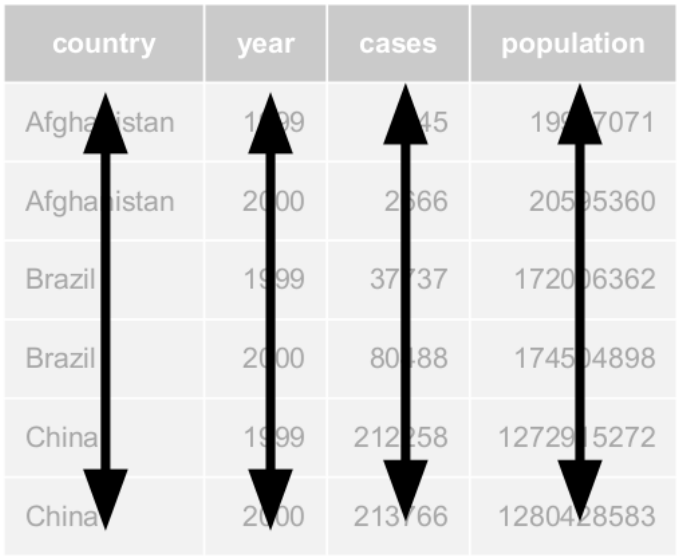
Pandas dataframes

Datos

country	year	cases	population
Afghanistan	1999	745	19987071
Afghanistan	2000	2666	20595360
Brazil	1999	37737	172006362
Brazil	2000	80488	174504898
China	1999	212258	1272915272
China	2000	213766	1280428583



Variables

Observations

Index

- Index
  df.index
- Set index to column
  df.set_index()
- Reset index of DataFrame to row numbers, moving index to columns.

  df.reset_index()

Columns

- Select multiple columns with specific names.

  df[['width','length','species']]
- Select single column with specific name.

  df['width'] or df.width
- Select columns whose name matches regular expression regex.

  df.filter(regex='regex')

Columns

- Rename the columns of a DataFrame

  df.rename(columns = {'y':'year'})
- Drop columns from DataFrame

  df.drop(columns=['Length','Height'])

Rows

- Select first n rows.

  df.head(n)
- Select last n rows.

  df.tail(n)
- Randomly select fraction of rows.

  df.sample(frac=0.1)
- Randomly select n rows.

  df.sample(n=10)

Rows

- Filter by condition

  df[df.Length > 7]
- Select and order top n entries.

  df.nlargest(5, 'Salary')
- Select and order bottom n entries

  df.nsmallest(n, 'value')

Sort

- Order rows by values of a column (low to high)
  df.sort_values('mpg')
- Order rows by values of a column (high to low).
  df.sort_values('mpg',ascending=False)
- Sort the index of a DataFrame
  df.sort_index()

Slicing

- Select all columns between x2 and x4 (inclusive).

  df.loc[‘Lebron James’,'x2':'x4']

- Select columns in positions 1, 2 and 5 (first column is 0).

  df.iloc[10:120,[1,2,5]]

- Select rows meeting logical condition, and only the specific columns .

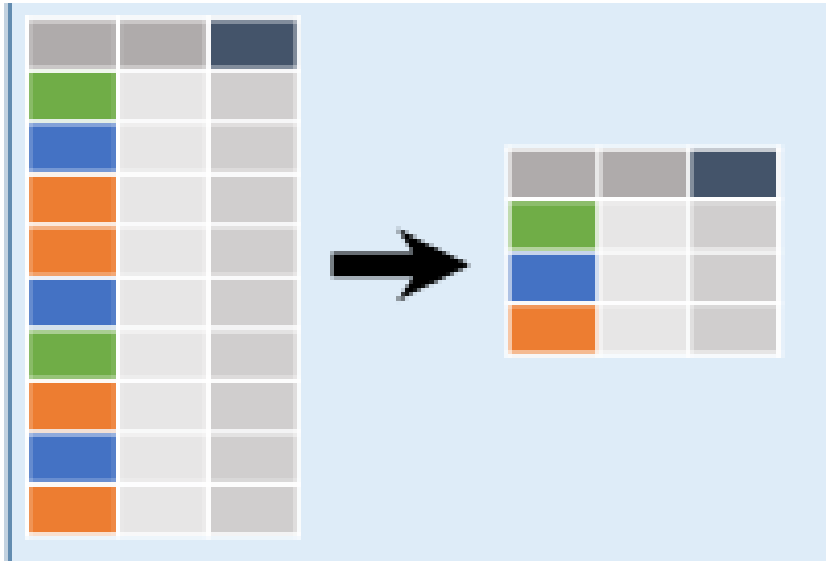
  df.loc[df['a'] > 10, ['a','c']]

Group by

- Return a GroupBy object, grouped by values

  df.groupby()
- Aggregate group using function

  agg(function)
- value_counts()



Iterable

- Columns

  df.iteritems()
- Rows

  df.iterrows()

Modify

- df.replace()
- df.apply()
- df.map()

https://www.delftstack.com/es/howto/python-pandas/pandas-replace-values-in-column/

Duplicates

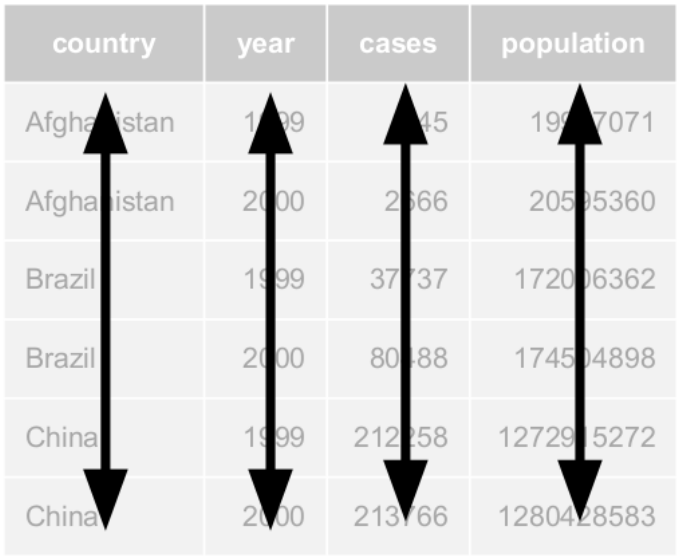
- Remove duplicate rows (only considers columns)

  df.drop_duplicates()

Pivot tables

Datos

country	year	cases	population
Afghanistan	1999	745	19987071
Afghanistan	2000	2666	20595360
Brazil	1999	37737	172006362
Brazil	2000	80488	174504898
China	1999	212258	1272915272
China	2000	213766	1280428583



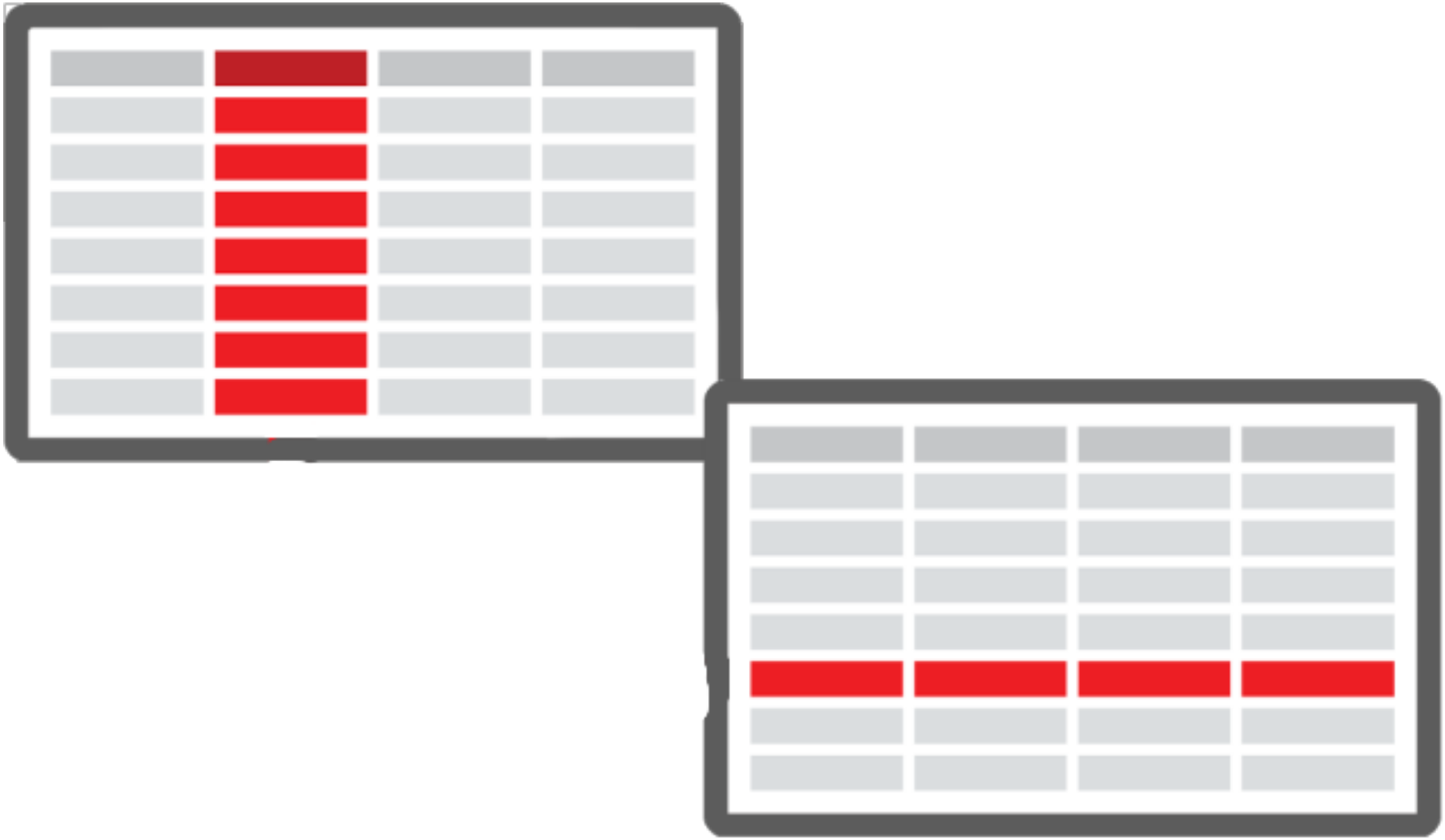
Variables

Observations

Datos

	country	1800	1801	1802	1803	1804	1805	1806	1807	1808	...	2009	2010	2011	2012	2013	2014	2015	2016	2017	2018
0	Afghanistan	28,2	28,2	28,2	28,2	28,2	28,2	28,1	28,1	28,1	...	55,7	56,2	56,7	57,2	57,7	57,8	57,9	58	58,4	58,7
1	Albania	35,4	35,4	35,4	35,4	35,4	35,4	35,4	35,4	35,4	...	75,9	76,3	76,7	77	77,2	77,4	77,6	77,7	77,9	78
2	Algeria	28,8	28,8	28,8	28,8	28,8	28,8	28,8	28,8	28,8	...	76,3	76,5	76,7	76,8	77	77,1	77,3	77,4	77,6	77,9
3	Angola	27	27	27	27	27	27	27	27	27	...	59,3	60,1	60,9	61,7	62,5	63,3	64	64,7	64,9	65,2
4	Antigua and Barbuda	33,5	33,5	33,5	33,5	33,5	33,5	33,5	33,5	33,5	...	76,9	76,8	76,9	77	77,3	77,1	77,2	77,3	77,4	77,6
5	Argentina	33,2	33,2	33,2	33,2	33,2	33,2	33,2	33,2	33,2	...	75,7	75,8	76	76,1	76,2	76,4	76,5	76,7	76,8	77
6	Armenia	34	34	34	34	34	34	34	34	34	...	73	73,3	73,8	74,3	75	75,4	75,4	75,7	75,8	76
7	Australia	34	34	34	34	34	34	34	34	34	...	81,8	82	82,2	82,3	82,5	82,6	82,6	82,5	82,7	82,9
8	Austria	34,4	34,4	34,4	34,4	34,4	34,4	34,4	34,4	34,4	...	80,3	80,5	80,7	80,9	81,1	81,3	81,4	81,5	81,7	81,8
9	Azerbaijan	29,2	29,2	29,2	29,2	29,2	29,2	29,2	29,2	29,2	...	68,8	69,1	69,9	70,2	71	71,5	71,8	72,1	72,2	72,3

pivot/unpivot



pivot/unpivot

	country	year	life_expectancy
0	Afghanistan	1800	28,2
1	Albania	1800	35,4
2	Algeria	1800	28,8
3	Angola	1800	27
4	Antigua and Barbuda	1800	33,5
5	Argentina	1800	33,2
6	Armenia	1800	34
7	Australia	1800	34
8	Austri		
9	Azerbaij		

	country	1800	1801	1802	1803	1804	1805	1806	1807	1808	...	2009	2010	2011	2012	2013	2014	2015	2016	2017	2018
0	Afghanistan	28,2	28,2	28,2	28,2	28,2	28,2	28,1	28,1	28,1	...	55,7	56,2	56,7	57,2	57,7	57,8	57,9	58	58,4	58,7
1	Albania	35,4	35,4	35,4	35,4	35,4	35,4	35,4	35,4	35,4	...	75,9	76,3	76,7	77	77,2	77,4	77,6	77,7	77,9	78
2	Algeria	28,8	28,8	28,8	28,8	28,8	28,8	28,8	28,8	28,8	...	76,3	76,5	76,7	76,8	77	77,1	77,3	77,4	77,6	77,9
3	Angola	27	27	27	27	27	27	27	27	27	...	59,3	60,1	60,9	61,7	62,5	63,3	64	64,7	64,9	65,2
4	Antigua and Barbuda	33,5	33,5	33,5	33,5	33,5	33,5	33,5	33,5	33,5	...	76,9	76,8	76,9	77	77,3	77,1	77,2	77,3	77,4	77,6
5	Argentina	33,2	33,2	33,2	33,2	33,2	33,2	33,2	33,2	33,2	...	75,7	75,8	76	76,1	76,2	76,4	76,5	76,7	76,8	77
6	Armenia	34	34	34	34	34	34	34	34	34	...	73	73,3	73,8	74,3	75	75,4	75,4	75,7	75,8	76
7	Australia	34	34	34	34	34	34	34	34	34	...	81,8	82	82,2	82,3	82,5	82,6	82,6	82,5	82,7	82,9
8	Austria	34,4	34,4	34,4	34,4	34,4	34,4	34,4	34,4	34,4	...	80,3	80,5	80,7	80,9	81,1	81,3	81,4	81,5	81,7	81,8
9	Azerbaijan	29,2	29,2	29,2	29,2	29,2	29,2	29,2	29,2	29,2	...	68,8	69,1	69,9	70,2	71	71,5	71,8	72,1	72,2	72,3

melt

- frame : DataFrame
- id_vars[tuple, list, or ndarray, optional] : Column(s) to use as identifier variables.
- value_vars[tuple, list, or ndarray, optional]: Column(s) to unpivot. If not specified, uses all columns that are not set as id_vars.
- var_name[scalar]: Name to use for the ‘variable’ column. If None it uses frame.columns.name or ‘variable’.
- value_name[scalar, default ‘value’]: Name to use for the ‘value’ column.
- col_level[int or string, optional]: If columns are a MultiIndex then use this level to melt.

pivot_table

- Index: Column to use to make new frame's index. If None, uses existing index.
- Columns: Column to use to make new frame's columns.
- Values: Column(s) to use for populating new frame's values. If not specified, all remaining columns will be used and the result will have hierarchically indexed columns.

pivot_table

```
1 vehicles.pivot_table(index=["Vehicle Class"], columns=["Cylinders"], values=["Combined MPG"], fill_value=0)
```

	2.0	3.0	4.0	5.0	6.0	8.0	10.0	12.0	16.0
Vehicle Class									
Compact Cars	0.0	36.333333	25.381877	21.943128	20.051046	16.536913	0.000000	13.757143	0
Large Cars	0.0	0.000000	24.275862	0.000000	20.379866	17.323529	0.000000	13.884058	0
Midsize Cars	0.0	0.000000	25.830698	19.710843	20.336047	16.837594	14.235294	13.050000	0
Midsize Station Wagons	0.0	0.000000	23.515924	20.229730	19.582822	16.590909	0.000000	0.000000	0
Midsize-Large Station Wagons	0.0	0.000000	20.769697	19.030303	19.397059	17.772727	0.000000	0.000000	0

Reshape

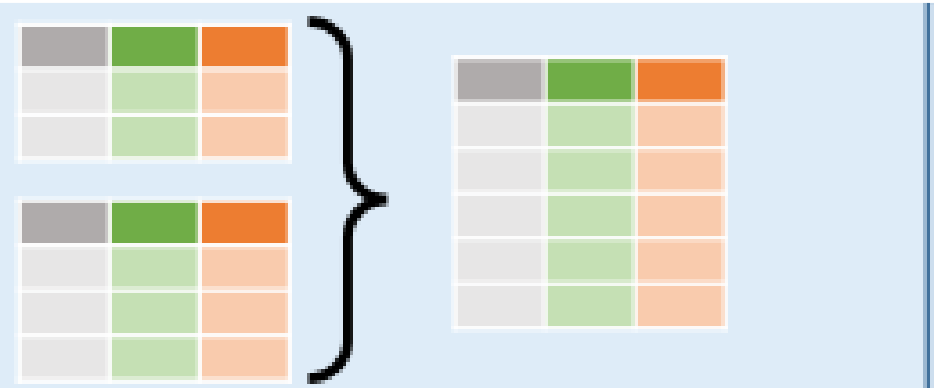
`pd.melt(df)`

Gather columns into rows.

`df.pivot(columns='var', values='val')`

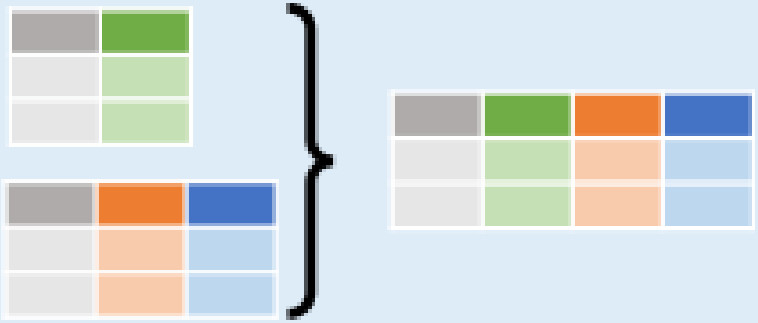
Spread rows into columns.

Concat



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

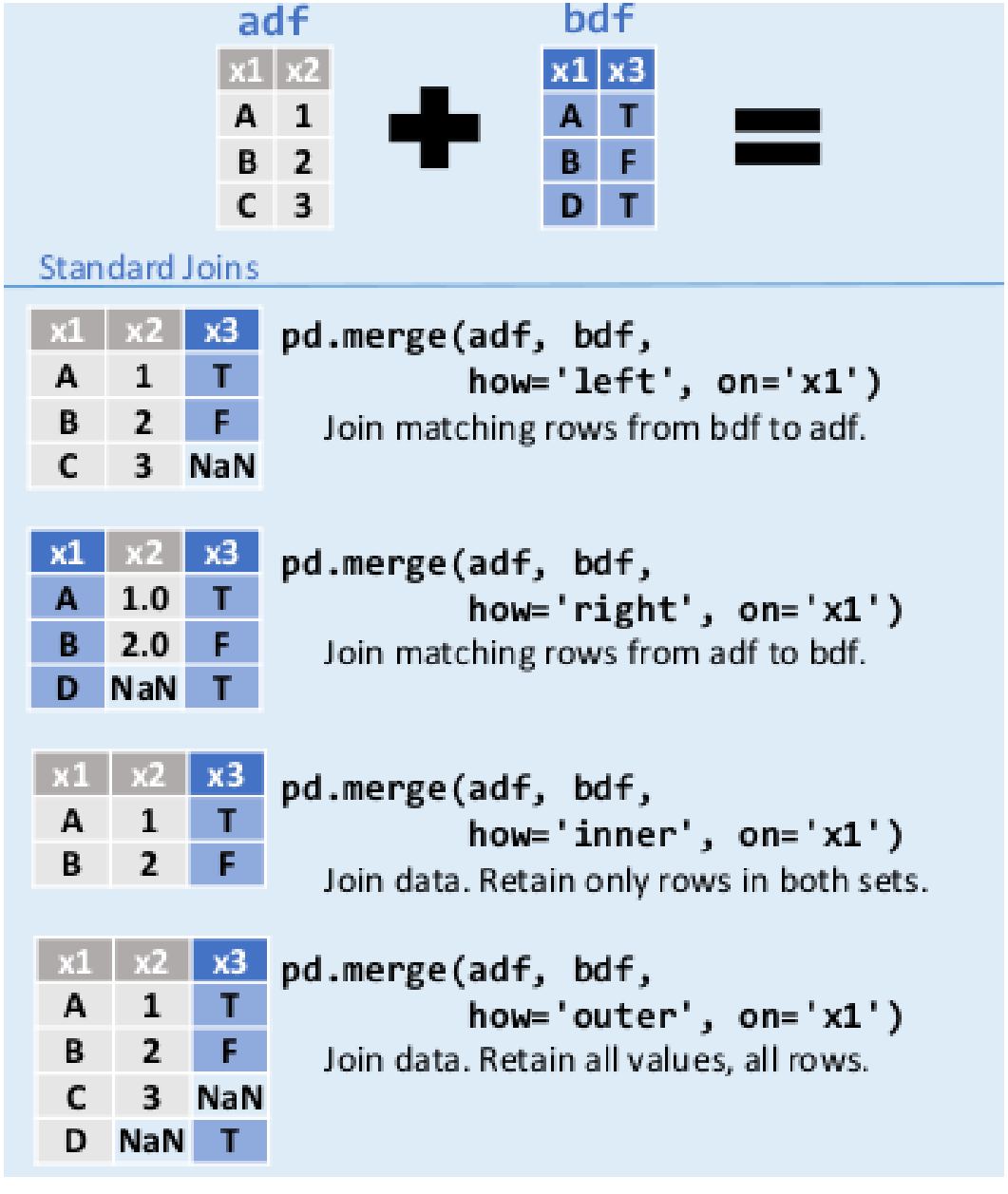
Append rows of DataFrames



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

Append columns of DataFrames

Merge


adf
x1 x2
A 1
B 2
C 3
bdf
x1 x3
A T
B F
D T
Standard Joins
x1 x2 x3
A 1 T
B 2 F
C 3 NaN
pd.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.

Missing values

Missing values

- Handling missing values is an essential preprocessing task that can drastically deteriorate your model when not done with sufficient care.
- Do I have missing values? How are they expressed in the data? Should I withhold samples with missing values? Or should I replace them? If so, which values should they be replaced with?

NaN values

- NaN is a NumPy value. np.NaN
- NaT is a Pandas value. pd.NaT
- None is a vanilla Python value. None

```
1 df.isna()
```

	a	b
0	NaN	NaN
1	NaT	7.0
2	5	NaN

	a	b
0	True	True
1	True	False
2	False	True

NaN values

- read_csv(…, na_values, keep_default_na, na_filter)
  - By default the following values are interpreted as NaN: ‘’, ‘#N/A’, ‘#N/A N/A’, ‘#NA’, ‘-1.#IND’, ‘-1.#QNAN’, ‘-NaN’, ‘-nan’, ‘1.#IND’, ‘1.#QNAN’, ‘<NA>’, ‘N/A’, ‘NA’, ‘NULL’, ‘NaN’, ‘n/a’, ‘nan’, ‘null’.

NaN values

- df.isna()
- df.isna().any()
- df.isna().sum()

Remove missing values

- Drop rows with null values (MCAR)
  - df = df.dropna(axis=0, how={'any', 'all'}, thres)
- Drop columns
  - df = df.dropna(axis=1, how, thres)
- Do not use missing values

Fill values

- Fill values
  - df.fillna(0)
  - df.fillna(df.mean())
  - df.fillna(axis=0, method='bfill')

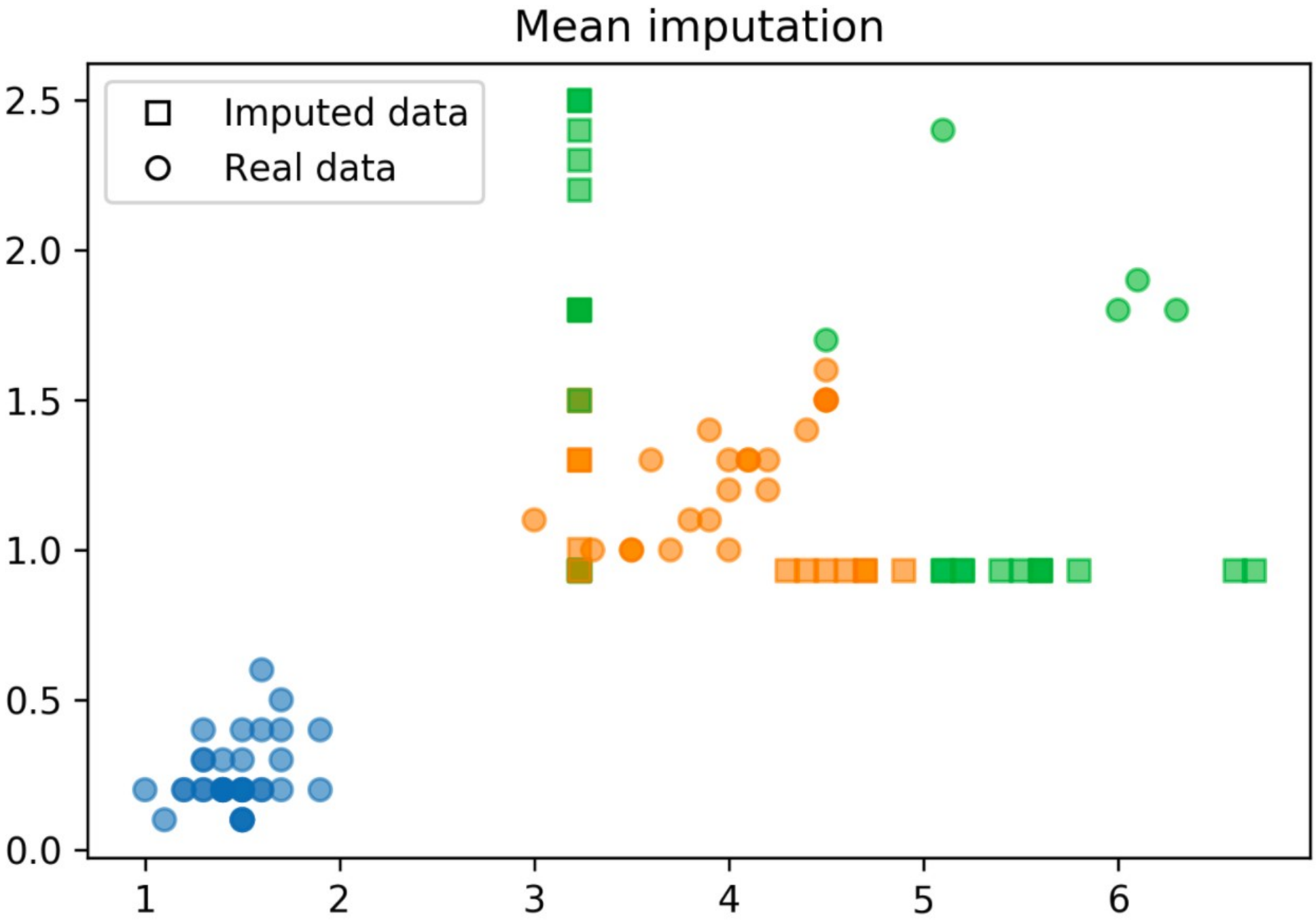
Handling Missing Data

- Drop rows with any column having NA/null data.

  df.dropna()

- Replace all NA/null data with value.

  df.fillna(value)


Mean imputation
Imputed data
Real data
2.5
2.0
1.5
1.0
0.5
0.0
1
2
3
4
5
6