ML

Obesity in focus

Data for healthier lives

Group7

Date

Abdul Rehman Khan 20231738

Duarte Luís Cardoso da Silva Oliveira 20231587

João Pedro Alonso Patrício 20231645

Srijan Dahal 20231738

Valentina Romeo 20241261

NOVA Information Management School

Instituto Superior de Estatística e Gestão de Informação

Universidade Nova de Lisboa

ABSTRACT

The prevalence of obesity has become a significant public health concern worldwide, necessitating effective tools for early detection and intervention. This study explores the use of machine learning techniques to determine obesity levels based on several features provided. The provided dataset was preprocessed to scale, handle missing values, standardize measurements and ensure balanced class distributions.

A range of machine learning models, including [Logistic Regression, Decision Trees, Random Forests, Naïve-bayes, K-nearest Neighbors (KNN), Multi-Layer Perceptron, Gradient Boosting, Stacking, AdaBoost, Bagging], were trained and evaluated. Each model was also evaluated to find out the best possible set of features for performing predictions later. Cross-validation techniques were employed to optimize model performance and avoid overfitting. The results indicated that Random Forest model achieved the highest accuracy of 98.4%, with notable improvements in precision and recall across all obesity categories. But the model used for predicting the test dataset differs. The model used is tuned "Bagging Classifier" which closely followed with a validation accuracy of 97.9%, showcasing the effectiveness of ensemble methods when combined with hyperparameter tuning

The findings demonstrate that machine learning can effectively classify obesity levels based on readily available data, offering a scalable solution for early detection and targeted interventions. However, the analysis also revealed limitations, the model giving the best accuracy does not compulsorily have to be the model that is to be deployed to that test dataset.

In conclusion, this work underscores the potential of machine learning to enhance public health initiatives by enabling efficient and accurate obesity assessment. Integrating these techniques into healthcare systems could facilitate timely intervention strategies, ultimately reducing the burden of obesity on individuals and society.

KEYWORDS

Machine Learning; Top Features 2

INTRODUCTION

According to the World Health Organization (WHO), global obesity has nearly tripled since 1975. As of recent data, more than **13% of the world's adult population** (about 650 million adults) is classified as obese. By glancing upon these staggering numbers, it is evident that obesity is an exponentially rising global public health challenge which needs to be addressed. There is an urgent need for methods and tools that can predict and assess obesity risks based on an individual's lifestyle, physical condition and other social factors.

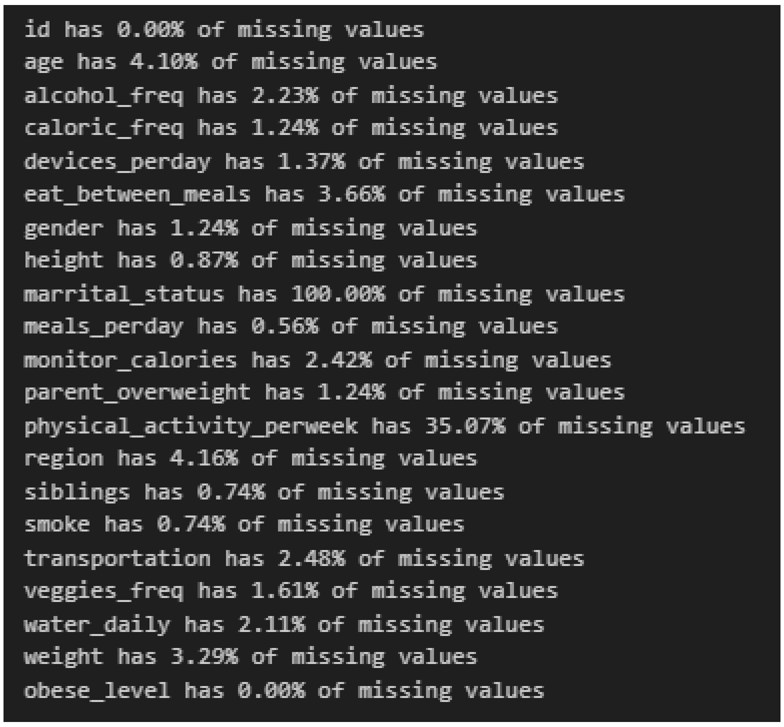
This project aims to develop a machine learning model that predicts obesity levels, providing unique insights and perspectives that could help address this global endemic. We will focus on exploring, analyzing and modeling a dataset containing given attributes of a set of individuals, so that we can understand how specific lifestyle choices, physical condition and social factors contribute to obesity.

DATA EXPLORATION

During the data exploration phase, we focused on identifying key patterns and issues in the dataset that would shape our subsequent data preprocessing and modeling steps.

MISSING VALUES

Upon initial examination of the dataset, we were quickly able to notice that the Column "***marital_status***" was entirely filled with missing values. As it was of no use or significance to help with the prediction, we decided to drop it later in the proccess.



```
id has 0.00% of missing values
age has 4.10% of missing values
alcohol_freq has 2.23% of missing values
caloric_freq has 1.24% of missing values
devices_perday has 1.37% of missing values
eat_between_meals has 3.66% of missing values
gender has 1.24% of missing values
height has 0.87% of missing values
marrital_status has 100.00% of missing values
meals_perday has 0.56% of missing values
monitor_calories has 2.42% of missing values
parent_overweight has 1.24% of missing values
physical_activity_perweek has 35.07% of missing values
region has 4.16% of missing values
siblings has 0.74% of missing values
smoke has 0.74% of missing values
transportation has 2.48% of missing values
veggies_freq has 1.61% of missing values
water_daily has 2.11% of missing values
weight has 3.29% of missing values
obese_level has 0.00% of missing values
```

Moreover, we observed that the variable ***"Physical_activity_perweek"*** has 35.07% missing values which we thought would need further exploration to understand how to deal with it.

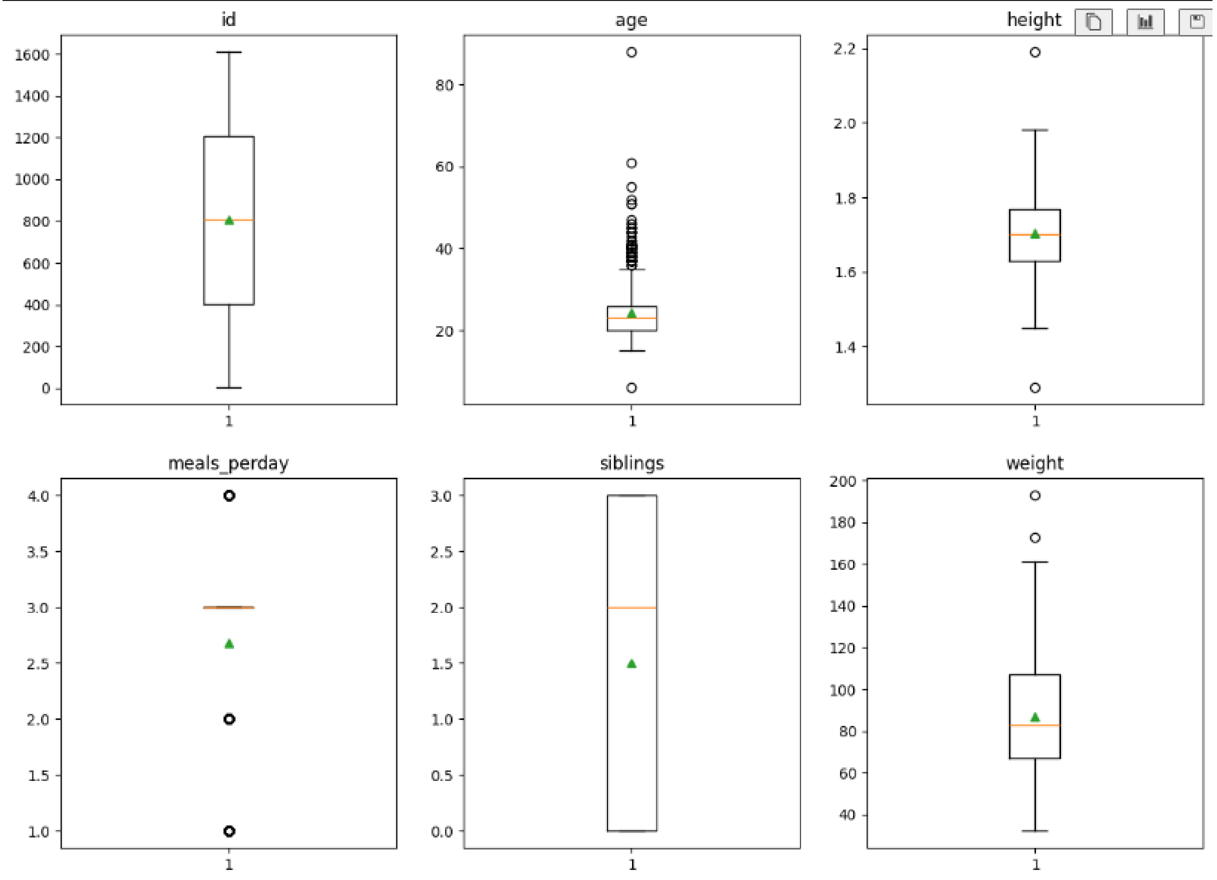
For the remaining variables, the proportion of missing values was relatively low, ranging from 0.56% to 4.10%. Since these values were of a very small percentages, we decided to address them later through imputation techniques, as they would not significantly impact our analyses.

OUTLIER DETECTION AND VARIANCE

Next, we decided to detect outliers and see the overall variability of the variables. To do so, we separated them into numerical and categorical variables to better understand the visualizations.

NUMERICAL VARIABLES

We used Boxplots to detect the outliers and analyze the variables.



Key Insights and Takeaways:

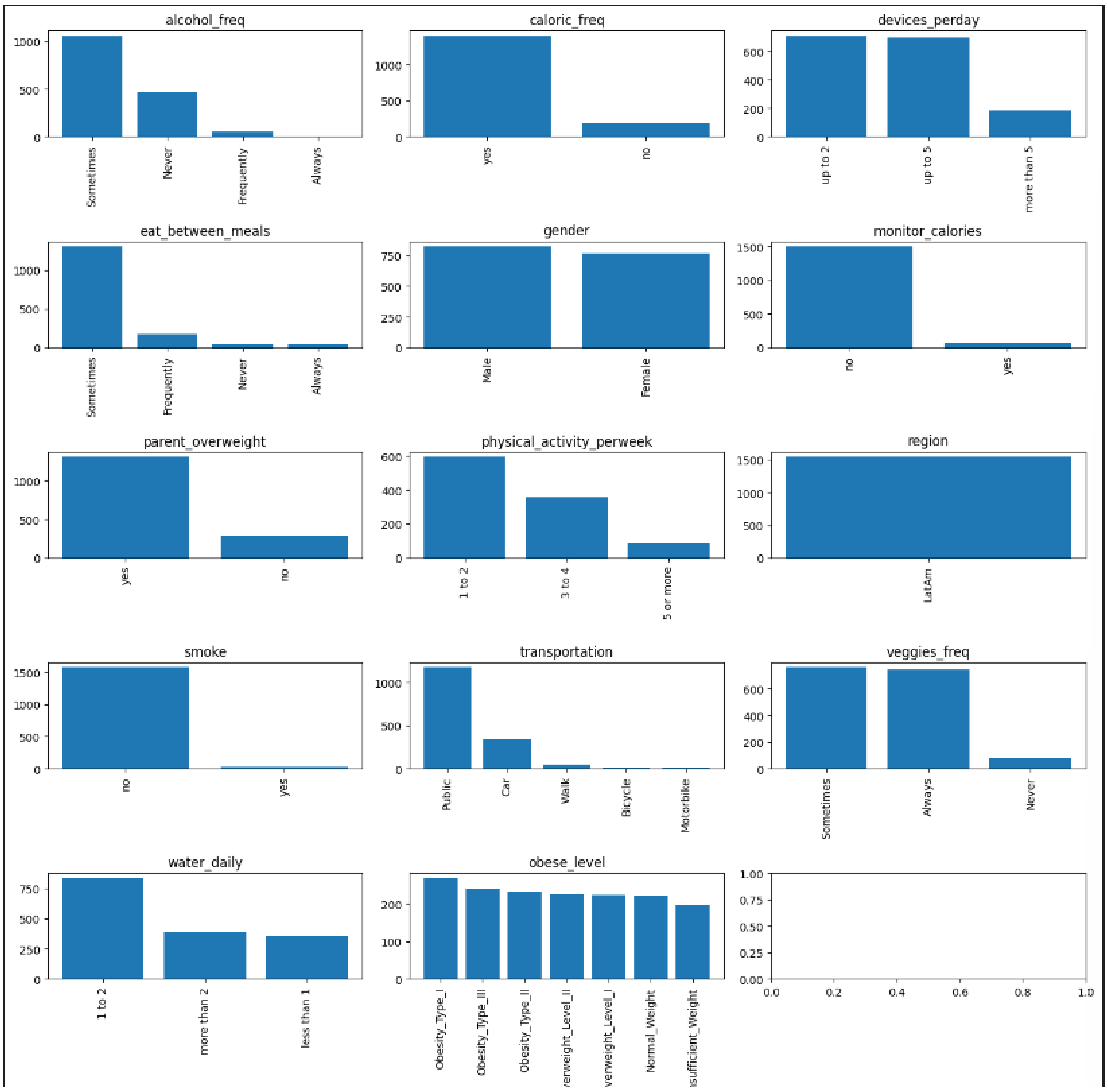
1. **Outliers**: The "age," "height," "meals_perday," and "weight" features exhibit visible outliers. Proper treatment (e.g., capping or removal) may be necessary to improve model performance.

2. **Variability**: Features like "weight" show higher variability as the data is more spread out and can be complemented by the histogram in the annex, while "meals_perday" has a small IQR, which could be explained by the common custom around the world of having three meals per day.

3. **Skewness**: The "age" and "weight" graphs suggest skewness, with data clustering toward lower values and extreme outliers toward higher values. The skewness can be better visualized by the histograms given in the annexes.

Categorical Variables

To understand categorical variables, we used bar charts.



Key Insights and Takeaways:

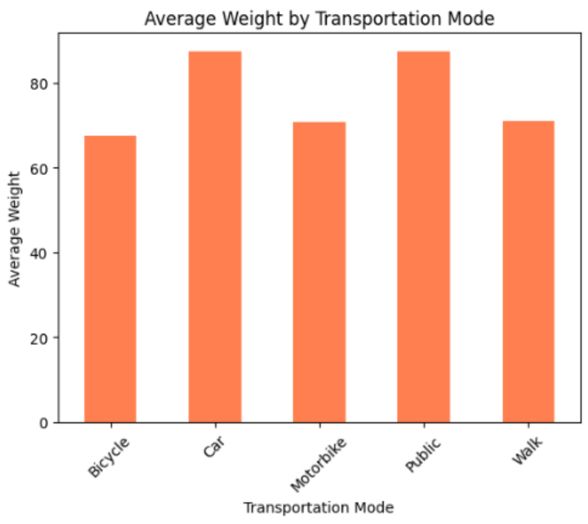
1. **Health Awareness**:
   - Most respondents do not monitor their calories or exercise frequently, reflecting low health awareness.
2. **Dietary Patterns**:
   - Snacking and inconsistent vegetable consumption are in great numbers.

3. **Lifestyle Risks**:
   - Limited physical activity, reliance on public transport, and extended device usage may foster sedentary behavior.

4. **Region-Specific**:
   - As all respondents are from Latin America, insights will surely be regionally relevant, but we may have some issues extrapolating conclusions to a global scenario.

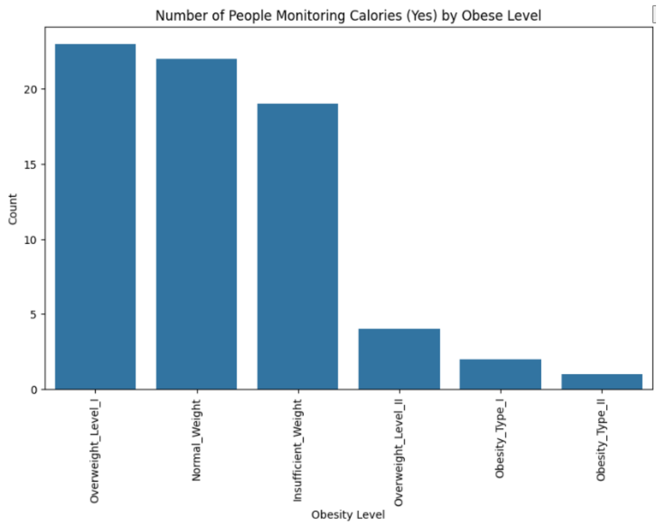
CROSS-VARIABLE TREND DETECTION

Average weight by transportation mode



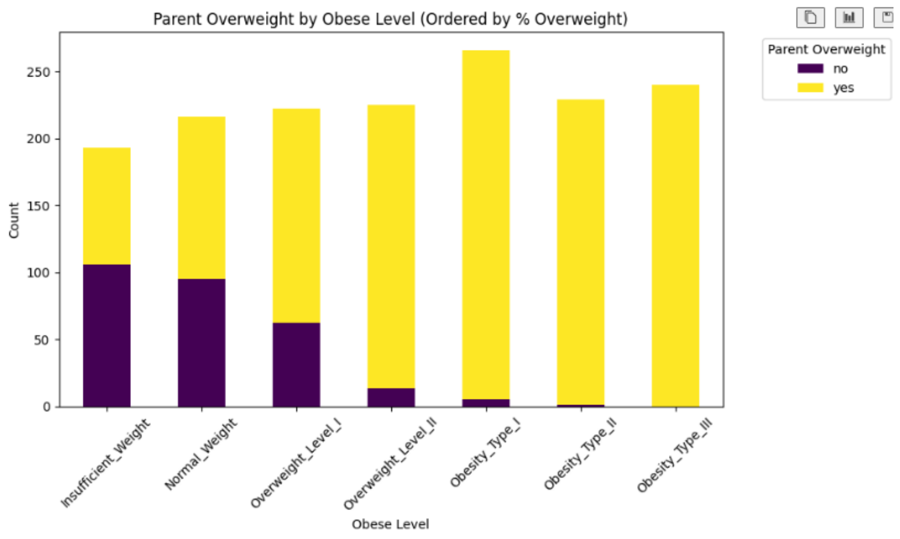
Here we can observe that the average weight of an individual using passive transportation such as public transport and/or car is higher than those who opt for a more active mode of transportation such as bicycle or walking.

PEOPLE MONITORING CALORIES BY OBESITY LEVEL



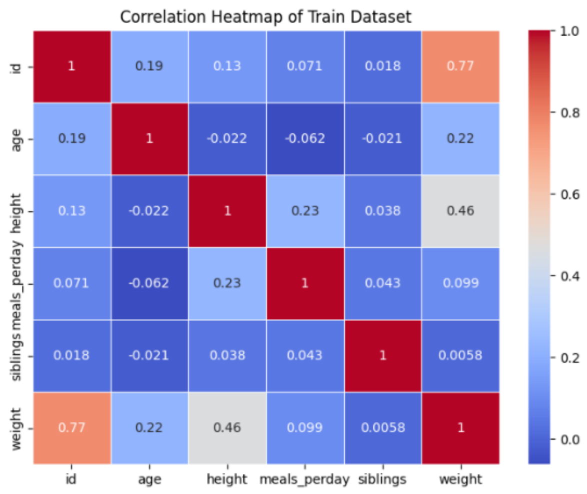
The graph highlights that individuals in the lower weight brackets are more consistent in tracking their caloric intake compared to those in higher weight categories. This trend suggests that dietary awareness and monitoring habits may play a pivotal role in maintaining a healthier weight. Conversely, the lack of calorie tracking observed among individuals in the overweight or obese categories could indicate a lower awareness of their dietary intake or a diminished focus on health-conscious behaviors. This disparity might stem from a combination of factors, such as limited nutritional education, lifestyle constraints, or even psychological barriers to monitoring dietary habits. Furthermore, it raises the possibility that while dietary awareness is essential for weight management, its absence might contribute to unregulated eating patterns, leading to progressive weight gain. The results underscore the importance of fostering widespread nutritional literacy and encouraging regular dietary self-assessment as preventative measures against obesity.

IS OBESITY HEREDITARY OR A LIFESTYLE EFFECT?



The data reveals a clear trend indicating that individuals with overweight or obese parents are more likely to experience obesity themselves. This correlation can be attributed to a combination of hereditary and environmental factors. On one hand, genetic predisposition may play a significant role, as research has shown that certain genetic markers can increase susceptibility to weight gain and metabolic conditions. On the other hand, the influence of parental lifestyle cannot be overlooked. Children often inherit their parents’ eating habits, activity levels, and overall attitudes toward health, which can either mitigate or exacerbate their risk of obesity. For instance, a sedentary household environment or unhealthy dietary practices may create a cycle of poor health behaviors that perpetuates obesity across generations. This dual influence underscores the complex interplay between nature and nurture in determining obesity outcomes, highlighting the need for interventions that address both genetic risk factors and lifestyle education within families.

Variable correlation



The analysis of variable correlations indicates that there are no strong relationships among the variables in the dataset. This lack of high correlation suggests that the variables do not exhibit multicollinearity, which is favorable for model performance. However, it also implies that correlation alone does not provide significant guidance for feature selection, as none of the variables show a dominant linear relationship with the target variable. This finding necessitates the use of alternative feature selection methods, such as evaluating feature importance through machine learning algorithms, to identify which variables contribute most meaningfully to predicting obesity levels.

Model Assessment

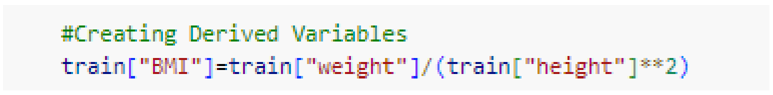
Model assessment was a very important part of our project. We decided to use k-fold cross-validation, since it is the more robust way of evaluating what are the best values

Data Preprocessing

Due to the use of k-fold, Data Preprocessing was divided into two parts, what we could do outside the folds, and the part that would have to be done inside, namely scaling and imputation.

DERIVED VARIABLES

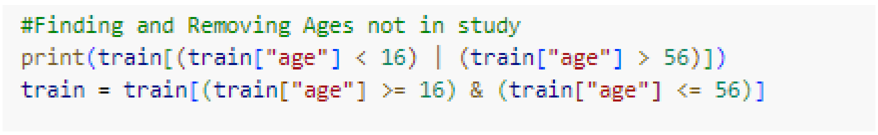
We only created one derived variable, the BMI. This is a very important metric when dealing with weight and its classification. The formula for BMI is weight/(height^2)



```
#Creating Derived Variables
train["BMI"]=train["weight"]/(train["height"]**2)
```

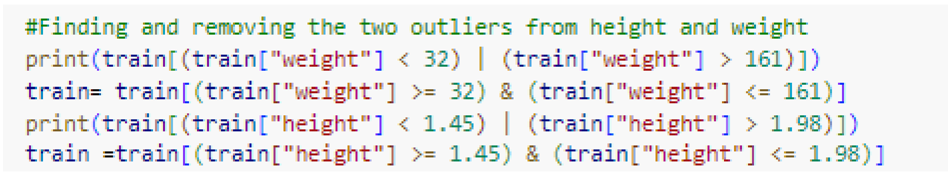
OUTLIERS

From the information provided in the directives, we knew that the participants' ages were between 16 and 56, so those outliers were easy to remove.



```
#Finding and Removing Ages not in study
print(train[(train["age"] < 16) | (train["age"] > 56)])
train = train[(train["age"] >= 16) & (train["age"] <= 56)]
```

As for the outliers observed in the boxplots, we decided to remove the 2 in "**height**", and the 2 in "**weight**". We decided to keep the ones from "**meals_per_day**", since we did not consider those actual outliers.



```
#Finding and removing the two outliers from height and weight
print(train[(train["weight"] < 32) | (train["weight"] > 161)])
train= train[(train["weight"] >= 32) & (train["weight"] <= 161)]
print(train[(train["height"] < 1.45) | (train["height"] > 1.98)])
train =train[(train["height"] >= 1.45) & (train["height"] <= 1.98)]
```

ENCODING

Data encoding is a critical step in preparing the dataset for machine learning techniques, as most algorithms require numerical input. Our encoding strategy was designed to handle different types of categorical and ordinal data in a systematic and logical manner.

We did encoding outside the k-folds, since it was the same process for every observation. There were three types of encoding: encoding ordinal columns and binary columns, and creating dummy variables for the others. Most of our columns were either ordinal or binary, the only one that needed dummy variables was "**transportation**". We also did mapping, attributing values to the columns "**devices_per_day**" and "**physical_activity_per_week**".

1.Encoding Ordinal Columns

Ordinal data has a natural order, so it is essential to preserve this order during encoding. The following columns were encoded as ordinal data:

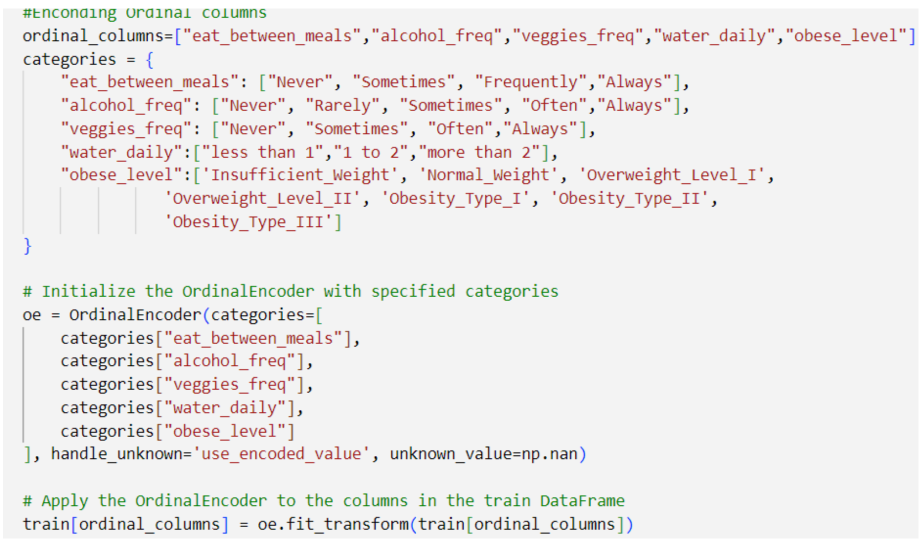
- eat_between_meals;
- alcohol_freq;
- veggies_freq;
- water_daily;
- obese_level.

We specified the categories explicitly for each column to ensure the encoding reflects the natural ordering of the values. For example:

- **obese_level**: Encoded from "Insufficient_Weight" (lowest level) to "Obesity_Type_III"(highest level).

- **water_daily**: Encoded from "less than 1" (lowest intake) to "more than 2" (highest intake).

The OrdinalEncoder was used with the handle_unknown= "use_encoded_value" and unknown_value=np.nan parameters to ensure robustness in handling unexpected or missing values.



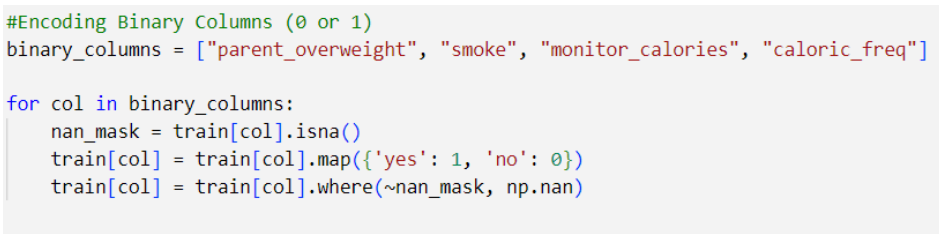
```
#Encoding Ordinal columns
ordinal_columns=["eat_between_meals","alcohol_freq","veggies_freq","water_daily","obese_level"]
categories = {
    "eat_between_meals": ["Never", "Sometimes", "Frequently","Always"],
    "alcohol_freq": ["Never", "Rarely", "Sometimes", "Often","Always"],
    "veggies_freq": ["Never", "Sometimes", "Often","Always"],
    "water_daily":["less than 1","1 to 2","more than 2"],
    "obese_level":['Insufficient_Weight', 'Normal_Weight', 'Overweight_Level_I',
                   'Overweight_Level_II', 'Obesity_Type_I', 'Obesity_Type_II',
                   'Obesity_Type_III']
}

# Initialize the OrdinalEncoder with specified categories
oe = OrdinalEncoder(categories=[
    categories["eat_between_meals"],
    categories["alcohol_freq"],
    categories["veggies_freq"],
    categories["water_daily"],
    categories["obese_level"]
], handle_unknown='use_encoded_value', unknown_value=np.nan)

# Apply the OrdinalEncoder to the columns in the train DataFrame
train[ordinal_columns] = oe.fit_transform(train[ordinal_columns])
```

2.Binary Columns

Binary columns were mapped to 0 and 1:

- Columns like parent_overweight, smoke, monitor_calories, and caloric_freq had values such as 'yes' and 'no', which were mapped to 1 and 0 respectively.
- Missing values were preserved by creating a mask before mapping and reapplying the NaN values afterward.

This approach maintains the interpretability of the binary variables while handling missing data effectively.



```
#Encoding Binary Columns (0 or 1)
binary_columns = ["parent_overweight", "smoke", "monitor_calories", "caloric_freq"]

for col in binary_columns:
    nan_mask = train[col].isna()
    train[col] = train[col].map({'yes': 1, 'no': 0})
    train[col] = train[col].where(~nan_mask, np.nan)
```

3. Encoding Columns with Assignable Numeric Values

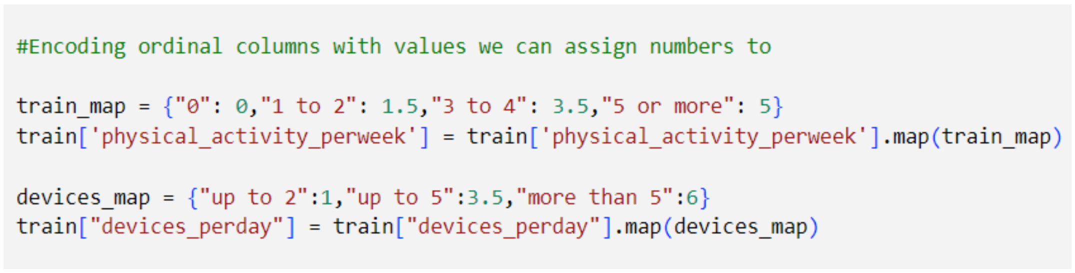
Certain columns had categories with values that could logically be assigned numeric equivalents:

- **physical_activity_perweek**: Mapped as:
  - "0" → 0;
  - "1 to 2" → 1.5;
  - "3 to 4" → 3.5;

    - “5 or more”→ 5.

- **devices_perday**: Mapped as:
    - “up to 2” → 1;
    - “up to 5” → 3.5;
    - “more than 5” → 6.

These mappings were designed to reflect the approximate midpoint or logical progression of the categories, ensuring numerical comparability.



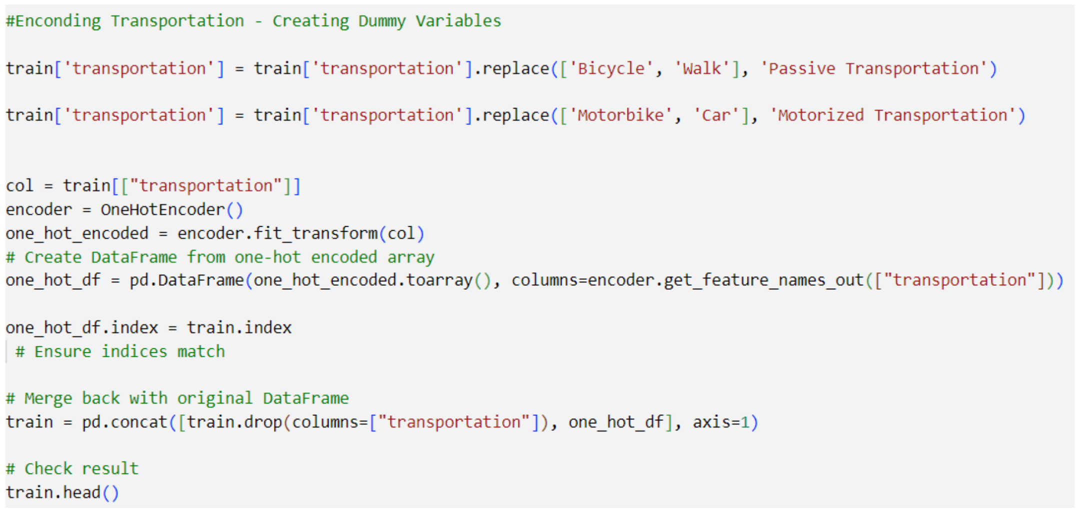
```
#Encoding ordinal columns with values we can assign numbers to

train_map = {"0": 0,"1 to 2": 1.5,"3 to 4": 3.5,"5 or more": 5}
train['physical_activity_perweek'] = train['physical_activity_perweek'].map(train_map)

devices_map = {"up to 2":1,"up to 5":3.5,"more than 5":6}
train["devices_perday"] = train["devices_perday"].map(devices_map)
```

4. One-Hot Encoding for Transportation

For the transportation column, we used one-hot encoding:

- Categories were grouped into broader classes:
    - “Bicycle' and 'Walk” → “Passive Transportation”;
    - “Motorbike” and 'Car' → “Motorized Transportation”;
    - “Public” → “Public Transportation”.

A OneHotEncoder was applied to create separate binary columns for each category. This ensures the transportation variable is treated as independent features without imposing an arbitrary order, preserving its nominal nature. One-hot encoding avoids biases and enhances compatibility with machine learning algorithms. It ensures no information is lost, making the categories easier to interpret and more effective for prediction. By capturing relationships between transportation modes, the method improves model performance while supporting fair and accurate data representation.



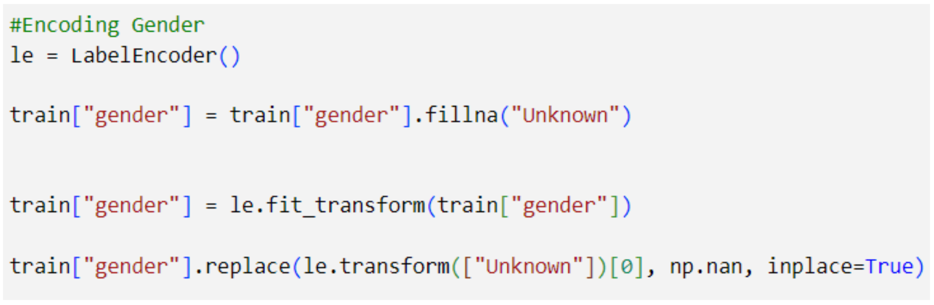
```
#Enconding Transportation - Creating Dummy Variables

train['transportation'] = train['transportation'].replace(['Bicycle', 'Walk'], 'Passive Transportation')

train['transportation'] = train['transportation'].replace(['Motorbike', 'Car'], 'Motorized Transportation')


col = train[["transportation"]]
encoder = OneHotEncoder()
one_hot_encoded = encoder.fit_transform(col)
# Create DataFrame from one-hot encoded array
one_hot_df = pd.DataFrame(one_hot_encoded.toarray(), columns=encoder.get_feature_names_out(["transportation"]))

one_hot_df.index = train.index
 # Ensure indices match

# Merge back with original DataFrame
train = pd.concat([train.drop(columns=["transportation"]), one_hot_df], axis=1)

# Check result
train.head()
```

5. Encoding Gender

The gender column was encoded using a LabelEncoder:

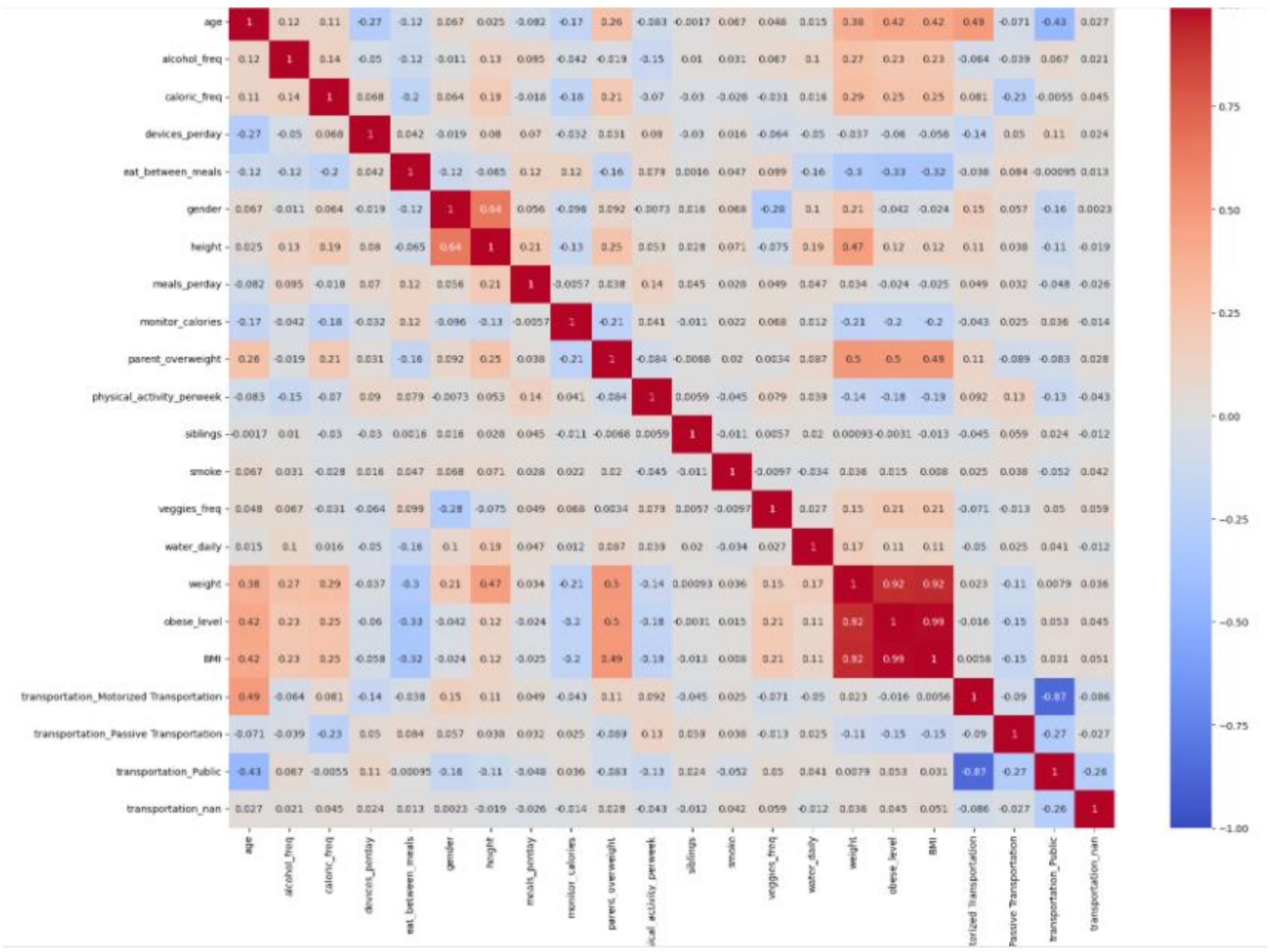
- Missing values were initially filled with "Unknown";
- The LabelEncoder assigned numerical values to the categories, but "Unknown" was replaced back with NaN to preserve missing data consistency.

After encoding, the "Male" gender will be assigned a value of 1 in the dataset, while the "Female" gender will typically be assigned a value of 0. The LabelEncoder automatically converts the unique categories into numerical labels.



```
#Encoding Gender
le = LabelEncoder()

train["gender"] = train["gender"].fillna("Unknown")


train["gender"] = le.fit_transform(train["gender"])

train["gender"].replace(le.transform(["Unknown"])[0], np.nan, inplace=True)
```

After the encoding, we can now check the new correlations.



We know have high correlations, for “**BMI**”, “**weight**” and “**Parent_overweight**” relating with the dataset.

Scaling

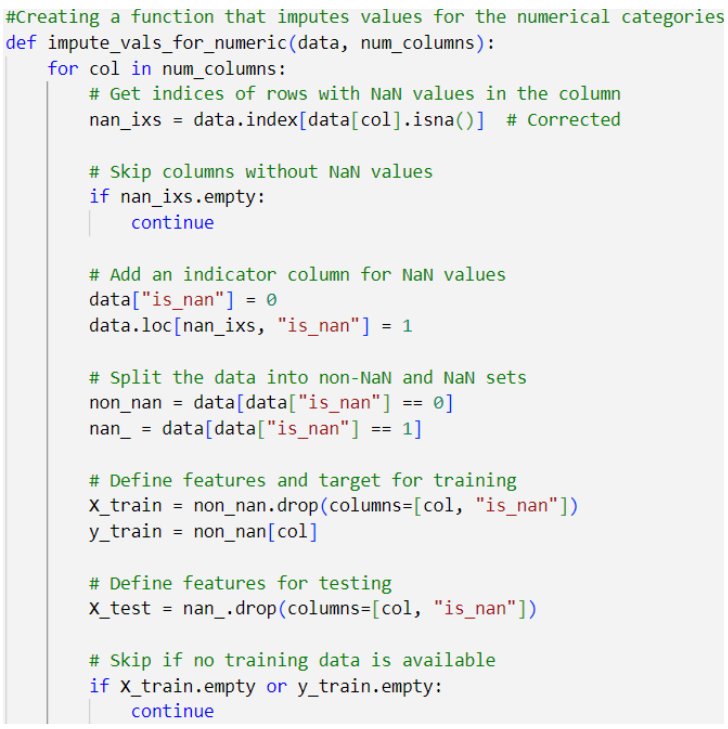
Scaling is crucial because it ensures that the features are on the same scale, preventing models from being biased toward variables with larger numerical ranges. We decided to use the **MinMax Scaler** because our dataset didn't have many outliers, and the scaling method ensures that all features are scaled to a uniform range, typically between 0 and 1.

We then created a function that applies the **MinMax scaling** to both the training and validation datasets, which will be used inside the K-fold cross-validation method. This ensures that the data is consistently scaled during each fold, allowing for fair and efficient model evaluation across different subsets of the dataset.

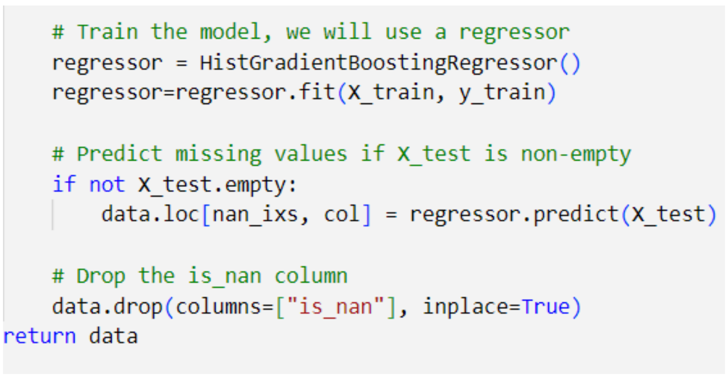
```python
#Creating a function that Scales train and Validation
def min_max_scale(train_for_scale, val_for_scale):
    min_max_scaler= MinMaxScaler().fit(train_for_scale)
    train_min_max_scaled=min_max_scaler.transform(train_for_scale)
    train_min_max_scaled=pd.DataFrame(train_min_max_scaled, columns=train_for_scale.columns).set_index(train_for_scale.index)
    val_min_max_scaled=min_max_scaler.transform(val_for_scale)
    val_min_max_scaled=pd.DataFrame(val_min_max_scaled, columns=val_for_scale.columns).set_index(val_for_scale.index)
    return train_min_max_scaled,val_min_max_scaled
```

Missing Values

To address the missing values we created a function “impute_vals_for_numeric” that imputes missing values in numerical columns using a regression-based approach, which was also applied during the K-fold cross-validation process. For each column with missing values, the function first identifies the rows with NaNs and creates an indicator column to mark them. The data is then split into two subsets: one containing rows with valid values and another with missing values. The valid data is used to train a **HistGradientBoostingRegressor**, which learns the relationship between the features and the target column. The trained model then predicts the missing values, which are used to fill in the NaNs. Finally, the temporary indicator column is dropped, and the updated dataset with imputed values is returned. By incorporating this imputation method in the K-fold cross-validation, we ensure that the missing values are properly handled during model evaluation, making the process more reliable and accurate. This method provides contextually appropriate imputations based on data patterns, ensuring better results.



```
#Creating a function that imputes values for the numerical categories
def impute_vals_for_numeric(data, num_columns):
    for col in num_columns:
        # Get indices of rows with NaN values in the column
        nan_ixs = data.index[data[col].isna()]  # Corrected

        # Skip columns without NaN values
        if nan_ixs.empty:
            continue

        # Add an indicator column for NaN values
        data["is_nan"] = 0
        data.loc[nan_ixs, "is_nan"] = 1

        # Split the data into non-NaN and NaN sets
        non_nan = data[data["is_nan"] == 0]
        nan_ = data[data["is_nan"] == 1]

        # Define features and target for training
        X_train = non_nan.drop(columns=[col, "is_nan"])
        y_train = non_nan[col]

        # Define features for testing
        X_test = nan_.drop(columns=[col, "is_nan"])

        # Skip if no training data is available
        if X_train.empty or y_train.empty:
            continue
```



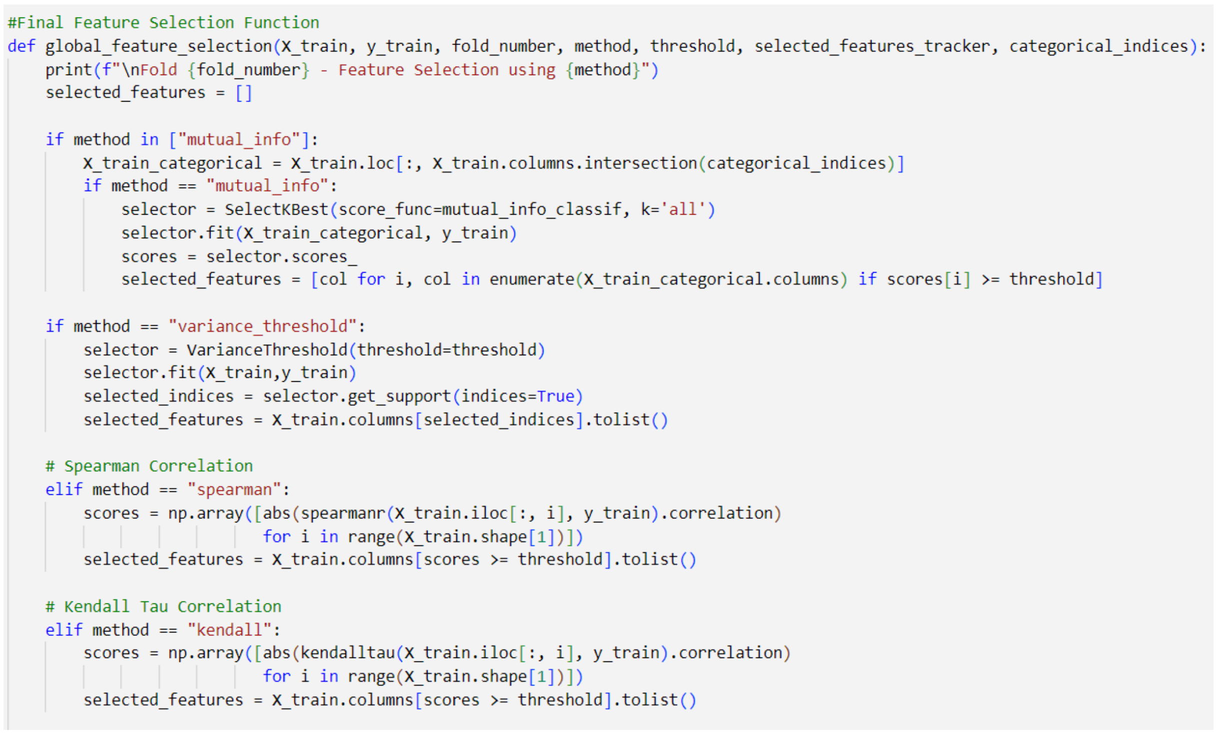
```
    # Train the model, we will use a regressor
    regressor = HistGradientBoostingRegressor()
    regressor=regressor.fit(X_train, y_train)

    # Predict missing values if X_test is non-empty
    if not X_test.empty:
        data.loc[nan_ixs, col] = regressor.predict(X_test)

    # Drop the is_nan column
    data.drop(columns=["is_nan"], inplace=True)
return data
```

Feature Selection

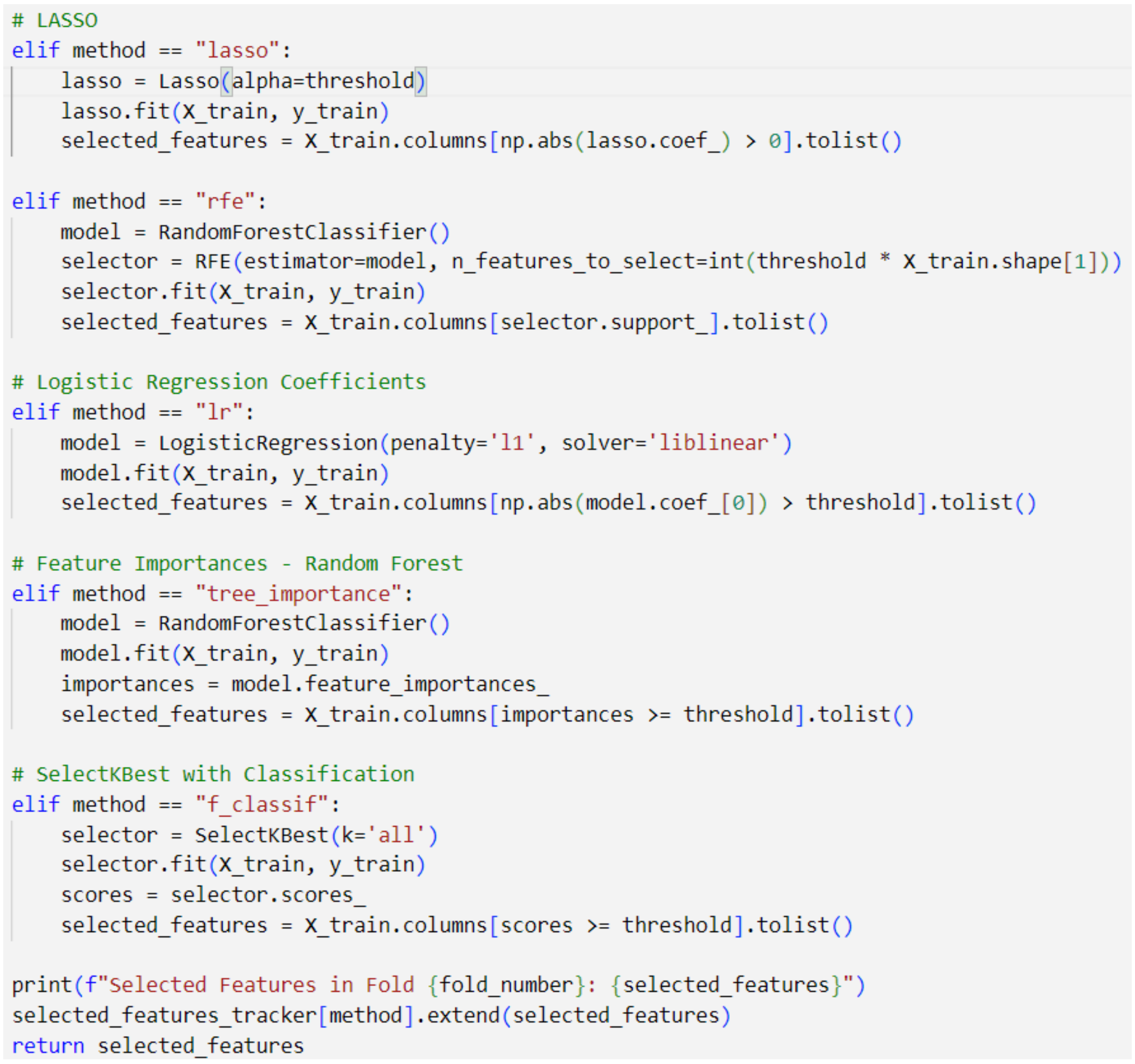
The function global_feature_selection performs feature selection on the training data using different methods and tracks the selected features across multiple folds. It starts by checking the selected method and applying the appropriate feature selection technique.

For categorical data, the function handles methods like "mutual_info" specifically by selecting categorical columns and applying the method's criteria for selecting important features. For methods like "variance_threshold", "spearman", and "kendall", it calculates scores for each feature or correlation and compares them to the threshold to select relevant features. In "lasso", "rfe", and "lr" ,the function uses model-based feature selection techniques, including penalization and recursive elimination, to identify important features. "tree_importance" uses the random forest model to compute feature importance scores, while "f_classif" applies SelectKBest to select features based on statistical significance.

Finally, the function prints the selected features for each fold and appends them to a tracker for feature selection, which keeps track of the selected features across all methods and folds.

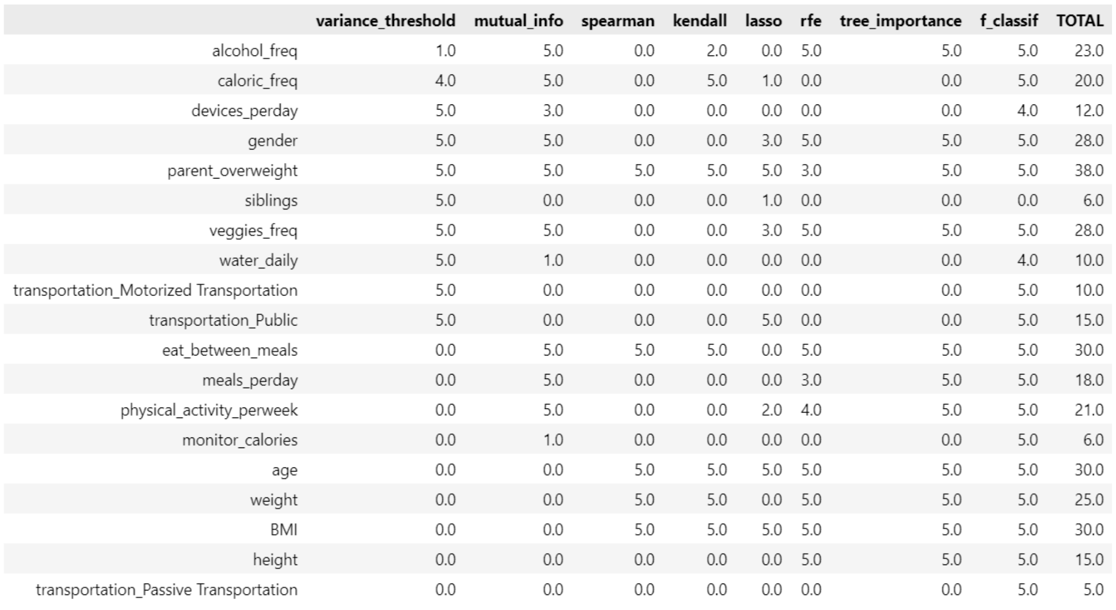


```python
#Final Feature Selection Function
def global_feature_selection(X_train, y_train, fold_number, method, threshold, selected_features_tracker, categorical_indices):
    print(f"\nFold {fold_number} - Feature Selection using {method}")
    selected_features = []

    if method in ["mutual_info"]:
        X_train_categorical = X_train.loc[:, X_train.columns.intersection(categorical_indices)]
        if method == "mutual_info":
            selector = SelectKBest(score_func=mutual_info_classif, k='all')
            selector.fit(X_train_categorical, y_train)
            scores = selector.scores_
            selected_features = [col for i, col in enumerate(X_train_categorical.columns) if scores[i] >= threshold]

    if method == "variance_threshold":
        selector = VarianceThreshold(threshold=threshold)
        selector.fit(X_train,y_train)
        selected_indices = selector.get_support(indices=True)
        selected_features = X_train.columns[selected_indices].tolist()

    # Spearman Correlation
    elif method == "spearman":
        scores = np.array([abs(spearmanr(X_train.iloc[:, i], y_train).correlation)
                           for i in range(X_train.shape[1])])
        selected_features = X_train.columns[scores >= threshold].tolist()

    # Kendall Tau Correlation
    elif method == "kendall":
        scores = np.array([abs(kendalltau(X_train.iloc[:, i], y_train).correlation)
                           for i in range(X_train.shape[1])])
        selected_features = X_train.columns[scores >= threshold].tolist()
```



```
# LASSO
elif method == "lasso":
    lasso = Lasso(alpha=threshold)
    lasso.fit(X_train, y_train)
    selected_features = X_train.columns[np.abs(lasso.coef_) > 0].tolist()

elif method == "rfe":
    model = RandomForestClassifier()
    selector = RFE(estimator=model, n_features_to_select=int(threshold * X_train.shape[1]))
    selector.fit(X_train, y_train)
    selected_features = X_train.columns[selector.support_].tolist()

# Logistic Regression Coefficients
elif method == "lr":
    model = LogisticRegression(penalty='l1', solver='liblinear')
    model.fit(X_train, y_train)
    selected_features = X_train.columns[np.abs(model.coef_[0]) > threshold].tolist()

# Feature Importances - Random Forest
elif method == "tree_importance":
    model = RandomForestClassifier()
    model.fit(X_train, y_train)
    importances = model.feature_importances_
    selected_features = X_train.columns[importances >= threshold].tolist()

# SelectKBest with Classification
elif method == "f_classif":
    selector = SelectKBest(k='all')
    selector.fit(X_train, y_train)
    scores = selector.scores_
    selected_features = X_train.columns[scores >= threshold].tolist()

print(f"Selected Features in Fold {fold_number}: {selected_features}")
selected_features_tracker[method].extend(selected_features)
return selected_features
```

After performing feature selection across multiple folds, we created a dictionary to track how many times each feature was selected for each method. This count was stored for each method in the features_dict dictionary. The dictionary was then converted into a DataFrame, where each row represents a feature, and columns show the selection counts for each method. A "TOTAL" column was added to summarize the overall count of feature selection across all methods. This approach provides insights into feature importance and consistency across different selection techniques.



	variance_threshold	mutual_info	spearman	kendall	lasso	rfe	tree_importance	f_classif	TOTAL
alcohol_freq	1.0	5.0	0.0	2.0	0.0	5.0	5.0	5.0	23.0
caloric_freq	4.0	5.0	0.0	5.0	1.0	0.0	0.0	5.0	20.0
devices_perday	5.0	3.0	0.0	0.0	0.0	0.0	0.0	4.0	12.0
gender	5.0	5.0	0.0	0.0	3.0	5.0	5.0	5.0	28.0
parent_overweight	5.0	5.0	5.0	5.0	5.0	3.0	5.0	5.0	38.0
siblings	5.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	6.0
veggies_freq	5.0	5.0	0.0	0.0	3.0	5.0	5.0	5.0	28.0
water_daily	5.0	1.0	0.0	0.0	0.0	0.0	0.0	4.0	10.0
transportation_Motorized Transportation	5.0	0.0	0.0	0.0	0.0	0.0	0.0	5.0	10.0
transportation_Public	5.0	0.0	0.0	0.0	5.0	0.0	0.0	5.0	15.0
eat_between_meals	0.0	5.0	5.0	5.0	0.0	5.0	5.0	5.0	30.0
meals_perday	0.0	5.0	0.0	0.0	0.0	3.0	5.0	5.0	18.0
physical_activity_perweek	0.0	5.0	0.0	0.0	2.0	4.0	5.0	5.0	21.0
monitor_calories	0.0	1.0	0.0	0.0	0.0	0.0	0.0	5.0	6.0
age	0.0	0.0	5.0	5.0	5.0	5.0	5.0	5.0	30.0
weight	0.0	0.0	5.0	5.0	0.0	5.0	5.0	5.0	25.0
BMI	0.0	0.0	5.0	5.0	5.0	5.0	5.0	5.0	30.0
height	0.0	0.0	0.0	0.0	0.0	5.0	5.0	5.0	15.0
transportation_Passive Transportation	0.0	0.0	0.0	0.0	0.0	0.0	0.0	5.0	5.0

Then two thresholds were created to identify the most consistent and significant features based on their total selection count across all methods and folds. The threshold for fine_feat6 is set to 25, meaning it includes features selected more than 25 times, while the threshold for fine_feat4 is set to 29, meaning it includes features selected more than 29 times. These thresholds help in selecting features with varying levels of importance and consistency: fine_feat6 captures moderately important features, and fine_feat4 highlights features with the highest consistency in their selection. These lists are useful for narrowing down the most important features for the final model.

Predictive Algorithms

Original set of alghoritms

- "Logistic Regression": LogisticRegression(),
- "DecisionTreeClassifier": DecisionTreeClassifier(),
- "Naive Bayes":GaussianNB(),
- "MLP": MLPClassifier(),
- "KNeighbors": KNeighborsClassifier(),
- "GradientBoost": GradientBoostingClassifier(),
- "RandomForestClassifier": RandomForestClassifier(),
- "Stacking": StackingClassifier(estimators=estimators_stacking,final_estimator=LogisticRegression()),
- "AdaBoost": AdaBoostClassifier(),
- "Bagging":BaggingClassifier().

The analysis involved evaluating a diverse set of algorithms, including logistic regression, decision trees, neural networks, and ensemble methods such as Random Forest, Gradient Boost, and Stacking, to assess their performance on different feature sets. For each model, training and validation scores were computed using all features, the top 4 features, and the top 6 features. The difference between training and validation scores was calculated to identify potential overfitting or underfitting. These results were compiled into a comprehensive summary table, allowing for a comparison of algorithm performance across feature subsets. This approach ensures a thorough evaluation to determine the most effective model and feature selection strategy.

	train_all_feat	val_all_feat	Difference	train_4_feat	val_4_feat	Difference	train_6_feat	val_6_feat	Difference
Logistic Regression	0.783854	0.752861	0.030993	0.716146	0.709535	0.006611	0.712424	0.702109	0.010315
DecisionTreeClassifier	1.000000	0.968848	0.031152	1.000000	0.947874	0.052126	1.000000	0.955318	0.044682
Naive Bayes	0.832272	0.808351	0.023921	0.626779	0.615440	0.011339	0.824645	0.813120	0.011526
MLP	0.928571	0.874061	0.054510	0.865438	0.853749	0.011688	0.863404	0.853094	0.010310
KNeighbors	0.807718	0.696690	0.111027	0.935680	0.898445	0.037235	0.914354	0.873413	0.040941
GradientBoost	1.000000	0.976296	0.023704	1.000000	0.954634	0.045366	1.000000	0.964114	0.035886
RandomForestClassifier	1.000000	0.981040	0.018960	1.000000	0.953280	0.046720	1.000000	0.961404	0.038596
Stacking	0.917232	0.880153	0.037078	0.945328	0.924157	0.021171	0.951591	0.931603	0.019988
AdaBoost	0.439065	0.419805	0.019260	0.439065	0.419805	0.019260	0.439065	0.419805	0.019260
Bagging	0.997292	0.974945	0.022347	0.994922	0.956667	0.038255	0.994923	0.957345	0.037577

Algorithm Optimization

To determine the optimal hyperparameters for all models, we performed grid search tuning on each algorithm's parameter space. The process was repeated multiple times for consistency, storing the results of each iteration. For each model, the mode of the selected hyperparameters across iterations was used to identify the best configuration.

```python
#Some parameters vary with different Grid search runs. Did my best but still can be some errors
models_for_final_optimization_all_features={"Bagging_Tuned":BaggingClassifier(bootstrap=True,max_samples=0.75, n_estimators=75),
                                            "GradientBoost":GradientBoostingClassifier(learning_rate=0.1,n_estimators=100,subsample=1.0),
                                            "RandomForest":RandomForestClassifier(n_estimators=100, bootstrap=False, max_depth=10),
                                            "MLP": MLPClassifier(solver="lbfgs", learning_rate="constant", learning_rate_init=0.1, batch_size=50)
                                            }
#KNEIGHBOR IS WITH 4 FEATURES
models_for_final_optimization_4_features={"KNeighbors":KNeighborsClassifier(algorithm="kd_tree",metric="manhattan", n_neighbors=5)}
```

In the image above we can see the best parameters for each model after the grid search.

In addition to individual model optimization, we explored stacking classifiers to try to leverage the strengths of multiple algorithms. Various combinations of base models, such as Logistic Regression, K-Nearest Neighbors, AdaBoost, Naive Bayes, Gradient Boosting, Random Forest, and Decision Trees, were tested as estimators.

	train_all_feat	val_all_feat	Difference	train_4_feat	val_4_feat	Difference	train_6_feat	val_6_feat	Difference
Bagging_Tuned	0.996446	0.970884	0.025562	NaN	NaN	NaN	NaN	NaN	NaN
GradientBoost	1.000000	0.975621	0.024379	NaN	NaN	NaN	NaN	NaN	NaN
RandomForest	1.000000	0.984423	0.015577	NaN	NaN	NaN	NaN	NaN	NaN
MLP	1.000000	0.949201	0.050799	NaN	NaN	NaN	NaN	NaN	NaN
KNeighbors	NaN	NaN	NaN	0.938557	0.902508	0.036049	NaN	NaN	NaN
LR,KNN,Ada	NaN	NaN	NaN	NaN	NaN	NaN	0.946851	0.914036	0.032815
LR,KNN,NB	NaN	NaN	NaN	NaN	NaN	NaN	0.951591	0.931603	0.019988
LR,Ada,NB	NaN	NaN	NaN	NaN	NaN	NaN	0.994414	0.969533	0.024882
KNN,Ada,NB	NaN	NaN	NaN	NaN	NaN	NaN	0.999323	0.968855	0.030468
nb,BT,GB	NaN	NaN	NaN	NaN	NaN	NaN	0.998307	0.970886	0.027421
nb,BT,RF	NaN	NaN	NaN	NaN	NaN	NaN	0.990013	0.965470	0.024544
GB,RF,DT	NaN	NaN	NaN	NaN	NaN	NaN	0.998815	0.964111	0.034704
BT,GB,DT	NaN	NaN	NaN	NaN	NaN	NaN	0.993907	0.963431	0.030476
BT,RF,DT	NaN	NaN	NaN	NaN	NaN	NaN	0.996107	0.968857	0.027249
BT,GB,RF	NaN	NaN	NaN	NaN	NaN	NaN	0.993907	0.963436	0.030471

RESULTS

1. **Model Performance**:
    - The **Random Forest model** achieved the highest validation accuracy (98.4%).
    - The **tuned Bagging Classifier** followed closely with 97.2%, demonstrating superior generalization and avoiding overfitting compared to other models.
    - Additional models, such as **Gradient Boosting** (97.6%) and **Multi-Layer Perceptron (MLP)** (93.7%), also performed well.
2. **Feature Importance**: The top features contributing to obesity prediction were:
    - **Parental obesity status**
    - **Frequency of eating between meals**
    - **Age**
    - **Body Mass Index (BMI)**
    - **Gender**
    - **Frequency of vegetable intake**
3. **Statistical Insights**:
    - Individuals engaging in regular physical activity (three or more days per week) exhibited lower obesity levels.
    - Daily vegetable consumption and calorie monitoring reduced obesity risks.
4. **Missing Values and Outliers**:
    - The marital_status feature was excluded due to missing data.
    - Outliers in variables such as age, height, and weight were treated to improve data distribution.
5. **Public Health Insights**:
    - Promoting physical activity and dietary awareness can mitigate obesity risks.
    - Family-targeted interventions could yield sustainable outcomes, addressing both hereditary and environmental factors.

DISCUSSION

Key Interpretations

1. **Feature Significance**:
   - Parental obesity status and BMI were pivotal predictors, consistent with findings by Whitaker et al. (1997) and Keys et al. (1972), respectively.
   - Dietary patterns, such as frequency of eating between meals and vegetable intake, further emphasize the role of nutrition in obesity prevention.

Broader Implications

1. **Behavioral Interventions**:
   - Campaigns should focus on fostering healthy eating habits and physical activity, as reinforced by studies like Gordon-Larsen et al. (2006).
   - Nutritional education programs targeting families can address intergenerational influences on obesity, as highlighted by Birch and Davison (2001).

Limitations and Future Directions

1. **Dataset Scope**:
   - The dataset's focus on Latin American individuals (ages 16–56) limits generalizability. Future studies should expand to more diverse populations.
2. **Socioeconomic Factors**:
   - Including variables like income and education could provide a more comprehensive analysis, as suggested by Sobal and Stunkard (1989).
3. **Feature Engineering**:
   - Investigating interactions between variables, such as physical activity and diet, could yield deeper insights (Hu et al., 2001).

Conclusion

This study demonstrates the potential of machine learning in obesity prediction, emphasizing the importance of behavioral and family-centered interventions. While the Random Forest model excelled in accuracy, the Bagging Classifier offered a balanced approach, making it more applicable for real-world settings. Addressing the limitations identified will further enhance the scope and impact of such studies in combating obesity on a global scale.

REFERENCES

1. Whitaker, R.C., Wright, J.A., Pepe, M.S., Seidel, K.D., & Dietz, W.H. (1997). Predicting obesity in young adulthood from childhood and parental obesity. *New England Journal of Medicine*, 337(13), 869-873.
2. Keys, A., Fidanza, F., Karvonen, M.J., Kimura, N., & Taylor, H.L. (1972). Indices of relative weight and obesity. *Journal of Chronic Diseases*, 25(6), 329-343.
3. Gordon-Larsen, P., Nelson, M.C., & Popkin, B.M. (2006). Longitudinal physical activity and sedentary behavior trends: Adolescence to adulthood. *American Journal of Preventive Medicine*, 31(6), 477-485.
4. Birch, L.L., & Davison, K.K. (2001). Family environmental factors influencing the developing behavioral controls of food intake and childhood overweight. *Pediatric Clinics of North America*, 48(4), 893-907.
5. Sobal, J., & Stunkard, A.J. (1989). Socioeconomic status and obesity: A review of the literature. *Psychological Bulletin*, 105(2), 260-275.
6. Mozaffarian, D., Hao, T., Rimm, E.B., Willett, W.C., & Hu, F.B. (2008). Changes in diet and lifestyle and long-term weight gain in women and men. *New England Journal of Medicine*, 359(3), 229-241.
7. Hu, F.B., Li, T.Y., Colditz, G.A., Willett, W.C., & Manson, J.E. (2001). Television watching and other sedentary behaviors in relation to risk of obesity and type 2 diabetes mellitus in women. *JAMA*, 289(14), 1785-1791.