INTRODUCTION TO PYTHON

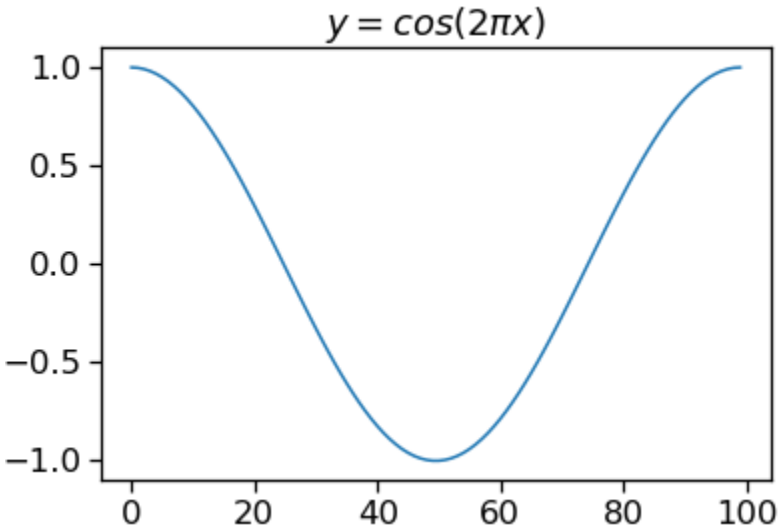
LECTURE 6: Data visualization

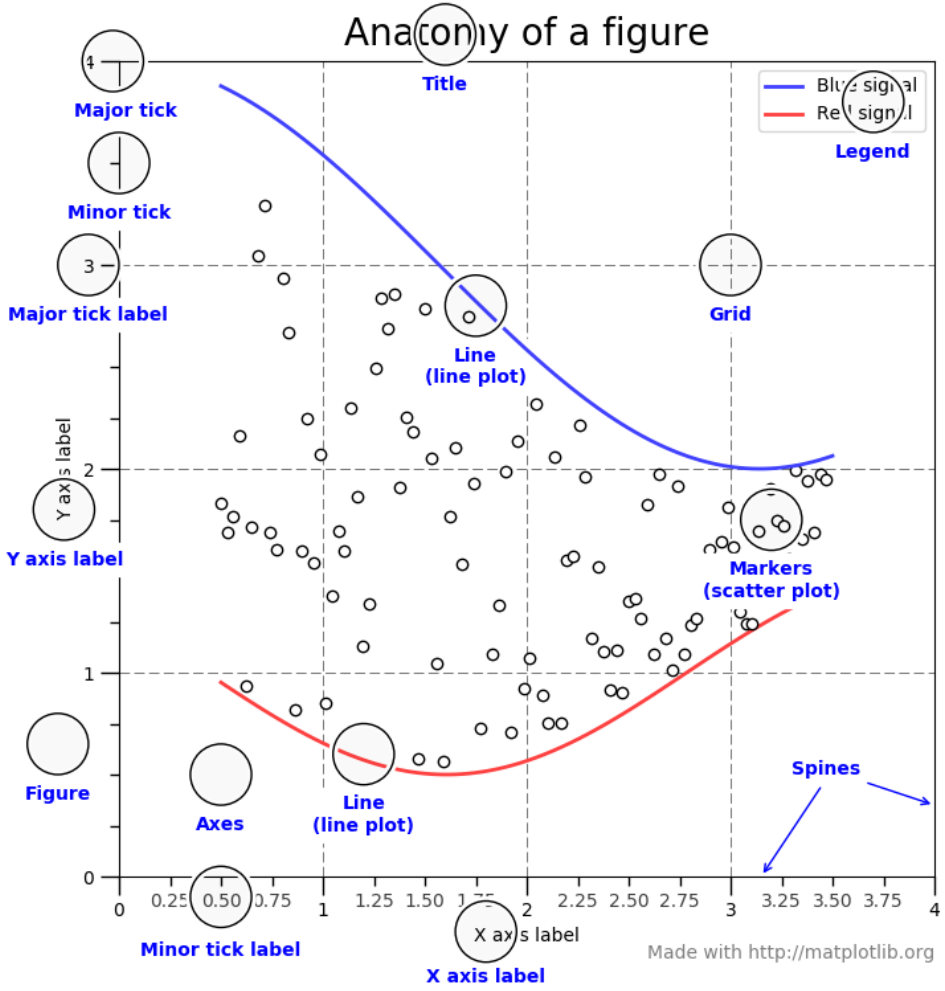
Asem Elshimi

asem.elshimi@austin.utexas.edu

Working with a sequence of numbers

```
xsequence=np.linspace(0,1,100)
y=np.cos(xsequence*2*np.pi)


fig=plt.figure()
ax=fig.add_subplot(1,1,1)
ax.plot(y)
ax.set_title('$y=cos(2 \pi x)$')
plt.show()
```




Anatomy of a figure
Title
Major tick
Minor tick
Major tick label
Legend
Grid
Line
(line plot)
Y axis label
Markers
(scatter plot)
Figure
Axes
Line
(line plot)
Spines
Minor tick label
X axis label
Made with http://matplotlib.org

Data visualization

What is wrong with this graph?

COURTESY OF FOX NEWS

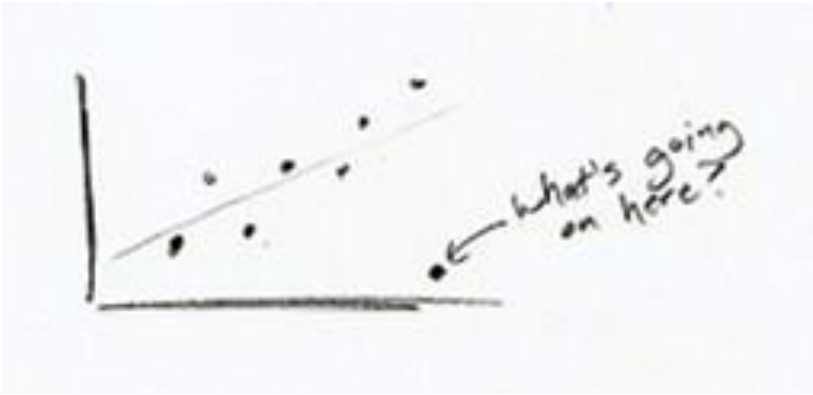
Communicating results with scientific graphs.

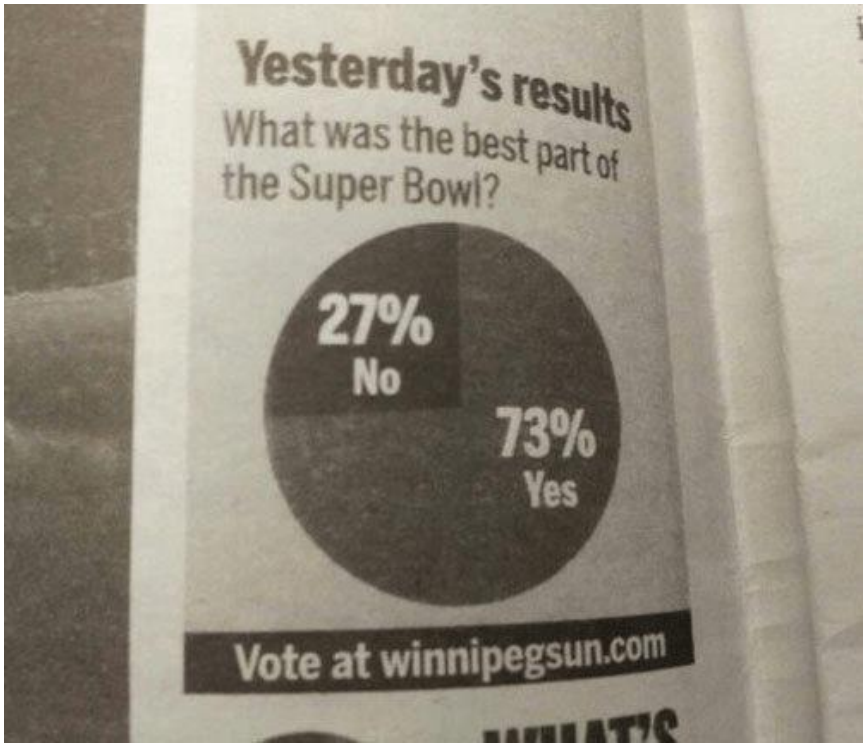
Do you need a graph?
- Maybe a table is sufficient

What types of variables do you have?
- Continuous, discrete, categorical.
- Independent and dependent variables.

What is your message?

Basic rules:
- Check the data.
- Explain encodings
- Label axes
- Include units
- Include your resources





Basic rules for graphs

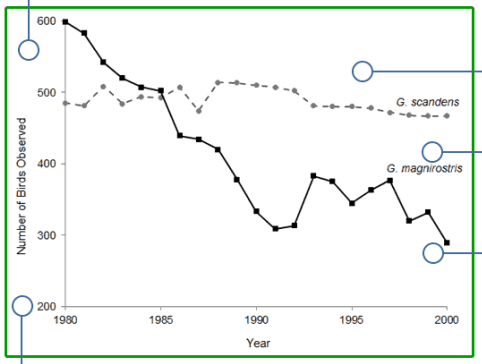
Can be interpreted in black and white

Title:

- **Descriptive:** Figure 1. Effects of dam construction on fish biodiversity.
- **Assertive:** Figure 1. Dam construction results in loss of fish biodiversity.

Line graph

Identifying trends over time | Comparing categories over time



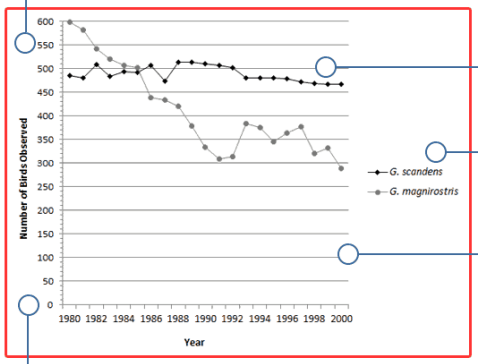
Both axes only include the necessary value labels and tick marks to make the key message easily understandable.

The white background and lack of gridlines produces a clean and uncluttered graph.

The categories are labelled directly so the audience can instantly identify the categories.

The data series with the dramatic downward trend is highlighted by using darker shading.

The y-axis begins at 200 which allows the data to fill the graph space. This creates a clear message and maximises impact.



Both axes have too many tick marks and too many value labels. The result is a graph that is cluttered and difficult to interpret.

Darker shading can be used to highlight important data for the audience. In this case, the data series showing a dramatic downward trend should have been highlighted.

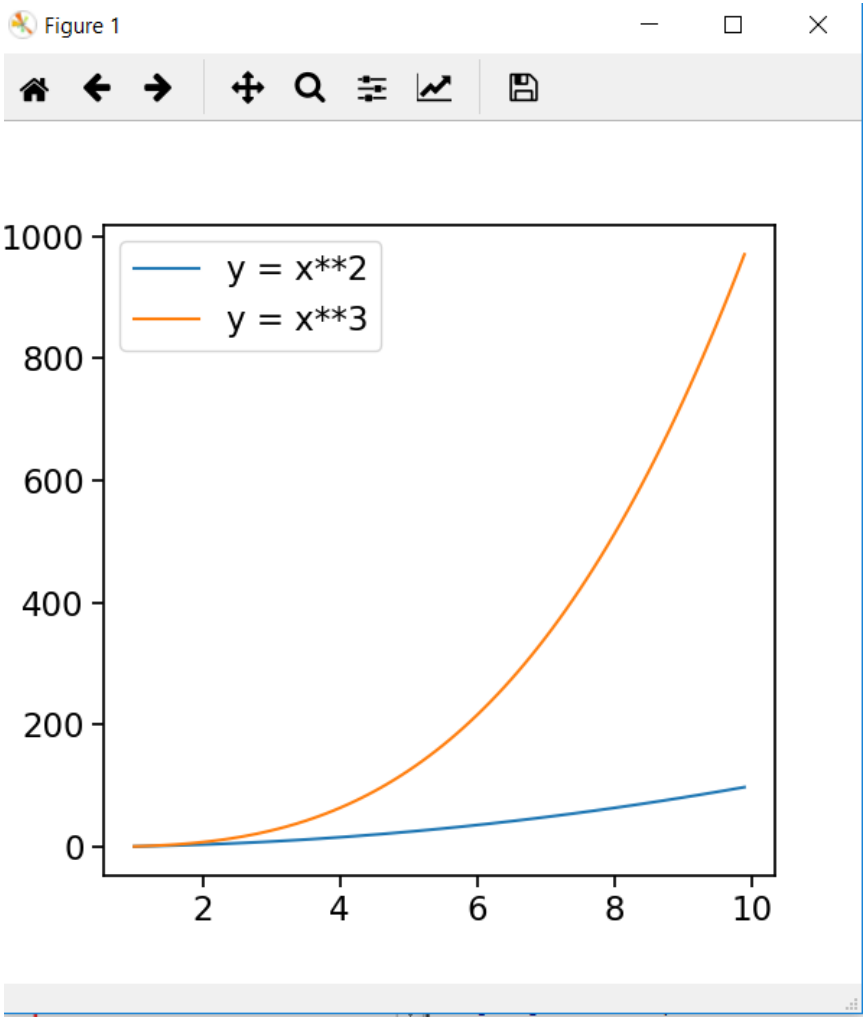
If possible, categories should be labelled directly rather than in a separate key. This allows your audience identify each category without having to decode a key.

Gridlines are not necessary to convey the graph message to the audience and only make the graph look cluttered.

For line graphs, the y-axis does not need to begin at 0. The y-axis scale should accomodate the data without excessive blank space and illustrate your key message without misrepresenting the data.

Line graph syntax in python

```
x=np.arange(1,10,0.1)


##plotting
plt.close("all")
fig = plt.figure(figsize=(6,6), dpi=100)
ax=fig.add_subplot(111)


ax.plot(x, x**2, label="y = x**2")
ax.plot(x, x**3, label="y = x**3")
ax.legend(loc=2); # upper left corner
```



Minimal syntax

Too many spines.

Too many numbers.

Isolated legend.

- Black and white.

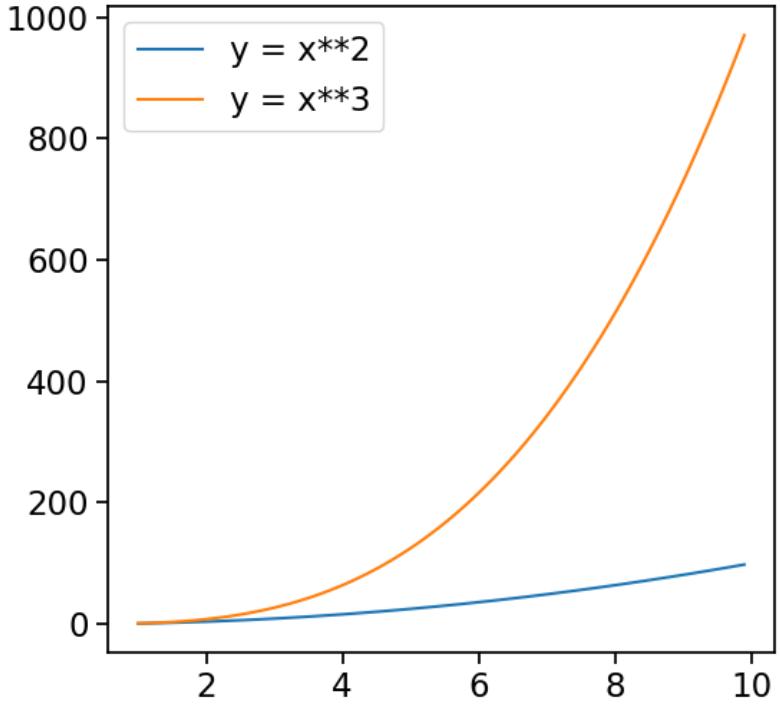
Data points missing

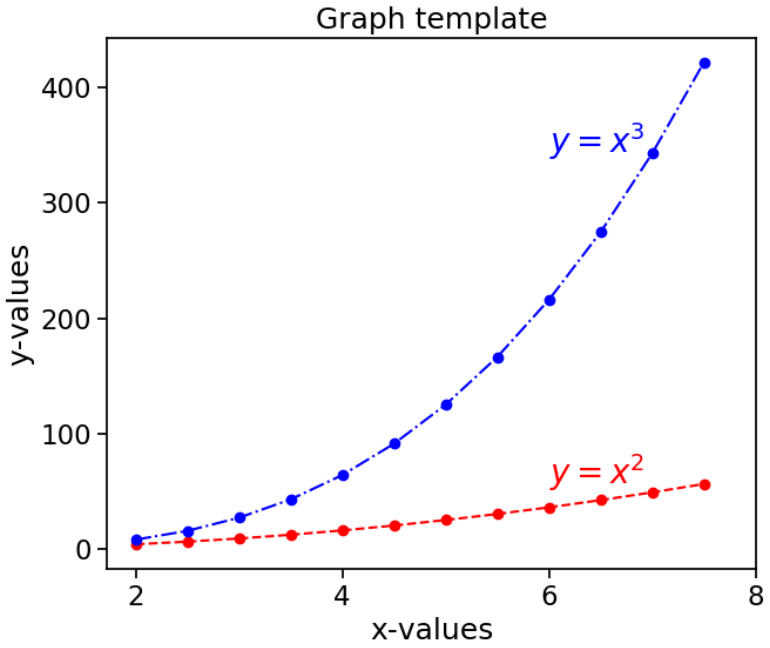
Missing labels

Scientific notation

```
x=np.arange(2,8,0.5)

##plotting
plt.close("all")
fig = plt.figure(figsize=(7,6), dpi=100)
ax=fig.add_subplot(111)

ax.plot(x, x**2,'r--o')
ax.plot(x, x**3, 'b-.o')

#labels
ax.set_xlabel('x-values')
ax.set_ylabel('y-values')
ax.set_title('Graph template')

#limits
#ax.set_ylim(bottom=0)
#ax.set_xlim(left=0)
```

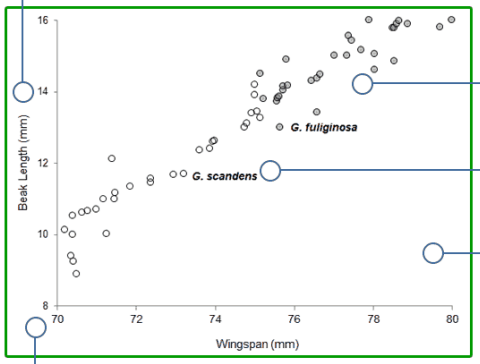
```
…

#ticks
ax.set_xticks(np.arange(min(x), max(x)+1, 2))

#annotations
ax.text(6,6**2+20, r"$y=x^2$", fontsize=20, color="red")
ax.text(7-1,7**3, r"$y=x^3$", fontsize=20, color="blue")

# Hide the right and top spines
#ax.spines['right'].set_visible(False)
#ax.spines['top'].set_visible(False)

#make sure labels dont overlap
plt.tight_layout()
```


y = x**2
y = x**3
1000
800
600
400
200
0
2
4
6
8
10


Graph template
$y = x^3$
$y = x^2$
y-values
x-values
400
300
200
100
0
2
4
6
8

Scatter plot

Identifying relationships between variables | Displaying trends and correlations



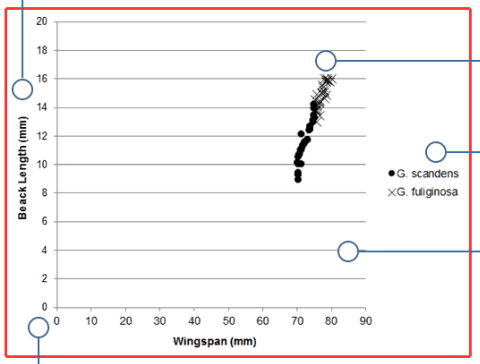
The axes are labelled in a standard font and include only enough value labels and tick marks to interpret the data.

The data maximises the available space. The distribution of data points will vary but you should aim to have your data spread across the graph space.

The categories are labelled directly so the audience can instantly identify the categories. Each category is distinguished by differences in shading.

The white background and lack of gridlines produces a clean and uncluttered graph.

The scale of both axes are appropriate for displaying the data in a meaningful way without an excess of blank space.



Axis labels don't need a bold font unless they could be confused with value labels. In this case a bold font is not necessary.

Data should not be bunched into a small region of the graph. If the data looks similar to this, the axis scales should be adjusted so the data is filling the graph space.

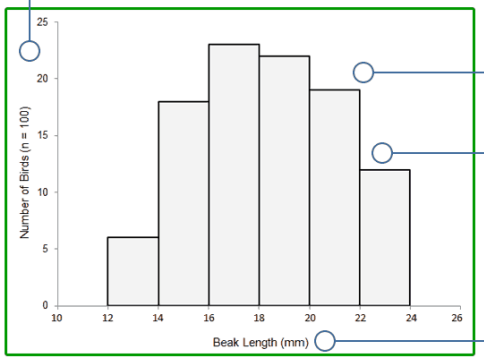
Categories in scatter plots can be labelled directly rather than using a key.This allows your audience to identify each category without having to decode a key.

Gridlines are not necessary to convey the graph message to the audience and only make the graph look cluttered.

For scatter plots, the x and y axes do not need to begin at 0. The scales should accomodate the data without excessive blank space and illustrate your key message without misrepresenting the data.

Histogram

Displaying distribution of continuous data | Useful for large data sets

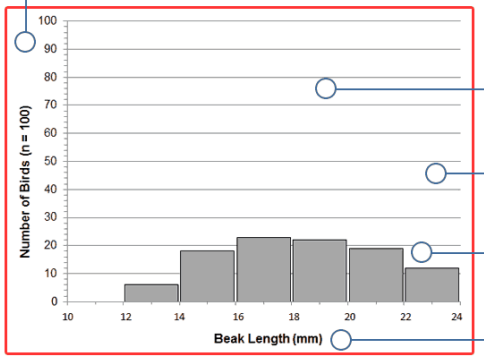


The y-axis scale begins at 0 and allows all the data to be seen without excessive blank space. Only the tick marks with a value are visible.

The white background and lack of gridlines produces a clean and uncluttered graph.

The vertical bars of the histogram are touching. The bars are lightly shaded to make the data stand out from the white background.

The axis labels are readable without using a bold font.



The y-axis includes too many tick marks. Graphs generally don't require the audience to determine exact values, therefore it is not necessary to provide marks for this number of values.

The scale of the y-axis has created a large amount of blank space. The scale on the axes should comfortably fit your data without any excess space.

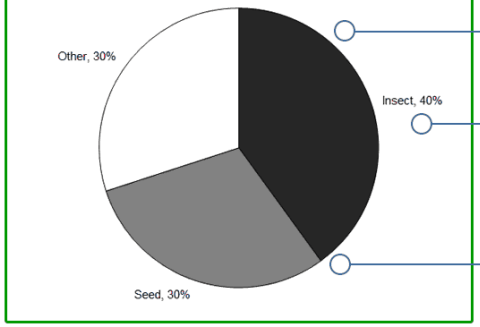
Gridlines are not necessary to convey the graph message to the audience and only make the graph look cluttered.

The vertical bars in histograms should not be separated by gaps.

The bold axis labels draw the eye away from the data. In this case it is not necessary to bold the axis labels to distinguish them from the axis values.

Pie Chart

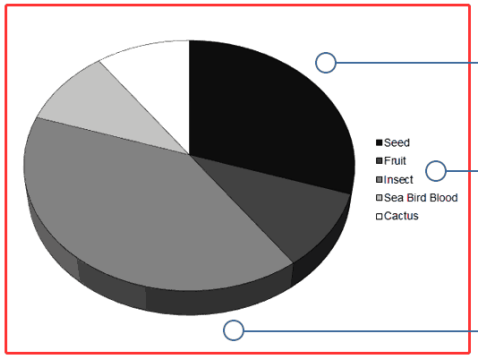
Comparing categorical data | Limited use for more than 4 categories



Three categories arranged from largest to smallest make this pie chart easy to interpret.

Pie chart categories should be labelled directly and include the percentage value to enable the audience to accurately interpret the chart.

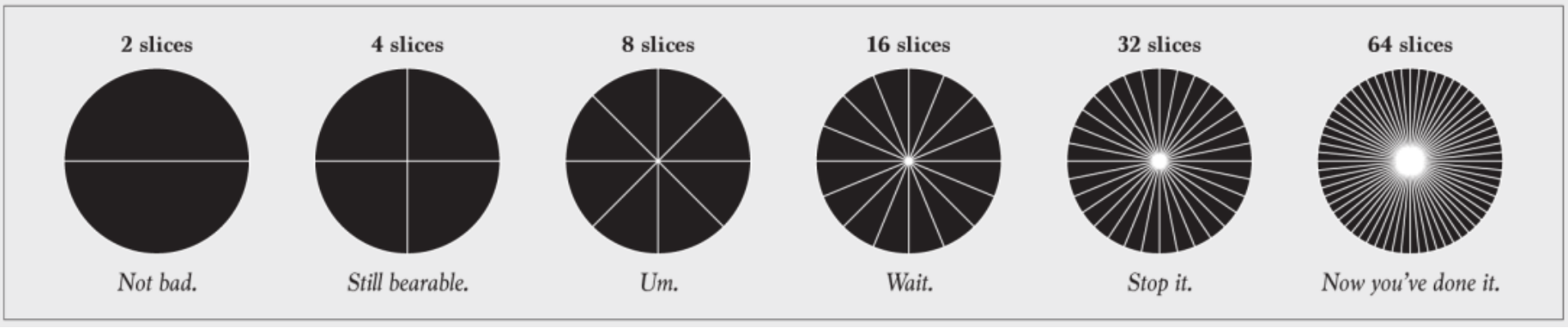
Pie charts should always be in a 2D format using distinct shading for each category. Never use patterns and only use colour for posters, presentations and websites.



Too many categories make this chart diffcult to interpret. The categories should be ordered from largest to smallest.

If possible, data series should be labelled directly rather than in a separate key. This allows your audience identify each data series without having to decode a key.

3D graphs should not be used. They may look fancy but they misrepresent the data.


2 slices
Not bad.
4 slices
Still bearable.
8 slices
Um.
16 slices
Wait.
32 slices
Stop it.
64 slices
Now you've done it.

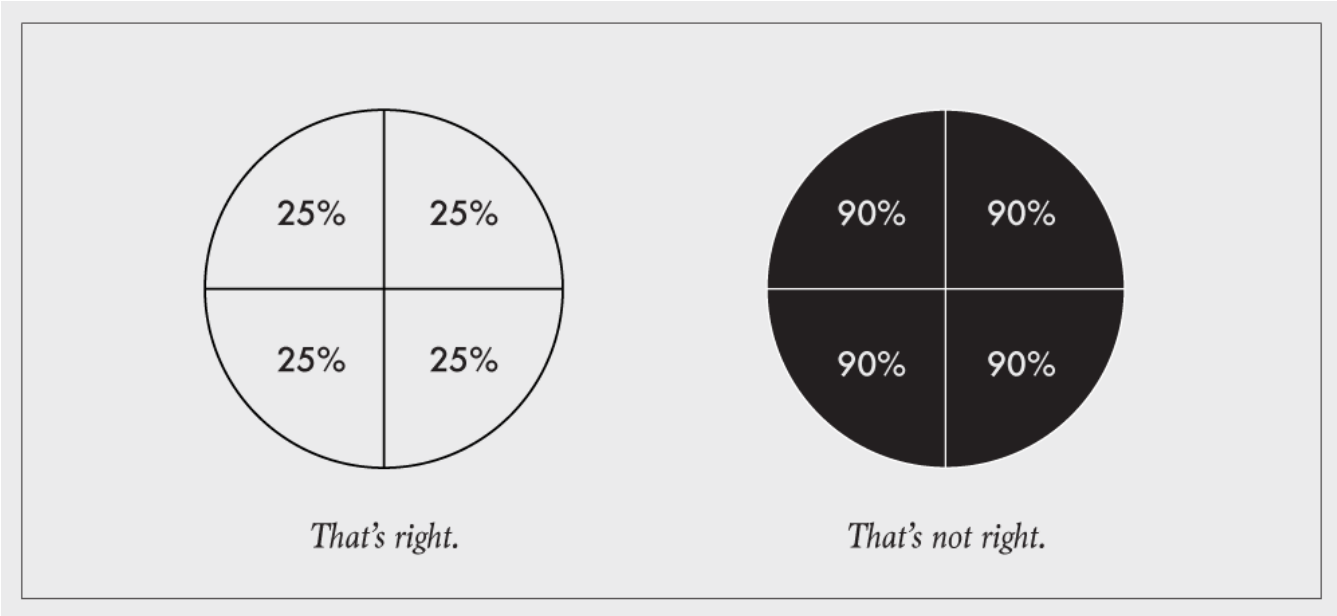
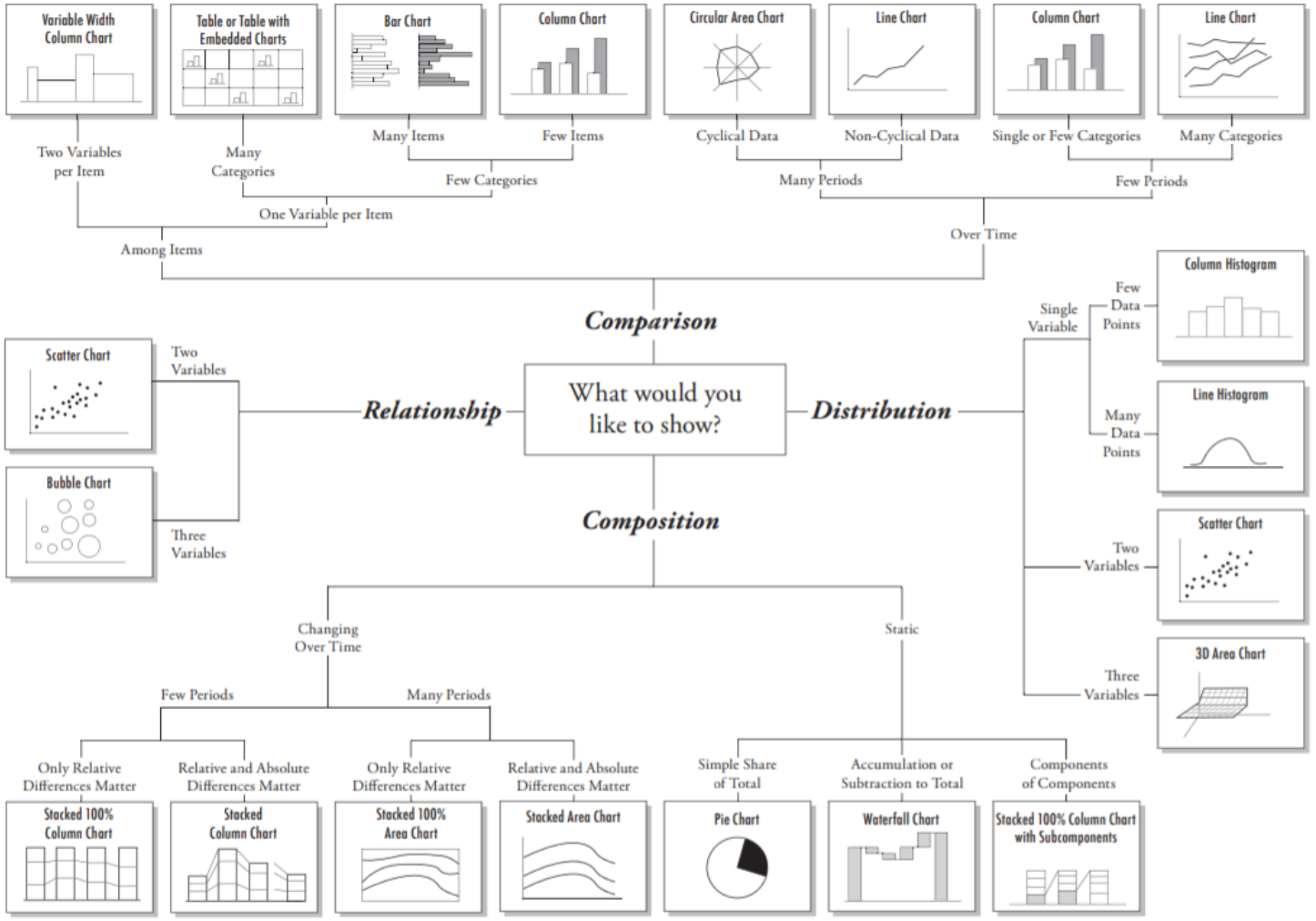

25%
25%
25%
25%
That's right.
90%
90%
90%
90%
That's not right.

Chart Suggestions—A Thought-Starter



HTTP://WWW.CLIPS.EDU.AU/WP-CONTENT/UPLOADS/CHOOSING-A-GOOD-GRAPH.PDF

How to speak MPL

Colornames:

- b: blue
- g: green
- r: red
- c: cyan
- m: magenta
- y: yellow
- k: black
- w: white

Full list: https://www.w3schools.com/Colors/colors_names.asp

Markers

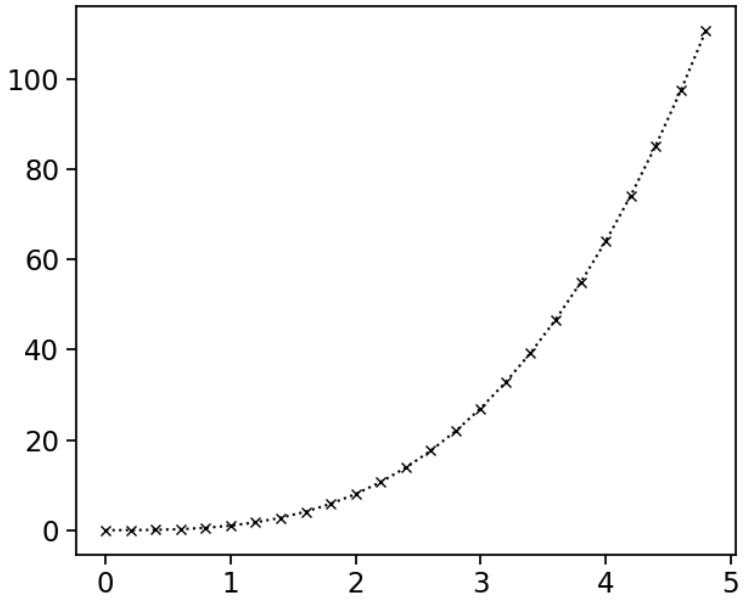
marker	description	marker	description	marker	description	marker	description
"."	point	"+"	plus	","	pixel	"x"	cross
"o"	circle	"D"	diamond	"d"	thin_diamond		
"8"	octagon	"s"	square	"p"	pentagon	"*"	star
"|"	vertical line	"_"	horizontal line	"h"	hexagon1	"H"	hexagon2
0	tickleft	4	caretleft	"<"	triangle_left	"3"	tri_left
1	tickright	5	caretright	">"	triangle_right	"4"	tri_right
2	tickup	6	caretup	"^"	triangle_up	"2"	tri_up
3	tickdown	7	caretdown	"v"	triangle_down	"1"	tri_down
"None"	nothing	None	default	" "	nothing	""	nothing

Linestyle

linestyle	description
'-'	solid
'--'	dashed
'-.'	dashdot
':'	dotted
'None'	draw nothing
' '	draw nothing
''	draw nothing

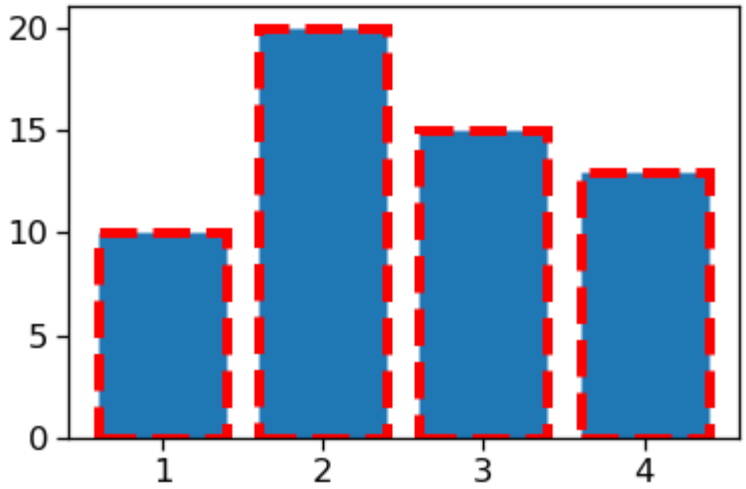
Example

```
ax=fig.add_subplot(111)
t = np.arange(0.0, 5.0, 0.2)
plt.plot( t, t**3,
          color='black',
          marker='x',
          linestyle=':')
plt.show()
```



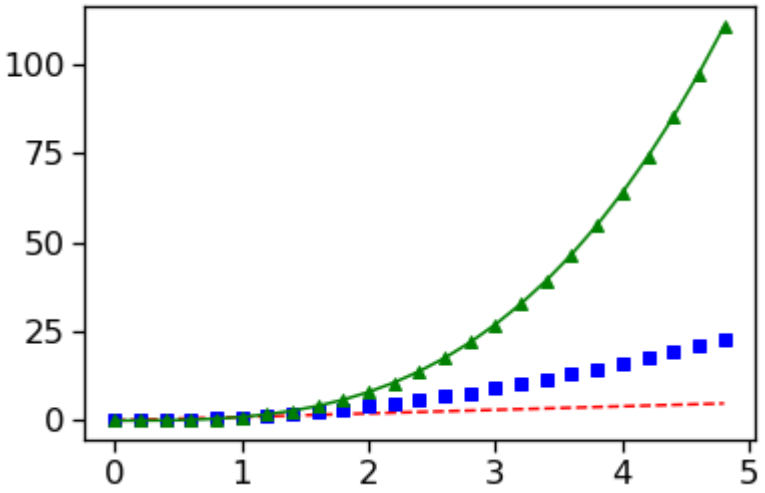
Style for other plots

```
fig, ax = plt.subplots(1, 1)
ax.bar([1, 2, 3, 4], [10, 20, 15, 13],
       linestyle='--', #linestyle
       ec='r', #color
       lw=5) #linewidth
plt.show()
```



Simple syntax, but too implicit!

```
fig, ax = plt.subplots(1, 1)
t = np.arange(0., 5., 0.2)
# red dashes, blue squares and green triangles
plt.plot(t, t, 'r--', t, t**2, 'bs', t, t**3, 'g^-')
plt.show()
```

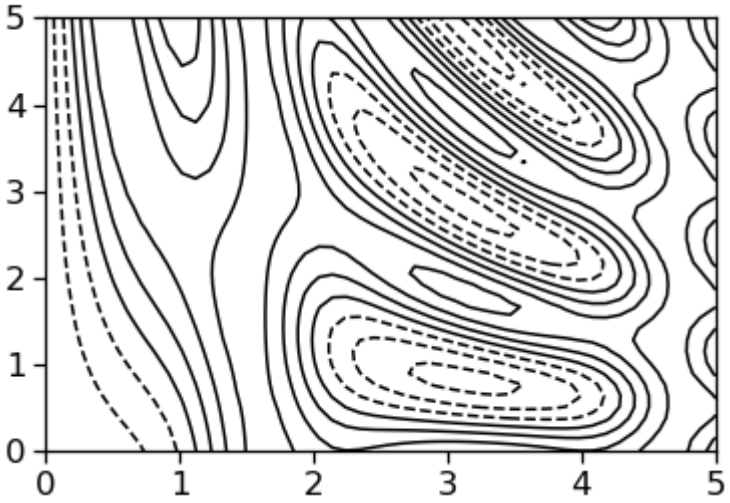


Property	Value Type
alpha	float
color or c	any matplotlib color
dash_capstyle	['butt', 'round' 'projecting']
dash_joinstyle	['miter' 'round' 'bevel']
dashes	sequence of on/off ink in points
drawstyle	[‘default’ ‘steps’ ‘steps-pre’
	‘steps-mid’ ‘steps-post’]
linestyle or ls	['-' '--' '-.' ':' 'None' ' ' '']
	and any drawstyle in combination with a
	linestyle, e.g. 'steps--'.
linewidth or lw	float value in points
marker	[0 1 2 3 4 5 6 7 'o' 'd' 'D' 'h' 'H'
	'' 'None' ' ' None '8' 'p' ','
	'+' 'x' '.' 's' '*' '_' '|'
	'1' '2' '3' '4' 'v' '<' '>' '^']
markeredgecolor or mec	any matplotlib color
markeredgewidth or mew	float value in points
markerfacecolor or mfc	any matplotlib color
markersize or ms	float
solid_capstyle	['butt' 'round' 'projecting']
solid_joinstyle	['miter' 'round' 'bevel']
visible	[True False]
zorder	any number

The 3rd dimension.

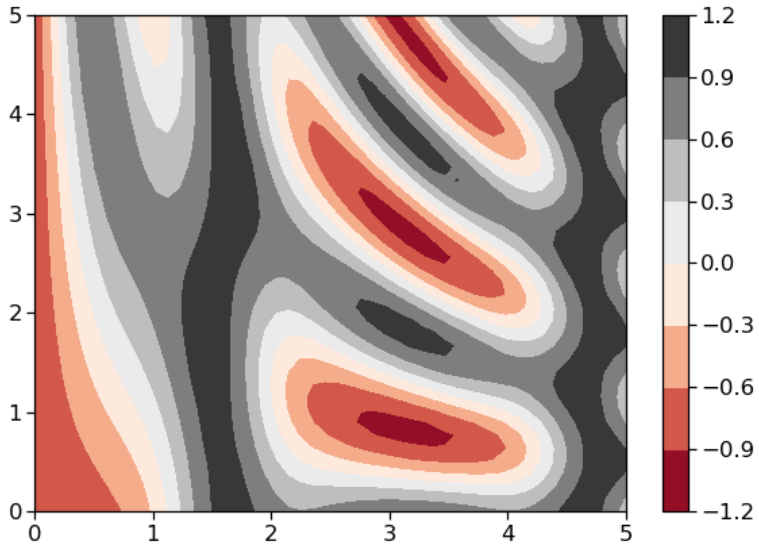
Contour plot

```
def f(x, y):
    return np.sin(x) ** 10 + np.cos(10 + y * x) * np.cos(x)

x = np.linspace(0, 5, 50)
y = np.linspace(0, 5, 40)

X, Y = np.meshgrid(x, y)
Z = f(X, Y)

plt.contour(X, Y, Z, colors='black');
```



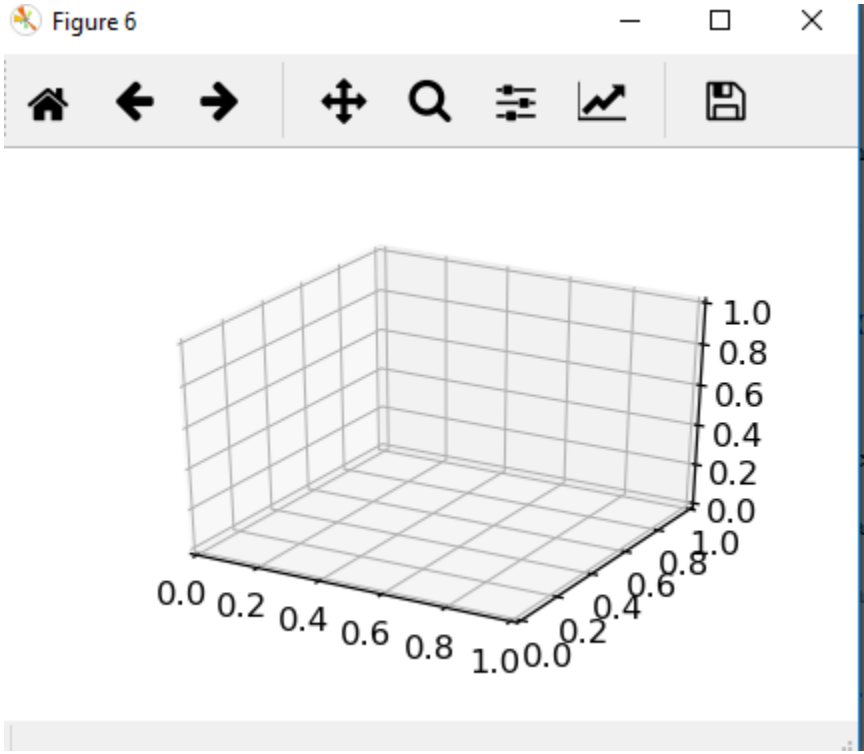
Colorful contour

```
def f(x, y):
    return np.sin(x) ** 10 + np.cos(10 + y * x) * np.cos(x)


x = np.linspace(0, 5, 50)
y = np.linspace(0, 5, 40)


X, Y = np.meshgrid(x, y)
Z = f(X, Y)
plt.contourf(X, Y, Z, cmap='RdGy');
plt.colorbar()
plt.tight_layout()
```

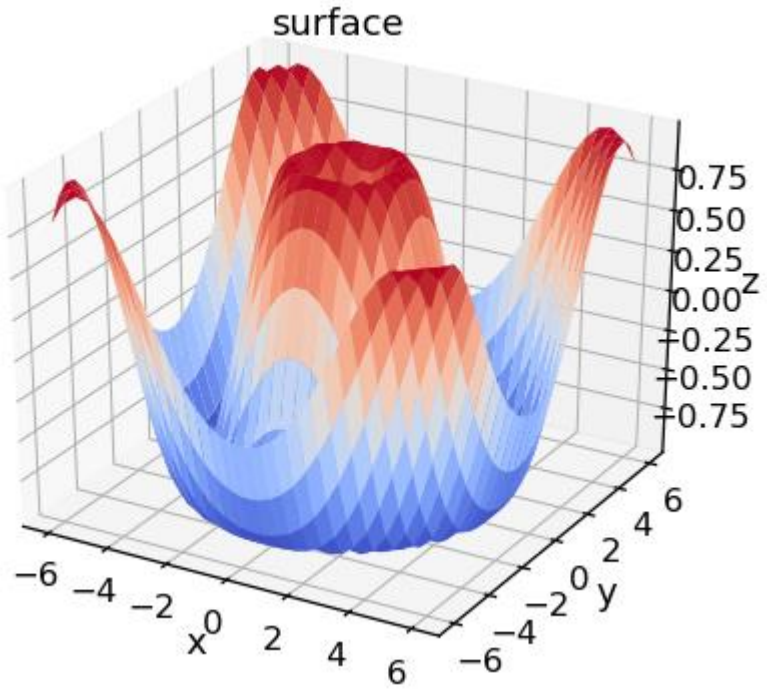


3D plotting

```
fig = plt.figure()
ax = fig.gca(projection="3d")
plt.show()
```



```
plt.close("all")
def f(x, y):
    return np.sin(np.sqrt(x ** 2 + y ** 2))

x = np.linspace(-6, 6, 30)
y = np.linspace(-6, 6, 30)

X, Y = np.meshgrid(x, y)
Z = f(X, Y)

fig = plt.figure()
ax = fig.gca(projection="3d")
ax.plot_surface(X, Y, Z,
                cmap='coolwarm')

ax.set_title('surface');
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('z');
plt.tight_layout()
plt.show()
```



Matplotlib backends

```
import matplotlib

matplotlib.get_backend()

matplotlib.use('Qt5Agg')
```

https://matplotlib.org/tutorials/introductory/usage.html

Backend	Description
Qt5Agg	Agg rendering in a Qt5 canvas (requires PyQt5). This backend can be activated in IPython with `%matplotlib qt5`.
ipympl	Agg rendering embedded in a Jupyter widget. (requires ipympl). This backend can be enabled in a Jupyter notebook with `%matplotlib ipympl`.
GTK3Agg	Agg rendering to a GTK 3.x canvas (requires PyGObject, and pycairo or cairocffi). This backend can be activated in IPython with `%matplotlib gtk3`.
macosx	Agg rendering into a Cocoa canvas in OSX. This backend can be activated in IPython with `%matplotlib osx`.
TkAgg	Agg rendering to a Tk canvas (requires TkInter). This backend can be activated in IPython with `%matplotlib tk`.
nbAgg	Embed an interactive figure in a Jupyter classic notebook. This backend can be enabled in Jupyter notebooks via `%matplotlib notebook`.
WebAgg	On `show()` will start a tornado server with an interactive figure.
GTK3Cairo	Cairo rendering to a GTK 3.x canvas (requires PyGObject, and pycairo or cairocffi).
Qt4Agg	Agg rendering to a Qt4 canvas (requires PyQt4 or `pyside`). This backend can be activated in IPython with `%matplotlib qt4`.
WXAgg	Agg rendering to a wxWidgets canvas (requires wxPython 4). This backend can be activated in IPython with `%matplotlib wx`.

Galleries

https://matplotlib.org/gallery.html

https://flowingdata.com/famous-movie-quotes-as-charts/

Pandas

Pandas

(**P**ython **an**d **D**ata **A**nalysi**s**)

https://pandas.pydata.org/pandas-docs/stable/getting_started/10min.html#min

A fast and efficient **DataFrame** object.

reading and writing data

Flexible **reshaping, slicing**, **fancy indexing**

Python with *pandas* is in use in a wide variety of **academic and commercial** domains, including Finance, Neuroscience, Economics, Statistics, Advertising, Web Analytics, and more.

```
import pandas as pd
```

Series

One dimensional fancy indexed arrays:

```
fruits = ['apples', 'oranges', 'cherries',
'pears']

quantities = [20, 33, 52, 10]

S = pd.Series(quantities, index=fruits)

S
```

```
apples      20
oranges     33
cherries    52
pears       10
dtype: int64
```

Add two series objects

```
fruits = ['apples', 'oranges', 'cherries',
'pears']

S = pd.Series([20, 33, 52, 10], index=fruits)

S2 = pd.Series([17, 13, 31, 32], index=fruits)

print(S + S2)

print("sum of S: ", sum(S))
```

```
apples      37
oranges     46
cherries    83
pears       42
dtype: int64
sum of S:  115
```

Accessing elements

```
print(S['apples'])
```

20

```
print(S[['apples', 'oranges', 'cherries']])
```

apples 20
oranges 33
cherries 52
dtype: int64

Operations

```
import numpy as np
```

```
S.apply(np.sin)
```

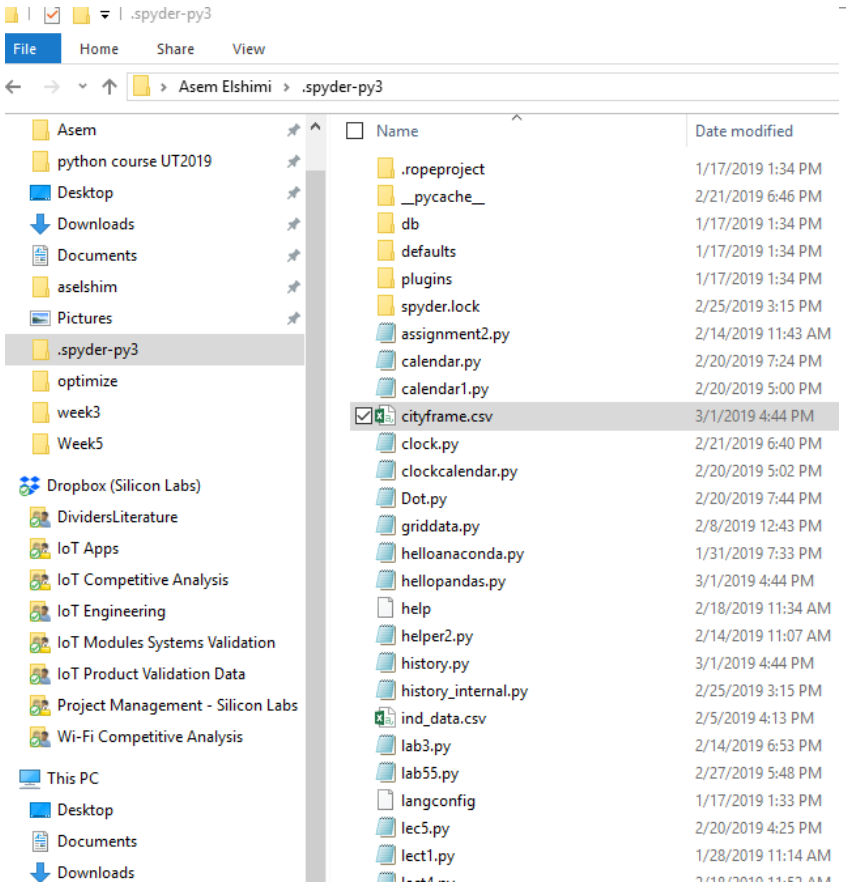
```
apples     30
oranges    43
cherries   52
pears      20
dtype: int64
```

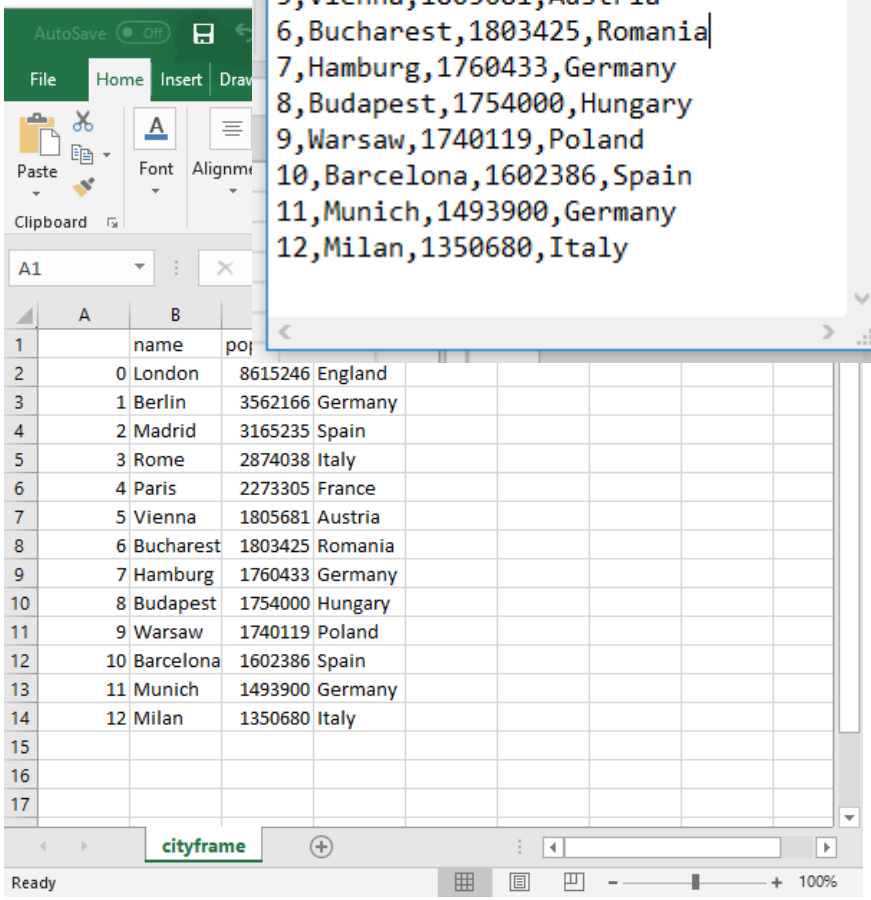
```
S.apply(lambda x: x if x > 50 else x+10 )
```

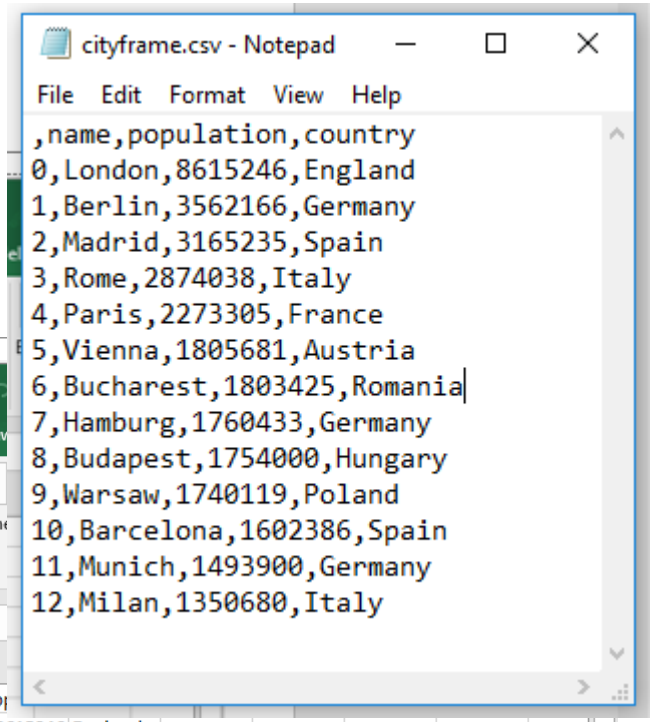
```
S[S>30] #boolean indexing
```

```
oranges    33
cherries   52
dtype: int64
```

.CSV


.spyder-py3
File
Home
Share
View
Asem Elshimi › .spyder-py3
Asem
python course UT2019
Desktop
Downloads
Documents
aselshim
Pictures
.spyder-py3
optimize
week3
Week5
Dropbox (Silicon Labs)
DividersLiterature
IoT Apps
IoT Competitive Analysis
IoT Engineering
IoT Modules Systems Validation
IoT Product Validation Data
Project Management - Silicon Labs
Wi-Fi Competitive Analysis
This PC
Desktop
Documents
Downloads
Name
Date modified
.ropeproject 1/17/2019 1:34 PM
__pycache__ 2/21/2019 6:46 PM
db 1/17/2019 1:34 PM
defaults 1/17/2019 1:34 PM
plugins 1/17/2019 1:34 PM
spyder.lock 2/25/2019 3:15 PM
assignment2.py 2/14/2019 11:43 AM
calendar.py 2/20/2019 7:24 PM
calendar1.py 2/20/2019 5:00 PM
cityframe.csv 3/1/2019 4:44 PM
clock.py 2/21/2019 6:40 PM
clockcalendar.py 2/20/2019 5:02 PM
Dot.py 2/20/2019 7:44 PM
griddata.py 2/8/2019 12:43 PM
helloanaconda.py 1/31/2019 7:33 PM
hellopandas.py 3/1/2019 4:44 PM
help 2/18/2019 11:34 AM
helper2.py 2/14/2019 11:07 AM
history.py 3/1/2019 4:44 PM
history_internal.py 2/25/2019 3:15 PM
ind_data.csv 2/5/2019 4:13 PM
lab3.py 2/14/2019 6:53 PM
lab55.py 2/27/2019 5:48 PM
langconfig 1/17/2019 1:33 PM
lec5.py 2/20/2019 4:25 PM
lect1.py 1/28/2019 11:14 AM


AutoSave
File
Home
Insert
Paste
Font
Clipboard
A1
A B
1 name
2 0 London 8615246 England
3 1 Berlin 3562166 Germany
4 2 Madrid 3165235 Spain
5 3 Rome 2874038 Italy
6 4 Paris 2273305 France
7 5 Vienna 1805681 Austria
8 6 Bucharest 1803425 Romania
9 7 Hamburg 1760433 Germany
10 8 Budapest 1754000 Hungary
11 9 Warsaw 1740119 Poland
12 10 Barcelona 1602386 Spain
13 11 Munich 1493900 Germany
14 12 Milan 1350680 Italy
15
16
17
cityframe
Ready
100%


cityframe.csv - Notepad
File Edit Format View Help
,name,population,country
0,London,8615246,England
1,Berlin,3562166,Germany
2,Madrid,3165235,Spain
3,Rome,2874038,Italy
4,Paris,2273305,France
5,Vienna,1805681,Austria
6,Bucharest,1803425,Romania
7,Hamburg,1760433,Germany
8,Budapest,1754000,Hungary
9,Warsaw,1740119,Poland
10,Barcelona,1602386,Spain
11,Munich,1493900,Germany
12,Milan,1350680,Italy

DataFrame

```
city_frame=pd.read_csv("cityframe.csv")
print(city_frame)
print(type(city_frame))

city_frame.head()
```

	name	population	country
0	London	8615246	England
1	Berlin	3562166	Germany
2	Madrid	3165235	Spain
3	Rome	2874038	Italy
4	Paris	2273305	France
5	Vienna	1805681	Austria
6	Bucharest	1803425	Romania
7	Hamburg	1760433	Germany
8	Budapest	1754000	Hungary
9	Warsaw	1740119	Poland
10	Barcelona	1602386	Spain
11	Munich	1493900	Germany
12	Milan	1350680	Italy

<class 'pandas.core.frame.DataFrame'>

	name	population	country
0	London	8615246	England
1	Berlin	3562166	Germany
2	Madrid	3165235	Spain
3	Rome	2874038	Italy
4	Paris	2273305	France

Fancy index

```
#Fancy index

city_frame.set_index("country", inplace=True)
```

country	name	population
England	London	8615246
Germany	Berlin	3562166
Spain	Madrid	3165235
Italy	Rome	2874038
France	Paris	2273305
Austria	Vienna	1805681
Romania	Bucharest	1803425
Germany	Hamburg	1760433
Hungary	Budapest	1754000
Poland	Warsaw	1740119
Spain	Barcelona	1602386
Germany	Munich	1493900
Italy	Milan	1350680

Accessing rows and colns

```
#access coln
city_frame['population']
type(city_frame['population']) #series is 1D dataframe
#for multiple indicies, pass a list:
city_frame[['population','name']]
type(city_frame[['population','name']])
```

```
country
          population       name
country
England     8615246     London
Germany     3562166     Berlin
Spain       3165235     Madrid
Italy       2874038       Rome
```

```
#access row
city_frame.loc["Germany"]
type(city_frame.loc["Germany"])
#for multiple indicies, pass a list:
print(city_frame.loc[["Germany",'Franc
```

```
            name  population
country
Germany   Berlin     3562166
Germany  Hamburg     1760433
Ge
```

```
            name  population
country
Germany   Berlin     3562166
Germany  Hamburg     1760433
Germany   Munich     1493900
France     Paris     2273305
```

```
ame'>
a
h
n
aFrame'>
```

Q: How to access a single element of a dataframe?

Adding new coln

```
area = [1572, 891.85, 605.77, 1285,
        105.4, 414.6, 228, 755,
        525.2, 517, 101.9, 310.4,
        181.8]
# area could have been designed as a list,
#a Series, an array or a scalar
city_frame["area"] = area
```

city_frame.head()

country	name	population	area
England	London	8615246	1572.00
Germany	Berlin	3562166	891.85
Spain	Madrid	3165235	605.77
Italy	Rome	2874038	1285.00
France	Paris	2273305	105.40

World population

LIVE EXAMPLE

Hierarchical indicies

```
shop1 = {"foo":{2010:23, 2011:25}, "bar":{2010:13, 2011:29}}

shop2 = {"foo":{2010:223, 2011:225}, "bar":{2010:213, 2011:229}}

shop1 = pd.DataFrame(shop1)

shop2 = pd.DataFrame(shop2)

both_shops = shop1 + shop2
```

```
shops = pd.concat([shop1, shop2], keys=["one", "two"])

shops.swaplevel()

shops.swaplevel().sort_index()
```

World population

BACK TO EXAMPLE

Useful methods

```
df.T

df.describe()

df.to_numpy()

df.tail(3)

df.sort_values(by='B')


df[df > 0]

df.apply(np.cumsum)
```

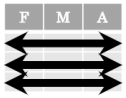
Data Wrangling with pandas Cheat Sheet http://pandas.pydata.org

Tidy Data – A foundation for wrangling in pandas

In a tidy data set:

Each **variable** is saved in its own **column**

&



Each **observation** is saved in its own **row**

Tidy data complements pandas's **vectorized operations**. pandas will automatically preserve observations as you manipulate variables. No other format works as intuitively with pandas.

M * A

Syntax – Creating DataFrames

	a	b	c
1	4	7	10
2	5	8	11
3	6	9	12

```
df = pd.DataFrame(
          {"a" : [4 ,5, 6],
           "b" : [7, 8, 9],
           "c" : [10, 11, 12]},
         index = [1, 2, 3])
```

Specify values for each column.

```
df = pd.DataFrame(
     [[4, 7, 10],
      [5, 8, 11],
      [6, 9, 12]],
     index=[1, 2, 3],
     columns=['a', 'b', 'c'])
```

Specify values for each row.

		a	b	c
n	v			
d	1	4	7	10
	2	5	8	11
e	2	6	9	12

```
df = pd.DataFrame(
          {"a" : [4 ,5, 6],
           "b" : [7, 8, 9],
           "c" : [10, 11, 12]},
index = pd.MultiIndex.from_tuples(
          [('d',1),('d',2),('e',2)],
            names=['n','v']))
```

Create DataFrame with a MultiIndex

Method Chaining

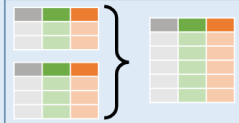
Most pandas methods return a DataFrame so that another pandas method can be applied to the result. This improves readability of code.

```
df = (pd.melt(df)
        .rename(columns={
                'variable' : 'var',
                'value' : 'val'})
        .query('val >= 200')
     )
```

Reshaping Data – Change the layout of a data set

pd.melt(df)
Gather columns into rows.

df.pivot(columns='var', values='val')
Spread rows into columns.



pd.concat([df1,df2])
Append rows of DataFrames

pd.concat([df1,df2], axis=1)
Append columns of DataFrames

df.sort_values('mpg')
Order rows by values of a column (low to high).

df.sort_values('mpg',ascending=False)
Order rows by values of a column (high to low).

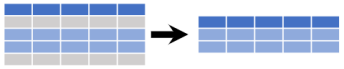
df.rename(columns = {'y':'year'})
Rename the columns of a DataFrame

df.sort_index()
Sort the index of a DataFrame

df.reset_index()
Reset index of DataFrame to row numbers, moving index to columns.

df.drop(columns=['Length','Height'])
Drop columns from DataFrame

Subset Observations (Rows)



df[df.Length > 7]
Extract rows that meet logical criteria.

df.drop_duplicates()
Remove duplicate rows (only considers columns).

df.head(n)
Select first n rows.

df.tail(n)
Select last n rows.

df.sample(frac=0.5)
Randomly select fraction of rows.

df.sample(n=10)
Randomly select n rows.

df.iloc[10:20]
Select rows by position.

df.nlargest(n, 'value')
Select and order top n entries.

df.nsmallest(n, 'value')
Select and order bottom n entries.

Logic in Python (and pandas)			
<	Less than	!=	Not equal to
>	Greater than	df.column.isin(values)	Group membership
==	Equals	pd.isnull(obj)	Is NaN
<=	Less than or equals	pd.notnull(obj)	Is not NaN
>=	Greater than or equals	&,|,~,^,df.any(),df.all()	Logical and, or, not, xor, any, all

Subset Variables (Columns)

df[['width','length','species']]
Select multiple columns with specific names.

df['width'] *or* **df.width**
Select single column with specific name.

df.filter(regex='*regex*')
Select columns whose name matches regular expression *regex*.

regex (Regular Expressions) Examples	
'\.'	Matches strings containing a period '.'
'Length$'	Matches strings ending with word 'Length'
'^Sepal'	Matches strings beginning with the word 'Sepal'
'^x[1-5]$'	Matches strings beginning with 'x' and ending with 1,2,3,4,5
'^(?!Species$).*'	Matches strings except the string 'Species'

df.loc[:,'x2':'x4']
Select all columns between x2 and x4 (inclusive).

df.iloc[:,[1,2,5]]
Select columns in positions 1, 2 and 5 (first column is 0).

df.loc[df['a'] > 10, ['a','c']]
Select rows meeting logical condition, and only the specific columns .

http://pandas.pydata.org/ This cheat sheet inspired by Rstudio Data Wrangling Cheatsheet (https://www.rstudio.com/wp-content/uploads/2015/02/data-wrangling-cheatsheet.pdf) Written by Irv Lustig, Princeton Consultants

Summarize Data

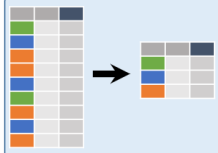
`df['w'].value_counts()`
Count number of rows with each unique value of variable
`len(df)`
# of rows in DataFrame.
`df['w'].nunique()`
# of distinct values in a column.
`df.describe()`
Basic descriptive statistics for each column (or GroupBy)

pandas provides a large set of **summary functions** that operate on different kinds of pandas objects (DataFrame columns, Series, GroupBy, Expanding and Rolling (see below)) and produce single values for each of the groups. When applied to a DataFrame, the result is returned as a pandas Series for each column. Examples:

`sum()`
Sum values of each object.
`count()`
Count non-NA/null values of each object.
`median()`
Median value of each object.
`quantile([0.25,0.75])`
Quantiles of each object.
`apply(function)`
Apply function to each object.

`min()`
Minimum value in each object.
`max()`
Maximum value in each object.
`mean()`
Mean value of each object.
`var()`
Variance of each object.
`std()`
Standard deviation of each object.

Group Data



`df.groupby(by="col")`
Return a GroupBy object, grouped by values in column named "col".

`df.groupby(level="ind")`
Return a GroupBy object, grouped by values in index level named "ind".

All of the summary functions listed above can be applied to a group. Additional GroupBy functions:

`size()`
Size of each group.

`agg(function)`
Aggregate group using function.

The examples below can also be applied to groups. In this case, the function is applied on a per-group basis, and the returned vectors are of the length of the original DataFrame.

`shift(1)`
Copy with values shifted by 1.
`rank(method='dense')`
Ranks with no gaps.
`rank(method='min')`
Ranks. Ties get min rank.
`rank(pct=True)`
Ranks rescaled to interval [0, 1].
`rank(method='first')`
Ranks. Ties go to first value.

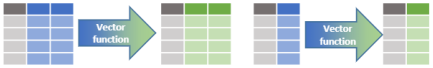
`shift(-1)`
Copy with values lagged by 1.
`cumsum()`
Cumulative sum.
`cummax()`
Cumulative max.
`cummin()`
Cumulative min.
`cumprod()`
Cumulative product.

Handling Missing Data

`df.dropna()`
Drop rows with any column having NA/null data.
`df.fillna(value)`
Replace all NA/null data with value.

Make New Columns

`df.assign(Area=lambda df: df.Length*df.Height)`
Compute and append one or more new columns.
`df['Volume'] = df.Length*df.Height*df.Depth`
Add single column.
`pd.qcut(df.col, n, labels=False)`
Bin column into n buckets.



pandas provides a large set of **vector functions** that operate on all columns of a DataFrame or a single selected column (a pandas Series). These functions produce vectors of values for each of the columns, or a single Series for the individual Series. Examples:

`max(axis=1)`
Element-wise max.
`clip(lower=-10,upper=10)`
Trim values at input thresholds

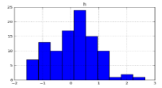
`min(axis=1)`
Element-wise min.
`abs()`
Absolute value.

Windows

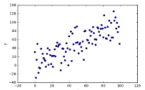
`df.expanding()`
Return an Expanding object allowing summary functions to be applied cumulatively.
`df.rolling(n)`
Return a Rolling object allowing summary functions to be applied to windows of length n.

Plotting

`df.plot.hist()`
Histogram for each column



`df.plot.scatter(x='w',y='h')`
Scatter chart using pairs of points



Combine Data Sets

adf

x1	x2
A	1
B	2
C	3

bdf

x1	x3
A	T
B	F
D	T

Standard Joins

x1	x2	x3
A	1	T
B	2	F
C	3	NaN

`pd.merge(adf, bdf, how='left', on='x1')`
Join matching rows from bdf to adf.

x1	x2	x3
A	1.0	T
B	2.0	F
D	NaN	T

`pd.merge(adf, bdf, how='right', on='x1')`
Join matching rows from adf to bdf.

x1	x2	x3
A	1	T
B	2	F

`pd.merge(adf, bdf, how='inner', on='x1')`
Join data. Retain only rows in both sets.

x1	x2	x3
A	1	T
B	2	F
C	3	NaN
D	NaN	T

`pd.merge(adf, bdf, how='outer', on='x1')`
Join data. Retain all values, all rows.

Filtering Joins

x1	x2
A	1
B	2

`adf[adf.x1.isin(bdf.x1)]`
All rows in adf that have a match in bdf.

x1	x2
C	3

`adf[~adf.x1.isin(bdf.x1)]`
All rows in adf that do not have a match in bdf.

ydf

x1	x2
A	1
B	2
C	3

zdf

x1	x2
B	2
C	3
D	4

Set-like Operations

x1	x2
B	2
C	3

`pd.merge(ydf, zdf)`
Rows that appear in both ydf and zdf (Intersection).

x1	x2
A	1
B	2
C	3
D	4

`pd.merge(ydf, zdf, how='outer')`
Rows that appear in either or both ydf and zdf (Union).

x1	x2
A	1

```
pd.merge(ydf, zdf, how='outer',
         indicator=True)
.query('_merge == "left_only"')
.drop(columns=['_merge'])
```
Rows that appear in ydf but not zdf (Setdiff).

http://pandas.pydata.org/ This cheat sheet inspired by Rstudio Data Wrangling Cheatsheet (https://www.rstudio.com/wp-content/uploads/2015/02/data-wrangling-cheatsheet.pdf) Written by Irv Lustig, Princeton Consultants

Pandas vs excel!

Analyze large datasets:

- Excel is sluggish at 10000 rows

More high level functions.

More file formats: CSV, HTML, SQL.

Automated procedures.

Co-existence!