SPRINT - 2

DATE	05 November 2022
Team ID	PNT2022TMID50144
Project Name	Project-Skill and job recommender

CODE:
app.py

```
from __future__ import print_function  # In python 2.
import sys
import sqlite3
from flask import Flask, request, g, redirect, url_for, render_template
import indeed as ind
import JaccardSimilarity as mal
import linkedin as lik

app = Flask(__name__.split('.')[0])   # create the application instance


def connect_db():
    """Connect to the specific database."""
    rv = sqlite3.connect('linked_deed.db')
    rv.row_factory = sqlite3.Row
    return rv


def get_db():
    """
    Open a new database connection.

    If there is none yet for the
    current application context.
    """
    if not hasattr(g, 'sqlite_db'):
        g.sqlite_db = connect_db()
    return g.sqlite_db


@app.teardown_appcontext
def close_db(error):
    """Close the database again at the end of the request."""
    if hasattr(g, 'sqlite_db'):
        g.sqlite_db.close()


@app.route('/')
def show_entries():
    db = get_db()
    cur = db.execute('SELECT DISTINCT WHAT, URL, DESCRIPTION FROM jobs_indeed
order by ACCURACY desc')
    entries = cur.fetchall()
    return render_template('jobs.html', entries=entries)


@app.route('/index', methods=['POST'])
def login():
    url = request.form['url']
    em = request.form['email']
    pwd = request.form['password']
    lik.main(em, pwd, url)
    db = get_db()
    cur = db.execute('SELECT LOCATION, DESIGNATIONS FROM linkedin_skills WHERE
ID=1')
    entry = cur.fetchone()
    # print(entry[1].split(", "), file=sys.stderr)
    return render_template('form.html', jobs=entry[1].split(", "),
location=entry[0].split(", "))


@app.route('/form', methods=['POST'])
def form():
    what = request.form['what']
    city = request.form['city']
    state = request.form['state']
    location = city + ", " + state
    db = get_db()
    db.execute('UPDATE linkedin_skills SET DESIGNATIONS = ? WHERE ID = 1',
(what, ))
    db.commit()
    db.execute('UPDATE linkedin_skills SET LOCATION = ? WHERE ID = 1',
(location, ))
    db.commit()
    ind.indeed_jobs(job=what, location=location)
    mal.main()
    return redirect(url_for('show_entries'))


if __name__ == '__main__':
    app.run(debug=True)
from __future__ import print_function  # In python 2.
import sys
import sqlite3
from flask import Flask, request, g, redirect, url_for, render_template
import indeed as ind
import JaccardSimilarity as mal
import linkedin as lik

app = Flask(__name__.split('.')[0])   # create the application instance


def connect_db():
```

```python
    """Connect to the specific database."""
    rv = sqlite3.connect('linked_deed.db')
    rv.row_factory = sqlite3.Row
    return rv


def get_db():
    """
    Open a new database connection.

    If there is none yet for the
    current application context.
    """
    if not hasattr(g, 'sqlite_db'):
        g.sqlite_db = connect_db()
    return g.sqlite_db


@app.teardown_appcontext
def close_db(error):
    """Close the database again at the end of the request."""
    if hasattr(g, 'sqlite_db'):
        g.sqlite_db.close()


@app.route('/')
def show_entries():
    db = get_db()
    cur = db.execute('SELECT DISTINCT WHAT, URL, DESCRIPTION FROM jobs_indeed
order by ACCURACY desc')
    entries = cur.fetchall()
    return render_template('jobs.html', entries=entries)


@app.route('/index', methods=['POST'])
def login():
    url = request.form['url']
    em = request.form['email']
    pwd = request.form['password']
    lik.main(em, pwd, url)
    db = get_db()
    cur = db.execute('SELECT LOCATION, DESIGNATIONS FROM linkedin_skills WHERE
ID=1')
    entry = cur.fetchone()
    # print(entry[1].split(", "), file=sys.stderr)
    return render_template('form.html', jobs=entry[1].split(", "),
location=entry[0].split(", "))


@app.route('/form', methods=['POST'])
def form():
    what = request.form['what']
    city = request.form['city']
    state = request.form['state']
    location = city + ", " + state
    db = get_db()
    db.execute('UPDATE linkedin_skills SET DESIGNATIONS = ? WHERE ID = 1',
(what, ))
    db.commit()
    db.execute('UPDATE linkedin_skills SET LOCATION = ? WHERE ID = 1',
(location, ))
    db.commit()
    ind.indeed_jobs(job=what, location=location)
    mal.main()
    return redirect(url_for('show_entries'))


if __name__ == '__main__':
    app.run(debug=True)
from __future__ import print_function  # In python 2.
import sys
import sqlite3
from flask import Flask, request, g, redirect, url_for, render_template
import indeed as ind
import JaccardSimilarity as mal
import linkedin as lik

app = Flask(__name__.split('.')[0])   # create the application instance


def connect_db():
    """Connect to the specific database."""
    rv = sqlite3.connect('linked_deed.db')
    rv.row_factory = sqlite3.Row
    return rv


def get_db():
    """
    Open a new database connection.

    If there is none yet for the
    current application context.
    """
    if not hasattr(g, 'sqlite_db'):
        g.sqlite_db = connect_db()
    return g.sqlite_db


@app.teardown_appcontext
def close_db(error):
    """Close the database again at the end of the request."""
    if hasattr(g, 'sqlite_db'):
        g.sqlite_db.close()
```

```
@app.route('/')
def show_entries():
    db = get_db()
    cur = db.execute('SELECT DISTINCT WHAT, URL, DESCRIPTION FROM jobs_indeed
order by ACCURACY desc')
    entries = cur.fetchall()
    return render_template('jobs.html', entries=entries)


@app.route('/index', methods=['POST'])
def login():
    url = request.form['url']
    em = request.form['email']
    pwd = request.form['password']
    lik.main(em, pwd, url)
    db = get_db()
    cur = db.execute('SELECT LOCATION, DESIGNATIONS FROM linkedin_skills WHERE
ID=1')
    entry = cur.fetchone()
    # print(entry[1].split(", "), file=sys.stderr)
    return render_template('form.html', jobs=entry[1].split(", "),
location=entry[0].split(", "))


@app.route('/form', methods=['POST'])
def form():
    what = request.form['what']
    city = request.form['city']
    state = request.form['state']
    location = city + ", " + state
    db = get_db()
    db.execute('UPDATE linkedin_skills SET DESIGNATIONS = ? WHERE ID = 1',
(what, ))
    db.commit()
    db.execute('UPDATE linkedin_skills SET LOCATION = ? WHERE ID = 1',
(location, ))
    db.commit()
    ind.indeed_jobs(job=what, location=location)
    mal.main()
    return redirect(url_for('show_entries'))


if __name__ == '__main__':
    app.run(debug=True)
```

Indeed.py

```
from bs4 import BeautifulSoup, SoupStrainer # For HTML parsing
import urllib2 # Website connections
import re # Regular expressions
from time import sleep # To prevent overwhelming the server between connections
import pandas as pd # For converting results to a dataframe and bar chart plots
import sqlite3
import unicodedata

job = "software developer"
location = "Victoria, BC, Canada"

def job_extractor(website):
    '''
    Cleans up the raw HTML
    '''
    company_name = "Unknown"
    try:
        site = urllib2.urlopen(website).read() # Connect to the job posting
    except:
        return   # Need this in case the website isn't there anymore or some
other weird connection problem

    soup_obj = BeautifulSoup(site,"html.parser") # Get the html from the site

    for script in soup_obj(["script", "style"]):
        script.extract() # Remove these two elements from the BS4 object

    company = soup_obj.find('span', {'class' : 'company'})
    if company:
        company_name = company.get_text()
    # print company_name

    content = soup_obj.get_text() # Get the text from this

    return content

def clean_up(dirty_data):
    """Clean up the dirty data."""
    clean_data = []

    for i in range(len(dirty_data)):
        temp = unicodedata.normalize('NFKD',
"".join(dirty_data[i]).replace('\n', ' ').
                                     replace('\[[0-9]*\]', "").replace(' +', '
').replace('\t', "").
                                     replace('\r', ' ')).encode('ascii',
'ignore')
        temp, sep, tail = temp.partition('Apply')
        clean_data.append(temp)

    return clean_data

def indeed_jobs(final_job =None, city = None, state = None):
    '''
    Inputs the location's city and state and then looks for the job in
Indeed.com
    '''
    # searching for data scientist exact fit("data scientist" on Indeed search)

    #Create the proper indeed url based on the search criteria
    # Make sure the city specified works properly if it has more than one word
(such as San Francisco)
    if final_job is not None:
        if city is not None:
            final_city = city.split()
            final_city = '+'.join(word for word in final_city)
            final_site_list = 'http://www.indeed.ca/jobs?q=' + final_job + '&l='
+ final_city + '%2C+' + state # Join all of our strings together so that indeed
will search correctly
        else:
            final_site_list = ['http://www.indeed.ca/jobs?q="', final_job, '"']
    else:
        if city is not None:
            final_city = city.split()
            final_city = '+'.join(word for word in final_city)
            final_site_list = 'http://www.indeed.ca/jobs?q=&l=' + final_city +
'%2C+' + state # Join all of our strings together so that indeed will search
correctly
        else:
            final_site_list = 'https://ca.indeed.com/jobs?q=&l=Nationwide'



    final_site = ''.join(final_site_list) # Merge the html address together into
one string


    base_url = 'https://www.indeed.ca/'

    print final_site_list


    try:
        html = urllib2.urlopen(final_site).read() # Open up the front page of
our search first
    except:
        'That city/state combination did not have any jobs. Exiting . . .' # In
case the city is invalid
        return
    soup = BeautifulSoup(html,"html.parser") # Get the html from the first page

    # Now find out how many jobs there were
    div = soup.find(id = 'searchCount')

    num_jobs_area = soup.find(id = 'searchCount').string.encode('utf-8') # Now
extract the total number of jobs found
                                                                        # The
'searchCount' object has this
```

```
    job_numbers = re.findall('\d+', num_jobs_area) # Extract the total jobs 
found from the search result

    # print job_numbers
    if len(job_numbers) > 3: # Have a total number of jobs greater than 1000
        total_num_jobs = (int(job_numbers[2])*1000) + int(job_numbers[3])
    else:
        total_num_jobs = int(job_numbers[2])

    city_title = city
    if city is None:
        city_title = 'Nationwide'

    print 'There were', total_num_jobs, 'jobs found,', city_title # Display how 
many jobs were found

    num_pages = total_num_jobs/10 # This will be how we know the number of times 
we need to iterate over each new
                                      # search result page
    job_descriptions = [] # Store all our descriptions in this list
    final_URLs = []       # Store final job URLs in this list


    for i in xrange(0,num_pages): # Loop through all of our search result pages
        print 'Getting page', i
        start_num = str(i*1) # Assign the multiplier of 10 to view the pages we 
want
        current_page = ''.join([final_site, '&start=', start_num])
        print "So that's the current page"
        print current_page
        # Now that we can view the correct 10 job returns, start collecting the 
text samples from each

        html_page = urllib2.urlopen(current_page).read() # Get the page

        page_obj = BeautifulSoup(html_page,"lxml") # Locate all of the job links
        for script in page_obj(["script", "style"]):
            script.extract()

        print "And that's the result col"
        job_link_area = page_obj.find(id = 'resultsCol') # The center column on 
the page where the job postings exist

        job_URLS = [base_url + link.get('href') for link in 
job_link_area.find_all('a', href=True)]

        job_URLS = filter(lambda x:'clk' in x, job_URLS)

        for j in xrange(0,len(job_URLS)):
            final_description = job_extractor(job_URLS[j])
            if final_description: # So that we only append when the website was 
accessed correctly
```

```python
                job_descriptions.append(final_description)
                final_URLs.append(job_URLS[j])

        new_job_desc = clean_up(job_descriptions)

        # code for database connectivity
        conn = sqlite3.connect('linkedeed.db')
        c = conn.cursor()
        c.execute('DROP TABLE IF EXISTS indeed_jobs')
        c.execute('CREATE TABLE indeed_jobs(ID INTEGER PRIMARY KEY
AUTOINCREMENT, WHAT TEXT, URL TEXT, DESCRIPTION TEXT)')
        for i in range(len(job_descriptions)):
            c.execute('INSERT INTO indeed_jobs (ID, WHAT, URL, DESCRIPTION)
VALUES(?, ?, ?, ?)', ((i + 1), final_job.replace("+", " "), final_URLs[i],
job_descriptions[i], ))
            conn.commit()
        c.close()
        conn.close()

        print 'Done with collecting the job postings!'
        print 'There were', len(job_descriptions), 'jobs successfully found.'


location_list = location.split(",", 1)
city = location_list[0]
state = location_list[1].strip()
final_job = job.lower().replace(' ', '+')

indeed_jobs(final_job = final_job, city = city, state = state)
from bs4 import BeautifulSoup, SoupStrainer # For HTML parsing
import urllib2 # Website connections
import re # Regular expressions
from time import sleep # To prevent overwhelming the server between connections
import pandas as pd # For converting results to a dataframe and bar chart plots
import sqlite3
import unicodedata

job = "software developer"
location = "Victoria, BC, Canada"

def job_extractor(website):
    '''
    Cleans up the raw HTML
    '''
    company_name = "Unknown"
    try:
        site = urllib2.urlopen(website).read() # Connect to the job posting
    except:
        return   # Need this in case the website isn't there anymore or some
other weird connection problem

    soup_obj = BeautifulSoup(site,"html.parser") # Get the html from the site
    for script in soup_obj(["script", "style"]):
        script.extract() # Remove these two elements from the BS4 object

    company = soup_obj.find('span', {'class' : 'company'})
    if company:
        company_name = company.get_text()
    # print company_name

    content = soup_obj.get_text() # Get the text from this

    return content

def clean_up(dirty_data):
    """Clean up the dirty data."""
    clean_data = []

    for i in range(len(dirty_data)):
        temp = unicodedata.normalize('NFKD',
"".join(dirty_data[i]).replace('\n', ' ').
                                     replace('\[[0-9]*\]', "").replace(' +', '
').replace('\t', "").
                                     replace('\r', ' ')).encode('ascii',
'ignore')
        temp, sep, tail = temp.partition('Apply')
        clean_data.append(temp)

    return clean_data

def indeed_jobs(final_job =None, city = None, state = None):
    '''
    Inputs the location's city and state and then looks for the job in
Indeed.com
    '''
    # searching for data scientist exact fit("data scientist" on Indeed search)

    #Create the proper indeed url based on the search criteria
    # Make sure the city specified works properly if it has more than one word
(such as San Francisco)
    if final_job is not None:
        if city is not None:
            final_city = city.split()
            final_city = '+'.join(word for word in final_city)
            final_site_list = 'http://www.indeed.ca/jobs?q=' + final_job + '&l='
+ final_city + '%2C+' + state # Join all of our strings together so that indeed
will search correctly
        else:
            final_site_list = ['http://www.indeed.ca/jobs?q="', final_job, '"']
    else:
        if city is not None:
            final_city = city.split()
            final_city = '+'.join(word for word in final_city)
            final_site_list = 'http://www.indeed.ca/jobs?q=&l=' + final_city +
'%2C+' + state # Join all of our strings together so that indeed will search
correctly
        else:
            final_site_list = 'https://ca.indeed.com/jobs?q=&l=Nationwide'



    final_site = ''.join(final_site_list) # Merge the html address together into
one string


    base_url = 'https://www.indeed.ca/'

    print final_site_list


    try:
        html = urllib2.urlopen(final_site).read() # Open up the front page of
our search first
    except:
        'That city/state combination did not have any jobs. Exiting . . .' # In
case the city is invalid
        return
    soup = BeautifulSoup(html,"html.parser") # Get the html from the first page

    # Now find out how many jobs there were
    div = soup.find(id = 'searchCount')

    num_jobs_area = soup.find(id = 'searchCount').string.encode('utf-8') # Now
extract the total number of jobs found
                                                                        # The
'searchCount' object has this

    job_numbers = re.findall('\d+', num_jobs_area) # Extract the total jobs
found from the search result

    # print job_numbers
    if len(job_numbers) > 3: # Have a total number of jobs greater than 1000
        total_num_jobs = (int(job_numbers[2])*1000) + int(job_numbers[3])
    else:
        total_num_jobs = int(job_numbers[2])

    city_title = city
    if city is None:
        city_title = 'Nationwide'

    print 'There were', total_num_jobs, 'jobs found,', city_title # Display how
many jobs were found

    num_pages = total_num_jobs/10 # This will be how we know the number of times
we need to iterate over each new
                                      # search result page
    job_descriptions = [] # Store all our descriptions in this list
    final_URLs = []       # Store final job URLs in this list


    for i in xrange(0,num_pages): # Loop through all of our search result pages
        print 'Getting page', i
        start_num = str(i*1) # Assign the multiplier of 10 to view the pages we
want
        current_page = ''.join([final_site, '&start=', start_num])
        print "So that's the current page"
        print current_page
        # Now that we can view the correct 10 job returns, start collecting the
text samples from each

        html_page = urllib2.urlopen(current_page).read() # Get the page

        page_obj = BeautifulSoup(html_page,"lxml") # Locate all of the job links
        for script in page_obj(["script", "style"]):
            script.extract()

        print "And that's the result col"
        job_link_area = page_obj.find(id = 'resultsCol') # The center column on
the page where the job postings exist

        job_URLS = [base_url + link.get('href') for link in
job_link_area.find_all('a', href=True)]

        job_URLS = filter(lambda x:'clk' in x, job_URLS)

        for j in xrange(0,len(job_URLS)):
            final_description = job_extractor(job_URLS[j])
            if final_description: # So that we only append when the website was
accessed correctly
                job_descriptions.append(final_description)
                final_URLs.append(job_URLS[j])

        new_job_desc = clean_up(job_descriptions)

        # code for database connectivity
        conn = sqlite3.connect('linkedeed.db')
        c = conn.cursor()
        c.execute('DROP TABLE IF EXISTS indeed_jobs')
        c.execute('CREATE TABLE indeed_jobs(ID INTEGER PRIMARY KEY
AUTOINCREMENT, WHAT TEXT, URL TEXT, DESCRIPTION TEXT)')
        for i in range(len(job_descriptions)):
            c.execute('INSERT INTO indeed_jobs (ID, WHAT, URL, DESCRIPTION)
VALUES(?, ?, ?, ?)', ((i + 1), final_job.replace("+", " "), final_URLs[i],
job_descriptions[i], ))
            conn.commit()
        c.close()
        conn.close()
        print 'Done with collecting the job postings!'
        print 'There were', len(job_descriptions), 'jobs successfully found.'


location_list = location.split(",", 1)
city = location_list[0]
state = location_list[1].strip()
final_job = job.lower().replace(' ', '+')

indeed_jobs(final_job = final_job, city = city, state = state)
from bs4 import BeautifulSoup, SoupStrainer # For HTML parsing
import urllib2 # Website connections
import re # Regular expressions
from time import sleep # To prevent overwhelming the server between connections
import pandas as pd # For converting results to a dataframe and bar chart plots
import sqlite3
import unicodedata

job = "software developer"
location = "Victoria, BC, Canada"

def job_extractor(website):
    '''
    Cleans up the raw HTML
    '''
    company_name = "Unknown"
    try:
        site = urllib2.urlopen(website).read() # Connect to the job posting
    except:
        return   # Need this in case the website isn't there anymore or some
other weird connection problem

    soup_obj = BeautifulSoup(site,"html.parser") # Get the html from the site

    for script in soup_obj(["script", "style"]):
        script.extract() # Remove these two elements from the BS4 object

    company = soup_obj.find('span', {'class' : 'company'})
    if company:
        company_name = company.get_text()
    # print company_name

    content = soup_obj.get_text() # Get the text from this

    return content

def clean_up(dirty_data):
    """Clean up the dirty data."""
    clean_data = []

    for i in range(len(dirty_data)):
        temp = unicodedata.normalize('NFKD',
"".join(dirty_data[i]).replace('\n', ' ').
```

```
                                     replace('\[[0-9]*\]', "").replace(' +', '
').replace('\t', "").
                                     replace('\r', ' ')).encode('ascii',
'ignore')
        temp, sep, tail = temp.partition('Apply')
        clean_data.append(temp)

    return clean_data

def indeed_jobs(final_job =None, city = None, state = None):
    '''
    Inputs the location's city and state and then looks for the job in
Indeed.com
    '''
    # searching for data scientist exact fit("data scientist" on Indeed search)

    #Create the proper indeed url based on the search criteria
    # Make sure the city specified works properly if it has more than one word
(such as San Francisco)
    if final_job is not None:
        if city is not None:
            final_city = city.split()
            final_city = '+'.join(word for word in final_city)
            final_site_list = 'http://www.indeed.ca/jobs?q=' + final_job + '&l='
+ final_city + '%2C+' + state # Join all of our strings together so that indeed
will search correctly
        else:
            final_site_list = ['http://www.indeed.ca/jobs?q="', final_job, '"']
    else:
        if city is not None:
            final_city = city.split()
            final_city = '+'.join(word for word in final_city)
            final_site_list = 'http://www.indeed.ca/jobs?q=&l=' + final_city +
'%2C+' + state # Join all of our strings together so that indeed will search
correctly
        else:
            final_site_list = 'https://ca.indeed.com/jobs?q=&l=Nationwide'



    final_site = ''.join(final_site_list) # Merge the html address together into
one string


    base_url = 'https://www.indeed.ca/'

    print final_site_list


    try:
        html = urllib2.urlopen(final_site).read() # Open up the front page of
our search first
    except:
        'That city/state combination did not have any jobs. Exiting . . .' # In
case the city is invalid
        return
    soup = BeautifulSoup(html,"html.parser") # Get the html from the first page

    # Now find out how many jobs there were
    div = soup.find(id = 'searchCount')

    num_jobs_area = soup.find(id = 'searchCount').string.encode('utf-8') # Now
extract the total number of jobs found
                                                                         # The
'searchCount' object has this

    job_numbers = re.findall('\d+', num_jobs_area) # Extract the total jobs
found from the search result

    # print job_numbers
    if len(job_numbers) > 3: # Have a total number of jobs greater than 1000
        total_num_jobs = (int(job_numbers[2])*1000) + int(job_numbers[3])
    else:
        total_num_jobs = int(job_numbers[2])

    city_title = city
    if city is None:
        city_title = 'Nationwide'

    print 'There were', total_num_jobs, 'jobs found,', city_title # Display how
many jobs were found

    num_pages = total_num_jobs/10 # This will be how we know the number of times
we need to iterate over each new
                                                # search result page
    job_descriptions = [] # Store all our descriptions in this list
    final_URLs = []       # Store final job URLs in this list


    for i in xrange(0,num_pages): # Loop through all of our search result pages
        print 'Getting page', i
        start_num = str(i*1) # Assign the multiplier of 10 to view the pages we
want
        current_page = ''.join([final_site, '&start=', start_num])
        print "So that's the current page"
        print current_page
        # Now that we can view the correct 10 job returns, start collecting the
text samples from each

        html_page = urllib2.urlopen(current_page).read() # Get the page

        page_obj = BeautifulSoup(html_page,"lxml") # Locate all of the job links
        for script in page_obj(["script", "style"]):
            script.extract()
        print "And that's the result col"
        job_link_area = page_obj.find(id = 'resultsCol') # The center column on
the page where the job postings exist

        job_URLS = [base_url + link.get('href') for link in
job_link_area.find_all('a', href=True)]

        job_URLS = filter(lambda x:'clk' in x, job_URLS)

        for j in xrange(0,len(job_URLS)):
            final_description = job_extractor(job_URLS[j])
            if final_description: # So that we only append when the website was
accessed correctly
                job_descriptions.append(final_description)
                final_URLs.append(job_URLS[j])

        new_job_desc = clean_up(job_descriptions)

        # code for database connectivity
        conn = sqlite3.connect('linkedeed.db')
        c = conn.cursor()
        c.execute('DROP TABLE IF EXISTS indeed_jobs')
        c.execute('CREATE TABLE indeed_jobs(ID INTEGER PRIMARY KEY
AUTOINCREMENT, WHAT TEXT, URL TEXT, DESCRIPTION TEXT)')
        for i in range(len(job_descriptions)):
            c.execute('INSERT INTO indeed_jobs (ID, WHAT, URL, DESCRIPTION)
VALUES(?, ?, ?, ?)', ((i + 1), final_job.replace("+", " "), final_URLs[i],
job_descriptions[i], ))
            conn.commit()
        c.close()
        conn.close()

        print 'Done with collecting the job postings!'
        print 'There were', len(job_descriptions), 'jobs successfully found.'


location_list = location.split(",", 1)
city = location_list[0]
state = location_list[1].strip()
final_job = job.lower().replace(' ', '+')

indeed_jobs(final_job = final_job, city = city, state = state)
```

jaccaradsimilarity.py

```
"""Implementing an algorithm to find out the best match jobs."""
import csv
import sqlite3
import unicodedata

global total_count

text = []
```

```
labels = []
filename = 'SkillsetUniversal.txt'


def read_txt(file, text):
    """Reading data from text files(universal skill list)."""
    with open(file, 'r') as f:
        reader = csv.reader(f)
        for row in reader:
            for item in row:
                text.append(item)
    return text


def select_from():
    """Reading data from tables."""
    conn = sqlite3.connect('linkedeed.db')
    c = conn.cursor()
    jobs = []
    for row in c.execute('SELECT DESCRIPTION FROM jobs_indeed'):
        jobs.append(row)
    c.close()
    c = conn.cursor()
    skills = []
    for row in c.execute('SELECT SKILLS FROM linkedin_skills'):
        skills.append(row)
    c.close()
    conn.close()
    return jobs, skills


def jobs_calc(job, text):
    """Counting how many skills match."""
    count = 0
    for uni_word in job:
        if uni_word in text:
            count += 1
    return count


def data_cleanup(jobs, skills, txt):
    """Clean up of each job description and skills."""
    for i in range(len(jobs)):
        jobs[i] = unicodedata.normalize('NFKD',
"".join(jobs[i])).encode('ascii','ignore')

    # cz skills will just be in 1 cell
    skills[0] = unicodedata.normalize('NFKD',
"".join(skills[0])).encode('ascii', 'ignore')
    skill_l = (skills[0].split(', '))
    skill_list = [item.lower() for item in skill_l]
    text = [item.lower() for item in txt]
    jobss = [item.lower() for item in jobs]
```

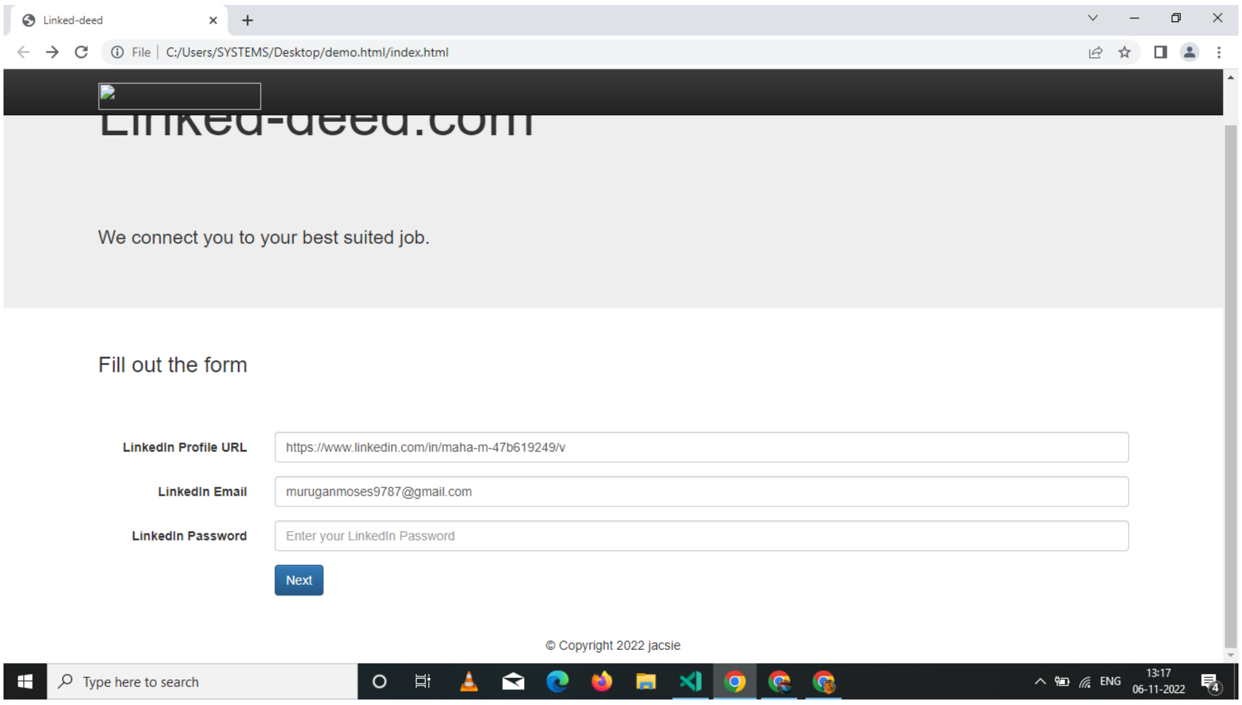
```python
    job_list = []
    for job in jobss:
        job_list.append(job.split(' '))
    return skill_list, text, job_list


def accuracy(skill_list, text, job_list):
    """Calculate match accuracy for each job."""
    # skills_required = []
    # skills_match = []
    match = []  # match depending upon the skills required.
    for job in job_list:
        x = jobs_calc(job, text)
        y = jobs_calc(skill_list, job)
        # skills_required.append(x)
        # skills_match.append(y)
        match.append((float(y) / (x + 1)) * 100)
    return match


def update_table(match):
    """Update it in sqlite and order this in desc."""
    conn = sqlite3.connect('linkedeed.db')
    sql = '''UPDATE jobs_indeed SET ACCURACY = ? WHERE ID = ?'''
    for i in range(len(match)):
        cur = conn.cursor()
        cur.execute(sql, (match[i], i + 1))
        cur.close()
    conn.commit()
    conn.close()


def main():
    """Start up."""
    txt = read_txt(filename, labels)
    jobs, skills = select_from()
    skill_list, text, job_list = data_cleanup(jobs, skills, txt)
    skills_you_have = jobs_calc(skill_list, text)

    match = accuracy(skill_list, text, job_list)
    update_table(match)

# main()
"""Implementing an algorithm to find out the best match jobs."""
import csv
import sqlite3
import unicodedata

global total_count

text = []
labels = []
filename = 'SkillsetUniversal.txt'
```

```python
def read_txt(file, text):
    """Reading data from text files(universal skill list)."""
    with open(file, 'r') as f:
        reader = csv.reader(f)
        for row in reader:
            for item in row:
                text.append(item)
    return text


def select_from():
    """Reading data from tables."""
    conn = sqlite3.connect('linkedeed.db')
    c = conn.cursor()
    jobs = []
    for row in c.execute('SELECT DESCRIPTION FROM jobs_indeed'):
        jobs.append(row)
    c.close()
    c = conn.cursor()
    skills = []
    for row in c.execute('SELECT SKILLS FROM linkedin_skills'):
        skills.append(row)
    c.close()
    conn.close()
    return jobs, skills


def jobs_calc(job, text):
    """Counting how many skills match."""
    count = 0
    for uni_word in job:
        if uni_word in text:
            count += 1
    return count


def data_cleanup(jobs, skills, txt):
    """Clean up of each job description and skills."""
    for i in range(len(jobs)):
        jobs[i] = unicodedata.normalize('NFKD',
"".join(jobs[i])).encode('ascii','ignore')

    # cz skills will just be in 1 cell
    skills[0] = unicodedata.normalize('NFKD',
"".join(skills[0])).encode('ascii', 'ignore')
    skill_l = (skills[0].split(', '))
    skill_list = [item.lower() for item in skill_l]
    text = [item.lower() for item in txt]
    jobss = [item.lower() for item in jobs]
    job_list = []
    for job in jobss:
        job_list.append(job.split(' '))
    return skill_list, text, job_list


def accuracy(skill_list, text, job_list):
    """Calculate match accuracy for each job."""
    # skills_required = []
    # skills_match = []
    match = []  # match depending upon the skills required.
    for job in job_list:
        x = jobs_calc(job, text)
        y = jobs_calc(skill_list, job)
        # skills_required.append(x)
        # skills_match.append(y)
        match.append((float(y) / (x + 1)) * 100)
    return match


def update_table(match):
    """Update it in sqlite and order this in desc."""
    conn = sqlite3.connect('linkedeed.db')
    sql = '''UPDATE jobs_indeed SET ACCURACY = ? WHERE ID = ?'''
    for i in range(len(match)):
        cur = conn.cursor()
        cur.execute(sql, (match[i], i + 1))
        cur.close()
    conn.commit()
    conn.close()


def main():
    """Start up."""
    txt = read_txt(filename, labels)
    jobs, skills = select_from()
    skill_list, text, job_list = data_cleanup(jobs, skills, txt)
    skills_you_have = jobs_calc(skill_list, text)

    match = accuracy(skill_list, text, job_list)
    update_table(match)

# main()
"""Implementing an algorithm to find out the best match jobs."""
import csv
import sqlite3
import unicodedata

global total_count

text = []
labels = []
filename = 'SkillsetUniversal.txt'
```

```python
def read_txt(file, text):
    """Reading data from text files(universal skill list)."""
    with open(file, 'r') as f:
        reader = csv.reader(f)
        for row in reader:
            for item in row:
                text.append(item)
    return text


def select_from():
    """Reading data from tables."""
    conn = sqlite3.connect('linkedeed.db')
    c = conn.cursor()
    jobs = []
    for row in c.execute('SELECT DESCRIPTION FROM jobs_indeed'):
        jobs.append(row)
    c.close()
    c = conn.cursor()
    skills = []
    for row in c.execute('SELECT SKILLS FROM linkedin_skills'):
        skills.append(row)
    c.close()
    conn.close()
    return jobs, skills


def jobs_calc(job, text):
    """Counting how many skills match."""
    count = 0
    for uni_word in job:
        if uni_word in text:
            count += 1
    return count


def data_cleanup(jobs, skills, txt):
    """Clean up of each job description and skills."""
    for i in range(len(jobs)):
        jobs[i] = unicodedata.normalize('NFKD',
"".join(jobs[i])).encode('ascii','ignore')

    # cz skills will just be in 1 cell
    skills[0] = unicodedata.normalize('NFKD',
"".join(skills[0])).encode('ascii', 'ignore')
    skill_l = (skills[0].split(', '))
    skill_list = [item.lower() for item in skill_l]
    text = [item.lower() for item in txt]
    jobss = [item.lower() for item in jobs]
    job_list = []
    for job in jobss:
        job_list.append(job.split(' '))
    return skill_list, text, job_list
```

```python
def accuracy(skill_list, text, job_list):
    """Calculate match accuracy for each job."""
    # skills_required = []
    # skills_match = []
    match = []  # match depending upon the skills required.
    for job in job_list:
        x = jobs_calc(job, text)
        y = jobs_calc(skill_list, job)
        # skills_required.append(x)
        # skills_match.append(y)
        match.append((float(y) / (x + 1)) * 100)
    return match


def update_table(match):
    """Update it in sqlite and order this in desc."""
    conn = sqlite3.connect('linkedeed.db')
    sql = '''UPDATE jobs_indeed SET ACCURACY = ? WHERE ID = ?'''
    for i in range(len(match)):
        cur = conn.cursor()
        cur.execute(sql, (match[i], i + 1))
        cur.close()
    conn.commit()
    conn.close()


def main():
    """Start up."""
    txt = read_txt(filename, labels)
    jobs, skills = select_from()
    skill_list, text, job_list = data_cleanup(jobs, skills, txt)
    skills_you_have = jobs_calc(skill_list, text)

    match = accuracy(skill_list, text, job_list)
    update_table(match)

# main()
```

Output:



Linked-deed.com

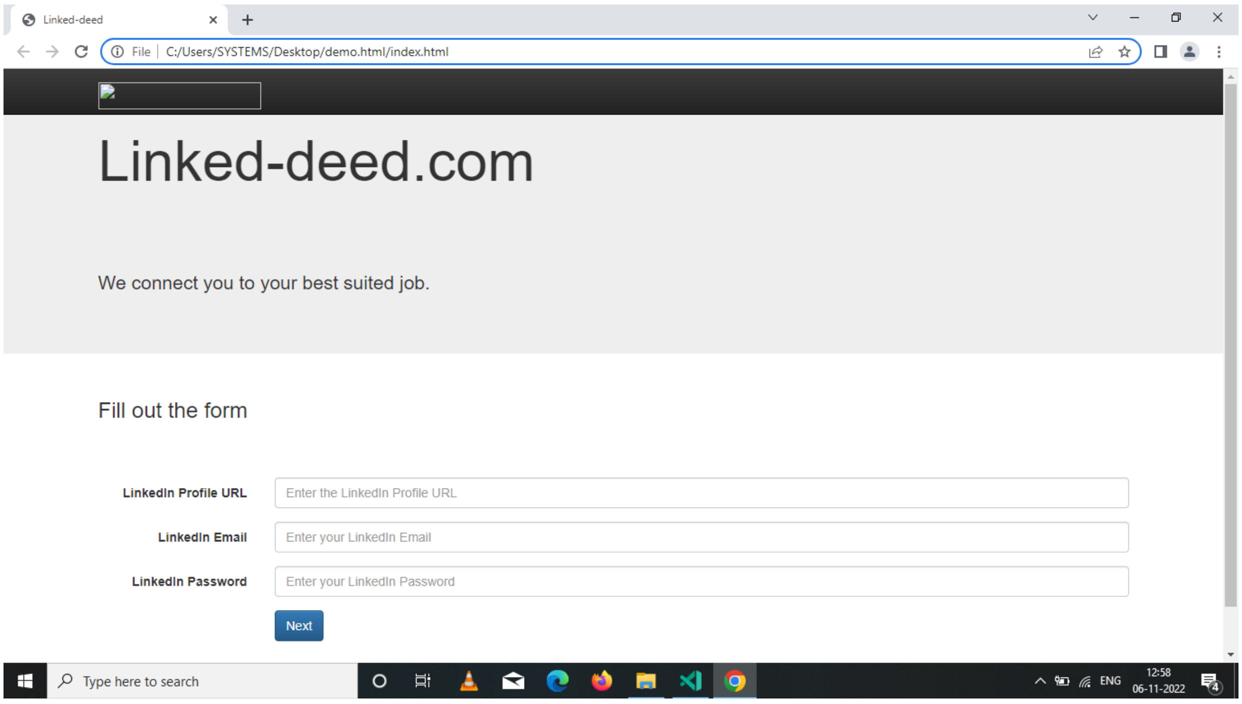
We connect you to your best suited job.

Fill out the form

LinkedIn Profile URL	https://www.linkedin.com/in/maha-m-47b619249/v
LinkedIn Email	muruganmoses9787@gmail.com
LinkedIn Password	Enter your LinkedIn Password

Next

© Copyright 2022 jacsie



Linked-deed.com

We connect you to your best suited job.

Fill out the form

LinkedIn Profile URL	Enter the LinkedIn Profile URL
LinkedIn Email	Enter your LinkedIn Email
LinkedIn Password	Enter your LinkedIn Password

Next