|  |  |
| --- | --- |
| Documentație  Proiect Inteligență Artificială – Machine Learning / Noiembrie 2022 | Overview Scopul acestui proiect este de a clasifica texte traduse în 5 limbi străine din engleza în funcție de dialectul lor nativ: dialect irlandez, englezesc sau scoțian. Această problemă de învățare supravegheată este una clasică de procesare a limbajului natural. Vom implementa diferite abordări pentru a observa care este mai bună pentru setul nostru de date.  Sorete Robert-Alexandru  Grupa 361 |

# **Pasul 1.1. Înțelegerea Setului de Date**

Pentru modelul nostru avem următoarele seturi de date de intrare disponibile:

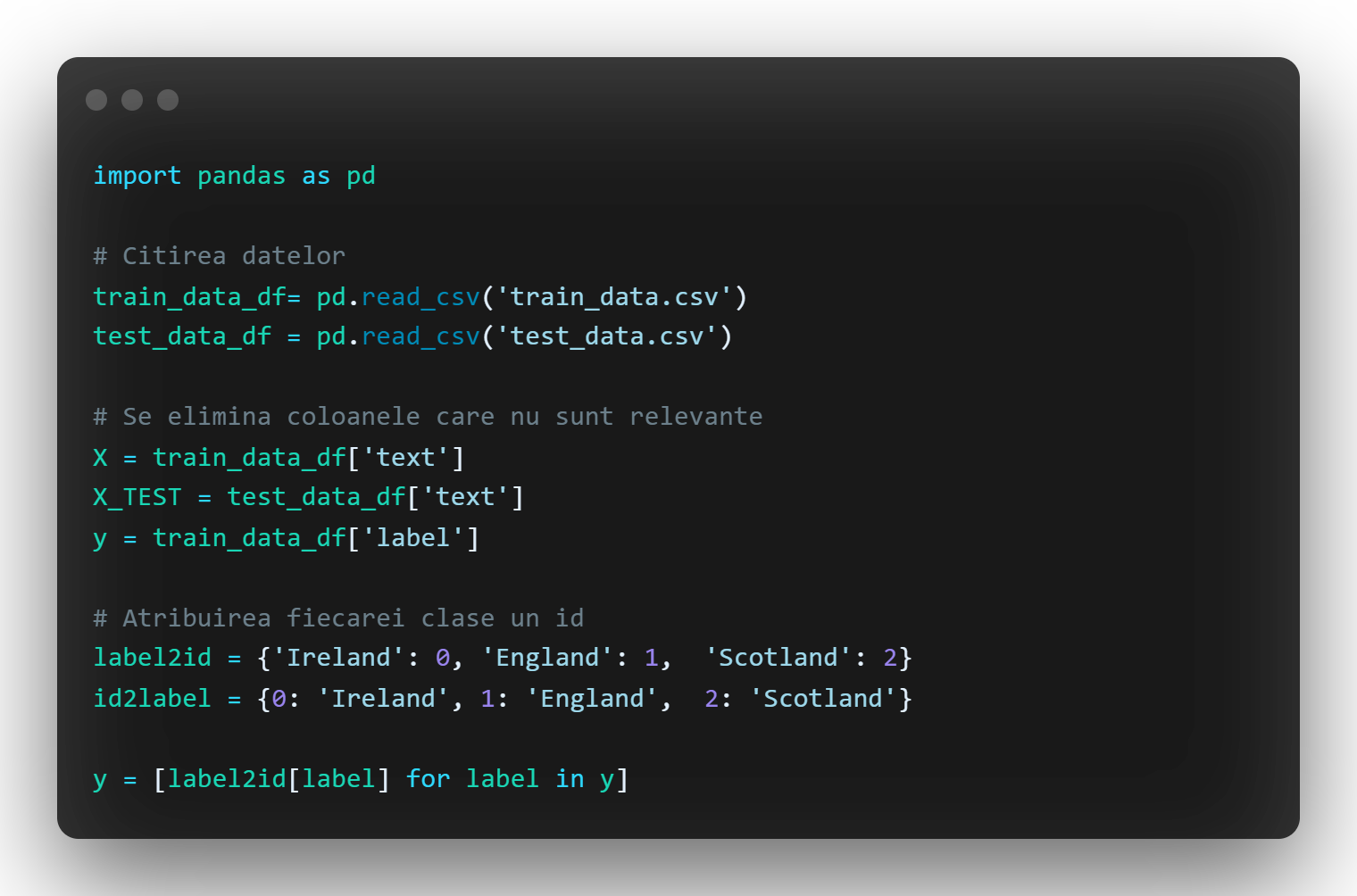
1. **train\_data.csv** - *date de antrenare*, format dintr-un .csv ce conține 41570 de exemple de antrenare împărțite în 3 coloane: limba textului, textul tradus în limba respectiva, și dialectul textului nativ (**eticheta**)
2. **test\_data.csv** - *date de testare* (conține textele ce trebuie clasificate)

Ca set de date de ieșire avem:

1. **submission.csv** - *predicția clasificatorului* nostru format din index și etichetă

# **Pasul 1.2. Citirea Setului de Date**

În implementarea acestei probleme vom citi tabelele cu ajutorul librăriei **pandas** pentru a le stoca sub forma unui **dataframe** și ne vom extrage informațiile utile în variabilele X, y reprezentând textele de antrenare, respectiv etichetele de antrenare, iar în X\_TEST textele de test.

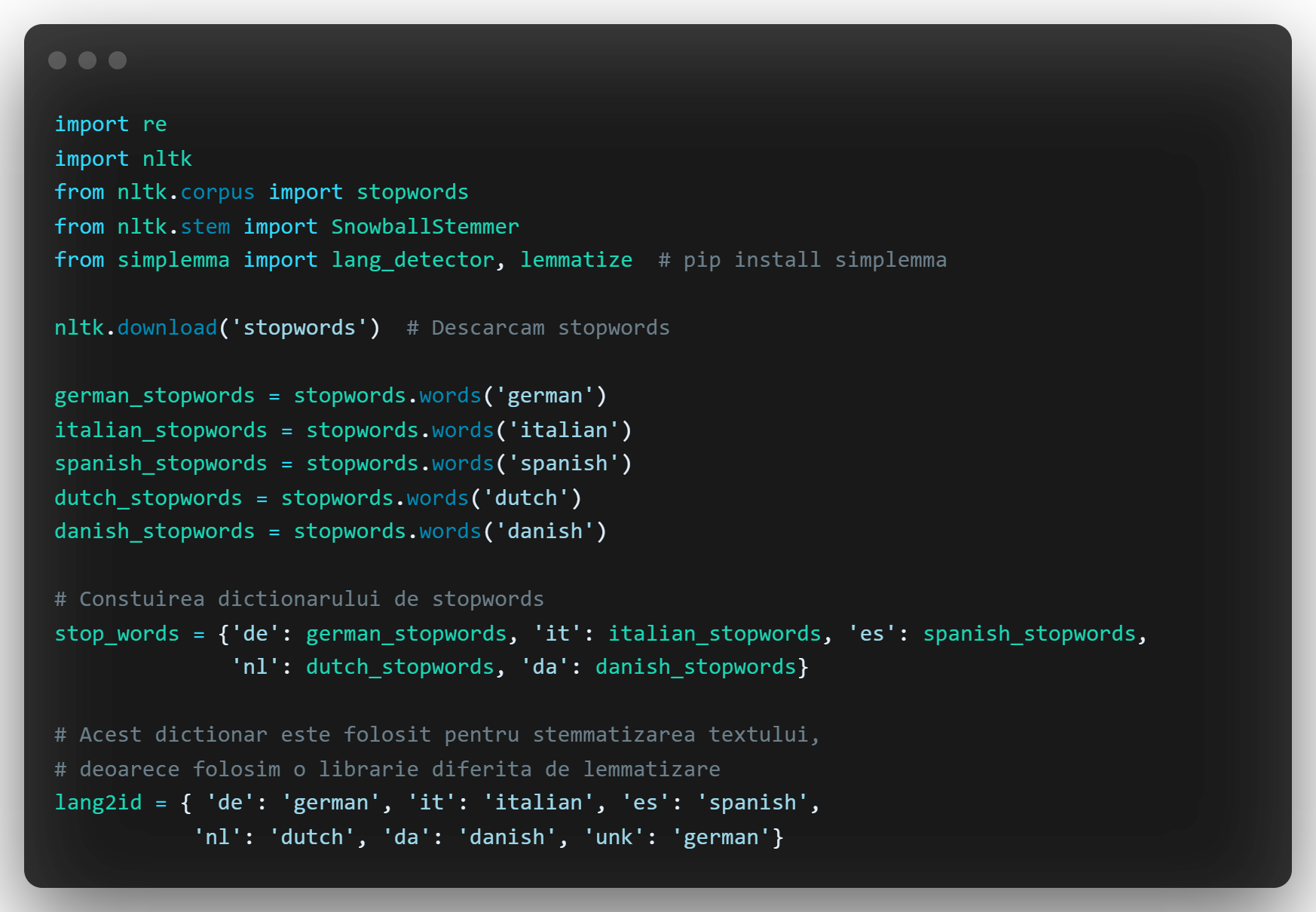


Pasul 1.3. Preprocesarea Datelor

Pentru a scăpa de caracterele inutile din text și a păstra cuvintele esențiale ce vor fi analizate de modelul nostru, vom folosi conceptul de preprocesare a textului. Putem efectua diferite metode de preprocesare a datelor de text astfel încât să păstrăm informațiile necesare pentru o estimare cât mai eficientă:

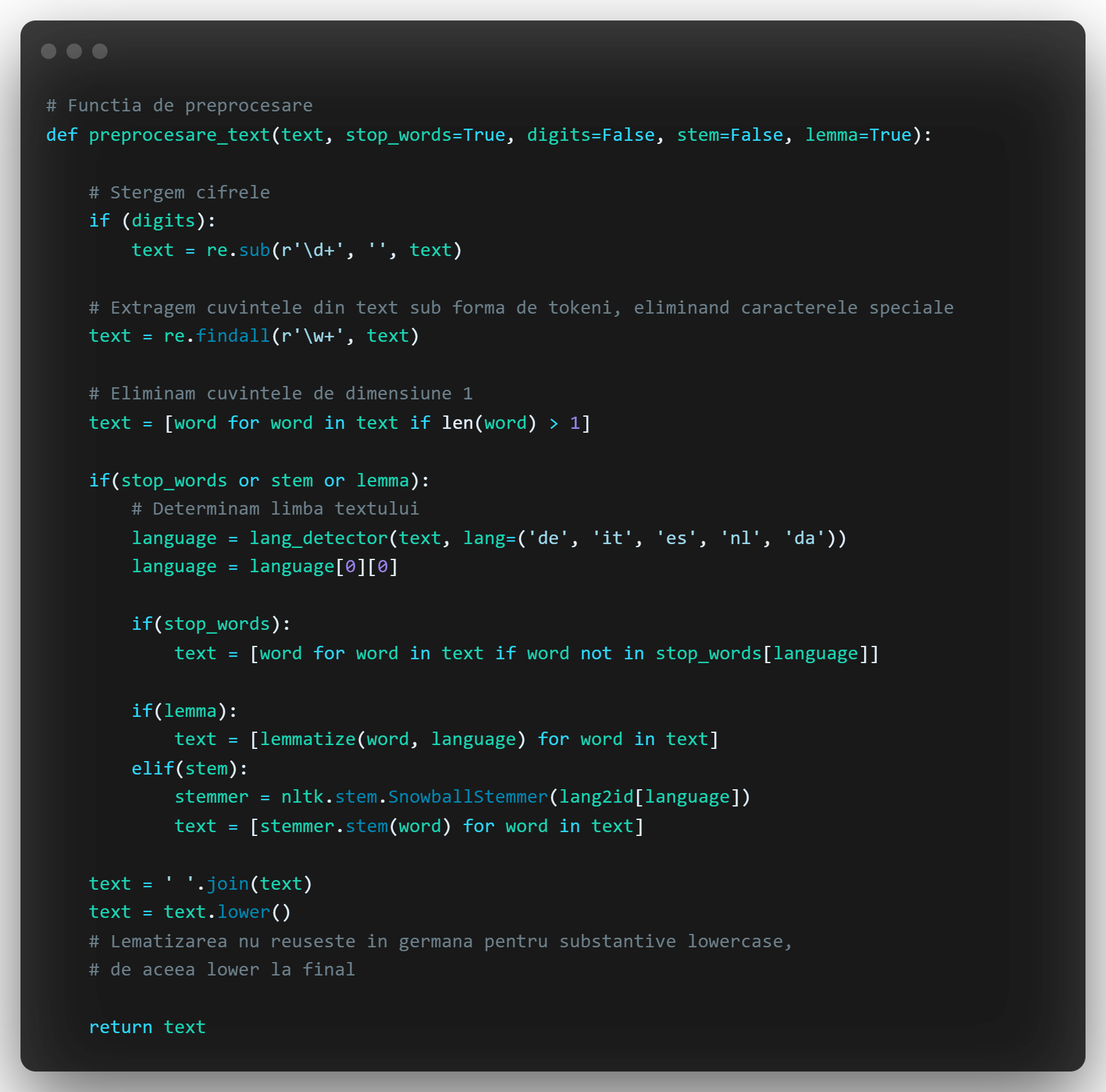
* ștergerea caracterelor speciale
* eliminarea **stopwords**
* **lematizarea/stematizarea** cuvintelor
* eliminarea cifrelor

Fiind un set de date destul de mare, nu ne putem da seama ce preprocesare este mai eficientă. Astfel, am ales să compun o funcție ce îmi va preprocesa datele în mai multe metode pentru a le putea compara și extrage pe cele mai optime.

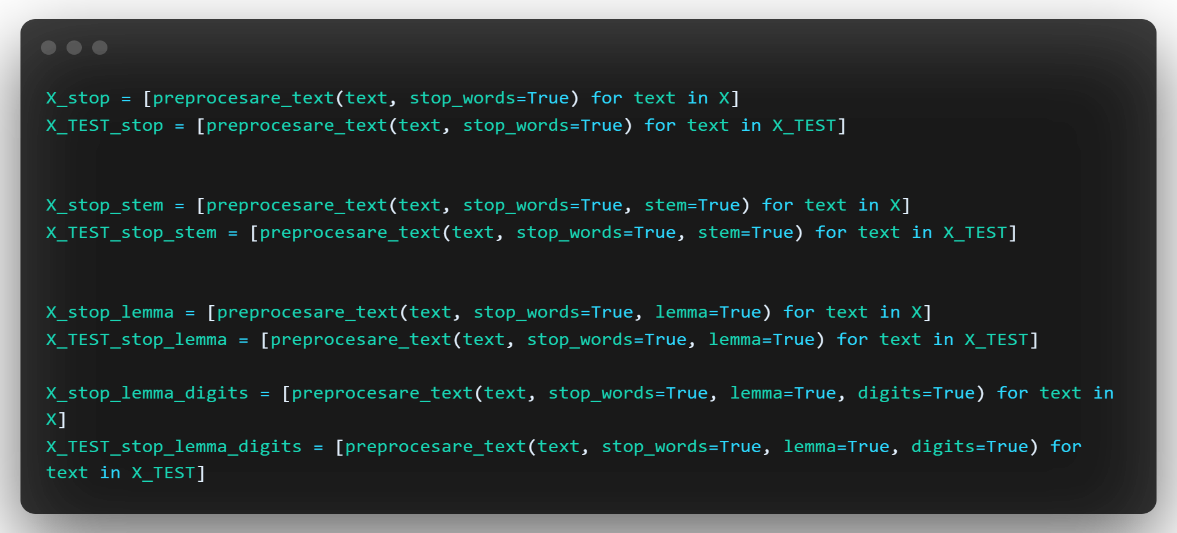


Lematizare vs Stematizare – ambele au scopul de a reduce formele flexionare și derivate ale unui cuvânt la o forma de baza comuna

Stematizarea se referă, de obicei, la un proces euristic brut care taie capetele cuvintelor în speranța atingerii corecte acestui obiectiv. Lematizarea se referă la a face lucrurile în mod corespunzător cu utilizarea unui vocabular și a unei analize morfologice a cuvintelor, urmărind în mod normal să elimine numai terminațiile flexionare și să returneze forma de baza sau de dicționar a unui cuvânt, care este cunoscută sub numele de lema.



Funcția este făcută astfel încât, să nu poți preprocesa utilizând lematizarea și stematizarea. Daca lemma și stem sunt ambele True, se va efectua doar lematizarea.

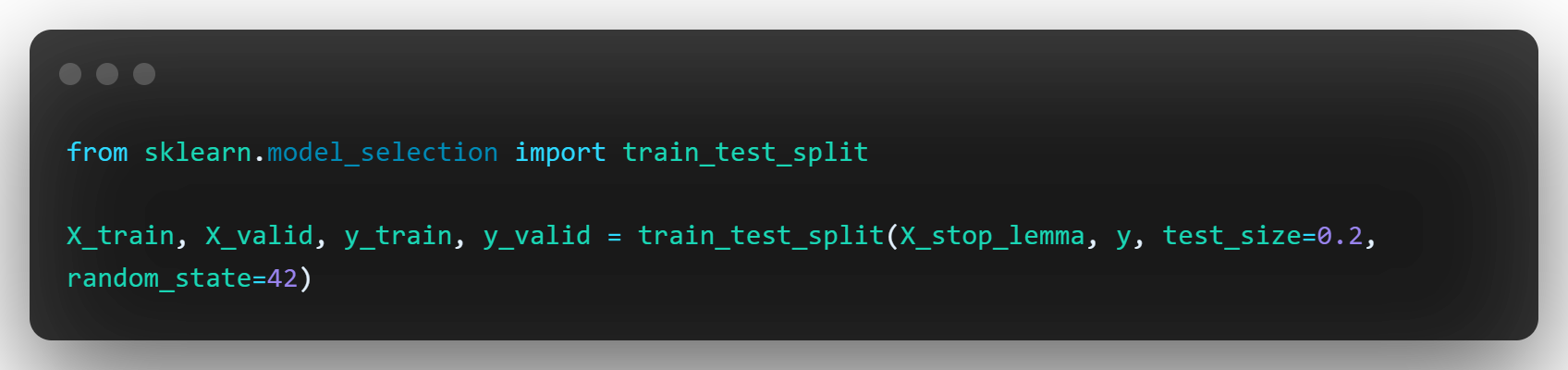


In continuarea acestui proiect, voi analiza setul de date ce folosește preprocesarea cu stopwords și lematizare (X\_stop\_lemma), deoarece s-a dovedit să fie cea mai performantă dintre preprocesări.

Timpii de preprocesare a datelor au fost de aprox: **20 min**. pentru datele de antrenare (41570 texte), **80 min**. pentru datele de test (13853 texte). Această discrepanță de timpi este redată de detectarea limbii la preprocesare. Datele de test, față de datele de antrenare, sunt amestecate în mod **aleator** în funcție de limbă (în train\_data fiind ordonate în funcție de limba), fapt care îngreunează procesul de identificare a limbii, dar și a lematizării.

Pasul 1.4. Train – Test – Validation

Pentru a diviza setul de date, sklearn ne pune la dispoziție modulul *model\_selection în* care avem funcția **train\_test\_split()**.



*Parametri*:

* X\_stop\_lemma 🡪 datele preprocesate
* y: 🡪 etichetele
* test\_size 🡪 dimensiunea datelor de test, în cazul nostru 20%
* random\_state 🡪 se comportă ca un seed, controlează modul de amestecare a datelor

Pasul 2.1. Bag Of **Words**

Conceptul „**Bag of Words**” este o reprezentare utilizată în procesarea limbajului natural. Procesul **BoW**, zis și „sac de cuvinte”, reprezintă strategia de a descrie documentele prin apariții de cuvinte, unde se ignoră gramatica și ordinea cuvintelor, păstrându-se doar frecvența acestora.

Principalele etape ale **BoW**:

* **tokenizarea** șirurilor și oferirea unui ID întreg pentru fiecare simbol posibil
* **numărarea** aparițiilor de cuvinte în fiecare document.
* **normalizarea** și ponderarea cuvintelor de importanță în scădere care apar în majoritatea documentelor.

Numim **vectorizare** procesul general de transformare a unei colecții de documente text în vectori de caracteristici numerice. (tokenizare, numărare, normalizare)

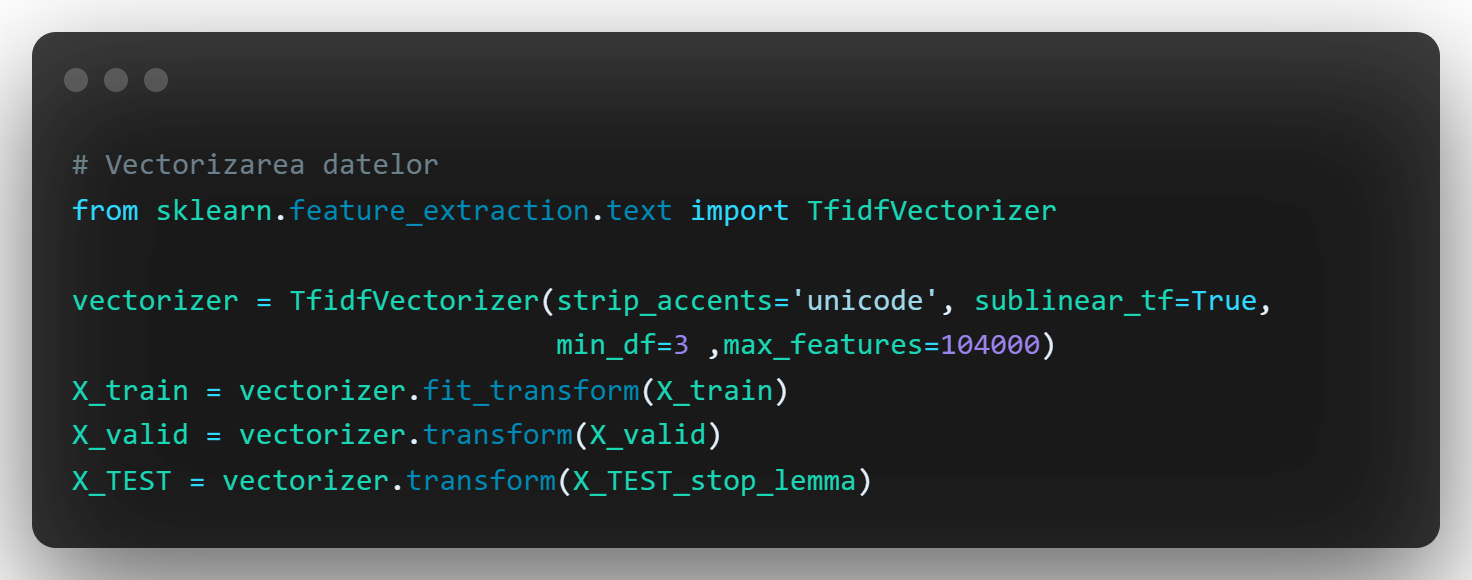
Folosind acest proces, putem converti o colecție de documente într-o matrice, fiecare document fiind un rând și fiecare cuvânt fiind coloana, iar valorile corespunzătoare (rând, coloană) fiind frecvența apariției fiecărui cuvânt în documentul respectiv.

În librăria **sklearn** exista 2 metode de implementare a unui BoW:

1. [CountVectorizer()](https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html)
2. [TfidfVectorizer()](https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html)

Într-un corpus de text mare, unele cuvinte vor fi foarte prezente (în special **cuvintele de legătură /stopwords**), transportând astfel foarte puține informații semnificative despre conținutul real al documentului. Dacă ar fi să transmitem datele de numărare directă direct unui clasificator, acei termeni foarte frecvenți **ar umbri** **frecvențele** termenilor mai rari, dar mai interesanți. Aici apare conceptul de **Tf-idf** (Term frequency – inverse document frequency).

Pasul 2.2. Vectorizarea Datelor cu TfidfVectorizer



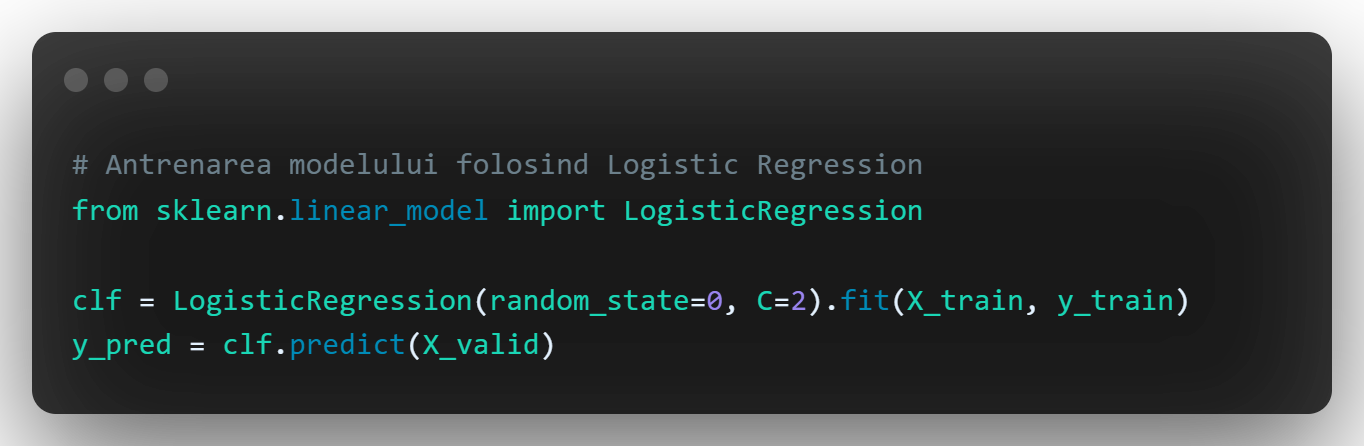
*Parametri*:

* **strip\_accents** 🡪 Normalizarea caracterelor {’ascii, ’unicode’}. Deoarece avem caractere din limbi   
  străine (ä, ö, ü, ø, æ, å) ce în normalizarea ’ascii’ ar dispărea, preferăm ’unicode’ pentru a păstra toate cuvintele în forma exactă.
* **sublinear\_tf** 🡪 Aplicarea scalării tf subliniară, adică înlocuirea tf cu 1 + log(tf).
* **min\_df**  🡪 Numărul minim de documente în care trebuie să apară un caracter/șir de caractere pentru a fi luat în considerare.
* **max\_features** 🡪 Păstrează primele max\_features cele mai relevante caracteristici ordonate în funcție de frecvența termenilor din corpus.

Pasul 3.1. Logistic Regression

**Regresia logistica** este un model statistic care folosește la bază **funcția logistică** pentru estimarea parametrilor unui model. În mod normal, regresia logistică clasifică un model binar (două clase).

In cazul nostru, unde avem mai multe clase, algoritmul de antrenare utilizează schema **One-vs-Rest** (OvR), ce ar putea reprezenta un clasificator puternic în antrenarea modelului nostru. În acest proiect vom folosi din modulul *sklearn.linear\_model* clasificatorul **LogisticRegression**.



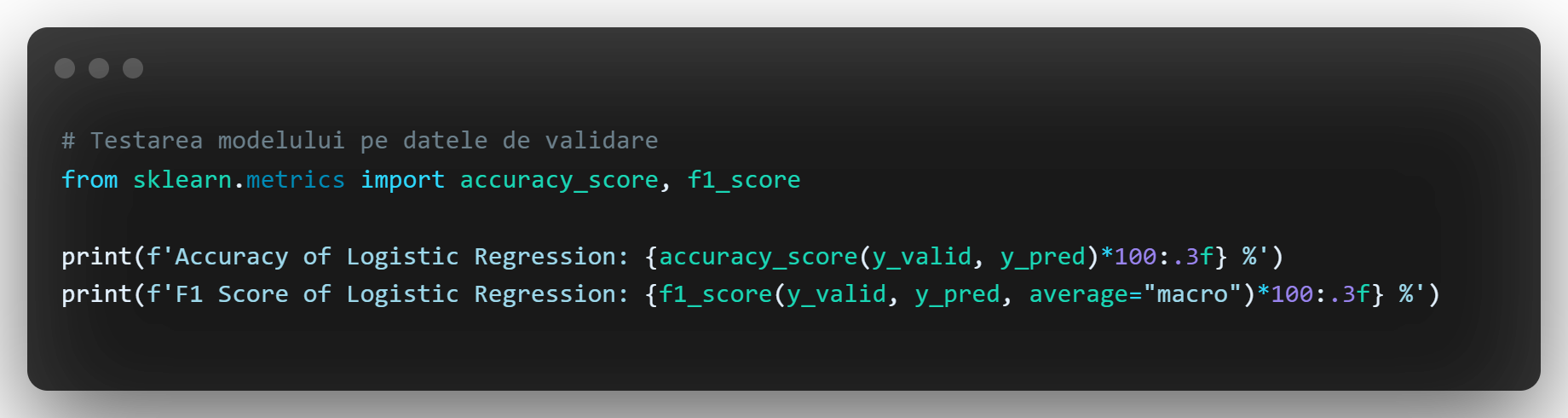
*Parametri*:

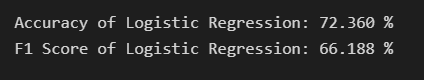
* C 🡪 **hiperparametru** ce controlează trade-off-ul dintre margine și acuratețe
* random\_state 🡪 se comportă ca un seed, controlează modul de amestecare a datelor

Antrenarea modelului durează în jur de 6.3 secunde.

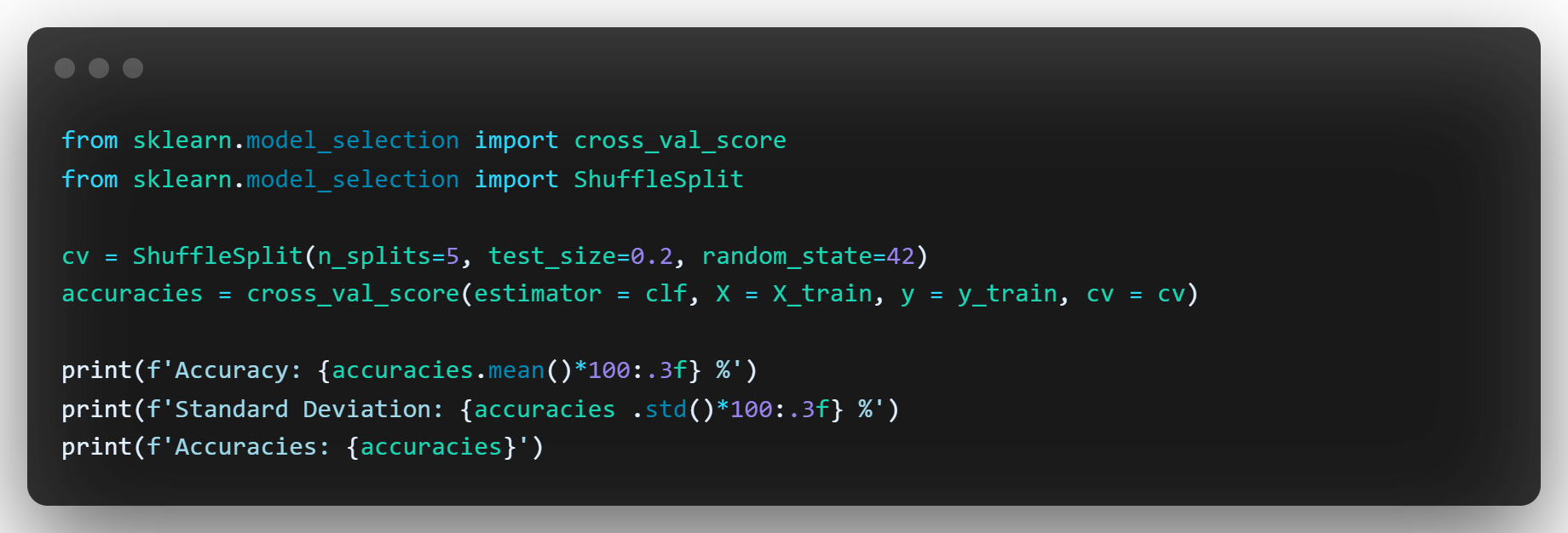
**Hiperparapemtrii** acestui proiect au fost aleși prin numeroase grid search-uri folosite din modulul *sklearn.model\_selection* cu funcția **GridSearchCV**.

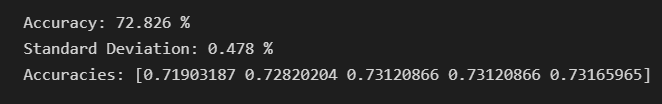
Pasul 3.2. Testarea modelului Logistic Regression



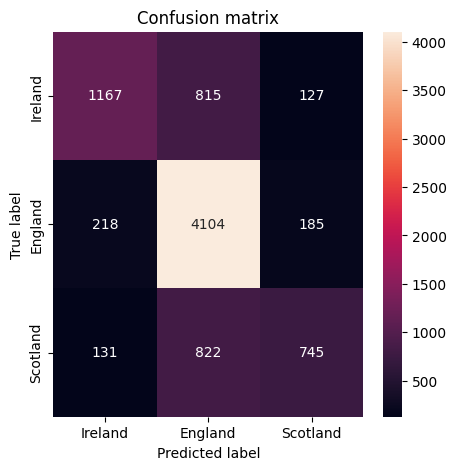


## 3.2.1. 5-Fold Cross-Validation





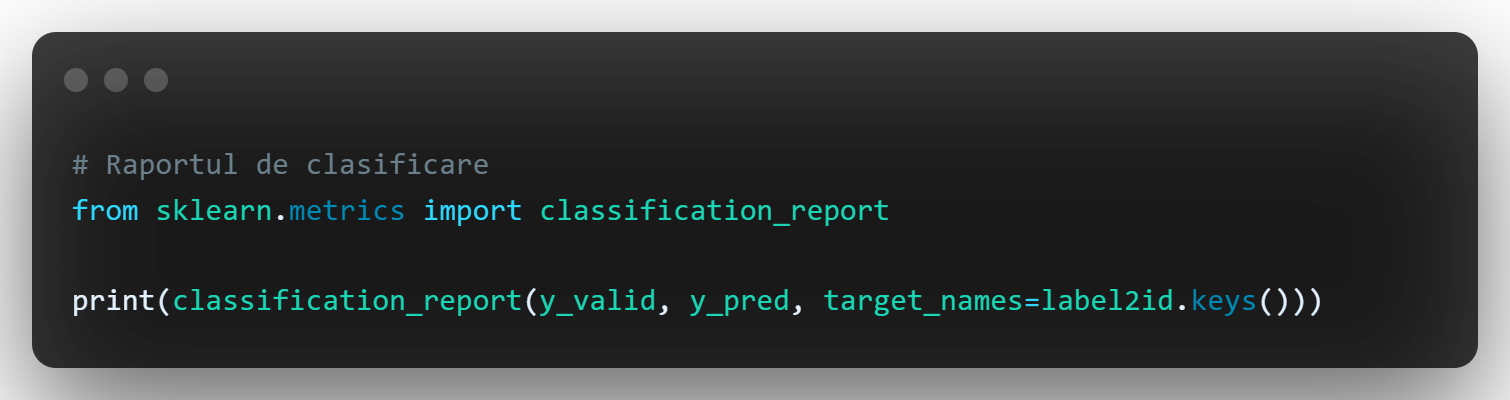
## 3.2.2. Matricea de confuzie

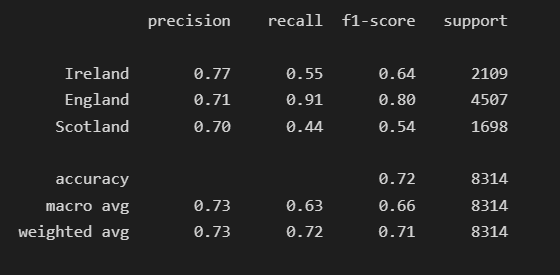


Putem observa cum majoritatea prezicerilor se îndreaptă spre Anglia. Acest fenomen apare deoarece majoritatea etichetelor din setul de date de antrenare aparțin Angliei, iar Logistic Regresiune, fiind un **algoritm probabilistic**, este de așteptat să influențeze predicția în funcție de probabilitatea de apariție a acestei clase.

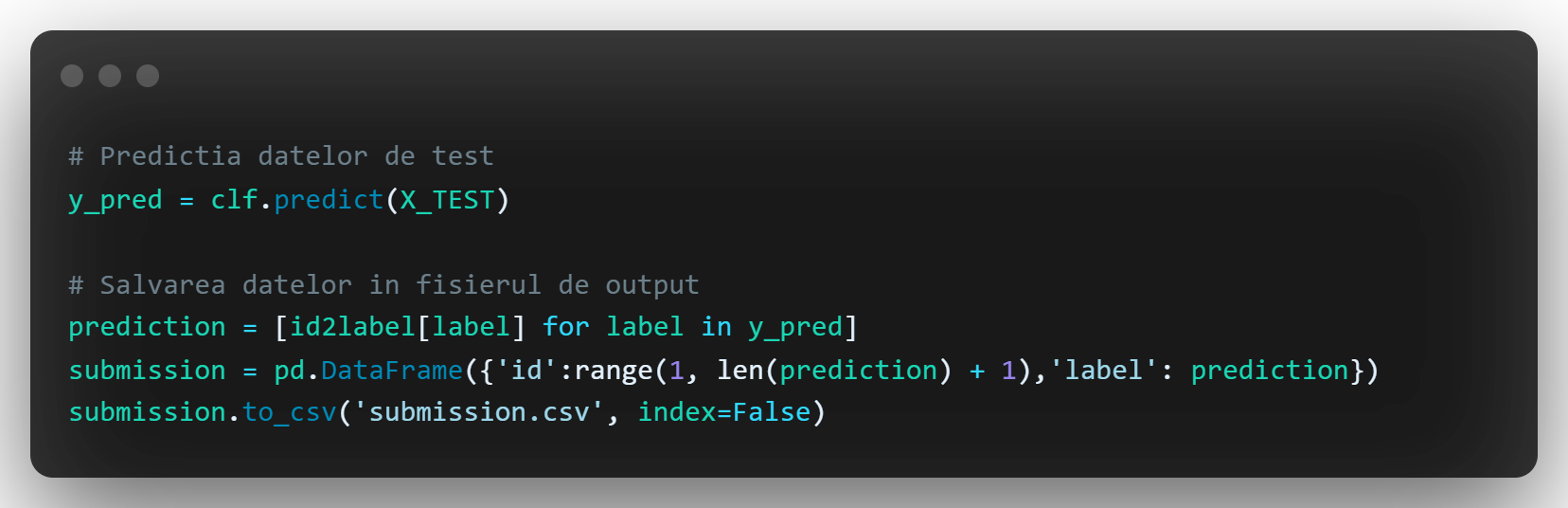
|  |  |
| --- | --- |
| Labels | Count |
| England | 22700 |
| Ireland | 10535 |
| Scotland | 8335 |

## 3.2.3. Raportul de clasificare





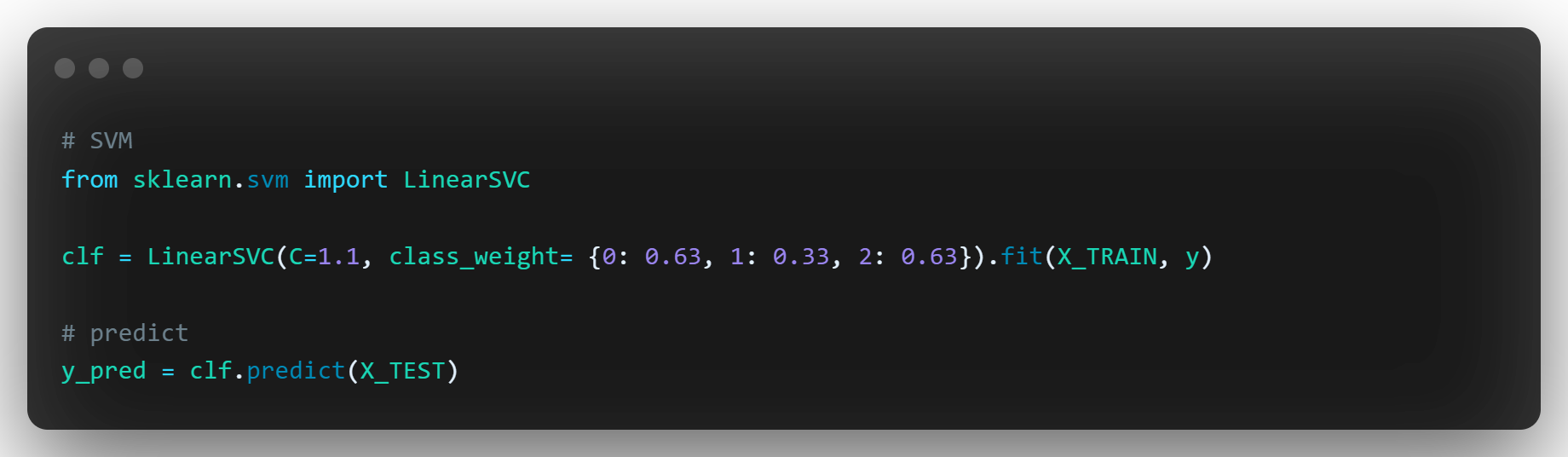
Pasul 3.3. Predicția datelor de test pe Kaggle



|  |  |
| --- | --- |
| Scorul Public (40% date test) | Scorul Privat |
| 0.70580 | 0.69901 |

Pasul 4.1. SVM

**SVM** este al doilea clasificator pe care l-am ales pentru rezolvarea acestui exercițiu. Din librăria sklearn folosim **LinearSVC** – Linear Support Vector Classification. Similar **SVC**-ului, este implementat în termeni lineari, ce ii oferă mai multa flexibilitate în alegerea penalităților și a funcțiilor de pierdere, scalând mai bine la un număr mare de date.

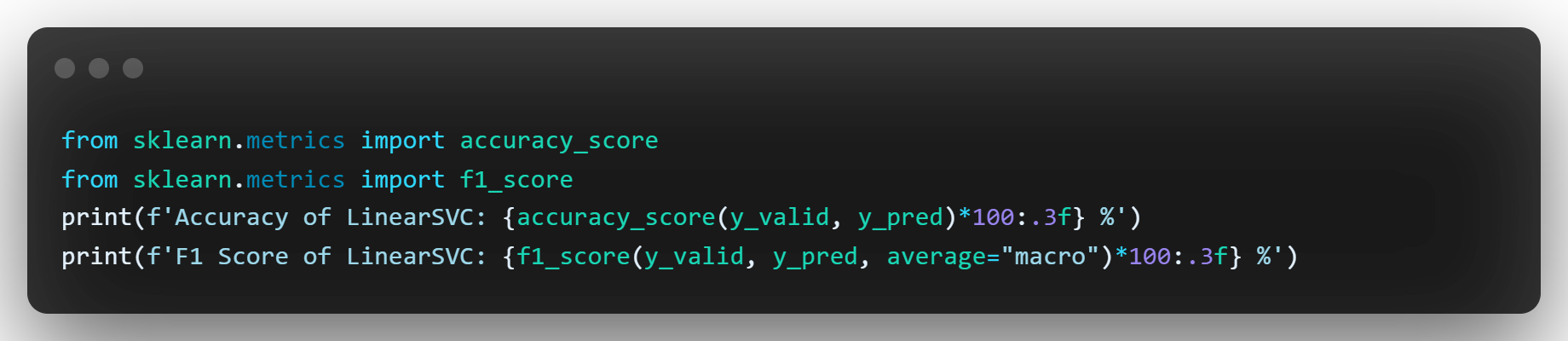


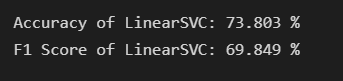
*Parametri*:

* C 🡪 **hiperparametru** ce controlează trade-off-ul dintre margine și acuratețe
* class\_weight 🡪 greutatea claselor, , folosim în special din cauza discrepanței dintre etichete

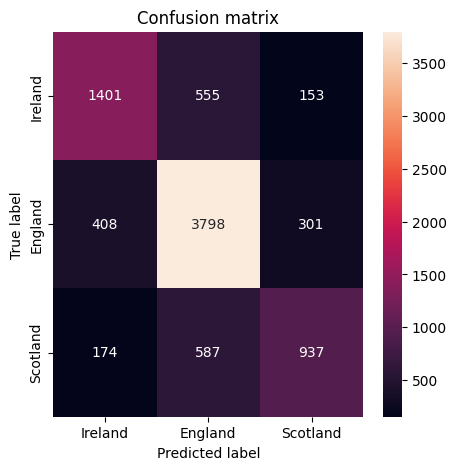
Timpul de antrenare al clasificatorului este de aprox. 1.9s

Pasul 4.2. Testarea modelului SVM

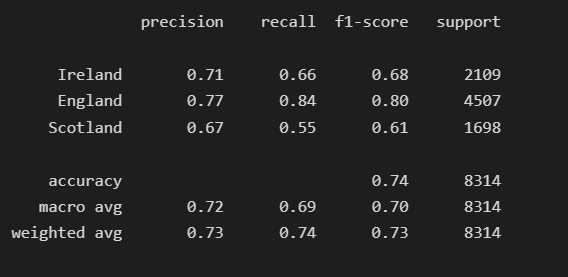




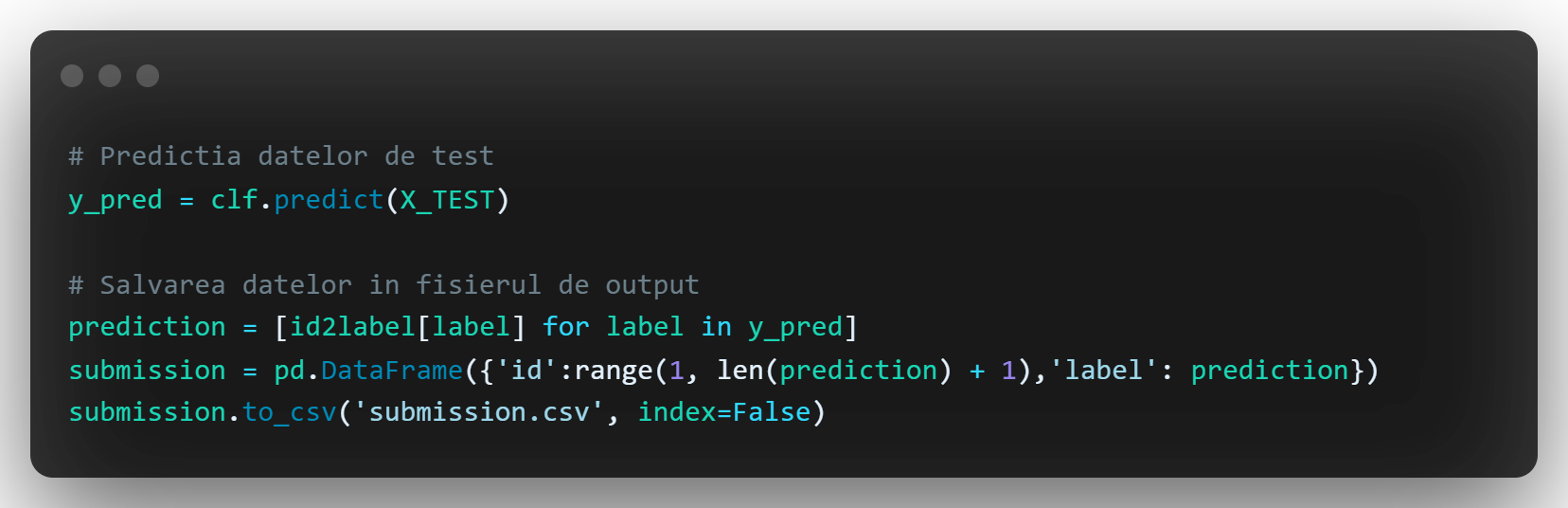
## 4.2.1. Matricea de confuzie



## 4.2.2. Raportul de clasificare



4.3. Predicția datelor de test pe Kaggle



|  |  |
| --- | --- |
| Scorul Public (40% date test) | Scorul Privat |
| 0.69336 | 0.68939 |

După această întreagă analiză, observăm cum SVM-ul are un scor mai mare pe datele locale, iar pe Kaggle a scos un punctaj mai mic decât Logistic Regression, care la rândul lui a obținut un scor mai mic local decât SVM-ul.