Python Extract

This document details my Python script to extract Medicare data. I'll go through it one important segment at a time; the entire script is at the bottom.

Start

First is a header with helpful metadata. The `#! /usr/bin/env python3` is just a standard way to reference python files that allows for the convenience of running them from the command line with `./filename.py`.

```
#! /usr/bin/env python3
"""
--------------------------------------------------------------------
Project: cms-age-discont
Program: 01mammogram_select.py
Author:  Kyle Barron <barronk@mit.edu>
Created: 12/12/2017, 5:15:29 PM
Updated: 12/12/2017, 5:15:29 PM
Purpose: Select data for age discontinuity analysis in a scalable way
Depends: data/raw_harm/PCTpct/op/
    opPCT_clms_raw_YEAR.dta
Output:  data/base/
    appts_30d_dob_mamm_2011.dta
"""
```

Importing packages

In Python, new functionality comes in the form of **packages**, which are bundles of code that allow you to do complicated things with few lines of code. I import five packages beyond the base language:

1. Add regular expression support for strings with the `re` package
2. Add support for working with rectangular dataframes with the `pandas` package. The code `import pandas as pd` means that I can reference the package with `pd` instead of writing `pandas` everywhere in my code.
3. Add `time` to allow me to time how long things take.
4. I import just the `path` command from the `os` package, which makes it easier to specify file paths.
5. I import the `relativedelta` command from the `dateutil` package, which makes it easy to deal with spans of time, like finding dates that are a month after a birthday.

```
import re
import pandas as pd
import time
from os import path
from dateutil.relativedelta import relativedelta
```

Make file paths

I create variables for `datadir`, `pct`, and `year`, because these values will change in the future. After, I create an object with the file paths for the demographic and outpatient data files. These are just like locals in Stata. The `path.join` function adds in a `/` (on Linux) whenever there's a `,`. So `op_path` holds the value `data/raw_harm/0001pct/op/op0001_clms_raw_2011.dta`.

```
datadir = 'data'
pct = '0001'
year = '2011'

# File paths:
idcharvar_path = path.join(
    datadir,
    'raw_harm',
    pct + 'pct',
    'denom_bene',
    'pop' + pct + '_demo_bene_idcharvar' + year + '.dta'
)
op_path = path.join(
    datadir,
    'raw_harm',
    pct + 'pct',
    'op',
    'op' + pct + '_clms_raw_' + year + '.dta'
)
```

Import Demographic Data

Next I want to import demographic data, so that I can match birth dates to `bene_ids`.

- The first line of code reads the entire Stata file at the path listed in `idcharvar_path`. It's basically the same as Stata's `use sdod sdob sex bene_id using `idcharvar_path'`. This is put into the object named `demo_bene`.
- The next line drops the rows for which `sex` is equal to `1`. In Python you *index* with `[]`. So `demo_bene['sex']` gives you just the single column of the data set `sex`. `demo_bene[demo_bene['sex'] == '1']` gives you the rows of the dataset for which `sex` is equal to 1. The command drops those rows.
- To minimize the dataset size before joining other data, I also drop rows for which the date of death is before the first day of the current year. So since `year` is currently set to `2011`, this drops all people who died before January 1, 2011.
- Lastly, I drop the columns `sex` and `sdod`.

```
# First get list of all bene_id date of birth combos for women
# Import demographic data
# This doesn't need to be iterated, because the 100% datasets are only ~2.5GB in size
demo_bene = pd.read_stata(idcharvar_path, columns = ['sdod', 'sdob', 'sex', 'bene_id'])

# Drop males
demo_bene.drop(demo_bene[demo_bene['sex'] == '1'].index, inplace = True)
```

```
# Drop if sdod is before year
demo_bene.drop(
    demo_bene[demo_bene['sdod'] < pd.Timestamp(year + '-01-01')].index,
    inplace = True
)
demo_bene.drop(['sex', 'sdod'], axis = 1, inplace = True)
```

Outpatient Encounters

Define columns to keep

The following generates the list of columns I want to keep from the outpatient claims file.

- The first line puts the list of all column names in the stata file into the object named `columns`.
- The second creates a list with three strings of variables that I want to keep.
- I want to keep all columns `icd_dgns_cd*` so that I can see if any of the diagnosis codes correspond to a mammogram. But Python doesn't recognize the use of the `*` there, so it returns an error. Instead, I loop over all the columns in `columns`. If the column name matches the regular expression string `icd_dgns_cd\d+`, which means the text `icd_dgns_cd` plus one or more digits, then it adds the variable name to the list `tokeep`.

!!! note For loops and if statements in Python rely on indentation to determine when the clause finishes. If a statement is meant to run after a loop, it must be placed at the same indentation level as the initial for loop statement.

```
# Find all op encounters by these people within 30 days of their birthday (ages 66-80)
## First, import outpatient data
# Create list of columns to retrieve from stata file
columns = list(pd.read_stata(op_path, iterator = True).variable_labels().keys())
tokeep = ['bene_id', 'clm_id', 'from_dt']
for column in columns:
    if re.search(r'icd_dgns_cd\d+', column):
        tokeep.append(column)
```

Loop over rows in outpatient files

This is the main part of the code.

- `df_extracted = []` creates an empty list. This list will hold all the clean data from each iteration of the for loop.
- `itr = pd.read_stata(op_path, columns = tokeep, chunksize = 10000)` creates an object that I can loop over. For each iteration of this loop, Python will read the Stata file at path `op_path`, the columns defined by `tokeep`, and 10,000 rows at a time. This is equivalent to `` use `tokeep' using `op_path' in 1/10000 `` in Stata (with a bit more finesse needed for accurately looping over the entire dataset 10,000 rows at a time).
- `for df in itr:` starts the for loop. `itr` is the name of the object I'm looping over, created in the previous statement. Since I use the name `df` in the for loop, the rest of the loop will be coded in terms of `df`. Each use of `df` refers to one chunk of 10,000 rows of the Stata file.
- `df = df.merge(demo_bene, on = 'bene_id')` merges the `demo_bene` dataset created earlier to the outpatient file. This is a **left join**, akin to `merge n:1 bene_id using demo_bene, keep(master)`. So I'm only left with the observations in `df` that have a match in `demo_bene`.

I'll add an assert statement later to make sure that the number of observations in `df` before and after the merge is equal.

- `df['diff'] = df.apply(lambda row: relativedelta(row['from_dt'], row['sdob']), axis = 1)` adds a new variable to `df` named `diff`. `df.apply(function, axis = 1)` means to apply a function to every row of the dataset. The function here is `lambda row: relativedelta(row['from_dt'], row['sdob'])`, which just means, for each row of the dataset, use the values of `from_dt` and `sdob` in the function `relativedelta`. This outputs a special type of object that is the difference in time between the two dates. In Stata, subtracting two dates gives you a constant: the number of days between them. But this special type in Python understands the calendar better. Doing something like `relativedelta('2011-05-25', '2005-01-13')` will give output that understands that was 6 years, 4 months, and 12 days, and means you don't have to rely on there being 365.25 days in a year.
- The next part keeps only the rows that meet specified criteria. This restricts the entire dataset `df` to those for whom the value of the `diff` column is at least 66 years and less than 80 years, and for whom the number of months is equal to zero. This means I'm restricting currently to those people whose encounter is no more than *a month* after their birthday, instead of *30 days* after. I can change this if you want.

```
df = df[
    (df['diff'].apply(lambda x: x.years) >= 66) &
    (df['diff'].apply(lambda x: x.years) <= 80) &
    (df['diff'].apply(lambda x: x.months) == 0)]
```

- Next I create an indicator variable for if the encounter contained a mammogram diagnosis. I create a variable `has_mammogram` that's `False` by default. I loop over all columns in the dataset that start with `icd_dgns_cd`. If a the value of `icd_dgns_cd*` in a row is either `V7612`, `V7611`, or `79380`, it makes the value of `has_mammogram` `True` for that row. It then drops each column as it goes, since those columns are no longer needed.
- We now have a clean dataset in `df`. But since this is in a for loop, we have to save `df` somehow, otherwise it'll be overwritten in the next iteration of the loop. `df_extracted.append(df)` adds `df` to the empty list `df_extracted` we created at the beginning of this section.
- Once the for loop finishes, each element of the list `df_extracted` is a rectangular data frame. The command `df_extracted = pd.concat(df_extracted, axis = 0)` *flattens* the list. It turns the list of data frames into a single data frame, concatenating each element.

```
df_extracted = []
itr = pd.read_stata(op_path, columns = tokeep, chunksize = 10000)
for df in itr:
    # I now have outpatient data; merge birthdays on `bene_id`
    # This does a *left join* by default
    df = df.merge(demo_bene, on = 'bene_id')

    # Keep if the appointment is within 30 days after their birthday.
    df['diff'] = df.apply(lambda row: relativedelta(row['from_dt'], row['sdob']), axis = 1)

    df = df[
        (df['diff'].apply(lambda x: x.years) >= 66) &
        (df['diff'].apply(lambda x: x.years) <= 80) &
        (df['diff'].apply(lambda x: x.months) == 0)]

    df['has_mammogram'] = False
    cols = [col for col in df if col.startswith('icd_dgns_cd')]
    # Test all columns in cols, and if text matches regex, makes 'has_mammogram' = True
    ## Mammogram values:
    ## V76.12 'Other Screening mammogram'
    ## 793.80 'Abnormal mammogram, unspecified'
    ## V76.11 'Screening mammogram for high-risk patient'
    for col in cols:
        df.loc[df[col].str.contains(r'V761(?:2|1)|79380'), 'has_mammogram'] = True
        df.drop(col, axis = 1, inplace = True)

    df_extracted.append(df)
    # break

df_extracted = pd.concat(df_extracted, axis = 0)
```

Export to Stata file

We now have the data frame `df_extracted` with the data we want from the entire outpatient data. We just need to export this data back to Stata format.

- First, I drop the `diff` column, because it holds the special type corresponding to the difference in datetimes, and this type isn't supported in Stata.
- Then I change the `has_mammogram` column from having values `True`/`False` to having values `1`/`0`.
- Next is the actual export the the Stata file. I export to the file path `data/base/appts_30d_dob_mamm_2011.dta`, I ask it to convert the two date columns to Stata's `td` format, and I ask it to not add the *index* column. The index column is an extra column in the Pandas package that just refers to `_n` in Stata.
- Last I record the end time of the program and show that value. The 1% sample in 2011 took about 500 seconds.

```
df_extracted.drop('diff', axis = 1, inplace = True)
df_extracted.has_mammogram = df_extracted.has_mammogram.astype(int)

df_extracted.to_stata(
    path.join(datadir, 'base', 'appts_30d_dob_mamm_2011.dta'),
    convert_dates = {
        'from_dt': 'td',
        'sdob': 'td'
    },
    write_index = False)

t1 = time.time()
print(t1 - t0)
```

Entire script

```
#! /usr/bin/env python3
"""
---------------------------------------------------------------------
```

```
Project: cms-age-discont
Program: 01mammogram_select.py
Author:  Kyle Barron <barronk@mit.edu>
Created: 12/12/2017, 5:15:29 PM
Updated: 12/12/2017, 5:15:29 PM
Purpose: Select data for age discontinuity analysis in a scalable way
Depends: data/raw_harm/PCTpct/op/
    opPCT_clms_raw_YEAR.dta
Output:  data/base/
    appts_30d_dob_mamm_2011.dta
"""

import re
import pandas as pd
import time
from os import path
from dateutil.relativedelta import relativedelta

t0 = time.time()

datadir = 'data'
pct = '0001'
year = '2011'

# File paths:
idcharvar_path = path.join(
    datadir,
    'raw_harm',
    pct + 'pct',
    'denom_bene',
    'pop' + pct + '_demo_bene_idcharvar' + year + '.dta'
)
op_path = path.join(
    datadir,
    'raw_harm',
    pct + 'pct',
    'op',
    'op' + pct + '_clms_raw_' + year + '.dta'
)

# 1) Unbalanced panel:
# I want anybody who had outpatient visit within 30 days of when they turn 66 - 80
# What fraction of those had mammograms?

# First get list of all bene_id date of birth combos for women
# Import demographic data
# This doesn't need to be iterated, because the 100% datasets are only 2.5 - 3 GB ish in size
demo_bene = pd.read_stata(idcharvar_path, columns = ['sdod', 'sdob', 'sex', 'bene_id'])

# Drop males
demo_bene.drop(demo_bene[demo_bene['sex'] == '1'].index, inplace = True)
```

```python
# Drop if sdod is before year
demo_bene.drop(
    demo_bene[demo_bene['sdod'] < pd.Timestamp(year + '-01-01')].index,
    inplace = True
)
demo_bene.drop(['sex', 'sdod'], axis = 1, inplace = True)

# Find all op encounters by these people within 30 days of their birthday (ages 66-80)
## First, import outpatient data
# Create list of columns to retrieve from stata file
columns = list(pd.read_stata(op_path, iterator = True).variable_labels().keys())
tokeep = ['bene_id', 'clm_id', 'from_dt']
for column in columns:
    if re.search(r'icd_dgns_cd\d+', column):
        tokeep.append(column)

df_extracted = []
itr = pd.read_stata(op_path, columns = tokeep, chunksize = 10000)
for df in itr:
    # I now have outpatient data; merge birthdays on `bene_id`
    # This does a *left join* by default
    df = df.merge(demo_bene, on = 'bene_id')

    # Keep if the appointment is within 30 days after their birthday.
    df['diff'] = df.apply(lambda row: relativedelta(row['from_dt'], row['sdob']), axis = 1)

    df = df[
        (df['diff'].apply(lambda x: x.years) >= 66) &
        (df['diff'].apply(lambda x: x.years) <= 80) &
        (df['diff'].apply(lambda x: x.months) == 0)]

    df['has_mammogram'] = False
    cols = [col for col in df if col.startswith('icd_dgns_cd')]
    # Test all columns in cols, and if text matches regex, makes 'has_mammogram' = True
    ## Mammogram values:
    ## V76.12 'Other Screening mammogram'
    ## 793.80 'Abnormal mammogram, unspecified'
    ## V76.11 'Screening mammogram for high-risk patient'
    for col in cols:
        df.loc[df[col].str.contains(r'V761(?:2|1)|79380'), 'has_mammogram'] = True
        df.drop(col, axis = 1, inplace = True)

    df_extracted.append(df)
    # break

df_extracted = pd.concat(df_extracted, axis = 0)

df_extracted.drop('diff', axis = 1, inplace = True)
df_extracted.has_mammogram = df_extracted.has_mammogram.astype(int)
df_extracted.to_stata(
    path.join(datadir, 'base', 'appts_30d_dob_mamm_2011.dta'),
    convert_dates = {
        'from_dt': 'td',
        'sdob': 'td'
    },
    write_index = False)

t1 = time.time()
print(t1 - t0)
# 525.5
```