Lecture 21
Pandas

Course: Practical Bioinformatics (BIOL 4220)
Instructor: Michael Landis
Email: michael.landis@wustl.edu

Lecture 21 outline

Last time: NumPy

This time: Pandas

- Pandas overview
- Pandas containers
- Pandas methods

pandas

The ***pandas*** package improves the data-processing and data-organizing capabilities of Python using a variety of new containers and methods

pandas containers are especially when:
- working with "labeled" data tables
- working with data tables with mixed data types
- exploring datasets through the interactive console
- applying complex operations to only specific rows/columns

Low memory usage, fast built-in methods
Convenient read/write file methods

https://pandas.pydata.org/

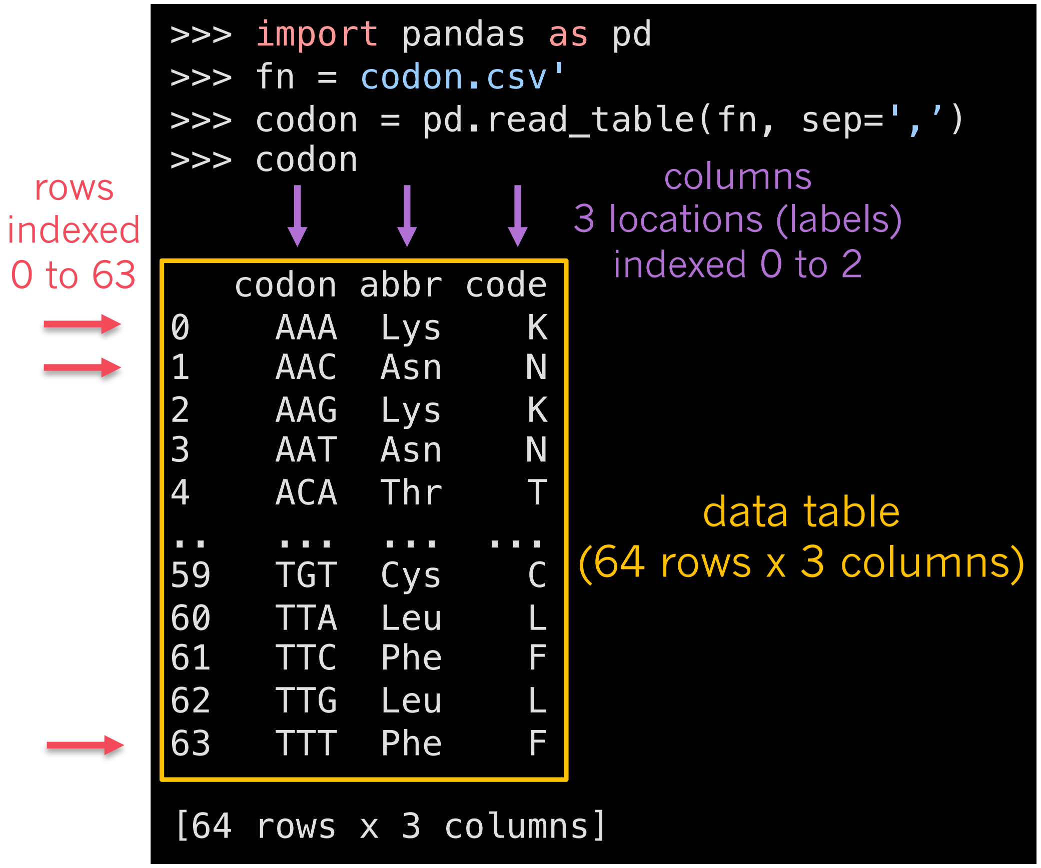
pd.DataFrame anatomy

```
$ cat codons.csv
codon,abbr,code
AAA,Lys,K
AAC,Asn,N
AAG,Lys,K
AAT,Asn,N
ACA,Thr,T
ACC,Thr,T
ACG,Thr,T
ACT,Thr,T
AGA,Arg,R

...

TTG,Leu,L
TTT,Phe,F
```

csv in shell



```
>>> import pandas as pd
>>> fn = codon.csv'
>>> codon = pd.read_table(fn, sep=',’)
>>> codon

   codon abbr code
0    AAA  Lys    K
1    AAC  Asn    N
2    AAG  Lys    K
3    AAT  Asn    N
4    ACA  Thr    T
..   ...  ...  ...
59   TGT  Cys    C
60   TTA  Leu    L
61   TTC  Phe    F
62   TTG  Leu    L
63   TTT  Phe    F

[64 rows x 3 columns]
```

Python code using Pandas

reading and writing files

```
>>> # read data into new DataFrame
>>> pd.read_<press tab>
pd.read_clipboard(  pd.read_gbq(         pd.read_parquet(    pd.read_sql_query(
pd.read_csv(        pd.read_hdf(         pd.read_pickle(     pd.read_sql_table(
pd.read_excel(      pd.read_html(        pd.read_sas(        pd.read_stata(
pd.read_feather(    pd.read_json(        pd.read_spss(       pd.read_table(
pd.read_fwf(        pd.read_orc(         pd.read_sql(

>>> # return values of DataFrame as new type
>>> df.to_<press tab>
df.to_clipboard(  df.to_hdf(        df.to_parquet(    df.to_string(
df.to_csv(        df.to_html(       df.to_period(     df.to_timestamp(
df.to_dict(       df.to_json(       df.to_pickle(     df.to_xarray(
df.to_excel(      df.to_latex(      df.to_records(
df.to_feather(    df.to_markdown(   df.to_sql(
df.to_gbq(        df.to_numpy(      df.to_stata(

>>> # example: read csv, then write to excel
>>> df = pd.read_csv(‘amino_acids.csv')
>>> df.to_excel(‘amino_acids.xlsx')
```

most standard formats supported
(csv, tsv, excel, json, SQL, etc.)

shape and axes

Extract info about the dimensions and size of the container

```
>>> codon = pd.read_csv('codons.csv', sep=',’)
>>> # shape returns ordered sizes of dimensions
>>> codon.shape
(64, 3)
>>> # total size is the product of all dimension sizes
>>> codon.size
192
>>> # labeled column names
>>> codon.columns
Index(['codon', 'abbr', 'code'], dtype='object’)
>>> # unlabeled row (index) names
>>> codon.index
RangeIndex(start=0, stop=64, step=1)
>>> # row and column info
>>> codon.axes
[RangeIndex(start=0, stop=64, step=1), Index(['codon', 'abbr', 'code’],
 dtype='object')]
```

Various helper methods to view container properties and contents

```
>>> # partial view
>>> codon
   codon abbr code
0    AAA  Lys    K
1    AAC  Asn    N
2    AAG  Lys    K
3    AAT  Asn    N
4    ACA  Thr    T
..   ...  ...  ...
59   TGT  Cys    C
60   TTA  Leu    L
61   TTC  Phe    F
62   TTG  Leu    L
63   TTT  Phe    F

[64 rows x 3 columns]

>>> # first three lines
>>> codon.head(3)
  codon abbr code
0   AAA  Lys    K
1   AAC  Asn    N
2   AAG  Lys    K
>>> # last three lines
>>> codon.tail(3)
   codon abbr code
61   TTC  Phe    F
62   TTG  Leu    L
63   TTT  Phe    F
```

```
>>> # overview of DataFrame properties
>>> codon.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 64 entries, 0 to 63
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   codon   64 non-null     object
 1   abbr    64 non-null     object
 2   code    64 non-null     object
dtypes: object(3)
memory usage: 1.6+ KB

>>> # summary of DataFrame contents
>>> codon.describe()
 codon abbr code
count     64   64   64
unique    64   21   21
top      AGC  Arg    L
freq       1    6    6
```

```
>>> aa.head(3)
         name abbr code mol_formula  mol_weight  hydrophob
0     alanine  Ala    A     C3H7NO2       89.10         41
1    arginine  Arg    R   C6H14N4O2      174.20        -14
2  asparagine  Asn    N    C4H8N2O3      132.12        -28
>>> aa.tail(3)
          name abbr code mol_formula  mol_weight  hydrophob
17  tryptophan  Trp    W  C11H12N2O2      204.23         97
18    tyrosine  Tyr    Y    C9H11NO3      181.19         63
19      valine  Val    V    C5H11NO2      117.15         76
```

```
>>> aa.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20 entries, 0 to 19
Data columns (total 6 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   name         20 non-null     object
 1   abbr         20 non-null     object
 2   code         20 non-null     object
 3   mol_formula  20 non-null     object
 4   mol_weight   20 non-null     float64
 5   hydrophob    20 non-null     int64
dtypes: float64(1), int64(1), object(4)
memory usage: 1.1+ KB
```

Various helper methods to view container properties and contents (different table)

```
>>> aa.describe()
       mol_weight   hydrophob
count   20.000000   20.000000
mean   136.903000   25.250000
std     30.863209   52.885054
min     75.070000  -55.000000
25%    118.627500  -16.250000
50%    132.615000   10.500000
75%    150.697500   74.500000
max    204.230000  100.000000
```

pd.Series and *pd.DataFrame*

```
>>> ## create a 1D Series
>>> data_1d = ['Homo','Pan','Gorilla', 'Pongo']
>>> df_1d = pd.Series(data_1d, name='Genus’)
0       Homo
1        Pan
2    Gorilla
3      Pongo
Name: Genus, dtype: object
>>> ## create a 2D DataFrame
>>> data_2d = [['Homo', 60],['Pan', 45],['Gorilla', 125], ['Pongo', 50]]
>>> df_2d = pd.DataFrame(data_2d, columns=['Genus','Mass_kg’])
>>> df_2d
     Genus  Mass_kg
0     Homo       60
1      Pan       45
2  Gorilla      125
3    Pongo       50
>>> ## DataFrames are built with Series
>>> df_2d['Genus’]
0       Homo
1        Pan
2    Gorilla
3      Pongo
Name: Genus, dtype: object
```

extract column from *df_2d*
with location label ‘Genus’

data types

```
>>> ## Compare the container types
>>> type(df_1d)
<class 'pandas.core.series.Series’>
>>> type(df_2d)
<class 'pandas.core.frame.DataFrame'>
>>> type(df_2d['Genus'])
<class 'pandas.core.series.Series’>
>>> ## Compare the element dtypes
>>> df_1d.dtype
dtype('O')
>>> df_2d.dtypes
Genus       object
Mass_kg      int64
dtype: object
>>> df_2d['Genus'].dtype
dtype('O')
```

Container types

- *pd.Series*
- *pd.DataFrame*

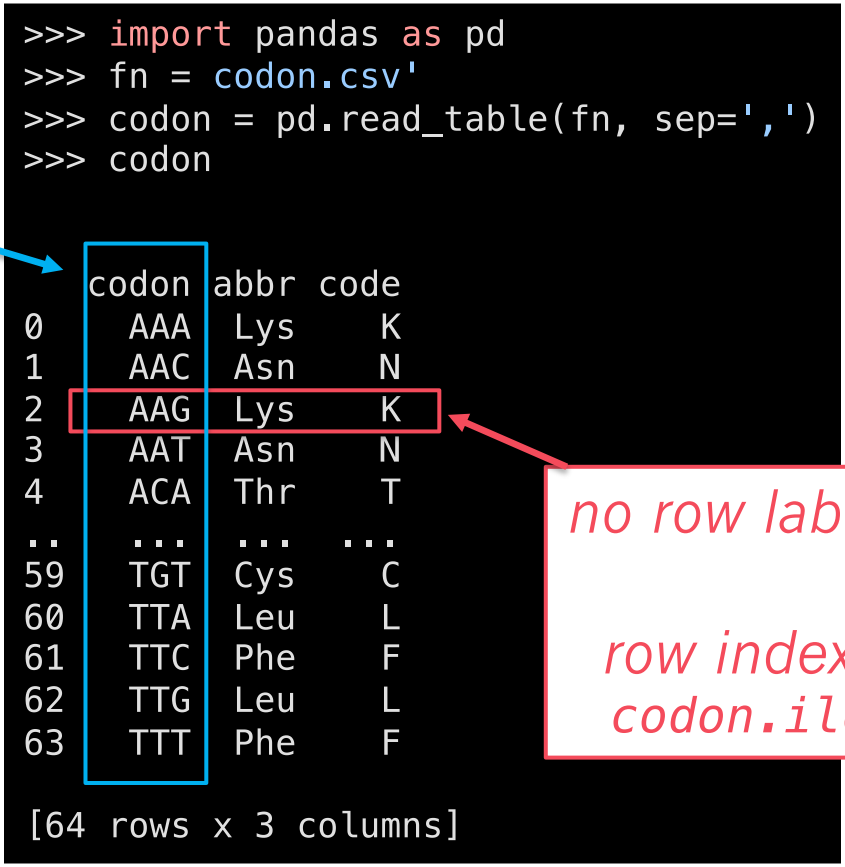
Element dtypes

- *object* (e.g. *string*)
- *int64*
- *float64*
- *bool*
- *datetime64*
- (a few others)

accessing elements

column label-location
`codon.loc[:, ‘codon’]`

column index-location
`codon.iloc[:, 0]`



```
>>> import pandas as pd
>>> fn = codon.csv'
>>> codon = pd.read_table(fn, sep=',')
>>> codon


   codon abbr  code
0    AAA  Lys     K
1    AAC  Asn     N
2    AAG  Lys     K
3    AAT  Asn     N
4    ACA  Thr     T
..   ...  ...   ...
59   TGT  Cys     C
60   TTA  Leu     L
61   TTC  Phe     F
62   TTG  Leu     L
63   TTT  Phe     F

[64 rows x 3 columns]
```

no row label-location

row index-location
`codon.iloc[2, :]`

accessing elements

```
>>> aa = pd.read_csv('amino_acids.csv', index_col='code')
>>> aa.columns
Index(['name', 'abbr', 'mol_formula', 'mol_weight', 'hydrophob'],
      dtype='object')
>>> aa.index
Index(['A', 'R', 'N', 'D', 'C', 'E', 'Q', 'G', 'H', 'I', 'L', 'K',
       'M', 'F', 'P', 'S', 'T', 'W', 'Y', 'V'],
      dtype='object', name='code')
```

df.columns and *df.index* return labels for accessing rows, columns, and/or elements

```
>>> aa.loc['A',:]
name             alanine
abbr                 Ala
mol_formula      C3H7NO2
mol_weight          89.1
hydrophob             41
Name: A, dtype: object
```

first position accesses row

```
>>> aa.loc[:,'mol_weight']
code
A      89.10
R     174.20
...
Y     181.19
V     117.15
Name: mol_weight, dtype: float64
```

second position accesses column

```
>>> aa.loc['A',:]['mol_weight']
89.1
>>> aa.loc[:,'mol_weight']['A']
89.1
>>> aa.loc['A','mol_weight']
89.1
>>> aa.iloc[0,:]['mol_weight']
89.1
>>> aa.iloc[:,3]['A']
89.1
>>> aa.iloc[0,3]
89.1
>>> aa.loc['A',:][3]
89.1
>>> aa.iloc[0,:]['mol_weight']
89.1
```

accessing the same element with *loc* and *iloc*

```
>>> aa.loc['A':'C','name':'abbr']
               name abbr
code
A           alanine  Ala
R          arginine  Arg
N        asparagine  Asn
D     aspartic acid  Asp
C          cysteine  Cys
```

slice indexing by label

```
>>> aa.loc[['A','R'],:]
          name abbr mol_formula  mol_weight  hydrophob
code
A      alanine  Ala     C3H7NO2        89.1         41
R     arginine  Arg   C6H14N4O2       174.2        -14
>>> aa.loc[['A','R'], ['mol_weight','hydrophob']]
      mol_weight  hydrophob
code
A           89.1         41
R          174.2        -14
>>> aa.iloc[[0,1],:]
          name abbr mol_formula  mol_weight  hydrophob
code
A      alanine  Ala     C3H7NO2        89.1         41
R     arginine  Arg   C6H14N4O2       174.2        -14
>>> aa.iloc[[0,1],[3,4]]
      mol_weight  hydrophob
code
A           89.1         41
R          174.2        -14
>>> aa.iloc[0:2,3:5]
      mol_weight  hydrophob
code
A           89.1         41
R          174.2        -14
```

more slice indexing examples using *loc* and *iloc*

```
>>> aa['mol_weight'] > 150
code
A     False
R      True
N     False
...
W      True
Y      True
V     False
Name: mol_weight, dtype: bool
```

use comparisons
(*e.g.* <, >, ==, !=)
to construct a
boolean *pd.Series*

...then use it to
extract those rows
matching value *True*

```
>>> aa[ aa['mol_weight'] > 150 ]
               name abbr mol_formula  mol_weight  hydrophob
code
R          arginine  Arg   C6H14N4O2      174.20        -14
H         histidine  His    C6H9N3O2      155.16        -31
F     phenylalanine  Phe    C9H11NO2      165.19         97
W        tryptophan  Trp  C11H12N2O2      204.23         97
Y          tyrosine  Tyr    C9H11NO3      181.19         63
```

the *isin()* method
can be used to find
entries that match
a provided set of values

```
>>> codon[ codon['code'].isin(['A','C']) ]
   codon abbr code
36   GCA  Ala    A
37   GCC  Ala    A
38   GCG  Ala    A
39   GCT  Ala    A
57   TGC  Cys    C
59   TGT  Cys    C
```

sorting

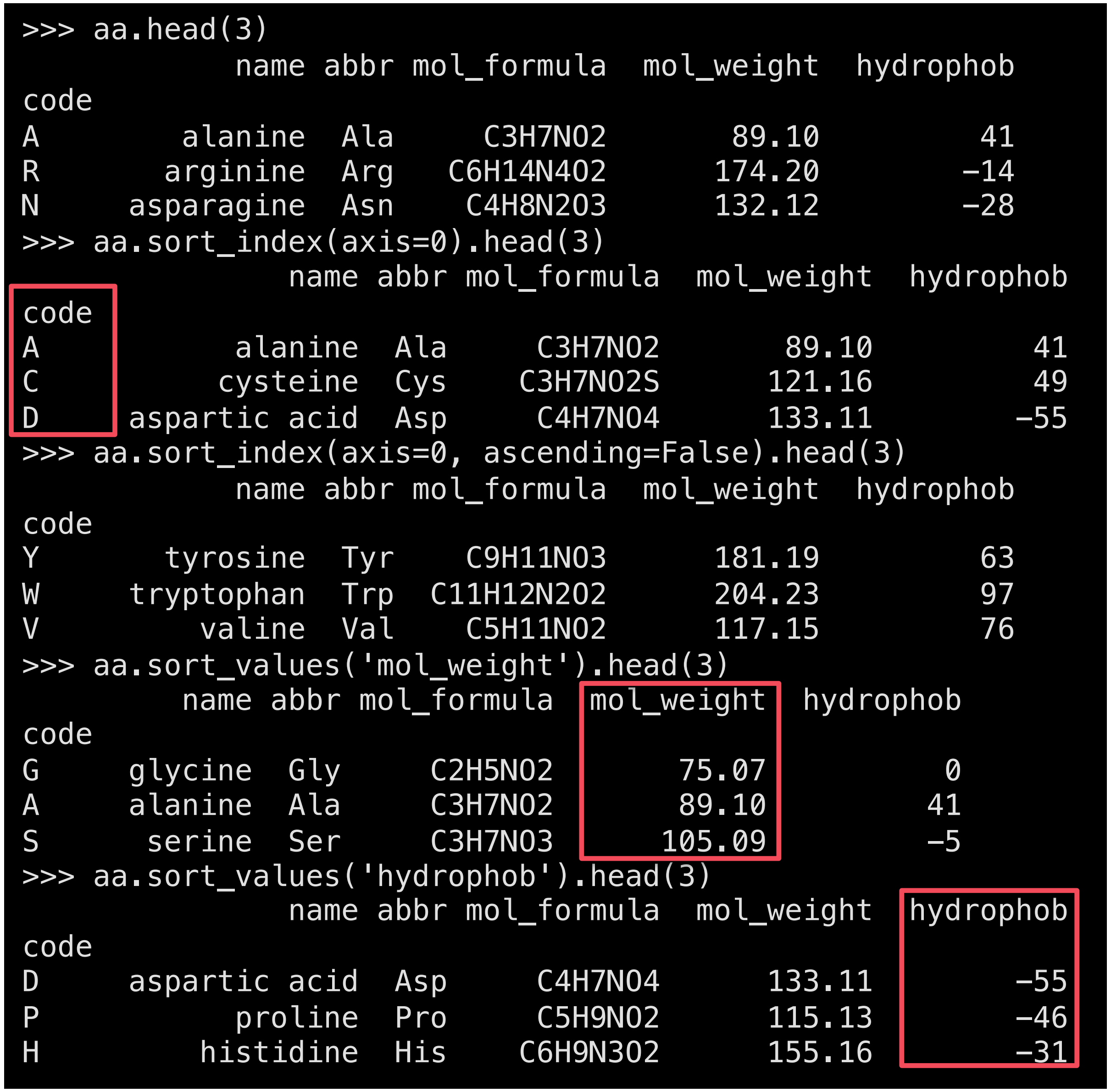
head shows first
N rows

sort_index against row
labels, then call *head(3)*

reverse *sort_index*

sort_values against
mol_weight values

sort_values against
hydrophob values



```
>>> aa.head(3)
                 name abbr mol_formula  mol_weight  hydrophob
code
A             alanine  Ala     C3H7NO2       89.10         41
R            arginine  Arg    C6H14N4O2     174.20        -14
N          asparagine  Asn     C4H8N2O3     132.12        -28
>>> aa.sort_index(axis=0).head(3)
                     name abbr mol_formula  mol_weight  hydrophob
code
A                 alanine  Ala     C3H7NO2       89.10         41
C                cysteine  Cys    C3H7NO2S      121.16         49
D          aspartic acid  Asp     C4H7NO4      133.11        -55
>>> aa.sort_index(axis=0, ascending=False).head(3)
                 name abbr mol_formula  mol_weight  hydrophob
code
Y            tyrosine  Tyr    C9H11NO3      181.19         63
W          tryptophan  Trp  C11H12N2O2      204.23         97
V              valine  Val    C5H11NO2      117.15         76
>>> aa.sort_values('mol_weight').head(3)
              name abbr mol_formula  mol_weight  hydrophob
code
G          glycine  Gly     C2H5NO2       75.07          0
A          alanine  Ala     C3H7NO2       89.10         41
S           serine  Ser     C3H7NO3      105.09         -5
>>> aa.sort_values('hydrophob').head(3)
                     name abbr mol_formula  mol_weight  hydrophob
code
D          aspartic acid  Asp     C4H7NO4      133.11        -55
P                proline  Pro     C5H9NO2      115.13        -46
H              histidine  His    C6H9N3O2      155.16        -31
```

helper methods

many helper methods are provided to summarize values in containers

```
>>> codon.value_counts('code')
code
S    6
R    6
L    6
A    4
V    4
T    4
G    4
P    4
O    3
I    3
Q    2
Y    2
N    2
C    2
K    2
H    2
F    2
E    2
D    2
W    1
M    1
dtype: int64
```

value_counts

```
>>> aa.quantile([0.05, 0.20, 0.5, 0.80, 0.95])
      mol_weight  hydrophob
0.05     88.3985     -46.45
0.20    116.7460     -24.00
0.50    132.6150      10.50
0.80    157.1660      80.20
0.95    182.3420      99.05
```

quantiles

```
>>> aa.loc[:,'mol_weight':'hydrophob'].rank()
      mol_weight  hydrophob
code
A            2.0       12.0
R           18.0        6.0
N           10.0        4.0
D           11.0        1.0
C            7.0       13.0
E           14.0       10.0
Q           12.0        7.0
G            1.0        9.0
H           16.0        3.0
I            8.5       19.0
L            8.5       20.0
K           13.0        5.0
M           15.0       15.0
F           17.0       17.5
P            4.0        2.0
S            3.0        8.0
T            6.0       11.0
W           20.0       17.5
Y           19.0       14.0
V            5.0       16.0
```

rank

apply statements

```
>>> df = pd.DataFrame(np.random.randn(3, 2),index=list('ABC'),
columns=list('XY'))
>>> df
          X         Y
A -0.885028  1.710104
B  0.712194 -1.530990
C -0.848945 -0.646218
>>> df.apply(lambda x: x+10)
           X          Y
A   9.114972  11.710104
B  10.712194   8.469010
C   9.151055   9.353782
>>> df.apply(lambda x: sum(x), axis=0)
X   -1.021779
Y   -0.467105
dtype: float64
>>> df.apply(lambda x: sum(x), axis=1)
A    0.825076
B   -0.818796
C   -1.495163
dtype: float64
```

apply a function iteratively against
each row (*axis=0*) or each column (*axis=1)*

melting

```
>>> aa.head(3)
        name abbr code mol_formula  mol_weight  hydrophob
0    alanine  Ala    A     C3H7NO2       89.10         41
1   arginine  Arg    R   C6H14N4O2      174.20        -14
2  asparagine  Asn    N    C4H8N2O3      132.12        -28
```

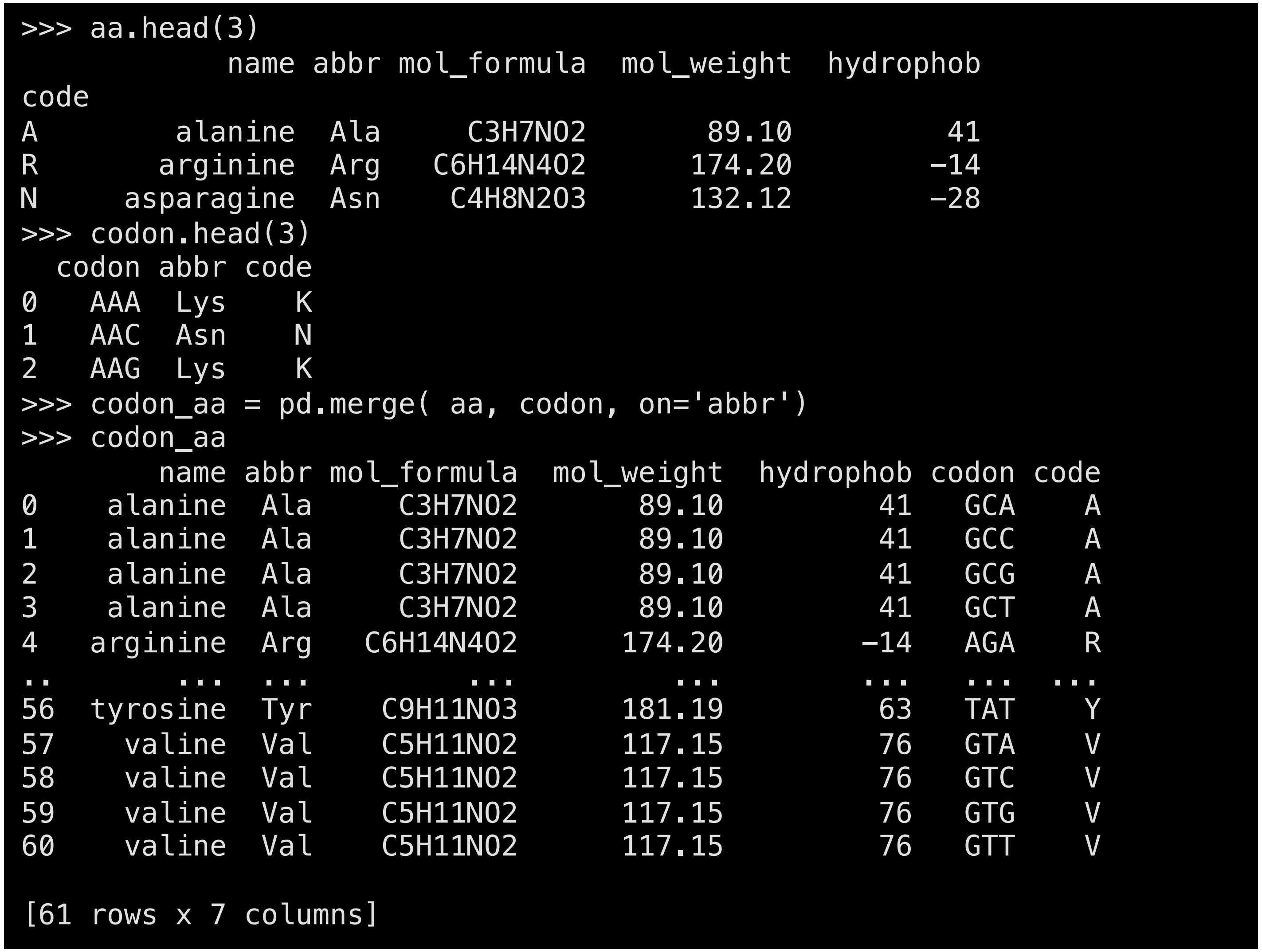
Standard format stores on AA per row,
multiple variables per AA

```
>>> pd.melt(aa, id_vars='abbr')
 abbr   variable          value
0   Ala       name        alanine
1   Arg       name       arginine
2   Asn       name     asparagine
3   Asp       name  aspartic acid
4   Cys       name       cysteine
..  ...        ...            ...
75  Ser  hydrophob             -5
76  Thr  hydrophob             13
77  Trp  hydrophob             97
78  Tyr  hydrophob             63
79  Val  hydrophob             76

[80 rows x 3 columns]
```

Melted format stores one variable per AA per row,
multiple rows per AA

merging

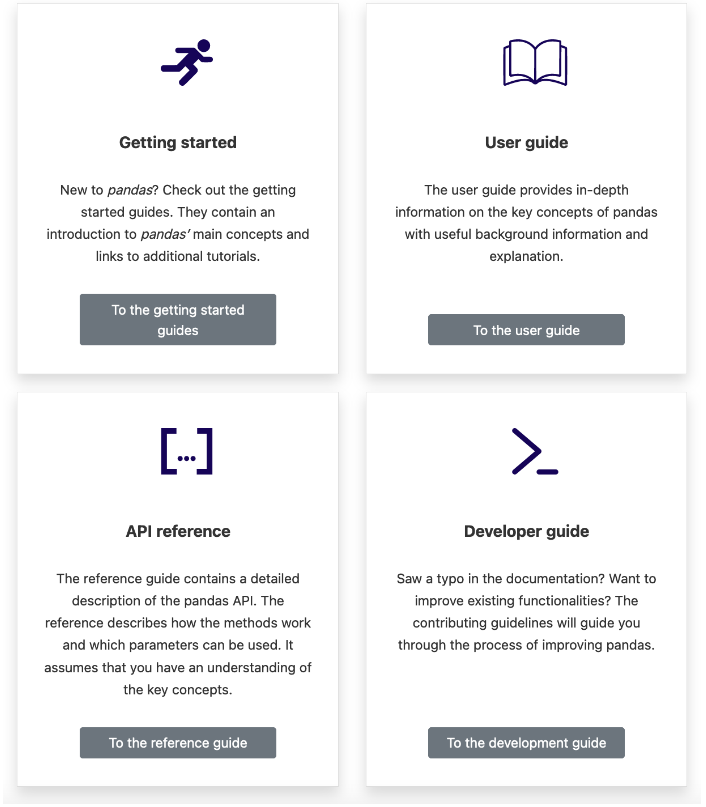


```
>>> aa.head(3)
            name abbr mol_formula  mol_weight  hydrophob
code
A        alanine  Ala     C3H7NO2       89.10         41
R       arginine  Arg   C6H14N4O2      174.20        -14
N     asparagine  Asn    C4H8N2O3      132.12        -28
>>> codon.head(3)
  codon abbr code
0   AAA  Lys    K
1   AAC  Asn    N
2   AAG  Lys    K
>>> codon_aa = pd.merge( aa, codon, on='abbr')
>>> codon_aa
        name abbr mol_formula  mol_weight  hydrophob codon code
0    alanine  Ala     C3H7NO2       89.10         41   GCA    A
1    alanine  Ala     C3H7NO2       89.10         41   GCC    A
2    alanine  Ala     C3H7NO2       89.10         41   GCG    A
3    alanine  Ala     C3H7NO2       89.10         41   GCT    A
4   arginine  Arg   C6H14N4O2      174.20        -14   AGA    R
..       ...  ...         ...         ...        ...   ...  ...
56  tyrosine  Tyr    C9H11NO3      181.19         63   TAT    Y
57    valine  Val    C5H11NO2      117.15         76   GTA    V
58    valine  Val    C5H11NO2      117.15         76   GTC    V
59    valine  Val    C5H11NO2      117.15         76   GTG    V
60    valine  Val    C5H11NO2      117.15         76   GTT    V

[61 rows x 7 columns]
```

pandas documentation

official documentation



Getting started

New to *pandas*? Check out the getting started guides. They contain an introduction to *pandas'* main concepts and links to additional tutorials.

To the getting started guides

User guide

The user guide provides in-depth information on the key concepts of pandas with useful background information and explanation.

To the user guide

API reference

The reference guide contains a detailed description of the pandas API. The reference describes how the methods work and which parameters can be used. It assumes that you have an understanding of the key concepts.

To the reference guide

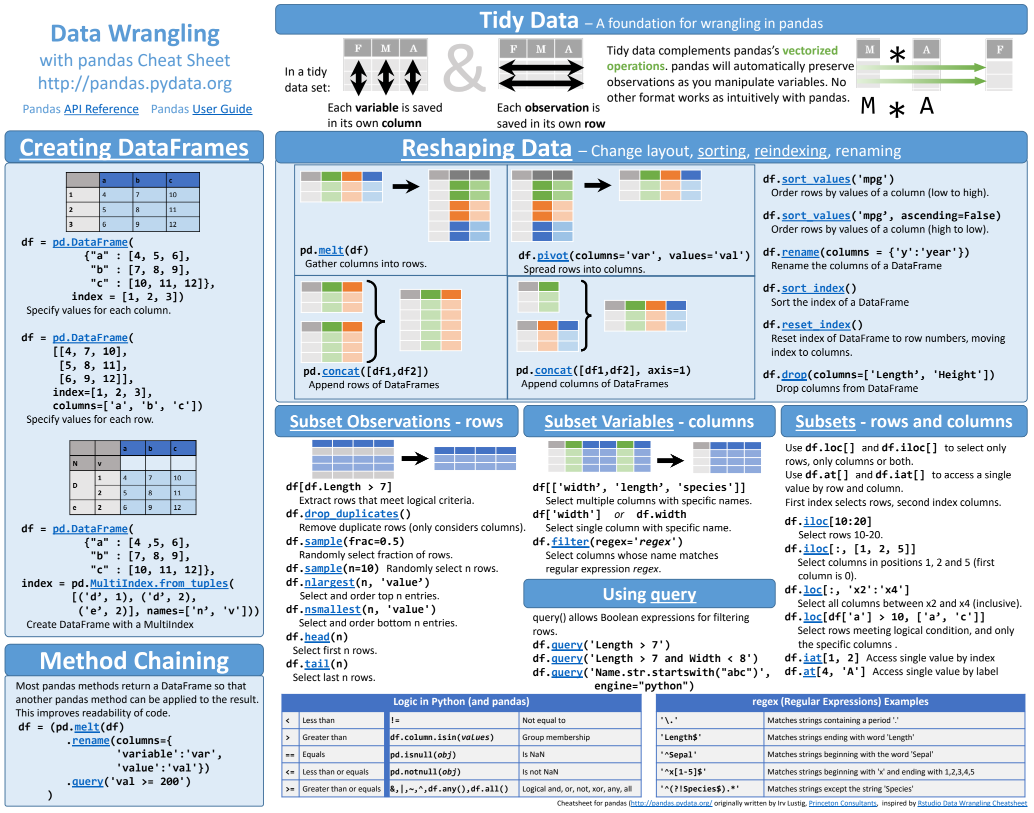
Developer guide

Saw a typo in the documentation? Want to improve existing functionalities? The contributing guidelines will guide you through the process of improving pandas.

To the development guide

https://pandas.pydata.org/docs/

cheat sheet



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

Pandas API Reference   Pandas User Guide

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

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 layout, sorting, reindexing, renaming

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

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

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

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

`df.head(n)` Select first n rows.

`df.tail(n)` Select last n rows.

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*.

Using query

query() allows Boolean expressions for filtering rows.

```
df.query('Length > 7')
df.query('Length > 7 and Width < 8')
df.query('Name.str.startswith("abc")',
                engine="python")
```

Subsets - rows and columns

Use `df.loc[]` and `df.iloc[]` to select only rows, only columns or both.
Use `df.at[]` and `df.iat[]` to access a single value by row and column.
First index selects rows, second index columns.

`df.iloc[10:20]` Select rows 10-20.

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

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

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

`df.iat[1, 2]` Access single value by index

`df.at[4, 'A']` Access single value by label

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

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'

Cheatsheet for pandas (http://pandas.pydata.org/ originally written by Irv Lustig, Princeton Consultants, inspired by Rstudio Data Wrangling Cheatsheet

https://pandas.pydata.org/Pandas_Cheat_Sheet.pdf

Overview for Lab 21