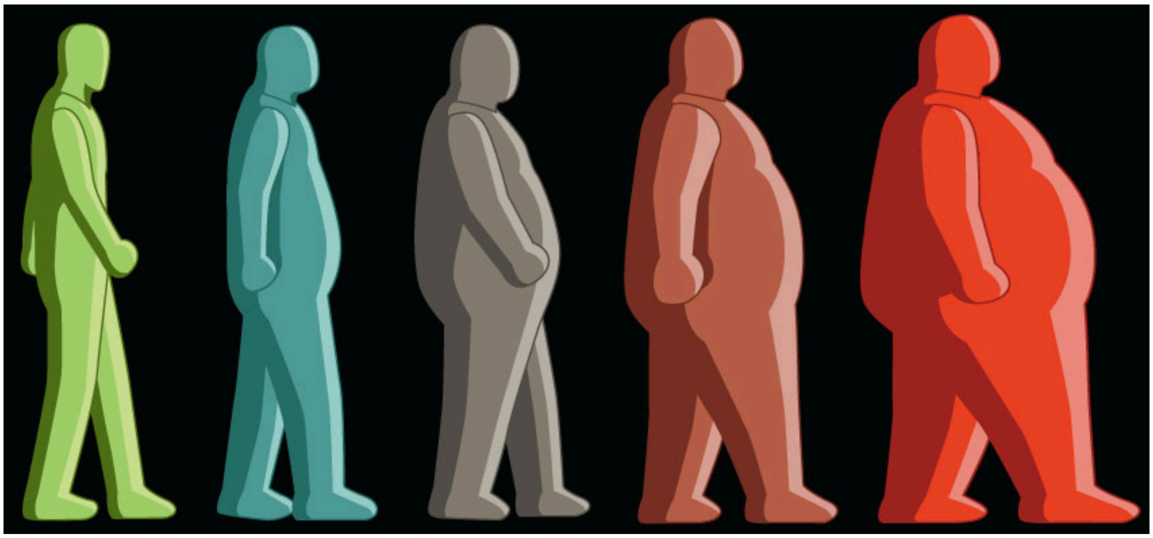
# Estimation of obesity levels based on eating habits and physical condition - DataTalks.Club's midterm project by Alexander D. Rios



Obesity is a global problem, and an individual's eating habits and physical condition can provide insights into their health status. Poor eating habits and a lack of physical activity can lead to severe obesity, which may have fatal consequences.

In this project, I used data on individuals from Colombia, Peru, and Mexico obtained from the UC Irvine Machine Learning Repository. The dataset contains 17 attributes and 2,111 records, with each record labeled using the class variable `NObesity` (Obesity Level). This variable categorizes obesity levels into the following classes:

- Insufficient Weight
- Normal Weight
- Overweight Level I
- Overweight Level II
- Obesity Type I
- Obesity Type II
- Obesity Type III

Seventy-seven percent of the data was generated synthetically using the Weka tool with the SMOTE filter, while 23% was collected directly from users via a web platform.

## Downloading the dataset

In [5]:
```
!wget https://archive.ics.uci.edu/static/public/544/estimation+of+obesity+le
!unzip -o /content/estimation+of+obesity+levels+based+on+eating+habits+and+p
```

```
--2024-11-25 14:16:53--  https://archive.ics.uci.edu/static/public/544/estima
tion+of+obesity+levels+based+on+eating+habits+and+physical+condition.zip
Resolving archive.ics.uci.edu (archive.ics.uci.edu)... 128.195.10.252
Connecting to archive.ics.uci.edu (archive.ics.uci.edu)|128.195.10.252|:44
3... connected.
HTTP request sent, awaiting response... 200 OK
Length: unspecified
Saving to: 'estimation+of+obesity+levels+based+on+eating+habits+and+physical+
condition.zip'

estimation+of+obesi     [ <=>                ]  56.32K  --.-KB/s    in 0.1s

2024-11-25 14:16:53 (433 KB/s) - 'estimation+of+obesity+levels+based+on+eatin
g+habits+and+physical+condition.zip' saved [57676]

Archive:  /content/estimation+of+obesity+levels+based+on+eating+habits+and+ph
ysical+condition.zip
  inflating: ObesityDataSet_raw_and_data_sinthetic.csv
```

# Installing some packages

In [56]:
```
!pip install mplcyberpunk
!pip install catboost

from IPython.display import clear_output

clear_output(wait=False)
```

# Feature description

| Feature | Description |
|---|---|
| Gender | |
| Age | |
| Height | |
| Weight | |
| family_history_with_overweight | Has a family member suffered or suffers from overweight? |
| FAVC | Do you eat high caloric food frequently? |
| FCVC | Do you usually eat vegetables in your meals? |
| NCP | How many main meals do you have daily? |
| CAEC | Do you eat any food between meals? |
| SMOKE | Do you smoke? |

| | |
|---|---|
| CH20 | How much water do you drink daily? |
| SCC | Do you monitor the calories you eat daily? |
| FAF | How often do you have physical activity? |
| TUE | How much time do you use technological devices such as cell phone, videogames, television, computer and others? |
| CALC | How often do you drink alcohol? |
| MTRANS | Which transportation do you usually use? |
| NObeyesdad | Obesity level |

# Loading the dataset

In [7]:
```
import pandas as pd
df_raw = pd.read_csv("/content/ObesityDataSet_raw_and_data_sinthetic.csv")
df_raw.head()
```

Out[7]:

| | Gender | Age | Height | Weight | family_history_with_overweight | FAVC | FCVC | NCP | |
|---|---|---|---|---|---|---|---|---|---|
| 0 | Female | 21.0 | 1.62 | 64.0 | yes | no | 2.0 | 3.0 | Son |
| 1 | Female | 21.0 | 1.52 | 56.0 | yes | no | 3.0 | 3.0 | Son |
| 2 | Male | 23.0 | 1.80 | 77.0 | yes | no | 2.0 | 3.0 | Son |
| 3 | Male | 27.0 | 1.80 | 87.0 | no | no | 3.0 | 3.0 | Son |
| 4 | Male | 22.0 | 1.78 | 89.8 | no | no | 2.0 | 1.0 | Son |

# Exploratory data analysis (EDA)

## Renaming the columns

In [8]:
```
df_raw = df_raw.rename(columns={"family_history_with_overweight": "overweigh
                       "FAVC":"eat_HC_food",
                       "FCVC":"eat_vegetables",
                       "NCP":"main_meals",
                       "CAEC":"snack",
                       "CH2O":"drink_water",
                       "SCC":"monitoring_calories",
                       "FAF":"physical_activity",
                       "TUE":"use_of_technology",
                       "CALC":"drink_alcohol",
                       "MTRANS":"transportation_type",
                       "NObeyesdad":"obesity_level"
                       }).rename(columns=str.lower)
df_raw.columns
```

Loading [MathJax]/extensions/Safe.js

Out[8]:
```
Index(['gender', 'age', 'height', 'weight', 'overweight_familiar',
       'eat_hc_food', 'eat_vegetables', 'main_meals', 'snack', 'smoke',
       'drink_water', 'monitoring_calories', 'physical_activity',
       'use_of_technology', 'drink_alcohol', 'transportation_type',
       'obesity_level'],
      dtype='object')
```

## Dropping duplicated

In [9]:
```
df_raw.duplicated().sum()
```

Out[9]:
```
24
```

In [10]:
```
df_raw = df_raw.drop_duplicates()
df_raw.duplicated().sum()
```

Out[10]:
```
0
```

## Looking for missing values

In [11]:
```
df_raw.isna().sum()
```

Out[11]:

| | 0 |
|---|---|
| **gender** | 0 |
| **age** | 0 |
| **height** | 0 |
| **weight** | 0 |
| **overweight_familiar** | 0 |
| **eat_hc_food** | 0 |
| **eat_vegetables** | 0 |
| **main_meals** | 0 |
| **snack** | 0 |
| **smoke** | 0 |
| **drink_water** | 0 |
| **monitoring_calories** | 0 |
| **physical_activity** | 0 |
| **use_of_technology** | 0 |
| **drink_alcohol** | 0 |
| **transportation_type** | 0 |
| **obesity_level** | 0 |

**dtype:** int64

# Statistical description

In [12]:

```
df_raw.describe(include="all").T
```

Out [12]:

| | count | unique | top | freq | mean | std | min |
|---|---|---|---|---|---|---|---|
| **gender** | 2087 | 2 | Male | 1052 | NaN | NaN | NaN |
| **age** | 2087.0 | NaN | NaN | NaN | 24.35309 | 6.368801 | 14.0 |
| **height** | 2087.0 | NaN | NaN | NaN | 1.702674 | 0.093186 | 1.45 |
| **weight** | 2087.0 | NaN | NaN | NaN | 86.85873 | 26.190847 | 39.0 |
| **overweight_familiar** | 2087 | 2 | yes | 1722 | NaN | NaN | NaN |
| **eat_hc_food** | 2087 | 2 | yes | 1844 | NaN | NaN | NaN |
| **eat_vegetables** | 2087.0 | NaN | NaN | NaN | 2.421466 | 0.534737 | 1.0 |
| **main_meals** | 2087.0 | NaN | NaN | NaN | 2.701179 | 0.764614 | 1.0 |
| **snack** | 2087 | 4 | Sometimes | 1761 | NaN | NaN | NaN |
| **smoke** | 2087 | 2 | no | 2043 | NaN | NaN | NaN |
| **drink_water** | 2087.0 | NaN | NaN | NaN | 2.004749 | 0.608284 | 1.0 |
| **monitoring_calories** | 2087 | 2 | no | 1991 | NaN | NaN | NaN |
| **physical_activity** | 2087.0 | NaN | NaN | NaN | 1.012812 | 0.853475 | 0.0 |
| **use_of_technology** | 2087.0 | NaN | NaN | NaN | 0.663035 | 0.608153 | 0.0 |
| **drink_alcohol** | 2087 | 4 | Sometimes | 1380 | NaN | NaN | NaN |
| **transportation_type** | 2087 | 5 | Public_Transportation | 1558 | NaN | NaN | NaN |
| **obesity_level** | 2087 | 7 | Obesity_Type_I | 351 | NaN | NaN | NaN |

# Looking for the classes in categorical features

In [13]:

```
df_raw.head()
```

Out [13]:

| | gender | age | height | weight | overweight_familiar | eat_hc_food | eat_vegetables | main_r |
|---|---|---|---|---|---|---|---|---|
| **0** | Female | 21.0 | 1.62 | 64.0 | yes | no | 2.0 | |
| **1** | Female | 21.0 | 1.52 | 56.0 | yes | no | 3.0 | |
| **2** | Male | 23.0 | 1.80 | 77.0 | yes | no | 2.0 | |
| **3** | Male | 27.0 | 1.80 | 87.0 | no | no | 3.0 | |
| **4** | Male | 22.0 | 1.78 | 89.8 | no | no | 2.0 | |

In [14]:

```
df_raw.select_dtypes("object").nunique()
```

Out [14]:

| | 0 |
|---|---|
| **gender** | 2 |
| **overweight_familiar** | 2 |
| **eat_hc_food** | 2 |
| **snack** | 4 |
| **smoke** | 2 |
| **monitoring_calories** | 2 |
| **drink_alcohol** | 4 |
| **transportation_type** | 5 |
| **obesity_level** | 7 |

**dtype:** int64

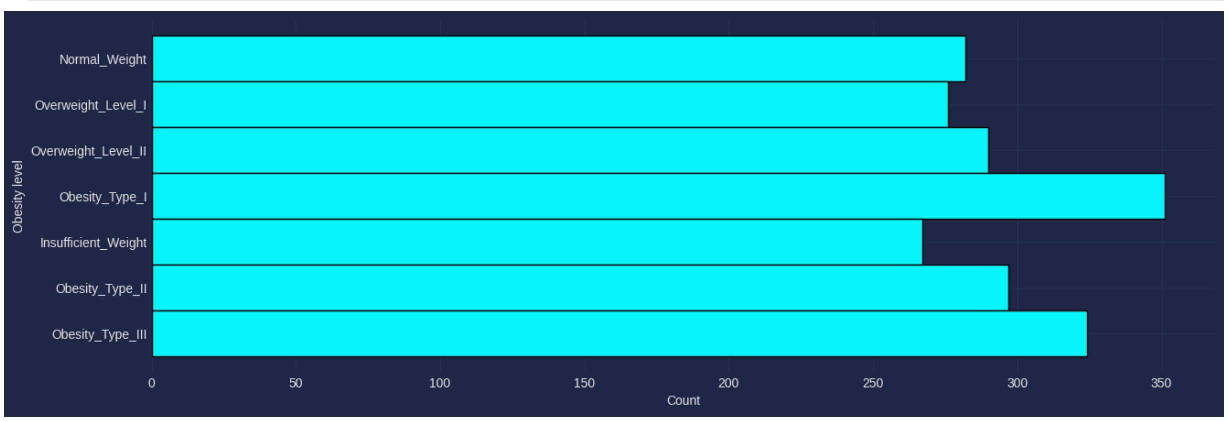
In [15]:
```
df = df_raw.copy()
```

# Data visualization

In [16]:
```
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import mplcyberpunk

plt.style.use("cyberpunk")
```

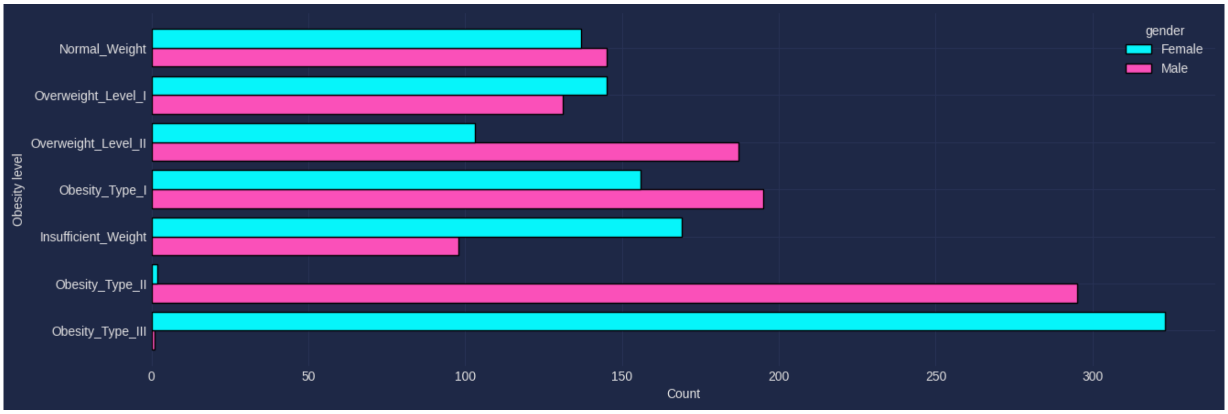
## Balance of the target variable

In [17]:
```
plt.figure(figsize=(15, 5))
sns.histplot(df, y="obesity_level", alpha=1)
plt.ylabel("Obesity level");
```



## The relationship between obesity levels and the gender

In [18]:
```
plt.figure(figsize=(15, 5))
sns.histplot(df, y="obesity_level", hue="gender", multiple="dodge", shrink=0
plt.ylabel("Obesity level");
```




The balance in the gender feature:

In [19]:
```
df.gender.value_counts(normalize=True)
```

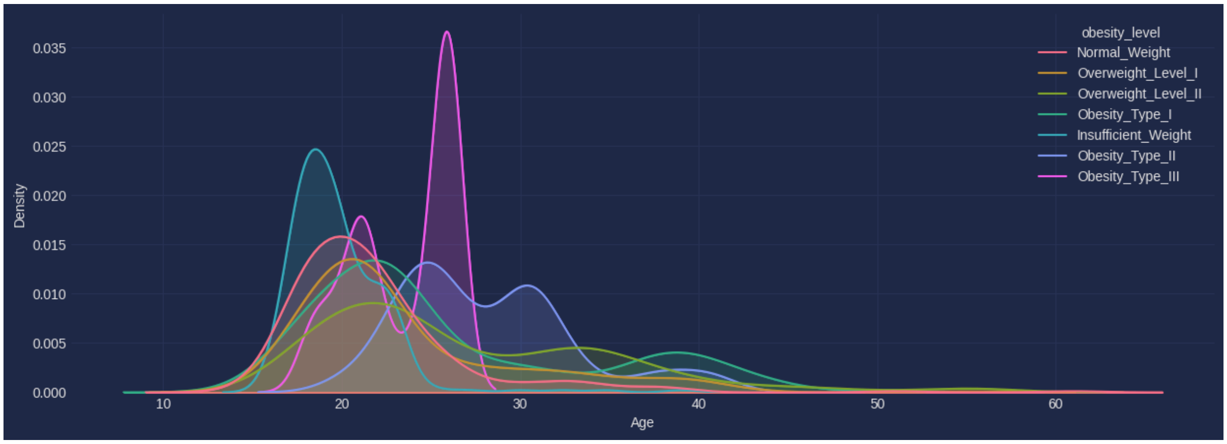
Out[19]:

| | proportion |
|---|---|
| **gender** | |
| **Male** | 0.504073 |
| **Female** | 0.495927 |

**dtype:** float64

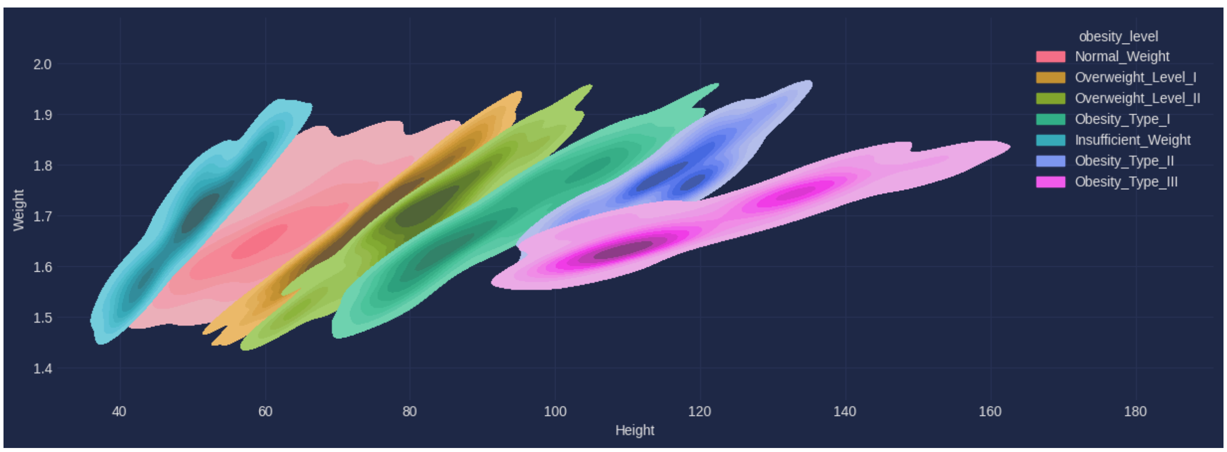
## The distribution of age in relation to obesity levels

In [20]:
```
plt.figure(figsize=(15, 5))
sns.kdeplot(df, x="age", hue="obesity_level", fill=True, alpha=.2)
sns.kdeplot(df, x="age", hue="obesity_level")
plt.xlabel("Age");
```



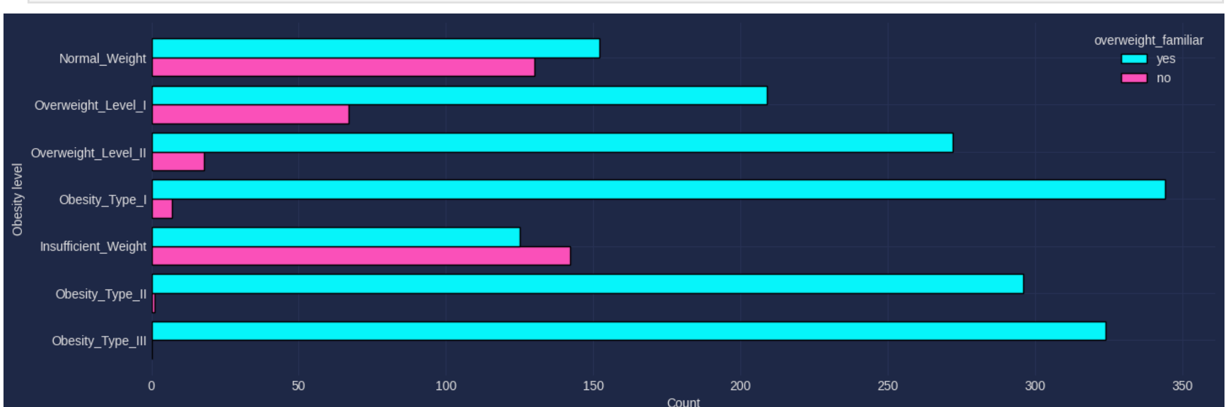

## Height and weight in relation to obesity levels

In [21]:
```
plt.figure(figsize=(15, 5))
sns.kdeplot(data=df, x="weight", y="height", hue="obesity_level", fill=True)
plt.xlabel("Height")
plt.ylabel("Weight");
```



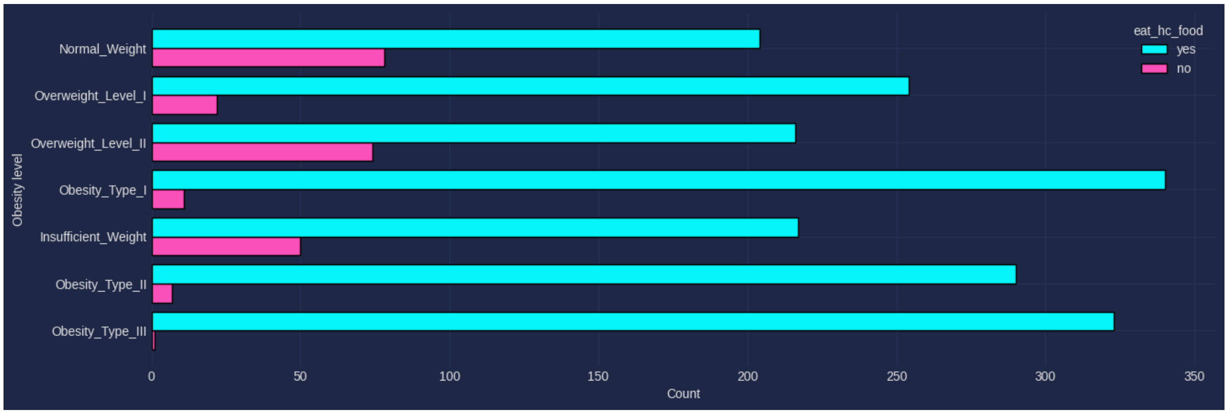

## The relationship between obesity levels and having overweight family members

In [22]:
```
plt.figure(figsize=(15, 5))
sns.histplot(df, y="obesity_level", hue="overweight_familiar", multiple="dod
plt.ylabel("Obesity level");
```



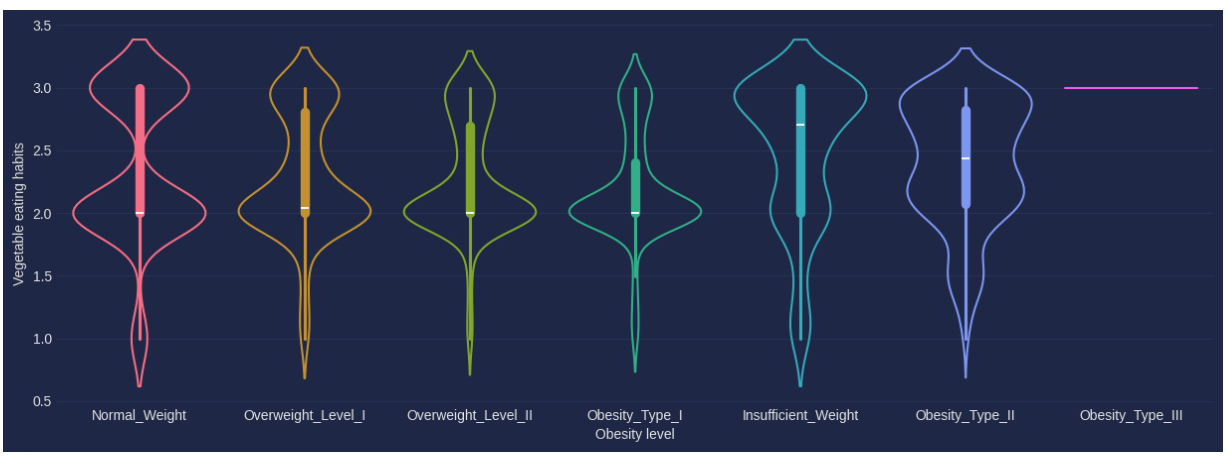
## The relationship between obesity levels and the consumption of high-calorie foods

In [23]:
```
plt.figure(figsize=(15, 5))
sns.histplot(df, y="obesity_level", hue="eat_hc_food", multiple="dodge", shr
plt.ylabel("Obesity level");
```




## The relationship between obesity levels and vegetable eating habits

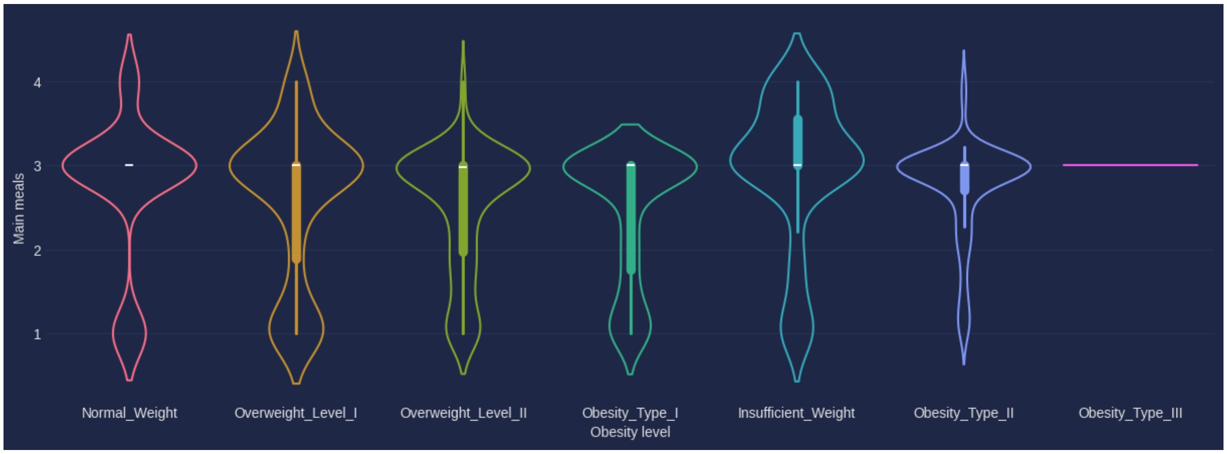
In [24]:
```
plt.figure(figsize=(15, 5))
sns.violinplot(data=df, x="obesity_level", y="eat_vegetables", hue="obesity_
plt.ylabel("Vegetable eating habits")
plt.xlabel("Obesity level");
```




## The relationship between obesity levels and the number of main meals

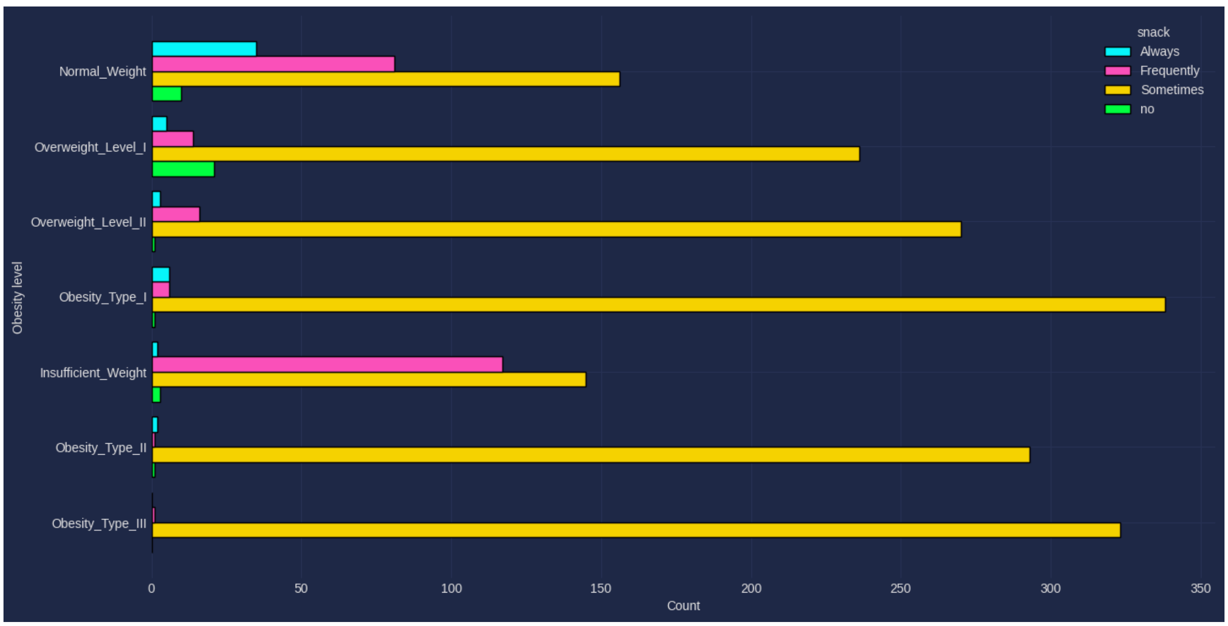
In [25]:
```
plt.figure(figsize=(15, 5))
sns.violinplot(data=df, x="obesity_level", y="main_meals", hue="obesity_leve
plt.ylabel("Main meals")
plt.xlabel("Obesity level");
```

Loading [MathJax]/extensions/Safe.js




## The relationship between obesity levels and snack eating habits

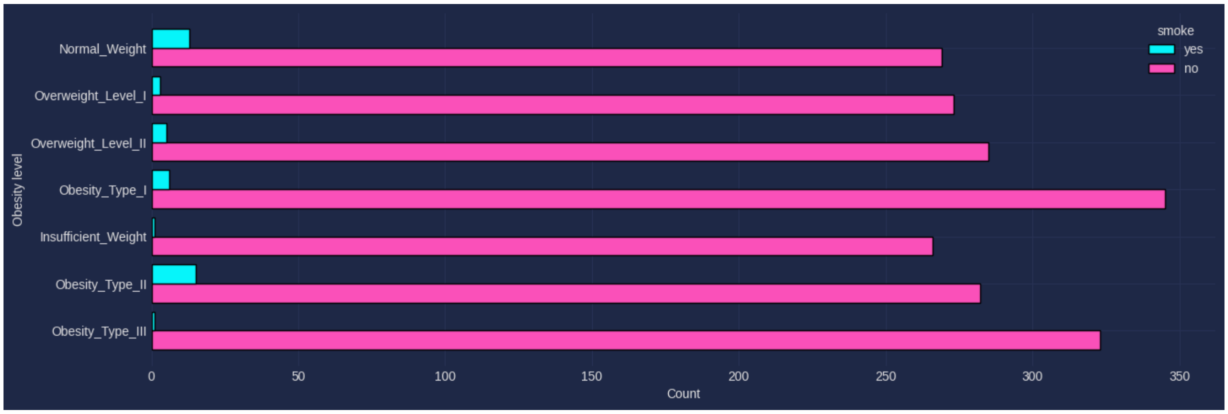
In [26]:
```
plt.figure(figsize=(15, 8))
sns.histplot(df, y="obesity_level", hue="snack", multiple="dodge", shrink=0.
plt.ylabel("Obesity level");
```




## The relationship between obesity levels and smoking habits

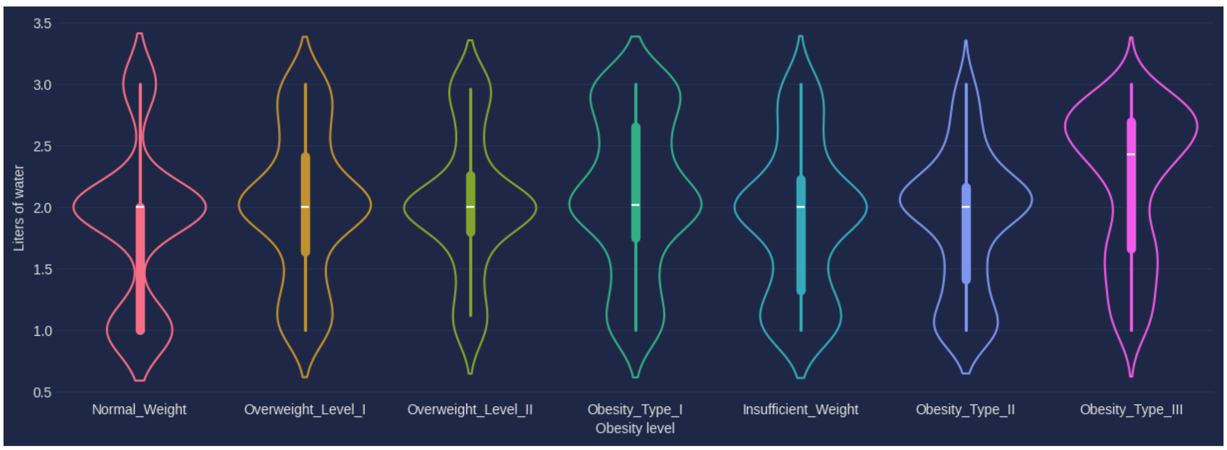
In [27]:
```
plt.figure(figsize=(15, 5))
sns.histplot(df, y="obesity_level", hue="smoke", multiple="dodge", shrink=0.
plt.ylabel("Obesity level");
```

Loading [MathJax]/extensions/Safe.js



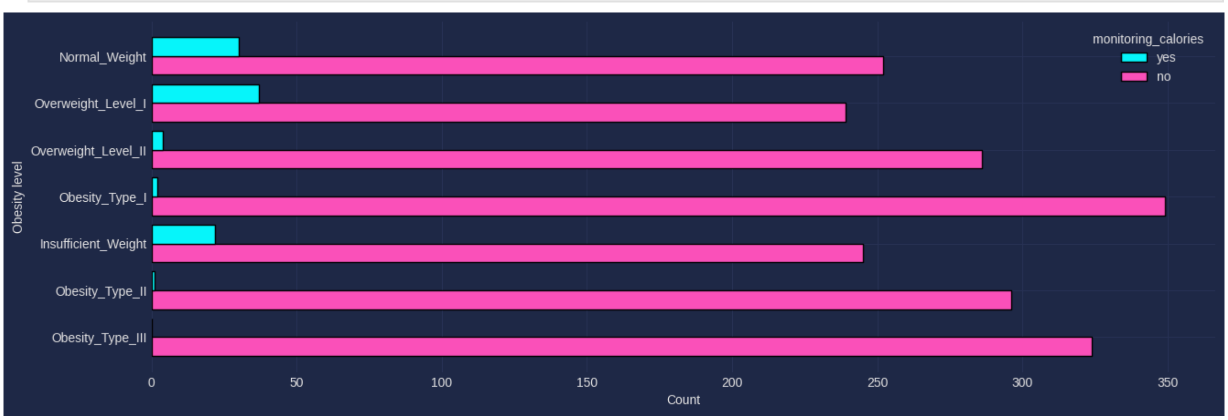

## The relationship between obesity levels and water consumption

In [28]:
```
plt.figure(figsize=(15, 5))
sns.violinplot(data=df, x="obesity_level", y="drink_water", hue="obesity_lev
plt.ylabel("Liters of water")
plt.xlabel("Obesity level");
```




## Calorie monitoring habits in relation to obesity levels

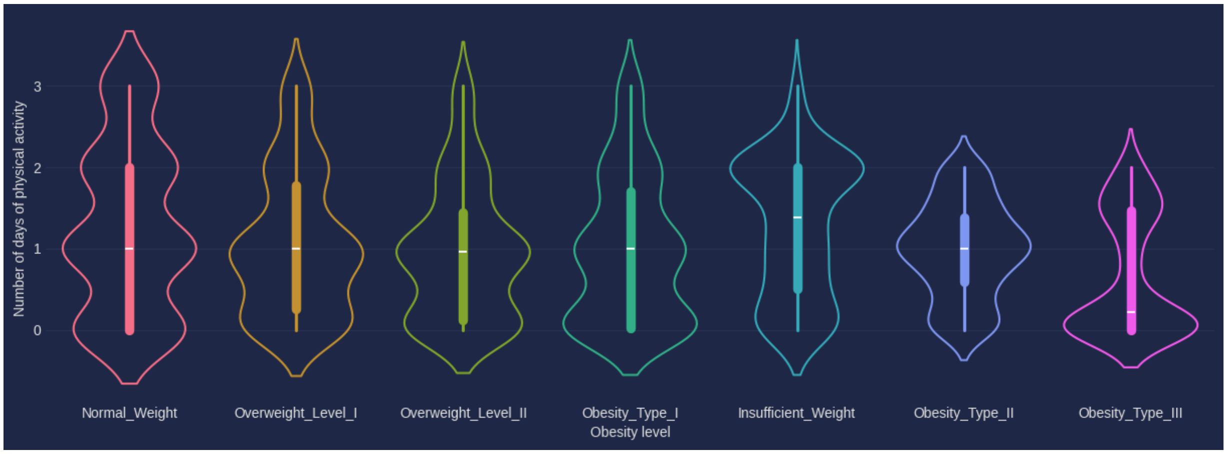
In [29]:
```
plt.figure(figsize=(15, 5))
sns.histplot(df, y="obesity_level", hue="monitoring_calories", multiple="dod
plt.ylabel("Obesity level");
```




## Physical activity in relation to obesity levels

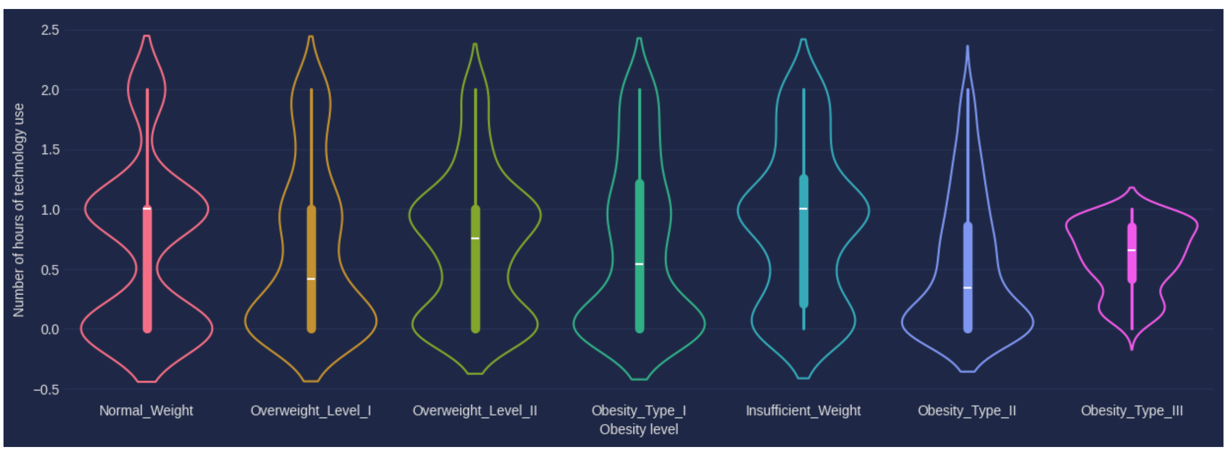
Loading [MathJax]/extensions/Safe.js

In [30]:
```
plt.figure(figsize=(15, 5))
sns.violinplot(data=df, x="obesity_level", y="physical_activity", hue="obesi
plt.ylabel("Number of days of physical activity")
plt.xlabel("Obesity level");
```



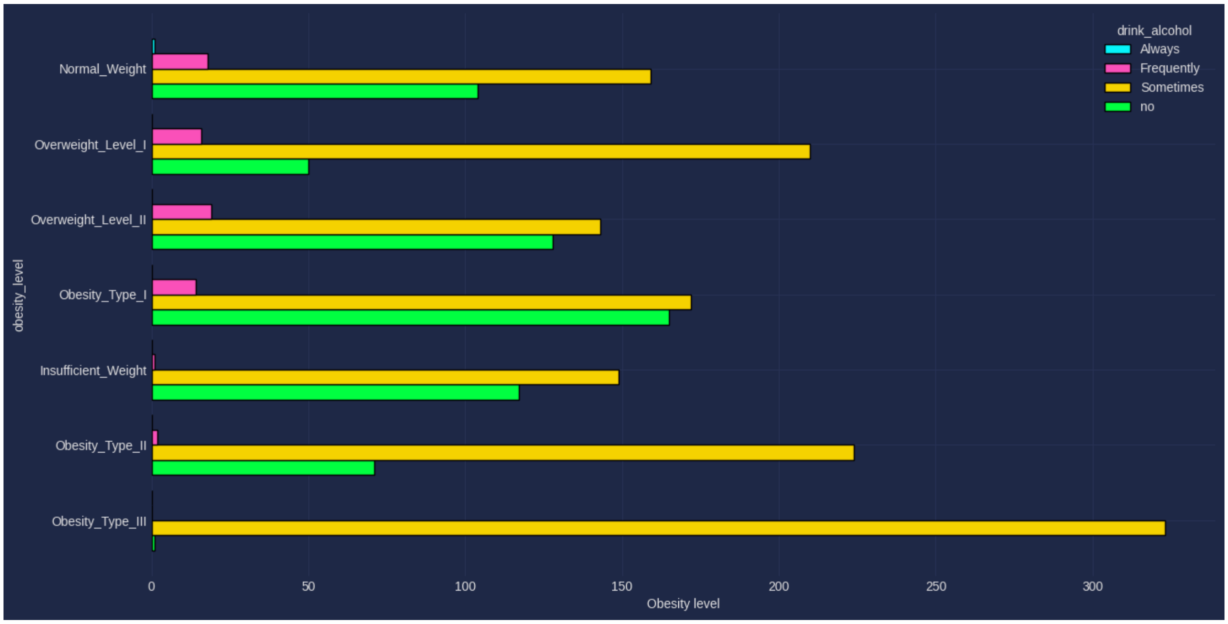

## Technology usage habits in relation to obesity levels

In [31]:
```
plt.figure(figsize=(15, 5))
sns.violinplot(data=df, x="obesity_level", y="use_of_technology", hue="obesi
plt.ylabel("Number of hours of technology use")
plt.xlabel("Obesity level");
```



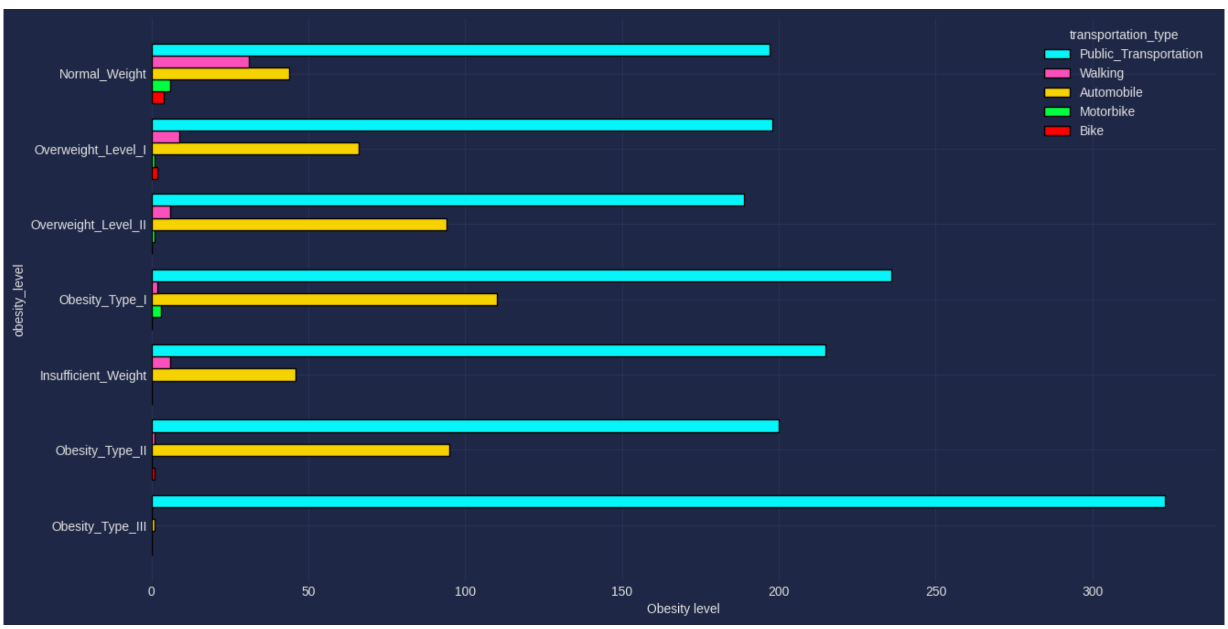

## Alcohol drinking habits in relation to obesity levels

In [32]:
```
plt.figure(figsize=(15, 8))
sns.histplot(df, y="obesity_level", hue="drink_alcohol", multiple="dodge", s
plt.xlabel("Obesity level");
```




## Obesity levels according to the type of transportation used

In [33]:
```
plt.figure(figsize=(15, 8))
sns.histplot(df, y="obesity_level", hue="transportation_type", multiple="dod
plt.xlabel("Obesity level");
```




# Splitting the dataset

## Setting seeds for reproducibility

In [34]:
```
import random
import os
import keras
```

Loading [MathJax]/extensions/Safe.js

```
seed_value = 42
os.environ['PYTHONHASHSEED'] = str(seed_value)
random.seed(seed_value)
np.random.seed(seed_value)
```

## Setting the validation framework

In [35]:
```
from sklearn.model_selection import train_test_split

df_full_train, df_test = train_test_split(df, test_size=0.15, random_state=s
df_train, df_val = train_test_split(df_full_train, test_size=0.15, random_st
len(df_train), len(df_test), len(df_val)
```

Out[35]:
```
(1507, 314, 266)
```

## Removing specific columns and separating features from the target column

According to the dataset's article, the data was labeled using the following equation:

$Mass\ body\ index = \frac{weight}{{height}^2}$

Therefore, we need to delete at least a feature into previous equation.

In [36]:
```
cols_drop = ["obesity_level", "weight"]
X_full_train, y_full_train = df_full_train.drop(cols_drop, axis=1), df_full_
X_train, y_train =  df_train.drop(cols_drop, axis=1), df_train["obesity_leve
X_val, y_val =  df_val.drop(cols_drop, axis=1), df_val["obesity_level"]
X_test, y_test =  df_test.drop(cols_drop, axis=1), df_test["obesity_level"]
```

# Standardization

In [37]:
```
from sklearn.preprocessing import StandardScaler

numeric_cols = X_train.select_dtypes(exclude=["object"]).columns

ss = StandardScaler().set_output(transform="pandas")

ss.fit(X_train[numeric_cols])
ss.transform(X_train[numeric_cols])
```

Out [37]:

| | age | height | eat_vegetables | main_meals | drink_water | physical_activity | us |
|---|---|---|---|---|---|---|---|
| **2071** | -0.859706 | 0.482798 | 1.091725 | 0.378982 | 0.667080 | -0.024754 | |
| **190** | -0.679409 | -1.084919 | -0.803418 | 0.378982 | -0.005343 | -0.007330 | |
| **1361** | -0.996350 | -0.153529 | 0.593585 | 0.378982 | -0.009926 | 0.410323 | |
| **2109** | 0.011829 | 0.409630 | 1.091725 | 0.378982 | 1.387169 | 0.157412 | |
| **325** | -0.520938 | -1.620791 | -0.803418 | -2.235837 | -1.639098 | -0.007330 | |
| **...** | ... | ... | ... | ... | ... | ... | |
| **1357** | -0.996350 | 0.891401 | -0.803418 | -0.306828 | 1.066065 | -0.007330 | |
| **1013** | 4.906080 | 0.729214 | -0.803418 | 0.378982 | -0.005343 | -0.007330 | |
| **1971** | -0.790813 | 1.243673 | 1.091725 | 0.378982 | 1.137754 | 0.609467 | |
| **1265** | -1.048027 | 0.399074 | -0.803418 | 0.378982 | -0.005343 | -1.191614 | |
| **1881** | 0.029785 | -0.069697 | 1.091725 | 0.378982 | 1.177656 | -0.792753 | |

1507 rows × 7 columns

# One-hot encoding of features

In [38]:

```
from sklearn.feature_extraction import DictVectorizer

dict_X_full_train = X_full_train.to_dict("records")
dict_X_train = X_train.to_dict("records")
dict_X_val = X_val.to_dict("records")
dict_X_test = X_test.to_dict("records")

dv = DictVectorizer(sparse=False).set_output(transform="pandas")
dv.fit(dict_X_train)

dv.transform(dict_X_train).head()
```

Out [38]:

| | age | drink_alcohol=Always | drink_alcohol=Frequently | drink_alcohol=Sometimes | ( |
|---|---|---|---|---|---|
| **0** | 18.862264 | 0.0 | 0.0 | 1.0 | |
| **1** | 20.000000 | 0.0 | 0.0 | 1.0 | |
| **2** | 18.000000 | 0.0 | 0.0 | 1.0 | |
| **3** | 24.361936 | 0.0 | 0.0 | 1.0 | |
| **4** | 21.000000 | 0.0 | 0.0 | 1.0 | |

5 rows × 30 columns

# Creating a Pipeline

In [39]:
```
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.pipeline import Pipeline

class MyStandardScaler(BaseEstimator, TransformerMixin):
    def __init__(self, numeric_cols):
        self.ss = StandardScaler().set_output(transform="pandas")
        self.numeric_cols = numeric_cols
        return

    def fit(self, X):
        self.ss.fit(X[self.numeric_cols])
        return self

    def transform(self, X):
        X[self.numeric_cols] = self.ss.transform(X[self.numeric_cols])
        return X.to_dict("records")

numeric_cols = X_train.select_dtypes(exclude=["object"]).columns
pipe = Pipeline([('ss', MyStandardScaler(numeric_cols=numeric_cols)), ('dv',

X_train = pipe.fit_transform(X_train)
X_full_train = pipe.transform(X_full_train)
X_val = pipe.transform(X_val)
X_test = pipe.transform(X_test)
```

# Label encoding

In [40]:
```
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()
le.fit(y_train)
y_full_train = le.transform(y_full_train)
y_train = le.transform(y_train)
y_val = le.transform(y_val)
y_test = le.transform(y_test)
```

# Computing weigths for classes and samples

In [41]:
```
from sklearn.utils.class_weight import compute_sample_weight, compute_class_

class_full_weight = compute_class_weight(class_weight="balanced", classes=np
class_full_weight = dict(zip(np.unique(y_full_train), class_full_weight))

class_weight = compute_class_weight(class_weight="balanced", classes=np.uniq
class_weight = dict(zip(np.unique(y_train), class_weight))

ll_weights = compute_sample_weight(
```

```
    class_weight=class_full_weight,
    y=y_full_train
)

sample_weights = compute_sample_weight(
    class_weight=class_weight,
    y=y_train
)
```

# Training the models

## Logistic Regression

In [42]:
```
from sklearn.metrics import roc_auc_score
```

In [ ]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV

parameters = {"max_iter":[200, 300, 400, 500],
              "C":[20, 15, 10],
              "class_weight":["balanced"],
              "solver":["lbfgs", "newton-cg", "sag", "saga"]}

lr = LogisticRegression(random_state=seed_value)
gs_lr = GridSearchCV(lr, param_grid=parameters, n_jobs=-1, cv=5, scoring="ro
gs_lr.fit(X_full_train, y_full_train, sample_weight=sample_full_weights)
```

```
/usr/local/lib/python3.10/dist-packages/numpy/ma/core.py:2820: RuntimeWarnin
g: invalid value encountered in cast
  _data = np.array(data, dtype=dtype, copy=copy,
```

Out[ ]:
```
▸ GridSearchCV
▸ best_estimator_: LogisticRegression
    ▸ LogisticRegression
```

In [ ]:
```
y_pred_test_lr = gs_lr.predict_proba(X_test)
roc_auc_score(y_test, y_pred_test_lr, multi_class="ovr")
```

Out[ ]:
```
0.889393203661662
```

## Decision Tree

In [ ]:
```
from sklearn.tree import DecisionTreeClassifier
parameters = {"criterion":["gini", "entropy", "log_loss"],
              "max_depth":[5, 6, 7, 8, 9],
              "max_leaf_nodes":[30, 40, 50, 60],
              "min_samples_split":[30, 40],
```

```
                 "class_weight":["balanced"],}

dt = DecisionTreeClassifier(random_state=seed_value)
gs_dt = GridSearchCV(dt, param_grid=parameters, n_jobs=-1, cv=5, scoring="ro
gs_dt.fit(X_full_train, y_full_train, sample_weight=sample_full_weights)
```

Out[ ]:

```
GridSearchCV
best_estimator_: DecisionTreeClassifier
DecisionTreeClassifier
```

In [ ]:

```
y_pred_test_dt = gs_dt.predict_proba(X_test)
roc_auc_score(y_test, y_pred_test_dt, multi_class="ovr")
```

Out[ ]: 0.9213027545513881

## Random Forest

In [ ]:

```
from sklearn.ensemble import RandomForestClassifier
parameters = {"criterion":["gini", "entropy", "log_loss"],
                 "n_estimators":[100, 200, 300, 400],
                 "max_depth":[5, 6, 7, 8, 9],
                 "class_weight":["balanced"],}

rf = RandomForestClassifier(random_state=seed_value)
gs_rf = GridSearchCV(rf, param_grid=parameters, n_jobs=-1, cv=5, scoring="ro
gs_rf.fit(X_full_train, y_full_train, sample_weight=sample_full_weights)
```

```
/usr/local/lib/python3.10/dist-packages/numpy/ma/core.py:2820: RuntimeWarnin
g: invalid value encountered in cast
  _data = np.array(data, dtype=dtype, copy=copy,
```

Out[ ]:

```
GridSearchCV
best_estimator_: RandomForestClassifier
RandomForestClassifier
```

In [ ]:

```
y_pred_test_rf = gs_rf.predict_proba(X_test)
roc_auc_score(y_test, y_pred_test_rf, multi_class="ovr")
```

Out[ ]: 0.9721096055019289

## Gradient Boosting with XGBoost

In [55]:

```
import xgboost as xgb

xgb.DMatrix(X_train, label=y_train, feature_names=X_train.columns.t
```

```python
dval = xgb.DMatrix(X_val, label=y_val, feature_names=X_val.columns.tolist())
dtest = xgb.DMatrix(X_test, label=y_test, feature_names=X_test.columns.tolis

xgb_params = {
    'eta': 0.1,
    'max_depth': 6,
    'gamma':0.0001,
    'min_child_weight': 1,
    'alpha':0.01,

    'objective': 'multi:softprob',
    'num_class':7,
    'nthread': 8,
    'eval_metric':'auc',
    'num_parallel_tree':5,

    'seed':seed_value,
    'verbosity': 1,
}

watchlist = [(dtrain, 'train'), (dval, 'val')]
xgb_clf = xgb.train(xgb_params, dtrain, num_boost_round=10000, early_stoppin
y_test_pred_xgb = xgb_clf.predict(dtest)
print(f"ROC:{roc_auc_score(y_test, y_test_pred_xgb, multi_class='ovr')}")
```

```
[0]	train-auc:0.96308	val-auc:0.88976
[200]	train-auc:1.00000	val-auc:0.97088
[400]	train-auc:1.00000	val-auc:0.97176
[600]	train-auc:1.00000	val-auc:0.97269
[800]	train-auc:1.00000	val-auc:0.97309
[942]	train-auc:1.00000	val-auc:0.97303
ROC:0.9806772041514711
```

# Gradient Boosting with CatBoost

According to this blog.

CatBoost operates on the principle of gradient boosting, where it builds the model in a stage-wise fashion. It starts with a simple model and incrementally improves it by adding new models that correct the errors made by the preceding ones.

CatBoost introduces several key innovations:

## Ordered Boosting

One of the core innovations of CatBoost is its ordered boosting mechanism. Traditional gradient boosting methods can suffer from prediction shift due to the overlap between the training data for the base models and the data used to calculate the gradients. CatBoost addresses this by introducing a random permutation of the dataset in each iteration and using only the data before each example in the permutation for training. This approach reduces overfitting and improves model robustness.

## Symmetric Trees

CatBoost builds balanced trees, also known as symmetric trees, as its base predictors. Unlike traditional gradient boosting methods that build trees leaf-wise or depth-wise, CatBoost's symmetric trees ensure that all leaf nodes at the same level share the same decision rule. This leads to faster execution and reduces the likelihood of overfitting.

In [53]:
```
from catboost import CatBoostClassifier

cbc = CatBoostClassifier(loss_function='MultiClass',
                         eval_metric='AUC',
                         iterations=5000,
                         depth=6,
                         classes_count=7,
                         class_weights=class_weight,
                         learning_rate=0.1,
                         od_type='Iter',
                         early_stopping_rounds=1000,
                         bootstrap_type='MVS',
                         sampling_frequency='PerTree',
                         random_seed=seed_value,
                         verbose=200)
cbc.fit(X_train, y_train, sample_weight=sample_weights, eval_set=(X_val, y_va

y_test_pred_cbc = cbc.predict_proba(X_test)
print(f"ROC:{roc_auc_score(y_test, y_test_pred_cbc, multi_class='ovr')}")
```

```
0:      test: 0.8701799 best: 0.8701799 (0)     total: 36.5ms   remaining: 6m
5s
200:    test: 0.9843814 best: 0.9843814 (200)   total: 6.27s    remaining: 5m
5s
400:    test: 0.9859230 best: 0.9860489 (384)   total: 9.02s    remaining: 3m
35s
600:    test: 0.9859990 best: 0.9862174 (434)   total: 11.7s    remaining: 3m
2s
800:    test: 0.9860843 best: 0.9865136 (706)   total: 14.4s    remaining: 2m
45s
1000:   test: 0.9861372 best: 0.9865136 (706)   total: 19s      remaining: 2m
51s
1200:   test: 0.9859280 best: 0.9865136 (706)   total: 21.9s    remaining: 2m
40s
1400:   test: 0.9860076 best: 0.9865136 (706)   total: 24.5s    remaining: 2m
30s
1600:   test: 0.9857638 best: 0.9865136 (706)   total: 27.2s    remaining: 2m
22s
Stopped by overfitting detector  (1000 iterations wait)

bestTest = 0.9865136358
bestIteration = 706

Shrink model to first 707 iterations.
ROC:0.9849353946292636
```

# Neural Networks

In [ ]:
```
import tensorflow as tf
from tensorflow.data import Dataset
from tensorflow.keras import Input, Model
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.losses import SparseCategoricalCrossentropy
from tensorflow.keras.optimizers import Adamax
from tensorflow.keras.metrics import SparseCategoricalAccuracy
```

In [ ]:
```
tf.keras.backend.clear_session()
keras.utils.set_random_seed(seed_value)

autotune = tf.data.AUTOTUNE

batch_size = 32

train_data = Dataset.from_tensor_slices((X_train.astype(float), y_train.astyp
val_data = Dataset.from_tensor_slices((X_val.astype(float), y_val.astype(float
test_data = Dataset.from_tensor_slices((X_test.astype(float), y_test.astype(fl

early = EarlyStopping(monitor='val_loss', patience=10)

inp = Input(shape=(30, ))
x = Dense(256, activation='relu')(inp)
x = Dense(128, activation='relu')(x)
x = Dropout(0.5)(x)
x = Dense(128, activation='relu')(x)
x = Dense(128, activation='relu')(x)
x = Dropout(0.5)(x)
x = Dense(7, activation='softmax')(x)

nn = Model(inputs=inp, outputs=x)
nn.compile(loss=SparseCategoricalCrossentropy(), optimizer=Adamax(learning_r
nn.summary()
```

**Model: "functional"**

| Layer (type) | Output Shape |
| --- | --- |
| input_layer (InputLayer) | (None, 30) |
| dense (Dense) | (None, 256) |
| dense_1 (Dense) | (None, 128) |
| dropout (Dropout) | (None, 128) |
| dense_2 (Dense) | (None, 128) |
| dense_3 (Dense) | (None, 128) |
| dropout_1 (Dropout) | (None, 128) |
| dense_4 (Dense) | (None, 7) |

**Total params:** 74,759 (292.03 KB)

**Trainable params:** 74,759 (292.03 KB)

**Non-trainable params:** 0 (0.00 B)

In [ ]:
```
epochs = 1000
history = nn.fit(train_data,
                epochs=epochs,
                verbose=1,
                validation_data=val_data,
                class_weight=class_weight,
                callbacks=[early]
                )
```

Loading [MathJax]/extensions/Safe.js

```
Epoch 1/1000
48/48 ━━━━━━━━━━━━━━━━━━━━ 3s 8ms/step - loss: 1.8327 - sparse_categorical_a
ccuracy: 0.2595 - val_loss: 1.2738 - val_sparse_categorical_accuracy: 0.4511
Epoch 2/1000
48/48 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 1.3165 - sparse_categorical_a
ccuracy: 0.4592 - val_loss: 1.0407 - val_sparse_categorical_accuracy: 0.6617
Epoch 3/1000
48/48 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 1.1631 - sparse_categorical_a
ccuracy: 0.5468 - val_loss: 0.9571 - val_sparse_categorical_accuracy: 0.6692
Epoch 4/1000
48/48 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 1.0944 - sparse_categorical_a
ccuracy: 0.5979 - val_loss: 0.9358 - val_sparse_categorical_accuracy: 0.6654
Epoch 5/1000
48/48 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 0.9901 - sparse_categorical_a
ccuracy: 0.6048 - val_loss: 0.9158 - val_sparse_categorical_accuracy: 0.6617
Epoch 6/1000
48/48 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 0.9382 - sparse_categorical_a
ccuracy: 0.6409 - val_loss: 0.8158 - val_sparse_categorical_accuracy: 0.6992
Epoch 7/1000
48/48 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 0.8868 - sparse_categorical_a
ccuracy: 0.6698 - val_loss: 0.8139 - val_sparse_categorical_accuracy: 0.7293
Epoch 8/1000
48/48 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 0.7843 - sparse_categorical_a
ccuracy: 0.7220 - val_loss: 0.7592 - val_sparse_categorical_accuracy: 0.7556
Epoch 9/1000
48/48 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 0.7551 - sparse_categorical_a
ccuracy: 0.7096 - val_loss: 0.7370 - val_sparse_categorical_accuracy: 0.7406
Epoch 10/1000
48/48 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 0.7274 - sparse_categorical_a
ccuracy: 0.7371 - val_loss: 0.7488 - val_sparse_categorical_accuracy: 0.7519
Epoch 11/1000
48/48 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 0.6668 - sparse_categorical_a
ccuracy: 0.7442 - val_loss: 0.7169 - val_sparse_categorical_accuracy: 0.7744
Epoch 12/1000
48/48 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 0.6312 - sparse_categorical_a
ccuracy: 0.7730 - val_loss: 0.7175 - val_sparse_categorical_accuracy: 0.7895
Epoch 13/1000
48/48 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 0.6090 - sparse_categorical_a
ccuracy: 0.7854 - val_loss: 0.7165 - val_sparse_categorical_accuracy: 0.7782
Epoch 14/1000
48/48 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 0.5573 - sparse_categorical_a
ccuracy: 0.8115 - val_loss: 0.7084 - val_sparse_categorical_accuracy: 0.8045
Epoch 15/1000
48/48 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 0.5673 - sparse_categorical_a
ccuracy: 0.7960 - val_loss: 0.7481 - val_sparse_categorical_accuracy: 0.7895
Epoch 16/1000
48/48 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 0.5021 - sparse_categorical_a
ccuracy: 0.8189 - val_loss: 0.7435 - val_sparse_categorical_accuracy: 0.7632
Epoch 17/1000
48/48 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 0.4850 - sparse_categorical_a
ccuracy: 0.8306 - val_loss: 0.7140 - val_sparse_categorical_accuracy: 0.8083
Epoch 18/1000
48/48 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 0.4586 - sparse_categorical_a
ccuracy: 0.8487 - val_loss: 0.7559 - val_sparse_categorical_accuracy: 0.8045
Epoch 19/1000
48/48 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 0.4490 - sparse_categorical_a
```

```
ccuracy: 0.8545 - val_loss: 0.8031 - val_sparse_categorical_accuracy: 0.8083
Epoch 20/1000
48/48 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 0.4681 - sparse_categorical_a
ccuracy: 0.8568 - val_loss: 0.7520 - val_sparse_categorical_accuracy: 0.8083
Epoch 21/1000
48/48 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 0.4161 - sparse_categorical_a
ccuracy: 0.8657 - val_loss: 0.7829 - val_sparse_categorical_accuracy: 0.7970
Epoch 22/1000
48/48 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 0.4019 - sparse_categorical_a
ccuracy: 0.8610 - val_loss: 0.8005 - val_sparse_categorical_accuracy: 0.7932
Epoch 23/1000
48/48 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 0.4139 - sparse_categorical_a
ccuracy: 0.8593 - val_loss: 0.8020 - val_sparse_categorical_accuracy: 0.7932
Epoch 24/1000
48/48 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 0.3586 - sparse_categorical_a
ccuracy: 0.8893 - val_loss: 0.8445 - val_sparse_categorical_accuracy: 0.7970
```

In [ ]:

```
y_pred_test_nn = nn.predict(test_data, batch_size=batch_size)

roc_auc_score(y_test, y_pred_test_nn, multi_class='ovr')
```

```
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step
```

Out[ ]: 0.9612609703428328

# Comparison of models

In [ ]:

```
from sklearn.metrics import f1_score, accuracy_score, precision_score, recal

def get_scores(y_true, y_pred, y_pred_proba):
    return {'AUC_ROC':roc_auc_score(y_true, y_pred_proba, multi_class='ovr')
            'F1_Score':f1_score(y_true, y_pred, average='weighted'),
            'Accuracy':accuracy_score(y_true, y_pred),
            'Precision':precision_score(y_true, y_pred, average='weighted'),
            'Recall':recall_score(y_true, y_pred, average='weighted')}
scores = []
for model in [gs_lr, gs_dt, gs_rf, xgb_clf, cbc, nn]:
    if model == xgb_clf:
        y_test_pred = np.argmax(model.predict(dtest), axis=1)
        y_test_pred_proba = model.predict(dtest)
    elif model == nn:
        y_test_pred = np.argmax(model.predict(test_data), axis=1)
        y_test_pred_proba = model.predict(test_data)
    else:
        y_test_pred = model.predict(X_test)
        y_test_pred_proba = model.predict_proba(X_test)
    scores.append(get_scores(y_test, y_test_pred, y_test_pred_proba).values(

comparison = pd.DataFrame(data=scores, index=["Logistic Regression", "Decisi
comparison.style.highlight_max(color = 'green', axis = 0).highlight_min(colo
```

```
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
```

Out[ ]:

| | AUC_ROC | F1_Score | Accuracy | Precision | Recall |
|---|---|---|---|---|---|
| **Logistic Regression** | 0.889393 | 0.621080 | 0.636943 | 0.631468 | 0.636943 |
| **Decision Tree** | 0.921303 | 0.682760 | 0.681529 | 0.691372 | 0.681529 |
| **Random Forest** | 0.972110 | 0.838787 | 0.840764 | 0.842998 | 0.840764 |
| **XGBoost** | 0.980677 | 0.866181 | 0.866242 | 0.867547 | 0.866242 |
| **CatBoost** | 0.984935 | 0.891044 | 0.891720 | 0.891502 | 0.891720 |
| **Neural Network** | 0.961261 | 0.831495 | 0.831210 | 0.835182 | 0.831210 |

# Confusion matrix

In [ ]:

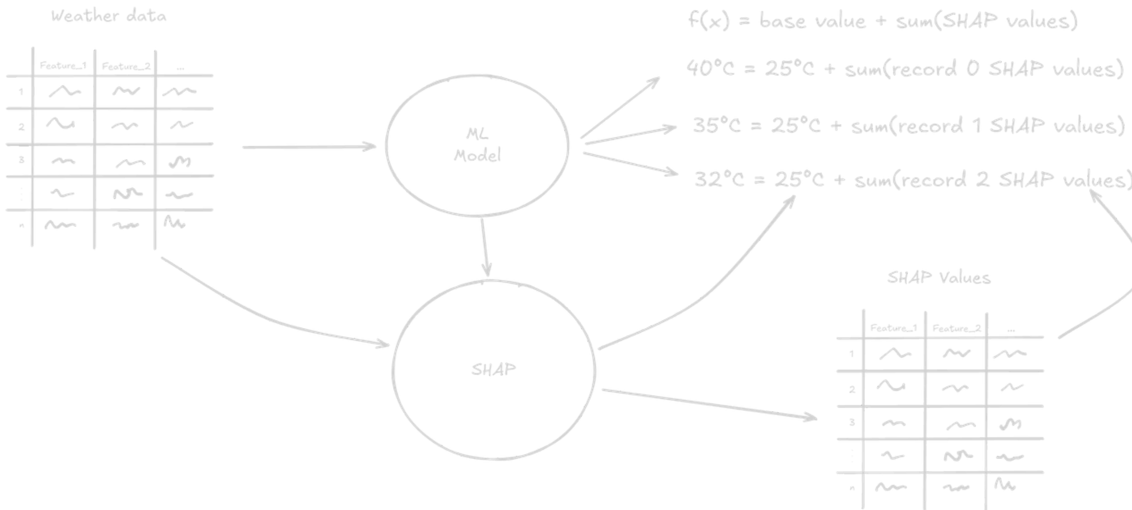
```
from sklearn.metrics import confusion_matrix

def my_cm(y_true, y_pred, title):
    cm_val = confusion_matrix(y_true, y_pred)
    cm_pgs = np.round(confusion_matrix(y_true, y_pred, normalize='true')*100

    formatted_text = (np.asarray([f"{pgs}%\n({val})" for val, pgs in zip(cm_

    sns.heatmap(cm_pgs, annot=formatted_text, fmt='', cmap='BuPu', yticklabe
    plt.title(title)
    plt.xlabel("Prediction")
    plt.ylabel("Expected")

    plt.subplots_adjust(hspace=0.5)
    return

y_test_pred_cbc = cbc.predict(X_test)

plt.figure(figsize=(10, 10))
my_cm(y_test, y_test_pred_cbc, title="Catboost")
```


## Feature importance analysis with SHAP

SHAP is a method that explains how individual predictions are made by a machine learning model. SHAP deconstructs a prediction into a sum of contributions from each of the model's input variables. For each instance in the data, the contribution from each input variable towards the model's prediction will vary depending on the values of the variables for that particular instance.

A machine learning model's prediction, f(x), can be represented as the sum of its computed SHAP values, plus a fixed base value, such that:

$f(x)=base\space value + SUM(SHAP\space values)$




In [ ]:
```python
import shap

shap.initjs()
```

In [ ]:
```python
explainer = shap.TreeExplainer(cbc)
shap_values = explainer(X_train, y_train)
```

In [ ]:
```python
plt.style.use("seaborn-v0_8")
for i in range(6):
    shap.plots.beeswarm(shap_values[..., i], show=False)
    plt.xlabel(f"SHAP value for {le.classes_[i]} class")
    plt.show()
```


height
drink_water
age
eat_vegetables
use_of_technology
physical_activity
main_meals
overweight_familiar=yes
overweight_familiar=no
Sum of 21 other features
−2
−1
0
1
2
3
SHAP value for Normal_Weight class
High
Feature value
Low


eat_vegetables
age
height
drink_water
main_meals
use_of_technology
overweight_familiar=yes
physical_activity
overweight_familiar=no
Sum of 21 other features
−2
−1
0
1
2
SHAP value for Obesity_Type_I class
High
Feature value
Low