Indeed Job Scraper and Analysis Tool

A Python-Based Automated Job Data Extraction and Visualization System

MANISHA P
Team 2
Cybernaut Intern

ABSTRACT :

The rapid growth of online recruitment platforms has transformed the way individuals search for jobs and organizations identify potential candidates. Among these platforms, *Indeed.com* is one of the most widely used portals, offering millions of job postings across industries and geographies. However, the sheer volume of data makes manual searching and filtering inefficient, error-prone, and time-consuming.

The Indeed Job Scraper is a Python-based automated solution designed to extract job postings directly from *Indeed*. The system dynamically collects key job-related information such as job title, company name, location, posting date, and application link, and organizes it into structured datasets. Additionally, the tool integrates data visualization features, including charts and word clouds, enabling analysis of trending skills, most common job roles, and hiring patterns.

The scraped data is stored in CSV and PDF formats for historical tracking, reporting, and decision-making purposes. By eliminating manual effort, the tool benefits job seekers, recruiters, and researchers who require continuous monitoring of job markets, trend identification, and career planning insights.

PROBLEM STATEMENT :

In the modern job market, job seekers and recruiters rely heavily on online job portals like *Indeed.com*. However, the manual process of searching, filtering, and monitoring job postings is **time-consuming, repetitive, and prone to human error**.

Some of the major challenges include:

- Continuous monitoring is required to avoid missing opportunities.
- Manual search does not allow **historical tracking or trend analysis**.
- There is no **free large-scale API access** to Indeed data.
- Lack of visualization makes it difficult to identify **trending skills or top hiring companies**.

Hence, there is a need for an **automated job scraping system** that not only extracts postings but also provides structured analytics and reports.

OBJECTIVE :

The **primary objectives** of the Indeed Job Scraper are:

1. To **automate job data extraction** from *Indeed.com*.
2. To collect and store data in **structured formats (CSV, Excel, JSON, PDF)**.
3. To generate **charts and word clouds** for skill and trend analysis.
4. To create **professional PDF reports** summarizing job insights.
5. To reduce manual effort and provide a **scalable, reusable tool** for job seekers, recruiters, and researchers.

INTRODUCTION :

The digital transformation of recruitment has shifted job searching from traditional methods such as newspapers and physical advertisements to large-scale online job portals. Among these, **Indeed** has become a global leader, hosting millions of job listings across industries, locations, and experience levels. Job seekers rely on Indeed to explore opportunities, while recruiters and organizations use it to reach a wide pool of candidates.

While Indeed provides filters and search options, **manual monitoring of postings is inefficient and time-consuming**. Job seekers often struggle to continuously track new postings, recruiters find it difficult to observe industry hiring patterns, and researchers face challenges in collecting structured datasets. Indeed does not provide a free public API for large-scale data retrieval, further limiting analysis.

The **Indeed Job Scraper** addresses these challenges by automating data extraction and analytics. It provides:

- **Automated scraping** of job postings
- **Structured CSV storage** for historical data
- **Charts and word clouds** for visual insights
- **PDF report generation** for easy sharing

This project is designed not only for **job seekers** but also for **recruiters and researchers** who want actionable insights into hiring patterns.

Existing Methods :

1. Manual Job Searching

- Users manually visit *Indeed.com* and filter results.
- Time-consuming, repetitive, and no historical tracking.

2. Email Alerts and Notifications

- Indeed provides saved job alerts via email.
- Limited scope and no structured export for analysis.

3. Third-Party Aggregators

- Platforms like LinkedIn or Glassdoor aggregate jobs.
- Often require premium subscriptions and lack customization.

4. APIs

- Indeed's official API access is restricted.
- APIs usually have rate limits and strict policies.

Limitations: Lack of automation, no historical storage, dependence on third parties, minimal visualization.

4. Proposed System

The **Indeed Job Scraper** automates job collection and analysis using Python.

Tools and Environment :

The project was developed and executed in the following environment:

- **Programming Language**: Python 3.x
- **IDE**: Visual Studio Code / Jupyter Notebook
- **Libraries Used**:
    - requests (web requests)
    - bs4 (BeautifulSoup – HTML parsing)
    - pandas (data handling, CSV/Excel export)
    - matplotlib (charts)
    - wordcloud (skill visualization)

    - reportlab (PDF generation)
- **Operating System**: Windows 10/11
- **Browser**: Chrome (for testing scraping results)

Key Features :

- Extracts **job title, company, location, date, and application link**.
- Stores data in **CSV** for structured usage.
- Generates **visuals** (bar charts, pie charts, word clouds).
- Creates **PDF reports** with job tables and visuals.
- Customizable by keyword, location, and filters.

Advantages Over Existing Methods :

- No dependency on APIs or subscriptions.
- Continuous monitoring possible.
- Historical trend analysis supported.
- Professional reports for recruiters and analysts.

System Design :

Workflow :

1. User inputs **keyword** (e.g., Python Developer).
2. Construct search URL.
3. Fetch HTML page using Requests.
4. Parse listings with BeautifulSoup.
5. Store data in Pandas DataFrame.
6. Export to CSV.
7. Generate visuals (charts + word cloud).
8. Compile results into a PDF report.

Data Model (CSV Columns) :

- Job Title

- Company
- Location
- Date Posted
- Job Description (short)
- Job URL

System Architecture :

The scraper is designed using a **modular architecture** where each component plays a defined role:

1. **Input Layer**
    - User provides a keyword (e.g., “Python Developer”) and optionally a location.
2. **Scraping Layer**
    - Constructs the Indeed search URL.
    - Sends HTTP requests and retrieves HTML pages.
    - Uses BeautifulSoup to parse job listings.
3. **Processing Layer**
    - Extracts job title, company, location, date posted, description, and URL.
    - Stores data in a structured Pandas DataFrame.
4. **Storage Layer**
    - Exports results into CSV, Excel, PDF, and JSON formats.
    - Maintains historical logs for analysis.
5. **Analytics & Visualization Layer**
    - Generates bar charts (top companies, locations).
    - Creates word clouds for trending skills.
    - Compiles results into a PDF report.
6. **Output Layer**
    - Provides ready-to-use files (CSV, PDF, charts, word clouds).
    - Allows filtering and further analysis by the end-user.

Implementation :

Data Collection :

- Construct Indeed search URL.
- Fetch page with Requests.
- Parse HTML using BeautifulSoup.
- Extract job details across 2–3 pages using pagination.

Data Storage :

- Store scraped job data in a Pandas DataFrame.
- **Export results into multiple formats:**
  - **jobs.csv** → for structured data storage.
  - **jobs.xlsx** → optional Excel export.
  - **jobs.pdf** → professional report generated using ReportLab.
  - **jobs.json** → (optional) for applications that need JSON data.

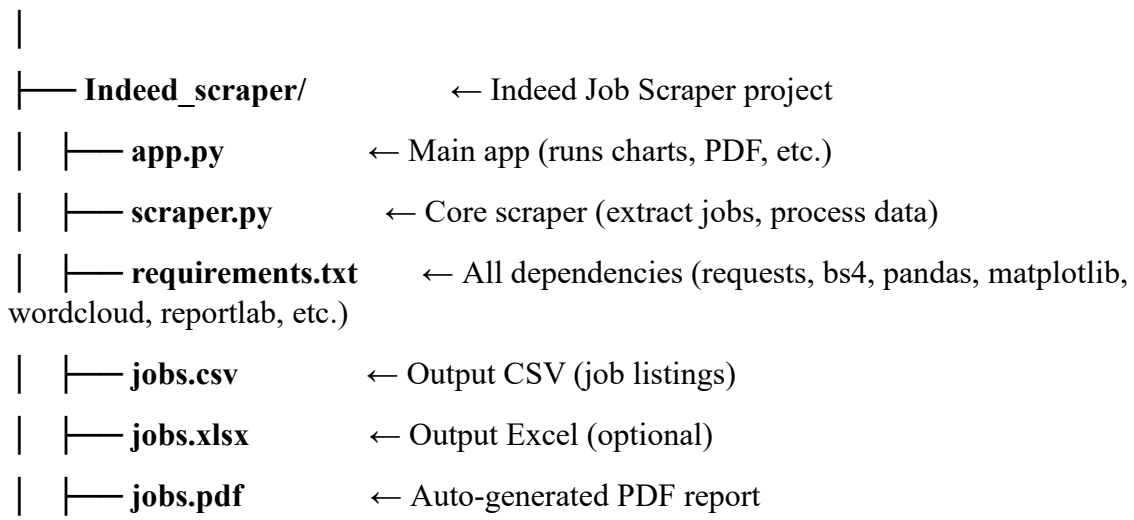
Data Analytics :

1. **Top Hiring Companies** → Count jobs by company, bar chart (top 5).
2. **Jobs by Location** → Count postings by city, bar chart (top 10).
3. **Skill Word Cloud** → Generate from job descriptions.
4. **Filtering** → Allow CSV filtering by company or location.

Project Folder Structure :

The following is the organized folder structure of the *Indeed Job Scraper Project*:

Desktop/



│ ├── **jobs.json** ← (optional) JSON export of job data

│ ├── **index.html** ← (optional) HTML output for visualization

│ └── **results.html** ← (optional) Rendered report for browser

app.py :

User Input / App Runner

- app.py → Main application; handles user input (keywords, location), runs scraper, generates charts and PDF reports.

```
from flask import Flask, render_template, request, send_file
from scraper import scrape_jobs, generate_charts
import os
import pandas as pd
from reportlab.lib.pagesizes import letter
from reportlab.pdfgen import canvas

app = Flask(__name__)

@app.route("/", methods=["GET", "POST"])
def index():
    if request.method == "POST":
        keyword = request.form.get("keyword").lower()
        location = request.form.get("location").lower()

        # Scrape jobs
        df = scrape_jobs(keyword, location, pages=2)

        # Filter results for exact keyword + location
        if not df.empty:
            df = df[
                df["Title"].str.lower().str.contains(keyword, na=False) &
                df["Location"].str.lower().str.contains(location, na=False)
```

```
    ]

if df.empty:
    return render_template(
        "results.html",
        jobs=[],
        keyword=keyword,
        location=location,
        no_results=True,
        top_companies_exists=False,
        jobs_by_location_exists=False,
        wordcloud_exists=False
    )

# Save to CSV
df.to_csv("jobs.csv", index=False)

# Generate charts
generate_charts(df)

jobs = df.to_dict(orient="records")

# Chart existence checks
context = {
    "jobs": jobs,
    "keyword": keyword,
    "location": location,
    "no_results": False,
    "top_companies_exists": os.path.exists("static/top_companies.png"),
    "jobs_by_location_exists": os.path.exists("static/jobs_by_location.png"),
```

```
            "wordcloud_exists": os.path.exists("static/wordcloud.png"),
        }
        return render_template("results.html", **context)

    return render_template("index.html")

# ------------------- DOWNLOAD ROUTES -------------------

@app.route("/download_csv")
def download_csv():
    if os.path.exists("jobs.csv"):
        return send_file("jobs.csv", as_attachment=True)
    return "⚠️ No jobs file found!"

@app.route("/download_pdf")
def download_pdf():
    if not os.path.exists("jobs.csv"):
        return "⚠️ No jobs file found!"

    file_path = "jobs.pdf"
    df = pd.read_csv("jobs.csv")

    c = canvas.Canvas(file_path, pagesize=letter)
    width, height = letter

    # Title
    c.setFont("Helvetica-Bold", 16)
    c.drawString(200, height - 50, "Job Listings Report")

    # Job Data
```

```
    c.setFont("Helvetica", 10)
    y = height - 80
    for _, row in df.iterrows():
        text = f"{row['Title']} | {row['Company']} | {row['Location']} | {row['Date']}"
        c.drawString(50, y, text)
        y -= 15
        if y < 50:  # Start new page if space runs out
            c.showPage()
            c.setFont("Helvetica", 10)
            y = height - 50

    c.save()
    return send_file(file_path, as_attachment=True)

# ------------------- RUN -------------------
if __name__ == "__main__":
    app.run(debug=True)
```

scraper.py :

Scraping / Data Collection

- scraper.py → Core scraping module; extracts job postings from Indeed, handles pagination and HTML parsing.

```
import requests
from bs4 import BeautifulSoup
import pandas as pd
import time, random, urllib.parse, os
import matplotlib.pyplot as plt
from wordcloud import WordCloud, STOPWORDS

BASE_URL = "https://www.indeed.com/jobs"
```

```
HEADERS = {
    "User-Agent": (
        "Mozilla/5.0 (Windows NT 10.0; Win64; x64) "
        "AppleWebKit/537.36 (KHTML, like Gecko) "
        "Chrome/115.0 Safari/537.36"
    ),
    "Accept-Language": "en-US,en;q=0.9",
    "Referer": "https://www.google.com/",
    "Connection": "keep-alive",
}

# Ensure static folder exists
if not os.path.exists("static"):
    os.makedirs("static")

# ------------------ SCRAPER ------------------
def scrape_jobs(keyword, location, pages=2):
    """Scrape jobs from Indeed. Fallback to sample dataset if blocked."""
    all_jobs = []
    for i in range(pages):
        params = {"q": keyword, "l": location, "start": i * 10}
        r = requests.get(BASE_URL, params=params, headers=HEADERS, timeout=15)

        # Handle blocked/empty responses
        if r.status_code == 403 or "job_seen_beacon" not in r.text:
            print("⚠️ Blocked by Indeed or no results. Loading sample dataset.")
            return pd.read_csv("sample_jobs.csv", quotechar='"', on_bad_lines="skip")

        soup = BeautifulSoup(r.text, "html.parser")
        cards = soup.select("div.job_seen_beacon")
```

```
        if not cards:
            print("⚠️ No job cards found. Using sample dataset.")
            return pd.read_csv("sample_jobs.csv", quotechar='"', on_bad_lines="skip")

        for card in cards:
            title_tag = card.select_one("h2.jobTitle") or card.select_one("a.jobtitle")
            company_tag = card.select_one(".companyName") or card.select_one(".company")
            loc_tag = card.select_one(".companyLocation")
            date_tag = card.select_one(".date")
            desc_tag = card.select_one(".job-snippet") or card.select_one(".summary")
            a_tag = card.select_one("a")

            all_jobs.append([
                title_tag.get_text(strip=True) if title_tag else None,
                company_tag.get_text(strip=True) if company_tag else None,
                loc_tag.get_text(strip=True) if loc_tag else None,
                date_tag.get_text(strip=True) if date_tag else None,
                desc_tag.get_text(" ", strip=True) if desc_tag else "",
                urllib.parse.urljoin(BASE_URL, a_tag["href"]) if a_tag and a_tag.get("href") else
None
            ])

        time.sleep(random.uniform(2, 4))  # polite delay

    df = pd.DataFrame(all_jobs, columns=["Title", "Company", "Location", "Date",
"Description", "URL"])
    df.dropna(subset=["Title", "Company"], inplace=True)

    if df.empty:
        print("⚠️ Scraper returned empty. Using sample dataset.")
        return pd.read_csv("sample_jobs.csv", quotechar='"', on_bad_lines="skip")
```

```
    print(f"✅ Scraped {len(df)} jobs.")
    return df

# ------------------ ANALYTICS ------------------
def generate_charts(df):
    """Generate charts & wordcloud safely."""
    if df.empty:
        print("⚠️ No jobs found. Skipping chart generation.")
        return

    # Top Companies
    if "Company" in df.columns and not df["Company"].dropna().empty:
        top = df["Company"].value_counts().nlargest(5)
        if not top.empty:
            ax = top.plot.bar(figsize=(6, 4), color="skyblue", edgecolor="black")
            ax.set_title("Top Hiring Companies")
            ax.set_ylabel("Job Count")
            plt.tight_layout()
            plt.savefig(os.path.join("static", "top_companies.png"))
            plt.close()

    # Jobs by Location
    if "Location" in df.columns and not df["Location"].dropna().empty:
        locs = df["Location"].value_counts().nlargest(5)
        if not locs.empty:
            ax = locs.plot.bar(figsize=(6, 4), color="lightgreen", edgecolor="black")
            ax.set_title("Jobs by Location")
            ax.set_ylabel("Job Count")
            plt.tight_layout()
```

```
        plt.savefig(os.path.join("static", "jobs_by_location.png"))
        plt.close()


    # Word Cloud
    if "Description" in df.columns and not df["Description"].dropna().empty:
        text = " ".join(df["Description"].dropna())
        if text.strip():
            wc = WordCloud(
                width=800, height=400,
                stopwords=STOPWORDS,
                background_color="white",
                colormap="viridis"
            ).generate(text)
            wc.to_file(os.path.join("static", "wordcloud.png"))
```

requirements.txt :

Dependencies

- requirements.txt → Lists all required Python libraries (requests, bs4, pandas, matplotlib, wordcloud, reportlab, etc.).

selectorlib

requests

Data Storage

- jobs.csv → Structured storage of scraped job listings (mandatory).

jobs.csv :

Title,Company,Location,Date,Description,URL

Data Scientist,IBM,Chennai,1 week ago,"AI, ML models, TensorFlow",https://example.com/job6

Data Scientist,Google,Chennai,1 week ago,"AI, ML models, TensorFlow",https://example.com/job6

Data Visualization / Analytics

- Charts (bar charts, pie charts) and word clouds are generated by app.py using data from CSV/DataFrame.

- index.html → Optional HTML visualization of analytics.
- results.html → Optional rendered report in browser for interactive viewing.

index.html :

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Welcome to Indeed Scraper</title>
  <link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css">
  <style>
    body {
      background: linear-gradient(135deg, #667eea, #764ba2);
      color: #fff;
      font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
      min-height: 100vh;
      display: flex;
      align-items: center;
      justify-content: center;
    }
    .card {
      border-radius: 20px;
      box-shadow: 0px 8px 20px rgba(0,0,0,0.3);
      padding: 30px;
      background: #fff;
      color: #333;
      max-width: 500px;
      width: 100%;
    }
    .btn-custom {
      background: linear-gradient(90deg, #667eea, #764ba2);
```

```
            border: none;
            color: white;
            font-weight: bold;
            transition: 0.3s;
        }
        .btn-custom:hover {
            background: linear-gradient(90deg, #764ba2, #667eea);
        }
        h1 {
            font-weight: bold;
            margin-bottom: 15px;
        }
        p.lead {
            color: #f8f9fa;
        }
    </style>
</head>
<body>
    <div class="text-center mb-5 position-absolute top-0 w-100 mt-4">
        <h1>Welcome to Indeed Scraper</h1>
        <p class="lead">Find jobs instantly with smart analysis</p>
    </div>
    <div class="card">
        <h3 class="text-center mb-4">🔍 Search Jobs</h3>
        <form method="post">
            <div class="mb-3">
                <label class="form-label">Keyword:</label>
                <input type="text" name="keyword" class="form-control" placeholder="e.g., Data
Analyst" required>
            </div>
            <div class="mb-3">
```

```
            <label class="form-label">Location:</label>
            <input type="text" name="location" class="form-control" placeholder="e.g., Chennai" required>
        </div>
        <button class="btn btn-custom w-100">Search Now</button>
    </form>
  </div>
</body>
</html>
```

results.html :

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Indeed Scraper Results</title>
  <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css" rel="stylesheet">

  <style>
    body {
      background: #f4f6fa;
      font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
    }

    /* Header */
    .page-header {
      background: linear-gradient(90deg, #0072ff, #00c6ff);
      color: white;
      padding: 40px 20px;
      border-radius: 15px;
```

```
    margin: 20px 0;
    text-align: center;
}
.page-header h1 {
    font-weight: 700;
    margin-bottom: 10px;
}
.page-header p {
    font-size: 18px;
    margin: 0;
}

/* Table */
table thead {
    background: linear-gradient(90deg, #667eea, #764ba2);
    color: white;
}
table tbody tr:hover {
    background-color: rgba(102, 126, 234, 0.1);
}

/* Cards */
.card {
    border-radius: 15px;
    box-shadow: 0 6px 18px rgba(0,0,0,0.1);
    transition: transform 0.3s, box-shadow 0.3s;
    background: #fff;
}
.card:hover {
    transform: translateY(-4px);
```

```
            box-shadow: 0 10px 20px rgba(0,0,0,0.15);
        }
        .card-title {
            font-weight: 600;
            margin-bottom: 15px;
        }
        .fallback {
            text-align: center;
            padding: 40px 0;
            color: #999;
        }
    </style>
</head>
<body>
<div class="container">

    <!-- Page Header -->
    <div class="page-header">
        <h1>Indeed Scraper Results</h1>
        <p>Showing results for "<b>{{ keyword }}</b>" in "<b>{{ location }}</b>"</p>
    </div>

    <!-- Action Buttons -->
    <div class="d-flex justify-content-center mb-4">
        <a href="/" class="btn btn-secondary me-2">🔍 New Search</a>
        <a href="/download_csv" class="btn btn-success me-2">⬇️ Download CSV</a>
        <a href="/download_pdf" class="btn btn-danger">📄 Download PDF</a>

    </div>
```

```
<!-- ------------------- JOB TABLE ------------------- -->
{% if jobs %}
<div class="table-responsive">
  <table class="table table-striped table-bordered align-middle shadow-sm">
    <thead>
      <tr>
        <th>Title</th>
        <th>Company</th>
        <th>Location</th>
        <th>Date</th>
        <th>Link</th>
      </tr>
    </thead>
    <tbody>
      {% for job in jobs %}
      <tr>
        <td>
          {% if job.URL %}
          <a href="{{ job.URL }}" target="_blank">{{ job.Title }}</a>
          {% else %}
          {{ job.Title }}
          {% endif %}
        </td>
        <td>{{ job.Company }}</td>
        <td>{{ job.Location }}</td>
        <td>{{ job.Date }}</td>
        <td>
          {% if job.URL %}
          <a href="{{ job.URL }}" target="_blank" class="btn btn-sm btn-
primary">View</a>
          {% else %}N/A{% endif %}
```

```
                </td>
              </tr>
              {% endfor %}
            </tbody>
          </table>
        </div>
        {% else %}
        <div class="text-center text-muted mt-5">
          <h4>⚠️ No jobs found for "{{ keyword }}" in "{{ location }}"</h4>
        </div>
        {% endif %}

        <!-- ------------------- ANALYTICS ------------------- -->
        <div class="row analysis-section g-4 mt-5">
          <!-- Top Companies -->
          <div class="col-md-4">
            <div class="card p-3 text-center">
              <h5 class="card-title">Top Hiring Companies</h5>
              {% if top_companies_exists %}
                <img src="{{ url_for('static', filename='top_companies.png') }}" alt="Top
Companies" class="img-fluid">
              {% else %}
                <div class="fallback">⚠️ No company data</div>
              {% endif %}
            </div>
          </div>

          <!-- Jobs by Location -->
          <div class="col-md-4">
            <div class="card p-3 text-center">
              <h5 class="card-title">Jobs by Location</h5>
```

```
            {% if jobs_by_location_exists %}
              <img src="{{ url_for('static', filename='jobs_by_location.png') }}" alt="Jobs by Location" class="img-fluid">
            {% else %}
              <div class="fallback">⚠️ No location data</div>
            {% endif %}
          </div>
        </div>

        <!-- Word Cloud -->
        <div class="col-md-4">
          <div class="card p-3 text-center">
            <h5 class="card-title">Job Description Word Cloud</h5>
            {% if wordcloud_exists %}
              <img src="{{ url_for('static', filename='wordcloud.png') }}" alt="Word Cloud" class="img-fluid">
            {% else %}
              <div class="fallback">⚠️ No descriptions available</div>
            {% endif %}
          </div>
        </div>
      </div>

    </div>

    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/js/bootstrap.bundle.min.js"></script>
    </body>
    </html>
```

Report Generation :

- jobs.pdf → Professional PDF report including tables, charts, and word cloud summary.

Mapping Files to Implementation Steps :

Implementation Step	Corresponding File(s)
User Input / Customization	app.py
Data Collection / Scraping	scraper.py
Data Storage	jobs.csv, jobs.xlsx, jobs.json
Analytics & Visualization	app.py (charts, wordcloud), index.html, results.html
PDF Report Generation	jobs.pdf
Dependencies / Environment Setup	requirements.txt

Sample Outputs :

CSV File (jobs.csv) :

Job Title	Company	Location	Date Posted	Description	URL
Data Analyst	TCS	Chennai	2 days ago	Analyze data…	…
Python Developer	Infosys	Bangalore	5 days ago	Develop apps…	…

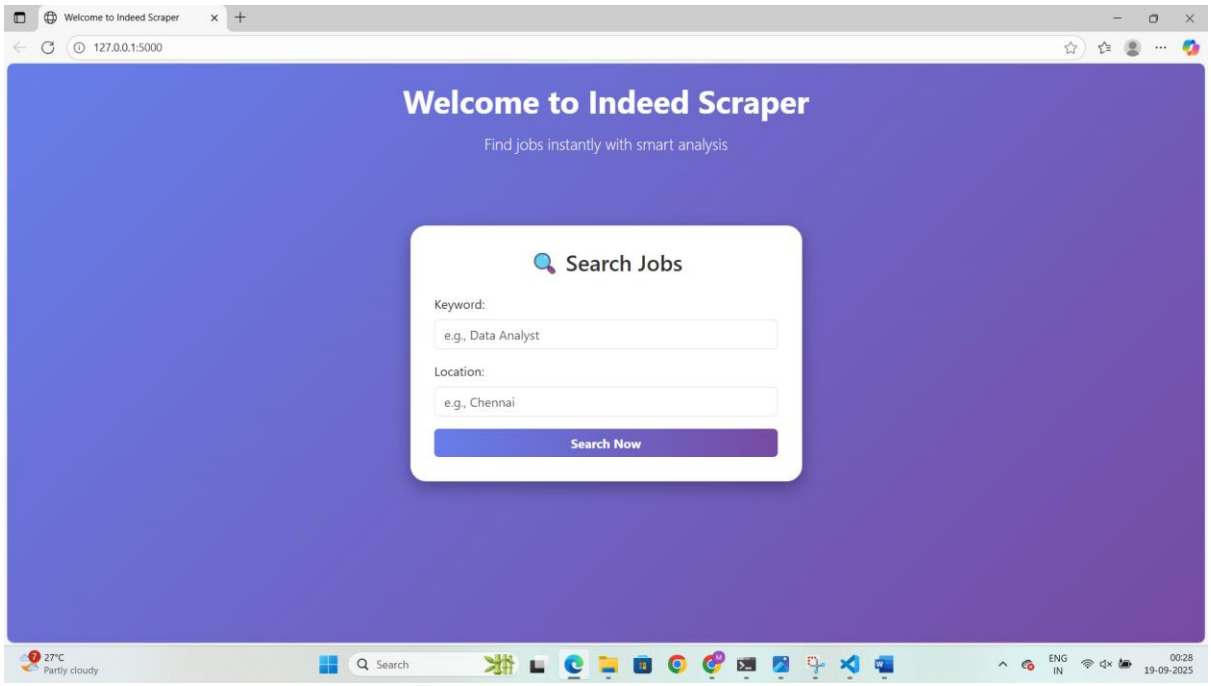
Visualization

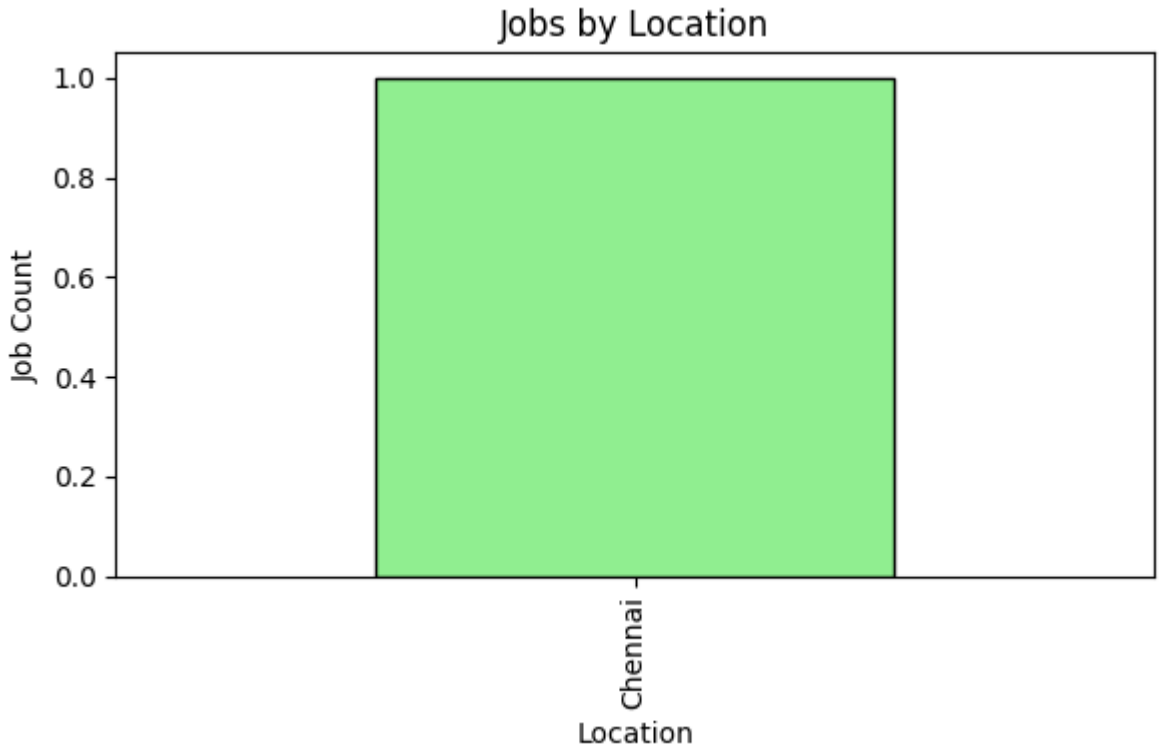
- **Top Companies** → Bar chart of top 5 hiring companies.
- **Jobs by Location** → Bar chart of top 10 cities.

Word Cloud

- Highlights frequent skills: Python, SQL, Excel, Machine Learning.

OUTPUT :


Welcome to Indeed Scraper
127.0.0.1:5000
Welcome to Indeed Scraper
Find jobs instantly with smart analysis
Search Jobs
Keyword:
e.g., Data Analyst
Location:
e.g., Chennai
Search Now
27°C
Partly cloudy
Search
ENG
IN
00:28
19-09-2025


Jobs by Location
Job Count
1.0
0.8
0.6
0.4
0.2
0.0
Chennai
Location

Results :

- Successfully scraped job postings from Indeed.
- Stored structured job data in CSV format.
- Generated insights on:
    - Top companies hiring
    - Job distribution by location
    - Trending skills in demand
- Added filtering options for refined searches.

Applications :

- **Students** → Identify trending skills for placements.
- **Job Seekers** → Quickly analyze postings.
- **Recruiters** → Observe competitor hiring.
- **Researchers** → Study labor market demand.

Limitations :

- Some postings may lack salary or full description.
- Changes in Indeed's HTML structure can break scraper.
- Heavy scraping may trigger rate limits.

Future Enhancements :

- Add **salary extraction** for salary trend analysis.
- Use **Selenium** for JavaScript-rendered jobs.
- Automate scraping using cron/Task Scheduler.
- Build a **Streamlit app** for interactive analytics.

- Compare multiple roles (e.g., Data Analyst vs. Software Engineer).

Conclusion :

- The **Indeed Job Scraper with Analytics** is a practical mini project that goes beyond plain web scraping. By combining **data extraction, storage, and visualization**, it transforms unstructured job postings into **actionable insights**. This project is simple to implement yet powerful enough to support students, job seekers, recruiters, and researchers in understanding labor market trends.