Exploring the Impact of the Neurologic Disease Migraine

Math 491 - Ind. Study

Nicolas Huang

Dr. Johnson

3/12/2023

Licensing Disclosure

All works done, hereinafter, falls under the GNU's General Public License 3 [GPL.V3]. If a copy is not provided within this project, a copy may be obtained at the following URL: https://www.gnu.org/licenses/gpl-3.0.en.html

In [81]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats as stats
import statsmodels.stats.proportion as sp
import os
import re
```

1. Import Data Set

In [2]:
```
original_data = pd.read_csv('phase3_deidentify.csv')  # Read csv file
original_data.head(20)
```

Out[2]:

	StartDate	EndDate	RecordedDate	ResponseId	ie_1	age	diagnosis_1	diagnosis_2	diagnosis_3	diagnosis_duration	...	Q84_4	Q84_5	Q84_6	Q84_7	Q84_8	Q84_9
0	3/27/2019 11:12	3/27/2019 11:24	3/27/2019 11:24	R_2ts85NcEzqOJoBB	4	27	1.0	1.0	NaN	5	...	3.0	3.0	3.0	3.0	3.0	3.0
1	3/27/2019 12:05	3/27/2019 12:21	3/27/2019 12:22	R_332AvILBeOnLK6W	4	39	NaN	1.0	NaN	5	...	1.0	1.0	1.0	1.0	1.0	1.0
2	3/27/2019 12:14	3/27/2019 13:10	3/27/2019 13:10	R_ZI5VZNgv59ZIcxz	4	33	NaN	1.0	NaN	5	...	1.0	1.0	1.0	1.0	1.0	1.0
3	3/27/2019 12:26	3/27/2019 13:11	3/27/2019 13:11	R_0DiivCGGTiATfFL	4	36	NaN	1.0	NaN	5	...	1.0	1.0	1.0	1.0	1.0	1.0
4	3/27/2019 12:58	3/27/2019 13:24	3/27/2019 13:24	R_1lABVgQLfw24sHB	4	32	1.0	NaN	NaN	5	...	1.0	2.0	1.0	1.0	1.0	1.0
5	3/27/2019 16:06	3/27/2019 16:27	3/27/2019 16:27	R_3G1XT7c87xz2dzj	4	36	NaN	1.0	NaN	5	...	2.0	2.0	1.0	1.0	1.0	1.0
6	3/27/2019 16:06	3/27/2019 16:46	3/27/2019 16:46	R_2tG8LS1lfDbWe69	4	61	1.0	NaN	NaN	5	...	1.0	1.0	1.0	1.0	1.0	1.0
7	3/27/2019 20:26	3/27/2019 20:43	3/27/2019 20:43	R_1kOEVaOnqE4JbSe	4	35	NaN	1.0	NaN	5	...	1.0	1.0	1.0	1.0	1.0	1.0
8	3/27/2019 21:40	3/27/2019 21:49	3/27/2019 21:49	R_2OOT37jz3ozHGwi	4	45	1.0	NaN	NaN	5	...	1.0	2.0	1.0	1.0	1.0	1.0
9	3/27/2019 13:39	3/27/2019 14:07	3/28/2019 5:19	R_6ta5PIs0y9iOTrb	4	36	1.0	NaN	NaN	5	...	2.0	1.0	1.0	1.0	1.0	1.0
10	3/29/2019 8:16	3/29/2019 8:33	3/29/2019 8:33	R_1rjrp1fdEopZbLS	4	57	1.0	NaN	NaN	5	...	1.0	1.0	1.0	1.0	1.0	1.0
11	3/29/2019 8:10	3/29/2019 8:41	3/29/2019 8:41	R_2qn1d8VnXB1QGXs	4	32	1.0	NaN	NaN	5	...	5.0	5.0	5.0	1.0	1.0	5.0
12	3/29/2019 8:10	3/29/2019 9:07	3/29/2019 9:07	R_2wjfUqLHY2Ga83E	4	52	1.0	NaN	NaN	5	...	2.0	3.0	1.0	1.0	2.0	4.0
13	3/29/2019 9:15	3/29/2019 9:52	3/29/2019 9:53	R_DdZ4xD5xnaAcaqd	4	60	NaN	1.0	NaN	5	...	2.0	1.0	1.0	1.0	1.0	4.0
14	3/29/2019 13:06	3/29/2019 15:59	3/29/2019 15:59	R_3HzdU7cYYyCt5Qp	4	45	1.0	1.0	NaN	5	...	1.0	1.0	1.0	1.0	1.0	3.0
15	3/29/2019 9:38	3/29/2019 10:04	3/30/2019 6:54	R_YY8UKpSNwwL57ZT	4	38	NaN	1.0	NaN	5	...	4.0	3.0	3.0	2.0	2.0	1.0
16	3/30/2019 9:02	3/30/2019 9:22	3/30/2019 9:22	R_1OqSzcjKZWkqvHz	4	67	1.0	NaN	NaN	5	...	1.0	2.0	2.0	2.0	1.0	2.0
17	3/30/2019 19:35	3/30/2019 21:03	3/30/2019 21:03	R_1BYs8M2DJC9pYqX	4	34	NaN	1.0	NaN	5	...	1.0	1.0	1.0	1.0	1.0	1.0
18	3/30/2019 21:18	3/30/2019 21:45	3/30/2019 21:46	R_tSBDWuXyvNSjIlP	4	27	1.0	NaN	NaN	5	...	1.0	1.0	1.0	1.0	1.0	1.0
19	3/31/2019 13:36	3/31/2019 14:17	3/31/2019 14:17	R_323EE9TxohdmYZ1	4	44	NaN	1.0	NaN	5	...	2.0	2.0	2.0	1.0	2.0	3.0

20 rows × 238 columns

2. Clean Up Data

In [3]:
```
migraine_data = original_data  # Copy master data for modification
migraine_data.head(10)
```

Out[3]:

```
   StartDate        EndDate          RecordedDate     ResponseId          ie_1  age  diagnosis_1  diagnosis_2  diagnosis_3  diagnosis_duration  ...  Q84_4  Q84_5  Q84_6  Q84_7  Q84_8  Q84_9  Q8
0  3/27/2019 11:12  3/27/2019 11:24  3/27/2019 11:24  R_2ts85NcEzqOJoBB   4     27          1.0          1.0          NaN                   5  ...    3.0    3.0    3.0    3.0    3.0    3.0
1  3/27/2019 12:05  3/27/2019 12:21  3/27/2019 12:22  R_332AvILBeOnLK6W   4     39          NaN          1.0          NaN                   5  ...    1.0    1.0    1.0    1.0    1.0    1.0
2  3/27/2019 12:14  3/27/2019 13:10  3/27/2019 13:10  R_ZI5VZNgv59ZIcxz   4     33          NaN          1.0          NaN                   5  ...    1.0    1.0    1.0    1.0    1.0    1.0
3  3/27/2019 12:26  3/27/2019 13:11  3/27/2019 13:11  R_0DiivCGGTiATfFL   4     36          NaN          1.0          NaN                   5  ...    1.0    1.0    1.0    1.0    1.0    1.0
4  3/27/2019 12:58  3/27/2019 13:24  3/27/2019 13:24  R_1lABVgQLfw24sHB   4     32          1.0          NaN          NaN                   5  ...    1.0    2.0    1.0    1.0    1.0    1.0
5  3/27/2019 16:06  3/27/2019 16:27  3/27/2019 16:27  R_3G1XT7c87xz2dzj   4     36          NaN          1.0          NaN                   5  ...    2.0    2.0    1.0    1.0    1.0    1.0
6  3/27/2019 16:06  3/27/2019 16:46  3/27/2019 16:46  R_2tG8LS1lfDbWe69   4     61          1.0          NaN          NaN                   5  ...    1.0    1.0    1.0    1.0    1.0    1.0
7  3/27/2019 20:26  3/27/2019 20:43  3/27/2019 20:43  R_1kOEVaOnqE4JbSe   4     35          NaN          1.0          NaN                   5  ...    1.0    1.0    1.0    1.0    1.0    1.0
8  3/27/2019 21:40  3/27/2019 21:49  3/27/2019 21:49  R_2OOT37jz3ozHGwi   4     45          1.0          NaN          NaN                   5  ...    1.0    2.0    1.0    1.0    1.0    1.0
9  3/27/2019 13:39  3/27/2019 14:07  3/28/2019 5:19   R_6ta5PIs0y9iOTrb   4     36          1.0          NaN          NaN                   5  ...    2.0    1.0    1.0    1.0    1.0    1.0

10 rows × 238 columns
```

In [4]:

```python
# Drop Date Columns, ie_1 (permission for survey), consent, and other columns that aren't needed
migraine_data = migraine_data.drop(columns = ['StartDate', 'EndDate', 'RecordedDate', 'ie_1', 'consent', 'ResponseId'])

# Add UserID column instead of ResponseID
migraine_data = migraine_data.assign(userID = range(len(migraine_data)))
migraine_data.head(10)
```

Out[4]:

```
   age  diagnosis_1  diagnosis_2  diagnosis_3  diagnosis_duration  mg_6_months  cm_3_years  id_migraine_1  id_migraine_2  id_migraine_3  ...  Q84_5  Q84_6  Q84_7  Q84_8  Q84_9  Q8
0   27          1.0          1.0          NaN                   5            4           1              2              1              1  ...    3.0    3.0    3.0    3.0    3.0
1   39          NaN          1.0          NaN                   5            5           2              1              1              1  ...    1.0    1.0    1.0    1.0    1.0
2   33          NaN          1.0          NaN                   5           30           2              1              1              2  ...    1.0    1.0    1.0    1.0    1.0
3   36          NaN          1.0          NaN                   5            5           2              1              1              1  ...    1.0    1.0    1.0    1.0    1.0
4   32          1.0          NaN          NaN                   5           14           1              1              1              2  ...    2.0    1.0    1.0    1.0    1.0
5   36          NaN          1.0          NaN                   5           10           1              1              1              1  ...    2.0    1.0    1.0    1.0    1.0
6   61          1.0          NaN          NaN                   5           30           1              1              1              1  ...    1.0    1.0    1.0    1.0    1.0
7   35          NaN          1.0          NaN                   5           30           2              1              1              1  ...    1.0    1.0    1.0    1.0    1.0
8   45          1.0          NaN          NaN                   5           24           1              1              1              1  ...    2.0    1.0    1.0    1.0    1.0
9   36          1.0          NaN          NaN                   5           30           1              1              1              1  ...    1.0    1.0    1.0    1.0    1.0

10 rows × 233 columns
```

2.1 Fix Race - Add New Column for race_6_text and race_6

In [5]:

```python
# Edit the ethnicity tab, if they're latine set to 1, if not set to 0 (easier to calculate)
migraine_data.loc[migraine_data['ethnicity'] == 2, 'ethnicity'] = 0 # Non-hispanic/latine is 2

migraine_data['ethnicity'].head(40)
```

Out[5]:
```
0     0
1     0
2     0
3     0
4     0
5     0
6     0
7     0
8     0
9     0
10    0
11    0
12    0
13    0
14    0
15    0
16    0
17    0
18    0
19    0
20    0
21    0
22    0
23    0
24    0
25    0
26    0
27    0
28    0
29    0
30    0
31    0
32    0
33    0
34    0
35    1
36    0
37    0
38    0
39    0
Name: ethnicity, dtype: int64
```

In [6]:
```python
# Check race other text, and see if any of the races they described fit what we already have
race_other = migraine_data.loc[lambda df: df['race_6_TEXT'].notnull()]  # If race_text isn't empty
race_other = race_other[['userID', 'ethnicity', 'race_1', 'race_2', 'race_3', 'race_4', 'race_5', 'race_6_TEXT']]

race_other.head()
```

Out[6]:

	userID	ethnicity	race_1	race_2	race_3	race_4	race_5	race_6_TEXT
12	12	0	1.0	NaN	NaN	NaN	1.0	German, Dutch
35	35	1	NaN	NaN	NaN	NaN	NaN	hispanic indian irish and german
82	82	0	NaN	NaN	NaN	NaN	NaN	Pakistani
108	108	1	NaN	NaN	NaN	NaN	NaN	Hispanic
148	148	0	NaN	NaN	NaN	NaN	NaN	South Asian (Indian)

In [7]:
```python
# Set race_2 "asian" to 1 for user 82 and 148
migraine_data.loc[(migraine_data['userID'] == 82) | (migraine_data['userID'] == 148), 'race_2'] = 1

# Set user 35's race_1 & 5 to 1
migraine_data.loc[(migraine_data['userID'] == 35), ['race_1', 'race_5']] = 1

# Confirm we fixed them
check = migraine_data.loc[(migraine_data['userID'] == 82) | (migraine_data['userID'] == 148) | (migraine_data['userID'] == 35)]
check[['userID', 'ethnicity', 'race_1', 'race_2', 'race_3', 'race_4', 'race_5', 'race_6_TEXT']].head()
```

Out[7]:

	userID	ethnicity	race_1	race_2	race_3	race_4	race_5	race_6_TEXT
35	35	1	1.0	NaN	NaN	NaN	1.0	hispanic indian irish and german
82	82	0	NaN	1.0	NaN	NaN	NaN	Pakistani
148	148	0	NaN	1.0	NaN	NaN	NaN	South Asian (Indian)

2.2 Replace all Null Values

In [8]:
```python
# Replace all NA's with zero (replace returns a new df)
migraine_data = migraine_data.replace(np.nan, 0)

migraine_data.head(10)
```

Out[8]:

	age	diagnosis_1	diagnosis_2	diagnosis_3	diagnosis_duration	mg_6_months	cm_3_years	id_migraine_1	id_migraine_2	id_migraine_3	...	Q84_5	Q84_6	Q84_7	Q84_8	Q84_9	Q8
0	27	1.0	1.0	0.0	5	4	1	2	1	1	...	3.0	3.0	3.0	3.0	3.0	
1	39	0.0	1.0	0.0	5	5	2	1	1	1	...	1.0	1.0	1.0	1.0	1.0	
2	33	0.0	1.0	0.0	5	30	2	1	1	2	...	1.0	1.0	1.0	1.0	1.0	
3	36	0.0	1.0	0.0	5	5	2	1	1	1	...	1.0	1.0	1.0	1.0	1.0	
4	32	1.0	0.0	0.0	5	14	1	1	1	2	...	2.0	1.0	1.0	1.0	1.0	
5	36	0.0	1.0	0.0	5	10	1	1	1	1	...	2.0	1.0	1.0	1.0	1.0	
6	61	1.0	0.0	0.0	5	30	1	1	1	1	...	1.0	1.0	1.0	1.0	1.0	
7	35	0.0	1.0	0.0	5	30	2	1	1	1	...	1.0	1.0	1.0	1.0	1.0	
8	45	1.0	0.0	0.0	5	24	1	1	1	1	...	2.0	1.0	1.0	1.0	1.0	
9	36	1.0	0.0	0.0	5	30	1	1	1	1	...	1.0	1.0	1.0	1.0	1.0	

10 rows × 233 columns

2.3 Separate Single and Multi-Race

In [9]:

```
# Check For Multiracial Surveyors
race_multi = migraine_data.loc[lambda df: (df['race_1'] + df['race_2'] + df['race_3'] + df['race_4'] + df['race_5']) > 1]

# Check if we have multirace
race_multi = race_multi[['userID', 'age', 'diagnosis_age', 'sex', 'ethnicity', 'race_1', 'race_2', 'race_3', 'race_4',
                                  'race_5', 'race_6', 'race_6_TEXT', 'diagnosis_1', 'diagnosis_2', 'diagnosis_3',
                                  'diagnosis_duration', 'mg_6_months', 'cm_3_years', 'id_migraine_1', 'id_migraine_2',
                                  'id_migraine_3', 'substance_use', 'psych_cond', 'other_diagnoses', 'other_diagnoses_3_TEXT',
                                  'ha_days_month', 'mg_days_month', 'red_fun_days_month']]

race_multi.head(20)
```

Out[9]:

	userID	age	diagnosis_age	sex	ethnicity	race_1	race_2	race_3	race_4	race_5	...	id_migraine_1	id_migraine_2	id_migraine_3	substance_use	psych_cond	other_diagnoses	o
5	5	36	28	1	0	1.0	0.0	0.0	0.0	1.0	...	1	1	1	2	2	3	
6	6	61	17	2	0	1.0	0.0	0.0	0.0	1.0	...	1	1	1	2	2	1	
12	12	52	35	2	0	1.0	0.0	0.0	0.0	1.0	...	1	1	1	2	2	1	
20	20	55	15	2	0	1.0	0.0	0.0	0.0	1.0	...	1	1	1	2	2	1	
35	35	55	30	2	1	1.0	0.0	0.0	0.0	1.0	...	1	1	1	2	2	1	
56	56	23	14	2	0	1.0	0.0	0.0	0.0	1.0	...	1	1	1	2	2	1	
84	84	54	38	2	0	0.0	1.0	0.0	0.0	1.0	...	1	1	1	2	2	1	
93	93	40	5	2	1	1.0	0.0	0.0	0.0	1.0	...	1	1	1	2	2	3	
121	121	29	24	2	0	0.0	0.0	1.0	0.0	1.0	...	1	1	1	2	2	1	
174	174	29	17	2	0	0.0	0.0	0.0	1.0	1.0	...	1	1	1	2	2	1	
199	199	49	20	2	0	1.0	0.0	0.0	0.0	1.0	...	1	1	1	2	2	1	

11 rows × 28 columns

In [10]:

```
# Separate Multiracial Surveyors from main df to allow easier and fairer demographic paneling
race_single = migraine_data.loc[lambda df: (df['race_1'] + df['race_2'] + df['race_3'] + df['race_4'] + df['race_5']) <= 1]
race_single = race_single[['userID', 'age', 'diagnosis_age', 'sex', 'ethnicity', 'race_1', 'race_2', 'race_3', 'race_4',
                                  'race_5', 'race_6', 'race_6_TEXT', 'diagnosis_1', 'diagnosis_2', 'diagnosis_3',
                                  'diagnosis_duration', 'mg_6_months', 'cm_3_years', 'id_migraine_1', 'id_migraine_2',
                                  'id_migraine_3', 'substance_use', 'psych_cond', 'other_diagnoses', 'other_diagnoses_3_TEXT',
                                  'ha_days_month', 'mg_days_month', 'red_fun_days_month']]

# Check if our Single Race DF has no Multirace
race_single.loc[lambda df: df['race_1'] + df['race_2'] + df['race_3'] + df['race_4'] + df['race_5'] <= 1].head(20)
```

Out[10]:

	userID	age	diagnosis_age	sex	ethnicity	race_1	race_2	race_3	race_4	race_5	...	id_migraine_1	id_migraine_2	id_migraine_3	substance_use	psych_cond	other_diagnoses	oth
0	0	27	15	2	0	0.0	0.0	0.0	0.0	1.0	...	2	1	1	2	2	1	
1	1	39	29	2	0	0.0	0.0	0.0	0.0	1.0	...	1	1	1	2	2	1	
2	2	33	20	2	0	0.0	0.0	0.0	0.0	1.0	...	1	1	2	2	2	3	
3	3	36	16	2	0	0.0	0.0	0.0	0.0	1.0	...	1	1	1	2	2	1	
4	4	32	12	2	0	0.0	0.0	0.0	0.0	1.0	...	1	1	2	2	2	1	
7	7	35	32	1	0	0.0	0.0	0.0	0.0	1.0	...	1	1	1	2	2	1	
8	8	45	40	2	0	0.0	0.0	0.0	0.0	1.0	...	1	1	1	2	2	1	
9	9	36	17	2	0	0.0	0.0	0.0	0.0	1.0	...	1	1	1	2	2	3	
10	10	57	54	1	0	0.0	0.0	0.0	0.0	1.0	...	1	1	1	2	2	1	
11	11	32	16	1	0	0.0	0.0	0.0	0.0	1.0	...	2	1	1	2	2	3	
13	13	60	50	1	0	0.0	0.0	0.0	0.0	1.0	...	1	1	1	2	2	3	
14	14	45	35	2	0	0.0	0.0	0.0	0.0	1.0	...	1	1	1	2	2	1	
15	15	38	16	1	0	0.0	0.0	0.0	0.0	1.0	...	1	1	1	2	2	1	
16	16	67	30	1	0	0.0	0.0	0.0	0.0	1.0	...	2	1	1	2	2	1	
17	17	34	26	1	0	0.0	0.0	0.0	0.0	1.0	...	1	1	1	2	2	1	
18	18	27	12	2	0	0.0	0.0	0.0	0.0	1.0	...	1	1	1	2	2	1	
19	19	44	10	2	0	0.0	0.0	0.0	0.0	1.0	...	1	1	1	2	2	3	
21	21	45	35	2	0	0.0	0.0	0.0	0.0	1.0	...	1	1	1	2	2	1	
22	22	34	15	2	0	0.0	0.0	0.0	0.0	1.0	...	1	1	1	2	2	1	
23	23	40	16	2	0	0.0	0.0	0.0	0.0	1.0	...	1	1	1	2	2	1	

20 rows × 28 columns

2.4 Separate Data into Parts for Easier Management

In [11]:

```python
# Separate Demographic for ALL race types (single/multiracial)
demographic_data = migraine_data[['userID', 'age', 'diagnosis_age', 'sex', 'ethnicity', 'race_1', 'race_2', 'race_3', 'race_4',
                                  'race_5', 'race_6', 'race_6_TEXT', 'diagnosis_1', 'diagnosis_2', 'diagnosis_3',
                                  'diagnosis_duration', 'mg_6_months', 'cm_3_years', 'id_migraine_1', 'id_migraine_2',
                                  'id_migraine_3', 'substance_use', 'psych_cond', 'other_diagnoses', 'other_diagnoses_3_TEXT',
                                  'ha_days_month', 'mg_days_month', 'red_fun_days_month']]

demographic_data.head(10)
```

Out[11]:

	userID	age	diagnosis_age	sex	ethnicity	race_1	race_2	race_3	race_4	race_5	...	id_migraine_1	id_migraine_2	id_migraine_3	substance_use	psych_cond	other_diagnoses	othe
0	0	27	15	2	0	0.0	0.0	0.0	0.0	1.0	...	2	1	1	2	2	1	
1	1	39	29	2	0	0.0	0.0	0.0	0.0	1.0	...	1	1	1	2	2	1	
2	2	33	20	2	0	0.0	0.0	0.0	0.0	1.0	...	1	1	2	2	2	3	
3	3	36	16	2	0	0.0	0.0	0.0	0.0	1.0	...	1	1	1	2	2	1	
4	4	32	12	2	0	0.0	0.0	0.0	0.0	1.0	...	1	1	2	2	2	1	
5	5	36	28	1	0	1.0	0.0	0.0	0.0	1.0	...	1	1	1	2	2	3	
6	6	61	17	2	0	1.0	0.0	0.0	0.0	1.0	...	1	1	1	2	2	1	
7	7	35	32	1	0	0.0	0.0	0.0	0.0	1.0	...	1	1	1	2	2	1	
8	8	45	40	2	0	0.0	0.0	0.0	0.0	1.0	...	1	1	1	2	2	1	
9	9	36	17	2	0	0.0	0.0	0.0	0.0	1.0	...	1	1	1	2	2	3	

10 rows × 28 columns

In [12]:

```python
# Create a function to allow us to re-sort the sectioned data with userID in the front instead of the end
def idInFront(df):
    temp = df.columns.tolist()
    temp = temp[-1:] + temp[:-1]

    return temp
```

In [13]:

```python
# Separate Triptans from data set
# triptan_data = migraine_data['ResponseId', 'triptan_1', 'triptan_2'... etc]
triptan_data = migraine_data.loc[:, 'triptan_1':'triptan_12']  # This does the above
triptan_data['userID'] = migraine_data['userID']

# Sort USERID in front of df
triptan_data = triptan_data[idInFront(triptan_data)]
triptan_data.head(20)
```

Out[13]:

	userID	triptan_1	triptan_2	triptan_3	triptan_4	triptan_5	triptan_6	triptan_7	triptan_8	triptan_9	triptan_10	triptan_11	triptan_12
0	0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0
1	1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0
2	2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0
3	3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0
4	4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0
5	5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0
6	6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0
7	7	1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0
8	8	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
9	9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0
10	10	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0
11	11	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0
12	12	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0
13	13	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0
14	14	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
15	15	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0
16	16	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
17	17	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0
18	18	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0
19	19	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

In [14]:
```python
# Separate Ergots
ergot_data = migraine_data.loc[:, 'ergot_78':'ergot_83']
ergot_data['userID'] = migraine_data['userID']

# Set userid to front
ergot_data = ergot_data[idInFront(ergot_data)]
ergot_data.head()
```

Out[14]:

	userID	ergot_78	ergot_79	ergot_84	ergot_80	ergot_81	ergot_82	ergot_83
0	0	0.0	0.0	0.0	0.0	0.0	0.0	1.0
1	1	0.0	0.0	0.0	0.0	0.0	0.0	1.0
2	2	0.0	0.0	0.0	0.0	0.0	0.0	1.0
3	3	0.0	0.0	0.0	0.0	0.0	0.0	1.0
4	4	0.0	0.0	0.0	0.0	0.0	0.0	1.0

In [15]:
```python
# Separate Combination Analgesics
analgesics_data = migraine_data.loc[:, 'comb_analgesic_1':'comb_analgesic_4']
analgesics_data['userID'] = migraine_data['userID']

analgesics_data = analgesics_data[idInFront(analgesics_data)]
analgesics_data.head()
```

Out[15]:

	userID	comb_analgesic_1	comb_analgesic_2	comb_analgesic_3	comb_analgesic_4
0	0	0.0	1.0	0.0	0.0
1	1	0.0	0.0	0.0	1.0
2	2	0.0	0.0	0.0	1.0
3	3	0.0	0.0	0.0	1.0
4	4	0.0	0.0	0.0	1.0

In [16]:
```python
# Separate NSAID's
nsaid_data = migraine_data.loc[:, 'nsaid_1':'nsaid_7']
nsaid_data['userID'] = migraine_data['userID']

nsaid_data = nsaid_data[idInFront(nsaid_data)]
nsaid_data.head()
```

Out[16]:

	userID	nsaid_1	nsaid_2	nsaid_3	nsaid_4	nsaid_5	nsaid_6	nsaid_7
0	0	0.0	0.0	0.0	0.0	0.0	0.0	1.0
1	1	1.0	0.0	0.0	0.0	0.0	0.0	0.0
2	2	0.0	1.0	0.0	0.0	0.0	0.0	0.0
3	3	0.0	0.0	0.0	0.0	0.0	0.0	1.0
4	4	0.0	0.0	0.0	0.0	0.0	0.0	1.0

In [17]:
```python
# Separate Headache Alternative Medication
ha_data = migraine_data.loc[:, 'ha_meds_other_1':'ha_meds_other_5']
ha_data['userID'] = migraine_data['userID']
```

```
ha_data = ha_data[idInFront(ha_data)]
ha_data.head()
```

Out[17]:

	userID	ha_meds_other_1	ha_meds_other_2	ha_meds_other_3	ha_meds_other_4	ha_meds_other_5
0	0	1.0	0.0	0.0	1.0	0.0
1	1	0.0	0.0	0.0	1.0	0.0
2	2	1.0	0.0	0.0	1.0	0.0
3	3	0.0	0.0	0.0	1.0	0.0
4	4	1.0	0.0	0.0	0.0	0.0

In [18]:

```
# Migraine Devices
devices_data = migraine_data.loc[:, 'devices_1':'devices_4']
devices_data['userID'] = migraine_data['userID']

devices_data = devices_data[idInFront(devices_data)]
devices_data.head()
```

Out[18]:

	userID	devices_1	devices_2	devices_3	devices_4
0	0	0.0	0.0	0.0	1.0
1	1	0.0	0.0	0.0	1.0
2	2	0.0	0.0	0.0	1.0
3	3	0.0	0.0	0.0	1.0
4	4	0.0	1.0	0.0	0.0

In [19]:

```
# Anti-convulsants
antiCon_data = migraine_data.loc[:, 'Anticonvulsant_1':'Anticonvulsant_5']
antiCon_data['userID'] = migraine_data['userID']

antiCon_data = antiCon_data[idInFront(antiCon_data)]
antiCon_data.head()
```

Out[19]:

	userID	Anticonvulsant_1	Anticonvulsant_2	Anticonvulsant_3	Anticonvulsant_6	Anticonvulsant_4	Anticonvulsant_5
0	0	0.0	0.0	0.0	0.0	0.0	1.0
1	1	0.0	0.0	0.0	0.0	0.0	1.0
2	2	0.0	0.0	0.0	0.0	0.0	1.0
3	3	0.0	0.0	0.0	0.0	0.0	1.0
4	4	0.0	0.0	0.0	0.0	0.0	1.0

In [20]:

```
# Beta blockers
betaBlock_data = migraine_data.loc[:, 'beta_blocker_1':'beta_blocker_6']
betaBlock_data['userID'] = migraine_data['userID']

betaBlock_data = betaBlock_data[idInFront(betaBlock_data)]
betaBlock_data.head()
```

Out[20]:

	userID	beta_blocker_1	beta_blocker_2	beta_blocker_3	beta_blocker_4	beta_blocker_5	beta_blocker_6
0	0	0.0	0.0	0.0	0.0	0.0	1.0
1	1	0.0	0.0	0.0	0.0	0.0	1.0
2	2	0.0	0.0	0.0	0.0	0.0	1.0
3	3	0.0	0.0	0.0	0.0	0.0	1.0
4	4	0.0	0.0	0.0	0.0	0.0	1.0

In [21]:

```
# Calcium Channel Blockers
ccb_data = migraine_data.loc[:, 'ccb_1':'ccb_4']
ccb_data['userID'] = migraine_data['userID']

ccb_data = ccb_data[idInFront(ccb_data)]
ccb_data.head()
```

Out[21]:

	userID	ccb_1	ccb_2	ccb_3	ccb_4
0	0	0.0	0.0	0.0	1.0
1	1	0.0	0.0	0.0	1.0
2	2	0.0	0.0	0.0	1.0
3	3	0.0	0.0	0.0	1.0
4	4	0.0	0.0	0.0	1.0

In [22]:

```
# Selective Serotonin Reuptake Inhibitor
ssri_data = migraine_data.loc[:, 'ssri_1':'ssri_4']
ssri_data['userID'] = migraine_data['userID']

ssri_data = ssri_data[idInFront(ssri_data)]
ssri_data.head()
```

Out[22]:

	userID	ssri_1	ssri_2	ssri_3	ssri_4
0	0	0.0	0.0	1.0	0.0
1	1	0.0	0.0	0.0	1.0
2	2	0.0	0.0	0.0	1.0
3	3	0.0	0.0	1.0	0.0
4	4	0.0	0.0	0.0	1.0

In [23]:

```python
# Tricyclic Antidepressants
tca_data = migraine_data.loc[:, 'tca_1':'tca_7']
tca_data['userID'] = migraine_data['userID']

tca_data = tca_data[idInFront(tca_data)]
tca_data.head()
```

Out[23]:

	userID	tca_1	tca_2	tca_3	tca_4	tca_5	tca_6	tca_7
0	0	0.0	0.0	0.0	0.0	0.0	0.0	1.0
1	1	0.0	0.0	0.0	0.0	0.0	0.0	1.0
2	2	0.0	0.0	0.0	0.0	0.0	0.0	1.0
3	3	0.0	0.0	0.0	0.0	0.0	0.0	1.0
4	4	0.0	0.0	0.0	0.0	0.0	0.0	1.0

In [24]:

```python
# Supplements
supp_data = migraine_data.loc[:, 'supplement_16':'supplement_21']
supp_data['userID'] = migraine_data['userID']

supp_data = supp_data[idInFront(supp_data)]
supp_data.head()
```

Out[24]:

	userID	supplement_16	supplement_17	supplement_18	supplement_19	supplement_20	supplement_21
0	0	1.0	0.0	0.0	1.0	0.0	0.0
1	1	0.0	0.0	0.0	0.0	0.0	1.0
2	2	0.0	0.0	0.0	0.0	0.0	1.0
3	3	0.0	0.0	0.0	0.0	0.0	1.0
4	4	0.0	0.0	0.0	0.0	0.0	1.0

In [25]:

```python
# CGRP Antibody Injections
cgrp_data = migraine_data.loc[:, 'cgrp_antibody_1':'cgrp_antibody_3']
cgrp_data['userID'] = migraine_data['userID']

cgrp_data = cgrp_data[idInFront(cgrp_data)]
cgrp_data.head()
```

Out[25]:

	userID	cgrp_antibody_1	cgrp_antibody_2	cgrp_antibody_4	cgrp_antibody_3
0	0	0.0	0.0	0.0	1.0
1	1	0.0	0.0	0.0	1.0
2	2	0.0	0.0	0.0	1.0
3	3	0.0	0.0	0.0	1.0
4	4	0.0	0.0	0.0	1.0

In [26]:

```python
# Other Medication
otherMed_data = migraine_data.loc[:, 'meds_other_1':'meds_other_5']
otherMed_data['userID'] = migraine_data['userID']

otherMed_data = otherMed_data[idInFront(otherMed_data)]
otherMed_data.head()
```

Out[26]:

	userID	meds_other_1	meds_other_2	meds_other_3	meds_other_4	meds_other_5
0	0	0.0	0.0	0.0	0.0	1.0
1	1	0.0	0.0	0.0	0.0	1.0
2	2	0.0	0.0	0.0	0.0	1.0
3	3	0.0	0.0	0.0	0.0	1.0
4	4	0.0	0.0	0.0	0.0	1.0

In [27]:

```python
# Qualtrics Survey Portion - Question block
qb_data = migraine_data.loc[:, 'qb1#1_1':'qb15#1_4']
qb_data['userID'] = migraine_data['userID']

qb_data = qb_data[idInFront(qb_data)]
qb_data.head()
```

Out[27]:

	userID	qb1#1_1	qb1#1_2	qb1#1_3	qb1#1_4	qb2#1_1	qb2#1_2	qb2#1_3	qb2#1_4	qb2#1_5	...	qb13#1_4	qb13#1_5	qb14#1_1	qb14#1_2	qb14#1_3	qb14#1_4	qb15#1_1	qb
0	0	3	3	3	3.0	3	3	3	3	3	...	3	3	3.0	3.0	3.0	3.0	3	
1	1	2	4	4	5.0	3	4	3	3	2	...	3	2	3.0	3.0	3.0	4.0	4	
2	2	4	3	4	1.0	2	4	2	2	4	...	2	2	3.0	3.0	2.0	4.0	4	
3	3	2	1	4	3.0	2	4	2	3	1	...	3	2	3.0	1.0	2.0	3.0	1	
4	4	5	5	3	4.0	4	3	3	4	5	...	1	4	5.0	5.0	5.0	5.0	5	

5 rows × 73 columns

3. Analyze Data

In [28]:

```python
# Create a helper function to find pairs for multiracial users
"""
Dataframe has to contain race_1 through race_5 as its columns
"""
def multiracePairs(df):
    # Create a dictionary to save a count of our race pairs
    pairs = {}

    # Iterate through the rows of our dataframe
    for i, row in df.iterrows():
        if row.sum() > 1:  # If its multiracial
            for j in range(1, 6): # Iterate through race 1-5 as first element in pair
                for k in range(j + 1, 6):  # Iterate through as second element in pair (e.g, 1, 2)
                    raceA = f"race_{j}"
                    raceB = f"race_{k}"
                    if row[raceA] == 1 and row[raceB] == 1:
                        key = f"{raceA}:{raceB}"
                        # If we already have the pair in the dictionary, add it by 1
                        if key in pairs:
                            pairs[key] += 1
                        else:
                            pairs[key] = 1

    return pairs
```

In [29]:

```python
# Find the demographic proportions (fix multiracial issue, percentages don't line up)
# summaryAll = demographic_data.loc[:, 'ethnicity':'race_5'].describe()  # Doesn't work, multiracial problem
summarySingle = race_single.loc[:, 'ethnicity':'race_5'].describe()
summaryMulti = race_multi.loc[:, 'ethnicity':'race_5'].describe()

# Save only the average of single race demographics for graphs
raceSingleAvg = summarySingle.loc[['mean'], 'race_1':'race_5']
summaryMulti
```

Out[29]:

	ethnicity	race_1	race_2	race_3	race_4	race_5
count	11.000000	11.000000	11.000000	11.000000	11.000000	11.0
mean	0.181818	0.727273	0.090909	0.090909	0.090909	1.0
std	0.404520	0.467099	0.301511	0.301511	0.301511	0.0
min	0.000000	0.000000	0.000000	0.000000	0.000000	1.0
25%	0.000000	0.500000	0.000000	0.000000	0.000000	1.0
50%	0.000000	1.000000	0.000000	0.000000	0.000000	1.0
75%	0.000000	1.000000	0.000000	0.000000	0.000000	1.0
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.0

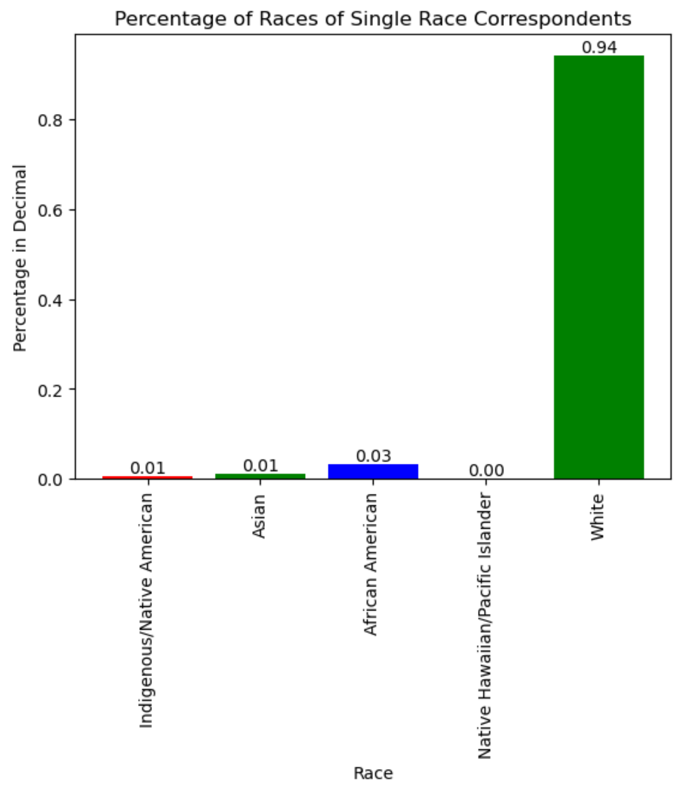
3.1 Graph Single Race Correspondents

In [30]:

```python
# Define a function to call to create data labels
def barGraphLabels(x, y):
    for x, y in zip(x, y):
        label = "{:.2f}".format(y)

        plt.annotate(label,
                     (x, y),
                     textcoords = "offset points",
                     xytext = (0, 2),
                     ha = 'center')
```

In [31]:

```python
# Save the axis for the graph
# raceNames = list(raceSingleAvg.columns)
raceNames = ['Indigenous/Native American', 'Asian', 'African American', 'Native Hawaiian/Pacific Islander', 'White']
raceAvg = list(raceSingleAvg.loc['mean'])

# Choose Which Graph to Use
# plt.pie(raceAvg, labels = raceNames)
plt.bar(raceNames, raceAvg, color = ['r', 'g', 'b', 'r', 'g'])
plt.xticks(raceNames, rotation = 'vertical', size = 10)
plt.ylabel('Percentage in Decimal')
plt.xlabel('Race')
plt.title('Percentage of Races of Single Race Correspondents')

# Add data labels to graph
barGraphLabels(raceNames, raceAvg)
plt.show()
```



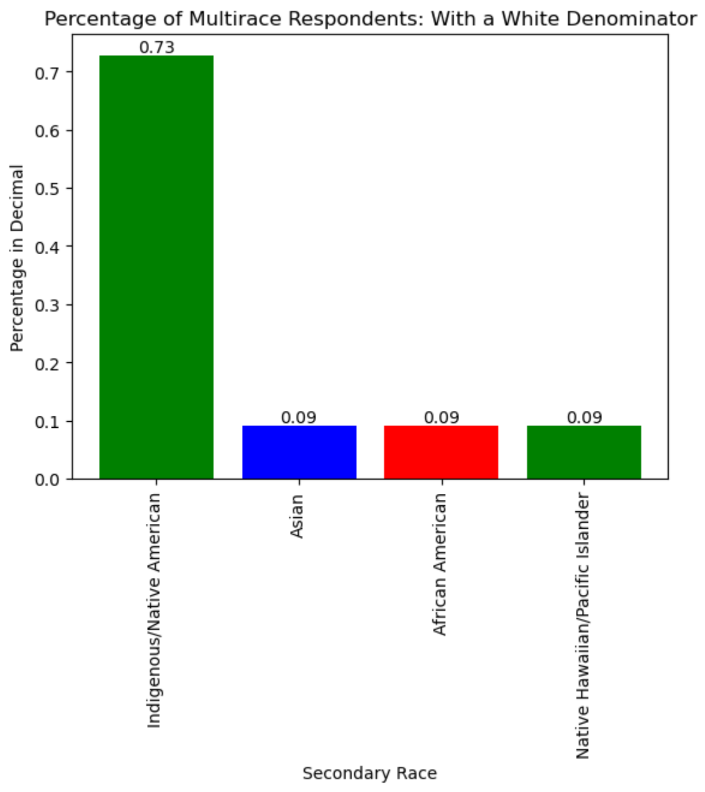
3.2 Graph Multiracial Correspondents

In [32]:
```python
# Calculate Multiracial Correspondents
racePairs = multiracePairs(race_multi.loc[:, 'race_1':'race_5'])  # Create a dictionary of our race pairs
multiRacetotal = 0
multiRaceCombo = []  # Save the Race pairs that are present

# Add how many racial pairs we have
for key in racePairs:
    multiRacetotal += racePairs[key]
    multiRaceCombo.append(key)
print(f"We have {multiRacetotal} multiracial pairs")
print(f"With these {multiRaceCombo} kinds of pairs")
```

```
We have 11 multiracial pairs
With these ['race_1:race_5', 'race_2:race_5', 'race_3:race_5', 'race_4:race_5'] kinds of pairs
```

In [33]:
```python
# Get the percentage of the racepairs
racePairsPercent = racePairs
for key in racePairsPercent:
    racePairsPercent[key] = racePairsPercent[key] / multiRacetotal

# racePairsPercent
```

In [34]:
```python
# Graph Multirace
plt.bar(raceNames[:-1], racePairsPercent.values(), color = ['g', 'b', 'r', 'g'])
plt.title("Percentage of Multirace Respondents: With a White Denominator")
plt.xticks(raceNames[:-1], rotation = 'vertical', size = 10)
plt.ylabel('Percentage in Decimal')
plt.xlabel('Secondary Race')
barGraphLabels(raceNames[:-1], racePairsPercent.values())
plt.show()
```



3.3 Perform Chi Square Test on our sample's racial demographic for single race

In [35]:
```
# Positional Key from our population statistic (2018 census)
# [Hispanic, Indigenous, Asian, African American, Native Hawaiian, White]
censusExpectedRace = [.1871, .0076, .0585, .1278, .0018, .6172]

observedRace = []
for race in summarySingle.iloc[1]:
    observedRace.append(race)

observedRace[5] -= 0.010582010582010547
observedRace
```

Out[35]:
```
[0.021164021164021163,
 0.005291005291005291,
 0.010582010582010581,
 0.031746031746031744,
 0.0,
 0.9312169312169313]
```

Looking at Single Race Only, Our Sample is Representative to the Population!

In [36]:
```
# Chi Square Test
print(f"For Single Race ONLY")
print(f"Null Hypothesis: The observed sample is representative of the population")
print(f"Alternate Hypothesis: The observed sample has a significant difference from the population\n")

# Save Chi Square statistic and p-value
chi_square_single_race, p_val_single = stats.chisquare(observedRace, censusExpectedRace)

# Get critical value based on distribution chart
critical_val_single = stats.chi2.ppf(1-0.05, df = len(observedRace) - 1)
print(f"Chi Square Goodness of Fit test of single race demographic is: {str(chi_square_single_race)}\n" +
     f"P value is: {str(p_val_single)}\n")

if chi_square_single_race > critical_val_single:  # Reject null
    print(f"Critical Value is {critical_val_single}, reject null hypothesis!")
elif chi_square_single_race <= critical_val_single:  # Fail to reject null
    print(f"Critical Value is {critical_val_single}, failed to reject null hypothesis!")
```

```
For Single Race ONLY
Null Hypothesis: The observed sample is representative of the population
Alternate Hypothesis: The observed sample has a significant difference from the population

Chi Square Goodness of Fit test of single race demographic is: 0.4208758568952532
P value is: 0.9947350449232272

Critical Value is 11.070497693516351, failed to reject null hypothesis!
```

3.3.1 Perform Chi Square Test on our sample's racial demographic including multirace

Including multirace, Our Sample is Representative to the Population!

In [37]:
```
# Use this to calculate number of people per race for single race, total race is 200! Multirace is 5.5%
summarySingle
```

Out[37]:

	ethnicity	race_1	race_2	race_3	race_4	race_5
count	189.000000	189.000000	189.000000	189.000000	189.0	189.000000
mean	0.021164	0.005291	0.010582	0.031746	0.0	0.941799
std	0.144313	0.072739	0.102595	0.175789	0.0	0.234745
min	0.000000	0.000000	0.000000	0.000000	0.0	0.000000
25%	0.000000	0.000000	0.000000	0.000000	0.0	1.000000
50%	0.000000	0.000000	0.000000	0.000000	0.0	1.000000
75%	0.000000	0.000000	0.000000	0.000000	0.0	1.000000
max	1.000000	1.000000	1.000000	1.000000	0.0	1.000000

In [38]:
```
# Check number of correspondents in single and multi df
a = len(race_single)
b = len(race_multi)
print(f"{a} & {b}")
```
```
189 & 11
```

In [39]:
```
# Create our observed and expected arrays for chi square testing

# [Hispanic, Indigenous, Asian, African American, Native Hawaiian, White, Multirace]
censusExpectedMultiRace = [.1830, .0074, .0572, .1250, .0018, .6039, .0217 ]

# Observed Race percentages WITH multirace percentage
observedRaceMulti = [(4/200), (1/200), (2/200), (6/200), (0/200), (176/200), (11/200)]
```

In [40]:
```
# Chi Square Test
print(f"Including MULTI-RACE")
print(f"Null Hypothesis: The observed sample is representative of the population")
print(f"Alternate Hypothesis: The observed sample has a significant difference from the population\n")

# Save Chi Square statistic and p-value
chi_square_multi_race, p_val_multi = stats.chisquare(observedRaceMulti, censusExpectedMultiRace)

# Get critical value based on distribution chart
critical_val_multi = stats.chi2.ppf(1-0.05, df = len(observedRaceMulti) - 1)
print(f"Chi Square Goodness of Fit test of multi race demographic is: {str(chi_square_multi_race)}\n" +
     f"P value is: {str(p_val_multi)}\n")

if chi_square_multi_race > critical_val_multi:  # Reject null
    print(f"Critical Value is {critical_val_multi}, reject null hypothesis!")
elif chi_square_single_race <= critical_val_multi:  # Fail to reject null
    print(f"Critical Value is {critical_val_multi}, failed to reject null hypothesis!")
```
```
Including MULTI-RACE
Null Hypothesis: The observed sample is representative of the population
Alternate Hypothesis: The observed sample has a significant difference from the population

Chi Square Goodness of Fit test of multi race demographic is: 0.4362448559750517
P value is: 0.9985300762155238

Critical Value is 12.591587243743977, failed to reject null hypothesis!
```

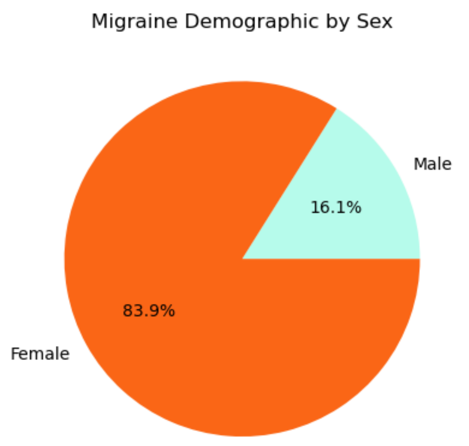
Check Sex Demographic Makeup

In [41]:
```
male = migraine_data.loc[lambda df: df['sex'] == 1]
female = migraine_data.loc[lambda df: df['sex'] == 2]
```

In [42]:
```
totalMale = len(male)
totalFemale = len(female)
totalSex = totalMale + totalFemale

print(f"Number of male correspondents: {totalMale}, or {((totalMale / totalSex) * 100):.2f} % of the sample")
print(f"Number of female correspondents: {totalFemale}, or {((totalFemale / totalSex) * 100):.2f} % of the sample")
```
```
Number of male correspondents: 32, or 16.08 % of the sample
Number of female correspondents: 167, or 83.92 % of the sample
```

In [43]:
```
# Graph Sex Demographic with Migraine
plt.pie([(totalMale / totalSex), (totalFemale / totalSex)], labels = ['Male', 'Female'], colors = ['#b6fbeb', '#fa6616'],
        autopct='%1.1f%%')
plt.title("Migraine Demographic by Sex")
```

Out[43]:
```
Text(0.5, 1.0, 'Migraine Demographic by Sex')
```



3.4 Explore Medications

In [44]:
```
# Create a function to calculate the percentage of users who use a medication

def percentageUsed(array, populationSize):
    return sum(array[1:-1]) / populationSize
```

In [45]:
```
# Create a function to count how many use a specific med

def medUsed(df):
    medCount = []
    for i in range(1, len(df.columns) - 1):  # Removed first column since it holds an Id
        count = 0
        for x, med in df.iterrows():
            count += med
        medCount.append(count)
    return medCount[1]
```

Triptans

In [46]:
```
# Visually see triptan table
tripStat = triptan_data.describe()
tripStat
```

Out[46]:

	userID	triptan_1	triptan_2	triptan_3	triptan_4	triptan_5	triptan_6	triptan_7	triptan_8	triptan_9	triptan_10	triptan_11	triptan_12
count	200.000000	200.000000	200.000000	200.000000	200.000000	200.000000	200.000000	200.000000	200.000000	200.000000	200.000000	200.000000	200.000000
mean	99.500000	0.075000	0.015000	0.020000	0.345000	0.195000	0.010000	0.055000	0.020000	0.045000	0.040000	0.080000	0.335000
std	57.879185	0.264052	0.121857	0.140351	0.476561	0.397195	0.099748	0.228552	0.140351	0.207824	0.196451	0.271974	0.473175
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	49.750000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
50%	99.500000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
75%	149.250000	0.000000	0.000000	0.000000	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	1.000000
max	199.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000

In [47]:
```
# Count how many users use triptans
tripCount = [None]  # Place empty val in 0th position since dataframe has Id in 0th position
for x in range(1, 13):
#     col = f"triptan_{x}"
    count = 0
    for i, triptan in triptan_data.iterrows():
        count += triptan
    tripCount.append(count)
```

In [48]:
```
# Save total of triptans used by participants
tripCount = tripCount[1]
```

In [49]:
```
# Calculate percentage of users who use
tripPercent = percentageUsed(tripCount, 200)
tripPercent
```

Out[49]:
```
0.9
```

Ergots

In [50]:
```
ergotStat = ergot_data.describe()
ergotStat
```

Out[50]:

	userID	ergot_78	ergot_79	ergot_84	ergot_80	ergot_81	ergot_82	ergot_83
count	200.000000	200.000000	200.000000	200.0	200.000000	200.0	200.0	200.000000
mean	99.500000	0.015000	0.015000	0.0	0.015000	0.0	0.0	0.955000
std	57.879185	0.121857	0.121857	0.0	0.121857	0.0	0.0	0.207824
min	0.000000	0.000000	0.000000	0.0	0.000000	0.0	0.0	0.000000
25%	49.750000	0.000000	0.000000	0.0	0.000000	0.0	0.0	1.000000
50%	99.500000	0.000000	0.000000	0.0	0.000000	0.0	0.0	1.000000
75%	149.250000	0.000000	0.000000	0.0	0.000000	0.0	0.0	1.000000
max	199.000000	1.000000	1.000000	0.0	1.000000	0.0	0.0	1.000000

In [51]:
```
# Count how many use ergots
ergotCount = [None]  # Place empty val in 0th position since dataframe has Id in 0th position
for x in range(1, 8):
    count = 0
    for i, ergot in ergot_data.iterrows():
        count += ergot
    ergotCount.append(count)
```

In [52]:
```
ergotCount = ergotCount[1]
```

In [53]:
```
# Calculate percentage of users who use an ergot
ergotPercent = percentageUsed(ergotCount, 200)
ergotPercent
```

Out[53]:
```
0.045
```

Analgesics

In [54]:
```
analgesicsStat = analgesics_data.describe()
analgesicsStat
```

Out[54]:

	userID	comb_analgesic_1	comb_analgesic_2	comb_analgesic_3	comb_analgesic_4
count	200.000000	200.000000	200.000000	200.000000	200.00000
mean	99.500000	0.110000	0.110000	0.025000	0.79000
std	57.879185	0.313675	0.313675	0.156517	0.40833
min	0.000000	0.000000	0.000000	0.000000	0.00000
25%	49.750000	0.000000	0.000000	0.000000	1.00000
50%	99.500000	0.000000	0.000000	0.000000	1.00000
75%	149.250000	0.000000	0.000000	0.000000	1.00000
max	199.000000	1.000000	1.000000	1.000000	1.00000

In [55]:
```
# Count how many use
analgesicsCount = medUsed(analgesics_data)
analgesicsCount
```

Out[55]:
```
userID              19900.0
comb_analgesic_1       22.0
comb_analgesic_2       22.0
comb_analgesic_3        5.0
comb_analgesic_4      158.0
Name: 0, dtype: float64
```

In [56]:
```
# Percentage who use
analgesicsPercent = percentageUsed(analgesicsCount, 200)
analgesicsPercent
```

Out[56]:
```
0.245
```

Nsaids

In [57]:
```
# Count how many use
nsaidsCount = medUsed(nsaid_data)
nsaidsCount

# Percentage who use
nsaidsPercent = percentageUsed(nsaidsCount, 200)
nsaidsPercent
```

Out[57]:
```
0.79
```

Alternative Meds

In [58]:
```
# Count how many use
haCount = medUsed(ha_data)
haCount

# Percentage who use
haPercent = percentageUsed(haCount, 200)
haPercent
```

Out[58]:
```
0.615
```

Devices

In [59]:
```
# Count how many use
deviceCount = medUsed(devices_data)
deviceCount

# Percentage who use
devicePercentage = percentageUsed(deviceCount, 200)
devicePercentage
```

Out[59]:
```
0.08
```

Anti-Convulsants

In [60]:
```
# Count how many use
antiCount = medUsed(antiCon_data)
antiCount

# Percentage used
antiPercentage = percentageUsed(antiCount, 200)
antiPercentage
```

Out[60]:
```
0.36
```

Beta Blockers

In [61]:
```
# Count how many use
betaCount = medUsed(betaBlock_data)
betaCount

# Percentage used
betaPercentage = percentageUsed(betaCount, 200)
betaPercentage
```

Out[61]:
```
0.2
```

Calcium Channel Blockers

In [62]:
```
# count how many use
ccbCount = medUsed(ccb_data)
ccbCount

# percentage used
ccbPercentage = percentageUsed(ccbCount, 200)
ccbPercentage
```

Out[62]:
```
0.075
```

Selective Serotonin Reuptake Inhibitors

In [63]:
```
# count how many use
ssriCount = medUsed(ssri_data)
ssriCount

# percentage used
ssriPercentage = percentageUsed(ssriCount, 200)
ssriPercentage
```

Out[63]:
```
0.155
```

Tricyclic Antidepressants

In [64]:
```
# count how many use
tcaCount = medUsed(tca_data)
tcaCount

# percentage used
tcaPercentage = percentageUsed(tcaCount, 200)
tcaPercentage
```

Out[64]:
```
0.085
```

Supplements

In [65]:
```
# count how many use
suppCount = medUsed(supp_data)
suppCount

# percentage used
suppPercentage = percentageUsed(suppCount, 200)
suppPercentage
```

Out[65]:
```
0.835
```

CGRP Anti-Body

In [66]:
```
# count how many use
cgrpCount = medUsed(cgrp_data)
cgrpCount

# percentage used
cgrpPercentage = percentageUsed(cgrpCount, 200)
cgrpPercentage
```

Out[66]:
```
0.29
```

Other Medication

In [67]:
```
# count how many use
otherCount = medUsed(otherMed_data)
otherCount

# percentage used
otherPercentage = percentageUsed(otherCount, 200)
otherPercentage
```

Out[67]:
```
0.25
```

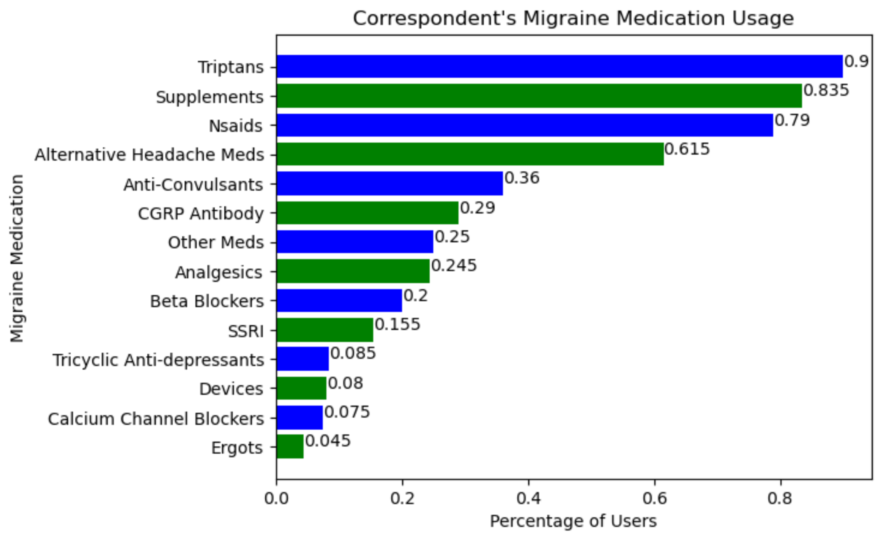
3.4.1 Graph Medication Usage

In [68]:
```
# Create the axes
medNames = {'Triptans':tripPercent, 'Ergots':ergotPercent, 'Analgesics':analgesicsPercent, 'Nsaids':nsaidsPercent,
            'Alternative Headache Meds':haPercent, 'Devices':devicePercentage,
           'Anti-Convulsants':antiPercentage, 'Beta Blockers':betaPercentage, 'Calcium Channel Blockers':ccbPercentage,
            'SSRI':ssriPercentage, 'Tricyclic Anti-depressants':tcaPercentage,
           'Supplements':suppPercentage, 'CGRP Antibody':cgrpPercentage, 'Other Meds':otherPercentage}

# Sort from least to greatest
medNames = dict(sorted(medNames.items(), key = lambda item: item[1], reverse = False))
medNames.values()
```

Out[68]:
```
dict_values([0.045, 0.075, 0.08, 0.085, 0.155, 0.2, 0.245, 0.25, 0.29, 0.36, 0.615, 0.79, 0.835, 0.9])
```

In [69]:
```
# Create the graph
plt.barh(list(medNames.keys()), list(medNames.values()), color = ['g', 'b'])

# Annotate
for y in range(len(list(medNames.keys()))):
    label = list(medNames.values())[y]
    plt.annotate(label, xy = (list(medNames.values())[y], y))

plt.ylabel('Migraine Medication')
plt.xlabel('Percentage of Users')
plt.title('Correspondent\'s Migraine Medication Usage')
```

Out[69]:
```
Text(0.5, 1.0, "Correspondent's Migraine Medication Usage")
```



3.5 Count Other Associated Diagnoses

In [70]:
```
# Grab only correspondents who answered yes to having other diagnoses
otherDiagData = migraine_data.loc[lambda df: df['other_diagnoses'] == 3]
percentageOtherDiag = len(otherDiagData) / len(migraine_data)

# Save the number of patients that have another diagnosis
numOfDiag = len(otherDiagData)
numOfDiag

# Grab only the text diagnoses
otherDiagData = otherDiagData['other_diagnoses_3_TEXT']
diagData = []

# Add all the diagnoses to our list
otherDiagData
for diag in otherDiagData:
    diagData.append(diag)
```

In [71]:
```
# Clean our diagnoses text to make it easier to search for keywords
cleanData = [diag.lower().replace(',', '').replace('.', '') for diag in diagData]
cleanData
```

Out[71]:
```
['menstrual migraines',
 'cte',
 'stress migraines',
 'arnold chiari malformation',
 'diagnosed with thunderclap headaches',
 'hemiplegic',
 'persistent daily headache',
 'no',
 'hemiplegic migraine',
 'hemiplegic',
 "as a child i was diagnosed with cluster headaches i'm not sure if they were right but that's what they said",
 'cervogenic migraines',
 'transformed migraine intractable daily headache pain',
 'trigeminal neuralgia',
 'fibromialgia',
 'tension headache and migraine',
 'medication overuse headache',
 'depression',
 'chronic migraine',
 'migrainous vertigo',
 'medication overuse headaches',
 'cervicogenic10',
 'ndph hashimoto’s diabetes depression',
 'chronic migraine with syncope',
 'cluster headaches',
 'vestibular migraine occipital neuralgia thunderclap headaches',
 'basilar migraine and familial hemiplegic migraine',
 'vestibular',
 'add',
 'chronic migraine without aura with intractable migraine so stated with status migrainosus',
 'unknown headache disorder with other neurological symptoms',
 'vestibular',
 'ocular migraines',
 'combination headache- migraine/tension headache',
 'cluster headaches',
 'vestibular migraine',
 'cluster',
 'visual/optical migraines',
 'hemiplegic',
 'menstrual migraine',
 'i have intracranial hypertension',
 'vestibular silent migraines',
 'chronic cluster headache',
 'with aura',
 'migraine with aura',
 'menstrual related migraine']
```

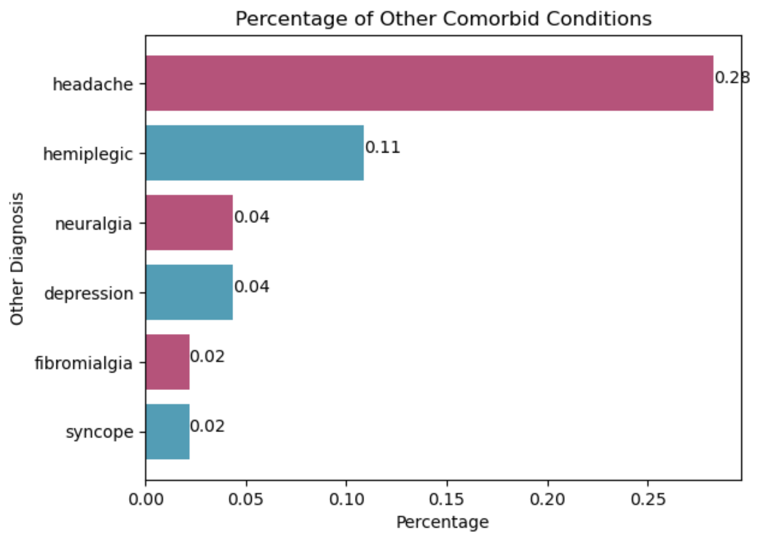
In [72]:
```python
# Create a dictionary of keywords to then search through our clean data
keywords = { 'migraine':0, 'hemiplegic':0, 'depression':0, 'syncope':0, 'headache':0, 'fibromialgia':0, 'neuralgia':0}

# Search through our clean data
for s in cleanData:
#     print(s)
#     print('--------------')
    for key in keywords:
        if re.search(key, s):
#             print(f"Found it! {key}")
            keywords[key] += 1  # If we get a match, add patient count to that diagnosis in the dictionary


# remove migraine from our keywords, save percentage of correspondents with these diagnoses
del(keywords['migraine'])
keywordsPercentage = {}

for key in keywords:
    keywordsPercentage[key] = keywords[key] / numOfDiag

keywordsPercentage
```

Out[72]:
```
{'hemiplegic': 0.10869565217391304,
 'depression': 0.043478260869565216,
 'syncope': 0.021739130434782608,
 'headache': 0.2826086956521739,
 'fibromialgia': 0.021739130434782608,
 'neuralgia': 0.043478260869565216}
```

In [73]:
```python
# Convert to list to graph
keywordsPercentSorted = dict(sorted(keywordsPercentage.items(), key = lambda item: item[1], reverse = False))
keywordName = list(keywordsPercentSorted.keys())
keywordVal = list(keywordsPercentSorted.values())
```

In [74]:
```python
# Graph the diagnoses
plt.barh(keywordName, keywordVal, color = ['#539db5', '#b5537a'])

# Annotate
for y in range(len(keywordName)):
    label = f"{keywordVal[y]:.2f}"
    plt.annotate(label, xy = (keywordVal[y], y))

plt.title("Percentage of Other Comorbid Conditions")
plt.ylabel("Other Diagnosis")
plt.xlabel("Percentage")
```

Out[74]:
```
Text(0.5, 0, 'Percentage')
```



3.6 Calculate Average PPMQ Score to Determine if Correspondents had a lower quality of life

In [75]:
```
# Save the 3 PPMQ Questions
ppmqDataq1 = migraine_data.loc[:, 'Q82_1':'Q73_10']
ppmqDataq2 = migraine_data.loc[:, 'Q84_1':'Q84_10']
ppmqDataq3 = migraine_data.loc[:, 'Q86_1':'Q86_3']
```

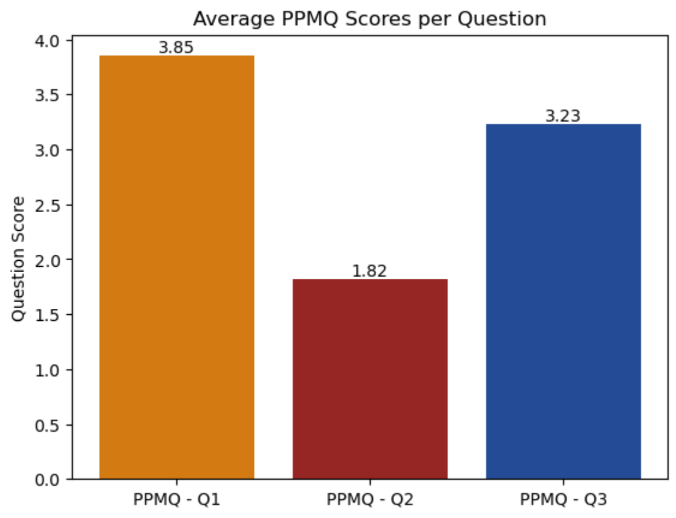
In [76]:
```
# Get average score for question 1 for every correspondent
q1 = []
for i, user in ppmqDataq1.iterrows():
    ave = sum(ppmqDataq1.iloc[i]) / len(ppmqDataq1.columns)
    q1.append(ave)
```

In [77]:
```
# Get average score for question 2 for every correspondent
q2 = []
for i, user in ppmqDataq2.iterrows():
    ave = sum(ppmqDataq2.iloc[i]) / len(ppmqDataq2.columns)
    q2.append(ave)
```

In [78]:
```
# Get average score for question 3 for every correspondent
q3 = []
for i, user in ppmqDataq3.iterrows():
    ave = sum(ppmqDataq3.iloc[i]) / len(ppmqDataq3.columns)
    q3.append(ave)
```

In [79]:
```
# Grab average score for each question
aveQ1 = sum(q1) / len(q1)
aveQ2 = sum(q2) / len(q2)
aveQ3 = sum(q3) / len(q3)
```

In [80]:
```
# Graph the average scores
questionNames = ["PPMQ - Q1", 'PPMQ - Q2', 'PPMQ - Q3']
questionAve = [aveQ1, aveQ2, aveQ3]

plt.bar(questionNames, questionAve, color = ['#d47a13', '#962624', '#244c96'])
plt.title("Average PPMQ Scores per Question")
plt.ylabel("Question Score")
barGraphLabels(questionNames, questionAve)
```



3.7 Use Proportion Test to Determine if PPMQ Scores Differ Between Female and Male

In [82]:
```
ppmqDataq1
```

Out[82]:

	Q82_1	Q82_2	Q82_3	Q82_4	Q82_5	Q82_6	Q82_7	Q82_8	Q82_9	Q73_1	Q73_2	Q73_3	Q73_4	Q73_5	Q73_6	Q73_7	Q73_8	Q73_9	Q73_10
0	1.0	3.0	3.0	3.0	2.0	1.0	1.0	3.0	3.0	4.0	3.0	2.0	2.0	3.0	6.0	6.0	6.0	6.0	5.0
1	5.0	5.0	6.0	6.0	4.0	6.0	5.0	5.0	5.0	4.0	5.0	6.0	5.0	6.0	5.0	3.0	3.0	2.0	4.0
2	3.0	3.0	3.0	3.0	3.0	3.0	4.0	4.0	4.0	3.0	3.0	3.0	4.0	4.0	4.0	2.0	2.0	3.0	4.0
3	5.0	7.0	6.0	7.0	4.0	6.0	5.0	4.0	6.0	4.0	3.0	6.0	4.0	6.0	5.0	2.0	4.0	4.0	4.0
4	1.0	2.0	3.0	4.0	2.0	6.0	4.0	0.0	4.0	4.0	2.0	1.0	3.0	2.0	2.0	1.0	1.0	1.0	1.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
195	3.0	1.0	3.0	1.0	3.0	3.0	3.0	2.0	2.0	2.0	3.0	2.0	2.0	2.0	2.0	1.0	1.0	3.0	4.0
196	3.0	3.0	3.0	3.0	3.0	3.0	5.0	5.0	5.0	5.0	6.0	5.0	6.0	5.0	6.0	3.0	3.0	3.0	7.0
197	5.0	3.0	3.0	3.0	2.0	3.0	5.0	5.0	3.0	5.0	5.0	6.0	6.0	6.0	6.0	1.0	1.0	2.0	1.0
198	5.0	4.0	5.0	5.0	4.0	5.0	6.0	6.0	6.0	6.0	4.0	4.0	4.0	4.0	5.0	4.0	4.0	7.0	7.0
199	6.0	7.0	7.0	7.0	7.0	7.0	7.0	7.0	7.0	7.0	7.0	7.0	7.0	7.0	7.0	1.0	1.0	1.0	4.0

200 rows × 19 columns

In [89]:
```
# Separate female/male respondents
maleOpinion = migraine_data.loc[lambda df: df['sex'] == 1]
femaleOpinion = migraine_data.loc[lambda df: df['sex'] == 2]

# Grab corresponding PPMQ scores for each sex
mOpinionq1 = maleOpinion.loc[:, 'Q82_1':'Q73_10']
mOpinionq2 = maleOpinion.loc[:, 'Q84_1':'Q84_10']
mOpinionq3 = maleOpinion.loc[:, 'Q86_1':'Q86_3']

fOpinionq1 = femaleOpinion.loc[:, 'Q82_1':'Q73_10']
fOpinionq2 = femaleOpinion.loc[:, 'Q84_1':'Q84_10']
fOpinionq3 = femaleOpinion.loc[:, 'Q86_1':'Q86_3']
```

In [113…
```
fOpinionq1
```

Out[113]:

	Q82_1	Q82_2	Q82_3	Q82_4	Q82_5	Q82_6	Q82_7	Q82_8	Q82_9	Q73_1	Q73_2	Q73_3	Q73_4	Q73_5	Q73_6	Q73_7	Q73_8	Q73_9	Q73_10
0	1.0	3.0	3.0	3.0	2.0	1.0	1.0	3.0	3.0	4.0	3.0	2.0	2.0	3.0	6.0	6.0	6.0	6.0	5.0
1	5.0	5.0	6.0	6.0	4.0	6.0	5.0	5.0	5.0	4.0	5.0	6.0	5.0	6.0	5.0	3.0	3.0	2.0	4.0
2	3.0	3.0	3.0	3.0	3.0	3.0	4.0	4.0	4.0	3.0	3.0	3.0	4.0	4.0	4.0	2.0	2.0	3.0	4.0
3	5.0	7.0	6.0	7.0	4.0	6.0	5.0	4.0	6.0	4.0	3.0	6.0	4.0	6.0	5.0	2.0	4.0	4.0	4.0
4	1.0	2.0	3.0	4.0	2.0	6.0	4.0	0.0	4.0	4.0	2.0	1.0	3.0	2.0	2.0	1.0	1.0	1.0	1.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
195	3.0	1.0	3.0	1.0	3.0	3.0	3.0	2.0	2.0	2.0	3.0	2.0	2.0	2.0	2.0	1.0	1.0	3.0	4.0
196	3.0	3.0	3.0	3.0	3.0	3.0	5.0	5.0	5.0	5.0	6.0	5.0	6.0	5.0	6.0	3.0	3.0	3.0	7.0
197	5.0	3.0	3.0	3.0	2.0	3.0	5.0	5.0	3.0	5.0	5.0	6.0	6.0	6.0	6.0	1.0	1.0	2.0	1.0
198	5.0	4.0	5.0	5.0	4.0	5.0	6.0	6.0	6.0	6.0	4.0	4.0	4.0	4.0	5.0	4.0	4.0	7.0	7.0
199	6.0	7.0	7.0	7.0	7.0	7.0	7.0	7.0	7.0	7.0	7.0	7.0	7.0	7.0	7.0	1.0	1.0	1.0	4.0

167 rows × 19 columns

In [118…
```
# Create Helper Function for Determining Average Scores in DF
def avePPMQ(df):
    q = []
    for i in range(len(df)):
        # Sum the scores in a row, then divide by number of columns for average
        ave = sum(df.iloc[i]) / len(df.columns)
        q.append(ave)
    return q
```

In [120…
```
# Calculate Female Average PPMQ Scores per Question
fQ1 = avePPMQ(fOpinionq1)
fQ2 = avePPMQ(fOpinionq2)
fQ3 = avePPMQ(fOpinionq3)

# Calculate Male Average PPMQ Scores per Question
mQ1 = avePPMQ(mOpinionq1)
mQ2 = avePPMQ(mOpinionq2)
mQ3 = avePPMQ(mOpinionq3)
```

In [129…
```
# Find the Proportions Where They are Satisfied on the PPMQ
# Q1 is 3 or less for increasing satisfaction
# Q2 is 2 or less
# Q3 is 3 or less

fQ1Count = fQ2Count = fQ3Count = 0
mQ1Count = mQ2Count = mQ3Count = 0

# Female Portion
for score in fQ1:
    if score <= 3:
        fQ1Count += 1
```

```
for score in fQ2:
    if score <= 2:
        fQ2Count += 1

for score in fQ3:
    if score <= 3:
        fQ3Count += 1

# Male Portion
for score in mQ1:
    if score <= 3:
        mQ1Count += 1

for score in mQ2:
    if score <= 2:
        mQ2Count += 1

for score in mQ3:
    if score <= 3:
        mQ3Count += 1
```

In [137…

```
# Create helper function to do Two Sample Proportion Test
def twoSampProp(list1, list2, size1, size2):
    results = sp.proportions_chisquare([list1, list2], [size1, size2])

    return results[0], results[1]  # Z-Score and P-Value
```

In [146…

```
# Conduct Two Sample Proportion Test Per Question
conf = 0.05

# PPMQ Q1
z, p = twoSampProp(mQ1Count, fQ1Count, len(mQ1), len(fQ1))

if p > conf:  # Fail to Rejust H0
    print("There is insufficient evidence to claim that the male group's opinion on treatment " +
          "differs from the female group's opinion \nregarding PPMQ Question 1. We FAIL to reject the null hypothesis." +
          f" Z Score: {z:.5f}, P-Value: {p:.5f}, with a confidence level: {conf}")
else:  # Reject the H0
    print("There is sufficient evidence to claim that the male group's opinion on treatment " +
          "differs from the female group's opinion \nregarding PPMQ Question 1. We REJECT the null hypothesis." +
          f" Z Score: {z:.5f}, P-Value: {p:.5f}, with a confidence level: {conf}")
```

```
There is insufficient evidence to claim that the male group's opinion on treatment differs from the female group's opinion
regarding PPMQ Question 1. We FAIL to reject the null hypothesis. Z Score: 0.61293, P-Value: 0.43369, with a confidence level: 0.05
```

In [147…

```
# PPMQ Q2
z, p = twoSampProp(mQ2Count, fQ2Count, len(mQ2), len(fQ2))

if p > conf:  # Fail to Rejust H0
    print("There is insufficient evidence to claim that the male group's opinion on treatment " +
          "differs from the female group's opinion \nregarding PPMQ Question 2. We FAIL to reject the null hypothesis." +
          f" Z Score: {z:.5f}, P-Value: {p:.5f}, with a confidence level: {conf}")
else:  # Reject the H0
    print("There is sufficient evidence to claim that the male group's opinion on treatment " +
          "differs from the female group's opinion \nregarding PPMQ Question 2. We REJECT the null hypothesis." +
          f" Z Score: {z:.5f}, P-Value: {p:.5f}, with a confidence level: {conf}")
```

```
There is insufficient evidence to claim that the male group's opinion on treatment differs from the female group's opinion
regarding PPMQ Question 2. We FAIL to reject the null hypothesis. Z Score: 0.06314, P-Value: 0.80160, with a confidence level: 0.05
```

In [148…

```
# PPMQ Q3
z, p = twoSampProp(mQ3Count, fQ3Count, len(mQ3), len(fQ3))

if p > conf:  # Fail to Rejust H0
    print("There is insufficient evidence to claim that the male group's opinion on treatment " +
          "differs from the female group's opinion \nregarding PPMQ Question 3. We FAIL to reject the null hypothesis." +
          f" Z Score: {z:.5f}, P-Value: {p:.5f}, with a confidence level: {conf}")
else:  # Reject the H0
    print("There is sufficient evidence to claim that the male group's opinion on treatment " +
          "differs from the female group's opinion \nregarding PPMQ Question 3. We REJECT the null hypothesis." +
          f" Z Score: {z:.5f}, P-Value: {p:.5f}, with a confidence level: {conf}")
```

```
There is insufficient evidence to claim that the male group's opinion on treatment differs from the female group's opinion
regarding PPMQ Question 3. We FAIL to reject the null hypothesis. Z Score: 0.00636, P-Value: 0.93644, with a confidence level: 0.05
```