rotina_1977_ok

November 3, 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 1987

3.1 Funções gerais assessórias

Função	Status
consulta_refext	OK
verifica_dummy	OK
verifica_range	OK
corrige_renda	OK
coord	OK

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


        def consulta_refext(row, ext_data_frame, col_ref, col_filt, col_search):
            """
            Traz valor de referência externa (em arquivo csv) baseado em valor de
            referência do arquivo de origem.
            :param row: registro ("linha") que será trabalhado nesta iteração.
            :param ext_data_frame: objeto do tipo dataframe que contém os dados
                                   que devem ser buscados. P.ex: dataframe com a
                                   relação ZONA e Coordenadas do centróide da ZONA.
            :param col_ref: nome da variável (coluna) do dataframe ext_data_frame que
                                servirá de refência para a localização. P.ex: ZONA
            :param col_filt: nome da variável (coluna) da linha (row) que contém o
                                valor a ser buscado e comparado com o col_ref no
                                ext_data_frame. P.ex: ZONA
            :param col_search: nome da variável (coluna) no dataframe ext_data_frame
                                que contém o valor a ser retornado. P.ex: CO_X
            return: Valor procurado
            Uso:
                od1977_ex['coluna que receberá o novo dado'] = od1977_ex.apply(
                                                               lambda row:
                                                                   consulta_refext(
                                                                       row,
                                                                       data_frame_externo,
                                                                       'reference column',
                                                                       'filter column',
                                                                       'searched column'),
                                                                       axis=1)
            """
            # 0 em geral é o caso que a informação não existe e/ou não foi respondida
            # como, por exemplo, UCOD_ESC quando a viagem não é relativa a Escola
            if row[col_filt] == 0 or row[col_filt] == 999:
                return row[col_filt]

            res = ext_data_frame[ext_data_frame[col_ref] == row[col_filt]][col_search]
            # Verificando se algum valor foi encontrado
            if not res.empty:
```

```
        return int(res)
    else:
        log_tela.warning('\n    Erro encontrado ao tentar buscar ' +
                         col_search + ' referente ao valor ' +
                            str(row[col_filt]) + ' da coluna ' + col_filt)
        # log_output.warning('Linha com erro: \n      ' + str(row))
        return None


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 corrige_renda(passo, df, variavel, deflator):
    """
    Passo que corrige um valor monetário no tempo de acordo com um deflator.
    Serve para fazer ajuste ao longo do tempo de forma a equalizar a
    renda (ou algum custo) de diversos anos para um único período a ser definido,
    quando do cálculo do deflator.
    :param passo: número do passo atual para registro/log
    :param df: DataFrame da OD a ser modificada
    :param variavel: Valor a ser modificado (nome da variável)
                        REN_FAM ou REN_IND ou VALOR_EST_AUTO
    :param deflator: Valor do deflator que deve ser utilizado para correção
    :return: dataframe modificado com valores corrigidos
    """
    log_tela.info('### PASSO ' + str(passo) + ' - Corrige renda' +
                            variavel + '\n Deflator utilizado: ' +
                            str(deflator))
```

```
    df[variavel] = df.apply(lambda row: row[variavel] * deflator, axis=1)
    log_output.info('\n\n================================================\n')

    return df


def coord(passo, df, coords, tipo, eixo):
    """
    Na linha i ler o valor da coluna "ZONA_TIPO" (ex.: ZONA_DOM),
        buscar o valor encontrado no arquivo , na coluna "ZONA_REF".
        Está sendo utilizada a ZONA pois para 1977 não há a
        informação das subzonas e suas coordenadas.
    Retornar o valor da coluna "CO_EIXO" (ex.: CO_X)
        da linha encontrada no dataframe coord e
        salvar o valor na coluna "CO_TIPO_EIXO" (ex.: CO_DOM_X)
        na linha atual do dataframe df

    :param passo: número do passo para o log
    :param df: Dafaframe a ser modificado (ex.: od1977)
    :param coords: dataframe com as coordenadas a serem consultadas
                                                (ex.: COORD-SUBZONA-1977.csv)
    :param tipo: Tipo da COORD sendo avaliada
                                (DOM ou ESC ou TRAB1 ou TRAB2 ou ORIG ou DEST)
    :param eixo: Eixo a ser consultado (X ou Y)
    :return: retorna o dataframe modificado
    """
    log_tela.info("### PASSO " + str(passo) + " - CO_" + tipo + "_" + eixo)

    log_tela.info("CO_" + tipo +
                  ": Consultando referência externa com os parâmetros: " +
                  "ZONA_REF, " +
                  "ZONA_" + tipo +
                  ', CO_' + eixo)

    df["CO_" + tipo + "_" + eixo] = df.apply(
                                        lambda row:
                                            consulta_refext(
                                                row,
                                                coords,
                                                'ZONA_REF',  # ERA SUBZONA
                                                'ZONA_' + tipo,  # ERA SUBZONA
                                                'CO_' + eixo),
                                            axis=1)
    log_output.info('\n\n================================================\n')

    return df
```

3.2 Funções Gerais

Função/Variável	Status
passo_ano	OK
passo_dia_sem	OK
passo_ucod	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_ucod(passo, df, ucod, tipo_ucod):
    """
    Na coluna "UCOD_DOM", linha i,
        ler o valor da linha i da coluna "ZONA_XXX", daí,
        buscar o mesmo valor na coluna "ZONA_REF" do arquivo UCOD-1977.csv.
    Ao encotrar, retornar o valor da mesma linha, só que da coluna "UCOD_XXX"
    [Teste: no banco completo, checar se o min == 1 e o max == 67]

    XXX = DOM ou ESC ou TRAB1 ou TRAB2 ou ORIG ou DEST

    :param passo: número do passo para o log
    :param df: Dafaframe a ser modificado
    :param ucod: dataframe com os códigos a serem consultados
    :param tipo_ucod: Tipo da UCOD sendo avaliada
            (DOM ou ESC ou TRAB1 ou TRAB2 ou ORIG ou DEST)
    :return: retorna o dataframe (df) modificado
    """
    log_tela.info("### PASSO " + str(passo) + " - UCOD_" + tipo_ucod)

    log_tela.info("UCOD_" + tipo_ucod +
                  ": Consultando referência externa com os parâmetros: " +
                  "ZONA_REF, " +
                  "ZONA_" + tipo_ucod +
                  ', UCOD_BUSCADA')

    df['UCOD_' + tipo_ucod] = df.apply(
                                lambda row:
                                    consulta_refext(
                                        row,
                                        ucod,
                                        'ZONA_REF',
                                        'ZONA_'+ tipo_ucod,
                                        'UCOD_BUSCADA'),
                                    axis=1)

    #  Verificando intervalo de valores - condições:
        # "UCOD_XXX >= 1" E "UCOD_XXX <= 67"
    verifica_range(df, 'UCOD_' + tipo_ucod, 1, 67)
    log_output.info('\n\n================================================\n')

    return df
```

3.3 Funções do Domicílio

Função/Variável	Status
passo_zona_dom	OK
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================================================\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================================================\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================================================\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================================================\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 = corrige_renda(passo, df, 'REN_FAM', deflator)

    return df


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

    * Excluir único registro na categoria 3
    * Substituir todos valores **1** 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 por 2
df.loc[df['CD_RENFAM']==1,'CD_RENFAM'] = 2

def melhora_cd_renfam(row):
    # Quando categoria original for 0 (respondeu) checar no campo REN_FAM
    #   se o valor é nulo.
    # Se for nulo, manter na categoria 0 (Renda Fam. Declarada como Zero),
    #   senão,
    #   mover para a categoria 1 (Renda Fam. Declarada e Maior que Zero).

    if row['CD_RENFAM'] == 0 and\
                                row['REN_FAM'] != 0 and\
                                row['REN_FAM'] is not None:
        df.loc[row.name, 'CD_RENFAM'] = 1

# Dividindo a categoria '0', "Respondeu", em:
#   - 0 "Renda Familiar Declarada como Zero" e
#   - 1 "Renda Familiar Declarada e Maior que Zero"
df.apply(melhora_cd_renfam, axis=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**

    # ####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
    # ----|----
    # 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 < 1 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

    # Verificando intervalo de valores - condições:
        # "SIT_FAM >= 1" E "SIT_FAM <= 6"
    verifica_range(df, 'SIT_FAM', 1, 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================================================\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
    # ----|----
    # 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', 1, 7)
    log_output.info('\n\n================================================\n')

    return df


def passo_setor_ativ(passo, df, df_setor):
    """
    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|Serviços
    # 8|Serviços Autônomos
    # 9|Outros
    # 10|Não se aplica

    [Teste: Checar se existe algum número < 1 ou > 10.
        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")

    df['SETOR_ATIV'] = df.apply(
                        lambda row:
                            consulta_refext(
                                row,
                                df_setor,
                                'COD',
                                'SETOR_ATIV',
                                'COD_UNIF'),
                            axis=1)

    #  Verificando intervalo de valores - condições:
        # "SETOR_ATIV >= 0" E "SETOR_ATIV <= 10"
    verifica_range(df, 'SETOR_ATIV', 0, 10)
    log_output.info('\n\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 = corrige_renda(passo, df, 'REN_IND', deflator)

    return df


def passo_cd_renind(passo, df):
    """
    Checar se existe algum erro na coluna "CD_RENIND"

    #### Categorias anteriores
    Valor|Descrição
    ----|----
    1|Não tem renda
    2|Não Declarou
    3|Declarou

    # ####Categorias novas
    # Valor|Descrição
    # ----|----
    # 0|Não declarou
    # 1|Tem renda
```

```
    # 2|Não tem renda

    [Teste: Checar se existe algum número < 1 ou > 3.
        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")

    # Converte o "Não Respondeu"/"Não Declarou" (3) em 0 (zero)
    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==========================================\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_modo1	OK
passo_modo2	OK
passo_modo3	OK
passo_modo4	OK
passo_modo_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|Demais viagens
    # 1|Primeira Viagem 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: sem retorno
    """
    log_tela.info("### PASSO " + str(passo) + " - F_VIAG")

    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================================================\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================================================\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================================================\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================================================\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

    :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'] = 0

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

    verifica_dummy(df, 'SERV_PAS_ORIG')
    log_output.info('\n\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

    :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'] = 0

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

    verifica_dummy(df, 'SERV_PAS_DEST')
    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_modo1(passo, df):
    """
    Não há o que fazer com os valores da coluna "MODO1"

    # ####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) + " - MODO1")

    # Verificando intervalo de valores - condições:
        # "MODO1 >= 0" E "MODO1 <= 12"
    verifica_range(df, 'MODO1', 0, 12)
    log_output.info('\n\n================================================\n')

    return df


def passo_modo2(passo, df):
    """
    Não há o que fazer com os valores da coluna "MODO2"

    # ####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) + " - MODO2")

    # Verificando intervalo de valores - condições:
        # "MODO2 >= 0" E "MODO2 <= 12"
    verifica_range(df, 'MODO2', 0, 12)
    log_output.info('\n\n================================================\n')

    return df

def passo_modo3(passo, df):
    """
    Não há o que fazer com os valores da coluna "MODO3"

    # ####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) + " - MODO3")

    # Verificando intervalo de valores - condições:
        # "MODO3 >= 0" E "MODO3 <= 12"
    verifica_range(df, 'MODO3', 0, 12)
    log_output.info('\n\n================================================\n')

    return df

def passo_modo4(passo, df):
    """
    Nada há que se fazer em relação à coluna "MODO4" - 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) + " - MODO4")

    log_output.info("MODO4: \n" +
                            str(df['MODO4'].value_counts()))
    log_output.info('\n\n================================================\n')

    return df


def passo_modo_prin(passo, df):
    """
    Não há o que fazer com os valores da coluna "MODO_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) + " - MODO_PRIN")

        # Verificando intervalo de valores - condições:
            # "MODO_PRIN >= 0" E "MODO_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 / Parqímetro
    # 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================================================\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 = corrige_renda(passo, df, 'VALOR_EST_AUTO', deflator)

    log_output.info('\n\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:
            od1977['DIST_VIAG'] = od1977.apply(calcula_DIST_VIAG, axis=1)
        Retorna: DIST_VIAG da respetiva 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================================================\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")

def atrib_tot_viag(row):
    df.loc[df['ID_PESS'] == row['ID_PESS'],'TOT_VIAG'] = row['NO_VIAG']

df.loc[:, ['ID_PESS', 'NO_VIAG']]\
    .groupby(['ID_PESS'], sort=False)\
    .agg({'NO_VIAG': max, 'ID_PESS': max})\
    .apply(atrib_tot_viag, axis=1)

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
passo_no_dom	OK
passo_id_dom	OK
passo_no_fam	OK
passo_id_fam	OK
passo_no_pess	OK
passo_id_pess	OK
passo_no_viag	Verificar
passo_id_viag	OK

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


        def passo_no_dom(passo, df):
            """
            Gerando "NO_DOM" como um subindíce de cada "ZONA_DOM"
            Para cada "ZONA_DOM" o "NO_DOM" será atualizado sempre que
                "F_DOM" for igual a 1
            Do contrário, se "F_DOM" for igual a zero, então "NO_DOM" será igual ao
                "NO_DOM" da linha anterior.
            :param passo: Número do passo atual para registro/log
            :param df:
            :return: retorna dataframe modificado com novo NO_DOM
            """
            log_tela.info("### PASSO " + str(passo) + " - NO_DOM")

            log_output.info("NO_DOM: INICIANDO GERAÇÃO DO NO_DOM")

            def gera_no_dom(row):
                # Use esta função com:
                #     dataframe.apply(gera_NO_DOM, axis=1)
                #
                # Retorna 0 se "NO_DOM" foi aplicado e 1 se houve algum erro.
                # row.name é o índice da linha sendo avaliada.

                # row.name == 0 é sinônimo de que estamos no primeiro registro da base
                if row.name == 0:
                    # Se for a primeira linha do dataframe, então NO_DOM = 1.
                    df.loc[row.name, 'NO_DOM'] = 1
                    return 0

                zona_anterior = df.loc[row.name - 1, 'ZONA_DOM']
                zona_atual = df.loc[row.name, 'ZONA_DOM']

                if zona_atual != zona_anterior:
                    # Se for a primeira linha ZONA_DOM, então NO_DOM = 1.
```

```
            # Esta lógica considera que o dataframe está ordenado por ZONA_DOM.
            # Este é um requisito forte!.
            df.loc[row.name, 'NO_DOM'] = 1
        elif row['F_DOM'] == 1:
            df.loc[row.name, 'NO_DOM'] = df.loc[row.name - 1, 'NO_DOM'] + 1
        elif row['F_DOM'] == 0:
            df.loc[row.name, 'NO_DOM'] = df.loc[row.name - 1, 'NO_DOM']
        else:
            log_tela.warning("NO_DOM: Erro na composição da linha "
                                    + str(row.name) + ':\n'
                                    + ' => row'
            )
            return 1
        return 0

    # A função gera_NO_DOM é chamada e devido ao fato dela retornar 1
        # se mal sucedida, isso possibilita somar e verificar a quantidade
        # de erros que aconteceram
    erros = df.apply(gera_no_dom, axis=1).sum()
    if erros > 0:
        log_tela.warning(
            "NO_DOM: Número de composições em que ocorreu algum erro: " +
            str(erros))
    else:
        log_tela.info("NO_DOM: Nenhum erro encontrado")
    log_output.info('\n\n================================================\n')

    return df


def passo_id_dom(passo, df):
    """
    Construir o "NO_DOM" e o "ID_DOM"

    Na coluna "ID_DOM", linha i, ler o valor da linha i da coluna "ZONA_DOM",
        e concatenar esse valor (com 3 dígitos) com o número do domicílio,
        que é o valor da linha i da coluna "NO_DOM" (com 4 dígitos).
    Resultado será um ID_DOM, que pode se repetir nas linhas, de 7 dígitos.
    Isso deve ser concatenado com o "Ano".
    Resultado = 8 dígitos
    :param passo: Número do passo atual para registro/log
    :param df:
    :return: retorna dataframe modificado com o ID_DOM gerado
    """
    log_tela.info("### PASSO " + str(passo) + " - ID_DOM")
    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:
            od1977['ID_DOM'] = od1977.apply(gera_ID_DOM, axis=1)
        Retorna: ID_DOM da respectiva linha
        """
        ano = int(row['ANO'])
```

```
        zona = int(row['ZONA_DOM'])
        no_dom = int(row['NO_DOM'])
        return int(str(ano)+str('%03d' % zona) + str('%04d' % no_dom))

    # Gerando "ID_DOM" como a concatenação das variáveis:
        # "ANO", "ZONA_DOM" e "NO_DOM"
    df['ID_DOM'] = df.apply(gera_id_dom, axis=1)
    log_output.info('\n\n===============================================\n')

    return df


def passo_no_fam(passo, df):
    """
    Gerando "NO_FAM" como subíndice do "ID_DOM"
    Para cada "ID_DOM", o "NO_FAM" será incrementado sempre que
        "F_FAM" for igual a 1
    Do contrário, caso "F_FAM" seja igual a 0, então "NO_FAM" receberá
        o valor de "NO_FAM" da linha anterior.
    :param passo: Número do passo atual para registro/log
    :param df:
    :return: dataframe modificado com o NO_FAM gerado
    """
    log_tela.info("### PASSO " + str(passo) + " - NO_FAM")

    log_output.info("NO_FAM: INICIANDO GERAÇÃO DO NO_FAM")

    def gera_no_fam(row):
        # Utilize esta função como:
        #     dataframe.apply(gera_NO_FAM, axis=1)
        #
        # Retorno 0 se o "NO_FAM" foi aplicado e 1 se ocorreu algum erro.
        # row.name é o índice da linha sendo analisada.

        # row.name == 0 é sinônimo de que estamos no primeiro registro da base
        if row.name == 0:
            # Se for a primeira linha do dataframe, então NO_FAM = 1.
            df.loc[row.name, 'NO_FAM'] = 1
            return 0

        domicilio_anterior = df.loc[row.name - 1, 'ID_DOM']
        domicilio_atual = df.loc[row.name, 'ID_DOM']

        if domicilio_atual != domicilio_anterior:
            # Se for a primeira linha do ID_DOM, então NO_FAM = 1.
            # Esta lógica considera que o dataframe está ordenado por ID_DOM.
            # Este é um requisito forte para execução.
            df.loc[row.name, 'NO_FAM'] = 1

        elif row['F_FAM'] == 1:
            df.loc[row.name, 'NO_FAM'] = df.loc[row.name - 1, 'NO_FAM'] + 1
        elif row['F_FAM'] == 0:
            df.loc[row.name, 'NO_FAM'] = df.loc[row.name - 1, 'NO_FAM']
        else:
```

```
            log_tela.warning("NO_FAM: Erro na composição da linha "
                                        + str(row.name) + ':\n'
                                        + ' => row'
            )
            return 1
        return 0

    # A função gera_NO_FAM é chamada e devido ao fato de retornar 1 se
        # ocorrer erro, é possível somar e verificar quantos erros ocorreram.
    erros = df.apply(gera_no_fam, axis=1).sum()
    if erros > 0:
        log_tela.warning(
            "NO_FAM: Número de composições em que ocorreu algum erro: " +
            str(erros))
    else:
        log_tela.info("NO_FAM: Nenhum erro encontrado")
    log_output.info('\n\n================================================\n')

    return df


def passo_id_fam(passo, df):
    """
    # Construir "ID_FAM"
    # Na coluna "ID_FAM", linha i, ler o valor da linha i da coluna "ID_DOM",
    # e concatenar esse valor (com 8 dígitos) com o número da família,
    # que é o valor da linha i da coluna "NO_FAM" (com 2 dígitos).
    #
    # Resultado será um ID_FAM, que pode se repetir nas linhas, de 10 dígitos.
    :param passo: Número do passo atual para registro/log
    :param df:
    :return: retorna o dataframe com o ID_FAM atualizado
    """
    log_tela.info("### PASSO " + str(passo) + " - ID_FAM")

    def gera_id_fam(row):
        """
        Gera o ID_FAM baseado no 'ID_DOM' e no 'NO_FAM'
        O argumento passado é a "linha".
            Uso:
            od1977['ID_FAM'] = od1977.apply(gera_ID_FAM, axis=1)
        Retorna: ID_FAM da respectiva linha
        """
        id_dom = int(row['ID_DOM'])
        no_fam = int(row['NO_FAM'])
        return int(str(id_dom) + str('%02d' % no_fam))

    # Gerando "ID_FAM" a partir da concatenação das variáveis:
    df['ID_FAM'] = df.apply(gera_id_fam, axis=1)
    log_output.info('\n\n================================================\n')

    return df
```

```
def passo_no_pess(passo, df):
    """
    Gerando "NO_PESS" como subíndice do "ID_FAM"
    Para cada "ID_FAM" o "NO_PESS" será incrementado sempre que
        "F_PESS" for igual a 1
    Do contrário, caso "F_PESS" seja igual a 0, então "NO_PESS" receberá
        o valor de "NO_PESS" da linha anterior.
    :param passo: Número do passo atual para registro/log
    :param df:
    :return: retorna o dataframe modificado com o novo NO_PESS
    """
    log_tela.info("### PASSO " + str(passo) + " - NO_PESS")

    log_output.info("NO_PESS: INICIANDO GERAÇÃO DO NO_PESS")

    def gera_no_pess(row):
        # Utilize essa função como:
        #     dataframe.apply(gera_NO_PESS, axis=1)
        #
        # Retorna 0 se "NO_PESS" foi gerado e 1 se ocorreu algum erro
        # row.name é o índice da linha específica.

        if row.name == 0:
            df.loc[row.name, 'NO_PESS'] = 1
            return 0

        familia_anterior = df.loc[row.name - 1, 'ID_FAM']
        familia_atual = df.loc[row.name, 'ID_FAM']

        if familia_atual != familia_anterior:
            # Se for a primeira linha de ID_FAM, então NO_PESS = 1.
            # Isto leva em consideração que o dataframe é ordenado por ID_FAM.
            # Este é um requerimento bem forte.

            df.loc[row.name, 'NO_PESS'] = 1
        elif row['F_PESS'] == 1:
            df.loc[row.name, 'NO_PESS'] = df.loc[row.name - 1, 'NO_PESS'] + 1
        elif row['F_PESS'] == 0:
            df.loc[row.name, 'NO_PESS'] = df.loc[row.name - 1, 'NO_PESS']
        else:
            log_tela.warning("NO_PESS: Erro na composição da linha "
                                    + str(row.name) + ':\n'
                                    + ' => ' + row
            )
            return 1
        return 0

    # A função gera_NO_PESS é chamada e devido ao fato de retornar 1 se
        # ocorrer erro, é possível somar e verificar a quantidade de erros.
    erros = df.apply(gera_no_pess, axis=1).sum()
    if erros > 0:
        log_tela.warning(
            "NO_PESS: Número de composições em que ocorreu algum erro: " +
            str(erros))
```

```
    else:
        log_tela.info("NO_PESS: Nenhum erro encontrado")
    log_output.info('\n\n================================================\n')

    return df


def passo_id_pess(passo, df):
    """
    Construir "ID_PESS" e "NO_PESS"
    Na coluna "ID_PESS", linha i, ler o valor da linha i da coluna "ID_FAM", e
        concatenar esse valor (10 dígitos) com o número da pessoa,
        que é o valor da linha i da coluna "NO_PESS" (com 2 dígitos).

    Resultado será um ID_PESS, que pode se repetir nas linhas, de 12 dígitos.
    :param passo: Número do passo atual para registro/log
    :param df:
    :return: retorna o dataframe com o ID_PESS atualizado
    """
    log_tela.info("### PASSO " + str(passo) + " - ID_PESS")

    def gera_id_pess(row):
        """
        Gera o ID_PESS baseado no 'ID_FAM' e no 'NO_PESS'
        O argumento passado é a "linha".
            Uso:
            od1977['ID_PESS'] = od1977.apply(gera_ID_PESS, axis=1)
        Retorna: ID_PESS da respectiva linha
        """
        id_fam = int(row['ID_FAM'])
        no_pess = int(row['NO_PESS'])
        return int(str(id_fam) + str('%02d' % no_pess))

    # Gerando "ID_PESS" from the concatenation of "ID_FAM" and "NO_PESS"
    df['ID_PESS'] = df.apply(gera_id_pess, axis=1)
    log_output.info('\n\n================================================\n')

    return df


def passo_no_viag(passo, df):
    """
    Gerando "NO_VIAG" como subíndice do "ID_PESS"

    # NO_VIAG = NONE (não respondeu)
    SE "FE_VIAG" for igual a zero, então a pessoa não respondeu
        sobre suas viagens. Assim, atribui-se NONE (NA) a "NO_VIAG".

    Caso contrário ("FE_VIAG" diferente de zero):
    # NO_VIAG = 0 (não fez viagem)
    SE "F_VIAG" for diferente de zero e ZONA_ORIG e ZONA_DEST forem
        iguais a zero, então a pessoa não fez viagem, atribui-se zero.

    Para cada "ID_PESS" o "NO_VIAG" será incrementado sempre que
```

```
    "F_PESS" for igual a 1

Do contrário, caso "F_VIAG" seja igual a 0, então "NO_VIAG" receberá
    o valor de "NO_VIAG" da linha anterior.
:param passo: Número do passo atual para registro/log
:param df:
:return: retorna o dataframe modificado com o novo NO_PESS
"""
log_tela.info("### PASSO " + str(passo) + " - NO_VIAG")

log_output.info("NO_VIAG: INICIANDO GERAÇÃO DO NO_VIAG")

def gera_no_viag(row):
    # Utilize esta função da seguinte forma:
    #     dataframe.apply(gera_NO_VIAG, axis=1)
    #
    # Retorna 0 se "NO_VIAG" foi gerado e 1 se ocorreu algum erro
    # row.name é o índice da linha específica.

    if row.name == 0:
        # Se for a primeira linha do dataframe, então NO_VIAG = 1.
        if row['FE_VIAG'] == 0:
            df.loc[row.name, 'NO_VIAG'] = 0
        else:
            df.loc[row.name, 'NO_VIAG'] = 1
        return 0

    pessoa_anterior = df.loc[row.name - 1, 'ID_PESS']
    pessoa_atual = df.loc[row.name, 'ID_PESS']

    if pessoa_atual != pessoa_anterior:
        # Se for a primeira linha de ID_PESS, então NO_PESS = 1.
        # Isto leva em consideração que o dataframe é ordenado por ID_PESS,
        # dentro de cada família.
        # Este é um requerimento bem forte.
        df.loc[row.name, 'NO_VIAG'] = 1
    elif row['F_VIAG'] == 1:
        df.loc[row.name, 'NO_VIAG'] = df.loc[row.name - 1, 'NO_VIAG'] + 1
    elif row['F_VIAG'] == 0:
        df.loc[row.name, 'NO_VIAG'] = df.loc[row.name - 1, 'NO_VIAG']
    else:
        log_tela.warning("NO_VIAG: Erro na composição da linha "
                         + str(row.name) + ':\n'
                         + ' => ' + row
        )
        return 1

    if row['FE_VIAG'] == 0:
        df.loc[row.name, 'NO_VIAG'] = 0
    return 0

# A função gera_NO_VIAG é chamada. Devido ao fato dela retornar 1 caso haja
    # algum erro, é possível somar e saber a quantidade de erros ocorrida
erros = df.apply(gera_no_viag, axis=1).sum()
```

```
    if erros > 0:
        log_tela.warning(
            "NO_VIAG: Número de composições em que ocorreu algum erro: " +
            str(erros))
    else:
        log_tela.info("NO_VIAG: Nenhum erro encontrado")
    log_output.info('\n\n================================================\n')

    return df


def passo_id_viag(passo, df):
    """
    Construir "ID_VIAG" e "NO_VIAG"
    Na coluna "ID_VIAG", linha i, ler o valor da linha i da coluna "ID_PESS", e
        concatenar esse valor (12 dígitos) com o número da pessoa,
        que é o valor da linha i da coluna "NO_VIAG" (com 2 dígitos).

    Resultado será um ID_VIAG, que pode se repetir nas linhas, 14 dígitos.
    :param passo: Número do passo atual para registro/log
    :param df:
    :return: retorna o dataframe com o ID_VIAG atualizado
    """
    log_tela.info("### PASSO " + str(passo) + " - ID_VIAG")

    def gera_id_viag(row):
        """
        Gera o ID_VIAG baseado no 'ID_PESS' e no 'NO_VIAG'
        O argumento passado é a "linha".
            Uso:
            od1977['ID_VIAG'] = od1977.apply(gera_ID_VIAG, axis=1)
        Retorna ID_VIAG da respectiva linha
        """
        id_pess = int(row['ID_PESS'])
        no_viag = int(row['NO_VIAG'])
        return int(str(id_pess) + str('%02d' % no_viag))

    # Gerando 'ID_VIAG' concatenando "ID_PESS" e "NO_VIAG"
    df['ID_VIAG'] = df.apply(gera_id_viag, axis=1)
    log_output.info('\n\n================================================\n')

    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")

            # 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 [3]: def passo_correcoes(df):
            # quando ID_PESS for igual a 115200690107,
                # SIT_FAM deve ser 4 (ao invés de 9)+
            # quando ID_PESS for igual a 115200690107 e NO_VIAG for igual a 2,
                # ZONA_ORIG=152 e SUBZONA_ORIG=451 e MUN_ORIG=23
            #
            pass
```

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.')
            od1977 = pd.read_csv('bases/OD_1977.csv', sep=';', decimal=',')

            log_output.info('Lendo arquivo auxiliar UCOD')
            ucod1977 = pd.read_csv('auxiliares/UCOD-1977.csv', sep=';')

            log_output.info('Lendo arquivo auxiliar do setor de atividade')
            setor_ativ1977 = pd.read_csv('auxiliares/setor_ativ-1977.csv', sep=';')

            log_output.info('Lendo arquivo auxiliar de Coordenadas das zonas')
            coord_subzona1977 = pd.read_csv('auxiliares/coord_zonas_1977.csv', sep=';')

            # Definindo variável de deflação de renda:
            deflator = 0.437

            # Filtrando dataframe para pegar apenas uma amostra
            # logging.info('\nFiltering the main dataframe to get just a sample')
            #od1977 = od1977[:1000].copy()

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

            log_output.info('Renomeando coluna UCOD para UCOD_DOM')
            od1977.rename(columns={'UCOD':'UCOD_DOM'}, inplace=True)

            # Esta coluna será adicionada no final do dataframe, como última coluna.
            log_output.info('Criando a coluna UCOD_ESC')
            od1977['UCOD_ESC']=None

            log_output.info('Criando a coluna UCOD_TRAB1')
            od1977['UCOD_TRAB1']=None

            log_output.info('Criando a coluna UCOD_TRAB2')
            od1977['UCOD_TRAB2']=None

            log_output.info('Criando a coluna UCOD_ORIG')
            od1977['UCOD_ORIG']=None

            log_output.info('Criando a coluna UCOD_DEST')
            od1977['UCOD_DEST']=None
```

```
log_output.info('Criando a coluna SERV_PAS_ORIG')
od1977['SERV_PAS_ORIG']=None

log_output.info('Criando a coluna SERV_PAS_DEST')
od1977['SERV_PAS_DEST']=None

log_output.info('Reordenando as colunas')
od1977 = od1977[['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', 'MODO1', 'MODO2', 'MODO3', 'MODO4', '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('Imprimindo a lista de colunas do dataframe da OD.')
log_tela.debug(od1977.columns.tolist())

#log_tela.debug('Descrevendo os dados de toda a base/dataframe- ' +
#                               'count, mean, std, min and max')
#log_tela.debug(od1977.describe())


#Contador de 'PASSO'
passo = 1

# -----
# #Variáveis Independentes de outras variáveis e sem consulta externa

# -----
# ##Passo: "ANO"
od1977 = passo_ano(passo, od1977)
passo += 1

# ----
# ##Passo: "DIA_SEM"
od1977 = passo_dia_sem(passo, od1977)
passo += 1

# -----
# ##Passo: UCODs
```

```
tipos_ucod = ['DOM', 'ESC', 'TRAB1', 'TRAB2', 'ORIG', 'DEST']

for tipo in tipos_ucod:
    od1977 = passo_ucod(passo, od1977, ucod1977, tipo)
    passo += 1

# -----
# ##Passo: "ZONA_DOM"
od1977 = passo_zona_dom(passo, od1977)
passo += 1

# -----
# ##Passo: "SUBZONA_DOM"
od1977 = passo_subzona_dom(passo, od1977)
passo += 1

# -----
# ##Passo: "MUN_DOM"
od1977 = passo_mun_dom(passo, od1977)
passo += 1

# -----
# ##Passo: "F_DOM"
od1977 = passo_f_dom(passo, od1977)
passo += 1

# -----
# ##Passo: "FE_DOM"
od1977 = passo_fe_dom(passo, od1977)
passo += 1

# -----
# ##Passo: "TIPO_DOM"
od1977 = passo_tipo_dom(passo, od1977)
passo += 1

# -----
# ##"TOT_FAM"
od1977 = passo_tot_fam(passo, od1977)
passo += 1

# -----
# ##Passo: "F_FAM"
od1977 = passo_f_fam(passo, od1977)
passo += 1

# -----
# ##"FE_FAM"
od1977 = passo_fe_fam(passo, od1977)
passo += 1

# -----
# ##Passo: "COND_MORA"
od1977 = passo_cond_mora(passo, od1977)
```

```
passo += 1

# -----
# ##"QT_AUTO"
od1977 = passo_qt_auto(passo, od1977)
passo += 1

# -----
# ##Passo: "QT_BICI"
od1977 = passo_qt_bici(passo, od1977)
passo += 1

# -----
# ##Passo: "QT_MOTO"
od1977 = passo_qt_moto(passo, od1977)
passo += 1

# -----
# ##PASSO: "REN_FAM"
od1977 = passo_ren_fam(passo, od1977, deflator)
passo += 1

# -----
# ##Passo: "CD_RENFAM"
od1977 = passo_cd_renfam(passo, od1977)
passo += 1

# -----
# ##Passo: "F_PESS"
od1977 = passo_f_pess(passo, od1977)
passo += 1

# -----
# ##"FE_PESS"
od1977 = passo_fe_pess(passo, od1977)
passo += 1

# -----
# ##Passo: "SIT_FAM"
od1977 = passo_sit_fam(passo, od1977)
passo += 1

# -----
# ##"IDADE"
od1977 = passo_idade(passo, od1977)
passo += 1

# -----
# ##Passo: "SEXO"
od1977 = passo_sexo(passo, od1977)
passo += 1

# -----
# ##Passo: "GRAU_INSTR"
```

```
od1977 = passo_grau_instr(passo, od1977)
passo += 1

# -----
# ##Passo: "OCUP"
od1977 = passo_ocup(passo, od1977)
passo += 1

# -----
# ##Passo: "SETOR_ATIV"
od1977 = passo_setor_ativ(passo, od1977, setor_ativ1977)
passo += 1

# -----
# ##Passo: "REN_IND"
od1977 = passo_ren_ind(passo, od1977, deflator)
passo += 1

# -----
# ##Passo: "CD_RENIND"
od1977 = passo_cd_renind(passo, od1977)
passo += 1

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

# -----
# ##Passo: "F_VIAG"
od1977 = passo_f_viag(passo, od1977)
passo += 1

# -----
# ##"FE_VIAG"
od1977 = passo_fe_viag(passo, od1977)
passo += 1

# -----
# ##Passo: "ZONA_ESC"
passo_zona_esc(passo, od1977)
passo += 1

# -----
# ##Passo: "SUBZONA_ESC"
od1977 = passo_subzona_esc(passo, od1977)
passo += 1

# -----
# ##Passo: "MUN_ESC"
od1977 = passo_mun_esc(passo, od1977)
passo += 1

# -----
# ##Passo: "ESTUDA"
# Este passo deve vir após os passos de localização da escola
od1977 = passo_estuda(passo, od1977)
```

```
passo += 1

# -----
# ##Passo: "ZONA_TRAB1"
od1977 = passo_zona_trab1(passo, od1977)
passo += 1

# -----
# ##Passo: "SUBZONA_TRAB1"
od1977 = passo_subzona_trab1(passo, od1977)
passo += 1

# -----
# ##Passo: "MUN_TRAB1"
od1977 = passo_mun_trab1(passo, od1977)
passo += 1

# -----
# ##Passo: "ZONA_TRAB2"
od1977 = passo_zona_trab2(passo, od1977)
passo += 1

# -----
# ##Passo: "SUBZONA_TRAB2"
od1977 = passo_subzona_trab2(passo, od1977)
passo += 1

# -----
# ##Passo: "MUN_TRAB2"
od1977 = passo_mun_trab2(passo, od1977)
passo += 1

# -----
# ##Passo: "ZONA_ORIG"
od1977 = passo_zona_orig(passo, od1977)
passo += 1

# -----
# ##Passo: "SUBZONA_ORIG"
od1977 = passo_subzona_orig(passo, od1977)
passo += 1

# -----
# ##Passo: "MUN_ORIG"
od1977 = passo_mun_orig(passo, od1977)
passo += 1

# -----
# ##Passo: "ZONA_DEST"
od1977 = passo_zona_dest(passo, od1977)
passo += 1

# -----
# ##Passo: "SUBZONA_DEST"
```

```
od1977 = passo_subzona_dest(passo, od1977)
passo += 1

# -----
# ##Passo: "MUN_DEST"
od1977 = passo_mun_dest(passo, od1977)
passo += 1

# -----
# ##Passo: "SERV_PAS_ORIG"
od1977 = passo_serv_pas_orig(passo, od1977)
passo += 1

# -----
# ##Passo: "SERV_PAS_DEST"
od1977 = passo_serv_pas_dest(passo, od1977)
passo += 1

# -----
# ##Passo: "MOTIVO_ORIG"
od1977 = passo_motivo_orig(passo, od1977)
passo += 1

# -----
# ##Passo: "MOTIVO_DEST"
od1977 = passo_motivo_dest(passo, od1977)
passo += 1

# -----
# ##Passo: "MODO1"
od1977 = passo_modo1(passo, od1977)
passo += 1

# -----
# ##Passo: "MODO2"
od1977 = passo_modo2(passo, od1977)
passo += 1

# -----
# ##Passo: "MODO3"
od1977 = passo_modo3(passo, od1977)
passo += 1

# -----
# ##Passo: "MODO4"
od1977 = passo_modo4(passo, od1977)
passo += 1

# -----
# ##Passo: "MODO_PRIN"
od1977 = passo_modo_prin(passo, od1977)
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"
od1977 = passo_tipo_est_auto(passo, od1977)
passo += 1

# -----
# ##Passo: "VALOR_EST_AUTO"
od1977 = passo_valor_est_auto(passo, od1977, deflator)
passo += 1

# -----
# ##Passo: Coordenadas
tipos_coord = ['DOM', 'ESC', 'TRAB1', 'TRAB2', 'ORIG', 'DEST']

for tipo in tipos_coord:
    od1977 = coord(passo, od1977, coord_subzona1977, tipo, 'X')
    passo += 1
    od1977 = coord(passo, od1977, coord_subzona1977, tipo, 'Y')
    passo += 1

# -----
# ##Passo: "DIST_VIAG"
od1977 = passo_dist_viag(passo, od1977)

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

# -----
# # ATENÇÃO: A ORDEM DESTA EXECUÇÃO É FUNDAMENTAL PARA A GERAÇÃO CORRETA
#               DOS INDICES.
# -----
# ##Passo: NO_DOM
od1977 = passo_no_dom(passo, od1977)
passo += 1

# -----
# ##Passo: "ID_DOM"
od1977 = passo_id_dom(passo, od1977)
passo += 1

# -----
# ##Passo: NO_FAM
od1977 = passo_no_fam(passo, od1977)
passo += 1

# -----
# ##Passo: "ID_FAM"
od1977 = passo_id_fam(passo, od1977)
passo += 1

# -----
```

```
        # ##Passo: NO_PESS
        od1977 = passo_no_pess(passo, od1977)
        passo += 1

        # -----
        # ##Passo: "ID_PESS"
        od1977 = passo_id_pess(passo, od1977)
        passo += 1

        # -----
        # ##Passo: NO_VIAG
        od1977 = passo_no_viag(passo, od1977)
        passo += 1

        # ----
        # ##Passo: "ID_VIAG"
        od1977 = passo_id_viag(passo, od1977)
        passo += 1

        # ----
        # ##Passo: "TOT_VIAG"
        od1977 = passo_tot_viag(passo, od1977)
        passo += 1

        # -----
        # ##Passo: "CD_ENTRE"
        od1977 = passo_cd_entre(passo, od1977)
        passo += 1

        log_tela.info('Salvando dataframe como arquivo CSV')
        # -----
        # ## Salvando o dataframe num arquivo local
        od1977.to_csv('outputs/novo_od1977.csv', sep=';', decimal=',')

        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()
```