Enhancing Your SAS® Viya® Workflows with Python:

Integrating Python's Open-Source Libraries with SAS® using PROC PYTHON

Ryan Paul Lafler
Miguel Ángel Bravo

Objectives and Goals

Python Programming Fundamentals

Essential Python Knowledge for Data Science

Accessing Python within SAS® Viya®

Investigating the Python Runtime in PROC PYTHON

Building a PROC PYTHON Data Workflow with Pandas

An Example Integration of Python within SAS

Python Fundamentals

- Python is an open-source interpreted language; scripts are compiled at runtime
- Supports community of third-party developers publishing libraries and packages on verified repos like **PyPi** (Python Package Index) and **GitHub**
- Libraries can be installed using **pip** (package installation manager) or using pre-bundled libraries installed by Conda environments
- **Object-oriented language** that leverages **classes** to define unique **instances** called **objects**
  - Objects share **properties** (attributes) that can add, adjust, define, delete and modify each instance of the class
- **Methods** are defined within classes to edit, process, and return new objects

Python Foundational Data Object Types

Provides several methods for storing data objects:

- **List**: Indexed and mutable structure that can contain the same or different object types
- **Tuple**: Indexed and immutable data structure where objects cannot be changed once created
- **Set**: Unordered list of objects that cannot contain duplicate values
- **Dictionary**: Unordered and mutable JSON-like structure that relies on *key* → *value* mappings

- These objects form the foundation for more advanced data objects likePandas DataFrames and NumPy Arrays

The Pandas Library for Data Science

- Pandas allows programmers to easily work with tabular, flat-file (i.e., CSV, delimited files), web-scraped, and JSON-like data
- Utilizes objects known as Pandas DataFrames to store data with multiple columns or a Pandas Series to store data in a column (vector)
- Automatically creates and **integer-based index** for rows and columns inside the DataFrame
- Provides methods that support:
  - Filtering, querying, and subsetting
  - Merging, joining, and concatenating
  - Transforming, modifying, and engineering new features

Pandas DataFrames and SAS Datasets

- Pandas DataFrames and SAS Datasets share many similarities including:
  - Mixed feature types including numeric, string, categorical, & datetime
  - Encoding and representation of missing values
  - Merging, joining, and concatenating different datasets
  - Generating statistical summaries that account for missing values
- Major difference: Pandas stores the *entire* dataset within Python's memory while SAS stores the data on disk
  - Must use lazy evaluation and data chunking strategies to handle big data
  - By storing data on disk, SAS can naturally handle big data by not using memory

Pandas DataFrames and SAS Datasets

	PatientID	Age	Gender	Ethnicity	SocioeconomicStatus	EducationLevel	BMI	Smoking	AlcoholConsumption	PhysicalActivity	...	Itching	Qualit
0	1	71	0	0	0	2	31.069414	1	5.128112	1.676220	...	7.556302	
1	2	34	0	0	1	3	29.692119	1	18.609552	8.377574	...	6.836766	
2	3	80	1	1	0	1	37.394822	1	11.882429	9.607401	...	2.144722	
3	4	40	0	2	0	1	31.329680	0	16.020165	0.408871	...	7.077188	
4	5	43	0	1	1	2	23.726311	0	7.944146	0.780319	...	3.553118	

KIDNEY_SAS | Table rows: 1659 | Columns: 54 of 54 | Rows 1 to 200

Enter expression

	PatientID	Age	Gender	Ethnicity	SocioeconomicStatus	EducationLevel	E
1	1	71	0	0	0	2	31.0694
2	2	34	0	0	1	3	29.6921
3	3	80	1	1	0	1	37.3948
4	4	40	0	2	0	1	31.3296
5	5	43	0	1	1	2	23.7263

Python Method to Import Data

```
def import_data(self, file_name, export_name, delim=',', libname='/') :
    try :
        # Import using the Pandas library
        df = pd.read_csv(
            file_name, delimiter = delim, header = 'infer'
        )
        export_path = f'{libname}.{export_name}'
        # Transform into a SAS Dataset
        SAS.df2sd(df, export_path)
        return df_copy
    except :
        raise Exception(f'There was an issue importing {file_name}')
```

About PROC PYTHON

- Installed with SAS Viya since 2021 → built-in alternative to SASPy that makes managing data between SAS and Python runtimes simpler
- Allows programmers to execute Python scripts within SAS and persist data between sessions
- Comes pre-installed with several libraries and dependencies including:
  - **Pandas, NumPy, SciPy, Xarray;**
  - **TensorFlow, Keras, Scikit-Learn;**
  - **Matplotlib, Seaborn**
- Creates automatic connection between SAS and Python runtimes to transfer data between 2 runtimes

Checking the PROC PYTHON Runtime

```
/* Check and verify Python Installation in SAS Viya */
PROC PYTHON ;
     SUBMIT ;

import sys
print(f'Installed Python Version: {sys.version}')

     ENDSUBMIT ;
RUN ;
```

Checking the PROC PYTHON Runtime

```
85   PROC PYTHON ;
86   SUBMIT
NOTE: Python initialized.
Python 3.9.18 (main, Jun 25 2024, 16:00:09)
[GCC 8.5.0 20210514 (Red Hat 8.5.0-20)] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
>>>
86 !         ;
87
88   import sys
89
90   print(f'Installed Python Version: {sys.version}')
91
92   ENDSUBMIT ;
93   RUN ;
>>>
Installed Python Version: 3.9.18 (main, Jun 25 2024, 16:00:09)
[GCC 8.5.0 20210514 (Red Hat 8.5.0-20)]
>>>
NOTE: PROCEDURE PYTHON used (Total process time):
      real time           0.68 seconds
      cpu time            0.02 seconds
```

Checking the Installed Python Libraries

```
/* Check installation of Python libraries in SAS Viya */
PROC PYTHON ;
     SUBMIT ;

# Import subprocess to run pip by accessing Python terminal in SAS Viya
import sys
import subprocess
# Run the pip freeze command to list all installed packages
installed_packages = subprocess.check_output([sys.executable, '-m', 'pip',
      'freeze'])
# Decode and print the result
print(installed_packages.decode("utf-8"))

     ENDSUBMIT ;
RUN ;
```

Checking the Installed Python Libraries

```
96   /* Check installation of Python libraries in system */
97   PROC PYTHON;
98        SUBMIT
NOTE: Resuming Python state from previous PROC PYTHON invocation.
98 !           ;
99
100  # Import subprocess to run pip commands
101  import subprocess
102
103  # Run the pip freeze command to list all installed packages
104  installed_packages = subprocess.check_output([sys.executable, '-m', 'pip', 'freeze'])
105
106  # Decode and print the result
107  print(installed_packages.decode("utf-8"))
108
109       ENDSUBMIT;
110  RUN;
>>>
absl-py==2.1.0
aiohttp==3.9.5
aiosignal==1.3.1
annotated-types==0.7.0
asttokens==2.4.1
astunparse==1.6.3
async-timeout==4.0.3
attrs==23.2.0
blinker==1.8.2
```

Running the PROC PYTHON Import Script

- Import and process delimited-file data as a SAS dataset using Python

  1. Import Python script file into PROC PYTHON using FILENAME
  2. Import Pandas library into the active Python runtime
  3. Retrieves macro variable string pathway pointing to CSV dataset and stores it as a Python string
  4. Imports CSV dataset as a Pandas DataFrame entirely in Python runtime's memory
  5. Coerces the Pandas DataFrame object to a SAS dataset stored temporarily in the *WORK* library (deleted when SAS Viya closes)

Defining References to the Target Dataset

```
/* Python script file input */

FILENAME SCRIPT DISK '/export/viya/homes/{user-
  email}/casuser/data_processing.py' ;


/* Define macro variable for dataset path */

%LET REFDATA = /export/viya/homes/{user-email}
  /casuser/Chronic_Kidney_Disease_data.csv ;
```

Running the Python Script File to Import Data

```
/* Submit Python code into PROC PYTHON */
PROC PYTHON INFILE=SCRIPT ;
      SUBMIT ;

# Python library imports
import pandas as pd
# Specify filename pathway
filename = SAS.symget('REFDATA')
# Retrieve method from Python script file to load the data
import_data(
      file_name = filename , export_name = 'KIDNEY_SAS'
)

      ENDSUBMIT ;
RUN ;
```

Investigating the *WORK.KIDNEY_SAS* Dataset

KIDNEY_SAS | Table rows: 1659 | Columns: 54 of 54 | Rows 1 to 200

Enter expression

	PatientID	Age	Gender	Ethnicity	SocioeconomicStatus	EducationLevel	E
1	1	71	0	0	0	2	31.0694
2	2	34	0	0	1	3	29.6921
3	3	80	1	1	0	1	37.3948
4	4	40	0	2	0	1	31.3296
5	5	43	0	1	1	2	23.7263

Processing in Pandas & Persisting to SAS Dataset

- We will add a new function to **`data_processing.py`** script file
  - New method will be called: **`make_indicator()`**
  - Creates new column(s) by making indicator (“dummy”) columns when values exceed certain numeric thresholds (i.e., age being divided in “young” or “old”)
- This will allow us to process DataFrame in Python’s memory by engineering new columns
- Must modify **PROC PYTHON** call to include this data processing step for the “PhysicalActivity” numeric column
  - Measured from 0 – 10 → threshold is 5 to classify as “non-active” or “active”
- Export & persist Pandas DataFrame to SAS Dataset on disk using **`SAS.df2sd()`** function

Defining Indicator Column Method

```
def make_indicator(
  df: pd.DataFrame,
  columns: list,
  thresholds: list
) -> pd.DataFrame :

    # Ensure lengths of lists are the same
    if len(columns) != len(thresholds) :
        raise ValueError("Number of columns is not the same as number of
              thresholds")

    # Make copy of original DataFrame
    df_copy = df.copy()
```

Defining Indicator Column Method

```
def make_indicator(
  df: pd.DataFrame,
  columns: list,
  thresholds: list
) -> pd.DataFrame :

  # Create new indicator column(s) and add to copy of original DataFrame
  for column, threshold in zip(columns, thresholds) :
        if column in df_copy.columns :
              df_copy[f"{column}_indicator"] = (df_copy[column] >
                    threshold).astype(int)
        else :
              raise ValueError(f"{column} not found in DataFrame")

  return df_copy
```

Importing, Processing, & Exporting Data to SAS

- Adding this method to the script file allows us to access it using **PYTHON** procedure's **INFILE** option
  - Automatically detects new methods added to Python script file
- We'll follow these steps to import, process, & export Pandas DataFrame to SAS Dataset
  1. Import CSV file as Pandas DataFrame with **`import_data()`** method
  2. Export the imported CSV to persistent directory → using custom LIBNAME
  3. Process Pandas DataFrame to include indicator feature for "PhysicalActivity"
  4. Export processed DataFrame as a SAS Dataset to persistent LIBNAME

PROC PYTHON to Process & Persist DataFrame

```
/* Submit Python code into PROC PYTHON */
PROC PYTHON INFILE=SCRIPT ;
     SUBMIT ;

# Python library imports
import pandas as pd

# Specify filename pathway
filename = SAS.symget('REFDATA')

# Retrieve method from Python script file to load the data
df = import_data(
     file_name = filename ,
     export_name = 'KIDNEY_SAS' ,
     libname = 'USERLIB'
)
```

PROC PYTHON to Process & Persist DataFrame

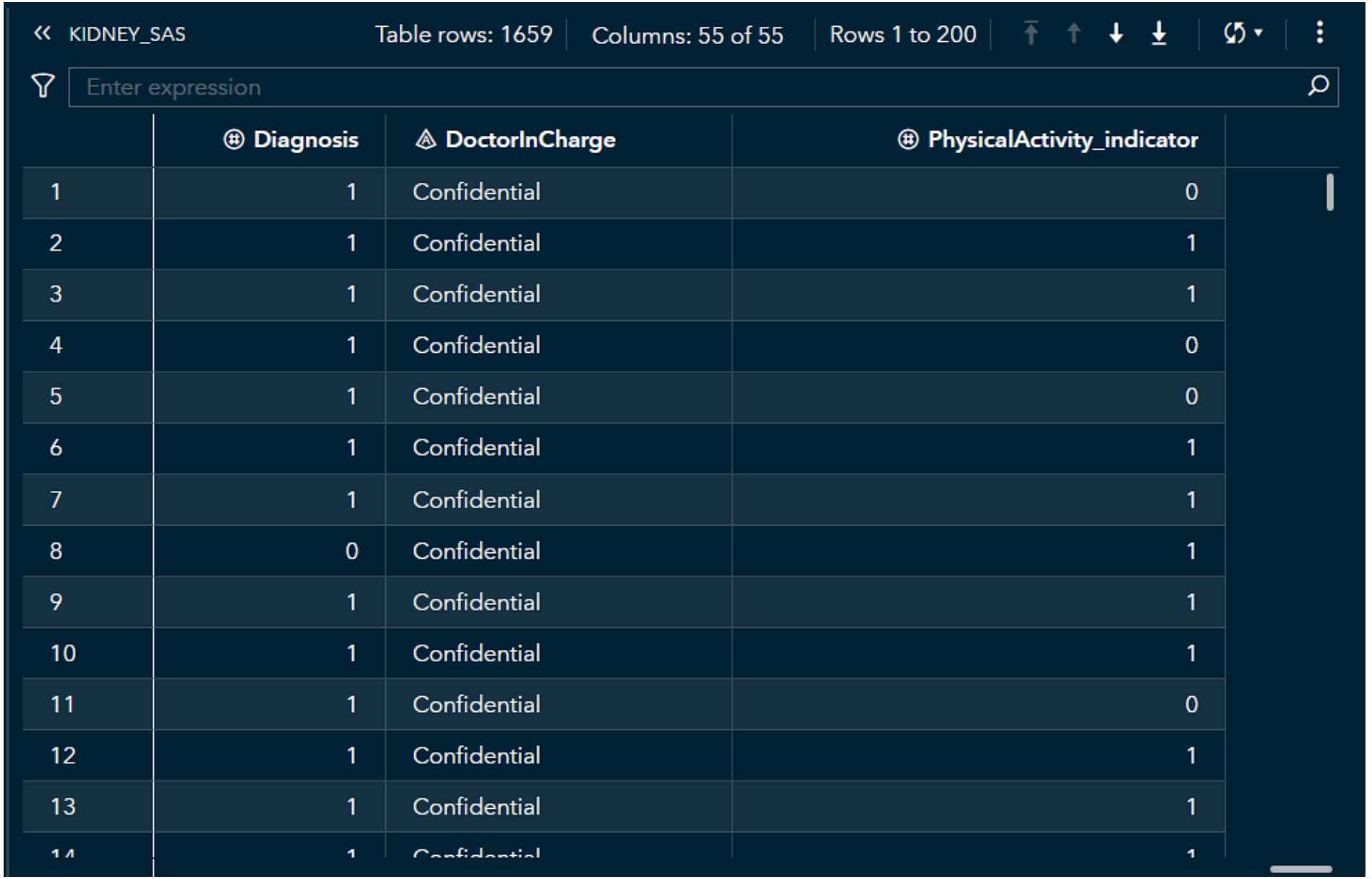
```
# Create indicator ("dummy") variables for specified columns
df = make_indicator(
     df = df,
     columns = ["PhysicalActivity"],
     thresholds = [5]
)

# Persist file to permanent path (LIBNAME)
export_path = f"USERLIB.KIDNEY_SAS"

# Export Pandas DataFrame as SAS Dataset to permanent path on Disk
SAS.df2sd(df, export_path)

     ENDSUBMIT ;
RUN ;
```

Processed SAS Dataset


KIDNEY_SAS Table rows: 1659 Columns: 55 of 55 Rows 1 to 200

Enter expression

	Diagnosis	DoctorInCharge	PhysicalActivity_indicator
1	1	Confidential	0
2	1	Confidential	1
3	1	Confidential	1
4	1	Confidential	0
5	1	Confidential	0
6	1	Confidential	1
7	1	Confidential	1
8	0	Confidential	1
9	1	Confidential	1
10	1	Confidential	1
11	1	Confidential	0
12	1	Confidential	1
13	1	Confidential	1
14	1	Confidential	1

Thank You for Attending!

Are there Questions, Discussions, and/or Comments?

Ryan Paul Lafler

rplafler@premier-analytics.com

LinkedIn.com/in/RyanPaulLafler

www.Premier-Analytics.com

Miguel Ángel Bravo

miguelangelbravo2000@gmail.com

www.LinkedIn.com/in/Miguel-Angel-Bravo/

www.mabravo.com