CGS698C Assignment 2

Jiyanshu Dhaka 220481

Part 1 A simple binomial model

1.1

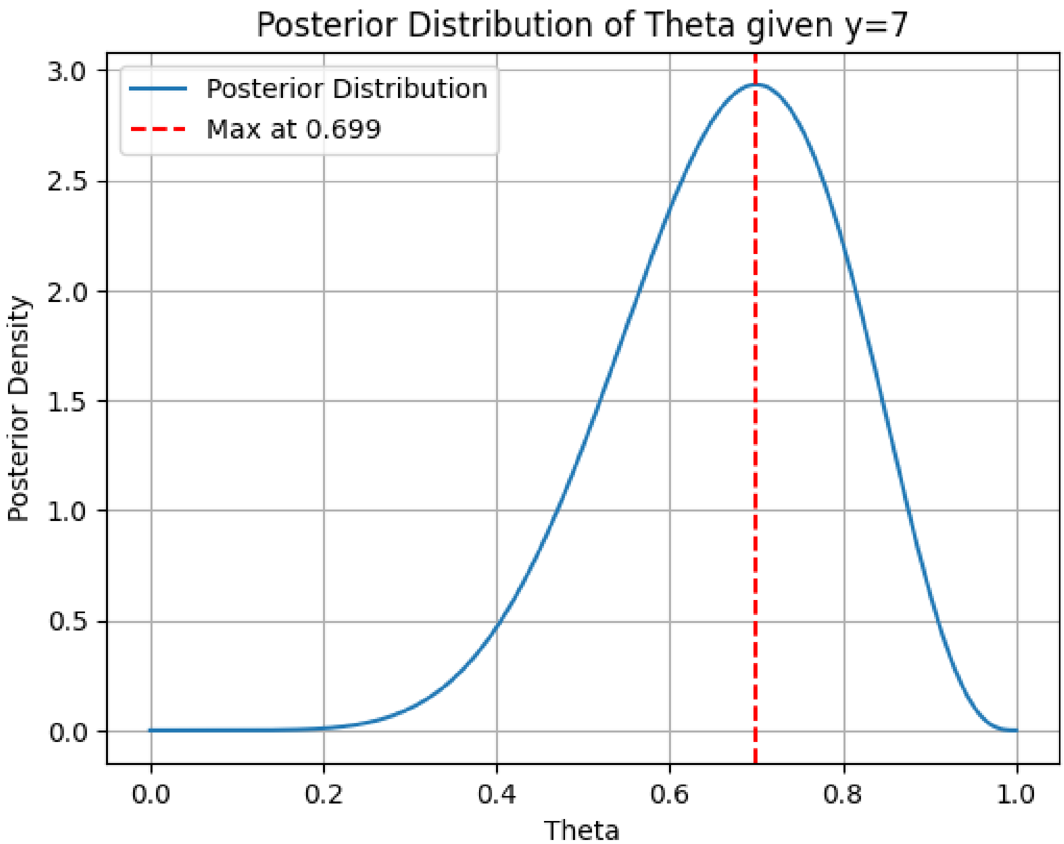
[ ]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.special import factorial

def likelihood(y, theta):
    return (factorial(10, exact=True) / (factorial(y, exact=True) *␣
 ↪factorial(10 - y, exact=True))) * (theta**y) * ((1 - theta)**(10 - y))

def prior(theta):
    if 0 <= theta <= 1:
        return 1
    else:
        return 0

def posterior(y, theta):
    return likelihood(y, theta) * prior(theta) * 11

y = 7
theta_values = [0.75, 0.25, 1]

print("Posterior Estimates:")
for theta in theta_values:
    print(f'Posterior value at theta = {theta:.2f}: {posterior(y, theta)}')
```

```
Posterior Estimates:
Posterior value at theta = 0.75: 2.7531051635742188
Posterior value at theta = 0.25: 0.03398895263671875
Posterior value at theta = 1.00: 0.0
```

1.2

[ ]:

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.special import comb

def posterior_density(theta, y=7, norm_factor=11):
    if 0 <= theta <= 1:
        likelihood = comb(10, y) * (theta**y) * ((1 - theta)**(10 - y))
        return norm_factor * likelihood
    else:
        return 0

thetas = np.linspace(0, 1, 500)
densities = [posterior_density(theta) for theta in thetas]

max_density = np.max(densities)
max_theta = thetas[np.argmax(densities)]

plt.plot(thetas, densities, label='Posterior Distribution')
plt.xlabel('Theta')
plt.ylabel('Posterior Density')
plt.title('Posterior Distribution of Theta given y=7')
plt.axvline(x=max_theta, color='r', linestyle='--', label=f'Max at {max_theta:.
 ↪3f}')
plt.legend()
plt.grid(True)

plt.show()
```



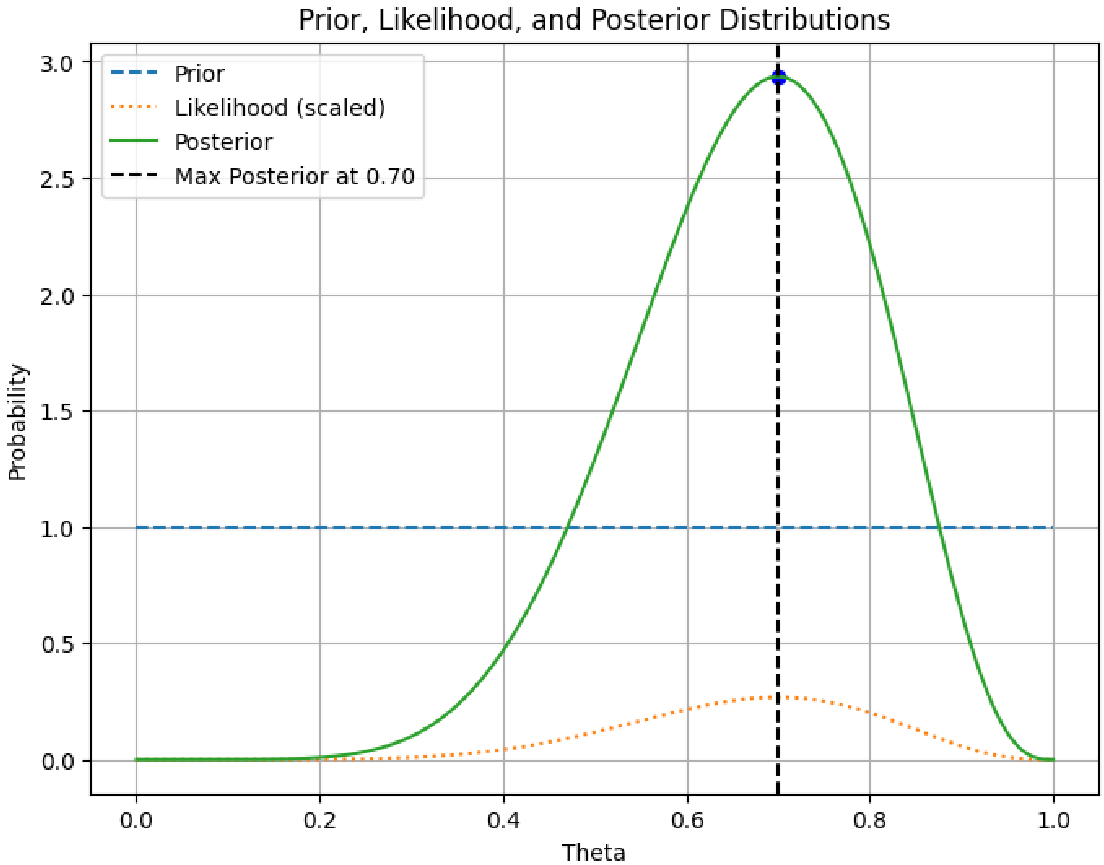
1.3

[ ]:
```python
import numpy as np
from scipy.special import comb
def posterior_density(theta, y=7, norm_factor=11):
    if 0 <= theta <= 1:
        likelihood = comb(10, y) * (theta**y) * ((1 - theta)**(10 - y))
        return norm_factor * likelihood
    else:
        return 0
thetas = np.linspace(0, 1, 500)
densities = [posterior_density(theta) for theta in thetas]
max_theta = thetas[np.argmax(densities)]
max_theta

print(f"Maximum posterior value is at theta = {max_theta} with a value of␣
↪{max_density:.4f}")
```

```
Maximum posterior value is at theta = 0.6993987975951903 with a value of 2.9351
```

1.4

[ ]:
```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.special import comb

def f(theta, y=7, n=11):
    if 0 <= theta <= 1:
        likelihood = comb(10, y) * (theta**y) * ((1 - theta)**(10 - y))
        return n * likelihood
    else:
        return 0

def g(theta, y=7):
    if 0 <= theta <= 1:
        return 120 * theta**7 * (1 - theta)**3
    else:
        return 0

x = np.linspace(0, 1, 500)
y1 = [1 if 0 <= t <= 1 else 0 for t in x]
y2 = [g(t) for t in x]
y3 = [f(t) for t in x]

m_t = x[np.argmax(y3)]
m_p = np.max(y3)

plt.figure(figsize=(8, 6))
plt.plot(x, y1, label='Prior', linestyle='--')
plt.plot(x, y2, label='Likelihood (scaled)', linestyle=':')
plt.plot(x, y3, label='Posterior')
plt.axvline(m_t, color='k', linestyle='--', label=f'Max Posterior at {m_t:.2f}')
plt.scatter(m_t, m_p, color='b')
plt.xlabel('Theta')
plt.ylabel('Probability')
plt.title('Prior, Likelihood, and Posterior Distributions')
plt.legend()
plt.grid(True)
plt.show()
```



Part 2: A Gaussian model of reading

2.1 Computing and Printing Unnormalized Posteriors

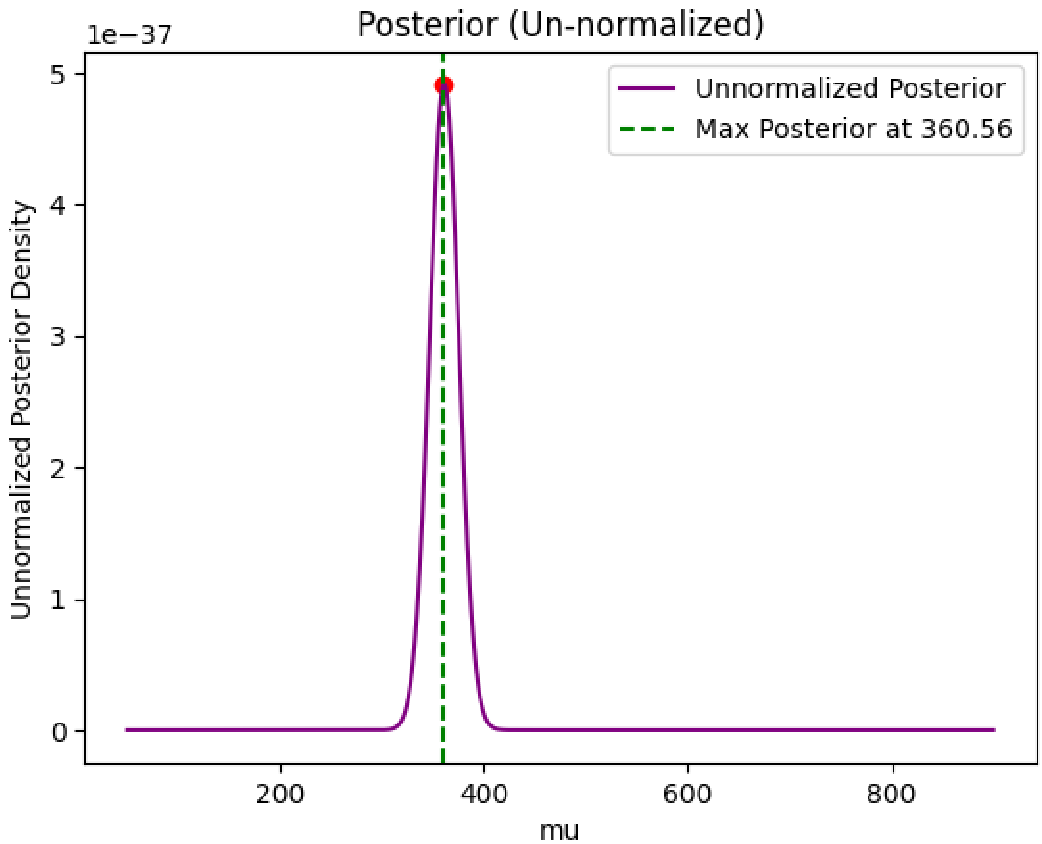
[ ]:
```
import numpy as np

def like_est(y, mu, sigma):
    n = len(y)
    return (1/((sigma*np.sqrt(2*np.pi))**n))*np.exp(-(np.sum((y - mu)**2))/
 ↪(2*(sigma**2)))

def prior_est(mu):
    mu_p = 250
    sigma_p = 25
    return (1/(np.sqrt(2*np.pi*(sigma_p**2))))*np.exp(-((mu - mu_p)**2)/
 ↪(2*(sigma_p**2)))

def post_est(y, mu, sigma):
    return like_est(y, mu, sigma)*prior_est(mu)
```

```
y = np.array([300.0, 270.0, 390.0, 450.0, 500.0, 290.0, 680.0, 450.0])
sigma = 50.0
mu_arr = [300.0, 900.0, 50.0]

print("(2.1)\n")
for mu in mu_arr:
    print(f'Value of Unnormalized Posterior at (mean = {mu:.2f}) is : ',␣
 ↪post_est(y, mu, sigma))
    print("\n")
```

```
Value of Unnormalized Posterior at (mean = 300.00) is :  6.824247957486409e-41


Value of Unnormalized Posterior at (mean = 900.00) is :  0.0


Value of Unnormalized Posterior at (mean = 50.00) is :  9.691373559300655e-138
```

2.2: Plotting Unnormalized Posterior Distribution

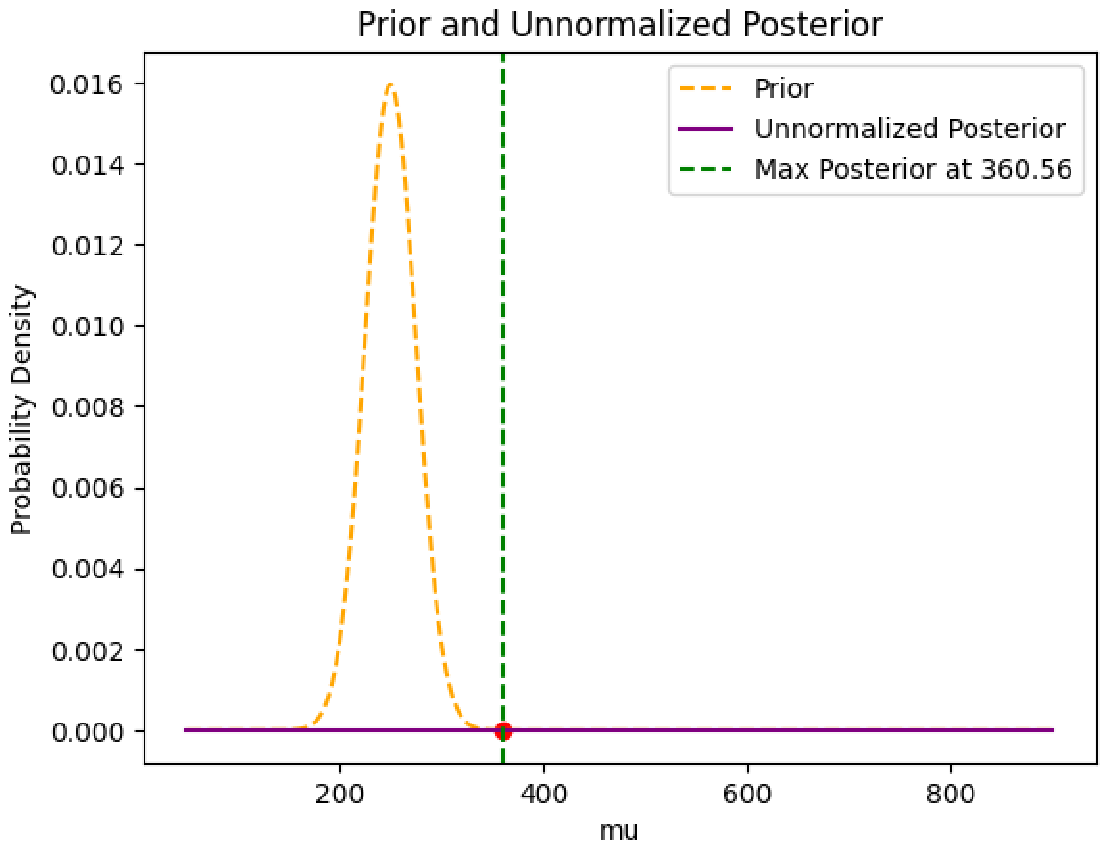
[ ]:
```
x_mu = np.linspace(50, 900, 1000)
y_post = [post_est(y, mu, sigma) for mu in x_mu]
max_idx = np.argmax(y_post)
max_mu = x_mu[max_idx]
max_post = y_post[max_idx]

plt.plot(x_mu, y_post, label='Unnormalized Posterior', linestyle='-',␣
 ↪color='purple')
plt.axvline(max_mu, color='green', linestyle='--', label=f'Max Posterior at␣
 ↪{max_mu:.2f}')
plt.scatter(max_mu, max_post, color='red')
plt.xlabel('mu')
plt.ylabel('Unnormalized Posterior Density')
plt.title('Posterior (Un-normalized)')
plt.legend()
plt.show()
print("(2.2)\n")
print("\n")
```



2.3 Plotting Prior and Scaled Unnormalized Posterior Distributions

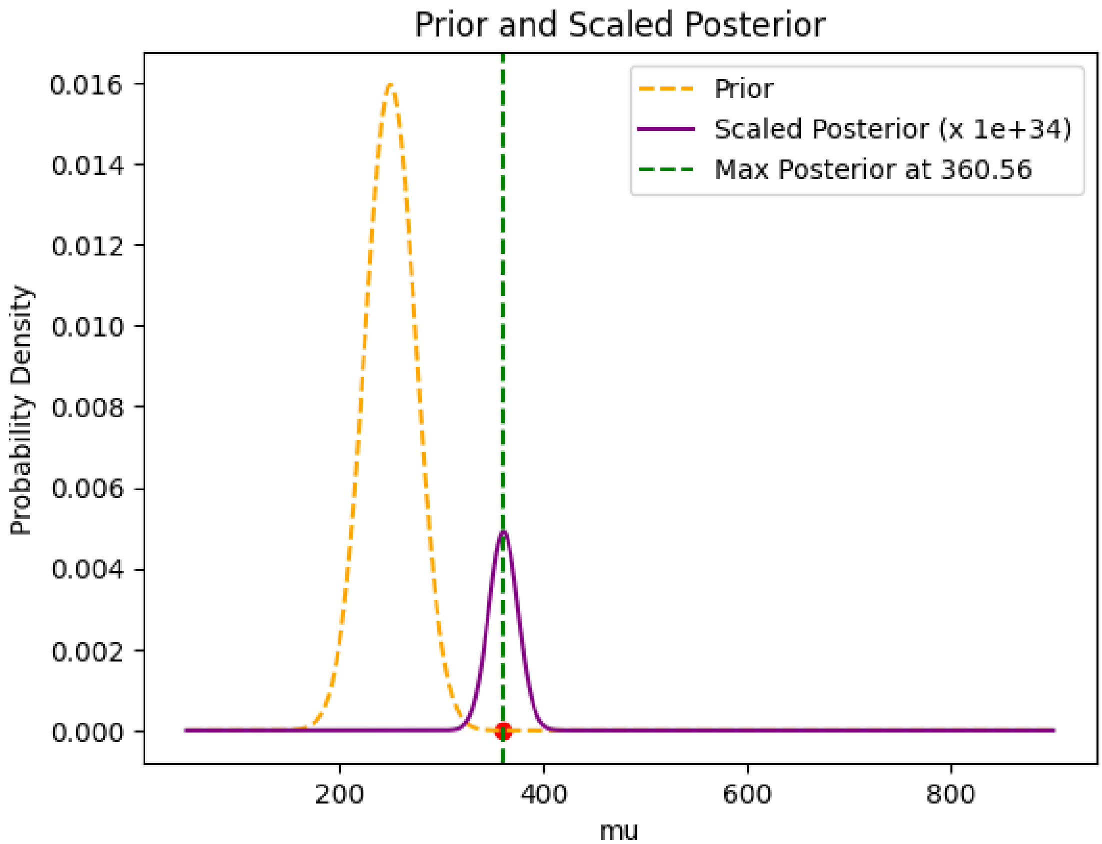
```
[ ]: x_mu = np.linspace(50, 900, 1000)
y_post = [post_est(y, mu, sigma) for mu in x_mu]
y_prior = np.array([prior_est(mu) for mu in x_mu])
max_idx = np.argmax(y_post)
max_mu = x_mu[max_idx]
max_post = y_post[max_idx]

plt.plot(x_mu, y_prior, label='Prior', linestyle='--', color='orange')
plt.plot(x_mu, y_post, label='Unnormalized Posterior', color='purple')
plt.axvline(max_mu, color='green', linestyle='--', label=f'Max Posterior at␣
↪{max_mu:.2f}')
plt.scatter(max_mu, max_post, color='red')
plt.xlabel('mu')
```

```
plt.ylabel('Probability Density')
plt.title('Prior and Unnormalized Posterior')
plt.legend()
plt.show()
print("2.3.1\n")

print("Posterior distribution appears flat due to its small values.")
print("Scaling posterior distribution.")
y_post_scaled = [(post_est(y, mu, sigma))*1e+34 for mu in x_mu]

plt.plot(x_mu, y_prior, label='Prior', linestyle='--', color='orange')
plt.plot(x_mu, y_post_scaled, label='Scaled Posterior (x 1e+34)',␣
 ↪color='purple')
plt.axvline(max_mu, color='green', linestyle='--', label=f'Max Posterior at␣
 ↪{max_mu:.2f}')
plt.scatter(max_mu, max_post, color='red')
plt.xlabel('mu')
plt.ylabel('Probability Density')
plt.title('Prior and Scaled Posterior')
plt.legend()
plt.show()
print("2.3.2\n")
print("\nScaled Posterior Distribution (x 1+34) appears alongside the prior␣
 ↪distribution.")
```



2.3.1

```
Posterior distribution appears flat due to its small values.
Scaling posterior distribution.
```



2.3.2

Scaled Posterior Distribution (x 1+34) appears alongside the prior distribution.

Part 3: The Bayesian learning

[ ]:
```
# Prior on   to generate predictions for day 5
# We start with the initial prior for  , which is
#   ~ Gamma(40, 2).
# Day 1:
# Observed data k1 = 25.
# Posterior distribution for   after Day 1:
#   ~ Gamma(40 + 25, 2 + 1) = Gamma(65, 3).
# Day 2:
# Observed data k2 = 20.
# Posterior distribution for   after Day 2:
#   ~ Gamma(65 + 20, 3 + 1) = Gamma(85, 4).
# Day 3:
# Observed data k3 = 23.
# Posterior distribution for   after Day 3:
```

```
#   ~ Gamma(85 + 23, 4 + 1) = Gamma(108, 5).
# Day 4:
# Observed data k4 = 27.
# Posterior distribution for   after Day 4:
#   ~ Gamma(108 + 27, 5 + 1) = Gamma(135, 6).
# The posterior distribution of   after Day 4,   ~ Gamma(135, 6), becomes
# the prior for predicting the number of accidents on Day 5.
```

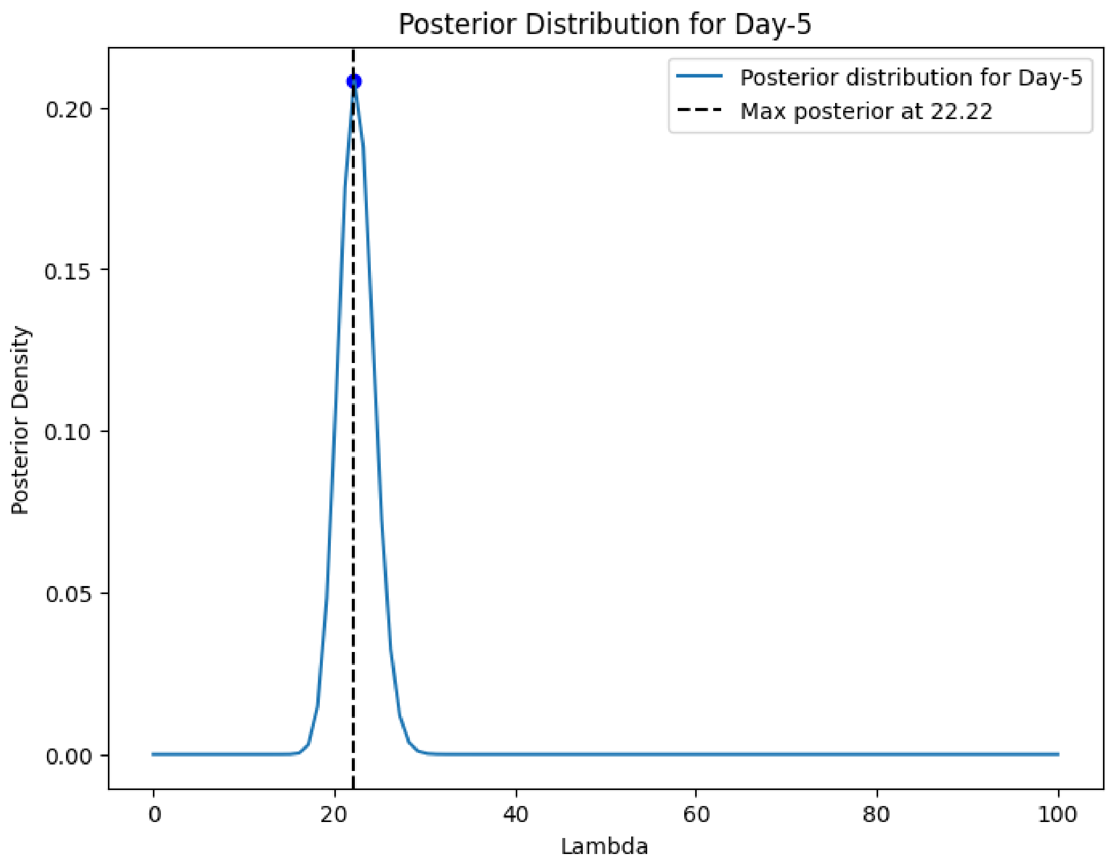
•

Same can be shown with the help of Code

[ ]:
```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import poisson, gamma

def likelihood(y, lam):
    return poisson.pmf(y, lam)

def prior(lam, a, b):
```

```
    return gamma.pdf(lam, a=a, scale=1/b)

def posterior(y, lam, a, b):
    return likelihood(y, lam) * prior(lam, a, b)

acc = np.array([25, 20, 23, 27])
a = 40.0
b = 2.0
x = np.linspace(0, 100, 100)

p = np.zeros(100)
for i in range(100):
    p[i] = prior(x[i], a, b)

for y in acc:
    l = np.zeros(100)
    for i in range(100):
        l[i] = likelihood(y, x[i])
        p[i] = l[i] * p[i]
    p /= np.sum(p)

m = np.argmax(p)
l = x[m]
mp = p[m]
plt.figure(figsize=(8, 6))
plt.plot(x, p, label='Posterior distribution for Day-5', linestyle='-')
plt.axvline(l, color='k', linestyle='--', label=f'Max posterior at {l:.2f}')
plt.scatter(l, mp, color='b')
plt.xlabel('Lambda')
plt.ylabel('Posterior Density')
plt.title('Posterior Distribution for Day-5')
plt.legend()


print("(a). Posterior Distribution for Day-5 will be: Gamma(135.0, 6.0)\n")
plt.show()
```

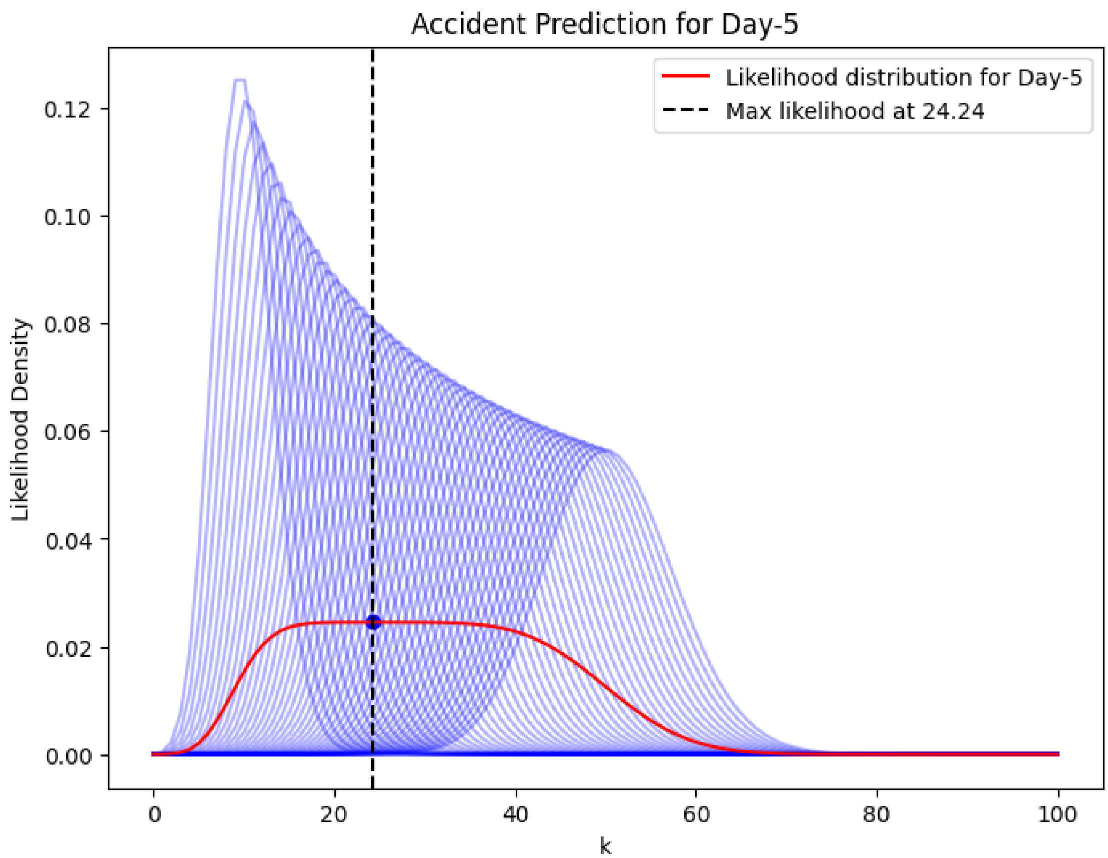
```
(a). Posterior Distribution for Day-5 will be: Gamma(135.0, 6.0)
```



[ ]:
```
a = np.array([25, 20, 23, 27])
k = np.linspace(0, 100, 100)
lamda = np.linspace(10, 50, 50)
p = []

for i in lamda:
    o = [likelihood(int(x), i) for x in k]
    p.append(o)

p = np.array(p)
n = np.mean(p, axis=0)

m = np.argmax(n)
o = k[m]
mp = n[m]
plt.figure(figsize=(8, 6))
for i in lamda:
    plt.plot(k, [likelihood(int(x), i) for x in k], linestyle='-', color='b',␣
 ↪alpha=0.3)
```

```
plt.plot(k, n, label='Likelihood distribution for Day-5', linestyle='-',␣
↪color='r')
plt.axvline(o, color='k', linestyle='--', label=f'Max likelihood at {o:.2f}')
plt.scatter(o, mp, color='b')
plt.xlabel('k')
plt.ylabel('Likelihood Density')
plt.title('Accident Prediction for Day-5')
plt.legend()


print("\n(b).\n")
print("ploting the likelihood distribution for Day - 5:\n")
print("Selected samples of (lambda) is : [10, 50], (as per the observations␣
↪from the graph)\n")
plt.show()
print(f"Estimated accidents on Day -5 will be around: {int(o)}\n\n")
```

```
(b).

ploting the likelihood distribution for Day - 5:

Selected samples of (lambda) is : [10, 50], (as per the observations from the
graph)
```



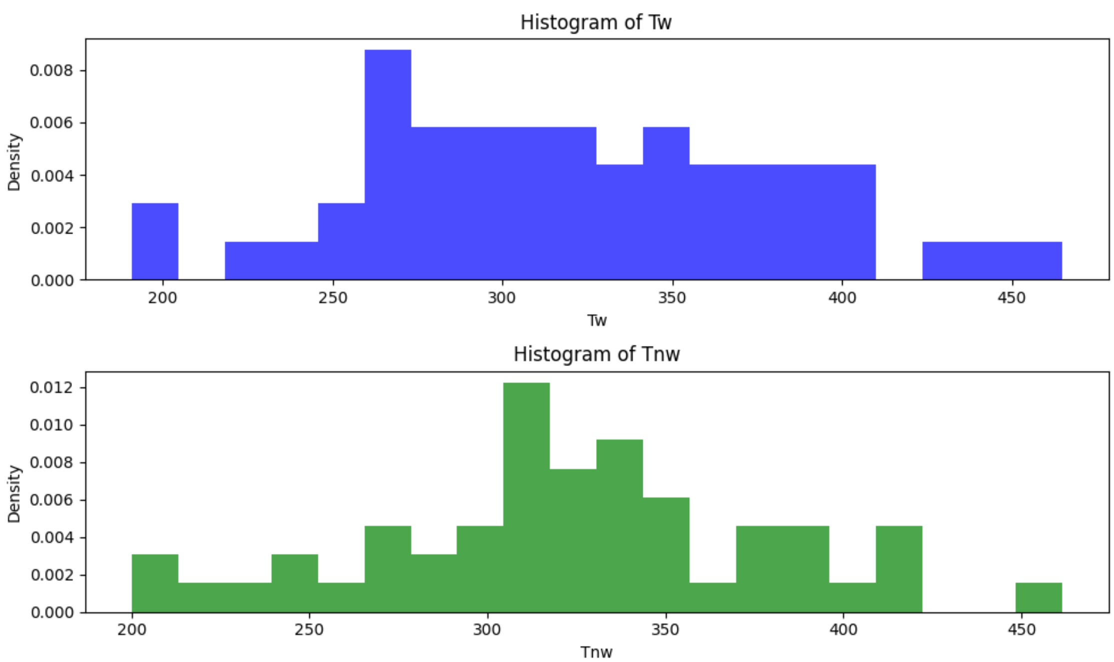

```
Estimated accidents on Day -5 will be around: 24
```

Part 4: Model building in the Bayesian framework

```
[ ]: from scipy.stats import norm, truncnorm

url = "https://raw.githubusercontent.com/yadavhimanshu059/CGS698C/main/notes/
 ↪Module-2/recognition.csv"
data = pd.read_csv(url)
Tw = data['Tw'].values
Tnw = data['Tnw'].values

print(data.head())

plt.figure(figsize=(10, 6))
plt.subplot(2, 1, 1)
plt.hist(Tw, bins=20, density=True, color='blue', alpha=0.7)
plt.title('Histogram of Tw')
```

```
plt.xlabel('Tw')
plt.ylabel('Density')

plt.subplot(2, 1, 2)
plt.hist(Tnw, bins=20, density=True, color='green', alpha=0.7)
plt.title('Histogram of Tnw')
plt.xlabel('Tnw')
plt.ylabel('Density')

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

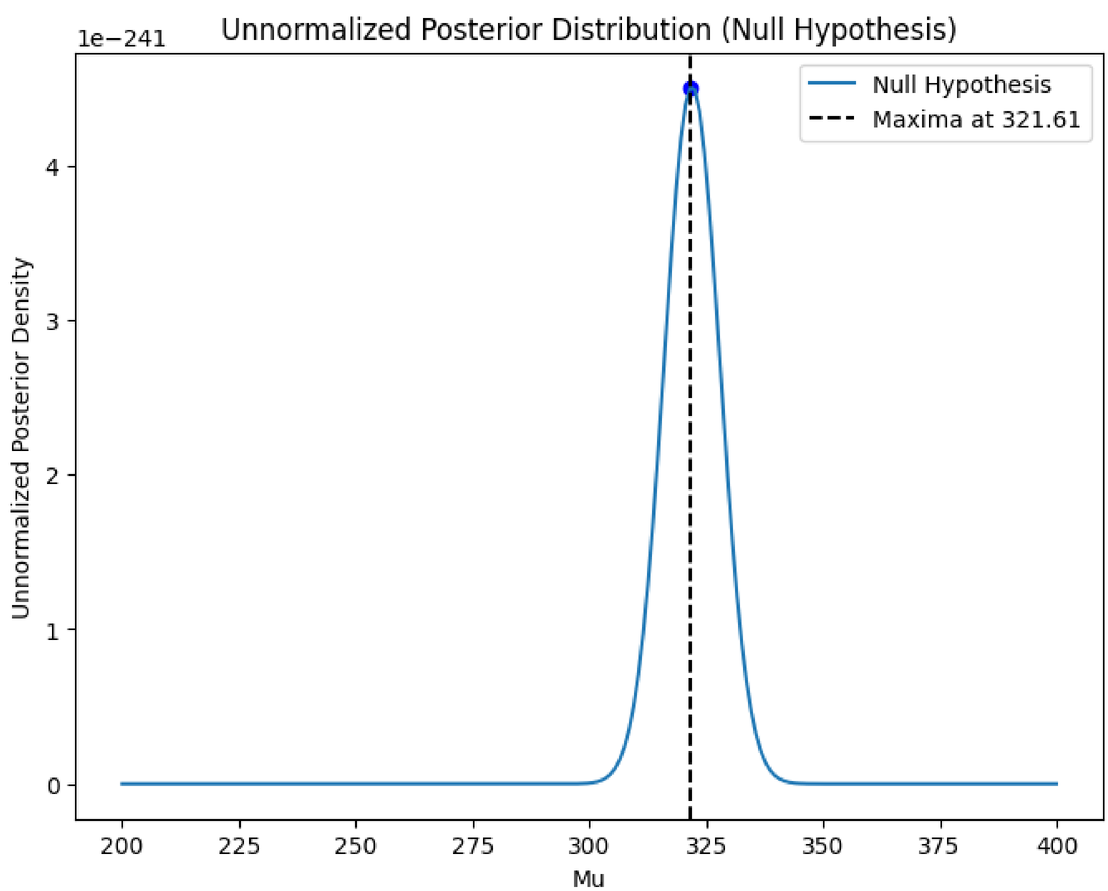
```
   Unnamed: 0          Tw         Tnw
0           1  285.077952  296.806019
1           2  267.518382  280.115725
2           3  289.920350  310.441680
3           4  399.067408  324.827633
4           5  359.988353  373.815164
```



4.5.1 Graph the unnormalized posterior distribution of   for the Null hypothesis model.

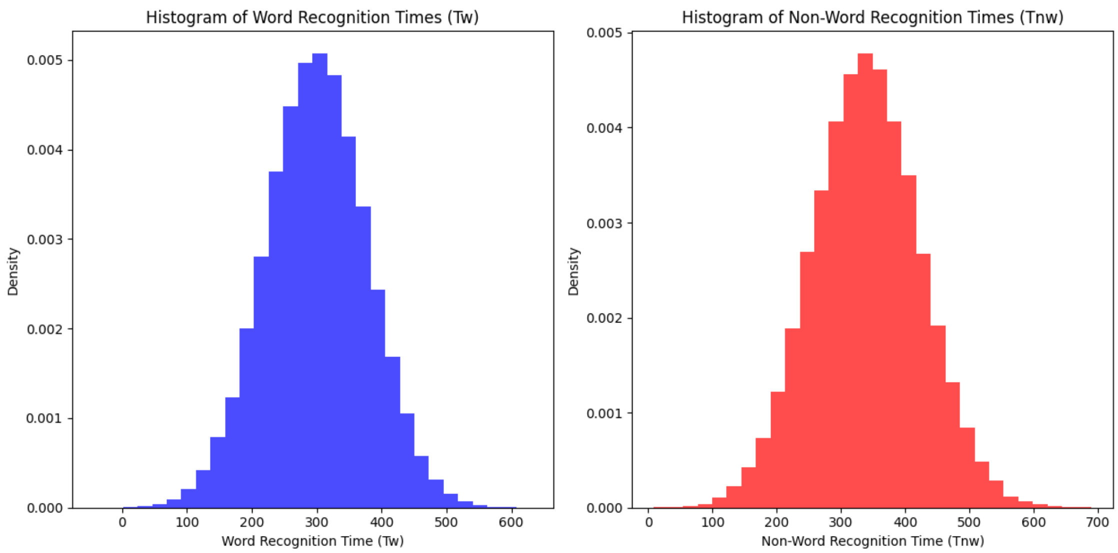
[ ]:
```
url = "https://raw.githubusercontent.com/yadavhimanshu059/CGS698C/main/notes/
↪Module-2/recognition.csv"
data = pd.read_csv(url)
data = data.iloc[:, 1:]
```

```
def like_est(mu, sig, delt):
    return (np.prod(norm.pdf(data["Tw"], mu, sig)) * np.prod(norm.
 ↪pdf(data["Tnw"], mu + delt, sig)))

x_lab = np.linspace(200, 400, 200)
sig = 60.0
delt = 0.0
y_post_null = [(norm.pdf(x, 300, 50) * like_est(x, sig, delt)) for x in x_lab]

plt.figure(figsize=(8, 6))
plt.plot(x_lab, y_post_null, label='Null Hypothesis', linestyle='-')
max_idx = np.argmax(y_post_null)
max_mu = x_lab[max_idx]
max_post = y_post_null[max_idx]
plt.axvline(max_mu, color='k', linestyle='--', label=f'Maxima at {max_mu:.2f}')
plt.scatter(max_mu, max_post, color='b')
plt.xlabel('Mu')
plt.ylabel('Unnormalized Posterior Density')
plt.title('Unnormalized Posterior Distribution (Null Hypothesis)')
plt.legend()
plt.show()
```



4.5.2 Generate prior predictions from the lexical-access model

[ ]:
```
m_mean = 300
m_std = 50
d_mean = 0
d_std = 50
s = 60
n = 100000

m_samples = np.random.normal(m_mean, m_std, n)
d_samples = truncnorm(a=0, b=np.inf, loc=d_mean, scale=d_std).rvs(n)

Tw_samples = [np.random.normal(m, s, 1)[0] for m in m_samples]
Tnw_samples = [np.random.normal(m + d, s, 1)[0] for m, d in zip(m_samples,␣
↪d_samples)]

plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)
plt.hist(Tw_samples, bins=30, alpha=0.7, color='blue', density=True)
plt.xlabel('Word Recognition Time (Tw)')
plt.ylabel('Density')
plt.title('Histogram of Word Recognition Times (Tw)')

plt.subplot(1, 2, 2)
plt.hist(Tnw_samples, bins=30, alpha=0.7, color='red', density=True)
plt.xlabel('Non-Word Recognition Time (Tnw)')
plt.ylabel('Density')
plt.title('Histogram of Non-Word Recognition Times (Tnw)')

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

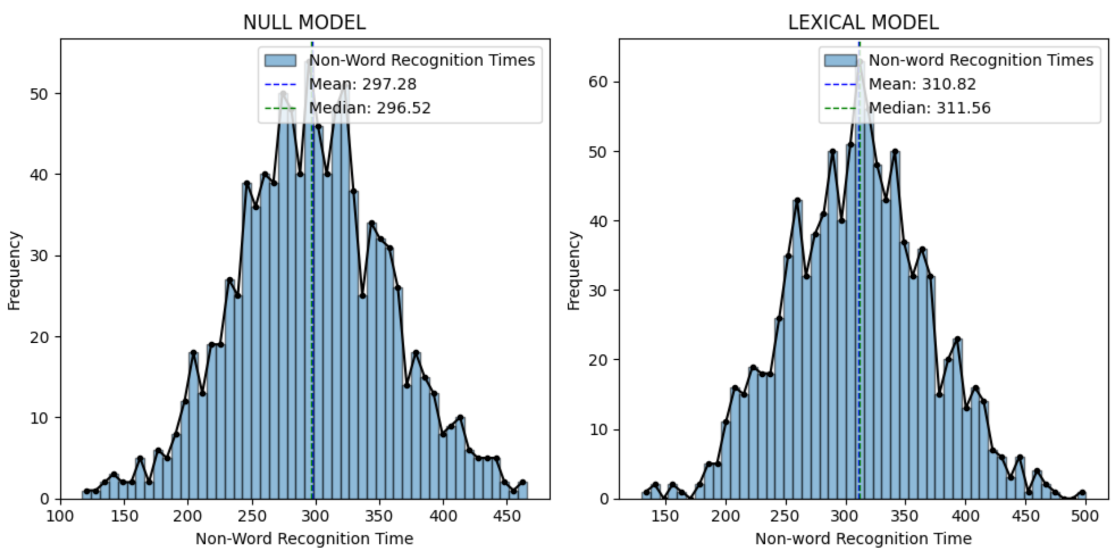


4.5.3: Compare the Prior Predictions of the Null Hypothesis Model and the Lexical Access Model

[ ]:
```python
print("Null model (delta = 0) : It follows the word recognition times' "
      "histogram for both 'Tw' and 'Tnw'.")
print("Thus, both models use the same prior for word recognition times.")
print("For non-word recognition times - \n")
mw = np.mean(word_recognition_times)
mdw = np.median(word_recognition_times)
mnw = np.mean(nonword_recognition_times)
mdnw = np.median(nonword_recognition_times)
fig, axs = plt.subplots(nrows=1, ncols=2, figsize=(10, 5))
cts, bns, patches = axs[0].hist(word_recognition_times, bins=50, alpha=0.5, label='Non-Word Recognition Times', edgecolor='black')
bc = 0.5 * (bns[:-1] + bns[1:])
axs[0].plot(bc, cts, linestyle='-', marker='.', color='k')
axs[0].axvline(mw, color='blue', linestyle='dashed', linewidth=1, label=f'Mean: {mw:.2f}')
axs[0].axvline(mdw, color='green', linestyle='dashed', linewidth=1, label=f'Median: {mdw:.2f}')
axs[0].set_xlabel('Non-Word Recognition Time')
axs[0].set_ylabel('Frequency')
axs[0].set_title('NULL MODEL')
axs[0].legend()
cts, bns, patches = axs[1].hist(nonword_recognition_times, bins=50, alpha=0.5, label='Non-word Recognition Times', edgecolor='black')
bc = 0.5 * (bns[:-1] + bns[1:])
axs[1].plot(bc, cts, linestyle='-', marker='.', color='k')
axs[1].axvline(mnw, color='blue', linestyle='dashed', linewidth=1, label=f'Mean: {mnw:.2f}')
```

```
axs[1].axvline(mdnw, color='green', linestyle='dashed', linewidth=1,␣
 ↪label=f'Median: {mdnw:.2f}')
axs[1].set_xlabel('Non-word Recognition Time')
axs[1].set_ylabel('Frequency')
axs[1].set_title('LEXICAL MODEL')
axs[1].legend()
plt.tight_layout()
plt.show()
print("\n")
print(f'NULL MODEL Prior: Mean = {mw:.2f}, Median = {mdw:.2f}')
print(f'LEXICAL MODEL Prior: Mean = {mnw:.2f}, Median = {mdnw:.2f}')
print("Comparison conclusion: Lexical Hypothesis suggests longer non-word␣
 ↪recognition times compared to Null Hypothesis.\n")
```

```
Null model (delta = 0) : It follows the word recognition times'␣histogram for
both 'Tw' and 'Tnw'.
Thus, both models use the same prior for word recognition times.
For non-word recognition times -
```



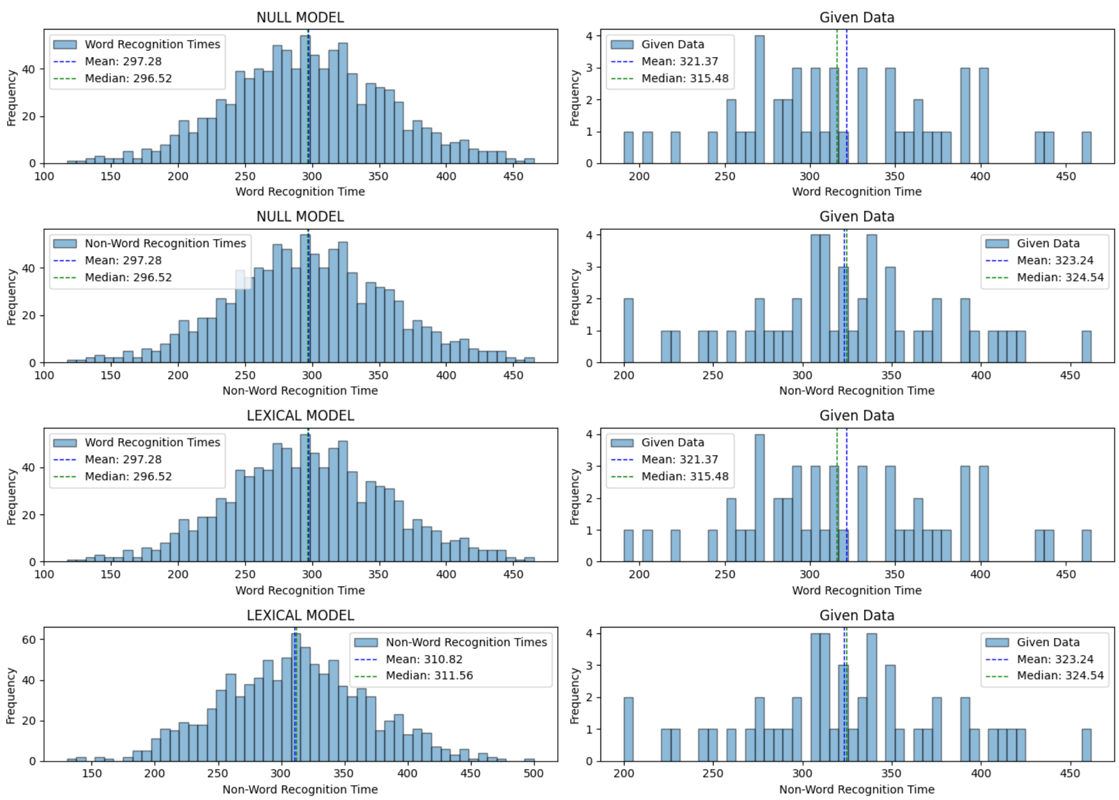

```
NULL MODEL Prior: Mean = 297.28, Median = 296.52
LEXICAL MODEL Prior: Mean = 310.82, Median = 311.56
Comparison conclusion: Lexical Hypothesis suggests longer non-word recognition
times compared to Null Hypothesis.
```

4.5.4 Compare the prior predictions of each model against the observed data Tw and Tnw. Which model seems more consistent with the data?

[ ]:
```
fig, axs = plt.subplots(nrows=4, ncols=2, figsize=(14, 10))
cts, bns, patches = axs[0, 0].hist(word_recognition_times, bins=50, alpha=0.5,␣
 ↪label='Word Recognition Times', edgecolor='black')
axs[0, 0].axvline(mw, color='blue', linestyle='dashed', linewidth=1,␣
 ↪label=f'Mean: {mw:.2f}')
axs[0, 0].axvline(mdw, color='green', linestyle='dashed', linewidth=1,␣
 ↪label=f'Median: {mdw:.2f}')
axs[0, 0].set_xlabel('Word Recognition Time')
axs[0, 0].set_ylabel('Frequency')
axs[0, 0].set_title('NULL MODEL')
axs[0, 0].legend()
cts, bns, patches = axs[0, 1].hist(dat["Tw"], bins=50, alpha=0.5, label='Given␣
 ↪Data', edgecolor='black')
axs[0, 1].axvline(np.mean(dat["Tw"]), color='blue', linestyle='dashed',␣
 ↪linewidth=1, label=f'Mean: {np.mean(dat["Tw"]):.2f}')
axs[0, 1].axvline(np.median(dat["Tw"]), color='green', linestyle='dashed',␣
 ↪linewidth=1, label=f'Median: {np.median(dat["Tw"]):.2f}')
axs[0, 1].set_xlabel('Word Recognition Time')
axs[0, 1].set_ylabel('Frequency')
axs[0, 1].set_title('Given Data')
axs[0, 1].legend()
cts, bns, patches = axs[1, 0].hist(word_recognition_times, bins=50, alpha=0.5,␣
 ↪label='Non-Word Recognition Times', edgecolor='black')
axs[1, 0].axvline(mw, color='blue', linestyle='dashed', linewidth=1,␣
 ↪label=f'Mean: {mw:.2f}')
axs[1, 0].axvline(mdw, color='green', linestyle='dashed', linewidth=1,␣
 ↪label=f'Median: {mdw:.2f}')
axs[1, 0].set_xlabel('Non-Word Recognition Time')
axs[1, 0].set_ylabel('Frequency')
axs[1, 0].set_title('NULL MODEL')
axs[1, 0].legend()
cts, bns, patches = axs[1, 1].hist(dat["Tnw"], bins=50, alpha=0.5, label='Given␣
 ↪Data', edgecolor='black')
axs[1, 1].axvline(np.mean(dat["Tnw"]), color='blue', linestyle='dashed',␣
 ↪linewidth=1, label=f'Mean: {np.mean(dat["Tnw"]):.2f}')
axs[1, 1].axvline(np.median(dat["Tnw"]), color='green', linestyle='dashed',␣
 ↪linewidth=1, label=f'Median: {np.median(dat["Tnw"]):.2f}')
axs[1, 1].set_xlabel('Non-Word Recognition Time')
axs[1, 1].set_ylabel('Frequency')
axs[1, 1].set_title('Given Data')
axs[1, 1].legend()
# For LEXICAL MODEL:
cts, bns, patches = axs[2, 0].hist(word_recognition_times, bins=50, alpha=0.5,␣
 ↪label='Word Recognition Times', edgecolor='black')
```

```
axs[2, 0].axvline(mw, color='blue', linestyle='dashed', linewidth=1, label=f'Mean: {mw:.2f}')
axs[2, 0].axvline(mdw, color='green', linestyle='dashed', linewidth=1, label=f'Median: {mdw:.2f}')
axs[2, 0].set_xlabel('Word Recognition Time')
axs[2, 0].set_ylabel('Frequency')
axs[2, 0].set_title('LEXICAL MODEL')
axs[2, 0].legend()
cts, bns, patches = axs[2, 1].hist(dat["Tw"], bins=50, alpha=0.5, label='Given Data', edgecolor='black')
axs[2, 1].axvline(np.mean(dat["Tw"]), color='blue', linestyle='dashed', linewidth=1, label=f'Mean: {np.mean(dat["Tw"]):.2f}')
axs[2, 1].axvline(np.median(dat["Tw"]), color='green', linestyle='dashed', linewidth=1, label=f'Median: {np.median(dat["Tw"]):.2f}')
axs[2, 1].set_xlabel('Word Recognition Time')
axs[2, 1].set_ylabel('Frequency')
axs[2, 1].set_title('Given Data')
axs[2, 1].legend()
cts, bns, patches = axs[3, 0].hist(nonword_recognition_times, bins=50, alpha=0.5, label='Non-Word Recognition Times', edgecolor='black')
axs[3, 0].axvline(mnw, color='blue', linestyle='dashed', linewidth=1, label=f'Mean: {mnw:.2f}')
axs[3, 0].axvline(mdnw, color='green', linestyle='dashed', linewidth=1, label=f'Median: {mdnw:.2f}')
axs[3, 0].set_xlabel('Non-Word Recognition Time')
axs[3, 0].set_ylabel('Frequency')
axs[3, 0].set_title('LEXICAL MODEL')
axs[3, 0].legend()
cts, bns, patches = axs[3, 1].hist(dat["Tnw"], bins=50, alpha=0.5, label='Given Data', edgecolor='black')
axs[3, 1].axvline(np.mean(dat["Tnw"]), color='blue', linestyle='dashed', linewidth=1, label=f'Mean: {np.mean(dat["Tnw"]):.2f}')
axs[3, 1].axvline(np.median(dat["Tnw"]), color='green', linestyle='dashed', linewidth=1, label=f'Median: {np.median(dat["Tnw"]):.2f}')
axs[3, 1].set_xlabel('Non-Word Recognition Time')
axs[3, 1].set_ylabel('Frequency')
axs[3, 1].set_title('Given Data')
axs[3, 1].legend()
plt.tight_layout()
plt.show()
print("Comparison of Prior Models Against Data:\n")

print("Absolute Error in Prior Model for Mean Word-Recognition in Null Model: ", (np.abs(mw - np.mean(dat["Tw"]))/np.mean(dat["Tw"]))*100 )
print("Absolute Error in Prior Model for Mean Non-Word-Recognition in Null Model: ", (np.abs(mw - np.mean(dat["Tnw"]))/np.mean(dat["Tnw"]))*100 )
```

```
print("\n")
print("Absolute Error in Prior Model for Mean Word-Recognition in Lexical Model: ", (np.abs(mw - np.mean(dat["Tw"]))/np.mean(dat["Tw"]))*100 )
print("Absolute Error in Prior Model for Mean Non-Word-Recognition in Lexical Model: ", (np.abs(mnw - np.mean(dat["Tnw"]))/np.mean(dat["Tnw"]))*100 )
print("\n")
print("Conclusion: The Lexical Model shows a lower absolute error, indicating a better fit.\n")
```




```
Comparison of Prior Models Against Data:

Absolute Error in Prior Model for Mean Word-Recognition in Null Model:
7.4959185692685395
Absolute Error in Prior Model for Mean Non-Word-Recognition in Null Model:
8.02942317485629


Absolute Error in Prior Model for Mean Word-Recognition in Lexical Model:
7.4959185692685395
Absolute Error in Prior Model for Mean Non-Word-Recognition in Lexical Model:
3.8429150966746644
```

Conclusion: The Lexical Model shows a lower absolute error, indicating a better fit.

4.5.5 Graph the unnormalized posterior distribution of   for the lexical-access model

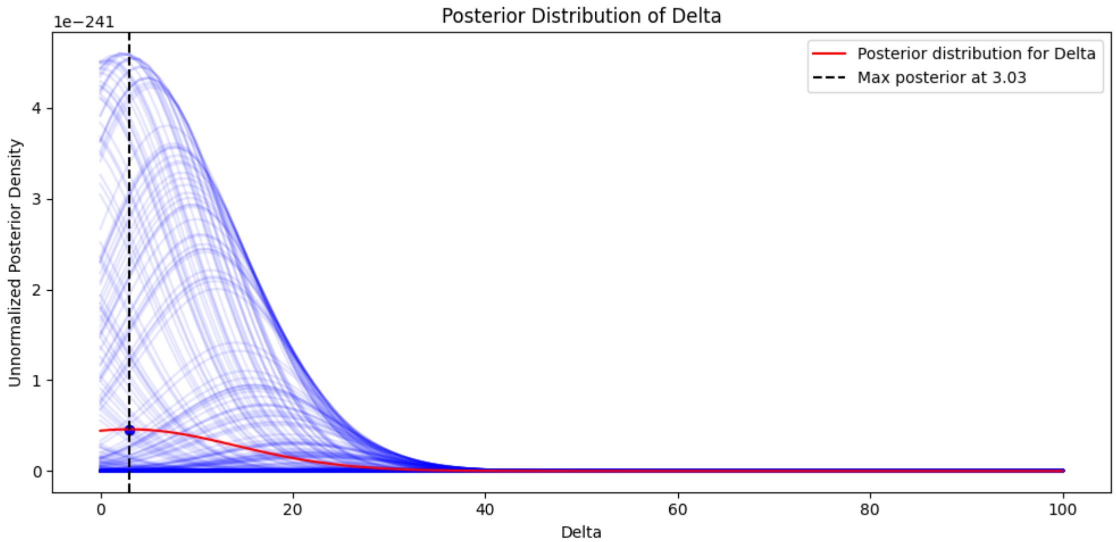
[ ]:
```
!pip install emcee
!conda install -c conda-forge emcee
```

```
Requirement already satisfied: emcee in /usr/local/lib/python3.10/dist-packages
(3.1.6)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages
(from emcee) (1.25.2)
/bin/bash: line 1: conda: command not found
```

[ ]:
```
import emcee
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import emcee
from scipy.stats import norm

url = "https://raw.githubusercontent.com/yadavhimanshu059/CGS698C/main/notes/
↪Module-2/recognition.csv"
data = pd.read_csv(url)
Tw = data['Tw'].values
Tnw = data['Tnw'].values

def log_prob(params, Tw, Tnw):
    m, d = params
    if d < 0:
        return -np.inf
    mp = np.log(np.exp(-(m - 300)**2 / (2 * 50**2)))
    dp = np.log(np.exp(-(d - 0)**2 / (2 * 50**2)))
    Twl = -0.5 * np.sum((Tw - m)**2 / 60**2 + np.log(2 * np.pi * 60**2))
    Tnwl = -0.5 * np.sum((Tnw - (m + d))**2 / 60**2 + np.log(2 * np.pi * 60**2))
    return mp + dp + Twl + Tnwl

ini = [300, 50]
nd = 2
nw = 50
ns = 2000
nb = 1000

p = ini + 1e-4 * np.random.randn(nw, nd)

sampler = emcee.EnsembleSampler(nw, nd, log_prob, args=(Tw, Tnw))
```

```
sampler.run_mcmc(p, ns, progress=True)

s = sampler.get_chain(discard=nb, flat=True)

ds = s[:, 1]

# Calculate posterior distribution
d_label = np.linspace(0, 100, 100)
mu_samples_ = np.random.normal(300, 50, 500)
posterior_distribution = []

fig, axes = plt.subplots(nrows=1, ncols=1, figsize=(10, 5))

for mu_ in mu_samples_:
    y_posterior = [np.exp(log_prob([mu_, delta], Tw, Tnw)) * np.abs(norm.
 ↪pdf(delta, 0, 50)) for delta in d_label]
    plt.plot(d_label, y_posterior, linestyle='-', color='b', alpha=0.1)
    posterior_distribution.append(y_posterior)

posterior_distribution = np.array(posterior_distribution)
mean_posterior = np.mean(posterior_distribution, axis=0)
max_index = np.argmax(mean_posterior)
max_delta = d_label[max_index]
max_posterior = mean_posterior[max_index]

plt.plot(d_label, mean_posterior, label='Posterior distribution for Delta',␣
 ↪linestyle='-', color='r')
plt.axvline(max_delta, color='k', linestyle='--', label=f'Max posterior at␣
 ↪{max_delta:.2f}')
plt.scatter(max_delta, max_posterior, color='b')
plt.xlabel('Delta')
plt.ylabel('Unnormalized Posterior Density')
plt.title('Posterior Distribution of Delta')
plt.legend()

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

```
100%|          | 2000/2000 [00:06<00:00, 318.45it/s]
```


Posterior Distribution of Delta
1e−241
Unnormalized Posterior Density
Delta
Posterior distribution for Delta
Max posterior at 3.03
0
1
2
3
4
0
20
40
60
80
100