****

**MINISTERUL EDUCAȚIEI ȘI CERCETĂRII**

**AL REPUBLICII MOLDOVA**

**Universitatea Tehnică a Moldovei**

**Facultatea Calculatoare, Informatică şi Microelectronică**

**Departamentul Informatică şi Ingineria Sistemelor**

**Disciplina:Învățarea automată**

**Teză de an**

**Tema:** **Corectarea erorilor în texte istorice**

A efectuat: A controlat:

Ciobanu Cristalin, Bumbu Tudor,

*student asist.univ.*

IA-212

**Chișinău 2023**

**CUPRINS**

[1. **INTRODUCERE** 3](#_Toc135390939)

[1 **ISTORICUL TEXTELOR. MATERIALE UTILIZATE**. 4](#_Toc135390940)

[1.1 Materiale pe care a fost scris textul istoric 4](#_Toc135390941)

[1.2 Formatul textelor istorice 5](#_Toc135390942)

[1.3 Importanța conservării și restaurării textelor istorice 5](#_Toc135390943)

[2 **CORECTAREA COMPUTERIZATĂ A ERORILOR** 7](#_Toc135390944)

[2.1 Vladimir Levenshtein - părintele corectării textelor 7](#_Toc135390945)

[2.2 Algoritmul Levenshtein 7](#_Toc135390946)

[2.3 Algortimul Wagner-Fischer 9](#_Toc135390947)

[3 **INSTRUMENTE UTILIZATE** 13](#_Toc135390948)

[3.1 Microsoft Visual Studio 13](#_Toc135390949)

[3.2 Python 13](#_Toc135390950)

[3.3 Biblioteci Python utilizate 14](#_Toc135390951)

[3.3.1 Biblioteca os: 15](#_Toc135390952)

[3.3.2 Biblioteca tkinter: 15](#_Toc135390953)

[3.3.3 Biblioteca filedialog din tkinter: 15](#_Toc135390954)

[3.3.4 Biblioteca tkinter.ttk: 15](#_Toc135390955)

[3.3.5 Biblioteca re: 15](#_Toc135390956)

[3.4 Pașii necesari pentru corectarea automată. Dicționar. 16](#_Toc135390957)

[4 **CONCLUZII** 20](#_Toc135390958)

[5 **BIBLIOGRAFIE** 21](#_Toc135390959)

[6 **ANEXE** 22](#_Toc135390960)

# INTRODUCERE

Corectarea erorilor în texte istorice are o importanță crucială. Istoria este un domeniu în care autenticitatea și acuratețea informațiilor sunt deosebit de valoroase pentru înțelegerea trecutului și influența sa asupra prezentului. Corectarea erorilor din texte istorice ne permite să ne asigurăm că informațiile pe care le obținem despre evenimente, personalități și procese istorice sunt cât mai exacte și complete posibil.

O eroare într-un text istoric poate distorsiona interpretarea și înțelegerea evenimentelor și poate conduce la perpetuarea miturilor și a informațiilor inexacte. Corectarea acestor erori aduce claritate și precizie în ceea ce privește aspectele istorice și contribuie la construirea unei imagini mai fidele a trecutului. În plus, corectarea erorilor în texte istorice are un impact semnificativ asupra cercetării istorice și a studiului științific al trecutului. Aceasta permite istoricilor, cercetătorilor și studenților să se bazeze pe surse corecte și să lucreze cu informații verificate, ceea ce duce la o dezvoltare mai solidă și riguroasă a cunoașterii istorice. Un alt aspect important al corectării erorilor în texte istorice este acela că aceasta ajută la contracararea dezinformării și manipulării istorice. Utilizarea incorectă sau intenționată a informațiilor istorice poate servi scopurilor politice, ideologice sau naționale și poate distorsiona percepția publicului asupra trecutului. Corectarea erorilor este o modalitate de a combate astfel de manipulări și de a promova o înțelegere bazată pe fapte.

În concluzie, corectarea erorilor în texte istorice are o importanță actuală și considerabilă în ceea ce privește înțelegerea și interpretarea trecutului. Aceasta contribuie la acuratețea informațiilor istorice, promovează cercetarea riguroasă și contracarează manipularea și dezinformarea. Prin eforturile de corectare, putem obține o imagine mai clară și mai autentică a istoriei noastre și a influenței sale asupra prezentului.

# ISTORICUL TEXTELOR. MATERIALE UTILIZATE.

## Materiale pe care a fost scris textul istoric

Textele istorice au apărut încă din antichitate, odată cu dezvoltarea scrisului și a capacității umane de a înregistra evenimentele și informațiile importante. Inițial, aceste texte erau înscrise pe table de lut sau papyrus, iar mai târziu pe pergament și hârtie.

Papyrusul era obținut din tulpina plantei de papyrus, care creștea în regiunile umede ale Egiptului antic și ale altor zone din Orientul Mijlociu. Procesul de obținere a papyrusului implica mai multe etape. În primul rând, tulpinile de papyrus erau tăiate în lungime și înmuiate în apă pentru a le separa în fibre subțiri. Aceste fibre erau apoi așezate orizontal și vertical, alternându-le, pentru a forma un strat dens. După aceea, stratul era presat și uscat, iar rezultatul era o foaie flexibilă și subțire de papyrus, pe care se putea scrie.

Textele erau scrise pe papyrus cu ajutorul unei scrieri numite hieratică sau demotică, care era un stil de scriere cursivă folosit în antichitate. Pentru a scrie pe papyrus, se foloseau stilouri din trestie, denumite calame, care erau tăiate la un capăt într-un unghi pentru a forma un vârf ascuțit. Cerneala folosită era, de obicei, obținută din amestecuri de apă, cărbune sau funingine și gumă arabică.

Tablele de lut erau folosite în principal în Mesopotamia antică pentru a înregistra informații și texte, inclusiv evenimente istorice. Aceste table erau făcute din lut și erau sub formă de plăci plate sau cilindrice. Pentru a scrie pe ele, se folosea un stilou ascuțit din trestie sau un alt instrument de scriere, iar textul era înscris prin apăsarea stiloului în lutul moale al tablei. După aceea, tablele de lut erau uscate sau coapte, iar textul rămânea imprimat în materialul ceramic.

Este important de menționat că papyrusul și tablele de lut erau materiale fragile și sensibile la deteriorare. Din acest motiv, multe dintre textele antice scrise pe aceste suporturi nu s-au păstrat în întregime, iar informațiile despre istorie și cultură provin în mare parte din fragmente și copii ulterioare.

Deci în concluzie, papyrusul era obținut din tulpinile plantei de papyrus și era folosit pentru a scrie texte istorice în antichitate. Textele erau scrise pe papyrus cu ajutorul stilourilor de trestie și cernelei. Tablele de lut, în schimb, erau făcute din lut și erau utilizate în Mesopotamia antică, iar scrierea se realiza prin apăsarea stiloului în lutul moale al tablei. Ambele suporturi erau folosite pentru a înregistra și a transmite informații despre trecut, iar astăzi ele oferă o importantă perspectivă asupra istoriei antice.

## Formatul textelor istorice

Formatul textelor istorice a evoluat de-a lungul timpului în funcție de suportul pe care erau înregistrate. În antichitate și Evul Mediu, textele istorice erau adesea cronici sau narative, care descriau evenimentele istorice într-o succesiune cronologică și povesteau despre acțiunile și personalitățile importante ale epocii respective. Ulterior, s-au dezvoltat și alte forme de texte istorice, precum tratate, monografii, eseuri sau articole de specialitate, care abordează subiecte istorice într-un mod mai analitic și detaliat.

Limbajul folosit în texte istorice depinde de perioada istorică și de contextul cultural în care au fost scrise. În textele mai vechi, precum cele din antichitate sau Evul Mediu, se folosea adesea limba și stilul specific al epocii respective, ceea ce poate face înțelegerea acestor texte mai dificilă pentru cititorii moderni. Cu toate acestea, pe măsură ce timpul a avansat, s-au dezvoltat și standardele de comunicare și stilurile de scriere, ceea ce a dus la utilizarea unui limbaj mai clar și mai accesibil în texte istorice.

Este important de menționat că, în textele istorice, există o tendință de a se utiliza un limbaj obiectiv și neutru, evitându-se în general exprimarea unor opinii subiective sau a judecăților de valoare. Scopul este de a oferi o descriere și o interpretare faptică a evenimentelor și proceselor istorice, bazată pe dovezi și surse verificabile.

Textele istorice au apărut ca rezultat al nevoii umane de a înregistra și transmite informații despre trecut. Formatul acestor texte a evoluat în funcție de suportul de înregistrare și de evoluția stilurilor de scriere. Limbajul folosit în texte istorice variază în funcție de perioada istorică și contextul cultural, iar obiectivitatea și utilizarea unui limbaj neutru sunt aspecte importante în aceste texte.

## Importanța conservării și restaurării textelor istorice

Importanța conservării și restaurării textelor istorice este deosebit de semnificativă pentru patrimoniul cultural și istoric al omenirii. Aceste texte reprezintă surse primare de informații despre evenimente, societăți, culturi și personalități din trecut și sunt cruciale pentru înțelegerea și interpretarea istoriei.

Iată câteva motive cheie pentru care conservarea și restaurarea textelor istorice sunt importante:

* Păstrarea patrimoniului cultural: textele istorice fac parte din patrimoniul cultural al umanității și conțin informații inestimabile despre identitatea noastră colectivă. Prin conservarea și restaurarea lor, ne asigurăm că aceste informații valoroase nu se pierd și că pot fi accesibile și studiate de generațiile actuale și viitoare.
* Relevanța pentru cercetare și studii academice: textele istorice sunt surse primare de informații pentru istorici, cercetători și studenți în domeniul istoriei. Prin conservarea și restaurarea lor, se oferă posibilitatea de a continua cercetarea și studiul în profunzime a trecutului, contribuind la dezvoltarea cunoașterii istorice și la avansarea în domeniu.
* Autenticitate și acuratețe: textele istorice autentice și restaurate corespunzător sunt esențiale pentru a asigura acuratețea informațiilor despre evenimente și procese istorice. Prin intermediul conservării și restaurării, se elimină deteriorarea și pierderea informațiilor, permițând astfel cercetătorilor să acceseze și să utilizeze surse fiabile și verificate pentru interpretarea trecutului.
* Completarea și reconstrucția poveștii istoriei: Uneori, textele istorice pot fi fragmentare sau incomplete din cauza deteriorării sau a pierderii parțiale. Prin intermediul restaurării, se poate restabili și completa aceste texte, oferind o perspectivă mai cuprinzătoare și mai clară asupra evenimentelor și proceselor istorice.
* Educație și învățare: Textele istorice conservate și restaurate au un rol esențial în educație și învățare. Ele permit elevilor, studenților și publicului larg să înțeleagă și să aprecieze moștenirea culturală și istorică, contribuind astfel la formarea unei conștiințe istorice și la dezvoltarea gândirii critice.

În concluzie, conservarea și restaurarea textelor istorice sunt de o importanță vitală pentru păstrarea și valorificarea patrimoniului cultural și istoric. Ele asigură accesul la informații autentice, contribuie la dezvoltarea cunoașterii istorice și sprijină cercetarea, educația și înțelegerea trecutului. Prin conservarea și restaurarea textelor istorice, ne asigurăm că povestea noastră colectivă rămâne vie și relevantă în continuare.

# CORECTAREA COMPUTERIZATĂ A ERORILOR

## Vladimir Levenshtein - părintele corectării textelor

Vladimir Iosifovich Levenshtein s-a născut pe 10 noiembrie 1935, în Moscova, Uniunea Sovietică. A fost un matematician și inginer cu contribuții semnificative în domeniul informaticii teoretice și al prelucrării limbajului natural. Levenshtein este cunoscut în special pentru dezvoltarea algoritmului distanței Levenshtein, care măsoară diferența între două șiruri de caractere.

Levenshtein și-a început studiile la Institutul de Fizică și Tehnologie din Moscova (MIPT), o instituție prestigioasă de învățământ superior în domeniul științelor exacte și tehnologiei. Ulterior, și-a continuat studiile la Universitatea de Stat din Moscova, unde și-a susținut teza de doctorat în 1965.

În cadrul tezei sale de doctorat, Levenshtein a prezentat algoritmul distanței Levenshtein, cunoscut și sub numele de distanța de editare. Acest algoritm calculează numărul minim de operații necesare (inserări, ștergeri și înlocuiri de caractere) pentru a transforma un șir de caractere în altul. Algoritmul a devenit ulterior un instrument esențial în domeniul informaticii teoretice și al prelucrării limbajului natural.

Algoritmul distanței Levenshtein a fost inițial conceput pentru rezolvarea problemelor de editare a șirurilor, cum ar fi corectarea automată a greșelilor de tastare sau compararea similarității dintre cuvinte. Cu toate acestea, acesta a fost extins și aplicat într-o varietate de alte domenii. De exemplu, în biologie moleculară, algoritmul este utilizat pentru a compara secvențele de ADN sau ARN și a identifica similarități sau diferențe între ele.

Contribuțiile lui Levenshtein au fost recunoscute pe scară largă în comunitatea științifică, iar el a primit mai multe distincții și premii pentru munca sa. De-a lungul carierei sale, a publicat numeroase lucrări științifice și a fost activ în dezvoltarea teoriei informației, a calculabilității și a teoriilor limbajului natural.

Algoritmul distanței Levenshtein rămâne un instrument important și utilizat în prelucrarea limbajului natural, învățarea automată și domeniile conexe. Contribuția lui Vladimir Iosifovich Levenshtein a avut un impact semnificativ asupra dezvoltării informaticii și a fost de o importanță deosebită în dezvoltarea metodelor de analiză și prelucrare a datelor textuale.

## Algoritmul Levenshtein

Algoritmul Levenshtein, numit și distanța Levenshtein, a fost introdus de matematicianul rus Vladimir Iosifovich Levenshtein în anul 1965. Acest algoritm reprezintă o măsură a diferenței dintre două șiruri de caractere și este folosit pentru a determina numărul minim de operații necesare pentru a transforma un șir de caractere în altul. Principiul de funcționare al algoritmului se bazează pe conceptul de programare dinamică.

Principiul de funcționare al algoritmului Levenshtein se poate rezuma în câțiva pași principali:

1. Inițializare: algoritmul începe prin inițializarea unei matrice de dimensiuni (m+1) x (n+1), unde m și n reprezintă lungimile celor două șiruri de caractere de comparat. Această matrice este utilizată pentru a stoca rezultate intermediare și pentru a calcula distanța Levenshtein.

2. Completarea matricei: fiecare celulă din matrice este completată în funcție de caracterul corespunzător din cele două șiruri de caractere. Algoritmul parcurge matricea de la stânga la dreapta și de sus în jos și calculează valorile în funcție de vecinii celulei curente. Valorile celulelor din matrice reprezintă distanța Levenshtein pentru prefixele șirurilor de caractere parțiale.

a. Dacă caracterele corespunzătoare sunt identice, se preia valoarea din diagonala superioară-stângă a celulei curente

b. Dacă caracterele corespunzătoare sunt diferite, se iau în considerare trei posibilități:

- Inserarea unui caracter: se adaugă 1 la valoarea din celula de deasupra (adică se trece la șirul de caractere din stânga).

- Ștergerea unui caracter: se adaugă 1 la valoarea din celula din stânga (adică se trece la șirul de caractere de sus).

- Înlocuirea unui caracter: se adaugă 1 la valoarea din diagonala superioară-stângă (adică se trece la șirul de caractere diagonal).

3. Calculul distanței Levenshtein: la finalul parcurgerii matricei, valoarea din celula din colțul dreapta-jos reprezintă distanța Levenshtein dintre cele două șiruri de caractere.

Exemple de funcționare a algoritmului Levenshtein:

Exemplu 1:

Șirul 1: "câine"

Șirul 2: "cățel"

Matricea de calculare:

| | c | ă | ț | e | l |

----------------------------

| 0 | 1 | 2 | 3 | 4 | 5 |

----------------------------

c | 1 | 0 | 1 | 2 | 3 | 4 |

----------------------------

â | 2 | 1 | 1 | 2 | 3 | 4 |

----------------------------

i | 3 | 2 | 2 | 3 | 4 | 5 |

----------------------------

n | 4 | 3 | 3 | 4 | 5 | 6 |

----------------------------

e | 5 | 4 | 4 | 5 | 4 | 5 |

Distanța Levenshtein: 5

Exemplu 2:

Șirul 1: "casa"

Șirul 2: "masa"

Matricea rezultată:

m a s a

0 1 2 3 4

c 1 1 2 3 4

a 2 1 2 3 4

s 3 2 2 2 3

a 4 3 3 3 2

Distanța de editare: 2

Astfel, algoritmul Levenshtein permite calcularea distanței dintre două șiruri de caractere și poate fi folosit într-o varietate de aplicații, cum ar fi corectarea automată a textului, verificarea ortografică și sugestii de completare în formulare online.

## Algortimul Wagner-Fischer

Algoritmul Wagner-Fischer, numit și algoritmul de editare a distanței minimă, a fost dezvoltat de către matematicienii Robert A. Wagner și Michael J. Fischer în 1974. Acest algoritm eficientizează calculul distanței Levenshtein prin utilizarea unor vectori în locul unei matrice, reducând astfel nevoia de spațiu de stocare.

Principiul de funcționare al algoritmului Wagner-Fischer poate fi rezumat în următorii pași:

1. Inițializare: se creează doi vectori de dimensiune n+1, unde n reprezintă lungimea primului șir de caractere. Primul vector, denumit predecesor, este inițializat cu valorile de la 0 la n, reprezentând distanța Levenshtein pentru șirul vid și primele caractere ale primului șir. Al doilea vector, denumit actual, este inițializat cu toate elementele fiind 0.

2. Completarea vectorilor: se parcurge fiecare caracter din cel de-al doilea șir de caractere și se actualizează valorile celor doi vectori în funcție de caracterele corespunzătoare din primul șir.

a. Dacă caracterele corespunzătoare sunt identice, se păstrează valoarea predecesorului corespunzător.

b. Dacă caracterele corespunzătoare sunt diferite, se ia în considerare minimul dintre celula de deasupra (reprezentând operația de ștergere), celula din stânga (reprezentând operația de inserție) și celula din diagonală-stânga (reprezentând operația de înlocuire), și se adaugă 1.

c. La fiecare iterație, se actualizează valorile vectorilor actual și predecesor pentru a reflecta noile calcule.

3. Calculul distanței Levenshtein: La finalul parcurgerii, valoarea din ultima celulă a vectorului actual reprezintă distanța Levenshtein dintre cele două șiruri de caractere.

Un aspect important al algoritmului Wagner-Fischer este că este un algoritm de tip bottom-up, deoarece calculează valorile celulelor vectorilor de la început până la final, bazându-se pe valorile celulelor precedente.

Algoritmul Wagner-Fischer este eficient din punct de vedere al performanței și poate fi folosit pentru a calcula distanța Levenshtein între două șiruri de caractere de lungimi diferite. De asemenea, acesta poate fi adaptat pentru a memora și reconstrui operațiile specifice (inserții, ștergeri, înlocuiri) necesare pentru a transforma un șir în celălalt.

Vom analiza un exemplu pentru a ilustra modul în care algoritmul Wagner-Fischer calculează distanța Levenshtein între două șiruri de caractere. Să presupunem că avem următoarele șiruri:

Șirul 1: "casa"

Șirul 2: "masa"

Pasul 1: Inițializare

Având în vedere că lungimea șirului 1 este 4 și lungimea șirului 2 este 4, vom crea doi vectori de dimensiune 5 (4+1) pentru a reține valorile intermediare.

Predecesor: [0, 1, 2, 3, 4]

Actual: [0, 0, 0, 0, 0]

Pasul 2: Completarea vectorilor

Vom parcurge caracterele din șirul 2 și actualizăm valorile celor doi vectori în funcție de caracterele corespunzătoare din șirul 1.

La prima iterație:

Caracterul curent din șirul 2 este 'm'.

Comparăm 'm' cu 'c' (primul caracter din șirul 1). Sunt diferite, deci luăm minimul dintre celula de deasupra (0) + 1, celula din stânga (0) + 1 și celula din diagonală-stânga (0). Astfel, valoarea curentă din vectorul Actual devine 1.

Predecesor: [0, 1, 2, 3, 4]

Actual: [0, 1, 0, 0, 0]

La a doua iterație:

Caracterul curent din șirul 2 este 'a'.

Comparăm 'a' cu 'a'. Sunt identice, deci valoarea curentă din vectorul Actual devine valoarea din diagonala-stânga (0).

Predecesor: [0, 1, 2, 3, 4]

Actual: [0, 1, 0, 0, 0]

La a treia iterație:

Caracterul curent din șirul 2 este 's'.

Comparăm 's' cu 's'. Sunt identice, deci valoarea curentă din vectorul Actual devine valoarea din diagonala-stânga (0).

Predecesor: [0, 1, 2, 3, 4]

Actual: [0, 1, 0, 0, 0]

La a patra și ultima iterație:

Caracterul curent din șirul 2 este 'a'.

Comparăm 'a' cu 'a'. Sunt identice, deci valoarea curentă din vectorul Actual devine valoarea din diagonala-stânga (0).

Predecesor: [0, 1, 2, 3, 4]

Actual: [0, 1, 0, 0, 0]

Pasul 3: Calculul distanței Levenshtein

La finalul parcurgerii, valoarea din ultima celulă a vectorului Actual reprezintă distanța Levenshtein dintre cele două șiruri de caractere.

Distanța Levenshtein: 0

Astfel, în acest exemplu, distanța Levenshtein între "casa" și "masa" este 0, deoarece cele două șiruri sunt identice.

Algoritmul Wagner-Fischer este utilizat în diverse domenii, cum ar fi prelucrarea limbajului natural, corectarea ortografică, recunoașterea vocală și compararea secvențelor ADN.

# INSTRUMENTE UTILIZATE

## Microsoft Visual Studio

Microsoft Visual Studio Code (VS Code) este un editor de cod sursă deschis și gratuit, dezvoltat de către Microsoft. Este unul dintre cele mai populare și utilizate instrumente pentru dezvoltarea de software. VS Code este disponibil pe platforme Windows, macOS și Linux.

Caracteristicile și funcționalitățile cheie ale Microsoft Visual Studio Code includ:

1. Interfață intuitivă: VS Code are o interfață de utilizator simplă și curată, care facilitează navigarea și utilizarea editorului.

2. Extensibilitate puternică: Prin intermediul sistemului său de extensii, VS Code poate fi personalizat și extins în funcție de nevoile și preferințele dezvoltatorului. Există o gamă largă de extensii disponibile pentru diferite limbaje de programare, teme, instrumente de productivitate și integrări cu alte servicii și platforme.

3. Editare avansată a codului: VS Code oferă funcționalități puternice de editare a codului, cum ar fi evidențierea sintactică, evidențierea erorilor, indentarea automată, completarea automată a codului, navigarea rapidă și multe altele. Acestea facilitează scrierea și modificarea eficientă a codului sursă.

4. Integrare cu sistemul de control al versiunilor: VS Code are suport integrat pentru sistemele de control al versiunilor, cum ar fi Git. Dezvoltatorii pot efectua operații de control al versiunilor direct din interfața editorului, cum ar fi commit-uri, push-uri și pull-uri.

5. Debogare și profilare: VS Code vine cu suport integrat pentru debogare și profilare a aplicațiilor. Dezvoltatorii pot efectua pauze în cod, investiga variabilele, urmări stiva de apel și identifica problemele de performanță.

6. Integrare cu terminalul: VS Code are un terminal integrat, care permite executarea comenzilor și interacțiunea directă cu linia de comandă a sistemului de operare.

7. Colaborare și lucru în echipă: VS Code facilitează colaborarea și lucru în echipă prin funcționalități precum partajarea de sesiuni de lucru, adnotarea și revizuirea codului, chat și altele.

Microsoft Visual Studio Code oferă un mediu de dezvoltare puternic și flexibil, potrivit atât pentru dezvoltatorii începători, cât și pentru cei experimentați. Prin intermediul extensibilității sale și a comunității active de dezvoltatori, este un instrument versatil și adaptabil la nevoile diferitelor proiecte și limbaje de programare.

## Python

Python este un limbaj de programare de nivel înalt, interpretat și general, care a fost creat de Guido van Rossum și lansat pentru prima dată în 1991. Este cunoscut pentru simplitatea sa și sintaxa clară, ceea ce îl face ușor de învățat și de utilizat pentru programatori de toate nivelele de experiență.

Caracteristici și avantaje ale limbajului Python includ:

1. Simplitate și citibilitate: Python are o sintaxă concisă și ușor de înțeles, care facilitează scrierea și citirea codului. Utilizează spații de indentare pentru a marca blocurile de cod, înlocuind astfel utilizarea acoladelor sau a cuvintelor cheie speciale, ceea ce contribuie la o structură clară și coerentă.

2. Versatilitate: Python este un limbaj de programare general, ceea ce înseamnă că poate fi utilizat pentru o varietate largă de aplicații și domenii, cum ar fi dezvoltarea web, analiza datelor, inteligența artificială, automatizarea sarcinilor, dezvoltarea jocurilor și multe altele. Există, de asemenea, o gamă bogată de biblioteci și cadre de lucru disponibile pentru Python, care extind funcționalitățile sale și facilitează dezvoltarea aplicațiilor complexe.

3. Biblioteci și ecosistem bogat: Python are un ecosistem vast de biblioteci și module, care acoperă o gamă largă de funcționalități și domenii. De exemplu, biblioteca NumPy oferă suport pentru operații matematice și științifice, biblioteca Pandas facilitează manipularea și analiza datelor, iar TensorFlow și PyTorch sunt folosite pentru dezvoltarea aplicațiilor de învățare automată.

4. Portabilitate: Python rulează pe diferite platforme, inclusiv Windows, macOS, Linux și sistemele de operare mobile. Acest lucru face posibilă dezvoltarea și rularea aplicațiilor Python pe diverse medii și dispozitive.

5. Comunitate activă: Python are o comunitate mare și activă de dezvoltatori care contribuie la dezvoltarea limbajului, crearea de biblioteci și oferirea de suport. Există resurse abundente, documentație, forumuri și grupuri de utilizatori care facilitează învățarea și rezolvarea problemelor.

Python este adesea considerat un limbaj prietenos pentru începători, dar este, în același timp, puternic și flexibil pentru dezvoltatorii avansați. Alegerea Python pentru dezvoltarea aplicațiilor oferă un echilibru între productivitate, performanță și claritate a codului.

## Biblioteci Python utilizate

Aceste biblioteci menționate mai jos sunt doar câteva exemple dintre multele disponibile în ecosistemul Python. Fiecare dintre ele oferă funcționalități specifice și facilitează dezvoltarea de aplicații în diferite domenii.

### Biblioteca os:

Biblioteca `os` oferă funcții pentru a interacționa cu sistemul de operare. Aceasta permite manipularea cestui de lucru, manipularea fișierelor și directoriilor, execuția de comenzi în linia de comandă, gestionarea variabilelor de mediu și multe altele. Este utilă pentru operațiuni de sistem de operare și gestionarea resurselor externe în aplicații Python.

### Biblioteca tkinter:

Biblioteca `tkinter` (interfață grafică Python) este o bibliotecă standard pentru crearea de interfețe grafice utilizator (GUI) în aplicațiile Python. Aceasta furnizează un set de instrumente și widget-uri grafice pentru a crea ferestre, butoane, casete de text, imagini și multe alte elemente interactivi. Cu ajutorul tkinter, poți crea aplicații cu interfață utilizator prietenoasă și interactivă.

### Biblioteca filedialog din tkinter:

Modulul `filedialog` din biblioteca tkinter oferă funcții pentru a afișa dialoguri de selectare a fișierelor și directoriilor. Acesta permite utilizatorului să navigheze prin sistemul de fișiere și să selecteze fișiere sau directoare specifice. Este util pentru a permite utilizatorului să selecteze fișiere de intrare sau de ieșire în aplicații cu interfață grafică.

### Biblioteca tkinter.ttk:

Biblioteca `tkinter.ttk` oferă un set suplimentar de widget-uri grafice pentru interfața tkinter. Aceste widget-uri, cunoscute și sub numele de widget-uri tematice, oferă un aspect mai modern și mai stilizat decât widget-urile clasice tkinter. Acestea includ butoane, cadrane, casete de selectare, liste derulante și altele.

### Biblioteca re:

Biblioteca `re` (expresii regulate) oferă funcționalități pentru a lucra cu expresii regulate în Python. Expresiile regulate sunt șabloane de text folosite pentru a căuta, verifica și manipula șiruri de caractere. Această bibliotecă permite căutarea și extragerea de șabloane, înlocuirea textului și alte operații complexe de manipulare a șirurilor de caractere.

Aceste biblioteci sunt doar câteva exemple dintre multele disponibile în ecosistemul Python. Fiecare dintre ele oferă funcționalități specifice și facilitează dezvoltarea de aplicații în diferite domenii. Este recomandat să consulți documentația oficială a bibliotecilor pentru a obține mai multe informații despre utilizare și funcționalități specifice.

## Pașii necesari pentru corectarea automată. Dicționar.

Corectarea automată a textului este un domeniu important în prelucrarea limbajului natural (NLP) și implică identificarea și corectarea erorilor ortografice sau gramaticale într-un text dat. Pentru a obține rezultate bune în corectarea automată, este esențial să se aibă un vocabular corect și complet.

Necesitatea de a crea un vocabular corect derivă din faptul că algoritmul de corectare automată folosește un dicționar sau o listă de cuvinte corecte pentru a verifica și compara cuvintele din textul de intrare. Dacă un cuvânt din textul de intrare nu este prezent în dicționar, algoritmul îl va considera incorect și poate sugera o corecție greșită sau nu va găsi nicio corecție adecvată.

Există mai multe abordări pentru construirea unui vocabular corect în corectarea automată:

1. Dicționar standard: se poate utiliza un dicționar standard care conține cuvinte corecte și acceptate în limba respectivă. Aceste dicționare pot fi disponibile în biblioteci specifice pentru NLP sau pot fi obținute din surse externe.

2. Listă de cuvinte comune: pentru a acoperi o gamă mai largă de cuvinte, se poate utiliza o listă de cuvinte comune care includ cuvinte frecvent utilizate în limbajul curent. Aceste liste pot fi colectate din diverse surse, cum ar fi texte de înaltă calitate, corpusuri lingvistice sau analize statistice ale textelor.

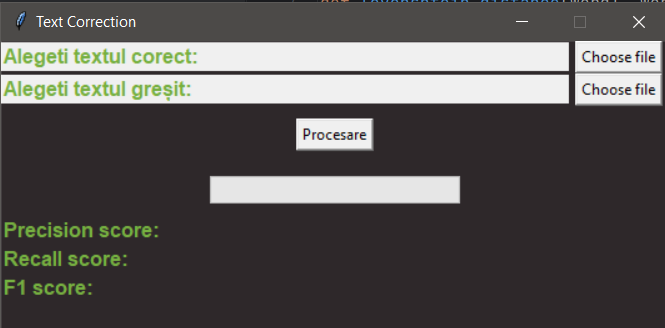
3. Extracție din corpus: se poate construi un vocabular bazat pe un corpus specific de texte relevante. Acest corpus poate fi constituit din texte dintr-un anumit domeniu sau dintr-o colecție de texte specifice. Extracția cuvintelor din corpus poate ajuta la obținerea unui vocabular mai adaptat și relevant pentru textele pe care se dorește a le corecta.

4. Tehnici de învățare automată: alte metode avansate implică utilizarea tehnicilor de învățare automată pentru a construi un model de corectare automată bazat pe un set de date etichetat. Aceste modele pot învăța dintr-un set de texte corecte și pot fi utilizate pentru a genera sugestii de corecție pentru texte noi.

Indiferent de abordarea aleasă, este important să se actualizeze și să se mențină vocabularul pe măsură ce se descoperă cuvinte noi sau înțelesuri noi ale cuvintelor existente.

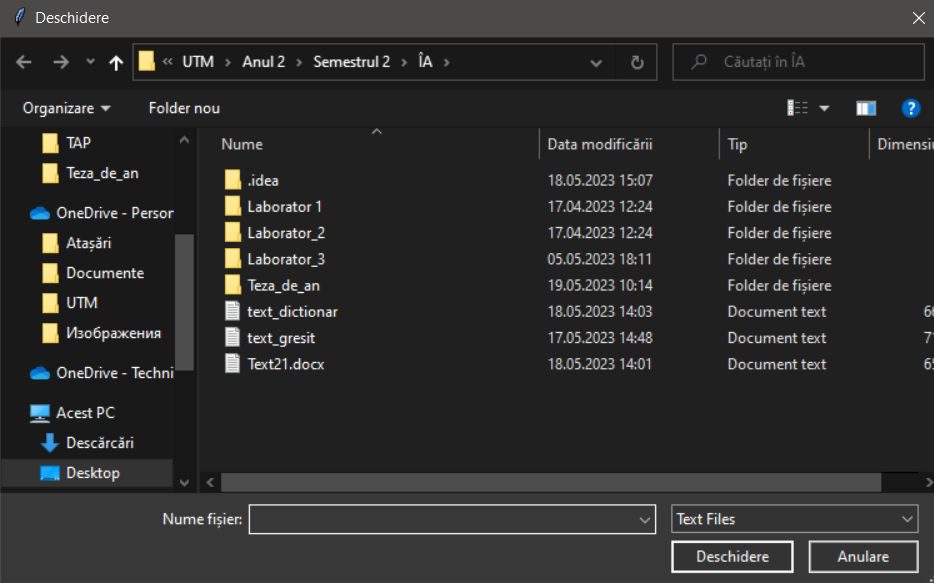
Un vocabular corect și complet este esențial în procesul de corectare automată pentru a asigura rezultate precise și fiabile. Cu toate acestea, trebuie avut în vedere că corectarea automată poate întâmpina provocări în cazul cuvintelor neconvenționale, nume proprii, termeni tehnici sau cuvinte din limbaj precum expresii, termeni și structuri gramaticale care sunt specifice unui anumit grup de oameni, unei regiuni sau unui context specific, unde nu există întotdeauna un consens asupra corectitudinii acestora.

Interfața de utilizator prezentată utilizează biblioteca Tkinter pentru a crea o interfață grafică simplă. Este construită pe baza framework-ului Tk, care este o bibliotecă de interfețe grafice în Python.



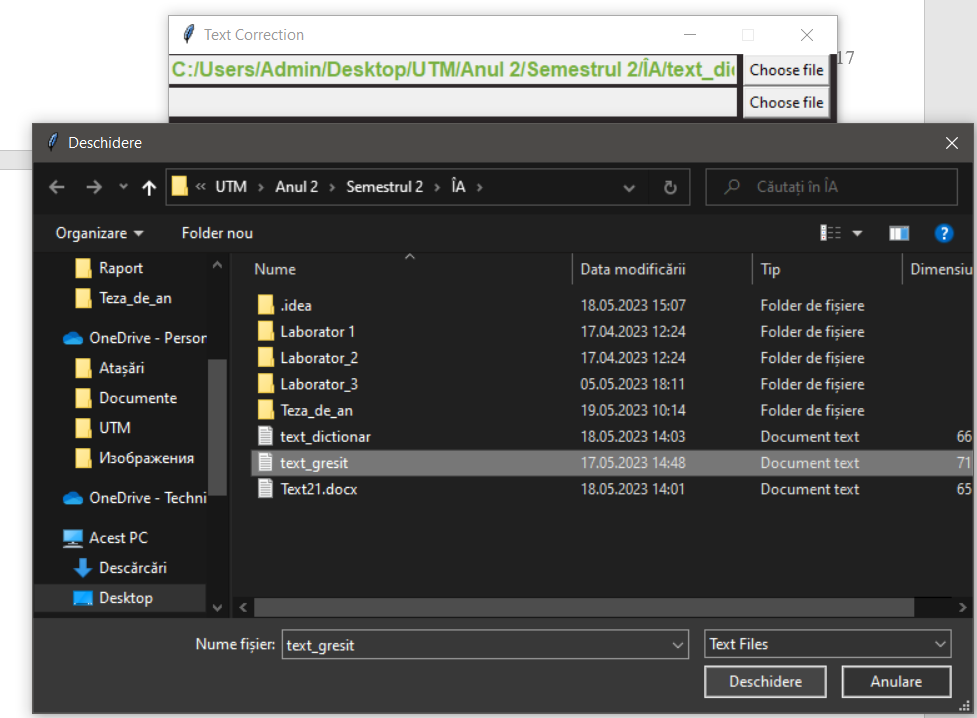
**Figura 4.4.1.** Interfața pentru aplicație

În figura 4.4.1 observăm câmpul "Alegeti textul corect" (correct\_file\_label). Acest câmp afișează calea către fișierul care conține textul corect, adică un text de referință pentru compararea și corectarea textului greșit. Utilizatorul poate selecta un fișier folosind butonul "Choose file" (correct\_file\_button).



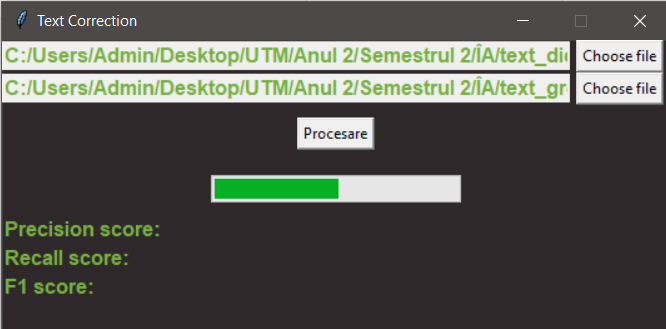
**Figura 4.4.2.** Selectare text corect

În figura 4.4.2 observăm butonul "Choose file" (correct\_file\_button). Acest buton deschide o fereastră de dialog pentru a permite utilizatorului să selecteze un fișier text care conține textul corect.



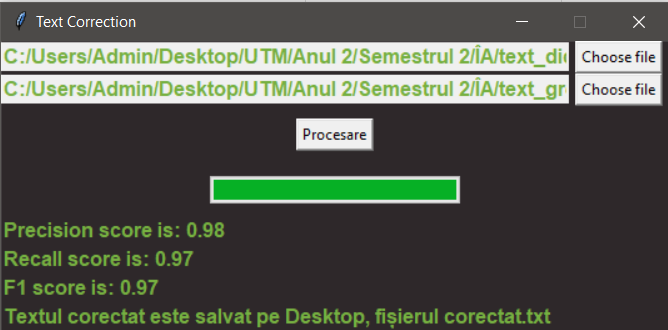
**Figura 4.4.3.** Selectare text greșit

În figura 4.4.3 observăm câmpul "Alegeti textul greșit" (text\_file\_label). Acest câmp afișează calea către fișierul care conține textul greșit, care trebuie corectat automat. Utilizatorul poate selecta un fișier folosind butonul "Choose file" (text\_file\_button).



**Figura 4.4.4.** Butonul procesare

În figura 4.4.4 observăm butonul "Procesare" (process\_button): Acest buton inițiază procesul de corectare automată a paragrafelor din textul greșit. Atunci când este apăsat, se apelează funcția `process\_paragraphs()` care preia fișierele selectate, construiește un vocabular corect din fișierul cu textul corect și aplică algoritmul de corectare automată pe paragrafele textului greșit. De asemenea este vizibilă bara de progres (progress\_bar). Această bară de progres indică progresul procesului de corectare automată a paragrafelor. Se actualizează pe măsură ce fiecare paragraf este procesat.



**Figura 4.4.5.** Precision, Recall și scorul F1

În figura 4.4.5 observăm câmpul "Precision score" (precision\_label): acest câmp afișează scorul de precizie obținut în urma procesului de corectare automată. Scorul de precizie reprezintă proporția corecțiilor corecte față de numărul total de corecții propuse. Câmpul "Recall score" (recall\_label). Acest câmp afișează scorul de acoperire obținut în urma procesului de corectare automată. Scorul de acoperire reprezintă proporția corecțiilor corecte față de numărul total de corecții necesare. Câmpul "F1 score" (f1\_score\_label). Acest câmp afișează scorul F1 obținut în urma procesului de corectare automată. Scorul F1 este o măsură a performanței care combină atât precizia, cât și acoperirea, oferind o valoare unică care indică calitatea corectării. Câmpul "Textul corectat este salvat pe Desktop, fișierul corectat.txt" (file\_saved\_label). Acest câmp afișează un mesaj informativ că textul corectat a fost salvat pe Desktop în fișierul "corectat.txt".

Interfața permite utilizatorului să selecteze fișierele de intrare (textul corect și textul greșit), să inițieze procesul de corectare automată și să afișeze scorurile de evaluare pentru procesul de corectare automată. De asemenea, informează utilizatorul că textul corectat a fost salvat într-un fișier specificat.

# CONCLUZII

Corectarea automată a textelor este un proces esențial într-o varietate de domenii, de la editarea de texte și traduceri până la analiza datelor și prelucrarea limbajului natural.

Una dintre componentele esențiale ale corectării automate a textelor este calcularea distanței de editare, cum ar fi distanța Levenshtein. Acest algoritm măsoară diferența între două cuvinte prin numărul minim de operații necesare pentru a le transforma unul în altul, cum ar fi adăugarea, ștergerea sau înlocuirea de caractere. Algoritmul Levenshtein oferă o bază solidă pentru dezvoltarea unor mecanisme de corectare automată eficiente și precise.

Un aspect important în corectarea automată a textelor este necesitatea de a avea un vocabular corect și actualizat. Un vocabular extins și precis asigură o mai bună acoperire a corectării automate și reduce erorile sau propunerea de corecții incorecte. Construirea și actualizarea constantă a unui astfel de vocabular necesită resurse și efort, dar are un impact semnificativ asupra calității procesului de corectare automată.

Algoritmul de corectare automată poate fi aplicat la diferite niveluri, de la nivel de cuvinte individuale până la nivel de fraze sau paragrafe întregi. Oferind posibilitatea de a corecta automat textul greșit, acest algoritm aduce multiple avantaje. În primul rând, economisește timp și efort în procesul de editare manuală a textelor, permițând o corectare rapidă și eficientă. De asemenea, reduce erorile umane și asigură o consistență a corectării în întregul text. În plus, prin utilizarea unor tehnologii avansate, cum ar fi învățarea automată și rețelele neurale, se pot obține performanțe superioare în corectarea automată a textelor.

Cu toate acestea, este important să ne amintim că corectarea automată a textelor nu înlocuiește complet editarea umană. Deși tehnologia poate realiza corecții precise și rapide, există contexte și sensuri subtile care necesită intervenția umană pentru o interpretare corectă. De aceea, corectarea automată a textelor este mai degrabă o unealtă complementară pentru editori și scriitori, oferindu-le suport și accelerând procesul de editare.

În concluzie, corectarea automată a textelor reprezintă o tehnologie semnificativă și în continuă dezvoltare, cu multiple aplicații și beneficii. Algoritmii de corectare automată, cum ar fi distanța Levenshtein, permit detectarea și corectarea rapidă a greșelilor din texte. Construirea și actualizarea unui vocabular corect reprezintă un aspect important pentru obținerea unor rezultate precise și de calitate. Utilizarea tehnologiilor avansate și a modelelor de învățare automată aduce îmbunătățiri semnificative în performanța corectării automate a textelor. Cu toate acestea, este esențial să se înțeleagă că corectarea automată a textelor nu elimină necesitatea editării umane, ci oferă suport și eficiență în procesul de editare. Prin combinarea abilităților umane cu puterea tehnologiei, putem obține texte corecte și de calitate superioară.

# BIBLIOGRAFIE

1. Documentația Python pentru diferite versiuni, sursă electronică, ” Python 3.11.3 documentation” [Accesat:15.05.23], disponibil: <https://docs.python.org/3/>
2. Algortimul Wagner-Fischer, sursă electronică,”Wagner–Fischer algorithm [Accesat:15.05.23],disponibil:<https://handwiki.org/wiki/Wagner%E2%80%93Fischer_algorithm>
3. Algoritmul Levenshtein și principiu de funcționare, sursă electronică, ” Levenshtein Distance Computation” , [Accesat:15.05.23], disponibil:<https://www.baeldung.com/cs/levenshtein-distance-computation>
4. Mediu de dezvoltare VS Code, sursă electronică ” Code editing.Redefined.”[Accesat:15.05.23], disponibil: <https://code.visualstudio.com/>

# ANEXE

Anexa 1:Codul sursă pentru corectarea automată a erorilor

import os

import tkinter as tk

from tkinter import filedialog

import tkinter.ttk as ttk

import re

def levenshtein\_distance(word1, word2):

m = len(word1)

n = len(word2)

# Initialize the rolling array with dimensions (m+1) x 2

matrix = [[0] \* 2 for \_ in range(m + 1)]

# Initialize the first row with values from 0 to m

for i in range(m + 1):

matrix[i][0] = i

# Iterate through the characters of the second word

for j in range(1, n + 1):

# Swap the rows in the matrix

matrix[0][1] = j

for i in range(1, m + 1):

# Calculate the minimum edit distance at each position

if word1[i - 1] == word2[j - 1]:

matrix[i][1] = matrix[i - 1][0]

else:

matrix[i][1] = min(

matrix[i - 1][0] + 1, # Deletion

matrix[i][0] + 1, # Insertion

matrix[i - 1][0] + 1 # Substitution

)

# Swap the columns in the matrix for the next iteration

for i in range(m + 1):

matrix[i][0] = matrix[i][1]

# Return the Levenshtein distance (bottom-right cell)

return matrix[m][0]

def calculate\_metrics(ground\_truth\_text, corrected\_text):

ground\_truth\_tokens = re.findall(r'\w+', ground\_truth\_text)

corrected\_tokens = re.findall(r'\w+', corrected\_text)

tp = sum(token in ground\_truth\_tokens for token in corrected\_tokens)

fp = len(corrected\_tokens) - tp

fn = len(ground\_truth\_tokens) - tp

precision = tp / (tp + fp) if tp + fp > 0 else 0.0

recall = tp / (tp + fn) if tp + fn > 0 else 0.0

f1\_score = (2 \* precision \* recall) / (precision + recall) if precision + recall > 0 else 0.0

return precision, recall, f1\_score

def choose\_correct\_file():

file\_path = filedialog.askopenfilename(filetypes=[("Text Files", "\*.txt")])

correct\_file\_var.set(file\_path)

correct\_file\_label.configure(text=file\_path)

def choose\_text\_file():

file\_path = filedialog.askopenfilename(filetypes=[("Text Files", "\*.txt")])

text\_file\_var.set(file\_path)

text\_file\_label.configure(text = file\_path)

def process\_paragraphs():

correct\_file = correct\_file\_var.get()

text\_file = text\_file\_var.get()

if not correct\_file or not text\_file:

return

vocab = {}

with open(correct\_file, 'r', encoding="UTF-8") as file:

text = file.read()

words = re.findall(r'\w+', text) # Split words without punctuation

total\_words = len(words)

for word in words:

if word not in vocab:

vocab[word] = 1

else:

vocab[word] += 1

for word in vocab:

vocab[word] /= total\_words

corectat\_file = os.path.join(os.path.expanduser("~"), "Desktop", "corectat.txt")

with open(corectat\_file, 'w', encoding="UTF-8") as fileW:

corrected\_text = ''

correct = ''

ground\_truth\_text = ''

with open(text\_file, 'r', encoding="UTF-8") as file:

paragraphs = file.readlines()

num\_paragraphs = len(paragraphs)

progress\_bar['maximum'] = num\_paragraphs

for index, paragraph in enumerate(paragraphs):

tokens = re.findall(r'\w+|[^\w\s]', paragraph)

for i in range(len(tokens)):

token = tokens[i]

if token.isalpha() and token not in vocab:

best\_token = token

best\_distance = float('inf')

# Find the best correction based on Levenshtein distance

for candidate in vocab:

distance = levenshtein\_distance(token, candidate)

if distance < best\_distance:

best\_token = candidate

best\_distance = distance

# Update the token if a better correction is found

if best\_token != token:

tokens[i] = best\_token

corrected\_text += ' '.join(tokens)

ground\_truth\_text += paragraph

correct += corrected\_text

fileW.write(corrected\_text)

fileW.write("\n")

corrected\_text = ''

progress\_bar['value'] = index + 1

window.update()

# Calculate metrics for the final result

precision, recall, f1\_score = calculate\_metrics(ground\_truth\_text, correct)

precision\_label['text'] = f"Precision score is: {precision:.2f}"

recall\_label['text'] = f"Recall score is: {recall:.2f}"

f1\_score\_label['text'] = f"F1 score is: {f1\_score:.2f}"

file\_saved\_label['text'] = "Textul corectat este salvat pe Desktop, fișierul corectat.txt"

# Create the main window

window = tk.Tk()

window.title("Text Correction")

window.configure(bg= '#2E282A')

window.resizable(False, False)

# Define a custom style for labels

custom\_label1\_style = ttk.Style()

custom\_label1\_style.configure('CustomLabel1.TLabel', foreground='#76B041', font=('Arial', 12, 'bold'))

custom\_label2\_style = ttk.Style()

custom\_label2\_style.configure('CustomLabel2.TLabel', foreground='#76B041', font=('Arial', 12, 'bold'), background = '#2E282A')

# Variables to store file paths

correct\_file\_var = tk.StringVar()

text\_file\_var = tk.StringVar()

# Label and Button for choosing the file with correct text

correct\_file\_label = ttk.Label(window, text="Alegeti textul corect:", width= 50, style='CustomLabel1.TLabel' )

correct\_file\_label.grid(row=0, column=0, sticky="w")

correct\_file\_button = tk.Button(window, text="Choose file", command=choose\_correct\_file)

correct\_file\_button.grid(row=0, column=1, padx=5)

# Label and Button for choosing the file with text to correct

text\_file\_label = ttk.Label(window, text="Alegeti textul greșit:", width = 50, style='CustomLabel1.TLabel')

text\_file\_label.grid(row=1, column=0, sticky="w")

text\_file\_button = tk.Button(window, text="Choose file", command=choose\_text\_file)

text\_file\_button.grid(row=1, column=1, padx=5)

# Button for processing paragraphs

process\_button = tk.Button(window, text="Procesare", command=process\_paragraphs)

process\_button.grid(row=2, column=0, columnspan=2, pady=10)

# Progress Bar

progress\_bar = ttk.Progressbar(window, length=200)

progress\_bar.grid(row=3, column=0, columnspan=2, pady=10)

# Label for Precision score

precision\_label = ttk.Label(window, text="Precision score: ", style='CustomLabel2.TLabel')

precision\_label.grid(row=4, column=0, sticky="w")

# Label for Recall score

recall\_label = ttk.Label(window, text="Recall score: ", style='CustomLabel2.TLabel')

recall\_label.grid(row=5, column=0, sticky="w")

# Label for F1 score

f1\_score\_label = ttk.Label(window, text="F1 score: ", style='CustomLabel2.TLabel')

f1\_score\_label.grid(row=6, column=0, sticky="w")

file\_saved\_label = ttk.Label(window, style = 'CustomLabel2.TLabel')

file\_saved\_label.grid(row = 7, column=0, sticky='nsew', columnspan= 2)

window.mainloop()