第八章 NumPy Pandas 与 Cartopy

8.1 NumPy 科学计算基础

8.1.1 NumPy 简介

基本介绍 NumPy (Numerical Python) 是 Python 科学计算生态系统的核心库，提供一个**强大的 N 维数组对象**。包含许多广播功能、线性代数、傅里叶变换、随机数生成功能等。其底层用 C 语言编写，数组操作比 Python 快得多。

重要特性
① **性能**：向量化操作(Vectorization 一次性对整个数组或矩阵进行操作)比 Python 列表快几个数量级。
② **内存效率**：在内存中连续存储
③ **功能**：提供了大量针对数组优化的数学函数，线性代数运算
④ **生态**：是 Pandas、SciPy、Matplotlib、Scikit-learn、TensorFlow、PyTorch 等的基础

8.1.2 ndarray 对象

核心 ndarray 是 NumPy 的核心数据结构，是一个包含相同类型元素的多维网格。

8.1.2.1 创建数组

列表创建 从 Python 列表或元组创建，使用 `np.array()`。

```
#从列表创建一维数组
list1 = [1,2,3,4,5]
arr1d=np.array(list1)
```

```
#从列表创建二位数组
list2 = [[1,2,3],[4,5,6]]
arr2d=np.array(list2)
```

内置函数
① `np.zeros()` 创建指定形状的全零数组。
② `np.ones()` 创建指定形状的全一数组
③ `np.empty()` 创建一个未初始化的数组，数组中的元素是随机的，依赖于内存状态
④ `np.full((2,4),-999.)` 创建指定形状并用 fill_value 填充的数组

序列数组
① `np.arange(start, stop, step)` 类似于 Python range，但返回 NumPy 数组，支持浮点数步长
② `np.linspace(start, stop, num)` 在[start, stop]范围内生成 num 个等间隔的样本点，包含 stop
③ `np.logspace(start, stop, num, base=10.0)` 在对数尺度上生成等间隔样本点

随机数组 `np.random.seed(int)` 设置随机数种子，用于复现结果
① `np.random.rand(d0, d1, ..., dn)` 创建 [0, 1) 之间均匀分布的随机数数组。
② `np.random.randn(d0, d1, ..., dn)` 符合标准正态分布 $\mu = 0, \sigma = 1$ 的随机数数组。
③ `np.random.randint(low, high, size)` 创建指定范围 [low, high) 内的随机整数数组。

8.1.2.2 数组属性

`ndim` 数组的**轴(维数)的个数**：广播时可能有维数不对应的情况，用于判断
`shape` 数组的**维度**。是一个整数元组，表示每个维度中数组的大小。例如 (n_rows, n_cols)
`size` 数组**元素的总个数**，等于 shape 属性中元组元素的乘积。
`dtype` 描述数组中**元素类型**，例如 int64、float32 等
`itemsize` 数组中**每个元素的字节大小**
`data` 包含实际数组元素的缓冲区，通常不需要直接使用。

8.1.3 数据类型

常见类型 int8 (8 位整型)，int16，int32，int64，uint8 (无符号整型)，float16，float32，float64，complex64，complex128，bool_，object_，string_，unicode_ 等

类型指定 在创建数组时通过 dtype 参数指定。

类型转换 使用 astype() 方法可以显式转换数组的数据类型（注意：通常会创建新数组）。

8.1.4 数组操作

8.1.4.1 基本索引与切片

总述 与 Python 列表类似，但扩展到了多维。

多维数组 使用逗号分隔的索引元组 `arr[row, col]` 或 `arr[dim1, dim2, dim3, ...]`。

注意 视图 (View) 和副本 (Copy)的区别：**基本切片操作返回的是原始数组的视图，而不是副本**。这意味着**修改视图会直接反映在原始数组上**。

如果需要副本（不影响原数组），使用 `.copy()` 方法。（嵌套使用 `deepcopy`）

不是所有的数组都需要拷贝，例如中间过程的数组可以修改。

8.1.4.2 高级索引

整数数组 整数数组索引 (Integer Array Indexing)：使用整数数组(或列表)指定要选取的元素。结果数组的形状与索引数组的形状相同。

布尔数组 布尔数组索引 (Boolean Array Indexing)：使用与目标数组形状相同的布尔数组，True 位置对应的元素会被选中。常用于数据过滤。

注意 高级索引创建数组的副本，而不是视图。

```
arr = np.arange(12).reshape((3, 4))
# [[ 0  1  2  3], [ 4  5  6  7], [ 8  9 10 11]]
arr[[0, 2]]      # 选择第 0 行和第 2 行
# [[ 0  1  2  3], [ 8  9 10 11]]
arr[:, [0, 3]]   # 选择特定列
# [[ 0  3], [ 4  7], [ 8 11]]
# 选择特定位置的元素 (例如 (0,1), (1,2), (2,0))
rows = np.array([0, 1, 2])
cols = np.array([1, 2, 0])
arr[rows, cols]   # [1 6 8]
```

```
names = np.array(['Bob', 'Joe', 'Will', 'Bob', 'Will',
'Joe', 'Joe'])
data = np.random.randn(7, 4) # 7x4 的随机数据
# 条件：名字是否为 'Bob'
is_bob = (names == 'Bob')
# [ True False False  True False False False]
# 使用布尔掩码选择 data 中对应的行
data[is_bob]  # 选出第 0 行和第 3 行
# 使用布尔索引进行赋值
data[names != 'Joe'] = 0 # 将不是 Joe 的行设置为 0
data[data < 0] = -1 # 将所有负数设为 -1
```

8.1.4.3 数组形状操作

`reshape` 改变数组的形状，而不改变其数据。总元素数量必须保持不变。可以作为数组的方 `arr.reshape()` 或 NumPy 函数 `np.reshape(arr, ...)` 使用。

可以使用-1 推断维度大小 `arr = np.arange(12)` `reshaped_auto = arr.reshape((2, -1))`

`flatten` 将多维数组展平为一维数组，总是返回副本。

`transpose` 转置：交换数组的轴。对于二维数组，就是行和列互换。更高维数组，需要指定轴的顺序。

```
arr3d = np.arange(24).reshape((2, 3, 4)) # (2, 3, 4)
transposed_3d = arr3d.transpose((1, 0, 2)) # 将轴 0 和 1 互换 (3, 2, 4)
```

更改维度 添加/删除维度 (`newaxis`, `expand_dims`, `squeeze`)

`newaxis()`	在指定位置插入一个新轴（长度为 1）
`expand_dims(arr, axis)`	在指定 `axis` 插入新轴
`squeeze(arr, axis=None)`	移除数组中长度为 1 的轴。

8.1.4.4 数组连接与分割

`concatenate` `np.concatenate((arr1, arr2, ...), axis=0)` 沿指定轴连接一系列数组。数组必须具有相同的形状。

`split` `np.split(arr, indices_or_sections, axis=0)` 将数组沿指定轴分割成多个子数组。

8.1.5 NumPy 通用函数 (ufunc)

概念 通用函数（ufunc）是对 ndarray 中的数据**执行元素级操作的函数**。它们是 NumPy 高性能的关键，因为底层循环是在 C 语言中实现的（向量化）。

元素级操作 函数分别应用于数组的每个元素。

向量化　避免显式的 Python 循环，代码更简洁，执行更高效。
支持广播　可以自动处理不同形状数组间的运算（见下一节）

8.1.5.1 常见的 ufunc

一元函数　`np.sqrt()`: 计算平方根　`np.exp()`: 计算指数 e^x　`np.log()`, `np.log10()`: $\log_{10}(x)$
`np.abs()`: 计算绝对值　`np.square()`: 计算平方　`np.ceil()`, `np.floor()`: 向上/向下取整
二元函数　`np.maximum(x, y)`, `np.minimum(x, y)`: 元素级的最大/最小值。

8.1.5.2 聚合函数

概念　这些函数对数组进行聚合操作（通常会降低维度）
示例　`arr.sum()` / `np.sum(arr)`: 计算所有元素的和。
`arr.mean()` / `np.mean(arr)`: 计算平均值。
`arr.std()` / `np.std(arr)`: 计算标准差。
`arr.var()` / `np.var(arr)`: 计算方差。

8.1.6 缺失数据的处理

8.1.6.1 NaN 的特性与检测

NaN　在实际数据中，缺失值非常常见。NumPy 使用**特殊浮点数 `np.nan` (Not a Number)** 来表示缺失值。
特性　① `np.nan` 是**浮点类型**。整数数组通常无法直接存储 NaN（除非是 `object` 类型，但这会损失性能）。
若要在一个整数数组中引入缺失，通常需要先将其转换为浮点类型。
② NaN 具有传染性：大多数涉及 NaN 的算术运算结果都是 NaN。
③ 注意：`np.nan == np.nan # False` (NaN 不等于自身!)
检测　必须使用 `np.isnan()` 函数

8.1.6.2 NaN-aware (忽略 NaN 的) 通用函数

概述　标准的聚合函数（如 `np.sum`, `np.mean`）遇到 NaN 时通常会返回 NaN。
NumPy 提供了一套对应的 NaN 版本函数来忽略 NaN 值进行计算。
通用函数　`np.nansum()`　计算非 NaN 元素的和
`np.nanmean()`　计算非 NaN 元素的平均值
`np.nanmin()`, `np.nanmax()`　计算非 NaN 元素的最小/最大值
`np.nanstd()`, `np.nanvar()`　计算非 NaN 元素的标准差/方差
`np.nanargmin()`, `np.nanargmax()`　找到非 NaN 元素的最小/最大值索引
`np.nancumsum()`, `np.nancumprod()`　计算非 NaN 元素的累积和/积（NaN 保持不变）

8.1.6.3 处理 NaN 的策略

移除行列　需要谨慎，不使用自带的，使用 pandas 提供的 `dropna()`方法
填充值　用特定值（如 0、平均值、中位数等）替换 NaN

```
arr_to_fill = np.array([10.0, 20.0, np.nan, 40.0, np.nan])
mean_val = np.nanmean(arr_to_fill)    arr_filled_mean = arr_to_fill.copy()
arr_filled_mean[nan_mask] = mean_val
```

8.1.7 掩码数组

概述　除了使用 NaN，`numpy.ma` 模块提供了掩码数组 (Masked Arrays)，它将一个普通 NumPy 数组与一个布尔掩码 (mask) 结合，`True` 表示对应数据无效或被屏蔽。
特性　① 可以在整数、布尔等类型数组中标记无效值，无需转换类型。
② 掩码可以表示多种“无效”状态。
③ 计算时会自动忽略被屏蔽元素

8.1.7.1 创建掩码数组

初始设置

```
data = np.array([1, 2, -99, 4, 5]) # -99 可能是无效值
mask = [False, False, True, False, False] # True 表示 -99 无效
```

法一　使用 ma.array(data, mask=mask)

```
masked_arr = ma.array(data, mask=mask) # [1 2 -4 5]
```

法二　使用 ma.masked_where()　**条件区间，满足某一范围的值即 masked**

```
arr = np.arange(6)
masked_cond = ma.masked_where(arr < 3, arr) # [---3 4 5]
```

法三　使用 ma.masked_values()　**设置无效标记值，该值处 masked**

```
arr_int = np.array([10, 20, 0, 30, 0]) # 假设 0 是无效标记
masked_val = ma.masked_values(arr_int, 0) # [10 20 -30 --]
```

8.1.7.2 掩码数组的操作与计算

算术运算　算术运算：结果在对应位置也被屏蔽，即有 masked 的值位置不参与任何计算

聚合函数　ma 模块的聚合函数忽略屏蔽值

填充值　使用 filled 函数。filled_arr = arr.filled(fill_value=-999)

8.1.7.3 ma 与 NaN 的区别

一般方法　ma 功能更加灵活（无需数据转换），建议使用

判断方法　**整数/布尔数据:** numpy.ma 是首选，避免强制类型转换。

区分无效原因: 掩码提供更多灵活性。

计算控制: numpy.ma 函数自动忽略屏蔽值。NaN 需要额外函数。

Pandas 集成: NaN 与 Pandas 结合更紧密、更常用。

性能: 通常 NaN + nan* 性能优异，**但 np.ma 在避免类型转换时可能有优势**。

8.1.8 高级数据筛选与操作

8.1.8.1 向量化单次判断条件 np.where()

方法描述　np.where (condition, x, y): **根据 condition (布尔数组)，True 时从 x 取值，False 时从 y 取值**

示例

```
arr_where = np.random.randn(3, 3) * 3
result_where = np.where(arr_where > 0, 1, -1)      # 正数变 1，负数/零 变 -1
indices_tuple = np.where(arr_where < -1)           # 只提供条件，返回满足条件的索引元组
```

8.1.8.2 向量化多重判断条件 np.select()

方法描述　np.select(condlist, choicelist, default): **根据 condlist (条件列表) 和 choicelist (选择值列表)** 进行选择。**条件与选择值执行的操作一一对应。**

示例

```
arr_select = np.arange(12)
condlist = [ arr_select < 4, (arr_select >= 4) & (arr_select < 8), arr_select > =8]
choicelist = [ arr_select * 10, arr_select * 100, arr_select * 1000 ]
result_select = np.select(condlist, choicelist, default=-1) # 默认值 -1
# [   0   10   20   30  400  500  600  700 8000 9000 10000 11000]
```

8.1.8.3 根据条件提取元素 extract

方法描述　np.extract(condition, arr): **提取满足 condition 的元素，返回一维数组**

示例

```
arr_extract = np.arange(12).reshape((3, 4))
condition_extract = (arr_extract % 4 == 0) # 能被 4 整除
extracted = np.extract(condition_extract, arr_extract) # [0 4 8]
```

8.1.8.4 按索引选取和放置

索取元素　np.take(arr, indices, axis)　按索引选取元素，可指定轴。

放置元素　np.put(arr, indices, values)　在展平数组的指定索引处放置值。

示例

```
arr_takeput = np.arange(10, 16).reshape((2, 3)) # [[10 11 12], [13 14 15]]
# take: 选取第 1 行 [[13 14 15]]
taken_rows = np.take(arr_takeput, [1], axis=0)
# put: 修改展平后索引为 0, 2, 4 的元素
```