2.2-section-result

February 14, 2025

1 Section 2 - Controlling for confounding factors

1.1 Example 2.2

Application 2.2: Determining if protein size explains the previously noted (**Example 1.2**) association of entanglements with disease in humans

- Larger proteins may be more prone to misfolding and thus causing a disease regardless of their entanglement status
- In this example, you will use the code below to carry out a logistic regression analysis of the relationship between disease and entanglement while treating protein size as a confounding factor

1.1.1 Step 0 - Load libraries

[1]:
```
import pandas as pd
import statsmodels.api as sm
import numpy as np
```

1.1.2 Step 1 - Load and explore the data

- We will reuse the same information from **Example 1.2** but load a new version that includes information about protein length

[2]:
```
# "data5" is a pandas DataFrame object
data_path = "/home/jovyan/data-store/data/iplant/home/shared/NCEMS/
 ↪BPS-training-2025/"
data5     = pd.read_csv(data_path + "entanglement-disease-association-length.
 ↪csv")

# print summary information
print ("Create a quick summary of the DataFrame:\n")
data5.info()

print ("\nPrint the first 10 rows of the DataFrame:\n")
data5.head(10)
```

```
Create a quick summary of the DataFrame:
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5365 entries, 0 to 5364
Data columns (total 4 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   gene            5365 non-null   object
 1   entanglement    5365 non-null   object
 2   disease-linked  5365 non-null   object
 3   Length          5365 non-null   int64
dtypes: int64(1), object(3)
memory usage: 167.8+ KB

Print the first 10 rows of the DataFrame:
```

[2]:

	gene	entanglement	disease-linked	Length
0	A0A075B759	Yes	No	164
1	A0A087WV53	No	No	238
2	A0A087X1C5	Yes	No	515
3	A0A096LP55	No	No	91
4	A0A0B4J2A2	Yes	No	164
5	A0A0B4J2D5	Yes	No	268
6	A0AVF1	No	Yes	554
7	A0AVI4	No	No	362
8	A0AVT1	Yes	No	1052
9	A0PJE2	Yes	No	317

- This dataset uses `Yes` and `No` rather than binary `1` and `0` - we will need to recode the columns `Entanglement` and `disease-linked` to be binary integers

1.1.3 Step 2 - Prepare for analysis

[3]:
```
# create two new columns with values recoded from Yes and No strings to binary 1 and 0 integers
recode_map                          = {"Yes": 1, "No": 0}
data5['disease-linked-binary'] = data5['disease-linked'].map(recode_map)
data5['entanglement-binary']        = data5['entanglement'].map(recode_map)

# add column of 1's corresponding to the intercept
data5['intercept'] = 1

# print a summary of the updated DataFrame
print ("\nHere is the updated DataFrame:\n")
data5.head(10)
```

```
Here is the updated DataFrame:
```

[3]:

	gene	entanglement	disease-linked	Length	disease-linked-binary	\
0	A0A075B759	Yes	No	164	0	
1	A0A087WV53	No	No	238	0	
2	A0A087X1C5	Yes	No	515	0	
3	A0A096LP55	No	No	91	0	
4	A0A0B4J2A2	Yes	No	164	0	
5	A0A0B4J2D5	Yes	No	268	0	
6	A0AVF1	No	Yes	554	1	
7	A0AVI4	No	No	362	0	
8	A0AVT1	Yes	No	1052	0	
9	A0PJE2	Yes	No	317	0	

	entanglement-binary	intercept
0	1	1
1	0	1
2	1	1
3	0	1
4	1	1
5	1	1
6	0	1
7	0	1
8	1	1
9	1	1

- With these three new columns of `disease-linked-binary`, `entanglement-binary`, and `intercept` we are ready to the run the analysis

1.1.4 Step 3 - Run the analysis

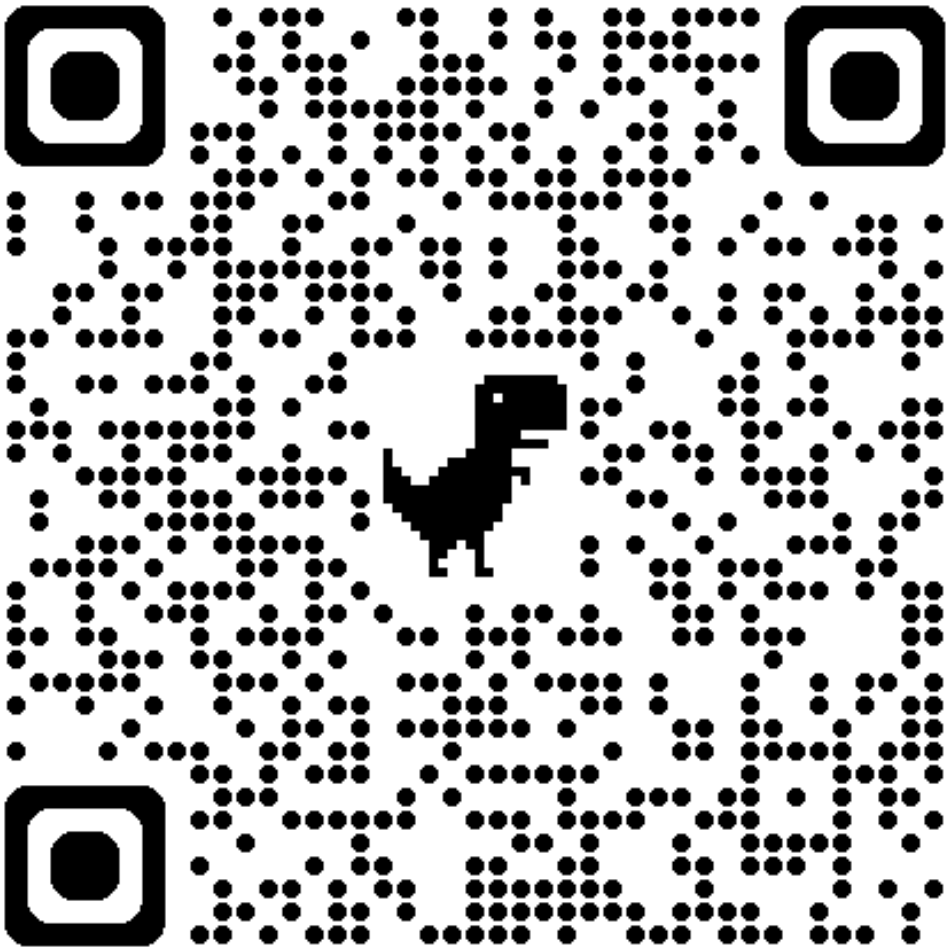
[4]:
```
# make two X datasets, one including the confounder and one excluding it

# X1 includes only the feature
X1 = data5[['intercept', 'entanglement-binary']]

# X2 includes both the feature and the confounder
X2 = data5[['intercept', 'entanglement-binary', 'Length']]

# define the dependent variable (i.e., the outcome)
y = data5['disease-linked-binary']

# create two LogisticRegression() objects, fit the models, get coefficients,␣
 ↪and compute odds ratios

# model1 will not include the confounder
model1  = sm.Logit(y, X1)
result1 = model1.fit(disp = 0)
```

```
# print a summary of result1
print ("\nResults when confounding factor IS NOT included:\n")

# get a summary of the results
odds_ratios = pd.DataFrame({"Coefficient": result1.params,
                            "OR"         : np.exp(result1.params),
                            "Lower CI"   : np.exp(result1.conf_int()[0]),
                            "Upper CI"   : np.exp(result1.conf_int()[1]),
                            "p-value"    : result1.pvalues}).
 ↪drop(index="intercept", errors="ignore")

# print the results
print (odds_ratios.round(3), "\n")

# model1 will not include the confounder
model2  = sm.Logit(y, X2)
result2 = model2.fit(disp = 0)

# print a summary of result1
print ("\nResults when confounding factor IS included:\n")

# get a summary of the results
odds_ratios = pd.DataFrame({"Coefficient": result2.params,
                            "OR"         : np.exp(result2.params),
                            "Lower CI"   : np.exp(result2.conf_int()[0]),
                            "Upper CI"   : np.exp(result2.conf_int()[1]),
                            "p-value"    : result2.pvalues}).
 ↪drop(index="intercept", errors="ignore")

# print the odds ratio
print (odds_ratios.round(3), "\n")
```

```
Results when confounding factor IS NOT included:

                     Coefficient     OR  Lower CI  Upper CI  p-value
entanglement-binary        0.823  2.278     2.014     2.577      0.0


Results when confounding factor IS included:

                     Coefficient     OR  Lower CI  Upper CI  p-value
entanglement-binary        0.635  1.887     1.657     2.149      0.0
Length                     0.001  1.001     1.001     1.001      0.0
```

1.1.5 Step 4 - Interpreting the results

- Use the quiz question at the QR code/link below to test your understanding



Quiz Link