CGS698C: BAYESIAN MODELS & DATA ANALYSIS

Assignment - 4

Arya Sorate (220217)

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

```
import pandas as pd

# Load the data
df_powerpose = pd.read_csv('/content/df_powerpose_99f68d8a-2128-47c5-
84c4-93af776c0468.csv')

# Display the first few rows of the dataframe
df_powerpose.head()


## Now, let's prepare the data by creating a column for the change in
testosterone levels:
# Calculate the change in testosterone levels
df_powerpose['delta_testosterone'] = df_powerpose['testm2'] -
df_powerpose['testm1']

# Display the first few rows to check the new column
df_powerpose.head()

## Next, we will perform the linear regression using statsmodels:
# Import necessary libraries for regression
import statsmodels.api as sm

# Define the dependent variable (delta_testosterone) and independent
variable (hptreat)
X = df_powerpose['hptreat'].map({'Low': 0, 'High': 1})  # Convert
categorical to numerical
y = df_powerpose['delta_testosterone']

# Add a constant to the independent variable (intercept)
X = sm.add_constant(X)

# Perform the linear regression
model = sm.OLS(y, X).fit()

# Print the summary of the regression results
print(model.summary())
```

```
## Finally, let's interpret the results:
# Interpreting the results
summary = model.summary()
print("Summary of regression results:\n", summary)

# Interpretation:
# If the coefficient for 'hptreat' is significantly positive, it
suggests that high power posing is associated with an increase in
testosterone levels.
# Check the p-value of the 'hptreat' coefficient to determine
statistical significance (typically p < 0.05 is considered
significant).

                            OLS Regression Results

======================================================================
========
Dep. Variable:     delta_testosterone   R-squared:
0.048
Model:                            OLS   Adj. R-squared:
0.022
Method:                 Least Squares   F-statistic:
1.869
Date:                Mon, 01 Jul 2024   Prob (F-statistic):
0.180
Time:                        11:17:51   Log-Likelihood:
-171.48
No. Observations:                  39   AIC:
347.0
Df Residuals:                      37   BIC:
350.3
Df Model:                           1

Covariance Type:            nonrobust

======================================================================
========
                 coef    std err          t      P>|t|      [0.025
0.975]
----------------------------------------------------------------------
--------
const         -4.3666      4.628     -0.944      0.351     -13.743
5.010
hptreat        8.8346      6.462      1.367      0.180      -4.259
21.928
======================================================================
========
Omnibus:                        0.353   Durbin-Watson:
2.135
Prob(Omnibus):                  0.838   Jarque-Bera (JB):
```

```
0.524
Skew:                           0.146   Prob(JB):
0.770
Kurtosis:                       2.513   Cond. No.
2.65
======================================================================
========

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is
correctly specified.
Summary of regression results:
                             OLS Regression Results

======================================================================
========
Dep. Variable:     delta_testosterone   R-squared:
0.048
Model:                            OLS   Adj. R-squared:
0.022
Method:                 Least Squares   F-statistic:
1.869
Date:                Mon, 01 Jul 2024   Prob (F-statistic):
0.180
Time:                        11:17:51   Log-Likelihood:
-171.48
No. Observations:                  39   AIC:
347.0
Df Residuals:                      37   BIC:
350.3
Df Model:                           1

Covariance Type:            nonrobust

======================================================================
========
                 coef    std err          t      P>|t|      [0.025
0.975]
----------------------------------------------------------------------
--------
const         -4.3666      4.628     -0.944      0.351     -13.743
5.010
hptreat        8.8346      6.462      1.367      0.180      -4.259
21.928
======================================================================
========
Omnibus:                        0.353   Durbin-Watson:
2.135
Prob(Omnibus):                  0.838   Jarque-Bera (JB):
0.524
```

```
Skew:                           0.146   Prob(JB):
0.770
Kurtosis:                       2.513   Cond. No.
2.65
======================================================================
========

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is
correctly specified.
```

Part 2: Poisson regression models and hypothesis testing

Exercise 2.1: Implement the Poisson Regression Model

The number of crossings **Ni** in a sentence follows a Poisson distribution with rate parameter λ**i**:

Ni ~ Poisson(λi)

The rate parameter λi is given by:

log(λi) = α + βLi

Here's the implementation of the model in Python:

```python
import numpy as np

def poisson_crossings(sentence_length, alpha, beta):
    """
    Function to compute the number of crossings in a sentence using a
Poisson model.

    Parameters:
    sentence_length (int): Length of the sentence (number of words)
    alpha (float): Expected rate of crossings in a sentence of average
length
    beta (float): Change in rate of crossings as a function of
sentence length

    Returns:
    int: Number of crossings
    """
    # Calculate lambda
    log_lambda = alpha + beta * sentence_length
    lambda_i = np.exp(log_lambda)

    # Generate number of crossings from Poisson distribution
    crossings = np.random.poisson(lambda_i)
    return crossings

# Example usage
sentence_length = 11  # Average sentence length
alpha = 0.15  # Example alpha
beta = 0.05  # Example beta

print("Number of crossings:", poisson_crossings(sentence_length,
alpha, beta))
```

```
Number of crossings: 2
```

Exercise 2.2: Generate Prior Predictions

Generate prior predictions of the model for sentences of length 4 under the prior assumptions:

α ~ Normal(0.15,0.1)

β ~ Normal(0.25,0.05)

```
# Generate prior predictions
def generate_prior_predictions(sentence_length, num_samples=1000):
    alpha_prior = np.random.normal(0.15, 0.1, num_samples)
    beta_prior = np.random.normal(0.25, 0.05, num_samples)

    predictions = [poisson_crossings(sentence_length, alpha, beta) for
alpha, beta in zip(alpha_prior, beta_prior)]

    return predictions

# Generate predictions for sentence length 4
sentence_length = 4
prior_predictions = generate_prior_predictions(sentence_length)

print("Prior predictions for sentence length 4:",
prior_predictions[:10])  # Display first 10 predictions
```

```
Prior predictions for sentence length 4: [4, 4, 6, 6, 4, 1, 3, 3, 3,
4]
```

Exercise 2.3: Fit Models M1 and M2 to the Data

Load the data and fit the models M1 and M2 using the statsmodels library. We'll use Poisson regression for this purpose.

```
import pandas as pd
import statsmodels.api as sm

# Load the data
crossings_data = pd.read_csv('/content/crossings_24a167f3-2f8f-4f5c-
bca9-884567bb1c33.csv')
```

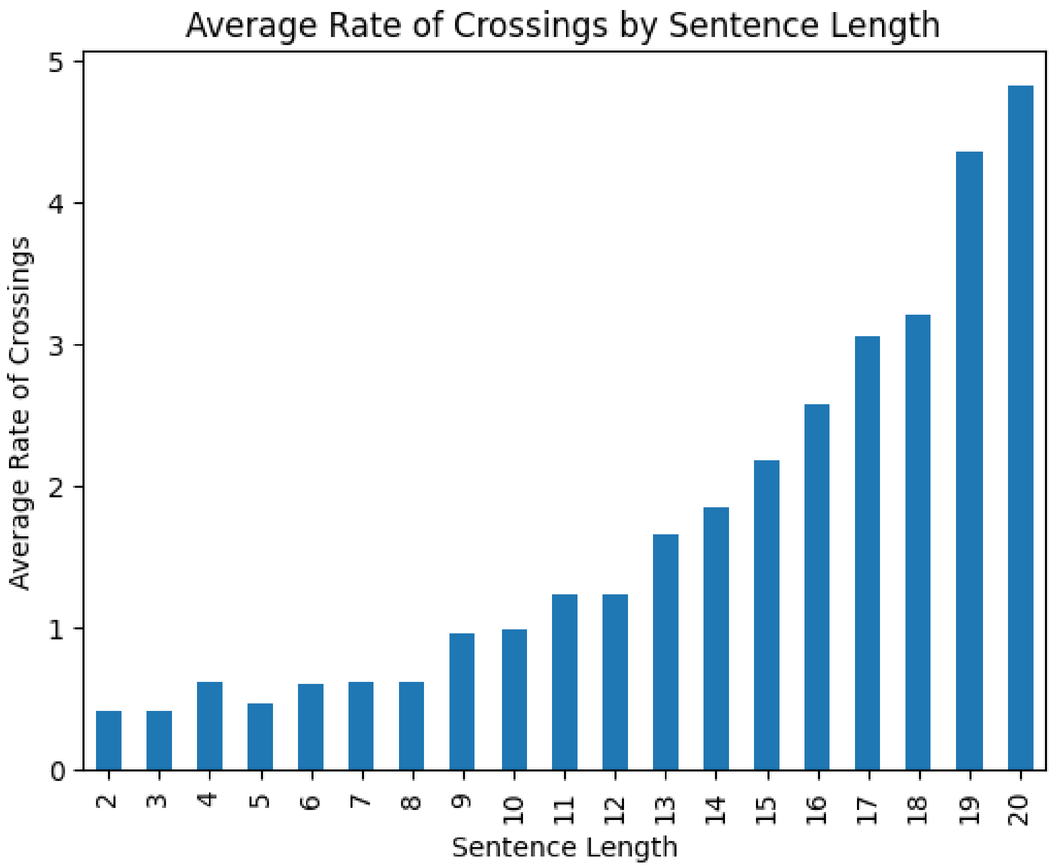
```
## Model M1: Rate of Crossings as a Function of Sentence Length

# Define the dependent variable (number of crossings) and independent
variables (sentence length)
X_M1 = crossings_data[['s.length']]
X_M1 = sm.add_constant(X_M1)  # Add a constant term for the intercept
y_M1 = crossings_data['nCross']

# Fit the Poisson regression model
model_M1 = sm.GLM(y_M1, X_M1, family=sm.families.Poisson()).fit()


##Model M2: Different Rates of Crossings for English and German

# Create an indicator variable for language (0 for English, 1 for
German)
crossings_data['s.id'] = (crossings_data['Language'] ==
'German').astype(int)

# Define the dependent variable and independent variables for Model M2
X_M2 = crossings_data[['s.length', 's.id']]
X_M2['interaction'] = X_M2['s.length'] * X_M2['s.id']  # Interaction
term
X_M2 = sm.add_constant(X_M2)  # Add a constant term for the intercept
y_M2 = crossings_data['nCross']

# Fit the Poisson regression model
model_M2 = sm.GLM(y_M2, X_M2, family=sm.families.Poisson()).fit()

# Summary for Model M1
summary_M1 = model_M1.summary()
print("Summary of Model M1:\n", summary_M1)

# Summary for Model M2
summary_M2 = model_M2.summary()
print("Summary of Model M2:\n", summary_M2)

# Interpretation:
# In Model M1, we look at the coefficients for 'length' to understand
how sentence length affects the number of crossings.
# In Model M2, we look at the coefficients for 'length', 's.id', and
'interaction' to understand how sentence length and language interact
to affect the number of crossings.

Summary of Model M1:
                  Generalized Linear Model Regression Results

======================================================================
========
Dep. Variable:                 nCross   No. Observations:                  
1900
Model:                            GLM   Df Residuals:                      
1898
Model Family:                 Poisson   Df Model:                          
1
Link Function:                    Log   Scale:                          
1.0000
Method:                          IRLS   Log-Likelihood:                
-2813.4
Date:                Tue, 02 Jul 2024   Deviance:                       
2272.1
Time:                        13:08:59   Pearson chi2:                 
2.08e+03
No. Iterations:                     5   Pseudo R-squ. (CS):             
0.6070
Covariance Type:            nonrobust                                         

======================================================================
========
                 coef    std err          z      P>|z|      [0.025      
0.975]
----------------------------------------------------------------------
--------
const         -1.4429      0.061    -23.755      0.000      -1.562      
-1.324
s.length       0.1494      0.004     38.505      0.000       0.142       
0.157
======================================================================
========
Summary of Model M2:
                 Generalized Linear Model Regression Results                  

======================================================================
========
Dep. Variable:                 nCross   No. Observations:                  
1900
Model:                            GLM   Df Residuals:                      
1896
Model Family:                 Poisson   Df Model:                          
3
Link Function:                    Log   Scale:                          
1.0000
Method:                          IRLS   Log-Likelihood:                
-2677.7
Date:                Tue, 02 Jul 2024   Deviance:                       
2000.7
Time:                        13:08:59   Pearson chi2:                 
1.82e+03
```

```
No. Iterations:                     5   Pseudo R-squ. (CS):
0.6593
Covariance Type:            nonrobust

======================================================================
=========
                  coef    std err          z      P>|z|      [0.025
0.975]
----------------------------------------------------------------------
---------
const          -0.9057      0.081    -11.168      0.000      -1.065
-0.747
s.length        0.0970      0.006     17.521      0.000       0.086
0.108
s.id           -1.0257      0.122     -8.433      0.000      -1.264
-0.787
interaction     0.0957      0.008     12.209      0.000       0.080
0.111
======================================================================
=========

<ipython-input-4-29f7cac8ba04>:26: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#
returning-a-view-versus-a-copy
  X_M2['interaction'] = X_M2['s.length'] * X_M2['s.id']  # Interaction
term
```

Exercise 2.4: Quantify evidence for the models M1 and M2 using k-fold cross-validation.

```python
import matplotlib.pyplot as plt

# Visualize the average rate of crossings by sentence length
crossings_data.groupby('s.length')['nCross'].mean().plot(kind='bar')
plt.xlabel('Sentence Length')
plt.ylabel('Average Rate of Crossings')
plt.title('Average Rate of Crossings by Sentence Length')
plt.show()

# Center the predictors
crossings_data['sentence_length_centered'] =
crossings_data['s.length'] - crossings_data['s.length'].mean()
crossings_data['language_ind_centered'] = crossings_data['s.id'] -
crossings_data['s.id'].mean()

## Prepare k-Fold Cross-Validation
from sklearn.model_selection import KFold
```

```
import numpy as np

# Set up k-fold cross-validation
k = 10  # Number of folds
kf = KFold(n_splits=k, shuffle=True, random_state=1)

# Initialize arrays to store log predictive densities
lpd_M1 = np.zeros(k)
lpd_M2 = np.zeros(k)

# Function to calculate log predictive density
def calculate_lpd(model, X_test, y_test):
    predictions = model.get_prediction(X_test)
    mean_pred = predictions.predicted_mean
    lpd = np.sum(y_test * np.log(mean_pred) - mean_pred)  # Poisson
log likelihood
    return lpd

## Fit the Models on Training Data and Evaluate on Test Data
for i, (train_index, test_index) in
enumerate(kf.split(crossings_data)):
    # Split the data into training and test sets
    train_data, test_data = crossings_data.iloc[train_index],
crossings_data.iloc[test_index]

    # Prepare training and test data for Model M1
    X_train_M1 = train_data[['sentence_length_centered']]
    X_train_M1 = sm.add_constant(X_train_M1)
    y_train_M1 = train_data['nCross']

    X_test_M1 = test_data[['sentence_length_centered']]
    X_test_M1 = sm.add_constant(X_test_M1)
    y_test_M1 = test_data['nCross']

    # Fit Model M1
    model_M1 = sm.GLM(y_train_M1, X_train_M1,
family=sm.families.Poisson()).fit()

    # Calculate log predictive density for Model M1
    lpd_M1[i] = calculate_lpd(model_M1, X_test_M1, y_test_M1)

    # Prepare training and test data for Model M2
    X_train_M2 = train_data[['sentence_length_centered',
'language_ind_centered']].copy()
    X_train_M2['interaction'] = X_train_M2['sentence_length_centered']
* X_train_M2['language_ind_centered']
    X_train_M2 = sm.add_constant(X_train_M2)
    y_train_M2 = train_data['nCross']

    X_test_M2 = test_data[['sentence_length_centered',
'language_ind_centered']].copy()
    X_test_M2['interaction'] = X_test_M2['sentence_length_centered'] *
X_test_M2['language_ind_centered']
    X_test_M2 = sm.add_constant(X_test_M2)
    y_test_M2 = test_data['nCross']

    # Fit Model M2
    model_M2 = sm.GLM(y_train_M2, X_train_M2,
family=sm.families.Poisson()).fit()

    # Calculate log predictive density for Model M2
    lpd_M2[i] = calculate_lpd(model_M2, X_test_M2, y_test_M2)

## Compare the Models Using the Log Predictive Densities python
# Calculate the mean log predictive density for both models
mean_lpd_M1 = np.mean(lpd_M1)
mean_lpd_M2 = np.mean(lpd_M2)

# Print the results
print(f'Mean log predictive density for Model M1: {mean_lpd_M1}')
print(f'Mean log predictive density for Model M2: {mean_lpd_M2}')

# Evidence in favor of M2 over M1
evidence = mean_lpd_M2 - mean_lpd_M1
print(f'Evidence in favor of Model M2 over Model M1: {evidence}')
```



```
Mean log predictive density for Model M1: -65.64008731874553
Mean log predictive density for Model M2: -52.13438926842021
Evidence in favor of Model M2 over Model M1: 13.505698050325321
```