

Selecting, Filtering and Sorting Data



Reindert-Jan Ekker

@rjekker <http://nl.linkedin.com/in/rjekker>



Overview



Indexing

- Basics: single rows and columns
- Indexing with slices and lists
- Using `loc` and `iloc`

Boolean filtering

Assigning values with indexing

Sorting



Demo



Indexing

- Basics: single rows and columns
- Indexing with slices and lists
- Using loc and iloc



Indexing: Single Value

Retrieve a column by label

This returns a **Series**

`df['col_name']` # if column index has type string

`df[5]` # if column index has type int

retrieve cell from column and row

`df['col_name']['row_name']`



Indexing: Lists and Slices

```
# Retrieve multiple columns in any order
```

```
# This returns a DataFrame
```

```
df[['Time', 'Temp', 'Pressure']]
```

```
# Slices return rows, not columns
```

```
# Also returns a DataFrame
```

```
df[2:35]
```

```
capitals['Palau':'Nauru'] # also works with labels
```



Indexing: loc

DataFrame.loc does row-based indexing with labels

Retrieve a single row

```
df.loc['San Marino']
```

Select column in the same operations

```
df.loc['San Marino', 'Population']
```

compare

```
df['Population']['San Marino']
```



Indexing: iloc

DataFrame.iloc does row-based indexing by position

Retrieve a single row

```
df.iloc[5]
```

Select column in the same operations

```
df.iloc[4, 2]
```

loc and iloc also support lists and slices



Demo



Boolean filtering

Assignment

Sorting



Boolean Filtering

Use Boolean comparisons in indexing operations

Retrieve all rows where column TEMP > 20

```
df[df['TEMP'] > 20]
```

Also works with loc and iloc

Select all columns with a mean over 6

```
grades.loc[:, grades.mean() > 6]
```



Assigning Values

Indexing operators allow you to assign to them

Update an entire column or row

```
grades['test_1'] += 1
```

```
grades.loc['Mary'] = [6, 8]
```

Also works with loc, iloc, etc.

```
grades.loc['John', ['test_1', 'test_2']] = 8
```



Assigning Values: Warning

Assigning values may give a warning

```
grades['test_2']['Ann'] = 8
```

SettingWithCopyWarning:

A value is trying to be set on a copy of a slice from a DataFrame

Did the assignment actually work?

Depends on the situation

Avoid chained indexing (use loc/iloc)



Sorting

```
# Sort by index
```

```
capitals.sort_index()
```

```
# Sort by a column
```

```
capitals.sort_values(by='Population')
```

```
# Sort by multiple columns
```

```
grades.sort_values(by=['test_1', 'test_2'])
```



Sorting: Arguments

```
# Reverse sort
```

```
capitals.sort_index(ascending=False)
```

```
# Sort rows, not columns
```

```
capitals.sort_index(axis=1)
```

```
# Sort original datastructure, don't return a copy
```

```
grades.sort_index(inplace=True)
```

```
# All arguments work both for sort_index and sort_values
```



Summary



Indexing

- Basics: single rows and columns
- Indexing with slices and lists
- Using `loc` and `iloc`

Boolean filtering

Assigning values with indexing

Sorting