
**Tema 5 Arbori de decizie**

# Sisteme Semantice

Draghici Andreea-Maria
Inginerie Software
IS 2.1

# Tema 5 Arbori de decizie

## Contents

## 1. Introducere în arborii de decizie

Arborii de decizie sunt metode de învățare supervizată utilizate pentru clasificare și regresie. Ideea principală este de a identifica caracteristicile (atributele) care contribuie cel mai mult la separarea seturilor de date, utilizând concepte precum **entropia** și **câștigul informațional**.

### 1.1 Componentele unui arbore de decizie:

- **Nod rădăcină**: Caracteristica cea mai informativă, care inițiază împărțirea datelor.
- **Noduri interne**: Puncte de decizie care reprezintă împărțiri ulterioare.
- **Frunze**: Rezultate finale care clasifică datele.

### 1.2 Concepte cheie:

1. **Entropia**: Măsoară impuritatea unui set de date.
2. **Câștigul informațional**: Diferența dintre entropia inițială și entropia după împărțirea datelor.

## 2. Algoritmul ID3 pentru construcția arborilor

Algoritmul ID3 este o metodă populară pentru generarea arborilor de decizie. Se bazează pe maximizarea câștigului informațional la fiecare pas.

### Pași principali:

1. **Calcularea entropiei inițiale** pentru caracteristica țintă:
    - Se determină proporțiile claselor din setul de date.
    - Se aplică formula entropiei.
2. **Calculul câștigului informațional** pentru fiecare atribut:
    - Pentru fiecare caracteristică, se împart datele în subseturi.
    - Se calculează entropia fiecărui subset și entropia ponderată a împărțirii.
    - Câștigul este diferența dintre entropia inițială și cea ponderată.
3. **Selectarea atributului cu cel mai mare câștig** pentru a forma un nod:
    - Atributul ales împarte datele în mod optim.

4. **Partiționarea datelor** pe baza valorilor atributului selectat:
    - Datele sunt separate în subseturi mai omogene.
5. **Recursivitate**: Repetarea procesului pentru fiecare subset până la atingerea unui criteriu de oprire:
    - Toate datele dintr-un subset aparțin aceleiași clase.
    - Nu mai sunt atribute disponibile pentru împărțire.

## 3. Implementare în Python

### 3.1 Fișierul solution.py

Codul din fișierul enumerat implementează un exemplu de utilizare a unui arbore decizional pentru a clasifica datele privind promovarea angajaților. Acesta include mai mulți pași esențiali de procesare a datelor, antrenare a modelului, vizualizare și calcul a metricilor.

**Generarea unui set de date:**

```
set_date = {
    "Experientă" : [5, 2, 7, 10, 1, 4, 6, 3],  # Ani de experientă
    "Educatie" : ["Licentă", "Liceu", "Master", "Doctorat", "Liceu", "Licentă", "Master", "Liceu"],
    "Ore_suplimentare" : ["Da", "Nu", "Da", "Nu", "Da", "Da", "Nu", "Nu"],
    "Vârstă" : [29, 22, 35, 45, 19, 30, 40, 25],
    "Departament" : ["IT", "HR", "IT", "Management", "HR", "IT", "Management", "HR"],
    "Promovat" : ["Da", "Nu", "Da", "Da", "Nu", "Da", "Nu", "Nu"]
}

# Convertim valorile categorice în valori numerice
set_date = pd.DataFrame(set_date)
set_date = pd.get_dummies(set_date, columns=["Educatie", "Ore_suplimentare", "Departament"], drop_first=True)
```

Se folosesc funcții de la pandas pentru a transforma datele categorice în valori numerice utilizând pd.get_dummies. De exemplu, coloana „Educație” este transformată în mai multe coloane binare (una pentru fiecare nivel educațional).

**Antrenarea arborelui decisional:**

- **criterion="entropy"**: Folosește entropia pentru a măsura impuritatea nodurilor.
- **random_state=42**: Asigură reproductibilitatea rezultatelor.

- **max_depth=4**: Arborele are o adâncime maximă de 4.

```
    # Separăm caracteristicile de țintă
    X = set_date.drop("Promovat", axis=1)
    y = set_date["Promovat"]

    # Creăm și antrenăm arborele decizional
    model = DecisionTreeClassifier(criterion="entropy", random_state=42, max_depth=4)
    model.fit(X, y)
```

Se utilizează biblioteca sklearn.

**Vizualizarea arborelui,** graficul ajută la interpretarea vizuală**:**

```
# Vizualizăm arborele decizional
def vizualizeaza_arbore(model, feature_names) :
    plt.figure(figsize=(9, 8))
    plot_tree(model, feature_names=feature_names, class_names=model.classes_, filled=True)
    plt.show()
```

**Calculul entropiei:** O funcție separata calculează entropia pentru orice subset:

```
# Functie pentru calcularea entropiei
def calculeaza_entropia(y) :
    valori_unice, frecvente = np.unique(y, return_counts=True)
    probabilitati = frecvente / len(y)
    entropia = -np.sum(probabilitati * np.log2(probabilitati))
    return entropia
```

**Câștigul informațional:** Implementarea compară entropia inițială cu cea condiționată de un atribut:

```python
# Calculează câștigul informational
def castig_informational(y, feature) :
    entropie_initiala = calculeaza_entropia(y)
    valori_unice, frecvente = np.unique(feature, return_counts=True)
    entropie_conditionala = 0

    for valoare, frecventa in zip(valori_unice, frecvente) :
        subset = y[feature == valoare]
        entropie_conditionala += (frecventa / len(feature)) * calculeaza_entropia(subset)

    return entropie_initiala - entropie_conditionala

# Exemplu calcul câștig informational
castiguri = {col : castig_informational(y, X[col]) for col in X.columns}
print("\nCâștiguri informationale pentru fiecare caracteristică:")
print(castiguri)
```

**Histogramă pentru fiecare caracteristică numerică**:

- Se afișează distribuția fiecărei caracteristici numerice sub formă de histogramă.
- Este util pentru a analiza densitatea valorilor caracteristicilor.

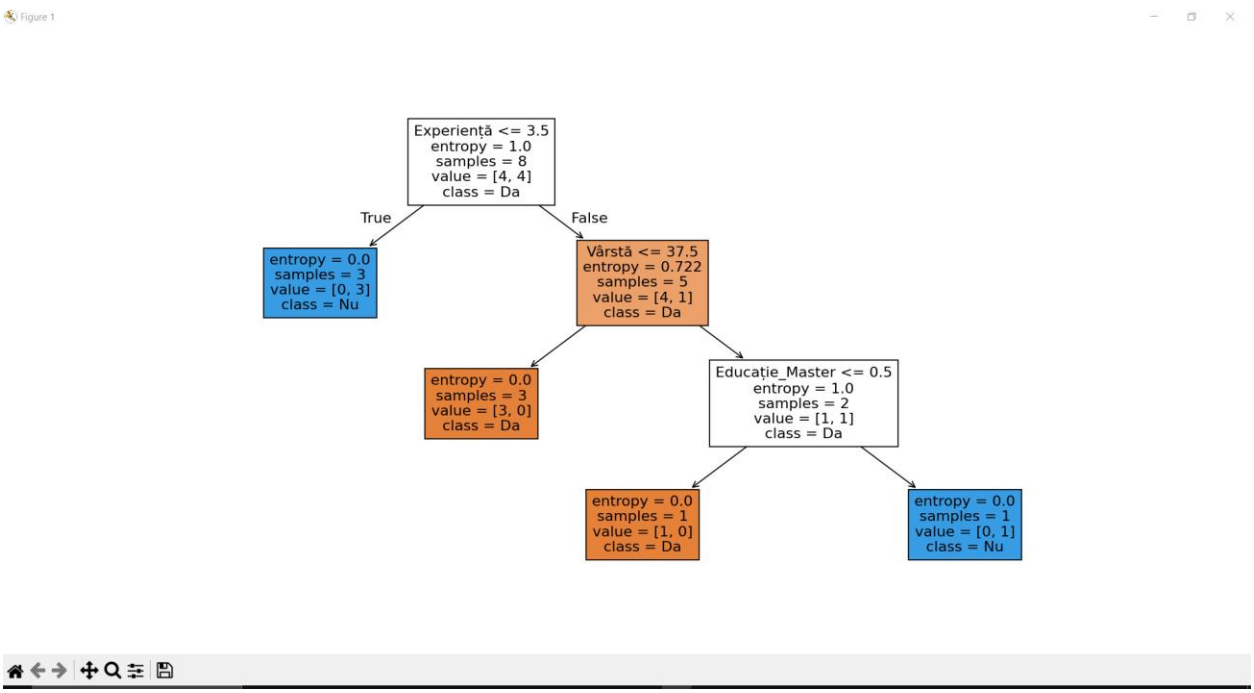
**Pairplot pentru relațiile dintre caracteristici numerice**:

- Vizualizează relațiile dintre toate caracteristicile numerice și clasa țintă.
- Include diagrama de densitate pentru distribuțiile univariate.

```python
    # Histogramă pentru fiecare caracteristică numerică
    X.hist(bins=10, figsize=(10, 8), color='skyblue', edgecolor='black')
    plt.suptitle("Histogramă pentru fiecare caracteristică numerică", fontsize=12)
    plt.show()

    # Pairplot pentru relatiile dintre caracteristici numerice
    sns.pairplot(set_date, diag_kind="kde", corner=True)
    plt.suptitle("Relatii între caracteristici (Pairplot)", fontsize=12)
    plt.show()
```

### 3.1.1 Output Solution.py



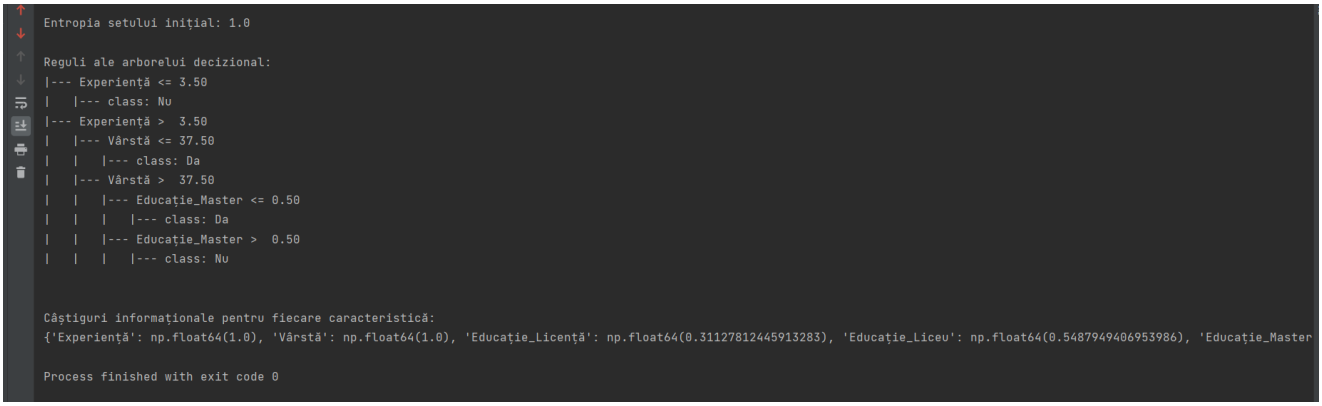

Aceasta este o vizualizare grafică a unui arbore decizional generat cu setul de date și modelul explicat anterior. Arborele decizional utilizează caracteristici precum **Experiență**, **Vârstă**, și **Educație_Master** pentru a prezice dacă o persoană este promovată („Da”) sau nu („Nu”).

### Elemente explicate din arbore:

1. **Rădăcina arborelui**:
   - Prima decizie este bazată pe **Experiență <= 3.5**.
   - Entropia inițială a setului este 1.0 (înseamnă că setul este complet impur, având atât clase „Da”, cât și „Nu” în proporții egale).
   - Setul de date este împărțit în două subseturi:
     - **True (stânga):** persoane cu experiență mai mică sau egală cu 3.5.
     - **False (dreapta):** persoane cu experiență mai mare de 3.5.
2. **Noduri de frunză (stânga)**:
   - În ramura **True**, avem un nod terminal (frunză) cu:
     - Entropie 0.0 (setul este complet pur, toate valorile sunt „Nu”).

- Acest subset conține 3 mostre și toate au fost clasificate ca **„Nu”**.

3. **Noduri intermediare (dreapta)**:
   - Dacă **Experiență > 3.5**, arborele verifică **Vârstă <= 37.5**:
     - **True (stânga):** persoane cu vârstă mai mică sau egală cu 37.5.
     - **False (dreapta):** persoane cu vârstă mai mare de 37.5.
4. **Continuarea ramurii pentru Vârstă**:
   - Pentru **Vârstă <= 37.5**, toate valorile sunt clasificate ca **„Da”**.
   - Pentru **Vârstă > 37.5**, este verificată educația, folosind condiția **Educație_Master <= 0.5**:
     - **True:** persoane care nu au master (valoarea este 0).
     - **False:** persoane care au master (valoarea este 1).
5. **Noduri terminale (frunze)**:
   - Persoanele fără master sunt clasificate ca **„Da”**.
   - Persoanele cu master sunt clasificate ca **„Nu”**.




Imaginea arată rezultatele scriptului Python executat, care include:

### 1. Entropia setului inițial

- Valoarea 1.0 indică faptul că setul de date inițial este complet impur. Aceasta înseamnă că atât clasele „Da” cât și „Nu” sunt distribuite în proporții egale.

### 2. Regulile arborelui decizional

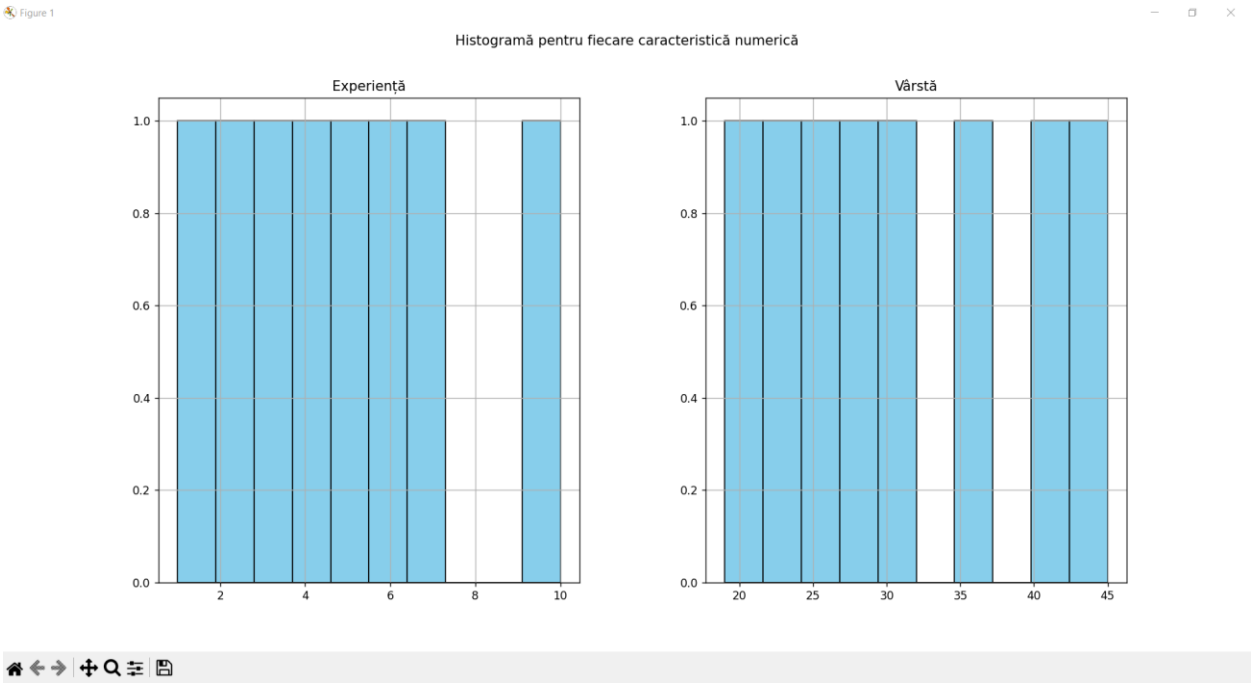
Regulile extrase din arborele decizional sunt structurate astfel:

1. **Experiență <= 3.50**:
    - Dacă această condiție este adevărată, clasa este **„Nu”**.
2. **Experiență > 3.50**:
    - Dacă această condiție este adevărată, se analizează:
        - **Vârstă <= 37.50**:
            - Dacă această condiție este adevărată, clasa este **„Da”**.
        - **Vârstă > 37.50**:
            - Dacă această condiție este adevărată, se analizează:
                - **Educație_Master <= 0.50**:
                    - Dacă această condiție este adevărată, clasa este **„Da”**.
                - **Educație_Master > 0.50**:
                    - Dacă această condiție este adevărată, clasa este **„Nu”**.

## 3. Câștigurile informaționale

Câștigul informațional pentru fiecare caracteristică este calculat și afișat:

- **Experiență**: 1.0 (cel mai mare câștig informațional, deci această caracteristică este prima folosită în arbore).
- **Vârstă**: 0.3113 (mai puțin relevantă decât „Experiență”).
- **Educație_Licență**: 0.0 (nu oferă informații suplimentare pentru împărțirea datelor).
- **Educație_Liceu**: 0.5488 (moderată relevanță).
- **Educație_Master**: 0.5581 (relevanță moderată, dar folosită la ramurile mai adânci).



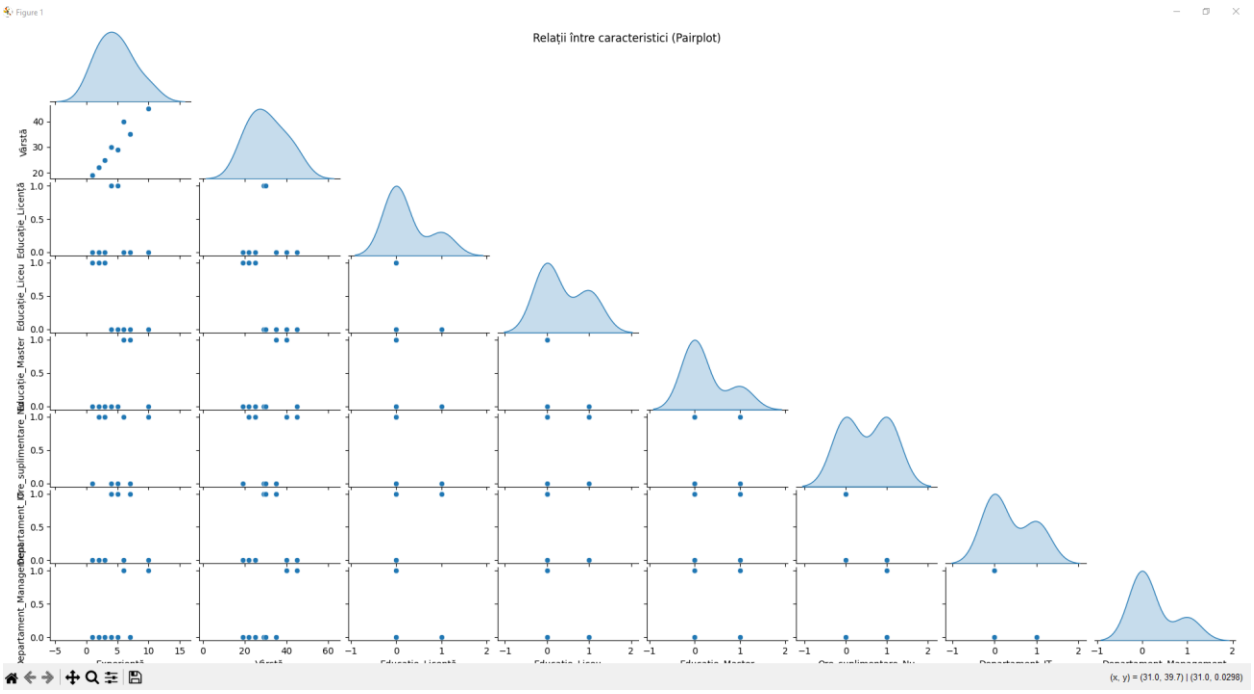

Acest grafic este o **histogramă** generată pentru caracteristicile numerice din setul de date (Experiență și Vârstă).

## Interpretarea graficului:

1. **Histogramă pentru Experiență:**
   - Afișează distribuția numărului de ani de experiență în setul de date.
   - Observăm că valorile sunt relativ uniform distribuite, fără concentrații mari pe anumite valori.
2. **Histogramă pentru Vârstă:**
   - Reprezintă distribuția vârstelor în setul de date.
   - Există o răspândire destul de uniformă, acoperind un interval de la 19 la 45 de ani.

## Scopul histogramelor:

- Identificarea distribuției fiecărei caracteristici.
- Detectarea eventualelor valori extreme sau distribuții neuniforme.




Imaginea este un **pairplot** generat pentru a analiza relațiile dintre caracteristicile numerice din setul de date. Acest tip de grafic ajută la înțelegerea corelațiilor și la observarea distribuției fiecărei caracteristici.

---

## Interpretarea graficului:

1. **Diagramele pe diagonală:**
   - Reprezintă distribuțiile univariabile ale fiecărei caracteristici numerice (e.g., Vârstă, Experiență).
   - Aceste diagrame utilizează estimări kernel de densitate (KDE) pentru a evidenția tendințele generale ale valorilor.
2. **Diagramele din afara diagonalei:**
   - Prezintă relațiile pereche între caracteristicile numerice. Fiecare punct reprezintă o instanță din setul de date.
   - Permite identificarea unor corelații sau tipare potențiale între caracteristici.

### Exemple de analize din grafic:

1. **Relația dintre Vârstă și Experiență:**
   - Se observă o tendință pozitivă între aceste două variabile, ceea ce este logic (persoanele mai în vârstă tind să aibă mai multă experiență).
2. **Distribuția variabilelor binare (Educație_Licență, Educație_Master):**
   - Valorile sunt bine separate și vizibile în relație cu alte caracteristici, dar nu prezintă o variație continuă (fiind codificate binar).

### Scopul acestui tip de vizualizare:

- Ajută la înțelegerea relațiilor dintre caracteristici, ceea ce poate ghida selecția caracteristicilor relevante.
- Permite observarea distribuțiilor și a potențialelor valori anormale.

## 3.2 Fișierul solution2.py

Codul prezentat implementează un arbore de decizie folosind algoritmul ID3, care a fost implementat manual. Acesta este un algoritm pentru construirea arborilor de decizie bazat pe măsura câștigului Information Gain.

```
9   class ID3Tree :
10      def __init__(self) :
11          self.tree = None
12
13      def calculeaza_entropia(self, y) :
14          valori_unice, frecvente = np.unique(y, return_counts=True)
15          probabilitati = frecvente / len(y)
16          entropia = -np.sum(probabilitati * np.log2(probabilitati))
17          return entropia
18
19      def castig_informational(self, y, feature) :
20          entropie_initiala = self.calculeaza_entropia(y)
21          valori_unice, frecvente = np.unique(feature, return_counts=True)
22          entropie_conditionala = 0
23
24          for valoare, frecventa in zip(valori_unice, frecvente) :
25              subset = y[feature == valoare]
26              entropie_conditionala += (frecventa / len(feature)) * self.calculeaza_entropia(subset)
27
28          gain = entropie_initiala - entropie_conditionala
29          return entropie_initiala, gain
```

```python
def alege_cel_mai_bun_atribut(self, X, y) :
    castiguri = {}
    entropii_initiale = {}
    for col in X.columns :
        entropii_initiale[col], castiguri[col] = self.castig_informational(y, X[col])

    cel_mai_bun = max(castiguri, key=castiguri.get)
    print("\nEntropii initiale pentru fiecare atribut:")
    for atribut, entropie in entropii_initiale.items() :
        print(f" {atribut}: {entropie:.4f}")

    print("\nCâștiguri informationale pentru fiecare atribut:")
    for atribut, castig in castiguri.items() :
        print(f" {atribut}: {castig:.4f}")

    print(f"\nCel mai bun atribut: {cel_mai_bun}")
    return cel_mai_bun
```

```python
def construieste_arbore(self, X, y) :
    # Dacă toate valorile țintă sunt identice, returnează acea valoare
    if len(np.unique(y)) == 1 :
        return y.iloc[0]

    # Dacă nu mai sunt atribute disponibile, returnează clasa majoritară
    if X.empty :
        return y.mode()[0]

    # Alege cel mai bun atribut pentru splitting
    cel_mai_bun_atribut = self.alege_cel_mai_bun_atribut(X, y)
    arbore = {cel_mai_bun_atribut : {}}

    # Creează ramuri pentru fiecare valoare a atributului
    for valoare in np.unique(X[cel_mai_bun_atribut]) :
        subset_X = X[X[cel_mai_bun_atribut] == valoare].drop(columns=[cel_mai_bun_atribut])
        subset_y = y[X[cel_mai_bun_atribut] == valoare]

        # Construiește recursiv arborele pentru fiecare subset
        arbore[cel_mai_bun_atribut][valoare] = self.construieste_arbore(subset_X, subset_y)

    return arbore

def fit(self, X, y) :
    self.tree = self.construieste_arbore(X, y)
```

```python
def predict_sample(self, sample, tree) :
    if not isinstance(tree, dict) :
        return tree

    atribut = next(iter(tree))
    valoare = sample[atribut]

    if valoare in tree[atribut] :
        return self.predict_sample(sample, tree[atribut][valoare])
    else :
        return None  # Valoarea nu există în arbore

def predict(self, X) :
    return X.apply(lambda sample : self.predict_sample(sample, self.tree), axis=1)
```

```python
def print_tree(self, tree=None, indent="", depth=0) :
    if tree is None :
        tree = self.tree

    if not isinstance(tree, dict) :
        print("  " * depth + f"[Leaf] {tree}")
        return

    for key, value in tree.items() :
        print("  " * depth + f"{key}:")
        for subkey, subtree in value.items() :
            print("  " * (depth + 1) + f"{subkey} ->")
            self.print_tree(subtree, indent, depth + 2)

def export_if_then(self, tree=None, indent="IF ") :
    if tree is None :
        tree = self.tree

    if not isinstance(tree, dict) :
        print(indent + f" THEN {tree}")
        return

    for key, value in tree.items() :
        for subkey, subtree in value.items() :
            self.export_if_then(subtree, indent + f"{key} = {subkey} AND ")
```

Clasa implementează un arbore de decizie folosind ID3. Aceasta are mai multe metode:

1. **calculeaza_entropia(self, y)**
    - Această funcție calculează entropia unui set de date S, utilizând formula:

$$\mathbf{Entropy(S) = H(S) = - \Sigma\ p_i * \log_2(p_i)}$$

Unde $p_i$ este proporția elementelor din clasa i.

**2. castig_informational(self, y, feature)**

- Calculează câștigul informațional pentru o caracteristică A. Câștigul informațional este definit ca diferența dintre entropia inițială a setului și entropia ponderată a subseturilor rezultate în urma împărțirii:

$$\mathbf{Gain(A) = Entropy(S) - \Sigma\ (|S_v| / |S|) * Entropy(S_v)}$$

Această metodă este utilizată pentru a alege atributul cel mai "informativ" pentru împărțire.

**3. alege_cel_mai_bun_atribut(self, X, y)**

- Iterează prin toate caracteristicile și determină cel mai mare câștig informațional pentru fiecare caracteristică. Alege atributul cu cel mai mare câștig informațional ca nod de împărțire.

**4. construieste_arbore(self, X, y)**

- Construiește arborele recursiv:
    - Dacă toate etichetele din y sunt identice, returnează această valoare (frunză).
    - Dacă nu mai sunt caracteristici disponibile, returnează clasa majoritară.
    - Alege cel mai bun atribut pentru împărțire și creează ramuri pentru fiecare valoare posibilă a acestuia.

**5. fit(self, X, y)**

- Construiește arborele de decizie pe baza datelor de antrenament.

6. **predict_sample și predict(self, X)**

- Realizează predicția pentru o mostră sau un set de date, traversând arborele de decizie până la o frunză.

7. **print_tree și export_if_then**

- Vizualizează arborele decizional ca text sau generează reguli IF-THEN pe baza structurii arborelui.

Codul implementează ID3 pentru a genera un arbore de decizie pe baza unui set de date sintetizat în funcție de caracteristici precum experiența, educația, orele suplimentare și departamentul.

- **Decizia nodului rădăcină:** Se calculează câștigul de informații pentru fiecare caracteristică, alegând cea cu câștigul maxim.
- **Validare și vizualizare:**
  - Este utilizată biblioteca scikit-learn pentru a construi un arbore similar și a-l vizualiza grafic. Acest lucru validează implementarea personalizată.

```
def main() :
    # Set de date extins pentru exemplificare
    set_date = {
        "Experientă" : [5, 2, 7, 10, 1, 4, 6, 3, 8, 9, 12, 15, 1, 2, 4, 5, 7, 3, 10, 11],
        "Educatie" : ["Licentă", "Liceu", "Master", "Doctorat", "Liceu", "Licentă", "Master", "Liceu", "Licentă",
                      "Doctorat", "Master", "Doctorat", "Liceu", "Liceu", "Licentă", "Licentă", "Master", "Liceu",
                      "Doctorat", "Licentă"],
        "Ore_suplimentare" : ["Da", "Nu", "Da", "Nu", "Nu", "Da", "Nu", "Nu", "Da", "Da", "Nu", "Nu", "Nu", "Da", "Nu",
                              "Da", "Nu", "Nu", "Da", "Nu"],
        "Departament" : ["IT", "HR", "IT", "Management", "HR", "IT", "Management", "HR", "IT", "Management", "IT", "HR",
                         "Management", "HR", "IT", "IT", "Management", "HR", "IT", "Management"],
        "Promovat" : ["Da", "Nu", "Da", "Da", "Nu", "Da", "Nu", "Nu", "Da", "Da", "Nu", "Da", "Nu", "Nu", "Da", "Da",
                      "Nu", "Nu", "Da", "Nu"]
    }

    df = pd.DataFrame(set_date)
```

Creează un set de date fictiv, reprezentând experiența, nivelul de educație, orele suplimentare, departamentul și dacă o persoană a fost promovată sau nu.

```
# Convertim valorile categorice în numerice (dacă este cazul)
df = pd.get_dummies(df, columns=["Educatie", "Ore_suplimentare", "Departament"], drop_first=True)

X = df.drop(columns=["Promovat"])
y = df["Promovat"]

# Convertim toate coloanele în tip numeric
X = X.apply(pd.to_numeric)
```

- Transformă variabilele categorice (Educație, Ore_suplimentare, Departament) în variabile numerice folosind pd.get_dummies().
- Selectează caracteristicile predictive (X) și ținta (y).
- Convertește toate valorile la tip numeric.

Algoritmii de învățare automată, inclusiv arborii decizionali, necesită valori numerice pentru antrenare.

```
# Construim arborele decizional folosind DecisionTreeClassifier pentru grafic
clf = DecisionTreeClassifier(criterion="entropy", max_depth=4, random_state=42)
clf.fit(X, y)

# Vizualizăm arborele decizional ca grafic
plt.figure(figsize=(9, 6))
plot_tree(clf, feature_names=X.columns, class_names=clf.classes_, filled=True, rounded=True)
plt.title("Arborele Decizional - Vizualizare Grafică")
plt.show()
```

- Creează un arbore decizional folosind criteriul entropiei pentru a calcula câștigul informațional.
- Limitează adâncimea maximă a arborelui la 4 pentru a evita suprainstruirea.
- Antrenează arborele pe datele preprocesate.
- Generează o reprezentare grafică a arborelui decizional folosind plot_tree.

```
# Construim arborele decizional folosind ID3Tree
id3 = ID3Tree()
id3.fit(X, y)
```

Construiește un arbore de decizie folosind o implementare manuală a algoritmului ID3 (din clasa definita anterior). Demonstrează cum se poate implementa un arbore decizional fără biblioteci externe.

```
# Afisăm arborele în format text
print(f"Numărul total de noduri: {clf.tree_.node_count}")
print(f"Numărul de frunze: {clf.get_n_leaves()}")
print(f"Adâncimea maximă a arborelui: {clf.get_depth()}")
```

Calculează și afișează:

- Numărul total de noduri.
- Numărul de frunze.
- Adâncimea maximă a arborelui.

```
# Exportăm regulile arborelui în format IF-THEN
print("\nReguli IF-THEN ale arborelui decizional:")
id3.export_if_then()
```

Traduce structura arborelui de decizie într-o serie de reguli logice de tipul IF-THEN. ermite interpretarea ușoară a deciziilor modelului.

```
# 2. Distributia clasei tintă
plt.figure(figsize=(6, 4))
y.value_counts().plot(kind="bar", color="skyblue")
plt.title("Distributia clasei tintă")
plt.xlabel("Clasă")
plt.ylabel("Frecventă")
plt.show()
```

Creează un grafic de tip bară care arată distribuția valorilor din clasa țintă (y). Identifică dezechilibre în clase (e.g., mai multe instanțe de „Nu” decât „Da”).

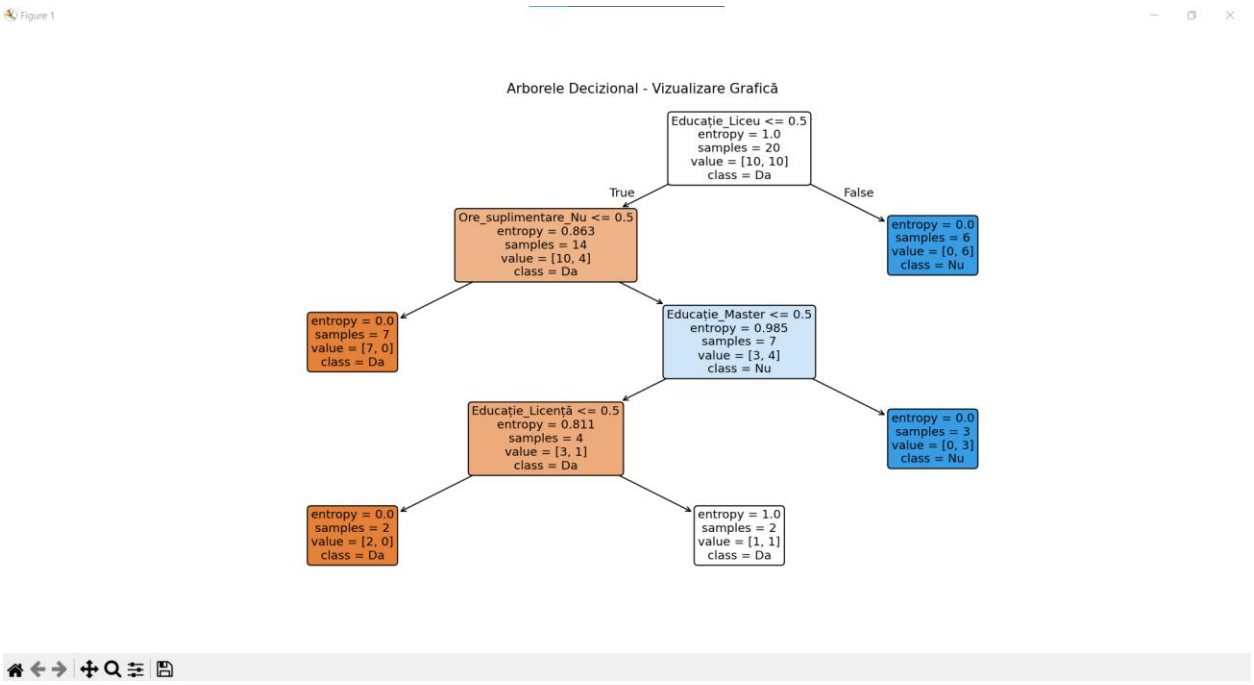
```
# 3. Matricea de corelatie pentru date numerice
plt.figure(figsize=(10, 5))
sns.heatmap(X.corr(), annot=True, fmt=".2f", cmap="coolwarm")
plt.title("Matricea de corelatie între caracteristicile numerice")
plt.show()
```

Generează o hartă de căldură pentru a vizualiza corelațiile dintre caracteristicile numerice. Analizează relațiile dintre caracteristici.

```
# 4. Boxplot pentru caracteristici numerice
plt.figure(figsize=(8, 8))
sns.boxplot(data=X)
plt.title("Boxplot pentru caracteristicile numerice")
plt.show()
```

Creează un boxplot care arată distribuția valorilor pentru fiecare caracteristică numerică. Identifică valori anormale și varianța caracteristicilor.

### 3.2.1 Output Solution2.py



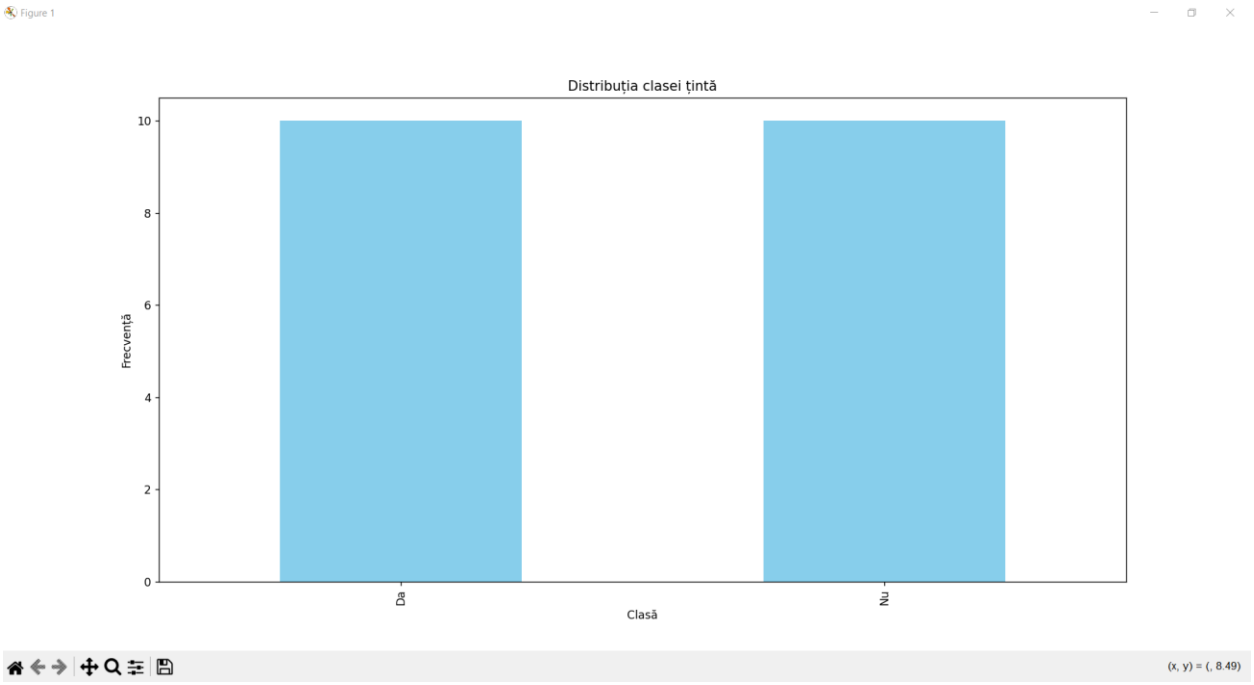
Imaginea reprezintă un arbore decizional generat cu DecisionTreeClassifier, bazat pe setul de date specificat. Acesta este un exemplu de clasificare folosind criteriul entropiei.

## Structura arborelui decizional

1. **Nodul rădăcină (nivelul 0)**
   - **Atribut:** Educație_Liceu( ≤0.5)
     - **Entropy:** 1.0 (maxim impur)
     - **Samples:** 20 (toate datele din set)
     - **Value:** [10, 10] (10 „Da” și 10 „Nu”)
     - **Clasă:** „Da” (decizia majoritară)
   - **Interpretare:** Prima împărțire este realizată pe baza caracteristicii Educație_Liceu. Aceasta determină două subseturi.
2. **Subarborele stâng (nivelul 1, Educație_Liceu <= 0.5)**
   - **Atribut:** Ore_suplimentare_Nu ≤0.5)
     - **Entropy:** 0.863 (mai puțin impur decât rădăcina)
     - **Samples:** 14
     - **Value:** [10, 4]
     - **Clasă:** „Da”
   - **Interpretare:** În subsetul stâng, se analizează dacă persoana nu lucrează ore suplimentare.
3. **Subarborele drept (nivelul 1, Educație_Liceu > 0.5)**
   - **Entropy:** 0.00 (impuritate minimă)
   - **Samples:** 6
   - **Value:** [0, 6]
   - **Clasă:** „Nu”
   - **Interpretare:** Persoanele care nu au terminat liceul (sau echivalent numeric) aparțin clasei „Nu”.
4. **Ramura stângă a Ore_suplimentare_Nu <= 0.5 (nivelul 2)**
   - **Entropy:** 0.00 (impuritate minimă)
   - **Samples:** 7
   - **Value:** [7, 0]

- **Clasă:** „Da”
- **Interpretare:** Persoanele care au lucrat ore suplimentare sunt promovate („Da”).

5. **Ramura dreaptă a Ore_suplimentare_Nu <= 0.5 (nivelul 2)**
   - **Atribut:** Educație_Master (≤0.5)
     - **Entropy:** 0.985
     - **Samples:** 7
     - **Value:** [3, 4]
     - **Clasă:** „Nu”
   - **Interpretare:** Se verifică dacă persoana are sau nu nivelul de educație „Master”.
6. **Ramura stângă a Educație_Master <= 0.5 (nivelul 3)**
   - **Atribut:** Educație_Licență (≤0.5)
     - **Entropy:** 0.811
     - **Samples:** 4
     - **Value:** [3, 1]
     - **Clasă:** „Da”
7. **Frunzele (nivelul 4)**
   - Pentru Educație_Licență <= 0.5: [2,0][2, 0][2,0], clasa „Da”.
   - Pentru Educație_Licență > 0.5: [1,1][1, 1][1,1], decizia este ambiguă.

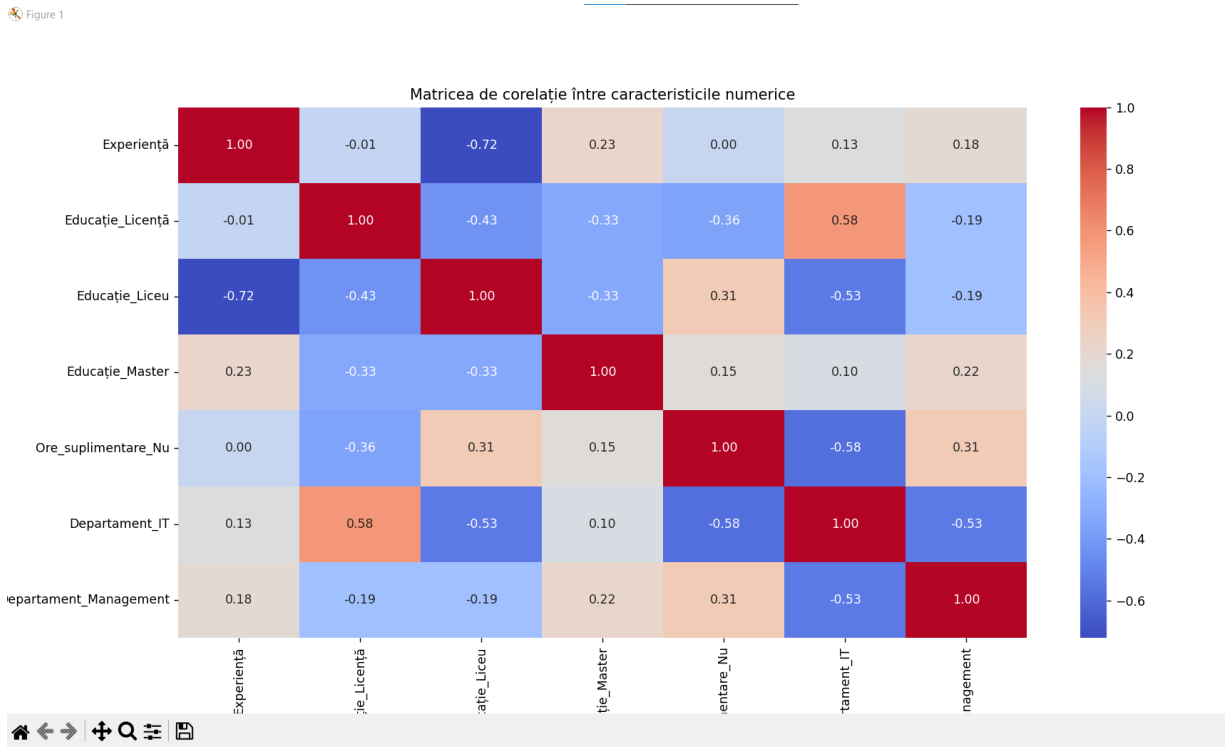



Graficul prezintă distribuția clasei țintă „Promovat” în setul de date. Aceasta este o vizualizare de bază pentru a înțelege proporțiile claselor „Da” și „Nu”.

## Analiza graficului

1. **Axe:**
   - **X:** Reprezintă cele două valori posibile ale clasei țintă: „Da” și „Nu”.
   - **Y:** Reprezintă frecvența fiecărei valori.
2. **Observații:**
   - Clasele „Da” și „Nu” au exact aceeași frecvență (10 instanțe fiecare).
   - Această distribuție este perfect echilibrată, ceea ce sugerează că modelul de clasificare nu va fi părtinitor față de o anumită clasă.
3. **Semnificație:**
   - Echilibrul între clase este ideal pentru algoritmi de învățare automată, cum ar fi arborii de decizie, deoarece reduce riscul de a favoriza o clasă în detrimentul celeilalte.
   - Acest tip de analiză este important înainte de antrenarea modelului pentru a evalua posibile dezechilibre.

Un astfel de grafic ajută la diagnosticarea potențialelor probleme legate de clasele dezechilibrate, care ar putea afecta performanța algoritmilor. În acest caz, putem spune că setul de date este bine echilibrat din perspectiva clasei țintă.




Imaginea prezintă o **matrice de corelație** între caracteristicile numerice din setul de date. Aceasta este reprezentată sub forma unei hărți de căldură (heatmap), în care valorile sunt colorate în funcție de intensitatea relației dintre perechi de caracteristici.

## Elemente-cheie ale graficului

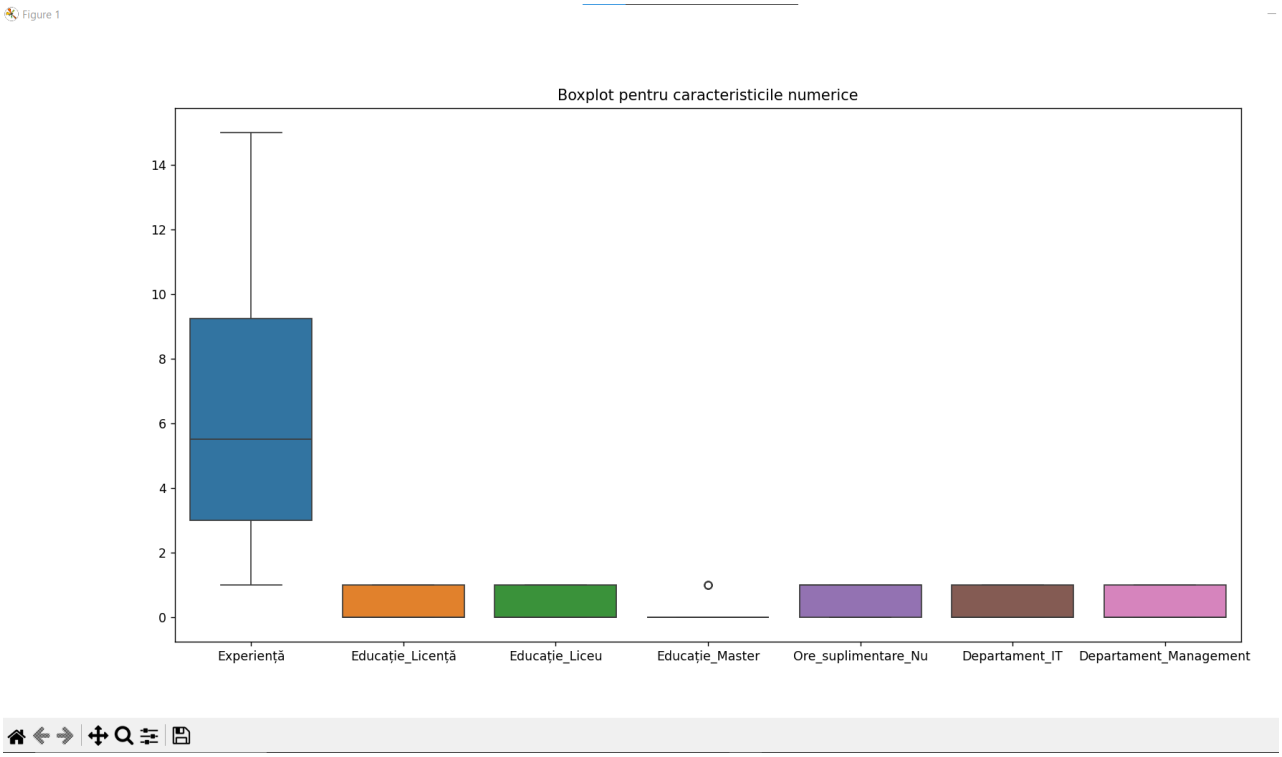
1. **Axe:**
   - Pe ambele axe se află caracteristicile numerice din setul de date (e.g., Experiență, Educație_Licență, Ore_suplimentare_Nu, etc.).
   - Matricea este simetrică față de diagonala principală.
2. **Diagonala principală:**
   - Valoarea este întotdeauna 1.0 pentru o caracteristică corelată cu ea însăși.
3. **Valori de corelație:**

- **Pozitive (> 0):** O relație direct proporțională între două caracteristici (pe măsură ce una crește, cealaltă crește).
- **Negative (< 0):** O relație invers proporțională între două caracteristici (pe măsură ce una crește, cealaltă scade).
- **Aproape de 0:** Lipsa unei relații semnificative.

4. **Culorile:**
   - **Roșu intens:** Corelație pozitivă puternică (e.g., valoarea 1.0 pe diagonală).
   - **Albastru intens:** Corelație negativă puternică.
   - **Alb sau culori pale:** Corelație slabă sau inexistentă.

**Interpretări din matrice:**

1. **Relații semnificative:**
   - Educație_Licență și Departament_IT au o corelație pozitivă moderată (0.58).
   - Educație_Liceu și Experiență au o corelație negativă puternică (−0.72), sugerând că nivelul de educație și experiența ar putea fi invers proporționale.
2. **Relații slabe:**
   - Ore_suplimentare_Nu și Experiență au o corelație aproape inexistentă (0.00).
   - Educație_Master și Educație_Licență au o corelație slab negativă (−0.33).
3. **Interpretare generală:**
   - Caracteristici precum educația și departamentul par să aibă o legătură moderată.
   - Experiența are relații variate cu alte caracteristici, iar impactul său este mai puternic asupra educației.




Imaginea prezintă un **boxplot** care ilustrează distribuția valorilor pentru caracteristicile numerice din setul de date. Boxplot-ul oferă o perspectivă asupra măsurilor descriptive (mediană, interval intercuartil, valori extreme) pentru fiecare caracteristică.

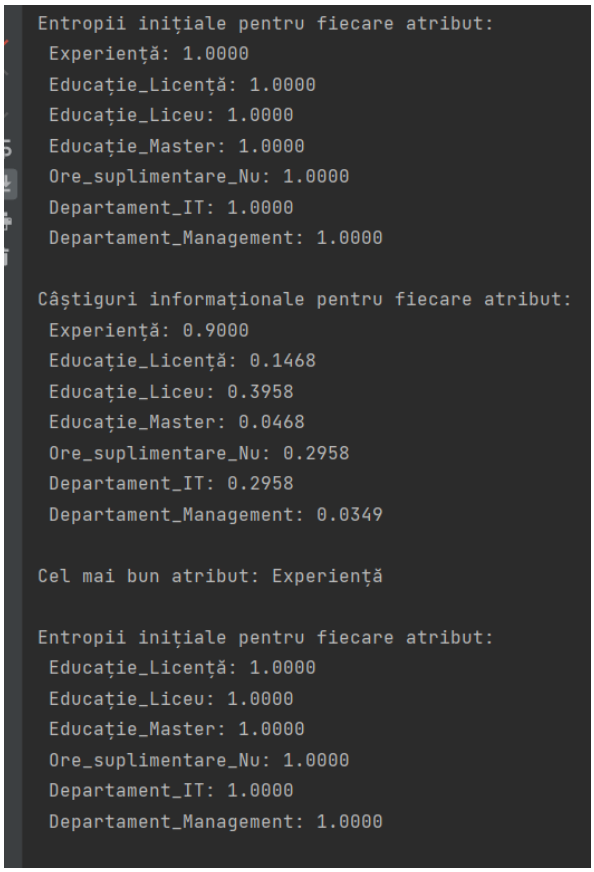
## Elemente-cheie ale boxplot-ului

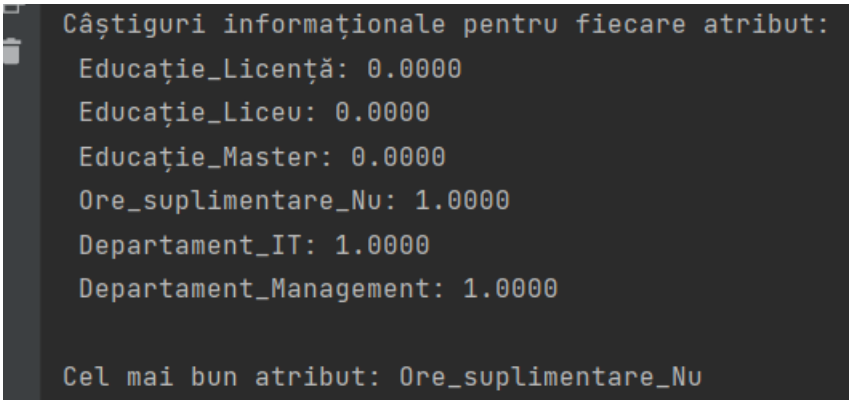
1. **Caracteristici numerice:**
   - Fiecare axă orizontală reprezintă o caracteristică numerică din setul de date (e.g., Experiență, Educație_Licență, etc.).
   - Valorile sunt standardizate numeric pentru reprezentare (de exemplu, valorile categorice convertite cu get_dummies).
2. **Componentele boxplot-ului:**
   - **Cutia:** Reprezintă intervalul intercuartil (IQR) – de la primul cuartil (Q1) la al treilea cuartil (Q3).
   - **Linia din mijlocul cutiei:** Reprezintă mediana.
   - **„Mustățile" (whiskers):** Extinderea până la valorile minime și maxime care nu sunt considerate outliers.
   - **Puncte individuale:** Reprezintă outliers (valori care se află în afara 1.5×IQR)

## Observații pentru fiecare caracteristică

1. **Experiență:**
    - Valorile sunt distribuite între aproximativ 11 și 15, cu o mediană în jurul valorii 7−8.
    - Nu sunt prezente outliers pentru această caracteristică.
    - Este singura caracteristică cu o gamă largă de valori.
2. **Educație_Licență, Educație_Liceu, Educație_Master:**
    - Aceste caracteristici au valori foarte restrânse (în principal 0 și 1, specifice codificării one-hot).
    - Educație_Master are un outlier la valoarea 11.
3. **Ore_suplimentare_Nu:**
    - Distribuția este binară (0 sau 1), reflectând prezența sau absența orelor suplimentare.
4. **Departament_IT, Departament_Management:**
    - De asemenea, distribuite în valori binare (0 și 1).

- Caracteristica Experiență este cea mai variabilă și oferă cele mai multe informații despre distribuția populației.
- Majoritatea caracteristicilor sunt binare, deci nu există o variație mare în distribuțiile lor.
- Outlier-ul prezent la Educație_Master sugerează că o singură persoană din setul de date are această caracteristică activată, iar acest lucru ar putea necesita verificări suplimentare.




## 1. Entropiile inițiale pentru fiecare atribut

Această imagine conține loguri generate în timpul construirii arborelui de decizie folosind algoritmul ID3. Logurile includ informații despre **entropiile inițiale**, **câștigurile informaționale** și **atributul selectat** pentru împărțire la fiecare pas.

## Entropiile inițiale pentru fiecare atribut

- **Definiție:** Entropia inițială măsoară impuritatea datelor înainte de împărțirea pe baza unui atribut. O entropie de 1.0 indică un set complet impur, în care toate clasele sunt distribuite uniform.
- **Observații:**
  - Toate atributele au entropia 1.0, ceea ce sugerează că fiecare dintre ele are o distribuție egală între clasele țintă la început.
  - Aceasta este o situație obișnuită pentru datele complet distribuite uniform.

## 2. Câștiguri informaționale pentru fiecare atribut

- Experiență: Câștig informațional 0.9000– cel mai mare câștig informațional, ceea ce indică faptul că acest atribut separă cel mai bine datele.
- Educație_Liceu: Câștig informațional 0.3958– al doilea cel mai mare câștig, sugerând că educația la nivel de liceu este, de asemenea, un atribut relevant.
- Atribute precum Educație_Master (0.0468) sau Departament_Management (0.0349) au câștig informațional mic, deci sunt mai puțin utile pentru împărțire.

## 3. Cel mai bun atribut

- **Selecția atributului:** Algoritmul selectează atributul cu cel mai mare câștig informațional pentru a fi folosit la împărțirea datelor în acest pas.
- **În acest caz:** Experiență este selectat deoarece are cel mai mare câștig informațional (0.9000).
- **De ce este selectat?** Conform teoriei ID3, cel mai informativ atribut este cel care reduce cel mai mult entropia datelor.

## 4. Iterația următoare

După selectarea atributului Experiență, se continuă procesul de împărțire pentru subseturile rămase, iar entropiile și câștigurile informaționale sunt recalculate pentru restul atributelor. Acest proces este repetat până când toate nodurile devin frunze (datele sunt complet împărțite).

## Concluzii:

- **Eficiența ID3:** Algoritmul ID3 folosește câștigul informațional pentru a selecta cel mai bun atribut, ceea ce duce la o împărțire optimă a datelor.
- **Semnificația entropiei:** Entropiile mari la început indică o clasă țintă distribuită uniform, ceea ce necesită mai multe împărțiri pentru a ajunge la un set de date „pur” (unde toate instanțele dintr-un subset aparțin aceleași clase).

```
Numărul total de noduri: 9
Numărul de frunze: 5
Adâncimea maximă a arborelui: 4
```

### 1. Numărul total de noduri: 9

- **Ce reprezintă:** Nodurile sunt elementele arborelui care conțin fie decizii (noduri interne), fie rezultate (noduri frunză).
- **Semnificație:**
  - Acest arbore are 9 noduri în total, ceea ce sugerează că datele au fost împărțite în mod rezonabil.
  - Un număr mai mare de noduri ar putea indica un arbore mai complex (și posibil suprainstruit).

### 2. Numărul de frunze: 5

- **Ce reprezintă:** Nodurile frunză sunt nodurile terminale ale arborelui, unde o decizie finală este atribuită unei clase.
- **Semnificație:**
  - Cele 5 frunze indică faptul că arborele clasifică datele în 5 subseturi distincte, fiecare corespondent unei clase finale.
  - Fiecare frunză conține date complet „pure” (toate datele din frunză aparțin aceleiași clase) sau aproape pure, în funcție de criteriul de oprire.

### 3. Adâncimea maximă a arborelui: 4

- **Ce reprezintă:** Adâncimea maximă este numărul maxim de niveluri din arbore, pornind de la rădăcină până la cea mai îndepărtată frunză.
- **Semnificație:**
  - Adâncimea de 4 indică faptul că unele decizii necesită până la 4 pași de verificare a atributelor pentru a ajunge la o concluzie.
  - O adâncime mai mare poate semnala o complexitate crescută, iar una mai mică arată că deciziile sunt luate mai rapid.
  - Este important ca adâncimea să nu fie prea mare pentru a evita suprainstruirea.

## Concluzii:

**Echilibru între complexitate și generalizare:**

- Arborele este moderat ca dimensiune (9 noduri și 5 frunze), ceea ce sugerează că modelul este probabil bine calibrat pentru setul de date.
- Adâncimea de 4 este rezonabilă și indică un echilibru între capacitatea de decizie și complexitatea modelului.

```
Reguli IF-THEN ale arborelui decizional:
IF Experiență = 1 AND  THEN Nu
IF Experiență = 2 AND  THEN Nu
IF Experiență = 3 AND  THEN Nu
IF Experiență = 4 AND  THEN Da
IF Experiență = 5 AND  THEN Da
IF Experiență = 6 AND  THEN Nu
IF Experiență = 7 AND Ore_suplimentare_Nu = False AND  THEN Da
IF Experiență = 7 AND Ore_suplimentare_Nu = True AND  THEN Nu
IF Experiență = 8 AND  THEN Da
IF Experiență = 9 AND  THEN Da
IF Experiență = 10 AND  THEN Da
IF Experiență = 11 AND  THEN Nu
IF Experiență = 12 AND  THEN Nu
IF Experiență = 15 AND  THEN Da
```

Această imagine conține **regulile IF-THEN** generate din arborele decizional construit. Aceste reguli descriu modul în care arborele ajunge la o decizie finală pe baza valorilor atributelor.

### Structura regulilor IF-THEN

1. **IF:** Specifică condiția care trebuie să fie îndeplinită pentru a ajunge la o anumită decizie.
2. **AND:** Unește mai multe condiții atunci când decizia depinde de mai multe atribute.
3. **THEN:** Reprezintă decizia finală a arborelui pentru condițiile date.

### Interpretarea regulilor

1. **Simplitatea regulilor:**
    - Reguli precum: IF Experiență = 1 AND THEN Nu

indică faptul că decizia pentru persoanele cu experiență de 1 an este direct „Nu”. Nu există alte condiții suplimentare de verificat.

2. **Reguli mai complexe:**
    - Observăm o regulă mai complex:

IF Experiență = 7 AND Ore_suplimentare_Nu = False AND THEN Da

IF Experiență = 7 AND Ore_suplimentare_Nu = True AND THEN Nu

Aceasta arată că pentru persoanele cu 7 ani de experiență, decizia depinde și de valoarea atributului Ore_suplimentare_Nu:

- Dacă persoana nu lucrează ore suplimentare (True), decizia este „Nu”.
- Dacă persoana lucrează ore suplimentare (False), decizia este „Da”.

**Deciziile în funcție de experiență:**

- Se observă o separare clară bazată pe experiență:
    - Persoanele cu experiență scăzută (1-3 ani) primesc „Nu”.
    - Persoanele cu experiență mare (8-15 ani) primesc în general „Da”.

- În intervalul mediu (4-7 ani), decizia poate varia în funcție de alți factori (e.g., Ore_suplimentare_Nu).

## Concluzii:

- **Relevanța atributelor:**
  - Experiență este cel mai semnificativ atribut, fiind folosit în toate regulile.
  - Ore_suplimentare_Nu apare doar în cazul experienței de 7 ani, sugerând că este relevant doar pentru un subset specific al datelor.
- **Decizii clare:**
  - Majoritatea regulilor conduc la decizii directe („Da” sau „Nu”), ceea ce indică un arbore bine optimizat.
- **Reguli redundante:**
  - Unele reguli par redundante, deoarece există multe valori de experiență care duc direct la „Da” sau „Nu” fără alte condiții. Acest lucru sugerează că arborele ar putea fi simplificat pentru o interpretare mai clară.

## 3.3 Fișierul solution3.py

```
set_date = {
    "Experientă" : [5, 2, 7, 10, 1, 4, 6, 3],  # Ani de experientă
    "Educatie" : ["Licență", "Liceu", "Master", "Doctorat", "Liceu", "Licență", "Master", "Liceu"],
    "Ore_suplimentare" : ["Da", "Nu", "Da", "Nu", "Da", "Da", "Nu", "Nu"],
    "Vârstă" : [29, 22, 35, 45, 19, 30, 40, 25],
    "Departament" : ["IT", "HR", "IT", "Management", "HR", "IT", "Management", "HR"],
    "Promovat" : ["Da", "Nu", "Da", "Da", "Nu", "Da", "Nu", "Nu"]
}
```

- Este definit un set de date fictiv, cu variabile categorice (Educație, Ore_suplimentare, Departament) și numerice (Experiență, Vârstă).
- Acest set este transformat într-un DataFrame Pandas pentru procesare.

```
# Convertim valorile categorice în valori numerice
set_date = pd.DataFrame(set_date)
set_date = pd.get_dummies(set_date, columns=["Educatie", "Ore_suplimentare", "Departament"], drop_first=True)

# Separăm caracteristicile de țintă
X = set_date.drop("Promovat", axis=1)
y = set_date["Promovat"]
```

- Variabilele categorice sunt convertite în numerice folosind pd.get_dummies.
- X conține caracteristicile predictive, iar y conține clasa țintă.

```
# Împărtim datele în seturi de antrenare și test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

- Datele sunt împărțite în 70% pentru antrenare și 30% pentru testare folosind train_test_split.

```
# Creăm și antrenăm arborele decizional
model = DecisionTreeClassifier(criterion="entropy", random_state=42, max_depth=4)
model.fit(X_train, y_train)
```

- Un arbore decizional este creat folosind criteriul de entropie (entropy) pentru măsurarea impurității.
- Parametrul max_depth limitează adâncimea maximă la 4 niveluri pentru a preveni suprainstruirea.

```
# Vizualizăm arborele decizional
def vizualizeaza_arbore(model, feature_names) :
    plt.figure(figsize=(9, 8))
    plot_tree(model, feature_names=feature_names, class_names=model.classes_, filled=True)
    plt.title("Arbore Decizional - Vizualizare Grafică")
    plt.show()

vizualizeaza_arbore(model, X.columns)
```

Arborele este vizualizat folosind plot_tree, care afișează structura nodurilor și a frunzelor.

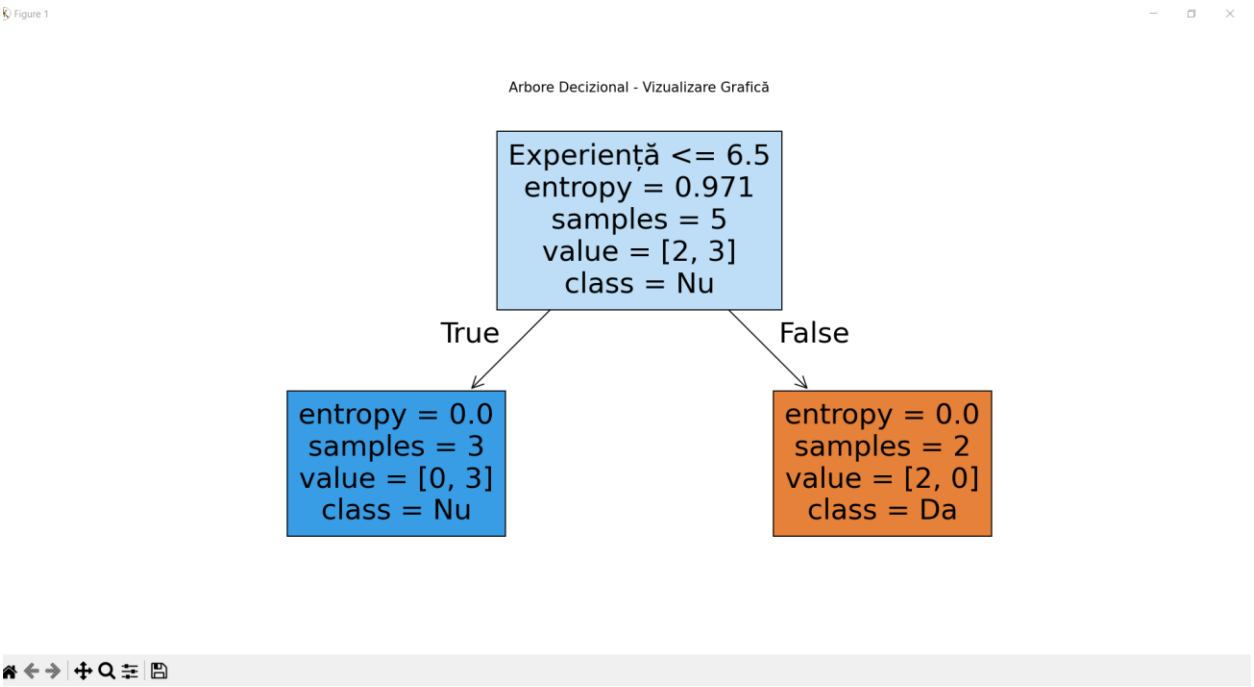
```python
# Evaluarea modelului pe setul de testare
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"\nAcuratetea modelului pe setul de testare: {accuracy:.2f}")

print("\nRaportul de clasificare:")
print(classification_report(y_test, y_pred))

# Matricea de confuzie
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=model.classes_, yticklabels=model.classes_)
plt.title("Matricea de confuzie")
plt.xlabel("Clase prezise")
plt.ylabel("Clase reale")
plt.show()
```

- Se calculează acuratețea modelului folosind accuracy_score.
- Se generează un raport de clasificare (classification_report) și o matrice de confuzie (confusion_matrix).

### 3.3.1 Output Solution3.py




Graficul afișat este o reprezentare vizuală a unui arbore decizional, construit folosind setul de date și criteriul entropiei.

## Structura arborelui:

1. **Nodul rădăcină:**
   - **Decizie:** Experiență <= 6.5
   - **Entropie:** 0.971 - o valoare destul de ridicată, indicând impuritate mare (amestec de clase).
   - **Samples:** 5 - numărul total de exemple în acest subset.
   - **Value:** [2,3][2, 3][2,3] - din cele 5 exemple, 2 sunt clasificate ca „Nu” și 3 ca „Da”.
   - **Clasă:** „Nu” - clasa majoritară.
2. **Ramura stângă (True - Experiență <= 6.5):**
   - **Entropie:** 0.0 - impuritate zero, ceea ce înseamnă că toate exemplele din acest subset aparțin aceleași clase.
   - **Samples:** 3 - numărul de exemple din subset.
   - **Value:** [0,3][0, 3][0,3] - toate cele 3 exemple sunt clasificate ca „Nu”.
   - **Clasă:** „Nu”.
3. **Ramura dreaptă (False - Experiență > 6.5):**
   - **Entropie:** 0.0 - impuritate zero, indicând un subset „pur”.
   - **Samples:** 2 - numărul de exemple din acest subset.
   - **Value:** [2,0][2, 0][2,0] - toate cele 2 exemple sunt clasificate ca „Da”.
   - **Clasă:** „Da”.

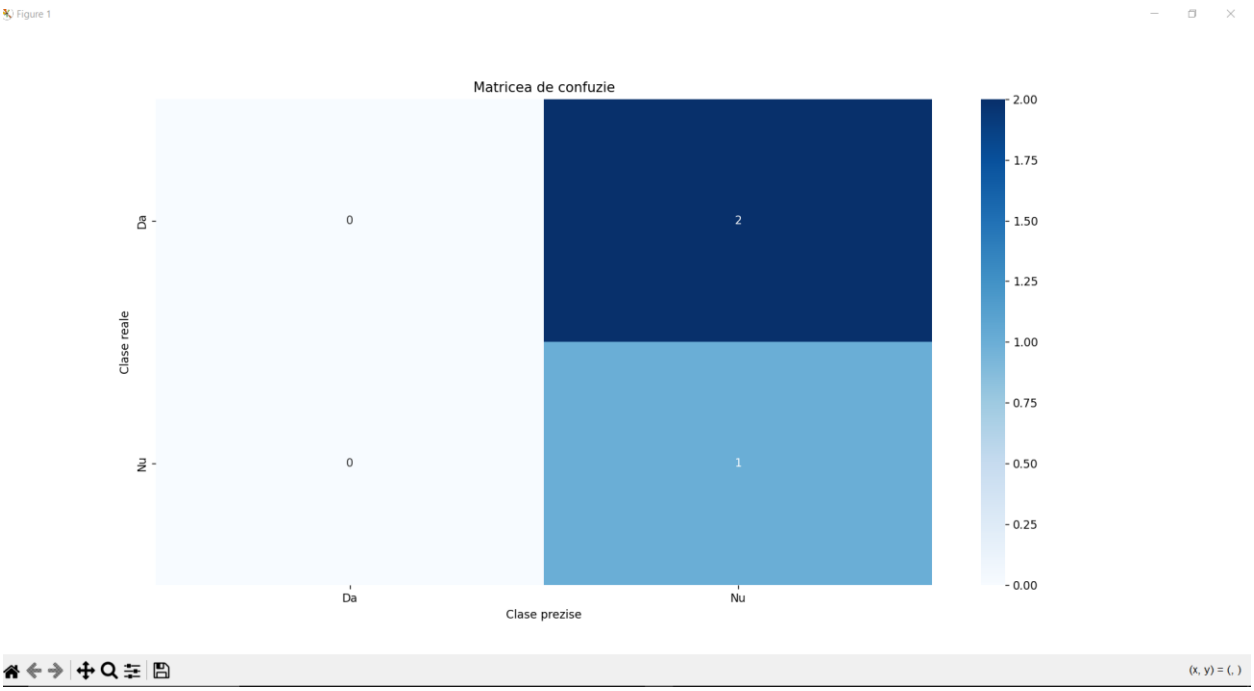
## Decizii luate de arbore:

- Dacă **Experiența** este mai mică sau egală cu 6.5 ani, persoana este clasificată ca „Nu”.
- Dacă **Experiența** este mai mare de 6.5 ani, persoana este clasificată ca „Da”.

## Analiză generală:

1. **Entropia:**
   - La rădăcina arborelui, entropia este ridicată, indicând o împărțire inițială incertă între clase.
   - După împărțire, entropia scade la 0.0 în ambele ramuri, ceea ce arată că setul de date a fost separat complet în clase pure.

2. **Complexitatea arborelui:**
    - Arborele are doar 2 niveluri (foarte simplu), deoarece setul de date are puține exemple și caracteristici bine separate.
3. **Relevanța atributelor:**
    - Experiență este cel mai informativ atribut (cel mai mare câștig informațional), ceea ce îl face să fie ales pentru împărțirea la rădăcină.




Graficul reprezintă **matricea de confuzie** generată pe baza predicțiilor modelului de arbore decizional aplicat pe setul de testare.

### Ce reprezintă matricea de confuzie?

Matricea de confuzie arată numărul de instanțe corect și incorect clasificate de model pentru fiecare clasă:

1. **Axe:**
    - **Axă verticală (Clase reale):** Valorile reale ale clasei țintă.
    - **Axă orizontală (Clase prezise):** Valorile prezise de model.
2. **Celulele:**

- Valoarea din fiecare celulă indică numărul de instanțe din setul de test care aparțin combinației respective (reale/presise).

Graficul reprezintă **matricea de confuzie** generată pe baza predicțiilor modelului de arbore decizional aplicat pe setul de testare.

**Interpretarea celulelor din grafic:**

1. **Diagonala principală (celule colorate mai intens):**
   - **(0,0):** Nicio instanță reală „Da” nu a fost prezisă corect ca „Da”.
   - **(1,1):** O instanță reală „Nu” a fost prezisă corect ca „Nu”.
2. **Celulele non-diagonale (erori):**
   - **(0,1):** 2 instanțe reale „Da” au fost clasificate greșit ca „Nu”.
   - **(1,0):** Nicio instanță reală „Nu” nu a fost clasificată greșit ca „Da”.

**Analiză generală:**

1. **Performanța modelului:**
   - Modelul are probleme în a clasifica corect instanțele din clasa „Da”, având 2 greșeli.
   - Instanțele din clasa „Nu” sunt clasificate fără erori.
2. **Acuratețea:**
   - Acuratețea este calculată ca:

   Acuratețea = (Numărul de instanțe corect clasificate) / (Numărul total de instanțe)

3. **Implicarea dezechilibrului claselor:**
   - Dezechilibrul între clasele „Da” și „Nu” poate afecta performanța. Modelul poate fi influențat de clasele dominante.

```
              precision    recall  f1-score   support

          Da       0.00      0.00      0.00         2
          Nu       0.33      1.00      0.50         1

    accuracy                           0.33         3
   macro avg       0.17      0.50      0.25         3
weighted avg       0.11      0.33      0.17         3


Process finished with exit code 0
```

Raportul de clasificare afișat analizează performanța modelului pe setul de testare folosind trei metrici principale: **precizia** (precision), **acoperirea** (recall) și **scorul F1** (f1-score).

### Secțiuni cheie ale raportului

1. **Clasa „Da”**:
   - **Precision (Precizie):** 0.00 - Modelul nu a prezis corect niciun exemplu din clasa „Da”.
   - **Recall (Acoperire):** 0.00 - Modelul nu a identificat corect niciun exemplu real din clasa „Da”.
   - **F1-score:** 0.00 - Media armonică a preciziei și acoperirii este zero, deoarece niciuna dintre acestea nu are valori pozitive.
   - **Support:** 2 - Numărul de instanțe reale din clasa „Da” este 2.
2. **Clasa „Nu”**:
   - **Precision (Precizie):** 0.33 - Din totalul predicțiilor „Nu”, doar 33% sunt corecte.
   - **Recall (Acoperire):** 1.00 - Modelul a identificat corect toate instanțele reale din clasa „Nu”.
   - **F1-score:** 0.50 - Media armonică a preciziei și acoperirii este moderată.
   - **Support:** 1 - Numărul de instanțe reale din clasa „Nu” este 1.
3. **Accuracy (Acuratețea):**
   - 0.33 - Acuratețea globală a modelului este de 33%, adică 1 din cele 3 instanțe de test au fost clasificate corect.
4. **Macro avg (Media macro):**
   - **Precizie:** 0.17 - Media aritmetică a preciziei pentru ambele clase.
   - **Acoperire:** 0.50 - Media aritmetică a acoperirii pentru ambele clase.
   - **F1-score:** 0.25 - Media aritmetică a scorului F1 pentru ambele clase.
5. **Weighted avg (Media ponderată):**
   - **Precizie:** 0.11 - Precizia medie ponderată în funcție de suportul fiecărei clase.
   - **Acoperire:** 0.33 - Acoperirea medie ponderată.
   - **F1-score:** 0.17 - Scorul F1 mediu ponderat.

## 4. Cod sursa

**Solution.py:**

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
from sklearn.tree import DecisionTreeClassifier, export_text, plot_tree


def main() :
    set_date = {
        "Experiență" : [5, 2, 7, 10, 1, 4, 6, 3],  # Ani de experiență
        "Educație" : ["Licență", "Liceu", "Master", "Doctorat", "Liceu", "Licență", "Master",
"Liceu"],
        "Ore_suplimentare" : ["Da", "Nu", "Da", "Nu", "Da", "Da", "Nu", "Nu"],
        "Vârstă" : [29, 22, 35, 45, 19, 30, 40, 25],
        "Departament" : ["IT", "HR", "IT", "Management", "HR", "IT", "Management", "HR"],
        "Promovat" : ["Da", "Nu", "Da", "Da", "Nu", "Da", "Nu", "Nu"]
    }

    # Convertim valorile categorice în valori numerice
    set_date = pd.DataFrame(set_date)
    set_date = pd.get_dummies(set_date, columns=["Educație", "Ore_suplimentare",
"Departament"], drop_first=True)

    # Separăm caracteristicile de țintă
    X = set_date.drop("Promovat", axis=1)
    y = set_date["Promovat"]

    # Creăm și antrenăm arborele decizional
```

```
model = DecisionTreeClassifier(criterion="entropy", random_state=42, max_depth=4)
model.fit(X, y)

# Funcție pentru calcularea entropiei
def calculeaza_entropia(y) :
    valori_unice, frecvente = np.unique(y, return_counts=True)
    probabilitati = frecvente / len(y)
    entropia = -np.sum(probabilitati * np.log2(probabilitati))
    return entropia

# Exemplu entropie
entropie_initiala = calculeaza_entropia(y)
print(f"\nEntropia setului inițial: {entropie_initiala}")

# Vizualizăm arborele decizional
def vizualizeaza_arbore(model, feature_names) :
    plt.figure(figsize=(9, 8))
    plot_tree(model, feature_names=feature_names, class_names=model.classes_, filled=True)
    plt.show()

vizualizeaza_arbore(model, X.columns)

# Exportăm regulile arborelui
tree_rules = export_text(model, feature_names=list(X.columns))
print("\nReguli ale arborelui decizional:")
print(tree_rules)

# Calculează câștigul informațional
def castig_informational(y, feature) :
    entropie_initiala = calculeaza_entropia(y)
    valori_unice, frecvente = np.unique(feature, return_counts=True)
```

```
    entropie_conditionala = 0

    for valoare, frecventa in zip(valori_unice, frecvente) :
        subset = y[feature == valoare]
        entropie_conditionala += (frecventa / len(feature)) * calculeaza_entropia(subset)

    return entropie_initiala - entropie_conditionala

# Exemplu calcul câștig informațional
castiguri = {col : castig_informational(y, X[col]) for col in X.columns}
print("\nCâștiguri informaționale pentru fiecare caracteristică:")
print(castiguri)

# Histogramă pentru fiecare caracteristică numerică
X.hist(bins=10, figsize=(10, 8), color='skyblue', edgecolor='black')
plt.suptitle("Histogramă pentru fiecare caracteristică numerică", fontsize=12)
plt.show()

# Pairplot pentru relațiile dintre caracteristici numerice
sns.pairplot(set_date, diag_kind="kde", corner=True)
plt.suptitle("Relații între caracteristici (Pairplot)", fontsize=12)
plt.show()


if __name__ == "__main__" :
    main()
```

**Solution2.py**

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
```

```
import seaborn as sns
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import plot_tree



class ID3Tree :
    def __init__(self) :
        self.tree = None

    def calculeaza_entropia(self, y) :
        valori_unice, frecvente = np.unique(y, return_counts=True)
        probabilitati = frecvente / len(y)
        entropia = -np.sum(probabilitati * np.log2(probabilitati))
        return entropia

    def castig_informational(self, y, feature) :
        entropie_initiala = self.calculeaza_entropia(y)
        valori_unice, frecvente = np.unique(feature, return_counts=True)
        entropie_conditionala = 0

        for valoare, frecventa in zip(valori_unice, frecvente) :
            subset = y[feature == valoare]
            entropie_conditionala += (frecventa / len(feature)) * self.calculeaza_entropia(subset)

        gain = entropie_initiala - entropie_conditionala
        return entropie_initiala, gain

    def alege_cel_mai_bun_atribut(self, X, y) :
        castiguri = {}
        entropii_initiale = {}
        for col in X.columns :
```

```
            entropii_initiale[col], castiguri[col] = self.castig_informational(y, X[col])

        cel_mai_bun = max(castiguri, key=castiguri.get)
        print("\nEntropii inițiale pentru fiecare atribut:")
        for atribut, entropie in entropii_initiale.items() :
            print(f" {atribut}: {entropie:.4f}")

        print("\nCâștiguri informaționale pentru fiecare atribut:")
        for atribut, castig in castiguri.items() :
            print(f" {atribut}: {castig:.4f}")

        print(f"\nCel mai bun atribut: {cel_mai_bun}")
        return cel_mai_bun

    def construieste_arbore(self, X, y) :
        # Dacă toate valorile țintă sunt identice, returnează acea valoare
        if len(np.unique(y)) == 1 :
            return y.iloc[0]

        # Dacă nu mai sunt atribute disponibile, returnează clasa majoritară
        if X.empty :
            return y.mode()[0]

        # Alege cel mai bun atribut pentru splitting
        cel_mai_bun_atribut = self.alege_cel_mai_bun_atribut(X, y)
        arbore = {cel_mai_bun_atribut : {}}

        # Creează ramuri pentru fiecare valoare a atributului
        for valoare in np.unique(X[cel_mai_bun_atribut]) :
            subset_X = X[X[cel_mai_bun_atribut] ==
valoare].drop(columns=[cel_mai_bun_atribut])
```

```
        subset_y = y[X[cel_mai_bun_atribut] == valoare]

        # Construiește recursiv arborele pentru fiecare subset
        arbore[cel_mai_bun_atribut][valoare] = self.construieste_arbore(subset_X, subset_y)

    return arbore

def fit(self, X, y) :
    self.tree = self.construieste_arbore(X, y)

def predict_sample(self, sample, tree) :
    if not isinstance(tree, dict) :
        return tree

    atribut = next(iter(tree))
    valoare = sample[atribut]

    if valoare in tree[atribut] :
        return self.predict_sample(sample, tree[atribut][valoare])
    else :
        return None  # Valoarea nu există în arbore

def predict(self, X) :
    return X.apply(lambda sample : self.predict_sample(sample, self.tree), axis=1)

def print_tree(self, tree=None, indent="", depth=0) :
    if tree is None :
        tree = self.tree

    if not isinstance(tree, dict) :
        print("  " * depth + f"[Leaf] {tree}")
```

```
            return

        for key, value in tree.items() :
            print("  " * depth + f"{key}:")
            for subkey, subtree in value.items() :
                print("  " * (depth + 1) + f"{subkey} ->")
                self.print_tree(subtree, indent, depth + 2)

    def export_if_then(self, tree=None, indent="IF ") :
        if tree is None :
            tree = self.tree

        if not isinstance(tree, dict) :
            print(indent + f" THEN {tree}")
            return

        for key, value in tree.items() :
            for subkey, subtree in value.items() :
                self.export_if_then(subtree, indent + f"{key} = {subkey} AND ")



# Exemplu de utilizare
def main() :
    # Set de date extins pentru exemplificare
    set_date = {
        "Experiență" : [5, 2, 7, 10, 1, 4, 6, 3, 8, 9, 12, 15, 1, 2, 4, 5, 7, 3, 10, 11],
        "Educație" : ["Licență", "Liceu", "Master", "Doctorat", "Liceu", "Licență", "Master",
"Liceu", "Licență",
                      "Doctorat", "Master", "Doctorat", "Liceu", "Liceu", "Licență", "Licență", "Master",
"Liceu",
                      "Doctorat", "Licență"],
```

```
    "Ore_suplimentare" : ["Da", "Nu", "Da", "Nu", "Nu", "Da", "Nu", "Nu", "Da", "Da", "Nu",
"Nu", "Nu", "Da", "Nu",
                          "Da", "Nu", "Nu", "Da", "Nu"],
    "Departament" : ["IT", "HR", "IT", "Management", "HR", "IT", "Management", "HR",
"IT", "Management", "IT", "HR",
                     "Management", "HR", "IT", "IT", "Management", "HR", "IT", "Management"],
    "Promovat" : ["Da", "Nu", "Da", "Da", "Nu", "Da", "Nu", "Nu", "Da", "Da", "Nu", "Da",
"Nu", "Nu", "Da", "Da",
                  "Nu", "Nu", "Da", "Nu"]
  }

  df = pd.DataFrame(set_date)

  # Convertim valorile categorice în numerice (dacă este cazul)
  df = pd.get_dummies(df, columns=["Educație", "Ore_suplimentare", "Departament"],
drop_first=True)

  X = df.drop(columns=["Promovat"])
  y = df["Promovat"]

  # Convertim toate coloanele în tip numeric
  X = X.apply(pd.to_numeric)

  # Construim arborele decizional folosind DecisionTreeClassifier pentru grafic
  clf = DecisionTreeClassifier(criterion="entropy", max_depth=4, random_state=42)
  clf.fit(X, y)

  # Vizualizăm arborele decizional ca grafic
  plt.figure(figsize=(9, 6))
  plot_tree(clf, feature_names=X.columns, class_names=clf.classes_, filled=True,
rounded=True)
```

```
plt.title("Arborele Decizional - Vizualizare Grafică")
plt.show()

# Construim arborele decizional folosind ID3Tree
id3 = ID3Tree()
id3.fit(X, y)

# Afișăm arborele în format text
print(f"Numărul total de noduri: {clf.tree_.node_count}")
print(f"Numărul de frunze: {clf.get_n_leaves()}")
print(f"Adâncimea maximă a arborelui: {clf.get_depth()}")

# Exportăm regulile arborelui în format IF-THEN
print("\nReguli IF-THEN ale arborelui decizional:")
id3.export_if_then()

# 2. Distribuția clasei țintă
plt.figure(figsize=(6, 4))
y.value_counts().plot(kind="bar", color="skyblue")
plt.title("Distribuția clasei țintă")
plt.xlabel("Clasă")
plt.ylabel("Frecvență")
plt.show()

# 3. Matricea de corelație pentru date numerice
plt.figure(figsize=(10, 5))
sns.heatmap(X.corr(), annot=True, fmt=".2f", cmap="coolwarm")
plt.title("Matricea de corelație între caracteristicile numerice")
plt.show()

# 4. Boxplot pentru caracteristici numerice
```

```
    plt.figure(figsize=(8, 8))
    sns.boxplot(data=X)
    plt.title("Boxplot pentru caracteristicile numerice")
    plt.show()



if __name__ == "__main__" :
    main()
```

**Solution3.py**

```
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier, plot_tree


def main() :
    set_date = {
        "Experiență" : [5, 2, 7, 10, 1, 4, 6, 3],  # Ani de experiență
        "Educație" : ["Licență", "Liceu", "Master", "Doctorat", "Liceu", "Licență", "Master",
"Liceu"],
        "Ore_suplimentare" : ["Da", "Nu", "Da", "Nu", "Da", "Da", "Nu", "Nu"],
        "Vârstă" : [29, 22, 35, 45, 19, 30, 40, 25],
        "Departament" : ["IT", "HR", "IT", "Management", "HR", "IT", "Management", "HR"],
        "Promovat" : ["Da", "Nu", "Da", "Da", "Nu", "Da", "Nu", "Nu"]
    }

    # Convertim valorile categorice în valori numerice
    set_date = pd.DataFrame(set_date)
    set_date = pd.get_dummies(set_date, columns=["Educație", "Ore_suplimentare",
"Departament"], drop_first=True)

    # Separăm caracteristicile de țintă
    X = set_date.drop("Promovat", axis=1)
    y = set_date["Promovat"]

    # Împărțim datele în seturi de antrenare și test
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

```
    # Creăm și antrenăm arborele decizional
    model = DecisionTreeClassifier(criterion="entropy", random_state=42, max_depth=4)
    model.fit(X_train, y_train)

    # Vizualizăm arborele decizional
    def vizualizeaza_arbore(model, feature_names) :
        plt.figure(figsize=(9, 8))
        plot_tree(model, feature_names=feature_names, class_names=model.classes_, filled=True)
        plt.title("Arbore Decizional - Vizualizare Grafică")
        plt.show()

    vizualizeaza_arbore(model, X.columns)

    # Evaluarea modelului pe setul de testare
    y_pred = model.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    print(f"\nAcuratețea modelului pe setul de testare: {accuracy:.2f}")

    print("\nRaportul de clasificare:")
    print(classification_report(y_test, y_pred))

    # Matricea de confuzie
    cm = confusion_matrix(y_test, y_pred)
    sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=model.classes_,
yticklabels=model.classes_)
    plt.title("Matricea de confuzie")
    plt.xlabel("Clase prezise")
    plt.ylabel("Clase reale")
    plt.show()


if __name__ == "__main__" :
    main()
```

## 5. Librării utilizate

matplotlib.pyplot - Utilizată pentru a crea grafice și vizualizări.

seaborn - O librărie pentru vizualizări statistice mai avansate, construită peste matplotlib.

numpy - Manipularea matricelor și realizarea de calcule numerice eficiente.

pandas - Manipularea datelor în format tabelar (DataFrame) și conversia variabilelor categorice în numerice.

sklearn.tree - Construirea și vizualizarea unui arbore decizional.

sklearn.model_selection - Împărțirea datelor în seturi de antrenare și test.

sklearn.metrics - Măsurarea performanței modelului.

## 6. Resurse

https://www.w3schools.com/python/python_ml_decision_tree.asp

https://www.w3schools.com/python/python_ml_confusion_matrix.asp

https://www.w3schools.com/python/python_ml_train_test.asp