data_collection_code

February 23, 2024

1 Data Collection Problem Set

1.1 Data Loading

[ ]:
```
import pandas as pd
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.common.exceptions import NoSuchElementException,␣
↪StaleElementReferenceException

# Load the domain list
df = pd.read_csv('domainsToInspect.csv')
domains = df['domain'].tolist()

# Load the ad block list
ad_domains = []
with open('Adblocklist.txt', 'r') as file:
    while True:
        # read the second line of each two line for redundency
        file.readline()
        ad_domain = file.readline()
        if not ad_domain:
            break
        ad_domains.append(ad_domain.strip())
```

1.2 Webdriver Initailization

[ ]:
```
from selenium import webdriver

# Create Chromeoptions instance
options = webdriver.ChromeOptions()

# set header and headless mode
options.add_argument('user-agent=Mozilla/5.0 (Macintosh; Intel Mac OS X␣
↪10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/121.0.0.0 Safari/537.
↪36')
options.add_argument('--headless')
```

```
# Adding argument to disable the AutomationControlled flag
options.add_argument("--disable-blink-features=AutomationControlled")

# Exclude the collection of enable-automation switches
options.add_experimental_option("excludeSwitches", ["enable-automation"])

# Turn-off userAutomationExtension
options.add_experimental_option("useAutomationExtension", False)

# Setting the driver path and requesting a page
driver = webdriver.Chrome(options=options)

# Changing the property of the navigator value for webdriver to undefined
driver.execute_script("Object.defineProperty(navigator, 'webdriver', {get: ()␣
↪=> undefined})")
```

1.3 Iframe Detection

[ ]:
```
from selenium.webdriver.support.ui import WebDriverWait
from selenium.common.exceptions import TimeoutException
from selenium.webdriver.support import expected_conditions as EC
import time

# store iframe information to dynamc_conntent
def extract_nested_iframes(driver, dynamic_content, iframe_path='',␣
↪verbose=False):
    try:
        # Wait for iframes to be present
        WebDriverWait(driver, 2).until(
            EC.presence_of_all_elements_located((By.TAG_NAME, 'iframe'))
        )
    except TimeoutException:
        if verbose:
            print("Timeout waiting for iframes to load")

    # Find all iframes at the current level
    iframes = driver.find_elements(By.TAG_NAME, 'iframe')

    for index, iframe in enumerate(iframes):
        # Construct a unique key for the iframe based on its path
        iframe_key = f'{iframe_path}iframe_{index}'

        try:
            src = iframe.get_attribute('src')
            dynamic_content[iframe_key] = src
            driver.switch_to.frame(iframe)

            # Recursively extract nested iframes
            extract_nested_iframes(driver, dynamic_content,␣
↪iframe_path=f'{iframe_key}>')

            # Switch back to the parent frame
            driver.switch_to.parent_frame()
        except Exception as e:
            if verbose:
                print(f'Error processing {iframe_key}: {e}')

            # Ensure the driver is switched back to the parent frame in case of␣
↪an error
            driver.switch_to.parent_frame()

# return the dynamic content of a url
def dynamic_content_extractor_nested(url):
    driver.get('http://'+url)
    time.sleep(5)
    dynamic_content = {}
    extract_nested_iframes(driver, dynamic_content=dynamic_content)
    return dynamic_content
```

[ ]:
```
import re

# check whether the url is in the ad block lst
def ad_url_check(dynamic_url):
    for ad_domain in ad_domains:
        regex_pattern = '.*' + ad_domain.replace('*', '.*').replace('.', r'\.')␣
↪+ '.*'
        if re.match(regex_pattern, dynamic_url):
            return True, ad_domain
    return False, None
```

[ ]:
```
from collections import Counter

# return nested counter of ad servers
def ad_url_nested_counter(url):
    dynamic_content = dynamic_content_extractor_nested(url)
    nested_counter = [Counter()]
    for k in dynamic_content:
        layer_idx = len(k.split('>'))
        if layer_idx > len(nested_counter):
            nested_counter.append(Counter())
        is_ad, ad_admin = ad_url_check(dynamic_content[k])
        if is_ad:
            nested_counter[layer_idx-1][ad_admin] += 1
    return nested_counter

# helper function for ad_server_nested_analysis
def ad_server_analysis(counter):
    uniq_ad_servers = len(counter)
    ad_servers = sum([v for k, v in counter.items()])
    return uniq_ad_servers, ad_servers

# return number of unique ad servers and total ad servers
def ad_server_nested_analysis(nested_counter):
    for i, counter in enumerate(nested_counter):
        uniq_ad_servers, ad_servers = ad_server_analysis(counter)
        if uniq_ad_servers:
            break
    return uniq_ad_servers, ad_servers
```

[ ]:
```
PATIENCE = 5

results = {}
failed_domains = []

# Traverse each url/domain in the list
for domain in domains:
    num_tries = 0
    while num_tries < PATIENCE:
        num_tries += 1
        try:
            nested_counter = ad_url_nested_counter(domain)
            uniq_ad_servers, ad_servers =␣
 ↪ad_server_nested_analysis(nested_counter)
            break
        except Exception as e:
            print(f'Try {num_tries} for {domain}: error {e} found when␣
 ↪browsing')
            uniq_ad_servers, ad_servers = 0, 0

    # Append the result to the results list
    results[domain] = (uniq_ad_servers, ad_servers)

    # if a domain is not loaded, store in list for further notice
    if num_tries == PATIENCE:
        failed_domains.append(domain)

# announce the websites where browsing is failed
if failed_domains:
    print(f'Check those websites again: {failed_domains}')

# Write the results to a tab-separated file
with open('ad_analysis.tsv', 'w') as f:
    for domain, ads_number in results.items():
        uniq_ads, total_ads = ads_number
        f.write(f'{domain}\t{uniq_ads}\t{total_ads}\n')

print(f'Analysis complete. The tab-separated file is saved.')
```

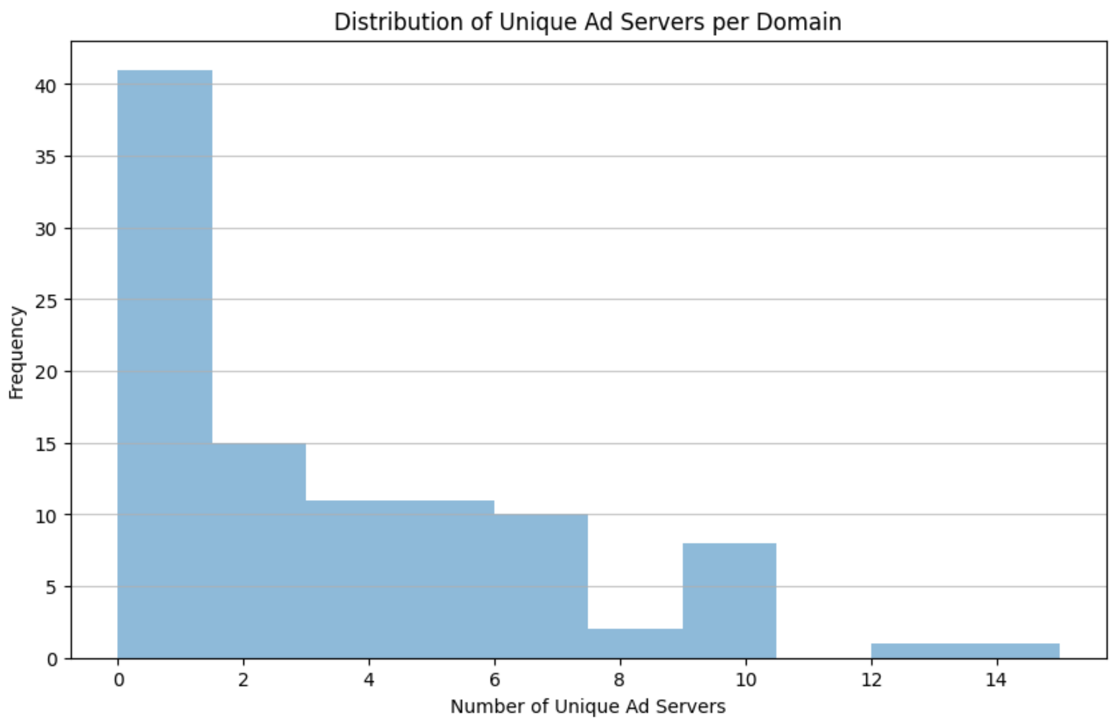
```
Analysis complete. The tab-separated file is saved.
```

1.4 Analysis (Histogram)

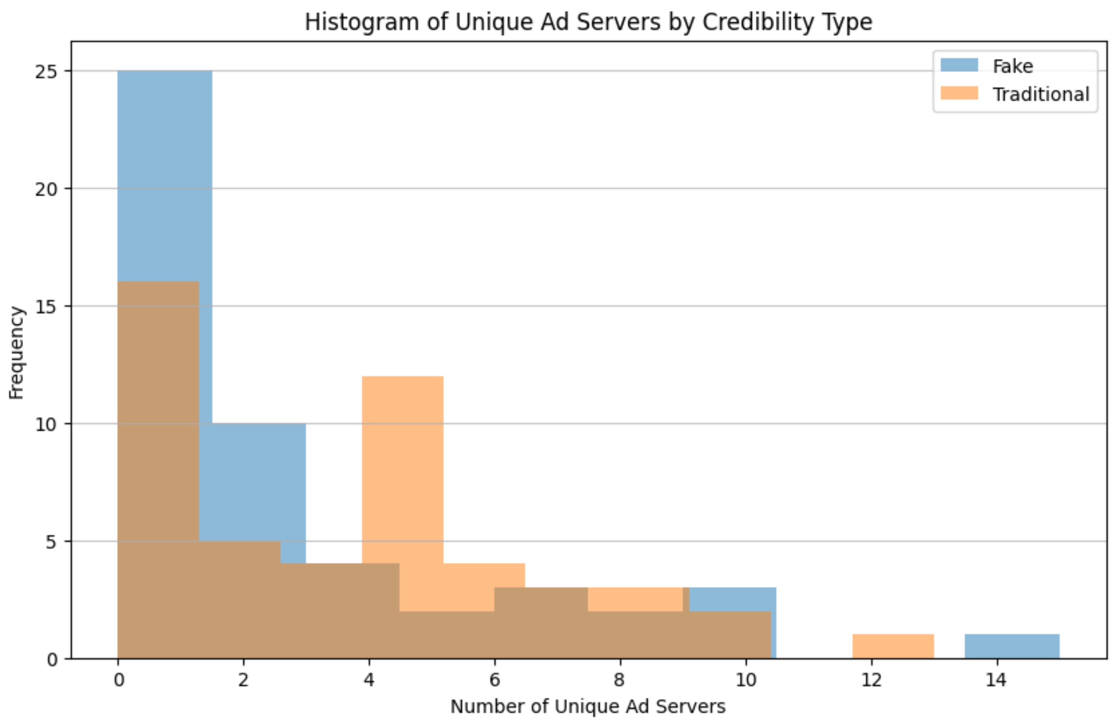
[ ]:
```
ads_data_df = pd.read_csv('ad_analysis.tsv', sep='\t', header=None)
ads_data_df.columns = ['domain', 'unique_ad_servers', 'total_ads']

merged_df = df.merge(ads_data_df, on='domain')
merged_df.head()
```

[ ]:
```
           domain   bias_rating        ave_m    cred_type  unique_ad_servers  \
0         msn.com   left-center  743000000.0  traditional                  1
1         cnn.com          left  535000000.0  traditional                  6
2  dailymail.co.uk         right  484020000.0         fake                 15
3      foxnews.com         right  391666666.0  traditional                  9
4     newsweek.com  right-center  344790000.0  traditional                  5

   total_ads
0          1
1          7
2         28
3          9
4          5
```

[ ]:
```
import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 6))
plt.hist(merged_df['unique_ad_servers'], alpha=0.5)
plt.title('Distribution of Unique Ad Servers per Domain')
plt.xlabel('Number of Unique Ad Servers')
plt.ylabel('Frequency')
plt.grid(axis='y', alpha=0.75)
plt.show()
```



```
[ ]: import numpy as np

     fake_df = merged_df[merged_df['cred_type'] == 'fake']
     traditional_df = merged_df[merged_df['cred_type'] == 'traditional']

     plt.figure(figsize=(10, 6))

     # Plot histogram for 'fake'
     plt.hist(fake_df['unique_ad_servers'], alpha=0.5, label='Fake')

     # Plot histogram for 'traditional'
     plt.hist(traditional_df['unique_ad_servers'], alpha=0.5, label='Traditional')

     plt.xlabel('Number of Unique Ad Servers')
     plt.ylabel('Frequency')
     plt.title('Histogram of Unique Ad Servers by Credibility Type')
     plt.legend(loc='upper right')
     plt.grid(axis='y', alpha=0.75)

     plt.show()
```


Histogram of Unique Ad Servers by Credibility Type
Fake
Traditional
Frequency
Number of Unique Ad Servers
0
5
10
15
20
25
0
2
4
6
8
10
12
14