```
import pandas as pd
import numpy as np
import re
import json
import requests
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import (
    StandardScaler,
    OrdinalEncoder,
    OneHotEncoder,
    LabelEncoder
)
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report
import joblib

#funcoes utilizadas
def criar_csv_de_dataframe(df: pd.DataFrame, nome_arquivo: str) -> None:
    """
    Gera um arquivo CSV a partir de um DataFrame.

    Args:
        df (pd.DataFrame): O DataFrame a ser salvo.
        nome_arquivo (str): O nome do arquivo CSV de saída (incluindo a extensão .csv).
    """
    if not isinstance(df, pd.DataFrame):
        print("Erro: O primeiro argumento deve ser um DataFrame do pandas.")
        return

    if not isinstance(nome_arquivo, str) or not nome_arquivo.lower().endswith('.csv'):
        print("Erro: O nome do arquivo deve ser uma string e terminar com '.csv'.")
        return

    try:
        df.to_csv(nome_arquivo, index=False)
        print(f"DataFrame salvo com sucesso em '{nome_arquivo}'")
    except Exception as e:
        print(f"Ocorreu um erro ao salvar o DataFrame em '{nome_arquivo}': {e}")

def renomear_colunas(df: pd.DataFrame, mapa_renomeacao: dict = None) -> pd.DataFrame:
    """
    Renomeia as colunas de um DataFrame utilizando um dicionário fornecido
    ou aplicando um padrão de formatação (minúsculas, espaços por '_').
    Retorna o DataFrame com as colunas renomeadas.
    """
    colunas_originais = df.columns.tolist()
    df_modificado = df.copy()

    if mapa_renomeacao:

        if any(col in mapa_renomeacao for col in colunas_originais):
            df_modificado.rename(columns=mapa_renomeacao, inplace=True)
        else:
            print('Não foi possível realizar a alteração dos nomes das colunas. Rever arquivo de entrada.')
    else:

        print('Não foi possível realizar a alteração dos nomes das colunas. Rever arquivo de entrada.')

    return df_modificado

def transformar_valores_string(df: pd.DataFrame, coluna: str, mapa_transformacao: dict):
    """
    Transforma os valores de uma coluna do tipo string para string utilizando um mapeamento.
    Modifica o DataFrame inplace.
    """
    if coluna not in df.columns:
        print(f"Erro: A coluna '{coluna}' não existe no DataFrame.")
        return

    if df[coluna].dtype != 'object':
        print(f"Erro: A coluna '{coluna}' não é do tipo 'object'. Nenhuma transformação será aplicada.")
        return

    valores_unicos_na_coluna = df[coluna].dropna().unique()
    valores_nao_mapeados = [valor for valor in valores_unicos_na_coluna if valor not in mapa_transformacao]

    if valores_nao_mapeados:
        print(f"Erro: Os seguintes valores únicos na coluna '{coluna}' não foram encontrados no mapeamento:")
```

```
        print(f"Erro: Os seguintes valores únicos na coluna '{coluna}' não foram encontrados no mapeamento.")
        print(valores_nao_mapeados)
        print("A transformação não será realizada pois nem todos os valores possuem um mapeamento.")
        return

    df[coluna] = df[coluna].map(mapa_transformacao)
    print(f"Coluna '{coluna}' transformada com sucesso.")


def ler_json_de_url(url: str) -> dict:
    """
    Lê um arquivo JSON de uma URL fornecida.
    """
    try:
        resposta = requests.get(url)
        resposta.raise_for_status()
        dict_dados = resposta.json()
        return dict_dados
    except Exception as e:
        print(f"Erro ao ler JSON de {url}: {e}")
        return None


class MtransGrouper(BaseEstimator, TransformerMixin):
    """Agrupa as categorias raras de 'mtrans'."""
    def fit(self, X, y=None):
        return self

    def transform(self, X):
        X_array = np.asarray(X)
        X_series = pd.Series(X_array.flatten(), name='mtrans')
        mtrans_agrupado = X_series.replace(
            ['moto', 'bicicleta', 'caminhando'], 'outros'
        )
        return mtrans_agrupado.values.reshape(-1, 1)

    def get_feature_names_out(self, input_features=None):
        if input_features is None:
            return ['mtrans_grouped']
        return input_features


class CalcGrouper(BaseEstimator, TransformerMixin):
    """Agrupa as categorias raras de 'calc'."""
    def fit(self, X, y=None):
        return self

    def transform(self, X):
        X_array = np.asarray(X)
        X_series = pd.Series(X_array.flatten(), name='calc')
        sempre_freq = X_series.replace(
            'sempre', 'frequentemente'
        )
        return sempre_freq.values.reshape(-1, 1)

    def get_feature_names_out(self, input_features=None):
        if input_features is None:
            return ['calc_grouped']
        return input_features

class RoundingTransformer(BaseEstimator, TransformerMixin):
    """Arredonda os dados sintéticos (ex: 2.45 -> 2)"""
    def fit(self, X, y=None):
        return self

    def transform(self, X, **kwargs):
        return np.round(X).astype(int)

    def get_feature_names_out(self, input_features=None):
        if input_features is None:
            raise ValueError("input_features must be provided for RoundingTransformer.get_feature_names_out")
        return input_features

#etapa - importacao de dados

df = pd.read_csv('https://raw.githubusercontent.com/bernardo-seg/tech-challenge-fase4-obesidade/refs/heads/main/dados/Obesit
mapa_colunas = ler_json_de_url('https://raw.githubusercontent.com/bernardo-seg/tech-challenge-fase4-obesidade/refs/heads/mai
mapa_valores_colunas = ler_json_de_url('https://raw.githubusercontent.com/bernardo-seg/tech-challenge-fase4-obesidade/refs/h

print("DataFrame e mapas JSON carregados com sucesso.")

df_processado = renomear_colunas(df, mapa_colunas)
```

```
#etapa - pre-processamento

# Transformar valores de string
if mapa_valores_colunas:
    if 'mapeamento_classificacao_peso_corporal' in mapa_valores_colunas:
        transformar_valores_string(df_processado, 'classificacao_peso_corporal' , mapa_valores_colunas['mapeamento_classific
    if 'mapeamento_mtrans' in mapa_valores_colunas:
         transformar_valores_string(df_processado,'mtrans' , mapa_valores_colunas['mapeamento_mtrans']['valores_novos_mtrans
    if 'mapeamento_frequencia' in mapa_valores_colunas:
        transformar_valores_string(df_processado,'caec' , mapa_valores_colunas['mapeamento_frequencia']['valores_novos_frequ
        transformar_valores_string(df_processado,'calc' , mapa_valores_colunas['mapeamento_frequencia']['valores_novos_frequ
    if 'mapeamento_genero' in mapa_valores_colunas:
        transformar_valores_string(df_processado,'genero' , mapa_valores_colunas['mapeamento_genero']['transformacao_genero'
    if 'mapeamento_sim_nao' in mapa_valores_colunas:
        colunas_sim_nao = ['historico_familiar', 'favc', 'fumante', 'scc']
        for coluna in colunas_sim_nao:
            transformar_valores_string(df_processado, coluna , mapa_valores_colunas['mapeamento_sim_nao']['transformacao_sim

criar_csv_de_dataframe(df_processado, 'obesidade_processado.csv')

print("Pré-processamentos iniciais aplicados e df_processado criado.")


coluna_alvo = 'classificacao_peso_corporal'

# Remover variaveis com vazamento ou risco extremo e colunas que não serão usadas no pipeline
variaveis_a_remover_de_X_antes_do_pipeline = [
    'peso', 'altura', 'fumante', 'scc'
]

X = df_processado.drop(columns=[coluna_alvo] + variaveis_a_remover_de_X_antes_do_pipeline)
y = df_processado[coluna_alvo]

# Codificar o Alvo (y) de texto para números
le = LabelEncoder()
y_codificada = le.fit_transform(y)

# Separar dados em treino e teste
X_treino, X_teste, y_treino, y_teste = train_test_split(
    X,
    y_codificada,
    test_size=0.2,
    random_state=42,
    stratify=y_codificada
)

print("Dados preparados: X, y definidos, y codificado e dividido em treino/teste.")

variaveis_continuas = ['idade']
variaveis_bin_nominal = ['genero', 'historico_familiar', 'favc']
variaveis_multi_nominal = ['mtrans']
variaveis_clean_ordenadas = ['caec']
variaveis_para_arredondar_e_codificar = ['fcvc', 'ncp', 'ch20', 'faf', 'tue']

pipeline_continua = Pipeline(steps=[
    ('scaler', StandardScaler())
])

# Pipeline que primeiro arredonda e depois codifica ordinalmente
pipeline_arrredonamento_ordenacao = Pipeline(steps=[
    ('rounder', RoundingTransformer()),
    ('encoder', OrdinalEncoder(handle_unknown='use_encoded_value', unknown_value=-1))
])

pipeline_ordenada_limpa = Pipeline(steps=[
    ('encoder', OrdinalEncoder(handle_unknown='use_encoded_value', unknown_value=-1))
])

pipeline_calc = Pipeline(steps=[
    ('grouper', CalcGrouper()),
    ('encoder', OrdinalEncoder(handle_unknown='use_encoded_value', unknown_value=-1))
])

pipeline_nominal_bin = Pipeline(steps=[
    ('encoder', OrdinalEncoder(handle_unknown='use_encoded_value', unknown_value=-1))
])

pipeline_nominal_multi = Pipeline(steps=[
    ('grouper', MtransGrouper()),
    ('encoder', OneHotEncoder(handle_unknown='ignore', drop='first'))
])

preprocessador = ColumnTransformer(
```

```
    transformers=[
        ('cont', pipeline_continua, variaveis_continuas),
        ('arredondada_ord', pipeline_arrredonamento_ordenacao, variaveis_para_arredondar_e_codificar),
        ('ord_limpa', pipeline_ordenada_limpa, variaveis_clean_ordenadas),
        ('calc_pipe', pipeline_calc, ['calc']),
        ('nom_bin', pipeline_nominal_bin, variaveis_bin_nominal),
        ('nom_multi', pipeline_nominal_multi, variaveis_multi_nominal)
    ],
    remainder='drop'
)

modelo_rf = RandomForestClassifier(
        random_state=42,
        n_jobs=-1,
        max_depth=20,
        min_samples_leaf=1,
        min_samples_split=2,
        n_estimators=100
    )

pipeline_completa_rf = Pipeline(steps=[
    ('preprocessor', preprocessador),
    ('model', modelo_rf)
])

print("Pipelines de pré-processamento e o modelo Random Forest definidos e combinados no pipeline completo.")

#etapa - treinameto
print("Iniciando o treinamento do pipeline Random Forest")

pipeline_completa_rf.fit(X_treino, y_treino)

y_prev_rf = pipeline_completa_rf.predict(X_teste)
rf_acuracia = accuracy_score(y_teste, y_prev_rf)
print(f"Acurácia do Pipeline Random Forest: {rf_acuracia * 100:.2f}%")

nomes_classes = le.classes_
print("\nRelatório de Classificação:")
classification_report(y_teste, y_prev_rf, target_names=nomes_classes)

print("\nPipeline de pré-processamento e modelagem concluído!")

print("Salvando artefatos...")
joblib.dump(pipeline_completa_rf, 'pipeline_obesidade_completo_rf.joblib')
joblib.dump(le, 'label_encoder_rf.joblib')
print("Artefatos salvos com sucesso!")
print(f"O LabelEncoder salvo contém as seguintes classes: {le.classes_}")
```

```
DataFrame e mapas JSON carregados com sucesso.
Coluna 'classificacao_peso_corporal' transformada com sucesso.
Coluna 'mtrans' transformada com sucesso.
Coluna 'caec' transformada com sucesso.
Coluna 'calc' transformada com sucesso.
Coluna 'genero' transformada com sucesso.
Coluna 'historico_familiar' transformada com sucesso.
Coluna 'favc' transformada com sucesso.
Coluna 'fumante' transformada com sucesso.
Coluna 'scc' transformada com sucesso.
DataFrame salvo com sucesso em 'obesidade_processado.csv'
Pré-processamentos iniciais aplicados e df_processado criado.
Dados preparados: X, y definidos, y codificado e dividido em treino/teste.
Pipelines de pré-processamento e o modelo Random Forest definidos e combinados no pipeline completo.
Iniciando o treinamento do pipeline Random Forest
Acurácia do Pipeline Random Forest: 83.69%

Relatório de Classificação:

Pipeline de pré-processamento e modelagem concluído!
Salvando artefatos...
Artefatos salvos com sucesso!
O LabelEncoder salvo contém as seguintes classes: ['obesidade_tipo_1' 'obesidade_tipo_2' 'obesidade_tipo_3'
 'peso_insuficiente' 'peso_normal' 'sobrepeso_tipo_1' 'sobrepeso_tipo_2']
```

verificação dados transformados

```
import joblib
import pandas as pd


print("Carregando o pipeline completo...")
pipeline_carregado = joblib.load('pipeline_obesidade_completo_rf.joblib')
print("Pipeline carregado com sucesso!")
```

```
preprocessador_carregado = pipeline_carregado.named_steps['preprocessor']


X_treino_transformado = preprocessador_carregado.fit_transform(X_treino)
X_teste_transformado = preprocessador_carregado.transform(X_teste)



nomes_features_transformadas = preprocessador_carregado.get_feature_names_out()


df_treino_transformado = pd.DataFrame(X_treino_transformado, columns=nomes_features_transformadas, index=X_treino.index)
df_teste_transformado = pd.DataFrame(X_teste_transformado, columns=nomes_features_transformadas, index=X_teste.index)

print("\n--- Visualização dos Dados de Treino Transformados ---")
display(df_treino_transformado.head())

print("\n--- Visualização dos Dados de Teste Transformados ---")
display(df_teste_transformado.head())
```

```
Carregando o pipeline completo...
Pipeline carregado com sucesso!

--- Visualização dos Dados de Treino Transformados ---
```

	cont__idade	arredondada_ord__fcvc	arredondada_ord__ncp	arredondada_ord__ch20	arredondada_ord__faf	arredondada_ord
1057	0.026740	1.0	2.0	2.0	0.0	
523	-0.518641	2.0	0.0	0.0	0.0	
1556	-0.672563	2.0	0.0	0.0	0.0	
1429	1.962814	1.0	1.0	2.0	2.0	
228	2.489398	2.0	2.0	2.0	0.0	

```
--- Visualização dos Dados de Teste Transformados ---
```

	cont__idade	arredondada_ord__fcvc	arredondada_ord__ncp	arredondada_ord__ch20	arredondada_ord__faf	arredondada_ord
1210	0.279714	2.0	2.0	2.0	2.0	
1548	1.005746	2.0	2.0	1.0	1.0	
1002	-0.026476	1.0	2.0	2.0	0.0	
394	-1.151912	1.0	2.0	0.0	0.0	
468	-0.676959	2.0	2.0	0.0	1.0	

Valores Categorias

Valores Únicos das Variáveis Categóricas Antes das Transformações do Pipeline

```
print("Valores Únicos ANTES das transformações do pipeline:")
categorical_cols_before = [
    'genero', 'historico_familiar', 'favc', 'caec', 'calc', 'mtrans', 'classificacao_peso_corporal'
]

for col in categorical_cols_before:
    if col in df_processado.columns:
        print(f"  Coluna '{col}': {sorted(df_processado[col].unique().tolist())}")
    else:
        print(f"  A coluna '{col}' não foi encontrada no df_processado.")
```

```
Valores Únicos ANTES das transformações do pipeline:
  Coluna 'genero': ['feminino', 'masculino']
  Coluna 'historico_familiar': ['nao', 'sim']
  Coluna 'favc': ['nao', 'sim']
  Coluna 'caec': ['as_vezes', 'frequentemente', 'nunca', 'sempre']
  Coluna 'calc': ['as_vezes', 'frequentemente', 'nunca', 'sempre']
  Coluna 'mtrans': ['bicicleta', 'caminhando', 'carro', 'moto', 'transporte_publico']
  Coluna 'classificacao_peso_corporal': ['obesidade_tipo_1', 'obesidade_tipo_2', 'obesidade_tipo_3', 'peso_insuficiente', 'p
```

Valores Únicos das Variáveis Categóricas Após as Transformações do Pipeline

```
print("Valores Únicos DEPOIS das transformações do pipeline (usando df_treino_transformado):")
```

```
categorical_cols_after = [
    'nom_bin__genero', 'nom_bin__historico_familiar', 'nom_bin__favc',
    'ord_limpa__caec', 'calc_pipe__calc',
    'nom_multi__mtrans_outros', 'nom_multi__mtrans_transporte_publico'
]


print(f"  Target (y_treino) codificado: {sorted(np.unique(y_treino).tolist())}")

for col in categorical_cols_after:
    if col in df_treino_transformado.columns:
        print(f"  Coluna '{col}': {sorted(df_treino_transformado[col].unique().tolist())}")
    else:
        print(f"  A coluna '{col}' não foi encontrada no df_treino_transformado.")
```

```
Valores Únicos DEPOIS das transformações do pipeline (usando df_treino_transformado):
  Target (y_treino) codificado: [0, 1, 2, 3, 4, 5, 6]
  Coluna 'nom_bin__genero': [0.0, 1.0]
  Coluna 'nom_bin__historico_familiar': [0.0, 1.0]
  Coluna 'nom_bin__favc': [0.0, 1.0]
  Coluna 'ord_limpa__caec': [0.0, 1.0, 2.0, 3.0]
  Coluna 'calc_pipe__calc': [0.0, 1.0, 2.0]
  Coluna 'nom_multi__mtrans_outros': [0.0, 1.0]
  Coluna 'nom_multi__mtrans_transporte_publico': [0.0, 1.0]
```