Homework Assignment (Problem Set) 4:

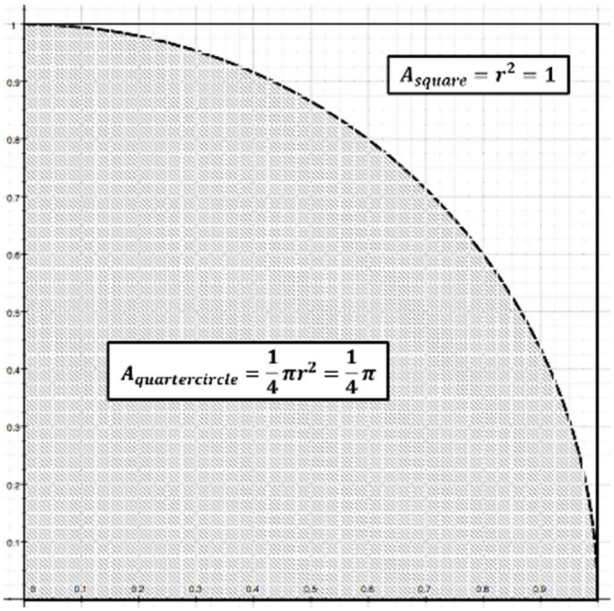
Note, Problem Set 3 directly focuses on Modules 7 and 8: Simulation and Monte Carlo Simulation

Steve Desilets

3 Questions

Rubric:
All questions worth 50 points
50 Points: Answer and solution are fully correct and detailed professionally.
25-49 Points: Answer and solution are deficient in some manner but mostly correct.
15-24 Points: Answer and solution are missing a key element or two.
1-14 Points: Answer and solution are missing multiple elements are significantly deficient/incomprehensible.
0 Points: No answer provided.

1. Perform Monte Carlo integration using R statistical programming or Python programming to estimate the value of π. To summarize the approach, consider the unit quarter circle illustrated in the figure below:



Generate N pairs of uniform random numbers (x, y), where $x \sim U(0,1)$ and $y \sim U(0,1)$, and each (x, y) pair represents a point in the unit square. To obtain an estimate of π, count the fraction of points that fall inside the unit quarter circle and multiply by 4. Note that the fraction of points that fall inside the quarter circle should tend to the ratio between the area of the unit quarter circle (i.e., ¼ π) as compared to area of the unit square (i.e., 1). We proceed step-by-step:

a) Create a function insidecircle that takes two inputs between 0 and 1 and returns 1 if these points fall within the unit circle.

The Python code below creates a function as described in the directions. The function "insidecircle" returns 1 if the inputs fall within the unit circle and 0 if the inputs do not fall within the unit circle.

```
In [63]: # Let's create a function as prescribed in the directions

def insidecircle(x, y):
    if (x**2) + (y**2) < 1:
        return(1)
    else:
        return(0)
```

b) Create a function estimatepi that takes a single input N, generates N pairs of uniform random numbers and uses insidecircle to produce an estimate of π as described above. In addition to the estimate of π, estimatepi should also return the standard error of this estimate, and a 95% confidence interval for the estimate.

The Python code below creates a function called estimatepi that takes one input, N, and generates N pairs of random numbers drawn from uniform distributions between 0 and 1. The code plugs those pairs of numbers into the insidecircle function and stores the outputs from insidecircle (which are indicator variables indicating whether the point is inside the unit circle) in a list called inside_circle_list. The function then calculates and prints the estimate for pi, as well as the 95% confidence interval for the value of pi and the width of the confidence interval.

To provide clarity on how the estimate for pi is calculated, we use the formula below to estimate pi:

$$\pi = 4 * \frac{Number\ of\ Points\ Generated\ That\ Fall\ Inside\ the\ Unit\ Circle}{Total\ Number\ of\ Points\ Generated}$$

Since the indicator variables (reflecting whether points fall within the unit circle) essentially are drawn from a binomial distribution, we modify the traditional formula for the standard error of the mean of a binomial distribution (given below) to find the standard error of the estimate of pi.

$$Standard\ Error(p) = \sqrt{\frac{p*(1-p)}{n}}$$

Since our estimate of pi equals 4*p in this simulation, we can find the standard error of the estimate for the mean of the binomial distribution using the formula below:

$$Standard\ Error(\pi) = 4 * \sqrt{\frac{p*(1-p)}{n}}$$

We use the above formula to solve for the standard error of the estimates of pi, which we subsequently use to calculate the 95% confidence interval for the value of pi.

In [193]:
```
# Let's create a function as prescribed in the directions

# For this function, we need to apply some knowledge of geometry.

# We know that the area of one fourth of the unit circle is pi/4.

# We also know that the area of the square between (0,0) and (1,1) is 1.

# The ratio of the area of one fourth of the unit circle to the area of this square is therefore equal to pi/4.

# The ratio of the area of one fourth of the unit circle to the area of the square is also going to equal the sum of
# in_circle_list divided by N.

# If we multiply the the figure described above by 4, then we will arrive at an estimate for pi.

from random import uniform
from statistics import stdev

in_circle_list = []

def estimatepi(N):
    for i in range(1, N+ 1):
        x = uniform(0,1)
        y = uniform(0,1)
        in_circle_list.append(insidecircle(x, y))

    pi_estimate = 4 * sum(in_circle_list) / N
    print(f"The estimate for pi is {pi_estimate}.")

    p = sum(in_circle_list) / N
    q = 1 - p

    pi_standard_error = 4 * ( ( (p * q) / N)**0.5)

    ci_95_percent = (pi_estimate - (1.96 * pi_standard_error), pi_estimate + (1.96 * pi_standard_error))

    ci_width = ci_95_percent[1] - ci_95_percent[0]

    print(f"The standard error of the estimate for pi is {pi_standard_error}.")
    print(f"The 95% confidence interval for the estimate of pi is {ci_95_percent}.")
    print(f"The width of the confidence interval is {ci_width}.")
    return(pi_estimate)
```

c) Use estimatepi to estimate π for N = 1000 to 10000 in increments of 500 and record the estimate, its standard error and the upper and lower bounds of the 95% CI. How large must N be in order to ensure that your estimate of π is within 0.1 of the true value?

The Python code below leverages the estimatepi function to generate 19 estimates for pi, as well as the standard error and 95% confidence intervals for those estimates. The first estimation uses 1,000 (X, Y)

coordinate pair and each subsequent estimation leverages 500 more simulated points than the last estimate did – all the way up to 10,000 simulated points leveraged for the final estimation of pi. To ensure that the estimate of pi is within 0.1 of the true value (with 95% confidence level), the width of the confidence interval must be less than 0.1. The first such iteration that we witness in which the width of the confidence interval is less than 0.1 occurs when the number of points generated for the estimation is 4,500 points.

In [194]:
```
n_list = []

for i in range(1, 20):
    n_list.append(500*i + 500)

for j in n_list:
    in_circle_list = []
    print(f"Results When Number Of Trials Equals {j}:")
    estimatepi(j)
    print(f" ")
```

```
Results When Number Of Trials Equals 1000:
The estimate for pi is 3.14.
The standard error of the estimate for pi is 0.051965373086315846.
The 95% confidence interval for the estimate of pi is (3.038147868750821, 3.2418521312491793).
The width of the confidence interval is 0.20370426249835827.

Results When Number Of Trials Equals 1500:
The estimate for pi is 3.1333333333333333.
The standard error of the estimate for pi is 0.04254844733207512.
The 95% confidence interval for the estimate of pi is (3.049938376562466, 3.2167282901042005).
The width of the confidence interval is 0.16678991354173434.

Results When Number Of Trials Equals 2000:
The estimate for pi is 3.196.
The standard error of the estimate for pi is 0.03584399531302279.
The 95% confidence interval for the estimate of pi is (3.1257457691864756, 3.2662542308135247).
The width of the confidence interval is 0.1405084616270491.

Results When Number Of Trials Equals 2500:
The estimate for pi is 3.0768.
The standard error of the estimate for pi is 0.0337075763590324.
The 95% confidence interval for the estimate of pi is (3.0107331503362964, 3.1428668496637036).
The width of the confidence interval is 0.13213369932740715.

Results When Number Of Trials Equals 3000:
The estimate for pi is 3.1466666666666665.
The standard error of the estimate for pi is 0.029917417198880185.
The 95% confidence interval for the estimate of pi is (3.0880285289568614, 3.2053048043764716).
The width of the confidence interval is 0.11727627541961017.

Results When Number Of Trials Equals 3500:
The estimate for pi is 3.145142857142857.
The standard error of the estimate for pi is 0.02771615226199179.
The 95% confidence interval for the estimate of pi is (3.090819198709353, 3.199466515576361).
The width of the confidence interval is 0.10864731686700768.

Results When Number Of Trials Equals 4000:
The estimate for pi is 3.13.
The standard error of the estimate for pi is 0.026091665335888393.
The 95% confidence interval for the estimate of pi is (3.0788603359416586, 3.181139664058341).
The width of the confidence interval is 0.1022793281166825.
```

```
Results When Number Of Trials Equals 4500:
The estimate for pi is 3.112888888888889.
The standard error of the estimate for pi is 0.024772199117324395.
The 95% confidence interval for the estimate of pi is (3.0643353786189333, 3.1614423991588447).
The width of the confidence interval is 0.09710702053991138.

Results When Number Of Trials Equals 5000:
The estimate for pi is 3.136.
The standard error of the estimate for pi is 0.02327876285372571.
The 95% confidence interval for the estimate of pi is (3.0903736248066975, 3.1816263751933027).
The width of the confidence interval is 0.09125275038660519.

Results When Number Of Trials Equals 5500:
The estimate for pi is 3.1258181818181816.
The standard error of the estimate for pi is 0.022289554918218792.
The 95% confidence interval for the estimate of pi is (3.082130654178473, 3.1695057094578902).
The width of the confidence interval is 0.08737505527941725.

Results When Number Of Trials Equals 6000:
The estimate for pi is 3.148.
The standard error of the estimate for pi is 0.021142752895495893.
The 95% confidence interval for the estimate of pi is (3.106560204324828, 3.189439795675172).
The width of the confidence interval is 0.08287959135034395.

Results When Number Of Trials Equals 6500:
The estimate for pi is 3.139692307692308.
The standard error of the estimate for pi is 0.020385143322403213.
The 95% confidence interval for the estimate of pi is (3.0997374267803974, 3.179647188604218).
The width of the confidence interval is 0.0799097618238207.

Results When Number Of Trials Equals 7000:
The estimate for pi is 3.1765714285714286.
The standard error of the estimate for pi is 0.01933050761830483.
The 95% confidence interval for the estimate of pi is (3.138683633639551, 3.2144592235033063).
The width of the confidence interval is 0.07577558986375532.

Results When Number Of Trials Equals 7500:
The estimate for pi is 3.1482666666666668.
The standard error of the estimate for pi is 0.01890849425424888.
The 95% confidence interval for the estimate of pi is (3.111206017928339, 3.1853273154049946).
The width of the confidence interval is 0.07412129747665563.
```

```
Results When Number Of Trials Equals 8000:
The estimate for pi is 3.1145.
The standard error of the estimate for pi is 0.018567073510653207.
The 95% confidence interval for the estimate of pi is (3.0781085359191196, 3.1508914640808805).
The width of the confidence interval is 0.07278292816176091.

Results When Number Of Trials Equals 8500:
The estimate for pi is 3.152470588235294.
The standard error of the estimate for pi is 0.01772937589974399.
The 95% confidence interval for the estimate of pi is (3.1177210114717955, 3.1872201649987923).
The width of the confidence interval is 0.06949915352699687.

Results When Number Of Trials Equals 9000:
The estimate for pi is 3.147111111111111.
The standard error of the estimate for pi is 0.01726954962664835.
The 95% confidence interval for the estimate of pi is (3.1132627938428805, 3.180959428379342).
The width of the confidence interval is 0.06769663453646135.

Results When Number Of Trials Equals 9500:
The estimate for pi is 3.1490526315789475.
The standard error of the estimate for pi is 0.016794980609385387.
The 95% confidence interval for the estimate of pi is (3.1161344695845523, 3.181970793573343).
The width of the confidence interval is 0.06583632398879047.

Results When Number Of Trials Equals 10000:
The estimate for pi is 3.1428.
The standard error of the estimate for pi is 0.016413434009980972.
The 95% confidence interval for the estimate of pi is (3.110629669340437, 3.1749703306595625).
The width of the confidence interval is 0.0643406613191253.
```

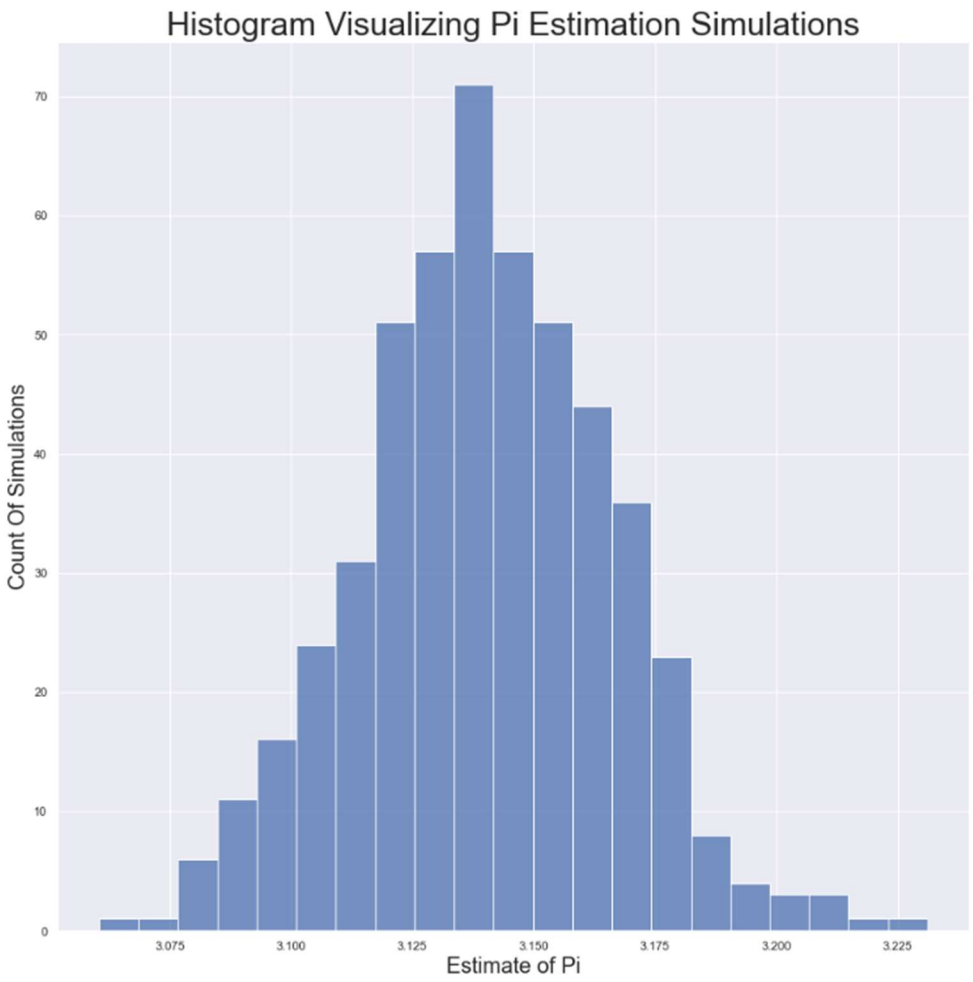
d) Using the value of N you determined in part c), run estimatepi 500 times and collect 500 different estimates of π. Produce a histogram of the estimates and note the shape of this distribution. Calculate the standard deviation of the estimates – does it match the standard error you obtained in part c)? What percentage of the estimates lies within the 95% CI you obtained in part c)?

The Python code below calculated 500 estimates for pi. To calculate each of those 500 estimates, the code generates 4,500 (X,Y) coordinate pairs via simulation (to match the value for N of 4,500 that we determined in part C). The code then visualizes the estimated values for pi in a histogram, which appears to be roughly normally distributed, with a center near the true value of pi – 3.14. The code prints the standard deviation of the 500 simulated values, which is 0.0259, which is close to the standard error of 0.0248 obtained in part C (when using 4,500 points to calculate pi). Last, the code, calculates the percentage of the simulated estimates for pi that lie within the 95% confidence interval of (3.0643353786189333, 3.1614423991588447) that was obtained in part C (when using 4,500 points to calculate pi); that percentage is 79.44%

In [216]:

```
from statistics import stdev
from IPython.utils import io

# Let's generate 500 estimates for pi. For each of these 500 simulations, we will need to generate 4,500 (X,Y) pairs

pi_simulations_list_1D = []

with io.capture_output() as captured:
    for j in range(1, 501):
        in_circle_list = []
        pi_estimate = estimatepi(4500)
        pi_simulations_list_1D.append(pi_estimate)

# Let's plot a histogram that depicts the distribution of daily bonuses
sns.set_theme(style="darkgrid")
plt.figure(figsize=(15, 15))

sns.histplot(pi_simulations_list_1D)

plt.xlabel("Estimate of Pi", fontsize = 20)
plt.ylabel("Count Of Simulations", fontsize = 20)
plt.title("Histogram Visualizing Pi Estimation Simulations", fontsize = 30)

# Let's calculate the standard deviation of the estimates of pi
pi_simulations_std_dev = stdev(pi_simulations_list_1D)
print(f"The standard deviation of the estimates of pi is {round(pi_simulations_std_dev,4)}.")

# Let's find the percentage of the estimates that lie within the 95% confidence interval that I calculated in part C
# when the number of (X, Y) pairs generated was 4,500 per estimate of pi.

inside_ci_indicator_list_1D = []

def inside_ci_function(lower_bound, upper_bound):

    for i in pi_simulations_list_1D:
        if i < lower_bound:
            inside_ci_indicator_list_1D.append(0)
        if i > upper_bound:
            inside_ci_indicator_list_1D.append(0)
        else:
            inside_ci_indicator_list_1D.append(1)

inside_ci_function(lower_bound = 3.0643353786189333, upper_bound = 3.1614423991588447)

percent_inside_ci = round(100 * sum(inside_ci_indicator_list_1D) / len(inside_ci_indicator_list_1D), 2 )

print(f"The percent of estimates for pi that lie within the 95% confidence interval identified in part C is {percent_inside_c
```

```
The standard deviation of the estimates of pi is 0.0259.
The percent of estimates for pi that lie within the 95% confidence interval identified in part C is 79.44%.
```


Histogram Visualizing Pi Estimation Simulations
Count Of Simulations
Estimate of Pi
0
10
20
30
40
50
60
70
3.075
3.100
3.125
3.150
3.175
3.200
3.225

2. A salesperson in a large bicycle shop is paid a bonus if he sells more than 4 bicycles a day. The probability of selling more than 4 bicycles a day is only 0.40. If the number of bicycles sold is greater than 4, the distribution of sales as shown below. The shop has four different models of bicycles. The amount of the bonus paid out varies by type. The bonus for model A is $10; 40% of the bicycles sold are of this type. Model B accounts for 35% of the sales and pays a bonus of $15. Model C has a bonus rating of $20 and makes up 20% of the sales. Finally, a model D pays a bonus of $25 for each sale but accounts for only 5% of the sales. Develop a simulation model to calculate the bonus a salesperson can expect in a day.

Table

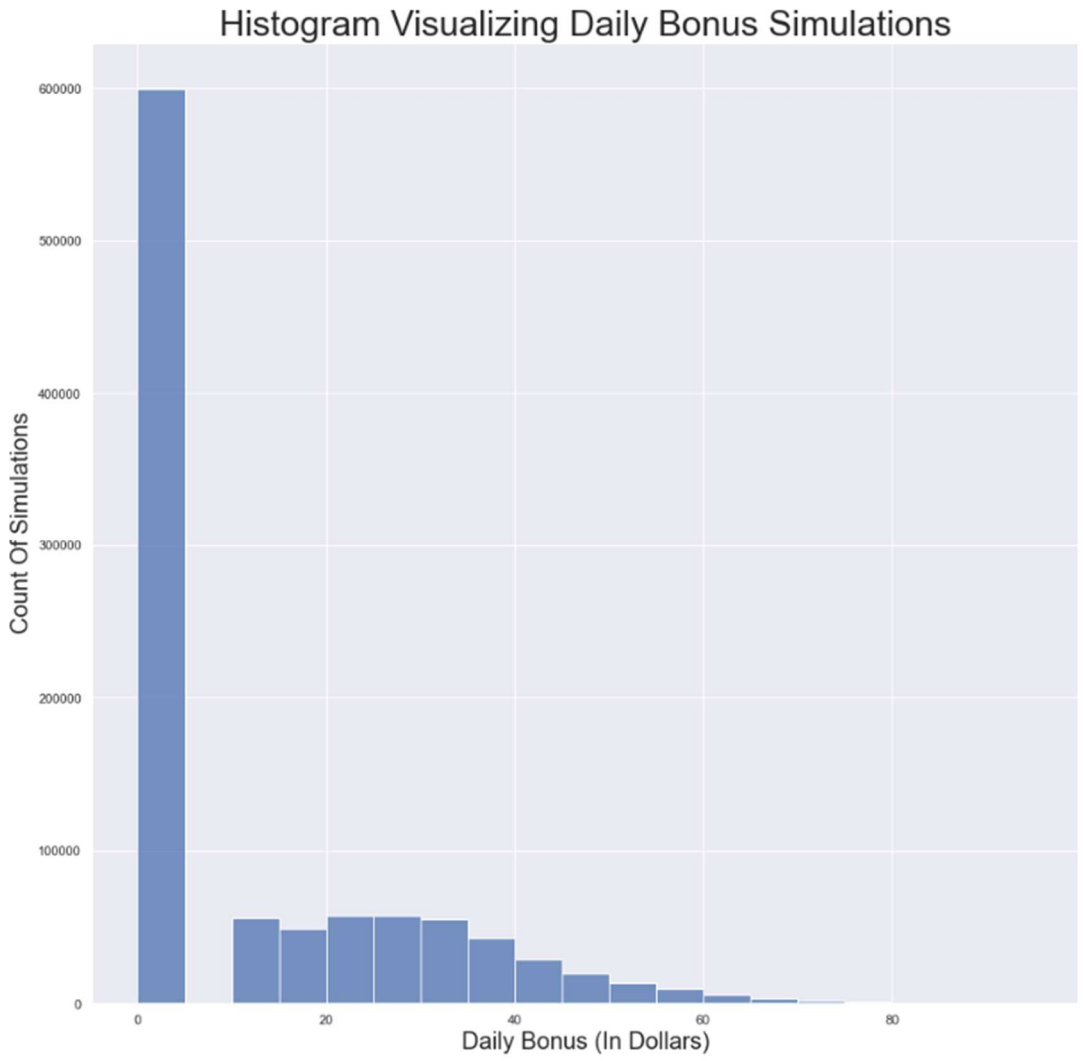
Number of Bicycles Sold	Probability
5	0.35
6	0.45
7	0.15
8	0.05

The Python code and output below display the results of a simulation model that calculates the bonus a salesperson can expect in a day. The code generates 1,000,000 bonus simulations, and the mean of these simulated bonuses is $11.02 The Python output also displays a histogram of the simulated bonus values as well.

In [184]:
```
# Let's find the mean bonus that the employee will receive via Monte Carlo Simulation
from numpy import random
from statistics import mean, stdev
import seaborn as sns


bonus_bikes_probability_distribution = [0.6, 0.4*0.35, 0.4*0.45, 0.4*0.15, 0.4*0.05]
bike_models_probability_distribution = [0.4, 0.35, 0.2, 0.05]
model_bonus_values_list = [10, 15, 20, 25]

bonus_bikes_list = []
bonus_values_list = []

def bike_bonus_simulation(number_of_simulations):
    for k in range(1, number_of_simulations + 1):
        bonus_bikes_list.append(random.choice([0, 1, 2, 3, 4], p = bonus_bikes_probability_distribution))

    for i in bonus_bikes_list:
        first_bike_bonus = 0
        second_bike_bonus = 0
        third_bike_bonus = 0
        fourth_bike_bonus = 0

        if i >= 1:
            first_bike_bonus = random.choice(model_bonus_values_list, p = bike_models_probability_distribution)
            if i >= 2:
                second_bike_bonus = random.choice(model_bonus_values_list, p = bike_models_probability_distribution)
                if i >= 3:
                    third_bike_bonus = random.choice(model_bonus_values_list, p = bike_models_probability_distribution)
                    if i == 4:
                        fourth_bike_bonus = random.choice(model_bonus_values_list, p = bike_models_probability_distribution)

        bonus_values_list.append(first_bike_bonus + second_bike_bonus + third_bike_bonus + fourth_bike_bonus)

    mean_bike_bonus = sum(bonus_values_list) / number_of_simulations

    print(f"The mean daily bonus is ${round(mean_bike_bonus,2)}.")

bike_bonus_simulation(number_of_simulations = 1000000)

# Let's plot a histogram that depicts the distribution of daily bonuses
sns.set_theme(style="darkgrid")
plt.figure(figsize=(15, 15))

sns.histplot(bonus_values_list, binwidth = 5)

plt.xlabel("Daily Bonus (In Dollars)", fontsize = 20)
plt.ylabel("Count Of Simulations", fontsize = 20)
plt.title("Histogram Visualizing Daily Bonus Simulations", fontsize = 30)
```

```
The mean daily bonus is $11.02.
```

Out[184]:
```
Text(0.5, 1.0, 'Histogram Visualizing Daily Bonus Simulations')
```


Histogram Visualizing Daily Bonus Simulations
Count Of Simulations
600000
500000
400000
300000
200000
100000
0
0
20
40
60
80
Daily Bonus (In Dollars)

3. Michael is 24 years old and has a 401(k) plan through his employer, a large financial institution. His company matches 50% of his contributions up to 6% of his salary. He currently contributes the maximum amount he can (i.e., 6%). In his 401(k), he has three funds. Investment A is a large-cap index fund, which has had an average annual growth over the past 10 years of 6.63% with a standard deviation of 13.46%. Investment B is a mid-cap index fund with a 10-year average annual growth of 9.89% and a standard deviation of 15.28%. Finally, Investment C is a small-cap Index fund with a 10-year average annual growth rate of 8.55% and a standard deviation of 16.90%. Fifty percent of his contribution is directed to Investment A, 25% to Investment B, and 25% to Investment C. His current salary is $48,000 and based on a compensation survey of financial institutions, he expects an average raise of 2.7% with a standard deviation of 0.4% each year. Develop a simulation model to predict his 401(k) balance at age 60.

The Python code and output displayed below reflect a simulation model that predicts the 401(k) balance for Michael at age 60. The Python code simulates 10,000 401(k) balances for Michael at age 60 and visualizes a histogram of these generated values. The code also finds that the mean of the generated values is $1,259,804.38.

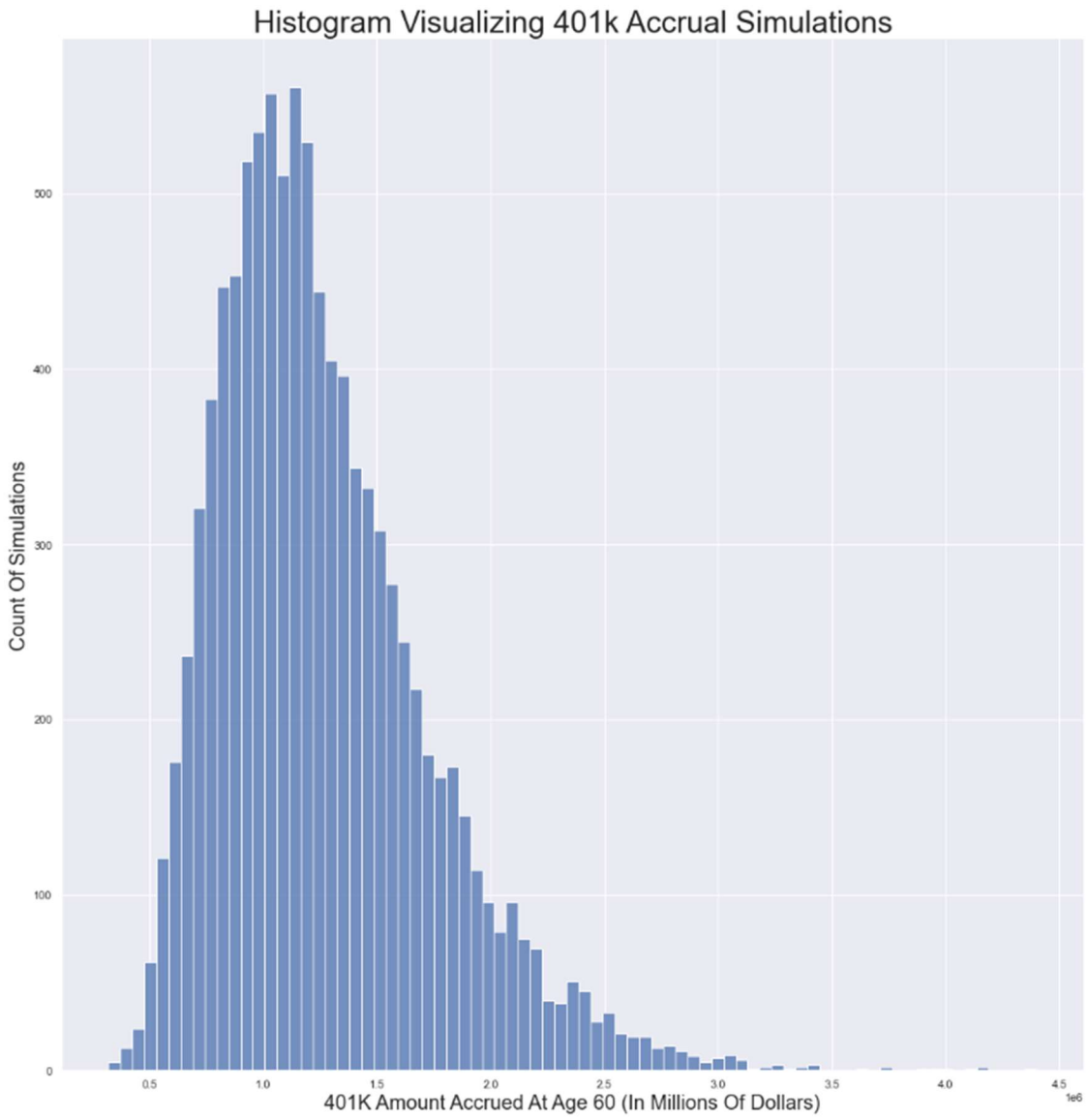
In [145]:

```
from numpy import random
import seaborn as sns


wealth_accrued_list = []

def retirement_account_simulator(number_of_simulations):

    for i in range(1, number_of_simulations + 1):

        # Simulate random fund annual returns and random annual raises
        investment_A_returns = list(random.normal(0.0663, 0.1346, 37))
        investment_B_returns = list(random.normal(0.0989, 0.1528, 37))
        investment_C_returns = list(random.normal(0.0855, 0.1690, 37))
        annual_raises = list(random.normal(0.027, 0.004, 37))

        # Calculate Michael's salary in each year
        annual_salaries = [48000]

        for i in range(0, 36):
            annual_salaries.append((1+annual_raises[i])*annual_salaries[i])

        # Calculate the annual contributions to each fund in each year
        annual_A_contributions = []
        annual_B_contributions = []
        annual_C_contributions = []

        for i in range(0, 37):
            annual_A_contributions.append(annual_salaries[i] * 0.06 * 1.5 * 0.5)
            annual_B_contributions.append(annual_salaries[i] * 0.06 * 1.5 * 0.25)
            annual_C_contributions.append(annual_salaries[i] * 0.06 * 1.5 * 0.25)

        # Calculate the beginning and end of year 401k values in each fund for the first year (when Michael is 24)
        start_of_year_A = [annual_A_contributions[0]]
        start_of_year_B = [annual_B_contributions[0]]
        start_of_year_C = [annual_C_contributions[0]]

        end_of_year_A = [annual_A_contributions[0] * (1 + investment_A_returns[0])]
        end_of_year_B = [annual_B_contributions[0] * (1 + investment_B_returns[0])]
        end_of_year_C = [annual_C_contributions[0] * (1 + investment_C_returns[0])]
        end_of_year_401ks = [end_of_year_A[0] + end_of_year_B[0] + end_of_year_C[0]]

        # Calculate the beginning and end of year 401k values in each fund for the next 36 years (when Michael is 25 - 60)
        # Note that we add all of Michael's contributions at the start of the year so that they can fully accrue interest tha
        # Also, note that we rebalance the portfolio (50% in A, 25% in B, and 25% in C) at the start of each year as well
        for i in range(1, 37):
            start_of_year_A.append((end_of_year_401ks[i - 1] * 0.5) + annual_A_contributions[i])
            start_of_year_B.append((end_of_year_401ks[i - 1] * 0.25) + annual_B_contributions[i])
            start_of_year_C.append((end_of_year_401ks[i - 1] * 0.25) + annual_C_contributions[i])

            end_of_year_A.append(start_of_year_A[i] * (1 + investment_A_returns[i]))
            end_of_year_B.append(start_of_year_B[i] * (1 + investment_B_returns[i]))
            end_of_year_C.append(start_of_year_C[i] * (1 + investment_C_returns[i]))

            end_of_year_401ks.append(end_of_year_A[i] + end_of_year_B[i] + end_of_year_C[i])

        # Find the total wealth accrued in the 401K account at the end of the year when Michael is 60
        wealth_accrued_list.append(round(end_of_year_401ks[-1],2))

retirement_account_simulator(number_of_simulations = 10000)

mean_401k_ammount = round(mean(wealth_accrued_list),2)

print(f"The mean amount accrued in the 401ks in this simulation study is ${mean_401k_ammount}." )

sns.set_theme(style="darkgrid")


sns.displot( wealth_accrued_list, height = 15)

plt.xlabel("401K Amount Accrued At Age 60 (In Millions Of Dollars)", fontsize = 20)
plt.ylabel("Count Of Simulations", fontsize = 20)
plt.title("Histogram Visualizing 401k Accrual Simulations", fontsize = 30)
```

```
The mean amount accrued in the 401ks in this simulation study is $1259804.38.
```

Out[145]:

```
Text(0.5, 1.0, 'Histogram Visualizing 401k Accrual Simulations')
```


Histogram Visualizing 401k Accrual Simulations
Count Of Simulations
0
100
200
300
400
500
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
1e6
401K Amount Accrued At Age 60 (In Millions Of Dollars)