HW0

September 1, 2021

1 Homework 0

This serves as your **zeroth** "homework" for the class. Give it a try yourself and let me know if you have any questions!

[1]:
```
# load numerical packages
import numpy as np
import scipy as sp
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

2 Uniform Density Shaping

Lets walk through a simple scientific computing problem together. We're going to build a simply density shaping function.

Numpy gives us a wonderful `random` library we can use. Lets focus on just one piece. We're going to pretend we only have access to one function: `rand`. Lets see how it works:

[2]:
```
np.random.rand?
```

```
Docstring:
rand(d0, d1, …, dn)

Random values in a given shape.

.. note::
    This is a convenience function for users porting code from Matlab,
    and wraps `random_sample`. That function takes a
    tuple to specify the size of the output, which is consistent with
    other NumPy functions like `numpy.zeros` and `numpy.ones`.

Create an array of the given shape and populate it with
random samples from a uniform distribution
over ``[0, 1)``.

Parameters
```

```
----------
d0, d1, …, dn : int, optional
    The dimensions of the returned array, must be non-negative.
    If no argument is given a single Python float is returned.

Returns
-------
out : ndarray, shape ``(d0, d1, …, dn)``
    Random values.

See Also
--------
random

Examples
--------
>>> np.random.rand(3,2)
array([[ 0.14022471,  0.96360618],  #random
       [ 0.37601032,  0.25528411],  #random
       [ 0.49313049,  0.94909878]]) #random
Type:      builtin_function_or_method
```

This function drawns numbers **uniformly** between $[0, 1]$. Lets draw one:

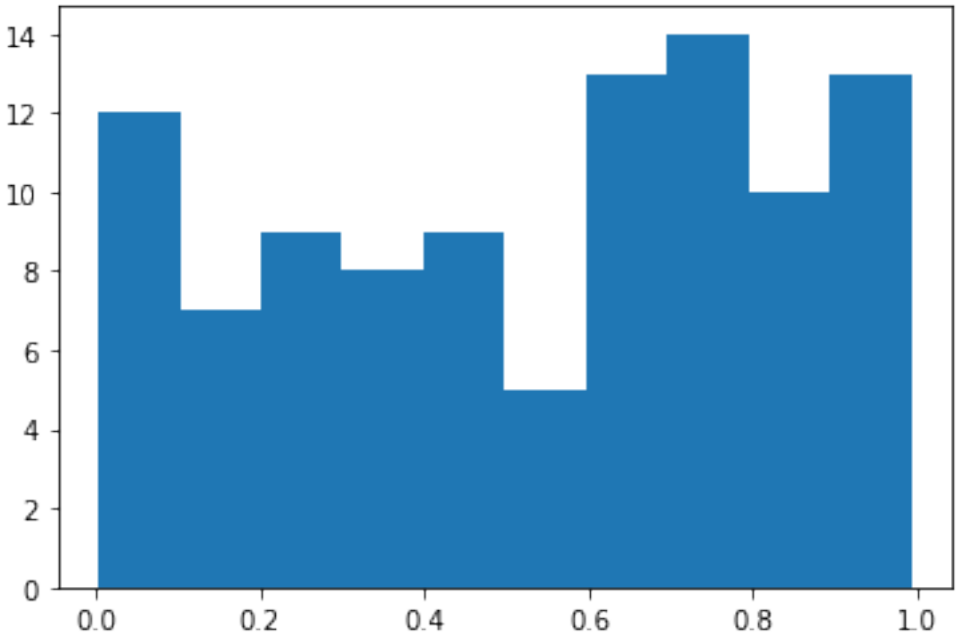
[3]:
```
np.random.rand()
```

[3]:
```
0.1769431807633962
```

Lets draw 100 samples, and use `matplotlib`'s `hist` function to visualize them:

[9]:
```
x = np.random.rand(100)
plt.hist(x)
```

[9]:
```
(array([12.,  7.,  9.,  8.,  9.,  5., 13., 14., 10., 13.]),
 array([0.00312198, 0.10198985, 0.20085772, 0.29972559, 0.39859346,
        0.49746133, 0.5963292 , 0.69519707, 0.79406494, 0.89293281,
        0.99180067]),
 <BarContainer object of 10 artists>)
```



`Seaborn` gives us many powerful built in utility function. A very useful one is its `distplot` function:

[8]:
```
sns.distplot?
```

```
Signature:
sns.distplot(
    a=None,
    bins=None,
    hist=True,
    kde=True,
    rug=False,
    fit=None,
    hist_kws=None,
    kde_kws=None,
    rug_kws=None,
    fit_kws=None,
    color=None,
    vertical=False,
    norm_hist=False,
    axlabel=None,
    label=None,
    ax=None,
    x=None,
)
Docstring:
DEPRECATED: Flexibly plot a univariate distribution of observations.
```

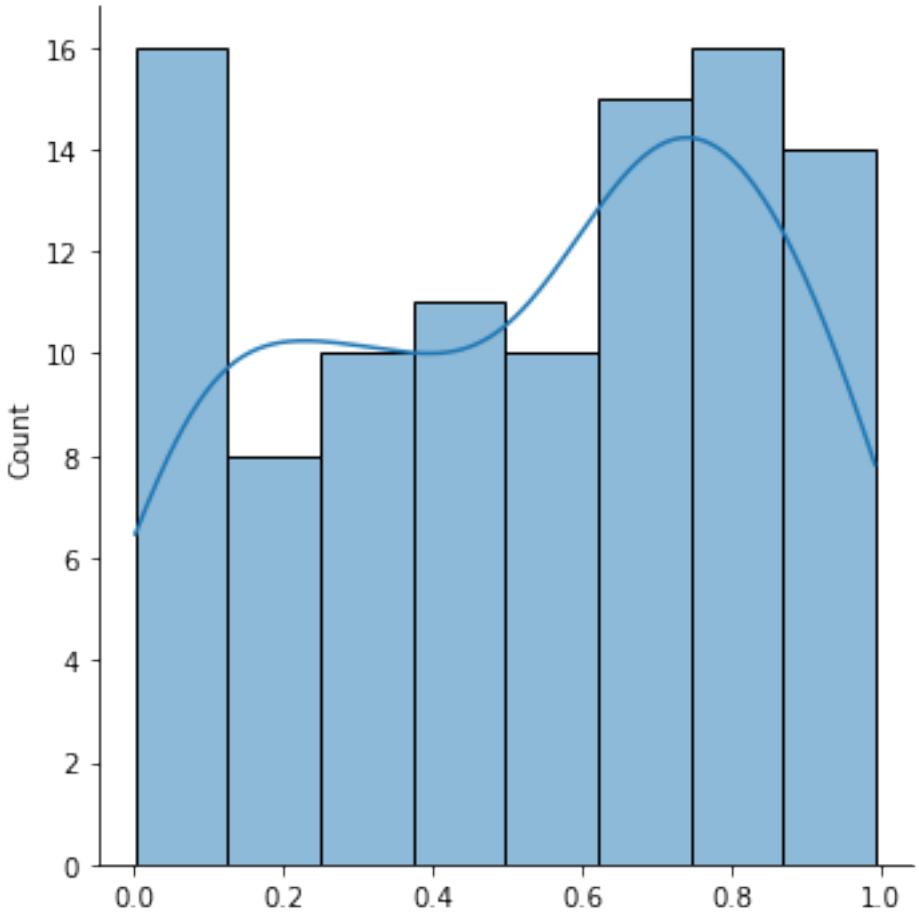
```
.. warning::
   This function is deprecated and will be removed in a future version.
   Please adapt your code to use one of two new functions:

   - :func:`displot`, a figure-level function with a similar flexibility
     over the kind of plot to draw
   - :func:`histplot`, an axes-level function for plotting histograms,
     including with kernel density smoothing

This function combines the matplotlib ``hist`` function (with automatic
calculation of a good default bin size) with the seaborn :func:`kdeplot`
and :func:`rugplot` functions. It can also fit ``scipy.stats``
distributions and plot the estimated PDF over the data.

Parameters
----------
a : Series, 1d-array, or list.
    Observed data. If this is a Series object with a ``name`` attribute,
    the name will be used to label the data axis.
bins : argument for matplotlib hist(), or None, optional
    Specification of hist bins. If unspecified, as reference rule is used
    that tries to find a useful default.
hist : bool, optional
    Whether to plot a (normed) histogram.
kde : bool, optional
    Whether to plot a gaussian kernel density estimate.
rug : bool, optional
    Whether to draw a rugplot on the support axis.
fit : random variable object, optional
    An object with `fit` method, returning a tuple that can be passed to a
    `pdf` method a positional arguments following a grid of values to
    evaluate the pdf on.
hist_kws : dict, optional
    Keyword arguments for :meth:`matplotlib.axes.Axes.hist`.
kde_kws : dict, optional
    Keyword arguments for :func:`kdeplot`.
rug_kws : dict, optional
    Keyword arguments for :func:`rugplot`.
color : matplotlib color, optional
    Color to plot everything but the fitted curve in.
vertical : bool, optional
    If True, observed values are on y-axis.
norm_hist : bool, optional
    If True, the histogram height shows a density rather than a count.
    This is implied if a KDE or fitted density is plotted.
axlabel : string, False, or None, optional
    Name for the support axis label. If None, will try to get it
```

```
    from a.name if False, do not set a label.
label : string, optional
    Legend label for the relevant component of the plot.
ax : matplotlib axis, optional
    If provided, plot on this axis.

Returns
-------
ax : matplotlib Axes
    Returns the Axes object with the plot for further tweaking.

See Also
--------
kdeplot : Show a univariate or bivariate distribution with a kernel
          density estimate.
rugplot : Draw small vertical lines to show each observation in a
          distribution.

Examples
--------

Show a default plot with a kernel density estimate and histogram with bin
size determined automatically with a reference rule:

.. plot::
    :context: close-figs

    >>> import seaborn as sns, numpy as np
    >>> sns.set_theme(); np.random.seed(0)
    >>> x = np.random.randn(100)
    >>> ax = sns.distplot(x)

Use Pandas objects to get an informative axis label:

.. plot::
    :context: close-figs

    >>> import pandas as pd
    >>> x = pd.Series(x, name="x variable")
    >>> ax = sns.distplot(x)

Plot the distribution with a kernel density estimate and rug plot:

.. plot::
    :context: close-figs

    >>> ax = sns.distplot(x, rug=True, hist=False)
```

```
Plot the distribution with a histogram and maximum likelihood gaussian
distribution fit:

.. plot::
    :context: close-figs

    >>> from scipy.stats import norm
    >>> ax = sns.distplot(x, fit=norm, kde=False)

Plot the distribution on the vertical axis:

.. plot::
    :context: close-figs

    >>> ax = sns.distplot(x, vertical=True)

Change the color of all the plot elements:

.. plot::
    :context: close-figs

    >>> sns.set_color_codes()
    >>> ax = sns.distplot(x, color="y")

Pass specific parameters to the underlying plot functions:

.. plot::
    :context: close-figs

    >>> ax = sns.distplot(x, rug=True, rug_kws={"color": "g"},
    …                   kde_kws={"color": "k", "lw": 3, "label": "KDE"},
    …                   hist_kws={"histtype": "step", "linewidth": 3,
    …                             "alpha": 1, "color": "g"})
File:      ~/opt/anaconda3/lib/python3.8/site-packages/seaborn/distributions.py
Type:      function
```

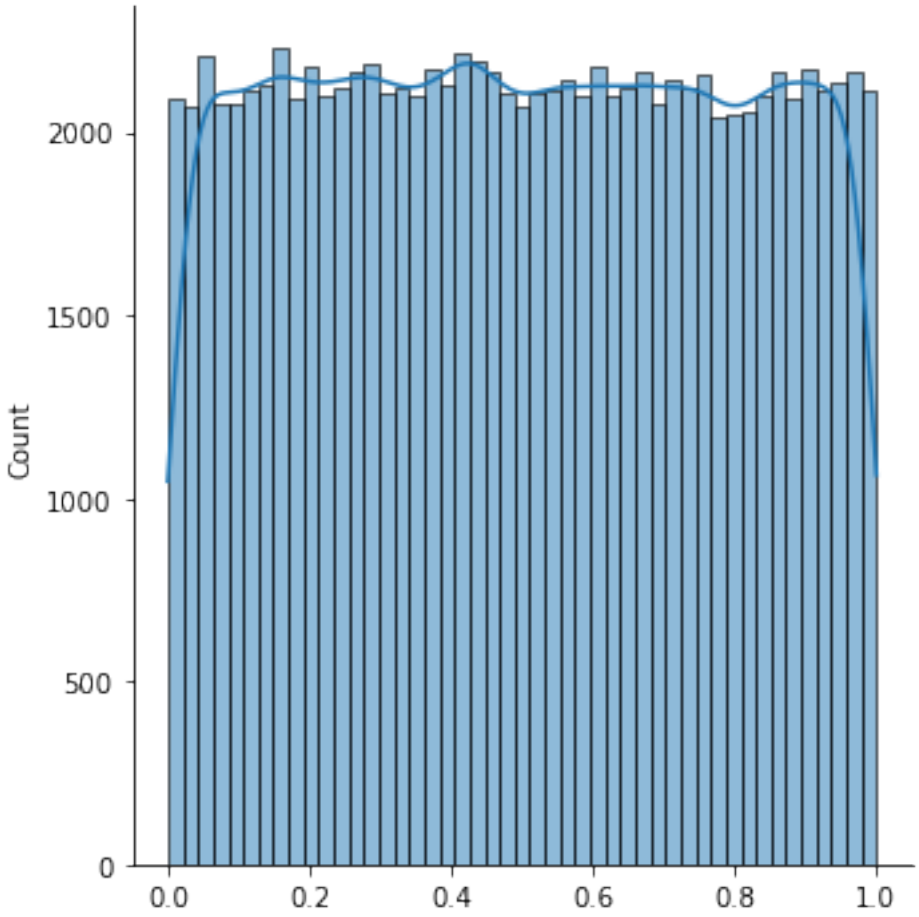
Lets use it to plot our histogram, and an estimated *density*:

[15]:
```
ax = sns.displot(x, kde=True)
```



What do we expect them to look like? Draw more samples and verify:

[16]:
```
ax = sns.displot(np.random.rand(100000), kde=True)
```



Ok, so we're confident that `rand` is indeed working correctly.

Now for the fun parts!

We're going to use `rand` to draw samples uniformly between some arbitrary $[a, b]$, and verify that it is indeed working.

Create a function `myrand` which allows someone to do the above.

[17]:
```
def myrand(n, a, b):
    x = np.random.rand(n)
    y = a + x * (b-a)
    return y
```

Use your `myrand` function to draw 1000 samples between $[3, 10]$:

[21]:
```
y = myrand(1000, 3, 10)
```

3 Coin Flipping

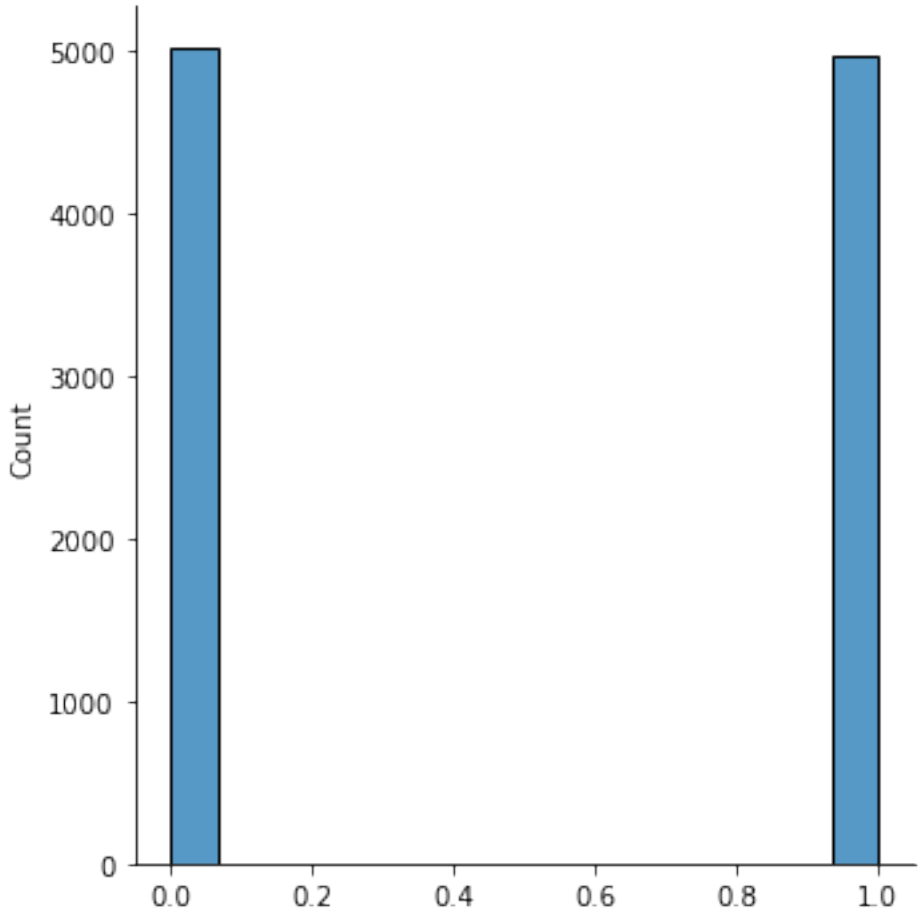
Now were going to do something more *subtle*.

Lets use `rand` to flip a **fair** coin:

[26]:
```
y = myrand(10000, 0, 1)
fair = [1 if i >= 0.5 else 0 for i in y]
```

Now plot it to verify:

[27]:
```
ax = sns.displot(fair)
```

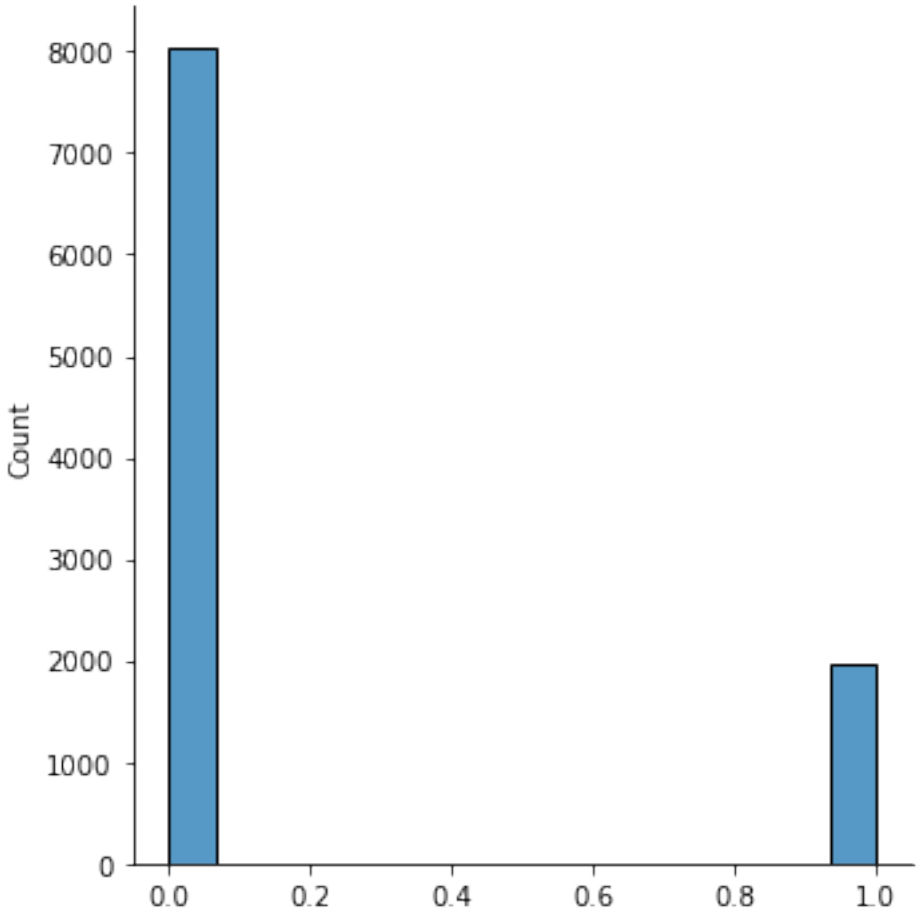


Now lets make a **biased** coin!

[28]:
```
y = myrand(10000, 0, 1)
biased = [1 if i >= 0.8 else 0 for i in y]
```

Plot it to verify!

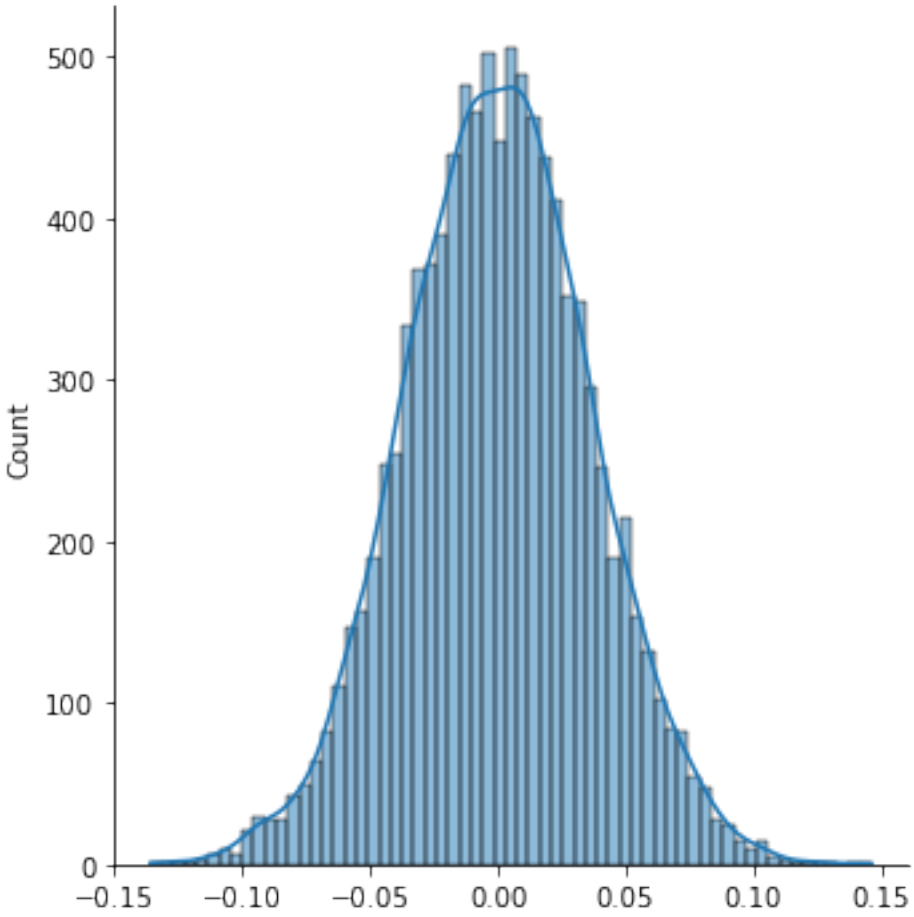
[29]:
```
ax = sns.displot(biased)
```



4 Bonus: On the Road To Gaussians!

Use the above, and reasoning from **first principles**, to see how we can turn a uniformly distributed random variable, into an arbitrary Guassian distribution!

[38]:
```
m = []
for i in range(0,10000):
    m.append(np.mean(myrand(1000, -2, 2)))
ax = sns.displot(m, kde=True)
```


Count
500
400
300
200
100
0
−0.15
−0.10
−0.05
0.00
0.05
0.10
0.15