Facultatea de Electronică, Telecomunicații și Tehnologii Informaționale

Licență 2023

Universitatea Politehnica Timișoara

SOLUȚII PENTRU IMPLEMENTAREA ÎN PRODUCȚIE A MODELELOR DE ÎNVĂȚARE AUTOMATĂ

Candidat: Petru-Alin Avram

Coordonator științific: Sl.dr.ing. Marian Bucos

Sesiunea: Iunie 2023

Facultatea de Electronică, Telecomunicații
și Tehnologii Informaționale

Universitatea Politehnica
Timișoara

ANEXA 1

Nume și prenume student: **Avram Petru-Alin**

Nume și prenume coordonator: **BUCOS MARIAN**

Titlul lucrării: **Soluții pentru implementarea în producție a modelelor de învățare automată**

Lucrarea trebuie finalizată ca:

- studiu teoretic
- implementare de metodă/algoritm într-un limbaj de programare

Structură lucrare:

Introducere
Studii/ lucrări similare
Implementare
Rezultate
Discuție
Contribuții
Concluzii

Data: 27.11.2022

Semnătură coordonator:

Semnătură student:

ANEXA 2

Nume și prenume student: **Avram Petru-Alin**

Scopul lucrării:

Scopul acestui proiect este de a arăta cum poate fi implementat un model de învățare automată în producție, folosind diverse tehnici și unelte de MLOps.

Cuvinte cheie:

Învățare automată, python, bentoml, model, spark, batch, date, mlops

Cuprinsul proiectului (titluri capitole):

Referințe bibliografice (3 lucrări semnificative):

[1] VibeThemes, “What is Machine Learning? | Machine Learning Techniques |

GangBoard," Online Courses with Experts for AWS, Python, Data Science, Selenium, RPA, Devops - More, May 20, 2019. https://www.gangboard.com/blog/what-is-machine-learning (accessed Mar. 18, 2023).
[2] G. Symeonidis, E. Nerantzis, A. Kazakis, and G. A. Papakostas, "MLOps -- Definitions, Tools and Challenges." arXiv, Jan. 01, 2022. Accessed: Apr. 11, 2023. [Online]. Available: http://arxiv.org/abs/2201.00162
[3] R. Subramanya, S. Sierla, and V. Vyatkin, "From DevOps to MLOps: Overview and Application to Electricity Market Forecasting," Applied Sciences, vol. 12, no. 19, p. 9851, Sep. 2022, doi: 10.3390/app12199851.
[4] "The Big Book Of MLOps," Databricks. https://www.databricks.com/explore/data-science-machine-learning/big-book-of-mlops (accessed Apr. 09, 2023).
[5] P. Ingle, "Top Machine Learning Model Deployment Tools For 2022," MarkTechPost, Oct. 14, 2022. https://www.marktechpost.com/2022/10/14/top-machine-learning-model-deployment-tools-for-2022/ (accessed Mar. 18, 2023).
[6] "A 4-Step Guide to Machine Learning Model Deployment." https://www.dominodatalab.com/blog/machine-learning-model-deployment (accessed Mar. 18, 2023).
[7] "What is Data Preparation? An In-Depth Guide to Data Prep," Business Analytics. https://www.techtarget.com/searchbusinessanalytics/definition/data-preparation (accessed Apr. 10, 2023).
[8] "What is Exploratory Data Analysis? | IBM." https://www.ibm.com/topics/exploratory-data-analysis (accessed Apr. 10, 2023).
[9] H. Patel, "What is Feature Engineering — Importance, Tools and Techniques for Machine Learning," Medium, Sep. 02, 2021. https://towardsdatascience.com/what-is-feature-engineering-importance-tools-and-techniques-for-machine-learning-2080b0269f10 (accessed Apr. 10, 2023).
[10] S. Alla and S. K. Adari, Beginning MLOps with MLFlow: Deploy Models in AWS SageMaker, Google Cloud, and Microsoft Azure. Berkeley, CA: Apress, 2021. doi: 10.1007/978-1-4842-6549-9.
[11] A. Grigorev, "MLOps in 10 Minutes," Medium, May 04, 2022. https://towardsdatascience.com/mlops-in-10-minutes-165c746a9b8e (accessed Apr. 05, 2023).
[12] "Model Deployment: 3 Steps - Top Tools in 2023." https://research.aimultiple.com/model-deployment/ (accessed Apr. 13, 2023).
[13] A. Luigi, "Batch Inference for Machine Learning Deployment (Deployment Series: Guide 03)," ML in Production, Feb. 19, 2020. https://mlinproduction.com/batch-inference-for-machine-learning-deployment-deployment-series-03/ (accessed Apr. 13, 2023).
[14] N. Gift and A. Deza, Practical MLOps. O'Reilly Media, Inc., 2021.
[15] Asigmo, "MLflow best practices and lessons learned," Asigmo, May 01, 2020. https://www.asigmo.com/post/mlflow-best-practices-and-lessons-learned (accessed Apr. 22, 2023).
[16] Y. Callaert, "Getting started with mlFlow," Medium, Aug. 26, 2019.

https://towardsdatascience.com/getting-started-with-mlflow-52eff8c09c61 (accessed Apr. 23, 2023).
[17] K. Kaewsanmua, “Best 8 Machine Learning Model Deployment Tools That You Need to Know,” neptune.ai, Jul. 21, 2022. https://neptune.ai/blog/best-8-machine-learning-model-deployment-tools (accessed Apr. 23, 2023).
[18] “MLflow Tracking — MLflow 2.3.0 documentation.” https://mlflow.org/docs/latest/tracking.html#concepts (accessed Apr. 23, 2023).
[19] “MLflow Projects — MLflow 2.3.0 documentation.” https://mlflow.org/docs/latest/projects.html (accessed Apr. 23, 2023).
[20] “MLflow Models — MLflow 2.3.1 documentation.” https://mlflow.org/docs/latest/models.html (accessed May 02, 2023).
[21] “MLflow Model Registry — MLflow 2.3.1 documentation.” https://mlflow.org/docs/latest/model-registry.html (accessed May 02, 2023).
[22] “MLflow - A platform for the machine learning lifecycle,” MLflow. https://mlflow.org/ (accessed Apr. 22, 2023).

Activități desfășurate (o prezentare succintă, numerotată):

Până în acest moment am scris doar despre partea teoretică a lucrării și anume ceea ce este în cuprins. Partea practică va fi implementată în zilele ce urmează.

Data:

Semnătură coordonator: Semnătură student:

CUPRINS

1. SINTEZĂ LUCRARE

În această lucrare voi aborda o temă actuală din domeniul învățării automate și anume găsirea de soluții pentru implementarea în producție a modelelor de învățare automată.

Sunt mulți oameni care excelează în crearea unor modele care pot prezice date din lumea reală, însă implementarea lor reprezintă o provocare și asta reiese din faptul că cele mai multe dintre ele nu ajung în producție.

Un om de știință a datelor trebuie să colecteze date, să le analizeze pentru a le identifica caracteristicile cheie, urmând ca mai apoi să le împartă în mai multe seturi. Aceste seturi de date vor fi folosite ulterior pentru antrenarea, testarea și validarea modelului. După toate aceste procese, urmează faza de implementare și monitorizare, unde putem vedea performanțele modelului și comportarea sa atunci când primește date noi, neprocesate.

Din fericire, au fost dezvoltate o mulțime de instrumente care ne simplifică munca în ceea ce privește implementarea modelelor în producție și nu numai. Multe dintre ele gestionează întregul ciclu de viață al învățării automate. MLflow, Amazon SageMaker, Seldon, Kubeflow, BentoML sunt câteva dintre instrumentele folosite pentru implementare, fiecare având avantajele și dezavantajele sale.

Cu MLflow am să vă arăt cât este de simplu să parcurgem toate aceste procese. Este un instrument care nu are multe cerințe pentru a putea fi folosit, complexitatea și costul utilizării sunt reduse, iar framework-urile de care dispune sunt printre cele mai utilizate în acest domeniu. Cuprinde patru componente principale care ne ajută să urmărim valorile modelului pentru a putea compara performanțele sale, să ținem evidența diverșilor parametri de înregistrare, a versiunilor de cod și a fișierelor generate, una dintre componente fiind chiar un depozit centralizat de modele.

Astfel, pornind de la un set de date preluat de la o companie de telecom voi ajunge să creez mai multe modele de învățare automată, iar pe baza performanțelor lor voi alege care este mai bun. MLflow dispune de o interfață de utilizator ușor de folosit și putem să vizualizăm experimentele, rulările și modelele înregistrate, precum și valorile calculate.

2. INTRODUCERE

2.1. ÎNVĂȚAREA AUTOMATĂ

Învățarea automată este una dintre ramurile inteligenței artificiale care dezvoltă algoritmi și modele statistice cu scopul de a ajuta calculatoarele să facă propriile decizii fără ca omul să intervină. Ele sunt învățate cum să se comporte și cum să învețe din experiență, bazându-se doar pe datele primite, fără a fi programate într-un mod explicit.

ML are numeroase aplicații practice, așa cum se poate vedea în figura de mai jos:

Fig. 2.1. Domenii de utilizare ale învățării automate [1].

Începând cu secolul XXI, lumea a început treptat să se împartă în oameni și mașini. Noi oamenii, prin felul în care am fost construiți, învățăm mereu lucruri noi, ne adaptăm cu ușurință la schimbări, ținem cont de experiențele trecute și reușim să evoluăm pe zi ce trece. Asta se încearcă și cu mașinile: să învețe și să evolueze constant tocmai pentru a ne ușura nouă munca.

Învățarea automată se folosește de algoritmi pentru a îmbunătăți performanța calculatoarelor. Acești algoritmi sunt antrenați cu cantități mari de date tocmai pentru ca atunci când au de-a face cu date noi, ei să poată să facă legături și predicții și pentru a putea lua decizii.

Calculatorul învață și construiește un model pe baza datelor transmise, iar în momentul în care sunt introduse date noi, acesta reușește să facă predicții pe baza celor învățate deja [1].

2.2. MLOPS

Cum învățarea automată a trecut de la cercetări academice la rezolvarea de probleme din lumea reală, găsirea de soluții pentru implementarea în producție a modelelor de învățare automată reprezintă un subiect foarte important. Atât crearea, cât și implementarea modelelor de învățare automată trec printr-un proces complex, aici intervenind MLOps [3].

Ce este MLOps? Pe scurt, MLOps = ML + DEV + OPS.

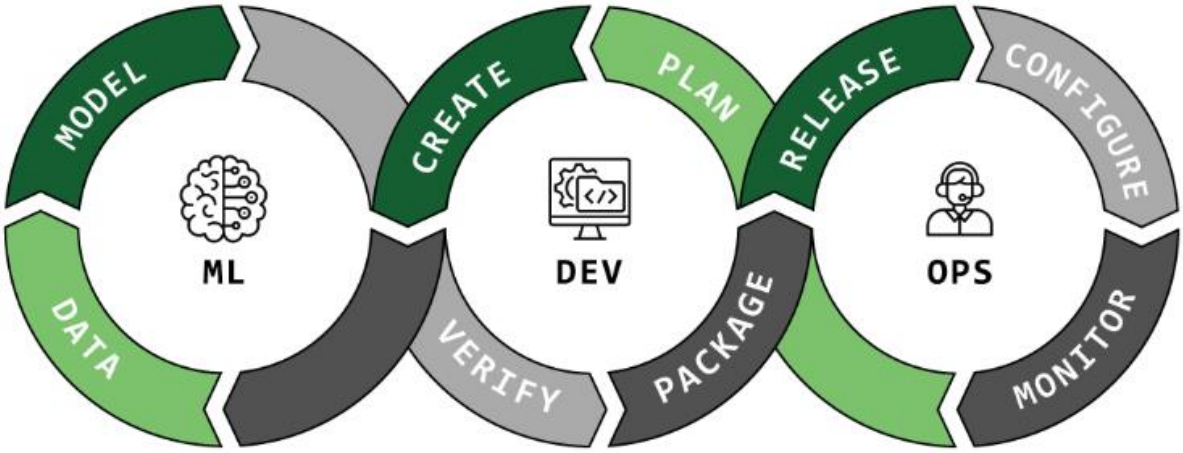

Fig. 2.2. Ciclul de viață MLOps [2].

Este o colecție de proceduri și automatizări folosită la gestionarea atât a modelelor și datelor, cât și a codului pentru a îmbunătăți sistemele ML din punct de vedere al stabilității performanței și al eficienței [4].

Implementarea în producție a modelelor de învățare automată reprezintă, pentru cei mai mulți, partea cea mai dificilă, fiind o provocare în ceea ce privește știința datelor. Aceasta presupune integrarea într-un mediu de producție existent a unui model de învățare automată care acceptă intrări și oferă rezultate pentru a lua decizii utile bazate pe date [5].

Cu toate că implementarea este doar a treia etapă a ciclului de viață al științei datelor (gestionare, dezvoltare, implementare și monitorizare), fiecare pas al creării unui model are în vedere implementarea [6].

De obicei, modelele sunt dezvoltate în contextul unor pachete de date pregătite cu meticulozitate, care au fost atât instruite, cât și testate [6]. Într-un proces MLOps sunt parcurși 3 pași în ceea ce privește datele:

- pregătirea datelor;
- analiza detaliată a datelor;
- ingineria caracteristicilor [4].

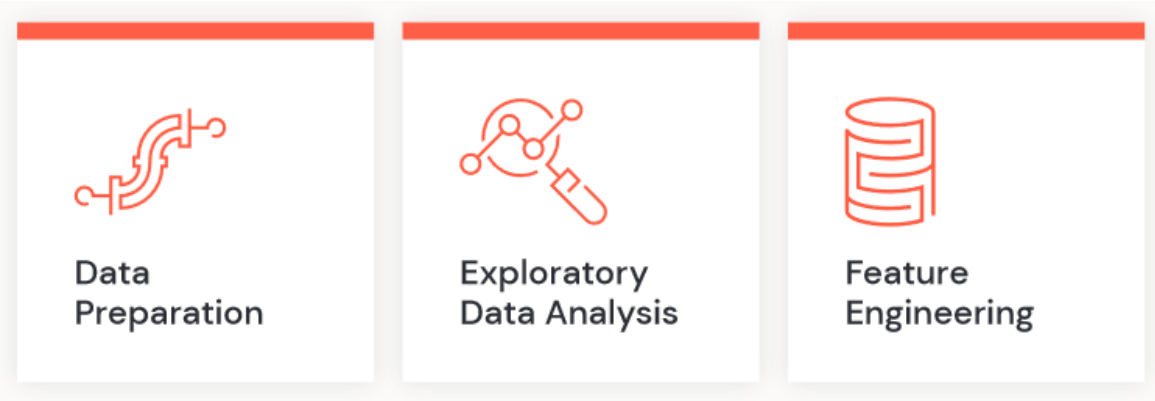

Fig. 2.3. Procesul datelor [4].

2.2.1. Pregătirea datelor

Pregătirea datelor este procesul în care sunt strânse, structurate și organizate datele. Profilarea, preprocesarea, validarea și curățarea datelor fac toate parte din pregătirea datelor. De multe ori implică și fuzionarea datelor din mai multe surse interne și externe [7].

Asigurarea faptului că datele brute sunt corecte și consecvente înainte de analiză și procesare este unul dintre obiectivele principale ale acestui proces. Pe măsură ce datele sunt create, acestea vor include frecvent erori precum valori lipsă sau inexactități și astfel o bună parte din sarcinile procesului de pregătire a datelor implică corectarea acestor erori și confirmarea calității datelor [7].

2.2.2. Analiza detaliată a datelor

Această analiză este folosită de oamenii de știință pentru a examina și analiza seturi de date pentru a identifica care sunt caracteristicile cheie ale datelor respective. Extrag aceste caracteristici pentru a putea crea modele, pentru a putea testa sau verifica diverse ipoteze și este o metodă eficientă de înțelegere a tiparelor ce se găsesc în cadrul datelor [8].

2.2.3. Ingineria caracteristicilor

Este procesul prin care sunt alese, modificate și convertite date neprocesate, brute în caracteristici [9]. Aceste caracteristici pot fi folosite în învățarea supravegheată și se împart în mai multe seturi: de instruire, testare, validare [4]. Altfel spus, ingineria caracteristicilor este o tehnică de ML care se folosește de date pentru a introduce noi variabile în setul de antrenament al modelelor pentru a le spori acuratețea [9].

O mare majoritate a modelelor produse în faza de dezvoltare nu se ridică la nivelul așteptărilor. Doar câteva modele trec de partea de test, iar acestea necesită un

angajament financiar semnificativ. Prin urmare, introducerea cu succes a unui model într-un mediu dinamic poate necesita atât o planificare atentă, cât și o pregătire adecvată [6].

Următorii pași sunt necesari atunci când vine vorba despre implementarea unui astfel de model în producție:

- antrenarea modelului;
- validarea modelului;
- implementarea modelului;
- monitorizarea modelului.

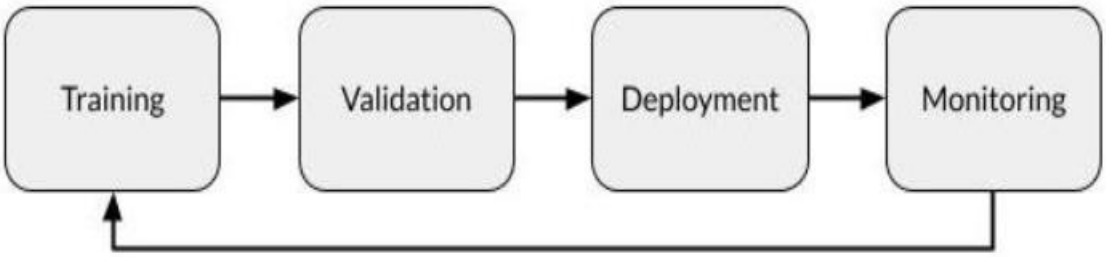

Fig. 2.4. Pași implementare model de învățare automată în producție [6].

2.2.4. Antrenarea modelului

Un model de învățare automată trebuie instruit înainte de a putea fi implementat. Aceasta presupune alegerea unui algoritm, definirea parametrilor acestuia și antrenarea sa folosind date curate și foarte bine pregătite [6].

Întregul proces se desfășoară într-un mediu corespunzător, care de cele mai multe ori este o platformă creată doar pentru cercetare, având resursele și instrumentele potrivite pentru o astfel de activitate [6].

În urma acestui antrenament se va determina care este modelul cel mai bun pe baza metricilor de evaluare [4].

2.2.5. Validarea modelului

Modelul trebuie să fie validat odată ce a fost antrenat și rezultatele au ieșit bune pentru a ne asigura că nu a fost un succes întâmplător, aberant. Modelul este validat prin retestarea lui, însă de data aceasta folosind date noi, comparând noile rezultate cu ceea ce a învățat în timpul antrenării. Testarea modelului într-un mediu special creat ajută la identificarea erorilor și rezolvarea lor, înainte ca acestea să afecteze producția [6].

Multe modele sunt adesea instruite, însă doar un număr mic dintre ele sunt validate. Pentru ca un model să fie validat, el trebuie să depășească un nivel standard de

performanță [4] și de obicei, numai cel mai eficient model din cele validate va fi implementat mai departe [6].

Examinarea materialelor de instruire este un alt pas în procesul de validare pentru a ne asigura că tehnica este una acceptabilă și corectă, mai ales pentru companii care au reguli și procese foarte bine definite, și că datele care au fost folosite se potrivesc cerințelor utilizatorilor [6].

2.2.6. Implementarea modelului

Implementarea propriu-zisă a modelului implică o serie de acțiuni:

- mutarea modelului în mediul în care acesta a fost dezvoltat, unde va dispune de sursa de date și de resursele hardware;
- integrarea modelului într-un proces (folosind, de exemplu, un API);
- instruirea utilizatorilor modelului: cum îl activează, cum i se accesează datele și cum evaluează rezultatele [6].

2.2.7. Monitorizarea modelului

După ce modelul a fost implementat cu succes în producție, începe faza de monitorizare a performanțelor modelului. Prin această monitorizare se asigură faptul că modelul are o funcționare corectă, cu predicții eficiente, timp de răspuns bun și rată de erori scăzută [6].

Prin această monitorizare se pot depista din timp toate erorile și permite revenirea la pașii anteriori de implementare pentru corectare. În caz contrar, pot apărea diverse erori (resurse neadecvate, flux de date conectat necorespunzător etc) care vor duce la degradarea performanței [4].

Evident, nu doar modelul trebuie monitorizat. Trebuie să se verifice că software-ul și resursele folosite funcționează corespunzător cerințelor [6].

Chiar dacă toate acestea au fost verificate, monitorizarea trebuie să continue. Asta presupune evaluarea regulată a performanței modelului în mediul său de implementare, adică un proces automat care să urmărească valorile pentru a putea observa când sau dacă apar schimbări în ceea ce privește precizia, acuratețea [6].

3. TIPURI DE IMPLEMENTARE ȘI INSTRUMENTE MLOPS

3.1. IMPLEMENTAREA MODELULUI: BATCH ȘI ONLINE

În funcție de cazul lor de utilizare, modelele pot fi implementate în două moduri: batch și online.

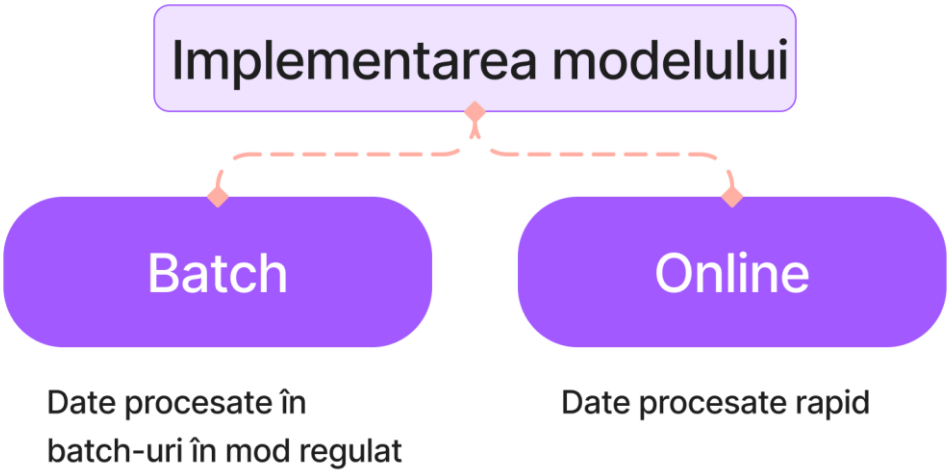

Fig. 3.1. Modurile de implementare ale modelelor după cazul de utilizare.

Modul batch este modul în care modelul este rulat periodic, iar rezultatele vin cu o oarecare latență. Procesarea datelor se face într-un mod regulat (din oră în oră, o dată pe zi, o dată pe săptămână etc.) în batch-uri, nu trebuie să răspundem la toate datele noi imediat [11, p. 10].

Este un mod simplu și foarte folositor atunci când nu este nevoie de rezultatele modelului în timp real. Avantajul este că pot fi construite modele complexe care să ofere rezultate precise din moment ce nu există vreo constrângere de latență [12].

De exemplu, presupunem că avem o companie de comerț electronic. Această companie trimite e-mailuri cu recomandări de servicii sau produse către clienții săi la începutul și finalul fiecărei săptămâni: luni și vineri dimineața, respectând fusurile orare locale. Recomandările vor fi generate pentru toată clientela, deci toți clienții vor contribui la crearea batch-ului de mostre. Este suficient ca predicțiile să se facă de două ori pe săptămână, înainte de e-mailurile de luni și vineri dimineața [13].

Modul online se află în contrast cu batch, deoarece trebuie să răspundem imediat la datele noi ce apar. Asta îl face mai complex pentru că are nevoie de funcționarea continuă a serviciilor modelului pentru a fi mereu pregătite de procesarea noilor date [11].

Având în vedere că trebuie să ofere rezultatele utilizatorilor finali în timp real, acest mod limitează complexitatea modelelor utilizate. Pe deasupra, modelul trebuie să fie pregătit de rulare oricând, iar asta presupune o solicitare mai mare din punct de vedere computațional [12].

Pentru exemplificare, să considerăm o aplicație de uber care oferă o estimare aproximativă a timpului în care șoferul ajunge să preia clientul. Pentru că nu se cunoaște nicio modalitate de a afla în avans ce client va comanda un uber și de unde, din ce locație, modelul ML trebuie implementat cu metoda online întrucât va oferi rezultate în timp real.

Atunci când nu știm ce metodă să folosim, încercăm să răspundem la întrebări precum:

- Folosim des predicțiile modelului?
- Pe ce ar trebui să se bazeze rezultatele modelului? Pe un batch de date sau cazuri individuale?
- Ce cantitate de putere de procesare putem oferi modelului [12]?

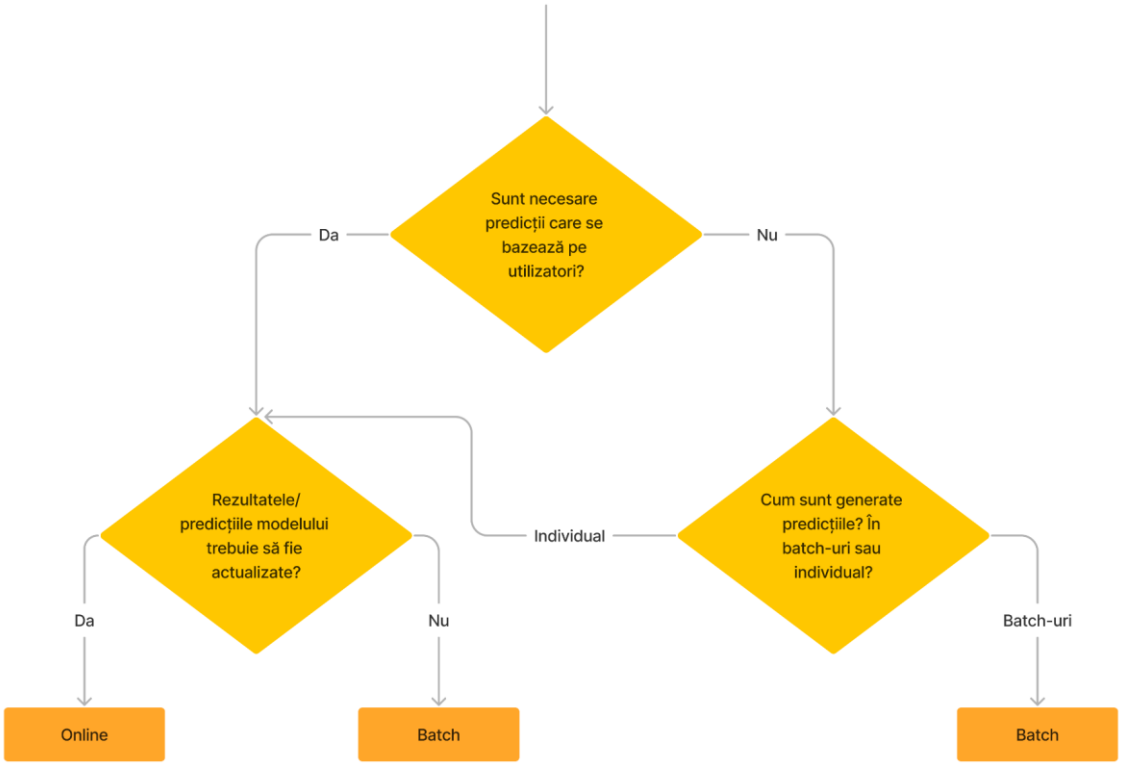

Fig. 3.2. Arbore decizional pentru modul de implementare al modelului.

Indiferent de caz, ni se oferă o gamă largă de instrumente cu ajutorul cărora ne este simplificată munca în ceea ce privește aplicarea principiilor MLOps și pot fi utilizate de oricine: MLflow, Amazon SageMaker, TensorFlow Serving, Kubeflow, BentoML, Torch Serve, Seldon, Cortex și multe altele [10].

În cele ce urmează, voi discuta despre câteva dintre aceste instrumente pentru a vedea care sunt principalele lor caracteristici, punctele slabe, punctele forte și cazurile lor de utilizare.

3.2. INSTRUMENTE MLOPS

În momentul de față există multe instrumente care folosesc una dintre cele două metode și sunt mai mulți factori decisivi în ceea ce privește alegerea unui instrument. De exemplu, ar fi natural dacă o organizație care utilizează serviciul de stocare Amazon S3 pentru a-și colecta datele să opteze pentru Amazon SageMaker [14].

În continuare am să vorbesc despre câteva dintre instrumentele des folosite de către inginerii de date.

3.2.1. MLflow

Fig. 3.3. MLflow logo [15].

3.2.1.1. Definiție și caracteristici

MLflow este o platformă de tip “open source” care gestionează întregul ciclu de viață al învățării automate. Dispune de componente care au capacitatea de a monitoriza modele, de a le stoca, încărca în producție și de a construi un pipeline [16].

Platforma poate fi folosită cu diverse biblioteci de ML, poate fi integrată în numeroase ecosisteme de programare și este organizată în patru componente principale, care permit experimentarea, implementarea, stocarea și reproductibilitatea modelelor:

- Urmărire: această componentă de urmărire este un API și o interfață de utilizator, având rolul de a ține evidența unor valori, parametrii de înregistrare, versiuni de cod și fișiere de ieșire atunci când este rulat codul de ML; este o componentă centrată pe conceptul de rulări care ne permite să interogăm experimente și să le înregistrăm [18];
- Proiecte: bazate în principal pe convenții, proiectele MLflow sunt niște formate care ambalează codul în așa manieră încât să fie refolosibil și reproductibil; aceste proiecte nu sunt nimic altceva decât niște directoare de fișiere sau repository Git care conțin codul nostru [19];

- Modele: modelele de ML pot fi împachetate sub forma unor formate standard, numite modele MLflow, care permit utilizarea acestora într-o gamă largă de instrumente; prin intermediul acestor formate putem salva modelele în diverse "arome", care vor fi recunoscute de programele folosite [20];
- Registrul modelelor: componenta este văzută ca un depozit centralizat de modele, un set de API-uri și o interfață de utilizator, unde sunt gestionate ciclurile de viață ale modelelor MLflow [21].

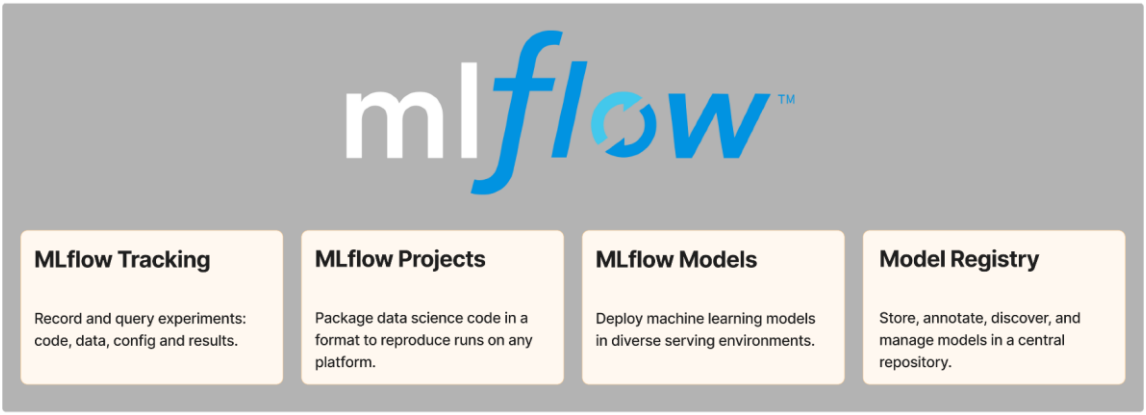

Fig. 3.4. Componentele oferite de MLflow [22].

3.2.1.2. Avantaje și dezavantaje

Avantaje:

- Este încorporat cu multe framework-uri de ML, precum Apache Spark, TensorFlow sau SciKit-Learn [23];
- Configurare ușoară a componentei de urmărire a modelului;
- Sunt oferite API-uri intuitive în ceea ce privește servirea modelelor;
- Rularea experimentelor este ușoară și foarte organizată, deoarece înregistrarea datelor este simplificată și practică;
- Poate funcționa cu orice cloud;
- Abordarea este de tipul "Code-first", adică dezvoltatorii pot defini obiectele model folosindu-se doar de clasele standard [17].

Dezavantaje:

- Modelele nu încorporează automat noi funcții, adică adăugarea de noi "arome" la fișierele generate ale modelului ML nu este automată;
- Implementarea modelelor pe diferite platforme nu este cea mai simplă sau ideală [17].

3.2.2. Amazon SageMaker

Fig. 3.5. Amazon SageMaker logo [24].

3.2.2.1. Definiție și caracteristici

Cei care lucrează în acest domeniu folosesc adesea mai multe unelte de-a lungul proiectelor: biblioteci, framework-uri pentru “deep learning”, iar unii își construiesc propriile instrumente pentru automatizare sau orchestrare. Pentru a putea gestiona toate acestea este nevoie de mult timp și suntem predispuși la erori.

Amazon SageMaker vine ca o soluție la ceea ce am spus mai devreme. A fost creat pentru a construi, antrena, îmbunătăți și implementa modele de ML într-un mod rapid și eficient, utilizând o infrastructură gestionată în întregime. Astfel, inginerii pot să se concentreze doar pe gestionarea modelelor, nu și a infrastructurii instrumentelor cu care lucrează [25].

Poate fi folosit la orice scară și fiind un produs Amazon putem beneficia și de alte servicii AWS pentru modele, precum Amazon S3 (serviciu de stocare) și Amazon Lambda (serviciu de monitorizare a performanțelor modelului) [26].

Principalele caracteristici ale acestui tool:

- Plătești cât consumi: Amazon SageMaker taxează utilizatorii doar pentru resursele pe care le folosesc, nu pentru toate care sunt puse la dispoziție, deci costurile pot fi reduse cu mult;
- Este ușor de utilizat: interfața de utilizator este prietenoasă și intuitivă, iar API-urile sunt destul de ușor de utilizat. Poate fi folosit și de oameni cu mai puțină experiență;
- Gestionat în întregime: deoarece SageMaker este o platformă complet gestionată, inginerii se pot concentra pe crearea și implementarea modelelor de ML, mai degrabă decât să se preocupe de problemele de infrastructură și administrare;
- Flexibil: există numeroși algoritmi și multe framework-uri de ML care sunt acceptate de SageMaker (PyTorch, TensorFlow, MXNet ș.a.m.d.);
- Scalabil: suportă seturi de date masive și modele mai complexe [26].

3.2.2.2. Avantaje și dezavantaje

Avantaje:

- Modelele de ML pot fi create, antrenate și implementate rapid, iar asta aduce un plus atunci când vine vorba de piață. Organizațiile pot astfel să își prezinte mai rapid noile produse și servicii;
- Costurile nu sunt fixe și pot fi reduse. Cu cât folosim mai puține resurse, cu atât plătim mai puțin, însă acesta reprezintă un avantaj doar la proiectele mici;
- Performanțele modelului sunt optimizate automat prin reglarea hiperparametriilor. Asta înseamnă o reducere de timp și efort care ar fi fost necesare pentru reglarea modelelor;
- Amazon SageMaker are încorporați algoritmi, modele deja antrenate și mai multe șabloane special create pentru a veni în ajutorul inginerilor de date sau a celor care sunt într-un proces de învățare are practicilor de ML să implementeze un model într-un timp scurt [26] [27];
- Beneficiază de Jupyter Notebook, care este o aplicație web de tip “open source” prin intermediul căreia putem atât crea, cât și partaja documente cu conținut divers: cod, imagini, videoclipuri, ecuații, text ș.a.m.d. Astfel, totul este ținut în același loc, este ușor de partajat (au format JSON), de convertit, de personalizat și nu este dependent de un limbaj [28].

Dezavantaje:

- Pentru proiectele de anvergură este foarte costisitor;
- Fiind creat de AWS, SageMaker este integrat cu diverse servicii AWS și va fi greu de trecut la un furnizor diferit de cloud;
- Pentru a-l putea folosi într-un mod eficient este nevoie de o persoană cu multă experiență și cunoștințe [26].
- Rulează greu pe seturi mari de date. Nu este la fel de rapid ca alte instrumente [23].

3.2.3. Seldon

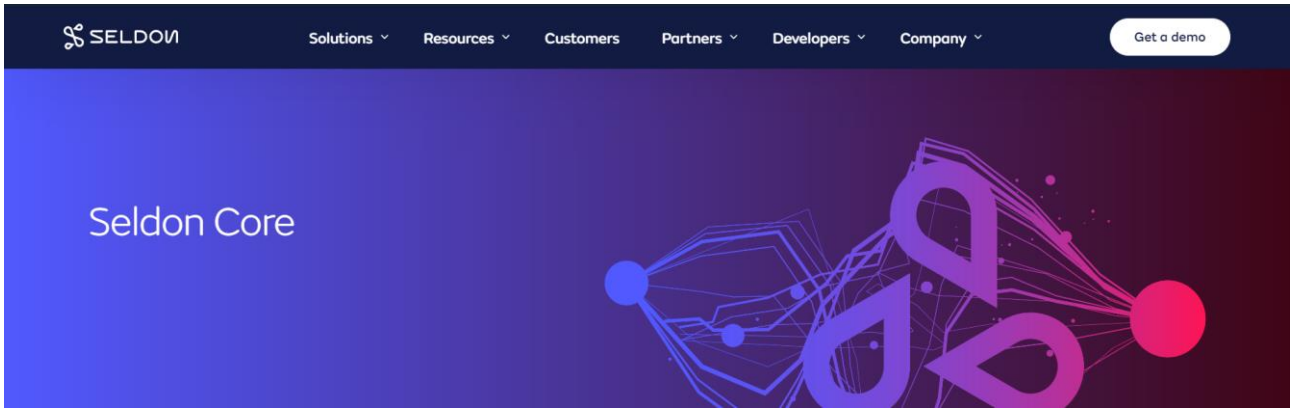

Fig. 3.6. Pagina oficială Seldon [29].

3.2.3.1. Definiție și caracteristici

Seldon este tot o platformă “open-source” cu care putem implementa modele de ML, însă de data aceasta pe Kubernetes, care este un sistem de implementare, scalare și gestionare a aplicațiilor ce sunt containerizate. Pentru a se asigura că totul merge conform intențiilor utilizatorului, Kubernetes orchestrează sarcinile de lucru și planifică containerele într-un cluster de calcul [30].

Aici dispunem de platforma Seldon Core cu ajutorul căruia putem rula și gestiona ușor modelele în producție la o scară mai mare. Aceasta are următoarele obiective:

- Permiterea implementării tuturor modelelor, indiferent de instrumentele de învățare automată sau limbajul de programare folosit la crearea lor;
- Gestionarea completă a ciclului de viață al modelului implementat, printre care și actualizarea graficului de rulare, scalarea, securitatea și monitorizarea;
- Convertirea modelelor ML în microservicii de producție REST și gRPC pentru integrarea mai ușoară în aplicații sau servicii cu nevoi de predicții [31].

Seldon Core are mai multe componente principale, însă eu am să menționez doar 3 din ele:

- Seldon Deployment CRD: este punctul forte al platformei Seldon Core. Cu ajutorul acestuia putem gestiona o parte din traficul real de producție și să implementăm rapid un model de inferență în clusterul Kubernetes. CRD (Custom Resource Definition) sunt niște extensii de la API-ul Kubernetes folosite la definirea graficului de inferență prin intermediul unor fișiere yaml, care odată aplicate clusterului determină crearea obiectelor Kubernetes de către Seldon Core Operator [32].
- Seldon Core Operator: implementările Seldon din interiorul clusterului Kubernetes sunt gestionate de această componentă. Este responsabil cu citirea definiției CRD și se asigură că au fost create componentele necesare. Funcționează într-o buclă infinită: mai întâi este observată starea actuală a clusterului, apoi se face diferența

față de starea cerută și la final, dacă trebuie, se va acționa prin aplicarea stării dorite [32].

Fig. 3.7. Modul de funcționare al lui Seldon Core Operator [32].

- Service Orchestrator: se ocupă cu controlul traficului intra-grafic. Când se face o cerere de inferență, are grijă ca structura graficului din CRD să fie transmisă tuturor nodurilor graficului într-o ordine corectă [32].

3.2.3.2. Avantaje și dezavantaje

Avantaje:

- Procesul de implementare este simplificat;
- Dispune de o infrastructură care este eficientă din punct de vedere al energiei și al costurilor: este posibilă implementarea și găzduirea mai multor modele pe un singur server;
- Poate rula pe orice cloud, nu depinde de framework-uri și acceptă biblioteci de învățare automată, limbi și instrumente de top;
- Utilizează Kubernetes: are avantajul de a crește productivitatea prin automatizarea proceselor manuale [29].

Dezavantaje:

- Este mai dificil de învățat, iar configurarea este puțin mai complexă [17].

3.2.4. Kubeflow

Fig. 3.8. Kubeflow logo [33].

3.2.4.1. Definiție și caracteristici

Platforma “open-source” Kubeflow se folosește, la fel ca și Seldon, de Kubernetes și a fost creată pentru ca fluxurile de lucru de învățare automată să fie mai ușor de implementat și gestionat.
Kubeflow face ca aceste implementări să fie simple, scalabile și portabile (de exemplu, începem experimentarea de pe un calculator sau laptop și apoi trecem în cloud sau un cluster local).

Componentele principale ale lui Kubeflow:

- Kubeflow Notebooks: pot fi rulate medii de dezvoltare care sunt bazate pe web în interiorul Pod-urilor din clusterul Kubernetes folosind Kubeflow Notebooks. Acceptă mai multe tipuri de notebooks, însă cele mai folosite sunt JupyterLab, RStudio și Visual Studio Code;
- Kubeflow Pipelines: datorită acestei platforme pot fi create și implementate fluxuri de lucru de învățare automată într-un mod scalabil și portabil. Kubeflow Pipelines constă într-o interfață de utilizator unde sunt gestionate și urmărite experimente, rulări și joburi, un motor care programează fluxurile cu mai mulți pași și un SDK ce definește și orchestrează pipeline-urile și componentele;
- Katib: este componenta care folosește tehnici complexe de optimizare pentru a regla hiperparametrul și a selecta automat modelul de învățare automată. Suportă numeroase framework-uri de învățare automată, precum PyTorch, TensorFlow, XGBoost ș.a.m.d.;
- Operatori de instruire: operatori oferiți de Kubeflow pentru a antrena modelele de ML. Ei pot aduce o îmbunătățire în ceea ce privește performanța și scalabilitatea modelelor [34].

3.2.4.2. Avantaje și dezavantaje

Avantaje:

- Cu Kubeflow Pipelines putem gestiona tot fluxul de muncă, dispunând de o interfață de utilizator simplă;
- Instrumentul dispune de servere de notebooks care sunt integrate tocmai pentru a putea face o experimentare rapidă și pentru a accesa cu ușurință resursele clusterului;
- Fiind un framework care se bazează pe Kubernetes pentru execuția codului și administrarea rețelelor și resurselor, Kubeflow poate fi numit un framework flexibil și extensibil în orice mediu cu Kubernetes;
- Pașii unui pipeline sunt izolați unul câte unul în containere. Unul dintre avantaje este că ajută dezvoltatorii atunci când, de exemplu, dorim să aflăm informații despre starea nodurilor. Printr-un simplu clic la una din etapele pipeline-ului putem accesa informații din Kubernetes precum: “0/3 noduri disponibile: 3 memorie insuficientă”;
- Kubeflow ne oferă posibilitatea de a personaliza totul [35].
- Acceptă framework-uri de învățare automată populare, precum PyTorch, scikit-learn, TensorFlow, XGBoost s.a.m.d.;
- Experimentele se pot face local și apoi implementate într-un cloud;
- Mediul oferit de Kubeflow este unul colaborativ, ajutând echipele să lucreze împreună la găsirea de soluții [36].

Dezavantaje:

- Este mai greu de instalat și configurat (ceea ce nu îl recomandă pentru aplicațiile care sunt la scară mică), mai ales pentru cei cărora subiectele Kubernetes și containerizare sunt noi;
- Are un grad de dificultate mai ridicat decât multe alte instrumente MLOps;
- Are o integrare limitată atunci când vine vorba de mediile ce nu folosesc Kubernetes. Desigur, are un nivel de flexibilitate și în acest caz, însă s-ar putea ca integrarea într-un mediu “non-Kubernetes” să necesite un efort suplimentar mai mare [17].

3.2.5. BentoML

Fig. 3.9. BentoML logo [37].

3.2.5.1. Definiție și caracteristici

BentoML este un framework foarte cunoscut de Python cu care putem împacheta cu ușurință modelele ML antrenate în servicii implementabile. Interfața de care dispune este una simplă orientată pe obiecte, creează servicii HTTP pentru fiecare model ambalat și prezintă un sistem centralizat unde sunt organizate modele și monitorizări ale proceselor de implementare [38].

Cei care folosesc BentoML urmăresc să:

- Grăbească și să standardizeze întregul proces de mutare a modelelor de învățare automată în producție;
- Realizeze servicii de predicție eficiente și scalabile;
- Existe o continuă implementare, monitorizare și operare în producție a serviciilor de predicție [39].

Caracteristicile principale:

- BentoML suportă foarte multe framework-uri populare din industria ML: TensorFlow, XGBoost, MLflow, Pytorch, Scikit-Learn ș.a.m.d.;
- Poate fi aplicat același cod pentru orice mod de implementare: online (ca punct final REST API sau ca un serviciu gRPC), offline (batch) sau inferență “streaming”;
- Pentru o tranziție ușoară în producție, procesul de standardizare constă în crearea de bento-uri. Acestea sunt arhive care conțin următoarele: fișiere cu codul sursă, modele, fișiere cu configurații și date;
- Este scalabil prin opțiunile disponibile de optimizare a performanței, precum maximizarea utilizării procesoarelor [37] [40].

3.2.5.2. Avantaje și dezavantaje

Avantaje:

- Un avantaj important este că implementarea se poate face pe mai multe platforme. În modul online avem trei soluții de bază: BentoML Cloud (este un mod simplu, rapid și la scară de implementare bento), Yatai (implementare pe Kubernetes a modelelor, la scară), Bentoctl (implementarea directă a modelelor pe servicii de cloud publice, precum Lambda, Azure, Heroku și altele). De asemenea, putem folosi Spark sau Dask pentru a rula joburi de inferență în batch-uri offline [37];
- Acceptă framework-uri populare de ML: TensorFlow, Pytorch ș.a.m.d.;
- Munca cercetătorilor din acest domeniu nu ar trebui să se rezume doar la pachetul de framework-uri folosit. Soluția de servire a modelelor trebuie să accepte atât un framework, cât și un cod personalizat, iar la acest capitol BentoML stă bine. Pentru a folosi BentoML trebuie implementat cod Python și acest lucru permite orice personalizare [41];
- Este un framework de servire a modelelor ușor de utilizat și configurat;
- Dispune de o interfață de utilizator unde sunt organizate modelele și sunt urmărite procedurile de implementare [5].

Dezavantaje:

- Nu dispune de funcționalități de scalare automată.

În cele ce urmează vă voi arăta cât este de simplu să parcurgem procesele necesare implementării unui model de învățare automată în producție folosind MLflow.

Am ales acest instrument pentru că poate fi folosit fără multe cerințe, are o complexitate redusă față de celelalte instrumente, este gratuit și dispune de numeroase framework-uri foarte utilizate în acest domeniu. De asemenea, beneficiez de o interfață de utilizator intuitivă unde pot vizualiza modelele, valorile calculate și multe altele.

4. IMPLEMENTARE

4.1. ANALIZA DATELOR

Pentru partea practică a acestei lucrări de licență am ales să folosesc un set de date din telecomunicatii (Telco) pentru a putea arăta cum sunt analizate și prelucrate datele, cum se creează, evaluează, validează un model și cum poate fi implementat local un astfel de model.

Setul de date folosit este “WA_Fn-UseC_-Telco-Customer-Churn.csv” [42], cu puține modificări aduse de mine (atribute schimbate).

Este un fișier .csv care conține atât date despre clienții actuali (la acea dată), cât și despre cei care au ales să renunțe la serviciile prestate de cei de la Telco. Pe acest aspect se va concentra modelul nostru de învățare automată: predicția clienților care sunt pe cale să se dezaboneze.

Voi parcurge pașii următori:

1. Instrumente și pachete

Pentru a analiza datele nu am nevoie decât de Jupyter notebooks care va rula pe un server local și pachetele următoare:

Fig. 4.1. Interfața de utilizator al lui Jupyter notebooks.

```
In [1]: import pandas as pd
        import missingno as msno
        import numpy as np
        import plotly.graph_objects as plgo
        import plotly.express as plex
        import seaborn as sn
        import matplotlib.pyplot as mplt
        from plotly.subplots import make_subplots
        from sklearn.preprocessing import LabelEncoder
        from sklearn.preprocessing import StandardScaler
```

Fig. 4.2. Pachetele necesare pentru analiza datelor.

2. Încărcarea datelor

Pentru a încărca datele din fișierul .csv dorit apelez funcția read_csv(), iar apoi verific
că datele au fost încărcate corect prin vizualizarea lor cu metoda head():

```
In [2]: data_file = 'WA_Fn-UseC_-Telco-Customer-Churn.csv'
        df = pd.read_csv(data_file)

In [3]: df.head()
```

Out[3]:

	IdClient	Gen	InVarsta	Partener	Dependent	DurataClient	ServiciuTelefon	LiniiMultiple	ServiciuInternet	SecuritateOnline	...	ProtectieDispozitiv	Suport
0	7590-VHVEG	Feminin	0	Da	Nu	1	Nu	Fara serviciu telefonic	DSL	Nu	...	Nu	
1	5575-GNVDE	Masculin	0	Nu	Nu	34	Da	Nu	DSL	Da	...	Da	
2	3668-QPYBK	Masculin	0	Nu	Nu	2	Da	Nu	DSL	Da	...	Nu	
3	7795-CFOCW	Masculin	0	Nu	Nu	45	Nu	Fara serviciu telefonic	DSL	Da	...	Da	
4	9237-HQITU	Feminin	0	Nu	Nu	2	Da	Nu	Fibra optica	Nu	...	Nu	

5 rows × 21 columns

Fig. 4.3. Încărcarea și vizualizarea setului de date.

3. Examinarea datelor

Rândurile din acest tabel reprezintă clienții, iar coloanele conțin atributele lor. Cu shape pot vedea că sunt 7039 de clienți și 21 de atribute:

```
In [4]: df.shape
Out[4]: (7039, 21)
```

Fig. 4.4. Numărul de clienți și de atribute din setul de date.

Cu dtypes verific care sunt aceste atribute și ce tip de date folosesc:

```
In [5]: df.dtypes
Out[5]: IdClient              object
        Gen                   object
        InVarsta               int64
        Partener              object
        Dependent             object
        DurataClient           int64
        ServiciuTelefon       object
        LiniiMultiple         object
        ServiciuInternet      object
        SecuritateOnline      object
        PlanRezervaOnline     object
        ProtectieDispozitiv   object
        SuportTehnic          object
        StreamingTV           object
        StreamingFilme        object
        Contract              object
        FacturareFaraHartie   object
        MetodaPlata           object
        TaxeLunar            float64
        TaxeTotal             object
        Abandon               object
        dtype: object
```

Fig. 4.5. Vizualizarea atributelor clienților și tipul de date al acestora.

Astfel, obțin informații precum:

- Clienții care s-au dezabonat: atributul “Abandon”;
- Serviciile de care dispun clienții: telefon, internet, securitate online ș.a.m.d.;

- Abonamentul clienților: tipul de contract, taxele lunare și totale, metoda de plată ș.a.m.d.;
- Date generale ale clienților: interval de vârstă, gen ș.a.m.d.;

4. Verificarea datelor lipsă
   Cu matrix() pot vizualiza și verifica rapid dacă lipsesc date:

```
In [6]: msno.matrix(df);
```

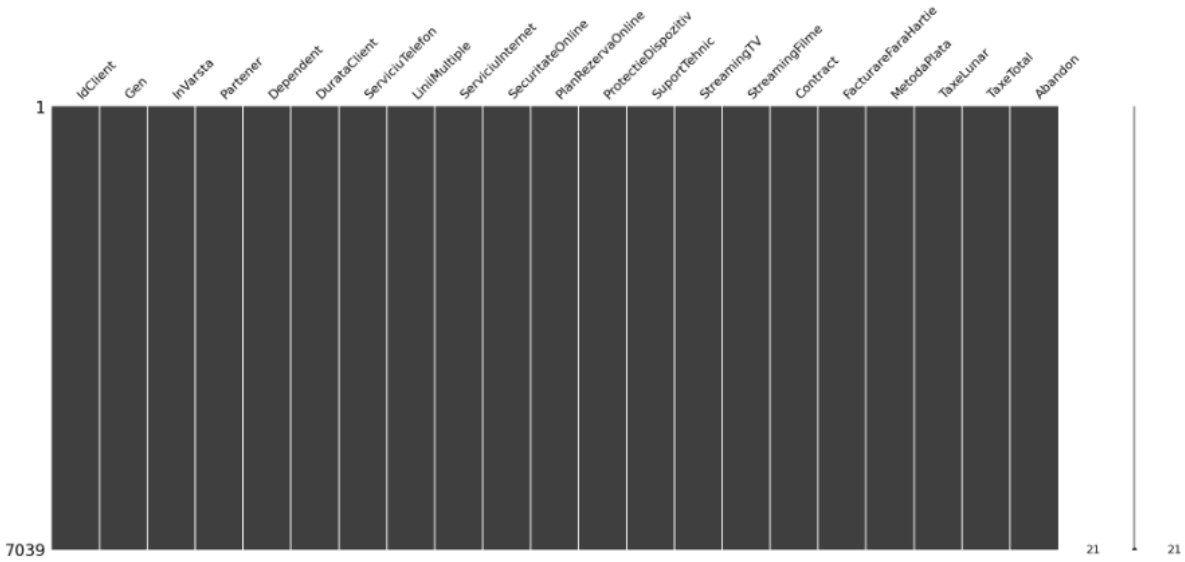

Fig. 4.6. Verificarea datelor lipsă.

Din imaginea de mai sus, se poate observa că nu sunt date lipsă.

5. Prepararea datelor pentru vizualizare
   Voi începe prin excluderea coloanei “IdClient” cu ajutorul funcției drop(), pentru că nu ne ajută cu nimic mai departe:

```
In [7]: df = df.drop(['IdClient'], axis = 1)
        df.shape
Out[7]: (7039, 20)
```

Fig. 4.7. Excluderea coloanei intitulată “IdClient”.

Pentru a mă asigura că nu am valori invalide (spații goale) voi analiza puțin datele. Voi căuta valori de 0 întrucât prezența lor semnifică faptul că avem valori care nu sunt valide:

```
In [8]: df['TaxeTotal'] = pd.to_numeric(df.TaxeTotal, errors='coerce')
        df.isnull().sum()

Out[8]: Gen                 0
        InVarsta            0
        Partener            0
        Dependent           0
        DurataClient        0
        ServiciuTelefon     0
        LiniiMultiple       0
        ServiciuInternet    0
        SecuritateOnline    0
        PlanRezervaOnline   0
        ProtectieDispozitiv 0
        SuportTehnic        0
        StreamingTV         0
        StreamingFilme      0
        Contract            0
        FacturareFaraHartie 0
        MetodaPlata         0
        TaxeLunar           0
        TaxeTotal          11
        Abandon             0
        dtype: int64
```

Fig. 4.8. Identificarea valorilor invalide.

Sunt 11 valori invalide în coloana “TaxeTotal”. Doresc să văd care sunt acestea:

```
In [9]: df[np.isnan(df['TaxeTotal'])]
```

Out[9]:

PlanRezervaOnline	ProtectieDispozitiv	SuportTehnic	StreamingTV	StreamingFilme	Contract	FacturareFaraHartie	MetodaPlata	TaxeLunar	TaxeTotal	Abandon
Nu	Da	Da	Da	Nu	Doi ani	Da	Transfer bancar (automat)	52.55	NaN	Nu
Fara serviciu de internet	Fara serviciu de internet	Fara serviciu de internet	Fara serviciu de internet	Fara serviciu de internet	Doi ani	Nu	Cec trimis prin posta	20.25	NaN	Nu
Da	Da	Nu	Da	Da	Doi ani	Nu	Cec trimis prin posta	80.85	NaN	Nu
Fara serviciu de internet	Fara serviciu de internet	Fara serviciu de internet	Fara serviciu de internet	Fara serviciu de internet	Doi ani	Nu	Cec trimis prin posta	25.75	NaN	Nu
Da	Da	Da	Da	Nu	Doi ani	Nu	Card de credit (automat)	56.05	NaN	Nu
Fara serviciu de internet	Fara serviciu de internet	Fara serviciu de internet	Fara serviciu de internet	Fara serviciu de internet	Doi ani	Nu	Cec trimis prin posta	19.85	NaN	Nu
Fara serviciu de internet	Fara serviciu de internet	Fara serviciu de internet	Fara serviciu de internet	Fara serviciu de internet	Doi ani	Nu	Cec trimis prin posta	25.35	NaN	Nu
Fara serviciu de internet	Fara serviciu de internet	Fara serviciu de internet	Fara serviciu de internet	Fara serviciu de internet	Doi ani	Nu	Cec trimis prin posta	20.00	NaN	Nu
Fara serviciu de internet	Fara serviciu de internet	Fara serviciu de internet	Fara serviciu de internet	Fara serviciu de internet	Un an	Da	Cec trimis prin posta	19.70	NaN	Nu
Da	Da	Da	Da	Nu	Doi ani	Nu	Cec trimis prin posta	73.35	NaN	Nu
Da	Nu	Da	Nu	Nu	Doi ani	Da	Transfer bancar (automat)	61.90	NaN	Nu

Fig. 4.9. Vizualizarea valorilor invalide.

În timpul analizei am observat că doar acestor rânduri le corespunde valoarea 0 în coloana “DurataClient”. Asta înseamnă că clienții nu au fost abonați decât pentru o perioadă foarte scurtă de timp și pot elimina acești clienți, deoarece nu vor afecta celelalte date:

```
In [10]: df[df['DurataClient'] == 0].index

Out[10]: Index([488, 753, 936, 1082, 1340, 3331, 3826, 4380, 5218, 6670, 6754], dtype='int64')

In [11]: df.drop(labels=df[df['DurataClient'] == 0].index, axis=0, inplace=True)
         df[df['DurataClient'] == 0].index

Out[11]: Index([], dtype='int64')
```

Fig. 4.10. Eliminarea clienților cu durată scurtă de abonare.

Acum toate datele sunt valide:

```
In [13]: df.isnull().sum()

Out[13]: Gen                0
         InVarsta           0
         Partener           0
         Dependent          0
         DurataClient       0
         ServiciuTelefon    0
         LiniiMultiple      0
         ServiciuInternet   0
         SecuritateOnline   0
         PlanRezervaOnline  0
         ProtectieDispozitiv 0
         SuportTehnic       0
         StreamingTV        0
         StreamingFilme     0
         Contract           0
         FacturareFaraHartie 0
         MetodaPlata        0
         TaxeLunar          0
         TaxeTotal          0
         Abandon            0
         dtype: int64
```

Fig. 4.11. Verificarea că nu mai sunt date invalide.

Coloana “InVarsta” are valori doar de “0” și “1”. Pentru a păstra o coerență față de celelalte valori, voi recodifica cele 2 valori în formatul “Nu” și “Da”:

```
In [14]: df["InVarsta"]= df["InVarsta"].map({0: "Nu", 1: "Da"})
         df.head()
```

Out[14]:

	Gen	InVarsta	Partener	Dependent	DurataClient	ServiciuTelefon	LiniiMultiple	ServiciuInternet	SecuritateOnline	PlanRezervaOnline	ProtectieDispozitiv
0	Feminin	Nu	Da	Nu	1	Nu	Fara serviciu telefonic	DSL	Nu	Da	Nu
1	Masculin	Nu	Nu	Nu	34	Da	Nu	DSL	Da	Nu	Da
2	Masculin	Nu	Nu	Nu	2	Da	Nu	DSL	Da	Da	Nu
3	Masculin	Nu	Nu	Nu	45	Nu	Fara serviciu telefonic	DSL	Da	Nu	Da
4	Feminin	Nu	Nu	Nu	2	Da	Nu	Fibra optica	Nu	Nu	Nu

Fig. 4.12. Recodificarea valorilor atributului “InVarsta” în formatul “Nu” și “Da”.

6. Vizualizarea datelor

Această etapă este importantă pentru a îmi putea da seama cum aș putea să îmi îmbunătățesc modelul în luarea deciziilor. O primă vizualizare a datelor va fi distribuția genului și abandonului:

```
In [16]: g_labels = ['Masculin', 'Feminin']
         a_labels = ['Nu', 'Da']
         # Folosesc tipul 'domain' pentru o vizualizare grafica de tip Pie
         fig = make_subplots(rows=1, cols=2, specs=[[{'type':'domain'}, {'type':'domain'}]])
         fig.add_trace(plgo.Pie(labels=g_labels, values=df['Gen'].value_counts(), name="Gen"),
                       1, 1)
         fig.add_trace(plgo.Pie(labels=a_labels, values=df['Abandon'].value_counts(), name="Abandon"),
                       1, 2)

         # Parametrul hole este pentru a crea o diagrama asemenea unei gogosi
         fig.update_traces(hole=.5, hoverinfo="label+percent+name", textfont_size=16)

         fig.update_layout(
             title_text="Distribuția genului și abandonului",
             annotations=[dict(text='Gen', x=0.19, y=0.5, font_size=21, showarrow=False),
                          dict(text='Abandon', x=0.84, y=0.5, font_size=21, showarrow=False)])
         fig.show()
```

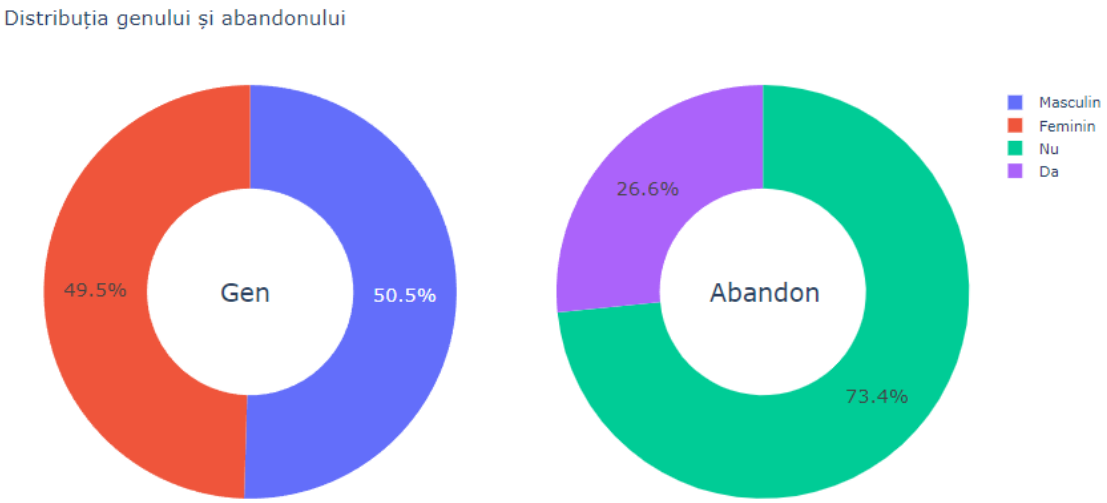

Fig. 4.13. Distribuția genului și abandonului.

De aici tragem concluzia că 26.6% dintre clienți renunță la abonament, iar procentul de bărbați este aproape egal cu cel de femei.

De asemenea, numărul persoanelor de gen feminin care abandonează este aproximativ egal cu cel al bărbaților care abandonează:

```
In [17]: df["Abandon"][df["Abandon"]=="Nu"].groupby(by=df["Gen"]).count()
Out[17]: Gen
         Feminin     2542
         Masculin    2618
         Name: Abandon, dtype: int64

In [18]: df["Abandon"][df["Abandon"]=="Da"].groupby(by=df["Gen"]).count()
Out[18]: Gen
         Feminin     939
         Masculin    929
         Name: Abandon, dtype: int64
```

Fig. 4.14. Numărul clienților care abandonează în funcție de gen.

Dintre clienții care beneficiază de un contract lunar, 75% au renunțat la aceste servicii, pe când cei cu contract de un an, respectiv doi ani au renunțat doar 13%, respectiv 3%:

```
In [19]: fig = plex.histogram(df, x="Abandon", y=None, color="Contract", barmode="group", title="<b>Distribuția contractelor clienților<b
         fig.update_yaxes(title_text='Numar', row=1, col=1)
         fig.show()
```

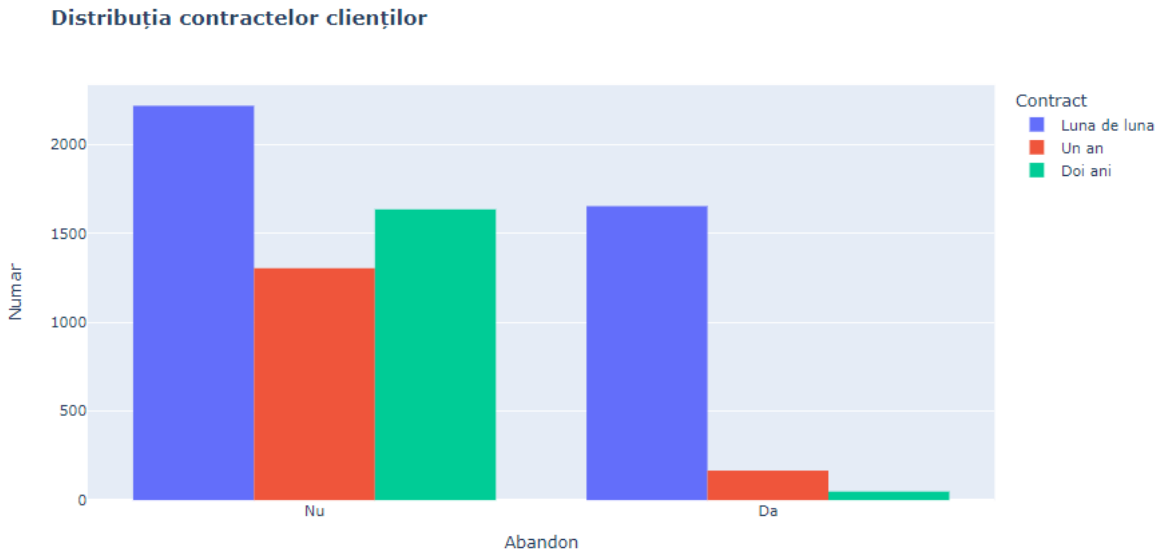

Fig. 4.15. Distribuția contractelor clienților în funcție de abandon.

O altă observație este că majoritatea celor care au ales cecul electronic ca metodă de plată s-au mutat la altă companie:

```
In [20]: fig = plex.histogram(df, x="Abandon", color="MetodaPlata", title="<b>Distribuția metodei de plată în funcție de abandon</b>")
         fig.update_yaxes(title_text='Numar', row=1, col=1)
         fig.show()
```

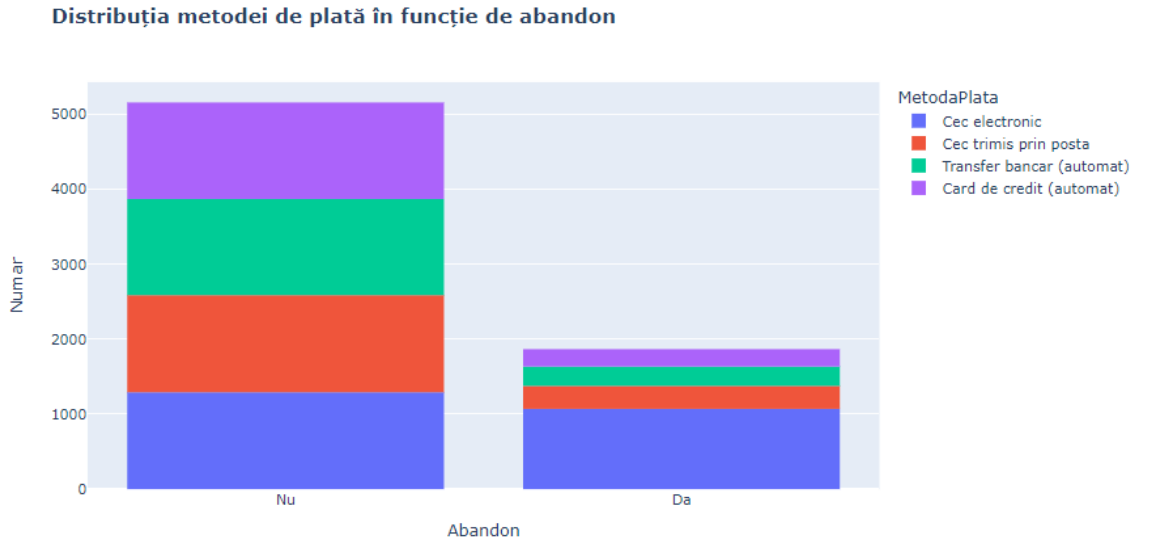

Fig. 4.16. Distribuția metodei de plată în funcție de abandon.

Cu toate că serviciul DSL este cel mai folosit, se poate vedea că cei mai mulți renunță la abonament dacă folosesc fibră optică:

```
In [23]: fig = plgo.Figure()

fig.add_trace(plgo.Bar(
  x = [['Abandon:Nu', 'Abandon:Nu', 'Abandon:Da', 'Abandon:Da'],
       ["Feminin", "Masculin", "Feminin", "Masculin"]],
  y = [964, 992, 219, 240],
  name = 'DSL',
))

fig.add_trace(plgo.Bar(
  x = [['Abandon:Nu', 'Abandon:Nu', 'Abandon:Da', 'Abandon:Da'],
       ["Feminin", "Masculin", "Feminin", "Masculin"]],
  y = [888, 909, 664, 632],
  name = 'Fibra optica',
))

fig.add_trace(plgo.Bar(
  x = [['Abandon:Nu', 'Abandon:Nu', 'Abandon:Da', 'Abandon:Da'],
       ["Feminin", "Masculin", "Feminin", "Masculin"]],
  y = [690, 717, 56, 57],
  name = 'Fara internet',
))

fig.update_layout(title_text="<b>Distribuția de abandon în funcție de serviciul de internet și gen</b>")
fig.show()
```

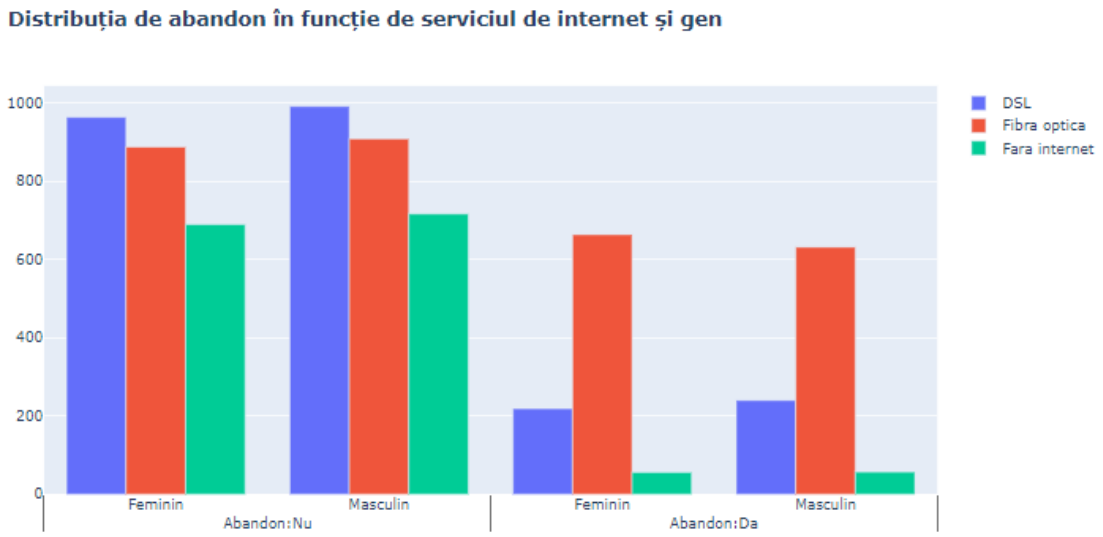

Fig. 4.17. Distribuția de abandon în funcție de serviciul de internet și gen.

Un alt grafic ne arată că majoritatea celor care sunt în vârstă aleg să renunțe la abonament:

```
In [24]: color_map = {"Da": '#ee73b0', "Nu": '#a94778'}
fig = plex.histogram(df, x="Abandon", color="InVarsta", title="<b>Distribuția abandonului în funcție de vârstă</b>", color_discr
fig.update_yaxes(title_text='Numar', row=1, col=1)
fig.show()
```

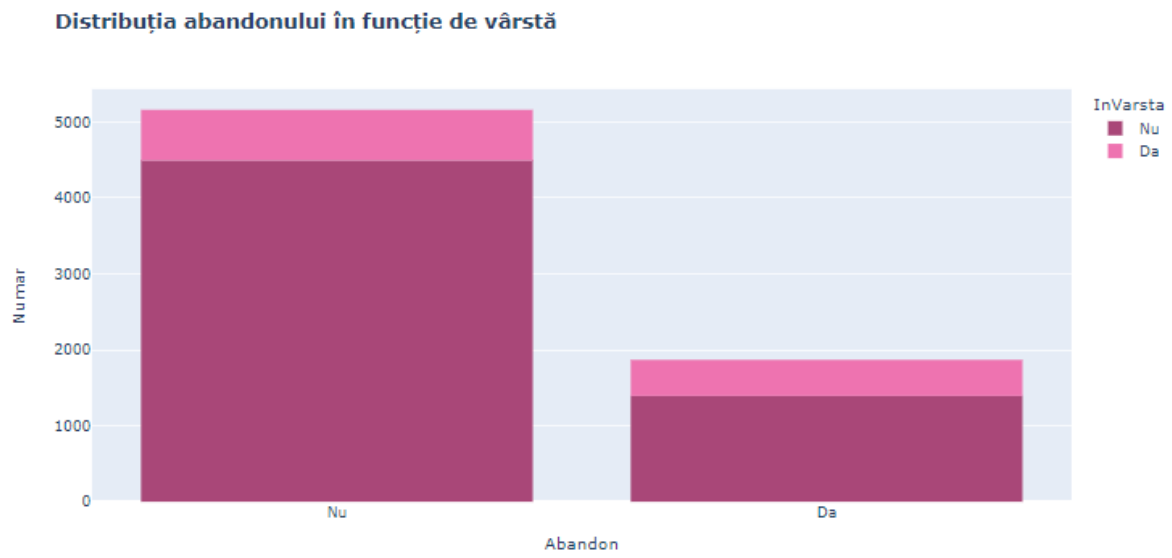

Fig. 4.18. Distribuția abandonului în funcție de vârstă.

Așa cum de altfel cei care au taxe lunare mai mari renunță mult mai des:

```
In [25]: sn.set_context("paper",font_scale=1.0)
ax = sn.kdeplot(df.TaxeLunar[(df["Abandon"] == 'Nu') ],
                color="Red", fill = True);
ax = sn.kdeplot(df.TaxeLunar[(df["Abandon"] == 'Da') ],
                ax =ax, color="Blue", fill= True);
ax.legend(["Nu abandoneaza","Abandoneaza"],loc='upper right');
ax.set_ylabel('Densitate');
ax.set_xlabel('Taxe lunare');
ax.set_title('Distribuția taxelor lunare în funcție de abandon');
```

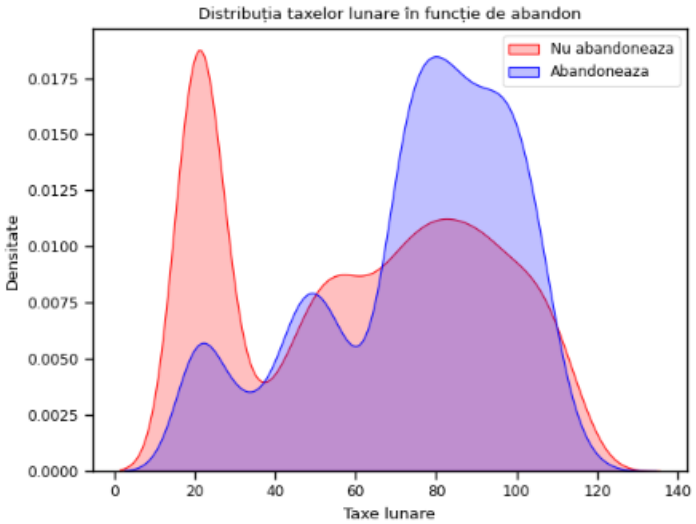

Fig. 4.19. Distribuția taxelor lunare în funcție de abandon.

Clienții noi au șanse foarte mari de abandon:

In [26]:
```
fig = plex.box(df, x='Abandon', y = 'DurataClient')

fig.update_yaxes(title_text='Durata client (luni)', row=1, col=1)
fig.update_xaxes(title_text='Abandon', row=1, col=1)

fig.update_layout(autosize=True, width=800, height=550,
    title_font=dict(size=25, family='Courier'),
    title='<b>Durată client vs abandon </b>',
)

fig.show()
```

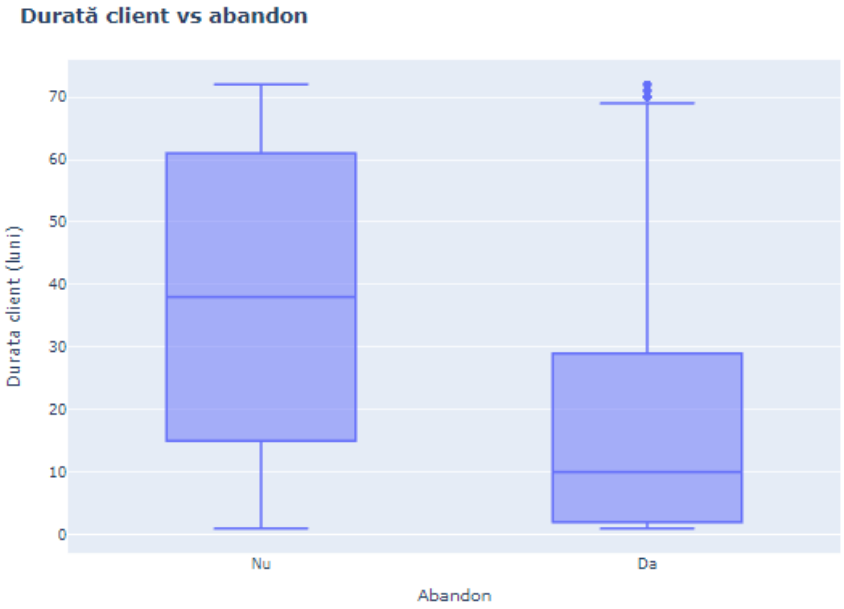

Fig. 4.20. Durată client vs abandon.

4.2. PREPROCESAREA DATELOR

Pachete necesare:

```
In [1]: from sklearn.preprocessing import LabelEncoder
        from sklearn.model_selection import train_test_split
        from sklearn.preprocessing import StandardScaler
```

Fig. 4.21. Pachetele necesare pentru preprocesarea datelor.

Pentru a putea folosi datele în antrenarea și evaluarea modelului, trebuie ca datele să fie de tip numeric. Setul nostru de date conține șiruri de caractere cu care nu se pot face calcule și de aceea ele trebuie mapate. Prin această mapare, “Da” va lua valoarea “0”, “Nu” va lua valoarea “1” și așa mai departe.

Cu LabelEncoder().fit_transform() voi converti fiecare coloană în valori numerice:

```
In [11]: def object_to_int(dataframe_series):
             if dataframe_series.dtype=='object':
                 dataframe_series = LabelEncoder().fit_transform(dataframe_series)
             return dataframe_series

In [12]: df = df.apply(lambda x: object_to_int(x))
         df.head()
```

Out[12]:

	Gen	InVarsta	Partener	Dependent	DurataClient	ServiciuTelefon	LiniiMultiple	ServiciuInternet	SecuritateOnline	PlanRezervaOnline	ProtectieDispozitiv	Sup
0	0	1	0	1	1	1	1	0	2	0	2	
1	1	1	1	1	34	0	2	0	0	2	0	
2	1	1	1	1	2	0	2	0	0	0	2	
3	1	1	1	1	45	1	1	0	0	2	0	
4	0	1	1	1	2	0	2	1	2	2	2	

Fig. 4.22. Convertirea datelor în valori numerice.

Împart setul de date în 3 subseturi: 70% antrenare, 20% testare și 10% validare:

```
In [13]: x = df.drop(columns = ['Abandon'])
         y = df['Abandon'].values

In [14]: x_train, x_test, y_train, y_test = train_test_split(x,y,test_size = 0.30, random_state = 40, stratify=y)
         x_test, x_val, y_test, y_val = train_test_split(x_test, y_test, test_size=0.333)

In [15]: print(x_train.shape)
         print(y_train.shape)
         print(x_test.shape)
         print(y_test.shape)
         print(x_val.shape)
         print(y_val.shape)

         (4919, 19)
         (4919,)
         (1406, 19)
         (1406,)
         (703, 19)
         (703,)
```

Fig. 4.23. Împărțirea datelor în cele 3 subseturi de antrenare, testare și validare.

Intervalele de valori pe care sunt distribuite caracteristicile numerice ale atributelor “DurataClient”, “TaxeLunar” și “TaxeTotal” sunt diferite. Este nevoie de o scalare pentru a fi reduse la același interval, iar pentru asta voi folosi StandardScaler():

```
In [37]: col_num = ["DurataClient", 'TaxeLunar', 'TaxeTotal']

         scaler= StandardScaler()

         x_train[col_num] = scaler.fit_transform(x_train[col_num])
         x_test[col_num] = scaler.transform(x_test[col_num])
         x_val[col_num] = scaler.transform(x_val[col_num])
```

Fig. 4.24. Scalarea caracteristicilor numerice.

4.3. ANTRENAREA ȘI EVALUAREA MODELELOR CU MLFLOW

La un model de învățare automată se pot urmări mai multe valori atunci când vine vorba de antrenare și evaluare, însă eu voi analiza două valori:

- acuratețea: raportul dintre numărul de predicții corecte și numărul total de predicții;
- auc: aria de sub curba “ROC”, curbă ce arată relația dintre FPR (false-positive rate) și TPR (true-positive rate); ia valori între 0 și 1, iar pentru un model cât mai bun trebuie ca această valoare să se apropie de 1 [43].

Pachete necesare:

In [1]:
```
import mlflow
import pandas as pd
import matplotlib.pyplot as mplt
import seaborn as sn
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import roc_curve, confusion_matrix, roc_auc_score
```

Fig. 4.25. Pachetele necesare pentru antrenarea și evaluarea modelului.

Îmi voi defini 2 funcții de antrenare și evaluare:

In [38]:
```
def train(model, x_train, y_train):
    model = model.fit(x_train, y_train)

    train_acc = model.score(x_train, y_train)
    mlflow.log_metric("train_acc", train_acc)

    print(f"Train accuracy for {str(model).split('(')[0]}: {train_acc:.3%}")

def evaluate(model, x_test, y_test):
    eval_acc = model.score(x_test, y_test)

    preds = model.predict(x_test)
    auc_score = roc_auc_score(y_test, preds)

    mlflow.log_metric("eval_acc", eval_acc)
    mlflow.log_metric("auc_score", auc_score)

    print(f"-----------{str(model).split('(')[0]}-----------")
    print(f"Auc Score: {auc_score:.3%}")
    print(f"Eval Accuracy: {eval_acc:.3%}")
```

Fig. 4.26. Definirea funcțiilor de antrenare și evaluare.

Scikit-learn ne pune la dispoziție numeroase metode și funcții special create pentru învățarea automată. Cu funcția train() voi antrena modelul și voi determina acuratețea de predicție folosind datele de antrenare, iar funcția evaluate() va folosi datele de test pentru a calcula acuratețea și valoarea auc.

Valorile vor fi înregistrate prin mlflow.log_metric(). Astfel, la fiecare rulare MLFlow va ține evidența acestor valori, iar eu le pot compara, dar voi intra în mai multe detalii mai târziu.

Acum voi crea 6 modele folosind algoritmii următori:

```
In [43]: knn_model = KNeighborsClassifier(n_neighbors = 11)
         svc_model = SVC(gamma='auto', probability=True)
         rf_model = RandomForestClassifier(n_estimators=500 , oob_score = True, n_jobs = -1,
                                           random_state =50, max_features = "auto",
                                           max_leaf_nodes = 30)
         lr_model = LogisticRegression()
         ab_model = AdaBoostClassifier()
         gb_model = GradientBoostingClassifier()

         model_list = [knn_model, svc_model, rf_model, lr_model, ab_model, gb_model]
```

Fig. 4.27. Lista de algoritmi utilizați.

Folosind funcțiile de antrenare și evaluare, obțin următoarele valori pentru cele 6 modele:

```
In [44]: for model in model_list:
             train(model, x_train, y_train)

         Train accuracy for KNeighborsClassifier: 81.297%
         Train accuracy for SVC: 80.382%
         Train accuracy for RandomForestClassifier: 81.236%
         Train accuracy for LogisticRegression: 79.854%
         Train accuracy for AdaBoostClassifier: 80.545%
         Train accuracy for GradientBoostingClassifier: 82.923%
```

Fig. 4.28. Acuratețea obținută în urma antrenării modelelor.

```
In [45]: for model in model_list:
             evaluate(model, x_test, y_test)

         -----------KNeighborsClassifier-----------
         Auc Score: 69.095%
         Eval Accuracy: 77.240%
         -----------SVC-----------
         Auc Score: 68.767%
         Eval Accuracy: 80.156%
         -----------RandomForestClassifier-----------
         Auc Score: 69.232%
         Eval Accuracy: 80.085%
         -----------LogisticRegression-----------
         Auc Score: 72.189%
         Eval Accuracy: 80.654%
         -----------AdaBoostClassifier-----------
         Auc Score: 71.686%
         Eval Accuracy: 80.797%
         -----------GradientBoostingClassifier-----------
         Auc Score: 69.921%
         Eval Accuracy: 79.587%
```

Fig. 4.29. Acuratețea și valoarea auc obținute în urma evaluării modelelor.

Curba unui model perfect redă întotdeauna o rată “true-positive” de 100% și este reprezentată în figura 4.30 de cele 2 drepte intersectate în punctul x din colțul din stânga sus, pe când o estimare aleatorie corespunde unei diagonale, adică are cele 2 rate de “true-positive” și “false-positive” egale cu 50%. În realitate, această curbă se va situa undeva între cele două curbe, însă cu cât este mai aproape de x-ul din stânga cu atât modelul este mai bun.

Reprezentarea curbei ROC:

In [46]:
```
colors = ['r', 'b', 'g', 'm', 'y', 'black']

def roc_curve_plot(model_list, x_test, y_test):
    mplt.plot([0, 1], [0, 1], 'k--' )
    i = 0
    for model in model_list:
        y_pred_prob = model.predict_proba(x_test)[:,1]
        preds = model.predict(x_test)

        fpr, tpr, thresholds = roc_curve(y_test, y_pred_prob)

        mplt.plot(fpr, tpr, label=str(model).split('(')[0], color = colors[i])
        mplt.legend(loc='lower right')
        i += 1

    mplt.xlabel('False Positive Rate')
    mplt.ylabel('True Positive Rate')
    mplt.title('Curba ROC', fontsize=16)
    mplt.savefig("roc_plot.png")
    mplt.show()

    mlflow.log_artifact("roc_plot.png")
```

In [47]:
```
roc_curve_plot(model_list, x_test, y_test)
```

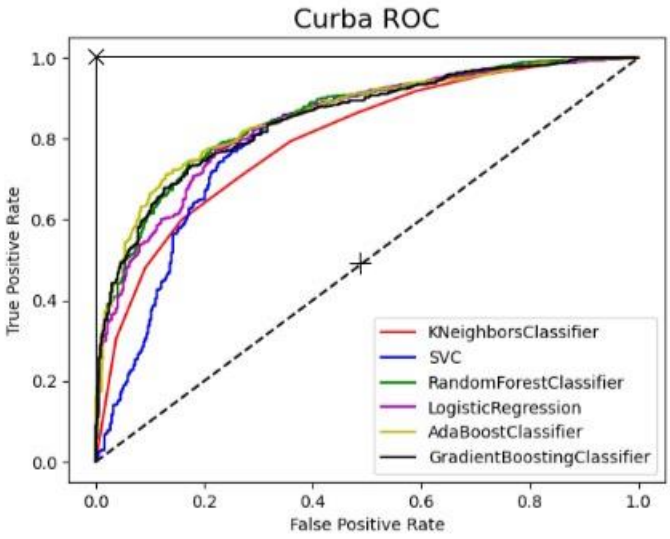

Fig. 4.30. Curba ROC a celor 6 modele.

4.4. ÎNREGISTRAREA MODELELOR ȘI VIZUALIZAREA RULĂRILOR PE MLFLOW UI

Pentru toate cele 6 modele voi executa următorul cod pentru a seta numele experimentelor, a începe rulările, a înregistra valorile și a salva graficele sub formă de artefacte:

```
In [70]: model = lr_model
         mlflow.set_experiment(str(model).split('(')[0]+"_experiment")
         with mlflow.start_run():
             train(model, x_train, y_train)
             evaluate(model, x_test, y_test)
             roc_curve_model(model, x_test, y_test)
             conf_matrix(model, x_test, y_test)
             mlflow.sklearn.log_model(model, "log_reg_model")
             print("Model run: ", mlflow.active_run().info.run_uuid)
             mlflow.end_run()
```

Fig. 4.31. Executarea codului sub forma unei rulări MLflow.

- mlflow.set_experiment(): pune o rulare sub numele unui experiment;
- with mlflow.start_run(): … mlflow.end_run(): începe rularea, execută ce se află în blocul de cod și încheie rularea.

După rularea cu succes a codului pentru fiecare model obțin un folder numit “mlruns” care conține cele 6 experimente:

```
C:\Users\LENOVO\Desktop\Licenta_Alin\Notebook\mlruns>dir
 Volume in drive C has no label.
 Volume Serial Number is E608-C449

 Directory of C:\Users\LENOVO\Desktop\Licenta_Alin\Notebook\mlruns

06/11/2023  08:48 PM    <DIR>          .
06/11/2023  08:48 PM    <DIR>          ..
06/11/2023  08:47 PM    <DIR>          .trash
06/11/2023  08:18 PM    <DIR>          0
06/11/2023  08:41 PM    <DIR>          209053017637879298
06/11/2023  08:39 PM    <DIR>          227651764106006198
06/11/2023  08:39 PM    <DIR>          517909120128043270
06/11/2023  08:40 PM    <DIR>          553444459674728715
06/11/2023  08:40 PM    <DIR>          684061903378515244
06/11/2023  08:41 PM    <DIR>          878791855094882518
06/11/2023  08:55 PM    <DIR>          models
               0 File(s)              0 bytes
              11 Dir(s)  305,778,835,456 bytes free
```

Fig. 4.32. Id-ul experimentelor obținute.

Se poate observa că numele experimentelor sunt sub forma unui șir de cifre, diferit față de ceea ce am setat cu mlflow.set_experiment(). Numele setat de mine va fi vizibil pe interfața de utilizator, nu și în directorul “mlruns”.

Dacă deschid aceste foldere, voi putea vedea rulările, valorile, artefactele și fișierele auto-generate pentru fiecare model, însă nu voi face asta. De acum înainte voi folosi interfața de utilizator al lui MLflow pentru că este ușor de folosit, pot vizualiza foarte ușor fiecare experiment, rulare, valoare, artefact și pot compara cu ușurință rezultatele modelelor.

Următorii pași sunt necesari pentru a putea folosi interfața de utilizator al lui MLflow:

- Conda environment trebuie să fie activ: aici se află pachetele instalate, inclusiv mlflow;
- deschiderea unei linii de comandă (cmd) în interiorul directorului unde se află fișierul Jupyter notebook și noul director “mlruns” creat în urma rulării codului de mai sus;
- în cmd trebuie rulată comanda “mlflow ui -p 1234”: comanda “mlflow ui” găzduiește local interfața de utilizator al lui MLFlow pe portul 5000, iar cu “-p 1234” specific ca găzduirea interfeței să fie pe portul 1234, nu pe 5000.

```
C:\Users\LENOVO\Desktop\Licenta_Alin\Notebook>dir
 Volume in drive C has no label.
 Volume Serial Number is E608-C449

 Directory of C:\Users\LENOVO\Desktop\Licenta_Alin\Notebook

06/11/2023  03:55 PM    <DIR>          .
06/11/2023  03:55 PM    <DIR>          ..
06/11/2023  01:06 AM    <DIR>          .ipynb_checkpoints
06/11/2023  12:55 AM         5,048,978 AnalizaDate.ipynb
06/11/2023  03:54 PM            30,804 lr_roc_plot.png
06/11/2023  03:45 PM    <DIR>          mlruns
06/11/2023  03:55 PM           221,251 Model.ipynb
06/11/2023  03:54 PM            19,136 sklearn_conf_matrix.png
06/11/2023  03:46 PM            57,543 sklearn_roc_plot.png
06/11/2023  01:10 AM            49,119 Untitled.ipynb
06/04/2023  04:55 PM         1,019,731 WA_Fn-UseC_-Telco-Customer-Churn.csv
               7 File(s)      6,446,562 bytes
               4 Dir(s)  309,342,863,360 bytes free

C:\Users\LENOVO\Desktop\Licenta_Alin\Notebook>mlflow ui -p 1234
INFO:waitress:Serving on http://127.0.0.1:1234
```

Fig. 4.33. Setarea interfeței de utilizator al lui MLFlow.

Dacă deschid un browser și scriu http://127.0.0.1:1234 voi putea utiliza interfața de utilizator, care arată cam așa:

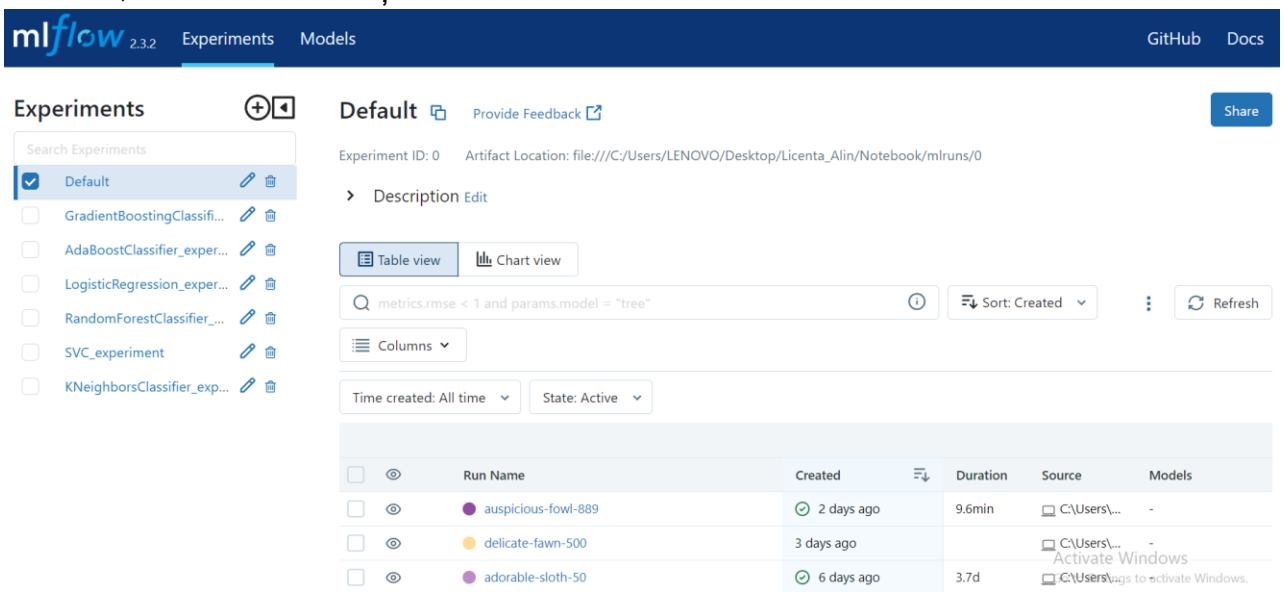

Fig. 4.34. Interfața de utilizator al lui MLflow.

Experimentele înregistrate se pot vedea în partea din stanga, lângă cel default, cu numele setat de mine anterior. De asemenea, pot fi văzute rulările finalizate împreună cu valorile pe care le-am înregistrat folosind mlflow.log_metric().

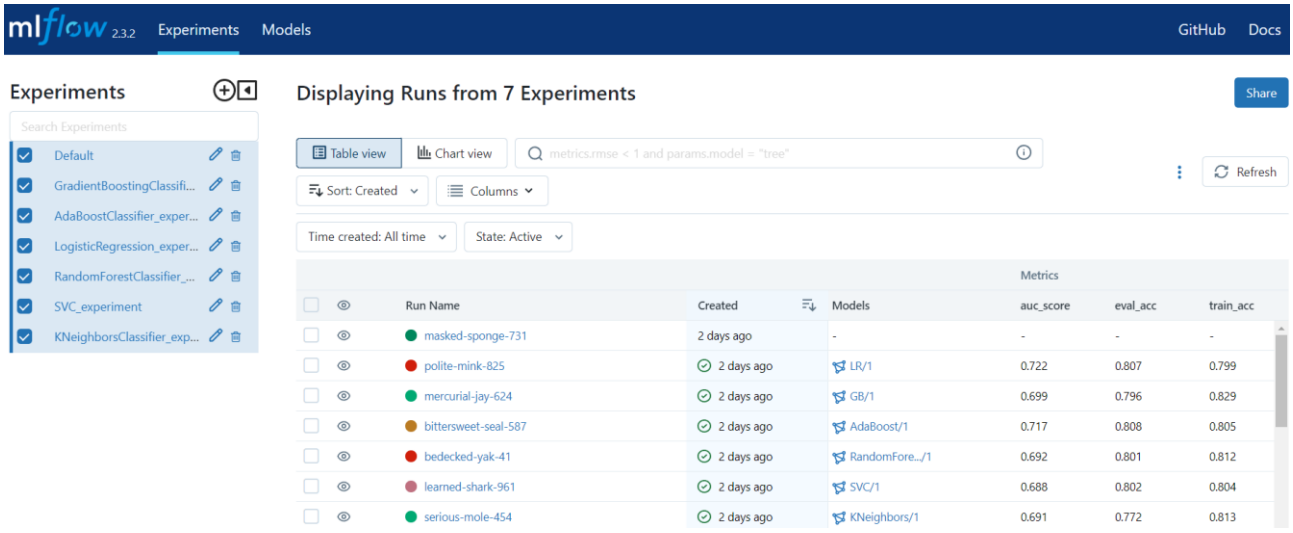

Fig. 4.35. Vizualizarea experimentelor, rulărilor și a valorilor.

Dacă fac click pe oricare dintre aceste rulări voi putea obține informații cu privire la durata rulării, statusul, id-ul, valorile obținute, artefactele înregistrate ș.a.m.d.:

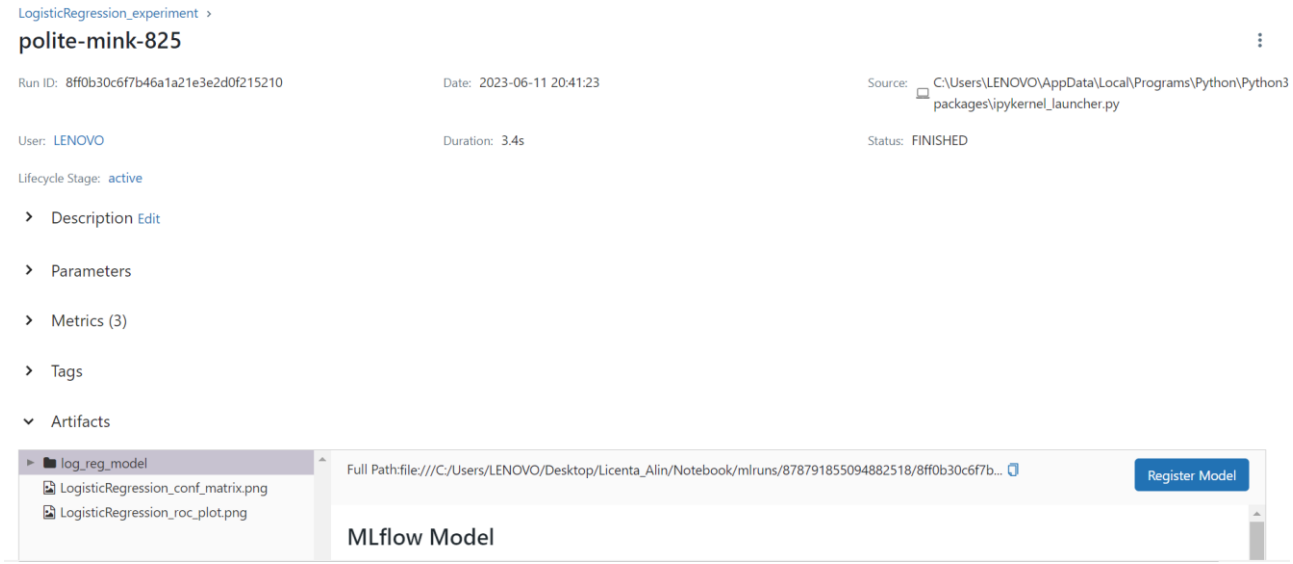

Fig. 4.36. Vizualizarea informațiilor despre rulare.

De aici se face și înregistrarea modelului prin apăsarea butonului “Register Model”. Toate modelele se vor găsi pe pagina “Models” cu numele, versiunea și stadiul fiecăruia:

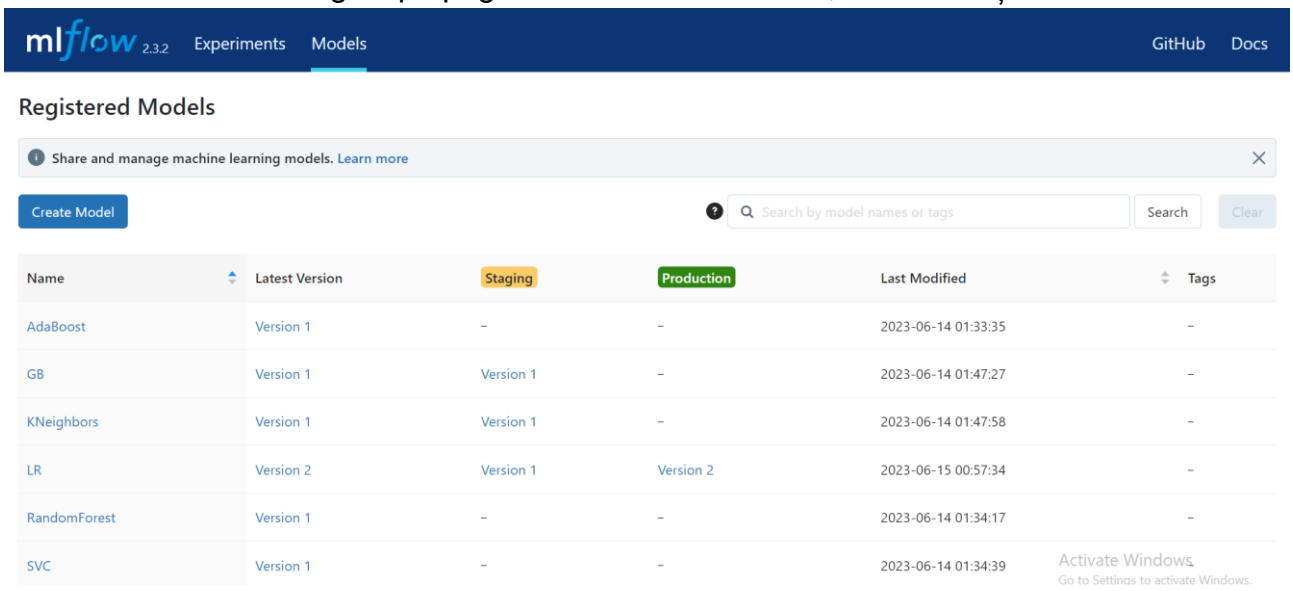

Fig. 4.37. Vizualizarea modelelor înregistrate.

Analizând valorile modelelor, “LogisticRegression” are valoarea auc cea mai mare și acuratețea aproape cea mai mare: 72.189%, respectiv 80.654%.

Pentru a îmi face o idee despre ce înseamnă aceste valori voi vizualiza matricea de confuzie, matrice pe care am creat-o și înregistrat-o prin apelarea funcției conf_matrix() (vezi figura 4.31). În figura 4.38 se poate vedea ce face această funcție.

```
In [64]: def conf_matrix(model, x_test, y_test):
             preds = model.predict(x_test)

             conf_matrix = confusion_matrix(y_test, preds)
             ax = sn.heatmap(conf_matrix, annot=True, fmt='d')
             ax.invert_xaxis()
             ax.invert_yaxis()
             mplt.ylabel('Actual')
             mplt.xlabel('Predicted')
             mplt.title("Confusion Matrix")
             mplt.savefig(str(model).split('(')[0]+"_conf_matrix.png")

             mlflow.log_artifact(str(model).split('(')[0]+"_conf_matrix.png")
```

Fig. 4.38. Funcția conf_matrix() care creează și înregistrează matricea de confuzie.

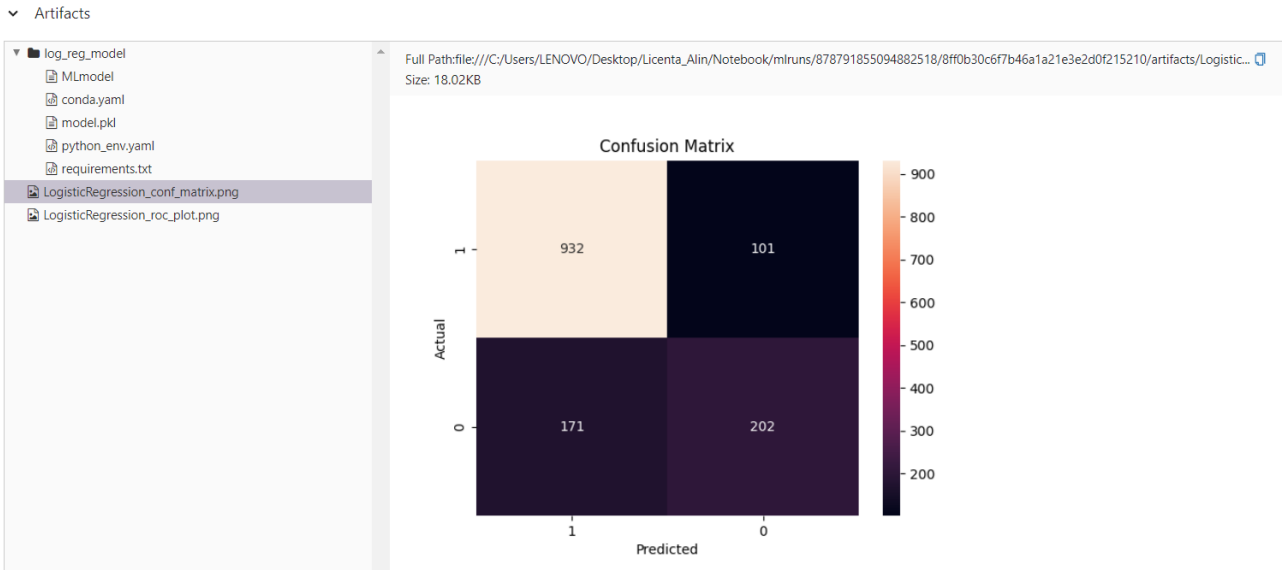

Fig. 4.39. Matricea de confuzie a modelului LogisticRegression pentru datele de test.

Din matrice obțin următoarele informații:

- din totalul de 1406 de valori reale, 1033 (932+101) reprezintă numărul clienților care nu au abandonat și 373 (171+202) reprezintă numărul clienților care au abandonat:
- din cele 1033 de valori algoritmul prezice că 932 nu au abandonat și 101 că au abandonat;
- din cele 373 de valori algoritmul prezice că 171 nu au abandonat și 202 că au abandonat.

Voi face un “cross-validation” cu datele de validare pentru a vedea cum se comportă modelul cu date noi, neprocesate. Rezultatele vor fi înregistrate și vor putea fi vizualizate pe interfață:

```
In [71]: from sklearn.model_selection import cross_val_score
         import numpy as np

         acc = cross_val_score(lr_model, x_val, y_val, scoring="roc_auc", cv=10)
         i = 0
         for el in acc:
             mlflow.log_metric("eval_acc_"+str(i), el)
             i+=1
         mlflow.log_metric("mean_eval_acc", np.mean(acc))
```

Fig. 4.40. Codul folosit pentru “cross-validation”.

5. REZULTATE

În figura 4.41 se poate vedea că modelul își păstrează consistența. Valorile obținute sunt cuprinse între 66.9 - 92.7%, însă majoritatea se învârt în jurul valorii de 80%. Media celor 10 valori obținute în urma acestei validări este de 82.9%, care este mai mare decât valoarea acurateții obținută cu datele de test sau de antrenare (0.799 antrenare, 0.807 testare).

Metrics (11)

Name	Value
eval_acc_0	0.883
eval_acc_1	0.834
eval_acc_2	0.859
eval_acc_3	0.892
eval_acc_4	0.829
eval_acc_5	0.669
eval_acc_6	0.927
eval_acc_7	0.771
eval_acc_8	0.799
eval_acc_9	0.824
mean_eval_acc	0.829

Fig. 4.41. Acuratețea modelului obținută după “cross-validation”.

Așadar, am obținut un model cu o valoare auc de peste 0.70 și o acuratețe de aproximativ 0.80. Performanța este una bună, iar modelul a fost înregistrat cu succes pe serverul local cu ajutorul lui MLflow. Acesta poate fi oricând apelat și retestat pe date noi.

6. CONCLUZII

Este dificil să alegem care este instrumentul potrivit pentru implementarea unui model în producție și contează foarte mult ce anume căutăm la un astfel de instrument. Domeniul de învățare automată s-a dezvoltat rapid și asta a dus la o creștere automată a numărului de instrumente pentru ușurarea muncii noastre și cu toate că sunt ghiduri și articole despre fiecare, utilizarea lor s-ar putea dovedi mai dificilă decât ar părea la prima vedere.

Fiecare are avantajele și dezavantajele sale, iar eu am considerat că cel mai potrivit instrument pentru implementarea modelului din această lucrare este MLflow. Cu ajutorul său am putut să arăt întregul proces de implementare al unui model de învățare automată.

Este un instrument destul de ușor de folosit, nu are multe cerințe pentru a face implementarea modelului, folosindu-se doar de pachetul mlflow și de un environment Conda. Un avantaj enorm este că dispune de o componentă foarte eficientă de urmărire a experimentelor și rulărilor prin care putem extrage orice informație care a fost înregistrată, precum valorile parametrilor, durata și numele rulărilor, compararea lor și multe altele.

Ne permite o foarte bună organizare a modelelor: fiecare model are un nume, propriile valori și fișiere. Astfel, ne este mai lejer să le ținem evidența. Dispune de o interfață de utilizator ușor de folosit, iar asta ne permite să vizualizăm ceea ce înregistrăm.

Așadar, consider că MLflow este o soluție eficientă atunci când vine vorba despre implementarea modelelor de învățare automată în producție.

7. BIBLIOGRAFIE

[1] VibeThemes, “What is Machine Learning? | Machine Learning Techniques | GangBoard,” *Online Courses with Experts for AWS, Python, Data Science, Selenium, RPA, Devops & More*, May 20, 2019. https://www.gangboard.com/blog/what-is-machine-learning (accessed Mar. 18, 2023).

[2] G. Symeonidis, E. Nerantzis, A. Kazakis, and G. A. Papakostas, “MLOps -- Definitions, Tools and Challenges.” arXiv, Jan. 01, 2022. Accessed: Apr. 11, 2023. [Online]. Available: http://arxiv.org/abs/2201.00162

[3] R. Subramanya, S. Sierla, and V. Vyatkin, “From DevOps to MLOps: Overview and Application to Electricity Market Forecasting,” *Applied Sciences*, vol. 12, no. 19, p. 9851, Sep. 2022, doi: 10.3390/app12199851.

[4] “The Big Book Of MLOps,” *Databricks*. https://www.databricks.com/explore/data-science-machine-learning/big-book-of-mlops (accessed Apr. 09, 2023).

[5] P. Ingle, “Top Machine Learning Model Deployment Tools For 2022,” *MarkTechPost*, Oct. 14, 2022. https://www.marktechpost.com/2022/10/14/top-machine-learning-model-deployment-tools-for-2022/ (accessed Mar. 18, 2023).

[6] “A 4-Step Guide to Machine Learning Model Deployment.” https://www.dominodatalab.com/blog/machine-learning-model-deployment (accessed Mar. 18, 2023).

[7] “What is Data Preparation? An In-Depth Guide to Data Prep,” *Business Analytics*. https://www.techtarget.com/searchbusinessanalytics/definition/data-preparation (accessed Apr. 10, 2023).

[8] “What is Exploratory Data Analysis? | IBM.” https://www.ibm.com/topics/exploratory-data-analysis (accessed Apr. 10, 2023).

[9] H. Patel, “What is Feature Engineering — Importance, Tools and Techniques for Machine Learning,” *Medium*, Sep. 02, 2021. https://towardsdatascience.com/what-is-feature-engineering-importance-tools-and-techniques-for-machine-learning-2080b0269f10 (accessed Apr. 10, 2023).

[10] S. Alla and S. K. Adari, *Beginning MLOps with MLFlow: Deploy Models in AWS SageMaker, Google Cloud, and Microsoft Azure*. Berkeley, CA: Apress, 2021. doi: 10.1007/978-1-4842-6549-9.

[11] A. Grigorev, “MLOps in 10 Minutes,” *Medium*, May 04, 2022. https://towardsdatascience.com/mlops-in-10-minutes-165c746a9b8e (accessed Apr. 05, 2023).

[12] “Model Deployment: 3 Steps & Top Tools in 2023.” https://research.aimultiple.com/model-deployment/ (accessed Apr. 13, 2023).

[13] A. Luigi, “Batch Inference for Machine Learning Deployment (Deployment Series: Guide 03),” *ML in Production*, Feb. 19, 2020. https://mlinproduction.com/batch-inference-for-machine-learning-deployment-deployment-series-03/ (accessed Apr. 13, 2023).

[14] N. Gift and A. Deza, *Practical MLOps*. O’Reilly Media, Inc., 2021.

[15] Asigmo, “MLflow best practices and lessons learned,” *Asigmo*, May 01, 2020. https://www.asigmo.com/post/mlflow-best-practices-and-lessons-learned (accessed Apr. 22, 2023).

[16] Y. Callaert, “Getting started with mlFlow,” *Medium*, Aug. 26, 2019. https://towardsdatascience.com/getting-started-with-mlflow-52eff8c09c61 (accessed Apr. 23,

2023).
[17] K. Kaewsanmua, “Best 8 Machine Learning Model Deployment Tools That You Need to Know,” *neptune.ai*, Jul. 21, 2022. https://neptune.ai/blog/best-8-machine-learning-model-deployment-tools (accessed Apr. 23, 2023).
[18] “MLflow Tracking — MLflow 2.3.0 documentation.” https://mlflow.org/docs/latest/tracking.html#concepts (accessed Apr. 23, 2023).
[19] “MLflow Projects — MLflow 2.3.0 documentation.” https://mlflow.org/docs/latest/projects.html (accessed Apr. 23, 2023).
[20] “MLflow Models — MLflow 2.3.1 documentation.” https://mlflow.org/docs/latest/models.html (accessed May 02, 2023).
[21] “MLflow Model Registry — MLflow 2.3.1 documentation.” https://mlflow.org/docs/latest/model-registry.html (accessed May 02, 2023).
[22] “MLflow - A platform for the machine learning lifecycle,” *MLflow*. https://mlflow.org/ (accessed Apr. 22, 2023).
[23] T. Shin, “Five Technologies to Deploy Your Machine Learning Models,” *Medium*, Mar. 11, 2021. https://towardsdatascience.com/five-technologies-to-deploy-your-machine-learning-models-bddaad69e0d4 (accessed May 23, 2023).
[24] A. Kothari, “7 Platform-as-a-Service (PaaS) for Machine Learning and AI Developers,” *Geekflare*, Dec. 20, 2019. https://geekflare.com/machine-learning-paas/ (accessed May 16, 2023).
[25] J. Simon, *Learn Amazon SageMaker: A guide to building, training, and deploying machine learning models for developers and data scientists*. Packt Publishing Ltd, 2020.
[26] “What is SageMaker in AWS?,” *GeeksforGeeks*, Jul. 22, 2020. https://www.geeksforgeeks.org/what-is-sagemaker-in-aws/ (accessed May 22, 2023).
[27] “Use Amazon SageMaker Built-in Algorithms or Pre-trained Models - Amazon SageMaker.” https://docs.aws.amazon.com/sagemaker/latest/dg/algos.html (accessed May 22, 2023).
[28] A. Kumaraswamy, “10 reasons why data scientists love Jupyter notebooks,” *Packt Hub*, Apr. 05, 2018. https://hub.packtpub.com/10-reasons-data-scientists-love-jupyter-notebooks/ (accessed May 23, 2023).
[29] “Seldon Core - OSS Model Deployment,” *Seldon*. https://www.seldon.io/solutions/open-source-projects/core (accessed May 28, 2023).
[30] S. Raghuram, “4 reasons you should use Kubernetes,” *InfoWorld*, Feb. 23, 2017. https://www.infoworld.com/article/3173266/4-reasons-you-should-use-kubernetes.html (accessed May 28, 2023).
[31] A. Housley, “Introducing Seldon Core — Machine Learning Deployment for Kubernetes,” *Medium*, Jan. 28, 2018. https://medium.com/seldon-open-source-machine-learning/introducing-seldon-core-machine-learning-deployment-for-kubernetes-e10e94c19fd8 (accessed May 28, 2023).
[32] “Overview of Seldon Core Components — seldon-core documentation.” https://docs.seldon.io/projects/seldon-core/en/latest/workflow/overview.html#seldon-core-operator (accessed May 29, 2023).
[33] “Kubeflow,” *Kubeflow*. https://www.kubeflow.org/ (accessed May 29, 2023).
[34] “Components,” *Kubeflow*. https://www.kubeflow.org/docs/components/ (accessed May 29, 2023).
[35] M. Simmons, “Kubeflow Pros and Cons: Kubeflow vs Airflow vs SageMaker,” *Datasparq Technology*, Apr. 13, 2022. https://medium.com/datasparq-technology/kubeflow-pros-and-cons-kubeflow-vs-airflow-vs-sagemaker-4942d7e7910a (accessed May 29, 2023).

[36] "A Comprehensive Comparison Between Kubeflow and Databricks." https://valohai.com/blog/kubeflow-vs-databricks/ (accessed May 30, 2023).
[37] "The Unified Model Serving Framework." BentoML, May 30, 2023. Accessed: May 30, 2023. [Online]. Available: https://github.com/bentoml/BentoML
[38] V. Karbhari, "ML Ops: Real World Model Deployments," *Medium*, Nov. 26, 2020. https://medium.com/acing-ai/ml-ops-real-world-model-deployments-afab1fff9361 (accessed Apr. 05, 2023).
[39] "Unified Model Serving Framework," *BentoML Documentation*. http://docs.bentoml.org/index.html (accessed May 30, 2023).
[40] "BentoML: Build, Ship, Scale AI Applications." https://www.bentoml.com/ (accessed May 31, 2023).
[41] "Machine Learning model serving tools comparison - KServe, Seldon Core, BentoML." https://getindata.com/blog/machine-learning-model-serving-tools-comaprison-kserve-seldon-core-bentoml/ (accessed May 31, 2023).
[42] "Telco Customer Churn." https://www.kaggle.com/datasets/blastchar/telco-customer-churn (accessed Jun. 04, 2023).
[43] E. Zvornicanin, "Accuracy vs AUC in Machine Learning | Baeldung on Computer Science," Dec. 16, 2021. https://www.baeldung.com/cs/ml-accuracy-vs-auc (accessed Jun. 12, 2023).