```
import requests
import pandas as pd
import sqlite3
from bs4 import BeautifulSoup
from datetime import datetime, timedelta
import time
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import numpy as np
import csv
from telegram import Update, Bot
from telegram.ext import Updater, CommandHandler, CallbackContext

# Constants
PUMPFUN_URL = "https://pump.fun"
DEXSCREENER_API_URL = "https://api.dexscreener.com/latest/dex"
RUGCHECK_API_URL = "https://api.rugcheck.xyz/v1/token"
GMGN_API_URL = "https://api.gmgn.com"
DATABASE_FILE = "tokens.db"
CSV_FILE = "valid_tokens.csv"
HISTORICAL_DAYS = 30  # Number of days to retrieve historical data
API_KEY_RUGCHECK = "your_rugcheck_api_key"  # Replace with your RugCheck API key
TELEGRAM_BOT_TOKEN = "7757308442:AAELt9a1nGo0OMgqThi_H0BR4yfp-2k6EOw"  # Replace
with your Telegram bot token

# Headers for API requests
headers = {
    "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/91.0.4472.124 Safari/537.36",
    "Authorization": f"Bearer {API_KEY_RUGCHECK}"
}

# Database setup
def setup_database():
    """Create the database and tables if they don't exist."""
    conn = sqlite3.connect(DATABASE_FILE)
    cursor = conn.cursor()
    cursor.execute('''
        CREATE TABLE IF NOT EXISTS tokens (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            contract_address TEXT,
            name TEXT,
            symbol TEXT,
            trading_volume REAL,
            holders INTEGER,
            price_change REAL,
            success_probability REAL,
            status TEXT,
            timestamp DATETIME
        )
    ''')
    conn.commit()
    conn.close()
```

```
# Scrape PumpFun for new tokens
def scrape_pumpfun_tokens():
    """Scrape new token launches from PumpFun."""
    try:
        response = requests.get(PUMPFUN_URL, headers=headers)
        response.raise_for_status()
        soup = BeautifulSoup(response.text, 'html.parser')

        tokens = []
        for item in soup.select('.token-item'):  # Adjust the selector based on
the actual HTML
            name = item.select_one('.token-name').text.strip()
            link = item.select_one('.token-link')['href']
            tokens.append({"name": name, "link": link})

        return tokens
    except Exception as e:
        print(f"Error scraping PumpFun: {e}")
        return []

# Fetch DexScreener data
def fetch_dexscreener_data(token_address):
    """Fetch token data from DexScreener API."""
    try:
        response = requests.get(f"{DEXSCREENER_API_URL}/tokens/{token_address}")
        response.raise_for_status()
        return response.json()
    except Exception as e:
        print(f"Error fetching DexScreener data for token {token_address}: {e}")
        return None

# Fetch RugCheck analysis
def analyze_token(contract_address):
    """Analyze a token using the RugCheck API."""
    try:
        response = requests.get(
            f"{RUGCHECK_API_URL}/{contract_address}",
            headers=headers
        )
        response.raise_for_status()
        return response.json()
    except Exception as e:
        print(f"Error analyzing token {contract_address}: {e}")
        return None

# Fetch GMGN historical data
def fetch_gmgn_historical_data(token_address):
    """Fetch historical data for a token from GMGN API."""
    try:
        end_date = datetime.now()
        start_date = end_date - timedelta(days=HISTORICAL_DAYS)
        response = requests.get(
            f"{GMGN_API_URL}/historical",
            params={
                "token_address": token_address,
                "start_date": start_date.strftime("%Y-%m-%d"),
                "end_date": end_date.strftime("%Y-%m-%d")
            }
        )
        response.raise_for_status()
        return response.json()
    except Exception as e:
        print(f"Error fetching GMGN historical data for token {token_address}:
{e}")
        return None

# Prepare training data for ML model
def prepare_training_data():
    """Prepare training data for the ML model."""
    conn = sqlite3.connect(DATABASE_FILE)
    query = "SELECT trading_volume, holders, price_change, success FROM tokens
WHERE success IS NOT NULL"
    data = pd.read_sql(query, conn)
    conn.close()

    if data.empty:
        raise ValueError("No training data available. Please collect historical
data first.")

    X = data[['trading_volume', 'holders', 'price_change']]
    y = data['success']
    return X, y

# Train ML model
def train_model(X, y):
    """Train a RandomForestClassifier to predict token success."""
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
    model = RandomForestClassifier(n_estimators=100, random_state=42)
    model.fit(X_train, y_train)

    y_pred = model.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    print(f"Model accuracy: {accuracy:.2f}")
    return model

# Predict success probability
def predict_success_probability(model, token_data):
    """Predict the success probability of a token."""
    features = np.array([[token_data['trading_volume'], token_data['holders'],
token_data['price_change']]])
    probability = model.predict_proba(features)[0][1]  # Probability of success
(class 1)
    return probability

# Save valid tokens to database
def save_to_database(token_data):
    """Save valid tokens to the database."""
    conn = sqlite3.connect(DATABASE_FILE)
    cursor = conn.cursor()
    cursor.execute('''
        INSERT INTO tokens (contract_address, name, symbol, trading_volume,
holders, price_change, success_probability, status, timestamp)
        VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?)
    ''', (
        token_data['contract_address'],
        token_data['name'],
        token_data['symbol'],
        token_data['trading_volume'],
        token_data['holders'],
        token_data['price_change'],
        token_data['success_probability'],
        token_data['status'],
        datetime.now()
    ))
    conn.commit()
    conn.close()
    print(f"Token {token_data['contract_address']} saved to the database.")

# Save valid tokens to CSV
def save_to_csv(token_data):
    """Save valid tokens to a CSV file."""
    file_exists = False
    try:
        with open(CSV_FILE, 'r'):
            file_exists = True
    except FileNotFoundError:
        pass

    with open(CSV_FILE, mode='a', newline='') as file:
        writer = csv.writer(file)
        if not file_exists:
            writer.writerow(['Contract Address', 'Name', 'Symbol', 'Trading
Volume', 'Holders', 'Price Change', 'Success Probability', 'Status',
'Timestamp'])

        writer.writerow([
            token_data['contract_address'],
            token_data['name'],
            token_data['symbol'],
            token_data['trading_volume'],
            token_data['holders'],
            token_data['price_change'],
            token_data['success_probability'],
            token_data['status'],
            datetime.now().strftime("%Y-%m-%d %H:%M:%S")
        ])

    print(f"Token {token_data['contract_address']} saved to {CSV_FILE}.")

# Telegram command handler
```

```
def analyse_command(update: Update, context: CallbackContext):
    """Handle the /analyse command."""
    update.message.reply_text("Starting token analysis...")

    # Train the ML model
    try:
        X, y = prepare_training_data()
        model = train_model(X, y)
    except ValueError as e:
        update.message.reply_text(f"Error: {e}")
        return

    # Scrape and analyze tokens
    tokens = scrape_pumpfun_tokens()
    if tokens:
        update.message.reply_text(f"Found {len(tokens)} new tokens.")
        for token in tokens:
            contract_address = token['link'].split('/')[-1]
            dex_data = fetch_dexscreener_data(contract_address)
            rugcheck_data = analyze_token(contract_address)

            if dex_data and rugcheck_data and rugcheck_data.get('status') ==
"GOOD":
                token_data = {
                    'contract_address': contract_address,
                    'name': dex_data.get('name', 'N/A'),
                    'symbol': dex_data.get('symbol', 'N/A'),
                    'trading_volume': dex_data.get('volume24h', 0),
                    'holders': dex_data.get('holders', 0),
                    'price_change': dex_data.get('priceChange24h', 0),
                    'status': rugcheck_data.get('status', 'N/A')
                }

                # Predict success probability
                token_data['success_probability'] =
predict_success_probability(model, token_data)

                # Save to database and CSV
                save_to_database(token_data)
                save_to_csv(token_data)

                # Send result to Telegram
                update.message.reply_text(
                    f"Token Analysis:\n"
                    f"Name: {token_data['name']}\n"
                    f"Symbol: {token_data['symbol']}\n"
                    f"Contract: {token_data['contract_address']}\n"
                    f"Trading Volume: {token_data['trading_volume']}\n"
                    f"Holders: {token_data['holders']}\n"
                    f"Price Change: {token_data['price_change']}%\n"
                    f"Success Probability:
{token_data['success_probability']:.2f}\n"
                    f"Status: {token_data['status']}"
                )
    else:
        update.message.reply_text("No new tokens found.")

# Main function
def main():
    """Main function to run the trading bot."""
    setup_database()

    # Initialize Telegram bot
    updater = Updater(TELEGRAM_BOT_TOKEN)
    dispatcher = updater.dispatcher

    # Add command handler
    dispatcher.add_handler(CommandHandler("analyse", analyse_command))

    # Start the bot
    updater.start_polling()
    updater.idle()

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