zekeLabs

Pandas for Data Wrangling & Statistical Modeling

Learning made Simpler !

www.zekeLabs.com

Agenda

- Introduction to Pandas
- Data Wrangling with Pandas
- Plotting & Visualization
- Statistical Data Modeling

Introduction to Pandas

- Introduction to Pandas
- Series and DataFrame objects
- Importing data
- Indexing, data selection and subsetting
- Hierarchical indexing
- Reading and writing files
- Date/time types
- String Operations
- Missing data
- Data summarization

Introduction to Pandas

- Open Source, High Performance, Easy-to-use data structure.
- Library for Data munging, preparation, analysis & modeling.
- Alternative to excel sheet
- Handle Time Series data
- Reads from different data formats
- Mutable in contents & size
- IO Tools to load from flat files, HDF5 etc.

Series

- Single vector of data like NumPy but with index
- Imagine series as one column of table

```
s = pd.Series(data=[1,2,3,4,5,6], index=['a','b','c','c','e','f'])
```

```
s
```

```
a    1
b    2
c    3
c    4
e    5
f    6
dtype: int64
```

Series Access

```
s = pd.Series(data=[1,2,3,4,5,6], index=['a','b','c','c','e','f'])
```

```
s['a':'c']
```

```
a    1
b    2
c    3
c    4
dtype: int64
```

```
s[1:3]
```

```
b    2
c    3
dtype: int64
```

```
s['d':]
```

```
e    5
f    6
dtype: int64
```

DataFrames

- Data Structure to store, view, manipulate multivariate data
- Tabular data structure
- Series represent univariate data
- Combine different series and create a dataframe

DataFrames

- Data Structure to store, view, manipulate multivariate data
- Tabular data structure
- Series represent univariate data
- Combine different series and create a dataframe

DataFrames - Creation from Series

```
ser1 = pd.Series([100,200,300,400], index=['a','b','c','d'])
ser2 = pd.Series([222,333,444,555,666], index=['a','c','d','b','e'])
```

```
df = pd.DataFrame({'ser1':ser1,'ser2':ser2})
```

```
df
```

	ser1	ser2
a	100.0	222
b	200.0	555
c	300.0	333
d	400.0	444
e	NaN	666

DataFrames - Creation

```
df = pd.DataFrame({'A':[1,2,3,4,5],
                   'B':[6,7,8,9,10],
                   'C':[7,5,4,3,2],
                  })
df
```

	A	B	C
0	1	6	7
1	2	7	5
2	3	8	4
3	4	9	3
4	5	10	2

Importing data

- Data should be loaded before anything could be done on it.

Format Type	Data Description	Reader	Writer
text	CSV	read_csv	to_csv
text	JSON	read_json	to_json
text	HTML	read_html	to_html
text	Local clipboard	read_clipboard	to_clipboard
binary	MS Excel	read_excel	to_excel
binary	HDF5 Format	read_hdf	to_hdf
binary	Feather Format	read_feather	to_feather
binary	Msgpack	read_msgpack	to_msgpack
binary	Stata	read_stata	to_stata
binary	SAS	read_sas	
binary	Python Pickle Format	read_pickle	to_pickle
SQL	SQL	read_sql	to_sql
SQL	Google Big Query	read_gbq	to_gbq

Reading Data

```
df = pd.read_csv('Data/credit-risk-data/cs-training.csv')
```

```
df.count()
```

```
Unnamed: 0                                150000
SeriousDlqin2yrs                          150000
RevolvingUtilizationOfUnsecuredLines      150000
age                                       150000
NumberOfTime30-59DaysPastDueNotWorse      150000
```

Reading Large Data

```
df = pd.read_csv('Data/credit-risk-data/cs-training.csv', chunksize=1000)
```

```
for d in df:
    print (d.count())
```

```
Unnamed: 0                              1000
SeriousDlqin2yrs                        1000
RevolvingUtilizationOfUnsecuredLines    1000
```

```
df = pd.read_csv('Data/credit-risk-data/cs-training.csv', nrows=1000)
```

```
df.count()
```

```
Unnamed: 0                              1000
SeriousDlqin2yrs                        1000
RevolvingUtilizationOfUnsecuredLines    1000
age                                     1000
```

Exploring Data

```
df.head()
```

	Unnamed: 0	SeriousDlqin2yrs	RevolvingUtilizationOfUnsecuredLines	age	NumberOfTime30-59DaysPastDueNotWorse	DebtRatio	MonthlyIncome	NumberOfOpenCreditLinesAndLoans	NumberO
0	1	1	0.766127	45	2	0.802982	9120.0	13	
1	2	0	0.957151	40	0	0.121876	2600.0	4	
2	3	0	0.658180	38	1	0.085113	3042.0	2	
3	4	0	0.233810	30	0	0.036050	3300.0	5	
4	5	0	0.907239	49	1	0.024926	63588.0	7	

```
df.tail()
```

	Unnamed: 0	SeriousDlqin2yrs	RevolvingUtilizationOfUnsecuredLines	age	NumberOfTime30-59DaysPastDueNotWorse	DebtRatio	MonthlyIncome	NumberOfOpenCreditLinesAndLoans	NumberC
995	996	0	0.357684	32	1	710.000000	NaN	6	
996	997	0	0.102951	43	1	0.252275	10000.0	18	
997	998	0	1.000000	60	1	10.171276	3000.0	4	
998	999	0	0.040283	54	0	0.135554	12400.0	11	
999	1000	0	0.352989	59	1	0.439556	13433.0	18	

Exploring Data -2

```
df = pd.read_csv('Data/credit-risk-data/cs-training.csv', nrows=1000, index_col='Unnamed: 0')
```

```
df.count()
```

```
SeriousDlqin2yrs                        1000
RevolvingUtilizationOfUnsecuredLines    1000
age                                     1000
NumberOfTime30-59DaysPastDueNotWorse    1000
DebtRatio                               1000
MonthlyIncome                            819
NumberOfOpenCreditLinesAndLoans         1000
NumberOfTimes90DaysLate                 1000
NumberRealEstateLoansOrLines            1000
NumberOfTime60-89DaysPastDueNotWorse    1000
NumberOfDependents                       967
dtype: int64
```

Exploring Data -3

```
df.describe()
```

	Unnamed: 0	SeriousDlqin2yrs	RevolvingUtilizationOfUnsecuredLines	age	NumberOfTime30-59DaysPastDueNotWorse	DebtRatio	Monthly
count	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	81
mean	500.500000	0.057000	4.717846	51.793000	0.266000	353.772448	661
std	288.819436	0.231959	98.649119	15.174466	0.771907	1167.736841	881
min	1.000000	0.000000	0.000000	22.000000	0.000000	0.000000	
25%	250.750000	0.000000	0.032362	40.000000	0.000000	0.167349	330
50%	500.500000	0.000000	0.159672	52.000000	0.000000	0.360422	521
75%	750.250000	0.000000	0.533372	62.250000	0.000000	0.750515	833
max	1000.000000	1.000000	2340.000000	97.000000	10.000000	15466.000000	20833

Exploring Data -3

```
]: df.describe()
```

	Unnamed: 0	SeriousDlqin2yrs	RevolvingUtilizationOfUnsecuredLines	age	NumberOfTime30-59DaysPastDueNotWorse	DebtRatio	Monthly
count	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	819
mean	500.500000	0.057000	4.717846	51.793000	0.266000	353.772448	661
std	288.819436	0.231959	98.649119	15.174466	0.771907	1167.736841	881
min	1.000000	0.000000	0.000000	22.000000	0.000000	0.000000	0
25%	250.750000	0.000000	0.032362	40.000000	0.000000	0.167349	330
50%	500.500000	0.000000	0.159672	52.000000	0.000000	0.360422	521
75%	750.250000	0.000000	0.533372	62.250000	0.000000	0.750515	833
max	1000.000000	1.000000	2340.000000	97.000000	10.000000	15466.000000	20833

Access Columns

```
movie_data[['Adam Cohen','Brenda Peterson']]
```

	Adam Cohen	Brenda Peterson
Goodfellas	4.5	2.0
Raging Bull	NaN	1.0
Roman Holiday	3.0	4.5
Scarface	3.0	1.5
The Apartment	1.0	5.0
Vertigo	3.5	3.0

```
movie_data['Adam Cohen']
```

```
Goodfellas       4.5
Raging Bull      NaN
Roman Holiday    3.0
Scarface         3.0
The Apartment    1.0
Vertigo          3.5
Name: Adam Cohen, dtype: float64
```

Access Rows By Index & Index-Values

```
movie_data = pd.read_json('https://raw.githubusercontent.com/zekelabs/machine-learning-for-beginners/master/movie.json.txt')
```

```
movie_data
```

	Adam Cohen	Bill Duffy	Brenda Peterson	Chris Duncan	Clarissa Jackson	David Smith	Julie Hammel	Samuel Miller
Goodfellas	4.5	4.5	2.0	NaN	2.5	4.5	3.0	5.0
Raging Bull	NaN	NaN	1.0	4.5	4.0	3.0	NaN	5.0
Roman Holiday	3.0	NaN	4.5	NaN	1.5	NaN	4.5	1.0
Scarface	3.0	5.0	1.5	NaN	4.5	4.5	2.5	3.5
The Apartment	1.0	1.0	5.0	1.5	1.0	1.0	NaN	1.0
Vertigo	3.5	4.5	3.0	NaN	5.0	4.0	NaN	NaN

Access Rows By Index & Index-Values

```
movie_data.iloc[2:5]
```

	Adam Cohen	Bill Duffy	Brenda Peterson	Chris Duncan	Clarissa Jackson	David Smith	Julie Hammel	Samuel Miller
Roman Holiday	3.0	NaN	4.5	NaN	1.5	NaN	4.5	1.0
Scarface	3.0	5.0	1.5	NaN	4.5	4.5	2.5	3.5
The Apartment	1.0	1.0	5.0	1.5	1.0	1.0	NaN	1.0

```
movie_data.loc['Goodfellas':'Scarface']
```

	Adam Cohen	Bill Duffy	Brenda Peterson	Chris Duncan	Clarissa Jackson	David Smith	Julie Hammel	Samuel Miller
Goodfellas	4.5	4.5	2.0	NaN	2.5	4.5	3.0	5.0
Raging Bull	NaN	NaN	1.0	4.5	4.0	3.0	NaN	5.0
Roman Holiday	3.0	NaN	4.5	NaN	1.5	NaN	4.5	1.0
Scarface	3.0	5.0	1.5	NaN	4.5	4.5	2.5	3.5

Filtering Rows

```
movie_data[movie_data['Adam Cohen'] > 3.5]
```

	Adam Cohen	Bill Duffy	Brenda Peterson	Chris Duncan	Clarissa Jackson	David Smith	Julie Hammel	Samuel Miller
Goodfellas	4.5	4.5	2.0	NaN	2.5	4.5	3.0	5.0

```
movie_data[(movie_data['Adam Cohen'] > 2.5) & (movie_data['David Smith'] > 2.5)]
```

	Adam Cohen	Bill Duffy	Brenda Peterson	Chris Duncan	Clarissa Jackson	David Smith	Julie Hammel	Samuel Miller
Goodfellas	4.5	4.5	2.0	NaN	2.5	4.5	3.0	5.0
Scarface	3.0	5.0	1.5	NaN	4.5	4.5	2.5	3.5
Vertigo	3.5	4.5	3.0	NaN	5.0	4.0	NaN	NaN

Missing Values

```
movie_data[movie_data['Chris Duncan'].notnull()]
```

	Adam Cohen	Bill Duffy	Brenda Peterson	Chris Duncan	Clarissa Jackson	David Smith	Julie Hammel	Samuel Miller
Raging Bull	NaN	NaN	1.0	4.5	4.0	3.0	NaN	5.0
The Apartment	1.0	1.0	5.0	1.5	1.0	1.0	NaN	1.0

```
movie_data[movie_data['Chris Duncan'].isnull()]
```

	Adam Cohen	Bill Duffy	Brenda Peterson	Chris Duncan	Clarissa Jackson	David Smith	Julie Hammel	Samuel Miller
Goodfellas	4.5	4.5	2.0	NaN	2.5	4.5	3.0	5.0
Roman Holiday	3.0	NaN	4.5	NaN	1.5	NaN	4.5	1.0
Scarface	3.0	5.0	1.5	NaN	4.5	4.5	2.5	3.5
Vertigo	3.5	4.5	3.0	NaN	5.0	4.0	NaN	NaN

Missing Values during load

```
]: import pandas as pd
   data = pd.read_csv('Data/credit-risk-data/cs-training.csv', index_col='Unnamed: 0', na_values=[0])
```

```
]: data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 150000 entries, 1 to 150000
Data columns (total 11 columns):
SeriousDlqin2yrs                          10026 non-null float64
RevolvingUtilizationOfUnsecuredLines      139122 non-null float64
age                                       149999 non-null float64
NumberOfTime30-59DaysPastDueNotWorse      23982 non-null float64
DebtRatio                                 145887 non-null float64
MonthlyIncome                             118635 non-null float64
NumberOfOpenCreditLinesAndLoans           148112 non-null float64
NumberOfTimes90DaysLate                   8338 non-null float64
NumberRealEstateLoansOrLines              93812 non-null float64
NumberOfTime60-89DaysPastDueNotWorse      7604 non-null float64
NumberOfDependents                        59174 non-null float64
dtypes: float64(11)
memory usage: 13.7 MB
```

Handling Missing Values

- Datasets in real world will have missing values
- If only few values of a column is present, we might drop the entire column
- If only few rows are missing, we might drop the rows
- We can’t afford to dropping lot of rows, it like reducing the dataset.

Handling Missing Values

- Fillna - Filling missing values

```
>>> df
     A    B    C  D
0  NaN  2.0  NaN  0
1  3.0  4.0  NaN  1
2  NaN  NaN  NaN  5
3  NaN  3.0  NaN  4
```

```
>>> df.fillna(0)
    A   B   C   D
0   0.0 2.0 0.0 0
1   3.0 4.0 0.0 1
2   0.0 0.0 0.0 5
3   0.0 3.0 0.0 4
```

```
>>> df.fillna(method='ffill')
    A   B   C   D
0   NaN 2.0 NaN 0
1   3.0 4.0 NaN 1
2   3.0 4.0 NaN 5
3   3.0 3.0 NaN 4
```

```
>>> values = {'A': 0, 'B': 1, 'C': 2, 'D': 3}
>>> df.fillna(value=values)
    A   B   C   D
0   0.0 2.0 2.0 0
1   3.0 4.0 2.0 1
2   0.0 1.0 2.0 5
3   0.0 3.0 2.0 4
```

Handling Missing Values

- dropna - dropping based on missing values

```
>>> df.dropna(axis=1, how='all')
     A    B  D
0  NaN  2.0  0
1  3.0  4.0  1
2  NaN  NaN  5
```

```
>>> df.dropna(axis=1, how='any')
   D
0  0
1  1
2  5
```

```
>>> df.dropna(axis=0, how='all')
     A    B   C  D
0  NaN  2.0 NaN  0
1  3.0  4.0 NaN  1
2  NaN  NaN NaN  5
```

```
>>> df.dropna(thresh=2)
     A    B   C  D
0  NaN  2.0 NaN  0
1  3.0  4.0 NaN  1
```

Handling Missing Values

- replace - Replace values given in ‘to_replace’ with ‘value’.

```
df.replace(np.nan, -2)
```

	A	B
0	1.0	2.0
1	2.0	3.0
2	-2.0	4.0
3	2.0	-2.0

```
df = pd.DataFrame({'A':[1,2,-1,2], 'B':[2,3,4,-1]})
```

```
df.replace(-1,np.nan)
```

	A	B
0	1.0	2.0
1	2.0	3.0
2	NaN	4.0
3	2.0	NaN

Duplicate Finding

```
df
```

	A	B	C
0	1	11	22
1	2	12	33
2	3	13	44
3	4	14	88
4	5	15	55
5	3	13	44
6	4	14	77

```
df[df.duplicated()]
```

	A	B	C
5	3	13	44

```
df[df.duplicated(subset=['A','B'])]
```

	A	B	C
5	3	13	44
6	4	14	77

Duplicate Dropping

```
df
```

	A	B	C
0	1	11	22
1	2	12	33
2	3	13	44
3	4	14	88
4	5	15	55
5	3	13	44
6	4	14	77

```
df.drop_duplicates(inplace=True, subset=['A','B'], keep='last')
```

```
df
```

	A	B	C
0	1	11	22
1	2	12	33
4	5	15	55
5	3	13	44
6	4	14	77

JSON Normalizing

- Handling semi-structured data

```python
with open('j.json') as json_data:
    data = json.load(json_data)
```

```python
data
```

```
[{'counties': [{'name': 'Dade', 'population': 12345},
   {'name': 'Palm Beach', 'population': 60000}],
  'info': {'governor': 'Rick Scott'},
  'shortname': 'FL',
  'state': 'Florida'}]
```

```python
from pandas.io.json import json_normalize
```

```python
json_normalize(data, 'counties', ['state', 'shortname',
                                  ['info','governor']])
```

	name	population	state	shortname	info.governor
0	Dade	12345	Florida	FL	Rick Scott
1	Palm Beach	60000	Florida	FL	Rick Scott

Working with Text Data

- Series with string data have '.str' as a module.
- Inside .str we have many string utility functions.

Method	Description
`cat()`	Concatenate strings
`split()`	Split strings on delimiter
`rsplit()`	Split strings on delimiter working from the end of the string
`get()`	Index into each element (retrieve i-th element)
`join()`	Join strings in each element of the Series with passed separator
`get_dummies()`	Split strings on the delimiter returning DataFrame of dummy variables
`contains()`	Return boolean array if each string contains pattern/regex
`replace()`	Replace occurrences of pattern/regex/string with some other string or the return value of a callable given the occurrence
`repeat()`	Duplicate values (`s.str.repeat(3)` equivalent to `x * 3`)
`pad()`	Add whitespace to left, right, or both sides of strings
`center()`	Equivalent to `str.center`
`ljust()`	Equivalent to `str.ljust`
`rjust()`	Equivalent to `str.rjust`
`zfill()`	Equivalent to `str.zfill`

Working with Text Data - 2

```
titanic_data = pd.read_csv('https://raw.githubusercontent.com/zekelabs/machine-learning-for-beginners/master/data/
```

```
titanic_data['namelen'] = titanic_data.Name.str.len()
```

```
titanic_data.head()
```

	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	namelen
PassengerId												
1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S	23
2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C	51
3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S	22
4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S	44
5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S	24

Working with Text Data - 3

`wrap()`	Split long strings into lines with length less than a given width
`slice()`	Slice each string in the Series
`slice_replace()`	Replace slice in each string with passed value
`count()`	Count occurrences of pattern
`startswith()`	Equivalent to `str.startswith(pat)` for each element
`endswith()`	Equivalent to `str.endswith(pat)` for each element
`findall()`	Compute list of all occurrences of pattern/regex for each string
`match()`	Call `re.match` on each element, returning matched groups as list
`extract()`	Call `re.search` on each element, returning DataFrame with one row for each element and one column for each regex capture group
`extractall()`	Call `re.findall` on each element, returning DataFrame with one row for each match and one column for each regex capture group
`len()`	Compute string lengths
`strip()`	Equivalent to `str.strip`
`rstrip()`	Equivalent to `str.rstrip`
`lstrip()`	Equivalent to `str.lstrip`

Handling DateTime data

- Time series data appears very often in datasets
- Loading columns as datetime when loading file
- Parsing Datetimes
- Display datetime with time zones
- Rounding datetimes
- Filtering data between certain datetime
- Creating ranges

Loading date

```
churn_data = pd.read_csv('churn.csv.txt', parse_dates=['last_trip_date','signup_date'])
```

```
churn_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50000 entries, 0 to 49999
Data columns (total 12 columns):
avg_dist                  50000 non-null float64
avg_rating_by_driver      49799 non-null float64
avg_rating_of_driver      41878 non-null float64
avg_surge                 50000 non-null float64
city                      50000 non-null object
last_trip_date            50000 non-null datetime64[ns]
phone                     49604 non-null object
signup_date               50000 non-null datetime64[ns]
```

Parsing DateTime

```
df = pd.DataFrame({'time':['31/Aug/2015:23:49:01 +0000','31/Aug/2015:23:49:01 +0000']})
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2 entries, 0 to 1
Data columns (total 1 columns):
time     2 non-null object
dtypes: object(1)
memory usage: 96.0+ bytes
```

Parsing DateTime - 2

```
df['time_ft'] = pd.to_datetime(df.time, format='%d/%b/%Y:%H:%M:%S +0000', utc=True)
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2 entries, 0 to 1
Data columns (total 2 columns):
time       2 non-null object
time_ft    2 non-null datetime64[ns, UTC]
dtypes: datetime64[ns, UTC](1), object(1)
memory usage: 112.0+ bytes
```

```
df.head()
```

	time	time_ft
0	31/Aug/2015:23:49:01 +0000	2015-08-31 23:49:01+00:00
1	31/Aug/2015:23:49:01 +0000	2015-08-31 23:49:01+00:00

Datetime with timezone

```
df.set_index('time_ft', inplace=True)
df.index = df.index.tz_convert('America/Los_Angeles')
df
```

	time
time_ft	
2015-08-31 16:49:01-07:00	31/Aug/2015:23:49:01 +0000
2015-08-31 15:49:01-07:00	31/Aug/2015:22:49:01 +0000

Datetime Rounding

```
df.index = df.index.floor('2H')
```

```
df
```

	time
time_ft	
2015-08-31 16:00:00-07:00	31/Aug/2015:23:49:01 +0000
2015-08-31 14:00:00-07:00	31/Aug/2015:22:49:01 +0000

Datetime Offset

```
df['time'] = pd.to_datetime(df.time, format='%d/%b/%Y:%H:%M:%S +0000', utc=True)
```

```
df['time'] + pd.DateOffset(weeks=1)
```

```
time_ft
2015-08-31 16:00:00-07:00   2015-09-07 23:49:01+00:00
2015-08-31 14:00:00-07:00   2015-09-07 22:49:01+00:00
Name: time, dtype: datetime64[ns, UTC]
```

Datetime Filter

```
data
```

date	battle_deaths
2015-05-01 18:47:05.069722	34
2015-05-01 18:47:05.119994	25
2014-05-02 18:47:05.178768	26
2014-05-02 18:47:05.230071	15
2014-05-02 18:47:05.230071	15
2014-05-02 18:47:05.280592	14
2014-05-03 18:47:05.332662	26
2014-05-03 18:47:05.385109	25
2014-05-04 18:47:05.436523	62
2014-05-04 18:47:05.486877	41

```
data['2014-05-04']
```

date	battle_deaths
2014-05-04 18:47:05.436523	62
2014-05-04 18:47:05.486877	41

```
data['2015']
```

date	battle_deaths
2015-05-01 18:47:05.069722	34
2015-05-01 18:47:05.119994	25

Data Wrangling with Pandas

- Reshaping DataFrame objects
- Pivoting
- Alignment
- Data aggregation and GroupBy operations
- Merging and joining DataFrame objects

GroupBy : split-apply-combine

Splitting data into groups based on certain criteria

Applying a function to each group independently

- Aggregate
- Transform
- Filtering

Combining the results into a data structure

GroupBy : splitting an object into groups

Concatenate

```
In [4]: frames = [df1, df2, df3]

In [5]: result = pd.concat(frames)
```

df1

	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1
2	A2	B2	C2	D2
3	A3	B3	C3	D3

df2

	A	B	C	D
4	A4	B4	C4	D4
5	A5	B5	C5	D5
6	A6	B6	C6	D6
7	A7	B7	C7	D7

df3

	A	B	C	D
8	A8	B8	C8	D8
9	A9	B9	C9	D9
10	A10	B10	C10	D10
11	A11	B11	C11	D11

Result

	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1
2	A2	B2	C2	D2
3	A3	B3	C3	D3
4	A4	B4	C4	D4
5	A5	B5	C5	D5
6	A6	B6	C6	D6
7	A7	B7	C7	D7
8	A8	B8	C8	D8
9	A9	B9	C9	D9
10	A10	B10	C10	D10
11	A11	B11	C11	D11

Concatenate

```
In [9]: result = pd.concat([df1, df4], axis=1, sort=False)
```

df1

	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1
2	A2	B2	C2	D2
3	A3	B3	C3	D3

df4

	B	D	F
2	B2	D2	F2
3	B3	D3	F3
6	B6	D6	F6
7	B7	D7	F7

Result

	A	B	C	D	B	D	F
0	A0	B0	C0	D0	NaN	NaN	NaN
1	A1	B1	C1	D1	NaN	NaN	NaN
2	A2	B2	C2	D2	B2	D2	F2
3	A3	B3	C3	D3	B3	D3	F3
6	NaN	NaN	NaN	NaN	B6	D6	F6
7	NaN	NaN	NaN	NaN	B7	D7	F7

Database-style DataFrame joining/merging

```
In [38]: left = pd.DataFrame({'key': ['K0', 'K1', 'K2', 'K3'],
   ....:                      'A': ['A0', 'A1', 'A2', 'A3'],
   ....:                      'B': ['B0', 'B1', 'B2', 'B3']})
   ....:

In [39]: right = pd.DataFrame({'key': ['K0', 'K1', 'K2', 'K3'],
   ....:                       'C': ['C0', 'C1', 'C2', 'C3'],
   ....:                       'D': ['D0', 'D1', 'D2', 'D3']})
   ....:

In [40]: result = pd.merge(left, right, on='key')
```

left

	key	A	B
0	K0	A0	B0
1	K1	A1	B1
2	K2	A2	B2
3	K3	A3	B3

right

	key	C	D
0	K0	C0	D0
1	K1	C1	D1
2	K2	C2	D2
3	K3	C3	D3

Result

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

Database-style DataFrame joining/merging

```
In [41]: left = pd.DataFrame({'key1': ['K0', 'K0', 'K1', 'K2'],
   ....:                      'key2': ['K0', 'K1', 'K0', 'K1'],
   ....:                      'A': ['A0', 'A1', 'A2', 'A3'],
   ....:                      'B': ['B0', 'B1', 'B2', 'B3']})
   ....:

In [42]: right = pd.DataFrame({'key1': ['K0', 'K1', 'K1', 'K2'],
   ....:                       'key2': ['K0', 'K0', 'K0', 'K0'],
   ....:                       'C': ['C0', 'C1', 'C2', 'C3'],
   ....:                       'D': ['D0', 'D1', 'D2', 'D3']})
   ....:

In [43]: result = pd.merge(left, right, on=['key1', 'key2'])
```

left

	key1	key2	A	B
0	K0	K0	A0	B0
1	K0	K1	A1	B1
2	K1	K0	A2	B2
3	K2	K1	A3	B3

right

	key1	key2	C	D
0	K0	K0	C0	D0
1	K1	K0	C1	D1
2	K1	K0	C2	D2
3	K2	K0	C3	D3

Result

	key1	key2	A	B	C	D
0	K0	K0	A0	B0	C0	D0
1	K1	K0	A2	B2	C1	D1
2	K1	K0	A2	B2	C2	D2

Join By Indexes

```
In [80]: result = left.join(right)
```

left

	A	B
K0	A0	B0
K1	A1	B1
K2	A2	B2

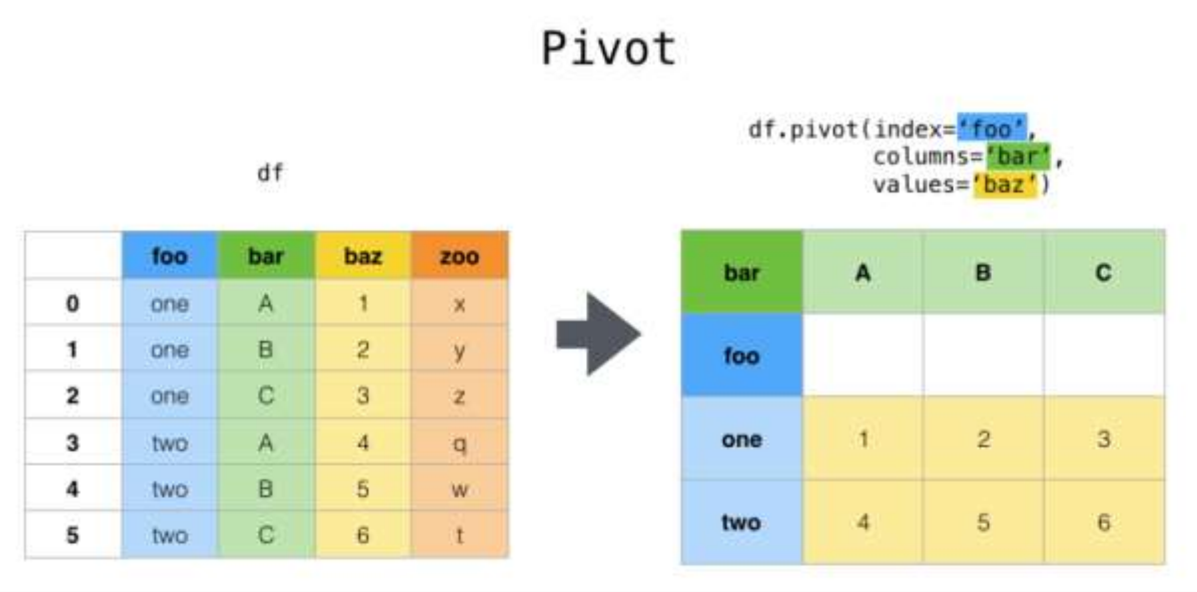
right

	C	D
K0	C0	D0
K2	C2	D2
K3	C3	D3

Result

	A	B	C	D
K0	A0	B0	C0	D0
K1	A1	B1	NaN	NaN
K2	A2	B2	C2	D2

Pivot



	foo	bar	baz	zoo
0	one	A	1	x
1	one	B	2	y
2	one	C	3	z
3	two	A	4	q
4	two	B	5	w
5	two	C	6	t

bar	A	B	C
foo			
one	1	2	3
two	4	5	6

Stacking

Pivot

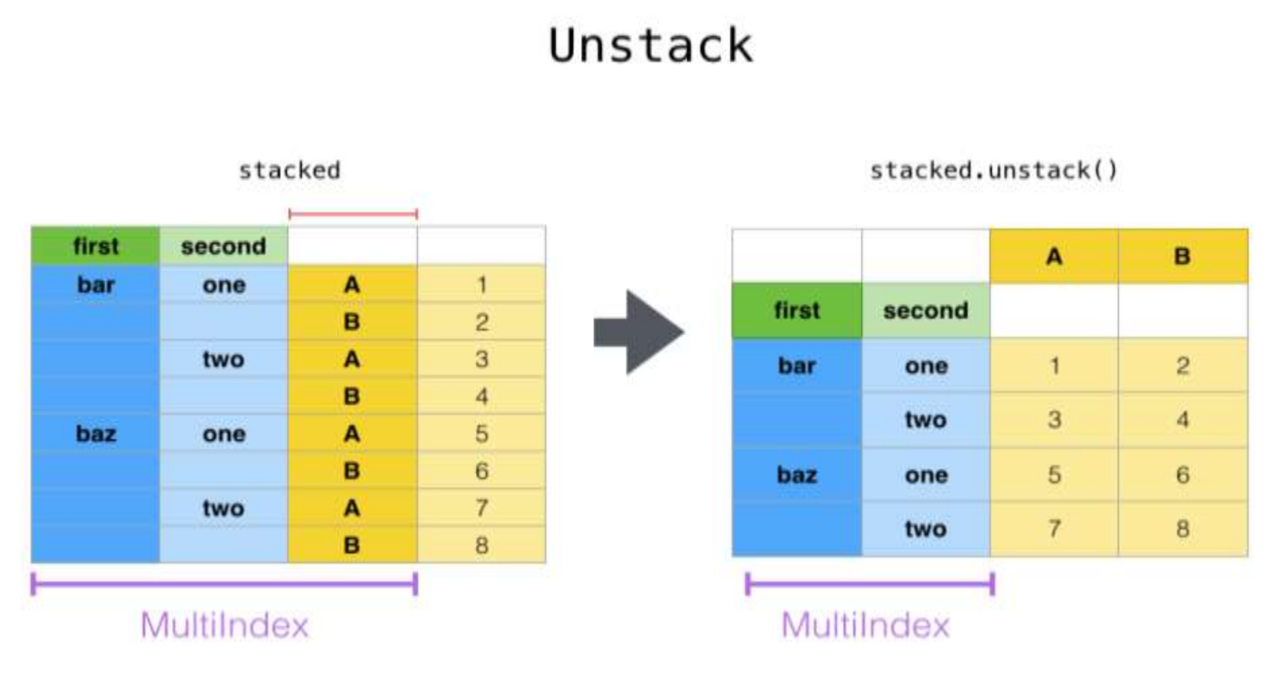
df

	foo	bar	baz	zoo
0	one	A	1	x
1	one	B	2	y
2	one	C	3	z
3	two	A	4	q
4	two	B	5	w
5	two	C	6	t

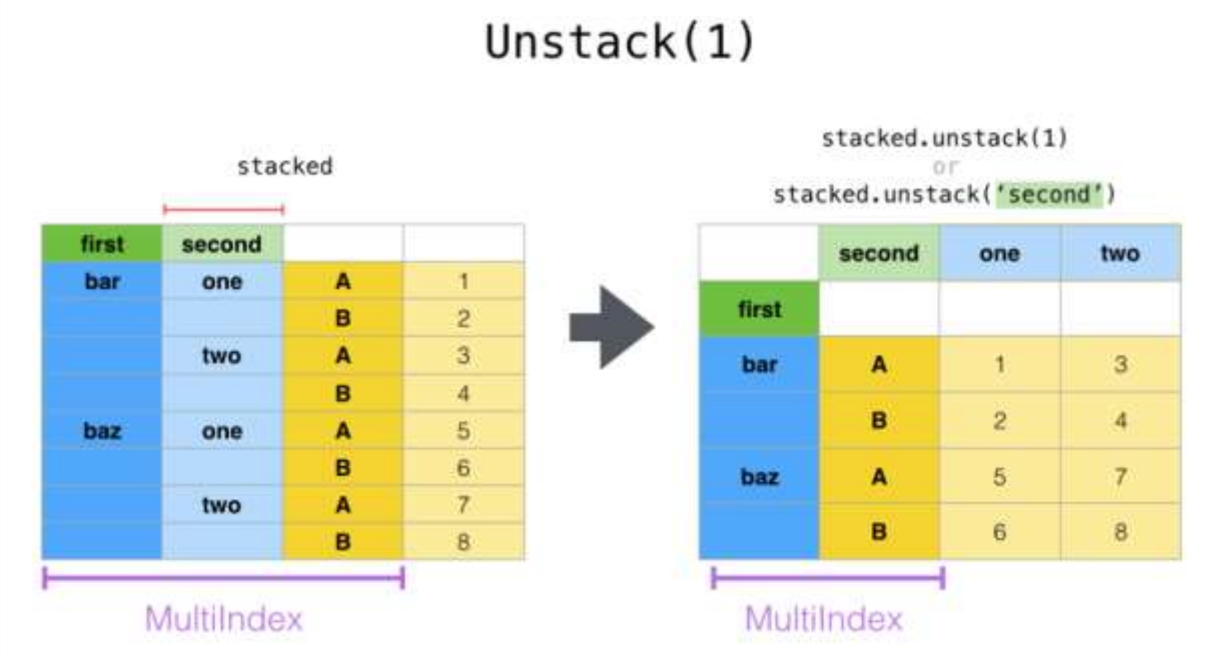
```
df.pivot(index='foo',
         columns='bar',
         values='baz')
```

bar	A	B	C
foo			
one	1	2	3
two	4	5	6

Unstacking

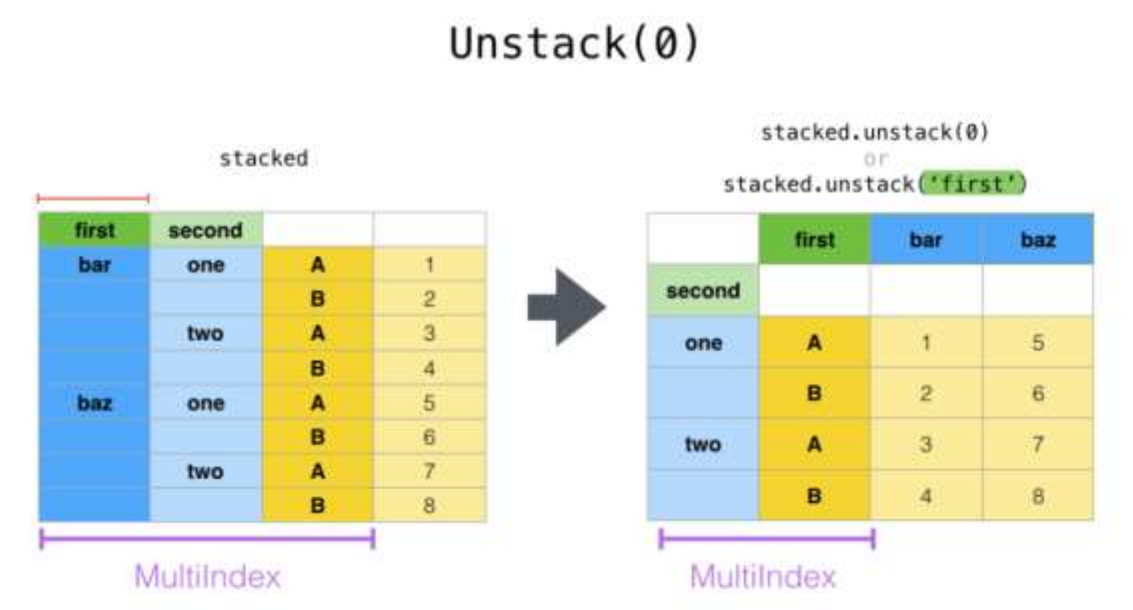

Unstack
stacked
first
second
bar
one
two
baz
one
two
A
B
A
B
A
B
A
B
1
2
3
4
5
6
7
8
MultiIndex
stacked.unstack()
A
B
first
second
bar
one
1
2
two
3
4
baz
one
5
6
two
7
8
MultiIndex

Unstacking



Unstack(1)
stacked
first
second
bar
one
A
1
B
2
two
A
3
B
4
baz
one
A
5
B
6
two
A
7
B
8
MultiIndex
stacked.unstack(1)
or
stacked.unstack('second')
second
one
two
first
bar
A
1
3
B
2
4
baz
A
5
7
B
6
8
MultiIndex

Unstacking



Unstack(0)
stacked
first
second
bar
one
A
1
B
2
two
A
3
B
4
baz
one
A
5
B
6
two
A
7
B
8
MultiIndex
stacked.unstack(0)
or
stacked.unstack('first')
first
bar
baz
second
one
A
1
5
B
2
6
two
A
3
7
B
4
8
MultiIndex

Melt

Melt

df3

	first	last	height	weight
0	John	Doe	5.5	130
1	Mary	Bo	6.0	150

```
df3.melt(id_vars=['first', 'last'])
```

	first	last	variable	value
0	John	Doe	height	5.5
1	Mary	Bo	height	6.0
2	John	Doe	weight	130
3	Mary	Bo	weight	150

Pivot Table

	Account	Name	Rep	Manager	Product	Quantity	Price	Status
0	714466	Trantow-Barrows	Craig Booker	Debra Henley	CPU	1	30000	presented
1	714466	Trantow-Barrows	Craig Booker	Debra Henley	Software	1	10000	presented
2	714466	Trantow-Barrows	Craig Booker	Debra Henley	Maintenance	2	5000	pending
3	737550	Fritsch, Russel and Anderson	Craig Booker	Debra Henley	CPU	1	35000	declined
4	146832	Kiehn-Spinka	Daniel Hilton	Debra Henley	CPU	2	65000	won

```
pd.pivot_table(df,
    index=["Manager","Status"],
    columns=["Product"],
    aggfunc=[np.sum],
    values=["Price"],
    fill_value=0,
    margins=True,
    dropna=True)
```

Can also use a dictionary:
aggfunc={"Quantity":len,
"Price":[np.sum,np.mean]}

		sum				
		Price				
	Product	CPU	Maintenance	Monitor	Software	All
Manager	Status					
Debra Henley	declined	70000	0	0	0	70000
	pending	40000	10000	0	0	50000
	presented	30000	0	0	20000	50000
	won	65000	0	0	0	65000
Fred Anderson	declined	65000	0	0	0	65000
	pending	0	5000	0	0	5000
	presented	30000	0	5000	10000	45000
	won	165000	7000	0	0	172000
All		465000	22000	5000	30000	522000

Computation

Splitting data into groups based on certain criteria

Applying a function to each group independently

- Aggregate
- Transform
- Filtering

Combining the results into a data structure

Computation Tool

- Statistical Functions
- Window Functions
- Aggregations

Plotting & Visualization

- Introduction of Matplotlib
- Time series plots
- Grouped plots
- Scatterplots
- Histograms
- Box-plot
- Pie Charts

I

Statistical Data Modeling

- Fitting data to probability distributions
- Linear models
- Spline models
- Time series analysis
- Bayesian models

Thank You !!!

THANK YOU

Let us know how can we help your organization to Upskill the employees to stay updated in the ever-evolving IT Industry.

Get in touch:

www.zekeLabs.com | +91-8095465880 | info@zekeLabs.com