Boolean Indexing

1405 Instructor: Ruiqing (Sam) Cao

Required Python Libraries for Today

Core

- NumPy, Pandas

```
import numpy as np
import pandas as pd
```

Visualization

- Matplotlib, Seaborn

```
from matplotlib import pyplot as plt
import seaborn as sns
```

Statistical learning

- scikit-learn, SciPy, statsmodels

```
import sklearn
import scipy
import statsmodel as sm
```

NumPy & Pandas Printing Format

It's better to print only *the first few decimal digits* of large real numbers. Set the print options to keep 3 decimal digits and supress scientific notation (for NumPy arrays and Pandas DataFrames):

- **NumPy**

```
np.set_printoptions(precision=3,suppress=True)
```

- **Pandas**

```
pd.options.display.float_format = '{:.3f}'.format
```

Filter & Modify Array by Condition

- **Boolean Indexing**: A powerful technique used to *filter* or *modify* existing arrays based on certain conditions

- Filter an array: *select only observations* that satisfy certain conditions

➢ *e.g., select subsample of companies with positive sales growth*

- Modify an array: *change the value of a variable* for the observations that satisfy certain conditions

➢ *e.g., replace all negative values in a data set with zeros*

Boolean Indexing

- Create a conditional Boolean array by

→ **Applying a conditional statement on the original array**

```
b = (conditional expression on a)
```

Example: Create a conditional array based on whether each element of the original array is an odd integer (not a multiple of 2)

```
a=np.array([1,2,3,4,5,6])
```
→ array([1, 2, 3, 4, 5, 6])

```
a%2==1
```
→ array([True, False, True, False, True, False])

Boolean Indexing

- Obtain an array containing the indexes that correspond to the True values in a conditional Boolean array b

```
np.where(b)
```

Example: Get the indexes of the original array the correspond to odd integers (not a multiple of 2)

```
a=np.array([1,2,3,4,5,6])
```
→ array([1, 2, 3, 4, 5, 6])

```
np.where(a%2==1)
```
→ array([0, 2, 4])

Filter an Array Using Boolean Indexes

- Filter the original array using a conditional Boolean array:
  - ➢ Original array a and conditional Boolean array b
  - ➢ Select a's elements where b is True:

```
a[b]
```

Example: Create a new array containing only odd integers (not multiples of 2) in the original array

```
b = (a%2==1)
```
→ array([True, False, True, False, True, False])

```
a1 = a[b]
```
→ array([1, 3, 5])

Modify an Array Using Boolean Indexes

`np.where()`

- `b` is a conditional Boolean array, `v1` are values for the condition `True`, and `v0` are values for the condition `False`

```
np.where(b,v1,v0)
```

...returns a new array where elements are drawn from `v1` where `b` is `True` and from `v0` where `b` is `False`

- `v1` and `v0` can be *arrays* or *scalars*
  - Arrays: Must have the same shape as `b`
  - Scalars: Automatically turns into an array with the same scalar value copied into an array of the same shape as `b` (aka Broadcasting)

Modify an Array Using Boolean Indexes

- Modify the original array using a conditional Boolean array: change the original values in the array a into 0 wherever the conditional array b is True, and keep other values the same

```
np.where(b,0,a)
```

Example: Create a new array that replaces odd integers in the original array with -99, and keeps other elements the same

```
b = (a%2==1)
```
→ array([True, False, True, False, True, False])

```
a2 = np.where(b,-99,a)
```
→ array([-99, 2, -99, 4, -99, 6])

Modify an Array Using Boolean Indexes

`np.select()` *→genarlized version of `np.where()`*

- `[b_0,b_1,…]` is a list of conditional Boolean arrays, `[v_0,v_1,…]` are values for each condition `b_0,v_1,…` being `True`

```
np.where([b_0,b_1,…],[v_0,v_1,…],default=0)
```

…returns a new array where elements are equal to `v_k` where `b_k` is `True`, and equal to `default` if none of the conditions are met

- `v_k` can be *an array* or *a scalar*
  - Arrays: Must have the same shape as `b_k`
  - Scalars: Automatically turns into an array with the same scalar value copied into an array of the same shape as `b_k` (aka Broadcasting)

Example: Censoring Data

- Left censoring: Replace all values *below* a threshold `tl` by that threshold (i.e., any value below `tl` becomes `tl`)

  `np.where(a<tl, tl, a)` or `np.where(a>=tl, a, tl)`

- Right censoring: Replace all values *above* a threshold `tr` by that threshold (i.e., any value below `tr` becomes `tr`)

  `np.where(a>tr, tr, a)` or `np.where(a<=tr, a, tr)`

Example: Remove Missing Values

- Generate Boolean masks for missing values with `np.isnan()`

```
b = np.isnan(a)
```

- Remove missing values by filtering on the original array

```
a[~b]
```

`~`: *bitwise NOT operator that flips* `True` → `False` *and* `False` → `True`

Example: Fill Missing Values

- Generate Boolean masks for missing values with `np.isnan()`

```
b = np.isnan(a)
```

- Fill missing values with zeros

```
np.where(b,0,a)
```

- Fill missing values with the sample average

```
avg_a= np.nanmean(a)
np.where(b,avg_a,a)
```

Missing Values & Data Cleaning

- Most real-world datasets contain missing values, but performing data analysis often requires a full dataset without missing values

- Here are two common approaches to handle missing data
  - *Remove* any data points where a variable has a missing value
  - *Fill* missing values using a pre-defined rule, such as replacing with zero, the mean, or the median of the variable across non-missing data points

Exercise: Handle Missing & Outlier Values

- Create and store some data for a single variable in the array `arr`:

```
arr = np.random.standard_normal(100)
mask = np.random.choice([True,False], 100, p=[0.05,0.95])
arr = np.where(mask,arr*1000,arr)
arr = np.where(np.abs(arr)>50,np.nan,arr)
```

1. Compute the mean, 5th percentile, and 95th percentile of the dataset
2. Remove values greater than 95th percentile
3. Remove values smaller than the 5th percentile
4. Replace missing values with the sample mean
5. Plot the frequency histogram of the cleaned dataset