

# rotina\_final\_1977

November 22, 2015

## 1 Setup Inicial

```
In [ ]: # coding: utf-8
import math
import logging
import time
import pandas as pd

# Faz os gráficos de output ficarem um pouco melhores
pd.set_option('display.mpl_style', 'default')
```

## 2 Definindo Loggers

Define loggers

Estes 'loggers' serão utilizados para salvar as saídas (outputs) em um arquivo de texto no diretório 'outputs'.

ref: <http://stackoverflow.com/questions/17035077/python-logging-to-multiple-log-files-from-different-classes>

**ATENÇÃO: RODAR O BLOCO ABAIXO APENAS UMA VEZ, SE ESTE BLOCO FOR EXECUTADO MAIS DE UMA VEZ OS LOGs SERÃO DUPLICADOS.**

```
In [ ]: log_formatter = logging.Formatter('%(asctime)s | %(levelname)s: %(message)s')

log_output = logging.getLogger('log_output')
FH_output = logging.FileHandler(
    'outputs/1977_output.log', mode='w')
FH_output.setFormatter(log_formatter)
log_output.setLevel(logging.INFO)
log_output.addHandler(FH_output)
log_output.propagate = False

# Este logger (log_tela) loga na tela e também
# no arquivo de output, junto com o conteúdo do
# logger log_output
log_tela = logging.getLogger('log_tela')
SH_tela = logging.StreamHandler()
SH_tela.setFormatter(log_formatter)
log_tela.addHandler(SH_tela)
log_tela.addHandler(FH_output)
log_tela.setLevel(logging.INFO)
log_tela.propagate = False
```

## 3 Funções de 1977

### 3.1 Funções gerais assessórias

Função	Status
verifica_dummy	OK
verifica_range	OK
pre_processamento	OK
coordenadas	OK

```
In [ ]: log_tela.info('Definindo as funções gerais assessórias')
log_output.info('\n\n=====')

def verifica_dummy(df, variavel):
    """
    Verifica se uma variável, dummy, contém algum valor diferente de 0 ou de 1.
    :param df: dataframe com os dados a serem verificados
    :param variavel: string com o nome da variável (coluna) que tem os dados
                    que devem ser verificados.
    :return: Sem retorno, apenas salva as avaliações nos logs.
    Uso:
        verifica_dummy(dataframe, 'coluna a ser verificada')
    """
    contador_de_erros = 0
    log_tela.info('Verificando a variável Dummy: ' + variavel)

    df_erros = df[(df[variavel]!=0) & (df[variavel]!=1)]
    if len(df_erros[variavel].value_counts()) > 0:
        log_tela.warning(variavel + ": " +
                        str(len(df_erros[variavel].value_counts())) +
                        " erros encontrados:\n" +
                        str(df_erros[variavel].value_counts()))
        # Imprimindo as linhas com problemas no log_output
        #log_output.warning("Linhas com erro:\n -----")
        #for index, linha in df_erros.iterrows():
        #    log_output.warning("Linha " + str(index+1) + ':\n => ' +
        #                        str(linha))
    else:
        log_tela.info(variavel + ": Nenhum erro encontrado.")

def verifica_range(df, variavel, valor_menor, valor_maior):
    """
    Verifica se uma variável, do tipo número inteiro, contém algum valor menor
    que "valor_menor" ou maior que "valor_maior".
    :param df: dataframe com os dados a serem verificados
    :param variavel: string com o nome da variável (coluna) que tem os dados
                    que devem ser verificados
    :param valor_menor: Valor mínimo esperado na variável (int ou float)
    :param valor_maior: Valor máximo esperado na variável (int ou float)
```

```

: return: Sem retorno, apenas salva as avaliações nos logs.
Uso:
    verifica_range(dataframe, 'nome_variavel', valor_menor, valor_maior)
"""
log_tela.info('Verificando Range da variável: ' + variavel)
# Obs: Registros inválidos: None (equivalente a NA)
log_output.info('\n\n' +
    '        - ' + 'Mínimo esperado: ' + str(valor_menor) + '\n' +
    '        - ' + 'Máximo esperado: ' + str(valor_maior) + '\n' +
    '        - ' + 'Total de registros: ' + str(len(df[variavel])) +
    '\n' +
    '        - ' + 'Registros nulos (NA): ' +
    str(df[variavel].isnull().sum()) + '\n'
    # '        - ' + 'Descrição da variável: \n' +
    # str(df[variavel].describe())
)

df_erros = df[(df[variavel] < valor_menor) | (df[variavel] > valor_maior)]

if len(df_erros[variavel].value_counts()) > 0:

    result = df_erros[variavel].value_counts().sort_index()
    #Verificando limite inferior
    if result.first_valid_index() < valor_menor:
        log_tela.warning(
            variavel + ': ' + 'Valor inteiro mínimo encontrado: ' +
            str(result.first_valid_index()) + ' - abaixo do esperado!')
    #Verificando limite superior
    if result.last_valid_index() > valor_maior:
        log_tela.warning(
            variavel + ': ' + 'Valor inteiro máximo encontrado: ' +
            str(result.last_valid_index()) + " - acima do esperado!")

    log_tela.warning(variavel + ': ' +
        str(len(df_erros[variavel].value_counts())) +
        ' valor(es) incorreto(s) ' +
        ' encontrado(s) nesta variável:\n' +
        str(df_erros[variavel].value_counts()))
    #log_output.warning("Linhas com erro:\n -----")
    #for index, linha in df_erros.iterrows():
    #    log_output.warning("Linha " + str(index+1) + ':\n => ' +
    #        str(linha))
else:
    log_tela.info(variavel + ": Nenhum erro encontrado.")

def pre_processamento(df):
    """
    Realiza algumas ações prévias ao processamento dos dados,
    removendo alguns registros e ajustando o dataframe.
    """

    log_tela.info('Criando/Renomeando colunas no dataframe principal')

```

```

log_output.info('Renomeando coluna UCOD para UCOD_DOM')
df.rename(columns={'UCOD': 'UCOD_DOM'}, inplace=True)

log_tela.info('Verificando se todas as variaveis esperadas'+
              'existem no dataframe.\n' +
              'Caso não exista alguma, ela é criada.')
variaveis = ['ANO', 'CD_ENTRE', 'DIA_SEM', 'UCOD_DOM', 'ZONA_DOM',
             'SUBZONA_DOM', 'MUN_DOM', 'CO_DOM_X', 'CO_DOM_Y', 'ID_DOM',
             'F_DOM', 'FE_DOM', 'NO_DOM', 'TIPO_DOM', 'TOT_FAM', 'ID_FAM',
             'F_FAM', 'FE_FAM', 'NO_FAM', 'COND_MORA', 'QT_AUTO', 'QT_BICI',
             'QT_MOTO', 'CD_RENFAM', 'REN_FAM', 'ID_PESS', 'F_PESS',
             'FE_PESS', 'NO_PESS', 'SIT_FAM', 'IDADE', 'SEXO', 'ESTUDA',
             'GRAU_INSTR', 'OCUP', 'SETOR_ATIV', 'CD_RENIND', 'REN_IND',
             'UCOD_ESC', 'ZONA_ESC', 'SUBZONA_ESC', 'MUN_ESC', 'CO_ESC_X',
             'CO_ESC_Y', 'UCOD_TRAB1', 'ZONA_TRAB1', 'SUBZONA_TRAB1',
             'MUN_TRAB1', 'CO_TRAB1_X', 'CO_TRAB1_Y', 'UCOD_TRAB2',
             'ZONA_TRAB2', 'SUBZONA_TRAB2', 'MUN_TRAB2', 'CO_TRAB2_X',
             'CO_TRAB2_Y', 'ID_VIAG', 'F_VIAG', 'FE_VIAG', 'NO_VIAG',
             'TOT_VIAG', 'UCOD_ORIG', 'ZONA_ORIG', 'SUBZONA_ORIG',
             'MUN_ORIG', 'CO_ORIG_X', 'CO_ORIG_Y', 'UCOD_DEST', 'ZONA_DEST',
             'SUBZONA_DEST', 'MUN_DEST', 'CO_DEST_X', 'CO_DEST_Y',
             'DIST_VIAG', 'SERV_PAS_ORIG', 'SERV_PAS_DEST', 'MOTIVO_ORIG',
             'MOTIVO_DEST', 'MOD01', 'MOD02', 'MOD03', 'MOD04', 'MODO_PRIN',
             'TIPO_VIAG', 'H_SAIDA', 'MIN_SAIDA', 'ANDA_ORIG', 'H_CHEG',
             'MIN_CHEG', 'ANDA_DEST', 'DURACAO', 'TIPO_EST_AUTO',
             'VALOR_EST_AUTO']

for variavel in variaveis:
    if variavel not in df.columns:
        # Se a variável não existe no dataframe então ela é criada
        # com valor padrão de NONE (NA)
        df[variavel] = None
        log_tela.info('Criando a variavel ' + variavel + ' no dataframe')

log_output.info('Reordenando as variáveis')
df = df[['ANO', 'CD_ENTRE', 'DIA_SEM', 'UCOD_DOM', 'ZONA_DOM',
        'SUBZONA_DOM', 'MUN_DOM', 'CO_DOM_X', 'CO_DOM_Y', 'ID_DOM',
        'F_DOM', 'FE_DOM', 'NO_DOM', 'TIPO_DOM', 'TOT_FAM', 'ID_FAM',
        'F_FAM', 'FE_FAM', 'NO_FAM', 'COND_MORA', 'QT_AUTO', 'QT_BICI',
        'QT_MOTO', 'CD_RENFAM', 'REN_FAM', 'ID_PESS', 'F_PESS',
        'FE_PESS', 'NO_PESS', 'SIT_FAM', 'IDADE', 'SEXO', 'ESTUDA',
        'GRAU_INSTR', 'OCUP', 'SETOR_ATIV', 'CD_RENIND', 'REN_IND',
        'UCOD_ESC', 'ZONA_ESC', 'SUBZONA_ESC', 'MUN_ESC', 'CO_ESC_X',
        'CO_ESC_Y', 'UCOD_TRAB1', 'ZONA_TRAB1', 'SUBZONA_TRAB1',
        'MUN_TRAB1', 'CO_TRAB1_X', 'CO_TRAB1_Y', 'UCOD_TRAB2',
        'ZONA_TRAB2', 'SUBZONA_TRAB2', 'MUN_TRAB2', 'CO_TRAB2_X',
        'CO_TRAB2_Y', 'ID_VIAG', 'F_VIAG', 'FE_VIAG', 'NO_VIAG',
        'TOT_VIAG', 'UCOD_ORIG', 'ZONA_ORIG', 'SUBZONA_ORIG',
        'MUN_ORIG', 'CO_ORIG_X', 'CO_ORIG_Y', 'UCOD_DEST', 'ZONA_DEST',
        'SUBZONA_DEST', 'MUN_DEST', 'CO_DEST_X', 'CO_DEST_Y',
        'DIST_VIAG', 'SERV_PAS_ORIG', 'SERV_PAS_DEST', 'MOTIVO_ORIG',
        'MOTIVO_DEST', 'MOD01', 'MOD02', 'MOD03', 'MOD04', 'MODO_PRIN',
        'TIPO_VIAG', 'H_SAIDA', 'MIN_SAIDA', 'ANDA_ORIG', 'H_CHEG',
        'MIN_CHEG', 'ANDA_DEST', 'DURACAO', 'TIPO_EST_AUTO',

```

```

        'VALOR_EST_AUTO']]

log_tela.info('Removendo registro com CD_RENFAM==5.')
df = df.drop(df[df['CD_RENFAM']==5].index)

log_tela.info('Removendo registros com SEXO==0.')
df = df.drop(df[df['SEXO']==0].index)

log_output.info('\n\n=====')
log_output.info('\n\n=====')

return df

def coordenadas(passo, df):
    """
    Itera sobre os registros do dataframe coords,
    a cada iteração, utiliza o valor da coluna ZONA
    para filtrar o dataframe df em cada um dos tipos de ZONA
    (DOM, ESC, TRAB1, TRAB2, ORIG, DEST) e substitui os valores
    das coordenadas X e Y de cada tipo (CO_DOM_X, CO_ESC_Y, etc) com
    os valores das colunas CO_X e CO_Y do dataframe coords.

    :param passo: número do passo para o log
    :param df: Dataframe a ser modificado (ex.: od1977)
    :return: retorna o dataframe modificado com todas as coordenadas aplicadas
    """
    log_tela.info("### PASSO " + str(passo) + " - COORDENADAS")

    def coord_aux(row):
        """
        Esta função irá receber uma linha (row) do dataframe coords e
        utilizar os valores desta linha para realizar as alterações
        de coordenadas no dataframe df.
        """
        # Começando pelo tipo DOM, alterando CO_DOM_X e depois CO_DOM_Y
        df.loc[df['ZONA_DOM'] == row['ZONA'], 'CO_DOM_X'] = row['CO_X']
        df.loc[df['ZONA_DOM'] == row['ZONA'], 'CO_DOM_Y'] = row['CO_Y']
        # Agora com o tipo ESC, alterando CO_ESC_X e depois CO_ESC_Y
        df.loc[df['ZONA_ESC'] == row['ZONA'], 'CO_ESC_X'] = row['CO_X']
        df.loc[df['ZONA_ESC'] == row['ZONA'], 'CO_ESC_Y'] = row['CO_Y']
        # Agora com o tipo TRAB1, alterando CO TRAB1_X e depois CO TRAB1_Y
        df.loc[df['ZONA TRAB1'] == row['ZONA'], 'CO TRAB1_X'] = row['CO_X']
        df.loc[df['ZONA TRAB1'] == row['ZONA'], 'CO TRAB1_Y'] = row['CO_Y']
        # Agora com o tipo TRAB2, alterando CO TRAB2_X e depois CO TRAB2_Y
        df.loc[df['ZONA TRAB2'] == row['ZONA'], 'CO TRAB2_X'] = row['CO_X']
        df.loc[df['ZONA TRAB2'] == row['ZONA'], 'CO TRAB2_Y'] = row['CO_Y']
        # Agora com o tipo ORIG, alterando CO ORIG_X e depois CO ORIG_Y
        df.loc[df['ZONA ORIG'] == row['ZONA'], 'CO ORIG_X'] = row['CO_X']
        df.loc[df['ZONA ORIG'] == row['ZONA'], 'CO ORIG_Y'] = row['CO_Y']
        # Agora com o tipo DEST, alterando CO DEST_X e depois CO DEST_Y
        df.loc[df['ZONA DEST'] == row['ZONA'], 'CO DEST_X'] = row['CO_X']
        df.loc[df['ZONA DEST'] == row['ZONA'], 'CO DEST_Y'] = row['CO_Y']

```

```

log_output.info('Lendo arquivo auxiliar de Coordenadas das subzonas')
coords = pd.read_csv('auxiliares/coord_zonas_1977.csv', sep=';', decimal=',')

# Esta variável out não é utilizada para nada além de evitar um
# monte de output que não será utilizado e que é gerado pelo método apply.
out = coords.apply(coord_aux, axis=1)

log_output.info('\n\n=====')

return df

```

## 3.2 Funções Gerais

Função/Variável	Status
passo_ano	OK
passo_dia_sem	OK
passo_ucods	OK

```
In [ ]: log_tela.info('Definindo as funções Gerais')
log_output.info('\n\n=====\\n')

def passo_ano(passo, df):
    """
    Preenche a coluna "ANO" com valor 1 em todas células
    Categorias:
    /valor/ano_correspondente/
    /-----/-----/
    /1/1977/
    /2/1987/
    /3/1997/
    /4/2007/

    :param passo: Número do passo atual para registro/log
    :param df: dataframe a ser modificado
    :return: retorna o dataframe modificado
    """

    log_tela.info("### PASSO " + str(passo) + " - ANO")

    # Definindo valor '1' para todas as células da coluna ANO
    df["ANO"] = 1

    return df

def passo_dia_sem(passo, df):
    """
    Não existe essa informação no banco de dados de 1977, logo, este campo será
    preenchido com 'None' (NA).

    # ####Categorias:
    # Valor/Descrição
    # -----/-----
    # 0/Não disponível
    # 2/Segunda-Feira
    # 3/Terça-Feira
    # 4/Quarta-Feira
    # 5/Quinta-Feira
    # 6/Sexta-Feira

    :param passo: Número do passo atual para registro/log
```

```

:param df: dataframe a ser modificado
:return: retorna o dataframe modificado
"""

log_tela.info("### PASSO " + str(passo) + " - DIA_SEM")

# Atribuindo "None" (Nulo/NA) para todos os registros
df['DIA_SEM'] = None

return df

def passo_ucods(passo, df):
    """
    Itera sobre os registros do dataframe ucods,
    a cada iteração utiliza o valor da coluna ZONA_REF
    para filtrar o dataframe df em cada um dos tipos de ZONA
    (DOM, ESC, TRAB1, TRAB2, ORIG, DEST) e substitui o valor
    da respectiva UCOD.

    :param passo: número do passo para o log
    :param df: Dataframe a ser modificado (ex.: od1977)
    :return: retorna o dataframe modificado com todas as UCODS aplicadas
    """

    log_tela.info("### PASSO " + str(passo) + " - UCODS")

    def ucod_aux(row):
        df.loc[df['ZONA_DOM']==row['ZONA_REF'], 'UCOD_DOM'] = row['UCOD_BUSCADA']
        df.loc[df['ZONA_ESC']==row['ZONA_REF'], 'UCOD_ESC'] = row['UCOD_BUSCADA']
        df.loc[df['ZONA_TRAB1']==row['ZONA_REF'], 'UCOD_TRAB1'] = row['UCOD_BUSCADA']
        df.loc[df['ZONA_TRAB2']==row['ZONA_REF'], 'UCOD_TRAB2'] = row['UCOD_BUSCADA']
        df.loc[df['ZONA_ORIG']==row['ZONA_REF'], 'UCOD_ORIG'] = row['UCOD_BUSCADA']
        df.loc[df['ZONA_DEST']==row['ZONA_REF'], 'UCOD_DEST'] = row['UCOD_BUSCADA']

    log_output.info('Lendo arquivo auxiliar UCOD')
    ucods = pd.read_csv('auxiliares/UCOD-1977.csv', sep=';', decimal=',')

    # Esta variável out não é utilizada para nada além de evitar um
    # monte de output que não será utilizado e que é gerado pelo método apply.
    out = ucods.apply(ucod_aux, axis=1)

    # Verificando intervalo de valores - condições:
    # "UCOD_XXX >= 1" E "UCOD_XXX <= 67"
    for tipo in ['DOM', 'ESC', 'TRAB1', 'TRAB2', 'ORIG', 'DEST']:
        verifica_range(df, 'UCOD_' + tipo, 1, 67)
    log_output.info('\n\n=====')

    return df

```

### 3.3 Funções do Domicílio

Função/Variável	Status
passo_zona_dom	OK



Função/Variável	Status
passo_subzona_dom	OK
passo_mun_dom	OK
passo_f_dom	OK
passo_fe_dom	OK
passo_tipo_dom	OK

```
In [ ]: log_tela.info('Definindo as funções do Domicílio')
log_output.info('\n\n=====\\n')

def passo_zona_dom(passo, df):
    """
    Checar se existe algum erro

    # ####Categorias:
    # > 1 a 243

    [Teste: Checar se existe algum número < 1 ou > 243.
     Se encontrar, retornar erro indicando em qual linha.]
    :param passo: passo
    :param df: dataframe
    :return: sem retorno
    """

    # Verificando intervalo de valores - condições:
    # "ZONA_DOM >= 1" E "ZONA_DOM <= 243"
    verifica_range(df, 'ZONA_DOM', 1, 243)
    log_output.info('\n\n=====\\n')

    return df

def passo_subzona_dom(passo, df):
    """
    Checar se existe algum erro

    # ####Categorias:
    # > 1 a 633

    [Teste: Checar se existe algum número < 1 ou > 633.
     Se encontrar, retornar erro indicando em qual linha.]
    :param passo: Número do passo atual para registro/log
    :param df:
    :return: sem retorno
    """

    log_tela.info("### PASSO " + str(passo) + " - SUBZONA_DOM")

    # Verificando intervalo de valores - condições:
    # "SUBZONA_DOM >= 1" E "SUBZONA_DOM <= 633"
```

```

verifica_range(df, 'SUBZONA_DOM', 1, 633)
log_output.info('\n\n=====\\n')

return df

def passo_mun_dom(passo, df):
    """
    Checar se existe algum erro

    # ####Categorias
    # > 1 a 27

    [Teste: Checar se existe algum número < 1 ou > 27.
     Se encontrar, retornar erro indicando em qual linha.]
    :param passo: Número do passo atual para registro/log
    :param df:
    :return: sem retorno
    """
    log_tela.info("### PASSO " + str(passo) + " - MUN_DOM")

    # Verificando intervalo de valores - condições:
    # "MUN_DOM >= 1" E "MUN_DOM <= 27"
    verifica_range(df, 'MUN_DOM', 1, 27)
    log_output.info('\n\n=====\\n')

    return df

def passo_f_dom(passo, df):
    """
    Checar se existe algum erro na coluna "F_DOM"

    # ####Categorias
    # Valor/Descrição
    # ----|----
    # 0/Demais registros
    # 1/Primeiro Registro do Domicílio

    [Teste: Checar se existe algum número diferente de 0 ou 1.
     Se encontrar, retornar erro indicando em qual linha.]
    :param passo: Número do passo atual para registro/log
    :param df:
    :return: sem retorno
    """
    log_tela.info("### PASSO " + str(passo) + " - F_DOM")

    verifica_dummy(df, 'F_DOM')
    log_output.info('\n\n=====\\n')

    return df

def passo_fe_dom(passo, df):

```

```

"""
Nada há que se fazer em relação aos dados da coluna "FE_DOM"

:param passo: Número do passo atual para registro/log
:param df:
:return: sem retorno
"""
log_tela.info("### PASSO " + str(passo) + " - FE_DOM")
log_output.info('\n\n=====\\n')

return df

def passo_tipo_dom(passo, df):
    """
    Checar se existe algum erro

    # ####Categorias anteriores / novas
    # Valor | Descrição
    # ----|----
    # 0/Não respondeu
    # 1/Particular
    # 2/Coletivo

    [Teste: Checar se existe algum número < 0 ou > 2.
     Se encontrar, retornar erro indicando em qual linha.]
    :param passo: Número do passo atual para registro/log
    :param df:
    :return:
    """
    log_tela.info("### PASSO " + str(passo) + " - TIPO_DOM")

    # Verificando intervalo de valores - condições:
    # "TIPO_DOM >= 0" E "TIPO_DOM <= 2"
    verifica_range(df, 'TIPO_DOM', 0, 2)
    log_output.info('\n\n=====\\n')

    return df

```

### 3.4 Funções da Família

Função/Variável	Status
passo_tot_fam	OK
passo_f_fam	OK
passo_fe_fam	OK
passo_cond_mora	OK
passo_qt_auto	OK
passo_qt_bici	OK
passo_qt_moto	OK
passo_ren_fam	OK
passo_cd_renfam	OK

```
In [ ]: log_tela.info('Definindo as funções da Família')
        log_output.info('\n\n===== \n')

def passo_tot_fam(passo, df):
    """
    Nada há que se fazer em relação aos dados da coluna "TOT_FAM"
    :param passo: Número do passo atual para registro/log
    :param df:
    :return: sem retorno
    """
    log_tela.info("### PASSO " + str(passo) + " - TOT_FAM")
    log_output.info('\n\n===== \n')

    return df

def passo_f_fam(passo, df):
    """
    Checar se existe algum erro na coluna "F_FAM"

    # ####Categorias
    # Valor/Descrição
    # ----/----
    # 0/Demais registros
    # 1/Primeiro Registro da Família

    [Teste: Checar se existe algum número diferente de 0 ou 1.
     Se encontrar, retornar erro indicando em qual linha.]
    :param passo: Número do passo atual para registro/log
    :param df:
    :return:
    """
    log_tela.info("### PASSO " + str(passo) + " - F_FAM")

    verifica_dummy(df, 'F_FAM')
```

```

log_output.info('\n\n=====\\n')

return df

def passo_fe_fam(passo, df):
    """
    Nada há que se fazer em relação aos dados da coluna "FE_FAM"
    :param passo: Número do passo atual para registro/log
    :param df:
    :return:
    """
    log_tela.info("### PASSO " + str(passo) + " - FE_FAM")
    log_output.info('\n\n=====\\n')

    return df

def passo_cond_mora(passo, df):
    """
    Substituir valores da coluna "COND_MORA"

    # * Substituir todos valores **1** por **2**
    # * Substituir todos valores **3** por **1**
    # * Substituir todos valores **4** por **3**
    # * Substituir todos valores **5** por **3**
    # * Substituir todos valores **9** por **0**

    # ####Categorias anteriores
    # Valor/Descrição
    # ----/----
    # 0/Não respondeu
    # 1/Própria paga
    # 2/Própria em pagamento
    # 3/Alugada
    # 4/Cedida
    # 5/Outro
    # 9/Erro no preenchimento do banco de dados

    # ####Categorias novas
    # Valor/Descrição
    # ----/----
    # 0/Não respondeu
    # 1/Alugada
    # 2/Própria
    # 3/Outros

    [Teste: Checar se existe algum número < 0 ou > 3.
     Se encontrar, retornar erro indicando em qual linha.]
    :param passo: passo
    :param df: dataframe
    :return: dataframe modificado
    """
    log_tela.info("### PASSO " + str(passo) + " - COND_MORA")

```

```

# Substituindo valor 1 por 2
df.loc[df['COND_MORA']==1,'COND_MORA'] = 2
# Substituindo valor 3 por 1
df.loc[df['COND_MORA']==3,'COND_MORA'] = 1
# Substituindo valor 4 por 3
df.loc[df['COND_MORA']==4,'COND_MORA'] = 3
# Substituindo valor 5 por 3
df.loc[df['COND_MORA']==5,'COND_MORA'] = 3
# Substituindo valor 9 por 0
df.loc[df['COND_MORA']==9,'COND_MORA'] = 0

# Verificando intervalo de valores - condições:
# "COND_MORA >= 0" E "COND_MORA <= 3"
verifica_range(df, 'COND_MORA', 0, 3)
log_output.info('\n\n=====')

return df

def passo_qt_auto(passo, df):
    """
    Nada há que se fazer em relação aos dados da coluna "QT_AUTO"
    :param passo: Número do passo atual para registro/log
    :param df:
    :return:
    """
    log_tela.info("### PASSO " + str(passo) + " - QT_AUTO")
    log_output.info('\n\n=====')

    return df

def passo_qt_bici(passo, df):
    """
    Não existe essa informação no banco de dados de 1977, logo,
    este campo será preenchido com 'None'.
    :param passo: Número do passo atual para registro/log
    :param df:
    :return: dataframe modificado
    """
    df['QT_BICI'] = None

    # Contando "QT_BICI" para verificar os valores após o procedimento
    log_tela.info('### PASSO ' + str(passo) + ' - QT_BICI' + '\n'
                  + ' ' + 'Total de registros: ' +
                  str(len(df['QT_BICI'])) + '\n' +
                  + ' ' + 'Soma de bicicletas "nulas": ' +
                  str(df['QT_BICI'].isnull().sum()))

    log_output.info('\n\n=====')

    return df

```

```

def passo_qt_moto(passo, df):
    """
    Não existe essa informação no banco de dados de 1977, logo,
    este campo será preenchido com 'None'.
    :param passo: Número do passo atual para registro/log
    :param df:
    :return: dataframe modificado
    """
    df['QT_MOTO'] = None

    # Contando "QT_MOTO" para verificar os valores após o procedimento
    log_tela.info('### PASSO ' + str(passo) + ' - QT_MOTO' + '\n'
                  + ' ' + 'Total de registros: ' +
                  str(len(df['QT_MOTO'])) + '\n' +
                  + ' ' + 'Soma de motos "nulas": ' +
                  str(df['QT_MOTO'].isnull().sum()))

    log_output.info('\n\n=====')

    return df

def passo_ren_fam(passo, df, deflator):
    """
    Corrige a renda familiar de acordo com o deflator passado na função.
    :param passo: Número do passo atual para registro/log
    :param df: dataframe
    :param deflator: Deflator utilizado para correção da renda.
    :return: sem retorno
    """
    log_tela.info("### PASSO " + str(passo) + " - REN_FAM")

    df['REN_FAM'] = df['REN_FAM'] * deflator

    return df

def passo_cd_renfam(passo, df):
    """
    Substituir valores da coluna "CD_RENFAM"

    * Substituir todos valores **1** por **2**
    * Substituir todos valores **3** por **2**
        Quando categoria original for 0 (respondeu)
            checar no campo REN_FAM se o valor é nulo.
            Se for nulo, manter na categoria 0 (Renda Familiar Declarada como Zero),
            senão, mover para a categoria 1 (Renda Familiar Declarada e
            Maior que Zero).

    # ####Categorias anteriores
    # Valor/Descrição
    # ----/----
    # 0/Respondeu
    """

```

```

# 1/Não Sabe
# 2/Não Respondeu
# 3/Não se Aplica

# ####Categorias novas
# Valor/Descrição
# ----/----
# 0/Renda Familiar Declarada como Zero
# 1/Renda Familiar Declarada e Maior que Zero
# 2/Renda Atribuída

[Teste: Checar se existe algum número < 0 ou > 2.
  Se encontrar, retornar erro indicando em qual linha.]
:param passo: Número do passo atual para registro/log
:param df:
:return: retorna o dataframe corrigido
"""
log_tela.info("### PASSO " + str(passo) + " - CD_RENFAM")

# Substituindo valor 1 (Não sabe) por 2 (Atribuída)
df.loc[df['CD_RENFAM']==1,'CD_RENFAM'] = 2

# Substituindo valor 3 (Não se Aplica) por 2 (Atribuída)
df.loc[df['CD_RENFAM']==3,'CD_RENFAM'] = 2

# Correção de REN_FAM
# REN_FAM só pode ser igual a zero se tiver sido declarada assim.
df.loc[df['REN_FAM']==0,'CD_RENFAM'] = 0

# Dividindo a categoria '0', "Respondeu", em:
# - 0 "Renda Familiar Declarada como Zero" e
# - 1 "Renda Familiar Declarada e Maior que Zero"
df.loc[(df['CD_RENFAM'] == 0) &
       (df['REN_FAM'] != 0) &
       (df['REN_FAM'].notnull()), 'CD_RENFAM'] = 1

# Verificando intervalo de valores - condições:
# "CD_RENFAM >= 0" E "CD_RENFAM <= 2"
verifica_range(df, 'CD_RENFAM', 0, 2)
log_output.info('\n\n=====\\n')

return df

```



### 3.5 Funções da pessoa

Função/Variável	Status
passo_f_pess	OK
passo_fe_pess	OK
passo_sit_fam	OK
passo_idade	OK
passo_sexo	OK
passo_grau_instr	OK
passo_ocup	OK
passo_setor_ativ	OK
passo_ren_ind	OK
passo_cd_renind	OK
passo_estuda	OK

```
In [ ]: log_tela.info('Definindo as funções da Pessoa')
        log_output.info('\n\n===== \n')

def passo_f_pess(passo, df):
    """
    Checar se existe algum erro na coluna "F_PESS"

    # ###Categorias
    # Valor/Descrição
    # ----/----
    # 0/Demais registros
    # 1/Primeiro Registro da Pessoa

    [Teste: Checar se existe algum número diferente de 0 ou 1.
     Se encontrar, retornar erro indicando em qual linha.]
    :param passo: Número do passo atual para registro/log
    :param df:
    :return:
    """
    log_tela.info("### PASSO " + str(passo) + " - F_PESS")

    verifica_dummy(df, 'F_PESS')
    log_output.info('\n\n===== \n')

    return df

def passo_fe_pess(passo, df):
    """
    Nada há que se fazer em relação aos dados das colunas "FE_PESS"
    :param passo: Número do passo atual para registro/log
    :param df:
```

```

: return:
"""
log_tela.info("### PASSO " + str(passo) + " - FE_PESS")
log_output.info('\n\n===== \n')

return df

def passo_sit_fam(passo, df):
    """
    Substituir todos valores **5** por **4**
    Substituir todos valores **6** por **5**
    Substituir todos valores **7** por **6**
    Substituir todos valores **9** por **6**
    Substituir todos valores **0** por **6** quando DURACAO!=0

    # ####Categorias anteriores
    # Valor/Descrição
    # ----/----
    # 1/Chefe
    # 2/Cônjuge
    # 3/Filho(a)
    # 4/Parente
    # 5/Agregado
    # 6/Empregado Residente
    # 7/Visitante não Residente

    # ####Categorias novas:
    # Valor/Descrição
    # ----/----
    # 0/ Não Respondeu/Não fez viagem
    # 1/ Pessoa Responsável
    # 2/ Cônjuge/Companheiro(a)
    # 3/ Filho(a)/Enteado(a)
    # 4/ Outro Parente / Agregado
    # 5/ Empregado Residente
    # 6/ Outros (visitante não residente / parente do empregado)

    [Teste: Checar se existe algum número < 0 ou > 6.
     Se encontrar, retornar erro indicando em qual linha.]
    :param passo: Número do passo atual para registro/log
    :param df:
    :return: retorna o dataframe modificado
    """
    log_tela.info("### PASSO " + str(passo) + " - SIT_FAM")

    # Substituindo valor 5 por 4
    df.loc[df['SIT_FAM']==5, 'SIT_FAM'] = 4
    # Substituindo valor 6 por 5
    df.loc[df['SIT_FAM']==6, 'SIT_FAM'] = 5
    # Substituindo valor 7 por 6
    df.loc[df['SIT_FAM']==7, 'SIT_FAM'] = 6
    # Substituindo valor 9 por 6
    df.loc[df['SIT_FAM']==9, 'SIT_FAM'] = 6

```

```

# Substituindo valor 0 por 6 quando DURACAO != 0
df.loc[(df['SIT_FAM']==9) & (df['DURACAO']!=0), 'SIT_FAM'] = 6

# Verificando intervalo de valores - condições:
# "SIT_FAM >= 0" E "SIT_FAM <= 6"
verifica_range(df, 'SIT_FAM', 0, 6)
log_output.info('\n\n=====\\n')

return df

def passo_idade(passo, df):
    """
    Nada há que se fazer em relação aos dados da coluna "IDADE"
    :param passo: Número do passo atual para registro/log
    :param df:
    :return:
    """
    log_tela.info("### PASSO " + str(passo) + " - IDADE")
    log_output.info('\n\n=====\\n')

    return df

def passo_sexo(passo, df):
    """
    #####Categorias anteriores
    Valor/Descrição
    ----/----
    1/Masculino
    2/Feminino

    #####Categorias novas
    Valor/Descrição
    ----/----
    0/Não Respondeu
    1/Masculino
    2/Feminino

    [Teste: Checar se existe algum número diferente de 0, 1 ou 2.
     Se encontrar, retornar erro indicando em qual linha.]
    :param passo: Número do passo atual para registro/log
    :param df:
    :return: retorna o dataframe modificado
    """
    log_tela.info("### PASSO " + str(passo) + " - SEXO")

    # Verificando intervalo de valores - condições:
    # "SEXO >= 0" E "SEXO <= 2"
    verifica_range(df, 'SEXO', 0, 2)
    log_output.info('\n\n=====\\n')

    return df

```

```

def passo_grau_instr(passo, df):
    """
    Substituir valores da coluna "GRAU_INSTR"

    Substituir todos valores **2** por **1**
    Substituir todos valores **3** por **1**
    Substituir todos valores **4** por **1**
    Substituir todos valores **5** por **2**
    Substituir todos valores **6** por **2**
    Substituir todos valores **7** por **3**
    Substituir todos valores **8** por **3**
    Substituir todos valores **9** por **4**

    # ####Categorias anteriores:
    # Valor/Descrição
    # ----/----
    # 1/Sem Instrução
    # 2/Primário Incompleto
    # 3/Primário Completo
    # 4/Ginasial Incompleto
    # 5/Ginasial Completo
    # 6/Colegial Incompleto
    # 7/Colegial Completo
    # 8/Universitário Incompleto
    # 9/Universitário Completo

    # ####Categorias novas
    # Valor/Descrição
    # ----/----
    # 0/Não declarou
    # 1/Não-Alfabetizado/Fundamental Incompleto
    # 2/Fundamental Completo/Médio Incompleto
    # 3/Médio Completo/Superior Incompleto
    # 4/Superior completo

    [Teste: Checar se existe algum número < 1 ou > 4.
     Se encontrar, retornar erro indicando em qual linha.]
    :param passo: Número do passo atual para registro/log
    :param df:
    :return: retorna dataframe modificado
    """
    log_tela.info("### PASSO " + str(passo) + " - GRAU_INSTR")

    # Substituindo valor 2 por 1
    df.loc[df['GRAU_INSTR']==2,'GRAU_INSTR'] = 1
    # Substituindo valor 3 por 1
    df.loc[df['GRAU_INSTR']==3,'GRAU_INSTR'] = 1
    # Substituindo valor 4 por 1
    df.loc[df['GRAU_INSTR']==4,'GRAU_INSTR'] = 1
    # Substituindo valor 5 por 2
    df.loc[df['GRAU_INSTR']==5,'GRAU_INSTR'] = 2
    # Substituindo valor 6 por 2
    df.loc[df['GRAU_INSTR']==6,'GRAU_INSTR'] = 2

```

```

# Substituindo valor 7 por 3
df.loc[df['GRAU_INSTR']==7,'GRAU_INSTR'] = 3
# Substituindo valor 8 por 3
df.loc[df['GRAU_INSTR']==8,'GRAU_INSTR'] = 3
# Substituindo valor 9 por 4
df.loc[df['GRAU_INSTR']==9,'GRAU_INSTR'] = 4

# Verificando intervalo de valores - condições:
# "GRAU_INSTR >= 0" E "GRAU_INSTR <= 4"
verifica_range(df, 'GRAU_INSTR', 0, 4)
log_output.info('\n\n=====')

return df

def passo_ocup(passo, df):
    """
    Substituir valores da coluna "OCUP"

    Somar 10 em todos valores (para facilitar a lógica).
    Substituir todos valores **10** por **0**
    Substituir todos valores **11** por **7**
    Substituir todos valores **12** por **6**
    Substituir todos valores **13** por **3**
    Substituir todos valores **14** por **5**
    Substituir todos valores **15** por **4**
    Substituir todos valores **16** por **2**
    Substituir todos valores *maiores do que 16* por **1**

    # ####Categorias anteriores:
    # Valor/Descrição
    # ----/----
    # 1/Estudante
    # 2/Prendas Domésticas
    # 3/Aposentado
    # 4/Sem Ocupação (nunca trabalhou)
    # 5/Desempregado
    # 6/Em licença
    # 7/em diante/diversas profissões

    # ####Categorias novas
    # Valor/Descrição
    # ----/----
    # 0/Não Respondeu
    # 1/Tem trabalho
    # 2/Em licença médica
    # 3/Aposentado / pensionista
    # 4/Desempregado
    # 5/Sem ocupação
    # 6/Dona de casa
    # 7/Estudante

    [Teste: Checar se existe algum número < 0 ou > 7.
    Se encontrar, retornar erro indicando em qual linha.]

```

```

:param passo: Número do passo atual para registro/log
:param df:
:return: retorna dataframe modificado
"""
log_tela.info("### PASSO " + str(passo) + " - OCUP")

df['OCUP'] = df['OCUP'] + 10
# Substituindo valor 10 por 0
df.loc[df['OCUP']==10,'OCUP'] = 0
# Substituindo valor 11 por 7
df.loc[df['OCUP']==11,'OCUP'] = 7
# Substituindo valor 12 por 6
df.loc[df['OCUP']==12,'OCUP'] = 6
# Substituindo valor 13 por 3
df.loc[df['OCUP']==13,'OCUP'] = 3
# Substituindo valor 14 por 5
df.loc[df['OCUP']==14,'OCUP'] = 5
# Substituindo valor 15 por 4
df.loc[df['OCUP']==15,'OCUP'] = 4
# Substituindo valor 16 por 2
df.loc[df['OCUP']==16,'OCUP'] = 2
# Substituindo valor >16 por 1
df.loc[df['OCUP']>16,'OCUP'] = 1

# Verificando intervalo de valores - condições:
# "OCUP >= 1" E "OCUP <= 7"
verifica_range(df, 'OCUP', 0, 7)
log_output.info('\n\n=====')

return df

def passo_setor_ativ(passo, df):
    """
    Substituir valores da coluna "SETOR_ATIV"

    Na coluna "SETOR_ATIV", linha i,
        ler o valor da linha i da coluna "SETOR_ATIV", daí,
        buscar o mesmo valor na coluna "COD" do arquivo setor_ativ-1977.csv.
    Ao achar, retornar o valor da mesma linha, só que da coluna "COD_UNIF"

    #####Categorias anteriores
    > ver arquivo .csv

    #####Categorias novas
    Valor/Descrição
    ----/----
    0/Não respondeu
    1/Agrícola
    2/Construção Civil
    3/Indústria
    4/Comércio
    5/Administração Pública
    6/Serviços de Transporte

```

```

7/Outros serviços
8/Outros
9/Não se aplica

[Teste: Checar se existe algum número < 0 ou > 9.
  Se encontrar, retornar erro indicando em qual linha.]
:param passo: Número do passo atual para registro/log
:param df:
:return: retorna dataframe modificado
"""
log_tela.info("### PASSO " + str(passo) + " - SETOR_ATIV")

log_output.info('Lendo arquivo de referência externa setor_ativ-1977.csv')
df_setor = pd.read_csv('auxiliares/setor_ativ-1977.csv', sep=';', decimal=',')

def setor_aux(row):
    df.loc[df['SETOR_ATIV']==row['COD'], 'SETOR_ATIV'] = row['COD_UNIF']

# Esta variável out não é utilizada para nada além de evitar um
# monte de output que não será utilizado e que é gerado pelo método apply.
out = df_setor.apply(setor_aux, axis=1)

# Verificando intervalo de valores - condições:
# "SETOR_ATIV >= 0" E "SETOR_ATIV <= 9"
verifica_range(df, 'SETOR_ATIV', 0, 9)
log_output.info('\n\n=====')

return df

def passo_ren_ind(passo, df, deflator):
    """
    Corrige a renda individual de acordo com o deflator passado na função.
    :param passo: Número do passo atual para registro/log
    :param df: dataframe
    :param deflator: Deflator utilizado para correção da renda.
    :return: sem retorno
    """
    log_tela.info("### PASSO " + str(passo) + " - REN_IND")

    df['REN_IND'] = df['REN_IND'] * deflator

    return df

def passo_cd_renind(passo, df):
    """
    Substituir **2** por **3**
    Substituir **1** por **2**
    Substituir **0** por **1**
    Substituir **3** (erro de preenchimento) por **0**

    ##### Categorias anteriores efetivas
    Valor Real/ Descrição /Valor Layout

```

```

-----/-----/-----
0      /Tem renda      /1
1      /Não tem renda/2
2      /Não declarou /3

####Categorias novas
Valor/Descrição
-----/-----
0      /Não declarou
1      /Tem renda
2      /Não tem renda

[Teste: Checar se existe algum número < 0 ou > 2.
Se encontrar, retornar erro indicando em qual linha.]
:param passo: Número do passo atual para registro/log
:param df:
:return: sem retorno
"""
log_tela.info("### PASSO " + str(passo) + " - CD_RENIND")

df.loc[df['CD_RENIND']==2,'CD_RENIND'] = 3
df.loc[df['CD_RENIND']==1,'CD_RENIND'] = 2
df.loc[df['CD_RENIND']==0,'CD_RENIND'] = 1
df.loc[df['CD_RENIND']==3,'CD_RENIND'] = 0

# Verificando intervalo de valores - condições:
# "CD_RENIND >= 0" E "CD_RENIND <= 2"
verifica_range(df, 'CD_RENIND', 0, 2)
log_output.info('\n\n=====\\n')

return df

def passo_estuda(passo, df):
    """
    Considera-se que ninguém estuda, exceto quem forneceu informação
    de localização da Escola (Zona/SubZona)

    #####
    #          ATENÇÃO          #
    # ESTA FUNÇÃO SÓ DEVE SER #
    # EXECUTADA APÓS O PASSO #
    #          ZONA_ESC          #
    #####

    ####Categorias novas
    Valor/Descrição
    ----/----
    0/Não estuda
    1/Estuda

[Teste: Checar se existe algum número diferente de 0 ou 1.
Se encontrar, retornar erro indicando em qual linha.]
:param passo: Número do passo atual para registro/log

```



```

:param df:
:return: retorna dataframe com campo ESCOLA resolvido
"""
log_tela.info("### PASSO " + str(passo) + " - ESTUDA")

df['ESTUDA'] = 0

# Substituindo todos que declararam zona escola diferente de zero
# com campo ESTUDA igual a 1.
df.loc[df['ZONA_ESC'] != 0, 'ESTUDA'] = 1

verifica_dummy(df, 'ESTUDA')
log_output.info('\n\n=====')

return df

```

### 3.6 Funções da Viagem

Função/Variável	Status
passo_f_viag	OK
passo_fe_viag	OK
passo_zona_esc	OK
passo_subzona_esc	OK
passo_mun_esc	OK
passo_zona_trab1	OK
passo_subzona_trab1	OK
passo_mun_trab1	OK
passo_zona_trab2	OK
passo_subzona_trab2	OK
passo_mun_trab2	OK
passo_zona_orig	OK
passo_subzona_orig	OK
passo_mun_orig	OK
passo_zona_dest	OK
passo_subzona_dest	OK
passo_mun_dest	OK
passo_serv_pas_orig	OK
passo_serv_pas_dest	OK
passo_motivo_orig	OK
passo_motivo_dest	OK
passo_mod01	OK
passo_mod02	OK
passo_mod03	OK
passo_mod04	OK
passo_mod0_prin	OK
passo_tipo_est_auto	OK
passo_valor_est_auto	OK
passo_dist_viag	OK
passo_tot_viag	OK

Obs: O passo\_dist\_viag só deve ser executado após os passos que calculam as coordenadas.

Obs2: O passo\_tot\_viag deve ser executado após os passos que geram id's.

```
In [ ]: log_tela.info('Definindo funções referentes às viagens')
        log_output.info('\n\n=====\\n')
```

```

def passo_f_viag(passo, df):
    """
    Checar se existe algum erro na coluna "F_VIAG"

    # ####Categorias
    # Valor/Descrição
    # ----/----
    # 0/Não fez viagem
    # 1/Fez viagem

    [Teste: Checar se existe algum número diferente de 0 ou 1.
     Se encontrar, retornar erro indicando em qual linha.]
    :param passo: Número do passo atual para registro/log
    :param df:
    :return: sem retorno
    """
    log_tela.info("### PASSO " + str(passo) + " - F_VIAG")

    # Setando F_VIAG=0 se DURACAO==0
    df.loc[df['DURACAO']==0, 'F_VIAG'] = 0

    verifica_dummy(df, 'F_VIAG')
    log_output.info('\n\n=====\\n')

    return df

def passo_fe_viag(passo, df):
    """
    # Nada há que se fazer em relação aos dados da coluna "FE_VIAG"
    :param passo: Número do passo atual para registro/log
    :param df:
    :return: sem retorno
    """
    log_tela.info("### PASSO " + str(passo) + " - FE_VIAG")
    log_output.info('\n\n=====\\n')

    return df

def passo_zona_esc(passo, df):
    """
    Checar se existe algum erro

    # ####Categorias:
    # > 1 a 243

    [Teste: Checar se existe algum número < 1 ou > 243.
     Se encontrar, retornar erro indicando em qual linha.]
    :param passo: Número do passo atual para registro/log
    :param df:
    :return: Sem retorno
    """

```

```

log_tela.info("### PASSO " + str(passo) + " - ZONA_ESC")

# Verificando intervalo de valores - condições:
# "ZONA_ESC >= 1" E "ZONA_ESC <= 243"
verifica_range(df, 'ZONA_ESC', 1, 243)
log_output.info('\n\n=====\\n')

return df

def passo_subzona_esc(passo, df):
    """
    Checar se existe algum erro

    # ####Categorias:
    # > 1 a 633

    [Teste: Checar se existe algum número < 1 ou > 633.
     Se encontrar, retornar erro indicando em qual linha.]
    :param passo: Número do passo atual para registro/log
    :param df:
    :return: sem retorno
    """
    log_tela.info("### PASSO " + str(passo) + " - SUBZONA_ESC")

    # Verificando intervalo de valores - condições:
    # "SUBZONA_ESC >= 1" E "SUBZONA_ESC <= 633"
    verifica_range(df, 'SUBZONA_ESC', 1, 633)
    log_output.info('\n\n=====\\n')

    return df

def passo_mun_esc(passo, df):
    """
    Checar se existe algum erro

    # ####Categorias
    # > 1 a 27

    [Teste: Checar se existe algum número < 1 ou > 27.
     Se encontrar, retornar erro indicando em qual linha.]
    :param passo: Número do passo atual para registro/log
    :param df:
    :return: sem retorno
    """
    log_tela.info("### PASSO " + str(passo) + " - MUN_ESC")

    # Verificando intervalo de valores - condições:
    # "MUN_ESC >= 1" E "MUN_ESC <= 27"
    verifica_range(df, 'MUN_ESC', 1, 27)
    log_output.info('\n\n=====\\n')

    return df

```

```

def passo_zona_trab1(passo, df):
    """
    Checar se existe algum erro

    # ####Categorias:
    # > 1 a 243

    [Teste: Checar se existe algum número < 1 ou > 243.
     Se encontrar, retornar erro indicando em qual linha.]
    :param passo: Número do passo atual para registro/log
    :param df:
    :return: sem retorno
    """
    log_tela.info("### PASSO " + str(passo) + " - ZONA_TRAB1")

    # Verificando intervalo de valores - condições:
    # "ZONA_TRAB1 >= 1" E "ZONA_TRAB1 <= 243"
    verifica_range(df, 'ZONA_TRAB1', 1, 243)
    log_output.info('\n\n=====\\n')

    return df

def passo_subzona_trab1(passo, df):
    """
    Checar se existe algum erro

    # ####Categorias:
    # > 1 a 633

    [Teste: Checar se existe algum número < 1 ou > 633.
     Se encontrar, retornar erro indicando em qual linha.]
    :param passo: Número do passo atual para registro/log
    :param df:
    :return: sem retorno
    """
    log_tela.info("### PASSO " + str(passo) + " - SUBZONA_TRAB1")

    # Verificando intervalo de valores - condições:
    # "SUBZONA_TRAB1 >= 1" E "SUBZONA_TRAB1 <= 633"
    verifica_range(df, 'SUBZONA_TRAB1', 1, 633)
    log_output.info('\n\n=====\\n')

    return df

def passo_mun_trab1(passo, df):
    """
    Checar se existe algum erro

    # ####Categorias
    # > 1 a 27

```

```

    [Teste: Checar se existe algum número < 1 ou > 27.
      Se encontrar, retornar erro indicando em qual linha.]
:param passo: Número do passo atual para registro/log
:param df:
:return: sem retorno
"""
log_tela.info("### PASSO " + str(passo) + " - MUN_TRAB1")

# Verificando intervalo de valores - condições:
# "MUN_TRAB1 >= 1" E "MUN_TRAB1 <= 27"
# trips that are not work purposed
verifica_range(df, 'MUN_TRAB1', 1, 27)
log_output.info('\n\n=====\\n')

return df

def passo_zona_trab2(passo, df):
    """
    Checar se existe algum erro

    # ####Categorias:
    # > 1 a 243

    [Teste: Checar se existe algum número < 1 ou > 243.
      Se encontrar, retornar erro indicando em qual linha.]
:param passo: Número do passo atual para registro/log
:param df:
:return: sem retorno
"""
log_tela.info("### PASSO " + str(passo) + " - ZONA_TRAB2")

# Verificando intervalo de valores - condições:
# "ZONA_TRAB2 >= 1" E "ZONA_TRAB2 <= 243"
verifica_range(df, 'ZONA_TRAB2', 1, 243)
log_output.info('\n\n=====\\n')

return df

def passo_subzona_trab2(passo, df):
    """
    Checar se existe algum erro

    # ####Categorias:
    # > 1 a 633

    [Teste: Checar se existe algum número < 1 ou > 633.
      Se encontrar, retornar erro indicando em qual linha.]
:param passo: Número do passo atual para registro/log
:param df:
:return: sem retorno
"""

```

```

log_tela.info("### PASSO " + str(passo) + " - SUBZONA_TRAB2")

# Verificando intervalo de valores - condições:
# "SUBZONA_TRAB2 >= 1" E "SUBZONA_TRAB2 <= 633"
verifica_range(df, 'SUBZONA_TRAB2', 1, 633)
log_output.info('\n\n=====')

return df

def passo_mun_trab2(passo, df):
    """
    Checar se existe algum erro

    # ####Categorias
    # > 1 a 27

    [Teste: Checar se existe algum número < 1 ou > 27.
     Se encontrar, retornar erro indicando em qual linha.]
    :param passo: Número do passo atual para registro/log
    :param df:
    :return: sem retorno
    """
    log_tela.info("### PASSO " + str(passo) + " - MUN_TRAB2")

    # Verificando intervalo de valores - condições:
    # "MUN_TRAB2 >= 1" E "MUN_TRAB2 <= 27"
    verifica_range(df, 'MUN_TRAB2', 1, 27)
    log_output.info('\n\n=====')

    return df

def passo_zona_orig(passo, df):
    """
    Checar se existe algum erro

    # ####Categorias:
    # > 1 a 243

    [Teste: Checar se existe algum número < 1 ou > 243.
     Se encontrar, retornar erro indicando em qual linha.]
    :param passo: Número do passo atual para registro/log
    :param df:
    :return: sem retorno
    """
    log_tela.info("### PASSO " + str(passo) + " - ZONA_ORIG")

    # Verificando intervalo de valores - condições:
    # "ZONA_ORIG >= 1" E "ZONA_ORIG <= 243"
    verifica_range(df, 'ZONA_ORIG', 1, 243)
    log_output.info('\n\n=====')

    return df

```

```

def passo_subzona_orig(passo, df):
    """
    Checar se existe algum erro

    # ####Categorias:
    # > 1 a 633

    [Teste: Checar se existe algum número < 1 ou > 633.
     Se encontrar, retornar erro indicando em qual linha.]
    :param passo: Número do passo atual para registro/log
    :param df:
    :return: sem retorno
    """
    log_tela.info("### PASSO " + str(passo) + " - SUBZONA_ORIG")

    # Verificando intervalo de valores - condições:
    # "SUBZONA_ORIG >= 1" E "SUBZONA_ORIG <= 633"
    verifica_range(df, 'SUBZONA_ORIG', 1, 633)
    log_output.info('\n\n=====\\n')

    return df

def passo_mun_orig(passo, df):
    """
    Checar se existe algum erro

    # ####Categorias
    # > 1 a 27

    [Teste: Checar se existe algum número < 1 ou > 27.
     Se encontrar, retornar erro indicando em qual linha.]
    :param passo: Número do passo atual para registro/log
    :param df:
    :return: sem retorno
    """
    log_tela.info("### PASSO " + str(passo) + " - MUN_ORIG")

    # Verificando intervalo de valores - condições:
    # "MUN_ORIG >= 1" E "MUN_ORIG <= 27"
    verifica_range(df, 'MUN_ORIG', 1, 27)
    log_output.info('\n\n=====\\n')

    return df

def passo_zona_dest(passo, df):
    """
    Checar se existe algum erro

    # ####Categorias:
    # > 1 a 243

```



```

    [Teste: Checar se existe algum número < 1 ou > 243.
     Se encontrar, retornar erro indicando em qual linha.]
:param passo: Número do passo atual para registro/log
:param df:
:return: sem retorno
"""
log_tela.info("### PASSO " + str(passo) + " - ZONA_DEST")

# Verificando intervalo de valores - condições:
# "ZONA_DEST >= 1" E "ZONA_DEST <= 243"
verifica_range(df, 'ZONA_DEST', 1, 243)
log_output.info('\n\n=====\\n')

return df

def passo_subzona_dest(passo, df):
    """
    Checar se existe algum erro

    # ####Categorias:
    # > 1 a 633

    [Teste: Checar se existe algum número < 1 ou > 633.
     Se encontrar, retornar erro indicando em qual linha.]
:param passo: Número do passo atual para registro/log
:param df:
:return: sem retorno
"""
log_tela.info("### PASSO " + str(passo) + " - SUBZONA_DEST")

# Verificando intervalo de valores - condições:
# "SUBZONA_DEST >= 1" E "SUBZONA_DEST <= 633"
verifica_range(df, 'SUBZONA_DEST', 1, 633)
log_output.info('\n\n=====\\n')

return df

def passo_mun_dest(passo, df):
    """
    Checar se existe algum erro

    # ####Categorias
    # > 1 a 27

    [Teste: Checar se existe algum número < 1 ou > 27.
     Se encontrar, retornar erro indicando em qual linha.]
:param passo: Número do passo atual para registro/log
:param df:
:return: sem retorno
"""
log_tela.info("### PASSO " + str(passo) + " - MUN_DEST")

```

```

# Verificando intervalo de valores - condições:
# "MUN_DEST >= 1" E "MUN_DEST <= 27"
verifica_range(df, 'MUN_DEST', 1, 27)
log_output.info('\n\n=====')

return df

def passo_serv_pas_orig(passo, df):
    """
    Atribuir valor à variável dummy que identifica se o
    motivo origem é servir passageiro(a)

    #####
    #         ATENÇÃO         #
    # ESTE PASSO DEVE        #
    # SER EXECUTADO          #
    # ANTES DO PASSO         #
    #     MOTIVO_ORIG        #
    #####

    # Atribuir 1 quando "MOTIVO_ORIG" for igual a 9

    ####Categorias novas
    Valor/Descrição
    ----/----
    0/Não Respondido
    1/Sim
    2/Não

    :param passo: Número do passo atual para registro/log
    :param df:
    :return: sem retorno
    """
    log_tela.info("### PASSO " + str(passo) + " - SERV_PAS_ORIG")

    df['SERV_PAS_ORIG'] = 2

    df.loc[df['MOTIVO_ORIG']==9, 'SERV_PAS_ORIG'] = 1

    verifica_range(df, 'SERV_PAS_ORIG', 0, 2)
    log_output.info('\n\n=====')

    return df

def passo_serv_pas_dest(passo, df):
    """
    Atribuir valor à variável dummy que identifica se o
    motivo destino é servir passageiro(a)

    #####
    #         ATENÇÃO         #

```

```

# ESTE PASSO DEVE #
# SER EXECUTADO #
# ANTES DO PASSO #
# MOTIVO_DEST #
#####

# Atribuir 1 quando "MOTIVO_DEST" for igual a 9

####Categorias novas
Valor/Descrição
----/----
0/Não Respondido
1/Sim
2/Não

:param passo: Número do passo atual para registro/log
:param df:
:return: sem retorno
"""
log_tela.info("### PASSO " + str(passo) + " - SERV_PAS_DEST")

df['SERV_PAS_DEST'] = 2

df.loc[df['MOTIVO_DEST']==9, 'SERV_PAS_DEST'] = 1

verifica_range(df, 'SERV_PAS_DEST', 0, 2)
log_output.info('\n\n=====\\n')

return df

def passo_motivo_orig(passo, df):
    """
    Substituir todos valores **6** por **11**
    Substituir todos valores **7** por **6**
    Substituir todos valores **8** por **7**
    Substituir todos valores **10** por **8**
    Substituir todos valores **11** por **9**

    # ####Categorias anteriores
    # Valor/Descrição
    # ----/----
    # 1/Trabalho/Indústria
    # 2/Trabalho/Comércio
    # 3/Trabalho/Serviços
    # 4/Escola/Educação
    # 5/Compras
    # 6/Negócios
    # 7/Médico/Dentista/Saúde
    # 8/Recreação/Visitas
    # 9/Servir Passageiro
    # 10/Residência

    # ####Categorias novas

```

```

# Valor/Descrição
# ----/----
# 0/Não respondeu/não fez viagem
# 1/Trabalho/Indústria
# 2/Trabalho/Comércio
# 3/Trabalho/Serviços
# 4/Educação
# 5/Compras
# 6/Saúde
# 7/Lazer
# 8/Residência
# 9/Outros

[Teste: Checar se existe algum número < 0 ou > 9.
  Se encontrar, retornar erro indicando em qual linha.]
:param passo: Número do passo atual para registro/log
:param df:
:return: dataframe modificado
"""
log_tela.info("### PASSO " + str(passo) + " - MOTIVO_ORIG")

# Substituindo valor 6 por 11
df.loc[df['MOTIVO_ORIG']==6,'MOTIVO_ORIG'] = 11
# Substituindo valor 7 por 6
df.loc[df['MOTIVO_ORIG']==7,'MOTIVO_ORIG'] = 6
# Substituindo valor 8 por 7
df.loc[df['MOTIVO_ORIG']==8,'MOTIVO_ORIG'] = 7
# Substituindo valor 10 por 8
df.loc[df['MOTIVO_ORIG']==10,'MOTIVO_ORIG'] = 8
# Substituindo valor 11 por 9
df.loc[df['MOTIVO_ORIG']==11,'MOTIVO_ORIG'] = 9

# Verificando intervalo de valores - condições:
# "MOTIVO_ORIG >= 0" E "MOTIVO_ORIG <= 9"
verifica_range(df, 'MOTIVO_ORIG', 0, 9)
log_output.info('\n\n=====\\n')

return df

def passo_motivo_dest(passo, df):
    """
    Substituir todos valores **6** por **11**
    Substituir todos valores **7** por **6**
    Substituir todos valores **8** por **7**
    Substituir todos valores **10** por **8**
    Substituir todos valores **11** por **9**

    # ####Categorias anteriores
    # Valor/Descrição
    # ----/----
    # 1/Trabalho/Indústria
    # 2/Trabalho/Comércio
    # 3/Trabalho/Serviços

```

```

# 4/Escola/Educação
# 5/Compras
# 6/Negócios
# 7/Médico/Dentista/Saúde
# 8/Recreação/Visitas
# 9/Servir Passageiro
# 10/Residência

# ####Categorias novas
# Valor/Descrição
# ----/----
# 0/Não respondeu/não fez viagem
# 1/Trabalho/Indústria
# 2/Trabalho/Comércio
# 3/Trabalho/Serviços
# 4/Educação
# 5/Compras
# 6/Saúde
# 7/Lazer
# 8/Residência
# 9/Outros

[Teste: Checar se existe algum número < 0 ou > 9.
  Se encontrar, retornar erro indicando em qual linha.]
:param passo: Número do passo atual para registro/log
:param df:
:return: dataframe modificado
"""
log_tela.info("### PASSO " + str(passo) + " - MOTIVO_DEST")

# Substituindo valor 6 por 11
df.loc[df['MOTIVO_DEST']==6,'MOTIVO_DEST'] = 11
# Substituindo valor 7 por 6
df.loc[df['MOTIVO_DEST']==7,'MOTIVO_DEST'] = 6
# Substituindo valor 8 por 7
df.loc[df['MOTIVO_DEST']==8,'MOTIVO_DEST'] = 7
# Substituindo valor 10 por 8
df.loc[df['MOTIVO_DEST']==10,'MOTIVO_DEST'] = 8
# Substituindo valor 11 por 9
df.loc[df['MOTIVO_DEST']==11,'MOTIVO_DEST'] = 9

# Verificando intervalo de valores - condições:
# "MOTIVO_DEST >= 0" E "MOTIVO_DEST <= 9"
verifica_range(df, 'MOTIVO_DEST', 0, 9)
log_output.info('\n\n=====\\n')

return df

def passo_mod01(passo, df):
    """
    Não há o que fazer com os valores da coluna "MOD01"

    # ####Categorias anteriores

```

```

# Valor/Descrição
# ----/----
# 1/Ônibus trólebus
# 2/Ônibus Escolar / Empresa
# 3/Dirigindo Automóvel
# 4/Passageiro de Automóvel
# 5/Táxi
# 6/Lotação / Perua
# 7/Metrô
# 8/Trem
# 9/Motocicleta
# 10/Bicicleta
# 11/A Pé
# 12/Outros

# ####Categorias novas
# Valor/Descrição
# ----/----
# 0/Não respondeu/não fez viagem
# 1/Ônibus
# 2/Ônibus Escolar / Empresa
# 3/Dirigindo Automóvel
# 4/Passageiro de Automóvel
# 5/Táxi
# 6/Lotação / Perua / Van / Microônibus
# 7/Metrô
# 8/Trem
# 9/Moto
# 10/Bicicleta
# 11/A Pé
# 12/Outros

[Teste: Checar se existe algum número < 0 ou > 12.
  Se encontrar, retornar erro indicando em qual linha.]
:param passo: Número do passo atual para registro/log
:param df:
:return: sem retorno
"""

log_tela.info("### PASSO " + str(passo) + " - MOD01")

# Verificando intervalo de valores - condições:
# "MOD01 >= 0" E "MOD01 <= 12"
verifica_range(df, 'MOD01', 0, 12)
log_output.info('\n\n=====\\n')

return df

def passo_mod02(passo, df):
    """
    Não há o que fazer com os valores da coluna "MOD02"

    # ####Categorias anteriores
    # Valor/Descrição

```

```

# ----/----
# 1/Ônibus trólebus
# 2/Ônibus Escolar / Empresa
# 3/Dirigindo Automóvel
# 4/Passageiro de Automóvel
# 5/Táxi
# 6/Lotação / Perua
# 7/Metrô
# 8/Trem
# 9/Motocicleta
# 10/Bicicleta
# 11/A Pé
# 12/Outros

# ####Categorias novas
# Valor/Descrição
# ----/----
# 0/Não respondeu/não fez viagem
# 1/Ônibus
# 2/Ônibus Escolar / Empresa
# 3/Dirigindo Automóvel
# 4/Passageiro de Automóvel
# 5/Táxi
# 6/Lotação / Perua / Van / Microônibus
# 7/Metrô
# 8/Trem
# 9/Moto
# 10/Bicicleta
# 11/A Pé
# 12/Outros

[Teste: Checar se existe algum número < 0 ou > 12.
 Se encontrar, retornar erro indicando em qual linha.]
:param passo: Número do passo atual para registro/log
:param df:
:return: sem retorno
"""

log_tela.info("### PASSO " + str(passo) + " - MOD02")

# Verificando intervalo de valores - condições:
# "MOD02 >= 0" E "MOD02 <= 12"
verifica_range(df, 'MOD02', 0, 12)
log_output.info('\n\n=====\\n')

return df

def passo_mod03(passo, df):
    """
    Não há o que fazer com os valores da coluna "MOD03"

    # ####Categorias anteriores
    # Valor/Descrição
    # ----/----
    # 1/Ônibus trólebus

```

```

# 2/Ônibus Escolar / Empresa
# 3/Dirigindo Automóvel
# 4/Passageiro de Automóvel
# 5/Táxi
# 6/Lotação / Perua
# 7/Metrô
# 8/Trem
# 9/Motocicleta
# 10/Bicicleta
# 11/A Pé
# 12/Outros

# ####Categorias novas
# Valor/Descrição
# ----/----
# 0/Não respondeu/não fez viagem
# 1/Ônibus
# 2/Ônibus Escolar / Empresa
# 3/Dirigindo Automóvel
# 4/Passageiro de Automóvel
# 5/Táxi
# 6/Lotação / Perua / Van / Microônibus
# 7/Metrô
# 8/Trem
# 9/Moto
# 10/Bicicleta
# 11/A Pé
# 12/Outros

[Teste: Checar se existe algum número < 0 ou > 12.
  Se encontrar, retornar erro indicando em qual linha.]
:param passo: Número do passo atual para registro/log
:param df:
:return: sem retorno
"""
log_tela.info("### PASSO " + str(passo) + " - MOD03")

# Verificando intervalo de valores - condições:
# "MOD03 >= 0" E "MOD03 <= 12"
verifica_range(df, 'MOD03', 0, 12)
log_output.info('\n\n=====\\n')

return df

def passo_mod04(passo, df):
    """
    Nada há que se fazer em relação à coluna "MOD04" - não há dados de 1977,
    coluna permanecerá vazia
    :param passo: Número do passo atual para registro/log
    :param df:
    :return: sem retorno
    """
    log_tela.info("### PASSO " + str(passo) + " - MOD04")

```



```

log_output.info("MOD04: \n" +
                 str(df['MOD04'].value_counts()))
log_output.info('\n\n===== \n')

return df

def passo_mod04_prin(passo, df):
    """
    Não há o que fazer com os valores da coluna "MOD04_PRIN"

    # ####Categorias anteriores
    # Valor/Descrição
    # ----/----
    # 1/Ônibus trólebus
    # 2/Ônibus Escolar / Empresa
    # 3/Dirigindo Automóvel
    # 4/Passageiro de Automóvel
    # 5/Táxi
    # 6/Lotação / Perua
    # 7/Metrô
    # 8/Trem
    # 9/Motocicleta
    # 10/Bicicleta
    # 11/A Pé
    # 12/Outros

    # ####Categorias novas
    # Valor/Descrição
    # ----/----
    # 0/Não respondeu/não fez viagem
    # 1/Ônibus
    # 2/Ônibus Escolar / Empresa
    # 3/Dirigindo Automóvel
    # 4/Passageiro de Automóvel
    # 5/Táxi
    # 6/Lotação / Perua / Van / Microônibus
    # 7/Metrô
    # 8/Trem
    # 9/Moto
    # 10/Bicicleta
    # 11/A Pé
    # 12/Outros

    [Teste: Checar se existe algum número < 0 ou > 12.
     Se encontrar, retornar erro indicando em qual linha.]
    :param passo: Número do passo atual para registro/log
    :param df:
    :return: sem retorno
    """
    log_tela.info("### PASSO " + str(passo) + " - MOD04_PRIN")

    # Verificando intervalo de valores - condições:
    # "MOD04_PRIN >= 0" E "MOD04_PRIN <= 12"

```

```

verifica_range(df, 'MODO_PRIN', 0, 12)
log_output.info('\n\n=====\\n')

return df

def passo_tipo_est_auto(passo, df):
    """
    Substituir valores da coluna "TIPO_EST_AUTO"

    Substituir todos valores **1** por **5**
    Substituir todos valores **2** por **2**
    Substituir todos valores **3** por **2**
    Substituir todos valores **4** por **3**
    Substituir todos valores **5** por **5**
    Substituir todos valores **6** por **4**
    Substituir todos valores **7** por **1**

    # ####Categorias anteriores
    # Valor/Descrição
    # ----/----
    # 1/Zona Azul / Parquímetro
    # 2/Estacionamento Avulso
    # 3/Estacionamento Mensal
    # 4/Estacionamento Próprio
    # 5/Meio-Fio / Logradouro
    # 6/Estacionamento Patrocinado
    # 7/Não estacionou

    # ####Categorias novas
    # Valor/Descrição
    # ----/----
    # 0/Não Respondeu
    # 1/Não Estacionou
    # 2/Estacionamento Particular (Avulso / Mensal)
    # 3/Estacionamento Próprio
    # 4/Estacionamento Patrocinado
    # 5/Rua (meio fio / zona azul / zona marrom / parquímetro)

    [Teste: Checar se existe algum número < 0 ou > 5.
     Se encontrar, retornar erro indicando em qual linha.]
    :param passo: Número do passo atual para registro/log
    :param df:
    :return: retorna dataframe modificado
    """
    log_tela.info("### PASSO " + str(passo) + " - TIPO_EST_AUTO")

    # Substituindo valor 1 por 5
    df.loc[df['TIPO_EST_AUTO']==1,'TIPO_EST_AUTO'] = 5
    # Substituindo valor 3 por 2
    df.loc[df['TIPO_EST_AUTO']==3,'TIPO_EST_AUTO'] = 2
    # Substituindo valor 4 por 3
    df.loc[df['TIPO_EST_AUTO']==4,'TIPO_EST_AUTO'] = 3
    # Substituindo valor 6 por 4

```

```

df.loc[df['TIPO_EST_AUTO']==6,'TIPO_EST_AUTO'] = 4
# Substituindo valor 7 por 1
df.loc[df['TIPO_EST_AUTO']==7,'TIPO_EST_AUTO'] = 1

# Verificando intervalo de valores - condições:
# "TIPO_EST_AUTO >= 0" E "TIPO_EST_AUTO <= 5"
verifica_range(df, 'TIPO_EST_AUTO', 0, 5)
log_output.info('\n\n=====')

return df

def passo_valor_est_auto(passo, df, deflator):
    """
    Corrige o valor da coluna "VALOR_EST_AUTO" pelo
    deflator passado como parâmetro

    :param passo: Número do passo atual para registro/log
    :param df:
    :param deflator: Deflator a ser utilizado para correção
    :return: sem retorno
    """
    log_tela.info("### PASSO " + str(passo) + " - VALOR_EST_AUTO")

    df['VALOR_EST_AUTO'] = df['VALOR_EST_AUTO'] * deflator

    log_output.info('\n\n=====')

    return df

def passo_dist_viag(passo, df):
    """
    Calcula-se a distância euclidiana
    (a partir da CO_ORIG_X;CO_ORIG_Y e CO_DEST_X;CO_DEST_Y)
    :param passo: Número do passo atual para registro/log
    :param df:
    :return: Retorna dataframe com DIST_VIAG calculada e preenchida
    """
    log_tela.info("### PASSO " + str(passo) + " - DIST_VIAG")

    def calcula_dist_viag(row):
        """
        Calcula a distância euclidiana dadas as coordenadas (x,y) de origem
        e coordenadas (x,y) de destino da viagem.
        O argumento passado é a "linha".
        Uso:
            df['DIST_VIAG'] = df.apply(calcula_DIST_VIAG, axis=1)
        Retorna: DIST_VIAG da respectiva linha
        """
        co_orig_x = float(row['CO_ORIG_X'])
        co_orig_y = float(row['CO_ORIG_Y'])
        co_dest_x = float(row['CO_DEST_X'])
        co_dest_y = float(row['CO_DEST_Y'])

```

```

        x2 = math.pow((co_orig_x - co_dest_x), 2)
        y2 = math.pow((co_orig_y - co_dest_y), 2)
        return math.sqrt( x2 + y2 )

# Calculando "DIST_VIAG" (distância euclidiana)
# das coordenadas de origem (CO_ORIG_X;CO_ORIG_Y) e
# das coordenadas de destino (CO_DEST_X;CO_DEST_Y)
df['DIST_VIAG'] = df.apply(calcula_dist_viag, axis=1)

log_output.info('\n\n=====')

return df

def passo_tot_viag(passo, df):
    """
    #####
    #          ATENÇÃO          #
    # ESTA FUNÇÃO SÓ DEVE SER #
    # EXECUTADA APÓS A GERAÇÃO#
    # DE TODOS OS ID'S, POIS #
    # HÁ UMA DESCONFIANÇA    #
    # QUANTO À QUALIDADE     #
    # DESTE DADO              #
    #####
    Calcula e confere o campo TOT_VIAG,
    baseado no maior valor de NO_VIAG para cada pessoa
    :param passo: Número do passo atual para registro/log
    :param df:
    :return: retorna o dataframe com o "TOT_VIAG" corrigido
    """
    log_tela.info("### PASSO " + str(passo) + " - TOT_VIAG")
    log_tela.warning('\n\n#####\n' +
        'Este passo DEVE ser executado apenas após a \n' +
        'execução dos passos que geram os novos IDs\n' +
        '#####\n' )

    # 'Calculando' o máximo de viagens para cada indivíduo
    # Agrupa por ID_PESS e encontra o máximo para NO_VIAG.
    # O resultado é um objeto do tipo "Series" cujo index é
    # ID_PESS e a variável é NO_VIAG. Em seguida transforma
    # esse objeto num DataFrame e depois renomeia a coluna
    # NO_VIAG para MAX_VIAG.
    df_max = pd.DataFrame(
        df.groupby('ID_PESS')['NO_VIAG'].max()).rename(
            columns={'NO_VIAG': 'MAX_VIAG'})
    # Criando um novo dataframe auxiliar apenas com ID_PESS, apenas para ficar mais leve

    # O passo seguinte é atribuir o MAX_VIAG à coluna TOT_VIAG
    # do dataframe df, linkando por ID_PESS. Isso é feito usando o
    # método 'merge' da biblioteca pandas.
    df['TOT_VIAG'] = pd.merge(df, df_max, how='left', left_on='ID_PESS',
        right_index=True)['MAX_VIAG']

```

```

log_output.info('TOT_VIAG:\n\n' +
                '    Situação final dos dados: \n' +
                str(df['TOT_VIAG'].describe()) + '\n' +
                '    TOT_VIAG: Situação final dos dados: \n' +
                str(df['TOT_VIAG'].value_counts()))

# Agora uma função que irá verificar se para todo "ID_PESS"
# o "TOT_VIAG" é igual ao 'NO_VIAG' máximo.
def verifica_no_viag_tot_viag(row):
    if row['NO_VIAG'] != row['TOT_VIAG']:
        log_output.warning('TOT_VIAG: Erro encontrado na linha '
                           + str(row) + ':\n'
                           + ' => ' + row
                           )
df.loc[:, ['ID_PESS', 'NO_VIAG', 'TOT_VIAG']] \
    .groupby('ID_PESS') \
    .agg({'NO_VIAG': 'max', 'ID_PESS': 'max', 'TOT_VIAG': 'max'}) \
    .apply(verifica_no_viag_tot_viag, axis=1)
log_output.info('\n\n===== \n')

return df

```

### 3.7 Funções que geram os “NO”s e os “ID”s

Função/Variável	Status
gera_nos_e_ids	OK

```
In [ ]: log_tela.info('Definindo funções que geram os "NO"s e os "ID"s')
log_output.info('\n\n=====')

def gera_nos_e_ids(passo, df):
    """
    Esta função gera todos os IDs e todos os NOs das variáveis:
    DOM (domicílio), FAM (família), PESS (pessoa) e VIAG (viagem)
    A ordem de geração é:
    NO_DOM, ID_DOM, NO_FAM, ID_FAM, NO_PESS, ID_PESS, NO_VIAG, ID_VIAG
    Esta ordem é fixa pois cada uma dessas variáveis depende da geração
    da variável anterior.

    Os NOs são gerados como se fossem subíndices.
    No caso dos domicílios, eles são contabilizados dentro de cada zona.
    Assim, cada novo domicílio que aparece na listagem recebe um número
    que representa sua posição na sequência de domicílios dentro da zona
    na qual ele está contido.
    As famílias seguem o mesmo raciocínio, com relação ao domicílio, as
    pessoas com relação às famílias e as viagens com relação às pessoas.

    Deve-se apenas tomar cuidado com as viagens, pois existem pessoas que
    não realizaram viagens. Neste caso, estes registros devem contabilizar
    o NO_VIAG como zero, e não como 1. Para contemplar estes casos, vamos
    utilizar a variável F_VIAG, que representa quando há ou não viagem
    naquele registro (F_VIAG==1 tem viagem, F_VIAG==0 não tem viagem).
    Assim, o que precisamos fazer para calcular o NO_VIAG é agrupar os
    registros por pessoa (ID_PESS) e, dentro de cada agrupamento, somar
    o valor de F_VIAG registro a registro (linha por linha)
    cumulativamente. No final do processo, atribui-se zero a NO_VIAG
    quando F_VIAG for igual a zero.
    """

    def gera_id_dom(row):
        """
        Gera o ID_DOM baseado no 'ANO', na 'ZONA_DOM' e no 'NO_DOM'
        O argumento passado é a "linha".
        Uso:
        df['ID_DOM'] = df.apply(gera_id_dom, axis=1)
        Retorna: ID_DOM da respectiva linha
        """
        # Formatando o ano para strign
        ano = str(row['ANO'])

        # Formatando a zona para ficar como string com 3 caracteres
        zona = str('%03d' % row['ZONA_DOM'])
```

```

# Formatando no_dom para ficar como string com 4 caracteres
no_dom = str('%04d' % row['NO_DOM'])

# Retornando o número inteiro correspondente à string
# concatenada de ano + zona + no_dom
return ano + zona + no_dom

def gera_id_fam(row):
    """
    Gera o ID_FAM baseado no 'ID_DOM' e no 'NO_FAM'
    O argumento passado é a "linha".
    Uso:
        df['ID_FAM'] = df.apply(gera_ID_FAM, axis=1)
    Retorna: ID_FAM da respectiva linha
    """
    # Formatando id_dom como string
    id_dom = str(row['ID_DOM'])

    # Formatando no_fam como string para ficar com 2 caracteres
    no_fam = str('%02d' % row['NO_FAM'])

    # Retornando o número inteiro correspondente à string
    # concatenada de ID_DOM com NO_FAM
    return id_dom + no_fam

def gera_id_pess(row):
    """
    Gera o ID_PESS baseado no 'ID_FAM' e no 'NO_PESS'
    O argumento passado é a "linha".
    Uso:
        df['ID_PESS'] = df.apply(gera_ID_PESS, axis=1)
    Retorna: ID_PESS da respectiva linha
    """
    # Formatando id_fam como string
    id_fam = str(row['ID_FAM'])

    # Formatando no_pess como string para ficar com 2 caracteres
    no_pess = str('%02d' % row['NO_PESS'])

    # Retornando o número inteiro correspondente à string
    # concatenada de ID_FAM com NO_PESS
    return id_fam + no_pess

def gera_id_viag(row):
    """
    Gera o ID_VIAG baseado no 'ID_PESS' e no 'NO_VIAG'
    O argumento passado é a "linha".
    Uso:
        df['ID_VIAG'] = df.apply(gera_ID_VIAG, axis=1)
    Retorna ID_VIAG da respectiva linha
    """
    # Formatando id_pess como string
    id_pess = str(row['ID_PESS'])

```

```

    # Formatando no_viag como string para ficar com 2 caracteres
    no_viag = str('%02d' % row['NO_VIAG'])

    # Retornando o número inteiro correspondente à string
    # concatenada de ID_PESS com NO_VIAG
    return id_pess + no_viag

#gera NO_DOM
log_tela.info("Gerando NO_DOM")
df['NO_DOM'] = df.groupby('ZONA_DOM', sort=False)['F_DOM'].cumsum()
log_tela.info("NO_DOM gerado")
log_tela.info("Gerando ID_DOM")
#gera ID_DOM
df['ID_DOM'] = df.apply(gera_id_dom, axis=1)
log_tela.info("ID_DOM gerado")

#gera NO_FAM
log_tela.info("Gerando NO_FAM")
df['NO_FAM'] = df.groupby('ID_DOM', sort=False)['F_FAM'].cumsum()
log_tela.info("NO_FAM gerado")
log_tela.info("Gerando ID_FAM")
#gera ID_FAM
df['ID_FAM'] = df.apply(gera_id_fam, axis=1)
log_tela.info("ID_FAM gerado")

#gera NO_PESS
log_tela.info("Gerando NO_PESS")
df['NO_PESS'] = df.groupby('ID_FAM', sort=False)['F_PESS'].cumsum()
log_tela.info("NO_PESS gerado")
log_tela.info("Gerando ID_PESS")
#gera ID_PESS
df['ID_PESS'] = df.apply(gera_id_pess, axis=1)
log_tela.info("ID_PESS gerado")

#gera NO_VIAG
log_tela.info("Gerando NO_VIAG")
df['NO_VIAG'] = df.groupby('ID_PESS', sort=False)['F_VIAG'].cumsum()
# Verificando se F_VIAG == 0 e zerando NO_VIAG nesse caso
df.loc[df['F_VIAG'] == 0, 'NO_VIAG'] = 0
log_tela.info("NO_VIAG gerado")
log_tela.info("Gerando ID_VIAG")
#gera ID_VIAG
df['ID_VIAG'] = df.apply(gera_id_viag, axis=1)
log_tela.info("ID_VIAG gerado")

return df

```



### 3.8 Função relativa à entrevista

Função/Variável	Status
passo_cd_entre	OK

Obs: o passo\_cd\_entre deve ser executado após o passo\_tot\_viag.

```
In [ ]: log_tela.info('Definindo função relativa à entrevista')
        log_output.info('\n\n=====\\n')

def passo_cd_entre(passo, df):
    """
    -----
    Substituir valores da coluna "CD_ENTRE"
    Todas entrevistas são consideradas "completas", segundo informações do Metrô

    * sem viagem: se TOT_VIAG == 0
    * com viagem: se TOT_VIAG != 0

    # ####Categorias novas
    # | Valor | Descrição |
    # | ----- | ----- |
    # | 0 | Completa sem viagem |
    # | 1 | Completa com viagem |

    [Teste: Checar se existe algum número diferente de 0 ou 1.
     Se encontrar, retornar erro indicando em qual linha.]
    :param passo: Número do passo atual para registro/log
    :param df:
    :return:
    """
    log_tela.info("### PASSO " + str(passo) + " - CD_ENTRE")
    log_tela.warning('\n\n#####\\n' +
                    'Este passo DEVE ser executado apenas após a \\n' +
                    'execução do passo que geram o TOT_VIAG.\\n' +
                    '#####\\n' )

    # Definindo 'CD_ENTRE' baseado no valor de 'TOT_VIAG'
    df.loc[df['TOT_VIAG'] == 0, 'CD_ENTRE'] = 0
    df.loc[df['TOT_VIAG'] != 0, 'CD_ENTRE'] = 1

    verifica_dummy(df, 'CD_ENTRE')
    log_output.info('\n\n=====\\n')

    return df

In [ ]: def passo_correcoes(passo, df):
        log_tela.info("### PASSO " + str(passo) + " - Correções")

        log_tela.info("Corrigindo 'SERV_PAS' caso TOT_VIAG seja nulo.")
        df.loc[df['TOT_VIAG'] == 0, 'SERV_PAS_ORIG'] = 0
```

```

df.loc[df['TOT_VIAG'] == 0, 'SERV_PAS_DEST'] = 0

log_tela.info("Corrigindo motivo_origem, provável domicílio")
df.loc[(df['MOTIVO_ORIG']==0) &
      (df['MOTIVO_DEST']!=0) &
      (df['ZONA_ORIG']==df['ZONA_DOM'])&
      (df['SUBZONA_ORIG']==df['SUBZONA_DOM']), 'MOTIVO_ORIG'] == 8

log_tela.info("Corrigindo motivo_destino, provável domicílio")
df.loc[(df['MOTIVO_ORIG']!=0) &
      (df['MOTIVO_DEST']==0) &
      (df['ZONA_DEST']==df['ZONA_DOM'])&
      (df['SUBZONA_DEST']==df['SUBZONA_DOM']), 'MOTIVO_DEST'] == 8

return df

```

## 4 Defining the main function

This function will load all needed files, then call all other step functions defined above. On the end this main function will save the transformed data into an csv file.

```
In [ ]: def main():
    start_time = time.time()

    log_tela.info("Horário de início da execução: %s" %
                  time.strftime("%H:%M", time.localtime(start_time)))

    # ----

    log_tela.info('Lendo arquivos CSV')

    log_output.info('Lendo CSV da base original da OD.')
    od = pd.read_csv('bases/OD_1977.csv', sep=';', decimal=',')

    # Definindo variável de deflação de renda:
    deflator = 0.422345901

    # Filtrando dataframe para pegar apenas uma amostra
    # logging.info('\nFiltering the main dataframe to get just a sample')
    # od = od[:1000].copy()

    log_output.info('\n\n===== \n')
    log_tela.info('Pré-Processamento.')
    od = pre_processamento(od)

    log_tela.info('Imprimindo a lista de variáveis do dataframe da OD.')
    log_tela.debug(od.columns.tolist())

    #Contador de 'PASSO'
    passo = 1

    # ----
    # ##Passo: "ANO"
    od = passo_ano(passo, od)
    passo += 1

    # ----
    # ##Passo: "DIA_SEM"
    od = passo_dia_sem(passo, od)
    passo += 1

    # ----
    # ##Passo: UCODs
    od = passo_ucods(passo, od)
    passo += 1

    # ----
    # ##Passo: "ZONA_DOM"
    od = passo_zona_dom(passo, od)
    passo += 1
```

```

# -----
# ##Passo: "SUBZONA_DOM"
od = passo_subzona_dom(passo, od)
passo += 1

# -----
# ##Passo: "MUN_DOM"
od = passo_mun_dom(passo, od)
passo += 1

# -----
# ##Passo: "F_DOM"
od = passo_f_dom(passo, od)
passo += 1

# -----
# ##Passo: "FE_DOM"
od = passo_fe_dom(passo, od)
passo += 1

# -----
# ##Passo: "TIPO_DOM"
od = passo_tipo_dom(passo, od)
passo += 1

# -----
# ##Passo: "TOT_FAM"
od = passo_tot_fam(passo, od)
passo += 1

# -----
# ##Passo: "F_FAM"
od = passo_f_fam(passo, od)
passo += 1

# -----
# ##Passo: "FE_FAM"
od = passo_fe_fam(passo, od)
passo += 1

# -----
# ##Passo: "COND_MORA"
od = passo_cond_mora(passo, od)
passo += 1

# -----
# ##Passo: "QT_AUTO"
od = passo_qt_auto(passo, od)
passo += 1

# -----
# ##Passo: "QT_BICI"
od = passo_qt_bici(passo, od)
passo += 1

```

```

# -----
# ##Passo: "QT_MOTO"
od = passo_qt_moto(passo, od)
passo += 1

# -----
# ##PASSO: "REN_FAM"
od = passo_ren_fam(passo, od, deflator)
passo += 1

# -----
# ##Passo: "CD_RENFAM"
od = passo_cd_renfam(passo, od)
passo += 1

# -----
# ##Passo: "F_PESS"
od = passo_f_pess(passo, od)
passo += 1

# -----
# ##Passo: "FE_PESS"
od = passo_fe_pess(passo, od)
passo += 1

# -----
# ##Passo: "SIT_FAM"
od = passo_sit_fam(passo, od)
passo += 1

# -----
# ##Passo: "IDADE"
od = passo_idade(passo, od)
passo += 1

# -----
# ##Passo: "SEXO"
od = passo_sexo(passo, od)
passo += 1

# -----
# ##Passo: "GRAU_INSTR"
od = passo_grau_instr(passo, od)
passo += 1

# -----
# ##Passo: "OCUP"
od = passo_ocup(passo, od)
passo += 1

# -----
# ##Passo: "SETOR_ATIV"
od = passo_setor_ativ(passo, od)

```

```

passo += 1

# -----
# ##Passo: "REN_IND"
od = passo_ren_ind(passo, od, deflator)
passo += 1

# -----
# ##Passo: "CD_RENIND"
od = passo_cd_renind(passo, od)
passo += 1

## O Passo estuda encontra-se mais abaixo, após MUN_ESC

# -----
# ##Passo: "F_VIAG"
od = passo_f_viag(passo, od)
passo += 1

# -----
# ##Passo: "FE_VIAG"
od = passo_fe_viag(passo, od)
passo += 1

# -----
# ##Passo: "ZONA_ESC"
od = passo_zona_esc(passo, od)
passo += 1

# -----
# ##Passo: "SUBZONA_ESC"
od = passo_subzona_esc(passo, od)
passo += 1

# -----
# ##Passo: "MUN_ESC"
od = passo_mun_esc(passo, od)
passo += 1

# -----
# ##Passo: "ESTUDA"
# Este passo deve vir após os passos de localização da escola
od = passo_estuda(passo, od)
passo += 1

# -----
# ##Passo: "ZONA_TRAB1"
od = passo_zona_trab1(passo, od)
passo += 1

# -----
# ##Passo: "SUBZONA_TRAB1"
od = passo_subzona_trab1(passo, od)
passo += 1

```

```

# -----
# ##Passo: "MUN_TRAB1"
od = passo_mun_trab1(passo, od)
passo += 1

# -----
# ##Passo: "ZONA_TRAB2"
od = passo_zona_trab2(passo, od)
passo += 1

# -----
# ##Passo: "SUBZONA_TRAB2"
od = passo_subzona_trab2(passo, od)
passo += 1

# -----
# ##Passo: "MUN_TRAB2"
od = passo_mun_trab2(passo, od)
passo += 1

# -----
# ##Passo: "ZONA_ORIG"
od = passo_zona_orig(passo, od)
passo += 1

# -----
# ##Passo: "SUBZONA_ORIG"
od = passo_subzona_orig(passo, od)
passo += 1

# -----
# ##Passo: "MUN_ORIG"
od = passo_mun_orig(passo, od)
passo += 1

# -----
# ##Passo: "ZONA_DEST"
od = passo_zona_dest(passo, od)
passo += 1

# -----
# ##Passo: "SUBZONA_DEST"
od = passo_subzona_dest(passo, od)
passo += 1

# -----
# ##Passo: "MUN_DEST"
od = passo_mun_dest(passo, od)
passo += 1

# -----
# ##Passo: "SERV_PAS_ORIG"
od = passo_serv_pas_orig(passo, od)

```

```

passo += 1

# -----
# ##Passo: "SERV_PAS_DEST"
od = passo_serv_pas_dest(passo, od)
passo += 1

# -----
# ##Passo: "MOTIVO_ORIG"
od = passo_motivo_orig(passo, od)
passo += 1

# -----
# ##Passo: "MOTIVO_DEST"
od = passo_motivo_dest(passo, od)
passo += 1

# -----
# ##Passo: "MOD01"
od = passo_mod01(passo, od)
passo += 1

# -----
# ##Passo: "MOD02"
od = passo_mod02(passo, od)
passo += 1

# -----
# ##Passo: "MOD03"
od = passo_mod03(passo, od)
passo += 1

# -----
# ##Passo: "MOD04"
od = passo_mod04(passo, od)
passo += 1

# -----
# ##Passo: "MOD0_PRIN"
od = passo_mod0_prin(passo, od)
passo += 1

# -----
# ##"TIPO_VIAG"; "H_SAIDA"; "MIN_SAIDA"; "ANDA_ORIG"; "H_CHEG"; "MIN_CHEG";
# "ANDA_DEST" e "DURACAO"
# Nada há que se fazer em relação aos dados das colunas acima mencionadas

# -----
# ##Passo: "TIPO_EST_AUTO"
od = passo_tipo_est_auto(passo, od)
passo += 1

# -----
# ##Passo: "VALOR_EST_AUTO"

```



```

od = passo_valor_est_auto(passo, od, deflator)
passo += 1

# -----
# ##Passo: Coordenadas
od = coordenadas(passo, od)
passo += 1

# -----
# ##Passo: "DIST_VIAG"
od = passo_dist_viag(passo, od)
passo += 1

## O passo TOT_VIAG apenas é chamado após a geração dos IDs.

# ----
# ##Passo: Gerando NO's e ID's
od = gera_nos_e_ids(passo, od)
passo += 1

# ----
# ##Passo: "TOT_VIAG"
od = passo_tot_viag(passo, od)
passo += 1

# -----
# ##Passo: "CD_ENTRE"
od = passo_cd_entre(passo, od)
passo += 1

# -----
# ##Passo: Correções
od = passo_correcoes(passo, od)
passo += 1

log_tela.info('Salvando dataframe como arquivo CSV')
# -----
# ## Salvando o dataframe num arquivo local
od.to_csv('outputs/1977_od.csv', sep=';', decimal=',', index=False)

log_tela.info("Output gerado. Arquivo: outputs/novo_od1977.csv")

log_tela.info("Tempo total de execução: %s segundos" %
              (time.time() - start_time))

log_tela.info("Horário de finalização: %s" %
              (time.strftime("%H:%M", time.localtime(time.time()))))

log_tela.info("Terminou o main")

```

## 5 RUN the main() function....

```
In [ ]: if __name__ == "__main__":  
        main()
```