

# CLI-Based Binance USDT-M Futures Trading Bot

Submitted by: Janvi Singh

Dated: 13/12/25

## OBJECTIVE:

The goal of this project is to build a command-line trading bot for Binance USDT-M Futures. The bot can place different types of orders like market, limit, OCO, TWAP, and grid orders. It also checks inputs for errors and keeps a log of all actions, so users can safely and easily test trading strategies on the Binance testnet.

## File structure:

### BinanceBot

```
|
| └─ src/                # All source code
|   └─ market_orders.py  # Market order logic
|   └─ limit_orders.py   # Limit order logic
|   └─ grid_orders.py     # Grid order logic
|   └─ advanced/         # Advanced order strategies
|       └─ oco.py        # OCO order logic
|       └─ twap.py       # TWAP order logic
|
| └─ bot.log              # Log file for actions, errors, executions
| └─ .env                 # Environment file with API_KEY and API_SECRET
| └─ README.md            # Instructions, setup, and usage
```

└─ report.pdf

# PDF report with screenshots, explanation

### Features Implemented:

This trading bot supports a variety of order types to cater to different trading strategies:

1. **Market Orders** – Executes a trade immediately at the current market price.
2. **Limit Orders** – Places an order at a specific price, only executing when the market reaches the set price.
3. **Stop-Limit Orders** – Triggers a limit order when the stop price is reached, allowing controlled entry or exit.
4. **OCO (One-Cancels-the-Other) Orders** – Simultaneously places a take-profit and stop-loss order; execution of one automatically cancels the other.
5. **TWAP (Time-Weighted Average Price) Orders** – Splits large orders into smaller chunks over a defined period to minimize market impact.
6. **Grid Orders** – Automates buy-low/sell-high strategy within a predefined price range, creating multiple orders above and below the current price.

### Validation & Logging

- **Validation:** Ensures symbol ends with USDT, quantity > 0, price > 0. Invalid inputs stop the order.
- **Logging:** All actions, responses, and errors are logged in bot.log.
- Example:

2025-12-13 12:45 [INFO] Placing LIMIT order: BUY 0.01 BTCUSDT at 45000

2025-12-13 12:45 [INFO] Response: {'orderId': 123456, 'status': 'NEW'}

## Flowchart of Bot Workflow

User CLI Input

|



Input Validation (symbol, quantity, price)

|



Order Type Selection

(Market / Limit / OCO / TWAP / Grid)

|



API Call to Binance Testnet

|



Logging & Confirmation

(Console output + bot.log)

|



Order Execution Result

Demonstration of Bot Functionality

### 1. Market Order

Command Used: `python src/market_orders.py BTCUSDT BUY 0.01`

The screenshot shows the Visual Studio Code editor with the file explorer on the left. The 'market\_order.py' file is open in the editor. The code includes imports for 'dotenv' and 'load\_dotenv', environment variable loading for 'BINANCE\_API\_KEY' and 'BINANCE\_API\_SECRET', and a logging setup. The terminal window at the bottom shows the command 'python src/market\_order.py BTCUSDT BUY 0.01' being executed, resulting in a simulated market order response.

```
src > market_order.py
4 from dotenv import load_dotenv
5
6 # Load .env
7 load_dotenv()
8 API_KEY = os.getenv("BINANCE_API_KEY")
9 API_SECRET = os.getenv("BINANCE_API_SECRET")
10
11 # Logging setup
12 logging.basicConfig(
13     filename='../bot.log',
14     level=logging.INFO,
15     format='%(asctime)s [%(levelname)s] %(message)s'
16 )
```

Terminal Output:

```
PS C:\Projects\BinanceBot> python src/market_order.py BTCUSDT BUY 0.01
[SIMULATION] Market order executed: {'symbol': 'BTCUSDT', 'side': 'BUY', 'type': 'MARKET', 'quantity': 0.01, 'status': 'SIMULATED'}
```

The screenshot shows the Visual Studio Code editor with the file explorer on the left. The 'bot.log' file is open in the editor, displaying a series of log messages. The terminal window at the bottom shows the command 'python src/market\_order.py BTCUSDT BUY 0.01' being executed, resulting in a simulated market order response.

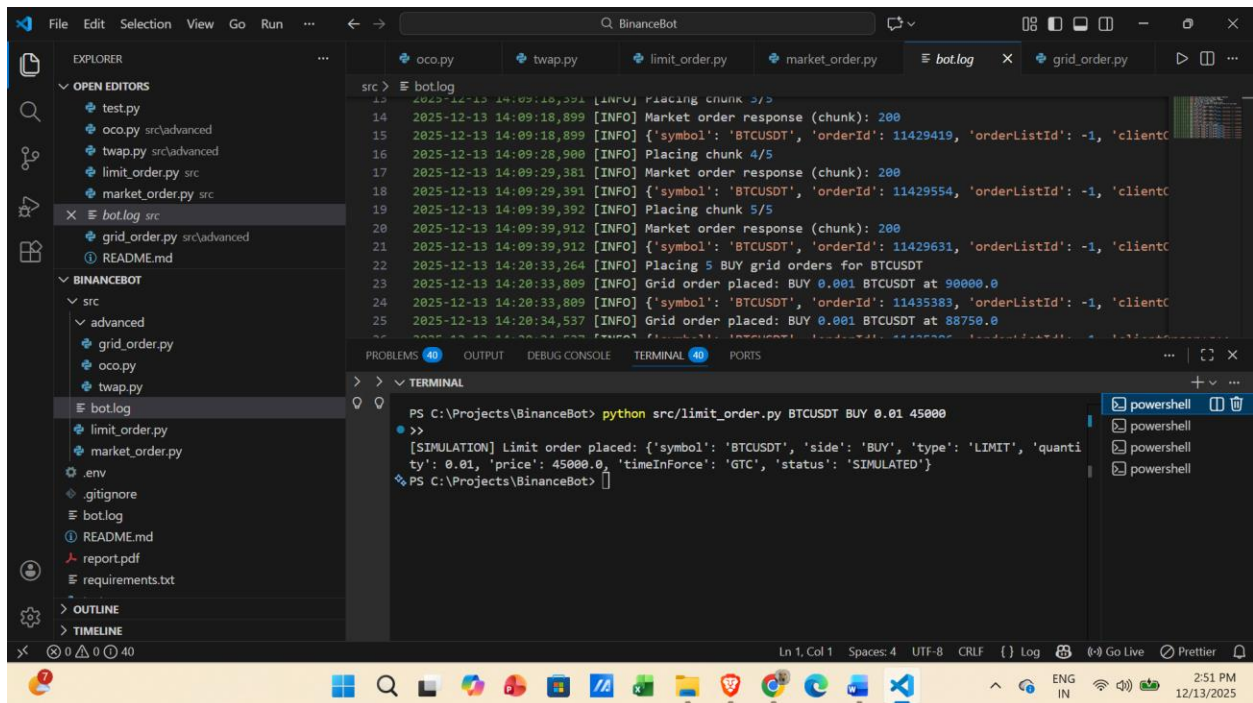
```
src > bot.log
14 2025-12-13 14:09:18,899 [INFO] Placing chunk 3/5
15 2025-12-13 14:09:18,899 [INFO] Market order response (chunk): 200
16 2025-12-13 14:09:18,899 [INFO] {'symbol': 'BTCUSDT', 'orderId': 11429419, 'orderListId': -1, 'clientOrderId': 11429419}
17 2025-12-13 14:09:28,908 [INFO] Placing chunk 4/5
18 2025-12-13 14:09:29,381 [INFO] Market order response (chunk): 200
19 2025-12-13 14:09:29,381 [INFO] {'symbol': 'BTCUSDT', 'orderId': 11429554, 'orderListId': -1, 'clientOrderId': 11429554}
20 2025-12-13 14:09:39,392 [INFO] Placing chunk 5/5
21 2025-12-13 14:09:39,912 [INFO] Market order response (chunk): 200
22 2025-12-13 14:09:39,912 [INFO] {'symbol': 'BTCUSDT', 'orderId': 11429631, 'orderListId': -1, 'clientOrderId': 11429631}
23 2025-12-13 14:20:33,264 [INFO] Placing 5 BUY grid orders for BTCUSDT
24 2025-12-13 14:20:33,809 [INFO] Grid order placed: BUY 0.001 BTCUSDT at 90000.0
25 2025-12-13 14:20:33,809 [INFO] {'symbol': 'BTCUSDT', 'orderId': 11435383, 'orderListId': -1, 'clientOrderId': 11435383}
26 2025-12-13 14:20:34,537 [INFO] Grid order placed: BUY 0.001 BTCUSDT at 88750.0
```

Terminal Output:

```
PS C:\Projects\BinanceBot> python src/market_order.py BTCUSDT BUY 0.01
[SIMULATION] Market order executed: {'symbol': 'BTCUSDT', 'side': 'BUY', 'type': 'MARKET', 'quantity': 0.01, 'status': 'SIMULATED'}
```

## 2. Limit Order

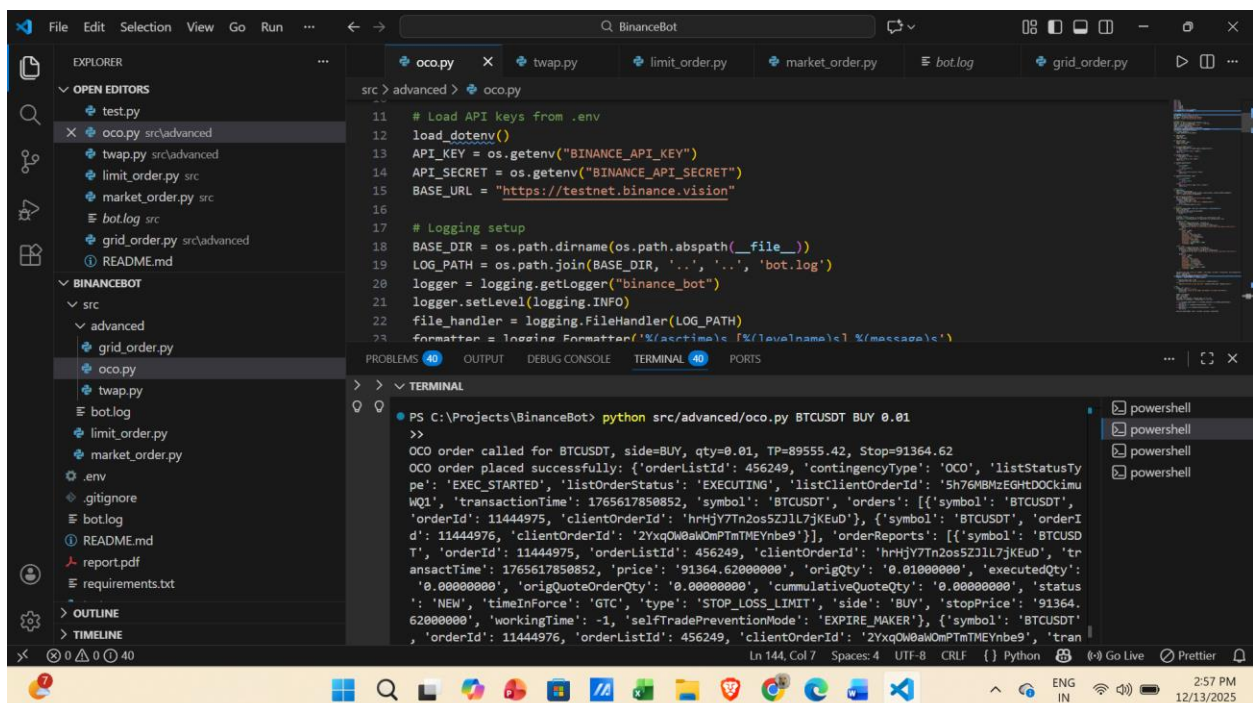
Command Used: python src/limit\_orders.py BTCUSDT SELL 0.01 45000



### 3. OCO Order

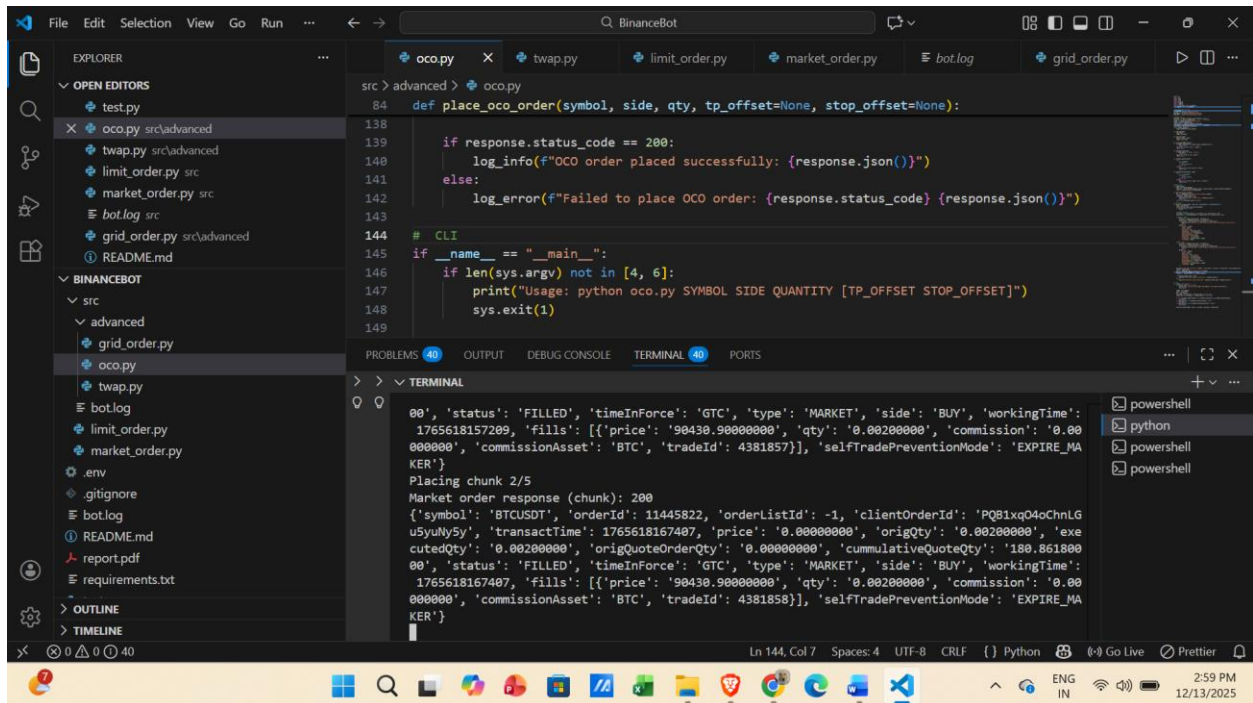
Command Used:

python src/advanced/oco.py BTCUSDT BUY 0.01



#### 4. Twap order:

Command: `python src/advanced/twap.py BTCUSDT BUY 0.01 5 10`



The screenshot shows a Visual Studio Code editor window with the following components:

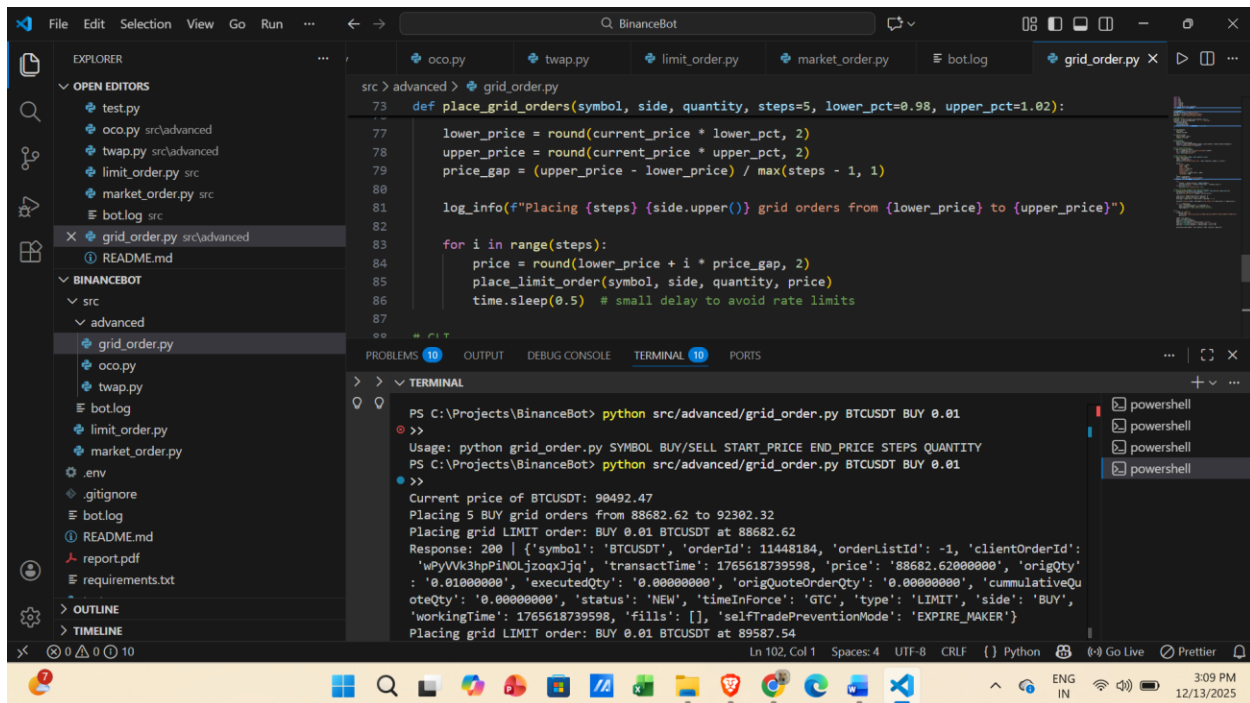
- EXPLORER:** Shows the project structure with folders 'OPEN EDITORS' and 'BINANCEBOT'. The 'BINANCEBOT' folder contains 'src' and 'advanced' subfolders. The 'advanced' folder contains 'grid\_order.py', 'twap.py', 'limit\_order.py', 'market\_order.py', 'bot.log', and 'README.md'.
- EDITOR:** Displays the code for 'twap.py'. The code defines a `place_oco_order` function and a CLI interface. The function takes parameters: `symbol`, `side`, `qty`, `tp_offset`, and `stop_offset`. It logs the order placement status and handles errors. The CLI interface takes arguments for symbol, side, quantity, and time intervals.
- TERMINAL:** Shows the output of the script execution. It displays the status of the order as 'FILLED', the time in force as 'GTC', and the type as 'MARKET'. It also shows the order details, including the symbol, order ID, and the quantity.

```
def place_oco_order(symbol, side, qty, tp_offset=None, stop_offset=None):
    138
    139     if response.status_code == 200:
    140         log_info(f"OCO order placed successfully: {response.json()}")
    141     else:
    142         log_error(f"Failed to place OCO order: {response.status_code} {response.json()}")
    143
    144 # CLI
    145 if __name__ == "__main__":
    146     if len(sys.argv) not in [4, 6]:
    147         print("Usage: python oco.py SYMBOL SIDE QUANTITY [TP_OFFSET STOP_OFFSET]")
    148         sys.exit(1)
    149
```

```
00', 'status': 'FILLED', 'timeInForce': 'GTC', 'type': 'MARKET', 'side': 'BUY', 'workingTime':
1765618157209, 'fills': [{'price': '90430.90000000', 'qty': '0.00200000', 'commission': '0.00
000000', 'commissionAsset': 'BTC', 'tradeId': 4381857}], 'selfTradePreventionMode': 'EXPIRE_M
KER'}
Placing chunk 2/5
Market order response (chunk): 200
{'symbol': 'BTCUSDT', 'orderId': 11445822, 'orderListId': -1, 'clientOrderId': 'PQ81xq04sChnLG
u5yulky5y', 'transactTime': 1765618167407, 'price': '0.00000000', 'origQty': '0.00200000', 'ex
cutedQty': '0.00200000', 'origQuoteOrderQty': '0.00000000', 'cumulativeQuoteQty': '180.861800
00', 'status': 'FILLED', 'timeInForce': 'GTC', 'type': 'MARKET', 'side': 'BUY', 'workingTime':
1765618167407, 'fills': [{'price': '90430.90000000', 'qty': '0.00200000', 'commission': '0.00
000000', 'commissionAsset': 'BTC', 'tradeId': 4381858}], 'selfTradePreventionMode': 'EXPIRE_M
KER'}
```

#### 5. Grid order

Command: `python src/advanced/grid_order.py BTCUSDT BUY 0.01 90000 91000 5`



## Key Highlights

- Works for both real and testnet accounts.
- Simulation mode ensures testing without funds.
- Modular CLI-based architecture allows easy extension.
- Logs all actions for debugging and audit purposes.
- 

## Conclusion:

- This project demonstrates a fully functional Binance USDT-M Futures trading bot with both core and advanced order types. Logging, validation, and simulation ensure safe and testable operations.

