

# 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/
|       ├── 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 interface with the following details:

- File Explorer:** On the left, it lists files under "src" and "BINANCEBOT".
- Editor:** The main editor area displays a Python script named "market\_order.py".
- Terminal:** The bottom terminal shows command-line output for executing the script.
- Bottom Bar:** Includes icons for file operations, a search bar, and various extensions like Prettier and Go Live.

```
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

```
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'}
PS C:\Projects\BinanceBot>
```

The screenshot shows a Visual Studio Code (VS Code) interface with the following details:

- File Explorer (Left):** Shows the project structure under "BINANCEBOT".
  - src:** Contains advanced, oco.py, twap.py, limit\_order.py, market\_order.py, and bot.log.
  - bot.log:** Selected in the Explorer.
  - README.md**
  - env**
  - gitignore**
  - bot.log**
  - REPORT.pdf**
  - requirements.txt**
- Editor (Top):** Displays the content of the selected file, "bot.log". The log output shows several INFO-level messages related to order placement and execution for the BTCUSDT market.
- Terminal (Bottom):** Shows a PowerShell session running a Python script to place a market order for 0.01 BTC at 88750.0.

```
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'}
```
- Status Bar (Bottom):** Shows the current file (Ln 1, Col 1), code style settings (Spaces: 4, CRLF), and other system information (Log, Go Live, Prettier).

## 2. Limit Order

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

```

src > PS C:\Projects\BinanceBot> python src/limit_order.py BTCUSDT BUY 0.01 45000
>>>
[SIMULATION] Limit order placed: {'symbol': 'BTCUSDT', 'side': 'BUY', 'type': 'LIMIT', 'quantity': 0.01, 'price': 45000.0, 'timeInForce': 'GTC', 'status': 'SIMULATED'}
PS C:\Projects\BinanceBot>

```

### 3. OCO Order

Command Used:

`python src/advanced/oco.py BTCUSDT BUY 0.01`

```

src > PS C:\Projects\BinanceBot> python src/advanced/oco.py BTCUSDT BUY 0.01
>>>
OCO order called for BTCUSDT, side=BUY, qty=0.01, TP=89555.42, Stop=91364.62
OCO order placed successfully: {'orderId': 456249, 'contingencyType': 'OCO', 'listStatusType': 'EXEC_STARTED', 'listOrderStatus': 'EXECUTING', 'listClientOrderId': '5h76MBMzE8HtDOCKiMuWQ1', 'transactionTime': 1765617850852, 'symbol': 'BTCUSDT', 'orders': [{"symbol": "BTCUSDT", "orderId": 11444976, "clientOrderId": "2Yxq0Wak0mPTmTMEYnb9"}, {"symbol": "BTCUSDT", "orderId": 11444975, "clientOrderId": "2Yxq0Wak0mPTmTMEYnb9"}]}, 'orderReports': [{"symbol": "BTCUSD T", "orderId": 11444975, "clientOrderId": 456249, "clientOrderId": "5h76MBMzE8HtDOCKiMuWQ1", "transactionTime": 1765617850852, "price": 91364.62000000, "origQty": "0.01000000", "executedQty": "0.00000000", "origQuoteOrderQty": "0.00000000", "cumulativeQuoteQty": "0.00000000", "status": "NEW", "timeInForce": "GTC", "type": "STOP_LOSS_LIMIT", "side": "BUY", "stopPrice": 91364.62000000, "workingTime": -1, "selfTradePreventionMode": "EXPIRE_MAKER"}, {"symbol": "BTCUSDT", "orderId": 11444976, "clientOrderId": "2Yxq0Wak0mPTmTMEYnb9", "tran

```

#### 4. Twap order:

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

The screenshot shows a code editor interface with multiple tabs open. The left sidebar displays a file tree under 'EXPLORER' and 'BINANCEBOT'. The main area shows several Python files: 'oco.py', 'twap.py', 'limit\_order.py', 'market\_order.py', 'bot.log', and 'grid\_order.py'. The 'oco.py' tab is active, displaying code for placing OCO orders. The terminal below shows command-line output related to market order responses.

```
src/advanced> oco.py
84     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
```

TERMINAL

```
00', 'status': 'FILLED', 'timeInForce': 'GTC', 'type': 'MARKET', 'side': 'BUY', 'workingTime': 1765618157209, 'fills': [{"price": "90438.90000000", "qty": "0.00200000", "commission": "0.000000", "commissionAsset": "BTC", "tradeId": 4381857}], 'selfTradePreventionMode': 'EXPIRE_MAKER'}  
Placing chunk 2/5  
Market order response (chunk): 200  
{'symbol': 'BTCUSDT', 'orderId': 11445822, 'orderListId': -1, 'clientOrderId': 'PQB1xqO4oChNgLusyvN5y', 'transactTime': 1765618167407, 'price': '0.00000000', 'origQty': '0.00200000', 'executedQty': '0.00200000', 'origQuoteOrderQty': '0.00000000', 'cumulativeQuoteQty': '188.86180000', 'status': 'FILLED', 'timeInForce': 'GTC', 'type': 'MARKET', 'side': 'BUY', 'workingTime': 1765618167407, 'fills': [{"price": "90438.90000000", "qty": "0.00200000", "commission": "0.000000", "commissionAsset": "BTC", "tradeId": 4381858}], 'selfTradePreventionMode': 'EXPIRE_MAKER'}
```

## 5. Grid order

Command: python src/advanced/grid\_order.py BTCUSDT BUY 0.01 90000 91000 5

The screenshot shows the Visual Studio Code interface with the following details:

- Explorer View:** Shows the project structure under "src\advanced". The file "grid\_order.py" is currently selected.
- Code Editor:** Displays the Python code for "grid\_order.py". The code defines a function "place\_grid\_orders" that places multiple grid orders based on current price, lower price, upper price, and steps.
- Terminal:** Shows the command-line output of running "grid\_order.py" for buying BTCUSDT at a price of 0.01. It includes usage instructions and a detailed response from the Binance API showing the placement of five grid orders.
- Bottom Status Bar:** Provides information like line count (Ln 102), column count (Col 1), file encoding (UTF-8), and date/time (12/13/2025).

## 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.

