Python and SAS for advanced analytics: A comparative case study (initial draft for feedback, 2/23/20)

Doug Thompson, PhD, Director of Advanced Analytics, Rush Health, Chicago IL

Introduction

The past few decades have seen the emergence of many software tools for advanced analytics. SAS was among the earlier ones and was arguably the gold standard for advanced analytics 20 or so years ago. Since then, several other tools for advanced analytics have entered the scene, most notably R and Python, both of which are freely available open source software, in contrast to SAS which is commercial software. Python in particular has gained popularity for advanced analytics in the past few years.

Many analysts have opinions about their preferred software, which are frequently expressed on the Internet and in other forums. While opinions abound regarding the relative merits of Python and SAS for advanced analytics, there have not been a lot of case studies comparing the two in specific advanced analytics examples. Such case studies provide a more concrete sense of the similarities and differences, facilitating more informed choices based on side-by-side comparison as opposed to opinions which may or may not be well-informed. The purpose of this paper is to describe such as case study.

The focus here is on advanced analytics applications of Python and SAS. Both software packages have many other uses, including data visualization, reporting, and data management. These will not be addressed. The comparisons of advanced analytics capabilities do not necessarily apply for other types of analyses; for example, some would say that Tableau (another commercial software package) is excellent for data visualization but crude for advanced analytics.

The case study involves a common task in advanced analytics: Estimating the association between two variables, after adjusting for potential confounders using regression techniques. The context of the case study is healthcare. Healthcare data is highly sensitive and protected by legal regulations. To avoid risks involved with sensitive data, the case study uses publically available healthcare cost and utilization data from the Medical Expenditure Panel Survey (MEPS). MEPS data includes fields similar to what a healthcare analyst might use if more granular healthcare data (e.g., claims or electronic medical records) were summarized at the person-year level.

Specifically, the case study looks at the association between having a primary healthcare provider (often abbreviated “PCP” for primary care provider) and 1) prior personal characteristics that may explain having a PCP (as opposed to not having one), as well as 2) subsequent healthcare expenditures. Theoretically, the PCP is supposed to coordinate a patient’s care across all providers (e.g., a variety of specialists) which is thought to foster better, more efficient healthcare, leading to better health outcomes as well as lower healthcare expenditures. Partly based on this theory, the U.S. Federal government (via the Centers for Medicare and Medicaid Services, AKA CMS) as well as commercial health insurers have heavily promoted the role of the PCP in healthcare through service delivery models such as Accountable Care Organizations.

In both Python and SAS, a given analysis can be conducted in many different ways. The analyses illustrated below show one way to conduct the analyses, but not necessarily the best or most efficient way.

Methods

Parallel analyses of MEPS data were conducted using Python and SAS. The Python analyses were conducted first, then mirrored using SAS. The SAS analyses generally followed the same steps as Python, with some exceptions (e.g., output formatting).

MEPS has been conducted annually since 1996 by the Agency for Healthcare Research and Quality, which is part of the U.S. Department of Health and Human Services. MEPS was designed to describe healthcare expenditures, healthcare utilization and health insurance among the U.S. non-institutionalized, non-military population. MEPS samples households. Information regarding each sampled household is collected for a 2-year period ("panel") in 5 "rounds" of interviews spaced across 2.5 years. This enables longitudinal analysis of healthcare for individuals in the sampled households during the 2-year period covered in the panel. MEPS consists of a series of overlapping panels. The data are freely available for download. Instructions for importing the data into SAS are available on the MEPS website. Although the data are de-identified to protect respondent anonymity, an individual's data can be tracked across time by tying it to an individual person ID ("DUPERSID").

The analyses in this paper used MEPS Panel 20, covering 2015 ("Y1") and 2016 ("Y2"). For some measures, Round 2 ("R2") is used to represent 2015 and Round 4 ("R4") is used to represent 2016. Data were limited to Panel 20 who had health insurance in both 2015 and 2016, and who had data in both 2015 and 2016 (the latter condition was true of the vast majority of panel 20 participants).

Two sets of analyses were conducted. Analysis 1 examined the association between personal characteristics in 2015 and having (or not having) a primary healthcare provider (PCP) in 2016. The personal characteristics included age, household income, medical expenditures, self-reported health, and whether or not the person had a managed care or "gatekeeper" type health insurance plan (which theoretically should encourage a PCP relationship). It was hypothesized that individuals with a managed care or gatekeeper-type insurance plan in 2015 would more likely to have a usual healthcare provider in 2016; it was also hypothesized that older and sicker individuals in 2015 would be more likely to have a usual healthcare provider in 2016. Analysis 2 examined the association between having a usual healthcare provider in 2015 and medical expenditures in 2016, adjusting for personal characteristics in 2015. It was hypothesized that having a usual healthcare provider in 2015 would be associated with lower total medical expenditures in 2016, after adjusting for individuals' age, health, 2015 medical expenditures and other factors.

Logistic regression was the primary modeling technique used in Analysis 1, while ordinary least squares regression was the primary modeling technique used in Analysis 2. In Analysis 2, expenditures were modeled on the raw scale as well as log transformed.

The analyses did not use the MEPS survey weights nor other survey design variables (strata, primary sampling units or PSUs). For studies examining associations among variables, survey weights are sometimes ignored, under the assumption that relationships among variables will not be materially impacted by differential observation weighting. Variance estimation was of only minor interest in the analysis.

Results

Each step in Analyses 1 and 2 is described, starting with Python, followed by parallel analyses using SAS.

1. Read in data

Python

MEPS data is published as SAS datasets. Python converts SAS datasets into Python data frames using simple syntax as illustrated below (pd.read_sas). Towards the beginning of a program, it is also convenient to import Python modules. The pandas module includes the primary functionality illustrated in this paper. The MEPS person ID "DUPERSID" is set as an index in the data frame "df" (setting an index is optional).

```
import pandas as pd

import numpy as np

df = pd.read_sas('C:\projects\MEPS\h193.sas7bdat')

df.set_index(['DUPERSID'],drop=False,inplace=True)
```

SAS

The permanent MEPS dataset mepsdat.h193 is read into a temporary dataset df (the SAS "work" library is implicitly called in this syntax). For use in subsetting, a new variable df2_ind is created, coded as 1 if the person participated in both the 2015 and 2016 surveys (YEARIND=1), and was in panel 20 and had health insurance throughout 2015 (INSURCY1=1) and 2016 (INSURCY2=1). A note is copied from the log indicating the number of observations (rows) and variables (columns). The SAS log by default gives the information that is called in Python shape, as illustrated below.

The code in this paper illustrates comments as well as executed commands; in Python, comments can be made with lines beginning with "#", while in SAS one way to make comments is to use a block of text beginning with "/*" and ending with "*/".

```
libname mepsdat 'C:\projects\MEPS';

data df;

set mepsdat.h193;

df2_ind = (YEARIND=1 and PANEL=20 and (INSURCY1 not in(-1,3,7)) and (INSURCY2 not in(-
1,3,7)));

run;

/*

NOTE: There were 17017 observations read from the data set MEPSDAT.H193.

NOTE: The data set WORK.DF has 17017 observations and 3592 variables.

*/
```

2. Select in-scope cases

Python

After the Python comment lines, in-scope rows are selected and read into data frame "df2". This is done using the .loc syntax (in Python, as in SAS, there are several ways to select subsets of rows and this is just one way). Rows representing individuals with data in both 2015 and 2016, in panel 20 and with

health insurance in both years are selected. The syntax is roughly similar to the SAS syntax used to define “df2_ind” but with a little more typing.

```
# Limit to yearind=1 (respondent in both 2015 and 2016) and panel=20 and

# had insurance in both years

df2 = df.loc[(df['YEARIND']==1) & (df['PANEL']==20) & (df['INSURCY1'].isin([-
1,3,7])==False) &

             (df['INSURCY2'].isin([-1,3,7])==False)]
```

The next few lines are for checking the resulting data frame (df2). Shape is similar to the default information output to the SAS log when a dataset is created, i.e., the number of rows and columns. The frequency of each value of each variable in df2 is checked to ensure that no rows with out-of-scope values are retained (e.g., -1, 3 or 7 for INSURCY1). Combining Python print statements with methods such as pd.value_counts, which displays the frequency of each level of a categorical variable, provides a nice, flexible way of displaying and annotating a variety of results.

```
print('df shape:', df.shape)

print('df2 shape:', df2.shape)

# Check that values are being properly excluded

print('INSURCY1 in df...', '\n', pd.value_counts(df[' INSURCY1']), '\n',

      ' INSURCY1 in df2...', '\n', pd.value_counts(df2['INSURCY1']), '\n')
```

Below is the output of the Python commands above. This shows the rows retained after keeping only the in-scope cases and confirms (using INSURCY1 as an example) that rows with out-of-scope values such as -1.0 are excluded from df2.

```
df shape: (17017, 3591)
df2 shape: (14422, 3591)


INSURCY1 in df...
  1.0    7943
 2.0    4698
 3.0    1870
 5.0    1023
 4.0     789
 6.0     417
-1.0     230
 7.0      25
 8.0      22
Name: INSURCY1, dtype: int64
 INSURCY1 in df2...
 1.0    7705
2.0    4556
5.0     995
4.0     741
6.0     403
8.0      22
```

SAS

The variable df2_ind defined when df was created is used in a where clause to retain only the in-scope observations within df2. The SAS log (pasted in between "/*" and "*/") shows that 14,422 observations (rows) are retained, exactly the same result as df2.shape in Python.

This is also a convenient place within the SAS code to define two new indicators, that is, variables coded as 1 or 0: has_usc_R2 and has_usc_R4. These are coded as 1 = has usual healthcare provider in Round 2 (or Round 4 respectively) vs. 0 = does not have a usual healthcare provider. Having a usual healthcare provider is indicated by HAVEUS2 = 1; importantly, we also include the requirement that the usual healthcare provider be a person as opposed to facility (PROVTY2 either 2 or 3) and that person must be either a physician with specialty of internal medicine, family medicine or pediatrics, or must be a nurse practitioner or physician assistant (TYPEPE2 1,2,3,9, or 10). This is how primary care physicians are frequently defined in Accountable Care Organizations.

```
data df2;
set df(where=(df2_ind=1));
has_usc_R2 = (HAVEUS2=1 and (PROVTY2 in(2,3)) and (TYPEPE2 in(1,2,3,9,10)));
has_usc_R4 = (HAVEUS4=1 and (PROVTY4 in(2,3)) and (TYPEPE4 in(1,2,3,9,10)));
run;
/*
NOTE: There were 14422 observations read from the data set WORK.DF.
      WHERE df2_ind=1;
NOTE: The data set WORK.DF2 has 14422 observations and 3592 variables.
*/
```

3. Define the main grouping variable (has usual care provider)

Python

This is the place where we define HAVEUS2 and HAVEUS4 in the Python program. The syntax used to define these indicators is roughly similar to the SAS syntax used above.

```
# Define having usual care providers in R2 and R4

# HAVEUS2=has usual care provider, PROVTY2=usual care provider is person (as opposed
to facility), and

# TYPEPE2 is usual care provider is MD (family med, internal med, peds) or NP or PA

df2['has_usc_R2'] = ((df2['HAVEUS2']==1) & (df2['PROVTY2'].isin([2,3])) &

                     (df2['TYPEPE2'].isin([1,2,3,9,10])))*1


df2['has_usc_R4'] = ((df2['HAVEUS4']==1) & (df2['PROVTY4'].isin([2,3])) &

                     (df2['TYPEPE4'].isin([1,2,3,9,10])))*1
```

In the Python code, some descriptive analyses were done with HAVEUS2 and HAVEUS4. One would expect there to be an association between having a usual healthcare provider in two consecutive years – one's primary healthcare provider is theoretically supposed to be someone with whom one has a long-term relationship, who coordinates all of one's healthcare, who handles preventive services, and so forth. Pd.crosstab is a method that provides a cross-tabluation of HAVEUSR2 vs. HAVEUSR4 – the fact that the counts are heavily concentrated on the diagonals suggests a strong association. Syntax is also shown to compute the percentage of individuals who had a usual healthcare provider in Round 2 and

who also had a usual healthcare provider in Round 4 – it is 69.72%. Most of those who had a usual healthcare provider in 2015 also had one in 2016, as one would expect.

```
# Crosstab having usual care provider in R2 vs R4

print('What is the association of have_usc_R2 and have_usc_R4?','\n',

      pd.crosstab(df2['has_usc_R2'],df2['has_usc_R4'],margins=True),'\n')


# Of those with a usual care provider in R2, what percent also had a usual care
provider in R4?

had_usc_r2 = df2.loc[(df2['has_usc_R2']==1)]

had_usc_r2 = had_usc_r2[['has_usc_R4']]

x=had_usc_r2.mean().astype(float).map("{:.2%}".format)

print('Percent with USC in R2 who also had USC in R4: ',x)
```

```
What is the association of have_usc_R2 and have_usc_R4?
 has_usc_R4     0     1    All
has_usc_R2
0           8423  1402   9825
1           1392  3205   4597
All         9815  4607  14422

Percent with USC in R2 who also had USC in R4:  has_usc_R4    69.72%
dtype: object
```

SAS

The grouping variables were already defined when creating df2, as described above. The code below shows one possible SAS syntax to quantify the association between HAVEUS2 and HAVEUS4 – all of the desired results are provided by default in SAS PROC FREQ. The PROC FREQ results exactly replicate the Python results shown above, as one can see in the results table. The highlighted result in the table indicates that 69.72% of those with a usual healthcare provider in Round 2 also had one in Round 4. Although the default results of PROC FREQ provide a lot of good information, it can be too much information (e.g., row, column and total percentages in every step). One can use options in PROC FREQ to limit the default output.

```
ods rtf file='C:\projects\SAS_python_compare\crosstab_usucare.rtf';
proc freq data=df2;
tables has_usc_R2*has_usc_R4;
run;
ods rtf close;
```

Table of has_usc_R2 by has_usc_R4			
has_usc_R2	has_usc_R4		
Frequency Percent Row Pct Col Pct	0	1	Total
0	8423 58.40 85.73 85.82	1402 9.72 14.27 30.43	9825 68.13
1	1392 9.65 30.28 14.18	3205 22.22 69.72 69.57	4597 31.87
Total	9815 68.06	4607 31.94	14422 100.00

4. Create adjustment variables

This is by far the most intensive part of the analysis from a coding perspective, due to the need to handle certain MEPS values such as "refused" or "don't know" (often denoted with negative sign in the MEPS data). One strategy is to replace such values with imputed values so that the cases do not get dropped in subsequent regression modeling analyses. It can be debated whether it is optimal to use the median or mode to replace responses such as "don't know". For certain purposes such as predictive modeling, especially when such values are rare, it may be fine to handle them this way. For other purposes, this may not be an optimal approach. Regardless, it is good to know how to execute such imputation, so code to execute it is illustrated here.

Python

First a Python data frame, including just the analytic adjustment variables and the person ID, is created (analysis1_predictors).

```
# Define variables that will be used for adjustment

analysis1_predictors =
df2[['DUPERSID','RTHLTH2','MNHLTH2','TOTEXPY1','AGEY1X','FAMINCY1',

                            'MCRPHOY1','MCDHMOY1','MCDMCY1','PRVHMOY1']]

print(analysis1_predictors.shape)

# print(analysis1_predictors.head())
```

Next, a custom function is created to convert to missing ("None") the values that are less than 0, which is how MEPS marks "refused," "don't know" and similar values. Variables with negative values converted to missings have the same names as the original variables, except the variable name starts with an underscore (e.g., RTHLTH2 becomes _RTHLTH2). The respective variable sets are listed in original_vars and recoded_vars.

```
# Create recoded variables, denoted by _ at the beginning, where non-informative
values are represented as missing values

def convert_to_missing(var_new,var):

    analysis1_predictors[var_new] = analysis1_predictors[var]

    analysis1_predictors.loc[analysis1_predictors[var_new]<0,var_new]=None


original_vars = ['RTHLTH2','MNHLTH2','TOTEXPY1','AGEY1X','FAMINCY1',
                             'MCRPHOY1','MCDHMOY1','MCDMCY1','PRVHMOY1']
recoded_vars = ['_RTHLTH2','_MNHLTH2','_TOTEXPY1','_AGEY1X','_FAMINCY1',
                             '_MCRPHOY1','_MCDHMOY1','_MCDMCY1','_PRVHMOY1']
```

A for loop is used to run through the function for all pairs of variables.

```
for x,y in zip(original_vars,recoded_vars):

    convert_to_missing(y,x)
```

Next, a function is created to validate the results of the process above, ensuring that it worked as intended. This compares the variable frequency distribution before vs. after negative values are replaced with missing values.

```
# Check selected variables to make sure that the conversion to missing worked as
intended

def validate_missing_conversion(x,y):

    print('Original distribution:',x,pd.value_counts(analysis1_predictors[x]),'\n',

          'New distribution:',y,pd.value_counts(analysis1_predictors[y]),'\n')


validate_missing_conversion('RTHLTH2','_RTHLTH2')

validate_missing_conversion('MCRPHOY1','_MCRPHOY1')
```

At this point, a new variable is created indicating whether the person's health insurance was a gatekeeper-type plan or not; in this case, mgd_care_ins_R2 is coded as 1. If the insurance plan is not a gatekeeper-type plan, then mgd_care_ins_R2 is coded as 0.

```
# Define composite variable -- covered by gatekeeper/managed care insurance plan

# MCRPHOY1 = COV BY MEDICARE MANAGED CARE - 12/31/15
```

```
# MCDHMOY1 = COV BY MCAID/SCHIP HMO-R3 TIL 12/31/15

# MCDMCY1 = CV MCD/CHIP GTKPR PLN-R3 TIL 12/31/15

# PRVHMOY1 = COVERED BY PRIVATE HMO-R3 TIL 12/31/15

analysis1_predictors['mgd_care_ins_R2'] = ((analysis1_predictors['_MCRPHOY1']==1) |
(analysis1_predictors['_MCDHMOY1']==1) |

                                           (analysis1_predictors['_MCDMCY1']==1) |
(analysis1_predictors['_PRVHMOY1']==1))*1

analysis1_predictors.loc[analysis1_predictors['mgd_care_ins_R2']<1,'mgd_care_ins_R2']=
0
```

The following sections of code impute the median of the non-missing values for missings (for continuous varibles) or the mode for categorical variables. Two separate functions are created to do this, impute_median_continuous and impute_mode_categorical.

In the impute_median_continuous function, most of the work is done by fillna().median() method. The rest of the code within the function is used for validation – the distribution of the variable before vs. after median imputation is displayed.

```
# Impute median for missing if continuous

def impute_median_continuous(var):

    print(var+', Before:',analysis1_predictors[var].mean(),',
missing:',analysis1_predictors[var].isnull().sum(),

          ', non-missing:',analysis1_predictors[var].notnull().sum())

    analysis1_predictors[var] =
analysis1_predictors[var].fillna(analysis1_predictors[var].median())

    print(var+', After:',analysis1_predictors[var].mean(),',
missing:',analysis1_predictors[var].isnull().sum(),

          ', non-missing:',analysis1_predictors[var].notnull().sum(),'\n')


impute_median_continuous('_TOTEXPY1')

impute_median_continuous('_AGEY1X')

impute_median_continuous('_FAMINCY1')
```

In the impute_mode_categorial function, the workhorse is the fillna(mode) method. Again, there is some code to check the frequency distribution before vs. after imputation of the mode.

```
# Impute mode for missing if categorical

def impute_mode_categorical(var,mode):

    print(var+', Before:','\n',pd.value_counts(analysis1_predictors[var]))

    analysis1_predictors[var] = analysis1_predictors[var].fillna(mode)

    print(var+', After:','\n',pd.value_counts(analysis1_predictors[var]))
```

```
impute_mode_categorical('_RTHLTH2',1)

impute_mode_categorical('_MNHLTH2',1)
```

Finally, data frame new_preds is created, including the person ID along with the variables with imputation, as well as the new variable mgd_care_ins_R2.

```
# new_preds = Resulting datasets of predictors, negative values converted to NA and
replaced with median or mode

new_preds =
analysis1_predictors[['DUPERSID','_RTHLTH2','_MNHLTH2','_TOTEXPY1','_AGEY1X','_FAMINCY
1','mgd_care_ins_R2']]

old_preds = df2[['DUPERSID','RTHLTH2','MNHLTH2','TOTEXPY1','AGEY1X','FAMINCY1',

                            'MCRPHOY1','MCDHMOY1','MCDMCY1','PRVHMOY1']]
```

Now we illustrate the output of the code described above. Output from removing values <0:

```
Original distribution: RTHLTH2  1.0    4692
 2.0    4224
 3.0    3677
 4.0    1369
 5.0     397
-1.0      57
-8.0       4
-7.0       2
Name: RTHLTH2, dtype: int64
 New distribution: _RTHLTH2 1.0    4692
2.0    4224
3.0    3677
4.0    1369
5.0     397
Name: _RTHLTH2, dtype: int64


Output after imputing the mode for missing:

_RTHLTH2, Before:
 1.0    4692
2.0    4224
3.0    3677
4.0    1369
5.0     397
Name: _RTHLTH2, dtype: int64
_RTHLTH2, After:
 1.0    4755
2.0    4224
3.0    3677
4.0    1369
5.0     397
```

Finally, the modified variables are merged back with the rest of the data, to create the new data frame df3 that is used for subsequent analyses.

```
# Join adjustment variables with original data frame

df3 = pd.merge(df2, new_preds, left_index=True, right_index=True, how='left')
```

```
print('df2 shape: ',df2.shape)

print('df3 shape: ',df3.shape)

df2 shape:  (14422, 3593)
df3 shape:  (14422, 3600)
```

The shape output shows, as would be expected, that df3 has the same number of rows as df2 (14,422 each) but df3 has a few more columns, i.e., the new variables with imputed values.

SAS

In SAS, arrays are defined including the original variables (original_vars) and the variables that will include imputed values for "refused," "don't know" etc (recoded_vars). A do-loop loops through the arrays, first copying original_vars{i} into recoded_vars{i}, then converting negative values of recoded_vars{i} to missing values, denoted with a period (".") in SAS for numeric variables. Immediately below this, indicator mgd_care_ins_R2 is defined, which has nothing to do with the imputation process used for the other adjustment variables, this is simply a convenient place to define it.

```
data df2b;
set df2;
array original_vars{*} RTHLTH2 MNHLTH2 TOTEXPY1 AGEY1X FAMINCY1
MCRPHOY1 MCDHMOY1 MCDMCY1 PRVHMOY1;
array recoded_vars{*} _RTHLTH2 _MNHLTH2 _TOTEXPY1 _AGEY1X _FAMINCY1
_MCRPHOY1 _MCDHMOY1 _MCDMCY1 _PRVHMOY1;

do i=1 to dim(original_vars);
   recoded_vars{i}=original_vars{i};
   if recoded_vars{i}<0 then recoded_vars{i}=.;
end;

mgd_care_ins_R2=(_MCRPHOY1=1 or _MCDHMOY1=1 or _MCDMCY1=1 or _PRVHMOY1=1);
run;
```

Next PROC MEANS and FREQ are used to find the median and mode of the continuous and categorical adjustment variables, respectively. Then in df3, these values are imputed for missings in the data set df2c. This approach is manual and inelegant, and it would not be practical if there were many adjustment variables. In the latter case, writing a macro which would read the median or mode into macro variables would be a more automated strategy. However, this would take a lot more code typing.

```
proc means data=df2b median;
var _TOTEXPY1 _AGEY1X _FAMINCY1;
run;
proc freq data=df2b;
tables _RTHLTH2 _MNHLTH2;
run;

data df2c;
set df2b;
if _TOTEXPY1=. then _TOTEXPY1=894;
if _AGEY1X=. then _AGEY1X=35;
if _FAMINCY1=. then _FAMINCY1=47840;
if _RTHLTH2=. then _RTHLTH2=1;
if _MNHLTH2=. then _MNHLTH2=1;
run;
```

Next we confirm that SAS produces the same results from imputation as obtained in Python:

```
ods rtf file='C:\projects\SAS_python_compare\illustrate_handling_miss_imp.rtf';
title 'Original variable and with values removed';
proc freq data=df2b;
tables RTHLTH2 _RTHLTH2;
run;
title 'Recoded variable with mode imputation';
proc freq data=df2c;
tables _RTHLTH2;
run;
title ' ';
ods rtf close;
```

Results are as follows, using RTHLTH2 as an example:

PERCEIVED HEALTH STATUS - RD 2				
RTHLTH2	Frequency	Percent	Cumulative Frequency	Cumulative Percent
-8	4	0.03	4	0.03
-7	2	0.01	6	0.04
-1	57	0.40	63	0.44
1	4692	32.53	4755	32.97
2	4224	29.29	8979	62.26
3	3677	25.50	12656	87.75
4	1369	9.49	14025	97.25
5	397	2.75	14422	100.00

_RTHLTH2	Frequency	Percent	Cumulative Frequency	Cumulative Percent
1	4692	32.68	4692	32.68
2	4224	29.42	8916	62.09
3	3677	25.61	12593	87.70
4	1369	9.53	13962	97.24
5	397	2.76	14359	100.00
Frequency Missing = 63				

_RTHLTH2	Frequency	Percent	Cumulative Frequency	Cumulative Percent
1	4755	32.97	4755	32.97
2	4224	29.29	8979	62.26
3	3677	25.50	12656	87.75
4	1369	9.49	14025	97.25
5	397	2.75	14422	100.00

5. Descriptive analysis of association between having a usual healthcare provider and the analytic covariates

Python

In Analysis 1, descriptive statistics are broken out by whether or not a respondent had a usual healthcare provider in round 4 (R4) of 2016 (has_usc_R4), which is the analytic outcome variable. Specifically, the analysis describes the distribution of potential predictors in 2015 based on whether or not the respondent had a usual healthcare provider in 2016.

To accomplish this in Python, first has_usc_R4 is tagged as the grouping variable in df3 using groupby. Then the agg method is used to define descriptive statistics for the R2 predictors (either mean, median, or both, the relevant statistics are chosen for each variable) and the results are output to desctab.

```
# Initial descriptive analysis

# Group by has_usc_R4 (the analysis dependent variable)

gb = df3.groupby(['has_usc_R4'])

# Use different summary statistics for different measures

desctab = gb.agg({'_RTHLTH2' : 'mean','_MNHLTH2' : 'mean',

        '_TOTEXPY1' : ['mean','median'],'_AGEY1X' : 'mean',

        '_FAMINCY1' : ['mean','median'],'mgd_care_ins_R2' : 'mean'})

Next a formatting dict is created to format the descriptive statistics.

desctab.index.name = 'Has usual care provider, R4'

format_dict = {('_RTHLTH2', 'mean'):'{:,.2f}',

               ('_MNHLTH2', 'mean'):'{:,.2f}',

               ('_TOTEXPY1', 'mean'):'${0:,.0f}',

               ('_TOTEXPY1', 'median'):'${0:,.0f}',

               ('_AGEY1X', 'mean'):'{:,.1f}',

               ('_FAMINCY1', 'mean'):'${0:,.0f}',

               ('_FAMINCY1', 'median'):'${0:,.0f}',
```

```
('mgd_care_ins_R2', 'mean'):'{:.0%}'}
```

Finally the table is printed:

```
desctab.style.format(format_dict)
```

The resulting table is displayed below.

	_RTHLTH2	_MNHLTH2	_TOTEXPY1		_AGEY1X	_FAMINCY1		mgd_care_ins_R2
	mean	mean	mean	median	mean	mean	median	mean
Has usual care provider, R4								
0	2.15	1.95	$3,844	$676	33.5	$62,825	$45,647	45%
1	2.29	2.03	$6,041	$1,582	42.4	$69,376	$52,400	42%

SAS

The descriptive statistics are generated using SAS PROC MEANS. Instead of the groupby method, breakouts by has_usc_R4 are achieved by using a CLASS statement in PROC MEANS. No attempt is made to format the results, but this can also be done in SAS, for example by putting the results to an output dataset using ODS and then formatting the resulting table with SAS format statements.

```
ods rtf file='C:\projects\SAS_python_compare\adjustment_descrpt.rtf';
proc means data=df2c mean median;
class has_usc_R4;
var _RTHLTH2 _MNHLTH2 _TOTEXPY1 _AGEY1X _FAMINCY1 mgd_care_ins_R2;
run;
ods rtf close;
```

The results are exactly the same as in Python, except for rounding and formatting.

has_usc_R4	N Obs	Variable	Mean	Median
0	9815	_RTHLTH2	2.1548650	2.0000000
		_MNHLTH2	1.9496689	2.0000000
		_TOTEXPY1	3844.28	676.0000000
		_AGEY1X	33.4615385	31.0000000
		_FAMINCY1	62824.76	45647.00
		mgd_care_ins_R2	0.4543046	0
1	4607	_RTHLTH2	2.2889082	2.0000000
		_MNHLTH2	2.0290862	2.0000000
		_TOTEXPY1	6040.92	1582.00
		_AGEY1X	42.3535924	46.0000000
		_FAMINCY1	69375.69	52400.00
		mgd_care_ins_R2	0.4180595	0

6. Logistic regression analysis

Python

The next step in Analysis 1 is to model the probability of having a usual healthcare provider in 2016 as a function of personal characteristics measured in 2015 (i.e., age, income, self-reported health status, healthcare expenditures and whether or not the respondent had a managed care insurance plan). Logistic regression is one of the most common techniques used to do this kind of modeling. The outcome is binary (has_usc_R4 coded as "yes" = 1 or "no" = 0) and the 2015 predictor variables are treated as continuous. One could argue that the self-reported health status variables should be treated as categorical, but that is not illustrated here.

The statsmodels library is used to execute the logistic regression.

```
import statsmodels.api as sm
```

Because the predictors are scaled differently, some of the model coefficients would appear very small if not rescaled, so that is done here for the predictors on a dollars scale. The rescaling is simply division by $10,000, so that the rescaled value is on a per-$10k scale.

```
# re-scale the $ variables because otherwise the coefficients are very small
df3['_TOTEXPY1_10k'] = df3['_TOTEXPY1']/10000
df3['_FAMINCY1_10k'] = df3['_FAMINCY1']/10000
```

If the model is to include an intercept term, that needs to be coded and represented in the model.

```
df3['intercept'] = 1
```

The following code estimates the logistic regression model. Basic results are printed.

```
logit_model =
sm.Logit(df3['has_usc_R4'],df3[['intercept','_RTHLTH2','_MNHLTH2','_TOTEXPY1_10k','_AG
EY1X',

                                                '_FAMINCY1_10k','mgd_care_ins_R2']])

result = logit_model.fit()

print(result.summary())
```

The results are shown below.

```
Optimization terminated successfully.
         Current function value: 0.609358
         Iterations 5
                           Logit Regression Results
==============================================================================
Dep. Variable:             has_usc_R4   No. Observations:                14422
Model:                          Logit   Df Residuals:                    14415
Method:                           MLE   Df Model:                            6
Date:                Sun, 19 Jan 2020   Pseudo R-squ.:                 0.02728
Time:                        12:26:02   Log-Likelihood:                -8788.2
converged:                       True   LL-Null:                       -9034.6
Covariance Type:            nonrobust   LLR p-value:                2.768e-103
==============================================================================
                      coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
intercept          -1.3942      0.058    -24.018      0.000      -1.508      -1.280
_RTHLTH2           -0.0094      0.023     -0.406      0.685      -0.055       0.036
_MNHLTH2           -0.0023      0.023     -0.097      0.923      -0.048       0.044
_TOTEXPY1_10k       0.0631      0.015      4.322      0.000       0.035       0.092
_AGEY1X             0.0150      0.001     17.962      0.000       0.013       0.017
_FAMINCY1_10k       0.0148      0.003      4.959      0.000       0.009       0.021
mgd_care_ins_R2    -0.0798      0.037     -2.149      0.032      -0.153      -0.007
==============================================================================
```

Greater healthcare expenditures, greater family income and older age in 2015 were associated with a greater likelihood of having a usual healthcare provider in 2016. Interestingly, having a managed care or gatekeeper-type of insurance plan in 2015 was also associated with increased likelihood of having a usual healthcare provider in 2016, but in the opposite direction as would be expected -- one would expect such an insurance plan to encourage a relationship with a usual healthcare provider.

SAS

In SAS, the modeling is accomplished using PROC LOGISTIC. An intercept term is included by default and does not need to be explicitly coded. Whereas Python by default models the probability of the 1 level, SAS by default models the probability of the 0 level; the “descending” option in the PROC LOGISTIC statement is one way to model the probability of 1.

```
data df2d;
set df2c;
_TOTEXPY1_10k=_TOTEXPY1/10000;
_FAMINCY1_10k=_FAMINCY1/10000;
run;

ods rtf file='C:\projects\SAS_python_compare\logistic_compare1.rtf';
proc logistic data=df2d descending;
```

```
model has_usc_R4 = _RTHLTH2 _MNHLTH2 _TOTEXPY1_10k _AGEY1X _FAMINCY1_10k
mgd_care_ins_R2;
run;
ods rtf close;
```

Partial output of SAS PROC LOGISTIC is shown below. The coefficients and standard errors are exactly the same as in Python. For statistical tests (null hypothesis that the parameter is 0), SAS by default uses Wald tests whereas Python by default uses z-tests. The conclusions are nearly the same.

Analysis of Maximum Likelihood Estimates					
Parameter	DF	Estimate	Standard Error	Wald Chi-Square	Pr > ChiSq
Intercept	1	-1.3942	0.0580	576.8483	<.0001
_RTHLTH2	1	-0.00945	0.0233	0.1644	0.6851
_MNHLTH2	1	-0.00226	0.0234	0.0093	0.9230
_TOTEXPY1_10k	1	0.0631	0.0146	18.6833	<.0001
_AGEY1X	1	0.0150	0.000838	322.6157	<.0001
_FAMINCY1_10k	1	0.0148	0.00298	24.5925	<.0001
mgd_care_ins_R2	1	-0.0798	0.0371	4.6198	0.0316

Analysis 2

The focus of Analysis 2 was to explain the association between having a usual healthcare provider in 2015 and total healthcare expenditures in 2016, after adjusting for other 2015 measures that might influence 2016 healthcare spend.

7. Define variables and descriptive analysis

Python

Matplotlib is first imported to execute some charts of 2016 healthcare expenditures in relation to personal characteristics measured in 2015. Data frame df4 is created, keeping only the variables that are needed for Analysis 2.

```
import matplotlib.pyplot as plt

from matplotlib.ticker import FuncFormatter

df4 =
df3[['TOTEXPY2','_TOTEXPY1_10k','_FAMINCY1_10k','_RTHLTH2','_MNHLTH2','_TOTEXPY1','_AG
EY1X',
           '_FAMINCY1','mgd_care_ins_R2','has_usc_R4','has_usc_R2']]
```

As an initial step, descriptive statistics are examined. For continuous variables measured in 2015, categories ("bins") were created, using either quartiles (for 2015 healthcare expenditures and family income) or pre-specified meaningful categories (age in 2015). The quartiles were provided by the describe() method. The pd.cut method was used to create categorized versions of these variables, with labels that were also specified.

```
df4.describe()

# For income and expenditures, group by quartile

_TOTEXPY1_bins = [0,200,894,3455,1000000]

_TOTEXPY1_bins_label = ['0 to 200','201 to 894','895 to 3455','3456 and up']

df4['_TOTEXPY1_binned'] = pd.cut(df4['_TOTEXPY1'], _TOTEXPY1_bins,
labels=_TOTEXPY1_bins_label)


_FAMINCY1_bins = [0,22000,47840,88996,1000000]

_FAMINCY1_bins_label = ['0 to 22000','22001 to 47840','47841 to 88996','88997 and up']

df4['_FAMINCY1_binned'] = pd.cut(df4['_FAMINCY1'], _FAMINCY1_bins,
labels=_FAMINCY1_bins_label)


# For age, group by intuitively meaningful categories (children, younger adult, older
adult, Medicare age)

_AGEY1X_bins = [0,18,44,64,110]

_AGEY1X_bins_label = ['0 to 18','19 to 44','45 to 64','65 and up']

df4['_AGEY1X_binned'] = pd.cut(df4['_AGEY1X'], _AGEY1X_bins,
labels=_AGEY1X_bins_label)
```

Next, the custom function exp2_byvar is created to provide descriptive statistics for 2016 medical expenditures ("exp2" in the function name) by personal characteristics measured in 2015 ("byvar" in the name). The 2015 personal characteristics used in the descriptive analysis are all categorical – specifically, the binned versions of 2015 medical expenditures, income and age, along with self-rated health (ordinal with a small number of categories), having a usual healthcare provider (binary) and having a managed care type of insurance plan (binary). The first step is to print a frequency table of each 2015 variable, using pd.value counts. Then groupby is used to create a temporary data frame, temp, grouped by the 2015 variable, showing the mean of 2016 medical expenditures (TOTEXP2) within each category. Then a figure is created of temp (plt.figure), with the y-axis formatted so that it clearly displays 2016 healthcare expenditures on the dollars scale.

```
def exp2_byvar(byvar):

    print('byvar is:',byvar,'\n',pd.value_counts(df4[byvar]))

    temp = df4['TOTEXPY2'].groupby(df4[byvar]).mean()

    print(temp)
```

```
    plt.figure();

    ax = temp.plot.bar()

    ax.yaxis.set_major_formatter(FuncFormatter(lambda y, _: '${0:,.0f}'.format(y)))

    print('\n')
```

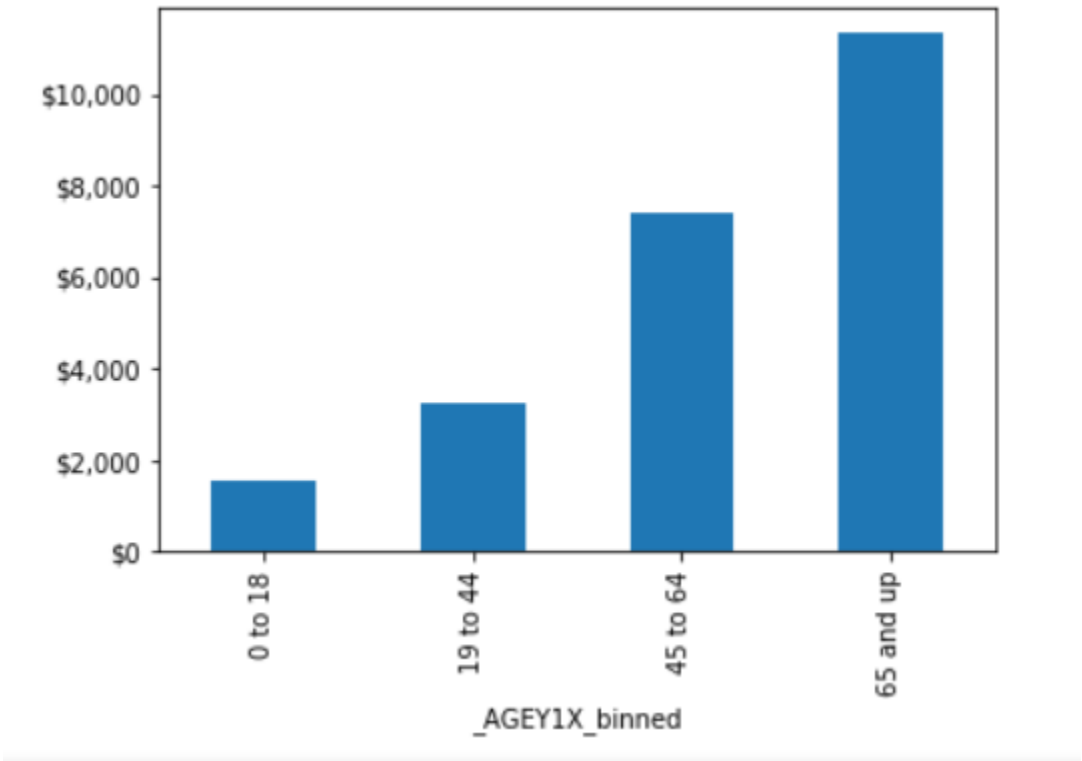
A for loop is used to call function exp2_byvars for each of the 2015 variables, as shown below.

```
byvars =
['_TOTEXPY1_binned','_FAMINCY1_binned','_AGEY1X_binned','_RTHLTH2','_MNHLTH2','mgd_car
e_ins_R2','has_usc_R2']

for var in byvars:

    exp2_byvar(var)
```

Below is an example of one of the graphs produced by this function, showing average 2016 medical expenditures by category of age in 2015.



SAS

First PROC UNIVARIATE is used to obtain the 25th, 50th and 75th percentiles of 2015 medical expenditures and family income.

```
proc univariate data=df2d;

var _TOTEXPY1 _FAMINCY1;

run;
```

The next step is to define the categories for each of the continuous variables. One way to do this is illustrated below.

```
data df2d;
set df2d;
if _TOTEXPY1 ne . then do;
   if _TOTEXPY1<=200 then _TOTEXPY1_bins=1;
   else if _TOTEXPY1<=894 then _TOTEXPY1_bins=2;
   else if _TOTEXPY1<=3455 then _TOTEXPY1_bins=3;
   else _TOTEXPY1_bins=4;
end;

if _FAMINCY1 ne . then do;
   if _FAMINCY1<=22000 then _FAMINCY1_bins=1;
   else if _FAMINCY1<=47840 then _FAMINCY1_bins=2;
   else if _FAMINCY1<=88996 then _FAMINCY1_bins=3;
   else _FAMINCY1_bins=4;
end;

if _AGEY1X ne . then do;
   if _AGEY1X<=18 then _AGEY1X_bins=1;
   else if _AGEY1X<=44 then _AGEY1X_bins=2;
   else if _AGEY1X<=64 then _AGEY1X_bins=3;
   else _AGEY1X_bins=4;
end;
run;
```

In SAS, PROC FORMAT is a convenient method to create custom formats for variables, as shown below.

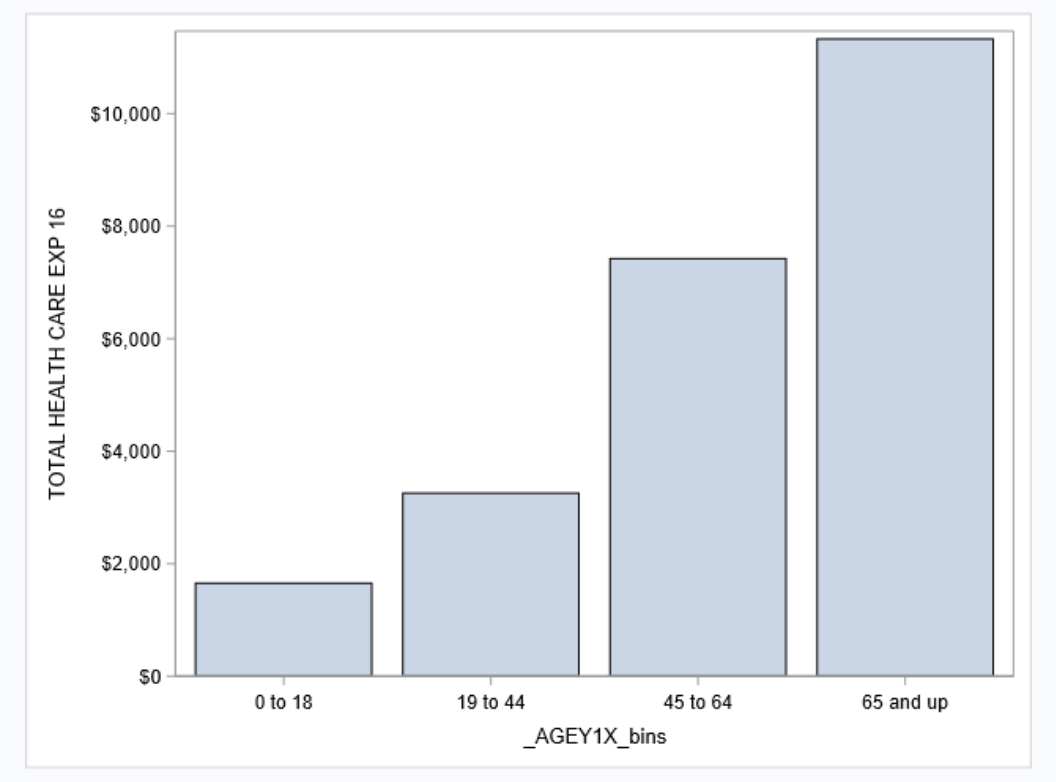
```
proc format;
value TOTEXPY1_binf 1='0 to 200' 2='201 to 894' 3='895 to 3455' 4='3456 and up';
value FAMINCY1_binf 1='0 to 22000' 2='22001 to 47840' 3='47841 to 88996' 4='88997 and
up';
value AGEY1X_binf 1='0 to 18' 2='19 to 44' 3='45 to 64' 4='65 and up';
run;
```

PROC SGPLOT is used to create a column chart showing mean 2016 medical expenditures for each 2015 age category.

```
proc sgplot data=df2d;
format TOTEXPY2 dollar10.;
```

```
format _AGEY1X_bins AGEY1X_binf.;
  vbar _AGEY1X_bins / response=TOTEXPY2 stat=mean;
run;
```

The figure below is the output of PROC SGPLOT. It is basically similar to the chart produced by Python.



8. Modeling the association of having a usual healthcare provider in year 1 with total medical expenditures in year 2

The final step in Analysis 2 is to use regression to model the association between 2016 healthcare expenditures and having a usual healthcare provider in 2015, after adjusting for other personal characteristics measured in 2015.

Python

The distribution of healthcare expenditures is usually skewed with a long right tail, because few individuals have very high expenditures and many values are concentrated at the low end of the scale. With this distribution, basic assumptions of ordinary least squares regression, such as normally distributed residuals, are unlikely to be met. Therefore, 2016 medical expenditures were log transformed prior to modeling. Models were created on both the original dollars scale and the log-transformed dollars scale.

```
# Log transform Y2 expenditures
df4['ln_TOTEXPY2'] = np.log(df4['TOTEXPY2']+1)
```

Next ordinary least squares regression models are created, after coding the intercept term as was done for the logistic regression models in Analysis 1. Separate models are created on the original and log-transformed dollars scales. Model fit and coefficients with statistical tests are printed.

```
df4['intercept'] = 1


# Original scale Y2 spend

ols_model =
sm.OLS(df4['TOTEXPY2'],df4[['intercept','_RTHLTH2','_MNHLTH2','_TOTEXPY1_10k','_AGEY1X
',


'_FAMINCY1_10k','mgd_care_ins_R2','has_usc_R2']])

# Log transformed Y2 spend

ols_ln_model =
sm.OLS(df4['ln_TOTEXPY2'],df4[['intercept','_RTHLTH2','_MNHLTH2','_TOTEXPY1_10k','_AGE
Y1X',


'_FAMINCY1_10k','mgd_care_ins_R2','has_usc_R2']])


result = ols_model.fit()

print('*** Model of Y2 spend on original scale ***','\n',result.summary(),'\n')

result_ln = ols_ln_model.fit()

print('*** Model of log-transformed Y2 spend ***','\n',result_ln.summary(),'\n')
```

The table below shows the output of the model of log-transformed 2016 medical expenditures.

```
*** Model of log-transformed Y2 spend ***
                            OLS Regression Results
==============================================================================
Dep. Variable:            ln_TOTEXPY2   R-squared:                       0.183
Model:                            OLS   Adj. R-squared:                  0.182
Method:                 Least Squares   F-statistic:                     459.7
Date:                Sun, 19 Jan 2020   Prob (F-statistic):               0.00
Time:                        14:14:32   Log-Likelihood:                -35245.
No. Observations:               14422   AIC:                         7.051e+04
Df Residuals:                   14414   BIC:                         7.057e+04
Df Model:                           7
Covariance Type:            nonrobust
==============================================================================
                     coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
intercept          3.6548      0.073     49.981      0.000       3.511       3.798
_RTHLTH2           0.2702      0.030      8.959      0.000       0.211       0.329
_MNHLTH2           0.0697      0.030      2.305      0.021       0.010       0.129
_TOTEXPY1_10k      0.4598      0.018     25.049      0.000       0.424       0.496
_AGEY1X            0.0302      0.001     27.909      0.000       0.028       0.032
_FAMINCY1_10k      0.0335      0.004      8.541      0.000       0.026       0.041
mgd_care_ins_R2   -0.0615      0.047     -1.298      0.194      -0.154       0.031
has_usc_R2         0.6611      0.050     13.106      0.000       0.562       0.760
```

```
==============================================================================
Omnibus:                     2041.786   Durbin-Watson:                   1.672
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             3018.405
Skew:                          -1.066   Prob(JB):                         0.00
Kurtosis:                       3.690   Cond. No.                         146.
==============================================================================
```

Most of the coefficients are in the direction that one would intuitively expect: Older individuals and ones who were less healthy in 2015, as indicated by lower self-rated health and higher 2015 medical expenditures, had greater medical expenditures in 2016. (Note that the self-rated health measures were coded as 1 = "excellent" to 5 = "poor".) Although not statistically significant at the conventional 0.05 level, it is also intuitive that the sign of having a managed care insurance plan in 2015 is negative. However, surprisingly, having a usual healthcare provider in 2015 was associated with greater total medical expenditures in 2016 – the opposite of the direction of association that one would expect.

SAS

In SAS, first dataset df2e is created, with log-transformed 2016 medical expenditures. Then regression models are created using SAS PROC REG.

```
data df2e;
set df2d;
ln_TOTEXPY2 = log(TOTEXPY2+1);
run;



ods rtf file='C:\projects\SAS_python_compare\spend_regression.rtf';
proc reg data=df2e;
model TOTEXPY2 =  _RTHLTH2 _MNHLTH2 _TOTEXPY1_10k _AGEY1X _FAMINCY1_10k
mgd_care_ins_R2 has_usc_R2 / vif;
run;
quit;

proc reg data=df2e;
model ln_TOTEXPY2 =  _RTHLTH2 _MNHLTH2 _TOTEXPY1_10k _AGEY1X _FAMINCY1_10k
mgd_care_ins_R2 has_usc_R2 / vif;
run;
quit;
ods rtf close;
```

Below shows basic output of the SAS model of log-transformed 2016 expenditures. The coefficients, standard errors, t-tests and R-squared values are exactly the same as in Python.

Root MSE	2.78754	R-Square	0.1825
Dependent Mean	6.09194	Adj R-Sq	0.1821
Coeff Var	45.75794		

Parameter Estimates						
Variable	DF	Parameter Estimate	Standard Error	t Value	Pr > |t|	Variance Inflation
Intercept	1	3.65483	0.07312	49.98	<.0001	0
_RTHLTH2	1	0.27019	0.03016	8.96	<.0001	1.98025
_MNHLTH2	1	0.06967	0.03023	2.30	0.0212	1.76616
_TOTEXPY1_10k	1	0.45985	0.01836	25.05	<.0001	1.10259
_AGEY1X	1	0.03018	0.00108	27.91	<.0001	1.22746
_FAMINCY1_10k	1	0.03349	0.00392	8.54	<.0001	1.07288
mgd_care_ins_R2	1	-0.06148	0.04737	-1.30	0.1944	1.02742
has_usc_R2	1	0.66115	0.05045	13.11	<.0001	1.02566

Discussion

The objective of this paper was to compare Python and SAS in an advanced analytics case study.

The case study showed that Python and SAS are similar in several respects:

1. Both are relatively easy to use – fairly advanced analytics can be conducted after writing a few dozen to a couple hundred lines of code.
2. Both are roughly close to the English language, thus fairly interpretable when reading the code.
3. Both are good at dealing with structured tabular data – importing, subsetting or "slicing," defining and transforming variables, and joining tables can all be done fairly easily.
4. Both exhibited good capabilities for regression modeling and yielded the same statistical estimates.
5. Both are good at looping through lists of variables and executing operations on each variable within a list.
6. Both have simple and attractive plotting capabilities (in the past, this was to not so much the case for SAS, but SAS's capabilities and ease of use for plotting have increased since the introduction of PROC SGPLOT).
7. The Jupyter Notebook interface, which was used for the Python analyses described in this paper, has a similar look-and-feel to the programming window interface of SAS Enterprise Guide.
8. Both Python and SAS gave informative error messages, while both gave warning messages that can be ambiguous (this may just be the nature of warning messages).

Despite the many similarities between Python and SAS, some differences are also apparent:

1. In some cases the Python syntax seemed a little more "wordy" with more typing than SAS, while in other cases the SAS syntax seemed more wordy – this seems to be situation-specific.
2. Python was excellent at flexibly imputing missing data (e.g., median, mode); to do the same operations in SAS would likely be more complicated.

3. Python has excellent capabilities for producing annotated output, mixing output tables with descriptive language using print statements. SAS's capabilities for this are a bit more clunky, for example, using title statements for PROCS and writing to the log, neither of which seems as nice as Python's capabilities in this aspect.
4. Some of the code seemed to run a bit slower in Python than in SAS, although the difference was not drastic and may be situation-specific.
5. SAS's regression modeling procedures have excellent default output; the same can be produced from Python/statsmodels, but seems to require more coding.

General comments

There are advantages and disadvantages to both Python and SAS. Python, as open source code, is free of charge. Python also has a large and growing user community. New analytic capabilities may be available earlier in Python. SAS is commercial software. It has been around for a long time and is considered to be very accurate and reliable, including by the U.S. Federal Government. CMS still publishes models and data in SAS. Anecdotally, based on a recent conversation with an analyst who does a lot of FDA work, although FDA does not require submissions to be in SAS, FDA staff will often check results using SAS, given that SAS has been used for a long time for FDA submissions and is considered very credible. Although SAS is not cheap, most organizations can afford some PC SAS 9.4 licenses at a minimum.

The goal of this case study was to compare Python and SAS in a relatively realistic advanced analytics example within the healthcare context. Some analyses were illustrated, but many more analyses could be done. For example, to look at the association between having a usual healthcare provider in 2015 and medical expenditures in 2016, it could be argued that regression analysis is neither optimal nor sufficient and other techniques may be needed, for example propensity matching, additional statistical sensitivity analyses, and/or instrumental variable approaches. The results from the analysis were interesting and somewhat surprising (counter to the initial hypotheses), but more analyses would need to be done before drawing more definitive conclusions about the issues explored in the case study.

The ideal for analytics might be to use both Python and SAS, whichever works better and is more convenient in a given analytic situation. To make it easier to use the two together and mix and match, it would be great if SAS had a "PROC PYTHON" similar to "PROC SQL". It is unknown whether SAS has any thoughts of doing something like this. However, the SAS Institute seems to be very open to having users mix and match programming languages (SAS's native language, R, Python, and SQL) and SAS's new platform, SAS Viya, is designed to facilitate this.

One bottom line conclusion needs to be stated: Given that Python's capabilities seemed similar to SAS, and Python is free, it would be difficult for most users to justify the expense of SAS, unless it was paid for by their employer.