

What is Siuba?

The `siuba` python library brings the power of R's `dplyr` and the tidyverse to Python. Gain access to functions like:

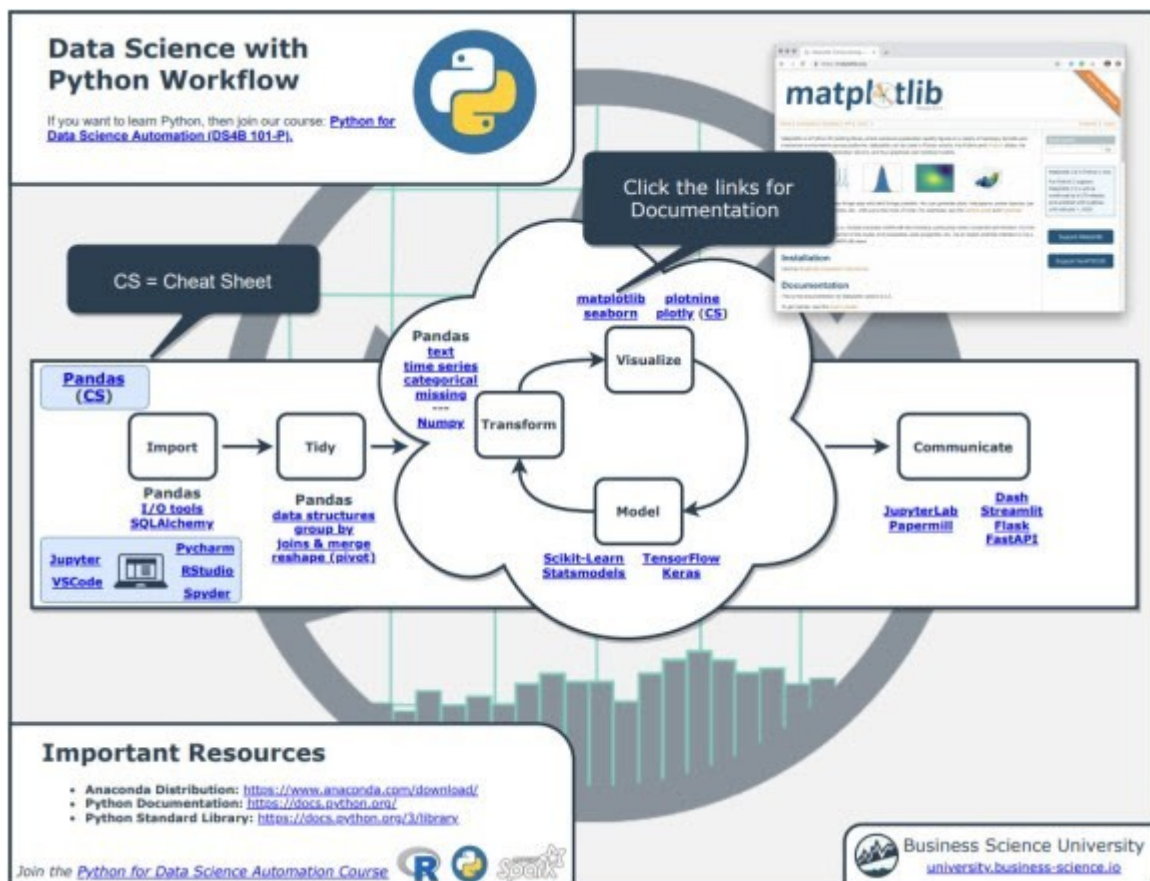
- `select()` – keep certain columns of data.
- `filter()` – keep certain rows of data.
- `mutate()` – create or modify an existing column of data.
- `summarize()` – reduce one or more columns down to a single number.
- `arrange()` – reorder the rows of data.

Before we get started, get the Python Cheat Sheet

`Siuba` is great for data wrangling in Python if you are coming from an R background. But, you might want to explore documentation for the entire Python Ecosystem. I'll use the [Ultimate Python Cheat Sheet](#).

Ultimate Python Cheat Sheet:

First, [Download the Ultimate Python Cheat Sheet](#). This gives you access to the entire Python Ecosystem at your fingertips via hyperlinked documentation and cheat sheets.



Navigate to “Coming From R?” Section

Next, go to the section, “Coming from R?”. You can quickly get to the Siuba Documentation.

Data Science with Python

Special Topics

- Time Series Forecasting**
 - [sktime](#) - Scikit-Learn Extension for Time Series
 - [statsmodels](#) - Time Series Analysis
 - [GluonTS](#) - MXNet/Gluon Deep Learning for Time Series
- Time Series Features**
 - [TSFresh](#) - Time Series Feature Engineering
 - [sklearn](#) - Time Series Features
 - [Pandas](#) - Time Series
 - [Arrow](#) - Human-Friendly Time
- EDA**
 - [pandas-profiling](#) - Generate exploratory report
 - [lux](#) - Intelligent visual explorations
- Web**
 - [beautifulsoup](#) - Extract data from HTML
 - [requests-html](#) - HTML Parsing
 - [scrapy](#) - Web crawling
- MS Office & PDF**
 - [xlsxwriter](#) - Create Excel Workbooks
 - [openpyxl](#) - Read/Write Excel
 - [xlwings](#) - Call python from Excel
 - [python-docx](#) - Word Documents
 - [python-pptx](#) - PowerPoint Documents
 - [pdfminer](#) - Text extraction from PDF
 - [textextract](#) - Extract text from any document
 - [PyPDF2](#) - Create PDF documents
 - [gsheets](#) - Google Sheets
- Text Analysis & NLP**
 - [NLTK](#) - Text Tokenization & Modeling
 - [spaCy](#) - NLP using Cython for Speed
 - [fuzzywuzzy](#) - Fuzzy String Matching
- Recommendation Systems**
 - [Annoy](#) - Approximate Nearest Neighbors
 - [LightFM](#) - Popular recommendation algo's
- Apps & APIs**
 - [FastAPI](#) - Web framework for building APIs in Python
 - [Flask](#) - Web Development
 - [Dash & Streamlit](#) - DS Web Frameworks
- MLOps**
 - [MLFlow](#) - Machine Learning Lifecycle Tracking
 - [Kubeflow](#) - Scalable AI/ML Jobs for Cloud
 - [Docker](#) - Containerization
 - [Kubernetes](#) - Orchestration
 - [Airflow](#) - Workflow Automation
 - [Jenkins](#) - CI/CD
- Machine Learning**
 - [Scikit-Learn](#) - ML in Python
 - [H2O](#) - Scalable & AutoML
 - [TPOT](#) - TPOT Automated ML Tool
 - [PyCaret](#) - PyCaret Low Code ML
 - [Dask ML](#) - Scalable ML with Dask
 - [ML Packages](#) - [XGBoost](#) - [LightGBM](#) - [CatBoost](#)
- Feature Engineering**
 - [sklearn-pandas](#) - Sklearn Extension for Pandas
 - [Featuretools](#) - Automated Feature Engineering
 - [category_encoders](#) - Categorical Encoding
 - [imbalanced-learn](#) - Resampling for Imbalanced
 - [fancyimpute](#) - Extended Imputation strategies
- Deep Learning**
 - [TensorFlow & Keras](#)
 - [PyTorch](#)
 - [MXNet](#) - [Gluon](#) & [GluonTS](#)
 - [OpenAI Gym](#) - Reinforcement Learning
- Image & Comp Vision**
 - [OpenCV](#) - Open Source Computer Vision
 - [Scikit Image](#) - Image Processing
 - [Pillow](#) - Python Imaging Library
- Speed & Scale**
 - [dask](#) - C++ Speed Up
 - [Dask \(CS\)](#) - Parallel Pandas & Scikit Learn
 - [RAPIDS \(CS\)](#) - GPU Accelerated Pandas
 - [PySpark](#) - Spark Clusters
 - [DaskML](#) - PySpark Extension for Humans
- Coming from R?**
 - [R-to-Pandas Comparison](#)
 - [siuba & rpydata](#) - dplyr/tidyr ports
 - [data.table](#) - data.table port
 - [ggplot2](#) - ggplot2 port

Join the [Python for Data Science Automation Course](#)

Business Science University
[university.business-science.io](#)

Explore Siuba

You have access to the Siuba Documentation at your fingertips.

Siuba — siuba version v0.0.24-6

siuba.readthedocs.io/en/latest/

siuba

Navigation

- Quickstart
- Key features
- Data Analysis guide
- Programming guide
- Examples
- Developer docs
- Reference
- User API
- Extras

Quick search

Go

Siuba

Siuba is a library for quick, scrappy data analysis in Python. It is a port of [dplyr](#), [tidyr](#), and other R Tidyverse libraries.

Learning

Documentation

Quickstart

Install siuba and use a basic data analysis

For me on GitHub

Onto the tutorial.

How Siuba Works

From the *Siuba Documentation*, you can see that there are “verbs”, “siu expressions”, and the pipe (>>). We’ll test these out in our tutorial.

concept	example	meaning
verb	<code>group_by(...)</code>	a function that operates on a table, like a DataFrame or SQL table
siu expression	<code>_.hp.mean()</code>	an expression created with <code>siuba._</code> , that represents actions you want to perform
pipe	<code>mtcars >> group_by(...)</code>	a syntax that allows you to chain verbs with the <code>>></code> operator

Taking Siuba for a Test Spin

Let’s try out `siuba`’s data wrangling capabilities.

Step 1: Load Libraries and Data

First, let’s load the libraries and data. From the libraries, we’ll import `numpy` and `pandas` along with:

- `_`: Needed to create “siu expressions”
- `dplyr.verbs`: We’ll import `group_by()`, `summarize()`, and `mutate()`

```

7  # LIBRARIES ----
8  import numpy as np
9  import pandas as pd
10
11  from siuba import _
12  from siuba.dply.verbs import group_by, mutate, select, summarize, ungroup
13
14  # DATASET ----
15
16  mpg_df = pd.read_csv("https://raw.githubusercontent.com/mwaskom/seaborn-data/master/mpg.csv")
17  mpg_df
18

```

[Get the code.](#)

We’ll also load the `mpg_df` data set.

[22] mpg_df

	mpg	cylinders	displacement	horsepower	weight	acceleration	model_year	origin	name
0	18.0	8	307.0	130.0	3504	12.0	70	usa	chevrolet chevelle malibu
1	15.0	8	350.0	165.0	3693	11.5	70	usa	buick skylark 320
2	18.0	8	318.0	150.0	3436	11.0	70	usa	plymouth satellite
3	16.0	8	304.0	150.0	3433	12.0	70	usa	amc rebel sst
4	17.0	8	302.0	140.0	3449	10.5	70	usa	ford torino
...
393	27.0	4	140.0	86.0	2790	15.6	82	usa	ford mustang gl
394	44.0	4	97.0	52.0	2130	24.6	82	europa	vw pickup
395	32.0	4	135.0	84.0	2295	11.6	82	usa	dodge rampage
396	28.0	4	120.0	79.0	2625	18.6	82	usa	ford ranger
397	31.0	4	119.0	82.0	2720	19.4	82	usa	chevy s-10

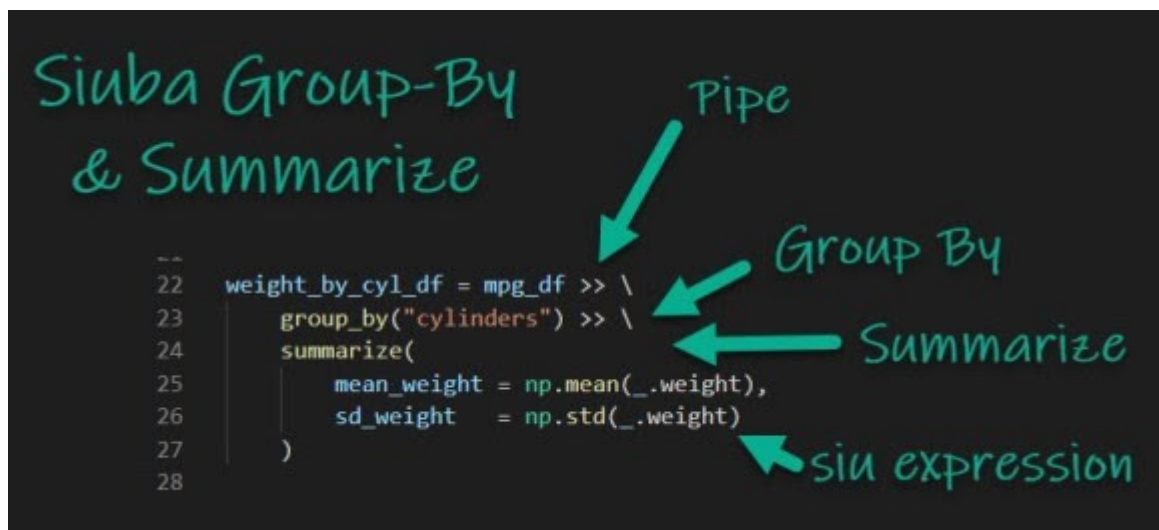
398 rows x 9 columns

Step 2: Group By and Summarize

Goal: Mean and Standard Deviation of weight by engine size

We'll perform a basic operation: `group_by()` and `summarize()` to get the mean and standard deviation of vehicle weight by engine size.

Group-By Summarize Code



[Get the code.](#)

Let's explain each operation in detail so you understand what's going on.

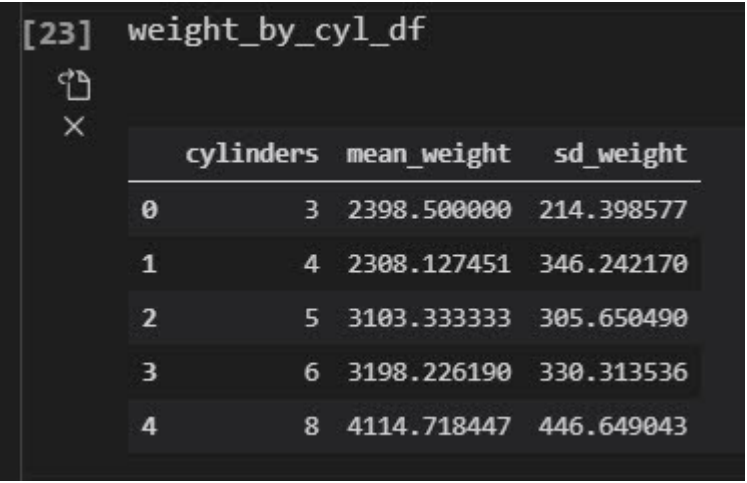
- **Pipe (`>>`):** This sends the output of a previous call (a Pandas DataFrame) as the first input of the next function.
- **Group By (`group_by()`):** This tells python that we want to perform an operation group-wise. We specify by the "cylinder" column.
- **Summarize (`summarize()`):** This tells python that, for each group, we want to summarize the weight to return the mean and standard deviation.
 - Inside the Summarize, we use "**siu expressions**", which allow us to reference columns (e.g. `_.weight`) while we perform the summarization.

- We take advantage of `numpy` for our summarization, using `np.mean()` and `np.std()` to calculate mean and standard deviation.

Code Output

This returns the vehicle weight summarized by the engine size (number of cylinders).

```
[23] weight_by_cyl_df
```



	cylinders	mean_weight	sd_weight
0	3	2398.500000	214.398577
1	4	2308.127451	346.242170
2	5	3103.333333	305.650490
3	6	3198.226190	330.313536
4	8	4114.718447	446.649043

Step 3: More Advanced Example (Group By and Mutate)

Goal: De-mean the mpg by average of each cylinder



We'll go through a more complex example using a `group_by()` and `mutate()`. See if you can figure out what's going on here. Tip – Try reading it like a paragraph in a text.

```
34 mpg_demeaned_by_cyl_df = mpg_df >> \  
35   select(._name, _.cylinders, _.mpg) >> \  
36   group_by(_.cylinders) >> \  
37   mutate(  
38     mean_mpg = np.mean(_.mpg)  
39   ) >> \  
40   ungroup() >> \  
41   mutate(  
42     mpg_demeaned_by_cyl = _.mpg - _.mean_mpg  
43   )  
44
```

[Get the code.](#)

This returns the vehicle fuel efficiency (mpg) de-meaned (removing the average by cylinder class). So now we can compare how the fuel efficiency of each vehicle model compares to the average fuel efficiency within groups of the same engine size.

[24] mpg_demeaned_by_cyl_df

	name	cylinders	mpg	mean_mpg	mpg_demeaned_by_cyl
0	chevrolet chevelle malibu	8	18.0	14.963107	3.036893
1	buick skylark 320	8	15.0	14.963107	0.036893
2	plymouth satellite	8	18.0	14.963107	3.036893
3	amc rebel sst	8	16.0	14.963107	1.036893
4	ford torino	8	17.0	14.963107	2.036893
...
393	ford mustang gl	4	27.0	29.286765	-2.286765
394	vw pickup	4	44.0	29.286765	14.713235
395	dodge rampage	4	32.0	29.286765	2.713235
396	ford ranger	4	28.0	29.286765	-1.286765
397	chevy s-10	4	31.0	29.286765	1.713235

398 rows x 5 columns

Step 4: From Siuba To Pandas

Goal: Format the de-meanded MPG to highlight most fuel efficient vehicles by engine size.

Siuba is great. It returns a `DataFrame`, which means we can use `Pandas`. One thing you might want to do is format the background in the table to highlight if vehicle model's fuel efficiency is above or below the average by engine size. We can do this with **Pandas Table Styles**.

```
[17] mpg_demeaned_by_cyl_df[['name', 'cylinders', 'mpg_demeaned_by_cyl']] \
     .sort_values('mpg_demeaned_by_cyl', ascending = False) \
     .style \
     .background_gradient()
```

	name	cylinders	mpg_demeaned_by_cyl
387	oldsmobile cutlass ciera (diesel)	6	18.014286
322	mazda glc	4	17.313235
329	honda civic 1500 gl	4	15.313235
325	vw rabbit c (diesel)	4	15.013235
394	vw pickup	4	14.713235
326	vw dasher (diesel)	4	14.113235
244	volkswagen rabbit custom diesel	4	13.813235
333	datsum 280-zx	6	12.714286
309	vw rabbit	4	12.213235
364	oldsmobile cutlass ls	8	11.636893
330	renault lecar deluxe	4	11.613235
324	datsum 210	4	11.513235
360	volvo diesel	6	10.714286
247	datsum b210 gx	4	10.113235
343	toyota starlet	4	9.813235
344	plymouth champ	4	9.713235
327	audi 5000s (diesel)	5	9.033333

[Get the code.](#)

Summary

This was a short introduction to `siuba`, which brings `dplyr` to python. If you're coming from R, `siuba` is a great package to warm yourself up to Python.

With that said, you're eventually going to want to learn `pandas`, the most widely used data wrangling tool in Python. Why?

- Most Python Teams use Pandas
- 99% of data wrangling code is written in Pandas

So, it makes sense to eventually learn Pandas to help with communication and working on R/Python teams.