Paper CT01

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

Jagadish Katam, Princeps Technologies, Hyderabad, India

ABSTRACT

This SAS-to-Python cheat sheet serves as a practical guide for statistical programmers transitioning from SAS to Python. It offers side-by-side code comparisons for essential data operations, enabling users to translate their existing SAS skills into efficient Python workflows using pandas, NumPy, seaborn, and matplotlib. The guide covers a wide range of tasks, including dataset manipulation (drop, keep, rename), conditional filtering, creation of derived variables, sorting, grouping, and summarizing data. It demonstrates how to perform conversions between text and numeric formats. Advanced topics such as dataset merging, reshaping with pivot/melt (analogous to PROC TRANSPOSE). Moreover, this offers Python equivalents for common PROC FREQ and PROC SUMMARY routines and presents rich visualizations with seaborn to replicate PROC SGPLOT. Overall, this resource bridges the SAS-Python divide by offering intuitive and readable Python code alternatives.

INTRODUCTION

SAS has long been a foundational tool for statistical programmers, particularly for data manipulation, summarization, and reporting workflows built around the DATA step and PROC procedures. As Python adoption continues to grow in analytics and data science environments, many organizations now expect SAS programmers to extend their skill sets to include Python. However, most Python learning resources are written from a general programming perspective and do not align with the established mental models used by experienced SAS users. This paper introduces a first-of-its-kind SAS-to-Python cheat sheet designed specifically for statistical programmers, providing direct, side-by-side translations of common SAS tasks into Python using pandas, NumPy, seaborn, and matplotlib. Rather than teaching Python in isolation, the examples are organized around familiar SAS workflows, such as conditional filtering, BY-group processing, PROC FREQ and PROC SUMMARY equivalents, data reshaping, and reporting to help SAS programmers efficiently transfer their existing knowledge into practical Python implementations.

Python Library Imports

All Python examples in this paper use pandas and NumPy for data manipulation and computation, along with matplotlib and seaborn for visualization, to provide Python equivalents for common SAS DATA step and PROC-based workflows.

```
+-------------------------------------------------+
| import pandas as pd                             |
| import numpy as np                              |
| import matplotlib.pyplot as plt                 |
| import seaborn as sns                           |
| from matplotlib.ticker import MaxNLocator       |
+-------------------------------------------------+
```

Creating a New Dataset

In Python, assigning a DataFrame to a new variable creates a reference to the original object. To create an independent copy, the `copy()` method is used, which most closely matches SAS DATA step behavior.

SAS	Python

`data new_data;`
`  set old_data;`
`run;`
	`old_data = pd.DataFrame(data)`
	`new_data = old_data.copy()`

Keeping and Dropping Variables

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	Python

`data new_data (keep=id);`
`  set old_data (drop=job_title);`
`run;`
	`# Drop a variable`
	`new_data =`
	`old_data.drop(columns=["job_title"])`
	
	`# Keep a variable`
	`new_data = old_data[["id"]]`

Dropping Variables Using Name Patterns

In pandas, this behavior can be replicated by dynamically identifying column names using string matching and then dropping the resulting list of variables.

SAS	Python

`data new_data (drop=temp:);`
`  set old_data;`
`run;`
	`new_data = old_data.drop(`
	`  columns=[`
	`  col for col in old_data.columns`
	`  if col.startswith("temp")`
	`  ]`
	`)`

Renaming Variables

The pandas `rename()` method provides functionality equivalent to the SAS RENAME statement and supports renaming one or multiple variables as needed.

SAS	Python

`data new_data;`
`  set old_data;`
`  rename old_name = new_name;`
`run;`
	`new_data = old_data.rename(`
	`  columns={`
	`  "old_name": "new_name"`
	`  }`
	`)`

CONDITIONAL FILTERING

Filtering Using an IN List

In SAS, the `IN` operator is used within a DATA step to retain observations that match any value in a specified list. In Python, the pandas `isin()` method provides equivalent functionality by returning a Boolean mask that is used to subset the DataFrame.

SAS	Python

`data new_data;`
`  set old_data;`
`  if year in (2010, 2011, 2012);`
`run;`
	`new_data = old_data[`
	`  old_data["year"].isin(`
	`  [2010, 2011, 2012]`
	`  )]`

Selecting the First Observation per BY Group

Unlike SAS, pandas require explicit sorting before group-based operations. Once sorted, the combination of groupby()` and `head()` replicates the behavior of the FIRST. indicator.

SAS	Python
data new_data; set old_data; by id; if first.id; run;	new_data = (old_data .sort_values("id") .groupby("id") .head(1))

Filtering Based on Date Values

In Python, date variables must be explicitly converted to datetime format before comparison. Once converted, filtering logic closely mirrors SAS date-based conditions.

SAS	Python
data new_data; set old_data; if dob > "25APR1990"d; run;	old_data["dob"] = pd.to_datetime(old_data["dob"]) new_data = old_data[old_data["dob"] > "1990-04-25"]

NEW VARIABLES AND CONDITIONAL EDITING

Creating a Derived Variable

This example shows how to create a new variable by performing arithmetic operations on existing variables.

SAS	Python
data new_data; set old_data; total_income = wages + benefits ; run;	**# Approach 1: Direct assignment** old_data["total_income"] = (old_data["wages"] + old_data["benefits"]) new_data = old_data **# Approach 2: Using assign()** new_data = old_data.assign(total_income=old_data["wages"] + old_data["benefits"]) **# Approach 3: Using eval()** old_data.eval("total_income = wages + benefits", inplace=True) new_data = old_data.copy()

Conditional Assignment Using IF/ELSE Logic

While the `apply()` approach mirrors row-wise SAS logic, the NumPy-based approach is fully vectorized and is generally preferred for performance and clarity in Python. Approach 1 is less efficient on large datasets and Approach 2 is preferred for performance.

SAS	Python

`data new_data;`
`  set old_data;`
`  if hours > 30 then full_time = "Y";`
`  else full_time = "N";`
`run;`
	`# Approach 1: Using apply()`
	`old_data["full_time"] = (`
	`  old_data["hours"]`
	`  .apply(lambda x: "Y" if x > 30 else "N")`
	`)`
	`new_data = old_data`
	
	`# Approach 2: Using NumPy (vectorized)`
	`old_data["full_time"] = np.where(`
	`  old_data["hours"] > 30, "Y", "N"`
	`)`
	`new_data = old_data.copy()`

Conditional Assignment with Multiple Conditions

In Python, multi-branch conditional logic can be implemented using functions combined with `apply()`. This approach provides flexibility and closely resembles the IF–ELSE structure commonly used in SAS DATA steps.

SAS	Python

`data new_data;`
`  set old_data;`
`  if temp > 20 then weather = "Warm";`
`  else if temp > 10 then weather = "Mild";`
`  else weather = "Cold";`
`run;`
	`# np.select() (Vectorized Conditions)`
	
	`conditions = [`
	`  old_data["temp"] > 20,`
	`  old_data["temp"] > 10`
	`]`
	`choices = ["Warm", "Mild"]`
	
	`old_data["weather"] = np.select(`
	`  conditions,`
	`  choices,`
	`  default="Cold"`
	`)`
	
	`# Define a function to implement conditional logic`
	`def calculate_weather(x):`
	`  if x > 20:`
	`  return "Warm"`
	`  elif x > 10:`
	`  return "Mild"`
	`  else:`
	`  return "Cold"`
	
	`# Approach 1: Apply a user-defined function`
	`old_data["weather"] = (`
	`  old_data["temp"]`
	`  .apply(calculate_weather)`
	`)`
	`new_data = old_data.copy()`

COUNTING AND SUMMARISING

One-Way Frequency Counts

This example shows how to generate a one-way frequency table for a categorical variable.

SAS	Python

`proc freq data = old_data ;`
`  table job_type ;`
`run;`
	`count =`
	`old_data["job_type"].value_counts(dropna=False)`
	`print(count)`

Two-Way Frequency Counts

In SAS, a two-way table is specified using the asterisk (*) notation. In pandas, grouping by multiple variables and applying `size()` produces an equivalent count of observations for each combination.

SAS	Python

`proc freq data=old_data;`
`  tables job_type*region;`
`run;`
	`count = (`
	`  old_data`
	`  .groupby(["job_type", "region"],`
	`  dropna=False)`
	`  .size()`
	`  .reset_index(name="count")`
	`)`
	
	`print(count)`

Grouped Summaries Using PROC SUMMARY

In this example, the groupby() function is used to group the data by job_type and region, which is equivalent to specifying CLASS variables in PROC SUMMARY. Setting as_index=False ensures that the grouping variables remain as regular columns in the output dataset, producing a flat, tabular structure suitable for reporting. The agg() function applies multiple summary statistics in a single step, calculating the total salary using the sum function and the number of records using the count function. This approach closely mirrors the core aggregation behavior of PROC SUMMARY with the NWAY option, generating one output row per unique combination of the grouping variables.

SAS	Python

`proc summary data=old_data nway;`
`  class job_type region;`
`  var salary;`
`  output out=new_data`
`  sum(salary)=total_salaries;`
`run;`
	`new_data = (`
	`  old_data`
	`  .groupby(["job_type", "region"],`
	`as_index=False)`
	`  .agg(`
	`  total_salaries=("salary", "sum"),`
	`  record_count=("salary", "count")`
	`  )`
	`)`

SORTING AND ROW-WISE OPERATIONS

Sorting with Multiple Keys and Order

This example demonstrates sorting data by multiple variables, including descending order.

SAS	Python
proc sort data=old_data out=new_data; by id descending income ; run;	old_data = pd.DataFrame(data) new_data = old_data.sort_values(by=["id", "income"], ascending=[True, False])

CONVERSIONS BETWEEN TEXT AND NUMERIC FORMATS

Numeric to character

In SAS, the `PUT` function converts a numeric variable to a character variable using a specified format, where the format controls the width and appearance of the resulting value. In Python, numeric-to-character conversion is performed using the `astype(str)` method, which converts each numeric value to its string representation. While Python does not apply fixed-width formatting by default, this approach provides the most common and practical equivalent for character conversion.

SAS	Python
data new_data; set old_data ; text_var = put(num_var, best.); run;	old_data = pd.DataFrame(data) old_data["text_var"] = (old_data["num_var"] .astype(str)) new_data = old_data.copy()

Character to numeric

In Python, character-to-numeric conversion can be performed using the astype() method when values are known to be numeric, or the to_numeric() function when additional flexibility is required. These approaches provide functionality equivalent to the SAS INPUT function, converting character values into numeric form for subsequent analysis.

SAS	Python
data new_data; set old_data ; num_var = input(text_var, 8.); run;	old_data = pd.DataFrame(data) **# Approach 1: Using astype()** old_data["num_var"] = (old_data["text_var"] .astype(int)) **# Approach 2: Using to_numeric()** old_data["num_var"] = pd.to_numeric(old_data["text_var"]) new_data = old_data.copy()

COMBINING DATASETS

Appending Datasets

In Python, datasets can be appended vertically using the pd.concat() function. Specifying axis=0 stacks the datasets row-wise, while ignore_index=True resets the row index to create a continuous sequence. This approach provides functionality equivalent to the SAS SET statement when combining multiple datasets.

SAS	Python

data new_data ;
set data_1 data_2 ;
run;
	data_1 = pd.DataFrame(data1)
	data_2 = pd.DataFrame(data2)
	
	# Concatenate datasets row-wise
	new_data = pd.concat(
	  [data_1, data_2],
	  axis=0,
	  ignore_index=True
	)

Left Join Using MERGE and IN= Logic

In Python, a left join is performed using the merge() function with how="left", which retains all observations from the left dataset. The indicator=True option adds a _merge column that identifies whether each record originated from the left dataset only, the right dataset only, or both datasets, providing functionality similar to SAS IN= dataset options.

SAS	Python
data new_data; merge data_1(in=in_1) data_2; by id; if in_1; run;	data_1 = pd.DataFrame(data1) data_2 = pd.DataFrame(data2) new_data = pd.merge(data_1, data_2, on="id", how="left", indicator=True)

STRING MANIPULATION

string-based filtering

In Python, string-based filtering is performed using the str.contains() method, which returns a Boolean mask that can be used to subset a DataFrame. The case=False option enables case-insensitive matching, while na=False ensures that missing values do not cause errors during evaluation. This approach provides functionality equivalent to the SAS FIND function used within a DATA step.

SAS	Python
data new_data; set old_data; if find(job_title , "Health"); run;	old_data = pd.DataFrame(data) new_data = old_data[old_data["job_title"] .str.contains("Health", case=False, na=False)]

substrings

In Python, substrings are extracted using zero-based indexing with the str[start:end] notation, where the end position is exclusive. This approach provides functionality equivalent to the SAS SUBSTR function, which uses one-based indexing and an explicit length parameter.

SAS	Python
data new_data; set old_data; substring = substr(big_string,3, 4); run;	old_data = pd.DataFrame(data) old_data["substring"] = (old_data["big_string"] .str[2:6]) new_data = old_data.copy()

TRANSPOSE/PIVOT

Reshaping data from long to wide format

In Python, reshaping data from long to wide format is performed using the pivot() method, which creates new columns based on distinct values of a specified variable. This approach provides functionality equivalent to PROC TRANSPOSE in SAS, where the BY, ID, and VAR statements control the structure of the output dataset.

SAS	Python
proc transpose data=long_data out=wide_data; by student ; id subject ; var grade ; run;	long_data = pd.DataFrame(data) wide_data = (long_data .pivot(index="student", columns="subject", values="grade") .reset_index())

Reshaping data from wide to long format

In Python, reshaping data from wide to long format is performed using the melt() function. The id_vars argument specifies identifier variables that remain unchanged, while value_vars identifies the columns to be transposed into rows. The var_name and value_name parameters control the names of the resulting variables, providing functionality equivalent to PROC TRANSPOSE with the NAME= option in SAS.

SAS	Python

proc transpose data=wide_data
out=long_data(rename=(col1=grade))
name=subject;
by student ;
var English Irish Maths;
run;
	wide_data = pd.DataFrame(data)
	
	# Identify ID variables
	id_vars = ["student"]
	
	# Dynamically determine value variables
	value_vars = [
	  col for col in wide_data.columns
	  if col not in id_vars

	]
	
	# Convert wide data to long format
	long_data = wide_data.melt(
	    id_vars=id_vars,
	    value_vars=value_vars,
	    var_name="subject",
	    value_name="grade"
	)

Creating functions to modify datasets

In Python, reusable data manipulation logic can be encapsulated in a function, similar to a SAS macro. The function accepts a DataFrame as input, modifies it by creating a new variable, and returns the updated dataset. This approach provides a simple and readable alternative to SAS macro-based DATA step processing.

SAS	Python
%macro add_variable(dataset_name); data &dataset_name; set &dataset_name; new_variable = 1; run; %mend; %add_variable(my_data);	def add_variable(dataset_name): dataset_name['new_variable'] = 1 return dataset_name my_data = add_variable(my_data)

FIGURES/GRAPHS

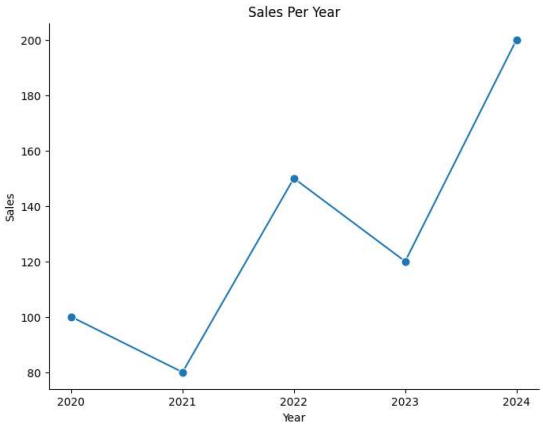
Line Plot Over Time

The following example demonstrates how a line plot with overlaid points is created in SAS using PROC SGPLOT and the equivalent implementation in Python using seaborn and matplotlib.

SAS	Python

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;
	# 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()



CONCLUSION

This paper presented a practical, first-of-its-kind SAS-to-Python cheat sheet designed specifically for statistical programmers seeking to transition from SAS to Python. By organizing examples around familiar SAS DATA step and PROC workflows and providing clear, side-by-side code comparisons, the paper demonstrated how common data manipulation, summarization, and visualization tasks can be translated into efficient Python implementations using pandas, NumPy, seaborn, and matplotlib. Rather than introducing Python concepts in isolation, the approach emphasized preserving the SAS mental model to reduce the learning curve and improve readability for experienced SAS users. As organizations increasingly adopt Python alongside or in place of SAS, this cheat sheet can serve both as an introductory learning aid and as an ongoing reference for practitioners working in mixed programming environments. Future work may extend this approach to additional SAS procedures, performance considerations, and integration with larger Python-based analytics workflows.

REFERENCES

McKinney, W. (2010). *Data Structures for Statistical Computing in Python*. Proceedings of the 9th Python in Science Conference (SciPy).

pandas Development Team. *pandas Documentation*. https://pandas.pydata.org/docs/

NumPy Developers. *NumPy Documentation*. https://numpy.org/doc/

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Author Name: Jagadish Katam
Company: Princeps Technologies Inc.
Address: Hyderabad, India
Email: jagadish.katam@princepstech.com
Website: https://www.princepstech.com/