Pandas Basics

Cheat Sheet

BecomingHuman.AI

DataCamp

Use the following import convention: >>> import pandas as pd

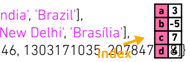
The Pandas library is built on NumPy and provides easy-to-use data structures and data analysis tools for the Python programming language.

Pandas Data Structures

Series

A one-dimensional labeled array a capable of holding any data type

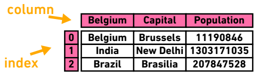
```
>>> s = pd.Series([3, -5, 7, 4], index=['a', 'b', 'c', 'd'])
```



a	3
b	-5
c	7
d	4

Data Frame

A two-dimensional labeled data structure with columns of potentially different types



	Belgium	Capital	Population
0	Belgium	Brussels	11190846
1	India	New Delhi	1303171035
2	Brazil	Brasilia	207847528

```
>>> data = {'Country': ['Belgium', 'India', 'Brazil'],
            'Capital': ['Brussels', 'New Delhi', 'Brasília'],
            'Population': [11190846, 1303171035, 207847528]}
>>> df = pd.DataFrame(data,
                      columns=['Country', 'Capital', 'Population'])
```

Dropping

>>> s.drop(['a', 'c'])	**Drop values from rows (axis=0)**
>>> df.drop('Country', axis=1)	**Drop values from columns(axis=1)**

Sort & Rank

>>> df.sort_index()	**Sort by labels along an axis**
>>> df.sort_values(by='Country')	**Sort by the values along an axis**
>>> df.rank()	**Assign ranks to entries**

Retrieving Series/ DataFrame Information

>>> df.shape	**(rows,columns)**
>>> df.index	**Describe index**
>>> df.columns	**Describe DataFrame columns**
>>> df.info()	**Info on DataFrame**
>>> df.count()	**Number of non-NA values**

Summary

>>> df.sum()	**Sum of values**
>>> df.cumsum()	**Cummulative sum of values**
>>> df.min()/df.max()	**Minimum/maximum values**
>>> df.idxmin()/df.idxmax()	**Minimum/Maximum index value**
>>> df.describe()	**Summary statistics**
>>> df.mean()	**Mean of values**
>>> df.median()	**Median of values**

Selection

Also see NumPy Arrays

Getting

```
>>> s['b']
 -5
>>> df[1:]
   Country Capital      Population
1  India    New Delhi  1303171035
2  Brazil   Brasília    207847528
```

>>> s['b']	**Get one element**
>>> df[1:]	**Get subset of a DataFrame**

Selecting, Boolean Indexing & Setting

By Position

```
>>> df.iloc[[0],[0]]
 'Belgium'
>>> df.iat([0],[0])
 'Belgium'
```

Select single value by row & column

By Label

```
>>> df.loc[[0], ['Country']]
 'Belgium'
>>> df.at([0], ['Country']) 'Belgium'
```

Select single value by row & column labels

By Label/Position

```
>>> df.ix[2]
  Country       Brazil
  Capital       Brasília
  Population    207847528
```

Select single row of subset of rows

```
>>> df.ix[:,'Capital']
  0 Brussels
  1 New Delhi
  2 Brasília
```

Select a single column of subset of columns

```
>>> df.ix[1,'Capital']
 'New Delhi'
```

Select rows and columns

Boolean Indexing

>>> s[~(s > 1)]	**Series s where value is not** >1
>>> s[(s < -1) | (s > 2)]	**s where value is** <-1 **or** >2
>>> df[df['Population']>1200000000]	**Use filter to adjust DataFrame**

Setting

>>> s['a'] = 6	**Set index a of Series s to 6**

Asking For Help

```
>>> help(pd.Series.loc)
```

Applying Functions

```
>>> f = lambda x: x*2
```

>>> df.apply(f)	**Apply function**
>>> df.applymap(f)	**Apply function element-wise**

Data Alignment

Internal Data Alignment

NA values are introduced in the indices that don't overlap:

```
>>> s3 = pd.Series([7, -2, 3], index=['a', 'c', 'd'])
>>> s + s3
  a 10.0
  b NaN
  c 5.0
  d 7.0
```

Arithmetic Operations with Fill Methods

You can also do the internal data alignment yourself with the help of the fill methods:

```
>>> s.add(s3, fill_value=0)
  a 10.0
  b -5.0
  c 5.0
  d 7.0
>>> s.sub(s3, fill_value=2)
>>> s.div(s3, fill_value=4)
```

I/O

Read and Write to CSV

```
>>> pd.read_csv('file.csv', header=None, nrows=5)
>>> df.to_csv('myDataFrame.csv')
```

Read and Write to Excel

```
>>> pd.read_excel('file.xlsx')
>>> pd.to_excel('dir/myDataFrame.xlsx', sheet_name='Sheet1')
```

Read multiple sheets from the same file

```
>>> xlsx = pd.ExcelFile('file.xls')
>>> df = pd.read_excel(xlsx, 'Sheet1')
```

Read and Write to SQL Query or Database Table

```
>>> from sqlalchemy import create_engine
>>> engine = create_engine('sqlite:///:memory:')
>>> pd.read_sql("SELECT * FROM my_table;", engine)
>>> pd.read_sql_table('my_table', engine)
>>> pd.read_sql_query("SELECT * FROM my_table;", engine)
```

read_sql()is **a convenience wrapper around** read_sql_table() and read_sql_query()

```
>>> pd.to_sql('myDf', engine)
```

Pandas
Cheat Sheet

BecomingHuman.AI

Pandas Data Structures

Pivot

```
>>> df3= df2.pivot(index='Date',                          Spread rows into columns
                   columns='Type',
                   values='Value')
```

	Date	Type	Value
0	2016-03-01	a	11.432
1	2016-03-02	b	13.031
2	2016-03-01	c	20.784
3	2016-03-03	a	99.906
4	2016-03-02	a	1.303
5	2016-03-03	c	20.784

Type	a	b	c
Date			
2016-03-01	11.432	NaN	20.784
2016-03-02	1.303	13.031	NaN
2016-03-03	99.906	NaN	20.784

Pivot Table

```
>>> df4 = pd.pivot_table(df2,                             Spread rows into columns
                         values='Value',
                         index='Date',
                         columns='Type'])
```

		0	1
1	5	0.233482	0.390959
2	4	0.184713	0.237102
3	3	0.433522	0.429401

Unstacked

1	5	0	0.233482
		1	0.390959
2	4	0	0.184713
		1	0.237102
3	3	0	0.433522
		1	0.429401

Stacked

Melt

```
>>> pd.melt(df2,                                          Gather columns into rows
            id_vars=["Date"],
            value_vars=["Type", "Value"],
            value_name="Observations")
```

	Date	Type	Value
0	2016-03-01	a	11.432
1	2016-03-02	b	13.031
2	2016-03-01	c	20.784
3	2016-03-03	a	99.906
4	2016-03-02	a	1.303
5	2016-03-03	c	20.784

	Date	Variable	Observations
0	2016-03-01	Type	a
1	2016-03-02	Type	b
2	2016-03-01	Type	c
3	2016-03-03	Type	a
4	2016-03-02	Type	a
5	2016-03-03	Type	c
6	2016-03-01	Value	11.432
7	2016-03-02	Value	13.031
8	2016-03-01	Value	20.784
9	2016-03-03	Value	99.906
10	2016-03-02	Value	1.303
11	2016-03-03	Value	20.784

Advanced Indexing

Also see NumPy Arrays

Selecting

```
>>> df3.loc[:,(df3>1).any()]                        Select cols with any vals >1
>>> df3.loc[:,(df3>1).all()]                          Select cols with vals > 1
>>> df3.loc[:,df3.isnull().any()]                       Select cols with NaN
>>> df3.loc[:,df3.notnull().all()]                    Select cols without NaN
```

Indexing With isin

```
>>> df[(df.Country.isin(df2.Type))]                      Find same elements
>>> df3.filter(items="a","b"])                              Filter on values
>>> df.select(lambda x: not x%5)                     Select specific elements
```

Where

```
>>> s.where(s > 0)                                          Subset the data
```

Query

```
>>> df6.query('second > first')                           Query DataFrame
```

Setting/Resetting Index

```
>>> df.set_index('Country')                                  Set the index
>>> df4 = df.reset_index()                                 Reset the index
>>> df = df.rename(index=str,                             Rename DataFrame
                   columns={"Country":"cntry",
                            "Capital":"cptl",
                            "Population":"ppltn"})
```

Reindexing

```
>>> s2 = s.reindex(['a','c','d','e','b'])
```

Forward Filling

```
>>> df.reindex(range(4),
               method='ffill')
Country Capital Population
 0  Belgium    Brussels     11190846
 1  India      New Delhi    1303171035
 2  Brazil     Brasília     207847528
 3  Brazil     Brasília     207847528
```

Forward Filling

```
>>> s3 = s.reindex(range(5),
                   method='bfill')
  0   3
  1   3
  2   3
  3   3
  4   3
```

MultiIndexing

```
>>> arrays = [np.array([1,2,3]),
              np.array([5,4,3])]
>>> df5 = pd.DataFrame(np.random.rand(3, 2), index=arrays)
>>> tuples = list(zip(*arrays))
>>> index = pd.MultiIndex.from_tuples(tuples,
                                      names=['first', 'second'])
>>> df6 = pd.DataFrame(np.random.rand(3, 2), index=index)
>>> df2.set_index(["Date", "Type"])
```

Duplicate Data

```
>>> s3.unique()                                        Return unique values
>>> df2.duplicated('Type')                                Check duplicates
>>> df2.drop_duplicates('Type', keep='last')               Drop duplicates
>>> df.index.duplicated()                                  Drop duplicates
```

Grouping Data

Aggregation

```
>>> df2.groupby(by=['Date','Type']).mean()
>>> df4.groupby(level=0).sum()
>>> df4.groupby(level=0).agg({'a':lambda x:sum(x)/len(x), 'b': np.sum})
```

Transformation

```
>>> customSum = lambda x: (x+x%2)
>>> df4.groupby(level=0).transform(customSum)
```

Missing Data

```
>>> df.dropna()                                             Drop NaN value
>>> df3.fillna(df3.mean())         Fill NaN values with a predetermined value
>>> df2.replace("a", "f")                           Replace values with others
```

Combining Data

data1

X1	X2
a	11.432
b	1.303
c	99.906

data2

X1	X3
a	20.784
b	NaN
d	20.784

Pivot

```
>>> pd.merge(data1,
             data2,
             how='left',
             on='X1')
```

X1	X2	X3
a	11.432	20.784
b	1.303	NaN
c	99.906	NaN

```
>>> pd.merge(data1,
             data2,
             how='right',
             on='X1')
```

X1	X2	X3
a	11.432	20.784
b	1.303	NaN
d	NaN	20.784

```
>>> pd.merge(data1,
             data2,
             how='inner',
             on='X1')
```

X1	X2	X3
a	11.432	20.784
b	1.303	NaN

```
>>> pd.merge(data1,
             data2,
             how='outer',
             on='X1')
```

X1	X2	X3
a	11.432	20.784
b	1.303	NaN
c	99.906	NaN
d	NaN	

Join

```
>>> data1.join(data2, how='right')
```

Concatenate

Vertical

```
>>> s.append(s2)
```

Horizontal/Vertical

```
>>> pd.concat([s,s2],axis=1, keys=['One','Two'])
>>> pd.concat([data1, data2], axis=1, join='inner')
```

Dates

```
>>> df2['Date']= pd.to_datetime(df2['Date'])
>>> df2['Date']= pd.date_range('2000-1-1', periods=6,
                               freq='M')
>>> dates = [datetime(2012,5,1), datetime(2012,5,2)]
>>> index = pd.DatetimeIndex(dates)
>>> index = pd.date_range(datetime(2012,2,1), end, freq='BM')
```

Visualization

```
>>> import matplotlib.pyplot as plt
```

```
>>> s.plot()
>>> plt.show()
```

```
>>> df2.plot()
>>> plt.show()
```

Data Wrangling with pandas Cheat Sheet

BecomingHuman.AI

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')
)
```

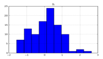
Windows

df.expanding()
Return an Expanding object allowing summary functions to be applied cumulatively.

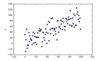
df.rolling(n)
Return a Rolling object allowing summary functions to be applied to windows of length n.

Windows

df.plot.hist()
Histogram for each column

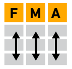


df.plot.scatter(x='w',y='h')
Scatter chart using pairs of points



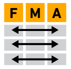
Tidy Data A foundation for wrangling in pandas

In a tidy data set:



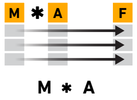
Each **variable** is saved in its own **column**

&

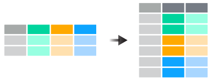


Each **observation** is saved in its own **row**

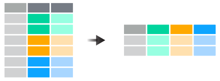
Tidy data complements pandas's vectorized operations. pandas will automatically preserve observations as you manipulate variables. No other format works as intuitively with pandas



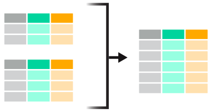
Reshaping Data Change the layout of a data set



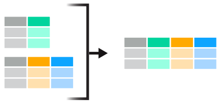
pd.melt(df)
Gather columns into rows.



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).

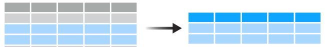
df.rename(columns = {'y':'year'})
Rename the columns of a DataFrame

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.

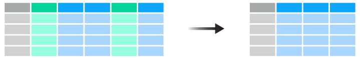
df.nlargest(n, 'value')
Select and order top n entries.

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
==	Equal to	pd.isnull(obj)	Is NaN
<=	Less than or equal to	pd.notnull(obj)	Is not NaN
>=	Greater than or equal to	&,|,~,^,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.

Logic in Python (and pandas)

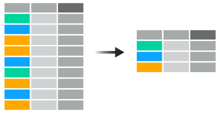
'\.'	Matches strings containing a period '.'
'Length$	Matches strings ending with word 'Length'
'^Sepal'	Matches strings beginning with the word 'Sepal'
'^x[1-5]$'	Matches strings beginning with 'x' and ending with 1,2,3,4,5
'^(?!Species$).*'	Matches strings except the string 'Species'

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 .

Windows



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

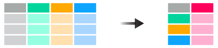
Summarise Data

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.

Combine Data Sets

ydf

x1	x2
A	1
B	2
C	3

+ zdf

x1	x2
B	2
C	3
D	4

=

adf

x1	x2
A	1
B	2
C	3

+ bdf

x1	x3
A	T
B	F
C	T

=

Set 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)

Standard Joins

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

dpd.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

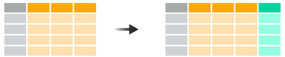
adf[~adf.x1.isin(bdf.x1)]
All rows in adf that do not have a match in bdf

Handling Missing Data

df.dropna()
Drop rows with any column having NA/null data.

df.fillna(value)

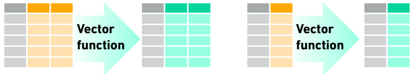
Make New Columns



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 allcolumns 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.