NumPy and Matplotlib Quick Reference

NumPy

Core: Creating Arrays

- `np.array(object)`: Creates an array from a list, tuple, etc.

```
>>> np.array([1, 2, 3])
array([1, 2, 3])
```

- `np.zeros(shape)`: Creates an array of zeros. For 2D arrays, `shape` is (rows, columns).

```
>>> np.zeros((2, 3))  # 2x3 array (2 rows, 3 columns)
array([[0., 0., 0.],
       [0., 0., 0.]])
```

- `np.ones(shape)`: Creates an array of ones.

```
>>> np.ones((3, 2))
array([[1., 1.],
       [1., 1.],
       [1., 1.]])
```

- `np.empty(shape)`: Creates an array without initializing entries. May contain garbage values.

```
>>> np.empty((2, 2))
array([[6.23042e-307, 6.23042e-307],
       [1.60212e-311, 0.00000e+000]])
```

- `np.arange(start, stop, step)`: Creates a 1D array with a range of values. Like Python's `range`.

```
>>> np.arange(0, 10, 2)
array([0, 2, 4, 6, 8])
```

- `np.linspace(start, stop, num)`: Creates a 1D array with evenly spaced values. `num` is the number of points.

```
>>> np.linspace(0, 1, 5)
array([0.  , 0.25, 0.5 , 0.75, 1.  ])
```

- `np.eye(n)`: Creates an identity matrix.

```
>>> np.eye(3)
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
```

- `np.full(shape, value)`: Creates an array filled with a scalar value.

```
>>> np.full((2,2), 7)
array([[7, 7],
       [7, 7]])
```

Generating Random Numbers

- `np.random.rand(d0, d1, ..., dn)`: Generates an array of the given shape with random floats in the interval [0, 1).

```
>>> np.random.rand(2, 3)
array([[0.123, 0.456, 0.789],
       [0.234, 0.567, 0.890]])
```

- `np.random.randn(d0, d1, ..., dn)`: Generates an array of the given shape with random floats from a standard normal distribution (mean 0, std dev 1).

```
>>> np.random.randn(2, 2)
array([[-0.2, 0.5],
       [ 1.1, -0.8]])
```

- `np.random.randint(low, high=None, size=None, dtype=int)`: Returns random integers from `low` (inclusive) to `high` (exclusive). If `high` is None, integers are from [0, `low`).

```
>>> np.random.randint(2, 10, size=(2, 3))
array([[8, 5, 7],
       [2, 9, 2]])
```

- `np.random.random_sample(size=None)`: Returns random floats in the interval [0.0, 1.0).

```
 >>> np.random.random_sample((2, 2))
    array([[0.5488135 , 0.71518936],
           [0.60276338, 0.54488318]])
```

Inspecting Arrays

- `arr.ndim`: Number of array dimensions (1 for 1D, 2 for 2D, etc.).
- `arr.shape`: Tuple of array dimensions (rows, columns, etc.).
- `arr.size`: Total number of elements in the array.
- `arr.dtype`: Data type of the array elements (e.g., `int64`, `float64`).

Manipulating Arrays

- `arr.reshape(shape)`: Changes the shape of the array without changing the data.

```
>>> arr = np.arange(6)
>>> arr.reshape((2, 3))
array([[0, 1, 2],
       [3, 4, 5]])
```

- `arr.flatten()` or `arr.ravel()`: Flattens the array into a 1D array.

```
>>> arr = np.array([[1, 2], [3, 4]])
>>> arr.flatten()
array([1, 2, 3, 4])
```

- `arr.T`: Transposes the array.

```
>>> arr = np.array([[1, 2], [3, 4]])
>>> arr.T
array([[1, 3],
       [2, 4]])
```

- `np.concatenate((arr1, arr2, ...), axis)`: Joins arrays along an existing axis.

```
>>> a = np.array([[1, 2], [3, 4]])
>>> b = np.array([[5, 6]])
>>> np.concatenate((a, b), axis=0)
array([[1, 2],
       [3, 4],
       [5, 6]])

>>> c = np.array([[7], [8]])
>>> np.concatenate((a, c), axis=1)
array([[1, 2, 7],
       [3, 4, 8]])
```

- `np.stack((arr1, arr2, ...), axis)`: Joins arrays along a new axis.

```
>>> a = np.array([[1, 2], [3, 4]])
>>> b = np.array([[5, 6], [7, 8]])
>>> np.stack((a, b), axis=0)
array([[[1, 2],
        [3, 4]],
       [[5, 6],
        [7, 8]]])

>>> np.stack((a, b), axis=1)
array([[[1, 2],
        [5, 6]],
       [[3, 4],
        [7, 8]]])
```

```
>>> np.stack((a, b), axis=2)
array([[[1, 5],
        [2, 6]],
       [[3, 7],
        [4, 8]]])
```

- `np.split(arr, indices_or_sections, axis)`: Splits an array into multiple sub-arrays.

```
>>> arr = np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]])

>>> np.split(arr, 2, axis=0)
[array([[1, 2, 3, 4],
       [5, 6, 7, 8]]),
 array([[ 9, 10, 11, 12]])]

>>> np.split(arr, [1, 3], axis=1)
[array([[ 1,  2],
       [ 5,  6],
       [ 9, 10]]),
 array([[ 3],
       [ 7],
       [11]]),
 array([[ 4],
       [ 8],
       [12]])]
```

Loading Data from Files

- From a TXT file:

```
>>> data = []
>>> with open('my_data.txt', 'r') as f:
...     for line in f:
...         row = line.strip().split(',')
...         row_values = [float(x) for x in row]
...         data.append(row_values)
>>> arr = np.array(data)
```

- From a CSV file:

```
>>> import csv
>>> data = []
>>> with open('my_data.csv', 'r') as f:
...         reader = csv.reader(f)
...         for row in reader:
...             row_values = [float(x) for x in row]
...             data.append(row_values)
>>> arr = np.array(data)
```

Array Operations

- `arr + scalar`, `arr * scalar`, etc.: Element-wise arithmetic operations.
- `arr + arr2`, `arr * arr2`, etc.: Element-wise arithmetic between arrays.
- `np.sin(arr)`, `np.cos(arr)`, `np.exp(arr)`, etc.: Element-wise mathematical functions.
- `np.sum(arr, axis=None)`: Sum of array elements.
- `np.mean(arr, axis=None)`: Mean of array elements.
- `np.max(arr, axis=None)`, `np.min(arr, axis=None)`: Maximum and minimum values.
- `np.argmax(arr, axis=None)`, `np.argmin(arr, axis=None)`: Indices of the maximum and minimum values.
- `np.dot(arr1, arr2)`: Dot product of two arrays.

```
>>> a = np.array([[1, 2], [3, 4]])
>>> b = np.array([[5, 6], [7, 8]])
>>> np.dot(a, b)
array([[19, 22],
       [43, 50]])
```

Descriptive Statistics

- `np.mean(arr, axis=None)`: Computes the arithmetic mean (average).

```
>>> arr = np.array([1, 2, 3, 4, 5])
>>> np.mean(arr)
3.0
```

- `np.median(arr, axis=None)`: Computes the median (middle value).

```
>>> arr = np.array([1, 2, 3, 4, 5])
>>> np.median(arr)
3.0
>>> arr = np.array([1, 2, 3, 4, 5, 6])
>>> np.median(arr)
3.5
```

- `np.std(arr, axis=None)`: Computes the standard deviation (spread of data).

```
>>> arr = np.array([1, 2, 3, 4, 5])
>>> np.std(arr)
1.4142135623730951
```

- `np.var(arr, axis=None)`: Computes the variance (square of the standard deviation).

```
>>> arr = np.array([1, 2, 3, 4, 5])
>>> np.var(arr)
2.0
```

- `np.percentile(arr, q, axis=None)`: Computes the q-th percentile.

```
>>> arr = np.array([1, 2, 3, 4, 5])
>>> np.percentile(arr, 25)
2.0
>>> np.percentile(arr, 75)
4.0
```

Indexing and Slicing

- `arr[index]`: Access a single element in a 1D array.
- `arr[start:stop:step]`: Slice the array.
- `arr[row_index, col_index]`: Access an element in a 2D array.
- `arr[row_start:row_stop, col_start:col_stop]`: Slice a 2D array.
- `arr[condition]`: Boolean array indexing (filter elements).

```
>>> arr = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
>>> arr[0, 1]
2
>>> arr[1:3, 0:2]
array([[4, 5],
       [7, 8]])
>>> arr[arr > 5]
array([6, 7, 8, 9])
```

Matplotlib

Core: Plotting

- `import matplotlib.pyplot as plt`: Import the main plotting module.
- `plt.plot(x, y, format_string, **kwargs)`: Creates a line plot.

```
>>> import matplotlib.pyplot as plt
>>> x = [1, 2, 3, 4]
>>> y = [10, 20, 25, 30]
>>> plt.plot(x, y, 'b-o', label='My Data')
>>> plt.show()
```

- `plt.scatter(x, y, **kwargs)`: Creates a scatter plot.

```
>>> plt.scatter(x, y, color='green', marker='*')
>>> plt.show()
```

- `plt.bar(x, height, width, **kwargs)`: Creates a bar chart.

```
>>> x = ['A', 'B', 'C', 'D']
>>> height = [20, 35, 30, 40]
>>> plt.bar(x, height, width=0.8)
>>> plt.show()
```

- `plt.hist(x, bins, **kwargs)`: Creates a histogram.

```
>>> import numpy as np
>>> data = np.random.randn(100)
>>> plt.hist(data, bins=20)
>>> plt.show()
```

- `plt.pie(x, labels, autopct, **kwargs)`: Creates a pie chart.

```
>>> x = [10, 20, 30, 40]
>>> labels = ['A', 'B', 'C', 'D']
>>> plt.pie(x, labels=labels, autopct='%.1f%%')
>>> plt.show()
```

- `plt.imshow(X, cmap, **kwargs)`: Displays an image or 2D array as an image.

```
>>> import numpy as np
>>> image_data = np.random.rand(10, 10)
>>> plt.imshow(image_data, cmap='viridis')
>>> plt.colorbar()
>>> plt.show()
```

Customizing Plots

- `plt.title(label)`: Sets the plot title.
- `plt.xlabel(label)`: Sets the x-axis label.
- `plt.ylabel(label)`: Sets the y-axis label.
- `plt.legend()`: Displays the legend.
- `plt.xlim(xmin, xmax)`, `plt.ylim(ymin, ymax)`: Sets the axis limits.
- `plt.xticks(ticks, labels)`, `plt.yticks(ticks, labels)`: Sets the axis ticks and labels.
- `plt.grid(True)`: Turns the grid on.
- `ax = plt.gca()`: Gets the current axes.
- `fig, ax = plt.subplots()`: Creates a figure and an axes.
- `plt.figure(figsize=(width, height))`: Creates a new figure with a specified size.
- `plt.savefig('filename.png')`: Saves the plot to a file.
- `plt.show()`: Displays the plot.

Subplots

- `plt.subplot(nrows, ncols, index)`: Adds a subplot to the current figure.

```
>>> plt.subplot(2, 1, 1)
>>> plt.plot([1, 2, 3], [1, 2, 1])
>>> plt.title('Top Plot')
>>> plt.subplot(2, 1, 2)
>>> plt.plot([1, 2, 3], [3, 2, 3])
>>> plt.title('Bottom Plot')
>>> plt.show()
```

- `fig, axes = plt.subplots(nrows, ncols)`: Creates a figure and an array of axes.

```
>>> fig, axes = plt.subplots(2, 2)
>>> axes[0, 0].plot([1, 2], [3, 4])
>>> axes[0, 1].plot([1, 2], [1, 2])
>>> axes[1, 0].plot([3, 4], [1, 2])
>>> axes[1, 1].plot([3, 4], [3, 4])
>>> plt.show()
```