Matplotlib Scatter

Creating Scatter Plots

With Pyplot, you can use the `scatter()` function to draw a scatter plot.

The `scatter()` function plots one dot for each observation. It needs two arrays of the same length, one for the values of the x-axis, and one for values on the y-axis:

```
import matplotlib.pyplot as plt
import numpy as np

x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
y = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])

plt.scatter(x, y)
plt.show()
```

The observation in the example above is the result of 13 cars passing by.

The X-axis shows how old the car is.

The Y-axis shows the speed of the car when it passes.

Are there any relationships between the observations?

It seems that the newer the car, the faster it drives, but that could be a coincidence, after all we only registered 13 cars.

Compare Plots

In the example above, there seems to be a relationship between speed and age, but what if we plot the observations from another day as well? Will the scatter plot tell us something else?

Draw two plots on the same figure:

```
import matplotlib.pyplot as plt
import numpy as np

#day one, the age and speed of 13 cars:
x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
y = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])
plt.scatter(x, y)

#day two, the age and speed of 15 cars:
x = np.array([2,2,8,1,15,8,12,9,7,3,11,4,7,14,12])
y = np.array([100,105,84,105,90,99,90,95,94,100,79,112,91,80,85])
plt.scatter(x, y)

plt.show()
```

Colors

You can set your own color for each scatter plot with the `color` or the `c` argument:

Set your own color of the markers:

```
import matplotlib.pyplot as plt
import numpy as np
x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
y = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])
plt.scatter(x, y, color = 'hotpink')

x = np.array([2,2,8,1,15,8,12,9,7,3,11,4,7,14,12])
y = np.array([100,105,84,105,90,99,90,95,94,100,79,112,91,80,85])
plt.scatter(x, y, color = '#88c999')
plt.show()
```

Set your own color of the markers:

```
import matplotlib.pyplot as plt
import numpy as np

x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
y = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])
colors = 
np.array(["red","green","blue","yellow","pink","black","orange","purple
","beige","brown","gray","cyan","magenta"])

plt.scatter(x, y, c=colors)

plt.show()
```

ColorMap

The Matplotlib module has a number of available colormaps.

A colormap is like a list of colors, where each color has a value that ranges from 0 to 100.

Here is an example of a colormap:

This colormap is called 'viridis' and as you can see it ranges from 0, which is a purple color, up to 100, which is a yellow color.

How to Use the ColorMap

You can specify the colormap with the keyword argument `cmap` with the value of the colormap, in this case `'viridis'` which is one of the built-in colormaps available in Matplotlib.

In addition, you have to create an array with values (from 0 to 100), one value for each point in the scatter plot:

Create a color array, and specify a colormap in the scatter plot:

```
import matplotlib.pyplot as plt
import numpy as np

x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
y = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])
colors = np.array([0, 10, 20, 30, 40, 45, 50, 55, 60, 70, 80, 90, 100])

plt.scatter(x, y, c=colors, cmap='viridis')

plt.show()
```

Include the actual colormap:

```
import matplotlib.pyplot as plt
import numpy as np

x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
y = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])
colors = np.array([0, 10, 20, 30, 40, 45, 50, 55, 60, 70, 80, 90, 100])

plt.scatter(x, y, c=colors, cmap='viridis')

plt.colorbar()

plt.show()
```

Colormap	Reverse Colormap
Accent	Accent_r

Blues	Blues_r
BrBG	BrBG_r
BuGn	BuGn_r
BuPu	BuPu_r
CMRmap	CMRmap_r
Dark2	Dark2_r
GnBu	GnBu_r
Greens	Greens_r
Greys	Greys_r
OrRd	OrRd_r
Oranges	Oranges_r
PRGn	PRGn_r
Paired	Paired_r
Pastel1	Pastel1_r
Pastel2	Pastel2_r
PiYG	PiYG_r
PuBu	PuBu_r
PuBuGn	PuBuGn_r
PuOr	PuOr_r
PuRd	PuRd_r
Purples	Purples_r
RdBu	RdBu_r
RdGy	RdGy_r
RdPu	RdPu_r
RdYlBu	RdYlBu_r
RdYlGn	RdYlGn_r
Reds	Reds_r
Set1	Set1_r
Set2	Set2_r
Set3	Set3_r
Spectral	Spectral_r

Wistia	Wistia_r
YlGn	YlGn_r
YlGnBu	YlGnBu_r
YlOrBr	YlOrBr_r
YlOrRd	YlOrRd_r
afmhot	afmhot_r
autumn	autumn_r
binary	binary_r
bone	bone_r
brg	brg_r
bwr	bwr_r
cividis	cividis_r
cool	cool_r
coolwarm	coolwarm_r
copper	copper_r
cubehelix	cubehelix_r
flag	flag_r
gist_earth	gist_earth_r
gist_gray	gist_gray_r
gist_heat	gist_heat_r
gist_ncar	gist_ncar_r
gist_rainbow	gist_rainbow_r
gist_stern	gist_stern_r
gist_yarg	gist_yarg_r
gnuplot	gnuplot_r
gnuplot2	gnuplot2_r
gray	gray_r
hot	hot_r
hsv	hsv_r
inferno	inferno_r
jet	jet_r

magma	magma_r
nipy_spectral	nipy_spectral_r
ocean	ocean_r
pink	pink_r
plasma	plasma_r
prism	prism_r
rainbow	rainbow_r
seismic	seismic_r
spring	spring_r
summer	summer_r
tab10	tab10_r
tab20	tab20_r
tab20b	tab20b_r
tab20c	tab20c_r
terrain	terrain_r
twilight	twilight_r
twilight_shifted	twilight_shifted_r
viridis	viridis_r
winter	winter_r

Size

You can change the size of the dots with the `s` argument.

Just like colors, make sure the array for sizes has the same length as the arrays for the x- and y-axis:

Set your own size for the markers:

```
import matplotlib.pyplot as plt
import numpy as np

x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
y = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])
sizes = np.array([20,50,100,200,500,1000,60,90,10,300,600,800,75])

plt.scatter(x, y, s=sizes)
```

```
plt.show()
```

Alpha

You can adjust the transparency of the dots with the `alpha` argument.

Just like colors, make sure the array for sizes has the same length as the arrays for the x- and y-axis:

Set your own size for the markers:

```
import matplotlib.pyplot as plt
import numpy as np

x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
y = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])
sizes = np.array([20,50,100,200,500,1000,60,90,10,300,600,800,75])

plt.scatter(x, y, s=sizes, alpha=0.5)

plt.show()
```

Combine Color Size and Alpha

You can combine a colormap with different sizes of the dots. This is best visualized if the dots are transparent:

Create random arrays with 100 values for x-points, y-points, colors and sizes:

```
import matplotlib.pyplot as plt
import numpy as np

x = np.random.randint(100, size=(100))
y = np.random.randint(100, size=(100))
colors = np.random.randint(100, size=(100))
sizes = 10 * np.random.randint(100, size=(100))

plt.scatter(x, y, c=colors, s=sizes, alpha=0.5, cmap='nipy_spectral')

plt.colorbar()

plt.show()
note - 0.80
```

Matplotlib Bars

Creating Bars

With Pyplot, you can use the `bar()` function to draw bar graphs:

Draw 4 bars:

```
import matplotlib.pyplot as plt
import numpy as np

x = np.array(["A", "B", "C", "D"])
y = np.array([3, 8, 1, 10])

plt.bar(x,y)
plt.show()
```

Horizontal Bars

If you want the bars to be displayed horizontally instead of vertically, use the `barh()` function:

Draw 4 horizontal bars:

```
import matplotlib.pyplot as plt
import numpy as np

x = np.array(["A", "B", "C", "D"])
y = np.array([3, 8, 1, 10])

plt.barh(x, y)
plt.show()
```

Bar Color

The `bar()` and `barh()` take the keyword argument `color` to set the color of the bars:

Draw 4 red bars:

```
import matplotlib.pyplot as plt
import numpy as np

x = np.array(["A", "B", "C", "D"])
```

```
y = np.array([3, 8, 1, 10])

plt.bar(x, y, color = "red")
plt.show()
```

Bar Width

The `bar()` takes the keyword argument `width` to set the width of the bars:

Draw 4 very thin bars:

```
import matplotlib.pyplot as plt
import numpy as np
x = np.array(["A", "B", "C", "D"])
y = np.array([3, 8, 1, 10])
plt.bar(x, y, width = 0.1)
plt.show()
```

The default width value is 0.8

Note: For horizontal bars, use `height` instead of `width`.

Bar Height

The `barh()` takes the keyword argument `height` to set the height of the bars:

Draw 4 skinny bars:

```
import matplotlib.pyplot as plt
import numpy as np

x = np.array(["A", "B", "C", "D"])
y = np.array([3, 8, 1, 10])

plt.barh(x, y, height = 0.1)
plt.show()
```

Matplotlib Histograms

Histogram

A histogram is a graph showing *frequency* distributions.

It is a graph showing the number of observations within each given interval.

Example: Say you ask for the height of 250 people, you might end up with a histogram like this:

You can read from the histogram that there are approximately:

2 people from 140 to 145cm
5 people from 145 to 150cm
15 people from 151 to 156cm
31 people from 157 to 162cm
46 people from 163 to 168cm
53 people from 168 to 173cm
45 people from 173 to 178cm
28 people from 179 to 184cm
21 people from 185 to 190cm
4 people from 190 to 195cm

Create Histogram

In Matplotlib, we use the `hist()` function to create histograms.

The `hist()` function creates a histogram using an array of numbers; the array is sent into the function as an argument.

For simplicity, we use NumPy to randomly generate an array with 250 values, with 170 values concentrating and a standard deviation of 10. Learn more about [Normal Data Distribution] in our [Machine Learning Tutorial].
A Normal Data Distribution by NumPy:

```
import numpy as np
x = np.random.normal(170, 10, 250)
print(x)
```

Result:

This will generate a *random* result, and could look like this:

The hist() function will read the array and produce a histogram:

A simple histogram:

```
import matplotlib.pyplot as plt

import numpy as np


x = np.random.normal(170, 10, 250)


plt.hist(x)

plt.show()
```

Pie Charts

Labels

Add labels to the pie chart with the labels parameter.

The labels parameter must be an array with one label for each wedge:

A simple pie chart:

```
import matplotlib.pyplot as plt

import numpy as np


y = np.array([35, 25, 25, 15])

mylabels = ["Apples", "Bananas", "Cherries", "Dates"]


plt.pie(y, labels = mylabels)

plt.show()
```

Start Angle

As mentioned the default start angle is at the x-axis, but you can change the start angle by specifying a `startangle` parameter.

The `startangle` parameter is defined with an angle in degrees, default angle is 0:

Start the first wedge at 90 degrees:

```
import matplotlib.pyplot as plt

import numpy as np


y = np.array([35, 25, 25, 15])

mylabels = ["Apples", "Bananas", "Cherries", "Dates"]


plt.pie(y, labels = mylabels, startangle = 90)

plt.show()
```

Explode

Maybe you want one of the wedges to stand out? The `explode` parameter allows you to do that.

The `explode` parameter, if specified, and not `None`, must be an array with one value for each wedge.

Each value represents how far from the center each wedge is displayed:

Pull the "Apples" wedge 0.2 from the center of the pie:

```
import matplotlib.pyplot as plt
import numpy as np
y = np.array([35, 25, 25, 15])
mylabels = ["Apples", "Bananas", "Cherries", "Dates"]
myexplode = [0.2, 0, 0, 0]
plt.pie(y, labels = mylabels, explode = myexplode)
plt.show()
```

Shadow

Add a shadow to the pie chart by setting the `shadows` parameter to `True`:

Add a shadow:

```
import matplotlib.pyplot as plt

import numpy as np


y = np.array([35, 25, 25, 15])

mylabels = ["Apples", "Bananas", "Cherries", "Dates"]

myexplode = [0.2, 0, 0, 0]


plt.pie(y, labels = mylabels, explode = myexplode, shadow = True)

plt.show()
```

Colors

You can set the color of each wedge with the `colors` parameter.

The `colors` parameter, if specified, must be an array with one value for each wedge:

Specify a new color for each wedge:

```
import matplotlib.pyplot as plt

import numpy as np


y = np.array([35, 25, 25, 15])

mylabels = ["Apples", "Bananas", "Cherries", "Dates"]

mycolors = ["black", "hotpink", "b", "#4CAF50"]

plt.pie(y, labels = mylabels, colors = mycolors)

plt.show()
```

You can use Hexadecimal color values, any of the 140 supported color names, or one of these shortcuts:

'r' - Red
'g' - Green
'b' - Blue
'c' - Cyan
'm' - Magenta
'y' - Yellow
'k' - Black
'w' - White

Legend

To add a list of explanations for each wedge, use the `legend()` function:

Add a legend:

```
import matplotlib.pyplot as plt

import numpy as np

y = np.array([35, 25, 25, 15])

mylabels = ["Apples", "Bananas", "Cherries", "Dates"]

plt.pie(y, labels = mylabels)

plt.legend()

plt.show()
```

Legend With Header

To add a header to the legend, add the `title` parameter to the `legend` function.
Add a legend with a header:

```
import matplotlib.pyplot as plt

import numpy as np

y = np.array([35, 25, 25, 15])

mylabels = ["Apples", "Bananas", "Cherries", "Dates"]
plt.pie(y, labels = mylabels)
plt.legend(title = "Four Fruits:")
plt.show()
```

Seaborn

Visualize Distributions With Seaborn

Seaborn is a library that uses Matplotlib underneath to plot graphs. It will be used to visualize random distributions.

nstall Seaborn.

If you have Python and PIP already installed on a system, install it using this command:

```
C:\Users\Your Name>pip install seaborn
```

If you use Jupyter, install Seaborn using this command:

```
C:\Users\Your Name>!pip install seaborn
```

Distplots

Distplot stands for distribution plot, it takes as input an array and plots a curve corresponding to the distribution of points in the array.

Import Matplotlib

Import the pyplot object of the Matplotlib module in your code using the following statement:

```
import matplotlib.pyplot as plt
```

You can learn about the Matplotlib module in our Matplotlib Tutorial.

Import Seaborn

Import the Seaborn module in your code using the following statement:

```
import seaborn as sns
```

Plotting a Distplot

```
import matplotlib.pyplot as plt

import seaborn as sns

sns.distplot([0, 1, 2, 3, 4, 5])

plt.show()
```

Plotting a Distplot Without the Histogram

```
import matplotlib.pyplot as plt

import seaborn as sns

sns.distplot([0, 1, 2, 3, 4, 5], hist=False)

plt.show()
```

Normal (Gaussian) Distribution

Normal Distribution

The Normal Distribution is one of the most important distributions.

It is also called the Gaussian Distribution after the German mathematician Carl Friedrich Gauss.

It fits the probability distribution of many events, eg. IQ Scores, Heartbeat etc.

Use the `random.normal()` method to get a Normal Data Distribution.
It has three parameters:
`loc` - (Mean) where the peak of the bell exists.
`scale` - (Standard Deviation) how flat the graph distribution should be.
`size` - The shape of the returned array.

Generate a random normal distribution of size 2x3:

```
from numpy import random

x = random.normal(size=(2, 3))

print(x)
```

Generate a random normal distribution of size 2x3 with mean at 1 and standard deviation of 2:

```
from numpy import random

x = random.normal(loc=1, scale=2, size=(2, 3))

print(x)
```

Visualization of Normal Distribution

```
from numpy import random

import matplotlib.pyplot as plt

import seaborn as sns

sns.distplot(random.normal(size=1000), hist=False)

plt.show()
```

Binomial Distribution

Binomial Distribution

Binomial Distribution is a *Discrete Distribution*.

It describes the outcome of binary scenarios, e.g. toss of a coin, it will either be head or tails.

It has three parameters:

`n` - number of trials.
`p` - probability of occurence of each trial (e.g. for toss of a coin 0.5 each).
`size` - The shape of the returned array.

Given 10 trials for coin toss generate 10 data points:

```
from numpy import random

x = random.binomial(n=10, p=0.5, size=10)

print(x)
```

Visualization of Binomial Distribution

```
from numpy import random

import matplotlib.pyplot as plt

import seaborn as sns

sns.distplot(random.binomial(n=10, p=0.5, size=1000), hist=True,
kde=False)

plt.show()
```

Difference Between Normal and Binomial Distribution

The main difference is that normal distribution is continous whereas binomial is discrete, but if there are enough data points it will be quite similar to normal distribution with certain loc and scale.

```
from numpy import random

import matplotlib.pyplot as plt

import seaborn as sns

sns.distplot(random.normal(loc=50, scale=5, size=1000), hist=False,
label='normal')

sns.distplot(random.binomial(n=100, p=0.5, size=1000), hist=False,
label='binomial')

plt.show()
```


Poisson Distribution
Uniform Distribution
Logistic Distribution
Multinomial Distribution
Exponential Distribution
Chi Square Distribution
Rayleigh Distribution
Pareto Distribution
Zipf Distribution

Poisson Distribution

Poisson Distribution is a *Discrete Distribution*.

It estimates how many times an event can happen in a specified time. e.g. If someone eats twice a day what is the probability he will eat thrice?

It has two parameters:

`lam` - rate or known number of occurrences e.g. 2 for above problem.

`size` - The shape of the returned array.

Generate a random 1x10 distribution for occurrence 2:

```
from numpy import random


x = random.poisson(lam=2, size=10)


print(x)
```

Visualization of Poisson Distribution

```
from numpy import random

import matplotlib.pyplot as plt

import seaborn as sns


sns.distplot(random.poisson(lam=2, size=1000), kde=False)


plt.show()
```

Difference Between Normal and Poisson Distribution

Normal distribution is continuous whereas poisson is discrete.

But we can see that similar to binomial for a large enough poisson distribution it will become similar to normal distribution with certain std dev and mean.

```
from numpy import random

import matplotlib.pyplot as plt

import seaborn as sns



sns.distplot(random.normal(loc=50, scale=7, size=1000), hist=False, label='normal')

sns.distplot(random.poisson(lam=50, size=1000), hist=False, label='poisson')



plt.show()
```

Difference Between Binomial and Poisson Distribution

Binomial distribution only has two possible outcomes, whereas poisson distribution can have unlimited possible outcomes.

But for very large `n` and near-zero `p` binomial distribution is near identical to poisson distribution such that `n * p` is nearly equal to `lam`.

```
from numpy import random

import matplotlib.pyplot as plt

import seaborn as sns



sns.distplot(random.binomial(n=1000, p=0.01, size=1000), hist=False,
label='binomial')

sns.distplot(random.poisson(lam=10, size=1000), hist=False,
label='poisson')



plt.show()
```

Uniform Distribution

Uniform Distribution

Used to describe probability where every event has equal chances of occuring.

E.g. Generation of random numbers.

It has three parameters:

`low` - lower bound - default 0 .0.

`high` - upper bound - default 1.0.

`size` - The shape of the returned array.

Create a 2x3 uniform distribution sample:

```
from numpy import random

x = random.uniform(size=(2, 3))

print(x)
```

Visualization of Uniform Distribution

```
from numpy import random
import matplotlib.pyplot as plt
import seaborn as sns

sns.distplot(random.uniform(size=1000), hist=False)

plt.show()
```

Logistic Distribution

Logistic Distribution is used to describe growth.

Used extensively in machine learning in logistic regression, neural networks etc.

It has three parameters:

`loc` - mean, where the peak is. Default 0.

`scale` - standard deviation, the flatness of distribution. Default 1.

`size` - The shape of the returned array.

Draw 2x3 samples from a logistic distribution with mean at 1 and stddev 2.0:

```
from numpy import random

x = random.logistic(loc=1, scale=2, size=(2, 3))

print(x)
```

Visualization of Logistic Distribution

```
from numpy import random

import matplotlib.pyplot as plt

import seaborn as sns

sns.distplot(random.logistic(size=1000), hist=False)

plt.show()
```

Difference Between Logistic and Normal Distribution

Both distributions are near identical, but logistic distribution has more area under the tails, meaning it represents more possibility of occurrence of an event further away from mean.

For higher value of scale (standard deviation) the normal and logistic distributions are near identical apart from the peak.

```
from numpy import random

import matplotlib.pyplot as plt

import seaborn as sns

sns.distplot(random.normal(scale=2, size=1000), hist=False,
label='normal')

sns.distplot(random.logistic(size=1000), hist=False, label='logistic')

plt.show()
```

Multinomial Distribution

Multinomial Distribution

The multinomial distribution is a generalization of binomial distribution.

It describes outcomes of multi-nomial scenarios, unlike binomial where scenarios must be only one of two. e.g. Blood type of a population, dice roll outcome.

It has three parameters:

`n` - number of possible outcomes (e.g. 6 for dice roll).

`pvals` - list of probabilties of outcomes (e.g. [1/6, 1/6, 1/6, 1/6, 1/6, 1/6] for dice roll).

`size` - The shape of the returned array.

Draw out a sample for dice roll:

```
from numpy import random

x = random.multinomial(n=6, pvals=[1/6, 1/6, 1/6, 1/6, 1/6, 1/6])

print(x)
```

Note: Multinomial samples will NOT produce a single value! They will produce one value for each `pval`.

Note: As they are generalization of binomial distribution their visual representation and similarity of normal distribution is same as that of multiple binomial distributions.

Exponential Distribution

Exponential distribution is used for describing time till next event e.g. failure/success etc.

It has two parameters:

`scale` - inverse of rate (see lam in poisson distribution) defaults to 1.0.

`size` - The shape of the returned array.

Draw out a sample for exponential distribution with 2.0 scale with 2x3 size:

```
from numpy import random

x = random.exponential(scale=2, size=(2, 3))

print(x)
```

Visualization of Exponential Distribution

```
from numpy import random

import matplotlib.pyplot as plt

import seaborn as sns


sns.distplot(random.exponential(size=1000), hist=False)


plt.show()
```

Relation Between Poisson and Exponential Distribution

Poisson distribution deals with a number of occurences of an event in a time period whereas exponential distribution deals with the time between these events.

Chi-Square Distribution

Chi Square Distribution

Chi Square distribution is used as a basis to verify the hypothesis.

It has two parameters:

`df` - (degree of freedom).

`size` - The shape of the returned array.

Draw out a sample for chi squared distribution with degree of freedom 2 with size 2x3:

```
from numpy import random

x = random.chisquare(df=2, size=(2, 3))

print(x)
```

Visualization of Chi Square Distribution

```
from numpy import random

import matplotlib.pyplot as plt

import seaborn as sns

sns.distplot(random.chisquare(df=1, size=1000), hist=False)

plt.show()
```

Rayleigh Distribution

Rayleigh Distribution

Rayleigh distribution is used in signal processing.

It has two parameters:

`scale` - (standard deviation) decides how flat the distribution will be default 1.0).

`size` - The shape of the returned array.

Draw out a sample for rayleigh distribution with scale of 2 with size 2x3:

```
from numpy import random

x = random.rayleigh(scale=2, size=(2, 3))

print(x)
```

Visualization of Rayleigh Distribution

```
from numpy import random

import matplotlib.pyplot as plt

import seaborn as sns

sns.distplot(random.rayleigh(size=1000), hist=False)

plt.show()
```

Similarity Between Rayleigh and Chi Square Distribution

At unit stddev and 2 degrees of freedom rayleigh and chi square represent the same distributions.

Pareto Distribution

Pareto Distribution

A distribution following Pareto's law i.e. 80-20 distribution (20% factors cause 80% outcome).

It has two parameter:

`a` - shape parameter.

`size` - The shape of the returned array.

Draw out a sample for pareto distribution with shape of 2 with size 2x3:

```
from numpy import random

x = random.pareto(a=2, size=(2, 3))

print(x)
```

Visualization of Pareto Distribution

Example

```
from numpy import random

import matplotlib.pyplot as plt

import seaborn as sns

sns.distplot(random.pareto(a=2, size=1000), kde=False)

plt.show()
```

Zipf Distribution

Zipf distributions are used to sample data based on zipf's law.

Zipf's Law: In a collection, the nth common term is 1/n times of the most common term. E.g. the 5th most common word in English occurs nearly 1/5 times as often as the most common word.

It has two parameters:

`a` - distribution parameter.

`size` - The shape of the returned array.

Draw out a sample for zipf distribution with distribution parameter 2 with size 2x3:

```
from numpy import random

x = random.zipf(a=2, size=(2, 3))

print(x)
```

Visualization of Zipf Distribution

Sample 1000 points but plotting only ones with value < 10 for more meaningful chart.

```
from numpy import random

import matplotlib.pyplot as plt

import seaborn as sns

x = random.zipf(a=2, size=1000)

sns.distplot(x[x<10], kde=False)

plt.show()
```