**Scraping Death Claims Settlement Percentages**

#### **Overview**

This documentation provides an in-depth look at the web scraping script designed to extract death claims settlement percentages from a specified website and securely encrypt the data for sharing. The script utilizes Selenium for web scraping and OpenSSL for data encryption and decryption.

**Python Scraping**

1. **Setting Up Beautiful Soup**: The script sets up Beautiful Soup and navigates to the target website containing the tables with death claims settlement data.

import requests

from bs4 import BeautifulSoup

import csv

url = <https://economictimes.indiatimes.com/wealth/insure/life-insurance/latest-life-insurance-claim-settlement-ratio-of-insurance-companies-in-india/articleshow/97366610.cms>

response = requests.get(url)

response.raise\_for\_status()

soup = BeautifulSoup(response.content, 'html.parser')

1. **Scraping the Tables**: The script defines a function extract\_table\_data to extract data from the specified tables and then convert them csv files.

def extract\_table\_data(table):

rows = table.find\_all('tr')

table\_data = []

for row in rows:

cells = row.find\_all(['td', 'th'])

row\_data = []

for cell in cells:

colspan = int(cell.get('colspan', 1))

cell\_text = cell.get\_text(strip=True)

row\_data.extend([cell\_text] \* colspan)

table\_data.append(row\_data)

return table\_data

def save\_to\_csv(table\_data, filename):

with open(filename, 'w', newline='', encoding='utf-8') as file:

writer = csv.writer(file)

for row in table\_data:

writer.writerow(row)

table1 = soup.find\_all('table')[0]

table1\_data = extract\_table\_data(table1)

save\_to\_csv(table1\_data, 'individual\_death\_claims\_by\_policies.csv')

table2 = soup.find\_all('table')[1]

table1\_data = extract\_table\_data(table2)

save\_to\_csv(table1\_data, 'individual\_death\_claims\_by\_amount.csv')

**Data Encryption**

1. **Encrypting Files**: I used OpenSSL to encrypt the generated CSV files with AES-256-CBC encryption, enhanced with PBKDF2 and a high iteration count for improved security.

openssl enc -aes-256-cbc -salt -pbkdf2 -iter 100000 -in {file\_path} -out {encrypted\_file\_path} -k {password}

1. **Explanation of Encryption Parameters**:

* -aes-256-cbc: Specifies the AES-256-CBC encryption algorithm.
* -salt: Adds a random salt to the encryption for better security.
* -pbkdf2: Uses PBKDF2 (Password-Based Key Derivation Function 2) for key derivation.
* -iter 100000: Specifies 100,000 iterations for the key derivation function to enhance security against brute-force attacks.

