## Importing libraries

```
import yfinance as yf
import pandas as pd
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, LSTM, Dropout
from sklearn.preprocessing import MinMaxScaler
import matplotlib.pyplot as plt
```

## Data importing and documentation

```
def download_data(ticker, start, end):
    data = yf.download(ticker, start=start, end=end)
    if isinstance(data.columns, pd.MultiIndex):
        data.columns = data.columns.get_level_values(0)
    return data
```

## Relative Strength Index (RSI) calculation formula :

```
def compute_RSI(series, period):
    delta = series.diff()
    up = delta.clip(lower=0)
    down = -delta.clip(upper=0)
    avg_gain = up.rolling(window=period).mean()
    avg_loss = down.rolling(window=period).mean()
    rs = avg_gain / avg_loss
    rsi = 100 - (100 / (1 + rs))
    return rsi
```

## Moving Average Convergence Divergence (MACD) Line calculation :

```
def compute_MACD(data, short_period=12, long_period=26, signal_period=9):
    ema_short = data['Close'].ewm(span=short_period, adjust=False).mean()
    ema_long = data['Close'].ewm(span=long_period, adjust=False).mean()
    macd_line = ema_short - ema_long
    signal_line = macd_line.ewm(span=signal_period, adjust=False).mean()
    return macd_line, signal_line
```

## Upper Bollinger Band Formula

```
def compute_BollingerBands(data, window=20, num_std=2):
    sma = data['Close'].rolling(window=window).mean()
    std = data['Close'].rolling(window=window).std()
    upper_band = sma + num_std * std
    lower_band = sma - num_std * std
    return sma, upper_band, lower_band
```

### Geodesic indicators calculation :

```
def calculate_indicators(data):
    data['SMA_short'] = data['Close'].rolling(window=20).mean()
    data['SMA_long'] = data['Close'].rolling(window=50).mean()
    macd_line, signal_line = compute_MACD(data)
    data['MACD'] = macd_line
    data['MACD_signal'] = signal_line
    data['RSI'] = compute_RSI(data['Close'], 14)
    bb_sma, upper_band, lower_band = compute_BollingerBands(data)
    data['BB_MA'] = bb_sma
    data['BB_upper'] = upper_band
    data['BB_lower'] = lower_band
    data = data.dropna()
    return data
```

### Reshaping for LSTM Input Format

```
def prepare_lstm_data(data, seq_len):
    scaler = MinMaxScaler(feature_range=(0, 1))
    scaled = scaler.fit_transform(data['Close'].values.reshape(-1, 1))
    X, y = [], []
    for i in range(seq_len, len(scaled)):
        X.append(scaled[i - seq_len:i, 0])
        y.append(scaled[i, 0])
    X, y = np.array(X), np.array(y)
    X = np.reshape(X, (X.shape[0], X.shape[1], 1))
    return X, y, scaler
```

### Model Compilation

```
def build_lstm_model(input_shape):
    model = Sequential()
    model.add(LSTM(50, return_sequences=True, input_shape=input_shape))
    model.add(LSTM(50))
    model.add(Dense(25))
    model.add(Dense(1))
    model.compile(optimizer='adam', loss='mean_squared_error')
    return model
```

### Signal Generation Using numPy first-order diff :

```
def generate_lstm_signals(predictions):
    signals = np.where(np.diff(predictions.flatten(), prepend=predictions[0]) > 0, 1, -1)
    return signals


def backtest_strategy(data, signals):
    initial_capital = 10000.0
    portfolio = pd.DataFrame(index=data.index)
    portfolio['signal'] = signals
    portfolio['price'] = data['Close']
    portfolio['holdings'] = 0.0
    portfolio['cash'] = initial_capital
    portfolio['total'] = initial_capital
    in_position = False
    shares = 0.0

    for i in range(1, len(portfolio)):
        if signals.iloc[i] == 1 and not in_position:
            shares = portfolio['cash'].iloc[i - 1] / portfolio['price'].iloc[i]
            in_position = True
            portfolio.iloc[i, portfolio.columns.get_loc('holdings')] = shares * portfolio['price'].iloc[i]
            portfolio.iloc[i, portfolio.columns.get_loc('cash')] = 0.0
        elif signals.iloc[i] == -1 and in_position:
            in_position = False
            portfolio.iloc[i, portfolio.columns.get_loc('cash')] = shares * portfolio['price'].iloc[i]
            shares = 0.0
            portfolio.iloc[i, portfolio.columns.get_loc('holdings')] = 0.0
        else:
            if in_position:
                portfolio.iloc[i, portfolio.columns.get_loc('holdings')] = shares * portfolio['price'].iloc[i]
                portfolio.iloc[i, portfolio.columns.get_loc('cash')] = 0.0
            else:
                portfolio.iloc[i, portfolio.columns.get_loc('cash')] = portfolio['cash'].iloc[i - 1]
        portfolio.iloc[i, portfolio.columns.get_loc('total')] = portfolio['cash'].iloc[i] + portfolio['holdings'].iloc[i]
    return portfolio
```

### Drawdown Calculation

```
def compute_max_drawdown(series):
    cum_max = series.cummax()
    dd = (series - cum_max) / cum_max
    return dd.min()


def evaluate_performance(portfolio):
    total_return = portfolio['total'].iloc[-1] / portfolio['total'].iloc[0] - 1
    returns = portfolio['total'].pct_change().dropna()
    sharpe = (returns.mean() / returns.std()) * np.sqrt(252) if returns.std() != 0 else 0
    max_dd = compute_max_drawdown(portfolio['total'])
    return total_return, sharpe, max_dd
```

### Generating Scatter Plot Visualization

```
import matplotlib.pyplot as plt

def plot_price(data, signals):
    buys = data.index[signals == 1]
    sells = data.index[signals == -1]

    plt.figure(figsize=(14, 7))
    plt.plot(data.index, data['Close'], label='Close Price', color='blue')
    plt.plot(data.index, data['SMA_short'], label='SMA Short', color='green')
    plt.plot(data.index, data['SMA_long'], label='SMA Long', color='red')

    # Scatter plot for Buy and Sell signals
    plt.scatter(buys, data.loc[buys, 'Close'], marker='^', color='green', label='Buy Signal', s=100)
    plt.scatter(sells, data.loc[sells, 'Close'], marker='v', color='red', label='Sell Signal', s=100)

    plt.title("AAPL Price with SMAs and Trading Signals")
    plt.xlabel("Date")
    plt.ylabel("Price")
    plt.legend()
    plt.grid(True)
    plt.show()
```

## Portfolio Visualization

```
def plot_portfolio(portfolio):
    plt.figure(figsize=(14,7))
    plt.plot(portfolio.index, portfolio['total'], label='Total Portfolio Value', color='orange')
    plt.title("Portfolio Value Over Time")
    plt.xlabel("Date")
    plt.ylabel("Total Value ($)")
    plt.legend()
    plt.show()
```

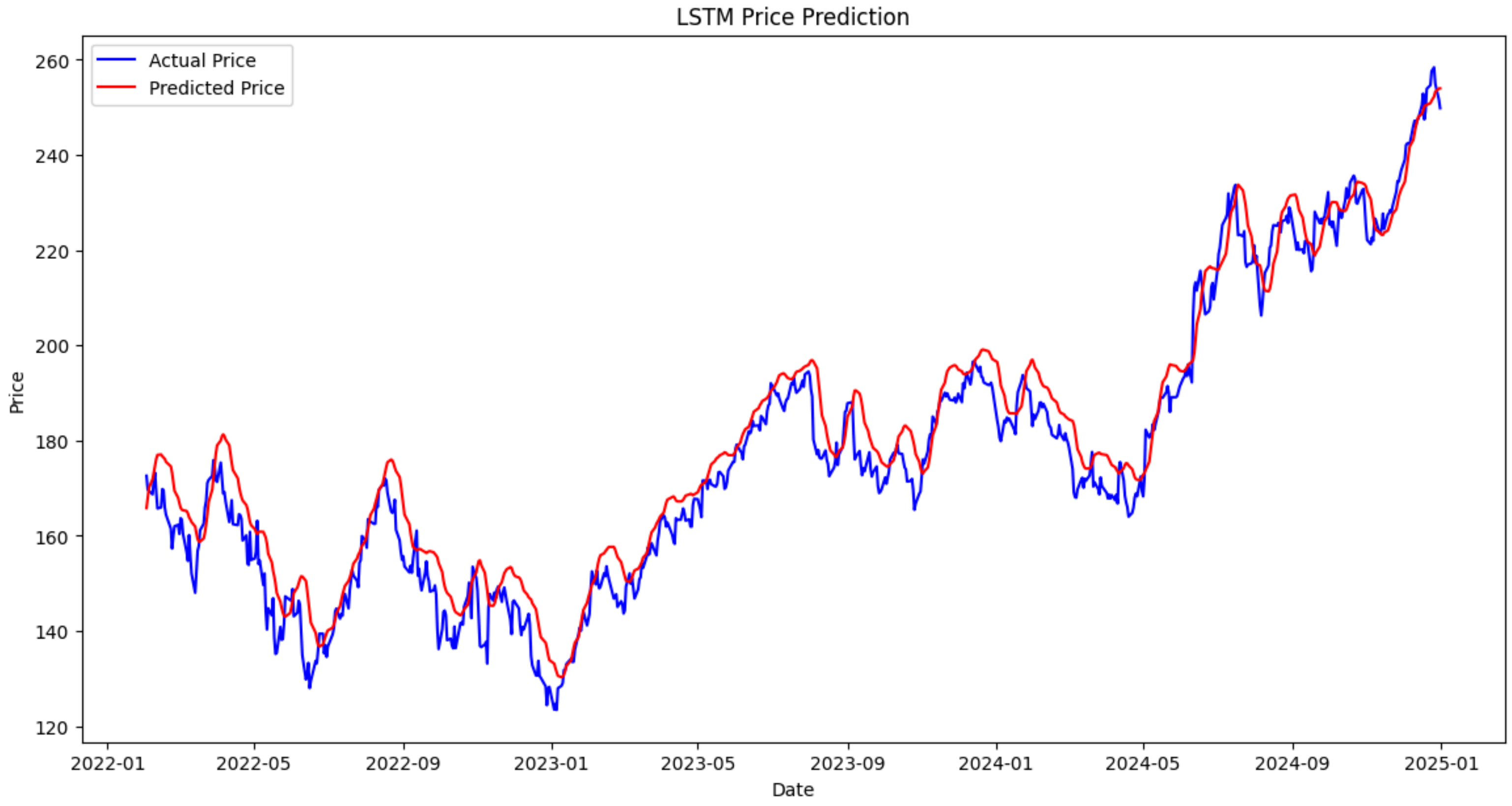
## LSTM final price prediction

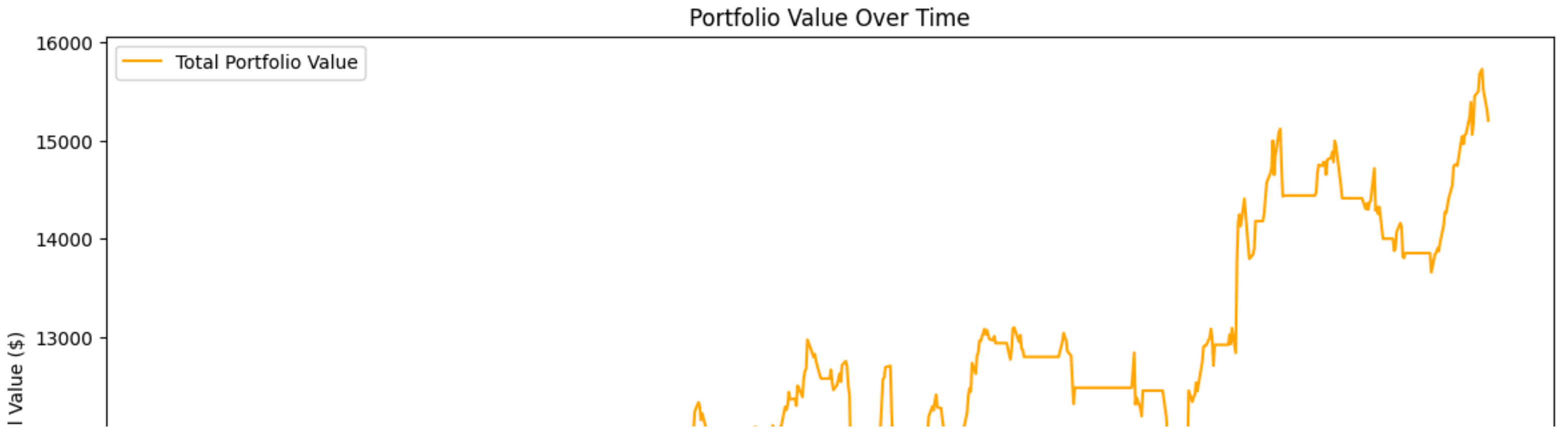
```
def plot_predictions(test_data, y_test_inv, predictions_inv):
    plt.figure(figsize=(14,7))
    plt.plot(test_data.index, y_test_inv.flatten(), label='Actual Price', color='blue')
    plt.plot(test_data.index, predictions_inv.flatten(), label='Predicted Price', color='red')
    plt.title("LSTM Price Prediction")
    plt.xlabel("Date")
    plt.ylabel("Price")
    plt.legend()
    plt.show()
```

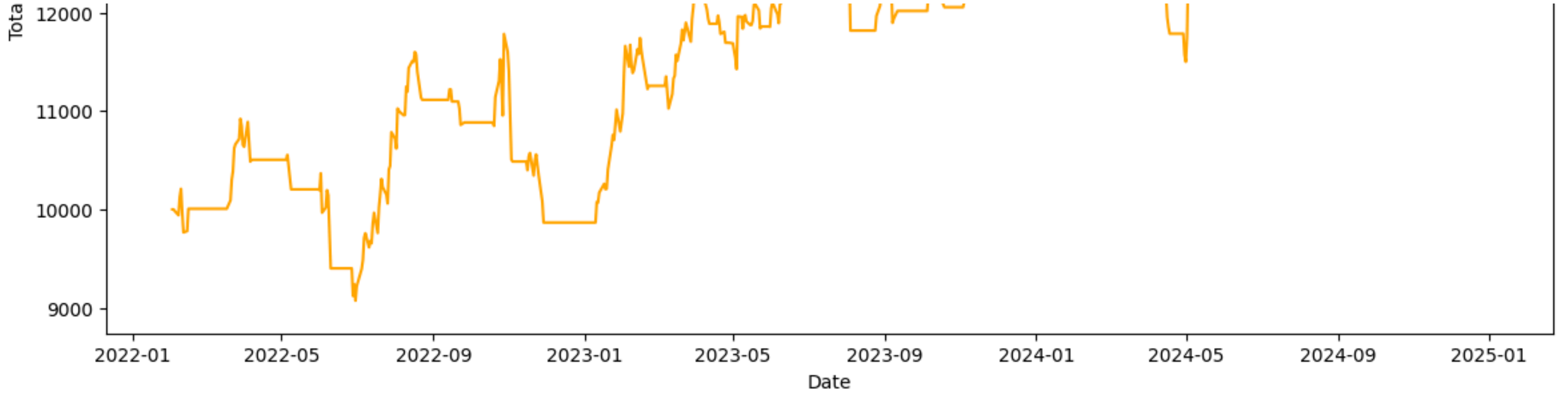
## Trading Signal Visualization

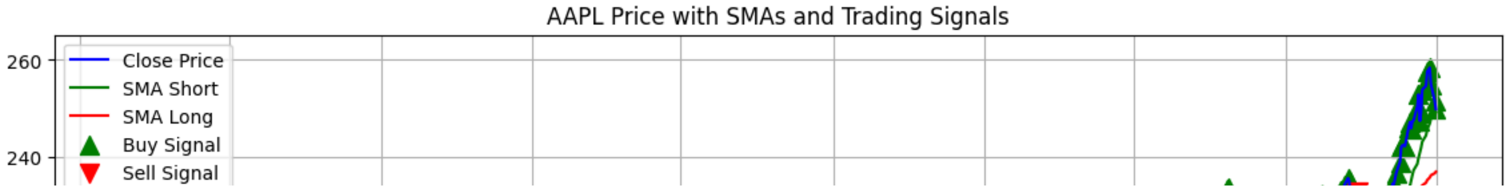
```
def main():
    ticker = 'AAPL'
    start = '2010-01-10'
    end = '2025-01-01'
    data = download_data(ticker, start, end)
    data = calculate_indicators(data)
    seq_len = 60
    X, y, scaler = prepare_lstm_data(data, seq_len)
    split = int(0.8 * len(X))
    X_train, y_train = X[:split], y[:split]
    X_test, y_test = X[split:], y[split:]
    model = build_lstm_model((X_train.shape[1], 1))
    model.fit(X_train, y_train, epochs=10, batch_size=32, verbose=0)
    predictions = model.predict(X_test)
    predictions_inv = scaler.inverse_transform(predictions)
    y_test_inv = scaler.inverse_transform(y_test.reshape(-1, 1))
    test_data = data[-len(y_test_inv):]
    lstm_signals = generate_lstm_signals(predictions_inv)
    lstm_signals_series = pd.Series(lstm_signals, index=test_data.index)
    portfolio_lstm = backtest_strategy(test_data, lstm_signals_series)
    ret_lstm, sharpe_lstm, dd_lstm = evaluate_performance(portfolio_lstm)
    print("LSTM Strategy Return:", ret_lstm)
    print("LSTM Strategy Sharpe:", sharpe_lstm)
    print("LSTM Strategy Max Drawdown:", dd_lstm)
    plot_predictions(test_data, y_test_inv, predictions_inv)
    plot_portfolio(portfolio_lstm)
    plot_price(test_data, lstm_signals_series)


if __name__ == '__main__':
    main()
```

```
/tmp/ipython-input-3-4269589612.py:2: FutureWarning: YF.download() has changed argument auto_adjust default to True
  data = yf.download(ticker, start=start, end=end)
[*********************100%***********************]  1 of 1 completed
/usr/local/lib/python3.11/dist-packages/keras/src/layers/rnn/rnn.py:200: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefe
  super().__init__(**kwargs)
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step
LSTM Strategy Return: 0.520233499397089
LSTM Strategy Sharpe: 0.8979670150205906
LSTM Strategy Max Drawdown: -0.16916856432516608
```








AAPL Price with SMAs and Trading Signals