solution

April 28, 2024

1 Tema 1 ML - Paunoiu Darius Alexandru

[ ]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt

# Assuming df is your DataFrame after loading the CSV
statistics_df = pd.read_csv("date_tema_1_iaut_2024.csv")
pd.set_option('display.max_columns', None)

RANDOM_STATES = [42, 10, 15, 21, 13, 30, 35, 37, 45, 53]
RANDOM_STATE = RANDOM_STATES[0]
# List of categorical columns you mentioned
def prelucrate_data(df):
    df['Sedentary_hours_daily'] = df['Sedentary_hours_daily'].str.replace(',',␣
 ↪'.').astype(float)
    df['Age'] = df['Age'].str.replace(',', '.').astype(float).astype(int)
    df['Est_avg_calorie_intake'] = df['Est_avg_calorie_intake'].astype(int)
    df['Height'] = df['Height'].str.replace(',', '.').astype(float)
    df['Water_daily'] = df['Water_daily'].str.replace(',', '.').astype(float)
    df['Weight'] = df['Weight'].str.replace(',', '.').astype(float)
    df['Physical_activity_level'] = df['Physical_activity_level'].str.
 ↪replace(',', '.').astype(float)
    df['Technology_time_use'] = df['Technology_time_use'].astype(object)
    df['Main_meals_daily'] = df['Main_meals_daily'].str.replace(',', '.').
 ↪astype(float).astype(int).astype(object)
    df['Regular_fiber_diet'] = df['Regular_fiber_diet'].str.replace(',', '.').
 ↪astype(float).astype(int).astype(object)

prelucrate_data(statistics_df)
print(statistics_df.dtypes)

# Splitting the DataFrame into train and test datasets
train_df, test_df = train_test_split(statistics_df, test_size=0.2,␣
 ↪random_state=42)

# Printing the shapes of the train and test datasets
```

```python
print("Train dataset shape:", train_df.shape)
print("Test dataset shape:", test_df.shape)

statistics_df.tail()
```

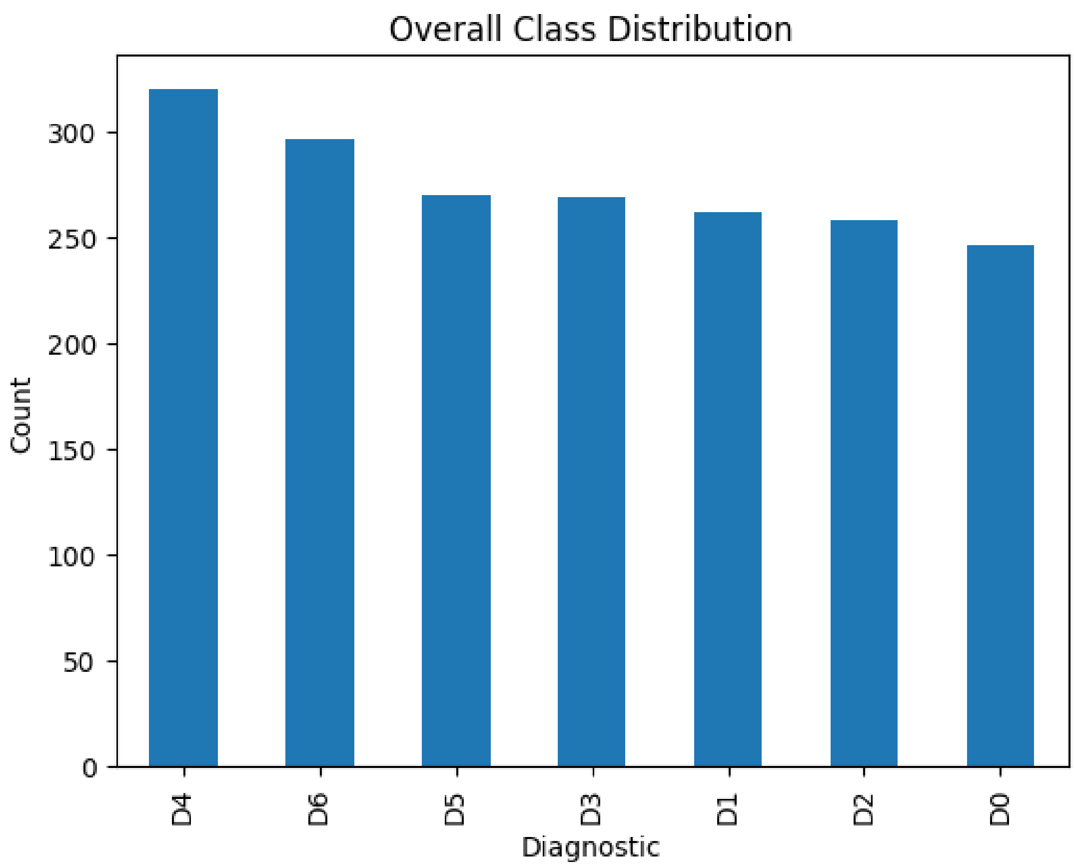
```
Transportation                     object
Regular_fiber_diet                 object
Diagnostic_in_family_history       object
High_calorie_diet                  object
Sedentary_hours_daily             float64
Age                                 int64
Alcohol                            object
Est_avg_calorie_intake              int64
Main_meals_daily                   object
Snacks                             object
Height                            float64
Smoker                             object
Water_daily                       float64
Calorie_monitoring                 object
Weight                            float64
Physical_activity_level           float64
Technology_time_use                object
Gender                             object
Diagnostic                         object
dtype: object
Train dataset shape: (1536, 19)
Test dataset shape: (385, 19)
```

[ ]:

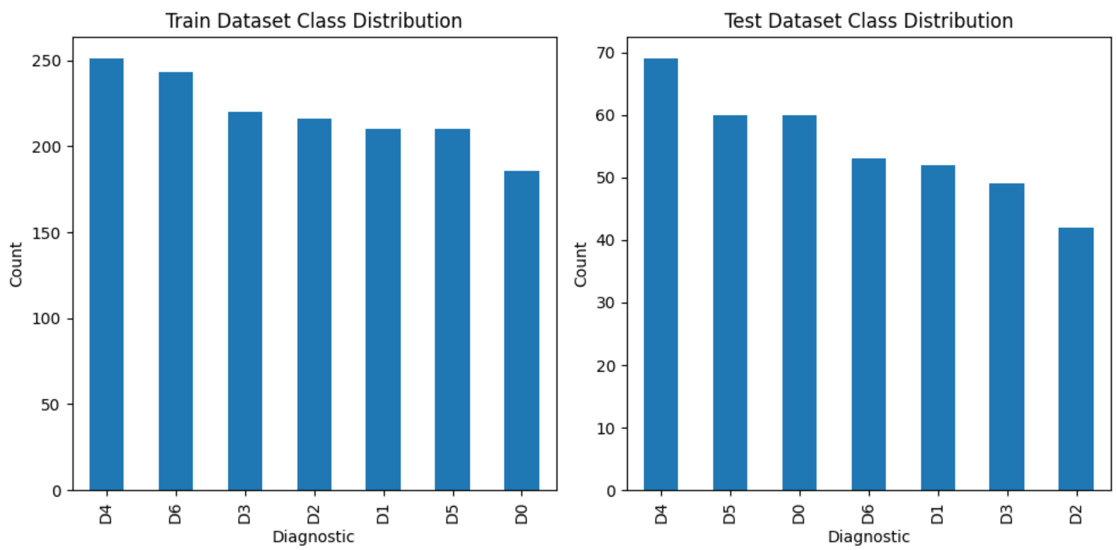
```
             Transportation Regular_fiber_diet Diagnostic_in_family_history  \
1916  Public_Transportation                  3                          yes
1917  Public_Transportation                  3                          yes
1918  Public_Transportation                  3                          yes
1919  Public_Transportation                  3                          yes
1920  Public_Transportation                  3                          yes

     High_calorie_diet  Sedentary_hours_daily  Age    Alcohol  \
1916               yes                   3.08   20  Sometimes
1917               yes                   3.00   21  Sometimes
1918               yes                   3.26   22  Sometimes
1919               yes                   3.61   24  Sometimes
1920               yes                   3.83   23  Sometimes

      Est_avg_calorie_intake Main_meals_daily     Snacks  Height Smoker  \
1916                    2744                3  Sometimes    1.71     no
1917                    2977                3  Sometimes    1.75     no
1918                    2422                3  Sometimes    1.75     no
1919                    2372                3  Sometimes    1.74     no
```

```
1920                    2336                3  Sometimes    1.74     no

      Water_daily Calorie_monitoring      Weight  Physical_activity_level  \
1916     1.728139                 no  131.408528                 1.676269
1917     2.005130                 no  133.742943                 1.341390
1918     2.054193                 no  133.689352                 1.414209
1919     2.852339                 no  133.346641                 1.139107
1920     2.863513                 no  133.472641                 1.026452

     Technology_time_use  Gender Diagnostic
1916                    1  Female         D6
1917                    1  Female         D6
1918                    1  Female         D6
1919                    1  Female         D6
1920                    1  Female         D6
```

[ ]:
```
# Class distribution overall
class_counts = statistics_df['Diagnostic'].value_counts()
class_counts.plot(kind='bar')
plt.xlabel('Diagnostic')
plt.ylabel('Count')
plt.title('Overall Class Distribution')
plt.show()
```



[ ]:
```
train_class_counts = train_df['Diagnostic'].value_counts()
test_class_counts = test_df['Diagnostic'].value_counts()

plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
train_class_counts.plot(kind='bar')
plt.xlabel('Diagnostic')
plt.ylabel('Count')
plt.title('Train Dataset Class Distribution')

plt.subplot(1, 2, 2)
test_class_counts.plot(kind='bar')
plt.xlabel('Diagnostic')
plt.ylabel('Count')
plt.title('Test Dataset Class Distribution')

plt.tight_layout()
plt.show()
```



[ ]:
```python
import pandas as pd
import numpy as np
from scipy.stats import tmean, tstd, median_abs_deviation, iqr, tmin, tmax

# Identify numerical columns
numerical_columns = statistics_df.select_dtypes(include=['int64', 'float64']).
↪columns

# Initialize a dictionary to store the results
results = {}

# Calculate the required statistics for each numerical column
for col in numerical_columns:
    results[col] = {
        'Mean': tmean(statistics_df[col]),
        'Standard Deviation': tstd(statistics_df[col]),
        'Mean Absolute Deviation': np.mean(np.abs(statistics_df[col] - np.
↪mean(statistics_df[col]))),
        'Min': tmin(statistics_df[col]),
        'Max': tmax(statistics_df[col]),
        'Difference between Min and Max': tmax(statistics_df[col]) -␣
↪tmin(statistics_df[col]),
        'Median': np.median(statistics_df[col]),  # SciPy does not have a␣
↪median function
        'Median Absolute Deviation': median_abs_deviation(statistics_df[col]),
        'Interquartile Range': iqr(statistics_df[col]),
    }
```

```
# Convert the results to a DataFrame
stats_df = pd.DataFrame(results).transpose()
stats_df
```

[ ]:

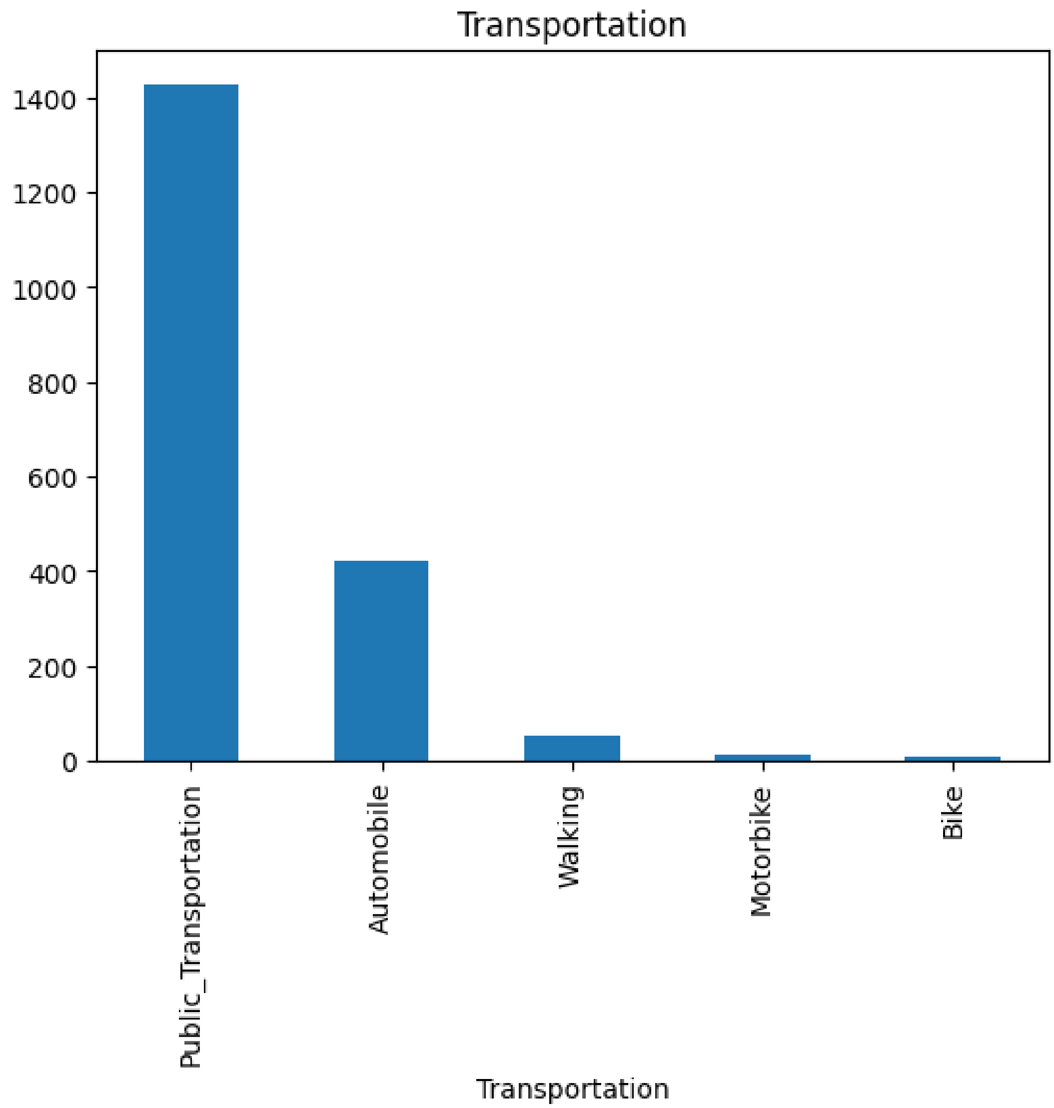
```
                                 Mean  Standard Deviation  \
Sedentary_hours_daily        3.693571           21.759835
Age                         44.454971          633.322337
Est_avg_calorie_intake    2253.687663          434.075794
Height                       3.573488           58.098160
Water_daily                  2.010367            0.611034
Weight                     205.637344         3225.653536
Physical_activity_level      1.012640            0.855526

                         Mean Absolute Deviation      Min       Max  \
Sedentary_hours_daily                   1.133885     2.21    956.58
Age                                    40.949876    15.00  19685.00
Est_avg_calorie_intake                375.362344  1500.00   3000.00
Height                                  3.738525     1.45   1915.00
Water_daily                             0.470801     1.00      3.00
Weight                                254.647671    -1.00  82628.00
Physical_activity_level                 0.702160     0.00      3.00

                         Difference between Min and Max       Median  \
Sedentary_hours_daily                            954.37     3.130000
Age                                            19670.00    22.000000
Est_avg_calorie_intake                          1500.00  2253.000000
Height                                          1913.55     1.700000
Water_daily                                        2.00     2.000000
Weight                                         82629.00    80.386078
Physical_activity_level                            3.00     1.000000

                         Median Absolute Deviation  Interquartile Range
Sedentary_hours_daily                     0.440000             0.870000
Age                                       3.000000             7.000000
Est_avg_calorie_intake                  380.000000           757.000000
Height                                    0.070000             0.140000
Water_daily                               0.444917             0.874479
Weight                                   24.386078            46.205365
Physical_activity_level                   0.815768             1.567523
```

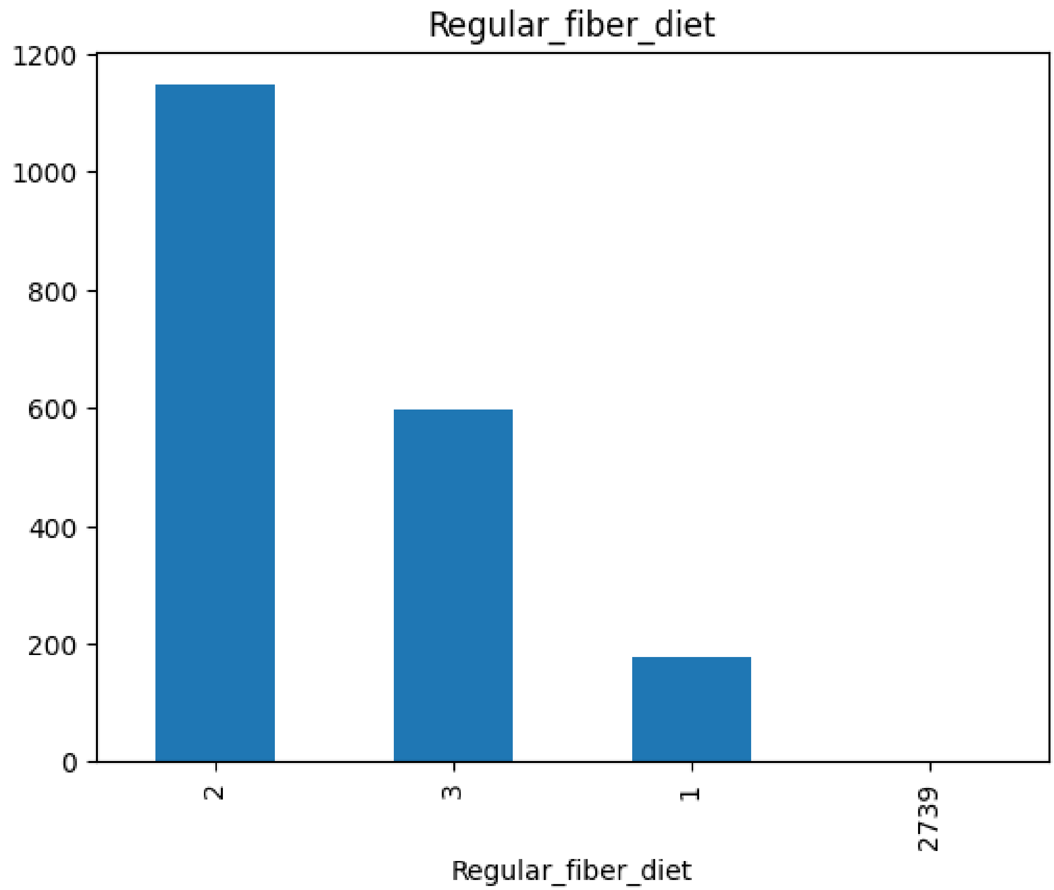
[ ]:

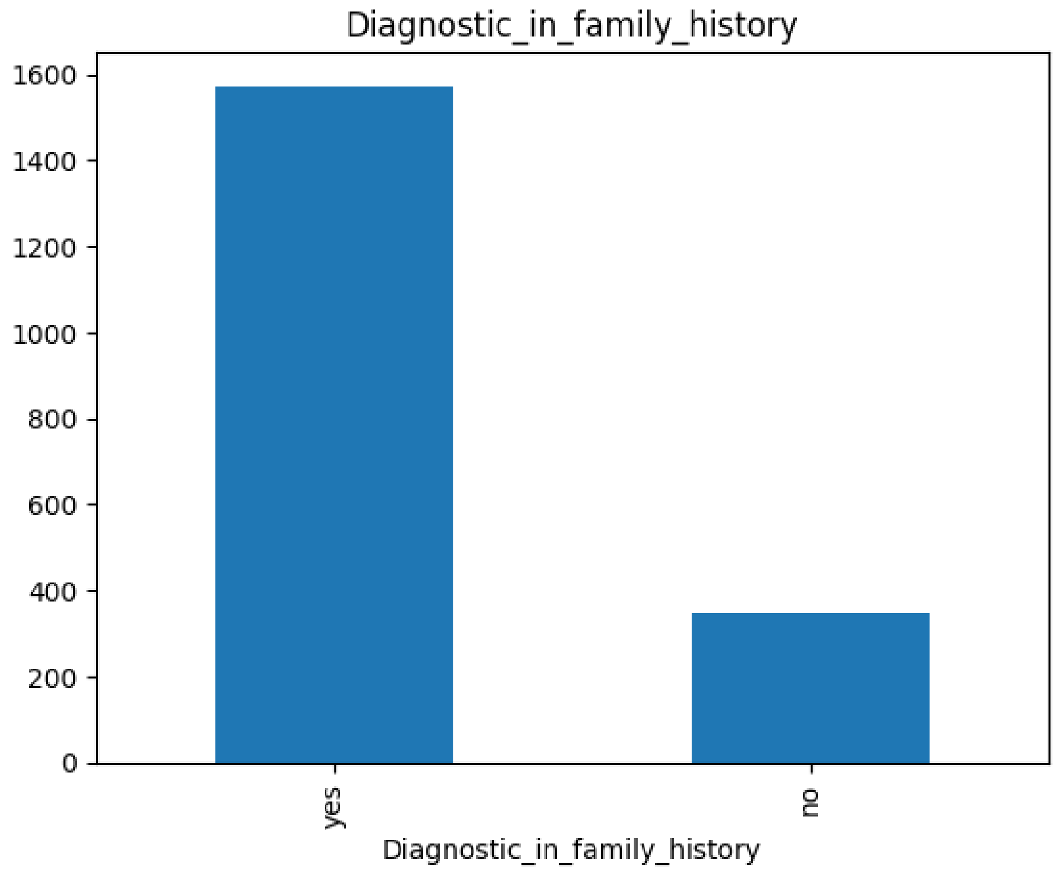
```
import pandas as pd
import matplotlib.pyplot as plt

# Identify discrete, nominal or ordinal columns
categorical_columns = statistics_df.select_dtypes(include=['object',␣
↪'category', 'int8']).columns
print(categorical_columns)
```

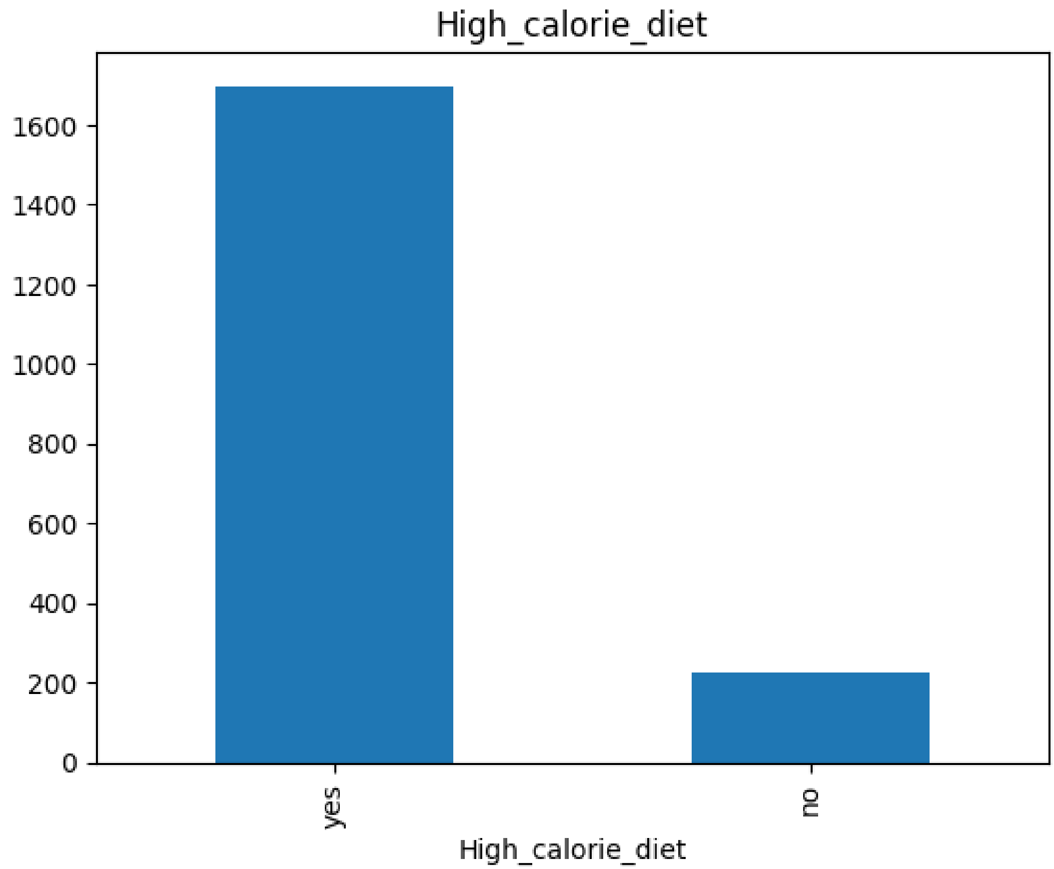
```python
# Initialize a dictionary to store the results
results = {}

# Calculate the count of unique values for each column
for col in categorical_columns:
    results[col] = statistics_df[col].nunique()

# Convert the results to a DataFrame
unique_counts_df = pd.DataFrame.from_dict(results, orient='index',␣
↪columns=['Unique Count'])

# Display the DataFrame
print(unique_counts_df)

# Plot a histogram for each column
for col in categorical_columns:
    statistics_df[col].value_counts().plot(kind='bar', title=col)
    plt.show()
```

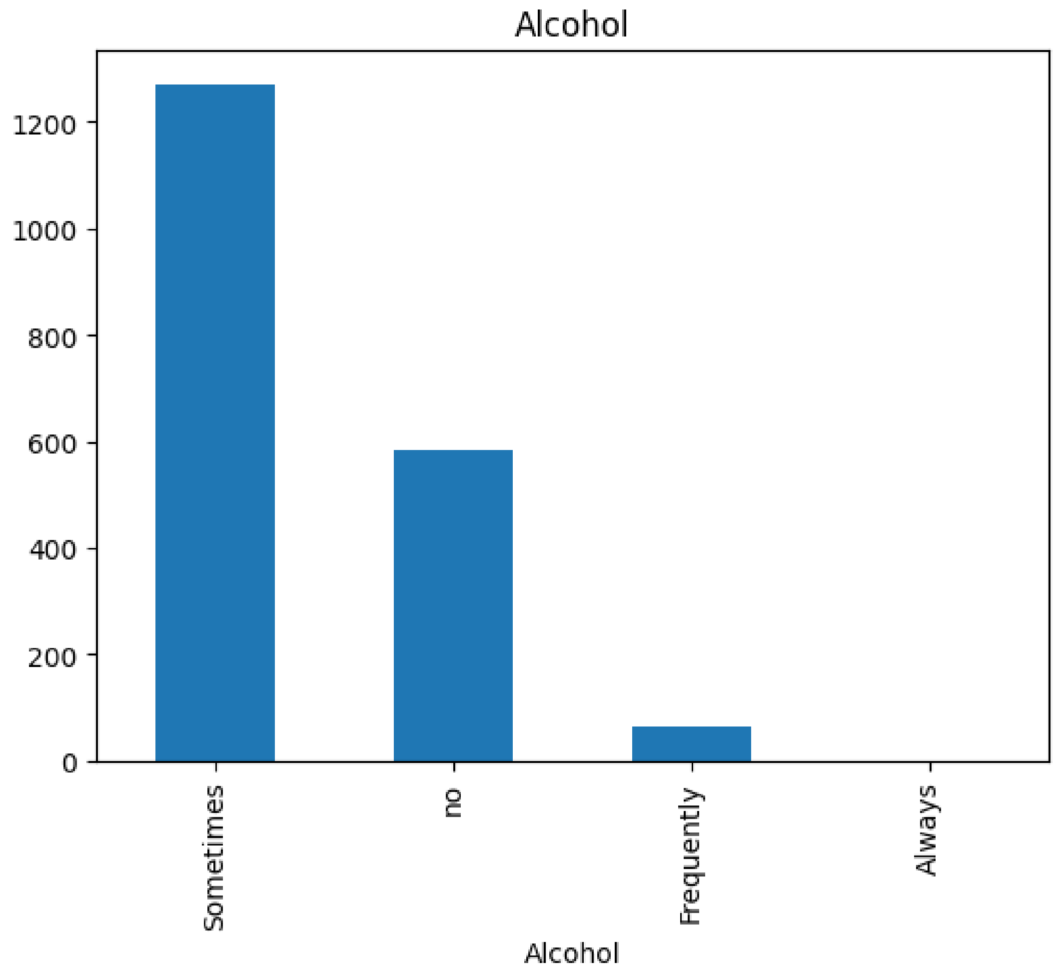
```
Index(['Transportation', 'Regular_fiber_diet', 'Diagnostic_in_family_history',
       'High_calorie_diet', 'Alcohol', 'Main_meals_daily', 'Snacks', 'Smoker',
       'Calorie_monitoring', 'Technology_time_use', 'Gender', 'Diagnostic'],
      dtype='object')
                              Unique Count
Transportation                           5
Regular_fiber_diet                       4
Diagnostic_in_family_history             2
High_calorie_diet                        2
Alcohol                                  4
Main_meals_daily                         4
Snacks                                   4
Smoker                                   2
Calorie_monitoring                       2
Technology_time_use                      4
Gender                                   2
Diagnostic                               7
```

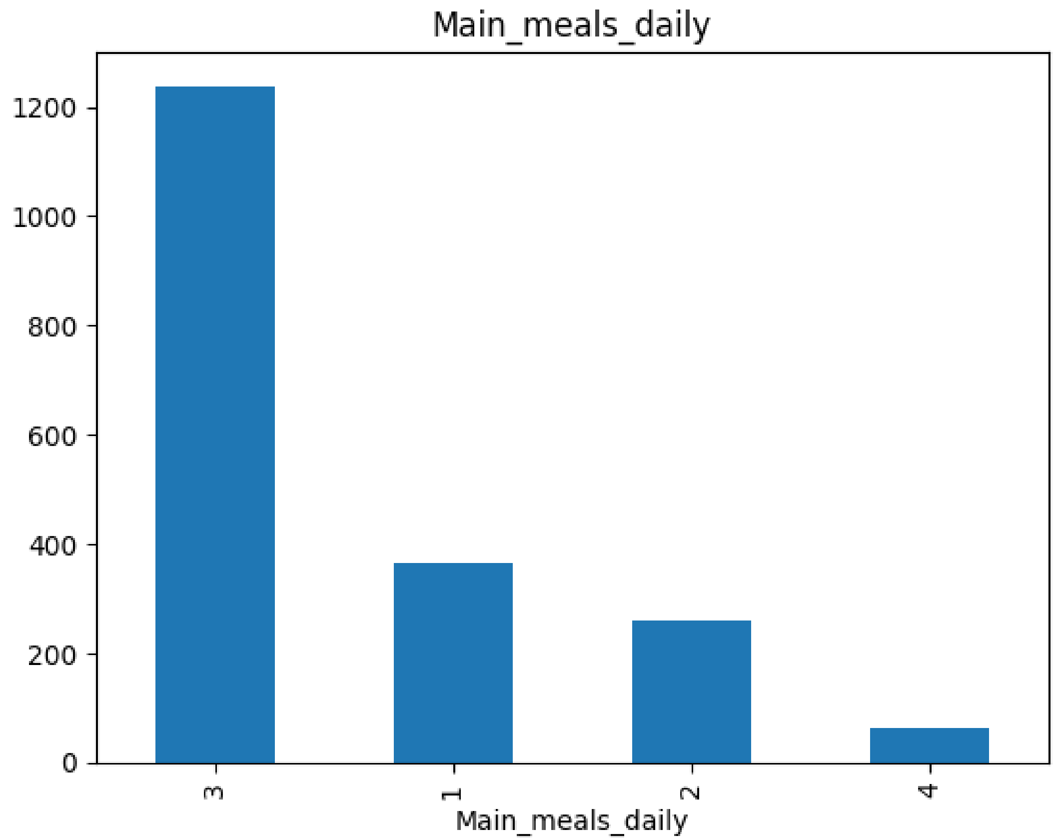

Transportation
1400
1200
1000
800
600
400
200
0
Public_Transportation
Automobile
Walking
Motorbike
Bike
Transportation

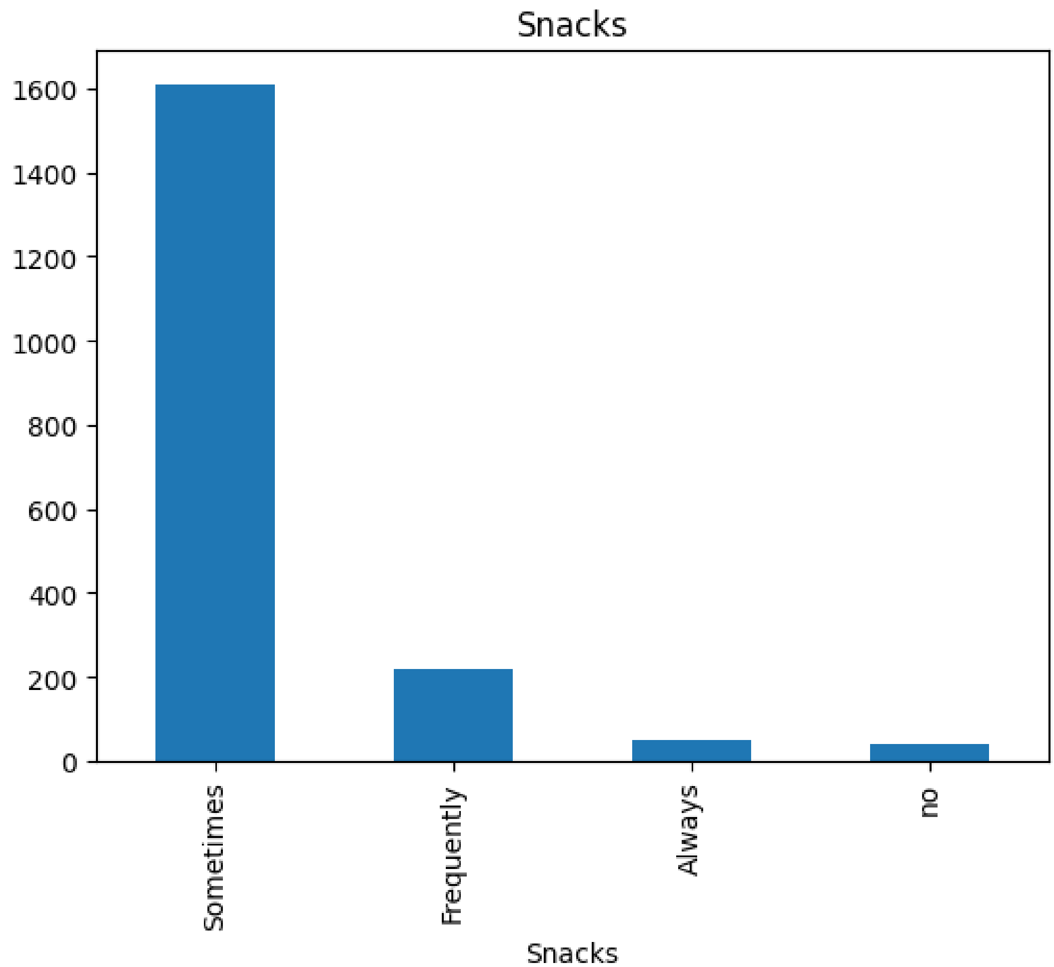

Regular_fiber_diet
1200
1000
800
600
400
200
0
2
3
1
2739
Regular_fiber_diet

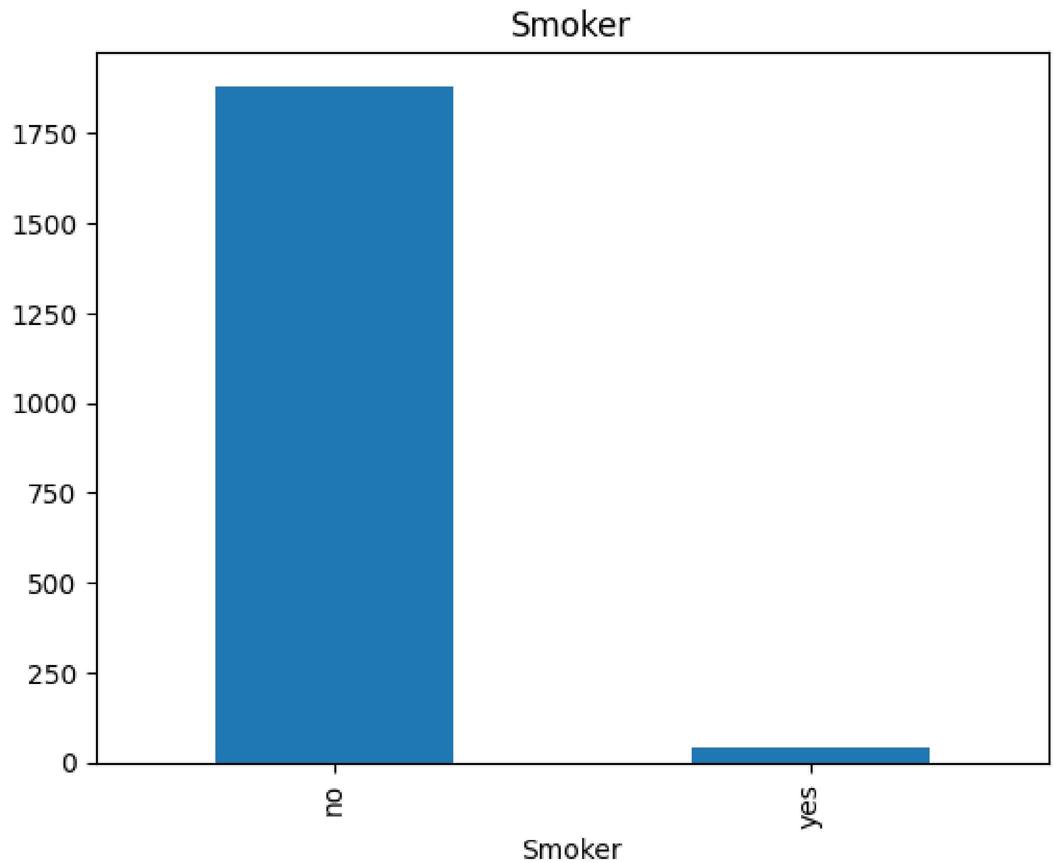

Diagnostic_in_family_history
1600
1400
1200
1000
800
600
400
200
0
yes
no
Diagnostic_in_family_history

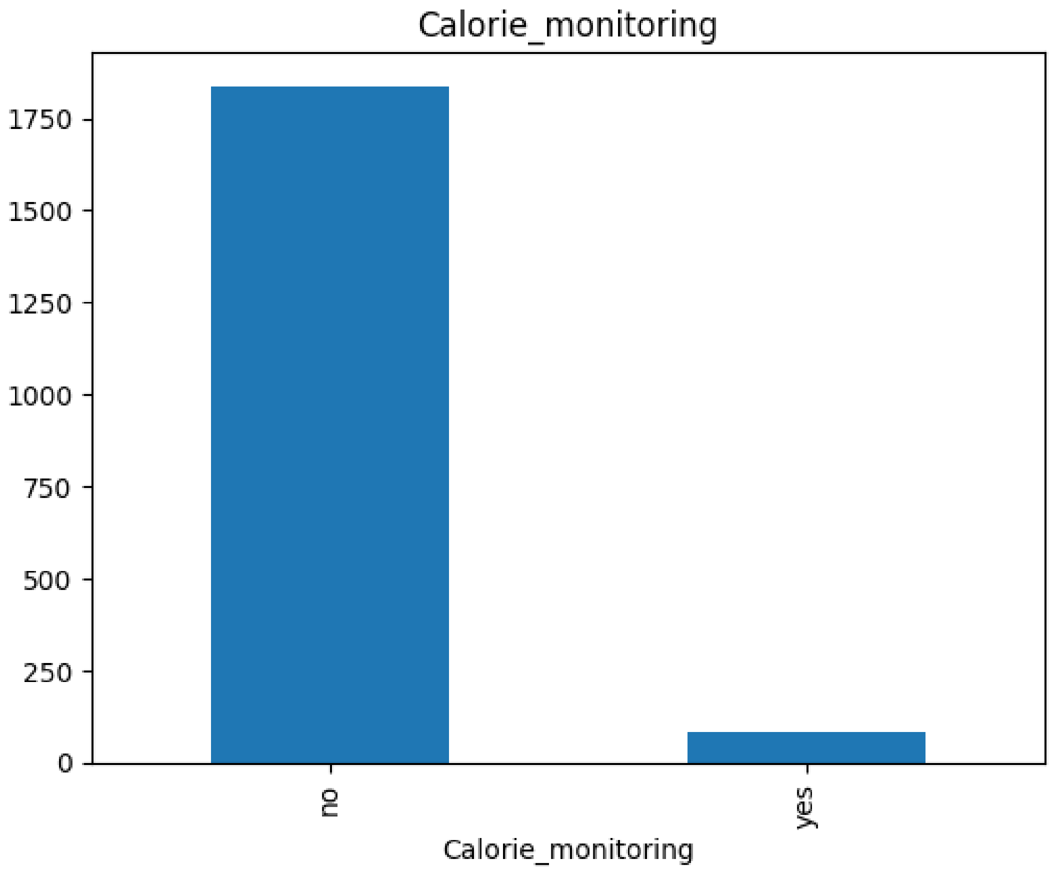

High_calorie_diet
1600
1400
1200
1000
800
600
400
200
0
yes
no
High_calorie_diet

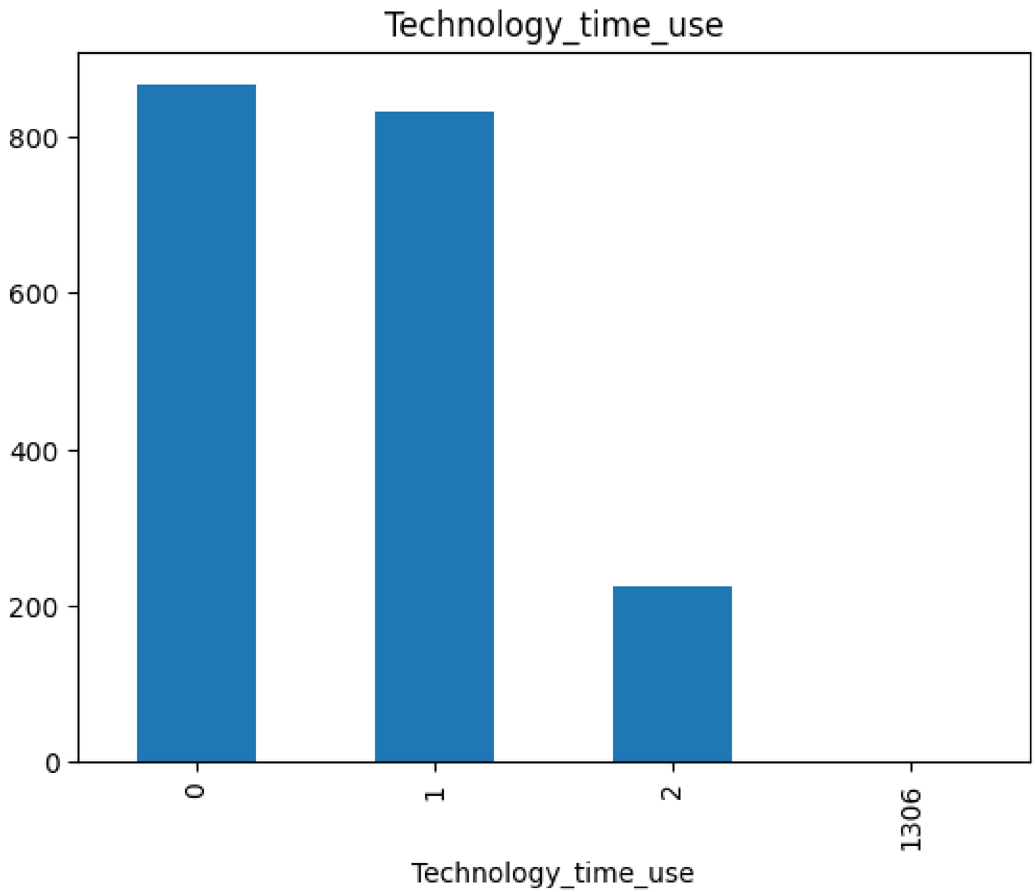

Alcohol
1200
1000
800
600
400
200
0
Sometimes
no
Frequently
Always
Alcohol

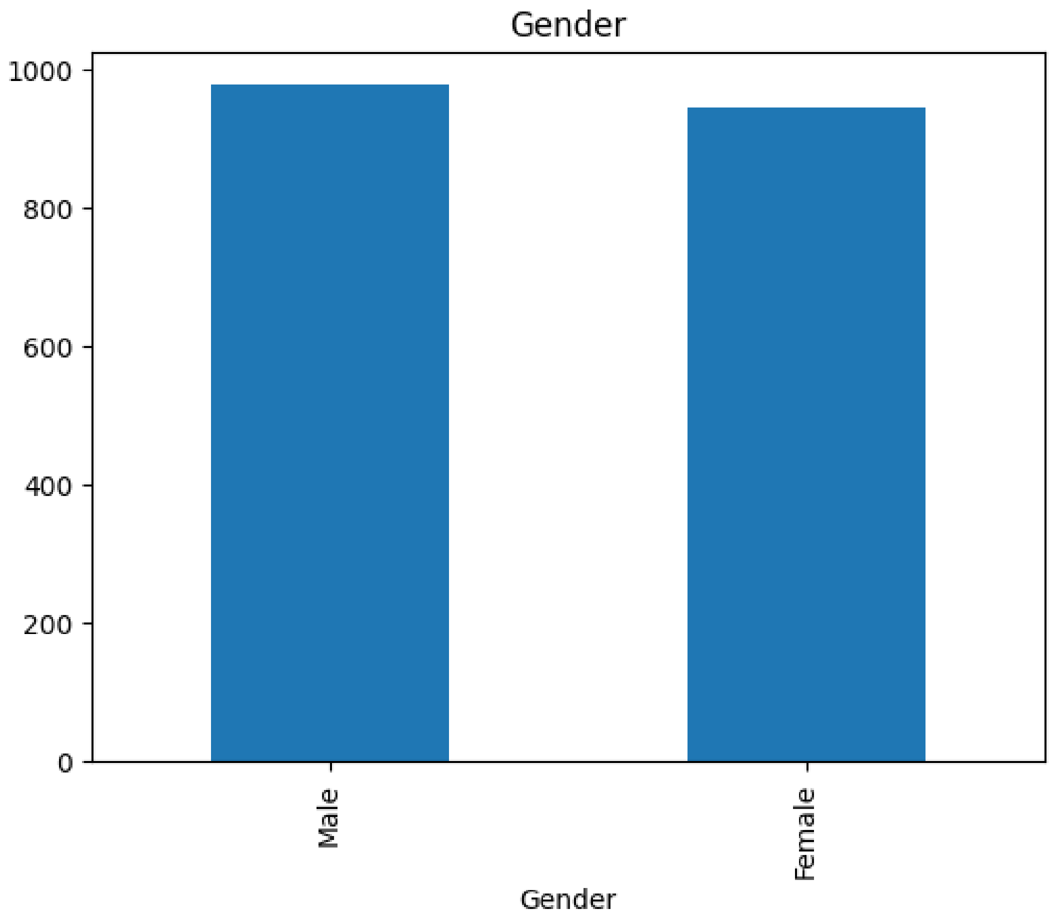

Main_meals_daily
1200
1000
800
600
400
200
0
3
1
2
4
Main_meals_daily

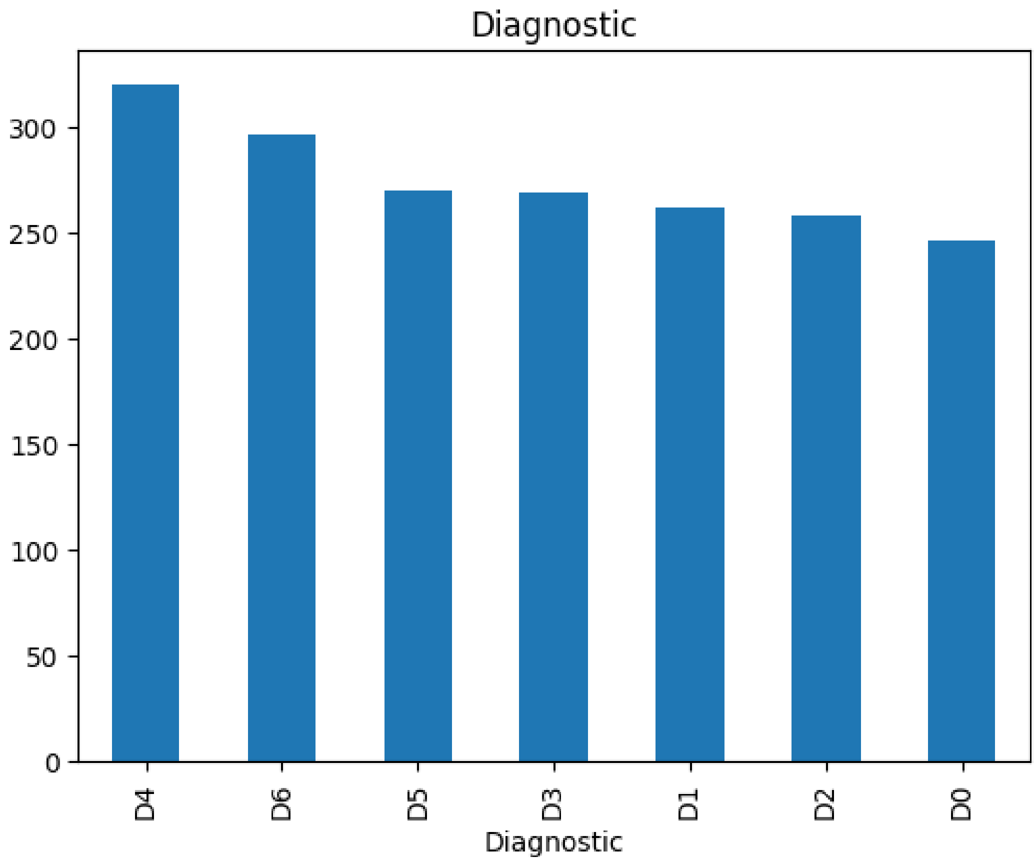

Snacks
1600
1400
1200
1000
800
600
400
200
0
Sometimes
Frequently
Always
no
Snacks


Smoker
1750
1500
1250
1000
750
500
250
0
no
yes
Smoker


Calorie_monitoring
1750
1500
1250
1000
750
500
250
0
no
yes
Calorie_monitoring


Technology_time_use
800
600
400
200
0
0
1
2
1306
Technology_time_use


Gender
1000
800
600
400
200
0
Male
Female
Gender



[ ]:
```python
import pandas as pd
from IPython.display import display

for column in statistics_df.columns:
    if statistics_df[column].dtype == 'object':
        statistics_df[column] = statistics_df[column].astype('category').cat.
 ↪codes

cov_attributes = statistics_df.cov()
display(cov_attributes)
```

	Transportation	Regular_fiber_diet \
Transportation	1.613173	0.088864
Regular_fiber_diet	0.088864	0.356925
Diagnostic_in_family_history	-0.048862	0.003358
High_calorie_diet	-0.029535	-0.012392
Sedentary_hours_daily	0.422834	-0.137983
Age	8.382093	-4.589061
Alcohol	-0.015630	-0.027903
Est_avg_calorie_intake	8.080411	-7.286559

Main_meals_daily	-0.001890	0.069033
Snacks	-0.026046	-0.027504
Height	1.200332	-0.415074
Smoker	-0.001645	0.002715
Water_daily	0.040653	0.037126
Calorie_monitoring	0.008656	0.008733
Weight	82.964571	-25.592608
Physical_activity_level	0.013698	-0.003526
Technology_time_use	0.146976	-0.035743
Gender	-0.087938	-0.096159
Diagnostic	0.031402	0.245788

	Diagnostic_in_family_history	High_calorie_diet \
Transportation	-0.048862	-0.029535
Regular_fiber_diet	0.003358	-0.012392
Diagnostic_in_family_history	0.148416	0.024699
High_calorie_diet	0.024699	0.103063
Sedentary_hours_daily	0.094913	0.071932
Age	4.174651	2.496830
Alcohol	0.006592	-0.014435
Est_avg_calorie_intake	-6.864424	2.786477
Main_meals_daily	0.015612	-0.005995
Snacks	0.031439	0.021901
Height	0.347992	0.223678
Smoker	0.000649	-0.002779
Water_daily	0.036532	0.002342
Calorie_monitoring	-0.012481	-0.012147
Weight	27.775319	16.861625
Physical_activity_level	-0.017956	-0.029297
Technology_time_use	0.006897	0.013275
Gender	0.019265	0.010898
Diagnostic	0.385575	0.154993

	Sedentary_hours_daily	Age	Alcohol \
Transportation	0.422834	8.382093	-0.015630
Regular_fiber_diet	-0.137983	-4.589061	-0.027903
Diagnostic_in_family_history	0.094913	4.174651	0.006592
High_calorie_diet	0.071932	2.496830	-0.014435
Sedentary_hours_daily	473.490415	-12.924928	0.376236
Age	-12.924928	401097.182477	-5.632406
Alcohol	0.376236	-5.632406	0.268677
Est_avg_calorie_intake	273.718418	-84.411467	-0.293501
Main_meals_daily	-0.735790	9.385148	-0.053053
Snacks	0.062952	3.126387	-0.011072
Height	-0.587676	-34.975499	-0.510293
Smoker	-0.009673	-0.338645	-0.006128
Water_daily	-0.071017	10.451026	-0.031022
Calorie_monitoring	-0.023988	-1.008210	-0.000697

```
Weight                          19.163702   -2773.142041  49.098309
Physical_activity_level         -0.476095      19.995552   0.036450
Technology_time_use              1.494694      -4.699622   0.013008
Gender                          -0.252078      10.209111   0.002635
Diagnostic                      -0.637862     -19.515486  -0.154355

                             Est_avg_calorie_intake  Main_meals_daily  \
Transportation                             8.080411         -0.001890
Regular_fiber_diet                        -7.286559          0.069033
Diagnostic_in_family_history              -6.864424          0.015612
High_calorie_diet                          2.786477         -0.005995
Sedentary_hours_daily                    273.718418         -0.735790
Age                                      -84.411467          9.385148
Alcohol                                   -0.293501         -0.053053
Est_avg_calorie_intake                188421.795103         -4.490091
Main_meals_daily                          -4.490091          0.692437
Snacks                                     0.611156         -0.045057
Height                                  -930.357530          0.915565
Smoker                                    -2.613285          0.004791
Water_daily                               -4.249024          0.038445
Calorie_monitoring                        -1.236498          0.002519
Weight                                -20457.233307        -21.213213
Physical_activity_level                   -1.738810          0.086682
Technology_time_use                        3.180425          0.019951
Gender                                    -5.736378          0.011937
Diagnostic                               -32.980618          0.081474

                                Snacks       Height    Smoker  Water_daily  \
Transportation               -0.026046     1.200332 -0.001645     0.040653
Regular_fiber_diet           -0.027504    -0.415074  0.002715     0.037126
Diagnostic_in_family_history  0.031439     0.347992  0.000649     0.036532
High_calorie_diet             0.021901     0.223678 -0.002779     0.002342
Sedentary_hours_daily         0.062952    -0.587676 -0.009673    -0.071017
Age                           3.126387   -34.975499 -0.338645    10.451026
Alcohol                      -0.011072    -0.510293 -0.006128    -0.031022
Est_avg_calorie_intake        0.611156  -930.357530 -2.613285    -4.249024
Main_meals_daily             -0.045057     0.915565  0.004791     0.038445
Snacks                        0.217145     0.269326 -0.004320     0.043916
Height                        0.269326  3375.396169  0.957860    -1.181213
Smoker                       -0.004320     0.957860  0.020400    -0.002802
Water_daily                   0.043916    -1.181213 -0.002802     0.373363
Calorie_monitoring           -0.010501    -0.083130  0.001704     0.001909
Weight                       21.545168  -182.428303 -2.569740   -28.511896
Physical_activity_level      -0.010111    -0.237567  0.001530     0.089479
Technology_time_use          -0.013616    -0.370402  0.000680     0.002872
Gender                        0.021538     0.072980  0.002425     0.034726
Diagnostic                    0.301313     2.670596 -0.001431     0.165964
```

	Calorie_monitoring	Weight
Transportation	0.008656	8.296457e+01
Regular_fiber_diet	0.008733	-2.559261e+01
Diagnostic_in_family_history	-0.012481	2.777532e+01
High_calorie_diet	-0.012147	1.686162e+01
Sedentary_hours_daily	-0.023988	1.916370e+01
Age	-1.008210	-2.773142e+03
Alcohol	-0.000697	4.909831e+01
Est_avg_calorie_intake	-1.236498	-2.045723e+04
Main_meals_daily	0.002519	-2.121321e+01
Snacks	-0.010501	2.154517e+01
Height	-0.083130	-1.824283e+02
Smoker	0.001704	-2.569740e+00
Water_daily	0.001909	-2.851190e+01
Calorie_monitoring	0.041361	-6.559527e+00
Weight	-6.559527	1.040484e+07
Physical_activity_level	0.015701	7.405877e+00
Technology_time_use	-0.001766	3.997895e+01
Gender	-0.008965	-2.128977e+01
Diagnostic	-0.076485	1.139335e+02

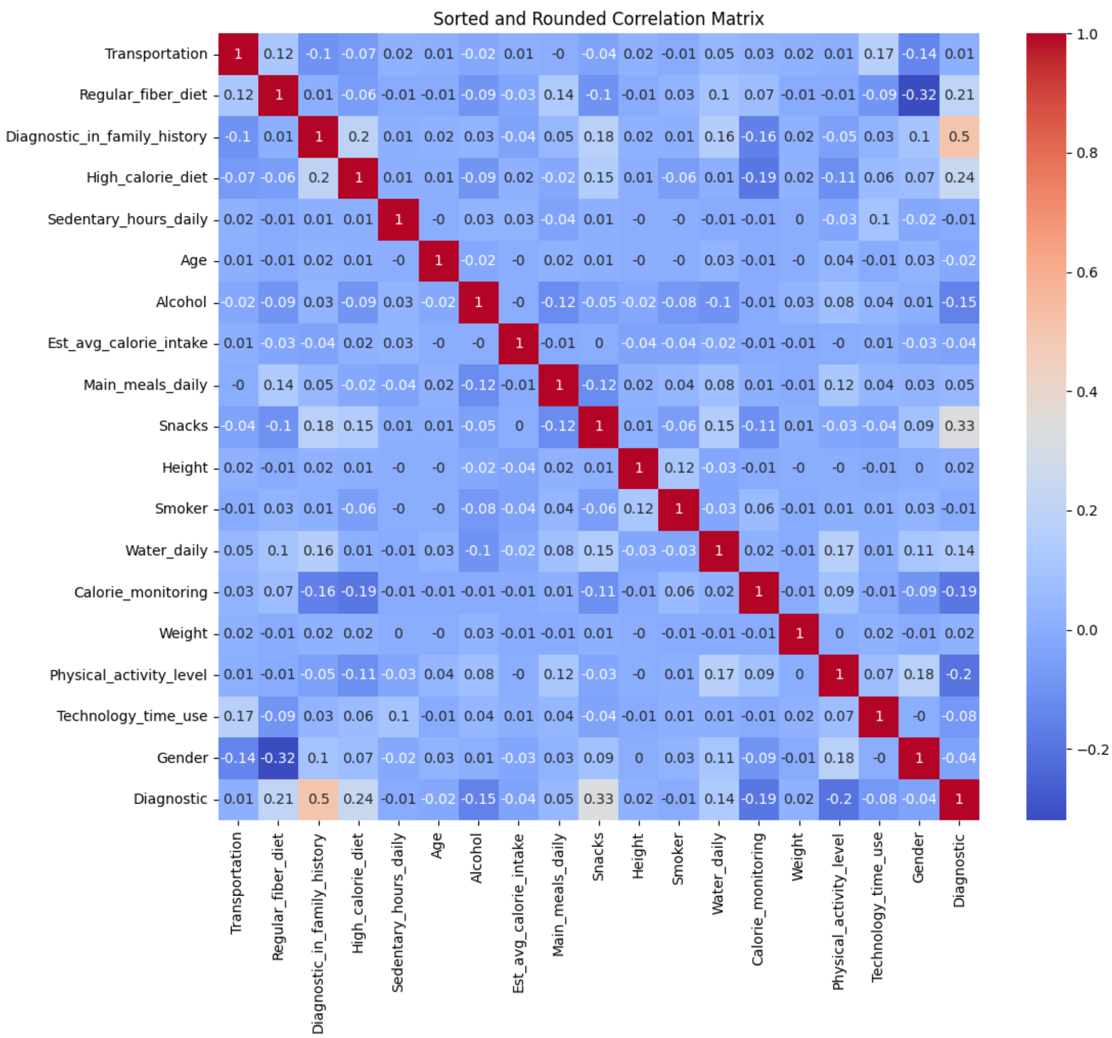
	Physical_activity_level	Technology_time_use
Transportation	0.013698	0.146976
Regular_fiber_diet	-0.003526	-0.035743
Diagnostic_in_family_history	-0.017956	0.006897
High_calorie_diet	-0.029297	0.013275
Sedentary_hours_daily	-0.476095	1.494694
Age	19.995552	-4.699622
Alcohol	0.036450	0.013008
Est_avg_calorie_intake	-1.738810	3.180425
Main_meals_daily	0.086682	0.019951
Snacks	-0.010111	-0.013616
Height	-0.237567	-0.370402
Smoker	0.001530	0.000680
Water_daily	0.089479	0.002872
Calorie_monitoring	0.015701	-0.001766
Weight	7.405877	39.978948
Physical_activity_level	0.731924	0.038979
Technology_time_use	0.038979	0.458564
Gender	0.078889	-0.001048
Diagnostic	-0.335834	-0.113624

	Gender	Diagnostic
Transportation	-0.087938	0.031402
Regular_fiber_diet	-0.096159	0.245788
Diagnostic_in_family_history	0.019265	0.385575
High_calorie_diet	0.010898	0.154993
Sedentary_hours_daily	-0.252078	-0.637862

```
Age                          10.209111  -19.515486
Alcohol                       0.002635   -0.154355
Est_avg_calorie_intake       -5.736378  -32.980618
Main_meals_daily              0.011937    0.081474
Snacks                        0.021538    0.301313
Height                        0.072980    2.670596
Smoker                        0.002425   -0.001431
Water_daily                   0.034726    0.165964
Calorie_monitoring           -0.008965   -0.076485
Weight                      -21.289766  113.933453
Physical_activity_level       0.078889   -0.335834
Technology_time_use          -0.001048   -0.113624
Gender                        0.250056   -0.035916
Diagnostic                   -0.035916    3.935906
```

[ ]:

```python
import seaborn as sns
correlation_matrix = statistics_df.corr()

# Round the values to a maximum of 3 decimals
rounded_corr_matrix = correlation_matrix.round(2)

# Create a heatmap of the sorted and rounded correlation matrix
plt.figure(figsize=(12, 10))
sns.heatmap(rounded_corr_matrix, annot=True, cmap='coolwarm')

# Set the title of the heatmap
plt.title('Sorted and Rounded Correlation Matrix')

# Display the heatmap
plt.show()
```




```
[ ]: import ast
from sklearn.base import ClassifierMixin
from sklearn.discriminant_analysis import StandardScaler
from sklearn.impute import IterativeImputer
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import (
    accuracy_score,
    classification_report,
    f1_score,
    make_scorer,
    precision_score,
    recall_score,
)
from sklearn.preprocessing import LabelEncoder
```

```python
from sklearn.feature_selection import (
    SelectPercentile,
    VarianceThreshold,
    chi2,
    f_classif,
)
from sklearn.model_selection import GridSearchCV

from matplotlib.backends.backend_pdf import PdfPages

def prepare_dataset():
    df = pd.read_csv("date_tema_1_iaut_2024.csv")
    prelucrate_data(df)
    # Replace -1 with NaN in the 'Weight' column
    df["Weight"] = df["Weight"].replace(-1, np.nan)

    # Initialize the IterativeImputer
    imputer = IterativeImputer()

    # Perform the imputation on the 'Weight' column
    df["Weight"] = imputer.fit_transform(df[["Weight"]])

    # Convert categorical columns to numerical
    le = LabelEncoder()
    for col in df.columns:
        df[col] = le.fit_transform(df[col])

    X = df.drop("Diagnostic", axis=1)
    y = df["Diagnostic"]

    # Create a VarianceThreshold object
    selector = VarianceThreshold(threshold=0.1)

    # Fit and transform the selector to the data
    features_before = X.columns
    X = pd.DataFrame(
        selector.fit_transform(X), columns=X.columns[selector.get_support()]
    )
    print(f"Features removed: {set(features_before) - set(X.columns)}")

    # Create a SelectPercentile object
    selector = SelectPercentile(f_classif, percentile=70)

    # Fit and transform the selector to the data
    features_before = X.columns
    X = pd.DataFrame(
        selector.fit_transform(X, y), columns=X.columns[selector.get_support()]
```

```python
    )
    print(f"Features removed: {set(features_before) - set(X.columns)}")

    # Standardize the features
    scaler = StandardScaler()
    X = scaler.fit_transform(X, y)
    return X, y


def find_best_params(classifier, param_grid, X, y, random_state=42):

    # Create train test
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.2, random_state=random_state
    )

    # Convert the custom scorer into a scorer that can be used with GridSearchCV

    scorers = {
        "accuracy": make_scorer(accuracy_score),
        "precision": make_scorer(precision_score, average="weighted"),
        "recall": make_scorer(recall_score, average="weighted"),
        "f1": make_scorer(f1_score, average="weighted"),
    }

    for scr in [accuracy_score, f1_score, precision_score, recall_score]:
        for class_label in np.unique(y):
            scorers[f"{scr.__name__}_D{class_label}"] = make_scorer(
                lambda y_true, y_pred, class_label: scr(
                    y_true == class_label, y_pred == class_label
                ),
                greater_is_better=True,
                class_label=class_label,
            )

    # Initialize a GridSearchCV
    grid_search = GridSearchCV(
        estimator=classifier,
        param_grid=param_grid,
        cv=5,
        scoring=scorers,
        refit="f1",
        n_jobs=4,
    )
    # Fit the GridSearchCV to the training data
    grid_search.fit(X_train, y_train)
```

```python
    # Print the best parameters
    print("Best parameters found: ", grid_search.best_params_)

    return grid_search, grid_search.best_params_


def evaluate_my_model(
    model: ClassifierMixin, grid_search: GridSearchCV, X, y, random_state=42
):
    # Create a DataFrame from cv_results_
    df = pd.DataFrame(grid_search.cv_results_)

    columns = [
        "params",
        "mean_test_accuracy",
        "std_test_accuracy",
        "mean_test_precision",
        "std_test_precision",
        "mean_test_recall",
        "std_test_recall",
        "mean_test_f1",
        "std_test_f1",
    ]
    for scr in [accuracy_score, f1_score, precision_score, recall_score]:
        for class_label in np.unique(y):
            columns.append(f"mean_test_{scr.__name__}_D{class_label}")
            columns.append(f"std_test_{scr.__name__}_D{class_label}")
    # Select the columns of interest
    df = df[columns]

    # Rename the columns
    df["params"] = df["params"].apply(lambda x: x.values())
    rename_params_to = ",".join([x for x in grid_search.best_params_])
    df = df.rename(columns={"params": rename_params_to})


    # Highlight the row with the best parameters
    def highlight_max(s):
        is_max = s == s.max()
        return ['font-weight: bold' if v else '' for v in is_max]

    # df.style.apply(highlight_max)
    df.style.highlight_max(color = 'pink', axis = 0)
    # df.style.apply(
    #     lambda x: [
    #         "font-weight: bold" if True else "" for _ in x
    #     ],
```

```
    #     axis=1,
    # )
    # display(df)
    dfs = []
    for class_label in np.unique(y):
        cols = [rename_params_to]
        for scr in [accuracy_score, f1_score, precision_score, recall_score]:
            cols.append(f"mean_test_{scr.__name__}_D{class_label}")
            cols.append(f"std_test_{scr.__name__}_D{class_label}")
        dfs.append(df[cols])
        renamed_cols = [rename_params_to]
        for scr in [accuracy_score, f1_score, precision_score, recall_score]:
            renamed_cols.append(f"{scr.__name__}_D{class_label}")
            renamed_cols.append(f"{scr.__name__}_D{class_label}_std")
        dfs[-1].columns = renamed_cols
        # display(dfs[-1])
        # fig, ax =plt.subplots(figsize=(12,4))
        # ax.axis('tight')
        # ax.axis('off')
        # the_table = ax.table(cellText=dfs[-1].values,colLabels=dfs[-1].
↪columns,loc='center')

        # #https://stackoverflow.com/questions/4042192/
↪reduce-left-and-right-margins-in-matplotlib-plot
        # pp = PdfPages(f"foo{class_label}.pdf")
        # pp.savefig(fig, bbox_inches='tight')
        # pp.close()
        print(dfs[-1].to_latex(
            index=False,  # To not include the DataFrame index as a column in␣
↪the table
            # caption="Comparison of ML Model Performance Metrics",  # The␣
↪caption to appear above the table in the LaTeX document
            # label="tab:model_comparison",  # A label used for referencing the␣
↪table within the LaTeX document
            # position="htbp",  # The preferred positions where the table␣
↪should be placed in the document ('here', 'top', 'bottom', 'page')
            # column_format="|l|l|l|l|",  # The format of the columns:␣
↪left-aligned with vertical lines between them
            # escape=False,  # Disable escaping LaTeX special characters in the␣
↪DataFrame
            # float_format="{:0.4f}".format  # Formats floats to two decimal␣
↪places
            ))


    X_train, X_test, y_train, y_test = train_test_split(
```

```
        X, y, test_size=0.2, random_state=random_state
    )
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred, average="weighted")
    precision = precision_score(y_test, y_pred, average="weighted")
    recall = recall_score(y_test, y_pred, average="weighted")
    print(
        classification_report(
            y_test, y_pred, target_names=["D0", "D1", "D2", "D3", "D4", "D5",␣
↪"D6"]
        )
    )

    return accuracy, f1, precision, recall
```

[ ]:
```
# Initialize a RandomForestClassifier
clf = RandomForestClassifier(random_state=42)


# Define the parameter grid
param_grid = {
    'n_estimators': [100, 150, 200],
    'max_depth': [15, 20, 25],
    'max_samples': [0.5, 0.7, 1.0],
}
X, y = prepare_dataset()
grid_search, best_params = find_best_params(clf, param_grid, X, y, RANDOM_STATE)
```

```
Features removed: {'Calorie_monitoring', 'Smoker'}
Features removed: {'Est_avg_calorie_intake', 'Physical_activity_level',
'Technology_time_use', 'Sedentary_hours_daily', 'Water_daily'}
Best parameters found:  {'max_depth': 20, 'max_samples': 1.0, 'n_estimators':
150}
```

[ ]:
```
best_clf = RandomForestClassifier(**best_params, random_state=RANDOM_STATE)
accuracy, f1, precision, recall = evaluate_my_model(best_clf, grid_search, X,␣
↪y, RANDOM_STATE)
print(f"Accuracy: {accuracy}")
print(f"F1 Score: {f1}")
print(f"Precision: {precision}")
print(f"Recall: {recall}")
print(grid_search.best_index_)
```

```
\begin{tabular}{lrrrrrrr}
\toprule
```

```
max_depth,max_samples,n_estimators & accuracy_score_D0 & accuracy_score_D0_std &
f1_score_D0 & f1_score_D0_std & precision_score_D0 & precision_score_D0_std &
recall_score_D0 & recall_score_D0_std \\
\midrule
dict_values([15, 0.5, 100]) & 0.935846 & 0.039221 & 0.935846 & 0.039221 &
0.935846 & 0.039221 & 0.935846 & 0.039221 \\
dict_values([15, 0.5, 150]) & 0.930441 & 0.035409 & 0.930441 & 0.035409 &
0.930441 & 0.035409 & 0.930441 & 0.035409 \\
dict_values([15, 0.5, 200]) & 0.941110 & 0.030649 & 0.941110 & 0.030649 &
0.941110 & 0.030649 & 0.941110 & 0.030649 \\
dict_values([15, 0.7, 100]) & 0.935846 & 0.039221 & 0.935846 & 0.039221 &
0.935846 & 0.039221 & 0.935846 & 0.039221 \\
dict_values([15, 0.7, 150]) & 0.935846 & 0.039221 & 0.935846 & 0.039221 &
0.935846 & 0.039221 & 0.935846 & 0.039221 \\
dict_values([15, 0.7, 200]) & 0.941252 & 0.042004 & 0.941252 & 0.042004 &
0.941252 & 0.042004 & 0.941252 & 0.042004 \\
dict_values([15, 1.0, 100]) & 0.935846 & 0.039221 & 0.935846 & 0.039221 &
0.935846 & 0.039221 & 0.935846 & 0.039221 \\
dict_values([15, 1.0, 150]) & 0.941252 & 0.042004 & 0.941252 & 0.042004 &
0.941252 & 0.042004 & 0.941252 & 0.042004 \\
dict_values([15, 1.0, 200]) & 0.935846 & 0.039221 & 0.935846 & 0.039221 &
0.935846 & 0.039221 & 0.935846 & 0.039221 \\
dict_values([20, 0.5, 100]) & 0.935846 & 0.039221 & 0.935846 & 0.039221 &
0.935846 & 0.039221 & 0.935846 & 0.039221 \\
dict_values([20, 0.5, 150]) & 0.930441 & 0.035409 & 0.930441 & 0.035409 &
0.930441 & 0.035409 & 0.930441 & 0.035409 \\
dict_values([20, 0.5, 200]) & 0.941110 & 0.030649 & 0.941110 & 0.030649 &
0.941110 & 0.030649 & 0.941110 & 0.030649 \\
dict_values([20, 0.7, 100]) & 0.935846 & 0.039221 & 0.935846 & 0.039221 &
0.935846 & 0.039221 & 0.935846 & 0.039221 \\
dict_values([20, 0.7, 150]) & 0.935846 & 0.039221 & 0.935846 & 0.039221 &
0.935846 & 0.039221 & 0.935846 & 0.039221 \\
dict_values([20, 0.7, 200]) & 0.941252 & 0.042004 & 0.941252 & 0.042004 &
0.941252 & 0.042004 & 0.941252 & 0.042004 \\
dict_values([20, 1.0, 100]) & 0.935846 & 0.039221 & 0.935846 & 0.039221 &
0.935846 & 0.039221 & 0.935846 & 0.039221 \\
dict_values([20, 1.0, 150]) & 0.941252 & 0.042004 & 0.941252 & 0.042004 &
0.941252 & 0.042004 & 0.941252 & 0.042004 \\
dict_values([20, 1.0, 200]) & 0.941252 & 0.042004 & 0.941252 & 0.042004 &
0.941252 & 0.042004 & 0.941252 & 0.042004 \\
dict_values([25, 0.5, 100]) & 0.935846 & 0.039221 & 0.935846 & 0.039221 &
0.935846 & 0.039221 & 0.935846 & 0.039221 \\
dict_values([25, 0.5, 150]) & 0.930441 & 0.035409 & 0.930441 & 0.035409 &
0.930441 & 0.035409 & 0.930441 & 0.035409 \\
dict_values([25, 0.5, 200]) & 0.941110 & 0.030649 & 0.941110 & 0.030649 &
0.941110 & 0.030649 & 0.941110 & 0.030649 \\
dict_values([25, 0.7, 100]) & 0.935846 & 0.039221 & 0.935846 & 0.039221 &
0.935846 & 0.039221 & 0.935846 & 0.039221 \\
```

```
dict_values([25, 0.7, 150]) & 0.935846 & 0.039221 & 0.935846 & 0.039221 &
0.935846 & 0.039221 & 0.935846 & 0.039221 \\
dict_values([25, 0.7, 200]) & 0.941252 & 0.042004 & 0.941252 & 0.042004 &
0.941252 & 0.042004 & 0.941252 & 0.042004 \\
dict_values([25, 1.0, 100]) & 0.935846 & 0.039221 & 0.935846 & 0.039221 &
0.935846 & 0.039221 & 0.935846 & 0.039221 \\
dict_values([25, 1.0, 150]) & 0.941252 & 0.042004 & 0.941252 & 0.042004 &
0.941252 & 0.042004 & 0.941252 & 0.042004 \\
dict_values([25, 1.0, 200]) & 0.941252 & 0.042004 & 0.941252 & 0.042004 &
0.941252 & 0.042004 & 0.941252 & 0.042004 \\
\bottomrule
\end{tabular}

\begin{tabular}{lrrrrrrrr}
\toprule
max_depth,max_samples,n_estimators & accuracy_score_D1 & accuracy_score_D1_std &
f1_score_D1 & f1_score_D1_std & precision_score_D1 & precision_score_D1_std &
recall_score_D1 & recall_score_D1_std \\
\midrule
dict_values([15, 0.5, 100]) & 0.790476 & 0.040963 & 0.790476 & 0.040963 &
0.790476 & 0.040963 & 0.790476 & 0.040963 \\
dict_values([15, 0.5, 150]) & 0.790476 & 0.034993 & 0.790476 & 0.034993 &
0.790476 & 0.034993 & 0.790476 & 0.034993 \\
dict_values([15, 0.5, 200]) & 0.800000 & 0.038686 & 0.800000 & 0.038686 &
0.800000 & 0.038686 & 0.800000 & 0.038686 \\
dict_values([15, 0.7, 100]) & 0.814286 & 0.046168 & 0.814286 & 0.046168 &
0.814286 & 0.046168 & 0.814286 & 0.046168 \\
dict_values([15, 0.7, 150]) & 0.828571 & 0.040963 & 0.828571 & 0.040963 &
0.828571 & 0.040963 & 0.828571 & 0.040963 \\
dict_values([15, 0.7, 200]) & 0.814286 & 0.034993 & 0.814286 & 0.034993 &
0.814286 & 0.034993 & 0.814286 & 0.034993 \\
dict_values([15, 1.0, 100]) & 0.819048 & 0.035635 & 0.819048 & 0.035635 &
0.819048 & 0.035635 & 0.819048 & 0.035635 \\
dict_values([15, 1.0, 150]) & 0.833333 & 0.026082 & 0.833333 & 0.026082 &
0.833333 & 0.026082 & 0.833333 & 0.026082 \\
dict_values([15, 1.0, 200]) & 0.828571 & 0.031587 & 0.828571 & 0.031587 &
0.828571 & 0.031587 & 0.828571 & 0.031587 \\
dict_values([20, 0.5, 100]) & 0.776190 & 0.063174 & 0.776190 & 0.063174 &
0.776190 & 0.063174 & 0.776190 & 0.063174 \\
dict_values([20, 0.5, 150]) & 0.785714 & 0.060234 & 0.785714 & 0.060234 &
0.785714 & 0.060234 & 0.785714 & 0.060234 \\
dict_values([20, 0.5, 200]) & 0.800000 & 0.044160 & 0.800000 & 0.044160 &
0.800000 & 0.044160 & 0.800000 & 0.044160 \\
dict_values([20, 0.7, 100]) & 0.809524 & 0.033672 & 0.809524 & 0.033672 &
0.809524 & 0.033672 & 0.809524 & 0.033672 \\
dict_values([20, 0.7, 150]) & 0.814286 & 0.046168 & 0.814286 & 0.046168 &
0.814286 & 0.046168 & 0.814286 & 0.046168 \\
dict_values([20, 0.7, 200]) & 0.814286 & 0.034993 & 0.814286 & 0.034993 &
```

```
0.814286 & 0.034993 & 0.814286 & 0.034993 \\
dict_values([20, 1.0, 100]) & 0.809524 & 0.039841 & 0.809524 & 0.039841 &
0.809524 & 0.039841 & 0.809524 & 0.039841 \\
dict_values([20, 1.0, 150]) & 0.833333 & 0.049943 & 0.833333 & 0.049943 &
0.833333 & 0.049943 & 0.833333 & 0.049943 \\
dict_values([20, 1.0, 200]) & 0.847619 & 0.046657 & 0.847619 & 0.046657 &
0.847619 & 0.046657 & 0.847619 & 0.046657 \\
dict_values([25, 0.5, 100]) & 0.776190 & 0.063174 & 0.776190 & 0.063174 &
0.776190 & 0.063174 & 0.776190 & 0.063174 \\
dict_values([25, 0.5, 150]) & 0.785714 & 0.060234 & 0.785714 & 0.060234 &
0.785714 & 0.060234 & 0.785714 & 0.060234 \\
dict_values([25, 0.5, 200]) & 0.800000 & 0.044160 & 0.800000 & 0.044160 &
0.800000 & 0.044160 & 0.800000 & 0.044160 \\
dict_values([25, 0.7, 100]) & 0.804762 & 0.040963 & 0.804762 & 0.040963 &
0.804762 & 0.040963 & 0.804762 & 0.040963 \\
dict_values([25, 0.7, 150]) & 0.814286 & 0.046168 & 0.814286 & 0.046168 &
0.814286 & 0.046168 & 0.814286 & 0.046168 \\
dict_values([25, 0.7, 200]) & 0.814286 & 0.034993 & 0.814286 & 0.034993 &
0.814286 & 0.034993 & 0.814286 & 0.034993 \\
dict_values([25, 1.0, 100]) & 0.809524 & 0.039841 & 0.809524 & 0.039841 &
0.809524 & 0.039841 & 0.809524 & 0.039841 \\
dict_values([25, 1.0, 150]) & 0.833333 & 0.049943 & 0.833333 & 0.049943 &
0.833333 & 0.049943 & 0.833333 & 0.049943 \\
dict_values([25, 1.0, 200]) & 0.847619 & 0.046657 & 0.847619 & 0.046657 &
0.847619 & 0.046657 & 0.847619 & 0.046657 \\
\bottomrule
\end{tabular}

\begin{tabular}{lrrrrrrrr}
\toprule
max_depth,max_samples,n_estimators & accuracy_score_D2 & accuracy_score_D2_std &
f1_score_D2 & f1_score_D2_std & precision_score_D2 & precision_score_D2_std &
recall_score_D2 & recall_score_D2_std \\
\midrule
dict_values([15, 0.5, 100]) & 0.814905 & 0.052647 & 0.814905 & 0.052647 &
0.814905 & 0.052647 & 0.814905 & 0.052647 \\
dict_values([15, 0.5, 150]) & 0.810254 & 0.064537 & 0.810254 & 0.064537 &
0.810254 & 0.064537 & 0.810254 & 0.064537 \\
dict_values([15, 0.5, 200]) & 0.810254 & 0.057443 & 0.810254 & 0.057443 &
0.810254 & 0.057443 & 0.810254 & 0.057443 \\
dict_values([15, 0.7, 100]) & 0.837844 & 0.066013 & 0.837844 & 0.066013 &
0.837844 & 0.066013 & 0.837844 & 0.066013 \\
dict_values([15, 0.7, 150]) & 0.837844 & 0.066013 & 0.837844 & 0.066013 &
0.837844 & 0.066013 & 0.837844 & 0.066013 \\
dict_values([15, 0.7, 200]) & 0.833298 & 0.064853 & 0.833298 & 0.064853 &
0.833298 & 0.064853 & 0.833298 & 0.064853 \\
dict_values([15, 1.0, 100]) & 0.837949 & 0.060662 & 0.837949 & 0.060662 &
0.837949 & 0.060662 & 0.837949 & 0.060662 \\
```

```
dict_values([15, 1.0, 150]) & 0.833298 & 0.063163 & 0.833298 & 0.063163 &
0.833298 & 0.063163 & 0.833298 & 0.063163 \\
dict_values([15, 1.0, 200]) & 0.828647 & 0.066876 & 0.828647 & 0.066876 &
0.828647 & 0.066876 & 0.828647 & 0.066876 \\
dict_values([20, 0.5, 100]) & 0.838055 & 0.041161 & 0.838055 & 0.041161 &
0.838055 & 0.041161 & 0.838055 & 0.041161 \\
dict_values([20, 0.5, 150]) & 0.828858 & 0.053619 & 0.828858 & 0.053619 &
0.828858 & 0.053619 & 0.828858 & 0.053619 \\
dict_values([20, 0.5, 200]) & 0.810254 & 0.057443 & 0.810254 & 0.057443 &
0.810254 & 0.057443 & 0.810254 & 0.057443 \\
dict_values([20, 0.7, 100]) & 0.828647 & 0.066876 & 0.828647 & 0.066876 &
0.828647 & 0.066876 & 0.828647 & 0.066876 \\
dict_values([20, 0.7, 150]) & 0.837844 & 0.066013 & 0.837844 & 0.066013 &
0.837844 & 0.066013 & 0.837844 & 0.066013 \\
dict_values([20, 0.7, 200]) & 0.842495 & 0.064977 & 0.842495 & 0.064977 &
0.842495 & 0.064977 & 0.842495 & 0.064977 \\
dict_values([20, 1.0, 100]) & 0.842495 & 0.061558 & 0.842495 & 0.061558 &
0.842495 & 0.061558 & 0.842495 & 0.061558 \\
dict_values([20, 1.0, 150]) & 0.842495 & 0.061558 & 0.842495 & 0.061558 &
0.842495 & 0.061558 & 0.842495 & 0.061558 \\
dict_values([20, 1.0, 200]) & 0.823996 & 0.071612 & 0.823996 & 0.071612 &
0.823996 & 0.071612 & 0.823996 & 0.071612 \\
dict_values([25, 0.5, 100]) & 0.838055 & 0.041161 & 0.838055 & 0.041161 &
0.838055 & 0.041161 & 0.838055 & 0.041161 \\
dict_values([25, 0.5, 150]) & 0.828858 & 0.053619 & 0.828858 & 0.053619 &
0.828858 & 0.053619 & 0.828858 & 0.053619 \\
dict_values([25, 0.5, 200]) & 0.810254 & 0.057443 & 0.810254 & 0.057443 &
0.810254 & 0.057443 & 0.810254 & 0.057443 \\
dict_values([25, 0.7, 100]) & 0.828647 & 0.066876 & 0.828647 & 0.066876 &
0.828647 & 0.066876 & 0.828647 & 0.066876 \\
dict_values([25, 0.7, 150]) & 0.837844 & 0.066013 & 0.837844 & 0.066013 &
0.837844 & 0.066013 & 0.837844 & 0.066013 \\
dict_values([25, 0.7, 200]) & 0.837844 & 0.069213 & 0.837844 & 0.069213 &
0.837844 & 0.069213 & 0.837844 & 0.069213 \\
dict_values([25, 1.0, 100]) & 0.842495 & 0.061558 & 0.842495 & 0.061558 &
0.842495 & 0.061558 & 0.842495 & 0.061558 \\
dict_values([25, 1.0, 150]) & 0.842495 & 0.061558 & 0.842495 & 0.061558 &
0.842495 & 0.061558 & 0.842495 & 0.061558 \\
dict_values([25, 1.0, 200]) & 0.823996 & 0.071612 & 0.823996 & 0.071612 &
0.823996 & 0.071612 & 0.823996 & 0.071612 \\
\bottomrule
\end{tabular}

\begin{tabular}{lrrrrrrrr}
\toprule
max_depth,max_samples,n_estimators & accuracy_score_D3 & accuracy_score_D3_std &
f1_score_D3 & f1_score_D3_std & precision_score_D3 & precision_score_D3_std &
recall_score_D3 & recall_score_D3_std \\
```

```
\midrule
dict_values([15, 0.5, 100]) & 0.877273 & 0.046798 & 0.877273 & 0.046798 &
0.877273 & 0.046798 & 0.877273 & 0.046798 \\
dict_values([15, 0.5, 150]) & 0.859091 & 0.048532 & 0.859091 & 0.048532 &
0.859091 & 0.048532 & 0.859091 & 0.048532 \\
dict_values([15, 0.5, 200]) & 0.859091 & 0.048532 & 0.859091 & 0.048532 &
0.859091 & 0.048532 & 0.859091 & 0.048532 \\
dict_values([15, 0.7, 100]) & 0.863636 & 0.014374 & 0.863636 & 0.014374 &
0.863636 & 0.014374 & 0.863636 & 0.014374 \\
dict_values([15, 0.7, 150]) & 0.872727 & 0.018182 & 0.872727 & 0.018182 &
0.872727 & 0.018182 & 0.872727 & 0.018182 \\
dict_values([15, 0.7, 200]) & 0.872727 & 0.018182 & 0.872727 & 0.018182 &
0.872727 & 0.018182 & 0.872727 & 0.018182 \\
dict_values([15, 1.0, 100]) & 0.890909 & 0.026504 & 0.890909 & 0.026504 &
0.890909 & 0.026504 & 0.890909 & 0.026504 \\
dict_values([15, 1.0, 150]) & 0.890909 & 0.017008 & 0.890909 & 0.017008 &
0.890909 & 0.017008 & 0.890909 & 0.017008 \\
dict_values([15, 1.0, 200]) & 0.886364 & 0.014374 & 0.886364 & 0.014374 &
0.886364 & 0.014374 & 0.886364 & 0.014374 \\
dict_values([20, 0.5, 100]) & 0.859091 & 0.046355 & 0.859091 & 0.046355 &
0.859091 & 0.046355 & 0.859091 & 0.046355 \\
dict_values([20, 0.5, 150]) & 0.854545 & 0.044536 & 0.854545 & 0.044536 &
0.854545 & 0.044536 & 0.854545 & 0.044536 \\
dict_values([20, 0.5, 200]) & 0.859091 & 0.048532 & 0.859091 & 0.048532 &
0.859091 & 0.048532 & 0.859091 & 0.048532 \\
dict_values([20, 0.7, 100]) & 0.859091 & 0.017008 & 0.859091 & 0.017008 &
0.859091 & 0.017008 & 0.859091 & 0.017008 \\
dict_values([20, 0.7, 150]) & 0.863636 & 0.020328 & 0.863636 & 0.020328 &
0.863636 & 0.020328 & 0.863636 & 0.020328 \\
dict_values([20, 0.7, 200]) & 0.854545 & 0.027273 & 0.854545 & 0.027273 &
0.854545 & 0.027273 & 0.854545 & 0.027273 \\
dict_values([20, 1.0, 100]) & 0.886364 & 0.032141 & 0.886364 & 0.032141 &
0.886364 & 0.032141 & 0.886364 & 0.032141 \\
dict_values([20, 1.0, 150]) & 0.890909 & 0.022268 & 0.890909 & 0.022268 &
0.890909 & 0.022268 & 0.890909 & 0.022268 \\
dict_values([20, 1.0, 200]) & 0.877273 & 0.027273 & 0.877273 & 0.027273 &
0.877273 & 0.027273 & 0.877273 & 0.027273 \\
dict_values([25, 0.5, 100]) & 0.859091 & 0.046355 & 0.859091 & 0.046355 &
0.859091 & 0.046355 & 0.859091 & 0.046355 \\
dict_values([25, 0.5, 150]) & 0.854545 & 0.044536 & 0.854545 & 0.044536 &
0.854545 & 0.044536 & 0.854545 & 0.044536 \\
dict_values([25, 0.5, 200]) & 0.859091 & 0.048532 & 0.859091 & 0.048532 &
0.859091 & 0.048532 & 0.859091 & 0.048532 \\
dict_values([25, 0.7, 100]) & 0.859091 & 0.017008 & 0.859091 & 0.017008 &
0.859091 & 0.017008 & 0.859091 & 0.017008 \\
dict_values([25, 0.7, 150]) & 0.863636 & 0.020328 & 0.863636 & 0.020328 &
0.863636 & 0.020328 & 0.863636 & 0.020328 \\
dict_values([25, 0.7, 200]) & 0.854545 & 0.027273 & 0.854545 & 0.027273 &
```

```
0.854545 & 0.027273 & 0.854545 & 0.027273 \\
dict_values([25, 1.0, 100]) & 0.886364 & 0.032141 & 0.886364 & 0.032141 &
0.886364 & 0.032141 & 0.886364 & 0.032141 \\
dict_values([25, 1.0, 150]) & 0.890909 & 0.022268 & 0.890909 & 0.022268 &
0.890909 & 0.022268 & 0.890909 & 0.022268 \\
dict_values([25, 1.0, 200]) & 0.877273 & 0.027273 & 0.877273 & 0.027273 &
0.877273 & 0.027273 & 0.877273 & 0.027273 \\
\bottomrule
\end{tabular}

\begin{tabular}{lrrrrrrrr}
\toprule
max_depth,max_samples,n_estimators & accuracy_score_D4 & accuracy_score_D4_std &
f1_score_D4 & f1_score_D4_std & precision_score_D4 & precision_score_D4_std &
recall_score_D4 & recall_score_D4_std \\
\midrule
dict_values([15, 0.5, 100]) & 0.920235 & 0.038074 & 0.920235 & 0.038074 &
0.920235 & 0.038074 & 0.920235 & 0.038074 \\
dict_values([15, 0.5, 150]) & 0.912235 & 0.032702 & 0.912235 & 0.032702 &
0.912235 & 0.032702 & 0.912235 & 0.032702 \\
dict_values([15, 0.5, 200]) & 0.912235 & 0.027375 & 0.912235 & 0.027375 &
0.912235 & 0.027375 & 0.912235 & 0.027375 \\
dict_values([15, 0.7, 100]) & 0.924235 & 0.038881 & 0.924235 & 0.038881 &
0.924235 & 0.038881 & 0.924235 & 0.038881 \\
dict_values([15, 0.7, 150]) & 0.932235 & 0.037148 & 0.932235 & 0.037148 &
0.932235 & 0.037148 & 0.932235 & 0.037148 \\
dict_values([15, 0.7, 200]) & 0.924235 & 0.023489 & 0.924235 & 0.023489 &
0.924235 & 0.023489 & 0.924235 & 0.023489 \\
dict_values([15, 1.0, 100]) & 0.916157 & 0.040962 & 0.916157 & 0.040962 &
0.916157 & 0.040962 & 0.916157 & 0.040962 \\
dict_values([15, 1.0, 150]) & 0.920157 & 0.035953 & 0.920157 & 0.035953 &
0.920157 & 0.035953 & 0.920157 & 0.035953 \\
dict_values([15, 1.0, 200]) & 0.928157 & 0.041305 & 0.928157 & 0.041305 &
0.928157 & 0.041305 & 0.928157 & 0.041305 \\
dict_values([20, 0.5, 100]) & 0.916235 & 0.038930 & 0.916235 & 0.038930 &
0.916235 & 0.038930 & 0.916235 & 0.038930 \\
dict_values([20, 0.5, 150]) & 0.916235 & 0.038930 & 0.916235 & 0.038930 &
0.916235 & 0.038930 & 0.916235 & 0.038930 \\
dict_values([20, 0.5, 200]) & 0.908235 & 0.027409 & 0.908235 & 0.027409 &
0.908235 & 0.027409 & 0.908235 & 0.027409 \\
dict_values([20, 0.7, 100]) & 0.916235 & 0.032179 & 0.916235 & 0.032179 &
0.916235 & 0.032179 & 0.916235 & 0.032179 \\
dict_values([20, 0.7, 150]) & 0.924235 & 0.029525 & 0.924235 & 0.029525 &
0.924235 & 0.029525 & 0.924235 & 0.029525 \\
dict_values([20, 0.7, 200]) & 0.920235 & 0.028454 & 0.920235 & 0.028454 &
0.920235 & 0.028454 & 0.920235 & 0.028454 \\
dict_values([20, 1.0, 100]) & 0.920157 & 0.033655 & 0.920157 & 0.033655 &
0.920157 & 0.033655 & 0.920157 & 0.033655 \\
```

```
dict_values([20, 1.0, 150]) & 0.928157 & 0.037231 & 0.928157 & 0.037231 &
0.928157 & 0.037231 & 0.928157 & 0.037231 \\
dict_values([20, 1.0, 200]) & 0.924157 & 0.038928 & 0.924157 & 0.038928 &
0.924157 & 0.038928 & 0.924157 & 0.038928 \\
dict_values([25, 0.5, 100]) & 0.916235 & 0.038930 & 0.916235 & 0.038930 &
0.916235 & 0.038930 & 0.916235 & 0.038930 \\
dict_values([25, 0.5, 150]) & 0.916235 & 0.038930 & 0.916235 & 0.038930 &
0.916235 & 0.038930 & 0.916235 & 0.038930 \\
dict_values([25, 0.5, 200]) & 0.908235 & 0.027409 & 0.908235 & 0.027409 &
0.908235 & 0.027409 & 0.908235 & 0.027409 \\
dict_values([25, 0.7, 100]) & 0.916235 & 0.032179 & 0.916235 & 0.032179 &
0.916235 & 0.032179 & 0.916235 & 0.032179 \\
dict_values([25, 0.7, 150]) & 0.924235 & 0.029525 & 0.924235 & 0.029525 &
0.924235 & 0.029525 & 0.924235 & 0.029525 \\
dict_values([25, 0.7, 200]) & 0.920235 & 0.028454 & 0.920235 & 0.028454 &
0.920235 & 0.028454 & 0.920235 & 0.028454 \\
dict_values([25, 1.0, 100]) & 0.920157 & 0.033655 & 0.920157 & 0.033655 &
0.920157 & 0.033655 & 0.920157 & 0.033655 \\
dict_values([25, 1.0, 150]) & 0.928157 & 0.037231 & 0.928157 & 0.037231 &
0.928157 & 0.037231 & 0.928157 & 0.037231 \\
dict_values([25, 1.0, 200]) & 0.924157 & 0.038928 & 0.924157 & 0.038928 &
0.924157 & 0.038928 & 0.924157 & 0.038928 \\
\bottomrule
\end{tabular}

\begin{tabular}{lrrrrrrrr}
\toprule
max_depth,max_samples,n_estimators & accuracy_score_D5 & accuracy_score_D5_std &
f1_score_D5 & f1_score_D5_std & precision_score_D5 & precision_score_D5_std &
recall_score_D5 & recall_score_D5_std \\
\midrule
dict_values([15, 0.5, 100]) & 0.971429 & 0.023328 & 0.971429 & 0.023328 &
0.971429 & 0.023328 & 0.971429 & 0.023328 \\
dict_values([15, 0.5, 150]) & 0.966667 & 0.032297 & 0.966667 & 0.032297 &
0.966667 & 0.032297 & 0.966667 & 0.032297 \\
dict_values([15, 0.5, 200]) & 0.966667 & 0.032297 & 0.966667 & 0.032297 &
0.966667 & 0.032297 & 0.966667 & 0.032297 \\
dict_values([15, 0.7, 100]) & 0.966667 & 0.032297 & 0.966667 & 0.032297 &
0.966667 & 0.032297 & 0.966667 & 0.032297 \\
dict_values([15, 0.7, 150]) & 0.966667 & 0.032297 & 0.966667 & 0.032297 &
0.966667 & 0.032297 & 0.966667 & 0.032297 \\
dict_values([15, 0.7, 200]) & 0.966667 & 0.032297 & 0.966667 & 0.032297 &
0.966667 & 0.032297 & 0.966667 & 0.032297 \\
dict_values([15, 1.0, 100]) & 0.971429 & 0.023328 & 0.971429 & 0.023328 &
0.971429 & 0.023328 & 0.971429 & 0.023328 \\
dict_values([15, 1.0, 150]) & 0.971429 & 0.023328 & 0.971429 & 0.023328 &
0.971429 & 0.023328 & 0.971429 & 0.023328 \\
dict_values([15, 1.0, 200]) & 0.971429 & 0.023328 & 0.971429 & 0.023328 &
```

```
0.971429 & 0.023328 & 0.971429 & 0.023328 \\
dict_values([20, 0.5, 100]) & 0.971429 & 0.023328 & 0.971429 & 0.023328 &
0.971429 & 0.023328 & 0.971429 & 0.023328 \\
dict_values([20, 0.5, 150]) & 0.971429 & 0.023328 & 0.971429 & 0.023328 &
0.971429 & 0.023328 & 0.971429 & 0.023328 \\
dict_values([20, 0.5, 200]) & 0.971429 & 0.023328 & 0.971429 & 0.023328 &
0.971429 & 0.023328 & 0.971429 & 0.023328 \\
dict_values([20, 0.7, 100]) & 0.966667 & 0.032297 & 0.966667 & 0.032297 &
0.966667 & 0.032297 & 0.966667 & 0.032297 \\
dict_values([20, 0.7, 150]) & 0.966667 & 0.032297 & 0.966667 & 0.032297 &
0.966667 & 0.032297 & 0.966667 & 0.032297 \\
dict_values([20, 0.7, 200]) & 0.966667 & 0.032297 & 0.966667 & 0.032297 &
0.966667 & 0.032297 & 0.966667 & 0.032297 \\
dict_values([20, 1.0, 100]) & 0.971429 & 0.023328 & 0.971429 & 0.023328 &
0.971429 & 0.023328 & 0.971429 & 0.023328 \\
dict_values([20, 1.0, 150]) & 0.971429 & 0.023328 & 0.971429 & 0.023328 &
0.971429 & 0.023328 & 0.971429 & 0.023328 \\
dict_values([20, 1.0, 200]) & 0.971429 & 0.023328 & 0.971429 & 0.023328 &
0.971429 & 0.023328 & 0.971429 & 0.023328 \\
dict_values([25, 0.5, 100]) & 0.971429 & 0.023328 & 0.971429 & 0.023328 &
0.971429 & 0.023328 & 0.971429 & 0.023328 \\
dict_values([25, 0.5, 150]) & 0.971429 & 0.023328 & 0.971429 & 0.023328 &
0.971429 & 0.023328 & 0.971429 & 0.023328 \\
dict_values([25, 0.5, 200]) & 0.971429 & 0.023328 & 0.971429 & 0.023328 &
0.971429 & 0.023328 & 0.971429 & 0.023328 \\
dict_values([25, 0.7, 100]) & 0.966667 & 0.032297 & 0.966667 & 0.032297 &
0.966667 & 0.032297 & 0.966667 & 0.032297 \\
dict_values([25, 0.7, 150]) & 0.966667 & 0.032297 & 0.966667 & 0.032297 &
0.966667 & 0.032297 & 0.966667 & 0.032297 \\
dict_values([25, 0.7, 200]) & 0.966667 & 0.032297 & 0.966667 & 0.032297 &
0.966667 & 0.032297 & 0.966667 & 0.032297 \\
dict_values([25, 1.0, 100]) & 0.971429 & 0.023328 & 0.971429 & 0.023328 &
0.971429 & 0.023328 & 0.971429 & 0.023328 \\
dict_values([25, 1.0, 150]) & 0.971429 & 0.023328 & 0.971429 & 0.023328 &
0.971429 & 0.023328 & 0.971429 & 0.023328 \\
dict_values([25, 1.0, 200]) & 0.971429 & 0.023328 & 0.971429 & 0.023328 &
0.971429 & 0.023328 & 0.971429 & 0.023328 \\
\bottomrule
\end{tabular}

\begin{tabular}{lrrrrrrrr}
\toprule
max_depth,max_samples,n_estimators & accuracy_score_D6 & accuracy_score_D6_std &
f1_score_D6 & f1_score_D6_std & precision_score_D6 & precision_score_D6_std &
recall_score_D6 & recall_score_D6_std \\
\midrule
dict_values([15, 0.5, 100]) & 0.995918 & 0.008163 & 0.995918 & 0.008163 &
0.995918 & 0.008163 & 0.995918 & 0.008163 \\
```

```
dict_values([15, 0.5, 150]) & 0.995918 & 0.008163 & 0.995918 & 0.008163 &
0.995918 & 0.008163 & 0.995918 & 0.008163 \\
dict_values([15, 0.5, 200]) & 0.995918 & 0.008163 & 0.995918 & 0.008163 &
0.995918 & 0.008163 & 0.995918 & 0.008163 \\
dict_values([15, 0.7, 100]) & 0.995918 & 0.008163 & 0.995918 & 0.008163 &
0.995918 & 0.008163 & 0.995918 & 0.008163 \\
dict_values([15, 0.7, 150]) & 0.995918 & 0.008163 & 0.995918 & 0.008163 &
0.995918 & 0.008163 & 0.995918 & 0.008163 \\
dict_values([15, 0.7, 200]) & 0.995918 & 0.008163 & 0.995918 & 0.008163 &
0.995918 & 0.008163 & 0.995918 & 0.008163 \\
dict_values([15, 1.0, 100]) & 0.995918 & 0.008163 & 0.995918 & 0.008163 &
0.995918 & 0.008163 & 0.995918 & 0.008163 \\
dict_values([15, 1.0, 150]) & 0.995918 & 0.008163 & 0.995918 & 0.008163 &
0.995918 & 0.008163 & 0.995918 & 0.008163 \\
dict_values([15, 1.0, 200]) & 0.995918 & 0.008163 & 0.995918 & 0.008163 &
0.995918 & 0.008163 & 0.995918 & 0.008163 \\
dict_values([20, 0.5, 100]) & 0.995918 & 0.008163 & 0.995918 & 0.008163 &
0.995918 & 0.008163 & 0.995918 & 0.008163 \\
dict_values([20, 0.5, 150]) & 0.995918 & 0.008163 & 0.995918 & 0.008163 &
0.995918 & 0.008163 & 0.995918 & 0.008163 \\
dict_values([20, 0.5, 200]) & 0.995918 & 0.008163 & 0.995918 & 0.008163 &
0.995918 & 0.008163 & 0.995918 & 0.008163 \\
dict_values([20, 0.7, 100]) & 0.995918 & 0.008163 & 0.995918 & 0.008163 &
0.995918 & 0.008163 & 0.995918 & 0.008163 \\
dict_values([20, 0.7, 150]) & 0.995918 & 0.008163 & 0.995918 & 0.008163 &
0.995918 & 0.008163 & 0.995918 & 0.008163 \\
dict_values([20, 0.7, 200]) & 0.995918 & 0.008163 & 0.995918 & 0.008163 &
0.995918 & 0.008163 & 0.995918 & 0.008163 \\
dict_values([20, 1.0, 100]) & 0.995918 & 0.008163 & 0.995918 & 0.008163 &
0.995918 & 0.008163 & 0.995918 & 0.008163 \\
dict_values([20, 1.0, 150]) & 0.995918 & 0.008163 & 0.995918 & 0.008163 &
0.995918 & 0.008163 & 0.995918 & 0.008163 \\
dict_values([20, 1.0, 200]) & 0.995918 & 0.008163 & 0.995918 & 0.008163 &
0.995918 & 0.008163 & 0.995918 & 0.008163 \\
dict_values([25, 0.5, 100]) & 0.995918 & 0.008163 & 0.995918 & 0.008163 &
0.995918 & 0.008163 & 0.995918 & 0.008163 \\
dict_values([25, 0.5, 150]) & 0.995918 & 0.008163 & 0.995918 & 0.008163 &
0.995918 & 0.008163 & 0.995918 & 0.008163 \\
dict_values([25, 0.5, 200]) & 0.995918 & 0.008163 & 0.995918 & 0.008163 &
0.995918 & 0.008163 & 0.995918 & 0.008163 \\
dict_values([25, 0.7, 100]) & 0.995918 & 0.008163 & 0.995918 & 0.008163 &
0.995918 & 0.008163 & 0.995918 & 0.008163 \\
dict_values([25, 0.7, 150]) & 0.995918 & 0.008163 & 0.995918 & 0.008163 &
0.995918 & 0.008163 & 0.995918 & 0.008163 \\
dict_values([25, 0.7, 200]) & 0.995918 & 0.008163 & 0.995918 & 0.008163 &
0.995918 & 0.008163 & 0.995918 & 0.008163 \\
dict_values([25, 1.0, 100]) & 0.995918 & 0.008163 & 0.995918 & 0.008163 &
0.995918 & 0.008163 & 0.995918 & 0.008163 \\
```

```
dict_values([25, 1.0, 150]) & 0.995918 & 0.008163 & 0.995918 & 0.008163 &
0.995918 & 0.008163 & 0.995918 & 0.008163 \\
dict_values([25, 1.0, 200]) & 0.995918 & 0.008163 & 0.995918 & 0.008163 &
0.995918 & 0.008163 & 0.995918 & 0.008163 \\
\bottomrule
\end{tabular}

              precision    recall  f1-score   support

          D0       0.96      0.80      0.87        60
          D1       0.75      0.88      0.81        52
          D2       0.86      0.86      0.86        42
          D3       0.88      0.90      0.89        49
          D4       0.94      0.86      0.89        69
          D5       0.91      0.98      0.94        60
          D6       0.96      0.98      0.97        53

    accuracy                           0.89       385
   macro avg       0.89      0.89      0.89       385
weighted avg       0.90      0.89      0.89       385

Accuracy: 0.8935064935064935
F1 Score: 0.8937436674764665
Precision: 0.8988330853810947
Recall: 0.8935064935064935
16
```

[ ]:
```python
# Initialize a RandomForestClassifier
from sklearn.ensemble import ExtraTreesClassifier


clf = ExtraTreesClassifier(random_state=RANDOM_STATE)


# Define the parameter grid
param_grid = {
    'n_estimators': [100, 150, 200, 250],
    'max_depth': [10, 15, 20, 25, 30],
    'max_samples': [0.5, 0.7, 0.8, 1.0],
    'bootstrap': [True]
}
X, y = prepare_dataset()
grid_search, best_params = find_best_params(clf, param_grid, X, y, RANDOM_STATE)
best_clf = ExtraTreesClassifier(**best_params, random_state=RANDOM_STATE)
accuracy, f1, precision, recall = evaluate_my_model(best_clf, X, y,␣
 ↪RANDOM_STATE)
print(f"Accuracy: {accuracy}")
```

```
print(f"F1 Score: {f1}")
print(f"Precision: {precision}")
print(f"Recall: {recall}")
```

```
Features removed: {'Calorie_monitoring', 'Smoker'}
Features removed: {'Est_avg_calorie_intake', 'Physical_activity_level',
'Technology_time_use', 'Sedentary_hours_daily', 'Water_daily'}
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
Cell In[166], line 16
      9 param_grid = {
     10     'n_estimators': [100, 150, 200, 250],
     11     'max_depth': [10, 15, 20, 25, 30],
     12     'max_samples': [0.5, 0.7, 0.8, 1.0],
     13     'bootstrap': [True]
     14 }
     15 X, y = prepare_dataset()
---> 16 grid_search, best_params =␣
 ↪find_best_params(clf, param_grid, X, y, RANDOM_STATE)
     17 best_clf = ExtraTreesClassifier(**best_params, random_state=RANDOM_STATE)
     18 accuracy, f1, precision, recall = evaluate_my_model(best_clf, X, y,␣
 ↪RANDOM_STATE)

Cell In[163], line 108, in find_best_params(classifier, param_grid, X, y,␣
 ↪random_state)
     99 grid_search = GridSearchCV(
    100     estimator=classifier,
    101     param_grid=param_grid,
   (…)
    105     n_jobs=4,
    106 )
    107 # Fit the GridSearchCV to the training data
--> 108 grid_search.fit(X_train, y_train)
    110 # Print the best parameters
    111 print("Best parameters found: ", grid_search.best_params_)

File ~/.local/lib/python3.10/site-packages/sklearn/base.py:1474, in _fit_context.
 ↪<locals>.decorator.<locals>.wrapper(estimator, *args, **kwargs)
   1467     estimator._validate_params()
   1469 with config_context(
   1470     skip_parameter_validation=(
   1471         prefer_skip_nested_validation or global_skip_validation
   1472     )
   1473 ):
-> 1474     return fit_method(estimator, *args, **kwargs)
```

```
File ~/.local/lib/python3.10/site-packages/sklearn/model_selection/_search.py:970, in BaseSearchCV.fit(self, X, y, **params)
    964     results = self._format_results(
    965         all_candidate_params, n_splits, all_out, all_more_results
    966     )
    968     return results
--> 970 self._run_search(evaluate_candidates)
    972 # multimetric is determined here because in the case of a callable
    973 # self.scoring the return type is only known after calling
    974 first_test_score = all_out[0]["test_scores"]

File ~/.local/lib/python3.10/site-packages/sklearn/model_selection/_search.py:1527, in GridSearchCV._run_search(self, evaluate_candidates)
   1525 def _run_search(self, evaluate_candidates):
   1526     """Search all candidates in param_grid"""
-> 1527     evaluate_candidates(ParameterGrid(self.param_grid))

File ~/.local/lib/python3.10/site-packages/sklearn/model_selection/_search.py:916, in BaseSearchCV.fit.<locals>.evaluate_candidates(candidate_params, cv, more_results)
    908 if self.verbose > 0:
    909     print(
    910         "Fitting {0} folds for each of {1} candidates,"
    911         " totalling {2} fits".format(
    912             n_splits, n_candidates, n_candidates * n_splits
    913         )
    914     )
--> 916 out = parallel(
    917     delayed(_fit_and_score)(
    918         clone(base_estimator),
    919         X,
    920         y,
    921         train=train,
    922         test=test,
    923         parameters=parameters,
    924         split_progress=(split_idx, n_splits),
    925         candidate_progress=(cand_idx, n_candidates),
    926         **fit_and_score_kwargs,
    927     )
    928     for (cand_idx, parameters), (split_idx, (train, test)) in product(
    929         enumerate(candidate_params),
    930         enumerate(cv.split(X, y, **routed_params.splitter.split)),
    931     )
    932 )
    934 if len(out) < 1:
    935     raise ValueError(
    936         "No fits were performed. "
    937         "Was the CV iterator empty? "
```

```
    938         "Were there no candidates?"
    939     )

File ~/.local/lib/python3.10/site-packages/sklearn/utils/parallel.py:67, in
↪Parallel.__call__(self, iterable)
     62 config = get_config()
     63 iterable_with_config = (
     64     (_with_config(delayed_func, config), args, kwargs)
     65     for delayed_func, args, kwargs in iterable
     66 )
---> 67 return super().__call__(iterable_with_config)

File ~/.local/lib/python3.10/site-packages/joblib/parallel.py:1952, in Parallel
↪__call__(self, iterable)
   1946 # The first item from the output is blank, but it makes the interpreter
   1947 # progress until it enters the Try/Except block of the generator and
   1948 # reach the first `yield` statement. This starts the aynchronous
   1949 # dispatch of the tasks to the workers.
   1950 next(output)
-> 1952 return output if self.return_generator else list(output)

File ~/.local/lib/python3.10/site-packages/joblib/parallel.py:1595, in Parallel
↪_get_outputs(self, iterator, pre_dispatch)
   1592     yield
   1594     with self._backend.retrieval_context():
-> 1595         yield from self._retrieve()
   1597 except GeneratorExit:
   1598     # The generator has been garbage collected before being fully
   1599     # consumed. This aborts the remaining tasks if possible and warn
   1600     # the user if necessary.
   1601     self._exception = True

File ~/.local/lib/python3.10/site-packages/joblib/parallel.py:1707, in Parallel
↪_retrieve(self)
   1702 # If the next job is not ready for retrieval yet, we just wait for
   1703 # async callbacks to progress.
   1704 if ((len(self._jobs) == 0) or
   1705     (self._jobs[0].get_status(
   1706         timeout=self.timeout) == TASK_PENDING)):
-> 1707     time.sleep(0.01)
   1708     continue
   1710 # We need to be careful: the job list can be filling up as
   1711 # we empty it and Python list are not thread-safe by
   1712 # default hence the use of the lock

KeyboardInterrupt:
```

[ ]:
```python
from xgboost import XGBClassifier

# Initialize a XGBClassifier
clf = XGBClassifier(random_state=RANDOM_STATE)

# Define the parameter grid
param_grid = {
    'n_estimators': [100, 150, 200],
    'max_depth': [10, 15, 20, 25],
    'learning_rate': [0.01, 0.1, 0.2],
}

X, y = prepare_dataset()
best_params = find_best_params(clf, param_grid, X, y, RANDOM_STATE)
best_clf = XGBClassifier(**best_params, random_state=RANDOM_STATE)
accuracy, f1, precision, recall = evaluate_my_model(best_clf, X, y,␣
 ↪RANDOM_STATE)
print(f"Accuracy: {accuracy}")
print(f"F1 Score: {f1}")
print(f"Precision: {precision}")
print(f"Recall: {recall}")
```

[ ]:
```python
from sklearn.svm import SVC

# Initialize a SVC
clf = SVC(random_state=RANDOM_STATE)

# Define the parameter grid
param_grid = {
    'C': [0.1, 1, 10, 100, 200, 300, 400, 500, 600, 700, 800, 900, 1000],
    'kernel': ['linear', 'rbf', 'poly', 'sigmoid']
}

X, y = prepare_dataset()
best_params = find_best_params(clf, param_grid, X, y, RANDOM_STATE)
best_clf = SVC(**best_params, random_state=RANDOM_STATE)
accuracy, f1, precision, recall = evaluate_my_model(best_clf, X, y,␣
 ↪RANDOM_STATE)
print(f"Accuracy: {accuracy}")
print(f"F1 Score: {f1}")
print(f"Precision: {precision}")
print(f"Recall: {recall}")
```