AI-Powered Stock Market Prediction Using Machine Learning and LSTM

By: Travis Johnson

Problem Statement & Objective

Investors face difficulty predicting market trends due to volatility and complexity. This Project builds multiple ML Models to Predict the stock chosen, which was Apple (AAPL).

Testing Assets

- Historical Stock Data
- Economic Indicators
- News Sentiment

ML Models

- Linear Regression
- XGBoost
- Long Short-Term Memory (LSTM)

Research Literature & Modeling Approach

Referenced academic papers and financial forecasting models.

Goal: Test which model best captures stock dynamics.

Key Techniques

- Lag feature creation (t-1, t-2, t-3)
- Integration of Federal Reserve Economic Data (FRED) data
- VADER-based Sentiment Scoring

Data Collection S&P 500

```python
# Step 1: Obtaining S&P 500 tickers from Wikipedia
def get_sp500_tickers():
    url = "https://en.wikipedia.org/wiki/List_of_S%26P_500_companies"
    html = requests.get(url).text
    soup = BeautifulSoup(html, 'html.parser')
    table = soup.find('table', {'id': 'constituents'})
    tickers = []

    for row in table.findAll('tr')[1:]:
        symbol = row.findAll('td')[0].text.strip()
        symbol = symbol.replace('.', '-')  # Yahoo Finance uses '-' not '.'
        tickers.append(symbol)

    return tickers

# Step 2: Download and save last 10 years of daily data
def download_sp500_data(tickers, save_dir="sp500_data"):
    os.makedirs(save_dir, exist_ok=True)
    end_date = datetime.datetime.today()
    start_date = end_date - datetime.timedelta(days=365 * 10)

    for ticker in tickers:
        try:
            print(f"Downloading {ticker}")
            df = yf.download(ticker, start=start_date.strftime('%Y-%m-%d'), end=end_date.strftime('%Y-%m-%d'))
            df.reset_index(inplace=True)  # Convert 'Date' index into column
            df["Ticker"] = ticker
            df.to_csv(f"{save_dir}/{ticker}.csv", index=False)
        except Exception as e:
            print(f"Failed for {ticker}: {e}")

    print("All files saved correctly with 'Date' column.")

# Running everything
tickers = get_sp500_tickers()
download_sp500_data(tickers)
```

```python
# Re-downloading full AAPL data from 2014 to today for any errors
df = yf.download("AAPL", start="2014-01-01")
df.reset_index(inplace=True)
df["Ticker"] = "AAPL"

# Save corrected data to CSV
df.to_csv("sp500_data/AAPL.csv", index=False)
print("AAPL data re-downloaded and saved.")
```

Collecting the Stock Data for the last 10 years for every company listed on the S&P 500

Data Collection from FRED

```
# Connectting to FRED with my custom API key
fred = Fred(api_key='0e376b94378b31d331908069cef3c2d8')

# Defining the date range
start_date = "2014-01-01"
end_date = datetime.datetime.today().strftime("%Y-%m-%d")

# Economic indicators to download
indicators = {
    "FEDFUNDS": "Interest_Rate",          # Federal Funds Rate
    "CPIAUCSL": "CPI",                    # Consumer Price Index
    "UNRATE": "Unemployment_Rate",        # Unemployment Rate
    "GS10": "10Y_Treasury",               # 10-Year Treasury Rate
    "GDPC1": "Real_GDP"                   # Real Gross Domestic Product
}

economic_data = pd.DataFrame()

# Downloading the data and merging into one DataFrame
for code, name in indicators.items():
    series = fred.get_series(code, start_date, end_date)
    df = series.to_frame(name)
    economic_data = pd.concat([economic_data, df], axis=1)

# Formatting and Saving
economic_data.index.name = "Date"
economic_data.reset_index(inplace=True)
economic_data["Date"] = pd.to_datetime(economic_data["Date"])
economic_data.to_csv("economic_indicators.csv", index=False)

print("Economic indicators saved to 'economic_indicators.csv'")
```

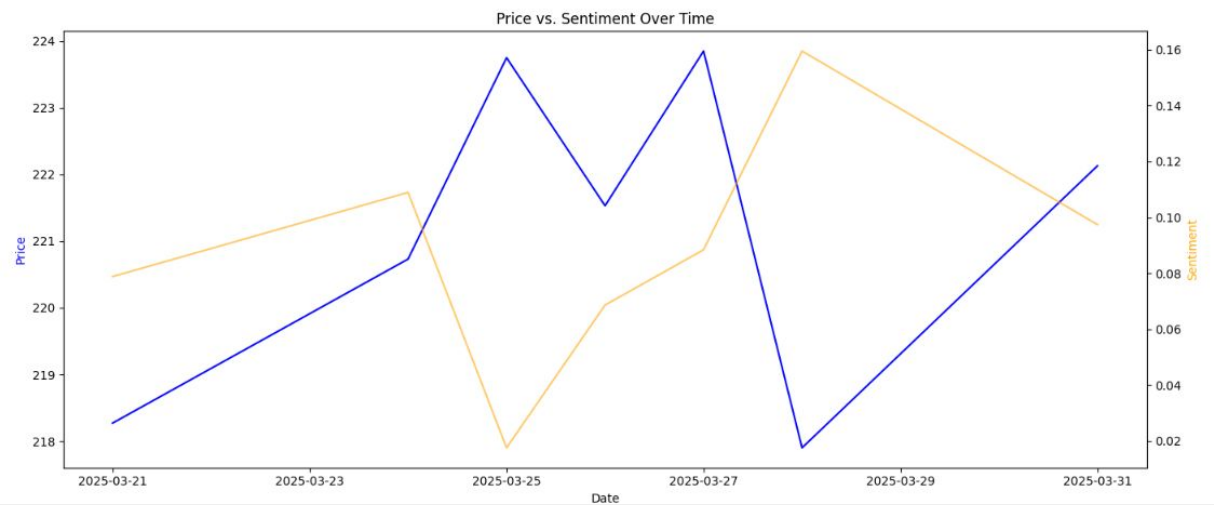
Collecting the Economic Data from the Federal Reserve Economic Data (FRED) Website

Economic Sentiment Collection

```python
# Install and load VADER
nltk.download('vader_lexicon')
sia = SentimentIntensityAnalyzer()

# --- API KEYS ---
newsapi_key = "75e4e306d7ed4839a925e631f0507058"
finnhub_key = "cvlkmbpr01qj3ume0540cvlkmbpr01qj3ume054g"

# --- Date Range (last 365 days) ---
end_date = datetime.date.today()
start_date = end_date - datetime.timedelta(days=365)

# --- Fetching data from NewsAPI ---
def fetch_newsapi_data(query="stock market"):
    url = f"https://newsapi.org/v2/everything?q={query}&from={start_date}&to={end_date}&sortBy=publishedAt&language=en&apiKey={newsapi_key}"
    response = requests.get(url)
    articles = response.json().get("articles", [])
    return [{"date": a["publishedAt"][:10], "headline": a["title"]} for a in articles]

# --- Fetching data from Finnhub (Example: AAPL) ---
def fetch_finnhub_data(symbol="AAPL"):
    url = f"https://finnhub.io/api/v1/company-news?symbol={symbol}&from={start_date}&to={end_date}&token={finnhub_key}"
    response = requests.get(url)
    articles = response.json()
    return [{"date": datetime.datetime.utcfromtimestamp(a["datetime"]).strftime('%Y-%m-%d'), "headline": a["headline"]} for a in articles]

# --- Combining news from both sources ---
newsapi_news = fetch_newsapi_data("stock market")
finnhub_news = fetch_finnhub_data("AAPL")  # You can loop for multiple tickers if needed

all_news = newsapi_news + finnhub_news
news_df = pd.DataFrame(all_news)
news_df["date"] = pd.to_datetime(news_df["date"])

# --- Applying Sentiment Scoring ---
news_df["sentiment_score"] = news_df["headline"].apply(lambda x: sia.polarity_scores(x)["compound"])

# --- Creating the Aggregate Daily Sentiment ---
daily_sentiment = news_df.groupby("date").agg({"sentiment_score": "mean"}).reset_index()
daily_sentiment.rename(columns={"sentiment_score": "avg_sentiment"}, inplace=True)

# --- Save to CSV ---
daily_sentiment.to_csv("combined_sentiment.csv", index=False)
print("Saved combined sentiment to 'combined_sentiment.csv'")
```

Collecting the Economic Sentiment from NewsAPI, and Finnhub for my Vader scoring

Exploratory Data Analysis



Correlation between News sentiment and actual stock price for AAPL

Methodology

Data Preprocessing

- Normalization
- Test/Train Split (80/20)

Models Trained

- Linear Regression (Baseline)
- XGBoost (Nonlinear, robust)
- LSTM (Sequential time-series model)

```
lr = LinearRegression()
lr.fit(X_train, y_train)
y_pred_lr = lr.predict(X_test)

rmse_lr = sqrt(mean_squared_error(y_test, y_pred_lr))
mape_lr = mean_absolute_percentage_error(y_test, y_pred_lr)

print(f"Linear Regression RMSE: {rmse_lr:.2f}, MAPE: {mape_lr:.2%}")
```

Linear Regression Code

```
xgb = XGBRegressor(objective="reg:squarederror", n_estimators=100)
xgb.fit(X_train, y_train)
y_pred_xgb = xgb.predict(X_test)

rmse_xgb = sqrt(mean_squared_error(y_test, y_pred_xgb))
mape_xgb = mean_absolute_percentage_error(y_test, y_pred_xgb)

print(f"XGBoost RMSE: {rmse_xgb:.2f}, MAPE: {mape_xgb:.2%}")
```

XGBoost Code

Results & Insights

Model Comparison

- Linear Regression → RMSE: 4.28, MAPE: 3.24%
- XGBoost → RMSE: 3.12, MAPE: 2.15%
- LSTM → Promising results but it needs more tuning for the data to be valuable

Insights

- The Economic indicators improved overall accuracy
- XGBoost captured nonlinearity better than the linear models

Conclusion & Future Work

The Models show a strong potential for accurate market prediction

Time-series and feature engineering are crucial for overall performance

Future Improvements

- Keep tuning the LSTM Model until it can deliver consistent data
- Incorporate Real time streaming data
- Research more stocks

Questions?