MSPA PREDICT 400

Discussion Topic: Week 9 Normal Probability Distribution and the Curve

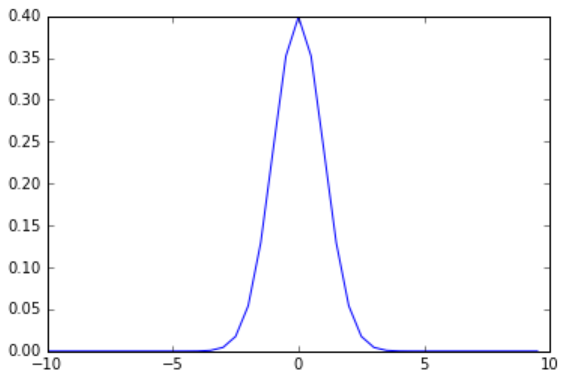
Introduction

This document presents the results of the ninth w eeks discussion topic for the Masters of Science in Predictive Analytics course: PREDICT 400. This assessment required the student to explain how the normal probability distribution related to the normal curve.

Normal Probability Distribution and the Curve

According to Lial, each normal probability distribution has associated w ith it a bell-shaped curve, called a normal curve. The below show s a normal probability distribution and its associated curve, across the range from -10 to 10.

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

points = np.arange(-10, 10, 0.5)
probabilities = norm.pdf(points)
plt.plot(points, probabilities)
plt.show()
```



Characteristics of the Normal Distribution

According to Interactive Mathematics, normal probability distributions have a number of properties: 1) The curve of the distribution is symmetrical about its mean value. 2) The mean is at the middle of the curve and divides its area into tw o halves. 3) The total area under the curve is equal to one. 4) It is completely determined by its mean and standard deviation values.

For all normal distributions the same percentage of observations are w ithin one standard deviation of the mean, w ithin tw o standard deviations of the mean, w ithin three standard deviations of the mean and so on. Likew ise, for any given normal distribution, approximately 68% of the data is w ithin one standard deviation of the mean, about 95% is w ithin tw o standard deviations of the mean, and about 99% is w ithin three standard deviations of the mean.

Formula demonstrating this for a set of normal data is show n below .

In [2]:
```
normdata = [36, 37, 38, 38, 39, 39, 40, 40, 40, 40, 41, 41, 41, 41, 41, 41, 42, 42, 42, 42, 42,
42, 42, 43, 43,
            43, 43, 43, 43, 43, 43, 44, 44, 44, 44, 44, 44, 44, 44, 44, 45, 45, 45, 45, 45, 45,
45, 45, 45, 45,
            46, 46, 46, 46, 46, 46, 46, 46, 46, 46, 47, 47, 47, 47, 47, 47, 47, 47, 47, 48, 48,
48, 48, 48, 48,
            48, 48, 49, 49, 49, 49, 49, 49, 49, 50, 50, 50, 50, 50, 50, 51, 51, 51, 51, 52, 52,
53, 53, 54, 55]

mean = float(sum(normdata)) / len(normdata)
print("mean:", mean)

variances = [(i - mean) ** 2 for i in normdata]
variance = float(sum(variances)) / len(variances)
print("variance", variance)

standard_deviation = variance ** (float(1)/2)
standard_deviations = [float(i - mean) / standard_deviation for i in normdata]
print("standard_deviation:", standard_deviation)
```

```
mean: 45.5
variance 15.21
standard_deviation: 3.9
```

In [3]:
```
def within_percentage(deviations, count):
    within = [i for i in deviations if i <= count and i >= -count]
    count = len(within)
    return float(count) / len(deviations)

within_one_percentage = within_percentage(standard_deviations, 1)
print("within_one_percentage:", within_one_percentage)

within_two_percentage = within_percentage(standard_deviations, 2)
print("within_two_percentage:", within_two_percentage)

within_three_percentage = within_percentage(standard_deviations, 3)
print("within_three_percentage:", within_three_percentage)
```

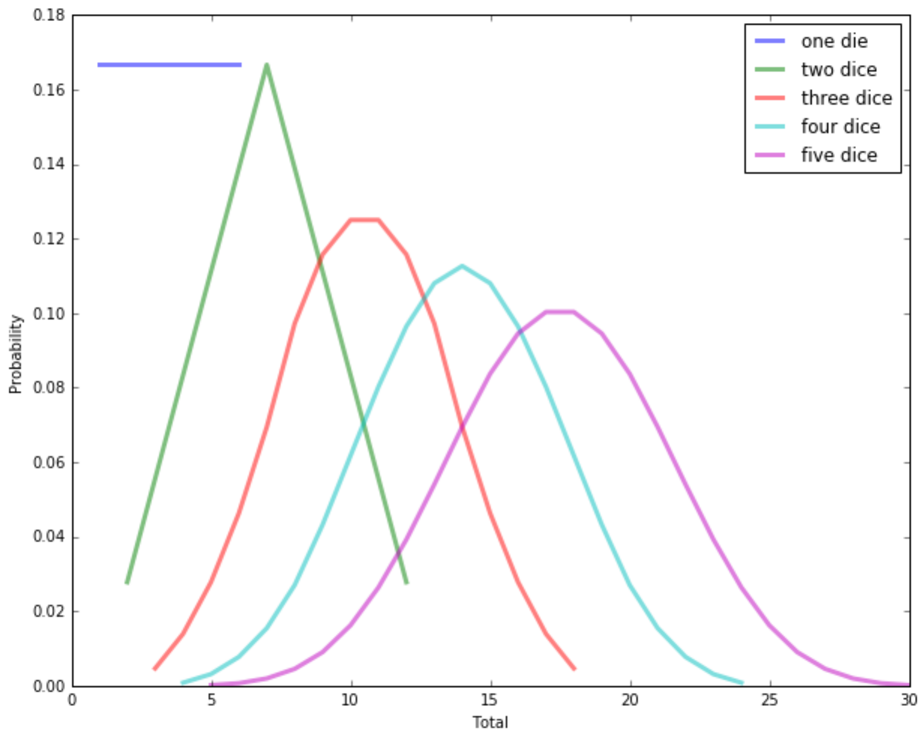
```
within_one_percentage: 0.68
within_two_percentage: 0.96
within_three_percentage: 1.0
```

Practical Application - Dice

We can use dice to demonstrate changes in probability distributions. Consider the example show n below . The plot presents probability outcomes for scenarios involving throw ing one, tw o, three, four and five dice. When one dice is throw n, its probability distribution remains uniform over the range of dice values. When throw ing tw o dice, the probability distribution becomes triangular w ith a mean of 6. As w e further increase the amount of dice throw n, the probability distributions become normal, represented by normal curves. The greater the amount of dice throw n, the greater the mean (or expected) value, and greater its variance (or range).

In [4]:
```
#Source: https://github.com/AllenDowney/PythonCounterPmf
from collections import Counter

class Pmf(Counter):
    def normalize(self):
        total = float(sum(self.values()))
        for key in self:
            self[key] /= total

    def __add__(self, other):
        pmf = Pmf()
        for key1, prob1 in self.items():
            for key2, prob2 in other.items():
                pmf[key1 + key2] += prob1 * prob2
        return pmf

    def __hash__(self):
        return id(self)

    def __eq__(self, other):
        return self is other

    def render(self):
        return zip(*sorted(self.items()))
```

In [5]:
```
import matplotlib.pyplot as plt
%matplotlib inline

d6 = Pmf([1,2,3,4,5,6])
pmf_ident = Pmf([0])

d6.normalize()
d6.name = 'one die'

d6_two = d6 + d6
d6_two.name = 'two dice'

d6_three = sum([d6]*3, pmf_ident)
d6_three.name = 'three dice'

d6_four = sum([d6]*4, pmf_ident)
d6_four.name = 'four dice'

d6_five = sum([d6]*5, pmf_ident)
d6_five.name = 'five dice'

plt.figure(figsize=(10,8))

for die in [d6, d6_two, d6_three, d6_four, d6_five]:
    xs, ys = die.render()
    plt.plot(xs, ys, label=die.name, linewidth=3, alpha=0.5)

plt.xlabel('Total')
plt.ylabel('Probability')
plt.legend()
plt.show()
```


Probability
Total
one die
two dice
three dice
four dice
five dice