PPMI Merge

2022-04-07

Here, we utilize PPMI subject clinical data files (PPMI Study Data) to create a single .csv file that contains longitudinal clinical information of PPMI subjects (included in Analytic Cohort).

Getting the data

1.) Get access to PPMI database and login.
2.) Click download from the navigation bar and select study data.
3.) Select ALL documents and zip files and click download.
4.) Unzip and save the folder on your computer.

Import required modules

```
import pandas as pd
import numpy as np
import boto3
from datetime import datetime
from dateutil import relativedelta

userdir = '/Users/areardon/Desktop/ppmi_merge/'
ppmi_download_path = userdir + 'PPMI_Study_Data_Download/'
invicro_data_path = userdir
genetics_path = userdir + 'genetic_data/'
version = '0.1' # File version to save as i.e. ppmi_merge_v0.1.csv

def create_cohort_df(xlsx, sheet) :
    cohort_df = pd.read_excel(xlsx, sheet)
    cols_delete = ['Unnamed', 'CONDATE'] # Columns to remove from sheet/df
    cohort_df = cohort_df.loc[:, ~cohort_df.columns.str.startswith(tuple(cols_delete))] #
↪   Remove columns in cols_delete

    # Create new columns for Enrollment.Subtype and Consensus.Subtype in pd_df
    cohort_df["Enrollment.Subtype"] = '' # Create new column called 'Enrollment.Subtype'
↪   in cohort_df
    cohort_df["Consensus.Subtype"] = '' # Create new column called 'Consensus.Subtype' in
↪   cohort_df

    if sheet == 'HC' :
        cohort_df['Subgroup'] = 'Healthy Control'
    if sheet == 'SWEDD' :
        cohort_df['Comments'] = '' # Replace comments with empty string because we don't
↪   want to merge comments for SWEDD sheet
```

```
    return cohort_df

def merge_columns(df : pd.DataFrame , old_df_columns : list, new_df_column_name : str,
↪   separator = str) :
    """
    Takes entries in each of old_df_columns and joins them together with a sepator of
↪   choice.  Removes
    empty/nan column entries.
    """
    df = df.replace(r'^\s*$', np.NaN, regex=True) # Fill in empty cells with nan
    df[new_df_column_name] = df[old_df_columns].agg(lambda x: x.dropna().str.cat(sep=
↪   separator), axis=1) # Combine columns
    df.drop(old_df_columns, axis = 1, inplace = True)
    return df

def getNamesFromDataframe(df, str) :
    col_names = [col for col in df.columns if str in col]
    return col_names

def merge_new_csv(df, csv_filename, list_cols, merge_on, merge_how) :
    demo_df = pd.read_csv(ppmi_download_path + csv_filename, skipinitialspace = True)
    demo_df = demo_df[list_cols]
    ppmi_merge = pd.merge(df, demo_df, on = merge_on, how = merge_how) # Merge
    return ppmi_merge

def isNaN(num):
    return num != num

def decode_0_1_no_yes(df, list) :
    for i in list :
        df[i] = df[i].astype(int, errors = "ignore")
        df[i].replace({0 : 'No' , 1 : 'Yes', 2 : 'Uncertain'}, inplace = True) # FIXME
↪   replace Uncertain with NA ?
    return df

def decode_none2severe(df, list):
    for i in list :
        df[i] = df[i].astype(int, errors = "ignore")
        df[i].replace({0 : 'None', 1 : 'Slight', 2: 'Mild', 3 : 'Moderate', 4 :
↪   'Severe'}, inplace = True)
    return df

def condensed_df(df, keep_col_list, rename_col_dict, drop_col_list) :
    new_df = df[keep_col_list]
    new_df.rename(columns = rename_col_dict, inplace = True)
    new_df.dropna(subset = drop_col_list, inplace = True)
    return new_df

#### CLINICAL INFO ####
# Create cohort df
xlsx = pd.ExcelFile(ppmi_download_path +
↪   'Consensus_Committee_Analytic_Datasets_28OCT21.xlsx') # Read in main xlsx file
pd_df = create_cohort_df(xlsx, 'PD') # Create Parkinson's Disease data frame from 'PD'
↪   sheet in 'Consensus_Committee_Analytic_Datasets_28OCT21.xlsx'
```

```
prodromal_df = create_cohort_df(xlsx, 'Prodromal')# Create Prodromal data frame from 'PD'
↪  sheet in 'Consensus_Committee_Analytic_Datasets_28OCT21.xlsx'
hc_df = create_cohort_df(xlsx, 'HC')# Create HC data frame from 'PD' sheet in
↪  'Consensus_Committee_Analytic_Datasets_28OCT21.xlsx'
swedd_df = create_cohort_df(xlsx, 'SWEDD') # Create SWEDD data frame from 'PD' sheet in
↪  'Consensus_Committee_Analytic_Datasets_28OCT21.xlsx'

# Decode Enrollment.Subtype ('Summary' sheet) and Consensus.Subtype ('Summary Analytic'
↪  sheet) of 'Consensus_Committee_Analytic_Datasets_28OCT21.xlsx'
pd_df.loc[(pd_df['ENRLPD'] == 1) & (pd_df['ENRLLRRK2'] == 0) & (pd_df['ENRLGBA'] == 0) &
↪  (pd_df['ENRLSNCA'] == 0), 'Enrollment.Subtype'] = '' # Sporadic - don't need to
↪  define here bc already covered in 'Subgroup' column in PD sheet
pd_df.loc[(pd_df['ENRLPD' ]== 1) & (pd_df['ENRLLRRK2'] == 1), 'Enrollment.Subtype'] =  '
↪  : LRRK2'
pd_df.loc[(pd_df['ENRLPD'] == 1) & (pd_df['ENRLGBA'] == 1), 'Enrollment.Subtype'] = ' :
↪  GBA'
pd_df.loc[(pd_df['ENRLPD'] == 1) & (pd_df['ENRLSNCA'] == 1), 'Enrollment.Subtype'] =  ' :
↪  SNCA'
pd_df.loc[(pd_df['CONPD'] == 1) & (pd_df['CONLRRK2'] == 0) | (pd_df['CONLRRK2'] == '.') &
↪  (pd_df['CONGBA'] == 0) | (pd_df['CONGBA'] == '.') & (pd_df['CONSNCA'] == 0) |
↪  (pd_df['CONSNCA'] == '.') , 'Consensus.Subtype'] = 'Sporadic'
pd_df.loc[(pd_df['CONPD'] == 1) & (pd_df['CONLRRK2']== 1) & (pd_df['CONGBA'] == 0) &
↪  (pd_df['CONSNCA'] == 0), 'Consensus.Subtype'] = 'Genetic : LRRK2'
pd_df.loc[(pd_df['CONPD'] == 1) & (pd_df['CONLRRK2'] == 0) & (pd_df['CONGBA'] == 1) &
↪  (pd_df['CONSNCA'] == 0) , 'Consensus.Subtype'] = 'Genetic : GBA'
pd_df.loc[(pd_df['CONPD'] == 1) & (pd_df['CONLRRK2'] == 1) & (pd_df['CONGBA'] == 1) &
↪  (pd_df['CONSNCA'] == 0), 'Consensus.Subtype'] = 'Genetic : LRRK2 + GBA'
pd_df.loc[(pd_df['CONPD'] == 1) & (pd_df['CONLRRK2'] == 0) & (pd_df['CONGBA'] == 0) &
↪  (pd_df['CONSNCA'] == 1) , 'Consensus.Subtype'] = 'Genetic : SNCA'
pd_df.loc[(pd_df['CONPD'] == 0) & (pd_df['CONPROD']== 0) , 'Consensus.Subtype'] =
↪  'non-PD' # FIXME
pd_df.loc[(pd_df['CONPD'] == 0) & (pd_df['CONPROD']== 1) & (pd_df['CONLRRK2'] == 1) &
↪  (pd_df['CONGBA'] == 0) & (pd_df['CONSNCA'] == 0), 'Consensus.Subtype'] = 'Genetic :
↪  LRRK2 Prodromal'
pd_df.loc[(pd_df['CONPD'] == 0) & (pd_df['CONPROD'] == 1) & (pd_df['CONLRRK2'] == 0) &
↪  (pd_df['CONGBA'] == 1) & (pd_df['CONSNCA'] == 0) , 'Consensus.Subtype'] = 'Genetic :
↪  GBA Prodromal'

# Decode Enrollment.Subtype and Consensus.Subtype to be categorical variables for
↪  prodromal df
prodromal_df.loc[(prodromal_df['ENRLPROD'] == 1) & (prodromal_df['ENRLLRRK2'] == 1),
↪  'Enrollment.Subtype'] = ' : LRRK2 Prodromal'
prodromal_df.loc[(prodromal_df['ENRLPROD'] == 1) & (prodromal_df['ENRLGBA'] == 1),
↪  'Enrollment.Subtype'] =  ' : GBA Prodromal'
prodromal_df.loc[(prodromal_df['ENRLPROD'] == 1) & (prodromal_df['ENRLSNCA'] == 1),
↪  'Enrollment.Subtype'] = ' : SNCA Prodromal'
prodromal_df.loc[(prodromal_df['ENRLPROD'] == 1) & (prodromal_df['ENRLHPSM'] == 1)
↪  ,'Enrollment.Subtype'] = '' # Hyposmia already covered in 'Subgroup' column
prodromal_df.loc[(prodromal_df['ENRLPROD'] == 1) & (prodromal_df['ENRLRBD'] == 1),
↪  'Enrollment.Subtype'] = '' # RBD already covered in 'Subgroup' column
prodromal_df.loc[(prodromal_df['CONPROD'] == 1) & (prodromal_df['PHENOCNV'] == 0) &
↪  (prodromal_df['CONLRRK2'] == 1) & (prodromal_df['CONGBA'] == 0) &
↪  (prodromal_df['CONSNCA'] == 0) ,'Consensus.Subtype'] = 'Genetic : LRRK2 Prodromal'
```

```
prodromal_df.loc[(prodromal_df['CONPROD'] == 1) & (prodromal_df['PHENOCNV'] == 1) &
↪  (prodromal_df['CONLRRK2'] == 1) & (prodromal_df['CONGBA'] == 0) &
↪  (prodromal_df['CONSNCA'] == 0), 'Consensus.Subtype'] = 'Genetic : LRRK2
↪  Phenoconverted'
prodromal_df.loc[(prodromal_df['CONPROD'] == 0) & (prodromal_df['CONLRRK2'] == 1) &
↪  (prodromal_df['CONGBA'] == 0) & (prodromal_df['CONSNCA'] == 0), 'Consensus.Subtype' ]
↪  = 'Genetic : LRRK2 not Prodromal'
prodromal_df.loc[(prodromal_df['CONPROD'] == 1) & (prodromal_df['PHENOCNV'] == 0) &
↪  (prodromal_df['CONLRRK2'] == 0) & (prodromal_df['CONGBA'] == 1) &
↪  (prodromal_df['CONSNCA'] == 0), 'Consensus.Subtype'] = 'Genetic : GBA Prodromal'
prodromal_df.loc[(prodromal_df['CONPROD'] == 1) & (prodromal_df['PHENOCNV'] == 1) &
↪  (prodromal_df['CONLRRK2'] == 0) & (prodromal_df['CONGBA'] == 1) &
↪  (prodromal_df['CONSNCA'] == 0) & (prodromal_df['CONRBD'] == 0), 'Consensus.Subtype']
↪  = 'Genetic : GBA Phenoconverted'
prodromal_df.loc[(prodromal_df['CONPROD'] == 0) & (prodromal_df['CONLRRK2'] == 0) &
↪  (prodromal_df['CONGBA'] == 1) & (prodromal_df['CONSNCA'] == 0) , 'Consensus.Subtype'
↪  ] = 'Genetic : GBA not Prodromal'
prodromal_df.loc[(prodromal_df['CONPROD'] == 1) & (prodromal_df['PHENOCNV'] == 0) &
↪  (prodromal_df['CONLRRK2'] == 1) & (prodromal_df['CONGBA'] == 1) &
↪  (prodromal_df['CONSNCA'] == 0) , 'Consensus.Subtype'] = 'Genetic : LRRK2 + GBA
↪  Prodromal'
prodromal_df.loc[(prodromal_df['CONPROD'] == 1) & (prodromal_df['PHENOCNV'] == 1) &
↪  (prodromal_df['CONLRRK2'] == 1) & (prodromal_df['CONGBA'] == 1) &
↪  (prodromal_df['CONSNCA'] == 0) ,'Consensus.Subtype'] = 'Genetic : LRRK2 + GBA
↪  Phenoconverted'
prodromal_df.loc[(prodromal_df['CONPROD'] == 0) & (prodromal_df['CONLRRK2'] == 1) &
↪  (prodromal_df['CONGBA'] == 1) & (prodromal_df['CONSNCA'] == 0) , 'Consensus.Subtype'
↪  ] = 'Genetic : LRRK2 + GBA not Prodromal'
prodromal_df.loc[(prodromal_df['CONPROD'] == 1) & (prodromal_df['PHENOCNV'] == 0) &
↪  (prodromal_df['CONLRRK2'] == 0) & (prodromal_df['CONGBA'] == 0) &
↪  (prodromal_df['CONSNCA'] == 1), 'Consensus.Subtype'] = 'Genetic : SNCA Prodromal'
prodromal_df.loc[(prodromal_df['CONPROD'] == 1) & (prodromal_df['PHENOCNV'] == 1) &
↪  (prodromal_df['CONLRRK2'] == 0) & (prodromal_df['CONGBA'] == 0) &
↪  (prodromal_df['CONSNCA'] == 1), 'Consensus.Subtype'] = 'Genetic : SNCA
↪  Phenoconverted'
prodromal_df.loc[(prodromal_df['CONPROD'] == 0) & (prodromal_df['CONLRRK2'] == 0) &
↪  (prodromal_df['CONGBA'] == 0) & (prodromal_df['CONSNCA'] == 1) , 'Consensus.Subtype'
↪  ] = 'Genetic : SNCA not Prodromal'
prodromal_df.loc[(prodromal_df['CONPROD'] == 0) & (prodromal_df['CONLRRK2'] == 0) &
↪  (prodromal_df['CONGBA'] == 0) & (prodromal_df['CONSNCA'] == 0) &
↪  (prodromal_df['CONHPSM'] == 0) & (prodromal_df['CONRBD'] == '.'), 'Consensus.Subtype'
↪  ] = 'No Mutation not Prodromal'
prodromal_df.loc[(prodromal_df['CONPROD'] == 1) & (prodromal_df['PHENOCNV'] == 0) &
↪  (prodromal_df['CONLRRK2'] == 0) & (prodromal_df['CONGBA'] == 0) &
↪  (prodromal_df['CONSNCA'] == 0) & (prodromal_df['CONHPSM'] == 1) &
↪  (prodromal_df['CONRBD'] == '.'), 'Consensus.Subtype'] = 'Hyposmia'
prodromal_df.loc[(prodromal_df['CONPROD'] == 1) & (prodromal_df['PHENOCNV'] == 1) &
↪  (prodromal_df['CONLRRK2'] == 0) & (prodromal_df['CONGBA'] == 0) &
↪  (prodromal_df['CONSNCA'] == 0) & (prodromal_df['CONHPSM'] == 1) &
↪  (prodromal_df['CONRBD'] == '.'), 'Consensus.Subtype' ] = 'Hyposmia : Phenoconverted'
prodromal_df.loc[(prodromal_df['CONPROD'] == 0) & (prodromal_df['CONLRRK2'] == 0) &
↪  (prodromal_df['CONGBA'] == 0) & (prodromal_df['CONSNCA'] == 0) &
↪  (prodromal_df['CONHPSM'] == 1) & (prodromal_df['CONRBD'] == '.'), 'Consensus.Subtype'
↪  ] =  'Hyposmia : not Prodromal'
```

```
prodromal_df.loc[(prodromal_df['CONPROD'] == 1) & (prodromal_df['CONRBD'] == 1) &
↪  (prodromal_df['PHENOCNV'] == 0) , 'Consensus.Subtype'] = 'RBD'
prodromal_df.loc[(prodromal_df['CONPROD'] == 1) & (prodromal_df['CONRBD'] == 1) &
↪  (prodromal_df['PHENOCNV'] == 1) & (prodromal_df['CONGBA'] == 0), 'Consensus.Subtype']
↪  = 'RBD : Phenoconverted'
prodromal_df.loc[(prodromal_df['CONPROD'] == 1) & (prodromal_df['CONRBD'] == 1) &
↪  (prodromal_df['PHENOCNV'] == 1) & (prodromal_df['CONGBA'] == 1), 'Consensus.Subtype'
↪  ] = 'RBD : Phenoconverted with GBA'

# Decode Enrollment.Subtype and Consensus.Subtype to be categorical variables for Healthy
↪  Control df
hc_df.loc[(hc_df['ENRLHC'] == 1), 'Enrollment.Subtype'] = '' # Healthy Control already
↪  covered in Subgroup column
hc_df.loc[(hc_df['CONHC'] == 1) & (hc_df['CONLRRK2'] == 0) & (hc_df['CONGBA'] == 0) &
↪  (hc_df['CONSNCA'] == 0), 'Consensus.Subtype'] = 'Healthy Control'
hc_df.loc[(hc_df['CONHC'] == 1) & (hc_df['CONLRRK2'] == 1) & (hc_df['CONGBA'] == 0) &
↪  (hc_df['CONSNCA'] == 0),'Consensus.Subtype'] = 'LRRK2'
hc_df.loc[(hc_df['CONHC'] == 1) & (hc_df['CONLRRK2'] == 0) & (hc_df['CONGBA'] == 1) &
↪  (hc_df['CONSNCA'] == 0), 'Consensus.Subtype'] = 'GBA'
hc_df.loc[(hc_df['CONHC'] == 1) & (hc_df['CONLRRK2'] == 1) & (hc_df['CONGBA'] == 1) &
↪  (hc_df['CONSNCA'] == 0) ,'Consensus.Subtype'] = 'LRRK + GBA'
hc_df.loc[(hc_df['CONHC'] == 1) & (hc_df['CONLRRK2'] == 0) & (hc_df['CONGBA'] == 0) &
↪  (hc_df['CONSNCA'] == 1), 'Consensus.Subtype'] = 'SNCA'
hc_df.loc[(hc_df['CONHC'] == 0), 'Consensus.Subtype'] = 'non-HC'  # FIXME

# Decode Enrollment.Subtype and Consensus.Subtype to be categorical variables for swedd
↪  df
swedd_df.loc[swedd_df['ENRLSWEDD'] == 1 , 'Enrollment.Subtype'] =  'SWEDD Legacy'
swedd_df.loc[swedd_df['CONSWEDD'] == 0 , 'Consensus.Subtype'] = 'SWEDD/PD Active'
swedd_df.loc[swedd_df['CONSWEDD'] == 1 ,'Consensus.Subtype'] =  'SWEDD/non-PD Active'

# Merge the four cohort dataframes (HC, Prodromal, PD, SWEDD)
full_df = pd_df.append([prodromal_df, hc_df, swedd_df]) # Concat all 4 cohort dfs

# Decode and re-organize full_df
full_df['CONPD'].replace({1 : 'Parkinson\'s Disease', 0 : ''}, inplace = True)
full_df['CONPROD'].replace({1 : 'Prodromal',  0 : ''}, inplace = True)
full_df['CONHC'].replace({1 : 'Healthy Control',  0 : ''}, inplace = True)
full_df['CONSWEDD'].replace({1 : 'SWEDD',  0 : 'SWEDD/PD'}, inplace = True)
full_df['Comments'].replace({'MSA' : 'Multiple System Atrophy'}, inplace = True)
full_df = decode_0_1_no_yes(full_df, ['PHENOCNV'])
full_df = merge_columns(full_df, ['CONPD', 'CONPROD', 'CONHC', 'CONSWEDD','Comments'],
↪  'Consensus.Diagnosis', ': ') # Get one column for Consensus Diagnosis with comments
↪  merged in
full_df = merge_columns(full_df, ['Subgroup', 'Enrollment.Subtype'], 'Enroll.Subtype',
↪  '') # Get one column for Enroll.Subtype
full_df = full_df.loc[:, ~full_df.columns.str.startswith(tuple(['CON','ENRL']))] # Remove
↪  columns that begin with CON and ENRL
full_df.rename(columns = {'Cohort' : 'Enroll.Diagnosis' , 'PHENOCNV' :
↪  'Subject.Phenoconverted'}, inplace = True)
full_df = full_df[['PATNO', 'Enroll.Diagnosis', 'Enroll.Subtype', 'Consensus.Diagnosis',
↪  'Consensus.Subtype','Subject.Phenoconverted','DIAG1','DIAG1VIS', 'DIAG2','DIAG2VIS']]
↪  # Reorganize column order
```

```
analytic_cohort_subids = full_df['PATNO'].unique() # subids for analytic cohort

## Add in age info at each visit
ppmi_merge = merge_new_csv(full_df, 'Age_at_visit.csv', ['PATNO', 'EVENT_ID',
↪  'AGE_AT_VISIT'], merge_on = ['PATNO'], merge_how = "outer")
ppmi_merge['EVENT_ID'].fillna('SC', inplace = True) # One subject (41358) has event_id as
↪  NaN - replace with 'SC' because later has a screening event id that we will merge
↪  info on for this sub

## Add demographics, vital signs, and pd diagnosis history info
ppmi_merge = merge_new_csv(ppmi_merge, 'Demographics.csv', ['PATNO','EVENT_ID','SEX',
↪  'HANDED',
↪  'BIRTHDT','AFICBERB','ASHKJEW','BASQUE','HISPLAT','RAASIAN','RABLACK','RAHAWOPI','RAINDALS','RANOS'
↪  merge_on = ['PATNO', 'EVENT_ID'], merge_how = "outer") # Bday, sex, handedness, race
ppmi_merge = merge_new_csv(ppmi_merge, 'Vital_Signs.csv', ['PATNO','EVENT_ID','INFODT',
↪  'WGTKG', 'HTCM'], merge_on = ['PATNO', 'EVENT_ID'], merge_how = "outer") # Visit
↪  date, weight and height
ppmi_merge = merge_new_csv(ppmi_merge,  'PD_Diagnosis_History.csv', ['PATNO', 'EVENT_ID',
↪  'SXDT','PDDXDT'], merge_on = ['PATNO', 'EVENT_ID'], merge_how = "outer") # First
↪  symptom date, PD diagnosis date
ppmi_merge['SEX'].replace({0 : 'Female', 1 : 'Male' }, inplace = True) # Decode sex
ppmi_merge['HANDED'].replace({1 : 'Right', 2 : 'Left', 3 : 'Mixed' }, inplace = True) #
↪  Decode handedness
ppmi_merge.rename(columns = {'AGE_AT_VISIT' : 'Age', 'SEX' : 'Sex', 'HANDED' : 'Handed',
↪  'BIRTHDT' : 'BirthDate', 'WGTKG' : 'Weight(kg)', 'HTCM' : 'Height(cm)', 'SXDT' :
↪  'First.Symptom.Date', 'PDDXDT': 'PD.Diagnosis.Date', 'AFICBERB' :
↪  'African.Berber.Race','ASHKJEW':'Ashkenazi.Jewish.Race', 'BASQUE' : 'Basque.Race',
↪  'HISPLAT' : 'Hispanic.Latino.Race', 'RAASIAN' : 'Asian.Race', 'RABLACK' :
↪  'African.American.Race', 'RAHAWOPI' : 'Hawian.Other.Pacific.Islander.Race',
↪  'RAINDALS' : 'Indian.Alaska.Native.Race', 'RANOS' : 'Not.Specified.Race', 'RAWHITE':
↪  'White.Race'}, inplace = True) # Rename columns

## Add a PD.Disease.Duration variable (in years)
ppmi_merge['PD.Diagnosis.Duration'] = '' # Initialize PD.Diagnosis.Duration variable
for row_num in range(len(ppmi_merge['PD.Diagnosis.Date'])) :
    if isinstance(ppmi_merge['PD.Diagnosis.Date'].loc[row_num], str) and
    ↪  isinstance(ppmi_merge['INFODT'].loc[row_num], str): # If we have both a PD
    ↪  Diagnosis date and an event id date
        diag_year = int(ppmi_merge['PD.Diagnosis.Date'].loc[row_num].split('/')[1]) #
↪  Diagnosis year
        diag_month = int(ppmi_merge['PD.Diagnosis.Date'].loc[row_num].split('/')[0]) #
↪  Diagnosis month
        event_year = int(ppmi_merge['INFODT'].loc[row_num].split('/')[1]) # Visit date
↪  year
        event_month = int(ppmi_merge['INFODT'].loc[row_num].split('/')[0]) # Visit date
↪  month
        diff = relativedelta.relativedelta(datetime(event_year, event_month, 1),
↪  datetime(diag_year, diag_month, 1)) # FIXME ASSUMPTION visit date and diagnosis date
↪  was the first of the month
        ppmi_merge['PD.Diagnosis.Duration'].iloc[row_num] = ((diff.years)*12 +
↪  diff.months)/12 # PD.Diagnosis.Duration in years

# Add Final (FNL) Event ID and info
```

```
ppmi_merge = merge_new_csv(ppmi_merge, 'Conclusion_of_Study_Participation.csv', ['PATNO',
↪  'EVENT_ID', 'COMPLT', 'WDRSN', 'WDDT'], merge_on = ['PATNO', 'EVENT_ID'], merge_how =
↪  "outer") # completed study, whithdrawal reason, withdrawal date
ppmi_merge = decode_0_1_no_yes(ppmi_merge, ['COMPLT'])
ppmi_merge['WDRSN'].replace({1 : 'Adverse Event', 2 : 'Completed study per protocol', 3 :
↪  'Death', 4 : 'Family, care-partner, or social issues', 5 : 'Lost to follow up', 6 :
↪  'Non-compliance with study procedures' , 7 : 'Transportation/Travel issues' , 8 :
↪  'Institutionalized' , 9 : 'Subject transitioning to a new cohort' , 10 : 'Subject
↪  withdrew consent', 11 : 'Investigator decision', 12 : 'Sponsor decision', 13 :
↪  'Informant/caregiver decision' , 20 : 'Other'}, inplace = True)
ppmi_merge.rename(columns = {'COMPLT' : 'Completed.Study' , 'WDRSN':
↪  'Reason.for.Withdrawal','WDDT' : 'Withdrawal.Date'}, inplace = True)

## Add diagnosis change info on event id
diag_vis1 = condensed_df(full_df, ['PATNO', 'DIAG1', 'DIAG1VIS'], {'DIAG1VIS' :
↪  'EVENT_ID'}, ['EVENT_ID'])
diag_vis2 = condensed_df(full_df, ['PATNO', 'DIAG2', 'DIAG2VIS'], {'DIAG2VIS' :
↪  'EVENT_ID'},['EVENT_ID'])
ppmi_merge.drop(['DIAG1', 'DIAG1VIS', 'DIAG2', 'DIAG2VIS'], axis = 1, inplace = True) #
↪  Drop these from ppmi_merge so there aren't duplicates when we merge the diag_vis dfs
ppmi_merge = pd.merge(diag_vis1, ppmi_merge, on = ['EVENT_ID', 'PATNO'], how = "outer" )
↪  # Merge in first diagnosis change
ppmi_merge = pd.merge(diag_vis2, ppmi_merge, on = ['EVENT_ID', 'PATNO'], how = "outer" )
↪  # Merge in second diagnosis change
ppmi_merge['DIAG1'].replace({ 'PD' : 'Parkinson\'s Disease', 'DLB': 'Dimentia with Lewy
↪  Bodies'}, inplace = True) # Decode
ppmi_merge['DIAG2'].replace({ 'MSA' : 'Multiple System Atrophy', 'DLB': 'Dimentia with
↪  Lewy Bodies'}, inplace = True) # Decode
ppmi_merge.rename(columns = {'DIAG1' : 'First.Diagnosis.Change', 'DIAG2' :
↪  'Second.Diagnosis.Change'}, inplace = True) # Rename columns

## Dominant side of disease
ppmi_merge = merge_new_csv(ppmi_merge,
↪  'PPMI_Original_Cohort_BL_to_Year_5_Dataset_Apr2020.csv',['PATNO', 'EVENT_ID',
↪  'DOMSIDE'], merge_on = ['PATNO', 'EVENT_ID'], merge_how = "outer") # Domside
ppmi_merge['DOMSIDE'].replace({1 : 'Left', 2 : 'Right', 3 : 'Symmetric'}, inplace = True)
↪  # Decode

## Participant Motor Function Questionnaire
ppmi_merge = merge_new_csv(ppmi_merge, 'Participant_Motor_Function_Questionnaire.csv',
↪  ['PATNO', 'EVENT_ID', 'PAG_NAME', 'CMPLBY2', 'TRBUPCHR', 'WRTSMLR', 'VOICSFTR',
↪  'POORBAL', 'FTSTUCK', 'LSSXPRSS', 'ARMLGSHK', 'TRBBUTTN', 'SHUFFLE', 'MVSLOW',
↪  'TOLDPD'], merge_on = ['PATNO', 'EVENT_ID'], merge_how = "outer")
ppmi_merge['CMPLBY2'].replace({1: 'Participant', 2 : 'Caregiver' , 3 : 'Participant and
↪  Caregivier'}, inplace = True)
ppmi_merge['PAG_NAME'].replace({'PQUEST' : 'Participant Motor Function Questionnaire'},
↪  inplace = True)
ppmi_merge = decode_0_1_no_yes(ppmi_merge, ['TRBUPCHR', 'WRTSMLR', 'VOICSFTR', 'POORBAL',
↪  'FTSTUCK', 'LSSXPRSS', 'ARMLGSHK', 'TRBBUTTN', 'SHUFFLE', 'MVSLOW', 'TOLDPD'])
ppmi_merge = ppmi_merge.rename(columns = {'PAG_NAME' : 'Motor.Function.Page.Name',
↪  'CMPLBY2' : 'Motor.Function.Source', 'TRBUPCHR' : 'Trouble.Rising.Chair', 'WRTSMLR' :
↪  'Writing.Smaller', 'VOICSFTR' : 'Voice.Softer' , 'POORBAL': 'Poor.Balance' ,
↪  'FTSTUCK' : 'Feet.Stuck', 'LSSXPRSS' : 'Less.Expressive' ,
↪  'ARMLGSHK':'Arms/Legs.Shake', 'TRBBUTTN' : 'Trouble.Buttons' , 'SHUFFLE' :
↪  'Shuffle.Feet' , 'MVSLOW' : 'Slow.Movements' , 'TOLDPD' : 'Been.Told.PD' })
```

```
## Cognitive symptoms - Cognitive Categorization
ppmi_merge = merge_new_csv(ppmi_merge, 'Cognitive_Categorization.csv',['PATNO' ,
↪  'EVENT_ID', 'PAG_NAME', 'COGDECLN', 'FNCDTCOG' , 'COGDXCL' ,'PTCGBOTH' , 'COGSTATE' ,
↪  'COGCAT_TEXT'], merge_on = ['PATNO', 'EVENT_ID'], merge_how = "outer") # Visit date,
↪  weight and height
ppmi_merge = decode_0_1_no_yes(ppmi_merge, ['COGDECLN', 'FNCDTCOG'])
ppmi_merge['COGDXCL'].replace({1 : '90 - 100%', 2 : '50 - 89%', 3 : '10 - 49%', 4 : '0 -
↪  9%'}, inplace = True)
ppmi_merge['PTCGBOTH'].replace({1 : 'Participant', 2 : 'Caregiver', 3 : 'Participant and
↪  Caregiver'}, inplace = True)
ppmi_merge['COGSTATE'].replace({1 : 'Normal Condition', 2 : 'Mild Cognitive Impairment',
↪  3 : 'Dimentia'}, inplace = True)
ppmi_merge = ppmi_merge.rename(columns = {'PAG_NAME' : 'Cognitive.Page.Name', 'COGDECLN'
↪  : 'Cognitive.Decline', 'FNCDTCOG' : 'Functional.Cognitive.Impairment', 'COGDXCL' :
↪  'Confidence.Level.Cognitive.Diagnosis', 'PTCGBOTH' : 'Cognitive.Source', 'COGSTATE' :
↪  'Cognitive.State' , 'COGCAT_TEXT' : 'Cognitive.Tscore.Cat'})
ppmi_merge['Cognitive.Page.Name'].replace({'COGCATG' : 'Cognitive Categorization'},
↪  inplace = True) # Rename

## Cognitive symptoms - MOCA
ppmi_merge = merge_new_csv(ppmi_merge,
↪  'Montreal_Cognitive_Assessment__MoCA_.csv',['PATNO', 'EVENT_ID', 'MCATOT'], merge_on
↪  = ['PATNO', 'EVENT_ID'], merge_how = "outer")
ppmi_merge.rename(columns = {'MCATOT' : 'MOCA.Total'}, inplace = True) # Rename

## Medication Status - Concomitant Med Log # FIXME not a useful way to show medication
↪  status
med_df = pd.read_csv(ppmi_download_path + 'Concomitant_Medication_Log.csv',
↪  skipinitialspace=True) # Medication history
med_df.replace({';' : ','}, regex = True, inplace = True) # Replace ';' with ',' in
↪  med_df
med_df['CMTRT'] = med_df['CMTRT'].str.title() # Capitalize all medication names
med_df['STARTDT'].fillna('NA', inplace = True) # Fillna
med_df['STOPDT'].fillna('NA', inplace = True) # Fillna
med_df = med_df.astype({"STARTDT" : 'str', "STOPDT" :'str'}) # Change start and stop date
↪  to strings
med_df = merge_columns(med_df, ['STARTDT', 'STOPDT'], 'Start_Stop', '-') # Merge columns
↪  Start and stop date together
med_df['Start_Stop'] = '(' + med_df['Start_Stop'].astype(str) + ')'# Put parenthesis
↪  around dates so when you merge it with LED_med_and dose it is more organized
med_df = merge_columns(med_df, ['CMTRT', 'Start_Stop'], 'Medication', ' ')
med_df = med_df.groupby(['PATNO','EVENT_ID'])['Medication'].apply('; '.join)
ppmi_merge = pd.merge(ppmi_merge, med_df, on = ['PATNO', 'EVENT_ID'], how = "outer") #
↪  Merge med_df in
ppmi_merge = ppmi_merge.sort_values(by = ['PATNO','Age']).reset_index(drop = True) # Sort
↪  values by subject and age (similar to event id bc age in order of event id)


## LEDD Medication Status - FIXME Assumtion : If stop date is NA we assume LEDD only
↪  occurred in that month
LEDD_med_df = pd.read_csv(ppmi_download_path + 'LEDD_Concomitant_Medication_Log.csv',
↪  skipinitialspace = True) # LEDD Medication history
```

```
LEDD_med_df = LEDD_med_df[['PATNO', 'LEDD', 'STARTDT', 'STOPDT']] # Keep only
ppmi_merge['INFODT'] = pd.to_datetime(ppmi_merge['INFODT'], format = '%m%Y', errors =
↪  'ignore') # change to INFODT to type datetime so we can sort according to date
LEDD_med_df['STARTDT'] = pd.to_datetime(LEDD_med_df['STARTDT'], format = '%m%Y', errors =
↪  'ignore') # change to STARTDT to type datetime so we can sort according to date
LEDD_med_df['STOPDT'] = pd.to_datetime(LEDD_med_df['STOPDT'], format = '%m%Y', errors =
↪  'ignore') # change to STOPDT to type datetime so we can sort according to date
LEDD_med_df['STOPDT2'] = LEDD_med_df['STOPDT'] # Initialize second stop date variable
LEDD_med_df['STOPDT2'].fillna(LEDD_med_df['STARTDT'], inplace = True) # Fill in NaN stop
↪  dates with start dates # ASSUMPTION START and STOP on same month
LEDD_med_df = LEDD_med_df.merge(LEDD_med_df.apply(lambda s: pd.date_range(s.STARTDT,
↪  s.STOPDT2, freq='MS', inclusive = 'both'),
↪  1).explode().rename('INFODT').dt.strftime('%m/%Y'), left_index=True,
↪  right_index=True) # Create a row for every month/year for LEDD
LEDD_med_df.drop(['STOPDT2'], axis = 1, inplace = True) # Drop
LEDD_med_df.rename(columns = {'STARTDT' : 'LEDD.STARTDT', 'STOPDT': 'LEDD.STOPDT'},
↪  inplace = True) # Rename

# Get a new df with LEDD dates merged onto corect event ids through infodts
ppmi_merge_temp = ppmi_merge[['PATNO', 'EVENT_ID', 'INFODT']]
ppmi_merge_temp = pd.merge(ppmi_merge_temp, LEDD_med_df, on = ['PATNO', 'INFODT'], how =
↪  "left")
ppmi_merge_temp = ppmi_merge_temp[['PATNO', 'EVENT_ID', 'LEDD', 'LEDD.STARTDT',
↪  'LEDD.STOPDT']] # keep only

# Df with LD formulas as variables
LD_only = ppmi_merge_temp[ppmi_merge_temp["LEDD"].str.contains("LD") == True] # Get df
↪  with only columns that contain LD formula (i.e. LD x0.33)
LD_only['STOPDT2'] = LD_only['LEDD.STOPDT'] # Initialize second stop date variable
LD_only['STOPDT2'].fillna(LD_only['LEDD.STARTDT'], inplace = True) # Fill in NaN stop
↪  dates with start dates # ASSUMPTION START and STOP on same month
LD_only = LD_only.merge(LD_only.apply(lambda s: pd.date_range(s['LEDD.STARTDT'],
↪  s.STOPDT2, freq='MS', inclusive = 'both'),
↪  1).explode().rename('INFODT').dt.strftime('%m/%Y'), left_index=True,
↪  right_index=True) # Create a row for every month/year for LEDD
LD_only.drop(['STOPDT2','EVENT_ID','LEDD.STARTDT','LEDD.STOPDT'], axis = 1, inplace =
↪  True) # Drop
ppmi_merge_LD_only = ppmi_merge[['PATNO', 'EVENT_ID', 'INFODT']]
ppmi_merge_LD_only = pd.merge(ppmi_merge_LD_only, LD_only, on = ['PATNO', 'INFODT'], how
↪  = "left")
ppmi_merge_LD_only = ppmi_merge_LD_only[['PATNO', 'EVENT_ID', 'LEDD']] # keep only
ppmi_merge_LD_only.rename(columns = {'LEDD' : 'LD'}, inplace = True)

# Df without LD formula var - only numeric LEDD vars
ppmi_merge_temp = ppmi_merge_temp[ppmi_merge_temp["LEDD"].str.contains("LD") == False] #
↪  Remove LD x .33 etc rows from this df
ppmi_merge_temp = ppmi_merge_temp.astype({'LEDD' : 'float'})
LEDD_max = ppmi_merge_temp.loc[ppmi_merge_temp.groupby(['PATNO',
↪  'EVENT_ID'])['LEDD'].idxmax()].reset_index(drop=True) # Get max
LEDD_max.rename(columns = {'LEDD' : 'LEDD.max'}, inplace = True) # Rename
ppmi_merge = pd.merge(ppmi_merge, LEDD_max, on = ['PATNO','EVENT_ID'], how = "left") #
↪  Merge into ppmi_merge
LEDD_sum = ppmi_merge_temp.groupby(['PATNO', 'EVENT_ID'])['LEDD'].sum().reset_index() #
↪  Get sum
```

```
LEDD_sum.rename(columns = {'LEDD' : 'LEDD.sum'}, inplace = True) # Rename
ppmi_merge = pd.merge(ppmi_merge, LEDD_sum, on = ['PATNO','EVENT_ID'], how = "left") #
↪  Merge into ppmi_merge
ppmi_merge = pd.merge(ppmi_merge, ppmi_merge_LD_only, on = ['PATNO','EVENT_ID'], how =
↪  "left") # Merge LD formlas into ppmi_merge

## GET LEVODOPA ONLY FOR BELOW CALCULATIONS - # FIXME - need more clarification before
↪  adding this
# LEDD_med_df = pd.read_csv(ppmi_download_path + 'LEDD_Concomitant_Medication_Log.csv',
↪  skipinitialspace = True)
# LEDD_med_df = LEDD_med_df[['PATNO','LEDTRT', 'LEDD', 'STARTDT', 'STOPDT']] # Keep only
# levodopa_only = LEDD_med_df[LEDD_med_df["LEDTRT"].str.contains("levodopa", case =
↪  False) == True]
# levodopa_only['STOPDT2'] = levodopa_only['STOPDT'] # Initialize second stop date
↪  variable
# levodopa_only['STOPDT2'].fillna(levodopa_only['STARTDT'], inplace = True) # Fill in NaN
↪  stop dates with start dates # ASSUMPTION START and STOP on same month
# levodopa_only = levodopa_only.merge(levodopa_only.apply(lambda s:
↪  pd.date_range(s['STARTDT'], s.STOPDT2, freq='MS', inclusive = 'both'),
↪  1).explode().rename('INFODT').dt.strftime('%m/%Y'), left_index=True,
↪  right_index=True) # Create a row for every month/year for LEDD
# levodopa_only.drop(['STOPDT2','STARTDT','STOPDT'], axis = 1, inplace = True)
# ppmi_merge_levodopa_only = ppmi_merge[['PATNO', 'EVENT_ID', 'INFODT']]
# ppmi_merge_levodopa_only = pd.merge(ppmi_merge_levodopa_only, levodopa_only, on =
↪  ['PATNO', 'INFODT'], how = "left")
# ppmi_merge_levodopa_only = ppmi_merge_levodopa_only[['PATNO', 'EVENT_ID', 'LEDD']] #
↪  keep only
# ppmi_merge_levodopa_only = ppmi_merge_levodopa_only.drop_duplicates() # with ongoing
↪  stop dates some overlap when merging on infodts so remove dups
# ppmi_merge_levodopa_only =
↪  ppmi_merge_levodopa_only[ppmi_merge_levodopa_only["LEDD"].str.contains("LD") ==
↪  False]
# ppmi_merge_levodopa_only = ppmi_merge_levodopa_only.astype({'LEDD' : 'float'})
# ppmi_merge_levodopa_only = ppmi_merge_levodopa_only.groupby(['PATNO',
↪  'EVENT_ID'])['LEDD'].sum().reset_index()
# ppmi_merge_levodopa_only.rename(columns = {'LEDD' : 'Levodopa.only'}, inplace = True)

# Calculate LD formulas and add to LEDD column numbers # FIXME need to switch calculation
↪  from LEDD to Levadopa only
for row_num in range(len(ppmi_merge)) :
    if ppmi_merge['LD'].loc[row_num] == 'LD x 0.33' :
        ppmi_merge['LEDD.max'].loc[row_num] = float(ppmi_merge['LEDD.max'].loc[row_num])
↪  * 0.33 + float(ppmi_merge['LEDD.max'].loc[row_num])
        ppmi_merge['LEDD.sum'].loc[row_num] = float(ppmi_merge['LEDD.sum'].loc[row_num])
↪  * 0.33 + float(ppmi_merge['LEDD.sum'].loc[row_num])
    elif ppmi_merge['LD'].loc[row_num] == '(150.0 + LD) x 0.33' :
        ppmi_merge['LEDD.max'].loc[row_num] = (150.0 +
↪  float(ppmi_merge['LEDD.max'].loc[row_num])) * 0.33 +
↪  float(ppmi_merge['LEDD.max'].loc[row_num])
        ppmi_merge['LEDD.sum'].loc[row_num] = (150.0 +
↪  float(ppmi_merge['LEDD.sum'].loc[row_num])) * 0.33 +
↪  float(ppmi_merge['LEDD.sum'].loc[row_num])
    elif ppmi_merge['LD'].loc[row_num] == '(600.0 + LD) x 0.33' :
```

```
        ppmi_merge['LEDD.max'].loc[row_num] =  (600.0 +
↪  float(ppmi_merge['LEDD.max'].loc[row_num])) * 0.33 +
↪  float(ppmi_merge['LEDD.max'].loc[row_num])
        ppmi_merge['LEDD.sum'].loc[row_num] =  (600.0 +
↪  float(ppmi_merge['LEDD.sum'].loc[row_num])) * 0.33 +
↪  float(ppmi_merge['LEDD.sum'].loc[row_num])
    elif ppmi_merge['LD'].loc[row_num] == 'LD x 0.5':
        ppmi_merge['LEDD.max'].loc[row_num] = float(ppmi_merge['LEDD.max'].loc[row_num])
↪  * 0.5 + float(ppmi_merge['LEDD.max'].loc[row_num])
        ppmi_merge['LEDD.sum'].loc[row_num] = float(ppmi_merge['LEDD.sum'].loc[row_num])
↪  * 0.5 + float(ppmi_merge['LEDD.sum'].loc[row_num])
    elif ppmi_merge['LD'].loc[row_num] =='(800.0 + LD) x 0.33':
        ppmi_merge['LEDD.max'].loc[row_num] = (800.0 +
↪  float(ppmi_merge['LEDD.max'].loc[row_num])) * 0.33 +
↪  float(ppmi_merge['LEDD.max'].loc[row_num])
        ppmi_merge['LEDD.sum'].loc[row_num] = (800.0 +
↪  float(ppmi_merge['LEDD.sum'].loc[row_num])) * 0.33 +
↪  float(ppmi_merge['LEDD.sum'].loc[row_num])
    elif ppmi_merge['LD'].loc[row_num] == '(300.0 + LD) x 0.33' :
        ppmi_merge['LEDD.max'].loc[row_num] = (300.0 +
↪  float(ppmi_merge['LEDD.max'].loc[row_num])) * 0.33 +
↪  float(ppmi_merge['LEDD.max'].loc[row_num])
        ppmi_merge['LEDD.sum'].loc[row_num] = (300.0 +
↪  float(ppmi_merge['LEDD.sum'].loc[row_num])) * 0.33 +
↪  float(ppmi_merge['LEDD.sum'].loc[row_num])
    elif ppmi_merge['LD'].loc[row_num] == '(225.0 + LD) x 0.33' :
        ppmi_merge['LEDD.max'].loc[row_num] = (225.0 +
↪  float(ppmi_merge['LEDD.max'].loc[row_num])) * 0.33 +
↪  float(ppmi_merge['LEDD.max'].loc[row_num])
        ppmi_merge['LEDD.sum'].loc[row_num] = (225.0 +
↪  float(ppmi_merge['LEDD.sum'].loc[row_num])) * 0.33 +
↪  float(ppmi_merge['LEDD.sum'].loc[row_num])
    elif ppmi_merge['LD'].loc[row_num] == '(450.0 + LD) x 0.33' :
        ppmi_merge['LEDD.max'].loc[row_num] = (450.0 +
↪  float(ppmi_merge['LEDD.max'].loc[row_num])) * 0.33 +
↪  float(ppmi_merge['LEDD.max'].loc[row_num])
        ppmi_merge['LEDD.sum'].loc[row_num] = (450.0 +
↪  float(ppmi_merge['LEDD.sum'].loc[row_num])) * 0.33 +
↪  float(ppmi_merge['LEDD.sum'].loc[row_num])
ppmi_merge.drop(['LD'], axis = 1, inplace = True)


## LEDD Medication Status - FIXME Assumption : If stop date is NA we assume therapy is
↪  ongoing
LEDD_med_df = pd.read_csv(ppmi_download_path + 'LEDD_Concomitant_Medication_Log.csv',
↪  skipinitialspace = True) # LEDD medication history
LEDD_med_df = LEDD_med_df[['PATNO', 'LEDD', 'STARTDT', 'STOPDT']] # keep only
ppmi_merge['INFODT'] = pd.to_datetime(ppmi_merge['INFODT'], format = '%m%Y', errors =
↪  'ignore') # Change to INFODT to type datetime so we can sort according to date
LEDD_med_df['STARTDT'] = pd.to_datetime(LEDD_med_df['STARTDT'], format = '%m%Y', errors =
↪  'ignore') # Change to INFODT to type datetime so we can sort according to date
LEDD_med_df['STOPDT'] = pd.to_datetime(LEDD_med_df['STOPDT'], format = '%m%Y', errors =
↪  'ignore') # Change to INFODT to type datetime so we can sort according to date
```

```
LEDD_med_df['STOPDT3'] = LEDD_med_df['STOPDT']
LEDD_med_df['STOPDT3'].fillna('01/2022', inplace = True) # ASSUMPTION fill in ongoing
↪  stopdate (12/2021 is max infodt in ppmi_merge file)
LEDD_med_df['STOPDT3'] = pd.to_datetime(LEDD_med_df['STOPDT3'], format = '%m%Y', errors =
↪  'ignore') # change to INFODT to type datetime so we can sort according to date
LEDD_med_df = LEDD_med_df.merge(LEDD_med_df.apply(lambda s: pd.date_range(s.STARTDT,
↪  s.STOPDT3, freq='MS', inclusive = 'both'),
↪  1).explode().rename('INFODT').dt.strftime('%m/%Y'), left_index=True,
↪  right_index=True) # Create a row for every month/year for LEDD
LEDD_med_df.rename(columns = {'LEDD' : 'LEDD.ongoing', 'STARTDT' :
↪  'LEDD.STARTDT.ongoing', 'STOPDT3': 'LEDD.STOPDT.ongoing'}, inplace = True) # Rename

# Get a new df with LEDD dates merged onto corect event ids through infodts
ppmi_merge_temp = ppmi_merge[['PATNO', 'EVENT_ID', 'INFODT']]
ppmi_merge_temp = pd.merge(ppmi_merge_temp, LEDD_med_df, on = ['PATNO', 'INFODT'], how =
↪  "left")
ppmi_merge_temp = ppmi_merge_temp[['PATNO', 'EVENT_ID', 'LEDD.ongoing',
↪  'LEDD.STARTDT.ongoing', 'LEDD.STOPDT.ongoing']] # keep only

# Df with LD formulas as variables
LD_only = ppmi_merge_temp[ppmi_merge_temp["LEDD.ongoing"].str.contains("LD")==True] # Get
↪  df with only columns that contain LD formula (i.e. LD x0.33)
LD_only['STOPDT2'] = LD_only['LEDD.STOPDT.ongoing'] # Initialize second stop date
↪  variable
LD_only['STOPDT2'].fillna('01/2022', inplace = True) # Fill in NaN stop dates with start
↪  dates # ASSUMPTION
LD_only['STOPDT2'] = pd.to_datetime(LD_only['STOPDT2'], format = '%m%Y', errors =
↪  'ignore') # change to INFODT to type datetime so we can sort according to date
LD_only = LD_only.merge(LD_only.apply(lambda s: pd.date_range(s['LEDD.STARTDT.ongoing'],
↪  s.STOPDT2, freq='MS', inclusive = 'both'),
↪  1).explode().rename('INFODT').dt.strftime('%m/%Y'), left_index=True,
↪  right_index=True) # Create a row for every month/year for LEDD
LD_only.drop(['STOPDT2','EVENT_ID','LEDD.STARTDT.ongoing','LEDD.STOPDT.ongoing'], axis =
↪  1, inplace = True) # Drop
ppmi_merge_LD_only = ppmi_merge[['PATNO', 'EVENT_ID', 'INFODT']]
ppmi_merge_LD_only = pd.merge(ppmi_merge_LD_only, LD_only, on = ['PATNO', 'INFODT'], how
↪  = "left")
ppmi_merge_LD_only = ppmi_merge_LD_only[['PATNO', 'EVENT_ID', 'LEDD.ongoing']] # keep
↪  only
ppmi_merge_LD_only.rename(columns = {'LEDD.ongoing' : 'LD'}, inplace = True)
ppmi_merge_LD_only = ppmi_merge_LD_only.drop_duplicates() # with ongoing stop dates some
↪  overlap when merging on infodts so remove dups

# Df without LD formula var - only numeric LEDD vars
ppmi_merge_temp = ppmi_merge_temp[ppmi_merge_temp["LEDD.ongoing"].str.contains("LD") ==
↪  False] # Remove LD from this df (add in later)
ppmi_merge_temp = ppmi_merge_temp.astype({'LEDD.ongoing' : 'float'})
LEDD_max = ppmi_merge_temp.loc[ppmi_merge_temp.groupby(['PATNO',
↪  'EVENT_ID'])['LEDD.ongoing'].idxmax()].reset_index(drop=True)
LEDD_max.rename(columns = {'LEDD.ongoing' : 'LEDD.ongoing.max'}, inplace = True) # Rename
ppmi_merge = pd.merge(ppmi_merge, LEDD_max, on = ['PATNO','EVENT_ID'], how = "left") #
↪  Merge into ppmi_merge
LEDD_sum = ppmi_merge_temp.groupby(['PATNO',
↪  'EVENT_ID'])['LEDD.ongoing'].sum().reset_index() #  Sum
```

```
LEDD_sum.rename(columns = {'LEDD.ongoing' : 'LEDD.ongoing.sum'}, inplace = True)
ppmi_merge = pd.merge(ppmi_merge, LEDD_sum, on = ['PATNO','EVENT_ID'], how = "left") #
↪  Merge into ppmi_merge
ppmi_merge = pd.merge(ppmi_merge, ppmi_merge_LD_only, on = ['PATNO','EVENT_ID'], how =
↪  "left") # Merge into ppmi_merge

# Calculate LD formulas and add to LEDD column numbers # FIXME need to switch calculation
↪  from LEDD to Levadopa only
for row_num in range(len(ppmi_merge)) :
    if ppmi_merge['LD'].loc[row_num] == 'LD x 0.33' :
        ppmi_merge['LEDD.ongoing.max'].loc[row_num] =
↪  float(ppmi_merge['LEDD.ongoing.max'].loc[row_num]) * 0.33 +
↪  float(ppmi_merge['LEDD.ongoing.max'].loc[row_num])
        ppmi_merge['LEDD.ongoing.sum'].loc[row_num] =
↪  float(ppmi_merge['LEDD.ongoing.sum'].loc[row_num]) * 0.33 +
↪  float(ppmi_merge['LEDD.ongoing.sum'].loc[row_num])
    elif ppmi_merge['LD'].loc[row_num] == '(150.0 + LD) x 0.33' :
        ppmi_merge['LEDD.ongoing.max'].loc[row_num] = (150.0 +
↪  float(ppmi_merge['LEDD.ongoing.max'].loc[row_num])) * 0.33 +
↪  float(ppmi_merge['LEDD.ongoing.max'].loc[row_num])
        ppmi_merge['LEDD.ongoing.sum'].loc[row_num] = (150.0 +
↪  float(ppmi_merge['LEDD.ongoing.sum'].loc[row_num])) * 0.33 +
↪  float(ppmi_merge['LEDD.ongoing.sum'].loc[row_num])
    elif ppmi_merge['LD'].loc[row_num] == '(600.0 + LD) x 0.33' :
        ppmi_merge['LEDD.ongoing.max'].loc[row_num] =  (600.0 +
↪  float(ppmi_merge['LEDD.ongoing.max'].loc[row_num])) * 0.33 +
↪  float(ppmi_merge['LEDD.ongoing.max'].loc[row_num])
        ppmi_merge['LEDD.ongoing.sum'].loc[row_num] =  (600.0 +
↪  float(ppmi_merge['LEDD.ongoing.sum'].loc[row_num])) * 0.33 +
↪  float(ppmi_merge['LEDD.ongoing.sum'].loc[row_num])
    elif ppmi_merge['LD'].loc[row_num] == 'LD x 0.5':
        ppmi_merge['LEDD.ongoing.max'].loc[row_num] =
↪  float(ppmi_merge['LEDD.ongoing.max'].loc[row_num]) * 0.5 +
↪  float(ppmi_merge['LEDD.ongoing.max'].loc[row_num])
        ppmi_merge['LEDD.ongoing.sum'].loc[row_num] =
↪  float(ppmi_merge['LEDD.ongoing.sum'].loc[row_num]) * 0.5 +
↪  float(ppmi_merge['LEDD.ongoing.sum'].loc[row_num])
    elif ppmi_merge['LD'].loc[row_num] =='(800.0 + LD) x 0.33':
        ppmi_merge['LEDD.ongoing.max'].loc[row_num] = (800.0 +
↪  float(ppmi_merge['LEDD.ongoing.max'].loc[row_num])) * 0.33 +
↪  float(ppmi_merge['LEDD.ongoing.max'].loc[row_num])
        ppmi_merge['LEDD.ongoing.sum'].loc[row_num] = (800.0 +
↪  float(ppmi_merge['LEDD.ongoing.sum'].loc[row_num])) * 0.33 +
↪  float(ppmi_merge['LEDD.ongoing.sum'].loc[row_num])
    elif ppmi_merge['LD'].loc[row_num] == '(300.0 + LD) x 0.33' :
        ppmi_merge['LEDD.ongoing.max'].loc[row_num] = (300.0 +
↪  float(ppmi_merge['LEDD.ongoing.max'].loc[row_num])) * 0.33 +
↪  float(ppmi_merge['LEDD.ongoing.max'].loc[row_num])
        ppmi_merge['LEDD.ongoing.sum'].loc[row_num] = (300.0 +
↪  float(ppmi_merge['LEDD.ongoing.sum'].loc[row_num])) * 0.33 +
↪  float(ppmi_merge['LEDD.ongoing.sum'].loc[row_num])
    elif ppmi_merge['LD'].loc[row_num] == '(225.0 + LD) x 0.33' :
        ppmi_merge['LEDD.ongoing.max'].loc[row_num] = (225.0 +
↪  float(ppmi_merge['LEDD.ongoing.max'].loc[row_num])) * 0.33 +
↪  float(ppmi_merge['LEDD.ongoing.max'].loc[row_num])
```

```
        ppmi_merge['LEDD.ongoing.sum'].loc[row_num] = (225.0 +
↪  float(ppmi_merge['LEDD.ongoing.sum'].loc[row_num])) * 0.33 +
↪  float(ppmi_merge['LEDD.ongoing.sum'].loc[row_num])
    elif ppmi_merge['LD'].loc[row_num] == '(450.0 + LD) x 0.33' :
        ppmi_merge['LEDD.ongoing.max'].loc[row_num] = (450.0 +
↪  float(ppmi_merge['LEDD.ongoing.max'].loc[row_num])) * 0.33 +
↪  float(ppmi_merge['LEDD.ongoing.max'].loc[row_num])
        ppmi_merge['LEDD.ongoing.sum'].loc[row_num] = (450.0 +
↪  float(ppmi_merge['LEDD.ongoing.sum'].loc[row_num])) * 0.33 +
↪  float(ppmi_merge['LEDD.ongoing.sum'].loc[row_num])
ppmi_merge['LEDD.STOPDT.ongoing'].replace({'01/2022'  : np.NaN}, inplace = True) #
↪  Replace filler stop date for ledd ongoing back to nan
ppmi_merge.drop(['LD'], axis = 1, inplace = True)


## Comorbidities # FIXME not useful column
comorbid_df = pd.read_csv(ppmi_download_path + 'Medical_Conditions_Log.csv',
↪  skipinitialspace=True) # Medication history
comorbid_df.replace({';' : ','}, regex = True, inplace = True) # Replace ';' with ','
comorbid_df = comorbid_df[['PATNO', 'EVENT_ID', 'MHDIAGDT','MHTERM']] # keep only
comorbid_df['MHTERM'] = comorbid_df['MHTERM'].str.capitalize() # Capitalize all MHTERM
↪  names
comorbid_df['MHDIAGDT'].fillna('NA', inplace = True) # If no diagnosis date - fill in
↪  with NA
comorbid_df['MHDIAGDT'] = '(' + comorbid_df['MHDIAGDT'].astype(str) + ')'# Put
↪  parentheses around diagnosis date
comorbid_df = comorbid_df.astype({"MHDIAGDT" : 'str'}) # Change date to string
comorbid_df = merge_columns(comorbid_df, ['MHTERM', 'MHDIAGDT'],
↪  'Medical.History.Description(Diagnosis.Date)' , ' ')
comorbid_df =
↪  comorbid_df.groupby(['PATNO','EVENT_ID'])['Medical.History.Description(Diagnosis.Date)'].apply(';
↪  '.join)
ppmi_merge = pd.merge(ppmi_merge, comorbid_df, on = ['PATNO','EVENT_ID'], how = "outer")

## Education (in years)
education_df = pd.read_csv(ppmi_download_path + 'Socio-Economics.csv', skipinitialspace =
↪  True) # Education info
education_df = education_df[['PATNO', 'EVENT_ID', 'EDUCYRS']] # Keep info
education_df.rename(columns = {'EDUCYRS' : 'Education.Years'}, inplace = True) # Rename
education_df = education_df.groupby('PATNO').mean().reset_index() # Take the mean of
↪  education years if there are 2 different number of years for one subject
ppmi_merge = pd.merge(ppmi_merge, education_df, on = ['PATNO'], how = "outer") # Merge

## Add in 'Analytic.Cohort' column
analytic_cohort = ppmi_merge['PATNO'].isin(analytic_cohort_subids)
ppmi_merge['Analytic.Cohort'] = '' # Initialize Analytic.Cohort col
ppmi_merge['Analytic.Cohort'].loc[analytic_cohort] = 'Analytic Cohort'
ppmi_merge['Analytic.Cohort'].fillna('Not Analytic Cohort', inplace = True)

## Reindex
ppmi_merge = ppmi_merge.reindex(columns = ['PATNO', 'EVENT_ID', 'INFODT' ,
↪  'Enroll.Diagnosis', 'Enroll.Subtype', 'Consensus.Diagnosis',
↪  'Consensus.Subtype','Analytic.Cohort','Subject.Phenoconverted','First.Diagnosis.Change',
↪  'Second.Diagnosis.Change',  'First.Symptom.Date', 'PD.Diagnosis.Date',
↪  'PD.Diagnosis.Duration','BirthDate', 'Age', 'Sex', 'Handed', 'Weight(kg)',
↪  'Height(cm)', 'Education.Years', 'DOMSIDE',
↪  'African.Berber.Race','Ashkenazi.Jewish.Race', 'Basque.Race', 'Hispanic.Latino.Race',
↪  'Asian.Race', 'African.American.Race', 'Hawian.Other.Pacific.Islander.Race',
↪  'Indian.Alaska.Native.Race', 'Not.Specified.Race',  'White.Race',
↪  'Motor.Function.Page.Name',    'Motor.Function.Source',    'Trouble.Rising.Chair',
```

```
## Add in other csvs
ppmi_merge = merge_new_csv(ppmi_merge, 'Modified_Boston_Naming_Test.csv', ['PATNO',
↪  'EVENT_ID', 'MBSTNSCR', 'MBSTNCRC', 'MBSTNCRR', 'MBSTNVRS'], merge_on = ['PATNO',
↪  'EVENT_ID'], merge_how = "outer")
ppmi_merge = merge_new_csv(ppmi_merge, 'Clock_Drawing.csv', ['PATNO', 'EVENT_ID',
↪  'CLCKTOT'], merge_on = ['PATNO', 'EVENT_ID'], merge_how = "outer")
ppmi_merge = merge_new_csv(ppmi_merge, 'Benton_Judgement_of_Line_Orientation.csv',
↪  ['PATNO', 'EVENT_ID', 'JLO_TOTCALC'], merge_on = ['PATNO', 'EVENT_ID'], merge_how =
↪  "outer")
ppmi_merge = merge_new_csv(ppmi_merge, 'Letter_-_Number_Sequencing.csv', ['PATNO',
↪  'EVENT_ID', 'LNS_TOTRAW'], merge_on = ['PATNO', 'EVENT_ID'], merge_how = "outer")
ppmi_merge = merge_new_csv(ppmi_merge, 'Modified_Semantic_Fluency.csv', ['PATNO',
↪  'EVENT_ID', 'DVS_SFTANIM'], merge_on = ['PATNO', 'EVENT_ID'], merge_how = "outer")
ppmi_merge = merge_new_csv(ppmi_merge,
↪  'Hopkins_Verbal_Learning_Test_-_Revised.csv',['PATNO', 'EVENT_ID',
↪  'DVT_DELAYED_RECALL', 'DVT_TOTAL_RECALL'], merge_on = ['PATNO', 'EVENT_ID'],
↪  merge_how = "outer")
ppmi_merge = merge_new_csv(ppmi_merge, 'Symbol_Digit_Modalities_Test.csv', ['PATNO',
↪  'EVENT_ID', 'SDMTOTAL'], merge_on = ['PATNO', 'EVENT_ID'], merge_how = "outer")
ppmi_merge.rename(columns = {'CLCKTOT' : 'Clock.Drawing.Total', 'JLOTOTCALC' :
↪  'JOLO.Total', 'LNS_TOTRAW' : 'Letter.Number.Sequencing.Total','DVS_SFTANIM' :
↪  'Semantic.Fluency.Total','DVT_TOTAL_RECALL' : 'DVT.Total.RECALL','SDMTOTAL' :
↪  'Symbol.Digit.Modalities.Total'}, inplace = True)

## REM Sleeep behavior disorder questionnaire
ppmi_merge = merge_new_csv(ppmi_merge, 'REM_Sleep_Behavior_Disorder_Questionnaire.csv',
↪  ['PATNO', 'EVENT_ID', 'PAG_NAME', 'PTCGBOTH', 'DRMVIVID', 'DRMAGRAC', 'DRMNOCTB',
↪  'SLPLMBMV', 'SLPINJUR', 'DRMVERBL', 'DRMFIGHT', 'DRMUMV', 'DRMOBJFL', 'MVAWAKEN',
↪  'DRMREMEM', 'SLPDSTRB', 'STROKE', 'HETRA', 'PARKISM', 'RLS', 'NARCLPSY', 'DEPRS',
↪  'EPILEPSY', 'BRNINFM', 'CNSOTH'], merge_on = ['PATNO', 'EVENT_ID'], merge_how =
↪  "outer")
ppmi_merge.rename(columns = {'PAG_NAME' : 'REM.Sleep.Behavior.Disorder.Page.Name'},
↪  inplace = True)
ppmi_merge['REM.Sleep.Behavior.Disorder.Page.Name'].replace({'REMSLEEP' : 'REM Sleep
↪  Behavior Disorder Questionnaire'}, inplace = True)
ppmi_merge['PTCGBOTH'].replace({1 : 'Participant', 2 : 'Caregiver', 3: 'Participant and
↪  Caregiver'}, inplace = True)
rem_list = ['DRMVIVID', 'DRMAGRAC', 'DRMNOCTB', 'SLPLMBMV', 'SLPINJUR', 'DRMVERBL',
↪  'DRMFIGHT', 'DRMUMV',   'DRMOBJFL', 'MVAWAKEN', 'DRMREMEM', 'SLPDSTRB', 'STROKE',
↪  'HETRA',    'PARKISM',  'RLS','NARCLPSY',   'DEPRS',    'EPILEPSY', 'BRNINFM',
↪  'CNSOTH']
ppmi_merge['RBDTotal.REM'] = ppmi_merge[rem_list].sum(axis = 1) # Add an RBDTotal.REM
↪  column
ppmi_merge = decode_0_1_no_yes(ppmi_merge, rem_list)
ppmi_merge.rename(columns = {'PTCGBOTH' : 'Sleep.Behavior.Source.REM','DRMVIVID' :
↪  'Vivid.Dreams.REM' , 'DRMAGRAC': 'Aggressive.or.Action-packed.Dreams.REM',
↪  'DRMNOCTB':'Nocturnal.Behaviour.REM','SLPLMBMV':'Move.Arms/legs.During.Sleep.REM',
↪  'SLPINJUR':'Hurt.Bed.Partner.REM',   'DRMVERBL':'Speaking.in.Sleep.REM', 'DRMFIGHT':
↪  'Sudden.Limb.Movements.REM',    'DRMUMV':'Complex.Movements.REM',
↪  'DRMOBJFL':'Things.Fell.Down.REM',  'MVAWAKEN':'My.Movements.Awake.Me.REM',
↪  'DRMREMEM':'Remember.Dreams.REM',   'SLPDSTRB':'Sleep.is.Disturbed.REM',
↪  'STROKE':'Stroke.REM',  'HETRA':'Head.Trauma.REM',  'PARKISM':'Parkinsonism.REM',
↪  'RLS':'RLS.REM',    'NARCLPSY':'Narcolepsy.REM',    'DEPRS':'Depression.REM',
↪  'EPILEPSY':'Epilepsy.REM',  'BRNINFM':'Inflammatory.Disease.of.the.Brain.REM',
↪  'CNSOTH':'Other.REM'}, inplace = True)
```

```
#### IMAGING INFO ####
# FIXME - laterality issue?  SUV ?
ppmi_merge = merge_new_csv(ppmi_merge,'DaTScan_Analysis.csv',
↪  ['PATNO','EVENT_ID','DATSCAN_DATE','DATSCAN_CAUDATE_R','DATSCAN_CAUDATE_L','DATSCAN_PUTAMEN_R','DATS
↪  merge_on = ['PATNO', 'EVENT_ID'], merge_how = "outer")
ppmi_merge = merge_new_csv(ppmi_merge,'DaTScan_Imaging.csv',
↪  ['PATNO','EVENT_ID','PAG_NAME','INFODT','DATSCAN','PREVDATDT','SCNLOC','SCNINJCT','VSINTRPT','VSRPTE
↪  merge_on = ['PATNO', 'EVENT_ID'], merge_how = "outer")
ppmi_merge['DATSCAN'].replace({0 : 'Not Completed', 1: 'Completed', 2 : 'Completed using
↪  a previously acquired DaTscan (i.e., acquired prior to participant\'s consent to
↪  PPMI)' }, inplace = True)
ppmi_merge['SCNLOC'].replace({ 1 : 'Site', 2 : 'IND'}, inplace = True )
ppmi_merge['SCNINJCT'].replace({ 1 : 'DaTSCAN', 2 : 'Beta-CIT'}, inplace = True)
ppmi_merge['VSINTRPT'].replace({ '1' : 'Consistent with evidence' , '2' : 'Not consistent
↪  with evidence', '3' : 'No visual interpretation report provided'}, inplace = True)
ppmi_merge['VSRPTELG'].replace({1 : 'Eligible', 2 : 'Not eligible'}, inplace = True)
ppmi_merge.rename(columns = {'INFODT_x' : 'INFODT', 'INFODT_y' : 'DaTScan.INFODT',
↪  'SCNLOC' : 'Location.Scan.Completed' , 'PREVDATDT' :
↪  'Date.DaTscan.Imaging.Completed.Previously', 'SCNINJCT' : 'Scan.Injection',
↪  'VSINTRPT' : 'Visual.Interpretation.Report', 'VSRPTELG' :
↪  'Visual.Interpretation.Report(eligible/not)'}, inplace = True)
ppmi_merge['INFODT'].fillna(ppmi_merge['DaTScan.INFODT'], inplace = True) # Fill in NaN
↪  infodts with datscan infodts if NA

# FIXME - After merging in datscan files, duplicate event ids being created one with
↪  DATSCAN as "Completed" one with DATSCAN as "not completed" - if there are both of
↪  these - keep only "Completed"
duplicate_datscan = ppmi_merge[['PATNO','EVENT_ID']].duplicated(keep = False) # Find
↪  locations of True for duplicated subs w/ 2 MRI at baseline
duplicate_datscan_index = ppmi_merge[duplicate_datscan == True].index.tolist() # Get
↪  index of duplicates
dup_subid_list = [] # Initialize duplicate subid list variable
[dup_subid_list.append(index) for index in duplicate_datscan_index if
↪  ppmi_merge['DATSCAN'][index] == 'Not Completed']# Get the indices of duplicate subids
↪  that were labeled as Not Completed
```

```
ppmi_merge = ppmi_merge.reset_index(drop = True)
[ppmi_merge.drop(index = i, axis = 1, inplace = True) for i in reversed(dup_subid_list)
↪  if ppmi_merge['DATSCAN'][i] == 'Not Completed'] # Get rid of the duplicate subids
↪  that were labeled as Not Completed


## MRI.csv
```

```
mri_df = pd.read_csv(ppmi_download_path + 'Magnetic_Resonance_Imaging__MRI_.csv',
↪  skipinitialspace=True)
mri_df = mri_df[[ 'PATNO', 'EVENT_ID', 'INFODT', 'MRICMPLT', 'MRIWDTI', 'MRIWRSS',
↪  'MRIRSLT', 'MRIRSSDF' ]] # Keep only
mri_df['MRICMPLT'].replace({0 : 'Not Completed', 1 : 'Completed'}, inplace = True) #
↪  Decode
```

```
mri_df = decode_0_1_no_yes(mri_df, ['MRIWDTI','MRIWRSS','MRIRSSDF'])
mri_df['MRIRSLT'].replace({1 : 'Normal', 2 : 'Abnormal, not clinically significant', 3 :
↪  'Abnormal, clinically significant'}, inplace = True) # Decode
mri_df.rename(columns = { 'INFODT' : 'Image.Acquisition.Date', 'MRICMPLT' :
↪  'MRI.Completed', 'MRIWDTI' : 'MRI.DTI' , 'MRIWRSS' : 'MRI.Resting.State' , 'MRIRSLT'
↪  : 'MRI.Results' , 'MRIRSSDF' : 'Resting.State.Dif.Day.PDMed.Use'}, inplace = True) #
↪  Rename

# FIXME Some subjects had two baseline rows - 1 with incomplete MRI.Completed and 1 with
↪  complete as MRI.Complete - I am only keeping the one that is complete bc the data we
↪  have on s3 is complete
duplicate_mri = mri_df[['PATNO','EVENT_ID']].duplicated(keep = False) # Find locations of
↪  True for duplicated subs w/ 2 MRI at baseline
duplicate_mri_index = mri_df[duplicate_mri == True].index.tolist() # Get index of
↪  duplicates
dup_subid_list = [] # Initialize duplicate subid list variable
[dup_subid_list.append(index) for index in duplicate_mri_index if
↪  mri_df['MRI.Completed'][index] == 'Not Completed']# Get the indices of duplicate
↪  subids that were labeled as Not Completed
```

```
mri_df = mri_df.reset_index(drop = True)
[mri_df.drop(index = i, axis = 1, inplace = True) for i in reversed(dup_subid_list) if
↪  mri_df['MRI.Completed'][i] == 'Not Completed'] # Get rid of the duplicate subids that
↪  were labeled as Not Completed (that also have another labeled as completed)
```

```
ppmi_merge = pd.merge(ppmi_merge, mri_df, on = ['PATNO','EVENT_ID'], how = "outer")

## For all subjects - see if we have T1 images in ppmi-image-data bucket for given dates
def search_s3(bucket, prefix, search_string):
    client = boto3.client('s3', region_name="us-east-1")
    paginator = client.get_paginator('list_objects')
    pages = paginator.paginate(Bucket=bucket, Prefix=prefix)
    keys = []
    for page in pages:
        contents = page['Contents']
        for c in contents:
            keys.append(c['Key'])
    if search_string:
        keys = [key for key in keys if search_string in key]
    return keys
keys = search_s3('invicro-ia-object-repository', 'refined/ppmi/data/PPMI/', 'T1w/') #
↪  PPMI1.0 and PPMI2.0

# Set a variable in ppmi_merge 'Subid.Date.TEMP' that is the subid and image acquisition
↪  date to match s3 image to
ppmi_merge['Subid.Date.TEMP'] = ''
for row_num in range(len(ppmi_merge['Image.Acquisition.Date'])) :
    if isinstance(ppmi_merge['Image.Acquisition.Date'].loc[row_num], str) :
        ppmi_merge['Subid.Date.TEMP'].iloc[row_num] =
↪  ppmi_merge["PATNO"].iloc[row_num].astype(str) + '/' +
↪  ppmi_merge["Image.Acquisition.Date"].iloc[row_num].split('/')[1] +
↪  ppmi_merge['Image.Acquisition.Date'].iloc[row_num].split('/')[0] # Combine subid and
↪  date into 1 col in df
```

```
subid_date_ordered = ppmi_merge['Subid.Date.TEMP'].dropna().tolist() # Make column into
↪  list

# Get keys of subjects in ppmi-image-data bucket with T1w/ image folder
woutppmi = [key.split('PPMI/')[1] for key in keys] # Remove PPMI/ from key
woutt1w = [key.split('/T1w/')[0] for key in woutppmi] # Remove 'T1w' from key
s3woutdate = [current_subid_date[:-2] for current_subid_date in woutt1w] # Remove the day
↪  from date - want only yearmonth i.e. 202106
matches = [current_subid_date for current_subid_date in subid_date_ordered if
↪  current_subid_date in s3woutdate] # PPMI images that are in S3 that have T1w

# Add in T1 s3 Info
s3_df = pd.DataFrame(columns = ['PATNO', 'EVENT_ID', 'T1.s3.Image.Name']) # create s3_df
↪  dataframe
for current_subid_date_temp in matches :
    for image_id in woutppmi :
        if current_subid_date_temp in image_id and image_id.endswith('.nii.gz'):
            image_id_split = image_id.split('/')
            s3_df = s3_df.append({'PATNO' : image_id_split[0], 'EVENT_ID' :
↪  ppmi_merge.loc[ppmi_merge['Subid.Date.TEMP']==current_subid_date_temp,'EVENT_ID'].iloc[0],
↪  'T1.s3.Image.Name' : image_id_split[-1]}, ignore_index = True)

# Create a column in s3_df for just object name so later we can merge T1 file with object
↪  name
s3_df['PATNO'] = s3_df['PATNO'].astype(int) # Needed for merge on PATNO
s3_df['Image_ID_merge'] = ''
for row_num in range(len(s3_df['T1.s3.Image.Name'])) :
    image0 = s3_df['T1.s3.Image.Name'].iloc[row_num].split('.')[0] # Get name of image
↪  before .nii.gz
    s3_df['Image_ID_merge'].iloc[row_num] = image0.split('-')[4] # Get ImageID from s3
↪  filename and put in Image_ID_merge column
ppmi_merge = pd.merge(ppmi_merge, s3_df, on = ['PATNO','EVENT_ID'], how = "outer")


#### MDS-UPDRS Scores UPDRS 1-4 as Numeric Variables ####
def setup_updrs_df(updrs_filename, drop_cols, decode_dict, rename_col_dict) :
    updrs_df = pd.read_csv(ppmi_download_path + updrs_filename, skipinitialspace = True)
    updrs_df.drop(drop_cols, axis = 1, inplace = True)
    updrs_df['PAG_NAME'].replace(decode_dict, inplace = True)
    updrs_df.rename(columns = rename_col_dict, inplace = True)
    return updrs_df

updrs_part1_df =
↪  setup_updrs_df('MDS-UPDRS_Part_I.csv',['ORIG_ENTRY','LAST_UPDATE','REC_ID'],
↪  {'NUPDRS1' : 'MDS-UPDRS Part I: Non-Motor Aspects of Experiences of Daily Living'},
↪  {'PAG_NAME' : 'UPDRS.Part1.Page.Name', 'NUPSOURC' : 'UPDRS.Part1.Source'})
updrs_part1_pq_df = setup_updrs_df('MDS-UPDRS_Part_I_Patient_Questionnaire.csv',
↪  ['ORIG_ENTRY','LAST_UPDATE','REC_ID','INFODT'], {'NUPDRS1P' : 'MDS-UPDRS Part I
↪  Patient Questionnaire: Non-Motor Aspects of Experiences of Daily Living', 'NUPDRSP' :
↪  'MDS-UPDRS Part IB and Part II'}, {'PAG_NAME' :
↪  'UPDRS.Part1.Patient.Questionnaire.Page.Name', 'NUPSOURC' : 'UPDRS.Part1.PQ.Source'})
updrs_part2_pq_df = setup_updrs_df('MDS_UPDRS_Part_II__Patient_Questionnaire.csv',
↪  ['ORIG_ENTRY','LAST_UPDATE','REC_ID','INFODT'], {'NUPDRS2P' : 'MDS-UPDRS Part II
↪  Patient Questionnaire: Motor Aspects of Experiences of Daily Living','NUPDRSP' :
↪  'MDS-UPDRS Part IB and Part II'}, {'PAG_NAME' : 'UPDRS.Part2.Page.Name','NUPSOURC' :
↪  'UPDRS.Part2.Source'})
```

```
updrs_part3_dos_df =
↪  setup_updrs_df('MDS-UPDRS_Part_III_ON_OFF_Determination___Dosing.csv',
↪  ['ORIG_ENTRY','LAST_UPDATE','REC_ID','INFODT'], {'NUPDRDOSE' : 'MDS-UPDRS Part III
↪  ON/OFF Determination & Dosing', 'NUPDRDOSER' : 'MDS-UPDRS Part III examination
↪  administered at remote visit'}, {'PAG_NAME' : 'UPDRS.Part3.Dosage.Page.Name'})
updrs_part4_motor_df = setup_updrs_df('MDS-UPDRS_Part_IV__Motor_Complications.csv',
↪  ['ORIG_ENTRY','LAST_UPDATE','REC_ID','INFODT'], {'NUPDRS4' : 'MDS-UPDRS Part IV:
↪  Motor Complications'}, {'PAG_NAME' : 'UPDRS.Part4.Page.Name'})

# UPDRS Part 3
updrs_part3_df = pd.read_csv(ppmi_download_path + 'MDS_UPDRS_Part_III.csv',
↪  skipinitialspace = True)
updrs_part3_df.drop(['ORIG_ENTRY','LAST_UPDATE','REC_ID','INFODT'], axis = 1, inplace =
↪  True)
updrs_part3_df.rename(columns = {'PAG_NAME' : 'UPDRS.Part3.Page.Name'}, inplace = True)

# Split up UPDRS Part 3 into four parts
nupdrs3 = updrs_part3_df.loc[updrs_part3_df['UPDRS.Part3.Page.Name'] == 'NUPDRS3']
nupdrs3A = updrs_part3_df.loc[updrs_part3_df['UPDRS.Part3.Page.Name'] == 'NUPDRS3A']
nupdr3OF = updrs_part3_df.loc[updrs_part3_df['UPDRS.Part3.Page.Name'] == 'NUPDR3OF']
nupdr3ON = updrs_part3_df.loc[updrs_part3_df['UPDRS.Part3.Page.Name'] == 'NUPDR3ON']

# Change all UPDRS dataframe cols begin with 'N' to floats
updrs_list = [updrs_part1_df, updrs_part1_pq_df,updrs_part2_pq_df, nupdrs3, nupdrs3A,
↪  nupdr3OF, nupdr3ON, updrs_part3_dos_df, updrs_part4_motor_df]
for df in updrs_list :
    for col_name in df :
        if col_name.startswith('N') :
            df[col_name] = pd.to_numeric(df[col_name], errors = 'coerce', downcast =
↪  'float')

# Create a copy of each dataframe to use later to create categorical versions of
↪  varibales
updrs_part1_df_copy = updrs_part1_df.copy()
updrs_part1_pq_df_copy = updrs_part1_pq_df.copy()
updrs_part2_pq_df_copy = updrs_part2_pq_df.copy()
nupdrs3_copy = nupdrs3.copy()
nupdrs3A_copy = nupdrs3A.copy()
nupdr3OF_copy = nupdr3OF.copy()
nupdr3ON_copy = nupdr3ON.copy()
updrs_part3_dos_df_copy = updrs_part3_dos_df.copy()
updrs_part4_motor_df_copy = updrs_part4_motor_df.copy()

# For all UPDRS df columns - add the respective extension for which UPDRS assessment it
↪  is
def add_extension_to_column_names(df, skip_col_list, ext):
    for col_name in df :
        if col_name not in skip_col_list :
            df.rename(columns = {str(col_name) : str(col_name) + ext }, inplace = True)
    return df

# Add extensions to updrs dfs
updrs_list_str = ['.UPDRS1','.UPDRS1','.UPDRS2', '.UPDRS3','.UPDRS3A', '.UPDR3OF',
↪  '.UPDR3ON','.UPDRDOSE','.UPDRS4'] # extensions
```

```
for i in range(len(updrs_list)) :
    updrs_list[i] = add_extension_to_column_names(updrs_list[i],
↪  ['PATNO','EVENT_ID','INFODT'], updrs_list_str[i])

# Merge the four UPDRS3 dfs together
updrs3 = pd.merge(nupdrs3, nupdrs3A, on = ['PATNO', 'EVENT_ID'], how = "outer")
updrs3 = pd.merge(updrs3, nupdr3OF,  on = ['PATNO', 'EVENT_ID'], how = "outer")
updrs3 = pd.merge(updrs3, nupdr3ON,  on = ['PATNO', 'EVENT_ID'], how = "outer")

# Create one df for updrs_numeric
updrs_numeric = pd.merge(updrs_part1_df, updrs_part1_pq_df , on = ['PATNO', 'EVENT_ID'],
↪  how = "outer")
updrs_numeric = pd.merge(updrs_numeric, updrs_part2_pq_df , on = ['PATNO', 'EVENT_ID'],
↪  how = "outer")
updrs_numeric = pd.merge(updrs_numeric, updrs3, on = ['PATNO', 'EVENT_ID'], how =
↪  "outer")
updrs_numeric = pd.merge(updrs_numeric,updrs_part3_dos_df , on = ['PATNO', 'EVENT_ID'],
↪  how = "outer")
updrs_numeric = pd.merge(updrs_numeric, updrs_part4_motor_df, on = ['PATNO', 'EVENT_ID'],
↪  how = "outer")

## Keep only variables in .Num that are numeric variables
numeric_vars = []
for col_name in updrs_numeric :
    if col_name.startswith('NP') or col_name.startswith('PATNO') or
    ↪  col_name.startswith('EVENT') or col_name.startswith('NHY'):
        numeric_vars.append(col_name)
updrs_numeric = updrs_numeric[numeric_vars]

# Rename columns in updrs_numeric
updrs_numeric.rename(columns = {'NHY.UPDRS3' : 'Hoehn.and.Yahr.Stage.UPDRS3',
'NP3BRADY.UPDRS3' : 'Global.Spontaneity.of.Movement.UPDRS3', 'NP3PTRMR.UPDRS3' :
'Postural.Tremor.Right.Hand.UPDRS3' , 'NP3PTRML.UPDRS3' : 'Postural.Tremor.Left.Hand.UPDRS3'
, 'NP3KTRMR.UPDRS3' : 'Kinetic.Tremor.Right.Hand.UPDRS3', 'NP3KTRML.UPDRS3' :
'Kinetic.Tremor.Left.Hand.UPDRS3', 'NP3RTARU.UPDRS3' : 'Rest.Tremor.Amplitude.RUE.UPDRS3',
'NP3RTALU.UPDRS3' : 'Rest.Tremor.Amplitude.LUE.UPDRS3', 'NP3RTARL.UPDRS3' :
'Rest.Tremor.Amplitude.RLE.UPDRS3' ,'NP3RTALL.UPDRS3' : 'Rest.Tremor.Amplitude.LLE.UPDRS3'
,'NP3RTALJ.UPDRS3' : 'Rest.Tremor.Amplitude.Lip.Jaw.UPDRS3', 'NP3RTCON.UPDRS3' :
'Constancy.of.Rest.Tremor.UPDRS3', 'NP3SPCH.UPDRS3' : 'UPDRS3.Speech.Difficulty.UPDRS3',
'NP3FACXP.UPDRS3' : 'Facial.Expression.Difficulty.UPDRS3' , 'NP3RIGN.UPDRS3' :
'Rigidity.Neck.UPDRS3' , 'NP3RIGRU.UPDRS3' : 'Rigidity.RUE.UPDRS3', 'NP3RIGLU.UPDRS3'
: 'Rigidity.LUE.UPDRS3', 'NP3RIGRL.UPDRS3' : 'Rigidity.RLE.UPDRS3', 'NP3RIGLL.UPDRS3'
: 'Rigidity.LLE.UPDRS3', 'NP3FTAPR.UPDRS3' : 'Finger.Tapping.Right.Hand.UPDRS3'
,'NP3FTAPL.UPDRS3' : 'Finger.Tapping.Left.Hand.UPDRS3' ,'NP3HMOVR.UPDRS3' :
'Hand.Movements.Right.Hand.UPDRS3', 'NP3HMOVL.UPDRS3' : 'Hand.Movements.Left.Hand.UPDRS3',
'NP3PRSPR.UPDRS3' : 'Pronation.Supination.Right.Hand.UPDRS3', 'NP3PRSPL.UPDRS3' :
'Pronation.Supination.Left.Hand.UPDRS3' , 'NP3TTAPR.UPDRS3' : 'Toe.Tapping.Right.Foot.UPDRS3'
, 'NP3TTAPL.UPDRS3' : 'Toe.Tapping.Left.Foot.UPDRS3', 'NP3LGAGR.UPDRS3' :
'Leg.Agility.Right.Leg.UPDRS3', 'NP3LGAGL.UPDRS3' : 'Leg.Agility.Left.Leg.UPDRS3',
'NP3RISNG.UPDRS3' : 'UPDRS3.Rising.from.Chair.UPDRS3', 'NP3GAIT.UPDRS3'
: 'Gait.Problems.UPDRS3' ,'NP3FRZGT.UPDRS3' : 'Freezing.of.Gait.UPDRS3'
,'NP3PSTBL.UPDRS3' : 'Postural.Stability.Problems.UPDRS3', 'NP3POSTR.UPDRS3'
: 'Posture.Problems.UPDRS3' , 'NP3TOT.UPDRS3':'UPDRS.Part3.Total.UPDRS3',
'NP3SPCH.UPDRS3A' : 'UPDRS3.Speech.Difficulty.UPDRS3A', 'NP3FACXP.UPDRS3A' :
'Facial.Expression.Difficulty.UPDRS3A' , 'NP3RIGN.UPDRS3A' : 'Rigidity.Neck.UPDRS3A'
, 'NP3RIGRU.UPDRS3A' : 'Rigidity.RUE.UPDRS3A', 'NP3RIGLU.UPDRS3A' :
'Rigidity.LUE.UPDRS3A', 'NP3RIGRL.UPDRS3A' : 'Rigidity.RLE.UPDRS3A', 'NP3RIGLL.UPDRS3A'
: 'Rigidity.LLE.UPDRS3A', 'NP3FTAPR.UPDRS3A' : 'Finger.Tapping.Right.Hand.UPDRS3A'
,'NP3FTAPL.UPDRS3A' : 'Finger.Tapping.Left.Hand.UPDRS3A' ,'NP3HMOVR.UPDRS3A' :
'Hand.Movements.Right.Hand.UPDRS3A', 'NP3HMOVL.UPDRS3A' : 'Hand.Movements.Left.Hand.UPDRS3A'
```

```
## Subscore info from :
# https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7242837/
#
↪  https://www.movementdisorders.org/MDS-Files1/PDFs/Rating-Scales/MDS-UPDRS_English_FINAL.pdf

## Add Brady-Rigidity, Tremor and PIGD Subscores
def calculate_subscores(df, new_col_name, subscore_list) :
    subscore_only = df[subscore_list] # extract df of only cols we are interested in for
↪  current subscore
    df[new_col_name] = 0 # initalize new col for subscore
    idx  = subscore_only.loc[pd.isnull(subscore_only).any(1), :].index
    df[new_col_name] = subscore_only.sum(axis = 1) # Get sum of subscore
    df[new_col_name].iloc[idx] = np.nan # Replace rows that should be nan with nan
    return df

updrs_numeric = calculate_subscores(updrs_numeric, 'Brady.Rigidity.Subscore.UPDRS3',
↪  ['Rigidity.Neck.UPDRS3',  'Rigidity.RUE.UPDRS3',  'Rigidity.LUE.UPDRS3',
↪  'Rigidity.RLE.UPDRS3',  'Rigidity.LLE.UPDRS3', 'Finger.Tapping.Right.Hand.UPDRS3' ,
↪  'Finger.Tapping.Left.Hand.UPDRS3' ,
↪  'Hand.Movements.Right.Hand.UPDRS3','Hand.Movements.Left.Hand.UPDRS3','Pronation.Supination.Right.Ha
↪  'Pronation.Supination.Left.Hand.UPDRS3',
↪  'Toe.Tapping.Right.Foot.UPDRS3','Toe.Tapping.Left.Foot.UPDRS3','Leg.Agility.Right.Leg.UPDRS3',
↪  'Leg.Agility.Left.Leg.UPDRS3'])
updrs_numeric = calculate_subscores(updrs_numeric, 'Brady.Rigidity.Subscore.UPDRS3A',
↪  ['Rigidity.Neck.UPDRS3A', 'Rigidity.RUE.UPDRS3A', 'Rigidity.LUE.UPDRS3A',
↪  'Rigidity.RLE.UPDRS3A', 'Rigidity.LLE.UPDRS3A', 'Finger.Tapping.Right.Hand.UPDRS3A'
↪  ,'Finger.Tapping.Left.Hand.UPDRS3A','Hand.Movements.Right.Hand.UPDRS3A',
↪  'Hand.Movements.Left.Hand.UPDRS3A', 'Pronation.Supination.Right.Hand.UPDRS3A' ,
↪  'Pronation.Supination.Left.Hand.UPDRS3A' , 'Toe.Tapping.Right.Foot.UPDRS3A',
↪  'Toe.Tapping.Left.Foot.UPDRS3A' , 'Leg.Agility.Right.Leg.UPDRS3A',
↪  'Leg.Agility.Left.Leg.UPDRS3A'])
updrs_numeric = calculate_subscores(updrs_numeric, 'Brady.Rigidity.Subscore.UPDR3ON',
↪  ['Rigidity.Neck.UPDR3ON', 'Rigidity.RUE.UPDR3ON', 'Rigidity.LUE.UPDR3ON',
↪  'Rigidity.RLE.UPDR3ON', 'Rigidity.LLE.UPDR3ON', 'Finger.Tapping.Right.Hand.UPDR3ON'
↪  ,'Finger.Tapping.Left.Hand.UPDR3ON','Hand.Movements.Right.Hand.UPDR3ON','Hand.Movements.Left.Hand.U
↪  'Pronation.Supination.Right.Hand.UPDR3ON', 'Pronation.Supination.Left.Hand.UPDR3ON',
↪  'Toe.Tapping.Right.Foot.UPDR3ON', 'Toe.Tapping.Left.Foot.UPDR3ON',
↪  'Leg.Agility.Right.Leg.UPDR3ON', 'Leg.Agility.Left.Leg.UPDR3ON'])
updrs_numeric = calculate_subscores(updrs_numeric, 'Brady.Rigidity.Subscore.UPDR3OF',
↪  ['Rigidity.Neck.UPDR3OF', 'Rigidity.RUE.UPDR3OF', 'Rigidity.LUE.UPDR3OF',
↪  'Rigidity.RLE.UPDR3OF', 'Rigidity.LLE.UPDR3OF', 'Finger.Tapping.Right.Hand.UPDR3OF'
↪  ,'Finger.Tapping.Left.Hand.UPDR3OF' ,
↪  'Hand.Movements.Right.Hand.UPDR3OF','Hand.Movements.Left.Hand.UPDR3OF',
↪  'Pronation.Supination.Right.Hand.UPDR3OF', 'Pronation.Supination.Left.Hand.UPDR3OF',
↪  'Toe.Tapping.Right.Foot.UPDR3OF', 'Toe.Tapping.Left.Foot.UPDR3OF',
↪  'Leg.Agility.Right.Leg.UPDR3OF','Leg.Agility.Left.Leg.UPDR3OF'])

updrs_numeric = calculate_subscores(updrs_numeric, 'Tremor.Subscore.UPDRS3',
↪  ['Tremor.UPDRS2', 'Postural.Tremor.Right.Hand.UPDRS3'
↪  ,'Postural.Tremor.Left.Hand.UPDRS3'
↪  ,'Kinetic.Tremor.Right.Hand.UPDRS3','Kinetic.Tremor.Left.Hand.UPDRS3',
↪  'Rest.Tremor.Amplitude.RUE.UPDRS3','Rest.Tremor.Amplitude.LUE.UPDRS3',
↪  'Rest.Tremor.Amplitude.RLE.UPDRS3' , 'Rest.Tremor.Amplitude.LLE.UPDRS3' ,
↪  'Rest.Tremor.Amplitude.Lip.Jaw.UPDRS3', 'Constancy.of.Rest.Tremor.UPDRS3'])
```

```
updrs_numeric = calculate_subscores(updrs_numeric, 'Tremor.Subscore.UPDRS3A',
↪  ['Tremor.UPDRS2', 'Postural.Tremor.Right.Hand.UPDRS3A'
↪  ,'Postural.Tremor.Left.Hand.UPDRS3A' , 'Kinetic.Tremor.Right.Hand.UPDRS3A',
↪  'Kinetic.Tremor.Left.Hand.UPDRS3A', 'Rest.Tremor.Amplitude.RUE.UPDRS3A',
↪  'Rest.Tremor.Amplitude.LUE.UPDRS3A', 'Rest.Tremor.Amplitude.RLE.UPDRS3A' ,
↪  'Rest.Tremor.Amplitude.LLE.UPDRS3A' , 'Rest.Tremor.Amplitude.Lip.Jaw.UPDRS3A',
↪  'Constancy.of.Rest.Tremor.UPDRS3A'])
updrs_numeric = calculate_subscores(updrs_numeric, 'Tremor.Subscore.UPDR3ON',
↪  ['Tremor.UPDRS2','Postural.Tremor.Right.Hand.UPDR3ON'
↪  ,'Postural.Tremor.Left.Hand.UPDR3ON',
↪  'Kinetic.Tremor.Right.Hand.UPDR3ON','Kinetic.Tremor.Left.Hand.UPDR3ON',
↪  'Rest.Tremor.Amplitude.RUE.UPDR3ON', 'Rest.Tremor.Amplitude.LUE.UPDR3ON',
↪  'Rest.Tremor.Amplitude.RLE.UPDR3ON' , 'Rest.Tremor.Amplitude.LLE.UPDR3ON' ,
↪  'Rest.Tremor.Amplitude.Lip.Jaw.UPDR3ON', 'Constancy.of.Rest.Tremor.UPDR3ON'])
updrs_numeric = calculate_subscores(updrs_numeric, 'Tremor.Subscore.UPDR3OF',
↪  ['Tremor.UPDRS2','Postural.Tremor.Right.Hand.UPDR3OF'
↪  ,'Postural.Tremor.Left.Hand.UPDR3OF' ,
↪  'Kinetic.Tremor.Right.Hand.UPDR3OF','Kinetic.Tremor.Left.Hand.UPDR3OF',
↪  'Rest.Tremor.Amplitude.RUE.UPDR3OF' , 'Rest.Tremor.Amplitude.LUE.UPDR3OF',
↪  'Rest.Tremor.Amplitude.RLE.UPDR3OF' ,'Rest.Tremor.Amplitude.LLE.UPDR3OF' ,
↪  'Rest.Tremor.Amplitude.Lip.Jaw.UPDR3OF', 'Constancy.of.Rest.Tremor.UPDR3OF'])

updrs_numeric = calculate_subscores(updrs_numeric, 'PIGD.Subscore.UPDRS3',
↪  ['Walking.Difficulty.UPDRS2' ,  'Freezing.while.Walking.UPDRS2'
↪  ,'Gait.Problems.UPDRS3'  , 'Freezing.of.Gait.UPDRS3' ,
↪  'Postural.Stability.Problems.UPDRS3'])
updrs_numeric = calculate_subscores(updrs_numeric, 'PIGD.Subscore.UPDRS3A',
↪  ['Walking.Difficulty.UPDRS2' ,
↪  'Freezing.while.Walking.UPDRS2','Gait.Problems.UPDRS3A' ,'Freezing.of.Gait.UPDRS3A' ,
↪  'Postural.Stability.Problems.UPDRS3A'])
updrs_numeric = calculate_subscores(updrs_numeric, 'PIGD.Subscore.UPDR3ON',
↪  ['Walking.Difficulty.UPDRS2' ,  'Freezing.while.Walking.UPDRS2'
↪  ,'Gait.Problems.UPDR3ON' , 'Freezing.of.Gait.UPDR3ON' ,
↪  'Postural.Stability.Problems.UPDR3ON'])
updrs_numeric = calculate_subscores(updrs_numeric, 'PIGD.Subscore.UPDR3OF',
↪  ['Walking.Difficulty.UPDRS2' ,  'Freezing.while.Walking.UPDRS2' ,
↪  'Gait.Problems.UPDR3OF' , 'Freezing.of.Gait.UPDR3OF' ,
↪  'Postural.Stability.Problems.UPDR3OF'])

updrs_numeric = add_extension_to_column_names(updrs_numeric, ['PATNO', 'EVENT_ID',
↪  'PIGD.Subscore.UPDRS3', 'Tremor.Subscore.UPDRS3', 'Brady.Rigidity.Subscore.UPDRS3',
↪  'PIGD.Subscore.UPDRS3A', 'Tremor.Subscore.UPDRS3A',
↪  'Brady.Rigidity.Subscore.UPDRS3A', 'PIGD.Subscore.UPDR3ON',
↪  'Tremor.Subscore.UPDR3ON', 'Brady.Rigidity.Subscore.UPDR3ON',
↪  'PIGD.Subscore.UPDR3OF', 'Tremor.Subscore.UPDR3OF',
↪  'Brady.Rigidity.Subscore.UPDR3OF'], '.Num') # Add a .Num extension to column names w
↪  updrs numeric vars


### UPDRS CATEGORICAL ####
# UPDRS3 (four dataframes) decode and rename in loop
updrs3_df_list = [nupdrs3_copy, nupdrs3A_copy, nupdr3ON_copy, nupdr3OF_copy]
ext_list = ['.UPDRS3', '.UPDRS3A', '.UPDR3ON', '.UPDR3OF']
```

```
count = 0
for df in [nupdrs3_copy, nupdrs3A_copy, nupdr3ON_copy, nupdr3OF_copy] :
    df = decode_none2severe(df, ['NP3SPCH', 'NP3FACXP', 'NP3RIGN', 'NP3RIGRU',
↪  'NP3RIGLU', 'NP3RIGRL', 'NP3RIGLL', 'NP3FTAPR', 'NP3FTAPL', 'NP3HMOVR', 'NP3HMOVL',
↪  'NP3PRSPR', 'NP3PRSPL', 'NP3TTAPR', 'NP3TTAPL', 'NP3LGAGR', 'NP3LGAGL', 'NP3RISNG',
↪  'NP3GAIT', 'NP3FRZGT', 'NP3PSTBL', 'NP3POSTR', 'NP3BRADY', 'NP3PTRMR', 'NP3PTRML',
↪  'NP3KTRMR', 'NP3KTRML', 'NP3RTARU', 'NP3RTALU', 'NP3RTARL', 'NP3RTALL', 'NP3RTALJ',
↪  'NP3RTCON'])
    df['DBS_STATUS'].replace({0 : 'OFF', 1 : 'ON'}, inplace = True)
    df = decode_0_1_no_yes(df, ['DYSKPRES','DYSKIRAT'])
    df['NHY'].replace({0 : 'Asymptomatic', 1 : 'Unilateral Movement Only', 2: 'Bilateral
↪  involvement without impairment of balance', 3 : 'Mild to moderate involvement',4:
↪  'Severe disability', 5 : 'Wheelchair bound or bedridden'}, inplace = True)
    df['PDTRTMNT'].replace({0 : False , 1: True}, inplace = True)
    df.rename(columns = {'DBS_STATUS' : 'Deep.Brain.Stimulation.Treatment' , 'NP3SPCH' :
↪  'UPDRS3.Speech.Difficulty', 'NP3FACXP' : 'Facial.Expression.Difficulty' , 'NP3RIGN' :
↪  'Rigidity.Neck' , 'NP3RIGRU' : 'Rigidity.RUE', 'NP3RIGLU' : 'Rigidity.LUE',
↪  'NP3RIGRL' : 'Rigidity.RLE', 'NP3RIGLL' : 'Rigidity.LLE', 'NP3FTAPR' :
↪  'Finger.Tapping.Right.Hand' ,'NP3FTAPL' : 'Finger.Tapping.Left.Hand' ,'NP3HMOVR' :
↪  'Hand.Movements.Right.Hand', 'NP3HMOVL' : 'Hand.Movements.Left.Hand','NP3PRSPR' :
↪  'Pronation.Supination.Right.Hand', 'NP3PRSPL' : 'Pronation.Supination.Left.Hand' ,
↪  'NP3TTAPR' : 'Toe.Tapping.Right.Foot' , 'NP3TTAPL' : 'Toe.Tapping.Left.Foot',
↪  'NP3LGAGR' : 'Leg.Agility.Right.Leg', 'NP3LGAGL' : 'Leg.Agility.Left.Leg', 'NP3RISNG'
↪  : 'UPDRS3.Rising.from.Chair', 'NP3GAIT' : 'Gait.Problems' ,'NP3FRZGT' :
↪  'Freezing.of.Gait' ,'NP3PSTBL' : 'Postural.Stability.Problems', 'NP3POSTR' :
↪  'Posture.Problems' , 'NP3TOT':'UPDRS.Part3.Total', 'RMONOFF' :
↪  'MDS-UPDRS.Part3.Remote.Visit.ON.or.OFF',
↪  'Most.Recent.PD.Med.Dose.Date.Time.Before.OFF.Exam' :
↪  'Most.Recent.PD.Med.Dose.Date.Time.Before.OFF.Exam', 'RMTOFFRSN' :
↪  'Reason.OFF.Exam.at.Remote.Visit',  'ONEXAM' : 'ON.Exam.Performed' ,'ONEXAMTM' :
↪  'ON.Exam.Time', 'ONNORSN' : 'Reason.ON.Exam.Not.Performed' , 'ONOFFORDER' :
↪  'ON.or.OFF.Exam.Performed.First' , 'Most.Recent.PD.Med.Dose.Date.Time.Before.ON.Exam'
↪  :'Most.Recent.PD.Med.Dose.Date.Time.Before.ON.Exam', 'PDMEDYN' : 'On.PD.Medication',
↪  'DBSONTM' : 'Time.DBS.Turned.on.before.ON.Exam', 'DBSOFFTM' :
↪  'Time.DBS.Turned.off.before.OFF.Exam', 'OFFNORSN' : 'Reason.OFF.Exam.Not.Performed' ,
↪  'OFFEXAM' : 'OFF.Exam.Performed', 'OFFEXAMTM' : 'OFF.Exam.Time', 'DBSYN' : 'Has.DBS',
↪  'HRPOSTMED' : 'Hours.btwn.PD.Med.and.UPDRS3.Exam', 'HRDBSOFF' :
↪  'Hours.btwn.DBS.Device.Off.and.UPDRS3.Exam', 'HRDBSON' :
↪  'Hours.btwn.DBS.Device.On.and.UPDRS3.Exam' , 'RMEXAM' :
↪  'Remote.UPDRS3.Exam','DYSKPRES' : 'Dyskinesias.Present', 'DYSKIRAT' :
↪  'Movements.Interefered.with.Ratings', 'NHY' : 'Hoehn.and.Yahr.Stage', 'PDTRTMNT' :
↪  'On.PD.Treatment','NP3BRADY' : 'Global.Spontaneity.of.Movement', 'NP3PTRMR' :
↪  'Postural.Tremor.Right.Hand' , 'NP3PTRML' : 'Postural.Tremor.Left.Hand' , 'NP3KTRMR'
↪  : 'Kinetic.Tremor.Right.Hand', 'NP3KTRML' : 'Kinetic.Tremor.Left.Hand', 'NP3RTARU' :
↪  'Rest.Tremor.Amplitude.RUE', 'NP3RTALU' : 'Rest.Tremor.Amplitude.LUE', 'NP3RTARL' :
↪  'Rest.Tremor.Amplitude.RLE' ,'NP3RTALL' : 'Rest.Tremor.Amplitude.LLE' ,'NP3RTALJ' :
↪  'Rest.Tremor.Amplitude.Lip.Jaw', 'NP3RTCON' : 'Constancy.of.Rest.Tremor', 'PDSTATE' :
↪  'Functional.State','EXAMTM' : 'UPDRS.Part3.Exam.Time' }, inplace = True)
    df = add_extension_to_column_names(df, ['PATNO','EVENT_ID','INFODT'],
↪  ext_list[count])
    count += 1

# Combine pd med dose date and time into one column
```

```
nupdrs3_copy = merge_columns(nupdrs3_copy, ['PDMEDDT.UPDRS3', 'PDMEDTM.UPDRS3'],
↪  'Most.Recent.PD.Med.Dose.Date.Time.UPDRS3',' ')
nupdrs3A_copy = merge_columns(nupdrs3A_copy, ['PDMEDDT.UPDRS3A', 'PDMEDTM.UPDRS3A'],
↪  'Most.Recent.PD.Med.Dose.Date.Time.UPDRS3A', ' ')
nupdr3ON_copy = merge_columns(nupdr3ON_copy, ['PDMEDDT.UPDR3ON', 'PDMEDTM.UPDR3ON'],
↪  'Most.Recent.PD.Med.Dose.Date.Time.UPDR3ON', ' ')
nupdr3OF_copy = merge_columns(nupdr3OF_copy, ['PDMEDDT.UPDR3OF', 'PDMEDTM.UPDR3OF'],
↪  'Most.Recent.PD.Med.Dose.Date.Time.UPDR3OF', ' ')

# Merge NUPDRS3, NUPDRS3A, NUPDR3ON, NUPDR3OF
updrs3_merge = pd.merge(nupdrs3_copy, nupdrs3A_copy, on = ['PATNO', 'EVENT_ID'], how =
↪  "outer")
updrs3_merge = pd.merge(updrs3_merge, nupdr3OF_copy,  on = ['PATNO', 'EVENT_ID'], how =
↪  "outer")
updrs3_merge = pd.merge(updrs3_merge, nupdr3ON_copy,  on = ['PATNO', 'EVENT_ID'], how =
↪  "outer")

# Rename page name variables
updrs3_merge['UPDRS.Part3.Page.Name.UPDRS3'].replace({'NUPDRS3' : 'MDS-UPDRS Part III (No
↪  Treatment)'}, inplace = True)
updrs3_merge['UPDRS.Part3.Page.Name.UPDRS3A'].replace({'NUPDRS3A' : 'MDS-UPDRS Part III
↪  (Post Treatment)'}, inplace = True)
updrs3_merge['UPDRS.Part3.Page.Name.UPDR3OF'].replace({'NUPDR3OF' : 'MDS-UPDRS Part III
↪  (OFF State)'}, inplace = True)
updrs3_merge['UPDRS.Part3.Page.Name.UPDR3ON'].replace({'NUPDR3ON' : 'MDS-UPDRS Part III
↪  (ON State)'}, inplace = True)

# Merge all UPDRS dfs together
updrs_cat = pd.merge(updrs_part1_df_copy, updrs_part1_pq_df_copy, on = ['PATNO',
↪  'EVENT_ID'], how = "outer")
updrs_cat = pd.merge(updrs_cat, updrs_part2_pq_df_copy, on = ['PATNO', 'EVENT_ID'], how =
↪  "outer")
updrs_cat = pd.merge(updrs_cat, updrs3_merge, on = ['PATNO', 'EVENT_ID'], how = "outer")
updrs_cat = pd.merge(updrs_cat, updrs_part3_dos_df_copy, on = ['PATNO', 'EVENT_ID'], how
↪  = "outer")
updrs_cat = pd.merge(updrs_cat, updrs_part4_motor_df_copy, on = ['PATNO', 'EVENT_ID'],
↪  how = "outer")

## Decode UPDRS df variables
# MNDS_UPDRS Part 1
updrs_cat['UPDRS.Part1.Source'].replace({1 : 'Patient' , 2 : 'Caregiver' , 3 : 'Patient
↪  and Caregiver'}, inplace = True)
updrs_cat = decode_none2severe(updrs_cat, ['NP1COG', 'NP1HALL', 'NP1DPRS', 'NP1ANXS',
↪  'NP1APAT', 'NP1DDS'])
updrs_cat.rename(columns = {'UPDRS.Part1.Source' : 'UPDRS.Part1.Source.UPDRS1', 'NP1COG'
↪  : 'Cognitive.Impairment.UPDRS1', 'NP1HALL' : 'Hallucinations.and.Psychosis.UPDRS1',
↪  'NP1DPRS' : 'Depressed.Moods.UPDRS1', 'NP1ANXS' : 'Anxious.Moods.UPDRS1', 'NP1APAT' :
↪  'Apathy.UPDRS1', 'NP1DDS' : 'Features.of.Dopamine.Dysregulation.Syndrome.UPDRS1',
↪  'NP1RTOT' : 'UPDRS.Part1.Rater.Completed.Total.UPDRS1'}, inplace = True)

# MDS_UPDRS_Part 1 Patient Questionnaire
updrs_cat['UPDRS.Part1.PQ.Source'].replace({1 : 'Patient', 2 : 'Caregiver', 3: 'Patient
↪  and Caregiver'}, inplace = True)
```

```
updsr_cat = decode_none2severe(updrs_cat, ['NP1SLPN', 'NP1SLPD', 'NP1PAIN', 'NP1URIN',
↪  'NP1CNST', 'NP1LTHD', 'NP1FATG'])
updrs_cat.rename(columns = {'UPDRS.Part1.PQ.Source' :
↪  'UPDRS.Part1.Patient.Questionnaire.Source.UPDRS1','NP1SLPN' :
↪  'Sleep.Problems.Night.UPDRS1', 'NP1SLPD' : 'Daytime.Sleepiness.UPDRS1', 'NP1PAIN' :
↪  'Pain.UPDRS1' , 'NP1URIN' : 'Urinary.Problems.UPDRS1', 'NP1CNST' :
↪  'Constipation.Problems.UPDRS1' , 'NP1LTHD' : 'Lightheadedness.on.Standing.UPDRS1' ,
↪  'NP1FATG' : 'Fatigue.UPDRS1', 'NP1PTOT' :
↪  'UPDRS.Part1.Patient.Completed.Total.UPDRS1'}, inplace = True)

# MDS_UPDRS Part 2
updrs_cat['UPDRS.Part2.Source'].replace({1 : 'Patient', 2 : 'Caregiver', 3 : 'Patient and
↪  Caregiver'}, inplace = True)
updrs_cat =
↪  decode_none2severe(updrs_cat,['NP2SPCH','NP2SALV','NP2SWAL','NP2EAT','NP2DRES','NP2HYGN','NP2HWRT',
updrs_cat.rename(columns = {'UPDRS.Part2.Source' : 'UPDRS.Part2.Source.UPDRS2', 'NP2SPCH'
↪  : 'UPDRS2.Speech.Difficulty.UPDRS2' , 'NP2SALV' : 'Saliva.Drooling.UPDRS2'
↪  ,'NP2SWAL': 'Chewing.Swallowing.Difficulty.UPDRS2', 'NP2EAT' :
↪  'Eating.Difficulty.UPDRS2', 'NP2DRES' : 'Dressing.Difficulty.UPDRS2', 'NP2HYGN' :
↪  'Hygiene.Difficulty.UPDRS2' , 'NP2HWRT' : 'Handwriting.Difficulty.UPDRS2' ,'NP2HOBB'
↪  : 'Hobbies.Difficulty.UPDRS2' ,'NP2TURN' : 'Turning.in.Bed.Difficulty.UPDRS2',
↪  'NP2TRMR' : 'Tremor.UPDRS2', 'NP2RISE' :
↪  'UPDRS2.Rising.from.Chair.Difficulty.UPDRS2', 'NP2WALK' : 'Walking.Difficulty.UPDRS2'
↪  ,'NP2FREZ' : 'Freezing.while.Walking.UPDRS2' , 'NP2PTOT':
↪  'UPDRS.Part2.Total.UPDRS2'}, inplace = True)

# MDS_UPDRS Part 3 On OFF determination Dosing
updrs_cat = decode_0_1_no_yes(updrs_cat, ['RMEXAM','DBSYN', 'OFFEXAM', 'OFFEXAM',
↪  'ONEXAM'])
updrs_cat['RMTOFFRSN'].replace({1 : 'ON state not reached', 2 : 'Scheduling issues', 3 :
↪  'Other reason'}, inplace = True)
updrs_cat['ONOFFORDER'].replace({1 : 'OFF', 2 : 'ON'}, inplace = True)
updrs_cat['RMONOFF'].replace({1 : 'OFF', 2 : 'ON'}, inplace = True)
updrs_cat['ONNORSN'].replace({1 : 'ON state not reached', 2 : 'Scheduling issues', 3 :
↪  'Other reason'}, inplace = True)
updrs_cat['PDMEDYN' ].replace({0 : False , 1 : True}, inplace = True)
updrs_cat = merge_columns(updrs_cat, ['ONPDMEDDT','ONPDMEDTM'],
↪  'Most.Recent.PD.Med.Dose.Date.Time.Before.ON.Exam', ' ')
updrs_cat['OFFNORSN'].replace({1 : 'Disease severity preventing turning off of DBS', 2 :
↪  'Did not bring medication to turn on' , 3 : 'Forgot to refrain from taking
↪  medication', 4 : 'Does not want to turn off DBS', 5 : 'Forgot to turn off DBS', 6 :
↪  'Forgot to bring DBS', 7 : 'Unsure if participant was full off', 8 : 'Site scheduling
↪  issues', 9 : 'Other reason'}, inplace = True)
updrs_cat = merge_columns(updrs_cat, ['OFFPDMEDDT','OFFPDMEDTM'],
↪  'Most.Recent.PD.Med.Dose.Date.Time.Before.OFF.Exam', ' ')
updrs_cat.rename(columns = {'RMONOFF' :
↪  'MDS-UPDRS.Part3.Remote.Visit.ON.or.OFF.UPDRDOSE',
↪  'Most.Recent.PD.Med.Dose.Date.Time.Before.OFF.Exam' :
↪  'Most.Recent.PD.Med.Dose.Date.Time.Before.OFF.Exam.UPDRDOSE', 'RMTOFFRSN' :
↪  'Reason.OFF.Exam.at.Remote.Visit.UPDRDOSE',  'ONEXAM' : 'ON.Exam.Performed.UPDRDOSE'
↪  ,'ONEXAMTM' : 'ON.Exam.Time.UPDRDOSE', 'ONNORSN' :
↪  'Reason.ON.Exam.Not.Performed.UPDRDOSE' , 'ONOFFORDER' :
↪  'ON.or.OFF.Exam.Performed.First.UPDRDOSE' ,
↪  'Most.Recent.PD.Med.Dose.Date.Time.Before.ON.Exam'
↪  :'Most.Recent.PD.Med.Dose.Date.Time.Before.ON.Exam.UPDRDOSE', 'PDMEDYN' :
↪  'On.PD.Medication.UPDRDOSE', 'DBSONTM' :
↪  'UPDRS3.Time.DBS.Turned.on.before.ON.Exam.UPDRDOSE', 'DBSOFFTM' :
↪  'UPDRS3.Time.DBS.Turned.off.before.OFF.Exam.UPDRDOSE','DBSONTM_y' :
↪  'UPDRS3.Dos.Time.DBS.Turned.on.before.ON.Exam.UPDRDOSE', 'DBSOFFTM_y' :
↪  'UPDRS3.Dos.Time.DBS.Turned.off.before.OFF.Exam.UPDRDOSE', 'OFFNORSN' :
```

```
# MDS_UPDRS Part 4
updrs_cat['NP4WDYSK'].replace({0 : 'No dyskinesias', 1: 'Slight: <= 25% of waking day',
↪  2: 'Mild : 26-50% of waking day', 3: 'Moderate: 51-75% of waking day', 4 : 'Severe: >
↪  75% of waking day'}, inplace = True)
updrs_cat['NP4DYSKI'].replace({0 : 'No dyskinesias', 1 : 'Slight', 2: 'Mild', 3 :
↪  'Moderate', 4: 'Severe'}, inplace = True)
updrs_cat['NP4OFF'].replace({0 : 'Normal: No OFF time', 1 : 'Slight: <= 25% of waking day
↪  ', 2: 'Mild : 26-50% of waking day', 3 : 'Moderate: 51-75% of waking day', 4 :
↪  'Severe: > 75% of waking day'}, inplace = True)
updrs_cat['NP4FLCTI'].replace({0 : 'Normal', 1: 'Slight', 2: 'Mild', 3: 'Moderate', 4:
↪  'Severe'}, inplace = True)
updrs_cat['NP4DYSTN'].replace({0 : 'No dystonia', 1: 'Slight: <= 25% of time in OFF
↪  state', 2: 'Mild : 26-50% of time in OFF state', 3 : 'Moderate: 51-75% of time in OFF
↪  state', 4: 'Severe: > 75% of time in OFF state'}, inplace = True)
updrs_cat['NP4FLCTX'].replace({0: 'Normal', 1 : 'Slight', 2: 'Mild', 3 : 'Moderate', 4 :
↪  'Severe'}, inplace = True)
updrs_cat['RMNOPRT3'].replace({1 : 'Visit was not conducted with video', 2 : 'Scheduling
↪  issues', 3: 'Other reason'}, inplace = True)
updrs_cat.rename(columns = {'RMNOPRT3':
↪  'Reason.UPDRSPart3.Not.Administered.Remote.Visit.UPDRDOSE','NP4WDYSKDEN':'4.1.Time.with.Dyskinesias
↪  'NP4WDYSKNUM' :'4.1.Total.Hours.Awake.UPDRS4', 'NP4WDYSKPCT' :
↪  'Percent.Dyskinesia.UPDRS4','NP4OFFDEN':'4.3.Total.Hours.OFF.UPDRS4', 'NP4OFFNUM' :
↪  '4.3.Total.Hours.Awake.UPDRS4','NP4OFFPCT' : 'Percent.OFF.UPDRS4', 'NP4DYSTNDEN'
↪  :'4.6.Total.Hours.OFF.with.Dystonia.UPDRS4',
↪  'NP4DYSTNNUM':'4.6.Total.Hours.OFF.UPDRS4',
↪  'NP4DYSTNPCT':'4.6.Percent.OFF.Dystonia.UPDRS4', 'NP4WDYSK' :
↪  'Time.Spent.with.Dyskinesias.UPDRS4',
↪  'NP4DYSKI':'Functional.Impact.of.Dyskinesias.UPDRS4','NP4OFF' :
↪  'Time.Spent.in.OFF.State.UPDRS4',
↪  'NP4FLCTI':'Functional.Impact.Fluctuations.UPDRS4',
↪  'NP4FLCTX':'Complexity.of.Motor.Fluctuations.UPDRS4' ,
↪  'NP4DYSTN':'Painful.OFF-state.Dystonia.UPDRS4' ,'NP4TOT':
↪  'UPDRS.Part4.Total.UPDRS4','NP4WDYSKDEN' : '4.1.Total.Hours.with.Dyskinesia',
↪  'NP4WDYSKNUM' : '4.1.Total.Hours.Awake', 'NP4WDYSKPCT' : '4.1.%.Dyskinesia',
↪  'NP4OFFDEN' :'4.3.Total.Hours.OFF' , 'NP4OFFNUM' :'4.3.Total.Hours.Awake' ,
↪  'NP4OFFPCT': '4.3.%.OFF', 'NP4DYSTNDEN' : '4.6.Total.Hours.OFF.with.Dystonia',
↪  'NP4DYSTNNUM' :'4.6.Total.Hours.OFF', 'NP4DYSTNPCT' : '4.6.%.OFF.Dystonia'
↪  ,'NP4DYSTNPCT' :'4.6.%.OFF.Dystonia'}, inplace = True)
updrs_cat = add_extension_to_column_names(updrs_cat, ['PATNO', 'EVENT_ID', 'INFODT'],
↪  '.Cat') # Add a .Cat extension to column names w updrs cat vars

# Create one df with UPDRS scores .Num (numeric) and all UPDRS scores .Cat (categorical)
updrs_cat.drop(['INFODT'], axis = 1, inplace = True)
updrs_merged = pd.merge(updrs_cat, updrs_numeric, on = ['PATNO', 'EVENT_ID'], how =
↪  "outer")
updrs_merged.replace({'UR' : np.nan}, inplace = True) # Unable to Rate --> nan
ppmi_merge = pd.merge(ppmi_merge, updrs_merged, on = ['PATNO', 'EVENT_ID'], how =
↪  "outer")

#### CLEAN UP DF ####
ppmi_merge.drop(['Subid.Date.TEMP'], axis = 1, inplace = True)
ppmi_merge.rename(columns = {'EVENT_ID': 'Event.ID', 'INFODT' :
↪  'Event.ID.Date','Medication' : 'Medication(Dates)' , 'DOMSIDE' :
↪  'Dominant.Side.Disease'}, inplace = True)
```

```
# Change names of event ids to be indicative of months
ppmi_merge['Event.ID'].replace({'FNL' : 'Final Visit', 'BL' : 'Baseline', 'SC' :
↪  'Screening', 'LOG' : 'Logs', 'PW' : 'Premature Withdrawal', 'R04' : 'Remote Visit
↪  Month 18', 'R06' : 'Remote Visit Month 18', 'R08' : 'Remote Visit Month 42', 'R10' :
↪  'Remote Visit Month 54', 'R12' : 'Remote Visit Month 66', 'R13' : 'Remote Visit Month
↪  78' , 'R14' : 'Remote Visit Month 90' , 'R15' : 'Remote Visit Month 102' , 'R16' :
↪  'Remote Visit Month 114', 'R17' : 'Remote Visit Month 126', 'RS1' : 'Re-screen', 'ST'
↪  : 'Symptomatic Therapy', 'U01' : 'Unscheduled Visit 1', 'U02' : 'Unscheduled Visit
↪  2', 'V01' : 'Visit Month 3', 'V02' : 'Visit Month 6', 'V03' : 'Visit Month 9', 'V04'
↪  : 'Visit Month 12', 'V05' : 'Visit Month 18', 'V06' : 'Visit Month 24', 'V07' :
↪  'Visit Month 30', 'V08' : 'Visit Month 36', 'V09' : 'Visit Month 42', 'V10' : 'Visit
↪  Month 48', 'V11' : 'Visit Month 54','V12' : 'Visit Month 60', 'V13' : 'Visit Month
↪  72', 'V14' : 'Visit Month 84', 'V15' : 'Visit Month 96', 'V16' : 'Visit Month 108',
↪  'V17' : 'Visit Month 120', 'V18' : 'Visit Month 132', 'P78' : 'Phone Visit (Month
↪  78)'}, inplace = True)

race_list = ['African.Berber.Race','Ashkenazi.Jewish.Race',
↪  'Basque.Race','Hispanic.Latino.Race', 'Asian.Race', 'African.American.Race',
↪  'Hawian.Other.Pacific.Islander.Race', 'Indian.Alaska.Native.Race',
↪  'Not.Specified.Race',  'White.Race']
for col_name in race_list :
    ppmi_merge[col_name].replace({0 : 'No', 1 : 'Yes', 2 : np.NaN}, inplace = True)
ppmi_merge.to_csv('/Users/areardon/Desktop/ppmi_mergex.csv')

# Fill in cells that are NA with subject information that we know from other event
↪  ids/rows for fixed variables
fixed_var_list = ['Enroll.Diagnosis', 'Enroll.Subtype','Consensus.Diagnosis',
↪  'Consensus.Subtype', 'Subject.Phenoconverted', 'BirthDate', 'Sex', 'Handed',
↪  'Analytic.Cohort','African.Berber.Race','Ashkenazi.Jewish.Race', 'Basque.Race',
↪  'Hispanic.Latino.Race', 'Asian.Race', 'African.American.Race',
↪  'Hawian.Other.Pacific.Islander.Race', 'Indian.Alaska.Native.Race',
↪  'Not.Specified.Race',  'White.Race'] # fixed variables
for col_name in fixed_var_list :
    ppmi_merge[col_name].fillna('NA', inplace = True)
    for row_num in range(len(ppmi_merge[col_name])):
        if ppmi_merge[col_name].iloc[row_num] == 'NA' : # if any entry is NA
            current_sub = ppmi_merge['PATNO'].iloc[row_num] # get current subid
            fixed_var_value = ppmi_merge.loc[(ppmi_merge['PATNO'] == current_sub ) &
↪  (ppmi_merge[col_name] != 'NA'), col_name].values # get value from another event id
            if fixed_var_value.any() :
                ppmi_merge.loc[row_num, col_name] = fixed_var_value[0] # fill in baseline
↪  value at NA
ppmi_merge = ppmi_merge.rename(columns = {'PATNO' : 'Subject.ID'})
ppmi_merge.replace({'NA' : np.nan}, inplace = True)

race_list = ['African.Berber.Race','Ashkenazi.Jewish.Race',
↪  'Basque.Race','Hispanic.Latino.Race', 'Asian.Race', 'African.American.Race',
↪  'Hawian.Other.Pacific.Islander.Race', 'Indian.Alaska.Native.Race',
↪  'Not.Specified.Race',  'White.Race']
for col_name in race_list :
    ppmi_merge[col_name].replace({'No' :False , 'Yes' : True, 2 : np.NaN}, inplace =
↪  True)
```

```
#### GENETICS INFO ####
lrrk2_genetics_df = pd.read_csv(genetics_path + 'lrrk2_geno_012_mac5_missing_geno.csv')
scna_genetics_df = pd.read_csv(genetics_path + 'scna_geno_012_mac5_missing_geno.csv')
apoe_genetics_df = pd.read_csv(genetics_path + 'apoe_geno_012_mac5_missing_geno.csv')
tmem_genetics_df = pd.read_csv(genetics_path + 'tmem175_geno_012_mac5_missing_geno.csv')
gba_genetics_df = pd.read_csv(genetics_path + 'gba_geno_012_mac5_missing_geno.csv')

def format_genetics_df(genetics_df : pd.DataFrame ) :
    """
    Format genetics_df to make merge-able with ppmi_merge
    """
    genetics_df.drop(['COUNTED', 'ALT', 'SNP', '(C)M'], axis = 1, inplace = True) #
↪   Remove unnecessary columns

    # Change column names to be just subid
    for col in genetics_df:
        if '_' in col :
            subid = int(col.split('_')[-1])
            genetics_df.rename(columns = {col : subid}, inplace = True)

    # Combine CHR and POS columns
    genetics_df['CHR'] = 'CHR' + genetics_df['CHR'].astype(str) # Need to be strings
↪   before you use merge_columns function
    genetics_df['POS'] = 'POS' + genetics_df['POS'].astype(str) # Need to be strings
↪   before you use merge_columns function
    genetics_df = merge_columns(genetics_df, ['CHR', 'POS'], 'Chromosome.Position', '.')

    # Pivot df so position is column name and subid is row
    genetics_df = genetics_df.T # Transpose df so that rows are subid
    genetics_df.rename(columns = genetics_df.iloc[-1], inplace = True) # Move Chr.Pos to
↪   column names
    genetics_df.index.names = ['Subject.ID'] # Rename index to 'Subject.ID'
    genetics_df = genetics_df.drop(['Chromosome.Position'], axis = 0) # Drop last row
↪   (repeat of col names)
    genetics_df = genetics_df.reset_index(drop = False)

    return genetics_df

lrrk2_genetics_df_formatted = format_genetics_df(lrrk2_genetics_df)
scna_genetics_df_formatted = format_genetics_df(scna_genetics_df)
apoe_genetics_df_formatted = format_genetics_df(apoe_genetics_df)
tmem_genetics_df_formatted = format_genetics_df(tmem_genetics_df)
gba_genetics_df_formatted = format_genetics_df(gba_genetics_df)

# Merge genetics dataframes together to create one genetics_df
genetics_df = pd.merge(lrrk2_genetics_df_formatted, scna_genetics_df_formatted, on =
↪   ['Subject.ID'], how = "outer")
genetics_df = pd.merge(genetics_df, apoe_genetics_df_formatted, on = ['Subject.ID'], how
↪   = "outer")
genetics_df = pd.merge(genetics_df, tmem_genetics_df_formatted, on = ['Subject.ID'], how
↪   = "outer")
```

```
genetics_df = pd.merge(genetics_df, gba_genetics_df_formatted, on = ['Subject.ID'], how =
↪  "outer")

# Change genetics col names int to float (remove .0 in all 'CHR.POS' columns)
for col_name in genetics_df :
    if col_name.startswith('CHR'):
        genetics_df[col_name] = genetics_df[col_name].fillna(-9999.0)
        genetics_df[col_name] = genetics_df[col_name].astype(int)
genetics_df.replace({-9999.0 : 'NA'}, inplace = True)

# Merge ppmi_merge with genetics df
ppmi_merge_genetics = pd.merge(ppmi_merge, genetics_df, on = 'Subject.ID', how = "outer")

#### T1 Info - Taylor's File ####
ppmi_t1_df = pd.read_csv(invicro_data_path + 'ppmi_mergewide_t1.csv') # Read in Taylor's
↪  T1 results file
ppmi_t1_df.rename(columns = {'u_hier_id_OR': 'Subject.ID'}, inplace = True) # Rename
↪  subject id column in Taylors df to match ppmi_merge

# Create a column for object name to merge on with ppmi_merge
ppmi_t1_df['Image_ID_merge'] = ''
for row_num in range(len(ppmi_t1_df['ImageID'])) :
    image_id = ppmi_t1_df['ImageID'].iloc[row_num].split('-')[2]
    ppmi_t1_df['Image_ID_merge'].iloc[row_num] = image_id

# Merge ppmi_merge_genetics  with t1 info
ppmi_merge = pd.merge(ppmi_merge_genetics, ppmi_t1_df, on =
↪  ['Subject.ID','Image_ID_merge'], how = "left") # Merge
ppmi_merge.drop(['Image_ID_merge'], axis = 1, inplace = True) # Drop

# Put full date in Image.Acquisition.Date column
for row_num in range(len(ppmi_merge['T1.s3.Image.Name'])) :
    if isinstance(ppmi_merge['T1.s3.Image.Name'].iloc[row_num],str) :
        date = ppmi_merge['T1.s3.Image.Name'].iloc[row_num].split('-')[2]
        ppmi_merge['Image.Acquisition.Date'].iloc[row_num] = date[4:6] + '/' + date[6:8]
↪  +'/' + date[0:4]

## Get Enrollment Diagnosis for subjects in Not Analytic Cohort - do this using the
↪  participants_status.csv
analytic = ppmi_merge[ppmi_merge['Analytic.Cohort'] == 'Analytic Cohort'] # Split up
↪  Analytic cohort df and not Analytic Cohort df
not_analytic = ppmi_merge[ppmi_merge['Analytic.Cohort'] == 'Not Analytic Cohort'] # Split
↪  up Analytic cohort df and not Analytic Cohort df
participant_status = pd.read_csv(ppmi_download_path + 'Participant_Status.csv') # Read in
↪  participant_status.csv
participant_status = participant_status[['PATNO', 'COHORT_DEFINITION']] # Keep only
participant_status.rename(columns = {'PATNO' : 'Subject.ID', 'COHORT_DEFINITION' :
↪  'Enroll.Diagnosis'}, inplace = True)
not_analytic_participant_status = pd.merge(not_analytic, participant_status, on =
↪  ['Subject.ID'], how = "left") # Merge not Atlantic subids with enrollment diagnosis
↪  in participant_status
not_analytic_participant_status.drop(['Enroll.Diagnosis_x'], axis = 1, inplace = True) #
↪  Remove the extra Enroll.Diagnosis created at merge
```

```
not_analytic_participant_status.rename(columns = {'Enroll.Diagnosis_y' :
↪  'Enroll.Diagnosis'}, inplace = True)
not_analytic_participant_status.loc[not_analytic_participant_status['Enroll.Diagnosis']
↪  == 'Healthy Control', 'Enroll.Subtype'] = 'Healthy Control' # For Healthy Control
↪  subjects in the Not Analytic Cohort - make 'Enroll.Subtype' = Healthy Control

# Merge df of Not Analytic and Analytic subjects and sort by SubID and Event.ID.Date
ppmi_merge = pd.concat([analytic, not_analytic_participant_status])

# Change Event.ID.Date to date time and corrected format
ppmi_merge['Event.ID.Date'] = ppmi_merge['Event.ID.Date'].astype(str)
ppmi_merge['Event.ID.Date'] = pd.to_datetime(ppmi_merge['Event.ID.Date'], errors =
↪  "ignore") # Change event.ID.Date column to date time so we can sort according to this
↪
ppmi_merge = ppmi_merge.sort_values(by = ['Subject.ID','Event.ID.Date']) # Sort values by
↪  subject and event id date
ppmi_merge['Event.ID.Date'] = ppmi_merge['Event.ID.Date'].astype(str) # Change
↪  Event.ID.Date back to string so we can reformat

# Reformat Event.ID.Date from pd.to_datetime to month/year
for row_num in range(len(ppmi_merge['Event.ID.Date'])):
    if ppmi_merge['Event.ID.Date'].iloc[row_num] != 'NaT':
        split = ppmi_merge['Event.ID.Date'].iloc[row_num].split('-')
        new_date = split[1] +'/' + split[0] # month/year format
        ppmi_merge['Event.ID.Date'].iloc[row_num] = new_date
ppmi_merge['Event.ID.Date'] = ppmi_merge['Event.ID.Date'].replace('NaT','NA')

#### Add in ppmi_qc_BA.csv - Brian sent on slack 1/31/22 ####
ppmi_qc_BA = pd.read_csv(invicro_data_path + 'ppmi_qc_BA.csv')
ppmi_qc_BA = ppmi_qc_BA.reset_index(drop = False) # Move index to first column so we can
↪  rename
ppmi_qc_BA.rename(columns = {'ID' : 'ImageID'}, inplace = True) # Rename ImageID

# Update ImageID column bc info from T1 file (where ImageID was created from) does not
↪  contain all the files from s3 (need this to merge in subs from QC csv)
for row_num in range(len(ppmi_merge['ImageID'])) :
    if isinstance(ppmi_merge['T1.s3.Image.Name'].iloc[row_num], str):
        imageID = ppmi_merge['T1.s3.Image.Name'].iloc[row_num].split('.')[0] # take info
↪  before .nii.gz
        ppmi_merge['ImageID'].iloc[row_num] = imageID.split('-')[1] + '-' +
↪  imageID.split('-')[2] + '-' + imageID.split('-')[4]
ppmi_merge = pd.merge(ppmi_merge, ppmi_qc_BA, on = ['ImageID'], how = "left") # Merge -
↪  keep only from ImageIDs we already have


#### BILATERAL SUBTYPE SCORES (Tremor and Brady) ####
brady_right3 = ['Dominant.Side.Disease', 'Rigidity.Neck.UPDRS3.Num',
↪  'Rigidity.RUE.UPDRS3.Num', 'Rigidity.RLE.UPDRS3.Num',
↪  'Finger.Tapping.Right.Hand.UPDRS3.Num', 'Hand.Movements.Right.Hand.UPDRS3.Num',
↪  'Pronation.Supination.Right.Hand.UPDRS3.Num', 'Toe.Tapping.Right.Foot.UPDRS3.Num',
↪  'Leg.Agility.Right.Leg.UPDRS3.Num']
brady_right3a = ['Dominant.Side.Disease', 'Rigidity.Neck.UPDRS3A.Num',
↪  'Rigidity.RUE.UPDRS3A.Num', 'Rigidity.RLE.UPDRS3A.Num',
↪  'Finger.Tapping.Right.Hand.UPDRS3A.Num', 'Hand.Movements.Right.Hand.UPDRS3A.Num',
↪  'Pronation.Supination.Right.Hand.UPDRS3A.Num', 'Toe.Tapping.Right.Foot.UPDRS3A.Num',
↪  'Leg.Agility.Right.Leg.UPDRS3A.Num']
```

```
brady_right3ON = ['Dominant.Side.Disease', 'Rigidity.Neck.UPDR3ON.Num',
↪  'Rigidity.RUE.UPDR3ON.Num', 'Rigidity.RLE.UPDR3ON.Num',
↪  'Finger.Tapping.Right.Hand.UPDR3ON.Num', 'Hand.Movements.Right.Hand.UPDR3ON.Num',
↪  'Pronation.Supination.Right.Hand.UPDR3ON.Num', 'Toe.Tapping.Right.Foot.UPDR3ON.Num',
↪  'Leg.Agility.Right.Leg.UPDR3ON.Num']
brady_right3OF = ['Dominant.Side.Disease', 'Rigidity.Neck.UPDR3OF.Num',
↪  'Rigidity.RUE.UPDR3OF.Num', 'Rigidity.RLE.UPDR3OF.Num',
↪  'Finger.Tapping.Right.Hand.UPDR3OF.Num', 'Hand.Movements.Right.Hand.UPDR3OF.Num',
↪  'Pronation.Supination.Right.Hand.UPDR3OF.Num', 'Toe.Tapping.Right.Foot.UPDR3OF.Num',
↪  'Leg.Agility.Right.Leg.UPDR3OF.Num']

brady_left3 = ['Dominant.Side.Disease', 'Rigidity.Neck.UPDRS3.Num',
↪  'Rigidity.LUE.UPDRS3.Num', 'Rigidity.LLE.UPDRS3.Num',
↪  'Finger.Tapping.Left.Hand.UPDRS3.Num', 'Hand.Movements.Left.Hand.UPDRS3.Num',
↪  'Pronation.Supination.Left.Hand.UPDRS3.Num', 'Toe.Tapping.Left.Foot.UPDRS3.Num',
↪  'Leg.Agility.Left.Leg.UPDRS3.Num']
brady_left3a = ['Dominant.Side.Disease', 'Rigidity.Neck.UPDRS3A.Num',
↪  'Rigidity.LUE.UPDRS3A.Num', 'Rigidity.LLE.UPDRS3A.Num',
↪  'Finger.Tapping.Left.Hand.UPDRS3A.Num', 'Hand.Movements.Left.Hand.UPDRS3A.Num',
↪  'Pronation.Supination.Left.Hand.UPDRS3A.Num', 'Toe.Tapping.Left.Foot.UPDRS3A.Num',
↪  'Leg.Agility.Left.Leg.UPDRS3A.Num']
brady_left3ON = ['Dominant.Side.Disease', 'Rigidity.Neck.UPDR3ON.Num',
↪  'Rigidity.LUE.UPDR3ON.Num', 'Rigidity.LLE.UPDR3ON.Num',
↪  'Finger.Tapping.Left.Hand.UPDR3ON.Num', 'Hand.Movements.Left.Hand.UPDR3ON.Num',
↪  'Pronation.Supination.Left.Hand.UPDR3ON.Num', 'Toe.Tapping.Left.Foot.UPDR3ON.Num',
↪  'Leg.Agility.Left.Leg.UPDR3ON.Num']
brady_left3OF = ['Dominant.Side.Disease', 'Rigidity.Neck.UPDR3OF.Num',
↪  'Rigidity.LUE.UPDR3OF.Num', 'Rigidity.LLE.UPDR3OF.Num',
↪  'Finger.Tapping.Left.Hand.UPDR3OF.Num', 'Hand.Movements.Left.Hand.UPDR3OF.Num',
↪  'Pronation.Supination.Left.Hand.UPDR3OF.Num', 'Toe.Tapping.Left.Foot.UPDR3OF.Num',
↪  'Leg.Agility.Left.Leg.UPDR3OF.Num']

brady_sym3 = ['Dominant.Side.Disease', 'Rigidity.Neck.UPDRS3.Num',
↪  'Rigidity.RUE.UPDRS3.Num', 'Rigidity.LUE.UPDRS3.Num', 'Rigidity.RLE.UPDRS3.Num',
↪  'Rigidity.LLE.UPDRS3.Num', 'Finger.Tapping.Right.Hand.UPDRS3.Num',
↪  'Finger.Tapping.Left.Hand.UPDRS3.Num', 'Hand.Movements.Right.Hand.UPDRS3.Num',
↪  'Hand.Movements.Left.Hand.UPDRS3.Num', 'Pronation.Supination.Right.Hand.UPDRS3.Num',
↪  'Pronation.Supination.Left.Hand.UPDRS3.Num', 'Toe.Tapping.Right.Foot.UPDRS3.Num',
↪  'Toe.Tapping.Left.Foot.UPDRS3.Num', 'Leg.Agility.Right.Leg.UPDRS3.Num',
↪  'Leg.Agility.Left.Leg.UPDRS3.Num']
brady_sym3a = ['Dominant.Side.Disease', 'Rigidity.Neck.UPDRS3A.Num',
↪  'Rigidity.RUE.UPDRS3A.Num', 'Rigidity.LUE.UPDRS3A.Num', 'Rigidity.RLE.UPDRS3A.Num',
↪  'Rigidity.LLE.UPDRS3A.Num', 'Finger.Tapping.Right.Hand.UPDRS3A.Num',
↪  'Finger.Tapping.Left.Hand.UPDRS3A.Num', 'Hand.Movements.Right.Hand.UPDRS3A.Num',
↪  'Hand.Movements.Left.Hand.UPDRS3A.Num',
↪  'Pronation.Supination.Right.Hand.UPDRS3A.Num',
↪  'Pronation.Supination.Left.Hand.UPDRS3A.Num', 'Toe.Tapping.Right.Foot.UPDRS3A.Num',
↪  'Toe.Tapping.Left.Foot.UPDRS3A.Num', 'Leg.Agility.Right.Leg.UPDRS3A.Num',
↪  'Leg.Agility.Left.Leg.UPDRS3A.Num']
brady_sym3ON = ['Dominant.Side.Disease', 'Rigidity.Neck.UPDR3ON.Num',
↪  'Rigidity.RUE.UPDR3ON.Num', 'Rigidity.LUE.UPDR3ON.Num', 'Rigidity.RLE.UPDR3ON.Num',
↪  'Rigidity.LLE.UPDR3ON.Num', 'Finger.Tapping.Right.Hand.UPDR3ON.Num',
↪  'Finger.Tapping.Left.Hand.UPDR3ON.Num', 'Hand.Movements.Right.Hand.UPDR3ON.Num',
↪  'Hand.Movements.Left.Hand.UPDR3ON.Num',
↪  'Pronation.Supination.Right.Hand.UPDR3ON.Num',
↪  'Pronation.Supination.Left.Hand.UPDR3ON.Num', 'Toe.Tapping.Right.Foot.UPDR3ON.Num',
↪  'Toe.Tapping.Left.Foot.UPDR3ON.Num', 'Leg.Agility.Right.Leg.UPDR3ON.Num',
↪  'Leg.Agility.Left.Leg.UPDR3ON.Num']
```

```
brady_sym3OF = ['Dominant.Side.Disease', 'Rigidity.Neck.UPDR3OF.Num',
↪  'Rigidity.RUE.UPDR3OF.Num', 'Rigidity.LUE.UPDR3OF.Num', 'Rigidity.RLE.UPDR3OF.Num',
↪  'Rigidity.LLE.UPDR3OF.Num', 'Finger.Tapping.Right.Hand.UPDR3OF.Num',
↪  'Finger.Tapping.Left.Hand.UPDR3OF.Num', 'Hand.Movements.Right.Hand.UPDR3OF.Num',
↪  'Hand.Movements.Left.Hand.UPDR3OF.Num',
↪  'Pronation.Supination.Right.Hand.UPDR3OF.Num',
↪  'Pronation.Supination.Left.Hand.UPDR3OF.Num', 'Toe.Tapping.Right.Foot.UPDR3OF.Num',
↪  'Toe.Tapping.Left.Foot.UPDR3OF.Num', 'Leg.Agility.Right.Leg.UPDR3OF.Num',
↪  'Leg.Agility.Left.Leg.UPDR3OF.Num']

# Make any dominant side of disease that are NA into 'Symmetric'
domsideisna = ppmi_merge['Dominant.Side.Disease'].isna()
ppmi_merge['Dominant.Side.Disease'].loc[domsideisna] = 'Symmetric'

def add_lateralized_subscores(df : pd.DataFrame, subscore_side_list : list, side : str,
↪  new_col_name : str) :
    """
    Include lateralized subscroes (i.e. Brady Rigidity and Tremor subscores) into
↪  dataframe.

    Arguments
    -------------------
    df : pd.DataFrame containing scores that make up subscore

    subscore_side_list : list containing column names of the scores that make up the
↪  subscore

    side : 'Left' or 'Right'

    new_col_name : name of new column name with lateralized subscore

    """
    subscore_side = df[subscore_side_list]  # Get dataframe of only columns in brady_left
    df[new_col_name] = 0 # Initialize lateralized variable
    subscore_side.loc[subscore_side['Dominant.Side.Disease'] != side, :] = np.nan # Make
↪  all rows nan if dominant side of disease is not left
    subscore_side_temp = subscore_side.drop('Dominant.Side.Disease',1) # Drop dominant
↪  side of disease - necessary because this cannot be summed in next line
    idx  = subscore_side_temp.loc[pd.isnull(subscore_side_temp).any(1), :].index
    df[new_col_name] = subscore_side_temp.sum(axis = 1) # Sum of all columns in each row
↪  where dom side is left
    df[new_col_name].iloc[idx] = np.nan # Fill in subscores that should be nans as nan

    return df


def add_symmetric_subscores(df : pd.DataFrame, subscore_side_list : list, side : str,
↪  new_col_name : str, ext : str) :
    subscore_side = df[subscore_side_list]  # Get dataframe of only columns in brady_left
    df[new_col_name] = 0 # Initialize lateralized variable
    subscore_side.loc[subscore_side['Dominant.Side.Disease'] != side, :] = np.nan # Make
↪  all rows nan if dominant side of disease is not side
    subscore_side_temp = subscore_side.drop('Dominant.Side.Disease',1) # Drop dominant
↪  side of disease - necessary because this cannot be summed in next line
```

```
    idx  = subscore_side_temp.loc[pd.isnull(subscore_side_temp).any(1), :].index
    x = subscore_side_temp.fillna(0) # because adding 1 plus nan equals nan
    if "Brady" in new_col_name :
        df[new_col_name] = x['Rigidity.Neck' + ext ] + (x['Rigidity.RUE' + ext] +
↪  x['Rigidity.LUE'+ ext ])/2 + (x['Rigidity.RLE'+ ext] + x['Rigidity.LLE'+ ext])/2  +
↪  (x['Finger.Tapping.Right.Hand'+ ext] +x['Finger.Tapping.Left.Hand'+ ext])/2 +
↪  (x['Hand.Movements.Right.Hand'+ ext] + x['Hand.Movements.Left.Hand'+ ext])/2 +
↪  (x['Pronation.Supination.Right.Hand'+ ext]+x['Pronation.Supination.Left.Hand'+
↪  ext])/2 +  (x['Toe.Tapping.Right.Foot'+ ext]+ x['Toe.Tapping.Left.Foot'+ ext])/2 +
↪  (x['Leg.Agility.Right.Leg'+ ext] + x['Leg.Agility.Left.Leg'+ ext])/2
        df[new_col_name].iloc[idx] = np.nan # Fill in subscores that should be nans as
↪  nan
    elif "Tremor" in new_col_name :
        df[new_col_name] = x['Tremor.UPDRS2.Num'] +  (x['Postural.Tremor.Right.Hand' +
↪  ext ]  + x['Postural.Tremor.Left.Hand' + ext ])/2 + (x['Kinetic.Tremor.Right.Hand' +
↪  ext ] + x['Kinetic.Tremor.Left.Hand'+ ext ])/2 + (x['Rest.Tremor.Amplitude.RUE' + ext
↪  ]+ x['Rest.Tremor.Amplitude.LUE' + ext])/2 + (x['Rest.Tremor.Amplitude.RLE' + ext ] +
↪  x['Rest.Tremor.Amplitude.LLE' + ext])/2 + x['Rest.Tremor.Amplitude.Lip.Jaw' + ext] +
↪  x['Constancy.of.Rest.Tremor' + ext]
        df[new_col_name].iloc[idx] = np.nan
    return df

ppmi_merge = add_lateralized_subscores(ppmi_merge, brady_left3, 'Left',
↪  'Brady.Rigidity.Subscore-left.UPDRS3')
ppmi_merge = add_lateralized_subscores(ppmi_merge, brady_left3a, 'Left',
↪  'Brady.Rigidity.Subscore-left.UPDRS3A')
ppmi_merge = add_lateralized_subscores(ppmi_merge, brady_left3ON, 'Left',
↪  'Brady.Rigidity.Subscore-left.UPDR3ON')
ppmi_merge = add_lateralized_subscores(ppmi_merge, brady_left3OF, 'Left',
↪  'Brady.Rigidity.Subscore-left.UPDR3OF')

ppmi_merge = add_lateralized_subscores(ppmi_merge, brady_right3, 'Right',
↪  'Brady.Rigidity.Subscore-right.UPDRS3')
ppmi_merge = add_lateralized_subscores(ppmi_merge, brady_right3a, 'Right',
↪  'Brady.Rigidity.Subscore-right.UPDRS3A')
ppmi_merge = add_lateralized_subscores(ppmi_merge, brady_right3ON, 'Right',
↪  'Brady.Rigidity.Subscore-right.UPDR3ON')
ppmi_merge = add_lateralized_subscores(ppmi_merge, brady_right3OF, 'Right',
↪  'Brady.Rigidity.Subscore-right.UPDR3OF')

ppmi_merge = add_symmetric_subscores(ppmi_merge, brady_sym3, 'Symmetric',
↪  'Brady.Rigidity.Subscore-sym.UPDRS3', '.UPDRS3.Num')
ppmi_merge = add_symmetric_subscores(ppmi_merge, brady_sym3a, 'Symmetric',
↪  'Brady.Rigidity.Subscore-sym.UPDRS3A', '.UPDRS3A.Num')
ppmi_merge = add_symmetric_subscores(ppmi_merge, brady_sym3ON, 'Symmetric',
↪  'Brady.Rigidity.Subscore-sym.UPDR3ON' , '.UPDR3ON.Num')
ppmi_merge = add_symmetric_subscores(ppmi_merge, brady_sym3OF, 'Symmetric',
↪  'Brady.Rigidity.Subscore-sym.UPDR3OF', '.UPDR3OF.Num')

# Combine left and right and sym subscores into same column
ppmi_merge["Brady.Rigidity.Subscore.lateralized.UPDRS3"] =
↪  ppmi_merge.pop("Brady.Rigidity.Subscore-right.UPDRS3").fillna(ppmi_merge.pop("Brady.Rigidity.Subsco
ppmi_merge["Brady.Rigidity.Subscore.lateralized.UPDRS3"] =
↪  ppmi_merge.pop("Brady.Rigidity.Subscore.lateralized.UPDRS3").fillna(ppmi_merge.pop("Brady.Rigidity.
```

```
ppmi_merge["Brady.Rigidity.Subscore.lateralized.UPDRS3A"] =
↪  ppmi_merge.pop("Brady.Rigidity.Subscore-right.UPDRS3A").fillna(ppmi_merge.pop("Brady.Rigidity.Subsc
ppmi_merge["Brady.Rigidity.Subscore.lateralized.UPDRS3A"] =
↪  ppmi_merge.pop("Brady.Rigidity.Subscore.lateralized.UPDRS3A").fillna(ppmi_merge.pop("Brady.Rigidity

ppmi_merge["Brady.Rigidity.Subscore.lateralized.UPDR3ON"] =
↪  ppmi_merge.pop("Brady.Rigidity.Subscore-right.UPDR3ON").fillna(ppmi_merge.pop("Brady.Rigidity.Subsc
ppmi_merge["Brady.Rigidity.Subscore.lateralized.UPDR3ON"] =
↪  ppmi_merge.pop("Brady.Rigidity.Subscore.lateralized.UPDR3ON").fillna(ppmi_merge.pop("Brady.Rigidity

ppmi_merge["Brady.Rigidity.Subscore.lateralized.UPDR3OF"] =
↪  ppmi_merge.pop("Brady.Rigidity.Subscore-right.UPDR3OF").fillna(ppmi_merge.pop("Brady.Rigidity.Subsc
ppmi_merge["Brady.Rigidity.Subscore.lateralized.UPDR3OF"] =
↪  ppmi_merge.pop("Brady.Rigidity.Subscore.lateralized.UPDR3OF").fillna(ppmi_merge.pop("Brady.Rigidity


#### TREMOR ####
tremor_right3 = ['Dominant.Side.Disease', 'Tremor.UPDRS2.Num',
↪  'Postural.Tremor.Right.Hand.UPDRS3.Num', 'Kinetic.Tremor.Right.Hand.UPDRS3.Num',
↪  'Rest.Tremor.Amplitude.RUE.UPDRS3.Num', 'Rest.Tremor.Amplitude.RLE.UPDRS3.Num',
↪  'Rest.Tremor.Amplitude.Lip.Jaw.UPDRS3.Num', 'Constancy.of.Rest.Tremor.UPDRS3.Num']
tremor_right3a = ['Dominant.Side.Disease', 'Tremor.UPDRS2.Num',
↪  'Postural.Tremor.Right.Hand.UPDRS3A.Num', 'Kinetic.Tremor.Right.Hand.UPDRS3A.Num',
↪  'Rest.Tremor.Amplitude.RUE.UPDRS3A.Num', 'Rest.Tremor.Amplitude.RLE.UPDRS3A.Num',
↪  'Rest.Tremor.Amplitude.Lip.Jaw.UPDRS3A.Num', 'Constancy.of.Rest.Tremor.UPDRS3A.Num']
tremor_right3ON = ['Dominant.Side.Disease', 'Tremor.UPDRS2.Num',
↪  'Postural.Tremor.Right.Hand.UPDR3ON.Num', 'Kinetic.Tremor.Right.Hand.UPDR3ON.Num',
↪  'Rest.Tremor.Amplitude.RUE.UPDR3ON.Num', 'Rest.Tremor.Amplitude.RLE.UPDR3ON.Num',
↪  'Rest.Tremor.Amplitude.Lip.Jaw.UPDR3ON.Num', 'Constancy.of.Rest.Tremor.UPDR3ON.Num']
tremor_right3OF = ['Dominant.Side.Disease', 'Tremor.UPDRS2.Num',
↪  'Postural.Tremor.Right.Hand.UPDR3OF.Num', 'Kinetic.Tremor.Right.Hand.UPDR3OF.Num',
↪  'Rest.Tremor.Amplitude.RUE.UPDR3OF.Num', 'Rest.Tremor.Amplitude.RLE.UPDR3OF.Num',
↪  'Rest.Tremor.Amplitude.Lip.Jaw.UPDR3OF.Num', 'Constancy.of.Rest.Tremor.UPDR3OF.Num']

tremor_left3 = ['Dominant.Side.Disease', 'Tremor.UPDRS2.Num',
↪  'Postural.Tremor.Left.Hand.UPDRS3.Num', 'Kinetic.Tremor.Left.Hand.UPDRS3.Num',
↪  'Rest.Tremor.Amplitude.LUE.UPDRS3.Num', 'Rest.Tremor.Amplitude.LLE.UPDRS3.Num',
↪  'Rest.Tremor.Amplitude.Lip.Jaw.UPDRS3.Num', 'Constancy.of.Rest.Tremor.UPDRS3.Num']
tremor_left3a = ['Dominant.Side.Disease', 'Tremor.UPDRS2.Num',
↪  'Postural.Tremor.Left.Hand.UPDRS3A.Num', 'Kinetic.Tremor.Left.Hand.UPDRS3A.Num',
↪  'Rest.Tremor.Amplitude.LUE.UPDRS3A.Num', 'Rest.Tremor.Amplitude.LLE.UPDRS3A.Num',
↪  'Rest.Tremor.Amplitude.Lip.Jaw.UPDRS3A.Num', 'Constancy.of.Rest.Tremor.UPDRS3A.Num']
tremor_left3ON = ['Dominant.Side.Disease', 'Tremor.UPDRS2.Num',
↪  'Postural.Tremor.Left.Hand.UPDR3ON.Num', 'Kinetic.Tremor.Left.Hand.UPDR3ON.Num',
↪  'Rest.Tremor.Amplitude.LUE.UPDR3ON.Num', 'Rest.Tremor.Amplitude.LLE.UPDR3ON.Num',
↪  'Rest.Tremor.Amplitude.Lip.Jaw.UPDR3ON.Num', 'Constancy.of.Rest.Tremor.UPDR3ON.Num']
tremor_left3OF = ['Dominant.Side.Disease', 'Tremor.UPDRS2.Num',
↪  'Postural.Tremor.Left.Hand.UPDR3OF.Num', 'Kinetic.Tremor.Left.Hand.UPDR3OF.Num',
↪  'Rest.Tremor.Amplitude.LUE.UPDR3OF.Num', 'Rest.Tremor.Amplitude.LLE.UPDR3OF.Num',
↪  'Rest.Tremor.Amplitude.Lip.Jaw.UPDR3OF.Num', 'Constancy.of.Rest.Tremor.UPDR3OF.Num']

tremor_sym3 = ['Dominant.Side.Disease', 'Tremor.UPDRS2.Num',
↪  'Postural.Tremor.Right.Hand.UPDRS3.Num', 'Postural.Tremor.Left.Hand.UPDRS3.Num',
↪  'Kinetic.Tremor.Right.Hand.UPDRS3.Num', 'Kinetic.Tremor.Left.Hand.UPDRS3.Num',
↪  'Rest.Tremor.Amplitude.RUE.UPDRS3.Num', 'Rest.Tremor.Amplitude.LUE.UPDRS3.Num',
↪  'Rest.Tremor.Amplitude.RLE.UPDRS3.Num', 'Rest.Tremor.Amplitude.LLE.UPDRS3.Num',
↪  'Rest.Tremor.Amplitude.Lip.Jaw.UPDRS3.Num', 'Constancy.of.Rest.Tremor.UPDRS3.Num']
```

```
tremor_sym3a = ['Dominant.Side.Disease', 'Tremor.UPDRS2.Num',
↪  'Postural.Tremor.Right.Hand.UPDRS3A.Num', 'Postural.Tremor.Left.Hand.UPDRS3A.Num',
↪  'Kinetic.Tremor.Right.Hand.UPDRS3A.Num', 'Kinetic.Tremor.Left.Hand.UPDRS3A.Num',
↪  'Rest.Tremor.Amplitude.RUE.UPDRS3A.Num', 'Rest.Tremor.Amplitude.LUE.UPDRS3A.Num',
↪  'Rest.Tremor.Amplitude.RLE.UPDRS3A.Num', 'Rest.Tremor.Amplitude.LLE.UPDRS3A.Num',
↪  'Rest.Tremor.Amplitude.Lip.Jaw.UPDRS3A.Num', 'Constancy.of.Rest.Tremor.UPDRS3A.Num']
tremor_sym3ON = ['Dominant.Side.Disease', 'Tremor.UPDRS2.Num',
↪  'Postural.Tremor.Right.Hand.UPDR3ON.Num', 'Postural.Tremor.Left.Hand.UPDR3ON.Num',
↪  'Kinetic.Tremor.Right.Hand.UPDR3ON.Num', 'Kinetic.Tremor.Left.Hand.UPDR3ON.Num',
↪  'Rest.Tremor.Amplitude.RUE.UPDR3ON.Num', 'Rest.Tremor.Amplitude.LUE.UPDR3ON.Num',
↪  'Rest.Tremor.Amplitude.RLE.UPDR3ON.Num', 'Rest.Tremor.Amplitude.LLE.UPDR3ON.Num',
↪  'Rest.Tremor.Amplitude.Lip.Jaw.UPDR3ON.Num', 'Constancy.of.Rest.Tremor.UPDR3ON.Num']
tremor_sym3OF = ['Dominant.Side.Disease', 'Tremor.UPDRS2.Num',
↪  'Postural.Tremor.Right.Hand.UPDR3OF.Num', 'Postural.Tremor.Left.Hand.UPDR3OF.Num',
↪  'Kinetic.Tremor.Right.Hand.UPDR3OF.Num', 'Kinetic.Tremor.Left.Hand.UPDR3OF.Num',
↪  'Rest.Tremor.Amplitude.RUE.UPDR3OF.Num', 'Rest.Tremor.Amplitude.LUE.UPDR3OF.Num',
↪  'Rest.Tremor.Amplitude.RLE.UPDR3OF.Num', 'Rest.Tremor.Amplitude.LLE.UPDR3OF.Num',
↪  'Rest.Tremor.Amplitude.Lip.Jaw.UPDR3OF.Num', 'Constancy.of.Rest.Tremor.UPDR3OF.Num']

ppmi_merge = add_lateralized_subscores(ppmi_merge, tremor_left3, 'Left',
↪  'Tremor.Subscore-left.UPDRS3')
ppmi_merge = add_lateralized_subscores(ppmi_merge, tremor_left3a, 'Left',
↪  'Tremor.Subscore-left.UPDRS3A')
ppmi_merge = add_lateralized_subscores(ppmi_merge, tremor_left3ON, 'Left',
↪  'Tremor.Subscore-left.UPDR3ON')
ppmi_merge = add_lateralized_subscores(ppmi_merge, tremor_left3OF, 'Left',
↪  'Tremor.Subscore-left.UPDR3OF')

ppmi_merge = add_lateralized_subscores(ppmi_merge, tremor_right3, 'Right',
↪  'Tremor.Subscore-right.UPDRS3')
ppmi_merge = add_lateralized_subscores(ppmi_merge, tremor_right3a, 'Right',
↪  'Tremor.Subscore-right.UPDRS3A')
ppmi_merge = add_lateralized_subscores(ppmi_merge, tremor_right3ON, 'Right',
↪  'Tremor.Subscore-right.UPDR3ON')
ppmi_merge = add_lateralized_subscores(ppmi_merge, tremor_right3OF, 'Right',
↪  'Tremor.Subscore-right.UPDR3OF')

ppmi_merge = add_symmetric_subscores(ppmi_merge, tremor_sym3, 'Symmetric',
↪  'Tremor.Subscore-sym.UPDRS3', '.UPDRS3.Num')
ppmi_merge = add_symmetric_subscores(ppmi_merge, tremor_sym3a, 'Symmetric',
↪  'Tremor.Subscore-sym.UPDRS3A', '.UPDRS3A.Num')
ppmi_merge = add_symmetric_subscores(ppmi_merge, tremor_sym3ON, 'Symmetric',
↪  'Tremor.Subscore-sym.UPDR3ON', '.UPDR3ON.Num')
ppmi_merge = add_symmetric_subscores(ppmi_merge, tremor_sym3OF, 'Symmetric',
↪  'Tremor.Subscore-sym.UPDR3OF', '.UPDR3OF.Num')


# Combine left and right and sym scores into same column (.lateralized)
ppmi_merge["Tremor.Subscore.lateralized.UPDRS3"] =
↪  ppmi_merge.pop("Tremor.Subscore-right.UPDRS3").fillna(ppmi_merge.pop("Tremor.Subscore-left.UPDRS3"))
ppmi_merge["Tremor.Subscore.lateralized.UPDRS3"] =
↪  ppmi_merge.pop("Tremor.Subscore.lateralized.UPDRS3").fillna(ppmi_merge.pop("Tremor.Subscore-sym.UPDR
```

```
ppmi_merge["Tremor.Subscore.lateralized.UPDRS3A"] =
↪  ppmi_merge.pop("Tremor.Subscore-right.UPDRS3A").fillna(ppmi_merge.pop("Tremor.Subscore-left.UPDRS3A
ppmi_merge["Tremor.Subscore.lateralized.UPDRS3A"] =
↪  ppmi_merge.pop("Tremor.Subscore.lateralized.UPDRS3A").fillna(ppmi_merge.pop("Tremor.Subscore-sym.UP

ppmi_merge["Tremor.Subscore.lateralized.UPDR3ON"] =
↪  ppmi_merge.pop("Tremor.Subscore-right.UPDR3ON").fillna(ppmi_merge.pop("Tremor.Subscore-left.UPDR3ON
ppmi_merge["Tremor.Subscore.lateralized.UPDR3ON"] =
↪  ppmi_merge.pop("Tremor.Subscore.lateralized.UPDR3ON").fillna(ppmi_merge.pop("Tremor.Subscore-sym.UP

ppmi_merge["Tremor.Subscore.lateralized.UPDR3OF"] =
↪  ppmi_merge.pop("Tremor.Subscore-right.UPDR3OF").fillna(ppmi_merge.pop("Tremor.Subscore-left.UPDR3OF
ppmi_merge["Tremor.Subscore.lateralized.UPDR3OF"] =
↪  ppmi_merge.pop("Tremor.Subscore.lateralized.UPDR3OF").fillna(ppmi_merge.pop("Tremor.Subscore-sym.UP

## If lateralized subscore is nan - input the non-lateralized score for Tremor and
↪  Brady.Rigidity
def fill_non_lat_subscore(df, subscore_lateralized_name, subscore_name) :
    latisna = df[subscore_lateralized_name].isna() & df[subscore_name].notna()
    df[subscore_lateralized_name].loc[latisna] = df[subscore_name].loc[latisna]
    return df

ppmi_merge = fill_non_lat_subscore(ppmi_merge, 'Tremor.Subscore.lateralized.UPDRS3',
↪  'Tremor.Subscore.UPDRS3')
ppmi_merge = fill_non_lat_subscore(ppmi_merge, 'Tremor.Subscore.lateralized.UPDRS3A',
↪  'Tremor.Subscore.UPDRS3A')
ppmi_merge = fill_non_lat_subscore(ppmi_merge, 'Tremor.Subscore.lateralized.UPDR3ON',
↪  'Tremor.Subscore.UPDR3ON')
ppmi_merge = fill_non_lat_subscore(ppmi_merge, 'Tremor.Subscore.lateralized.UPDR3OF',
↪  'Tremor.Subscore.UPDR3OF')

ppmi_merge = fill_non_lat_subscore(ppmi_merge,
↪  'Brady.Rigidity.Subscore.lateralized.UPDRS3', 'Brady.Rigidity.Subscore.UPDRS3')
ppmi_merge = fill_non_lat_subscore(ppmi_merge,
↪  'Brady.Rigidity.Subscore.lateralized.UPDRS3A', 'Brady.Rigidity.Subscore.UPDRS3A')
ppmi_merge = fill_non_lat_subscore(ppmi_merge,
↪  'Brady.Rigidity.Subscore.lateralized.UPDR3ON', 'Brady.Rigidity.Subscore.UPDR3ON')
ppmi_merge = fill_non_lat_subscore(ppmi_merge,
↪  'Brady.Rigidity.Subscore.lateralized.UPDR3OF', 'Brady.Rigidity.Subscore.UPDR3OF')


## Inlcude columns for bestEventID (bestScreening, bestBaseline, etc) and denote the
↪  highest resnetGrade with True (else = False)
myevs = ppmi_merge['Event.ID'].unique() # Unique event ids
uids = ppmi_merge['Subject.ID'].unique() # Unique subject ids
for myev in myevs :
    mybe = "best" + myev # Create best Visit column
    ppmi_merge[mybe] = False # Set all best visit to be False
    for u in uids :
        selu = ppmi_merge.loc[(ppmi_merge['Subject.ID'] == u) & (ppmi_merge['Event.ID']
↪  == myev) & (ppmi_merge['resnetGrade'].notna())] # For one subject at one event id if
↪  resnetGrade not na
        if len(selu) == 1 : # If there is one event id for that subject
```

```
            idx = selu.index # Ge the index
            ppmi_merge.loc[idx, mybe] = True
        if len(selu) > 1 : # IF there is more than one event id for that subject and
        ↪  resnet grade is not na
            maxidx = selu[['resnetGrade']].idxmax() # Get the higher resnetGrade for each
↪  visit if there are more than one
            ppmi_merge.loc[maxidx, mybe] = True

## Include a column for bestAtImage.Acquisition.Date - denote the one or highest
↪  resnetGrade with True (else = False)
ppmi_merge['bestAtImage.Acquisition.Date'] = False # Initialize
↪  bestAtImage.Acuqisition.Date col
for myev in myevs :
    for u in uids :
        selu = ppmi_merge.loc[(ppmi_merge['Subject.ID'] == u) & (ppmi_merge['Event.ID']
↪  == myev) & (ppmi_merge['resnetGrade'].notna())]
        if len(selu) == 1 : # If there is one event id for that subject
            idx = selu.index # Get the index
            ppmi_merge.loc[idx, 'bestAtImage.Acquisition.Date'] = True
        if len(selu) > 1 : # IF there is more than one event id for that subject and
        ↪  resnet grade is not na
            maxidx = selu[['resnetGrade']].idxmax() # Get the higher resnetGrade for each
↪  visit if there are more than one
            ppmi_merge.loc[maxidx, 'bestAtImage.Acquisition.Date'] = True

# DX simplified column
ppmi_merge['DXsimplified'] = '' # Initialize DXsimplfied
ppmi_merge.loc[ppmi_merge['Consensus.Subtype'] == 'Healthy Control', 'DXsimplified'] =
↪  'HC'
ppmi_merge.loc[ppmi_merge['Enroll.Diagnosis'] == 'Healthy Control', 'DXsimplified'] =
↪  'HC'
ppmi_merge.loc[ppmi_merge['Enroll.Diagnosis'] == 'Parkinson\'s Disease', 'DXsimplified']
↪  = 'Sporadic_PD'
ppmi_merge.loc[ppmi_merge['Enroll.Diagnosis'] == 'SWEDD', 'DXsimplified'] = "nonPDorMSA"
ppmi_merge.loc[ppmi_merge['Enroll.Diagnosis'] == "Prodromal", 'DXsimplified'] =
↪  "Sporadic_Pro"
ppmi_merge.loc[ppmi_merge['Consensus.Subtype'] == "GBA", 'DXsimplified'] = "GBA_HC"
ppmi_merge.loc[ppmi_merge['Consensus.Subtype'] == "Genetic : GBA", 'DXsimplified'] =
↪  "GBA_PD"
ppmi_merge.loc[ppmi_merge['Consensus.Subtype'] == "Genetic : GBA not Prodromal",
↪  'DXsimplified'] = "GBA_PD"
ppmi_merge.loc[ppmi_merge['Consensus.Subtype'] == "Genetic : GBA Prodromal",
↪  'DXsimplified'] = "GBA_Pro"
ppmi_merge.loc[ppmi_merge['Consensus.Subtype'] == "Genetic : LRRK2", 'DXsimplified'] =
↪  "LRRK2_PD"
ppmi_merge.loc[ppmi_merge['Consensus.Subtype'] == "Genetic : LRRK2 + GBA",
↪  'DXsimplified'] = "LRRK2_PD"
ppmi_merge.loc[ppmi_merge['Consensus.Subtype'] == "Genetic : LRRK2 + GBA not Prodromal",
↪  'DXsimplified'] = "LRRK2_PD"
ppmi_merge.loc[ppmi_merge['Consensus.Subtype'] == "Genetic : LRRK2 + GBA Prodromal",
↪  'DXsimplified'] = "LRRK2_Pro"
ppmi_merge.loc[ppmi_merge['Consensus.Subtype'] == "Genetic : LRRK2 not Prodromal",
↪  'DXsimplified'] = "LRRK2_PD"
```

```
ppmi_merge.loc[ppmi_merge['Consensus.Subtype'] == "Genetic : LRRK2 Phenoconverted",
↪  'DXsimplified'] = "LRRK2_Pro"
ppmi_merge.loc[ppmi_merge['Consensus.Subtype'] ==  "Genetic : LRRK2 Prodromal",
↪  'DXsimplified'] = "LRRK2_Pro"
ppmi_merge.loc[ppmi_merge['Consensus.Subtype'] ==  "Genetic : SNCA", 'DXsimplified'] =
↪  "SNCA_PD"
ppmi_merge.loc[ppmi_merge['Consensus.Subtype'] == "Genetic : SNCA Prodromal",
↪  'DXsimplified'] = "SNCA_Pro"
ppmi_merge.loc[ppmi_merge['Consensus.Subtype'] == "Hyposmia" , 'DXsimplified'] =
↪  "Sporadic_Pro"
ppmi_merge.loc[ppmi_merge['Consensus.Subtype'] == "Hyposmia : Phenoconverted",
↪  'DXsimplified'] = "Sporadic_Pro"
ppmi_merge.loc[ppmi_merge['Consensus.Subtype'] == "No Mutation not Prodromal",
↪  'DXsimplified'] = np.NaN
ppmi_merge.loc[ppmi_merge['Consensus.Subtype'] == "non-HC", 'DXsimplified'] = np.NaN
ppmi_merge.loc[ppmi_merge['Consensus.Subtype'] == "non-PD", 'DXsimplified'] =
↪  "nonPDorMSA"
ppmi_merge.loc[ppmi_merge['Consensus.Subtype'] == "RBD", 'DXsimplified'] = "Sporadic_Pro"
ppmi_merge.loc[ppmi_merge['Consensus.Subtype'] == "RBD : Phenoconverted", 'DXsimplified']
↪  = "Sporadic_Pro"
ppmi_merge.loc[ppmi_merge['Consensus.Subtype'] ==  "RBD : Phenoconverted with GBA",
↪  'DXsimplified'] = "GBA_Pro"
ppmi_merge.loc[ppmi_merge['Consensus.Subtype'] ==  "Sporadic", 'DXsimplified'] =
↪  "Sporadic_PD"
ppmi_merge.loc[ppmi_merge['Consensus.Subtype'] ==  "SWEDD/non-PD Active", 'DXsimplified']
↪  = "nonPDorMSA"
ppmi_merge.loc[ppmi_merge['Consensus.Subtype'] ==  "SWEDD/PD Active", 'DXsimplified'] =
↪  np.NaN

## Add in min PD Disease duration
# Create a simplified diagnosis group of just HC, PD, Prodromal, or nonPDorMSA
ppmi_merge['DXs2'] = ''
ppmi_merge['DXsimplified'].fillna('NA', inplace = True)
ppmi_merge.loc[ppmi_merge['DXsimplified'].str.contains('HC'), 'DXs2'] = 'HC'
ppmi_merge.loc[ppmi_merge['DXsimplified'].str.contains('_PD'), 'DXs2'] = 'PD'
ppmi_merge.loc[ppmi_merge['DXsimplified'].str.contains('_Pro'), 'DXs2'] = 'Pro'
ppmi_merge.loc[ppmi_merge['DXsimplified'].str.contains('nonPDorMSA'), 'DXs2'] =
↪  'nonPDorMSA'

# If HC, Prodromal or nonPDorMSA - fill in with 0 for PD.Min.Disease.Duration
ppmi_merge.loc[ppmi_merge['DXs2'] == 'HC', 'PD.Min.Disease.Duration'] = 0
ppmi_merge.loc[ppmi_merge['DXs2'] == 'Pro', 'PD.Min.Disease.Duration'] = 0
ppmi_merge.loc[ppmi_merge['DXs2'] == 'nonPDorMSA', 'PD.Min.Disease.Duration'] = 0

# For PD subjects, fill in PD.Min.Disease.Duration
ppmi_merge['PD.Diagnosis.Duration'].fillna('', inplace = True)

subids = ppmi_merge['Subject.ID'].unique() # Unique subject ids
for subid in subids :
    df = ppmi_merge[(ppmi_merge['Subject.ID'] == subid) & (ppmi_merge['DXs2'] == 'PD') &
↪  (ppmi_merge['PD.Diagnosis.Duration'] != '')]
    if len(df) > 0 :
        print(df['PD.Diagnosis.Duration'])
```

```
        mydd = min(df['PD.Diagnosis.Duration'])
        ppmi_merge.loc[ppmi_merge['Subject.ID'] == subid, 'PD.Min.Disease.Duration'] =
↪  mydd

## Add in Visit column
```

```
ppmi_merge['Visit'] = np.NaN # Initialize Visit col
searchfor = ['Baseline', 'Visit Month '] # Strings to search for
temp = ppmi_merge['Event.ID'].str.contains('|'.join(searchfor)) # locations of where row
↪  contains str: baseline or visit month
ppmi_merge['Visit'].loc[temp == True] = ppmi_merge['Event.ID'].loc[temp == True] # Fill
↪  in 'Visit' col with event.ID for baseline or visit month
ppmi_merge['Visit'] = ppmi_merge['Visit'].str.replace("Remote Visit Month ", "") #
↪  Replace Remote visit month with '' (want only month number)
ppmi_merge['Visit'] = ppmi_merge['Visit'].str.replace("Visit Month ", "") # Replace Visit
↪  month with '' (want only month number)
ppmi_merge['Visit'] = ppmi_merge['Visit'].str.replace("Baseline", "0") # Replace baseline
↪  with 0
ppmi_merge['Visit'] = ppmi_merge['Visit'].fillna(9999) # Filling with 9999 so we can
↪  change this col to int
ppmi_merge['Visit'] = ppmi_merge['Visit'].astype(int) # str to int
ppmi_merge['Visit'] = ppmi_merge['Visit'].replace(9999, np.nan)  # Replace 9999 with nan

# Final re-organization of ppmi_merge and save
ppmi_merge.set_index('Subject.ID', inplace = True)
ppmi_merge.fillna('NA', inplace = True)
ppmi_merge.to_csv(userdir + 'ppmi_merge_v' + version + '.csv')
```