PPMI Merge

2022-04-25

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

Getting the data

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

Import required modules

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

userdir = '/Users/areardon/Desktop/ppmi_merge/'
ppmi_download_path = userdir + 'PPMI_Study_Data_Download/'
invicro_data_path = userdir
genetics_path = userdir + 'genetic_data/'
version = '0.0.2' # File version to save as i.e. ppmi_merge_v0.1.csv

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

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

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

    return cohort_df

def merge_columns(df : pd.DataFrame , old_df_columns : list, new_df_column_name : str, separator = str)
    """
```

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

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

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

def isNaN(num):
    return num != num

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

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

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

#### CLINICAL INFO ####
# Create cohort df
xlsx = pd.ExcelFile(ppmi_download_path + 'Consensus_Committee_Analytic_Datasets_28OCT21.xlsx') # Read i
pd_df = create_cohort_df(xlsx, 'PD') # Create Parkinson's Disease data frame from 'PD' sheet in 'Consen
prodromal_df = create_cohort_df(xlsx, 'Prodromal')# Create Prodromal data frame from 'PD' sheet in 'Con
hc_df = create_cohort_df(xlsx, 'HC')# Create HC data frame from 'PD' sheet in 'Consensus_Committee_Anal
swedd_df = create_cohort_df(xlsx, 'SWEDD') # Create SWEDD data frame from 'PD' sheet in 'Consensus_Comm

# Decode Enrollment.Subtype ('Summary' sheet) and Consensus.Subtype ('Summary Analytic' sheet) of 'Cons
pd_df.loc[(pd_df['ENRLPD'] == 1) & (pd_df['ENRLLRRK2'] == 0) & (pd_df['ENRLGBA'] == 0) & (pd_df['ENRLSN
pd_df.loc[(pd_df['ENRLPD' ]== 1) & (pd_df['ENRLLRRK2'] == 1), 'Enrollment.Subtype'] =  ' : LRRK2'
pd_df.loc[(pd_df['ENRLPD'] == 1) & (pd_df['ENRLGBA'] == 1), 'Enrollment.Subtype'] = ' : GBA'
pd_df.loc[(pd_df['ENRLPD'] == 1) & (pd_df['ENRLSNCA'] == 1), 'Enrollment.Subtype'] =  ' : SNCA'
pd_df.loc[(pd_df['CONPD'] == 1) & (pd_df['CONLRRK2'] == 0) | (pd_df['CONLRRK2'] == '.') & (pd_df['CONGB
```

```
pd_df.loc[(pd_df['CONPD'] == 1) & (pd_df['CONLRRK2']== 1) & (pd_df['CONGBA'] == 0) & (pd_df['CONSNCA'] =
pd_df.loc[(pd_df['CONPD'] == 1) & (pd_df['CONLRRK2'] == 0) & (pd_df['CONGBA'] == 1) & (pd_df['CONSNCA']
pd_df.loc[(pd_df['CONPD'] == 1) & (pd_df['CONLRRK2'] == 1) & (pd_df['CONGBA'] == 1) & (pd_df['CONSNCA']
pd_df.loc[(pd_df['CONPD'] == 1) & (pd_df['CONLRRK2'] == 0) & (pd_df['CONGBA'] == 0) & (pd_df['CONSNCA']
pd_df.loc[(pd_df['CONPD'] == 0) & (pd_df['CONPROD']== 0) , 'Consensus.Subtype'] = 'non-PD' # FIXME
pd_df.loc[(pd_df['CONPD'] == 0) & (pd_df['CONPROD']== 1) & (pd_df['CONLRRK2'] == 1) & (pd_df['CONGBA'] =
pd_df.loc[(pd_df['CONPD'] == 0) & (pd_df['CONPROD'] == 1) & (pd_df['CONLRRK2'] == 0) & (pd_df['CONGBA']

# Decode Enrollment.Subtype and Consensus.Subtype to be categorical variables for prodromal df
prodromal_df.loc[(prodromal_df['ENRLPROD'] == 1) & (prodromal_df['ENRLLRRK2'] == 1), 'Enrollment.Subtype
prodromal_df.loc[(prodromal_df['ENRLPROD'] == 1) & (prodromal_df['ENRLGBA'] == 1), 'Enrollment.Subtype'
prodromal_df.loc[(prodromal_df['ENRLPROD'] == 1) & (prodromal_df['ENRLSNCA'] == 1), 'Enrollment.Subtype
prodromal_df.loc[(prodromal_df['ENRLPROD'] == 1) & (prodromal_df['ENRLHPSM'] == 1) ,'Enrollment.Subtype
prodromal_df.loc[(prodromal_df['ENRLPROD'] == 1) & (prodromal_df['ENRLRBD'] == 1), 'Enrollment.Subtype'
prodromal_df.loc[(prodromal_df['CONPROD'] == 1) & (prodromal_df['PHENOCNV'] == 0) & (prodromal_df['CONLL
prodromal_df.loc[(prodromal_df['CONPROD'] == 1) & (prodromal_df['PHENOCNV'] == 1) & (prodromal_df['CONLL
prodromal_df.loc[(prodromal_df['CONPROD'] == 0) & (prodromal_df['CONLRRK2'] == 1) & (prodromal_df['CONGB
prodromal_df.loc[(prodromal_df['CONPROD'] == 1) & (prodromal_df['PHENOCNV'] == 0) & (prodromal_df['CONLL
prodromal_df.loc[(prodromal_df['CONPROD'] == 1) & (prodromal_df['PHENOCNV'] == 1) & (prodromal_df['CONLL
prodromal_df.loc[(prodromal_df['CONPROD'] == 0) & (prodromal_df['CONLRRK2'] == 0) & (prodromal_df['CONGB
prodromal_df.loc[(prodromal_df['CONPROD'] == 1) & (prodromal_df['PHENOCNV'] == 0) & (prodromal_df['CONLL
prodromal_df.loc[(prodromal_df['CONPROD'] == 1) & (prodromal_df['PHENOCNV'] == 1) & (prodromal_df['CONLL
prodromal_df.loc[(prodromal_df['CONPROD'] == 0) & (prodromal_df['CONLRRK2'] == 1) & (prodromal_df['CONGB
prodromal_df.loc[(prodromal_df['CONPROD'] == 1) & (prodromal_df['PHENOCNV'] == 0) & (prodromal_df['CONLL
prodromal_df.loc[(prodromal_df['CONPROD'] == 1) & (prodromal_df['PHENOCNV'] == 1) & (prodromal_df['CONLL
prodromal_df.loc[(prodromal_df['CONPROD'] == 0) & (prodromal_df['CONLRRK2'] == 0) & (prodromal_df['CONGB
prodromal_df.loc[(prodromal_df['CONPROD'] == 0) & (prodromal_df['CONLRRK2'] == 0) & (prodromal_df['CONGB
prodromal_df.loc[(prodromal_df['CONPROD'] == 1) & (prodromal_df['PHENOCNV'] == 0) & (prodromal_df['CONLL
prodromal_df.loc[(prodromal_df['CONPROD'] == 1) & (prodromal_df['PHENOCNV'] == 1) & (prodromal_df['CONLL
prodromal_df.loc[(prodromal_df['CONPROD'] == 0) & (prodromal_df['CONLRRK2'] == 0) & (prodromal_df['CONGB
prodromal_df.loc[(prodromal_df['CONPROD'] == 1) & (prodromal_df['CONRBD'] == 1) & (prodromal_df['PHENOCN
prodromal_df.loc[(prodromal_df['CONPROD'] == 1) & (prodromal_df['CONRBD'] == 1) & (prodromal_df['PHENOCN
prodromal_df.loc[(prodromal_df['CONPROD'] == 1) & (prodromal_df['CONRBD'] == 1) & (prodromal_df['PHENOCN

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

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

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

# Decode and re-organize full_df
full_df['CONPD'].replace({1 : 'Parkinson\'s Disease', 0 : ''}, inplace = True)
```

```
full_df['CONPROD'].replace({1 : 'Prodromal',  0 : ''}, inplace = True)
full_df['CONHC'].replace({1 : 'Healthy Control',  0 : ''}, inplace = True)
full_df['CONSWEDD'].replace({1 : 'SWEDD',  0 : 'SWEDD/PD'}, inplace = True)
full_df['Comments'].replace({'MSA' : 'Multiple System Atrophy'}, inplace = True)
full_df = decode_0_1_no_yes(full_df, ['PHENOCNV'])
full_df = merge_columns(full_df, ['CONPD', 'CONPROD', 'CONHC', 'CONSWEDD','Comments'], 'Consensus.Diagn
full_df = merge_columns(full_df, ['Subgroup', 'Enrollment.Subtype'], 'Enroll.Subtype', '') # Get one co
full_df = full_df.loc[:, ~full_df.columns.str.startswith(tuple(['CON','ENRL']))] # Remove columns that
full_df.rename(columns = {'Cohort' : 'Enroll.Diagnosis' , 'PHENOCNV' : 'Subject.Phenoconverted'}, inpla
full_df = full_df[['PATNO', 'Enroll.Diagnosis', 'Enroll.Subtype', 'Consensus.Diagnosis', 'Consensus.Sub
analytic_cohort_subids = full_df['PATNO'].unique() # subids for analytic cohort

## Add in age info at each visit
ppmi_merge = merge_new_csv(full_df, 'Age_at_visit.csv', ['PATNO', 'EVENT_ID', 'AGE_AT_VISIT'], merge_on
ppmi_merge['EVENT_ID'].fillna('SC', inplace = True) # One subject (41358) has event_id as NaN - replace

## Add demographics, vital signs, and pd diagnosis history info
ppmi_merge = merge_new_csv(ppmi_merge, 'Demographics.csv', ['PATNO','EVENT_ID','SEX', 'HANDED', 'BIRTHD
ppmi_merge = merge_new_csv(ppmi_merge, 'Vital_Signs.csv', ['PATNO','EVENT_ID','INFODT', 'WGTKG', 'HTCM'
ppmi_merge = merge_new_csv(ppmi_merge,  'PD_Diagnosis_History.csv', ['PATNO', 'EVENT_ID', 'SXDT','PDDXD
ppmi_merge['SEX'].replace({0 : 'Female', 1 : 'Male' }, inplace = True) # Decode sex
ppmi_merge['HANDED'].replace({1 : 'Right', 2 : 'Left', 3 : 'Mixed' }, inplace = True) # Decode handedne
ppmi_merge.rename(columns = {'AGE_AT_VISIT' : 'Age', 'SEX' : 'Sex', 'HANDED' : 'Handed', 'BIRTHDT' : 'B

## 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'
        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, dia
        ppmi_merge['PD.Diagnosis.Duration'].iloc[row_num] = ((diff.years)*12 + diff.months)/12 # PD.Dia

# Add Final (FNL) Event ID and info
ppmi_merge = merge_new_csv(ppmi_merge, 'Conclusion_of_Study_Participation.csv', ['PATNO', 'EVENT_ID', '
ppmi_merge = decode_0_1_no_yes(ppmi_merge, ['COMPLT'])
ppmi_merge['WDRSN'].replace({1 : 'Adverse Event', 2 : 'Completed study per protocol', 3 : 'Death', 4 :
ppmi_merge.rename(columns = {'COMPLT' : 'Completed.Study' , 'WDRSN': 'Reason.for.Withdrawal','WDDT' : '

## Add diagnosis change info on event id
diag_vis1 = condensed_df(full_df, ['PATNO', 'DIAG1', 'DIAG1VIS'], {'DIAG1VIS' : 'EVENT_ID'}, ['EVENT_ID
diag_vis2 = condensed_df(full_df, ['PATNO', 'DIAG2', 'DIAG2VIS'], {'DIAG2VIS' : 'EVENT_ID'},['EVENT_ID'
ppmi_merge.drop(['DIAG1', 'DIAG1VIS', 'DIAG2', 'DIAG2VIS'], axis = 1, inplace = True) # Drop these from
ppmi_merge = pd.merge(diag_vis1, ppmi_merge, on = ['EVENT_ID', 'PATNO'], how = "outer" ) # Merge in fir
ppmi_merge = pd.merge(diag_vis2, ppmi_merge, on = ['EVENT_ID', 'PATNO'], how = "outer" ) # Merge in sec
ppmi_merge['DIAG1'].replace({ 'PD' : 'Parkinson\'s Disease', 'DLB': 'Dimentia with Lewy Bodies'}, inpla
ppmi_merge['DIAG2'].replace({ 'MSA' : 'Multiple System Atrophy', 'DLB': 'Dimentia with Lewy Bodies'}, i
ppmi_merge.rename(columns = {'DIAG1' : 'First.Diagnosis.Change', 'DIAG2' : 'Second.Diagnosis.Change'},

## Dominant side of disease
ppmi_merge = merge_new_csv(ppmi_merge, 'PPMI_Original_Cohort_BL_to_Year_5_Dataset_Apr2020.csv',['PATNO'
```

```
ppmi_merge['DOMSIDE'].replace({1 : 'Left', 2 : 'Right', 3 : 'Symmetric'}, inplace = True) # Decode

## Participant Motor Function Questionnaire
ppmi_merge = merge_new_csv(ppmi_merge, 'Participant_Motor_Function_Questionnaire.csv', ['PATNO', 'EVENT
ppmi_merge['CMPLBY2'].replace({1: 'Participant', 2 : 'Caregiver' , 3 : 'Participant and Caregivier'}, in
ppmi_merge['PAG_NAME'].replace({'PQUEST' : 'Participant Motor Function Questionnaire'}, inplace = True)
ppmi_merge = decode_0_1_no_yes(ppmi_merge, ['TRBUPCHR', 'WRTSMLR', 'VOICSFTR', 'POORBAL', 'FTSTUCK', 'LS
ppmi_merge = ppmi_merge.rename(columns = {'PAG_NAME' : 'Motor.Function.Page.Name', 'CMPLBY2' : 'Motor.Fu

## Cognitive symptoms - Cognitive Categorization
ppmi_merge = merge_new_csv(ppmi_merge, 'Cognitive_Categorization.csv',['PATNO' , 'EVENT_ID', 'PAG_NAME'
ppmi_merge = decode_0_1_no_yes(ppmi_merge, ['COGDECLN', 'FNCDTCOG'])
ppmi_merge['COGDXCL'].replace({1 : '90 - 100%', 2 : '50 - 89%', 3 : '10 - 49%', 4 : '0 - 9%'}, inplace =
ppmi_merge['PTCGBOTH'].replace({1 : 'Participant', 2 : 'Caregiver', 3 : 'Participant and Caregiver'}, in
ppmi_merge['COGSTATE'].replace({1 : 'Normal Condition', 2 : 'Mild Cognitive Impairment', 3 : 'Dimentia']
ppmi_merge = ppmi_merge.rename(columns = {'PAG_NAME' : 'Cognitive.Page.Name', 'COGDECLN' : 'Cognitive.De
ppmi_merge['Cognitive.Page.Name'].replace({'COGCATG' : 'Cognitive Categorization'}, inplace = True) # R

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

## Medication Status - Concomitant Med Log # FIXME not a useful way to show medication status
med_df = pd.read_csv(ppmi_download_path + 'Concomitant_Medication_Log.csv', skipinitialspace=True) # Me
med_df.replace({';' : ','}, regex = True, inplace = True) # Replace ';' with ',' in med_df
med_df['CMTRT'] = med_df['CMTRT'].str.title() # Capitalize all medication names
med_df['STARTDT'].fillna('NA', inplace = True) # Fillna
med_df['STOPDT'].fillna('NA', inplace = True) # Fillna
med_df = med_df.astype({"STARTDT" : 'str', "STOPDT" :'str'}) # Change start and stop date to strings
med_df = merge_columns(med_df, ['STARTDT', 'STOPDT'], 'Start_Stop', '-') # Merge columns Start and stop
med_df['Start_Stop'] = '(' + med_df['Start_Stop'].astype(str) + ')'# Put parenthesis around dates so wh
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 sub


## LEDD Medication Status - FIXME Assumtion : If stop date is NA we assume LEDD only occurred in that m
LEDD_med_df = pd.read_csv(ppmi_download_path + 'LEDD_Concomitant_Medication_Log.csv', skipinitialspace =
LEDD_med_df = LEDD_med_df[['PATNO', 'LEDD', 'STARTDT', 'STOPDT']] # Keep only
ppmi_merge['INFODT'] = pd.to_datetime(ppmi_merge['INFODT'], format = '%m%Y', errors = 'ignore') # chang
LEDD_med_df['STARTDT'] = pd.to_datetime(LEDD_med_df['STARTDT'], format = '%m%Y', errors = 'ignore') # c
LEDD_med_df['STOPDT'] = pd.to_datetime(LEDD_med_df['STOPDT'], format = '%m%Y', errors = 'ignore') # cha
LEDD_med_df['STOPDT2'] = LEDD_med_df['STOPDT'] # Initialize second stop date variable
LEDD_med_df['STOPDT2'].fillna(LEDD_med_df['STARTDT'], inplace = True) # Fill in NaN stop dates with sta
LEDD_med_df = LEDD_med_df.merge(LEDD_med_df.apply(lambda s: pd.date_range(s.STARTDT, s.STOPDT2, freq='M
LEDD_med_df.drop(['STOPDT2'], axis = 1, inplace = True) # Drop
LEDD_med_df.rename(columns = {'STARTDT' : 'LEDD.STARTDT', 'STOPDT': 'LEDD.STOPDT'}, inplace = True) # R

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

```
# Df with LD formulas as variables
LD_only = ppmi_merge_temp[ppmi_merge_temp["LEDD"].str.contains("LD") == True] # Get df with only columns
LD_only['STOPDT2'] = LD_only['LEDD.STOPDT'] # Initialize second stop date variable
LD_only['STOPDT2'].fillna(LD_only['LEDD.STARTDT'], inplace = True) # Fill in NaN stop dates with start
LD_only = LD_only.merge(LD_only.apply(lambda s: pd.date_range(s['LEDD.STARTDT'], s.STOPDT2, freq='MS',
LD_only.drop(['STOPDT2','EVENT_ID','LEDD.STARTDT','LEDD.STOPDT'], axis = 1, inplace = True) # Drop
ppmi_merge_LD_only = ppmi_merge[['PATNO', 'EVENT_ID', 'INFODT']]
ppmi_merge_LD_only = pd.merge(ppmi_merge_LD_only, LD_only, on = ['PATNO', 'INFODT'], how = "left")
ppmi_merge_LD_only = ppmi_merge_LD_only[['PATNO', 'EVENT_ID', 'LEDD']] # keep only
ppmi_merge_LD_only.rename(columns = {'LEDD' : 'LD'}, inplace = True)
ppmi_merge_LD_only = ppmi_merge_LD_only.drop_duplicates() # with ongoing stop dates some overlap when m

# Df without LD formula var - only numeric LEDD vars
ppmi_merge_temp = ppmi_merge_temp[ppmi_merge_temp["LEDD"].str.contains("LD") == False] # Remove LD x .3
ppmi_merge_temp = ppmi_merge_temp.astype({'LEDD' : 'float'})
LEDD_sum = ppmi_merge_temp.groupby(['PATNO', 'EVENT_ID'])['LEDD'].sum().reset_index() # Get sum
LEDD_sum.rename(columns = {'LEDD' : 'LEDD.sum'}, inplace = True) # Rename
ppmi_merge = pd.merge(ppmi_merge, LEDD_sum, on = ['PATNO','EVENT_ID'], how = "left") # Merge into ppmi_
ppmi_merge = pd.merge(ppmi_merge, ppmi_merge_LD_only, on = ['PATNO','EVENT_ID'], how = "left") # Merge

## If in LD column there is a (i.e. 150 + LD) * 0.33) - create a new duplicate column with just 150 var
import re
def func(row):
   if '(' in str(row['LEDD']):
        row2 = row.copy()
        # make edits to row2
        temp = row2['LEDD']
        row2['LEDD'] = re.search('\(([0-9]+)', temp).group(1)
        return pd.concat([row, row2], axis=1)
   return row

## GET LEVODOPA ONLY FOR BELOW CALCULATIONS - # FIXME - need more clarification before adding this
LEDD_med_df = pd.read_csv(ppmi_download_path + 'LEDD_Concomitant_Medication_Log.csv', skipinitialspace =
LEDD_med_df = LEDD_med_df[['PATNO','LEDTRT', 'LEDD', 'STARTDT', 'STOPDT']] # Keep only
search_for = ['levodopa', 'sinemet', 'rytary', 'stalevo'] # FIXME per Pavan comments but there might be
levodopa_only = LEDD_med_df[LEDD_med_df["LEDTRT"].str.contains('|'.join(search_for), case = False) == T
levodopa_only = pd.concat([func(row) for _, row in levodopa_only.iterrows()], ignore_index=True, axis=1
levodopa_only['STOPDT2'] = levodopa_only['STOPDT'] # Initialize second stop date variable
levodopa_only['STOPDT2'].fillna(levodopa_only['STARTDT'], inplace = True) # Fill in NaN stop dates with
levodopa_only = levodopa_only.merge(levodopa_only.apply(lambda s: pd.date_range(s['STARTDT'], s.STOPDT2
levodopa_only.drop(['STOPDT2','STARTDT','STOPDT'], axis = 1, inplace = True)
ppmi_merge_levodopa_only = ppmi_merge[['PATNO', 'EVENT_ID', 'INFODT']]
ppmi_merge_levodopa_only = pd.merge(ppmi_merge_levodopa_only, levodopa_only, on = ['PATNO', 'INFODT'],
ppmi_merge_levodopa_only = ppmi_merge_levodopa_only[['PATNO', 'EVENT_ID', 'LEDD']] # keep only
ppmi_merge_levodopa_only = ppmi_merge_levodopa_only.drop_duplicates() # with ongoing stop dates some ov
ppmi_merge_levodopa_only = ppmi_merge_levodopa_only[ppmi_merge_levodopa_only["LEDD"].str.contains("LD")
ppmi_merge_levodopa_only = ppmi_merge_levodopa_only.astype({'LEDD' : 'float'})
ppmi_merge_levodopa_only = ppmi_merge_levodopa_only.groupby(['PATNO', 'EVENT_ID'])['LEDD'].sum().reset_
ppmi_merge_levodopa_only.rename(columns = {'LEDD' : 'Levodopa.only'}, inplace = True)
ppmi_merge = pd.merge(ppmi_merge, ppmi_merge_levodopa_only, on = ['PATNO','EVENT_ID'], how = "left") #

# Calculate LD formulas and add to LEDD column numbers # FIXME need to switch calculation from LEDD to
ppmi_merge['LEDD.sum'].fillna(0, inplace = True)
```

```
for row_num in range(len(ppmi_merge)) :
    if ppmi_merge['LD'].loc[row_num] == 'LD x 0.33' :
        ppmi_merge['LEDD.sum'].loc[row_num] = float(ppmi_merge['Levodopa.only'].loc[row_num]) * 0.33 + i
    elif ppmi_merge['LD'].loc[row_num] == '(150.0 + LD) x 0.33' :
        ppmi_merge['LEDD.sum'].loc[row_num] = float(ppmi_merge['Levodopa.only'].loc[row_num]) * 0.33 + i
    elif ppmi_merge['LD'].loc[row_num] == '(600.0 + LD) x 0.33' :
        ppmi_merge['LEDD.sum'].loc[row_num] =  float(ppmi_merge['Levodopa.only'].loc[row_num]) * 0.33 +
    elif ppmi_merge['LD'].loc[row_num] == 'LD x 0.5':
        ppmi_merge['LEDD.sum'].loc[row_num] = float(ppmi_merge['Levodopa.only'].loc[row_num]) * 0.5 + fl
    elif ppmi_merge['LD'].loc[row_num] =='(800.0 + LD) x 0.33':
        ppmi_merge['LEDD.sum'].loc[row_num] = float(ppmi_merge['Levodopa.only'].loc[row_num]) * 0.33 + i
    elif ppmi_merge['LD'].loc[row_num] == '(300.0 + LD) x 0.33' :
        ppmi_merge['LEDD.sum'].loc[row_num] = float(ppmi_merge['Levodopa.only'].loc[row_num]) * 0.33 + i
    elif ppmi_merge['LD'].loc[row_num] == '(225.0 + LD) x 0.33' :
        ppmi_merge['LEDD.sum'].loc[row_num] = float(ppmi_merge['Levodopa.only'].loc[row_num]) * 0.33 + i
    elif ppmi_merge['LD'].loc[row_num] == '(450.0 + LD) x 0.33' :
        ppmi_merge['LEDD.sum'].loc[row_num] = float(ppmi_merge['Levodopa.only'].loc[row_num]) * 0.33 + i
ppmi_merge.drop(['LD'], axis = 1, inplace = True)
ppmi_merge = ppmi_merge.drop_duplicates() # with ongoing stop dates some overlap when merging on infodt
ppmi_merge['LEDD.sum'].replace({0 : np.NaN}, inplace = True)
ppmi_merge.drop(['Levodopa.only'], axis = 1, inplace = True) # Drop

## LEDD Medication Status - FIXME Assumption : If stop date is NA we assume therapy is ongoing
LEDD_med_df = pd.read_csv(ppmi_download_path + 'LEDD_Concomitant_Medication_Log.csv', skipinitialspace =
LEDD_med_df = LEDD_med_df[['PATNO', 'LEDD', 'STARTDT', 'STOPDT']] # keep only
ppmi_merge['INFODT'] = pd.to_datetime(ppmi_merge['INFODT'], format = '%m%Y', errors = 'ignore') # Chang
LEDD_med_df['STARTDT'] = pd.to_datetime(LEDD_med_df['STARTDT'], format = '%m%Y', errors = 'ignore') # C
LEDD_med_df['STOPDT'] = pd.to_datetime(LEDD_med_df['STOPDT'], format = '%m%Y', errors = 'ignore') # Cha
LEDD_med_df['STOPDT3'] = LEDD_med_df['STOPDT']
LEDD_med_df['STOPDT3'].fillna('01/2022', inplace = True) # ASSUMPTION fill in ongoing stopdate (12/2021
LEDD_med_df['STOPDT3'] = pd.to_datetime(LEDD_med_df['STOPDT3'], format = '%m%Y', errors = 'ignore') # c
LEDD_med_df = LEDD_med_df.merge(LEDD_med_df.apply(lambda s: pd.date_range(s.STARTDT, s.STOPDT3, freq='M
LEDD_med_df.rename(columns = {'LEDD' : 'LEDD.ongoing', 'STARTDT' : 'LEDD.STARTDT.ongoing', 'STOPDT3': '

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

# Df with LD formulas as variables
LD_only = ppmi_merge_temp[ppmi_merge_temp["LEDD.ongoing"].str.contains("LD")==True] # Get df with only
LD_only['STOPDT2'] = LD_only['LEDD.STOPDT.ongoing'] # Initialize second stop date variable
LD_only['STOPDT2'].fillna('01/2022', inplace = True) # Fill in NaN stop dates with start dates # ASSUMP
LD_only['STOPDT2'] = pd.to_datetime(LD_only['STOPDT2'], format = '%m%Y', errors = 'ignore') # change to
LD_only = LD_only.merge(LD_only.apply(lambda s: pd.date_range(s['LEDD.STARTDT.ongoing'], s.STOPDT2, freq
LD_only.drop(['STOPDT2','EVENT_ID','LEDD.STARTDT.ongoing','LEDD.STOPDT.ongoing'], axis = 1, inplace = T
ppmi_merge_LD_only = ppmi_merge[['PATNO', 'EVENT_ID', 'INFODT']]
ppmi_merge_LD_only = pd.merge(ppmi_merge_LD_only, LD_only, on = ['PATNO', 'INFODT'], how = "left")
ppmi_merge_LD_only = ppmi_merge_LD_only[['PATNO', 'EVENT_ID', 'LEDD.ongoing']] # keep only
ppmi_merge_LD_only.rename(columns = {'LEDD.ongoing' : 'LD'}, inplace = True)
ppmi_merge_LD_only = ppmi_merge_LD_only.drop_duplicates() # with ongoing stop dates some overlap when m

# Df without LD formula var - only numeric LEDD vars
```

```
ppmi_merge_temp = ppmi_merge_temp[ppmi_merge_temp["LEDD.ongoing"].str.contains("LD") == False] # Remove
ppmi_merge_temp = ppmi_merge_temp.astype({'LEDD.ongoing' : 'float'})
LEDD_sum = ppmi_merge_temp.groupby(['PATNO', 'EVENT_ID'])['LEDD.ongoing'].sum().reset_index() #  Sum
LEDD_sum.rename(columns = {'LEDD.ongoing' : 'LEDD.ongoing.sum'}, inplace = True)
ppmi_merge = pd.merge(ppmi_merge, LEDD_sum, on = ['PATNO','EVENT_ID'], how = "left") # Merge into ppmi_
ppmi_merge = pd.merge(ppmi_merge, ppmi_merge_LD_only, on = ['PATNO','EVENT_ID'], how = "left") # Merge

## If in LD column there is a (i.e. 150 + LD) * 0.33) - create a new duplicate column with just 150 var
## GET LEVODOPA ONLY FOR BELOW CALCULATIONS - # FIXME - need more clarification before adding this
LEDD_med_df = pd.read_csv(ppmi_download_path + 'LEDD_Concomitant_Medication_Log.csv', skipinitialspace =
LEDD_med_df = LEDD_med_df[['PATNO','LEDTRT', 'LEDD', 'STARTDT', 'STOPDT']] # Keep only
search_for = ['levodopa', 'sinemet', 'rytary', 'stalevo'] # FIXME per Pavan comments but there might be
levodopa_only = LEDD_med_df[LEDD_med_df["LEDTRT"].str.contains('|'.join(search_for), case = False) == T
levodopa_only = pd.concat([func(row) for _, row in levodopa_only.iterrows()], ignore_index=True, axis=1
levodopa_only['STOPDT2'] = levodopa_only['STOPDT'] # Initialize second stop date variable
levodopa_only['STOPDT2'].fillna('01/2022', inplace = True) # Fill in NaN stop dates with start dates # A
levodopa_only = levodopa_only.merge(levodopa_only.apply(lambda s: pd.date_range(s['STARTDT'], s.STOPDT2
levodopa_only.drop(['STOPDT2','STARTDT','STOPDT'], axis = 1, inplace = True)
ppmi_merge_levodopa_only = ppmi_merge[['PATNO', 'EVENT_ID', 'INFODT']]
ppmi_merge_levodopa_only = pd.merge(ppmi_merge_levodopa_only, levodopa_only, on = ['PATNO', 'INFODT'], H
ppmi_merge_levodopa_only = ppmi_merge_levodopa_only[['PATNO', 'EVENT_ID', 'LEDD']] # keep only
ppmi_merge_levodopa_only = ppmi_merge_levodopa_only.drop_duplicates() # with ongoing stop dates some ov
ppmi_merge_levodopa_only = ppmi_merge_levodopa_only[ppmi_merge_levodopa_only["LEDD"].str.contains("LD")
ppmi_merge_levodopa_only = ppmi_merge_levodopa_only.astype({'LEDD' : 'float'})
ppmi_merge_levodopa_only = ppmi_merge_levodopa_only.groupby(['PATNO', 'EVENT_ID'])['LEDD'].sum().reset_i
ppmi_merge_levodopa_only.rename(columns = {'LEDD' : 'Levodopa.only'}, inplace = True)
ppmi_merge = pd.merge(ppmi_merge, ppmi_merge_levodopa_only, on = ['PATNO','EVENT_ID'], how = "left") # A

# Calculate LD formulas and add to LEDD column numbers # FIXME need to switch calculation from LEDD to
for row_num in range(len(ppmi_merge)) :
    if ppmi_merge['LD'].loc[row_num] == 'LD x 0.33' :
        ppmi_merge['LEDD.ongoing.sum'].loc[row_num] = float(ppmi_merge['Levodopa.only'].loc[row_num]) *
    elif ppmi_merge['LD'].loc[row_num] == '(150.0 + LD) x 0.33' :
        ppmi_merge['LEDD.ongoing.sum'].loc[row_num] = float(ppmi_merge['Levodopa.only'].loc[row_num]) *
    elif ppmi_merge['LD'].loc[row_num] == '(600.0 + LD) x 0.33' :
        ppmi_merge['LEDD.ongoing.sum'].loc[row_num] =  float(ppmi_merge['Levodopa.only'].loc[row_num]) *
    elif ppmi_merge['LD'].loc[row_num] == 'LD x 0.5':
        ppmi_merge['LEDD.ongoing.sum'].loc[row_num] = float(ppmi_merge['Levodopa.only'].loc[row_num]) *
    elif ppmi_merge['LD'].loc[row_num] =='(800.0 + LD) x 0.33':
        ppmi_merge['LEDD.ongoing.sum'].loc[row_num] = float(ppmi_merge['Levodopa.only'].loc[row_num]) *
    elif ppmi_merge['LD'].loc[row_num] == '(300.0 + LD) x 0.33' :
        ppmi_merge['LEDD.ongoing.sum'].loc[row_num] = float(ppmi_merge['Levodopa.only'].loc[row_num]) *
    elif ppmi_merge['LD'].loc[row_num] == '(225.0 + LD) x 0.33' :
        ppmi_merge['LEDD.ongoing.sum'].loc[row_num] = float(ppmi_merge['Levodopa.only'].loc[row_num]) *
    elif ppmi_merge['LD'].loc[row_num] == '(450.0 + LD) x 0.33' :
        ppmi_merge['LEDD.ongoing.sum'].loc[row_num] = float(ppmi_merge['Levodopa.only'].loc[row_num]) *
#ppmi_merge['LEDD.STOPDT.ongoing'].replace({'01/2022'  : np.NaN}, inplace = True) # Replace filler stop
ppmi_merge.drop(['LD'], axis = 1, inplace = True)


## Comorbidities # FIXME not useful column
comorbid_df = pd.read_csv(ppmi_download_path + 'Medical_Conditions_Log.csv', skipinitialspace=True) # M
comorbid_df.replace({';' : ','}, regex = True, inplace = True) # Replace ';' with ','
```

```
comorbid_df = comorbid_df[['PATNO', 'EVENT_ID', 'MHDIAGDT','MHTERM']] # keep only
comorbid_df['MHTERM'] = comorbid_df['MHTERM'].str.capitalize() # Capitalize all MHTERM names
comorbid_df['MHDIAGDT'].fillna('NA', inplace = True) # If no diagnosis date - fill in with NA
comorbid_df['MHDIAGDT'] = '(' + comorbid_df['MHDIAGDT'].astype(str) + ')'# Put parentheses around diagn
comorbid_df = comorbid_df.astype({"MHDIAGDT" : 'str'}) # Change date to string
comorbid_df = merge_columns(comorbid_df, ['MHTERM', 'MHDIAGDT'], 'Medical.History.Description(Diagnosis
comorbid_df = comorbid_df.groupby(['PATNO','EVENT_ID'])['Medical.History.Description(Diagnosis.Date)'].a
ppmi_merge = pd.merge(ppmi_merge, comorbid_df, on = ['PATNO','EVENT_ID'], how = "outer")

## Education (in years)
education_df = pd.read_csv(ppmi_download_path + 'Socio-Economics.csv', skipinitialspace = True) # Educa
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
ppmi_merge = pd.merge(ppmi_merge, education_df, on = ['PATNO'], how = "outer") # Merge

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

## Reindex
ppmi_merge = ppmi_merge.reindex(columns = ['PATNO', 'EVENT_ID', 'INFODT' , 'Enroll.Diagnosis', 'Enroll.S

## Add in other csvs
ppmi_merge = merge_new_csv(ppmi_merge, 'Modified_Boston_Naming_Test.csv', ['PATNO', 'EVENT_ID', 'MBSTNS
ppmi_merge = merge_new_csv(ppmi_merge, 'Clock_Drawing.csv', ['PATNO', 'EVENT_ID', 'CLCKTOT'], merge_on =
ppmi_merge = merge_new_csv(ppmi_merge, 'Benton_Judgement_of_Line_Orientation.csv', ['PATNO', 'EVENT_ID'
ppmi_merge = merge_new_csv(ppmi_merge, 'Letter_-_Number_Sequencing.csv', ['PATNO', 'EVENT_ID', 'LNS_TOT
ppmi_merge = merge_new_csv(ppmi_merge, 'Modified_Semantic_Fluency.csv', ['PATNO', 'EVENT_ID', 'DVS_SFTA
ppmi_merge = merge_new_csv(ppmi_merge, 'Hopkins_Verbal_Learning_Test_-_Revised.csv',['PATNO', 'EVENT_ID
ppmi_merge = merge_new_csv(ppmi_merge, 'Symbol_Digit_Modalities_Test.csv', ['PATNO', 'EVENT_ID', 'SDMTO
ppmi_merge.rename(columns = {'CLCKTOT' : 'Clock.Drawing.Total', 'JLOTOTCALC' : 'JOLO.Total', 'LNS_TOTRA

## REM Sleeep behavior disorder questionnaire
ppmi_merge = merge_new_csv(ppmi_merge, 'REM_Sleep_Behavior_Disorder_Questionnaire.csv', ['PATNO', 'EVEN
ppmi_merge.rename(columns = {'PAG_NAME' : 'REM.Sleep.Behavior.Disorder.Page.Name'}, inplace = True)
ppmi_merge['REM.Sleep.Behavior.Disorder.Page.Name'].replace({'REMSLEEP' : 'REM Sleep Behavior Disorder
ppmi_merge['PTCGBOTH'].replace({1 : 'Participant', 2 : 'Caregiver', 3: 'Participant and Caregiver'}, inp
rem_list = ['DRMVIVID', 'DRMAGRAC', 'DRMNOCTB', 'SLPLMBMV', 'SLPINJUR', 'DRMVERBL', 'DRMFIGHT', 'DRMUMV
ppmi_merge['RBDTotal.REM'] = ppmi_merge[rem_list].sum(axis = 1) # Add an RBDTotal.REM column
ppmi_merge = decode_0_1_no_yes(ppmi_merge, rem_list)
ppmi_merge.rename(columns = {'PTCGBOTH' : 'Sleep.Behavior.Source.REM','DRMVIVID' : 'Vivid.Dreams.REM' ,


#### IMAGING INFO ####
# FIXME - laterality issue?  SUV ?
ppmi_merge = merge_new_csv(ppmi_merge,'DaTScan_Analysis.csv', ['PATNO','EVENT_ID','DATSCAN_DATE','DATSC
ppmi_merge = merge_new_csv(ppmi_merge,'DaTScan_Imaging.csv', ['PATNO','EVENT_ID','PAG_NAME','INFODT','DA
ppmi_merge['DATSCAN'].replace({0 : 'Not Completed', 1: 'Completed', 2 : 'Completed using a previously a
ppmi_merge['SCNLOC'].replace({ 1 : 'Site', 2 : 'IND'}, inplace = True )
ppmi_merge['SCNINJCT'].replace({ 1 : 'DaTSCAN', 2 : 'Beta-CIT'}, inplace = True)
```

```
ppmi_merge['VSINTRPT'].replace({ '1' : 'Consistent with evidence' , '2' : 'Not consistent with evidence
ppmi_merge['VSRPTELG'].replace({1 : 'Eligible', 2 : 'Not eligible'}, inplace = True)
ppmi_merge.rename(columns = {'INFODT_x' : 'INFODT', 'INFODT_y' : 'DaTScan.INFODT', 'SCNLOC' : 'Location
ppmi_merge['INFODT'].fillna(ppmi_merge['DaTScan.INFODT'], inplace = True) # Fill in NaN infodts with da

# FIXME - After merging in datscan files, duplicate event ids being created one with DATSCAN as "Comple
duplicate_datscan = ppmi_merge[['PATNO','EVENT_ID']].duplicated(keep = False) # Find locations of True
duplicate_datscan_index = ppmi_merge[duplicate_datscan == True].index.tolist() # Get index of duplicate
dup_subid_list = [] # Initialize duplicate subid list variable
[dup_subid_list.append(index) for index in duplicate_datscan_index if ppmi_merge['DATSCAN'][index] == '
```

```
ppmi_merge = ppmi_merge.reset_index(drop = True)
[ppmi_merge.drop(index = i, axis = 1, inplace = True) for i in reversed(dup_subid_list) if ppmi_merge['
```

```
## MRI.csv
```

```
mri_df = pd.read_csv(ppmi_download_path + 'Magnetic_Resonance_Imaging__MRI_.csv', skipinitialspace=True
mri_df = mri_df[[ 'PATNO', 'EVENT_ID', 'INFODT', 'MRICMPLT', 'MRIWDTI', 'MRIWRSS', 'MRIRSLT', 'MRIRSSDF
mri_df['MRICMPLT'].replace({0 : 'Not Completed', 1 : 'Completed'}, inplace = True) # Decode
mri_df = decode_0_1_no_yes(mri_df, ['MRIWDTI','MRIWRSS','MRIRSSDF'])
mri_df['MRIRSLT'].replace({1 : 'Normal', 2 : 'Abnormal, not clinically significant', 3 : 'Abnormal, cli
mri_df.rename(columns = { 'INFODT' : 'Image.Acquisition.Date', 'MRICMPLT' : 'MRI.Completed', 'MRIWDTI'

# FIXME Some subjects had two baseline rows - 1 with incomplete MRI.Completed and 1 with complete as MR
duplicate_mri = mri_df[['PATNO','EVENT_ID']].duplicated(keep = False) # Find locations of True for dupl
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] == 'No
```

```
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.Compl
```

```
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

# Set a variable in ppmi_merge 'Subid.Date.TEMP' that is the subid and image acquisition date to match
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) + '
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 - w
matches = [current_subid_date for current_subid_date in subid_date_ordered if current_subid_date in s3w

# 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['

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


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

updrs_part1_df = setup_updrs_df('MDS-UPDRS_Part_I.csv',['ORIG_ENTRY','LAST_UPDATE','REC_ID'], {'NUPDRS1
updrs_part1_pq_df = setup_updrs_df('MDS-UPDRS_Part_I_Patient_Questionnaire.csv', ['ORIG_ENTRY','LAST_UP
updrs_part2_pq_df = setup_updrs_df('MDS_UPDRS_Part_II__Patient_Questionnaire.csv', ['ORIG_ENTRY','LAST_
updrs_part3_dos_df = setup_updrs_df('MDS-UPDRS_Part_III_ON_OFF_Determination___Dosing.csv', ['ORIG_ENTR
updrs_part4_motor_df = setup_updrs_df('MDS-UPDRS_Part_IV__Motor_Complications.csv', ['ORIG_ENTRY','LAST_

# 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)

# Change all UPDRS dataframe cols begin with 'N' to floats
updrs_list = [updrs_part1_df, updrs_part1_pq_df,updrs_part2_pq_df, updrs_part3_df, updrs_part3_dos_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()
updrs_part3_df_copy = updrs_part3_df.copy()
updrs_part3_dos_df_copy = updrs_part3_dos_df.copy()
updrs_part4_motor_df_copy = updrs_part4_motor_df.copy()

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

# Add extensions to updrs dfs
updrs_list_str = ['.UPDRS1','.UPDRS1','.UPDRS2', '.UPDRS3','.UPDRDOSE','.UPDRS4'] # extensions
for i in range(len(updrs_list)) :
    updrs_list[i] = add_extension_to_column_names(updrs_list[i], ['PATNO','EVENT_ID','INFODT'], updrs_li

# 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, updrs_part3_df, 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
        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' : 'Globa
updrs_numeric.to_csv('/Users/areardon/Desktop/updrs_numeric.csv')
## Subscore info from :
# https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7242837/
# https://www.movementdisorders.org/MDS-Files1/PDFs/Rating-Scales/MDS-UPDRS_English_FINAL.pdf

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

updrs_numeric = calculate_subscores(updrs_numeric, 'Brady.Rigidity.Subscore.UPDRS3',    ['Rigidity.Neck
updrs_numeric = calculate_subscores(updrs_numeric, 'Tremor.Subscore.UPDRS3', ['Tremor.UPDRS2', 'Postura
updrs_numeric = calculate_subscores(updrs_numeric, 'PIGD.Subscore.UPDRS3', ['Walking.Difficulty.UPDRS2'
```

```
updrs_numeric = add_extension_to_column_names(updrs_numeric, ['PATNO', 'EVENT_ID', 'PIGD.Subscore.UPDRS3


### UPDRS CATEGORICAL ####
# UPDRS3 (four dataframes) decode and rename in loop
updrs_part3_df_copy = decode_none2severe(updrs_part3_df_copy, ['NP3SPCH', 'NP3FACXP', 'NP3RIGN', 'NP3RI
updrs_part3_df_copy['DBS_STATUS'].replace({0 : 'OFF', 1 : 'ON'}, inplace = True)
updrs_part3_df_copy = decode_0_1_no_yes(updrs_part3_df_copy, ['DYSKPRES','DYSKIRAT'])
updrs_part3_df_copy['NHY'].replace({0 : 'Asymptomatic', 1 : 'Unilateral Movement Only', 2: 'Bilateral i
updrs_part3_df_copy['PDTRTMNT'].replace({0 : 'OFF' , 1: 'ON'}, inplace = True)
updrs_part3_df_copy.rename(columns = {'DBS_STATUS' : 'Deep.Brain.Stimulation.Treatment' , 'NP3SPCH' : '
updrs_part3_df_copy = add_extension_to_column_names(updrs_part3_df_copy, ['PATNO','EVENT_ID','INFODT'],

# Combine pd med dose date and time into one column
updrs3_merge = merge_columns(updrs_part3_df_copy, ['PDMEDDT.UPDRS3', 'PDMEDTM.UPDRS3'], 'Most.Recent.PD
updrs3_merge.to_csv("/Users/areardon/Desktop/updrs3_merge.csv")
# Rename page name variables
# FIXME upDRS
#updrs3_merge['PAG_NAME'].replace({'NUPDRS3' : 'MDS-UPDRS Part III (No Treatment)','NUPDRS3A' : 'MDS-UP

# Merge all UPDRS dfs together
updrs_cat = pd.merge(updrs_part1_df_copy, updrs_part1_pq_df_copy, on = ['PATNO', 'EVENT_ID'], how = "ou
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'}
updrs_cat = decode_none2severe(updrs_cat, ['NP1COG', 'NP1HALL', 'NP1DPRS', 'NP1ANXS', 'NP1APAT', 'NP1DD
updrs_cat.rename(columns = {'UPDRS.Part1.Source' : 'UPDRS.Part1.Source.UPDRS1', 'NP1COG' : 'Cognitive.I

# MDS_UPDRS_Part 1 Patient Questionnaire
updrs_cat['UPDRS.Part1.PQ.Source'].replace({1 : 'Patient', 2 : 'Caregiver', 3: 'Patient and Caregiver'}
updsr_cat = decode_none2severe(updrs_cat, ['NP1SLPN', 'NP1SLPD', 'NP1PAIN', 'NP1URIN', 'NP1CNST', 'NP1L
updrs_cat.rename(columns = {'UPDRS.Part1.PQ.Source' : 'UPDRS.Part1.Patient.Questionnaire.Source.UPDRS1'

# MDS_UPDRS Part 2
updrs_cat['UPDRS.Part2.Source'].replace({1 : 'Patient', 2 : 'Caregiver', 3 : 'Patient and Caregiver'}, i
updrs_cat = decode_none2severe(updrs_cat,['NP2SPCH','NP2SALV','NP2SWAL','NP2EAT','NP2DRES','NP2HYGN','N
updrs_cat.rename(columns = {'UPDRS.Part2.Source' : 'UPDRS.Part2.Source.UPDRS2', 'NP2SPCH' : 'UPDRS2.Spe

# MDS_UPDRS Part 3 On OFF determination Dosing
updrs_cat = decode_0_1_no_yes(updrs_cat, ['RMEXAM','DBSYN', 'OFFEXAM', 'OFFEXAM', 'ONEXAM'])
updrs_cat['RMTOFFRSN'].replace({1 : 'ON state not reached', 2 : 'Scheduling issues', 3 : 'Other reason'
updrs_cat['ONOFFORDER'].replace({1 : 'OFF', 2 : 'ON'}, inplace = True)
updrs_cat['RMONOFF'].replace({1 : 'OFF', 2 : 'ON'}, inplace = True)
updrs_cat['ONNORSN'].replace({1 : 'ON state not reached', 2 : 'Scheduling issues', 3 : 'Other reason'},
updrs_cat['PDMEDYN' ].replace({0 : False , 1 : True}, inplace = True)
updrs_cat = merge_columns(updrs_cat, ['ONPDMEDDT','ONPDMEDTM'], 'Most.Recent.PD.Med.Dose.Date.Time.Befo
updrs_cat['OFFNORSN'].replace({1 : 'Disease severity preventing turning off of DBS', 2 : 'Did not bring
updrs_cat = merge_columns(updrs_cat, ['OFFPDMEDDT','OFFPDMEDTM'], 'Most.Recent.PD.Med.Dose.Date.Time.Be
```

```
updrs_cat.rename(columns = {'RMONOFF' : 'MDS-UPDRS.Part3.Remote.Visit.ON.or.OFF.UPDRDOSE',  'Most.Recen

# MDS_UPDRS Part 4
updrs_cat['NP4WDYSK'].replace({0 : 'No dyskinesias', 1: 'Slight: <= 25% of waking day', 2: 'Mild : 26-5
updrs_cat['NP4DYSKI'].replace({0 : 'No dyskinesias', 1 : 'Slight', 2: 'Mild', 3 : 'Moderate', 4: 'Severe
updrs_cat['NP4OFF'].replace({0 : 'Normal: No OFF time', 1 : 'Slight: <= 25% of waking day ', 2: 'Mild :
updrs_cat['NP4FLCTI'].replace({0 : 'Normal', 1: 'Slight', 2: 'Mild', 3: 'Moderate', 4: 'Severe'}, inplac
updrs_cat['NP4DYSTN'].replace({0 : 'No dystonia', 1: 'Slight: <= 25% of time in OFF state', 2: 'Mild : 1
updrs_cat['NP4FLCTX'].replace({0: 'Normal', 1 : 'Slight', 2: 'Mild', 3 : 'Moderate', 4 : 'Severe'}, inpl
updrs_cat['RMNOPRT3'].replace({1 : 'Visit was not conducted with video', 2 : 'Scheduling issues', 3: 'O
updrs_cat.rename(columns = {'RMNOPRT3': 'Reason.UPDRSPart3.Not.Administered.Remote.Visit.UPDRDOSE','NP4W
updrs_cat = add_extension_to_column_names(updrs_cat, ['PATNO', 'EVENT_ID', 'INFODT'], '.Cat') # Add a .
updrs_cat.to_csv("/Users/areardon/Desktop/updrs_cat.csv")
# Create one df with UPDRS scores .Num (numeric) and all UPDRS scores .Cat (categorical)
updrs_cat.drop(['INFODT','PATNO','EVENT_ID'], axis = 1, inplace = True)
updrs_merged = pd.concat([updrs_cat, updrs_numeric], axis = 1)
#updrs_merged = pd.merge(updrs_cat, updrs_numeric, on = ['PATNO', 'EVENT_ID'], how = "outer")
updrs_merged.replace({'UR' : np.nan}, inplace = True) # Unable to Rate --> nan
ppmi_merge = pd.merge(ppmi_merge, updrs_merged, on = ['PATNO', 'EVENT_ID'], how = "outer")
ppmi_merge.to_csv('/Users/areardon/Desktop/ppmi_mergex.csv')

#### 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' : 'Medicati

# Change names of event ids to be indicative of months
ppmi_merge['Event.ID'].replace({'FNL' : 'Final Visit', 'BL' : 'Baseline', 'SC' : 'Screening', 'LOG' : 'U
ppmi_merge.to_csv('/Users/areardon/Desktop/ppmi_mergex.csv')


race_list = ['African.Berber.Race','Ashkenazi.Jewish.Race', 'Basque.Race','Hispanic.Latino.Race', 'Asian
for col_name in race_list :
    ppmi_merge[col_name].replace({0 : 'No', 1 : 'Yes', 2 : np.NaN}, inplace = True)

# Fill in cells that are NA with subject information that we know from other event ids/rows for fixed v
fixed_var_list = ['Enroll.Diagnosis', 'Enroll.Subtype','Consensus.Diagnosis', 'Consensus.Subtype', 'Subj
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_nam
            if fixed_var_value.any() :
                ppmi_merge.loc[row_num, col_name] = fixed_var_value[0] # fill in baseline value at NA
ppmi_merge = ppmi_merge.rename(columns = {'PATNO' : 'Subject.ID'})
ppmi_merge.replace({'NA' : np.nan}, inplace = True)

race_list = ['African.Berber.Race','Ashkenazi.Jewish.Race', 'Basque.Race','Hispanic.Latino.Race', 'Asian
for col_name in race_list :
    ppmi_merge[col_name].replace({'No' :False , 'Yes' : True, 2 : np.NaN}, inplace = True)


#### GENETICS INFO ####
```

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

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

    # 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 mer
    genetics_df['POS'] = 'POS' + genetics_df['POS'].astype(str) # Need to be strings before you use mer
    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 na
    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'], ho
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 fil
ppmi_t1_df.rename(columns = {'u_hier_id_OR': 'Subject.ID'}, inplace = True) # Rename subject id column

# 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 = "lef
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
analytic = ppmi_merge[ppmi_merge['Analytic.Cohort'] == 'Analytic Cohort'] # Split up Analytic cohort df
not_analytic = ppmi_merge[ppmi_merge['Analytic.Cohort'] == 'Not Analytic Cohort'] # Split up Analytic c
participant_status = pd.read_csv(ppmi_download_path + 'Participant_Status.csv') # Read in participant_s
participant_status = participant_status[['PATNO', 'COHORT_DEFINITION']] # Keep only
participant_status.rename(columns = {'PATNO' : 'Subject.ID', 'COHORT_DEFINITION' : 'Enroll.Diagnosis'},
not_analytic_participant_status = pd.merge(not_analytic, participant_status, on = ['Subject.ID'], how =
not_analytic_participant_status.drop(['Enroll.Diagnosis_x'], axis = 1, inplace = True) # Remove the ext
not_analytic_participant_status.rename(columns = {'Enroll.Diagnosis_y' : 'Enroll.Diagnosis'}, inplace =
not_analytic_participant_status.loc[not_analytic_participant_status['Enroll.Diagnosis'] == 'Healthy Con

# 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 e
ppmi_merge = ppmi_merge.sort_values(by = ['Subject.ID','Event.ID.Date']) # Sort values by subject and e
ppmi_merge['Event.ID.Date'] = ppmi_merge['Event.ID.Date'].astype(str) # Change Event.ID.Date back to st

# 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
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] + '-'
ppmi_merge = pd.merge(ppmi_merge, ppmi_qc_BA, on = ['ImageID'], how = "left") # Merge - keep only from


#### BILATERAL SUBTYPE SCORES (Tremor and Brady) ####
brady_right3 = ['Dominant.Side.Disease', 'Rigidity.Neck.UPDRS3.Num', 'Rigidity.RUE.UPDRS3.Num', 'Rigidi
brady_right3a = ['Dominant.Side.Disease', 'Rigidity.Neck.UPDRS3A.Num', 'Rigidity.RUE.UPDRS3A.Num', 'Rig
brady_right3ON = ['Dominant.Side.Disease', 'Rigidity.Neck.UPDR3ON.Num', 'Rigidity.RUE.UPDR3ON.Num', 'Ri
brady_right3OF = ['Dominant.Side.Disease', 'Rigidity.Neck.UPDR3OF.Num', 'Rigidity.RUE.UPDR3OF.Num', 'Ri

brady_left3 = ['Dominant.Side.Disease', 'Rigidity.Neck.UPDRS3.Num', 'Rigidity.LUE.UPDRS3.Num', 'Rigidity
brady_left3a = ['Dominant.Side.Disease', 'Rigidity.Neck.UPDRS3A.Num', 'Rigidity.LUE.UPDRS3A.Num', 'Rigi
brady_left3ON = ['Dominant.Side.Disease', 'Rigidity.Neck.UPDR3ON.Num', 'Rigidity.LUE.UPDR3ON.Num', 'Rig
brady_left3OF = ['Dominant.Side.Disease', 'Rigidity.Neck.UPDR3OF.Num', 'Rigidity.LUE.UPDR3OF.Num', 'Rig

brady_sym3 = ['Dominant.Side.Disease', 'Rigidity.Neck.UPDRS3.Num', 'Rigidity.RUE.UPDRS3.Num', 'Rigidity
brady_sym3a = ['Dominant.Side.Disease', 'Rigidity.Neck.UPDRS3A.Num', 'Rigidity.RUE.UPDRS3A.Num', 'Rigid
brady_sym3ON = ['Dominant.Side.Disease', 'Rigidity.Neck.UPDR3ON.Num', 'Rigidity.RUE.UPDR3ON.Num', 'Rigi
brady_sym3OF = ['Dominant.Side.Disease', 'Rigidity.Neck.UPDR3OF.Num', 'Rigidity.RUE.UPDR3OF.Num', 'Rigi

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

def add_lateralized_subscores(df : pd.DataFrame, subscore_side_list : list, side : str, new_col_name : s
    """
    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 i
    subscore_side_temp = subscore_side.drop('Dominant.Side.Disease',1) # Drop dominant side of disease
    idx  = subscore_side_temp.loc[pd.isnull(subscore_side_temp).any(1), :].index
    df[new_col_name] = subscore_side_temp.sum(axis = 1) # Sum of all columns in each row where dom side
    df[new_col_name].iloc[idx] = np.nan # Fill in subscores that should be nans as nan

    return df
```

```
def add_symmetric_subscores(df : pd.DataFrame, subscore_side_list : list, side : str, new_col_name : st
    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 i
    subscore_side_temp = subscore_side.drop('Dominant.Side.Disease',1) # Drop dominant side of disease
    idx  = subscore_side_temp.loc[pd.isnull(subscore_side_temp).any(1), :].index
    x = subscore_side_temp.fillna(0) # because adding 1 plus nan equals nan
    if "Brady" in new_col_name :
        df[new_col_name] = x['Rigidity.Neck' + ext ] + (x['Rigidity.RUE' + ext] + x['Rigidity.LUE'+ ext
        df[new_col_name].iloc[idx] = np.nan # Fill in subscores that should be nans as nan
    elif "Tremor" in new_col_name :
        df[new_col_name] = x['Tremor.UPDRS2.Num'] +  (x['Postural.Tremor.Right.Hand' + ext ]  + x['Postu
        df[new_col_name].iloc[idx] = np.nan
    return df

ppmi_merge = add_lateralized_subscores(ppmi_merge, brady_left3, 'Left', 'Brady.Rigidity.Subscore-left.U
ppmi_merge = add_lateralized_subscores(ppmi_merge, brady_right3, 'Right', 'Brady.Rigidity.Subscore-right
ppmi_merge = add_symmetric_subscores(ppmi_merge, brady_sym3, 'Symmetric', 'Brady.Rigidity.Subscore-sym.U

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

#### TREMOR ####
tremor_right3 = ['Dominant.Side.Disease', 'Tremor.UPDRS2.Num', 'Postural.Tremor.Right.Hand.UPDRS3.Num',
tremor_left3 = ['Dominant.Side.Disease', 'Tremor.UPDRS2.Num', 'Postural.Tremor.Left.Hand.UPDRS3.Num', '
tremor_sym3 = ['Dominant.Side.Disease', 'Tremor.UPDRS2.Num', 'Postural.Tremor.Right.Hand.UPDRS3.Num', '

ppmi_merge = add_lateralized_subscores(ppmi_merge, tremor_left3, 'Left', 'Tremor.Subscore-left.UPDRS3')
ppmi_merge = add_lateralized_subscores(ppmi_merge, tremor_right3, 'Right', 'Tremor.Subscore-right.UPDRS
ppmi_merge = add_symmetric_subscores(ppmi_merge, tremor_sym3, 'Symmetric', 'Tremor.Subscore-sym.UPDRS3'


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

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

ppmi_merge = fill_non_lat_subscore(ppmi_merge, 'Tremor.Subscore.lateralized.UPDRS3', 'Tremor.Subscore.U
ppmi_merge = fill_non_lat_subscore(ppmi_merge, 'Brady.Rigidity.Subscore.lateralized.UPDRS3', 'Brady.Rig


## Inlcude columns for bestEventID (bestScreening, bestBaseline, etc) and denote the highest resnetGrad
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
        if len(selu) == 1 : # If there is one event id for that subject
            idx = selu.index # Ge the index
            ppmi_merge.loc[idx, mybe] = True
        if len(selu) > 1 : # IF there is more than one event id for that subject and resnet grade is no
            maxidx = selu[['resnetGrade']].idxmax() # Get the higher resnetGrade for each visit if ther
            ppmi_merge.loc[maxidx, mybe] = True

## Include a column for bestAtImage.Acquisition.Date - denote the one or highest resnetGrade with True
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
        if len(selu) == 1 : # If there is one event id for that subject
            idx = selu.index # Get the index
            ppmi_merge.loc[idx, 'bestAtImage.Acquisition.Date'] = True
        if len(selu) > 1 : # IF there is more than one event id for that subject and resnet grade is no
            maxidx = selu[['resnetGrade']].idxmax() # Get the higher resnetGrade for each visit if ther
            ppmi_merge.loc[maxidx, 'bestAtImage.Acquisition.Date'] = True

# DX simplified column
ppmi_merge['DXsimplified'] = '' # Initialize DXsimplfied
ppmi_merge.loc[ppmi_merge['Consensus.Subtype'] == 'Healthy Control', 'DXsimplified'] = 'HC'
ppmi_merge.loc[ppmi_merge['Enroll.Diagnosis'] == 'Healthy Control', 'DXsimplified'] = 'HC'
ppmi_merge.loc[ppmi_merge['Enroll.Diagnosis'] == 'Parkinson\'s Disease', 'DXsimplified'] = 'Sporadic_PD
ppmi_merge.loc[ppmi_merge['Enroll.Diagnosis'] == 'SWEDD', 'DXsimplified'] = "nonPDorMSA"
ppmi_merge.loc[ppmi_merge['Enroll.Diagnosis'] == "Prodromal", 'DXsimplified'] = "Sporadic_Pro"
ppmi_merge.loc[ppmi_merge['Consensus.Subtype'] == "GBA", 'DXsimplified'] = "GBA_HC"
ppmi_merge.loc[ppmi_merge['Consensus.Subtype'] == "Genetic : GBA", 'DXsimplified'] = "GBA_PD"
ppmi_merge.loc[ppmi_merge['Consensus.Subtype'] == "Genetic : GBA not Prodromal", 'DXsimplified'] = "GBA_
ppmi_merge.loc[ppmi_merge['Consensus.Subtype'] == "Genetic : GBA Prodromal", 'DXsimplified'] = "GBA_Pro
ppmi_merge.loc[ppmi_merge['Consensus.Subtype'] == "Genetic : LRRK2", 'DXsimplified'] = "LRRK2_PD"
ppmi_merge.loc[ppmi_merge['Consensus.Subtype'] == "Genetic : LRRK2 + GBA", 'DXsimplified'] = "LRRK2_PD"
ppmi_merge.loc[ppmi_merge['Consensus.Subtype'] == "Genetic : LRRK2 + GBA not Prodromal", 'DXsimplified']
ppmi_merge.loc[ppmi_merge['Consensus.Subtype'] == "Genetic : LRRK2 + GBA Prodromal", 'DXsimplified'] = '
ppmi_merge.loc[ppmi_merge['Consensus.Subtype'] == "Genetic : LRRK2 not Prodromal", 'DXsimplified'] = "LR
ppmi_merge.loc[ppmi_merge['Consensus.Subtype'] == "Genetic : LRRK2 Phenoconverted", 'DXsimplified'] = "L
ppmi_merge.loc[ppmi_merge['Consensus.Subtype'] ==  "Genetic : LRRK2 Prodromal", 'DXsimplified'] = "LRRK
ppmi_merge.loc[ppmi_merge['Consensus.Subtype'] ==  "Genetic : SNCA", 'DXsimplified'] = "SNCA_PD"
ppmi_merge.loc[ppmi_merge['Consensus.Subtype'] == "Genetic : SNCA Prodromal", 'DXsimplified'] = "SNCA_P
ppmi_merge.loc[ppmi_merge['Consensus.Subtype'] == "Hyposmia" , 'DXsimplified'] = "Sporadic_Pro"
ppmi_merge.loc[ppmi_merge['Consensus.Subtype'] == "Hyposmia : Phenoconverted", 'DXsimplified'] = "Sporad
ppmi_merge.loc[ppmi_merge['Consensus.Subtype'] == "No Mutation not Prodromal", 'DXsimplified'] = np.NaN
ppmi_merge.loc[ppmi_merge['Consensus.Subtype'] == "non-HC", 'DXsimplified'] = np.NaN
ppmi_merge.loc[ppmi_merge['Consensus.Subtype'] == "non-PD", 'DXsimplified'] = "nonPDorMSA"
ppmi_merge.loc[ppmi_merge['Consensus.Subtype'] == "RBD", 'DXsimplified'] = "Sporadic_Pro"
ppmi_merge.loc[ppmi_merge['Consensus.Subtype'] == "RBD : Phenoconverted", 'DXsimplified'] = "Sporadic_P
ppmi_merge.loc[ppmi_merge['Consensus.Subtype'] ==  "RBD : Phenoconverted with GBA", 'DXsimplified'] = "
ppmi_merge.loc[ppmi_merge['Consensus.Subtype'] ==  "Sporadic", 'DXsimplified'] = "Sporadic_PD"
ppmi_merge.loc[ppmi_merge['Consensus.Subtype'] ==  "SWEDD/non-PD Active", 'DXsimplified'] = "nonPDorMSA
ppmi_merge.loc[ppmi_merge['Consensus.Subtype'] ==  "SWEDD/PD Active", 'DXsimplified'] = np.NaN
```

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

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

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

subids = ppmi_merge['Subject.ID'].unique() # Unique subject ids
for subid in subids :
    df = ppmi_merge[(ppmi_merge['Subject.ID'] == subid) & (ppmi_merge['DXs2'] == 'PD') & (ppmi_merge['PD
    if len(df) > 0 :
        print(df['PD.Diagnosis.Duration'])
        mydd = min(df['PD.Diagnosis.Duration'])
        ppmi_merge.loc[ppmi_merge['Subject.ID'] == subid, 'PD.Min.Disease.Duration'] = mydd

## Add in Visit column
```

```
ppmi_merge['Visit'] = np.NaN # Initialize Visit col
searchfor = ['Baseline', 'Visit Month '] # Strings to search for
temp = ppmi_merge['Event.ID'].str.contains('|'.join(searchfor)) # locations of where row contains str:
ppmi_merge['Visit'].loc[temp == True] = ppmi_merge['Event.ID'].loc[temp == True] # Fill in 'Visit' col
ppmi_merge['Visit'] = ppmi_merge['Visit'].str.replace("Remote Visit Month ", "") # Replace Remote visit
ppmi_merge['Visit'] = ppmi_merge['Visit'].str.replace("Visit Month ", "") # Replace Visit month with ''
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
ppmi_merge['Visit'] = ppmi_merge['Visit'].astype(int) # str to int
ppmi_merge['Visit'] = ppmi_merge['Visit'].replace(9999, np.nan)  # Replace 9999 with nan

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