# INDEED SCRAPER - Project Report

Done By Prithika

## INTRODUCTION

The **Indeed Scraper** is a Python-based tool that automates the extraction of job listings from **Indeed**, one of the world's largest job search platforms. It collects key details such as **job title, company, location, salary (if available), rating, job description, posting date, and job URL**.

This scraper is valuable for **job seekers** who want to quickly compile opportunities, **recruiters** tracking hiring patterns and salary trends, and **researchers** analyzing job market dynamics. By exporting results into **CSV format**, it enables easy **analysis, visualization, and long-term tracking** of job opportunities, supporting smarter decision-making and data-driven insights.

## FEATURES

- Search and extract job listings by keyword and location.
- Collect details: title, company, location, salary, rating, date, description.
- Multi-page scraping for more results.
- Optional job descriptions via Requests + BeautifulSoup.
- Multi-threaded fetching for faster scraping.
- Save results in CSV format.

## TECHNOLOGY USED

- Python – Core language.
- Selenium – Dynamic page scraping.
- Requests + BeautifulSoup – Job description extraction.
- Pandas – Data storage and export.
- ThreadPoolExecutor – Concurrent scraping.
- Webdriver-Manager – Automatic ChromeDriver setup.

## INSTALL REQUIRED LIBARARIES

**# Core dependencies**

- pip install selenium
- pip install webdriver-manager
- pip install requests
- pip install beautifulsoup4
- pip install pandas

## PYTHON LIBRARIES USED

- selenium → Browser automation for job listings.
- webdriver-manager → Manages ChromeDriver automatically.
- requests → Fetches job description pages.
- beautifulsoup4 → Parses HTML content.
- pandas → Stores and exports data to CSV.
- concurrent.futures → Multi-threading for faster scraping.

## CODING:

Step by step process

### Step 1: Import Required Modules

```
import argparse, json, os, random, re, time

from datetime import datetime, timedelta

from concurrent.futures import ThreadPoolExecutor, as_completed

try:

    import pandas as pd

except:

    pd = None
```

```
try:
    import requests
    from bs4 import BeautifulSoup
except:
    requests = None; BeautifulSoup = None
try:
    from selenium import webdriver
    from selenium.webdriver.common.by import By
    from selenium.webdriver.support.ui import WebDriverWait
    from selenium.webdriver.support import expected_conditions as EC
    from selenium.webdriver.chrome.service import Service as
ChromeService
    from webdriver_manager.chrome import ChromeDriverManager
    SELENIUM_OK = True
except:
    SELENIUM_OK = False
```

## Step 2: Define Utility Functions

```
UA_POOL = [
    "Mozilla/5.0 (Windows NT 10.0; Win64; x64) Chrome/123 Safari/537.36",
    "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) Safari/605.1.15",
]
def clean_text(s):
```

```
    return re.sub(r"\s+", " ", str(s or "")).strip()
def parse_date_posted(text):
    today = datetime.today().date()
    t = text.lower()
    if "today" in t or "just posted" in t:
        return str(today)
    m = re.search(r"(\d+)\s*day", t)
    if m:
        return str(today - timedelta(days=int(m.group(1))))
    return str(today)
def build_search_url(query, location, start=0):
    from urllib.parse import urlencode, quote_plus
    return "https://www.indeed.com/jobs?" + urlencode(
        {"q": query, "l": location, "start": start},
        quote_via=quote_plus
    )
```

## Step 3: Configure and Launch WebDriver

```
def make_driver(headless=True):
    opts = webdriver.ChromeOptions()
    if headless:
        opts.add_argument("--headless=new")
    opts.add_argument(f"--user-agent={random.choice(UA_POOL)}")
```

```
    return webdriver.Chrome(
        service=ChromeService(ChromeDriverManager().install()),
        options=opts
    )
```

## Step 4: Scrape Job Listings

```
def scrape_list_page(driver, url):
    driver.get(url)
    WebDriverWait(driver, 10).until(
        EC.presence_of_all_elements_located((By.CSS_SELECTOR, "div.job_seen_beacon"))
    )
    out = []
    for c in driver.find_elements(By.CSS_SELECTOR, "div.job_seen_beacon"):
        def safe(css, attr=None):
            try:
                el = c.find_element(By.CSS_SELECTOR, css)
                return el.get_attribute(attr) if attr else el.text
            except: return ""
        link = safe("a", "href")
        if link.startswith("/"):
            link = "https://www.indeed.com" + link
        out.append({
            "job_title": clean_text(safe("h2.jobTitle")),
```

```
            "company": clean_text(safe("span.companyName")),
            "location": clean_text(safe("div.companyLocation")),
            "salary": clean_text(safe("div.metadata.salary-snippet-container")),
            "job_description": "",
            "date_posted": parse_date_posted(clean_text(safe("span.date"))),
            "job_url": link
        })
    return “ ”
```

## Step 5: Fetch Job Descriptions

```
def fetch_description(url):
    if not (requests and BeautifulSoup):
        return ""
    try:
        r = requests.get(url, headers={"User-Agent":
random.choice(UA_POOL)}, timeout=15)
        soup = BeautifulSoup(r.text, "html.parser")
        node = soup.select_one("#jobDescriptionText")
        return clean_text(node.get_text("\n")) if node else ""
    except:
        return ""
```

## Step 6: Save Extracted Data

```
def save_outputs(rows, base):
```

```
    if pd:
        pd.DataFrame(rows, columns=["job_title","company","location",
                                    "salary","job_description",
                                    "date_posted","job_url"]
        ).to_csv(base+".csv", index=False, encoding="utf-8")
```

## Step 7: Main Function to Run Scraper

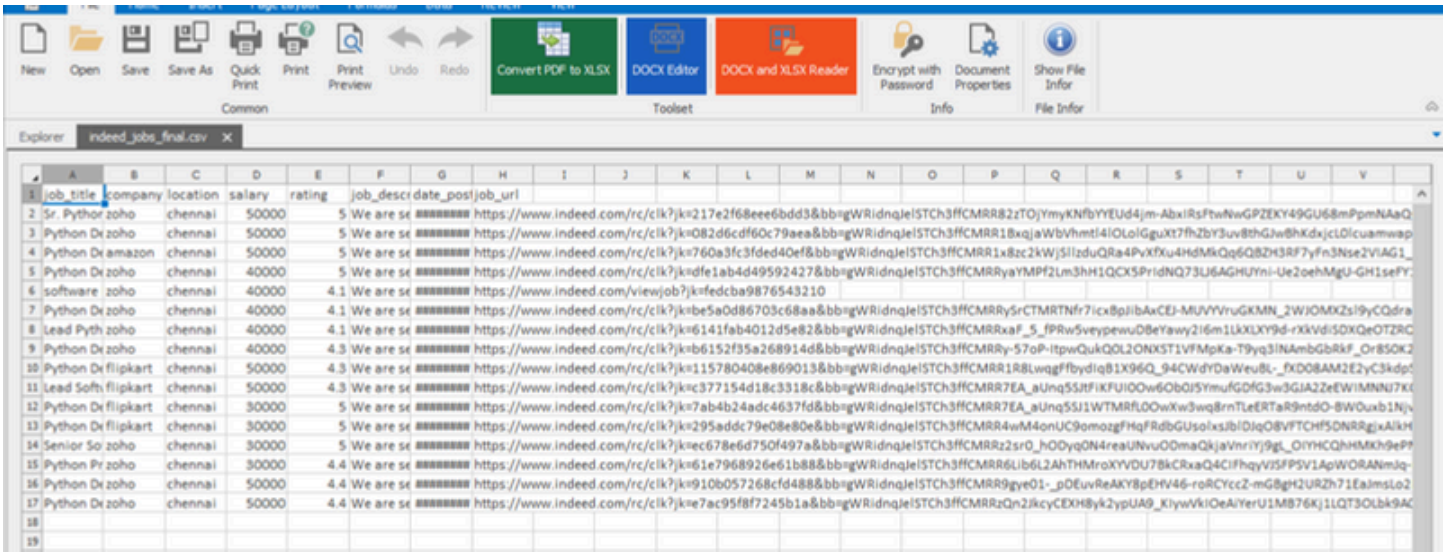
```
def run(query, location, pages=1, fetch_desc=False):
    driver = make_driver()
    try:
        rows = []
        for p in range(pages):
            rows.extend(scrape_list_page(driver, build_search_url(query, location, p*10)))
            time.sleep(1)
        if fetch_desc:
            with ThreadPoolExecutor(max_workers=5) as ex:
                futs = {ex.submit(fetch_description, r["job_url"]): i for i, r in enumerate(rows)}
                for fut in as_completed(futs):
                    rows[futs[fut]]["job_description"] = fut.result()
        save_outputs(rows, f"indeed_jobs_{datetime.now().strftime('%Y%m%d_%H%M%S')}")
    finally:
```

```
    driver.quit()
```

## Step 8: Entry Point

```
if __name__== "__main__":
    run("Python Developer", "Chennai", pages=1, fetch_desc=True)
```

## OUTPUT


## OUTPUT OF URL LINK-WEB PAGE

**LINK: [INDEED SCRAPER]**

## TBOUBLESHOOTING TABLE

### Troubleshooting Table

| Issue | Cause | Solution |
|---|---|---|
| CSV file not saving | CSV already open or locked by another program | Close the file before running. Use `try-except` to handle `PermissionError`. |
| Job description not fetched | `requests` or `BeautifulSoup` not installed | Install using `pip install requests beautifulsoup4`. |
| WebDriver not compatible | Chrome version mismatch with ChromeDriver | Update Chrome and reinstall driver (`pip install webdriver-manager --upgrade`). |
| No job listings scraped | Indeed changed its HTML structure | Inspect updated DOM, update CSS selectors in `scrape_list_page()`. |
| Timeout waiting for elements | Page loading slowly | Increase `WebDriverWait` timeout or use `time.sleep()` as fallback. |
| Script crashes without quitting browser | Driver not closed properly | Wrap code in `try-finally` block (already implemented with `driver.quit()`). |
| Headless mode blocked | Some sites detect headless browsers | Run with `headless=False` or add anti-detection options. |
| Requests blocked / Captcha | Indeed anti-scraping protections | Use proxies, rotate User-Agents, add random delays between requests. |

## SCOPE OF THE PROJECT

- Collects job listings from Indeed.
- Extracts job title, company, location, salary, rating, posting date, description, and URL.
- Outputs results in CSV for Excel, Power BI, or analytics tools.
- Supports multi-page scraping for comprehensive data.
- Runs locally with Python, Chrome, and required libraries.
- Useful for job seekers, recruiters, and researchers.
- Can be extended with database storage, scheduling, filtering, or visualization.

## METHODOLOGY

- Initialize Chrome WebDriver (headless optional).

- Build Indeed search URL with query, location, and pagination.
- Load job listings and wait for job cards.
- Extract job details from each card.
- Optionally fetch job descriptions via Requests & BeautifulSoup with multithreading.
- Structure data into dictionaries and save to timestamped CSV using pandas.
- Quit WebDriver to close the browser.

## IMPLEMENTATION

- Implemented using Python with libraries such as Selenium, Requests, BeautifulSoup, and Pandas.
- Selenium automates Chrome to scrape job listings from Indeed.
- Job details are extracted (title, company, location, salary, rating, posting date, and URL).
- Requests + BeautifulSoup fetch detailed job descriptions.
- Data is structured and saved into CSV format for easy analysis.

## CONCLUSION

The Indeed Scraper successfully automates the process of collecting job postings, reducing manual effort. It provides a structured dataset useful for job seekers, recruiters, and researchers. By storing data in CSV format, it enables further analysis, visualization, and long-term tracking of job market trends.