SAS vs. Python examples

Here is a list of some common SAS operations as well as an example of how to run the same code in Python. This will be available in the git repository for the Python-course and on the following internal web page:
http://146.192.254.241:8089/g020029/SAS_VS_PY.html

Let me know if there are any other commonly used SAS operations that you would like me to add!

Table of contents

Note: All examples use the `pandas` library. You only have to import it once per session. The import statement has been included in several examples, just remember that it can be ignored if `pandas` is already imported.

Creating some test data

Back to top

Creating some data for test purposes is a great way to learn the basics of Python.

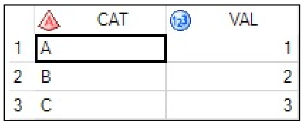
```
* Creating some test data - ALTERNATIVE 1;
proc sql;
        create table WORK.DT(
                CAT varchar(1),
                VAL numeric(1)
        );
insert into WORK.DF values("A", 1);
insert into WORK.DF values("B", 2);
insert into WORK.DF values("C", 3);
quit;


* Creating some test data - ALTERNATIVE 2;
data DT;
   input CAT $ VAL;
   datalines;
A 1
B 2
C 3
;
```

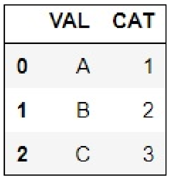
```
import pandas as pd

# Creating some test data
dt = pd.DataFrame({
    "CAT": ["A", "B", "C"],
    "VAL": [1, 2, 3]
})
```

Results



	CAT	VAL
1	A	1
2	B	2
3	C	3



	VAL	CAT
0	A	1
1	B	2
2	C	3

Reading from database

Back to top

Use one of the functions from `gj_common_py` to read from DB2. You need to have a `.netrc` file for the code to work.

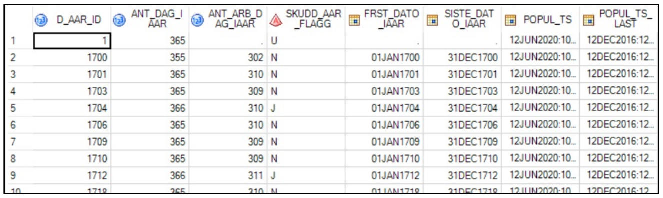
Note that we are sending DB2 queries directly, and they can be a lot more complicated than this example shows.

```
* Reading from database;
libname INFO db2 db=IVHP01 schema=G00V user=&user_id. pw=&pwd.   ac
data DBDATA;
        set INFO.D_AAR;
run;
```

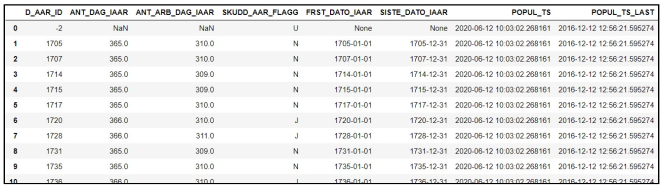
```
from gj_common_py.db_funcs import run_sql_with_ibmdb
import pandas as pd

# Reading from database
dbdata = pd.DataFrame(run_sql_with_ibmdb("IVHP01", "select *
```

Results



	D_AAR_ID	ANT_DAG_IAAR	ANT_ARB_DAG_IAAR	SKUDD_AAR_FLAGG	FRST_DATO_IAAR	SISTE_DATO_IAAR	POPUL_TS	POPUL_TS_LAST
1	1	365	.	U	.	.	12JUN2020:10...	12DEC2016:12...
2	1700	355	302	N	01JAN1700	31DEC1700	12JUN2020:10...	12DEC2016:12...
3	1701	365	310	N	01JAN1701	31DEC1701	12JUN2020:10...	12DEC2016:12...
4	1703	365	309	N	01JAN1703	31DEC1703	12JUN2020:10...	12DEC2016:12...
5	1704	366	310	J	01JAN1704	31DEC1704	12JUN2020:10...	12DEC2016:12...
6	1706	365	310	N	01JAN1706	31DEC1706	12JUN2020:10...	12DEC2016:12...
7	1709	365	309	N	01JAN1709	31DEC1709	12JUN2020:10...	12DEC2016:12...
8	1710	365	309	N	01JAN1710	31DEC1710	12JUN2020:10...	12DEC2016:12...
9	1712	366	311	J	01JAN1712	31DEC1712	12JUN2020:10...	12DEC2016:12...
10	1718	365	310	N	01JAN1718	31DEC1718	12JUN2020:10...	12DEC2016:12...



	D_AAR_ID	ANT_DAG_IAAR	ANT_ARB_DAG_IAAR	SKUDD_AAR_FLAGG	FRST_DATO_IAAR	SISTE_DATO_IAAR	POPUL_TS	POPUL_TS_LAST
0	-2	NaN	NaN	U	None	None	2020-06-12 10:03:02.268161	2016-12-12 12:56:21.595274
1	1705	365.0	310.0	N	1705-01-01	1705-12-31	2020-06-12 10:03:02.268161	2016-12-12 12:56:21.595274
2	1707	365.0	310.0	N	1707-01-01	1707-12-31	2020-06-12 10:03:02.268161	2016-12-12 12:56:21.595274
3	1714	365.0	309.0	N	1714-01-01	1714-12-31	2020-06-12 10:03:02.268161	2016-12-12 12:56:21.595274
4	1715	365.0	309.0	N	1715-01-01	1715-12-31	2020-06-12 10:03:02.268161	2016-12-12 12:56:21.595274
5	1717	365.0	310.0	N	1717-01-01	1717-12-31	2020-06-12 10:03:02.268161	2016-12-12 12:56:21.595274
6	1720	366.0	310.0	J	1720-01-01	1720-12-31	2020-06-12 10:03:02.268161	2016-12-12 12:56:21.595274
7	1728	366.0	311.0	J	1728-01-01	1728-12-31	2020-06-12 10:03:02.268161	2016-12-12 12:56:21.595274
8	1731	365.0	309.0	N	1731-01-01	1731-12-31	2020-06-12 10:03:02.268161	2016-12-12 12:56:21.595274
9	1735	365.0	310.0	N	1735-01-01	1735-12-31	2020-06-12 10:03:02.268161	2016-12-12 12:56:21.595274
10	1736	366.0	310.0	J	1736-01-01	1736-12-31	2020-06-12 10:03:02.268161	2016-12-12 12:56:21.595274

Reading/Writing to CSV

Back to top

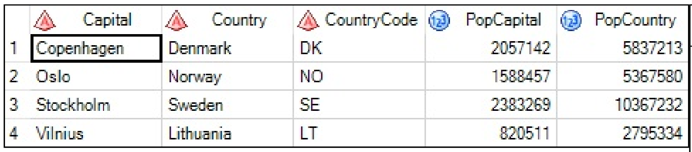
The pandas library has a function for reading CSV files: `read_csv()`.

```
* Reading from CSV;
proc import datafile='/mnt_g/GF_Prising/G020029/exampleData.csv'
out=TESTDATA dbms=csv replace;
getnames=yes;
run;
```

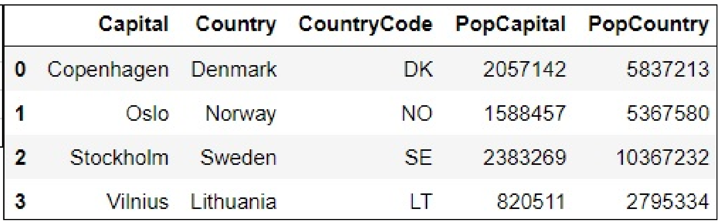
```
import pandas as pd

# Reading from CSV
testdata = pd.read_csv('/mnt_g/GF_Prising/G020029/exampleData
```

Results

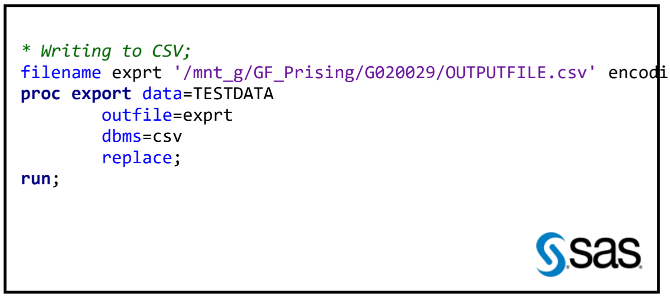


	Capital	Country	CountryCode	PopCapital	PopCountry
1	Copenhagen	Denmark	DK	2057142	5837213
2	Oslo	Norway	NO	1588457	5367580
3	Stockholm	Sweden	SE	2383269	10367232
4	Vilnius	Lithuania	LT	820511	2795334

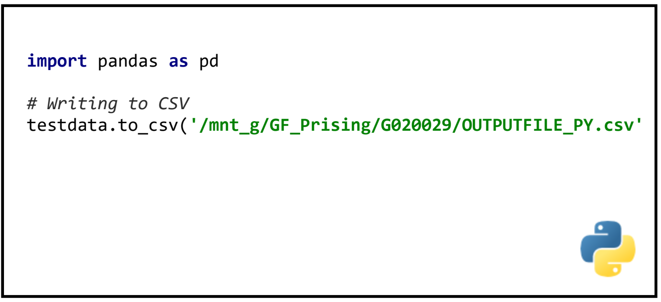


	Capital	Country	CountryCode	PopCapital	PopCountry
0	Copenhagen	Denmark	DK	2057142	5837213
1	Oslo	Norway	NO	1588457	5367580
2	Stockholm	Sweden	SE	2383269	10367232
3	Vilnius	Lithuania	LT	820511	2795334

Pandas includes the row numbers (or index) by default. They can be omitted by including the `index=False` argument to the function: `testdata.to_csv('out.csv', index=False)`.



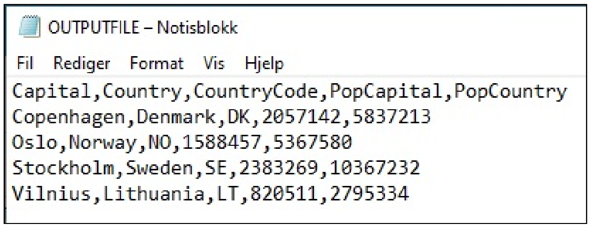
```
* Writing to CSV;
filename exprt '/mnt_g/GF_Prising/G020029/OUTPUTFILE.csv' encoding
proc export data=TESTDATA
        outfile=exprt
        dbms=csv
        replace;
run;
```



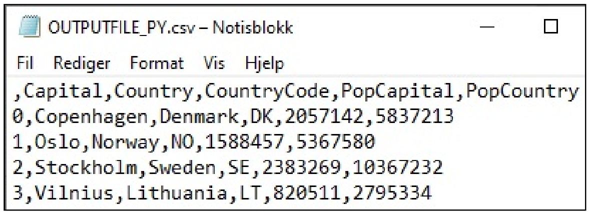
```
import pandas as pd

# Writing to CSV
testdata.to_csv('/mnt_g/GF_Prising/G020029/OUTPUTFILE_PY.csv'
```

Results



```
Capital,Country,CountryCode,PopCapital,PopCountry
Copenhagen,Denmark,DK,2057142,5837213
Oslo,Norway,NO,1588457,5367580
Stockholm,Sweden,SE,2383269,10367232
Vilnius,Lithuania,LT,820511,2795334
```

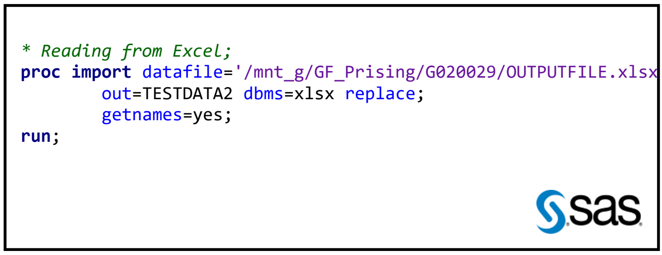


```
,Capital,Country,CountryCode,PopCapital,PopCountry
0,Copenhagen,Denmark,DK,2057142,5837213
1,Oslo,Norway,NO,1588457,5367580
2,Stockholm,Sweden,SE,2383269,10367232
3,Vilnius,Lithuania,LT,820511,2795334
```

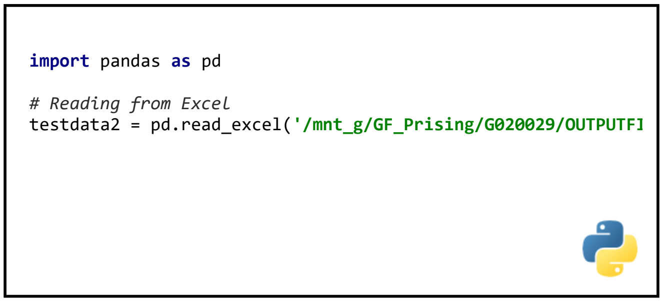
Reading/Writing to Excel

Back to top

The pandas library has a function for reading Excel files: `to_excel()`.



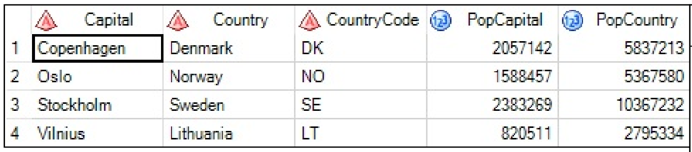
```
* Reading from Excel;
proc import datafile='/mnt_g/GF_Prising/G020029/OUTPUTFILE.xlsx'
        out=TESTDATA2 dbms=xlsx replace;
        getnames=yes;
run;
```



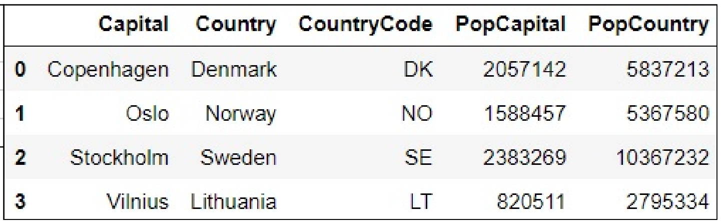
```
import pandas as pd

# Reading from Excel
testdata2 = pd.read_excel('/mnt_g/GF_Prising/G020029/OUTPUTFI
```

Results



	Capital	Country	CountryCode	PopCapital	PopCountry
1	Copenhagen	Denmark	DK	2057142	5837213
2	Oslo	Norway	NO	1588457	5367580
3	Stockholm	Sweden	SE	2383269	10367232
4	Vilnius	Lithuania	LT	820511	2795334



	Capital	Country	CountryCode	PopCapital	PopCountry
0	Copenhagen	Denmark	DK	2057142	5837213
1	Oslo	Norway	NO	1588457	5367580
2	Stockholm	Sweden	SE	2383269	10367232
3	Vilnius	Lithuania	LT	820511	2795334

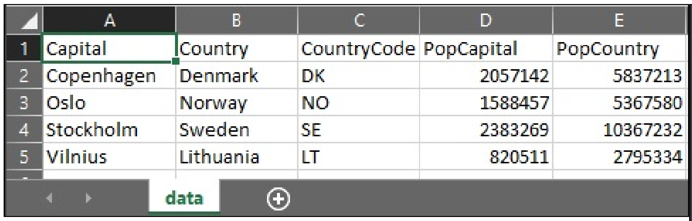
The pandas library has a function for writing Excel files: `write_excel()`. By default it includes the index which are omitted with `index=False`.

```
* Writing to Excel;
proc export data=TESTDATA
        outfile="/mnt_g/GF_Prising/G020029/OUTPUTFILE.xlsx"
        dbms=XLSX
        replace;
        sheet='data';
run;
```

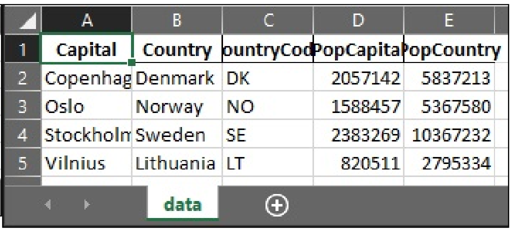
```
import pandas as pd

# Writing to Excel
testdata.to_excel('/mnt_g/GF_Prising/G020029/OUTPUTFILE_PY.xl
```

Results



	A	B	C	D	E
1	Capital	Country	CountryCode	PopCapital	PopCountry
2	Copenhagen	Denmark	DK	2057142	5837213
3	Oslo	Norway	NO	1588457	5367580
4	Stockholm	Sweden	SE	2383269	10367232
5	Vilnius	Lithuania	LT	820511	2795334



	A	B	C	D	E
1	Capital	Country	ountryCod	PopCapita	opCountry
2	Copenhag	Denmark	DK	2057142	5837213
3	Oslo	Norway	NO	1588457	5367580
4	Stockholm	Sweden	SE	2383269	10367232
5	Vilnius	Lithuania	LT	820511	2795334

Reading/Writing to sas7bdat

Back to top

SAS can store data directly in folders as 'sas7bdat' files. Python has many different alternatives, but here we use `pickle` files.

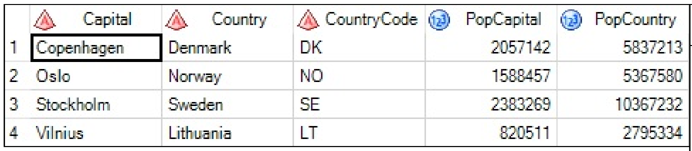
```
* Reading from sas7bdat file;
libname MYFOLDER '/mnt_g/GF_Prising/G020029/MYFOLDER/';

data TESTDATA3;
        set MYFOLDER.OUTTABLE;
run;
```

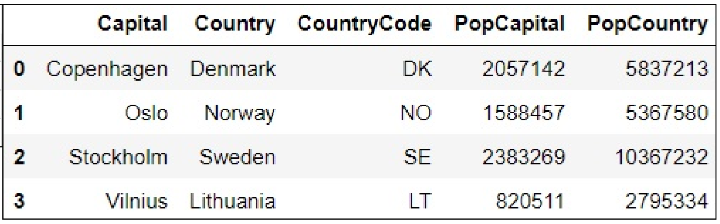
```
import pandas as pd

# Reading pickle file
testdata3 = pd.read_pickle("/mnt_g/GF_Prising/G020029/MYFOLDE
```

Results



	Capital	Country	CountryCode	PopCapital	PopCountry
1	Copenhagen	Denmark	DK	2057142	5837213
2	Oslo	Norway	NO	1588457	5367580
3	Stockholm	Sweden	SE	2383269	10367232
4	Vilnius	Lithuania	LT	820511	2795334



	Capital	Country	CountryCode	PopCapital	PopCountry
0	Copenhagen	Denmark	DK	2057142	5837213
1	Oslo	Norway	NO	1588457	5367580
2	Stockholm	Sweden	SE	2383269	10367232
3	Vilnius	Lithuania	LT	820511	2795334

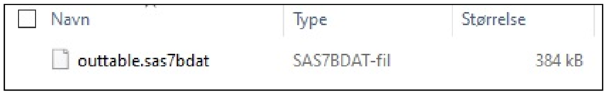
```
* Writing to sas7bdat file;
libname MYFOLDER '/mnt_g/GF_Prising/G020029/MYFOLDER/';

data MYFOLDER.OUTTABLE;
        set TESTDATA;
run;
```

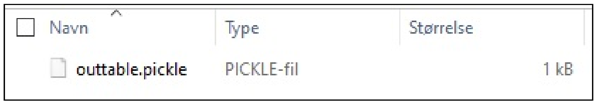
```
import pandas as pd

# Writing to pickle file
testdata.to_pickle("/mnt_g/GF_Prising/G020029/MYFOLDER/outtak
```

Results



Navn	Type	Størrelse
outtable.sas7bdat	SAS7BDAT-fil	384 kB



Navn	Type	Størrelse
outtable.pickle	PICKLE-fil	1 kB

Create new table

Back to top

In SAS it is common to create new tables using the DATA step. When using `pandas`, all operations are typically done directly in memory, or 'in-place'. This makes it more efficient and faster. To make a copy of the data, it has to be explicitly done with the `copy()` function.

If a table is copied with `newtab = testdata`, both will point to the same memory, and any changes made to one table will affect both!

```
* Create new table;
data NEWTAB;
        set TESTDATA;
run;
```

```
import pandas as pd

# Create new table
newtab = testdata.copy()
```

Getting first N rows

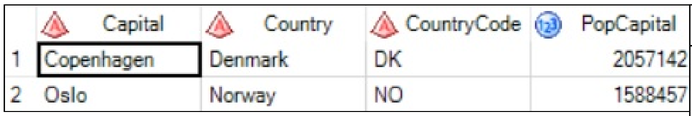
Back to top

An alternative is using `testdata.head(n=2)`. There is also `testdata.tail(n=2)` for getting the last rows in the table.

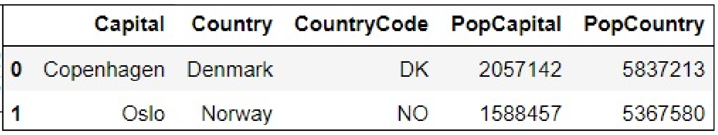
```
* Getting first N rows;
data SUBSET;
        set TESTDATA(obs=2);
run;
```

```
# Getting first N rows
subset = testdata[:2]
```

Results



	Capital	Country	CountryCode	PopCapital
1	Copenhagen	Denmark	DK	2057142
2	Oslo	Norway	NO	1588457

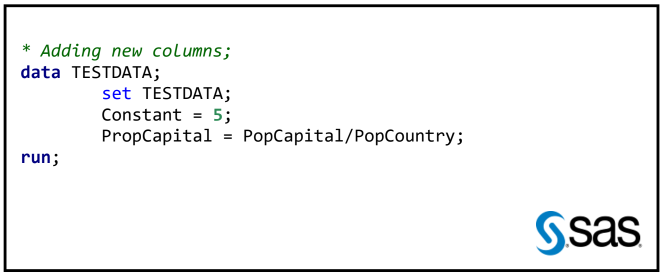


	Capital	Country	CountryCode	PopCapital	PopCountry
0	Copenhagen	Denmark	DK	2057142	5837213
1	Oslo	Norway	NO	1588457	5367580

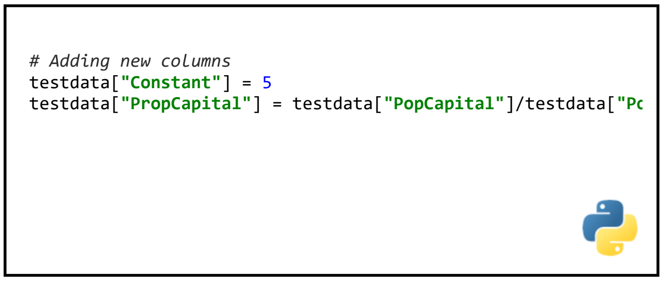
Adding and Removing columns

Back to top

Assigning new columns is typically done directly into the table.

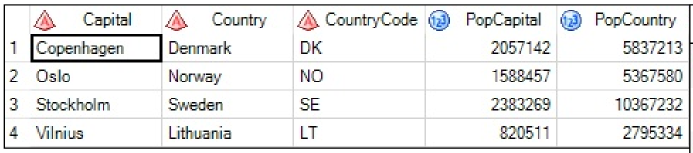


```
* Adding new columns;
data TESTDATA;
        set TESTDATA;
        Constant = 5;
        PopCapital = PopCapital/PopCountry;
run;
```

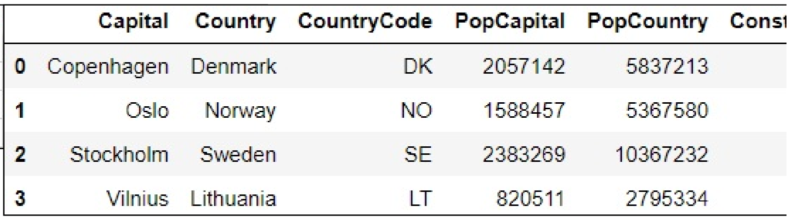


```
# Adding new columns
testdata["Constant"] = 5
testdata["PopCapital"] = testdata["PopCapital"]/testdata["Pc
```

Results

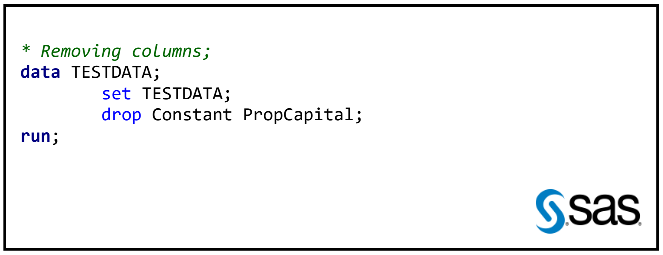


	Capital	Country	CountryCode	PopCapital	PopCountry
1	Copenhagen	Denmark	DK	2057142	5837213
2	Oslo	Norway	NO	1588457	5367580
3	Stockholm	Sweden	SE	2383269	10367232
4	Vilnius	Lithuania	LT	820511	2795334

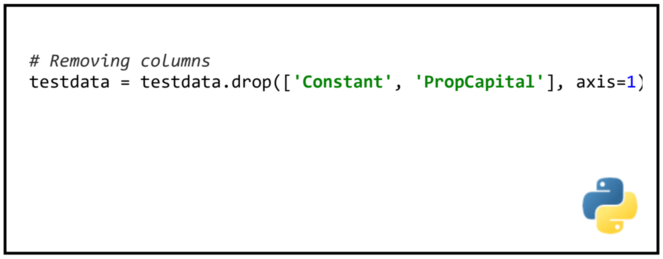


	Capital	Country	CountryCode	PopCapital	PopCountry	Const
0	Copenhagen	Denmark	DK	2057142	5837213	
1	Oslo	Norway	NO	1588457	5367580	
2	Stockholm	Sweden	SE	2383269	10367232	
3	Vilnius	Lithuania	LT	820511	2795334	

Note we assign the table back into `testdata`. We include `axis=1` to signify that we are specifying column names. (By default it targets the column index/number).

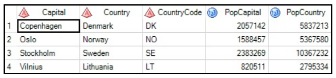


```
* Removing columns;
data TESTDATA;
        set TESTDATA;
        drop Constant PropCapital;
run;
```

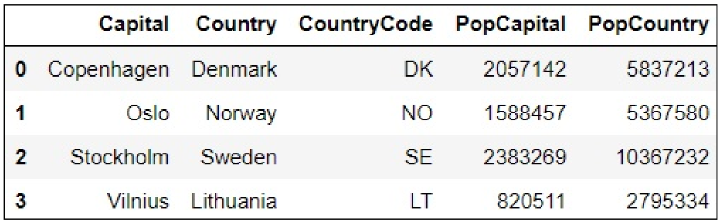


```
# Removing columns
testdata = testdata.drop(['Constant', 'PropCapital'], axis=1)
```

Results



	Capital	Country	CountryCode	PopCapital	PopCountry
1	Copenhagen	Denmark	DK	2057142	5837213
2	Oslo	Norway	NO	1588457	5367580
3	Stockholm	Sweden	SE	2383269	10367232
4	Vilnius	Lithuania	LT	820511	2795334

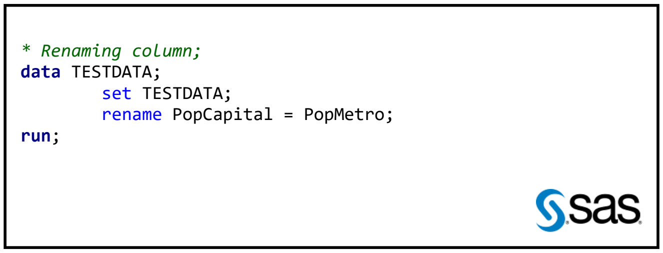


	Capital	Country	CountryCode	PopCapital	PopCountry
0	Copenhagen	Denmark	DK	2057142	5837213
1	Oslo	Norway	NO	1588457	5367580
2	Stockholm	Sweden	SE	2383269	10367232
3	Vilnius	Lithuania	LT	820511	2795334

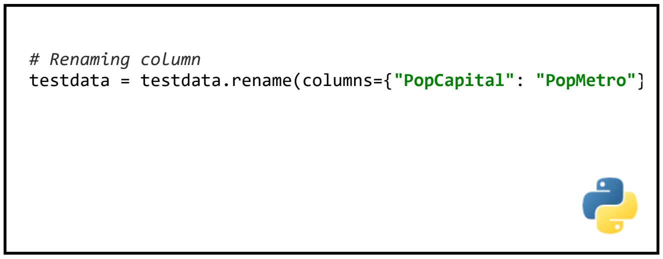
Renaming column

Back to top

Renaming is done with the `rename()` function. Several columns can be renamed by adding them to the `dictionary` (the part specified in `{}`) and separating them with commas.

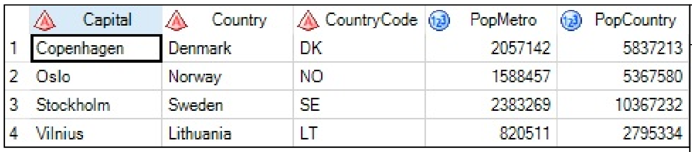


```
* Renaming column;
data TESTDATA;
        set TESTDATA;
        rename PopCapital = PopMetro;
run;
```

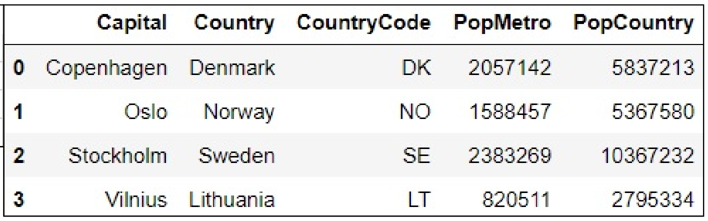


```
# Renaming column
testdata = testdata.rename(columns={"PopCapital": "PopMetro"}
```

Results



	Capital	Country	CountryCode	PopMetro	PopCountry
1	Copenhagen	Denmark	DK	2057142	5837213
2	Oslo	Norway	NO	1588457	5367580
3	Stockholm	Sweden	SE	2383269	10367232
4	Vilnius	Lithuania	LT	820511	2795334

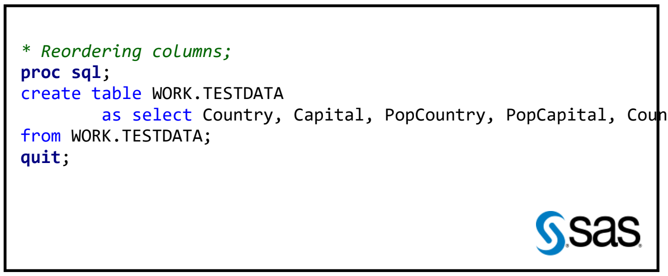


	Capital	Country	CountryCode	PopMetro	PopCountry
0	Copenhagen	Denmark	DK	2057142	5837213
1	Oslo	Norway	NO	1588457	5367580
2	Stockholm	Sweden	SE	2383269	10367232
3	Vilnius	Lithuania	LT	820511	2795334

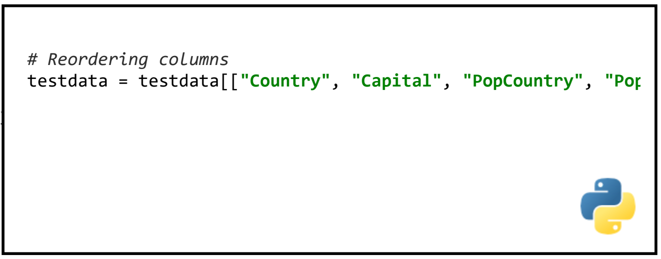
Reordering columns

Back to top

We pass a list of the columns in the desired order.

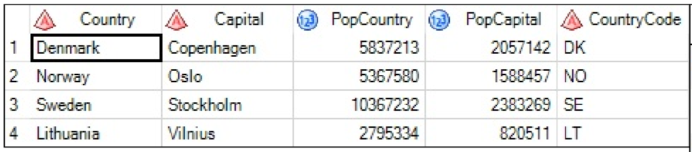


```
* Reordering columns;
proc sql;
create table WORK.TESTDATA
        as select Country, Capital, PopCountry, PopCapital, Countr
from WORK.TESTDATA;
quit;
```

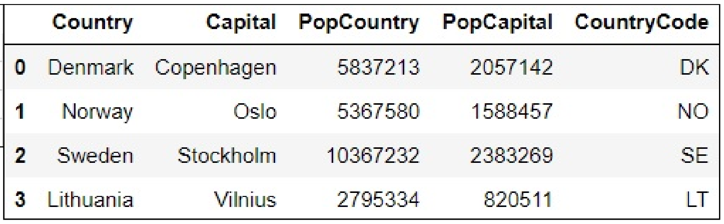


```
# Reordering columns
testdata = testdata[["Country", "Capital", "PopCountry", "Pop
```

Results



	Country	Capital	PopCountry	PopCapital	CountryCode
1	Denmark	Copenhagen	5837213	2057142	DK
2	Norway	Oslo	5367580	1588457	NO
3	Sweden	Stockholm	10367232	2383269	SE
4	Lithuania	Vilnius	2795334	820511	LT



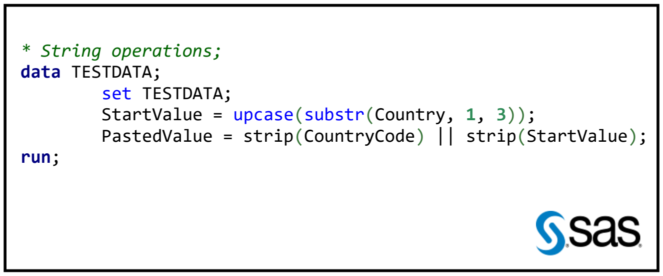
	Country	Capital	PopCountry	PopCapital	CountryCode
0	Denmark	Copenhagen	5837213	2057142	DK
1	Norway	Oslo	5367580	1588457	NO
2	Sweden	Stockholm	10367232	2383269	SE
3	Lithuania	Vilnius	2795334	820511	LT

String operations

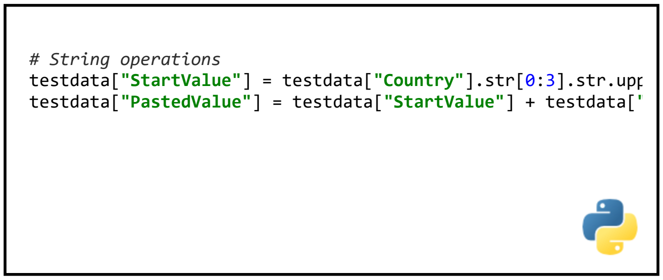
Back to top

We apply string operations by chaining together functions. Pasting string variables is done with a simple +.

Note the difference in selecting the substring. Python substring starts on 0 and uses the "up to but not including" to set the upper limit.

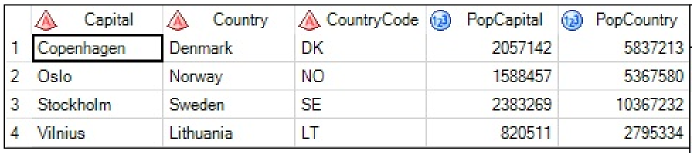


```
* String operations;
data TESTDATA;
        set TESTDATA;
        StartValue = upcase(substr(Country, 1, 3));
        PastedValue = strip(CountryCode) || strip(StartValue);
run;
```

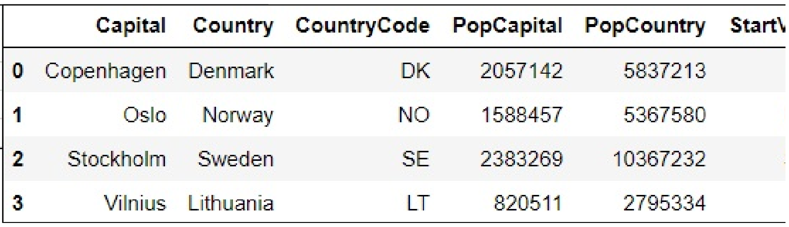


```
# String operations
testdata["StartValue"] = testdata["Country"].str[0:3].str.upp
testdata["PastedValue"] = testdata["StartValue"] + testdata['
```

Results



	Capital	Country	CountryCode	PopCapital	PopCountry
1	Copenhagen	Denmark	DK	2057142	5837213
2	Oslo	Norway	NO	1588457	5367580
3	Stockholm	Sweden	SE	2383269	10367232
4	Vilnius	Lithuania	LT	820511	2795334



	Capital	Country	CountryCode	PopCapital	PopCountry	StartV
0	Copenhagen	Denmark	DK	2057142	5837213	
1	Oslo	Norway	NO	1588457	5367580	
2	Stockholm	Sweden	SE	2383269	10367232	
3	Vilnius	Lithuania	LT	820511	2795334	

Where statement

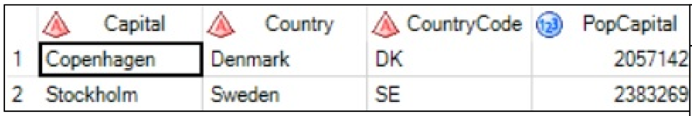
Back to top

In SAS you run through the whole table in a DATA-step and apply the filter. The `pandas` library has a lot of different methods for subsetting (or 'slicing') the dataframe. In this example we are using `loc`, but there are others such as `iloc` or just using `[]`. They are typically targeted directly at the rows.

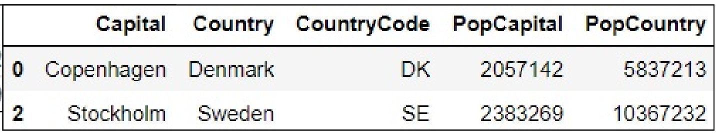
```
* Where statement;
data SUBSET2;
        set TESTDATA;
        where PopCapital > 2000000;
run;
```

```
# Where statement
subset2 = testdata.loc[testdata['PopCapital'] > 2000000]
```

Results



	Capital	Country	CountryCode	PopCapital
1	Copenhagen	Denmark	DK	2057142
2	Stockholm	Sweden	SE	2383269



	Capital	Country	CountryCode	PopCapital	PopCountry
0	Copenhagen	Denmark	DK	2057142	5837213
2	Stockholm	Sweden	SE	2383269	10367232

If/else condition

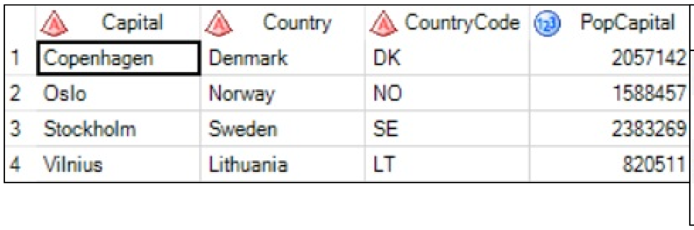
Back to top

The easiest way to use if/else conditions in Python is in a custom function. An alternative way is by using row operations as demonstrated here.

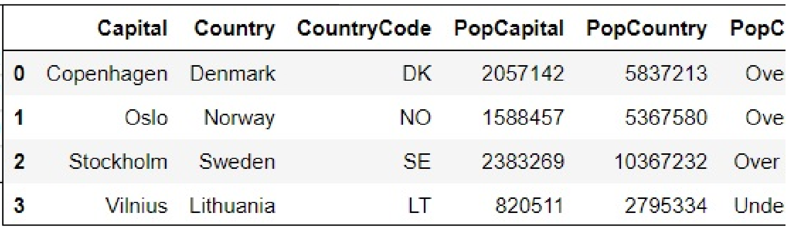
```
* If/else condition;
data TESTDATA;
        set TESTDATA;
        length PopClass $ 10;
        if PopCountry > 10000000 then PopClass = 'Over 10M';
        else if PopCountry > 5000000 then PopClass = 'Over 5M';
        else PopClass = 'Under 5M';
run;
```

```
# If/else condition
testdata.loc[testdata['PopCountry'] < 5000000, 'PopClass'] =
testdata.loc[testdata['PopCountry'] > 5000000, 'PopClass'] =
testdata.loc[testdata['PopCountry'] > 10000000, 'PopClass'] =
```

Results



	Capital	Country	CountryCode	PopCapital
1	Copenhagen	Denmark	DK	2057142
2	Oslo	Norway	NO	1588457
3	Stockholm	Sweden	SE	2383269
4	Vilnius	Lithuania	LT	820511



	Capital	Country	CountryCode	PopCapital	PopCountry	PopC
0	Copenhagen	Denmark	DK	2057142	5837213	Ove
1	Oslo	Norway	NO	1588457	5367580	Ove
2	Stockholm	Sweden	SE	2383269	10367232	Over
3	Vilnius	Lithuania	LT	820511	2795334	Unde

Merge/join tables

Back to top

Showing SAS-examples with both PROC SQL and MERGE.

Input

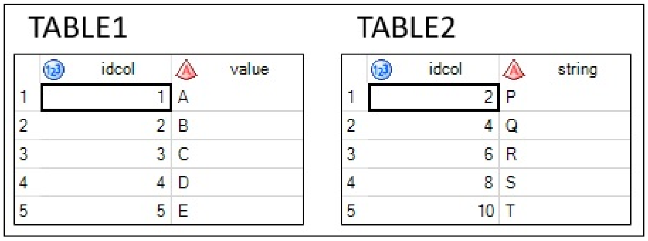


TABLE1

	idcol	value
1	1	A
2	2	B
3	3	C
4	4	D
5	5	E

TABLE2

	idcol	string
1	2	P
2	4	Q
3	6	R
4	8	S
5	10	T

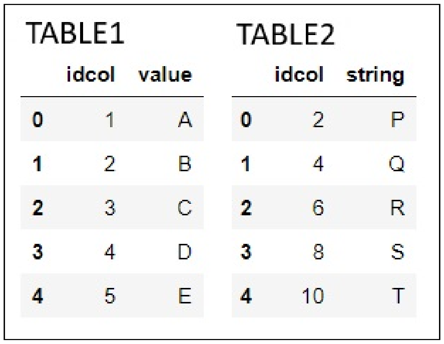


TABLE1

	idcol	value
0	1	A
1	2	B
2	3	C
3	4	D
4	5	E

TABLE2

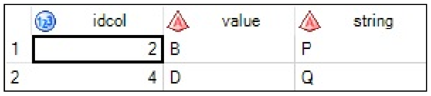
	idcol	string
0	2	P
1	4	Q
2	6	R
3	8	S
4	10	T

Joining tables is done with the `merge()` function. The default join type is `inner`. If there is more than one key column, they are passed as a list:
`on=["key1", "key2", "key3"]`.

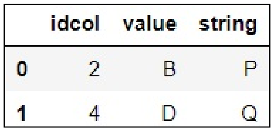
```
* Inner join;
proc sql;
create table JOINED as
select t1.idcol, t1.value, t2.string
from WORK.TABLE1 as t1
inner join WORK.TABLE2 as t2
        on t1.idcol = t2.idcol;
quit;
```

```
# Inner join
joined = pd.merge(table1, table2, on="idcol")
```

Results



	idcol	value	string
1	2	B	P
2	4	D	Q



	idcol	value	string
0	2	B	P
1	4	D	Q

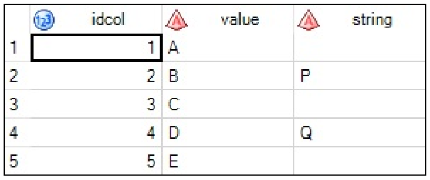
Left join is very similar to inner join, it just has to be specified in `how="left"`. Here you can also specify `"outer"` or `"right"`.

Also worthy of note, the `merge()` function does NOT require the input tables to be sorted.

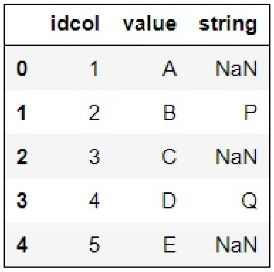
```
* Left join;
data WORK.LEFTJOIN;
        merge WORK.TABLE1(in=a) WORK.TABLE2(in=b);
        by idcol;
        if a;
run;
```

```
# Left join
leftjoin = pd.merge(table1, table2, how="left", on="idcol")
```

Results



	idcol	value	string
1	1	A	
2	2	B	P
3	3	C	
4	4	D	Q
5	5	E	



	idcol	value	string
0	1	A	NaN
1	2	B	P
2	3	C	NaN
3	4	D	Q
4	5	E	NaN

Append tables

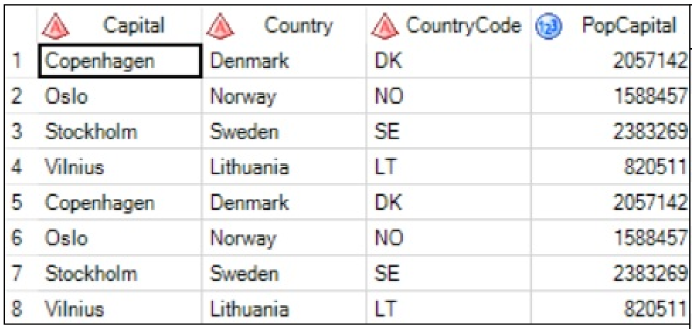
Back to top

Appending tables is done with `append()` or with `concat()`.

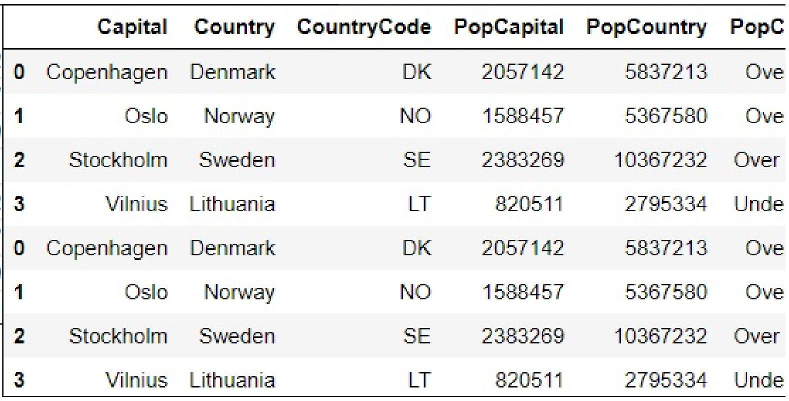
```
* Append tables;
data LARGETABLE;
        set TESTDATA TESTDATA;
run;
```

```
# Append tables
largetable = testdata.append(testdata)
```

Results



	Capital	Country	CountryCode	PopCapital
1	Copenhagen	Denmark	DK	2057142
2	Oslo	Norway	NO	1588457
3	Stockholm	Sweden	SE	2383269
4	Vilnius	Lithuania	LT	820511
5	Copenhagen	Denmark	DK	2057142
6	Oslo	Norway	NO	1588457
7	Stockholm	Sweden	SE	2383269
8	Vilnius	Lithuania	LT	820511



	Capital	Country	CountryCode	PopCapital	PopCountry	PopC
0	Copenhagen	Denmark	DK	2057142	5837213	Ove
1	Oslo	Norway	NO	1588457	5367580	Ove
2	Stockholm	Sweden	SE	2383269	10367232	Over
3	Vilnius	Lithuania	LT	820511	2795334	Unde
0	Copenhagen	Denmark	DK	2057142	5837213	Ove
1	Oslo	Norway	NO	1588457	5367580	Ove
2	Stockholm	Sweden	SE	2383269	10367232	Over
3	Vilnius	Lithuania	LT	820511	2795334	Unde

Aggregating a table

Back to top

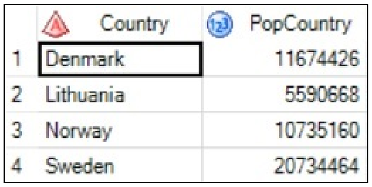
The 'class' variable is specified in the `groupby()` function. The function `agg("sum")` will sum up all numeric columns. To specify what column we want to sum (the 'var' in SAS), we pass a dictionary (parts in `{}`) specifying the column and operation.

We also pass `set_index=False` to get a pandas dataframe back. If not specified, we would get a slightly different kind of table.

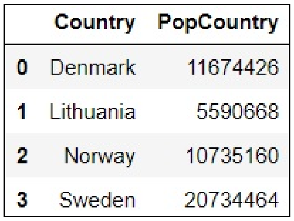
```
* Aggregating a table;
proc summary data=LARGETABLE missing nway noprint;
        class Country;
        var PopCountry;
        output out=DOUBLEPOP(drop=_:) sum=;
run;
```

```
# Aggregating a table
doublepop = largetable.groupby('Country', as_index=False).agg
```

Results



	Country	PopCountry
1	Denmark	11674426
2	Lithuania	5590668
3	Norway	10735160
4	Sweden	20734464



	Country	PopCountry
0	Denmark	11674426
1	Lithuania	5590668
2	Norway	10735160
3	Sweden	20734464

Sorting a table

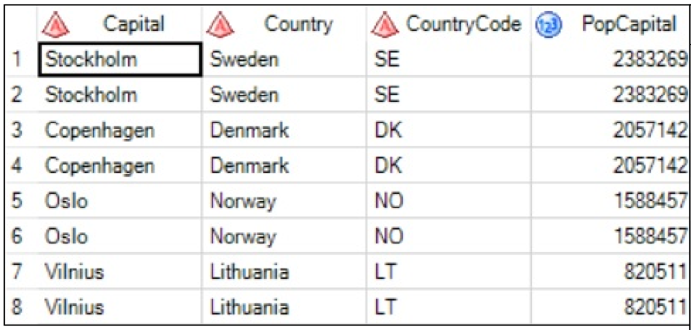
Back to top

Sorting is done with `sort_values()`. By default it sorts ascending which can be switched to descending with `ascending=False`.

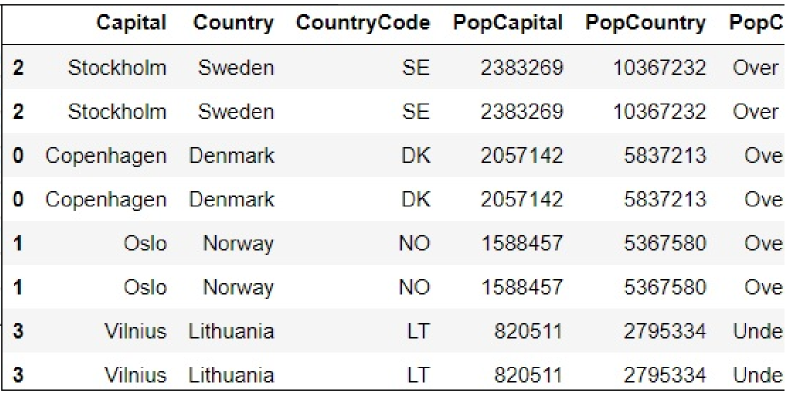
```
* Sorting a table - descending;
proc sort data=LARGETABLE out=SORT1;
        by descending PopCountry;
run;
```

```
# Sorting a table - descending
sort1 = largetable.sort_values(by="PopCountry", ascending=Fal
```

Results



	Capital	Country	CountryCode	PopCapital
1	Stockholm	Sweden	SE	2383269
2	Stockholm	Sweden	SE	2383269
3	Copenhagen	Denmark	DK	2057142
4	Copenhagen	Denmark	DK	2057142
5	Oslo	Norway	NO	1588457
6	Oslo	Norway	NO	1588457
7	Vilnius	Lithuania	LT	820511
8	Vilnius	Lithuania	LT	820511



	Capital	Country	CountryCode	PopCapital	PopCountry	PopC
2	Stockholm	Sweden	SE	2383269	10367232	Over
2	Stockholm	Sweden	SE	2383269	10367232	Over
0	Copenhagen	Denmark	DK	2057142	5837213	Ove
0	Copenhagen	Denmark	DK	2057142	5837213	Ove
1	Oslo	Norway	NO	1588457	5367580	Ove
1	Oslo	Norway	NO	1588457	5367580	Ove
3	Vilnius	Lithuania	LT	820511	2795334	Unde
3	Vilnius	Lithuania	LT	820511	2795334	Unde

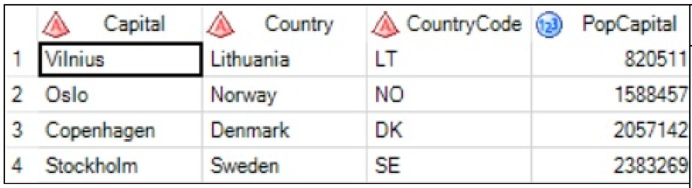
Sorting and removing duplicates are two different operations in pandas. We can do both at the same time by chaining functions `sort_values()` and `drop_duplicates()`. We have to specify "PopCountry" in both.

Since ascending sort is default, it does not need to be specified, just like in SAS.

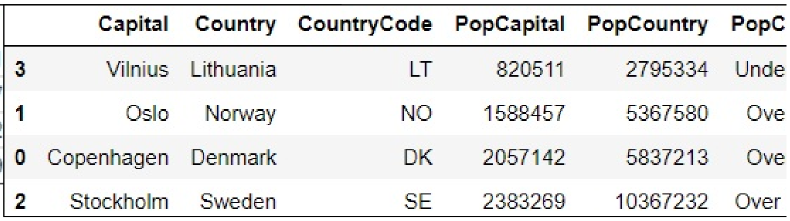
```
* Sorting a table - ascending and unique;
proc sort data=LARGETABLE out=SORT2 nodup;
        by PopCountry;
run;
```

```
# Sorting a table - ascending and unique
sort2 = largetable.sort_values(by="PopCountry").drop_duplicat
```

Results



	Capital	Country	CountryCode	PopCapital
1	Vilnius	Lithuania	LT	820511
2	Oslo	Norway	NO	1588457
3	Copenhagen	Denmark	DK	2057142
4	Stockholm	Sweden	SE	2383269



	Capital	Country	CountryCode	PopCapital	PopCountry	PopC
3	Vilnius	Lithuania	LT	820511	2795334	Unde
1	Oslo	Norway	NO	1588457	5367580	Ove
0	Copenhagen	Denmark	DK	2057142	5837213	Ove
2	Stockholm	Sweden	SE	2383269	10367232	Over

Custom function

Back to top

It's not so common to define your own functions in SAS, but it is used a lot in Python.

Note the different arguments used in the `round()` function. In Python you specify how many decimals you want.

PS. The function in this example is completely nonsensical!

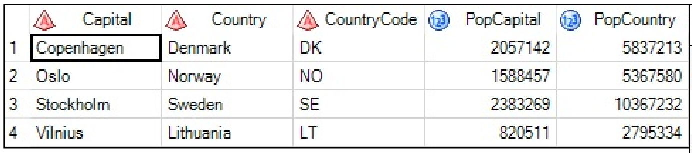
```
* Custom function;
proc fcmp outlib=work.funcs.myfuncs;
function SomeCalculation(totalPop, cityPop);
        return ((3.14*cityPop**2)/(200*totalPop - 128000));
endsub;
run;

*Running function;
options cmplib=work.funcs;
data COMPVALUE;
        set TESTDATA;
        CalcValue = SomeCalculation(PopCountry, PopCapital);
        CalcValue = round(CalcValue, .01);
run;
```

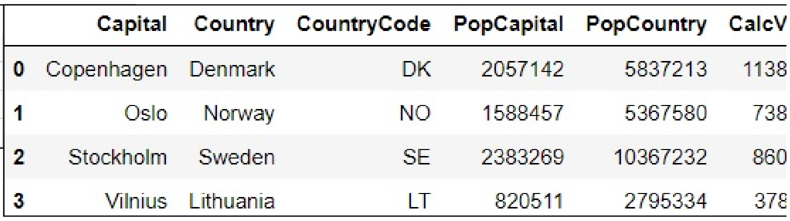
```
# Custom function
def SomeCalculation(totalPop, cityPop):
    return (3.14*cityPop**2)/(200*totalPop - 128000)

# Running function
testdata["CalcValue"] = SomeCalculation(testdata["PopCountry"
```

Results



	Capital	Country	CountryCode	PopCapital	PopCountry
1	Copenhagen	Denmark	DK	2057142	5837213
2	Oslo	Norway	NO	1588457	5367580
3	Stockholm	Sweden	SE	2383269	10367232
4	Vilnius	Lithuania	LT	820511	2795334



	Capital	Country	CountryCode	PopCapital	PopCountry	CalcV
0	Copenhagen	Denmark	DK	2057142	5837213	1138
1	Oslo	Norway	NO	1588457	5367580	738
2	Stockholm	Sweden	SE	2383269	10367232	860
3	Vilnius	Lithuania	LT	820511	2795334	378

Proc freq Example

Back to top

The PROCs in SAS don't always have a corresponding function in Python or pandas, but it is usually straight forward to recreate the logic.

```
* Calculate frequencies;
proc freq data=LARGETABLE;
```

```
        output out=RESULT;
        tables Country / nocum;
run;
```

```
# Calculate frequencies
result = largetable["Country"].value_counts().reset_index()
result.columns = ["Country","Frequency"]
result["Percent"] = 100*result["Frequency"]/result["Frequency
```

Results

The FREQ Procedure

Country	Frequency	Percent
Denmark	2	25.00
Lithuania	2	25.00
Norway	2	25.00
Sweden	2	25.00

	Country	Frequency	Percent
0	Denmark	2	25.0
1	Norway	2	25.0
2	Lithuania	2	25.0
3	Sweden	2	25.0

If there are any PROCs that you use a lot, it is possible to write a function for it in Python.

This function only supports one 'tables' or 'class' variable, but can easily be expanded

```
* Calculate frequencies 2;
proc freq data=LARGETABLE;
        output out=RESULT;
        tables Country;
run;
```

```
def PROCFREQ(data, variable):
    nrows = data.shape[0]
    output = data[variable].value_counts().reset_index()
    output.columns = [variable, "Frequency"]
    output["Percent"] = 100*output["Frequency"]/nrows
    output["CumFrequency"] = output["Frequency"].cumsum()
    output["CumPercent"] = output["Percent"].cumsum()
    return output

# Calculate frequencies 2
result = PROCFREQ(largetable, "Country")
```

Results

The FREQ Procedure

Country	Frequency	Percent	Cumulative Frequency	Cumulative Percent
Denmark	2	25.00	2	25.00
Lithuania	2	25.00	4	50.00
Norway	2	25.00	6	75.00
Sweden	2	25.00	8	100.00

	Country	Frequency	Percent	CumFrequency	CumPercent
0	Denmark	2	25.0	2	25.0
1	Norway	2	25.0	4	50.0
2	Lithuania	2	25.0	6	75.0
3	Sweden	2	25.0	8	100.0

Deleting a table

Back to top

There isn't really any delete functionality in Python. Memory used for a huge object can be freed by simply assigning the variable to `None`.

```
* Deleting a table - ALTERNATIVE 1;
proc sql;
drop table WORK.TESTDATA;
quit;
```

```
# Deleting a table
testdata = None
```

```
* Deleting a table - ALTERNATIVE 2;
proc datasets library = WORK nolist;
  delete TESTDATA;
run;

* Deleting a table - ALTERNATIVE 3;
proc delete data = WORK.TESTDATA;
run;
```

That's it! That's all!! :-)