h1n1-vaccine

May 22, 2024

1 Predicting H1N1 Vaccine Uptake: Insights from the 2009 National H1N1 Flu Survey

[ ]:
```
from IPython.display import Image
Image(filename='Vaccine_uptake.jpg')
```

1.1 Final Project Submission

Please fill out: * Student name: Wambui Githinji * Student pace: Part time * Scheduled project review date/time: * Instructor name: William Okomba and Noah Kandie * Blog post URL:

1.2 TABLE OF CONTENTS

1.3 INTRODUCTION

The H1N1 influenza virus, commonly known as swine flu, gained global attention during the 2009 H1N1 pandemic. This strain of influenza prompted widespread concern due to its potential for rapid transmission and severe health consequences. In response to the pandemic, the National 2009 H1N1 Flu Survey was conducted to gather data on various aspects of the outbreak, including vaccination uptake among the population. Understanding how people's backgrounds, opinions, and health behaviors influence their vaccination decisions is crucial for shaping effective public health strategies.s.

1.4 BUSINESS UNDERSTANDING

Vaccine uptake is a crucial aspect of public health, directly impacting disease prevention and population well-being. By analyzing the factors influencing vaccine acceptance, public health authorities can develop targeted strategies to improve vaccination rates and enhance overall community immunity. Understanding and predicting vaccine uptake enables proactive measures, such as tailored communication campaigns and accessible vaccination programs, to address barriers and increase vaccination coverage. This analysis aids in identifying groups at risk of low vaccine uptake and informs resource allocation for effective public health interventions, ultimately contributing to better health outcomes and disease control.

1.5 PROBLEM STATEMENT

- The goal of this project is to develop a predictive model that can accurately assess the probability of individuals opting for the H1N1 vaccine based on various demographic, socio-economic, and attitudinal factors.

1.6 OBJECTIVES

1. Develop a Robust Predictive Model capable of estimating the likelihood of H1N1 vaccine uptake for individuals.
2. Feature importance: Identify Key Predictors of H1N1 Vaccine Acceptance
3. Generate insights and recommendations: Analyze the outcomes of the analysis and models to derive actionable insights and recommendations for enhancing vaccine uptake strategies.

1.7 MODEL SUCCESS CRITERIA

- In predicting vaccine uptake, it's crucial to identify as many true positive cases (individuals taking the vaccine) as possible. Recall, which measures how effectively the model identifies these positive cases, is ideal for this purpose. Understanding the characteristics associated with vaccine uptake helps tailor campaigns, ensuring resources are used efficiently to boost vaccination rates in specific groups.
- The F1 Score, which is a balance between precision and recall, ensures a fair trade-off, considering both false positives and false negatives. This aligns well with the goal of accurately predicting vaccine uptake.
- For evaluating model performance, I'll use the following metrics:

1. AUC-ROC score of 80% or higher
2. Balanced Recall, especially since the target variable is heavily imbalanced, aiming for 50% or more
3. Accuracy of 80% or higher
4. F1 Score of 50% or more

- These metrics provide a comprehensive evaluation of the model's ability to predict vaccine uptake accurately and reliably.

1.8 DATA UNDERSTANDING

The data for this project originates from the National 2009 H1N1 Flu Survey conducted in the United States during the 2009 Influenza outbreak. You can access the datasets on the Driven Data website. Primarily, the dataset comprises categorical variables with binary and numerical values. Additionally, certain columns contain coded data.

For all binary variables: 0 = No; 1 = Yes.

h1n1_concern - Level of concern about the H1N1 flu.

- 0 = Not at all concerned; 1 = Not very concerned; 2 = Somewhat concerned; 3 = Very concerned.

h1n1_knowledge - Level of knowledge about H1N1 flu.

- 0 = No knowledge; 1 = A little knowledge; 2 = A lot of knowledge.

behavioral_antiviral_meds - Has taken antiviral medications. (binary)

behavioral_avoidance - Has avoided close contact with others with flu-like symptoms. (binary)

behavioral_face_mask - Has bought a face mask. (binary)

behavioral_wash_hands - Has frequently washed hands or used hand sanitizer. (binary)

behavioral_large_gatherings - Has reduced time at large gatherings. (binary)

behavioral_outside_home - Has reduced contact with people outside of own household. (binary)

behavioral_touch_face - Has avoided touching eyes, nose, or mouth. (binary)

doctor_recc_h1n1 - H1N1 flu vaccine was recommended by doctor. (binary)

doctor_recc_seasonal - Seasonal flu vaccine was recommended by doctor. (binary)

chronic_med_condition - Has any of the following chronic medical conditions: asthma or an other lung condition, diabetes, a heart condition, a kidney condition, sickle cell anemia or other anemia, a neurological or neuromuscular condition, a liver condition, or a weakened immune system caused by a chronic illness or by medicines taken for a chronic illness. (binary)

child_under_6_months - Has regular close contact with a child under the age of six months. (binary)

health_worker - Is a healthcare worker. (binary)

health_insurance - Has health insurance. (binary)

opinion_h1n1_vacc_effective - Respondent's opinion about H1N1 vaccine effectiveness.

- 1 = Not at all effective; 2 = Not very effective; 3 = Don't know; 4 = Somewhat effective; 5 = Very effective.

opinion_h1n1_risk - Respondent's opinion about risk of getting sick with H1N1 flu without vaccine.

- 1 = Very Low; 2 = Somewhat low; 3 = Don't know; 4 = Somewhat high; 5 = Very high.

opinion_h1n1_sick_from_vacc - Respondent's worry of getting sick from taking H1N1 vaccine.

- 1 = Not at all worried; 2 = Not very worried; 3 = Don't know; 4 = Somewhat worried; 5 = Very worried.

opinion_seas_vacc_effective - Respondent's opinion about seasonal flu vaccine effectiveness.

- 1 = Not at all effective; 2 = Not very effective; 3 = Don't know; 4 = Somewhat effective; 5 = Very effective.

opinion_seas_risk - Respondent's opinion about risk of getting sick with seasonal flu without vaccine.

- 1 = Very Low; 2 = Somewhat low; 3 = Don't know; 4 = Somewhat high; 5 = Very high.

opinion_seas_sick_from_vacc - Respondent's worry of getting sick from taking seasonal flu vaccine.

- 1 = Not at all worried; 2 = Not very worried; 3 = Don't know; 4 = Somewhat worried; 5 = Very worried.

age_group - Age group of respondent.

education - Self-reported education level.

race - Race of respondent.

sex - Sex of respondent.

income_poverty - Household annual income of respondent with respect to 2008 Census poverty thresholds.

marital_status - Marital status of respondent.

rent_or_own - Housing situation of respondent.

employment_status - Employment status of respondent.

hhs_geo_region - Respondent's residence using a 10-region geographic classification defined by the U.S. Dept. of Health and Human Services. Values are represented as short random character strings.

census_msa - Respondent's residence within metropolitan statistical areas (MSA) as defined by the U.S. Census.

household_adults - Number of other adults in household, top-coded to 3.

household_children - Number of children in household, top-coded to 3.

employment_industry - Type of industry respondent is employed in. Values are represented as short random character strings.

employment_occupation - Type of occupation of respondent. Values are represented as short random character strings.

IMPORTING LIBRARIES

[2]:
```python
# Import libraries

import pandas as pd
import numpy as np
import random
import math
import warnings
warnings.filterwarnings('ignore')


from sklearn.impute import SimpleImputer
from sklearn.preprocessing import LabelEncoder, OneHotEncoder,  StandardScaler
from sklearn.compose import ColumnTransformer
from sklearn.model_selection import train_test_split, cross_val_score,␣
↪GridSearchCV
from imblearn.over_sampling import SMOTE
from sklearn.linear_model import Lasso, Ridge, LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix,␣
↪roc_auc_score, f1_score, precision_score, recall_score,  accuracy_score, auc
from sklearn.metrics import precision_recall_curve, ConfusionMatrixDisplay,␣
↪roc_curve
from statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.pipeline import Pipeline



import matplotlib.pyplot as plt
import seaborn as sns
```

1.9 DATA LOADING

[3]:
```python
# Loading the 1st dataset - training features

f1 = r"training_set_features.csv"
data_1 = pd.read_csv(f1)
```

[4]:
```python
# Loading the 2nd dataset - labels

f2 = r"training_set_labels.csv"
data_2 = pd.read_csv(f2)
```

1.10 DATA INSPECTION AND UNDERSTANDING

[5]:
```python
# Preview the 1st five rows of our features dataset

data_1.head()
```

[5]:
```
   respondent_id  h1n1_concern  h1n1_knowledge  behavioral_antiviral_meds  \
0              0           1.0             0.0                        0.0
1              1           3.0             2.0                        0.0
2              2           1.0             1.0                        0.0
3              3           1.0             1.0                        0.0
4              4           2.0             1.0                        0.0

   behavioral_avoidance  behavioral_face_mask  behavioral_wash_hands  \
0                   0.0                   0.0                    0.0
1                   1.0                   0.0                    1.0
2                   1.0                   0.0                    0.0
3                   1.0                   0.0                    1.0
4                   1.0                   0.0                    1.0

   behavioral_large_gatherings  behavioral_outside_home  \
0                          0.0                      1.0
1                          0.0                      1.0
2                          0.0                      0.0
3                          1.0                      0.0
4                          1.0                      0.0

   behavioral_touch_face  …             income_poverty  marital_status  \
0                    1.0  …              Below Poverty     Not Married
1                    1.0  …              Below Poverty     Not Married
2                    0.0  …  <= $75,000, Above Poverty     Not Married
3                    0.0  …              Below Poverty     Not Married
4                    1.0  …  <= $75,000, Above Poverty         Married

   rent_or_own   employment_status  hhs_geo_region                census_msa  \
0          Own  Not in Labor Force        oxchjgsf                   Non-MSA
1         Rent            Employed        bhuqouqj  MSA, Not Principle  City
2          Own            Employed        qufhixun  MSA, Not Principle  City
3         Rent  Not in Labor Force        lrircsnp       MSA, Principle City
4          Own            Employed        qufhixun  MSA, Not Principle  City

   household_adults  household_children  employment_industry  \
0               0.0                 0.0                  NaN
1               0.0                 0.0             pxcmvdjn
2               2.0                 0.0             rucpziij
3               0.0                 0.0                  NaN
4               1.0                 0.0             wxleyezf
```

```
   employment_occupation
0                    NaN
1               xgwztkwe
2               xtkaffoo
3                    NaN
4               emcorrxb

[5 rows x 36 columns]
```

[6]:
```
# Preview the last five rows of our features data set

data_1.tail()
```

[6]:
```
       respondent_id  h1n1_concern  h1n1_knowledge  behavioral_antiviral_meds  \
26702          26702           2.0             0.0                        0.0
26703          26703           1.0             2.0                        0.0
26704          26704           2.0             2.0                        0.0
26705          26705           1.0             1.0                        0.0
26706          26706           0.0             0.0                        0.0

       behavioral_avoidance  behavioral_face_mask  behavioral_wash_hands  \
26702                   1.0                   0.0                    0.0
26703                   1.0                   0.0                    1.0
26704                   1.0                   1.0                    1.0
26705                   0.0                   0.0                    0.0
26706                   1.0                   0.0                    0.0

       behavioral_large_gatherings  behavioral_outside_home  \
26702                          0.0                      1.0
26703                          0.0                      0.0
26704                          1.0                      0.0
26705                          0.0                      0.0
26706                          0.0                      0.0

       behavioral_touch_face  …             income_poverty  marital_status  \
26702                    0.0  …  <= $75,000, Above Poverty     Not Married
26703                    0.0  …  <= $75,000, Above Poverty     Not Married
26704                    1.0  …                        NaN     Not Married
26705                    NaN  …  <= $75,000, Above Poverty         Married
26706                    0.0  …  <= $75,000, Above Poverty         Married

       rent_or_own   employment_status  hhs_geo_region  \
26702          Own  Not in Labor Force        qufhixun
26703         Rent            Employed        lzgpxyit
26704          Own                 NaN        lzgpxyit
26705         Rent            Employed        lrircsnp
```

```
26706          Own  Not in Labor Force       mlyzmhmf

                    census_msa  household_adults  household_children  \
26702                  Non-MSA               0.0                 0.0
26703       MSA, Principle City              1.0                 0.0
26704  MSA, Not Principle  City              0.0                 0.0
26705                  Non-MSA               1.0                 0.0
26706       MSA, Principle City              1.0                 0.0

       employment_industry  employment_occupation
26702                  NaN                    NaN
26703             fcxhlnwr               cmhcxjea
26704                  NaN                    NaN
26705             fcxhlnwr               haliazsg
26706                  NaN                    NaN

[5 rows x 36 columns]
```

[7]:

```python
# Checking the info and uniformity of our dataframe

data_1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 26707 entries, 0 to 26706
Data columns (total 36 columns):
 #   Column                       Non-Null Count  Dtype
---  ------                       --------------  -----
 0   respondent_id                26707 non-null  int64
 1   h1n1_concern                 26615 non-null  float64
 2   h1n1_knowledge               26591 non-null  float64
 3   behavioral_antiviral_meds    26636 non-null  float64
 4   behavioral_avoidance         26499 non-null  float64
 5   behavioral_face_mask         26688 non-null  float64
 6   behavioral_wash_hands        26665 non-null  float64
 7   behavioral_large_gatherings  26620 non-null  float64
 8   behavioral_outside_home      26625 non-null  float64
 9   behavioral_touch_face        26579 non-null  float64
 10  doctor_recc_h1n1             24547 non-null  float64
 11  doctor_recc_seasonal         24547 non-null  float64
 12  chronic_med_condition        25736 non-null  float64
 13  child_under_6_months         25887 non-null  float64
 14  health_worker                25903 non-null  float64
 15  health_insurance             14433 non-null  float64
 16  opinion_h1n1_vacc_effective  26316 non-null  float64
 17  opinion_h1n1_risk            26319 non-null  float64
 18  opinion_h1n1_sick_from_vacc  26312 non-null  float64
 19  opinion_seas_vacc_effective  26245 non-null  float64
```

```
 20   opinion_seas_risk            26193 non-null   float64
 21   opinion_seas_sick_from_vacc  26170 non-null   float64
 22   age_group                    26707 non-null   object
 23   education                    25300 non-null   object
 24   race                         26707 non-null   object
 25   sex                          26707 non-null   object
 26   income_poverty               22284 non-null   object
 27   marital_status               25299 non-null   object
 28   rent_or_own                  24665 non-null   object
 29   employment_status            25244 non-null   object
 30   hhs_geo_region               26707 non-null   object
 31   census_msa                   26707 non-null   object
 32   household_adults             26458 non-null   float64
 33   household_children           26458 non-null   float64
 34   employment_industry          13377 non-null   object
 35   employment_occupation        13237 non-null   object
dtypes: float64(23), int64(1), object(12)
memory usage: 7.3+ MB
```

[8]:
```
# Checking data numerical summaries

data_1.describe()
```

[8]:
```
       respondent_id  h1n1_concern  h1n1_knowledge  behavioral_antiviral_meds  \
count   26707.000000  26615.000000    26591.000000               26636.000000
mean    13353.000000      1.618486        1.262532                   0.048844
std      7709.791156      0.910311        0.618149                   0.215545
min         0.000000      0.000000        0.000000                   0.000000
25%      6676.500000      1.000000        1.000000                   0.000000
50%     13353.000000      2.000000        1.000000                   0.000000
75%     20029.500000      2.000000        2.000000                   0.000000
max     26706.000000      3.000000        2.000000                   1.000000

       behavioral_avoidance  behavioral_face_mask  behavioral_wash_hands  \
count          26499.000000          26688.000000           26665.000000
mean               0.725612              0.068982               0.825614
std                0.446214              0.253429               0.379448
min                0.000000              0.000000               0.000000
25%                0.000000              0.000000               1.000000
50%                1.000000              0.000000               1.000000
75%                1.000000              0.000000               1.000000
max                1.000000              1.000000               1.000000

       behavioral_large_gatherings  behavioral_outside_home  \
count                  26620.00000             26625.000000
mean                       0.35864                 0.337315
std                        0.47961                 0.472802
```

```
min                        0.00000                0.000000
25%                        0.00000                0.000000
50%                        0.00000                0.000000
75%                        1.00000                1.000000
max                        1.00000                1.000000

       behavioral_touch_face  …  health_worker  health_insurance  \
count           26579.000000  …   25903.000000       14433.00000
mean                0.677264  …       0.111918           0.87972
std                 0.467531  …       0.315271           0.32530
min                 0.000000  …       0.000000           0.00000
25%                 0.000000  …       0.000000           1.00000
50%                 1.000000  …       0.000000           1.00000
75%                 1.000000  …       0.000000           1.00000
max                 1.000000  …       1.000000           1.00000

       opinion_h1n1_vacc_effective  opinion_h1n1_risk  \
count                 26316.000000       26319.000000
mean                      3.850623           2.342566
std                       1.007436           1.285539
min                       1.000000           1.000000
25%                       3.000000           1.000000
50%                       4.000000           2.000000
75%                       5.000000           4.000000
max                       5.000000           5.000000

       opinion_h1n1_sick_from_vacc  opinion_seas_vacc_effective  \
count                 26312.000000                 26245.000000
mean                      2.357670                     4.025986
std                       1.362766                     1.086565
min                       1.000000                     1.000000
25%                       1.000000                     4.000000
50%                       2.000000                     4.000000
75%                       4.000000                     5.000000
max                       5.000000                     5.000000

       opinion_seas_risk  opinion_seas_sick_from_vacc  household_adults  \
count       26193.000000                 26170.000000      26458.000000
mean            2.719162                     2.118112          0.886499
std             1.385055                     1.332950          0.753422
min             1.000000                     1.000000          0.000000
25%             2.000000                     1.000000          0.000000
50%             2.000000                     2.000000          1.000000
75%             4.000000                     4.000000          1.000000
max             5.000000                     5.000000          3.000000

       household_children
```

```
count        26458.000000
mean             0.534583
std              0.928173
min              0.000000
25%              0.000000
50%              0.000000
75%              1.000000
max              3.000000

[8 rows x 24 columns]
```

[9]:
```python
# Checking the shape of our dataframe

data_1.shape
```

[9]:
```
(26707, 36)
```

[10]:
```python
# Previewing our labels dataset

data_2.head()
```

[10]:
```
   respondent_id  h1n1_vaccine  seasonal_vaccine
0              0             0                 0
1              1             0                 1
2              2             0                 0
3              3             0                 1
4              4             0                 0
```

[11]:
```python
# Checking the shape of our dataframe

data_2.shape
```

[11]:
```
(26707, 3)
```

- Our features and labels datasets share similar rows, facilitating a straightforward merge of the datasets.

Merging our datasets

[12]:
```python
# Let's merge our two datasets

merged_df = pd.merge(data_1, data_2, on='respondent_id', how='left')
```

[13]:
```python
# Previewing the merged dataframe

merged_df.head()
```

[13]:
```
   respondent_id  h1n1_concern  h1n1_knowledge  behavioral_antiviral_meds  \
0              0           1.0             0.0                        0.0
1              1           3.0             2.0                        0.0
2              2           1.0             1.0                        0.0
3              3           1.0             1.0                        0.0
4              4           2.0             1.0                        0.0

   behavioral_avoidance  behavioral_face_mask  behavioral_wash_hands  \
0                   0.0                   0.0                    0.0
1                   1.0                   0.0                    1.0
2                   1.0                   0.0                    0.0
3                   1.0                   0.0                    1.0
4                   1.0                   0.0                    1.0

   behavioral_large_gatherings  behavioral_outside_home  \
0                          0.0                      1.0
1                          0.0                      1.0
2                          0.0                      0.0
3                          1.0                      0.0
4                          1.0                      0.0

   behavioral_touch_face  …  rent_or_own   employment_status  \
0                    1.0  …          Own  Not in Labor Force
1                    1.0  …         Rent            Employed
2                    0.0  …          Own            Employed
3                    0.0  …         Rent  Not in Labor Force
4                    1.0  …          Own            Employed

   hhs_geo_region                census_msa  household_adults  \
0        oxchjgsf                   Non-MSA               0.0
1        bhuqouqj  MSA, Not Principle  City               0.0
2        qufhixun  MSA, Not Principle  City               2.0
3        lrircsnp       MSA, Principle City               0.0
4        qufhixun  MSA, Not Principle  City               1.0

   household_children  employment_industry  employment_occupation  \
0                 0.0                  NaN                    NaN
1                 0.0             pxcmvdjn               xgwztkwe
2                 0.0             rucpziij               xtkaffoo
3                 0.0                  NaN                    NaN
4                 0.0             wxleyezf               emcorrxb

   h1n1_vaccine  seasonal_vaccine
0             0                 0
1             0                 1
2             0                 0
3             0                 1
```

```
4            0                  0

[5 rows x 38 columns]
```

1.11 DATA CLEANING

[14]:
```python
# Making a copy of the merged data set to retain an original copy.

merged_copy = merged_df.copy()
```

Checking for completeness of our data

[15]:
```python
# Check for missing values in our merged dataframe

missing_values = merged_df.isnull().mean()
missing_values
```

[15]:
```
respondent_id                  0.000000
h1n1_concern                   0.003445
h1n1_knowledge                 0.004343
behavioral_antiviral_meds      0.002658
behavioral_avoidance           0.007788
behavioral_face_mask           0.000711
behavioral_wash_hands          0.001573
behavioral_large_gatherings    0.003258
behavioral_outside_home        0.003070
behavioral_touch_face          0.004793
doctor_recc_h1n1               0.080878
doctor_recc_seasonal           0.080878
chronic_med_condition          0.036358
child_under_6_months           0.030704
health_worker                  0.030104
health_insurance               0.459580
opinion_h1n1_vacc_effective    0.014640
opinion_h1n1_risk              0.014528
opinion_h1n1_sick_from_vacc    0.014790
opinion_seas_vacc_effective    0.017299
opinion_seas_risk              0.019246
opinion_seas_sick_from_vacc    0.020107
age_group                      0.000000
education                      0.052683
race                           0.000000
sex                            0.000000
income_poverty                 0.165612
marital_status                 0.052720
rent_or_own                    0.076459
employment_status              0.054780
```

```
hhs_geo_region                  0.000000
census_msa                      0.000000
household_adults                0.009323
household_children              0.009323
employment_industry             0.499120
employment_occupation           0.504362
h1n1_vaccine                    0.000000
seasonal_vaccine                0.000000
dtype: float64
```

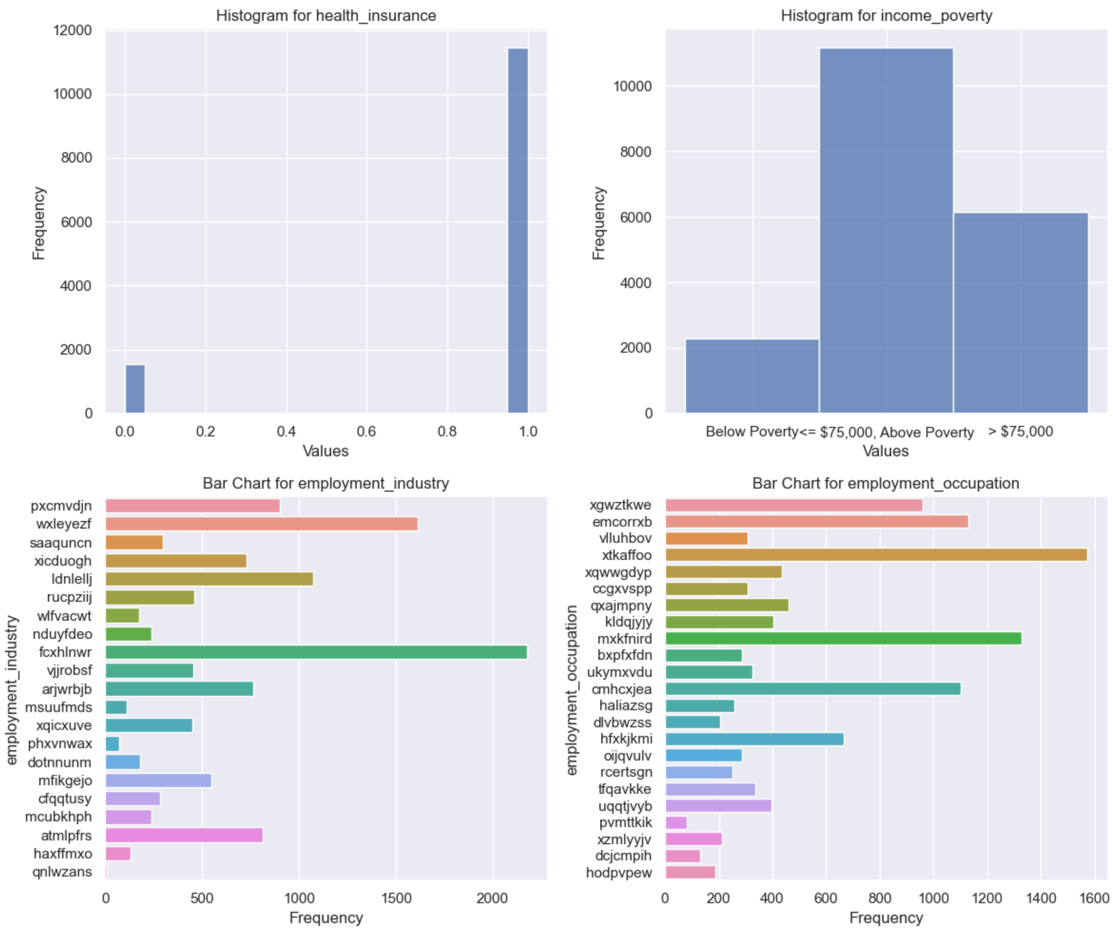
Handling missing values

[16]:
```python
# Drop missing values in columns with less than 10% missing values

# Set 10% ie 0.1 threshold
threshold = 0.1

# Select respective columns
below_threshold_columns = missing_values[missing_values <= threshold].index

# Drop null values in the selected columns
merged_df = merged_df.dropna(subset=below_threshold_columns)

# Confirm dropna
merged_df.isnull().mean()
```

[16]:
```
respondent_id                   0.000000
h1n1_concern                    0.000000
h1n1_knowledge                  0.000000
behavioral_antiviral_meds       0.000000
behavioral_avoidance            0.000000
behavioral_face_mask            0.000000
behavioral_wash_hands           0.000000
behavioral_large_gatherings     0.000000
behavioral_outside_home         0.000000
behavioral_touch_face           0.000000
doctor_recc_h1n1                0.000000
doctor_recc_seasonal            0.000000
chronic_med_condition           0.000000
child_under_6_months            0.000000
health_worker                   0.000000
health_insurance                0.400737
opinion_h1n1_vacc_effective     0.000000
opinion_h1n1_risk               0.000000
opinion_h1n1_sick_from_vacc     0.000000
opinion_seas_vacc_effective     0.000000
opinion_seas_risk               0.000000
```

```
opinion_seas_sick_from_vacc    0.000000
age_group                      0.000000
education                      0.000000
race                           0.000000
sex                            0.000000
income_poverty                 0.095256
marital_status                 0.000000
rent_or_own                    0.000000
employment_status              0.000000
hhs_geo_region                 0.000000
census_msa                     0.000000
household_adults               0.000000
household_children             0.000000
employment_industry            0.459834
employment_occupation          0.464440
h1n1_vaccine                   0.000000
seasonal_vaccine               0.000000
dtype: float64
```

[17]:
```python
# Visualize data in these columns before imputing

column_names =  ['health_insurance', 'income_poverty', 'employment_industry',␣
 ↪'employment_occupation']

# Set style
sns.set(style="darkgrid")

# Create subplots
fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(12, 10))
axes = axes.flatten()

# Plot histograms for all columns
for i, column in enumerate(column_names):
    if column in ['employment_industry', 'employment_occupation']:
        sns.countplot(y=column, data=merged_df, ax=axes[i])
        axes[i].set_title(f'Bar Chart for {column}')
        axes[i].set_ylabel(column)
        axes[i].set_xlabel('Frequency')
    else:
        sns.histplot(merged_df[column], bins=20, kde=False, ax=axes[i])
        axes[i].set_title(f'Histogram for {column}')
        axes[i].set_xlabel('Values')
        axes[i].set_ylabel('Frequency')

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



- The health_insurance column has a relatively high percentage of missing values (45.95%). From the visual we can see that most of the people in our dataset had health insurance and were vaccinated.
- In as much as health insurance is a significant factor in understanding healthcare-related decisions including vaccination decisions, imputing this column with the mode would be creating bias because we would be missing and imputing almost half the values. What if we impute with the binary value of 1 and majority of those people actually did not have health insurance?
- I will drop the column for now.
- I will also drop the 'employment_industry', 'employment_occupation', and 'hhs_geo_region' columns as the random characters cannot be interpreted and are therefore unnecessary.

[18]:
```
merged_df.describe()
```

[18]:

	respondent_id	h1n1_concern	h1n1_knowledge	behavioral_antiviral_meds \
count	21710.000000	21710.000000	21710.000000	21710.000000
mean	13358.714325	1.613450	1.289728	0.048503
std	7707.001900	0.895736	0.603993	0.214832

```
min        0.000000      0.000000        0.000000                          0.000000
25%     6703.250000      1.000000        1.000000                          0.000000
50%    13333.500000      2.000000        1.000000                          0.000000
75%    20042.750000      2.000000        2.000000                          0.000000
max    26706.000000      3.000000        2.000000                          1.000000

       behavioral_avoidance  behavioral_face_mask  behavioral_wash_hands  \
count          21710.000000          21710.000000           21710.000000
mean               0.734961              0.067526               0.832381
std                0.441364              0.250937               0.373536
min                0.000000              0.000000               0.000000
25%                0.000000              0.000000               1.000000
50%                1.000000              0.000000               1.000000
75%                1.000000              0.000000               1.000000
max                1.000000              1.000000               1.000000

       behavioral_large_gatherings  behavioral_outside_home  \
count                 21710.000000             21710.000000
mean                      0.356656                 0.334592
std                       0.479023                 0.471859
min                       0.000000                 0.000000
25%                       0.000000                 0.000000
50%                       0.000000                 0.000000
75%                       1.000000                 1.000000
max                       1.000000                 1.000000

       behavioral_touch_face  …  opinion_h1n1_vacc_effective  \
count           21710.000000  …                 21710.000000
mean                0.684892  …                     3.897743
std                 0.464570  …                     0.992184
min                 0.000000  …                     1.000000
25%                 0.000000  …                     3.000000
50%                 1.000000  …                     4.000000
75%                 1.000000  …                     5.000000
max                 1.000000  …                     5.000000

       opinion_h1n1_risk  opinion_h1n1_sick_from_vacc  \
count       21710.000000                 21710.000000
mean            2.343206                     2.347444
std             1.285311                     1.356257
min             1.000000                     1.000000
25%             1.000000                     1.000000
50%             2.000000                     2.000000
75%             4.000000                     4.000000
max             5.000000                     5.000000

       opinion_seas_vacc_effective  opinion_seas_risk  \
```

```
count                  21710.000000       21710.000000
mean                       4.051958           2.739613
std                        1.068120           1.387962
min                        1.000000           1.000000
25%                        4.000000           2.000000
50%                        4.000000           2.000000
75%                        5.000000           4.000000
max                        5.000000           5.000000

       opinion_seas_sick_from_vacc  household_adults  household_children  \
count                  21710.000000      21710.000000        21710.000000
mean                       2.107969          0.899816            0.531322
std                        1.327120          0.753242            0.925185
min                        1.000000          0.000000            0.000000
25%                        1.000000          0.000000            0.000000
50%                        2.000000          1.000000            0.000000
75%                        3.000000          1.000000            1.000000
max                        5.000000          3.000000            3.000000

       h1n1_vaccine  seasonal_vaccine
count  21710.000000      21710.000000
mean       0.226716          0.479272
std        0.418717          0.499582
min        0.000000          0.000000
25%        0.000000          0.000000
50%        0.000000          0.000000
75%        0.000000          1.000000
max        1.000000          1.000000

[8 rows x 26 columns]
```

[19]:
```
# Checking value counts for income_poverty column

value_counts = merged_df['income_poverty'].value_counts()
value_counts
```

[19]:
```
income_poverty
<= $75,000, Above Poverty    11185
> $75,000                     6159
Below Poverty                 2298
Name: count, dtype: int64
```

[20]:
```
#Initialize the imputer with the strategy 'most_frequent'
imputer = SimpleImputer(strategy='most_frequent')

# Impute income_poverty with the most frequent value
```

```python
merged_df['income_poverty'] = imputer.
  ↪fit_transform(merged_df[['income_poverty']]).ravel()

# Confirm that there are no missing values
missing_values = merged_df['income_poverty'].isna().mean()
missing_values
```

[20]: 0.0

[21]:
```python
merged_df.isna().sum()
```

[21]:
```
respondent_id                      0
h1n1_concern                       0
h1n1_knowledge                     0
behavioral_antiviral_meds          0
behavioral_avoidance               0
behavioral_face_mask               0
behavioral_wash_hands              0
behavioral_large_gatherings        0
behavioral_outside_home            0
behavioral_touch_face              0
doctor_recc_h1n1                   0
doctor_recc_seasonal               0
chronic_med_condition              0
child_under_6_months               0
health_worker                      0
health_insurance                8700
opinion_h1n1_vacc_effective        0
opinion_h1n1_risk                  0
opinion_h1n1_sick_from_vacc        0
opinion_seas_vacc_effective        0
opinion_seas_risk                  0
opinion_seas_sick_from_vacc        0
age_group                          0
education                          0
race                               0
sex                                0
income_poverty                     0
marital_status                     0
rent_or_own                        0
employment_status                  0
hhs_geo_region                     0
census_msa                         0
household_adults                   0
household_children                 0
employment_industry             9983
employment_occupation          10083
```

```
h1n1_vaccine                      0
seasonal_vaccine                  0
dtype: int64
```

[22]:
```
# Columns to get value counts for
columns = ['h1n1_vaccine', 'seasonal_vaccine']

# Loop through each column and print value counts
for column in columns:
    value_counts = merged_df[column].value_counts()
    print(f"Value counts for '{column}':")
    print(value_counts)
    print("\n")  # Adds a new line for better readability between outputs
```

```
Value counts for 'h1n1_vaccine':
h1n1_vaccine
0    16788
1     4922
Name: count, dtype: int64


Value counts for 'seasonal_vaccine':
seasonal_vaccine
0    11305
1    10405
Name: count, dtype: int64
```

- From the value counts above, we can see that the `h1n1 vaccine` has a much lower uptake than the `seasonal flu vaccine`. This observation led to my decision to drop the `seasonal flu vaccine` data and focus on investigating the reasons behind the low uptake of the `h1n1 vaccine`.

[23]:
```
# Dropping columns not necessary to our analysis

df = merged_df.drop(columns=['employment_industry' , 'employment_occupation' ,␣
↪'health_insurance' , 'hhs_geo_region' , 'household_adults' ,
                             'household_children', 'marital_status' ,␣
↪'rent_or_own' , 'seasonal_vaccine' , 'doctor_recc_seasonal' ,
                             'opinion_seas_vacc_effective','opinion_seas_risk',␣
↪'opinion_seas_sick_from_vacc' ])
```

[24]:
```
df.isna().sum()
```

[24]:
```
respondent_id                 0
h1n1_concern                  0
h1n1_knowledge                0
```

```
behavioral_antiviral_meds      0
behavioral_avoidance           0
behavioral_face_mask           0
behavioral_wash_hands          0
behavioral_large_gatherings    0
behavioral_outside_home        0
behavioral_touch_face          0
doctor_recc_h1n1               0
chronic_med_condition          0
child_under_6_months           0
health_worker                  0
opinion_h1n1_vacc_effective    0
opinion_h1n1_risk              0
opinion_h1n1_sick_from_vacc    0
age_group                      0
education                      0
race                           0
sex                            0
income_poverty                 0
employment_status              0
census_msa                     0
h1n1_vaccine                   0
dtype: int64
```

- Our data now has no missing values.

Checking for duplicates

[25]:
```python
# Checking for duplicates in df

duplicates = df[df.duplicated()]

if duplicates.empty:
 print("No duplicates found.")
else:
 print("Duplicates found.")
 print(duplicates)
```

```
No duplicates found.
```

[26]:
```python
# Checking for duplicates using the unique identifier column

df[df.duplicated(subset=["respondent_id"])]
```

[26]:
```
Empty DataFrame
Columns: [respondent_id, h1n1_concern, h1n1_knowledge,
behavioral_antiviral_meds, behavioral_avoidance, behavioral_face_mask,
behavioral_wash_hands, behavioral_large_gatherings, behavioral_outside_home,
```

```
behavioral_touch_face, doctor_recc_h1n1, chronic_med_condition,
child_under_6_months, health_worker, opinion_h1n1_vacc_effective,
opinion_h1n1_risk, opinion_h1n1_sick_from_vacc, age_group, education, race, sex,
income_poverty, employment_status, census_msa, h1n1_vaccine]
Index: []

[0 rows x 25 columns]
```

- Our dataframe does not have duplicates.

Checking for placeholders

[27]:
```
potential_placeholders = [" " , "-", "--", "?", "??" , "#","#####" , "-1" ,␣
↪"9999", "999" , "unknown", "missing", "na" , "n/a" , "Nan"]

# Loop through each column and check for potential placeholders
found_placeholder = False
for column in df.columns:
    unique_values = df[column].unique()
    for value in unique_values:
        if pd.isna(value) or (isinstance(value, str) and value.strip().lower()␣
↪in potential_placeholders):
            count = (df[column] == value).sum()
            print(f"Column '{column}': Found {count} occurrences of potential␣
↪placeholder '{value}'")
            found_placeholder = True
    if not found_placeholder:
        print("No potential placeholders found in the DataFrame.")
```

```
No potential placeholders found in the DataFrame.
No potential placeholders found in the DataFrame.
No potential placeholders found in the DataFrame.
No potential placeholders found in the DataFrame.
No potential placeholders found in the DataFrame.
No potential placeholders found in the DataFrame.
No potential placeholders found in the DataFrame.
No potential placeholders found in the DataFrame.
No potential placeholders found in the DataFrame.
No potential placeholders found in the DataFrame.
No potential placeholders found in the DataFrame.
No potential placeholders found in the DataFrame.
No potential placeholders found in the DataFrame.
No potential placeholders found in the DataFrame.
No potential placeholders found in the DataFrame.
No potential placeholders found in the DataFrame.
No potential placeholders found in the DataFrame.
No potential placeholders found in the DataFrame.
No potential placeholders found in the DataFrame.
```

```
No potential placeholders found in the DataFrame.
No potential placeholders found in the DataFrame.
No potential placeholders found in the DataFrame.
No potential placeholders found in the DataFrame.
No potential placeholders found in the DataFrame.
No potential placeholders found in the DataFrame.
```

Checking for outliers

- To facilitate outlier detection and potential handling, I'll divide my dataframe into two: one containing numerical columns and the other containing categorical columns.

[28]:
```python
# Preserve our unique identifier column
respondent_id = 'respondent_id'

# Columns to exclude
columns_to_exclude = ['age_group', 'education', 'race', 'sex',␣
↪'income_poverty', 'employment_status', 'census_msa']

# Columns to include (remaining columns + respondent_id)
columns_to_include = [col for col in df.columns if col not in␣
↪columns_to_exclude]

# Create a new dataset df_numerical
df_numerical = df[columns_to_include]

# Creating df_categorical with the excluded columns + respondent_id
df_categorical = df[[respondent_id] + columns_to_exclude]

print("Numerical DataFrame:")
df_numerical.head()
```

```
Numerical DataFrame:
```

[28]:
```
   respondent_id  h1n1_concern  h1n1_knowledge  behavioral_antiviral_meds  \
0              0           1.0             0.0                        0.0
1              1           3.0             2.0                        0.0
3              3           1.0             1.0                        0.0
4              4           2.0             1.0                        0.0
5              5           3.0             1.0                        0.0

   behavioral_avoidance  behavioral_face_mask  behavioral_wash_hands  \
0                   0.0                   0.0                    0.0
1                   1.0                   0.0                    1.0
3                   1.0                   0.0                    1.0
4                   1.0                   0.0                    1.0
5                   1.0                   0.0                    1.0
```

```
   behavioral_large_gatherings  behavioral_outside_home  \
0                          0.0                      1.0
1                          0.0                      1.0
3                          1.0                      0.0
4                          1.0                      0.0
5                          0.0                      0.0

   behavioral_touch_face  doctor_recc_h1n1  chronic_med_condition  \
0                    1.0               0.0                    0.0
1                    1.0               0.0                    0.0
3                    0.0               0.0                    1.0
4                    1.0               0.0                    0.0
5                    1.0               0.0                    0.0

   child_under_6_months  health_worker  opinion_h1n1_vacc_effective  \
0                   0.0            0.0                          3.0
1                   0.0            0.0                          5.0
3                   0.0            0.0                          3.0
4                   0.0            0.0                          3.0
5                   0.0            0.0                          5.0

   opinion_h1n1_risk  opinion_h1n1_sick_from_vacc  h1n1_vaccine
0                1.0                          2.0             0
1                4.0                          4.0             0
3                3.0                          5.0             0
4                3.0                          2.0             0
5                2.0                          1.0             0
```

[29]:

```python
print("Categorical DataFrame:")
df_categorical.head()
```

```
Categorical DataFrame:
```

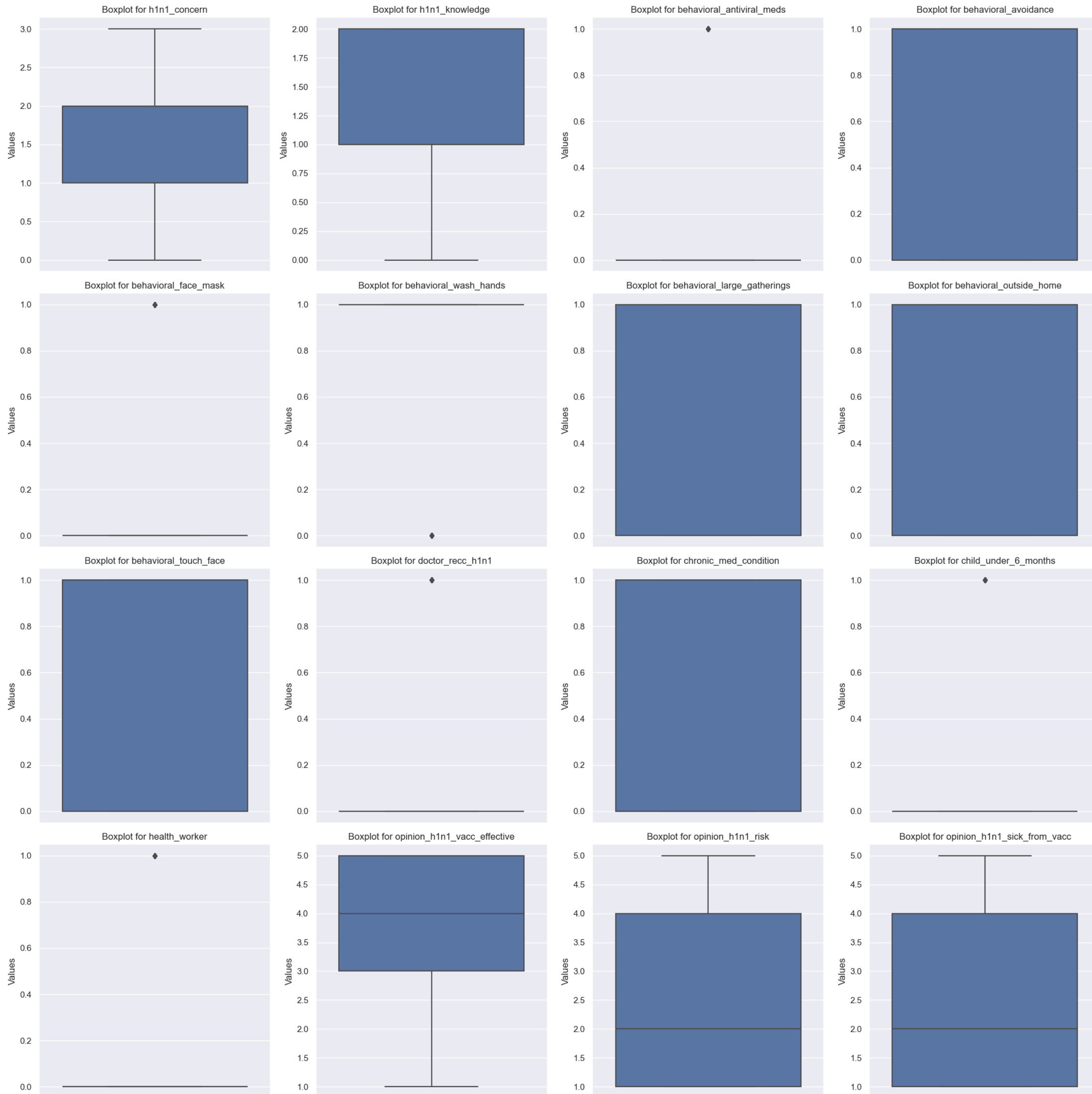
[29]:

```
   respondent_id      age_group     education   race     sex  \
0              0  55 - 64 Years      < 12 Years  White  Female
1              1  35 - 44 Years        12 Years  White    Male
3              3      65+ Years        12 Years  White  Female
4              4  45 - 54 Years  Some College  White  Female
5              5      65+ Years        12 Years  White    Male

              income_poverty   employment_status               census_msa
0              Below Poverty  Not in Labor Force                  Non-MSA
1              Below Poverty            Employed  MSA, Not Principle  City
3              Below Poverty  Not in Labor Force       MSA, Principle City
4  <= $75,000, Above Poverty            Employed  MSA, Not Principle  City
5  <= $75,000, Above Poverty            Employed       MSA, Principle City
```

[30]:
```
# Checking for outliers using IQR
# Loop through numerical columns except 'respondent_id'
for column in df_numerical.columns:
    if column != 'respondent_id':
        # Calculate IQR
        q1 = df_numerical[column].quantile(0.25)
        q3 = df_numerical[column].quantile(0.75)
        iqr = q3 - q1

        # Calculate outlier boundaries
        lower_bound = q1 - 1.5 * iqr
        upper_bound = q3 + 1.5 * iqr

# Count outliers
num_outliers = ((df_numerical[column] < lower_bound) | (df_numerical[column] >␣
 ↪upper_bound)).sum()

# Print the result
print(f"Column: {column}, Number of outliers: {num_outliers}")
```

```
Column: h1n1_vaccine, Number of outliers: 4922
```

[31]:
```
# Select numerical feature variables excluding 'respondent_id'
numerical_columns = df_numerical.drop(columns=['respondent_id']).
 ↪select_dtypes(include=['int64', 'float64']).columns

# Create subplots
fig, axes = plt.subplots(nrows=4, ncols=4, figsize=(20, 20))
axes = axes.flatten()

# Plot boxplots for numerical feature variables excluding 'respondent_id'
for i, column in enumerate(numerical_columns):
    if i < len(axes):  # Check if there are more plots than columns
        sns.boxplot(y=df_numerical[column], ax=axes[i])
        axes[i].set_title(f'Boxplot for {column}')
        axes[i].set_ylabel('Values')

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



[32]:
```python
for column in df_numerical:
    value_counts = df_numerical[column].value_counts()
    print(f"Value counts for {column}:")
    print(value_counts)
    print()
```

```
Value counts for respondent_id:
respondent_id
0        1
17782    1
17790    1
17789    1
17788    1
```

```
        ..
8937     1
8936     1
8935     1
8933     1
26706    1
Name: count, Length: 21710, dtype: int64

Value counts for h1n1_concern:
h1n1_concern
2.0    8731
1.0    6844
3.0    3574
0.0    2561
Name: count, dtype: int64

Value counts for h1n1_knowledge:
h1n1_knowledge
1.0    11968
2.0     8016
0.0     1726
Name: count, dtype: int64

Value counts for behavioral_antiviral_meds:
behavioral_antiviral_meds
0.0    20657
1.0     1053
Name: count, dtype: int64

Value counts for behavioral_avoidance:
behavioral_avoidance
1.0    15956
0.0     5754
Name: count, dtype: int64

Value counts for behavioral_face_mask:
behavioral_face_mask
0.0    20244
1.0     1466
Name: count, dtype: int64

Value counts for behavioral_wash_hands:
behavioral_wash_hands
1.0    18071
0.0     3639
Name: count, dtype: int64

Value counts for behavioral_large_gatherings:
```

```
behavioral_large_gatherings
0.0    13967
1.0     7743
Name: count, dtype: int64

Value counts for behavioral_outside_home:
behavioral_outside_home
0.0    14446
1.0     7264
Name: count, dtype: int64

Value counts for behavioral_touch_face:
behavioral_touch_face
1.0    14869
0.0     6841
Name: count, dtype: int64

Value counts for doctor_recc_h1n1:
doctor_recc_h1n1
0.0    16865
1.0     4845
Name: count, dtype: int64

Value counts for chronic_med_condition:
chronic_med_condition
0.0    15493
1.0     6217
Name: count, dtype: int64

Value counts for child_under_6_months:
child_under_6_months
0.0    19902
1.0     1808
Name: count, dtype: int64

Value counts for health_worker:
health_worker
0.0    19210
1.0     2500
Name: count, dtype: int64

Value counts for opinion_h1n1_vacc_effective:
opinion_h1n1_vacc_effective
4.0    9955
5.0    6187
3.0    3392
2.0    1513
1.0     663
```

```
Name: count, dtype: int64

Value counts for opinion_h1n1_risk:
opinion_h1n1_risk
2.0    8392
1.0    6637
4.0    4525
5.0    1441
3.0     715
Name: count, dtype: int64

Value counts for opinion_h1n1_sick_from_vacc:
opinion_h1n1_sick_from_vacc
2.0    7643
1.0    7424
4.0    4810
5.0    1757
3.0      76
Name: count, dtype: int64

Value counts for h1n1_vaccine:
h1n1_vaccine
0    16788
1     4922
Name: count, dtype: int64
```

- Due to the nature of our dataset, I will not handle outliers.

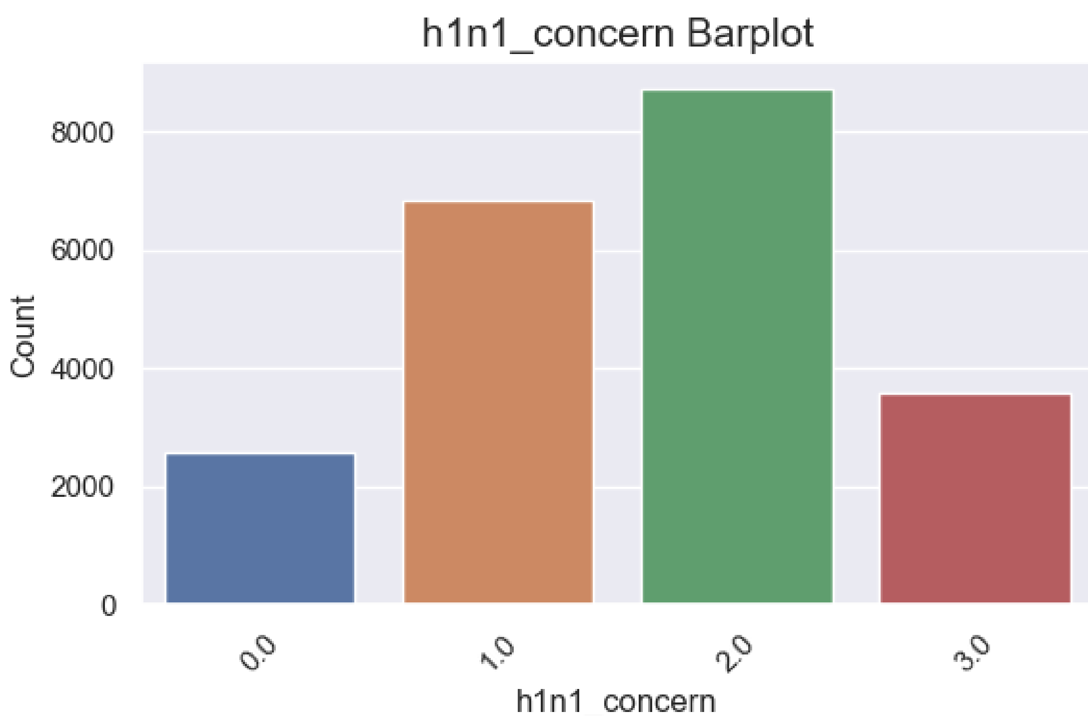
[33]:
```
# Set the 'respondent_id' column as the index in df_numerical

df_numerical.set_index('respondent_id', inplace=True)

# Set the 'respondent_id' column as the index in df_categorical\

df_categorical.set_index('respondent_id', inplace=True)
```

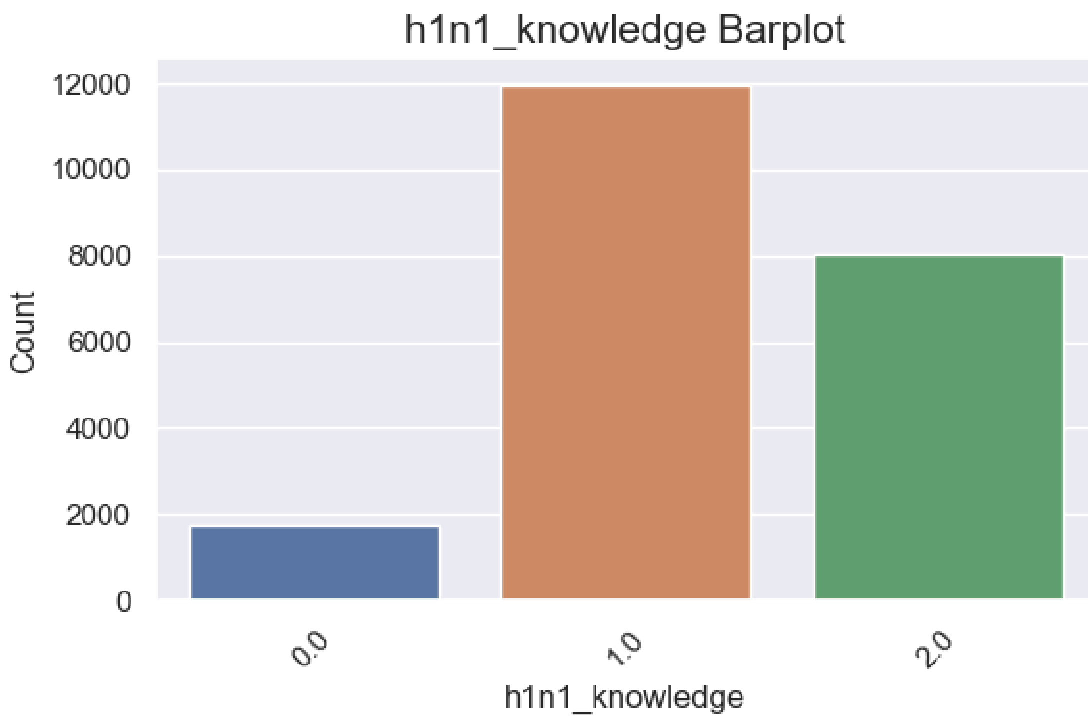
1.12 EXPLORATORY DATA ANALYSIS

Univariate analysis

[34]:
```
# Columns to create bar plots
columns_to_plot = ['h1n1_concern', 'h1n1_knowledge']

# Loop through each column in the list
for col in columns_to_plot:
    plt.figure(figsize=(6, 4))

    # Grid style
```

```
    sns.set_style("darkgrid")

    # Create the bar plot
    sns.barplot(x=df_numerical[col].value_counts().index, y=df_numerical[col].
↪value_counts())

    plt.title(f'{col} Barplot', fontsize=15)
    plt.xlabel(col, fontsize=12)
    plt.ylabel('Count', fontsize=12)
    plt.xticks(rotation=45)

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

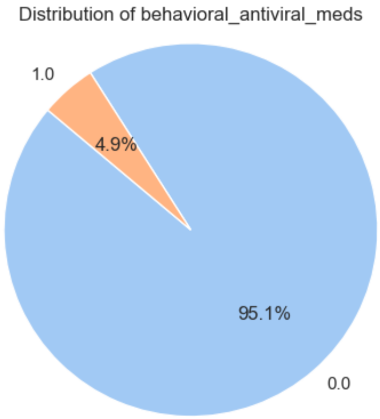


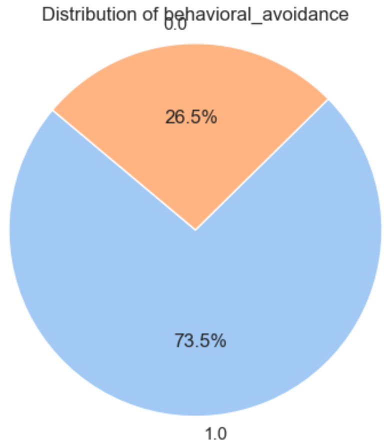


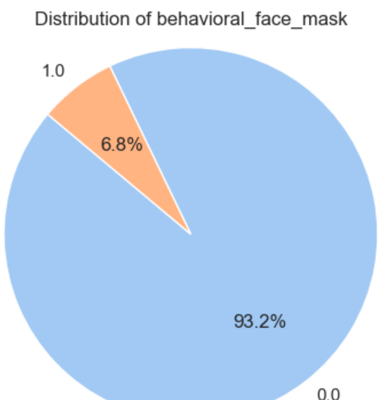
- From the bar plots above, it is evident that the majority of people expressed a moderate level of concern regarding the h1n1 vaccine and had limited knowledge about it.

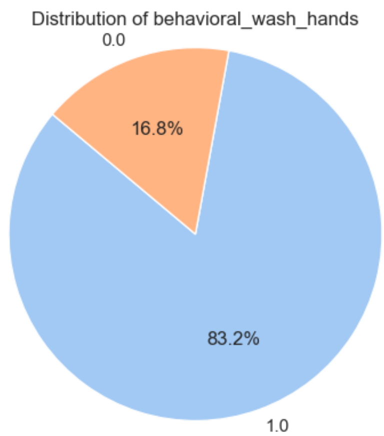
[35]:
```
# Define the columns to analyze
columns_to_analyze = [
    'behavioral_antiviral_meds',
    'behavioral_avoidance',
    'behavioral_face_mask',
    'behavioral_wash_hands',
    'behavioral_large_gatherings',
    'behavioral_outside_home',
    'behavioral_touch_face'
]

# Calculate number of rows needed for subplots
num_columns = 2
num_rows = (len(columns_to_analyze) + num_columns - 1) // num_columns

# Create subplots
fig, axes = plt.subplots(nrows=num_rows, ncols=num_columns, figsize=(10,␣
 ↪4*num_rows))

# Flatten the axes array for easy indexing
axes = axes.flatten()
```

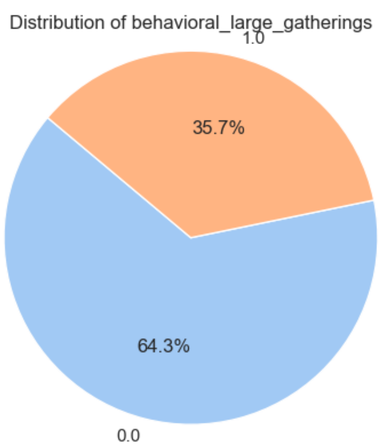
```
# Set colors
colors = sns.color_palette("pastel")

# Loop through each column and create a pie chart
for i, column in enumerate(columns_to_analyze):
    labels = df_numerical[column].value_counts().index
    sizes = df_numerical[column].value_counts().values
    axes[i].pie(sizes, labels=labels, autopct='%1.1f%%', startangle=140,␣
 ↪colors=colors)
    axes[i].set_title(f'Distribution of {column}', fontsize=12)  # Set the␣
 ↪title of the plot
    axes[i].axis('equal')  # Equal aspect ratio ensures that pie is drawn as a␣
 ↪circle

# Hide any unused subplots
for j in range(i + 1, len(axes)):
    fig.delaxes(axes[j])

# Adjust layout
plt.tight_layout()
plt.show()
```

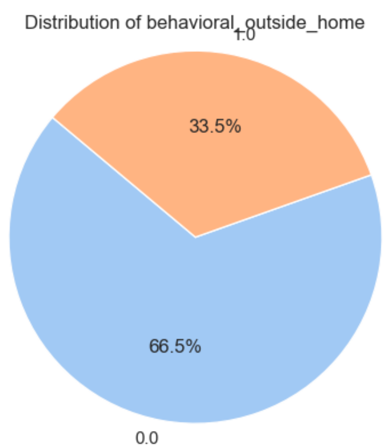

Distribution of behavioral_antiviral_meds
1.0
4.9%
95.1%
0.0

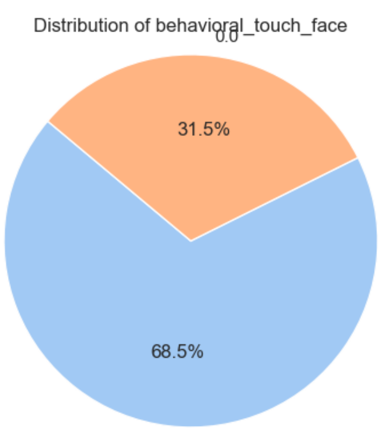

Distribution of behavioral_avoidance
0.0
26.5%
73.5%
1.0


Distribution of behavioral_face_mask
1.0
6.8%
93.2%
0.0


Distribution of behavioral_wash_hands
0.0
16.8%
83.2%
1.0


Distribution of behavioral_large_gatherings
1.0
35.7%
64.3%
0.0

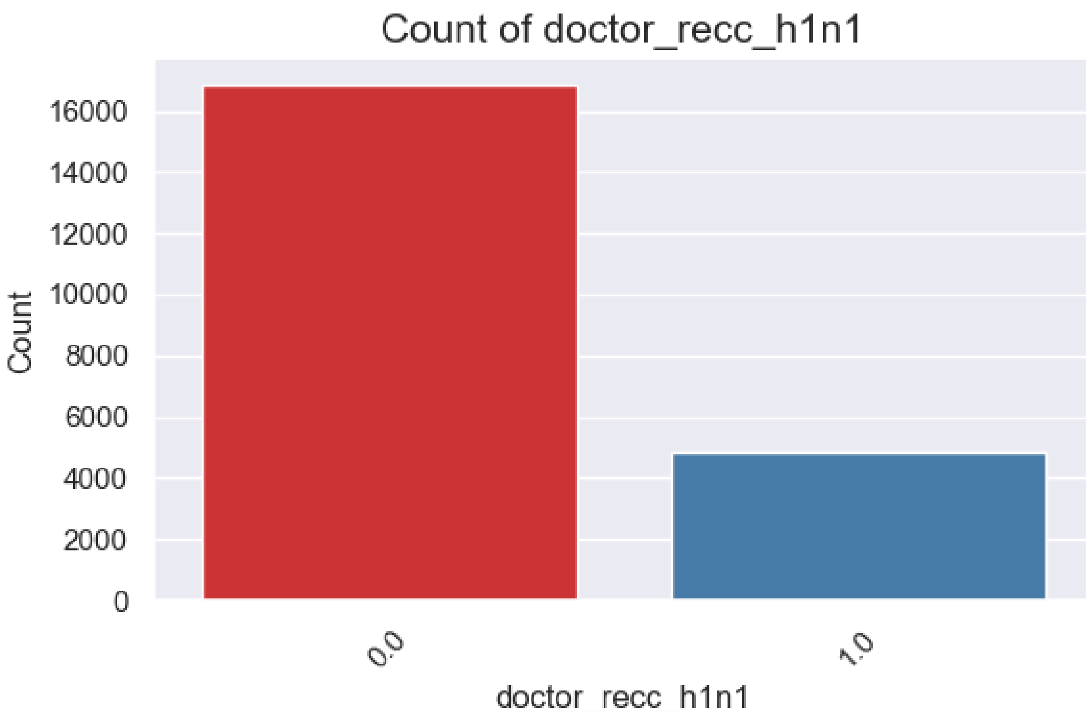

Distribution of behavioral_outside_home
1.0
33.5%
66.5%
0.0

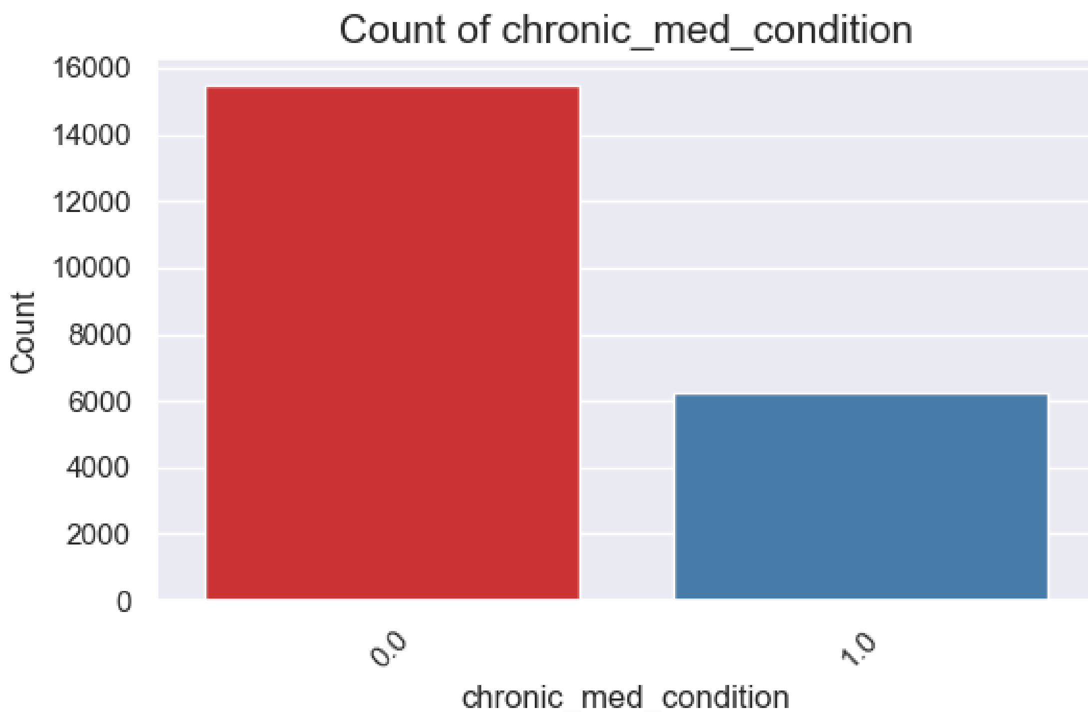

Distribution of behavioral_touch_face
0.0
31.5%
68.5%
1.0

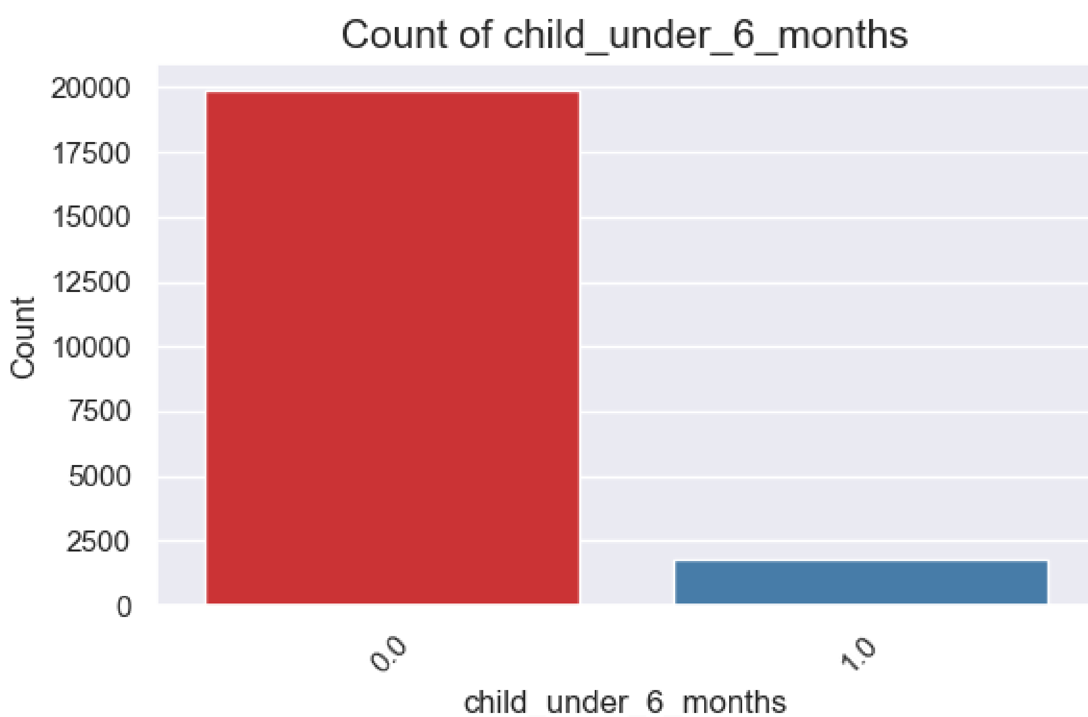
- The individuals in our dataset exhibited below-average behavioral characteristics in terms of taking antiviral medication, avoiding close contact with people showing flu-like symptoms, purchasing face masks, frequently washing and sanitizing hands, reducing time spent at large gatherings, minimizing contact with people outside their own household, and avoiding touching their faces.

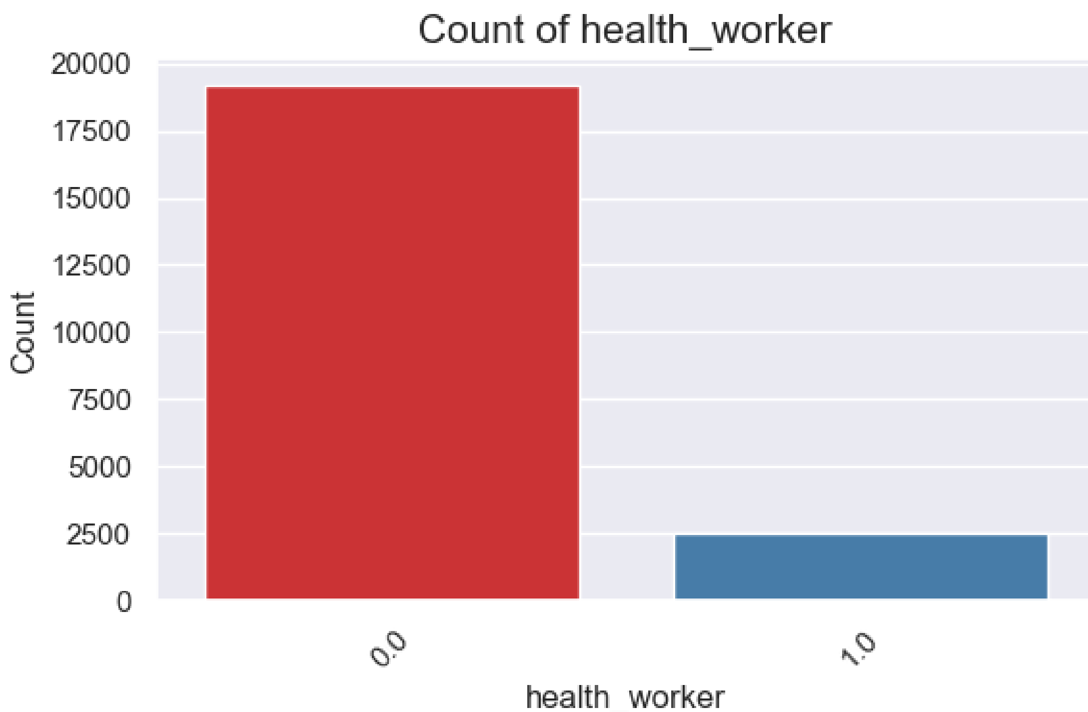
[36]:
```python
# Define columns to analyze
categorical_columns = [
    'doctor_recc_h1n1',
    'chronic_med_condition',
    'child_under_6_months',
    'health_worker'
]

ordinal_columns = [
    'opinion_h1n1_vacc_effective',
    'opinion_h1n1_risk',
    'opinion_h1n1_sick_from_vacc'
]

# Set style for plots
sns.set_style("darkgrid")

# Function to plot categorical variables
def plot_categorical(col):
    plt.figure(figsize=(6, 4))
    sns.countplot(x=col, data=df_numerical, palette="Set1")
    plt.title(f'Count of {col}', fontsize=15)
    plt.xlabel(col, fontsize=12)
    plt.ylabel('Count', fontsize=12)
    plt.xticks(rotation=45)
    plt.tight_layout()
    plt.show()

# Function to plot ordinal variables
def plot_ordinal(col):
    plt.figure(figsize=(6, 4))
    values = df_numerical[col].value_counts().sort_index()
    plt.bar(values.index, values, color='lightblue')
    plt.title(f'Count of {col}', fontsize=15)
    plt.xlabel(col, fontsize=12)
    plt.ylabel('Count', fontsize=12)
    plt.xticks(rotation=45)
    plt.tight_layout()
    plt.show()
```

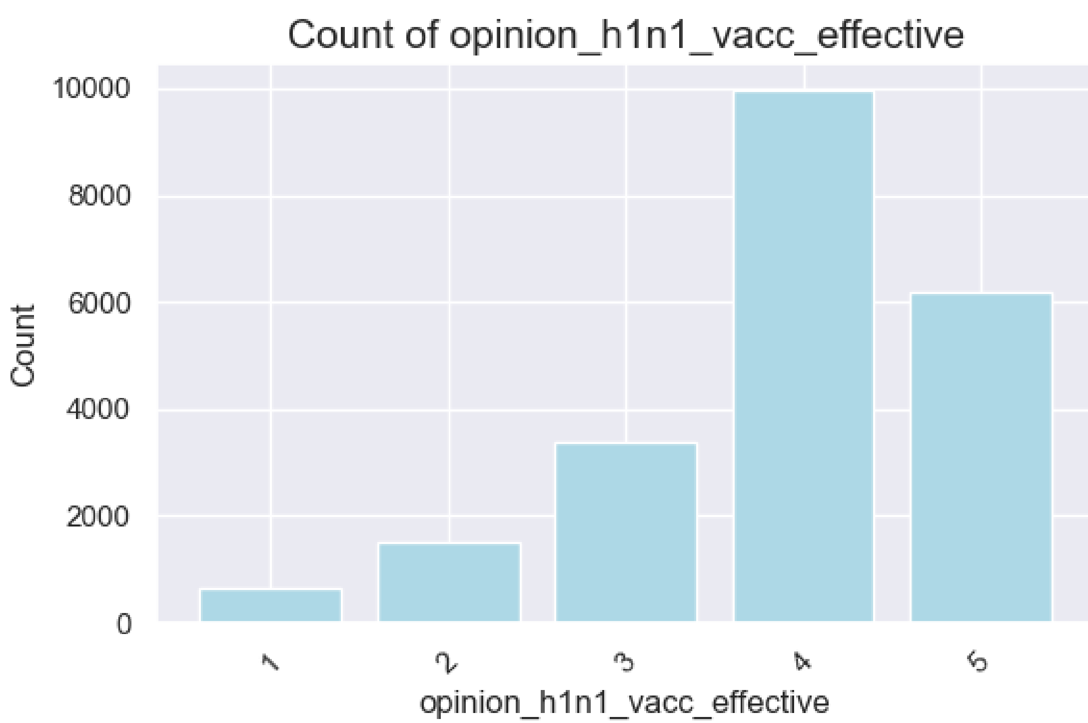
```python
# Plot categorical variables
for col in categorical_columns:
    plot_categorical(col)

# Plot ordinal variables
for col in ordinal_columns:
    plot_ordinal(col)
```

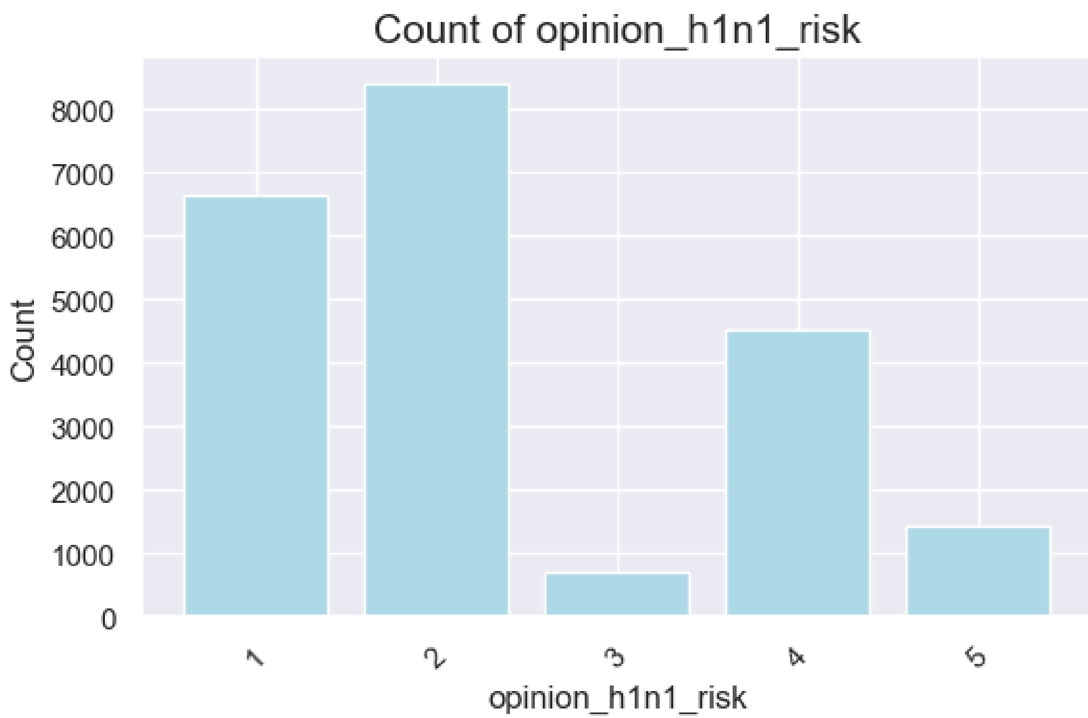


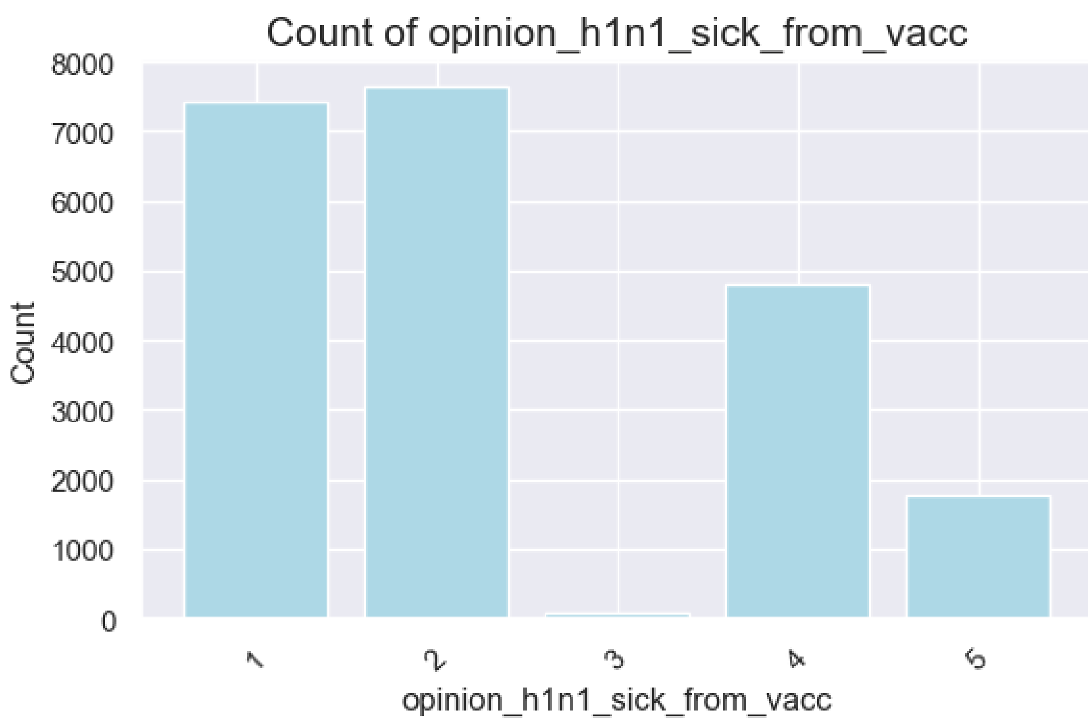

Count of chronic_med_condition
16000
14000
12000
10000
8000
6000
4000
2000
0
Count
0.0
1.0
chronic_med_condition


Count of child_under_6_months
20000
17500
15000
12500
10000
7500
5000
2500
0
Count
0.0
1.0
child_under_6_months


Count of health_worker
Count
20000
17500
15000
12500
10000
7500
5000
2500
0
0.0
1.0
health_worker


Count of opinion_h1n1_vacc_effective
Count
10000
8000
6000
4000
2000
0
1
2
3
4
5
opinion_h1n1_vacc_effective



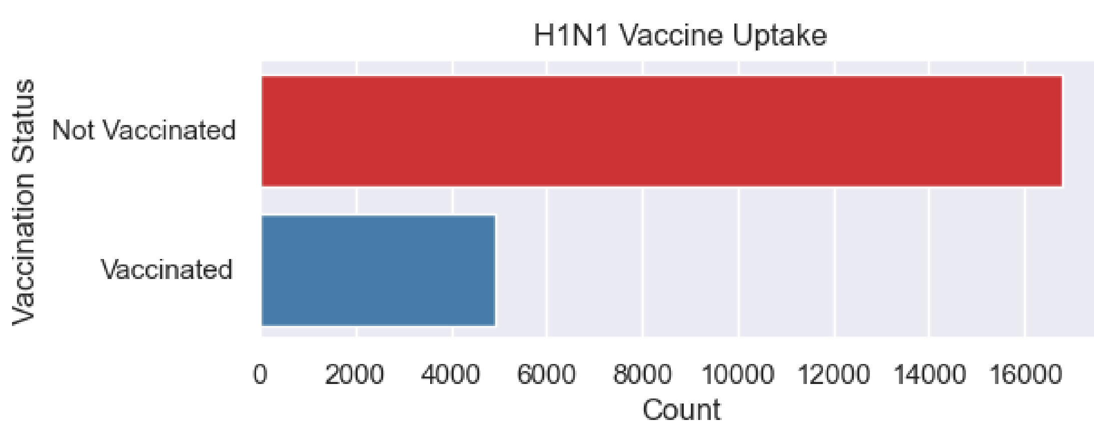


- A majority of the respondents did not receive a recommendation from their doctor to get the

H1N1 vaccine.

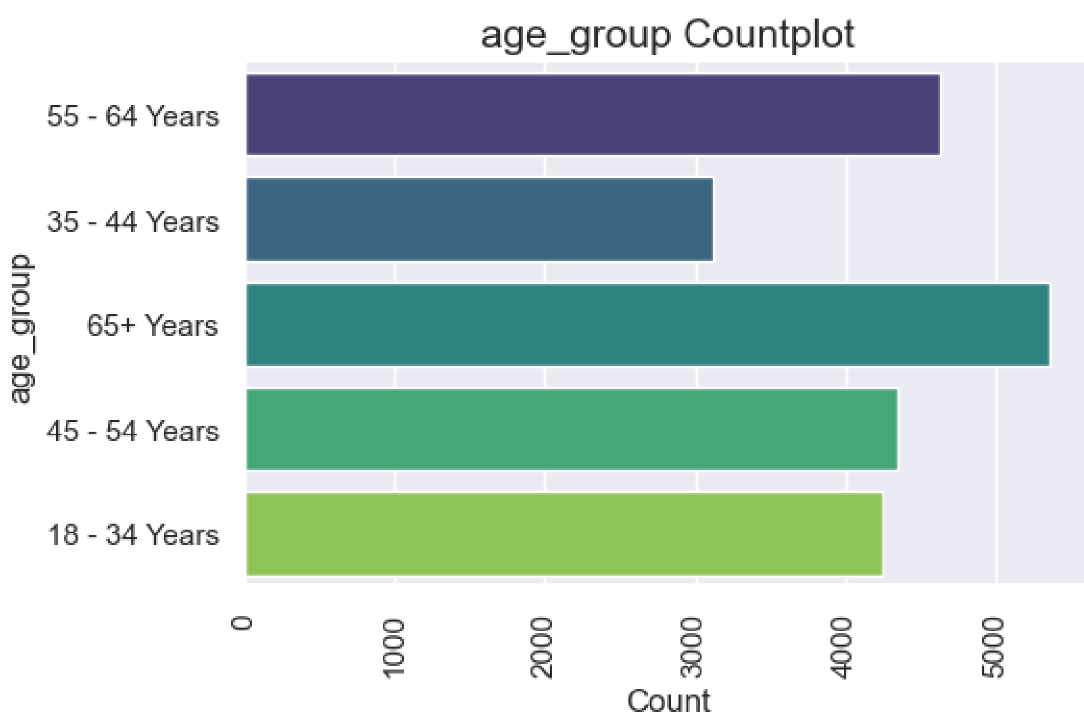
- Most respondents do not have a chronic medical condition. However, a substantial minority do, which is an important factor to consider for public health interventions.
- The vast majority of respondents do not have a child under 6 months old, indicating that this specific demographic concern is relevant to a small portion of the population.
- Most respondents are not health workers, which may affect their exposure to health-related information and practices.
- The majority of respondents believe that the H1N1 vaccine is effective indicating a general positive perception of the vaccine's effectiveness.
- A significant portion of respondents perceive the risk of H1N1 as low to moderate, suggesting a relatively low level of concern about contracting H1N1.
- A majority of respondents do not believe that they will get sick from the H1N1 vaccine, indicating relatively high confidence in the vaccine's safety. However, some have concerns.

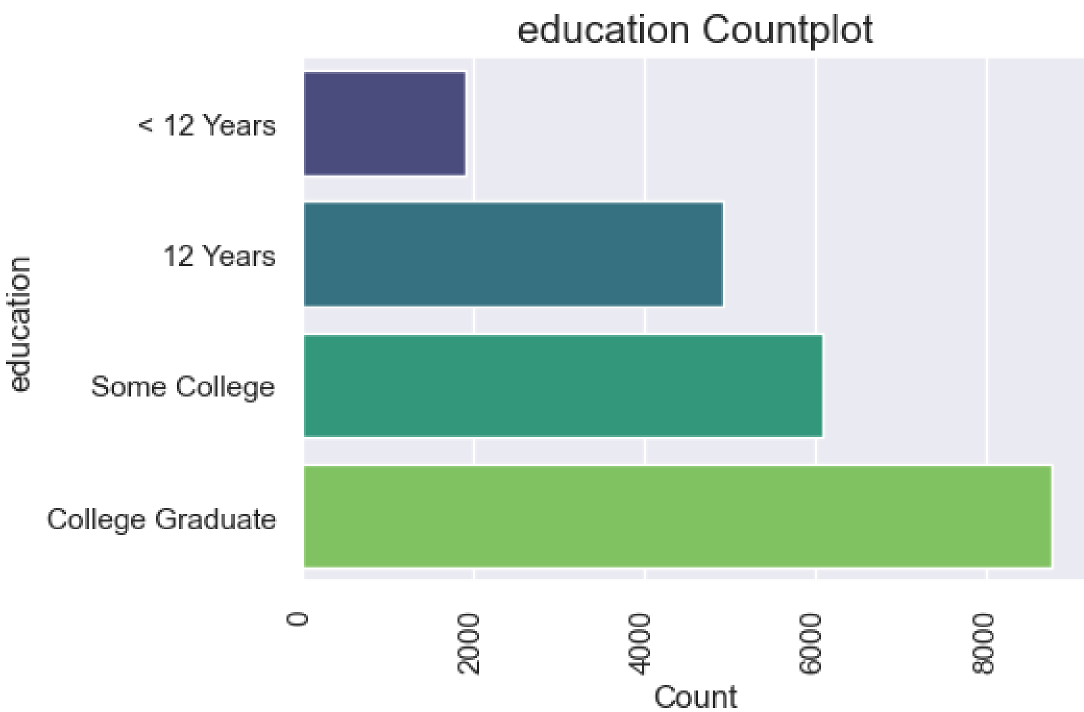
[37]:
```
import seaborn as sns
import matplotlib.pyplot as plt

# Data for h1n1_vaccine
h1n1_vaccine_data = {'Status': ['Not Vaccinated', 'Vaccinated'], 'Count':␣
↪[16788, 4922]}

# Convert to DataFrame
h1n1_vaccine_df = pd.DataFrame(h1n1_vaccine_data)

# Create a horizontal bar plot
plt.figure(figsize=(6, 2))
sns.barplot(x='Count', y='Status', data=h1n1_vaccine_df, palette='Set1')
plt.title('H1N1 Vaccine Uptake')
plt.xlabel('Count')
plt.ylabel('Vaccination Status')
plt.show()
```

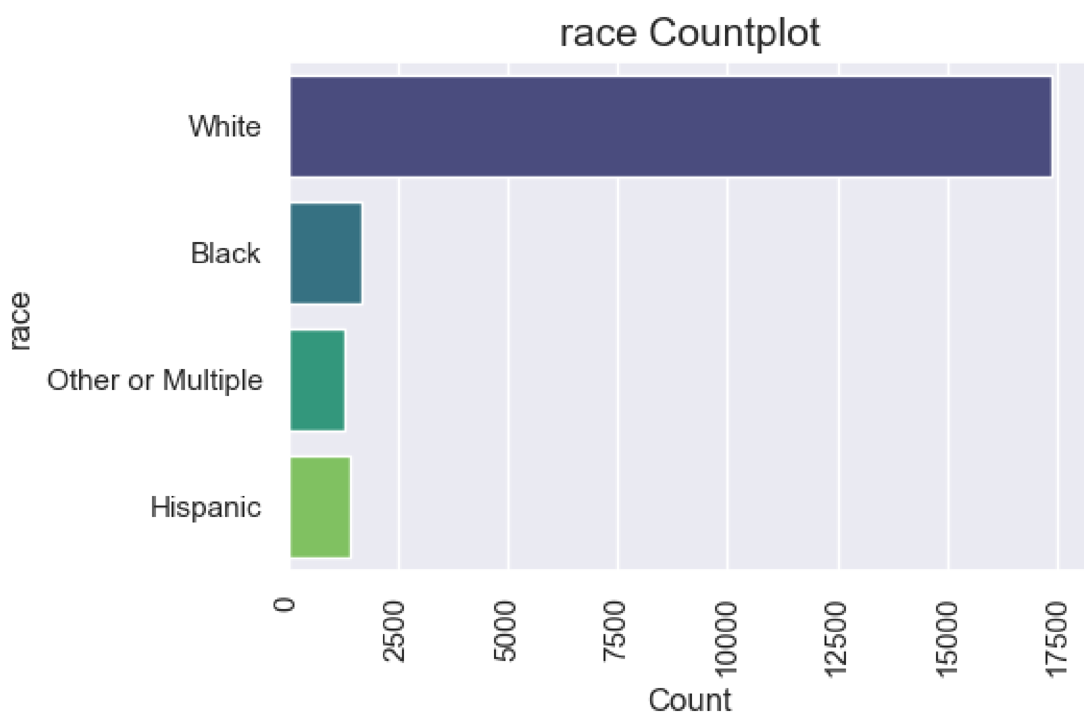


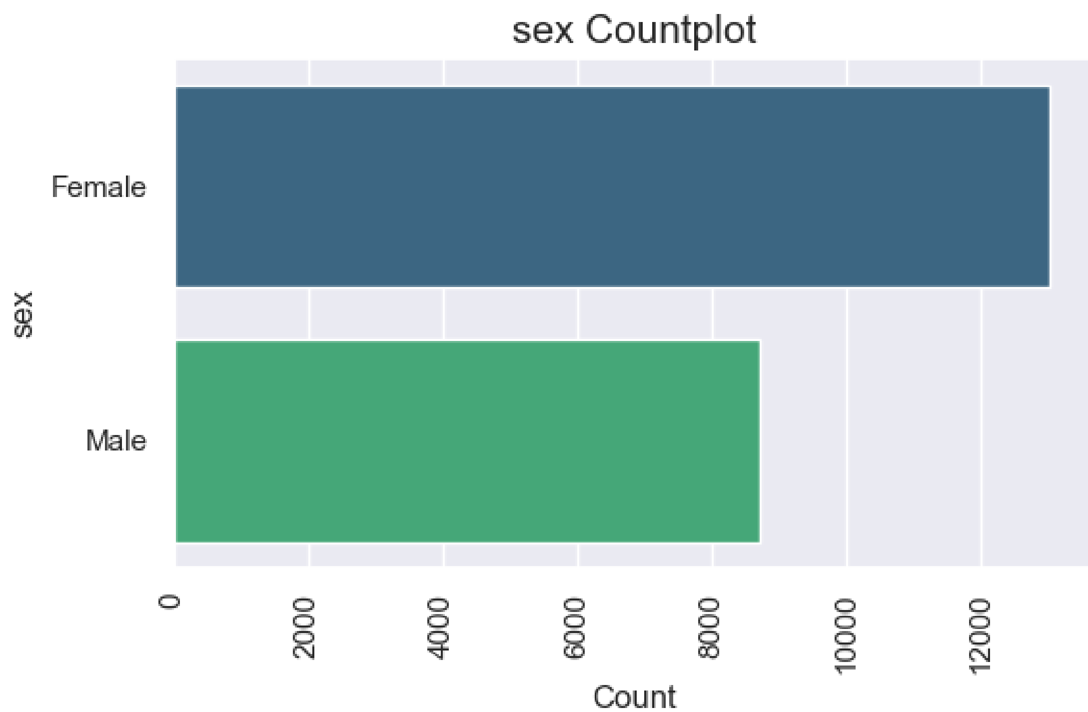
- The majority of respondents did not receive the H1N1 vaccine.

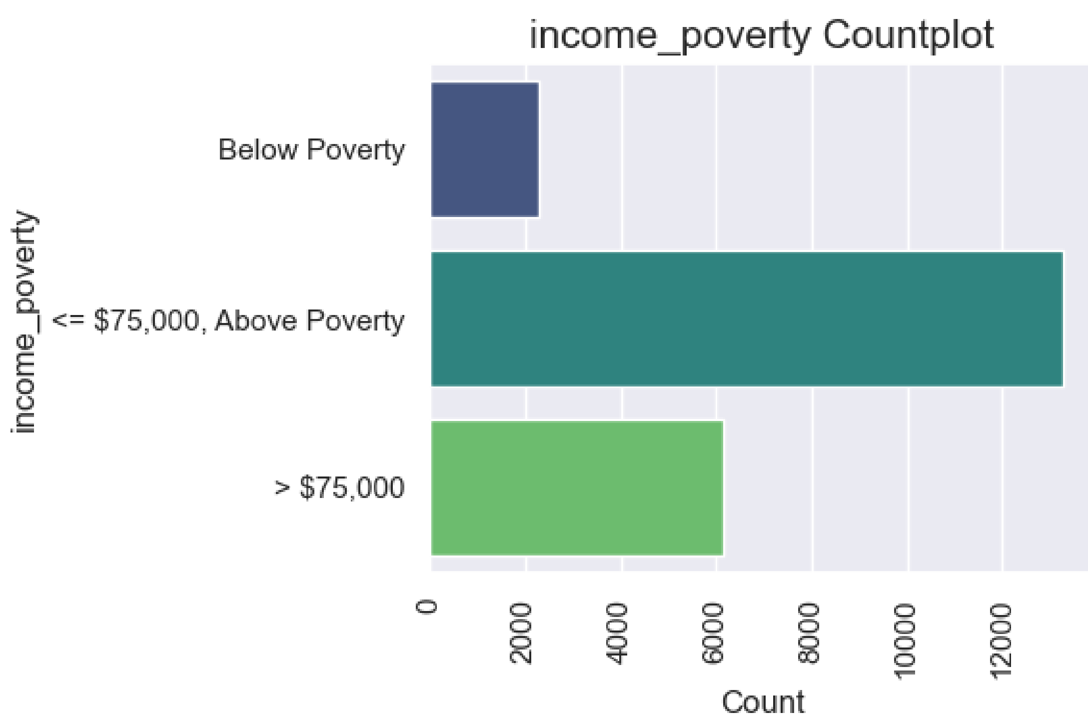
[38]:
```python
# Convert to DataFrame
df_2 = pd.DataFrame(df_categorical)


# Loop through each column in df_2 that has an object data type
for col in df_2.select_dtypes(include='object').columns:
    plt.figure(figsize=(6, 4))

    # Set grid style
    sns.set_style("darkgrid")

    sns.countplot(y=df_2[col], palette="viridis")
    plt.title(f'{col} Countplot', fontsize=15)
    plt.xlabel('Count', fontsize=12)
    plt.ylabel(col, fontsize=12)
    plt.xticks(rotation=90)
    plt.tight_layout()
    plt.show()
```

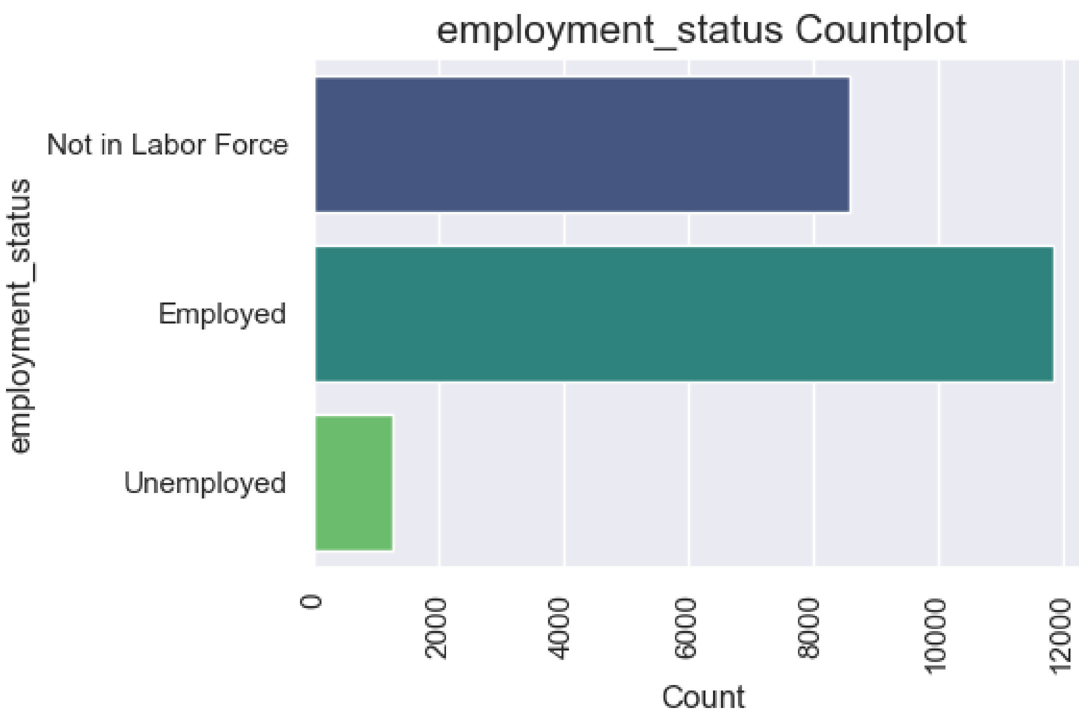


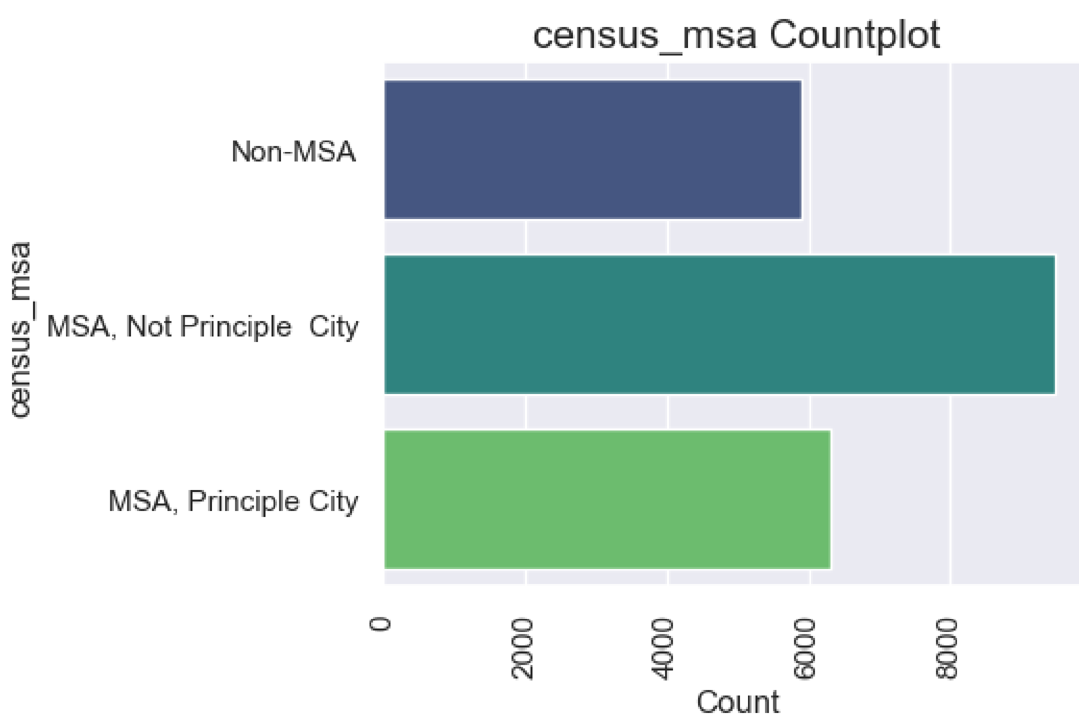

education Countplot
education
< 12 Years
12 Years
Some College
College Graduate
0
2000
4000
6000
8000
Count


race Countplot
race
White
Black
Other or Multiple
Hispanic
0
2500
5000
7500
10000
12500
15000
17500
Count


sex Countplot
Female
Male
sex
0
2000
4000
6000
8000
10000
12000
Count


income_poverty Countplot
Below Poverty
<= $75,000, Above Poverty
> $75,000
income_poverty
0
2000
4000
6000
8000
10000
12000
Count


employment_status Countplot
employment_status
Not in Labor Force
Employed
Unemployed
0
2000
4000
6000
8000
10000
12000
Count

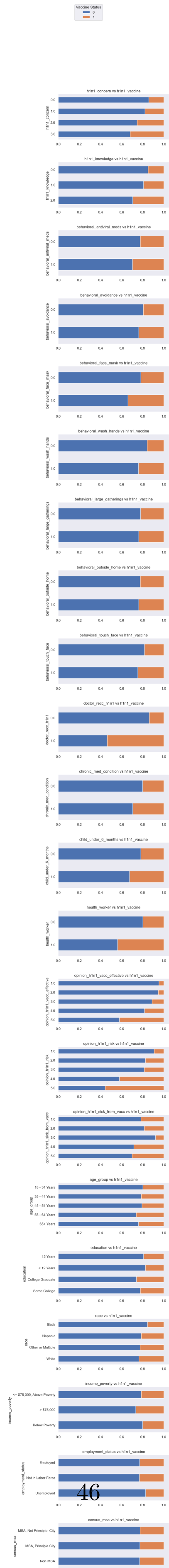

census_msa Countplot
census_msa
Non-MSA
MSA, Not Principle City
MSA, Principle City
0
2000
4000
6000
8000
Count

- The majority of respondents fall within the age groups of 65+ years.
- College graduates represent the largest group among respondents.
- White respondents make up the majority of the sample, followed by smaller proportions of Black, Hispanic, and Other/Multiple races.
- Females are more represented in the dataset compared to males.
- A higher proportion of respondents are at or below the poverty line.
- Most respondents are employed, followed by those not in the labor force and a smaller number of unemployed individuals.
- The majority of respondents reside in Metropolitan Statistical Areas (MSAs), with a notable portion in the MSA's principle city.

Bivariate analysis

[39]:
```
import matplotlib.pyplot as plt

def vaccination_rate_plot(col, target, data, ax=None):
    """Stacked bar chart of vaccination rate for `target` against
    `col`.

    Args:
        col (string): column name of feature variable
        target (string): column name of target variable
        data (pandas DataFrame): dataframe that contains columns
            `col` and `target`
        ax (matplotlib axes object, optional): matplotlib axes
            object to attach plot to
    """
    counts = (data[[target, col]]
                  .groupby([target, col])
                  .size()
                  .unstack(target)
             )
    group_counts = counts.sum(axis='columns')
    props = counts.div(group_counts, axis='index')

    props.plot(kind="barh", stacked=True, ax=ax)
    ax.invert_yaxis()
    ax.legend().remove()
    ax.grid(False)  # Remove gridlines

# List of columns to plot
cols_to_plot = ['h1n1_concern', 'h1n1_knowledge', 'behavioral_antiviral_meds',
       'behavioral_avoidance', 'behavioral_face_mask', 'behavioral_wash_hands',
       'behavioral_large_gatherings', 'behavioral_outside_home',
       'behavioral_touch_face', 'doctor_recc_h1n1', 'chronic_med_condition',
```

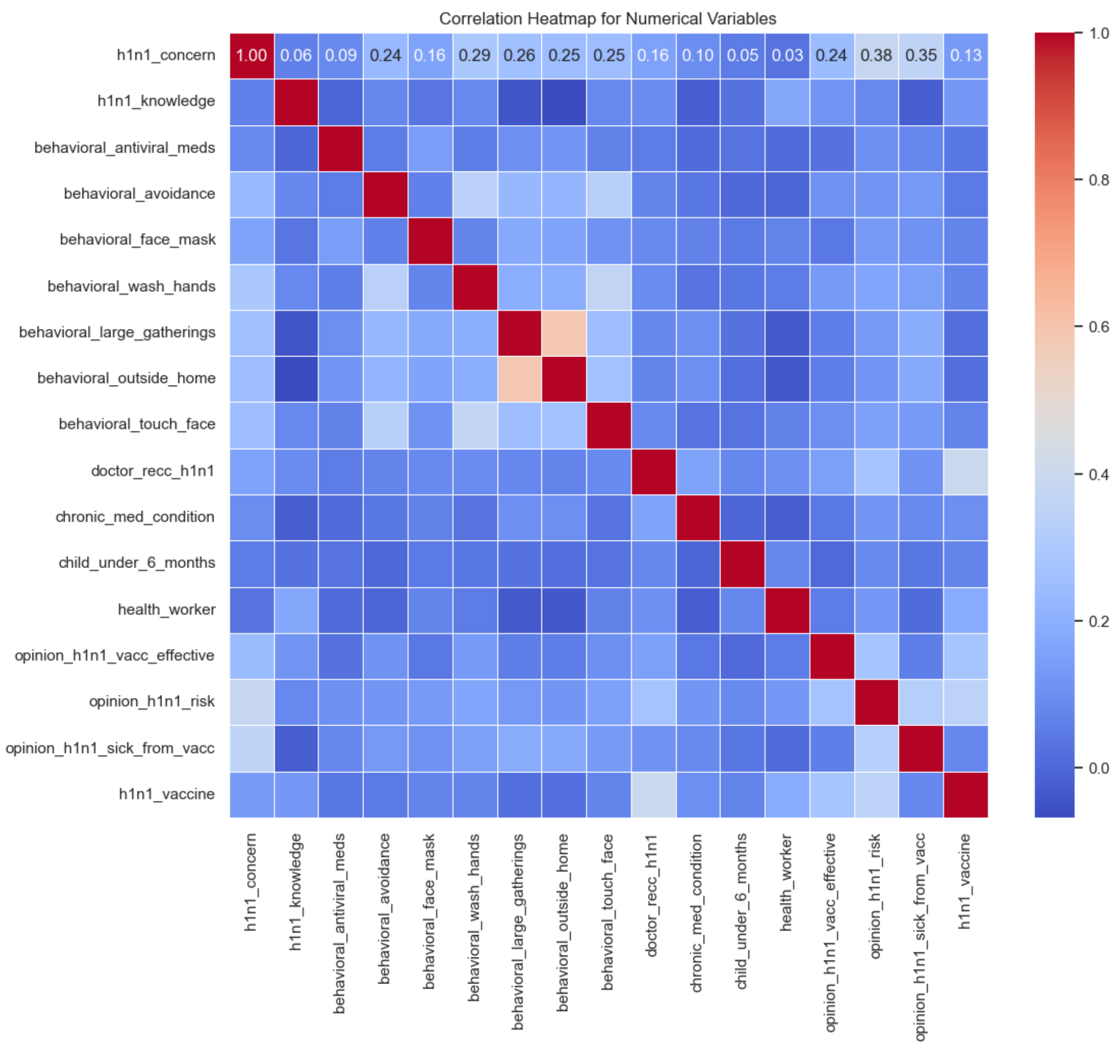
```
        'child_under_6_months', 'health_worker', 'opinion_h1n1_vacc_effective',
        'opinion_h1n1_risk', 'opinion_h1n1_sick_from_vacc' , 'age_group',␣
 ↪'education', 'race',
        'income_poverty', 'employment_status', 'census_msa']

# Calculate number of rows and columns for subplots
num_rows = len(cols_to_plot)
num_cols = 1  # Only one column for h1n1 vaccine

# Create subplots with adjusted space and layout
fig, ax = plt.subplots(
    num_rows, num_cols, figsize=(8, len(cols_to_plot)*3)  # Increased figsize␣
 ↪to accommodate more subplots
)

# Loop over each column for h1n1 vaccine
for idx, col in enumerate(cols_to_plot):
    # Plot for 'h1n1_vaccine'
    vaccination_rate_plot(
        col, 'h1n1_vaccine', df, ax=ax[idx]
    )
    ax[idx].set_title(f'{col} vs h1n1_vaccine')

# Add legends to the first subplot
handles, labels = ax[0].get_legend_handles_labels()
fig.legend(handles, labels, loc='upper center', bbox_to_anchor=(0.5, 1.05),␣
 ↪title='Vaccine Status')

# Adjust layout
plt.tight_layout(pad=3)  # Increase spacing between subplots
plt.show()
```


Vaccine Status
0
1
h1n1_concern vs h1n1_vaccine
h1n1_knowledge vs h1n1_vaccine
behavioral_antiviral_meds vs h1n1_vaccine
behavioral_avoidance vs h1n1_vaccine
behavioral_face_mask vs h1n1_vaccine
behavioral_wash_hands vs h1n1_vaccine
behavioral_large_gatherings vs h1n1_vaccine
behavioral_outside_home vs h1n1_vaccine
behavioral_touch_face vs h1n1_vaccine
doctor_recc_h1n1 vs h1n1_vaccine
chronic_med_condition vs h1n1_vaccine
child_under_6_months vs h1n1_vaccine
health_worker vs h1n1_vaccine
opinion_h1n1_vacc_effective vs h1n1_vaccine
opinion_h1n1_risk vs h1n1_vaccine
opinion_h1n1_sick_from_vacc vs h1n1_vaccine
age_group vs h1n1_vaccine
18 - 34 Years
35 - 44 Years
45 - 54 Years
55 - 64 Years
65+ Years
education vs h1n1_vaccine
12 Years
< 12 Years
College Graduate
Some College
race vs h1n1_vaccine
Black
Hispanic
Other or Multiple
White
income_poverty vs h1n1_vaccine
<= $75,000, Above Poverty
> $75,000
Below Poverty
employment_status vs h1n1_vaccine
Employed
Not in Labor Force
Unemployed
census_msa vs h1n1_vaccine
MSA, Not Principle City
MSA, Principle City
Non-MSA

- We can observe the interplay between behavioral, geographical, and opinion-based features and their impact on the uptake of the H1N1 vaccine.

Multivariate analysis Correlation between our target variable and our predictor (numerical) variables

[40]:
```
# Compute the correlation matrix
corr = df_numerical.corr()

# Plot the heatmap
plt.figure(figsize=(12, 10))
sns.heatmap(corr, annot=True, cmap='coolwarm', fmt=".2f", linewidths=0.5)
plt.title('Correlation Heatmap for Numerical Variables')
plt.show()
```



[41]:
```python
# Extract correlation coefficients with 'price'
h1n1_correlations = corr['h1n1_vaccine']

# Sort correlation coefficients in descending order
h1n1_correlations_sorted = h1n1_correlations.sort_values(ascending=False)

# Print correlation coefficients
print("Correlation Coefficients with H1N1 (Descending Order):")
print(h1n1_correlations_sorted)
```

```
Correlation Coefficients with H1N1 (Descending Order):
h1n1_vaccine                   1.000000
doctor_recc_h1n1               0.396728
opinion_h1n1_risk              0.347309
opinion_h1n1_vacc_effective    0.275346
health_worker                  0.182383
h1n1_concern                   0.134559
h1n1_knowledge                 0.125305
chronic_med_condition          0.101601
opinion_h1n1_sick_from_vacc    0.080862
behavioral_wash_hands          0.074812
behavioral_face_mask           0.073053
child_under_6_months           0.070511
behavioral_touch_face          0.070322
behavioral_avoidance           0.046990
behavioral_antiviral_meds      0.037520
behavioral_large_gatherings    0.019186
behavioral_outside_home        0.017751
Name: h1n1_vaccine, dtype: float64
```

1.13 DATA PREPROCESSING

ENCODING OUR CATEGORICAL VARIABLES

[42]:
```python
# Viewing our data types

df_2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 21710 entries, 0 to 26706
Data columns (total 7 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   age_group          21710 non-null  object
 1   education          21710 non-null  object
 2   race               21710 non-null  object
 3   sex                21710 non-null  object
```

```
 4   income_poverty     21710 non-null  object
 5   employment_status  21710 non-null  object
 6   census_msa         21710 non-null  object
dtypes: object(7)
memory usage: 1.3+ MB
```

[43]:
```python
# Display unique values in df_2

for column in df_2.columns:
    unique_values = df_2[column].unique()
    print(f"Unique values in column '{column}':")
    print(unique_values)
    print()
```

```
Unique values in column 'age_group':
['55 - 64 Years' '35 - 44 Years' '65+ Years' '45 - 54 Years'
 '18 - 34 Years']

Unique values in column 'education':
['< 12 Years' '12 Years' 'Some College' 'College Graduate']

Unique values in column 'race':
['White' 'Black' 'Other or Multiple' 'Hispanic']

Unique values in column 'sex':
['Female' 'Male']

Unique values in column 'income_poverty':
['Below Poverty' '<= $75,000, Above Poverty' '> $75,000']

Unique values in column 'employment_status':
['Not in Labor Force' 'Employed' 'Unemployed']

Unique values in column 'census_msa':
['Non-MSA' 'MSA, Not Principle  City' 'MSA, Principle City']
```

- We will handle our values under income_poverty using mapping to convert them into numerical representations. This approach facilitates the identification of trends and patterns related to income levels in relation to the H1N1 vaccine uptake.
- For the categorical columns, I will perform One Hot Encoding.

[44]:
```python
# Mapping for income_poverty

income_poverty_mapping = {
    'Below Poverty': 0,
    '<= $75,000, Above Poverty': 1,
    '> $75,000': 2
```

```
}

df_2['income_poverty'] = df_2['income_poverty'].map(income_poverty_mapping)


# List of categorical columns
categorical_columns = ['race', 'sex', 'employment_status', 'census_msa',␣
↪'age_group', 'education']

# Perform one-hot encoding
df_encoded = pd.get_dummies(df_2, columns=categorical_columns, drop_first=True,␣
↪dtype=int)
```

- To streamline the dataset, I'll convert all float values to integers. This ensures consistency across the numerical dataframe

[45]:
```
# Convert float values to integers

df_numerical = df_numerical.astype(int)
df_numerical.head()
```

[45]:
```
               h1n1_concern  h1n1_knowledge  behavioral_antiviral_meds  \
respondent_id
0                         1               0                          0
1                         3               2                          0
3                         1               1                          0
4                         2               1                          0
5                         3               1                          0

               behavioral_avoidance  behavioral_face_mask  \
respondent_id
0                                 0                     0
1                                 1                     0
3                                 1                     0
4                                 1                     0
5                                 1                     0

               behavioral_wash_hands  behavioral_large_gatherings  \
respondent_id
0                                  0                            0
1                                  1                            0
3                                  1                            1
4                                  1                            1
5                                  1                            0

               behavioral_outside_home  behavioral_touch_face  \
respondent_id
```

```
0                              1                 1
1                              1                 1
3                              0                 0
4                              0                 1
5                              0                 0

             doctor_recc_h1n1  chronic_med_condition  child_under_6_months  \
respondent_id
0                           0                      0                     0
1                           0                      0                     0
3                           0                      1                     0
4                           0                      0                     0
5                           0                      0                     0

             health_worker  opinion_h1n1_vacc_effective  opinion_h1n1_risk  \
respondent_id
0                        0                            3                  1
1                        0                            5                  4
3                        0                            3                  3
4                        0                            3                  3
5                        0                            5                  2

             opinion_h1n1_sick_from_vacc  h1n1_vaccine
respondent_id
0                                      2             0
1                                      4             0
3                                      5             0
4                                      2             0
5                                      1             0
```

[46]:

```python
# Merge the DataFrames based on their indices

df_concat = pd.concat([df_numerical, df_encoded], axis=1)
df_concat.head()
```

[46]:

```
             h1n1_concern  h1n1_knowledge  behavioral_antiviral_meds  \
respondent_id
0                       1               0                          0
1                       3               2                          0
3                       1               1                          0
4                       2               1                          0
5                       3               1                          0

             behavioral_avoidance  behavioral_face_mask  \
respondent_id
0                               0                     0
1                               1                     0
```

```
3                                 1                     0
4                                 1                     0
5                                 1                     0

               behavioral_wash_hands  behavioral_large_gatherings  \
respondent_id
0                                  0                            0
1                                  1                            0
3                                  1                            1
4                                  1                            1
5                                  1                            0

               behavioral_outside_home  behavioral_touch_face  \
respondent_id
0                                    1                      1
1                                    1                      1
3                                    0                      0
4                                    0                      1
5                                    0                      1

               doctor_recc_h1n1  …  employment_status_Unemployed  \
respondent_id                    …
0                             0  …                             0
1                             0  …                             0
3                             0  …                             0
4                             0  …                             0
5                             0  …                             0

               census_msa_MSA, Principle City  census_msa_Non-MSA  \
respondent_id
0                                           0                   1
1                                           0                   0
3                                           1                   0
4                                           0                   0
5                                           1                   0

               age_group_35 - 44 Years  age_group_45 - 54 Years  \
respondent_id
0                                    0                        0
1                                    1                        0
3                                    0                        0
4                                    0                        1
5                                    0                        0

               age_group_55 - 64 Years  age_group_65+ Years  \
respondent_id
0                                    1                    0
```

```
1                                  0                   0
3                                  0                   1
4                                  0                   0
5                                  0                   1

              education_< 12 Years  education_College Graduate  \
respondent_id
0                                1                           0
1                                0                           0
3                                0                           0
4                                0                           0
5                                0                           0

              education_Some College
respondent_id
0                                  0
1                                  0
3                                  0
4                                  1
5                                  0

[5 rows x 33 columns]
```

All of our values are now numerical. Let's preview our new dataframe;

[47]:
```
df_concat.columns
```

[47]:
```
Index(['h1n1_concern', 'h1n1_knowledge', 'behavioral_antiviral_meds',
       'behavioral_avoidance', 'behavioral_face_mask', 'behavioral_wash_hands',
       'behavioral_large_gatherings', 'behavioral_outside_home',
       'behavioral_touch_face', 'doctor_recc_h1n1', 'chronic_med_condition',
       'child_under_6_months', 'health_worker', 'opinion_h1n1_vacc_effective',
       'opinion_h1n1_risk', 'opinion_h1n1_sick_from_vacc', 'h1n1_vaccine',
       'income_poverty', 'race_Hispanic', 'race_Other or Multiple',
       'race_White', 'sex_Male', 'employment_status_Not in Labor Force',
       'employment_status_Unemployed', 'census_msa_MSA, Principle City',
       'census_msa_Non-MSA', 'age_group_35 - 44 Years',
       'age_group_45 - 54 Years', 'age_group_55 - 64 Years',
       'age_group_65+ Years', 'education_< 12 Years',
       'education_College Graduate', 'education_Some College'],
      dtype='object')
```

[48]:
```
df_concat.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 21710 entries, 0 to 26706
Data columns (total 33 columns):
 #   Column                                Non-Null Count  Dtype
```

```
---  ------                               --------------  -----
 0   h1n1_concern                         21710 non-null  int32
 1   h1n1_knowledge                       21710 non-null  int32
 2   behavioral_antiviral_meds            21710 non-null  int32
 3   behavioral_avoidance                 21710 non-null  int32
 4   behavioral_face_mask                 21710 non-null  int32
 5   behavioral_wash_hands                21710 non-null  int32
 6   behavioral_large_gatherings          21710 non-null  int32
 7   behavioral_outside_home              21710 non-null  int32
 8   behavioral_touch_face                21710 non-null  int32
 9   doctor_recc_h1n1                     21710 non-null  int32
 10  chronic_med_condition                21710 non-null  int32
 11  child_under_6_months                 21710 non-null  int32
 12  health_worker                        21710 non-null  int32
 13  opinion_h1n1_vacc_effective          21710 non-null  int32
 14  opinion_h1n1_risk                    21710 non-null  int32
 15  opinion_h1n1_sick_from_vacc          21710 non-null  int32
 16  h1n1_vaccine                         21710 non-null  int32
 17  income_poverty                       21710 non-null  int64
 18  race_Hispanic                        21710 non-null  int32
 19  race_Other or Multiple               21710 non-null  int32
 20  race_White                           21710 non-null  int32
 21  sex_Male                             21710 non-null  int32
 22  employment_status_Not in Labor Force 21710 non-null  int32
 23  employment_status_Unemployed         21710 non-null  int32
 24  census_msa_MSA, Principle City       21710 non-null  int32
 25  census_msa_Non-MSA                   21710 non-null  int32
 26  age_group_35 - 44 Years              21710 non-null  int32
 27  age_group_45 - 54 Years              21710 non-null  int32
 28  age_group_55 - 64 Years              21710 non-null  int32
 29  age_group_65+ Years                  21710 non-null  int32
 30  education_< 12 Years                 21710 non-null  int32
 31  education_College Graduate           21710 non-null  int32
 32  education_Some College               21710 non-null  int32
dtypes: int32(32), int64(1)
memory usage: 3.0 MB
```

[49]:
```
# Obtain the target and features from the DataFrame

y = df_concat['h1n1_vaccine']
X = df_concat.drop(columns='h1n1_vaccine')
```

TRAIN-TEST SPLIT

[50]:
```
# Splitting the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
 ↪random_state=42)
```

```
# Verify the shapes of the splits
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: (17368, 32)
X_test shape: (4342, 32)
y_train shape: (17368,)
y_test shape: (4342,)
```

FEATURE SCALING

- We standardize our data to ensure it is on the same scale.
- This ensures that both the training and test data are scaled consistently using the same parameters derived from the training data, maintaining uniformity across the dataset preprocessing.

[51]:
```
# Initialize the StandardScaler
scaler = StandardScaler()

# Fit the scaler to the training data and transform it
X_train_scaled = scaler.fit_transform(X_train)
```

CHECKING FOR MULTICOLLINEARITY

- We will check for multicollinearity using the Variance Inflation Factor.

[52]:
```
# Calculate VIF for each feature
def calculate_vif(X):
    vif_data = pd.DataFrame()
    vif_data["Variable"] = X.columns
    vif_data["VIF"] = [variance_inflation_factor(X.values, i) for i in range(X.
 ↪shape[1])]
    return vif_data

vif_data = calculate_vif(pd.DataFrame(X_train_scaled, columns=X_train.columns))
vif_data
```

[52]:

	Variable	VIF
0	h1n1_concern	1.466682
1	h1n1_knowledge	1.221782
2	behavioral_antiviral_meds	1.063649
3	behavioral_avoidance	1.259469
4	behavioral_face_mask	1.089905
5	behavioral_wash_hands	1.298419
6	behavioral_large_gatherings	1.627522
7	behavioral_outside_home	1.649195

```
8                    behavioral_touch_face  1.316435
9                        doctor_recc_h1n1  1.130644
10                   chronic_med_condition  1.114700
11                     child_under_6_months  1.032408
12                           health_worker  1.124852
13             opinion_h1n1_vacc_effective  1.150002
14                       opinion_h1n1_risk  1.393076
15             opinion_h1n1_sick_from_vacc  1.242573
16                          income_poverty  1.338345
17                           race_Hispanic  1.779146
18                  race_Other or Multiple  1.669494
19                              race_White  2.491995
20                                sex_Male  1.113072
21  employment_status_Not in Labor Force  1.594287
22            employment_status_Unemployed  1.103462
23         census_msa_MSA, Principle City  1.212680
24                      census_msa_Non-MSA  1.217210
25                  age_group_35 - 44 Years  1.544823
26                  age_group_45 - 54 Years  1.729281
27                  age_group_55 - 64 Years  1.781677
28                      age_group_65+ Years  2.208127
29                     education_< 12 Years  1.326419
30              education_College Graduate  1.994416
31                  education_Some College  1.668725
```

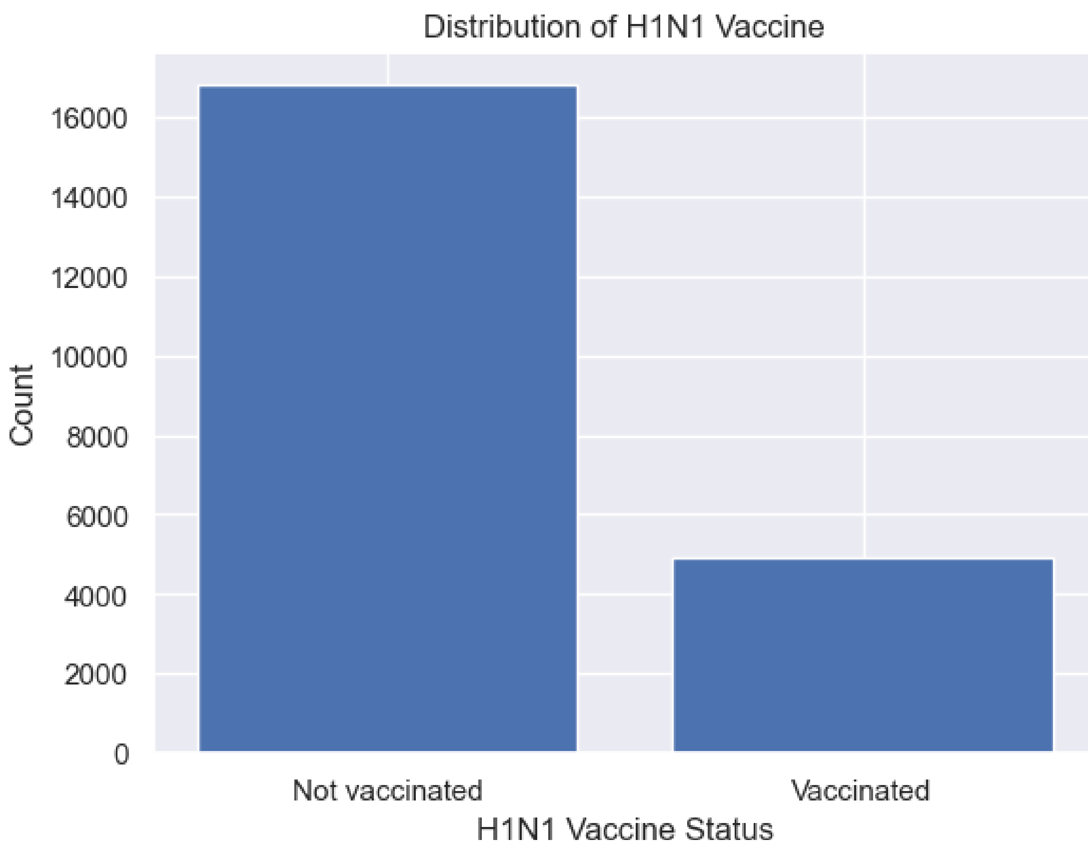
- The dataset exhibits generally low multicollinearity among the features, with most VIF values close to 1, indicating minimal multicollinearity.

CLASS IMBALANCE

[53]:
```python
# Checking for class imbalance
value_counts = df_concat['h1n1_vaccine'].value_counts()
print(value_counts)
```

```
h1n1_vaccine
0    16788
1     4922
Name: count, dtype: int64
```

[54]:
```python
# Visualizing class imbalance

plt.bar(value_counts.index, value_counts.values)
plt.xlabel('H1N1 Vaccine Status')
plt.ylabel('Count')
plt.title('Distribution of H1N1 Vaccine')
plt.xticks(value_counts.index, ['Not vaccinated', 'Vaccinated'])
plt.show()
```



- The data shows a class imbalance in the target variable, H1N1 Vaccine.
- There are significantly more instances of Class 0 (Not vaccinated) than Class 1 (Vaccinated).

Solve for class Imbalance

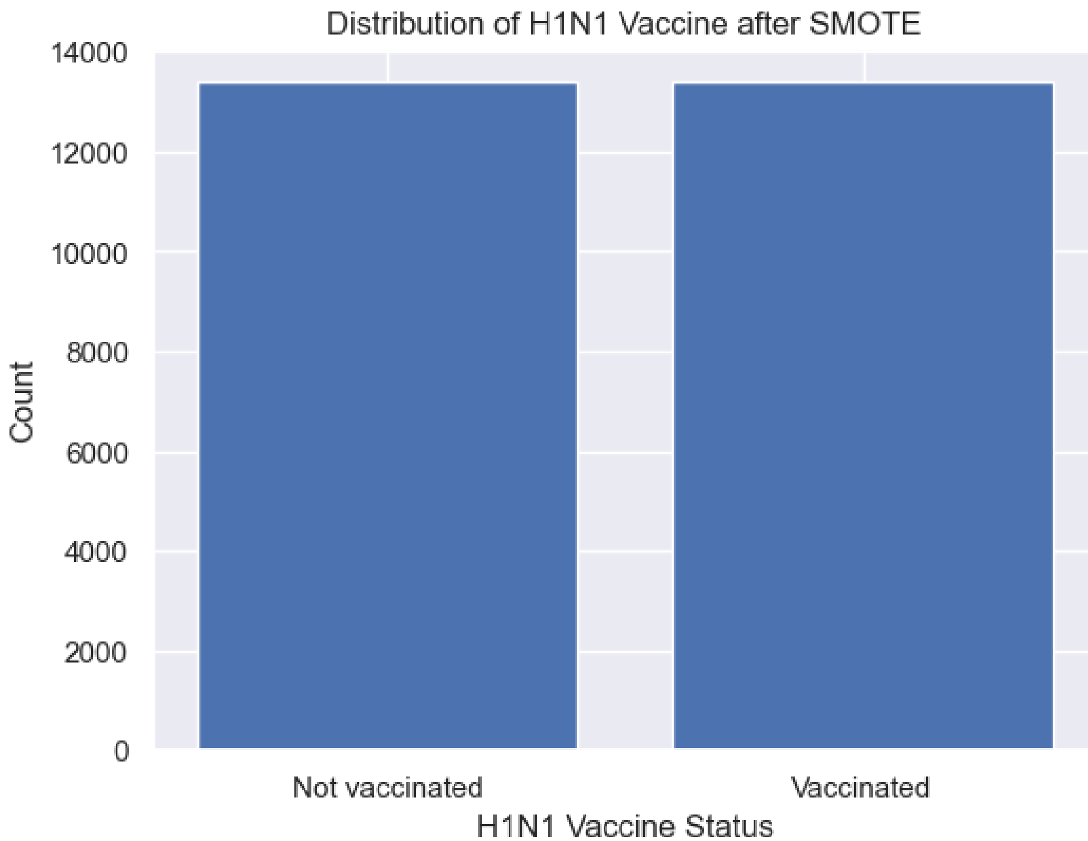
- We will apply SMOTE (Synthetic Minority Over-sampling Technique) to prevent potential underfitting or biasing of the model towards the majority class.

[55]:
```
# Apply SMOTE to oversample the minority class
smote = SMOTE(random_state=42)
X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_train)
```

[56]:
```
# Checking for class imbalance after SMOTE
value_counts_resampled = y_train_resampled.value_counts()
print(value_counts_resampled)

# Visualizing class imbalance after SMOTE
plt.bar(value_counts_resampled.index, value_counts_resampled.values)
plt.xlabel('H1N1 Vaccine Status')
plt.ylabel('Count')
plt.title('Distribution of H1N1 Vaccine after SMOTE')
```

```
plt.xticks(value_counts_resampled.index, ['Not vaccinated', 'Vaccinated'])
plt.show()
```

```
h1n1_vaccine
0    13391
1    13391
Name: count, dtype: int64
```



- The H1N1 Vaccine status is now balanced.

[57]:
```
# Fit the scaler to the resampled training data and transform it
X_train_resampled_scaled = scaler.fit_transform(X_train_resampled)

# Transform the test data using the same scaler
X_test_scaled = scaler.transform(X_test)
```

1.14 MODELLING

BASELINE MODEL

- For my baseline model I will build a Logistic Regression Model.

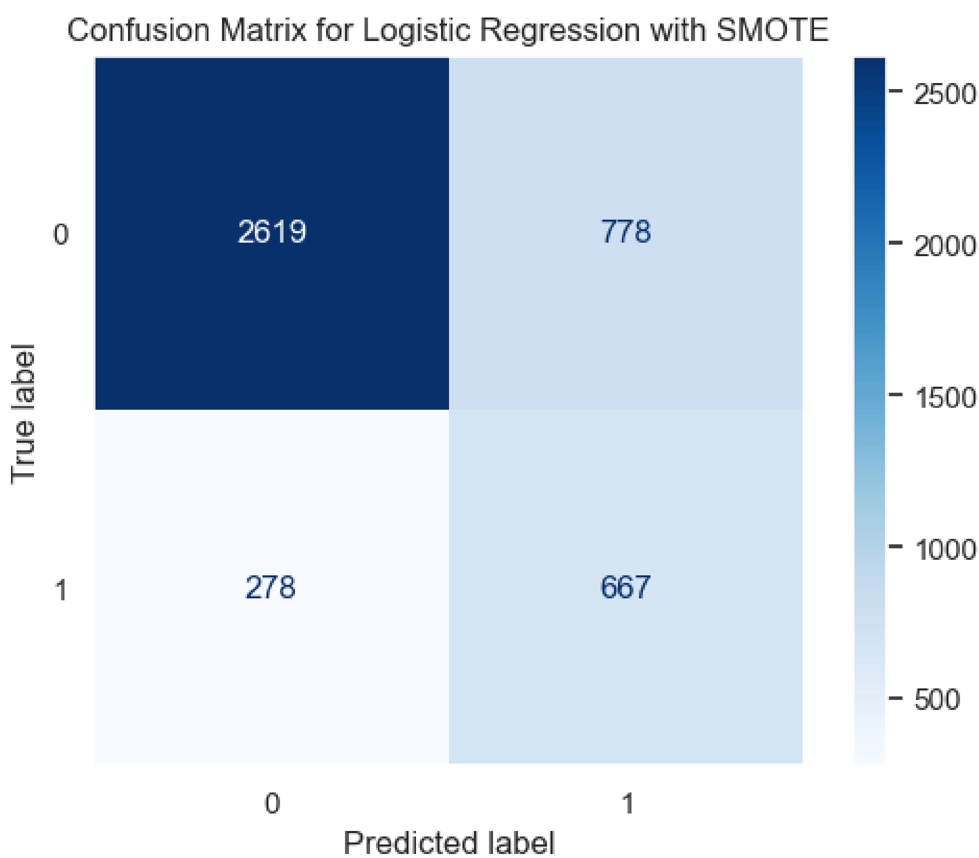
1. LOGISTIC REGRESSION MODEL

[58]:
```python
# Initialize logistic regression model
log_model = LogisticRegression(random_state=42)

# Fit the model to training data
log_model.fit(X_train_resampled_scaled, y_train_resampled)

# Predict on the test set
y_pred_test = log_model.predict(X_test_scaled)

# Define confusion matrix
def conf_matrix(y_test, y_pred_test):
    cm = {'TP': 0, 'TN': 0, 'FP': 0, 'FN': 0}
    for ind, label in enumerate(y_test):
        pred = y_pred_test[ind]
        if label == 1:
            if label == pred:
                cm['TP'] += 1
            else:
                cm['FN'] += 1
        else:
            if label == pred:
                cm['TN'] += 1
            else:
                cm['FP'] += 1
    return cm


# Accuracy score
accuracy = accuracy_score(y_test, y_pred_test)
print("\nAccuracy Score:", accuracy)


# Confusion matrix function
cm_dict = conf_matrix(y_test, y_pred_test)
print("\nCustom Confusion Matrix:", cm_dict)


# Classification report
print("\nClassification Report:")
print(classification_report(y_test, y_pred_test))
```

```
Accuracy Score: 0.7567941040994933

Custom Confusion Matrix: {'TP': 667, 'TN': 2619, 'FP': 778, 'FN': 278}
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.90      0.77      0.83      3397
           1       0.46      0.71      0.56       945

    accuracy                           0.76      4342
   macro avg       0.68      0.74      0.70      4342
weighted avg       0.81      0.76      0.77      4342
```

[59]:
```python
# Visualizing the confusion matrix

cnf_matrix = confusion_matrix(y_test, y_pred_test)
disp = ConfusionMatrixDisplay(confusion_matrix=cnf_matrix,
 ↪display_labels=log_model.classes_)
disp.plot(cmap=plt.cm.Blues)
plt.title('Confusion Matrix for Logistic Regression with SMOTE')
plt.grid(False)
plt.show()
```



- **True Positives:** Instances where the model correctly predicted individuals who opted for the H1N1 vaccine. In this case, there are 667 individuals who took the vaccine and were correctly identified by the model.
- **True Negatives:** Instances where the model correctly predicted individuals who did not opt for the H1N1 vaccine. In this case 2619 individuals who did not take the vaccine were correctly identified by the model.
- **False Positives:** Instances where the model incorrectly predicted individuals as vaccine takers when they did not. 778 individuals were falsely classified as vaccine takers.
- **False Negatives:** Instances where the model incorrectly predicted individuals as non-vaccine takers when they actually took the vaccine. 278 individuals who took the vaccine were mistakenly classified as non-takers.

Hyperparameter tuning for Logistic Regression using GridSearchCV

- This will help improve the model's predictive power and generalization to unseen data

[60]:
```python
# Define the logistic regression model
logistic = LogisticRegression()

# Define the hyperparameters grid
param_grid = {'C': [0.1, 1, 10, 100, 1000]}

# Grid search with 5-fold cross-validation
grid_search = GridSearchCV(estimator=logistic, param_grid=param_grid, cv=5,␣
 ↪scoring='accuracy')

# Fit the grid search to data
grid_search.fit(X_train_resampled_scaled, y_train_resampled)

# Print best hyperparameters
print("Best hyperparameters:", grid_search.best_params_)

# Get the best model
best_logistic = grid_search.best_estimator_

# Make predictions on the test set using the best model
y_pred_test_best = best_logistic.predict(X_test_scaled)

# Calculate accuracy on the test set using the best model
test_accuracy_best = accuracy_score(y_test, y_pred_test_best)
print("Test Accuracy (Best Model):", test_accuracy_best)

# Classification report
print("\nClassification Report:")
print(classification_report(y_test, y_pred_test))
```

```
Best hyperparameters: {'C': 1}
Test Accuracy (Best Model): 0.7567941040994933
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.90      0.77      0.83      3397
           1       0.46      0.71      0.56       945

    accuracy                           0.76      4342
   macro avg       0.68      0.74      0.70      4342
weighted avg       0.81      0.76      0.77      4342
```

- The accuracy of both logistic regression models is identical, suggesting that hyperparameter tuning did not significantly affect performance. Next, I will explore a decision tree classifier model

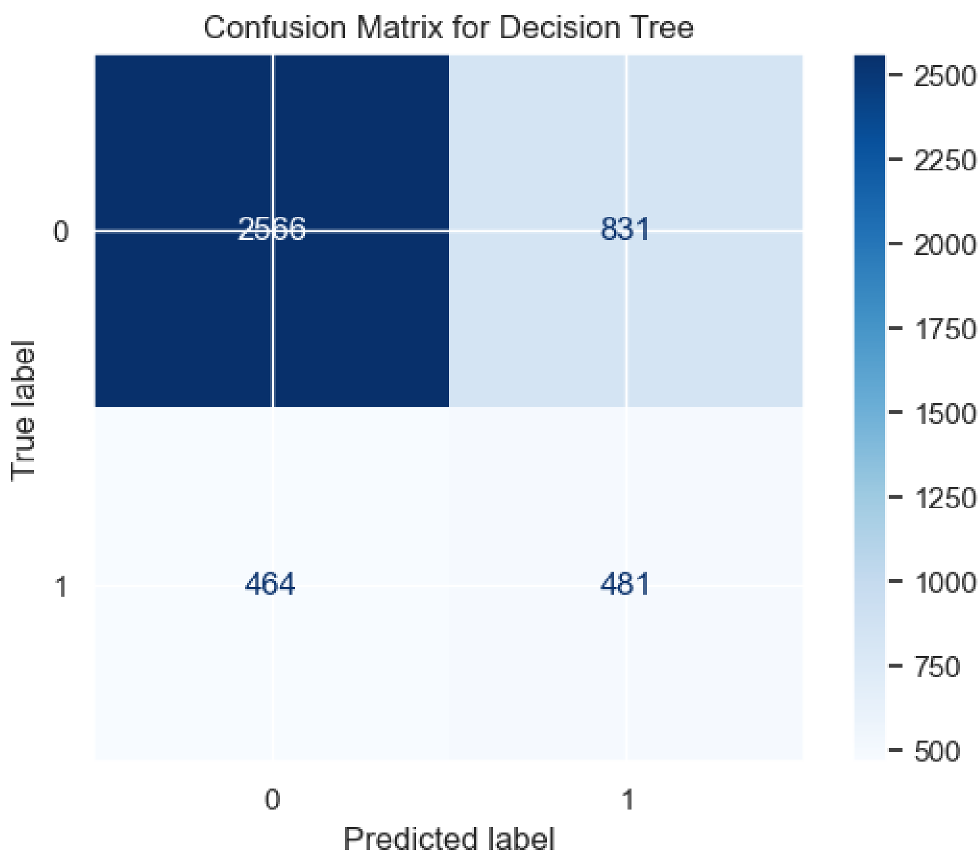
2. DECISION TREE CLASSIFIER MODEL

[61]:
```python
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report, confusion_matrix,␣
 ↪accuracy_score

# Initialize decision tree classifier
decision_tree = DecisionTreeClassifier(random_state=42)

# Fit the model to the training data
decision_tree.fit(X_train_resampled_scaled, y_train_resampled)

# Make predictions on test set
y_pred_test_dt = decision_tree.predict(X_test_scaled)

# Calculate accuracy on test set
test_accuracy_dt = accuracy_score(y_test, y_pred_test_dt)
print("\nTest Accuracy (Decision Tree):", test_accuracy_dt)

# Print the confusion matrix
print("\nConfusion Matrix (Decision Tree):")
print(confusion_matrix(y_test, y_pred_test_dt))

# Print the classification report
print("\nClassification Report (Decision Tree):")
print(classification_report(y_test, y_pred_test_dt))
```

```
Test Accuracy (Decision Tree): 0.7017503454629204

Confusion Matrix (Decision Tree):
[[2566  831]
 [ 464  481]]
```

```
Classification Report (Decision Tree):
              precision    recall  f1-score   support

           0       0.85      0.76      0.80      3397
           1       0.37      0.51      0.43       945

    accuracy                           0.70      4342
   macro avg       0.61      0.63      0.61      4342
weighted avg       0.74      0.70      0.72      4342
```

[62]:
```python
# Visualizing the confusion matrix

cnf_matrix = confusion_matrix(y_test, y_pred_test_dt)
disp = ConfusionMatrixDisplay(confusion_matrix=cnf_matrix,␣
 ↪display_labels=decision_tree.classes_)
disp.plot(cmap=plt.cm.Blues)
plt.title('Confusion Matrix for Decision Tree')
plt.show()
```



- **True Positives:** Instances where the model correctly predicted individuals who opted for the H1N1 vaccine. In this case, there are 481 individuals who took the vaccine and were correctly identified by the model.
- **True Negatives:** Instances where the model correctly predicted individuals who did not opt for the H1N1 vaccine. In this case 2566 individuals who did not take the vaccine were correctly identified by the model.
- **False Positives:** Instances where the model incorrectly predicted individuals as vaccine takers when they did not. 831 individuals were falsely classified as vaccine takers.
- **False Negatives:** Instances where the model incorrectly predicted individuals as non-vaccine takers when they actually took the vaccine. 464 individuals who took the vaccine were mistakenly classified as non-takers.

Hyperparameter tuning for Decision Tree Classifier using GridSearchCV

[63]:
```
# Define the parameter grid
param_grid = {
    'criterion': ['gini', 'entropy'],
    'max_depth': [None, 10, 20, 30, 40, 50],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

# Initialize the Decision Tree Classifier
decision_tree = DecisionTreeClassifier(random_state=42)

# Initialize GridSearchCV
grid_search = GridSearchCV(estimator=decision_tree, param_grid=param_grid,
 ↪cv=5, scoring='accuracy', n_jobs=-1)

# Perform the grid search on the resampled training data
grid_search.fit(X_train_resampled_scaled, y_train_resampled)

# Get the best parameters and best score
best_params = grid_search.best_params_
best_score = grid_search.best_score_

print("Best Parameters:", best_params)
print("Best Accuracy Score:", best_score)

# Initialize Decision Tree Classifier with best parameters
best_decision_tree = DecisionTreeClassifier(**best_params, random_state=42)

# Fit the model to the training data
best_decision_tree.fit(X_train_resampled_scaled, y_train_resampled)

# Make predictions on the test set
y_pred_test_best_dt = best_decision_tree.predict(X_test_scaled)
```

```
# Calculate accuracy on the test set
test_accuracy_best_dt = accuracy_score(y_test, y_pred_test_best_dt)
print("\nTest Accuracy (Best Decision Tree):", test_accuracy_best_dt)

# Print the confusion matrix
print("\nConfusion Matrix (Best Decision Tree):")
print(confusion_matrix(y_test, y_pred_test_best_dt))

# Print the classification report
print("\nClassification Report (Best Decision Tree):")
print(classification_report(y_test, y_pred_test_best_dt))
```

```
Best Parameters: {'criterion': 'entropy', 'max_depth': None, 'min_samples_leaf':
1, 'min_samples_split': 2}
Best Accuracy Score: 0.7907946064023494

Test Accuracy (Best Decision Tree): 0.6992169507139567

Confusion Matrix (Best Decision Tree):
[[2538  859]
 [ 447  498]]

Classification Report (Best Decision Tree):
              precision    recall  f1-score   support

           0       0.85      0.75      0.80      3397
           1       0.37      0.53      0.43       945

    accuracy                           0.70      4342
   macro avg       0.61      0.64      0.61      4342
weighted avg       0.75      0.70      0.72      4342
```

- The model performs reasonably well in predicting the class 0, achieving high precision, recall, and f1-score. However, its performance is comparatively weaker in predicting the class 1, as indicated by lower precision, recall, and f1-score values.
- Let's explore KNN, a non-parametric approach that adapts well to complex data patterns, offering potential improvements in predictive performance.

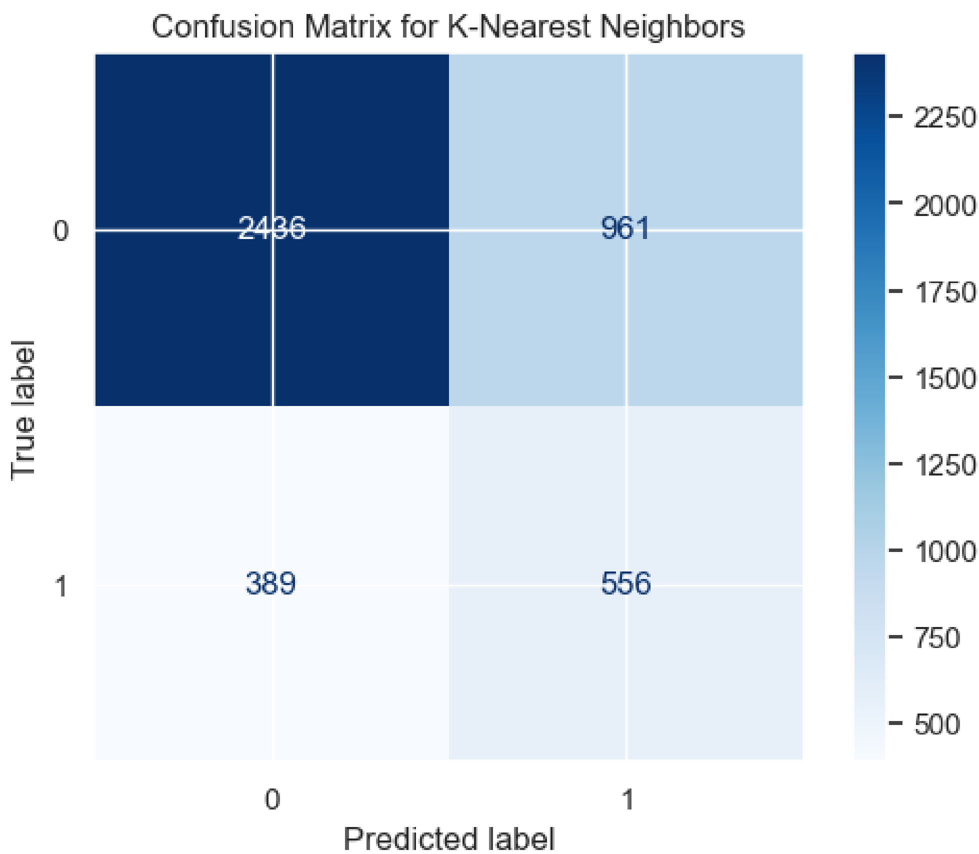
3. K-NEAREST NEIGHBORS MODEL

[64]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix, classification_report,␣
↪accuracy_score, ConfusionMatrixDisplay
import matplotlib.pyplot as plt

# Initialize the K-Nearest Neighbors model
knn = KNeighborsClassifier()
```

```python
# Train the model
knn.fit(X_train_resampled_scaled, y_train_resampled)

# Make predictions on the test set
y_pred_knn = knn.predict(X_test_scaled)

# Calculate accuracy on the test set
test_accuracy_knn = accuracy_score(y_test, y_pred_knn)
print("Test Accuracy (K-Nearest Neighbors):", test_accuracy_knn)

# Print the confusion matrix
print("\nConfusion Matrix (K-Nearest Neighbors):")
print(confusion_matrix(y_test, y_pred_knn))

# Print the classification report
print("\nClassification Report (K-Nearest Neighbors):")
print(classification_report(y_test, y_pred_knn))
```

```
Test Accuracy (K-Nearest Neighbors): 0.6890833717181023

Confusion Matrix (K-Nearest Neighbors):
[[2436  961]
 [ 389  556]]

Classification Report (K-Nearest Neighbors):
              precision    recall  f1-score   support

           0       0.86      0.72      0.78      3397
           1       0.37      0.59      0.45       945

    accuracy                           0.69      4342
   macro avg       0.61      0.65      0.62      4342
weighted avg       0.75      0.69      0.71      4342
```

[65]:
```python
# Visualizing the confusion matrix
cnf_matrix_knn = confusion_matrix(y_test, y_pred_knn)
disp = ConfusionMatrixDisplay(confusion_matrix=cnf_matrix_knn,␣
 ↪display_labels=knn.classes_)
disp.plot(cmap=plt.cm.Blues)
plt.title('Confusion Matrix for K-Nearest Neighbors')
plt.show()
```



- **True Positives:** Instances where the model correctly predicted individuals who opted for the H1N1 vaccine. In this case, there are 556 individuals who took the vaccine and were correctly identified by the model.
- **True Negatives:** Instances where the model correctly predicted individuals who did not opt for the H1N1 vaccine. In this case 2436 individuals who did not take the vaccine were correctly identified by the model.
- **False Positives:** Instances where the model incorrectly predicted individuals as vaccine takers when they did not. 961 individuals were falsely classified as vaccine takers.
- **False Negatives:** Instances where the model incorrectly predicted individuals as non-vaccine takers when they actually took the vaccine. 389 individuals who took the vaccine were mistakenly classified as non-takers.

Hyperparameter tuning for K-Nearest Neighbors using GridSearchCV

[66]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix,␣
 ↪classification_report

# Define the parameter grid
```

```python
param_grid_knn = {
    'n_neighbors': [3, 5, 7, 9, 11, 13, 15],
    'weights': ['uniform', 'distance'],
    'metric': ['euclidean', 'manhattan', 'minkowski']
}

# Initialize the K-Nearest Neighbors classifier
knn = KNeighborsClassifier()

# Initialize GridSearchCV
grid_search_knn = GridSearchCV(estimator=knn, param_grid=param_grid_knn, cv=5,␣
 ↪scoring='accuracy', n_jobs=-1)

# Perform the grid search on the resampled training data
grid_search_knn.fit(X_train_resampled_scaled, y_train_resampled)

# Get the best parameters and best score
best_params_knn = grid_search_knn.best_params_
best_score_knn = grid_search_knn.best_score_

print("Best Parameters for KNN:", best_params_knn)
print("Best Accuracy Score for KNN:", best_score_knn)

# Initialize K-Nearest Neighbors Classifier with best parameters
best_knn = KNeighborsClassifier(n_neighbors=best_params_knn['n_neighbors'],
                                weights=best_params_knn['weights'],
                                metric=best_params_knn['metric'])

# Fit the model to the training data
best_knn.fit(X_train_resampled_scaled, y_train_resampled)

# Make predictions on the test set
y_pred_test_best_knn = best_knn.predict(X_test_scaled)

# Print the confusion matrix
print("\nConfusion Matrix (Best KNN):")
print(confusion_matrix(y_test, y_pred_test_best_knn))

# Print the classification report
print("\nClassification Report (Best KNN):")
print(classification_report(y_test, y_pred_test_best_knn))
```

```
Best Parameters for KNN: {'metric': 'manhattan', 'n_neighbors': 3, 'weights':
'distance'}
Best Accuracy Score for KNN: 0.8175280979860234

Confusion Matrix (Best KNN):
```

```
[[2484  913]
 [ 389  556]]

Classification Report (Best KNN):
              precision    recall  f1-score   support

           0       0.86      0.73      0.79      3397
           1       0.38      0.59      0.46       945

    accuracy                           0.70      4342
   macro avg       0.62      0.66      0.63      4342
weighted avg       0.76      0.70      0.72      4342
```

- Let's now utilize the XGBoost model which enables the utilization of ensemble methods and gradient boosting, potentially enhancing predictive accuracy and scalability.

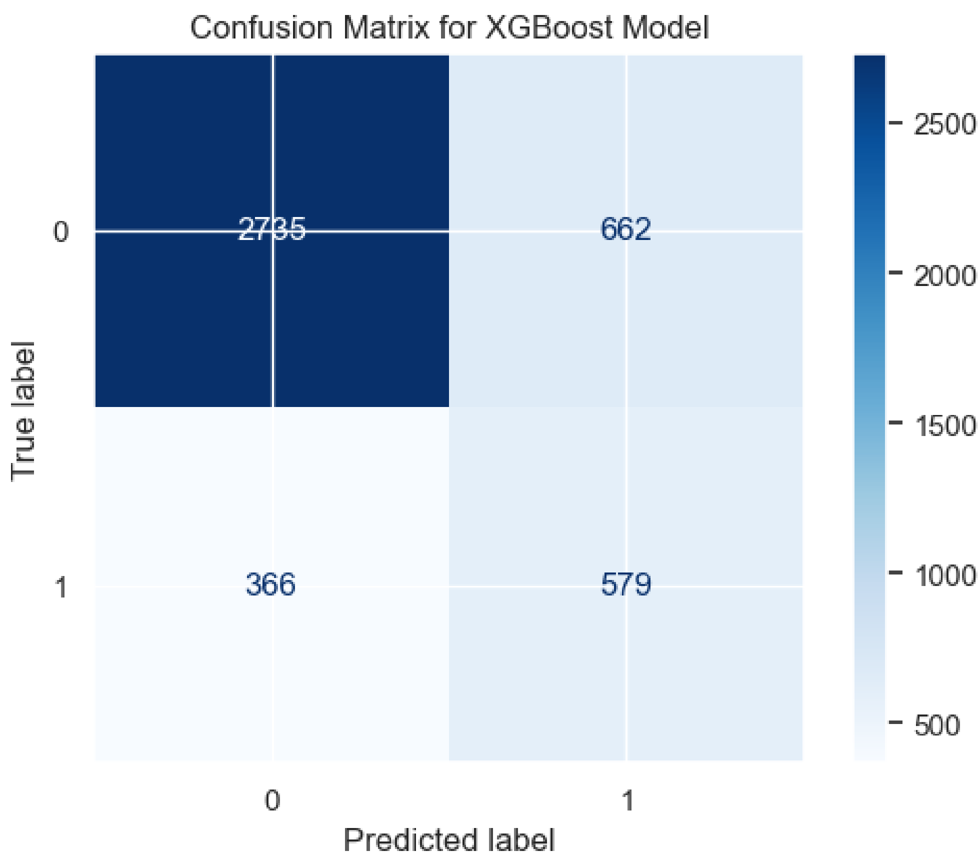
4. XGBOOST model

[67]:
```python
from xgboost import XGBClassifier

# Initialize XGBoost Classifier
xgb_model = XGBClassifier(random_state=42)

# Fit the model to the training data
xgb_model.fit(X_train_resampled_scaled, y_train_resampled)

# Make predictions on the test set
y_pred_test_xgb = xgb_model.predict(X_test_scaled)

# Calculate accuracy on the test set
test_accuracy_xgb = accuracy_score(y_test, y_pred_test_xgb)
print("Test Accuracy (XGBoost Model):", test_accuracy_xgb)

# Print the confusion matrix
print("\nConfusion Matrix (XGBoost Model):")
print(confusion_matrix(y_test, y_pred_test_xgb))

# Print the classification report
print("\nClassification Report (XGBoost Model):")
print(classification_report(y_test, y_pred_test_xgb))
```

```
Test Accuracy (XGBoost Model): 0.7632427452786734

Confusion Matrix (XGBoost Model):
[[2735  662]
 [ 366  579]]

Classification Report (XGBoost Model):
```

```
              precision    recall  f1-score   support

           0       0.88      0.81      0.84      3397
           1       0.47      0.61      0.53       945

    accuracy                           0.76      4342
   macro avg       0.67      0.71      0.69      4342
weighted avg       0.79      0.76      0.77      4342
```

[68]:
```python
from sklearn.metrics import ConfusionMatrixDisplay

# Calculate confusion matrix for XGBoost model
cnf_matrix_xgb = confusion_matrix(y_test, y_pred_test_xgb)

# Plot confusion matrix
disp_xgb = ConfusionMatrixDisplay(confusion_matrix=cnf_matrix_xgb,␣
 ↪display_labels=xgb_model.classes_)
disp_xgb.plot(cmap=plt.cm.Blues)
plt.title('Confusion Matrix for XGBoost Model')
plt.show()
```



- **True Positives:** Instances where the model correctly predicted individuals who opted for the H1N1 vaccine. In this case, there are 579 individuals who took the vaccine and were correctly identified by the model.
- **True Negatives:** Instances where the model correctly predicted individuals who did not opt for the H1N1 vaccine. In this case 2735 individuals who did not take the vaccine were correctly identified by the model.
- **False Positives:** Instances where the model incorrectly predicted individuals as vaccine takers when they did not. 366 individuals were falsely classified as vaccine takers.
- **False Negatives:** Instances where the model incorrectly predicted individuals as non-vaccine takers when they actually took the vaccine. 662 individuals who took the vaccine were mistakenly classified as non-takers.

Hyperparameter tuning for XGBOOST Classifier using GridSearchCV

[69]:
```
from xgboost import XGBClassifier

# Initialize XGBoost Classifier
xgb_model = XGBClassifier(random_state=42)

# Define parameter grid
param_grid_xgb = {
    'max_depth': [3, 4, 5],
    'learning_rate': [0.1, 0.01, 0.001],
    'n_estimators': [100, 200, 300],
    'gamma': [0, 0.1, 0.2],
    'min_child_weight': [1, 3, 5],
    'subsample': [0.5, 0.7, 0.9],
    'colsample_bytree': [0.5, 0.7, 0.9]
}

# Perform Grid Search with Cross-Validation
grid_search_xgb = GridSearchCV(estimator=xgb_model, param_grid=param_grid_xgb,␣
 ↪cv=3, scoring='accuracy', n_jobs=-1, verbose=2)

# Fit the Grid Search to the data
grid_search_xgb.fit(X_train_resampled_scaled, y_train_resampled)

# Print the best hyperparameters
print("Best hyperparameters:", grid_search_xgb.best_params_)

# Get the best model
best_xgb_model = grid_search_xgb.best_estimator_

# Make predictions on the test set using the best model
y_pred_test_xgb = best_xgb_model.predict(X_test_scaled)
```

```python
# Calculate accuracy on the test set using the best model
test_accuracy_xgb = accuracy_score(y_test, y_pred_test_xgb)
print("Test Accuracy (Best XGBoost Model):", test_accuracy_xgb)

# Print the confusion matrix
print("\nConfusion Matrix (Best XGBoost Model):")
print(confusion_matrix(y_test, y_pred_test_xgb))

# Print the classification report
print("\nClassification Report (Best XGBoost Model):")
print(classification_report(y_test, y_pred_test_xgb))
```

```
Fitting 3 folds for each of 2187 candidates, totalling 6561 fits
Best hyperparameters: {'colsample_bytree': 0.9, 'gamma': 0.2, 'learning_rate':
0.1, 'max_depth': 5, 'min_child_weight': 1, 'n_estimators': 300, 'subsample':
0.7}
Test Accuracy (Best XGBoost Model): 0.7655458314140949

Confusion Matrix (Best XGBoost Model):
[[2705  692]
 [ 326  619]]

Classification Report (Best XGBoost Model):
              precision    recall  f1-score   support

           0       0.89      0.80      0.84      3397
           1       0.47      0.66      0.55       945

    accuracy                           0.77      4342
   macro avg       0.68      0.73      0.70      4342
weighted avg       0.80      0.77      0.78      4342
```

1. LOGISTIC REGRESSION MODEL

Accuracy: The model is 76% accurate in predicting vaccine uptake.

Precision (Class 0): 90% of the predictions for not taking the vaccine are correct.

Recall (Class 0): 77% of the instances where the vaccine wasn't taken are identified.

F1-score (Class 0): The balance between precision and recall for class 0 is 83%.

Precision (Class 1): 46% of the predictions for taking the vaccine are accurate.

Recall (Class 1): 71% of the instances where the vaccine was taken are captured.

F1-score (Class 1): The balance between precision and recall for class 1 is 56%.

2. DECISION TREE MODEL

Accuracy: The model is correct 79% of the time.

Precision (Class 0): 85% of the predictions for not taking the vaccine are accurate.

Recall (Class 0): 75% of the actual instances where the vaccine wasn't taken are identified.

F1-score (Class 0): The balance between precision and recall for class 0 is 80%.

Precision (Class 1): Only 37% of the predictions for taking the vaccine are accurate.

Recall (Class 1): 53% of the actual instances where the vaccine was taken are captured.

F1-score (Class 1): The balance between precision and recall for class 1 is 43%, lower than class 0.

3. K-Nearest Neighbours

Accuracy: The model is 81% accurate in predicting vaccine uptake.

Precision (Class 0): 90% of the predictions for not taking the vaccine are correct

Recall (Class 0): 77% of the instances where the vaccine wasn't taken are identified

F1-score (Class 0): The balance between precision and recall for class 0 is 83%

Precision (Class 1): 46% of the predictions for taking the vaccine are accurate

Recall (Class 1): 71% of the instances where the vaccine was taken are captured.

4. XGBOOST model

Accuracy: The XGBoost model achieves an accuracy of 77% in predicting vaccine uptake.

Precision (Class 0): 89% of the predictions for not taking the vaccine are correct.

Recall (Class 0): 80% of the instances where the vaccine wasn't taken are identified.

F1-score (Class 0): The balance between precision and recall for class 0 is 84%.

Precision (Class 1): 47% of the predictions for taking the vaccine are accurate.

Recall (Class 1): 66% of the instances where the vaccine was taken are captured.

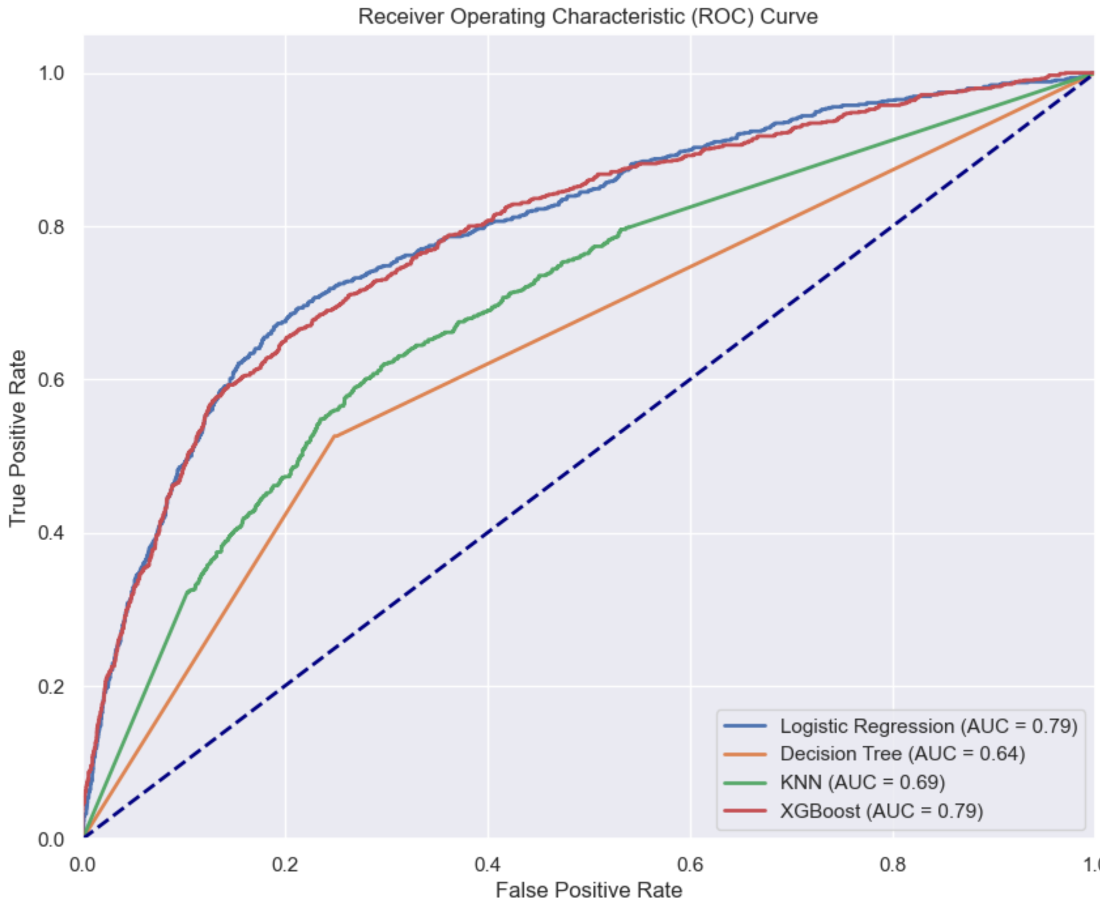
F1-score (Class 1): The balance between precision and recall for class 1 is 55%.

Creating a single plot for all AUC-ROC curves

[70]:
```
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

# List of models
models = {
    'Logistic Regression': best_logistic,
    'Decision Tree': best_decision_tree,
    'KNN': best_knn,
    'XGBoost': best_xgb_model
}

# Initialize a figure for the ROC curves
plt.figure(figsize=(10, 8))
```

```python
for model_name, model in models.items():
    # Calculate predicted probabilities
    y_prob_test = model.predict_proba(X_test_scaled)[:, 1]

    # Calculate ROC curve
    fpr, tpr, _ = roc_curve(y_test, y_prob_test)

    # Calculate AUC score
    roc_auc = auc(fpr, tpr)
    print(f"\nROC AUC Score ({model_name}):", roc_auc)

    # Plot ROC curve
    plt.plot(fpr, tpr, lw=2, label=f'{model_name} (AUC = {roc_auc:.2f})')

# Plot the diagonal line for random guessing
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')

# Configure plot
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc="lower right")
plt.show()
```

```
ROC AUC Score (Logistic Regression): 0.7925522519870474

ROC AUC Score (Decision Tree): 0.6380327802464983

ROC AUC Score (KNN): 0.6945013729823857

ROC AUC Score (XGBoost): 0.7882484233676461
```



Logistic Regression * ROC AUC Score: 0.7925522519870474 * An ROC AUC score of approximately 0.793 indicates a good level of discrimination, meaning the model is capable of distinguishing between positive and negative classes.

Decision Tree * ROC AUC Score: 0.6380327802464983 * An ROC AUC score of approximately 0.638 suggests that the decision tree is not performing very well. It indicates a relatively low ability to distinguish between the positive and negative classes, likely due to overfitting or the inherent limitations of the model with the given dataset.

K-Nearest Neighbors (KNN) * ROC AUC Score: 0.6945013729823857 * An ROC AUC score of approximately 0.695 indicates moderate performance. While better than the decision tree, KNN's performance is still not on par with logistic regression.

XGBoost * ROC AUC Score: 0.7882484233676461 * An ROC AUC score of approximately 0.788 is quite close to that of logistic regression, indicating strong performance.

- Both Logistic Regression and XGBoost have ROC AUC scores close to 0.8, indicating strong discriminatory power. Logistic regression slightly edges out XGBoost, but both are significantly better than the decision tree and KNN.

[71]:
```python
from sklearn.metrics import confusion_matrix, classification_report

# Confusion matrix and classification report for Logistic Regression
print("Confusion Matrix (Logistic Regression):")
print(confusion_matrix(y_test, y_pred_test_best))
print("\nClassification Report (Logistic Regression):")
print(classification_report(y_test, y_pred_test_best))

# Confusion matrix and classification report for XGBoost
print("\nConfusion Matrix (XGBoost):")
print(confusion_matrix(y_test, y_pred_test_xgb))
print("\nClassification Report (XGBoost):")
print(classification_report(y_test, y_pred_test_xgb))
```

```
Confusion Matrix (Logistic Regression):
[[2619  778]
 [ 278  667]]

Classification Report (Logistic Regression):
              precision    recall  f1-score   support

           0       0.90      0.77      0.83      3397
           1       0.46      0.71      0.56       945

    accuracy                           0.76      4342
   macro avg       0.68      0.74      0.70      4342
weighted avg       0.81      0.76      0.77      4342


Confusion Matrix (XGBoost):
[[2705  692]
 [ 326  619]]

Classification Report (XGBoost):
              precision    recall  f1-score   support

           0       0.89      0.80      0.84      3397
           1       0.47      0.66      0.55       945

    accuracy                           0.77      4342
   macro avg       0.68      0.73      0.70      4342
weighted avg       0.80      0.77      0.78      4342
```

Logistic Regression:

- Has a higher recall for Class 1 (0.71), indicating it is better at identifying actual positive cases.

- Has a lower precision for Class 1 (0.46), meaning more false positives.
- It has fewer false negatives (278) compared to XGBoost, but more false positives (778).

XGBoost:

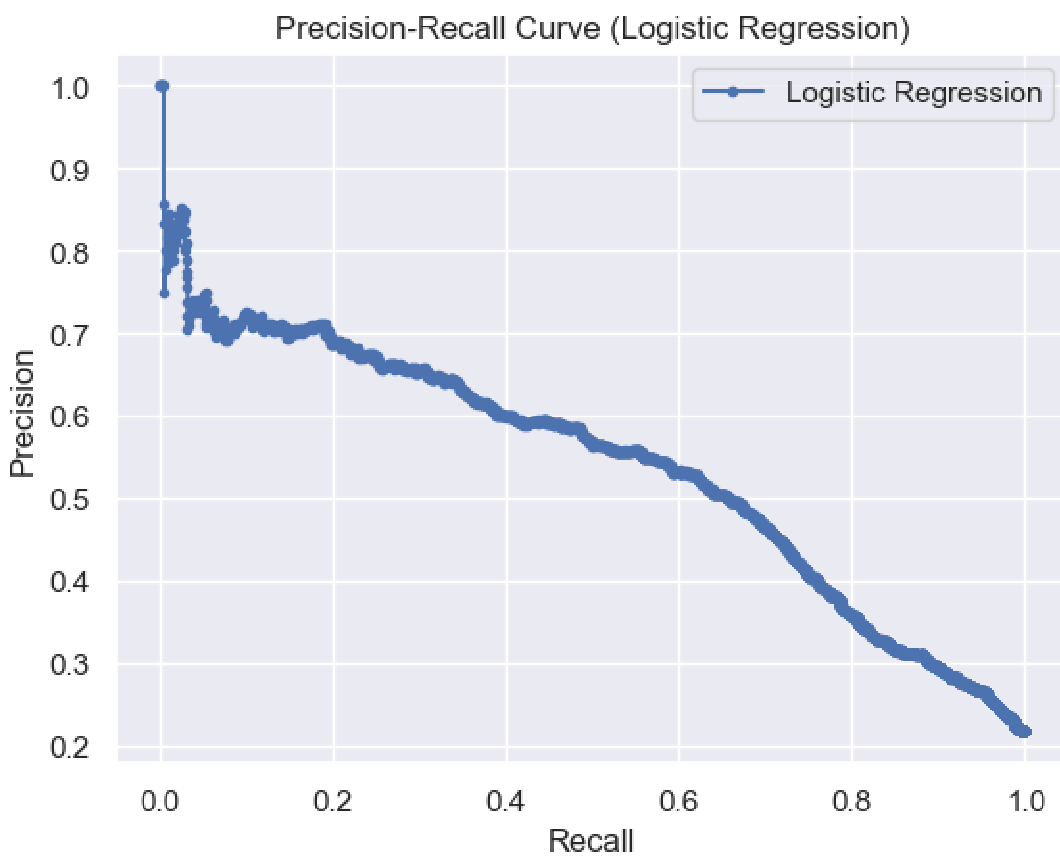
- Has a higher recall for Class 0 (0.80), slightly better overall accuracy (0.77), and higher F1-score for Class 0 (0.84).
- Has a lower recall for Class 1 (0.66) compared to logistic regression, indicating it misses more actual positive cases.
- It has more false negatives (326) compared to logistic regression, but fewer false positives (692).

Logistic Regression is better at identifying actual positive cases of vaccine uptake (higher recall for Class 1). XGBoost provides a slightly better overall accuracy and F1-score for the class 0, but misses more positive cases. Given the importance of capturing as many positive cases as possible in this context, Logistic Regression may be more suitable for ensuring higher recall in identifying individuals who took the vaccine.

Calculating precision and recall at Optimum threshold Logistic Regression

[79]:
```
# Calculate precision and recall for different thresholds (Logistic Regression)
y_pred_proba_logistic = best_logistic.predict_proba(X_test_scaled)[:, 1]
precision_logreg, recall_logreg, thresholds_logreg =␣
 ↪precision_recall_curve(y_test, y_pred_proba_logistic)

# Plot precision-recall curve (Logistic Regression)
plt.plot(recall_logreg, precision_logreg, marker='.', label='Logistic␣
 ↪Regression')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve (Logistic Regression)')
plt.grid(True)
plt.legend()
plt.show()

# Find the F1-score-maximizing threshold (Logistic Regression)
f1_scores_logreg = 2 * (precision_logreg * recall_logreg) / (precision_logreg +␣
 ↪recall_logreg)
optimal_threshold_index_logreg = f1_scores_logreg.argmax()
optimal_threshold_logreg = thresholds_logreg[optimal_threshold_index_logreg]

# Evaluate model performance at the optimal threshold (Logistic Regression)
y_pred_optimal_threshold_logreg = (y_pred_proba_logistic >␣
 ↪optimal_threshold_logreg).astype(int)
precision_optimal_logreg = precision_logreg[optimal_threshold_index_logreg]
recall_optimal_logreg = recall_logreg[optimal_threshold_index_logreg]
f1_score_optimal_logreg = f1_scores_logreg[optimal_threshold_index_logreg]

print("\nLogistic Regression (Optimal Threshold):")
```

```
print("Optimal Threshold:", optimal_threshold_logreg)
print("Precision at Optimal Threshold:", precision_optimal_logreg)
print("Recall at Optimal Threshold:", recall_optimal_logreg)
print("F1 Score at Optimal Threshold:", f1_score_optimal_logreg)
```

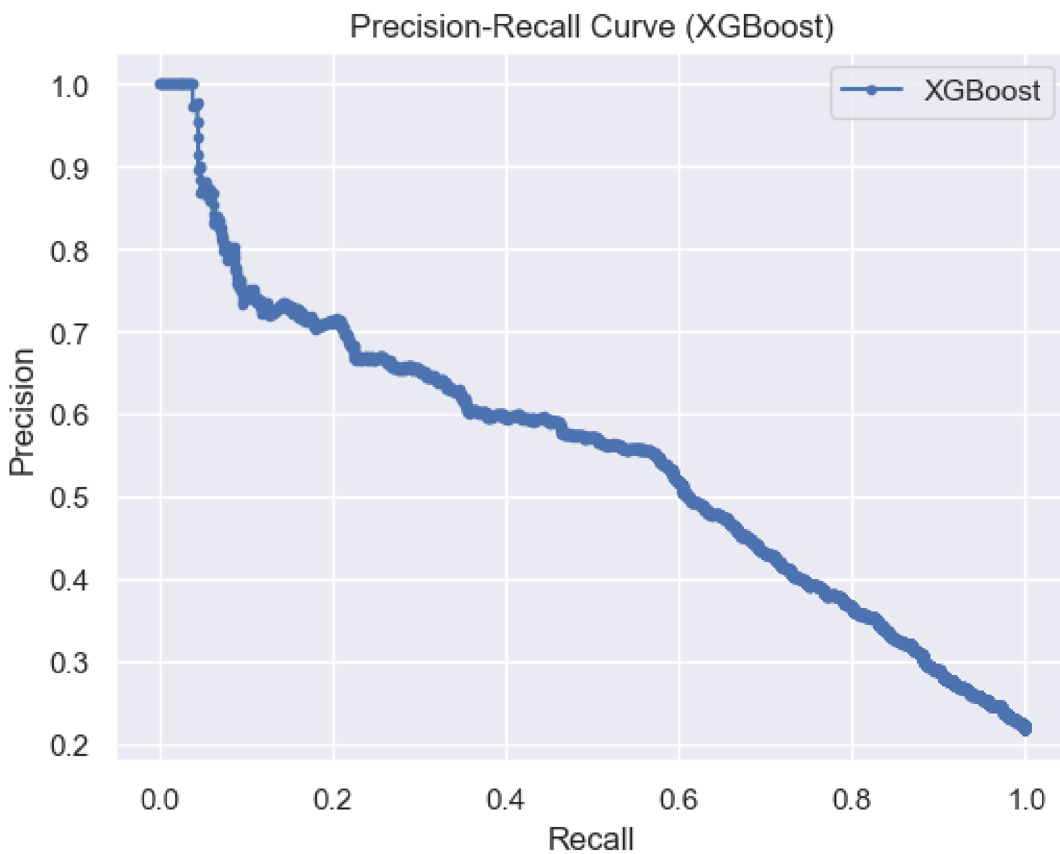



```
Logistic Regression (Optimal Threshold):
Optimal Threshold: 0.5888149079485616
Precision at Optimal Threshold: 0.5288288288288289
Recall at Optimal Threshold: 0.6211640211640211
F1 Score at Optimal Threshold: 0.5712895377128954
```

XGBoost

[81]:
```
# Predict probabilities for the precision-recall curve
y_pred_proba_xgb = best_xgb_model.predict_proba(X_test_scaled)[:, 1]

# Calculate precision and recall for different thresholds (XGBoost)
precision_xgb, recall_xgb, thresholds_xgb = precision_recall_curve(y_test,␣
↪y_pred_proba_xgb)
```

```python
# Plot precision-recall curve (XGBoost)
plt.plot(recall_xgb, precision_xgb, marker='.', label='XGBoost')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve (XGBoost)')
plt.grid(True)
plt.legend()
plt.show()

# Find the F1-score-maximizing threshold (XGBoost)
f1_scores_xgb = 2 * (precision_xgb * recall_xgb) / (precision_xgb + recall_xgb)
optimal_threshold_index_xgb = f1_scores_xgb.argmax()
optimal_threshold_xgb = thresholds_xgb[optimal_threshold_index_xgb]

# Evaluate model performance at the optimal threshold (XGBoost)
y_pred_optimal_threshold_xgb = (y_pred_proba_xgb > optimal_threshold_xgb).
↪astype(int)
precision_optimal_xgb = precision_xgb[optimal_threshold_index_xgb]
recall_optimal_xgb = recall_xgb[optimal_threshold_index_xgb]
f1_score_optimal_xgb = f1_scores_xgb[optimal_threshold_index_xgb]

print("\nXGBoost (Optimal Threshold):")
print("Optimal Threshold:", optimal_threshold_xgb)
print("Precision at Optimal Threshold:", precision_optimal_xgb)
print("Recall at Optimal Threshold:", recall_optimal_xgb)
print("F1 Score at Optimal Threshold:", f1_score_optimal_xgb)
```




```
XGBoost (Optimal Threshold):
Optimal Threshold: 0.60837084
Precision at Optimal Threshold: 0.5531697341513292
Recall at Optimal Threshold: 0.5724867724867725
F1 Score at Optimal Threshold: 0.56266250650026
```

- We will now use the optimal thresholds above to get the final predictions.

[82]:
```
# Get final predictions
y_pred_final_logistic = (y_pred_proba_logistic > 0.5888).astype(int)

# Evaluate model performance with the optimal threshold
print("Final Classification Report (Logistic Regression):")
print(classification_report(y_test, y_pred_final_logistic))

# Calculate accuracy
accuracy_logistic = accuracy_score(y_test, y_pred_final_logistic)
# Print accuracy
print("Accuracy (Logistic Regression):", accuracy_logistic)
```

```
Final Classification Report (Logistic Regression):
              precision    recall  f1-score   support

           0       0.89      0.85      0.87      3397
           1       0.53      0.62      0.57       945

    accuracy                           0.80      4342
   macro avg       0.71      0.73      0.72      4342
weighted avg       0.81      0.80      0.80      4342

Accuracy (Logistic Regression): 0.7970981114693689
```

[84]:

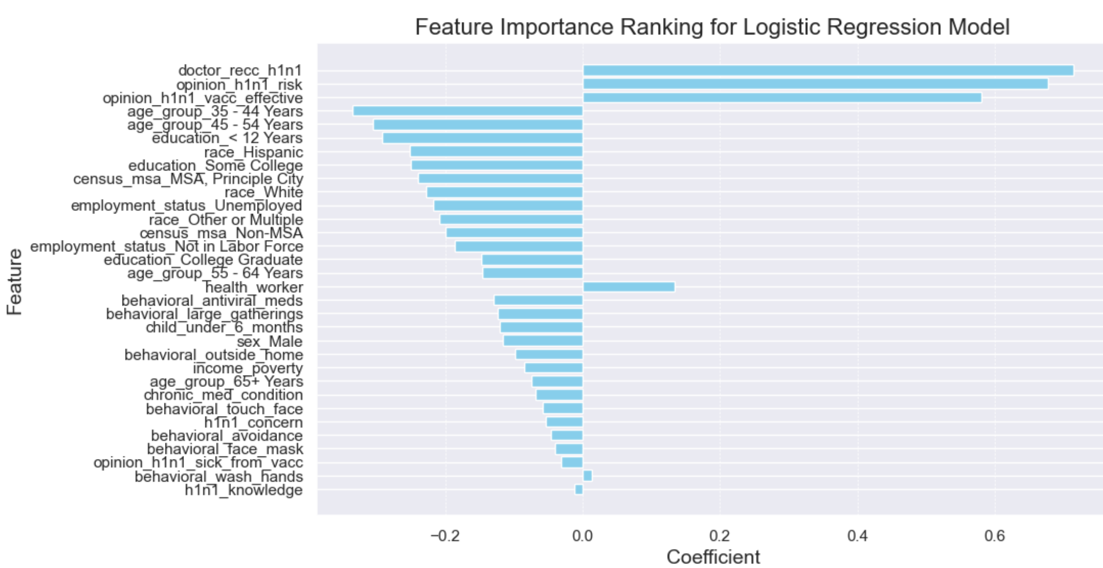
```
# Apply the optimal threshold to get final predictions
y_pred_final_xgb = (y_pred_proba_xgb > 0.6084).astype(int)

# Evaluate the model performance with the optimal threshold
print("Final Classification Report (XGBoost):")
print(classification_report(y_test, y_pred_final_xgb))

# Calculate accuracy
accuracy_xgboost = accuracy_score(y_test, y_pred_final_xgb)

# Print accuracy
print("Accuracy (XGBoost):", accuracy_xgboost)
```

```
Final Classification Report (XGBoost):
              precision    recall  f1-score   support

           0       0.88      0.87      0.88      3397
           1       0.55      0.57      0.56       945

    accuracy                           0.81      4342
   macro avg       0.72      0.72      0.72      4342
weighted avg       0.81      0.81      0.81      4342

Accuracy (XGBoost): 0.8060801473975127
```

- Logistic Regression appears to be the more suitable model for predicting H1N1 vaccine uptake.
- It provides a balanced performance with decent precision and recall for both classes, resulting in a high overall accuracy.
- The logistic regression model exhibits a higher recall for the positive class (1) compared to the XGBoost model (0.62 vs. 0.57).
- This indicates that the logistic regression model is better at correctly identifying individuals who are likely to uptake the H1N1 vaccine, which is crucial for this project.
- Therefore, considering the importance of recall in health-related tasks, the logistic regression model may be preferred over the XGBoost model in this scenario.

1.15 FEATURE IMPORTANCE

[90]:
```python
# Step 2: Calculate feature importance
coefficients = log_model.coef_[0]
features = X_train.columns  # Get feature names from the original DataFrame

# Create a DataFrame for feature importance
importance_df_logreg = pd.DataFrame({
    'Feature': features,
    'Coefficient': coefficients
})

# Sort the DataFrame by the absolute value of coefficients
importance_df_logreg = importance_df_logreg.reindex(importance_df_logreg.Coefficient.abs().sort_values(ascending=False).index)

# Step 3: Plot feature importance
plt.figure(figsize=(10, 6))
plt.barh(importance_df_logreg['Feature'], importance_df_logreg['Coefficient'], color='skyblue')
plt.xlabel('Coefficient', fontsize=14)
plt.ylabel('Feature', fontsize=14)
plt.title('Feature Importance Ranking for Logistic Regression Model', fontsize=16)
plt.gca().invert_yaxis()

# Adjust the width of the grid lines
plt.grid(axis='x', linestyle='--', linewidth=0.5)

plt.show()
```



- The feature importance ranking for predicting H1N1 vaccine uptake highlights several crucial factors.
- At the forefront is the impact of a doctor's recommendation, indicating that individuals are more inclined to receive the vaccine when advised by their healthcare provider. Close behind is the respondent's perception of the risk associated with contracting H1N1 flu without vaccination, emphasizing the role of perceived susceptibility in vaccine decision-making. Additionally, the belief in the effectiveness of the H1N1 vaccine emerges as another influential factor, suggesting that confidence in vaccine efficacy positively influences uptake. Furthermore, being a healthcare worker and practicing frequent hand hygiene through handwashing or sanitizer use also significantly contribute to vaccine uptake.

1.16 RECOMMENDATIONS

1. Promote Doctor Recommendations: Encourage healthcare providers to actively recommend the H1N1 vaccine to their patients.
2. Public health messaging should emphasize the health risks associated with contracting H1N1 flu without vaccination.
3. Address Perceived Risks: public health messaging should address and clarify any misconceptions or concerns regarding the perceived risks associated with H1N1 vaccinations.
4. Strengthened awareness and communication: Strengthened communication about the effectiveness of the H1N1 vaccine to enhance public confidence and trust in its efficacy would positively influence uptake rates.
5. Target Health Workers: Promote vaccination among healthcare workers and emphasize the importance of their role as advocates for vaccination within their communities.
6. Behavioural interventions: Promoting preventive behaviors such as frequent hand hygiene practices and avoiding large gatherings in high-risk situations.
7. Implementation of targeted interventions to address underlying concerns related to vaccine safety and side effects, aiming to increase confidence in vaccination among hesitant individuals.s.

1.17 NEXT STEPS

1. Training: Effective training of healthcare providers to effectively communicate the importance and benefits of H1N1 vaccination to their patients.
2. Further research: Further research can be conducted to understand the specific reasons behind negative associations with certain predictors, allowing for tailored interventions to address barriers to vaccine acceptance and uptake.
3. Monitoring and evaluation: Monitor and evaluate the impact of implemented strategies on vaccine acceptance and uptake rates, adjusting approaches as needed for continuous and sustainable improvement.

[ ]:
```

```