~\OneDrive\Desktop\Hackathon\Complete_code.py

```python
# Importing necessary libraries
import pandas as pd
import nltk
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
import string
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score
import tkinter as tk
from tkinter import ttk
from tkinter import messagebox

# Loading the datasets
resume_df = pd.read_excel('updated_resume_dataset.xlsx')
internship_df = pd.read_excel('updated_internship_dataset.xlsx')

# Preprocessing
resume_df.fillna('', inplace=True)
internship_df.fillna('', inplace=True)

nltk.download('punkt')
nltk.download('stopwords')
# defining function for preprocessing skills
def preprocess_skills(skills):
    if not isinstance(skills, str) or skills.strip() == '':
        return []
    tokens = word_tokenize(skills.lower())
    tokens = [word for word in tokens if word not in stopwords.words('english') and word not in string.punctuation]
    return tokens

resume_df['processed_Skills'] = resume_df['Skills'].apply(preprocess_skills)
internship_df['processed_Required_Skills'] = internship_df['Required Skills'].apply(preprocess_skills)

# Creating a set of unique skills
all_Skills = resume_df['processed_Skills'].sum() + internship_df['processed_Required_Skills'].sum()
unique_Skills = set(all_Skills)
Skill_to_index = {skill: idx for idx, skill in enumerate(unique_Skills)}

# Converting skills to vectors(numerical vectors)
def skills_to_vector(skills):
    vector = [0] * len(Skill_to_index)
    for skill in skills:
        if skill in Skill_to_index:
            vector[Skill_to_index[skill]] += 1
    return vector

resume_df['Skill_vector'] = resume_df['processed_Skills'].apply(skills_to_vector)
```

```
internship_df['Required_Skill_vector'] = internship_df['processed_Required_Skills'].apply(skills_to_vector)

#defining a function for matching  using Jaccard similarity
def calculate_similarity(resume_skills, internship_skills):
    set_resume_skills = set(resume_skills)
    set_internship_skills = set(internship_skills)
    intersection = set_resume_skills.intersection(set_internship_skills)
    union = set_resume_skills.union(set_internship_skills)
    if len(union) == 0:
        return 0
    return len(intersection) / len(union)

# Defining a function to match internships based on similarity score
def match_internships(resume):
    results = []
    for index, internship in internship_df.iterrows():
        similarity_score = calculate_similarity(resume['processed_Skills'], internship['processed_Required_Skills'])
        if similarity_score > 0.5:
            results.append({
                'internship_title': internship['Title'],
                'company': internship['Company'],
                'location': internship['Location'],
                'description': internship['Description'],
                'similarity_score': similarity_score
            })

    # Sorting results by similarity score
    results = sorted(results, key=lambda x: x['similarity_score'], reverse=True)
    return results

# defining a function to display matched internships in a pop-up window
def show_results(results, resume_name):
    if not results:
        messagebox.showinfo("No Matches", f"No internships matched for {resume_name}.")
        return

    results_window = tk.Toplevel(root)
    results_window.title(f"Matched Internships for {resume_name}")
    results_window.geometry("600x400")
    results_window.configure(bg='#fafafa')

    # Displaying top matching internship details
    top_match = results[0]
    match_title = f"Title: {top_match['internship_title']}"
    match_company = f"Company: {top_match['company']}"
    match_location = f"Location: {top_match['location']}"
    match_description = f"Description: {top_match['description']}"

    tk.Label(results_window, text=match_title, font=('Helvetica', 14), bg='#fafafa').pack(pady=10)
    tk.Label(results_window, text=match_company, font=('Helvetica', 12), bg='#fafafa').pack(pady=10)
```

```
    tk.Label(results_window, text=match_location, font=('Helvetica', 12), bg='#fafafa').pack(pady=10)
    tk.Label(results_window, text=match_description, font=('Helvetica', 12), wraplength=500, bg='#fafafa').pack(pady=10)
    tk.Button(results_window, text="Close", command=results_window.destroy, bg='#00796b', fg='white', font=('Helvetica', 12)).pack(pady=20)

#Defining a function to find and match internships based on applicant name
def find_applicant_and_match_internships():
    applicant_name = entry_name.get().strip()
    if not applicant_name:
        messagebox.showwarning("Input Error", "Please enter a valid applicant name.")
        return

    matching_resume = resume_df[resume_df['Name'].str.contains(applicant_name, case=False)]
    if matching_resume.empty:
        messagebox.showinfo("No Results", f"No resume found for applicant: {applicant_name}")
    else:
        resume = matching_resume.iloc[0]
        matched_internships = match_internships(resume)
        show_results(matched_internships, resume['Name'])

# Creating the main application window
root = tk.Tk()
root.title("SkillSync-Resume Based Internship Matcher")
root.geometry("800x600")
root.configure(bg='#e0f7fa')

#Making style configuration for buttons and labels
style = ttk.Style()
style.configure('TButton', font=('Helvetica', 12), padding=10)
style.configure('TLabel', font=('Helvetica', 12), padding=10)
style.configure('TEntry', font=('Helvetica', 12))

# Creating the user interface elements
title_label = ttk.Label(root, text="SkillSync-Resume Based Internship Matcher", font=('Helvetica', 24), background='#e0f7fa')
title_label.pack(pady=20)

name_label = ttk.Label(root, text="Enter Applicant Name:", background='#e0f7fa')
name_label.pack(pady=10)

entry_name = ttk.Entry(root, width=40)
entry_name.pack(pady=10)

search_button = ttk.Button(root, text="Find Matching Internships", command=find_applicant_and_match_internships)
search_button.pack(pady=20)

# Run the Tkinter main loop to display the window
root.mainloop()

```