

APAC 2026

SAS to Python: A Hands-On Cheat Sheet for Statistical Programmers

Jagadish Katam,
Principes Technologies

Python adoption is rapidly increasing across analytics and data science

Help SAS programmers transition to Python

Mapping SAS DATA step and PROC logic to Python

Provide side-by-side SAS and Python comparisons

Core data manipulation and summarization patterns

Reshaping, merging, and visualization examples



Python Libraries Used



pandas – Data Manipulation

The cornerstone library for DATA step-style processing. Pandas provides DataFrames (similar to SAS datasets) and methods for filtering, transforming, and aggregating data.

```
import pandas as pd
```

NumPy – Numerical Operations

Delivers high-performance numerical computing and vectorized operations. Think of it as the computational engine behind pandas, handling array operations efficiently.

```
import numpy as np
```

matplotlib – Core Plotting

The foundational plotting library that provides granular control over visualizations. It's the Python equivalent of SAS/GRAPH, offering extensive customization options.

```
import matplotlib.pyplot as plt
```

seaborn – Statistical Visualizations

Built on matplotlib, seaborn simplifies creating sophisticated statistical graphics. It's comparable to PROC SGPLOT with prettier defaults and less code.

```
import seaborn as sns
```

- In Python, assigning a DataFrame to a new variable creates a reference to the original object, not an independent copy.
- To create an independent copy that matches SAS DATA step behavior, you must explicitly use the `copy()` method.

SAS Approach

```
data new_data;  
  set old_data;  
run;
```

The DATA step automatically creates an independent copy. Any modifications to `new_data` won't affect `old_data`.

Python Approach

```
old_data = pd.DataFrame(data)  
new_data = old_data.copy()
```

Key difference: Python requires explicit `.copy()` to create independent datasets.

The `copy()` method creates an independent DataFrame. Without it, `new_data` would just be another reference to the same data object.

- In pandas, variables are retained or removed by explicitly selecting or dropping columns from the DataFrame. This approach provides fine-grained control similar to SAS KEEP and DROP options.

SAS: KEEP and DROP

```
data new_data (keep=id);  
  set old_data (drop=job_title);  
run;
```

Python: Column Selection

```
# Drop a variable  
new_data = old_data.drop(  
  columns=["job_title"]  
)  
  
# Keep a variable  
new_data = old_data[["id"]]
```

1

IN List Filtering

SAS: `if year in (2010, 2011, 2012);`

Python:

```
old_data[old_data["year"].isin(
[2010, 2011, 2012]
)]
```

2

First Observation

SAS: `by id; if first.id;`

Python:

```
new_data = (
    old_data
    .sort_values("id")
    .groupby("id")
    .head(1)
)
```

3

Date Filtering

SAS: `if dob > "25APR1990"d;`

Python:

```
old_data["dob"] = pd.to_datetime(
    old_data["dob"]
)
new_data = old_data[
    old_data["dob"] > "1990-04-25"
]
```

01

Direct Assignment

```
old_data["total_income"] = (  
    old_data["wages"] +  
    old_data["benefits"]  
)
```

02

Using assign()

```
new_data = old_data.assign(  
    total_income=old_data["wages"]  
    + old_data["benefits"]  
)
```

03

Using eval()

```
old_data.eval(  
    "total_income = wages +  
    benefits",  
    inplace=True  
)
```

❏ SAS uses simple arithmetic: `total_income = wages + benefits;`

SAS Approach

```
data new_data;  
  set data_1 data_2;  
run;
```

Stack datasets vertically using SET statement

Python Approach

```
new_data = pd.concat(  
    [data_1, data_2],  
    axis=0,  
    ignore_index=True  
)
```

Use pd.concat() with axis=0 for row-wise stacking



Left Join with Merge Logic



SAS MERGE Statement

```
data new_data;  
  merge data_1(in=in_1) data_2;  
  by id;  
  if in_1;  
run;
```

Python merge() Function

```
new_data = pd.merge(  
    data_1,  
    data_2,  
    on="id",  
    how="left",  
    indicator=True  
)
```

The indicator=True option provides functionality similar to SAS IN= dataset options, tracking record origins

SAS FIND Function

```
data new_data;  
set old_data;  
if find( job_title , "Health" );  
run;
```

Case-sensitive substring search within DATA step

Python str.contains()

```
new_data = old_data[  
    old_data["job_title"]  
    .str.contains(  
        "Health",  
        case=False,  
        na=False  
    )  
]
```

Flexible pattern matching with case control and null handling

1

SAS SUBSTR

One-based indexing with explicit length parameter

```
substring = substr(  
    big_string, 3, 4  
);
```

2

Python Slicing

Zero-based indexing with exclusive end position

```
substring =  
    big_string.str[2:6]
```

□ **Key Difference:** SAS uses position 3 with length 4, while Python uses indices 2 through 6 (exclusive) to extract the same substring



Long to Wide Format Reshaping



PROC TRANSPOSE

```
proc transpose data=long_data  
out=wide_data;  
by student ;  
id subject ;  
var grade ;  
run;
```

pandas pivot()

```
wide_data = (  
    long_data  
    .pivot(  
        index="student",  
        columns="subject",  
        values="grade"  
    )  
    .reset_index()  
)
```

Transform student grades from multiple rows per student into one row with subject columns

01

Identify Variables

Specify ID variables (student) and value variables (English, Irish, Maths)

```
proc transpose
  data=wide_data
  out=long_data(rename=(col1= grade))
  name=subject;
  by student;
  var English Irish
      Maths;
run;
```

02

Apply Transformation

Use PROC TRANSPOSE in SAS or melt() in Python

03

Name Output Columns

Control variable names: subject for column names, grade for values

```
# Dynamically determine value variables
id_vars = ["student"]
value_vars = [
    col for col in wide_data.columns
    if col not in id_vars
]

long_data = wide_data.melt(
    id_vars= id_vars,
    value_vars=value_vars,
    var_name="subject",
    value_name="grade"
)
```

SAS Approach

```
data new_data;  
  set old_data;  
  text_var = put(num_var, best.);  
run;
```

The PUT function converts numeric to character using specified format width.

Python Approach

```
old_data["text_var"] = (  
    old_data["num_var"]  
    .astype(str)  
)  
new_data = old_data.copy()
```

The `astype(str)` method converts each numeric value to string representation.

SAS Approach

```
data new_data;  
  set old_data;  
  num_var = input(text_var, 8.);  
run;
```

INPUT function converts character values to numeric form.

Python Approach

```
# Using astype()  
old_data["num_var"] = (  
    old_data["text_var"]  
    .astype(int)  
)  
  
# Using to_numeric()  
old_data["num_var"] = pd.to_numeric(  
    old_data["text_var"]  
)
```

Two flexible methods for character-to-numeric conversion.

SAS Approach

```
proc freq data=old_data;  
  tables job_type*region;  
run;
```

* notation creates cross-tabulation of two variables.

Python Approach

```
count = (  
    old_data  
    .groupby(  
        ["job_type", "region"],  
        dropna=False  
    )  
    .size()  
    .reset_index(name="count")  
)  
print(count)
```

Groupby with size() produces equivalent counts for each combination. By default, pandas DROPS missing values in groupby. To keep them, you must use dropna=False

SAS Macro Approach

```
%macro add_variable(dataset_name);  
data &dataset_name;  
set &dataset_name;  
new_variable = 1;  
run;  
%mend;  
  
%add_variable( my_data );
```

Python Function Approach

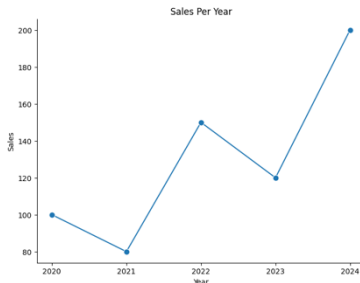
```
def add_variable(dataset_name):  
    dataset_name['new_variable'] = 1  
    return dataset_name  
  
my_data = add_variable(my_data)
```

The function is defined using the `def` keyword and accepts a pandas DataFrame as an input parameter. Inside the function, a new variable is created and assigned a constant value across all rows.

SAS PROC SGPLOT

```
proc sgplot data=my_data;  
  series x=year y=sales;  
  scatter x=year y=sales;  
  xaxis integer;  
  yaxis min=0;  
  title "Sales Over Time";  
run;
```

Both approaches create professional line plots with markers, integer x-axes, and clear titles.



Python with Seaborn

Create the plot

```
plt.figure(figsize=(8, 6))  
sns.lineplot(data=my_data, x='year', y='sales', marker='o', markersize=8) # Line with points
```

Set the x-axis to display only integer values (no fractional years)

```
plt.gca().xaxis.set_major_locator(MaxNLocator(integer=True))
```

Remove grid lines

```
plt.grid(False)
```

Access the current axis

```
ax = plt.gca()
```

Remove the top and right spines (often associated with x2 and y2 axes)

```
ax.spines['top'].set_visible(False)  
ax.spines['right'].set_visible(False)
```

Add titles and labels

```
plt.title('Sales Over Time')  
plt.xlabel('Year')  
plt.ylabel('Sales')
```

Display grid and show the plot

```
plt.show()
```



Bridging the Gap

This cheat sheet preserves the SAS mental model while introducing Python equivalents, reducing the learning curve for experienced programmers.



Practical Translation

Side-by-side comparisons demonstrate how DATA step and PROC workflows translate to pandas, NumPy, seaborn, and matplotlib.



Future Applications

As organizations adopt Python alongside SAS, this resource serves as both a learning aid and ongoing reference for mixed programming environments.

References & Contact

Key References

- McKinney, W. (2010). Data Structures for Statistical Computing in Python. Proceedings of the 9th Python in Science Conference
- pandas Development Team. pandas Documentation. pandas.pydata.org/docs/
- NumPy Developers. NumPy Documentation. numpy.org/doc/



**The Global Healthcare
Data Science Community**

Contact Channels

📞 UK +44 1843 609600
✉ office@phuse.global
🌐 phuse.global

Social Media

🐦 [@phusetwitla](https://twitter.com/phusetwitla)
📘 [/phusebook](https://facebook.com/phusebook)
📺 [/phusetube](https://youtube.com/phusetube)
🌐 [/company/phuse](https://company.phuse)

Contact Information

Author: Jagadish Katam

Company: Princeps Technologies Inc.

Location: Hyderabad, India

Email: jagadish.katam@princepstech.com

Website: www.princepstech.com

Thank You