Assignment 3

Arpita Chaurasia 220207

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

```
import pandas as pd

# Load the dataset
df_powerpose = pd.read_csv('/content/df_powerpose.csv')

# Calculating the change in testosterone levels after treatment
df_powerpose['delta_testosterone'] = df_powerpose['testm2'] -
df_powerpose['testm1']

# Perform linear regression using statsmodels
import statsmodels.api as sm

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

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

# Fit the linear regression model
model = sm.OLS(B, A).fit()

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

# Interpreting the results
print("Summary of linear regression results:\n", model.summary())

# Interpretation
# A significantly positive coefficient for 'hptreat' suggests that
high power posing is associated
# with a higher increase in testosterone levels compared to low power
posing.
# To assess if this association is statistically significant, we check
the p-value of
# the 'hptreat' coefficient. A value < 0.05 typically indicates
statistical significance.'''
```

```
                            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:                        19:55:44   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 linear 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:                        19:55:44   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

1. A function calculate_crossings is defined wherein lambda_i is calculated using the formula given. After that, crossings are generated as random values samples taken from the poisson distribution of lambda_i calculated above.
2. Usage of the above model demonstrated using example values of parameters sentence_length, alpha and beta. It prints the number of crossings.

```
import numpy as np

def calculate_crossings(sentence_length, alpha, beta):
    """
    Calculates the number of crossings in a sentence using a Poisson
model.

    Parameters:
    sentence_length (int): Length of the sentence in 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.
    """
    # Compute lambda_i (rate parameter for Poisson distribution) from
log of lambda_i,
    # which is (alpha + (beta*sentence_length))
    lambda_i = np.exp(alpha + (beta * sentence_length))

    # Generating number of crossings from Poisson distribution of
lambda_i, by taking random samples.
    crossings = np.random.poisson(lambda_i)

    return crossings

# Example usage of above model
sentence_length = 11  # sentence length
alpha = 0.2 # Example alpha (expected crossings for average length
sentence)
beta = 0.03 # Example beta (change in crossings with sentence length)

# Calculate and print the number of crossings
num_crossings = calculate_crossings(sentence_length, alpha, beta)
print("Number of crossings:", num_crossings)
```

```
Number of crossings: 4
```

Exercise 2.2

We use the given prior data:

α ~ Normal(0.15,0.1)

β ~ Normal(0.25,0.05)

and the sentence length = 4, as given. Prior predictions are calculated using the same calculate_crossings function. The result of the function, ni is the number of crossings sampled from the poisson distribution of lambda_i. These results are stored in a predictions[ ] array, which gives the required prior predictions for given sentence length of 4.

```
import numpy as np

# Function to calculate the expected rate parameter lambda_i from
alpha and beta, same as part 2.1
def calculate_crossings(alpha, beta, sentence_length):
   # Calculate lambda_i using equation (2) in question,
     lambda_i = np.exp(alpha + beta * sentence_length)
     # Draw samples from Poisson distribution with lambda_i
     ni = np.random.poisson(lambda_i)
     return ni


# Function to generate prior predictions for crossing dependencies
def generate_prior_predictions(sentence_length, samples=10000):
    # Define prior distributions for alpha and beta
    alpha_prior = np.random.normal(0.15, 0.1, samples)  # Prior for
alpha
    beta_prior = np.random.normal(0.25, 0.05, samples)  # Prior for
beta
    # Generate predictions based on the Poisson distribution
    predictions = []
    for alpha, beta in zip(alpha_prior, beta_prior):
        #The samples generated in calculate_crossings are the prior
predictions of the model.
        ni = calculate_crossings(alpha, beta, sentence_length)
        predictions.append(ni)
    return predictions

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

# Display the results
print(f"Prior predictions of the model for sentence length
{sentence_length}:")
print(prior_predictions[:20])  # Display first 20 predictions
```

```
Prior predictions of the model for sentence length 4:
[2, 4, 2, 3, 5, 5, 2, 2, 2, 1, 1, 4, 2, 5, 2, 1, 4, 5, 3, 4]
```

Exercise 2.3

We defined separate variables for both models M1 and M2

X_M1: stores sentence length

Y_M1: stores number of crossings

X_M2: stores sentence length along with an id indicating language(0 for English, 1 for German)

Y_M2: stores number of crossings

Model M1:

We use sentence length data independent of language to see its effect on number of crossings.

Model M2: We use sentence length data along with the language data to see how both of them interact, by observing their effect on number of crossings.

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

# Load the data
crossings_data = pd.read_csv('/content/crossings.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 for Model M1
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.loc[:, 'interaction'] = crossings_data[['s.length', 's.id']]
X_M2.loc[:, 'interaction'] = X_M2['s.length'] * X_M2['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']
```

```r
# Fit the Poisson regression model for Model M2
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 's.length' to
understand how sentence length affects the number of crossings.
# In Model M2, we look at the coefficients for 's.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:                        07:55:15   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:                        07:55:15   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
==============================================================================
=========
```

Effect of Sentence Length: Both models show a positive and significant effect of sentence length on the number of crossings (nCross).

Language Difference: Model M2 explicitly captures differences between English and German sentences (s.id), showing that German sentences tend to have fewer crossings than English sentences.

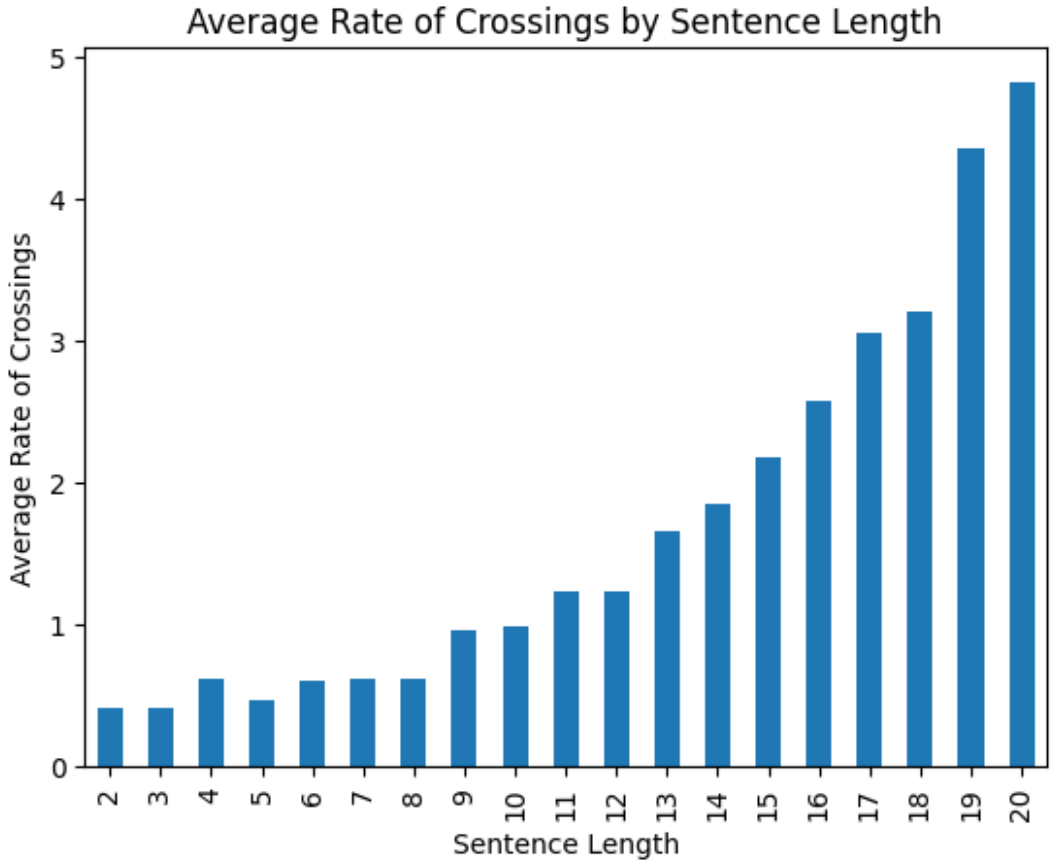
Interaction Effect: Model M2 includes an interaction term (interaction), indicating that the effect of sentence length on crossings varies depending on the language.

Exercise 2.4

```
import pandas as pd
import numpy as np
import statsmodels.api as sm
from sklearn.model_selection import KFold
import matplotlib.pyplot as plt

# Load the data
crossings_data = pd.read_csv('/content/crossings.csv')

# Visualize the average rate of crossings by sentence length by bar
graph
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 sentence length and language id
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()

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

# Initialize arrays to store log predictive densities(lpd)
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)             # test data
taken
    mean_pred = predictions.predicted_mean                 # mean of
test data
    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 tata
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]

    ####  M1  ####

    # 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)

    ####  M2  ####

    # 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)

# Calculate the mean log predictive density for both models M1 & M2
mean_lpd_M1 = np.mean(lpd_M1)
mean_lpd_M2 = np.mean(lpd_M2)

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: -65.38386264638359
Evidence in favor of Model M2 over Model M1: 0.25622467236193813
```

Visualizing Data: We start by visualizing the average rate of crossings by sentence length to get an initial understanding of the data (bar graph of the same).

Centering Predictors: We center the predictors s.length(sentence length) and s.id (language id) to ensure they have a mean of zero, which is a standard preprocessing step.

k-fold Cross-Validation Setup: We set up a 10-fold cross-validation (k=10) using KFold from sklearn.model_selection. This splits the data into 10 folds for training and testing.

Model Fitting and Evaluation: Inside the cross-validation loop:

We fit Model M1 and Model M2 using sm.GLM from statsmodels.api, specifying Poisson family for the likelihood. For each fold, we calculate the log predictive density (lpd) using a helper function calculate_lpd that computes the Poisson log likelihood based on predicted means.

Comparing Models: After cross-validation, we compute the mean log predictive density for both Model M1 and Model M2 across all folds.

Evidence Calculation: Finally, we calculate the evidence in favor of Model M2 over Model M1 by subtracting their mean log predictive densities.