Statistical Concepts

March 30, 2022

1 Basic Statistical Concepts

Prepared by: Himalaya Kakshapati

1.1 Mean

A mean is the simple mathematical average of a set of two or more numbers.

$$\mu(x) = \frac{1}{n}\sum_{i=1}^{n} x_i$$

[77]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

[78]:
```
rng = np.random.RandomState(0)
x = rng.randint(10, size=20)
x
```

[78]:
```
array([5, 0, 3, 3, 7, 9, 3, 5, 2, 4, 7, 6, 8, 8, 1, 6, 7, 7, 8, 1])
```

[79]:
```
n = len(x)
x_mean = sum(x)/n
x_mean
```

[79]:
```
5.0
```

[80]:
```
np.mean(x)
```

[80]:
```
5.0
```

1.2 Median

The median is the middle number in a sorted, ascending or descending, list of numbers and can be more descriptive of that data set than the average.

[81]:
```
# function to find median
def get_median(x):
```

```
    x_sorted = np.sort(x)

    mid_x_idx = len(x)//2 # index of middle value

    if len(x) % 2 == 0: # if even num of elements
        x_median = (x_sorted[mid_x_idx] + x_sorted[mid_x_idx - 1])/2
    else: # odd num of elements
        x_median = x_sorted[mid_x_idx]

    return x_median
```

[82]:
```
x_odd = rng.randint(10, size=11)
x_odd
```

[82]:
```
array([5, 9, 8, 9, 4, 3, 0, 3, 5, 0, 2])
```

[83]:
```
np.sort(x_odd)
```

[83]:
```
array([0, 0, 2, 3, 3, 4, 5, 5, 8, 9, 9])
```

[84]:
```
get_median(x_odd)
```

[84]:
```
4
```

[85]:
```
np.median(x_odd)
```

[85]:
```
4.0
```

[86]:
```
get_median(x)
```

[86]:
```
5.5
```

[87]:
```
np.median(x)
```

[87]:
```
5.5
```

1.3 Variance

Variance measures variability from the average or mean.

$$var(x) = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2$$

[88]:
```
x
```

[88]:
```
array([5, 0, 3, 3, 7, 9, 3, 5, 2, 4, 7, 6, 8, 8, 1, 6, 7, 7, 8, 1])
```

[89]:
```
def get_variance_population(x):
    n = len(x)
    x_mean = np.mean(x)
    big_sigma = 0

    for x_i in x:
        big_sigma += (x_i - x_mean)**2

    return big_sigma/n


def get_variance_sample(x):
    n = len(x)
    x_mean = np.mean(x)
    big_sigma = 0

    for x_i in x:
        big_sigma += (x_i - x_mean)**2

    return big_sigma/(n-1)
```

[90]:
```
get_variance_population(x)
```

[90]: 7.0

[91]:
```
np.var(x)
```

[91]: 7.0

1.4 Standard Deviation

Standard Deviation (σ) is the square root of variance.

$\sigma = \sqrt{var(x)}$

[92]:
```
x_std = np.sqrt(get_variance_population(x))
x_std
```

[92]: 2.6457513110645907

[93]:
```
np.std(x)
```

[93]: 2.6457513110645907

1.5 Covariance

Covariance is a measure of how much two random variables vary together. It's similar to variance - variance tells you how a single variable varies, covariance tells you how two variables vary together.

$$cov(x,y) = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})$$

[94]:
```
np.random.seed(1)
y = np.random.randint(10, size=20)
y
```

[94]:
```
array([5, 8, 9, 5, 0, 0, 1, 7, 6, 9, 2, 4, 5, 2, 4, 2, 4, 7, 7, 9])
```

[95]:
```
x
```

[95]:
```
array([5, 0, 3, 3, 7, 9, 3, 5, 2, 4, 7, 6, 8, 8, 1, 6, 7, 7, 8, 1])
```

[96]:
```
def get_covariance_population(x, y):
    n = len(x)
    x_mean = np.mean(x)
    y_mean = np.mean(y)
    big_sigma = 0

    for i in range(n):
        big_sigma += (x[i] - x_mean)*(y[i] - y_mean)

    return big_sigma/n


def get_covariance_sample(x, y):
    n = len(x)
    x_mean = np.mean(x)
    y_mean = np.mean(y)
    big_sigma = 0

    for i in range(len(x)):
        big_sigma += (x[i] - x_mean)*(y[i] - y_mean)

    return big_sigma/(n-1) # sample cov
```

[97]:
```
get_covariance_sample(x, y)
```

[97]:
```
-3.947368421052631
```

[98]:
```
get_variance_sample(x)
```

[98]:
```
7.368421052631579
```

[99]:
```
get_variance_sample(y)
```

[99]:
```
8.694736842105263
```

[100]:
```
np.cov(x, y)
```

[100]:
```
array([[ 7.36842105, -3.94736842],
       [-3.94736842,  8.69473684]])
```

1.6 Correlation

Correlation is a measure of the strength of a linear relationship between two quantitative variables.

- Correlation coefficients are used to measure the strength of the relationship between two variables.
- Pearson correlation is the one most commonly used in statistics. This measures the strength and direction of a linear relationship between two variables.
- Values always range between -1 (strong negative relationship) and +1 (strong positive relationship). Values at or close to zero imply a weak or no linear relationship.
- Correlation coefficient values less than +0.8 or greater than -0.8 are not considered significant.

$$cor(x, y) = \frac{cov(x, y)}{\sigma(x) * \sigma(y)}$$

[101]:
```
# cor_x_y = get_covariance(x, y)/np.sqrt(get_variance(x) * get_variance(y))
cor_x_y = get_covariance_population(x, y)/(np.std(x) * np.std(y))
cor_x_y
```

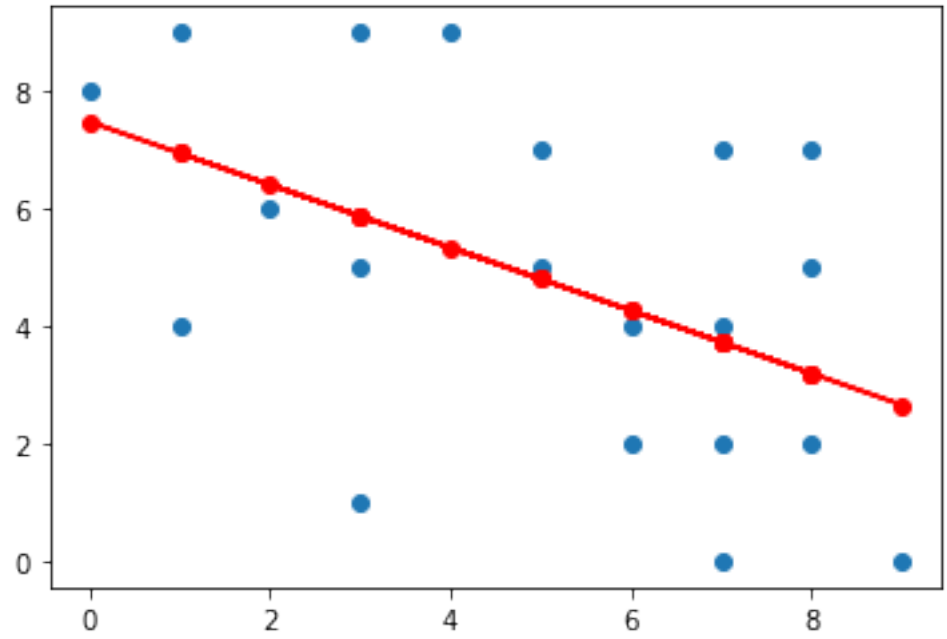
[101]:
```
-0.49316497387410163
```

[102]:
```
np.corrcoef(x, y)
```

[102]:
```
array([[ 1.        , -0.49316497],
       [-0.49316497,  1.        ]])
```

[103]:
```
plt.scatter(x, y)
z = np.polyfit(x, y, 1)
p = np.poly1d(z)
plt.plot(x, p(x), "r-o")
```

[103]:
```
[<matplotlib.lines.Line2D at 0x7fba02c69580>]
```



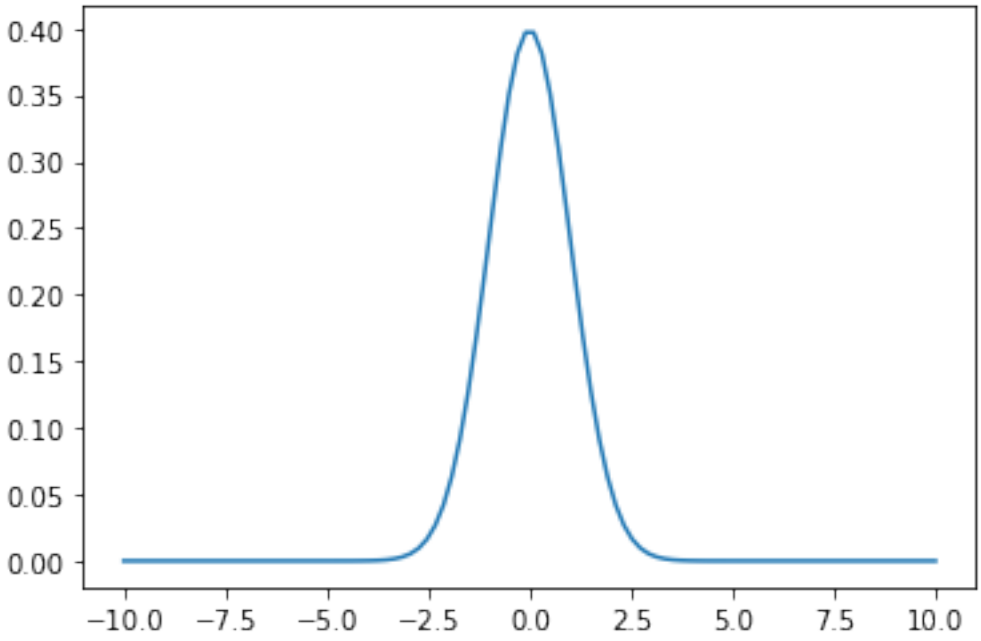
1.7 Normal Distribution

Normal distribution, also known as the Gaussian distribution, is a probability distribution that is symmetric about the mean, showing that data near the mean are more frequent in occurrence than data far from the mean. In graph form, normal distribution will appear as a bell curve.

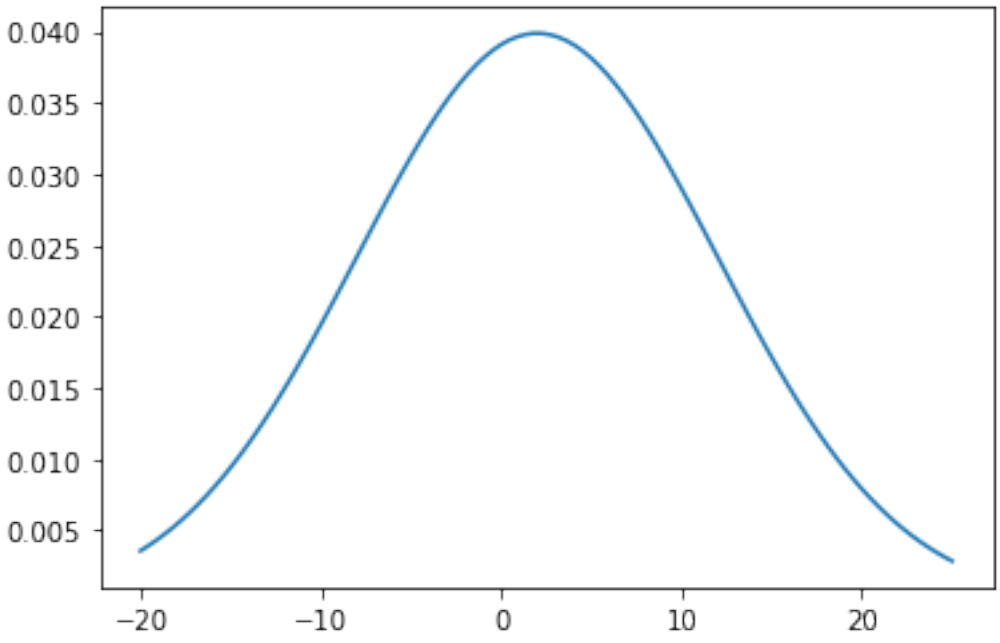
$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{1}{2}(\frac{x-\mu}{\sigma})^2}$$

[104]:
```
def plot_normal_curve(mu=0, sigma=1, x_left=-10, x_right=10):
    x = np.linspace(x_left, x_right, 100)

    a = 1/(sigma*np.sqrt(2*np.pi))
    b = -1/2*((x-mu)/sigma)**2
    f_x = a * np.exp(b)

    plt.plot(x,f_x)
```

[105]:
```
plot_normal_curve() # standard normal ==> mean = 0; std = 1
```



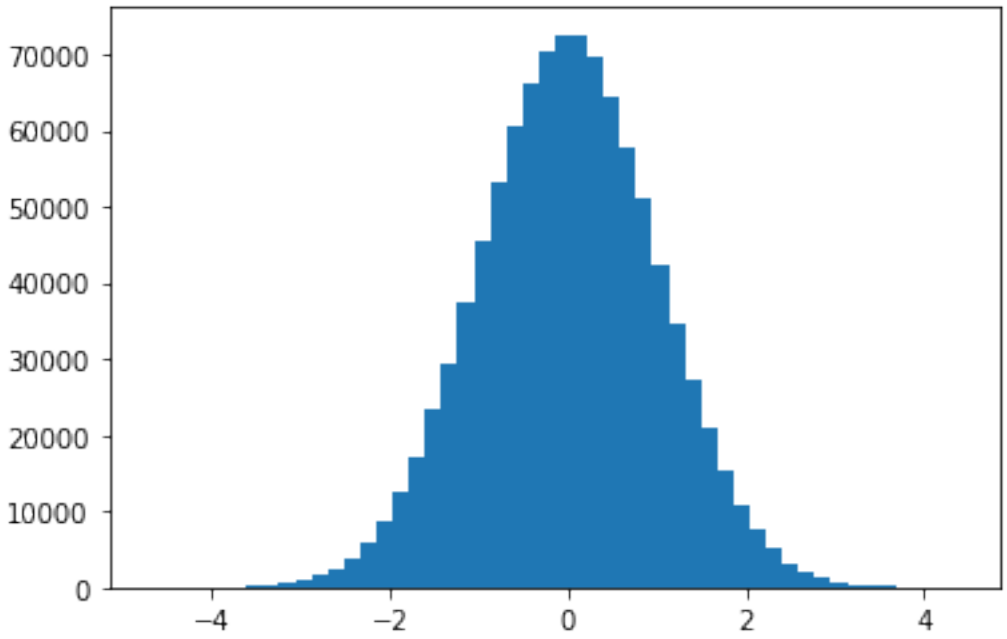
[106]:
```
plot_normal_curve(2, 10, -20, 25) # normal distribution curve with mean = 2 and␣
↪std = 10
```



[107]:
```
x_normal = np.random.normal(0, 1, 1000000)
x.shape
```

[107]:
```
(20,)
```

[108]:
```
plt.hist(x_normal, bins=50);
```



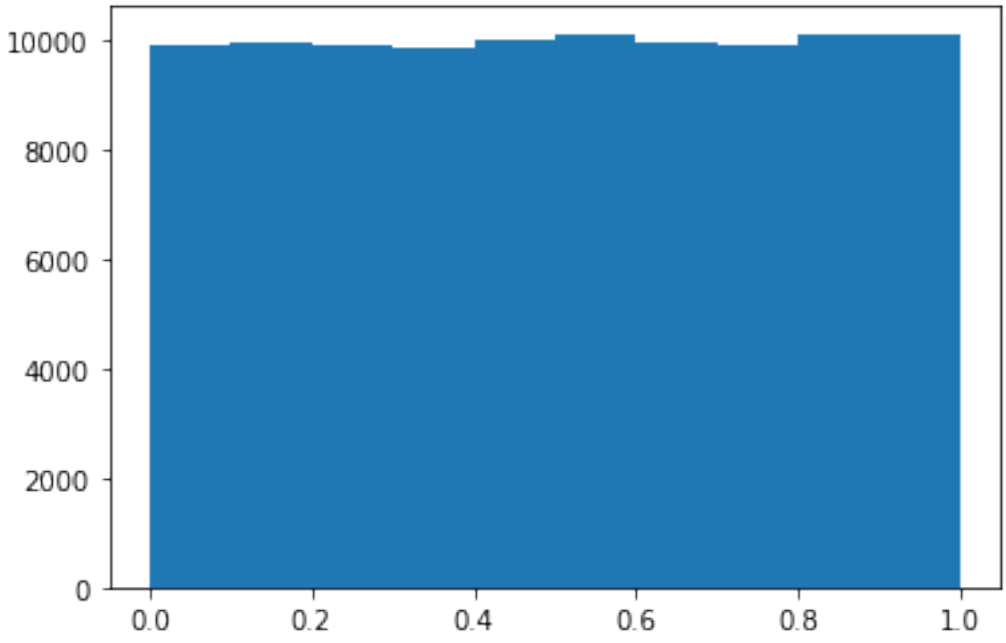
1.8 Uniform Distribution

Uniform distribution refers to a type of probability distribution in which all outcomes are equally likely.

[119]:
```
x_uniform = np.random.random(100000)
x_uniform.shape
```

[119]:
```
(100000,)
```

[120]:
```
plt.hist(x_uniform);
```


10000
8000
6000
4000
2000
0
0.0
0.2
0.4
0.6
0.8
1.0