Assignment: **Reinforcement Learning for Stock Market Trading Strategy**

Objective:

Implement a Reinforcement Learning (RL)-based trading strategy using one of the following algorithms: Q-Learning, DQN, Double DQN, or Dueling DQN. The agent should learn an optimal policy to trade stocks based on historical price movements and technical indicators. You can choose to focus on intraday trading, swing trading, or long-term investing.

Steps to Complete the Assignment:

1. Data Collection & Preprocessing

- Select a market: Choose a stock market dataset (e.g., NSE/BSE, NYSE, NASDAQ).
- Choose a timeframe: Select intraday (5min, 15min, 1-hour), daily, or weekly data.
- Obtain stock price data using APIs like Yahoo Finance, Alpha Vantage, or Quandl.
- Compute technical indicators such as:
    - Trend indicators: Moving Averages (SMA, EMA), MACD.
    - Momentum indicators: RSI, Stochastic Oscillator.
    - Volatility indicators: Bollinger Bands, ATR.
    - Volume indicators: OBV, VWAP.
- Normalize the dataset to improve training performance.

---

2. Define the RL Components

State Space (Observations):

- Use a **feature vector representation** of technical indicators and price history.
- Example state representation:

$$S_t = [RSI_t, MACD_t, Bollinger\ Bands_t, Volume_t, ..., S_{t-1}, S_{t-2}, ...]$$

- You can choose a **single time step** or a **sequence of past time steps** (for LSTM-based architectures).

Action Space:

The agent should decide one of the following at each time step:

- Buy: Enter a long position.
- Sell: Exit or short a position.

- Hold: No action.

Reward Function:

- Profit-based reward: Reward the agent based on the profit/loss of executed trades.
- Penalties: Apply penalties for excessive trades to prevent overtrading.
- Drawdown management: Introduce negative rewards for high drawdowns.

---

3. Choose One of the RL Algorithms

Select one from the following:

- Q-Learning: A table-based RL method for simple state-action mapping.
- DQN (Deep Q-Network): Uses deep learning to approximate Q-values.
- Double DQN: Improves DQN by reducing overestimation bias.
- Dueling DQN: Enhances DQN by separately estimating state value and action advantage.

---

4. Implement the RL Agent

- For Q-Learning:
    - Define a Q-table mapping states to actions.
    - Use the Bellman Equation for Q-value updates.
- For DQN, Double DQN, and Dueling DQN:
    - Design a neural network architecture with fully connected layers.
    - Implement experience replay and target network updates.
    - Train using mini-batches of past experiences.
- **For Dueling DQN:**
    - Create two separate streams: **State-value function** $V(s)$ and **Action-advantage function** $A(s,a)$.
    - Use the **dueling Q-value formula**:

$$Q(s,a) = V(s) + A(s,a) - \frac{1}{|\mathcal{A}|}\sum_{a'} A(s,a')$$

---

5. Train and Evaluate the Model

- Training:
    - Split the dataset into training and testing sets.
    - Train the RL agent using reward maximization over multiple episodes.
- Evaluation Metrics:
    - Total Returns
    - Sharpe Ratio
    - Maximum Drawdown
    - Win/Loss Ratio
    - Annualized Volatility
- Baseline Comparison:
    - Compare the RL strategy against a rule-based strategy (e.g., Moving Average Crossover).
    - Compare with a machine learning model (e.g., Random Forest, LSTM).

---

6. Visualizations & Report

- Equity Curve: Show the cumulative profit/loss over time.
- Trading Signals on Price Chart: Overlay buy/sell signals on historical price data.
- Q-Values Over Time: Plot how the agent's decision-making evolves.
- Final Report:
    - Explain the model architecture and training process.
    - Discuss results, strengths, and limitations.
    - Compare RL performance with baseline strategies.

Deliverables:

1. Python Code (Google Colab/Jupyter Notebook).
2. Report (PDF/Markdown) summarizing findings.
3. Visualizations of results and comparisons.