Python Intro

Python学习网站 http://c.biancheng.net/view/2210.html

List Function

- append(item) - adds an item to the end of the list
- extend(iterable) - appends all the items from an iterable (e.g. list, tuple) to the end of the list
- insert(index, item) - inserts an item at a specific index in the list
- remove(item) - removes the first occurrence of an item from the list
- pop(index) - removes and returns the item at a specific index in the list, or the last item if no index is specified
- clear() - removes all items from the list
- index(item) - returns the index of the first occurrence of an item in the list
- count(item) - returns the number of occurrences of an item in the list
- sort() - sorts the items in the list in ascending order
- reverse() - reverses the order of the items in the list
- copy() - returns a shallow copy of the list

Dictionary Function

- dictionary.clear(): Removes all items from the dictionary.
- dictionary.copy(): Returns a shallow copy of the dictionary.
- dictionary.fromkeys(keys, value): Returns a new dictionary with the given keys and value.
- dictionary.get(key, default): Returns the value for the given key, or the default value if the key is not found in the dictionary.
- dictionary.items(): Returns a view of the dictionary's items as a list of key-value pairs.
- dictionary.keys(): Returns a view of the dictionary's keys.
- dictionary.pop(key, default): Removes the specified key and returns its value. If the key is not found, it returns the default value.
- dictionary.popitem(): Removes and returns an (key, value) pair from the dictionary.
- dictionary.setdefault(key, default): Returns the value for the given key if it exists in the dictionary, otherwise it adds the key with the default value.
- dictionary.update(other_dictionary): Merges the contents of another dictionary into the current dictionary.
- dictionary.values(): Returns a view of the dictionary's values.

for-enumerate: It allows you to iterate over a sequence and also keep track of the index of the current item: **for index, item in enumerate(sequence)**
for-zip: It allows you to iterate over multiple sequences in parallel: **for item1, item2 in zip(sequence1, sequence2)**

***args** Accept any number of positional arguments. It is typically used when you don't know in advance how many arguments the function will be passed.
Keyword arguments: These are passed by specifying the parameter name and its value. This allows you to specify the arguments out of order or to specify only certain arguments while leaving the rest with their default values
If you do not know the number of arguments that will be passed into your function:
****kwargs** allows a function to accept any number of keyword arguments. It is typically used when you don't know in advance how many keyword arguments the function will be passed. When calling a function, you pass any additional keyword arguments as a dictionary.

Numpy

Creating NumPy Arrays

Using e.g. np.array() np.ones() np.zeros() np.random.rand()
np.linespace(0,10,100)

Reshape

```
a = np.array([1, 2, 3, 4, 5, 6])
print("Original Array: ", a)

b = np.reshape(a, (2, 3), order='C') # reshape into 2 rows and 3 columns, C-style order
print("Reshaped Array (C-style order): \n", b)

c = np.reshape(a, (2, 3), order='F') # reshape into 2 rows and 3 columns, Fortran-style order
print("Reshaped Array (Fortran-style order): \n", c)
```

```
A3 = np.random.rand(2, 3, 4) #2 arrays of 3x4 OR 2 arrays of 3 arrays of 4 elements each.
B3 = A3.flatten() #flattens the A3 array into a one-dimensional array.
```

Indexing with boolean array

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

mask = (a > 3) ## create a boolean array that represents the condition "element is greater than 3"

b = a[mask] # use the boolean array to extract elements from the original array

print("Extracted Array: \n", b)
```

```
Extracted Array:
 [4 5]
```

Linear Algebra functions

numpy.linalg is a sub-library within the Numpy library that provides linear algebra functions for operations such as matrix multiplication, transpose, determinant, eigenvalues, and singular value decomposition. It allows you to perform common linear algebra operations on arrays and matrices

- inv: matrix inverse 逆矩阵
- eig: eigenvalues and eigenvectors of a square matrix 特征值
- svd: singular value decomposition of a matrix 奇异值分解
- det: determinant of a square matrix 行列式

- norm: matrix or vector norm 范数

```
vec1 = np.array([1,2,3,4])
vec2 = np.array([6,7,8,8])
```

```
a = np.random.rand(5,5)
```

```
np.linalg.inv(a)
```

```
array([[ 0.19543304, -1.02064159,  1.47823768, -0.18431977, -0.27609924],
       [ 1.41953527, -4.7826376 , -0.77976915,  5.82085789, -0.10890117],
       [ 0.8805026 ,  0.77784514, -1.58607796, -2.20681665,  1.76607206],
       [ 0.76861812,  2.2760327 ,  0.30642752,  0.67326987, -1.92558238],
       [-3.35818191,  5.45654535,  0.80962399, -6.30019124,  1.52769211]])
```

```
np.linalg.norm(vec1, ord = 2) # l2-norm
```

```
5.477225575051661
```

```
np.linalg.norm(vec1, ord = 1) # l1-norm
```

```
10.0
```

L2-Norm (often just called "Euclidean norm"):

- Defined for a vector $\mathbf{v} = [v_1, v_2, ..., v_n]$ as:
  $||\mathbf{v}||_2 = \sqrt{v_1^2 + v_2^2 + ... + v_n^2}$
- It's essentially the Euclidean distance of the vector from the origin.
- In NumPy, it's computed using **`np.linalg.norm(vec1, ord=2)`**.

L1-Norm:

- Defined for a vector $\mathbf{v} = [v_1, v_2, ..., v_n]$ as:
  $||\mathbf{v}||_1 = |v_1| + |v_2| + ... + |v_n|$
- It's the sum of the absolute values of the vector components.
- In NumPy, it's computed using **`np.linalg.norm(vec1, ord=1)`**.

Pandas

1. It provides data structures such as **Series (1-dimensional)** and **DataFrame (2-dimensional)** for handling and processing data. It also has advanced options for handling missing data and dealing with time series data.
2. Pandas has features for reading and writing data in different formats such as CSV, Excel, SQL, and more. (read_csv, read_excel, read_sql functions)
3. Pandas has powerful data manipulation capabilities, including merging, joining, grouping, and pivoting data.

4. Pandas is efficient and optimized for performance, making it a popular choice among data scientists and analysts.

There are several ways to create a Pandas DataFrame, including:

1. From a dictionary: You can create a DataFrame from a dictionary where keys become column names and values become the data in the columns.
2. From a list of lists or tuples: You can create a DataFrame from a list of lists or tuples, where each inner list or tuple represents a row in the DataFrame.
3. From a Numpy array: You can create a DataFrame from a Numpy array, which is a multi-dimensional array for numerical operations.

```
import pandas as pd
#From a dictionary
D = {
    'feature1': [1, 2, 3, 4],
    'feature2': [10, 20, 30, 40],
    'feature3': ['A', 'B', 'C', 'D']
}

df = pd.DataFrame(D)
```

```
#From list
D2 =  [
    [1, 10, 'A'],
    [2, 20, 'B'],
    [3, 30, 'C'],
    [4, 40, 'D']
]

df = pd.DataFrame(D2, columns=['feature1', 'feature2', 'feature3'])
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4 entries, 0 to 3
Data columns (total 3 columns):
 #   Column    Non-Null Count  Dtype 
---  ------    --------------  ----- 
 0   feature1  4 non-null      int64 
 1   feature2  4 non-null      int64 
 2   feature3  4 non-null      object
dtypes: int64(2), object(1)
memory usage: 224.0+ bytes
```

```
#From a numpy array
data = np.array([[1, 10, 'A'], [2, 20, 'B'], [3, 30, 'C'], [4, 40, 'D']])

df = pd.DataFrame(data, columns=['feature1', 'feature2', 'feature3'])
```

```
df.info()
```

```
from sklearn.datasets import load_iris
iris = load_iris()
```

```
df = pd.DataFrame(iris.data,columns=iris.feature_names)
df['target'] = pd.Series(iris.target)
```

```
df.info()
```

```
df.head()
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

- 从 sklearn.datasets 导入 load_iris:
  - load_iris 是一个函数，当调用时，它返回一个包含 Iris 数据集的对象。Iris 数据集是一个经典的数据集，经常用于数据分析和机器学习的示例中。它包含了 150 个样本，分为 3 类，每类 50 个样本。每个样本有 4 个特征，分别是：萼片长度、萼片宽度、花瓣长度和花瓣宽度。
- **加载 Iris 数据集**:
  - 通过调用 load_iris() 函数来加载数据集，并将返回的对象存储在 iris 变量中。
- **创建一个 DataFrame**:
  - 使用 pandas 创建一个 DataFrame，其中数据 (iris.data) 是 Iris 数据集的主要部分 (每个样本的四个特征值)，列名 (iris.feature_names) 则是这四个特征的描述。
- **添加一个 'target' 列到 DataFrame**:
  - iris.target 包含了每个样本的类标签 (0、1 或 2，对应于 setosa、versicolor 和 virginica 三个品种的鸢尾花)。这一行代码将这些类标签作为新的 'target' 列添加到 DataFrame 中。

Indexing

Use the **iloc** property to access rows by integer-based position.
Use the **loc** property to access rows by label. If your DataFrame has a default integer-based index, you can use the loc property with index labels.

Use column name to access data for a particular folumn f = df['feature_name']

```
df.iloc[0]
```

```
sepal length (cm)    5.1
sepal width (cm)     3.5
petal length (cm)    1.4
petal width (cm)     0.2
target               0.0
Name: 0, dtype: float64
```

```
df.iloc[0:4]
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0

```
df.loc[:,['sepal length (cm)', 'sepal width (cm)']]
```

	sepal length (cm)	sepal width (cm)
0	5.1	3.5
1	4.9	3.0
2	4.7	3.2
3	4.6	3.1
4	5.0	3.6
...	...	...
145	6.7	3.0
146	6.3	2.5
147	6.5	3.0
148	6.2	3.4

Slicing

```
df.loc[1:20,'sepal length (cm)']
```

```
3      4.6
4      5.0
5      5.4
6      4.6
7      5.0
8      4.4
9      4.9
10     5.4
11     4.8
12     4.8
13     4.3
14     5.8
15     5.7
16     5.4
17     5.1
18     5.7
19     5.1
20     5.4
Name: sepal length (cm), dtype: float64
```

```
maks = df.columns != 'sepal length (cm)'
```

```
df.iloc[:,maks]
```

	sepal width (cm)	petal length (cm)	petal width (cm)	target
0	3.5	1.4	0.2	0
1	3.0	1.4	0.2	0
2	3.2	1.3	0.2	0
3	3.1	1.5	0.2	0
4	3.6	1.4	0.2	0
...	...	...	...	...
145	3.0	5.2	2.3	2
146	2.5	5.0	1.9	2
147	3.0	5.2	2.0	2
148	3.4	5.4	2.3	2
149	3.0	5.1	1.8	2

Scikit-learn

1. It contains a wide range of algorithms for supervised and unsupervised learning, including regression, classification, clustering, dimensionality reduction, and model selection
2. It features a simple and efficient API, making it easy to use for practitioners and researchers

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler, StandardScaler, RobustScaler
from sklearn.datasets import load_iris
```

Feature Scaling
Normalization
Normalisation, also known as min-max scaling, is a scaling technique whereby the values in a column are shifted so that they are bounded between a fixed range of 0 and 1.
X_new = (X - X_min) / (X_max - X_min)
Standardization
Standardisation or Z-score normalisation is another scaling technique whereby the values in a column are rescaled so that they demonstrate the properties of a standard Gaussian distribution, that is mean = 0 and variance = 1.
X_new = (X - mean) / std

Tensor Flow

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
import tensorflow as tf
mnist = tf.keras.datasets.mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 [==============================] - 0s 0us/step
```

```
x_train.shape
```

```
(60000, 28, 28)
```

```
model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10)
])
```

这里首先定义了一个变量 mnist，它是 TensorFlow 中的一个辅助模块，用于加载 MNIST 数据集。然后将数据集分为训练集 (x_train, y_train) 和测试集 (x_test, y_test)。
由于 MNIST 图像的像素值在 0 到 255 之间，这行代码将所有的像素值都除以 255，使它们位于 0 到 1 的范围内，这是一个常见的数据预处理步骤。
这行代码返回训练数据的形状。对于 MNIST，这应该是 (60000, 28, 28)，表示有 60,000 个 28x28 的图像。
这里定义了一个简单的顺序模型，其中包括以下层：

- Flatten：这将 28x28 的图像平坦化为一个 784 的向量。

- Dense：一个包含 128 个神经元的全连接层，使用 ReLU 作为激活函数。
- Dropout：一个随机丢弃 20% 神经元的层，用于防止过拟合。
- Dense：另一个全连接层，有 10 个神经元，对应于 10 个类 (数字 0-9)。

```
loss_fn = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)

model.compile(optimizer='adam',loss=loss_fn, metrics=['accuracy'])

model.fit(x_train, y_train, epochs=5)
```

```
Epoch 1/5
1875/1875 [==============================] - 11s 3ms/step - loss: 0.2960 - accuracy: 0.9138
Epoch 2/5
1875/1875 [==============================] - 5s 3ms/step - loss: 0.1429 - accuracy: 0.9573
Epoch 3/5
1875/1875 [==============================] - 5s 3ms/step - loss: 0.1047 - accuracy: 0.9685
Epoch 4/5
1875/1875 [==============================] - 5s 3ms/step - loss: 0.0862 - accuracy: 0.9731
Epoch 5/5
1875/1875 [==============================] - 5s 2ms/step - loss: 0.0758 - accuracy: 0.9761
<keras.src.callbacks.History at 0x7de43ca98ac0>
```

```
model.evaluate(x_test,  y_test, verbose=2)
```

```
313/313 - 1s - loss: 0.0723 - accuracy: 0.9765 - 1s/epoch - 5ms/step
[0.07228932529687881, 0.9764999747276306]
```

这定义了一个损失函数，它是用于多类分类问题的交叉熵损失。
配置了模型的学习过程，使用 Adam 优化器、之前定义的损失函数和准确度作为评估标准。
使用训练数据集 (x_train, y_train) 训练模型，总共进行 5 轮迭代。
使用测试集 (x_test, y_test) 评估模型的性能。verbose=2 表示在评估过程中显示进度。