# Step-by-Step Blueprint for a Dexscreener Script

## 1. Objective

The goal is to create a script that fetches token data from the Dexscreener API, filters tokens based on specific criteria, and sends notifications (via Telegram) for tokens that meet these criteria.

## 2. Prerequisites

Before starting, ensure you have the following:

* 1. Python installed: Python 3.x is recommended.
* 2. Dexscreener API access: No API key is needed for basic access.
* 3. Telegram Bot token: Generated using BotFather on Telegram.
* 4. Chat ID: Use tools like IDBot to get your chat ID.

5. Python Libraries: Install required libraries using:

pip install requests pandas python-telegram-bot

## 3. Key Features

The script will:

* 1. Fetch token data from the Dexscreener API.
* 2. Filter tokens based on the following criteria:
* - Minimum 24-hour trading volume (e.g., $500,000).
* - Minimum liquidity (e.g., $100,000).
* - Minimum price difference across DEXs (e.g., 3%).
* 3. Focus on tokens traded on Binance Smart Chain (BSC).
* 4. Send real-time alerts for matching tokens via Telegram.

## 4. Full Python Script

import requests  
from telegram import Bot  
  
# Telegram Bot Configuration  
TELEGRAM\_BOT\_TOKEN = 'your\_telegram\_bot\_token'  
TELEGRAM\_CHAT\_ID = 'your\_telegram\_chat\_id'  
  
# Dexscreener API URL for Binance Smart Chain (BSC)  
API\_URL = "https://api.dexscreener.com/latest/dex/search/?chain=bsc"  
  
# Criteria for Filtering Tokens  
MIN\_VOLUME = 500000 # Minimum daily trading volume in USD  
MIN\_LIQUIDITY = 100000 # Minimum liquidity in USD  
MIN\_PRICE\_DIFFERENCE = 3 # Minimum price difference percentage  
  
# Function to Fetch Token Data  
def fetch\_tokens():  
 try:  
 response = requests.get(API\_URL)  
 response.raise\_for\_status()  
 return response.json()  
 except Exception as e:  
 print(f"Error fetching data: {e}")  
 return None  
  
# Function to Filter Tokens  
def filter\_tokens(data):  
 filtered = []  
 for token in data['pairs']:  
 volume = float(token.get('volume', 0))  
 liquidity = float(token.get('liquidity', {}).get('usd', 0))  
 price\_diff = float(token.get('priceDifference', 0))  
  
 if volume >= MIN\_VOLUME and liquidity >= MIN\_LIQUIDITY and price\_diff >= MIN\_PRICE\_DIFFERENCE:  
 filtered.append({  
 'name': token.get('baseToken', {}).get('name', 'Unknown'),  
 'symbol': token.get('baseToken', {}).get('symbol', 'Unknown'),  
 'volume': volume,  
 'liquidity': liquidity,  
 'price\_difference': price\_diff,  
 'dexes': [market['name'] for market in token.get('markets', [])]  
 })  
 return filtered  
  
# Function to Send Telegram Alert  
def send\_telegram\_alert(tokens):  
 bot = Bot(token=TELEGRAM\_BOT\_TOKEN)  
 for token in tokens:  
 message = (  
 f"🚀 Token: {token['name']} ({token['symbol']})\n"  
 f"📊 Volume: ${token['volume']:,}\n"  
 f"💧 Liquidity: ${token['liquidity']:,}\n"  
 f"🔄 Price Difference: {token['price\_difference']}%\n"  
 f"🔗 Dexes: {', '.join(token['dexes'])}\n"  
 )  
 bot.send\_message(chat\_id=TELEGRAM\_CHAT\_ID, text=message)  
  
# Main Function  
def main():  
 print("Fetching token data...")  
 data = fetch\_tokens()  
 if data:  
 print("Filtering tokens...")  
 tokens = filter\_tokens(data)  
 if tokens:  
 print(f"Found {len(tokens)} tokens meeting the criteria.")  
 send\_telegram\_alert(tokens)  
 else:  
 print("No tokens meet the criteria.")  
 else:  
 print("Failed to fetch token data.")  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

## 5. Steps to Implement

1. 1. Set Up Telegram Bot:

* - Create a bot using BotFather.
* - Replace 'your\_telegram\_bot\_token' with the bot's token.
* - Get your chat ID using IDBot and replace 'your\_telegram\_chat\_id'.

1. 2. Define Criteria:

* - Adjust MIN\_VOLUME, MIN\_LIQUIDITY, and MIN\_PRICE\_DIFFERENCE as per your requirements.

1. 3. Run the Script:

* Execute the script manually or set it to run periodically.

1. 4. Automate Execution:

* Use cron (Linux/Mac) or Task Scheduler (Windows) to run the script every 15 minutes or as needed.

## 6. Potential Enhancements

* 1. Add Logging: Record token data in a file or database for analysis.
* 2. Integrate Other Chains: Modify the API URL to monitor tokens on other chains (e.g., Ethereum, Polygon).
* 3. Build a Dashboard: Use tools like Streamlit to display token metrics in real-time.
* 4. Add Error Handling: Improve robustness for API downtime or connection issues.