

Suraj Lamichhane  
19708  
Probability and Statics  
Week 4 Assignment:

Qno.1)

Code:

```
# Qno.1
!pip install scipy
from scipy.stats import norm

# Given data
mean = 10.3 # mean
std_dev = 0.65 # standard deviation

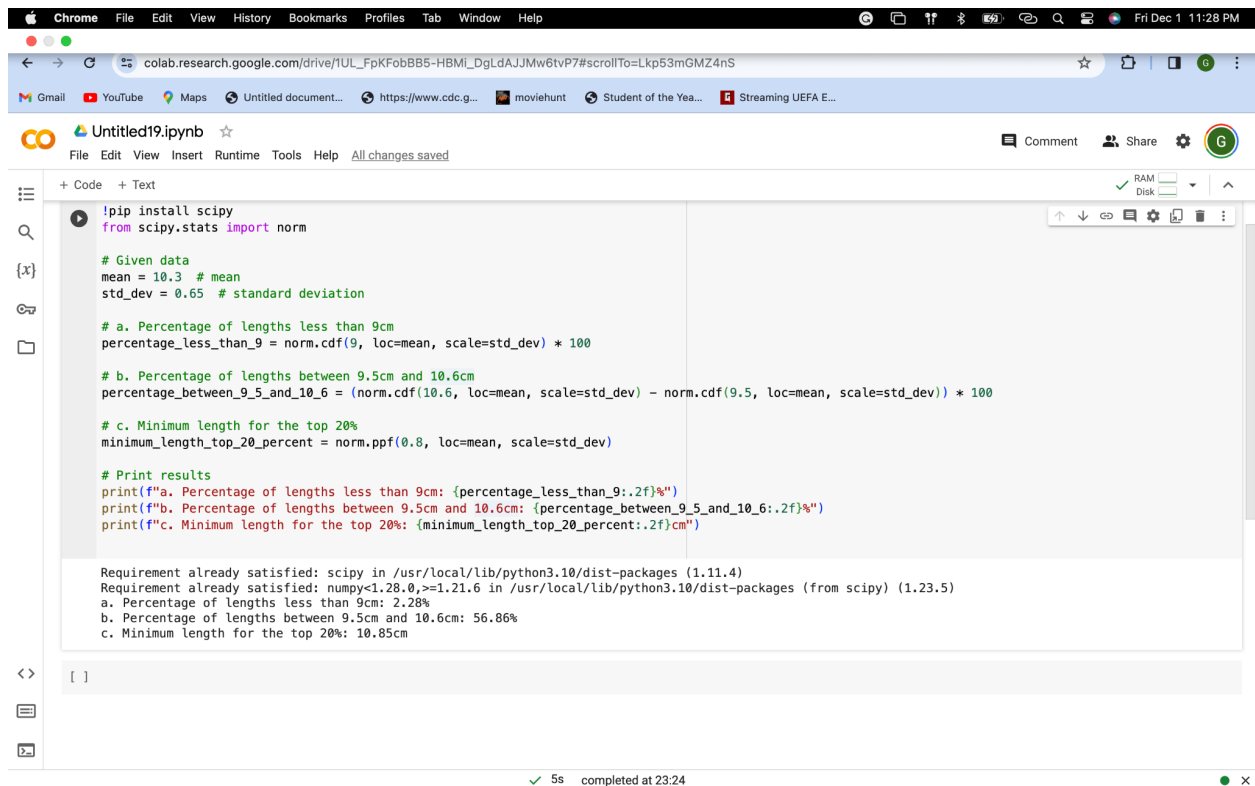
# a. Percentage of lengths less than 9cm
percentage_less_than_9 = norm.cdf(9, loc=mean, scale=std_dev) * 100

# b. Percentage of lengths between 9.5cm and 10.6cm
percentage_between_9_5_and_10_6 = (norm.cdf(10.6, loc=mean, scale=std_dev)
- norm.cdf(9.5, loc=mean, scale=std_dev)) * 100

# c. Minimum length for the top 20%
minimum_length_top_20_percent = norm.ppf(0.8, loc=mean, scale=std_dev)

# Print results
print(f"a. Percentage of lengths less than 9cm:
{percentage_less_than_9:.2f}%")
print(f"b. Percentage of lengths between 9.5cm and 10.6cm:
{percentage_between_9_5_and_10_6:.2f}%")
print(f"c. Minimum length for the top 20%:
{minimum_length_top_20_percent:.2f}cm")
```

Output:



The screenshot shows a Google Colab notebook titled 'Untitled19.ipynb'. The code cell contains a Python script that uses the `scipy.stats` module to calculate probabilities and percentiles for a normal distribution. The script defines a normal distribution with a mean of 10.3 and a standard deviation of 0.65. It then calculates the percentage of lengths less than 9cm, the percentage of lengths between 9.5cm and 10.6cm, and the minimum length for the top 20% of the distribution. The output of the script is displayed below the code cell.

```
!pip install scipy
from scipy.stats import norm

# Given data
mean = 10.3 # mean
std_dev = 0.65 # standard deviation

# a. Percentage of lengths less than 9cm
percentage_less_than_9 = norm.cdf(9, loc=mean, scale=std_dev) * 100

# b. Percentage of lengths between 9.5cm and 10.6cm
percentage_between_9_5_and_10_6 = (norm.cdf(10.6, loc=mean, scale=std_dev) - norm.cdf(9.5, loc=mean, scale=std_dev)) * 100

# c. Minimum length for the top 20%
minimum_length_top_20_percent = norm.ppf(0.8, loc=mean, scale=std_dev)

# Print results
print(f"a. Percentage of lengths less than 9cm: {percentage_less_than_9:.2f}%")
print(f"b. Percentage of lengths between 9.5cm and 10.6cm: {percentage_between_9_5_and_10_6:.2f}%")
print(f"c. Minimum length for the top 20%: {minimum_length_top_20_percent:.2f}cm")
```

Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages (1.11.4)  
Requirement already satisfied: numpy<1.28.0,>=1.21.6 in /usr/local/lib/python3.10/dist-packages (from scipy) (1.23.5)  
a. Percentage of lengths less than 9cm: 2.28%  
b. Percentage of lengths between 9.5cm and 10.6cm: 56.86%  
c. Minimum length for the top 20%: 10.85cm

Qno2)

```
# Qno.2
import numpy as np

# Parameters for X
mean_X = 10
std_dev_X = 3

# Parameters for Y
mean_Y = 15
std_dev_Y = 8

# Calculations
```

```

mean_X_plus_Y = mean_X + mean_Y
variance_X_plus_Y = std_dev_X**2 + std_dev_Y**2

mean_X_minus_Y = mean_X - mean_Y
variance_X_minus_Y = std_dev_X**2 + std_dev_Y**2

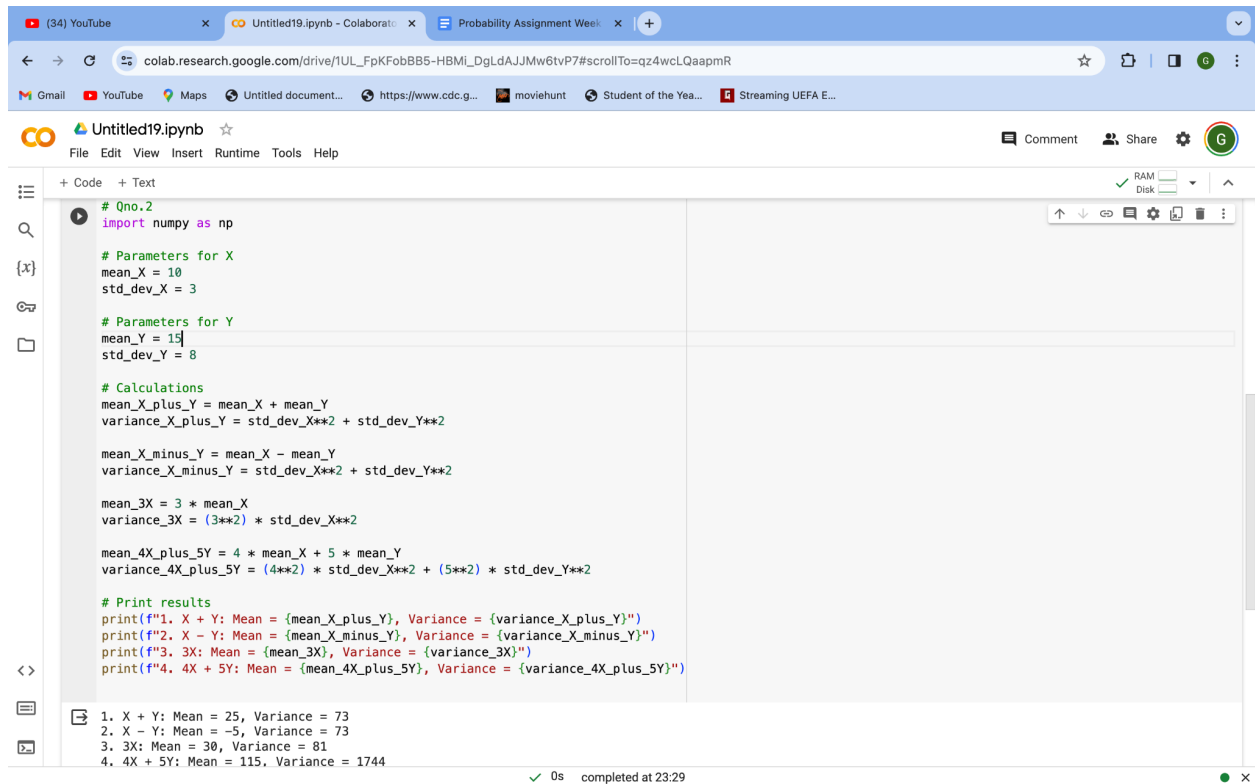
mean_3X = 3 * mean_X
variance_3X = (3**2) * std_dev_X**2

mean_4X_plus_5Y = 4 * mean_X + 5 * mean_Y
variance_4X_plus_5Y = (4**2) * std_dev_X**2 + (5**2) * std_dev_Y**2

# Print results
print(f"1. X + Y: Mean = {mean_X_plus_Y}, Variance = {variance_X_plus_Y}")
print(f"2. X - Y: Mean = {mean_X_minus_Y}, Variance = {variance_X_minus_Y}")
print(f"3. 3X: Mean = {mean_3X}, Variance = {variance_3X}")
print(f"4. 4X + 5Y: Mean = {mean_4X_plus_5Y}, Variance = {variance_4X_plus_5Y}")

```

Output:



```
# Qno.2
import numpy as np

# Parameters for X
mean_X = 10
std_dev_X = 3

# Parameters for Y
mean_Y = 15
std_dev_Y = 8

# Calculations
mean_X_plus_Y = mean_X + mean_Y
variance_X_plus_Y = std_dev_X**2 + std_dev_Y**2

mean_X_minus_Y = mean_X - mean_Y
variance_X_minus_Y = std_dev_X**2 + std_dev_Y**2

mean_3X = 3 * mean_X
variance_3X = (3**2) * std_dev_X**2

mean_4X_plus_5Y = 4 * mean_X + 5 * mean_Y
variance_4X_plus_5Y = (4**2) * std_dev_X**2 + (5**2) * std_dev_Y**2

# Print results
print(f"1. X + Y: Mean = {mean_X_plus_Y}, Variance = {variance_X_plus_Y}")
print(f"2. X - Y: Mean = {mean_X_minus_Y}, Variance = {variance_X_minus_Y}")
print(f"3. 3X: Mean = {mean_3X}, Variance = {variance_3X}")
print(f"4. 4X + 5Y: Mean = {mean_4X_plus_5Y}, Variance = {variance_4X_plus_5Y}")
```

1. X + Y: Mean = 25, Variance = 73  
2. X - Y: Mean = -5, Variance = 73  
3. 3X: Mean = 30, Variance = 81  
4. 4X + 5Y: Mean = 115, Variance = 1744

0s completed at 23:29

Qno.3)

Code:

```
#Qno.3
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import binom

# Parameters
p = 0.05
n = 100

# Generate binomial distribution
binomial_dist = binom(n, p)
```

```

# Calculate mean and standard deviation
mean_calculated = n * p
std_dev_calculated = np.sqrt(n * p * (1 - p))

# Generate random samples from the binomial distribution
samples = binomial_dist.rvs(size=1000)

# Plot histogram of the samples
plt.hist(samples, bins=np.arange(0, n+2)-0.5, density=True, alpha=0.75,
color='skyblue', edgecolor='black')

# Plot the theoretical probability mass function
x = np.arange(0, n+1)
pmf = binomial_dist.pmf(x)
plt.vlines(x, 0, pmf, colors='red', linewidth=2, label='Theoretical PMF')

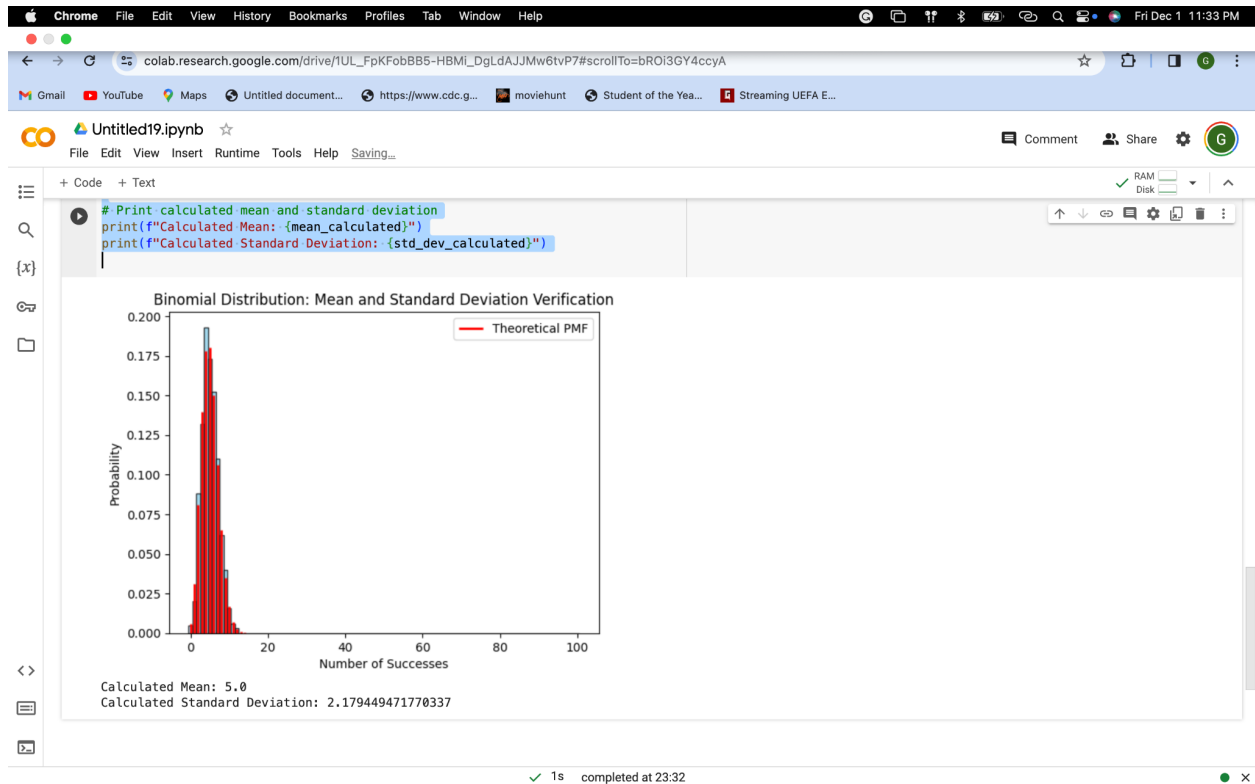
# Add labels and title
plt.xlabel('Number of Successes')
plt.ylabel('Probability')
plt.title('Binomial Distribution: Mean and Standard Deviation
Verification')
plt.legend()

# Show the plot
plt.show()

# Print calculated mean and standard deviation
print(f"Calculated Mean: {mean_calculated}")
print(f"Calculated Standard Deviation: {std_dev_calculated}")

```

Output:



Qno.4)

Code:

```
#Qno4
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import binom, norm

# Parameters
n = 100
p = 0.2

# Calculate np and nq
np_value = n * p
```

```

nq_value = n * (1 - p)

# Generate binomial distribution
binomial_dist = binom(n, p)

# Generate random samples from the binomial distribution
samples = binomial_dist.rvs(size=1000)

# Plot histogram of the samples
plt.hist(samples, bins=np.arange(0, n+2)-0.5, density=True, alpha=0.75,
color='skyblue', edgecolor='black')

# Plot the theoretical probability mass function
x = np.arange(0, n+1)
pmf = binomial_dist.pmf(x)
plt.vlines(x, 0, pmf, colors='red', linewidth=2, label='Binomial PMF')

# Plot the normal distribution for comparison
x_normal = np.linspace(0, n, 1000)
pdf_normal = norm.pdf(x_normal, loc=np_value, scale=np.sqrt(n * p * (1 -
p)))
plt.plot(x_normal, pdf_normal, label='Normal Distribution', color='green',
linestyle='dashed')

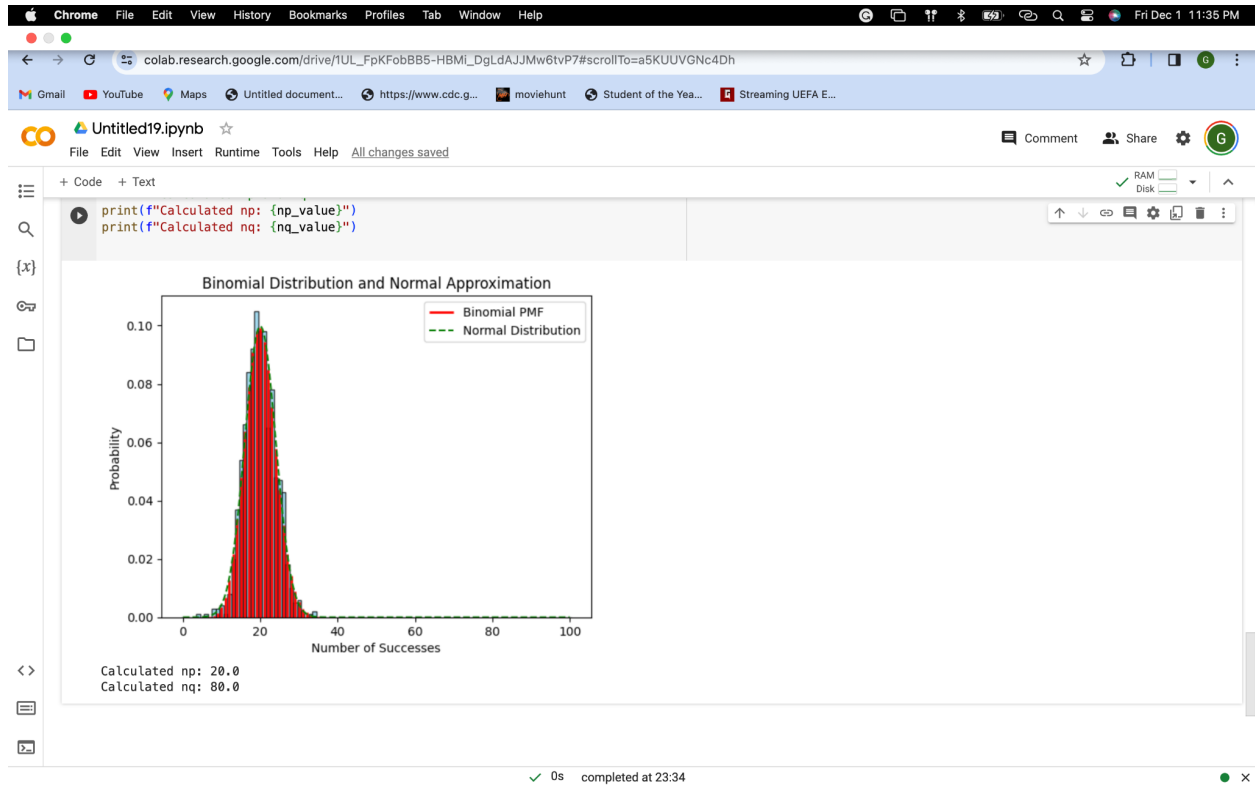
# Add labels and title
plt.xlabel('Number of Successes')
plt.ylabel('Probability')
plt.title('Binomial Distribution and Normal Approximation')
plt.legend()

# Show the plot
plt.show()

# Print calculated np and nq
print(f"Calculated np: {np_value}")
print(f"Calculated nq: {nq_value}")

```

Output:



Qno.5)

Code:

```
import numpy as np
from scipy.stats import norm

# Parameters
n = 12
p = 0.5

# Calculate mean and standard deviation
```



```

mean = n * p
std_dev = np.sqrt(n * p * (1 - p))

# Apply continuity correction
lower_bound = 6 - 0.5
upper_bound = 6 + 0.5

# Use cumulative distribution function (CDF) for the normal distribution
probability = norm.cdf(upper_bound, loc=mean, scale=std_dev) -
norm.cdf(lower_bound, loc=mean, scale=std_dev)

print(f"The probability of getting exactly 6 heads in 12 coin tosses is
approximately: {probability:.4f}")

```

Output:

The screenshot shows a Google Colab notebook interface. The top bar indicates the browser is Chrome, and the address bar shows the Colab URL. The notebook is titled 'Untitled19.ipynb'. The code cell contains the following Python code:

```

# Qno.5
import numpy as np
from scipy.stats import norm

# Parameters
n = 12
p = 0.5

# Calculate mean and standard deviation
mean = n * p
std_dev = np.sqrt(n * p * (1 - p))

# Apply continuity correction
lower_bound = 6 - 0.5
upper_bound = 6 + 0.5

# Use cumulative distribution function (CDF) for the normal distribution
probability = norm.cdf(upper_bound, loc=mean, scale=std_dev) - norm.cdf(lower_bound, loc=mean, scale=std_dev)

print(f"The probability of getting exactly 6 heads in 12 coin tosses is approximately: {probability:.4f}")

```

The output of the code is displayed below the code cell: "The probability of getting exactly 6 heads in 12 coin tosses is approximately: 0.2272". The status bar at the bottom indicates that the code was completed at 23:59.

Qno.6)

Code:

```
import numpy as np
from scipy.stats import norm

# Given data
n = 150 # number of batteries
p_defective = 0.06 # defective rate

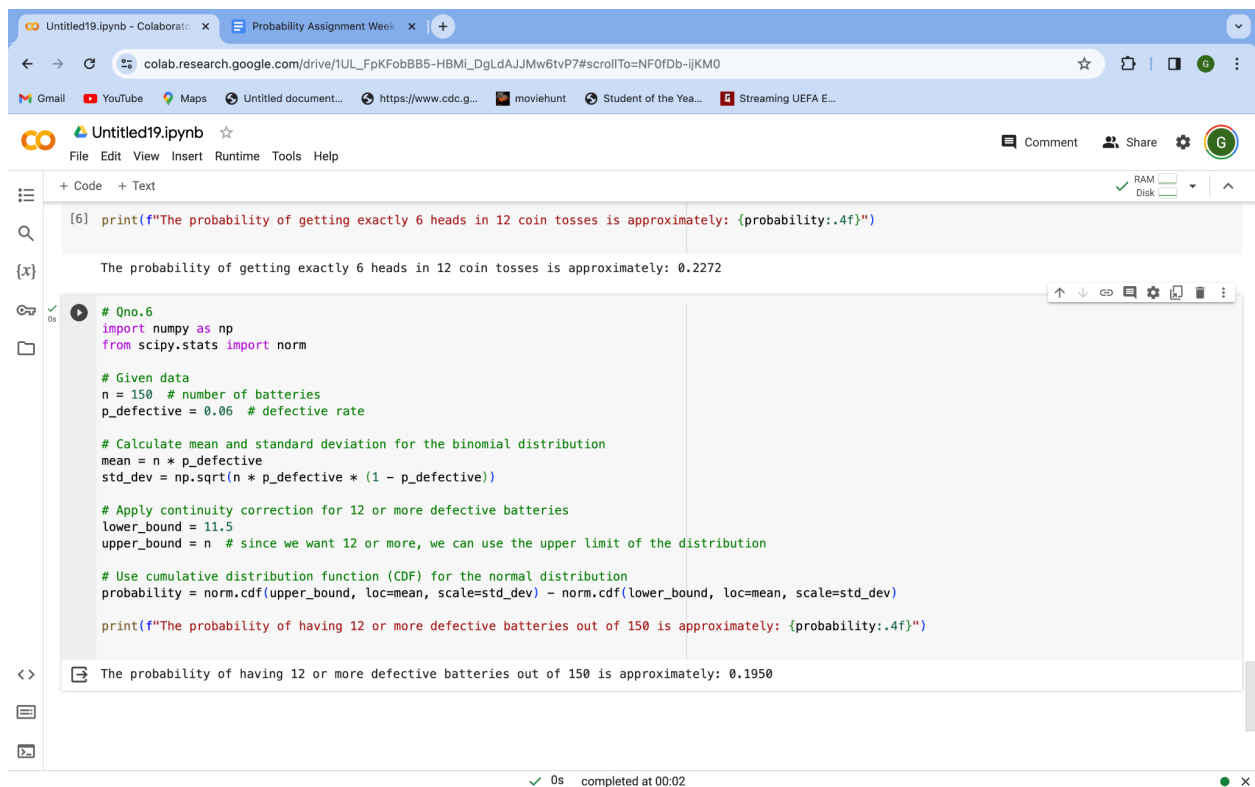
# Calculate mean and standard deviation for the binomial distribution
mean = n * p_defective
std_dev = np.sqrt(n * p_defective * (1 - p_defective))

# Apply continuity correction for 12 or more defective batteries
lower_bound = 11.5
upper_bound = n # since we want 12 or more, we can use the upper limit of
the distribution

# Use cumulative distribution function (CDF) for the normal distribution
probability = norm.cdf(upper_bound, loc=mean, scale=std_dev) -
norm.cdf(lower_bound, loc=mean, scale=std_dev)

print(f"The probability of having 12 or more defective batteries out of
150 is approximately: {probability:.4f}")
```

Output:



The screenshot shows a Google Colab notebook titled 'Untitled19.ipynb'. The browser address bar shows the URL: [colab.research.google.com/drive/1UL\\_FpKFobBB5-HBML\\_DgLdAJJMw6tvP7#scrollTo=NF0fDb-ijKM0](https://colab.research.google.com/drive/1UL_FpKFobBB5-HBML_DgLdAJJMw6tvP7#scrollTo=NF0fDb-ijKM0). The notebook interface includes a menu bar (File, Edit, View, Insert, Runtime, Tools, Help) and a toolbar with icons for code, text, and output. The code cell contains the following Python code:

```
[6] print(f"The probability of getting exactly 6 heads in 12 coin tosses is approximately: {probability:.4f}")
```

The output of this cell is: "The probability of getting exactly 6 heads in 12 coin tosses is approximately: 0.2272".

The next code cell is titled '# Qno.6' and contains the following Python code:

```
# Qno.6
import numpy as np
from scipy.stats import norm

# Given data
n = 150 # number of batteries
p_defective = 0.06 # defective rate

# Calculate mean and standard deviation for the binomial distribution
mean = n * p_defective
std_dev = np.sqrt(n * p_defective * (1 - p_defective))

# Apply continuity correction for 12 or more defective batteries
lower_bound = 11.5
upper_bound = n # since we want 12 or more, we can use the upper limit of the distribution

# Use cumulative distribution function (CDF) for the normal distribution
probability = norm.cdf(upper_bound, loc=mean, scale=std_dev) - norm.cdf(lower_bound, loc=mean, scale=std_dev)

print(f"The probability of having 12 or more defective batteries out of 150 is approximately: {probability:.4f}")
```

The output of this cell is: "The probability of having 12 or more defective batteries out of 150 is approximately: 0.1950".

The bottom status bar shows a green checkmark, '0s', and 'completed at 00:02'.

Qno.7)

Code:

```
# Qno.7
import numpy as np
import matplotlib.pyplot as plt

# Function to generate random numbers from a t-distribution
def generate_t_distribution_samples(size, df):
    return np.random.standard_t(df, size=size)

# Parameters
```

```

total_samples = 15
sample_size = 30
degrees_of_freedom = 10

# Generate 100 random numbers from t-distribution
population = generate_t_distribution_samples(100, degrees_of_freedom)

# Create 15 sampling groups
sampling_groups = [np.random.choice(population, size=sample_size,
replace=False) for _ in range(total_samples)]

# Calculate mean and standard deviation for each sampling group
means = [np.mean(group) for group in sampling_groups]
std_devs = [np.std(group, ddof=1) for group in sampling_groups]

# Calculate overall mean and standard deviation based on CLT
overall_mean = np.mean(means)
overall_std_dev = np.std(means, ddof=1) / np.sqrt(sample_size)

# Plot histogram of means
plt.hist(means, bins=15, density=True, alpha=0.75, color='skyblue',
edgecolor='black')
plt.xlabel('Sample Mean')
plt.ylabel('Probability Density')
plt.title('Distribution of Sample Means')
plt.show()

# Print results
print(f"Overall Mean: {overall_mean}")
print(f"Overall Standard Deviation (CLT): {overall_std_dev}")

```

Output: