



Data Analysis and Visualization using Pandas and Matplotlib

Hao Ji, Data Scientist

USC Center for Advanced Research Computing

Content

Introduction

Pandas

Matplotlib

Anaconda

Summary

Section 1: Introduction

Introduction

Section 2: Pandas

Pandas

Section 3: Matplotlib

Matplotlib

Section 4: Anaconda

Anaconda

Section 5: Summary

Summary

Introduction

Pandas: open source data analysis and manipulation tool in Python

By the end of this lecture, you should be able to:

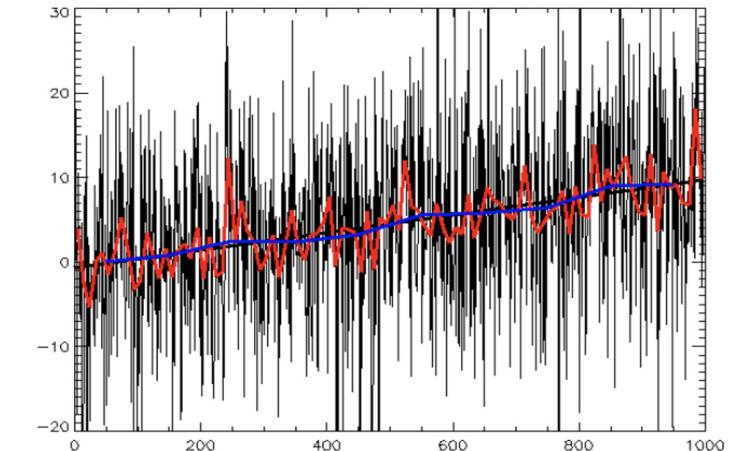
- Describe the value of Pandas to Data Science in Python
- Highlight the key data structures of Pandas

Documentation Page: https://pandas.pydata.org/docs/user_guide/index.html

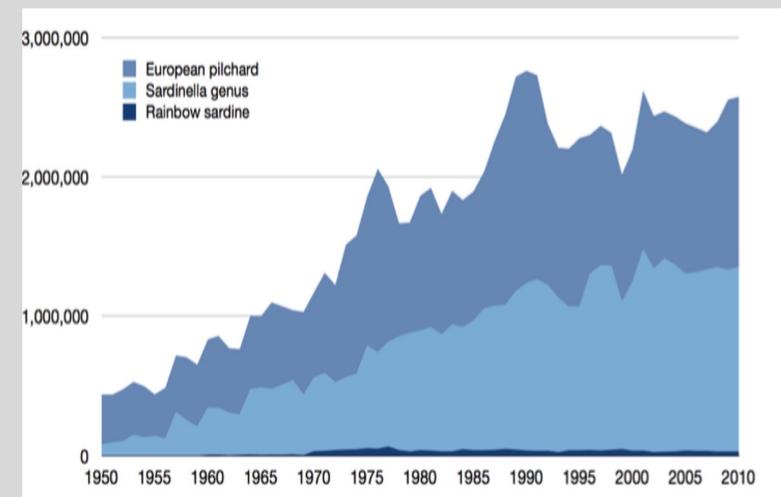
Benefits of Using Pandas

Key Benefits:

- Data variety support
- Data transformation & integration
- Data visualization



Time Series Data



Data Visualization

Using Anaconda on CARC

Anaconda: package and environment manager primarily used for open-source data science packages for the Python and R programming languages.

User Guides **Using Anaconda**

HPC Basics
Data Management
Software and Programming
 Software Module System
 Building Code With CMake
 Using MPI
 Using GPUs
 Using Julia
 Using Python
Using Anaconda
 Using R
 Using Stata
 Using MATLAB
 Using Rust
 Using Launcher
 Using Singularity
 Using Tmux
 Installing Jupyter Kernels
 Horovod for Distributed Deep Learning
Project and Allocation Management
Hybrid Cloud Computing
Secure Computing

Anaconda is a package and environment manager primarily used for open-source data science packages for the Python and R programming languages. It also supports other programming languages like C, C++, FORTRAN, Java, Scala, Ruby, and Lua.

Using Anaconda on CARC systems

Begin by logging in. You can find instructions for this in the [Getting Started with Discovery](#) or [Getting Started with Endeavour](#) user guides.

To use Anaconda, first load the corresponding module:

```
module purge
module load conda
```

This module is based on the minimal Miniconda installer. Included in all versions of Anaconda, **Conda** is the package and environment manager that installs, runs, and updates packages and their dependencies. This module also provides **Mamba**, which is a drop-in replacement for most **conda** commands that enables faster package solving, downloading, and installing.

The next step is to initialize your shell to use Conda and Mamba:

```
mamba init bash
source ~/.bashrc
```

<https://www.carc.usc.edu/user-information/user-guides/software-and-programming/anaconda>

```
mamba create -n 01-23
mamba activate 01-23
salloc
mamba install python
mamba install pandas
mamba install matplotlib
```

Creating Jupyter Kernel

A **Jupyter kernel** is a programming language-specific process that executes the code contained in a Jupyter notebook.

User Guides

HPC Basics
Data Management
Software and Programming

Software Module System
Building Code With CMake
Using MPI
Using GPUs
Using Julia
Using Python
Using Anaconda
Using R
Using Stata
Using MATLAB
Using Rust
Using Launcher
Using Singularity
Using Tmux

Installing Jupyter Kernels

Project and Allocation Management
Hybrid Cloud Computing
Secure Computing

Installing Jupyter Kernels

This user guide provides instructions for installing Jupyter kernels when using [CARC OnDemand](#). For more information about OnDemand and using Jupyter notebooks, see the [Getting Started with CARC OnDemand user guide](#).

A **Jupyter kernel** is a programming language-specific process that executes the code contained in a Jupyter notebook. The following provides installation instructions for a few popular Jupyter kernels, which will be installed in your home directory at `~/.local/share/jupyter/kernels`. Install the kernels when logged in to CARC systems before accessing them via the Jupyter OnDemand interactive app. To learn more about installing software on CARC systems using the software module system, see the [Software Module System user guide](#).

When installing kernels, make sure to use descriptive names in order to distinguish among them. Once installed, when launching Jupyter on OnDemand, the kernels will show up on a Launcher tab (File > New Launcher) and when selecting kernels through other methods.

Many software kernels are available for use with Jupyter. See a full list here:
<https://github.com/jupyter/jupyter/wiki/Jupyter-kernels>

Python

The default kernel is for Python 3.9.2, and this is ready to be used when Jupyter is launched. To use other versions of Python, enter a set of commands like the following:

```
module load usc python/<version>
python -m ipykernel install --user --name py376 --display-name "Python 3.7.6"
```

<https://www.carc.usc.edu/user-information/user-guides/software-and-programming/jupyter-kernels>

CARC Ondemand

Web Address: <https://carc-ondemand.usc.edu>

The screenshot shows the main navigation bar of the CARC OnDemand website. It includes links for About, Services, User Information, Education & Outreach, News & Events, and a prominent yellow "User Support" button. The USC logo and tagline "Advanced Research Computing Enabling scientific breakthroughs at scale" are also visible.

User Guides

Getting Started with CARC OnDemand

HPC Basics

[Getting Started with CARC OnDemand](#)

[Getting Started with Discovery](#)

[Discovery Resource Overview](#)

[Getting Started with Endeavour](#)

[Endeavour Resource Overview](#)

[Running Jobs on CARC Systems](#)

[Slurm Job Script Templates](#)

Data Management

[Software and Programming](#)

[Project and Allocation](#)

[Management](#)

[Hybrid Cloud Computing](#)

[Secure Computing](#)

- Easy file management
- Command line shell access
- Slurm job management
- Access to interactive applications, including Jupyter notebooks and RStudio Server

OnDemand is available to all CARC users. To access OnDemand, you must belong to an active project in the [CARC User Portal](#).

[Intro to CARC OnDemand video](#)

[Log in to CARC OnDemand](#)

Note: We recommend using OnDemand in a private browser to avoid potential permissions issues related to your browser's cache. If you're using a private browser and still encounter permissions issues, please [submit a help ticket](#).

The screenshot shows the user interface of the CARC OnDemand service. It features a top navigation bar with links for CARC OnDemand, Files, Jobs, Clusters, Interactive Apps, My Interactive Sessions, Help, and Log Out. Below the navigation is the USC logo and tagline. A main content area displays a message about OnDemand providing integrated access to HPC resources. At the bottom, there is a "powered by" logo for OPEN OnDemand and a note about the version: "OnDemand version: v1.8.18".

CARC Ondemand

CARC OnDemand Files ▾ Jobs ▾ Clusters ▾ Interactive Apps ▾ My Interactive Sessions Help ▾ Logged in as haoji Log Out

- Home Directory
- /project/hpcroot
- /project/wjendrzej_120
- /scratch/haoji
- /scratch2/haoji

Advanced Research Computing
Enabling scientific breakthroughs at scale

es an integrated, single access point for all of your HPC resources.

CARC Ondemand



Servers

- JupyterLab
- RStudio Server

research Computing
breakthroughs at scale

OnDemand provides an integrated, single access point for all of your HPC resources.

CARC Ondemand

The screenshot shows the CARC OnDemand web interface. At the top, there is a navigation bar with links for 'CARC OnDemand', 'Files', 'Jobs', 'Clusters', 'Interactive Apps', and 'My Interactive Sessions'. On the right side of the nav bar are 'Help', 'Logged in as haoji', and 'Log Out' buttons.

The main content area has a breadcrumb navigation: 'Home / My Interactive Sessions / JupyterLab'. On the left, there is a sidebar titled 'Interactive Apps' with 'Servers' listed. Under 'Servers', 'JupyterLab' is highlighted with a blue background and white text, while 'RStudio Server' is shown below it.

The main form is titled 'JupyterLab'. It contains the following fields:

- Cluster:** A dropdown menu set to 'discovery'.
- Account:** A dropdown menu set to 'wjendrzej_120'.
- Partition:** A dropdown menu set to 'main'.
- Number of CPUs:** A dropdown menu set to '1'.
- Memory (GB):** An input field with a dropdown arrow, currently empty.
- GPU Type (optional):** An input field with a dropdown arrow, currently empty.

Below the 'Number of CPUs' field, there is a note: 'Number of CPU cores to allocate.' Below the 'Memory (GB)' field, there is a note: 'Amount of memory to allocate. If left blank, a default of 2 GB of memory per CPU core will be allocated.'

An orange arrow points from the 'Discovery Resource Overview' link in the 'Number of CPUs' note to the URL in the callout box.

<https://www.carc.usc.edu/user-information/user-guides/hpc-basics/discovery-resources>

10

Pandas Data Structure: Series

- 1-dimensional labelled array
- Supports many data types
- Axis labels -> index (get and set values by index label)

Pandas Data Structure: Series

```
[ ]: data = np.random.rand(5)  
pd.Series(data)
```

```
0    0.531397  
1    0.010047  
2    0.929603  
3    0.627307  
4    0.637857  
dtype: float64
```

```
[ ]: pd.Series(data, index = ['Astronomy', 'Geology', 'Economics', 'English', 'History'])
```

```
Astronomy    0.531397  
Geology      0.010047  
Economics    0.929603  
English       0.627307  
History       0.637857  
dtype: float64
```

Pandas Data Structure: DataFrame

```
[ ]: planets_data = np.array([[0.330, 4879, 3.7, 88.0],  
[4.87, 12104, 8.9, 224.7],  
[5.97, 12756, 9.8, 365.2 ],  
[0.642, 6792, 3.7, 687.0],  
[1898, 142984, 23.1, 4331],  
[568, 120536, 9.0, 10747],  
[86.8, 51118, 8.7, 30589],  
[102, 49528, 11.0, 59800]])  
planets_data
```

```
array([[3.30000e-01, 4.87900e+03, 3.70000e+00, 8.80000e+01],  
[4.87000e+00, 1.21040e+04, 8.90000e+00, 2.24700e+02],  
[5.97000e+00, 1.27560e+04, 9.80000e+00, 3.65200e+02],  
[6.42000e-01, 6.79200e+03, 3.70000e+00, 6.87000e+02],  
[1.89800e+03, 1.42984e+05, 2.31000e+01, 4.33100e+03],  
[5.68000e+02, 1.20536e+05, 9.00000e+00, 1.07470e+04],  
[8.68000e+01, 5.11180e+04, 8.70000e+00, 3.05890e+04],  
[1.02000e+02, 4.95280e+04, 1.10000e+01, 5.98000e+04]])
```

Pandas Data Structure: DataFrame

```
[ ]: planets = pd.DataFrame(planets_data,
                           columns=['mass', 'diameter', 'gravity', 'period'],
                           index=['Mercury', 'Venus', 'Earth', 'Mars', 'Jupiter', 'Saturn', 'Uranus', 'Neptune'])
      planets
```

	mass	diameter	gravity	period
Mercury	0.330	4879.0	3.7	88.0
Venus	4.870	12104.0	8.9	224.7
Earth	5.970	12756.0	9.8	365.2
Mars	0.642	6792.0	3.7	687.0
Jupiter	1898.000	142984.0	23.1	4331.0
Saturn	568.000	120536.0	9.0	10747.0
Uranus	86.800	51118.0	8.7	30589.0
Neptune	102.000	49528.0	11.0	59800.0

Data Integration

By the end of this lecture, you should be able to:

- Describe the efficient and easy to use methods that pandas provides for importing data into memory
- Identify functions such as ‘read_csv’ for reading a CSV file into a DataFrame
- Discuss about other data sources that pandas can directly import from

Pandas: Data Ingestion



Pandas: Data Ingestion

read_csv:

```
[ ]: solar_system = pd.read_csv('solar_system_abbr.csv', index_col=0)
solar_system
```

Name	Type	Order from Sun	Diameter	Mass
Sun	Star	0	1392000.0	333000.00000
Mercury	Terrestrial planet	1	4878.0	0.05500
Venus	Terrestrial planet	2	12104.0	0.81500
Earth	Terrestrial planet	3	12756.0	1.00000
Mars	Terrestrial planet	4	6787.0	0.10700
Jupiter	Gas giant	6	142800.0	318.00000
Saturn	Gas giant	7	120000.0	95.00000
Uranus	Gas giant	8	51118.0	15.00000
Neptune	Gas giant	9	49528.0	17.00000
Ceres	Dwarf planet	5	974.6	0.00016
Pluto	Dwarf planet	10	2300.0	0.00200
Haumea	Dwarf planet	11	1300.0	0.00070
Makemake	Dwarf planet	12	1420.0	0.00067
Eris	Dwarf planet	13	2326.0	0.00280

Pandas: Descriptive Statistics

By the end of this section, you should be able to:

- Describe the capabilities of Pandas for performing statistical analysis on data
- Leverage frequently used functions such as `describe()`
- Explore other statistical functions in Pandas, which is constantly evolving

Pandas Data Structure: DataFrame

```
[ ]: solar_system.describe()
```

	Order from Sun	Diameter	Mass
count	14.0000	1.400000e+01	14.000000
mean	6.5000	1.285923e+05	23817.641666
std	4.1833	3.665235e+05	88988.845412
min	0.0000	9.746000e+02	0.000160
25%	3.2500	2.306500e+03	0.002200
50%	6.5000	9.445500e+03	0.461000
75%	9.7500	5.072050e+04	16.500000
max	13.0000	1.392000e+06	333000.000000

Pandas Data Structure: DataFrame

General functax for calling these function is `data_frame.func()`

`Func = min()`

`Func= max()`

`Func = std()`

`Func = mode()`

`Func = median()`

`Func = any()`

`Func = all()`

Pandas: Data Cleaning

By the end of this section, you should be able to:

- Explain why there is a need to clean data
- Describe data cleaning as an activity
- Leverage key methods pandas provides for data cleaning

Pandas: Data Cleaning

Real-world data is messy!

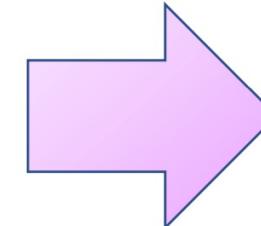
- Missing values
- Outliers in the data
- Invalid data (e.g. negative values for age)
- NaN value (`np.nan`)
- None value

Pandas: Data Cleaning

df.replace()

	0	1
0	-0.349596	-2.017159
1	9999.000000	9999.000000
2	9999.000000	9999.000000
3	0.113889	0.616122
4	0.014707	-1.731660
5	9999.000000	9999.000000
6	1.233087	0.720138
7	9999.000000	9999.000000
8	9999.000000	9999.000000
9	9999.000000	9999.000000

9999.0000



```
df=df.replace(9999.0, 0)  
df
```

	0	1
0	-0.349596	-2.017159
1	0.000000	0.000000
2	0.000000	0.000000
3	0.113889	0.616122
4	0.014707	-1.731660
5	0.000000	0.000000
6	1.233087	0.720138
7	0.000000	0.000000
8	0.000000	0.000000
9	0.000000	0.000000

0.0000

Pandas: Data Cleaning

df.dropna()

	0	1	2
0	NaN	NaN	-0.335410
1	NaN	NaN	0.685743
2	0.077005	0.085073	0.565144
3	0.394961	-1.829587	0.494039
4	1.486227	-0.480726	-0.127278
5	NaN	NaN	-0.047668
6	NaN	NaN	-1.504804
7	-0.530518	-0.881817	-2.687352
8	0.825376	1.042468	-0.311527
9	0.097617	1.373572	-0.682435

df.dropna(axis=0)

	0	1	2
2	0.077005	0.085073	0.565144
3	0.394961	-1.829587	0.494039
4	1.486227	-0.480726	-0.127278
7	-0.530518	-0.881817	-2.687352
8	0.825376	1.042468	-0.311527
9	0.097617	1.373572	-0.682435

df.dropna(axis=1)

	2
0	-0.335410
1	0.685743
2	0.565144
3	0.494039
4	-0.127278
5	-0.047668
6	-1.504804
7	-2.687352
8	-0.311527
9	-0.682435

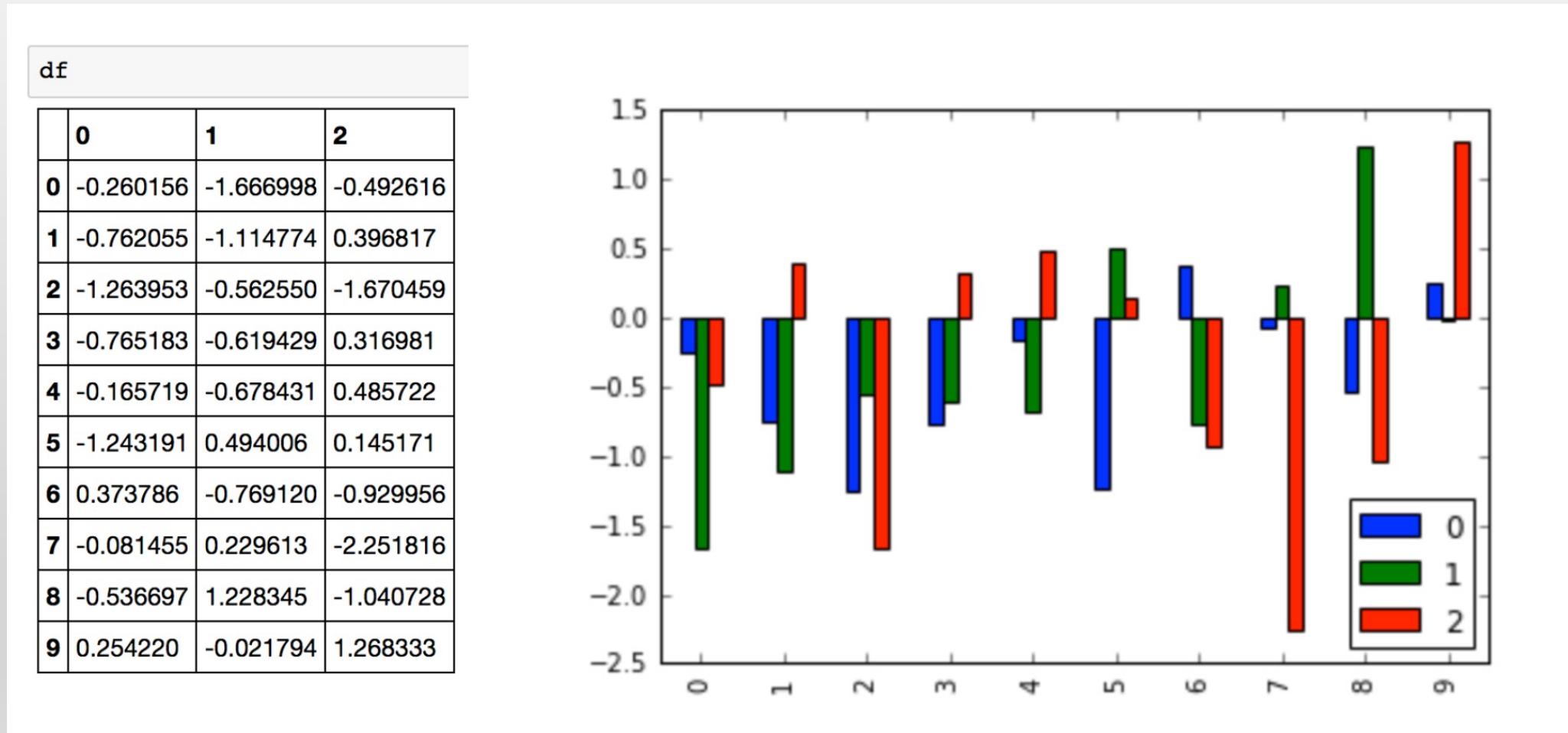
Pandas: Data Visualization

By the end of this section, you should be able to:

- Identify key plotting functions of Pandas
- Recognize the ease of utilization of Pandas methods for visualization

Pandas: Data Visualization

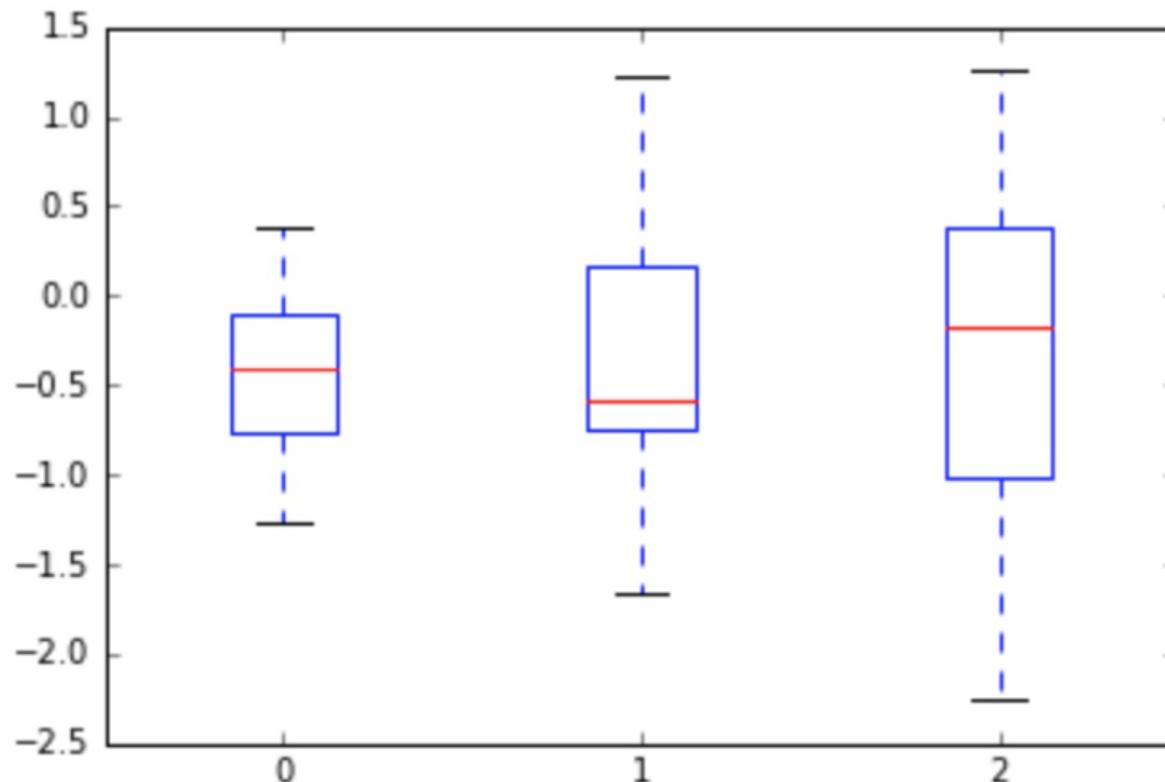
Comand: df.plot.bar()



Pandas: Data Visualization

Comand: df.plot.box()

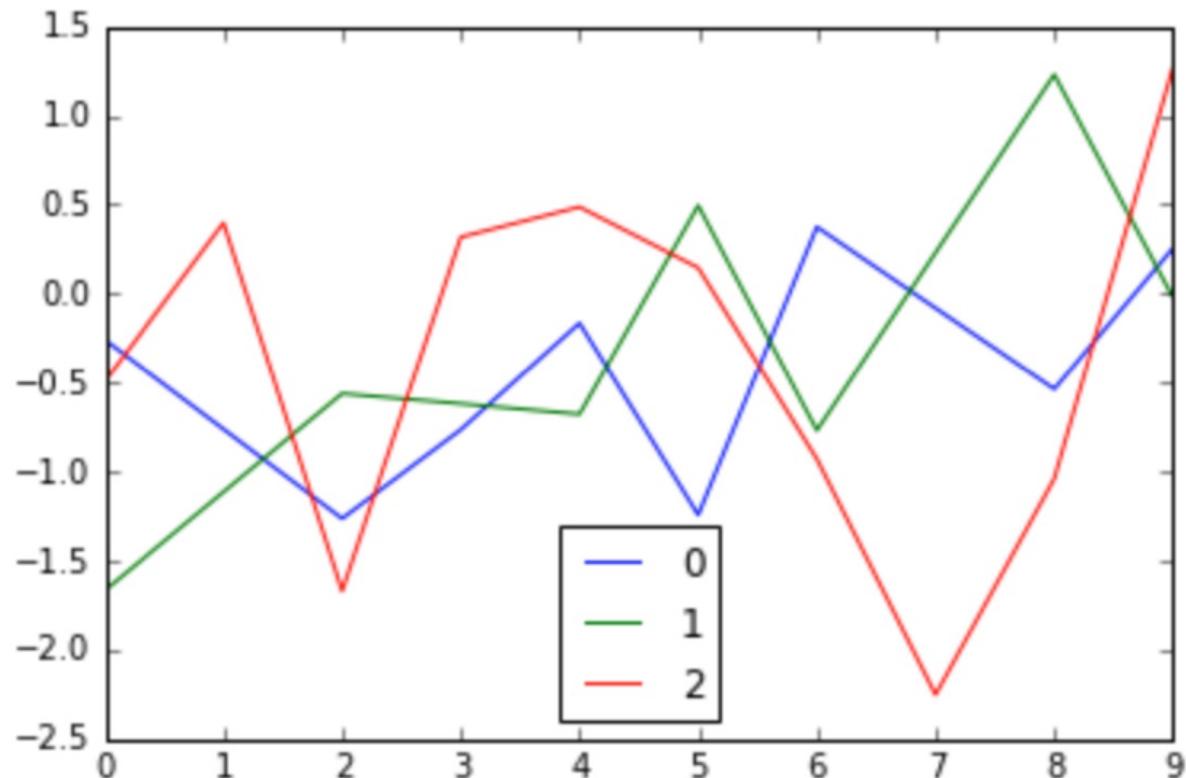
df	0	1	2
0	-0.260156	-1.666998	-0.492616
1	-0.762055	-1.114774	0.396817
2	-1.263953	-0.562550	-1.670459
3	-0.765183	-0.619429	0.316981
4	-0.165719	-0.678431	0.485722
5	-1.243191	0.494006	0.145171
6	0.373786	-0.769120	-0.929956
7	-0.081455	0.229613	-2.251816
8	-0.536697	1.228345	-1.040728
9	0.254220	-0.021794	1.268333



Pandas: Data Visualization

Comand: df.plot()

	0	1	2
0	-0.260156	-1.666998	-0.492616
1	-0.762055	-1.114774	0.396817
2	-1.263953	-0.562550	-1.670459
3	-0.765183	-0.619429	0.316981
4	-0.165719	-0.678431	0.485722
5	-1.243191	0.494006	0.145171
6	0.373786	-0.769120	-0.929956
7	-0.081455	0.229613	-2.251816
8	-0.536697	1.228345	-1.040728
9	0.254220	-0.021794	1.268333



Pandas: Frequent data operations

By the end of this section, you should be able to:

- Handpick data (rows or columns) in a DataFrame using Pandas methods
- Add or Delete rows or columns in a DataFrame
- Perform aggregation operations / group by

Pandas: Frequent data operations

Slice out columns:

df

	sensor1	sensor2	sensor3
0	-0.260156	-1.666998	-0.492616
1	-0.762055	-1.114774	0.396817
2	-1.263953	-0.562550	-1.670459
3	-0.765183	-0.619429	0.316981
4	-0.165719	-0.678431	0.485722
5	-1.243191	0.494006	0.145171
6	0.373786	-0.769120	-0.929956
7	-0.081455	0.229613	-2.251816
8	-0.536697	1.228345	-1.040728
9	0.254220	-0.021794	1.268333

df['sensor1']

```
0   -0.260156
1   -0.762055
2   -1.263953
3   -0.765183
4   -0.165719
5   -1.243191
6    0.373786
7   -0.081455
8   -0.536697
9    0.254220
Name: sensor1, dtype: float64
```

Pandas: Frequent data operations

Filter out rows:

```
df
```

	sensor1	sensor2	sensor3
0	-0.260156	-1.666998	-0.492616
1	-0.762055	-1.114774	0.396817
2	-1.263953	-0.562550	-1.670459
3	-0.765183	-0.619429	0.316981
4	-0.165719	-0.678431	0.485722
5	-1.243191	0.494006	0.145171
6	0.373786	-0.769120	-0.929956
7	-0.081455	0.229613	-2.251816
8	-0.536697	1.228345	-1.040728
9	0.254220	-0.021794	1.268333

```
#Select rows where sensor2 is positive  
df[df['sensor2'] > 0]
```

	sensor1	sensor2	sensor3
5	-1.243191	0.494006	0.145171
7	-0.081455	0.229613	-2.251816
8	-0.536697	1.228345	-1.040728

Pandas: Frequent data operations

Add a new row:

df

	sensor1	sensor2	sensor3
0	-0.260156	-1.666998	-0.492616
1	-0.762055	-1.114774	0.396817
2	-1.263953	-0.562550	-1.670459
3	-0.765183	-0.619429	0.316981
4	-0.165719	-0.678431	0.485722
5	-1.243191	0.494006	0.145171
6	0.373786	-0.769120	-0.929956
7	-0.081455	0.229613	-2.251816
8	-0.536697	1.228345	-1.040728
9	0.254220	-0.021794	1.268333

df.loc[10] = [11, 22, 33, 44]

df

	sensor1	sensor2	sensor3	sensor4
0	-0.260156	-1.666998	-0.492616	0.242671
1	-0.762055	-1.114774	0.396817	0.157464
2	-1.263953	-0.562550	-1.670459	2.790434
3	-0.765183	-0.619429	0.316981	0.100477
4	-0.165719	-0.678431	0.485722	0.235925
5	-1.243191	0.494006	0.145171	0.021075
6	0.373786	-0.769120	-0.929956	0.864819
7	-0.081455	0.229613	-2.251816	5.070676
8	-0.536697	1.228345	-1.040728	1.083115
9	0.254220	-0.021794	1.268333	1.608669
10	11.000000	22.000000	33.000000	44.000000

Pandas: Frequent data operations

Delete a row:

df

	sensor1	sensor2	sensor3
0	-0.260156	-1.666998	-0.492616
1	-0.762055	-1.114774	0.396817
2	-1.263953	-0.562550	-1.670459
3	-0.765183	-0.619429	0.316981
4	-0.165719	-0.678431	0.485722
5	-1.243191	0.494006	0.145171
6	0.373786	-0.769120	-0.929956
7	-0.081455	0.229613	-2.251816
8	-0.536697	1.228345	-1.040728
9	0.254220	-0.021794	1.268333

df.drop(df.index[5])

	sensor1	sensor2	sensor3	sensor4
0	-0.260156	-1.666998	-0.492616	0.242671
1	-0.762055	-1.114774	0.396817	0.157464
2	-1.263953	-0.562550	-1.670459	2.790434
3	-0.765183	-0.619429	0.316981	0.100477
4	-0.165719	-0.678431	0.485722	0.235925
6	0.373786	-0.769120	-0.929956	0.864819
7	-0.081455	0.229613	-2.251816	5.070676
8	-0.536697	1.228345	-1.040728	1.083115
9	0.254220	-0.021794	1.268333	1.608669

Pandas: Frequent data operations

Delete a column:

```
df
```

	sensor1	sensor2	sensor3	sensor4
0	-0.260156	-1.666998	-0.492616	0.242671
1	-0.762055	-1.114774	0.396817	0.157464
2	-1.263953	-0.562550	-1.670459	2.790434
3	-0.765183	-0.619429	0.316981	0.100477
4	-0.165719	-0.678431	0.485722	0.235925
5	-1.243191	0.494006	0.145171	0.021075
6	0.373786	-0.769120	-0.929956	0.864819
7	-0.081455	0.229613	-2.251816	5.070676
8	-0.536697	1.228345	-1.040728	1.083115
9	0.254220	-0.021794	1.268333	1.608669

```
del df['sensor1']
```

```
df
```

	sensor2	sensor3	sensor4
0	-1.666998	-0.492616	0.242671
1	-1.114774	0.396817	0.157464
2	-0.562550	-1.670459	2.790434
3	-0.619429	0.316981	0.100477
4	-0.678431	0.485722	0.235925
5	0.494006	0.145171	0.021075
6	-0.769120	-0.929956	0.864819
7	0.229613	-2.251816	5.070676
8	1.228345	-1.040728	1.083115
9	-0.021794	1.268333	1.608669

Pandas: Merging DataFrames

By the end of this section, you should be able to:

- Explain that data is usually distributed across different locations and tables
- Combine data from distinct DataFrames to obtain the big picture
- Distinguish among different ways to combine datasets

Pandas: Merging DataFrames

Example Dataframes

left

	_key1	_key2	city	user_name
0	K0	z0	city_0	user_0
1	K1	z1	city_1	user_1
2	K2	z2	city_2	user_2
3	K3	z3	city_3	user_3

right

	_key1	_key2	hire_date	profession
0	K0	z0	h_0	p_0
1	K1	z1	h_1	p_1
2	K2	z2	h_2	p_2
3	K3	z3	h_3	p_3

Pandas: Merging DataFrames

Pandas.concat(): stack dataframes

```
pd.concat([left, left])
```

	_key1	_key2	city	user_name
0	K0	z0	city_0	user_0
1	K1	z1	city_1	user_1
2	K2	z2	city_2	user_2
3	K3	z3	city_3	user_3

	_key1	_key2	city	user_name
0	K0	z0	city_0	user_0
1	K1	z1	city_1	user_1
2	K2	z2	city_2	user_2
3	K3	z3	city_3	user_3

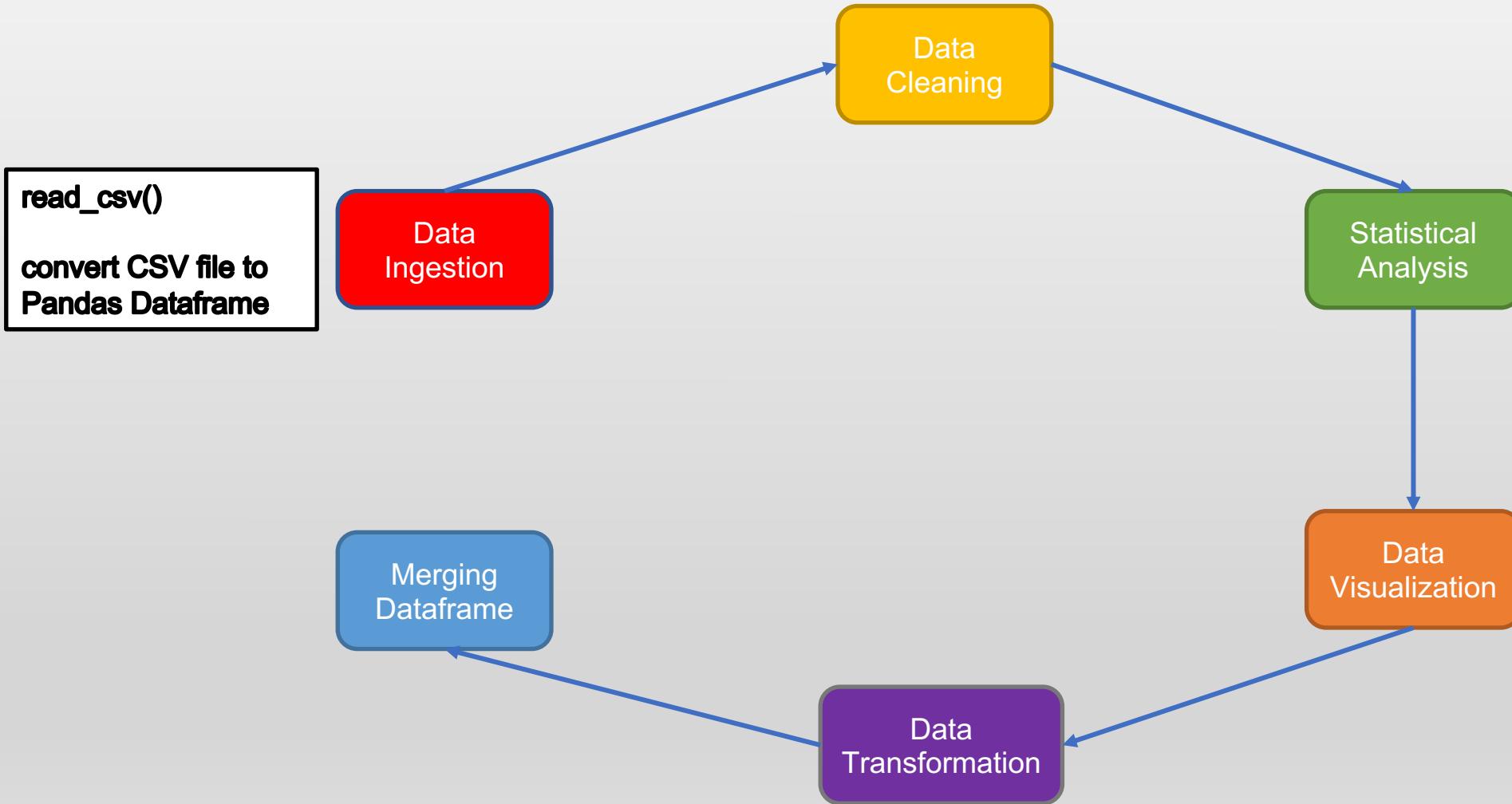
Pandas: Merging DataFrames

Pandas.concat(): stack dataframes

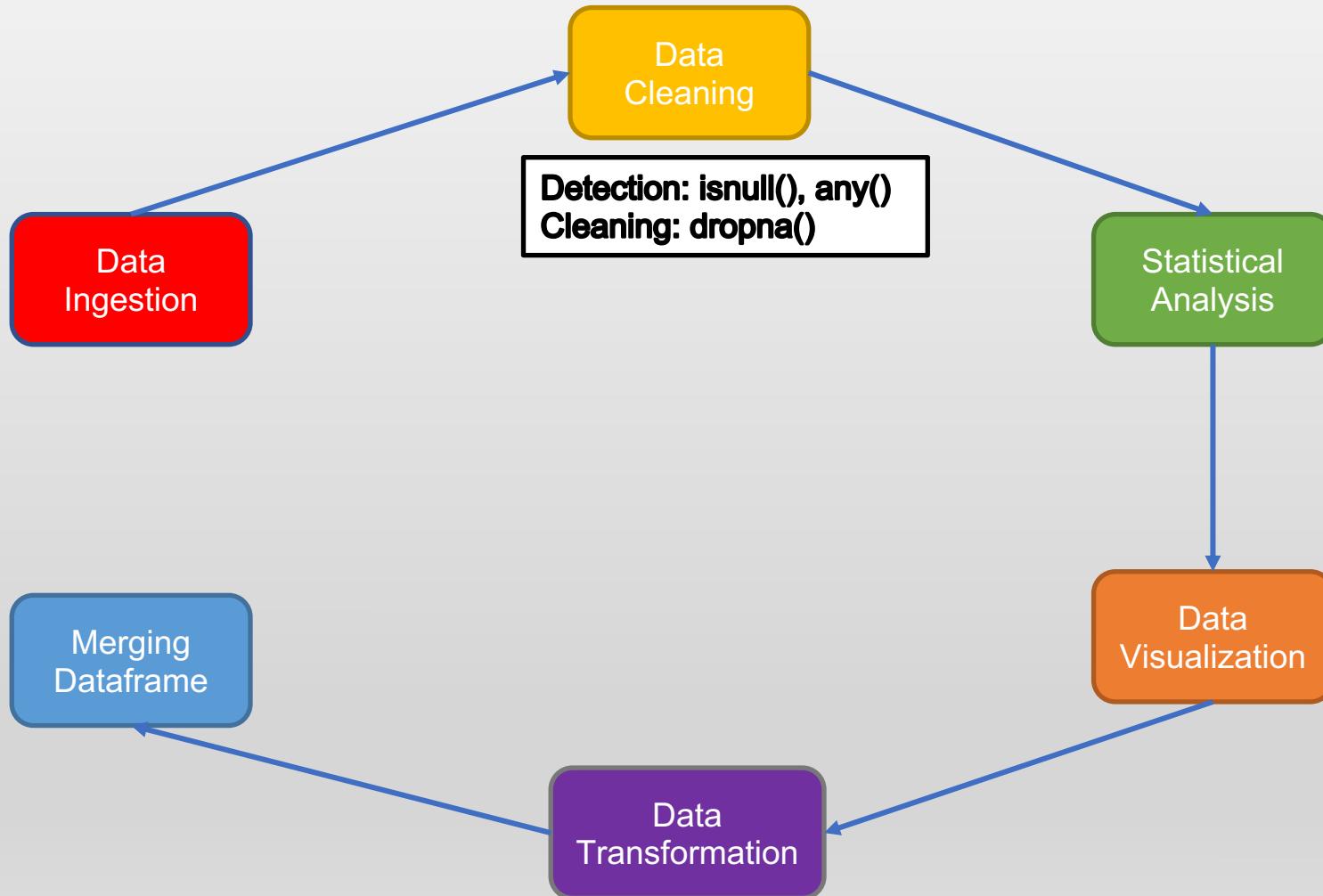
```
pd.concat([left, right])
```

	_key1	_key2	city	hire_date	profession	user_name
0	K0	z0	city_0	NaN	NaN	user_0
1	K1	z1	city_1	NaN	NaN	user_1
2	K2	z2	city_2	NaN	NaN	user_2
3	K3	z3	city_3	NaN	NaN	user_3
0	K0	z0	NaN	h_0	p_0	NaN
1	K1	z1	NaN	h_1	p_1	NaN
2	K2	z2	NaN	h_2	p_2	NaN
3	K3	z3	NaN	h_3	p_3	NaN

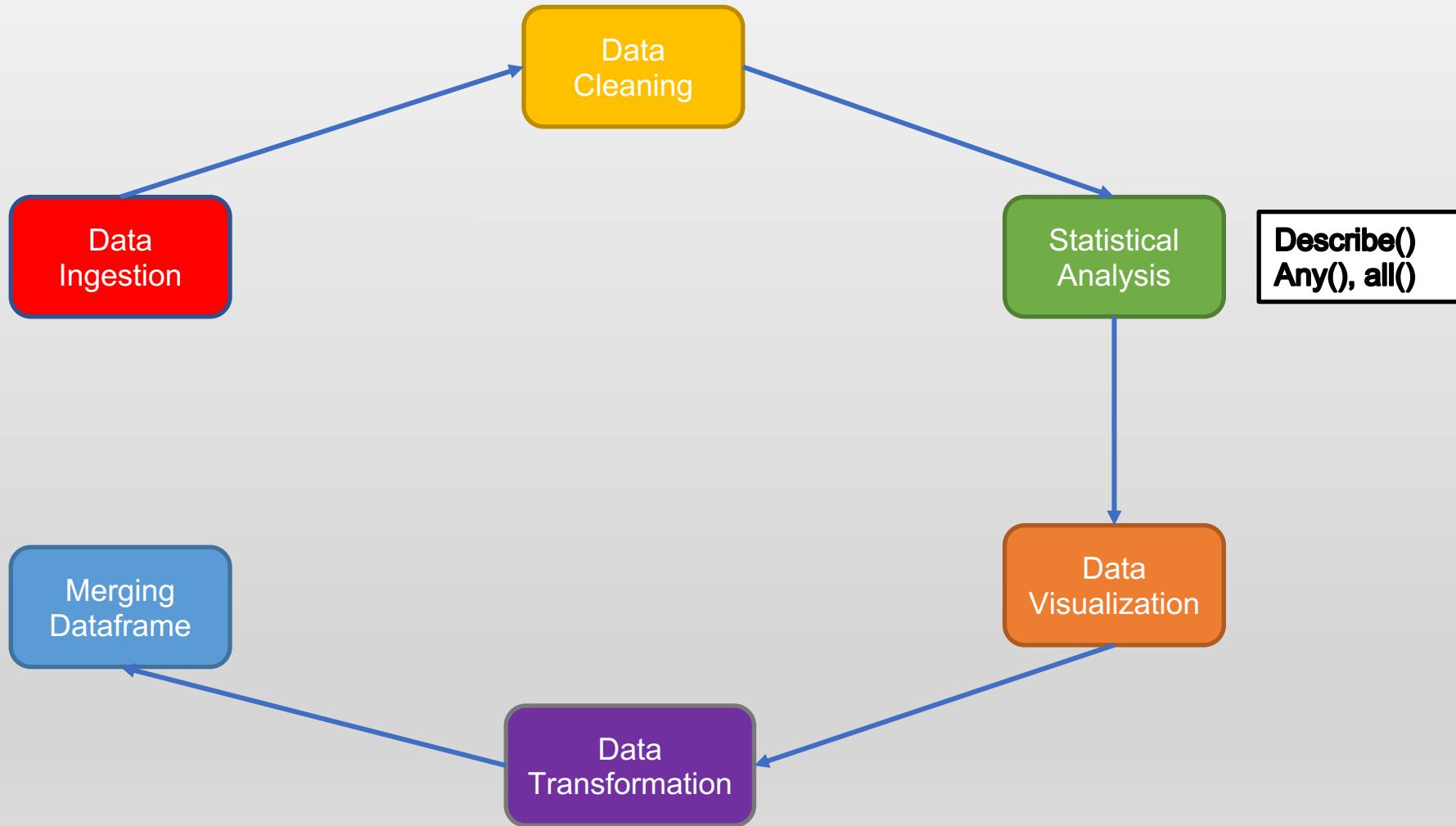
Pandas: Summary



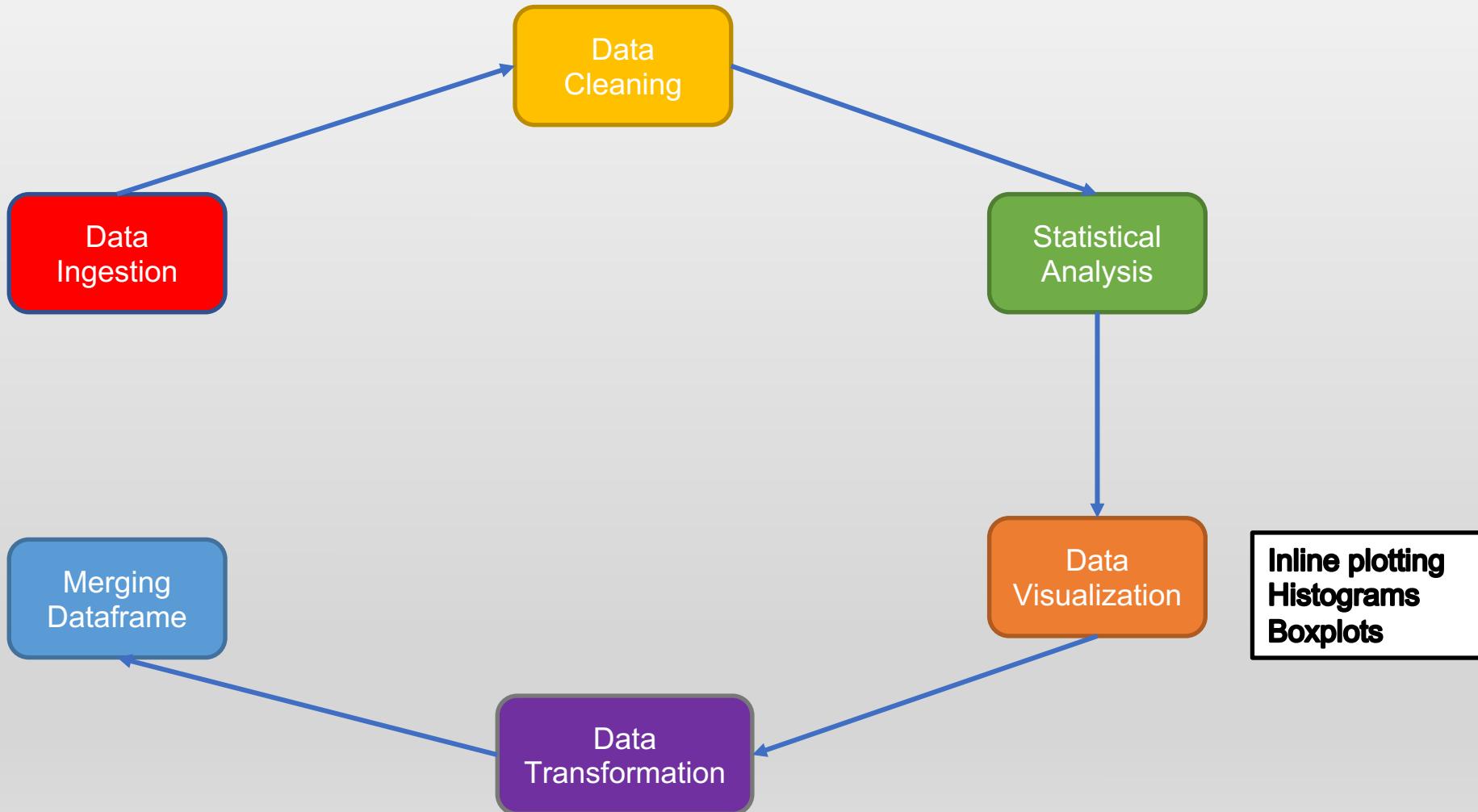
Pandas: Case Study



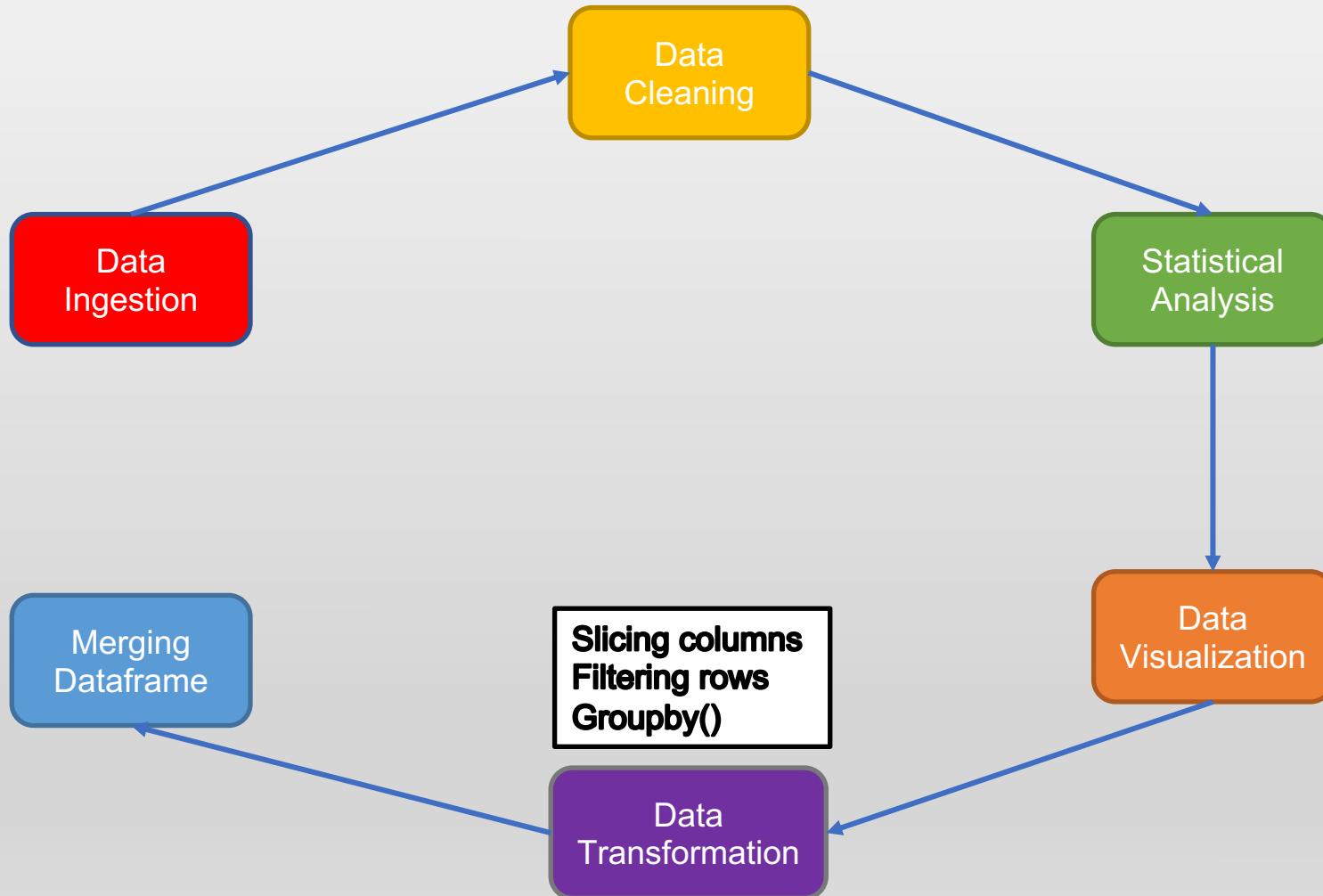
Pandas: Case Study



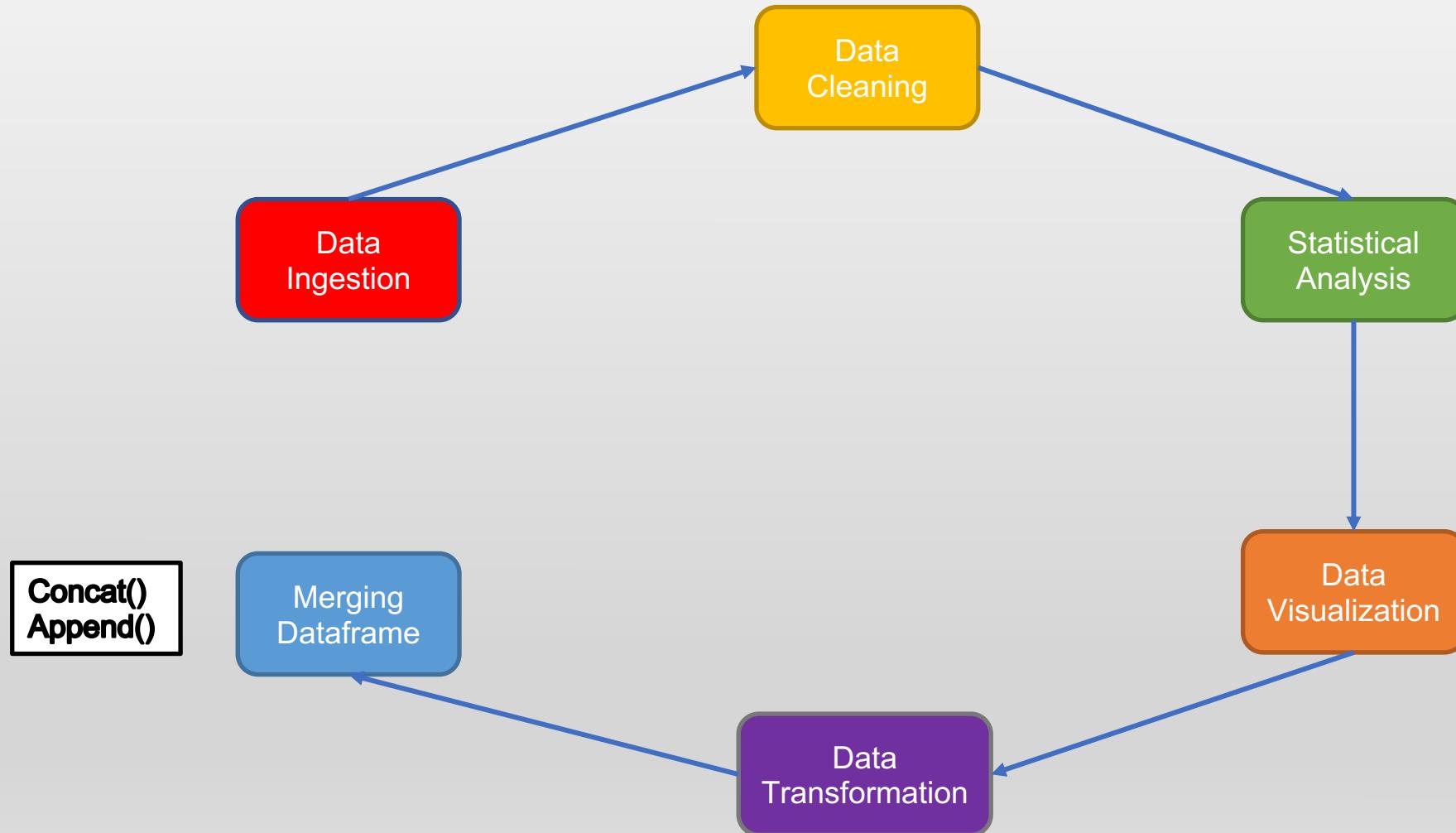
Pandas: Case Study



Pandas: Case Study



Pandas: Case Study



Matplotlib

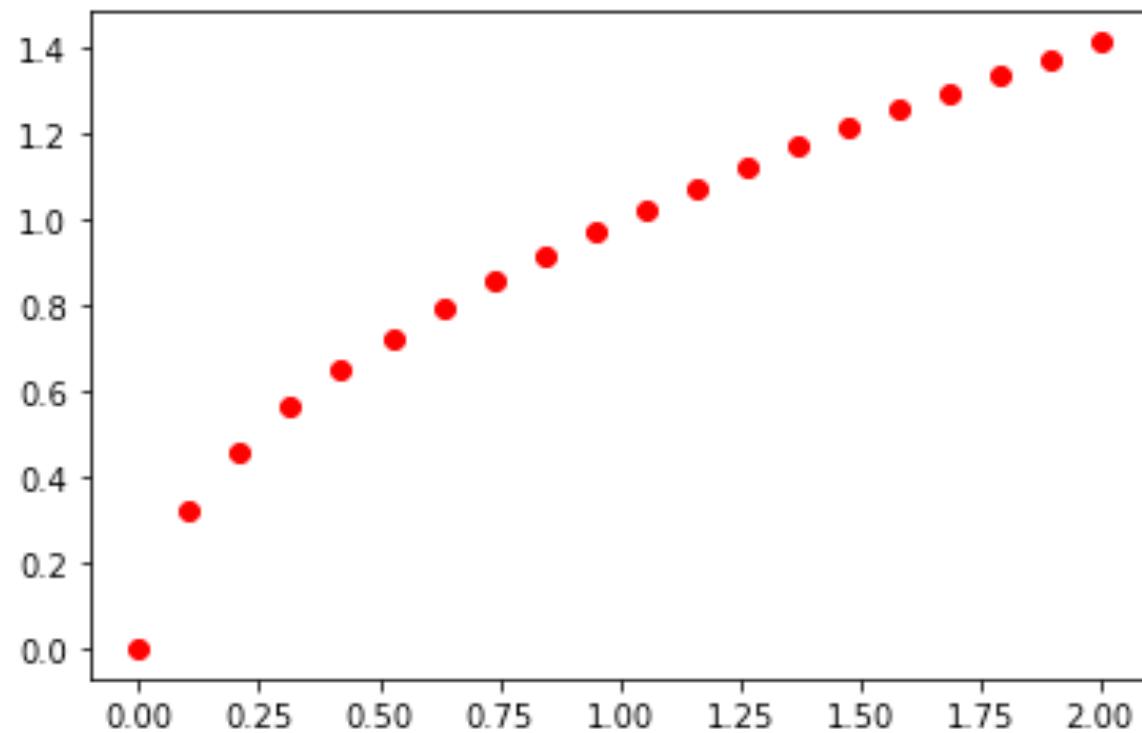
1. 2D plotting single line figure

```
[ ]: %matplotlib inline  
import numpy as np  
import matplotlib.pyplot as plt  
x = np.linspace(0.0, 2.0, 20)  
x
```

```
array([ 0.          ,  0.10526316,  0.21052632,  0.31578947,  0.42105263,  
       0.52631579,  0.63157895,  0.73684211,  0.84210526,  0.94736842,  
       1.05263158,  1.15789474,  1.26315789,  1.36842105,  1.47368421,  
       1.57894737,  1.68421053,  1.78947368,  1.89473684,  2.          ])
```

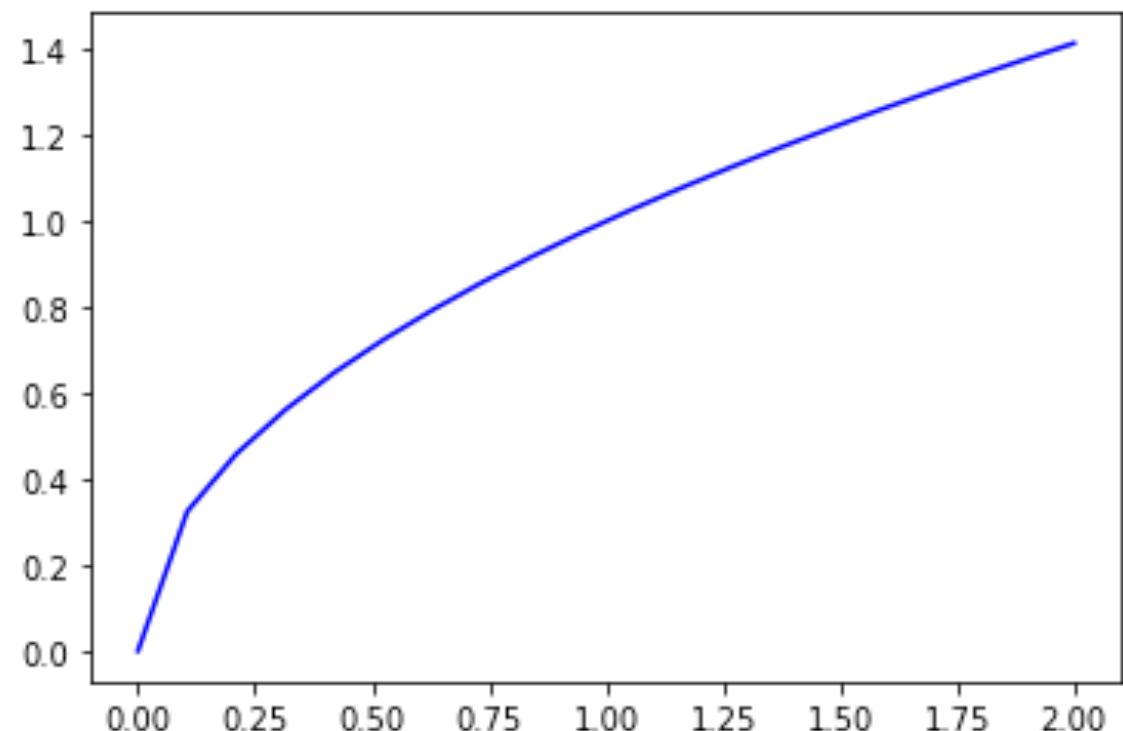
Matplotlib

```
[ ]: plt.plot(x, np.sqrt(x), 'ro')
plt.show()
```



Matplotlib

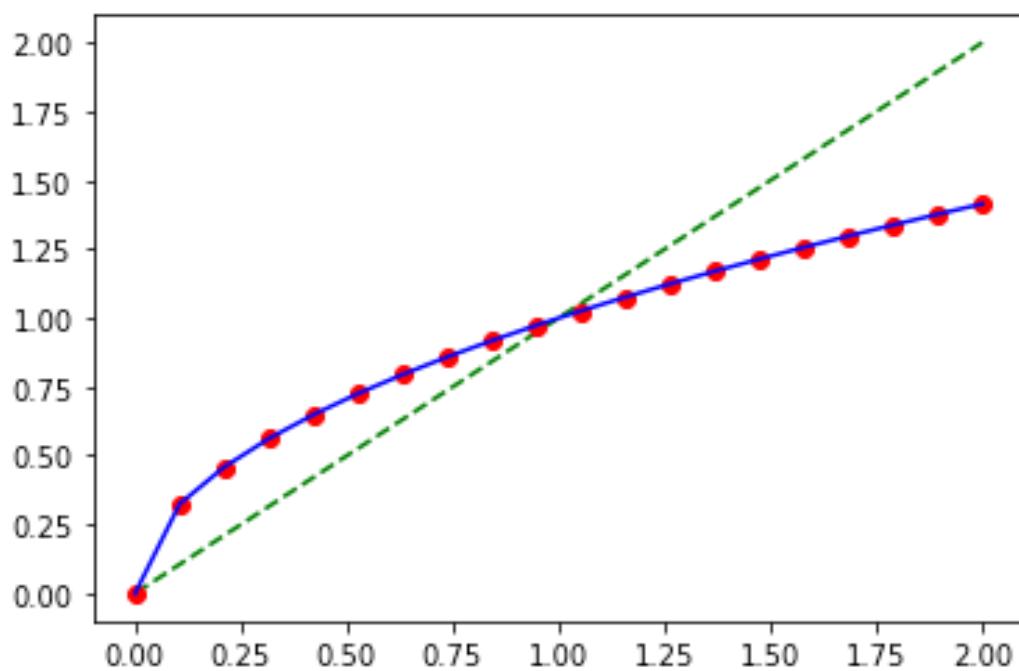
```
[ ]: plt.plot(x, np.sqrt(x), 'b-')
plt.show()
```



Matplotlib

2. Multiple lines on the same figure

```
[ ]: plt.plot(x, x, 'g--', label='green') # green dashed line  
plt.plot(x, np.sqrt(x), 'ro', label='red') # red circles  
plt.plot(x, np.sqrt(x), 'b-', label='blue') # blue line  
plt.show()
```

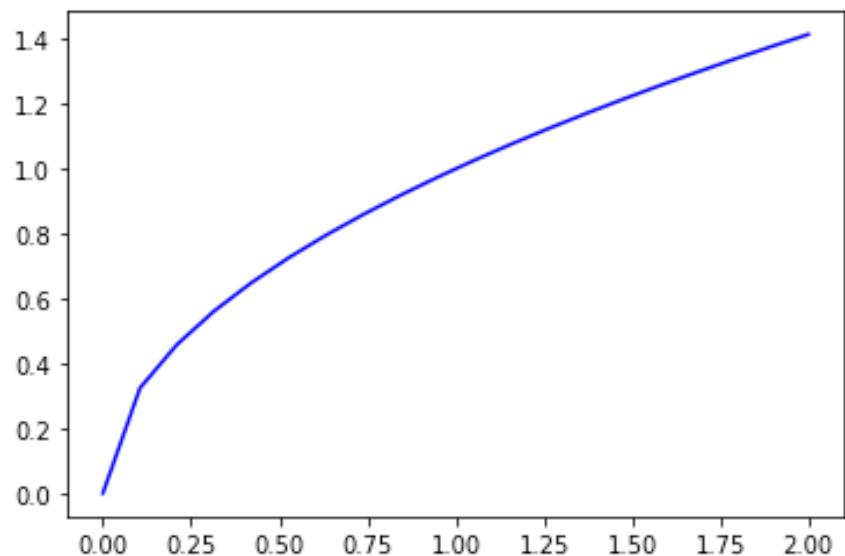


Matplotlib

3. Save a figure to a file

```
[ ]: plt.plot(x, np.sqrt(x), 'b-')

plt.savefig('myfig.png') # can also save directly to pdf by just changing the file extension
```



Using Anaconda & Juptyer Kernel on CARC



Summary

Import



Transform



Visualize

