PharmaSUG 2026 - Paper-223

The Evolution of Open-Source Technologies in the Pharmaceutical Industry: Python as a Cost-Effective Solution for Clinical Statistical Programming

Ramesh Potluri, Servier Pharmaceuticals

ABSTRACT

SAS has traditionally been the standard platform for statistical analysis and clinical reporting in the pharmaceutical industry. However, advancements in open-source technologies and evolving regulatory perspectives have driven increased adoption of alternative tools that support automation and advanced analytics. Python offers a powerful and cost-effective solution for automated tool development and machine learning integration. This paper describes the setup of Python and associated development environments, reviews key libraries and functions that enable SAS-equivalent functionality — including data type conversion, sorting, case manipulation, merging, transposing, subsetting, and conditional column creation — and presents practical examples of demographic and adverse event table generation aligned with clinical reporting requirements. AI could generate programming, but programmers who know the language efficiently handle AI.

INTRODUCTION

SAS has anchored pharmaceutical statistical programming for decades — from SDTM mapping and ADaM derivation through TFL generation and FDA submission packages. The FDA's 2021 Statistical Software Clarifying Statement confirmed that submissions using validated open-source tools are acceptable (FDA, 2021), and Python adoption in clinical trials has grown steadily as a result.

Python brings three concrete advantages over SAS: zero licensing cost, native machine learning integration, and a data-wrangling ecosystem that mirrors SAS data-step logic closely enough that experienced clinical programmers can be productive within weeks. This paper delivers a complete, reproducible clinical reporting workflow — from environment setup through ADaM data loading, core Python operations, and formatted TFL output — with side-by-side Python, R, and SAS comparisons throughout.

The two TFLs demonstrated — a Demographics and Baseline Characteristics table (TFL-01) and an Adverse Events by SOC/PT table (TFL-02) — are required outputs in virtually every clinical study report (CSR) and collectively exercise the full set of programming operations needed in daily clinical work. All variable names follow CDISC ADaM Implementation Guide v1.4 conventions (CDISC, 2023).

INSTALLATION AND ENVIRONMENT SETUP

Anaconda Python Distribution

Download Anaconda from https://www.anaconda.com/docs/main. It bundles Python, Jupyter, the conda package manager, pandas, numpy, and scipy in a single installer — analogous to a full SAS installation. Conda virtual environments pin package versions per project, preventing dependency conflicts common in shared clinical programming servers.

Visual Studio Code

VS Code (https://code.visualstudio.com) is the recommended editor. Key features for clinical programmers include: Explorer panel for project navigation (comparable to SAS Enterprise Guide), integrated terminal, Jupyter cell-mode execution, and a debug pane with breakpoints (F9) and step-through (F5).

Key Python Libraries

- pandas — data manipulation: filtering, grouping, merging, pivoting
- numpy — numeric operations: np.where(), np.select(), rounding
- pyreadstat — read SAS7BDAT and XPT files directly into DataFrames
- os / pathlib — file and directory management for batch workflows

PYTHON METHODOLOGY: CORE OPERATORS AND FUNCTIONS FOR TFL OUTPUTS

Clinical TFL programming relies on a consistent set of data manipulation operations regardless of language. The table below maps each essential operation to its Python implementation. These operations collectively underpin every demographic summary, adverse event table, laboratory shift table, and efficacy analysis output in clinical trials.

Operation	Python Code Example

1. Convert Data Types (numeric, string, date)	`df["AGE"] = pd.to_numeric(df["AGE"], errors="coerce")  # to numeric; invalid -> NaN`
	`df["USUBJID"] = df["USUBJID"].astype(str)  # to string`
	`df["RFSTDTC"] = pd.to_datetime(df["RFSTDTC"], errors="coerce")  # to date`

2. Sort Rows of Data	`df = df.sort_values("TRT01P")  # single key`
	`df = df.sort_values(["AEBODSYS", "AEDECOD", "TRT01A"])  # multiple keys`
	`df = df.reset_index(drop=True)  # reset index after sort`

3. Change Values to Upper / Lower Case	`df["SEX"]  = df["SEX"].str.upper()  # UPPERCASE (SAS: upcase())`
	`df["param"]  = df["param"].str.lower()  # lowercase (SAS: lowcase())`
	`df["AEBODSYS"] = df["AEBODSYS"].str.title()  # Title Case`
	`df["PARAM"]  = df["PARAM"].str.strip()  # remove leading/trailing spaces`

4. Identify Invalid Data Using IN and NOT IN	`valid  = df[df["SEX"].isin(["M", "F"])]  # IN (SAS: where sex in ("M","F"))`
	`invalid = df[~df["RACE"].isin([  # NOT IN (~ = logical NOT)`
	`  "WHITE", "BLACK OR AFRICAN AMERICAN",`
	`  "ASIAN", "AMERICAN INDIAN OR ALASKA NATIVE", "OTHER"])]`
	`print(invalid["RACE"].value_counts())  # review unexpected values`

5. Concatenate Rows (stack datasets)	`combined = pd.concat([age_rows, sex_race_rows], ignore_index=True)`
	`# SAS: set a b;`
	`all_ae  = pd.concat([ae_overall, aesoc, socpt], ignore_index=True)`

6. Extracting Statistics	`# PROC FREQ equivalent — frequency counts`
	`freq = (df.groupby('TRT01P')`
	`  .size().reset_index(name='count'))`
	
	`# PROC MEANS — multiple stats in one call`
	`stats = (df.groupby('TRT01P')`
	`  .agg(n  = ('AGE','count'),`
	`  mean  = ('AGE','mean'),`
	`  std  = ('AGE','std'),`
	`  median = ('AGE','median'),`
	`  q1  = ('AGE', lambda x: x.quantile(.25)),`
	`  q3  = ('AGE', lambda x: x.quantile(.75)),`
	`  min  = ('AGE','min'),`
	`  max  = ('AGE','max'))`
	`  .reset_index())`

7. Merge / Join Tables (Inner, Left, Outer)	`adae_s = ae_s.merge(adsl_s, on="USUBJID", how="inner")  # inner (SAS: if a and b)`
	`s_r  = sex_race.merge(denom, on="TRT01P", how="left")  # left (SAS: if a)`
	`full  = df1.merge(df2, on="KEY", how="outer")  # outer (SAS: if a or b)`

8. Transpose Tables (Long to Wide)	`final = df.pivot_table(  # SAS: proc transpose`
	`  index=["section", "sub", "stat"],`
	`  columns="TRT01P",`

	values="value", aggfunc="first").reset_index() final.columns = ["section","sub","stat"] + [f"col{c}" for c in final.columns[3:]]

9. Select (Keep) Specific Columns	adsl_s = adsl[["USUBJID","TRT01P","AGE","SEX","RACE"]] # keep (SAS: keep=)
	df = df.drop(columns=["_NAME_","ord","ord2"]) # drop (SAS: drop=)
	col_cols = [c for c in df.columns if c.startswith("col")] # pattern match

10. Subset / Filter Rows of Data	itt = adsl[adsl["ITTFL"] == "Y"] # single condition
	teae = adae[(adae["TRTEMFL"]=="Y") & # multiple conditions
	(adae["AESEV"].isin(["SEVERE","MODERATE"]))]
	itt2 = adsl.query("ITTFL == 'Y'") # query() alternative

11. Dynamically Create a New Column	df["value"] = np.where(# SAS if/else equivalent df["count"] > 0, df["count"].astype(str) + " (" + (100 * df["count"] / df["bign"]).round(1).astype(str) + ")", "0 (0.0)") conditions = [df["AEBODSYS"]=="", df["AEBODSYS"]!=""] choices = ["Subjects with any TEAE", df["AEBODSYS"].str.strip()] df["label"] = np.select(conditions, choices) # np.select = SAS select/when

12. Count Distinct Values for a Single Column	df["USUBJID"].nunique() # total unique subjects
	df.groupby("TRT01A")["USUBJID"].nunique() # unique subjects by group
	df["RACE"].value_counts() # frequency table
	df_uniq = df.drop_duplicates(subset=["USUBJID","TRT01A"]) # SAS nodupkey
	n = df_uniq.groupby("TRT01A").size().reset_index(name="n")

These twelve operations are sufficient to produce the majority of clinical TFL outputs. The pandas library implements all of them in a syntax that experienced SAS programmers find conceptually familiar within a short learning period.

PYTHON CLINICAL REPORTING WORKFLOW

The end-to-end Python pipeline for clinical TFL generation mirrors standard SAS practice and consists of five steps: (1) select the analysis population using population flags, (2) compute statistics or frequency counts, (3) merge with treatment arm denominators to form n(%) and (4) sort and transpose from long to wide format. Each step maps directly to a pandas or numpy operation, as demonstrated in Sections 5 and 6.

DEMOGRAPHICS TABLE (TFL-01)

Table Shell

Table 1 presents the standard demographics shell consistent with CDISC ADaM and clinical study report conventions

Parameter	Statistic	Placebo (N=XX)	Drug A (N=XX)	Drug B (N=XX)	Total (N=XX)
Age (Years)	N				
	Mean (SD)				
	Median				

	Q1, Q3				
	Min, Max				
Sex					
Female	n (%)				
Male	n (%)				
Race					
White	n (%)				
Black or African American	n (%)				
Amer. Indian or AK Native	n (%)				
Asian	n (%)				
Other	n (%)				

Table 1. Demographics and Baseline Characteristics - Table Shell

Step-by-Step Code: Python, R, and SAS

The following table provides complete, step-by-step code for generating the Demographics table in all three languages. Each line is preserved exactly as implemented.

Operation	Python	R	SAS

Package Installation	`import pandas as pd`
	`import numpy as np`
		`library(haven)`
		`library(dplyr)`
		`library(tidyr)`
		`library(stringr)`
			`NA`

Read in ADSL	`adsl = pd.read_sas(`
	`  "path/adsl.sas7bdat",`
	`  encoding='latin1')`
		`adsl <-read_xpt(`
		`  "path/adsl.xpt")`
			`libname adam "path"`
			`  access=readonly;`

Select ITT Population	`adsl_itt = adsl[`
	`  adsl["ITTFL"]=="Y"`
	`].copy()`
	# Convert AGE to numeric
	`adsl_itt["AGE"] = (`
	`  pd.to_numeric(`
	`  adsl_itt["AGE"],`
	`  errors="coerce"))`
		`adsl_itt <- adsl %>%`
		`  filter(ITTFL == "Y")`
			`data adsl_itt;`
			`  set adsl;`
			`  where ittfl ='Y';`
			`run;`

Denominator Counts	`denom = (`
	`  adsl_itt`
	`  .groupby("TRT01P")`
	`  .size()`
	`  .reset_index(`
	`  name="bign")`
	`  .sort_values("TRT01P")`
	`)`
		`denom <- adsl_itt %>%`
		`  count(TRT01P,`
		`  name="bign") %>%`
		`  arrange(TRT01P)`
			/*Denominators*/
			`proc freq`
			`  data=adsl_itt noprint;`
			`  tables trt01p /`
			`  out=denom(rename=`
			`  (count=bign));`
			`run;`
			`proc sort data=denom;`
			`  by trt01p;`
			`run;`

Age Summary (Continuous)	`age_stats = (`
	`  adsl_itt`
	`  .groupby("TRT01P")`
	`  .agg(`
	`  n=("AGE","count"),`
	`  mean=("AGE","mean"),`
	`  std=("AGE","std"),`
	`  median=("AGE",`
	`  "median"),`
	`  q1=("AGE",`
	`  lambda x:`
	`  x.quantile(0.25)),`
	`  q3=("AGE",`
	`  lambda x:`
	`  x.quantile(0.75)),`
	`  min=("AGE","min"),`
	`  max=("AGE","max"))`
	`  .reset_index()`
	`)`
		`age_stats <-`
		`  adsl_itt %>%`
		`  group_by(TRT01P) %>%`
		`  summarise(`
		`  n=sum(!is.na(AGE)),`
		`  mean=mean(AGE,`
		`  na.rm=TRUE),`
		`  std=sd(AGE,`
		`  na.rm=TRUE),`
		`  median=median(AGE,`
		`  na.rm=TRUE),`
		`  q1=quantile(AGE,`
		`  0.25,na.rm=TRUE),`
		`  q3=quantile(AGE,`
		`  0.75,na.rm=TRUE),`
		`  min=min(AGE,`
		`  na.rm=TRUE),`
		`  max=max(AGE,`
		`  na.rm=TRUE),`
		`  .groups="drop")`
			`proc means`
			`  data=adsl_itt`
			`  n mean std`
			`  median q1 q3`
			`  min max;`
			`  class trt01p;`
			`  var age;`
			`  output out=age_stats`
			`  (drop=_type_`
			`  _freq_)`
			`  n=n mean=mean`
			`  std=std`
			`  median=median`
			`  q1=q1 q3=q3`
			`  min=min max=max;`
			`run;`

Sex and Race Frequencies (Categorical)	sex_count = (
	  adsl_itt
	  .groupby(
	  ["SEX","TRT01P"])
	  .size()
	  .reset_index(
	  name="count"))
	race_count = (
	  adsl_itt
	  .groupby(
	  ["RACE","TRT01P"])
	  .size()
	  .reset_index(
	  name="count"))
	# SEX block
	sex_count=(
	  sex_count.assign(
	  section="Sex",
	  ord=2,
	  ord2=sex_count["SEX"]
	  .apply(lambda x:
	  1 if x=="F" else 2),
	  sub=sex_count["SEX"]))
	# RACE block
	def race_order(r):
	  if r=="WHITE":return 1
	  if r=="ASIAN":return 2
	  if r=="BLACK OR " "AFRICAN AMERICAN":
	  return 3
	  if r=="AMERICAN INDIAN" " OR ALASKA NATIVE":
	  return 4
	  if r=="MISSING":return 5
	  return 99
	race_count=(
	  race_count.assign(
	  section="Race",
	  ord=3,
	  ord2=race_count["RACE"]
	  .apply(race_order),
	  sub=race_count["RACE"]))
	sex_race=pd.concat(
	  [sex_count,race_count],
	  ignore_index=True)
		# SEX counts
		sex_count <-
		  adsl_itt %>%
		  count(SEX,TRT01P,
		  name="count")
		# RACE counts
		race_count <-
		  adsl_itt %>%
		  count(RACE,TRT01P,
		  name="count")
		# Combine + order
		sex_race <- bind_rows(
		  sex_count %>%
		  mutate(
		  section="Sex",
		  ord=2,
		  ord2=if_else(
		  SEX=="F",1,2),
		  sub=SEX),
		  race_count %>%
		  mutate(
		  section="Race",
		  ord=3,
		  ord2=case_when(
		  RACE=="WHITE"~1,
		  RACE=="ASIAN"~2,
		  RACE=="BLACK OR "
		"AFRICAN" " AMERICAN"~3,
		  RACE=="AMERICAN" "
		INDIAN OR" " ALASKA"
		" NATIVE"~4,
		  RACE=="MISSING"~5,
		  TRUE~99),
		  sub=RACE))
			proc freq
			  data=adsl_itt noprint;
			  tables sex*trt01p /
			  out=sex_count
			  (drop=percent);
			run;
			proc freq
			  data=adsl_itt noprint;
			  tables race*trt01p /
			  out=race_count
			  (drop=percent);
			run;
			data sex_race;
			  length sub $40
			  section $30;
			  set sex_count(in=a)
			  race_count(in=b);
			  /*SEX block*/
			  if a then do;
			  section='Sex';
			  ord=2;
			  if sex="F"
			  then ord2=1;
			  else ord2=2;
			  sub=sex;
			  end;
			  /*RACE block*/
			  else if b then do;
			  section='Race';
			  ord=3;
			  if race="WHITE"
			  then ord2=1;
			  else if race="ASIAN"
			  then ord2=2;
			  else if race=
			  "BLACK OR AFRICAN"
			  ||" AMERICAN"
			  then ord2=3;
			  else if race=
			  "AMERICAN INDIAN"
			  ||" OR ALASKA"
			  ||" NATIVE"
			  then ord2=4;
			  else if race=
			  "MISSING"
			  then ord2=5;
			  else ord2=99;
			  sub=race;
			  end;
			run;

Merge Counts with Denominators to Compute N(%)	s_r_perc=(
	  sex_race.merge(
	  denom,
	  on="TRT01P",
	  how="left"))
	s_r_perc["value"]=(
	  np.where(
	  s_r_perc["count"]>0,
	  s_r_perc["count"]
	  .astype(str)
	  +" ("+
	  (100*s_r_perc
	  ["count"]
	  /s_r_perc["bign"])
	  .round(1)
	  .astype(str)+")",
	  "0 (0.0)"))
	s_r_perc["stat"]="n(%)"
		s_r_perc <-
		  sex_race %>%
		  left_join(denom,
		  by="TRT01P") %>%
		  mutate(
		  value=ifelse(
		  count>0,
		  paste0(count,
		  " (",
		  sprintf("%.1f",
		  (count/bign)
		  *100),")"),
		  "0 (0.0)"),
		  stat="n(%)")
			proc sort
			  data=sex_race;
			  by trt01p section;
			run;
			data s_r_perc;
			  merge sex_race
			  denom;
			  by trt01p;
			  if count>0 then
			  value=
			  strip(put(
			  count,3.))
			  ||" ("
			  ||strip(put(
			  (count/bign)
			  *100,5.1))
			  ||")";
			  else
			  value="0 (0.0)";
			  stat="n(%)";
			run;

Age Continuous Statistics Formatting	age_rows=[]
	for _,row in
	  age_stats.iterrows():
	  trt=row["TRT01P"]
	  age_rows.append([
	  1,"Age (Years)",
	  1," ",trt,"n",
	  str(int(row["n"]))])
	  age_rows.append([
	  1,"Age (Years)",
	  2," ",trt,
	  "Mean, SD",
	  f"{row['mean']:.1f}"
	  f" , {row['std']:.2f}"])
	  age_rows.append([
	  1,"Age (Years)",
	  3," ",trt,"Median",
	  f"{row['median']:.1f}"])
	  age_rows.append([
	  1,"Age (Years)",
	  4," ",trt,"Q1, Q3",
		age_s <- age_stats %>%
		  filter(TRT01P!="")%>%
		  mutate(
		  section=
		  "Age (Years)",
		  ord=1) %>%
		  tidyr::uncount(5) %>%
		  group_by(TRT01P) %>%
		  mutate(
		  stat=c("n",
		  "Mean, SD",
		  "Median",
		  "Q1, Q3",
		  "Min, Max"),
		  ord2=1:5,
		  value=case_when(
		  stat=="n"~
		  as.character(n),
		  stat=="Mean, SD"~
		  paste0(
		  sprintf("%.1f",
			data age_s(where=(
			  trt01p ne ''));
			  length section
			  stat $30
			  value $40;
			  set age_stats;
			  section=
			  'Age (Years)';
			  ord=1;
			  stat="n";
			  ord2=1;
			  value=strip(put(
			  n,best.));
			  output;
			  stat="Mean, SD";
			  ord2=2;
			  value=
			  strip(put(
			  mean,5.1))
			  ||" , "
			  ||strip(put(

	f"{row['q1']:.1f}"
	f" , {row['q3']:.1f}"])
	age_rows.append([
	1,"Age (Years)",
	5," ",trt,"Min, Max",
	f"{row['min']:.1f}"
	f" , {row['max']:.1f}"])
	age_s=pd.DataFrame(
	age_rows,columns=[
	"ord","section",
	"ord2","sub",
	"TRT01P","stat",
	"value"])
		mean)," , ",
		sprintf("%.2f",
		std)),
		stat=="Median"~
		sprintf("%.1f",
		median),
		stat=="Q1, Q3"~
		paste0(
		sprintf("%.1f",
		q1)," , ",
		sprintf("%.1f",
		q3)),
		stat=="Min, Max"~
		paste0(
		sprintf("%.1f",
		min)," , ",
		sprintf("%.1f",
		max)))) %>%
		ungroup()
			std,5.2));
			output;
			stat="Median";
			ord2=3;
			value=strip(put(
			median,5.1));
			output;
			stat="Q1, Q3";
			ord2=4;
			value=
			strip(put(
			q1,5.1))
			||" , "
			||strip(put(
			q3,5.1));
			output;
			stat="Min, Max";
			ord2=5;
			value=
			strip(put(
			min,5.1))
			||" , "
			||strip(put(
			max,5.1));
			output;
			run;

Combine Age + Sex/Race	pref=pd.concat([age_s, s_r_perc[["ord","section", "ord2","TRT01P", "stat","value"]]], ignore_index=True)	pref <- bind_rows(age_s, s_r_perc)	data pref; set age_s s_r_perc; run;

Transpose to Table Format	pref_sorted=(
	pref.sort_values([
	"ord","section",
	"ord2","sub",
	"stat"]))
	final=(
	pref_sorted
	.pivot_table(
	index=["ord",
	"section","ord2",
	"sub","stat"],
	columns="TRT01P",
	values="value",
	aggfunc="first")
	.reset_index())
	# Rename columns
	final.columns=(
	["ord","section",
	"ord2","sub","stat"]
	+[f"col{c}" for c
	in final.columns[5:]])
		pref_sorted <-
		pref %>%
		arrange(ord,section,
		ord2,sub,stat)
		final <-
		pref_sorted %>%
		select(ord,section,
		ord2,sub,stat,
		TRT01P,value) %>%
		pivot_wider(
		names_from=TRT01P,
		values_from=value,
		names_prefix="col")
			proc sort data=pref;
			by ord section
			ord2 sub stat;
			run;
			proc transpose
			data=pref
			out=final
			prefix=col;
			by ord section
			ord2 sub stat;
			id trt01p;
			var value;
			run;

ADVERSE EVENT TABLE (TFL-02)

Table Shell

Table 2 presents the standard AE summary shell organized by System Organ Class (SOC) and Preferred Term (PT), consistent with MedDRA coding conventions.

System Organ Class / Preferred Term	Placebo (N=XX)	Drug A (N=XX)	Drug B (N=XX)	Total (N=XX)
Subjects with any TEAE				
System Organ Class 1				
Preferred Term 1	xx (xx%)	xx (xx%)	xx (xx%)	xx (xx%)
Preferred Term 2	xx (xx%)	xx (xx%)	xx (xx%)	xx (xx%)
System Organ Class 2				
Preferred Term 1	xx (xx%)	xx (xx%)	xx (xx%)	xx (xx%)
Preferred Term 2	xx (xx%)	xx (xx%)	xx (xx%)	xx (xx%)

Table 2. Treatment-Emergent Adverse Events by System Organ Class and Preferred Term - Shell

Step-by-Step Code: Python, R, and SAS

Operation	Python	R	SAS

Package Installation & Read ADSL/ADAE	`import pandas as pd`
	`import numpy as np`
	
	`adsl=pd.read_sas(`
	`  "path/adsl.sas7bdat",`
	`  encoding='latin1')`
	`adae=pd.read_sas(`
	`  "path/adae.sas7bdat",`
	`  encoding='latin1')`
		`library(dplyr)`
		`library(tidyr)`
		`library(haven)`
		
		`adsl<-read_sas(`
		`  "path/adsl.sas7bdat")`
		`adae<-read_sas(`
		`  "path/adae.sas7bdat")`
			`libname adam "path"`
			`  access=readonly;`
			`data adsl;`
			`  set adam.adsl;`
			`run;`
			`data adae;`
			`  set adam.adae;`
			`run;`

Select Safety Population; Filter to TEAEs; Merge ADSL+ADAE	`adsl_s=(adsl[`
	`  adsl["SAFFL"]=="Y"]`
	`  [["USUBJID",`
	`  "TRT01A"]])`
	`ae_s=adae[`
	`  adae["TRTEMFL"]`
	`  =="Y"]`
	`adae_s=(ae_s.merge(`
	`  adsl_s,`
	`  on="USUBJID",`
	`  how="inner"))`
		`adsl_s <- adsl %>%`
		`  filter(SAFFL=="Y")%>%`
		`  select(USUBJID,`
		`  TRT01A)`
		`ae_s <- adae %>%`
		`  filter(`
		`  TRTEMFL=="Y")`
		`adae_s <- ae_s %>%`
		`  inner_join(adsl_s,`
		`  by="USUBJID")`
			`data adsl_s;`
			`  set adam.adsl;`
			`  where saffl="Y";`
			`run;`
			`data ae_s;`
			`  set adam.adae;`
			`  where trtemfl="Y";`
			`run;`
			`data adae_s;`
			`  merge ae_s(in=a)`
			`  adsl_s(in=b);`
			`  by usubjid;`
			`  if a and b;`
			`run;`

Denominator N per Treatment Arm	`denom=(`
	`  adsl_s`
	`  .groupby("TRT01A")`
	`  .size()`
	`  .reset_index(`
	`  name="N"))`
		`denom <- adsl_s %>%`
		`  count(TRT01A,`
		`  name="N")`
			`proc freq`
			`  data=adsl_s`
			`  noprint;`
			`  tables trt01a /`
			`  out=denom(rename=`
			`  (count=N));`
			`run;`

Subject-Level Counts: Overall / SOC / SOC-PT (drop_duplicates = SAS nodupkey)	`ae1s=(`
	`  adae_s[["USUBJID",`
	`  "TRT01A"]]`
	`  .drop_duplicates()`
	`  .groupby("TRT01A")`
	`  .size()`
	`  .reset_index(`
	`  name="n"))`
	`aesoc=(`
	`  adae_s[["USUBJID",`
	`  "TRT01A",`
	`  "AEBODSYS"]]`
	`  .drop_duplicates()`
	`  .groupby([`
	`  "TRT01A",`
	`  "AEBODSYS"])`
	`  .size()`
	`  .reset_index(`
	`  name="n"))`
	`socpt=(`
	`  adae_s[["USUBJID",`
	`  "TRT01A",`
	`  "AEBODSYS",`
	`  "AEDECOD"]]`
	`  .drop_duplicates()`
	`  .groupby([`
	`  "TRT01A",`
	`  "AEBODSYS",`
	`  "AEDECOD"])`
	`  .size()`
	`  .reset_index(`
	`  name="n"))`
		`ae1s <- adae_s %>%`
		`  distinct(USUBJID,`
		`  TRT01A) %>%`
		`  group_by(TRT01A)%>%`
		`  summarise(n=n(),`
		`  .groups="drop")`
		`aesoc <- adae_s %>%`
		`  distinct(USUBJID,`
		`  TRT01A,`
		`  AEBODSYS) %>%`
		`  group_by(TRT01A,`
		`  AEBODSYS) %>%`
		`  summarise(n=n(),`
		`  .groups="drop")`
		`socpt <- adae_s %>%`
		`  distinct(USUBJID,`
		`  TRT01A,`
		`  AEBODSYS,`
		`  AEDECOD) %>%`
		`  group_by(TRT01A,`
		`  AEBODSYS,`
		`  AEDECOD) %>%`
		`  summarise(n=n(),`
		`  .groups="drop")`
			`/*Overall*/`
			`proc sort`
			`  data=adae_s`
			`  nodupkey`
			`  out=ae1;`
			`  by usubjid trt01a;`
			`run;`
			`proc freq`
			`  data=ae1 noprint;`
			`  tables trt01a /`
			`  out=ae1s(rename=`
			`  (count=n));`
			`run;`
			`/*By SOC*/`
			`proc sort`
			`  data=adae_s`
			`  nodupkey`
			`  out=aesoc_u;`
			`  by usubjid trt01a`
			`  aebodsys;`
			`run;`
			`proc freq`
			`  data=aesoc_u`
			`  noprint;`
			`  tables`
			`  trt01a*aebodsys/`
			`  out=aesoc(rename=`
			`  (count=n));`
			`run;`
			`/*By SOC+PT*/`
			`proc sort`
			`  data=adae_s`
			`  nodupkey`
			`  out=aesocpt;`
			`  by usubjid trt01a`
			`  aebodsys`
			`  aedecod;`
			`run;`

Stack Levels; Compute n(%); Transpose to Wide Format	`ae1s["ord"]=1`
	`ae1s["ord2"]=0`
	`aesoc["ord"]=2`
	`aesoc["ord2"]=1`
	`socpt["ord"]=2`
	`socpt["ord2"]=2`
	`all_ae=pd.concat(`
	`  [ae1s,aesoc,socpt],`
	`  ignore_index=True)`
	`ae_t=(all_ae.merge(`
	`  denom,`
	`  on="TRT01A",`
	`  how="left"))`
	`ae_t["result"]=(`
	`  np.where(`
	`  ae_t["n"]>0,`
		`ae1s<-mutate(ae1s,`
		`  ord=1,ord2=0)`
		`aesoc<-mutate(aesoc,`
		`  ord=2,ord2=1)`
		`socpt<-mutate(socpt,`
		`  ord=2,ord2=2)`
		`all_ae<-bind_rows(`
		`  ae1s,aesoc,socpt)`
		`ae_t <- all_ae %>%`
		`  left_join(denom,`
		`  by="TRT01A") %>%`
		`  mutate(result=`
		`  if_else(n>0,`
		`  paste0(n," (",`
		`  sprintf("%.1f",`
		`  (n/N)*100),")"),`
			`data all;`
			`  set ae1s (in=A)`
			`  aesoc(in=B)`
			`  socpt(in=C);`
			`  if A then do;`
			`  ord=1;ord2=0;`
			`  end;`
			`  else ord=2;`
			`  if B then ord2=1;`
			`  if C then ord2=2;`
			`run;`
			`data ae_t;`
			`  merge all(in=a)`
			`  denom;`
			`  by trt01a;`
			`  if a;`

```
    ae_t["n"]
    .astype(str)
    +" ("+
    (100*ae_t["n"]
    /ae_t["N"])
    .round(1)
    .astype(str)+")",
    "0 (0.0)"))
final=(ae_t
  .pivot_table(
    index=["ord",
      "AEBODSYS",
      "ord2",
      "AEDECOD"],
    columns="TRT01A",
    values="result",
    aggfunc="first")
  .reset_index())
```

```
    "0 (0.0)"))
final <- ae_t %>%
  pivot_wider(
    id_cols=c(ord,
      AEBODSYS,ord2,
      AEDECOD),
    names_from=
      TRT01A,
    values_from=
      result,
    names_prefix=
      "col")
```

```
  if n>0 then
    result=
      strip(put(
        n,3.))
      ||" ("
      ||strip(put(
        (n/N)*100,
        5.1))
      ||")";
  else
    result=
      "0 (0.0)";
run;
proc transpose
  data=ae_t
  out=final
  (drop=_name_)
  prefix=col;
  by ord aebodsys
     ord2 aedecod;
  id trt01a;
  var result;
run;
```

SAS VS. PYTHON VS. R: PLATFORM COMPARISON

Table 3 summarizes key differences across the three platforms most commonly used for pharmaceutical clinical statistical programming.

Feature	SAS	Python	R
License Cost	High (per-seat)	Free (open-source)	Free (open-source)
Read SAS7BDAT / XPT	Native	pyreadstat / pandas	haven / SASxport
ML / AI Integration	Limited	Excellent	Moderate
Regulatory Acceptance	Established	Accepted w/ validation	Accepted w/ validation
Scripting & Automation	Moderate	Excellent	Good
RTF / PDF Output	Excellent	Good (pharmaRTF-py)	Excellent (r2rtf)
Pharma Community	Large	Largest overall	Large
Learning Curve (SAS users)	—	~2-3 months	~2-3 months

Table 3. Platform Feature Comparison: SAS, Python, and R

Python and R eliminate per-seat licensing for clinical programming teams. Python's edge lies in its ML ecosystem and scripting flexibility; R's edge lies in mature pharma-specific RTF output libraries (r2rtf, pharmaRTF) with established regulatory history. Migration from SAS is comparable for both: the four core SAS procedure equivalents — PROC SORT, PROC MEANS, PROC FREQ, and PROC TRANSPOSE — all have direct pandas counterparts that experienced programmers typically master within two to three months.

Regulatory acceptance for Python and R requires validation per 21 CFR Part 11: pinned dependencies (requirements.txt or conda environment.yml), version-controlled scripts (Git), independent QC programming, and a Software Development Plan documenting reproducibility and audit trail.

CONCLUSION

This paper demonstrated that Python — using pandas, numpy, and pyreadstat — can replicate the full SAS clinical reporting. The twelve core methodology operations documented — data type conversion, sorting, case manipulation, IN/NOT IN filtering, row concatenation, inner/left/outer joins, transposition, column selection, row subsetting, conditional column creation, and distinct value counting — cover the majority of operations required in daily clinical TFL programming.

Python eliminates per-seat licensing costs, integrates natively with modern ML frameworks, and supports automation pipelines compatible with Git, CI/CD, and cloud environments. Clinical teams building Python competency today are well positioned for the next generation of adaptive trial designs, NLP-assisted medical coding review, and real-world evidence analytics. AI-assisted code generation accelerates initial script development, but clinical programmers with deep Python expertise remain essential for validation, edge-case resolution, and regulatory compliance.

REFERENCES

[1] FDA (2021). Statistical Software Clarifying Statement. U.S. Food and Drug Administration, Silver Spring, MD.

[2] CDISC (2023). ADaM Implementation Guide v1.4. Clinical Data Interchange Standards Consortium.

[3] PhUSE (2023). Open-Source Programming in Clinical Reporting. PhUSE Computational Science Symposium Technical Report.

[4] McKinney, W. (2010). Data Structures for Statistical Computing in Python. Proc. 9th Python in Science Conference, 56-61.

[5] Harris, C.R. et al. (2020). Array programming with NumPy. Nature, 585, 357-362.

[6] R Core Team (2023). R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna.

[7] SAS Institute Inc. (2023). SAS/STAT 15.3 User Guide. SAS Institute, Cary, NC.

[8] ICH E9 (1998). Statistical Principles for Clinical Trials. International Conference on Harmonisation.

[9] Anaconda, Inc. (2024). Anaconda Distribution Documentation. https://www.anaconda.com/docs/main

[10] PharmaSUG (2024). Proceedings of PharmaSUG 2024. Pharmaceutical Users Software Exchange. https://www.pharmasug.org

CONTACT INFORMATION

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

Ramesh Potluri
Servier Pharmaceuticals
ramesh.potluri1987@gmail.com
LinkedIn: www.linkedin.com/in/ramesh-potluri-a81bb6197