深 圳 大 学 实 验 报 告

课程名称：概率论与数理统计

实验项目名称：**The central limit theorem**

学院：电子与信息工程学院

专业：电子信息工程

指导教师：陈昌盛老师

报告人：贾苏健 学号：**2022280485**

班级：**06**

实验时间：**2023.12.07**

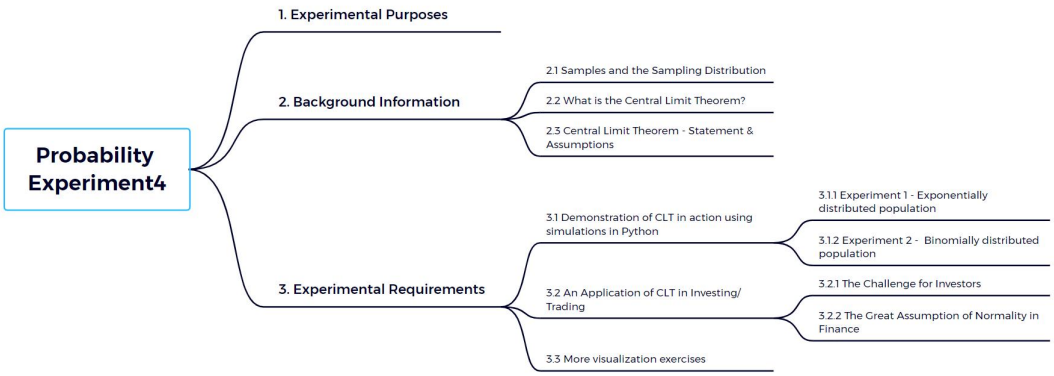
实验报告提交时间：**2023.12.14**

教务处制

I. Aim of Experiment:

- Familiar with the central limit theorem.
- Understand the implementation of the central limit theorem in python.
- Know how to visualize data in different distributions.

II. Experiment Content:



1. Demonstration of CLT in action using simulations in Python

In this section, the central limit theorem (CLT) is demonstrated through simulation using Python. The primary emphasis is that, for any population distribution, the distribution of the sample means tends to follow a normal distribution when the sample size is sufficiently large. Two experiments have been conducted to validate this point:

Experiment 1 - Exponentially distributed population

(1) Population Distribution:

1) The population follows an exponential distribution, commonly used to simulate the expected time one needs to wait for an event to occur.

2) The exponential distribution is defined by its probability density function:

$$f(x) = \frac{1}{\theta} e^{-\frac{x}{\theta}}, \ \ where\ x > 0$$

(2) Simulation Steps:

1) Parameter Calculation:

Set the rate parameter for the exponential distribution population θ to 4 .

Calculate the population mean μ and standard deviation σ based on the

rate parameter.

2) Small Sample Simulation $\mathrm{n} = 2$:

Draw 50 random samples, each of size 2, from the population.

Calculate the sample mean for each sample and plot the distribution of these sample means.

The results indicate that even for smaller sample sizes, such as 2, the distribution of sample means looks significantly different from the exponential population distribution. It appears more like a poor approximation to the normal distribution, showing some positive skewness.

3) Large Sample Simulation $\mathrm{n} = 500$:

Draw 50 random samples, each of size 500, from the population.

Calculate the sample mean for each sample and plot the distribution of these sample means.

The results show that as the sample size increases, the distribution of sample means becomes closer to a normal distribution.

(3) Validation:

Compare the mean and standard deviation of the 50 sample means from the Large Sample Simulation ($n = 500$) with the theoretical values predicted by the Central Limit Theorem (CLT).

Experiment 2 - Binomially distributed population

(1) Population Distribution:

The population follows a binomial distribution with parameters $k = 30$ and $p = 0.9$.

(2) Simulation Steps:

Large Sample Simulation $n = 500$:

Draw 50 random samples, each of size 500, from the binomial distribution population.

Calculate the sample mean for each sample and plot the distribution of these sample means.

The results show that similar to the exponential distribution, as the sample size increases, the distribution of sample means tends towards a normal distribution.

(3) Validation:

Compare the mean of the sample means with the population mean, and compare the standard deviation of the sample means with the values predicted by the Central Limit Theorem (CLT).

2. An Application of CLT in Investing/Trading

In this section, the application of the Central Limit Theorem (CLT) to an investment/trading scenario is explored:

(1) Scenario:

An investor (Jyoti) is contemplating investing in a stock (ITC) over the next five years.

The investor is risk-averse and aims to select a stock with an average monthly return that is predominantly non-negative.

Jyoti wants to employ a statistical framework to determine the average monthly return of the ITC stock with a certain level of confidence.

(2) Steps:

1) Theoretical Framework:

Discuss the assumption of normality in financial models, such as the Black-Scholes option pricing model.

Introduce CLT as a statistical framework based on return distribution assumptions.

2) Data Analysis:

Obtain and analyze the daily closing data of ITC stock for the past 10 years.

Visualize the distribution of daily logarithmic returns to assess the assumption of normality.

(3) Conclusion:

The visualization of daily logarithmic returns reveals deviations from normality, emphasizing the challenges of assuming normality in financial data.

3. More visualization exercises

In this section, additional visualization exercises were conducted using hourly weather data for the city of Detroit:

(1) Temperature Distribution:

A histogram of temperature data was plotted using 100 bins to visualize the distribution.

This distribution was compared to a normal distribution curve.

(2) Multi-Plot Visualization:

Variation in humidity, atmospheric pressure, and temperature in Detroit was

visualized separately in different plots.

A 7-day moving average line chart for temperature was created.

Other exercises were performed, including visualizing the temperature, humidity, and atmospheric pressure of different cities in the same plot and creating interactive checkboxes for city visibility.

(3) Weather Data Exploration:

Temperature, humidity, and atmospheric pressure variations were explored through visualization.

These exercises demonstrate how to use continuous distributions to approximate observed distributions of weather data and explore various aspects of temperature, humidity, and atmospheric pressure changes.

III. Experiment Process：

1. Demonstration of CLT in action using simulations in Python

In this section, the central limit theorem (CLT) is demonstrated through simulation using Python. The primary emphasis is that, for any population distribution, the distribution of the sample means tends to follow a normal distribution when the sample size is sufficiently large. Two experiments have been conducted to validate this point:

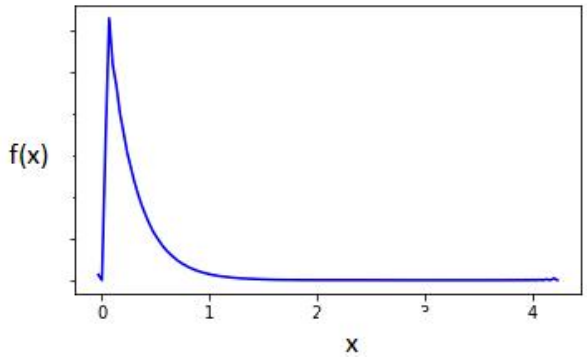
Experiment 1 - Exponentially distributed population

(1) Population Distribution:

1) The population follows an exponential distribution, commonly used to simulate the expected time one needs to wait for an event to occur.

2) The exponential distribution is defined by its probability density function:

$$f(x) = \begin{cases} \frac{1}{\theta} e^{-\frac{x}{\theta}} & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases}$$



(2) Simulation Steps:

1) Parameter Calculation:

Set the rate parameter for the exponential distribution population θ to 4 .

Calculate the population mean μ and standard deviation σ based on the rate parameter.

```
# rate parameter for the exponentially distributed population
theta = 4.0
# Population mean (mu), representing mean by parameter theta
mu = theta
# Population standard deviation (sd), representing sd by parameter theta
sd = theta
```

2) Small Sample Simulation $n = 2$:

Draw 50 random samples, each of size 2, from the population.

Calculate the sample mean for each sample and plot the distribution of these sample means.

The results indicate that even for smaller sample sizes, such as 2, the distribution of sample means looks significantly different from the exponential population distribution. It appears more like a poor approximation to the normal distribution, showing some positive skewness.

```
# drawing 50 random samples of size 2 from the exponentially distributed population
sample_size = 2
df2 = pd.DataFrame(index= ['x1', 'x2'])

for i in range(1, 51):
    exponential_sample = np.random.exponential(theta, sample_size)
    col = f'sample {i}'
    df2[col] = exponential_sample
```

```
# Taking a peek at the samples
df2
```

```
# Calculating sample means and plotting their distribution
df2_sample_means = df2.mean()
sns.distplot(df2_sample_means)
```

3) Large Sample Simulation $n = 500$:

Draw 50 random samples, each of size 500, from the population.

Calculate the sample mean for each sample and plot the distribution of these sample means.

The results show that as the sample size increases, the distribution of sample means becomes closer to a normal distribution.

```
sample_size = 500
sample_num = 50
df500 = pd.DataFrame(index=range(1, sample_size + 1))
for i in range(1, sample_num + 1):
    exponential_sample_large = np.random.exponential(theta, sample_size)
    col = f'sample {i}'
    df500[col] = exponential_sample_large
```

```
df500_sample_means = df500.mean()
sns.distplot(df500_sample_means)
```

(3) Validation:

Compare the mean and standard deviation of the 50 sample means from the Large Sample Simulation ($n = 500$) with the theoretical values predicted by the Central Limit Theorem (CLT).

```
# An estimate of the standard deviation of the sampling distribution can be obtained as:
sample_data_std_deviation  = df500_sample_means.std()
print("Estimate of standard deviation of the sampling distribution:",
sample_data_std_deviation )
```

```
# If you calculate correctly, the above value is very close to the value stated by the CLT, which
is:
sd/np.sqrt(sample_size)
```

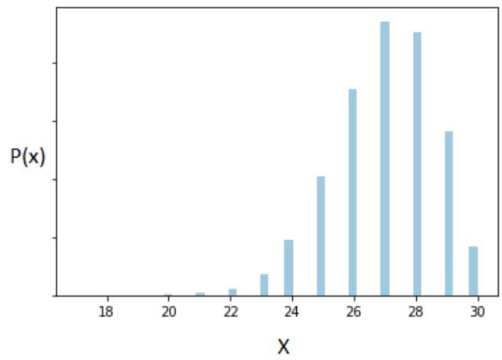
Experiment 2 - Binomially distributed population

(1) Population Distribution:

The population follows a binomial distribution with parameters $k = 30$ and $p = 0.9$.

$$P(x) = \begin{cases} \binom{k}{x} p^x (1-p)^{k-x} & \text{if } x = 0, 1, 2, \ldots, k \\ 0 & \text{otherwise} \end{cases}$$

where, $0 \leq p \leq 1$



(2) Simulation Steps:

Large Sample Simulation $n = 500$:

Draw 50 random samples, each of size 500, from the binomial distribution population.

Calculate the sample mean for each sample and plot the distribution of these sample means.

The results show that similar to the exponential distribution, as the sample size increases, the distribution of sample means tends towards a normal distribution.

```
# drawing 50 random samples of size 500 from a Binomial distribution with parameters k= 30 and
p=0.9
k = 30
p = 0.9
sample_size = 500
sample_num = 50

df500_binomial = pd.DataFrame(index=range(1, sample_size + 1))
```

```
for i in range(1, sample_num + 1):
    binomial_sample_large = np.random.binomial(k, p, sample_size)
    col = f'sample {i}'
    df500_binomial[col] = binomial_sample_large
```

```
df500_binomial_sample_means = df500_binomial.mean()
sns.distplot(df500_binomial_sample_means)
```

(3) Validation:

Compare the mean of the sample means with the population mean, and compare the standard deviation of the sample means with the values predicted by the Central Limit Theorem (CLT).

```
# Please compute the mean of sample means. If you calculate correctly, mean of sample means is
close to the population mean.
mean_of_sample_means = df500_binomial_sample_means.mean()
```

```
mean_of_sample_means_n = 0

for i in range(0, sample_num ):
    mean_of_sample_means_n += df500_binomial_sample_means[i]
mean_of_sample_means_n /= sample_num

print("Mean of sample means:", mean_of_sample_means)
print("Mean of sample means:", mean_of_sample_means_n)


# Please compute the standard deviation of sample means. If you calculate correctly, standard deviation of sample means is close to population standard deviation divided by square root of sample size.
std_dev_of_sample_means = df500_binomial_sample_means.std()
std_dev_of_sample_means_n = 0

for i in range(0, sample_num ):
    std_dev_of_sample_means_n += (df500_binomial_sample_means[i] - mean_of_sample_means_n) ** 2

std_dev_of_sample_means_n /= sample_num
std_dev_of_sample_means_n = np.sqrt(std_dev_of_sample_means_n)

print("Standard deviation of sample means:", std_dev_of_sample_means)
print("Standard deviation of sample means:", std_dev_of_sample_means_n)
```

2. An Application of CLT in Investing/Trading

In this section, the application of the Central Limit Theorem (CLT) to an investment/trading scenario is explored:

(1) Scenario:

An investor (Jyoti) is contemplating investing in a stock (ITC) over the next five years.

The investor is risk-averse and aims to select a stock with an average monthly return that is predominantly non-negative.

Jyoti wants to employ a statistical framework to determine the average monthly return of the ITC stock with a certain level of confidence.

(2) Steps:

1) Theoretical Framework:

Discuss the assumption of normality in financial models, such as the Black-Scholes option pricing model.

Introduce CLT as a statistical framework based on return distribution

assumptions.

```
# standard imports
import pandas as pd
import numpy as np
import pyfolio as pf
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"
pd.set_option('display.max_rows', None)

# Fetching the ITC stock data from yfinance library for the past 10 years
ITC = pd.read_csv('ITC.NS.csv')
daily_data = ITC.copy().round(4)
```

```
# Calculating daily Log returns
daily_data['daily_return']= np.log(daily_data['Adj Close']/daily_data['Adj Close'].shift())
daily_data.dropna(inplace=True)
```

```
# Taking a peek at the fetched data
daily_data.head()
```

2) Data Analysis:

Obtain and analyze the daily closing data of ITC stock for the past 10 years.

Visualize the distribution of daily logarithmic returns to assess the assumption of normality.

```
# Visualizing the daily log returns
sns.set(style="white", palette="muted", color_codes=True)
plt.figure(figsize=(12,5))
```

```
plt.subplot(1,2,1)
# Plot a simple histogram with binsize determined automatically.
# Please add the labels of the x, y axes and titles in the figure.
# Tips: sns.lineplot(daily_data.index,daily_data['daily_return'], color="r")
plt.plot(daily_data.index, daily_data['daily_return'], color="r")
plt.xlabel("Date")
plt.ylabel("Daily Log Returns")
plt.title("Daily Log Returns Over Time")
```

```
plt.subplot(1,2,2)
# Plot a simple histogram with binsize determined automatically.
# Please add the labels of the x, y axes and titles in the figure.
# Tips: sns.distplot(daily_data['daily_return'], kde=False, color="r")
sns.distplot(daily_data['daily_return'], kde=False, color="r")
```

```
plt.xlabel("Daily Log Returns")
plt.ylabel("Frequency")
plt.title("Distribution of Daily Log Returns")
plt.tight_layout()
plt.show()
```

(3) Conclusion:

The visualization of daily logarithmic returns reveals deviations from normality, emphasizing the challenges of assuming normality in financial data.

3. More visualization exercises

In this section, additional visualization exercises were conducted using hourly weather data for the city of Detroit:

(1) Temperature Distribution:

A histogram of temperature data was plotted using 100 bins to visualize the distribution.

This distribution was compared to a normal distribution curve.

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from scipy.stats import norm
```

```
temperature_data = pd.read_csv('temperature.csv')
detroit_temperature = temperature_data['Detroit']
plt.figure(figsize=(12, 12))
sns.histplot(detroit_temperature, bins=100, kde=False, color="blue", stat="density")
xmin, xmax = plt.xlim()
x = np.linspace(xmin, xmax, 100)
p = norm.pdf(x, detroit_temperature.mean(), detroit_temperature.std())
```

```
plt.plot(x, p, 'k', linewidth=2)
plt.xlabel("Temperature (C)")
plt.ylabel("Density")
plt.title("Distribution of Temperature in Detroit")
plt.show()
```

(2) Multi-Plot Visualization:

Variation in humidity, atmospheric pressure, and temperature in Detroit was visualized separately in different plots.

A 7-day moving average line chart for temperature was created.

Other exercises were performed, including visualizing the temperature, humidity, and atmospheric pressure of different cities in the same plot and creating interactive checkboxes for city visibility.

```
humidity_data = pd.read_csv('humidity.csv')
pressure_data = pd.read_csv('pressure.csv')
```

```
merged_data = pd.DataFrame({
    'Date': temperature_data['datetime'],
    'Temperature': temperature_data['Detroit'],
    'Humidity': humidity_data['Detroit'],
    'Pressure': pressure_data['Detroit']
})

merged_data['Date'] = pd.to_datetime(merged_data['Date'])
merged_data.set_index('Date', inplace=True)
resampled_data = merged_data.resample('D').mean()
window_size = 7
smoothed_data = resampled_data.rolling(window=window_size).mean()
```

```
plt.figure(figsize=(12, 12))
# 湿度
plt.subplot(3, 1, 1)
plt.plot(smoothed_data.index, smoothed_data['Humidity'], color='b')
plt.xlabel("Date")
plt.ylabel("Humidity")
plt.title("Smoothed Humidity Variation in Detroit")
```

```
# 气压
plt.subplot(3, 1, 2)
plt.plot(smoothed_data.index, smoothed_data['Pressure'], color='r')
plt.xlabel("Date")
plt.ylabel("Pressure")
plt.title("Smoothed Pressure Variation in Detroit")
```

```
# 温度
plt.subplot(3, 1, 3)
plt.plot(smoothed_data.index, smoothed_data['Temperature'], color='g')
plt.xlabel("Date")
plt.ylabel("Temperature (C)")
plt.title("Smoothed Temperature Variation in Detroit")
```

```
plt.tight_layout()
plt.show()
```

(3) Weather Data Exploration:

Temperature, humidity, and atmospheric pressure variations were explored through visualization.

These exercises demonstrate how to use continuous distributions to approximate observed distributions of weather data and explore various aspects of temperature, humidity, and atmospheric pressure changes.

```
import pandas as pd
import matplotlib.pyplot as plt
import ipywidgets as widgets
from IPython.display import display

temperature_data = pd.read_csv('temperature.csv')
temperature_data['datetime'] = pd.to_datetime(temperature_data['datetime'])
temperature_data.set_index('datetime', inplace=True)
resampled_data = temperature_data.resample('D').mean()

window_size = 7
smoothed_data = resampled_data.rolling(window=window_size).mean()

plt.figure(figsize=(12, 8))

initial_cities = ['Vancouver', 'Portland', 'San Francisco', 'Seattle', 'Los Angeles']
lines = {city: {'line': None, 'visible': True} for city in initial_cities}
for city in initial_cities:
    lines[city]['line'], = plt.plot(smoothed_data.index, smoothed_data[city], label=city)

plt.xlabel("Date")
plt.ylabel("7-Day Moving Average Temperature")
plt.title("7-Day Moving Average Temperature for Different Cities")
plt.legend()

checkbox_options = {city: widgets.Checkbox(description=city, value=True) for city in
initial_cities}
checkboxes = [checkbox_options[city] for city in initial_cities]
checkbox_container = widgets.HBox(checkboxes)
display(checkbox_container)

def update_visibility(change):
    for city, checkbox in checkbox_options.items():
        lines[city]['visible'] = checkbox.value
```

```
        lines[city]['line'].set_visible(lines[city]['visible'])
        print(f"City: {city}, Visible: {lines[city]['visible']}")
    plt.draw()

for checkbox in checkboxes:
    checkbox.observe(update_visibility, names='value')

plt.show()
```

IV. Data Logging and Processing:

1. Demonstration of CLT in action using simulations in Python

Experiment 1 - Exponentially distributed population

Data Acquisition: Obtain exponential random distribution using the **np.random.exponential** function.

```
for i in range(1, 51):
    exponential_sample = np.random.exponential(theta, sample_size)
    col = f'sample {i}'
    df2[col] = exponential_sample
```

Experiment_draft.ipynb ● Experiment 4.ipynb ● 数据查看器 - df2 ×

c:\Users\HP\Python_project\Probability\Experiment\Experiment4\Experiment 4.ipynb > df2 (2, 50)

	index	sample 1	sample 2	sample 3	sample 4	sample 5	sample 6	sample 7	sample 8
0	x1	2.06109706	1.1515766587	7.4920320238	1.4346119007	11.420443...	3.7478933...	7.0048711394	0.84110005...
1	x2	10.188970...	7.8778722147	3.176720814	0.0842326477	12.484997...	0.9857038...	2.0898850697	3.61624496...

Experiment 2 - Binomially distributed population

Data Acquisition: Obtain exponential random distribution using the **np.random.exponential** function.

```
sample_size = 500
sample_num = 50
df500 = pd.DataFrame(index=range(1, sample_size + 1))

for i in range(1, sample_num + 1):
    exponential_sample_large = np.random.exponential(theta, sample_size)
```

```
    col = f'sample {i}'
    df500[col] = exponential_sample_large
```

	sample 1	sample 2	sample 3	sample 4	sample 5	sample 6	sample 7	sample 8	sample 9	sample 10	...
1	0.129926	1.973918	0.513561	0.491963	4.035040	3.104559	1.691499	0.148011	0.367192	0.929294	...
2	5.404039	0.819062	6.843216	2.389920	0.924985	0.019902	22.989047	5.747818	1.349662	2.778028	...
3	1.285877	8.871819	0.052478	0.135484	0.007493	9.005566	0.996265	4.455125	0.174381	1.586709	...
4	1.869372	2.178113	0.018058	1.139256	5.844479	3.769432	4.460035	3.120554	0.537432	3.722418	...
5	8.607907	1.513730	0.901487	0.007510	4.212222	0.092138	0.169779	1.259047	5.365109	3.630046	...
...	...	...	...	...	...	...	...	...	...	...	...
496	9.399404	1.237989	2.629913	2.848426	0.411869	0.361067	8.857628	1.204304	2.006676	3.275098	...
497	2.390042	6.389668	7.114527	9.118013	3.646025	5.010857	1.540356	12.956246	7.337386	9.144083	...
498	0.906639	5.412618	0.524920	16.754684	1.960354	2.640571	3.338145	3.447527	12.561068	6.511013	...
499	14.804343	7.439908	6.657782	6.677561	10.656890	11.628107	1.284226	1.033237	1.690431	0.669259	...
500	3.158472	2.837769	0.358805	9.593125	20.393480	2.402541	2.748983	1.615262	14.381485	2.250954	...

500 rows × 50 columns

2. An Application of CLT in Investing/Trading

Data Acquisition: Fetched ITC stock data from the yfinance library for the past 10 years and saved it into a DataFrame named ITC.

```
# Fetching the ITC stock data from yfinance library for the past 10 years
ITC = pd.read_csv('ITC.NS.csv')
```

Data Processing: Firstly, a new DataFrame named daily_data was created by copying the ITC DataFrame and rounding all numerical values to four decimal places. Subsequently, daily logarithmic returns were calculated by taking the natural logarithm of the ratio of the closing price for the current day to that of the previous day. Finally, rows containing missing values were removed to ensure data integrity.

```
daily_data = ITC.copy().round(4)

# Calculating daily Log returns
daily_data['daily_return']= np.log(daily_data['Adj Close']/daily_data['Adj Close'].shift())
daily_data.dropna(inplace=True)
```

3. More visualization exercises

(1) Temperature Distribution:

```
 # 读取小时天气数据集
temperature_data = pd.read_csv('temperature.csv')
# 提取底特律市的温度记录
detroit_temperature = temperature_data['Detroit']
```

(2) Multi-Plot Visualization:

Data Acquisition:

```
# 读取湿度和气压数据
humidity_data = pd.read_csv('humidity.csv')
pressure_data = pd.read_csv('pressure.csv')
```

Data Processing: The smoothing process is carried out using a method called moving average. Specifically, this involves creating a rolling window that includes a certain number of data points (with a window size specified as **window_size**, set to 7 days in this case), and then calculating the average of the data within each window. Consequently, the resulting new sequence, **smoothed_data**, comprises the moving average values of the original data.

```
# 使用移动平均来平滑曲线
window_size = 7  # 7 天的移动平均窗口
smoothed_data = resampled_data.rolling(window=window_size).mean()
```

(3) Weather Data Exploration:

```
# 读取小时天气数据集
temperature_data = pd.read_csv('temperature.csv')

# 将日期列设置为索引
temperature_data['datetime'] = pd.to_datetime(temperature_data['datetime'])
temperature_data.set_index('datetime', inplace=True)

# 降采样数据，每天一个数据点，使用每天的平均值
resampled_data = temperature_data.resample('D').mean()

# 使用移动平均来平滑曲线
window_size = 7  # 7 天的移动平均窗口
smoothed_data = resampled_data.rolling(window=window_size).mean()
```

V. Experimental Results and Analysis:

1. Demonstration of CLT in action using simulations in Python

Experiment 1 - Exponentially distributed population

1) Parameter Calculation:

Set the rate parameter for the exponential distribution population θ to 4 .

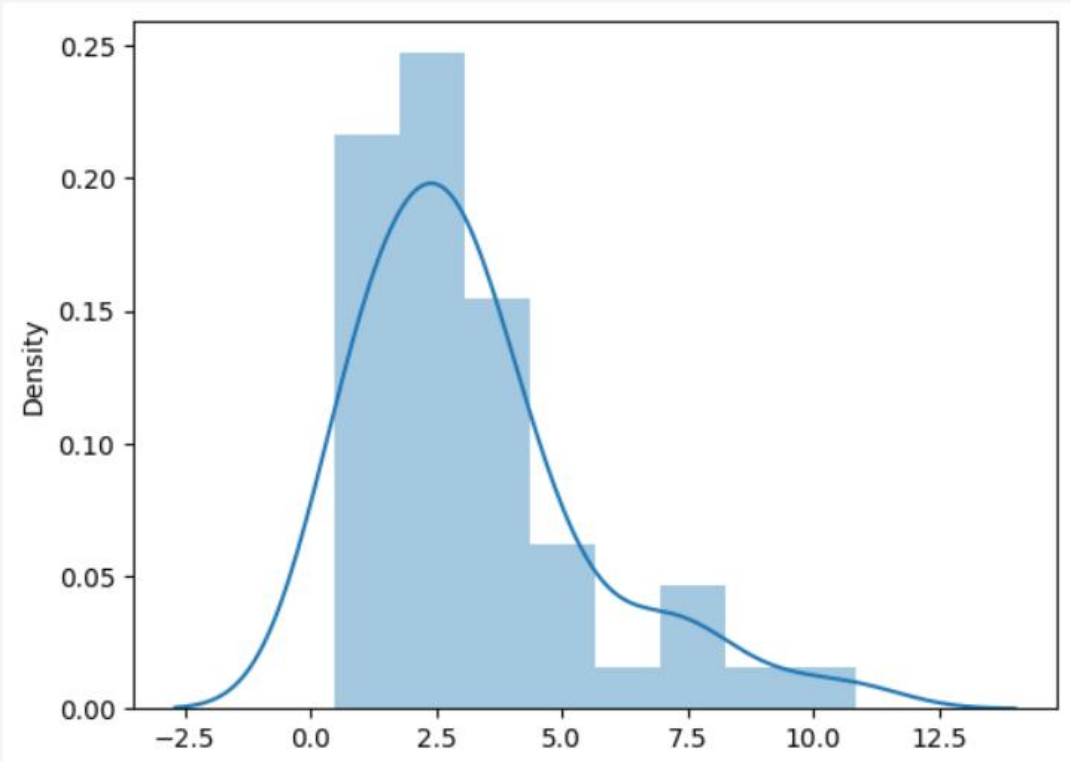
Calculate the population mean μ and standard deviation σ based on the rate parameter.

2) Small Sample Simulation $n = 2$:

Draw 50 random samples, each of size 2, from the population.

Calculate the sample mean for each sample and plot the distribution of these sample means.

The results indicate that even for smaller sample sizes, such as 2, the distribution of sample means looks significantly different from the exponential population distribution. It appears more like a poor approximation to the normal distribution, showing some positive skewness.

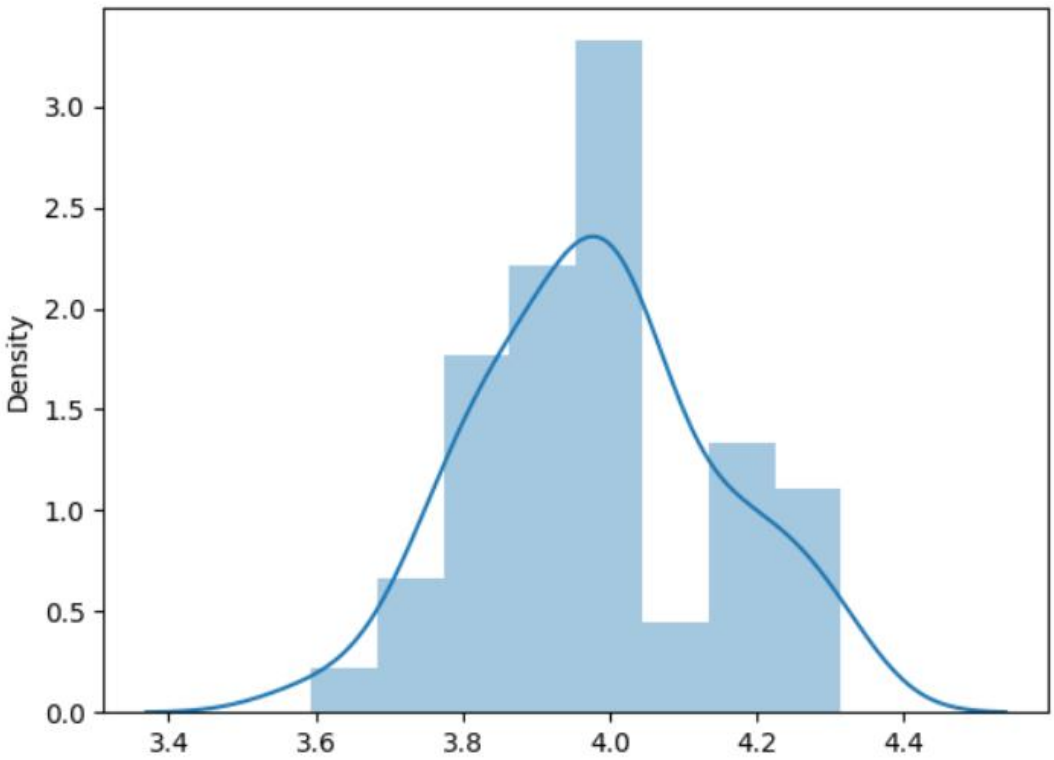


3) Large Sample Simulation $n = 500$:

Draw 50 random samples, each of size 500, from the population.

Calculate the sample mean for each sample and plot the distribution of these sample means.

The results show that as the sample size increases, the distribution of sample means becomes closer to a normal distribution.




```
#·An·estimate·of·the·standard·deviation·of·the·sampling·distribution·can·be·obtained·as:
sample_data_std_deviation·=·df500_sample_means.std()
print("Estimate·of·standard·deviation·of·the·sampling·distribution:",·sample_data_std_deviation·)
```

✓ 0.0s

```
Estimate of standard deviation of the sampling distribution: 0.19079234611729667
```

```
# If you calculate correctly, the above value is very close to the value stated by the CLT, which is:
sd/np.sqrt(sample_size)
```

✓ 0.0s

```
0.17888543819998318
```

Validation:

Compare the mean and standard deviation of the 50 sample means from the Large Sample Simulation ($n = 500$) with the theoretical values predicted by the Central Limit Theorem (CLT).

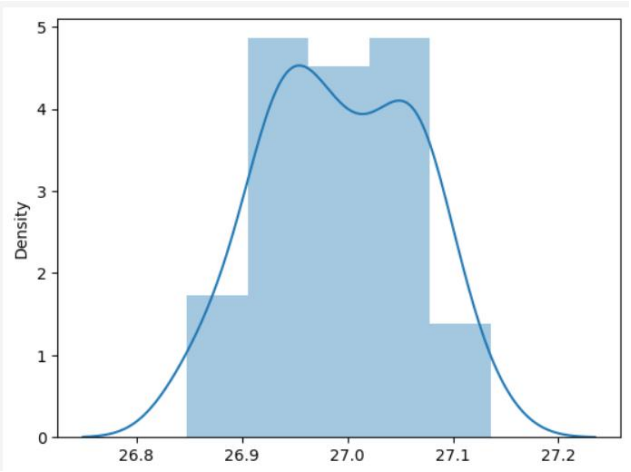
Experiment 2 - Binomially distributed population

Large Sample Simulation $n = 500$:

Draw 50 random samples, each of size 500, from the binomial distribution population.

Calculate the sample mean for each sample and plot the distribution of these sample means.

The results show that similar to the exponential distribution, as the sample size increases, the distribution of sample means tends towards a normal distribution.




```
mean_of_sample_means = df500_binomial_sample_means.mean()

mean_of_sample_means_n = 0

for i in range(0, sample_num ):
    mean_of_sample_means_n += df500_binomial_sample_means[i]

mean_of_sample_means_n /= sample_num

print("Mean of sample means:", mean_of_sample_means)
print("Mean of sample means:", mean_of_sample_means_n)
```

✓ 0.0s

```
Mean of sample means: 27.006
Mean of sample means: 27.006000000000007
```

```
std_dev_of_sample_means = df500_binomial_sample_means.std()

std_dev_of_sample_means_n = 0

for i in range(0, sample_num ):
    std_dev_of_sample_means_n += (df500_binomial_sample_means[i] - mean_of_sample_means_n) ** 2

std_dev_of_sample_means_n /= sample_num
std_dev_of_sample_means_n = np.sqrt(std_dev_of_sample_means_n)

print("Standard deviation of sample means:", std_dev_of_sample_means)
print("Standard deviation of sample means:", std_dev_of_sample_means_n)
```

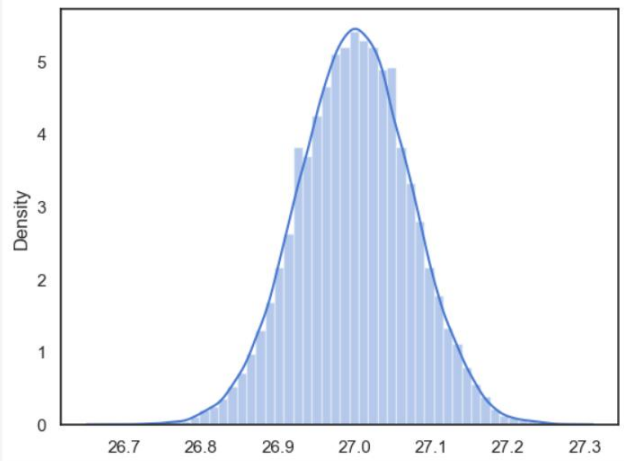
✓ 0.0s

```
Standard deviation of sample means: 0.0798314551069301
Standard deviation of sample means: 0.0790291085613396
```

$When\ n\ = 50000,$

```
500 rows × 50000 columns

<Axes: ylabel='Density'>
```




```python
mean_of_sample_means = df500_binomial_sample_means.mean()

mean_of_sample_means_n = 0

for i in range(0, sample_num ):         # 500个样本的均值，这里从0开始
    mean_of_sample_means_n += df500_binomial_sample_means[i]

mean_of_sample_means_n /= sample_num

print("Mean of sample means:", mean_of_sample_means)
print("Mean of sample means:", mean_of_sample_means_n)
#
```

✓ 0.3s

```
Mean of sample means: 26.99994316
Mean of sample means: 26.999943159999997
```

```python
std_dev_of_sample_means = df500_binomial_sample_means.std()

std_dev_of_sample_means_n = 0

for i in range(0, sample_num ):         # 500个样本的均值，这里从0开始
    std_dev_of_sample_means_n += (df500_binomial_sample_means[i] - mean_of_sample_means_n) ** 2

std_dev_of_sample_means_n /= sample_num
std_dev_of_sample_means_n = np.sqrt(std_dev_of_sample_means_n)

print("Standard deviation of sample means:", std_dev_of_sample_means)
print("Standard deviation of sample means:", std_dev_of_sample_means_n)

#
```

✓ 0.4s

```
Standard deviation of sample means: 0.07320140994641264
Standard deviation of sample means: 0.07320067792865023
```

Validation:

Compare the mean of the sample means with the population mean, and compare the standard deviation of the sample means with the values predicted by the Central Limit Theorem (CLT).

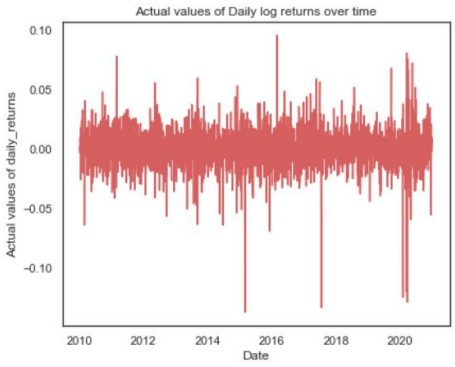
The size of the sample, represented by **sample_size** here, is chosen based on requirements. Initially, following Experiment 1, I selected n = 500. However, the calculated mean and variance deviated significantly from the theoretical values. Therefore, in the second attempt, I adjusted the sample size to n = 50000. Although the computation was notably slower, the results were more in line with the theoretical values.

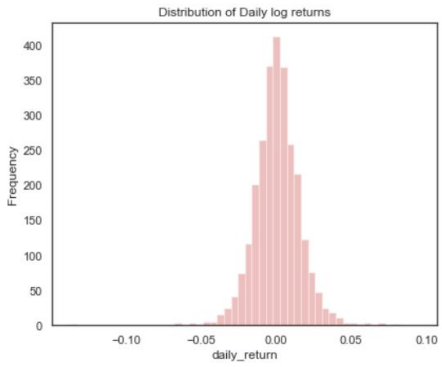
2. An Application of CLT in Investing/Trading

Data Analysis:

Obtain and analyze the daily closing data of ITC stock for the past 10 years.

Visualize the distribution of daily logarithmic returns to assess the assumption of normality.





Conclusion:

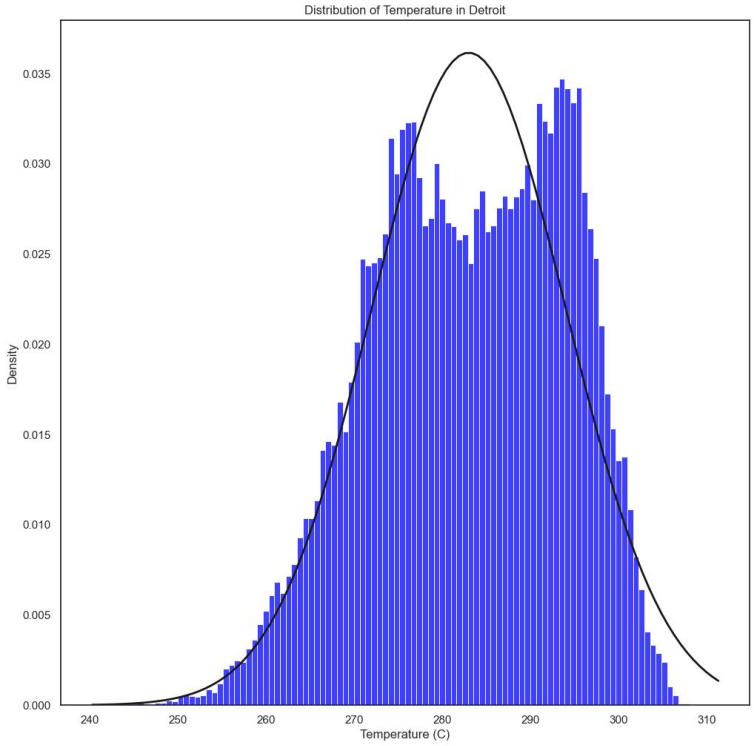
The visualization of daily logarithmic returns reveals deviations from normality, emphasizing the challenges of assuming normality in financial data.

3. More visualization exercises

(1) Temperature Distribution:

A histogram of temperature data was plotted using 100 bins to visualize the distribution.

This distribution was compared to a normal distribution curve.

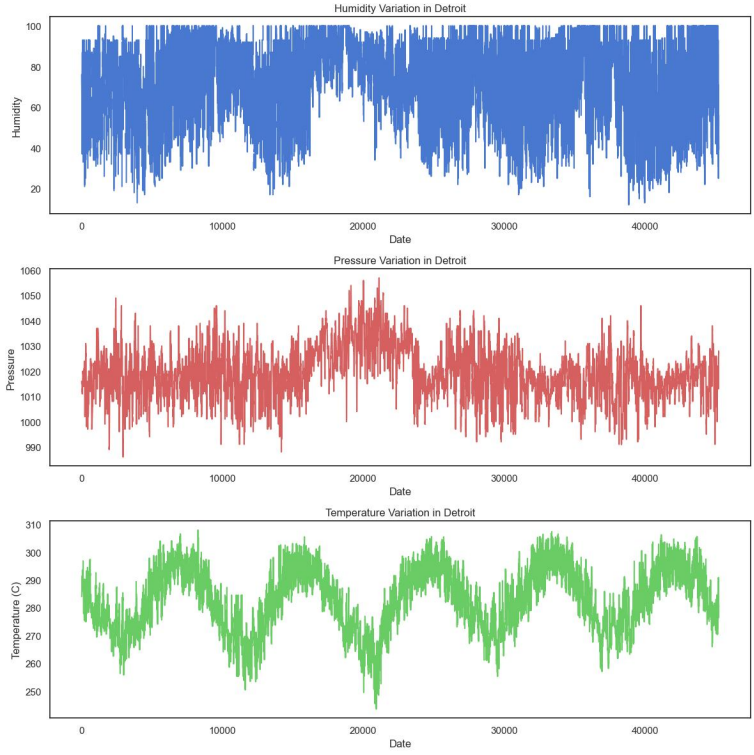


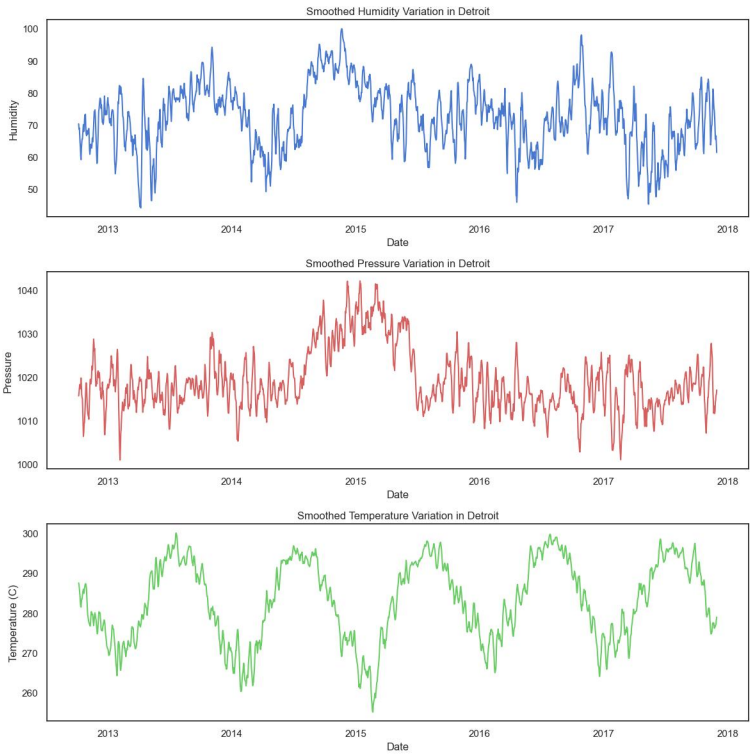
(2) Multi-Plot Visualization:

Variation in humidity, atmospheric pressure, and temperature in Detroit was visualized separately in different plots.

A 7-day moving average line chart for temperature was created.

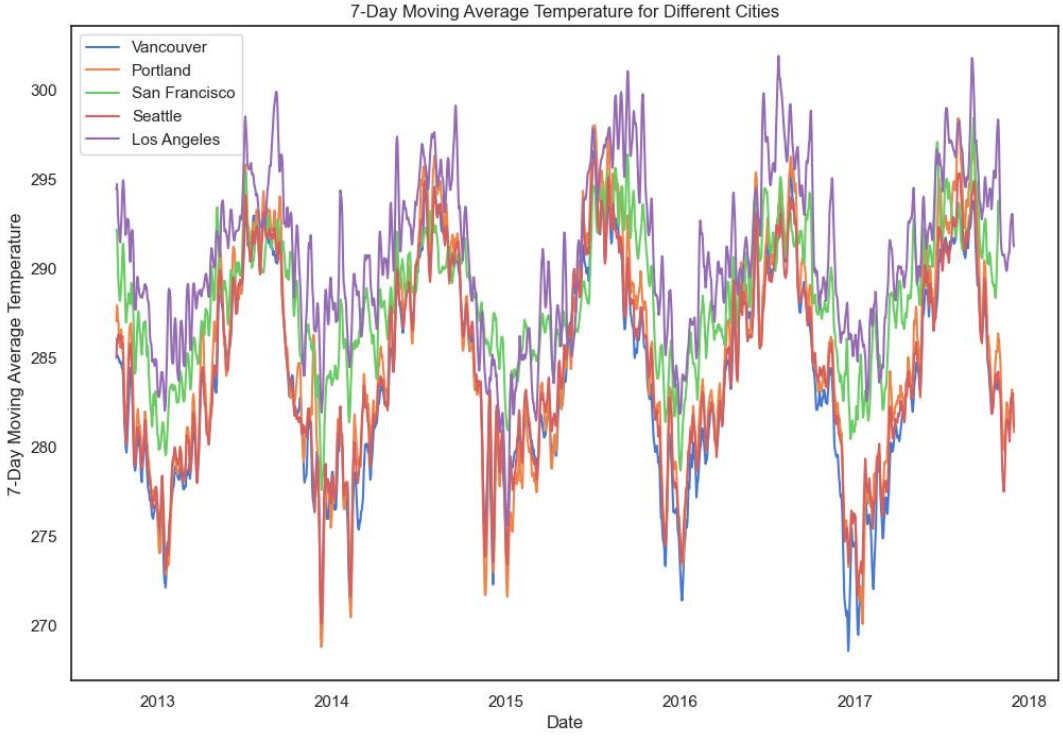
Other exercises were performed, including visualizing the temperature, humidity, and atmospheric pressure of different cities in the same plot and creating interactive checkboxes for city visibility.





(3) Weather Data Exploration:

Temperature, humidity, and atmospheric pressure variations were explored through visualization.



VI. Some of My Understandings Regarding the Central Limit Theorem：

In the classroom, we examined the distribution of the average of X and plotted the frequency distribution of the average of X through experiments. When the number of experiments is sufficiently large, this distribution approximates a normal distribution curve. However, it is important to note that the vertical axis in the graph represents **frequency rather than probability**.

If we were to plot the probability distribution of the average $\bar{X}$, it might provide a better understanding. In this case, the only factor influencing the normal distribution is the sample size, which is the number of variables being summed.

Let's take the example of rolling dice:

If there are two dice, the range of possible values for the sum is 2-12.

If there are three dice, the range of possible values for the sum is 3-18.

And so on...

If there are n dice, the range of possible values for the sum is n-6n. It becomes clearer when we take the average. The result is dividing the unit length of 5 in the range of 1-6 into 5n+1 parts. As n increases, the partition becomes finer, approaching a normal distribution. Refer to the following graph for a concrete example:

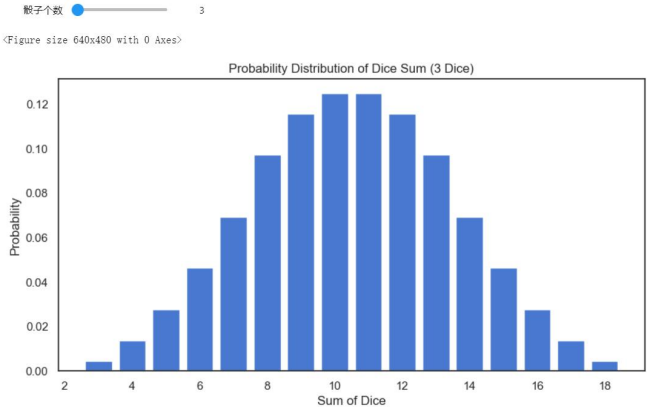
Taking the example of rolling dice, assuming:

If there are two dice, the range of possible values for the sum is 2-12.

If there are three dice, the range of possible values for the sum is 3-18.

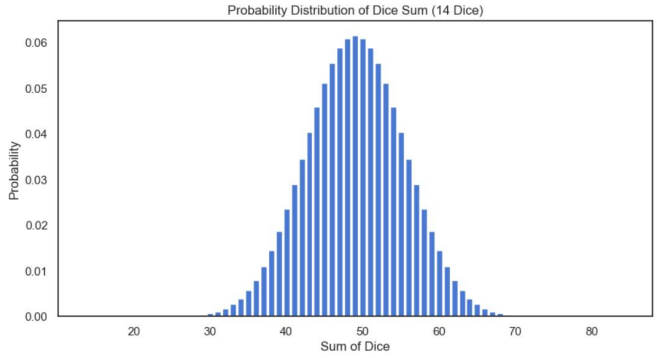
And so on...

If there are n dice, the range of possible values for the sum is **n-6n**. By taking the average of these sums, we divide each unit length of 5 in the range of 1-6 into **5n+1** parts. As n increases, the partition becomes finer, approaching a normal distribution. Please refer to the following graph for details.



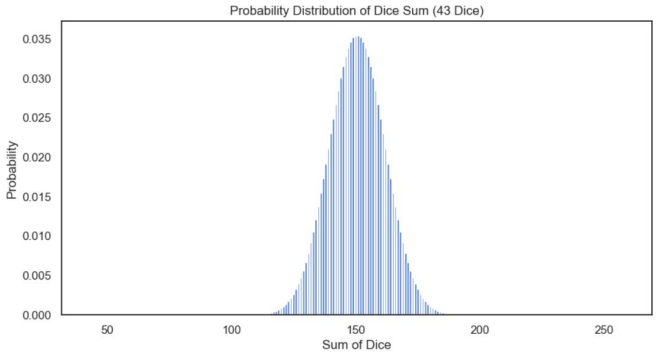
骰子个数 14

<Figure size 640x480 with 0 Axes>



骰子个数 43

<Figure size 640x480 with 0 Axes>



The code is as follows: Interactive tools from **ipywidgets** and **widgets** are used to dynamically select the sample size.

```
from ipywidgets import interact, widgets
import matplotlib.pyplot as plt
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "last"
```

```
def calculate_dice_probability(n):
    dp = [[0] * (6 * n + 1) for _ in range(n + 1)]
```

```
    for j in range(1, 7):
        dp[1][j] = 1
```

```
    for i in range(2, n + 1):
        for j in range(i, 6 * i + 1):
            for k in range(1, 7):
                if j - k > 0:
                    dp[i][j] += dp[i - 1][j - k]
```

```
    total_ways = 6 ** n
    probabilities = [count / total_ways for count in dp[n][n:]]
    return probabilities
```

```
def update_plot(n):
    result = calculate_dice_probability(n)
    plt.clf()
    plt.figure(figsize=(10, 5))
    plt.bar(range(n, 6 * n + 1), result)
    plt.xlabel('Sum of Dice')
    plt.ylabel('Probability')
    plt.title(f'Probability Distribution of Dice Sum ({n} Dice)')
    plt.show()

dice_slider = widgets.IntSlider(value=1, min=1, max=50, step=1, description='骰子个数')
interact(update_plot, n=dice_slider)
```

指导教师批阅意见：

成绩评定：

指导教师签字：

年　　月　　日

备注：