PPMI Merge

2022-03-21

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

## PPMI Study Data path
userdir = '/Users/areardon/Desktop/ppmi_merge/'
ppmi_download_path = userdir + 'PPMI_Study_Data_Download/'
invicro_data_path =  userdir
genetics_path = userdir + 'genetic_data/'

#### CLINICAL INFO ####
## Create the initial dataframe with the subjects listed in
↪  Consensus_Committee_Analytic_Datasets_28OCT21.xlsx
xlsx = pd.ExcelFile(ppmi_download_path +
↪  'Consensus_Committee_Analytic_Datasets_28OCT21.xlsx') # Read in main xlsx file

# Create Parkinson's Disease data frame from 'PD' sheet in
↪  'Consensus_Committee_Analytic_Datasets_28OCT21.xlsx'
pd_df = pd.read_excel(xlsx, 'PD') # Parkinson's subject sheet
cols_delete = ['Unnamed', 'CONDATE'] # Columns to remove from sheet/ df
pd_df = pd_df.loc[:, ~pd_df.columns.str.startswith(tuple(cols_delete))] # Remove columns
↪  in cols_delete

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

# Decode Enrollment.Subtype and Consensus.Subtype to be categorical variables
```

```
for row_num in range(len(pd_df)) :
    # Enroll Subtype - info from 'Summary' sheet of
    ↪  'Consensus_Committee_Analytic_Datasets_28OCT21.xlsx'
    if pd_df['ENRLPD'].iloc[row_num] == 1 and pd_df['ENRLLRRK2'].iloc[row_num] == 0 and
    ↪  pd_df['ENRLGBA'].iloc[row_num] == 0 and pd_df['ENRLSNCA'].iloc[row_num] == 0 :
        pd_df['Enrollment.Subtype'].iloc[row_num] = '' # Sporadic - don't need to define
↪  here bc already covered in 'Subgroup' column in PD sheet
    if pd_df['ENRLPD'].iloc[row_num] == 1 and pd_df['ENRLLRRK2'].iloc[row_num] == 1 :
        pd_df['Enrollment.Subtype'].iloc[row_num] = ' : LRRK2'
    if pd_df['ENRLPD'].iloc[row_num] == 1 and pd_df['ENRLGBA'].iloc[row_num] == 1 :
        pd_df['Enrollment.Subtype'].iloc[row_num] = ' : GBA'
    if pd_df['ENRLPD'].iloc[row_num] == 1 and pd_df['ENRLSNCA'].iloc[row_num] == 1 :
        pd_df['Enrollment.Subtype'].iloc[row_num] = ' : SNCA'

    # Consensus Subtype - info from 'Summary Analytic' shset of
    ↪  'Consensus_Committee_Analytic_Datasets_28OCT21.xlsx' - need to include '.'
    ↪  (unknowns) in order for numbers ot match up with 'Summary Analytic' sheet Ns
    if pd_df['CONPD'].iloc[row_num] == 1 and pd_df['CONLRRK2'].iloc[row_num] == 0 or
    ↪  pd_df['CONLRRK2'].iloc[row_num] == '.' and pd_df['CONGBA'].iloc[row_num] == 0 or
    ↪  pd_df['CONGBA'].iloc[row_num] == '.' and pd_df['CONSNCA'].iloc[row_num] == 0 or
    ↪  pd_df['CONSNCA'].iloc[row_num] == '.' :
        pd_df['Consensus.Subtype'].iloc[row_num] = 'Sporadic'
    if pd_df['CONPD'].iloc[row_num] == 1 and pd_df['CONLRRK2'].iloc[row_num] == 1 and
    ↪  pd_df['CONGBA'].iloc[row_num] == 0 and pd_df['CONSNCA'].iloc[row_num] == 0 :
        pd_df['Consensus.Subtype'].iloc[row_num] = 'Genetic : LRRK2'
    if pd_df['CONPD'].iloc[row_num] == 1 and pd_df['CONLRRK2'].iloc[row_num] == 0 and
    ↪  pd_df['CONGBA'].iloc[row_num] == 1 and pd_df['CONSNCA'].iloc[row_num] == 0 :
        pd_df['Consensus.Subtype'].iloc[row_num] = 'Genetic : GBA'
    if pd_df['CONPD'].iloc[row_num] == 1 and pd_df['CONLRRK2'].iloc[row_num] == 1 and
    ↪  pd_df['CONGBA'].iloc[row_num] == 1 and pd_df['CONSNCA'].iloc[row_num] == 0 :
        pd_df['Consensus.Subtype'].iloc[row_num] = 'Genetic : LRRK2 + GBA'
    if pd_df['CONPD'].iloc[row_num] == 1 and pd_df['CONLRRK2'].iloc[row_num] == 0 and
    ↪  pd_df['CONGBA'].iloc[row_num] == 0 and pd_df['CONSNCA'].iloc[row_num] == 1 :
        pd_df['Consensus.Subtype'].iloc[row_num] = 'Genetic : SNCA'
    if pd_df['CONPD'].iloc[row_num] == 0 and pd_df['CONPROD'].iloc[row_num] == 0 :
        pd_df['Consensus.Subtype'].iloc[row_num] = 'non-PD'
    if pd_df['CONPD'].iloc[row_num] == 0 and pd_df['CONPROD'].iloc[row_num] == 1 and
    ↪  pd_df['CONLRRK2'].iloc[row_num] == 1 and pd_df['CONGBA'].iloc[row_num] == 0 and
    ↪  pd_df['CONSNCA'].iloc[row_num] == 0 :
        pd_df['Consensus.Subtype'].iloc[row_num] = 'Genetic : LRRK2 Prodromal'
    if pd_df['CONPD'].iloc[row_num] == 0 and pd_df['CONPROD'].iloc[row_num] == 1 and
    ↪  pd_df['CONLRRK2'].iloc[row_num] == 0 and pd_df['CONGBA'].iloc[row_num] == 1 and
    ↪  pd_df['CONSNCA'].iloc[row_num] == 0 :
        pd_df['Consensus.Subtype'].iloc[row_num] = 'Genetic : GBA Prodromal'

# Create Prodromal data frame
prodromal_df = pd.read_excel(xlsx, 'Prodromal') # Prodromal subjects
prodromal_df = prodromal_df.loc[:,
↪  ~prodromal_df.columns.str.startswith(tuple(cols_delete))] # Remove columns in
↪  cols_delete

# Create Consensus.Subtype and Enrollment.Subtype in prodromal_df
prodromal_df["Consensus.Subtype"] = ''# Create new column called 'Subtype Renamed in
↪  prodromal_df
```

```
prodromal_df["Enrollment.Subtype"] = '' # Create new column called 'Enrollment.Subtype'
↪  in pd_df

# Decode Enrollment.Subtype and Consensus.Subtype to be categorical variables
for row_num in range(len(prodromal_df)) :
    # Enrollment Subtype
    if prodromal_df['ENRLPROD'].iloc[row_num] == 1 and
    ↪  prodromal_df['ENRLLRRK2'].iloc[row_num] == 1 :
        prodromal_df['Enrollment.Subtype'].iloc[row_num] = ' : LRRK2 Prodromal'
    if prodromal_df['ENRLPROD'].iloc[row_num] == 1 and
    ↪  prodromal_df['ENRLGBA'].iloc[row_num] == 1 :
        prodromal_df['Enrollment.Subtype'].iloc[row_num] = ' : GBA Prodromal'
    if prodromal_df['ENRLPROD'].iloc[row_num] == 1 and
    ↪  prodromal_df['ENRLSNCA'].iloc[row_num] == 1 :
        prodromal_df['Enrollment.Subtype'].iloc[row_num] = ' : SNCA Prodromal'
    if prodromal_df['ENRLPROD'].iloc[row_num] == 1 and
    ↪  prodromal_df['ENRLHPSM'].iloc[row_num] == 1 :
        prodromal_df['Enrollment.Subtype'].iloc[row_num] = '' # Hyposmia already covered
↪  in 'Subgroup' column
    if prodromal_df['ENRLPROD'].iloc[row_num] == 1 and
    ↪  prodromal_df['ENRLRBD'].iloc[row_num] == 1 :
        prodromal_df['Enrollment.Subtype'].iloc[row_num] = '' # RBD already covered in
↪  'Subgroup' column

    # Consensus Subtype
    if prodromal_df['CONPROD'].iloc[row_num] == 1 and
    ↪  prodromal_df['PHENOCNV'].iloc[row_num] == 0 and
    ↪  prodromal_df['CONLRRK2'].iloc[row_num] == 1 and
    ↪  prodromal_df['CONGBA'].iloc[row_num] == 0 and
    ↪  prodromal_df['CONSNCA'].iloc[row_num] == 0 :
        prodromal_df['Consensus.Subtype'].iloc[row_num] = 'Genetic : LRRK2 Prodromal'
    if prodromal_df['CONPROD'].iloc[row_num] == 1 and
    ↪  prodromal_df['PHENOCNV'].iloc[row_num] == 1 and
    ↪  prodromal_df['CONLRRK2'].iloc[row_num] == 1 and
    ↪  prodromal_df['CONGBA'].iloc[row_num] == 0 and
    ↪  prodromal_df['CONSNCA'].iloc[row_num] == 0 :
        prodromal_df['Consensus.Subtype'].iloc[row_num] = 'Genetic : LRRK2
↪  Phenoconverted'
    if prodromal_df['CONPROD'].iloc[row_num] == 0 and
    ↪  prodromal_df['CONLRRK2'].iloc[row_num] == 1 and
    ↪  prodromal_df['CONGBA'].iloc[row_num] == 0 and
    ↪  prodromal_df['CONSNCA'].iloc[row_num] == 0 :
        prodromal_df['Consensus.Subtype'].iloc[row_num] = 'Genetic : LRRK2 not Prodromal'
    if prodromal_df['CONPROD'].iloc[row_num] == 1 and
    ↪  prodromal_df['PHENOCNV'].iloc[row_num] == 0 and
    ↪  prodromal_df['CONLRRK2'].iloc[row_num] == 0 and
    ↪  prodromal_df['CONGBA'].iloc[row_num] == 1 and
    ↪  prodromal_df['CONSNCA'].iloc[row_num] == 0 :
        prodromal_df['Consensus.Subtype'].iloc[row_num] = 'Genetic : GBA Prodromal'
    if prodromal_df['CONPROD'].iloc[row_num] == 1 and
    ↪  prodromal_df['PHENOCNV'].iloc[row_num] == 1 and
    ↪  prodromal_df['CONLRRK2'].iloc[row_num] == 0 and
    ↪  prodromal_df['CONGBA'].iloc[row_num] == 1 and
    ↪  prodromal_df['CONSNCA'].iloc[row_num] == 0 and
    ↪  prodromal_df['CONRBD'].iloc[row_num] == 0 :
```

```
        prodromal_df['Consensus.Subtype'].iloc[row_num] = 'Genetic : GBA Phenoconverted'
    if prodromal_df['CONPROD'].iloc[row_num] == 0 and
    ↪  prodromal_df['CONLRRK2'].iloc[row_num] == 0 and
    ↪  prodromal_df['CONGBA'].iloc[row_num] == 1 and
    ↪  prodromal_df['CONSNCA'].iloc[row_num] == 0 :
        prodromal_df['Consensus.Subtype'].iloc[row_num] = 'Genetic : GBA not Prodromal'
    if prodromal_df['CONPROD'].iloc[row_num] == 1 and
    ↪  prodromal_df['PHENOCNV'].iloc[row_num] == 0 and
    ↪  prodromal_df['CONLRRK2'].iloc[row_num] == 1 and
    ↪  prodromal_df['CONGBA'].iloc[row_num] == 1 and
    ↪  prodromal_df['CONSNCA'].iloc[row_num] == 0 :
        prodromal_df['Consensus.Subtype'].iloc[row_num] = 'Genetic : LRRK2 + GBA
↪  Prodromal'
    if prodromal_df['CONPROD'].iloc[row_num] == 1 and
    ↪  prodromal_df['PHENOCNV'].iloc[row_num] == 1 and
    ↪  prodromal_df['CONLRRK2'].iloc[row_num] == 1 and
    ↪  prodromal_df['CONGBA'].iloc[row_num] == 1 and
    ↪  prodromal_df['CONSNCA'].iloc[row_num] == 0 :
        prodromal_df['Consensus.Subtype'].iloc[row_num] = 'Genetic : LRRK2 + GBA
↪  Phenoconverted'
    if prodromal_df['CONPROD'].iloc[row_num] == 0 and
    ↪  prodromal_df['CONLRRK2'].iloc[row_num] == 1 and
    ↪  prodromal_df['CONGBA'].iloc[row_num] == 1 and
    ↪  prodromal_df['CONSNCA'].iloc[row_num] == 0 :
        prodromal_df['Consensus.Subtype'].iloc[row_num] = 'Genetic : LRRK2 + GBA not
↪  Prodromal'
    if prodromal_df['CONPROD'].iloc[row_num] == 1 and
    ↪  prodromal_df['PHENOCNV'].iloc[row_num] == 0 and
    ↪  prodromal_df['CONLRRK2'].iloc[row_num] == 0 and
    ↪  prodromal_df['CONGBA'].iloc[row_num] == 0 and
    ↪  prodromal_df['CONSNCA'].iloc[row_num] == 1 :
        prodromal_df['Consensus.Subtype'].iloc[row_num] = 'Genetic : SNCA Prodromal'
    if prodromal_df['CONPROD'].iloc[row_num] == 1 and
    ↪  prodromal_df['PHENOCNV'].iloc[row_num] == 1 and
    ↪  prodromal_df['CONLRRK2'].iloc[row_num] == 0 and
    ↪  prodromal_df['CONGBA'].iloc[row_num] == 0 and
    ↪  prodromal_df['CONSNCA'].iloc[row_num] == 1 :
        prodromal_df['Consensus.Subtype'].iloc[row_num] = 'Genetic : SNCA Phenoconverted'
    if prodromal_df['CONPROD'].iloc[row_num] == 0 and
    ↪  prodromal_df['CONLRRK2'].iloc[row_num] == 0 and
    ↪  prodromal_df['CONGBA'].iloc[row_num] == 0 and
    ↪  prodromal_df['CONSNCA'].iloc[row_num] == 1 :
        prodromal_df['Consensus.Subtype'].iloc[row_num] = 'Genetic : SNCA not Prodromal'
    if prodromal_df['CONPROD'].iloc[row_num] == 0 and
    ↪  prodromal_df['CONLRRK2'].iloc[row_num] == 0 and
    ↪  prodromal_df['CONGBA'].iloc[row_num] == 0 and
    ↪  prodromal_df['CONSNCA'].iloc[row_num] == 0 and
    ↪  prodromal_df['CONHPSM'].iloc[row_num] == 0 and
    ↪  prodromal_df['CONRBD'].iloc[row_num] == '.':
        prodromal_df['Consensus.Subtype'].iloc[row_num] = 'No Mutation not Prodromal'
    if prodromal_df['CONPROD'].iloc[row_num] == 1 and
    ↪  prodromal_df['PHENOCNV'].iloc[row_num] == 0 and
    ↪  prodromal_df['CONLRRK2'].iloc[row_num] == 0 and
    ↪  prodromal_df['CONGBA'].iloc[row_num] == 0 and
    ↪  prodromal_df['CONSNCA'].iloc[row_num] == 0 and
    ↪  prodromal_df['CONHPSM'].iloc[row_num] == 1 and
    ↪  prodromal_df['CONRBD'].iloc[row_num] == '.':
```

```
        prodromal_df['Consensus.Subtype'].iloc[row_num] = 'Hyposmia'
    if prodromal_df['CONPROD'].iloc[row_num] == 1 and
    ↪  prodromal_df['PHENOCNV'].iloc[row_num] == 1 and
    ↪  prodromal_df['CONLRRK2'].iloc[row_num] == 0 and
    ↪  prodromal_df['CONGBA'].iloc[row_num] == 0 and
    ↪  prodromal_df['CONSNCA'].iloc[row_num] == 0 and
    ↪  prodromal_df['CONHPSM'].iloc[row_num] == 1 and
    ↪  prodromal_df['CONRBD'].iloc[row_num] == '.':
        prodromal_df['Consensus.Subtype'].iloc[row_num] = 'Hyposmia : Phenoconverted'
    if prodromal_df['CONPROD'].iloc[row_num] == 0 and
    ↪  prodromal_df['CONLRRK2'].iloc[row_num] == 0 and
    ↪  prodromal_df['CONGBA'].iloc[row_num] == 0 and
    ↪  prodromal_df['CONSNCA'].iloc[row_num] == 0 and
    ↪  prodromal_df['CONHPSM'].iloc[row_num] == 1 and
    ↪  prodromal_df['CONRBD'].iloc[row_num] == '.':
        prodromal_df['Consensus.Subtype'].iloc[row_num] = 'Hyposmia : not Prodromal'
    if prodromal_df['CONPROD'].iloc[row_num] == 1 and
    ↪  prodromal_df['CONRBD'].iloc[row_num] == 1 and
    ↪  prodromal_df['PHENOCNV'].iloc[row_num] == 0 :
        prodromal_df['Consensus.Subtype'].iloc[row_num] = 'RBD'
    if prodromal_df['CONPROD'].iloc[row_num] == 1 and
    ↪  prodromal_df['CONRBD'].iloc[row_num] == 1 and
    ↪  prodromal_df['PHENOCNV'].iloc[row_num] == 1 and
    ↪  prodromal_df['CONGBA'].iloc[row_num] == 0:
        prodromal_df['Consensus.Subtype'].iloc[row_num] = 'RBD : Phenoconverted'
    if prodromal_df['CONPROD'].iloc[row_num] == 1 and
    ↪  prodromal_df['CONRBD'].iloc[row_num] == 1 and
    ↪  prodromal_df['PHENOCNV'].iloc[row_num] == 1 and
    ↪  prodromal_df['CONGBA'].iloc[row_num] == 1:
        prodromal_df['Consensus.Subtype'].iloc[row_num] = 'RBD : Phenoconverted with GBA'

# Create Healthy Control dataframe
hc_df = pd.read_excel(xlsx, 'HC') # Control subjects
hc_df = hc_df.loc[:, ~hc_df.columns.str.startswith(tuple(cols_delete))] # Remove columns
↪  that begin with ENRL

# Create Enrollment.Subtype and Consensus.Subtype columns in hc_df
hc_df['Enrollment.Subtype'] = ''
hc_df['Consensus.Subtype'] = '' # Create new column called 'Subtype Renamed in hc_df
hc_df['Subgroup'] = 'Healthy Control' #Because HC sheet does not have subgroup column -
↪  add it in and make them all Healthy Control

# Decode Enrollment.Subtype and Consensus.Subtype to be categorical variables
for row_num in range(len(hc_df)) :
    # Enrollment
    if hc_df['ENRLHC'].iloc[row_num] == 1 :
        hc_df['Enrollment.Subtype'].iloc[row_num] = ''

    # Consensus
    if hc_df['CONHC'].iloc[row_num] == 1 and hc_df['CONLRRK2'].iloc[row_num] == 0 and
    ↪  hc_df['CONGBA'].iloc[row_num] == 0 and hc_df['CONSNCA'].iloc[row_num] == 0  :
        hc_df['Consensus.Subtype'].iloc[row_num] = 'Healthy Control'
    if hc_df['CONHC'].iloc[row_num] == 1 and hc_df['CONLRRK2'].iloc[row_num] == 1 and
    ↪  hc_df['CONGBA'].iloc[row_num] == 0 and hc_df['CONSNCA'].iloc[row_num] == 0  :
```

```
        hc_df['Consensus.Subtype'].iloc[row_num] = 'LRRK2'
    if hc_df['CONHC'].iloc[row_num] == 1 and hc_df['CONLRRK2'].iloc[row_num] == 0 and
    ↪  hc_df['CONGBA'].iloc[row_num] == 1 and hc_df['CONSNCA'].iloc[row_num] == 0  :
        hc_df['Consensus.Subtype'].iloc[row_num] = 'GBA'
    if hc_df['CONHC'].iloc[row_num] == 1 and hc_df['CONLRRK2'].iloc[row_num] == 1 and
    ↪  hc_df['CONGBA'].iloc[row_num] == 1 and hc_df['CONSNCA'].iloc[row_num] == 0  :
        hc_df['Consensus.Subtype'].iloc[row_num] = 'LRRK + GBA'
    if hc_df['CONHC'].iloc[row_num] == 1 and hc_df['CONLRRK2'].iloc[row_num] == 0 and
    ↪  hc_df['CONGBA'].iloc[row_num] == 0 and hc_df['CONSNCA'].iloc[row_num] == 1  :
        hc_df['Consensus.Subtype'].iloc[row_num] = 'SNCA'
    if hc_df['CONHC'].iloc[row_num] == 0 :
        hc_df['Consensus.Subtype'].iloc[row_num] = 'non-HC' # FIXME

# Create SWEDD dataframe
swedd_df = pd.read_excel(xlsx, 'SWEDD') # SWEDD subjects
swedd_df = swedd_df.loc[:, ~swedd_df.columns.str.startswith(tuple(cols_delete))]

# Clean up swedd_df dataframe
swedd_df["Comments"].replace({'SWEDD/PD' : '', 'SWEDD/non-PD' : ''}, inplace = True)

# Create new columns called Enrollment.Subtype and Consensus.Subtype
swedd_df['Enrollment.Subtype'] = ''
swedd_df['Consensus.Subtype'] = '' # Create new column called 'Consensus.Subtype' in
↪  swedd_df

# Decode Enrollment.Subtype and Consensus.Subtype to be categorical variables
for row_num in range(len(swedd_df)) :
    # Enrollment
    if swedd_df['ENRLSWEDD'].iloc[row_num] == 1 :
        swedd_df['Enrollment.Subtype'].iloc[row_num] = 'SWEDD Legacy'

    # Consensus
    if swedd_df['CONSWEDD'].iloc[row_num] == 0 :
        swedd_df['Consensus.Subtype'].iloc[row_num] = 'SWEDD/PD Active'
    if swedd_df['CONSWEDD'].iloc[row_num] == 1 :
        swedd_df['Consensus.Subtype'].iloc[row_num] = 'SWEDD/non-PD Active'

# Merge the four 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['PHENOCNV'].replace({0 : 'No', 1 : 'Yes'}, inplace = True)

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

# Decode and re-organize full_df
full_df = merge_columns(full_df, ['CONPD', 'CONPROD', 'CONHC', 'CONSWEDD'],
↪   'Consensus.Diagnosis_temp', ', ') # Get one column for Consensus Diagnosis called
↪   Consensus.Diagnosis_temp
full_df = merge_columns(full_df, ['Consensus.Diagnosis_temp', 'Comments'],
↪   'Consensus.Diagnosis', ': ') # Include the 'Comments' column in with
↪   Consensus.Diagnosis_temp into a new column called 'Consensus.Diagnosis'
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()

### Add in clinical, genetic and T1 data
## Add in age info at each visit - for all subjects in Age_at_visit.csv
age_df = pd.read_csv(ppmi_download_path + 'Age_at_visit.csv', skipinitialspace = True) #
↪   Age info
age_df.rename(columns = {'AGE_AT_VISIT' : 'Age'}, inplace = True) # Rename column
ppmi_merge = pd.merge(full_df, age_df, on = 'PATNO', how = "outer") # Merge full_df with
↪   age_df
# If 'EVENT_ID' column is empty - fill with LOG - for subject 41358 which has no age
↪   information so the df is filled in with EVENT_ID as empty - change this to LOG
ppmi_merge['EVENT_ID'].fillna('LOG', inplace = True) # Replace nan cells with 'LOG' in
↪   EVENT_ID column -- FIXME (not sure if I should fill this with LOG)

## Add birth date, sex, and handedness
bday_df = pd.read_csv(ppmi_download_path + 'Demographics.csv', skipinitialspace = True) #
↪   Demographics info
bday_df = bday_df[['PATNO','EVENT_ID','SEX', 'HANDED', 'BIRTHDT']] # Keep only subject
↪   id, event_id, sex and handedness and birth date
bday_df['SEX'].replace({0 : 'Female', 1 : 'Male' }, inplace = True) # Decode sex
bday_df['HANDED'].replace({1 : 'Right', 2 : 'Left', 3 : 'Mixed' }, inplace = True) #
↪   Decode handedness
bday_df.rename(columns = {'SEX' : 'Sex', 'HANDED' : 'Handed', 'BIRTHDT' : 'BirthDate'},
↪   inplace = True) # Rename columns
```

```
ppmi_merge = pd.merge(ppmi_merge, bday_df, on = ['PATNO','EVENT_ID'], how = "outer") #
↪  Merge ppmi_merge with bday_df (demographics info)

## Visit date, weight and height
vital_df = pd.read_csv(ppmi_download_path + 'Vital_Signs.csv', skipinitialspace=True) #
↪  Visit date, weight, height
vital_df = vital_df[['PATNO','EVENT_ID','INFODT', 'WGTKG', 'HTCM']] # Keep only subject
↪  id, event_id, infodt, weight, height
vital_df.rename(columns = {'WGTKG' : 'Weight(kg)', 'HTCM' : 'Height(cm)'}, inplace =
↪  True) # Rename columns
ppmi_merge = pd.merge(ppmi_merge, vital_df, on = ['PATNO', 'EVENT_ID'], how = "outer" ) #
↪  Merge ppmi_merge with vital_df

## Add in first symptom date, and PD diagnosis date
diag_date_df = pd.read_csv(ppmi_download_path + 'PD_Diagnosis_History.csv') # PD history
diag_date_df = diag_date_df[['PATNO', 'EVENT_ID', 'SXDT','PDDXDT']] # Keep only subject
↪  id, event id, first symptom date, pd diagnosis date
diag_date_df.rename(columns = {'SXDT' : 'First.Symptom.Date', 'PDDXDT':
↪  'PD.Diagnosis.Date'}, inplace = True) # Rename columns
ppmi_merge = pd.merge(ppmi_merge, diag_date_df, on = ['PATNO', 'EVENT_ID'], how =
↪  "outer") # Merge ppmi_merge with diag_date_df

## 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 was the first of
↪  the month, 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
fnl_df = pd.read_csv(ppmi_download_path + 'Conclusion_of_Study_Participation.csv',
↪  skipinitialspace=True) # Age info
fnl_df = fnl_df[['PATNO', 'EVENT_ID', 'COMPLT', 'WDRSN', 'WDDT']] # Keep only subject id,
↪  event id, complt, wdrsn, wdddt
fnl_df['COMPLT'].replace({1.0 : 'Yes', 0 : 'No'}, inplace = True)
fnl_df['WDRSN'].replace({1 : 'Adverse Event', 2.0 : 'Completed study per protocol',3.0 :
↪  'Death', 4.0 : 'Family, care-partner, or social issues', 5.0 : 'Lost to follow up',
↪  6.0 : 'Non-compliance with study procedures' , 7.0 : 'Transportation/Travel issues' ,
↪  8.0 : 'Institutionalized' , 9.0 : 'Subject transitioning to a new cohort' , 10.0 :
↪  'Subject withdrew consent', 11.0 : 'Investigator decision', 12.0 : 'Sponsor
↪  decision', 13.0 : 'Informant/caregiver decision' , 20.0 : 'Other'}, inplace = True)
```

```
fnl_df.rename(columns = {'COMPLT' : 'Completed.Study' , 'WDRSN':
↪  'Reason.for.Withdrawal','WDDT' : 'Withdrawal.Date'}, inplace = True)
ppmi_merge = pd.merge(ppmi_merge, fnl_df, on = ['PATNO', 'EVENT_ID'], how = "outer") #
↪  Merge ppmi_merge with fnl_df

## Diagnosis change
# Get condensed df of event ids and diagnosis of Phenoconverted 1st time
diag_vis1 = full_df[['PATNO', 'DIAG1', 'DIAG1VIS']]
diag_vis1.rename(columns = {'DIAG1VIS' : 'EVENT_ID'}, inplace = True)
diag_vis1.dropna(subset = ['EVENT_ID'], inplace = True)

# Get condensed df of event ids and daignosis of Phenoconverted 2nd time
diag_vis2 = full_df[['PATNO', 'DIAG2', 'DIAG2VIS']]
diag_vis2.rename(columns = {'DIAG2VIS' : 'EVENT_ID'}, inplace = True)
diag_vis2.dropna(subset = ['EVENT_ID'], inplace = True)

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 (DOMSIDE)
domside_df = pd.read_csv(ppmi_download_path +
↪  'PPMI_Original_Cohort_BL_to_Year_5_Dataset_Apr2020.csv', skipinitialspace = True) #
↪  Domside info
domside_df = domside_df[['PATNO', 'EVENT_ID', 'DOMSIDE']] # Keep only subject id, event
↪  id, and DOMSIDE
domside_df['DOMSIDE'].replace({1 : 'Left', 2 : 'Right', 3 : 'Symmetric'}, inplace = True)
↪  # Decode
ppmi_merge = pd.merge(ppmi_merge, domside_df, on = ['PATNO', 'EVENT_ID'], how = "outer" )
↪  # Merge ppmi_merge with domside_df

## Motor Symptoms
motor_df = pd.read_csv(ppmi_download_path +
↪  'Participant_Motor_Function_Questionnaire.csv', skipinitialspace = True) # Motor
↪  skills info
motor_df.drop(['REC_ID', 'INFODT', 'ORIG_ENTRY', 'LAST_UPDATE'], axis = 1, inplace =
↪  True) # Drop these columns from df
motor_df['CMPLBY2'].replace({1: 'Participant', 2 : 'Caregiver' , 3 : 'Participant and
↪  Caregivier'}, inplace = True)
motor_df['PAG_NAME'].replace({'PQUEST' : 'Participant Motor Function Questionnaire'},
↪  inplace = True)
motor_df['TRBUPCHR'].replace({0 : 'No' , 1 : 'Yes', 2 : 'Uncertain'}, inplace = True)
motor_df['WRTSMLR'].replace({0 : 'No' , 1 : 'Yes', 2 : 'Uncertain'}, inplace = True)
motor_df['VOICSFTR'].replace({0 : 'No' , 1 : 'Yes', 2 : 'Uncertain'}, inplace = True)
```

```
motor_df['POORBAL'].replace({0 : 'No' , 1 : 'Yes', 2 : 'Uncertain'}, inplace = True)
motor_df['FTSTUCK'].replace({0 : 'No' , 1 : 'Yes', 2 : 'Uncertain'}, inplace = True)
motor_df['LSSXPRSS'].replace({0 : 'No' , 1 : 'Yes', 2 : 'Uncertain'}, inplace = True)
motor_df['ARMLGSHK'].replace({0 : 'No' , 1 : 'Yes', 2 : 'Uncertain'}, inplace = True)
motor_df['TRBBUTTN' ].replace({0 : 'No' , 1 : 'Yes', 2 : 'Uncertain'}, inplace = True)
motor_df['SHUFFLE'].replace({0 : 'No' , 1 : 'Yes', 2 : 'Uncertain'}, inplace = True)
motor_df['MVSLOW'].replace({0 : 'No' , 1 : 'Yes', 2 : 'Uncertain'}, inplace = True)
motor_df['TOLDPD'].replace({0 : 'No' , 1 : 'Yes', 2 : 'Uncertain'}, inplace = True)
motor_df = motor_df.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' })
ppmi_merge = pd.merge(ppmi_merge, motor_df, on = ['PATNO', 'EVENT_ID'], how = "outer" )

## Cognitive symptoms - Cognitive Categorization
cog_df = pd.read_csv(ppmi_download_path + 'Cognitive_Categorization.csv',
↪  skipinitialspace = True) # Cognitive info
cog_df = cog_df[['PATNO' , 'EVENT_ID', 'PAG_NAME', 'COGDECLN', 'FNCDTCOG' , 'COGDXCL'
↪  ,'PTCGBOTH' , 'COGSTATE' , 'COGCAT_TEXT']] # Keep only
cog_df['COGDECLN'].replace({0 : 'No', 1 : 'Yes'}, inplace = True)
cog_df['FNCDTCOG'].replace({0 : 'No', 1 : 'Yes'}, inplace = True)
cog_df['COGDXCL'].replace({1 : '90 - 100%', 2 : '50 - 89%', 3 : '10 - 49%', 4 : '0 -
↪  9%'}, inplace = True)
cog_df['PTCGBOTH'].replace({1 : 'Participant', 2 : 'Caregiver', 3 : 'Participant and
↪  Caregiver'}, inplace = True)
cog_df['COGSTATE'].replace({1 : 'Normal Condition', 2 : 'Mild Cognitive Impairment', 3 :
↪  'Dimentia'}, inplace = True)
cog_df = cog_df.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'})
cog_df['Cognitive.Page.Name'].replace({'COGCATG' : 'Cognitive Categorization'}, inplace =
↪  True) # Rename
ppmi_merge = pd.merge(ppmi_merge, cog_df, on = ['PATNO', 'EVENT_ID'], how = "outer" ) #
↪  Merge

## Cognitive symptoms - MOCA
moca_df = pd.read_csv(ppmi_download_path +'Montreal_Cognitive_Assessment__MoCA_.csv',
↪  skipinitialspace = True) # Montreal Cognitive Assessment info
moca_df = moca_df[['PATNO','EVENT_ID', 'MCATOT']] # Keep only
moca_df.rename(columns = {'MCATOT' : 'MOCA.Total'}, inplace = True) # Rename
ppmi_merge = pd.merge(ppmi_merge, moca_df, on = ['PATNO','EVENT_ID'], how = "outer" ) #
↪  Merge

## Medication Status - Concomitant Med Log
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 ','
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
for index in med_df['Start_Stop'].index :
    med_df['Start_Stop'].iloc[index] = '(' + med_df['Start_Stop'][index] + ')' # 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
LEDD_med_df = pd.read_csv(ppmi_download_path + 'LEDD_Concomitant_Medication_Log.csv', skipinitialspace=True) # Medication history
LEDD_med_df.replace({';' : ','}, regex = True, inplace = True) # Replace all ';' in LEDD_med_df with ',' so that we can use ';' as a separator in next few steps
LEDD_med_df['LEDTRT'] = LEDD_med_df['LEDTRT'].str.title() # Capitalize all medication names
LEDD_med_df['STARTDT'].fillna('NA', inplace = True) # Fillna
LEDD_med_df['STOPDT'].fillna('NA', inplace = True) # Fillna
LEDD_med_df['LEDD'].fillna('NA', inplace = True) # Fillna
LEDD_med_df = LEDD_med_df.astype({"STARTDT" : 'str', "STOPDT" : 'str', "LEDD" : 'str'}) # Change start and stop date to strings
LEDD_med_df = merge_columns(LEDD_med_df, ['LEDTRT', 'LEDDOSSTR'], 'LEDD_med_and_dose', ' ')
LEDD_med_df = merge_columns(LEDD_med_df, ['STARTDT', 'STOPDT'], 'Start_Stop', '-')
for row_num in range(len(LEDD_med_df['Start_Stop'])) :
    LEDD_med_df['Start_Stop'].iloc[row_num] = '(' + LEDD_med_df['Start_Stop'][row_num] + ')' # Put parenthesis around dates so when you merge it with LED_med_and dose it is more organized
    LEDD_med_df['LEDD'].iloc[row_num] = '[' + LEDD_med_df['LEDD'][row_num] + ']' # Put brackets around LEDD DOSE so when you merge it with LED_med_and dose it is more organized
LEDD_med_df = merge_columns(LEDD_med_df, ['LEDD_med_and_dose', 'LEDD'], 'LEDD.Medication[LEDD]', ' ')
LEDD_med_df = merge_columns(LEDD_med_df, ['LEDD.Medication[LEDD]', 'Start_Stop'], 'LEDD.Medication[LEDD](Dates)', ' ')
LEDD_med_df = LEDD_med_df.groupby(['PATNO','EVENT_ID'])['LEDD.Medication[LEDD](Dates)'].apply('; '.join)
ppmi_merge = pd.merge(ppmi_merge, LEDD_med_df, on = ['PATNO', 'EVENT_ID'], how = "outer")

## Comorbidities
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
for row_num in range(len(comorbid_df['MHDIAGDT'])) :
    comorbid_df['MHDIAGDT'].iloc[row_num] = '(' + comorbid_df['MHDIAGDT'][row_num] + ')'
↪  # 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)
# FIXME make integer
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
ppmi_merge['Analytic.Cohort'] = '' # Initialize Analytic.Cohort col
for row_num in range(len(ppmi_merge['Analytic.Cohort'])) :
    if ppmi_merge['PATNO'].iloc[row_num] in analytic_cohort_subids :
        ppmi_merge['Analytic.Cohort'].iloc[row_num] = 'Analytic Cohort'
    else :
        ppmi_merge['Analytic.Cohort'].iloc[row_num] = 'Not Analytic Cohort'

## 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', 'Motor.Function.Page.Name',
↪  'Motor.Function.Source',    'Trouble.Rising.Chair', 'Writing.Smaller',
↪  'Voice.Softer', 'Poor.Balance', 'Feet.Stuck',   'Less.Expressive',
↪  'Arms/Legs.Shake',  'Trouble.Buttons',  'Shuffle.Feet', 'Slow.Movements',
↪  'Been.Told.PD', 'Cognitive.Page.Name',  'Cognitive.Source', 'Cognitive.Decline',
↪  'Functional.Cognitive.Impairment',  'Confidence.Level.Cognitive.Diagnosis',
↪  'Cognitive.State',  'Cognitive.Tscore.Cat', 'MOCA.Total',   'Medication',
↪  'LEDD.Medication[LEDD](Dates)',  'Medical.History.Description(Diagnosis.Date)'])

## Modified Boston Naming test - BAR edit
boston_naming_df = pd.read_csv(ppmi_download_path + 'Modified_Boston_Naming_Test.csv')
boston_naming_df = boston_naming_df[['PATNO', 'EVENT_ID', 'MBSTNSCR', 'MBSTNCRC',
↪  'MBSTNCRR', 'MBSTNVRS']]
ppmi_merge = pd.merge(ppmi_merge, boston_naming_df, on = ['PATNO', 'EVENT_ID'], how =
↪  "outer") # Merge
```

```
## Clock Drawing total - BAR edit
clock_drawing_df = pd.read_csv(ppmi_download_path + 'Clock_Drawing.csv')
clock_drawing_df = clock_drawing_df[['PATNO', 'EVENT_ID', 'CLCKTOT']]
clock_drawing_df.rename(columns = {'CLCKTOT' : 'Clock.Drawing.Total'}, inplace = True) #
↪  Rename
ppmi_merge = pd.merge(ppmi_merge, clock_drawing_df, on = ['PATNO', 'EVENT_ID'], how =
↪  "outer")

## Benton Judgement of Line Orientation
benton_df = pd.read_csv(ppmi_download_path + 'Benton_Judgement_of_Line_Orientation.csv')
benton_df = benton_df[['PATNO', 'EVENT_ID', 'JLO_TOTCALC']]
benton_df.rename(columns = {'JLOTOTCALC' : 'JOLO.Total'}, inplace = True)
ppmi_merge = pd.merge(ppmi_merge, benton_df, on = ['PATNO', 'EVENT_ID'], how = "outer")

## Letter Number Sequencing
letter_number_df = pd.read_csv(ppmi_download_path + 'Letter_-_Number_Sequencing.csv')
letter_number_df = letter_number_df[['PATNO', 'EVENT_ID', 'LNS_TOTRAW']]
letter_number_df.rename(columns = {'LNS_TOTRAW' : 'Letter.Number.Sequencing.Total'},
↪  inplace = True)
ppmi_merge = pd.merge(ppmi_merge, letter_number_df, on = ['PATNO', 'EVENT_ID'], how =
↪  "outer")

## Modified Semantic Fluency
semantic_fluency = pd.read_csv(ppmi_download_path + 'Modified_Semantic_Fluency.csv')
semantic_fluency = semantic_fluency[['PATNO', 'EVENT_ID', 'DVS_SFTANIM']]
semantic_fluency.rename(columns = {'DVS_SFTANIM' : 'Semantic.Fluency.Total'}, inplace =
↪  True)
ppmi_merge = pd.merge(ppmi_merge, semantic_fluency, on = ['PATNO', 'EVENT_ID'], how =
↪  "outer")

## Hopkin's Verbal Learning
hopkins_df = pd.read_csv(ppmi_download_path +
↪  'Hopkins_Verbal_Learning_Test_-_Revised.csv')
hopkins_df = hopkins_df[['PATNO', 'EVENT_ID', 'DVT_DELAYED_RECALL', 'DVT_TOTAL_RECALL']]
hopkins_df.rename(columns = {'DVT_TOTAL_RECALL' : 'DVT.Total.RECALL'}, inplace = True)
ppmi_merge = pd.merge(ppmi_merge, hopkins_df, on = ['PATNO', 'EVENT_ID'], how = "outer")

## Symbol Digit Modalities Test
symbol_df = pd.read_csv(ppmi_download_path + 'Symbol_Digit_Modalities_Test.csv')
symbol_df = symbol_df[['PATNO', 'EVENT_ID', 'SDMTOTAL']]
symbol_df.rename(columns = {'SDMTOTAL' : 'Symbol.Digit.Modalities.Total'}, inplace =
↪  True)
ppmi_merge = pd.merge(ppmi_merge, symbol_df, on = ['PATNO', 'EVENT_ID'], how = "outer")

## REM Sleeep behavior disorder questionnaire
rem_df = pd.read_csv(ppmi_download_path +
↪  'REM_Sleep_Behavior_Disorder_Questionnaire.csv')
rem_df.drop(['REC_ID','INFODT','ORIG_ENTRY', 'LAST_UPDATE'], axis = 1, inplace = True)
rem_df.rename(columns = {'PAG_NAME' : 'REM.Sleep.Behavior.Disorder.Page.Name'}, inplace =
↪  True)
rem_df['REM.Sleep.Behavior.Disorder.Page.Name'].replace({'REMSLEEP' : 'REM Sleep Behavior
↪  Disorder Questionnaire'}, inplace = True)
rem_df['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']
rem_df['RBDTotal.REM'] = rem_df[rem_list].sum(axis = 1) # Add an RBDTotal.REM column (per
↪  BA edits)
for rem_item in rem_list :
  rem_df[rem_item].replace({0 : 'No', 1 : 'Yes'}, inplace = True)
rem_df.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)
ppmi_merge = pd.merge(ppmi_merge, rem_df, on = ['PATNO','EVENT_ID'], how = "outer")

## Sort ppmi_merge by PATNO and INFODT
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

#### IMAGING INFO ####
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['MRIWDTI'].replace({0 : 'No', 1 : 'Yes'}, inplace = True) # Decode
mri_df['MRIWRSS'].replace({0 : 'No', 1 : 'Yes'}, inplace = True) # Decode
mri_df['MRIRSSDF'].replace({0 : 'No', 1 : 'Yes'}, inplace = True) # Decode
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 - do we want to get rid of these?
# 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 Taylor's 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 ####

# UPDRS Part 1
updrs_part1_df = pd.read_csv(ppmi_download_path + 'MDS-UPDRS_Part_I.csv',
↪  skipinitialspace = True)
updrs_part1_df.drop(['ORIG_ENTRY','LAST_UPDATE','REC_ID'], axis = 1, inplace = True)
updrs_part1_df.rename(columns = {'PAG_NAME' : 'UPDRS.Part1.Page.Name', 'NUPSOURC' :
↪  'UPDRS.Part1.Source'}, inplace = True)
updrs_part1_df['UPDRS.Part1.Page.Name'].replace({'NUPDRS1' : 'MDS-UPDRS Part I: Non-Motor
↪  Aspects of Experiences of Daily Living'}, inplace = True)

# UPDRS Part 1 Patient Questionnaire
updrs_part1_pq_df = pd.read_csv(ppmi_download_path +
↪  'MDS-UPDRS_Part_I_Patient_Questionnaire.csv', skipinitialspace = True)
updrs_part1_pq_df.drop(['ORIG_ENTRY','LAST_UPDATE','REC_ID','INFODT'], axis = 1, inplace
↪  = True)
updrs_part1_pq_df.rename(columns = {'PAG_NAME' :
↪  'UPDRS.Part1.Patient.Questionnaire.Page.Name', 'NUPSOURC' : 'UPDRS.Part1.PQ.Source'},
↪  inplace = True)
updrs_part1_pq_df['UPDRS.Part1.Patient.Questionnaire.Page.Name'].replace({'NUPDRS1P' :
↪  'MDS-UPDRS Part I Patient Questionnaire: Non-Motor Aspects of Experiences of Daily
↪  Living', 'NUPDRSP' : 'MDS-UPDRS Part IB and Part II'}, inplace = True)

# UPDRS Part 2
updrs_part2_pq_df = pd.read_csv(ppmi_download_path +
↪  'MDS_UPDRS_Part_II__Patient_Questionnaire.csv', skipinitialspace = True)
updrs_part2_pq_df.drop(['ORIG_ENTRY','LAST_UPDATE','REC_ID','INFODT'], axis = 1, inplace
↪  = True)
updrs_part2_pq_df.rename(columns = {'PAG_NAME' : 'UPDRS.Part2.Page.Name','NUPSOURC' :
↪  'UPDRS.Part2.Source'}, inplace = True)
updrs_part2_pq_df['UPDRS.Part2.Page.Name'].replace({'NUPDRS2P' : 'MDS-UPDRS Part II
↪  Patient Questionnaire: Motor Aspects of Experiences of Daily Living','NUPDRSP' :
↪  'MDS-UPDRS Part IB and Part II'}, inplace = True)

# 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']

# UPDRS Part 3 on/off determination dosing
updrs_part3_dos_df = pd.read_csv(ppmi_download_path +
↪  'MDS-UPDRS_Part_III_ON_OFF_Determination___Dosing.csv', skipinitialspace = True)
updrs_part3_dos_df.drop(['ORIG_ENTRY','LAST_UPDATE','REC_ID','INFODT'], axis = 1, inplace
↪  = True)
updrs_part3_dos_df.rename(columns = {'PAG_NAME' : 'UPDRS.Part3.Dosage.Page.Name'},
↪  inplace = True)
updrs_part3_dos_df['UPDRS.Part3.Dosage.Page.Name'].replace({'NUPDRDOSE' : 'MDS-UPDRS Part
↪  III ON/OFF Determination & Dosing', 'NUPDRDOSER' : 'MDS-UPDRS Part III examination
↪  administered at remote visit'}, inplace = True)

# UPDRS Part 4
updrs_part4_motor_df = pd.read_csv(ppmi_download_path +
↪  'MDS-UPDRS_Part_IV__Motor_Complications.csv', skipinitialspace = True)
updrs_part4_motor_df.drop(['ORIG_ENTRY','LAST_UPDATE','REC_ID','INFODT'], axis = 1,
↪  inplace = True)
updrs_part4_motor_df.rename(columns = {'PAG_NAME' : 'UPDRS.Part4.Page.Name'}, inplace =
↪  True)
updrs_part4_motor_df['UPDRS.Part4.Page.Name'].replace({'NUPDRS4' : 'MDS-UPDRS Part IV:
↪  Motor Complications'}, inplace = True)

# 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
```

```
updrs_list_str = ['.UPDRS1','.UPDRS1','.UPDRS2', '.UPDRS3','.UPDRS3A', '.UPDR3OF',
↪  '.UPDR3ON','.UPDRDOSE','.UPDRS4'] # extensions
count = 0
for df in updrs_list :
    for col_name in df :
        if col_name != 'PATNO' and col_name != 'EVENT_ID' and col_name != 'INFODT':
            df.rename(columns = {str(col_name) : str(col_name) +
↪  (updrs_list_str[count])}, inplace = True) # Add extension
    count += 1

# 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' :
```

```
## Add Brady-Rigidity, Tremor and PIGD Subscores
# Brady-Rigidity Subscore
brady_only = updrs_numeric[['Rigidity.Neck.UPDRS3', 'Rigidity.RUE.UPDRS3',
↪  'Rigidity.LUE.UPDRS3', 'Rigidity.RLE.UPDRS3',
↪  'Rigidity.LLE.UPDRS3','Rigidity.Neck.UPDRS3A','Rigidity.RUE.UPDRS3A','Rigidity.LUE.UPDRS3A',
↪  'Rigidity.RLE.UPDRS3A', 'Rigidity.LLE.UPDRS3A', 'Rigidity.Neck.UPDR3ON'
↪  ,'Rigidity.RUE.UPDR3ON', 'Rigidity.LUE.UPDR3ON',
↪  'Rigidity.RLE.UPDR3ON','Rigidity.LLE.UPDR3ON', 'Rigidity.Neck.UPDR3OF',
↪  'Rigidity.RUE.UPDR3OF', 'Rigidity.LUE.UPDR3OF', 'Rigidity.RLE.UPDR3OF',
↪  'Rigidity.LLE.UPDR3OF','Finger.Tapping.Right.Hand.UPDRS3'
↪  ,'Finger.Tapping.Left.Hand.UPDRS3' ,'Finger.Tapping.Right.Hand.UPDRS3A'
↪  ,'Finger.Tapping.Left.Hand.UPDRS3A' ,'Finger.Tapping.Right.Hand.UPDR3ON'
↪  ,'Finger.Tapping.Left.Hand.UPDR3ON' ,'Finger.Tapping.Right.Hand.UPDR3OF'
↪  ,'Finger.Tapping.Left.Hand.UPDR3OF' ,
↪  'Hand.Movements.Right.Hand.UPDRS3','Hand.Movements.Left.Hand.UPDRS3','Hand.Movements.Right.Hand.UPDR
↪  'Hand.Movements.Left.Hand.UPDRS3A','Hand.Movements.Right.Hand.UPDR3ON','Hand.Movements.Left.Hand.UPD
↪  'Pronation.Supination.Right.Hand.UPDRS3', 'Pronation.Supination.Left.Hand.UPDRS3',
↪  'Pronation.Supination.Right.Hand.UPDRS3A' , 'Pronation.Supination.Left.Hand.UPDRS3A'
↪  , 'Pronation.Supination.Right.Hand.UPDR3ON',
↪  'Pronation.Supination.Left.Hand.UPDR3ON', 'Pronation.Supination.Right.Hand.UPDR3OF',
↪  'Pronation.Supination.Left.Hand.UPDR3OF', 'Toe.Tapping.Right.Foot.UPDRS3',
↪  'Toe.Tapping.Left.Foot.UPDRS3', 'Toe.Tapping.Right.Foot.UPDRS3A' ,
↪  'Toe.Tapping.Left.Foot.UPDRS3A', 'Toe.Tapping.Right.Foot.UPDRS3A' ,
↪  'Toe.Tapping.Left.Foot.UPDR3ON', 'Toe.Tapping.Right.Foot.UPDRS3',
↪  'Toe.Tapping.Left.Foot.UPDR3ON','Leg.Agility.Right.Leg.UPDRS3',
↪  'Leg.Agility.Right.Leg.UPDRS3A', 'Leg.Agility.Right.Leg.UPDR3ON',
↪  'Leg.Agility.Right.Leg.UPDR3OF', 'Leg.Agility.Left.Leg.UPDRS3',
↪  'Leg.Agility.Left.Leg.UPDRS3A', 'Leg.Agility.Left.Leg.UPDR3ON',
↪  'Leg.Agility.Left.Leg.UPDR3OF']]
updrs_numeric['Brady.Rigidity.Subscore'] = 0
idx = brady_only.index[brady_only.isnull().all(1)]
updrs_numeric['Brady.Rigidity.Subscore'] = brady_only.sum(axis=1)
updrs_numeric['Brady.Rigidity.Subscore'].iloc[idx] = np.nan

# Tremor Subscore
tremor_only = updrs_numeric[['Tremor.UPDRS2', 'Postural.Tremor.Right.Hand.UPDRS3'
↪  ,'Postural.Tremor.Left.Hand.UPDRS3' ,'Postural.Tremor.Right.Hand.UPDRS3A'
↪  ,'Postural.Tremor.Left.Hand.UPDRS3A' ,'Postural.Tremor.Right.Hand.UPDR3OF'
↪  ,'Postural.Tremor.Left.Hand.UPDR3OF' ,'Postural.Tremor.Right.Hand.UPDR3ON'
↪  ,'Postural.Tremor.Left.Hand.UPDR3ON'
↪  ,'Kinetic.Tremor.Right.Hand.UPDRS3','Kinetic.Tremor.Left.Hand.UPDRS3',
↪  'Kinetic.Tremor.Right.Hand.UPDRS3A',
↪  'Kinetic.Tremor.Left.Hand.UPDRS3A','Kinetic.Tremor.Right.Hand.UPDR3OF','Kinetic.Tremor.Left.Hand.UPD
↪  'Kinetic.Tremor.Right.Hand.UPDR3ON','Kinetic.Tremor.Left.Hand.UPDR3ON',
↪  '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', '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',
↪  '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', '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.UPDRS3', 'Constancy.of.Rest.Tremor.UPDRS3A',
↪  'Constancy.of.Rest.Tremor.UPDR3OF', 'Constancy.of.Rest.Tremor.UPDR3ON']]
```

```
updrs_numeric['Tremor.Subscore'] = 0
idx = tremor_only.index[tremor_only.isnull().all(1)]
updrs_numeric['Tremor.Subscore'] = tremor_only.sum(axis=1)
updrs_numeric['Tremor.Subscore'].iloc[idx] = np.nan

# PIGD subscore
pigd_only = updrs_numeric[['Walking.Difficulty.UPDRS2' ,  'Freezing.while.Walking.UPDRS2'
↪  ,'Gait.Problems.UPDRS3' ,'Gait.Problems.UPDRS3A' ,'Gait.Problems.UPDR3ON' ,
↪  'Gait.Problems.UPDR3OF' , 'Freezing.of.Gait.UPDRS3' ,'Freezing.of.Gait.UPDRS3A' ,
↪  'Freezing.of.Gait.UPDR3OF' , 'Freezing.of.Gait.UPDR3ON' ,
↪  'Postural.Stability.Problems.UPDRS3', 'Postural.Stability.Problems.UPDRS3A',
↪  'Postural.Stability.Problems.UPDR3OF', 'Postural.Stability.Problems.UPDR3ON']]
updrs_numeric['PIGD.Subscore'] = 0
idx = pigd_only.index[pigd_only.isnull().all(1)]
updrs_numeric['PIGD.Subscore'] = pigd_only.sum(axis=1)
updrs_numeric['PIGD.Subscore'].iloc[idx] = np.nan

# Add .Num (Numeric) extension to these numeric variables
for col_name in updrs_numeric:
    if col_name != 'PATNO' and col_name != 'EVENT_ID' and col_name != 'PIGD.Subscore' and
    ↪  col_name != 'Tremor.Subscore' and col_name != 'Brady.Rigidity.Subscore' :
        updrs_numeric.rename(columns = {str(col_name) : str(col_name) + '.Num'}, inplace
↪  = True) # Add .Num extension

### 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']
nupdrs3_colname_list = ['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']

# Decode, rename columns and add extension for updrs3 dfs in updrs3_df_list
count = 0
for df in [nupdrs3_copy, nupdrs3A_copy, nupdr3ON_copy, nupdr3OF_copy] :
    for col_name in nupdrs3_colname_list :
        df[col_name].replace({0 : 'None', 1 : 'Slight', 2: 'Mild', 3 : 'Moderate', 4 :
↪  'Severe'}, inplace = True)
    df['DBS_STATUS'].replace({0 : 'OFF', 1 : 'ON'}, inplace = True)
    df['DYSKPRES'].replace({0 : 'No', 1: 'Yes'}, inplace = True)
    df['DYSKIRAT'].replace({0: 'No', 1 : 'Yes'}, inplace = True)
    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)

    # Rename columns
    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' :
```

```
    for col in df :
        if col != 'PATNO' and col != 'EVENT_ID' and col != 'INFODT' :
            df.rename(columns = {str(col) : str(col) + ext_list[count]}, inplace = True)
↪  # Add extension
    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)
np1_list = ['NP1COG', 'NP1HALL', 'NP1DPRS', 'NP1ANXS', 'NP1APAT', 'NP1DDS']
for col in np1_list :
    updrs_cat[col].replace({0 : 'None', 1 : 'Slight', 2: 'Mild', 3 : 'Moderate', 4 :
↪  'Severe'}, inplace = True)
```

```
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)
np1_pq_list= ['NP1SLPN', 'NP1SLPD', 'NP1PAIN', 'NP1URIN', 'NP1CNST', 'NP1LTHD',
↪  'NP1FATG']
for col in np1_pq_list :
    updrs_cat[col].replace({0 : 'None', 1 : 'Slight', 2: 'Mild', 3 : 'Moderate', 4 :
↪  'Severe'}, inplace = True)
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({1.0 : 'Patient', 2.0 : 'Caregiver', 3.0 :
↪  'Patient and Caregiver'}, inplace = True)
np2_list =
↪  ['NP2SPCH','NP2SALV','NP2SWAL','NP2EAT','NP2DRES','NP2HYGN','NP2HWRT','NP2HOBB','NP2TURN','NP2TRMR'
for col in np2_list :
    updrs_cat[col].replace({0.0 : 'None', 1.0 : 'Slight', 2.0: 'Mild', 3.0 : 'Moderate',
↪  4.0 : 'Severe'}, inplace = True)
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['RMEXAM'].replace({0.0 : 'No', 1.0 : 'Yes'}, inplace = True)
updrs_cat['DBSYN'].replace({0.0 : 'No', 1.0 : 'Yes'}, inplace = True)
updrs_cat['OFFEXAM'].replace({0.0 : 'No', 1.0 : 'Yes'}, inplace = True)
updrs_cat['OFFEXAM'].replace({0.0 : 'No', 1.0 : 'Yes'}, inplace = True)
updrs_cat['ONEXAM'].replace({0.0 : 'No', 1.0 : 'Yes'}, inplace = True)
updrs_cat['RMTOFFRSN'].replace({1.0 : 'ON state not reached', 2.0 : 'Scheduling issues',
↪  3.0 : 'Other reason'}, inplace = True)
updrs_cat['ONOFFORDER'].replace({1.0 : 'OFF', 2.0 : 'ON'}, inplace = True)
updrs_cat['RMONOFF'].replace({1.0 : 'OFF', 2.0 : 'ON'}, inplace = True)
updrs_cat['ONNORSN'].replace({1.0 : 'ON state not reached', 2.0 : 'Scheduling issues',
↪  3.0 : 'Other reason'}, inplace = True)
```

```
updrs_cat['PDMEDYN' ].replace({0 : False , 1.0 : 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({1.0 : 'Disease severity preventing turning off of DBS',
↪  2.0 : 'Did not bring medication to turn on' , 3.0 : 'Forgot to refrain from taking
↪  medication', 4.0 : 'Does not want to turn off DBS', 5.0 : 'Forgot to turn off DBS',
↪  6.0 : 'Forgot to bring DBS', 7.0 : 'Unsure if participant was full off', 8.0 : 'Site
↪  scheduling issues', 9.0 : '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' :
↪  'Reason.OFF.Exam.Not.Performed.UPDRDOSE' , 'OFFEXAM' : 'OFF.Exam.Performed.UPDRDOSE',
↪  'OFFEXAMTM' : 'OFF.Exam.Time.UPDRDOSE', 'DBSYN' : 'Has.DBS.UPDRDOSE', 'HRPOSTMED' :
↪  'Hours.btwn.PD.Med.and.UPDRS3.Exam.UPDRDOSE', 'HRDBSOFF' :
↪  'Hours.btwn.DBS.Device.Off.and.UPDRS3.Exam.UPDRDOSE', 'HRDBSON' :
↪  'Hours.btwn.DBS.Device.On.and.UPDRS3.Exam.UPDRDOSE' , 'RMEXAM' :
↪  'Remote.UPDRS3.Exam.UPDRDOSE'}, inplace = True)

# 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'}, inplace = True)
```

```
updrs_cat.rename(columns = {'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'
↪  }, inplace = True)
updrs_cat.rename(columns = {'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)

# Add .Cat extensino to updrs_cat col names
for col in updrs_cat:
    if col != 'PATNO' and col != 'EVENT_ID' and col != 'INFODT':
        updrs_cat.rename(columns = {str(col) : str(col) + '.Cat'}, inplace = True)

# 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)
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)

# 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'] # 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)

#### 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_list1 = ['Dominant.Side.Disease','Rigidity.Neck.UPDRS3', 'Rigidity.RUE.UPDRS3',
↪  'Rigidity.LUE.UPDRS3', 'Rigidity.RLE.UPDRS3',
↪  'Rigidity.LLE.UPDRS3','Rigidity.Neck.UPDRS3A','Rigidity.RUE.UPDRS3A','Rigidity.LUE.UPDRS3A',
↪  'Rigidity.RLE.UPDRS3A', 'Rigidity.LLE.UPDRS3A', 'Rigidity.Neck.UPDR3ON'
↪  ,'Rigidity.RUE.UPDR3ON', 'Rigidity.LUE.UPDR3ON',
↪  'Rigidity.RLE.UPDR3ON','Rigidity.LLE.UPDR3ON', 'Rigidity.Neck.UPDR3OF',
↪  'Rigidity.RUE.UPDR3OF', 'Rigidity.LUE.UPDR3OF', 'Rigidity.RLE.UPDR3OF',
↪  'Rigidity.LLE.UPDR3OF','Finger.Tapping.Right.Hand.UPDRS3'
↪  ,'Finger.Tapping.Left.Hand.UPDRS3' ,'Finger.Tapping.Right.Hand.UPDRS3A'
↪  ,'Finger.Tapping.Left.Hand.UPDRS3A' ,'Finger.Tapping.Right.Hand.UPDR3ON'
↪  ,'Finger.Tapping.Left.Hand.UPDR3ON' ,'Finger.Tapping.Right.Hand.UPDR3OF'
↪  ,'Finger.Tapping.Left.Hand.UPDR3OF' ,
↪  'Hand.Movements.Right.Hand.UPDRS3','Hand.Movements.Left.Hand.UPDRS3','Hand.Movements.Right.Hand.UPDI
↪  'Hand.Movements.Left.Hand.UPDRS3A','Hand.Movements.Right.Hand.UPDR3ON','Hand.Movements.Left.Hand.UPI
↪  'Pronation.Supination.Right.Hand.UPDRS3', 'Pronation.Supination.Left.Hand.UPDRS3',
↪  'Pronation.Supination.Right.Hand.UPDRS3A' , 'Pronation.Supination.Left.Hand.UPDRS3A'
↪  , 'Pronation.Supination.Right.Hand.UPDR3ON',
↪  'Pronation.Supination.Left.Hand.UPDR3ON', 'Pronation.Supination.Right.Hand.UPDR3OF',
↪  'Pronation.Supination.Left.Hand.UPDR3OF', 'Toe.Tapping.Right.Foot.UPDRS3',
↪  'Toe.Tapping.Left.Foot.UPDRS3', 'Toe.Tapping.Right.Foot.UPDRS3A' ,
↪  'Toe.Tapping.Left.Foot.UPDRS3A', 'Toe.Tapping.Right.Foot.UPDR3OF' ,
↪  'Toe.Tapping.Left.Foot.UPDR3OF', 'Toe.Tapping.Right.Foot.UPDR3ON',
↪  'Toe.Tapping.Left.Foot.UPDR3ON','Leg.Agility.Right.Leg.UPDRS3',
↪  'Leg.Agility.Right.Leg.UPDRS3A', 'Leg.Agility.Right.Leg.UPDR3ON',
↪  'Leg.Agility.Right.Leg.UPDR3OF', 'Leg.Agility.Left.Leg.UPDRS3',
↪  'Leg.Agility.Left.Leg.UPDRS3A', 'Leg.Agility.Left.Leg.UPDR3ON',
↪  'Leg.Agility.Left.Leg.UPDR3OF']

# Make brady_list1 ready for ppmi_merge
new_list = []
for i in range(len(brady_list1)) :
    if brady_list1[i].startswith('Dominant') :
        new_list.append(brady_list1[i])
        continue
    else :
        temp = brady_list1[i].replace(brady_list1[i], brady_list1[i]+'.Num')
        new_list.append(temp)

# Create three original lists to be used below
brady_right = new_list.copy()
brady_left = new_list.copy()
brady_sym = new_list.copy()

# Get list of only right side scores and symmetric scores for brady subscore
for i in brady_right:
    if 'Left' in i or 'LUE' in i or 'LLE' in i :
        brady_right.remove(i)

# Get list of only left side scores and symmetric scores for brady subscore
for i in brady_left :
    if 'Right' in i or 'RUE' in i or 'RLE' in i :
```

```
        brady_left.remove(i)

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.index[subscore_side_temp.isnull().all(1)] # Get idx where
↪   all column values of the same row are nan - this is because these will fill in as 0
↪   when in reality they aren't zero they should be nans
    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

ppmi_merge = add_lateralized_subscores(ppmi_merge, brady_left, 'Left',
↪  'Brady.Rigidity.Subscore-left')
ppmi_merge = add_lateralized_subscores(ppmi_merge, brady_right, 'Right',
↪  'Brady.Rigidity.Subscore-right')

## BRADY SYM
brady_only_sym = ppmi_merge[brady_sym] # Get dataframe of only columns in brady sym
ppmi_merge['Brady.Rigidity.Subscore-sym'] = 0 # Initialize lateralized variable
brady_only_sym.loc[brady_only_sym['Dominant.Side.Disease'] != 'Symmetric', : ] = np.nan #
↪  Make all rows nan if dominant side of disease is not symmetric
brady_sym_temp= brady_only_sym.drop('Dominant.Side.Disease', 1) # Drop dominant side of
↪  disease - necessary because this cannot be summed in next line
idx = brady_sym_temp.index[brady_sym_temp.isnull().all(1)] # Get idx of where all column
↪  values of the same row are nan - this is because these will fill in as 0 when in
↪  reality they aren't zero they should be nans
x = brady_sym_temp.fillna(0) # because adding 1 plus nan equals nan
ppmi_merge['Brady.Rigidity.Subscore-sym'] = x['Rigidity.Neck.UPDRS3.Num'] +
↪  (x['Rigidity.RUE.UPDRS3.Num'] + x['Rigidity.LUE.UPDRS3.Num'])/2 +
↪  (x['Rigidity.RLE.UPDRS3.Num'] + x['Rigidity.LLE.UPDRS3.Num'])/2  +
↪  x['Rigidity.Neck.UPDRS3A.Num']+ (x['Rigidity.RUE.UPDRS3A.Num']+
↪  x['Rigidity.LUE.UPDRS3A.Num'])/2 + (x['Rigidity.RLE.UPDRS3A.Num']
↪  +x['Rigidity.LLE.UPDRS3A.Num'])/2 + x['Rigidity.Neck.UPDR3ON.Num'] +
↪  (x['Rigidity.RUE.UPDR3ON.Num'] +x['Rigidity.LUE.UPDR3ON.Num'])/2 +
↪  (x['Rigidity.RLE.UPDR3ON.Num']+x['Rigidity.LLE.UPDR3ON.Num'])/2 +
```

```
ppmi_merge['Brady.Rigidity.Subscore-sym'].iloc[idx] = np.nan # Fill in subscores that
↪  should be nans as nan

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

#### TREMOR ####
tremor_list1 = ['Dominant.Side.Disease','Tremor.UPDRS2',
↪  'Postural.Tremor.Right.Hand.UPDRS3' ,'Postural.Tremor.Left.Hand.UPDRS3'
↪  ,'Postural.Tremor.Right.Hand.UPDRS3A' ,'Postural.Tremor.Left.Hand.UPDRS3A'
↪  ,'Postural.Tremor.Right.Hand.UPDR3OF' ,'Postural.Tremor.Left.Hand.UPDR3OF'
↪  ,'Postural.Tremor.Right.Hand.UPDR3ON' ,'Postural.Tremor.Left.Hand.UPDR3ON'
↪  ,'Kinetic.Tremor.Right.Hand.UPDRS3','Kinetic.Tremor.Left.Hand.UPDRS3',
↪  'Kinetic.Tremor.Right.Hand.UPDRS3A',
↪  'Kinetic.Tremor.Left.Hand.UPDRS3A','Kinetic.Tremor.Right.Hand.UPDR3OF','Kinetic.Tremor.Left.Hand.UP
↪  'Kinetic.Tremor.Right.Hand.UPDR3ON','Kinetic.Tremor.Left.Hand.UPDR3ON',
↪  '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', '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',
↪  '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', '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.UPDRS3', 'Constancy.of.Rest.Tremor.UPDRS3A',
↪  'Constancy.of.Rest.Tremor.UPDR3OF', 'Constancy.of.Rest.Tremor.UPDR3ON']

# Make tremor_list1 ready for ppmi_merge
new_list = []
for i in range(len(tremor_list1)) :
    if tremor_list1[i].startswith('Dominant') :
        new_list.append(tremor_list1[i])
        continue
    else :
        temp = tremor_list1[i].replace(tremor_list1[i], tremor_list1[i]+'.Num')
        new_list.append(temp)

# Create three original lists to be used below
tremor_right = new_list.copy()
tremor_left = new_list.copy()
tremor_sym = new_list.copy()

# Remove lefts from tremor_right
for i in tremor_right:
    if 'Left' in i or 'LUE' in i or 'LLE' in i :
        tremor_right.remove(i)

# Remove rights from tremor_left
```

```
for i in tremor_left :
    if 'Right' in i or 'RUE' in i or 'RLE' in i :
        tremor_left.remove(i)

ppmi_merge = add_lateralized_subscores(ppmi_merge, tremor_left, 'Left',
↪  'Tremor.Subscore-left')
ppmi_merge = add_lateralized_subscores(ppmi_merge, tremor_right, 'Right',
↪  'Tremor.Subscore-right')

## TREMOR SYM
tremor_only_sym = ppmi_merge[tremor_sym] # Get dataframe of only columns in tremor_left
ppmi_merge['Tremor.Subscore-sym'] = 0 # Initialize lateralized variable
tremor_only_sym.loc[tremor_only_sym['Dominant.Side.Disease'] != 'Symmetric', : ] = np.nan
↪  # Make all rows nan if dominant side of disease is not left
tremor_sym_temp= tremor_only_sym.drop('Dominant.Side.Disease', 1)
idx = tremor_sym_temp.index[tremor_sym_temp.isnull().all(1)]
x = tremor_sym_temp
x = x.fillna(0) # because adding 1 plus nan equals nan
ppmi_merge['Tremor.Subscore-sym'] = x['Tremor.UPDRS2.Num'] +
↪  (x['Postural.Tremor.Right.Hand.UPDRS3.Num']  +
↪  x['Postural.Tremor.Left.Hand.UPDRS3.Num'])/2 +
↪  (x['Postural.Tremor.Right.Hand.UPDRS3A.Num'] +
↪  x['Postural.Tremor.Left.Hand.UPDRS3A.Num'])/2 +
↪  (x['Postural.Tremor.Right.Hand.UPDR3OF.Num'] +
↪  x['Postural.Tremor.Left.Hand.UPDR3OF.Num'])/2 +
↪  (x['Postural.Tremor.Right.Hand.UPDR3ON.Num'] +
↪  x['Postural.Tremor.Left.Hand.UPDR3ON.Num'])/2 +
↪  (x['Kinetic.Tremor.Right.Hand.UPDRS3.Num'] +
↪  x['Kinetic.Tremor.Left.Hand.UPDRS3.Num'])/2 +
↪  (x['Kinetic.Tremor.Right.Hand.UPDRS3A.Num'] +
↪  x['Kinetic.Tremor.Left.Hand.UPDRS3A.Num'])/2 +
↪  (x['Kinetic.Tremor.Right.Hand.UPDR3OF.Num'] +
↪  x['Kinetic.Tremor.Left.Hand.UPDR3OF.Num'])/2 +
↪  (x['Kinetic.Tremor.Right.Hand.UPDR3ON.Num'] +
↪  x['Kinetic.Tremor.Left.Hand.UPDR3ON.Num'])/2 +
↪  (x['Rest.Tremor.Amplitude.RUE.UPDRS3.Num']+
↪  x['Rest.Tremor.Amplitude.LUE.UPDRS3.Num'])/2 +
↪  (x['Rest.Tremor.Amplitude.RLE.UPDRS3.Num'] +
↪  x['Rest.Tremor.Amplitude.LLE.UPDRS3.Num'])/2 +
↪  x['Rest.Tremor.Amplitude.Lip.Jaw.UPDRS3.Num'] +
↪  (x['Rest.Tremor.Amplitude.RUE.UPDRS3A.Num'] +
↪  x['Rest.Tremor.Amplitude.LUE.UPDRS3A.Num'])/2 +
↪  (x['Rest.Tremor.Amplitude.RLE.UPDRS3A.Num'] +
↪  x['Rest.Tremor.Amplitude.LLE.UPDRS3A.Num'])/2 +
↪  x['Rest.Tremor.Amplitude.Lip.Jaw.UPDRS3A.Num'] +
↪  (x['Rest.Tremor.Amplitude.RUE.UPDR3OF.Num'] +
↪  x['Rest.Tremor.Amplitude.LUE.UPDR3OF.Num'])/2 +
↪  (x['Rest.Tremor.Amplitude.RLE.UPDR3OF.Num'] +
↪  x['Rest.Tremor.Amplitude.LLE.UPDR3OF.Num'])/2 +
↪  x['Rest.Tremor.Amplitude.Lip.Jaw.UPDR3OF.Num'] +
↪  (x['Rest.Tremor.Amplitude.RUE.UPDR3ON.Num'] +
↪  x['Rest.Tremor.Amplitude.LUE.UPDR3ON.Num'])/2 +
↪  (x['Rest.Tremor.Amplitude.RLE.UPDR3ON.Num'] +
↪  x['Rest.Tremor.Amplitude.LLE.UPDR3ON.Num'])/2 +
↪  x['Rest.Tremor.Amplitude.Lip.Jaw.UPDR3ON.Num'] +
↪  x['Constancy.of.Rest.Tremor.UPDRS3.Num'] + x['Constancy.of.Rest.Tremor.UPDRS3A.Num']
↪  +  x['Constancy.of.Rest.Tremor.UPDR3OF.Num'] +
↪  x['Constancy.of.Rest.Tremor.UPDR3ON.Num']
```

```
ppmi_merge['Tremor.Subscore-sym'].iloc[idx] = np.nan

# Combine left and right and sym scores into same column (.lateralized)
ppmi_merge["Tremor.Subscore.lateralized"] =
↪  ppmi_merge.pop("Tremor.Subscore-right").fillna(ppmi_merge.pop("Tremor.Subscore-left"))
ppmi_merge["Tremor.Subscore.lateralized"] =
↪  ppmi_merge.pop("Tremor.Subscore.lateralized").fillna(ppmi_merge.pop("Tremor.Subscore-sym"))
lateralized_scores = ppmi_merge[['Subject.ID', 'Event.ID', 'Dominant.Side.Disease',
↪  'Brady.Rigidity.Subscore.lateralized', 'Tremor.Subscore.lateralized']]
ppmi_merge['Brady.Rigidity.Subscore.lateralized'] =
↪  lateralized_scores['Brady.Rigidity.Subscore.lateralized']
ppmi_merge['Tremor.Subscore.lateralized'] =
↪  lateralized_scores['Tremor.Subscore.lateralized']

## If lateralized subscore is nan - input the non-lateralized score for Tremor and
↪  Brady.Rigidity
# Tremor
latisna = ppmi_merge['Tremor.Subscore.lateralized'].isna() &
↪  ppmi_merge['Tremor.Subscore'].isna()
ppmi_merge["Tremor.Subscore.lateralized"].loc[latisna] =
↪  ppmi_merge["Tremor.Subscore"].loc[latisna]

# Brady.Rigidity -
latisna = ppmi_merge['Brady.Rigidity.Subscore.lateralized'].isna() &
↪  ppmi_merge['Brady.Rigidity.Subscore'].isna()
ppmi_merge["Brady.Rigidity.Subscore.lateralized"].loc[latisna] =
↪  ppmi_merge["Brady.Rigidity.Subscore"].loc[latisna]

# 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'

## Inlcude columns for bestEventID (bestScreening, bestBaseline, etc) and denote the
↪  highest resnetGrade with True (else = False)
# BA edit
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[mybe].iloc[idx] = 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[mybe].iloc[maxidx] = 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['bestAtImage.Acquisition.Date'].iloc[idx] = 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['bestAtImage.Acquisition.Date'].iloc[maxidx] = True

# DX simplified column - BA edits
ppmi_merge['DXsimplified'] = '' # Initialize DXsimplfied

selgba = ppmi_merge['Consensus.Subtype'] == 'Healthy Control'
ppmi_merge['DXsimplified'].loc[selgba == True] = 'HC'

selgba = ppmi_merge['Enroll.Diagnosis'] == 'Healthy Control'
ppmi_merge['DXsimplified'].loc[selgba == True] = 'HC'

selgba = ppmi_merge['Enroll.Diagnosis'] == "Parkinson's Disease"
ppmi_merge['DXsimplified'].loc[selgba == True] = "Sporadic_PD" # NOTE: default to
↪  sporadic PD unless othewise noted

selgba = ppmi_merge['Enroll.Diagnosis'] == "SWEDD"
ppmi_merge['DXsimplified'].loc[selgba == True] = "nonPDorMSA"

selgba = ppmi_merge['Enroll.Diagnosis'] == "Prodromal"
ppmi_merge['DXsimplified'].loc[selgba == True] = "Sporadic_Pro"

selgba = ppmi_merge['Consensus.Subtype'] == "GBA"
ppmi_merge['DXsimplified'].loc[selgba == True] = "GBA_HC"

selgba = ppmi_merge['Consensus.Subtype'] == "Genetic : GBA"
ppmi_merge['DXsimplified'].loc[selgba == True] = "GBA_PD"

selgba = ppmi_merge['Consensus.Subtype'] == "Genetic : GBA not Prodromal" # GBA_PD
ppmi_merge['DXsimplified'].loc[selgba == True] = "GBA_PD"

selgba = ppmi_merge['Consensus.Subtype'] == "Genetic : GBA Prodromal"
ppmi_merge['DXsimplified'].loc[selgba == True] ="GBA_Pro"

selgba = ppmi_merge['Consensus.Subtype'] == "Genetic : LRRK2"
ppmi_merge['DXsimplified'].loc[selgba == True] ="LRRK2_PD"

selgba = ppmi_merge['Consensus.Subtype'] == "Genetic : LRRK2 + GBA"
ppmi_merge['DXsimplified'].loc[selgba == True] ="LRRK2_PD"
```

```
selgba = ppmi_merge['Consensus.Subtype'] ==  "Genetic : LRRK2 + GBA not Prodromal"
ppmi_merge['DXsimplified'].loc[selgba == True] = "LRRK2_PD"

selgba = ppmi_merge['Consensus.Subtype'] ==  "Genetic : LRRK2 + GBA Prodromal"
ppmi_merge['DXsimplified'].loc[selgba == True] = "LRRK2_Pro"

selgba = ppmi_merge['Consensus.Subtype'] == "Genetic : LRRK2 not Prodromal"
ppmi_merge['DXsimplified'].loc[selgba == True] = "LRRK2_PD"

selgba = ppmi_merge['Consensus.Subtype'] ==  "Genetic : LRRK2 Phenoconverted"
ppmi_merge['DXsimplified'].loc[selgba == True] = "LRRK2_Pro"

selgba = ppmi_merge['Consensus.Subtype'] ==  "Genetic : LRRK2 Prodromal" # GBA_PD
ppmi_merge['DXsimplified'].loc[selgba == True] = "LRRK2_Pro"

selgba = ppmi_merge['Consensus.Subtype'] ==  "Genetic : SNCA" # GBA_PD
ppmi_merge['DXsimplified'].loc[selgba == True] = "SNCA_PD"

selgba = ppmi_merge['Consensus.Subtype'] == "Genetic : SNCA Prodromal" # GBA_PD
ppmi_merge['DXsimplified'].loc[selgba == True] = "SNCA_Pro"

selgba = ppmi_merge['Consensus.Subtype'] ==  "Hyposmia" # GBA_PD
ppmi_merge['DXsimplified'].loc[selgba == True] = "Sporadic_Pro"

selgba = ppmi_merge['Consensus.Subtype'] ==  "Hyposmia : Phenoconverted" # GBA_PD
ppmi_merge['DXsimplified'].loc[selgba == True] = "Sporadic_Pro"

selgba = ppmi_merge['Consensus.Subtype'] == "No Mutation not Prodromal" # GBA_PD
ppmi_merge['DXsimplified'].loc[selgba == True] == np.NaN
```

```
selgba = ppmi_merge['Consensus.Subtype'] ==  "non-HC" # GBA_PD
ppmi_merge['DXsimplified'].loc[selgba == True] == np.NaN
```

```
selgba = ppmi_merge['Consensus.Subtype'] ==  "non-PD" # GBA_PD
ppmi_merge['DXsimplified'].loc[selgba == True] == "nonPDorMSA"
```

```
selgba = ppmi_merge['Consensus.Subtype'] ==  "RBD" # GBA_PD
ppmi_merge['DXsimplified'].loc[selgba == True] == "Sporadic_Pro"
```

```
selgba = ppmi_merge['Consensus.Subtype'] ==  "RBD : Phenoconverted" # GBA_PD
ppmi_merge['DXsimplified'].loc[selgba == True] == "Sporadic_Pro"
```

```
selgba = ppmi_merge['Consensus.Subtype'] ==  "RBD : Phenoconverted with GBA" # GBA_PD
ppmi_merge['DXsimplified'].loc[selgba == True] == "GBA_Pro"
```

```
selgba = ppmi_merge['Consensus.Subtype'] ==  "Sporadic"
ppmi_merge['DXsimplified'].loc[selgba == True] == "Sporadic_PD"
```

```
selgba = ppmi_merge['Consensus.Subtype'] ==  "SWEDD/non-PD Active"
ppmi_merge['DXsimplified'].loc[selgba == True] == "nonPDorMSA"
```

```
selgba = ppmi_merge['Consensus.Subtype'] ==  "SWEDD/PD Active"
ppmi_merge['DXsimplified'].loc[selgba == True] == np.NaN

## Add in Visit column - BA edits
```

```
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_v0.csv')
```