intw7mbnz

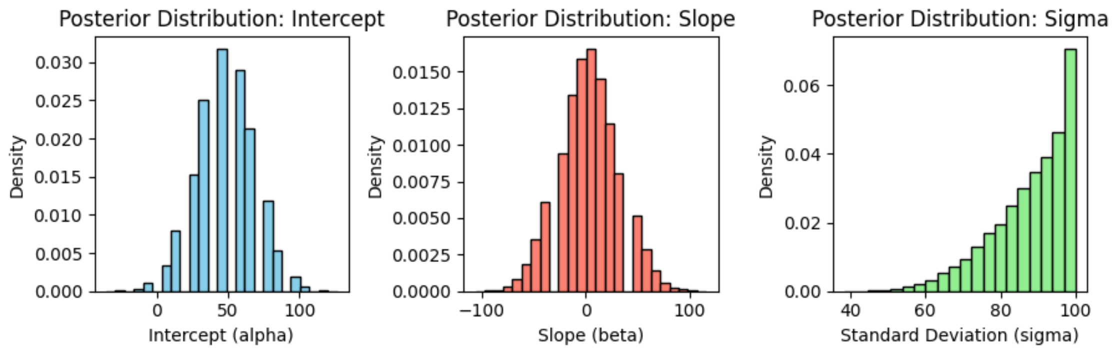
April 6, 2024

0.0.1 Part 1: A simple linear regression: Power posing and testosterone

[ ]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# Load the CSV file into a DataFrame
df = pd.read_csv("/content/testosterone.csv")

# Define the treatment variable (0 for low, 1 for high)
df['treatment'] = (df['hptreat'] == 'High').astype(int)

# Define the likelihood function for Bayesian linear regression
def likelihood(alpha, beta, sigma, x, y):
    mu = alpha + beta * x
    return np.exp(-0.5 * np.sum((y - mu) ** 2) / sigma ** 2)

# Define the prior distribution for alpha (intercept)
def prior_alpha(alpha):
    return 1  # Flat prior

# Define the prior distribution for beta (slope)
def prior_beta(beta):
    return 1  # Flat prior

# Define the prior distribution for sigma (standard deviation)
def prior_sigma(sigma):
    return 1 / sigma  # Jeffrey's prior

# Compute the unnormalized posterior distribution
def unnormalized_posterior(alpha, beta, sigma, x, y):
    return likelihood(alpha, beta, sigma, x, y) * prior_alpha(alpha) *␣
 ↪prior_beta(beta) * prior_sigma(sigma)

# Define the grid for alpha, beta, and sigma
alpha_values = np.linspace(-500, 500, 100)
beta_values = np.linspace(-500, 500, 100)
sigma_values = np.linspace(0.1, 100, 100)
```

```
# Compute the posterior distribution on the grid
posterior = np.zeros((len(alpha_values), len(beta_values), len(sigma_values)))
for i, alpha in enumerate(alpha_values):
    for j, beta in enumerate(beta_values):
        for k, sigma in enumerate(sigma_values):
            posterior[i, j, k] = unnormalized_posterior(alpha, beta, sigma,␣
 ↪df['treatment'], df['testm2'])

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

[26]:
```
# Flatten the posterior distribution
flattened_posterior = posterior.reshape(-1)

# Sample from the flattened posterior distribution
n_samples = 30000  # Adjust the number of samples as needed
samples_indices = np.random.choice(flattened_posterior.size, size=n_samples,␣
 ↪p=flattened_posterior.flatten())
alpha_samples = []
beta_samples = []
sigma_samples = []

# Convert the sampled indices back to 3D indices
for index in samples_indices:
    i, j, k = np.unravel_index(index, posterior.shape)
    alpha_samples.append(alpha_values[i])
    beta_samples.append(beta_values[j])
    sigma_samples.append(sigma_values[k])

# Plot histograms of the posterior samples for alpha, beta, and sigma
plt.figure(figsize=(9, 3))

plt.subplot(1, 3, 1)
plt.hist(alpha_samples, bins=25, density=True, color='skyblue',␣
 ↪edgecolor='black')
plt.xlabel('Intercept (alpha)')
plt.ylabel('Density')
plt.title('Posterior Distribution: Intercept')

plt.subplot(1, 3, 2)
plt.hist(beta_samples, bins=25, density=True, color='salmon', edgecolor='black')
plt.xlabel('Slope (beta)')
plt.ylabel('Density')
plt.title('Posterior Distribution: Slope')

plt.subplot(1, 3, 3)
plt.hist(sigma_samples, bins=20, density=True, color='lightgreen',␣
  ↪edgecolor='black')
plt.xlabel('Standard Deviation (sigma)')
plt.ylabel('Density')
plt.title('Posterior Distribution: Sigma')

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

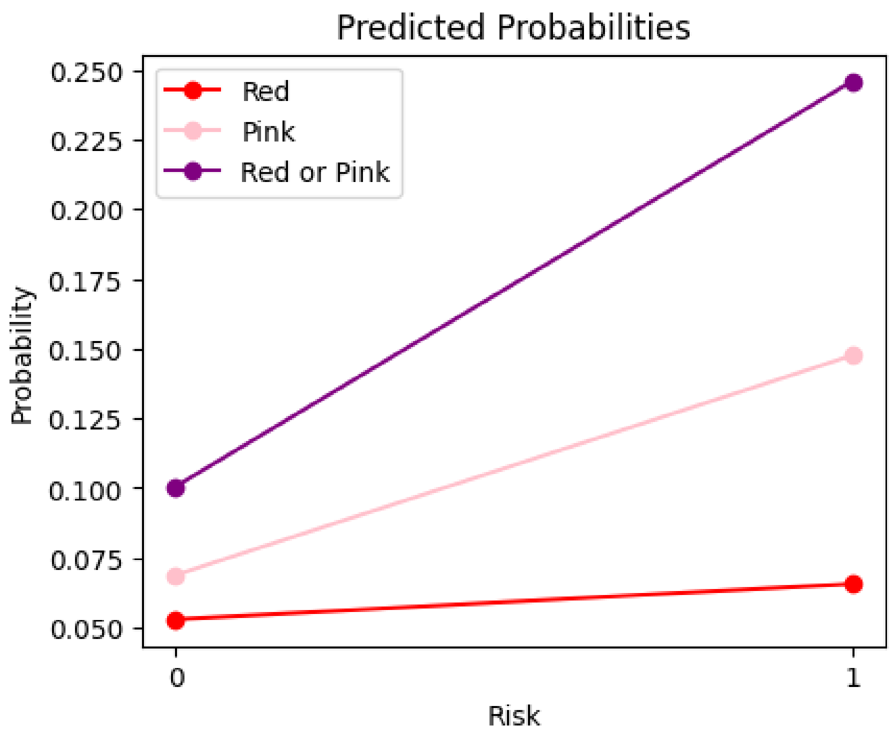


As we can see from the above graphs, the slope can take either positive or negative values, which means that the testosterone levels after treatment may be higher or lower when assigned a high power pose to the subject. So, by this analysis, the hypothesis is incorrect.

0.0.2 Part 2: Logisitic regression model

[15]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.special import expit
from scipy.optimize import minimize

# Load the data
data = pd.read_csv("/content/red ovulation.csv")

# Define sigmoid function (logistic function)
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

# Define negative log-likelihood function
def negative_log_likelihood(params, X, y):
    alpha, beta = params
    log_odds = alpha + beta * X
    p = sigmoid(log_odds)
    likelihood = np.sum(y * np.log(p) + (1 - y) * np.log(1 - p))
    return -likelihood

# Define prior function (regularization)
def prior(params):
    alpha, beta = params
    # Set priors (adjust as needed)
    alpha_prior = (alpha - 0) / 1.5  # Normal prior with mean 0 and standard␣
 ↪deviation 1.5
    beta_prior = (beta - 0) / 0.5  # Normal prior with mean 0 and standard␣
 ↪deviation 0.5
    # Combine priors
    prior_penalty = alpha_prior + beta_prior
    return prior_penalty

# Define negative log-posterior function (likelihood + prior)
def negative_log_posterior(params, X, y):
    likelihood = negative_log_likelihood(params, X, y)
    prior_penalty = prior(params)
    return likelihood + prior_penalty

# Initial guess for parameters
initial_guess = np.zeros(2)

# Optimize parameters (alpha and beta) for red model
result_red = minimize(negative_log_posterior, initial_guess,␣
 ↪args=(data['risk'], data['red']), method='BFGS')
alpha_opt_red, beta_opt_red = result_red.x

# Optimize parameters (alpha and beta) for pink model
result_pink = minimize(negative_log_posterior, initial_guess,␣
 ↪args=(data['risk'], data['pink']), method='BFGS')
alpha_opt_pink, beta_opt_pink = result_pink.x

# Optimize parameters (alpha and beta) for red or pink model
result_redorpink = minimize(negative_log_posterior, initial_guess,␣
 ↪args=(data['risk'], data['redorpink']), method='BFGS')
alpha_opt_redorpink, beta_opt_redorpink = result_redorpink.x

# Compute predicted probabilities for each model
x_values = np.array([0, 1])  # Possible values of risk
log_odds_red = alpha_opt_red + beta_opt_red * x_values
predicted_probs_red = sigmoid(log_odds_red)

log_odds_pink = alpha_opt_pink + beta_opt_pink * x_values
predicted_probs_pink = sigmoid(log_odds_pink)

log_odds_redorpink = alpha_opt_redorpink + beta_opt_redorpink * x_values
```

```
predicted_probs_redorpink = sigmoid(log_odds_redorpink)

# Plot the predicted probabilities for all models
plt.figure(figsize=(5, 4))
plt.plot(x_values, predicted_probs_red, marker='o', color='red', label='Red')
plt.plot(x_values, predicted_probs_pink, marker='o', color='pink', label='Pink')
plt.plot(x_values, predicted_probs_redorpink, marker='o', color='purple',␣
 ↪label='Red or Pink')
plt.xlabel('Risk')
plt.ylabel('Probability')
plt.title('Predicted Probabilities')
plt.xticks([0, 1])  # Set x-axis ticks to only show 0 and 1
plt.legend()
plt.show()
```



The probabilities of wearing red, pink and red or pink do indeed see an increase when the females are at the risk of becoming pregnant or are ovulating. But a higher level of increase is seen in pink as compared to red and the highest increase is in the probability of wearing red or pink both combined (based on the slope we see in the graph).

```
[ ]:
```