

BEE2041 Empirical Project Blog

In [59]: `pip install selenium --upgrade`

```
Requirement already up-to-date: selenium in c:\users\socor\anaconda3\lib\site-packages (4.19.0)
Requirement already satisfied, skipping upgrade: typing_extensions>=4.9.0 in c:\users\socor\anaconda3\lib\site-packages (from selenium) (4.10.0)
Collecting certifi>=2021.10.8
  Downloading certifi-2024.2.2-py3-none-any.whl (163 kB)
Requirement already satisfied, skipping upgrade: trio-websocket~=0.9 in c:\users\socor\anaconda3\lib\site-packages (from selenium) (0.11.1)
Requirement already satisfied, skipping upgrade: trio~=0.17 in c:\users\socor\anaconda3\lib\site-packages (from selenium) (0.25.0)
Collecting urllib3[socks]<3,>=1.26
  Downloading urllib3-2.2.1-py3-none-any.whl (121 kB)
Requirement already satisfied, skipping upgrade: wsproto>=0.14 in c:\users\socor\anaconda3\lib\site-packages (from trio-websocket~=0.9->selenium) (1.2.0)
Requirement already satisfied, skipping upgrade: exceptiongroup; python_version < "3.11" in c:\users\socor\anaconda3\lib\site-packages (from trio-websocket~=0.9->selenium) (1.2.0)
Requirement already satisfied, skipping upgrade: outcome in c:\users\socor\anaconda3\lib\site-packages (from trio~=0.17->selenium) (1.3.0.post0)
Requirement already satisfied, skipping upgrade: attrs>=23.2.0 in c:\users\socor\anaconda3\lib\site-packages (from trio~=0.17->selenium) (23.2.0)
Requirement already satisfied, skipping upgrade: sortedcontainers in c:\users\socor\anaconda3\lib\site-packages (from trio~=0.17->selenium) (2.2.2)
Requirement already satisfied, skipping upgrade: sniffio>=1.3.0 in c:\users\socor\anaconda3\lib\site-packages (from trio~=0.17->selenium) (1.3.1)
Requirement already satisfied, skipping upgrade: idna in c:\users\socor\anaconda3\lib\site-packages (from trio~=0.17->selenium) (2.10)
Requirement already satisfied, skipping upgrade: cffi>=1.14; os_name == "nt" and implementation_name != "pypy" in c:\users\socor\anaconda3\lib\site-packages (from trio~=0.17->selenium) (1.14.0)
Requirement already satisfied, skipping upgrade: pysocks!=1.5.7,<2.0,>=1.5.6; extra == "socks" in c:\users\socor\anaconda3\lib\site-packages (from urllib3[socks]<3,>=1.26->selenium) (1.7.1)
Requirement already satisfied, skipping upgrade: h11<1,>=0.9.0 in c:\users\socor\anaconda3\lib\site-packages (from wsproto>=0.14->trio-websocket~=0.9->selenium) (0.14.0)
Requirement already satisfied, skipping upgrade: pycparser in c:\users\socor\anaconda3\lib\site-packages (from cffi>=1.14; os_name == "nt" and implementation_name != "pypy"->trio~=0.17->selenium) (2.20)
Installing collected packages: certifi, urllib3
  Attempting uninstall: certifi
    Found existing installation: certifi 2020.6.20
    Uninstalling certifi-2020.6.20:
      Successfully uninstalled certifi-2020.6.20
  Attempting uninstall: urllib3
    Found existing installation: urllib3 1.25.9
    Uninstalling urllib3-1.25.9:
      Successfully uninstalled urllib3-1.25.9
Successfully installed certifi-2024.2.2 urllib3-2.2.1
Note: you may need to restart the kernel to use updated packages.

ERROR: requests 2.24.0 has requirement urllib3!=1.25.0,!<1.25.1,<1.26,>=1.21.1, but you'll have urllib3 2.2.1 which is incompatible.
```

We need a list of all PCCs/force areas. let us scrape that list:

```
In [46]: ▶ url = 'https://www.police.uk/'
response = requests.get(url)
soup = BeautifulSoup(response.text, 'html.parser')
soup
```

```
Out[46]: <!DOCTYPE html>
<html lang="en-US"><head><title>Just a moment...</title><meta content="text/h
tml; charset=utf-8" http-equiv="Content-Type"/><meta content="IE=Edge" http-e
quiv="X-UA-Compatible"/><meta content="noindex,nofollow" name="robots"/><meta
content="width=device-width,initial-scale=1" name="viewport"/><style>{*{box-si
zing:border-box;margin:0;padding:0}html{line-height:1.15;-webkit-text-size-ad
just:100%;color:#313131}button,html{font-family:system-ui,-apple-system,Blink
MacSystemFont,Segoe UI,Roboto,Helvetica Neue,Arial,Noto Sans,sans-serif,Apple
Color Emoji,Segoe UI Emoji,Segoe UI Symbol,Noto Color Emoji}@media (prefers-c
olor-scheme:dark){body{background-color:#222;color:#d9d9d9}body a{color:#fff}
body a:hover{color:#ee730a;text-decoration:underline}body .lds-ring div{borde
r-color:#999 transparent transparent}body .font-red{color:#b20f03}body .big-b
utton,body .pow-button{background-color:#4693ff;color:#1d1d1d}body #challenge
-success-text{background-image:url(
aHR0cDovL3d3dy53My5vcmcvMjAwMCAwIDI2IDI2Ij48cGF0aCBmaWxsPSIjZDlkOWQ5IiBkPSJNMtMgMGExMy
AxMyAwIDEgMCAwIDI2IDEzIDEzIDAgMCAwIDA0MTJtMCAyNGExMSAwIDEgMCAwLTiYIDEeIDE
xIDAgMCAxIDAgMjIiLz48cGF0aCBmaWxsPSIjZDlkOWQ5IiBkPSJtMTAuOTU1IDE2LjA1NS0zLjk1
LTQuMTI1LTUuNDQ1IDEuMzg1IDUuMzcgNS42MSA5LjQ5NS05LjYtMSA40Mi0xLjQwNXoiLz48L3N2Z
```

```
In [58]: ❸ from selenium import webdriver
          from bs4 import BeautifulSoup

          driver = webdriver.Chrome(r'C:\Users\socor\Downloads\chromedriver-win64\chromedri
          driver.get('https://www.police.uk/')

          # Let the page load. Consider using WebDriverWait for better practice.
          import time
          time.sleep(5) # Adjust sleep time as needed.

          soup = BeautifulSoup(driver.page_source, 'html.parser')
          print(soup)

          driver.quit()
```

File "<ipython-input-58-c29cab7ae0dd>", line 1

```
pip install selenium --upgrade
```

^

SyntaxError: invalid syntax

```
pip install cloudscraper
```

```
Collecting cloudscraper
  Downloading cloudscraper-1.2.71-py2.py3-none-any.whl (99 kB)
Collecting requests-toolbelt>=0.9.1
  Downloading requests_toolbelt-1.0.0-py2.py3-none-any.whl (54 kB)
Requirement already satisfied: requests>=2.9.2 in c:\users\socor\anaconda3\lib\site-packages (from cloudscraper) (2.24.0)
Requirement already satisfied: pyparsing>=2.4.7 in c:\users\socor\anaconda3\lib\site-packages (from cloudscraper) (2.4.7)
Collecting urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1
  Downloading urllib3-1.25.11-py2.py3-none-any.whl (127 kB)
Requirement already satisfied: chardet<4,>=3.0.2 in c:\users\socor\anaconda3\lib\site-packages (from requests>=2.9.2->cloudscraper) (3.0.4)
Requirement already satisfied: certifi>=2017.4.17 in c:\users\socor\anaconda3\lib\site-packages (from requests>=2.9.2->cloudscraper) (2024.2.2)
Requirement already satisfied: idna<3,>=2.5 in c:\users\socor\anaconda3\lib\site-packages (from requests>=2.9.2->cloudscraper) (2.10)
Installing collected packages: requests-toolbelt, cloudscraper, urllib3
  Attempting uninstall: urllib3
    Found existing installation: urllib3 2.2.1
    Uninstalling urllib3-2.2.1:
      Successfully uninstalled urllib3-2.2.1
Successfully installed cloudscraper-1.2.71 requests-toolbelt-1.0.0 urllib3-1.25.11
Note: you may need to restart the kernel to use updated packages.

ERROR: selenium 4.19.0 has requirement urllib3[socks]<3,>=1.26, but you'll have urllib3 1.25.11 which is incompatible.
```

```
import cloudscraper
url = 'https://www.police.uk/'
scraper = cloudscraper.create_scraper()
res = scraper.get(url)
print(res.status_code)
res.text
```

403

```
'<!DOCTYPE html><html lang="en-US"><head><title>Just a moment...</title><meta http-equiv="Content-Type" content="text/html; charset=UTF-8"><meta http-equiv="X-UA-Compatible" content="IE=Edge"><meta name="robots" content="noindex,nofollow"><meta name="viewport" content="width=device-width,initial-scale=1"><style>*{box-sizing:border-box;margin:0;padding:0}html{line-height:1.15;-webkit-text-size-adjust:100%;color:#313131}button,html{font-family:system-ui,-apple-system,BlinkMacSystemFont,Segoe UI,Roboto,Helvetica Neue,Arial,Noto Sans,sans-serif,Apple Color Emoji,Segoe UI Emoji,Segoe UI Symbol,Noto Color Emoji}@media (prefers-color-scheme:dark){body{background-color:#222;color:#d9d9d9}body a{color:#fff}body a:hover{color:#ee730a;text-decoration:underline}body .lds-ring div{border-color:#999 transparent transparent}body .font-red{color:#b20f03}body .big-button,body .pow-button{background-color:#4693ff;color:#1d1d1d}body #challenge-success-text{background-image:url(
```

```
In [66]: ▶ import cfscrape
url = 'https://www.police.uk/'
scrape = cfscrape.create_scraper()
res = scrape.get(url)
print(res.status_code)
```

403

```
In [64]: ▶ pip install cfscrape
```

Collecting cfscrape

Downloading cfscrape-2.1.1-py3-none-any.whl (12 kB)

Requirement already satisfied: requests>=2.6.1 in c:\users\socor\anaconda3\lib\site-packages (from cfscrape) (2.24.0)

Requirement already satisfied: certifi>=2017.4.17 in c:\users\socor\anaconda3\lib\site-packages (from requests>=2.6.1->cfscrape) (2024.2.2)

Requirement already satisfied: idna<3,>=2.5 in c:\users\socor\anaconda3\lib\site-packages (from requests>=2.6.1->cfscrape) (2.10)

Requirement already satisfied: chardet<4,>=3.0.2 in c:\users\socor\anaconda3\lib\site-packages (from requests>=2.6.1->cfscrape) (3.0.4)

Requirement already satisfied: urllib3!=1.25.0,!1.25.1,<1.26,>=1.21.1 in c:\users\socor\anaconda3\lib\site-packages (from requests>=2.6.1->cfscrape) (1.25.11)

Installing collected packages: cfscrape

Successfully installed cfscrape-2.1.1

Note: you may need to restart the kernel to use updated packages.

```
In [9]: ▶ from selenium import webdriver
from selenium.webdriver.chrome.service import Service
from selenium.webdriver.common.keys import Keys
import time

# Specify the path to chromedriver if it's not in your PATH
chromedriver_path = r"C:\Users\socor\Downloads\chromedriver-win64\chromedriver-wi

# Initialize the WebDriver (assuming Chrome)
service = Service(executable_path=chromedriver_path)
driver = webdriver.Chrome(service=service)

try:
    # Navigate to a website
    driver.get("http://police.uk")

    # Wait for 5 seconds to see the page
    time.sleep(5)

    # Optionally, interact with the website
    # For example, search for 'Selenium' in Wikipedia
    # search_box = driver.find_element_by_name('q')
    # search_box.send_keys('Selenium')
    # search_box.send_keys(Keys.RETURN)
    # time.sleep(5)

    print("Selenium is working fine!")
except Exception as e:
    print(f"An error occurred: {e}")
finally:
    # Close the browser
    driver.quit()
```

Selenium is working fine!

Having established that Selenium is capable of accessing the police.uk website, let's start building an ethical bot! Firstly, we accessed the <https://police.uk/robots.txt> (<https://police.uk/robots.txt>) page and found certain URLs needed to be disallowed. I decided to start by caching the robots.txt file so that my bot could refer to it without sending repeated requests to the site. My bot would then check URLs against those contained in the robot.txt file and would return a "robot.txt error" rather than crawl the forbidden URL:

```
In [19]: ▶ from urllib.robotparser import RobotFileParser
from urllib.parse import urlparse

def can_crawl(url):
    """
    Check if the crawler can crawl a given URL based on the site's robots.txt.
    """
    parsed_url = urlparse(url)
    robots_url = f"{parsed_url.scheme}://{parsed_url.netloc}/robots.txt"

    rp = RobotFileParser()
    rp.set_url(robots_url)
    rp.read()

    return rp.can_fetch("FriendlyUniStudentResearcher", url)

def crawl(url):
    """
    Attempt to crawl a URL, respecting robots.txt rules.
    """
    if can_crawl(url):
        try:
            response = requests.get(url)
            # Process the response here (e.g., parse HTML, follow links, etc.)
            print(f"Successfully crawled: {url}")
        except Exception as e:
            print(f"An error occurred while crawling {url}: {e}")
    else:
        print(f"robots.txt error: Crawling not allowed for {url}")

# Example usage
urls_to_crawl = [
    "http://police.uk/mediacentre",
    "http://police.uk/?u=media",
    "http://police.uk"
    # Add other URLs you're interested in
]

for url in urls_to_crawl:
    crawl(url)
```

```
robots.txt error: Crawling not allowed for http://police.uk/mediacentre (http://
police.uk/mediacentre)
robots.txt error: Crawling not allowed for http://police.uk/?u=media (http://pol
ice.uk/?u=media)
robots.txt error: Crawling not allowed for http://police.uk (http://police.uk)
```

It is customary to include a specific "user-agent" to identify your bot and make it possible for website administrators to contact you with concerns:

```
In [ ]: ▶ session = requests.Session()

# Set the custom user-agent for all requests made with this session
session.headers.update({
    'User-Agent': "FriendlyUniStudentResearcher/1.0 (+mailto:soc204@exeter.ac.uk)"
})
response = session.get()
```

```
In [7]: ▶ import requests
import json

# Define your custom user-agent string
user_agent = "FriendlyUniStudentResearcher/1.0 (+mailto:soc204@exeter.ac.uk)"

# Set the headers for your request to include your custom user-agent
headers = {
    'User-Agent': user_agent
}

# The URL for testing headers (httpbin.org is useful for HTTP requests testing)
test_url = "https://httpbin.org/headers"

# Make the request with your headers
response = requests.get(test_url, headers=headers)

# Parse the JSON response
response_json = response.json()

# Extract and print the User-Agent header from the response
print("User-Agent received by httpbin.org:")
print(response_json['headers']['User-Agent'])
```

User-Agent received by httpbin.org:
FriendlyUniStudentResearcher/1.0 (+mailto:soc204@exeter.ac.uk)

```
In [ ]: ▶ from selenium import webdriver
from selenium.webdriver.chrome.service import Service
from selenium.webdriver.common.keys import Keys
import time
from selenium.webdriver.chrome.options import Options

options = Options()
user_agent = "FriendlyUniStudentResearcher/1.0 (+mailto:soc204@exeter.ac.uk)"
options.add_argument(f'user-agent={user_agent}')
# Specify the path to chromedriver if it's not in your PATH
chromedriver_path = r"C:\Users\socor\Downloads\chromedriver-win64\chromedriver-wi

# Initialize the WebDriver (assuming Chrome)
service = Service(executable_path=chromedriver_path)
driver = webdriver.Chrome(service=service)

session = requests.Session()

# Set the custom user-agent for all requests made with this session
session.headers.update({
    'User-Agent': "FriendlyUniStudentResearcher/1.0 (+mailto:soc204@exeter.ac.uk)"
})
response = session.get()

driver.quit()
```

```
In [22]: ▶ url = 'https://www.police.uk/'
parsed_url = urlparse(url)
robots_url = f'{parsed_url.scheme}://{parsed_url.netloc}/robots.txt'
rp = RobotFileParser()
rp.set_url(robots_url)
rp.read()
for line in rp.default_entry.rulelines:
    print(f"Allow: {line.allowance} Path: {line.path}")

# Check if the root URL is allowed
print(rp.can_fetch("*", "https://www.police.uk/"))
rp.can_fetch("*", url)
```

```
-----
AttributeError                                Traceback (most recent call last)
<ipython-input-22-dd4376a6c90c> in <module>
      5 rp.set_url(robots_url)
      6 rp.read()
----> 7 for line in rp.default_entry.rulelines:
      8     print(f"Allow: {line.allowance} Path: {line.path}")
      9

AttributeError: 'NoneType' object has no attribute 'rulelines'
```

```
In [18]: ▶ from urllib.parse import urlparse
from urllib.robotparser import RobotFileParser

# Initialize a cache dictionary
robots_cache = {}

def cache_robots_data(url):
    """
    Fetches and caches the robots.txt data for the given URL's domain.
    """
    # Parse the domain from the given URL
    parsed_url = urlparse(url)
    base_url = f"{parsed_url.scheme}://{parsed_url.netloc}"
    robots_url = f"{base_url}/robots.txt"

    # Check if we already have cached data for this domain
    if base_url in robots_cache:
        print("Using cached robots.txt data.")
        return robots_cache[base_url]
    else:
        print("Fetching new robots.txt data.")
        # Initialize a RobotFileParser instance
        rp = RobotFileParser()
        rp.set_url(robots_url)
        rp.read() # Fetch and parse the robots.txt

        # Cache the RobotFileParser instance for future use
        robots_cache[base_url] = rp

    return rp

# Example usage
rp = cache_robots_data('https://www.police.uk')
rp
```

Fetching new robots.txt data.

Out[18]: <urllib.robotparser.RobotFileParser at 0x22aa7399bb0>

Data Collection

```
In [7]: ▶ from selenium.webdriver.chrome.options import Options
def establish_user_agent(user_agent, chromedriver_path):
    chrome_options = Options()
    chrome_options.add_argument(f"user-agent={user_agent}")
    return chrome_options
```

```
In [8]: ▶ from selenium import webdriver
from selenium.webdriver.chrome.service import Service
def init_chrome_webdriver(chromedriver_path, chrome_options):
    chrome_options.add_argument("--no-sandbox") # This parameter helps in avoidi
    chrome_options.add_argument("--disable-gpu") # Disables GPU hardware acceler
    chrome_options.add_argument("--log-level=3") # This will only show fatal err
    service = Service(executable_path=chromedriver_path)
    driver = webdriver.Chrome(service=service, options=chrome_options)
    return driver
```



```
In [9]: > import time
import json
from selenium.webdriver.common.by import By
def test_user_agent(driver, user_agent):
    driver.get("https://httpbin.org/user-agent")
    time.sleep(5)
    response_data = json.loads(driver.find_element(By.TAG_NAME, "body").text)
    echoed_user_agent = response_data["user-agent"]

    if echoed_user_agent != user_agent:
        print("User-Agent does not match the expected value. Quitting...")
        raise Exception("User-Agent does not match the expected value.")
```

```
In [10]: > def is_target_disallowed(target, disallowed_dict):
        """
        Check if the target path matches any of the disallowed paths.

        :param target_path: The target path to check
        :param disallowed_paths: A dictionary of disallowed paths from robots.txt
        :return: True if the target path is disallowed, False otherwise
        """

        # Extract base URL from the target
        parsed_url = urlparse(target)
        base_url = f"{parsed_url.scheme}://{parsed_url.netloc}"

        # Retrieve the list of disallowed patterns for the base URL
        disallowed_patterns = disallowed_dict.get(base_url, [])

        # Normalize target path
        target_pattern = f'{parsed_url.path}?{parsed_url.query}'.rstrip("?")
        target_path = target_pattern.rstrip("/")

        for pattern in disallowed_patterns:
            # Normalize disallowed path
            pattern = pattern.rstrip("/")

            # Check if the target pattern starts with the disallowed pattern
            if target_path.startswith(pattern):
                return True

            # Checking for file extension disallowance, e.g., '*.aspx$'
            if pattern.endswith('$'):
                base_pattern = pattern[1:-1]
                if target_path.endswith(base_pattern):
                    return True

        return False
```

```

In [11]: ▶ from urllib.parse import urlparse
import re
def establish_bot_permissions(driver, target, existing_disallowed=None):
    parsed_url = urlparse(target)
    base_url = f"{parsed_url.scheme}://{parsed_url.netloc}"

    # Initialize the dictionary if not provided
    if existing_disallowed is None:
        existing_disallowed = {}

    # If the base URL is already in the dictionary, return it
    if base_url in existing_disallowed:
        if is_target_disallowed(target, existing_disallowed):
            print('This URL is not allowed to be crawled in line with robots.txt')
            raise Exception(f"Target path {target} is disallowed.")
        else:
            print(f"{target} is not disallowed")
        return existing_disallowed

    # Navigate to relevant robots.txt file
    robots_url = f"{base_url}/robots.txt"
    driver.get(robots_url)
    time.sleep(1)

    # Scrape disallowed patterns
    robots_txt_content = driver.find_element(By.TAG_NAME, "body").text
    disallow_pattern = r"Disallow: ([^\n]+)"
    disallowed_paths = re.findall(disallow_pattern, robots_txt_content)
    existing_disallowed[base_url] = disallowed_paths

    if is_target_disallowed(target, existing_disallowed):
        print('This URL is not allowed to be crawled in line with robots.txt')
        raise Exception(f"Target path {target} is disallowed.")
    else:
        print(f"{target} is not disallowed")
    return existing_disallowed

```

```
In [16]: ► from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC

def get_force_areas(driver, target):
    try:
        driver.get(target)
        all_buttons = WebDriverWait(driver, 10).until(
            EC.presence_of_all_elements_located((By.CSS_SELECTOR, ".js-crime-stat
        )

        if len(all_buttons) > 1:
            toggle_button = all_buttons[1] # Select the second button
            driver.execute_script("arguments[0].scrollIntoView(true);", toggle_bu
            toggle_button.click()
            time.sleep(2)
        else:
            print("Not enough buttons found.")

        tables = driver.find_elements(By.TAG_NAME, 'table')
        table = tables[-1]
        driver.execute_script("arguments[0].scrollIntoView(true);", table)
        rows = table.find_elements(By.TAG_NAME, 'tr')
        force_areas = []

        for row in rows:
            cells = row.find_elements(By.TAG_NAME, 'td')
            if cells:
                text = cells[0].text.strip()
                force_areas.append(text)
    except Exception as e:
        print(f"An error occurred while processing: {e}")
    return force_areas
```

```
In [13]: ▶ from selenium.webdriver.common.keys import Keys
def navigate_to_force_area_performance(driver, area, disallowed_patterns, force_a
try:
    all_search_inputs = WebDriverWait(driver, 10).until(
        EC.visibility_of_all_elements_located((By.CSS_SELECTOR, "input[type='
    )

    # Make sure there are at least two search bars
    if len(all_search_inputs) >= 2:
        search_input = all_search_inputs[1] # Select the second search input
    else:
        raise Exception("Less than two search inputs found on the page.")

    search_input.click()
    # Clear the search field first in case there's any pre-filled text
    search_input.clear()
    # Enter the area name into the search field
    search_input.send_keys(area)
    # Search!
    search_input.send_keys(Keys.ENTER) # Press Enter directly via Selenium

    time.sleep(1)
    #Check if this new page is disallowed
    target = driver.current_url
    disallowed_patterns = establish_bot_permissions(driver, target, disallowed_

    driver.get(target)
    time.sleep(1)
    print(f"Navigation to the {area} performance page is successful.")

except Exception as e:
    print(f"An error occurred while navigating to the {area} performance page
return force_area_urls
```

```

In [14]: ► def get_jurisdictions(driver, area, disallowed_patterns, force_area_jurisdictions):
link = driver.find_elements(By.XPATH, "//a[.//h3[contains(@class, 'c-link-pan
if len(link)<1:
    print("No data available")
    jurisdictions[area]={}
    return jurisdictions
link = WebDriverWait(driver, 10).until(
    EC.visibility_of_element_located((By.XPATH, "//a[.//h3[contains(@clas
    )
target = link.get_attribute('href')
disallowed_patterns = establish_bot_permissions(driver, target, disallowed_patt
driver.get(target)
time.sleep(1)
all_buttons = driver.find_elements(By.CSS_SELECTOR, ".js-crime-stats-table-to
if len(all_buttons) > 1:
    toggle_button = all_buttons[1]
    driver.execute_script("arguments[0].scrollIntoView(true);", toggle_button
    toggle_button.click()
    time.sleep(1)
else:
    print("Not enough buttons found.")
    jurisdictions[area]={}
    return jurisdictions

tables = driver.find_elements(By.TAG_NAME, 'table')
table = tables[2]
driver.execute_script("arguments[0].scrollIntoView(true);", table)
rows = table.find_elements(By.TAG_NAME, 'tr')
force_area_jurisdictions = {}

for row in rows:
    cells = row.find_elements(By.TAG_NAME, 'td')
    if cells:
        text = cells[0].text.strip()
        force_area_jurisdictions[text]=cells[1].text.strip()
jurisdictions[area] = force_area_jurisdictions
return jurisdictions

```

```

In [15]: ▶ def get_force_area_finances(driver, area, disallowed_patterns, financial_reserves=
navigate_to_force_area_performance(driver, area, disallowed_patterns)
link = driver.find_elements(By.XPATH, "//a[./h3[contains(@class, 'c-link-pan
if len(link)<1:
    print("No data available")
    financial_reserves[area]={}
    return financial_reserves
link = WebDriverWait(driver, 10).until(
    EC.visibility_of_element_located((By.XPATH, "//a[./h3[contains(@clas
    )
target = link.get_attribute('href')
disallowed_patterns = establish_bot_permissions(driver, target, disallowed_patt
driver.get(target)
time.sleep(1)
all_buttons = driver.find_elements(By.CSS_SELECTOR, ".js-crime-stats-table-to
if len(all_buttons) > 1:
    toggle_button = all_buttons[0]
    driver.execute_script("arguments[0].scrollIntoView(true);", toggle_button
    toggle_button.click()
    time.sleep(1)
else:
    print("Not enough buttons found.")
    return financial_reserves

tables = driver.find_elements(By.TAG_NAME, 'table')
table = tables[-2]
driver.execute_script("arguments[0].scrollIntoView(true);", table)
rows = table.find_elements(By.TAG_NAME, 'tr')

for row in rows:
    cells = row.find_elements(By.TAG_NAME, 'td')
    if cells:
        year = cells[0].text.strip()
        financial_reserves[area][year]['General fund']=cells[1].text.strip()
        financial_reserves[area][year]['Earmarked reserves']=cells[2].text.st
        financial_reserves[area][year]['Total resource reserves']=cells[3].te
        financial_reserves[area][year]['Capital reserves']=cells[4].text.stri
    return financial_reserves
# {force_area:{Mar 2018: {General Fund: 10000, Earmarked Reserves: 10000, Total R
# {force_area_keys:{Year_keys:{Fund_type_keys:Values}}}

```

What follows is the webscraping script- remember to recreate this script's output, you must have first downloaded the relevant chromedriver for your machine from <https://googlechromelabs.github.io/chrome-for-testing/#stable> (<https://googlechromelabs.github.io/chrome-for-testing/#stable>), and provide the path to your own version of the chromedriver where prompted in the script. You may also wish to use your own user-agent. It is recommended that your user-agent contains a (+mailto:emailaddress) string so that any crawling of the bot that raises concerns with the service provider can be mediated by them reaching out to you.

```

In [17]: ▶ # Setup User-Agent
user_agent = "FriendlyUniStudentResearcher/1.0 (+mailto:soc204@exeter.ac.uk)"

#Provide the path to your own version of the chromedriver
chromedriver_path = r"C:\Users\socor\Downloads\chromedriver-win64\chromedriver-wi

chrome_options = establish_user_agent(user_agent, chromedriver_path)

# Initialize the WebDriver (assuming Chrome)
driver = init_chrome_webdriver(chromedriver_path,chrome_options)

# Set target URL
target = 'https://www.police.uk/pu/your-area/avon-somerset-constabulary/performan

try:
    # Navigate to a website that echoes back the user-agent
    test_user_agent(driver, user_agent)

    # Navigate to target website robots.txt and save the disallowed patterns
    disallowed_patterns = establish_bot_permissions(driver, target)

    # Collect the names of Force areas for which data is available
    Force_Areas = get_force_areas(driver, target)

    target = 'https://www.police.uk/pu/performance/'
    disallowed_patterns = establish_bot_permissions(driver, target, disallowed_pa
    driver.get(target)
    force_area_urls = {}
    jurisdictions = {}
    financial_reserves = {}
    Periods = ('Mar 2011', 'Mar 2012', 'Mar 2013', 'Mar 2014', 'Mar 2015', 'Mar 2
    Reserves = ('General fund', 'Earmarked reserves', 'Total resource reserves', '
    for area in Force_Areas:
        period_dict={}
        for period in Periods:
            reserves_dict={}
            for reserve_type in Reserves:
                reserves_dict[reserve_type] = None
                period_dict[period] = reserves_dict
                financial_reserves[area] = period_dict
    # Target each force area's performance data
    for area in Force_Areas[:-1]:
        force_area_urls = navigate_to_force_area_performance(driver, area, disall
        jurisdictions = get_jurisdictions(driver, area, disallowed_patterns)
        #get force area's historical financial reserves
        financial_reserves = get_force_area_finances(driver, area, disallowed_pat

        driver.get('https://www.police.uk/pu/performance/')
        time.sleep(2)

    time.sleep(10)

except Exception as e:
    print(f"An error occurred: {e}")
finally:
    # Close the browser
    driver.quit()

```

<https://www.police.uk/pu/your-area/avon-somerset-constabulary/performance/financial-reserves/> (<https://www.police.uk/pu/your-area/avon-somerset-constabulary/performance/financial-reserves/>) is not disallowed

<https://www.police.uk/pu/performance/> (<https://www.police.uk/pu/performance/>) is not disallowed

<https://www.police.uk/pu/your-area/avon-somerset-constabulary/performance/performance-avon-somerset/?tc=AN004> (<https://www.police.uk/pu/your-area/avon-somerset-constabulary/performance/performance-avon-somerset/?tc=AN004>) is not disallowed

Navigation to the Avon and Somerset Constabulary performance page is successful.

<https://www.police.uk/pu/your-area/avon-somerset-constabulary/performance/compare-your-area/?tc=AN004> (<https://www.police.uk/pu/your-area/avon-somerset-constabulary/performance/compare-your-area/?tc=AN004>) is not disallowed

<https://www.police.uk/pu/your-area/avon-somerset-constabulary/performance/performance-avon-somerset/?tc=AN004> (<https://www.police.uk/pu/your-area/avon-somerset-constabulary/performance/performance-avon-somerset/?tc=AN004>) is not disallowed

Navigation to the Avon and Somerset Constabulary performance page is successful.

In [18]: `print(financial_reserves)`

```
{'Avon and Somerset Constabulary': {'Mar 2011': {'General fund': '£6.7m', 'Earmarked reserves': '£25.6m', 'Total resource reserves': '£32.2m', 'Capital reserves': '£2.1m'}, 'Mar 2012': {'General fund': '£7.5m', 'Earmarked reserves': '£29.6m', 'Total resource reserves': '£37.1m', 'Capital reserves': '£2.4m'}, 'Mar 2013': {'General fund': '£13.6m', 'Earmarked reserves': '£28.7m', 'Total resource reserves': '£42.2m', 'Capital reserves': '£1.9m'}, 'Mar 2014': {'General fund': '£14.4m', 'Earmarked reserves': '£31.3m', 'Total resource reserves': '£45.7m', 'Capital reserves': '£4.1m'}, 'Mar 2015': {'General fund': '£14.4m', 'Earmarked reserves': '£28.8m', 'Total resource reserves': '£43.2m', 'Capital reserves': '£17.0m'}, 'Mar 2016': {'General fund': '£10.4m', 'Earmarked reserves': '£29.3m', 'Total resource reserves': '£39.7m', 'Capital reserves': '£11.6m'}, 'Mar 2017': {'General fund': '£10.4m', 'Earmarked reserves': '£26.7m', 'Total resource reserves': '£37.1m', 'Capital reserves': '£7.1m'}, 'Mar 2018': {'General fund': '£10.4m', 'Earmarked reserves': '£25.3m', 'Total resource reserves': '£35.7m', 'Capital reserves': '£0.3m'}}, 'Bedfordshire Police': {'Mar 2011': {'General fund': '£2.9m', 'Earmarked reserves': '£6.5m', 'Total resource reserves': '£9.4m', 'Capital reserves': '£0.4m'}, 'Mar 2012': {'General fund': '£2.9m', 'Earmarked reserves': '£6.6m', 'Total resource reserves': '£9.5m', 'Capital reserves': '£0.4m'}, 'Mar 2013': {'General fund': '£2.9m', 'Earmarked reserves': '£6.6m', 'Total resource reserves': '£9.5m', 'Capital reserves': '£0.4m'}}
```

Data Cleaning

The webpage states that data is not available for "City of London Police" force area, so we'll add that force area manually.

In [20]: `Force_Areas.append("City of London Police")`

We need to store that data in a pandas series to unlock better functionality:


```
In [21]: ► import pandas as pd
Force_Area = pd.Series(Force_Areas)
Force_Area
```

```
Out[21]: 0      Avon and Somerset Constabulary
1           Bedfordshire Police
2      Cambridgeshire Constabulary
3           Cheshire Constabulary
4           Cleveland Police
5           Cumbria Constabulary
6      Derbyshire Constabulary
7      Devon & Cornwall Police
8           Dorset Police
9           Durham Constabulary
10          Dyfed-Powys Police
11           Essex Police
12      Gloucestershire Constabulary
13          Greater Manchester Police
14           Gwent Police
15           Hampshire Constabulary
16      Hertfordshire Constabulary
17           Humberside Police
18           Kent Police
19      Lancashire Constabulary
20           Leicestershire Police
21           Lincolnshire Police
22           Merseyside Police
23                  MOPAC
24      Norfolk Constabulary
25      North Wales Police
26      North Yorkshire Police
27      Northamptonshire Police
28      Northumbria Police
29      Nottinghamshire Police
30      South Wales Police
31      South Yorkshire Police
32      Staffordshire Police
33      Suffolk Constabulary
34           Surrey Police
35           Sussex Police
36      Thames Valley Police
37      Warwickshire Police
38      West Mercia Police
39      West Midlands Police
40      West Yorkshire Police
41           Wiltshire Police
42      Total England & Wales
43      City of London Police
dtype: object
```

```
In [ ]: ►
```

<https://github.com/SOCStudentUoE/BEE2041-Empirical-Assignment>
(<https://github.com/SOCStudentUoE/BEE2041-Empirical-Assignment>)

