Aggregation

Use aggregation techniques to combine NumPy data and arrays.

Chapter Goals:

- Learn how to aggregate data in NumPy
- Write code to obtain sums and concatenations of NumPy arrays

A. Summation

In the chapter on Math, we calculated the sum of individual values between multiple arrays. To sum the values within a single array, we use the `np.sum` function.

The function takes in a NumPy array as its required argument, and uses the `axis` keyword argument in the same way as described in previous chapters. If the `axis` keyword argument is not specified, `np.sum` returns the overall sum of the array.

The code below shows how to use `np.sum`.

```
arr = np.array([[0, 72, 3],
                [1, 3, -60],
                [-3, -2, 4]])
print(np.sum(arr))
print(repr(np.sum(arr, axis=0)))
print(repr(np.sum(arr, axis=1)))
```

In addition to regular sums, NumPy can perform cumulative sums using `np.cumsum`. Like `np.sum`, `np.cumsum` also takes in a NumPy array as a required argument and uses the `axis` argument. If the `axis` keyword argument is not specified, `np.cumsum` will return the cumulative sums for the flattened array.

The code below shows how to use `np.cumsum`. For a 2-D NumPy array, setting

`axis=0` returns an array with cumulative sums across each column, while

`axis=1` returns the array with cumulative sums across each row. Not setting `axis` returns a cumulative sum across all the values of the flattened array.

```
arr = np.array([[0, 72, 3],
                [1, 3, -60],
                [-3, -2, 4]])
print(repr(np.cumsum(arr)))
print(repr(np.cumsum(arr, axis=0)))
print(repr(np.cumsum(arr, axis=1)))
```

B. Concatenation

An important part of aggregation is combining multiple datasets. In NumPy, this equates to combining multiple arrays into one. The function we use to do this is `np.concatenate`.

Like the summation functions, `np.concatenate` uses the `axis` keyword argument. However, the default value for `axis` is `0` (i.e. dimension 0). Furthermore, the required argument for `np.concatenate` is a list of arrays, which the function combines into a single array.

The code below shows how to use `np.concatenate`, which aggregates arrays by joining them along a specific dimension. For 2-D arrays, not setting the `axis` argument (defaults to `axis=0`) concatenates the arrays vertically. When we set `axis=1`, the arrays are concatenated horizontally.

```
arr1 = np.array([[0, 72, 3],
                 [1, 3, -60],
                 [-3, -2, 4]])
arr2 = np.array([[-15, 6, 1],
                 [8, 9, -4],
                 [5, -21, 18]])
print(repr(np.concatenate([arr1, arr2])))
print(repr(np.concatenate([arr1, arr2], axis=1)))
print(repr(np.concatenate([arr2, arr1], axis=1)))
```

Time to Code!

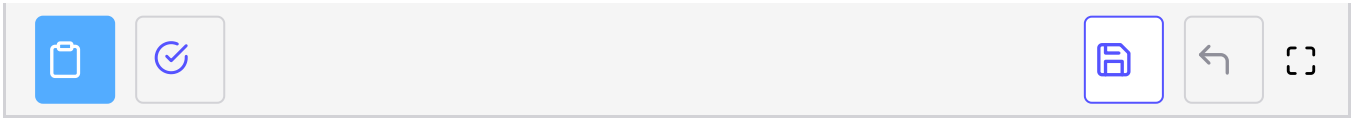
Each coding exercise in this chapter will be to complete a small function that takes in 2-D NumPy matrices as input. The first function to complete is `get_sums`, which returns the overall sum and column sums of `data`.

Set `total_sum` equal to `np.sum` applied to `data`.

Set `col_sum` equal to `np.sum` applied to `data`, with `axis` set to `0`.

Return a tuple of `total_sum` and `col_sum`, in that order.

```
def get_sums(data):
  # CODE HERE
  pass
```

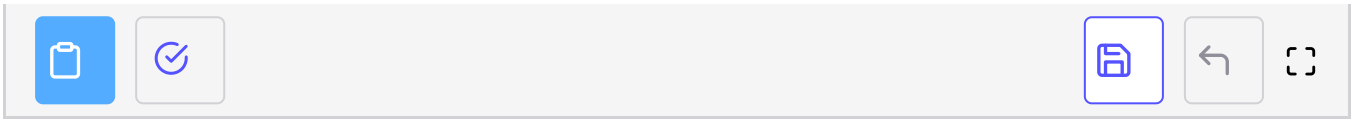


The next function to complete is `get_cumsum`, which returns the cumulative sums for each row of `data`.

Set `row_cumsum` equal to `np.cumsum` applied to `data` with `axis` set to `1`.

Then return `row_cumsum`.

```
def get_cumsum(data):
  # CODE HERE
  pass
```



The final function, `concat_arrays`, takes in two 2-D NumPy arrays as input. It returns the column-wise and row-wise concatenations of the input arrays.

Set `col_concat` equal to `np.concatenate` applied to a list of `data1`, `data2`, in that order.

Set `row_concat` equal to `np.concatenate` applied to a list of `data1`, `data2`, in that order. The `axis` keyword argument should be set to 1.

Return a tuple containing `col_concat` and `row_concat`, in that order.

```
def concat_arrays(data1, data2):
  # CODE HERE
  pass
```