Tema 1 - Invatare Automata 2024

```
getDependencies = True #@param ["False", "True"] {type:"raw"}
```

getDependencies: True

```
if getDependencies:
    !pip install seaborn
    !pip install scipy
    !pip install scikit-learn
    !pip install category_encoders
    from google.colab import drive
    drive.mount('/content/drive')

# Import required libraries
import pandas as pd
import numpy as np
import seaborn as sns
sns.set()
import matplotlib.pyplot as plt
%matplotlib inline
import category_encoders as ce
import traceback
import time

from scipy.stats import pointbiserialr, chi2_contingency, f_oneway
from sklearn.feature_selection import VarianceThreshold, SelectPercentile
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import OneHotEncoder, LabelEncoder, OrdinalEncoder, StandardScaler, MinMaxScaler, RobustScaler, MaxAbsScaler, Powe
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import SimpleImputer, IterativeImputer
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix
```

```
Requirement already satisfied: seaborn in /usr/local/lib/python3.10/dist-packages (0.13.1)
Requirement already satisfied: numpy!=1.24.0,>=1.20 in /usr/local/lib/python3.10/dist-packages (from seaborn) (1.25.2)
Requirement already satisfied: pandas>=1.2 in /usr/local/lib/python3.10/dist-packages (from seaborn) (2.0.3)
Requirement already satisfied: matplotlib!=3.6.1,>=3.4 in /usr/local/lib/python3.10/dist-packages (from seaborn) (3.7.1)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (1.2.
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (4.5
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (1.4
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (24.0)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (9.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (3.1.
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.10/dist-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=1.2->seaborn) (2023.4)
Requirement already satisfied: tzdata>=2022.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=1.2->seaborn) (2024.1)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.7->matplotlib!=3.6.1,>=3.4->
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages (1.11.4)
Requirement already satisfied: numpy<1.28.0,>=1.21.6 in /usr/local/lib/python3.10/dist-packages (from scipy) (1.25.2)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.10/dist-packages (1.2.2)
Requirement already satisfied: numpy>=1.17.3 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (1.25.2)
Requirement already satisfied: scipy>=1.3.2 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (1.11.4)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (1.4.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (3.4.0)
Requirement already satisfied: category_encoders in /usr/local/lib/python3.10/dist-packages (2.6.3)
Requirement already satisfied: numpy>=1.14.0 in /usr/local/lib/python3.10/dist-packages (from category_encoders) (1.25.2)
Requirement already satisfied: scikit-learn>=0.20.0 in /usr/local/lib/python3.10/dist-packages (from category_encoders) (1.2.2)
Requirement already satisfied: scipy>=1.0.0 in /usr/local/lib/python3.10/dist-packages (from category_encoders) (1.11.4)
Requirement already satisfied: statsmodels>=0.9.0 in /usr/local/lib/python3.10/dist-packages (from category_encoders) (0.14.2)
Requirement already satisfied: pandas>=1.0.5 in /usr/local/lib/python3.10/dist-packages (from category_encoders) (2.0.3)
Requirement already satisfied: patsy>=0.5.1 in /usr/local/lib/python3.10/dist-packages (from category_encoders) (0.5.6)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.10/dist-packages (from pandas>=1.0.5->category_encoders)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=1.0.5->category_encoders) (2023.4)
Requirement already satisfied: tzdata>=2022.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=1.0.5->category_encoders) (2024.1
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages (from patsy>=0.5.1->category_encoders) (1.16.0)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=0.20.0->category_encoders) (
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=0.20.0->category_enco
Requirement already satisfied: packaging>=21.3 in /usr/local/lib/python3.10/dist-packages (from statsmodels>=0.9.0->category_encoders) (
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

Citim setul de date

```
DATASET_PATH = "/content/drive/MyDrive/Tema1_ML"
df = pd.read_csv(f"{DATASET_PATH}/date_tema_1_iaut_2024.csv")
```

3.1. Explorarea Datelor (Exploratory Data Analysis)

```
num_examples = df.shape[0]
print("Numărul de exemple din setul de date:", num_examples)
```

```
Numărul de exemple din setul de date: 1921
```

```
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1921 entries, 0 to 1920
Data columns (total 19 columns):
 #   Column                          Non-Null Count  Dtype 
---  ------                          --------------  ----- 
 0   Transportation                  1921 non-null   object
 1   Regular_fiber_diet              1921 non-null   object
 2   Diagnostic_in_family_history    1921 non-null   object
 3   High_calorie_diet               1921 non-null   object
 4   Sedentary_hours_daily           1921 non-null   object
 5   Age                             1921 non-null   object
 6   Alcohol                         1921 non-null   object
 7   Est_avg_calorie_intake          1921 non-null   int64 
 8   Main_meals_daily                1921 non-null   object
 9   Snacks                          1921 non-null   object
 10  Height                          1921 non-null   object
 11  Smoker                          1921 non-null   object
 12  Water_daily                     1921 non-null   object
 13  Calorie_monitoring              1921 non-null   object
 14  Weight                          1921 non-null   object
 15  Physical_activity_level         1921 non-null   object
 16  Technology_time_use             1921 non-null   int64 
 17  Gender                          1921 non-null   object
 18  Diagnostic                      1921 non-null   object
dtypes: int64(2), object(17)
memory usage: 285.3+ KB
None
```

Observam un esantion mic de date

```
print(df.head());
```

```
           Transportation Regular_fiber_diet Diagnostic_in_family_history  \
0  Public_Transportation                  2                          yes   
1  Public_Transportation                  3                          yes   
2  Public_Transportation                  2                          yes   
3               Walking                   3                           no   
4  Public_Transportation                  2                           no   

  High_calorie_diet Sedentary_hours_daily Age     Alcohol  \
0                no                 3,73  21          no   
1                no                 2,92  21   Sometimes   
2                no                 3,85  23  Frequently   
3                no                 3,01  27  Frequently   
4                no                 2,73  22   Sometimes   

   Est_avg_calorie_intake Main_meals_daily     Snacks Height Smoker  \
0                    2474                3  Sometimes   1,62     no   
1                    2429                3  Sometimes   1,52    yes   
2                    2656                3  Sometimes    1,8     no   
3                    2260                3  Sometimes    1,8     no   
4                    1895                1  Sometimes   1,78     no   

  Water_daily Calorie_monitoring Weight Physical_activity_level  \
0           2                 no     64                       0   
1           3                yes     56                       3   
2           2                 no     77                       2   
3           2                 no     -1                       2   
4           2                 no   89,8                       0   

   Technology_time_use  Gender Diagnostic
```

```
0                   1  Female         D1
1                   0  Female         D1
2                   1    Male         D1
3                   0    Male         D2
4                   0    Male         D3
```

Convertim atributele categorice numerice din format obiect (string) in format numeric

```
def convert_categorical_to_numeric(df):
    numeric_categorical_attributes = []

    for column in df.columns:
        # Verificăm dacă tipul de date al coloanei nu este deja numeric
        if df[column].dtype == 'object':
            # Convertim doar coloana curentă în tipul de date 'str'
            df[column] = df[column].astype(str)
            try:
                df[column] = df[column].str.replace(',', '.').astype(float)
                numeric_categorical_attributes.append(column)
            except ValueError:
                pass
    print("Atributele categorice convertite cu succes în numere:")
    print(numeric_categorical_attributes)

# Utilizare
convert_categorical_to_numeric(df)
```

```
Atributele categorice convertite cu succes în numere:
['Regular_fiber_diet', 'Sedentary_hours_daily', 'Age', 'Main_meals_daily', 'Height', 'Water_daily', 'Weight', 'Physical_activity_level']
```

Descoperim atributele numerice si categorice

```
# Identificarea atributelor numerice și categorice
numeric_attributes = df.select_dtypes(include=np.number).columns.tolist()
categorical_attributes = df.select_dtypes(exclude=np.number).columns.tolist()

# Afișarea atributele numerice și categorice
print("Atribute numerice:")
print(numeric_attributes)
print("\nAtribute categorice:")
print(categorical_attributes)
```

```
Atribute numerice:
['Regular_fiber_diet', 'Sedentary_hours_daily', 'Age', 'Est_avg_calorie_intake', 'Main_meals_daily', 'Height', 'Water_daily', 'Weight',

Atribute categorice:
['Transportation', 'Diagnostic_in_family_history', 'High_calorie_diet', 'Alcohol', 'Snacks', 'Smoker', 'Calorie_monitoring', 'Gender', '
```

Vizualizarea valorilor posibile pentru fiecare atribut numeric

```
def unique_values(column, dataframe):
    print(f"Valori unice pentru {column}, sortate:")
    print(dataframe[column].value_counts().sort_index())
    print(f"Total raspunsuri posibile: {dataframe[column].nunique()}")
    print()

# Afișarea valorilor unice pentru fiecare atribut numeric, sortate
for column in numeric_attributes:
    unique_values(column, df)
```

```
3.000000    146
Name: count, Length: 1147, dtype: int64
Total raspunsuri posibile: 1147

Valori unice pentru Weight, sortate:
Weight
-1.000000        190
 39.000000         1
 39.101805         1
 39.371523         1
 39.695295         1
                ...
 160.935351        1
 165.057269        1
 80539.000000      1
 82039.000000      1
 82628.000000      1
Name: count, Length: 1264, dtype: int64
Total raspunsuri posibile: 1264

Valori unice pentru Physical_activity_level, sortate:
Physical_activity_level
0.000000    381
0.000096      1
0.000272      1
0.000454      1
0.001015      1
           ...
2.939733      1
2.971832      1
2.998981      1
2.999918      1
3.000000     68
Name: count, Length: 1083, dtype: int64
Total raspunsuri posibile: 1083

Valori unice pentru Technology_time_use, sortate:
Technology_time_use
0       865
1       831
2       224
1306      1
Name: count, dtype: int64
Total raspunsuri posibile: 4
```

Calculam p-value pentru atributele numerice folosind ANOVA (Analiza Variației)

```
def calculate_anova_p_values(numeric_columns, target_column, dataframe):
    numeric_p_values = {}
    for column in numeric_columns:
        # Calculăm p-value folosind ANOVA
        p_value = f_oneway(*[dataframe[dataframe[target_column] == label][column] for label in dataframe[target_column].unique()])[1]
        numeric_p_values[column] = p_value
    return numeric_p_values

# Utilizare
target_column = "Diagnostic"
numeric_p_values = calculate_anova_p_values(numeric_attributes, target_column, df)

p_value_threshold = 0.05

# Listă pentru atributele numerice relevante
numeric_features = []

# Sortăm p-values-urile în ordine crescătoare
sorted_numeric_p_values = sorted(numeric_p_values.items(), key=lambda x: x[1])

print(f"Atributele cu p-value < {p_value_threshold}:")
for attribute, p_value in sorted_numeric_p_values:
    if p_value < p_value_threshold:
        numeric_features.append(attribute)
        print(f"{attribute}: {p_value}")

print("\n-----------------------------------------------------")
print(f"Atributele cu p-value >= {p_value_threshold}:")
for attribute, p_value in sorted_numeric_p_values:
    if p_value >= p_value_threshold:
        print(f"{attribute}: {p_value}")

print("\nLista de atribute numerice relevante:")
print(numeric_features)
```

```
Atributele cu p-value < 0.05:
Main_meals_daily: 1.0439849937990176e-28
Physical_activity_level: 4.1385763108655385e-17
Water_daily: 9.290933849587134e-17
Age: 0.04691944484302147

-----------------------------------------------------
Atributele cu p-value >= 0.05:
Est_avg_calorie_intake: 0.07135651633628821
Weight: 0.2680734718334284
Technology_time_use: 0.3965841022223923
Sedentary_hours_daily: 0.40455226509910797
Regular_fiber_diet: 0.41415163346177447
Height: 0.5819402028950371

Lista de atribute numerice relevante:
['Main_meals_daily', 'Physical_activity_level', 'Water_daily', 'Age']
```

Identificarea valorilor NaN mascate (exemplu: -1 pt atribute strict pozitive)

```
def discover_negative_nan(df, column_name):
    df.loc[df[column_name] <= 0, column_name] = np.nan
    num_nan_values = df[column_name].isna().sum()
    print(f"Numărul de valori NaN pentru coloana '{column_name}' este: {num_nan_values}")

# Utilizare
discover_negative_nan(df, 'Weight')
```

```
Numărul de valori NaN pentru coloana 'Weight' este: 190
```

Verificam pentru fiecare atribut cate exemple au valori nule (NaN)

```
# Calculăm numărul de valori NaN pentru fiecare atribut în parte
nan_counts = df.isna().sum()

# Afișăm numărul de valori NaN pentru fiecare atribut
print("Numărul de valori NaN pentru fiecare atribut:")
print(nan_counts)
```

```
Numărul de valori NaN pentru fiecare atribut:
Transportation                    0
Regular_fiber_diet                0
Diagnostic_in_family_history      0
High_calorie_diet                 0
Sedentary_hours_daily             0
Age                               0
Alcohol                           0
Est_avg_calorie_intake            0
Main_meals_daily                  0
Snacks                            0
Height                            0
Smoker                            0
Water_daily                       0
Calorie_monitoring                0
Weight                          190
Physical_activity_level           0
Technology_time_use               0
Gender                            0
Diagnostic                        0
dtype: int64
```

Eliminam outlierii

```
def eliminate_outliers(df, columns, coeficient_outliers, outliers_percent_threshold):
    outliers_info = {}

    for column in columns:
        std_dev = df[column].std()
        coeficient = coeficient_outliers

        outlier_limit = coeficient * std_dev
        outliers = df[df[column] > outlier_limit][column]
        outliers_count = len(outliers)
        total_count = len(df)
        percent = (outliers_count / total_count) * 100

        if percent > outliers_percent_threshold:
            while percent > outliers_percent_threshold:
                coeficient += 1
                outlier_limit = coeficient * std_dev
                outliers = df[df[column] > outlier_limit][column]
                outliers_count = len(outliers)
                percent = (outliers_count / total_count) * 100

        # Modificăm data frame-ul și eliminăm outlierii
        outliers_info[column] = {
            'outliers': outliers.tolist(),
            'std_dev_times_outliers': (outliers - std_dev).tolist(),
            'outlier_limit': outlier_limit
        }
        df = df[df[column] <= outlier_limit]

    for column, info in outliers_info.items():
        print(f"Outlieri pentru coloana '{column}': {info['outliers']}")
        print(f"Diferența față de std dev pentru outlieri: {info['std_dev_times_outliers']}")
        print(f"Limita pentru coloana '{column}' este valoarea {info['outlier_limit']}\n")

    return df
```

Aplicam eliminarea valorilor extreme (Outliers) doar pentru atributele numerice relevante (features)

```
OUTLIERS_PERCENT = 1 # Procentul maxim de Outliers admis din total
COEFICIENT_OUTLIERS = 3 # De la ce valoare in sus (COEFICIENT_OUTLIERS * std_dev) consideram un numar ca fiind Outlier

df = eliminate_outliers(df, numeric_features, COEFICIENT_OUTLIERS, OUTLIERS_PERCENT)
```

```
Outlieri pentru coloana 'Main_meals_daily': []
Diferența față de std dev pentru outlieri: []
Limita pentru coloana 'Main_meals_daily' este valoarea 4.675074334107315

Outlieri pentru coloana 'Physical_activity_level': []
Diferența față de std dev pentru outlieri: []
Limita pentru coloana 'Physical_activity_level' este valoarea 3.4221025699209124

Outlieri pentru coloana 'Water_daily': []
```

```
Diferența față de std dev pentru outlieri: []
Limita pentru coloana 'Water_daily' este valoarea 3.0551710222578725

Outlieri pentru coloana 'Age': [19627.0, 19685.0]
Diferența față de std dev pentru outlieri: [18993.688162923285, 19051.688162923285]
Limita pentru coloana 'Age' este valoarea 1899.9355112301407
```

Calculam p-value pentru atributele numerice folosind Testul Chi-Squared

```
def calculate_chi_squared_p_values(categorical_columns, target_column, dataframe):
    categorical_p_values = {}
    for col in categorical_columns:
        # Nu adugam si targetul in lista de categorical_
        if col != target_column:
          # Construim tabloul de contingenta intre atributul categoric și atributul țintă
          contingency_table = pd.crosstab(dataframe[col], dataframe[target_column])

          # Calculam valoarea p folosind testul χ²
          chi2, p_value, _, _ = chi2_contingency(contingency_table)
          categorical_p_values[col] = p_value

    return categorical_p_values


# Utilizare
target_column = "Diagnostic"
categorical_p_values = calculate_chi_squared_p_values(categorical_attributes, target_column, df)

p_value_threshold = 0.05

# Listă pentru atributele categorice relevante
categorical_features = []

# Sortăm p-values-urile în ordine crescătoare
sorted_categorical_p_values = sorted(categorical_p_values.items(), key=lambda x: x[1])

print(f"Atributele categorice cu p-value < {p_value_threshold}:")
for attribute, p_value in sorted_categorical_p_values:
    if p_value < p_value_threshold:
        categorical_features.append(attribute)
        print(f"{attribute}: {p_value}")

print("\n-----------------------------------------------------")
print(f"Atributele categorice cu p-value >= {p_value_threshold}:")
for attribute, p_value in sorted_categorical_p_values:
    if p_value >= p_value_threshold:
        print(f"{attribute}: {p_value}")

print("\nLista de atribute categorice relevante:")
print(categorical_features)
```

```
Atributele categorice cu p-value < 0.05:
Snacks: 2.363507857062593e-138
Gender: 3.7389609161985087e-125
Diagnostic_in_family_history: 5.988839640471714e-119
Alcohol: 9.612007227696994e-56
Transportation: 1.304031832484721e-44
High_calorie_diet: 3.992374842310777e-42
Calorie_monitoring: 9.766290734496677e-22
Smoker: 3.52262992745387e-05

-----------------------------------------------------
Atributele categorice cu p-value >= 0.05:

Lista de atribute categorice relevante:
['Snacks', 'Gender', 'Diagnostic_in_family_history', 'Alcohol', 'Transportation', 'High_calorie_diet', 'Calorie_monitoring', 'Smoker']
```

Encoding (Translatarea datelor categorice in valori numerice)

Vizualizarea valorilor posibile pentru fiecare atribut categoric

```
# Afișarea valorilor unice pentru fiecare atribut categoric
for column in categorical_attributes:
    unique_values(column, df)
```

```
Valori unice pentru High_calorie_diet, sortate:
High_calorie_diet
no      224
yes    1695
Name: count, dtype: int64
Total raspunsuri posibile: 2

Valori unice pentru Alcohol, sortate:
Alcohol
Always           1
Frequently      66
Sometimes     1267
no             585
Name: count, dtype: int64
Total raspunsuri posibile: 4

Valori unice pentru Snacks, sortate:
Snacks
Always          48
Frequently     221
Sometimes     1607
no              43
Name: count, dtype: int64
Total raspunsuri posibile: 4

Valori unice pentru Smoker, sortate:
Smoker
no     1879
yes      40
Name: count, dtype: int64
Total raspunsuri posibile: 2

Valori unice pentru Calorie_monitoring, sortate:
Calorie_monitoring
no     1836
yes      83
Name: count, dtype: int64
Total raspunsuri posibile: 2

Valori unice pentru Gender, sortate:
Gender
Female    944
Male      975
Name: count, dtype: int64
Total raspunsuri posibile: 2

Valori unice pentru Diagnostic, sortate:
Diagnostic
D0    246
D1    262
D2    256
D3    269
D4    320
D5    270
D6    296
Name: count, dtype: int64
Total raspunsuri posibile: 7
```

Definim functiile pentru diferite metode de encoding

```
def one_hot_encode(df_categorical):
    # Inițializăm encoder-ul
    onehot_encoder = OneHotEncoder(sparse_output=False)

    # Aplicăm One-Hot Encoding
    onehot_encoded = onehot_encoder.fit_transform(df_categorical)

    # Transformăm rezultatul înapoi într-un DataFrame
    onehot_df = pd.DataFrame(onehot_encoded, columns=onehot_encoder.get_feature_names_out(df_categorical.columns))

    return onehot_df

def label_encode(df_categorical):
    # Inițializăm encoder-ul
    label_encoder = LabelEncoder()

    # Aplicăm Label Encoding pe fiecare coloană
    label_encoded_df = df_categorical.apply(label_encoder.fit_transform)

    return label_encoded_df

def binary_encode(df_categorical):
    # Inițializăm encoder-ul
    binary_encoder = ce.BinaryEncoder(cols=df_categorical.columns.tolist())

    # Aplicăm Binary Encoding
    binary_encoded_df = binary_encoder.fit_transform(df_categorical)

    return binary_encoded_df

def ordinal_encode(df_categorical):
    # Inițializăm encoder-ul
    ordinal_encoder = ce.OrdinalEncoder()

    # Aplicăm Ordinal Encoding
    ordinal_encoded_df = ordinal_encoder.fit_transform(df_categorical)

    return ordinal_encoded_df

def target_encode(df_categorical, target_column):
    # Inițializăm encoder-ul
    target_encoder = ce.TargetEncoder()

    # Aplicăm Target Encoding
    target_encoded_df = target_encoder.fit_transform(df_categorical, target_column)

    return target_encoded_df
```

Folosim one_hot_encode ca exemplu

```
df_encoded = one_hot_encode(df[categorical_attributes])
print(df_encoded.head())
```

```
   Transportation_Automobile  Transportation_Bike  Transportation_Motorbike  \
0                        0.0                  0.0                       0.0
1                        0.0                  0.0                       0.0
2                        0.0                  0.0                       0.0
3                        0.0                  0.0                       0.0
4                        0.0                  0.0                       0.0

   Transportation_Public_Transportation  Transportation_Walking  \
0                                   1.0                    0.0
1                                   1.0                    0.0
2                                   1.0                    0.0
3                                   0.0                    1.0
4                                   1.0                    0.0

   Diagnostic_in_family_history_no  Diagnostic_in_family_history_yes  \
0                              0.0                               1.0
1                              0.0                               1.0
2                              0.0                               1.0
3                              1.0                               0.0
4                              1.0                               0.0

   High_calorie_diet_no  High_calorie_diet_yes  Alcohol_Always  ...  \
0                   1.0                    0.0             0.0  ...
1                   1.0                    0.0             0.0  ...
2                   1.0                    0.0             0.0  ...
```

```
3                  1.0                   0.0            0.0  ...
4                  1.0                   0.0            0.0  ...

   Calorie_monitoring_yes  Gender_Female  Gender_Male  Diagnostic_D0  \
0                     0.0            1.0          0.0            0.0
1                     1.0            1.0          0.0            0.0
2                     0.0            0.0          1.0            0.0
3                     0.0            0.0          1.0            0.0
4                     0.0            0.0          1.0            0.0

   Diagnostic_D1  Diagnostic_D2  Diagnostic_D3  Diagnostic_D4  Diagnostic_D5  \
0            1.0            0.0            0.0            0.0            0.0
1            1.0            0.0            0.0            0.0            0.0
2            1.0            0.0            0.0            0.0            0.0
3            0.0            1.0            0.0            0.0            0.0
4            0.0            0.0            1.0            0.0            0.0

   Diagnostic_D6
0            0.0
1            0.0
2            0.0
3            0.0
4            0.0

[5 rows x 30 columns]
```

Cream setul de features, in urma prelucrarilor de date

```
features = numeric_features + df_encoded.columns.tolist()
print(features)

# Concatenăm DataFrame-urile df și df_encoded pe axa coloanelor (axis=1)
df = pd.concat([df, df_encoded], axis=1)
```

```
['Main_meals_daily', 'Physical_activity_level', 'Water_daily', 'Age', 'Transportation_Automobile', 'Transportation_Bike', 'Transportatio
```

```
num_examples = df.shape[0]
print("Numărul de exemple din setul de date:", num_examples)
```

```
Numărul de exemple din setul de date: 1921
```

Vizualizam metricile de referinta

```
# Calculăm metricile de interes pentru fiecare atribut în parte
metrics = df.describe()
print(metrics)
```

```
std             62.472149             21.771171     6.409236
min              1.000000              2.210000    15.000000
25%              2.000000              2.770000    19.967065
50%              2.387426              3.130000    22.829681
75%              3.000000              3.640000    26.000000
max           2739.000000            956.580000    61.000000

       Est_avg_calorie_intake  Main_meals_daily       Height  Water_daily  \
count             1919.000000       1919.000000  1919.000000  1919.000000
mean              2253.688900          2.682517     3.575305     2.009824
std                434.099708          0.779014    58.128416     0.611096
min               1500.000000          1.000000     1.450000     1.000000
25%               1871.500000          2.658558     1.630000     1.605075
50%               2253.000000          3.000000     1.700000     2.000000
75%               2628.000000          3.000000     1.770000     2.480416
max               3000.000000          4.000000  1915.000000     3.000000

             Weight  Physical_activity_level  Technology_time_use  ...  \
count   1729.000000              1919.000000          1919.000000  ...
mean     228.481515                 1.011580             1.346535  ...
std     3399.367719                 0.855339            29.805439  ...
min       39.000000                 0.000000             0.000000  ...
25%       65.912688                 0.115671             0.000000  ...
50%       83.325800                 1.000000             1.000000  ...
75%      108.090006                 1.682700             1.000000  ...
max    82628.000000                 3.000000          1306.000000  ...
```

```
std                 0.203476       0.500065     0.500065       0.334390
min                 0.000000       0.000000     0.000000       0.000000
25%                 0.000000       0.000000     0.000000       0.000000
50%                 0.000000       0.000000     1.000000       0.000000
75%                 0.000000       1.000000     1.000000       0.000000
max                 1.000000       1.000000     1.000000       1.000000

       Diagnostic_D1  Diagnostic_D2  Diagnostic_D3  Diagnostic_D4  \
count    1919.000000    1919.000000    1919.000000    1919.000000
mean        0.136529       0.133403       0.140177       0.166754
std         0.343439       0.340098       0.347261       0.372853
min         0.000000       0.000000       0.000000       0.000000
25%         0.000000       0.000000       0.000000       0.000000
50%         0.000000       0.000000       0.000000       0.000000
75%         0.000000       0.000000       0.000000       0.000000
max         1.000000       1.000000       1.000000       1.000000

       Diagnostic_D5  Diagnostic_D6
count    1919.000000    1919.000000
mean        0.140698       0.154247
std         0.347801       0.361280
min         0.000000       0.000000
25%         0.000000       0.000000
50%         0.000000       0.000000
75%         0.000000       0.000000
max         1.000000       1.000000

[8 rows x 40 columns]
```

1. Analiza echilibrului de clase

```
# Plecam de la setul de date original
df = pd.read_csv(f"{DATASET_PATH}/date_tema_1_iaut_2024.csv")
```

```
# Convertim atributele categorice in numerice
convert_categorical_to_numeric(df)
```

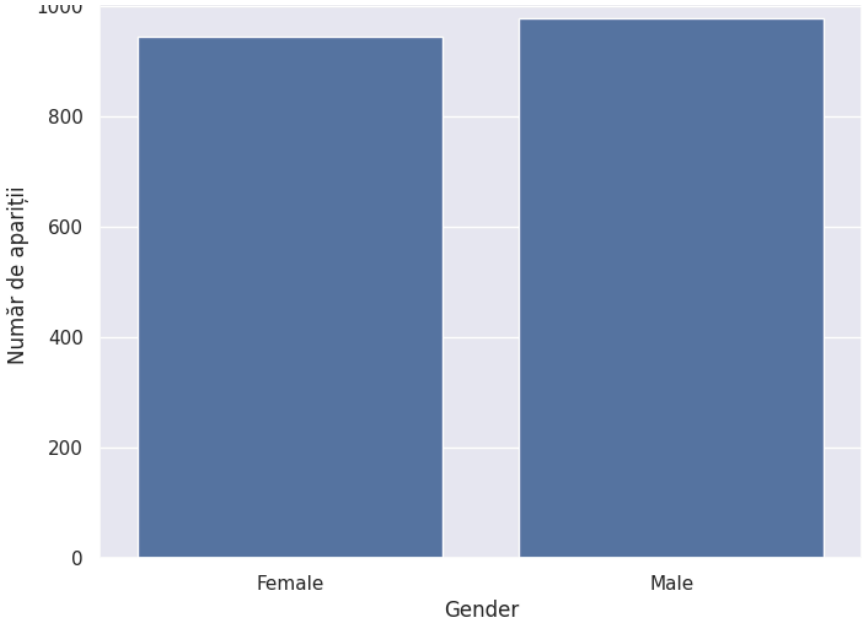
```
Atributele categorice convertite cu succes în numere:
['Regular_fiber_diet', 'Sedentary_hours_daily', 'Age', 'Main_meals_daily', 'Height', 'Water_daily', 'Weight', 'Physical_activity_level']
```

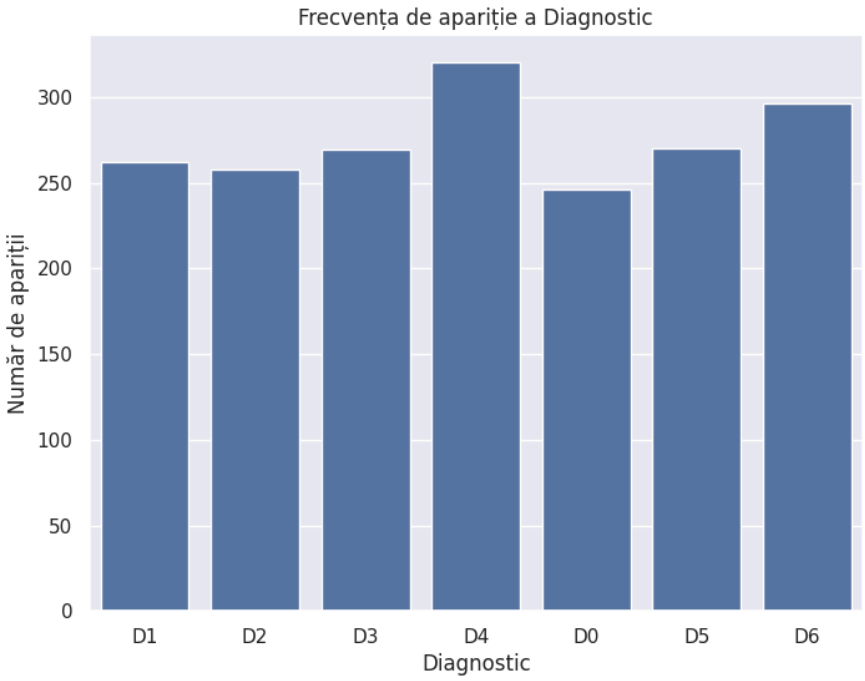
```
# Identificarea atributelor numerice și categorice
numeric_attributes = df.select_dtypes(include=np.number).columns.tolist()
categorical_attributes = df.select_dtypes(exclude=np.number).columns.tolist()

# Afișarea atributele numerice și categorice
print("Atribute numerice:")
print(numeric_attributes)
print("\nAtribute categorice:")
print(categorical_attributes)
```

```
Atribute numerice:
['Regular_fiber_diet', 'Sedentary_hours_daily', 'Age', 'Est_avg_calorie_intake', 'Main_meals_daily', 'Height', 'Water_daily', 'Weight',

Atribute categorice:
['Transportation', 'Diagnostic_in_family_history', 'High_calorie_diet', 'Alcohol', 'Snacks', 'Smoker', 'Calorie_monitoring', 'Gender', '
```

Analiza echilibrului de clase pentru atributele categorice

```
# Afișează un count plot pentru o coloana categorica
def plot_categorical_attributes_countplot(df, column):
    plt.figure(figsize=(8, 6))
    sns.countplot(data=df, x=column)
    plt.title(f'Frecvența de apariție a {column}')
    plt.xlabel(column)
    plt.ylabel('Număr de apariții')
    plt.show()


for column in categorical_attributes:
    plot_categorical_attributes_countplot(df, column)
```


1000
800
600
400
200
0
Număr de apariții
Female
Male
Gender


Frecvența de apariție a Diagnostic
300
250
200
150
100
50
0
Număr de apariții
D1
D2
D3
D4
D0
D5
D6
Diagnostic

Analiza echilibrului de clase pentru atributele numerice

```
# Afișează o distribuție gaussiană pentru o coloană numerică
def plot_numeric_gaussian_distribution(df, column):
    plt.figure(figsize=(10, 6))
    sns.histplot(df[column], kde=True)
    plt.title(f'Distribuția gaussiană a coloanei {column}')
    plt.xlabel(column)
    plt.ylabel('Densitate')
    print(f"{column} has min={df[column].min()} and max={df[column].max()}")
    # Specificăm limitele axei x
    plt.xlim(df[column].min(), df[column].max())

    plt.show()
```

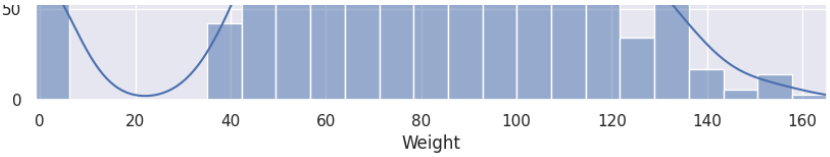
Eliminam valorile foarte extreme, care impiedica vizualizarea

```
OUTLIERS_PERCENT = 1 # Procentul maxim de Outliers admis din total
COEFICIENT_OUTLIERS = 3 # De la ce valoare in sus (COEFICIENT_OUTLIERS * std_dev) consideram un numar ca fiind Outlier

df_copy = eliminate_outliers(df, numeric_attributes, COEFICIENT_OUTLIERS, OUTLIERS_PERCENT)
```

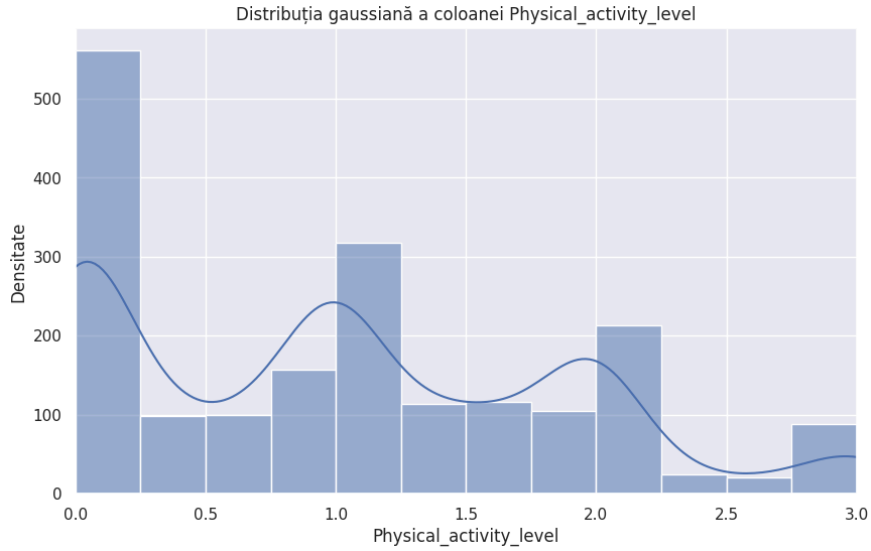
```
     Outlieri pentru coloana 'Regular_fiber_diet': [2739.0]
     Diferența față de std dev pentru outlieri: [2676.5603825004314]
     Limita pentru coloana 'Regular_fiber_diet' este valoarea 187.3188524987052

     Outlieri pentru coloana 'Sedentary_hours_daily': [956.58]
     Diferența față de std dev pentru outlieri: [934.8145156796575]
     Limita pentru coloana 'Sedentary_hours_daily' este valoarea 65.29645296102771

     Outlieri pentru coloana 'Age': [19627.0, 19685.0]
     Diferența față de std dev pentru outlieri: [18993.35855044086, 19051.35855044086]
     Limita pentru coloana 'Age' este valoarea 1900.9243486774217

     Outlieri pentru coloana 'Est_avg_calorie_intake': []
     Diferența față de std dev pentru outlieri: []
     Limita pentru coloana 'Est_avg_calorie_intake' este valoarea 3038.709888141891

     Outlieri pentru coloana 'Main_meals_daily': []
     Diferența față de std dev pentru outlieri: []
     Limita pentru coloana 'Main_meals_daily' este valoarea 4.673272004393935

     Outlieri pentru coloana 'Height': [1683.0, 1915.0]
     Diferența față de std dev pentru outlieri: [1624.84128360941, 1856.84128360941]
```

```
Limita pentru coloana 'Height' este valoarea 174.47614917177003

Outlieri pentru coloana 'Water_daily': []
Diferența față de std dev pentru outlieri: []
Limita pentru coloana 'Water_daily' este valoarea 3.0563175497433854

Outlieri pentru coloana 'Weight': [80539.0, 82628.0, 82039.0]
Diferența față de std dev pentru outlieri: [77308.30042899738, 79397.30042899738, 78808.30042899738]
Limita pentru coloana 'Weight' este valoarea 9692.09871300787

Outlieri pentru coloana 'Physical_activity_level': []
Diferența față de std dev pentru outlieri: []
Limita pentru coloana 'Physical_activity_level' este valoarea 3.421443675721591

Outlieri pentru coloana 'Technology_time_use': []
Diferența față de std dev pentru outlieri: []
Limita pentru coloana 'Technology_time_use' este valoarea 2.0254784639247636
```

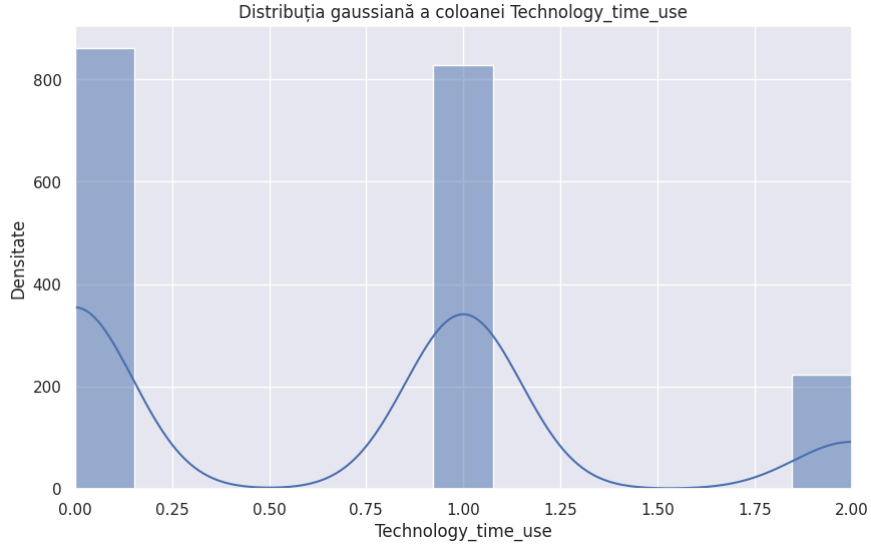
```
for column in numeric_attributes:
    plot_numeric_gaussian_distribution(df_copy, column)
```




```
Physical_activity_level has min=0.0 and max=3.0
```




```
Technology_time_use has min=0 and max=2
```



2. Vizualizarea datelor

Pentru analiza atributelor numerice:

```
def calculate_mad(series):
    mean = series.mean()
    absolute_deviations = abs(series - mean)
    mad = absolute_deviations.mean()
    return mad



def analyze_numeric_attributes(df, numeric_attributes):
    statistics_dict = {}
    for column in numeric_attributes:
        column_statistics = {
            'Medie': df[column].mean(),
            'Abaterea standard': df[column].std(),
            'Abaterea medie absolută': calculate_mad(df[column]),
            'Valoare minimă': df[column].min(),
            'Valoare maximă': df[column].max(),
            'Diferența de valori maxime și minime': df[column].max() - df[column].min(),
            'Mediană': df[column].median(),
            'Abaterea mediană absolută': calculate_mad(abs(df[column] - df[column].median())),
            'Intervalul intercuartil': df[column].quantile(0.75) - df[column].quantile(0.25)
        }
        statistics_dict[column] = column_statistics
    return statistics_dict
```

Pentru analiza atributelor categorice:

```
def analyze_categorical_attributes(column):
    print("\nAnaliză pentru coloana categorică:", column.name)
    print("Valori unice:", column.unique())
    print("\nHistogramă:")
    column.value_counts().plot(kind='bar')
    plt.title('Histograma pentru ' + column.name)
    plt.xlabel(column.name)
    plt.ylabel('Frecvență')
    plt.show()
```

Pentru analiza de covarianță între atribute:

```
def analyze_covariance(df):
    print("\nAnaliză de covarianță între coloanele numerice:")
    covariance_matrix = df.cov()
    print(covariance_matrix)
```

Pentru analiza de covarianță între atribut și clasă:

```
def analyze_covariance_with_class(df, class_column):
    print("\nAnaliză de covarianță între atribute și clasă:")
    for column in df.select_dtypes(include='number').columns:
        covariance_with_class = df[[column, class_column]].cov().iloc[0, 1]
        print(f"Covarianța între '{column}' și '{class_column}' este: {covariance_with_class}")
```

Analiza pentru atribute numerice

```
numeric_statistics = analyze_numeric_attributes(df, numeric_attributes)
for column, statistics in numeric_statistics.items():
    print(f"Analiză pentru coloana numerică '{column}':")
    for statistic, value in statistics.items():
        print(f"{statistic}: {value}")
    print()
```

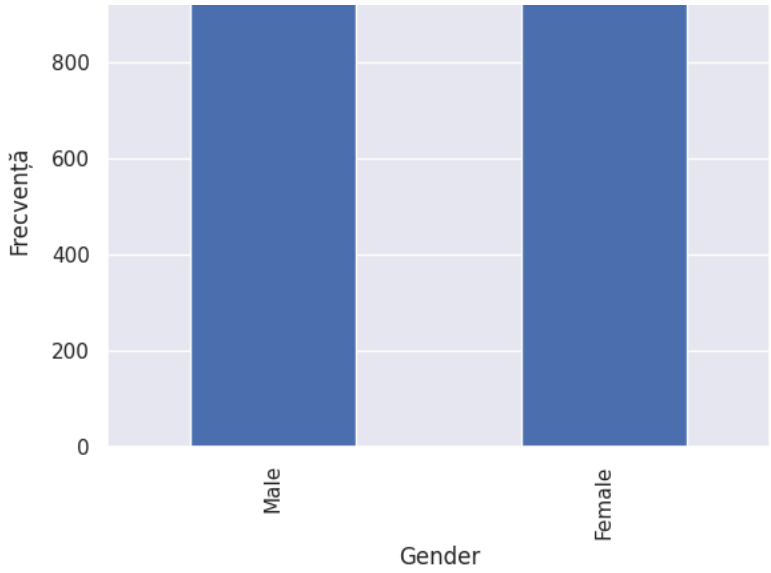
```
Valoare minimă: 1.45
Valoare maximă: 1915.0
Diferența de valori maxime și minime: 1913.55
Mediană: 1.7
Abaterea mediană absolută: 3.7383692067808036
Intervalul intercuartil: 0.14000000000000012

Analiză pentru coloana numerică 'Water_daily':
Medie: 2.010367264445601
Abaterea standard: 0.6110342044515745
Abaterea medie absolută: 0.47080058590915874
Valoare minimă: 1.0
Valoare maximă: 3.0
Diferența de valori maxime și minime: 2.0
Mediană: 2.0
Abaterea mediană absolută: 0.3563987719013474
Intervalul intercuartil: 0.8744789999999998

Analiză pentru coloana numerică 'Weight':
Medie: 205.63734420249872
Abaterea standard: 3225.6535358208953
Abaterea medie absolută: 254.64767097073664
Valoare minimă: -1.0
Valoare maximă: 82628.0
Diferența de valori maxime și minime: 82629.0
Mediană: 80.386078
Abaterea mediană absolută: 254.5539531306053
Intervalul intercuartil: 46.20536499999999

Analiză pentru coloana numerică 'Physical_activity_level':
Medie: 1.0126402805830297
Abaterea standard: 0.8555256424802281
Abaterea medie absolută: 0.7021597492776216
Valoare minimă: 0.0
Valoare maximă: 3.0
Diferența de valori maxime și minime: 3.0
Mediană: 1.0
Abaterea mediană absolută: 0.40545751065418223
Intervalul intercuartil: 1.567523

Analiză pentru coloana numerică 'Technology_time_use':
Medie: 1.3456533055700157
Abaterea standard: 29.789928441759592
Abaterea medie absolută: 1.5109089081173832
Valoare minimă: 0
Valoare maximă: 1306
Diferența de valori maxime și minime: 1306
Mediană: 1.0
Abaterea mediană absolută: 1.3573698845143172
Intervalul intercuartil: 1.0
```

Analiza pentru atribute numerice (fara Outlieri foarte extremi)

```
discover_negative_nan(df_copy, 'Weight')

numeric_statistics_no_outliers = analyze_numeric_attributes(df_copy, numeric_attributes)
for column, statistics in numeric_statistics_no_outliers.items():
    print(f"Analiză pentru coloana numerică '{column}':")
    for statistic, value in statistics.items():
        print(f"{statistic}: {value}")
    print()
```

```
Valoare maximă: 3.0
Diferența de valori maxime și minime: 2.0
Mediană: 2.0
Abaterea mediană absolută: 0.3565552626967315
Intervalul intercuartil: 0.8768779999999998

Analiză pentru coloana numerică 'Weight':
Medie: 86.77543685481997
Abaterea standard: 26.24922608081751
Abaterea medie absolută: 21.89662900606755
Valoare minimă: 39.0
Valoare maximă: 165.057269
Diferența de valori maxime și minime: 126.05726899999999
Mediană: 83.2839925
Abaterea mediană absolută: 12.449792629873563
Intervalul intercuartil: 42.56080974999999

Analiză pentru coloana numerică 'Physical_activity_level':
Medie: 1.0120893080543933
Abaterea standard: 0.8553609189303978
Abaterea medie absolută: 0.7019277278173744
Valoare minimă: 0.0
Valoare maximă: 3.0
Diferența de valori maxime și minime: 3.0
Mediană: 1.0
Abaterea mediană absolută: 0.4053436104398995
Intervalul intercuartil: 1.5678555

Analiză pentru coloana numerică 'Technology_time_use':
Medie: 0.6663179916317992
Abaterea standard: 0.6751594879749212
Abaterea medie absolută: 0.6001043836767563
Valoare minimă: 0
Valoare maximă: 2
Diferența de valori maxime și minime: 2
Mediană: 1.0
Abaterea mediană absolută: 0.49103657148859436
Intervalul intercuartil: 1.0
```

```
# Analiza pentru atribute categorice
for column in categorical_attributes:
    analyze_categorical_attributes(df[column])
```




```
Analiză pentru coloana categorică: Diagnostic
Valori unice: ['D1' 'D2' 'D3' 'D4' 'D0' 'D5' 'D6']

Histogramă:
```