Tema1_ML2024

May 1, 2024

1 Tema 1 - Invatare Automata 2024

Fotin Andrei-Stefan 343C3

[58]:
```
getDependencies = True #@param ["False", "True"] {type:"raw"}
```

[2]:
```
if getDependencies:
    !pip install seaborn
    !pip install scipy
    !pip install scikit-learn
    !pip install category_encoders
    !pip install pandas

# 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, PowerTransformer,␣
 ↪QuantileTransformer
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
```

```
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: seaborn in
/home/stefan/.local/lib/python3.10/site-packages (0.13.2)
Requirement already satisfied: numpy!=1.24.0,>=1.20 in
/home/stefan/.local/lib/python3.10/site-packages (from seaborn) (1.26.4)
Requirement already satisfied: pandas>=1.2 in
/home/stefan/.local/lib/python3.10/site-packages (from seaborn) (2.2.2)
Requirement already satisfied: matplotlib!=3.6.1,>=3.4 in
/home/stefan/.local/lib/python3.10/site-packages (from seaborn) (3.8.4)
Requirement already satisfied: contourpy>=1.0.1 in
/home/stefan/.local/lib/python3.10/site-packages (from
matplotlib!=3.6.1,>=3.4->seaborn) (1.2.1)
Requirement already satisfied: cycler>=0.10 in
/home/stefan/.local/lib/python3.10/site-packages (from
matplotlib!=3.6.1,>=3.4->seaborn) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in
/home/stefan/.local/lib/python3.10/site-packages (from
matplotlib!=3.6.1,>=3.4->seaborn) (4.51.0)
Requirement already satisfied: kiwisolver>=1.3.1 in
/home/stefan/.local/lib/python3.10/site-packages (from
matplotlib!=3.6.1,>=3.4->seaborn) (1.4.5)
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>=8 in /usr/lib/python3/dist-packages (from
matplotlib!=3.6.1,>=3.4->seaborn) (9.0.1)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/lib/python3/dist-
packages (from matplotlib!=3.6.1,>=3.4->seaborn) (2.4.7)
Requirement already satisfied: python-dateutil>=2.7 in
/usr/local/lib/python3.10/dist-packages (from matplotlib!=3.6.1,>=3.4->seaborn)
(2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in /usr/lib/python3/dist-packages
(from pandas>=1.2->seaborn) (2022.1)
Requirement already satisfied: tzdata>=2022.7 in
/home/stefan/.local/lib/python3.10/site-packages (from pandas>=1.2->seaborn)
(2024.1)
Requirement already satisfied: six>=1.5 in /usr/lib/python3/dist-packages (from
python-dateutil>=2.7->matplotlib!=3.6.1,>=3.4->seaborn) (1.16.0)
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: scipy in /home/stefan/.local/lib/python3.10/site-
packages (1.13.0)
Requirement already satisfied: numpy<2.3,>=1.22.4 in
/home/stefan/.local/lib/python3.10/site-packages (from scipy) (1.26.4)
Defaulting to user installation because normal site-packages is not writeable
```

```
Requirement already satisfied: scikit-learn in
/home/stefan/.local/lib/python3.10/site-packages (1.4.2)
Requirement already satisfied: numpy>=1.19.5 in
/home/stefan/.local/lib/python3.10/site-packages (from scikit-learn) (1.26.4)
Requirement already satisfied: scipy>=1.6.0 in
/home/stefan/.local/lib/python3.10/site-packages (from scikit-learn) (1.13.0)
Requirement already satisfied: joblib>=1.2.0 in
/home/stefan/.local/lib/python3.10/site-packages (from scikit-learn) (1.4.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in
/home/stefan/.local/lib/python3.10/site-packages (from scikit-learn) (3.4.0)
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: category_encoders in
/home/stefan/.local/lib/python3.10/site-packages (2.6.3)
Requirement already satisfied: numpy>=1.14.0 in
/home/stefan/.local/lib/python3.10/site-packages (from category_encoders)
(1.26.4)
Requirement already satisfied: scikit-learn>=0.20.0 in
/home/stefan/.local/lib/python3.10/site-packages (from category_encoders)
(1.4.2)
Requirement already satisfied: scipy>=1.0.0 in
/home/stefan/.local/lib/python3.10/site-packages (from category_encoders)
(1.13.0)
Requirement already satisfied: statsmodels>=0.9.0 in
/home/stefan/.local/lib/python3.10/site-packages (from category_encoders)
(0.14.2)
Requirement already satisfied: pandas>=1.0.5 in
/home/stefan/.local/lib/python3.10/site-packages (from category_encoders)
(2.2.2)
Requirement already satisfied: patsy>=0.5.1 in
/home/stefan/.local/lib/python3.10/site-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)
(2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in /usr/lib/python3/dist-packages
(from pandas>=1.0.5->category_encoders) (2022.1)
Requirement already satisfied: tzdata>=2022.7 in
/home/stefan/.local/lib/python3.10/site-packages (from
pandas>=1.0.5->category_encoders) (2024.1)
Requirement already satisfied: six in /usr/lib/python3/dist-packages (from
patsy>=0.5.1->category_encoders) (1.16.0)
Requirement already satisfied: joblib>=1.2.0 in
/home/stefan/.local/lib/python3.10/site-packages (from scikit-
learn>=0.20.0->category_encoders) (1.4.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in
/home/stefan/.local/lib/python3.10/site-packages (from scikit-
learn>=0.20.0->category_encoders) (3.4.0)
Requirement already satisfied: packaging>=21.3 in
```

```
/usr/local/lib/python3.10/dist-packages (from
statsmodels>=0.9.0->category_encoders) (24.0)
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: pandas in
/home/stefan/.local/lib/python3.10/site-packages (2.2.2)
Requirement already satisfied: numpy>=1.22.4 in
/home/stefan/.local/lib/python3.10/site-packages (from pandas) (1.26.4)
Requirement already satisfied: python-dateutil>=2.8.2 in
/usr/local/lib/python3.10/dist-packages (from pandas) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in /usr/lib/python3/dist-packages
(from pandas) (2022.1)
Requirement already satisfied: tzdata>=2022.7 in
/home/stefan/.local/lib/python3.10/site-packages (from pandas) (2024.1)
Requirement already satisfied: six>=1.5 in /usr/lib/python3/dist-packages (from
python-dateutil>=2.8.2->pandas) (1.16.0)
```

Citim setul de date

[3]:
```
DATASET_PATH = ""
df = pd.read_csv(f"date_tema_1_iaut_2024.csv")
```

2 3.1. Explorarea Datelor (Exploratory Data Analysis)

[4]:
```
num_examples = df.shape[0]
print("Numărul de exemple din setul de date:", num_examples)
```

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

[5]:
```
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

[6]:
```
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

[7]:
```
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

[8]:
```
# 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',
'Physical_activity_level', 'Technology_time_use']

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

Vizualizarea valorilor posibile pentru fiecare atribut numeric

[9]:
```python
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)
```

```
Valori unice pentru Regular_fiber_diet, sortate:
Regular_fiber_diet
1.000000        28
1.003566         1
1.005578         1
1.008760         1
1.031149         1
               ...
2.997524         1
2.997951         1
2.998441         1
3.000000       597
2739.000000      1
Name: count, Length: 733, dtype: int64
Total raspunsuri posibile: 733

Valori unice pentru Sedentary_hours_daily, sortate:
Sedentary_hours_daily
2.21      10
2.22       8
2.23      10
2.24       7
2.25       7
          ..
4.64       2
4.65       3
4.66       4
4.67       1
956.58     1
Name: count, Length: 239, dtype: int64
Total raspunsuri posibile: 239

Valori unice pentru Age, sortate:
Age
15.000000       1
16.000000       7
16.093234       1
```

```
16.129279       1
16.172992       1
               ..
55.246250       1
56.000000       1
61.000000       1
19627.000000    1
19685.000000    1
Name: count, Length: 1282, dtype: int64
Total raspunsuri posibile: 1282

Valori unice pentru Est_avg_calorie_intake, sortate:
Est_avg_calorie_intake
1500    2
1501    1
1502    1
1503    2
1504    1
       ..
2991    3
2995    2
2996    2
2998    2
3000    1
Name: count, Length: 1081, dtype: int64
Total raspunsuri posibile: 1081

Valori unice pentru Main_meals_daily, sortate:
Main_meals_daily
1.000000    178
1.000283      1
1.000414      1
1.000610      1
1.001383      1
           …
3.995957      1
3.998618      1
3.998766      1
3.999591      1
4.000000     62
Name: count, Length: 584, dtype: int64
Total raspunsuri posibile: 584

Valori unice pentru Height, sortate:
Height
1.45         1
1.46         1
1.48         3
```

1.49	3
1.50	14
1.51	11
1.52	18
1.53	25
1.54	18
1.55	29
1.56	35
1.57	26
1.58	25
1.59	27
1.60	66
1.61	55
1.62	83
1.63	68
1.64	60
1.65	79
1.66	52
1.67	63
1.68	57
1.69	50
1.70	112
1.71	63
1.72	66
1.73	39
1.74	61
1.75	110
1.76	89
1.77	67
1.78	58
1.79	61
1.80	56
1.81	34
1.82	47
1.83	34
1.84	38
1.85	35
1.86	18
1.87	20
1.88	10
1.89	4
1.90	7
1.91	11
1.92	3
1.93	4
1.94	1
1.98	2
1683.00	1

```
1915.00      1
Name: count, dtype: int64
Total raspunsuri posibile: 52

Valori unice pentru Water_daily, sortate:
Water_daily
1.000000    195
1.000463      1
1.000536      1
1.000544      1
1.000695      1
           …
2.991671      1
2.993448      1
2.994515      1
2.999495      1
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)

[10]:
```python
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)

[11]:
```
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)

[12]:
```
# 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

[13]:
```python
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)

[14]:
```
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

[15]:
```
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  ²
            chi2, p_value, _, _ = chi2_contingency(contingency_table)
            categorical_p_values[col] = p_value

    return categorical_p_values
```

[16]:
```
# 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

[17]:
```
# Afișarea valorilor unice pentru fiecare atribut categoric
for column in categorical_attributes:
    unique_values(column, df)
```

```
Valori unice pentru Transportation, sortate:
Transportation
Automobile               423
Bike                       7
Motorbike                 11
Public_Transportation    1425
Walking                    53
Name: count, dtype: int64
Total raspunsuri posibile: 5

Valori unice pentru Diagnostic_in_family_history, sortate:
Diagnostic_in_family_history
no      348
yes    1571
Name: count, dtype: int64
Total raspunsuri posibile: 2

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

[18]:
```python
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
```

One-Hot Encoding:

```
Pro-uri:
    Menține toate informațiile despre categorii în coloane binare separate.
    Funcționează bine cu algoritmi care presupun independența între caracteristici.
Contra-uri:
    Poate duce la seturi de date de dimensiuni mari, în special cu un număr mare de categorii u
    Poate cauza isprave de multicolinearitate dacă nu este gestionat corect.
```

Label Encoding:

```
Pro-uri:
    Metodă simplă și directă de encoding.
    Reduce dimensionalitatea în comparație cu one-hot encoding.
Contra-uri:
    Atribuie o ordine categoriilor, ceea ce ar putea induce în eroare unele algoritme.
    Nu este potrivit pentru algoritmi care presupun o distanță egală între categorii.
```

Binary Encoding:

```
Pro-uri:
    Reduce dimensionalitatea în comparație cu one-hot encoding.
    Păstrează informațiile despre ordinea categoriilor.
Contra-uri:
    Introduce ordinalitate care ar putea să nu fie potrivită pentru unele algoritme.
    Necesită prelucrare suplimentară pentru gestionarea valorilor lipsă.
```

Ordinal Encoding:

```
Pro-uri:
    Păstrează relația ordonată între categorii.
    Reduce dimensionalitatea în comparație cu one-hot encoding.
Contra-uri:
    Presupune ordinalitate care ar putea să nu fie precisă sau potrivită pentru toate variabile
    Utilizare limitată pentru categoriile non-ordonate.
```

Target Encoding:

```
Pro-uri:
    Captivează informații despre variabila țintă, îmbunătățind potențial performanța modelului
    Reduce dimensionalitatea în comparație cu one-hot encoding.
Contra-uri:
    Predispus la supradaptare, în special cu variabile categorice de înaltă cardinalitate.
    Sensibil la valori extreme în variabila țintă.
```

Folosim one_hot_encode ca exemplu

[19]:
```
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

[20]:
```
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', 'Transportation_Motorbike',
'Transportation_Public_Transportation', 'Transportation_Walking',
'Diagnostic_in_family_history_no', 'Diagnostic_in_family_history_yes',
'High_calorie_diet_no', 'High_calorie_diet_yes', 'Alcohol_Always',
'Alcohol_Frequently', 'Alcohol_Sometimes', 'Alcohol_no', 'Snacks_Always',
'Snacks_Frequently', 'Snacks_Sometimes', 'Snacks_no', 'Smoker_no', 'Smoker_yes',
'Calorie_monitoring_no', 'Calorie_monitoring_yes', 'Gender_Female',
'Gender_Male', 'Diagnostic_D0', 'Diagnostic_D1', 'Diagnostic_D2',
'Diagnostic_D3', 'Diagnostic_D4', 'Diagnostic_D5', 'Diagnostic_D6']
```

[21]:
```
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

[22]:
```
# Calculăm metricile de interes pentru fiecare atribut în parte
metrics = df.describe()
print(metrics)
```

```
       Regular_fiber_diet  Sedentary_hours_daily          Age  \
count         1919.000000            1919.000000  1919.000000
mean             3.846266               3.694033    24.353520
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  …

       Calorie_monitoring_yes  Gender_Female  Gender_Male  Diagnostic_D0  \
count             1919.000000    1919.000000  1919.000000    1919.000000
mean                 0.043252       0.491923     0.508077       0.128192
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]
```

3 1. Analiza echilibrului de clase

[23]:
```
# Plecam de la setul de date original
df = pd.read_csv(f"date_tema_1_iaut_2024.csv")
```

[24]:
```
# 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']
```

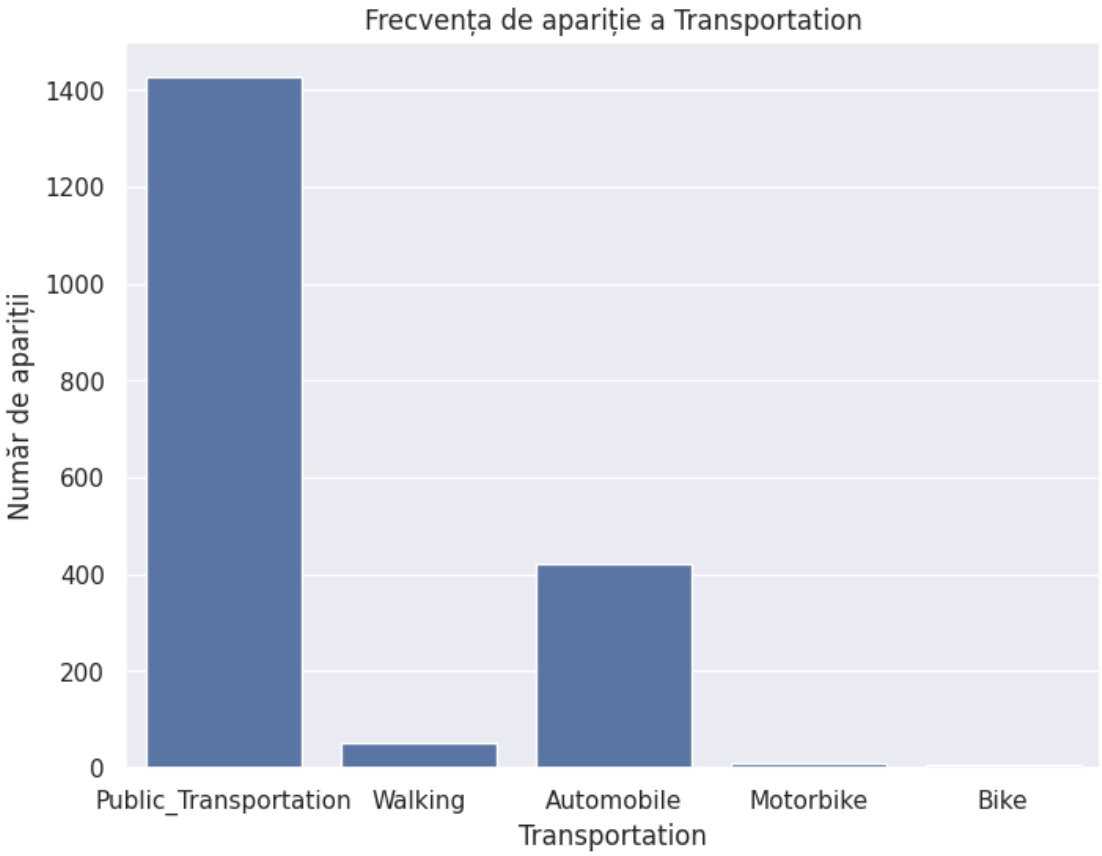
[25]:
```
# Identificarea atributelor numerice și categorice
numeric_attributes = df.select_dtypes(include=np.number).columns.tolist()
categorical_attributes = df.select_dtypes(exclude=np.number).columns.tolist()

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

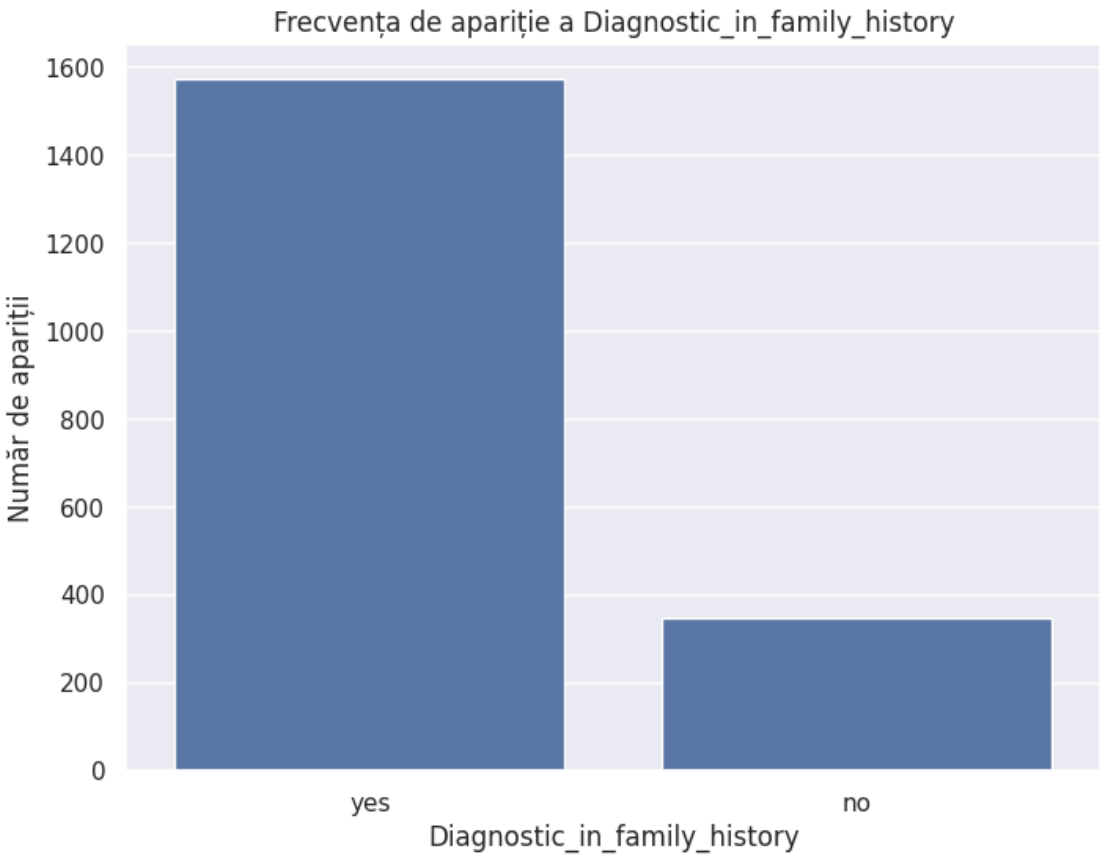
```
Atribute numerice:
['Regular_fiber_diet', 'Sedentary_hours_daily', 'Age', 'Est_avg_calorie_intake',
'Main_meals_daily', 'Height', 'Water_daily', 'Weight',
'Physical_activity_level', 'Technology_time_use']

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

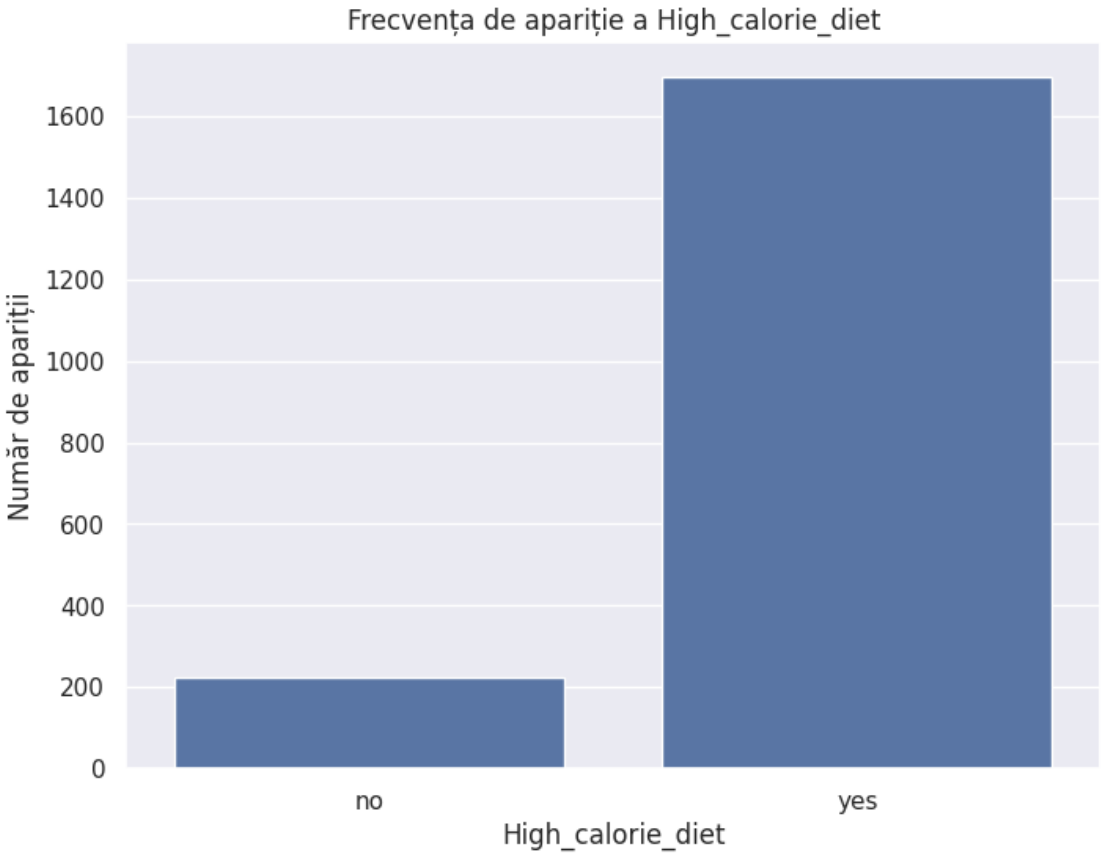
Analiza echilibrului de clase pentru atributele categorice

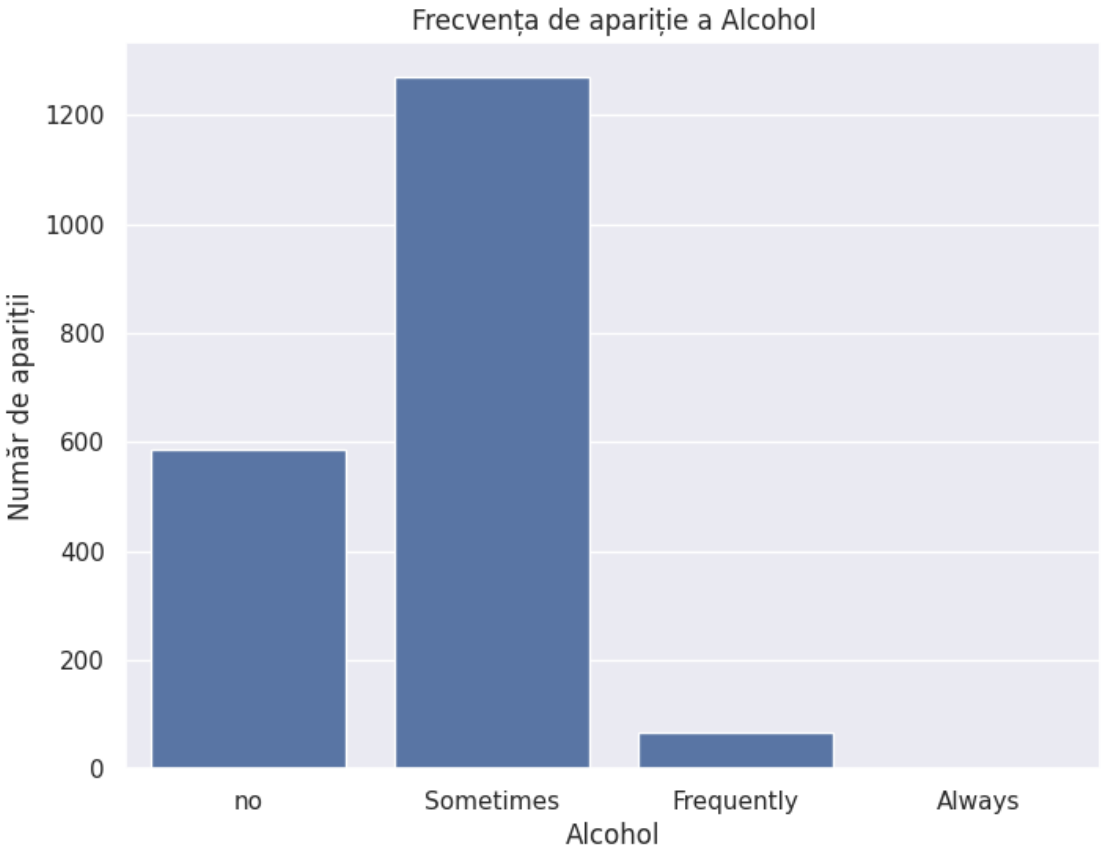
[26]:
```
# Afișează un count plot pentru o coloana categorica
def plot_categorical_attributes_countplot(df, column):
    plt.figure(figsize=(8, 6))
    sns.countplot(data=df, x=column)
    plt.title(f'Frecvența de apariție a {column}')
    plt.xlabel(column)
    plt.ylabel('Număr de apariții')
    plt.show()
```

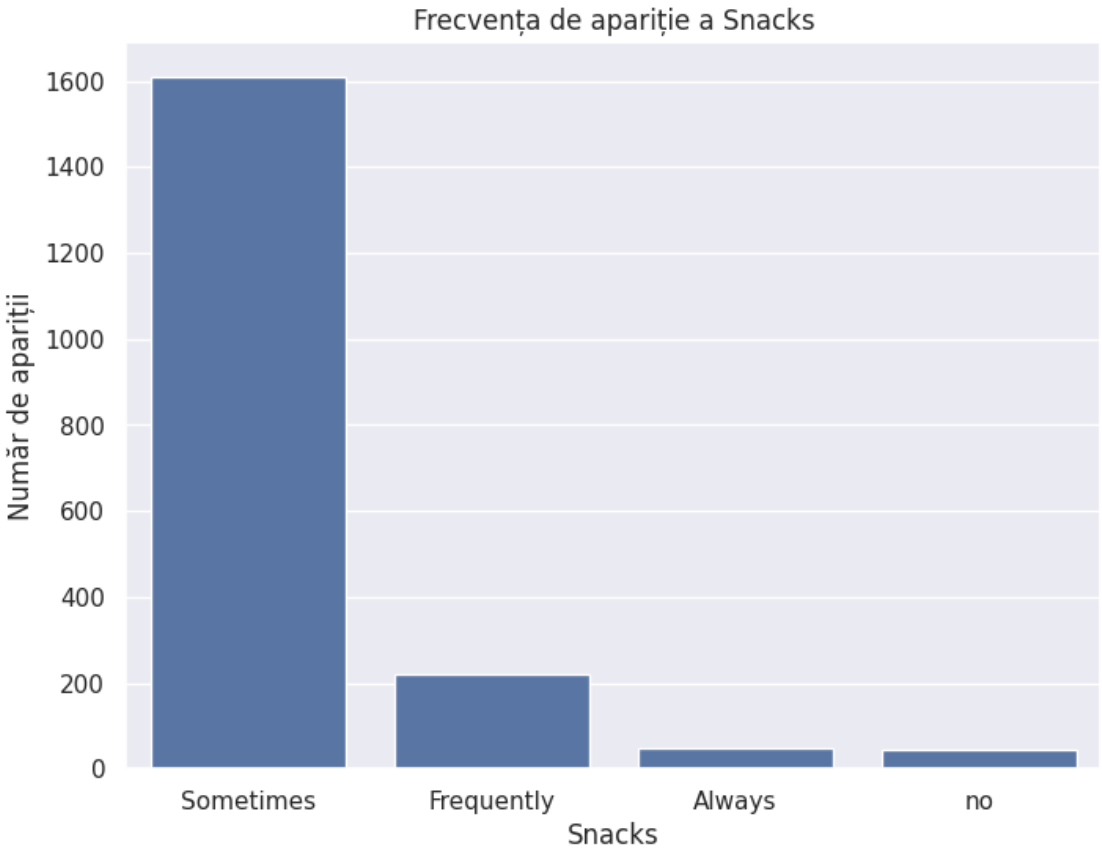
[27]:
```
for column in categorical_attributes:
    plot_categorical_attributes_countplot(df, column)
```

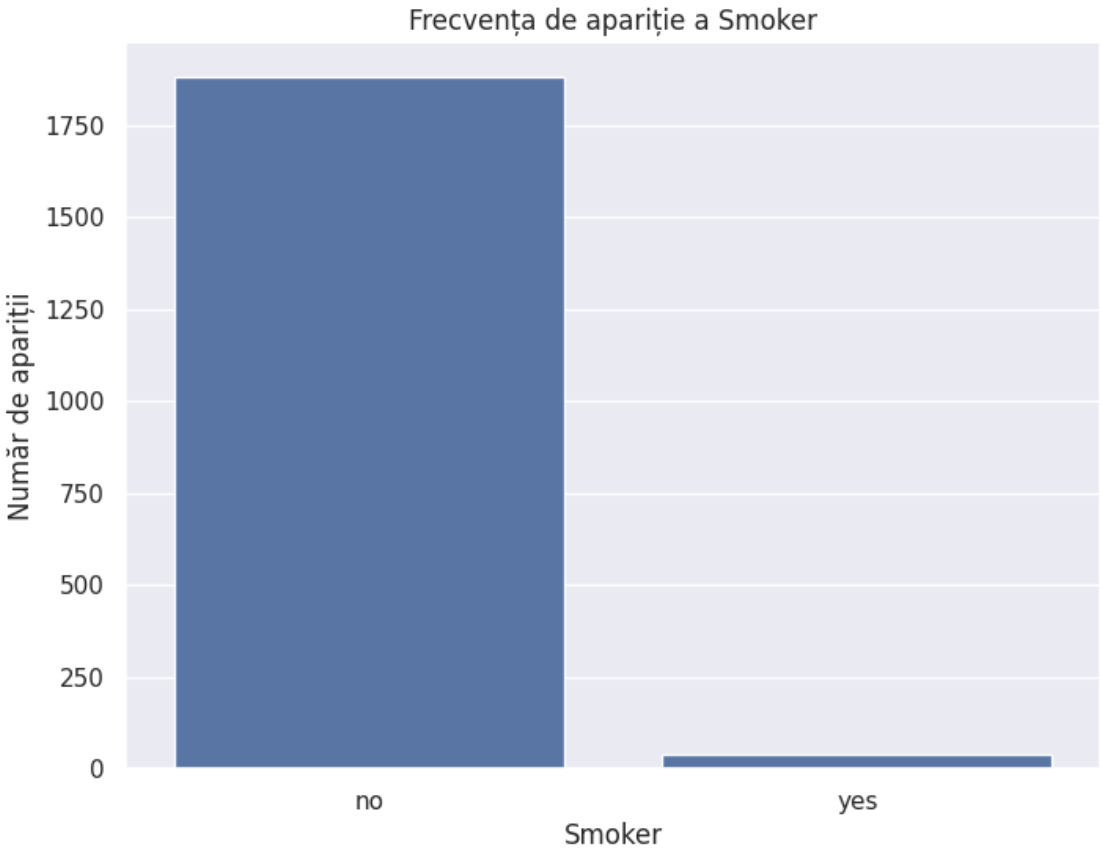

Frecvența de apariție a Transportation
Număr de apariții
0
200
400
600
800
1000
1200
1400
Public_Transportation
Walking
Automobile
Motorbike
Bike
Transportation

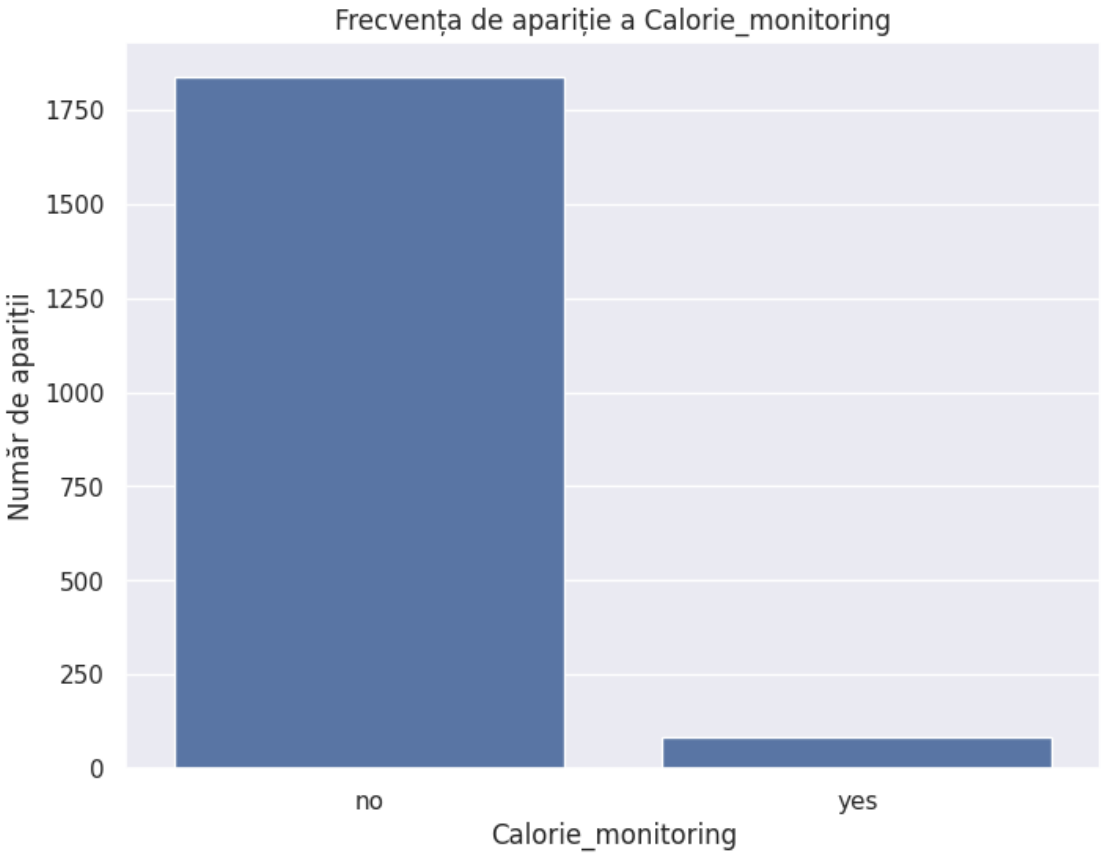

Frecvența de apariție a Diagnostic_in_family_history
Număr de apariții
1600
1400
1200
1000
800
600
400
200
0
yes
no
Diagnostic_in_family_history

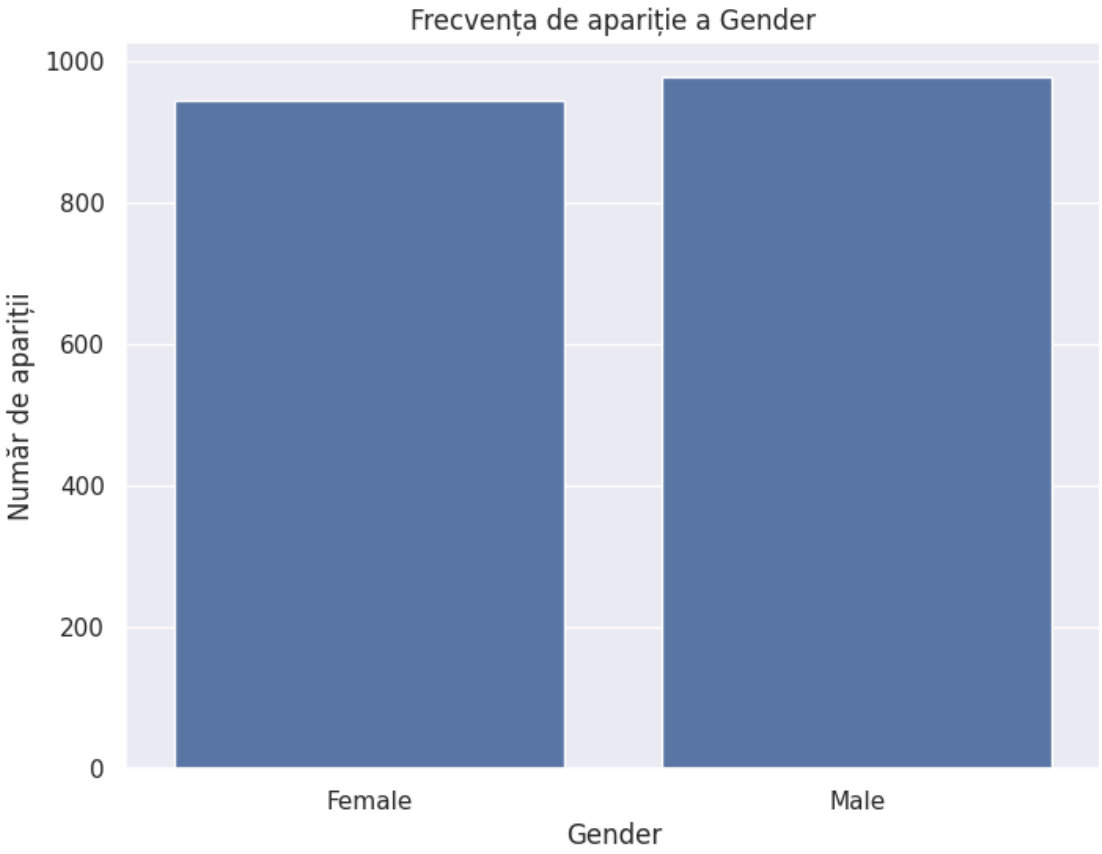

Frecvența de apariție a High_calorie_diet
Număr de apariții
0
200
400
600
800
1000
1200
1400
1600
no
yes
High_calorie_diet

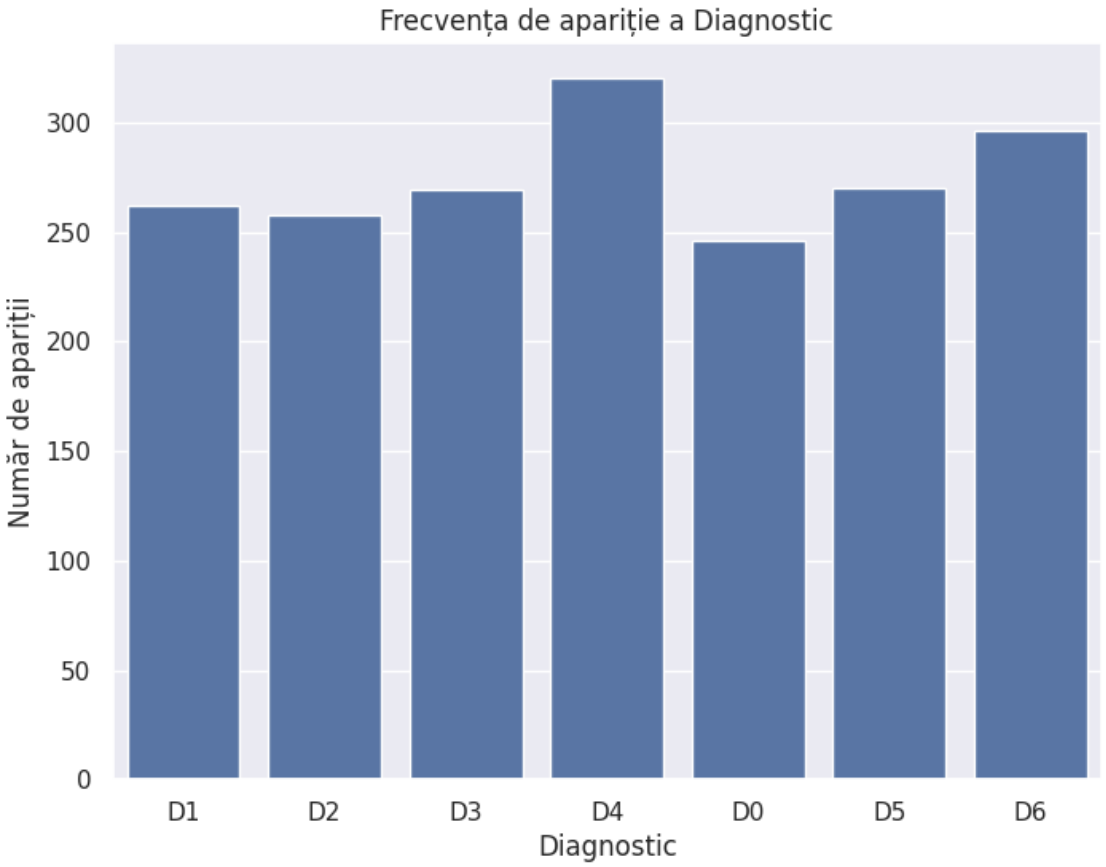

Frecvența de apariție a Alcohol
Număr de apariții
0
200
400
600
800
1000
1200
no
Sometimes
Frequently
Always
Alcohol


Frecvența de apariție a Snacks
Număr de apariții
0
200
400
600
800
1000
1200
1400
1600
Sometimes
Frequently
Always
no
Snacks


Frecvența de apariție a Smoker
Număr de apariții
1750
1500
1250
1000
750
500
250
0
no
yes
Smoker


Frecvența de apariție a Calorie_monitoring
Număr de apariții
1750
1500
1250
1000
750
500
250
0
no
yes
Calorie_monitoring


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



Analiza echilibrului de clase pentru atributele numerice

[28]:
```
# 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

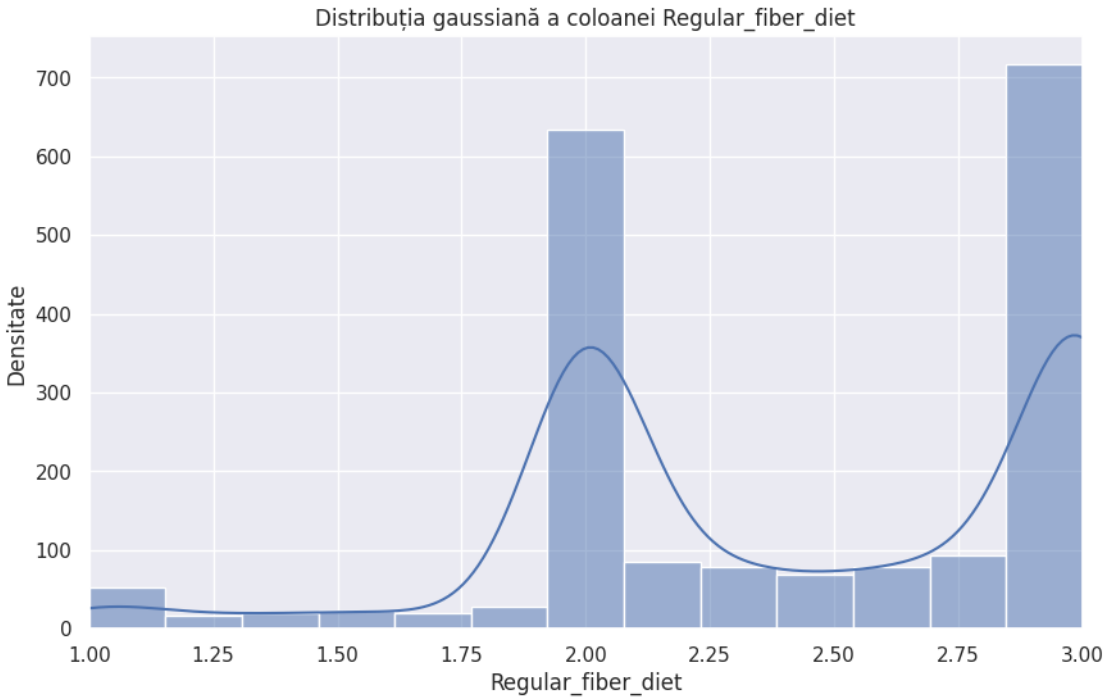
[29]:
```
OUTLIERS_PERCENT = 1 # Procentul maxim de Outliers admis din total
COEFICIENT_OUTLIERS = 3 # De la ce valoare in sus (COEFICIENT_OUTLIERS *␣
 ↪std_dev) consideram un numar ca fiind Outlier
```

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

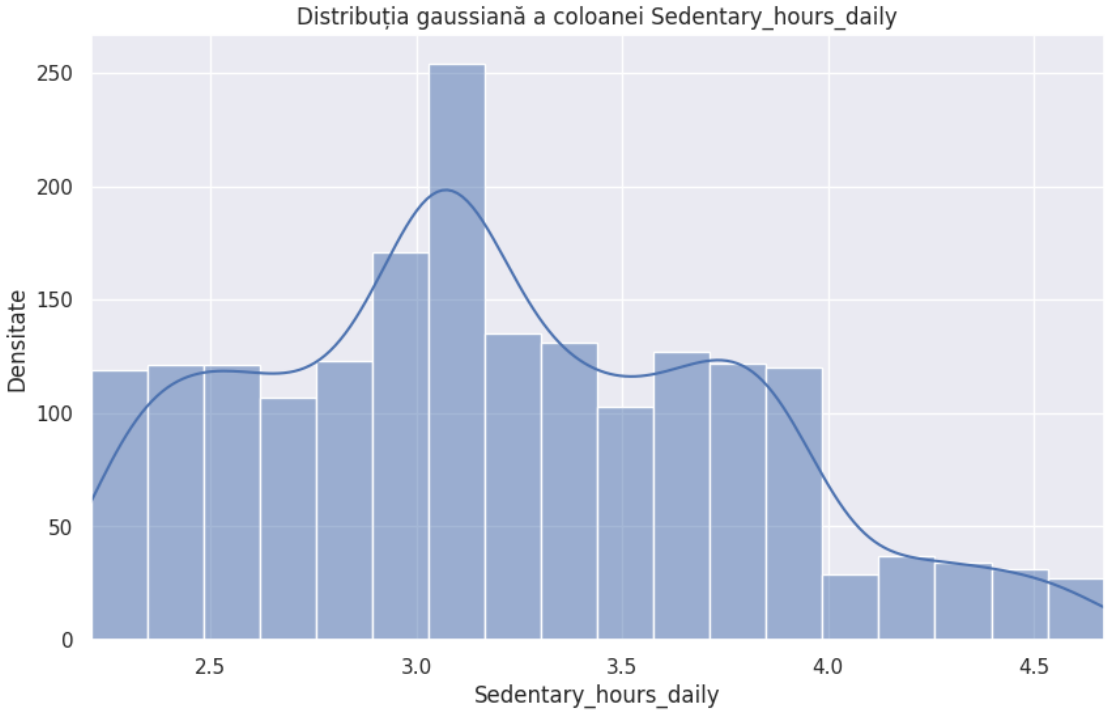
```
Outlieri pentru coloana 'Regular_fiber_diet': [2739.0]
Diferența față de std dev pentru outlieri: [2676.5603825004314]
Limita pentru coloana 'Regular_fiber_diet' este valoarea 187.3188524987052

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
```

[30]:
```
for column in numeric_attributes:
    plot_numeric_gaussian_distribution(df_copy, column)
```

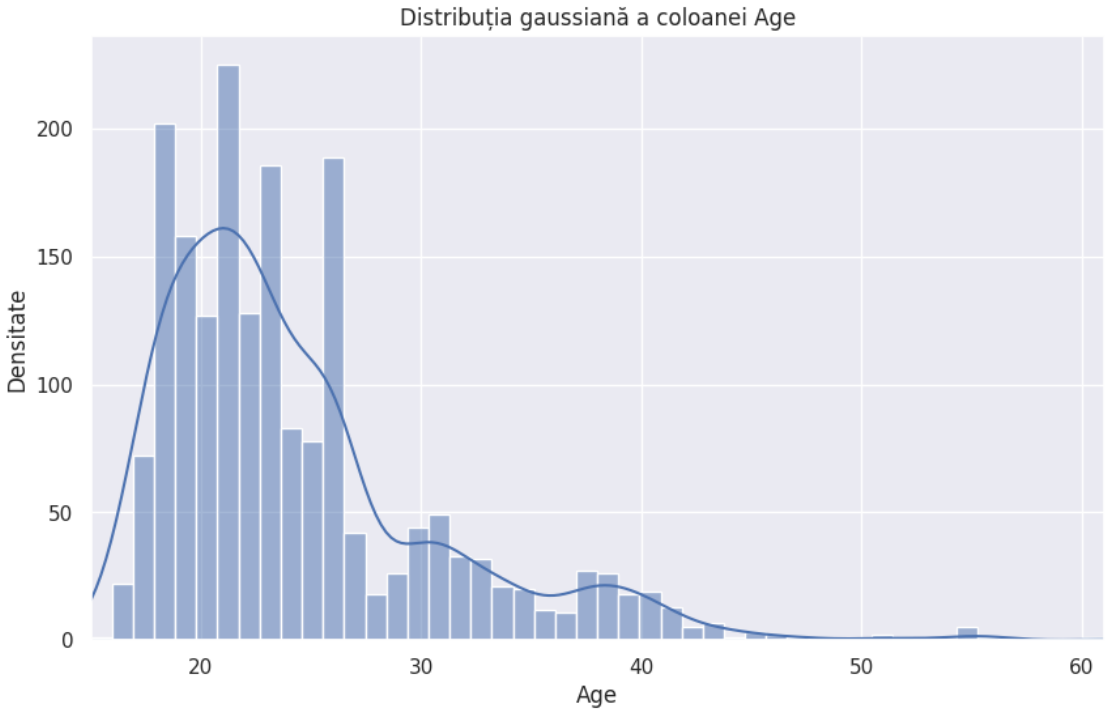
```
Regular_fiber_diet has min=1.0 and max=3.0
```



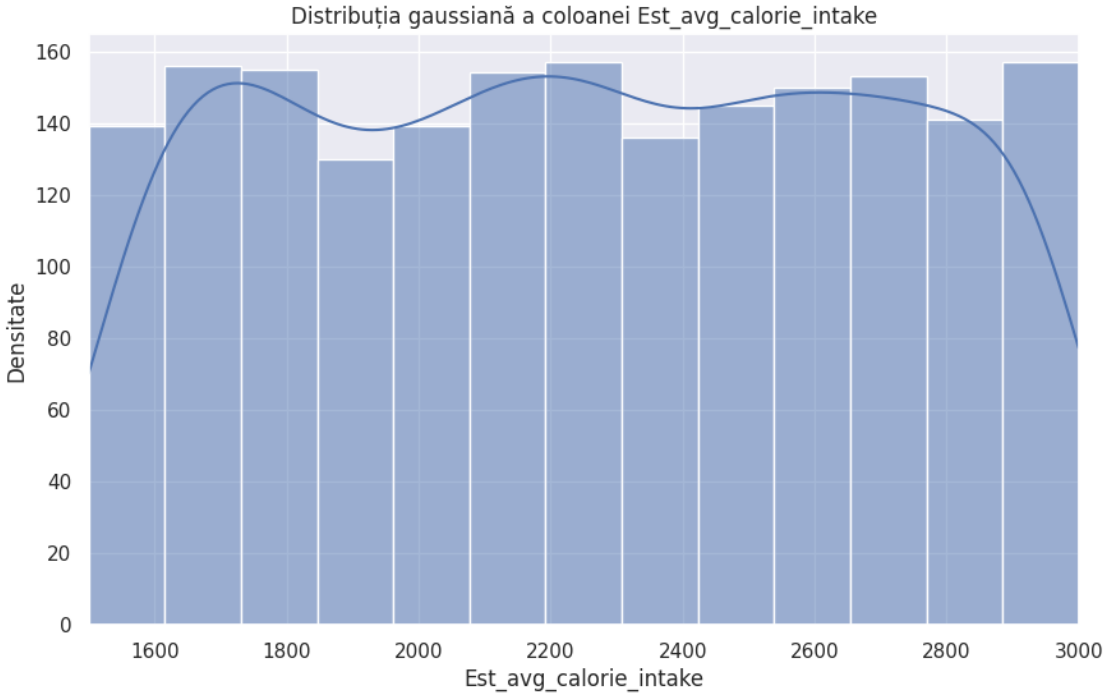

```
Sedentary_hours_daily has min=2.21 and max=4.67
```



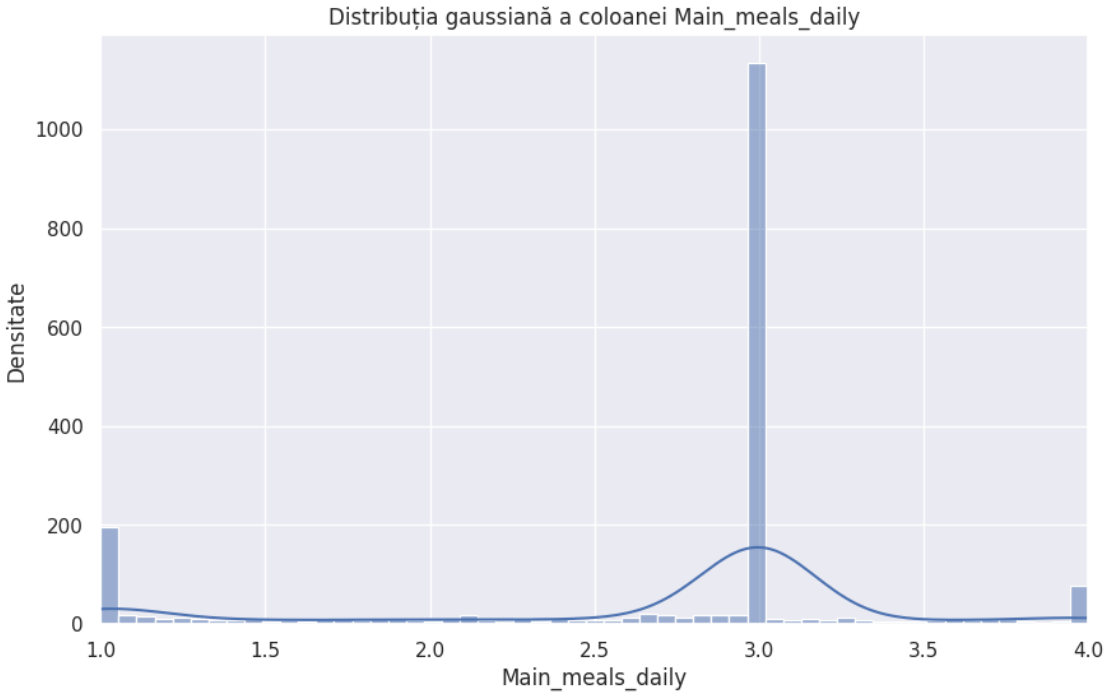

```
Age has min=15.0 and max=61.0
```



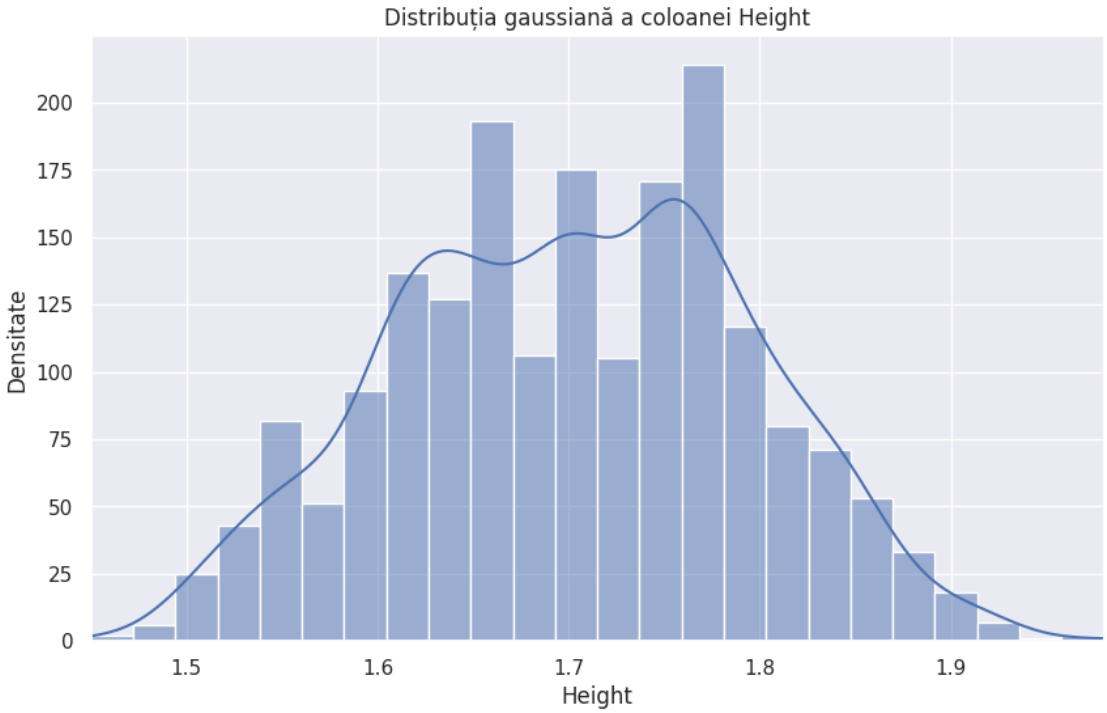
`Est_avg_calorie_intake has min=1500 and max=3000`



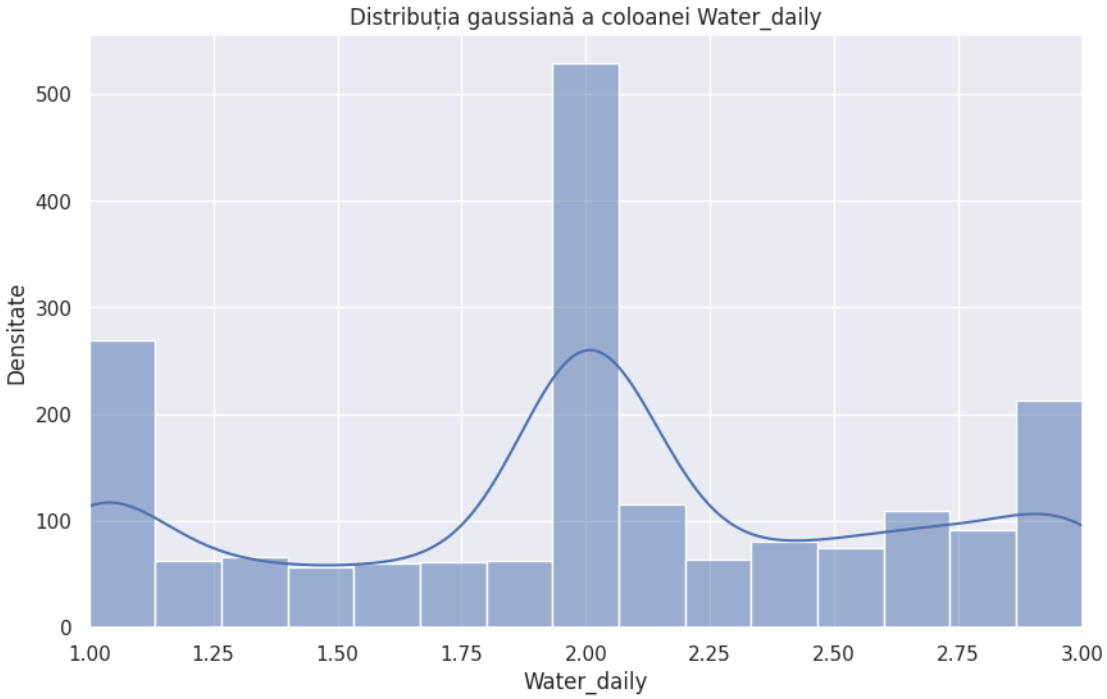
`Main_meals_daily has min=1.0 and max=4.0`



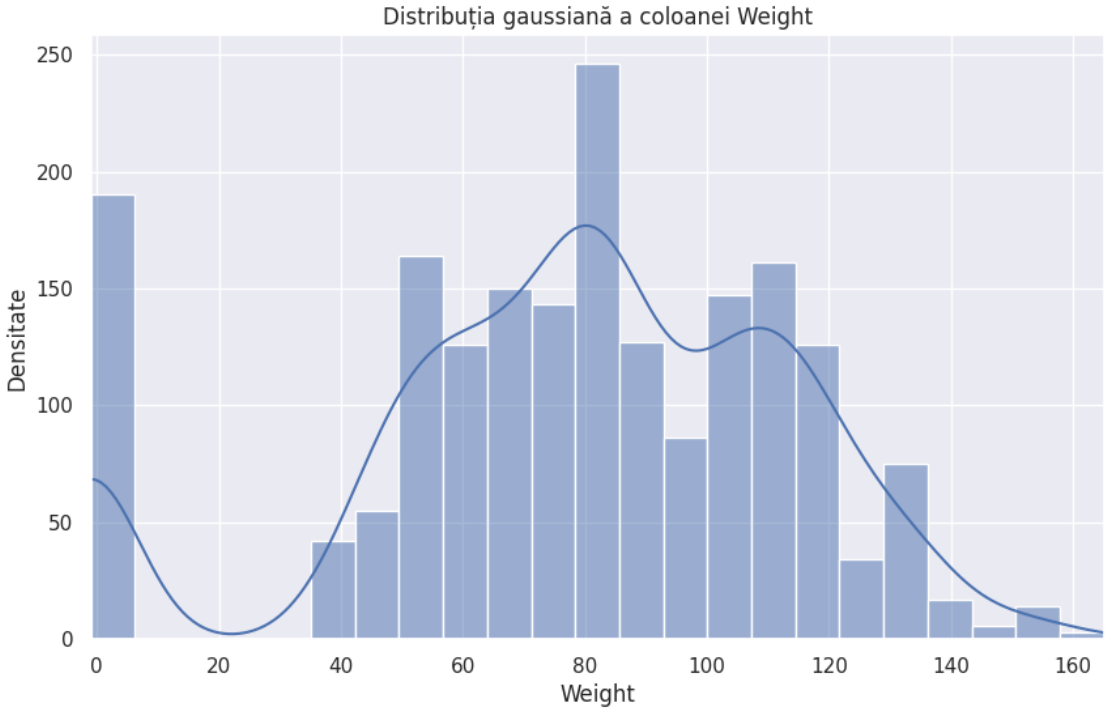
```
Height has min=1.45 and max=1.98
```



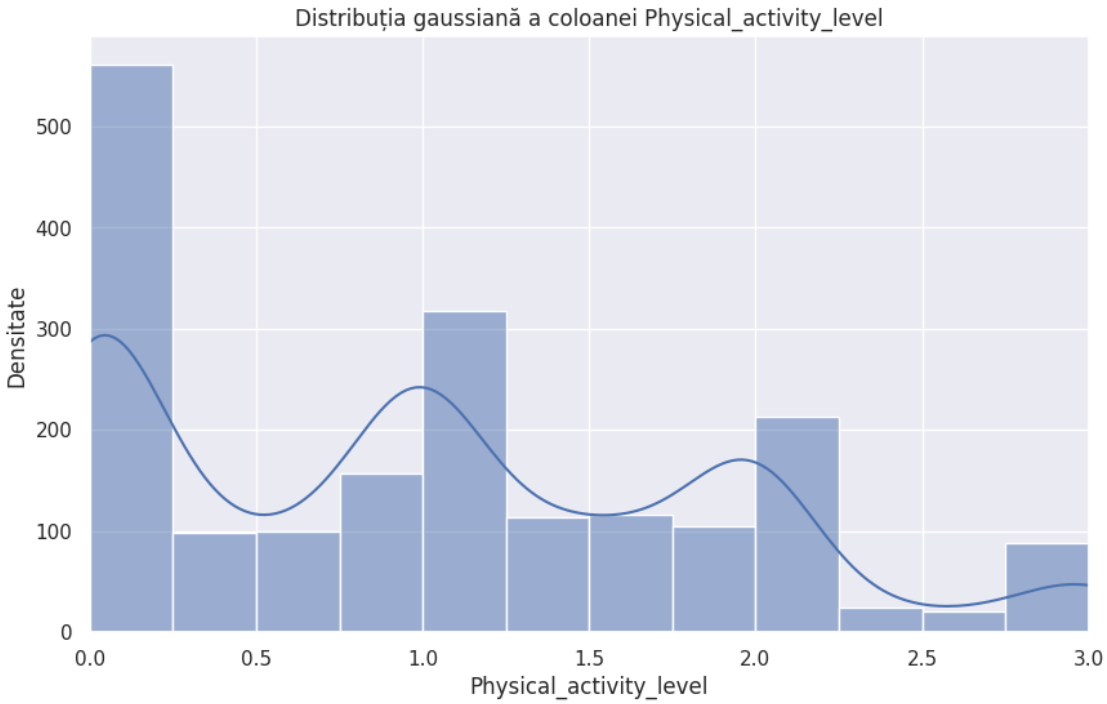

```
Water_daily has min=1.0 and max=3.0
```



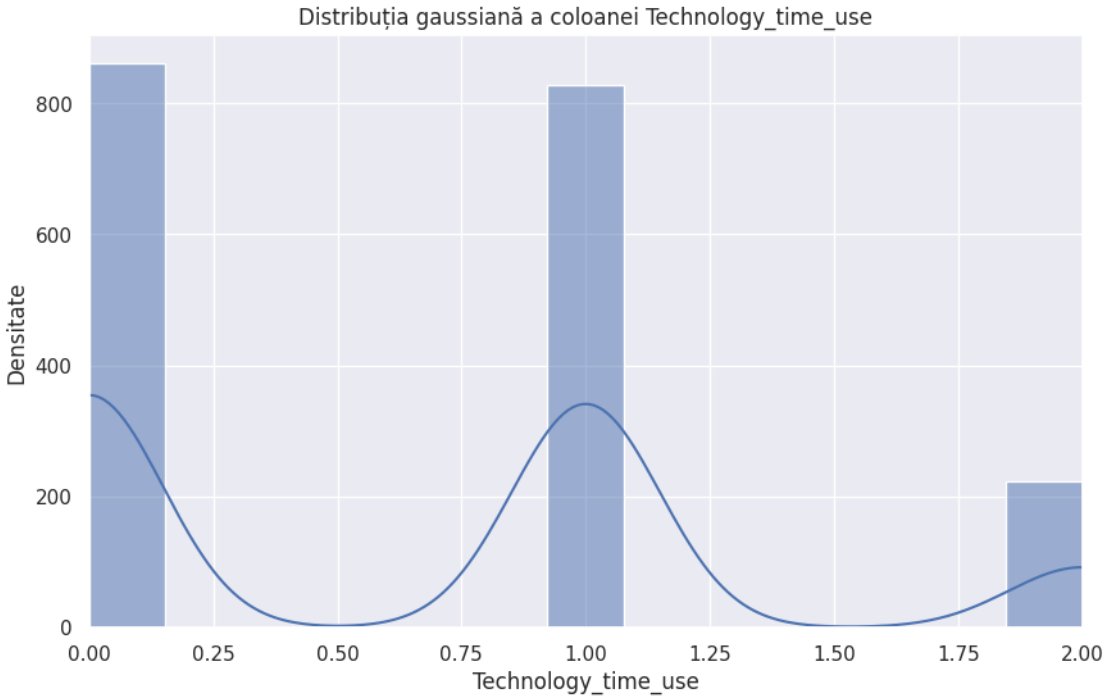

```
Weight has min=-1.0 and max=165.057269
```



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



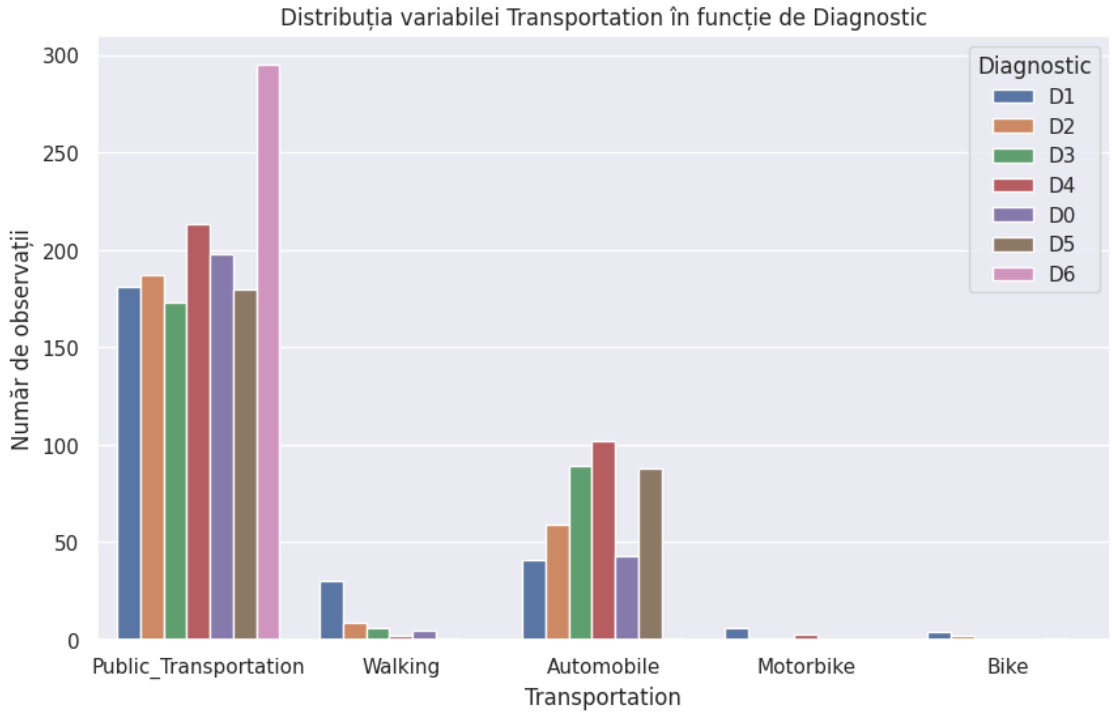

```
Technology_time_use has min=0 and max=2
```

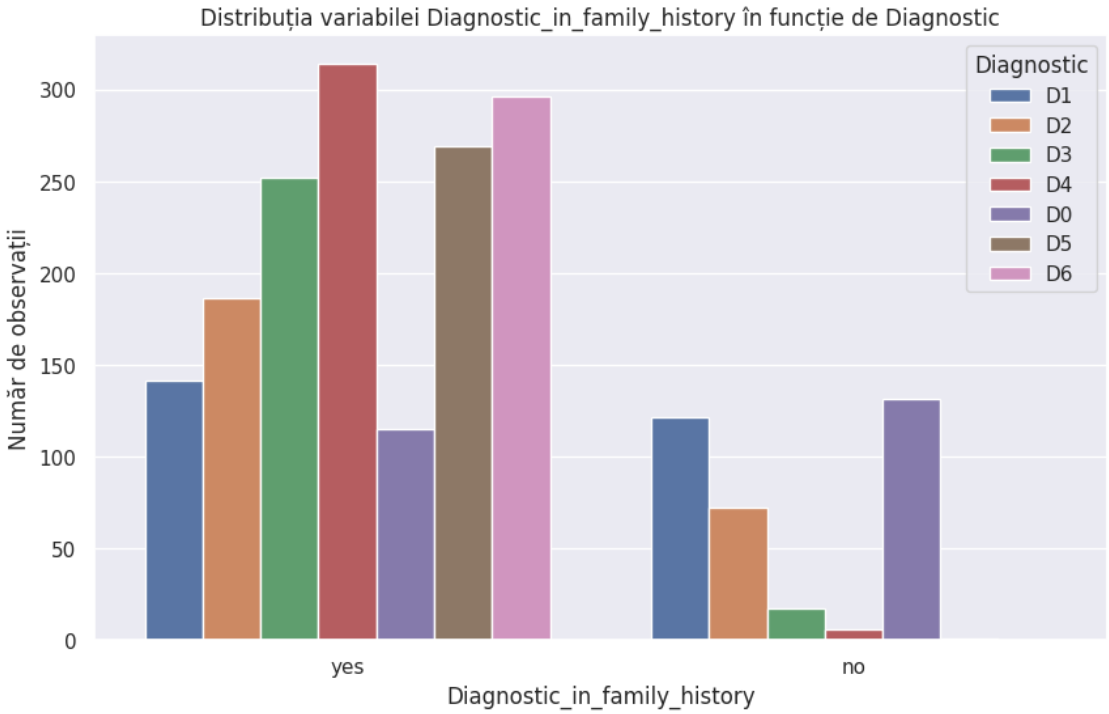


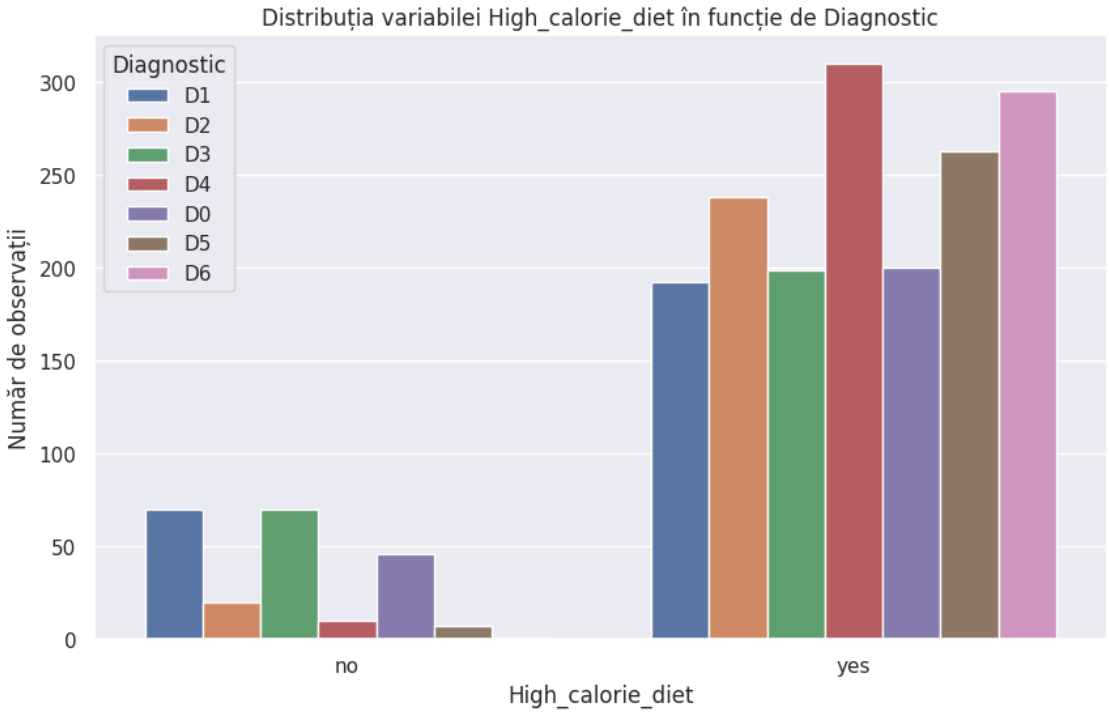
Ilustram legatura dintre fiecare atribut si variabila tinta

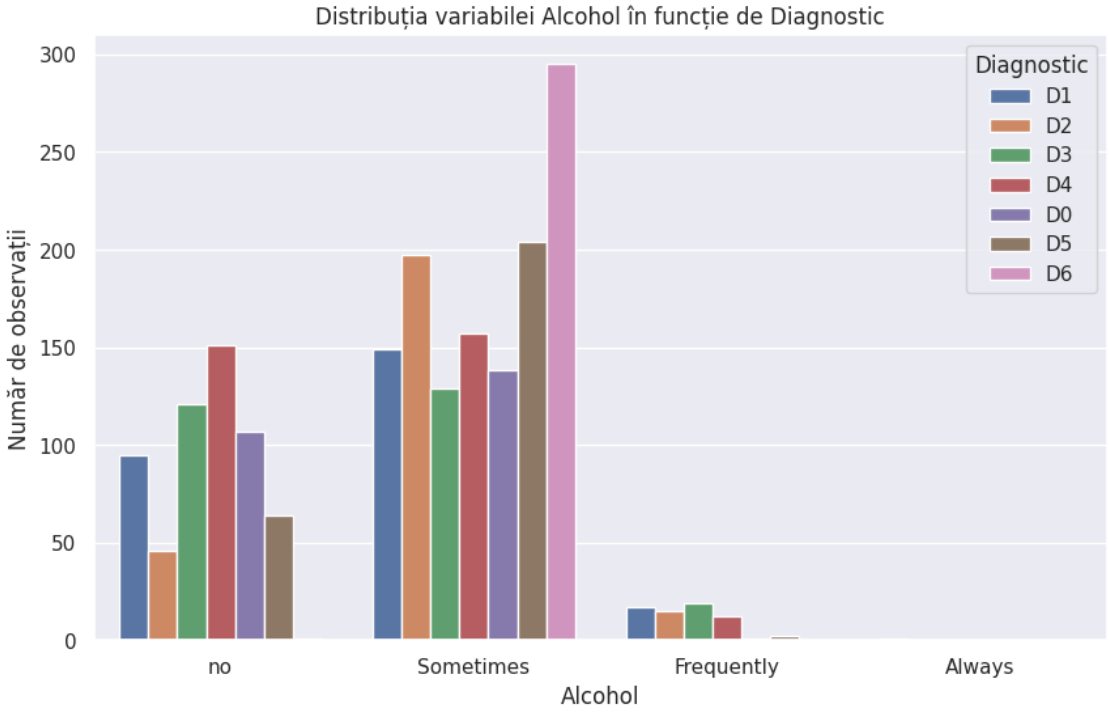
[125]:
```
# Afișează distribuția pentru o variabilă categorică în funcție de variabila
 ↪țintă
def plot_categorical_distribution(df, target_column, categorical_column):
    plt.figure(figsize=(10, 6))
    sns.countplot(data=df, x=categorical_column, hue=target_column)
    plt.title(f'Distribuția variabilei {categorical_column} în funcție de
 ↪{target_column}')
    plt.xlabel(categorical_column)
    plt.ylabel('Număr de observații')
    plt.legend(title=target_column)
    plt.show()
```

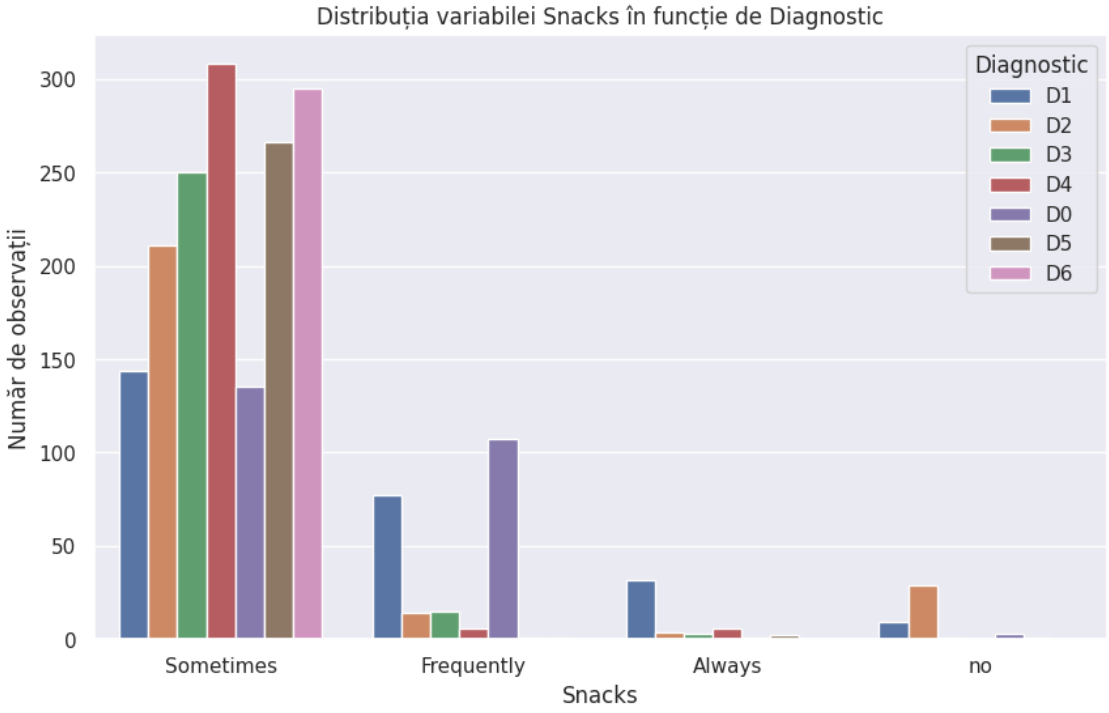
[126]:
```
# Apelați funcția pentru fiecare variabilă categorică
for column in categorical_attributes:
    plot_categorical_distribution(df, "Diagnostic", column)
```

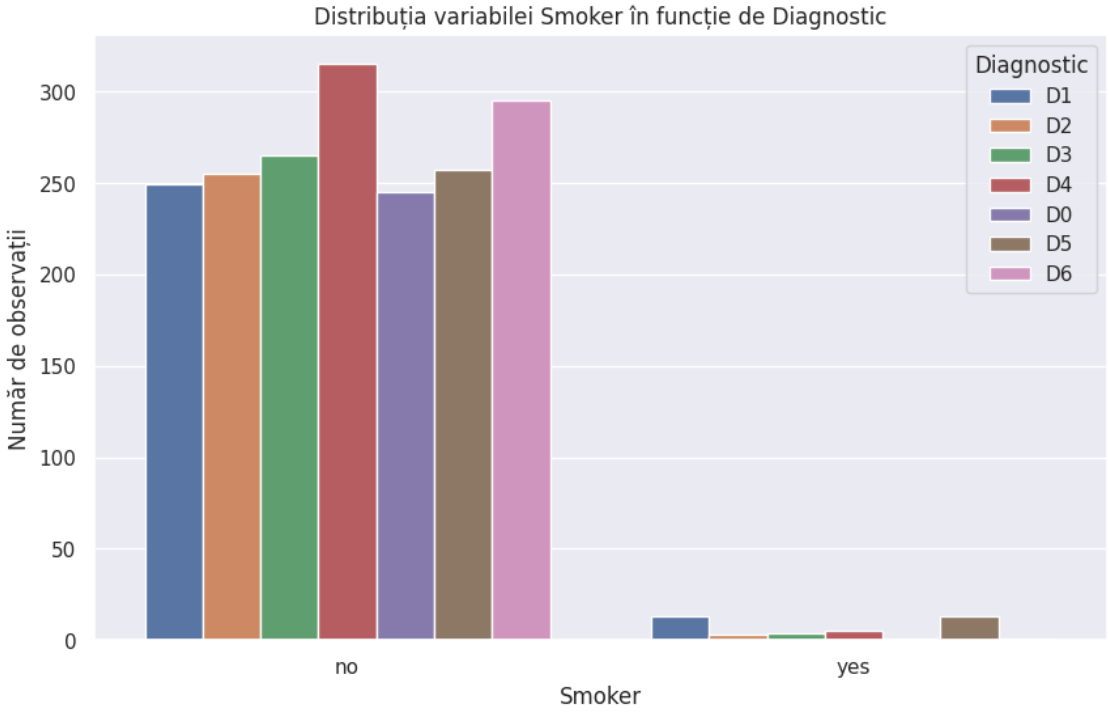


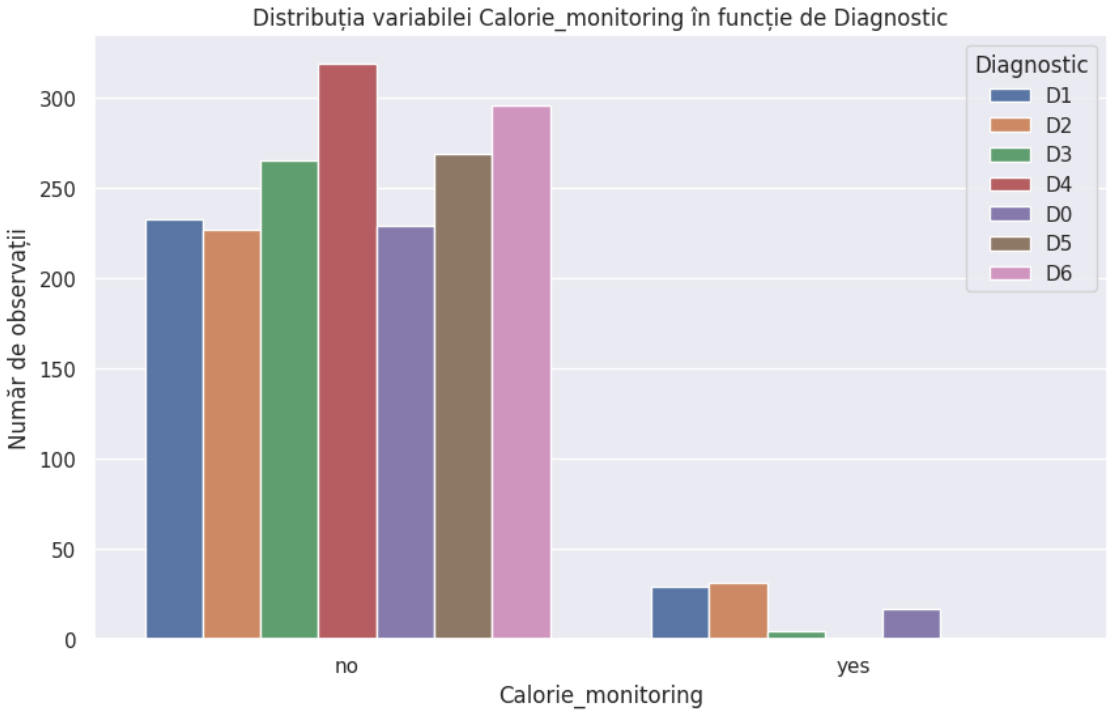

Distribuția variabilei Diagnostic_in_family_history în funcție de Diagnostic
Diagnostic
D1
D2
D3
D4
D0
D5
D6
Număr de observații
0
50
100
150
200
250
300
yes
no
Diagnostic_in_family_history

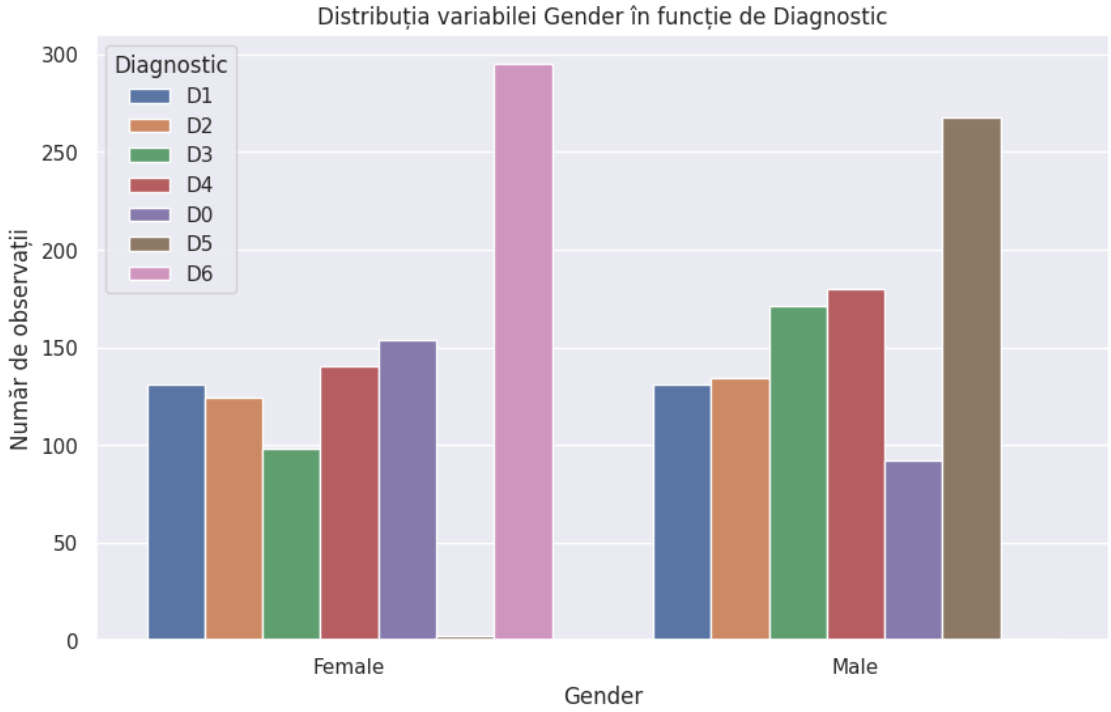

Distribuția variabilei High_calorie_diet în funcție de Diagnostic
Diagnostic
D1
D2
D3
D4
D0
D5
D6
Număr de observații
0
50
100
150
200
250
300
no
yes
High_calorie_diet


Distribuția variabilei Alcohol în funcție de Diagnostic
Număr de observații
300
250
200
150
100
50
0
no
Sometimes
Frequently
Always
Alcohol
Diagnostic
D1
D2
D3
D4
D0
D5
D6

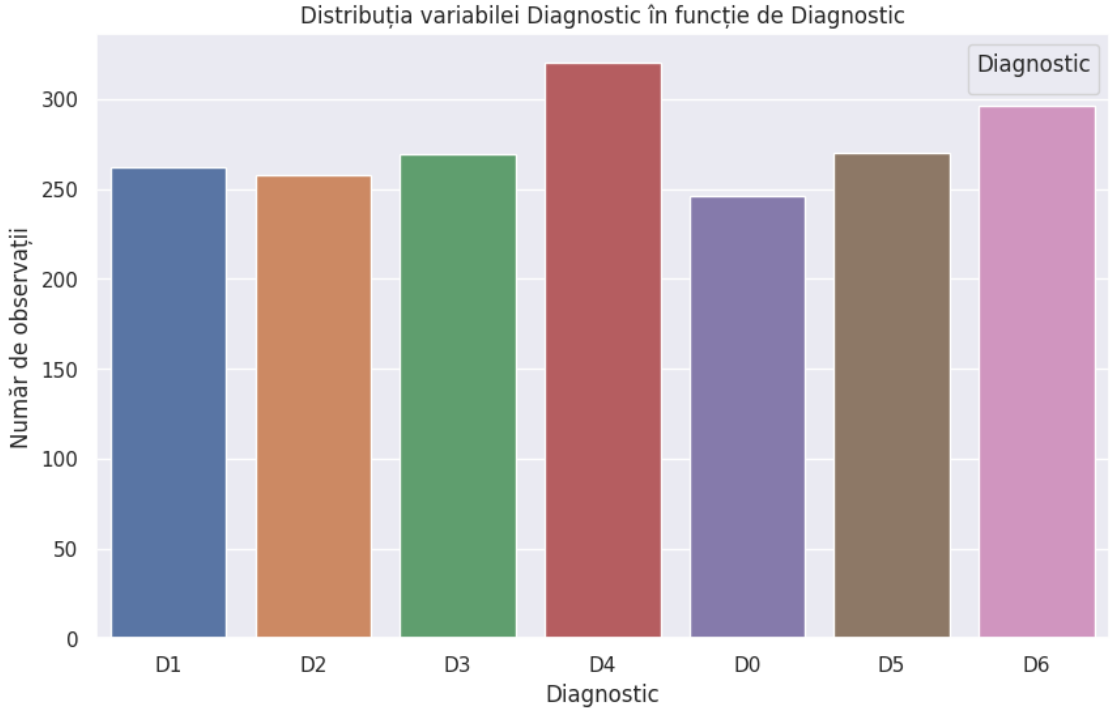

Distribuția variabilei Snacks în funcție de Diagnostic
Număr de observații
300
250
200
150
100
50
0
Sometimes
Frequently
Always
no
Snacks
Diagnostic
D1
D2
D3
D4
D0
D5
D6


Distribuția variabilei Smoker în funcție de Diagnostic
Diagnostic
D1
D2
D3
D4
D0
D5
D6
Număr de observații
300
250
200
150
100
50
0
no
yes
Smoker


Distribuția variabilei Calorie_monitoring în funcție de Diagnostic
Diagnostic
D1
D2
D3
D4
D0
D5
D6
Număr de observații
300
250
200
150
100
50
0
no
yes
Calorie_monitoring

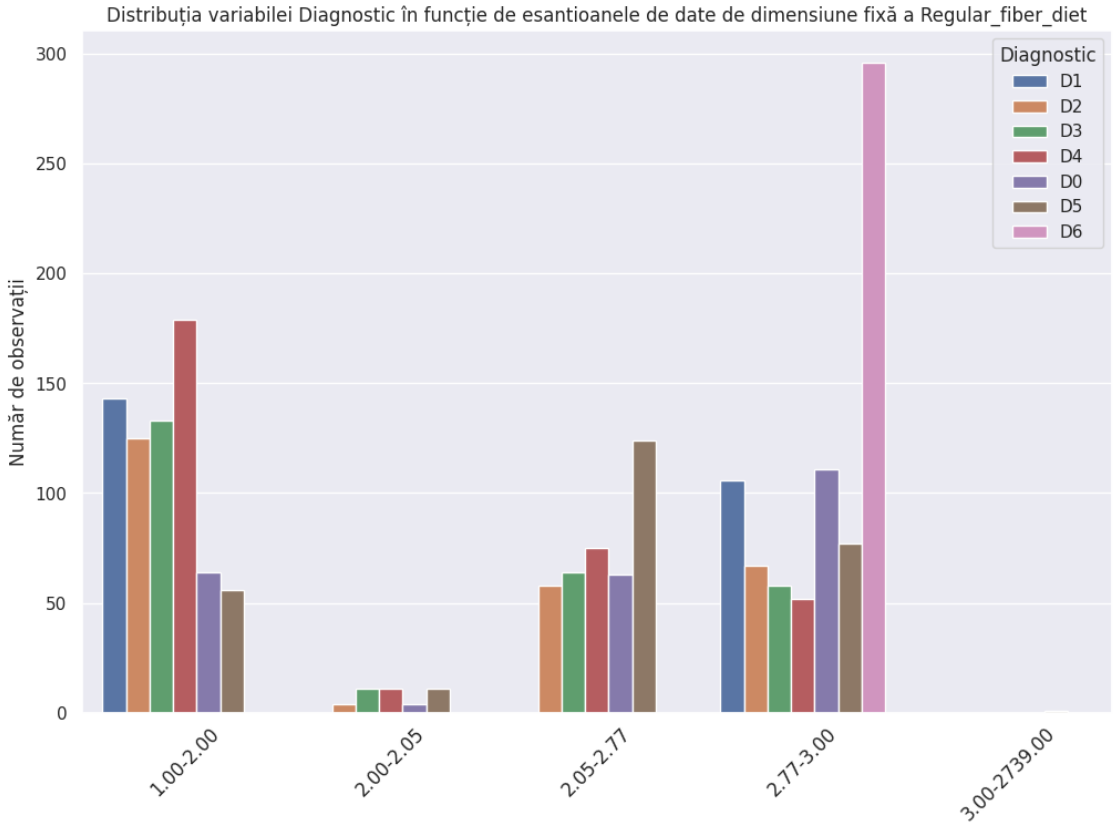


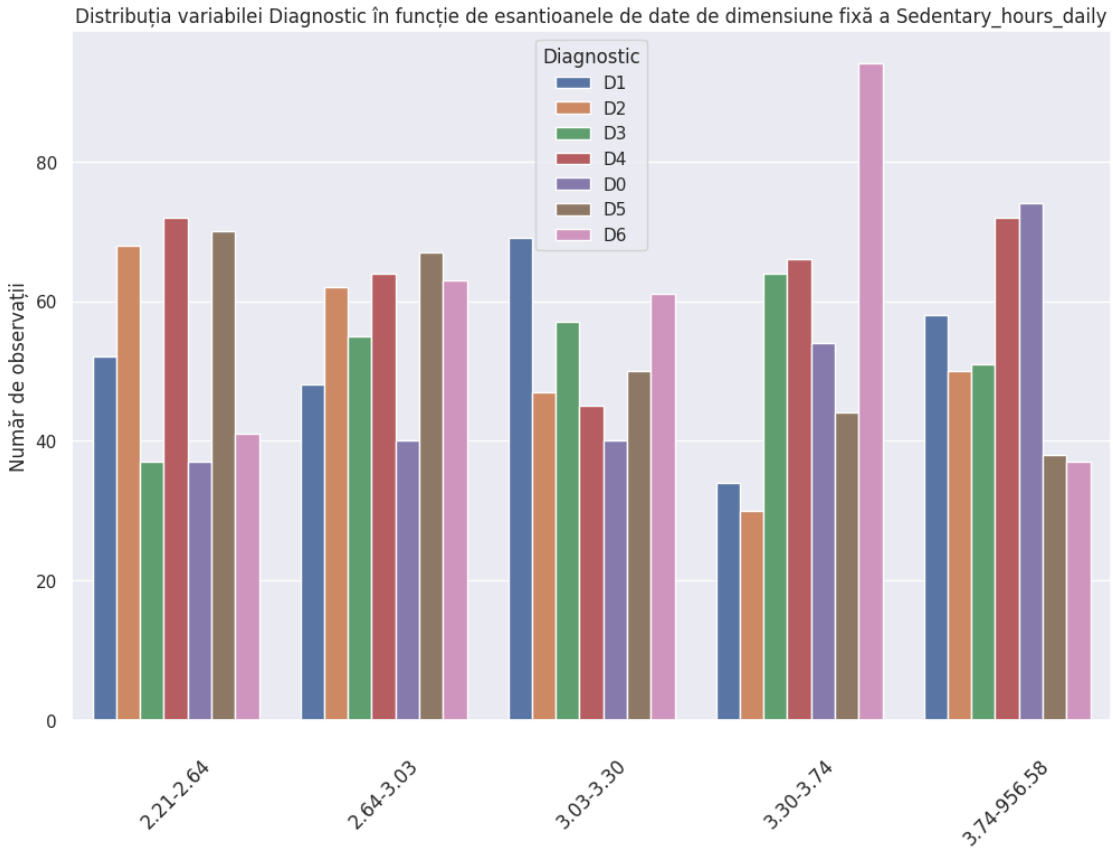

```
No artists with labels found to put in legend.  Note that artists whose label
start with an underscore are ignored when legend() is called with no argument.
```

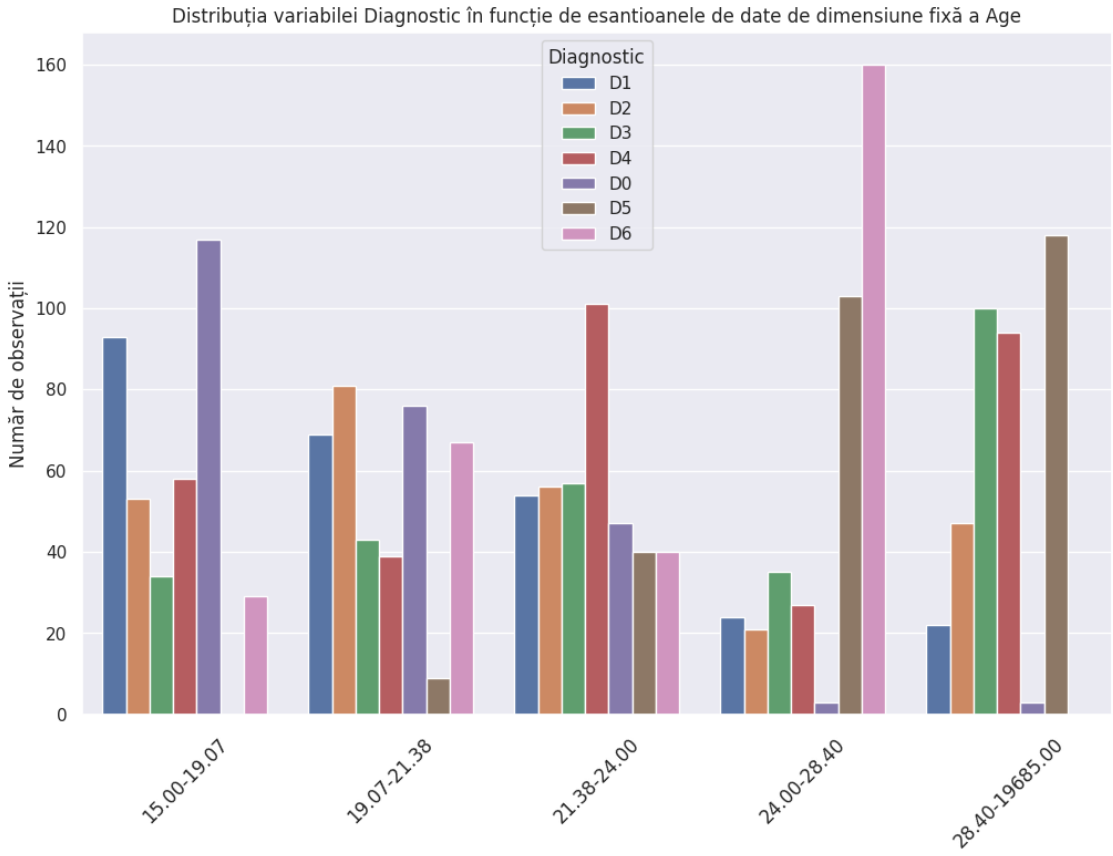


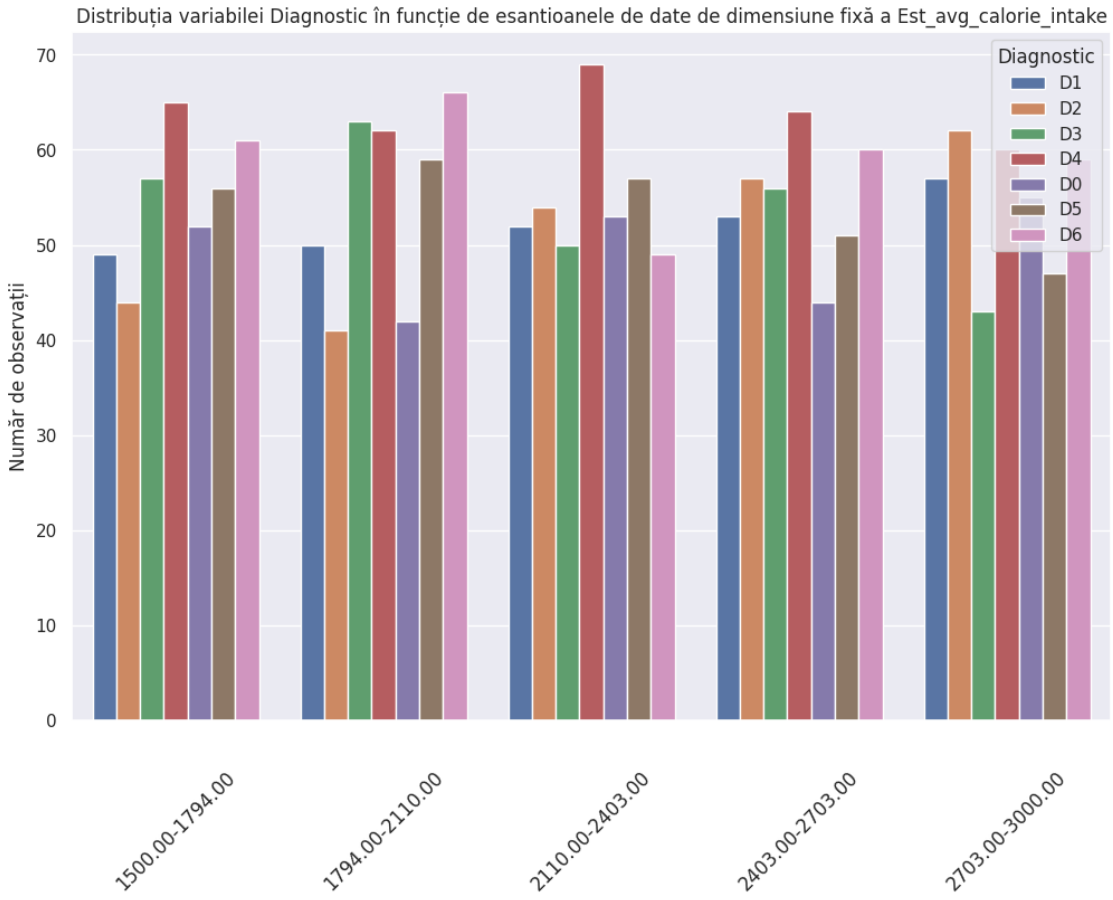
[82]:
```
# Afișează distribuția variabilei țintă în funcție de esantioanele de date de␣
↪dimensiune fixă
def plot_target_distribution_by_fixed_samples(df, target_column,␣
↪numeric_column, num_samples=5):
    # Calculează percentile pentru esantionul de fiecare 20%
    percentiles = np.arange(0, 101, 20)

    # Calculează valorile percentile
    percentile_values = np.percentile(df[numeric_column], percentiles)

    # Gruparea datelor în esantioanele definite de percentile_values
    df['Sample'] = pd.cut(df[numeric_column], bins=percentile_values,␣
↪labels=range(1, num_samples + 1), duplicates='drop')

    # Trasează distribuția pentru fiecare esantion de dimensiune fixă
    plt.figure(figsize=(12, 8))
    ax = sns.countplot(data=df, x='Sample', hue=target_column)
    plt.title(f'Distribuția variabilei {target_column} în funcție de␣
↪esantioanele de date de dimensiune fixă a {numeric_column}')
    plt.xlabel('')  # Elimină eticheta axei x
    plt.ylabel('Număr de observații')
    plt.legend(title=target_column)

    # Ascunde etichetele axei x
    ax.set_xticklabels([])

    # Adaugă etichete pentru intervalul de valori de sub axa x
    for i, (p, q) in enumerate(zip(percentile_values[:-1], percentile_values[1:
↪])):
        plt.text(i, -5, f"{p:.2f}-{q:.2f}", rotation=45, ha='center', va='top')

    plt.show()
```

[83]:
```
for column in numeric_attributes:
    plot_target_distribution_by_fixed_samples(df, "Diagnostic", column)
```


Distribuția variabilei Diagnostic în funcție de esantioanele de date de dimensiune fixă a Regular_fiber_diet
Diagnostic
D1
D2
D3
D4
D0
D5
D6
Număr de observații
300
250
200
150
100
50
0
1.00-2.00
2.00-2.05
2.05-2.77
2.77-3.00
3.00-2739.00


Distribuția variabilei Diagnostic în funcție de esantioanele de date de dimensiune fixă a Sedentary_hours_daily
Diagnostic
D1
D2
D3
D4
D0
D5
D6
Număr de observații
0
20
40
60
80
2.21-2.64
2.64-3.03
3.03-3.30
3.30-3.74
3.74-956.58


Distribuția variabilei Diagnostic în funcție de esantioanele de date de dimensiune fixă a Age
Diagnostic
D1
D2
D3
D4
D0
D5
D6
Număr de observații
0
20
40
60
80
100
120
140
160
15.00-19.07
19.07-21.38
21.38-24.00
24.00-28.40
28.40-19685.00



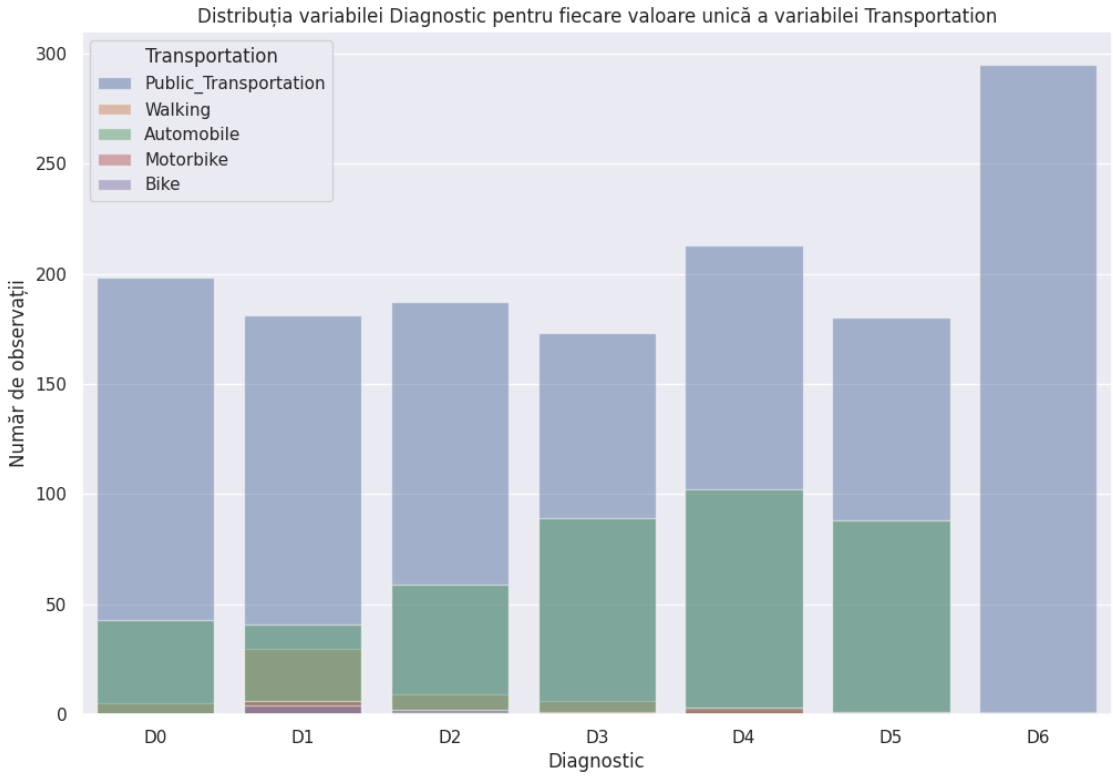

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[83], line 2
      1 for column in numeric_attributes:
----> 2     plot_target_distribution_by_fixed_samples(df, "Diagnostic", column)

Cell In[82], line 10, in plot_target_distribution_by_fixed_samples(df,␣
 ↪target_column, numeric_column, num_samples)
      7 percentile_values = np.percentile(df[numeric_column], percentiles)
      9 # Gruparea datelor în esantioanele definite de percentile_values
---> 10 df['Sample'] =␣
 ↪pd.cut(df[numeric_column], bins=percentile_values, labels=range(1, num_sample + 1), duplic
     12 # Trasează distribuția pentru fiecare esantion de dimensiune fixă
     13 plt.figure(figsize=(12, 8))

File ~/.local/lib/python3.10/site-packages/pandas/core/reshape/tile.py:257, in␣
 ↪cut(x, bins, right, labels, retbins, precision, include_lowest, duplicates,␣
 ↪ordered)
    254     if not bins.is_monotonic_increasing:
```

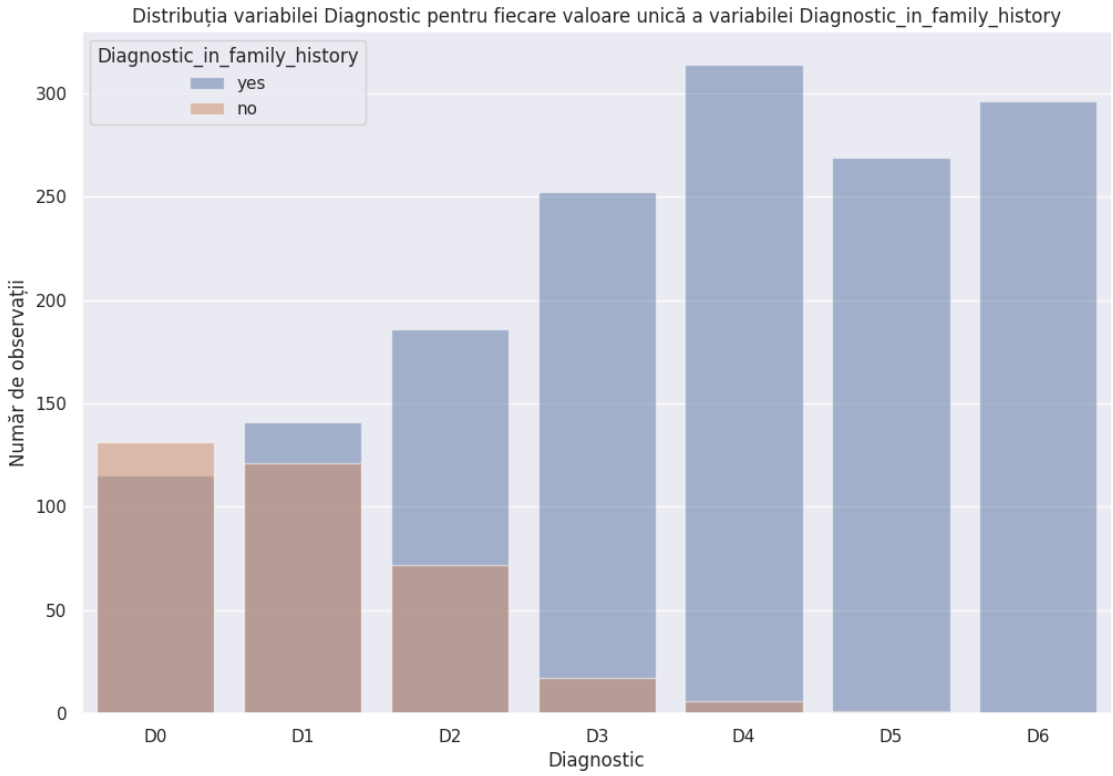
```
    255         raise ValueError("bins must increase monotonically.")
--> 257 fac, bins = _bins_to_cuts(
    258     x_idx,
    259     bins,
    260     right=right,
    261     labels=labels,
    262     precision=precision,
    263     include_lowest=include_lowest,
    264     duplicates=duplicates,
    265     ordered=ordered,
    266 )
    268 return _postprocess_for_cut(fac, bins, retbins, original)

File ~/.local/lib/python3.10/site-packages/pandas/core/reshape/tile.py:493, in␣
 ↪_bins_to_cuts(x_idx, bins, right, labels, precision, include_lowest,␣
 ↪duplicates, ordered)
    491 else:
    492     if len(labels) != len(bins) - 1:
--> 493         raise ValueError(
    494             "Bin labels must be one fewer than the number of bin edges"
    495         )
    497 if not isinstance(getattr(labels, "dtype", None), CategoricalDtype):
    498     labels = Categorical(
    499         labels,
    500         categories=labels if len(set(labels)) == len(labels) else None,
    501         ordered=ordered,
    502     )

ValueError: Bin labels must be one fewer than the number of bin edges
```

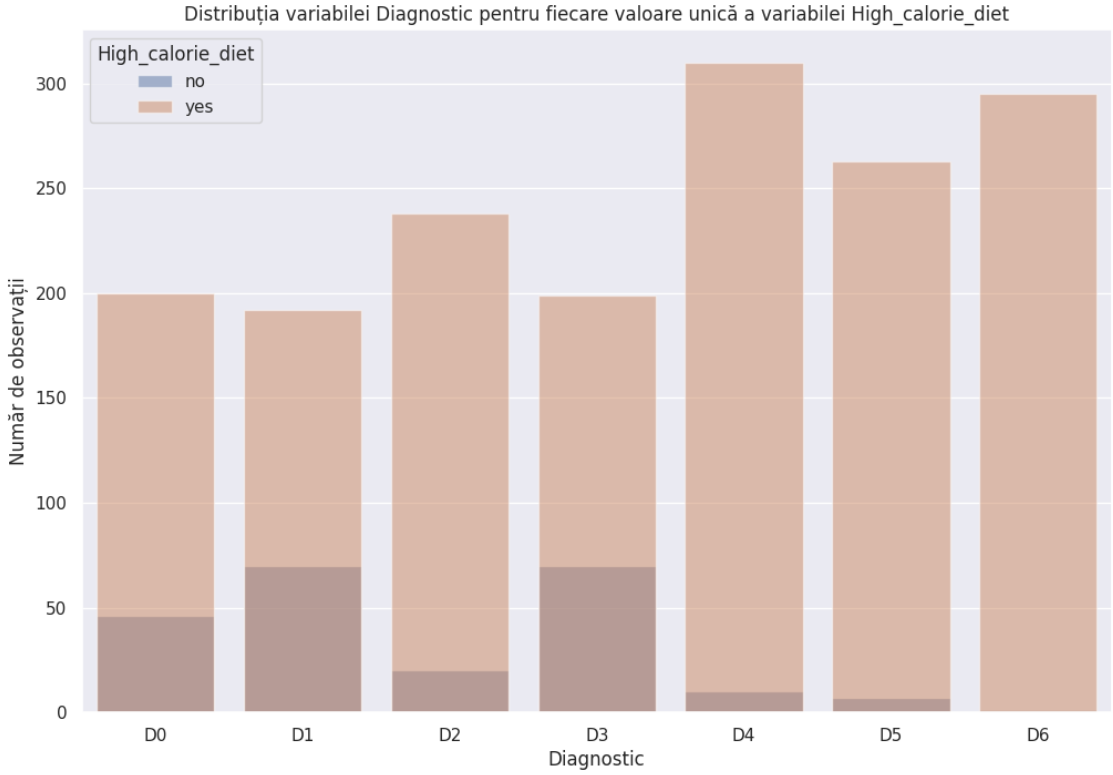
[112]:
```
# Definează ordinea dorită a categoriilor
desired_order = ['D0', 'D1', 'D2', 'D3', 'D4', 'D5', 'D6']

# Afișează distribuția variabilei țintă pentru fiecare valoare unică a␣
 ↪variabilei categorice, toate graficele în aceeași figură
def plot_target_distribution_by_category(df, target_column, categorical_column):
    unique_categories = df[categorical_column].unique()

    # Creează o figură și axă pentru trasearea graficelor
    plt.figure(figsize=(12, 8))

    # Trasează distribuția pentru fiecare valoare unică a variabilei categorice
    for category in unique_categories:
        subset = df[df[categorical_column] == category]
        sns.countplot(data=subset, x=target_column, label=str(category),␣
 ↪alpha=0.5, order=desired_order)
```

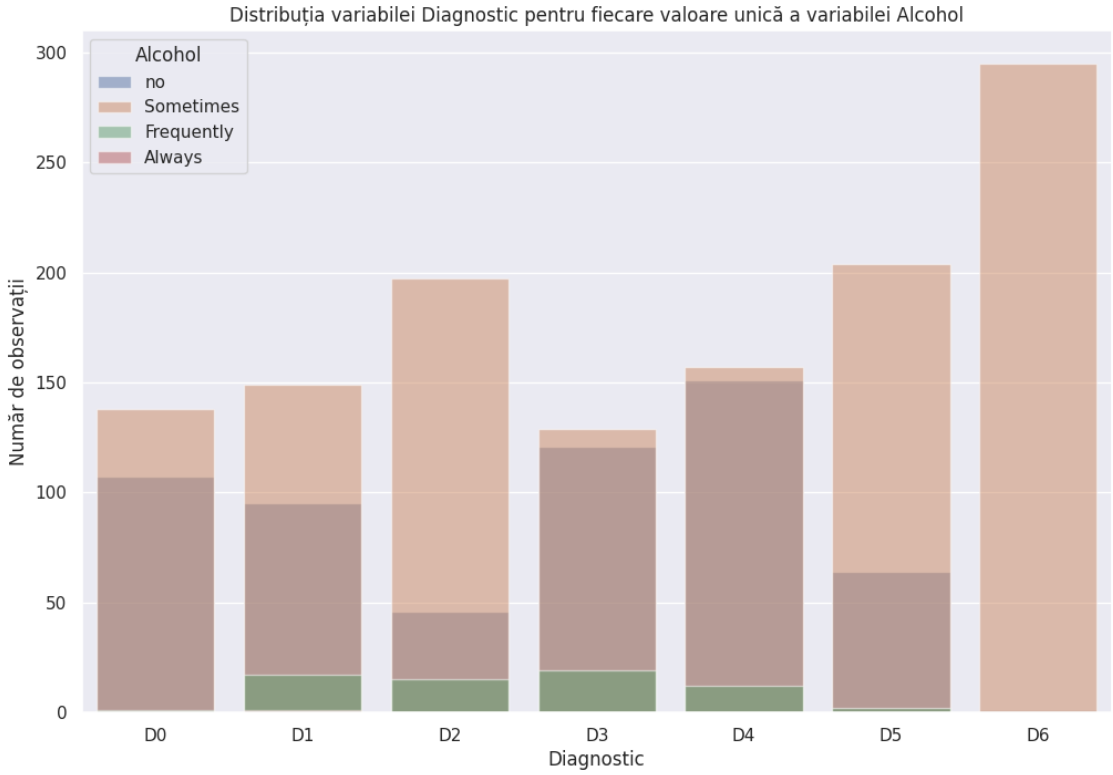
```
    plt.title(f'Distribuția variabilei {target_column} pentru fiecare valoare␣
↪unică a variabilei {categorical_column}')
    plt.xlabel(target_column)
    plt.ylabel('Număr de observații')
    plt.legend(title=categorical_column)
    plt.show()
```

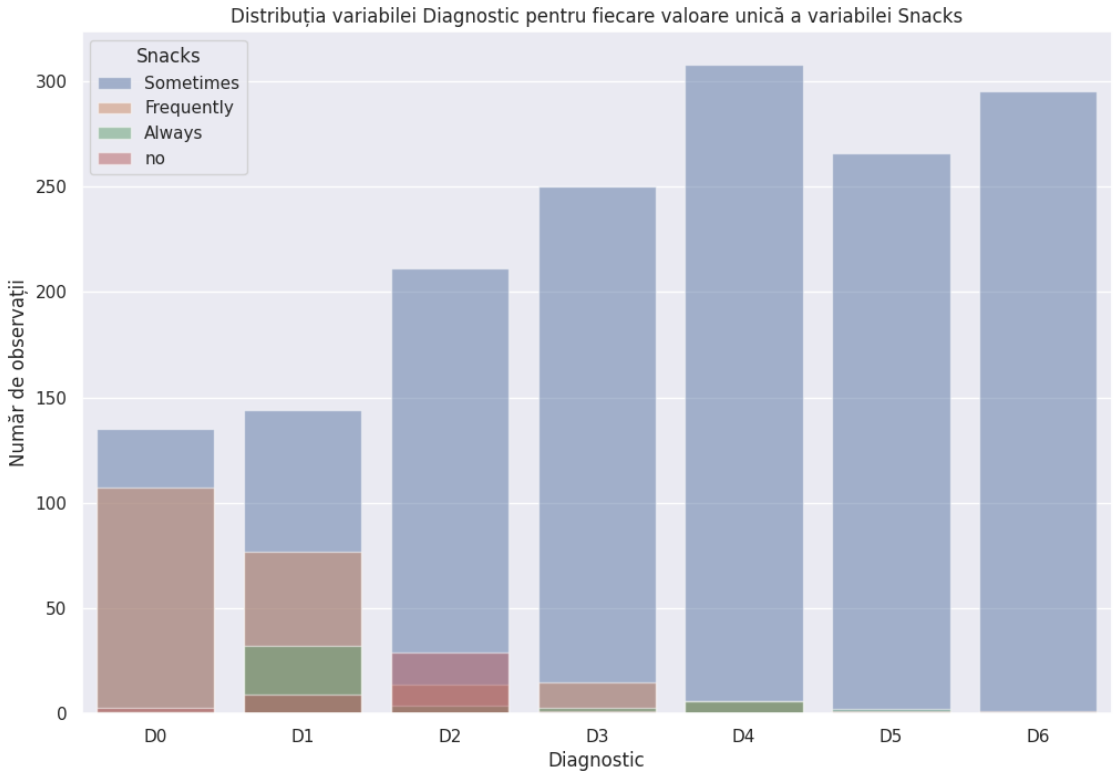
[113]:
```
# Apelați funcția pentru fiecare atribut categoric
for column in categorical_attributes:
    plot_target_distribution_by_category(df, "Diagnostic", column)
```

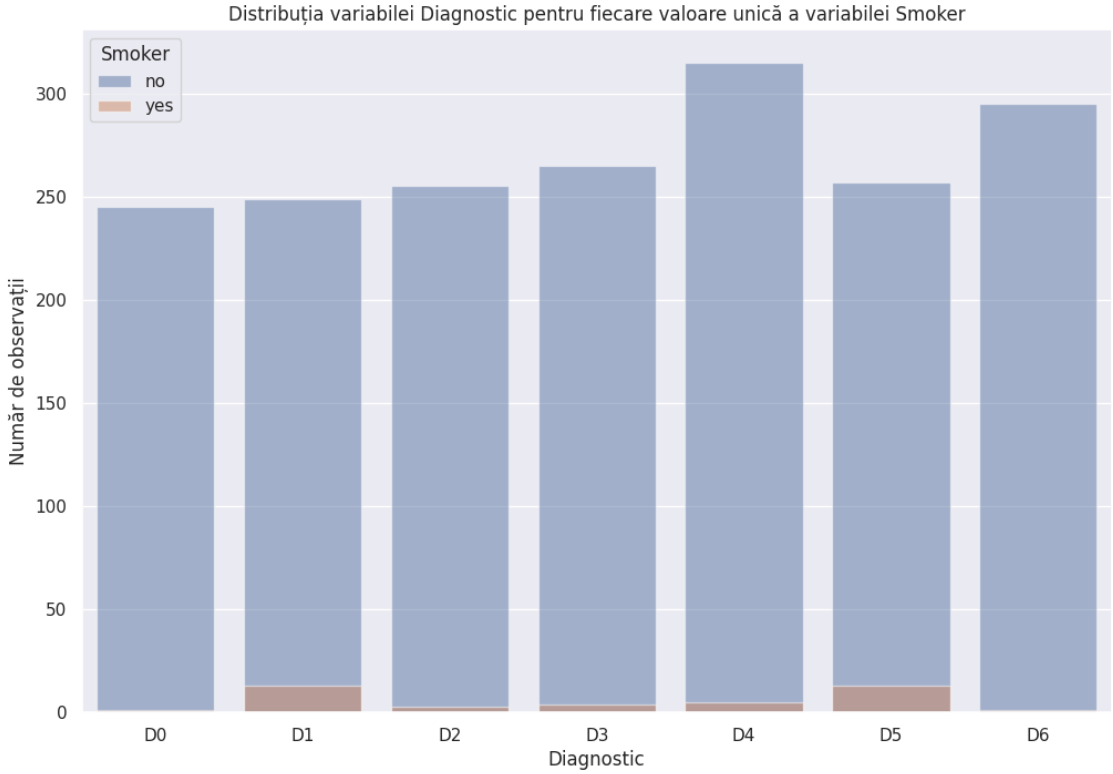


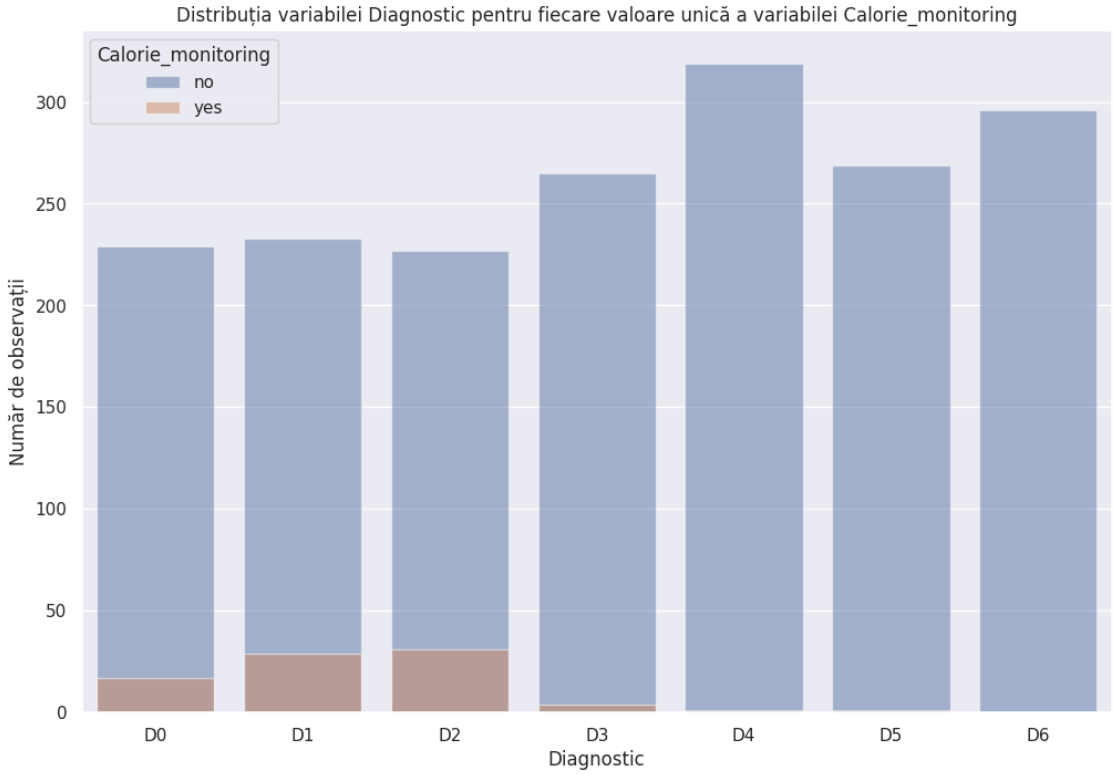

Distribuția variabilei Diagnostic pentru fiecare valoare unică a variabilei Diagnostic_in_family_history
Diagnostic_in_family_history
yes
no
Număr de observații
0
50
100
150
200
250
300
D0
D1
D2
D3
D4
D5
D6
Diagnostic

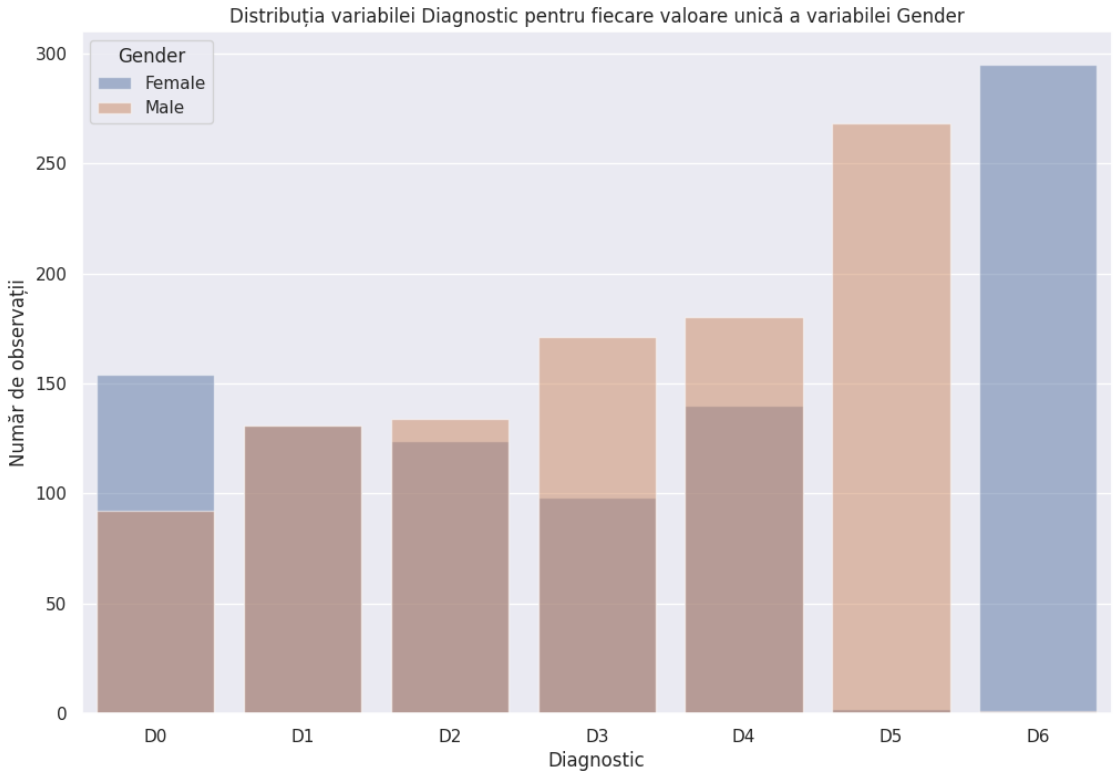

Distribuția variabilei Diagnostic pentru fiecare valoare unică a variabilei High_calorie_diet
High_calorie_diet
no
yes
Număr de observații
0
50
100
150
200
250
300
D0
D1
D2
D3
D4
D5
D6
Diagnostic

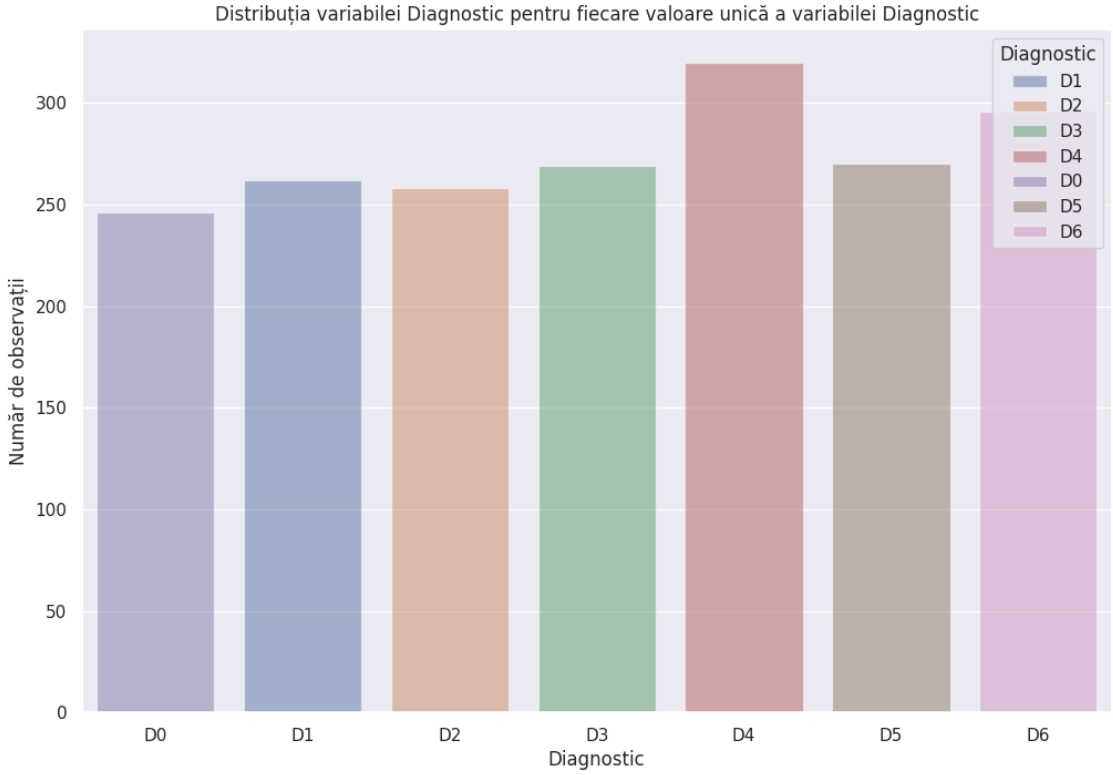

Distribuția variabilei Diagnostic pentru fiecare valoare unică a variabilei Alcohol
Alcohol
no
Sometimes
Frequently
Always
Număr de observații
0
50
100
150
200
250
300
D0
D1
D2
D3
D4
D5
D6
Diagnostic


Distribuția variabilei Diagnostic pentru fiecare valoare unică a variabilei Snacks
Snacks
Sometimes
Frequently
Always
no
Număr de observații
0
50
100
150
200
250
300
D0
D1
D2
D3
D4
D5
D6
Diagnostic


Distribuția variabilei Diagnostic pentru fiecare valoare unică a variabilei Smoker
Smoker
no
yes
Număr de observații
0
50
100
150
200
250
300
D0
D1
D2
D3
D4
D5
D6
Diagnostic


Distribuția variabilei Diagnostic pentru fiecare valoare unică a variabilei Calorie_monitoring
Calorie_monitoring
no
yes
Număr de observații
0
50
100
150
200
250
300
D0
D1
D2
D3
D4
D5
D6
Diagnostic


Distribuția variabilei Diagnostic pentru fiecare valoare unică a variabilei Gender
Gender
Female
Male
Număr de observații
0
50
100
150
200
250
300
D0
D1
D2
D3
D4
D5
D6
Diagnostic


Distribuția variabilei Diagnostic pentru fiecare valoare unică a variabilei Diagnostic
Diagnostic
D1
D2
D3
D4
D0
D5
D6
Număr de observații
0
50
100
150
200
250
300
D0
D1
D2
D3
D4
D5
D6
Diagnostic

4 2. Vizualizarea datelor

Pentru analiza atributelor numerice:

[31]:
```
def calculate_mad(series):
    mean = series.mean()
    absolute_deviations = abs(series - mean)
    mad = absolute_deviations.mean()
    return mad
```

[32]:
```
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:

[33]:
```
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:

[34]:
```
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:

[35]:
```
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

[36]:
```
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()
```

```
Analiză pentru coloana numerică 'Regular_fiber_diet':
Medie: 3.8449373862571576
Abaterea standard: 62.4396174995684
Abaterea medie absolută: 2.8476367127680824
Valoare minimă: 1.0
Valoare maximă: 2739.0
Diferența de valori maxime și minime: 2738.0
Mediană: 2.387426
Abaterea mediană absolută: 2.847173785096963
Intervalul intercuartil: 1.0

Analiză pentru coloana numerică 'Sedentary_hours_daily':
Medie: 3.693571056741281
Abaterea standard: 21.759834908880748
Abaterea medie absolută: 1.1338850443643116
Valoare minimă: 2.21
Valoare maximă: 956.58
Diferența de valori maxime și minime: 954.37
Mediană: 3.13
Abaterea mediană absolută: 1.0310052920662904
Intervalul intercuartil: 0.8700000000000001

Analiză pentru coloana numerică 'Age':
Medie: 44.79250626392504
Abaterea standard: 633.3118370767136
```

```
Abaterea medie absolută: 40.94688616864699
Valoare minimă: 15.0
Valoare maximă: 19685.0
Diferența de valori maxime și minime: 19670.0
Mediană: 22.829753
Abaterea mediană absolută: 40.89983783532187
Intervalul intercuartil: 6.02834

Analiză pentru coloana numerică 'Est_avg_calorie_intake':
Medie: 2253.68766267569
Abaterea standard: 434.07579419142866
Abaterea medie absolută: 375.36234408538627
Valoare minimă: 1500
Valoare maximă: 3000
Diferența de valori maxime și minime: 1500
Mediană: 2253.0
Abaterea mediană absolută: 189.2663097071438
Intervalul intercuartil: 757.0

Analiză pentru coloana numerică 'Main_meals_daily':
Medie: 2.683471861009891
Abaterea standard: 0.7791790556845525
Abaterea medie absolută: 0.5954195898322087
Valoare minimă: 1.0
Valoare maximă: 4.0
Diferența de valori maxime și minime: 3.0
Mediană: 3.0
Abaterea mediană absolută: 0.5858545638769935
Intervalul intercuartil: 0.341361

Analiză pentru coloana numerică 'Height':
Medie: 3.5734877667881313
Abaterea standard: 58.09815976912103
Abaterea medie absolută: 3.7385247521774323
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)

[37]:
```
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()
```

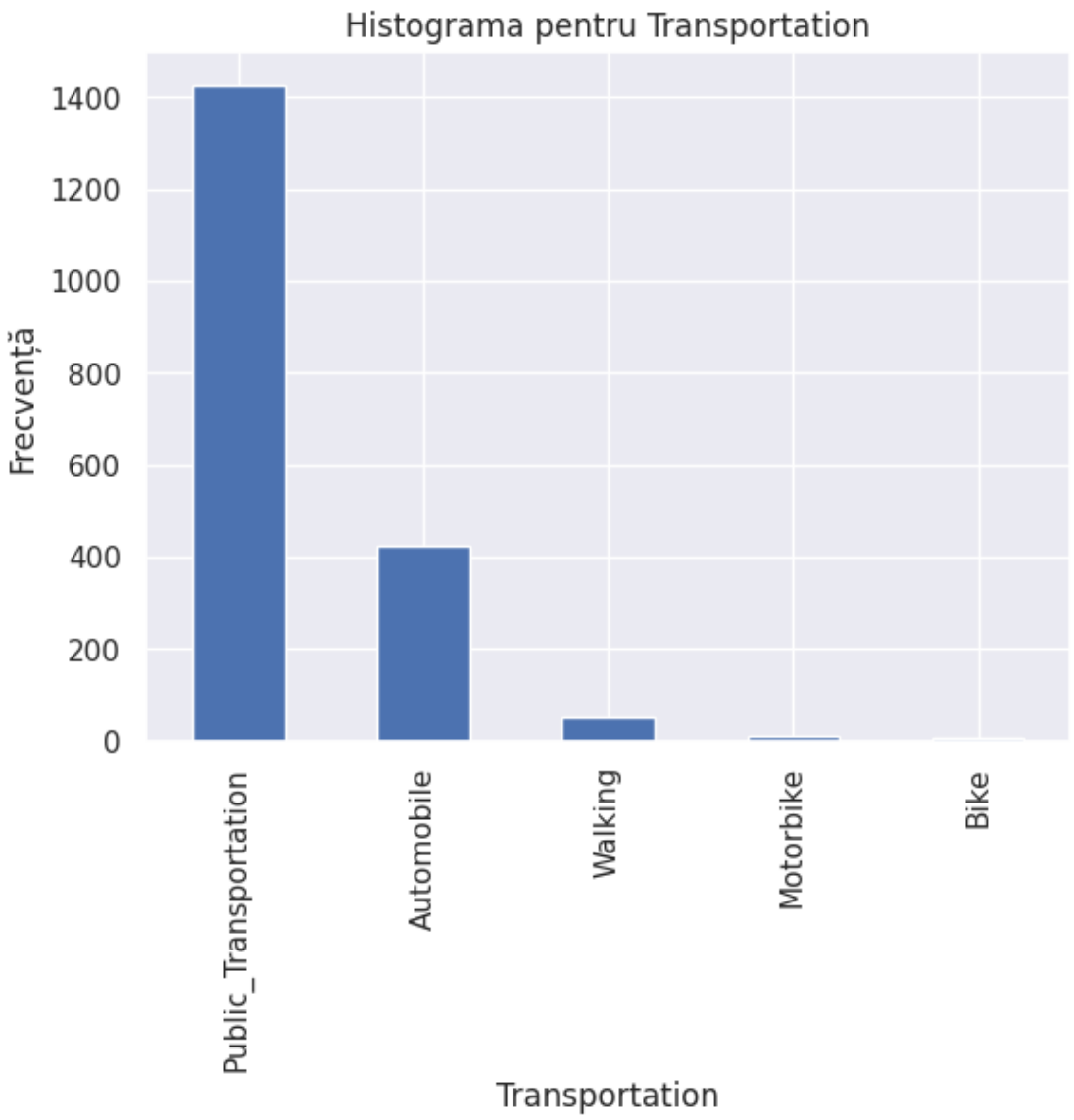
```
Numărul de valori NaN pentru coloana 'Weight' este: 190
Analiză pentru coloana numerică 'Regular_fiber_diet':
Medie: 2.4208481255230123
Abaterea standard: 0.5337871366466765
Abaterea medie absolută: 0.4787342769428056
Valoare minimă: 1.0
Valoare maximă: 3.0
Diferența de valori maxime și minime: 2.0
Mediană: 2.392422
Abaterea mediană absolută: 0.17269574705012175
Intervalul intercuartil: 1.0

Analiză pentru coloana numerică 'Sedentary_hours_daily':
Medie: 3.196307531380753
Abaterea standard: 0.5753305109633888
Abaterea medie absolută: 0.47045444101118683
Valoare minimă: 2.21
Valoare maximă: 4.67
Diferența de valori maxime și minime: 2.46
Mediană: 3.13
Abaterea mediană absolută: 0.2839333169937501
Intervalul intercuartil: 0.8624999999999998

Analiză pentru coloana numerică 'Age':
Medie: 24.36244938493724
Abaterea standard: 6.414437374618447
Abaterea medie absolută: 4.817102573703191
Valoare minimă: 15.0
Valoare maximă: 61.0
Diferența de valori maxime și minime: 46.0
Mediană: 22.830928999999998
Abaterea mediană absolută: 3.2251370961106693
Intervalul intercuartil: 6.0222275

Analiză pentru coloana numerică 'Est_avg_calorie_intake':
Medie: 2254.3059623430963
Abaterea standard: 433.767146303113
Abaterea medie absolută: 374.88505431452535
Valoare minimă: 1500
Valoare maximă: 3000
Diferența de valori maxime și minime: 1500
Mediană: 2253.5
Abaterea mediană absolută: 189.54099740025558
Intervalul intercuartil: 756.0
```

```
Analiză pentru coloana numerică 'Main_meals_daily':
Medie: 2.6828592515690377
Abaterea standard: 0.7796353185903193
Abaterea medie absolută: 0.5959053255884438
Valoare minimă: 1.0
Valoare maximă: 4.0
Diferența de valori maxime și minime: 3.0
Mediană: 3.0
Abaterea mediană absolută: 0.5865901933341066
Intervalul intercuartil: 0.3414012500000001

Analiză pentru coloana numerică 'Height':
Medie: 1.7021757322175732
Abaterea standard: 0.09320826529327833
Abaterea medie absolută: 0.07704570123072077
Valoare minimă: 1.45
Valoare maximă: 1.98
Diferența de valori maxime și minime: 0.53
Mediană: 1.7
Abaterea mediană absolută: 0.04253016622608145
Intervalul intercuartil: 0.14000000000000012

Analiză pentru coloana numerică 'Water_daily':
Medie: 2.0110308158995815
Abaterea standard: 0.6113253616172403
Abaterea medie absolută: 0.47114246384867214
Valoare minimă: 1.0
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
```

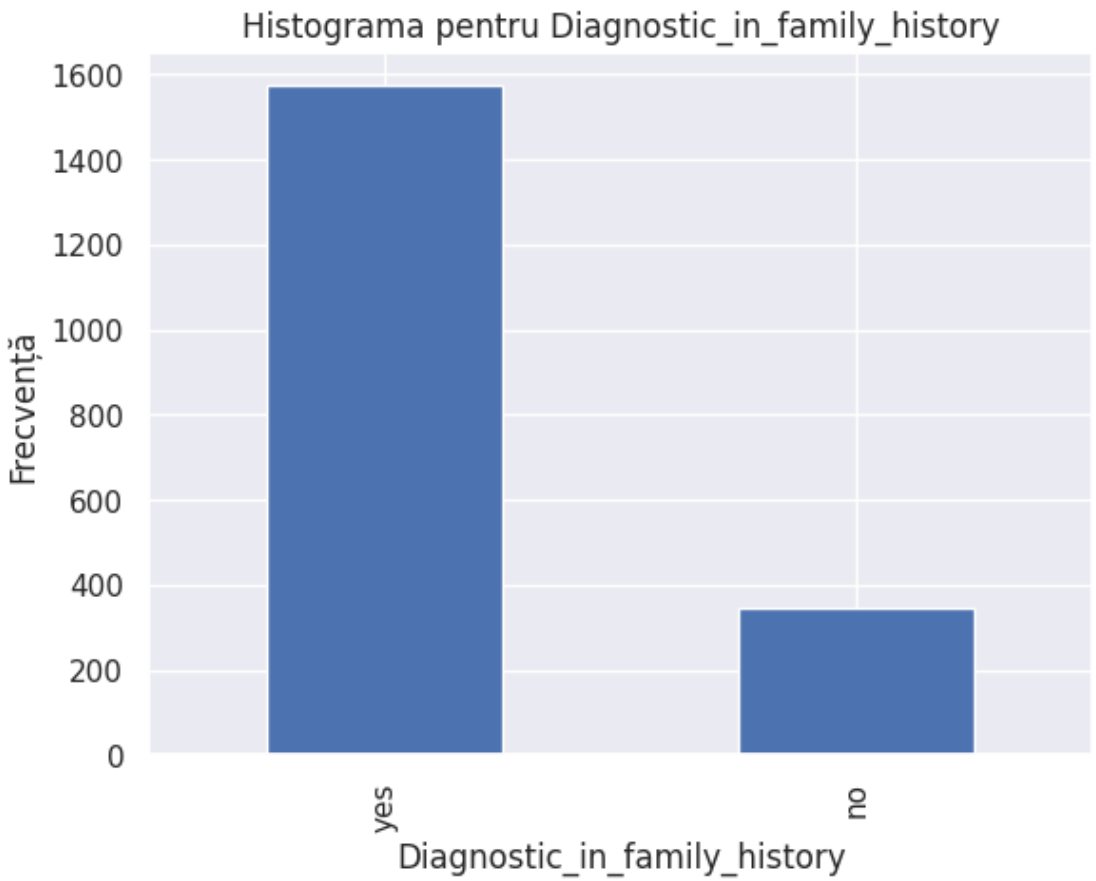
```
Valoare minimă: 0.0
Valoare maximă: 3.0
Diferența de valori maxime și minime: 3.0
Mediană: 1.0
Abaterea mediană absolută: 0.4053436104398995
Intervalul intercuartil: 1.5678555

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
```

[38]:
```
# Analiza pentru atribute categorice
for column in categorical_attributes:
    analyze_categorical_attributes(df[column])
```

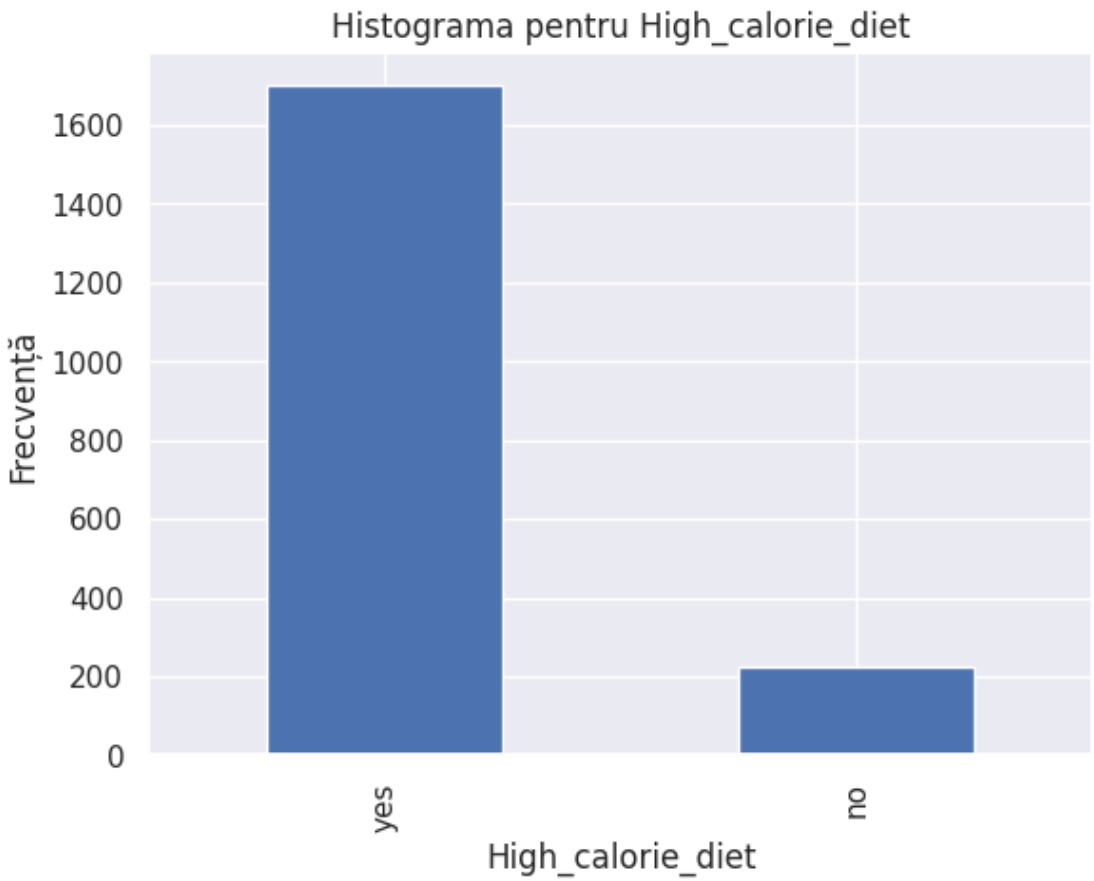
```
Analiză pentru coloana categorică: Transportation
Valori unice: ['Public_Transportation' 'Walking' 'Automobile' 'Motorbike'
'Bike']

Histogramă:
```



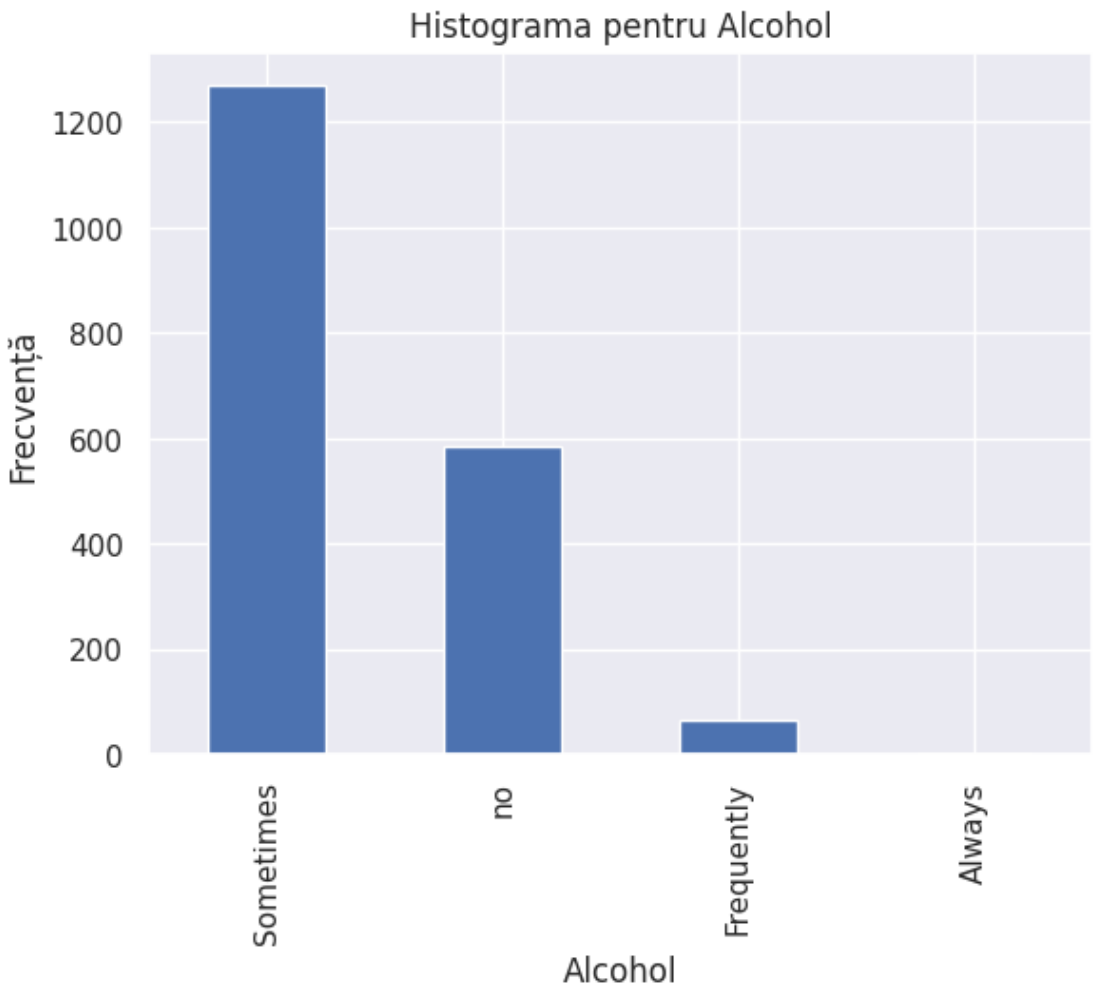

```
Analiză pentru coloana categorică: Diagnostic_in_family_history
Valori unice: ['yes' 'no']

Histogramă:
```



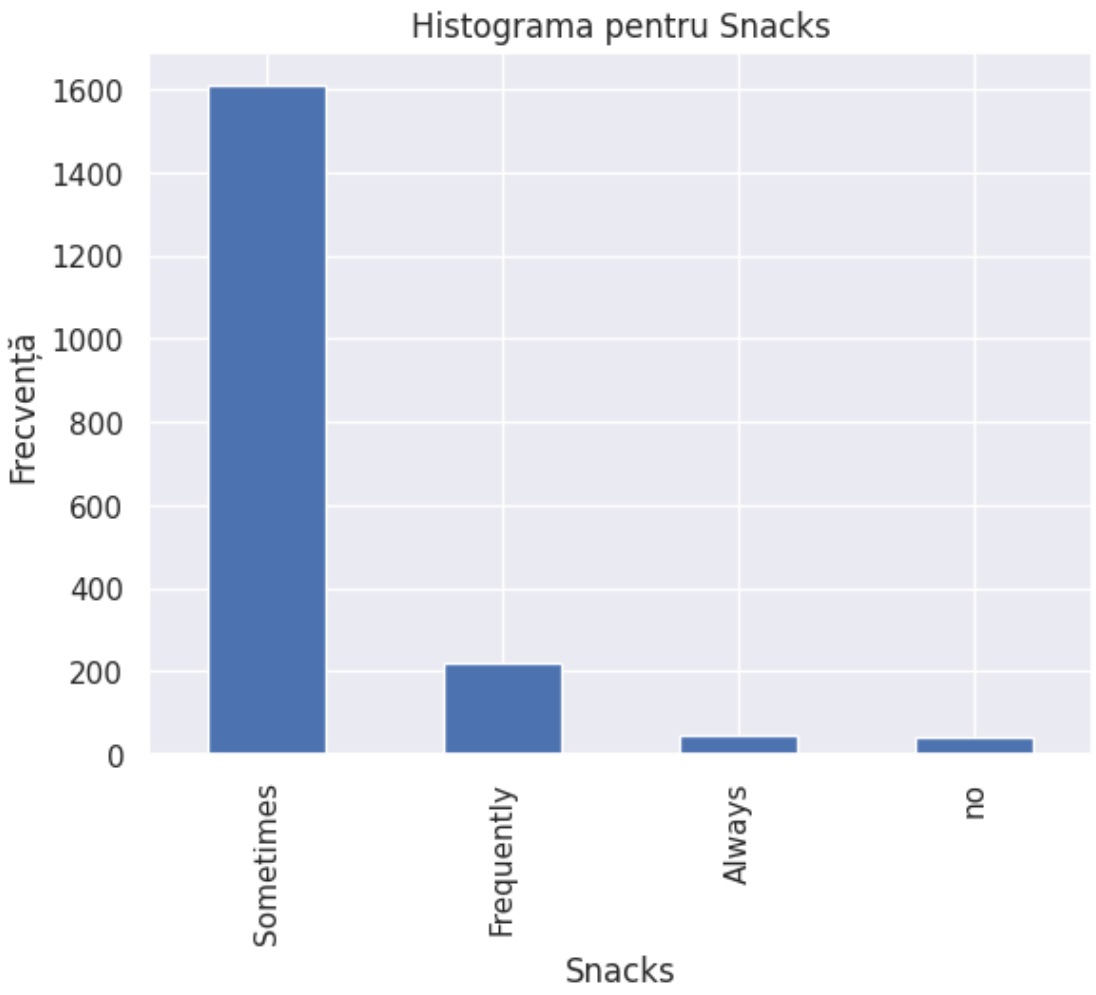

```
Analiză pentru coloana categorică: High_calorie_diet
Valori unice: ['no' 'yes']

Histogramă:
```



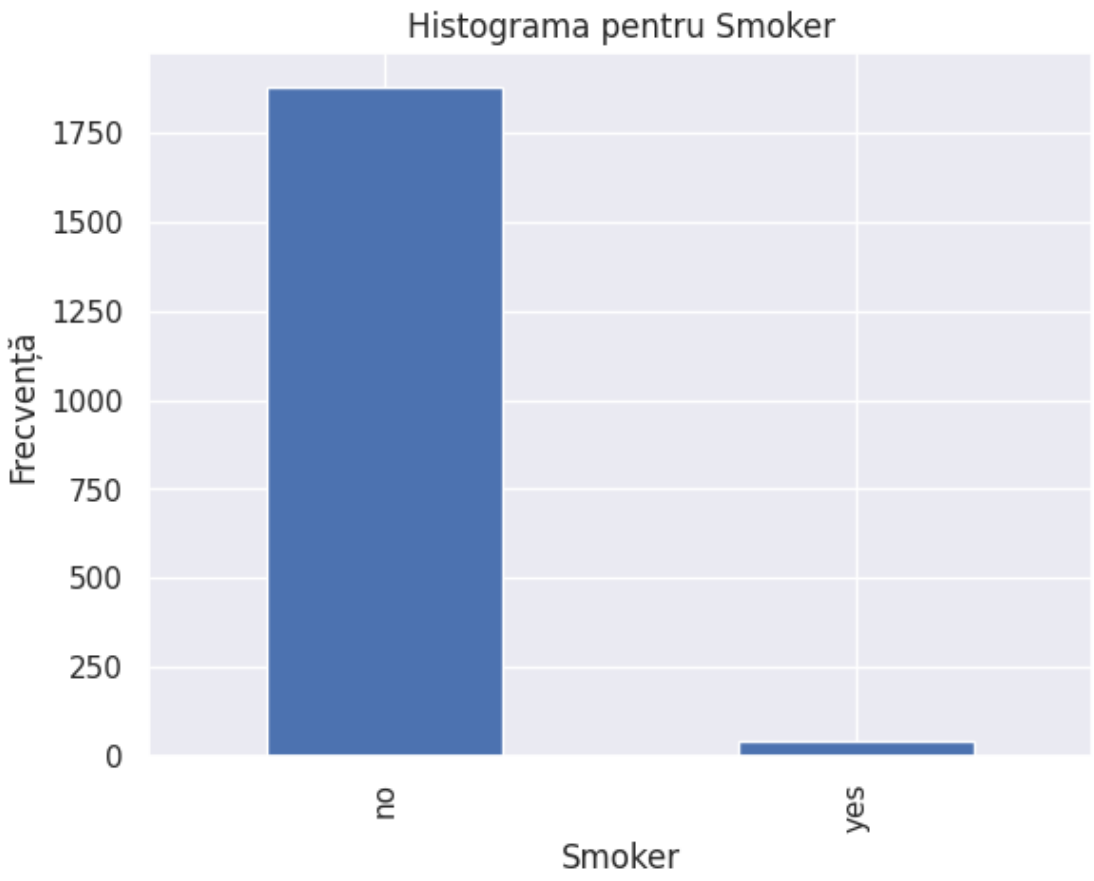

```
Analiză pentru coloana categorică: Alcohol
Valori unice: ['no' 'Sometimes' 'Frequently' 'Always']

Histogramă:
```




```
Analiză pentru coloana categorică: Snacks
Valori unice: ['Sometimes' 'Frequently' 'Always' 'no']

Histogramă:
```

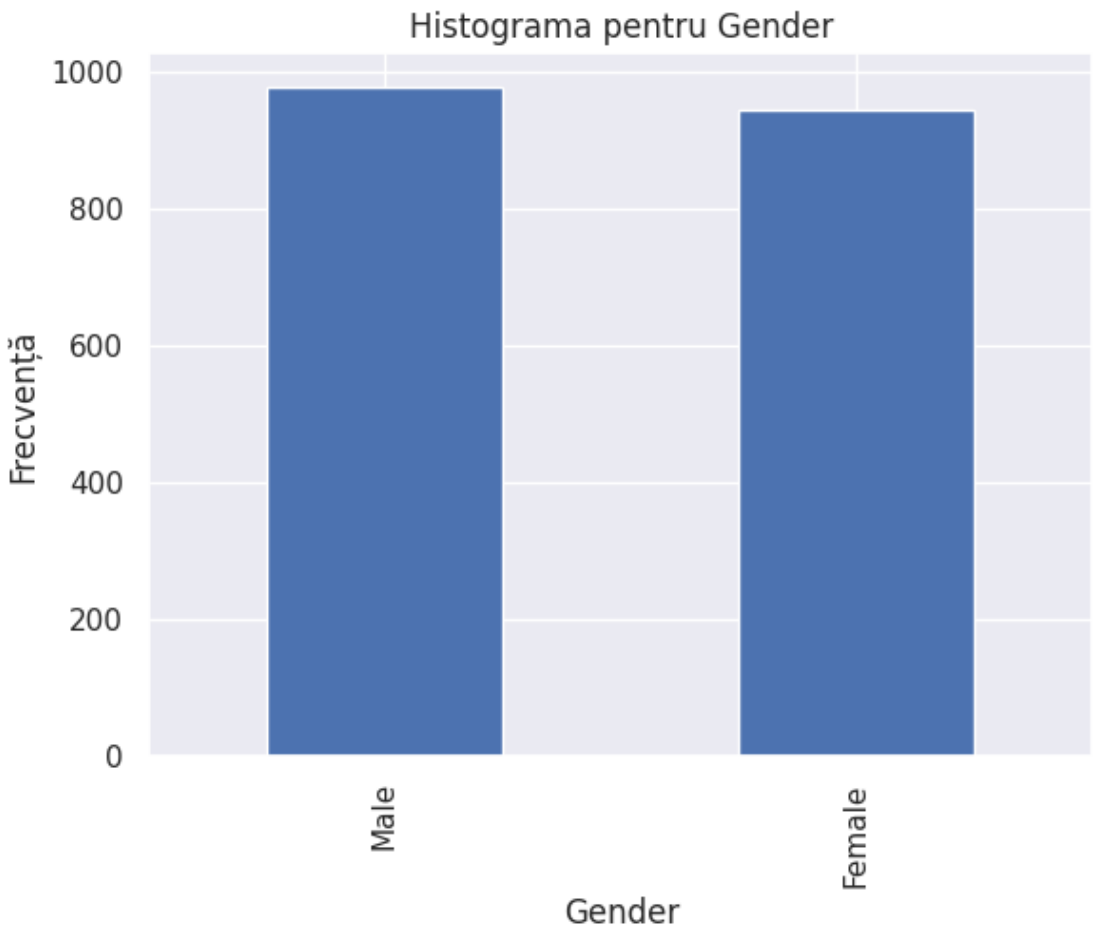



```
Analiză pentru coloana categorică: Smoker
Valori unice: ['no' 'yes']

Histogramă:
```

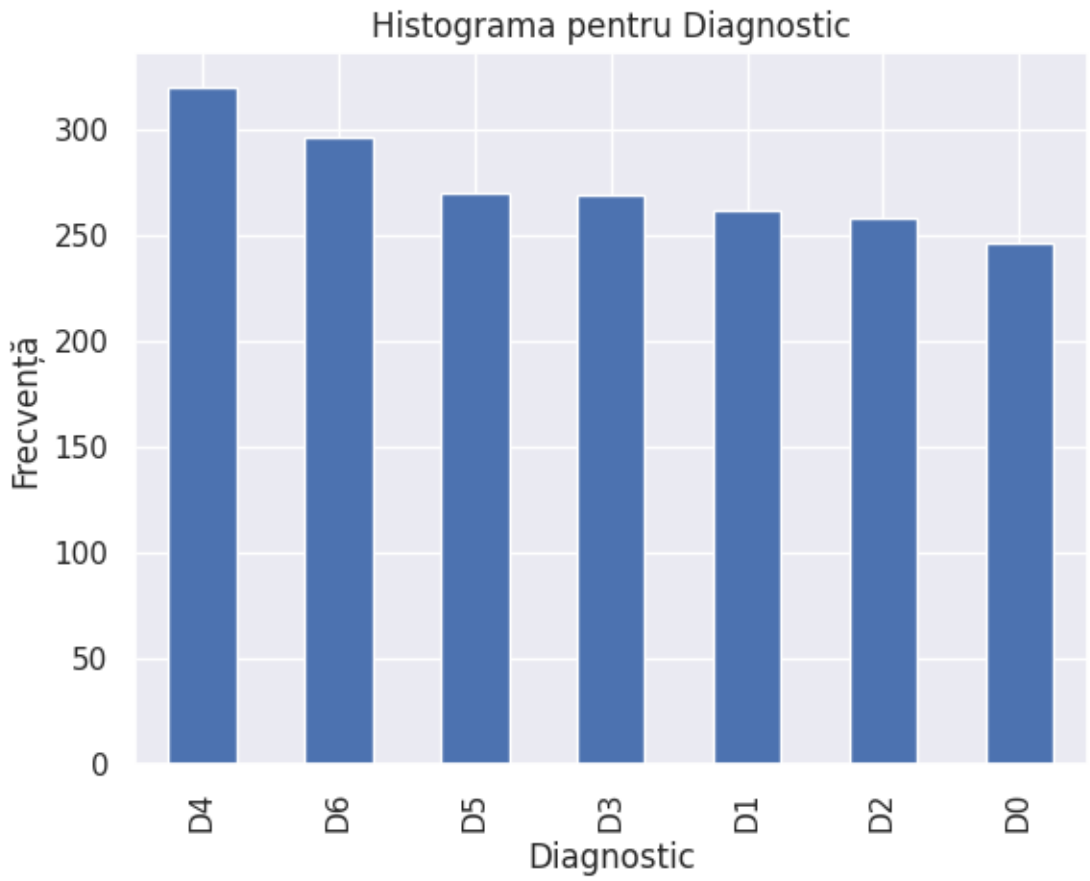



```
Analiză pentru coloana categorică: Calorie_monitoring
Valori unice: ['no' 'yes']

Histogramă:
```


```
Analiză pentru coloana categorică: Gender
Valori unice: ['Female' 'Male']

Histogramă:
```




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

Histogramă:
```



[39]:
```
# Analiza de covarianță între atribute
analyze_covariance(df[numeric_attributes])
```

```
Analiză de covarianță între coloanele numerice:
                        Regular_fiber_diet  Sedentary_hours_daily  \
Regular_fiber_diet                3898.705833              -2.044016
Sedentary_hours_daily               -2.044016             473.490415
Age                                -28.298519             -13.086870
Est_avg_calorie_intake            -411.186479             273.718418
Main_meals_daily                     0.098487              -0.611959
Height                              -2.769186              -0.587676
Water_daily                         -0.239886              -0.071017
Weight                            -135.811353              19.163702
Physical_activity_level              0.128235              -0.476095
Technology_time_use                 -2.233738             648.167098

                                    Age  Est_avg_calorie_intake  \
Regular_fiber_diet          -28.298519             -411.186479
```

```
Sedentary_hours_daily       -13.086870             273.718418
Age                      401083.882981             -88.784631
Est_avg_calorie_intake      -88.784631          188421.795103
Main_meals_daily             18.484813              -4.573814
Height                      -34.922604            -930.357530
Water_daily                  10.450899              -4.249024
Weight                    -2773.866097          -20457.233307
Physical_activity_level      19.992288              -1.738810
Technology_time_use         -22.876826             371.218433

                         Main_meals_daily       Height  Water_daily  \
Regular_fiber_diet               0.098487    -2.769186    -0.239886
Sedentary_hours_daily           -0.611959    -0.587676    -0.071017
Age                             18.484813   -34.922604    10.450899
Est_avg_calorie_intake          -4.573814  -930.357530    -4.249024
Main_meals_daily                 0.607120     0.611016     0.032494
Height                           0.611016  3375.396169    -1.181213
Water_daily                      0.032494    -1.181213     0.373363
Weight                          -0.380838  -182.428303   -28.511896
Physical_activity_level          0.095262    -0.237567     0.089479
Technology_time_use             -0.844501    -1.675769    -0.099792

                               Weight  Physical_activity_level  \
Regular_fiber_diet      -1.358114e+02                 0.128235
Sedentary_hours_daily    1.916370e+01                -0.476095
Age                     -2.773866e+03                19.992288
Est_avg_calorie_intake  -2.045723e+04                -1.738810
Main_meals_daily        -3.808376e-01                 0.095262
Height                  -1.824283e+02                -0.237567
Water_daily             -2.851190e+01                 0.089479
Weight                   1.040484e+07                 7.405877
Physical_activity_level  7.405877e+00                 0.731924
Technology_time_use     -5.287232e+01                -0.648245

                         Technology_time_use
Regular_fiber_diet                 -2.233738
Sedentary_hours_daily             648.167098
Age                               -22.876826
Est_avg_calorie_intake            371.218433
Main_meals_daily                   -0.844501
Height                             -1.675769
Water_daily                        -0.099792
Weight                            -52.872324
Physical_activity_level            -0.648245
Technology_time_use               887.439837
```

[40]:
```
for column in numeric_attributes:
    analyze_covariance_with_class(df, column)
```

```
Analiză de covarianță între atribute și clasă:
Covarianța între 'Regular_fiber_diet' și 'Regular_fiber_diet' este:
3898.7058334924154
Covarianța între 'Sedentary_hours_daily' și 'Regular_fiber_diet' este:
-2.044015582817564
Covarianța între 'Age' și 'Regular_fiber_diet' este: -28.29851855076531
Covarianța între 'Est_avg_calorie_intake' și 'Regular_fiber_diet' este:
-411.18647923137803
Covarianța între 'Main_meals_daily' și 'Regular_fiber_diet' este:
0.09848677385841101
Covarianța între 'Height' și 'Regular_fiber_diet' este: -2.7691858949260015
Covarianța între 'Water_daily' și 'Regular_fiber_diet' este:
-0.23988565297368641
Covarianța între 'Weight' și 'Regular_fiber_diet' este: -135.81135328527245
Covarianța între 'Physical_activity_level' și 'Regular_fiber_diet' este:
0.12823549479193203
Covarianța între 'Technology_time_use' și 'Regular_fiber_diet' este:
-2.233738467434762

Analiză de covarianță între atribute și clasă:
Covarianța între 'Regular_fiber_diet' și 'Sedentary_hours_daily' este:
-2.044015582817564
Covarianța între 'Sedentary_hours_daily' și 'Sedentary_hours_daily' este:
473.49041526174545
Covarianța între 'Age' și 'Sedentary_hours_daily' este: -13.086870154588814
Covarianța între 'Est_avg_calorie_intake' și 'Sedentary_hours_daily' este:
273.71841803856506
Covarianța între 'Main_meals_daily' și 'Sedentary_hours_daily' este:
-0.6119592853836088
Covarianța între 'Height' și 'Sedentary_hours_daily' este: -0.58767610733342
Covarianța între 'Water_daily' și 'Sedentary_hours_daily' este:
-0.07101683654380038
Covarianța între 'Weight' și 'Sedentary_hours_daily' este: 19.163701704771256
Covarianța între 'Physical_activity_level' și 'Sedentary_hours_daily' este:
-0.47609527780979155
Covarianța între 'Technology_time_use' și 'Sedentary_hours_daily' este:
648.1670983428767

Analiză de covarianță între atribute și clasă:
Covarianța între 'Regular_fiber_diet' și 'Age' este: -28.29851855076531
Covarianța între 'Sedentary_hours_daily' și 'Age' este: -13.086870154588814
Covarianța între 'Age' și 'Age' este: 401083.8829814817
Covarianța între 'Est_avg_calorie_intake' și 'Age' este: -88.78463111752522
```

```
Covarianța între 'Main_meals_daily' și 'Age' este: 18.484813296844845
Covarianța între 'Height' și 'Age' este: -34.92260353512932
Covarianța între 'Water_daily' și 'Age' este: 10.450898858224777
Covarianța între 'Weight' și 'Age' este: -2773.866096816051
Covarianța între 'Physical_activity_level' și 'Age' este: 19.992288346468047
Covarianța între 'Technology_time_use' și 'Age' este: -22.876826228253233

Analiză de covarianță între atribute și clasă:
Covarianța între 'Regular_fiber_diet' și 'Est_avg_calorie_intake' este:
-411.18647923137803
Covarianța între 'Sedentary_hours_daily' și 'Est_avg_calorie_intake' este:
273.71841803856506
Covarianța între 'Age' și 'Est_avg_calorie_intake' este: -88.78463111752522
Covarianța între 'Est_avg_calorie_intake' și 'Est_avg_calorie_intake' este:
188421.79510291948
Covarianța între 'Main_meals_daily' și 'Est_avg_calorie_intake' este:
-4.573813759580246
Covarianța între 'Height' și 'Est_avg_calorie_intake' este: -930.3575298645453
Covarianța între 'Water_daily' și 'Est_avg_calorie_intake' este:
-4.249024305902423
Covarianța între 'Weight' și 'Est_avg_calorie_intake' este: -20457.233306793463
Covarianța între 'Physical_activity_level' și 'Est_avg_calorie_intake' este:
-1.738809785234469
Covarianța între 'Technology_time_use' și 'Est_avg_calorie_intake' este:
371.21843332465744

Analiză de covarianță între atribute și clasă:
Covarianța între 'Regular_fiber_diet' și 'Main_meals_daily' este:
0.09848677385841101
Covarianța între 'Sedentary_hours_daily' și 'Main_meals_daily' este:
-0.6119592853836088
Covarianța între 'Age' și 'Main_meals_daily' este: 18.484813296844845
Covarianța între 'Est_avg_calorie_intake' și 'Main_meals_daily' este:
-4.573813759580246
Covarianța între 'Main_meals_daily' și 'Main_meals_daily' este:
0.6071200008174704
Covarianța între 'Height' și 'Main_meals_daily' este: 0.611016217708976
Covarianța între 'Water_daily' și 'Main_meals_daily' este: 0.03249449276365082
Covarianța între 'Weight' și 'Main_meals_daily' este: -0.38083756006293185
Covarianța între 'Physical_activity_level' și 'Main_meals_daily' este:
0.09526162991469111
Covarianța între 'Technology_time_use' și 'Main_meals_daily' este:
-0.8445008050575866

Analiză de covarianță între atribute și clasă:
Covarianța între 'Regular_fiber_diet' și 'Height' este: -2.7691858949260015
Covarianța între 'Sedentary_hours_daily' și 'Height' este: -0.58767610733342
Covarianța între 'Age' și 'Height' este: -34.92260353512932
```

```
Covarianța între 'Est_avg_calorie_intake' și 'Height' este: -930.3575298645453
Covarianța între 'Main_meals_daily' și 'Height' este: 0.611016217708976
Covarianța între 'Height' și 'Height' este: 3375.3961685583113
Covarianța între 'Water_daily' și 'Height' este: -1.1812127185217631
Covarianța între 'Weight' și 'Height' este: -182.4283025252796
Covarianța între 'Physical_activity_level' și 'Height' este:
-0.23756680714057624
Covarianța între 'Technology_time_use' și 'Height' este: -1.6757686860142293

Analiză de covarianță între atribute și clasă:
Covarianța între 'Regular_fiber_diet' și 'Water_daily' este:
-0.23988565297368641
Covarianța între 'Sedentary_hours_daily' și 'Water_daily' este:
-0.07101683654380038
Covarianța între 'Age' și 'Water_daily' este: 10.450898858224777
Covarianța între 'Est_avg_calorie_intake' și 'Water_daily' este:
-4.249024305902423
Covarianța între 'Main_meals_daily' și 'Water_daily' este: 0.03249449276365082
Covarianța între 'Height' și 'Water_daily' este: -1.1812127185217631
Covarianța între 'Water_daily' și 'Water_daily' este: 0.37336279900976826
Covarianța între 'Weight' și 'Water_daily' este: -28.51189616023247
Covarianța între 'Physical_activity_level' și 'Water_daily' este:
0.08947910795176638
Covarianța între 'Technology_time_use' și 'Water_daily' este:
-0.09979211853743712

Analiză de covarianță între atribute și clasă:
Covarianța între 'Regular_fiber_diet' și 'Weight' este: -135.81135328527245
Covarianța între 'Sedentary_hours_daily' și 'Weight' este: 19.163701704771256
Covarianța între 'Age' și 'Weight' este: -2773.866096816051
Covarianța între 'Est_avg_calorie_intake' și 'Weight' este: -20457.233306793463
Covarianța între 'Main_meals_daily' și 'Weight' este: -0.38083756006293185
Covarianța între 'Height' și 'Weight' este: -182.4283025252796
Covarianța între 'Water_daily' și 'Weight' este: -28.51189616023247
Covarianța între 'Weight' și 'Weight' este: 10404840.733153842
Covarianța între 'Physical_activity_level' și 'Weight' este: 7.40587744986486
Covarianța între 'Technology_time_use' și 'Weight' este: -52.8723240960725

Analiză de covarianță între atribute și clasă:
Covarianța între 'Regular_fiber_diet' și 'Physical_activity_level' este:
0.12823549479193203
Covarianța între 'Sedentary_hours_daily' și 'Physical_activity_level' este:
-0.47609527780979155
Covarianța între 'Age' și 'Physical_activity_level' este: 19.992288346468047
Covarianța între 'Est_avg_calorie_intake' și 'Physical_activity_level' este:
-1.738809785234469
Covarianța între 'Main_meals_daily' și 'Physical_activity_level' este:
0.09526162991469111
```

```
Covarianța între 'Height' și 'Physical_activity_level' este:
-0.23756680714057624
Covarianța între 'Water_daily' și 'Physical_activity_level' este:
0.08947910795176638
Covarianța între 'Weight' și 'Physical_activity_level' este: 7.40587744986486
Covarianța între 'Physical_activity_level' și 'Physical_activity_level' este:
0.7319241249412074
Covarianța între 'Technology_time_use' și 'Physical_activity_level' este:
-0.6482446959932977

Analiză de covarianță între atribute și clasă:
Covarianța între 'Regular_fiber_diet' și 'Technology_time_use' este:
-2.233738467434762
Covarianța între 'Sedentary_hours_daily' și 'Technology_time_use' este:
648.1670983428767
Covarianța între 'Age' și 'Technology_time_use' este: -22.876826228253233
Covarianța între 'Est_avg_calorie_intake' și 'Technology_time_use' este:
371.21843332465744
Covarianța între 'Main_meals_daily' și 'Technology_time_use' este:
-0.8445008050575866
Covarianța între 'Height' și 'Technology_time_use' este: -1.6757686860142293
Covarianța între 'Water_daily' și 'Technology_time_use' este:
-0.09979211853743712
Covarianța între 'Weight' și 'Technology_time_use' este: -52.8723240960725
Covarianța între 'Physical_activity_level' și 'Technology_time_use' este:
-0.6482446959932977
Covarianța între 'Technology_time_use' și 'Technology_time_use' este:
887.4398365651604
```

4.1 Concluzii EDA

- Din graficele realizate cu scopul de data vizualization, observam domeniul de valori posibile pentru fiecare atribut, cat si densitatea acestor valori (ponderile), dar si distributia lor. Astfel putem intui integritatea datelor (daca avem valori foarte extreme, neverosimile) si putem decide o strategie de encodare, scalare, tratare a valorilor lipsa.
- Scaler: Este necesar deoarece atributele au valori foarte diferite, unele in intervalul [0-3], altele [1-2], altele [60-160] etc.
- Encoding: Encodarea atributelor non-numerice in valori numerice este un pas obligatoriu si care modifica in sine datele de intrare. Astfel este foarte important sa alegem metoda cea mai buna pentru dataset-ul nostru. Obiectivul este sa folosim un Encoder care se muleaza cat mai bine pe scenariul datelor noastre. Ele pot fi categorice sau ordinale, astfel incat in unele cazuri sa poata fi relevanta pastrarea unei ordini intre acestea. Totodata putem gasi relevanta numarul de aparitii al fiecarei categorii, astfel incat se preteaza un CountEncoder. Cazurile sunt diverse si prezinta avantaje / dezavantaje atat fata de considerentele mentionate mai sus, cat si asupra distributiei datelor.
- Distributia datelor: Iată câteva exemple de distribuții de date și cum arată acestea:

  - Distribuția normală (Gaussiană): Aceasta este o distribuție cu o formă de clopot, cu valori mai frecvente în jurul valorii medii și cu o dispersie a datelor care scade simetric față de media centrală. Exemplu: Înălțimile unei populații, scorurile unui test standardizat etc.
  - Distribuția uniformă: Toate valorile sunt egale de-a lungul întregului interval. Exemplu: Aruncarea unui zar echilibrat (dacă zarul este corect, fiecare față are aceeași probabilitate de a fi aleasă).
  - Distribuția exponențială: Probabilitatea că un eveniment are loc într-un anumit interval de timp este invers proporțională cu timpul. Exemplu: Durata de viață a unui dispozitiv electronic.
  - Distribuția binomială: Reprezintă numărul de succese într-un număr fix de încercări identice și independente, fiecare cu o probabilitate fixă de succes. Exemplu: Numărul de capete într-o serie de aruncări de monedă.

- Valori lipsa: Este ideal sa tratam valorile lipsa, astfel incat sa putem dispune de un dataset mai robust si numeros. Exista mai multe strategii, cele populare fiind inlocuirea cu min / max (pentru outliers) sau mean / median pentru valorile lipsa. Vom trata valorile lipsa ale lui Weight (aproximativ 10% din total)
- Selectie features: Dupa cum am analizat anterior valorile p-values obtinute pentru fiecare atribut, am constatat care dintre acestea sunt relevante pentru a atinge target-ul. Astfel le vom folosi doar pe acele pentru partea de invatare. Putem automatiza procesul folosind selectors.
- Modele de invatare: Evident ca vom utiliza mai multe modele de invatare cu configuratii diferite, pentru a putea realiza o concluzie clara asupra performantelor acestora si a determina metoda de analiza cea mai eficienta
- Evaluarea algoritmului: Vom utiliza diferite metrici, in special Accuracy-ul obtinut dar ne vom orienta si dupa celelalte metrici, mai ales cand o sa avem rezultate asemanatoare intre diversi algoritmi propusi. Totodata este importanta si consistenta modelului in a obtine date similare pentru dataset-uri diferite (randomizam datasetul, astfel incat sa obtinem un set de date mai bogat / variat, sa eliminam bias-ul generat de numarul mic de date sau distributia acestora care poate fi favorabila unui model, fata de celelalte)

5 3.2. Extragerea manuală a atributelor și utilizarea algoritmilor clasici de Învățare Automată

Importam setul de date si il pre-procesam

[41]:
```
# Plecam de la setul de date original
df = pd.read_csv(f"date_tema_1_iaut_2024.csv")

pd.set_option('future.no_silent_downcasting', True)

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

```
# Specificam coloana target
target_column = "Diagnostic"
```

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

Definim o functie care permite utilizarea de configuratii diferite pentru analiza si prelucrarea setului de date

[42]:
```
def preprocess_data(df, target_column, scaler, imputer, encoder):
    # Separate features and target
    X = df.drop(target_column, axis=1)
    y = df[target_column]

    # Separate numeric and non-numeric columns
    numeric_cols = X.select_dtypes(include=[np.number]).columns
    non_numeric_cols = X.select_dtypes(exclude=[np.number]).columns

    # Iterate over non-numeric columns
    for column in non_numeric_cols:
        # Treat the boolean variables by encoding them to 0 / 1
        if X[column].nunique() == 2:
            X[column] = X[column].replace({"yes": 1, "no": 0})
            non_numeric_cols = non_numeric_cols.drop(column)

    # Impute missing values for numeric columns
    imputed_data = imputer.fit_transform(X[numeric_cols])
    X_imputed = pd.DataFrame(imputed_data, columns=numeric_cols)

    # Encode non-numeric columns
    encoder.fit(X[non_numeric_cols])
    encoded_data = encoder.transform(X[non_numeric_cols])
    encoded_cols = encoder.get_feature_names_out(non_numeric_cols)
    X_encoded = pd.DataFrame(encoded_data, columns=encoded_cols)

    # Concatenate numeric and encoded categorical data
    X_final = pd.concat([X_imputed, X_encoded], axis=1)

    # Standardize numeric features
    X_scaled = pd.DataFrame(scaler.fit_transform(X_final), columns=X_final.
 ↪columns)

    return X_scaled, y

def evaluate_model(model, X_test, y_test):
    y_pred = model.predict(X_test)
```

```
    accuracy = round(accuracy_score(y_test, y_pred), 3)
    precision = np.round(precision_score(y_test, y_pred, average=None), 3)
    recall = np.round(recall_score(y_test, y_pred, average=None), 3)
    f1 = np.round(f1_score(y_test, y_pred, average=None), 3)

    return accuracy, precision, recall, f1

def get_confusion_matrix(model, X_test, y_test):
    y_pred = model.predict(X_test)
    cm = confusion_matrix(y_test, y_pred)

    return cm

def run_grid_search(model, param_grid, X_train, y_train):
    grid_search = GridSearchCV(model, param_grid, cv=5)
    grid_search.fit(X_train, y_train)

    best_params = grid_search.best_params_
    best_model = grid_search.best_estimator_

    return best_params, best_model

def classify_data(df, target_column, scaler, imputer, encoder, selector,␣
 ↪classifier, param_grid, test_size=0.2):
    # Preprocesare date
    X, y = preprocess_data(df, target_column, scaler, imputer, encoder)

    # Selectare caracteristici
    X_selected = selector.fit_transform(X, y)

    # Împărțire date în seturi de antrenare și testare
    X_train, X_test, y_train, y_test = train_test_split(X_selected, y,␣
 ↪test_size=test_size)

    # Căutare hiperparametri
    best_params, best_model = run_grid_search(classifier, param_grid, X_train,␣
 ↪y_train)

    # Evaluare model
    accuracy, precision, recall, f1 = evaluate_model(best_model, X_test, y_test)

    # Matrice de confuzie
    confusion_matrix = get_confusion_matrix(best_model, X_test, y_test)

    return {
        "Best Parameters": best_params,
        "Accuracy": accuracy,
```

```
        "Precision": precision,
        "Recall": recall,
        "F1": f1,
        "Confusion Matrix": confusion_matrix
    }
```

Cream o functie care primeste detaliile configuratiei si executa clasificarea

[43]:
```
def compare_classify_configurations(df, target_column, scalers, imputers,␣
↪encoders, selectors, classifiers, param_grids):
    all_results = {}

    total_configurations = len(scalers) * len(imputers) * len(encoders) *␣
↪len(selectors) * len(classifiers)
    configuration_count = 0

    # Iterate over all combinations of pre-processors and classifiers
    for scaler_name, scaler in scalers.items():
        for imputer_name, imputer in imputers.items():
            for encoder_name, encoder in encoders.items():
                for selector_name, selector in selectors.items():
                    for classifier_name, classifier in classifiers.items():
                        param_grid = param_grids[classifier_name]

                        # Incrementăm numărul configurației testate
                        configuration_count += 1

                        # Construim numele configurației
                        config_name =␣
↪f"{scaler_name}_{imputer_name}_{encoder_name}_{selector_name}_{classifier_name}"

                        # Afisăm progresul
                        print(f"Testing configuration {configuration_count}/
↪{total_configurations}: {config_name}")

                        start_time = time.time()
                        result = classify_data(df.copy(), target_column,␣
↪scaler, imputer, encoder, selector, classifier, param_grid)
                        end_time = time.time()

                        execution_time = end_time - start_time
                        print(f"Duration: {execution_time} seconds")

                        # Salvăm rezultatele pentru configurația curentă
                        all_results[(scaler_name, imputer_name, encoder_name,␣
↪selector_name, classifier_name)] = {
                            "Configuration": config_name,
```

```
                            "Results": result,
                            "Duration": round(execution_time, 3)
                        }

    return all_results
```

5.0.1 Analiza comparativa

Evident ca am dori sa utilizam cat mai multe configuratii diverse, in speta o varietate cat mai mare de metode de abordare a temelor urmatoare (scalers, imputers, encoders, selectors etc.), dar nu ne permite timpul de rulare pe o masina non-optimizata pentru workload specific Machine Learning.

Definim scalerii utilizati pentru standardizare

[44]:
```
scalers = {
    "MinMaxScaler": MinMaxScaler()
}
```

Exemple de abordari pentru Scaling

- Standardizare (z-score normalization):
  - Pro: Menține distribuția datelor și înseamnă că datele standardizate au o medie de 0 și o deviație standard de 1, ceea ce este util pentru modelele care presupun o distribuție normală.
  - Contra: Poate fi afectată de valori aberante (outliers), deoarece medie și deviația standard sunt sensibile la acestea.
- MinMax Scaling:
  - Pro: Scala valorile într-un interval specific (de obicei [0, 1]), ceea ce poate fi util pentru algoritmi care presupun valori în acest interval sau pentru imagini, unde valorile pixelilor sunt adesea între 0 și 255.
  - Contra: Sensibilitate la valori aberante. Valorile extrem de mici sau extrem de mari pot afecta rezultatele scalării.
- Robust Scaling:
  - Pro: Robust la prezența valorilor aberante, deoarece se bazează pe mediană și interquartile range.
  - Contra: Poate fi dificil de interpretat pentru cei care nu sunt familiarizați cu conceptul de interquartile range.
- MaxAbs Scaling:
  - Pro: Scala valorile astfel încât cel mai mare absolut dintre toate elementele să fie 1. Poate fi util pentru seturile de date rare, cum ar fi cele folosite în tehnici de procesare a limbajului natural.
  - Contra: Poate fi sensibil la valori aberante și poate duce la pierderea informațiilor despre distribuția datelor.
- Unit Vector Scaling:
  - Pro: Scalare în așa fel încât norma fiecărui rând al setului de date să fie 1. Este utilă în special în contextul modelării vectorilor de termeni frecvenți.
  - Contra: Poate duce la pierderea informațiilor despre distribuția valorilor și poate fi sensibilă la valori aberante.

Definim imputers utilizati pentru tratarea valorilor lipsa

[45]:
```
imputers = {
    "SimpleImputer_median": SimpleImputer(strategy='median'),
    "IterativeImputer": IterativeImputer()
}
```

Exemple de abordări pentru Imputare

- SimpleImputer_median:
  - Pro: Simplu de utilizat și înțeles, imputează valorile lipsă folosind mediana fiecărei caracteristici (coloane).
  - Contra: Poate subestima variația datelor, deoarece imputează valori constante în locul valorilor lipsă, ceea ce poate să nu fie întotdeauna ideal în cazul unor seturi de date mai complexe.
- IterativeImputer:
  - Pro: Poate fi util pentru a estima și imputa valori lipsă bazate pe relațiile dintre caracteristici. Acesta imputează valori lipsă folosind iterații și modele predictivite.
  - Contra: Poate fi mai computațional intensiv și poate necesita mai mult timp pentru a fi antrenat și ajustat decât SimpleImputer. De asemenea, este mai complex din punct de vedere al implementării și interpretării rezultatelor.
- KNNImputer:
  - Pro: Poate estima valori lipsă bazate pe caracteristicile similare ale vecinilor cei mai apropiați. Este util pentru seturile de date în care există corelații între caracteristici.
  - Contra: Poate fi sensibil la mărimea vecinătății și la prezența unor valori aberante. De asemenea, poate fi computațional intensiv pentru seturile de date mari.
- MeanImputer:
  - Pro: Simplu de implementat și înțeles, imputează valorile lipsă folosind media fiecărei caracteristici.
  - Contra: Sensibilitate la valori aberante și potențialul de a distorsiona distribuția datelor, mai ales în cazul seturilor de date cu variație mare.
- MedianImputer:
  - Pro: Similar cu MeanImputer, dar folosește mediana în loc de media pentru a imputa valorile lipsă.
  - Contra: Poate fi mai robustă la prezența valorilor aberante decât MeanImputer, dar poate ignora variația și distribuția datelor.
- MostFrequentImputer:
  - Pro: Imputează valorile lipsă folosind cea mai frecventă valoare din fiecare caracteristică.
  - Contra: Poate introduce bias în date, în special în cazul în care valoarea cea mai frecventă este deja dominantă în setul de date.

Definim encoders utilizati pentru tratarea atributelor categorice

[46]:
```
class CustomOrdinalEncoder(BaseEstimator, TransformerMixin):
    def __init__(self, custom_mappings=None):
        self.custom_mappings = custom_mappings
        self.encoders = {}
```

```
    def fit(self, X, y=None):
        for column in X.columns:
            if column in self.custom_mappings:
                ordinal_encoder = OrdinalEncoder(categories=[list(self.
↪custom_mappings[column].keys())])
            else:
                ordinal_encoder = OrdinalEncoder()
            ordinal_encoder.fit(X[[column]])
            self.encoders[column] = ordinal_encoder
        return self

    def transform(self, X):
        encoded_data = X.copy()
        for column, encoder in self.encoders.items():
            encoded_data[column] = encoder.transform(X[[column]])
        return encoded_data

    def get_feature_names_out(self, input_features):
        for feature in input_features:
            if feature not in self.custom_mappings:
                print(f"Feature {feature} has no rule for encoding set")
        return input_features
```

Definim cum tratam din punct de vedere ordinal coloanele

[47]:
```
# Define custom mappings for ordinal columns
custom_mappings = {
    "Transportation": {"Walking": 0, "Bike": 1, "Motorbike": 2,␣
↪"Public_Transportation": 3, "Automobile": 4},
    "Alcohol": {"no": 0, "Sometimes": 1, "Frequently": 2, "Always": 3},
    "Snacks": {"no": 0, "Sometimes": 1, "Frequently": 2, "Always": 3},
    "Diagnostic": {"D0": 0, "D1": 1, "D2": 2, "D3": 3, "D4": 4, "D5": 5, "D6":␣
↪6}
}
```

[48]:
```
encoders = {
    "CountEncoder": ce.CountEncoder(),
    "CustomOrdinalEncoder" : CustomOrdinalEncoder(custom_mappings)
}
```

Exemple de abordări pentru Encodere

- CountEncoder:
  - Pro: Util pentru codificarea datelor categorice în câmpuri numerice, menținându-le formatul original.
  - Contra: Poate fi sensibil la variabilele rare sau la suprasolicitarea memoriei pentru seturile de date mari.

- OrdinalEncoder:
  - Pro: Converteste valorile categorice în numere ordonate, ceea ce poate fi util pentru algoritmii care folosesc datele ordonate.
  - Contra: Poate induce un ordin artificial între categorii care nu există în realitate sau care nu sunt relevante.
- OneHotEncoder:
  - Pro: Transformă variabilele categorice într-un format binar, fiecare categorie fiind reprezentată printr-un vector de valori binare.
  - Contra: Poate duce la creșterea dimensionalității setului de date, ceea ce poate afecta performanța modelului sau poate necesita mai multă memorie.
- TargetEncoder:
  - Pro: Codifică variabilele categorice folosind informații despre variabila țintă, ceea ce poate îmbunătăți performanța modelului.
  - Contra: Poate duce la overfitting în cazul seturilor de date mici sau poate introduce bias dacă există corelații puternice între variabilele țintă și predictor.

Definim selectors utilizati pentru selectarea feature-urilor

[49]:
```
selectors = {
    "VarianceThreshold": VarianceThreshold(),
    "SelectPercentile": SelectPercentile(percentile=50),
}
```

Exemple de abordări pentru Selectori

- VarianceThreshold:
  - Pro: Elimină caracteristicile cu variație scăzută, ceea ce poate fi util pentru reducerea dimensionalității setului de date și eliminarea caracteristicilor constante sau aproape constante.
  - Contra: Nu ia în considerare relația dintre caracteristici și variabila țintă, ceea ce poate duce la eliminarea caracteristicilor relevante pentru predicție.
- SelectPercentile:
  - Pro: Selectează caracteristicile pe baza unei măsuri de importanță, cum ar fi scorurile de testare, și reține un procentaj specific de cele mai relevante caracteristici.
  - Contra: Poate fi sensibil la overfitting dacă nu este folosită o validare încrucișată adecvată. De asemenea, poate ignora interacțiunile între caracteristici, selectând doar cele mai relevante caracteristici individual.

Definim clasificatorii utilizati

[50]:
```
classifiers = {
    "RandomForestClassifier": RandomForestClassifier(),
    "ExtraTreesClassifier": ExtraTreesClassifier(),
    "SVC": SVC(),
    "GradientBoostingClassifier": GradientBoostingClassifier()
}
```

Exemple de clasificatori

- RandomForestClassifier:
  - Pro: Potrivit pentru seturi de date mari, dimensionalități mari și date cu caracteristici diverse. Poate gestiona valori lipsă și valori aberante. Poate oferi importanța caracteristicilor.
  - Contra: Poate fi predispus la overfitting dacă nu sunt stabilite corect hiperparametrii. Necesită mai mult timp pentru antrenare decât alte modele mai simple.
- ExtraTreesClassifier:
  - Pro: Similar cu RandomForestClassifier, dar se bazează pe mai multe arbori de decizie și selectează caracteristicile la întâmplare, ceea ce poate duce la o reducere a variabilității și a overfitting-ului.
  - Contra: Poate fi mai puțin interpretabil decât alte modele, deoarece nu oferă importanța caracteristicilor la fel de clar ca RandomForestClassifier.
- SVC (Support Vector Classifier):
  - Pro: Potrivit pentru seturile de date cu o separare liniară sau aproape liniară între clase. Poate fi eficient pentru seturi de date mici sau medii.
  - Contra: Sensibil la scala valorilor caracteristicilor și la selecția de parametri. Poate fi dificil de interpretat și de configurat.
- GradientBoostingClassifier:
  - Pro: Potrivit pentru seturi de date cu o mare variabilitate și cu relații complexe între caracteristici și variabila țintă. Poate gestiona valori lipsă și valori aberante. Poate oferi importanța caracteristicilor.
  - Contra: Sensibil la overfitting dacă nu sunt stabilite hiperparametrii adecvați. Poate necesita mai mult timp pentru antrenare decât modelele mai simple.

Definim hiper-parametrii pentru fiecare clasificator

[51]:
```
param_grids = {
    "RandomForestClassifier": {
        "n_estimators": [10, 50, 100],
        "max_depth": [None, 10, 20],
        "max_features": [0.5, 0.75, 1.0]
    },
    "ExtraTreesClassifier": {
        "n_estimators": [10, 50, 100],
        "max_depth": [None, 10, 20],
        "max_features": [0.5, 0.75, 1.0]
    },
    "SVC": {
        "kernel": ['linear', 'poly', 'rbf', 'sigmoid'],
        "C": [0.1, 1, 10]
    },
    "GradientBoostingClassifier": {
        "n_estimators": [10, 50, 100],
        "max_depth": [3, 5, 7],
        "learning_rate": [0.01, 0.1, 0.5]
    }
}
```

[52]:
```
try:
    results = compare_classify_configurations(df, target_column, scalers,␣
 ↪imputers, encoders, selectors, classifiers, param_grids)
except Exception as e:
    # Print the traceback if an exception occurs
    traceback.print_exc()

results_list = list(results.items())

# Sort the list descending by Accuracy
sorted_results = sorted(results_list, key=lambda x:␣
 ↪x[1]['Results']['Accuracy'], reverse=True)
```

```
Testing configuration 1/32: MinMaxScaler_SimpleImputer_median_CountEncoder_Varia
nceThreshold_RandomForestClassifier
Duration: 27.146404266357422 seconds
Testing configuration 2/32: MinMaxScaler_SimpleImputer_median_CountEncoder_Varia
nceThreshold_ExtraTreesClassifier
Duration: 12.440772294998169 seconds
Testing configuration 3/32:
MinMaxScaler_SimpleImputer_median_CountEncoder_VarianceThreshold_SVC
Duration: 5.270175218582153 seconds
Testing configuration 4/32: MinMaxScaler_SimpleImputer_median_CountEncoder_Varia
nceThreshold_GradientBoostingClassifier
Duration: 230.5630021095276 seconds
Testing configuration 5/32: MinMaxScaler_SimpleImputer_median_CountEncoder_Selec
tPercentile_RandomForestClassifier
Duration: 15.482861995697021 seconds
Testing configuration 6/32: MinMaxScaler_SimpleImputer_median_CountEncoder_Selec
tPercentile_ExtraTreesClassifier
Duration: 10.068654775619507 seconds
Testing configuration 7/32:
MinMaxScaler_SimpleImputer_median_CountEncoder_SelectPercentile_SVC
Duration: 7.104823589324951 seconds
Testing configuration 8/32: MinMaxScaler_SimpleImputer_median_CountEncoder_Selec
tPercentile_GradientBoostingClassifier
Duration: 155.3911647796631 seconds
Testing configuration 9/32: MinMaxScaler_SimpleImputer_median_CustomOrdinalEncod
er_VarianceThreshold_RandomForestClassifier
Duration: 24.86739182472229 seconds
Testing configuration 10/32: MinMaxScaler_SimpleImputer_median_CustomOrdinalEnco
der_VarianceThreshold_ExtraTreesClassifier
Duration: 15.25394082069397 seconds
Testing configuration 11/32:
MinMaxScaler_SimpleImputer_median_CustomOrdinalEncoder_VarianceThreshold_SVC
Duration: 8.681888103485107 seconds
Testing configuration 12/32: MinMaxScaler_SimpleImputer_median_CustomOrdinalEnco
```

```
der_VarianceThreshold_GradientBoostingClassifier
Duration: 352.8365955352783 seconds
Testing configuration 13/32: MinMaxScaler_SimpleImputer_median_CustomOrdinalEnco
der_SelectPercentile_RandomForestClassifier
Duration: 17.13239288330078 seconds
Testing configuration 14/32: MinMaxScaler_SimpleImputer_median_CustomOrdinalEnco
der_SelectPercentile_ExtraTreesClassifier
Duration: 11.94562578201294 seconds
Testing configuration 15/32:
MinMaxScaler_SimpleImputer_median_CustomOrdinalEncoder_SelectPercentile_SVC
Duration: 7.252595901489258 seconds
Testing configuration 16/32: MinMaxScaler_SimpleImputer_median_CustomOrdinalEnco
der_SelectPercentile_GradientBoostingClassifier
Duration: 159.50502681732178 seconds
Testing configuration 17/32: MinMaxScaler_IterativeImputer_CountEncoder_Variance
Threshold_RandomForestClassifier
Duration: 28.48652696609497 seconds
Testing configuration 18/32: MinMaxScaler_IterativeImputer_CountEncoder_Variance
Threshold_ExtraTreesClassifier
Duration: 15.92652153968811 seconds
Testing configuration 19/32:
MinMaxScaler_IterativeImputer_CountEncoder_VarianceThreshold_SVC
Duration: 6.766685247421265 seconds
Testing configuration 20/32: MinMaxScaler_IterativeImputer_CountEncoder_Variance
Threshold_GradientBoostingClassifier
Duration: 242.10566234588623 seconds
Testing configuration 21/32: MinMaxScaler_IterativeImputer_CountEncoder_SelectPe
rcentile_RandomForestClassifier
Duration: 18.249093055725098 seconds
Testing configuration 22/32:
MinMaxScaler_IterativeImputer_CountEncoder_SelectPercentile_ExtraTreesClassifier
Duration: 12.241750717163086 seconds
Testing configuration 23/32:
MinMaxScaler_IterativeImputer_CountEncoder_SelectPercentile_SVC
Duration: 9.300937175750732 seconds
Testing configuration 24/32: MinMaxScaler_IterativeImputer_CountEncoder_SelectPe
rcentile_GradientBoostingClassifier
Duration: 163.56428050994873 seconds
Testing configuration 25/32: MinMaxScaler_IterativeImputer_CustomOrdinalEncoder_
VarianceThreshold_RandomForestClassifier
Duration: 26.717425107955933 seconds
Testing configuration 26/32: MinMaxScaler_IterativeImputer_CustomOrdinalEncoder_
VarianceThreshold_ExtraTreesClassifier
Duration: 12.945031642913818 seconds
Testing configuration 27/32:
MinMaxScaler_IterativeImputer_CustomOrdinalEncoder_VarianceThreshold_SVC
Duration: 5.614708423614502 seconds
Testing configuration 28/32: MinMaxScaler_IterativeImputer_CustomOrdinalEncoder_
```

```
VarianceThreshold_GradientBoostingClassifier
Duration: 242.673513174057 seconds
Testing configuration 29/32: MinMaxScaler_IterativeImputer_CustomOrdinalEncoder_
SelectPercentile_RandomForestClassifier
Duration: 16.39113187789917 seconds
Testing configuration 30/32: MinMaxScaler_IterativeImputer_CustomOrdinalEncoder_
SelectPercentile_ExtraTreesClassifier
Duration: 10.14215874671936 seconds
Testing configuration 31/32:
MinMaxScaler_IterativeImputer_CustomOrdinalEncoder_SelectPercentile_SVC
Duration: 7.126906871795654 seconds
Testing configuration 32/32: MinMaxScaler_IterativeImputer_CustomOrdinalEncoder_
SelectPercentile_GradientBoostingClassifier
Duration: 172.72219896316528 seconds
```

Afișarea rezultatelor sortate

[57]:
```python
for index, details in enumerate(sorted_results):
    config = details[0]
    result = details[1]["Results"]
    duration = details[1]["Duration"]
    print(f"Configuration: {config}")
    print(f"Accuracy: {result['Accuracy']}")
    print(f"Best Parameters: {result['Best Parameters']}")
    print(f"Precision: {result['Precision']}")
    print(f"Recall: {result['Recall']}")
    print(f"F1: {result['F1']}")
    print(f"Duration (s): {duration}")
    print(f"Confusion Matrix:\n{result['Confusion Matrix']}")
    print("-" * 150)
```

```
Configuration: ('MinMaxScaler', 'SimpleImputer_median', 'CustomOrdinalEncoder',
'VarianceThreshold', 'ExtraTreesClassifier')
Accuracy: 0.94
Best Parameters: {'max_depth': None, 'max_features': 1.0, 'n_estimators': 100}
Precision: [0.981 0.833 0.906 0.98  0.926 0.966 0.981]
Recall: [0.981 0.87  0.906 0.893 0.955 0.982 0.981]
F1: [0.981 0.851 0.906 0.935 0.94  0.974 0.981]
Duration (s): 15.254
Confusion Matrix:
[[52  1  0  0  0  0  0]
 [ 1 40  3  0  1  0  1]
 [ 0  4 48  0  1  0  0]
 [ 0  2  2 50  2  0  0]
 [ 0  1  0  1 63  1  0]
 [ 0  0  0  0  1 56  0]
 [ 0  0  0  0  0  1 53]]
----------------------------------------------------------------------------------------------------
```

```
----------------------------------------------------------------------
Configuration: ('MinMaxScaler', 'IterativeImputer', 'CustomOrdinalEncoder',
'VarianceThreshold', 'GradientBoostingClassifier')
Accuracy: 0.932
Best Parameters: {'learning_rate': 0.5, 'max_depth': 5, 'n_estimators': 100}
Precision: [0.942 0.863 0.933 0.935 0.869 1.    0.982]
Recall: [0.925 0.898 0.875 0.935 0.964 0.921 1.   ]
F1: [0.933 0.88  0.903 0.935 0.914 0.959 0.991]
Duration (s): 242.674
Confusion Matrix:
[[49  4  0  0  0  0  0]
 [ 2 44  2  0  0  0  1]
 [ 0  1 42  1  4  0  0]
 [ 1  2  1 58  0  0  0]
 [ 0  0  0  2 53  0  0]
 [ 0  0  0  1  4 58  0]
 [ 0  0  0  0  0  0 55]]
----------------------------------------------------------------------------
----------------------------------------------------------------------
Configuration: ('MinMaxScaler', 'SimpleImputer_median', 'CountEncoder',
'VarianceThreshold', 'GradientBoostingClassifier')
Accuracy: 0.927
Best Parameters: {'learning_rate': 0.5, 'max_depth': 5, 'n_estimators': 100}
Precision: [0.938 0.92  0.94  0.889 0.894 0.94  0.97 ]
Recall: [0.938 0.885 0.94  0.923 0.922 0.904 0.97 ]
F1: [0.938 0.902 0.94  0.906 0.908 0.922 0.97 ]
Duration (s): 230.563
Confusion Matrix:
[[45  2  0  0  0  0  1]
 [ 2 46  1  2  1  0  0]
 [ 0  2 47  0  1  0  0]
 [ 0  0  2 48  1  1  0]
 [ 0  0  0  3 59  2  0]
 [ 1  0  0  1  2 47  1]
 [ 0  0  0  0  2  0 65]]
----------------------------------------------------------------------------
----------------------------------------------------------------------
Configuration: ('MinMaxScaler', 'IterativeImputer', 'CountEncoder',
'VarianceThreshold', 'GradientBoostingClassifier')
Accuracy: 0.917
Best Parameters: {'learning_rate': 0.5, 'max_depth': 5, 'n_estimators': 100}
Precision: [0.93  0.875 0.84  0.821 0.963 0.955 1.   ]
Recall: [1.    0.824 0.808 0.958 0.881 0.955 0.985]
F1: [0.964 0.848 0.824 0.885 0.92  0.955 0.993]
Duration (s): 242.106
Confusion Matrix:
[[40  0  0  0  0  0  0]
 [ 2 42  6  1  0  0  0]
```

```
 [ 0  4 42  5  1  0  0]
 [ 0  2  0 46  0  0  0]
 [ 1  0  1  3 52  2  0]
 [ 0  0  1  1  1 64  0]
 [ 0  0  0  0  0  1 67]]
--------------------------------------------------------------------------------
--------------------------------------------------------------------
Configuration: ('MinMaxScaler', 'IterativeImputer', 'CustomOrdinalEncoder',
'VarianceThreshold', 'RandomForestClassifier')
Accuracy: 0.917
Best Parameters: {'max_depth': 20, 'max_features': 0.5, 'n_estimators': 100}
Precision: [0.95  0.849 0.889 0.911 0.928 0.956 0.941]
Recall: [0.927 0.882 0.857 0.911 0.889 0.956 1.   ]
F1: [0.938 0.865 0.873 0.911 0.908 0.956 0.97 ]
Duration (s): 26.717
Confusion Matrix:
[[38  3  0  0  0  0  0]
 [ 2 45  3  1  0  0  0]
 [ 0  4 48  2  1  0  1]
 [ 0  0  2 51  3  0  0]
 [ 0  0  1  2 64  2  3]
 [ 0  1  0  0  1 43  0]
 [ 0  0  0  0  0  0 64]]
--------------------------------------------------------------------------------
--------------------------------------------------------------------
Configuration: ('MinMaxScaler', 'SimpleImputer_median', 'CustomOrdinalEncoder',
'VarianceThreshold', 'GradientBoostingClassifier')
Accuracy: 0.914
Best Parameters: {'learning_rate': 0.5, 'max_depth': 5, 'n_estimators': 100}
Precision: [0.922 0.849 0.851 0.946 0.938 0.945 0.932]
Recall: [0.959 0.849 0.816 0.841 0.952 1.    0.982]
F1: [0.94  0.849 0.833 0.891 0.945 0.972 0.957]
Duration (s): 352.837
Confusion Matrix:
[[47  2  0  0  0  0  0]
 [ 1 45  4  0  0  0  3]
 [ 1  5 40  3  0  0  0]
 [ 2  1  1 53  3  2  1]
 [ 0  0  2  0 60  1  0]
 [ 0  0  0  0  0 52  0]
 [ 0  0  0  0  1  0 55]]
--------------------------------------------------------------------------------
--------------------------------------------------------------------
Configuration: ('MinMaxScaler', 'IterativeImputer', 'CountEncoder',
'VarianceThreshold', 'RandomForestClassifier')
Accuracy: 0.909
Best Parameters: {'max_depth': 10, 'max_features': 0.75, 'n_estimators': 100}
Precision: [0.981 0.774 0.833 0.936 0.95  0.921 0.981]
```

```
Recall: [0.927 0.941 0.784 0.863 0.851 1.    1.   ]
F1: [0.953 0.85  0.808 0.898 0.898 0.959 0.99 ]
Duration (s): 28.487
Confusion Matrix:
[[51  4  0  0  0  0  0]
 [ 0 48  2  0  0  0  1]
 [ 0  8 40  2  1  0  0]
 [ 0  1  2 44  2  2  0]
 [ 1  1  4  1 57  3  0]
 [ 0  0  0  0  0 58  0]
 [ 0  0  0  0  0  0 52]]
--------------------------------------------------------------------------------
--------------------------------------------------------------------
Configuration: ('MinMaxScaler', 'SimpleImputer_median', 'CountEncoder',
'VarianceThreshold', 'RandomForestClassifier')
Accuracy: 0.896
Best Parameters: {'max_depth': None, 'max_features': 0.5, 'n_estimators': 100}
Precision: [0.932 0.737 0.923 0.879 0.885 0.983 0.937]
Recall: [0.872 0.894 0.787 0.85  0.958 0.921 1.   ]
F1: [0.901 0.808 0.85  0.864 0.92  0.951 0.967]
Duration (s): 27.146
Confusion Matrix:
[[41  5  0  0  0  0  1]
 [ 1 42  1  1  0  0  2]
 [ 1  8 48  2  1  0  1]
 [ 0  1  3 51  4  1  0]
 [ 0  1  0  1 46  0  0]
 [ 1  0  0  3  1 58  0]
 [ 0  0  0  0  0  0 59]]
--------------------------------------------------------------------------------
--------------------------------------------------------------------
Configuration: ('MinMaxScaler', 'IterativeImputer', 'CustomOrdinalEncoder',
'VarianceThreshold', 'ExtraTreesClassifier')
Accuracy: 0.891
Best Parameters: {'max_depth': 20, 'max_features': 1.0, 'n_estimators': 100}
Precision: [0.891 0.682 0.791 0.921 1.    0.965 1.   ]
Recall: [0.932 0.882 0.773 0.853 0.87  0.965 0.977]
F1: [0.911 0.769 0.782 0.885 0.931 0.965 0.989]
Duration (s): 12.945
Confusion Matrix:
[[41  2  0  1  0  0  0]
 [ 3 45  2  1  0  0  0]
 [ 0  9 34  1  0  0  0]
 [ 0  5  4 58  0  1  0]
 [ 1  3  3  2 67  1  0]
 [ 1  1  0  0  0 55  0]
 [ 0  1  0  0  0  0 43]]
--------------------------------------------------------------------------------
```

```
----------------------------------------------------------------------
Configuration: ('MinMaxScaler', 'SimpleImputer_median', 'CustomOrdinalEncoder',
'VarianceThreshold', 'RandomForestClassifier')
Accuracy: 0.886
Best Parameters: {'max_depth': None, 'max_features': 0.5, 'n_estimators': 100}
Precision: [0.87  0.818 0.923 0.878 0.892 0.915 0.896]
Recall: [0.93  0.849 0.842 0.8   0.841 0.931 1.   ]
F1: [0.899 0.833 0.881 0.837 0.866 0.923 0.945]
Duration (s): 24.867
Confusion Matrix:
[[40  1  0  0  1  0  1]
 [ 3 45  2  0  0  0  3]
 [ 1  4 48  2  1  0  1]
 [ 0  5  1 36  2  1  0]
 [ 2  0  1  3 58  4  1]
 [ 0  0  0  0  3 54  1]
 [ 0  0  0  0  0  0 60]]
----------------------------------------------------------------------------
----------------------------------------------------------------------
Configuration: ('MinMaxScaler', 'SimpleImputer_median', 'CountEncoder',
'VarianceThreshold', 'ExtraTreesClassifier')
Accuracy: 0.883
Best Parameters: {'max_depth': None, 'max_features': 0.75, 'n_estimators': 100}
Precision: [0.941 0.809 0.696 0.8   0.929 1.    0.968]
Recall: [0.96  0.644 0.812 0.8   0.981 1.    0.968]
F1: [0.95  0.717 0.75  0.8   0.954 1.    0.968]
Duration (s): 12.441
Confusion Matrix:
[[48  2  0  0  0  0  0]
 [ 1 38 12  6  2  0  0]
 [ 0  3 39  4  2  0  0]
 [ 1  3  5 40  0  0  1]
 [ 0  0  0  0 52  0  1]
 [ 0  0  0  0  0 63  0]
 [ 1  1  0  0  0  0 60]]
----------------------------------------------------------------------------
----------------------------------------------------------------------
Configuration: ('MinMaxScaler', 'IterativeImputer', 'CountEncoder',
'VarianceThreshold', 'ExtraTreesClassifier')
Accuracy: 0.883
Best Parameters: {'max_depth': None, 'max_features': 1.0, 'n_estimators': 50}
Precision: [0.936 0.686 0.855 0.846 0.912 0.967 0.952]
Recall: [0.83  0.745 0.825 0.917 0.867 0.967 1.   ]
F1: [0.88  0.714 0.839 0.88  0.889 0.967 0.976]
Duration (s): 15.927
Confusion Matrix:
[[44  7  0  2  0  0  0]
 [ 3 35  5  2  0  0  2]
```

```
 [ 0  6 47  1  3  0  0]
 [ 0  1  0 44  2  1  0]
 [ 0  1  3  2 52  1  1]
 [ 0  1  0  1  0 58  0]
 [ 0  0  0  0  0  0 60]]
--------------------------------------------------------------------------------
--------------------------------------------------------------------
Configuration: ('MinMaxScaler', 'SimpleImputer_median', 'CountEncoder',
'SelectPercentile', 'ExtraTreesClassifier')
Accuracy: 0.696
Best Parameters: {'max_depth': None, 'max_features': 0.75, 'n_estimators': 100}
Precision: [0.774 0.625 0.577 0.596 0.692 0.754 0.812]
Recall: [0.837 0.636 0.517 0.509 0.706 0.78  0.897]
F1: [0.804 0.631 0.545 0.549 0.699 0.767 0.852]
Duration (s): 10.069
Confusion Matrix:
[[41  0  2  0  2  1  3]
 [ 5 35  6  6  3  0  0]
 [ 3 10 30  6  3  4  2]
 [ 1  7  4 28  7  7  1]
 [ 0  2  6  3 36  1  3]
 [ 2  2  3  3  0 46  3]
 [ 1  0  1  1  1  2 52]]
--------------------------------------------------------------------------------
--------------------------------------------------------------------
Configuration: ('MinMaxScaler', 'IterativeImputer', 'CountEncoder',
'SelectPercentile', 'RandomForestClassifier')
Accuracy: 0.683
Best Parameters: {'max_depth': 20, 'max_features': 0.5, 'n_estimators': 100}
Precision: [0.769 0.66  0.585 0.5   0.705 0.636 0.838]
Recall: [0.702 0.729 0.48  0.5   0.672 0.724 0.891]
F1: [0.734 0.693 0.527 0.5   0.688 0.677 0.864]
Duration (s): 18.249
Confusion Matrix:
[[40  5  5  2  0  5  0]
 [ 5 35  1  6  1  0  0]
 [ 3  6 24  2  6  6  3]
 [ 2  3  4 22  8  3  2]
 [ 0  3  4  4 43  7  3]
 [ 1  1  3  6  2 42  3]
 [ 1  0  0  2  1  3 57]]
--------------------------------------------------------------------------------
--------------------------------------------------------------------
Configuration: ('MinMaxScaler', 'SimpleImputer_median', 'CustomOrdinalEncoder',
'SelectPercentile', 'ExtraTreesClassifier')
Accuracy: 0.678
Best Parameters: {'max_depth': 20, 'max_features': 0.75, 'n_estimators': 100}
Precision: [0.825 0.571 0.574 0.581 0.661 0.712 0.765]
```

```
Recall: [0.77  0.711 0.551 0.51  0.672 0.597 0.897]
F1: [0.797 0.634 0.562 0.543 0.667 0.649 0.825]
Duration (s): 11.946
Confusion Matrix:
[[47  4  3  2  2  0  3]
 [ 5 32  2  4  2  0  0]
 [ 2  4 27  2  4  6  4]
 [ 1  5  4 25  7  4  3]
 [ 0  5  5  5 41  2  3]
 [ 1  5  5  5  6 37  3]
 [ 1  1  1  0  0  3 52]]
--------------------------------------------------------------------------------
--------------------------------------------------------------------
Configuration: ('MinMaxScaler', 'IterativeImputer', 'CustomOrdinalEncoder',
'SelectPercentile', 'RandomForestClassifier')
Accuracy: 0.668
Best Parameters: {'max_depth': 10, 'max_features': 0.75, 'n_estimators': 100}
Precision: [0.711 0.603 0.628 0.545 0.631 0.755 0.764]
Recall: [0.744 0.623 0.474 0.444 0.661 0.8   0.948]
F1: [0.727 0.613 0.54  0.49  0.646 0.777 0.846]
Duration (s): 16.391
Confusion Matrix:
[[32  4  2  0  2  0  3]
 [ 5 38  5  9  4  0  0]
 [ 6  7 27  4  8  4  1]
 [ 1  7  6 24  7  6  3]
 [ 0  5  2  6 41  2  6]
 [ 1  2  0  1  2 40  4]
 [ 0  0  1  0  1  1 55]]
--------------------------------------------------------------------------------
--------------------------------------------------------------------
Configuration: ('MinMaxScaler', 'IterativeImputer', 'CountEncoder',
'SelectPercentile', 'ExtraTreesClassifier')
Accuracy: 0.66
Best Parameters: {'max_depth': 20, 'max_features': 0.5, 'n_estimators': 50}
Precision: [0.744 0.556 0.536 0.605 0.667 0.672 0.847]
Recall: [0.627 0.745 0.566 0.51  0.594 0.729 0.833]
F1: [0.681 0.636 0.55  0.553 0.628 0.699 0.84 ]
Duration (s): 12.242
Confusion Matrix:
[[32  7  2  0  4  2  4]
 [ 4 35  3  2  2  1  0]
 [ 3  5 30  5  7  2  1]
 [ 3  4  7 26  3  7  1]
 [ 0  7 11  2 38  4  2]
 [ 1  4  2  6  2 43  1]
 [ 0  1  1  2  1  5 50]]
--------------------------------------------------------------------------------
```

```
----------------------------------------------------------------------
Configuration: ('MinMaxScaler', 'SimpleImputer_median', 'CountEncoder',
'SelectPercentile', 'RandomForestClassifier')
Accuracy: 0.636
Best Parameters: {'max_depth': 10, 'max_features': 0.5, 'n_estimators': 100}
Precision: [0.868 0.535 0.513 0.51  0.692 0.633 0.724]
Recall: [0.611 0.745 0.408 0.417 0.562 0.792 0.932]
F1: [0.717 0.623 0.455 0.459 0.621 0.704 0.815]
Duration (s): 15.483
Confusion Matrix:
[[33  9  2  4  3  3  0]
 [ 2 38  4  6  1  0  0]
 [ 0 10 20  6  4  4  5]
 [ 2  4  8 25  8  7  6]
 [ 0  5  5  6 36  6  6]
 [ 1  4  0  1  0 38  4]
 [ 0  1  0  1  0  2 55]]
--------------------------------------------------------------------------------
----------------------------------------------------------------------
Configuration: ('MinMaxScaler', 'IterativeImputer', 'CustomOrdinalEncoder',
'SelectPercentile', 'ExtraTreesClassifier')
Accuracy: 0.629
Best Parameters: {'max_depth': 20, 'max_features': 0.75, 'n_estimators': 100}
Precision: [0.727 0.578 0.511 0.538 0.667 0.696 0.647]
Recall: [0.667 0.617 0.442 0.429 0.687 0.684 0.846]
F1: [0.696 0.597 0.474 0.477 0.676 0.69  0.733]
Duration (s): 10.142
Confusion Matrix:
[[32  4  2  1  1  1  7]
 [ 3 37  6  7  6  1  0]
 [ 3  5 23  3  6  6  6]
 [ 1  9  4 21  7  4  3]
 [ 2  6  3  6 46  2  2]
 [ 3  3  4  1  1 39  6]
 [ 0  0  3  0  2  3 44]]
--------------------------------------------------------------------------------
----------------------------------------------------------------------
Configuration: ('MinMaxScaler', 'IterativeImputer', 'CustomOrdinalEncoder',
'SelectPercentile', 'GradientBoostingClassifier')
Accuracy: 0.629
Best Parameters: {'learning_rate': 0.1, 'max_depth': 5, 'n_estimators': 100}
Precision: [0.788 0.61  0.471 0.544 0.631 0.608 0.746]
Recall: [0.605 0.893 0.356 0.525 0.512 0.608 0.922]
F1: [0.684 0.725 0.405 0.534 0.566 0.608 0.825]
Duration (s): 172.722
Confusion Matrix:
[[26  9  2  1  4  0  1]
 [ 0 50  0  3  3  0  0]
```

```
 [ 2  8 16  5  6  5  3]
 [ 1  6  4 31  9  5  3]
 [ 2  6  7 12 41  9  3]
 [ 1  1  5  5  2 31  6]
 [ 1  2  0  0  0  1 47]]
--------------------------------------------------------------------------------
--------------------------------------------------------------------
Configuration: ('MinMaxScaler', 'SimpleImputer_median', 'CountEncoder',
'SelectPercentile', 'GradientBoostingClassifier')
Accuracy: 0.613
Best Parameters: {'learning_rate': 0.1, 'max_depth': 5, 'n_estimators': 50}
Precision: [0.73  0.583 0.5   0.512 0.58  0.606 0.782]
Recall: [0.519 0.724 0.358 0.468 0.635 0.754 0.782]
F1: [0.607 0.646 0.418 0.489 0.606 0.672 0.782]
Duration (s): 155.391
Confusion Matrix:
[[27  6 10  1  2  4  2]
 [ 5 42  0  7  4  0  0]
 [ 3 10 19  4 12  4  1]
 [ 1  4  3 22  5  7  5]
 [ 0  7  2  6 40  5  3]
 [ 1  3  3  3  3 43  1]
 [ 0  0  1  0  3  8 43]]
--------------------------------------------------------------------------------
--------------------------------------------------------------------
Configuration: ('MinMaxScaler', 'IterativeImputer', 'CustomOrdinalEncoder',
'SelectPercentile', 'SVC')
Accuracy: 0.608
Best Parameters: {'C': 10, 'kernel': 'rbf'}
Precision: [0.805 0.441 0.514 0.7   0.642 0.549 0.615]
Recall: [0.688 0.385 0.375 0.389 0.606 0.789 0.868]
F1: [0.742 0.411 0.434 0.5   0.623 0.647 0.72 ]
Duration (s): 7.127
Confusion Matrix:
[[33  4  1  1  2  0  7]
 [ 4 15  3  1  8  5  3]
 [ 2  5 18  0  4  9 10]
 [ 2  3  7 21 10  9  2]
 [ 0  4  5  5 43  5  9]
 [ 0  3  1  2  0 45  6]
 [ 0  0  0  0  0  9 59]]
--------------------------------------------------------------------------------
--------------------------------------------------------------------
Configuration: ('MinMaxScaler', 'SimpleImputer_median', 'CustomOrdinalEncoder',
'SelectPercentile', 'GradientBoostingClassifier')
Accuracy: 0.605
Best Parameters: {'learning_rate': 0.1, 'max_depth': 7, 'n_estimators': 100}
Precision: [0.717 0.629 0.429 0.373 0.5   0.656 0.817]
```

```
Recall: [0.611 0.75  0.36  0.38  0.481 0.714 0.841]
F1: [0.66  0.684 0.391 0.376 0.491 0.684 0.829]
Duration (s): 159.505
Confusion Matrix:
[[33  3  5  6  3  1  3]
 [ 3 39  4  3  2  1  0]
 [ 3  8 18  6  8  6  1]
 [ 2  7  8 19  5  6  3]
 [ 3  4  4 12 26  2  3]
 [ 1  1  2  5  4 40  3]
 [ 1  0  1  0  4  5 58]]
--------------------------------------------------------------------------------
--------------------------------------------------------------------
Configuration: ('MinMaxScaler', 'IterativeImputer', 'CountEncoder',
'SelectPercentile', 'GradientBoostingClassifier')
Accuracy: 0.597
Best Parameters: {'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 100}
Precision: [0.659 0.523 0.514 0.469 0.625 0.587 0.75 ]
Recall: [0.574 0.654 0.34  0.418 0.541 0.74  0.944]
F1: [0.614 0.581 0.409 0.442 0.58  0.655 0.836]
Duration (s): 163.564
Confusion Matrix:
[[27  7  1  3  4  2  3]
 [ 4 34  4  5  5  0  0]
 [ 6  8 18  4  8  7  2]
 [ 2  7  7 23  6  7  3]
 [ 1  7  5 11 40  7  3]
 [ 1  2  0  3  1 37  6]
 [ 0  0  0  0  0  3 51]]
--------------------------------------------------------------------------------
--------------------------------------------------------------------
Configuration: ('MinMaxScaler', 'SimpleImputer_median', 'CustomOrdinalEncoder',
'SelectPercentile', 'RandomForestClassifier')
Accuracy: 0.579
Best Parameters: {'max_depth': 20, 'max_features': 0.5, 'n_estimators': 100}
Precision: [0.564 0.484 0.608 0.5   0.388 0.7   0.766]
Recall: [0.537 0.674 0.456 0.518 0.442 0.609 0.79 ]
F1: [0.55  0.564 0.521 0.509 0.413 0.651 0.778]
Duration (s): 17.132
Confusion Matrix:
[[22  9  2  3  3  1  1]
 [ 3 31  3  3  6  0  0]
 [ 6  8 31  7  8  4  4]
 [ 3  6  4 29  7  4  3]
 [ 1  9  2  5 19  4  3]
 [ 0  0  7 11  5 42  4]
 [ 4  1  2  0  1  5 49]]
--------------------------------------------------------------------------------
```

```
----------------------------------------------------------------------
Configuration: ('MinMaxScaler', 'SimpleImputer_median', 'CustomOrdinalEncoder',
'SelectPercentile', 'SVC')
Accuracy: 0.571
Best Parameters: {'C': 10, 'kernel': 'rbf'}
Precision: [0.705 0.6   0.632 0.421 0.62  0.472 0.573]
Recall: [0.674 0.421 0.444 0.32  0.611 0.708 0.81 ]
F1: [0.689 0.495 0.522 0.364 0.615 0.567 0.671]
Duration (s): 7.253
Confusion Matrix:
[[31  2  3  0  4  3  3]
 [ 7 24  5  5 10  4  2]
 [ 3  2 24  5  6  9  5]
 [ 1  6  6 16  7  7  7]
 [ 1  4  0 11 44  4  8]
 [ 1  2  0  1  0 34 10]
 [ 0  0  0  0  0 11 47]]
----------------------------------------------------------------------------
----------------------------------------------------------------------
Configuration: ('MinMaxScaler', 'IterativeImputer', 'CountEncoder',
'SelectPercentile', 'SVC')
Accuracy: 0.571
Best Parameters: {'C': 10, 'kernel': 'rbf'}
Precision: [0.667 0.512 0.481 0.476 0.57  0.625 0.59 ]
Recall: [0.553 0.431 0.255 0.364 0.634 0.789 0.925]
F1: [0.605 0.468 0.333 0.412 0.6   0.698 0.721]
Duration (s): 9.301
Confusion Matrix:
[[26  7  4  1  3  2  4]
 [ 7 22  3  6  8  1  4]
 [ 3  4 13  5  7 11  8]
 [ 1  8  3 20 15  4  4]
 [ 0  1  3  9 45  5  8]
 [ 2  1  1  1  1 45  6]
 [ 0  0  0  0  0  4 49]]
----------------------------------------------------------------------------
----------------------------------------------------------------------
Configuration: ('MinMaxScaler', 'SimpleImputer_median', 'CountEncoder',
'SelectPercentile', 'SVC')
Accuracy: 0.551
Best Parameters: {'C': 10, 'kernel': 'rbf'}
Precision: [0.64  0.655 0.522 0.472 0.427 0.549 0.627]
Recall: [0.667 0.38  0.214 0.309 0.556 0.833 0.881]
F1: [0.653 0.481 0.304 0.374 0.483 0.662 0.732]
Duration (s): 7.105
Confusion Matrix:
[[32  3  0  4  2  3  4]
 [ 9 19  2  5  8  4  3]
```

```
 [ 5  2 12  0 18 11  8]
 [ 2  3  5 17 19  5  4]
 [ 1  0  3  8 35  8  8]
 [ 1  2  1  1  0 45  4]
 [ 0  0  0  1  0  6 52]]
--------------------------------------------------------------------------------
--------------------------------------------------------------------------
Configuration: ('MinMaxScaler', 'IterativeImputer', 'CountEncoder',
'VarianceThreshold', 'SVC')
Accuracy: 0.512
Best Parameters: {'C': 10, 'kernel': 'rbf'}
Precision: [0.711 0.556 0.357 0.436 0.522 0.364 0.532]
Recall: [0.744 0.339 0.182 0.378 0.522 0.34  0.985]
F1: [0.727 0.421 0.241 0.405 0.522 0.352 0.691]
Duration (s): 6.767
Confusion Matrix:
[[32  1  0  1  3  0  6]
 [ 6 20 11  3  8  3  8]
 [ 4  7 10  5 11 10  8]
 [ 0  5  1 17 10  6  6]
 [ 1  2  4 11 36  9  6]
 [ 2  0  2  2  1 16 24]
 [ 0  1  0  0  0  0 66]]
--------------------------------------------------------------------------------
--------------------------------------------------------------------------
Configuration: ('MinMaxScaler', 'IterativeImputer', 'CustomOrdinalEncoder',
'VarianceThreshold', 'SVC')
Accuracy: 0.509
Best Parameters: {'C': 10, 'kernel': 'rbf'}
Precision: [0.727 0.529 0.533 0.41  0.426 0.566 0.476]
Recall: [0.558 0.409 0.271 0.262 0.509 0.652 0.909]
F1: [0.632 0.462 0.36  0.32  0.464 0.606 0.625]
Duration (s): 5.615
Confusion Matrix:
[[24  6  2  4  2  1  4]
 [ 6 18  5  2  5  2  6]
 [ 2  2 16  7 10 11 11]
 [ 1  4  4 16 21  8  7]
 [ 0  3  2  7 29  6 10]
 [ 0  1  1  3  1 43 17]
 [ 0  0  0  0  0  5 50]]
--------------------------------------------------------------------------------
--------------------------------------------------------------------------
Configuration: ('MinMaxScaler', 'SimpleImputer_median', 'CountEncoder',
'VarianceThreshold', 'SVC')
Accuracy: 0.491
Best Parameters: {'C': 10, 'kernel': 'poly'}
Precision: [0.568 0.432 0.5   0.317 0.559 0.556 0.469]
```

```
Recall: [0.481 0.284 0.404 0.283 0.508 0.565 0.978]
F1: [0.521 0.342 0.447 0.299 0.532 0.56  0.634]
Duration (s): 5.27
Confusion Matrix:
[[25  9  1  4  3  0 10]
 [10 19  8  2  9  9 10]
 [ 5  2 19 10  2  5  4]
 [ 2  6  2 13  9  6  8]
 [ 2  3  6 10 33  7  4]
 [ 0  5  2  2  3 35 15]
 [ 0  0  0  0  0  1 45]]
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
Configuration: ('MinMaxScaler', 'SimpleImputer_median', 'CustomOrdinalEncoder',
'VarianceThreshold', 'SVC')
Accuracy: 0.481
Best Parameters: {'C': 10, 'kernel': 'rbf'}
Precision: [0.771 0.364 0.424 0.467 0.54  0.477 0.434]
Recall: [0.443 0.421 0.298 0.23  0.5   0.412 1.   ]
F1: [0.562 0.39  0.35  0.308 0.519 0.442 0.605]
Duration (s): 8.682
Confusion Matrix:
[[27 10  6  2  2  1 13]
 [ 6 16  3  1  6  0  6]
 [ 0  6 14  4  2 11 10]
 [ 1  6  6 14 18  4 12]
 [ 1  6  3  6 34  7 11]
 [ 0  0  1  3  1 21 25]
 [ 0  0  0  0  0  0 59]]
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
```

Salvam Top 5 configuratii separat

[54]:
```python
top_5_configurations = {}

for index, details in enumerate(sorted_results):
    if index < 5:
        config = details[0]
        result = details[1]["Results"]
        duration = details[1]["Duration"]

        top_5_configurations[config] = {
            "Accuracy": result['Accuracy'],
            "Best Parameters": result['Best Parameters'],
            "Precision": result['Precision'],
            "Recall": result['Recall'],
            "F1": result['F1'],
```

```
            "Duration": duration,
            "Confusion Matrix": result['Confusion Matrix']
        }
```

Rulam cele 5 configuratii de mai multe ori

[55]:
```
# Dictionary to store results run multiple times for each configuration in the top 5
results_for_top_5_configurations = {}

# Number of runs for each configuration
num_runs = 3

# Iterate over each configuration from the top 5
for config, details in top_5_configurations.items():
    # Initialize a list to store results for this configuration
    results_for_this_config = []

    scaler = scalers[config[0]]  # Get the scaler using the corresponding key from the configuration
    imputer = imputers[config[1]]  # Get the imputer using the corresponding key from the configuration
    encoder = encoders[config[2]]  # Get the encoder using the corresponding key from the configuration
    selector = selectors[config[3]]  # Get the selector using the corresponding key from the configuration
    classifier = classifiers[config[4]]  # Get the classifier using the corresponding key from the configuration
    param_grid = param_grids[config[4]]  # Get the parameter grid for the current classifier

    # Iterate over the specified number of runs for this configuration
    for run in range(num_runs):
        try:
          print(f"Configuration: {config} Run {run + 1} / {num_runs}")
          start_time = time.time()
          result = classify_data(df.copy().sample(frac=1).reset_index(drop=True), target_column, scaler, imputer, encoder, selector, classifier, param_grid)
          end_time = time.time()
          execution_time = end_time - start_time
          print(f"Duration: {execution_time} seconds")
        except Exception as e:
          # Print the traceback if an exception occurs
          traceback.print_exc()

        # Store the result and duration inside a dictionary
```

```
        result_with_duration = {"result": result, "duration": execution_time}

        # Add the results to the list for this configuration
        results_for_this_config.append(result_with_duration)

    # Add the list of results for this configuration to the final dictionary
    results_for_top_5_configurations[config] = results_for_this_config
```

```
Configuration: ('MinMaxScaler', 'SimpleImputer_median', 'CustomOrdinalEncoder',
'VarianceThreshold', 'ExtraTreesClassifier') Run 1 / 3
Duration: 13.572567701339722 seconds
Configuration: ('MinMaxScaler', 'SimpleImputer_median', 'CustomOrdinalEncoder',
'VarianceThreshold', 'ExtraTreesClassifier') Run 2 / 3
Duration: 13.296485424041748 seconds
Configuration: ('MinMaxScaler', 'SimpleImputer_median', 'CustomOrdinalEncoder',
'VarianceThreshold', 'ExtraTreesClassifier') Run 3 / 3
Duration: 12.426087856292725 seconds
Configuration: ('MinMaxScaler', 'IterativeImputer', 'CustomOrdinalEncoder',
'VarianceThreshold', 'GradientBoostingClassifier') Run 1 / 3
Duration: 239.0780987739563 seconds
Configuration: ('MinMaxScaler', 'IterativeImputer', 'CustomOrdinalEncoder',
'VarianceThreshold', 'GradientBoostingClassifier') Run 2 / 3
Duration: 234.12397861480713 seconds
Configuration: ('MinMaxScaler', 'IterativeImputer', 'CustomOrdinalEncoder',
'VarianceThreshold', 'GradientBoostingClassifier') Run 3 / 3
Duration: 240.16717863082886 seconds
Configuration: ('MinMaxScaler', 'SimpleImputer_median', 'CountEncoder',
'VarianceThreshold', 'GradientBoostingClassifier') Run 1 / 3
Duration: 237.08165621757507 seconds
Configuration: ('MinMaxScaler', 'SimpleImputer_median', 'CountEncoder',
'VarianceThreshold', 'GradientBoostingClassifier') Run 2 / 3
Duration: 232.0034921169281 seconds
Configuration: ('MinMaxScaler', 'SimpleImputer_median', 'CountEncoder',
'VarianceThreshold', 'GradientBoostingClassifier') Run 3 / 3
Duration: 237.31809067726135 seconds
Configuration: ('MinMaxScaler', 'IterativeImputer', 'CountEncoder',
'VarianceThreshold', 'GradientBoostingClassifier') Run 1 / 3
Duration: 234.2506618499756 seconds
Configuration: ('MinMaxScaler', 'IterativeImputer', 'CountEncoder',
'VarianceThreshold', 'GradientBoostingClassifier') Run 2 / 3
Duration: 245.5537302494049 seconds
Configuration: ('MinMaxScaler', 'IterativeImputer', 'CountEncoder',
'VarianceThreshold', 'GradientBoostingClassifier') Run 3 / 3
Duration: 247.46071863174438 seconds
Configuration: ('MinMaxScaler', 'IterativeImputer', 'CustomOrdinalEncoder',
'VarianceThreshold', 'RandomForestClassifier') Run 1 / 3
Duration: 26.78124237060547 seconds
```

```
Configuration: ('MinMaxScaler', 'IterativeImputer', 'CustomOrdinalEncoder',
'VarianceThreshold', 'RandomForestClassifier') Run 2 / 3
Duration: 28.169500827789307 seconds
Configuration: ('MinMaxScaler', 'IterativeImputer', 'CustomOrdinalEncoder',
'VarianceThreshold', 'RandomForestClassifier') Run 3 / 3
Duration: 26.111685514450073 seconds
```

Display the results for each configuration from the top 5

[56]:
```python
for config, results in results_for_top_5_configurations.items():
    print(f"\nResults for configuration: {config}")
    for i, result in enumerate(results):
        # Get desired data
        duration = result['duration']
        result = result['result']
        # Show the data
        print(f"Run {i + 1}:")
        print(f"Accuracy: {result['Accuracy']}")
        print(f"Best Parameters: {result['Best Parameters']}")
        print(f"Precision: {result['Precision']}")
        print(f"Recall: {result['Recall']}")
        print(f"F1: {result['F1']}")
        print(f"Duration: {duration} seconds"),
        print(f"Confusion Matrix:\n{result['Confusion Matrix']}")
        print("-" * 150)
    print("=" * 150)
```

```
Results for configuration: ('MinMaxScaler', 'SimpleImputer_median',
'CustomOrdinalEncoder', 'VarianceThreshold', 'ExtraTreesClassifier')
Run 1:
Accuracy: 0.906
Best Parameters: {'max_depth': None, 'max_features': 1.0, 'n_estimators': 50}
Precision: [0.964 0.746 0.879 0.952 0.896 0.974 0.984]
Recall: [0.93  0.922 0.864 0.727 0.968 0.925 1.   ]
F1: [0.946 0.825 0.872 0.825 0.93  0.949 0.992]
Duration: 13.572567701339722 seconds
Confusion Matrix:
[[53  3  0  0  0  0  1]
 [ 2 47  1  1  0  0  0]
 [ 0  5 51  0  3  0  0]
 [ 0  7  6 40  1  1  0]
 [ 0  1  0  1 60  0  0]
 [ 0  0  0  0  3 37  0]
 [ 0  0  0  0  0  0 61]]
------------------------------------------------------------------------------------
------------------------------------------------------------------
Run 2:
```

```
Accuracy: 0.87
Best Parameters: {'max_depth': None, 'max_features': 1.0, 'n_estimators': 100}
Precision: [0.897 0.712 0.922 0.809 0.884 0.951 0.962]
Recall: [0.897 0.797 0.734 0.826 0.897 1.    1.   ]
F1: [0.897 0.752 0.817 0.817 0.891 0.975 0.981]
Duration: 13.296485424041748 seconds
Confusion Matrix:
[[52  6  0  0  0  0  0]
 [ 6 47  2  2  1  0  1]
 [ 0  8 47  5  4  0  0]
 [ 0  4  1 38  3  0  0]
 [ 0  1  1  2 61  2  1]
 [ 0  0  0  0  0 39  0]
 [ 0  0  0  0  0  0 51]]
--------------------------------------------------------------------------------
--------------------------------------------------------------------
Run 3:
Accuracy: 0.883
Best Parameters: {'max_depth': 20, 'max_features': 0.75, 'n_estimators': 100}
Precision: [0.855 0.73  0.943 0.889 0.882 0.962 0.958]
Recall: [0.887 0.836 0.794 0.833 0.923 0.944 0.979]
F1: [0.87  0.78  0.862 0.86  0.902 0.953 0.968]
Duration: 12.426087856292725 seconds
Confusion Matrix:
[[47  5  0  0  0  0  1]
 [ 7 46  1  0  0  0  1]
 [ 1  5 50  2  4  1  0]
 [ 0  3  2 40  2  1  0]
 [ 0  2  0  3 60  0  0]
 [ 0  1  0  0  2 51  0]
 [ 0  1  0  0  0  0 46]]
--------------------------------------------------------------------------------
--------------------------------------------------------------------
================================================================================
====================================================================

Results for configuration: ('MinMaxScaler', 'IterativeImputer',
'CustomOrdinalEncoder', 'VarianceThreshold', 'GradientBoostingClassifier')
Run 1:
Accuracy: 0.93
Best Parameters: {'learning_rate': 0.5, 'max_depth': 7, 'n_estimators': 50}
Precision: [0.959 0.929 0.854 0.962 0.941 0.911 0.94 ]
Recall: [0.904 0.912 0.872 0.943 0.928 0.953 0.984]
F1: [0.931 0.92  0.863 0.952 0.934 0.932 0.962]
Duration: 239.0780987739563 seconds
Confusion Matrix:
[[47  1  1  0  0  0  3]
 [ 2 52  2  0  0  0  1]
```

```
 [ 0  2 41  2  2  0  0]
 [ 0  0  1 50  1  1  0]
 [ 0  1  2  0 64  2  0]
 [ 0  0  1  0  1 41  0]
 [ 0  0  0  0  0  1 63]]
--------------------------------------------------------------------------------
--------------------------------------------------------------------
Run 2:
Accuracy: 0.932
Best Parameters: {'learning_rate': 0.5, 'max_depth': 7, 'n_estimators': 100}
Precision: [0.936 0.839 0.816 0.964 0.967 1.    0.984]
Recall: [0.936 0.904 0.816 0.931 0.937 0.982 1.   ]
F1: [0.936 0.87  0.816 0.947 0.952 0.991 0.992]
Duration: 234.12397861480713 seconds
Confusion Matrix:
[[44  2  0  0  0  0  1]
 [ 2 47  2  1  0  0  0]
 [ 1  6 40  1  1  0  0]
 [ 0  1  3 54  0  0  0]
 [ 0  0  4  0 59  0  0]
 [ 0  0  0  0  1 54  0]
 [ 0  0  0  0  0  0 61]]
--------------------------------------------------------------------------------
--------------------------------------------------------------------
Run 3:
Accuracy: 0.906
Best Parameters: {'learning_rate': 0.5, 'max_depth': 5, 'n_estimators': 100}
Precision: [0.87  0.864 0.825 0.897 0.931 0.966 0.966]
Recall: [0.904 0.927 0.805 0.867 0.885 0.933 1.   ]
F1: [0.887 0.895 0.815 0.881 0.908 0.949 0.982]
Duration: 240.16717863082886 seconds
Confusion Matrix:
[[47  2  1  0  0  0  2]
 [ 2 51  2  0  0  0  0]
 [ 0  5 33  2  1  0  0]
 [ 2  1  3 52  2  0  0]
 [ 1  0  1  3 54  2  0]
 [ 2  0  0  1  1 56  0]
 [ 0  0  0  0  0  0 56]]
--------------------------------------------------------------------------------
--------------------------------------------------------------------
================================================================================
====================================================================

Results for configuration: ('MinMaxScaler', 'SimpleImputer_median',
'CountEncoder', 'VarianceThreshold', 'GradientBoostingClassifier')
Run 1:
Accuracy: 0.932
```

```
Best Parameters: {'learning_rate': 0.5, 'max_depth': 5, 'n_estimators': 50}
Precision: [0.961 0.875 0.889 0.979 0.942 0.945 0.951]
Recall: [0.961 0.982 0.842 0.904 0.891 0.963 0.983]
F1: [0.961 0.926 0.865 0.94  0.916 0.954 0.967]
Duration: 237.08165621757507 seconds
Confusion Matrix:
[[49  1  0  0  0  0  1]
 [ 0 56  1  0  0  0  0]
 [ 1  6 48  1  0  0  1]
 [ 0  1  2 47  2  0  0]
 [ 1  0  2  0 49  2  1]
 [ 0  0  1  0  1 52  0]
 [ 0  0  0  0  0  1 58]]
--------------------------------------------------------------------------------
--------------------------------------------------------------------
Run 2:
Accuracy: 0.938
Best Parameters: {'learning_rate': 0.5, 'max_depth': 5, 'n_estimators': 100}
Precision: [0.944 0.855 0.93  0.938 0.925 1.    0.973]
Recall: [0.879 0.94  0.833 0.938 0.969 1.    0.986]
F1: [0.911 0.895 0.879 0.938 0.947 1.    0.979]
Duration: 232.0034921169281 seconds
Confusion Matrix:
[[51  2  0  1  2  0  2]
 [ 2 47  1  0  0  0  0]
 [ 0  6 40  1  1  0  0]
 [ 0  0  1 45  2  0  0]
 [ 0  0  1  1 62  0  0]
 [ 0  0  0  0  0 45  0]
 [ 1  0  0  0  0  0 71]]
--------------------------------------------------------------------------------
--------------------------------------------------------------------
Run 3:
Accuracy: 0.919
Best Parameters: {'learning_rate': 0.5, 'max_depth': 5, 'n_estimators': 100}
Precision: [0.936 0.852 0.849 0.956 0.908 0.957 0.984]
Recall: [0.978 0.852 0.818 0.878 0.986 0.978 0.94 ]
F1: [0.957 0.852 0.833 0.915 0.945 0.967 0.962]
Duration: 237.31809067726135 seconds
Confusion Matrix:
[[44  1  0  0  0  0  0]
 [ 1 46  6  1  0  0  0]
 [ 0  4 45  1  4  1  0]
 [ 1  3  1 43  0  1  0]
 [ 0  0  1  0 69  0  0]
 [ 0  0  0  0  0 44  1]
 [ 1  0  0  0  3  0 63]]
--------------------------------------------------------------------------------
```

```
------------------------------------------------------------------
================================================================================
================================================================

Results for configuration: ('MinMaxScaler', 'IterativeImputer', 'CountEncoder',
'VarianceThreshold', 'GradientBoostingClassifier')
Run 1:
Accuracy: 0.935
Best Parameters: {'learning_rate': 0.5, 'max_depth': 5, 'n_estimators': 50}
Precision: [0.909 0.891 0.914 0.959 0.918 1.    0.964]
Recall: [0.976 0.851 0.898 0.94  0.933 0.982 1.   ]
F1: [0.941 0.87  0.906 0.949 0.926 0.991 0.981]
Duration: 234.2506618499756 seconds
Confusion Matrix:
[[40  1  0  0  0  0  0]
 [ 1 57  4  1  2  0  2]
 [ 0  5 53  0  1  0  0]
 [ 0  1  1 47  1  0  0]
 [ 3  0  0  1 56  0  0]
 [ 0  0  0  0  1 54  0]
 [ 0  0  0  0  0  0 53]]
--------------------------------------------------------------------------------
------------------------------------------------------------------
Run 2:
Accuracy: 0.938
Best Parameters: {'learning_rate': 0.5, 'max_depth': 5, 'n_estimators': 100}
Precision: [0.938 0.841 0.92  0.944 0.97  0.98  0.982]
Recall: [0.9   0.93  0.793 0.962 1.    0.98  1.   ]
F1: [0.918 0.883 0.852 0.953 0.985 0.98  0.991]
Duration: 245.5537302494049 seconds
Confusion Matrix:
[[45  3  1  1  0  0  0]
 [ 1 53  3  0  0  0  0]
 [ 1  7 46  2  1  0  1]
 [ 1  0  0 51  0  1  0]
 [ 0  0  0  0 64  0  0]
 [ 0  0  0  0  1 48  0]
 [ 0  0  0  0  0  0 54]]
--------------------------------------------------------------------------------
------------------------------------------------------------------
Run 3:
Accuracy: 0.901
Best Parameters: {'learning_rate': 0.5, 'max_depth': 5, 'n_estimators': 100}
Precision: [0.882 0.8   0.905 0.881 0.912 0.943 0.955]
Recall: [0.882 0.9   0.76  0.867 0.925 0.943 1.   ]
F1: [0.882 0.847 0.826 0.874 0.919 0.943 0.977]
Duration: 247.46071863174438 seconds
Confusion Matrix:
```

```
[[45  2  1  0  2  0  1]
 [ 2 36  1  1  0  0  0]
 [ 1  5 38  5  0  1  0]
 [ 0  2  1 52  4  1  0]
 [ 2  0  1  0 62  1  1]
 [ 1  0  0  1  0 50  1]
 [ 0  0  0  0  0  0 64]]
--------------------------------------------------------------------------------
--------------------------------------------------------------------
================================================================================
====================================================================

Results for configuration: ('MinMaxScaler', 'IterativeImputer',
'CustomOrdinalEncoder', 'VarianceThreshold', 'RandomForestClassifier')
Run 1:
Accuracy: 0.912
Best Parameters: {'max_depth': 20, 'max_features': 0.5, 'n_estimators': 100}
Precision: [1.    0.784 0.857 0.909 0.952 0.918 0.95 ]
Recall: [0.927 0.851 0.857 0.862 0.909 0.978 1.   ]
F1: [0.962 0.816 0.857 0.885 0.93  0.947 0.974]
Duration: 26.78124237060547 seconds
Confusion Matrix:
[[51  4  0  0  0  0  0]
 [ 0 40  6  0  1  0  0]
 [ 0  2 48  5  0  1  0]
 [ 0  4  1 50  2  1  0]
 [ 0  0  1  0 60  2  3]
 [ 0  1  0  0  0 45  0]
 [ 0  0  0  0  0  0 57]]
--------------------------------------------------------------------------------
--------------------------------------------------------------------
Run 2:
Accuracy: 0.904
Best Parameters: {'max_depth': None, 'max_features': 0.5, 'n_estimators': 100}
Precision: [0.9   0.875 0.909 0.911 0.862 0.92  0.951]
Recall: [0.915 0.831 0.909 0.837 0.943 0.902 0.983]
F1: [0.908 0.852 0.909 0.872 0.901 0.911 0.967]
Duration: 28.169500827789307 seconds
Confusion Matrix:
[[54  4  0  0  0  0  1]
 [ 4 49  4  0  0  0  2]
 [ 0  1 50  1  2  1  0]
 [ 1  1  1 41  3  2  0]
 [ 1  0  0  2 50  0  0]
 [ 0  1  0  1  3 46  0]
 [ 0  0  0  0  0  1 58]]
--------------------------------------------------------------------------------
--------------------------------------------------------------------
```

```
Run 3:
Accuracy: 0.909
Best Parameters: {'max_depth': 20, 'max_features': 0.5, 'n_estimators': 100}
Precision: [0.94  0.759 0.958 0.914 0.935 0.94  0.921]
Recall: [0.94  0.891 0.793 0.869 0.921 0.979 0.983]
F1: [0.94  0.82  0.868 0.891 0.928 0.959 0.951]
Duration: 26.111685514450073 seconds
Confusion Matrix:
[[47  2  0  0  0  0  1]
 [ 2 41  1  0  0  0  2]
 [ 0  7 46  3  1  0  1]
 [ 0  3  1 53  3  1  0]
 [ 0  0  0  2 58  2  1]
 [ 1  0  0  0  0 47  0]
 [ 0  1  0  0  0  0 58]]
--------------------------------------------------------------------------------
--------------------------------------------------------------------
================================================================================
====================================================================
```

5.0.2 Conclusions

Acuratetea algoritmului

- Se observa clar ca GradientBoostingClassifier obtine cele mai bune si robuste rezultate (varianta cat mai mica), ceea ce indica faptul ca este cel mai performant model si ca este mai susceptibil la dataset-uri mici, randomness-ul distributiei datelor etc.
- Desi observam ca si RandomForestClassifier si ExtraTreesClassifier pot obtine in anumite cazuri mai izolate (o data la cateva rulari) rezultate foarte bune, aceastea nu au consistenta in a face predictii bune.
  - In speta se observa ca prin rularea repetata a configuratiilor care au obtinut cele mai bune 5 acurateti, doar GradientBoostingClassifier obtine rezultate similare (deviatie standard mica). Totusi si celelalte 2 modele au un comportament similar dar cu deviatii mai mari. De exemplu GradientBoostingClassifier are marja de eroare a acuratetii de aproximativ 3%, iar ceilalti algoritmi pot avea 6%.
  - Totodata observam ca SVC obtine cele mai proase rezultate (cumva si logic, din cauza naturii problemei explorate)
- Durata de executie a modelelor a reprezentat un dezavantaj pentru acest studiu de caz, deoarece mereu timpul este important. Observam ca GradientBoostingClassifier are timpi de rulare de pana la 9-12 ori mai mari fata de ceilalti algoritmi, ceea ce poate reprezenta un inconvenient mare in raport cu RandomForestClassifier si ExtraTreesClassifier, care obtin performante similare (mai slabe cu maxim 10%) intr-un timp mult mai scurt.
- In final se va utiliza configuratia care se preteaza cel mai mult pe cazul de utilizare, mereu exista un trade-off din dorinta de a prioritiza nevoile aplicatiei (timp vs acuratetea predictiei)