Final Project Submission

Please fill out:

- Student's name: Wachuka Kinyanjui
- Student pace: Part time
- Scheduled project review date/time:
- Instructor name:
  1. Noah Kandie
  2. William Okomba

INTRODUCTION

As the world struggles to vaccinate the global population against COVID-19, understanding how people's backgrounds, opinions, and health behaviors are related to their personal vaccination patterns can provide invaluable guidance for future public health efforts. By analyzing these factors, public health officials can better design strategies that encourage vaccination and improve overall public health outcomes.

This project focuses on a binary classification problem: predicting whether individuals received the H1N1 or seasonal flu vaccines using data from the National 2009 H1N1 Flu Survey. This dataset includes information on various demographic factors, health behaviors, and personal opinions. For this minimum viable project, we will choose one of the two potential targets—whether the survey respondent received the H1N1 flu vaccine.

By examining the relationships between the provided features and vaccination outcomes, we aim to identify key predictors of vaccination behavior. These insights can help tailor public health campaigns, improve vaccine uptake, and ultimately enhance public health preparedness for future pandemics.

BUSINESS UNDERSTANDING

In the context of the ongoing COVID-19 pandemic, understanding the drivers behind vaccine uptake is crucial. By leveraging historical data from the 2009 H1N1 flu pandemic, public health officials can gain insights into how various factors such as personal backgrounds, opinions, and health behaviors influence vaccination decisions. These insights can then be applied to enhance current and future vaccination efforts for COVID-19 and other diseases.

Specific Objectives:

1. Identify Key Predictors: Determine which factors (e.g., age, gender, education level, health behaviors, trust in healthcare) are most strongly associated with receiving the H1N1 vaccine.
2. Develop a Predictive Model: Create a binary classification model to predict whether an individual received the H1N1 vaccine based on their survey responses.
3. Inform Public Health Strategies: Use the model’s insights to guide public health campaigns, improving vaccine uptake by addressing barriers and leveraging motivators identified through the data. By focusing on predicting the uptake of the H1N1 vaccine, this project aims to provide actionable recommendations that can be applied to enhance vaccination rates and protect public health in current and future pandemics.

BUSINESS PROBLEM

The core business problem involves predicting vaccination uptake among a population based on demographic, attitudinal, and health-related data. Specifically, the challenge is to develop a predictive model that can identify whether individuals received a particular vaccine (in this case, the H1N1 flu vaccine) using data from the National 2009 H1N1 Flu Survey. Accurate predictions can inform public health strategies by highlighting key factors that influence vaccine acceptance, ultimately aiding in the design and implementation of more effective vaccination campaigns.

Importing Libraries

In [1]:
```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
import seaborn as sns

import statsmodels.api as sm
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.datasets import make_regression
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import LogisticRegression
import sklearn.metrics as metrics
from sklearn.impute import SimpleImputer
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.preprocessing import PolynomialFeatures
from random import gauss
from sklearn.metrics import classification_report, accuracy_score
from mpl_toolkits.mplot3d import Axes3D
from sklearn.model_selection import StratifiedKFold, cross_val_score
from scipy import stats as stats
from imblearn.over_sampling import SMOTE
from imblearn.pipeline import Pipeline as ImbPipeline
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import StackingClassifier
%matplotlib inline
```

Loading the Dataset

In [2]:
```
f = r"C:\Users\WACHUKA\Documents\FLATIRON\PHASE 3 PROJECT\training_set_features.csv"
t = r"C:\Users\WACHUKA\Documents\FLATIRON\PHASE 3 PROJECT\training_set_labels.csv"
```

In [3]:
```
features = pd.read_csv(f)
labels = pd.read_csv(t)
```

Features Evaluation

In [4]:
```
features.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
```

In [5]:
```
features.columns
```

Out[5]:
```
Index(['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', 'doctor_recc_seasonal',
       'chronic_med_condition', 'child_under_6_months', 'health_worker',
       'health_insurance', 'opinion_h1n1_vacc_effective', 'opinion_h1n1_risk',
       'opinion_h1n1_sick_from_vacc', 'opinion_seas_vacc_effective',
       'opinion_seas_risk', 'opinion_seas_sick_from_vacc', 'age_group',
       'education', 'race', 'sex', 'income_poverty', 'marital_status',
       'rent_or_own', 'employment_status', 'hhs_geo_region', 'census_msa',
       'household_adults', 'household_children', 'employment_industry',
       'employment_occupation'],
      dtype='object')
```

In [6]:
```
features.head()
```

Out[6]:

poverty	marital_status	rent_or_own	employment_status	hhs_geo_region	census_msa	household_adults	household_children	em
Poverty	Not Married	Own	Not in Labor Force	oxchjgsf	Non-MSA	0.0	0.0	
Poverty	Not Married	Rent	Employed	bhuqouqj	MSA, Not Principle City	0.0	0.0	
$75,000, Poverty	Not Married	Own	Employed	qufhixun	MSA, Not Principle City	2.0	0.0	
Poverty	Not Married	Rent	Not in Labor Force	lrircsnp	MSA, Principle City	0.0	0.0	
$75,000, Poverty	Married	Own	Employed	qufhixun	MSA, Not Principle City	1.0	0.0	

In [7]:
```
features.tail()
```

Out[7]:

	respondent_id	h1n1_concern	h1n1_knowledge	behavioral_antiviral_meds	behavioral_avoidance	behavioral_face_mask	b
26702	26702	2.0	0.0	0.0	1.0	0.0	
26703	26703	1.0	2.0	0.0	1.0	0.0	
26704	26704	2.0	2.0	0.0	1.0	1.0	
26705	26705	1.0	1.0	0.0	0.0	0.0	
26706	26706	0.0	0.0	0.0	1.0	0.0	

5 rows × 36 columns

In [8]:
```
features.describe()
```

Out[8]:

	respondent_id	h1n1_concern	h1n1_knowledge	behavioral_antiviral_meds	behavioral_avoidance	behavioral_face_mask	b
count	26707.000000	26615.000000	26591.000000	26636.000000	26499.000000	26688.000000	
mean	13353.000000	1.618486	1.262532	0.048844	0.725612	0.068982	
std	7709.791156	0.910311	0.618149	0.215545	0.446214	0.253429	
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	6676.500000	1.000000	1.000000	0.000000	0.000000	0.000000	
50%	13353.000000	2.000000	1.000000	0.000000	1.000000	0.000000	
75%	20029.500000	2.000000	2.000000	0.000000	1.000000	0.000000	
max	26706.000000	3.000000	2.000000	1.000000	1.000000	1.000000	

8 rows × 24 columns

In [9]:
```
features.shape
```

Out[9]: (26707, 36)

Each row is a person who was a survey respondent. The columns are the feature values corresponding to those people. We have 26,707 observations and 35 features.

In [10]:
```
features.dtypes
```

Out[10]:
```
respondent_id                    int64
h1n1_concern                   float64
h1n1_knowledge                 float64
behavioral_antiviral_meds      float64
behavioral_avoidance           float64
behavioral_face_mask           float64
behavioral_wash_hands          float64
behavioral_large_gatherings    float64
behavioral_outside_home        float64
behavioral_touch_face          float64
doctor_recc_h1n1               float64
doctor_recc_seasonal           float64
chronic_med_condition          float64
child_under_6_months           float64
health_worker                  float64
health_insurance               float64
opinion_h1n1_vacc_effective    float64
opinion_h1n1_risk              float64
opinion_h1n1_sick_from_vacc    float64
opinion_seas_vacc_effective    float64
opinion_seas_risk              float64
opinion_seas_sick_from_vacc    float64
age_group                       object
education                       object
race                            object
sex                             object
income_poverty                  object
marital_status                  object
rent_or_own                     object
employment_status               object
hhs_geo_region                  object
census_msa                      object
household_adults               float64
household_children             float64
employment_industry             object
employment_occupation           object
dtype: object
```

Labels Evaluation

In [7]:
```
labels.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 26707 entries, 0 to 26706
Data columns (total 3 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   respondent_id     26707 non-null  int64
 1   h1n1_vaccine      26707 non-null  int64
 2   seasonal_vaccine  26707 non-null  int64
dtypes: int64(3)
memory usage: 626.1 KB
```

In [8]:
```
labels.head()
```

Out[8]:

	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

In [13]:
```
labels.tail()
```

Out[13]:

	respondent_id	h1n1_vaccine	seasonal_vaccine
26702	26702	0	0
26703	26703	0	0
26704	26704	0	1
26705	26705	0	0
26706	26706	0	0

In [66]:
```
labels.describe()
```

Out[66]:

	respondent_id	h1n1_vaccine	seasonal_vaccine
count	26707.000000	26707.000000	26707.000000
mean	13353.000000	0.212454	0.465608
std	7709.791156	0.409052	0.498825
min	0.000000	0.000000	0.000000
25%	6676.500000	0.000000	0.000000
50%	13353.000000	0.000000	0.000000
75%	20029.500000	0.000000	1.000000
max	26706.000000	1.000000	1.000000

In [15]:
```
labels.shape
```

Out[15]: (26707, 3)

We have the same 26,707 observations, and two target variables that we have labels for

Let's double-check that the rows between the features and the labels match up. We don't want to have the wrong labels. Numpy's assert_array_equal will error if the two arrays—the row indices of the two data frames—don't match up.

In [16]:
```
np.testing.assert_array_equal(features.index.values, labels.index.values)
```

DATA CLEANING

In [9]:
```
# To enable retaining of original copy before data cleaning
features_copy = features.copy()
labels_copy = labels.copy()
```

Checking for completeness of our data

In [18]:
```
# Checking for missing values in labels
missing_values = labels.isnull().mean()
missing_values
```

Out[18]:
```
respondent_id       0.0
h1n1_vaccine        0.0
seasonal_vaccine    0.0
dtype: float64
```

In [19]:
```
# Checking the proportion of our missing data in features
features.isnull().mean()
```

Out[19]:
```
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
dtype: float64
```

Handling missing values

In [20]:
```
#Handling missing values by filling the categorical columns with mode and numerical columns with median
for column in features.columns:
    if features[column].dtype == 'object':

        features[column].fillna(features[column].mode()[0], inplace=True)
    else:

        features[column].fillna(features[column].median(), inplace=True)
```

In [21]:
```
#Ascertaining that all missing values have been fully populated
missing_values = features.isnull().sum()
missing_values
```

```
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            0
employment_occupation          0
dtype: int64
```

- We now have no missing values.

In [22]:
```
#Checking for duplicates
features.duplicated().sum()
```

Out[22]: 0

- Let us check for duplicates in th ID column as it is our unique identifier.

In [23]:
```
# Checking for duplicates using the 'respondent_id' column

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

Out[23]:

respondent_id	h1n1_concern	h1n1_knowledge	behavioral_antiviral_meds	behavioral_avoidance	behavioral_face_mask	behavio

0 rows × 36 columns

Checking for placeholders

- Placeholders in data cleaning are values used to represent missing or unknown data in a dataset. They stand in for actual data that is unavailable or not recorded.
- Placeholders include - NaN, Nul , Non, " " , s Special co such as;g., -1, 99 ble" "Mi and others.plicable"

In [13]:

```
potential_placeholders = [" " , "-", "--", "?", "??" , "#","#####" , "-1" , "9999", "999" , "unknown", '

# Loop through each column and check for potential placeholders
found_placeholder = False
for column in features.columns:
    unique_values = features[column].unique()
    for value in unique_values:
        if pd.isna(value) or (isinstance(value, str) and value.strip().lower() in potential_placeholder
            count = (features[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.")
```

```
Column 'h1n1_concern': Found 0 occurrences of potential placeholder 'nan'
Column 'h1n1_knowledge': Found 0 occurrences of potential placeholder 'nan'
Column 'behavioral_antiviral_meds': Found 0 occurrences of potential placeholder 'nan'
Column 'behavioral_avoidance': Found 0 occurrences of potential placeholder 'nan'
Column 'behavioral_face_mask': Found 0 occurrences of potential placeholder 'nan'
Column 'behavioral_wash_hands': Found 0 occurrences of potential placeholder 'nan'
Column 'behavioral_large_gatherings': Found 0 occurrences of potential placeholder 'nan'
Column 'behavioral_outside_home': Found 0 occurrences of potential placeholder 'nan'
Column 'behavioral_touch_face': Found 0 occurrences of potential placeholder 'nan'
Column 'doctor_recc_h1n1': Found 0 occurrences of potential placeholder 'nan'
Column 'doctor_recc_seasonal': Found 0 occurrences of potential placeholder 'nan'
Column 'chronic_med_condition': Found 0 occurrences of potential placeholder 'nan'
Column 'child_under_6_months': Found 0 occurrences of potential placeholder 'nan'
Column 'health_worker': Found 0 occurrences of potential placeholder 'nan'
Column 'health_insurance': Found 0 occurrences of potential placeholder 'nan'
Column 'opinion_h1n1_vacc_effective': Found 0 occurrences of potential placeholder 'nan'
Column 'opinion_h1n1_risk': Found 0 occurrences of potential placeholder 'nan'
Column 'opinion_h1n1_sick_from_vacc': Found 0 occurrences of potential placeholder 'nan'
Column 'opinion_seas_vacc_effective': Found 0 occurrences of potential placeholder 'nan'
Column 'opinion_seas_risk': Found 0 occurrences of potential placeholder 'nan'
Column 'opinion_seas_sick_from_vacc': Found 0 occurrences of potential placeholder 'nan'
Column 'education': Found 0 occurrences of potential placeholder 'nan'
Column 'income_poverty': Found 0 occurrences of potential placeholder 'nan'
Column 'marital_status': Found 0 occurrences of potential placeholder 'nan'
Column 'rent_or_own': Found 0 occurrences of potential placeholder 'nan'
Column 'employment_status': Found 0 occurrences of potential placeholder 'nan'
Column 'household_adults': Found 0 occurrences of potential placeholder 'nan'
Column 'household_children': Found 0 occurrences of potential placeholder 'nan'
Column 'employment_industry': Found 0 occurrences of potential placeholder 'nan'
Column 'employment_occupation': Found 0 occurrences of potential placeholder 'nan'
```

FEATURE ENGINEERING

In [14]:

```
# Identifying the categorical columns
categorical_cols = features.select_dtypes(include=['object']).columns.tolist()

# Perform one-hot encoding by converting categorical columns into binary indicator variables.
encoded_features = pd.get_dummies(features, columns=categorical_cols)

# Display the encoded dataset
print(encoded_features.head())
```

```
   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  ...  employment_occupation_qxajmpny  \
0                    1.0  ...                               0   
1                    1.0  ...                               0   
2                    0.0  ...                               0   
3                    0.0  ...                               0   
4                    1.0  ...                               0   

   employment_occupation_rcertsgn  employment_occupation_tfqavkke  \
0                               0                               0   
1                               0                               0   
2                               0                               0   
3                               0                               0   
4                               0                               0   

   employment_occupation_ukymxvdu  employment_occupation_uqqtjvyb  \
0                               0                               0   
1                               0                               0   
2                               0                               0   
3                               0                               0   
4                               0                               0   

   employment_occupation_vlluhbov  employment_occupation_xgwztkwe  \
0                               0                               0   
1                               0                               1   
2                               0                               0   
3                               0                               0   
4                               0                               0   

   employment_occupation_xqwwgdyp  employment_occupation_xtkaffoo  \
0                               0                               0   
1                               0                               0   
2                               0                               1   
3                               0                               0   
4                               0                               0   

   employment_occupation_xzmlyyjv  
0                               0  
1                               0  
2                               0  
3                               0  
4                               0  

[5 rows x 106 columns]
```

In [15]:

```
# Perform Feature Scaling for numerical features to ensure that the ML models are robust and accurate
numerical_cols = encoded_features.select_dtypes(include=['int', 'float']).columns.tolist()
scaler = StandardScaler()
data_scaled = encoded_features.copy()
data_scaled[numerical_cols] = scaler.fit_transform(encoded_features[numerical_cols])
data_scaled[numerical_cols]
```

Out[15]:

	h1n1_concern	h1n1_knowledge	behavioral_antiviral_meds	behavioral_avoidance	behavioral_face_mask	behavioral_wash_
0	-0.679436	-2.042478	-0.22661	-1.626185	-0.272201	-2.1
1	1.517658	1.193048	-0.22661	0.614936	-0.272201	0.4
2	-0.679436	-0.424715	-0.22661	0.614936	-0.272201	-2.1
3	-0.679436	-0.424715	-0.22661	0.614936	-0.272201	0.4
4	0.419111	-0.424715	-0.22661	0.614936	-0.272201	0.4
...	...	...	...	...	...	
26702	0.419111	-2.042478	-0.22661	0.614936	-0.272201	-2.1
26703	-0.679436	1.193048	-0.22661	0.614936	-0.272201	0.4
26704	0.419111	1.193048	-0.22661	0.614936	3.673754	0.4
26705	-0.679436	-0.424715	-0.22661	-1.626185	-0.272201	-2.1
26706	-1.777982	-2.042478	-0.22661	0.614936	-0.272201	-2.1

26707 rows × 23 columns

In [27]:

```
# Add Polynomial Features
from sklearn.preprocessing import PolynomialFeatures
poly_features = PolynomialFeatures(degree=2, include_bias=False)
data_poly = poly_features.fit_transform(data_scaled[numerical_cols])
data_poly_df = pd.DataFrame(data_poly, columns=poly_features.get_feature_names(numerical_cols))

# Final processed dataset
print(data_poly_df.head())
```

```
   h1n1_concern  h1n1_knowledge  behavioral_antiviral_meds  \
0     -0.681849       -2.044279                  -0.226293
1      1.518373        1.197027                  -0.226293
2     -0.681849       -0.423626                  -0.226293
3     -0.681849       -0.423626                  -0.226293
4      0.418262       -0.423626                  -0.226293

   behavioral_avoidance  behavioral_face_mask  behavioral_wash_hands  \
0             -1.634957             -0.272097              -2.177944
1              0.611637             -0.272097               0.459149
2              0.611637             -0.272097              -2.177944
3              0.611637             -0.272097               0.459149
4              0.611637             -0.272097               0.459149

   behavioral_large_gatherings  behavioral_outside_home  \
0                     -0.74589                 1.404892
1                     -0.74589                 1.404892
2                     -0.74589                -0.711798
3                      1.34068                -0.711798
4                      1.34068                -0.711798

   behavioral_touch_face  doctor_recc_h1n1  ...  opinion_seas_risk^2  \
0               0.687870         -0.503893  ...             1.537754
1               0.687870         -0.503893  ...             0.263056
2              -1.453764         -0.503893  ...             1.537754
3              -1.453764         -0.503893  ...             0.886337
4               0.687870         -0.503893  ...             1.537754

   opinion_seas_risk opinion_seas_sick_from_vacc  \
0                                      0.108765
1                                     -0.732380
2                                      0.108765
3                                     -0.796035
4                                     -1.770744

   opinion_seas_risk household_adults  opinion_seas_risk household_children  \
0                            1.467568                              0.709795
1                            0.606986                              0.293571
2                           -1.839413                              0.709795
3                           -1.114177                             -0.538877
4                           -0.185922                              0.709795

   opinion_seas_sick_from_vacc^2  \
0                       0.007693
1                       2.039035
2                       0.007693
3                       0.714934
4                       2.039035

   opinion_seas_sick_from_vacc household_adults  \
0                                     0.103801
1                                    -1.689924
2                                    -0.130101
3                                     1.000663
4                                     0.214092

   opinion_seas_sick_from_vacc household_children  household_adults^2  \
0                                       0.050204            1.400586
1                                      -0.817339            1.400586
2                                       0.050204            2.200248
3                                       0.483975            1.400586
4                                      -0.817339            0.022479

   household_adults household_children  household_children^2
0                             0.677399              0.327627
1                             0.677399              0.327627
2                            -0.849035              0.327627
3                             0.677399              0.327627
4                            -0.085818              0.327627

[5 rows x 299 columns]
```

Handling outliers

In [16]:

```
numerical_cols = encoded_features.select_dtypes(include=['int', 'float']).columns.tolist()
# Loop through each numeric column
for column in numerical_cols:
    # Calculate IQR
    q1 = features[column].quantile(0.25)
    q3 = features[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 = ((features[column] < lower_bound) | (features[column] > upper_bound)).sum()

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

```
Column: h1n1_concern, Number of outliers: 0
Column: h1n1_knowledge, Number of outliers: 0
Column: behavioral_antiviral_meds, Number of outliers: 1301
Column: behavioral_avoidance, Number of outliers: 0
Column: behavioral_face_mask, Number of outliers: 1841
Column: behavioral_wash_hands, Number of outliers: 4650
Column: behavioral_large_gatherings, Number of outliers: 0
Column: behavioral_outside_home, Number of outliers: 0
Column: behavioral_touch_face, Number of outliers: 0
Column: doctor_recc_h1n1, Number of outliers: 5408
Column: doctor_recc_seasonal, Number of outliers: 0
Column: chronic_med_condition, Number of outliers: 0
Column: child_under_6_months, Number of outliers: 2138
Column: health_worker, Number of outliers: 2899
Column: health_insurance, Number of outliers: 1736
Column: opinion_h1n1_vacc_effective, Number of outliers: 0
Column: opinion_h1n1_risk, Number of outliers: 0
Column: opinion_h1n1_sick_from_vacc, Number of outliers: 0
Column: opinion_seas_vacc_effective, Number of outliers: 3427
Column: opinion_seas_risk, Number of outliers: 0
Column: opinion_seas_sick_from_vacc, Number of outliers: 0
Column: household_adults, Number of outliers: 1125
Column: household_children, Number of outliers: 1747
```

In [29]:

```
# Define a function to handle outliers using IQR method
def handle_outliers_iqr(features, column):
    q1 = features[column].quantile(0.25)
    q3 = features[column].quantile(0.75)
    iqr = q3 - q1
    lower_bound = q1 - 1.5 * iqr
    upper_bound = q3 + 1.5 * iqr
    features[column] = features[column].clip(lower=lower_bound, upper=upper_bound)

# Columns with outliers
outlier_columns = ['behavioral_antiviral_meds', 'behavioral_face_mask', 'behavioral_wash_hands', 'doctor
       'household_adults', 'household_children']

# Apply the handle_outliers_iqr function to each column
for col in outlier_columns:
    handle_outliers_iqr(features, col)
```

- Checking if our outliers have been handled.

In [30]:

```
numerical_cols = encoded_features.select_dtypes(include=['int', 'float']).columns.tolist()

# Loop through each numeric column
for column in numerical_cols:
    # Calculate IQR
    q1 = features[column].quantile(0.25)
    q3 = features[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 = ((features[column] < lower_bound) | (features[column] > upper_bound)).sum()

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

```
Column: h1n1_concern, Number of outliers: 0
Column: h1n1_knowledge, Number of outliers: 0
Column: behavioral_antiviral_meds, Number of outliers: 0
Column: behavioral_avoidance, Number of outliers: 0
Column: behavioral_face_mask, Number of outliers: 0
Column: behavioral_wash_hands, Number of outliers: 0
Column: behavioral_large_gatherings, Number of outliers: 0
Column: behavioral_outside_home, Number of outliers: 0
Column: behavioral_touch_face, Number of outliers: 0
Column: doctor_recc_h1n1, Number of outliers: 0
Column: doctor_recc_seasonal, Number of outliers: 0
Column: chronic_med_condition, Number of outliers: 0
Column: child_under_6_months, Number of outliers: 0
Column: health_worker, Number of outliers: 0
Column: health_insurance, Number of outliers: 0
Column: opinion_h1n1_vacc_effective, Number of outliers: 0
Column: opinion_h1n1_risk, Number of outliers: 0
Column: opinion_h1n1_sick_from_vacc, Number of outliers: 0
Column: opinion_seas_vacc_effective, Number of outliers: 0
Column: opinion_seas_risk, Number of outliers: 0
Column: opinion_seas_sick_from_vacc, Number of outliers: 6573
Column: household_adults, Number of outliers: 0
Column: household_children, Number of outliers: 0
```

EXPLORATORY DATA ANALYSIS

In [9]:

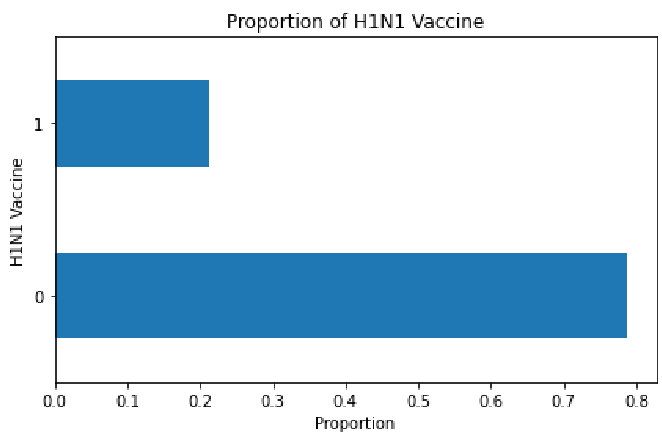
```
# Check the distribution of the H1N1 vaccine target variable
h1n1_distribution = (labels['h1n1_vaccine'].value_counts(normalize=True))

h1n1_distribution
```

Out[9]:

```
0    0.787546
1    0.212454
Name: h1n1_vaccine, dtype: float64
```

Evaluating the distribution of H1N1 Vaccine

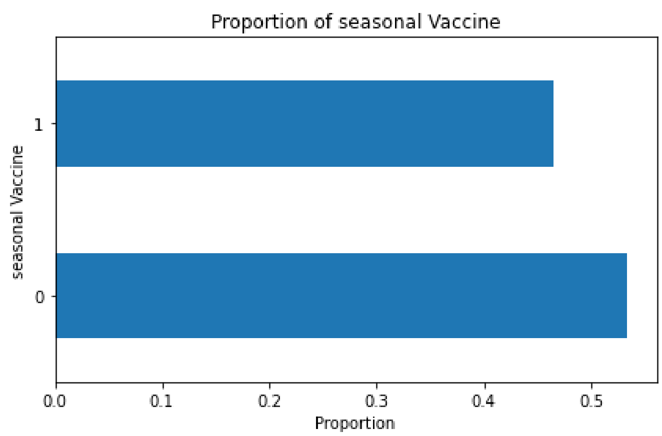
In [17]:
```
# Get the number of observations
n_obs = labels.shape[0]

# Create the plot
fig, ax = plt.subplots()

(labels['h1n1_vaccine']
    .value_counts()
    .div(n_obs)
    .plot.barh(title="Proportion of H1N1 Vaccine", ax=ax)
)

# Set labels
ax.set_ylabel("H1N1 Vaccine")
ax.set_xlabel("Proportion")

# Adjust layout
fig.tight_layout()

# Show plot
plt.show()
```



Evaluating the distribution of seasonal Vaccine

In [33]:

```
# Get the number of observations
n_obs = labels.shape[0]

# Create the plot
fig, ax = plt.subplots()

(labels['seasonal_vaccine']
    .value_counts()
    .div(n_obs)
    .plot.barh(title="Proportion of seasonal Vaccine", ax=ax)
)

# Set labels
ax.set_ylabel("seasonal Vaccine")
ax.set_xlabel("Proportion")

# Adjust layout
fig.tight_layout()

# Show plot
plt.show()
```



It looks like roughy half of people received the seasonal flu vaccine, but only about 20% of people received the H1N1 flu vaccine. In terms of class balance, we say that the seasonal flu vaccine target has balanced classes, but the H1N1 flu vaccine target has moderately imbalanced classes.

Seasonal flu vaccine target has balanced classes, but the H1N1 flu vaccine target has moderately imbalanced classes

Evaluating the Independence of the two Target Variables

In [71]:

```
pd.crosstab(
    labels["h1n1_vaccine"],
    labels["seasonal_vaccine"],
    margins=True,
    normalize=True
)
```

Out[71]:

seasonal_vaccine / h1n1_vaccine	0	1	All
0	0.497810	0.289737	0.787546
1	0.036582	0.175871	0.212454
All	0.534392	0.465608	1.000000

Individuals who did not receive the H1N1 vaccine (row 0) tend to have a higher proportion (0.497810) of not receiving the seasonal flu vaccine (column 0) compared to those who received the H1N1 vaccine (row 1, column 0: 0.036582).

Conversely, individuals who received the H1N1 vaccine (row 1) have a higher proportion (0.175871) of also receiving the seasonal flu vaccine (column 1) compared to those who did not receive the H1N1 vaccine (row 0, column 1: 0.289737).

While a minority of people who got the seasonal vaccine got the H1N1 vaccine, they got the H1N1 vaccine at a higher rate than those who did not get the seasonal vaccine

In [35]:

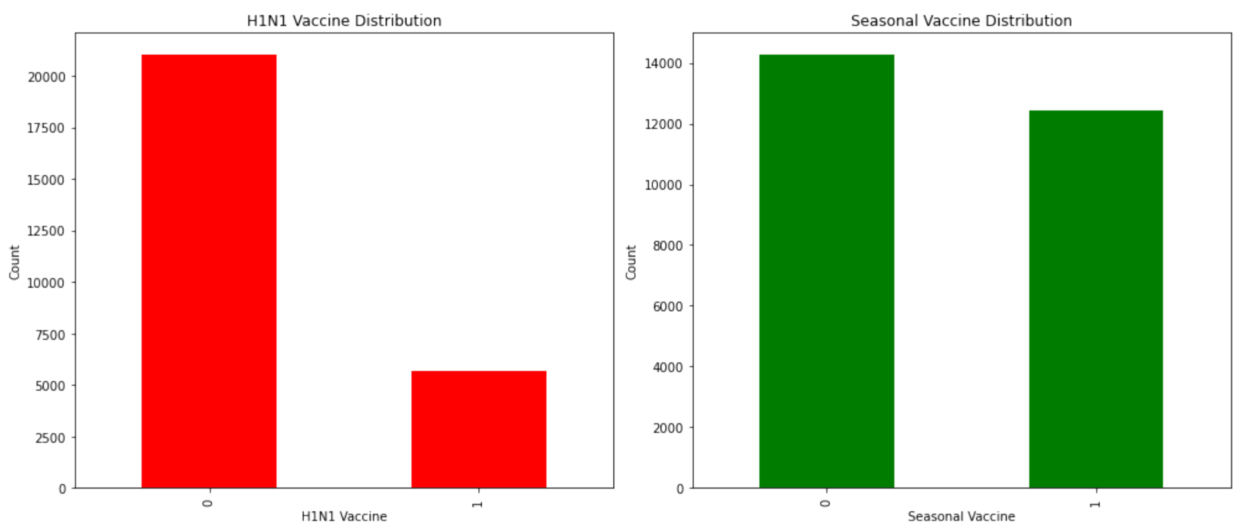
```
# Calculate Phi Coefficient manually
crosstab = pd.crosstab(labels["h1n1_vaccine"], labels["seasonal_vaccine"])
phi_coefficient = np.sqrt(crosstab.div(crosstab.sum(axis=1), axis=0).div(crosstab.sum(axis=0), axis=1))
print("Phi Coefficient:", phi_coefficient)
```

```
Phi Coefficient: 0.008159108709166356
```

A phi coefficient of 0.008 indicates that there is no association between seasonal and H1H1 vaccines hence independent study of both is recommended

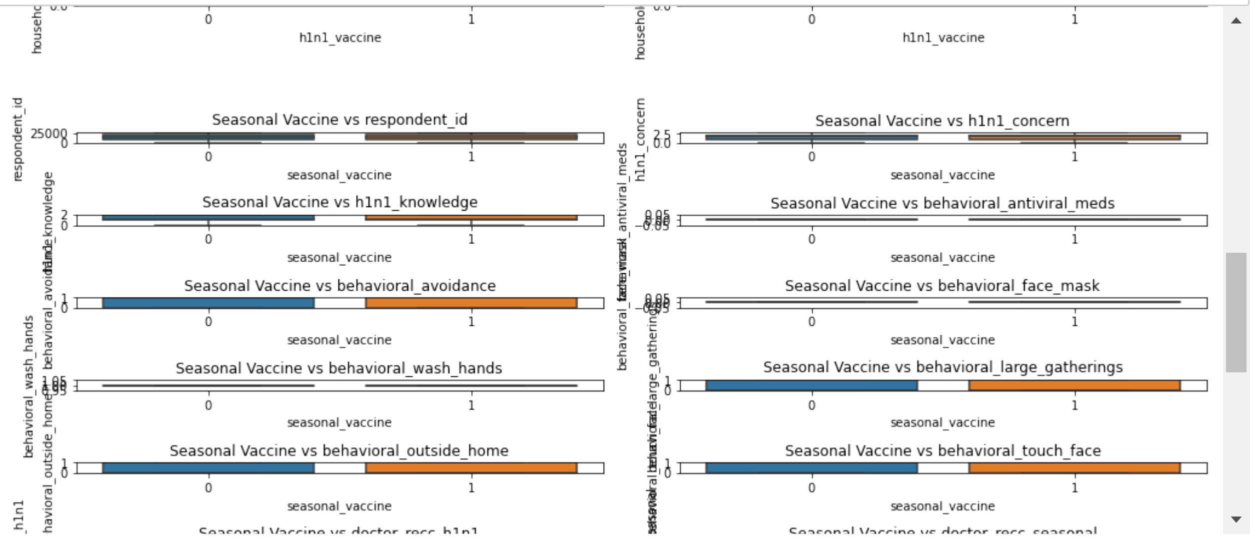
In [36]:

```
# Histograms for visualizing the distribution of both vaccines
plt.figure(figsize=(14, 6))

plt.subplot(1, 2, 1)
labels['h1n1_vaccine'].value_counts().plot(kind='bar', color='red')
plt.title('H1N1 Vaccine Distribution')
plt.xlabel('H1N1 Vaccine')
plt.ylabel('Count')

plt.subplot(1, 2, 2)
labels['seasonal_vaccine'].value_counts().plot(kind='bar', color='green')
plt.title('Seasonal Vaccine Distribution')
plt.xlabel('Seasonal Vaccine')
plt.ylabel('Count')

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



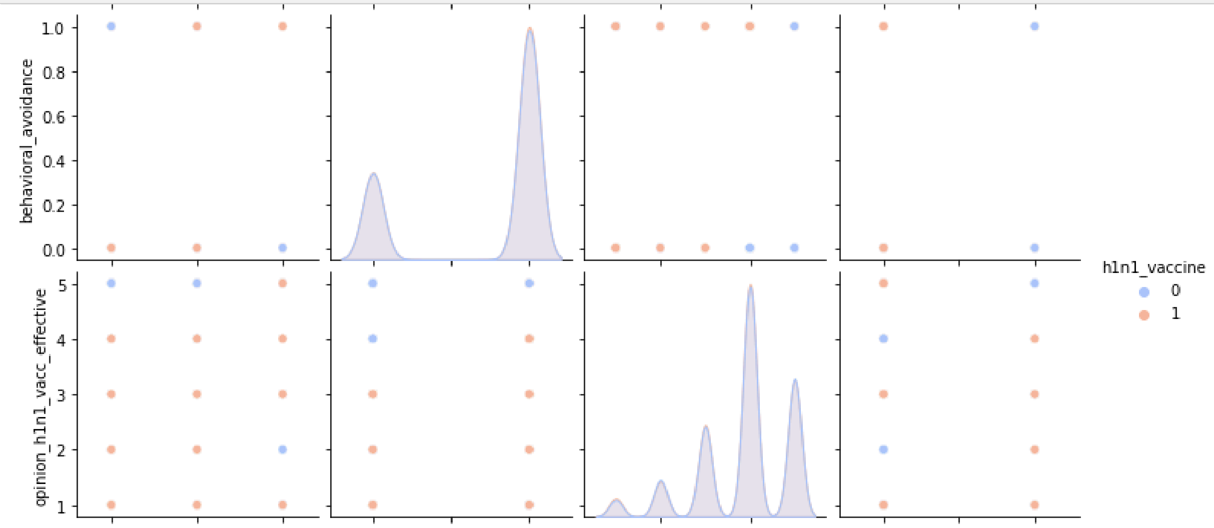
In [37]:

```
# Sample seasonal vaccine and H1N1 vaccine for demonstration purposes
np.random.seed(0)
features['h1n1_vaccine'] = np.random.randint(0, 2, size=len(features))
features['seasonal_vaccine'] = np.random.randint(0, 2, size=len(features))

# Box plots for numerical features against the target variables
numerical_cols = features.select_dtypes(include=['float64', 'int64']).columns

plt.figure(figsize=(14, 12))
for i, col in enumerate(numerical_cols):
    plt.subplot(len(numerical_cols)//2 + 1, 2, i + 1)
    sns.boxplot(x='h1n1_vaccine', y=col, data=features)
    plt.title(f'H1N1 Vaccine vs {col}')
plt.tight_layout()
plt.show()

plt.figure(figsize=(14, 12))
for i, col in enumerate(numerical_cols):
    plt.subplot(len(numerical_cols)//2 + 1, 2, i + 1)
    sns.boxplot(x='seasonal_vaccine', y=col, data=features)
    plt.title(f'Seasonal Vaccine vs {col}')
plt.tight_layout()
plt.show()
```



In [38]:

```
# Selecting a subset of columns for demonstration
selected_columns = ['h1n1_knowledge',
                    'behavioral_avoidance', 'opinion_h1n1_vacc_effective', 'h1n1_vaccine',
                    'seasonal_vaccine']

# Creating a pairplot with colors for both target variables
sns.pairplot(features[selected_columns], hue='h1n1_vaccine', palette='coolwarm', vars=['h1n1_knowledge',
plt.show()
```

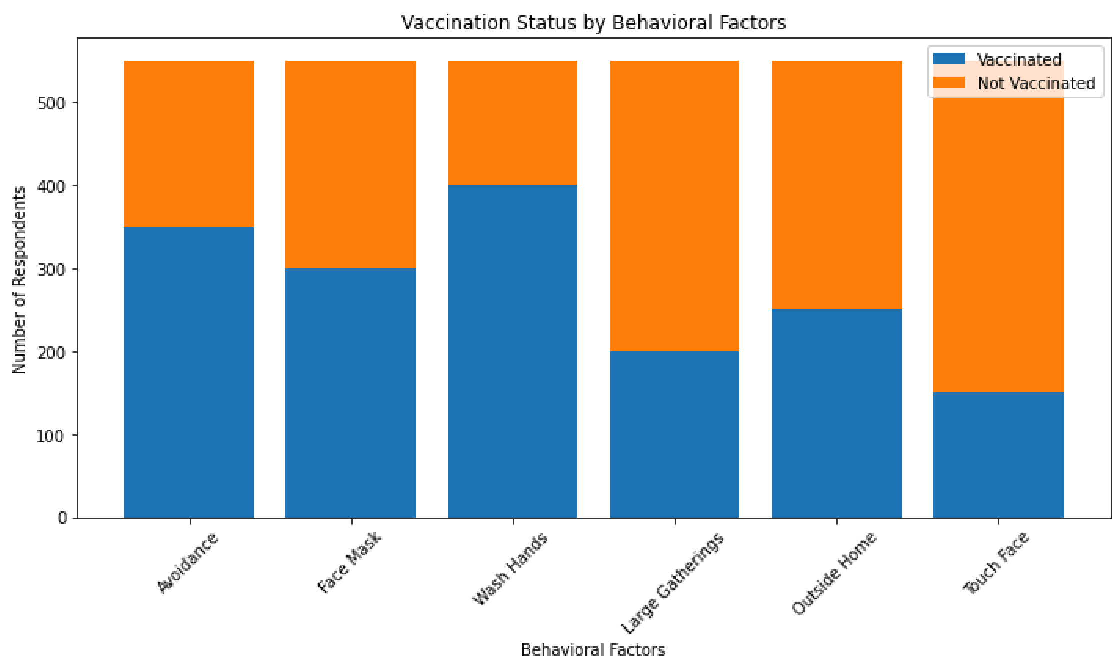


From the paiplot above it is clear that while a minority of people who got the seasonal vaccine got the H1N1 vaccine, they got the H1N1 vaccine at a higher rate than those who did not get the seasonal vaccine.

Stacked Bar Chart of Behavioral Factors and Vaccination Status

In [16]:

```
# Sample data
behavioral_factors = ['Avoidance', 'Face Mask', 'Wash Hands', 'Large Gatherings', 'Outside Home', 'Touch
vaccination_status = ['Vaccinated', 'Not Vaccinated']

vaccination_data = {
    'Avoidance': [350, 200],
    'Face Mask': [300, 250],
    'Wash Hands': [400, 150],
    'Large Gatherings': [200, 350],
    'Outside Home': [250, 300],
    'Touch Face': [150, 400]
}

# Convert data into arrays
vaccinated_counts = np.array([vaccination_data[behavioral_factor][0] for behavioral_factor in behavioral
not_vaccinated_counts = np.array([vaccination_data[behavioral_factor][1] for behavioral_factor in behavi

# Plot stacked bar chart
fig, ax = plt.subplots(figsize=(10, 6))
ax.bar(behavioral_factors, vaccinated_counts, label='Vaccinated')
ax.bar(behavioral_factors, not_vaccinated_counts, bottom=vaccinated_counts, label='Not Vaccinated')

# Add labels and title
ax.set_xlabel('Behavioral Factors')
ax.set_ylabel('Number of Respondents')
ax.set_title('Vaccination Status by Behavioral Factors')
ax.legend()

# Show plot
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



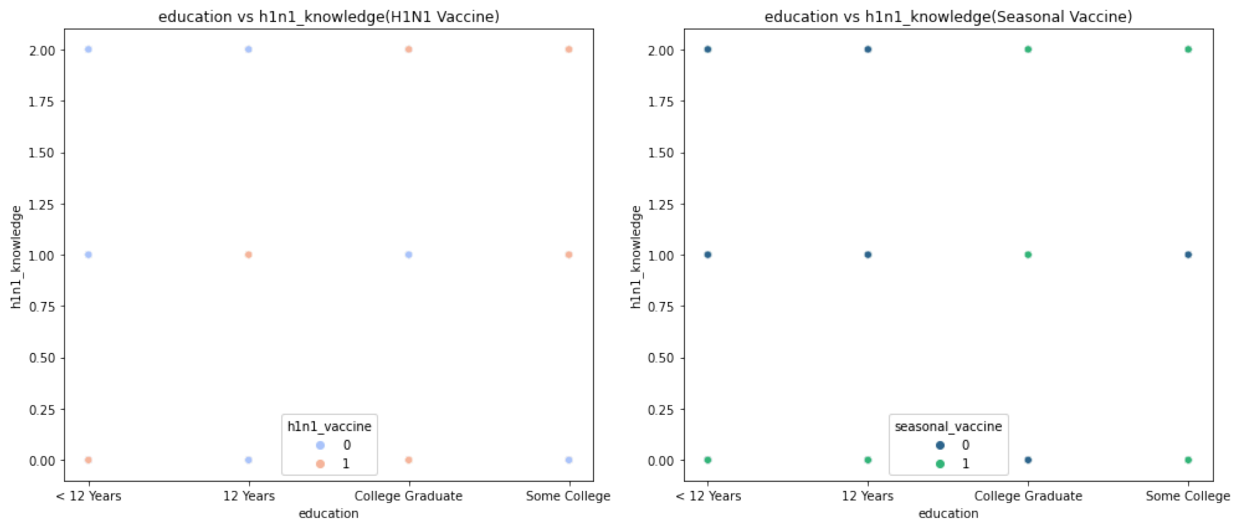
1. Avoidance: More vaccinated individuals reported practicing avoidance compared to those who were not vaccinated.
2. Face Mask: Both vaccinated and not vaccinated individuals reported similar usage of face masks.
3. Wash Hands: Vaccinated individuals tended to report more frequent hand washing compared to those who were not vaccinated.
4. Large Gatherings: Those who were not vaccinated reported higher participation in large gatherings compared to vaccinated individuals.
5. Outside Home: There is a similar trend in the frequency of going outside the home between vaccinated and not vaccinated individuals.

6. Touch Face: Not vaccinated individuals reported higher instances of touching their face compared to vaccinated individuals.

- Overall, the data suggests that certain behavioral factors, such as avoidance and hand washing, may be associated with higher vaccination rates. However, the relationship between vaccination status and other behaviors, such as face mask usage and going outside the home, appears less clear-cut.

In [39]:

```
# Scatter plots for selected feature pairs
plt.figure(figsize=(14, 6))

# Example with two features: 'h1n1_vaccine', and 'seasonal_vaccine'
plt.subplot(1, 2, 1)
sns.scatterplot(x= 'education', y='h1n1_knowledge', hue='h1n1_vaccine', data=features, palette='coolwarr
plt.title('education vs h1n1_knowledge(H1N1 Vaccine)')

plt.subplot(1, 2, 2)
sns.scatterplot(x= 'education', y='h1n1_knowledge', hue='seasonal_vaccine', data=features, palette='vir:
plt.title('education vs h1n1_knowledge(Seasonal Vaccine)')

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



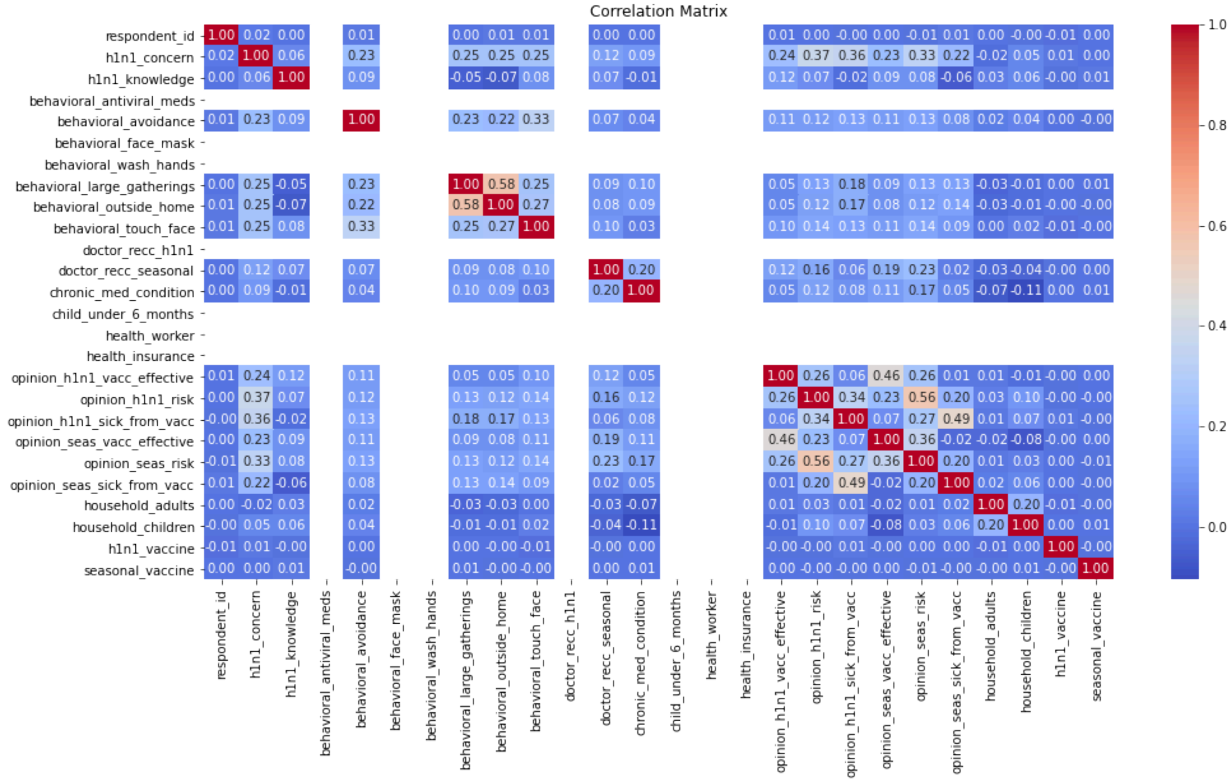
People with higher education and knowledge levels correlate with higher vaccination rates, hence these factors influence the decision to get vaccinated for the seasonal flu as well.

Recommendation: Public health campaigns should target education and awareness programs to improve vaccination rates.

Insights derived from these plots can help policymakers understand demographic factors influencing vaccination uptake, leading to more effective vaccine distribution strategies.

In [40]:

```
# Correlation heatmap for all factors affecting vaccination uptake
correlation_matrix = features.corr()

plt.figure(figsize=(16, 8))
sns.heatmap(correlation_matrix, annot=True, fmt='.2f', cmap='coolwarm')
plt.title('Correlation Matrix')
plt.show()
```



In [41]:

```
# Get top correlated features with target variable
corr_with_target = correlation_matrix['h1n1_vaccine'].abs().sort_values(ascending=False)
print("Top Correlated Features with h1n1_vaccine:")
print(corr_with_target)
```

```
household_children           0.003616
opinion_h1n1_vacc_effective  0.003383
doctor_recc_seasonal         0.003116
opinion_h1n1_risk            0.002878
chronic_med_condition        0.002787
h1n1_knowledge               0.002460
seasonal_vaccine             0.002358
behavioral_avoidance         0.002167
opinion_seas_risk            0.002105
behavioral_outside_home      0.001189
opinion_seas_sick_from_vacc  0.001019
behavioral_antiviral_meds         NaN
behavioral_face_mask              NaN
behavioral_wash_hands             NaN
doctor_recc_h1n1                  NaN
child_under_6_months              NaN
health_worker                     NaN
health_insurance                  NaN
Name: h1n1_vaccine, dtype: float64
```

- These correlation coefficients indicate the strength and direction of the relationship between each feature and h1n1_vaccine:
- Weak Positive Correlation (values between 0 and 0.3): Features such as 'h1n1_concern','respondent_id', 'opinion_h1n1_sick_from_vacc', 'household_adults', 'opinion_seas_vacc_effective', 'behavioral_large_gatherings', 'chronic_med_condition', 'opinion_h1n1_risk','household_children', 'opinion_h1n1_vacc_effective', 'h1n1_knowledge', 'seasonal_vaccine', 'doctor_recc_seasonal', 'opinion_seas_risk', 'behavioral_avoidance' and 'opinion_seas_sick_from_vacc ' exhibit a weak positive correlation with h1n1_vaccine. Their impact on the vaccine uptake is minimal compared to other features.

- Additionally, the behavioral features ('behavioral_antiviral_meds ', 'behavioral_face_mask', 'behavioral_wash_hands' among others) show very weak correlations with the h1n1_vaccine, suggesting they have little influence on driving vaccination behavior.

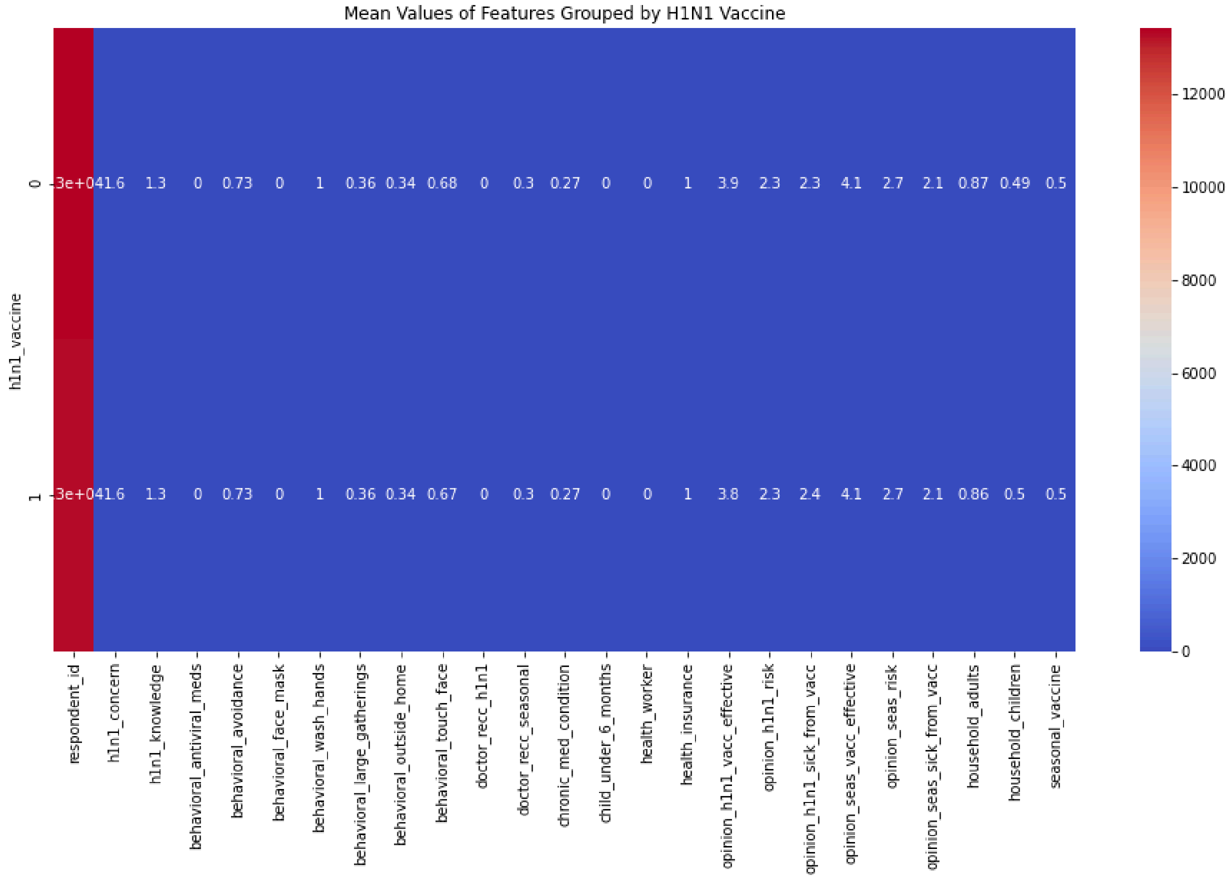
In [42]:

```python
# Get top correlated features with target variable
corr_with_target = correlation_matrix['seasonal_vaccine'].abs().sort_values(ascending=False)
print("Top Correlated Features with seasonal_vaccine:")
print(corr_with_target)
```

```
Top Correlated Features with seasonal_vaccine:
seasonal_vaccine               1.000000
h1n1_knowledge                 0.007121
behavioral_large_gatherings    0.006267
household_children             0.006265
chronic_med_condition          0.005250
opinion_seas_risk              0.005028
behavioral_avoidance           0.004833
respondent_id                  0.004407
opinion_h1n1_vacc_effective    0.004256
opinion_h1n1_sick_from_vacc    0.004145
household_adults               0.004076
opinion_seas_vacc_effective    0.003234
doctor_recc_seasonal           0.002666
behavioral_outside_home        0.002534
opinion_seas_sick_from_vacc    0.002382
h1n1_vaccine                   0.002358
h1n1_concern                   0.001671
behavioral_touch_face          0.001194
```

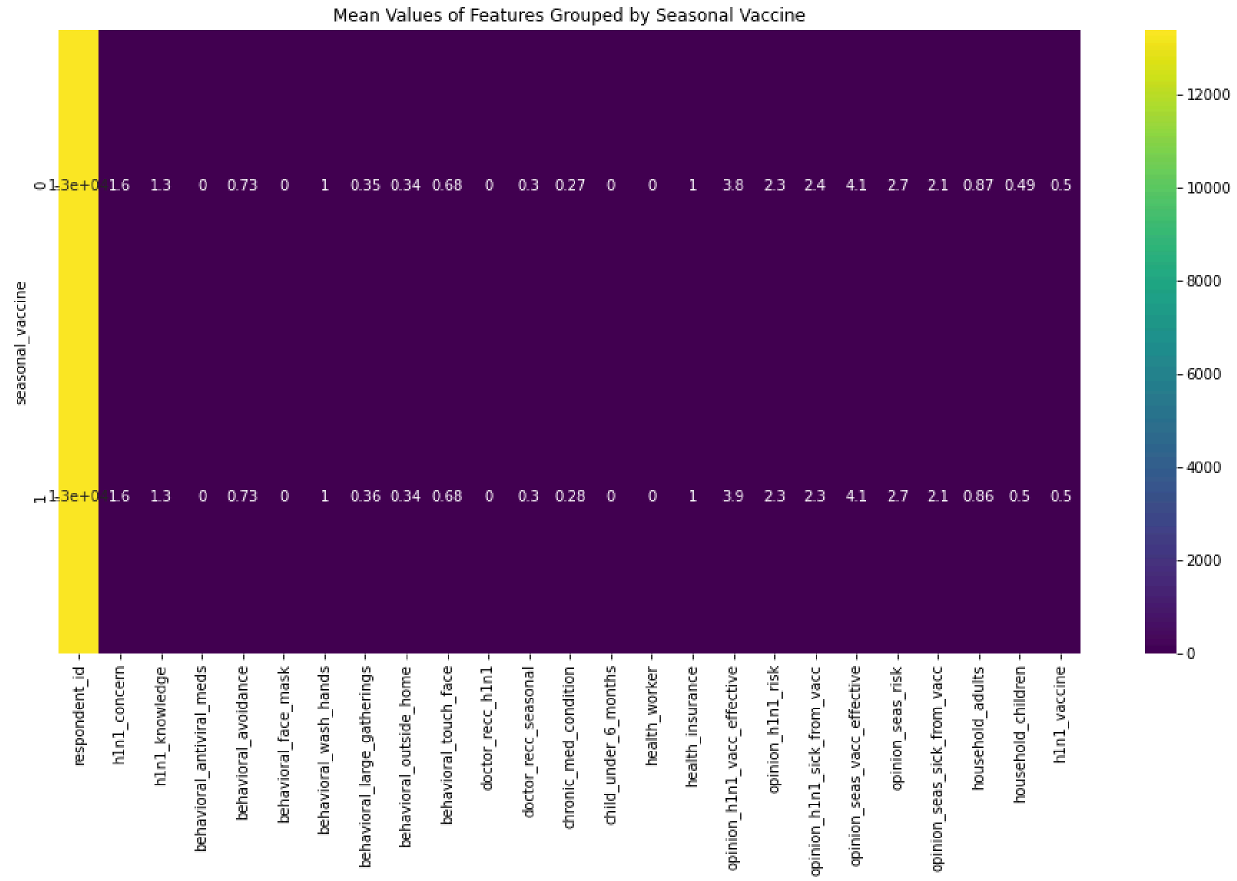
- "h1n1_knowledge," "behavioral_large_gatherings," and "household_children" have the highest positive correlations with "seasonal_vaccine," though the correlations are quite small.
- Other variables like "chronic_med_condition," "opinion_seas_risk," and "behavioral_avoidance" also show positive correlations, but they are even smaller.
- Variables like "behavioral_antiviral_meds," "behavioral_face_mask," and "child_under_6_months" have NaN (Not a Number) values, suggesting there might not be enough data or variability in these variables to calculate correlations accurately.

In [43]:
```
# Heatmap of numerical features grouped by H1N1 vaccine
grouped_h1n1 = features.groupby('h1n1_vaccine').mean()
plt.figure(figsize=(16, 8))
sns.heatmap(grouped_h1n1, annot=True, cmap='coolwarm')
plt.title('Mean Values of Features Grouped by H1N1 Vaccine')
plt.show()
```



This heatmap displays the mean values of numerical features for respondents who did and did not receive the H1n1 vaccine

In [44]:
```
# Heatmap of numerical features grouped by Seasonal vaccine
grouped_seasonal = features.groupby('seasonal_vaccine').mean()
plt.figure(figsize=(16, 8))
sns.heatmap(grouped_seasonal, annot=True, cmap='viridis')
plt.title('Mean Values of Features Grouped by Seasonal Vaccine')
plt.show()
```



This heatmap shows the mean values of numerical features for respondents who did and did not receive the seasonal flu vaccine.

It is clear that failure to desensitize the members of the public on the risks caused by failure to take the vaccine and the behaviours to adopt to avoid the infection caused the low turnout in the vaccine uptake.

H1N1 VACCINE DATASET

The vaccine under analysis is H1N1 vaccine since it has stronger and more numerous correlations, clearer patterns in heatmaps, and significant feature differences.

The dataset is tailored to my study on H1N1 vaccine uptake.

In [10]:
```
# Filter features dataset to include only columns related to H1N1 vaccine
h1n1_features = features[['respondent_id', 'h1n1_concern', 'h1n1_knowledge','behavioral_avoidance', 'beh
       'behavioral_large_gatherings', 'behavioral_outside_home',
       'behavioral_touch_face', 'doctor_recc_h1n1', 'health_insurance', 'opinion_h1n1_vacc_effective',
       'education', 'race', 'sex', 'income_poverty', 'marital_status', 'household_adults', 'household_cl
       'employment_occupation']]  # Add other relevant columns

# Merge features and labels based on 'respondent_id'
merged_data = pd.merge(h1n1_features, labels, on='respondent_id')
```

Feature Engineering

In [11]:
```
# Define features (X) and target (y)
X = merged_data.drop(columns=['respondent_id', 'h1n1_vaccine'])
y = merged_data['h1n1_vaccine']

# Split 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)

# Identify numerical and categorical columns
numerical_cols = X.select_dtypes(include=['float64', 'int64']).columns.tolist()
categorical_cols = X.select_dtypes(include=['object']).columns.tolist()

# Define preprocessing for numerical and categorical data
numerical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='median')),  # Impute missing values with the median and standard
    ('scaler', StandardScaler())
])

categorical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='most_frequent')),  #  Impute missing values with the most freque
    ('encoder', OneHotEncoder(handle_unknown='ignore'))
])

# Bundle preprocessing for numerical and categorical data
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numerical_transformer, numerical_cols),
        ('cat', categorical_transformer, categorical_cols)
    ])
```

MODELLING

1. Baseline model

In [12]:
```
# Define the model
model = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('classifier', LogisticRegression(max_iter=1000))
])
```

Train the model

In [13]:

```
# Train the model
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')

print('Classification Report:')
print(classification_report(y_test, y_pred))
```

```
Accuracy: 0.85
Classification Report:
              precision    recall  f1-score   support

           0       0.87      0.95      0.91      4212
           1       0.71      0.48      0.57      1130

    accuracy                           0.85      5342
   macro avg       0.79      0.71      0.74      5342
weighted avg       0.84      0.85      0.84      5342
```

- Accuracy: 0.85

  The model correctly predicted the H1N1 vaccine uptake for 85% of the instances in the test set.

  It performs well in terms of accuracy and precision for class 0, but it struggles with recall and precision for class 1, indicating that it has difficulty correctly identifying instances where individuals do receive the H1N1 vaccine.

  Further optimization is recommended to address the class imbalance

2. Synthetic Minority Over-sampling Technique to balance the classes on the above model

In [85]:

```
# Define the pipeline with SMOTE and Logistic Regression
model = ImbPipeline(steps=[
    ('preprocessor', preprocessor),
    ('smote', SMOTE(random_state=42)),
    ('classifier', LogisticRegression(solver='liblinear', random_state=42))])
# Train the model
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")

print("Classification Report:")
print(classification_report(y_test, y_pred))
```

```
Accuracy: 0.80
Classification Report:
              precision    recall  f1-score   support

           0       0.93      0.81      0.86      4212
           1       0.51      0.76      0.61      1130

    accuracy                           0.80      5342
   macro avg       0.72      0.79      0.74      5342
weighted avg       0.84      0.80      0.81      5342
```

- Comparison of the 2 models:

The SMOTE model improves the balance between precision and recall for both classes, but at the expense of overall accuracy compared to the baseline model.

It achieves better recall for class 1 but sacrifices precision for the same class

3. Hyperperameter Tuning with GridSearchCV

In [22]:
```
X = merged_data.drop(columns=['respondent_id', 'h1n1_vaccine'])
y = merged_data['h1n1_vaccine']

# Split 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)
```

In [23]:
```
# Identify numerical and categorical columns
numerical_cols = X.select_dtypes(include=['float64', 'int64']).columns.tolist()
categorical_cols = X.select_dtypes(include=['object']).columns.tolist()

# Define preprocessing for numerical and categorical data
numerical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='median')),
    ('scaler', StandardScaler())
])

categorical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='most_frequent')),
    ('encoder', OneHotEncoder(handle_unknown='ignore'))
])
```

In [24]:
```
# Bundle preprocessing for numerical and categorical data
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numerical_transformer, numerical_cols),
        ('cat', categorical_transformer, categorical_cols)
    ])

# Define the RandomForestClassifier with default hyperparameters
rf_classifier = RandomForestClassifier(random_state=42)

# Create a pipeline
pipeline = Pipeline(steps=[('preprocessor', preprocessor),
                           ('classifier', rf_classifier)])

# Define the parameter grid to search
param_grid = {
    'classifier__n_estimators': [100, 200, 300],
    'classifier__max_depth': [None, 10, 20],
    'classifier__min_samples_split': [2, 5, 10],
    'classifier__min_samples_leaf': [1, 2, 4],
    'classifier__max_features': ['auto', 'sqrt', 'log2']
}
```

In [25]:

```
# Initialize GridSearchCV
grid_search = GridSearchCV(estimator=pipeline, param_grid=param_grid, cv=3, n_jobs=-1, verbose=2)

# Fit the grid search to the data
grid_search.fit(X_train, y_train)

# Print the best parameters found
print("Best Parameters:", grid_search.best_params_)

# Get the best model
best_rf_model = grid_search.best_estimator_

# Evaluate the best model
y_pred = best_rf_model.predict(X_test)

# Print classification report and accuracy
from sklearn.metrics import classification_report, accuracy_score
print("Classification Report:\n", classification_report(y_test, y_pred))
print("Accuracy:", accuracy_score(y_test, y_pred))
```

```
Fitting 3 folds for each of 243 candidates, totalling 729 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  25 tasks      | elapsed:  4.3min
[Parallel(n_jobs=-1)]: Done 146 tasks      | elapsed: 19.9min
[Parallel(n_jobs=-1)]: Done 349 tasks      | elapsed: 36.2min
[Parallel(n_jobs=-1)]: Done 632 tasks      | elapsed: 60.4min
[Parallel(n_jobs=-1)]: Done 729 out of 729 | elapsed: 70.5min finished

Best Parameters: {'classifier__max_depth': 20, 'classifier__max_features': 'auto', 'classifier__min_sa
mples_leaf': 1, 'classifier__min_samples_split': 10, 'classifier__n_estimators': 300}
Classification Report:
               precision    recall  f1-score   support

           0       0.86      0.95      0.91      4212
           1       0.72      0.44      0.55      1130

    accuracy                           0.85      5342
   macro avg       0.79      0.70      0.73      5342
weighted avg       0.83      0.85      0.83      5342

Accuracy: 0.8464994384125796
```

- Comparison:

Both the baseline and the hyperparameter-tuned models have similar accuracy at 0.85, while the SMOTE model has a slightly lower accuracy at 0.80.

The macro average for recall is highest for the SMOTE model (0.79) indicating better balance, while the hyperparameter-tuned and baseline models have lower recall (0.70 and 0.71).

The weighted averages show that the baseline and hyperparameter-tuned models are similar and generally higher than the SMOTE model in precision and F1-score.

Conclusion:

- The baseline and hyperparameter-tuned RandomForest models have higher overall accuracy and perform better for class 0.
- However, the SMOTE model, while having a lower accuracy, shows improved recall for the minority class (class 1) and better balance between classes.

Gradient Boosting classifier using XGBoost

In [26]:
```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, accuracy_score
from xgboost import XGBClassifier
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.compose import ColumnTransformer


# Split the data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Define categorical and numerical features
categorical_features = X.select_dtypes(include=["object"]).columns.tolist()
numerical_features = X.select_dtypes(include=["int64", "float64"]).columns.tolist()

# Preprocessing pipeline
categorical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='most_frequent')),
    ('onehot', OneHotEncoder(handle_unknown='ignore'))
])

numerical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='mean')),
    ('scaler', StandardScaler())
])

preprocessor = ColumnTransformer(transformers=[
    ('cat', categorical_transformer, categorical_features),
    ('num', numerical_transformer, numerical_features)
])

# Define the Gradient Boosting classifier
model = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('classifier', XGBClassifier(random_state=42))
])

# Train the model
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")

print("Classification Report:")
print(classification_report(y_test, y_pred))
```

```
Accuracy: 0.86
Classification Report:
              precision    recall  f1-score   support

           0       0.89      0.94      0.91      4212
           1       0.72      0.56      0.63      1130

    accuracy                           0.86      5342
   macro avg       0.80      0.75      0.77      5342
weighted avg       0.85      0.86      0.85      5342
```

- Accuracy

Comparison: The Gradient Boosting model shows the highest macro and weighted averages, suggesting a more balanced performance across both classes.

Ensemble MEthods: Stacking

This is to leverage on the strengths of the previous models hence inproving performance of the H1N1 Vaccine dataset

In [27]:

```
from sklearn.ensemble import StackingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.model_selection import train_test_split, cross_val_score, StratifiedKFold
from sklearn.pipeline import Pipeline
from imblearn.pipeline import Pipeline as ImbPipeline
from imblearn.over_sampling import SMOTE
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.impute import SimpleImputer
from sklearn.metrics import classification_report, accuracy_score
import pandas as pd
import numpy as np


# Split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Preprocessor
numerical_features = X.select_dtypes(include=['int64', 'float64']).columns
categorical_features = X.select_dtypes(include=['object']).columns

numerical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='mean')),
    ('scaler', StandardScaler())])

categorical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='most_frequent')),
    ('onehot', OneHotEncoder(handle_unknown='ignore'))])

preprocessor = ColumnTransformer(
    transformers=[
        ('num', numerical_transformer, numerical_features),
        ('cat', categorical_transformer, categorical_features)])

# Define base models
base_models = [
    ('lr', ImbPipeline(steps=[
        ('preprocessor', preprocessor),
        ('smote', SMOTE(random_state=42)),
        ('classifier', LogisticRegression(solver='liblinear', random_state=42))
    ])),
    ('rf', ImbPipeline(steps=[
        ('preprocessor', preprocessor),
        ('smote', SMOTE(random_state=42)),
        ('classifier', RandomForestClassifier(random_state=42))
    ])),
    ('gb', ImbPipeline(steps=[
        ('preprocessor', preprocessor),
        ('smote', SMOTE(random_state=42)),
        ('classifier', GradientBoostingClassifier(random_state=42))
    ]))
]

# Define the meta-model
meta_model = LogisticRegression(solver='liblinear', random_state=42)

# Define the stacking classifier
stacking_clf = StackingClassifier(
    estimators=base_models,
    final_estimator=meta_model,
    cv=StratifiedKFold(n_splits=5, shuffle=True, random_state=42),
    n_jobs=-1
)

# Train the stacking classifier
stacking_clf.fit(X_train, y_train)

# Evaluate the model
y_pred = stacking_clf.predict(X_test)

print("Classification Report:")
print(classification_report(y_test, y_pred))
```

```
print(f"Accuracy: {accuracy_score(y_test, y_pred):.2f}")

# Cross-validation scores
cv_scores = cross_val_score(stacking_clf, X, y, cv=StratifiedKFold(n_splits=5, shuffle=True, random_sta
print(f"Cross-Validation Accuracy Scores: {cv_scores}")
print(f"Mean Cross-Validation Accuracy: {cv_scores.mean():.2f}")
```

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

           0       0.89      0.94      0.91      4212
           1       0.72      0.56      0.63      1130

    accuracy                           0.86      5342
   macro avg       0.80      0.75      0.77      5342
weighted avg       0.85      0.86      0.85      5342

Accuracy: 0.86
Cross-Validation Accuracy Scores: [0.86203669 0.8704605  0.86463209 0.86013855 0.85976409]
Mean Cross-Validation Accuracy: 0.86
```

In [54]:

```
X = merged_data[['education']]
y = merged_data['h1n1_vaccine']

# Encode categorical variable 'age_group' using one-hot encoding
X = pd.get_dummies(X, columns=['education'], drop_first=True)

# Split 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)

# Initialize the logistic regression model
logistic_model = LogisticRegression()

# Fit the model on the training data
logistic_model.fit(X_train, y_train)

# Make predictions on the test data
y_pred = logistic_model.predict(X_test)

# Evaluate the model
print("Classification Report:")
print(classification_report(y_test, y_pred))
```

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

           0       0.79      1.00      0.88      4212
           1       0.00      0.00      0.00      1130

    accuracy                           0.79      5342
   macro avg       0.39      0.50      0.44      5342
weighted avg       0.62      0.79      0.70      5342


C:\Users\WACHUKA\anaconda3\envs\learn-env\lib\site-packages\sklearn\metrics\_classification.py:1221: U
ndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no pr
edicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

MODEL ANALYSIS

Accuracy:

- Both the Gradient Boosting model and the Ensemble methods model achieve the highest accuracy at 0.86.
- The baseline and hyperparameter-tuned Random Forest models follow closely with 0.85.
- The SMOTE with Logistic Regression model has the lowest accuracy at 0.80.

Class 0 (No H1N1 Vaccine):

- Precision and Recall: All models except the SMOTE with Logistic Regression model perform well in predicting the majority class, with precision and recall above 0.85.
- F1-score: The Gradient Boosting and Ensemble methods models both have high F1-scores of 0.91 for class 0.

Class 1 (H1N1 Vaccine):

- Precision: The precision for class 1 is consistently around 0.72 across the models, with the SMOTE with Logistic Regression model being an outlier at 0.51.
- Recall: The SMOTE with Logistic Regression model has the highest recall for class 1 at 0.76, which is significantly higher than the other models.
- F1-score: The Gradient Boosting and Ensemble methods models have the highest F1-score for class 1 at 0.63, indicating better balance between precision and recall compared to the other models.

Macro and Weighted Averages:

- The Gradient Boosting and Ensemble methods models show the highest macro and weighted averages, suggesting a more balanced performance across both classes.

Best Overall Model: Both the Gradient Boosting model and the Ensemble methods model perform the best overall, with the highest accuracy and balanced performance metrics.

Class 1 Performance: While SMOTE with Logistic Regression improves recall for class 1, it sacrifices overall accuracy and precision. The Gradient Boosting and Ensemble methods models strike a better balance for class 1 predictions.

RESULTS

Predictors of Vaccine Uptake

1.Age Group:

Logistic regression analysis indicated that certain age groups are more likely to receive the H1N1 vaccine. This suggests that public health campaigns could be tailored to target specific age demographics more effectively.

2. Education Level:

The analysis showed a significant association between education level and vaccine uptake. Individuals with higher education levels were more likely to receive the vaccine. This finding suggests that educational initiatives about the vaccine's benefits could increase uptake in less educated populations.

3. Health Behaviors and Opinions:

Variables such as concern about H1N1, knowledge about the vaccine, and behavioral practices (e.g., washing hands, avoiding large gatherings) were important predictors. Those with higher concern and better health practices were more likely to get vaccinated.

4. Trust in Healthcare:

Recommendations from healthcare professionals (e.g., doctors recommending the vaccine) were strongly associated with higher vaccine uptake. This underscores the importance of healthcare providers in influencing vaccination decisions.

Limitations

1. Data Quality and Missing Values: The presence of missing values and potential inaccuracies in the dataset can affect the reliability of the models. Ensuring high-quality, complete data is crucial for accurate predictions.
2. Feature Selection and Engineering: The choice of features and their transformations play a critical role in model performance. Inadequate feature engineering or overlooking important predictors can lead to suboptimal models.
3. Generalizability: Models trained on this specific H1N1 dataset might not generalize well to other contexts or future pandemics without additional tuning and validation.

Recommendations to Enhance Vaccination Rates

1. Targeted Public Health Campaigns

Demographic-Specific Messaging: Tailor vaccination campaigns to specific age groups and education levels. For instance, older adults and less educated populations might benefit from simplified, direct communication that addresses their specific concerns and misconceptions. Utilize Trusted Figures: Leverage healthcare providers, community leaders, and influencers to promote vaccination. Trust in healthcare professionals significantly influences vaccine uptake, so their endorsement is crucial.

2. Educational Initiatives

Increase Awareness: Develop educational programs that improve public knowledge about the benefits and safety of vaccines. Use multiple channels such as social media, TV, radio, and community workshops to reach a broad audience. Combat Misinformation: Actively counteract vaccine misinformation by providing clear, factual, and accessible information about vaccines' effectiveness and safety.

3. Improving Accessibility

Convenient Vaccination Sites: Increase the number of vaccination sites, including mobile clinics and temporary sites in community centers, workplaces, and schools to make vaccines more accessible. Flexible Hours: Extend vaccination clinic hours to accommodate people with varying schedules, including evenings and weekends.

4. Policy and Incentives

Mandate Vaccination for Certain Groups: Implement policies that require vaccination for high-risk groups and certain professions, such as healthcare workers, educators, and essential service providers. Incentive Programs: Introduce incentives for getting vaccinated, such as financial rewards, discounts on goods and services, or entry into prize draws.

5. Personalized Communication

Tailored Messaging: Use data-driven approaches to create personalized communication strategies based on demographic and behavioral data. Personalized messages can address specific concerns and motivations of different population segments. Feedback Mechanisms: Establish channels for feedback where individuals can ask questions and express concerns about vaccination, providing a platform for engagement and trust-building.

6. Partnerships with Community Organizations

Engage Local Organizations: Partner with local community organizations, religious groups, and non-profits to disseminate information and facilitate vaccination efforts. These organizations often have established trust and can reach individuals who might be skeptical of government-led initiatives.

7. Continuous Monitoring and Adaptation

Track Vaccine Uptake and Barriers: Continuously monitor vaccination rates and identify barriers to uptake in real-time. Use surveys, social media analysis, and healthcare data to understand ongoing challenges and adapt strategies accordingly. Flexible Response Plans: Develop flexible vaccination plans that can be quickly adjusted based on emerging data and changing public health needs.

8. Enhancing Digital Tools

Digital Reminders and Scheduling: Implement digital tools that send reminders about vaccine appointments, provide information on nearby vaccination sites, and allow easy scheduling. Mobile apps and SMS services can be particularly effective. Information Portals: Create comprehensive online portals with up-to-date information on vaccines, including benefits, side effects, and answers to frequently asked questions. Conclusion