Advanced Anti-Detection Web Scraper - Complete Guide

📋 Table of Contents

Overview

The Advanced Anti-Detection Web Scraper is a production-grade Python library designed to bypass most website protections while scraping. It combines multiple anti-detection techniques to mimic human browsing behavior and avoid bot detection.

Key Capabilities

- **Rotating User Agents**: 26+ realistic, up-to-date user agents (Chrome, Firefox, Safari, Edge)
- **Advanced Headers**: Browser fingerprint simulation with Sec-CH-UA headers
- **Browser Automation**: Playwright integration with stealth mode for JavaScript-heavy sites
- **Proxy Support**: Proxy rotation with health checking
- **Human-Like Behavior**: Random delays, mouse movements, scroll simulation
- **Smart Retry Logic**: Exponential backoff with automatic fallback to browser automation
- **Session Management**: Persistent cookies and session pooling

---

Features

1. Rotating User Agents

- ✅ 26 realistic user agents (updated December 2025)
- ✅ Desktop and mobile variants
- ✅ Multiple browsers: Chrome, Firefox, Safari, Edge
- ✅ Multiple operating systems: Windows, macOS, Linux, Android, iOS
- ✅ Dynamic UA generation with fake-useragent library

2. Advanced Request Headers

- ✅ Randomized Accept-Language headers
- ✅ Realistic Referer headers
- ✅ Accept-Encoding with compression support
- ✅ DNT (Do Not Track) randomization
- ✅ Sec-CH-UA headers for Chrome/Edge
- ✅ Browser fingerprint simulation

3. Browser Automation (Playwright)

- ✅ Headless and headed modes
- ✅ Stealth scripts to hide webdriver properties
- ✅ JavaScript execution support
- ✅ Dynamic content loading
- ✅ Cookie and session management
- ✅ Screenshot capabilities

4. Proxy Support

- ✅ HTTP/HTTPS/SOCKS4/SOCKS5 proxy support
- ✅ Proxy rotation (random or round-robin)
- ✅ Health checking and monitoring
- ✅ Automatic proxy failover
- ✅ Proxy statistics tracking

5. Human-Like Behavior

- ✅ Random delays (1-5 seconds, configurable)
- ✅ Exponential backoff on failures
- ✅ Mouse movement simulation (Bezier curves)
- ✅ Scroll simulation with reading pauses
- ✅ Typing speed simulation
- ✅ Click delays
- ✅ Page load waiting

6. Session Management

- ✅ Persistent cookies across requests
- ✅ Session pooling
- ✅ Cookie jar management
- ✅ Authentication token handling

7. Request Fingerprinting Avoidance

- ✅ Randomized request timing
- ✅ Varied request patterns
- ✅ Browser-specific header sets
- ✅ Platform-specific behaviors

8. Error Handling & Retry Logic

- ✅ Smart retry with exponential backoff
- ✅ Rate limiting handling (429 errors)

- ✅ 403/503 error handling
- ✅ Automatic fallback to browser automation
- ✅ Comprehensive error logging

Installation

Prerequisites

- Python 3.8+
- pip package manager

Install Dependencies

```
cd /home/ubuntu/basketball_app/python-scraper
pip install -r requirements.txt
```

Install Playwright Browsers

```
# For local installation (if you have permissions)
python -m playwright install chromium

# For user-level installation
PLAYWRIGHT_BROWSERS_PATH=$HOME/.cache/ms-playwright python -m playwright install chro-
mium
```

Quick Start

Basic Usage

```
from utils import AntiDetectionScraper

# Simple scraping
with AntiDetectionScraper(rotate_user_agents=True) as scraper:
    response = scraper.get("https://example.com")
    print(response.text)
```

With Browser Automation

```
from utils import AntiDetectionScraper

# Use browser for JavaScript-heavy sites
with AntiDetectionScraper(
    rotate_user_agents=True,
    use_browser=True,
    headless=True
) as scraper:
    response = scraper.get("https://javascript-heavy-site.com")
    print(response.text)
```

Quick Scrape Function

```python
from utils import scrape_url

# One-liner scraping
html = scrape_url("https://example.com", use_browser=False, rotate_user_agents=True)
print(html)
```

Core Components

1. UserAgentRotator

Manages a pool of realistic user agents.

```python
from utils import UserAgentRotator

rotator = UserAgentRotator()

# Get random user agent
ua = rotator.get_random_user_agent(device_type="desktop")

# Get browser-specific user agent
chrome_ua = rotator.get_chrome_user_agent(os_type="windows")
firefox_ua = rotator.get_firefox_user_agent(os_type="macos")
safari_ua = rotator.get_safari_user_agent(device="iphone")

# Get complete browser profile with matching headers
profile = rotator.get_browser_profile(browser="chrome", os_type="windows")
print(profile["User-Agent"])
print(profile["sec-ch-ua"])
```

2. ProxyManager

Handles proxy rotation and health checking.

```python
from utils import ProxyManager

# Initialize with proxy list
proxies = [
    "http://proxy1.example.com:8080",
    "http://user:pass@proxy2.example.com:3128",
    "socks5://proxy3.example.com:1080",
]

manager = ProxyManager(proxies=proxies)

# Get random proxy
proxy = manager.get_random_proxy()

# Check proxy health
manager.check_all_proxies()

# Get statistics
stats = manager.get_statistics()
print(f"Healthy proxies: {stats['healthy_proxies']}/{stats['total_proxies']}")

# Load from file
manager = ProxyManager.from_file("proxies.txt")

# Load from environment variable
manager = ProxyManager.from_env("PROXIES")  # PROXIES=http://proxy1:8080,http://
proxy2:8080
```

3. HumanBehavior

Simulates human-like browsing patterns.

```python
from utils import HumanBehavior

behavior = HumanBehavior(min_delay=1.0, max_delay=3.0, typing_speed_wpm=40)

# Random delay between requests
behavior.random_delay()

# Exponential backoff for retries
behavior.exponential_backoff(attempt=2, base_delay=1.0, max_delay=60.0)

# Typing delay
behavior.typing_delay("Hello, World!")

# Reading delay
behavior.reading_delay(word_count=100)

# Mouse movement
behavior.mouse_movement_delay(distance_pixels=500)

# Scroll simulation
behavior.scroll_delay(scroll_amount=500)

# Generate mouse path (for browser automation)
path = behavior.generate_mouse_path(start=(0, 0), end=(500, 300))
```

4. StealthBrowser (Playwright)

Browser automation with anti-detection features.

```python
from utils import StealthBrowser
import asyncio

async def scrape_with_browser():
    async with StealthBrowser(headless=True) as browser:
        # Navigate to URL
        await browser.goto("https://example.com")

        # Get content
        html = await browser.get_content()

        # Interact with page
        await browser.click("#button")
        await browser.type_text("#input", "Hello")
        await browser.scroll(500)

        # Take screenshot
        await browser.screenshot("screenshot.png")

        # Execute JavaScript
        result = await browser.evaluate("document.title")
        print(result)

# Run
asyncio.run(scrape_with_browser())
```

Synchronous Wrapper:

```python
from utils import StealthBrowserSync

with StealthBrowserSync(headless=True) as browser:
    browser.goto("https://example.com")
    html = browser.get_content()
    print(html)
```

5. AntiDetectionScraper (Main Class)

Combines all components into one powerful scraper.

```python
from utils import AntiDetectionScraper

scraper = AntiDetectionScraper(
    # User agent settings
    rotate_user_agents=True,
    use_fake_ua=True,

    # Proxy settings
    proxies=["http://proxy:8080"],
    rotate_proxies=True,
    check_proxy_health=True,

    # Human behavior
    min_delay=1.0,
    max_delay=5.0,
    enable_jitter=True,

    # Session management
    use_session=True,
    persist_cookies=True,

    # Browser automation
    use_browser=False,
    headless=True,

    # Retry logic
    max_retries=3,
    backoff_factor=2.0,
)

# GET request
response = scraper.get("https://example.com")

# POST request
response = scraper.post("https://api.example.com", json={"key": "value"})

# Get BeautifulSoup object
soup = scraper.get_soup("https://example.com")

# Download file
scraper.download_file("https://example.com/file.pdf", "output.pdf")

# Get statistics
stats = scraper.get_statistics()
print(stats)

# Close scraper
scraper.close()
```

Usage Examples

Example 1: Scrape NBA Stats

```
from utils import AntiDetectionScraper

with AntiDetectionScraper(
    rotate_user_agents=True,
    min_delay=2.0,
    max_delay=4.0,
    max_retries=4,
) as scraper:
    url = "https://stats.nba.com/stats/leagueleaders"
    params = {
        "LeagueID": "00",
        "PerMode": "PerGame",
        "Season": "2023-24",
        "StatCategory": "FG3_PCT",
    }

    headers = {
        "Accept": "application/json",
        "Origin": "https://www.nba.com",
        "Referer": "https://www.nba.com/",
    }

    response = scraper.get(url, headers=headers, params=params)
    data = response.json()
    print(f"Found {len(data['resultSet']['rowSet'])} players")
```

Example 2: Scrape Basketball-Reference with Browser Fallback

```
from utils import AntiDetectionScraper

with AntiDetectionScraper(
    rotate_user_agents=True,
    min_delay=3.0,
    max_delay=5.0,
    max_retries=4,
    use_browser=False,  # Will auto-fallback if blocked
) as scraper:
    url = "https://www.basketball-reference.com/leaders/fg3_pct_career.html"

    # This will try requests first, then fallback to browser if blocked
    soup = scraper.get_soup(url, fallback_to_browser=True)

    table = soup.find("table", {"id": "nba"})
    if table:
        rows = table.find("tbody").find_all("tr")[:10]
        for row in rows:
            cols = row.find_all(["td", "th"])
            if len(cols) >= 2:
                print(cols[1].text.strip())
```

Example 3: Use Proxies

```python
from utils import AntiDetectionScraper

proxies = [
    "http://proxy1.example.com:8080",
    "http://user:pass@proxy2.example.com:3128",
]

with AntiDetectionScraper(
    rotate_user_agents=True,
    proxies=proxies,
    rotate_proxies=True,
    check_proxy_health=True,
) as scraper:
    response = scraper.get("https://httpbin.org/ip")
    print(response.json())  # Should show proxy IP
```

Example 4: Batch Scraping with Statistics

```python
from utils import AntiDetectionScraper

urls = [
    "https://example.com/page1",
    "https://example.com/page2",
    "https://example.com/page3",
]

with AntiDetectionScraper(rotate_user_agents=True, min_delay=2.0, max_delay=4.0) as
scraper:
    for url in urls:
        try:
            soup = scraper.get_soup(url)
            print(f"✓ {url}: {len(soup.text)} chars")
        except Exception as e:
            print(f"✗ {url}: {e}")

    # Print statistics
    stats = scraper.get_statistics()
    print(f"\nTotal requests: {stats['total_requests']}")
    print(f"Success rate: {stats['success_rate']*100:.1f}%")
    print(f"Total retries: {stats['total_retries']}")
```

Configuration

Environment Variables

```
# Proxies (comma-separated)
export PROXIES="http://proxy1:8080,http://proxy2:8080,socks5://proxy3:1080"

# Custom delays
export SCRAPE_DELAY_SECONDS=3
export MAX_RETRIES=4

# Playwright browser path (if needed)
export PLAYWRIGHT_BROWSERS_PATH=$HOME/.cache/ms-playwright
```

Configuration File (config.py)

```
# Scraping Configuration
SCRAPE_DELAY_SECONDS = 3
MAX_RETRIES = 4

# User Agent Settings
ROTATE_USER_AGENTS = True
USE_FAKE_UA = True

# Proxy Settings
PROXIES = []  # List of proxy URLs
ROTATE_PROXIES = False
CHECK_PROXY_HEALTH = True

# Browser Settings
USE_BROWSER = False
HEADLESS = True

# Human Behavior
MIN_DELAY = 1.0
MAX_DELAY = 5.0
ENABLE_JITTER = True
```

Best Practices

1. Respect Website Terms of Service

- Always check robots.txt
- Follow rate limits
- Don’t overload servers

2. Use Appropriate Delays

- Start with 2-5 second delays
- Increase delays for aggressive anti-bot systems
- Use random delays, not fixed intervals

3. Rotate User Agents

- Enable UA rotation for all scraping

- Use device-appropriate UAs (desktop for desktop sites)

4. Handle Errors Gracefully

- Implement proper error handling
- Log failures for debugging
- Use exponential backoff for retries

5. Use Browser Automation Wisely

- Reserve browser mode for JavaScript-heavy sites
- Browser mode is slower but more effective
- Use headless mode for production

6. Monitor Success Rates

- Check scraper statistics regularly
- Adjust delays if success rate drops
- Rotate proxies if getting blocked

7. Avoid Detection Patterns

- Don’t scrape in regular intervals
- Vary request patterns
- Add random pauses

8. Test Incrementally

- Start with small batches
- Test with different UAs and proxies
- Monitor for blocks before scaling up

Troubleshooting

Issue: Getting 403 Forbidden Errors

Solutions:

1. Enable user agent rotation
2. Increase delays between requests
3. Use browser automation mode
4. Add proxies
5. Check if site requires specific headers

```
scraper = AntiDetectionScraper(
    rotate_user_agents=True,
    min_delay=5.0,  # Increase delay
    max_delay=10.0,
    use_browser=True,  # Use browser
)
```

Issue: Getting 429 Too Many Requests

Solutions:

1. Increase delays significantly

2. Use exponential backoff
3. Rotate proxies
4. Reduce concurrent requests

```
scraper = AntiDetectionScraper(
    min_delay=10.0,  # Much longer delay
    max_delay=20.0,
    max_retries=5,
    backoff_factor=3.0,  # Longer backoff
)
```

Issue: JavaScript Content Not Loading

Solutions:

1. Enable browser automation
2. Wait for dynamic content
3. Use appropriate selectors

```
scraper = AntiDetectionScraper(use_browser=True)
response = scraper.get(url, fallback_to_browser=True)
```

Issue: Playwright Browsers Not Found

Solutions:

1. Install browsers: `python -m playwright install chromium`
2. Set browser path: `PLAYWRIGHT_BROWSERS_PATH=$HOME/.cache/ms-playwright`
3. Check permissions

Issue: Proxy Errors

Solutions:

1. Check proxy health
2. Verify proxy format
3. Test proxies manually

```
manager = ProxyManager(proxies=["http://proxy:8080"])
manager.check_all_proxies()
stats = manager.get_statistics()
print(stats)
```

API Reference

AntiDetectionScraper

Constructor

```
AntiDetectionScraper(
    rotate_user_agents=True,
    use_fake_ua=True,
    proxies=None,
    rotate_proxies=False,
    check_proxy_health=True,
    min_delay=1.0,
    max_delay=5.0,
    enable_jitter=True,
    use_session=True,
    persist_cookies=True,
    use_browser=False,
    headless=True,
    max_retries=3,
    backoff_factor=2.0,
)
```

Methods

`get(url, headers=None, params=None, timeout=30, allow_redirects=True, fall-back_to_browser=True)`
- Perform GET request with anti-detection
- Returns: `requests.Response`

`post(url, data=None, json=None, headers=None, timeout=30)`
- Perform POST request with anti-detection
- Returns: `requests.Response`

`get_soup(url, **kwargs)`
- Get BeautifulSoup object for URL
- Returns: `BeautifulSoup`

`download_file(url, filepath, chunk_size=8192)`
- Download file with progress
- Returns: None

`get_statistics()`
- Get scraping statistics
- Returns: `Dict` with success rate, request counts, etc.

`close()`
- Close scraper and cleanup resources
- Returns: None

Performance Metrics

Typical Success Rates (based on site difficulty)

Site Type	Success Rate	Recommended Settings
Simple sites	95-100%	Default settings
Medium protection	80-95%	Rotate UAs, 3-5s delays
Strong protection	60-80%	Browser mode, proxies, 5-10s delays
Aggressive anti-bot	40-60%	Browser + proxies + long delays

Speed Comparison

Mode	Speed	Success Rate	Use Case
Direct requests	Fast (0.5-2s)	Medium	Simple sites, APIs
With delays	Medium (2-5s)	High	Most websites
Browser automation	Slow (5-15s)	Very High	JavaScript-heavy, protected sites

---

License

This scraper is part of the Basketball Shooter Analysis project.

Support

For issues or questions:
1. Check the troubleshooting section
2. Review test_anti_detection.py for examples
3. Check logs in `logs/` directory
4. Consult individual module documentation

Last Updated: December 13, 2025
Version: 1.0.0