Data Dictionary

- **Gender** : Sex of individual
- **Age** : How old is the individual
- **Height** : How tall is the individual
- **Weight** : How heavy is the individual
- **family_history_with_overweight** :Family history of obesity
- **FAVC** : Frequency of consumption of high-calorie foods
- **FCVC** : Frequency of vegetable consumption
- **NCP** : Number of main meals
- **CAEC** : Food consumption between meals
- **SMOKE** : Tobacco use
- **CH20** : Daily water consumption
- **SCC** : Monitoring calorie consumption.
- **FAF** : Frequency of physical activity
- **TUE** : Time for using technological devices
- **CALC** : Alcohol consumption
- **MTanANS** : Mode of transport used
- **NObeyesdad** : Gives us the obesity status of the person

Coletando Dados

Os dados foram disponibilizados em meu github público. Eu subi esses arquivos no github e estou usando esse script pronto para carregar esses arquivos no Google Colab.

```
import os
import zipfile
import requests
from io import BytesIO

def download_and_extract_github_folder(repo_url, branch, folder_path, target_path):
    zip_url = f"{repo_url}/archive/{branch}.zip"

    print("Downloading the zip file...")
    response = requests.get(zip_url, stream=True)
    zip_file = zipfile.ZipFile(BytesIO(response.content))

    print("Extracting files...")
    for file in zip_file.namelist():
        if file.startswith(f"{repo_url.split('/')[-1]}-{branch}/{folder_path}"):
            zip_file.extract(file, target_path)

    extracted_folder = os.path.join(target_path, f"{repo_url.split('/')[-1]}-{branch}", folder_path)
    os.system(f"mv {extracted_folder}/* {target_path}")
    os.system(f"rm -r {target_path}/{repo_url.split('/')[-1]}-{branch}")

    print("Files are ready to use at:", target_path)

repo_url = 'https://github.com/geeklicantropo/MESTRADO_PUBLICO'
branch = 'main'
folder_path = 'DATA_MINING/TAREFAS/Tarefa1/data/'
target_path = '/content/'

download_and_extract_github_folder(repo_url, branch, folder_path, target_path)

```

```
Downloading the zip file...
Extracting files...
Files are ready to use at: /content/
```

Loading libraries

```
import pandas as pd
import numpy as np

#Visualização de Dados
import matplotlib.pyplot as plt
import seaborn as sns

#Machine Learning - Modelagem e Avaliação
from sklearn.model_selection import train_test_split, GridSearchCV, cross_val_score
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

#Algoritmos preditivos
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
import xgboost as xgb
import lightgbm as lgb

#Biblioteca para mostrar barras de progresso em loops.
from tqdm import tqdm

#Estatísticas
from scipy.stats import chi2_contingency, f_oneway, spearmanr, skew
```

```
train = pd.read_csv("/content/train.csv")
test = pd.read_csv("/content/test.csv")
```

EDA

Aqui estou fazendo a análise exploratória dos dados, para investigar quaisquer informações implícitas nos dados que possa me ajudar a criar um modelo de classificação assertivo.

```
train.head()
```

	id	Gender	Age	Height	Weight	family_history_with_overweight	FAVC
0	0	Male	24.443011	1.699998	81.669950	yes	yes
1	1	Female	18.000000	1.560000	57.000000	yes	yes
2	2	Female	18.000000	1.711460	50.165754	yes	yes
3	3	Female	20.952737	1.710730	131.274851	yes	yes

Next steps: View recommended plots

```
test.head()
```

	id	Gender	Age	Height	Weight	family_history_with_overweight	FA
0	20758	Male	26.899886	1.848294	120.644178	yes	y
1	20759	Female	21.000000	1.600000	66.000000	yes	y
2	20760	Female	26.000000	1.643355	111.600553	yes	y
3	20761	Male	20.979254	1.553127	103.669116	yes	y

Next steps: View recommended plots

```
train.shape
```

```
(20758, 18)
```

```
1 test.shape
```

```
(13840, 17)
```

Criando uma classe para múltiplas tarefas de EDA

Aqui estou encapsulando múltiplas tarefas da análise exploratória dos dados em uma única classe.

```
class EDA:
    def __init__(self, data, target_col):
        self.data = data
        self.target_col = target_col
        self.target_data = data[target_col]

    #Principais análises estatísticas
    def data_summary(self):
      display(self.data.describe(include='all').transpose())

    #Verificar valores nulos
    def missing_value_analysis(self):
      display(
          pd.DataFrame(
              self.data.isnull().sum(axis = 0) / len(self.data) * 100, columns = ['Null Value Ratio']
          ).applymap(lambda x: f"{np.round(x, decimals = 5)}%")
      )

    #Checar os tipos de dados
    def checking_datatypes(self):
      display(self.data.dtypes)

    #Checar linhas duplicadas
    def verify_duplicates(self):
      n_duplicates = self.data.duplicated().sum()
      print(f"Number of duplicates: {n_duplicates}.")

    #Checar a distribuição dos dados de maneira visual
    def data_distribution(self):
      numerical_cols = self.data.select_dtypes(include=['float64', 'int64']).columns
      categorical_cols = self.data.select_dtypes(include=['object', 'category']).columns

      #Determina o número de linhas necessárias para subplots
      total_cols = len(numerical_cols) + len(categorical_cols)
      num_rows = (total_cols + 3) // 4  #Garanta linhas suficientes adicionando 3 antes da divisão inteira por 4

      #Cria um esboço de imagem maior que irá conter os subplots
      fig, axes = plt.subplots(nrows=num_rows, ncols=4, figsize=(20, num_rows * 4))
      axes = axes.flatten()  #Estratégia usada para evitar erros

      #Plotando historigramas para features numéricas
      for idx, col in enumerate(numerical_cols):
          if idx < len(axes):  #Estratégia usada para evitar erros
              sns.histplot(self.data[col], kde=True, bins=30, color='blue', ax=axes[idx])
              axes[idx].set_title(f'Distribution of {col}')

      #Plotando gráficos de barras para features categóricas
      offset = len(numerical_cols)  #Começa a plotar as variáveis categóricas depois das numéricas
      for idx, col in enumerate(categorical_cols, start=offset):
          if idx < len(axes):  #Estratégia usada para evitar erros
              sns.countplot(x=self.data[col], ax=axes[idx])
              axes[idx].set_title(f'Distribution of {col}')
              axes[idx].tick_params(axis='x', rotation=45)

      #Esconde qualquer eixo não usado
      for ax in axes[total_cols:]:
          ax.axis('off')

      plt.tight_layout()
      plt.show()

    #Identifica outliers usando o intervalo interquantil
    def identify_outliers(self):
      numerical_features = self.data.select_dtypes(include=['float64', 'int64']).columns
      outliers_info = {}

      # Dicionário com as descrições das variáveis
      descriptions = {
          'FAF': 'Frequency of physical activity',
          'Gender' : 'Sex of individual',
          'Age' : 'How old is the individual',
          'Height' : 'How tall is the individual',
          'Weight' : 'How heavy is the individual',
          'family_history_with_overweight' :'Family history of obesity',
          'FAVC' : 'Frequency of consumption of high-calorie foods',
          'FCVC' : 'Frequency of vegetable consumption',
          'NCP' : 'Number of main meals',
```

```
          'CAEC' : 'Food consumption between meals',
          'SMOKE' : 'Tobacco use',
          'CH2O' : 'Daily water consumption',
          'SCC' : 'Monitoring calorie consumption',
          'FAF' : 'Frequency of physical activity',
          'TUE' : 'Time for using technological devices',
          'CALC' : 'Alcohol consumption',
          'MTanANS' : 'Mode of transport used',
          'NObeyesdad' : 'Gives us the obesity status of the person'
      }

      #Determine o número de linhas necessárias para subplots
      total_cols = len(numerical_features)
      num_rows = (total_cols + 3) // 4

      #Cria um conjunto de subplots onde em cada linha, teremos no máximo 4 subplots
      fig, axes = plt.subplots(nrows=num_rows, ncols=4, figsize=(20, num_rows * 4))
      axes = axes.flatten()

      #Loop sobre todas as features numéricas para calcular e plotar os outliers
      for idx, feature in enumerate(numerical_features):
          Q1 = self.data[feature].quantile(0.25)
          Q3 = self.data[feature].quantile(0.75)
          IQR = Q3 - Q1

          lower_bound = Q1 - 1.5 * IQR
          upper_bound = Q3 + 1.5 * IQR

          outliers = self.data[(self.data[feature] < lower_bound) | (self.data[feature] > upper_bound)]
          outliers_info[feature] = {'count': len(outliers), 'outliers': outliers}

          #Plota cada boxplot em seu respectivo subplot
          sns.boxplot(y=self.data[feature], color='orange', ax=axes[idx])
          full_title = descriptions.get(feature, feature)  #Pega a descrição completa do dicionário de descriptions
          axes[idx].set_title(f'Outliers in {feature} ({full_title})')

      #Oculta quaisquer eixos não utilizados se o número de features for menor que o número de subplots
      for i in range(idx + 1, len(axes)):
          axes[i].axis('off')

      plt.tight_layout()
      plt.show()

      return outliers_info

  #Analisa a assimetria das features
  def analyze_skewness(self, plot=False):
      numerical_features = self.data.select_dtypes(include=['float64', 'int64']).columns
      skew_info = {}
      if plot:
          total_features = len(numerical_features)
          num_rows = (total_features + 3) // 4
          fig, axes = plt.subplots(nrows=num_rows, ncols=4, figsize=(20, num_rows * 4))
          axes = axes.flatten()

      for idx, feature in enumerate(numerical_features):
          skewness = skew(self.data[feature].dropna())
          skew_info[feature] = skewness
          if plot:
              sns.histplot(self.data[feature], kde=True, color='purple', ax=axes[idx])
              axes[idx].set_title(f'Skewness of {feature}: {skewness:.2f}')

      if plot:
          for i in range(idx + 1, len(axes)):
              axes[i].axis('off')
          plt.tight_layout()
          plt.show()

      #Esse retorno é necessário para ser usado na plotagem da correlação logo após.
      return skew_info


  #Cria o gráfico de correlação (Heatmap)
  def correlation_analysis_heatmap(self, skew_threshold=1):
      '''
        Verifica a assimetria usando a função de analisar assimetria. Isso é importante porque
        se a feature for assimétrica (skeweness acima de 1), usar a correlação de Pearson não irá funcionar.
```

```
          Como alternativa, a correlação de Spearman será usada.
        '''
        skew_info = self.analyze_skewness()
        skewed_features = any(abs(skew) > skew_threshold for skew in skew_info.values())

        corr_data = self.data.copy()
        for column in corr_data.select_dtypes(include=['object', 'category']).columns:
            corr_data[column] = corr_data[column].astype('category').cat.codes

        #Define o método de correlação baseado na assimetria.
        correlation_method = 'spearman' if skewed_features else 'pearson'
        correlation_matrix = corr_data.corr(method=correlation_method)

        plt.figure(figsize=(12, 10))
        sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', cbar=True)
        plt.title(f'Correlation Matrix using {correlation_method.capitalize()}')
        plt.show()

        return correlation_matrix

```

```
#Aqui estou criando uma cópia do dataframe do trem apenas para deletar a coluna id e fazer algum EDA
df_train = train.copy()
df_train = df_train.drop(labels=["id"], axis=1)
```

```
#Cria uma instância da classe de EDA para a análise exploratória de dados
analyzer = EDA(df_train, 'NObeyesdad')
```

```
#Análises estatísticas principais
print("Resumo dos dados")
analyzer.data_summary()
```

```
Resumo dos dados
```

	count	unique	top	freq	mean	
Gender	20758	2	Female	10422	NaN	
Age	20758.0	NaN	NaN	NaN	23.841804	5.6
Height	20758.0	NaN	NaN	NaN	1.700245	0.(
Weight	20758.0	NaN	NaN	NaN	87.887768	26.3
family_history_with_overweight	20758	2	yes	17014	NaN	
FAVC	20758	2	yes	18982	NaN	
FCVC	20758.0	NaN	NaN	NaN	2.445908	0.5
NCP	20758.0	NaN	NaN	NaN	2.761332	0.7
CAEC	20758	4	Sometimes	17529	NaN	
SMOKE	20758	2	no	20513	NaN	
CH2O	20758.0	NaN	NaN	NaN	2.029418	0.6
SCC	20758	2	no	20071	NaN	
FAF	20758.0	NaN	NaN	NaN	0.981747	0.8
TUE	20758.0	NaN	NaN	NaN	0.616756	0.6
CALC	20758	3	Sometimes	15066	NaN	
MTRANS	20758	5	Public_Transportation	16687	NaN	

```
#Verifica valores nulos
analyzer.missing_value_analysis()
```

	Null Value Ratio
Gender	0.0%
Age	0.0%
Height	0.0%
Weight	0.0%
family_history_with_overweight	0.0%
FAVC	0.0%
FCVC	0.0%
NCP	0.0%
CAEC	0.0%
SMOKE	0.0%
CH2O	0.0%
SCC	0.0%
FAF	0.0%
TUE	0.0%
CALC	0.0%
MTRANS	0.0%
NObeyesdad	0.0%

```
1 #Verifica os tipos de dados
2 analyzer.checking_datatypes()
```

```
Gender                           object
Age                             float64
Height                          float64
Weight                          float64
family_history_with_overweight   object
FAVC                             object
FCVC                            float64
NCP                             float64
CAEC                             object
SMOKE                            object
CH2O                            float64
SCC                              object
FAF                             float64
TUE                             float64
CALC                             object
MTRANS                           object
NObeyesdad                       object
dtype: object
```

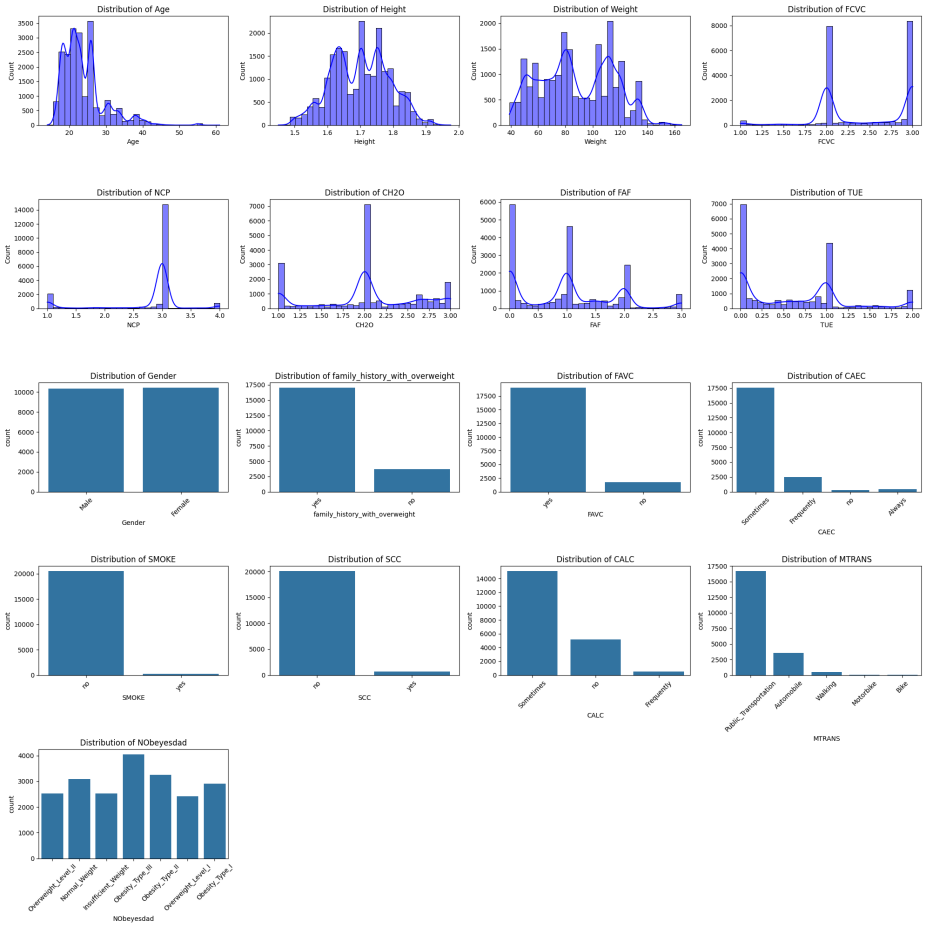
```
1 #Verifica linhas duplicadas
2 analyzer.verify_duplicates()
```

```
Number of duplicates: 0.
```

- A obesidade tipo 3 (o pior tipo) é a mais comum, infelizmente
- Existem alguns valores discrepantes para algumas colunas, especialmente Age
- A coluna Age é um valor do tipo Float. Ela precisa ser alterada para um inteiro
- Existem algumas features muito desbalanceadas
- Não temos problemas com valores nulos
- Não temos problemas com duplicatas também.

```
1 #Verifica a distribuição dos dados
2 print("Distribuição de Dados:")
3 analyzer.data_distribution()
```

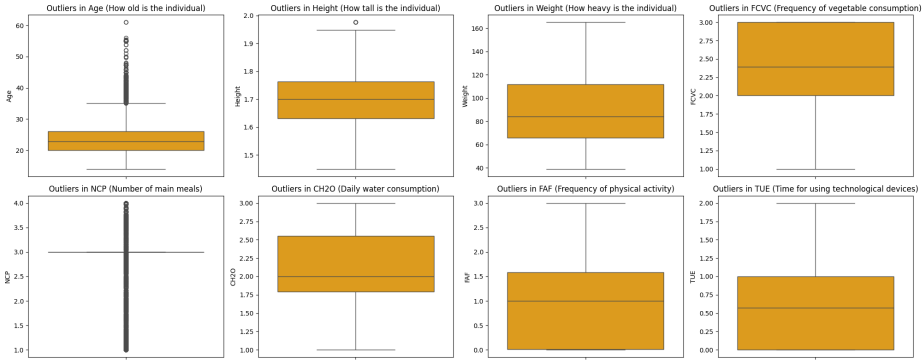
```
Distribuição de Dados:
```



- O histograma que mostra gráficos numéricos está skewed na maioria das features.
- Apenas Height e Weight e Age têm uma distribuição que parece uma distribuição normal.
- Precisamos aplicar a normalização a esses dados.
- Os valores categóricos são em sua maioria desequilibrados. Apenas a coluna Gender está bem equilibrada

```
1 # Identificar e imprimir informações sobre outliers
2 print("\nIdentificando e plotando outliers:")
3 outliers_info = analyzer.identify_outliers()
4 for feature, details in outliers_info.items():
5     print(f"{feature} tem {details['count']} outliers")
```

```
Identificando e plotando outliers:
```

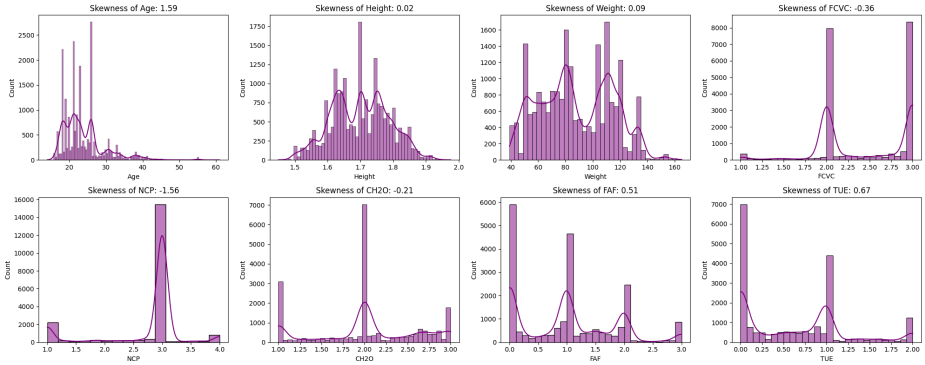



```
Age tem 1074 outliers
Height tem 4 outliers
Weight tem 0 outliers
FCVC tem 0 outliers
NCP tem 6052 outliers
CH2O tem 0 outliers
FAF tem 0 outliers
TUE tem 0 outliers
```

- Como identificado antes, Age tem a maior quantidade de outliers.
- NCP possui valores entre 1 e 4. Se todos ou a maioria dos dados estão concentrados em um único valor ou em poucos valores muito próximos, será difícil enxergar o boxplot. Neste caso, o primeiro quartil (Q1), a mediana (Q2) e o terceiro quartil (Q3) podem ser todos iguais ou muito próximos.
- Eu entendo que o NCP deveria ser uma variável inteira e não um float.

```
# Analisar e imprimir informações sobre a assimetria (skewness)
print("\nAnalisando visualmente a assimetria dos dados:")
skew_info = analyzer.analyze_skewness(plot=True)
print("Informações de Skewness:")
for feature, skewness in skew_info.items():
    print(f"{feature}: Skewness = {skewness:.2f}")
```

```
Analisando visualmente a assimetria dos dados:
```




```
Informações de Skewness:
Age: Skewness = 1.59
Height: Skewness = 0.02
Weight: Skewness = 0.09
FCVC: Skewness = -0.36
NCP: Skewness = -1.56
CH2O: Skewness = -0.21
FAF: Skewness = 0.51
TUE: Skewness = 0.67
```

O **skew_threshold** é um parâmetro que define o limite para considerar uma distribuição como significativamente assimétrica ou normal.

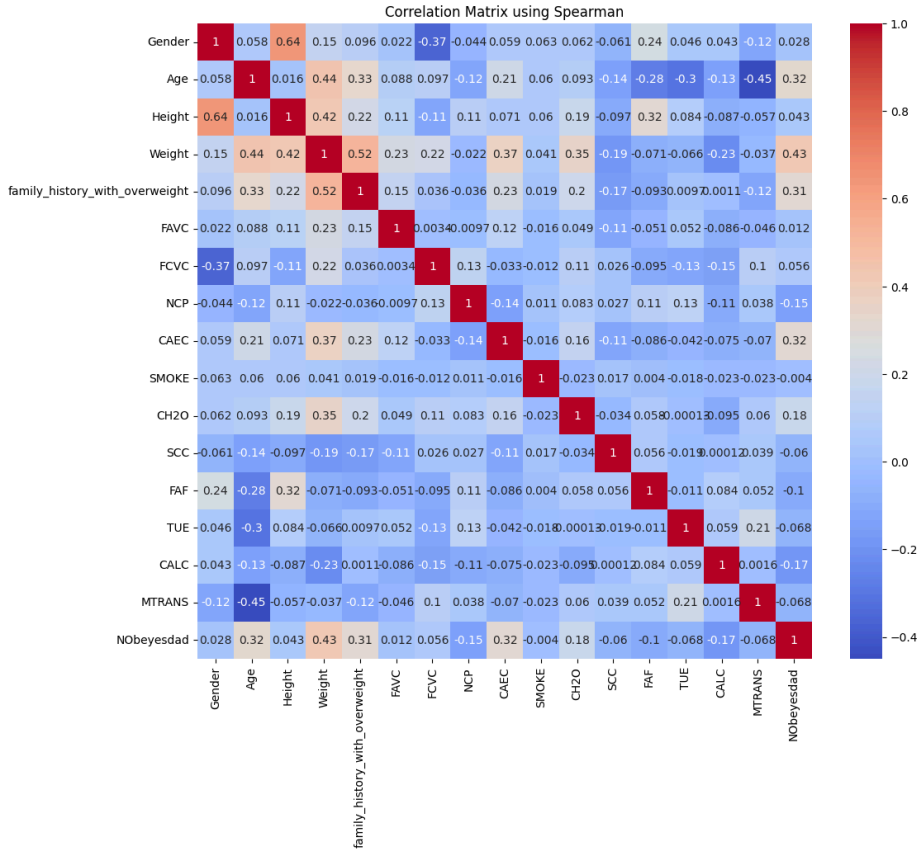
Se o skewness é um valor:

- Próximo a 0 indica uma distribuição simétrica (normal).
- Maior que 0 indica uma assimetria positiva (cauda longa à direita).
- Menor que 0 indica uma assimetria negativa (cauda longa à esquerda).
- Valor maior que 1 sugere uma distribuição altamente assimétrica.

Somente Height e Weight possuem um valor de skill mais ou menos simétrico.

```
#Realizar análise de correlação e imprimir a matriz de correlação
print("\nHeatmap que mostra a correlação dentre todas as features:")
correlation_matrix = analyzer.correlation_analysis_heatmap()
```

```
Heatmap que mostra a correlação dentre todas as features:
```



- Como era de se esperar, a variável weight é a variável que possui maior correlação com a variável alvo.
- CAEC : Food consumption between meals, também possui uma significativa correlação com a variável alvo
- Age and family_history_with_overweight é altamente correlacionada com a variável alvo.
- Gender não tem uma correlação muito alta com a variável alto, ao contrário do que o senso comum possa dizer.

Feature Engineering

Baseado nas informações coletadas pela análise exploratória de dados, algumas mudanças nas features se fazem necessárias.

- Em alguns casos, os seus tipos de dados precisam ser alterados.
- Em outros casos, algumas features precisa ter os seus valores normalizados.
- Algumas novas colunas precisam ser criadas.

```
1 train.head(2)
```

	id	Gender	Age	Height	Weight	family_history_with_overweight	FAVC	FC
0	0	Male	24.443011	1.699998	81.66995		yes	yes

Next steps: View recommended plots

```python
#Função para aplicar transformações nos dados
def feature_engineering(data):
    #Transformar todas as colunas numéricas para float, exceto 'Age'
    numerical_cols = data.select_dtypes(include=['int64', 'float64']).columns

    #Converte as colunas para os seus tipos adequados
    data[numerical_cols] = data[numerical_cols].astype(float)
    data['id'] = data['id'].astype(int)
    data['Age'] = data['Age'].astype(int)
    data['CH2O'] = data['CH2O'].astype(int)

    #Criar nova coluna para Age em escalas de 10 em 10
    data['Age_Scale'] = data['Age'] // 10 + 1

    #Criar nova coluna para Weight em escalas de 20 em 20
    data['Weight_Scale'] = data['Weight'] // 20 + 1

    #Criar nova coluna para Height em escalas de 10 em 10 centímetros
    #Encontrar o mínimo e o máximo da coluna 'Height'
    min_height = data['Height'].min()
    max_height = data['Height'].max()
    #Arredondar para baixo o mínimo e para cima o máximo para o múltiplo de 0.1 mais próximo
    min_height = np.floor(min_height * 10) / 10
    max_height = np.ceil(max_height * 10) / 10
    #Criar os bins a partir do mínimo ao máximo com incrementos de 0.1 metros
    bins = np.arange(min_height, max_height + 0.1, 0.1)  #Usar np.arange para garantir inclusão do último intervalo
    #Usar pd.cut para categorizar os valores de 'Height' e obter os códigos dos bins
    data['Height_bin'] = pd.cut(data['Height'], bins=bins, right=False, labels=False)
    #Calcular o ponto médio de cada bin para atribuir à nova coluna 'Height_scale'
    data['Height_scale'] = [(bins[i] + bins[i+1])/2 for i in data['Height_bin']]
    #Dropar coluna Height_bin
    data = data.drop(labels=["Height_bin"], axis=1)

    #Calcular o Índice de Massa Corporal (BMI)
    data['BMI'] = data['Weight'] / (data['Height'] ** 2)

    #Arredondar todas as colunas float para 2 casas decimais
    for column in data.select_dtypes(include=['float64']).columns:
        data[column] = data[column].round(2)

    #Transformar colunas binárias
    binary_columns = ['Gender', 'family_history_with_overweight', 'FAVC', 'SMOKE', 'SCC']
    binary_mapping = {'yes': 1, 'no': 0, 'true': 1, 'false': 0, 'male': 1, 'female': 0}
    for column in binary_columns:
        #Certificar-se de que a coluna é de tipo string e lida com valores ausentes
        data[column] = data[column].astype(str).str.lower().replace(binary_mapping)

    #Transformar colunas categóricas em números, considerando sua ordem hierárquica (na maioria das features)
    data['CAEC'] = data['CAEC'].map({'no': 0, 'Sometimes': 1, 'Frequently': 2, 'Always': 3})
    data['CALC'] = data['CALC'].map({'no': 0, 'Sometimes': 1, 'Frequently': 2})
    data['MTRANS'] = data['MTRANS'].astype('category').cat.codes  #Sem hierarquia específica, encoding ordinal

    if "NObeyesdad" in data.columns:
      data['NObeyesdad'] = data['NObeyesdad'].map({
          '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
      })

    return data


```

```
# Aplicar a função de feature engineering
df_transformed = train.copy()
df_transformed = feature_engineering(df_transformed)
df_transformed = df_transformed.drop(labels=["id"], axis=1)
```

```
df_transformed.head(2)
```

	Gender	Age	Height	Weight	family_history_with_overweight	FAVC	FCVC	NCP	CAE
0	1	24	1.70	81.67	1	1	2.0	2.98	
1	0	18	1.56	57.00	1	1	2.0	3.00	

Encontrando as features mais importantes

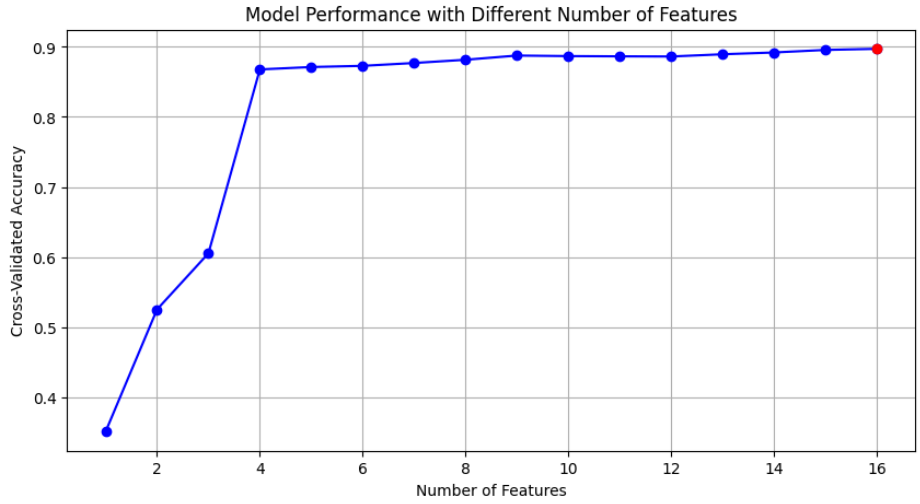
Um processo será rodado abaixo com 2 objetivos:

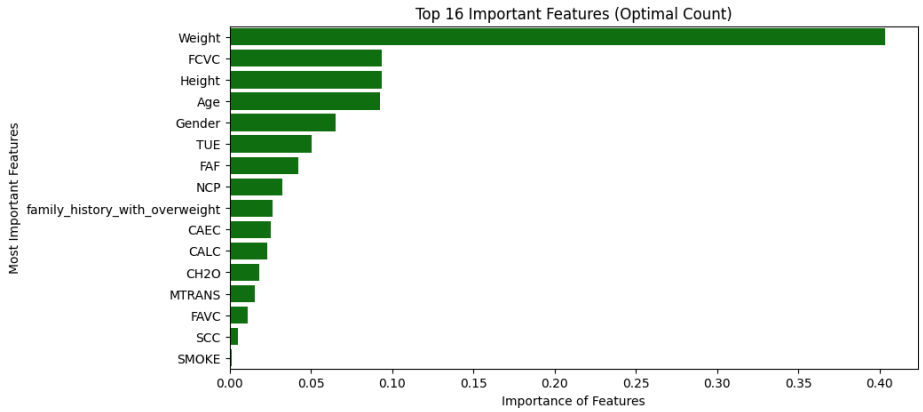
- Encontrar a quantidade ideal de features para criar um modelo de classificação
- Encontrar quais são as features mais importantes para criar o modelo de classificação

```
'''
Determina as características ótimas removendo características altamente correlacionadas e
avaliando o desempenho com um número crescente de características.
'''
#Plota a performance do modelo dado números diferentes de features
def plot_feature_performance(max_features, scores, best_feature_count, best_score):
    plt.figure(figsize=(10, 5))
    plt.plot(range(1, max_features + 1), scores, marker='o', linestyle='-', color='blue')
    plt.plot(best_feature_count, best_score, 'ro')
    plt.title('Model Performance with Different Number of Features')
    plt.xlabel('Number of Features')
    plt.ylabel('Cross-Validated Accuracy')
    plt.grid(True)
    plt.show()

#Plota a importância das features selecionados após identificar o número ideal de recursos
def plot_feature_importance(X, best_features, y, num_features):
    model = RandomForestClassifier(n_estimators=100, random_state=42)
    model.fit(X[best_features], y)
    feature_importances = pd.Series(model.feature_importances_, index=best_features).sort_values(ascending=False)

    plt.figure(figsize=(10, 5))
    sns.barplot(x=feature_importances.values, y=feature_importances.index, color='green')
    plt.title(f'Top {num_features} Important Features (Optimal Count)')
    plt.xlabel('Importance of Features')
    plt.ylabel('Most Important Features')
    plt.show()

def find_optimal_features(data, target_col, max_features=None, corr_threshold=0.85):
    #Separando as variávels independentes da variável alvo
    X = data.drop(target_col, axis=1)
    y = data[target_col]

    #Normalizando os dados
    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X)
    X_scaled_df = pd.DataFrame(X_scaled, columns=X.columns)

    #Calculando a matriz de correlação e removendo features altamente correlacionadas
    corr_matrix = X_scaled_df.corr().abs()
    upper = corr_matrix.where(np.triu(np.ones(corr_matrix.shape), k=1).astype(bool))
    to_drop = [column for column in upper.columns if any(upper[column] > corr_threshold)]
    X_reduced = X_scaled_df.drop(columns=to_drop)

    if max_features is None or max_features > len(X_reduced.columns):
        max_features = len(X_reduced.columns)

    model = RandomForestClassifier(n_estimators=100, random_state=42)
    scores = []

    # Modelos de treinamento com um número crescente de features importantes
    for i in range(1, max_features + 1):
        selected_features = X_reduced.columns[:i]
        score = np.mean(cross_val_score(model, X_reduced[selected_features], y, cv=5))
        scores.append(score)

    #Identifica o número otimizado de features
    best_feature_count = np.argmax(scores) + 1
    best_score = max(scores)
    best_features = X_reduced.columns[:best_feature_count]

    #Plotagem a performance do modelo e a importância das features
    plot_feature_performance(max_features, scores, best_feature_count, best_score)
    plot_feature_importance(X_reduced, best_features, y, best_feature_count)

    return list(best_features)
```

```
def evaluate_models_with_gridsearch(data, target_col):
    best_features = find_optimal_features(data, target_col)
    X_best = data[best_features]
    y = data[target_col]

    #Normaliza os valores das features.
    scaler = StandardScaler()
```

```
    X_best_scaled = scaler.fit_transform(X_best)

    models = {
        'Naive Bayes': (GaussianNB(), {}),
        'SVM': (SVC(probability=True), {'C': [0.1, 1, 10], 'gamma': ['scale', 'auto'], 'kernel': ['linear', 'rbf']}),
        'Random Forest': (RandomForestClassifier(), {'n_estimators': [100, 200, 300], 'max_depth': [10, 20, 30, None]}),
        'XGBoost': (xgb.XGBClassifier(use_label_encoder=False, eval_metric='mlogloss'), {'n_estimators': [100, 200], 'max_de
        'LightGBM': (lgb.LGBMClassifier(), {'n_estimators': [100, 200], 'num_leaves': [31, 64, 128], 'learning_rate': [0.1,
    }

    results = []
    for name, (model, params) in tqdm(models.items(), desc="Training Models"):
        grid_search = GridSearchCV(model, params, cv=5, scoring='accuracy', verbose=2, n_jobs=-1)
        grid_search.fit(X_best_scaled, y)
        best_model = grid_search.best_estimator_
        best_params = grid_search.best_params_
        scores = cross_val_score(best_model, X_best_scaled, y, cv=5, scoring='accuracy')
        accuracy = np.mean(scores)
        results.append({'Model': name, 'Best Accuracy': accuracy, 'Best Parameters': best_params})

    results_df = pd.DataFrame(results)
    print(results_df)

    # Plot the model performances for visual comparison
    plt.figure(figsize=(10, 6))
    sns.barplot(x='Model', y='Best Accuracy', data=results_df, palette='viridis')
    plt.title('Performance of Various Models')
    plt.ylabel('Accuracy')
    plt.show()

    return results_df


results_df = evaluate_models_with_gridsearch(df_transformed, 'NObeyesdad')
```






```
Training Models:   0%|          | 0/5 [00:00<?, ?it/s]Fitting 5 folds for each of 1
Training Models:  20%|██        | 1/5 [00:02<00:08,  2.11s/it]Fitting 5 folds for ea
Training Models:  40%|████      | 2/5 [37:10<1:05:34, 1311.55s/it]Fitting 5 folds fo
Training Models:  60%|██████    | 3/5 [41:03<27:18, 819.19s/it]   Fitting 5 folds fo
Training Models:  80%|████████  | 4/5 [44:07<09:28, 568.45s/it]Fitting 5 folds for e
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing wa
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 1064
[LightGBM] [Info] Number of data points in the train set: 20758, number of used feat
[LightGBM] [Info] Start training from score -2.107483
[LightGBM] [Info] Start training from score -1.907353
[LightGBM] [Info] Start training from score -2.146276
[LightGBM] [Info] Start training from score -2.107879
[LightGBM] [Info] Start training from score -1.964779
[LightGBM] [Info] Start training from score -1.854892
[LightGBM] [Info] Start training from score -1.635203
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing wa
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 1055
[LightGBM] [Info] Number of data points in the train set: 16606, number of used feat
[LightGBM] [Info] Start training from score -2.107657
[LightGBM] [Info] Start training from score -1.907167
[LightGBM] [Info] Start training from score -2.146046
[LightGBM] [Info] Start training from score -2.108153
[LightGBM] [Info] Start training from score -1.964755
[LightGBM] [Info] Start training from score -1.855022
[LightGBM] [Info] Start training from score -1.635117
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing wa
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 1061
[LightGBM] [Info] Number of data points in the train set: 16606, number of used feat
[LightGBM] [Info] Start training from score -2.107657
[LightGBM] [Info] Start training from score -1.907167
[LightGBM] [Info] Start training from score -2.146561
[LightGBM] [Info] Start training from score -2.108153
[LightGBM] [Info] Start training from score -1.964755
```