CGS698C: BAYESIAN MODELS & DATA ANALYSIS

Arya Sorate (Roll no. 220217)

Assignment 2

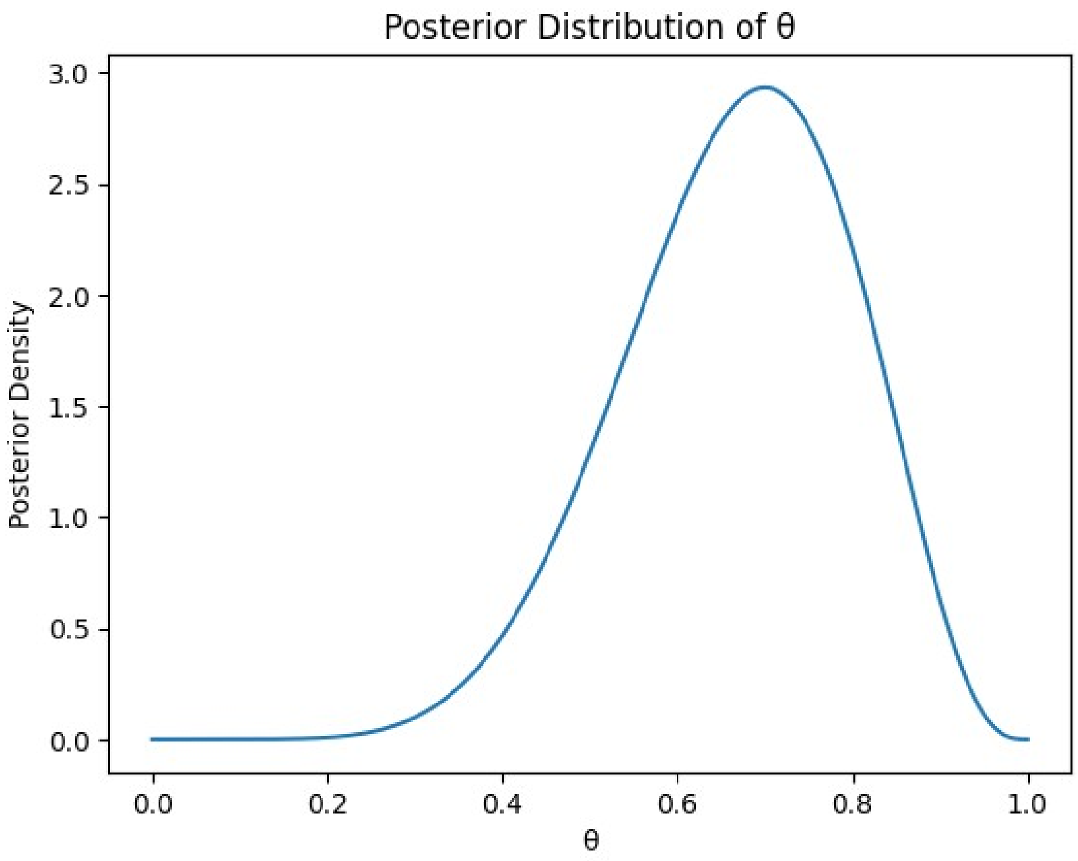
PART 1: *A simple binomial model*

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

### PART 1.1 ###
def binomial_likelihood(y, n, theta):
    return binom.pmf(y,n,theta)

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

def posterior(theta, y, marginal_likelihood):
    n = 10
    likelihood = binomial_likelihood(y,n,theta)
    prior_prob = prior(theta)

    posterior_density = (likelihood * prior_prob) /
marginal_likelihood
    return posterior_density

# Given data
y = 7
marginal_likelihood = 1 / 11  # given marginal likelihood

# Values of theta to estimate posterior density
theta_values = [0.75, 0.25, 1]

print("Answer of Part 1.1")
# Calculate posterior density for each value of theta
for theta in theta_values:
    posterior_density = posterior(theta, y, marginal_likelihood)
    print(f"Posterior density for theta = {theta}:
{posterior_density}")
print(" ")

## PART 1.2 ###
theta_values = np.linspace(0, 1, 1000)  # Create a vector of
```

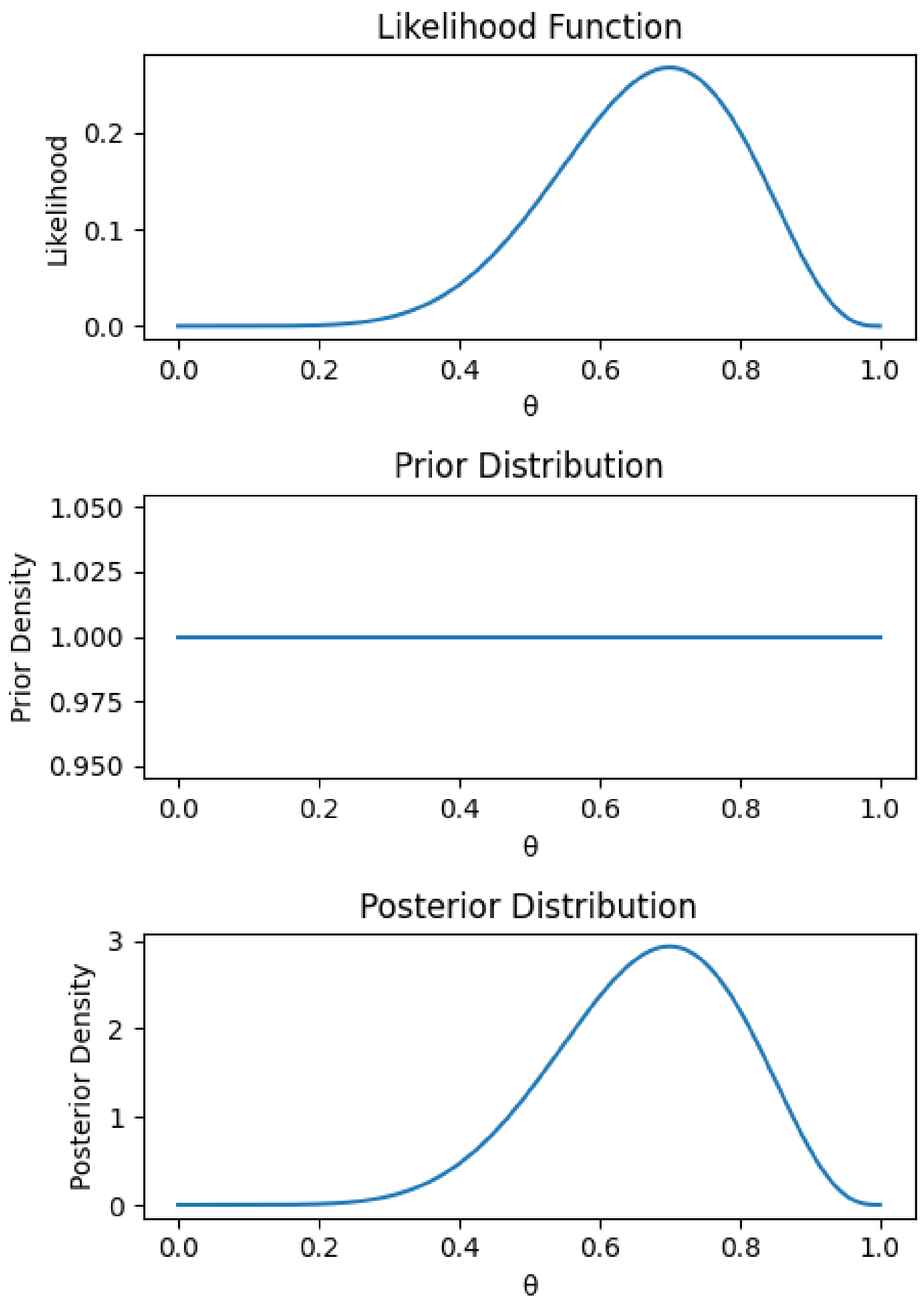
```
equidistant values of theta

# Calculate posterior density for each value of theta
posterior_densities = [posterior(theta, y, marginal_likelihood) for
theta in theta_values]
print(" ")
print("Answer of Part 1.2")
# Plot the posterior distribution
plt.plot(theta_values, posterior_densities)
plt.title('Posterior Distribution of θ')
plt.xlabel('θ')
plt.ylabel('Posterior Density')
plt.show()
print(" ")

### PART 1.3 ###
# Find the value of θ with the maximum posterior density
max_post_density_index = np.argmax(posterior_densities)
theta_max_post_density =
np.round(theta_values[max_post_density_index],4)
print(" ")
print("Answer of Part 1.3")
print(f"The value of θ with the maximum posterior density is =
{theta_max_post_density}")
print(" ")
likelihood_values = binomial_likelihood(y, 10, theta_values)

### PART 1.4 ###
# Calculate prior distribution for each value of theta
prior_values = [prior(theta) for theta in theta_values]

# Calculate posterior distribution for each value of theta
posterior_values = [posterior(theta, y, marginal_likelihood) for theta
in theta_values]

print(" ")
print("Answer of Part 1.4")
# Plot all distributions together
plt.figure(figsize=(5, 7))

# Plot the likelihood function
plt.subplot(3, 1, 1)
plt.plot(theta_values, likelihood_values)
plt.title('Likelihood Function')
plt.xlabel('θ')
plt.ylabel('Likelihood')


# Plot the prior distribution
plt.subplot(3, 1, 2)
```

```python
plt.plot(theta_values, prior_values)
plt.title('Prior Distribution')
plt.xlabel('θ')
plt.ylabel('Prior Density')


# Plot the posterior distribution
plt.subplot(3, 1, 3)
plt.plot(theta_values, posterior_values)
plt.title('Posterior Distribution')
plt.xlabel('θ')
plt.ylabel('Posterior Density')


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

```
Answer of Part 1.1
Posterior density for theta = 0.75: 2.7531051635742174
Posterior density for theta = 0.25: 0.033988952636718 74
Posterior density for theta = 1: 0.0


Answer of Part 1.2
```



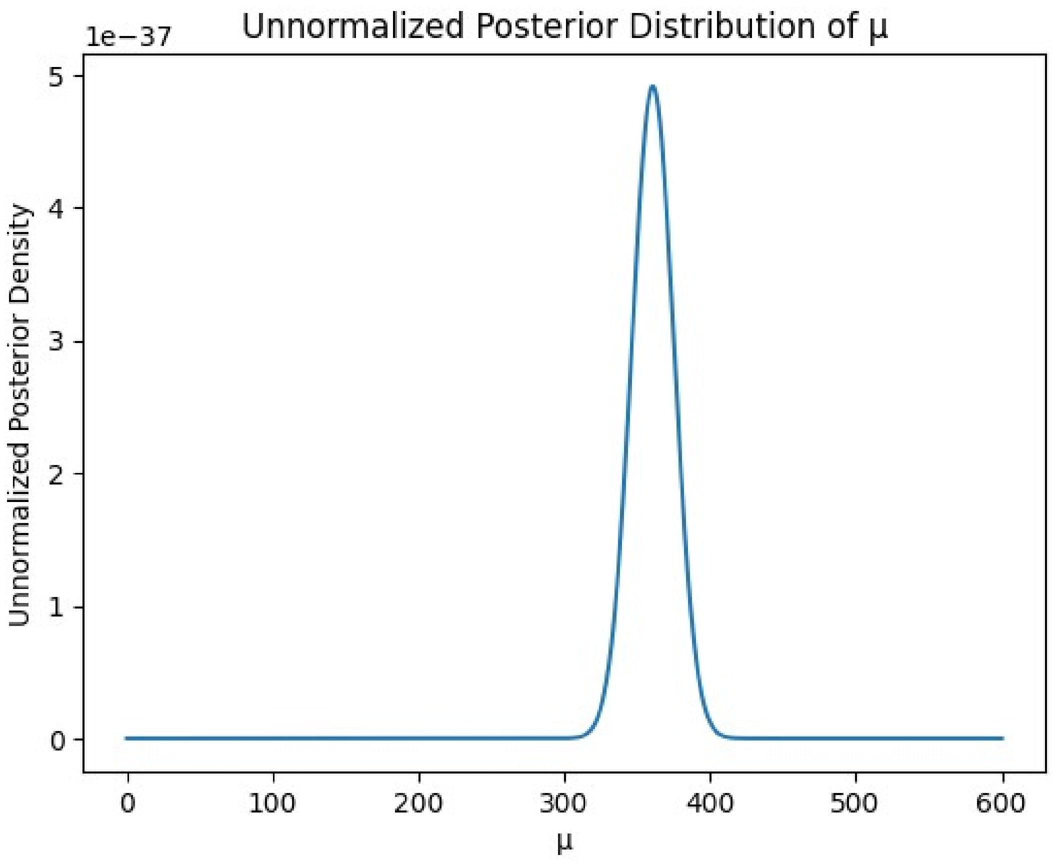
```
Answer of Part 1.3
The value of θ with the maximum posterior density is = 0.6997


Answer of Part 1.4
```



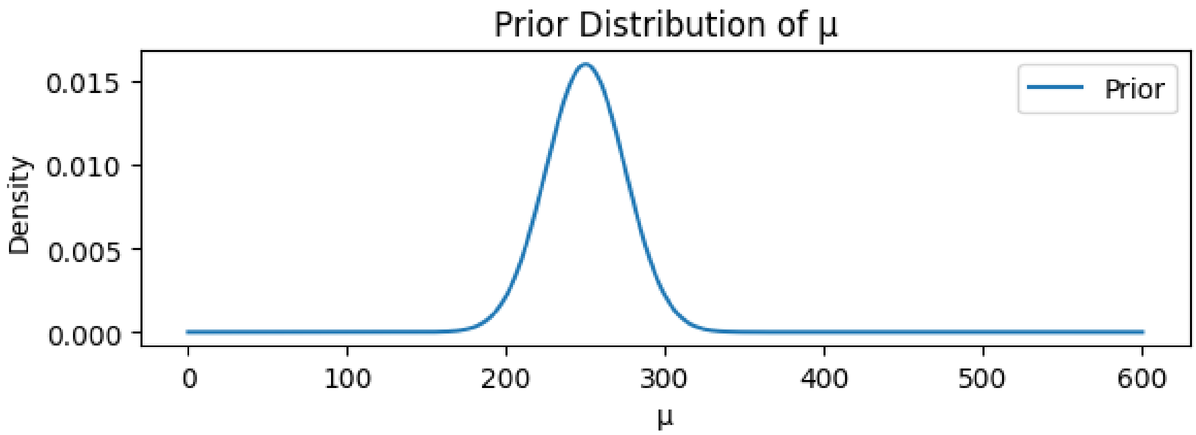
PART 2: *A Gaussian model of reading*

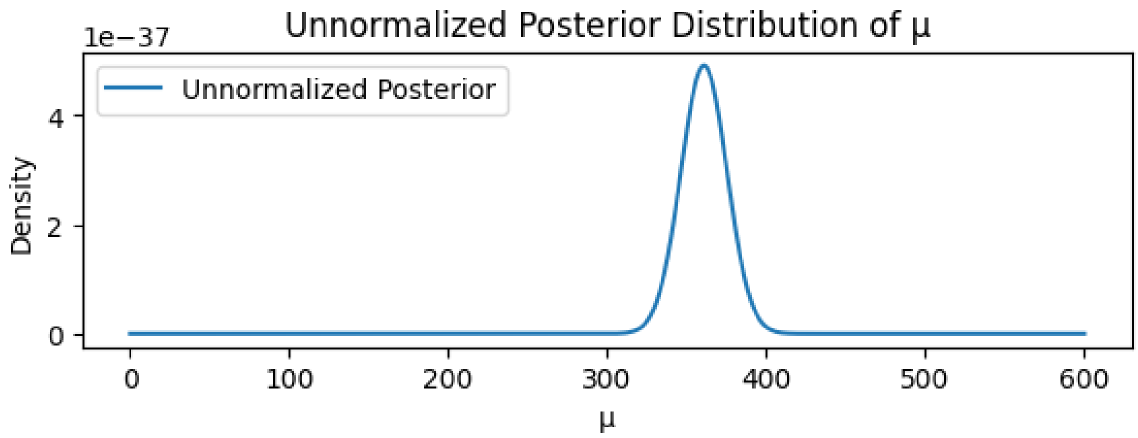
```
import numpy as np

### PART 2.1 ###
# Likelihood function
def likelihood(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))
```

```
# Prior distribution for μ
def prior(mu):
    mu_prior = (1 / (25 * np.sqrt(2 * np.pi))) * np.exp(-0.5 * ((mu -
250) / 25)**2)
    return mu_prior

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

# Values of μ to calculate unnormalized posterior density
mu_values = [300, 900, 50]
print("Answer of Part 2.1")
# Calculate unnormalized posterior density for each value of μ
for mu in mu_values:
    unnormalized_posterior_density = likelihood(y, mu, sigma) *
prior(mu)
    print(f"Unnormalized posterior density for μ = {mu}:
{unnormalized_posterior_density}")
print(" ")

### PART 2.2 ###
# Values of μ to calculate unnormalized posterior density
mu_values = np.linspace(0, 600, 1000)

# Calculate unnormalized posterior density for each value of μ
unnormalized_posterior_densities = [likelihood(y, mu, sigma) *
prior(mu) for mu in mu_values]

print(" ")
print("Answer for Part 2.2")
# Plot the unnormalized posterior distribution
plt.plot(mu_values, unnormalized_posterior_densities)
plt.title('Unnormalized Posterior Distribution of μ')
plt.xlabel('μ')
plt.ylabel('Unnormalized Posterior Density')
plt.show()
print(" ")

### PART 2.3 ###
# Calculate prior distribution for each value of μ
prior_densities = [prior(mu) for mu in mu_values]
print(" ")
print("Answer for 2.3")
# Plot the prior distribution
plt.subplot(2, 1, 1)
plt.plot(mu_values, prior_densities, label='Prior')
plt.title('Prior Distribution of μ')
plt.xlabel('μ')
plt.ylabel('Density')
plt.legend()

# Plot the unnormalized posterior distribution
plt.subplot(2, 1, 2)
plt.plot(mu_values, unnormalized_posterior_densities,
label='Unnormalized Posterior')
plt.title('Unnormalized Posterior Distribution of μ')
plt.xlabel('μ')
plt.ylabel('Density')
plt.legend()

plt.tight_layout()
plt.show()

Answer of Part 2.1
Unnormalized posterior density for μ = 300: 6.824247957486404e-41
Unnormalized posterior density for μ = 900: 0.0
Unnormalized posterior density for μ = 50: 9.691373559300646e-138


Answer for Part 2.2
```



```
Answer for 2.3
```





PART 3: *The Bayesian learning*

```python
import numpy as np
from scipy.stats import gamma

# Given data
data = [25, 20, 23, 27]

# Prior parameters
prior_shape = 40
prior_rate = 2

# Function to calculate posterior distribution
def posterior(data, prior_shape, prior_rate):
    posterior_shape = prior_shape + sum(data)
    posterior_rate = prior_rate + len(data)
    return posterior_shape, posterior_rate
```

```
# Calculate posterior distribution for each day
posterior_shape, posterior_rate = posterior(data, prior_shape,
prior_rate)

# Prior for day 5
prior_shape_day_5 = posterior_shape
prior_rate_day_5 = posterior_rate

# Predictions for day 5
mean_accidents_day_5 = prior_shape_day_5 / prior_rate_day_5

print("Prior distribution parameters for day 5 (Gamma):")
print("Shape:", prior_shape_day_5)
print("Rate:", prior_rate_day_5)
print("Therfore, Gamma(",prior_shape_day_5,',',prior_rate_day_5,")")
print("\nNumber of road accidents predicted to happen on day 5:",
mean_accidents_day_5)

Prior distribution parameters for day 5 (Gamma):
Shape: 135
Rate: 6
Therfore, Gamma( 135 , 6 )

Number of road accidents predicted to happen on day 5: 22.5
```

PART 4: *Model building in the Bayesian framework*

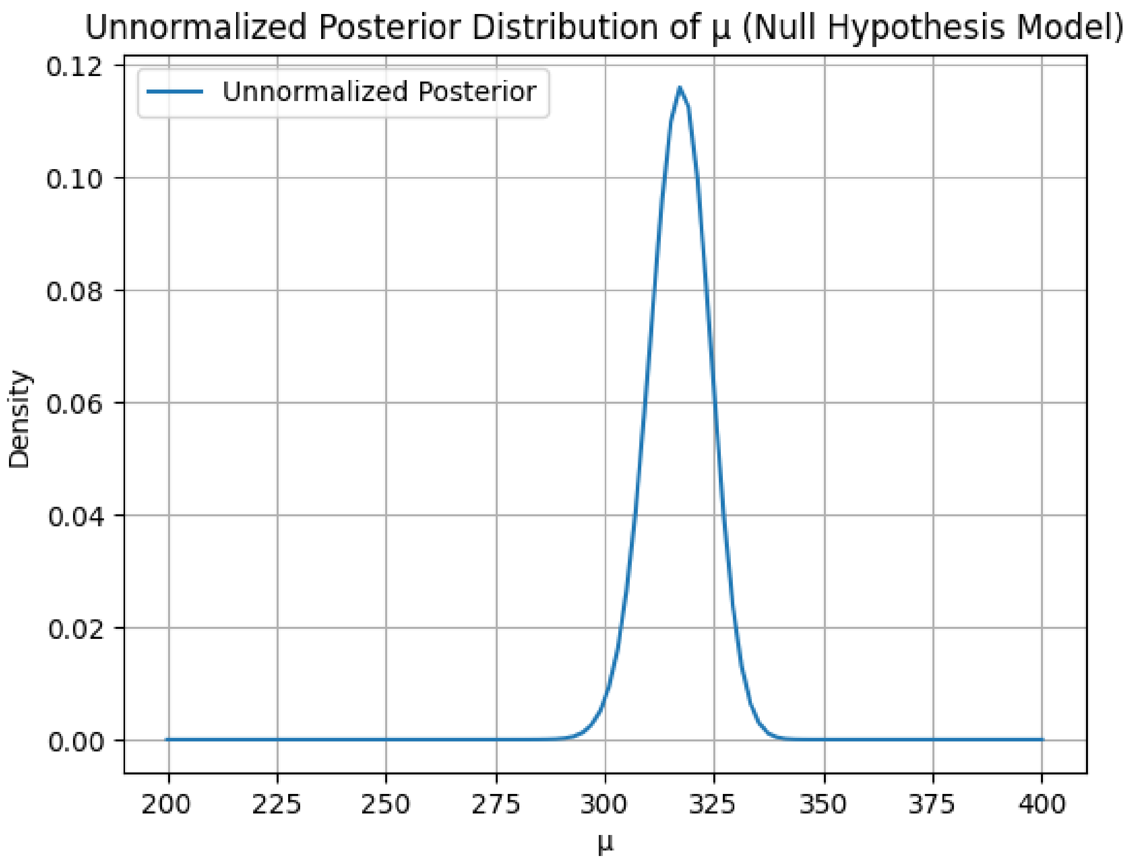
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm, truncnorm
import pandas as pd

### PART 4.5.1 ###
# Read the data from CSV file
data =
pd.read_csv("https://raw.githubusercontent.com/yadavhimanshu059/CGS698
C/main/notes/Module-2/recognition.csv")

# Extracting Tw and Tnw from the dataframe
Tw = data['Tw'].values
Tnw = data['Tnw'].values

# Other parameters
σ = 60
a = 0
b = np.inf
prior_μ_mean = 300
prior_μ_sd = 50
prior_δ_mean = 0
prior_δ_sd = 50
```

```
# Define the likelihood function
def likelihood(μ, δ, Tw, Tnw, σ):
    likelihood_word = np.prod(norm.pdf(Tw, loc=μ, scale=σ))
    likelihood_nonword = np.prod(norm.pdf(Tnw, loc=μ + δ, scale=σ))
    return likelihood_word * likelihood_nonword

# Define the prior distributions
def prior_μ(μ):
    return norm.pdf(μ, loc=prior_μ_mean, scale=prior_μ_sd)

def prior_δ(δ):
    return truncnorm.pdf(δ, a, b, loc=prior_δ_mean, scale=prior_δ_sd)

# Calculate unnormalized posterior for various values of μ
μ_values = np.linspace(200, 400, 100)
δ_values = np.linspace(-100, 100, 100)
posterior_values = np.zeros_like(μ_values)

for i, μ in enumerate(μ_values):
    for δ in δ_values:
        posterior_values[i] += likelihood(μ, δ, Tw, Tnw, σ) *
prior_μ(μ) * prior_δ(δ)

# Normalize the posterior distribution
posterior_values /= np.sum(posterior_values)

print("Answer of Part 4.5.1")
# Plot the unnormalized posterior distribution
plt.plot(μ_values, posterior_values, label='Unnormalized Posterior')
plt.xlabel('μ')
plt.ylabel('Density')
plt.title('Unnormalized Posterior Distribution of μ (Null Hypothesis
Model)')
plt.legend()
plt.grid(True)
plt.show()
print(" ")

### PART 4.5.2 ###
n_samples = 1000

# Draw samples from prior distributions
μ_samples = np.random.normal(prior_μ_mean, prior_μ_sd, n_samples)
δ_samples = truncnorm.rvs((0 - prior_δ_mean) / prior_δ_sd, np.inf,
loc=prior_δ_mean, scale=prior_δ_sd, size=n_samples)

# Generate non-word recognition times
lexical_access_non_word_recognition_times = μ_samples + δ_samples +
np.random.normal(0, σ, n_samples)
```

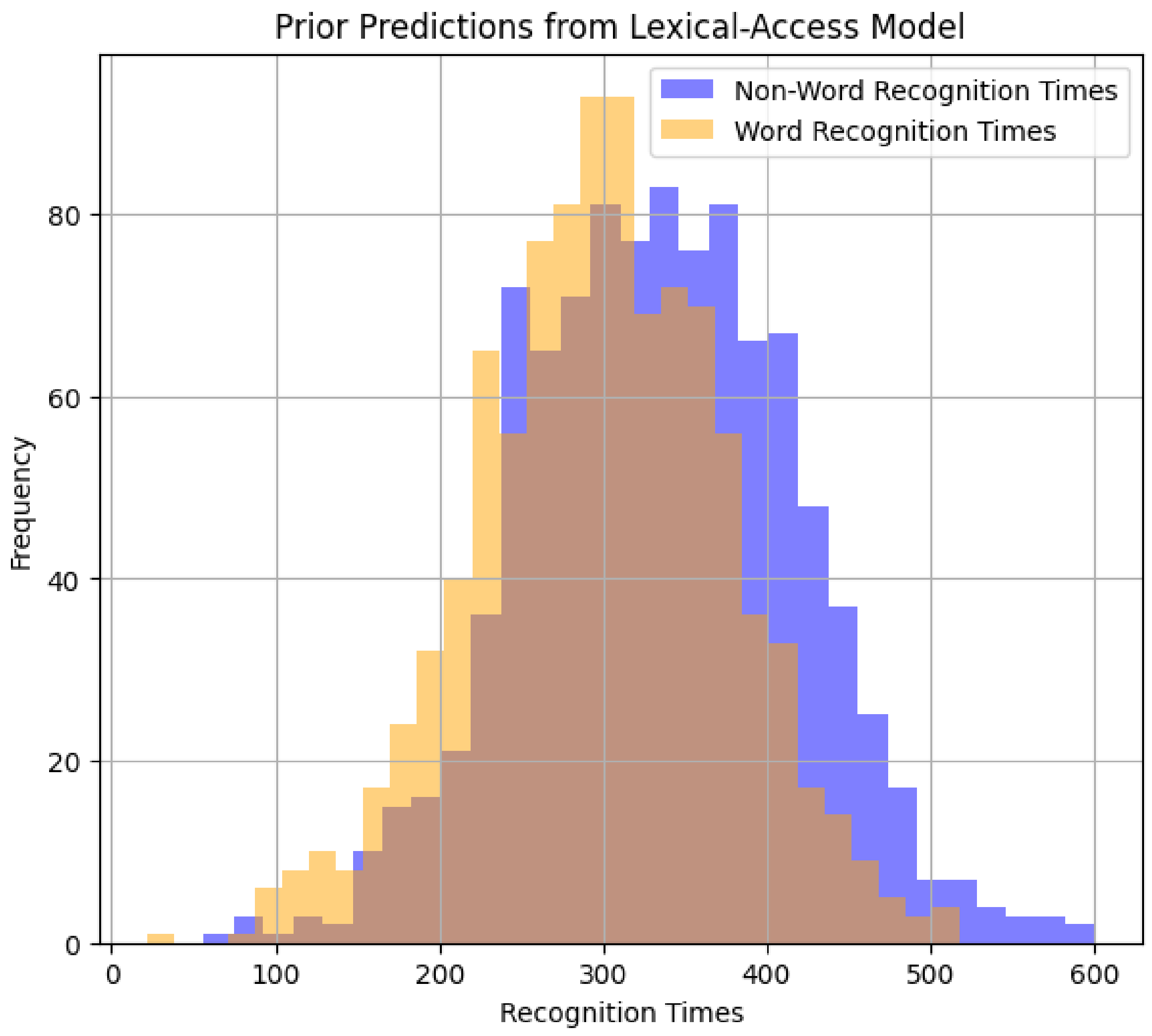
```
# Generate word recognition times
lexical_access_word_recognition_times = μ_samples +
np.random.normal(0, σ, n_samples)

print(" ")
print("Answer of Part 4.5.2")
# Plot histograms
plt.figure(figsize=(7, 6))
plt.hist(lexical_access_non_word_recognition_times, bins=30,
alpha=0.5, label='Non-Word Recognition Times', color='blue')
plt.hist(lexical_access_word_recognition_times, bins=30, alpha=0.5,
label='Word Recognition Times', color='orange')
plt.xlabel('Recognition Times')
plt.ylabel('Frequency')
plt.title('Prior Predictions from Lexical-Access Model')
plt.legend()
plt.grid(True)
plt.show()
print(" ")

### PART 4.5.3 ###
null_hypothesis_word_recognition_times =
np.random.normal(prior_μ_mean, σ, n_samples)
null_hypothesis_nonword_recognition_times =
np.random.normal(prior_μ_mean, σ, n_samples)
print(" ")
print("Answer of Part 4.5.3")
# Plot histograms for comparison
plt.figure(figsize=(6, 8))
plt.subplot(2, 1, 1)
plt.hist(null_hypothesis_word_recognition_times, bins=30, alpha=0.5,
label='Word', color='blue')
plt.hist(null_hypothesis_nonword_recognition_times, bins=30,
alpha=0.5, label='Non-Word', color='orange')
plt.xlabel('Recognition Times')
plt.ylabel('Frequency')
plt.title('Null Hypothesis Model')
plt.legend()
plt.grid(True)

# Plot histograms for the lexical-access model
plt.subplot(2, 1, 2)
plt.hist(lexical_access_word_recognition_times, bins=30, alpha=0.5,
label='Word', color='green')
plt.hist(lexical_access_non_word_recognition_times, bins=30,
alpha=0.5, label='Non-Word', color='red')
plt.xlabel('Recognition Times')
plt.ylabel('Frequency')
plt.title('Lexical Access Model')
plt.legend()
plt.grid(True)

plt.tight_layout()
plt.show()
print(" ")

### Part 4.5.4 ###
print(" ")
print("Answer of Part 4.5.4")
# Plot histograms for comparison
plt.figure(figsize=(10, 15))

# Plot subplots for word recognition times (Tw)
plt.subplot(3, 1, 1)

# Plot observed data for Tw
plt.hist(Tw, bins=30, alpha=0.5, label='Observed Data', color='blue')

# Plot prior predictions for the null hypothesis model for Tw
plt.hist(null_hypothesis_word_recognition_times, bins=30, alpha=0.5,
label='Null Hypothesis', color='green')

# Plot prior predictions for the lexical-access model for Tw
plt.hist(lexical_access_word_recognition_times, bins=30, alpha=0.5,
label='Lexical Access', color='purple')

# Add vertical lines indicating the peak for Tw
observed_peak_Tw_bin = np.argmax(np.histogram(Tw, bins=30)[0])
observed_peak_Tw_value = np.histogram(Tw, bins=30)[1]
[observed_peak_Tw_bin]
plt.axvline(x=observed_peak_Tw_value, color='blue', linestyle='--',
label=f'Peak = {observed_peak_Tw_value}')
plt.text(observed_peak_Tw_value, 150, f'{observed_peak_Tw_value}',
fontsize=10, color='blue')

null_peak_Tw_bin =
np.argmax(np.histogram(null_hypothesis_word_recognition_times,
bins=30)[0])
null_peak_Tw_value =
np.histogram(null_hypothesis_word_recognition_times, bins=30)[1]
[null_peak_Tw_bin]
plt.axvline(x=null_peak_Tw_value, color='green', linestyle='--',
label=f'Peak = {null_peak_Tw_value}')
plt.text(null_peak_Tw_value, 150, f'{null_peak_Tw_value}',
fontsize=10, color='green')

lexical_peak_Tw_bin =
np.argmax(np.histogram(lexical_access_word_recognition_times, bins=30)
[0])
```

```
lexical_peak_Tw_value =
np.histogram(lexical_access_word_recognition_times, bins=30)[1]
[lexical_peak_Tw_bin]
plt.axvline(x=lexical_peak_Tw_value, color='purple', linestyle='--',
label=f'Peak = {lexical_peak_Tw_value}')
plt.text(lexical_peak_Tw_value, 150, f'{lexical_peak_Tw_value}',
fontsize=10, color='purple')

plt.xlabel('Word Recognition Times')
plt.ylabel('Frequency')
plt.title('Comparison of Prior Predictions with Observed Data (Word
Recognition)')
plt.legend()
plt.grid(True)

# Plot subplots for non-word recognition times (Tnw)
plt.subplot(3, 1, 2)

# Plot observed data for Tnw
plt.hist(Tnw, bins=30, alpha=0.5, label='Observed Data',
color='orange')

# Plot prior predictions for the null hypothesis model for Tnw
plt.hist(null_hypothesis_nonword_recognition_times, bins=30,
alpha=0.5, label='Null Hypothesis', color='red')

# Plot prior predictions for the lexical-access model for Tnw
plt.hist(lexical_access_non_word_recognition_times, bins=30,
alpha=0.5, label='Lexical Access', color='green')

# Add vertical lines indicating the peak for Tnw
observed_peak_Tnw_bin = np.argmax(np.histogram(Tnw, bins=30)[0])
observed_peak_Tnw_value = np.histogram(Tnw, bins=30)[1]
[observed_peak_Tnw_bin]
plt.axvline(x=observed_peak_Tnw_value, color='orange', linestyle='--',
label=f'Peak = {observed_peak_Tnw_value}')
plt.text(observed_peak_Tnw_value, 150, f'{observed_peak_Tnw_value}',
fontsize=10, color='orange')

null_peak_Tnw_bin =
np.argmax(np.histogram(null_hypothesis_nonword_recognition_times,
bins=30)[0])
null_peak_Tnw_value =
np.histogram(null_hypothesis_nonword_recognition_times, bins=30)[1]
[null_peak_Tnw_bin]
plt.axvline(x=null_peak_Tnw_value, color='red', linestyle='--',
label=f'Peak = {null_peak_Tnw_value}')
plt.text(null_peak_Tnw_value, 150, f'{null_peak_Tnw_value}',
fontsize=10, color='red')
```

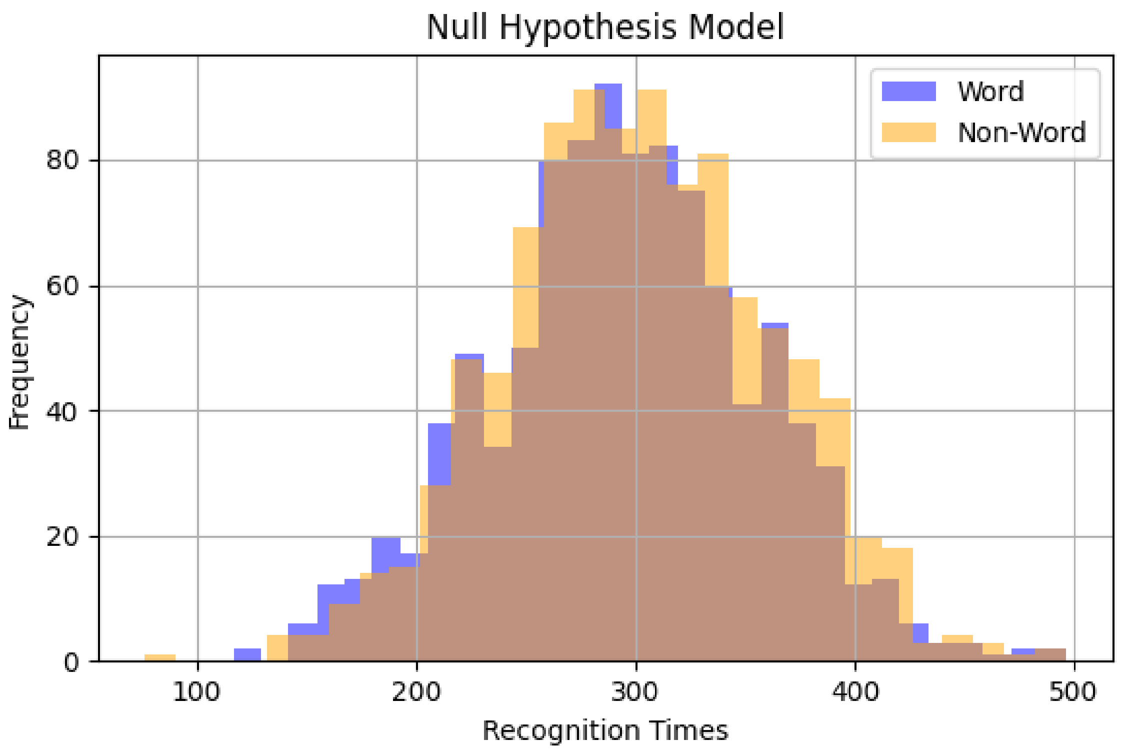
```
lexical_peak_Tnw_bin =
np.argmax(np.histogram(lexical_access_non_word_recognition_times,
bins=30)[0])
lexical_peak_Tnw_value =
np.histogram(lexical_access_non_word_recognition_times, bins=30)[1]
[lexical_peak_Tnw_bin]
plt.axvline(x=lexical_peak_Tnw_value, color='green', linestyle='--',
label=f'Peak = {lexical_peak_Tnw_value}')
plt.text(lexical_peak_Tnw_value, 150, f'{lexical_peak_Tnw_value}',
fontsize=10, color='green')

plt.xlabel('Non-Word Recognition Times')
plt.ylabel('Frequency')
plt.title('Comparison of Prior Predictions with Observed Data (Non-
Word Recognition)')
plt.legend()
plt.grid(True)

plt.tight_layout()
plt.show()
print(" ")
### Part 4.5.5 ###
# Sample values for δ from the prior distribution
δ_samples = truncnorm.rvs((0 - prior_δ_mean) / prior_δ_sd, np.inf,
loc=prior_δ_mean, scale=prior_δ_sd, size=n_samples)

# Likelihood function for the lexical-access model
def likelihood(μ, δ, Tw, Tnw):
    return np.prod(norm.pdf(Tw, loc=μ, scale=σ)) *
np.prod(norm.pdf(Tnw, loc=μ + δ, scale=σ))

# Prior distribution for δ
def prior(δ):
    return norm.pdf(δ, loc=prior_δ_mean, scale=prior_δ_sd)

# Compute unnormalized posterior distribution for δ
def posterior(δ, Tw, Tnw):
    return likelihood(prior_μ_mean, δ, Tw, Tnw) * prior(δ)

# Compute posterior values for the samples of δ
posterior_values = [posterior(δ, Tw, Tnw) for δ in δ_samples]

print(" ")
print("Answer of Part 4.5.5")
# Plot the unnormalized posterior distribution of δ
plt.plot(δ_samples, posterior_values, '.', alpha=0.5)
plt.xlabel('δ')
plt.ylabel('Unnormalized Posterior')
plt.title('Unnormalized Posterior Distribution of δ (Lexical-Access
Model)')
plt.show()

Answer of Part 4.5.1
```

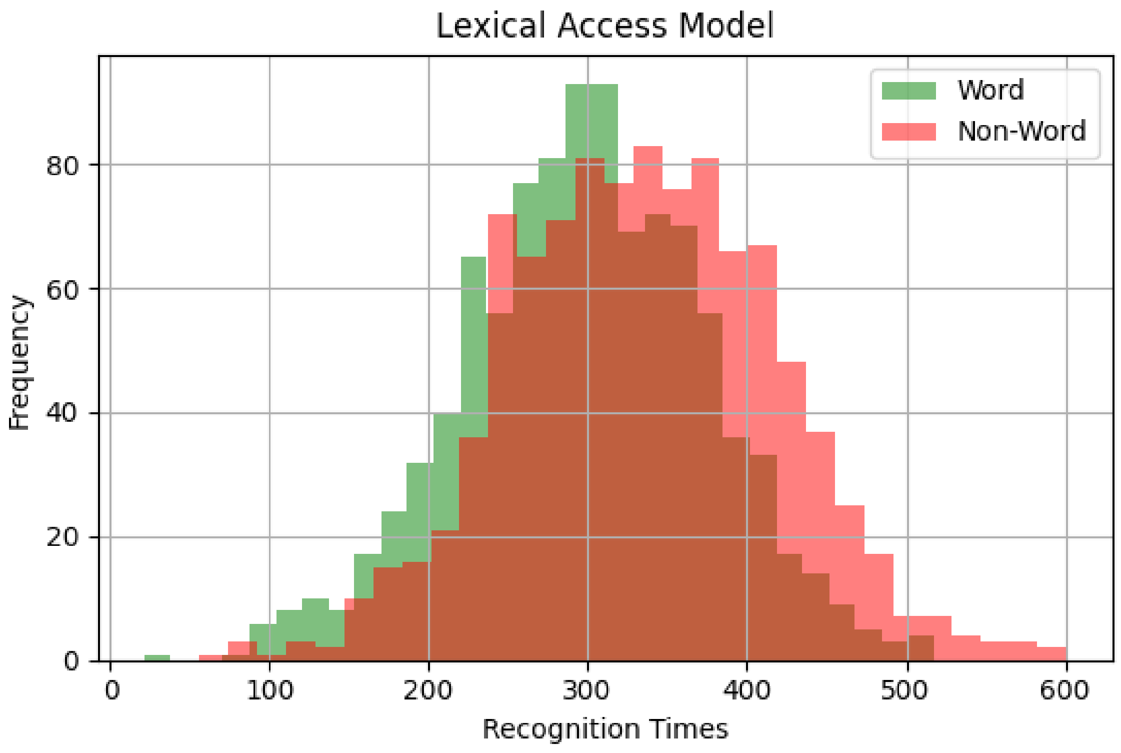


```


Answer of Part 4.5.2
```

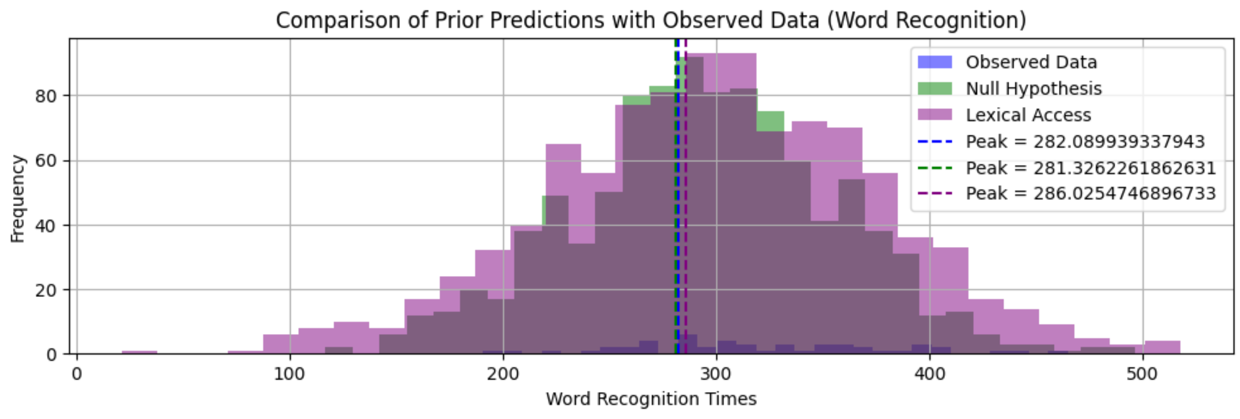


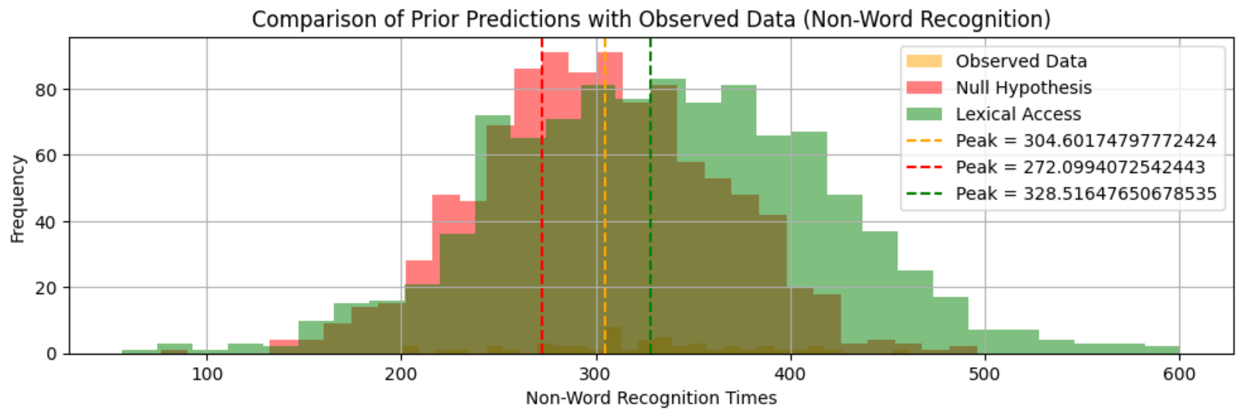
```
Answer of Part 4.5.3
```



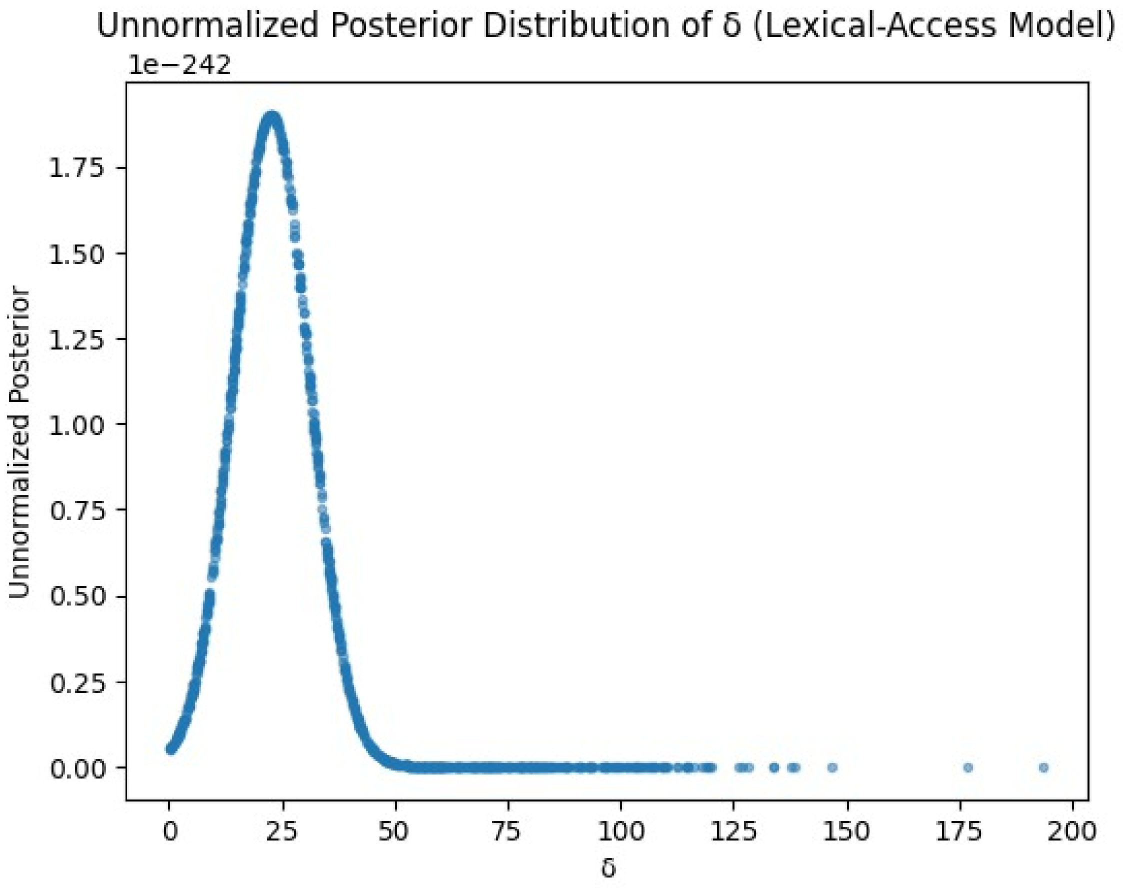


```
Answer of Part 4.5.4
```





```
Answer of Part 4.5.5
```



Conclusions: In part 4.5.3, we are expected to compare the prior predictions of the null hypothesis model and the lexical access model. We can do this by comparing their histograms. Hence, we plot the histograms for both of them.

In part 4.5.4, We observe that for Word-Recognition times, Null hypothesis is closer to the given data as compared to the Lexical access, as seen by comparing the peaks. On the other hand, for Non Word-Recognition times, Both models were equally close but Lexical access model was slightly better than Null hypothesis model.