1.a

The mean is the sum of all the terms, divided by the number of terms

1.b

The median is simply the middle value of the data set

1.c

The mode is the most repeated value

1.d

i.)

In a Gaussian distribution, the mean, median, mode are the same value, given a large enough sample size

ii.)

In a Poisson distribution, the mode is the most accurate way to describe the peak of the distribution, the mean will be displaced toward the tail of the graph, and the median will typically be very far displace towards the tail.

iii.)

In a Binomial distribution, assuming equal probabilities,it is similar to the Gaussian, but in bar graph form. It will depend on the number of bins in the binomial ditribution if it is odd, the mean, median, mode will all be the middle bar. If the bins are even, the mean and median will be in the middle, between the the central bars. The Mode, however, will lie in one or the other, or might contain 2 terms, depending on the data points.

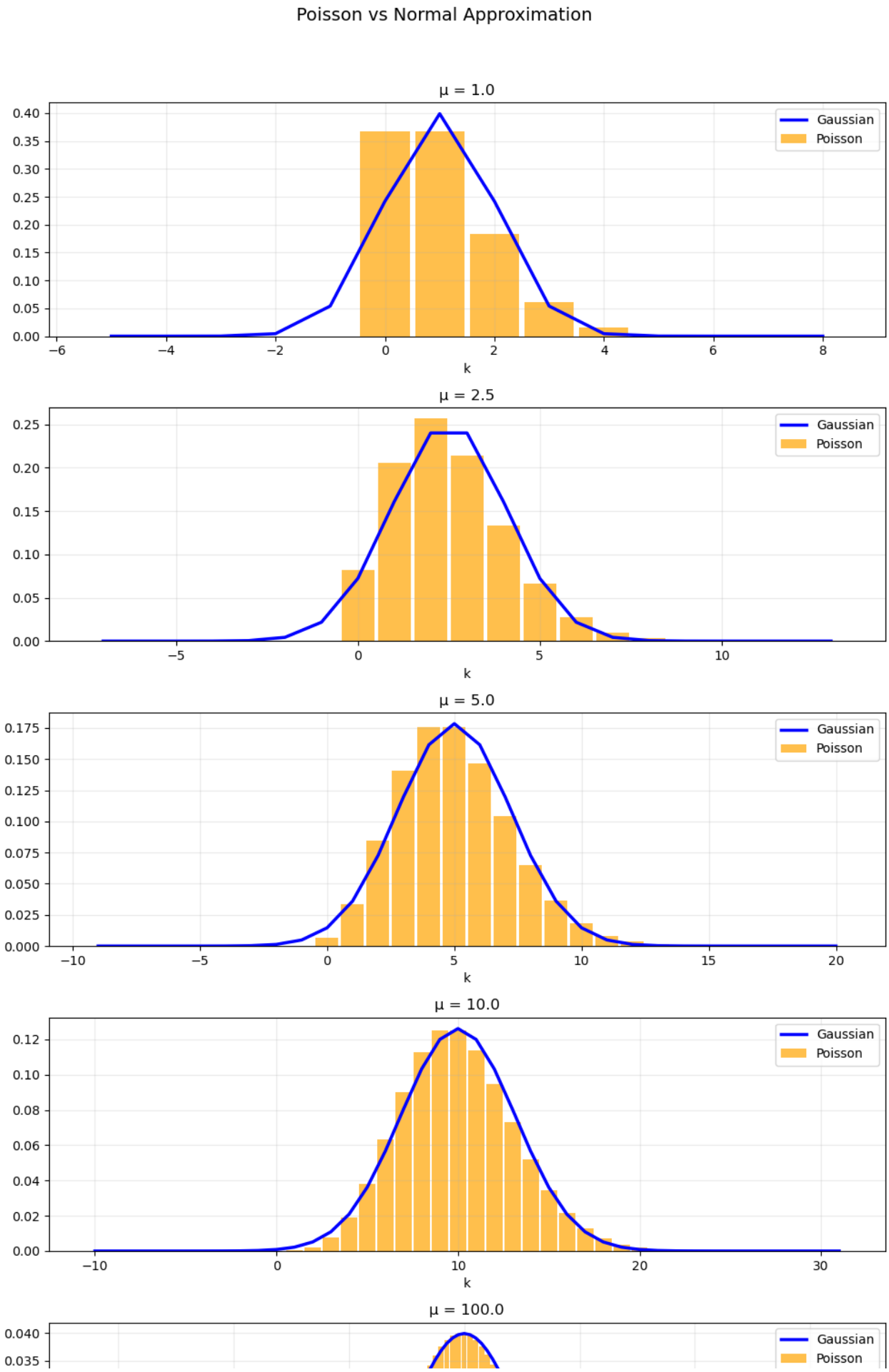
This will

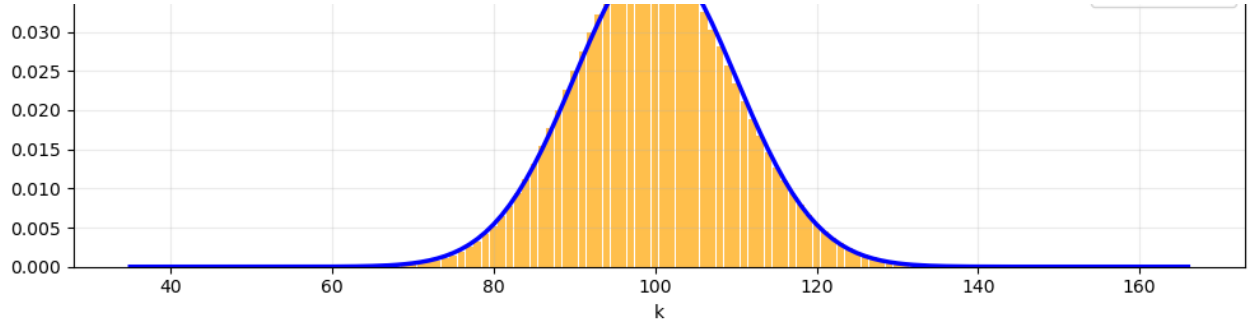
iiii.)

In a Binomial distribution, with uneven probabilities, it will be similar to the Poisson, but again, in bar graph form.

2.a-e

In [95]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm, poisson

mu_list = [1.0, 2.5, 5.0, 10.0, 100.0]

fig, axes = plt.subplots(5, 1, figsize=(10, 18), sharex=False)

for i, ax in enumerate(axes):
    mu = mu_list[i]
    sigma = np.sqrt(mu)
    k_min = max(-15, int(mu - 6.5 * sigma))
    k_max = int(mu + 6.5 * sigma) + 2
    k = np.arange(k_min, k_max, 1.0)

    ax.bar(k, poisson.pmf(k, mu),
           color='orange', alpha=0.7, width=0.9,
           label='Poisson', align='center')

    ax.plot(k, norm.pdf(k, mu, sigma),
            color='blue', lw=2.5, label='Gaussian')

    ax.set_title(f"μ = {mu}")
    ax.legend(loc='upper right')
    ax.grid(True, alpha=0.25)
    ax.set_xlabel("k")

fig.suptitle("Poisson vs Normal Approximation", fontsize=14, y=0.98)
plt.tight_layout(rect=[0, 0, 1, 0.96])
plt.show()
```


Poisson vs Normal Approximation
μ = 1.0
μ = 2.5
μ = 5.0
μ = 10.0
μ = 100.0
Gaussian
Poisson
k



2.f

I would say a gaussian is a good approximation when mu >= 100, depending on the application.

3

a

Binomial is great when there is only 2 options for probability

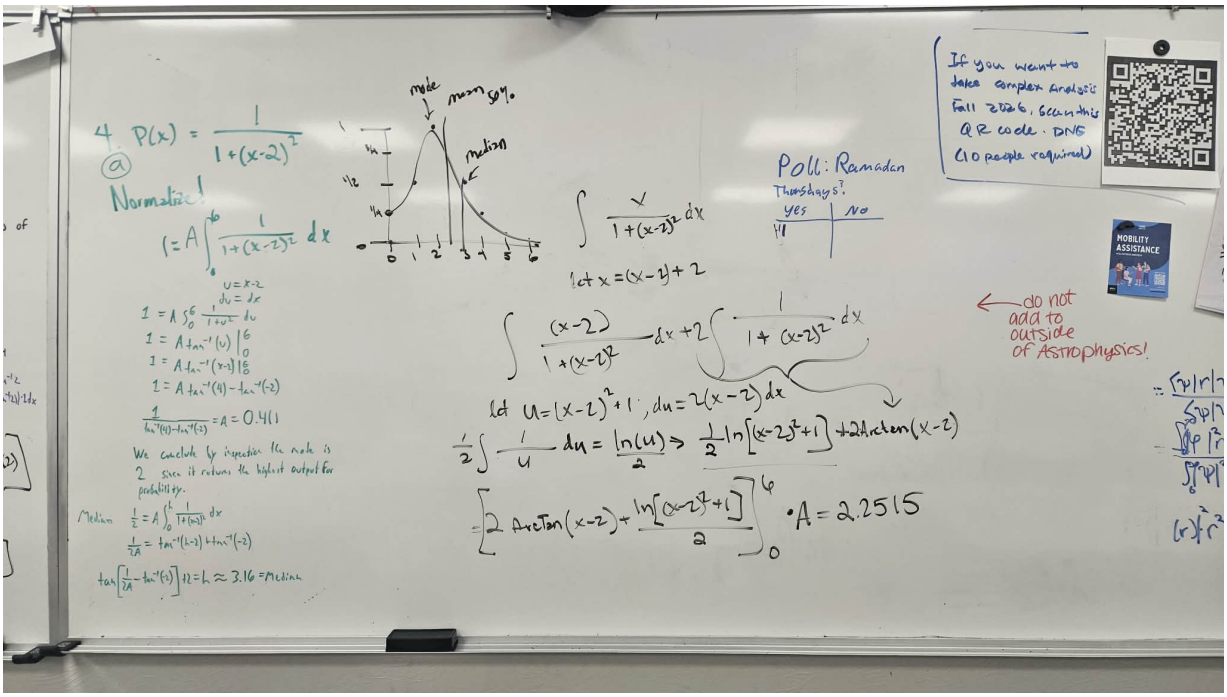
b

Poisson is great when the value cannot be less than zero, and the values are likely to be small. For example, less than 10.

c

Gaussian is great to decribe situations when there is an even distribution or normal. It also only applies to continuous variables over an infinite range.

4



a

mean = 2.25 median = 3 mode = 2

b

In [166…
```
from numpy import arctan

x= (arctan(0)+arctan(1))/(arctan(4)+arctan(2))
print(f'{x:.2f}')
```

```
0.32
```

c

In [204…
```
from numpy import arctan, log, sqrt

A= 1/(arctan(4)+arctan(2))

def f(x):
    return 2*arctan(x-2)+log((x-2)**2+1)/2

mu = A*(f(6)-f(0))

s = sqrt(A*(f(6-mu)-A*f(0-mu)))


print(f"Mean: {mu:.2f}")
print(f"sigma: {s:.2f}")

s1= (arctan(mu+s-2)-arctan(mu-s-2))/(arctan(4)+arctan(2))
s2= (arctan(mu+s+s-2)-arctan(mu-s-s-2))/(arctan(4)+arctan(2))

print(f"Probability of being within 1 sigma: {s1:.3f}")
print(f"Probability of being within 2 sigma: {s2:.3f}")
```

```
Mean: 2.25
sigma: 1.16
Probability of being within 1 sigma: 0.697
Probability of being within 2 sigma: 0.955
```

5.a

In [103…
```
from math import comb

print(f"Fraction of tosses to result in 3 heads out of 6 coins:")
print(f'{comb(6,3)}/{2**6}')
```

```
Fraction of tosses to result in 3 heads out of 6 coins:
20/64
```

5.b

In [104…
```
from math import comb

success = 2*comb(6,2)+ comb(6,3)

print("Fraction of tosses to result in 2, 3, or 4 heads:")
print(f"{success}/{2**6}")
```

```
Fraction of tosses to result in 2, 3, or 4 heads:
50/64
```

5.c

In [109…
```
from math import comb

five = comb(6,5)*2

print(f"Fraction of tosses expected to result in 5 coins showing the same:")
print(f"{five}/{2**6}")
```

```
Fraction of tosses expected to result in 5 coins showing the same:
12/64
```

6.

In [130…

```
from math import floor
from scipy.stats import binom

sigma = binom.stats(5,0.65)

print(f'Proability of heads: {floor(sigma[0])} +- {sigma[1]}')
```

```
Proability of heads: 3 +- 1.1375
```

7.a

In [142…

```
from scipy.stats import poisson

r=52/695
t=5

n=poisson.mean(r*t)**3


print(f'Probability of hitting 3 or more home runs is a game: \n{n:.3f}')
```

```
Probability of hitting 3 or more home runs is a game:
0.052
```

7.b

In [146…

```
from scipy.stats import poisson

r=52/695
t=5

n=(1-poisson.mean(r*t))**10

print(f"Probaility of not hitting a home run 10 games in a row: \n{n:.3f}")
```

```
Probaility of not hitting a home run 10 games in a row:
0.009
```

7.c)

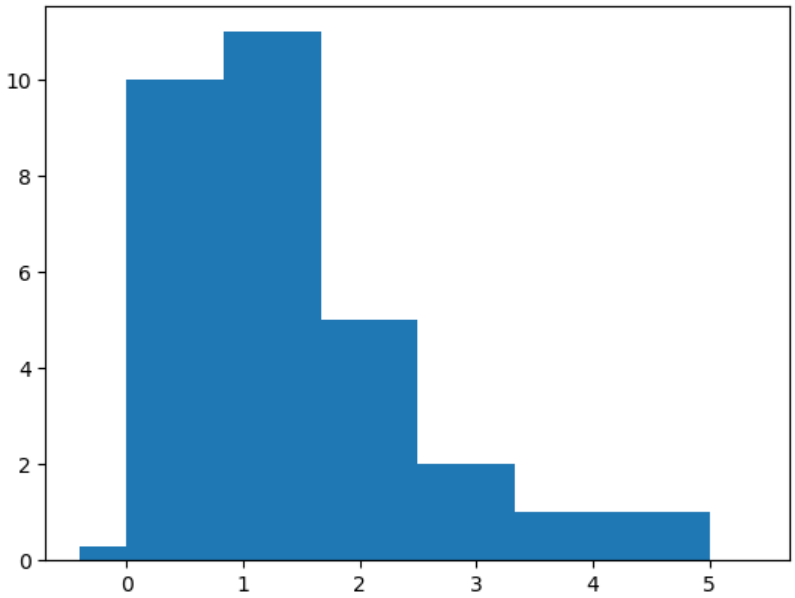
This statement is not true. This player has a 37 percent chance to hit a home run in a game. No matter what has happened in previous games.

Side not, this line of thinking is why casinos make money

8.

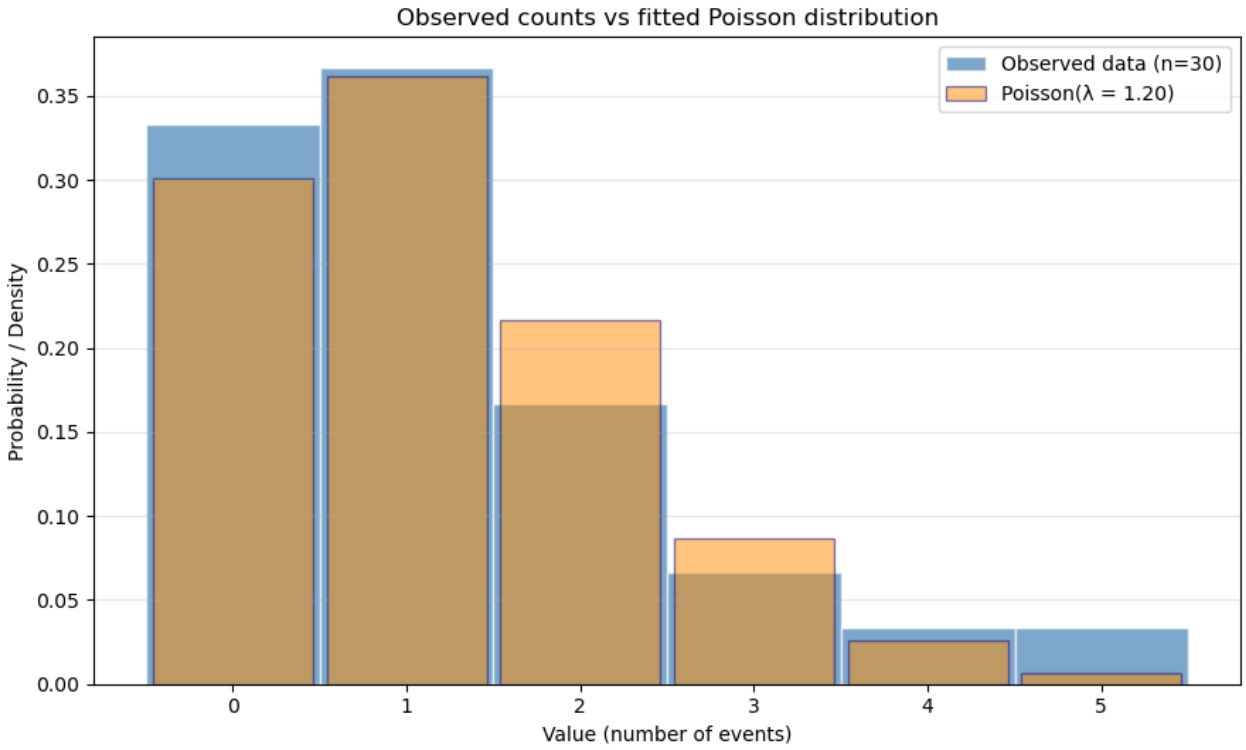
In [160…

```
from scipy.stats import poisson
from numpy import mean, arange
from matplotlib import pyplot as plt

N=[0,2,1,1,1,0,1,1,1,0,0,5,2,2,3,0,1,0,1,3,2,1,1,0,0,0,1,4,0,2]

x=arange(0,max(N)+1)
P=poisson.pmf(x,mean(N))

plt.hist(N, bins=6)
plt.bar(x,P)
plt.show()
```



In [163…

```
import numpy as np
from scipy.stats import poisson
import matplotlib.pyplot as plt

N = [0,2,1,1,1,0,1,1,1,0,0,5,2,2,3,0,1,0,1,3,2,1,1,0,0,0,1,4,0,2]

mu = np.mean(N)
var = np.var(N)
n = len(N)

x = np.arange(0, max(N) + 1)
pmf = poisson.pmf(x, mu)

plt.figure(figsize=(9, 5.5))

# Normalized histogram of the data
plt.hist(N, bins=np.arange(-0.5, max(N)+1.5, 1),
         density=True, alpha=0.7, color='steelblue', edgecolor='white',
         label=f'Observed data (n={n})')

# Poisson probability mass function
plt.bar(x, pmf, width=0.92, alpha=0.5, color='darkorange',
        edgecolor='navy', label=f'Poisson(λ = {mu:.2f})')

plt.xlabel('Value (number of events)')
plt.ylabel('Probability / Density')
plt.title('Observed counts vs fitted Poisson distribution')
plt.xticks(x)
plt.grid(True, axis='y', alpha=0.3)
plt.legend()
plt.tight_layout()
plt.show()
```



Yes, it matches, with resonable uncertainty

In [ ]: