This executable notebook will help you complete Pset 3.

If you haven't used Colab before, it's very similar to Jupyter / IPython / R Notebooks: cells containing Python code can be interactively run, and their outputs will be interpolated into this document. If you haven't used any such software before, we recommend taking a quick tour of Colab.

Now, a few Colab-specific things to note about execution before we get started:

- Google offers free compute (including GPU compute!) on this notebook, but *only for a limited time*. Your session will be automatically closed after 12 hours. That means you'll want to finish within 12 hours of starting, or make sure to save your intermediate work (see the next bullet).
- You can save and write files from this notebook, but they are *not guaranteed to persist*. For this reason, we'll mount a Google Drive account and write to that Drive when any files need to be kept permanently.
- You should keep this tab open until you're completely finished with the notebook. If you close the tab, your session will be marked as "Idle" and may be terminated.

Getting started

First, make a copy of this notebook so you can make your own changes. Click *File -> Save a copy in Drive*.

What you need to do

Read through this notebook and execute each cell in sequence, making modifications and adding code where necessary. You should execute all of the code as instructed, and make sure to write code or textual responses wherever the text **TODO** shows up in text and code cells.

When you're finished, choose *File -> Download .ipynb*. You will upload this `.ipynb` file as part of your submission.

1) Logistic Regression

Background: logistic regression for binomial ordering preferences

We'll walk you through the example of logistic regression that we covered during class, where we took a dataset of binomial expressions and inferred the relative strengths of the short-before-long and frequent-before-infrequent ordering preferences. We first load the dataset, which consists of a number of binomial expressions each of which was observed once in a sample of the Brown corpus, in the order given in the dataset. In this dataset, `Syl` and `Freq` respectively denote whether the observed ordering matches the preference (an entry of `1`), violates the preference (an entry of `-1`), or is irrelevant for the preference (an entry of `0`, indicating that either ordering would satisfy the preference). `Percept` indicates matching or violation of the perceptual markedness preference, and `Response` is a dummy variable whose value is always `1`, which we will use in fitting the logistic regression model.

In [ ]:
```
import statsmodels.api as sm
import pandas as pd
import numpy as np
d = pd.read_csv("https://gist.githubusercontent.com/scaperex/b577698c3f497f43df453d28c
d
```

Out[ ]:

	Binomial	Percept	Syl	Freq	Response
1	abused and neglected	0	1	1	1
2	accept and hire	0	0	1	1
3	achieved and maintained	0	0	1	1
4	actively and continually	0	1	-1	1
5	adding and using	0	0	-1	1
...	...	...	...	...	...
326	wide and varied	0	1	1	1
327	wiry and fit	0	-1	-1	1
328	WordPerfect and Lotus	0	-1	-1	1
329	worried and troubled	0	0	1	1
330	young and energetic	0	1	1	1

330 rows × 5 columns

Recall that logistic regression involves the following equations for predictors $\{X_i\}$:

$\eta = \sum_i \beta_i X_i$ (the **linear predictor**)

$P(\text{outcome=success}) = \frac{e^\eta}{1+e^\eta}$ (outcomes are Bernoulli distributed around the mean resulting from a logistic transformation of the linear predictor)

We have two predictors: X_1 is `Syl` and X_2 is `Freq`. We use the `statsmodels` Python package to fit this logistic regression model to our dataset and infer the parameter weights $\{\beta_i\}$, which correspond to the preference strengths. In `statsmodels`, as in most software packages implementing logistic regression, it is a convention that the numeric coding of the outcome or response is `1` for "success" and `0` otherwise. Also as in most software packages for logistic regression, we use matrix formats to represent the response & predictors: so if there are M predictors and N observations, then the predictor set is represented as an $M \times N$ matrix and the response variable is represented as a $1 \times N$ matrix (effectively a length-N

vector). We split our dataset into predictor and response matrices, and then fit a logistic regression model.

(In `statsmodels`, as with many statistical software packages, logistic regression is implemented as a special case of the more general framework of generalized linear models (GLMs), which is why the third line of the below cell looks the way it does. We won't be covering GLMs in this class, but you may encounter them in other statistics classes or, perhaps less likely, in machine-learning classes.)

In [ ]:
```
x = d[["Syl","Freq"]]
y = d[["Response"]]
m = sm.GLM(y,x,family=sm.families.Binomial()) # first argument is response, second arg
m_results = m.fit()
print(m_results.summary())
```

```
                 Generalized Linear Model Regression Results                  
==============================================================================
Dep. Variable:               Response   No. Observations:                  330
Model:                            GLM   Df Residuals:                      328
Model Family:                Binomial   Df Model:                            1
Link Function:                  Logit   Scale:                          1.0000
Method:                          IRLS   Log-Likelihood:                -213.95
Date:                Wed, 17 May 2023   Deviance:                       427.90
Time:                        10:42:07   Pearson chi2:                     330.
No. Iterations:                     4   Pseudo R-squ. (CS):             -2.657
Covariance Type:            nonrobust                                         
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
Syl            0.4825      0.154      3.131      0.002       0.180       0.784
Freq           0.4019      0.122      3.296      0.001       0.163       0.641
==============================================================================
```

The `coef` results of `0.48` for `Syl` and `0.40` match those we covered in class.

How well are we able to predict the ordering of a binomial we haven't previously seen will occur in? To estimate this, we'll create a random 80/20 train/test split of our binomials data, estimate our logistic regression weights using the training dataset, and then see how often our prediction is successful ($P(success) > 0.5$ for the observed ordering of the test-set binomial). First we create our train/test split:

In [ ]:
```
import math, random
N = d.shape[0]
N_train = math.floor(N*4/5)
idx = list(range(N))
random.seed(3) # so that results will be reproducible from run to run
random.shuffle(idx)
idx_train = idx[0:N_train]
idx_test = idx[N_train:N]
print(idx_train)
print(idx_test)
d_train = d.iloc[idx_train]
d_test = d.iloc[idx_test]
print(d_train)
```

```
[272, 78, 13, 285, 211, 48, 188, 291, 292, 191, 244, 46, 233, 311, 139, 308, 70, 250,
287, 222, 192, 264, 54, 252, 163, 269, 180, 17, 238, 147, 38, 22, 220, 280, 41, 99, 2
39, 299, 288, 89, 135, 95, 146, 231, 42, 131, 312, 207, 224, 302, 138, 249, 289, 3, 2
74, 229, 142, 62, 12, 263, 171, 51, 124, 329, 165, 31, 120, 88, 226, 29, 304, 201, 3
6, 149, 58, 205, 122, 170, 127, 65, 102, 190, 0, 25, 230, 87, 206, 52, 169, 91, 97, 2
09, 182, 101, 59, 123, 193, 37, 268, 16, 254, 44, 144, 126, 293, 134, 105, 115, 130,
214, 200, 23, 73, 114, 107, 103, 40, 266, 159, 4, 166, 100, 28, 277, 283, 72, 113, 5
5, 325, 20, 43, 112, 57, 175, 82, 186, 24, 245, 261, 270, 61, 290, 56, 128, 232, 322,
265, 318, 204, 327, 177, 221, 275, 185, 47, 93, 260, 300, 228, 151, 76, 116, 219, 64,
94, 168, 178, 181, 294, 125, 237, 155, 173, 90, 314, 85, 160, 328, 321, 258, 161, 24
1, 262, 212, 5, 216, 148, 251, 2, 217, 140, 195, 257, 326, 284, 256, 234, 295, 184, 1
62, 110, 286, 235, 158, 26, 271, 174, 152, 164, 313, 133, 117, 213, 324, 35, 92, 80,
86, 255, 279, 39, 67, 156, 74, 104, 50, 8, 296, 153, 27, 19, 1, 79, 9, 60, 129, 71, 2
36, 225, 141, 84, 150, 183, 96, 248, 194, 157, 319, 11, 30, 315, 106, 196, 109, 75, 1
0, 305, 210, 34, 176, 45, 247, 246, 301]
[63, 145, 136, 108, 53, 167, 83, 143, 14, 172, 306, 208, 273, 179, 197, 259, 215, 29
8, 223, 316, 111, 253, 69, 18, 49, 187, 68, 227, 202, 218, 198, 323, 137, 15, 154, 2
1, 81, 32, 7, 199, 267, 307, 118, 77, 203, 243, 281, 276, 317, 98, 119, 282, 132, 24
0, 6, 310, 33, 297, 320, 242, 309, 189, 66, 278, 303, 121]
                      Binomial  Percept  Syl  Freq  Response
273          stained and waxed        0    0     1         1
79   Czechoslovakia and Hungary        0   -1    -1         1
14              anger and spite        0   -1     1         1
286    swiftly and aggressively        0    1    -1         1
212                pull and tug        0    0     1         1
..                         ...      ...  ...   ...       ...
177         muddling and chilling        0    0    -1         1
46          check and discipline        0    1     1         1
248         sewing and quilting         0    0     1         1
247        sewing and needlework        0    1     1         1
302           totally and morally        0    0     1         1

[264 rows x 5 columns]
```

And now we train a logistic model on only the training set, predict success probability for the observed binomials in the test set, and see how often we "succeed":

In [ ]:

```
x_train = d_train[["Syl","Freq"]]
y_train = d_train[["Response"]]
m = sm.GLM(y_train,x_train,family=sm.families.Binomial()) # first argument is response
m_results = m.fit()
print(m_results.summary())
x_test = d_test[["Syl","Freq"]]
y_predicted = m_results.predict(x_test)
np.mean(y_predicted>0.5)
```

```
                 Generalized Linear Model Regression Results
==============================================================================
Dep. Variable:               Response   No. Observations:                  264
Model:                            GLM   Df Residuals:                      262
Model Family:                Binomial   Df Model:                            1
Link Function:                  Logit   Scale:                          1.0000
Method:                          IRLS   Log-Likelihood:                -170.38
Date:                Wed, 17 May 2023   Deviance:                       340.77
Time:                        10:42:13   Pearson chi2:                     264.
No. Iterations:                     4   Pseudo R-squ. (CS):             -2.636
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
Syl            0.5005      0.171      2.931      0.003       0.166       0.835
Freq           0.4203      0.136      3.102      0.002       0.155       0.686
==============================================================================
```

Out[ ]:

```
0.6363636363636364
```

The answer: apparently somewhat better than 50/50 chance!

Another measure of how well a model fits a dataset is the log-likelihood it assigns to the data.

In [ ]:

```python
sum(np.log(y_predicted)) # large (less negative) values indicate better fit.
```

Out[ ]:

```
-43.59055060217463
```

Accuracy and Log-likelihood

Note, in the binary classification case, accuracy is defined as:

$$Acc = \frac{1}{N}\sum_i 1\{\hat{y_i} == y_i\}$$

Where

$$\hat{y_i} = 1 \textbf{ if } p(\hat{x_i}) > 0.5 \textbf{ else } 0$$

And log likelihood is defined as:

$$L = \sum_i [y_i \cdot log(p(x_i)) + (1 - y_i) \cdot log(1 - p(x_i))]$$

A new application of logistic regression: the dative alternation

The work you need to do for this pset involves applying logistic regression to a new case, the **dative alternation**, which we studied in a previous pset. We will use the `dative` dataset from Bresnan et al. (2007). First we load the dataset:

In [ ]:

```python
dat = pd.read_csv("https://gist.githubusercontent.com/scaperex/278815a736401d36021aa9f
dat
```

Out[ ]:

	Unnamed: 0	Speaker	Modality	Verb	SemanticClass	LengthOfRecipient	AnimacyOfRec	DefinOf
0	1	NaN	written	feed	t	1	animate	def
1	2	NaN	written	give	a	2	animate	def
2	3	NaN	written	give	a	1	animate	def
3	4	NaN	written	give	a	1	animate	def
4	5	NaN	written	offer	c	2	animate	def
...	...	...	...	...	...	...	...	
3258	3258	S1190	spoken	tell	c	1	animate	def
3259	3259	S1423	spoken	give	a	1	animate	def
3260	3260	S1680	spoken	give	a	4	animate	indef
3261	3261	S1680	spoken	give	a	1	inanimate	def
3262	3262	S1023	spoken	pay	a	1	animate	def

3263 rows × 16 columns

We see that it uses text values for some of the variables we are interested in (the response variable `RealizationOfRecipient`, and the variables expressing length and pronominality of theme and object). We create numeric versions of these variables, arbitrarily coding a double object outcome as `1` ("success") and a prepositional dative outcome as `0`.

In [ ]:

```
dat["Response"] = [1 if x =="NP" else 0 for x in dat["RealizationOfRecipient"]]
dat["RecPro"] = [1 if x =="pronominal" else 0 for x in dat["PronomOfRec"]]
dat["ThemePro"] = [1 if x =="pronominal" else 0 for x in dat["PronomOfTheme"]]
dat[["RealizationOfRecipient","Response","PronomOfRec","RecPro","PronomOfTheme","Theme
```

Out[ ]:

	RealizationOfRecipient	Response	PronomOfRec	RecPro	PronomOfTheme	ThemePro
0	NP	1	pronominal	1	nonpronominal	0
1	NP	1	nonpronominal	0	nonpronominal	0
2	NP	1	nonpronominal	0	nonpronominal	0
3	NP	1	pronominal	1	nonpronominal	0
4	NP	1	nonpronominal	0	nonpronominal	0
...	...	...	...	...	...	...
3258	NP	1	pronominal	1	pronominal	1
3259	NP	1	pronominal	1	nonpronominal	0
3260	PP	0	nonpronominal	0	nonpronominal	0
3261	NP	1	pronominal	1	nonpronominal	0
3262	NP	1	pronominal	1	nonpronominal	0

3263 rows × 6 columns

In [ ]:

```
## TODO: create numeric variables for PronomOfTheme
dat["logLengthOfTheme"]=np.log2(dat["LengthOfTheme"])
dat["logLengthOfRecipient"]=np.log2(dat["LengthOfRecipient"])
```

To capture the possibility of an overall preference for one construction or the other, we add an "intercept" term to the logistic regression model, by creating a new `Dummy` variable in the data frame. We then fit a baseline model using only the intercept and find that there is an overall majority preference for the **DO** realization in this dataset (the intercept's fitted weight is greater than 0). We also see that the intercept-only model simply recapitulates the sample mean.

In [ ]:

```
dat["Dummy"] = 1
x = dat[["Dummy"]]
y = dat[["Response"]]
m = sm.GLM(y,x,family=sm.families.Binomial()) # first argument is response, second arg
m_results = m.fit()
print(m_results.summary())
print("Predicted proportion of DO outcomes based on fitted intercept-only model:", rou
print("Proportion of data with DO outcome:", round(np.mean(y["Response"]),4)) # same c
```

```
                 Generalized Linear Model Regression Results
==============================================================================
Dep. Variable:               Response   No. Observations:                 3263
Model:                            GLM   Df Residuals:                     3262
Model Family:                Binomial   Df Model:                            0
Link Function:                  Logit   Scale:                          1.0000
Method:                          IRLS   Log-Likelihood:                -1870.5
Date:                Wed, 17 May 2023   Deviance:                       3741.1
Time:                        10:42:29   Pearson chi2:                 3.26e+03
No. Iterations:                     4   Pseudo R-squ. (CS):         -2.220e-16
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
Dummy          1.0450      0.040     26.189      0.000       0.967       1.123
==============================================================================
Predicted proportion of DO outcomes based on fitted intercept-only model: 0.7398
Proportion of data with DO outcome: 0.7398
```

Task: In the below code boxes, complete the five parts of the problem specified in the pset PDF.

In [ ]:

```
## TODO: define and implement an 80/20 train/test random split of the "dative" dataset
#just like we did before
N = dat.shape[0]
N_train = math.floor(N*4/5)
idx = list(range(N))
random.seed(3)
random.shuffle(idx)
idx_train = idx[0:N_train]
idx_test = idx[N_train:N]
print(idx_train)
print(idx_test)
d_train = dat.iloc[idx_train]
d_test = dat.iloc[idx_test]
print(d_train)
```

```
[942, 2352, 2947, 2247, 2138, 990, 2484, 226, 2762, 476, 3122, 2996, 592, 1809, 1672,
1014, 3162, 480, 713, 2706, 2214, 1169, 76, 1244, 1586, 1610, 2900, 1534, 2037, 3193,
1622, 2326, 554, 597, 2467, 1090, 1890, 1461, 2986, 2957, 5, 1478, 1192, 2622, 107, 3
025, 346, 1702, 2862, 2013, 375, 320, 1756, 1667, 3010, 281, 1281, 2932, 1895, 1448,
619, 1451, 463, 1959, 743, 374, 2348, 2021, 2213, 2350, 672, 2687, 52, 1171, 3024, 31
75, 602, 1303, 1714, 229, 1682, 1561, 2647, 1314, 1038, 2621, 2184, 2809, 1679, 1370,
2045, 1036, 1658, 304, 3166, 3146, 2497, 473, 845, 938, 2486, 1458, 2613, 1491, 680,
870, 345, 92, 2191, 67, 2009, 2239, 1579, 501, 489, 1888, 755, 2260, 778, 828, 2182,
580, 567, 1466, 1360, 568, 1070, 1084, 897, 2361, 717, 2222, 1190, 472, 276, 3113, 27
70, 1071, 670, 458, 2976, 3198, 1212, 1806, 779, 328, 2415, 647, 2733, 2340, 540, 270
8, 1347, 2211, 1703, 3208, 917, 186, 2987, 1008, 178, 922, 2119, 130, 1273, 493, 173
6, 2693, 1429, 2519, 2704, 1674, 2029, 389, 3012, 1495, 1908, 270, 2386, 2812, 2757,
2546, 1494, 86, 1738, 2840, 626, 2960, 1647, 1420, 1074, 2568, 71, 3082, 2457, 878, 8
94, 2857, 524, 650, 1548, 3234, 2952, 2367, 3004, 913, 2220, 481, 179, 1930, 3058, 32
06, 1801, 1239, 560, 601, 2778, 3152, 51, 1934, 80, 1419, 3124, 1449, 2985, 2000, 145
9, 1172, 3205, 16, 101, 1053, 3227, 2530, 2395, 1979, 159, 589, 1859, 2063, 33, 2359,
999, 264, 700, 729, 2620, 2631, 335, 2252, 1925, 223, 2807, 3130, 614, 788, 1608, 191
9, 3112, 1949, 1208, 2040, 2389, 1573, 2269, 1488, 1138, 2018, 692, 2236, 228, 2911,
492, 275, 289, 2581, 965, 2031, 2165, 48, 256, 952, 777, 477, 806, 1918, 1807, 517, 8
37, 1309, 1588, 2152, 380, 1422, 35, 2874, 2611, 3221, 427, 1657, 129, 2468, 2053, 19
66, 2278, 2307, 2602, 142, 1662, 2884, 2136, 158, 623, 3062, 872, 1163, 2059, 2357, 2
249, 564, 2103, 2010, 1717, 333, 2580, 2, 3032, 2977, 2085, 2477, 313, 1017, 266, 21
0, 2163, 3213, 2287, 2560, 2185, 1136, 1137, 482, 2674, 633, 1020, 2248, 3037, 2893,
790, 3042, 2561, 1433, 83, 2654, 1577, 1991, 841, 1956, 49, 1560, 1182, 133, 1387, 88
9, 1877, 1206, 2913, 411, 2323, 309, 2301, 40, 2079, 1604, 516, 1858, 2109, 1769, 322
8, 2093, 203, 1469, 3050, 2015, 1899, 190, 1256, 2670, 1629, 723, 852, 2965, 1443, 88
1, 3001, 2151, 2387, 2848, 2132, 2794, 1252, 337, 1157, 997, 1332, 2699, 1840, 2038,
1900, 1928, 1364, 1423, 1726, 737, 1855, 2853, 314, 3023, 156, 752, 3091, 260, 1945,
2265, 2155, 1815, 673, 42, 948, 2543, 2288, 194, 321, 659, 279, 1845, 2906, 1875, 51
8, 2968, 720, 829, 2513, 3143, 2870, 3141, 2306, 2973, 523, 25, 2172, 1824, 2774, 122
2, 3109, 2671, 1162, 1254, 2759, 2429, 3167, 3051, 1166, 2412, 61, 2972, 2502, 34, 1
8, 1857, 498, 10, 31, 2842, 908, 2012, 452, 3054, 55, 2653, 1396, 2104, 1406, 1446, 2
299, 468, 932, 1253, 3138, 553, 562, 2096, 716, 1516, 2090, 2902, 1315, 3014, 607, 3
9, 2131, 84, 3071, 2228, 214, 2479, 2149, 3173, 1799, 1751, 1987, 1271, 418, 479, 318
9, 100, 3006, 2332, 164, 1215, 2399, 954, 1002, 1559, 970, 3084, 2432, 301, 1829, 43
5, 216, 1340, 2092, 2558, 3045, 1585, 2464, 884, 26, 1383, 635, 70, 1606, 307, 236, 3
231, 1880, 1377, 2661, 2378, 242, 928, 1508, 2576, 2343, 862, 1931, 2139, 3168, 2106,
2701, 2170, 1120, 1112, 3254, 1278, 994, 1632, 1116, 391, 1179, 1517, 986, 1384, 900,
515, 404, 3169, 88, 2471, 1768, 2761, 2821, 3078, 230, 1701, 2772, 830, 1745, 1069, 2
372, 2746, 1410, 2002, 253, 998, 730, 3180, 3118, 2135, 2368, 735, 3184, 1685, 2865,
72, 1080, 1616, 738, 1089, 3255, 2933, 2554, 2590, 1234, 400, 118, 2619, 431, 2122, 1
800, 2652, 1004, 1797, 763, 666, 984, 3083, 2143, 165, 1661, 1219, 1940, 394, 1345, 1
76, 1151, 1503, 2073, 1307, 2102, 937, 2485, 2955, 1115, 2593, 3212, 32, 2820, 1784,
2681, 1430, 1331, 1841, 839, 3114, 1348, 2940, 603, 382, 3178, 2055, 2244, 1083, 112
9, 3204, 2839, 2797, 879, 2698, 1780, 1654, 2114, 3072, 1571, 2100, 1916, 1500, 1942,
1524, 1530, 1553, 902, 1872, 724, 2544, 2714, 677, 2212, 1636, 1350, 2575, 211, 2917,
2642, 2312, 390, 261, 1316, 1589, 1576, 3225, 541, 3216, 1788, 2373, 2450, 269, 2832,
1202, 2261, 1675, 2245, 1338, 2117, 162, 629, 294, 1641, 710, 378, 1255, 1099, 1139,
2969, 2624, 712, 929, 2818, 1127, 1113, 2992, 1590, 2454, 377, 1262, 2057, 1000, 159
9, 3191, 2735, 2512, 1965, 1723, 824, 2046, 1729, 3171, 1018, 3262, 791, 2651, 2390,
273, 3217, 2080, 918, 3241, 2243, 1498, 474, 1405, 1823, 947, 686, 1374, 2204, 2147,
385, 1329, 771, 945, 2255, 573, 2264, 584, 2074, 1178, 2345, 1031, 1187, 769, 2503, 3
116, 2501, 1135, 521, 2297, 810, 1648, 594, 1232, 2567, 2805, 1985, 161, 485, 975, 18
3, 1149, 3210, 2667, 955, 2011, 2953, 3117, 3202, 65, 1054, 2834, 1079, 1753, 2509, 5
08, 1876, 598, 1341, 1375, 2240, 2722, 1591, 326, 2788, 2926, 45, 2047, 2927, 308, 12
16, 332, 2526, 103, 687, 1457, 2979, 868, 1249, 1664, 2703, 1851, 1802, 2137, 2974, 3
002, 1746, 1752, 182, 1808, 2476, 1091, 574, 136, 2869, 2043, 2514, 1999, 728, 1048,
1238, 1188, 1452, 832, 2232, 1343, 1257, 1962, 804, 946, 2041, 1811, 1339, 972, 3095,
3159, 535, 2140, 1282, 1261, 257, 2882, 844, 3093, 1464, 1122, 1839, 3230, 56, 1893,
2529, 2470, 1264, 2173, 1868, 640, 102, 221, 3086, 3034, 1778, 1727, 466, 608, 2405,
```

1618, 395, 2922, 75, 1033, 2659, 1128, 1695, 1243, 962, 2552, 3096, 298, 2065, 548, 2
880, 532, 2095, 3207, 825, 448, 2625, 920, 1424, 168, 2692, 3003, 79, 288, 1680, 958,
2792, 2066, 3222, 964, 1065, 2678, 2971, 3107, 2492, 2997, 2051, 357, 2525, 2511, 173
9, 2603, 1260, 1572, 222, 1952, 2946, 2851, 1533, 1337, 2856, 2738, 2688, 3125, 784,
1562, 634, 1861, 455, 2403, 334, 1454, 3108, 3247, 1794, 2380, 126, 38, 3218, 1221, 3
250, 215, 818, 3134, 340, 2146, 1558, 287, 1789, 1521, 2105, 2630, 1235, 93, 1848, 22
85, 2034, 3087, 3053, 1483, 2337, 1058, 2824, 3020, 2120, 8, 2542, 2803, 1827, 356, 1
291, 1352, 773, 1804, 2420, 1246, 1773, 1007, 1325, 1227, 3233, 1762, 2364, 2765, 282
6, 1523, 2088, 2604, 1600, 1796, 2838, 358, 2162, 1540, 2274, 1465, 1263, 2804, 1266,
2226, 924, 3094, 2465, 675, 1613, 2259, 1619, 2344, 768, 943, 405, 1763, 2081, 981, 1
467, 2410, 3081, 1, 988, 1292, 2419, 2482, 996, 3052, 1563, 684, 833, 1240, 648, 233,
97, 1359, 1322, 347, 1915, 655, 1627, 2305, 1849, 254, 2702, 3150, 706, 2291, 1198, 1
504, 1259, 2052, 637, 1105, 923, 1957, 1186, 1655, 1317, 3177, 1092, 2697, 514, 874,
1320, 238, 1463, 2490, 2938, 2205, 595, 1754, 2989, 2099, 144, 569, 1148, 2961, 1566,
1039, 1741, 2740, 1411, 2993, 1665, 3235, 682, 188, 2365, 1447, 1882, 1330, 1643, 221
0, 1237, 407, 577, 3066, 2017, 1207, 1550, 1944, 1455, 37, 1856, 520, 3077, 2158, 90
7, 1335, 1019, 1150, 1620, 322, 3069, 1803, 1730, 263, 1124, 2739, 2330, 2209, 249, 2
37, 1050, 1699, 1184, 1220, 2113, 398, 44, 1546, 2956, 1213, 2928, 2333, 3102, 2428,
3139, 1030, 1029, 775, 3185, 2864, 2458, 1663, 846, 63, 1878, 467, 2141, 449, 2548, 2
742, 2254, 744, 2238, 2026, 2943, 2858, 772, 2286, 2190, 3244, 2200, 2267, 2875, 287
8, 1660, 365, 200, 653, 2867, 1522, 2949, 823, 2632, 1678, 2510, 1274, 690, 1402, 147
2, 2316, 1950, 536, 2314, 2555, 421, 525, 2854, 528, 2315, 2830, 2827, 59, 2339, 179
3, 1470, 1376, 1831, 681, 863, 2341, 2257, 2086, 1349, 1015, 12, 1462, 1049, 3238, 30
18, 336, 1777, 2694, 3163, 801, 1614, 2586, 486, 2230, 2444, 1431, 1326, 1716, 1153,
632, 271, 1582, 1828, 1006, 2784, 624, 606, 2720, 2293, 27, 983, 1040, 119, 2535, 127
2, 969, 3061, 1353, 2181, 848, 2409, 1481, 2498, 1022, 2384, 1927, 1426, 1077, 2520,
1721, 1625, 1312, 1308, 258, 1063, 898, 303, 2635, 2582, 581, 2899, 2773, 2071, 925,
2223, 3129, 631, 2472, 873, 1750, 1068, 1621, 2998, 2404, 3056, 543, 1045, 1924, 930,
402, 3201, 1575, 2967, 1712, 1583, 50, 3044, 593, 499, 3059, 1117, 1932, 2366, 1485,
809, 196, 803, 851, 1929, 470, 2534, 1638, 3131, 1646, 926, 3140, 3075, 1284, 3179, 2
300, 2058, 23, 795, 1892, 3068, 2907, 1830, 671, 1061, 2505, 66, 13, 349, 850, 576, 2
461, 4, 1866, 1637, 3101, 1251, 1358, 1507, 1867, 800, 3249, 1564, 2295, 2391, 195, 2
82, 163, 2608, 2860, 2606, 530, 330, 3252, 1369, 2437, 1425, 239, 306, 591, 3065, 99
2, 2524, 2284, 2069, 241, 2276, 2623, 1904, 1487, 3236, 89, 2565, 1436, 1724, 424, 14
96, 2541, 1846, 3, 1743, 883, 348, 64, 3257, 3133, 794, 1199, 557, 1114, 2941, 527, 8
80, 875, 1708, 2496, 1158, 1032, 876, 2665, 2549, 2587, 1711, 551, 1005, 2556, 3064,
338, 290, 2459, 1896, 683, 1598, 2573, 170, 277, 2755, 2082, 2218, 702, 1556, 529, 25
64, 456, 1365, 2918, 2115, 1170, 542, 1656, 1395, 2559, 537, 2225, 3203, 760, 2790, 2
25, 433, 1489, 3259, 2064, 2890, 1401, 565, 1731, 2747, 1757, 1659, 646, 1055, 2116,
2527, 1772, 2912, 3176, 206, 2433, 342, 1362, 1826, 339, 412, 1551, 987, 1242, 2440,
1482, 2939, 748, 1180, 1132, 836, 401, 953, 1611, 3047, 353, 995, 1484, 654, 2317, 11
34, 663, 2903, 2673, 789, 1152, 387, 1428, 2004, 511, 371, 1161, 2791, 3172, 865, 71
8, 2771, 1144, 1389, 866, 22, 2707, 2836, 2383, 2495, 2745, 3013, 3039, 3092, 3188, 2
577, 1214, 1774, 3192, 191, 2157, 2618, 1299, 711, 2852, 2176, 1009, 3033, 914, 2019,
1609, 2889, 1871, 1747, 2023, 3049, 714, 2382, 2483, 3126, 2325, 1633, 1785, 858, 146
8, 2658, 2392, 1552, 1623, 1267, 831, 708, 2237, 1883, 2877, 2767, 1816, 1404, 859, 1
064, 3209, 361, 2054, 36, 1761, 936, 2724, 2423, 114, 2235, 1628, 618, 1286, 3160, 12
24, 869, 1104, 2270, 807, 1086, 1538, 1400, 2369, 2954, 2474, 747, 1373, 441, 1805, 1
432, 2418, 502, 2250, 1156, 354, 2723, 622, 2627, 425, 227, 2430, 696, 1421, 2817, 53
9, 2609, 2656, 583, 2515, 2521, 2499, 3005, 2721, 1850, 1937, 2262, 1394, 787, 505, 1
021, 669, 2506, 1545, 381, 645, 434, 813, 1026, 245, 935, 2539, 2148, 3132, 1010, 233
5, 2304, 2754, 886, 2881, 122, 915, 2121, 1076, 1511, 1479, 1197, 1574, 644, 3055, 26
34, 3040, 746, 3060, 2819, 3158, 1204, 2924, 2751, 1173, 2242, 2753, 1864, 1354, 933,
1894, 582, 811, 3242, 1688, 960, 3157, 2101, 368, 2487, 2129, 2962, 490, 2156, 1042,
2164, 1060, 1313, 106, 2813, 2035, 1327, 413, 2198, 462, 1193, 2615, 2716, 679, 3214,
2732, 192, 2563, 3170, 982, 664, 2980, 2896, 2760, 1742, 3115, 1862, 1838, 171, 1415,
1527, 2719, 198, 1775, 3016, 2177, 1351, 1287, 3029, 885, 1302, 1047, 3027, 796, 286,
561, 1978, 2336, 1704, 2847, 2091, 2016, 719, 1630, 57, 1836, 3121, 2273, 355, 1686,
1837, 2302, 1131, 1889, 628, 2385, 3099, 2294, 2381, 1043, 2133, 2206, 2966, 694, 220
2, 436, 1640, 2362, 2718, 1696, 2626, 1510, 910, 1961, 642, 1501, 2346, 2180, 60, 40

```
3, 3239, 2727, 656, 428, 167, 285, 1812, 1935, 2107, 2280, 0, 2731, 662, 1911, 2145,
2233, 2435, 1792, 2050, 1334, 2743, 899, 2981, 1992, 2934, 761, 1948, 24, 740, 1147,
1456, 2221, 1776, 1881, 882, 240, 1810, 1011, 1123, 2705, 2605, 1052, 2682, 153, 432,
1958, 3008, 1972, 484, 121, 596, 2814, 1250, 1735, 2406, 871, 854, 141, 105, 1265, 24
91, 1067, 2676, 1270, 2030, 2024, 2370, 2400, 685, 2717, 265, 927, 546, 2695, 578, 31
65, 1321, 1492, 698, 1499, 453, 1025, 3098, 252, 373, 552, 609, 2452, 2126, 1710, 299
9, 1765, 704, 3026, 1578, 3048, 2887, 734, 2640, 2448, 2823, 764, 2075, 1037, 3089, 6
61, 447, 1229, 1209, 299, 667, 903, 2988, 2844, 3237, 2901, 2690, 1568, 1427, 1075, 2
689, 1403, 1969, 826, 3240, 2785, 1781, 323, 2669, 1174, 234, 2044, 1787, 2726, 838,
116, 1998, 108, 2025, 414, 1946, 544, 649, 343, 2614, 1758, 615, 367, 1531, 1737, 153
5, 2329, 605, 1217, 1549, 410, 2356, 1705, 934, 2728, 305, 905, 2313, 1102, 1101, 95
0, 834, 2798, 906, 2657, 3153, 2643, 3030, 707, 2711, 43, 968, 641, 1734, 1440, 2408,
2183, 1832, 1947, 1118, 725, 131, 503, 2425, 3182, 759, 1923, 15, 393, 2516, 157, 123
6, 1873, 1920, 721, 3246, 2494, 604, 297, 235, 3253, 213, 575, 2951, 2680, 2324, 887,
3080, 1594, 783, 2638, 2741, 311, 1967, 2833, 99, 300, 3097, 267, 1684, 2783, 388, 21
12, 360, 2683, 2664, 369, 111, 802, 291, 17, 1324, 2528, 1381, 2533, 2033, 2347, 364,
3074, 1854, 613, 590, 688, 1993, 2815, 2601, 193, 1185, 1097, 1891, 2897, 2097, 2360,
284, 1155, 21, 2876, 2801, 325, 3073, 2822, 2028, 2296, 1693, 2553, 1709, 2179, 2984,
1154, 1607, 1300, 1297, 155, 2258, 2596, 201, 1295, 2422, 2595, 741, 1416, 2320, 244
9, 1834, 1994, 137, 1520, 310, 3031, 419, 1096, 1146, 512, 1997, 2963, 1296, 125, 111
9, 1879, 944, 2283, 1687, 2216, 572, 2736, 636, 1159, 1853, 1046, 1140, 1863, 3085, 2
930, 1671, 416, 571, 302, 749, 3154, 2445, 2994, 2537, 1697, 2110, 3142, 2451, 247, 1
989, 69, 185, 691, 68, 3229, 547, 1382, 2084, 1689, 3147, 1013, 2175, 2466, 1759, 218
8, 180, 1390, 1066, 2713, 2508, 2588, 1072, 1276, 1843, 384, 2426, 1385, 1770, 1133,
54, 3035, 2855, 2892, 2234, 2353, 1453, 1907, 835, 2904, 205, 678, 85, 610, 2174, 304
1, 58, 41, 2909, 993, 1218, 2355, 1529, 2195, 2668, 1168, 1181, 1473, 3038, 2154, 325
6, 1897, 1277, 3103, 660, 1526, 1536, 1164, 2919, 627, 2710, 317, 2277, 1683, 3106, 1
818, 423, 585, 1176, 805, 2709, 2397, 154, 2650, 2532, 1130, 849, 797, 166, 1933, 310
0, 2127, 2442, 2628, 3156, 2837, 3181, 2434, 74, 1196, 2958, 1922, 1666, 1983, 1676,
2879, 1506, 1715, 2845, 1898, 2846, 847, 676, 2504, 892, 2646, 344, 1798, 1764, 967,
1372, 255, 1435, 2744, 877, 2124, 274, 2908, 1493, 2545, 292, 1984, 786, 2282, 1399,
2780, 1225, 2831, 2920, 2208, 478, 843, 1554, 1817, 2675, 454, 1825, 1650, 1417, 198
8, 445, 912, 1634, 1905, 2488, 2679, 2686, 816, 415, 98, 2885, 169, 324, 727, 2808, 7
70, 2192, 1306, 2068, 1718, 2737, 2303, 901, 855, 1177, 941, 2571, 47, 586, 2518, 19
9, 985, 861, 331, 2641, 451, 1275, 3111, 430, 1635, 798, 295, 6, 3258, 1366, 1910, 18
42, 197, 231, 971, 1311, 3028, 2811, 1912, 2725, 2948, 630, 2868, 1995, 1584, 3226, 2
685, 3232, 1413, 1968, 1820, 939, 2469, 1698, 1486, 753, 2599, 2786, 2886, 715, 1649,
2197, 1368, 2241, 1596, 2078, 469, 1691, 754, 1201, 359, 1615, 1248, 1565, 140, 1439,
2591, 1392, 1953, 822, 1356, 2087, 3079, 220, 2799, 149, 2060, 2557, 78, 370, 124, 30
46, 2873, 366, 1783, 232, 383, 1298, 1412, 1380, 2538, 2990, 2298, 2776, 2001, 2612,
1906, 1126, 1191, 212, 507, 2991, 475, 2585, 674, 3000, 1943, 1593, 386, 989, 896, 10
9, 2020, 2769, 160, 2431, 2945, 209, 134, 3127, 1301, 2583, 1651, 2446, 244, 756, 200
8, 1813, 2441, 450, 570, 112, 689, 2062, 1653, 2729, 699, 91, 857, 651, 799, 2186, 18
84, 2816, 1514, 2691, 856, 11, 916, 1645, 2443, 1480, 2168, 1833, 1519, 1502, 1051, 1
52, 1865, 2781, 2648, 1644, 2610, 1902, 95, 1247, 2003, 853, 1728, 2201, 2964, 1938,
2036, 1603, 658, 820, 2662, 1673, 174, 446, 2891, 1986, 2929, 1790, 819, 3164, 840, 1
407, 2871, 139, 2462, 329, 550, 1408, 1344, 2905, 3120, 77, 2207, 2042, 318, 2072, 95
6, 1474, 2802, 207, 2795, 1913, 555, 625, 1160, 2636, 457, 1819, 2763, 151, 867, 300
9, 792, 814, 1886, 2144, 709, 2014, 957, 440, 9, 780, 3057, 2978, 1982, 2796, 2167, 2
579, 3135, 2388, 2263, 1294, 1569, 1668, 1822, 123, 1200, 1513, 1024, 827, 732, 3043,
697, 2644, 2231, 793, 1631, 599, 2489, 2424, 2921, 1602, 1706, 2322, 774, 757, 2578,
2810, 461, 408, 2417, 1107, 620, 459, 2748, 3076, 2696, 2863, 1835, 1652, 1363, 1450,
2108, 1012, 2914, 2067, 1960, 172, 559, 219, 1570, 1109, 1088, 2787, 1471, 1605, 235
8, 2224, 2268, 742, 439, 2898, 722, 1279, 1975]
[1874, 765, 1231, 2272, 1720, 2447, 2666, 579, 2292, 443, 2829, 2775, 1518, 2806, 61
2, 2584, 745, 2319, 1205, 243, 1442, 2083, 2022, 464, 2153, 2196, 2318, 1386, 438, 10
87, 1210, 1844, 1509, 2835, 2161, 396, 1027, 2160, 3021, 460, 1771, 312, 1125, 3215,
1305, 3260, 2935, 1601, 2006, 3199, 2639, 860, 319, 1990, 1555, 3200, 1357, 1371, 12
0, 28, 1378, 2376, 1852, 1460, 3136, 2995, 1082, 379, 483, 351, 372, 1733, 750, 1283,
2959, 695, 1044, 150, 2547, 766, 563, 1059, 776, 2779, 1041, 890, 283, 2655, 961, 94
```

```
0, 1914, 187, 2916, 296, 1971, 113, 2281, 2672, 2377, 1917, 842, 2453, 1964, 2766, 58
8, 1976, 1319, 978, 1476, 2334, 705, 821, 1567, 1963, 1098, 94, 1258, 891, 1094, 273
0, 2394, 2438, 2983, 1211, 2203, 3187, 3088, 2894, 2398, 600, 815, 1095, 558, 762, 28
83, 1814, 2756, 3161, 2039, 392, 665, 611, 1642, 2633, 3123, 1981, 2118, 2374, 3015,
3090, 657, 1690, 963, 2049, 1537, 341, 2850, 2279, 1434, 1624, 2455, 1505, 1035, 161
2, 2523, 1860, 1023, 2645, 736, 991, 316, 739, 2310, 1328, 2253, 250, 2266, 409, 227
1, 437, 976, 2166, 2094, 2308, 148, 363, 1226, 2371, 1165, 1973, 2758, 471, 104, 496,
2936, 96, 733, 1539, 2700, 406, 2311, 376, 2910, 3149, 1592, 3137, 587, 1939, 20, 240
7, 422, 2178, 2309, 1304, 444, 2734, 1541, 2171, 2712, 504, 2828, 1901, 497, 1073, 29
75, 1532, 1333, 115, 533, 1744, 3070, 1740, 2888, 2416, 1106, 2194, 895, 1003, 73, 28
66, 1713, 2843, 1722, 3007, 2032, 2570, 1525, 217, 1767, 2551, 177, 2607, 2872, 2217,
3017, 2475, 1887, 1528, 3224, 731, 2321, 3019, 1557, 14, 2275, 522, 506, 1755, 1346,
782, 1085, 1477, 1245, 2402, 1595, 2649, 3194, 1438, 293, 1288, 643, 1293, 2007, 205
6, 2111, 352, 135, 1977, 280, 2915, 510, 189, 519, 224, 7, 1323, 3219, 208, 1732, 212
8, 2159, 2531, 1100, 1016, 3104, 204, 1639, 2338, 2574, 1996, 30, 526, 1121, 980, 70
1, 488, 2189, 500, 3128, 904, 1441, 251, 1954, 3243, 1034, 513, 1398, 465, 2507, 218,
1694, 2199, 2150, 81, 2970, 979, 1228, 2061, 703, 494, 1970, 3251, 202, 3211, 1760, 8
12, 2098, 2841, 143, 3145, 1001, 491, 1175, 2411, 278, 931, 1397, 2227, 1183, 2125, 2
715, 1870, 132, 909, 3245, 2289, 1779, 2089, 2550, 1955, 19, 817, 638, 1692, 145, 167
7, 2327, 1719, 1189, 1707, 2597, 1418, 1903, 1110, 2123, 1195, 3196, 531, 3067, 1414,
2536, 350, 2629, 1393, 1141, 751, 2354, 1290, 781, 2169, 138, 495, 921, 1108, 1847, 2
328, 893, 2663, 3186, 327, 693, 1795, 919, 2764, 1367, 1318, 2460, 417, 397, 2859, 26
60, 1782, 2777, 767, 1342, 977, 3220, 2677, 911, 2048, 1269, 538, 3011, 2800, 3174, 1
10, 429, 1544, 1230, 1081, 3110, 3063, 1280, 758, 1490, 726, 1310, 2463, 3261, 1223,
1869, 2572, 1028, 1512, 87, 2768, 248, 2414, 90, 2005, 3022, 1142, 2895, 2493, 1444,
2782, 1445, 1909, 1361, 2569, 2594, 246, 545, 2589, 2413, 2522, 1542, 3155, 1289, 237
5, 3151, 46, 1388, 2246, 1241, 2134, 1057, 1791, 3197, 973, 2931, 2950, 2598, 1766, 1
111, 2076, 2540, 420, 2290, 2789, 2439, 2637, 1581, 2130, 1885, 1543, 3144, 566, 147
5, 1285, 1391, 173, 2825, 639, 1078, 2500, 1194, 1670, 808, 128, 2193, 2456, 442, 31
5, 29, 1268, 147, 966, 2070, 1143, 3248, 1355, 2401, 2942, 1547, 184, 3119, 2517, 247
8, 181, 487, 1700, 3148, 1749, 1203, 82, 617, 1681, 272, 1409, 362, 1980, 2616, 1974,
259, 509, 1167, 1093, 2349, 2592, 864, 2684, 2923, 426, 2331, 2342, 2219, 1336, 2861,
668, 2849, 2749, 2481, 1145, 117, 2793, 1379, 951, 2393, 1669, 2396, 2187, 1437, 235
1, 1580, 2077, 1725, 3223, 2566, 3190, 1786, 2752, 1056, 888, 2027, 556, 146, 399, 14
97, 549, 1821, 2363, 2982, 1617, 1748, 2925, 1587, 2944, 2436, 1936, 1103, 127, 3195,
1233, 175, 2421, 3105, 652, 262, 3183, 2750, 62, 3036, 1597, 2142, 621, 2600, 949, 61
6, 2617, 1626, 1951, 2251, 2215, 1926, 2937, 785, 959, 2256, 1062, 1921, 53, 2480, 26
8, 2379, 2562, 1941, 2473, 1515, 534, 2229, 2427, 974]
      Unnamed: 0 Speaker Modality  Verb SemanticClass  LengthOfRecipient  \
942          942   S1140   spoken  give             a                  1
2352        2352   S1098   spoken  give             a                  1
2947        2947   S1531   spoken  give             t                  1
2247        2247   S1268   spoken  show             c                  1
2138        2138   S1235   spoken  mail             t                  1
...          ...     ...      ...   ...           ...                ...
439          440     NaN  written  give             a                  3
2898        2898   S1487   spoken  send             t                  1
722          723     NaN  written  sell             t                  2
1279        1279   S1101   spoken  send             t                  1
1975        1975   S1130   spoken  give             a                  2

     AnimacyOfRec  DefinOfRec    PronomOfRec  LengthOfTheme  ...  \
942     inanimate    definite     pronominal              1  ...
2352      animate    definite     pronominal              6  ...
2947      animate    definite     pronominal              6  ...
2247      animate    definite     pronominal             24  ...
2138      animate    definite     pronominal              2  ...
...           ...         ...            ...            ...  ...
439       animate    definite  nonpronominal              7  ...
2898      animate    definite     pronominal              1  ...
```

```
722       animate    definite  nonpronominal            3  ...
1279      animate    definite     pronominal            3  ...
1975      animate  indefinite     pronominal            3  ...

      PronomOfTheme RealizationOfRecipient AccessOfRec AccessOfTheme Response  \
942   nonpronominal                     NP       given    accessible        1
2352  nonpronominal                     NP       given    accessible        1
2947  nonpronominal                     NP       given           new        1
2247  nonpronominal                     NP       given           new        1
2138  nonpronominal                     PP       given           new        0
...             ...                    ...         ...           ...      ...
439   nonpronominal                     NP         new           new        1
2898  nonpronominal                     NP       given         given        1
722   nonpronominal                     PP  accessible           new        0
1279  nonpronominal                     NP       given         given        1
1975  nonpronominal                     PP  accessible    accessible        0

     RecPro  ThemePro  logLengthOfTheme  logLengthOfRecipient  Dummy
942       1         0          0.000000              0.000000      1
2352      1         0          2.584963              0.000000      1
2947      1         0          2.584963              0.000000      1
2247      1         0          4.584963              0.000000      1
2138      1         0          1.000000              0.000000      1
...     ...       ...               ...                   ...    ...
439       0         0          2.807355              1.584963      1
2898      1         0          0.000000              0.000000      1
722       0         0          1.584963              1.000000      1
1279      1         0          1.584963              0.000000      1
1975      1         0          1.584963              1.000000      1

[2610 rows x 22 columns]
```

In [ ]:
```
def log_likelihood(predictions, labels):
    values = [y*np.log(predicted)+(1-y)*np.log(1-predicted) for y, predicted in zip(la
    return sum(values)[0]
```

In [ ]:
```
def accuracy(predictions, labels):
    labelled_predictions = [1 if prediction > 0.5 else 0 for prediction in predictions
    correct = [1 if prediction == label else 0 for prediction, label in zip(labelled_p
    return sum(correct)/len(labels)
```

In [ ]:
```
## TODO: Fit a logistic regression model to the training set that uses only recipient
##       and an intercept term.
##       What is its classification accuracy on the held-out test dataset? How about i
x_train = d_train[["RecPro","Dummy"]]
y_train = d_train[["Response"]]
m = sm.GLM(y_train,x_train,family=sm.families.Binomial())
m_results = m.fit()
print(m_results.summary())
x_test = d_test[["RecPro","Dummy"]]
y_test = d_test[["Response"]]
y_predicted = m_results.predict(x_test)
print("Predicted proportion of DO outcomes based on fitted intercept and recipient pro
print("Proportion of data with DO outcome:", round(np.mean(dat["Response"]),4))
print(f'Classification Accuracy: {round(accuracy(y_predicted, y_test.values), 4)}')
print(f'Log-Likelihood {round(log_likelihood(y_predicted, y_test), 4)}')
```

```
                 Generalized Linear Model Regression Results
==============================================================================
Dep. Variable:               Response   No. Observations:                 2610
Model:                            GLM   Df Residuals:                     2608
Model Family:                Binomial   Df Model:                            1
Link Function:                  Logit   Scale:                          1.0000
Method:                          IRLS   Log-Likelihood:                -1231.6
Date:                Wed, 17 May 2023   Deviance:                       2463.2
Time:                        10:42:37   Pearson chi2:                 2.61e+03
No. Iterations:                     5   Pseudo R-squ. (CS):             0.1861
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
RecPro         2.2048      0.103     21.490      0.000       2.004       2.406
Dummy         -0.0797      0.064     -1.246      0.213      -0.205       0.046
==============================================================================
Predicted proportion of DO outcomes based on fitted intercept and recipient pronomina
lity model: 0.7351
Proportion of data with DO outcome: 0.7398
Classification Accuracy: 0.7305
Log-Likelihood -317.1015
```

TODO: interpretation goes here.

The findings suggest that the model's suitability is significant. The significantly low p-value and the coefficient of the variable both indicate that Recipient Pronominality is a reliable indicator for DO. Furthermore, the predicted proportion of DO outcomes (0.735) closely resembled the actual proportion (0.739), although this measure alone may not be highly informative. However, the classification accuracy ultimately reached only 0.61, surpassing random performance but not achieving exceptional results. Additionally, the log-likelihood yielded a considerably negative value, suggesting that the fit may not be as strong as initially anticipated.

In [ ]:

```
## TODO: Add theme pronominality as a predictor to the model and see whether that impr
##       model’s predictive power as assessed by held-out classification accuracy and
x_train = d_train[["RecPro","ThemePro","Dummy"]]
y_train = d_train[["Response"]]
m = sm.GLM(y_train,x_train,family=sm.families.Binomial())
m_results = m.fit()
print(m_results.summary())
x_test = d_test[["RecPro","ThemePro","Dummy"]]
y_test = d_test[["Response"]]
y_predicted = m_results.predict(x_test)
print("Predicted proportion of DO outcomes based on fitted intercept and recipient pro
print("Proportion of data with DO outcome:", round(np.mean(dat["Response"]),4))
print(f'Classification Accuracy: {round(accuracy(y_predicted, y_test.values), 4)}')
print(f'Log-Likelihood {round(log_likelihood(y_predicted, y_test), 4)}')
```

```
                 Generalized Linear Model Regression Results
==============================================================================
Dep. Variable:               Response   No. Observations:                 2610
Model:                            GLM   Df Residuals:                     2607
Model Family:                Binomial   Df Model:                            2
Link Function:                  Logit   Scale:                          1.0000
Method:                          IRLS   Log-Likelihood:                -1040.9
Date:                Wed, 17 May 2023   Deviance:                       2081.7
Time:                        10:42:41   Pearson chi2:                 2.59e+03
No. Iterations:                     6   Pseudo R-squ. (CS):             0.2967
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
RecPro         2.9470      0.139     21.169      0.000       2.674       3.220
ThemePro      -3.0431      0.172    -17.657      0.000      -3.381      -2.705
Dummy          0.1137      0.067      1.699      0.089      -0.017       0.245
==============================================================================
Predicted proportion of DO outcomes based on fitted intercept and recipient pronomina
lity model: 0.7227
Proportion of data with DO outcome: 0.7398
Classification Accuracy: 0.7871
Log-Likelihood -268.9947
```

TODO: interpretation goes here.

These results demonstrate a substantial improvement. The p-values once again affirm the significance of both variables in the regression model. Notably, the accuracy has shown a notable improvement, reaching 0.787, which is statistically significant. Moreover, the log-likelihood of this model has improved significantly, indicating a stronger fit. Based on these findings, we can confidently conclude that the Theme Pronominality variable plays a valuable role in enhancing the predictive power and fit of the model.

In [ ]:
```
## TODO: Determine whether additionally adding theme and recipient length (in number c
##       to the model further improves fit. Try both raw length or log-transformed ler
##       Which gives better performance?
```

In [ ]:
```
#Length
x_train = d_train[["RecPro", "ThemePro","LengthOfTheme","LengthOfRecipient","Dummy"]]
y_train = d_train[["Response"]]
m = sm.GLM(y_train,x_train,family=sm.families.Binomial())
m_results = m.fit()
print(m_results.summary())
x_test = d_test[["RecPro", "ThemePro","LengthOfTheme","LengthOfRecipient","Dummy"]]
y_test = d_test[["Response"]]
y_predicted = m_results.predict(x_test)
print("Predicted proportion of DO outcomes based on fitted intercept and recipient prc
print("Proportion of data with DO outcome:", round(np.mean(dat["Response"]),4))
print(f'Classification Accuracy: {round(accuracy(y_predicted, y_test.values), 4)}')
print(f'Log-Likelihood {round(log_likelihood(y_predicted, y_test), 4)}')
```

```
                 Generalized Linear Model Regression Results
==============================================================================
Dep. Variable:               Response   No. Observations:                 2610
Model:                            GLM   Df Residuals:                     2605
Model Family:                Binomial   Df Model:                            4
Link Function:                  Logit   Scale:                          1.0000
Method:                          IRLS   Log-Likelihood:                -877.43
Date:                Wed, 17 May 2023   Deviance:                       1754.9
Time:                        10:42:43   Pearson chi2:                 3.96e+03
No. Iterations:                     6   Pseudo R-squ. (CS):             0.3795
Covariance Type:            nonrobust
=====================================================================================
                        coef    std err          z      P>|z|      [0.025      0.975]
-------------------------------------------------------------------------------------
RecPro                2.6148      0.161     16.276      0.000       2.300       2.930
ThemePro             -2.7541      0.177    -15.544      0.000      -3.101      -2.407
LengthOfTheme         0.2663      0.024     10.967      0.000       0.219       0.314
LengthOfRecipient    -0.4029      0.043     -9.384      0.000      -0.487      -0.319
Dummy                 0.0573      0.153      0.373      0.709      -0.243       0.358
=====================================================================================
Predicted proportion of DO outcomes based on fitted intercept and recipient pronomina
lity model: 0.7282
Proportion of data with DO outcome: 0.7398
Classification Accuracy: 0.8453
Log-Likelihood -238.0128
```

In [ ]:

```
#Log-transformed Length
x_train = d_train[["RecPro", "ThemePro","logLengthOfTheme","logLengthOfRecipient","Dur
y_train = d_train[["Response"]]
m = sm.GLM(y_train,x_train,family=sm.families.Binomial())
m_results = m.fit()
print(m_results.summary())
x_test = d_test[["RecPro", "ThemePro","logLengthOfTheme","logLengthOfRecipient","Dummy
y_test = d_test[["Response"]]
y_predicted = m_results.predict(x_test)
print("Predicted proportion of DO outcomes based on fitted intercept and recipient pro
print("Proportion of data with DO outcome:", round(np.mean(dat["Response"]),4))
print(f'Classification Accuracy: {round(accuracy(y_predicted, y_test.values), 4)}')
print(f'Log-Likelihood {round(log_likelihood(y_predicted, y_test), 4)}')
```

```
                 Generalized Linear Model Regression Results
==============================================================================
Dep. Variable:               Response   No. Observations:                 2610
Model:                            GLM   Df Residuals:                     2605
Model Family:                Binomial   Df Model:                            4
Link Function:                  Logit   Scale:                          1.0000
Method:                          IRLS   Log-Likelihood:                -878.94
Date:                Wed, 17 May 2023   Deviance:                       1757.9
Time:                        10:42:46   Pearson chi2:                 2.70e+03
No. Iterations:                     6   Pseudo R-squ. (CS):             0.3788
Covariance Type:            nonrobust
==================================================================================
===
                           coef    std err          z      P>|z|      [0.025      0.9
75]
----------------------------------------------------------------------------------
---
RecPro                   2.2690      0.180     12.639      0.000       1.917       2.
621
ThemePro                -2.5424      0.181    -14.031      0.000      -2.898      -2.
187
logLengthOfTheme         0.7543      0.061     12.421      0.000       0.635       0.
873
logLengthOfRecipient    -0.9315      0.086    -10.788      0.000      -1.101      -0.
762
Dummy                   -0.0972      0.161     -0.605      0.545      -0.412       0.
218
==================================================================================
===
Predicted proportion of DO outcomes based on fitted intercept and recipient pronomina
lity model: 0.7256
Proportion of data with DO outcome: 0.7398
Classification Accuracy: 0.8469
Log-Likelihood -231.4777
```

The inclusion of both the word length of the theme and recipient has further improved the model. Since the log-transformed lengths slightly outperformed the raw length, we will report the statistics based on these variables. The accuracy has significantly improved to an impressive 0.847, showcasing commendable predictive performance. Additionally, the log-likelihood has increased even further, indicating a substantial enhancement in predictive power and an improved fit. Overall, these covariates are deemed significant and contribute significantly to the model's overall performance.

As previously discussed, the coefficient values indicate that the original variables, recipient and theme pronominiality, hold significant importance in the model. The length variables, on the other hand, do not contribute as substantially. However, we learned in class about Panini's Law and its relationship to Ordering Preferences, where shorter words in terms of syllables tend to take precedence. Although this is not directly applicable to our specific case, there is a connection between word length and number of syllables (albeit imperfect). Intuitively, this connection makes sense, and exploring a model that incorporates this relationship would be intriguing.

Considering the high accuracy achieved without including the length variables, there is a possibility of overfitting. Moreover, incorporating length variables aligns with linguistic sensibility. Additionally, in our case, larger DO phrases are often reported as awkward, suggesting that length may influence the outcome. To integrate these variables into the model, we propose incorporating first-degree interactions between the variables. It is important to note that it would not make sense linguistically to add interactions between variables that pertain to different objects. Therefore, we will introduce the variables 'RecPro & LengthofRecipient' as well as 'ThemePro & LengthofTheme'. Furthermore, we will utilize log-transformed length rather than the raw length, as it yielded a better fit in our case.

In [ ]:

```
#Multiply by log-transformed length
dat['logMultRec'] = dat['RecPro']*dat['logLengthOfRecipient']
dat['logMultTheme'] = dat['ThemePro']*dat['logLengthOfTheme']
N = dat.shape[0]
N_train = math.floor(N*4/5)
idx = list(range(N))
random.seed(3)
random.shuffle(idx)
idx_train = idx[0:N_train]
idx_test = idx[N_train:N]
d_train = dat.iloc[idx_train]
d_test = dat.iloc[idx_test]
```

In [ ]:

```
x_train = d_train[["logMultRec", "logMultTheme","Dummy"]]
y_train = d_train[["Response"]]
m = sm.GLM(y_train,x_train,family=sm.families.Binomial())
m_results = m.fit()
print(m_results.summary())
x_test = d_test[["logMultRec", "logMultTheme","Dummy"]]
y_test = d_test[["Response"]]
y_predicted = m_results.predict(x_test)
print("Predicted proportion of DO outcomes based on fitted intercept and recipient pro
print("Proportion of data with DO outcome:", round(np.mean(dat["Response"]),4))
print(f'Classification Accuracy: {round(accuracy(y_predicted, y_test.values), 4)}')
print(f'Log-Likelihood {round(log_likelihood(y_predicted, y_test), 4)}')
```

```
                 Generalized Linear Model Regression Results
==============================================================================
Dep. Variable:               Response   No. Observations:                 2610
Model:                            GLM   Df Residuals:                     2607
Model Family:                Binomial   Df Model:                            2
Link Function:                  Logit   Scale:                          1.0000
Method:                          IRLS   Log-Likelihood:                -1486.4
Date:                Wed, 17 May 2023   Deviance:                       2972.9
Time:                        10:42:50   Pearson chi2:                 2.61e+03
No. Iterations:                     4   Pseudo R-squ. (CS):            0.01054
Covariance Type:            nonrobust
================================================================================
                   coef    std err          z      P>|z|      [0.025      0.975]
--------------------------------------------------------------------------------
logMultRec      -0.7854      0.214     -3.673      0.000      -1.205      -0.366
logMultTheme    -0.3457      0.098     -3.531      0.000      -0.538      -0.154
Dummy            1.0878      0.046     23.613      0.000       0.997       1.178
================================================================================
Predicted proportion of DO outcomes based on fitted intercept and recipient pronomina
lity model: 0.7353
Proportion of data with DO outcome: 0.7398
Classification Accuracy: 0.7519
Log-Likelihood -361.9777
```

Given the significant loss of accuracy and goodness of fit observed, it is prudent to explore alternative approaches. Therefore, we will now consider interactions between the two 'Pronominality' variables and the two 'length' variables. This adjustment aims to capture potential synergistic effects between these variables and potentially improve the model's performance.

In [ ]:

```
#Interactions between the two 'Pronominality' variables and the two 'length' variables
dat['MultRecTheme'] = dat['RecPro']*dat['ThemePro']
dat['MultLength'] = dat['logLengthOfRecipient']*dat['logLengthOfTheme']
N = dat.shape[0]
N_train = math.floor(N*4/5)
idx = list(range(N))
random.seed(3)
random.shuffle(idx)
idx_train = idx[0:N_train]
idx_test = idx[N_train:N]
d_train = dat.iloc[idx_train]
d_test = dat.iloc[idx_test]
```

In [ ]:

```
x_train = d_train[["MultRecTheme", "MultLength","Dummy"]]
y_train = d_train[["Response"]]
m = sm.GLM(y_train,x_train,family=sm.families.Binomial())
m_results = m.fit()
print(m_results.summary())
x_test = d_test[["MultRecTheme", "MultLength","Dummy"]]
y_test = d_test[["Response"]]
y_predicted = m_results.predict(x_test)
print("Predicted proportion of DO outcomes based on fitted intercept and recipient pro
print("Proportion of data with DO outcome:", round(np.mean(dat["Response"]),4))
print(f'Classification Accuracy: {round(accuracy(y_predicted, y_test.values), 4)}')
print(f'Log-Likelihood {round(log_likelihood(y_predicted, y_test), 4)}')
```

```
                 Generalized Linear Model Regression Results
==============================================================================
Dep. Variable:               Response   No. Observations:                 2610
Model:                            GLM   Df Residuals:                     2607
Model Family:                Binomial   Df Model:                            2
Link Function:                  Logit   Scale:                          1.0000
Method:                          IRLS   Log-Likelihood:                -1369.8
Date:                Wed, 17 May 2023   Deviance:                       2739.7
Time:                        10:42:54   Pearson chi2:                 2.59e+03
No. Iterations:                     4   Pseudo R-squ. (CS):            0.09512
Covariance Type:            nonrobust
================================================================================
                  coef    std err          z      P>|z|      [0.025      0.975]
--------------------------------------------------------------------------------
MultRecTheme   -1.6340      0.146    -11.179      0.000      -1.921      -1.348
MultLength     -0.2984      0.024    -12.610      0.000      -0.345      -0.252
Dummy           1.5210      0.058     26.073      0.000       1.407       1.635
================================================================================
Predicted proportion of DO outcomes based on fitted intercept and recipient pronomina
lity model: 0.7416
Proportion of data with DO outcome: 0.7398
Classification Accuracy: 0.7596
Log-Likelihood -336.9262
```

These results are truly impressive. We have achieved a slight improvement in both accuracy and model fit by enhancing the log-likelihood. Moreover, we have accomplished this while reducing the number of variables to only two. This approach, which involves breaking down our variables into separate components representing length and pronominality, is not only statistically beneficial but also linguistically logical. By adopting this approach, we have successfully maintained the essence of the original model while simultaneously enhancing efficiency.

In summary, we have successfully constructed a model that exhibits improved fit and linguistic soundness without sacrificing much accuracy. This was achieved by incorporating first-degree interactions between the relevant variables. By doing so, we have enhanced the model's performance and captured the intricate relationships between the variables more effectively.

2) Word embeddings

The below code and text are for the second problem on the pset. Note that the second code chunk will take several minutes to run, but only needs to be run once, which will download the GLoVe vectors and save them on your Google drive in a new folder named *096222-pset-3* (about 1GB for the glove.6B.zip dataset). When done with the pset you may delete the files to free up space.

In [ ]:

```
from google.colab import drive
drive.mount('/content/gdrive')
GDRIVE_DIR = "/content/gdrive/My Drive/096222-pset-3"
```

```
Mounted at /content/gdrive
```

In [ ]:

```
# This code chunk needs to be run only the first time through the pset.
# It downloads the GLoVe word embeddings and saves them to your Google drive.
!time wget http://nlp.stanford.edu/data/glove.6B.zip
```

```
!unzip glove.6B.zip
!mkdir -p "$GDRIVE_DIR"
!mv glove.6B.300d.txt "$GDRIVE_DIR/"
```

```
--2023-05-17 10:43:23--  http://nlp.stanford.edu/data/glove.6B.zip
Resolving nlp.stanford.edu (nlp.stanford.edu)... 171.64.67.140
Connecting to nlp.stanford.edu (nlp.stanford.edu)|171.64.67.140|:80... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://nlp.stanford.edu/data/glove.6B.zip [following]
--2023-05-17 10:43:23--  https://nlp.stanford.edu/data/glove.6B.zip
Connecting to nlp.stanford.edu (nlp.stanford.edu)|171.64.67.140|:443... connected.
HTTP request sent, awaiting response... 301 Moved Permanently
Location: https://downloads.cs.stanford.edu/nlp/data/glove.6B.zip [following]
--2023-05-17 10:43:23--  https://downloads.cs.stanford.edu/nlp/data/glove.6B.zip
Resolving downloads.cs.stanford.edu (downloads.cs.stanford.edu)... 171.64.64.22
Connecting to downloads.cs.stanford.edu (downloads.cs.stanford.edu)|171.64.64.22|:44
3... connected.
HTTP request sent, awaiting response... 200 OK
Length: 862182613 (822M) [application/zip]
Saving to: ‘glove.6B.zip’

glove.6B.zip        100%[===================>] 822.24M  4.81MB/s    in 2m 49s

2023-05-17 10:46:12 (4.87 MB/s) - ‘glove.6B.zip’ saved [862182613/862182613]


real    2m49.597s
user    0m1.591s
sys     0m4.526s
Archive:  glove.6B.zip
  inflating: glove.6B.50d.txt
  inflating: glove.6B.100d.txt
  inflating: glove.6B.200d.txt
  inflating: glove.6B.300d.txt
```

In [ ]:
```
import sys
import numpy

def read_vectors_from_file(filename):
    d = {}
    with open(filename, 'rt') as infile:
        for line in infile:
            word, *rest = line.split()
            d[word] = numpy.array(list(map(float, rest)))
    return d

e = read_vectors_from_file(GDRIVE_DIR + "/glove.6B.300d.txt")
```

In [ ]:
```
e['apples']
```

Out[ ]:
```
array([-0.17994  ,  0.076623 ,  0.15722  , -0.22001  , -0.018468 ,
       -0.23543  ,  0.066769 ,  0.31273  ,  0.13766  , -0.10719  ,
        0.042323 , -0.22365  ,  0.15889  , -0.13794  ,  0.018843 ,
        0.26975  , -0.53504  , -0.54033  ,  0.013745 ,  0.27357  ,
       -0.37072  ,  0.25398  ,  0.25217  ,  0.20234  ,  0.031093 ,
       -0.55085  , -0.93268  , -0.064355 ,  0.073996 , -0.28748  ,
       -0.73238  ,  0.038759 , -0.23089  , -0.35184  , -0.40089  ,
        0.15194  ,  0.083998 ,  0.3137   , -0.80714  , -0.4338   ,
        0.66056  , -0.28821  , -0.11314  , -0.0079687,  0.28257  ,
       -0.047495 , -0.63175  ,  0.29189  ,  0.0064372,  0.57836  ,
       -0.053689 , -0.31578  , -0.078192 , -0.39074  , -1.0015   ,
       -0.65737  , -0.30738  , -0.26731  , -0.18491  ,  0.067175 ,
        0.14621  , -0.013356 , -0.18675  ,  0.28284  , -0.18525  ,
       -0.075742 , -0.16288  ,  0.12174  , -0.54502  ,  0.10662  ,
        0.086968 , -0.04665  , -0.20161  ,  0.053088 , -1.0036   ,
       -0.13441  ,  0.81115  ,  0.016895 ,  0.027232 , -0.31431  ,
       -0.66949  ,  0.085227 ,  0.30046  , -0.17164  ,  0.10544  ,
       -0.22445  , -0.60275  ,  0.23061  , -0.047089 , -0.58511  ,
        0.44815  , -0.074066 , -0.14275  , -0.15313  , -0.63952  ,
       -0.094444 , -0.21364  ,  0.087407 , -0.17284  ,  0.56991  ,
        0.071645 ,  0.011137 ,  0.2267   , -0.71376  , -0.96206  ,
       -0.19973  ,  0.014132 ,  0.23701  , -0.35592  ,  0.13589  ,
        0.24791  ,  0.13386  ,  0.29055  , -0.55914  ,  0.44929  ,
       -0.21718  , -0.42051  ,  0.95901  ,  0.48805  , -0.006123 ,
        0.047679 , -0.67585  , -0.50386  ,  0.41547  , -0.95469  ,
        0.1084   , -0.13225  ,  0.81998  , -0.39     ,  0.29343  ,
       -0.51845  ,  0.90005  ,  0.8312   ,  0.35276  ,  0.076735 ,
       -0.070346 ,  0.14675  ,  0.22855  , -0.3421   ,  0.34676  ,
        0.56451  ,  0.68692  , -0.43837  , -0.44616  ,  0.6452   ,
       -0.31362  ,  0.24     , -0.39258  ,  0.74207  , -0.37836  ,
       -0.92141  , -0.024509 ,  0.46447  , -0.11092  , -0.72349  ,
       -0.13231  , -0.446    , -0.025783 ,  0.087329 , -0.018828 ,
        0.10109  ,  0.40279  ,  0.4081   , -0.45704  ,  0.61521  ,
        0.20585  ,  0.24611  , -0.47398  ,  0.31816  , -0.32364  ,
       -0.8207   , -0.0055949, -0.10262  , -0.056061 ,  0.32711  ,
       -0.32271  ,  0.69101  , -0.017224 ,  0.092357 , -0.44683  ,
        0.19494  ,  0.081131 ,  0.36321  , -0.33085  ,  0.075969 ,
       -0.34788  ,  1.314    , -0.52115  ,  0.64394  ,  0.28914  ,
       -0.41288  ,  1.1367   , -0.093191 , -0.38916  , -0.66092  ,
       -0.33191  ,  0.091428 ,  0.11462  , -0.29782  , -0.18357  ,
        0.43218  , -0.38981  ,  0.20815  , -0.1044   , -0.12044  ,
        0.1654   ,  0.54256  ,  0.85342  ,  0.54981  , -0.47756  ,
        0.14017  ,  0.17094  , -0.1258   ,  0.26912  , -0.25852  ,
       -0.75258  ,  1.051    ,  0.20071  , -0.19395  , -0.46276  ,
        0.66577  ,  0.076325 , -0.45036  ,  0.15611  , -0.54071  ,
        0.5769   ,  0.22945  ,  0.3742   ,  0.257    ,  0.21808  ,
       -0.1865   ,  0.05094  , -0.068712 , -0.24761  ,  0.35928  ,
        0.62262  ,  0.1641   , -0.19284  ,  0.084023 , -0.39765  ,
       -0.64286  , -0.16724  , -0.47489  ,  0.30388  , -0.65713  ,
        0.10427  , -0.32936  ,  0.49474  , -0.44321  , -0.39947  ,
        0.5853   ,  0.61492  , -0.69749  ,  0.18777  ,  0.39172  ,
        0.1848   , -0.61889  ,  0.22717  ,  0.26755  , -0.15587  ,
        0.11458  , -0.34497  ,  0.086328 , -0.27064  ,  0.54732  ,
        0.075819 ,  0.01787  , -0.57434  ,  0.06019  ,  0.28917  ,
       -0.43404  ,  0.84522  , -0.18297  ,  0.37544  , -0.073685 ,
       -0.14497  , -0.88175  , -0.33445  , -0.71079  , -0.5085   ,
       -0.069451 , -0.18155  , -0.41812  ,  0.10961  ,  0.34082  ,
        0.42849  ,  0.49135  ,  0.24293  ,  0.26177  ,  0.42277  ,
        0.41787  , -0.24921  ,  0.5677   ,  0.080152 , -0.11313  ,
       -0.53238  , -0.4329   ,  0.16515  ,  0.29339  ,  0.045954 ])
```

Implement and test the cosine measure of word similarity.

In [ ]:
```
import numpy as np
```

In [ ]:
```
## Write a function to compute the cosine similarity between two word vectors.
##       Demonstrate that it's symmetric with a few examples.
def cosine_similarity(x: np.ndarray, y: np.ndarray) -> float:
    dot_product = np.dot (x,y)
    x_norm = np.linalg.norm(x)
    y_norm = np.linalg.norm(y)
    res = dot_product/ (x_norm* y_norm)
    return res

def verify(x):
  if x: print("Verified")
  else: print("Failure to verify")

## Use some examples to demonstrate symmetry of your implementation.
verify(cosine_similarity(e['apples'],e['oranges'])==cosine_similarity(e['oranges'],e['

## TODO: add a few more examples here.
verify(cosine_similarity(e['car'],e['truck'])==cosine_similarity(e['truck'],e['car']))
verify(cosine_similarity(e['mars'],e['venus'])==cosine_similarity(e['venus'],e['mars']
verify(cosine_similarity(e['warm'],e['cool'])==cosine_similarity(e['cool'],e['warm']))
verify(cosine_similarity(e['red'],e['blue'])==cosine_similarity(e['blue'],e['red']))
```

```
Verified
Verified
Verified
Verified
Verified
```

In [ ]:
```
## Verify the sanity checks in part 1b of the pset PDF.
verify(cosine_similarity(e['car'],e['truck']) > cosine_similarity(e['car'],e['person']
verify(cosine_similarity(e['mars'],e['venus']) > cosine_similarity(e['mars'],e['goes']
verify(cosine_similarity(e['warm'],e['cool']) > cosine_similarity(e['warm'],e['yesterd
verify(cosine_similarity(e['red'],e['blue']) > cosine_similarity(e['red'],e['fast']))
```

```
Verified
Verified
Verified
Verified
```

In [ ]:
```
## TODO: come up with two examples that demonstrate correct similarity relations.
verify(cosine_similarity(e['lion'],e['tiger']) > cosine_similarity(e['lion'],e['car'])
verify(cosine_similarity(e['small'],e['big']) > cosine_similarity(e['small'],e['mars']
```

```
Verified
Verified
```

In [ ]:
```
## TODO: come up with two examples where cosine similarity doesn't align with your int
verify(cosine_similarity(e['lion'],e['cat']) > cosine_similarity(e['lion'],e['dog']))
verify(cosine_similarity(e['car'],e['driver']) > cosine_similarity(e['car'],e['truck']
```

```
Failure to verify
Failure to verify
```

{lion, cat} vs. {lion, dog}: We believe that "lion" is generally associated with the feline family and therefore would be more strongly associated with "cat", while "dog" is generally associated with animals, a concept that encompasses a wider range of species than just cats.

{car, driver} vs. {car, truck}: During our comparison of "car" and "driver" versus "car" and "truck," we noted that both pairs are associated with transportation and driving. However, they represent distinct entities and concepts within that domain. "Car" and "truck" are both tangible objects used for transportation, while "driver" represents a concept related to the individual operating the vehicle. This distinction could potentially explain why "car" was found to be more similar to "truck" than to "driver."

In [ ]:
```
## TODO: extra credit goes here if you want to do it.
def euclidean_distance(x: np.ndarray, y: np.ndarray):
  sqrt = np.square(x - y)
  sum = np.sum(sqrt)
  res = np.sqrt(sum)
  return res
```

In [ ]:
```
#Comparing of the same pair of words of the cosine similarity
verify(euclidean_distance(e['apples'],e['oranges'])==euclidean_distance(e['oranges'],e
verify(euclidean_distance(e['car'],e['truck']) > euclidean_distance(e['car'],e['persor
verify(euclidean_distance(e['mars'],e['venus']) > euclidean_distance(e['mars'],e['goes
verify(euclidean_distance(e['warm'],e['cool']) > euclidean_distance(e['warm'],e['yeste
verify(euclidean_distance(e['red'],e['blue']) > euclidean_distance(e['red'],e['fast'])
```

```
Verified
Failure to verify
Failure to verify
Failure to verify
Failure to verify
```

In [ ]:
```
## TODO: come up with two examples that demonstrate correct similarity relations.
#The same test
verify(euclidean_distance(e['lion'],e['tiger']) > euclidean_distance(e['lion'],e['car'
verify(euclidean_distance(e['small'],e['big']) > euclidean_distance(e['small'],e['mars
```

```
Failure to verify
Failure to verify
```

In [ ]:
```
## TODO: come up with two examples where cosine similarity doesn't align with your int
#The same test
verify(euclidean_distance(e['lion'],e['cat']) > euclidean_distance(e['lion'],e['dog'])
verify(euclidean_distance(e['car'],e['driver']) > euclidean_distance(e['car'],e['truck
```

```
Verified
Verified
```

Our observation revealed that for every "Verified" pair of similarity scores calculated using cosine similarity, we encountered "Failure to verify" pairs when employing Euclidean distance, and vice versa. This discrepancy can be attributed to the fundamental distinction in how these two metrics capture similarities between vectors. Cosine similarity focuses on measuring the similarity in direction or orientation of two vectors, while Euclidean distance considers both the similarity in magnitude and direction of the difference between them. Consequently, vectors with similar orientations but different magnitudes may be deemed similar by cosine similarity but dissimilar by Euclidean distance, resulting in contrasting similarity outcomes.

The analogies task

Given words *w1*, *w2*, and *w3*, find a word *x* such that *w1* : *w2* :: *w3* : *x*. For example, for the analogy problem *France*:*Paris* :: *England*:*x*, the answer should be *London*. To solve analogies using semantic vectors, letting $e(w)$ indicate the embedding for a word w, calculate a vector y = $e(w_2)$ − $e(w_1)$ + $e(w_3)$ and find the word whose vector is closest to y.

TODO: Explain why the analogy-solving method makes sense.

The analogy-solving method is logical because it utilizes word embeddings that are based on the inherent semantic relationships encoded within word vectors. Word embeddings assign words to high-dimensional vectors in a semantic space, where words with similar meanings are situated closer to each other compared to words with dissimilar meanings. These spatial relationships between vectors effectively capture the semantic connections between words.

By computing a vector that encapsulates the semantic relationship between two given words in an analogy, we can utilize this vector to predict an unknown word that shares a similar relationship to the third word. This prediction is accomplished by identifying the word whose vector is closest to the computed vector, utilizing a distance metric such as cosine similarity or Euclidean distance.

In [ ]:
```
## Write a function to calculate y as described above.
def analogy_vector(w1: str, w2: str, w3: str, e: dict) -> np.ndarray: # note that the
  y = e[w2]- e[w1] + e[w3]
  return y
```

In [ ]:
```
## Write a function to find the k nearest neighbors to y.
def analogy(w1: str, w2: str, w3: str, e: dict, k=5):
  y= analogy_vector(w1, w2, w3, e)
  nn= {} #nearest neighbors
  top_keys = []

  for word in e:
     nn[word] = cosine_similarity(y, e[word])
  nn = dict(sorted(nn.items(), key=lambda item: item[1], reverse= True))

  for i, key in enumerate(nn.keys()):
    if key!= w3:
      top_keys.append(key)
    if i == k:
        break

  return top_keys
```

In [ ]:
```
## Are the top 5 results for the following analogies sensible?
print(analogy("france","paris","england",e))
print(analogy("man","woman","king",e))
print(analogy("tall","taller","warm",e))
print(analogy("tall","short","england",e))
```

```
['london', 'manchester', 'birmingham', 'middlesex', 'liverpool']
['queen', 'monarch', 'throne', 'princess', 'mother']
['warmer', 'warmed', 'cooler', 'drier', 'colder']
['short', 'following', 'wales', 'ireland', 'britain']
```

It's good to see that most of the results make sense. Regarding the fourth analogy, it seems like the model may be struggling with finding an appropriate analogy since there isn't really an opposite of a county. It's possible that the model is picking up on other associations with the word "county", such as location or population size, which might explain why words like "city" and "following" are appearing in the results. Nonetheless, it's interesting to see how the model is attempting to find a relationship between the given words.

Let's analyze each analogy individually:

In the first analogy, "London" being the closest word to the vector "y" is as we anticipated. Additionally, all the other four words being cities is in line with our understanding.

Moving to the second analogy, "queen" being the closest word to the vector "y" is the correct analogy. We can observe that the other four words are related either to strong women or to monarchy, which is conceptually relevant.

In the third analogy, "warmer" being the closest word to the vector "y" indicates that the algorithm successfully recognizes similar semantic contexts. The presence of other words related to "cold" and "warm" is logical given the given words and context.

For the fourth analogy, the absence of an exact analogy is understandable since "tall" and "short" are antonyms, and there is no direct opposite for a county. Consequently, the appearance of other cities and the word "short" in the results can be considered reasonable. However, it is unclear why the word "following" is one of the closest words, as it has a different meaning.

Overall, while most of the results align with our expectations and make sense, there may be instances where further examination is needed to understand the associations made by the algorithm.

In [ ]:

```
## TODO: come up with 4 more analogies, 2 of which work in your opinion, and 2 of whic

#Some of the examples were taken from Psychometric exams, for the sport :)

#analogies which we believe work
print(analogy("mother","father","girl",e))
print(analogy("big","bigger","small",e))

#analogies which we believe won't work
print(analogy("house","person","kennel",e))
print(analogy("drink","water","play",e))
```

```
['boy', 'boys', 'father', 'son', 'man']
['larger', 'smaller', 'bigger', 'tiny', 'large']
['akc', 'person', 'purebred', 'breed', 'ukc']
['water', 'playing', 'played', 'plays', 'game']
```

TODO: Did you notice any patterns or generalizations while exploring possible analogies? For the ones that went wrong, why do you think they went wrong?

We agree that obtaining accurate results in analogies often requires precise alignment between the words. The presence of plural, comparative, and superlative forms of words can pose challenges for the algorithm to accurately predict appropriate contexts and analogies.

Let's examine the examples provided:

1) Paris is the **capital** of France, just as London is the **capital** of England.

2) "smaller" is the comparative form of the adjective "small", just like "big" and "bigger". Similar to the given example of "tall":"taller" :: "hot": "warmer". We can see that the algorithm is a little inaccurate in executing words correctly, and chooses larger before smaller.

3) "mother" and "father" are identified as male and female, just as "girl" and "boy" are identified as male and female.

For the examples that didn't work, the relationship between the words is a bit more complex:

1) A house is a **living place for humans**, while a kennel is a **living place for dogs**. We believe that the complication in understanding the context detracts from the algorithm's ability to understand the intent. That is, if the context is not direct, or alternatively does not point directly to the reason for the context - the algorithm will not be accurate.

2) We believed that since the act of drinking is related to **water**, the act of playing would be related to **game** and we would get the correct analogy. However, the best word we got was "water". In addition, the results also include variations of "play", which supports our previous conclusion. We believe that the analogy is also a bit complicated because you don't just drink water, and there are other things that can be played, and not just a game.

Indeed, complex analogies can pose challenges for the algorithm, even if they appear logical and structured to human thinking. While the algorithm excels at capturing certain linguistic patterns and relationships, it may struggle with more intricate contexts that require deeper understanding or contextual knowledge.

Analogical reasoning often relies on the ability to identify subtle similarities and abstract relationships between words. Human thinking is influenced by various factors, including background knowledge, cultural context, and personal experiences, which allow us to make connections that may not be apparent solely from word embeddings.

The algorithm's performance is limited to the information it has been trained on and the patterns it has learned from the data. It may struggle to grasp the intricacies of certain analogies that require additional contextual understanding beyond the surface-level associations present in the word embeddings.

While the algorithm may fall short in complex scenarios, it is important to acknowledge its capabilities in recognizing more straightforward analogies and capturing certain linguistic patterns accurately. Continued research and advancements in natural language processing aim to improve the algorithms' ability to handle more complex analogies and align them more closely with human reasoning.

3) Using semantic vectors to decode brain activation

Load the data

In [ ]:
```
# Download and extract the data and learn_decoder.py
!wget --load-cookies /tmp/cookies.txt "https://docs.google.com/uc?export=download&conf
!unzip files.zip
!rm files.zip
```

```
--2023-05-17 10:40:12--  https://docs.google.com/uc?export=download&confirm=t&id=1xZa
orRH-xxjfochvSesAhOTUg82_Xq56
Resolving docs.google.com (docs.google.com)... 172.253.123.139, 172.253.123.100, 172.
253.123.102, ...
Connecting to docs.google.com (docs.google.com)|172.253.123.139|:443... connected.
HTTP request sent, awaiting response... 303 See Other
Location: https://doc-0g-54-docs.googleusercontent.com/docs/securesc/ha0ro937gcuc7l7d
effksulhg5h7mbp1/nao6715j3gc8qe14fkbgtkb20njc7dnm/1684320000000/01333689271208460322/
*/1xZaorRH-xxjfochvSesAhOTUg82_Xq56?e=download&uuid=0b5ba3aa-662d-415e-9033-ad663ed54
4c1 [following]
Warning: wildcards not supported in HTTP.
--2023-05-17 10:40:12--  https://doc-0g-54-docs.googleusercontent.com/docs/securesc/h
a0ro937gcuc7l7deffksulhg5h7mbp1/nao6715j3gc8qe14fkbgtkb20njc7dnm/1684320000000/013336
89271208460322/*/1xZaorRH-xxjfochvSesAhOTUg82_Xq56?e=download&uuid=0b5ba3aa-662d-415e
-9033-ad663ed544c1
Resolving doc-0g-54-docs.googleusercontent.com (doc-0g-54-docs.googleusercontent.co
m)... 172.217.203.132, 2607:f8b0:400c:c07::84
Connecting to doc-0g-54-docs.googleusercontent.com (doc-0g-54-docs.googleusercontent.
com)|172.217.203.132|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 97708666 (93M) [application/x-zip-compressed]
Saving to: ‘files.zip’

files.zip           100%[===================>]  93.18M   121MB/s    in 0.8s

2023-05-17 10:40:13 (121 MB/s) - ‘files.zip’ saved [97708666/97708666]

Archive:  files.zip
  inflating: stimuli_180concepts.txt
  inflating: learn_decoder.py
  inflating: vectors_180concepts.GV42B300.txt
  inflating: imaging_data.csv
```

In [ ]:
```
#Let's load the functions from learn_decoder.py
from learn_decoder import *

#and the data
data = read_matrix("imaging_data.csv", sep=",")
vectors = read_matrix("vectors_180concepts.GV42B300.txt", sep=" ")
concepts = np.genfromtxt('stimuli_180concepts.txt', dtype=np.dtype('U')) #The names of
```

You can verify for your self what learn_decoder consists of by going to Files and opening it.

What are the Accuracy scores?

Define a function that computes rank-based accuracy score, then, iterate over the 18 folds. For each fold, train the decoder **using the** `learn_decoder` **function** (the function is already imported from `learn_decoder.py`) on the fold train data, obtain the predicions on the fold test data, and store both the accuracy score of each concept (use the labels from `concepts`) as well as the average score of the 10 concepts.

In [ ]:
```
#calculate the rank of the true vector based on cosine similarity
def rank_based_accuracy (decoded_vec, true_vec):
  rank= {}
  vec_index = -1

  for i in range(vectors.shape[0]):
    rank[i]= cosine_similarity(vectors[i], decoded_vec)
    if np.array_equal(true_vec, vectors[i]):
      vec_index=i

  rankings = dict(sorted(rank.items(), key=lambda item: item[1], reverse= True))
  final_rank = list(rankings).index(vec_index)

  return final_rank+1
```

In [ ]:
```
from numpy.lib.function_base import average
ranks = {} #rank of each concept
avg_ranking = {} #avg rank of each fold

def calculate_rank(decoder_res, test_set, test_vectors, index):
  test_ranking= []
  key= (index/10)+1
  for i in range(len(test_set)):
    dot_prod = np.dot(test_set[i],decoder_res) #semantic vector - the model's best gue
    current_rank= rank_based_accuracy(dot_prod, test_vectors[i]) #rank of the current
    test_ranking.append(current_rank)
    ranks[concepts[index]]= current_rank
    index= index+1

    avg_ranking[key]= average(test_ranking)
```

In [ ]:
```
from sklearn.model_selection import KFold

# Set up KFold with k = 18
kf = KFold(n_splits=18, shuffle=False)

accuracy_scores = []
```

```
i=0
# Iterate through the 18 different training and test sets
for train_index, test_index in kf.split(data):
    train_data, test_data = data[train_index], data[test_index]
    train_vectors, test_vectors = vectors[train_index], vectors[test_index]

    decoder_res= learn_decoder(train_data, train_vectors) #decoder matrix

    calculate_rank(decoder_res, test_data, test_vectors, i) #calculate rank for each c

    if i==180:
      i=180
    else:
      i+=10
```

In [ ]:

```
for i in avg_ranking:
   print(f'average score of fold num {i} is: {avg_ranking[i]}')
```

```
average score of fold num 1.0 is: 66.7
average score of fold num 2.0 is: 62.3
average score of fold num 3.0 is: 60.4
average score of fold num 4.0 is: 70.6
average score of fold num 5.0 is: 81.3
average score of fold num 6.0 is: 74.5
average score of fold num 7.0 is: 77.0
average score of fold num 8.0 is: 46.7
average score of fold num 9.0 is: 105.1
average score of fold num 10.0 is: 39.1
average score of fold num 11.0 is: 65.6
average score of fold num 12.0 is: 56.5
average score of fold num 13.0 is: 36.9
average score of fold num 14.0 is: 66.0
average score of fold num 15.0 is: 41.7
average score of fold num 16.0 is: 36.8
average score of fold num 17.0 is: 39.7
average score of fold num 18.0 is: 87.5
```

Now let's plot the averaged accuracy score for each fold

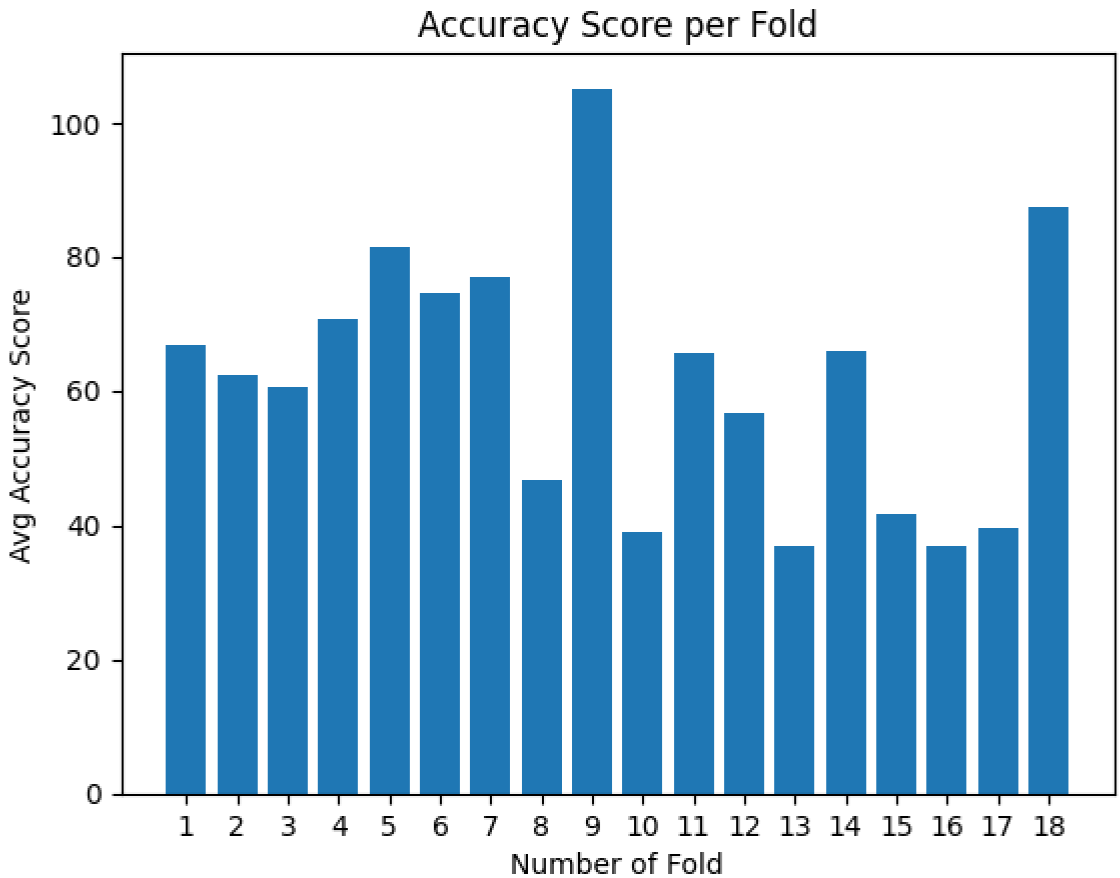
In [ ]:

```
import matplotlib.pyplot as plt
# Plot the accuracy scores for each fold

x_axis =np.arange(1,19)
y_axis = avg_ranking.values()
plt.bar(x_axis, y_axis)
x_tick = plt.xticks(range(min(x_axis), max(x_axis)+1))
plt.title("Accuracy Score per Fold")
plt.xlabel("Number of Fold")
plt.ylabel("Avg Accuracy Score")
```

Out[ ]:

```
Text(0, 0.5, 'Avg Accuracy Score')
```



Which concepts can be decoded with more or less success?

We'll consider a "successful concept" as a concept with an average rank lower than 90, as a score of 90 represents random noise.

In [ ]:
```
keys_values_greater_than_90 = []
for key, value in ranks.items():
    if value > 90:
        keys_values_greater_than_90.append((key, value))


keys_values_greater_than_90 = dict(sorted(dict(keys_values_greater_than_90).items(),ke
print(f'Failed Concepts:{keys_values_greater_than_90}' )
```

```
Failed Concepts:{'ball': 94, 'king': 94, 'weak': 96, 'deliberately': 97, 'trial': 98,
'clothes': 99, 'gun': 99, 'dedication': 101, 'burn': 102, 'engine': 105, 'garbage': 1
13, 'jungle': 114, 'liar': 115, 'prison': 117, 'counting': 118, 'noise': 118, 'wash':
119, 'dessert': 120, 'camera': 126, 'driver': 126, 'invention': 126, 'weather': 126,
'charity': 128, 'illness': 128, 'disturb': 134, 'bed': 135, 'obligation': 135, 'willi
ngly': 139, 'invisible': 140, 'residence': 141, 'emotionally': 145, 'kindness': 145,
'ignorance': 146, 'mathematical': 150, 'vacation': 150, 'sin': 152, 'sew': 157, 'eleg
ance': 158, 'usable': 158, 'movie': 159, 'dissolve': 164, 'electron': 168, 'deceive':
171, 'applause': 175, 'cockroach': 178, 'argumentatively': 180}
```

In [ ]:
```
keys_values_smaller_than_90 = []
for key, value in ranks.items():
    if value < 90:
        keys_values_smaller_than_90.append((key, value))
```

```
keys_values_smaller_than_90 = dict(sorted(dict(keys_values_smaller_than_90).items(),ke
print(f'Successful Concepts:{keys_values_smaller_than_90}' )
```

```
Successful Concepts:{'do': 1, 'food': 1, 'time': 1, 'great': 2, 'laugh': 4, 'stupid':
5, 'lady': 6, 'left': 6, 'hair': 7, 'money': 7, 'ability': 8, 'big': 8, 'play': 8, 'r
elationship': 8, 'crazy': 9, 'music': 9, 'picture': 9, 'building': 10, 'constructio
n': 10, 'feeling': 10, 'extremely': 11, 'dinner': 12, 'silly': 12, 'help': 13, 'ligh
t': 13, 'wear': 13, 'word': 13, 'read': 14, 'shape': 14, 'show': 14, 'soul': 14, 'dam
age': 15, 'fish': 15, 'skin': 15, 'successful': 15, 'team': 15, 'event': 16, 'qualit
y': 16, 'art': 17, 'attitude': 17, 'mountain': 17, 'road': 18, 'dog': 19, 'unaware':
20, 'poor': 21, 'taste': 21, 'tried': 21, 'angry': 22, 'pain': 22, 'business': 24, 'p
lan': 25, 'science': 25, 'war': 25, 'student': 28, 'body': 29, 'sign': 29, 'broken':
31, 'pleasure': 31, 'tree': 31, 'carefully': 32, 'star': 33, 'economy': 34, 'level':
35, 'movement': 35, 'smiling': 35, 'solution': 35, 'election': 36, 'typical': 36, 'us
eless': 36, 'challenge': 37, 'law': 37, 'sound': 37, 'tool': 38, 'sell': 40, 'magic':
42, 'sad': 42, 'brain': 43, 'dangerous': 44, 'fight': 44, 'material': 44, 'personalit
y': 44, 'professional': 44, 'protection': 44, 'bag': 45, 'dig': 45, 'blood': 46, 'phi
losophy': 47, 'impress': 48, 'land': 48, 'bear': 49, 'investigation': 51, 'sugar': 5
1, 'news': 52, 'table': 52, 'experiment': 53, 'pig': 53, 'sexy': 53, 'smart': 53, 'te
xture': 54, 'bird': 58, 'ship': 58, 'job': 59, 'spoke': 59, 'emotion': 60, 'plant': 6
0, 'code': 61, 'beer': 62, 'beat': 63, 'dance': 64, 'gold': 64, 'flow': 65, 'apartmen
t': 66, 'reaction': 66, 'cook': 67, 'suspect': 67, 'argument': 68, 'accomplished': 6
9, 'device': 70, 'charming': 71, 'computer': 72, 'nation': 72, 'dressing': 74, 'medic
ation': 76, 'collection': 77, 'doctor': 77, 'bar': 79, 'disease': 80, 'religious': 8
3, 'toy': 83, 'marriage': 85, 'mechanism': 87, 'seafood': 87, 'hurting': 88, 'deliver
y': 89}
```

In [ ]:

```
print(f'successful concepts: {keys_values_smaller_than_90}')
print(f'failed_concepts: {keys_values_greater_than_90}')

print(f'number of successful concepts: {len(keys_values_smaller_than_90)}')
print(f'number of failed_concepts: {len(keys_values_greater_than_90)}')
```

```
successful concepts: {'do': 1, 'food': 1, 'time': 1, 'great': 2, 'laugh': 4, 'stupi
d': 5, 'lady': 6, 'left': 6, 'hair': 7, 'money': 7, 'ability': 8, 'big': 8, 'play':
8, 'relationship': 8, 'crazy': 9, 'music': 9, 'picture': 9, 'building': 10, 'construc
tion': 10, 'feeling': 10, 'extremely': 11, 'dinner': 12, 'silly': 12, 'help': 13, 'li
ght': 13, 'wear': 13, 'word': 13, 'read': 14, 'shape': 14, 'show': 14, 'soul': 14, 'd
amage': 15, 'fish': 15, 'skin': 15, 'successful': 15, 'team': 15, 'event': 16, 'quali
ty': 16, 'art': 17, 'attitude': 17, 'mountain': 17, 'road': 18, 'dog': 19, 'unaware':
20, 'poor': 21, 'taste': 21, 'tried': 21, 'angry': 22, 'pain': 22, 'business': 24, 'p
lan': 25, 'science': 25, 'war': 25, 'student': 28, 'body': 29, 'sign': 29, 'broken':
31, 'pleasure': 31, 'tree': 31, 'carefully': 32, 'star': 33, 'economy': 34, 'level':
35, 'movement': 35, 'smiling': 35, 'solution': 35, 'election': 36, 'typical': 36, 'us
eless': 36, 'challenge': 37, 'law': 37, 'sound': 37, 'tool': 38, 'sell': 40, 'magic':
42, 'sad': 42, 'brain': 43, 'dangerous': 44, 'fight': 44, 'material': 44, 'personalit
y': 44, 'professional': 44, 'protection': 44, 'bag': 45, 'dig': 45, 'blood': 46, 'phi
losophy': 47, 'impress': 48, 'land': 48, 'bear': 49, 'investigation': 51, 'sugar': 5
1, 'news': 52, 'table': 52, 'experiment': 53, 'pig': 53, 'sexy': 53, 'smart': 53, 'te
xture': 54, 'bird': 58, 'ship': 58, 'job': 59, 'spoke': 59, 'emotion': 60, 'plant': 6
0, 'code': 61, 'beer': 62, 'beat': 63, 'dance': 64, 'gold': 64, 'flow': 65, 'apartmen
t': 66, 'reaction': 66, 'cook': 67, 'suspect': 67, 'argument': 68, 'accomplished': 6
9, 'device': 70, 'charming': 71, 'computer': 72, 'nation': 72, 'dressing': 74, 'medic
ation': 76, 'collection': 77, 'doctor': 77, 'bar': 79, 'disease': 80, 'religious': 8
3, 'toy': 83, 'marriage': 85, 'mechanism': 87, 'seafood': 87, 'hurting': 88, 'deliver
y': 89}
failed_concepts: {'ball': 94, 'king': 94, 'weak': 96, 'deliberately': 97, 'trial': 9
8, 'clothes': 99, 'gun': 99, 'dedication': 101, 'burn': 102, 'engine': 105, 'garbag
e': 113, 'jungle': 114, 'liar': 115, 'prison': 117, 'counting': 118, 'noise': 118, 'w
ash': 119, 'dessert': 120, 'camera': 126, 'driver': 126, 'invention': 126, 'weather':
126, 'charity': 128, 'illness': 128, 'disturb': 134, 'bed': 135, 'obligation': 135,
'willingly': 139, 'invisible': 140, 'residence': 141, 'emotionally': 145, 'kindness':
145, 'ignorance': 146, 'mathematical': 150, 'vacation': 150, 'sin': 152, 'sew': 157,
'elegance': 158, 'usable': 158, 'movie': 159, 'dissolve': 164, 'electron': 168, 'dece
ive': 171, 'applause': 175, 'cockroach': 178, 'argumentatively': 180}
number of successful concepts: 134
number of failed_concepts: 46
```

Here are the 10 that failed the most and the 10 that succeeded the most:

In [ ]:
```python
import heapq

# Get the 10 largest elements from the dictionary based on their values
largest_items = dict(heapq.nlargest(10, keys_values_greater_than_90.items(), key=lambd

print(f'10 with the largest rank: {largest_items}')
```

```
10 with the largest rank: {'argumentatively': 180, 'cockroach': 178, 'applause': 175,
'deceive': 171, 'electron': 168, 'dissolve': 164, 'movie': 159, 'elegance': 158, 'usa
ble': 158, 'sew': 157}
```

In [ ]:
```python
# Get the 10 smallest elements from the dictionary based on their values
largest_items = dict(heapq.nlargest(10, keys_values_smaller_than_90.items(), key=lambd

print(f'10 with the smallest rank: {largest_items}')
```

```
10 with the smallest rank: {'delivery': 89, 'hurting': 88, 'mechanism': 87, 'seafoo
d': 87, 'marriage': 85, 'religious': 83, 'toy': 83, 'disease': 80, 'bar': 79, 'collec
tion': 77}
```

Are the results satisfactory, in your opinion? Why or why not?

The results are satisfactory. It can be seen that 134 results are successful compared to 46 that failed(out of 180). Furthermore, the top 10 most successful results contain common words in everyday language such as "do", "food", "time" and "great".

Export to PDF

Run the following cell to download the notebook as a nicely formatted pdf file.

In [ ]:
```
# Add to a new cell at the end of the notebook and run the follow code,
# which will save the notebook as pdf in your google drive (allow the permissions) and

!wget -nc https://raw.githubusercontent.com/brpy/colab-pdf/master/colab_pdf.py

from colab_pdf import colab_pdf

# If you saved the notebook in the default location in your Google Drive,
# and didn't change the name of the file, the code should work as is.
# If not, adapt accordingly.

colab_pdf(file_name='Copy of Pset_3.ipynb', notebookpath="/content/drive/MyDrive/Colab
```

```
File ‘colab_pdf.py’ already there; not retrieving.


WARNING: apt does not have a stable CLI interface. Use with caution in scripts.


WARNING: apt does not have a stable CLI interface. Use with caution in scripts.

E: Unable to locate package texlive-generic-recommended
[NbConvertApp] WARNING | pattern '$notebookpath$file_name' matched no files
This application is used to convert notebook files (*.ipynb)
        to various other formats.

        WARNING: THE COMMANDLINE INTERFACE MAY CHANGE IN FUTURE RELEASES.

Options
=======
The options below are convenience aliases to configurable class-options,
as listed in the "Equivalent to" description-line of the aliases.
To see all configurable class-options for some <cmd>, use:
    <cmd> --help-all

--debug
    set log level to logging.DEBUG (maximize logging output)
    Equivalent to: [--Application.log_level=10]
--show-config
    Show the application's configuration (human-readable format)
    Equivalent to: [--Application.show_config=True]
--show-config-json
    Show the application's configuration (json format)
    Equivalent to: [--Application.show_config_json=True]
--generate-config
    generate default config file
    Equivalent to: [--JupyterApp.generate_config=True]
-y
    Answer yes to any questions instead of prompting.
    Equivalent to: [--JupyterApp.answer_yes=True]
--execute
    Execute the notebook prior to export.
    Equivalent to: [--ExecutePreprocessor.enabled=True]
--allow-errors
    Continue notebook execution even if one of the cells throws an error and include
the error message in the cell output (the default behaviour is to abort conversion).
This flag is only relevant if '--execute' was specified, too.
    Equivalent to: [--ExecutePreprocessor.allow_errors=True]
--stdin
    read a single notebook file from stdin. Write the resulting notebook with default
basename 'notebook.*'
    Equivalent to: [--NbConvertApp.from_stdin=True]
--stdout
    Write notebook output to stdout instead of files.
    Equivalent to: [--NbConvertApp.writer_class=StdoutWriter]
--inplace
    Run nbconvert in place, overwriting the existing notebook (only
            relevant when converting to notebook format)
    Equivalent to: [--NbConvertApp.use_output_suffix=False --NbConvertApp.export_form
at=notebook --FilesWriter.build_directory=]
--clear-output
    Clear output of current file and save in place,
            overwriting the existing notebook.
```

```
    Equivalent to: [--NbConvertApp.use_output_suffix=False --NbConvertApp.export_form
at=notebook --FilesWriter.build_directory= --ClearOutputPreprocessor.enabled=True]
--no-prompt
    Exclude input and output prompts from converted document.
    Equivalent to: [--TemplateExporter.exclude_input_prompt=True --TemplateExporter.e
xclude_output_prompt=True]
--no-input
    Exclude input cells and output prompts from converted document.
            This mode is ideal for generating code-free reports.
    Equivalent to: [--TemplateExporter.exclude_output_prompt=True --TemplateExporter.
exclude_input=True --TemplateExporter.exclude_input_prompt=True]
--allow-chromium-download
    Whether to allow downloading chromium if no suitable version is found on the syst
em.
    Equivalent to: [--WebPDFExporter.allow_chromium_download=True]
--disable-chromium-sandbox
    Disable chromium security sandbox when converting to PDF..
    Equivalent to: [--WebPDFExporter.disable_sandbox=True]
--show-input
    Shows code input. This flag is only useful for dejavu users.
    Equivalent to: [--TemplateExporter.exclude_input=False]
--embed-images
    Embed the images as base64 dataurls in the output. This flag is only useful for t
he HTML/WebPDF/Slides exports.
    Equivalent to: [--HTMLExporter.embed_images=True]
--sanitize-html
    Whether the HTML in Markdown cells and cell outputs should be sanitized..
    Equivalent to: [--HTMLExporter.sanitize_html=True]
--log-level=<Enum>
    Set the log level by value or name.
    Choices: any of [0, 10, 20, 30, 40, 50, 'DEBUG', 'INFO', 'WARN', 'ERROR', 'CRITIC
AL']
    Default: 30
    Equivalent to: [--Application.log_level]
--config=<Unicode>
    Full path of a config file.
    Default: ''
    Equivalent to: [--JupyterApp.config_file]
--to=<Unicode>
    The export format to be used, either one of the built-in formats
            ['asciidoc', 'custom', 'html', 'latex', 'markdown', 'notebook', 'pdf', 'p
ython', 'rst', 'script', 'slides', 'webpdf']
            or a dotted object name that represents the import path for an
            ``Exporter`` class
    Default: ''
    Equivalent to: [--NbConvertApp.export_format]
--template=<Unicode>
    Name of the template to use
    Default: ''
    Equivalent to: [--TemplateExporter.template_name]
--template-file=<Unicode>
    Name of the template file to use
    Default: None
    Equivalent to: [--TemplateExporter.template_file]
--theme=<Unicode>
    Template specific theme(e.g. the name of a JupyterLab CSS theme distributed
    as prebuilt extension for the lab template)
    Default: 'light'
    Equivalent to: [--HTMLExporter.theme]
--sanitize_html=<Bool>
```

```
    Whether the HTML in Markdown cells and cell outputs should be sanitized.This
    should be set to True by nbviewer or similar tools.
    Default: False
    Equivalent to: [--HTMLExporter.sanitize_html]
--writer=<DottedObjectName>
    Writer class used to write the
                                    results of the conversion
    Default: 'FilesWriter'
    Equivalent to: [--NbConvertApp.writer_class]
--post=<DottedOrNone>
    PostProcessor class used to write the
                                    results of the conversion
    Default: ''
    Equivalent to: [--NbConvertApp.postprocessor_class]
--output=<Unicode>
    overwrite base name use for output files.
                can only be used when converting one notebook at a time.
    Default: ''
    Equivalent to: [--NbConvertApp.output_base]
--output-dir=<Unicode>
    Directory to write output(s) to. Defaults
                              to output to the directory of each notebook. To rec
over
                              previous default behaviour (outputting to the curre
nt
                              working directory) use . as the flag value.
    Default: ''
    Equivalent to: [--FilesWriter.build_directory]
--reveal-prefix=<Unicode>
    The URL prefix for reveal.js (version 3.x).
            This defaults to the reveal CDN, but can be any url pointing to a copy
            of reveal.js.
            For speaker notes to work, this must be a relative path to a local
            copy of reveal.js: e.g., "reveal.js".
            If a relative path is given, it must be a subdirectory of the
            current directory (from which the server is run).
            See the usage documentation
            (https://nbconvert.readthedocs.io/en/latest/usage.html#reveal-js-html-sli
deshow)
            for more details.
    Default: ''
    Equivalent to: [--SlidesExporter.reveal_url_prefix]
--nbformat=<Enum>
    The nbformat version to write.
            Use this to downgrade notebooks.
    Choices: any of [1, 2, 3, 4]
    Default: 4
    Equivalent to: [--NotebookExporter.nbformat_version]

Examples
--------

    The simplest way to use nbconvert is

            > jupyter nbconvert mynotebook.ipynb --to html

            Options include ['asciidoc', 'custom', 'html', 'latex', 'markdown', 'note
book', 'pdf', 'python', 'rst', 'script', 'slides', 'webpdf'].

            > jupyter nbconvert --to latex mynotebook.ipynb
```

```
                Both HTML and LaTeX support multiple output templates. LaTeX includes
                'base', 'article' and 'report'.  HTML includes 'basic', 'lab' and
                'classic'. You can specify the flavor of the format used.

                > jupyter nbconvert --to html --template lab mynotebook.ipynb

                You can also pipe the output to stdout, rather than a file

                > jupyter nbconvert mynotebook.ipynb --stdout

                PDF is generated via latex

                > jupyter nbconvert mynotebook.ipynb --to pdf

                You can get (and serve) a Reveal.js-powered slideshow

                > jupyter nbconvert myslides.ipynb --to slides --post serve

                Multiple notebooks can be given at the command line in a couple of
                different ways:

                > jupyter nbconvert notebook*.ipynb
                > jupyter nbconvert notebook1.ipynb notebook2.ipynb

                or you can specify the notebooks list in a config file, containing::

                    c.NbConvertApp.notebooks = ["my_notebook.ipynb"]

                > jupyter nbconvert --config mycfg.py

To see all available configurables, use `--help-all`.
```

Out[ ]:
```
'File Download Unsuccessful. Saved in Google Drive'
```