COURSE PROJECT

Machine Learning

Prepared By :
Vijay choudhary
AU21B1003

Contents

Introduction

Project Title: Automated Technical Analysis for Efficient Financial Market Analysis

The Automated Technical Analysis System is designed to assist traders and investors in making informed decisions by automating the process of technical analysis. This system leverages machine learning algorithms and historical stock data to identify key patterns and trends in the financial markets. The implementation utilizes Python programming language and various libraries such as pandas, numpy, yfinance, and plotly.

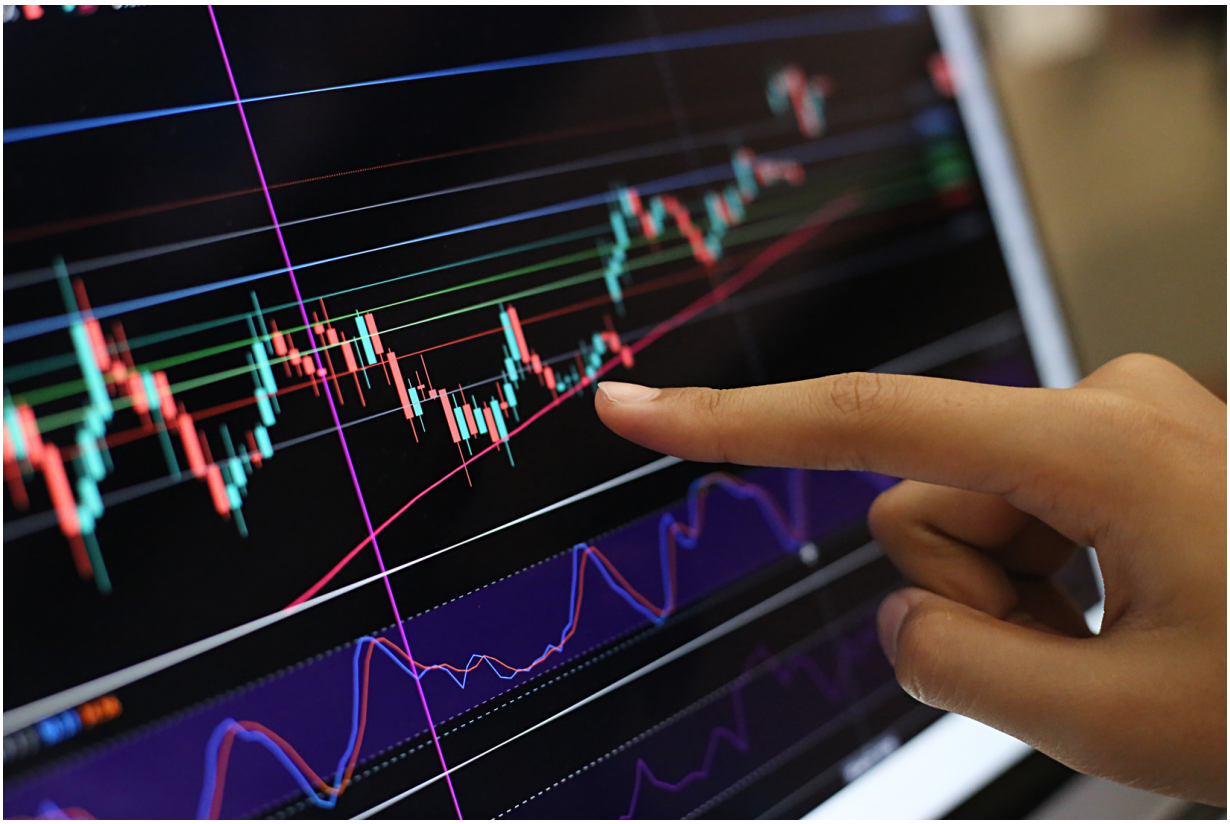
Problem Statement:

In the dynamic landscape of financial markets, traders and investors are inundated with a plethora of stock options, each presenting unique opportunities and challenges. However, the process of manually conducting technical analysis to identify optimal investment opportunities is time-consuming and prone to human error. This manual approach becomes increasingly arduous when managing large watchlists or searching for the best-performing stocks.

The primary challenge faced by traders and investors is the need to meticulously analyze numerous stock charts, employing various technical indicators and chart patterns to make informed decisions. This process involves significant time investment and often leads to cognitive overload, hindering the efficiency of investment strategies. Additionally, the necessity to document and report on the analysis further exacerbates the workload.

Project Objective:

The objective of this project is to develop an automated system capable of identifying stock chart patterns and conducting technical analysis with high accuracy and efficiency. By leveraging machine learning algorithms and data-driven techniques, this system aims to streamline the decision-making process for traders and investors, thereby reducing manual labor and enhancing investment outcomes.



Key Deliverables:

- **Price Action Interpretation:** Implement machine learning models to analyze price action dynamics, including candlestick patterns, support and resistance levels, and trend formations. The system will identify key price movements and patterns indicative of potential market trends and reversals.

- **Stock Chart Pattern Recognition:** Develop algorithms capable of identifying a wide array of stock chart patterns, such as head and shoulders, triangles, flags, and channels, among others. By analyzing historical price data, the system will automatically detect significant chart patterns, aiding in the identification of potential trading opportunities.

Key Deliverables:

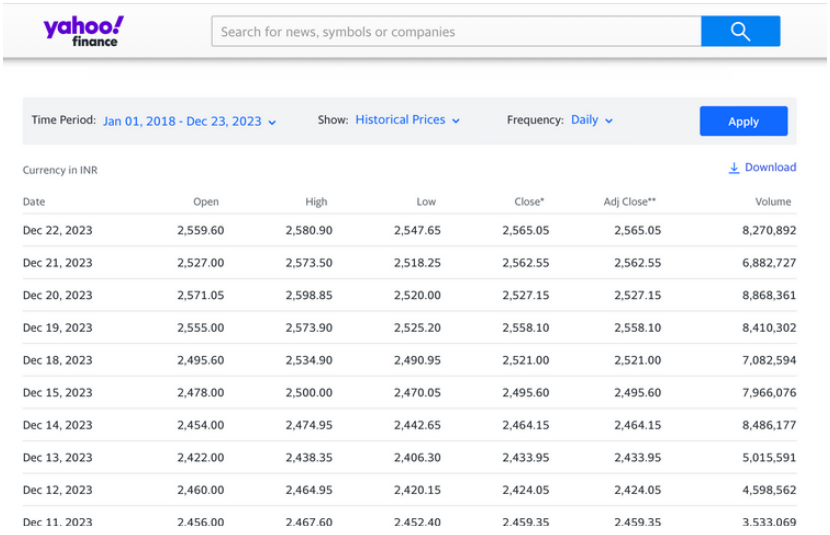
- **Automated Technical Analysis Engine:** Integrate the price action interpretation and stock chart pattern recognition modules into a cohesive automated technical analysis engine. This engine will analyze historical stock data and generate comprehensive reports outlining key technical indicators, trend analyses, and potential trade signals.

- **User-Friendly Interface:** Design an intuitive user interface that enables traders and investors to input their desired stocks or watchlists and receive automated technical analysis reports. The interface will provide interactive visualizations and actionable insights, empowering users to make informed investment decisions efficiently.

Implementation Details:

1.Fetching Historical Stock Data:

- The system utilizes the Yahoo Finance API through the yfinance library to fetch historical stock data for the specified ticker symbols.
- Users are prompted to input the number of days for historical data and the time interval for data sampling.



Date	Open	High	Low	Close*	Adj Close**	Volume
Dec 22, 2023	2,559.60	2,580.90	2,547.65	2,565.05	2,565.05	8,270,892
Dec 21, 2023	2,527.00	2,573.50	2,518.25	2,562.55	2,562.55	6,882,727
Dec 20, 2023	2,571.05	2,598.85	2,520.00	2,527.15	2,527.15	8,868,361
Dec 19, 2023	2,555.00	2,573.90	2,525.20	2,558.10	2,558.10	8,410,302
Dec 18, 2023	2,495.60	2,534.90	2,490.95	2,521.00	2,521.00	7,082,594
Dec 15, 2023	2,478.00	2,500.00	2,470.05	2,495.60	2,495.60	7,966,076
Dec 14, 2023	2,454.00	2,474.95	2,442.65	2,464.15	2,464.15	8,486,177
Dec 13, 2023	2,422.00	2,438.35	2,406.30	2,433.95	2,433.95	5,015,591
Dec 12, 2023	2,460.00	2,464.95	2,420.15	2,424.05	2,424.05	4,598,562
Dec 11, 2023	2,456.00	2,467.60	2,452.40	2,459.35	2,459.35	3,533,069

```python
import yfinance as yf
from datetime import datetime, timedelta

# Function to fetch historical stock data
def fetch_stock_data(ticker_symbol, day_range, time_interval):
    end_date = datetime.now()
    start_date = end_date - timedelta(days=day_range*7//5)
    data = yf.download(ticker_symbol, start=start_date, end=end_date, interval=time_interval)
    return data.dropna()
data.tail(10)
```

Date	Open	High	Low	Close	Adj Close	Volume
2024-01-19	3603.000000	3655.850098	3603.000000	3646.000000	3646.000000	1913923
2024-01-22	3646.000000	3646.000000	3646.000000	3646.000000	3646.000000	0
2024-01-23	3637.600098	3661.350098	3536.199951	3551.000000	3551.000000	1742142
2024-01-24	3570.000000	3617.949951	3521.699951	3589.199951	3589.199951	3711521
2024-01-25	3599.899902	3618.250000	3564.800049	3593.449951	3593.449951	2206540
2024-01-29	3604.000000	3733.850098	3600.100098	3708.000000	3708.000000	2072982
2024-01-30	3715.949951	3737.899902	3624.899902	3633.300049	3633.300049	1454989
2024-01-31	3520.000000	3538.000000	3387.050049	3479.750000	3479.750000	9637606
2024-02-01	3480.000000	3493.750000	3371.000000	3398.000000	3398.000000	6149780
2024-02-02	3402.699951	3436.350098	3361.000000	3376.050049	3376.050049	7579802

2.Preprocessing Stock Data:

- The fetched data is preprocessed to extract relevant columns such as Open, High, Low, and Close prices.
- Pivot points are identified based on the specified criteria, indicating potential reversal or continuation patterns in the stock's price movement.

```
# Function to preprocess stock data
def preprocess_stock_data(data):
    df_ticker = data[['Open', 'High', 'Low', 'Close' ]].reset_index()
    df_ticker.columns = ['time', 'open', 'high', 'low', 'close']
    df_ticker = df_ticker.dropna()
    df_ticker.reset_index(drop=True, inplace=True)
    # Add 'pivot' and 'pointpos' columns to the DataFrame
    df_ticker['pivot'] = df_ticker.apply(lambda x: pivot_id(df_ticker, x.name, 3, 3), axis=1)
    df_ticker['pointpos'] = df_ticker.apply(lambda row: point_position(row), axis=1)
    return df_ticker
df_ticker.tail(10)
```

	time	open	high	low	close	pivot	pointpos
46	2024-01-19	3603.000000	3655.850098	3603.000000	3646.000000	0	NaN
47	2024-01-22	3646.000000	3646.000000	3646.000000	3646.000000	0	NaN
48	2024-01-23	3637.600098	3661.350098	3536.199951	3551.000000	0	NaN
49	2024-01-24	3570.000000	3617.949951	3521.699951	3589.199951	1	3521.698951
50	2024-01-25	3599.899902	3618.250000	3564.800049	3593.449951	0	NaN
51	2024-01-29	3604.000000	3733.850098	3600.100098	3708.000000	0	NaN
52	2024-01-30	3715.949951	3737.899902	3624.899902	3633.300049	2	3737.900902
53	2024-01-31	3520.000000	3538.000000	3387.050049	3479.750000	0	NaN
54	2024-02-01	3480.000000	3493.750000	3371.000000	3398.000000	0	NaN
55	2024-02-02	3402.699951	3436.350098	3361.000000	3376.050049	0	NaN

3.Pivot Calculation Functions:

1.pivot_id Function:

- The function calculates pivot points for recent candles based on specified parameters **n1** and **n2**.
- It iterates over the recent candles within the range **[l – n1, l + n2]** to determine pivot points.
- The function returns a numeric value indicating the type of pivot point:
- **0** for no pivot
- **1** for low pivot
- **2** for high pivot
- **3** for both low and high pivot

```
# Function to calculate pivot points for recent candles
def pivot_id(df1, l, n1, n2):
    # Check if there are enough recent candles to calculate pivot points
    if l - n1 < 0 or l + n2 >= len(df1):
        return 0
    # Initialize pivot flags
    piv_id_low = 1
    piv_id_high = 1
    # Loop over recent candles to check for pivot points
    for i in range(l - n1, l + n2 + 1):
        # Check for low pivot
        if df1.low[l] > df1.low[i]:
            piv_id_low = 0
        # Check for high pivot
        if df1.high[l] < df1.high[i]:
            piv_id_high = 0
    # Determine pivot type
    if piv_id_low and piv_id_high:
        return 3  # Both low and high pivot
    elif piv_id_low:
        return 1  # Low pivot
    elif piv_id_high:
        return 2  # High pivot
    else:
        return 0  # No pivot
```

3.Pivot Calculation Functions:

2.point_position Function:

- This function calculates the position of pivot points based on their type.
- If a low pivot is detected, it subtracts a small value (1e-3) from the low price to position the pivot point slightly below the candlestick.
- If a high pivot is detected, it adds a small value (1e-3) to the high price to position the pivot point slightly above the candlestick.
- If no pivot is detected, it returns np.nan.

```
# Function to calculate the position of pivot points for recent candles
def point_position(x):
    # Calculate position based on pivot type
    if x['pivot'] == 1:
        return x['low'] - 1e-3  # Low pivot
    elif x['pivot'] == 2:
        return x['high'] + 1e-3  # High pivot
    else:
        return np.nan  # No pivot
print(point_position)
```

4.Linear Regression Analysis:

- Linear regression models are employed to analyze the pivot points and identify trends in the stock's price movement.
- Separate linear regression models are trained for low pivot points and high pivot points, considering recent candlestick patterns.
- Training and testing data splits are performed to evaluate the models' performance and predict future trends.

```
def calculate_linear_regression(df_ticker, back_candles, test_size=0.2):
    xxmin = np.array([])
    minim = np.array([])
    xxmax = np.array([])
    high = np.array([])

    for i in range(max(0, len(df_ticker) - back_candles), len(df_ticker)):
        if df_ticker.iloc[i].pivot == 1:
            minim = np.append(minim, df_ticker.iloc[i].low)
            xxmin = np.append(xxmin, i)
        if df_ticker.iloc[i].pivot == 2:
            xxmax = np.append(xxmax, i)
            high = np.append(high, df_ticker.iloc[i].high)

    xxmin_train, xxmin_test, minim_train, minim_test = train_test_split(xxmin, minim, test_size=test_size, random_st
    xxmax_train, xxmax_test, high_train, high_test = train_test_split(xxmax, high, test_size=test_size, random_state

    # Initialize Linear Regression models
    model_min = LinearRegression()
    model_max = LinearRegression()

    # Fit linear regression for low pivot points
    xxmin_train_reshaped = xxmin_train.reshape(-1, 1)
    model_min.fit(xxmin_train_reshaped, minim_train)
    slmin = model_min.coef_[0]
    intercmin = model_min.intercept_

    # Fit linear regression for high pivot points
    xxmax_train_reshaped = xxmax_train.reshape(-1, 1)
    model_max.fit(xxmax_train_reshaped, high_train)
    slmax = model_max.coef_[0]
    intercmax = model_max.intercept_

    return xxmin_train, xxmin_test, minim_train, minim_test, xxmax_train, xxmax_test, high_train, high_test, slmin,
    # Function to predict trend based on slopes of regression lines
```

4.Linear Regression Analysis:

- Linear regression models are employed to analyze the pivot points and identify trends in the stock's price movement.
- Separate linear regression models are trained for low pivot points and high pivot points, considering recent candlestick patterns.
- Training and testing data splits are performed to evaluate the models' performance and predict future trends.

```
def calculate_linear_regression(df_ticker, back_candles, test_size=0.2):
    xxmin = np.array([])
    minim = np.array([])
    xxmax = np.array([])
    high = np.array([])

    for i in range(max(0, len(df_ticker) - back_candles), len(df_ticker)):
        if df_ticker.iloc[i].pivot == 1:
            minim = np.append(minim, df_ticker.iloc[i].low)
            xxmin = np.append(xxmin, i)
        if df_ticker.iloc[i].pivot == 2:
            xxmax = np.append(xxmax, i)
            high = np.append(high, df_ticker.iloc[i].high)

    xxmin_train, xxmin_test, minim_train, minim_test = train_test_split(xxmin, minim, test_size=test_size, random_st
    xxmax_train, xxmax_test, high_train, high_test = train_test_split(xxmax, high, test_size=test_size, random_state

    # Initialize Linear Regression models
    model_min = LinearRegression()
    model_max = LinearRegression()

    # Fit linear regression for low pivot points
    xxmin_train_reshaped = xxmin_train.reshape(-1, 1)
    model_min.fit(xxmin_train_reshaped, minim_train)
    slmin = model_min.coef_[0]
    intercmin = model_min.intercept_

    # Fit linear regression for high pivot points
    xxmax_train_reshaped = xxmax_train.reshape(-1, 1)
    model_max.fit(xxmax_train_reshaped, high_train)
    slmax = model_max.coef_[0]
    intercmax = model_max.intercept_

    return xxmin_train, xxmin_test, minim_train, minim_test, xxmax_train, xxmax_test, high_train, high_test, slmin,
    # Function to predict trend based on slopes of regression lines
```

5.Trend Prediction and Visualization:

- The system predicts the trend based on the slopes of the regression lines generated by the linear regression models.
- Candlestick charts are generated using the plotly library, displaying historical price data, pivot points, and regression lines.
- Trend predictions for both training and testing data are displayed on the charts, providing insights into potential market movements.


```
# Print R-squared values
print(f"R-squared (Train) - Low: {r_squared_min_train:.2f}%, High: {r_squared_max_train:.2f}%")
print(f"R-squared (Test) - Low: {r_squared_min_test:.2f}%, High: {r_squared_max_test:.2f}%")
```

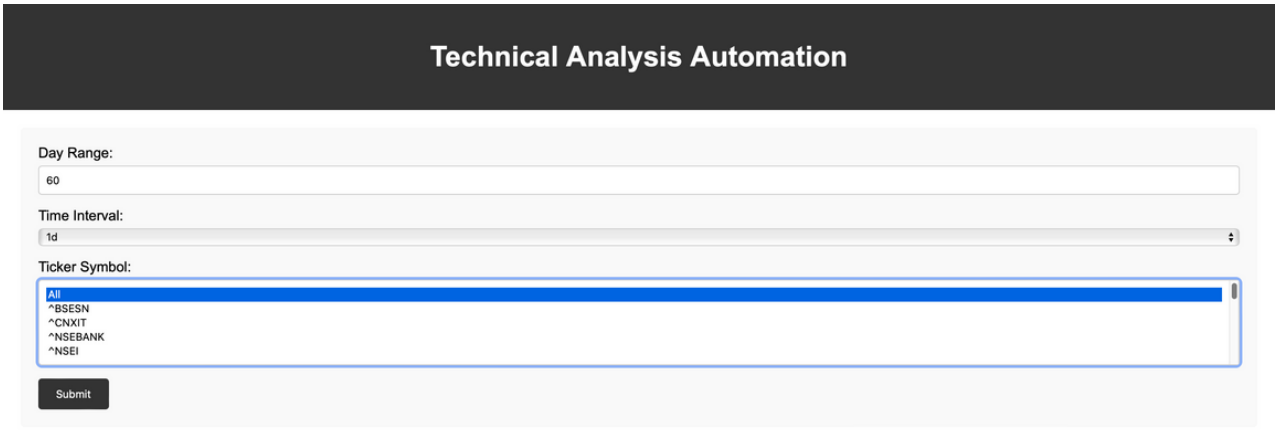
```
R-squared (Train) - Low: 0.97%, High: 1.00%
R-squared (Test) - Low: nan%, High: nan%
```

Usage:

- **Input Parameters:**
  1. Users provide input parameters including the number of days for historical data and the time interval for data sampling.
  2. Ticker symbols of interest are specified in the code.

- **Visualization:**
  1. Upon execution, the system generates candlestick charts for each specified ticker symbol, displaying historical price data and technical analysis indicators.
  2. Pivot points, linear regression lines, and trend predictions are overlaid on the charts, aiding users in identifying potential trading opportunities.

Usage:


Technical Analysis Automation
Day Range:
60
Time Interval:
1d
Ticker Symbol:
All
^BSESN
^CNXIT
^NSEBANK
^NSEI
Submit


Candlestick Charts
Candlestick Chart for ^BSESN
Candlestick Chart for ^CNXIT
Candlestick Chart for ^NSEBANK
Candlestick Chart for ^NSEI
trace 0
pivot
trace 2
trace 3

Conclusion:

The Automated Technical Analysis System offers a powerful tool for traders and investors to streamline the process of technical analysis and make data-driven decisions in the financial markets. By automating the identification of key patterns and trends, this system enhances efficiency and accuracy, enabling users to stay ahead in an ever-changing market environment.

Future Enhancements:

1. Integration with real-time data sources for up-to-date analysis.
2. Implementation of additional technical indicators and pattern recognition algorithms.
3. Development of a user-friendly interface for seamless interaction and customization.
4. Incorporation of advanced machine learning techniques for predictive modeling and risk assessment.

References:

https://www.investopedia.com/articles/technical/112601.asp

https://www.jpmorgan.com/technology/technology-blog/searching-for-patterns

https://finance.yahoo.com/quote/%5ENSEBANK/history?p=%5ENSEBANK

https://plotly.com/python/candlestick-charts/

https://ieeexplore.ieee.org/document/10146731/metrics#metrics

Thank You