Assignment-2

Part 1:

June 14, 2024

1.1

1.1 Estimate the Posterior Density

Given the data: $y = 7$ and the marginal likelihood: $\int L(\theta|y)p(\theta)\,d\theta = \frac{1}{11}$

We can use Bayes' rule to calculate the posterior density $p(\theta|y)$ for each value of θ.

(a) For $\theta = 0.75$:

$$p(\theta = 0.75|y) = \frac{L(\theta = 0.75|y) \cdot p(\theta = 0.75)}{\int L(\theta|y)p(\theta)\,d\theta}$$
$$= \frac{\frac{10!}{7!(10-7)!} \cdot (0.75)^7 \cdot (1-0.75)^{10-7} \cdot 1}{\frac{1}{11}}$$
$$= 11 \cdot \frac{10!}{7!(10-7)!} \cdot (0.75)^7 \cdot (1-0.75)^{10-7}$$
$$= 2.753105$$

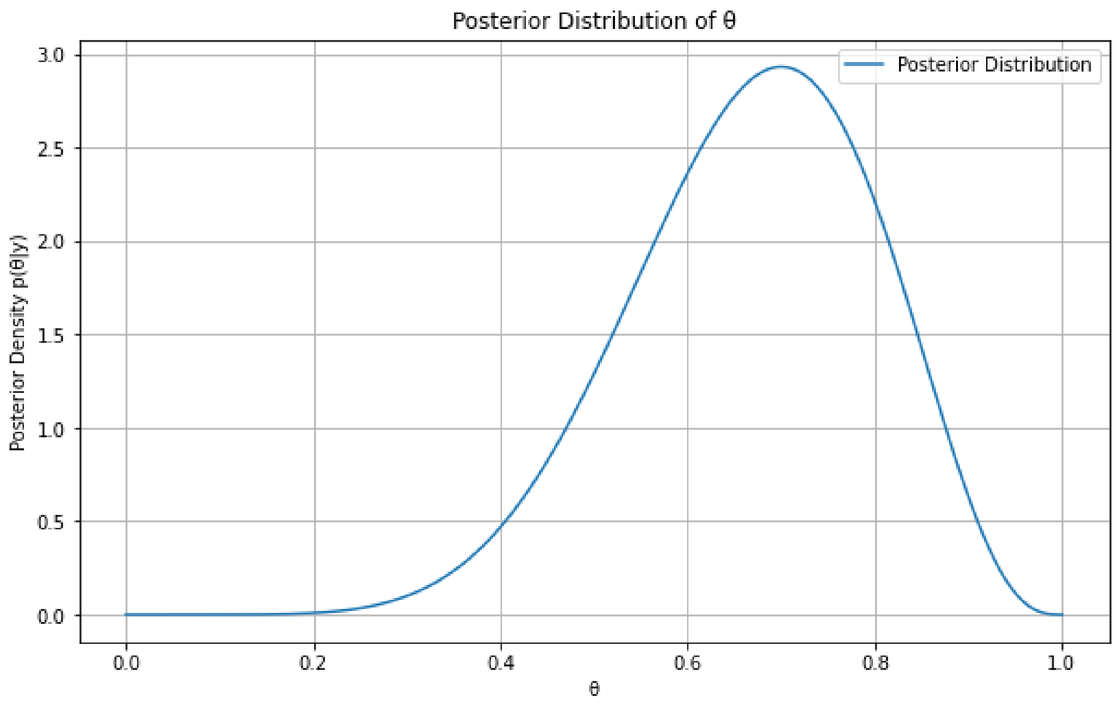
(b) For $\theta = 0.25$:

$$p(\theta = 0.25|y) = \frac{L(\theta = 0.25|y) \cdot p(\theta = 0.25)}{\int L(\theta|y)p(\theta)\,d\theta}$$
$$= \frac{\frac{10!}{7!(10-7)!} \cdot (0.25)^7 \cdot (1-0.25)^{10-7} \cdot 1}{\frac{1}{11}}$$
$$= 11 \cdot \frac{10!}{7!(10-7)!} \cdot (0.25)^7 \cdot (1-0.25)^{10-7}$$
$$= 0.0339889526$$

(c) For $\theta = 1$:

$$p(\theta = 1|y) = \frac{L(\theta = 1|y) \cdot p(\theta = 1)}{\int L(\theta|y)p(\theta)\,d\theta}$$
$$= \frac{\frac{10!}{7!(10-7)!} \cdot (1)^7 \cdot (1-1)^{10-7} \cdot 1}{\frac{1}{11}}$$
$$= 11 \cdot \frac{10!}{7!(10-7)!} \cdot (1)^7 \cdot (1-1)^{10-7}$$
$$= 0$$

1.2 [1]:
```
import numpy as np
import matplotlib.pyplot as plt

# Function to calculate the posterior density
def posterior_density(theta, y=7):
    if 0 <= theta <= 1:
        return 1320 * (theta**7) * ((1 - theta)**3)
    else:
        return 0

# Function to calculate the likelihood
def likelihood(theta, y=7):
    if 0 <= theta <= 1:
        return 120 * (theta**7) * ((1 - theta)**3)
    else:
        return 0

# Create a vector of   values
theta_values = np.linspace(0, 1, 1000)
posterior_values = np.array([posterior_density(theta) for theta in␣
 ↪theta_values])
likelihood_values = np.array([likelihood(theta) for theta in theta_values])
prior_values = np.ones_like(theta_values)  # Uniform prior between 0 and 1

# Part 1.2: Plot the posterior distribution
plt.figure(figsize=(10, 6))
plt.plot(theta_values, posterior_values, label='Posterior Distribution')
plt.title('Posterior Distribution of  ')
plt.xlabel(' ')
plt.ylabel('Posterior Density p( |y)')
plt.legend()
plt.grid(True)
plt.show()
```



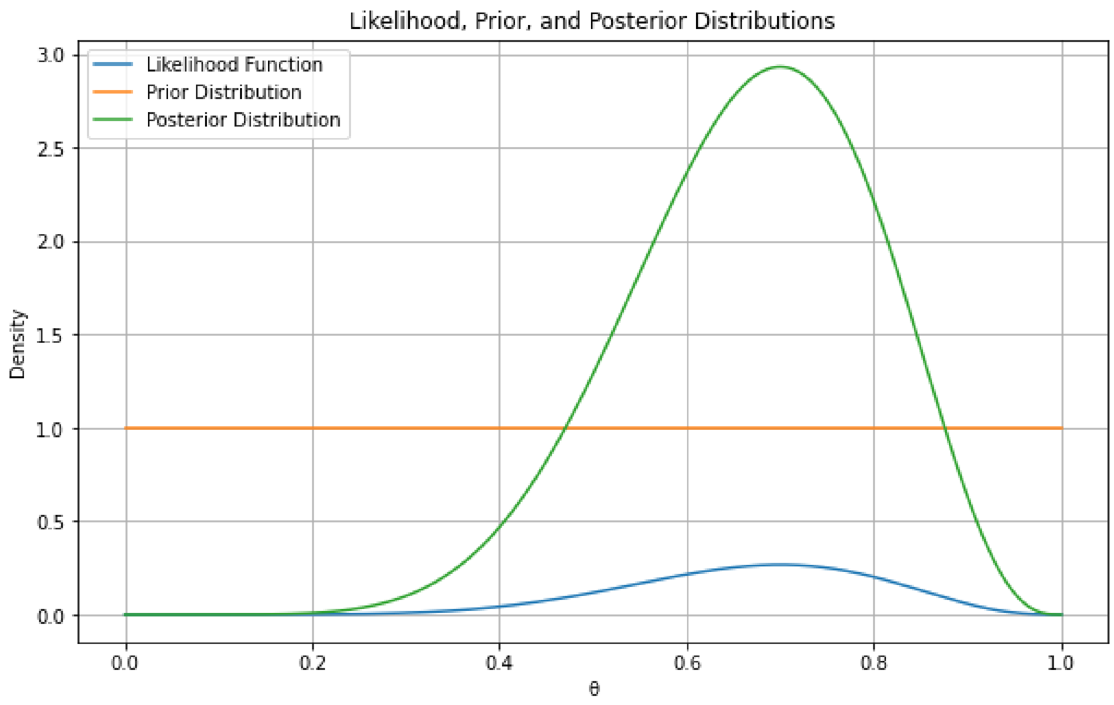
1.3 [2]:

```
# Part 1.3: Find the value of   with the maximum posterior density
max_theta = theta_values[np.argmax(posterior_values)]
max_density = np.max(posterior_values)
print(f"Value of   with the maximum posterior density: {max_theta}")
print(f"Maximum posterior density: {max_density}")
```

```
Value of   with the maximum posterior density: 0.6996996996996997
Maximum posterior density: 2.9351009522941216
```

1.4 [3]:

```
# Part 1.4: Compare the likelihood, prior, and posterior distributions
plt.figure(figsize=(10, 6))
plt.plot(theta_values, likelihood_values, label='Likelihood Function')
plt.plot(theta_values, prior_values, label='Prior Distribution')
plt.plot(theta_values, posterior_values, label='Posterior Distribution')
plt.title('Likelihood, Prior, and Posterior Distributions')
plt.xlabel(' ')
plt.ylabel('Density')
plt.legend()
plt.grid(True)
plt.show()
```



Part 2:

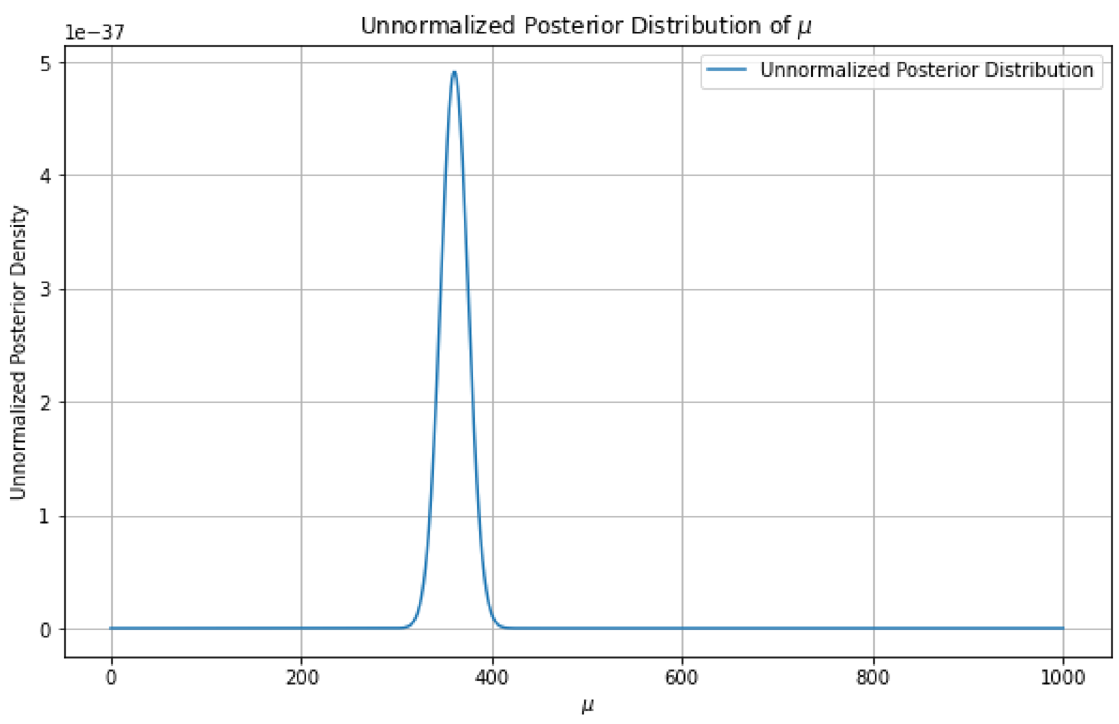
2.1 [4]:
```python
import numpy as np
from scipy.stats import norm

# Given data
y = np.array([300, 270, 390, 450, 500, 290, 680, 450])
n = len(y)
sigma = 50

# Likelihood function
def likelihood(mu, y, sigma):
    return (1 / (sigma * np.sqrt(2 * np.pi))) ** n * np.exp(-np.sum((y - mu) **␣
 ↪2) / (2 * sigma ** 2))

# Prior distribution
def prior(mu):
    return norm.pdf(mu, 250, 25)

# Unnormalized posterior density
def unnormalized_posterior(mu, y, sigma):
    return likelihood(mu, y, sigma) * prior(mu)

# Values of μ to calculate
mu_values_to_calculate = [300, 900, 50]
```

```
posterior_densities = [unnormalized_posterior(mu, y, sigma) for mu in␣
 ↪mu_values_to_calculate]

# Print the results
for mu, density in zip(mu_values_to_calculate, posterior_densities):
    print(f"Unnormalized posterior density for µ = {mu}: {density}")
```

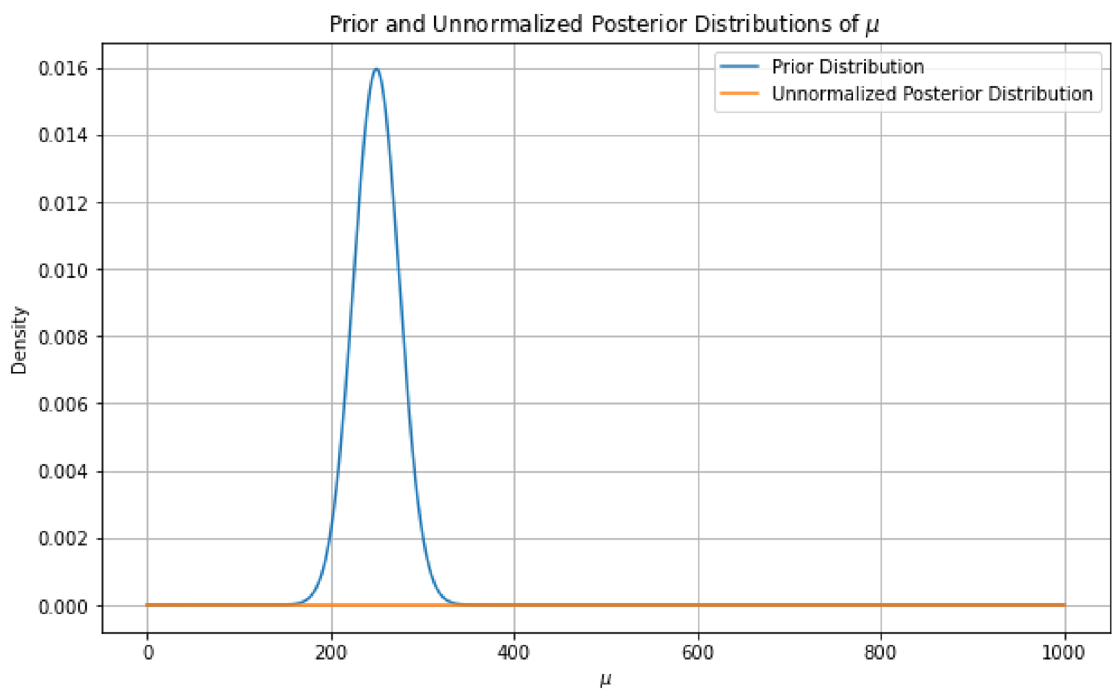
```
Unnormalized posterior density for µ = 300: 6.824247957486406e-41
Unnormalized posterior density for µ = 900: 0.0
Unnormalized posterior density for µ = 50: 9.691373559300647e-138
```

2.2 [5]:

```
import matplotlib.pyplot as plt

# Range of µ values for the graph
mu_values = np.linspace(0, 1000, 1000)
posterior_values = np.array([unnormalized_posterior(mu, y, sigma) for mu in␣
 ↪mu_values])

# Plot the unnormalized posterior distribution
plt.figure(figsize=(10, 6))
plt.plot(mu_values, posterior_values, label='Unnormalized Posterior␣
 ↪Distribution')
plt.title('Unnormalized Posterior Distribution of $\mu$')
plt.xlabel('$\mu$')
plt.ylabel('Unnormalized Posterior Density')
plt.legend()
plt.grid(True)
plt.show()
```

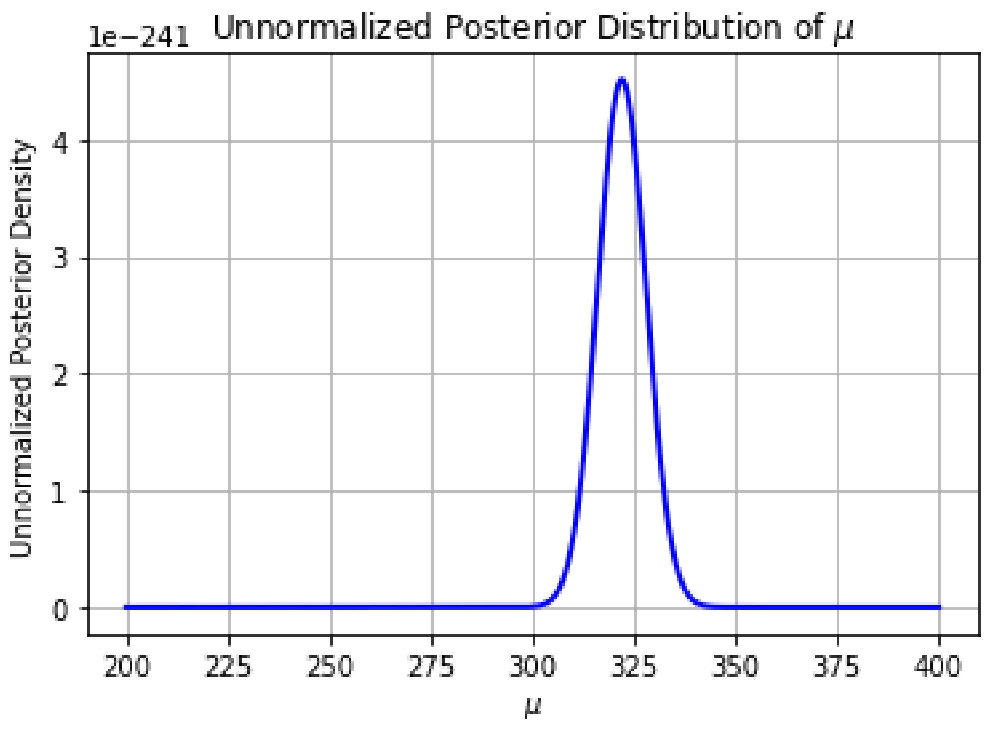


2.3 [6]:
```
# Prior distribution values
prior_values = np.array([prior(mu) for mu in mu_values])

# Plot the prior and unnormalized posterior distributions
plt.figure(figsize=(10, 6))
plt.plot(mu_values, prior_values, label='Prior Distribution')
plt.plot(mu_values, posterior_values, label='Unnormalized Posterior␣
 ↪Distribution')
plt.title('Prior and Unnormalized Posterior Distributions of $\mu$')
plt.xlabel('$\mu$')
plt.ylabel('Density')
plt.legend()
plt.grid(True)
plt.show()
```



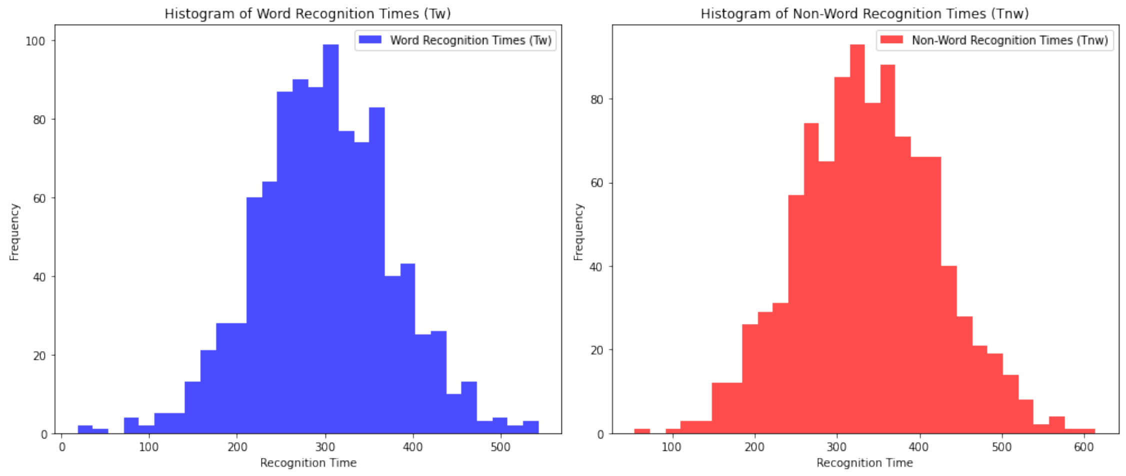
Part 4:

4.5.1 [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy.stats import norm

# Load the data from the URL
url = "https://raw.githubusercontent.com/yadavhimanshu059/CGS698C/main/notes/
 ↪Module-2/recognition.csv"
data = pd.read_csv(url)

# Extract Tw and Tnw columns
Tw = data['Tw'].values
Tnw = data['Tnw'].values

# Define parameters
sigma = 60
mu_prior_mean = 300
mu_prior_sd = 50

# Define a sequence of mu values to evaluate the posterior
mu_values = np.linspace(200, 400, 1000)

# Define likelihood functions
def likelihood_Tw(mu):
    return np.prod(norm.pdf(Tw, loc=mu, scale=sigma))

def likelihood_Tnw(mu):
    return np.prod(norm.pdf(Tnw, loc=mu, scale=sigma))

# Define prior for mu
def prior_mu(mu):
    return norm.pdf(mu, loc=mu_prior_mean, scale=mu_prior_sd)

# Calculate unnormalized posterior
posterior_unnormalized = np.array([likelihood_Tw(mu) * likelihood_Tnw(mu) *␣
  ↪prior_mu(mu) for mu in mu_values])

# Plot the unnormalized posterior distribution
plt.plot(mu_values, posterior_unnormalized, lw=2, color='blue')
plt.xlabel(r'$\mu$')
plt.ylabel('Unnormalized Posterior Density')
plt.title('Unnormalized Posterior Distribution of $\mu$')
plt.grid(True)
plt.show()
```



4.5.2 [2]:
```
import numpy as np
import matplotlib.pyplot as plt
```

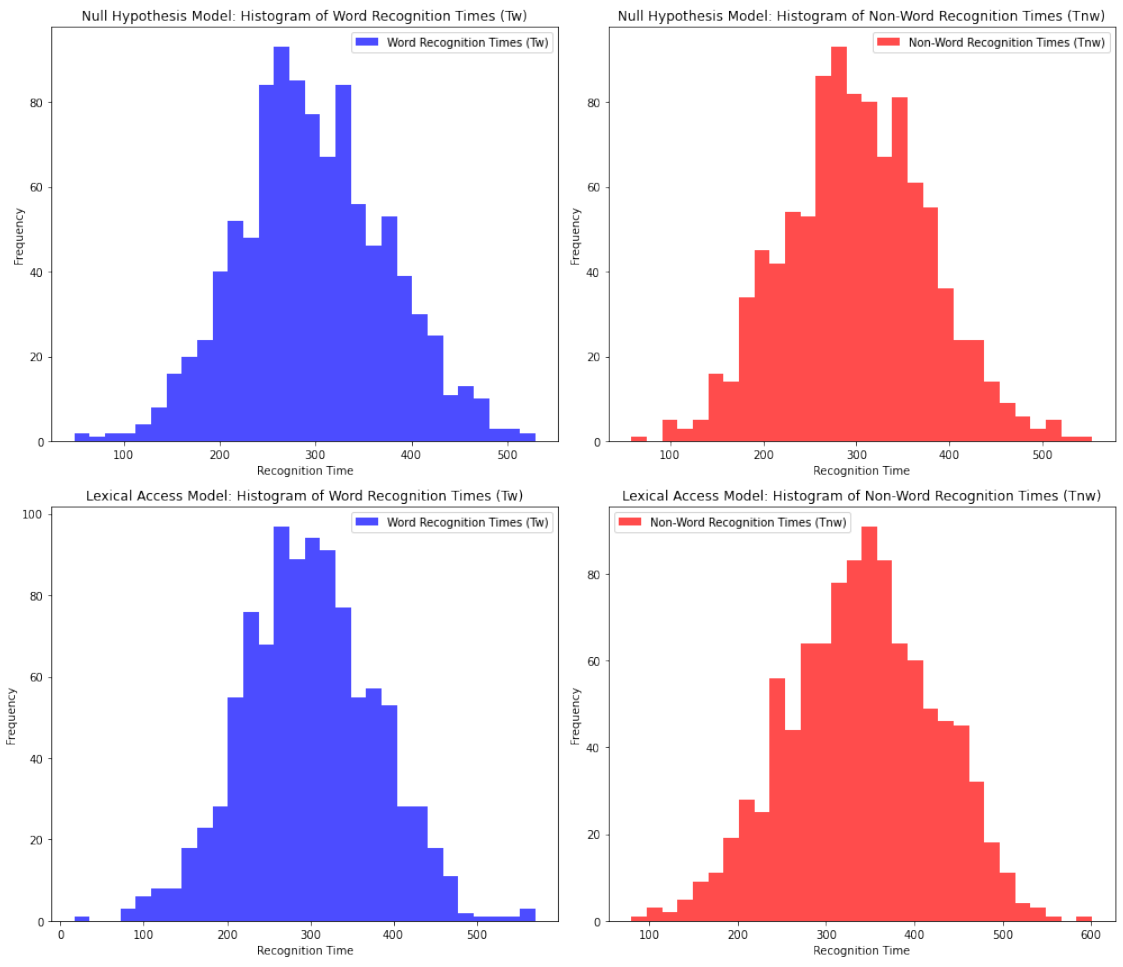
```python
from scipy.stats import norm, truncnorm

# Set the parameters
mu_prior_mean = 300
mu_prior_sd = 50
sigma = 60
delta_prior_mean = 0
delta_prior_sd = 50
num_samples = 1000  # Number of samples to draw

# Function to draw samples from truncated normal distribution
def truncated_normal(mean, sd, lower, upper, size):
    a, b = (lower - mean) / sd, (upper - mean) / sd
    return truncnorm.rvs(a, b, loc=mean, scale=sd, size=size)

# Draw samples for mu from its prior distribution
mu_samples = np.random.normal(mu_prior_mean, mu_prior_sd, num_samples)

# Draw samples for delta from its truncated normal prior distribution
delta_samples = truncated_normal(delta_prior_mean, delta_prior_sd, 0, np.inf,␣
 ↪num_samples)

# Generate word recognition times Tw
Tw_samples = np.random.normal(mu_samples, sigma)

# Generate non-word recognition times Tnw
Tnw_samples = np.random.normal(mu_samples + delta_samples, sigma)

# Plot the histograms of recognition times
plt.figure(figsize=(14, 6))

plt.subplot(1, 2, 1)
plt.hist(Tw_samples, bins=30, color='blue', alpha=0.7, label='Word Recognition␣
 ↪Times (Tw)')
plt.xlabel('Recognition Time')
plt.ylabel('Frequency')
plt.title('Histogram of Word Recognition Times (Tw)')
plt.legend()

plt.subplot(1, 2, 2)
plt.hist(Tnw_samples, bins=30, color='red', alpha=0.7, label='Non-Word␣
 ↪Recognition Times (Tnw)')
plt.xlabel('Recognition Time')
plt.ylabel('Frequency')
plt.title('Histogram of Non-Word Recognition Times (Tnw)')
plt.legend()
plt.tight_layout()
plt.show()
```



4.5.3 [3]:

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm, truncnorm

# Set the parameters
mu_prior_mean = 300
mu_prior_sd = 50
sigma = 60
delta_prior_mean = 0
delta_prior_sd = 50
num_samples = 1000  # Number of samples to draw

# Function to draw samples from truncated normal distribution
def truncated_normal(mean, sd, lower, upper, size):
    a, b = (lower - mean) / sd, (upper - mean) / sd
    return truncnorm.rvs(a, b, loc=mean, scale=sd, size=size)

# Generate prior predictions for the null hypothesis model
mu_samples_null = np.random.normal(mu_prior_mean, mu_prior_sd, num_samples)
Tw_samples_null = np.random.normal(mu_samples_null, sigma)
Tnw_samples_null = np.random.normal(mu_samples_null, sigma)

# Generate prior predictions for the lexical access model
mu_samples_lexical = np.random.normal(mu_prior_mean, mu_prior_sd, num_samples)
delta_samples_lexical = truncated_normal(delta_prior_mean, delta_prior_sd, 0,␣
↪np.inf, num_samples)
Tw_samples_lexical = np.random.normal(mu_samples_lexical, sigma)
```

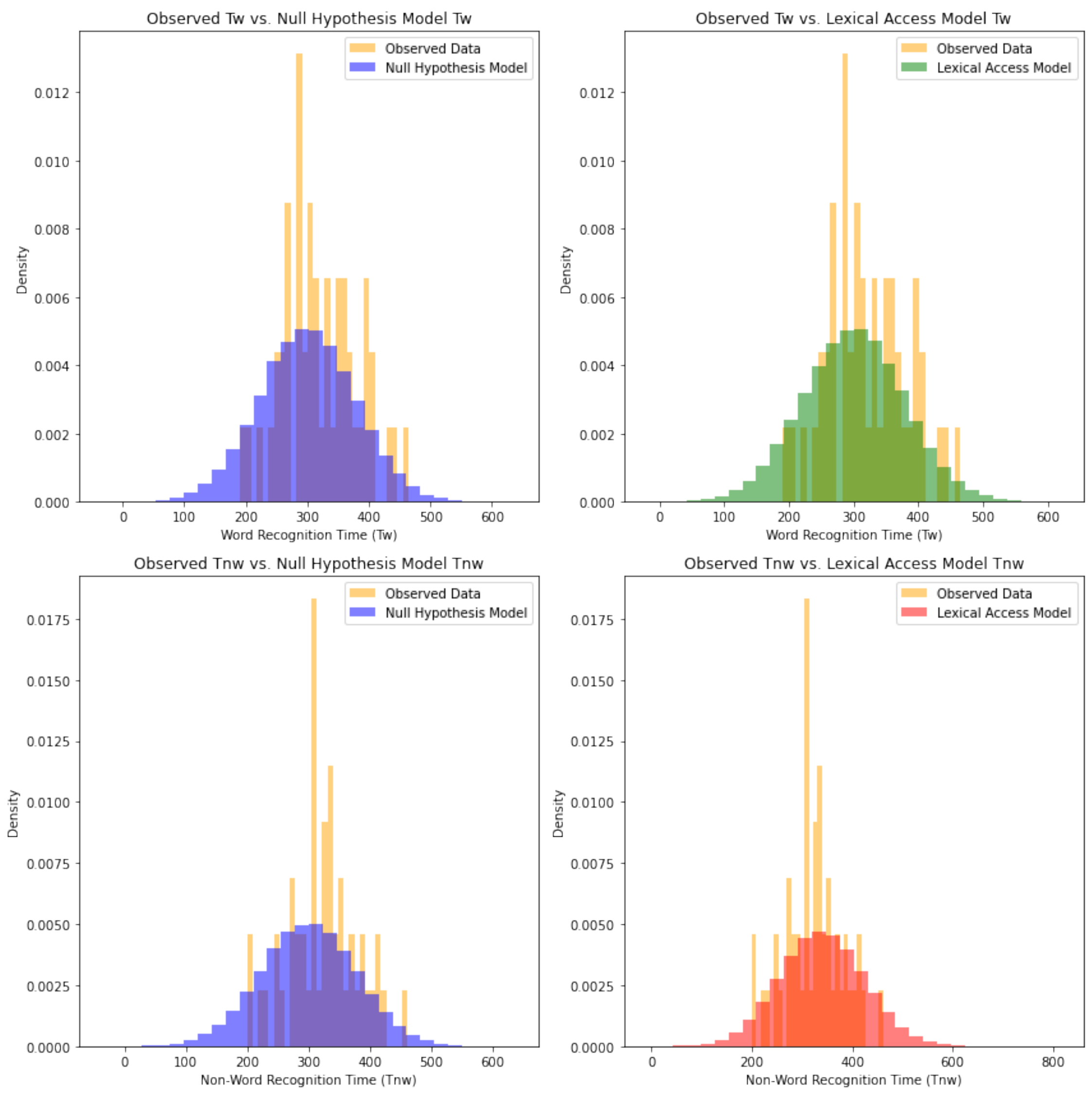
```
Tnw_samples_lexical = np.random.normal(mu_samples_lexical +␣
  ↪delta_samples_lexical, sigma)

# Plot the histograms of recognition times for the null hypothesis model
plt.figure(figsize=(14, 12))

plt.subplot(2, 2, 1)
plt.hist(Tw_samples_null, bins=30, color='blue', alpha=0.7, label='Word␣
  ↪Recognition Times (Tw)')
plt.xlabel('Recognition Time')
plt.ylabel('Frequency')
plt.title('Null Hypothesis Model: Histogram of Word Recognition Times (Tw)')
plt.legend()

plt.subplot(2, 2, 2)
plt.hist(Tnw_samples_null, bins=30, color='red', alpha=0.7, label='Non-Word␣
  ↪Recognition Times (Tnw)')
plt.xlabel('Recognition Time')
plt.ylabel('Frequency')
plt.title('Null Hypothesis Model: Histogram of Non-Word Recognition Times␣
  ↪(Tnw)')
plt.legend()

# Plot the histograms of recognition times for the lexical access model
plt.subplot(2, 2, 3)
plt.hist(Tw_samples_lexical, bins=30, color='blue', alpha=0.7, label='Word␣
  ↪Recognition Times (Tw)')
plt.xlabel('Recognition Time')
plt.ylabel('Frequency')
plt.title('Lexical Access Model: Histogram of Word Recognition Times (Tw)')
plt.legend()

plt.subplot(2, 2, 4)
plt.hist(Tnw_samples_lexical, bins=30, color='red', alpha=0.7, label='Non-Word␣
  ↪Recognition Times (Tnw)')
plt.xlabel('Recognition Time')
plt.ylabel('Frequency')
plt.title('Lexical Access Model: Histogram of Non-Word Recognition Times (Tnw)')
plt.legend()

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



4.5.4 [5]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import truncnorm

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

# Parameters for the models
mu_prior_mean = 300
mu_prior_std = 50
delta_prior_mean = 0
delta_prior_std = 50
```

```
sigma = 60
num_samples = 100000

# Generate prior samples for both models
mu_samples_null = np.random.normal(mu_prior_mean, mu_prior_std, num_samples)
mu_samples_lexical = np.random.normal(mu_prior_mean, mu_prior_std, num_samples)
delta_samples_lexical = truncnorm(a=0, b=np.inf, loc=delta_prior_mean,␣
 ↪scale=delta_prior_std).rvs(num_samples)

# Generate predictions for the Null Hypothesis Model
Tw_samples_null = np.random.normal(mu_samples_null, sigma, num_samples)
Tnw_samples_null = np.random.normal(mu_samples_null, sigma, num_samples)

# Generate predictions for the Lexical Access Model
Tw_samples_lexical = np.random.normal(mu_samples_lexical, sigma, num_samples)
Tnw_samples_lexical = np.random.normal(mu_samples_lexical +␣
 ↪delta_samples_lexical, sigma, num_samples)

# Plot the predictions and observed data
plt.figure(figsize=(12, 12))

# Histogram for Observed Tw vs. Null Hypothesis Model Tw
plt.subplot(2, 2, 1)
plt.hist(Tw_observed, bins=30, alpha=0.5, color='orange', density=True,␣
 ↪label='Observed Data')
plt.hist(Tw_samples_null, bins=30, alpha=0.5, color='blue', density=True,␣
 ↪label='Null Hypothesis Model')
plt.xlabel('Word Recognition Time (Tw)')
plt.ylabel('Density')
plt.legend()
plt.title('Observed Tw vs. Null Hypothesis Model Tw')

# Histogram for Observed Tw vs. Lexical Access Model Tw
plt.subplot(2, 2, 2)
plt.hist(Tw_observed, bins=30, alpha=0.5, color='orange', density=True,␣
 ↪label='Observed Data')
plt.hist(Tw_samples_lexical, bins=30, alpha=0.5, color='green', density=True,␣
 ↪label='Lexical Access Model')
plt.xlabel('Word Recognition Time (Tw)')
plt.ylabel('Density')
plt.legend()
plt.title('Observed Tw vs. Lexical Access Model Tw')

# Histogram for Observed Tnw vs. Null Hypothesis Model Tnw
plt.subplot(2, 2, 3)
```

```python
plt.hist(Tnw_observed, bins=30, alpha=0.5, color='orange', density=True,␣
 ↪label='Observed Data')
plt.hist(Tnw_samples_null, bins=30, alpha=0.5, color='blue', density=True,␣
 ↪label='Null Hypothesis Model')
plt.xlabel('Non-Word Recognition Time (Tnw)')
plt.ylabel('Density')
plt.legend()
plt.title('Observed Tnw vs. Null Hypothesis Model Tnw')

# Histogram for Observed Tnw vs. Lexical Access Model Tnw
plt.subplot(2, 2, 4)
plt.hist(Tnw_observed, bins=30, alpha=0.5, color='orange', density=True,␣
 ↪label='Observed Data')
plt.hist(Tnw_samples_lexical, bins=30, alpha=0.5, color='red', density=True,␣
 ↪label='Lexical Access Model')
plt.xlabel('Non-Word Recognition Time (Tnw)')
plt.ylabel('Density')
plt.legend()
plt.title('Observed Tnw vs. Lexical Access Model Tnw')

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

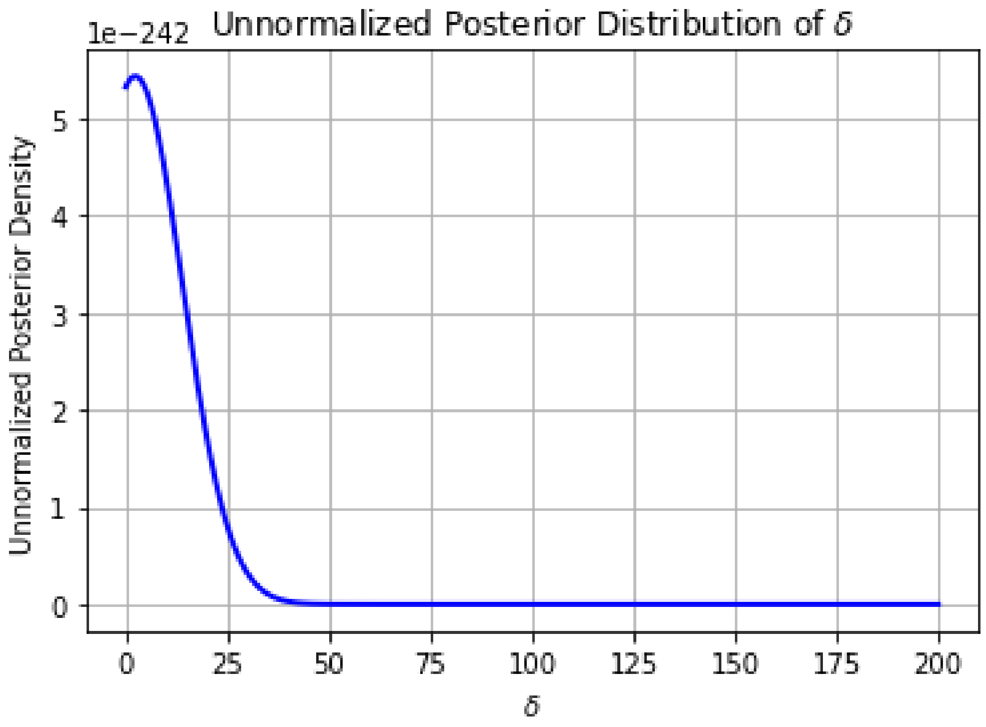


Both graphs are looking similar but for Tnw lexical access model is slightly better than Null hypothesis

4.5.5 [4]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy.stats import norm, truncnorm

# Load the observed data from the URL
url = "https://raw.githubusercontent.com/yadavhimanshu059/CGS698C/main/notes/
 ↪Module-2/recognition.csv"
data = pd.read_csv(url)

# Extract Tw and Tnw columns
Tw_observed = data['Tw'].values
Tnw_observed = data['Tnw'].values
```

```
# Set the parameters
mu_prior_mean = 300
mu_prior_sd = 50
sigma = 60
delta_prior_mean = 0
delta_prior_sd = 50

# Define a range of delta values to evaluate
delta_values = np.linspace(0, 200, 1000)

# Define the prior for mu
def prior_mu(mu):
    return norm.pdf(mu, loc=mu_prior_mean, scale=mu_prior_sd)

# Define the truncated normal prior for delta
def prior_delta(delta):
    return truncnorm.pdf(delta, a=0, b=np.inf, loc=delta_prior_mean,␣
 ↪scale=delta_prior_sd)

# Calculate the likelihoods
def likelihood_Tw(mu):
    return np.prod(norm.pdf(Tw_observed, loc=mu, scale=sigma))

def likelihood_Tnw(mu, delta):
    return np.prod(norm.pdf(Tnw_observed, loc=mu + delta, scale=sigma))

# Calculate the unnormalized posterior
def unnormalized_posterior(delta):
    posterior = 0
    for mu in np.linspace(200, 400, 100):  # Integrate over a range of mu values
        posterior += likelihood_Tw(mu) * likelihood_Tnw(mu, delta) *␣
 ↪prior_mu(mu)
    return posterior * prior_delta(delta)

# Evaluate the unnormalized posterior for each delta
posterior_values = np.array([unnormalized_posterior(delta) for delta in␣
 ↪delta_values])

# Plot the unnormalized posterior distribution of delta
plt.plot(delta_values, posterior_values, lw=2, color='blue')
plt.xlabel(r'$\delta$')
plt.ylabel('Unnormalized Posterior Density')
plt.title('Unnormalized Posterior Distribution of $\delta$')
plt.grid(True)
plt.show()
```


Unnormalized Posterior Distribution of δ
1e−242
Unnormalized Posterior Density
0
1
2
3
4
5
0
25
50
75
100
125
150
175
200
δ

Part 3:

1

3.1 3.1 Calculate the Prior for Day 5

Day 1:

- Prior: $\lambda \sim \text{Gamma}(40, 2)$
- Data: $k_1 = 25$
- Posterior after day 1: $\lambda \sim \text{Gamma}(40 + 25, 2 + 1) = \text{Gamma}(65, 3)$

Day 2:

- Prior: $\lambda \sim \text{Gamma}(65, 3)$
- Data: $k_2 = 20$
- Posterior after day 2: $\lambda \sim \text{Gamma}(65 + 20, 3 + 1) = \text{Gamma}(85, 4)$

Day 3:

- Prior: $\lambda \sim \text{Gamma}(85, 4)$
- Data: $k_3 = 23$
- Posterior after day 3: $\lambda \sim \text{Gamma}(85 + 23, 4 + 1) = \text{Gamma}(108, 5)$

Day 4:

- Prior: $\lambda \sim \text{Gamma}(108, 5)$
- Data: $k_4 = 27$
- Posterior after day 4: $\lambda \sim \text{Gamma}(108 + 27, 5 + 1) = \text{Gamma}(135, 6)$

So, the prior on λ to generate predictions for day 5 is $\lambda \sim \text{Gamma}(135, 6)$.

3.2 **3.2 Predicting Road Accidents on Day 5**

To predict the number of road accidents on day 5, we use the expected value (mean) of the Gamma distribution.

For $\lambda \sim \text{Gamma}(\alpha, \beta)$:

- The mean μ of the distribution is given by: $\mu = \frac{\alpha}{\beta}$

For $\lambda \sim \text{Gamma}(135, 6)$:

- The mean is $\mu = \frac{135}{6} = 22.5$

Therefore, the predicted number of road accidents on day 5 is 22.5.