

# Stock Market Trading Platform

## Source Code for Academic Report

S. Praveen Kumar  
School of Computer Science  
Anna University  
Chennai, India

### 1. Introduction

This document contains the key source code components of the Stock Market Trading Platform that can be included in academic reports and publications. The platform is built using Python Flask framework with yfinance for real-time stock data, and provides features such as real-time stock charting with technical indicators, AI-powered trading signals, portfolio management, live news updates, and predictive analytics.

### 2. Complete Source Code

```
"""
Stock Market Trading Platform - Source Code for Report
=====

This file contains the key source code components of the Stock Market Trading Platform
that can be included in academic reports and publications.

The platform is built using Python Flask framework with yfinance for real-time stock data,
and provides features such as:
- Real-time stock charting with technical indicators
- AI-powered trading signals
- Portfolio management
- Live news updates
- Predictive analytics
"""

# Core Dependencies
import flask
import yfinance as yf
import pandas as pd
import numpy as np
import mysql.connector
from datetime import datetime, timedelta

# Flask Application Setup
app = Flask(__name__)
app.secret_key = 'your_secret_key_here'

# Main Routes

@app.route('/stock_graph')
def stock_graph():
    """Display interactive stock charts with technical indicators"""
    if 'username' not in session:
```

```

return redirect(url_for('login'))

username = session['username']
symbol = request.args.get('symbol', 'AAPL')
period = request.args.get('period', '1d')

return render_template('stock_graph.html',
username=username,
symbol=symbol,
period=period)

@app.route('/get_chart_data')
def get_chart_data():
    """Fetch stock data and calculate technical indicators for charting"""
    symbol = request.args.get('symbol', 'AAPL')
    period = request.args.get('period', '1y')

```

```

try:
    # Map period to yfinance period
    period_map = {
        '1d': '1d',
        '1wk': '5d',
        '1w': '5d',
        '1mo': '1mo',
        '1m': '1mo',
        '3mo': '3mo',
        '3m': '3mo',
        '6mo': '6mo',
        '6m': '6mo',
        '1y': '1y',
        '5y': '5y',
        'max': 'max',
        'all': 'max'
    }

    # Map period to interval
    interval_map = {
        '1d': '5m', # 5-minute intervals for 1 day
        '1wk': '30m', # 30-minute intervals for 1 week
        '1w': '30m',
        '1mo': '1d', # 1-day intervals for 1 month
        '1m': '1d',
        '3mo': '1d',
        '3m': '1d',
        '6mo': '1d',
        '6m': '1d',
        '1y': '1d',
        '5y': '1wk',
        'max': '1wk',
        'all': '1wk'
    }

    yf_period = period_map.get(period, '1y')
    yf_interval = interval_map.get(period, '1d')

    # Fetch stock data
    stock = yf.Ticker(symbol)
    hist = stock.history(period=yf_period, interval=yf_interval)

    # If no data, try with default 1-month period
    if hist.empty:
        hist = stock.history(period='1mo', interval='1d')

    if hist.empty:
        return jsonify({

```

```
'error': 'No data available',
```

```
'symbol': symbol  
}), 404
```

```
# Convert to Indian Rupees and prepare data
```

```
dates = []
```

```
for idx in hist.index:
```

```
try:
```

```
dates.append(idx.strftime('%Y-%m-%d %H:%M:%S'))
```

```
except:
```

```
dates.append(str(idx))
```

```
prices = [float(val) * 83.0 for val in hist['Close']] # Convert to INR
```

```
highs = [float(val) * 83.0 for val in hist['High']] # Convert to INR
```

```
lows = [float(val) * 83.0 for val in hist['Low']] # Convert to INR
```

```
# Calculate technical indicators for chart overlay
```

```
# Moving Averages (only for periods longer than 1 day)
```

```
ma20 = [None] * len(prices)
```

```
ma50 = [None] * len(prices)
```

```
rsi = [50.0] * len(prices)
```

```
if period != '1d' and len(hist) >= 20:
```

```
hist['MA20'] = hist['Close'].rolling(window=min(20, len(hist))).mean()
```

```
hist['MA50'] = hist['Close'].rolling(window=min(50, len(hist))).mean()
```

```
# Convert moving averages to INR
```

```
ma20 = [float(val) * 83.0 if not pd.isna(val) else None for val in hist['MA20']]
```

```
ma50 = [float(val) * 83.0 if not pd.isna(val) else None for val in hist['MA50']]
```

```
# RSI calculation
```

```
if len(hist) >= 14:
```

```
delta = hist['Close'].diff()
```

```
gain = (delta.where(delta > 0, 0)).rolling(window=14).mean()
```

```
loss = (-delta.where(delta < 0, 0)).rolling(window=14).mean()
```

```
rs = gain / loss
```

```
rsi_series = 100 - (100 / (1 + rs))
```

```
# Convert RSI to list
```

```
rsi = []
```

```
try:
```

```
# Handle case where rsi_series might be a scalar by converting to Series
```

```
if isinstance(rsi_series, (int, float)) or (hasattr(rsi_series, 'ndim') and rsi_series.ndim == 0):
```

```
rsi_series = pd.Series([float(rsi_series)] * len(dates))
```

```
elif not isinstance(rsi_series, pd.Series):
```

```
rsi_series = pd.Series(rsi_series)
```

```
for i in range(len(rsi_series)):
```

```
val = rsi_series.iloc[i]
```

```
if not pd.isna(val):
```

```
rsi.append(float(val))
```

```
else:
```

```
rsi.append(None)
```

```
except:
```

```
# Fallback if the above approach doesn't work
```

```
rsi = [50.0] * len(dates)
```

```
# Enhanced AI features
```

```
# Calculate volatility
```

```
if len(prices) > 1:
```

```
returns = []
```

```
for i in range(1, len(prices)):
```

```

returns.append((prices[i] - prices[i-1]) / prices[i-1])
volatility = np.std(returns) * np.sqrt(252) if len(returns) > 1 else 0
else:
volatility = 0

# Calculate trend direction
if len(prices) > 1:
price_change = prices[-1] - prices[0]
trend_direction = "Bullish" if price_change > 0 else "Bearish" if price_change < 0 else
"Neutral"
else:
trend_direction = "Neutral"

# Calculate support and resistance levels (only for longer periods)
support_level = 0
resistance_level = 0
if period != '1d' and len(prices) > 0:
support_level = min(prices) * 0.99
resistance_level = max(prices) * 1.01

# Calculate next day prediction
if len(prices) > 1:
# Simple momentum-based prediction
recent_changes = []
for i in range(max(0, len(prices) - 10), len(prices)):
if i > 0:
recent_changes.append(prices[i] - prices[i-1])

if len(recent_changes) > 0:
avg_change = sum(recent_changes) / len(recent_changes)
# Add some randomness based on volatility
volatility_factor = volatility * prices[-1] * 0.1
random_factor = np.random.normal(0, volatility_factor)
next_day_prediction = prices[-1] + avg_change + random_factor
else:
next_day_prediction = prices[-1]
else:
next_day_prediction = prices[-1] if prices else 0

return jsonify({

```

```

'symbol': symbol,
'data': {
'x': dates,
'y': prices,
'ma20': ma20,
'ma50': ma50,
'rsi': rsi
},
'high': highs,
'low': lows,
'ai_features': {
'volatility': round(volatility, 4),
'trend_direction': trend_direction,
'support_level': round(support_level, 2),
'resistance_level': round(resistance_level, 2),
'next_day_prediction': round(next_day_prediction, 2),
'data_points': len(dates)
}
})
except Exception as e:
print(f"Error fetching chart data for {symbol}: {e}")
import traceback
traceback.print_exc()
return jsonify({
'error': 'Error loading chart data',

```

```

'message': str(e),
'symbol': symbol
}), 500

@app.route('/get_trading_signal')
def get_trading_signal():
    """Generate AI-powered trading signals with confidence metrics"""
    symbol = request.args.get('symbol', 'AAPL')

    # Simple trading signal logic (in a real app, this would be more sophisticated)
    try:
        # Fetch stock data
        stock = yf.Ticker(symbol)
        hist = stock.history(period="3mo")

        if hist.empty:
            return jsonify({
                'signal': 'HOLD',
                'reason': 'No data available'
            })

        # Calculate simple moving averages
        hist['MA20'] = hist['Close'].rolling(window=20).mean()
        hist['MA50'] = hist['Close'].rolling(window=50).mean()

```

```

current_price = hist['Close'].iloc[-1]
ma20 = hist['MA20'].iloc[-1]
ma50 = hist['MA50'].iloc[-1]

# Simple signal logic
if current_price > ma20 and ma20 > ma50:
    signal = 'BUY'
    reason = f'GO HIGH: Price: █{current_price * 83:.2f} | Bullish trend'
elif current_price < ma20 and ma20 < ma50:
    signal = 'SELL'
    reason = f'GO LOW: Price: █{current_price * 83:.2f} | Bearish trend'
else:
    signal = 'HOLD'
    reason = f'NEUTRAL: Price: █{current_price * 83:.2f} | Wait for clearer signal'

return jsonify({
    'signal': signal,
    'reason': reason
})
except Exception as e:
    print(f"Error generating trading signal for {symbol}: {e}")
    return jsonify({
        'signal': 'HOLD',
        'reason': 'Error generating signal'
    })

@app.route('/get_news')
def get_news():
    """Generate live market news updates every 5 minutes"""
    # Generate dynamic news based on real stock movements
    articles = []
    current_date = datetime.now().strftime('%B %d, %Y')

    # Get top movers
    top_stocks = ['AAPL', 'MSFT', 'GOOGL', 'AMZN', 'TSLA', 'NVDA', 'META']
    movers = []

    for symbol in top_stocks:
        try:
            stock = yf.Ticker(symbol)
            hist = stock.history(period='5d')

```

```

if len(hist) >= 2:
    current = float(hist['Close'].iloc[-1])
    previous = float(hist['Close'].iloc[-2])
    change_pct = ((current - previous) / previous) * 100
    info = stock.info
    name = info.get('shortName', symbol)
    movers.append({
        'symbol': symbol,
        'name': name,

        'change': change_pct,
        'price': current * 83.0
    })
except:
    pass

# Sort by absolute change
movers.sort(key=lambda x: abs(x['change']), reverse=True)

# Create news from top 3 movers
if len(movers) >= 1:
    top = movers[0]
    direction = "surges" if top['change'] > 0 else "drops"
    articles.append({
        "title": f"{top['name']} {direction} {abs(top['change']):.1f}% - {current_date}",
        "summary": f"{top['name']} ({top['symbol']}) shows significant movement with {top['change']:.2f}% change. Current price: ■{top['price']:.2f}. Market analysts monitoring closely.",
        "timestamp": datetime.now().strftime('%H:%M')
    })

if len(movers) >= 2:
    second = movers[1]
    direction = "rallies" if second['change'] > 0 else "declines"
    articles.append({
        "title": f"{second['name']} {direction} {abs(second['change']):.1f}% in Latest Trading",
        "summary": f"{second['name']} experiences {abs(second['change']):.2f}% {'gain' if second['change'] > 0 else 'loss'} at ■{second['price']:.2f}. Trading volume indicates strong investor interest.",
        "timestamp": datetime.now().strftime('%H:%M')
    })

if len(movers) >= 3:
    third = movers[2]
    direction = "gains" if third['change'] > 0 else "loses"
    articles.append({
        "title": f"{third['name']} {direction} momentum as {third['symbol']} moves {abs(third['change']):.1f}%",
        "summary": f"{third['name']} continues {direction} momentum with {abs(third['change']):.2f}% movement. Current valuation at ■{third['price']:.2f} attracts investor attention.",
        "timestamp": datetime.now().strftime('%H:%M')
    })

# Add market overview
major_indices = ['SPY', 'QQQ', 'DIA'] # S&P 500, NASDAQ, Dow Jones ETFs
market_data = []
total_change = 0

for symbol in major_indices:
    try:
        stock = yf.Ticker(symbol)
        hist = stock.history(period='5d')
        if len(hist) >= 2:
            current = float(hist['Close'].iloc[-1])
            previous = float(hist['Close'].iloc[-2])

```

```
change_pct = ((current - previous) / previous) * 100
```

```
total_change += change_pct
market_data.append({
    'symbol': symbol,
    'change': change_pct
})
except:
    pass

# Calculate market sentiment from real data
if market_data:
    avg_change = total_change / len(market_data)
    if avg_change > 0.5:
        market_sentiment_label = "Bullish"
    elif avg_change < -0.5:
        market_sentiment_label = "Bearish"
    else:
        market_sentiment_label = "Neutral"

articles.append({
    "title": f"Market Update: {market_sentiment_label} Sentiment Prevails - {current_date}",
    "summary": f"Major indices show {market_sentiment_label.lower()} sentiment with average movement of {avg_change:.2f}%. Investors remain {market_sentiment_label.lower()} on economic outlook.",
    "timestamp": datetime.now().strftime('%H:%M')
})

# Add fallback if no articles generated
if not articles:
    articles = [
        {
            "title": f"Market Update - {current_date}",
            "summary": "Markets are currently active. Check individual stocks for latest price movements and trading opportunities.",
            "timestamp": datetime.now().strftime('%H:%M')
        }
    ]

return jsonify({
    'articles': articles,
    'last_updated': datetime.now().strftime('%Y-%m-%d %H:%M:%S')
})

@app.route('/predict_next_week')
def predict_next_week():
    """Generate stock price predictions for the next week"""
    if 'username' not in session:
        return redirect(url_for('login'))

    username = session['username']

    # Sample stock symbols for predictions
    stock_symbols = ['AAPL', 'MSFT', 'GOOGL', 'AMZN', 'TSLA', 'NVDA', 'META', 'NFLX', 'AMD', 'INTC']
```

```
go_high_predictions = []
go_low_predictions = []
neutral_predictions = []

# Generate predictions for each stock
for symbol in stock_symbols:
    try:
        # Fetch stock data using yfinance
        stock = yf.Ticker(symbol)
```

```

hist = stock.history(period="3mo")

if hist.empty:
    continue

# Get current price
current_price = float(hist['Close'].iloc[-1])

# Calculate technical indicators
# Moving averages
hist['MA20'] = hist['Close'].rolling(window=20).mean()
hist['MA50'] = hist['Close'].rolling(window=50).mean()

ma20 = hist['MA20'].iloc[-1]
ma50 = hist['MA50'].iloc[-1]

# RSI calculation
delta = hist['Close'].diff()
gain = (delta.where(delta > 0, 0)).rolling(window=14).mean()
loss = (-delta.where(delta < 0, 0)).rolling(window=14).mean()
rs = gain / loss
rsi = 100 - (100 / (1 + rs))
rsi_value = 50.0 # Default
try:
    rsi_value = float(rsi.iloc[-1])
except:
    try:
        rsi_value = float(pd.Series(rsi).iloc[-1])
    except:
        pass

# Get company name
try:
    info = stock.info
    company_name = info.get('longName', symbol)
except:
    company_name = symbol

# Simple prediction logic
buy_signals = 0
sell_signals = 0

```

```

# Moving average signals
if current_price > ma20:
    buy_signals += 1
else:
    sell_signals += 1

if ma20 > ma50:
    buy_signals += 1
else:
    sell_signals += 1

# RSI signals
if rsi_value < 30:
    buy_signals += 1
elif rsi_value > 70:
    sell_signals += 1
else:
    # Neutral RSI
    pass

# Create prediction object
prediction = {
    'symbol': symbol,

```

```

'name': company_name,
'current_price': current_price,
'rsi': round(rsi_value, 2),
'buy_signals': buy_signals,
'sell_signals': sell_signals,
'recommendation': 'Strong Buy' if buy_signals >= 2 else ('Strong Sell' if sell_signals >= 2
else 'Hold')
}

# Categorize predictions
if buy_signals >= 2:
go_high_predictions.append(prediction)
elif sell_signals >= 2:
go_low_predictions.append(prediction)
else:
neutral_predictions.append(prediction)

except Exception as e:
print(f"Error generating prediction for {symbol}: {e}")
continue

return render_template('predict_next_week.html',
username=username,
go_high_predictions=go_high_predictions,
go_low_predictions=go_low_predictions,
neutral_predictions=neutral_predictions)

```

```

@app.route('/prediction_chart/<symbol>')
def prediction_chart(symbol):
    """Display prediction charts with historical and predicted prices"""
    if 'username' not in session:
        return redirect(url_for('login'))

    username = session['username']

    try:
        # Fetch stock data using yfinance
        stock = yf.Ticker(symbol)
        hist = stock.history(period="1mo") # Last 30 days for historical data

        if hist.empty:
            return render_template('prediction_chart.html',
username=username,
error="No data available for this stock symbol")

        # Get current price and company name
        current_price = float(hist['Close'].iloc[-1]) * 83.0 # Convert to INR
        try:
            info = stock.info
            company_name = info.get('longName', symbol)
        except:
            company_name = symbol

        # Prepare historical data
        historical_dates = [idx.strftime('%Y-%m-%d') for idx in hist.index]
        historical_prices = [float(val) * 83.0 for val in hist['Close']] # Convert to INR
        historical_ma20 = [float(val) * 83.0 if not pd.isna(val) else None for val in hist['MA20']]
        if 'MA20' in hist.columns else []
        historical_ma50 = [float(val) * 83.0 if not pd.isna(val) else None for val in hist['MA50']]
        if 'MA50' in hist.columns else []

        # Calculate RSI for historical data
        delta = hist['Close'].diff()
        gain = (delta.where(delta > 0, 0)).rolling(window=14).mean()
        loss = (-delta.where(delta < 0, 0)).rolling(window=14).mean()
        rs = gain / loss

```

```

rsi_series = 100 - (100 / (1 + rs))
# Convert to list safely
historical_rsi = []
try:
# Handle case where rsi_series might be a scalar by converting to Series
if isinstance(rsi_series, (int, float)) or (hasattr(rsi_series, 'ndim') and rsi_series.ndim
== 0):
rsi_series = pd.Series([float(rsi_series)] * len(historical_dates))
elif not isinstance(rsi_series, pd.Series):
rsi_series = pd.Series(rsi_series)

for i in range(len(rsi_series)):
val = rsi_series.iloc[i]
if not pd.isna(val):

```

```

historical_rsi.append(float(val))
else:
historical_rsi.append(None)
except:
# Fallback if the above approach doesn't work
historical_rsi = [50.0] * len(historical_dates)

# Simple prediction for next 7 days (mock data for now)
# In a real application, this would use a machine learning model
prediction_dates = []
prediction_prices = []
last_date = hist.index[-1]

# Generate 7 days of prediction data
for i in range(1, 8):
next_date = last_date + timedelta(days=i)
prediction_dates.append(next_date.strftime('%Y-%m-%d'))
# Simple linear prediction based on recent trend
price_change = (historical_prices[-1] - historical_prices[-2]) if len(historical_prices) >
1 else 0
predicted_price = historical_prices[-1] + (price_change * i * 0.8) # Dampen the prediction
prediction_prices.append(float(predicted_price))

# Prepare moving averages for predictions (simplified)
prediction_ma20 = [float(val) for val in prediction_prices] # Simplified
prediction_rsi = [historical_rsi[-1]] * 7 if historical_rsi and historical_rsi[-1] is not
None else [50.0] * 7 # Simplified - keep last RSI value

return render_template('prediction_chart.html',
username=username,
symbol=symbol,
company_name=company_name,
current_price=current_price,
historical_dates=historical_dates,
historical_prices=historical_prices,
historical_ma20=historical_ma20,
historical_ma50=historical_ma50,
historical_rsi=historical_rsi,
prediction_dates=prediction_dates,
prediction_prices=prediction_prices,
prediction_ma20=prediction_ma20,
prediction_rsi=prediction_rsi)
except Exception as e:
print(f"Error generating prediction chart for {symbol}: {e}")
import traceback
traceback.print_exc()
return render_template('prediction_chart.html',
username=username,
error=f"Error loading prediction data: {str(e)}")

if __name__ == '__main__':

```

```
app.run(debug=True)
```

Generated on December 05, 2025 at 01:09