Introduction to Pandas

21 July 2020

Roadmap

Gene expression input file

	Gene 1	Gene 2	Gene 3	Gene 4
Sample A				
Sample B				
Sample C				

Use **pandas** to read in data as a **dataframe**

Use functions to manipulate the data:

1. **Access the data**
2. **Calculate statistics**
3. **Find patterns**

1. Initialize pandas DataFrame

Roadmap

Gene expression input file

	Gene 1	Gene 2	Gene 3	Gene 4
Sample A				
Sample B				
Sample C				

Use **pandas** to read in data as a **dataframe**

Use functions to manipulate the data:

1. **Access the data**
2. **Calculate statistics**
3. **Find patterns**

Import pandas library

```
1 # Step 1: Import modules
2 import pandas as pd # import pandas module
```

- We set pd as an alias for pandas

Read data into DataFrame

```
1 data = pd.read_csv(
2              data_file)          # file name
```

For more parameters see : http://pandas.pydata.org/pandas-docs/stable/

Read data into DataFrame

```
1 data = pd.read_csv(
2              data_file,          # file name
3              sep = '\t',         # file is tab-delimited
4              header = 0,         # Use first row as column header
5              index_col = 0)      # Use first column as index
```

For more parameters see : http://pandas.pydata.org/pandas-docs/stable/.

Read data into DataFrame

```
1 data = pd.read_csv(
2              gene_expression_file,
3              sep = '\t',
4              header = 0,
5              index_col = 0)
```

	BTNL8	LINC01134	HEATR4	ACO1	PLPP3
SRR493937	0.316291	0.037657	0.271263	7.680846	35.5811
SRR493938	0.211909	0.089802	0.270260	7.783635	34.7091
SRR493939	0.031951	0.180184	0.242934	3.674145	9.25606
SRR493940	0.072871	0.188795	0.302474	3.471724	9.36842
SRR493941	0.314067	0.089359	0.211705	6.003360	53.3867

Manually create DataFrame using a dictionary

```
1 data = pd.DataFrame(
2              data = {“treatmentA”: [2,4],  # data defined in a
3                      “treatmentB”: [4,2]# dictionary
4                      },
5              index= [“Billy”, “Bob”]
3              )
```

	treatmentA	treatmentB
Billy	2	4
Bob	4	2

There are other ways to define a dataframe

You can also create a dataframe using different combinations of lists and dictionaries: https://pbpython.com/pandas-list-dict.html

Extra examples

We can create a DataFrame from a Dictionary of pandas series

```
1 dictOfSeries = {'treatmentA':pd.Series([2,4],['Billy','Bob']),
2                 'treatmentB':pd.Series([4,2],['Billy','Bob'])}
3 df = pd.DataFrame(dictOfSeries)
4 df
```

	treatmentA	treatmentB
Billy	2	4
Bob	4	2

Or we can create a DataFrame from a Dictionary of Lists

```
1  import pandas as pd # import pandas module
2
3  # create dictionary of lists
4  exp_dictOfList = {
5      "treatmentA":[2,4],
6      "treatmentB":[4,2]
7  }
8
9  # initialize the DataFrame
10 gene_frame = pd.DataFrame(exp_dictOfList,
11                            index=["Billy","Bob"]) # row names
12
13 gene_frame
```

	treatmentA	treatmentB
Billy	2	4
Bob	4	2

We can create a DataFrame from a List of Lists

```
1  import pandas as pd # import pandas module
2
3  # create list of lits
4  exp_listOflist = [
5      [2,4],
6      [4,2]
7  ]
8
9  # initialize the DataFrame
10 gene_frame = pd.DataFrame(exp_listOflist,
11                            columns = ["treatmentA","treatmentB"],
12                            index = ["Billy","Bob"]) # row names
13
14
15
16 gene_frame
```

	treatmentA	treatmentB
Billy	2	4
Bob	4	2

Or we can create a DataFrame from a List of Dictionaries

```
1  import pandas as pd # import pandas module
2
3  # create list of dictionaries
4  exp_listOfDict = [
5      {"treatmentA":2,"treatmentB":4},
6      {"treatmentA":4,"treatmentB":2}
7  ]
8
9  # initialize the DataFrame
10 gene_frame = pd.DataFrame(exp_listOfDict,
11                            index=["Billy","Bob"]) # row names
12
13
14
15 gene_frame
```

	treatmentA	treatmentB
Billy	2	4
Bob	4	2

We can create a DataFrame from a Dictionary of Dictionaries

```
1  import pandas as pd # import pandas module
2
3  # create dictionary of dictionary
4  exp_dictOfDict = {
5      "treatmentA":{"Billy":2, "Bob":4},
6      "treatmentB":{"Billy":4, "Bob":2}
7  }
8
9  # initialize the DataFrame
10 gene_frame = pd.DataFrame(exp_dictOfDict)
11
12 gene_frame
```

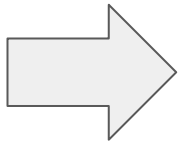
	treatmentA	treatmentB
Billy	2	4
Bob	4	2

2. Access data in DataFrame

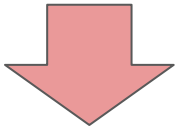
Roadmap

Gene expression input file

	Gene 1	Gene 2	Gene 3	Gene 4
Sample A				
Sample B				
Sample C				



Use **pandas** to read in data as a **dataframe**



Use functions to manipulate the data:

1. **Access the data**
2. **Calculate statistics**
3. **Find patterns**

Select a column of a DataFrame

`peoples_df`

name	age	inches
Bob	56	71.0
Sue	21	63.5
Tom	22	68.5
Pam	35	62.0

Select a column of a DataFrame

`peoples_df`

name	age	inches
Bob	56	71.0
Sue	21	63.5
Tom	22	68.5
Pam	35	62.0

Select a column of a DataFrame

```
1 peoples_df["inches"]
```

```
name
Bob     71.0
Sue     63.5
Tom     68.5
Pam     62.0
Name: inches, dtype: float64
```

Select a row

`peoples_df`

name	age	inches
Bob	56	71.0
Sue	21	63.5
Tom	22	68.5
Pam	35	62.0

Select a row by index name

```
1 peoples_df.loc[["Sue", "Tom"]]
```

name	age	inches
Sue	21	63.5
Tom	22	68.5

Select a row by integer

	name	age	inches
0	Bob	56	71.0
1	Sue	21	63.5
2	Tom	22	68.5
3	Pam	35	62.0

Row names

Row integer location

```
1 peoples_df.loc[["Sue", "Tom"]]
```

name	age	inches
Sue	21	63.5
Tom	22	68.5

```
1 peoples_df.iloc[[1,2]]
```

name	age	inches
Sue	21	63.5
Tom	22	68.5

Select a row and column

	name	age	inches
0	Bob	56	71.0
1	Sue	21	63.5
2	Tom	22	68.5
3	Pam	35	62.0

Row names

Row integer location

Rows you want

Columns you want

```
1 peoples_df.loc[["Sue","Tom"],"inches"]
```

```
name
Sue    63.5
Tom    68.5
Name: inches, dtype: float64
```

```
1 peoples_df.iloc[[1,2],1]
```

```
name
Sue    63.5
Tom    68.5
Name: inches, dtype: float64
```

Selecting an Index or Column From a DataFrame using .iloc & .loc

- [<column_names>] - command to get values in DataFrame based on column headers
- .iloc[<row_idx>, <col_idx>] - command to get values in DataFrame based on row and column locations
- .loc[<row_names>, <col_names>] - command to get values in DataFrame based on row values and column values

Basic DataFrame Functions -- properties of your DataFrame

- bed_df.**head(n=*X*)** : gets the first ***X*** rows (default: n=5 if not stated)
- bed_df.**tail(n=*X*)** : gets the last ***X*** rows (default: n=5 if not stated)
- bed_df.**columns** : gets column names
- bed_df.**index** : gets row names
- bed_df.**dtypes** : gets datatypes of each column
- bed_df.**shape** : gets height and width of the DataFrame in a list format
- **len(**bed_df**)** : gets the height of the DataFrame
- bed_df.**describe**(): generates descriptive statistics

Extra examples

Set index

`peoples_df`

	age	inches	name
0	56	71.0	Bob
1	21	63.5	Sue
2	22	68.5	Tom
3	35	62.0	Pam

Set index

```
1 peoples_df = peoples_df.set_index("name")
2 peoples_df
```

name	age	inches
Bob	56	71.0
Sue	21	63.5
Tom	22	68.5
Pam	35	62.0

Select a row by index name

```
1 peoples_df.loc[["Sue", "Tom"]]
```

name	age	inches
Sue	21	63.5
Tom	22	68.5

Which rows will print after the following command (from which row indexes)?

peoples_df

	age	inches	name
0	56	71.0	Bob
1	21	63.5	Sue
2	22	68.5	Tom
3	35	62.0	Pam

```
1 peoples_df.iloc[range(1,3),:]
```

Which rows will print after the following command (from which row indexes)?

peoples_df

	age	inches	name
0	56	71.0	Bob
1	21	63.5	Sue
2	22	68.5	Tom
3	35	62.0	Pam

```
1 peoples_df.iloc[range(1,3),:]
```

variable name for DataFrame

Which rows will print after the following command (from which row indexes)?

peoples_df

	age	inches	name
0	56	71.0	Bob
1	21	63.5	Sue
2	22	68.5	Tom
3	35	62.0	Pam

```
1 peoples_df.iloc[range(1,3),:]
```

variable name for DataFrame

get values in DataFrame based on row and column locations

Which rows will print after the following command (from which row indexes)?

peoples_df

	age	inches	name
0	56	71.0	Bob
1	21	63.5	Sue
2	22	68.5	Tom
3	35	62.0	Pam

This gives a range of numbers from 1-2 for row locations

```
1 peoples_df.iloc[range(1,3),:]
```

variable name for DataFrame

get values in DataFrame based on row and column locations

Which rows will print after the following command (from which row indexes)?

peoples_df

	age	inches	name
0	56	71.0	Bob
1	21	63.5	Sue
2	22	68.5	Tom
3	35	62.0	Pam

This gives a range of numbers from 1-2 for row locations

```
1 peoples_df.iloc[range(1,3),:]
```

variable name for DataFrame

get values in DataFrame based on row and column locations

This colon symbol represents all of the columns

Which rows will print after the following command (from which row indexes)?

```
1 import pandas as pd
2 peoples_df = pd.DataFrame(
3     {"name" : ["Bob", "Sue", "Tom", "Pam"],
4      "age" : [56, 21, 22, 35],
5      "inches" : [71, 63.5, 68.5, 62]})
6 peoples_df
```

	age	inches	name
0	56	71.0	Bob
1	21	63.5	Sue
2	22	68.5	Tom
3	35	62.0	Pam

```
1 peoples_df.iloc[range(1,3),:]
```

	age	inches	name
1	21	63.5	Sue
2	22	68.5	Tom

ANSWER: This prints rows with indexes 1 & 2 since they are the second and third row

Which rows will print after the following command?

```
1 peoples_df
```

	age	inches	name
0	56	71.0	Bob
1	21	63.5	Sue
2	22	68.5	Tom
3	35	62.0	Pam

```
1 peoples_df.loc[range(1,3),:]
```

Which rows will print after the following command?

```
1 peoples_df
```

	age	inches	name
0	56	71.0	Bob
1	21	63.5	Sue
2	22	68.5	Tom
3	35	62.0	Pam

```
1 peoples_df.loc[range(1,3),:]
```

	age	inches	name
1	21	63.5	Sue
2	22	68.5	Tom

ANSWER: This also prints rows with indexes 1 & 2 since the value of these row indexes are 1 & 2 respectively since we do NOT have row names

Which rows will print after the following command?

```
1 peoples_df
```

name	age	inches
Bob	56	71.0
Sue	21	63.5
Tom	22	68.5
Pam	35	62.0

```
1 peoples_df.loc[1,:]
```

Which rows will print after the following command?

```
1 peoples_df
```

name	age	inches
Bob	56	71.0
Sue	21	63.5
Tom	22	68.5
Pam	35	62.0

```
1 peoples_df.loc[1,:]
```

ANSWER: This prints an ERROR since there is no row name “1”

Which rows will print after the following command?

```
1 peoples_df
```

name	age	inches
Bob	56	71.0
Sue	21	63.5
Tom	22	68.5
Pam	35	62.0

```
1 peoples_df.loc[["Bob","Tom"],:]
```

Which rows will print after the following command?

```
1 peoples_df
```

name	age	inches
Bob	56	71.0
Sue	21	63.5
Tom	22	68.5
Pam	35	62.0

```
1 peoples_df.loc[["Bob","Tom"],:]
```

name	age	inches
Bob	56	71.0
Tom	22	68.5

ANSWER: This prints the rows with row names “Bob” and “Tom”

What value would this print?

```
1 peoples_df
```

name	age	inches
Bob	56	71.0
Sue	21	63.5
Tom	22	68.5
Pam	35	62.0

```
1 peoples_df.iloc[1,1]
```

What value would this print?

```
1 peoples_df
```

name	age	inches
Bob	56	71.0
Sue	21	63.5
Tom	22	68.5
Pam	35	62.0

```
1 peoples_df.iloc[1,1]
```

ANSWER: 63.5

What value would this print?

```
1 peoples_df
```

name	age	inches
Bob	56	71.0
Sue	21	63.5
Tom	22	68.5
Pam	35	62.0

```
1 peoples_df.loc["Sue","age"]
```

What value would this print?

```
1 peoples_df
```

name	age	inches
Bob	56	71.0
Sue	21	63.5
Tom	22	68.5
Pam	35	62.0

```
1 peoples_df.loc["Sue","age"]
```

ANSWER: 21

Which column will print after the following command?

```
1 peoples_df
```

name	age	inches
Bob	56	71.0
Sue	21	63.5
Tom	22	68.5
Pam	35	62.0

```
1 peoples_df.loc[:,"age"]
```

Which column will print after the following command?

```
1 peoples_df
```

name	age	inches
Bob	56	71.0
Sue	21	63.5
Tom	22	68.5
Pam	35	62.0

```
1 peoples_df.loc[:,"age"]
```

```
name
Bob     56
Sue     21
Tom     22
Pam     35
Name: age, dtype: int64
```

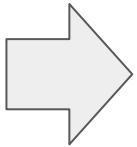
ANSWER: The “age” column

But why is the output formatted differently?

Select a subset of your data

```
1 peoples_df["age"] < 50
```

```
name
Bob    False
Sue     True
Tom     True
Pam     True
Name: age, dtype: bool
```



```
1 peoples_df.loc[peoples_df["age"]<50]
```

name	age	inches
Sue	21	63.5
Tom	22	68.5
Pam	35	62.0

What value would this print?

```
1 peoples_df
```

name	age	inches
Bob	56	71.0
Sue	21	63.5
Tom	22	68.5
Pam	35	62.0

```
1 min(peoples_df.loc[peoples_df.loc[:,"age"]<50,"inches"])
```

What value would this print?

```
1 peoples_df
```

name	age	inches
Bob	56	71.0
Sue	21	63.5
Tom	22	68.5
Pam	35	62.0

```
1 min(peoples_df.loc[peoples_df.loc[:,"age"]<50,"inches"])
```

ANSWER: 62.0

What row(s) would this print?

```
1  peoples_df
```

name	age	inches
Bob	56	71.0
Sue	21	63.5
Tom	22	68.5
Pam	35	62.0

Note: for multiple conditions inside .loc or .iloc we only use a single ampersand

```
1  peoples_df.loc[(peoples_df.loc[:,"age"]<50) &
2                 (peoples_df.loc[:,"inches"]>65),:]
```

What row(s) would this print?

```
1  peoples_df
```

name	age	inches
Bob	56	71.0
Sue	21	63.5
Tom	22	68.5
Pam	35	62.0

```
1  peoples_df.loc[(peoples_df.loc[:,"age"]<50) &
2                 (peoples_df.loc[:,"inches"]>65),:]
```

ANSWER:

name		
Tom	22	68.5

3. Explore your data

Roadmap

Gene expression input file

	Gene 1	Gene 2	Gene 3	Gene 4
Sample A				
Sample B				
Sample C				

Use **pandas** to read in data as a **dataframe**

Use functions to manipulate the data:

1. Access the data
2. Calculate statistics
3. **Find patterns**

Quality check your data

```
1  peoples_2_df["inches"].isna()
```

```
0    False
1    False
2    False
3    False
Name: inches, dtype: bool
```

```
1  peoples_2_df.isna()
```

	age	inches	name
0	True	False	False
1	False	False	False
2	False	False	False
3	False	False	False

peoples_2_df

	age	inches	name
0	NaN	71.0	Bob
1	21.0	63.5	Sue
2	22.0	0.0	Tom
3	35.0	62.0	Pam

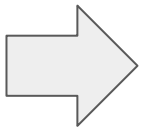
Quality check your data

```
1  peoples_2_df["inches"].isna()
```

```
0    False
1    False
2    False
3    False
Name: inches, dtype: bool
```

```
1  peoples_2_df.isna()
```

	age	inches	name
0	True	False	False
1	False	False	False
2	False	False	False
3	False	False	False



```
1  peoples_2_df.isna().sum()
```

```
age       1
inches    0
name      0
dtype: int64
```

False (Not NaN) = 0
True (is NaN) = 1

Quality check your data

```
1  peoples_2_df == 0
```

	age	inches	name
0	False	False	False
1	False	False	False
2	False	True	False
3	False	False	False

```
1  (peoples_2_df == 0).sum()
```

```
age       0
inches    1
name      0
dtype: int64
```

False (Not 0) = 0
True (is 0) = 1

peoples_2_df

	age	inches	name
0	NaN	71.0	Bob
1	21.0	63.5	Sue
2	22.0	0.0	Tom
3	35.0	62.0	Pam

What is the distribution of your data?

```
1 peoples_df["inches"].mean()
```

```
66.25
```

```
1 peoples_df.mean()
```

```
age       33.50
inches    66.25
dtype: float64
```

What is the distribution of your data?

```
1  print(peoples_df["inches"].mean())
2  print(peoples_df["inches"].std())
3  print(peoples_df["inches"].min())
4  print(peoples_df["inches"].max())
```

```
66.25
4.2130748865881795
62.0
71.0
```

Additional functions

You can also use .any() to find if there is at least 1 TRUE value

You can also use .all() to find if all values are TRUE

4. Find patterns in your data

Roadmap

Gene expression input file

	Gene 1	Gene 2	Gene 3	Gene 4
Sample A				
Sample B				
Sample C				

Use **pandas** to read in data as a **dataframe**

Use functions to manipulate the data:

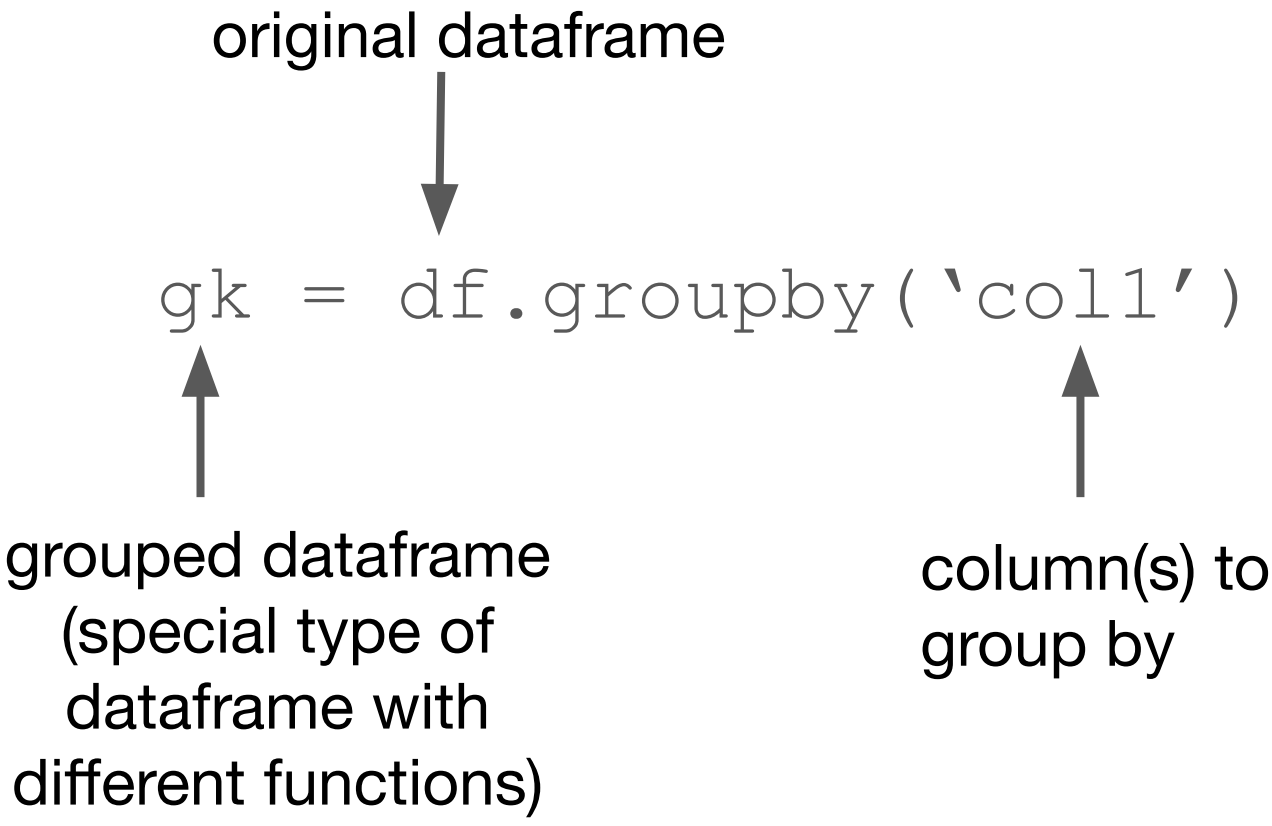
1. Access the data
2. Calculate statistics
3. Find patterns

What is grouping?

- Grouping means to split a data frame into groups based on some criteria.
- Given a list of ChIP-seq peaks, I want to group the peaks by the genes they are annotated to.

	peak_name	**gene_name**
0	peak1	geneA
1	peak2	geneB
2	peak3	geneC
3	peak4	geneC

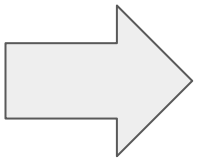
Groupby function call



df

	peak	gene
0	peak1	geneA
1	peak2	geneB
2	peak3	geneC
3	peak4	geneC
4	peak5	geneC
5	peak6	geneE
6	peak7	geneE
7	peak8	geneH
8	peak9	geneH

```
df.groupby('gene')
```



	peak	gene
0	peak1	geneA

	peak	gene
1	peak2	geneB

	peak	gene
2	peak3	geneC
3	peak4	geneC
4	peak5	geneC

	peak	gene
5	peak6	geneE
6	peak7	geneE

	peak	gene
7	peak8	geneH
8	peak9	geneH

Looping through grouped Data Frames

The format for looping through grouped Data Frames is:

```
for <group>,<group_df> in <groupby_df>
```

Example

peak_gene_df

	peak	gene
0	peak1	geneA
1	peak2	geneB
2	peak3	geneC
3	peak4	geneE
4	peak5	geneE
5	peak6	geneC
6	peak7	geneC

```
1 peak_gene_grouped = peak_gene_df.groupby("gene")
2 for gene_name, gene_df in peak_gene_grouped:
3   print(gene_name)
4   print(gene_df)
```

```
geneA
    peak1   geneA
geneB
    peak2   geneB
geneC
    peak3   geneC
    peak6   geneC
    peak7   geneC
geneE
    peak4   geneE
    peak5   geneE
```

Perform functions on groups

`peak_gene_df`

	peak	gene	log10_pvalue
0	peak1	geneA	30
1	peak2	geneB	10
2	peak3	geneC	11
3	peak4	geneE	15
4	peak5	geneE	23
5	peak6	geneC	99
6	peak7	geneC	102

```
peak_gene_df.groupby("gene").mean(
)
```

gene	log10_pvalue
geneA	30
geneB	10
geneC	70.667
geneE	19

Combining DataFrames

Up to this point, we have been working with a single table at a time, but most of the time we work with many tables that can be related to each other in some way.

Concatenating DataFrames

It is easy to merge **multiple dataframes that have the same columns** together using the pandas concat() function

```
1 df1
```

	ONE	TWO
0	0.181987	0.853200
1	0.124461	-0.705579
2	0.079081	0.562427

```
1 df2
```

	ONE	TWO
0	0.010352	-0.203493
1	0.036690	0.383092
2	0.136860	-0.739893

```
1 pd.concat([df1,df2])
```

	ONE	TWO
0	0.181987	0.853200
1	0.124461	-0.705579
2	0.079081	0.562427
0	0.010352	-0.203493
1	0.036690	0.383092
2	0.136860	-0.739893

Concatenating DataFrames

It is easy to merge **multiple dataframes that have the same columns** together using the pandas concat() function

```
1 df1
```

	ONE	TWO
0	0.181987	0.853200
1	0.124461	-0.705579
2	0.079081	0.562427

```
1 df2
```

	ONE	TWO
0	0.010352	-0.203493
1	0.036690	0.383092
2	0.136860	-0.739893

```
1 pd.concat([df1,df2], ignore_index=True)
```

	ONE	TWO
0	0.181987	0.853200
1	0.124461	-0.705579
2	0.079081	0.562427
3	0.010352	-0.203493
4	0.036690	0.383092
5	0.136860	-0.739893

Use **ignore_index** to reset the indices so that there are none that are redundant

Merging DataFrames

Say you have tables that contain data about the same set of items (found in rows) but contain different information about those items (some different columns)

Peak ID	Gene ID	Distance From Peak to Gene
1	AT5G61850	50
2	AT5G61850	0
3	AT1G17180	0
4	AT2G11180	210

Gene ID	Function
AT1G17180	Ribosomal
AT1G17190	Respiratory
AT1G17200	Respiratory
AT1G17201	Stress Response

Merging DataFrames

```
df1.merge(df2, left_on='lkey', right_on='rkey')
```

df1

Peak ID	Gene ID	Distance From Peak to Gene
1	AT5G61850	50
2	AT5G61850	0
3	AT1G17180	0
4	AT2G11180	210

df2

Gene ID	Function
AT1G17180	Ribosomal
AT1G17190	Respiratory
AT1G17200	Respiratory
AT1G17201	Stress Response

Merging DataFrames

Peak ID	Gene ID	Distance From Peak to Gene	Function
1	AT5G61850	50	NA
2	AT5G61850	0	NA
3	AT1G17180	0	Ribosomal
4	AT2G11180	210	NA
NA	AT1G17190	NA	Respiratory
NA	AT1G17200	NA	Respiratory
NA	AT1G17201	NA	Stress Response

Extra examples

Access Specific Groups

```
1 peak_gene_df
```

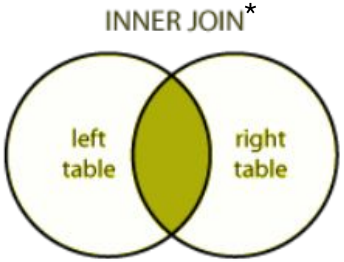
	peak	gene	log10_pvalue
0	peak1	geneA	30
1	peak2	geneB	10
2	peak3	geneC	11
3	peak4	geneE	15
4	peak5	geneE	23
5	peak6	geneC	99
6	peak7	geneC	102

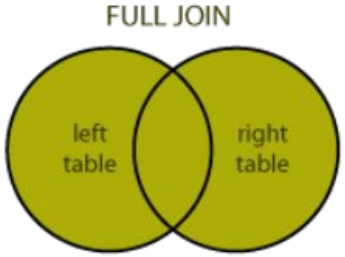
```
1 # get dataframe for geneC only
2 peak_gene_grouped = \
3   peak_gene_df.groupby("gene")
4 peak_gene_grouped.get_group("geneE")
```

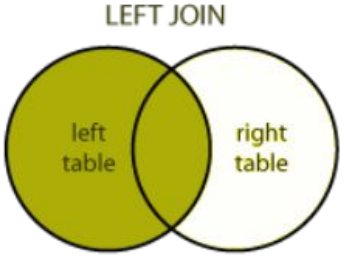
	peak	gene	log10_pvalue
2	peak3	geneC	11
5	peak6	geneC	99
6	peak7	geneC	102

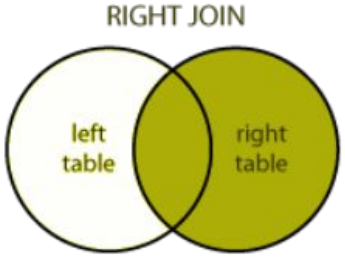
Different Ways to join a DataFrame

- Not all data frames have the exact same foreign keys that match the primary keys. Some keys should match but each data frame may have keys that are not found in the other.
  - There are four primary ways to join two tables together









* INNER JOIN is the default for pandas merge

Different Ways to join a DataFrame

IDs in df1: a,b,c IDs in df2: a,b,d

Different Ways to join a DataFrame

inner join

IDs in df1: a,b,c IDs in df2: a,b,d ----> IDs in merged: a,b

Different Ways to join a DataFrame

inner join

IDs in df1: a,b,c IDs in df2: a,b,d ----> IDs in merged: a,b

left join

IDs in df1: a,b,c IDs in df2: a,b,d ----> IDs in merged: a,b,c

Different Ways to join a DataFrame

inner join

IDs in df1: a,b,c IDs in df2: a,b,d ----> IDs in merged: a,b

left join

IDs in df1: a,b,c IDs in df2: a,b,d ----> IDs in merged: a,b,c

right join

IDs in df1: a,b,c IDs in df2: a,b,d ----> IDs in merged: a,b,d

Different Ways to join a DataFrame

inner join

IDs in df1: a,b,c IDs in df2: a,b,d ----> IDs in merged: a,b

left join

IDs in df1: a,b,c IDs in df2: a,b,d ----> IDs in merged: a,b,c

right join

IDs in df1: a,b,c IDs in df2: a,b,d ----> IDs in merged: a,b,d

outer join

IDs in df1: a,b,c IDs in df2: a,b,d ----> IDs in merged: a,b,c,d

6. Writing a dataframe to a file

Writing DataFrame to File

Writing DataFrame to File

The syntax for exporting a DataFrame is :

```
dataFrame.to_csv(path_to_outputFile, arguments)
```

For example, to export a tab-delimited file from DataFrame `test_df` with a header and no row-names to the file “test.txt” we use the command:

```
test_df.to_csv("test.txt", sep="\t", header=True, index=False)
```

It is useful for any data that looks like a spreadsheet