# POLITENICO DI TORINO

Collegio di Ingegneria Gestionale

Corso di Laurea in Ingegneria Gestionale dell'Informazione

# RELAZIONE DI TIROCINIO

Presso

## Ermes Cyber Security


Studente: **Sara Cabodi**

Matricola: **214556**


A.A. **2017/2018**


Tutore Accademico: **Fulvio Corno**

Tutore Aziendale: **Hassan Metwalley**

# INDICE

# 1. INTRODUZIONE

Il seguente lavoro descrive l'attività svolta presso l'azienda Ermes Cyber Security, start up incubata presso l'I3P (Incubatore Imprese Innovative del Politecnico di Torino) nell'aprile 2017. L'azienda si inserisce nel mercato della sicurezza informatica, in grande espansione negli ultimi anni grazie ad un forte sviluppo e diffusione della tecnologia. In questo panorama, la gestione della privacy è diventata sempre più problematica e sono state cercate delle soluzioni con diverse strategie e differenti obiettivi. Ermes è nata con l'idea di fornire un servizio di blocco di contenuti indesiderati e di protezione nei confronti dei Web tracker, strumenti che permettono il tracciamento dei dati degli utenti su internet, che possono poi essere analizzati dai Web analytics. La consapevolezza dei danni generati dai Web tracker è infatti aumentata esponenzialmente negli ultimi anni a seguito di impressionanti azioni di hacking e data leakage (fuoriuscita di dati), che hanno interessato grandi aziende ed enti governativi. Per quanto riguarda il blocco di contenuti potenzialmente pericolosi, sono attualmente presenti sul mercato diverse soluzioni. Ad esempio, aziende come Zscaler, Cisco o Check Point forniscono firewall aziendali specializzati nella protezione di attacchi informatici provenienti dall'esterno della rete aziendale, ma purtroppo sono architetturalmente inefficaci contro i Web tracker. Il tentativo di risposta al problema del Web tracking da parte di queste aziende si limita a semplici liste di bloccaggio trimestrali o semestrali che vengono aggiornate manualmente. Tale pratica si rivela del tutto inefficace vista la continua metamorfosi temporale e territoriale dei Web tracker, ed è considerata pertanto non soddisfacente dal mercato. L'implementazione di misure difensive è infatti più complessa ma, pur se molto interessante, non verrà trattata in quanto esula dagli obiettivi di questo lavoro. L'attività svolta si concentra infatti sull'analisi delle prestazioni del filtraggio del traffico web, prendendo in analisi dei concorrenti già presenti sul mercato delle estensioni per browser web di tipo desktop. Si analizzano le prestazioni di browser scelti in due modalità: senza alcun tipo di estensione e con una estensione (con caratteristiche di sicurezza informatica) alla volta. L'estensione di Ermes non è stata testata in quanto ancora in fase di sviluppo. Il programma sviluppato prevede tuttavia un'integrazione molto semplice una volta disponibile lo strumento.

L'elaborato è strutturato in modo da guidare il lettore nella comprensione del lavoro svolto. Nel secondo capitolo si descrive in modo più dettagliato l'azienda,

nel terzo capitolo si spiegano gli obiettivi. In particolare, oltre ai motivi che hanno portato al lavoro svolto, vengono esposti la struttura del programma realizzato, gli strumenti e le tecnologie che ne hanno permesso l'implementazione. Per ognuno di questi ultimi viene presentata una breve descrizione, i motivi e le modalità di utilizzo nell'attività svolta, spesso corredati di esempi esplicativi. Nel quarto capitolo viene affrontato il vero e proprio svolgimento del lavoro. Quest'ultimo è suddiviso in tre fasi principali, corrispondenti a fasi temporali e funzionali. La prima descrive le modalità di collezione dei dati relativi alla navigazione, facendo ricorso agli strumenti analizzati nel capitolo precedente e vagliandone la configurazione e l'utilizzo nel problema affrontato. La seconda presenta scelte e motivazioni riguardanti metriche, siti e strutture dati. Le prime vagliano le possibilità presentate nel paragrafo precedente e ne fanno un'analisi più dettagliata, sui siti viene fatto un approfondimento inerente la quantità e la tipologia di pagine web da utilizzare per la raccolta di dati ed infine viene presentata una lista di strutture dati presenti. Per ogni struttura dati si propongono una breve descrizione, una o più ragioni di utilizzo, l'implementazione e i dati per cui viene usata. Nell'ultima fase vengono presentate le modalità di rappresentazione dei dati raccolti. Ogni tipologia di rappresentazione viene descritta e trattata in base agli strumenti utilizzati ed alle informazioni riprodotte. Per ogni tipologia sono anche riportati una o più immagini destinate a migliorare la comprensione del lettore e fornire degli esempi diretti dei dati trattati. Infine, nel quinto capitolo si riassumono l'attività e le tempistiche dedicate alle varie parti, si fa/cerca di fare un'analisi critica del lavoro svolto e si propongono delle possibili implementazioni o aggiunte future.

# 2. ERMES CYBER SECURITY

Ermes Cyber Security offre una piattaforma per permettere alle aziende di riprendere il controllo delle informazioni che espongono sul Web e dare loro la possibilità di difendersi da tutti i rischi generati dai Web tracker.

I Web tracker sono servizi nati con lo scopo di spiare le operazioni che gli utenti svolgono sul Web e ricavarne profili dettagliati contenenti numerose informazioni personali ed altamente private, che normalmente vengono utilizzate dai giganti dell'advertising digitale per creare pubblicità mirate. Questi servizi oggi si sono altamente evoluti, diventando invasivi al punto di riuscire non solo a identificare in modo puntuale un utente ed i suoi interessi, ma anche la sua dotazione informatica, facendo quello che in gergo tecnico viene chiamato *fingerprinting* (rilevare le versioni dei sistemi operativi sui vari dispositivi, applicativi usati, stato dell'antivirus etc.). L'evoluzione di questi servizi è passata anche da un cambiamento del loro business, con la nascita di centinaia di *data broker*, sottodivisioni (o talvolta enti indipendenti) preposti alla vendita di dati ed informazioni altamente private e sensibili. Queste dinamiche hanno provocato l'esplosione di un vero e proprio mercato in cui la compravendita di dati permette a qualsiasi ente di ottenere segreti e vulnerabilità di ciascun utente. L'utilizzo di queste informazioni da realtà poco trasparenti, come hacker o agenzie criminali, rende ogni utente estremamente vulnerabile ad attacchi esterni non solo per quanto riguarda la sua privacy, ma anche per quanto riguarda la sicurezza propria e di ogni suo dispositivo.

In questo contesto il team di Ermes ha ideato ed ingegnerizzato una soluzione in grado di bloccare totalmente la fuoriuscita di informazioni verso i Web tracker. La tecnologia di Ermes consiste in un sistema direttamente installato su ogni client, sia esso fisso o mobile, che si prende carico di filtrare il traffico Web, autorizzando quello lecito e bloccando quello associato ai Web tracker. Nonostante la piattaforma sia completamente automatica e non richieda alcun intervento umano, le aziende possono facilmente customizzare la loro esprienza tramite un'apposita dashboard.

Ermes offre:

- Miglior filtraggio traffico Web: Ermes può vedere, filtrare e manipolare qualsiasi tipo di traffico Web;
- Protezione di tutti i dispositivi aziendali: Ermes è in grado di proteggere tutti i dispositivi aziendali (desktop, notebook, smartphone, tablet) ovunque essi si trovino (a lavoro, a casa, in aeroporto, ecc.) e con qualsiasi tipo di connessione (via cavo, wifi, 3G, ecc.) senza necessitare dell'uso della VPN;
- Identificazione automatica Web tracker: grazie ad un algoritmo brevettato (in fase di estensione PCT) basato su machine learning, big data ed intelligenza artificiale, la
  soluzione rileva automaticamente nuovi Web tracker o vecchi che hanno mutato il loro comportamento. Grazie a questo algoritmo, Ermes garantisce una protezione continua aggiornata in tempo reale.

# 3. OBIETTIVO, STRUMENTI e TECNOLOGIE

## 3.1 Obiettivo dell'attività

Il lavoro qui descritto riguarda l'implementazione di un programma in linguaggio Python per il benchmarking di prestazioni, fatto su estensioni per browser web con caratteristiche di sicurezza informatica.

Questo obiettivo è nato dalla necessità, da parte dell'azienda Ermes Cyber Security, di testare le prestazioni di un proprio prodotto, ancora in fase di sviluppo, in modo tale da confrontarle con gli attuali concorrenti sul mercato. Inoltre, le informazioni ricavate dalle analisi fatte sono destinate ad un utilizzo di mercato, in modo da poter presentare al cliente informazioni testate sulle prestazioni del prodotto in vendita.

L'obiettivo del lavoro ha dunque natura tecnica, per quanto riguarda la parte relativa ai test ed alla loro implementazione, e natura commerciale in quanto i risultati verranno impiegati nella vendita del prodotto al cliente.

Si è scelto di confrontare profili differenti di uno stesso browser (es. Google Chrome senza e con diverse estensioni) su vari siti web raggruppati in classi in base a criteri decisi. Si visita ogni sito più volte su ogni profilo a scopo statistico. Le statistiche vengono raccolte e visualizzate sia per classe che a livello globale.

Per poter svolgere il compito assegnato, si è richiesto anzitutto di dedicare le prime settimane all'apprendimento del linguaggio Python e delle tecniche utilizzate per la visita automatizzata di pagine web da parte di browser, quali Google Chrome e Firefox. Il restante tempo è stato dedicato alla realizzazione e al perfezionamento del programma per l'analisi delle prestazioni.

## 3.2 Strumenti utilizzati

Il programma è stato interamente sviluppato in *Python* [1], un linguaggio di programmazione ad alto livello definito all'inizio degli anni novanta, ma largamente diffusosi solo a partire dal 2012. Si è utilizzata la versione 3.6.4, la più recente e stabile a disposizione, sia per le prove che per l'implementazione dello script finale.

Per poter usufruire interamente dei package disponibili in *Python*, ma sviluppati da terze parti, si è utilizzato *pip* [2], un sistema di gestione dei pacchetti, che permette di installare e gestire tutti i pacchetti software disponibili, ma non automaticamente installati con Python. Alcuni dei pacchetti installati direttamente tramite *pip* e maggiormente utilizzati in questo lavoro sono: *selenium* [3], *matplotlib* [4], *pandas* [5] e *xlsxwriter* [6]. Altri due pacchetti fondamentali per lo sviluppo del programma sono *browsermob-proxy* [7] e *PyChromeDevTools* [8]. Questi ultimi, a differenza dei precedenti, necessitano di una procedura d'installazione più laboriosa, che richiede alcune istruzioni da terminale dopo l'installazione dello strumento tramite *pip*.

Lo script python si articola in quattro fasi principali:

1. **Apertura del browser e visita delle pagine web desiderate**
   Questa fase viene implementata tramite due possibili strumenti: (vedi pt 3)
   - Selenium
   - PyChromeDevTools
2. **Raccolta delle informazioni sulle visite**
   La raccolta può essere effettuata con due differenti strumenti: (vedi pt 3)
   - Browsermob-proxy
   - PyChromeDevTools
3. **Registrazione delle informazioni in strutture dati intermedie**
   La registrazione intermedia avviene in due modalità:
   1. Le informazioni sulla navigazione di un singolo sito vengono registrate in un file di testo, se relative a tempi e dati scaricati, e in un file xlsx, se relative alla tipologia e quantità di oggetti scaricati
   2. Le informazioni riguardanti i siti vengono registrate in due strutture dati globali:
      - Due liste di n siti, una per le informazioni sui tempi e una sui dati. Ad ogni posizione di una singola lista viene aggiunto un dizionario, corrispondente ad uno specifico sito, con le informazioni relative ad ogni tipologia di profilo
      - Una lista di n siti in cui vengono registrate per ogni sito e per ogni profilo le informazioni relative alla tipologia e quantità di oggetti scaricati

4. **Rappresentazione e visualizzazione delle statistiche**
   La rappresentazione delle statistiche avviene in maniera differente per quanto riguarda la tipologia di metrica da raffigurare e il livello di dettaglio:
   1. Livello di classe (10 siti affini per tipologia):
      - Tempi e dati: rappresentazione delle informazioni raccolte nelle strutture dati globali suddette tramite il pacchetto matplotlib.pyplot e sotto forma di tre istogrammi, uno per ognuna delle due metriche e uno per rappresentare gli eventuali errori
      - Oggetti: rappresentazione delle informazioni raccolte nelle suddette strutture dati globali, aggiunta di una rappresentazione migliore tramite il pacchetto pandas (vedi punto 3) e raffigurazione tramite il pacchetto matplotlib.pyplot di tre istogrammi formati da barre impilate (*stacked bar chart*), uno per ogni tipologia di oggetto
   2. Livello globale (40 siti):
      Rappresentazione delle percentuali delle prestazioni dei profili con estensione rispetto a quello pulito di tutte le classi in un unico grafico sotto forma di tre istogrammi, uno per ogni tipologia di metrica.

Si è deciso di utilizzare Linux come sistema operativo. Nello specifico si è scelta la versione 16.04 TLS della distribuzione Ubuntu, in quanto largamente diffusa e supportata a livello open source.

Per quanto riguarda l'ambiente di sviluppo si è scelto PyCharm [9]. Quest'ultimo è un IDE (Integrated Development Environment) sviluppato dalla compagnia JetBrains che fornisce un ambiente di sviluppo specifico per Python. Sono inoltre disponibili su questa piattaforma uno strumento per l'analisi del codice, un debugger grafico, uno strumento per il testing di unità ed infine una console.

## 3.3 Tecnologie utilizzate

La sezione che segue presenta una breve panoramica sulle tecnologie, gli strumenti e le librerie maggiormente utilizzate per l'implementazione del programma.

## SELENIUM

Questo strumento è stato utilizzato per automatizzare l'apertura e chiusura del browser e la visita delle pagine web. Di questo strumento sono disponibili quattro differenti tipologie. In questo lavoro si è scelta Selenium WebDriver.
Di seguito viene mostrato un esempio di script Python che implementa lo strumento effettuando apertura del browser, visita del sito web di Python e chiusura dello stesso utilizzando Google Chrome come browser. Per prima cosa occorre importare il package corretto di Selenium, ovvero webdriver, in seguito, si procede con l'impostazione del driver desiderato, la get, ovvero la navigazione, della pagina ed infine si effettua la chiusura del browser.

```
1  from selenium import webdriver
2
3  driver = webdriver.Chrome()
4  driver.get("http://www.python.org")
5  driver.close()
```

**Figura 3.3.1: esempio di utilizzo di Selenium WebDriver**

## BROWSERMOB PROXY

Browsermob proxy è uno strumento utilizzato per la raccolta di informazioni sulla navigazione di una pagina web. Nello specifico, permette di catturare i dati di navigazione in un file har in cui si possono successivamente reperire tutte le informazioni. Più in dettaglio, le informazioni presenti nel file sono poste in un dizionario, ovvero una struttura dati di Python che associa ad ogni chiave (univoca) uno o più valori. Al suo interno vi sono altri dizionari che contengono informazioni di vario genere, da quelle sulla versione di strumento utlizzata a quelle riguardanti le richieste o risposte http.

Nel problema affrontato, questa tecnologia deve essere necessariamente abbinata a selenium, in quanto svolge solo la funzione di raccolta e restituzione, ma non quella di navigazione. In aggiunta, questo strumento necessita la presenza del driver relativo al browser utilizzato, ad esempio chromedriver per google chrome oppure geckodriver per mozilla firefox. Occorre poi prestare attenzione alla compatibilità tra le versioni di questi ultimi e quelle dei browser utilizzati, in quanto non tutte le versioni sono sempre supportate. Questa tecnologia, inoltre, si

basa sulla presenza di un server ed un proxy che filtrano le informazioni di navigazione e ne permettono la cattura.

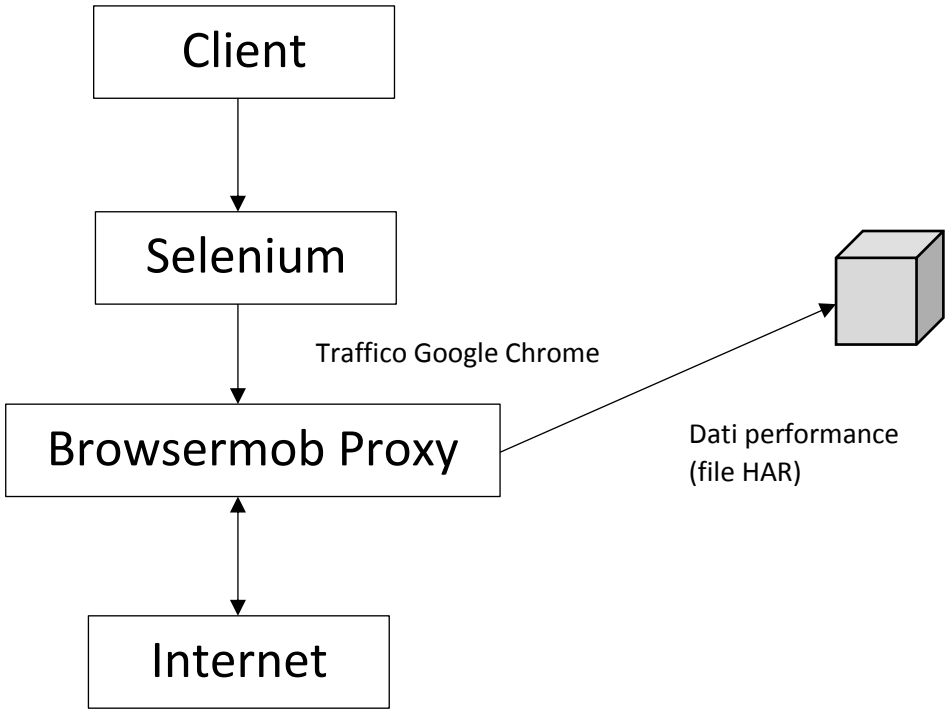



**Figura 3.3.3: rappresentazione schematica dell'interazione tra i vari genti descritti.**

Nella pagina seguente viene presentato un esempio di implementazione dello strumento insieme a selenium. Nell'esempio si utilizza google chrome ed il suo driver, chromedriver. In particolare, si inizializza il server, passando come parametro la stringa con il path per il file installato di browsermob, si procede facendolo partire e creando il proxy, in seguito si svolgono le operazioni di configurazione del driver di Google Chrome e del webdriver. Una volta configurato Chrome, si crea un nuovo file har, si visita la pagina desiderata tramite Selenium e si recupera il file har dal proxy e lo si stampa.

```python
from browsermobproxy import Server
from selenium import webdriver
import time

# settare il path per browsermob proxy
server = Server("/home/sara/browsermob-proxy-2.1.4/bin/browsermob-proxy")
server.start()
proxy = server.create_proxy()

chromedriver = "/usr/local/bin/chromedriver"  # path per il driver
chrome_options = webdriver.ChromeOptions()
chrome_options.add_argument("--proxy-server={0}".format(proxy.proxy))
chrome_options.add_argument('user-data-dir=/home/sara/.config/google-chrome/Profile '+str(1))
chrome_options.add_argument('--disable-application-cache')  # stessa cosa di clear cache??
driver = webdriver.Chrome(executable_path=chromedriver, chrome_options=chrome_options)
driver.set_page_load_timeout(90)

proxy.new_har("python")

start = time.time()
driver.get("http://www.python.org")
finish = time.time()

har = proxy.har  # returns a HAR JSON blob
print(har)

server.stop()
driver.quit()
```

**Figura 3.3.2: esempio di utilizzo di Browsermob-proxy con selenium e chromedriver.**

## PYCHROMEDEVTOOLS

Un'alternativa all'utilizzo di selenium e browsermob proxy è rappresentata da uno strumento implementato da un dottorando in ingegneria elettronica del Politecnico di Torino chiamato PyChromeDevTools [10]. Questo strumento è scritto interamente in python, è disponibile sulla piattaforma GitHub ed è ideato per interagire con lo strumento per sviluppatori di google chrome. Per il corretto funzionamento di questo strumento occorre aprire da terminale una istanza di google chrome, necessariamente l'unica aperta, con l'opzione di debugging remoto ('google-chrome --remote-debugging-port=9222').

I vantaggi di questo strumento sono:

- Maggior velocità d'esecuzione rispetto ai precedenti strumenti
- Assenza di proxy intermedi che in alcuni casi possono impedire il corretto funzionamento di browsermob proxy, ad esempio nel caso di un proxy aziendale
- Interazione diretta con i domini dello strumento per sviluppatori, la quale risulta più intuitiva e immediata

Tra gli svantaggi si trovano invece:

- Utilizzo ristretto a google chrome, in quanto ideato specificatamente per il suo strumento per sviluppatori
- Necessaria unica istanza di chrome fatta partire manualmente da terminale
- Necessaria la comprensione e l'implementazione della teoria degli eventi, ovvero la successione temporale di più azioni automatiche, durante il caricamento della pagina, o fatte dall'utente in seguito, in quanto bisogna attendere eventi pertinenti e corrispondenti ai risultati cercati (vedi paragrafo 4.2.1)
  Più nel dettaglio verranno analizzati gli eventi inerenti al caricamento dei vari oggetti nella pagina

Per l'implementazione definitiva del programma è stato scelto quest'ultimo strumento, in quanto si è ritenuto che i vantaggi fossero più rilevanti degli svantaggi, (in aggiunta ad un'ulteriore complicazione di implementazione con browsermob proxy dovuta al proxy aziendale sulla macchina virtuale utilizzata).

Di seguito viene presentato un esempio di implementazione di PyChromeDevTools con visita della pagina di python e stampa del tempo impiegato a compiere il caricamento della stessa. In particolare, nello script viene inizializzata un'interfaccia di Chrome tramite il package installato di PyChromeDevTools, poi vengono attivati i domini di Network e Page per permettere l'interazione con essi. In seguito, si visita la pagina, si attende che sia lanciato da javascript l'evento 'onload', che corrisponde all'istante di caricamento della pagina, e si stampa la differenza tra quest'ultimo e quello di partenza, corrispondente al tempo totale di caricamento.

```python
import PyChromeDevTools
import time

chrome = PyChromeDevTools.ChromeInterface()
chrome.Network.enable()
chrome.Page.enable()

start_time = time.time()
chrome.Page.navigate(url="http://www.python.org")
chrome.wait_event("Page.loadEventFired", timeout=60)
end_time = time.time()

print("Page Loading Time:", end_time-start_time)
```

**Figura 3.3.3: esempio di utilizzo di PyChromeDevTools con visita della pagina web di Python, attesa dell'evento 'onload' e stampa del tempo di caricamento totale della pagina.**

## MATPLOTLIB

Di questa libreria viene utilizzato principalmente il modulo Pyplot. Si tratta di un insieme di funzioni che permettono di generare rapidamente vari tipi di grafici. Nel lavoro descritto viene implementato principalmente per due tipologie di istogrammi, quella semplice e quella impilata (stacked). Più in dettaglio, si sfruttano le possibilità di creare dei sotto-grafici (subplots) all'interno di una stessa figura. Infatti, come verrà approfondito nel paragrafo 4.3, è stato necessario accorpare più grafici in una sola figura.

## PANDAS

Questa libreria fornisce strutture dati e strumenti di analisi dei dati ad alte prestazioni e facili da usare per il linguaggio di programmazione Python. Nel caso specifico è stata utilizzata per la sua possibilità di implementare e gestire i dataframe. Questi ultimi permettono di rappresentare insiemi di dati eterogenei aggiungendo, se necessario, più livelli di dettaglio. Essi permettono una migliore visualizzazione degli insiemi di dati, senza dover ricorrere ad un file esterno o ad una rappresentazione tabellare, e consentono una più rapida ed intuitiva gestione dei dati anche a differenti livelli di dettaglio. Inoltre, questa libreria permette la

creazione di dataframe da diverse tipologie di strutture dati quali dizionari o serie di essi, dizionari di array o liste, liste di dizionari, dizionari di tuple e serie.

Nelle figure seguenti è rappresentato un esempio di dataframe eseguito con una struttura simile a quella effettivamente implementata nello script ma con numeri generati in modo casuale. Sulle righe si trovano i siti e le tipologie di blocco degli oggetti, mentre sulle colonne sono presenti i vari profili implementati e una colonna finale con la somma dei valori delle colonne precedenti.

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

N_BLOCCHI, N_PROFILI = 3, 3
N_SITI = 5
nomi = ['s1', 's2', 's3', 's4', 's5']
siti = []
for nome in nomi:
    s = [nome]*N_BLOCCHI
    siti.extend(s)

blocchi = ['sempre', 'mai', 'a volte'] * N_SITI
profili = ['pulito', 'adblock', 'ublock']

arrays = [np.array(siti), np.array(blocchi)]
df = pd.DataFrame(np.random.randint(0, 20,
                  size=(N_SITI*N_BLOCCHI, N_PROFILI)),
                  index=arrays, columns=profili)
df['somma'] = df.sum(1)
print(df)
```

```
             pulito  adblock  ublock  somma
s1 sempre        12        2      14     28
   mai           14        8      16     38
   a volte       18        3       2     23
s2 sempre         7        2       8     17
   mai           18       16      15     49
   a volte       14       18       0     32
s3 sempre         3        3       8     14
   mai            1        6      18     25
   a volte        4       15       7     26
s4 sempre        14        8       2     24
   mai           12       18      13     43
   a volte       13        9      13     35
s5 sempre         4       15       5     24
   mai           14       14       8     36
   a volte       18       13      12     43
```

**Figura 3.3.4: esempio di utilizzo di Pandas con relativo output. Impostazione di un DataFrame con sulle righe 5 siti e 3 tipi di blocchi per ogni sito e sulle colonne 3 profili e una colonna complessiva con la somma dei valori dei precedenti. DataFrame simile a quello effettivamente realizzato nello script finale.**

## XLSXWRITER

Xlsxwriter è un modulo Python per creare file Excel XLSX. Tramite questo modulo è possibile scrivere del testo, numeri, formule, collegamenti ipertestuali in vari fogli di lavoro di un file excel. Nel contesto dello script qui discusso è stato utilizzato per memorizzare le informazioni relative agli oggetti scaricati nelle varie visite con i differenti profili per ogni singolo sito. In particolare, per ognuno di questi ultimi sono state realizzate tre tipologie di scrittura di informazioni:

- Una tabella per ogni profilo contenente per ogni visita la quantità delle tre tipologie di oggetti presi in esame, ovvero script, CSS e immagini
- Tre elenchi di url scaricate, corrispondenti ai tre tipi di oggetti, per ogni profilo con un'estensione colorando in rosso, in verde o giallo le url mettendone in evidenza la frequenza di blocco, in questo caso corrispondenti a sempre, mai o a volte
- Tre elenchi di url scaricate, corrispondenti ai tre tipi di oggetti, per ogni profilo con un'estensione rappresentanti eventuali url aggiunte dall'estensione durante il blocco e che quindi non risultano in quelle del profilo pulito, ma sono frutto di reindirizzamenti interni all'azione di blocco

# 4. SVOLGIMENTO DEL TIROCINIO

## 4.1 Prima fase: Analisi delle interfacce e dei metodi per la raccolta delle metriche

### 4.1.1 Browser a disposizione

Per lo svolgimento dell'attività è stato essenziale valutare su quali browser web effettuare i test e per quali motivi. Il panorama dei browser web racchiude opzioni come *Google Chrome, Mozilla Firefox, Edge, Opera* e *Safari*.

Innanzitutto è risultato fondamentale, se non quasi scontato, utilizzare uno dei browser per i quali gli sviluppatori di Ermes stavano sviluppando l'estensione.

Inoltre, ogni browser web possiede una propria configurazione ed un certo livello di compatibilità con gli strumenti descritti nel paragrafo 3.3, oltre che una più o meno cospicua diffusione a livello di utilizzo da parte di clienti singoli e aziende.

Questi fattori hanno portato alla decisione di analizzare in primis Google Chrome ed in un secondo momento, in modo meno approfondito, Mozilla Firefox. Per lo sviluppo completo e definitivo è stata confermata la scelta iniziale.

Per entrambi i browser, si è affrontato il tema dei profili. L'obiettivo, infatti, consta nel paragonare su uno stesso browser le prestazioni di estensioni diverse tra loro o rispetto al browser pulito, ovvero in assenza di qualsiasi tipologia di blocco. È dunque necessario individuare quali concorrenti prendere in considerazione e verificare che siano disponibili per entrambi i browser scelti. Si è deciso di esaminare due principali estensioni concorrenti: Adblock Plus [11] e Ublock [12]. Sulla seconda si sono effettuate delle modifiche, tali da evidenziare maggiormente le differenze di prestazione, ad esempio aggiungendo un filtro per bloccare tutti i contenuti, oppure aggiungendone uno per bloccare solo i contenuti provenienti da terze parti. Si è quindi deciso che ogni estensione sarebbe andata a caratterizzare uno specifico profilo, escluso quello con il browser pulito.

In definitiva, i test sono stati svolti con i seguenti profili:

- Profilo 1 → nessuna estensione, ovvero pulito
- Profilo 2 → Adblock Plus
- Profilo 3 → Ublock in versione base, ovvero senza filtri aggiunti

- Profilo 4 → Ublock con aggiunta del filtro per bloccare i contenuti provenienti da terze parti (‘*$third-party’)
- Profilo 5 → Estensione di Ermes Cyber Security (non implementata, ma possibile realizzazione futura)

I profili vengono preparati manualmente e in seguito richiamati a turno nella funzione dello script in cui viene impostato Google Chrome con il profilo corrente da analizzare.

Di seguito viene mostrata la funzione utilizzata nello script definitivo per impostare il profilo passato come parametro. Si sfrutta lo strumento PyChromeDevTools senza bisogno di lanciare manualmente l’istanza di Chrome, ma automatizzando questo passaggio.

```
89  def set_profilo(num_profilo):
90      if num_profilo not in range(1, N_PROFILI+1):
91          pass
92      else:
93          # impostazione del path per il profilo passato come paramentro
94          PROFILE_DIR = '/home/sara/.config/google-chrome/profilo'+str(num_profilo)
95          # faccio partire il profilo desiderato come se fosse scritto da terminale
96          CHROME_CMD = "google-chrome --remote-debugging-port=" + str(PORT) + \
97                       " --user-data-dir=" + PROFILE_DIR
98          # parte il processo che apre l'istanza di chrome con il profilo richiesto
99          proc = subprocess.Popen(CHROME_CMD, shell=True, stdin=None, stdout=FNULL,
100                                  stderr=subprocess.STDOUT, close_fds=True, preexec_fn=os.setsid)
101         time.sleep(SHORT_TIMEOUT)
102         return proc
```

**Figura 4.1.1: funzione per l’impostazione del profilo con il quale si vogliono effettuare le visite.**

## 4.1.2 Strumenti per scaricare le informazioni sulla navigazione

Una volta individuati i browser sui quali effettuare i test, è stato necessario trovare un modo per raccogliere le informazioni riguardanti la navigazione. Tramite Selenium è possibile aprire il browser desiderato, visitare una o più pagine e chiudere il browser, ma non vi è un metodo per poter raccogliere delle informazioni sulle prestazioni della visita.

In seguito a varie ricerche si sono individuati due possibili strumenti:

- Browsermob Proxy
- PyChromeDevTools

Il primo consiste in una tecnologia che permette di catturare le informazioni delle richieste e risposte http ed esportare i dati relativi alle performance sotto forma di file har. Questo approccio può ed è stato utilizzato combinandolo con Selenium. Infatti, l'uso di questo strumento permette di colmare la lacuna sulla raccolta di informazioni, ma non sostituisce le funzionalità fornite da Selenium. Questo approccio è utilizzabile con entrambi i browser scelti, Mozilla Firefox e Google Chrome, sebbene ci siano delle piccole modifiche da apportare volendo testare l'uno o l'altro browser. Nello specifico, la differenza principale è rappresentata dall'impostazione del driver: per quanto riguarda Mozilla Firefox vi è la creazione di un `FirefoxProfile()`, invece per Google Chrome vengono impostate le cosiddette `ChromeOptions()`, alle quali viene aggiunta una proprietà riguardante il proxy (`add_argument("--proxy-server={0}".format(proxy.proxy))`).

Il secondo strumento, invece, sostituisce la parte di test svolta con Selenium ed è sviluppato per interagire specificatamente con lo strumento per sviluppatori (*developer tools*) di Google Chrome, ed è quindi solo possibile testare questo browser. In questo secondo caso, infatti, si velocizzano le operazioni di raccolta dei dati andando ad interagire direttamente con gli oggetti presenti nel pannello di Chrome, ma è necessario aprire precedentemente da linea di comando una istanza del suddetto browser in modalità remota. Tuttavia, è possibile automatizzare questo ultimo passaggio, ma sono necessari ulteriori strumenti e capacità che nel caso in esame sono stati forniti dal dottorando sviluppatore dello strumento stesso. Il vantaggio fondamentale rispetto allo strumento precedente è costituito dall'assenza di un proxy aggiuntivo, che, ad esempio nel caso di connessioni su reti aziendali con un proxy, può rivelarsi un problema per l'implementazione di Browsermob-proxy in quanto il proxy di quest'ultimo non riesce a registrare le informazioni riguardanti la visita.

In entrambi i casi si tratta di attendere un determinato momento per raccogliere le informazioni desiderate. Questo istante temporale può variare a seconda delle metriche desiderate. Con l'utilizzo di browsermob proxy e selenium, nell'istruzione di visita della pagina risulta insita l'attesa del caricamento completo sia della pagina che di tutte le possibili informazioni che vengono poi scaricate nelle istruzioni successive tramite browsermob. Per quanto riguarda

PyChromeDevTools, invece, il discorso risulta più complicato poiché si aggiunge la teoria degli eventi. Analizzando lo strumento per sviluppatori e la struttura di una pagina web, ci si accorge infatti che i vari oggetti di una pagina vengono caricati in istanti temporali differenti. Ad alcuni di questi momenti sono stati assegnati degli eventi, ad esempio l'inizio o fine del caricamento di un determinato oggetto. Dunque, a seconda delle metriche da collezionare, è necessario attendere gli eventi corretti al fine di collezionare le informazioni coerenti con la ricerca desiderata.

## 4.2 Seconda fase: Scelta delle metriche da collezionare, organizzazione e realizzazione del test

### 4.2.1 Quali metriche collezionare e motivazioni

Una volta analizzati i metodi per la raccolta delle informazioni sulla navigazione, ci si è posti il problema di quali informazioni analizzare e perché.

La scelta sui primi due importanti fattori da analizzare è ricaduta sul *tempo* di caricamento della pagina e sulla quantità di *dati*, ovvero di byte, scaricati durante la visita. Per quanto riguarda la prima metrica è possibile effettuare l'analisi in diversi modi. Infatti oltre alle informazioni ricavabili dai file ottenuti con i due strumenti definiti nel paragrafo 4.1.2, ovvero Browsermob Proxy e PyChromeDevTools, è possibile ricavare il tempo di caricamento della pagina registrando i tempi di inizio e fine dell'istruzione di visita della pagina ed in seguito sottrarli ottenendo la misura desiderata. Questo tipo di implementazione è la più intuitiva e di facile applicazione utilizzando selenium, mentre richiede più attenzione nel caso di interazione con lo strumento per gli sviluppatori di chrome. Infatti, come già anticipato nel paragrafo 3.3 parlando di PyChromeDevTools, sorge il problema di quale o quali eventi attendere per il completo caricamento della pagina.

Con il termine eventi sono indicati gli HTML DOM Events [13], ossia degli strumenti che permettono a JavaScript di registrare diversi 'event handlers' (gestori di eventi) sugli elementi di una pagina HTML. Vi sono delle tipologie di eventi scaturite dall'interazione della pagina con l'utente, come ad esempio l'evento 'onclick' che si verifica quando l'utente clicca si un bottone, oppure

tipologie di eventi che sono legate al caricamento dei vari oggetti della pagina, come ad esempio l'evento 'onload' che viene lanciato alla fine del caricamento.

Per questo programma si sono presi in considerazione più eventi relativi a due principali domini: Network e Page. Inerenti al primo, sono stati analizzati i seguenti eventi:

- Network.loadingFinished (lanciato quando la richiesta http ha finito il caricamento)
- Network.responseReceived (lanciato quando la risposta http è disponibile)

Per quanto riguarda il dominio Page, sono stati analizzati:

- Page.frameNavigated (lanciato quando la navigazione del frame è completata)
- Page.loadEventFired (lanciato quando la pagina ha completato il caricamento)
- Page.DOMContentEventFired (lanciato quando il contenuto del DOM è stato caricato)

In seguito ad una serie di prove e ad un consulto con il realizzatore di PyChromeDevTools, si è giunti alla conclusione che per il caso in esame fossero adeguati gli ultimi due eventi relativi al dominio della pagina, ovvero Page.loadEventFired e Page.DOMContentEventFired, in quanto la loro attesa garantisca il completo caricamento della pagina desiderato. Inoltre, si è compreso come i due eventi fossero sia sequenziali che sommabili, rendendo in tal modo la raccolta delle informazioni sulla navigazione completa.

Per rendere più completa l'analisi, si è inoltre deciso di ricercare la quantità e la tipologia di oggetti scaricati per ogni visita e su ogni profilo. Infatti, ogni pagina possiede un numero variabile di tipologie di oggetti ed ognuna di queste ha al suo interno un certo numero di collegamenti ipertestuali, che individuano unicamente un oggetto. Si è deciso di analizzare principalmente tre tipologie: Script, Stylesheet, Image, ovvero oggetti relativi a script, generalmente in linguaggio JavaScript, a fogli di stile scritti in linguaggio CSS ed a immagini. Dalle informazioni ricavate tramite i due metodi menzionati precedentemente (Browsermob Proxy e PyChromeDevTools), si sono estratte le informazioni relative alle url dei vari oggetti ed alla loro tipologia in modo da poter fare ulteriori

confronti sia qualitativi che quantitativi a livelli differenti di dettaglio. Si è infatti scelto di presentare queste informazioni a livello di classi tramite un grafico globale ed a livello di singolo sito con un foglio Excel, come verrà spiegato in modo più approfondito nel paragrafo 4.3.

Viene mostrata di seguito la funzione implementata nel programma per la raccolta delle suddette metriche. Nella versione finale, come già detto, è stato utilizzato PyChromeDevTools. La funzione sottostante viene richiamata ogni volta che un sito viene visitato con un determinato profilo. Vengono passati come parametri i messages, che corrispondono all'unione dei due dizionari ricavati dall'attesa degli eventi Page.loadEventFired e Page.DOMContentEventFired, l'identificativo del sito, i numeri del profilo e della visita correnti. La funzione ritorna la quantità di dati scaricata in Megabyte e aggiunge il numero di oggetti scaricati alle tipologie presenti nella struttura dati designata a contenerli (nello script indicata con 'info_siti').

```
def calcola_dimensione(messages, id_sito, profilo, visita):
    data = 0
    for m in messages:
        if "method" in m and m["method"] == "Network.responseReceived":
            if "Content-Length" in m["params"]["response"]["headers"]:
                data += int(m["params"]["response"]["headers"]["Content-Length"])
            if "content-length" in m["params"]["response"]["headers"]:
                data += int(m["params"]["response"]["headers"]["content-length"])
            if "type" in m["params"]:  # cerco le tipologie di oggetti scaricate
                tipo = m["params"]["type"]
                if tipo in CAMPI:  # se la tipologia corrisponde ad una di quelle cercate
                    # aggiungo la url al set del sito, del profilo, del tipo, della visita corretti
                    info_siti[id_sito][profilo][tipo][visita].add(m["params"]["response"]["url"])
    dati = data / 1000000
    return dati  # in MB
```

**Figura 4.2.1.1: funzione per il calcolo dei dati scaricati e l'aggiunta degli oggetti alla struttura dati.**

## 4.2.2 Quanti e quali siti visitare

In seguito alle decisioni sopra elencate, è sorto il problema riguardante la scelta dei siti e la loro quantità. Per l'obiettivo posto, si è innanzitutto deciso di prendere prevalentemente in considerazione siti italiani, con eventuali eccezioni, e siti molto conosciuti. In un primo momento, si sono fatte delle prove su siti statici, di modo da facilitarne la visita ed evitare grandi cambiamenti tra diversi istanti temporali tra le visite. In un secondo momento, invece, si è deciso di prendere in considerazione tipologie di siti più dinamici e con alto contenuto di oggetti al loro

interno, di modo da rendere più dettagliata l'analisi. Sono quindi stati scelti i siti di giornali online, di compravendita, di turismo, come hotels o compagnie aeree. Sono invece stati scartati siti a basso contenuto di oggetti, come ad esempio google.it, oppure siti con registrazione e login necessari, come ad esempio tutti i siti di social network.

Per una buona parte del lavoro di implementazione, si è scelto di limitare il numero di siti da visitare a dieci, nella fattispecie siti di giornali online. Questa scelta è stata presa per limitare la durata dei vari test, ma permettere comunque un'analisi significativa di base. Nell'ultima parte del lavoro, si è ampliato il numero di siti a quaranta, con un'ulteriore suddivisione in classi da dieci siti ciascuna, in cui ogni classe corrisponde ad una macro-tipologia. Si è deciso di considerare le seguenti quattro classi: newspapers, e-commerce, turism, electronics.

I siti, in tutte le implementazioni, prove varie e versione definitiva, sono stati salvati uno per riga in un file di testo. Questo viene letto all'inizio dello script e i siti immagazzinati in una lista, su cui si itera per lo svolgimento delle operazioni sui siti stessi.

La scelta sulla quantità di visite da effettuare per ogni sito e su ogni profilo differente è inizialmente ricaduta su dieci. Per ogni decina di visite si è deciso di scartare i valori massimo e minimo ed effettuare la media sui restanti. Le motivazioni principali sono stati i vincoli temporali, in quanto un numero maggiore di visite o di siti avrebbe avuto un maggiore impatto sul tempo totale di esecuzione, e di tipo statistico, poiché effettuare la media su meno di otto valori sarebbe stato poco indicativo. In un secondo momento, è stato poi aumentato a trenta il numero di visite per un singolo sito su un determinato profilo ed è eventualmente aumentabile di un numero a piacere in futuro, in quanto lo script è stato implementato in modo da necessitare il solo cambiamento della costante che rappresenta il numero delle visite.

### 4.2.3 Strutture dati scelte

Il linguaggio Python offre quattro tipi di strutture per le collezioni di dati: liste, tuple, insiemi (o set) e dizionari [14]. Nel programma qui descritto vengono usate tutte tranne le tuple.

Di seguito viene presentata una breve introduzione ad ognuna delle strutture sopra elencate e si dice perché e per quale tipologia di informazione, nonché il modo in cui viene utilizzata.

## LISTA

Una *lista* è una serie ordinata di valori, ognuno identificato da un indice. I valori che la compongono sono chiamati elementi e possono essere di vario tipo, ad esempio una lista può essere formata da stringhe e numeri insieme. Inoltre, è possibile creare delle liste annidate quando uno o più elementi di una lista sono a loro volta delle liste.

Nel programma vengono spesso usate delle liste numeriche quando si effettuano delle iterazioni. Più precisamente, Python offre la funzione range che prende due argomenti e ritorna una lista che contiene tutti gli interi a partire dal primo (incluso) fino al secondo (escluso).

## INSIEME O SET

Un oggetto *set* (insieme) è una collezione non ordinata di valori immutabili. Gli usi comuni includono l'esame dei membri dell'insieme, la rimozione dei duplicati da una sequenza, e la computazione di operazioni matematiche quali intersezione, unione, differenza, e differenza simmetrica. Esistono correntemente due tipi di insiemi built-in, `set` e `frozenset`. Il tipo set è mutabile, in quanto il contenuto può essere cambiato usando metodi come `add()` e `remove()`. Poiché è mutabile, non ha un valore hash e non può venire usato come una chiave di dizionario o un elemento di un altro insieme. Il tipo `frozenset` è immutabile e supporta l'hash -- il suo contenuto non può venire alterato dopo la creazione; tuttavia, può venire usato come una chiave di dizionario o come un elemento di un altro insieme.

Nel programma questa tipologia di struttura viene utilizzata solamente nella versione mutabile, ovvero `set`, per registrare gli indirizzi degli oggetti scaricati (`info_siti[id_sito][profilo][tipo][visita]` corrisponde ad un set di url per la visita indicata) visto che non è necessario un ordine e serve, invece, non avere duplicati.

## DIZIONARIO

Un oggetto mappa tiene traccia dei valori immutabili per gli oggetti arbitrari. Le mappe sono oggetti mutabili. Correntemente esiste solo un tipo mappa, il *dizionario*. Le chiavi di un dizionario sono valori quasi arbitrari. Collezioni quali liste, dizionari o altri tipi mutabili (che sono confrontati per valore piuttosto che per identità dell'oggetto) non possono venire usati come chiavi. I tipi numerici usati per le chiavi obbediscono alle regole normali per il confronto numerico: se due numeri (intero e reale) confrontati sono uguali (come 1 e 1.0), allora possono essere usati in modo intercambiabile per indicizzare le stesse voci del dizionario. I dizionari vengono creati mettendo una lista di coppie *chiave*: *valore* separati da virgole, tra parentesi graffe, per esempio: `{'studente1': 237894, 'studente2': 242734}` o `{237894: 'studente1', 242734: 'studente2'}`.

Nel programma si implementano spesso dizionari come elementi di liste. Più nello specifico, nel caso delle strutture per contenere le metriche sono usati dei dizionari per raccogliere le metriche dei differenti profili usati come chiave (`statistiche_dati` e `statistiche_tempi` sono due liste con un dizionario per ogni elemento, che corrisponde ad un sito) oppure sui diversi tipi di oggetti scaricati (`info_siti` e `info_max` contengono le informazioni sugli oggetti, la prima con un livello di dettaglio maggiore su ogni visita e la seconda contenente una media tra le visite per ogni sito, entrambe contenetenti dei dizionari con chiavi corrispondenti ai tipi di oggetti presi in analisi). In entrambi i casi queste strutture riguardano il sito corrispondente alla loro posizione nella lista.

Nel programma vi sono tre principali informazioni, più gli eventuali errori, che devono essere memorizzate in strutture dati appropriate:

- Le informazioni relative ai TEMPI
- Le informazioni relative ai DATI
- Le informazioni relative agli OGGETTI
- Le eventuali informazioni relative agli ERRORI

La lista viene utilizzata in egual modo per raccogliere le informazioni sui tempi e sui dati. Entrando più nel dettaglio, per ogni classe di siti, vengono create due liste, una per i tempi e l’altra per i dati. Iterando sui siti, si aggiunge come elemento di

ciascuna delle due liste un dizionario. Quest'ultimo ha come chiavi i numeri dei profili (trasformati in stringhe) e come valori delle liste. Ognuna di queste liste è formata dalla media della metrica corrispondente calcolata per il sito e profilo, indicati dalla posizione e dalla chiave, e da un'ulteriore lista contenente le differenze della media con i valori massimo e minimo. Questi ultimi valori serviranno per le error bars dei grafici.

```
statistiche_dati, statistiche_tempi, errori = [], [], {}  # due liste in cui ogni valore e un dizionario

for j in range(N_SITI):  # inizializzo il dict con un id per ogni sito e una lista
    # in cui metterò i valori delle medie
    value_tempi, value_dati = {}, {}
    for k in range(1, N_PROFILI + 1):
        value_tempi[str(k)] = [0, [0] * 2]  # per ogni chiave metto 2 valori : 1 = media, 2 = yerr(lista)
        value_dati[str(k)] = [0, [0] * 2]
    statistiche_dati.insert(j, value_dati)
    statistiche_tempi.insert(j, value_tempi)
```

**Figura 4.2.3.1: Inizializzazione delle strutture dati contenenti le informazioni sui tempi e dati scaricati.**

Le liste vengono anche utilizzate per la memorizzazione delle eventuali informazioni inerenti gli errori di caricamento della pagina. Gli errori presi in considerazione riguardano la fuoriuscita dal timeout impostato per il caricamento della pagina. Utilizzando Browsermob Proxy vi è una probabilità maggiore, generalmente intorno al 3-5%, di manifestazione di questo tipo di errori rispetto a quella di PyChromeDevTools che è pressoché nulla. In entrambi i casi, gli errori vengono memorizzati per ogni classe in una lista di lunghezza pari al numero di siti. Ogni elemento della lista è un numero che rappresenta la quantità di errori avvenuti nelle visite del sito corrispondente alla posizione.

Per quanto riguarda gli oggetti, vengono utilizzate tutte e tre le strutture dati contemporaneamente. Le informazioni sugli oggetti vengono registrate in due strutture dati annidate per ogni classe: una che raccoglie dettagliatamente per ogni visita un insieme di url per ogni tipo di oggetto ed una che registra, a livello più generale su tutte le visite, quante volte i profili con un'estensione scaricano gli oggetti scaricati dal profilo pulito. La prima struttura è una lista di lunghezza pari al numero di siti presenti nella classe. Ogni elemento di questa lista è a sua volta una lista di lunghezza pari al numero di profili. Ogni elemento di quest'ultima è

un dizionario le cui chiavi corrispondono alle tipologie di oggetti considerate e i valori sono dei set di url di oggetti scaricati.

```
info_siti = [[{campo: [set() for s in range(N_VISITE)] for campo in CAMPI}
              for i in range(N_PROFILI)] for j in range(N_SITI)]
```

**Figura 4.2.3.2: Inizializzazione della prima struttura dati contenente le informazioni sulle url di oggetti scaricate suddivise per tipo.**

La seconda è sempre una lista di lunghezza pari al numero di siti presenti nella classe, ma questa volta ogni elemento è un dizionario. Ogni dizionario ha come chiavi i tipi di oggetti ricercati e come valori di nuovo dei dizionari. Questi ultimi vengono inizializzati in un'altra posizione del programma poiché le chiavi devono corrispondere all'insieme massimo di url di oggetti scaricate con il profilo pulito, mentre i valori saranno delle liste che conterranno i contatori di quante volte ogni oggetto, ovvero url, è stato scaricato con il profilo corrispondente alla posizione dell'elemento nella lista.

```
info_max = [{campo: {} for campo in CAMPI} for j1 in range(N_SITI)]
```

**Figura 4.2.3.3: Inizializzazione della seconda struttura dati contenente le informazioni sui contatori degli oggetti scaricati per ogni profilo.**

# 4.3 Terza fase: Implementazione e visualizzazione delle statistiche raccolte

## 4.3.1 Modalità di rappresentazione delle varie informazioni e motivazioni

In questo paragrafo verranno spiegate e illustrate le modalità per la rappresentazione delle informazioni raccolte. Nel programma sono state implementate quattro tipologie di rappresentazioni che corrispondono a diverse metriche e differenti livelli di dettaglio:

1. Rappresentazione con Matplotlib degli istogrammi per i tempi, i dati e gli errori
2. Rappresentazione con Matplotlib degli stacked bar chart (istogrammi impilati) per le informazioni sui tre tipi di oggetti considerati e con Pandas per il DataFrame [15] degli stessi
3. Rappresentazione con Excel (Xlsxwriter) per ogni sito delle tre tipologie di oggetti, per una visione più dettagliata di ciò che viene rappresentato nel punto 2
4. Rappresentazione con Matplotlib delle statistiche finali con le percentuali delle prestazioni dei profili con estensioni rispetto a quello pulito.

La prima tipologia di rappresentazione riguarda le informazioni di tempi, dati ed errori. Questa viene eseguita alla fine delle operazioni svolte sui siti di ogni classe ed è una sola *figure* (figura) che contiene al suo interno tre *subplot* (sotto-grafici), ognuno dei quali è un istogramma. Il terzo, ovvero quello che rappresenta gli errori, è il più semplice ed è un istogramma che rappresenta, per ogni sito sull'asse delle x, la numerosità degli errori, il valore sull'asse delle y. Come già anticipato, con PyChromeDevTools è raro che si presentino degli errori di fuoriuscita dal timeout temporale, ma si è scelto di mantenere comunque il grafico per i casi contrari. I grafici sui tempi e sui dati sono invece formati da tante barre per ogni valore delle x (siti) quanti sono i profili presi in esame. Ogni barra corrisponde alla media della metrica espressa sull'asse delle y del colore del profilo corrispondente segnato in legenda. Ogni barra ha anche una error bar che ha come estremi il minimo ed il massimo valore trovati nelle `n` visite effettuate. Quest'ultimo indicatore aiuta a capire quanto sia accurato ed indicativo il valore della media rappresentato.

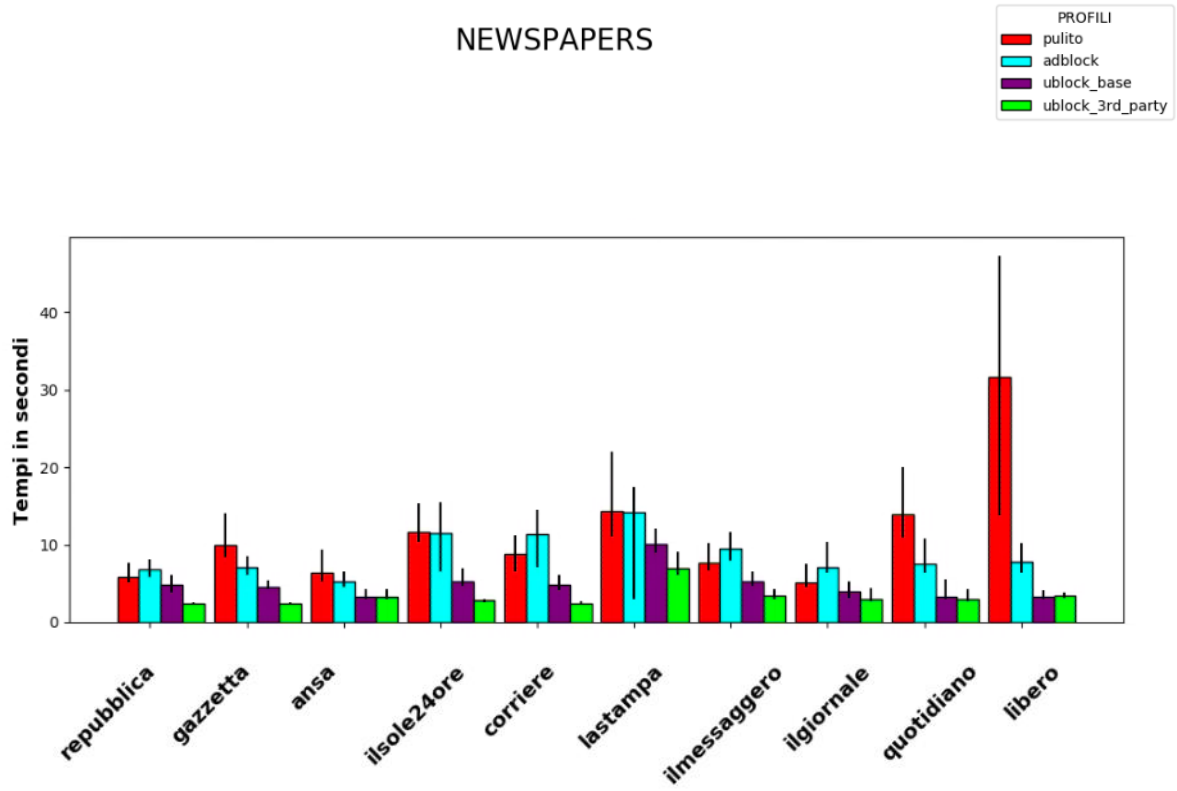



**Figura 4.3.1.1: Esempio di grafico sui tempi sui siti appartenenti alla classe dei giornali online. Quello sui dati è analogo e il terzo non riportato rappresenta gli errori. Ogni barra è dotata di un'error bar e in alto a destra si trova la legenda.**

Gli oggetti sono rappresentati separatamente in quanto richiedono un'ulteriore suddivisione al loro interno. La struttura del grafico è molto simile a quella delle altre metriche in quanto vi è una sola *figure* suddivisa in tre *subplots*. Ognuno di questi singoli grafici è uno *stacked bar chart*, ovvero un istogramma in cui ogni barra è formato da tre blocchi impilati, ognuno corrispondente alla frequenza con cui gli oggetti sono stati scaricati dal profilo corrispondente alla barra completa. Come prima, infatti, per ogni valore delle x, ovvero ogni sito, sono presenti n barre corrispondenti gli n profili considerati. Ogni barra è poi l'insieme di tre blocchi. Ogni blocco corrisponde ad una frequenza di scaricamento degli oggetti della tipologia espressa nel grafico. Sono state scelte tre fasce di frequenza: 0-10% corrispondente al 'mai' è rappresentata in rosso, 10-90% corrispondente al 'a volte' è rappresentata in giallo e 90-100% corrispondente al 'sempre' è rappresentata in verde. Per ogni sito in ogni grafico corrispondente al tipo di oggetto (script, stylesheet e image), la dimensione totale delle barre corrisponde alla massima numerosità degli oggetti scaricati nelle varie visite con il profilo pulito. Sulla base di questa vengono calcolate le frequenze degli oggetti scaricati

con gli altri profili. Infatti, ognuno dei tre blocchi che formano una barra rappresenta la numerosità di url di oggetti scaricate con la frequenza corrispondente al colore. Prima di generare il grafico, però, viene costruito un DataFrame con tutte le informazioni che vengono poi rappresentate in forma di istogramma. Viene fatta questa ulteriore rappresentazione per garantire una miglior comprensione dell'analisi fatta e una diversa visualizzazione dei dati ricavati. Il DataFrame, infatti, consente una visualizzazione più simile a quella tabellare, garantendo la visualizzazione dei vari livelli di dettaglio delle informazioni, com'è stato precedentemente dimostrato con l'esempio raffigurato nella figura3.3.4.

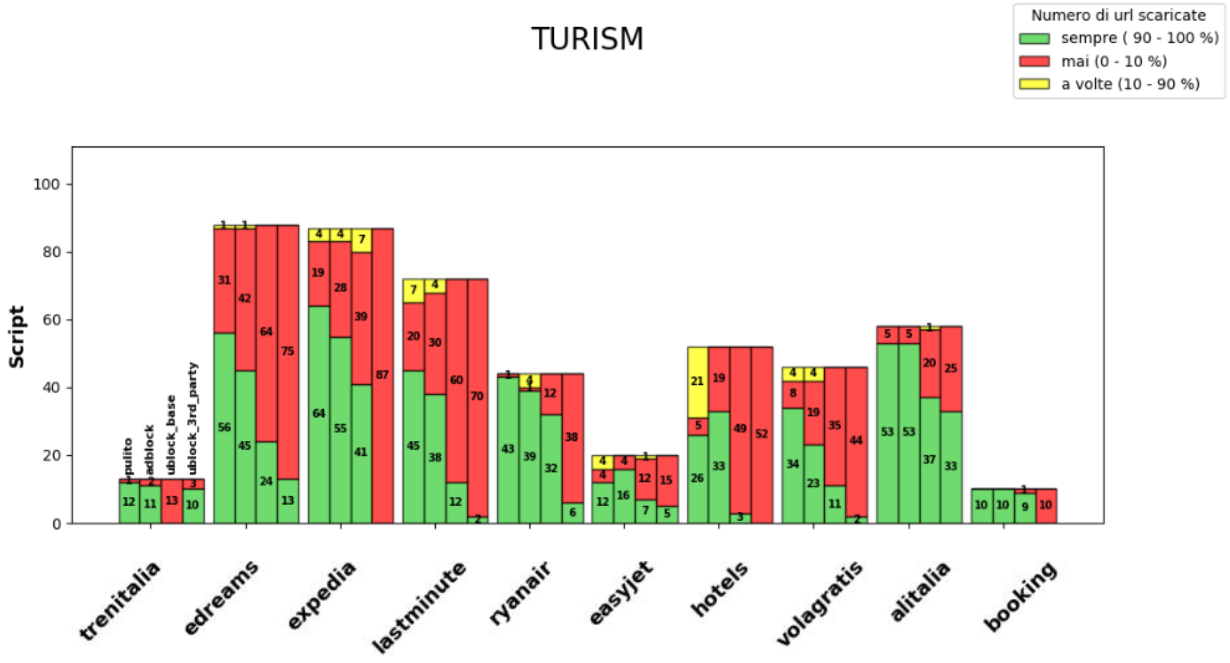



**Figura 4.3.1.2: Esempio di grafico impilato (stacked bar chart) sulla classe contenente siti per turismo. È riportato il primo dei tre grafici, quello sugli Script, che compongono la figura completa. Gli altri due sono analoghi a questo.**

Le informazioni a livello di classe rappresentate nel grafico sovrastante sono anche memorizzate in file con estensione *.xls*, uno per ogni sito. Si è utilizzata la libreria Xlsxwriter che permette di creare e gestire un file Excel tramite il linguaggio Python. Nello specifico, per ogni sito è stato creato un *workbook* ed al suo interno un *worksheet*, ovvero un foglio di lavoro. In quest'ultimo sono state create tabelle ed elenchi per rappresentare tre tipi di informazione per i vari profili:

- Tabelle per la rappresentazione della quantità di oggetti suddivisi per tipologia per ogni visita (tutti i profili)
- Elenchi di url di oggetti scaricati, colorate in base alla loro frequenza di scaricamento, calcolata facendo riferimento all'insieme massimo di url del tipo considerato scaricato con il profilo pulito (solo profili con estensione)
- Elenchi di eventuali url aggiuntive rispetto all'insieme di quelle scaricate con il profilo pulito, poiché alcuni blocchi delle estensioni generano dei reindirizzamenti che non devono essere considerati, ma comunque mostrati per capire su quali oggetti vengono fatti (solo profili con estensione)

La prima tipologia di tabella viene implementata per ogni profilo, compreso quello pulito, ed è formata da tante colonne quanti sono i tipi di oggetti indagati più una per il numero della visita corrispondente. Le righe corrispondono, quindi, al numero di visite effettuate per ogni profilo su un singolo sito.

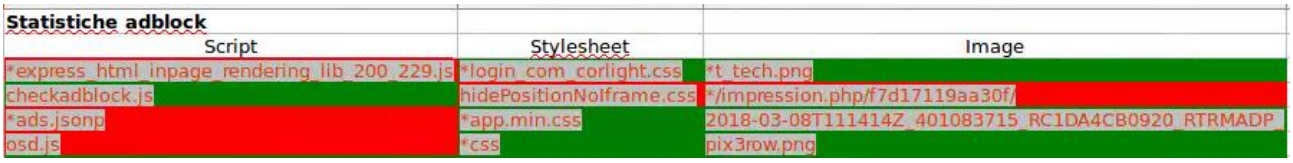
| PULITO | | | |
|---|---|---|---|
| Visita | Script | Stylesheet | Image |
| 1 | 83 | 5 | 91 |
| 2 | 71 | 8 | 126 |
| 3 | 72 | 8 | 126 |
| 4 | 71 | 8 | 126 |
| 5 | 71 | 8 | 126 |
| 6 | 71 | 8 | 126 |
| 7 | 72 | 8 | 126 |
| 8 | 76 | 8 | 136 |
| 9 | 71 | 8 | 126 |
| 10 | 84 | 8 | 138 |

**Figura 4.3.1.3: Esempio di tabella con la quantità di oggetti scaricati per tipologia e per visita. Tabella riferita al profilo pulito e inserita nel file con il nome_del_sito.xls. Numero di visite ridotto a 10 per la rappresentazione.**

Per la realizzazione degli elenchi, descritti brevemente nel secondo punto, è necessario cercare e salvare l'insieme massimo di url per ogni tipo di oggetto del profilo senza estensioni. All'interno di questo insieme vengono segnate con un asterisco le url che non vengono sempre scaricate con il profilo pulito. L'insieme massimo risulta essere il termine di paragone per comprendere con quale frequenza gli altri profili scaricano i vari oggetti. Infatti, le url presenti nell'elenco sono quelle

dell'insieme massimo, ma colorate in base alla frequenza con cui sono scaricate dal profilo considerato. I colori sono:

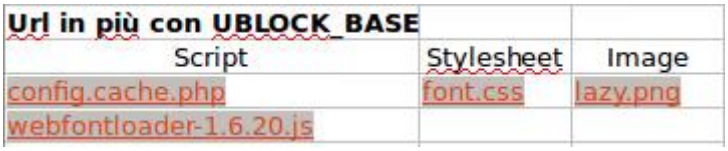
- VERDE se l'oggetto in esame è stato scaricato dal 90 al 100% delle volte
- GIALLO se l'oggetto in esame è stato scaricato dal 10 al 90% delle volte
- ROSSO se l'oggetto in esame è stato scaricato dallo 0 al 10% delle volte



| Statistiche adblock | | |
|---|---|---|
| Script | Stylesheet | Image |
| *express_html_inpage_rendering_lib_200_229.js | *login_com_corlight.css | *t_tech.png |
| checkadblock.js | hidePositionNoIframe.css | */impression.php/f7d17119aa30f/ |
| *ads.jsonp | *app.min.css | 2018-03-08T111414Z_401083715_RC1DA4CB0920_RTRMADP_ |
| osd.js | *css | pix3row.png |

**Figura 4.3.1.4: Esempio di estratto dell'elenco di oggetti con la loro frequenza di scaricamento per il profilo con l'estensione Adblock Plus.**

Gli elenchi descritti nel terzo punto sono invece frutto di *redirect*, ovvero reindirizzamenti, interni all'estensione. Più precisamente, quando un oggetto viene bloccato da un'estensione, può succedere che questa operazione generi un reindirizzamento ad un'altra pagina, che però non viene visualizzata, ma risulta nelle url di oggetti scaricati. Occorre, dunque, trattare in modo diverso questi oggetti per il tipo di analisi desiderata. Queste url non sono conteggiate nelle tabelle ed elenchi sopra descritti, ma vengono visualizzate in un elenco aggiuntivo, suddivise per tipo e per profilo.



| Url in più con UBLOCK_BASE | | |
|---|---|---|
| Script | Stylesheet | Image |
| config.cache.php | font.css | lazy.png |
| webfontloader-1.6.20.js | | |

**Figura 4.3.1.5: Esempio di estratto dell'elenco di oggetti scaricati in più, rispetto al profilo pulito, con il profilo con estensione Ublock senza filtri aggiunti.**

L'ultima tipologia di grafico viene implementata a livello globale. La struttura dati di riferimento è stat_tot, un dizionario le cui chiavi corrispondono alle metriche considerate ed i valori sono due liste una dentro l'altra. La più esterna colleziona i dati sulle varie classi più quella totale e la lista più interna contiene le medie dei valori calcolate per ogni profilo. Di seguito viene presentata l'inizializzazione

della struttura con l'aggiunta di un livello di dettaglio delle tipologie di oggetti considerate, indicate come campi, nella parte inerente alla terza metrica.

```
stat_tot = {metrica: [[0 for profilo in range(N_PROFILI - 1)] for classe in range(N_CLASSI + 1)]
            for metrica in METRICHE}  # N_CLASSI + 1 per la colonna del totale
for c in range(N_CLASSI + 1):
    for profilo in range(N_PROFILI - 1):
        stat_tot['oggetti'][c][profilo] = [0 for campo in range(N_CAMPI)]  # vado ad aggiungere la parte
        # sui campi per quanto riguarda gli oggetti
```

**Figura 4.3.1.6: inizializzazione della struttura dati contenente le informazioni riassuntive su tutte le metriche di tutte le classi più una globale. Queste informazioni sono visualizzate nella prossima figura.**

L'impostazione del grafico è simile a quelle precedenti: si utilizza la libreria Matplotlib per creare una sola *figure* suddivisa in tre *subplots*, ognuno dei quali è dedicato ad una delle tre metriche, ovvero tempi, dati e oggetti. I primi due grafici sono istogrammi che rappresentano per classe la variazione media percentuale, in aumento o riduzione, dei valori con i profili con un'estensione rispetto a quello pulito. Sull'asse delle x sono quindi presenti le n classi considerate più una, che rappresenta la media generale tra le classi. Per ognuno di questi valori sono presenti tante barre quanti sono i profili con un'estensione, ovvero il numero totale di profili meno uno. L'ultimo grafico, invece, ha una struttura più simile al secondo grafico descritto in precedenza, ovvero quello per la rappresentazione degli oggetti con barre impilate. I valori delle x sono rappresentati dalle classi più un ultimo valore riassuntivo. Per ogni valore delle x sono rappresentate tante barre quanti sono i profili con un'estensione. Ognuna di queste barre è composta da tanti blocchi quanti sono i tipi di oggetti presi in considerazione. Ognuno di questi blocchi rappresenta la variazione media percentuale di oggetti scaricati dl tipo indicato dal colore.

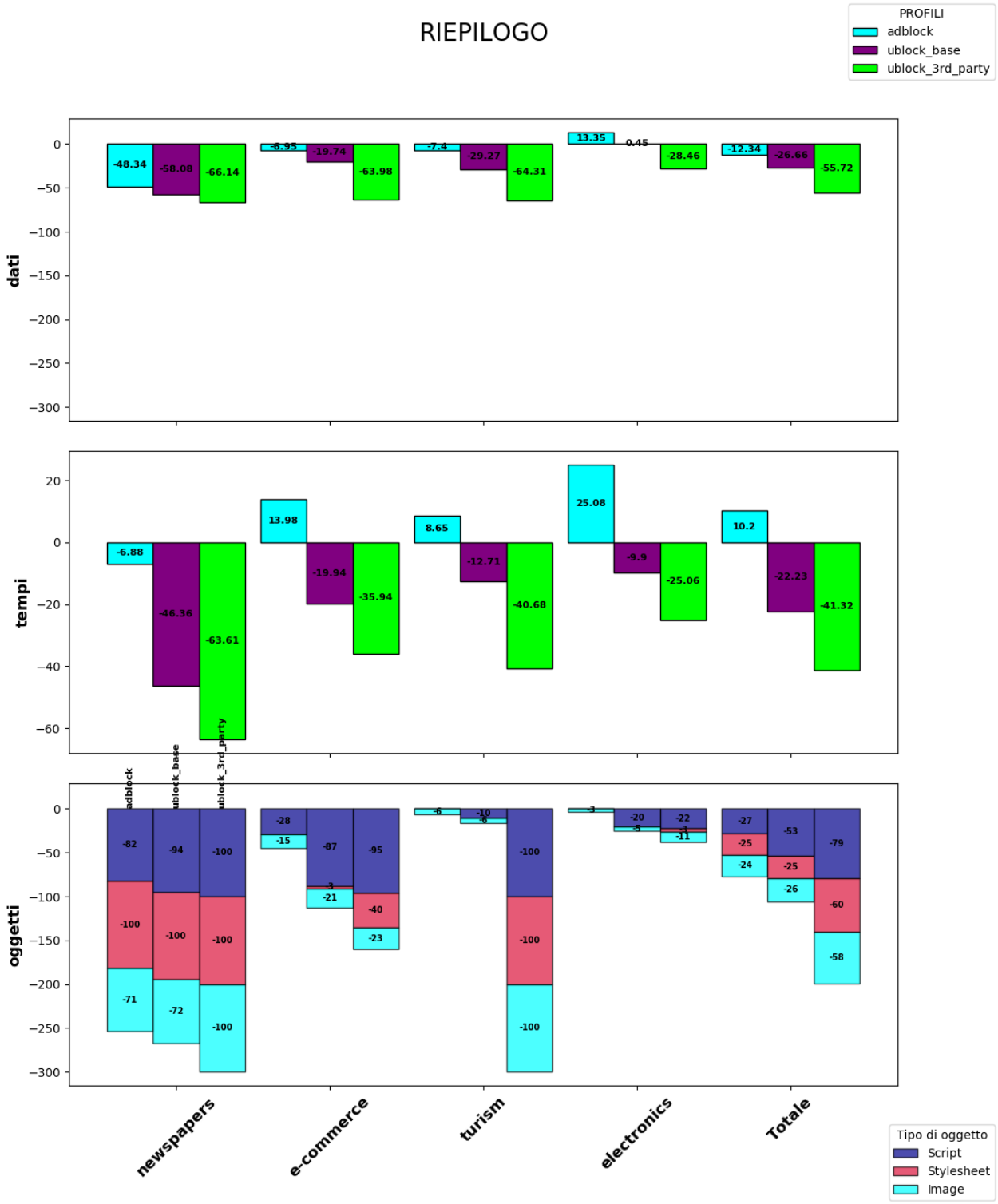



**Figura 4.3.1.7: Esempio di grafico completo riassuntivo. Ogni sotto-grafico rappresenta una delle tre metriche analizzate. I numeri all'interno dei blocchi e delle barre rappresentano la variazione media percentuale per il profilo e la classe corrispondenti.**

## 4.4 Dati sperimentali

Il programma di test è stato sperimentato su un insieme di siti, mediante esecuzioni ripetute, raccolta di dati e visualizzazione di statistiche, come descritto nei paragrafi precedenti. Seppure preliminari, su un solo browser con un insieme limitato di estensioni, i dati ottenuti sono in linea con gli obiettivi preposti, in quanto riescono a catturare e visualizzare differenze di esecuzione e di prestazione tra i diversi profili e siti.

I test svolti hanno portato alla collezione e rappresentazione di tre tipologie di metriche: tempi di caricamento delle pagine, dati scaricati in MB, quantità di tre tipologie di oggetti, cioè script, fogli di stile e immagini. Ognuna di queste quantità è stata registrata per ogni visita effettuata per ogni sito e profilo. In totale sono state fatte trenta visite su quaranta siti suddivisi in quattro classi. Si sono analizzati quattro profili: il primo senza estensioni, il secondo con l'estensione 'Adblock Plus' [1], il terzo con l'estensione 'Ublock' [2] ed il quarto con l'ultima estensione e l'aggiunta di un filtro per bloccare i domini provenienti da terze parti. I grafici visti nel paragrafo precedente rappresentano le medie dei valori raccolti nelle singole visite su più livelli. Ognuna delle metriche è infatti rappresentata sia a livello di classe che a livello globale. Come già evidenziato, i valori raccolti sono preliminari e richiedono ulteriori passi di validazione e ingegnerizzazione.

I test svolti con la versione attuale del programma hanno messo in evidenza alcuni trend ripetuti e delle caratteristiche comuni. Per quanto riguarda i tempi di caricamento, si è riscontrata una tendenza del secondo profilo, Adblock Plus, ad aumentare un po' rispetto al profilo pulito, mentre gli altri due profili a diminuire proporzionalmente alla quantità di blocchi, ossia una diminuzione maggiore per l'ultimo profilo con l'aggiunta del filtro per le terze parti.

Sui dati, invece, i risultati ottenuti mettono in evidenza una graduale diminuzione di MB dal primo all'ultimo profilo, in linea con quelle che erano le aspettative iniziali. In entrambi i grafici sono tuttavia presenti picchi anomali, ad esempio un valore di MB dell'ultimo profilo molto maggiore del primo, i quali sono di difficile interpretazione, tenuto conto che il programma ignora, per scelta progettuale, i dettagli interni dei browser e degli strumenti utilizzati.

Per quanto riguarda gli oggetti, i grafici impilati sottolineano bene la graduale variazione della quantità di blocchi effettuati sui contenuti. Si evidenzia infatti un incremento di contenuti bloccati dal primo all'ultimo profilo, che nella maggior parte dei casi blocca quasi tutti gli oggetti. Questa tendenza è ben rappresentata anche nel grafico finale di riepilogo in cui tutte le tipologie di oggetti analizzate hanno una variazione media percentuale negativa rispetto al primo profilo, in base al quale sono state calcolate. Inoltre, è visibile il graduale aumento in valore assoluto di queste percentuali passando dal secondo al quarto profilo.

# 5. VALUTAZIONI CONCLUSIVE

Il tirocinio ha rappresentato un'esperienza formativa indubbiamente molto valida e interessante, sia dal punto di vista tecnico che delle modalità operative e di realizzazione del lavoro proposto.

Le prime settimane del tirocinio sono state dedicate ad attività di preparazione, quali lo studio [16] e l'implementazione di prove preliminari, orientate a familiarizzare con il linguaggio di programmazione, mai utilizzato precedentemente in ambito scolastico. Successivamente, il lavoro si è rivolto all'implementazione del programma in versione 'ridotta', cioè concentrandosi sulle strategie e modalità di collezione dei dati, scegliendo di implementare i test con un numero limitato di siti e di visite. L'ultima fase, le ultime settimane di lavoro, è stata dedicata alla versione finale completa del programma, operante su un numero maggiore siti e con più visite, ripetute a scopi statistici. Inoltre, si è definita e realizzata la presentazione dei dati, a livello globale e per classi, in modo da avere una visione d'insieme sulle analisi effettuate.

Dal punto di vista metodologico, il programma è stato pianificato e realizzato con una strategia modulare, sia a livello di strutture dati che di metodi/funzioni, seguendo criteri empirici basati su dimensione del codice e funzionalità svolta. Si possono individuare tre famiglie di funzioni: quelle che implementano la visita dei siti e la raccolta delle metriche, quelle per la rappresentazione su fogli Excel dei dati raccolti ed infine quelle che riguardano la rappresentazione tramite grafici delle metriche. Il programma, nella versione attuale, è monolitico, cioè non separa le tre fasi, che vengono attivate in una unica esecuzione: questo al fine di semplificare le strutture dati, evitando di affrontare il problema della raccolta dati su file con un opportuno formato. Più in dettaglio, la parte dedicata alla rappresentazione viene attivata direttamente durante la visita e raccolta dei dati. Questo aspetto, pur semplificando la scrittura del codice, ha richiesto un maggior numero di esecuzioni, per raccogliere nuovi dati, durante la messa a punto del programma. D'altro canto, questa tipologia permette una manipolazione diretta dei dati e la possibilità di costruire dei grafici intermedi, come quelli fatti sulle metriche delle singole classi, e, di conseguenza, verificarne il funzionamento aggiornando ogni volta i dati collezionati.

Il programma realizzato propone statistiche su tempi di caricamento, dati scaricati espressi in MB, quantità e tipologia di oggetti scaricati. Le statistiche sono rappresentate a livello globale e per classi di siti opportunamente individuate. I test effettuati mettono in evidenza differenze percettibili tra le varie configurazioni del browser. In alcuni casi si riscontrano anomalie, quali picchi di traffico dati, di difficile interpretazione, vista la natura modulare del programma al di sopra ed al di fuori dei dettagli interni dei browser e degli strumenti utilizzati.

Si potrebbe cercare in futuro di rendere più efficiente l'analisi dividendo lo script in due programmi differenti, uno per la collezione delle statistiche e l'altro per la loro rappresentazione. Inoltre, i numeri dei siti e delle visite potrebbe essere ampliato in modo da fornire statistiche più accurate e rappresentative. Sarebbe possibile creare delle liste di siti 'su misura' del cliente, ossia fare un'analisi customizzata sui bisogni specifici, che possono essere differenti sia tra varie aziende che tra clienti singoli. Inoltre, visto che il programma sviluppato è implementabile solo per Google Chrome si potrebbe cercare un nuovo metodo o portare avanti quello con Browsermob Proxy per poter effettuare un'analisi analoga su altri browser.

# 6. BIBLIOGRAFIA e SITOGRAFIA


1. https://docs.python.org/3/
2. https://pip.pypa.io/en/stable/
3. http://selenium-python.readthedocs.io/
4. https://matplotlib.org/api/pyplot_summary.html
5. https://pandas.pydata.org
6. http://xlsxwriter.readthedocs.io/
7. https://browsermob-proxy-py.readthedocs.io/en/stable/
8. https://github.com/marty90/PyChromeDevTools
9. https://www.jetbrains.com/pycharm/
10. Martino Trevisan
11. https://adblockplus.org/
12. https://www.ublock.org/
13. https://chromedevtools.github.io/devtools-protocol/
14. https://docs.python.org/2/tutorial/datastructures.html
15. https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.html
16. Marco Beri, *Python 3*, Apogeo, 2010