ML_prerequisites

August 19, 2022

0.0.1 Linear Algebra

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
```

[2]:
```
np.linspace(0,100,10) # start, finish, number of points
```

[2]:
```
array([  0.        ,  11.11111111,  22.22222222,  33.33333333,
        44.44444444,  55.55555556,  66.66666667,  77.77777778,
        88.88888889, 100.        ])
```

[3]:
```
f = np.linspace(0,100,9) # start, finish, number of points
f
```

[3]:
```
array([  0. ,  12.5,  25. ,  37.5,  50. ,  62.5,  75. ,  87.5, 100. ])
```

[4]:
```
f*3.14
```

[4]:
```
array([  0.  ,  39.25,  78.5 , 117.75, 157.  , 196.25, 235.5 , 274.75,
       314.  ])
```

$y = 3(f + 5)$

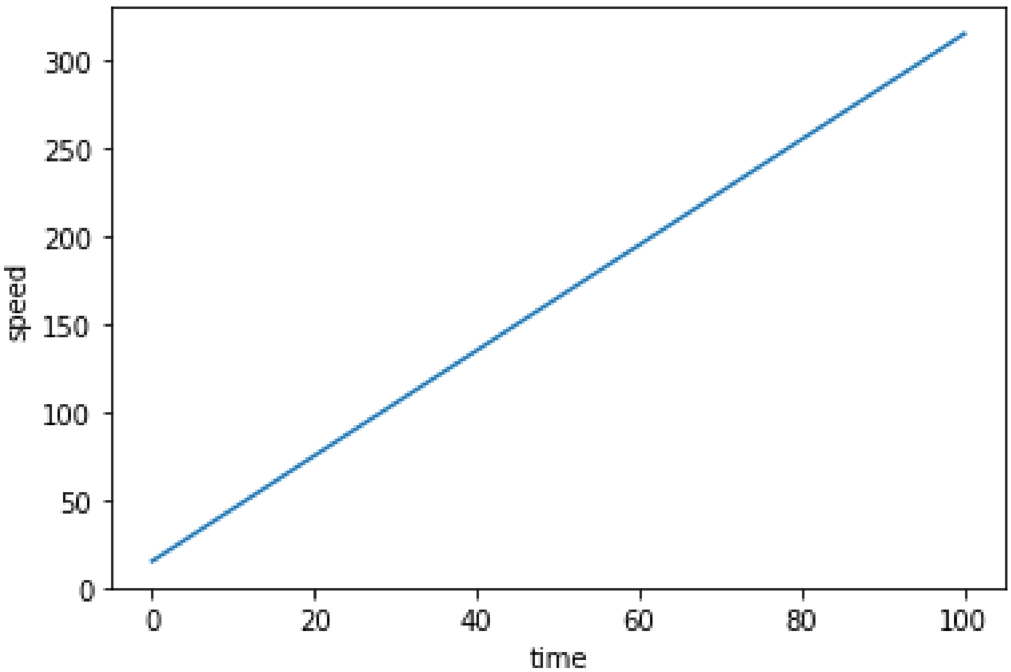
[5]:
```
y = 3*(f + 5)
```

[6]:
```
y
```

[6]:
```
array([ 15. ,  52.5,  90. , 127.5, 165. , 202.5, 240. , 277.5, 315. ])
```

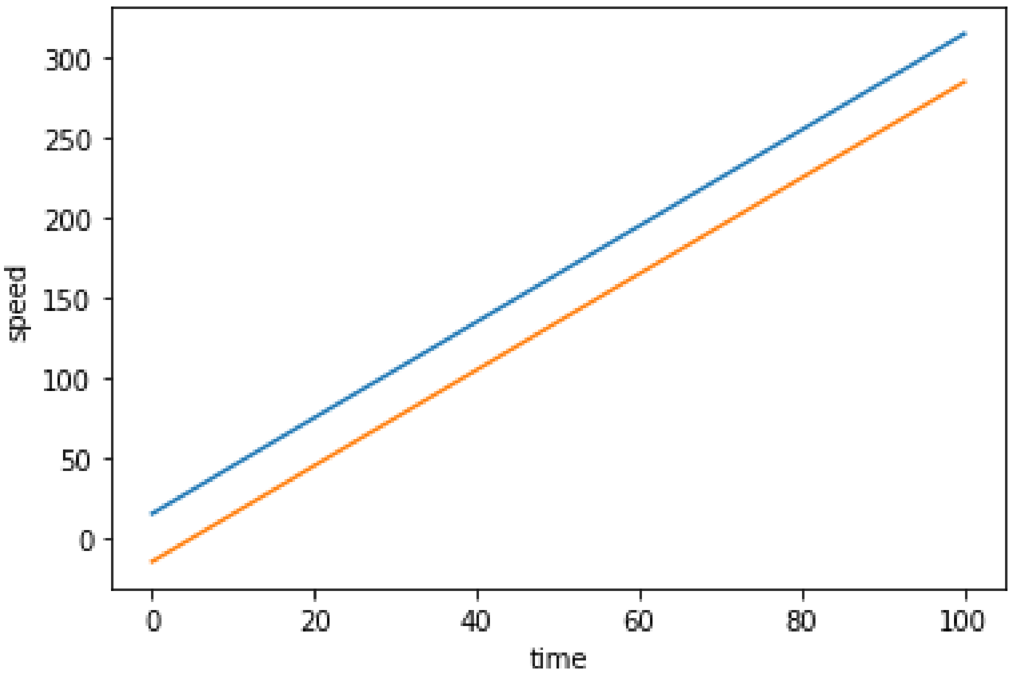
Plot f vs y

[7]:
```
fig, ax = plt.subplots()
plt.xlabel('time')
plt.ylabel('speed')
ax.plot(f, y)
plt.show()
```



[8]:
```
Y = 3*(f - 5)
```

[9]:
```
fig, ax = plt.subplots()
plt.xlabel('time')
plt.ylabel('speed')
ax.plot(f, y)
ax.plot(f, Y)
plt.show()
```



0.0.2 Scalars

[10]:
```
x = 15
x
```

[10]: 15

[11]:
```
y = 40
y
```

[11]: 40

[12]:
```
the_sum = x + y
the_sum
```

[12]: 55

0.0.3 Vectors

[13]:
```
x = np.array([12, 10, -4])
x
```

[13]: array([12, 10, -4])

[14]:
```
len(x), x.shape, len(x.shape)
```

[14]: (3, (3,), 1)

[15]:
```
x[0], x[1]
```

[15]: (12, 10)

[16]:
```
x[-1]
```

[16]: -4

Transpose

[17]:
```
x_t = x.T
x_t
```

[17]: array([12, 10, -4])

[18]:
```
x = np.array([[12, 14, -4],
              [90, 30, 20]])
x
```

[18]:
```
array([[12, 14, -4],
       [90, 30, 20]])
```

[19]:
```
x.shape, len(x.shape)
```

[19]: ((2, 3), 2)

[20]:
```
x_t = x.T
x_t
```

[20]:
```
array([[12, 90],
       [14, 30],
       [-4, 20]])
```

[21]:
```
x_t.shape
```

[21]: (3, 2)

[22]:
```
X = np.array([[16, 3],
              [2, 4],
              [10, 20]])
X
```

[22]:
```
array([[16,  3],
       [ 2,  4],
       [10, 20]])
```

[23]:
```
X*2
```

[23]:
```
array([[32,  6],
       [ 4,  8],
       [20, 40]])
```

[24]:
```
X*2 + 2
```

[24]:
```
array([[34,  8],
       [ 6, 10],
       [22, 42]])
```

[25]:
```
X.sum()
```

[25]:
```
55
```

[26]:
```
X.sum(axis = 1)
```

[26]:
```
array([19,  6, 30])
```

[37]:
```
X.sum(axis = 0)
```

[37]:
```
array([28, 27])
```

0.0.4 The Dot Product

[38]:
```
x = np.array([20, 10, 0])
y = np.array([13, 15, 5])
np.dot(x,y)
```

[39]:
```
np.dot(x,y)
```

[39]:
```
410
```

Task 1

1. Write a function which can take in values either two scalars or two vectors and perform the following operations
   a. Multiplication
   b. Addition
   c. Subtraction

[27]:
```
def my_func(X,Y,operation):

    if operation == 'add':
        result = X + Y
    elif operation == 'sub':
        result = X - Y
    else:
        if len(X)>1:
            result = np.dot(X,Y)
        else:
            result = X*Y
    return result
```

[28]:
```
x = np.array([20, 10, 0])
y = np.array([13, 15, 5])
np.dot(x,y)
```

[28]:
```
410
```

[29]:
```
my_func(np.array([10]),np.array([20]),'dot')
```

[29]:
```
array([200])
```

[30]:
```
my_func(np.array([10]),np.array([20]),'add')
```

[30]:
```
array([30])
```

[31]:
```
my_func(np.array([10]),np.array([20]),'sub')
```

[31]:
```
array([-10])
```

0.0.5 Eigen Values and Eigen Vectors

[32]:
```
A = np.array([[1, 2],
              [3, 4]])
A
```

[32]:
```
array([[1, 2],
       [3, 4]])
```

[33]:
```
lambdas, V = np.linalg.eig(A)
```

[34]:
```
V
```

[34]:
```
array([[-0.82456484, -0.41597356],
       [ 0.56576746, -0.90937671]])
```

[35]:
```
lambdas
```

[35]:
```
array([-0.37228132,  5.37228132])
```

[36]:
```
v = V[:,1]
v
```

[36]:
```
array([-0.41597356, -0.90937671])
```

[37]:
```
lambda_ = lambdas[1]
lambda_
```

[37]:
```
5.372281323269014
```

[70]:
```
Av = np.dot(A, v)
Av
```

[70]:
```
array([-2.23472698, -4.88542751])
```

[71]:
```
lambda_ * v
```

[71]:
```
array([-2.23472698, -4.88542751])
```

0.0.6 Probability

Calculate the probability of throwing three heads in five tosses

[38]:
```
from math import factorial
```

[39]:
```
def flipcoin_(n, k):
    n_c_k = factorial(n)/(factorial(k)*factorial(n-k))
    return n_c_k/2**n
```

[40]:
```
flipcoin_(5,3)
```

[40]:
```
0.3125
```

[41]:
```
[h for h in range(6)]
```

[41]:
```
[0, 1, 2, 3, 4, 5]
```

[42]:
```
list(zip([0,1,2,3,4,5],[flipcoin_(5,h) for h in range(6)]))
```

[42]:
```
[(0, 0.03125),
 (1, 0.15625),
 (2, 0.3125),
```

```
 (3, 0.3125),
 (4, 0.15625),
 (5, 0.03125)]
```

[ ]:
```
#Number of Coin Tosses
```

[48]:
```
ns = np.array([2, 4, 6, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096])
ns
```

[48]:
```
array([   2,    4,    6,    8,   16,   32,   64,  128,  256,  512, 1024,
       2048, 4096])
```

[49]:
```
np.random.seed(42)
```

[50]:
```
np.random.binomial(10,0.15)     # Number of flips, prob of heads should return␣
 ↪the number of flips which are heads
```

[50]:
```
1
```

[55]:
```
heads_count = [np.random.binomial(n, 0.5) for n in ns]
heads_count
```

[55]:
```
[2, 1, 3, 4, 5, 17, 34, 75, 134, 244, 528, 1055, 2088]
```

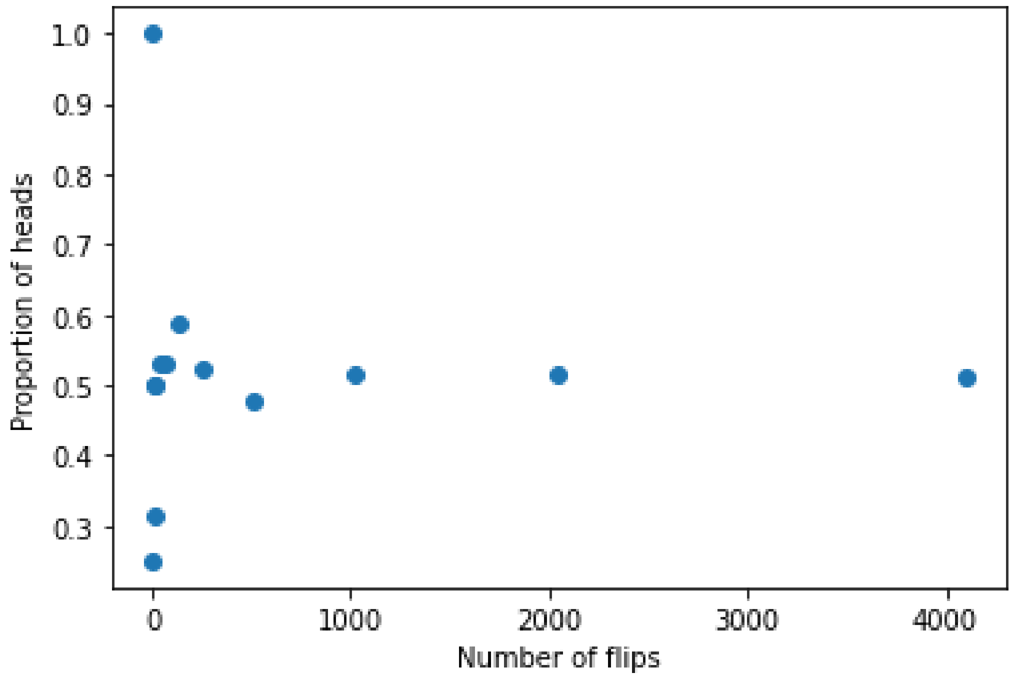
[56]:
```
list(zip(ns,heads_count))
```

[56]:
```
[(2, 2),
 (4, 1),
 (6, 3),
 (8, 4),
 (16, 5),
 (32, 17),
 (64, 34),
 (128, 75),
 (256, 134),
 (512, 244),
 (1024, 528),
 (2048, 1055),
 (4096, 2088)]
```

[57]:
```
heads_proportion = heads_count/ns  #this gives if i am tossing coin for these␣
 ↪no of times then 0.75 ot the times i am going to get a head
heads_proportion
```

[57]:
```
array([1.        , 0.25      , 0.5       , 0.5       , 0.3125    ,
       0.53125   , 0.53125   , 0.5859375 , 0.5234375 , 0.4765625 ,
       0.515625  , 0.51513672, 0.50976562])
```

[58]:
```
plt.xlabel('Number of flips')
plt.ylabel('Proportion of heads')
plt.scatter(ns, heads_proportion)
plt.show()
```



0.0.7 Statistics

[59]:
```
num_of_experiments = 1000
heads_count = np.random.binomial(5, 0.5, num_of_experiments)
# heads_count
```

[60]:
```
heads, counts = np.unique(heads_count, return_counts=True)
```

[61]:
```
heads
```

[61]:
```
array([0, 1, 2, 3, 4, 5])
```

[62]:
```
counts
```

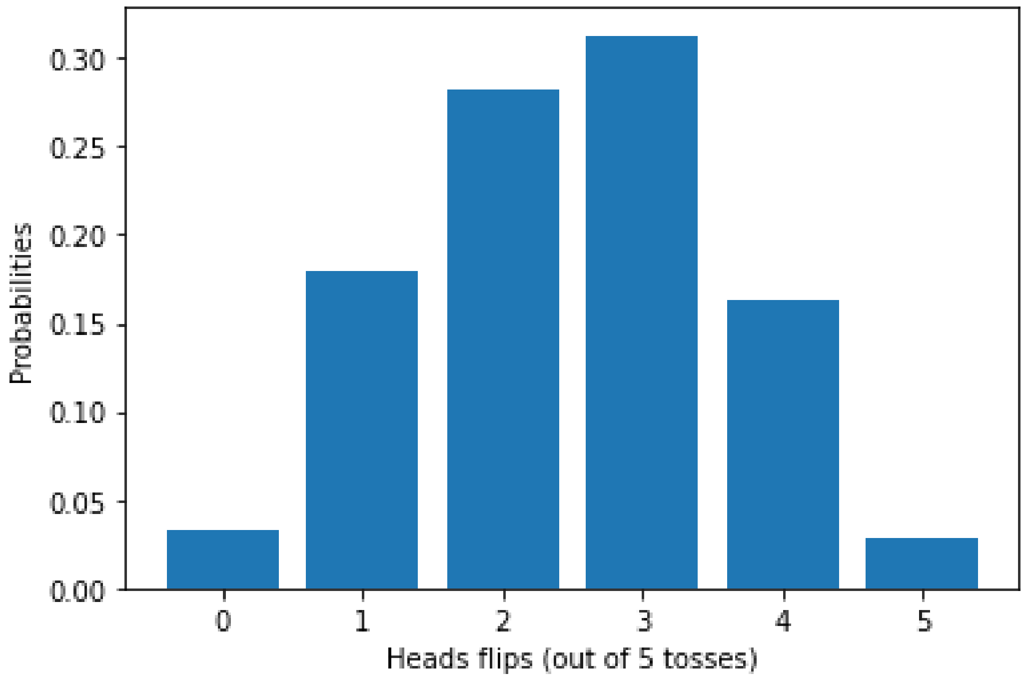
[62]:
```
array([ 33, 179, 283, 313, 163,  29])
```

[63]:
```
pro_event = counts/num_of_experiments
pro_event
```

[63]: array([0.033, 0.179, 0.283, 0.313, 0.163, 0.029])

[64]:
```
plt.bar(heads, pro_event)
plt.xlabel('Heads flips (out of 5 tosses)')
plt.ylabel('Probabilities')
```

[64]: Text(0, 0.5, 'Probabilities')



0.0.8 Measures of Central Tendency

[65]:
```
np.mean(heads_count)
```

[65]: 2.481

[66]:
```
np.median(heads_count)
```

[66]: 3.0

0.0.9 Measures of Dispersion

[67]:
```
## Variance
np.var(heads_count)
```

[67]: 1.305639

[68]:
```
## Standard Deviation
np.std(heads_count)
```

[68]: 1.1426456143529367

0.0.10 Measures of Realtedness

[69]:
```
from sklearn.datasets import load_iris
import pandas as pd

data = load_iris()
features = data.data
```

[70]:
```
data = pd.DataFrame(features, columns= data.feature_names)
data.head()
```

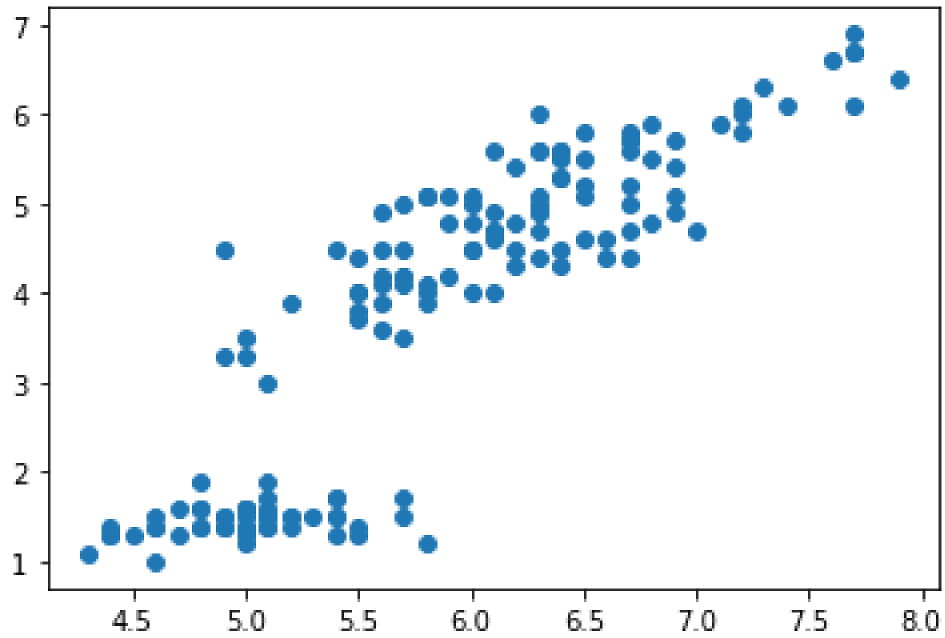
[70]:

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

[71]:
```
x = data['sepal length (cm)']
y = data['petal length (cm)']
```

[72]:
```
plt.scatter(x,y)
```

[72]: <matplotlib.collections.PathCollection at 0x7efbc5f72590>




```
[73]: ## Covariance
      np.cov(x, y)
```

```
[73]: array([[0.68569351, 1.27431544],
             [1.27431544, 3.11627785]])
```

```
[74]: np.var(x), np.var(y)
```

```
[74]: (0.6811222222222222, 3.0955026666666674)
```

```
[75]: from scipy.stats import pearsonr
```

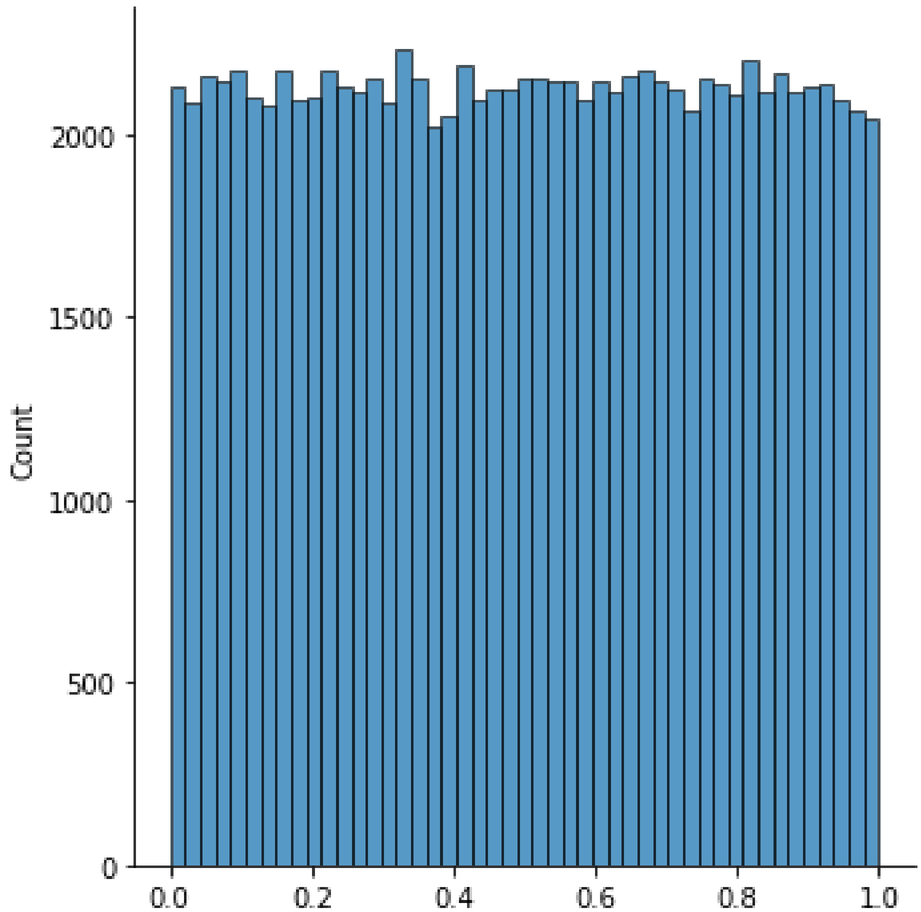
```
[76]: ## Correlation
      pearsonr(x,y)[0]
```

```
[76]: 0.8717537758865831
```

0.1 Distributions

```
[161]: import seaborn as sns
       uniform_distr = np.random.uniform(size = 100000)
       sns.displot(uniform_distr)
```

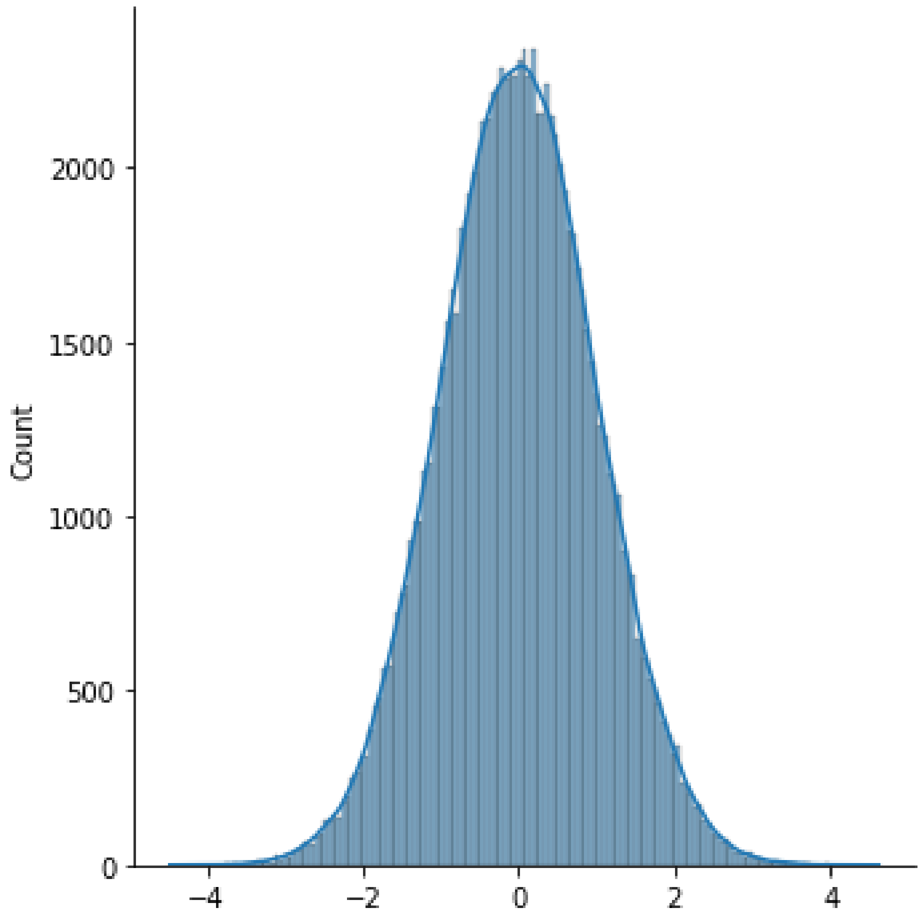
```
[161]: <seaborn.axisgrid.FacetGrid at 0x7fcb036cfe50>
```



[164]:
```
## Gaussian

gaussian_distribution = np.random.normal(size = 100000)
sns.displot(gaussian_distribution, kde = True)
```

[164]: <seaborn.axisgrid.FacetGrid at 0x7fcb03a2f2d0>



[165]:
```
np.mean(gaussian_distribution)
```

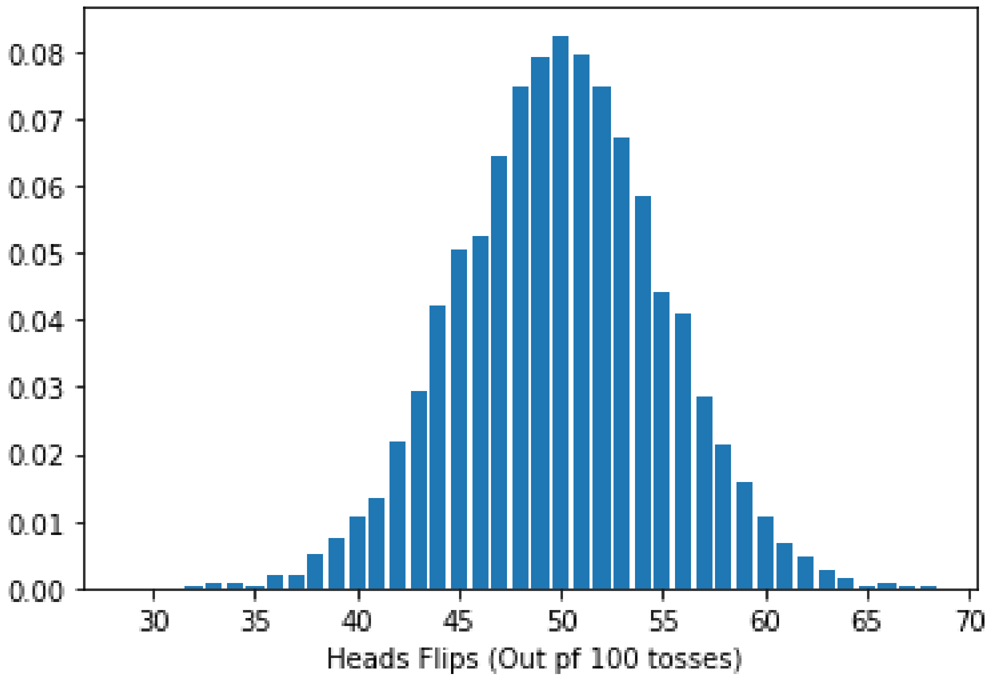
[165]: -0.004290446848348691

[166]:
```
np.std(gaussian_distribution)
```

[166]: 0.9998075949949708

[171]:
```
num_of_experiments = 10000
heads_count = np.random.binomial(100, 0.5, num_of_experiments)
heads, counts = np.unique(heads_count, return_counts= True)
pro_heads = counts/num_of_experiments
```

[174]:
```
plt.bar(heads, pro_heads)
plt.xlabel(' Heads Flips (Out pf 100 tosses)')
plt.show()
```


0.08
0.07
0.06
0.05
0.04
0.03
0.02
0.01
0.00
30
35
40
45
50
55
60
65
70
Heads Flips (Out pf 100 tosses)