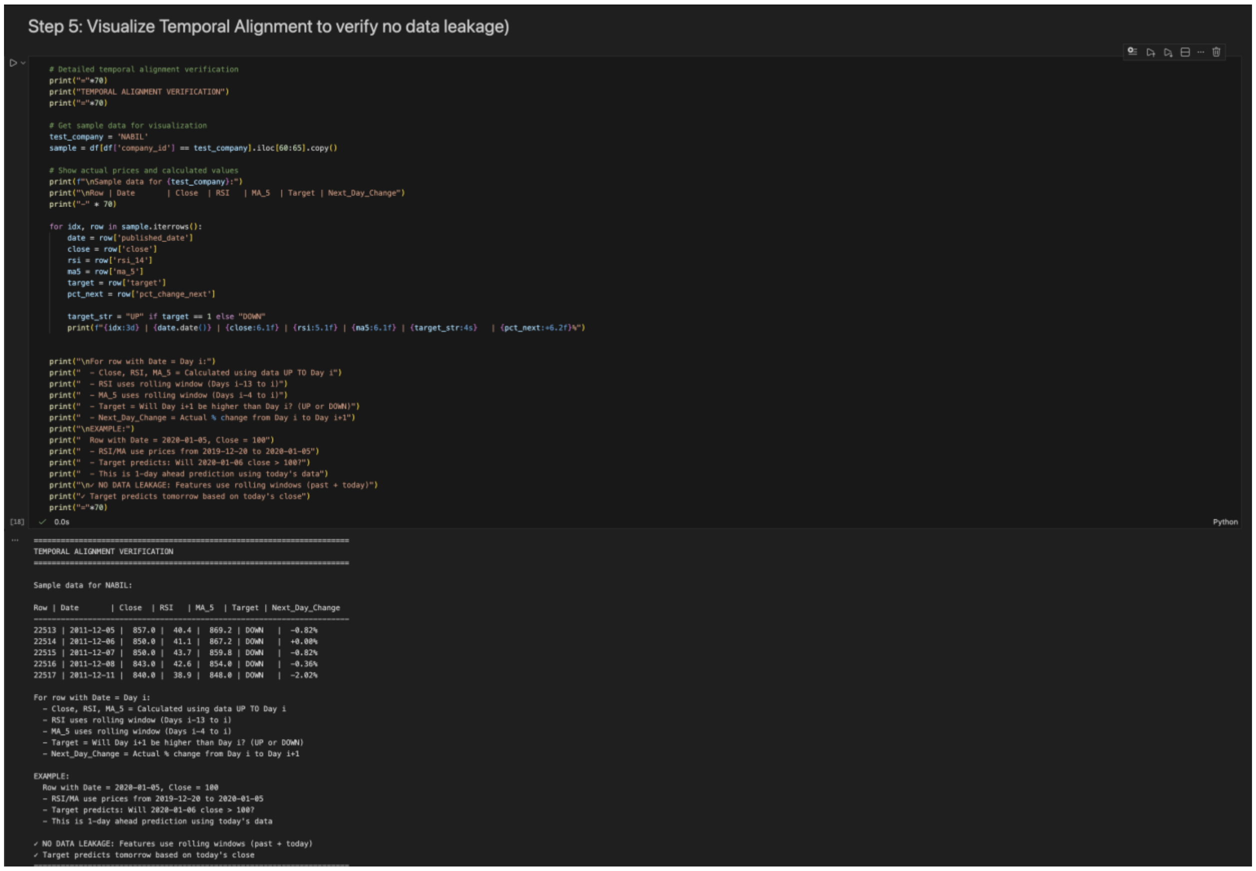


Step 4: Create Target Variable (Next-Day Movement)

```python
# Create target: predict tomorrow's direction based on today's close

def add_target(group):
    group = group.copy()
    # Calculate next day's percentage change
    group['pct_change_next'] = group['close'].pct_change().shift(-1) * 100
    # Binary target: 1 if price goes UP tomorrow, 0 if DOWN
    group['target'] = (group['pct_change_next'] > 0).astype(int)
    return group

print("Creating target variable...")
df = df.groupby('company_id', group_keys=False).apply(add_target)

print("\nTarget variable created")
print(f"\nTarget distribution:")
target_counts = df['target'].value_counts()
target_pct = df['target'].value_counts(normalize=True) * 100
print(f"  DOWN (0): {target_counts[0]:,} ({target_pct[0]:.1f}%)")
print(f"  UP (1): {target_counts[1]:,} ({target_pct[1]:.1f}%)")
balance = target_pct[0] / target_pct[1]
print(f"  Balance ratio: {balance:.2f}:1")
```

```
Creating target variable...

Target variable created

Target distribution:
  DOWN (0): 29,990 (57.6%)
  UP (1): 22,036 (42.4%)
  Balance ratio: 1.36:1
```

Figure 11: Creating target variables

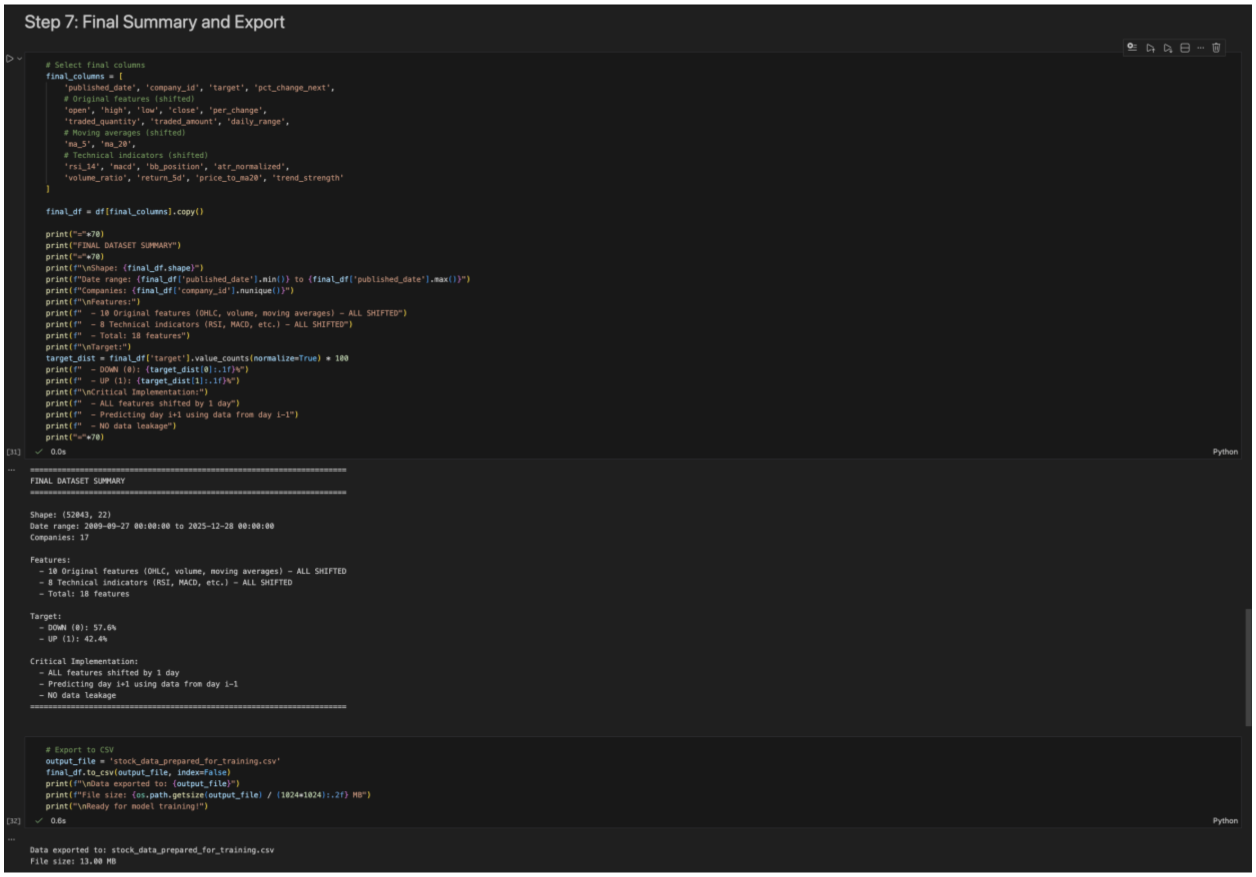


Step 5: Visualize Temporal Alignment to verify no data leakage)

```python
# Detailed temporal alignment verification
print("="*70)
print("TEMPORAL ALIGNMENT VERIFICATION")
print("="*70)

# Get sample data for visualization
test_company = 'NABIL'
sample = df[df['company_id'] == test_company].iloc[60:65].copy()

# Show actual prices and calculated values
print(f"\nSample data for {test_company}:")
print("\nRow | Date       | Close  | RSI   | MA_5  | Target | Next_Day_Change")
print("-" * 70)

for idx, row in sample.iterrows():
    date = row['published_date']
    close = row['close']
    rsi = row['rsi_14']
    ma5 = row['ma_5']
    target = row['target']
    pct_next = row['pct_change_next']

    target_str = "UP" if target == 1 else "DOWN"
    print(f"{idx:3d} | {date.date()} | {close:6.1f} | {rsi:5.1f} | {ma5:6.1f} | {target_str:4s}   | {pct_next:+6.2f}%")


print("\nFor row with Date = Day i:")
print("  - Close, RSI, MA_5 = Calculated using data UP TO Day i")
print("  - RSI uses rolling window (Days i-13 to i)")
print("  - MA_5 uses rolling window (Days i-4 to i)")
print("  - Target = Will Day i+1 be higher than Day i? (UP or DOWN)")
print("  - Next_Day_Change = Actual % change from Day i to Day i+1")
print("\nEXAMPLE:")
print("  Row with Date = 2020-01-05, Close = 100")
print("  - RSI/MA use prices from 2019-12-20 to 2020-01-05")
print("  - Target predicts: Will 2020-01-06 close > 100?")
print("  - This is 1-day ahead prediction using today's data")
print("\n✓ NO DATA LEAKAGE: Features use rolling windows (past + today)")
print("✓ Target predicts tomorrow based on today's close")
print("="*70)
```

```
======================================================================
TEMPORAL ALIGNMENT VERIFICATION
======================================================================

Sample data for NABIL:

Row | Date       | Close  | RSI   | MA_5  | Target | Next_Day_Change
----------------------------------------------------------------------
22513 | 2011-12-05 |  857.0 |  40.4 |  869.2 | DOWN   |  -0.82%
22514 | 2011-12-06 |  850.0 |  41.1 |  867.2 | DOWN   |  +0.00%
22515 | 2011-12-07 |  850.0 |  43.7 |  859.8 | DOWN   |  -0.82%
22516 | 2011-12-08 |  843.0 |  42.6 |  854.0 | DOWN   |  -0.36%
22517 | 2011-12-11 |  840.0 |  38.9 |  848.0 | DOWN   |  -2.02%

For row with Date = Day i:
  - Close, RSI, MA_5 = Calculated using data UP TO Day i
  - RSI uses rolling window (Days i-13 to i)
  - MA_5 uses rolling window (Days i-4 to i)
  - Target = Will Day i+1 be higher than Day i? (UP or DOWN)
  - Next_Day_Change = Actual % change from Day i to Day i+1

EXAMPLE:
  Row with Date = 2020-01-05, Close = 100
  - RSI/MA use prices from 2019-12-20 to 2020-01-05
  - Target predicts: Will 2020-01-06 close > 100?
  - This is 1-day ahead prediction using today's data

✓ NO DATA LEAKAGE: Features use rolling windows (past + today)
✓ Target predicts tomorrow based on today's close
```

Figure 12: Visualizing data alignment



Step 7: Final Summary and Export

```python
# Select final columns
final_columns = [
    'published_date', 'company_id', 'target', 'pct_change_next',
    # Original features (shifted)
    'open', 'high', 'low', 'close', 'per_change',
    'traded_quantity', 'traded_amount', 'daily_range',
    # Moving averages (shifted)
    'ma_5', 'ma_20',
    # Technical indicators (shifted)
    'rsi_14', 'macd', 'bb_position', 'atr_normalized',
    'volume_ratio', 'return_5d', 'price_to_ma20', 'trend_strength'
]

final_df = df[final_columns].copy()

print("="*70)
print("FINAL DATASET SUMMARY")
print("="*70)
print(f"\nShape: {final_df.shape}")
print(f"Date range: {final_df['published_date'].min()} to {final_df['published_date'].max()}")
print(f"Companies: {final_df['company_id'].nunique()}")
print(f"\nFeatures:")
print(f"  - 10 Original features (OHLC, volume, moving averages) - ALL SHIFTED")
print(f"  - 8 Technical indicators (RSI, MACD, etc.) - ALL SHIFTED")
print(f"  - Total: 18 features")
print(f"\nTarget:")
target_dist = final_df['target'].value_counts(normalize=True) * 100
print(f"  - DOWN (0): {target_dist[0]:.1f}%")
print(f"  - UP (1): {target_dist[1]:.1f}%")
print(f"\nCritical Implementation:")
print(f"  - ALL features shifted by 1 day")
print(f"  - Predicting day i+1 using data from day i-1")
print(f"  - NO data leakage")
print("="*70)
```

[31] ✓ 0.0s Python

```
======================================================================
FINAL DATASET SUMMARY
======================================================================

Shape: (52043, 22)
Date range: 2009-09-27 00:00:00 to 2025-12-28 00:00:00
Companies: 17

Features:
  - 10 Original features (OHLC, volume, moving averages) - ALL SHIFTED
  - 8 Technical indicators (RSI, MACD, etc.) - ALL SHIFTED
  - Total: 18 features

Target:
  - DOWN (0): 57.6%
  - UP (1): 42.4%

Critical Implementation:
  - ALL features shifted by 1 day
  - Predicting day i+1 using data from day i-1
  - NO data leakage
======================================================================
```

```python
# Export to CSV
output_file = 'stock_data_prepared_for_training.csv'
final_df.to_csv(output_file, index=False)
print(f"\nData exported to: {output_file}")
print(f"File size: {os.path.getsize(output_file) / (1024*1024):.2f} MB")
print("\nReady for model training!")
```

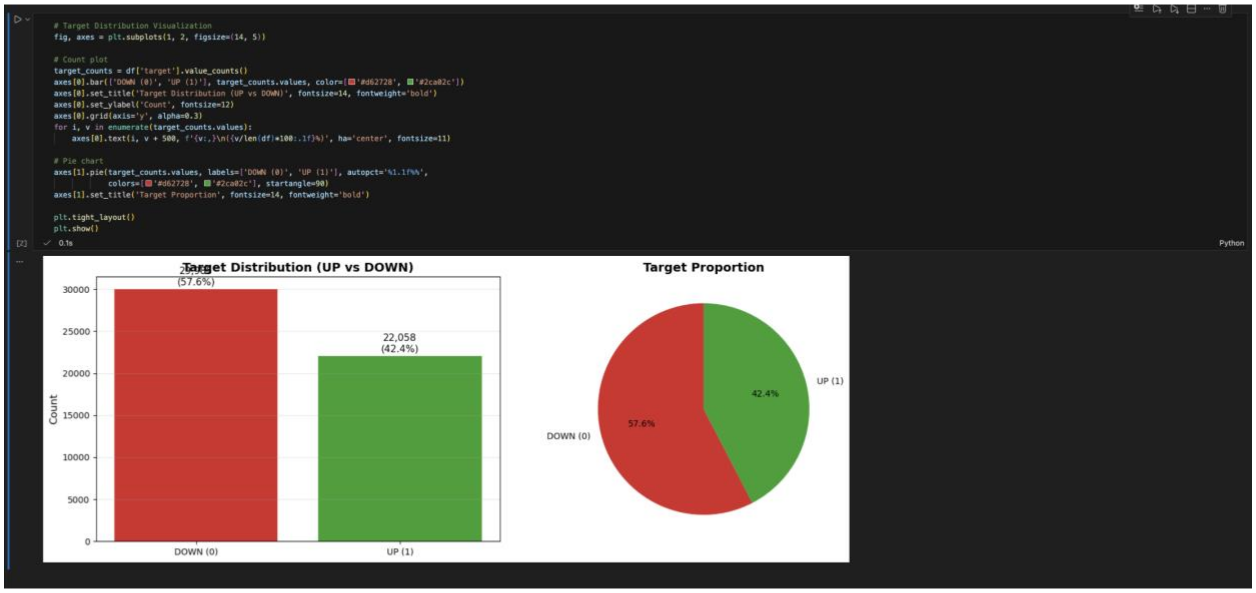
[32] ✓ 0.6s Python

```
Data exported to: stock_data_prepared_for_training.csv
File size: 13.00 MB
```

Figure 13:Final dataset summary and exporting them

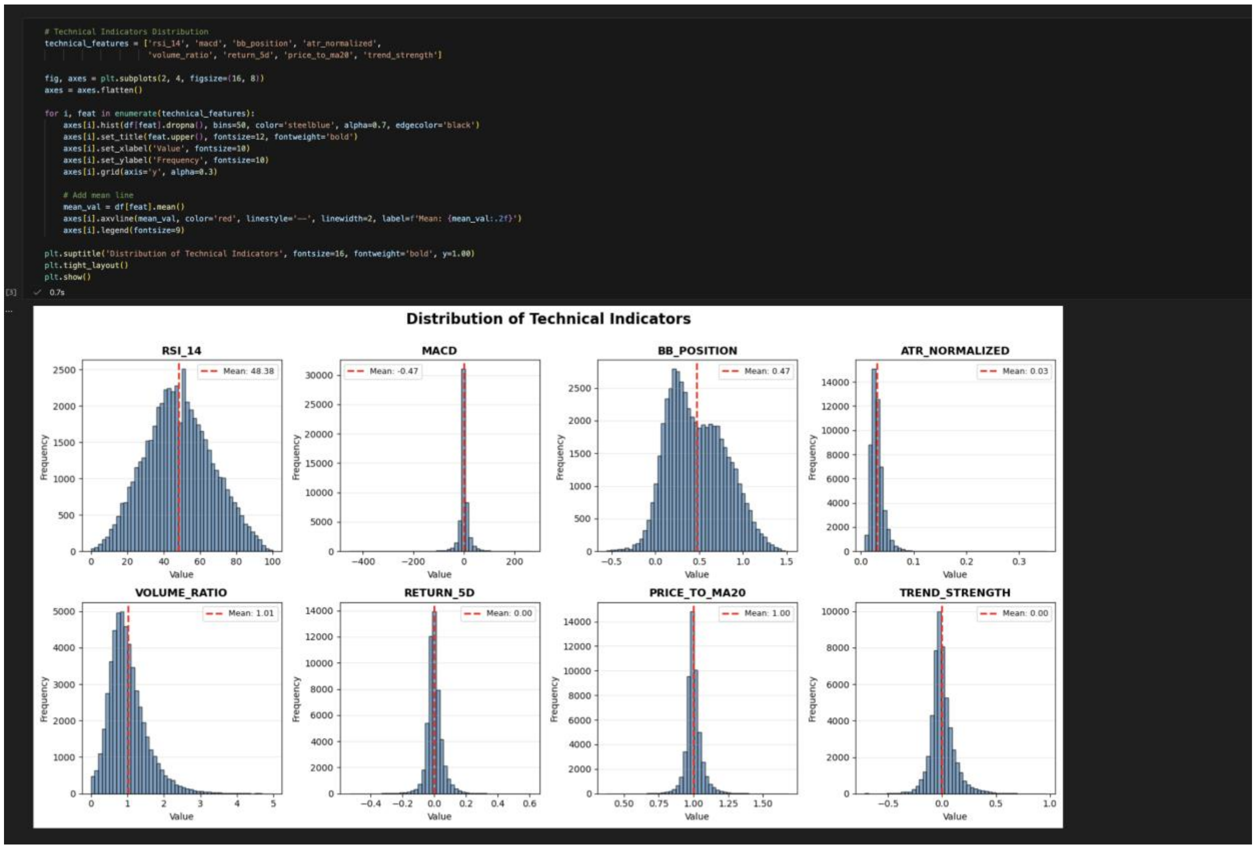
4.4.1.3 Dataset Visualization

To achieve an answer to the question of how NEPSE stock price models may be trained, the various exploratory visualizations would be incorporated to draw a rough outline of the data information. The data will be analyzed with the help of such visualizations in order to discover the distribution of the data and potential anomalies. The following visualizations are included:



```python
# Target Distribution Visualization
fig, axes = plt.subplots(1, 2, figsize=(14, 5))

# Count plot
target_counts = df['target'].value_counts()
axes[0].bar(['DOWN (0)', 'UP (1)'], target_counts.values, color=['#d62728', '#2ca02c'])
axes[0].set_title('Target Distribution (UP vs DOWN)', fontsize=14, fontweight='bold')
axes[0].set_ylabel('Count', fontsize=12)
axes[0].grid(axis='y', alpha=0.3)
for i, v in enumerate(target_counts.values):
    axes[0].text(i, v + 500, f'{v:,}\n({v/len(df)*100:.1f}%)', ha='center', fontsize=11)

# Pie chart
axes[1].pie(target_counts.values, labels=['DOWN (0)', 'UP (1)'], autopct='%1.1f%%',
            colors=['#d62728', '#2ca02c'], startangle=90)
axes[1].set_title('Target Proportion', fontsize=14, fontweight='bold')

plt.tight_layout()
plt.show()
```

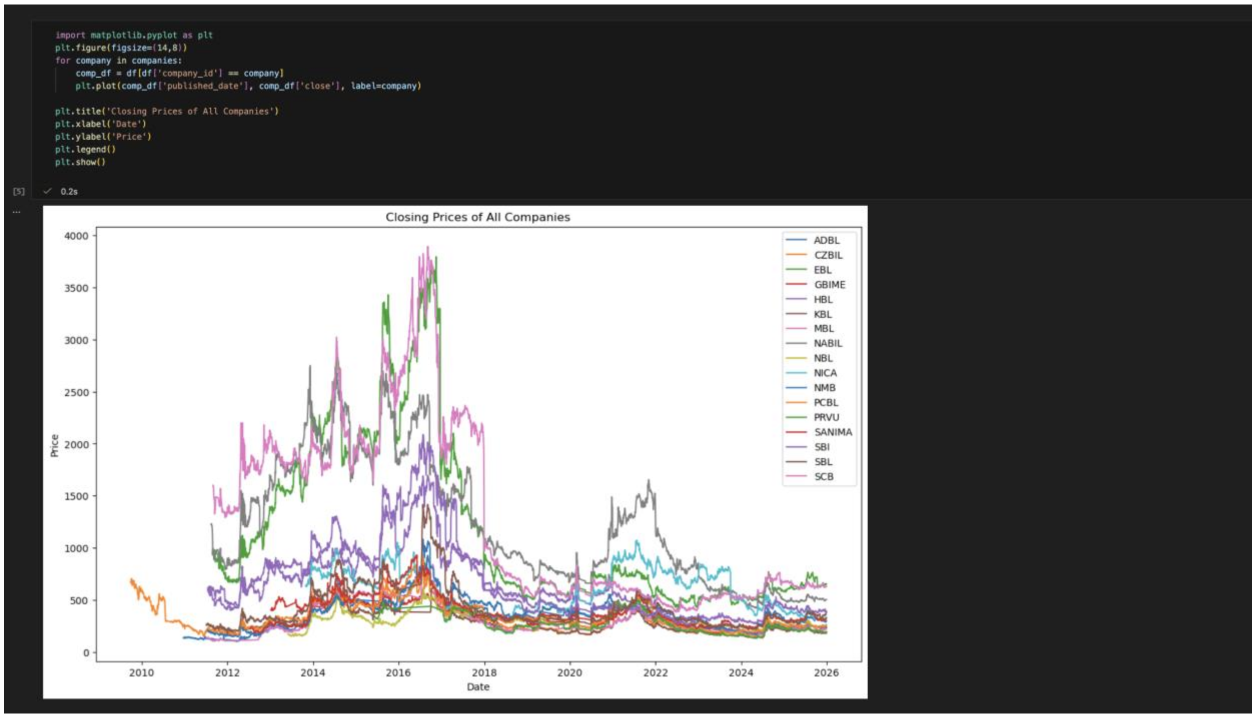
Figure 14: Target Distribution Visualization



```python
# Technical Indicators Distribution
technical_features = ['rsi_14', 'macd', 'bb_position', 'atr_normalized',
                      'volume_ratio', 'return_5d', 'price_to_ma20', 'trend_strength']

fig, axes = plt.subplots(2, 4, figsize=(16, 8))
axes = axes.flatten()

for i, feat in enumerate(technical_features):
    axes[i].hist(df[feat].dropna(), bins=50, color='steelblue', alpha=0.7, edgecolor='black')
    axes[i].set_title(feat.upper(), fontsize=12, fontweight='bold')
    axes[i].set_xlabel('Value', fontsize=10)
    axes[i].set_ylabel('Frequency', fontsize=10)
    axes[i].grid(axis='y', alpha=0.3)

    # Add mean line
    mean_val = df[feat].mean()
    axes[i].axvline(mean_val, color='red', linestyle='--', linewidth=2, label=f'Mean: {mean_val:.2f}')
    axes[i].legend(fontsize=9)

plt.suptitle('Distribution of Technical Indicators', fontsize=16, fontweight='bold', y=1.00)
plt.tight_layout()
plt.show()
```

```
[3]  ✓ 0.7s
```

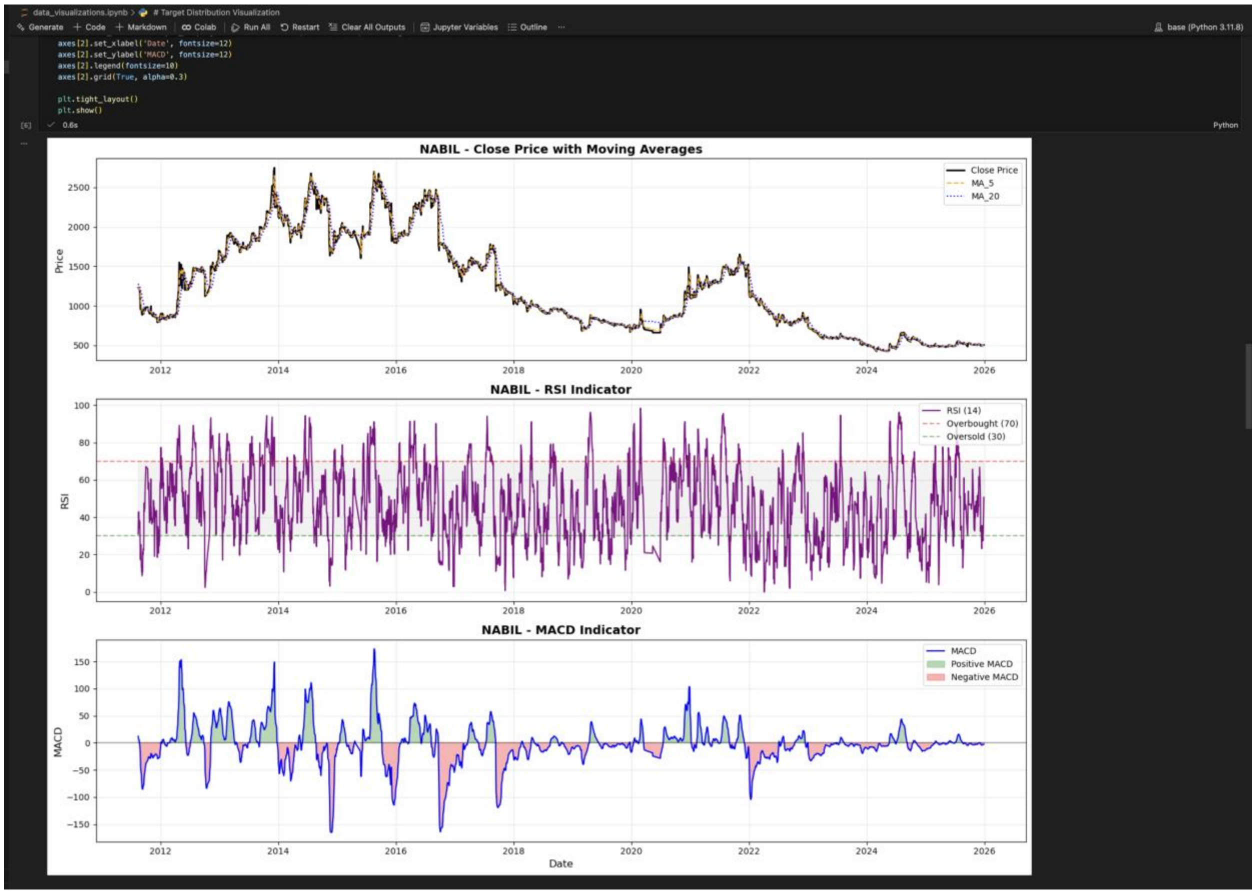
Figure 15: Technical indicators distribution



```python
import matplotlib.pyplot as plt
plt.figure(figsize=(14,8))
for company in companies:
    comp_df = df[df['company_id'] == company]
    plt.plot(comp_df['published_date'], comp_df['close'], label=company)

plt.title('Closing Prices of All Companies')
plt.xlabel('Date')
plt.ylabel('Price')
plt.legend()
plt.show()
```

```
[5]  ✓ 0.2s
```

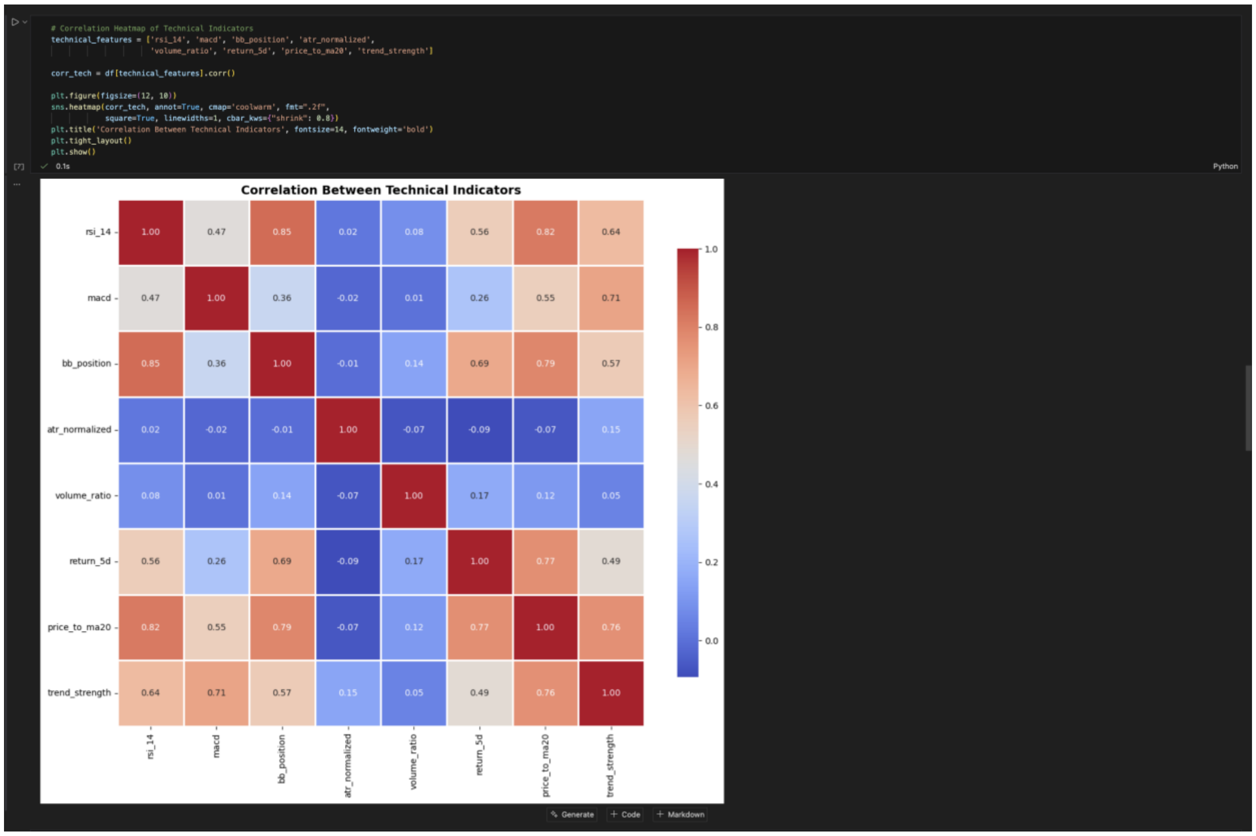
Figure 16: Visualization of closing price of all companies



```
axes[2].set_xlabel('Date', fontsize=12)
axes[2].set_ylabel('MACD', fontsize=12)
axes[2].legend(fontsize=10)
axes[2].grid(True, alpha=0.3)

plt.tight_layout()
plt.show()
```

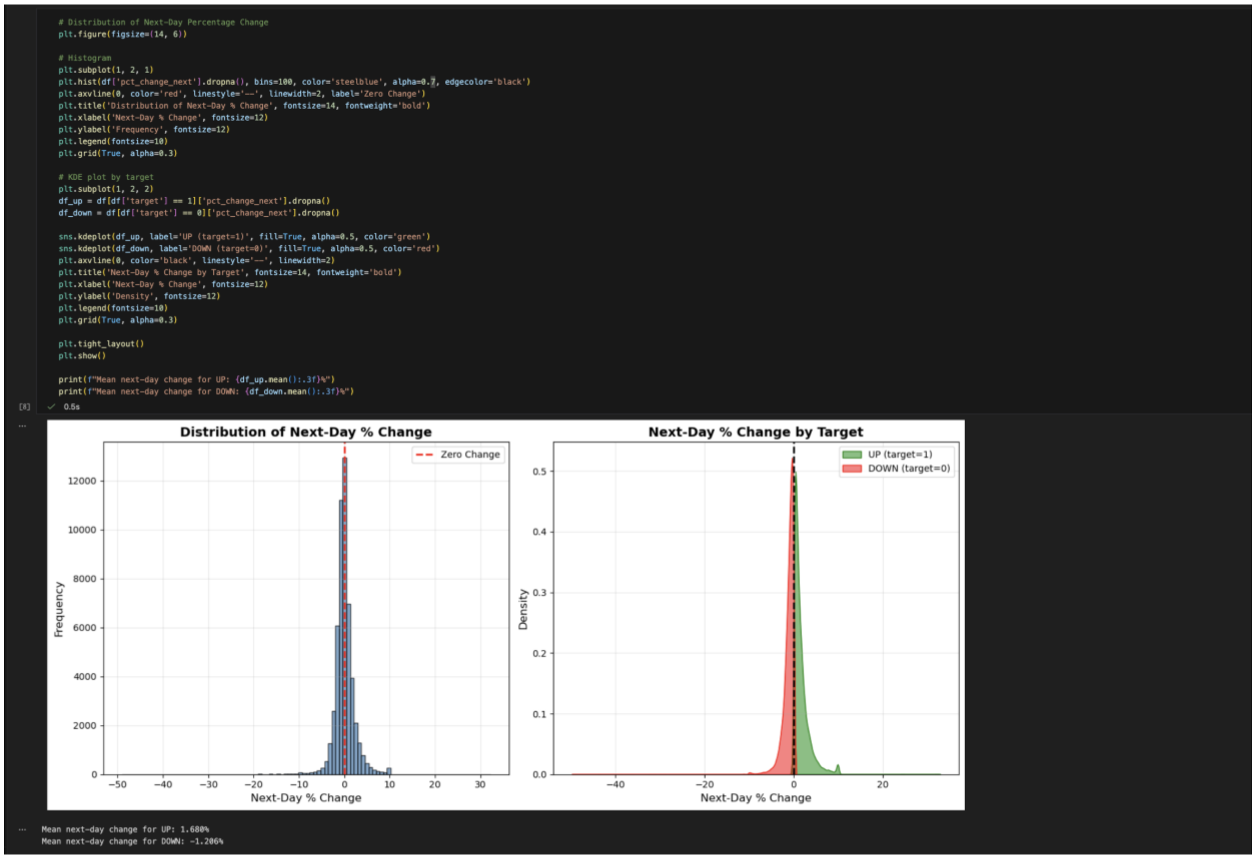
```
[6] ✓ 0.6s
```

Figure 17: RSI and MACD Visualization of NABIL Stock



```
# Correlation Heatmap of Technical Indicators
technical_features = ['rsi_14', 'macd', 'bb_position', 'atr_normalized',
                      'volume_ratio', 'return_5d', 'price_to_ma20', 'trend_strength']

corr_tech = df[technical_features].corr()

plt.figure(figsize=(12, 10))
sns.heatmap(corr_tech, annot=True, cmap='coolwarm', fmt=".2f",
            square=True, linewidths=1, cbar_kws={"shrink": 0.8})
plt.title('Correlation Between Technical Indicators', fontsize=14, fontweight='bold')
plt.tight_layout()
plt.show()
```

Figure 18: Technical indicators correlation

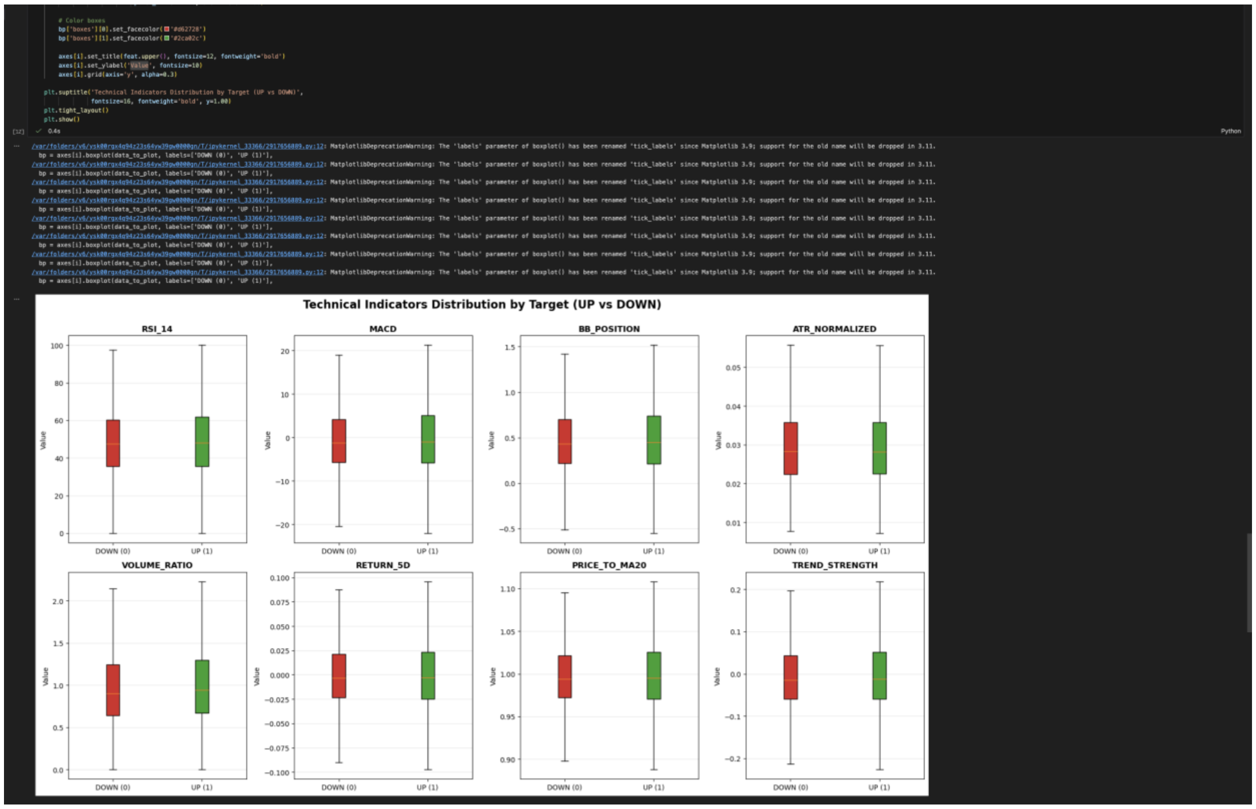


```python
# Distribution of Next-Day Percentage Change
plt.figure(figsize=(14, 6))

# Histogram
plt.subplot(1, 2, 1)
plt.hist(df['pct_change_next'].dropna(), bins=100, color='steelblue', alpha=0.7, edgecolor='black')
plt.axvline(0, color='red', linestyle='--', linewidth=2, label='Zero Change')
plt.title('Distribution of Next-Day % Change', fontsize=14, fontweight='bold')
plt.xlabel('Next-Day % Change', fontsize=12)
plt.ylabel('Frequency', fontsize=12)
plt.legend(fontsize=10)
plt.grid(True, alpha=0.3)

# KDE plot by target
plt.subplot(1, 2, 2)
df_up = df[df['target'] == 1]['pct_change_next'].dropna()
df_down = df[df['target'] == 0]['pct_change_next'].dropna()

sns.kdeplot(df_up, label='UP (target=1)', fill=True, alpha=0.5, color='green')
sns.kdeplot(df_down, label='DOWN (target=0)', fill=True, alpha=0.5, color='red')
plt.axvline(0, color='black', linestyle='--', linewidth=2)
plt.title('Next-Day % Change by Target', fontsize=14, fontweight='bold')
plt.xlabel('Next-Day % Change', fontsize=12)
plt.ylabel('Density', fontsize=12)
plt.legend(fontsize=10)
plt.grid(True, alpha=0.3)

plt.tight_layout()
plt.show()

print(f"Mean next-day change for UP: {df_up.mean():.3f}%")
print(f"Mean next-day change for DOWN: {df_down.mean():.3f}%")
```

[8] ✓ 0.5s

```
Mean next-day change for UP: 1.680%
Mean next-day change for DOWN: -1.206%
```

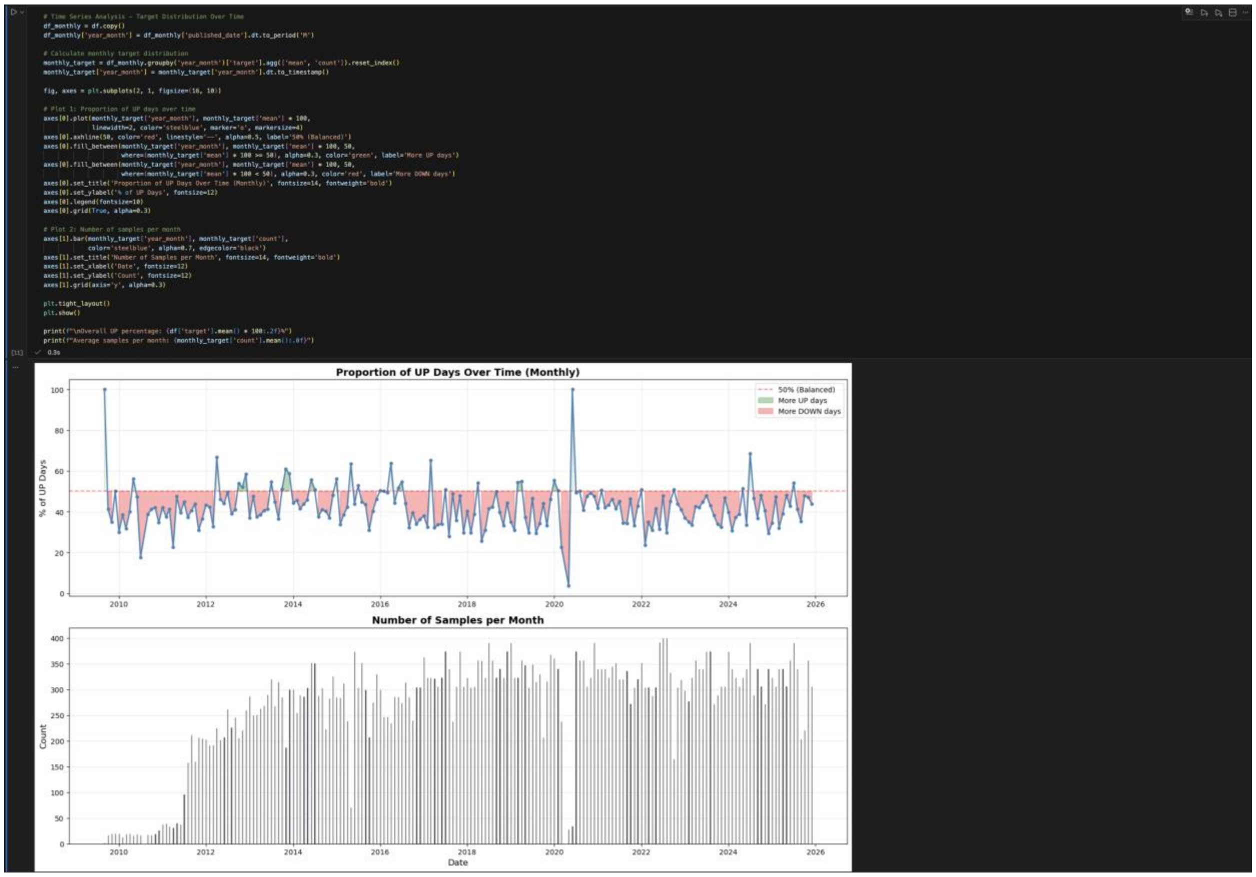
Figure 19: Distribution of next day % change



```
        # Color boxes
        bp['boxes'][0].set_facecolor('#d62728')
        bp['boxes'][1].set_facecolor('#2ca02c')

        axes[i].set_title(feat.upper(), fontsize=12, fontweight='bold')
        axes[i].set_ylabel('Value', fontsize=10)
        axes[i].grid(axis='y', alpha=0.3)

    plt.suptitle('Technical Indicators Distribution by Target (UP vs DOWN)',
                 fontsize=16, fontweight='bold', y=1.00)
    plt.tight_layout()
    plt.show()
```

```
/var/folders/v6/ysk00rgx4q94z23s64yw39gw0000gn/T/ipykernel_33366/2917656809.py:12: MatplotlibDeprecationWarning: The 'labels' parameter of boxplot() has been renamed 'tick_labels' since Matplotlib 3.9; support for the old name will be dropped in 3.11.
  bp = axes[i].boxplot(data_to_plot, labels=['DOWN (0)', 'UP (1)'],
/var/folders/v6/ysk00rgx4q94z23s64yw39gw0000gn/T/ipykernel_33366/2917656809.py:12: MatplotlibDeprecationWarning: The 'labels' parameter of boxplot() has been renamed 'tick_labels' since Matplotlib 3.9; support for the old name will be dropped in 3.11.
  bp = axes[i].boxplot(data_to_plot, labels=['DOWN (0)', 'UP (1)'],
/var/folders/v6/ysk00rgx4q94z23s64yw39gw0000gn/T/ipykernel_33366/2917656809.py:12: MatplotlibDeprecationWarning: The 'labels' parameter of boxplot() has been renamed 'tick_labels' since Matplotlib 3.9; support for the old name will be dropped in 3.11.
  bp = axes[i].boxplot(data_to_plot, labels=['DOWN (0)', 'UP (1)'],
/var/folders/v6/ysk00rgx4q94z23s64yw39gw0000gn/T/ipykernel_33366/2917656809.py:12: MatplotlibDeprecationWarning: The 'labels' parameter of boxplot() has been renamed 'tick_labels' since Matplotlib 3.9; support for the old name will be dropped in 3.11.
  bp = axes[i].boxplot(data_to_plot, labels=['DOWN (0)', 'UP (1)'],
/var/folders/v6/ysk00rgx4q94z23s64yw39gw0000gn/T/ipykernel_33366/2917656809.py:12: MatplotlibDeprecationWarning: The 'labels' parameter of boxplot() has been renamed 'tick_labels' since Matplotlib 3.9; support for the old name will be dropped in 3.11.
  bp = axes[i].boxplot(data_to_plot, labels=['DOWN (0)', 'UP (1)'],
/var/folders/v6/ysk00rgx4q94z23s64yw39gw0000gn/T/ipykernel_33366/2917656809.py:12: MatplotlibDeprecationWarning: The 'labels' parameter of boxplot() has been renamed 'tick_labels' since Matplotlib 3.9; support for the old name will be dropped in 3.11.
  bp = axes[i].boxplot(data_to_plot, labels=['DOWN (0)', 'UP (1)'],
/var/folders/v6/ysk00rgx4q94z23s64yw39gw0000gn/T/ipykernel_33366/2917656809.py:12: MatplotlibDeprecationWarning: The 'labels' parameter of boxplot() has been renamed 'tick_labels' since Matplotlib 3.9; support for the old name will be dropped in 3.11.
  bp = axes[i].boxplot(data_to_plot, labels=['DOWN (0)', 'UP (1)'],
/var/folders/v6/ysk00rgx4q94z23s64yw39gw0000gn/T/ipykernel_33366/2917656809.py:12: MatplotlibDeprecationWarning: The 'labels' parameter of boxplot() has been renamed 'tick_labels' since Matplotlib 3.9; support for the old name will be dropped in 3.11.
  bp = axes[i].boxplot(data_to_plot, labels=['DOWN (0)', 'UP (1)'],
```

Figure 20: Technical indicators distribution by target



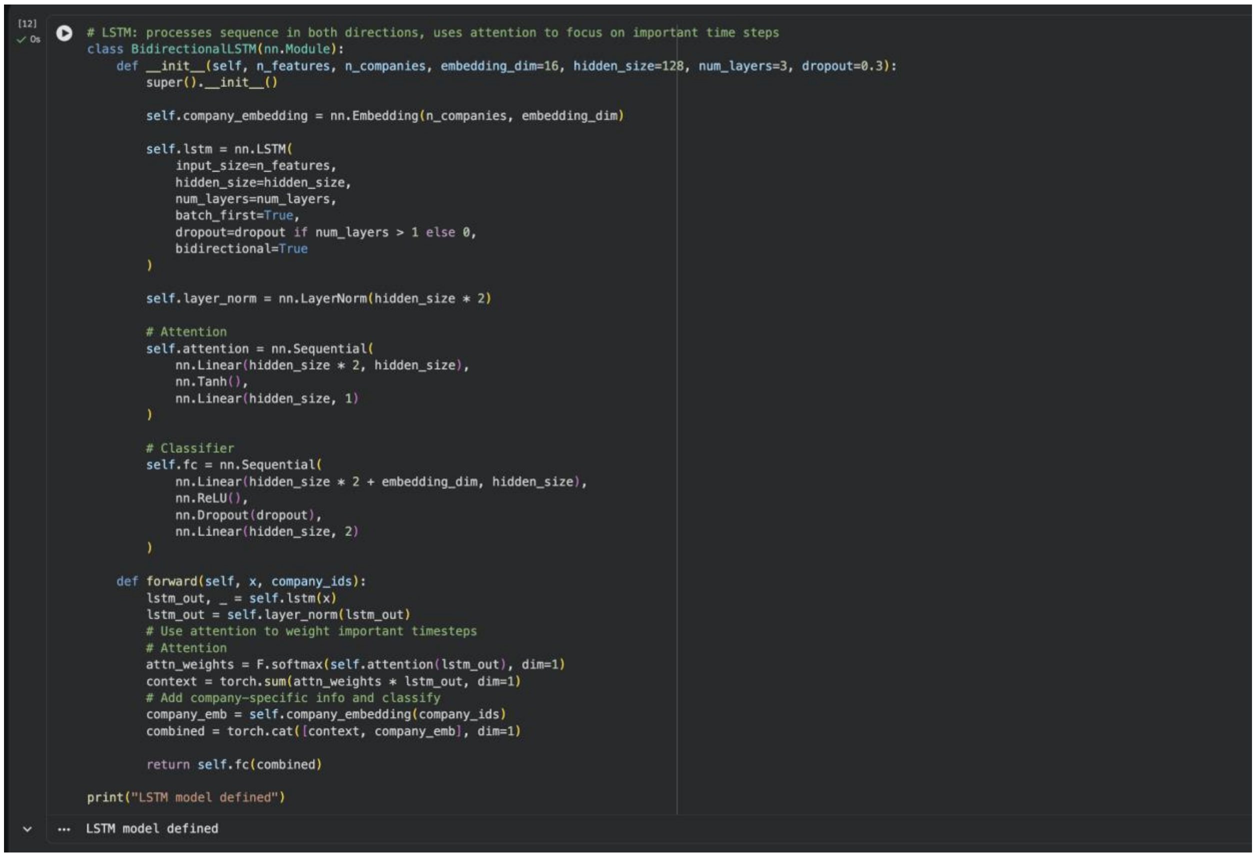
```python
# Time Series Analysis - Target Distribution Over Time
df_monthly = df.copy()
df_monthly['year_month'] = df_monthly['published_date'].dt.to_period('M')

# Calculate monthly target distribution
monthly_target = df_monthly.groupby('year_month')['target'].agg(['mean', 'count']).reset_index()
monthly_target['year_month'] = monthly_target['year_month'].dt.to_timestamp()

fig, axes = plt.subplots(2, 1, figsize=(16, 10))

# Plot 1: Proportion of UP days over time
axes[0].plot(monthly_target['year_month'], monthly_target['mean'] * 100,
             linewidth=2, color='steelblue', marker='o', markersize=4)
axes[0].axhline(50, color='red', linestyle='--', alpha=0.5, label='50% (Balanced)')
axes[0].fill_between(monthly_target['year_month'], monthly_target['mean'] * 100, 50,
                     where=(monthly_target['mean'] * 100 >= 50), alpha=0.3, color='green', label='More UP days')
axes[0].fill_between(monthly_target['year_month'], monthly_target['mean'] * 100, 50,
                     where=(monthly_target['mean'] * 100 < 50), alpha=0.3, color='red', label='More DOWN days')
axes[0].set_title('Proportion of UP Days Over Time (Monthly)', fontsize=14, fontweight='bold')
axes[0].set_ylabel('% of UP Days', fontsize=12)
axes[0].legend(fontsize=10)
axes[0].grid(True, alpha=0.3)

# Plot 2: Number of samples per month
axes[1].bar(monthly_target['year_month'], monthly_target['count'],
            color='steelblue', alpha=0.7, edgecolor='black')
axes[1].set_title('Number of Samples per Month', fontsize=14, fontweight='bold')
axes[1].set_xlabel('Date', fontsize=12)
axes[1].set_ylabel('Count', fontsize=12)
axes[1].grid(axis='y', alpha=0.3)

plt.tight_layout()
plt.show()

print(f"\nOverall UP percentage: {df['target'].mean() * 100:.2f}%")
print(f"Average samples per month: {monthly_target['count'].mean():.0f}")
```

[11] ✓ 0.3s

Figure 21: Target distribution by time(month)

4.4.1.4 Model Implementation

- **LSTM:** Bidirectional, 3 layers, 128 units, attention mechanism

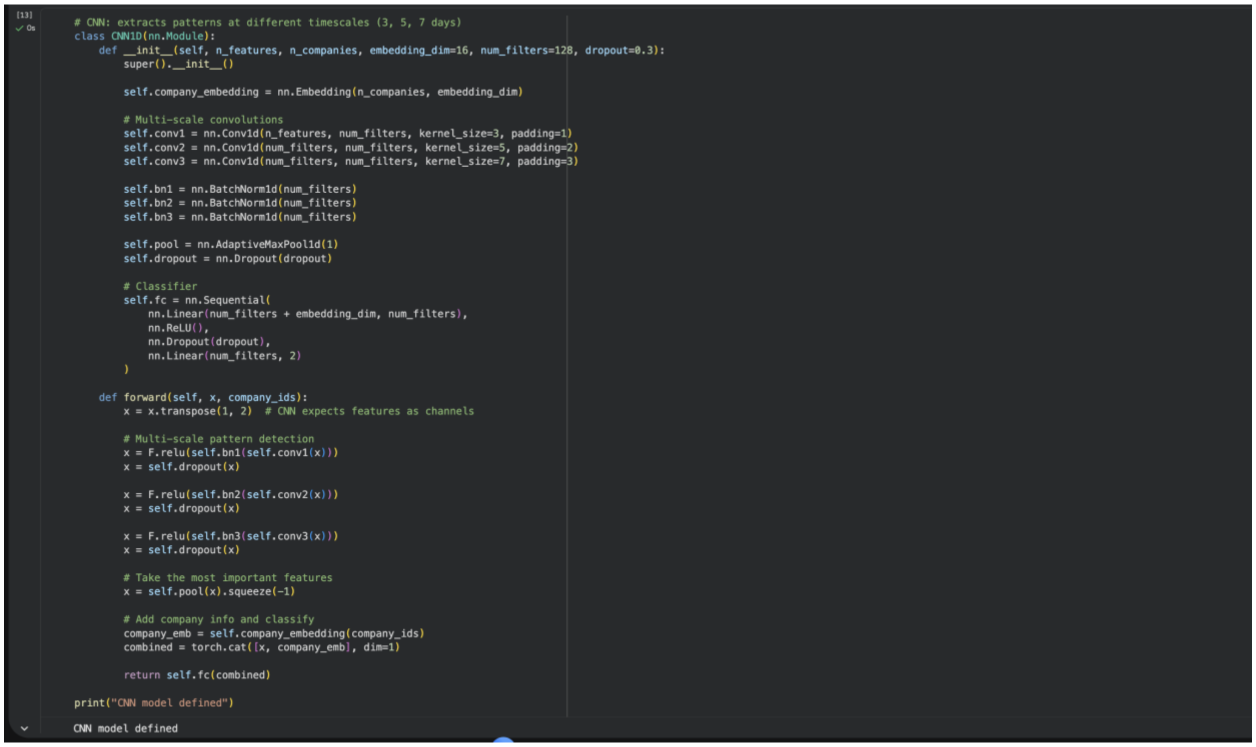


```
# LSTM: processes sequence in both directions, uses attention to focus on important time steps
class BidirectionalLSTM(nn.Module):
    def __init__(self, n_features, n_companies, embedding_dim=16, hidden_size=128, num_layers=3, dropout=0.3):
        super().__init__()

        self.company_embedding = nn.Embedding(n_companies, embedding_dim)

        self.lstm = nn.LSTM(
            input_size=n_features,
            hidden_size=hidden_size,
            num_layers=num_layers,
            batch_first=True,
            dropout=dropout if num_layers > 1 else 0,
            bidirectional=True
        )

        self.layer_norm = nn.LayerNorm(hidden_size * 2)

        # Attention
        self.attention = nn.Sequential(
            nn.Linear(hidden_size * 2, hidden_size),
            nn.Tanh(),
            nn.Linear(hidden_size, 1)
        )

        # Classifier
        self.fc = nn.Sequential(
            nn.Linear(hidden_size * 2 + embedding_dim, hidden_size),
            nn.ReLU(),
            nn.Dropout(dropout),
            nn.Linear(hidden_size, 2)
        )

    def forward(self, x, company_ids):
        lstm_out, _ = self.lstm(x)
        lstm_out = self.layer_norm(lstm_out)
        # Use attention to weight important timesteps
        # Attention
        attn_weights = F.softmax(self.attention(lstm_out), dim=1)
        context = torch.sum(attn_weights * lstm_out, dim=1)
        # Add company-specific info and classify
        company_emb = self.company_embedding(company_ids)
        combined = torch.cat([context, company_emb], dim=1)

        return self.fc(combined)

print("LSTM model defined")
```

```
LSTM model defined
```

Figure 22: LSTM Model Architecture

- **CNN:** Multi-scale convolutions (kernels 3, 5, 7), 128 filters each

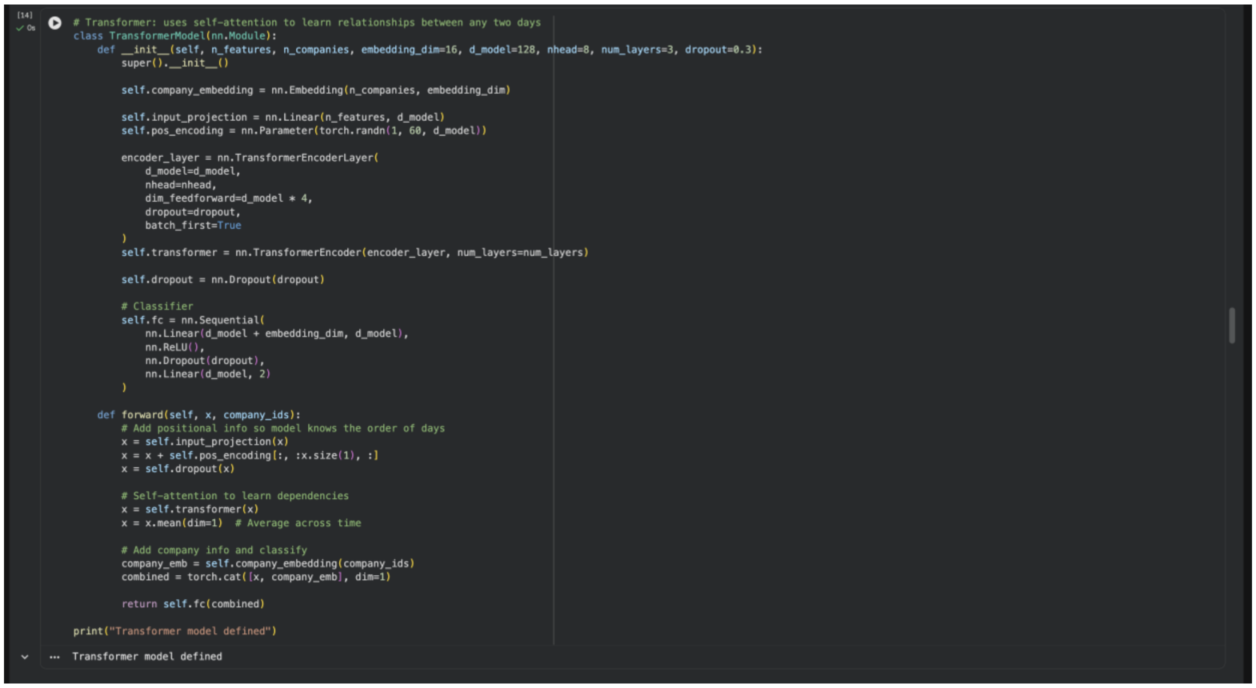


```
[13]
✓ 0s
# CNN: extracts patterns at different timescales (3, 5, 7 days)
class CNN1D(nn.Module):
    def __init__(self, n_features, n_companies, embedding_dim=16, num_filters=128, dropout=0.3):
        super().__init__()

        self.company_embedding = nn.Embedding(n_companies, embedding_dim)

        # Multi-scale convolutions
        self.conv1 = nn.Conv1d(n_features, num_filters, kernel_size=3, padding=1)
        self.conv2 = nn.Conv1d(num_filters, num_filters, kernel_size=5, padding=2)
        self.conv3 = nn.Conv1d(num_filters, num_filters, kernel_size=7, padding=3)

        self.bn1 = nn.BatchNorm1d(num_filters)
        self.bn2 = nn.BatchNorm1d(num_filters)
        self.bn3 = nn.BatchNorm1d(num_filters)

        self.pool = nn.AdaptiveMaxPool1d(1)
        self.dropout = nn.Dropout(dropout)

        # Classifier
        self.fc = nn.Sequential(
            nn.Linear(num_filters + embedding_dim, num_filters),
            nn.ReLU(),
            nn.Dropout(dropout),
            nn.Linear(num_filters, 2)
        )

    def forward(self, x, company_ids):
        x = x.transpose(1, 2)  # CNN expects features as channels

        # Multi-scale pattern detection
        x = F.relu(self.bn1(self.conv1(x)))
        x = self.dropout(x)

        x = F.relu(self.bn2(self.conv2(x)))
        x = self.dropout(x)

        x = F.relu(self.bn3(self.conv3(x)))
        x = self.dropout(x)

        # Take the most important features
        x = self.pool(x).squeeze(-1)

        # Add company info and classify
        company_emb = self.company_embedding(company_ids)
        combined = torch.cat([x, company_emb], dim=1)

        return self.fc(combined)

print("CNN model defined")
```

```
CNN model defined
```

Figure 23: CNN Model Architecture

- **Transformer:** 3 encoder layers, 8 attention heads, d_model=128



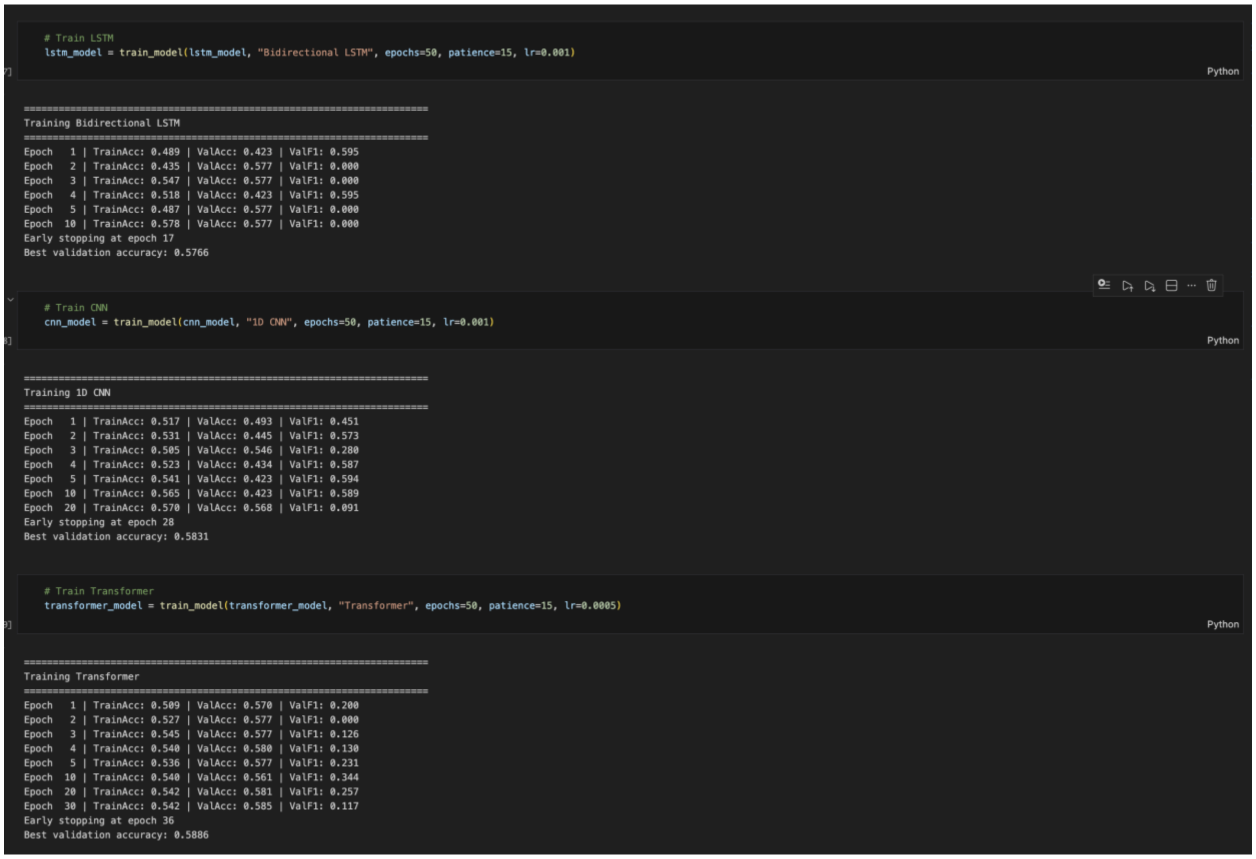
```
[14]
# Transformer: uses self-attention to learn relationships between any two days
class TransformerModel(nn.Module):
    def __init__(self, n_features, n_companies, embedding_dim=16, d_model=128, nhead=8, num_layers=3, dropout=0.3):
        super().__init__()

        self.company_embedding = nn.Embedding(n_companies, embedding_dim)

        self.input_projection = nn.Linear(n_features, d_model)
        self.pos_encoding = nn.Parameter(torch.randn(1, 60, d_model))

        encoder_layer = nn.TransformerEncoderLayer(
            d_model=d_model,
            nhead=nhead,
            dim_feedforward=d_model * 4,
            dropout=dropout,
            batch_first=True
        )
        self.transformer = nn.TransformerEncoder(encoder_layer, num_layers=num_layers)

        self.dropout = nn.Dropout(dropout)

        # Classifier
        self.fc = nn.Sequential(
            nn.Linear(d_model + embedding_dim, d_model),
            nn.ReLU(),
            nn.Dropout(dropout),
            nn.Linear(d_model, 2)
        )

    def forward(self, x, company_ids):
        # Add positional info so model knows the order of days
        x = self.input_projection(x)
        x = x + self.pos_encoding[:, :x.size(1), :]
        x = self.dropout(x)

        # Self-attention to learn dependencies
        x = self.transformer(x)
        x = x.mean(dim=1)  # Average across time

        # Add company info and classify
        company_emb = self.company_embedding(company_ids)
        combined = torch.cat([x, company_emb], dim=1)

        return self.fc(combined)

print("Transformer model defined")
```

```
Transformer model defined
```

Figure 24: Transformer Model Architecture

4.4.1.5 Training Configuration

- Computed class weights
- Loss function: CrossEntropyLoss with weights
- Optimizer: AdamW (lr=0.001 for LSTM/CNN, lr=0.0005 for Transformer)
- Learning rate scheduler: ReduceLROnPlateau (patience=5)
- Early stopping
- Regularization: Dropout (0.3), gradient clipping (max_norm=1.0)



```python
# Train LSTM
lstm_model = train_model(lstm_model, "Bidirectional LSTM", epochs=50, patience=15, lr=0.001)
```

Python

```
======================================================================
Training Bidirectional LSTM
======================================================================
Epoch   1 | TrainAcc: 0.489 | ValAcc: 0.423 | ValF1: 0.595
Epoch   2 | TrainAcc: 0.435 | ValAcc: 0.577 | ValF1: 0.000
Epoch   3 | TrainAcc: 0.547 | ValAcc: 0.577 | ValF1: 0.000
Epoch   4 | TrainAcc: 0.518 | ValAcc: 0.423 | ValF1: 0.595
Epoch   5 | TrainAcc: 0.487 | ValAcc: 0.577 | ValF1: 0.000
Epoch  10 | TrainAcc: 0.578 | ValAcc: 0.577 | ValF1: 0.000
Early stopping at epoch 17
Best validation accuracy: 0.5766
```

```python
# Train CNN
cnn_model = train_model(cnn_model, "1D CNN", epochs=50, patience=15, lr=0.001)
```

Python

```
======================================================================
Training 1D CNN
======================================================================
Epoch   1 | TrainAcc: 0.517 | ValAcc: 0.493 | ValF1: 0.451
Epoch   2 | TrainAcc: 0.531 | ValAcc: 0.445 | ValF1: 0.573
Epoch   3 | TrainAcc: 0.505 | ValAcc: 0.546 | ValF1: 0.280
Epoch   4 | TrainAcc: 0.523 | ValAcc: 0.434 | ValF1: 0.587
Epoch   5 | TrainAcc: 0.541 | ValAcc: 0.423 | ValF1: 0.594
Epoch  10 | TrainAcc: 0.565 | ValAcc: 0.423 | ValF1: 0.589
Epoch  20 | TrainAcc: 0.570 | ValAcc: 0.568 | ValF1: 0.091
Early stopping at epoch 28
Best validation accuracy: 0.5831
```

```python
# Train Transformer
transformer_model = train_model(transformer_model, "Transformer", epochs=50, patience=15, lr=0.0005)
```

Python

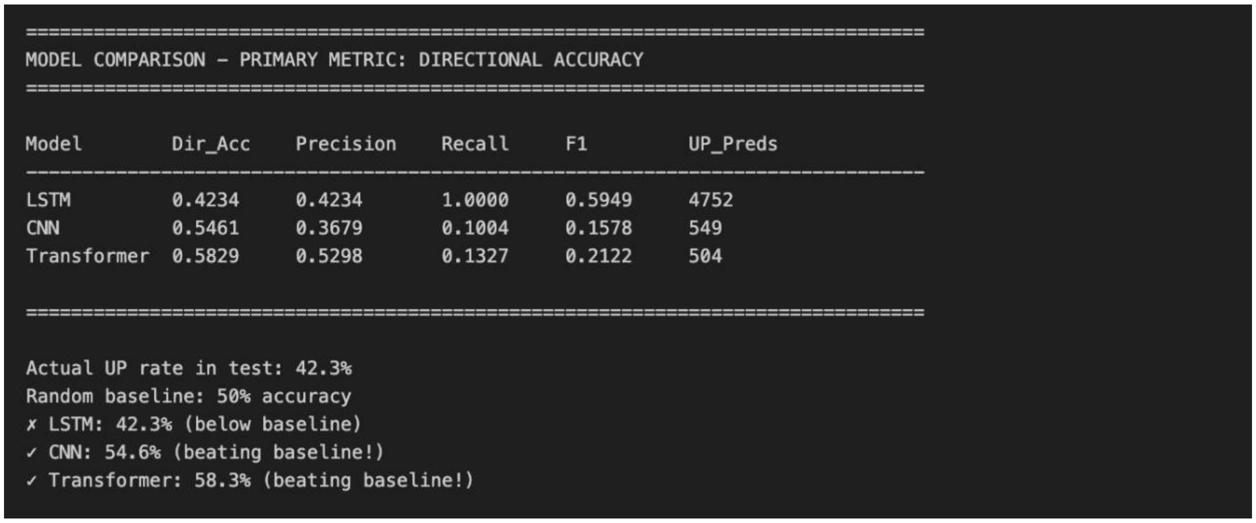
```
======================================================================
Training Transformer
======================================================================
Epoch   1 | TrainAcc: 0.509 | ValAcc: 0.570 | ValF1: 0.200
Epoch   2 | TrainAcc: 0.527 | ValAcc: 0.577 | ValF1: 0.000
Epoch   3 | TrainAcc: 0.545 | ValAcc: 0.577 | ValF1: 0.126
Epoch   4 | TrainAcc: 0.540 | ValAcc: 0.580 | ValF1: 0.130
Epoch   5 | TrainAcc: 0.536 | ValAcc: 0.577 | ValF1: 0.231
Epoch  10 | TrainAcc: 0.540 | ValAcc: 0.561 | ValF1: 0.344
Epoch  20 | TrainAcc: 0.542 | ValAcc: 0.581 | ValF1: 0.257
Epoch  30 | TrainAcc: 0.542 | ValAcc: 0.585 | ValF1: 0.117
Early stopping at epoch 36
Best validation accuracy: 0.5886
```

Figure 25: Model Training

4.4.1.6 Evaluation

- Generated predictions on test set
- Calculated metrics: Accuracy, Precision, Recall, F1-Score
- Created confusion matrices
- Compared against 50% baseline
- Statistical significance testing

4.4.1.7 Final Test Results



```
================================================================================
MODEL COMPARISON - PRIMARY METRIC: DIRECTIONAL ACCURACY
================================================================================

Model        Dir_Acc    Precision    Recall     F1         UP_Preds
--------------------------------------------------------------------------------
LSTM         0.4234     0.4234       1.0000     0.5949     4752
CNN          0.5461     0.3679       0.1004     0.1578     549
Transformer  0.5829     0.5298       0.1327     0.2122     504

================================================================================

Actual UP rate in test: 42.3%
Random baseline: 50% accuracy
✗ LSTM: 42.3% (below baseline)
✓ CNN: 54.6% (beating baseline!)
✓ Transformer: 58.3% (beating baseline!)
```

Figure 26: FiInal Test Results

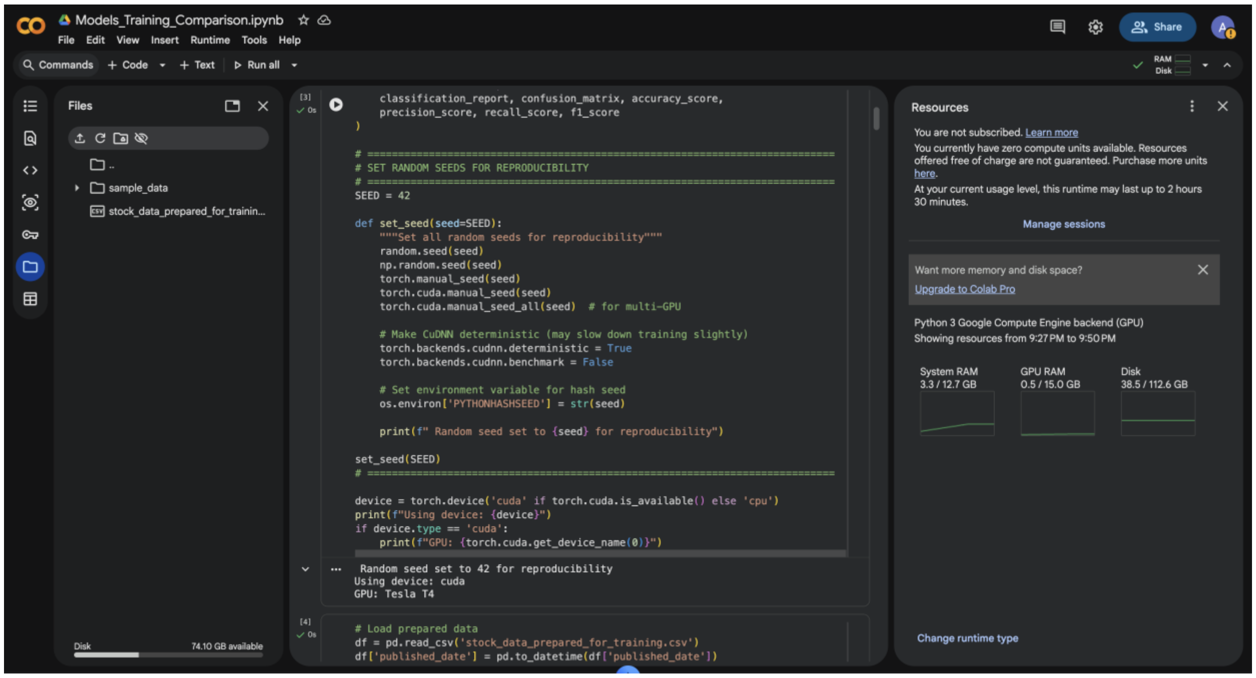
4.4.2 Tools and Technologies

4.4.2.1 Programming:

- Python 3.9+

4.4.2.2 Primary Development Platform: Google Colab

Google Colab was selected as the primary development environment for several key reasons. It provides free access to NVIDIA Tesla T4 GPU with 16GB VRAM, eliminating the need for expensive local hardware. The platform comes with pre-installed deep learning frameworks (PyTorch, TensorFlow), saving setup time and preventing version conflicts. The browser-based Jupyter notebook interface enables interactive development with immediate code execution feedback. Integration with Google Drive allows easy data storage and retrieval.


Models_Training_Comparison.ipynb
File Edit View Insert Runtime Tools Help
Commands + Code + Text ▷ Run all
Share
RAM
Disk
Files
sample_data
stock_data_prepared_for_trainin...
Disk 74.10 GB available
[3]
[4]
Resources
Manage sessions
Change runtime type

```
    classification_report, confusion_matrix, accuracy_score,
    precision_score, recall_score, f1_score
)

# ============================================================================
# SET RANDOM SEEDS FOR REPRODUCIBILITY
# ============================================================================
SEED = 42

def set_seed(seed=SEED):
    """Set all random seeds for reproducibility"""
    random.seed(seed)
    np.random.seed(seed)
    torch.manual_seed(seed)
    torch.cuda.manual_seed(seed)
    torch.cuda.manual_seed_all(seed)  # for multi-GPU

    # Make CuDNN deterministic (may slow down training slightly)
    torch.backends.cudnn.deterministic = True
    torch.backends.cudnn.benchmark = False

    # Set environment variable for hash seed
    os.environ['PYTHONHASHSEED'] = str(seed)

    print(f" Random seed set to {seed} for reproducibility")

set_seed(SEED)
# ============================================================================

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print(f"Using device: {device}")
if device.type == 'cuda':
    print(f"GPU: {torch.cuda.get_device_name(0)}")
```

```
 Random seed set to 42 for reproducibility
Using device: cuda
GPU: Tesla T4
```

```
# Load prepared data
df = pd.read_csv('stock_data_prepared_for_training.csv')
df['published_date'] = pd.to_datetime(df['published_date'])
```

You are not subscribed. Learn more
You currently have zero compute units available. Resources offered free of charge are not guaranteed. Purchase more units here.
At your current usage level, this runtime may last up to 2 hours 30 minutes.

Want more memory and disk space?
Upgrade to Colab Pro

Python 3 Google Compute Engine backend (GPU)
Showing resources from 9:27 PM to 9:50 PM

System RAM 3.3 / 12.7 GB
GPU RAM 0.5 / 15.0 GB
Disk 38.5 / 112.6 GB

Figure 27: Using Cuda cores in colab

4.4.2.3 Libraries:

- Pandas (data handling)
- NumPy (numerical operations)
- Matplotlib/Seaborn (visualization)
- Pytorch (deep learning)
- Scikit-learn (preprocessing and metrics)

4.4.2.4 Environment:

- Jupyter Notebook
- Google Colab (GPU support)
- VS Code

4.4.2.5 Version Control:

- Git and GitHub

4.4.2.6 Hardware:

GPU: NVIDIA Tesla T4

- Architecture: Turing
- CUDA Cores: 2,560
- Tensor Cores: 320
- Memory: 16 GB GDDR6
- Memory Bandwidth: 320 GB/s
- FP32 Performance: 8.1 TFLOPS
- Training acceleration: ~10-15x faster than CPU

System Memory: 12 GB RAM

- Sufficient for loading full dataset
- Handled batch processing without memory errors

Storage: ~100 GB in Colab environment

- Adequate for dataset (~13 MB) and model checkpoints

5 Conclusion

In this study, we examined the application of LSTM, CNN, and Transformer models to the prediction of closing stock market prices of NEPSE. A common database was created so that there would be equal comparison of all the models. Theoretically, LSTM involves the capturing of long-term trends, CNN pays attention to short-term features, and Transformers are high-performing with attention-based models.

Artificial intelligence forecasting will be effective to inform decision-making, support financial analysts, and risk management in such emerging markets such as Nepal Stock Exchange. Even though the displayed project is only in the form of a conceptual design as opposed to real implementation result, it shows that modern AI methods can be used to solve real-world finances prediction problems.

5.1 Further Work

The path forward requires both deepening (improving current approach with better features, longer horizons, ensembles) and broadening (new techniques like RL, alternative data, hybrid models). The immediate focus should be:

- Using advanced models like Informer, LSTM-CNN hybrids, Temporal Fusion Transformers
- Test longer prediction horizons
- Implement ensemble methods
- Add sentiments data
- Implement trading strategy - Even 55% accuracy can be profitable with proper risk management.

6 References

Seetharaman, A. P. a. A., 2021. *Importance of Machine Learning in Making Investment Decision in Stock Market.* [Online] Available at: https://journals.sagepub.com/doi/10.1177/02560909211059992 [Accessed 12 December 2025].

Hapuarachchi, A. S., 2025. *THE IMPORTANCE OF MACHINE LEARNING,* s.l.: Research Gate.

Wenjie Lu, J. L. Y. L. A. S. J. W., 2020. *A CNN-LSTM-Based Model to Forecast Stock Prices.* [Online] Available at: https://onlinelibrary.wiley.com/doi/10.1155/2020/6622927 [Accessed 12 December 2025].

Nawa Raj Pokhrel, K. R. D. ,. R. R. ,. H. N. B. ,. R. K. K. ,. B. R. ,. W. E. H., 2022. *Predicting Nepse Index Price Using Deep Learning Models.* [Online] Available at: https://www.researchgate.net/publication/360921805_Predicting_Nepse_Index_Price_Using_Deep_Learning_Models
[Accessed 12 December 2025].

Thomas Fischer, C. K., 2018. *Deep learning with long short-term memory networks for financial market predictions.* [Online] Available at: https://www.sciencedirect.com/science/article/abs/pii/S0377221717310652
[Accessed 12 December 2025].

T.O. Kehinde, W. A. K. S.-H. C., 2023. *Financial Market Forecasting using RNN, LSTM, BiLSTM, GRU and Transformer-Based Deep Learning Algorithms ,* Michigan: ieomsociety.

Omer Berat Sezera, M. O. E. D., 2017. *A Deep Neural-Network Based Stock Trading System Based on Evolutionary Optimized Technical Analysis Parameters.* [Online] Available at: https://www.researchgate.net/publication/320370508_A_Deep_Neural-Network_Based_Stock_Trading_System_Based_on_Evolutionary_Optimized_Technical_Analysis_Parameters

[Accessed 12 December 2025].

Ekanta Rai, S. T. S. U. J. K. L., 2025. *Stock Prediction Based on Transformer Model,* Kathmandu: NCCS Research Journal.

Sepp Hochreiter, J. S., 1997. *Long short-term memory,* s.l.: PubMed.

Avinash, 2021. *Hands-On Stock Price Time Series Forecasting using Deep Convolutional Networks.* [Online]
Available at: https://www.analyticsvidhya.com/blog/2021/08/hands-on-stock-price-time-series-forecasting-using-deep-convolutional-networks/
[Accessed 12 December 2025].

Rogerio Pereira dos Santos, J. P. M.-C. V. R. Q. L., 2025. *Deep learning in time series forecasting with transformer models and RNNs.* [Online]
Available at: https://peerj.com/articles/cs-3001/
[Accessed 12 december 2025].

Draxler, T. C. a. R. R., 2014. *Root mean square error (RMSE) or mean absolute error (MAE)? – Arguments against avoiding RMSE in the literature,* College Park: European Geosciences Union.

Kumar, A., 2023. *Mean Squared Error or R-Squared – Which one to use?.* [Online]
Available at: https://vitalflux.com/mean-square-error-r-squared-which-one-to-use/
[Accessed 16 December 2025].

Liu, D., 2023. *Overview of Common Time Series Forecast Error Metrics.* [Online]
Available at: https://www.dannidanliu.com/overview-of-common-time-series-forecast-error-metrics/
[Accessed 16 December 2025].