190031920

A Nikhil Reddy

DS Practical 6

In [1]:
```
%matplotlib inline
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import scipy.stats as stats
```

In [2]:
```
gender_submission = pd.read_csv("gender_submission.csv")
gender_submission
```

Out[2]:

	PassengerId	Survived
0	892	0
1	893	1
2	894	0
3	895	0
4	896	1
...	...	...
413	1305	0
414	1306	1
415	1307	0
416	1308	0
417	1309	0

418 rows × 2 columns

In [3]:
```
test = pd.read_csv("test.csv")
test
```

Out[3]:

	PassengerId	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	892	3	Kelly, Mr. James	male	34.5	0	0	330911	7.8292	NaN	Q
1	893	3	Wilkes, Mrs. James (Ellen Needs)	female	47.0	1	0	363272	7.0000	NaN	S
2	894	2	Myles, Mr. Thomas Francis	male	62.0	0	0	240276	9.6875	NaN	Q
3	895	3	Wirz, Mr. Albert	male	27.0	0	0	315154	8.6625	NaN	S
4	896	3	Hirvonen, Mrs. Alexander (Helga E Lindqvist)	female	22.0	1	1	3101298	12.2875	NaN	S
...	...	...	...	...	...	...	...	...	...	...	...
413	1305	3	Spector, Mr. Woolf	male	NaN	0	0	A.5. 3236	8.0500	NaN	S
414	1306	1	Oliva y Ocana, Dona. Fermina	female	39.0	0	0	PC 17758	108.9000	C105	C
415	1307	3	Saether, Mr. Simon Sivertsen	male	38.5	0	0	SOTON/O.Q. 3101262	7.2500	NaN	S
416	1308	3	Ware, Mr. Frederick	male	NaN	0	0	359309	8.0500	NaN	S
417	1309	3	Peter, Master. Michael J	male	NaN	1	1	2668	22.3583	NaN	C

418 rows × 11 columns

In [4]:
```
train = pd.read_csv("train.csv")
train
```

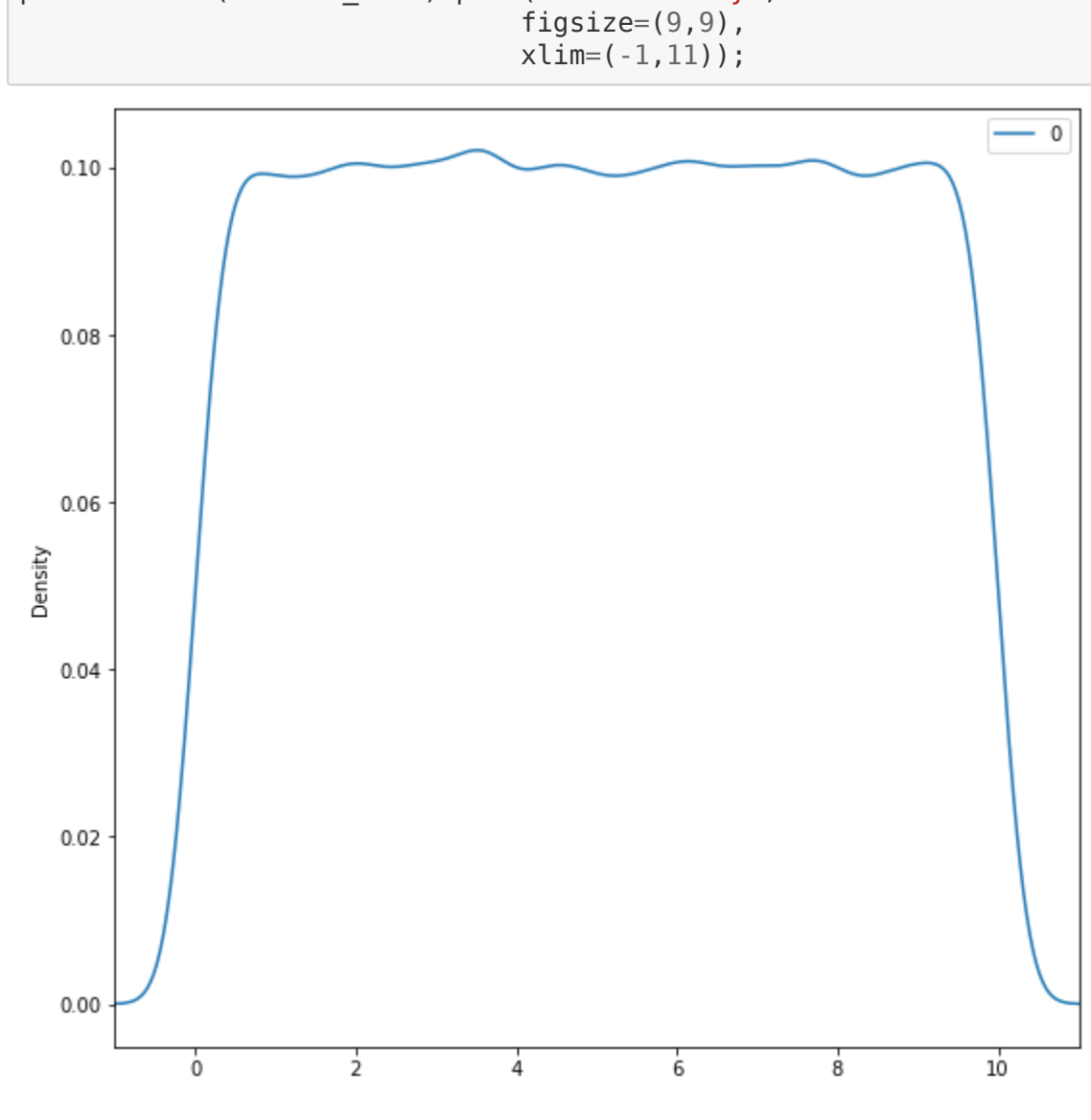
Out[4]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S
...	...	...	...	...	...	...	...	...	...	...	...	...
886	887	0	2	Montvila, Rev. Juozas	male	27.0	0	0	211536	13.0000	NaN	S
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	112053	30.0000	B42	S
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W./C. 6607	23.4500	NaN	S
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	111369	30.0000	C148	C
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	370376	7.7500	NaN	Q

891 rows × 12 columns

In [5]:
```
uniform_data = stats.uniform.rvs(size=100000,  # Generate 100000 numbers
                                 loc = 0,      # From 0
                                 scale=10)     # To 10
```

In [6]:
```
pd.DataFrame(uniform_data).plot(kind="density",  # Plot the distribution
                               figsize=(9,9),
                               xlim=(-1,11));
```



In [7]:
```
stats.uniform.cdf(x=2.5,
                  loc=0,
                  scale=10)
```

Out[7]: 0.25

In [8]:
```
stats.uniform.ppf(q=0.4,
                  loc=0,
                  scale=10)
```

Out[8]: 4.0

In [9]:
```
for x in range(-1,12,3):
    print("Density at x value " + str(x))
    print( stats.uniform.pdf(x, loc=0, scale=10) )
```

```
Density at x value -1
0.0
Density at x value 2
0.1
Density at x value 5
0.1
Density at x value 8
0.1
Density at x value 11
0.0
```

In [10]:
```
import random

random.randint(0,10)
```

Out[10]: 3

In [11]:
```
random.choice([2,4,6,9]) # Get a random element from a sequence
```

Out[11]: 2

In [12]:
```
random.random()         # Get a real number between 0 and 1
```

Out[12]: 0.1842998493668535

In [13]:
```
random.uniform(0,10)    # Get a real in the specified range
```

Out[13]: 7.0853763171845605

In [14]:
```
random.seed(12)  # Set the seed to an arbitrary value

print([random.uniform(0,10) for x in range(4)])

random.seed(12)  # Set the seed to the same value

print([random.uniform(0,10) for x in range(4)])
```

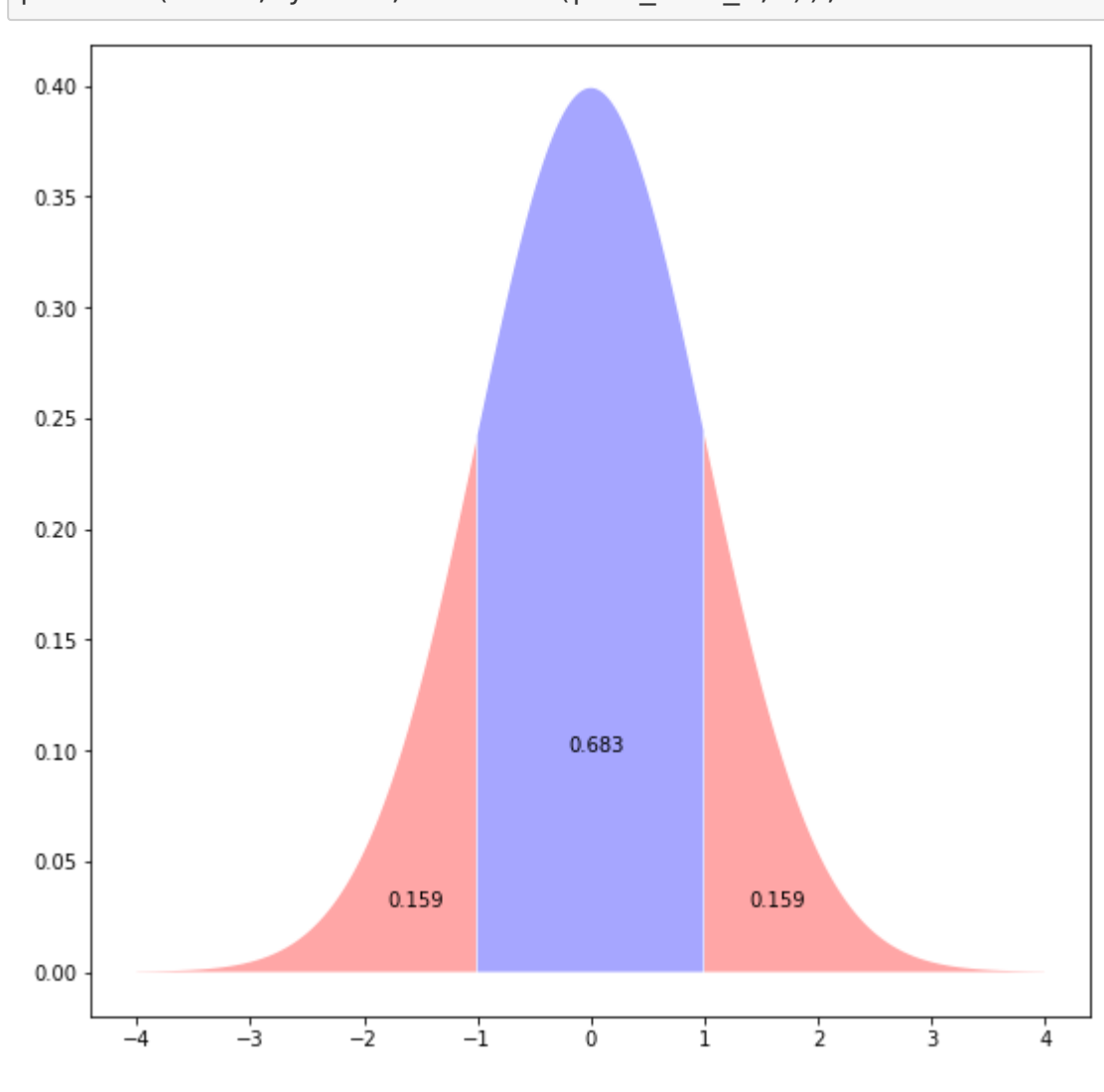
```
[4.7457067868854805, 6.574725026572553, 6.664104711248381, 1.4260035292536777]
[4.7457067868854805, 6.574725026572553, 6.664104711248381, 1.4260035292536777]
```

Normal Distribution

In [15]:
```
prob_under_minus1 = stats.norm.cdf(x= -1,
                                   loc = 0,
                                   scale= 1)

prob_over_1 = 1 - stats.norm.cdf(x= 1,
                                 loc = 0,
                                 scale= 1)

between_prob = 1-(prob_under_minus1+prob_over_1)

print(prob_under_minus1, prob_over_1, between_prob)
```

```
0.15865525393145707 0.15865525393145707 0.6826894921370859
```

In [16]:
```
# Plot normal distribution areas*

plt.rcParams["figure.figsize"] = (9,9)

plt.fill_between(x=np.arange(-4,-1,0.01),
                 y1= stats.norm.pdf(np.arange(-4,-1,0.01)) ,
                 facecolor='red',
                 alpha=0.35)

plt.fill_between(x=np.arange(1,4,0.01),
                 y1= stats.norm.pdf(np.arange(1,4,0.01)) ,
                 facecolor='red',
                 alpha=0.35)

plt.fill_between(x=np.arange(-1,1,0.01),
                 y1= stats.norm.pdf(np.arange(-1,1,0.01)) ,
                 facecolor='blue',
                 alpha=0.35)

plt.text(x=-1.8, y=0.03, s= round(prob_under_minus1,3))
plt.text(x=-0.2, y=0.1, s= round(between_prob,3))
plt.text(x=1.4, y=0.03, s= round(prob_over_1,3));
```



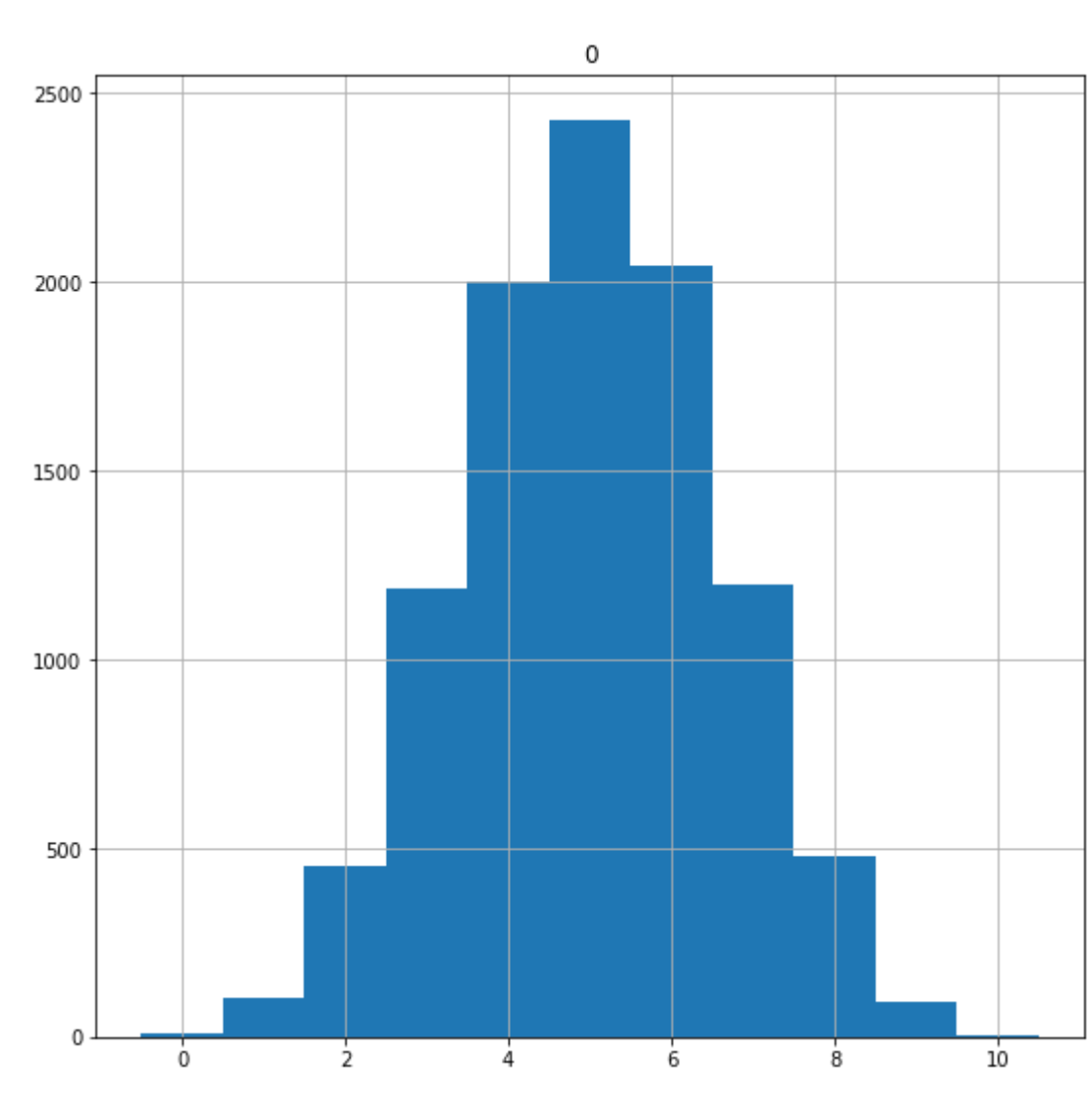
In [17]:
```
print( stats.norm.ppf(q=0.025) ) # Find the quantile for the 2.5% cutoff

print( stats.norm.ppf(q=0.975) ) # Find the quantile for the 97.5% cutoff
```

```
-1.9599639845400545
1.959963984540054
```

Binomial Distribution

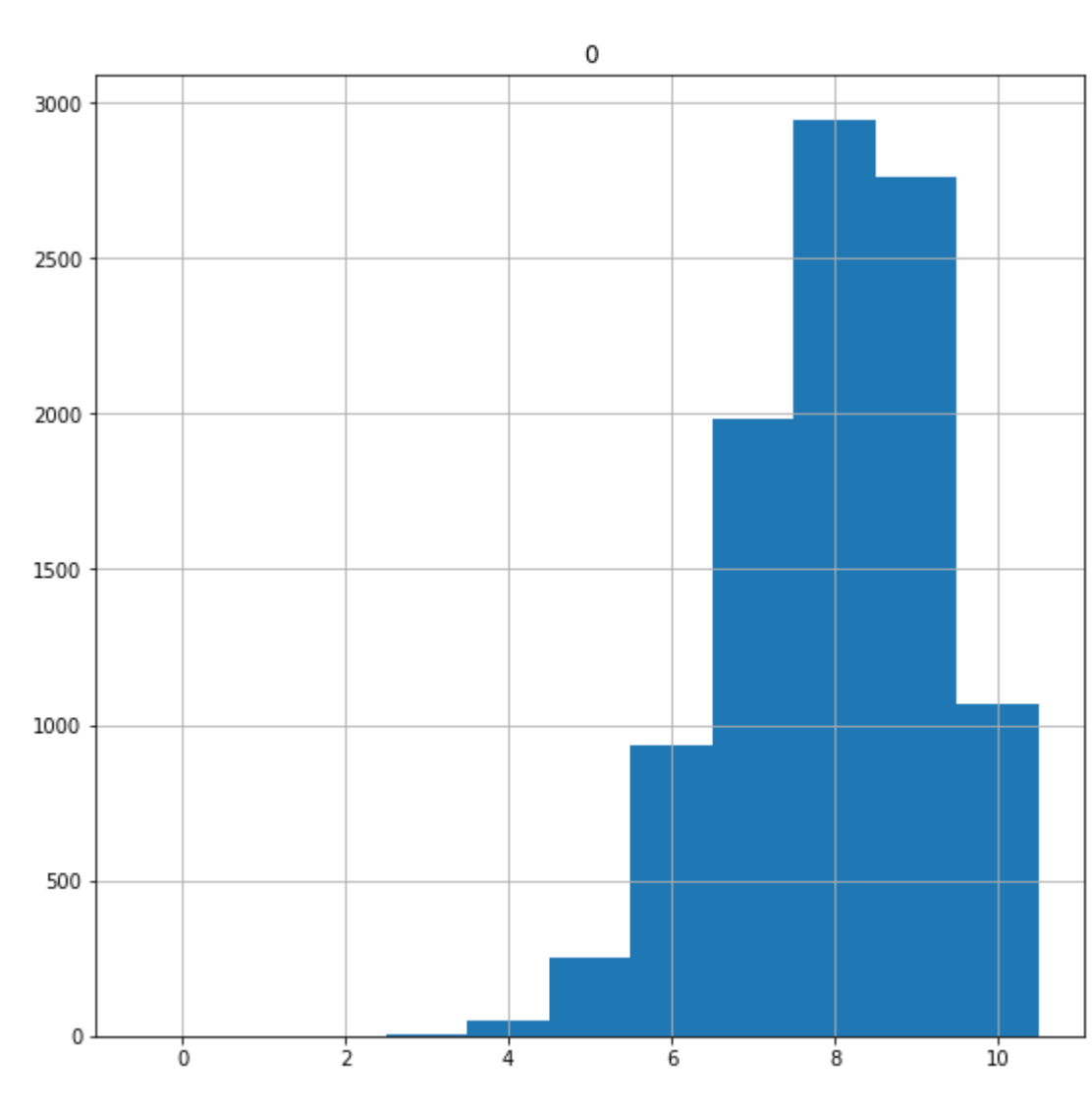
In [18]:
```
fair_coin_flips = stats.binom.rvs(n=10,        # Number of flips per trial
                                  p=0.5,       # Success probability
                                  size=10000)  # Number of trials

print( pd.crosstab(index="counts", columns= fair_coin_flips))

pd.DataFrame(fair_coin_flips).hist(range=(-0.5,10.5), bins=11);
```

```
col_0   0    1    2     3     4     5     6     7    8   9   10
row_0
counts  7  101  453  1190  2002  2430  2041  1197  479  94   6
```



In [19]:
```
biased_coin_flips = stats.binom.rvs(n=10,      # Number of flips per trial
                                    p=0.8,     # Success probability
                                    size=10000)  # Number of trials

# Print table of counts
print( pd.crosstab(index="counts", columns= biased_coin_flips))

# Plot histogram
pd.DataFrame(biased_coin_flips).hist(range=(-0.5,10.5), bins=11);
```

```
col_0   2  3   4    5    6     7     8     9    10
row_0
counts  1  8  50  253  934  1985  2945  2760  1064
```



In [20]:
```
stats.binom.cdf(k=5,        # Probability of k = 5 successes or less
                n=10,       # With 10 flips
                p=0.8)      # And success probability 0.8
```

Out[20]: 0.032793497599999964

In [21]:
```
1 - stats.binom.cdf(k=8,        # Probability of k = 9 successes or more
                    n=10,       # With 10 flips
                    p=0.8)      # And success probability 0.8
```

Out[21]: 0.37580963840000003

In [22]:
```
stats.binom.pmf(k=5,        # Probability of k = 5 successes
                n=10,       # With 10 flips
                p=0.5)      # And success probability 0.5
```

Out[22]: 0.24609375000000025

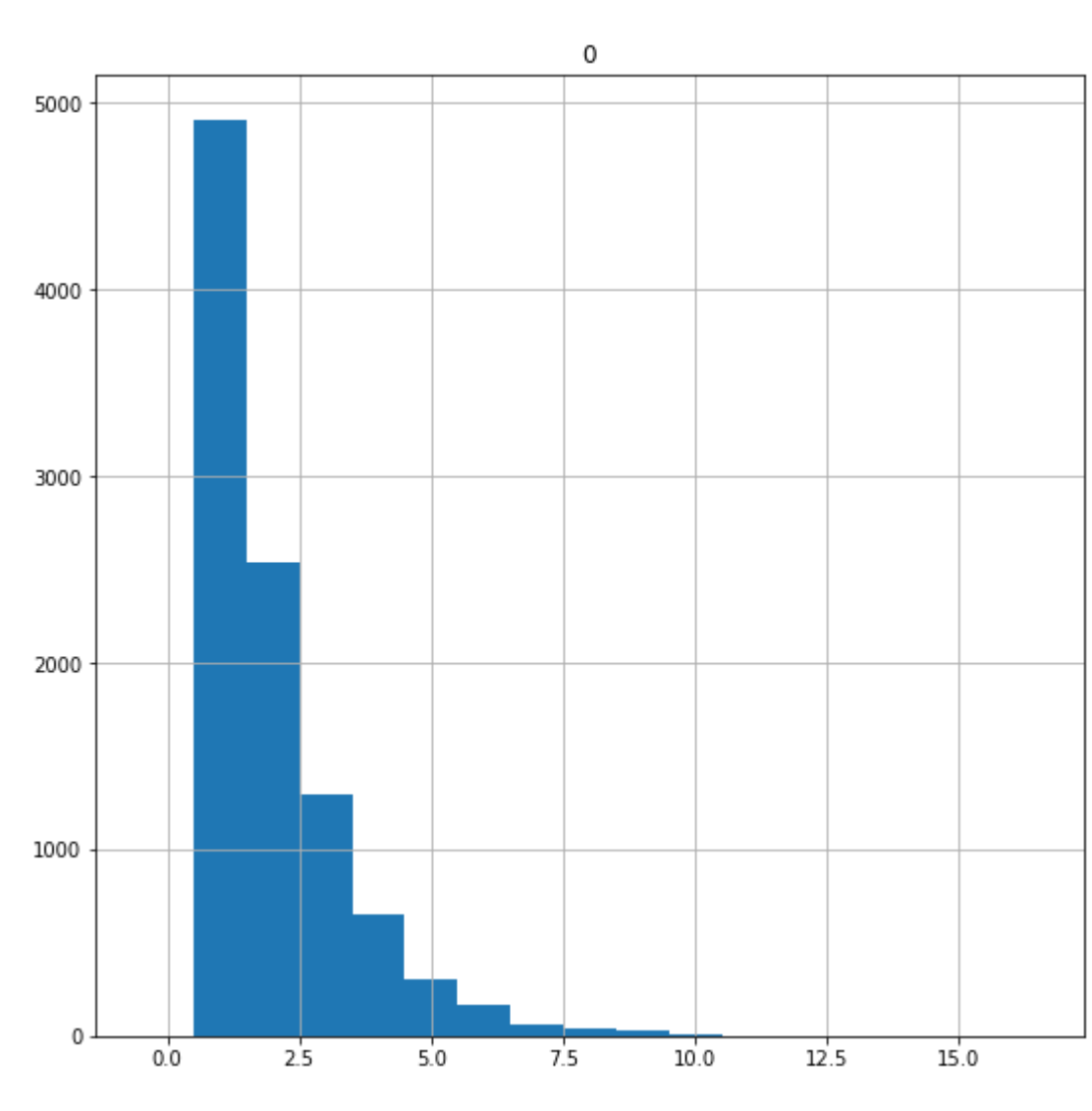
In [23]:
```
stats.binom.pmf(k=8,        # Probability of k = 8 successes
                n=10,       # With 10 flips
                p=0.8)      # And success probability 0.8
```

Out[23]: 0.301989888

Geometric and Exponential Distributions

In [24]:
```
random.seed(12)

flips_till_heads = stats.geom.rvs(size=10000,  # Generate geometric data
                                  p=0.5)       # With success prob 0.5

# Print table of counts
print( pd.crosstab(index="counts", columns= flips_till_heads))

# Plot histogram
pd.DataFrame(flips_till_heads).hist(range=(-0.5,max(flips_till_heads)+0.5)
                                    , bins=max(flips_till_heads)+1);
```

```
col_0      1     2     3    4    5    6   7   8   9  10  11  12  13  15  16
row_0
counts  4907  2541  1292  646  307  162  61  43  27   7   3   1   1   1   1
```



In [25]:
```
first_five = stats.geom.cdf(k=5,   # Prob of success in first 5 flips
                            p=0.5)

1 - first_five
```

Out[25]: 0.03125

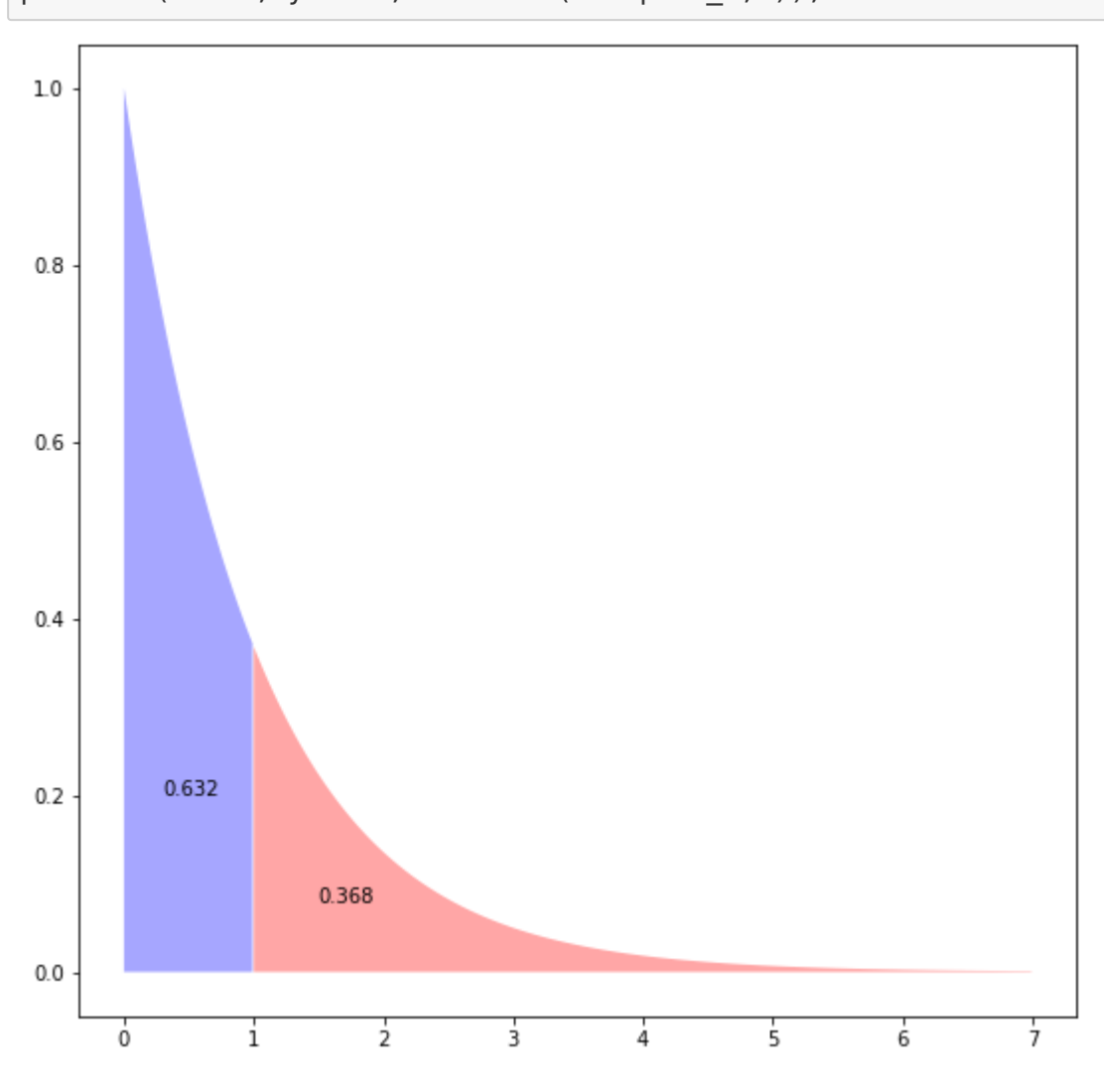
In [26]:
```
stats.geom.pmf(k=2,   # Prob of needing exactly 2 flips to get first success
               p=0.5)
```

Out[26]: 0.25

In [27]:
```
 # Get the probability of waiting more than 1 time unit before a success

prob_1 = stats.expon.cdf(x=1,
                         scale=1)  # Arrival rate

1 - prob_1
```

Out[27]: 0.36787944117144233

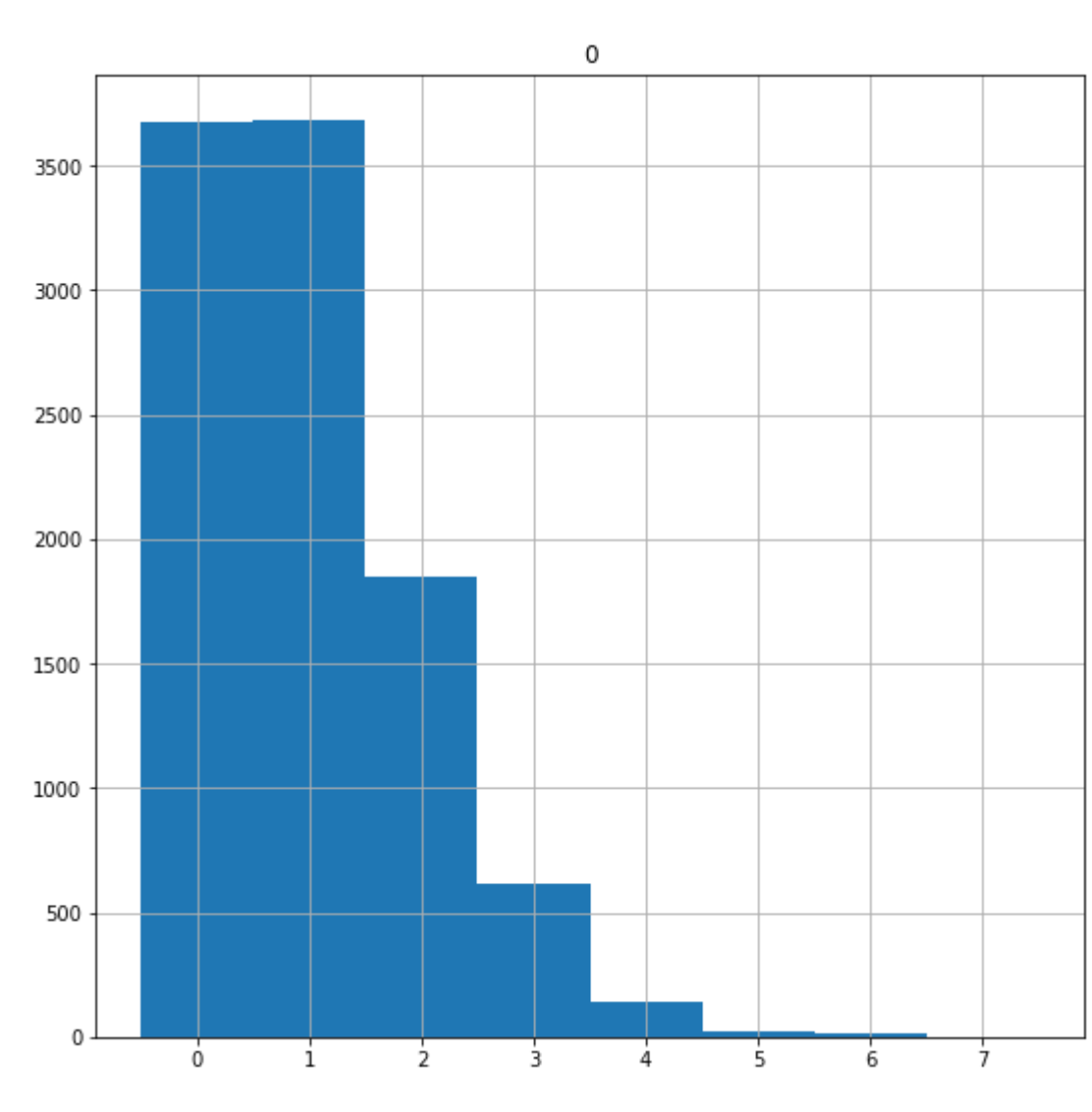
In [28]:
```
plt.fill_between(x=np.arange(0,1,0.01),
                 y1= stats.expon.pdf(np.arange(0,1,0.01)) ,
                 facecolor='blue',
                 alpha=0.35)

plt.fill_between(x=np.arange(1,7,0.01),
                 y1= stats.expon.pdf(np.arange(1,7,0.01)) ,
                 facecolor='red',
                 alpha=0.35)


plt.text(x=0.3, y=0.2, s= round(prob_1,3))
plt.text(x=1.5, y=0.08, s= round(1 - prob_1,3));
```



Poisson Distribution

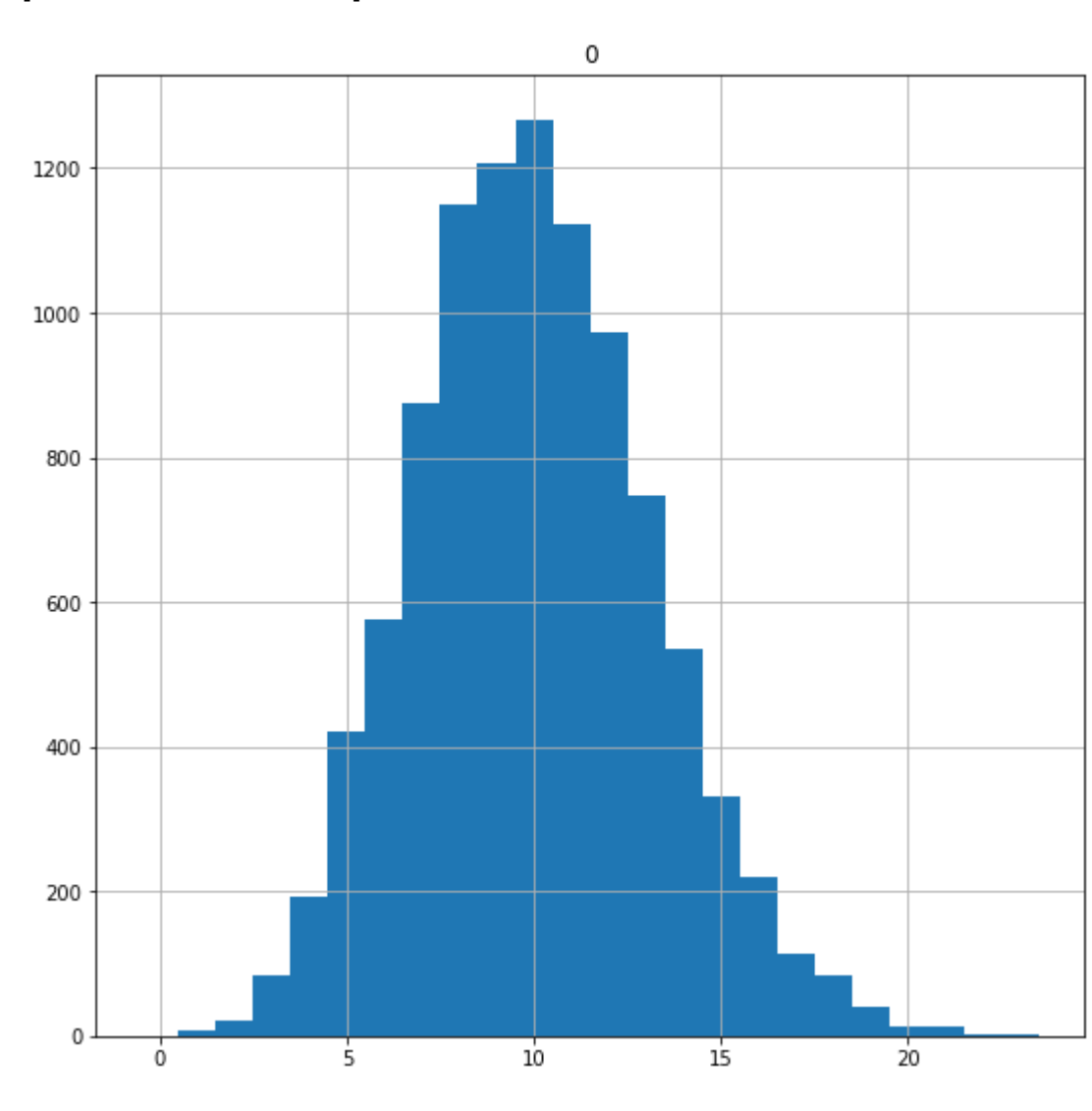
In [29]:
```
random.seed(12)

arrival_rate_1 = stats.poisson.rvs(size=10000,  # Generate Poisson data
                                   mu=1 )       # Average arrival time 1

# Print table of counts
print( pd.crosstab(index="counts", columns= arrival_rate_1))

# Plot histogram
pd.DataFrame(arrival_rate_1).hist(range=(-0.5,max(arrival_rate_1)+0.5)
                                  , bins=max(arrival_rate_1)+1);
```

```
col_0      0     1     2    3    4   5   6  7
row_0
counts  3673  3683  1846  618  142  26  11  1
```



In [30]:
```
random.seed(12)

arrival_rate_10 = stats.poisson.rvs(size=10000,  # Generate Poisson data
                                    mu=10 )      # Average arrival time 10

# Print table of counts
print( pd.crosstab(index="counts", columns= arrival_rate_10))

# Plot histogram
pd.DataFrame(arrival_rate_10).hist(range=(-0.5,max(arrival_rate_10)+0.5)
                                   , bins=max(arrival_rate_10)+1);
```

```
col_0   1   2   3    4    5    6    7     8     9    10  ...   14   15   16  \
row_0                                                   ...
counts  8  22  84  192  421  575  875  1150  1208  1267  ...  535  332  221

col_0    17  18  19  20  21  22  23
row_0
counts  114  83  40  14  12   2   2

[1 rows x 23 columns]
```



In [31]:
```
stats.poisson.cdf(k=5,     # Check the probability of 5 arrivals or less
                  mu=10)   # With arrival rate 10
```

Out[31]: 0.06708596287903189

In [32]:
```
stats.poisson.pmf(k=10,    # Check the prob f exactly 10 arrivals
                  mu=10)   # With arrival rate 10
```

Out[32]: 0.12511003572113372