Imports

In [1]:
```
import itertools
import os
import time
import warnings

import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns

from imblearn.combine import SMOTEENN
from imblearn.over_sampling import ADASYN, KMeansSMOTE, RandomOverSampler,
from imblearn.pipeline import make_pipeline

from sklearn.compose import ColumnTransformer
from sklearn.ensemble import BaggingClassifier, ExtraTreesClassifier, Rando
from sklearn.impute import SimpleImputer
from sklearn.linear_model import LinearRegression, LogisticRegression
from sklearn.metrics import accuracy_score, f1_score, plot_confusion_matrix
from sklearn.model_selection import GridSearchCV, cross_val_score, cross_va
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier, KNeighborsRegressor
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import LabelEncoder, Normalizer, OneHotEncoder,
from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor
from sklearn import set_config

import xgboost as xgb
from xgboost import XGBClassifier, XGBRegressor

from statsmodels.stats.outliers_influence import variance_inflation_factor

# Ignore warnings
warnings.filterwarnings("ignore")

# Set global config for scikit-learn
set_config(display='diagram')
```

EDA

Import csv and read into Pandas Dataframe

In [2]:
```
df = pd.read_csv("data/ObesityDataSet_raw_and_data_sinthetic.csv")
display(df.head())
display(df.info())
display(df.describe())
```

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2111 entries, 0 to 2110
Data columns (total 17 columns):
 #   Column                          Non-Null Count  Dtype  
---  ------                          --------------  -----  
 0   Gender                          2111 non-null   object 
 1   Age                             2111 non-null   float64
 2   Height                          2111 non-null   float64
 3   Weight                          2111 non-null   float64
 4   family_history_with_overweight  2111 non-null   object 
 5   FAVC                            2111 non-null   object 
 6   FCVC                            2111 non-null   float64
 7   NCP                             2111 non-null   float64
 8   CAEC                            2111 non-null   object 
 9   SMOKE                           2111 non-null   object 
 10  CH2O                            2111 non-null   float64
 11  SCC                             2111 non-null   object 
 12  FAF                             2111 non-null   float64
 13  TUE                             2111 non-null   float64
 14  CALC                            2111 non-null   object 
 15  MTRANS                          2111 non-null   object 
 16  NObeyesdad                      2111 non-null   object 
dtypes: float64(8), object(9)
memory usage: 280.5+ KB
```

```
None
```

	Age	Height	Weight	FCVC	NCP	CH2O	FAF
count	2111.000000	2111.000000	2111.000000	2111.000000	2111.000000	2111.000000	2111.000000
mean	24.312600	1.701677	86.586058	2.419043	2.685628	2.008011	1.010298
std	6.345968	0.093305	26.191172	0.533927	0.778039	0.612953	0.850592
min	14.000000	1.450000	39.000000	1.000000	1.000000	1.000000	0.000000
25%	19.947192	1.630000	65.473343	2.000000	2.658738	1.584812	0.124505
50%	22.777890	1.700499	83.000000	2.385502	3.000000	2.000000	1.000000
75%	26.000000	1.768464	107.430682	3.000000	3.000000	2.477420	1.666678
max	61.000000	1.980000	173.000000	3.000000	4.000000	3.000000	3.000000

Check for NAN values

In [3]:
```
df.isna().sum().sum()
```

Out[3]: 0

Check correlation

In [4]:
```
df.corr()
```

Out[4]:

	Age	Height	Weight	FCVC	NCP	CH2O	FAF	TUE
Age	1.000000	-0.025958	0.202560	0.016291	-0.043944	-0.045304	-0.144938	-0.296931
Height	-0.025958	1.000000	0.463136	-0.038121	0.243672	0.213376	0.294709	0.051912
Weight	0.202560	0.463136	1.000000	0.216125	0.107469	0.200575	-0.051436	-0.071561
FCVC	0.016291	-0.038121	0.216125	1.000000	0.042216	0.068461	0.019939	-0.101135
NCP	-0.043944	0.243672	0.107469	0.042216	1.000000	0.057088	0.129504	0.036326
CH2O	-0.045304	0.213376	0.200575	0.068461	0.057088	1.000000	0.167236	0.011965
FAF	-0.144938	0.294709	-0.051436	0.019939	0.129504	0.167236	1.000000	0.058562
TUE	-0.296931	0.051912	-0.071561	-0.101135	0.036326	0.011965	0.058562	1.000000

Check value counts per column

In [5]:
```
[display(df[i].value_counts()) for i in df.columns]
```

```
Male      1068
Female    1043
Name: Gender, dtype: int64

18.000000    128
26.000000    101
21.000000     96
23.000000     89
19.000000     59
            ...
19.314964      1
21.900120      1
23.421726      1
18.312665      1
61.000000      1
Name: Age, Length: 1402, dtype: int64

1.700000    60
1.650000    50
1.600000    43
```

Data Preparation

Round synthetic data to corrrespond to questionaire

In [6]:
```
# data came with floats where there should be int responses
cols_to_round= ["FCVC", "NCP", "CH2O", "FAF", "TUE", "Age"]

# quick function to correct this
def column_rounder(df, cols):
    for col in cols:
        df[col] = df[col].round(0)
    return df

df[cols_to_round] = column_rounder(df[cols_to_round], cols_to_round)
df.head()
```

Out[6]:

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

Manually encode that None response would be a 0

In [7]:
```
# encoding FCVC to better understand the value
df.FCVC = [x-1 for x in df.FCVC]
df.FCVC.value_counts()
```

Out[7]:
```
1.0    1013
2.0     996
0.0     102
Name: FCVC, dtype: int64
```

Manual encode weight classes in order of severity

In [8]:
```
target_categories = {
'Insufficient_Weight':0,
 'Normal_Weight':1,
 'Overweight_Level_I':2,
 'Overweight_Level_II':3,
 'Obesity_Type_I':4,
 'Obesity_Type_II':5,
 'Obesity_Type_III':6
 }

df.NObeyesdad = [target_categories[key] for key in df.NObeyesdad]
df.head()
```

Out[8]:

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

Label encode relevant columns

In [9]:
```
cols_to_le = ["Gender", "family_history_with_overweight", "FAVC", "SMOKE",
for i in cols_to_le:
    le = LabelEncoder()
    df[i] = le.fit_transform(df[i])
df.head()
```

Out[9]:

	Gender	Age	Height	Weight	family_history_with_overweight	FAVC	FCVC	NCP	CAEC	SM
0	0	21.0	1.62	64.0	1	0	1.0	3.0	Sometimes	
1	0	21.0	1.52	56.0	1	0	2.0	3.0	Sometimes	
2	1	23.0	1.80	77.0	1	0	1.0	3.0	Sometimes	
3	1	27.0	1.80	87.0	0	0	2.0	3.0	Sometimes	
4	1	22.0	1.78	89.8	0	0	1.0	1.0	Sometimes	

Encoding into a seperate df to show a heatmap

In [10]:

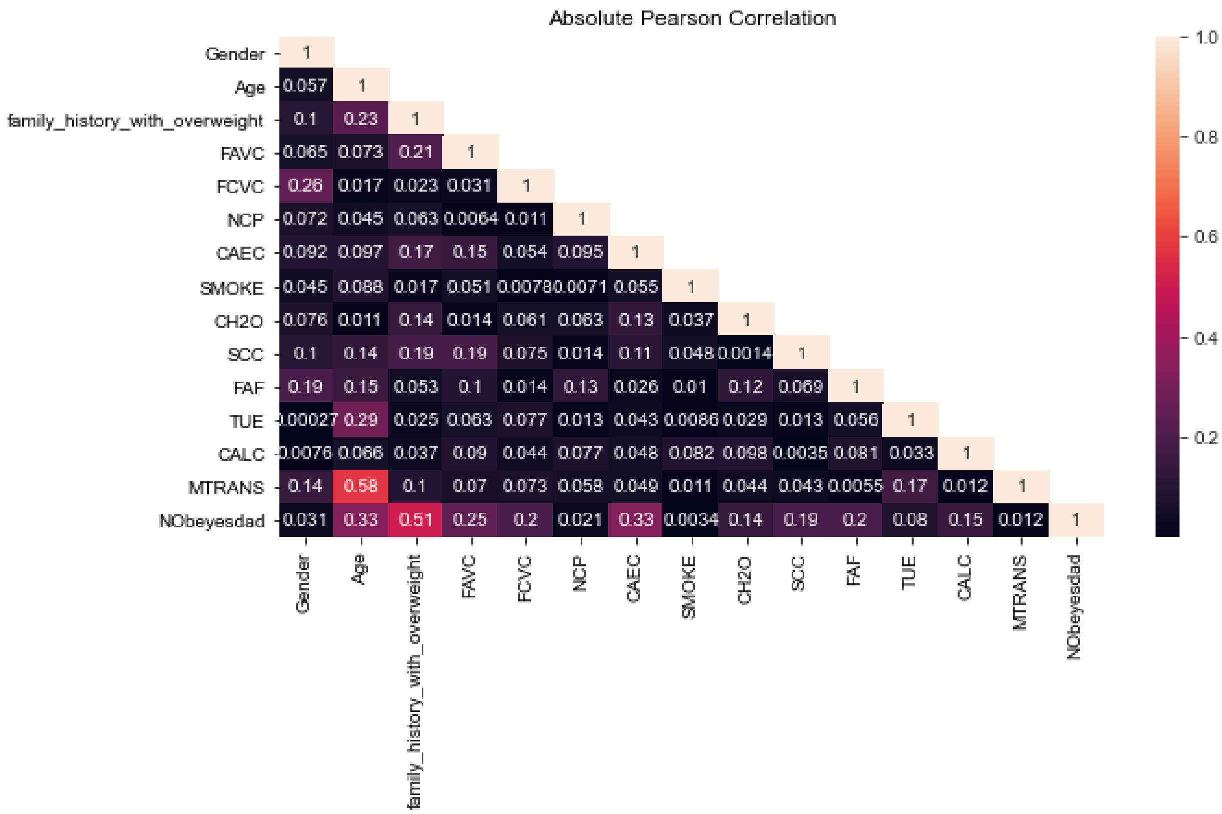
```
demo_df = df
cols_to_ord = ["CAEC", "CALC"]
demo_df = demo_df.drop(columns=["Height", "Weight"])

ord_enc = OrdinalEncoder(categories=[["no", "Sometimes", "Frequently", "Alw
                                      ["no", "Sometimes", "Frequently",

demo_df[cols_to_ord] = ord_enc.fit_transform(demo_df[cols_to_ord])

demo_df["Age"] = [np.log(x) for x in demo_df["Age"]]
demo_df
```

Out[10]:

	Gender	Age	family_history_with_overweight	FAVC	FCVC	NCP	CAEC	SMOKE	CH2O
0	0	3.044522	1	0	1.0	3.0	1.0	0	2.0
1	0	3.044522	1	0	2.0	3.0	1.0	1	3.0
2	1	3.135494	1	0	1.0	3.0	1.0	0	2.0
3	1	3.295837	0	0	2.0	3.0	1.0	0	2.0
4	1	3.091042	0	0	1.0	1.0	1.0	0	2.0
...	...	...	...	...	...	...	...	...	...
2106	0	3.044522	1	1	2.0	3.0	1.0	0	2.0
2107	0	3.091042	1	1	2.0	3.0	1.0	0	2.0
2108	0	3.135494	1	1	2.0	3.0	1.0	0	2.0
2109	0	3.178054	1	1	2.0	3.0	1.0	0	3.0
2110	0	3.178054	1	1	2.0	3.0	1.0	0	3.0

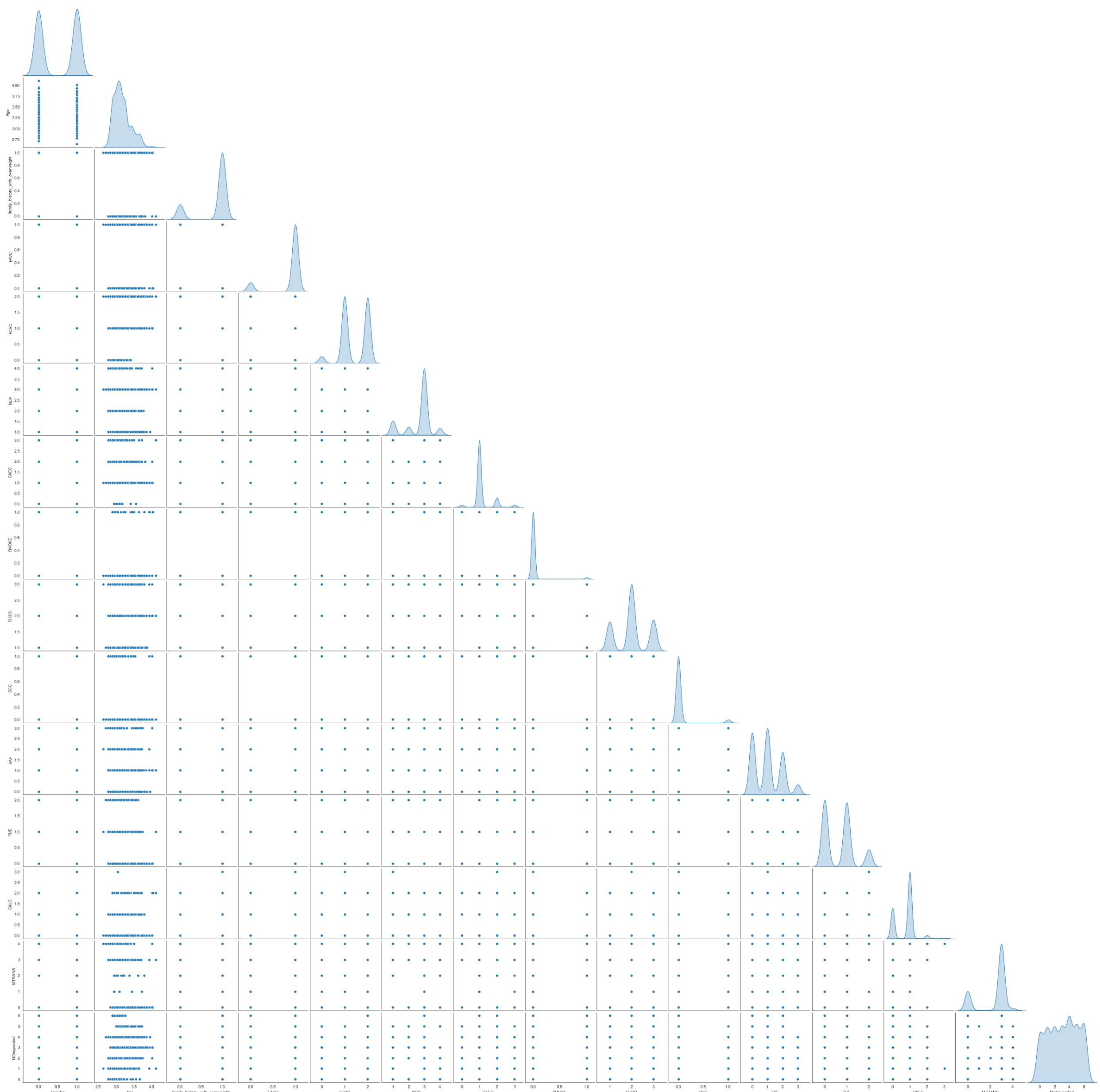
2111 rows × 15 columns

In [11]:
```
# correlation heatmap
corr_df = demo_df.corr().abs()

fig, ax = plt.subplots(figsize=(10,5))
sns.set_style('white')
mask = np.triu(np.ones_like(corr_df), k=1)
sns.heatmap(corr_df, annot=True, mask=mask, ax=ax).set(title='Absolute Pear

plt.show()
print("numeric abs(correlation) order: \n{}".format(corr_df['NObeyesdad'].s
```




```
numeric abs(correlation) order: 
family_history_with_overweight    0.505148
Age                               0.332940
CAEC                              0.329350
FAVC                              0.247793
FAF                               0.197186
FCVC                              0.195705
SCC                               0.194508
CALC                              0.151752
CH2O                              0.138171
TUE                               0.079528
Gender                            0.031464
NCP                               0.020931
MTRANS                            0.011818
SMOKE                             0.003442
Name: NObeyesdad, dtype: float64
```

In [12]:
```
# plot only lower triangle due to symmetry
# shows mostly categorical distributions
sns.pairplot(demo_df,
             corner=True,
             diag_kind='kde')
plt.show()
```



Train test split

In [13]:
```
# dropping target, Height, and Weight
col_to_drop = ["NObeyesdad", "Height", "Weight"]
X = df.drop(columns=col_to_drop)
y = df["NObeyesdad"]

X_train, X_test, y_train, y_test = train_test_split(X, y, stratify = y, tes
```

Inspect X train

In [14]:
```
X_train
```

Out[14]:

	Gender	Age	family_history_with_overweight	FAVC	FCVC	NCP	CAEC	SMOKE	CH2O
442	1	26.0	1	1	1.0	3.0	Sometimes	0	2.0
253	0	26.0	1	1	1.0	1.0	Sometimes	0	2.0
554	1	16.0	0	1	1.0	2.0	Sometimes	0	3.0
1500	1	24.0	1	1	1.0	3.0	Sometimes	0	1.0
359	0	33.0	1	0	1.0	3.0	Sometimes	0	2.0
...	...	...	...	...	...	...	...	...	...
434	1	19.0	0	1	1.0	4.0	Frequently	0	2.0
840	1	20.0	1	1	2.0	3.0	Sometimes	0	3.0
1794	1	31.0	1	1	2.0	2.0	Sometimes	0	1.0
155	0	31.0	1	1	2.0	1.0	Frequently	0	2.0
1329	0	23.0	1	1	1.0	1.0	Sometimes	0	2.0

1688 rows × 14 columns

Data preparation for X_train

Ordinal Encode

In [15]:
```
cols_to_ord = ["CAEC", "CALC"]

ord_enc = OrdinalEncoder(categories=[["no", "Sometimes", "Frequently", "Alw
                                     ["no", "Sometimes", "Frequently", "Alwa

X_train[cols_to_ord] = ord_enc.fit_transform(X_train[cols_to_ord])
X_test[cols_to_ord] = ord_enc.transform(X_test[cols_to_ord])
X_train.head()
```

Out[15]:

	Gender	Age	family_history_with_overweight	FAVC	FCVC	NCP	CAEC	SMOKE	CH2O	SCC
442	1	26.0	1	1	1.0	3.0	1.0	0	2.0	0
253	0	26.0	1	1	1.0	1.0	1.0	0	2.0	0
554	1	16.0	0	1	1.0	2.0	1.0	0	3.0	0
1500	1	24.0	1	1	1.0	3.0	1.0	0	1.0	0
359	0	33.0	1	0	1.0	3.0	1.0	0	2.0	0

See correlations of X_train

In [16]:
```
X_train.corr()
```

Out[16]:

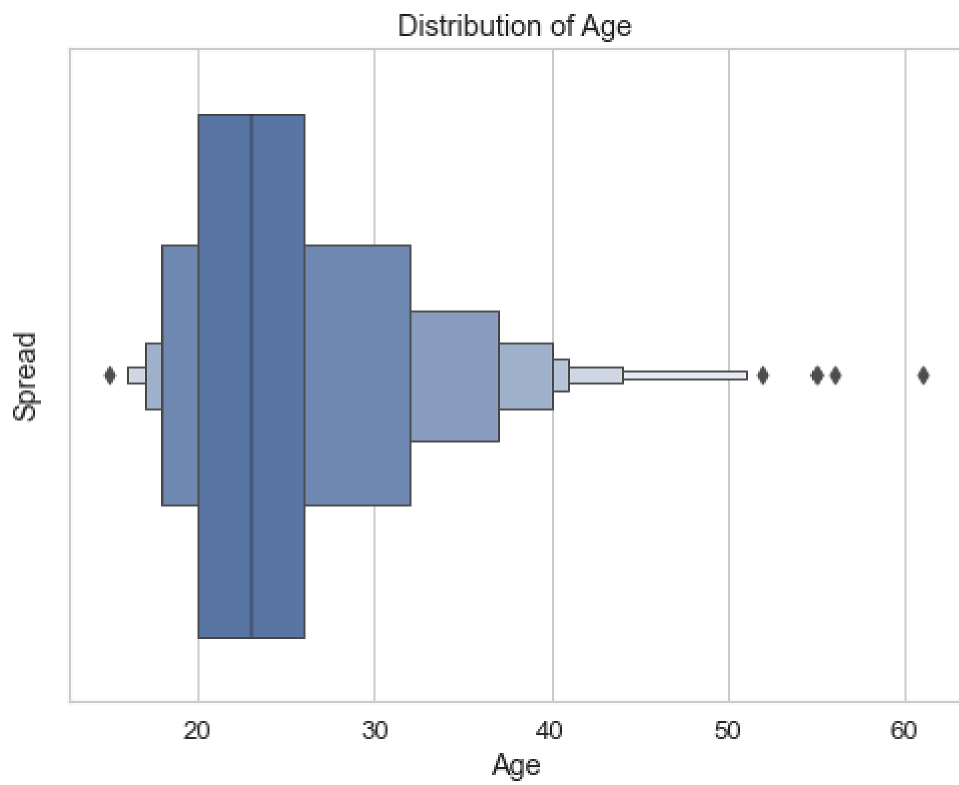
	Gender	Age	family_history_with_overweight	FAVC	FC
Gender	1.000000	0.045974	0.095035	0.078526	-0.272
Age	0.045974	1.000000	0.218257	0.083431	0.000
family_history_with_overweight	0.095035	0.218257	1.000000	0.204672	0.009
FAVC	0.078526	0.083431	0.204672	1.000000	-0.059
FCVC	-0.272743	0.000779	0.009883	-0.059913	1.000
NCP	0.081745	-0.053270	0.046047	-0.011639	0.041
CAEC	-0.092890	-0.082336	-0.158420	-0.166281	0.069
SMOKE	0.045388	0.094705	0.003502	-0.064057	0.002
CH2O	0.056896	-0.034033	0.128790	0.012077	0.052
SCC	-0.098695	-0.121402	-0.180247	-0.199856	0.076
FAF	0.202470	-0.127657	-0.058395	-0.118977	0.021
TUE	0.015315	-0.283199	0.024205	0.068491	-0.087
CALC	0.006922	0.036785	-0.029362	0.119627	0.035
MTRANS	-0.143400	-0.606381	-0.106733	-0.079010	0.083

In [17]:
```
# Make sure values are encoded properly
X_train.FCVC.value_counts()
```

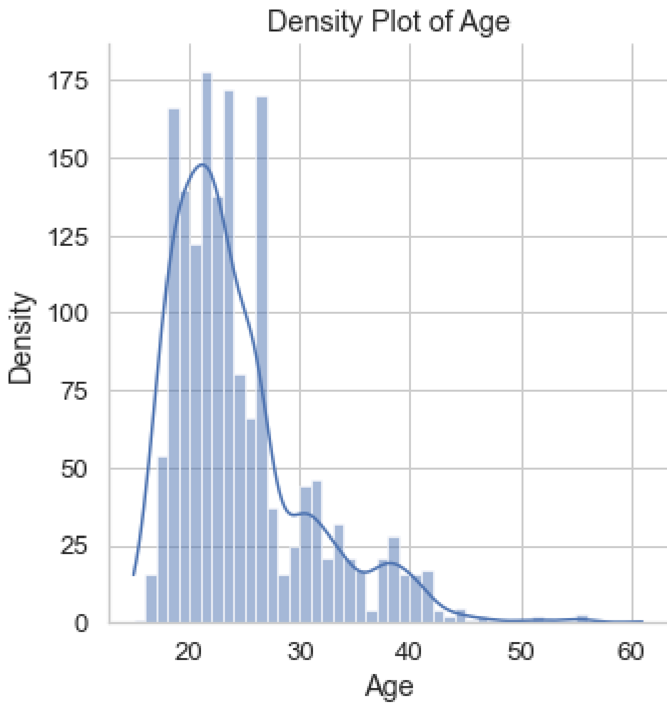
Out[17]:
```
1.0    817
2.0    789
0.0     82
Name: FCVC, dtype: int64
```

Normalize Age

In [18]:
```
# Distribution is Laplace
sns.set(style='whitegrid', font_scale=1.2)
plt.figure(figsize=(8, 6))
sns.boxenplot(X_train.Age)
plt.ylabel('Spread')
plt.title('Distribution of Age')
plt.show()

plt.figure(figsize=(8, 6))
sns.displot(X_train.Age, kde=True)
plt.ylabel('Density')
plt.title('Density Plot of Age')
plt.show()
```



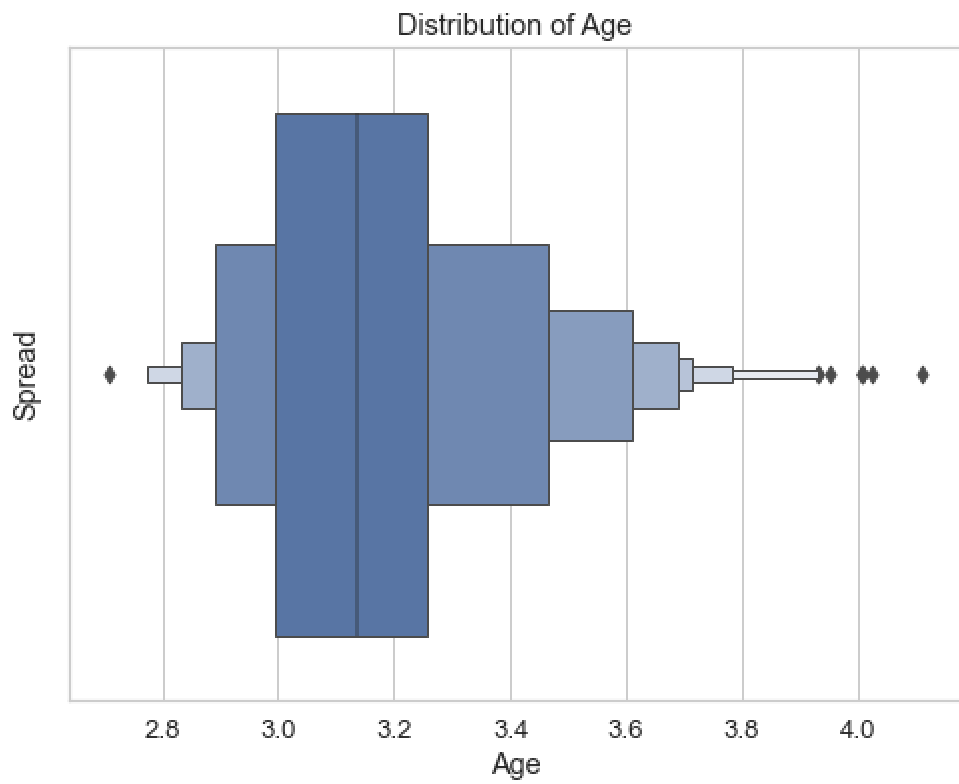

```
<Figure size 576x432 with 0 Axes>
```



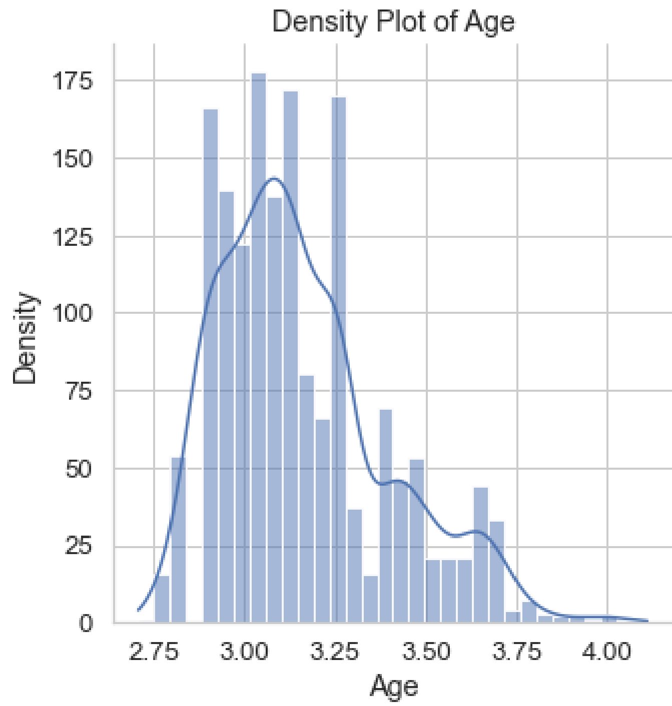
Log Transform Age

In [19]:
```
# Log transform for a more normal distribution
X_train["Age"] = [np.log(x) for x in X_train["Age"]]
X_test["Age"] = [np.log(x) for x in X_test["Age"]]
```

In [20]:
```
# show Age after transform
sns.set(style='whitegrid', font_scale=1.2)
plt.figure(figsize=(8, 6))
sns.boxenplot(X_train.Age)
plt.ylabel('Spread')
plt.title('Distribution of Age')
plt.show()

plt.figure(figsize=(8, 6))
sns.displot(X_train.Age, kde=True)
plt.ylabel('Density')
plt.title('Density Plot of Age')
plt.show()
```




```
<Figure size 576x432 with 0 Axes>
```



Visualize columns in X_train

In [21]:
```
# quick inspect each column in X train to see distributions
sns.set(style='whitegrid', font_scale=1.2)
[display(sns.displot(X_train[x]), X_train[x].value_counts(), plt.title(f"Di
```

```
<seaborn.axisgrid.FacetGrid at 0x7ff729b65100>

1    846
0    842
Name: Gender, dtype: int64

Text(0.5, 1.0, 'Distribution of Gender')

<seaborn.axisgrid.FacetGrid at 0x7ff72a098880>

3.044522    178
3.135494    172
3.258097    170
2.890372    166
2.944439    140
3.091042    138
2.995732    122
3.178054     80
3.218876     66
2.833213     54
```

Check VIF

In [22]:
```
vif_data = pd.DataFrame()
vif_data["feature"] = X_train.columns

# calculating VIF for each feature
vif_data["VIF"] = [variance_inflation_factor(X_train.values, i) for i in ra

print(vif_data)
```

```
                          feature        VIF
0                          Gender   2.373448
1                             Age  38.073479
2  family_history_with_overweight   6.446876
3                            FAVC   9.606411
4                            FCVC   7.732581
5                             NCP  12.027061
6                            CAEC   7.313921
7                           SMOKE   1.051737
8                            CH2O  10.133575
9                             SCC   1.128639
10                            FAF   2.495818
11                            TUE   2.045901
12                           CALC   3.138217
13                         MTRANS   4.390265
```

VIF for Age is very high. Models were tested with and without Age. Age helped the models and boosted test accuracy across the board by about 7 basis points.

Model Testing

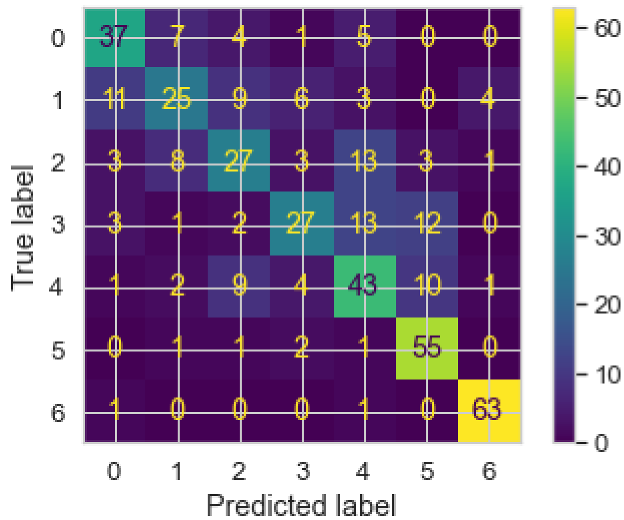
Test 1: Logistic Regression__OneHotEncoder

In [23]:

```
steps = [("hot_enc", OneHotEncoder(handle_unknown="ignore")), ("clf", Logis

pipeline = Pipeline(steps)


# Train the pipeline (tranformations & predictor)
pipeline.fit(X_train, y_train)

# Predict using the pipeline (includes the transfomers & trained predictor)
predicted = pipeline.predict(X_test)

#estimated time to run commented out is around 40 min
pipe_grid = {
            # "hot_enc__drop": ["first", "if_binary"],
            # "hot_enc__handle_unknown": ["ignore"],
            # "clf__penalty": ["l1", "l2", "elasticnet", None],
            # "clf__max_iter": [x for x in range(0, 100, 10)],
            # "clf__solver": ["lbfgs", "liblinear", "newton-cg", "newton-ch
            }


gs_pipe = GridSearchCV(estimator=pipeline,
                       param_grid=pipe_grid,
                       n_jobs=-1
                       )

gs_pipe.fit(X_train, y_train)
display(gs_pipe.best_params_)
display(gs_pipe.best_score_)
display(gs_pipe.best_estimator_)

best_model = gs_pipe.best_estimator_
y_pred = best_model.predict(X_test)
plot_confusion_matrix(best_model,X_test,y_test);
```

```
{}

0.6344793074991659
```

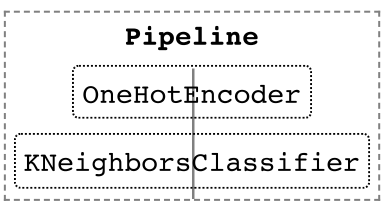
```
Pipeline
OneHotEncoder
LogisticRegression
```

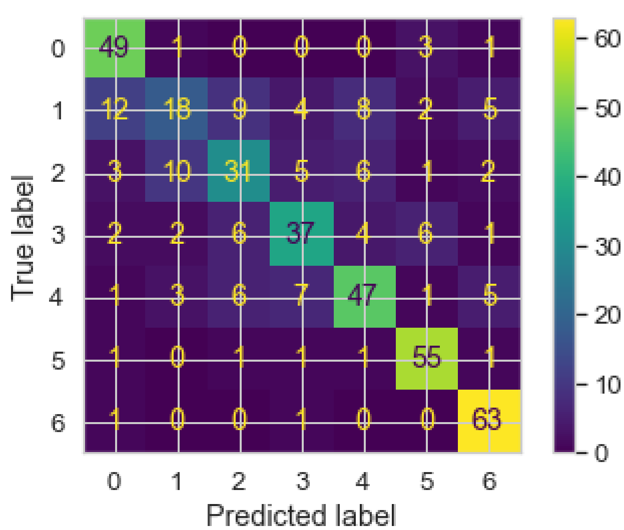


Test 2: K Nearest Neighbor__OneHotEncoder

In [24]:
```
steps = [("hot_enc", OneHotEncoder(handle_unknown="ignore")), ("knn", KNeig

pipeline = Pipeline(steps)

# Train the pipeline (tranformations & predictor)
pipeline.fit(X_train, y_train)

# Predict using the pipeline (includes the transfomers & trained predictor)
predicted = pipeline.predict(X_test)

#estimated time to run commented out is around 20 min
pipe_grid = {
            #"hot_enc__handle_unknown": ["ignore", "infrequent_if_exist"],
            # 'knn__n_neighbors': [3, 5, 7, 9, 11, 13, 15, 18],
            # 'knn__p': [1, 2, 3, 4]
             }


gs_pipe = GridSearchCV(estimator=pipeline,
                       param_grid=pipe_grid,
                       n_jobs=-1)

gs_pipe.fit(X_train, y_train)
display(gs_pipe.best_params_)
display(gs_pipe.best_score_)
display(gs_pipe.best_estimator_)
best_model = gs_pipe.best_estimator_
y_pred = best_model.predict(X_test)
plot_confusion_matrix(best_model,X_test,y_test);
```

```
{}

0.6872017277404175
```





Test 3: Decision Tree__OneHotEncoder

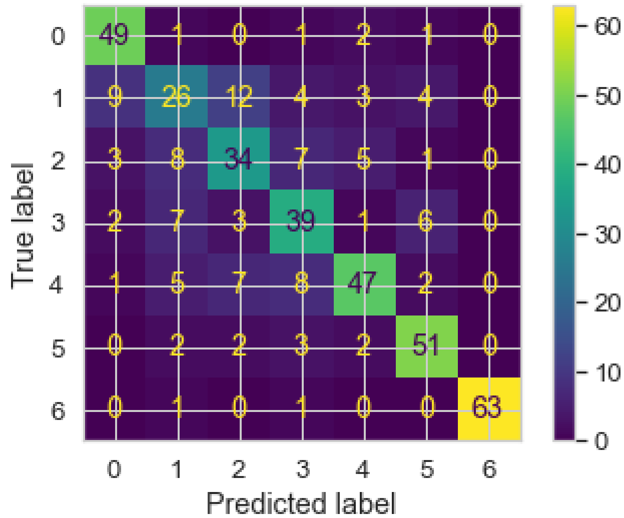
In [25]:
```
steps = [("hot_enc", OneHotEncoder(handle_unknown="ignore")), ('rf_clf', De

pipeline = Pipeline(steps)

# Train the pipeline (tranformations & predictor)
pipeline.fit(X_train, y_train)

# Predict using the pipeline (includes the transfomers & trained predictor)
predicted = pipeline.predict(X_test)

#estimated time to run commented out is around 40 min

pipe_grid = {
            # #"hot_enc__handle_unknown": ["ignore", "infrequent_if_exist"]
            # "rf_clf__criterion": ["gini", "entropy", "log_loss"],
            #  "rf_clf__splitter": ["best", "random"],
            #  "rf_clf__max_depth": [x for x in range(0,100,10)],
            #  "rf_clf__max_features": ["auto", "sqrt", "log2"]
            }

second_pipe = GridSearchCV(estimator=pipeline,
                       param_grid=pipe_grid,
                       n_jobs=-1)

second_pipe.fit(X_train, y_train)
display(second_pipe.best_params_)
display(second_pipe.best_score_)
display(second_pipe.best_estimator_)
best_model = second_pipe.best_estimator_
y_pred = best_model.predict(X_test)
plot_confusion_matrix(best_model,X_test,y_test);
```

```
{}

0.6931592716801573
```

Pipeline

OneHotEncoder

DecisionTreeClassifier



Test 4: Random Forest__OneHotEncoder

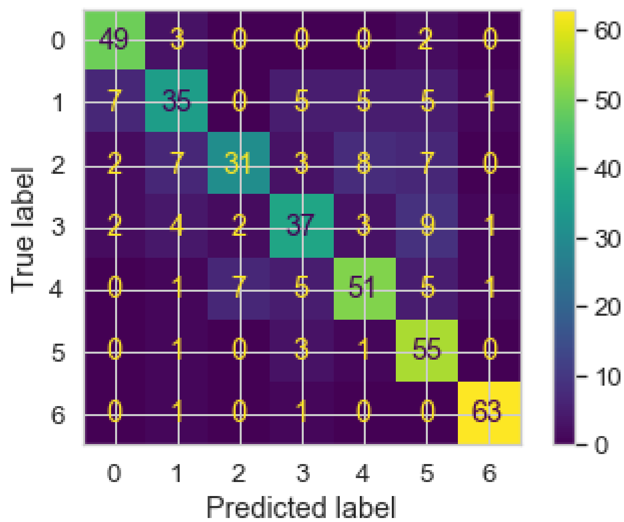
In [26]:
```
steps = [("hot_enc", OneHotEncoder(handle_unknown="ignore")), ('rf', Random

pipeline = Pipeline(steps)

# Train the pipeline (tranformations & predictor)
pipeline.fit(X_train, y_train)

# Predict using the pipeline (includes the transfomers & trained predictor)
predicted = pipeline.predict(X_test)

pipe_grid = {
            #"hot_enc__handle_unknown": ["ignore"],
            # "rf__n_estimators": [x for x in range(450,500,10)],
             "rf__criterion": ["entropy"],
             "rf__max_depth": [11],
             "rf__max_features": ["sqrt"]
            }

second_pipe = GridSearchCV(estimator=pipeline,
                       param_grid=pipe_grid,
                       n_jobs=-1)

second_pipe.fit(X_train, y_train)
display(second_pipe.best_params_)
display(second_pipe.best_score_)
display(second_pipe.best_estimator_)
best_model = second_pipe.best_estimator_
y_pred = best_model.predict(X_test)
plot_confusion_matrix(best_model,X_test,y_test);
```

```
{'rf__criterion': 'entropy', 'rf__max_depth': 11, 'rf__max_features': 'sq
rt'}

0.7417098309132092
```

Pipeline

OneHotEncoder

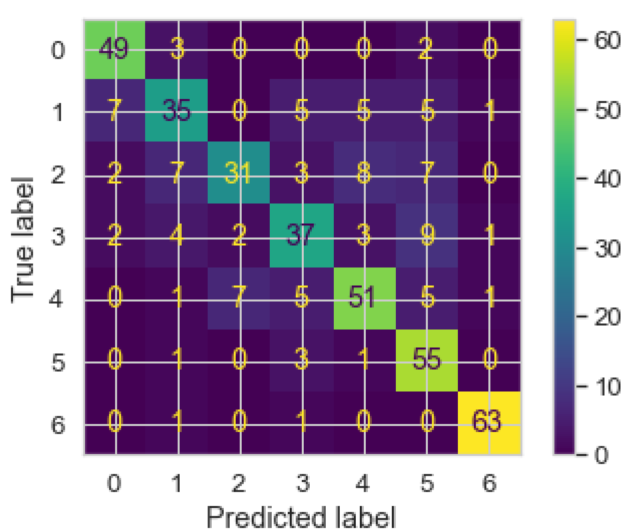
RandomForestClassifier



Test 5: XGBoost__StandardScaler

In [27]:
```
steps = [('std_scaler', StandardScaler()),('xgb_model', xgb.XGBClassifier()

pipeline = Pipeline(steps)

# Train the pipeline (tranformations & predictor)
pipeline.fit(X_train, y_train)

y_pred = pipeline.predict(X_test)
print(pipeline.score(X_test, y_test))
plot_confusion_matrix(best_model,X_test,y_test);
```

```
0.8037825059101655
```



Test 6: XGBoost__StandardScaler__SMOTE

In [28]:
```
# Define the pipeline
pipeline = make_pipeline(SMOTE(random_state=42),
                         StandardScaler(),
                         xgb.XGBClassifier())

# Train the pipeline (transformations & predictor)
pipeline.fit(X_train, y_train)

# Predict using the pipeline (includes the transformers & trained predictor
predicted = pipeline.predict(X_test)

# Define the parameter grid for the pipeline
pipe_grid = {
    'smote__k_neighbors': [1, 3, 5]
}

# Define the GridSearchCV object
second_pipe = GridSearchCV(
    estimator=pipeline,
    param_grid=pipe_grid,
    verbose=2,
    n_jobs=-1
)

# Fit the GridSearchCV object
second_pipe.fit(X_train, y_train)

# Display the best hyperparameters, best score, and best estimator
display(second_pipe.best_params_)
display(second_pipe.best_score_)
display(second_pipe.best_estimator_)

# Get the best model from the GridSearchCV object
best_model = second_pipe.best_estimator_

# Use the best model to make predictions on the test set
y_pred = best_model.predict(X_test)

# Plot the confusion matrix
plot_confusion_matrix(best_model, X_test, y_test);
```

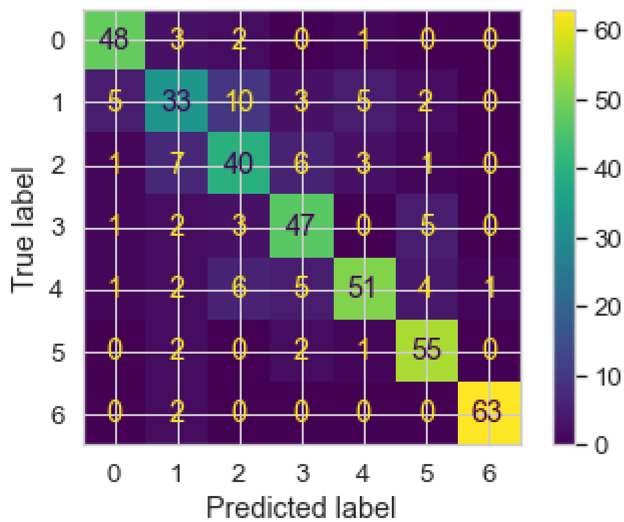
```
Fitting 5 folds for each of 3 candidates, totalling 15 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent worker
s.
[Parallel(n_jobs=-1)]: Done   8 out of  15 | elapsed:    3.1s remaining:
2.7s
[Parallel(n_jobs=-1)]: Done  15 out of  15 | elapsed:    5.7s finished

{'smote__k_neighbors': 5}

0.7831878917704073
```

Pipeline

SMOTE

StandardScaler

XGBClassifier



Test 7: RandomForest__StandardScaler

In [29]:
```
steps = [('std_scaler', StandardScaler()), ('rf', RandomForestClassifier(ra

pipeline = Pipeline(steps)

# Train the pipeline (tranformations & predictor)
pipeline.fit(X_train, y_train)

# Predict using the pipeline (includes the transfomers & trained predictor)
predicted = pipeline.predict(X_test)

pipe_grid = {
            "rf__n_estimators": [457],
            "rf__criterion": ["entropy"],
            "rf__max_depth": [11],
            "rf__max_features": ["log2"]
            }

# note: default scoring is aaccuracy
second_pipe = GridSearchCV(estimator = pipeline,
                  param_grid = pipe_grid)

second_pipe.fit(X_train, y_train)
display(second_pipe.best_params_)
display(second_pipe.best_score_)
display(second_pipe.best_estimator_)
best_model = second_pipe.best_estimator_
feat_imp = best_model["rf"].feature_importances_
feat_imp_series = pd.Series(feat_imp,
    index = X_train.columns).sort_values(
    ascending = False)
display(feat_imp)
best_model.fit(X_train,y_train)
y_pred = best_model.predict(X_test)
print(best_model.score(X_test, y_test))
plot_confusion_matrix(best_model,X_test,y_test);
```

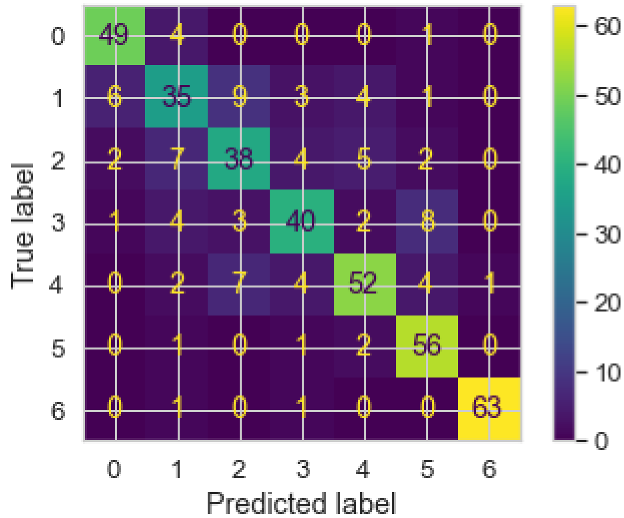
```
{'rf__criterion': 'entropy',
 'rf__max_depth': 11,
 'rf__max_features': 'log2',
 'rf__n_estimators': 457}

0.7849577721981282
```

Pipeline

StandardScaler

RandomForestClassifier

```
array([0.12165457, 0.19170952, 0.08088179, 0.03886634, 0.09000181,
       0.07543686, 0.07454219, 0.00450825, 0.05122347, 0.01491348,
       0.06597757, 0.0530521 , 0.08183445, 0.05539759])

0.7872340425531915
```



In [30]:
```
# show columns of X_train
X_train.columns
```

Out[30]:
```
Index(['Gender', 'Age', 'family_history_with_overweight', 'FAVC', 'FCVC',
       'NCP', 'CAEC', 'SMOKE', 'CH2O', 'SCC', 'FAF', 'TUE', 'CALC', 'MTRA
NS'],
      dtype='object')
```

Test 8: Random Forest__StandardScaler__SMOTE

In [31]:
```
# Create pipeline with SMOTE, StandardScaler and RandomForestClassifier
pipeline = make_pipeline(
    # OneHotEncoder(handle_unknown="ignore", sparse=False),
    SMOTE(random_state=42, k_neighbors=1),
    StandardScaler(),
    RandomForestClassifier(random_state=42, n_jobs=-1)
)

# Train the pipeline (transformations & predictor)
pipeline.fit(X_train, y_train)

# Predict using the pipeline (includes the transformers & trained predictor
predicted = pipeline.predict(X_test)

# Define the grid of hyperparameters to search over
pipe_grid = {
    "randomforestclassifier__n_estimators": [456],
    "randomforestclassifier__criterion": ["entropy"],
    "randomforestclassifier__max_depth": [11],
    "randomforestclassifier__max_features": ["sqrt"],
    "randomforestclassifier__class_weight": ["balanced_subsample"],
    "randomforestclassifier__min_impurity_decrease": [0.0]
}

# Perform grid search to find the best combination of hyperparameters
second_pipe = GridSearchCV(
    estimator=pipeline,
    param_grid=pipe_grid,
    n_jobs=-1,
    verbose=2
)

second_pipe.fit(X_train, y_train)
display(second_pipe.best_params_)
display(second_pipe.best_score_)
display(second_pipe.best_estimator_)
best_model = second_pipe.best_estimator_
best_model.fit(X_train,y_train)
y_pred = best_model.predict(X_test)
print(best_model.score(X_test, y_test))
plot_confusion_matrix(best_model,X_test,y_test);
```

```
Fitting 5 folds for each of 1 candidates, totalling 5 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent worker
s.
[Parallel(n_jobs=-1)]: Done   2 out of   5 | elapsed:    2.4s remaining:
3.7s
[Parallel(n_jobs=-1)]: Done   5 out of   5 | elapsed:    2.5s remaining:
0.0s
[Parallel(n_jobs=-1)]: Done   5 out of   5 | elapsed:    2.5s finished
```

```
{'randomforestclassifier__class_weight': 'balanced_subsample',
 'randomforestclassifier__criterion': 'entropy',
 'randomforestclassifier__max_depth': 11,
 'randomforestclassifier__max_features': 'sqrt',
 'randomforestclassifier__min_impurity_decrease': 0.0,
 'randomforestclassifier__n_estimators': 456}
```

```
0.7890997840324478
```

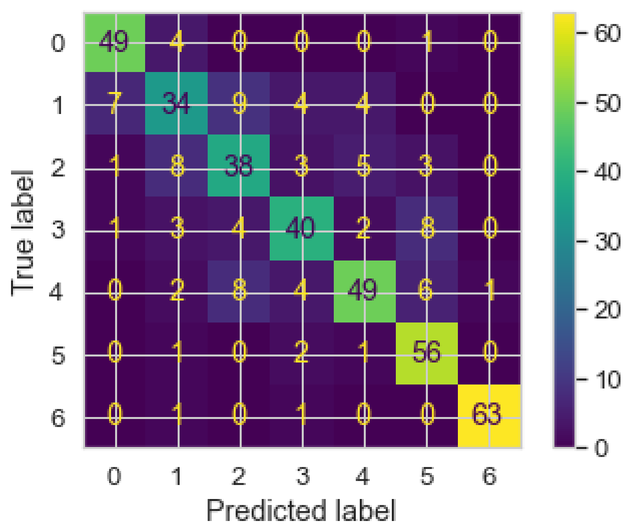
Pipeline

SMOTE

StandardScaler

RandomForestClassifier

```
0.7777777777777778
```



Test 9: Random Forest__StandardScaler__SMOTE

In [32]:

```python
# Create pipeline with SMOTE, StandardScaler and RandomForestClassifier
pipeline = make_pipeline(
    # OneHotEncoder(handle_unknown="ignore", sparse=False),
    SMOTE(random_state=42, k_neighbors=1),
    StandardScaler(),
    RandomForestClassifier(random_state=42, n_jobs=-1)
)

# Train the pipeline (transformations & predictor)
pipeline.fit(X_train, y_train)

# Predict using the pipeline (includes the transformers & trained predictor
predicted = pipeline.predict(X_test)

# Define the grid of hyperparameters to search over
pipe_grid = {
    "randomforestclassifier__n_estimators": [457],
    "randomforestclassifier__criterion": ["entropy"],
    "randomforestclassifier__max_depth": [11],
    "randomforestclassifier__max_features": ["sqrt","log2"],
    "randomforestclassifier__class_weight": ["balanced"],
    "randomforestclassifier__min_impurity_decrease": [0.00000001,],
}

# Perform grid search to find the best combination of hyperparameters
second_pipe = GridSearchCV(
    estimator=pipeline,
    param_grid=pipe_grid,
    n_jobs=-1
)

second_pipe.fit(X_train, y_train)
display(second_pipe.best_params_)
display(second_pipe.best_score_)
display(second_pipe.best_estimator_)
best_model = second_pipe.best_estimator_
y_pred = best_model.predict(X_test)
plot_confusion_matrix(best_model,X_test,y_test);
```

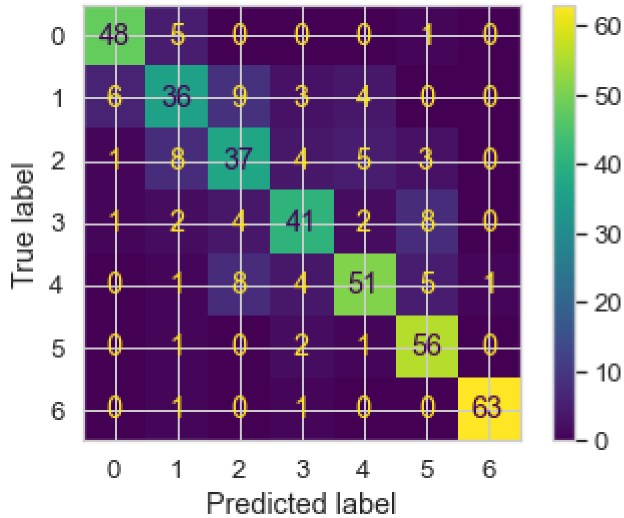
```
{'randomforestclassifier__class_weight': 'balanced',
 'randomforestclassifier__criterion': 'entropy',
 'randomforestclassifier__max_depth': 11,
 'randomforestclassifier__max_features': 'sqrt',
 'randomforestclassifier__min_impurity_decrease': 1e-08,
 'randomforestclassifier__n_estimators': 457}

0.7891050515337208
```

Pipeline

SMOTE

StandardScaler

RandomForestClassifier



Test 10: Random Forest__StandardScaler__RandomOverSampler

In [33]:
```
# Define the pipeline
pipeline = make_pipeline(
    RandomOverSampler(random_state=42),
    StandardScaler(),
    RandomForestClassifier(random_state=42)
)

# Train the pipeline (transformations & predictor)
pipeline.fit(X_train, y_train)

# Predict using the pipeline (includes the transformers & trained predictor
predicted = pipeline.predict(X_test)

# Define the grid of hyperparameters to search over
pipe_grid = {
    "randomoversampler__sampling_strategy": [{0:281,1:281,2:281,3:281,4:281
    "randomforestclassifier__n_estimators": [457],
    "randomforestclassifier__criterion": ["entropy"],
    "randomforestclassifier__max_depth": [11],
    "randomforestclassifier__max_features": ["log2"],
    "randomforestclassifier__class_weight": ["balanced", "balanced_subsampl
}

# Perform grid search to find the best combination of hyperparameters
second_pipe = GridSearchCV(
    estimator=pipeline,
    param_grid=pipe_grid,
    n_jobs=-1
)

second_pipe.fit(X_train, y_train)
display(second_pipe.best_params_)
display(second_pipe.best_score_)
display(second_pipe.best_estimator_)
best_model = second_pipe.best_estimator_
y_pred = best_model.predict(X_test)
plot_confusion_matrix(best_model,X_test,y_test);
```

```
{'randomforestclassifier__class_weight': 'balanced',
 'randomforestclassifier__criterion': 'entropy',
 'randomforestclassifier__max_depth': 11,
 'randomforestclassifier__max_features': 'log2',
 'randomforestclassifier__n_estimators': 457,
 'randomoversampler__sampling_strategy': {0: 281,
  1: 281,
  2: 281,
  3: 281,
  4: 281,
  5: 300,
  6: 259}}
```

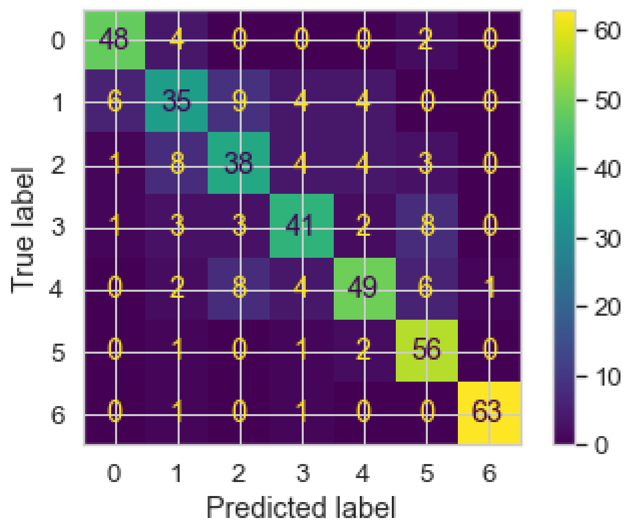
```
0.7920636314153775
```

Pipeline

RandomOverSampler

StandardScaler

RandomForestClassifier



Test 11: Guassian Bayes__StandardScaler__SMOTE

In [34]:
```
# Create pipeline with SMOTE, StandardScaler and RandomForestClassifier
pipeline = make_pipeline(
    # OneHotEncoder(handle_unknown="ignore", sparse=False),
    SMOTE(random_state=42, k_neighbors=1),
    StandardScaler(),
    GaussianNB()
)

# Train the pipeline (transformations & predictor)
pipeline.fit(X_train, y_train)

# Predict using the pipeline (includes the transformers & trained predictor
predicted = pipeline.predict(X_test)

# Define the grid of hyperparameters to search over
pipe_grid = {
}

# Perform grid search to find the best combination of hyperparameters
second_pipe = GridSearchCV(
    estimator=pipeline,
    param_grid=pipe_grid,
    verbose=2,
    n_jobs=-1
)

second_pipe.fit(X_train, y_train)
display(second_pipe.best_params_)
display(second_pipe.best_score_)
display(second_pipe.best_estimator_)
best_model = second_pipe.best_estimator_
y_pred = best_model.predict(X_test)
plot_confusion_matrix(best_model,X_test,y_test);
```

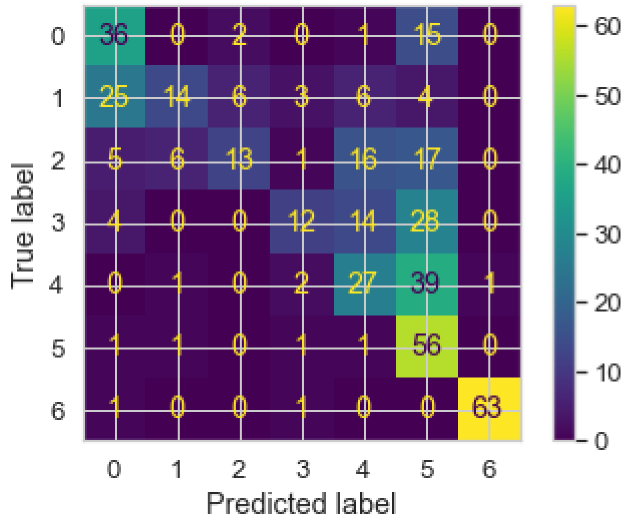
```
Fitting 5 folds for each of 1 candidates, totalling 5 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent worker
s.
[Parallel(n_jobs=-1)]: Done   2 out of   5 | elapsed:    0.0s remaining:
0.1s
[Parallel(n_jobs=-1)]: Done   5 out of   5 | elapsed:    0.1s remaining:
0.0s
[Parallel(n_jobs=-1)]: Done   5 out of   5 | elapsed:    0.1s finished

{}

0.5095201306340316
```

Pipeline

SMOTE

StandardScaler

GaussianNB



Test 12: Random Forest__StandardScaler__RandomOverSampler

In [35]:
```
# Define the pipeline
pipeline = make_pipeline(
    RandomOverSampler(random_state=42),
    StandardScaler(),
    RandomForestClassifier(random_state=42)
)

# Train the pipeline (transformations & predictor)
pipeline.fit(X_train, y_train)

# Predict using the pipeline (includes the transformers & trained predictor
predicted = pipeline.predict(X_test)

# Define the grid of hyperparameters to search over
pipe_grid = {
    "randomoversampler__sampling_strategy": ["minority","minority","minorit
    "randomforestclassifier__n_estimators": [457],
    "randomforestclassifier__criterion": ["entropy"],
    "randomforestclassifier__max_depth": [11],
    "randomforestclassifier__max_features": ["log2"]
}

# Perform grid search to find the best combination of hyperparameters
second_pipe = GridSearchCV(
    estimator=pipeline,
    param_grid=pipe_grid,
    verbose=2
)

second_pipe.fit(X_train, y_train)
display(second_pipe.best_params_)
display(second_pipe.best_score_)
display(second_pipe.best_estimator_)
best_model = second_pipe.best_estimator_
y_pred = best_model.predict(X_test)
plot_confusion_matrix(best_model,X_test,y_test);
```

```
Fitting 5 folds for each of 7 candidates, totalling 35 fits
[CV] randomforestclassifier__criterion=entropy, randomforestclassifier__m
ax_depth=11, randomforestclassifier__max_features=log2, randomforestclass
ifier__n_estimators=457, randomoversampler__sampling_strategy=minority

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent w
orkers.

[CV]  randomforestclassifier__criterion=entropy, randomforestclassifier__
max_depth=11, randomforestclassifier__max_features=log2, randomforestclas
sifier__n_estimators=457, randomoversampler__sampling_strategy=minority,
total=   1.0s
[CV] randomforestclassifier__criterion=entropy, randomforestclassifier__m
ax_depth=11, randomforestclassifier__max_features=log2, randomforestclass
ifier__n_estimators=457, randomoversampler__sampling_strategy=minority

[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    1.0s remaining:
0.0s
```

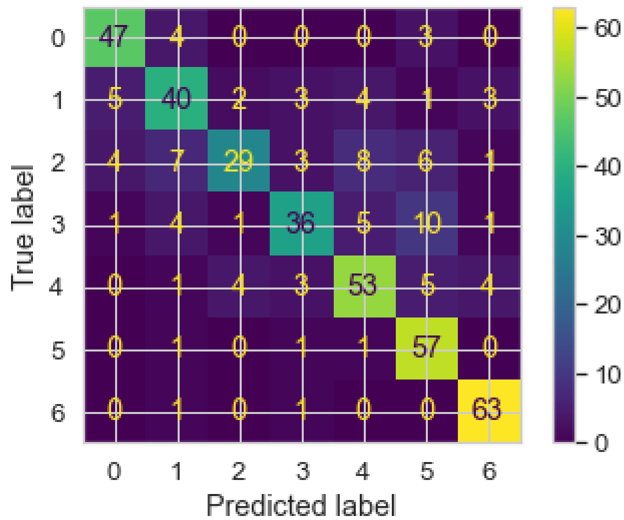
Test 13: Random Forest__Polynomial Features__OneHotEncoder

In [36]:
```
steps = [("hot_enc", OneHotEncoder(handle_unknown="ignore")), ("poly", Poly

pipeline = Pipeline(steps)

# Train the pipeline (tranformations & predictor)
pipeline.fit(X_train, y_train)

# Predict using the pipeline (includes the transfomers & trained predictor)
predicted = pipeline.predict(X_test)

pipe_grid = {
            #"hot_enc__handle_unknown": ["ignore"],
            "rf__n_estimators": [464],
            "rf__criterion": ["entropy"],
            "rf__max_depth": [11],
            "rf__max_features": ["log2"]
            }

# note: default scoring is aaccuracy
second_pipe = GridSearchCV(estimator = pipeline,
                  param_grid = pipe_grid,
                  n_jobs=-1)

second_pipe.fit(X_train, y_train)
display(second_pipe.best_params_)
display(second_pipe.best_score_)
display(second_pipe.best_estimator_)
best_model = second_pipe.best_estimator_
y_pred = best_model.predict(X_test)
plot_confusion_matrix(best_model,X_test,y_test);
```

```
{'rf__criterion': 'entropy',
 'rf__max_depth': 11,
 'rf__max_features': 'log2',
 'rf__n_estimators': 464}

0.7417221217495127
```

Pipeline

```
OneHotEncoder
PolynomialFeatures
RandomForestClassifier
```



Baseline Model: Random Forest__StandardScaler__SMOTE

In [37]:
```
def gen_weights(num_cat):
    """
    Generate all possible combinations of weights for a given number of cat

    Args:
    - num_cat (int): the number of categories to generate weights for.

    Returns:
    - A list of dictionaries, where each dictionary represents a unique com
      The keys of each dictionary are the category numbers (0 to num_cat -
    """
    master_weights = []
    weights = {}
    for cat in range(num_cat):
        weights.update({cat: .1})

    for i in range(num_cat):
        weight_values = list(np.arange(.4, 1.2, 0.2))
        weight_combinations = list(itertools.product(weight_values, repeat=
        for combination in weight_combinations:
            temp_weights = weights.copy()
            for j in range(num_cat):
                temp_weights[j] = combination[j]
            master_weights.append(temp_weights.copy())

    return list(map(dict, set(frozenset(d.items()) for d in master_weights)

weights = gen_weights(7)

len(weights)
```

Out[37]: 16384

In [38]:
```
# Create pipeline with SMOTE, StandardScaler and RandomForestClassifier
pipeline = make_pipeline(
    # OneHotEncoder(handle_unknown="ignore", sparse=False),
    SMOTE(random_state=42, k_neighbors=1),
    StandardScaler(),
    RandomForestClassifier(random_state=42)
)

# Train the pipeline (transformations & predictor)
pipeline.fit(X_train, y_train)

# Predict using the pipeline (includes the transformers & trained predictor
predicted = pipeline.predict(X_test)

# Define the grid of hyperparameters to search over
pipe_grid = {
    "randomforestclassifier__n_estimators": [456],
    "randomforestclassifier__criterion": ["entropy"],
    "randomforestclassifier__max_depth": [11],
    "randomforestclassifier__max_features": ["sqrt"],
    "randomforestclassifier__class_weight": [{0: 1.2000000000000002, 1: 1.0
    "randomforestclassifier__min_impurity_decrease": [0],
    "randomforestclassifier__min_samples_split": [2],
    "randomforestclassifier__min_samples_leaf": [1],
    "randomforestclassifier__max_samples": [0.6],
    "randomforestclassifier__bootstrap": [True],
    "randomforestclassifier__oob_score": [True],
    "randomforestclassifier__warm_start": [True]
}

# Perform grid search to find the best combination of hyperparameters
second_pipe = GridSearchCV(
    estimator=pipeline,
    param_grid=pipe_grid,
    verbose=2,
    n_jobs=-1
)

second_pipe.fit(X_train, y_train)
display(second_pipe.best_params_)
display(second_pipe.best_score_)
display(second_pipe.best_estimator_)
best_model = second_pipe.best_estimator_
feat_imp = best_model["randomforestclassifier"].feature_importances_
feat_imp_series = pd.Series(feat_imp,
    index = X_train.columns).sort_values(
    ascending = False)
display(feat_imp)
X_train.columns
best_model.fit(X_train,y_train)
y_pred = best_model.predict(X_test)

print("accuracy on test", best_model.score(X_test, y_test))
print("recall = ", recall_score(y_test, y_pred, average="macro"))
print("f1_score = ", f1_score(y_test, y_pred, average="macro"))
```

```
plot_confusion_matrix(best_model,X_test,y_test);
```

```
Fitting 5 folds for each of 1 candidates, totalling 5 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent worker
s.
[Parallel(n_jobs=-1)]: Done   2 out of   5 | elapsed:    2.1s remaining:
3.1s
[Parallel(n_jobs=-1)]: Done   5 out of   5 | elapsed:    2.1s remaining:
0.0s
[Parallel(n_jobs=-1)]: Done   5 out of   5 | elapsed:    2.1s finished

{'randomforestclassifier__bootstrap': True,
 'randomforestclassifier__class_weight': {0: 1.2000000000000002,
  1: 1.0,
  2: 1.0,
  3: 0.8,
  4: 1.2000000000000002,
  5: 0.6,
  6: 0.6},
 'randomforestclassifier__criterion': 'entropy',
 'randomforestclassifier__max_depth': 11,
 'randomforestclassifier__max_features': 'sqrt',
 'randomforestclassifier__max_samples': 0.6,
 'randomforestclassifier__min_impurity_decrease': 0,
 'randomforestclassifier__min_samples_leaf': 1,
 'randomforestclassifier__min_samples_split': 2,
 'randomforestclassifier__n_estimators': 456,
 'randomforestclassifier__oob_score': True,
 'randomforestclassifier__warm_start': True}

0.7956156831071233
```

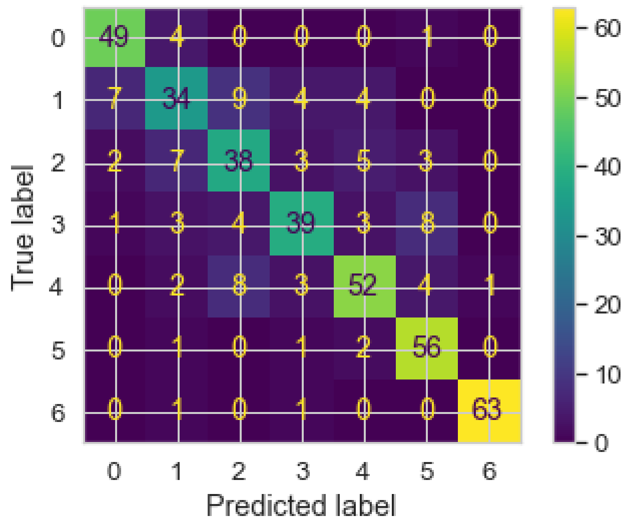
Pipeline

SMOTE

StandardScaler

RandomForestClassifier

```
array([0.09379391, 0.19810038, 0.08431835, 0.04367938, 0.07814037,
       0.07699818, 0.08260688, 0.00448546, 0.05568322, 0.01522204,
       0.07017576, 0.0590599 , 0.08200814, 0.05572803])

accuracy on test 0.7825059101654847
recall =  0.7809459651824184
f1_score =  0.7772203360117341
```

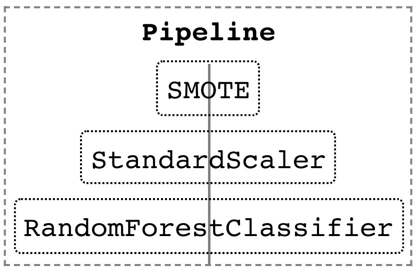


Further validating and comparing base model vs other options using a StackingRegressor

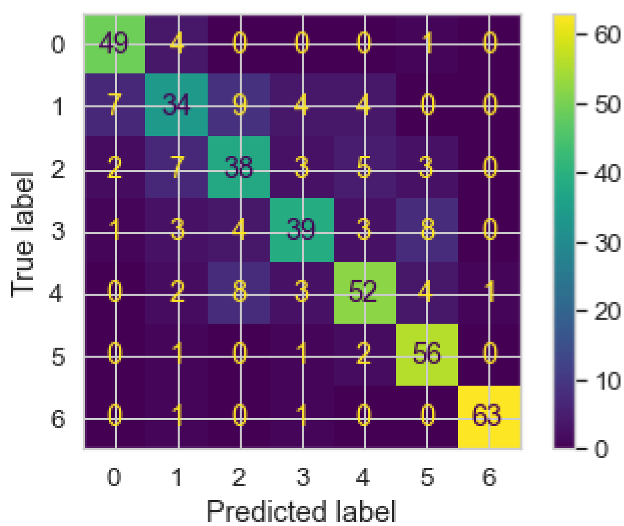
In [39]:
```
from imblearn.pipeline import Pipeline


# Create pipeline with SMOTE, StandardScaler and RandomForestClassifier
pipeline = Pipeline([
    ('smote', SMOTE(random_state=42, k_neighbors=1)),
    ('scaler', StandardScaler()),
    ('rf', RandomForestClassifier(random_state=42))
])

# Train the pipeline (transformations & predictor)
pipeline.fit(X_train, y_train)

# Predict using the pipeline (includes the transformers & trained predictor
predicted = pipeline.predict(X_test)

# Define the grid of hyperparameters to search over
pipe_grid = {
    "rf__n_estimators": [456],
    "rf__criterion": ["entropy"],
    "rf__max_depth": [11],
    "rf__max_features": ["sqrt"],
    "rf__class_weight": [{0: 1.2000000000000002, 1: 1.0, 2: 1.0, 3: 0.8, 4:
    "rf__min_impurity_decrease": [0],
    "rf__min_samples_split": [2],
    "rf__min_samples_leaf": [1],
    "rf__max_samples": [0.6],
    "rf__bootstrap": [True],
    "rf__oob_score": [True],
    "rf__warm_start": [True]
}

# Perform grid search to find the best combination of hyperparameters
second_pipe = GridSearchCV(
    estimator=pipeline,
    param_grid=pipe_grid,
    verbose=2,
    n_jobs=-1
)

second_pipe.fit(X_train, y_train)

# Fit the stacking regressor to the predicted X values and the true labels
estimators = [
    ('lr', LinearRegression()),
    ('knn', KNeighborsRegressor()),
    ('rt', DecisionTreeRegressor()),
]
sr = StackingRegressor(estimators)
final_p = Pipeline([('model', sr)])
final_p.fit(predicted.reshape(-1, 1), y_test)

# Get the best model from the grid search
best_model = second_pipe.best_estimator_
display(best_model)
display(second_pipe.best_score_)
# Get the feature importances of the best model
```

```
feat_imp = best_model.named_steps['rf'].feature_importances_
feat_imp_series = pd.Series(feat_imp, index=X_train.columns).sort_values(as
display(feat_imp)
# Make predictions using the best model
y_pred = best_model.predict(X_test)

# Plot the confusion matrix for the best model
plot_confusion_matrix(best_model, X_test, y_test);
```

```
Fitting 5 folds for each of 1 candidates, totalling 5 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent worker
s.
[Parallel(n_jobs=-1)]: Done   2 out of   5 | elapsed:    1.9s remaining:
2.8s
[Parallel(n_jobs=-1)]: Done   5 out of   5 | elapsed:    2.0s remaining:
0.0s
[Parallel(n_jobs=-1)]: Done   5 out of   5 | elapsed:    2.0s finished
```




```
0.7956156831071233

array([0.09379391, 0.19810038, 0.08431835, 0.04367938, 0.07814037,
       0.07699818, 0.08260688, 0.00448546, 0.05568322, 0.01522204,
       0.07017576, 0.0590599 , 0.08200814, 0.05572803])
```



Final Model

In [40]:
```
pipeline = make_pipeline(
    # OneHotEncoder(handle_unknown="ignore", sparse=False),
    # SMOTE(random_state=42, k_neighbors=1),
    StandardScaler(),
    xgb.XGBClassifier())

# Train the pipeline (tranformations & predictor)
pipeline.fit(X_train, y_train)

y_pred = pipeline.predict(X_test)

pipe_grid = {
    'xgbclassifier__learning_rate': [0.1],
    'xgbclassifier__n_estimators': [200],
    'xgbclassifier__max_depth': [7],
    'xgbclassifier__min_child_weight': [1],
    'xgbclassifier__gamma': [0],
    'xgbclassifier__subsample': [0.8],
    'xgbclassifier__colsample_bytree': [0.6]
}

second_pipe = GridSearchCV(
    estimator=pipeline,
    param_grid=pipe_grid,
    verbose=2,
    n_jobs=-1,
    cv=5
)

second_pipe.fit(X_train, y_train)
display(second_pipe.best_params_)
display(second_pipe.best_score_)
display(second_pipe.best_estimator_)
best_model = second_pipe.best_estimator_
best_model.fit(X_train,y_train)
y_pred = best_model.predict(X_test)

feat_imp = best_model["xgbclassifier"].feature_importances_
feat_imp_series = pd.Series(feat_imp,
    index = X_train.columns).sort_values(
    ascending = False)
display(feat_imp)
display(X_train.columns)

print("accuracy on test", best_model.score(X_test, y_test))
print("recall = ", recall_score(y_test, y_pred, average="macro"))
print("f1_score = ", f1_score(y_test, y_pred, average="macro"))
plot_confusion_matrix(best_model,X_test,y_test);
```

```
Fitting 5 folds for each of 1 candidates, totalling 5 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent worker
s.
[Parallel(n_jobs=-1)]: Done   2 out of   5 | elapsed:    3.3s remaining:
4.9s
[Parallel(n_jobs=-1)]: Done   5 out of   5 | elapsed:    3.3s remaining:
0.0s
[Parallel(n_jobs=-1)]: Done   5 out of   5 | elapsed:    3.3s finished
```

```
{'xgbclassifier__colsample_bytree': 0.6,
 'xgbclassifier__gamma': 0,
 'xgbclassifier__learning_rate': 0.1,
 'xgbclassifier__max_depth': 7,
 'xgbclassifier__min_child_weight': 1,
 'xgbclassifier__n_estimators': 200,
 'xgbclassifier__subsample': 0.8}
```

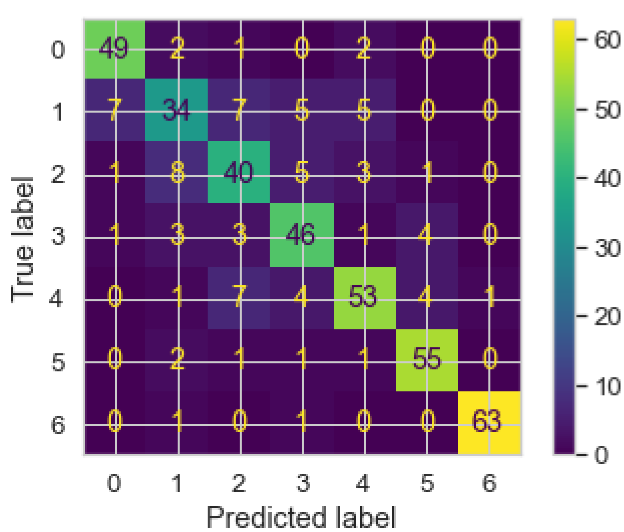
```
0.7950169437957614
```

```
Pipeline
StandardScaler
XGBClassifier
```

```
array([0.1143202 , 0.0627237 , 0.12640324, 0.07093425, 0.07636473,
       0.06553428, 0.11182177, 0.04112323, 0.03709772, 0.06637838,
       0.04186976, 0.04007904, 0.07396457, 0.07138525], dtype=float32)
```

```
Index(['Gender', 'Age', 'family_history_with_overweight', 'FAVC', 'FCVC',
       'NCP', 'CAEC', 'SMOKE', 'CH2O', 'SCC', 'FAF', 'TUE', 'CALC', 'MTRA
NS'],
      dtype='object')
```

```
accuracy on test 0.8037825059101655
recall =  0.8027733168127257
f1_score =  0.7997581009802529
```



Create probability distribution to inspect Gender by weight

class in the data

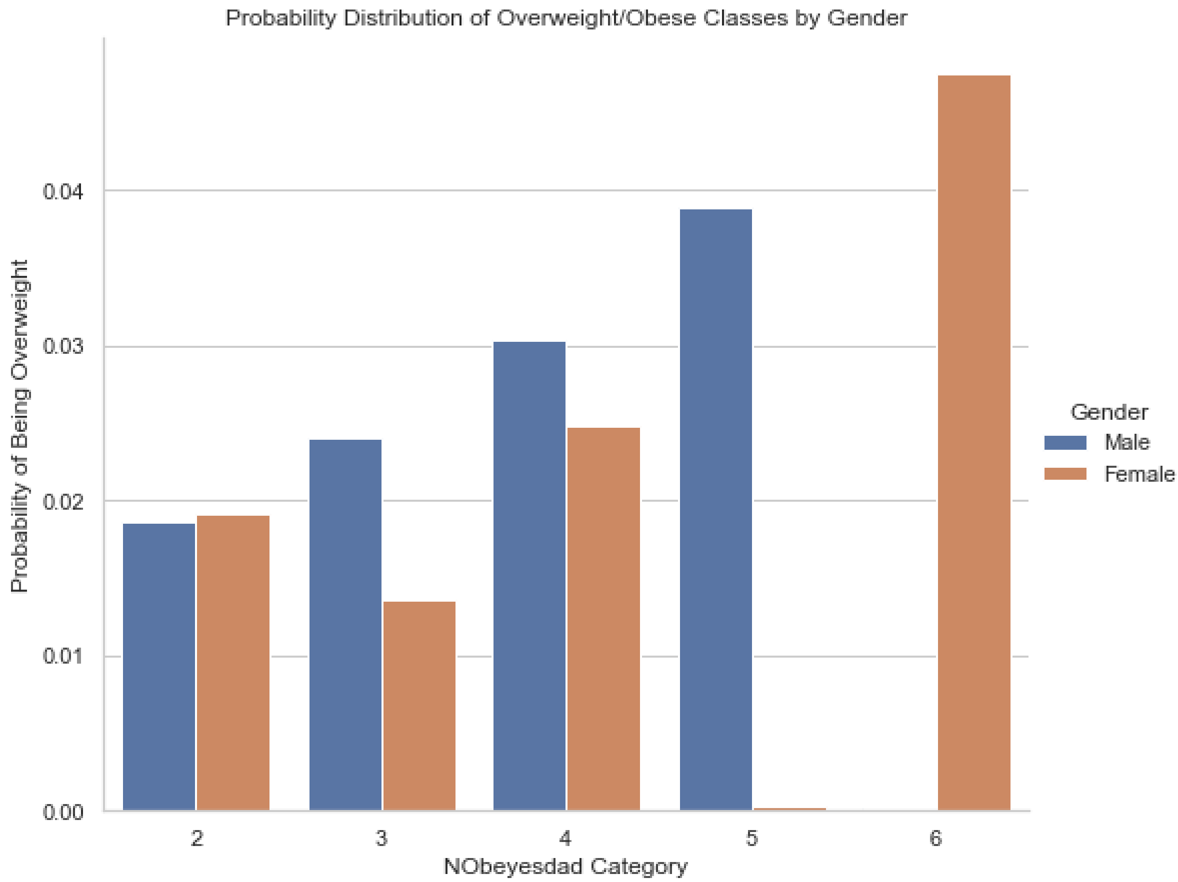
In [41]:
```
# Returns an iteration of probabilities for males and females for their wei

# Count the number of males and females
num_male = (df['Gender'] == 1).sum()
num_female = (df['Gender'] == 0).sum()

# Initialize a list to store the results
probabilities_male = []
probabilities_female = []

# Calculate the probabilities for each category of NObeyesdad equal to 2
for i in range(2, 7):
    # Count the number of overweight males and females
    num_overweight_male = ((df['Gender'] == 1) & (df['NObeyesdad'] == i)).s
    num_overweight_female = ((df['Gender'] == 0) & (df['NObeyesdad'] == i))

    # Calculate the overall probabilities
    p_male = num_male / len(df)
    p_female = num_female / len(df)
    p_overweight = (num_overweight_male + num_overweight_female) / len(df)

    # Calculate the conditional probability for males
    p_overweight_given_male = (num_overweight_male / len(df)) * p_overweigh
    probabilities_male.append(p_overweight_given_male)

    # Calculate the conditional probability for females
    p_overweight_given_female = (num_overweight_female / len(df)) * p_overw
    probabilities_female.append(p_overweight_given_female)

# Print the results for males
for i, prob in enumerate(probabilities_male):
    category = i + 2
    print("If you are male and in category", category, "the probability of

# Print the results for females
for i, prob in enumerate(probabilities_female):
    category = i + 2
    print("If you are female and in category", category, "the probability o


# Create a data frame with the probabilities for males and females
data = pd.DataFrame({
    'Gender': ['Male']*5 + ['Female']*5,
    'Category': [2, 3, 4, 5, 6]*2,
    'Probability': probabilities_male + probabilities_female
})

# Set the style of the chart
sns.set(style="whitegrid")

# Create a bar plot with the probabilities
sns.catplot(x="Category", y="Probability", hue="Gender", data=data, kind="b
plt.xlabel("NObeyesdad Category")
plt.ylabel("Probability of Being Overweight")
plt.title("Probability Distribution of Overweight/Obese Classes by Gender")
```

```
data
```

```
If you are male and in category 2 the probability of being overweight is:
0.01865118861962575
If you are male and in category 3 the probability of being overweight is:
0.024053601874965618
If you are male and in category 4 the probability of being overweight is:
0.03035863507896039
If you are male and in category 5 the probability of being overweight is:
0.03886144805965541
If you are male and in category 6 the probability of being overweight is:
0.0001437095151666764
If you are female and in category 2 the probability of being overweight i
s: 0.019098244914439406
If you are female and in category 3 the probability of being overweight i
s: 0.013566339490946612
If you are female and in category 4 the probability of being overweight i
s: 0.024869048716647902
If you are female and in category 5 the probability of being overweight i
s: 0.00026978257976639737
If you are female and in category 6 the probability of being overweight i
s: 0.04753078541702529
```

Out[41]:

	Gender	Category	Probability
0	Male	2	0.018651
1	Male	3	0.024054
2	Male	4	0.030359
3	Male	5	0.038861
4	Male	6	0.000144
5	Female	2	0.019098
6	Female	3	0.013566
7	Female	4	0.024869
8	Female	5	0.000270
9	Female	6	0.047531



Now we see that there are no male samples in the Obesity class 6; reason for misclassification

Best iterations on base models stored

In [42]:
```
{'randomforestclassifier__bootstrap': True,
 'randomforestclassifier__class_weight': 'balanced_subsample',
 'randomforestclassifier__criterion': 'entropy',
 'randomforestclassifier__max_depth': 11,
 'randomforestclassifier__max_features': 'sqrt',
 'randomforestclassifier__max_samples': 0.6,
 'randomforestclassifier__min_impurity_decrease': 0,
 'randomforestclassifier__min_samples_leaf': 1,
 'randomforestclassifier__min_samples_split': 2,
 'randomforestclassifier__n_estimators': 457,
 'randomforestclassifier__oob_score': True,
 'randomforestclassifier__warm_start': True}
0.7369708356012853

{'randomforestclassifier__bootstrap': True,
 'randomforestclassifier__class_weight': 'balanced_subsample',
 'randomforestclassifier__criterion': 'entropy',
 'randomforestclassifier__max_depth': 11,
 'randomforestclassifier__max_features': 'sqrt',
 'randomforestclassifier__max_samples': 0.5,
 'randomforestclassifier__min_impurity_decrease': 1e-08,
 'randomforestclassifier__min_samples_leaf': 1,
 'randomforestclassifier__min_samples_split': 2,
 'randomforestclassifier__n_estimators': 457,
 'randomforestclassifier__oob_score': True,
 'randomforestclassifier__warm_start': True}
0.7363791196249538

{'randomforestclassifier__bootstrap': True,
 'randomforestclassifier__class_weight': 'balanced_subsample',
 'randomforestclassifier__criterion': 'entropy',
 'randomforestclassifier__max_depth': 11,
 'randomforestclassifier__max_features': 'sqrt',
 'randomforestclassifier__max_samples': 0.6,
 'randomforestclassifier__min_impurity_decrease': 0,
 'randomforestclassifier__min_samples_leaf': 1,
 'randomforestclassifier__min_samples_split': 2,
 'randomforestclassifier__n_estimators': 457,
 'randomforestclassifier__oob_score': True,
 'randomforestclassifier__warm_start': True}
0.7369708356012853


{'randomforestclassifier__bootstrap': True,
 'randomforestclassifier__class_weight': {0: 1.0,
  1: 1.15,
  5: 1.2999999999999998,
  6: 1.2999999999999998,
  2: 1.15,
  3: 1.15,
  4: 1.2999999999999998},
 'randomforestclassifier__criterion': 'entropy',
 'randomforestclassifier__max_depth': 11,
 'randomforestclassifier__max_features': 'sqrt',
 'randomforestclassifier__max_samples': 0.6,
 'randomforestclassifier__min_impurity_decrease': 0,
 'randomforestclassifier__min_samples_leaf': 1,
```

```
 'randomforestclassifier__min_samples_split': 2,
 'randomforestclassifier__n_estimators': 456,
 'randomforestclassifier__oob_score': True,
 'randomforestclassifier__warm_start': True}
0.7944392744894914

{'randomforestclassifier__bootstrap': True,
 'randomforestclassifier__class_weight': {0: 1.2000000000000002,
  2: 1.0,
  3: 0.8,
  1: 1.0,
  5: 0.6,
  6: 0.6,
  4: 1.2000000000000002},
 'randomforestclassifier__criterion': 'entropy',
 'randomforestclassifier__max_depth': 11,
 'randomforestclassifier__max_features': 'sqrt',
 'randomforestclassifier__max_samples': 0.6,
 'randomforestclassifier__min_impurity_decrease': 0,
 'randomforestclassifier__min_samples_leaf': 1,
 'randomforestclassifier__min_samples_split': 2,
 'randomforestclassifier__n_estimators': 456,
 'randomforestclassifier__oob_score': True,
 'randomforestclassifier__warm_start': True}
0.7956156831071233

y_train.value_counts()

{'randomforestclassifier__bootstrap': True,
 'randomforestclassifier__class_weight': {0: 1.2000000000000002,
  2: 1.0,
  3: 0.8,
  1: 1.0,
  5: 0.6,
  6: 0.6,
  4: 1.2000000000000002},
 'randomforestclassifier__criterion': 'entropy',
 'randomforestclassifier__max_depth': 11,
 'randomforestclassifier__max_features': 'sqrt',
 'randomforestclassifier__max_samples': 0.6,
 'randomforestclassifier__min_impurity_decrease': 0,
 'randomforestclassifier__min_samples_leaf': 1,
 'randomforestclassifier__min_samples_split': 2,
 'randomforestclassifier__n_estimators': 456,
 'randomforestclassifier__oob_score': True,
 'randomforestclassifier__warm_start': True}
0.7956156831071233


{'randomforestclassifier__bootstrap': True,
 'randomforestclassifier__class_weight': {0: 1.2000000000000002,
  1: 1.0,
  2: 1.0,
  3: 0.8,
  4: 1.2000000000000002,
  5: 0.6,
  6: 0.6},
```

```
 'randomforestclassifier__criterion': 'entropy',
 'randomforestclassifier__max_depth': 11,
 'randomforestclassifier__max_features': 'sqrt',
 'randomforestclassifier__max_samples': 0.6,
 'randomforestclassifier__min_impurity_decrease': 0,
 'randomforestclassifier__min_samples_leaf': 1,
 'randomforestclassifier__min_samples_split': 2,
 'randomforestclassifier__n_estimators': 456,
 'randomforestclassifier__oob_score': True,
 'randomforestclassifier__warm_start': True}
0.7956156831071233
```

Out[42]: 0.7956156831071233