Probability of Volunteering?

Pet Ownership and Education in Edmonton, CA

I found an interesting dataset from Edmonton, Canada, that surveys residents for demographics, and information on their pet ownership. I noticed that they captured information on whether or not participants volunteered, and I wondered, *does pet ownership have a relationship with volunteering?* To measure pet ownership, I created columns for whether or not the participant owned a pet, and how many pets they owned. I also brought in information on educational attainment and household income.

What follows is

- data cleaning
- missingness analysis
- exploratory data analysis
- logistic regression modelling and evaluation
- prediction metrics gathered.

CONCLUSION: Overall, I found that there is a small amount of evidence that educational attainment has a positive relationship with whether a participant volunteers. There is no relationship between pet ownership and volunteering. However, as the model does not explain volunteering very well, my confidence in this interpretation is low.

Data Import and Cleaning

In [2]:
```
# metadata here: http://www.opendatanetwork.com/dataset/data.edmonton.ca/5i9e-rgab
import pandas as pd
df = pd.read_json("https://data.edmonton.ca/resource/5i9e-rgab.json")
df.head()
```

Out[2]:

	responsedate	completiondate	q18_ownpets	q19_petkinds_dog	q19_petkinds_bird	q19_petkinds_fish	q19_petkinds_cat	q1
0	2015-05-19T21:44:00.000	2015-05-19T22:02:00.000	Yes	1.0	0.0	0.0	0.0	
1	2015-05-12T11:51:00.000	2015-05-12T11:55:00.000	Yes	1.0	0.0	0.0	0.0	
2	2015-05-12T09:19:00.000	2015-05-12T09:37:00.000	Yes	1.0	0.0	0.0	1.0	
3	2015-05-12T11:50:00.000	2015-05-12T12:05:00.000	Yes	0.0	0.0	0.0	1.0	
4	2015-05-15T13:04:00.000	2015-05-15T13:14:00.000	Yes	1.0	0.0	0.0	0.0	

5 rows × 51 columns

In [3]:
```
full_df = df.copy()
```

In [4]:
```
columns_to_sum = ['q19_petkinds_dog', 'q19_petkinds_bird',
                  'q19_petkinds_fish', 'q19_petkinds_cat',
                  'q19_petkinds_marsupial', 'q19_petkinds_insect',
                  'q19_petkinds_reptile', 'q19_petkinds_rodent', 'q19_petkinds_other']
df['sum_pet_types'] = df[columns_to_sum].sum(axis=1)

cols = ['q18_ownpets','q12_employment_status_study_profiling_questionnaire_2014',
        'q13_volunteer_study_profiling_questionnaire_2014',
        'q14a_primary_transportation_study_profiling_questionnaire_2014',
        'q15_household_income_study_profiling_questionnaire_2014',
        'q16_education_study_profiling_questionnaire_2014', 'sum_pet_types', 'q20_numberpets']
df = df[cols]
```

In [5]:
```
df[cols].head()
```

Out[5]:

	q18_ownpets	q12_employment_status_study_profiling_questionnaire_2014	q13_volunteer_study_profiling_questionnaire_2014	q1
0	Yes	Employed full-time (30+ hours a week)	Yes	
1	Yes	Employed full-time (30+ hours a week)	Yes	
2	Yes	Employed full-time (30+ hours a week)	No	
3	Yes	Employed full-time (30+ hours a week)	Yes	
4	Yes	Employed full-time (30+ hours a week)	Yes	

In [6]:
```
df.columns
```

Out[6]:
```
Index(['q18_ownpets',
       'q12_employment_status_study_profiling_questionnaire_2014',
       'q13_volunteer_study_profiling_questionnaire_2014',
       'q14a_primary_transportation_study_profiling_questionnaire_2014',
       'q15_household_income_study_profiling_questionnaire_2014',
       'q16_education_study_profiling_questionnaire_2014', 'sum_pet_types',
       'q20_numberpets'],
      dtype='object')
```

In [7]:
```
#Q: In the last 12 months, did you do any activities without pay on behalf of a group or an organization as a volun
df.q16_education_study_profiling_questionnaire_2014.value_counts()
```

Out[7]:
```
q16_education_study_profiling_questionnaire_2014
University undergraduate degree                                                          365
College / technical school graduate                                                      229
Post-graduate degree                                                                     226
High school graduate                                                                     118
Professional school graduate (e.g. medicine, dentistry, veterinary medicine, optometry)   36
Elementary/grade school graduate                                                            6
Name: count, dtype: int64
```

In [8]:
```
df.rename(columns={'q13_volunteer_study_profiling_questionnaire_2014': 'volunteer',
                   'q15_household_income_study_profiling_questionnaire_2014':'household_income',
                   'q16_education_study_profiling_questionnaire_2014': 'educational_attainment'}, inplace=True)
print(df.columns)
```

```
Index(['q18_ownpets',
       'q12_employment_status_study_profiling_questionnaire_2014', 'volunteer',
       'q14a_primary_transportation_study_profiling_questionnaire_2014',
       'household_income', 'educational_attainment', 'sum_pet_types',
       'q20_numberpets'],
      dtype='object')
```

In [9]:
```
# create a dict of old and new values
recode = {"Employed full-time (30+ hours a week)": "Full Time",
          "Employed part-time (0-30 hours a week)": "Part Time",
          "Retired" : "Retired",
          "Homemaker" : "Homemaker",
          "Unemployed" : "Unemployed",
          "Post-secondary student" : "Student",
          "High School Student": "Student",
          "Permanently unable to Work": "Unable to Work",
          "Other (Specify)" : "Other"}

# recode the variable
df["employment"] = df["q12_employment_status_study_profiling_questionnaire_2014"].map(recode)

# check value counts
df.employment.value_counts()
```

Out[9]:
```
employment
Full Time         652
Retired           116
Part Time          89
Student            33
Homemaker          33
Other              29
Unemployed         18
Unable to Work     10
Name: count, dtype: int64
```

In [10]:
```
# create a dict of old and new values
recode = {"University undergraduate degree": "Undergraduate",
          "College / technical school graduate": "College or Tech",
          "Post-graduate degree" : "Post-Graduate",
          "High school graduate" : "High School",
```

```
         "Professional school graduate (e.g. medicine, dentistry, veterinary medicine, optometry)" : "Professional
         "Elementary/grade school graduate" : "Grade School",
         }

# recode the variable
df.educational_attainment = df.educational_attainment.map(recode)
```

In [11]:
```
df = df.drop(columns =['q12_employment_status_study_profiling_questionnaire_2014','q14a_primary_transportation_stud
```

In [12]:
```
edu_order = ['Grade School','High School',  'College or Tech',
             'Undergraduate', 'Post-Graduate', 'Professional Post-Graduate']
```

In [13]:
```
df['educational_attainment_encoded'] = pd.Categorical(df['educational_attainment'],
     categories=edu_order, ordered=True)
```

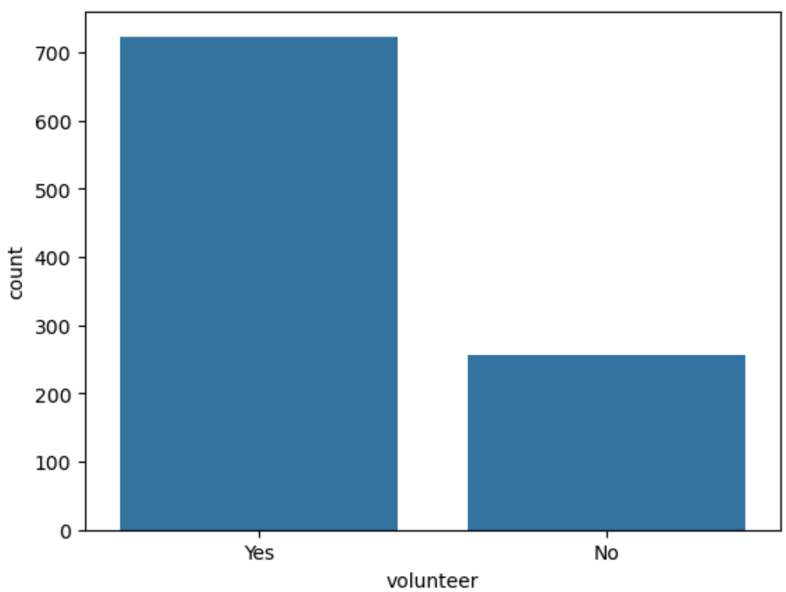
In [14]:
```
income_order = ['Under $25,000', '$25,000 to $49,999',
                '$50,000 to $74,999', '$75,000 to $99,999',
                '$100,000 to $124,999', '$125,000 to $149,999',
                '$150,000 to $199,999', '$200,000 and over',
                "Don't know", 'Prefer not to answer']
df['household_income_encoded'] = pd.Categorical(df['household_income'],
     categories=income_order, ordered=True)
```

In [15]:
```
import pandas as pd
income_order = ['Prefer not to answer', 'Under $20,000', '$20,000 to $29,999', '$30,000 to $39,999',
                '$40,000 to $49,999', '$50,000 to $59,999', '$60,000 to $79,999',
                '$80,000 to $99,999', '$100,000 to $149,000', "$150,000 and over"]
df['household_income_encoded'] = pd.Categorical(df['household_income'], categories=income_order, ordered=True)
```

Graphing Distributions

In [16]:
```
import seaborn as sns
import matplotlib.pyplot as plt

sns.countplot(x='volunteer', data=df)
plt.show()
```



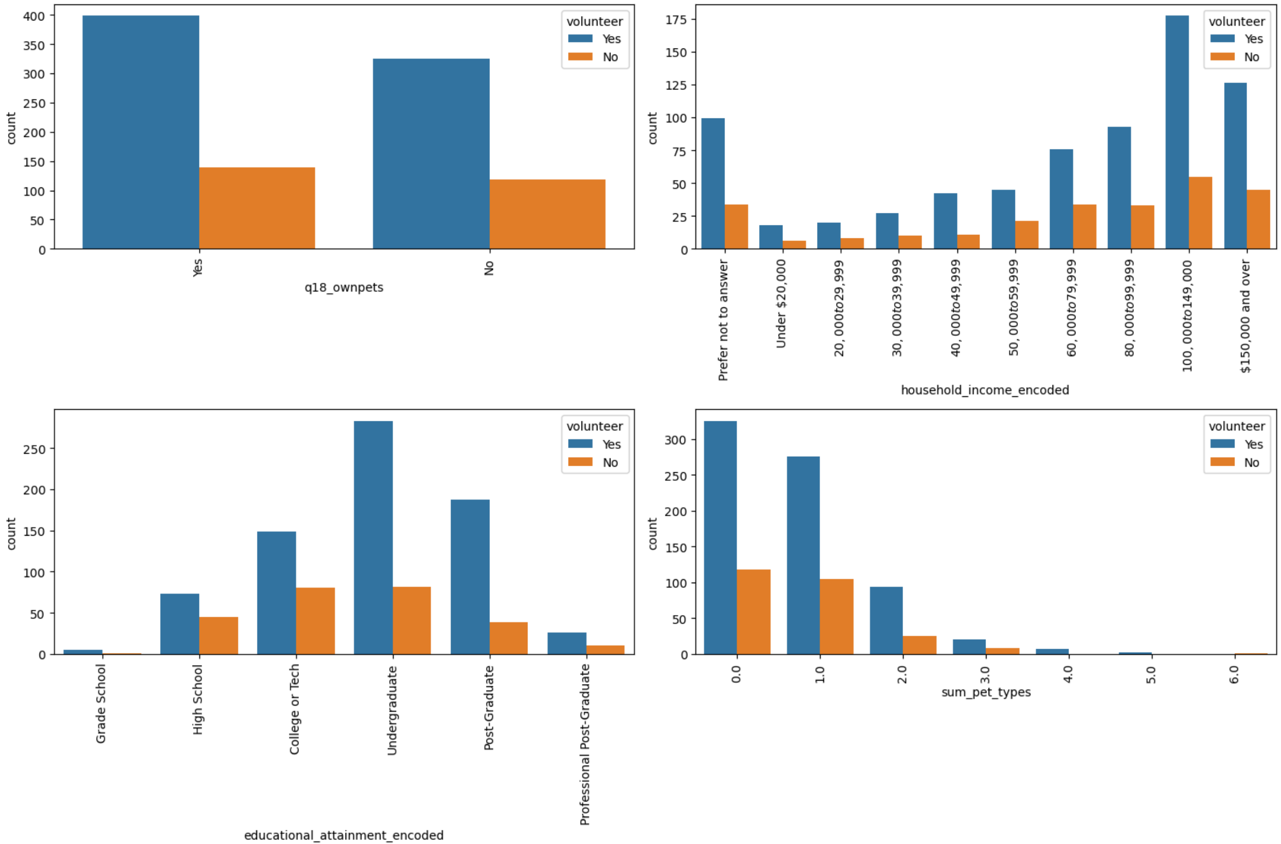
In [17]:
```
# Create a grid of subplots
fig, axes = plt.subplots(2, 2, figsize=(15, 10))

# Plot the distribution of each column
sns.countplot(x='q18_ownpets', data=df, ax=axes[0, 0], hue='volunteer')
sns.countplot(x='household_income_encoded', data=df, ax=axes[0, 1], hue='volunteer')
sns.countplot(x='educational_attainment_encoded', data=df, ax=axes[1, 0], hue='volunteer')
sns.countplot(x='sum_pet_types', data=df, ax=axes[1, 1],hue='volunteer')

# Rotate x-axis labels for better readability
for ax in axes.flat:
  ax.tick_params(axis='x', rotation=90)

# Adjust layout and display the plot
```

```
plt.tight_layout()
plt.show()
```



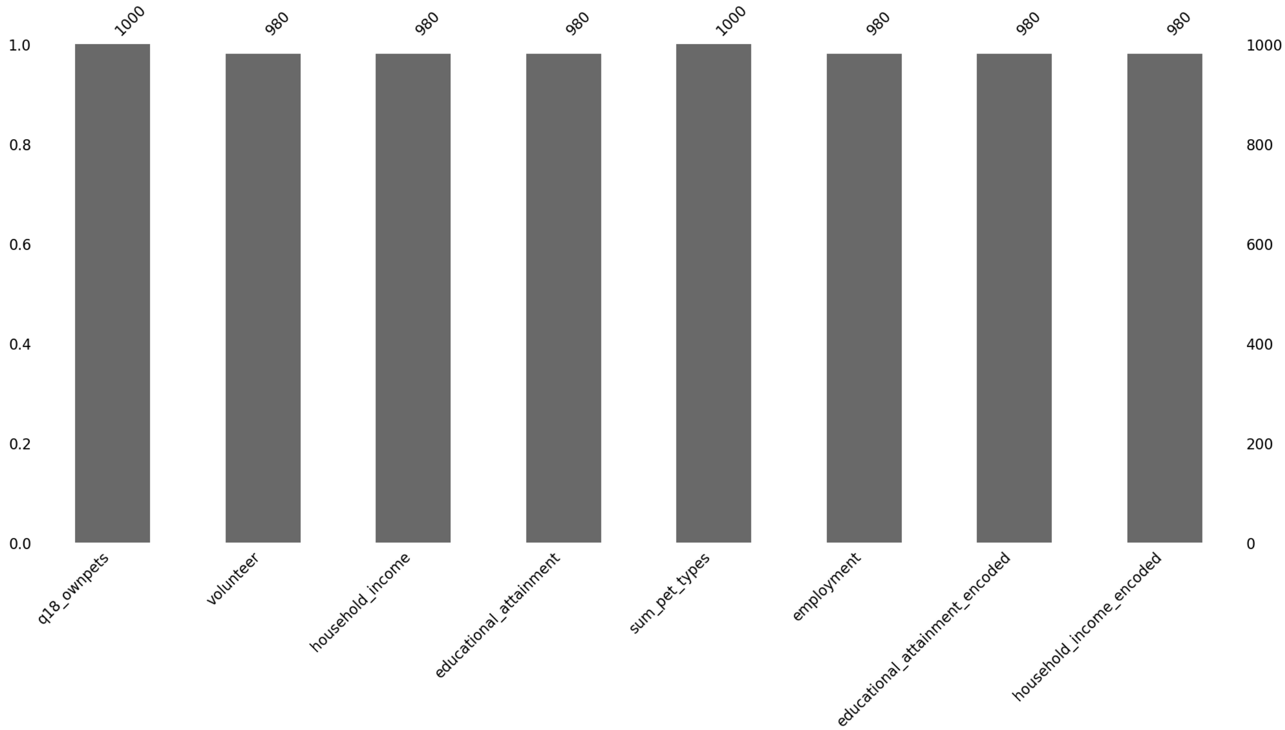
Two-thirds of the dataset volunteers, and visually it seems as though these people are well spread out among all categories. But there could be a relationship here that's hard to visually detect.

Missingness Analysis

In [18]:
```
%matplotlib inline
import missingno as msno

msno.bar(df)
```

Out[18]: <Axes: >



In [19]:
```
print(f"nulls in the dataset:\n {df.isnull().sum()} ")
```

```
nulls in the dataset:
 q18_ownpets                       0
volunteer                        20
household_income                 20
educational_attainment           20
sum_pet_types                     0
employment                       20
educational_attainment_encoded   20
household_income_encoded         20
dtype: int64
```

Dropping missing values:

In [20]:
```
df = df.dropna()
df.isnull().sum()
```

Out[20]:
```
q18_ownpets                       0
volunteer                         0
household_income                  0
educational_attainment            0
sum_pet_types                     0
employment                        0
educational_attainment_encoded    0
household_income_encoded          0
dtype: int64
```

Transform Dataframe for Modeling

In [21]:
```
from sklearn.preprocessing import LabelEncoder

label_encoder = LabelEncoder()
label_mappings = {}

# Encoding categorical columns, preserving order for ordinal variables
for col in df.columns:
  if col in ['educational_attainment_encoded', 'household_income_encoded']:
    print('hsi')
    df[col] = df[col].cat.codes  # Use cat.codes to preserve order
  elif col in ['q18_ownpets', 'employment','volunteer']:
    df[col] = label_encoder.fit_transform(df[col])
    label_mappings[col] = dict(zip(label_encoder.classes_, label_encoder.transform(label_encoder.classes_)))
```

```
hsi
hsi
```

In [22]:
```
print(label_mappings)
df.head()
```

```
{'q18_ownpets': {'No': 0, 'Yes': 1}, 'volunteer': {'No': 0, 'Yes': 1}, 'employment': {'Full Time': 0, 'Homemaker': 1, 'Other': 2, 'Part Time': 3, 'Retired': 4, 'Student': 5, 'Unable to Work': 6, 'Unemployed': 7}}
```

Out[22]:

	q18_ownpets	volunteer	household_income	educational_attainment	sum_pet_types	employment	educational_attainment_enco
0	1	1	$80,000 to $99,999	Post-Graduate	1.0	0	
1	1	1	$100,000 to $149,000	Undergraduate	4.0	0	
2	1	0	$100,000 to $149,000	Undergraduate	2.0	0	
3	1	1	$80,000 to $99,999	Undergraduate	1.0	0	
4	1	1	$100,000 to $149,000	Undergraduate	2.0	0	

In [23]:
```
df.drop(columns = ['educational_attainment', 'household_income'], inplace=True)
```

In [24]:
```
from sklearn.model_selection import train_test_split

X = df.drop('volunteer', axis=1)
y = df['volunteer']
# Split data into train and test sets, balancing according to 'volunteer'
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, stratify=y, random_state=42)

print("X_train shape:", X_train.shape)
print("X_test shape:", X_test.shape)
print("y_train shape:", y_train.shape)
print("y_test shape:", y_test.shape)
```

```
X_train shape: (784, 5)
X_test shape: (196, 5)
y_train shape: (784,)
y_test shape: (196,)
```

In [25]:
```
X_train.head()
```

Out[25]:

	q18_ownpets	sum_pet_types	employment	educational_attainment_encoded	household_income_encoded
36	0	0.0	0	5	8
124	1	1.0	0	3	7
317	0	0.0	4	3	3
660	1	1.0	3	3	7
285	0	0.0	0	3	4

Modeling

In [27]:
```
from imblearn.under_sampling import RandomUnderSampler


rus = RandomUnderSampler(random_state=42)
X_train_resampled, y_train_resampled = rus.fit_resample(X_train, y_train) # Resample the training data

print("Class distribution after undersampling:")
print(y_train_resampled.value_counts())
```

```
Class distribution after undersampling:
volunteer
0    206
1    206
Name: count, dtype: int64
```

In [ ]:
```
import numpy as np
import statsmodels.api as sm

# Add a constant to the independent variables
X_train_const = sm.add_constant(X_train_resampled)
X_test_const = sm.add_constant(X_test)

# Fit the logistic regression model with class weights
logit_model = sm.Logit(y_train_resampled, X_train_const)
result = logit_model.fit()

print(result.summary())
```

```
Optimization terminated successfully.
         Current function value: 0.674781
         Iterations 5
                           Logit Regression Results
==============================================================================
Dep. Variable:              volunteer   No. Observations:                  412
Model:                          Logit   Df Residuals:                      406
Method:                           MLE   Df Model:                            5
Date:                Thu, 19 Sep 2024   Pseudo R-squ.:                 0.02650
Time:                        20:49:53   Log-Likelihood:                -278.01
converged:                       True   LL-Null:                       -285.58
Covariance Type:            nonrobust   LLR p-value:                  0.009805
==================================================================================================
                                     coef    std err          z      P>|z|      [0.025      0.975]
--------------------------------------------------------------------------------------------------
const                             -1.0402      0.374     -2.778      0.005      -1.774      -0.306
q18_ownpets                       -0.4437      0.354     -1.254      0.210      -1.137       0.250
sum_pet_types                      0.3346      0.212      1.581      0.114      -0.080       0.749
employment                         0.0470      0.057      0.820      0.412      -0.065       0.159
educational_attainment_encoded     0.3324      0.097      3.420      0.001       0.142       0.523
household_income_encoded           0.0141      0.037      0.383      0.702      -0.058       0.086
==================================================================================================
```

There is no quanitifiable effect of pet ownership on volunteering in this dataset. Educational attainment is the only significant predictor at the 0.05 alpha level using a z, meaning that I can reject the null hypothesis that this coefficient is equal to 0.

The model does not do a great job of explaining the distribution of Volunteering. This is evidenced by the log-likelihood, which does not improve much between the full and null model.

This is not a huge surprise - it's very believable that there are things outside of what is captured in these data that influence if a person volunteers or not.

In [ ]:
```
result.wald_test_terms()
```

```
/usr/local/lib/python3.10/dist-packages/statsmodels/base/model.py:1912: FutureWarning: The behavior of wald_test wil
l change after 0.14 to returning scalar test statistic values. To get the future behavior now, set scalar to True. T
o silence this message while retaining the legacy behavior, set scalar to False.
  warnings.warn(
```

Out[ ]:
```
<class 'statsmodels.stats.contrast.WaldTestResults'>
                                                  chi2                P>chi2  df constraint
const                             [[7.717067273255586]]   0.005470118467288624              1
q18_ownpets                      [[1.5720592949842729]]    0.20990820603335095              1
sum_pet_types                    [[2.5001510530292776]]    0.11383537911798212              1
employment                       [[0.6731437972893429]]     0.4119576787943148              1
educational_attainment_encoded   [[11.698516267051433]]  0.0006254995472184443              1
household_income_encoded         [[0.1467165324940548]]     0.7016931151107725              1
```

The wald test confirms that the Educational Attainment Encoded column is the only variable that is significantly different from 0. Going forward, I'll only interpret this variable.

Interpretaion of Model

Education on Volunteering

In [ ]:
```
coefficients = result.params
odds_ratios = pd.DataFrame({
    'Variable': coefficients.index,
    'Coefficient': coefficients.values,
    'Odds Ratio': np.exp(coefficients.values)
})

print(odds_ratios)
```

```
                         Variable  Coefficient  Odds Ratio
0                           const    -1.040198    0.353385
1                     q18_ownpets    -0.443692    0.641663
2                   sum_pet_types     0.334633    1.397428
3                      employment     0.047016    1.048138
4  educational_attainment_encoded     0.332367    1.394265
5        household_income_encoded     0.014092    1.014192
```

Educational attainment has a positive odds ratio, meaning that the odds of the outcome are higher in the more educated groups

In [ ]:
```
# Create a baseline prediction (probability of positive response with all features at 0)
baseline_prediction = 1 / (1 + np.exp(-result.params['const']))
```

```
# Initialize lists to store results
features = []
coefficients_list = []
probability_changes = []
new_probabilities = []

# Iterate through the coefficients (excluding the intercept)
for feature, coefficient in coefficients.items():
  if feature != 'const':
    log_odds_change = coefficient # Calculate the change in log-odds for a one-unit increase in the feature
    probability_change = np.exp(log_odds_change) / (1 + np.exp(log_odds_change)) # Calculate the change in probabil
    new_probability = baseline_prediction * probability_change # Calculate the new probability of a positive respon

    # Append results to lists
    features.append(feature)
    coefficients_list.append(coefficient)
    probability_changes.append(probability_change)
    new_probabilities.append(new_probability)

# Create a DataFrame from the results
results_df = pd.DataFrame({
    'Feature': features,
    'Coefficient': coefficients_list,
    'Probability Change': probability_changes,
    'New Probability': new_probabilities
})

results_df
```

Out[ ]:

	Feature	Coefficient	Probability Change	New Probability
0	q18_ownpets	-0.413478	0.398078	0.201827
1	sum_pet_types	0.290896	0.572215	0.290115
2	employment	0.070421	0.517598	0.262424
3	educational_attainment_encoded	0.364635	0.590162	0.299214
4	household_income_encoded	-0.009459	0.497635	0.252303

There is a greater probability of volunteering given educational attainment: 29%. I don't believe this is a trustworthy estimate, given that this model doesn't explain the distribution of volunteering well, as shown above.

In [ ]:

```
from sklearn.metrics import accuracy_score, roc_auc_score

y_pred_prob = result.predict(X_test_const) # Calculate predicted probabilities for the test set
y_pred = (y_pred_prob >= 0.5).astype(int) # Predict classes for the test set

accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

auc = roc_auc_score(y_test, y_pred_prob)
print("AUC:", auc)
```

```
Accuracy: 0.7397959183673469
AUC: 0.5485463150777552
```

The accuracy, or true positive rate, is not bad, but the AUC is hardly better than average.