

# Introduction to Data Science

## Data Science Essentials

---



# Goals for today

- Review last session coding tasks
- Learn ways to find help (module API, Stack Overflow, etc)
- Merging in *pandas*



# Errors happen!!

- Opportunity to gain knowledge
- Read the output (stack trace) from the bottom up to see the error and what caused it
- If you have no idea what is causing the error, try copying it, and pasting to a Google search

---

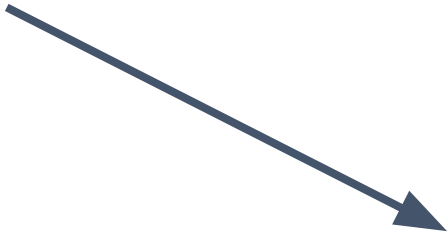
```
In [2]: x + 2|
```

```
-----  
NameError                                Traceback (most recent call last)  
<ipython-input-2-2c7a9192c558> in <module>()  
----> 1 x + 2  
  
NameError: name 'x' is not defined
```

---

Often when you get an error, you will get a long stack trace.

Start looking down here



```
In [34]: df.rename(lambda x: x.upper)
```

```
.....
AttributeError                                Traceback (most recent call last)
<ipython-input-34-18dc4f7801aa> in <module>()
----> 1 df.rename(lambda x: x.upper)

~/anaconda3/lib/python3.6/site-packages/pandas/util/_decorators.py in wrapper(*args, **kwargs)
    195     @wraps(func)
    196     def wrapper(*args, **kwargs):
--> 197         return func(*args, **kwargs)
    198
    199     if not PY2:

~/anaconda3/lib/python3.6/site-packages/pandas/core/frame.py in rename(self, *args, **kwargs)
    4023         kwargs.pop('axis', None)
    4024         kwargs.pop('mapper', None)
-> 4025         return super(DataFrame, self).rename(**kwargs)
    4026
    4027     @Substitution(**_shared_doc_kwargs)

~/anaconda3/lib/python3.6/site-packages/pandas/core/generic.py in rename(self, *args, **kwargs)
    1089         level = self.axes[axis].get_level_number(level)
    1090         result._data = result._data.rename_axis(f, axis=baxis, copy=copy,
-> 1091                                                level=level)
    1092         result._clear_item_cache()
    1093

~/anaconda3/lib/python3.6/site-packages/pandas/core/internals/managers.py in rename_axis(self, mapper, axis, copy,
level)
    169     """
    170     obj = self.copy(deep=copy)
--> 171     obj.set_axis(axis, _transform_index(self.axes[axis], mapper, level))
    172     return obj
    173

~/anaconda3/lib/python3.6/site-packages/pandas/core/internals/managers.py in _transform_index(index, func, level)
    2002     return MultiIndex.from_tuples(items, names=index.names)
    2003     else:
-> 2004         items = [func(x) for x in index]
    2005         return Index(items, name=index.name, tupleize_cols=False)
    2006

~/anaconda3/lib/python3.6/site-packages/pandas/core/internals/managers.py in <listcomp>(.0)
    2002     return MultiIndex.from_tuples(items, names=index.names)
    2003     else:
-> 2004         items = [func(x) for x in index]
    2005         return Index(items, name=index.name, tupleize_cols=False)
    2006

<ipython-input-34-18dc4f7801aa> in <lambda>(x)
----> 1 df.rename(lambda x: x.upper)

AttributeError: 'int' object has no attribute 'upper'
```



- **Be as specific as you can: search for python + package + what you are trying to do.**
- **Copy the error from Jupyter and paste it right in the search box**
- **Pay attention to the dates of results - sometimes blog posts, etc. are outdated**
- **If you're not sure what text to use try asking your question exactly like you would ask another person!**



# Stack Overflow

<https://stackoverflow.com/>

- Question is at the top
- Answers underneath
- Pay attention to upvotes and the selected answer

**Example:**

<https://stackoverflow.com/questions/11346283/rename-columns-in-pandas>

# Package API Reference

<http://pandas.pydata.org/pandas-docs/stable/api.html>

- Can be harder to read when you are first learning
- Get into all of the nitty-gritty. You can even view the source code if you want. This can be powerful as your knowledge of Python grows.

## Example:

<https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.rename.html>

# Help within Jupyter

- shift + tab after keyword in a Jupyter cell
- ? + keyword in a Jupyter cell

In [26]: `pd.concat?`

**Signature:** `pd.concat(objs, axis=0, join='outer', join_axes=None, ignore_index=False, keys=None, levels=None, names=None, verify_integrity=False, sort=None, copy=True)`

**Docstring:**

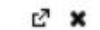
Concatenate pandas objects along a particular axis with optional set logic along the other axes.

Can also add a layer of hierarchical indexing on the concatenation axis, which may be useful if the labels are the same (or overlapping) on the passed axis number.

**Parameters**

-----

**objs** : a sequence or mapping of Series, DataFrame, or Panel objects  
If a dict is passed, the sorted keys will be used as the `keys` argument, unless it is passed, in which case the values will be selected (see below). Any None objects will be dropped silently unless they are all None in which case a ValueError will be raised



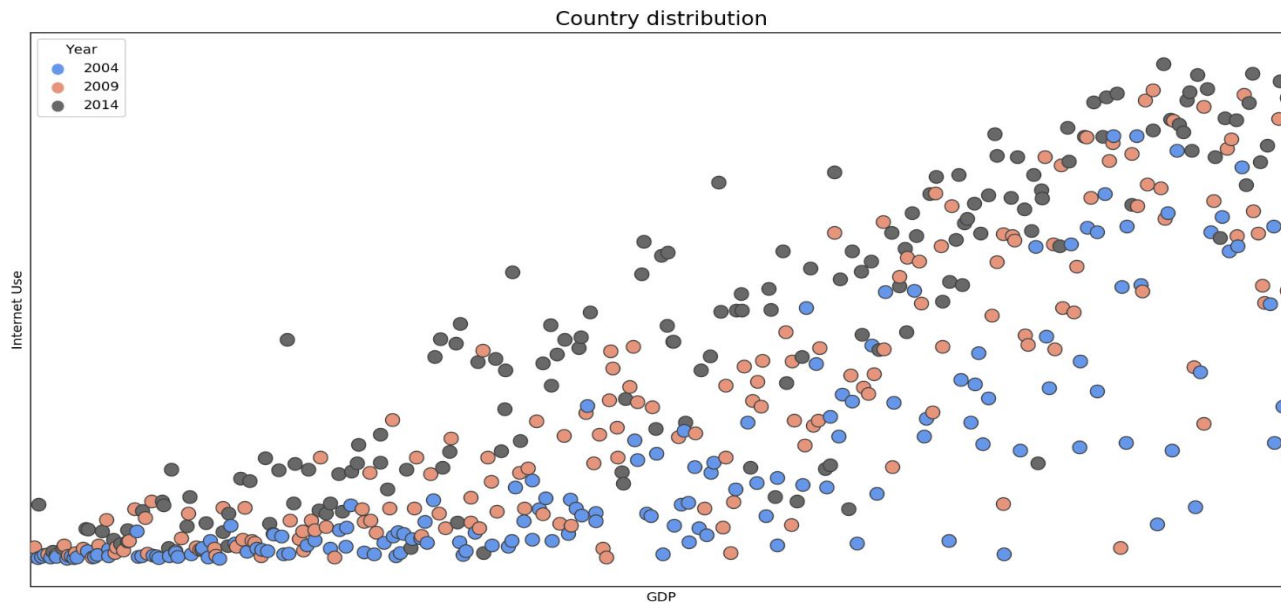


# Annotate Your Work with Markdown

## Country GDP and internet usage distributions

Plotting of Year with x-axis as GDP\_Per\_Capita and y-axis as Internet\_Users\_Pct.

```
In [8]: plt.figure(figsize=(16,8), clear=True);  
  
ax1 = sns.stripplot(x="GDP_Per_Capita",  
                   y="Internet_Users_Pct",  
                   data=df_04_09_14,  
                   jitter=2,  
                   hue="Year",  
                   size=10,  
                   linewidth=.8,  
                   dodge=True);  
  
ax1.set_xlabel('GDP');  
ax1.set_ylabel('Internet Use');  
ax1.set_yticks([]);  
ax1.set_xticks([]);  
ax1.axes.set_title('Country distribution', fontsize=15);
```



Observing the plot ax1 above, we notice that in general, there looks to be a positive correlation between GDP and internet usage. This correlation seems strongest in years 2009 and 2014.

- Comment on choices made
- Comment on trends observed
- Note anomalies/surprises

<https://www.markdownguide.org/cheat-sheet/>



pandas – <https://pandas.pydata.org/pandas-docs/stable/api.html>

- **String Methods:**
  - **.str.lower()** - convert a column to lowercase
  - **.str.upper()** - convert a column to uppercase
  - **.str.split()** - divide a string column into a list by specifying a delimiter character
  - **.str.replace()** - replace each instance of a string with a different string
- **df.describe() and series.describe()** – returns statistical info (count, mean, sd, quartiles)
- **pd.merge()** - Combines two DataFrames by joining along one or more columns



## Merging two DataFrames

***pd.merge***(<df1>, <df2>, **on** = <col or list of cols to join on>, **how** = <join\_type>)

left				right			
	key	A	B		key	C	D
0	K0	A0	B0	0	K0	C0	D0
1	K1	A1	B1	1	K1	C1	D1
2	K2	A2	B2	2	K2	C2	D2
3	K3	A3	B3	3	K3	C3	D3

- Need one or more “key” columns to join on
- Pastes matching rows together along the key column(s)

## Merging two DataFrames

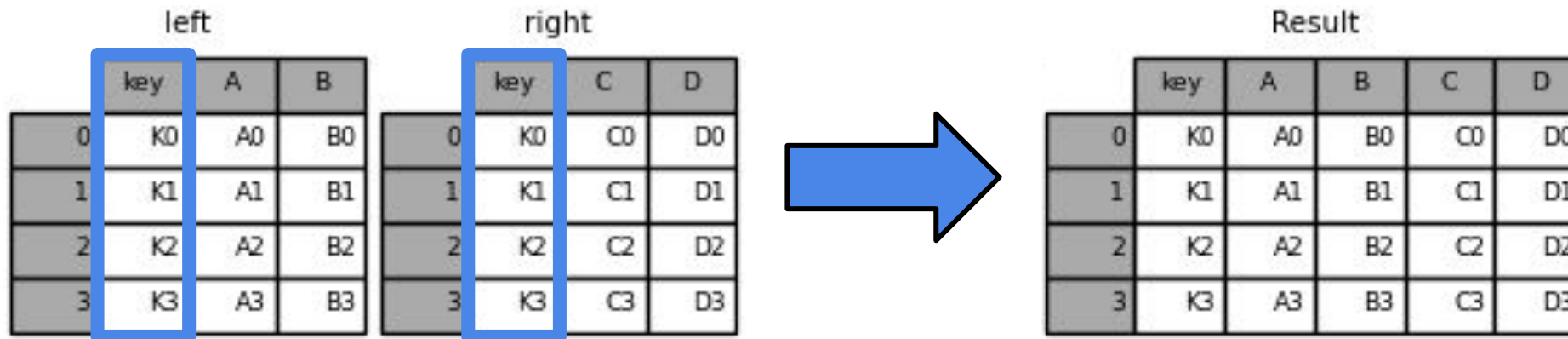
***pd.merge***(<df1>, <df2>, **on** = <col or list of cols to join on>, **how** = <join\_type>)

left				right			
	key	A	B		key	C	D
0	K0	A0	B0	0	K0	C0	D0
1	K1	A1	B1	1	K1	C1	D1
2	K2	A2	B2	2	K2	C2	D2
3	K3	A3	B3	3	K3	C3	D3

- Need one or more “key” columns to join on
- Pastes matching rows together along the key column(s)

## Merging two DataFrames

***pd.merge***(<df1>, <df2>, **on** = <col or list of cols to join on>, **how** = <join\_type>)

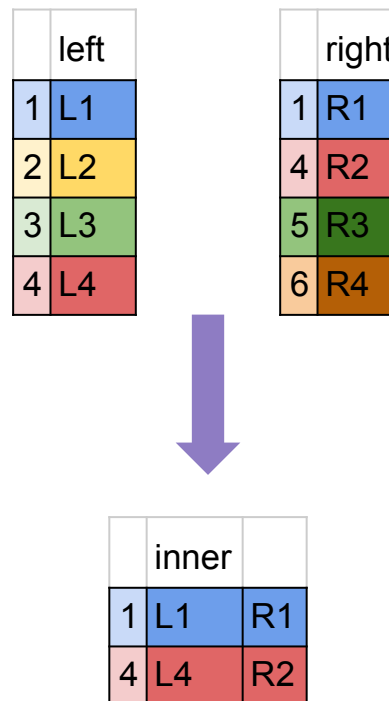


- Need one or more “key” columns to join on
- Pastes matching rows together along the key column(s)

# INNER JOIN

An **INNER JOIN** keeps **only the rows that have matching values in both tables.**

This is the default type of join when using `pd.merge()`.



# LEFT JOIN and RIGHT JOIN

A **LEFT JOIN** keeps all rows from the left table and all matching rows from the right table. A **RIGHT JOIN** works similarly, except all rows from the right table are kept.

*how = "left" or how = "right"*

	left		right
1	L1	1	R1
2	L2	4	R2
3	L3	5	R3
4	L4	6	R4



	left	
1	L1	R1
2	L2	
3	L3	
4	L4	R2

	left		right
1	L1	1	R1
2	L2	4	R2
3	L3	5	R3
4	L4	6	R4



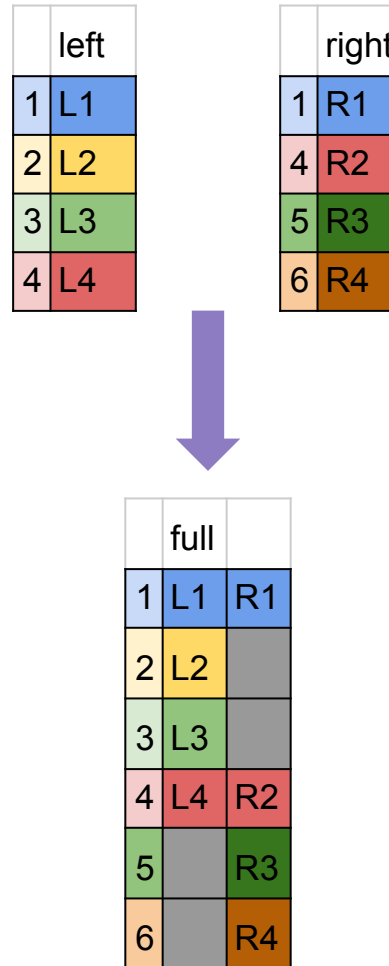
	right	
1	L1	R1
4	L4	R2
5		R3
6		R4



# OUTER/FULL JOIN

AN **OUTER JOIN** keeps all rows from both tables.

*how = "outer"*





# Questions?

