- Group: **14**
- Student pace: **Part-time**
- Scheduled project review date/time: **13/02/2024**
- Instructor name:
- Blog post URL: N/A

Predicting H1N1 Flu Vaccination Status ¶

Overview

As the world struggles to vaccinate the global population against COVID-19, an understanding of how people's backgrounds, opinions, and health behaviors are related to their personal vaccination patterns can provide guidance for future public health efforts. Such findings can guide policymakers and public health professionals develop public health efforts to improve vaccine uptake to mitigate spread of preventable communicable dieases.

This project utilises data from a United States' conducted National 2009 H1N1 Flu Survey to predict whether someone revieved H1N1 flu vaccines. Gaining deeper insights into how these attributes correlate with individual vaccination behaviors can offer valuable direction for upcoming public health initiatives.

Business Understanding

The National 2009 H1N1 Flu Survey data was downloaded from DrivenData and the purpose of this project is to use data to forecast whether or not a person received the H1N1 flu vaccination, using features such as social, economic, and demographic background, opinions on risks of illness and vaccine effectiveness, and behaviors towards mitigating transmission, etcetera. The findings would also be applicable for use by the Kenya's Ministry of Health to more effectively target public health initiatives that boost vaccination rates and localise for other communicable dieases like influenza.

Research Questions

1. How may false information regarding vaccines be effectively refuted to encourage factual knowledge and vaccination acceptance of the flu?

2. Which major groups are most likely to be deceived about the flu shot, especially those with lower levels of education? How might focused messaging be developed to allay their fears?
3. How can resource allocation for flu vaccination programs be optimized by the use of predictive modeling, guaranteeing effective distribution to high-risk populations and regions?
4. What programs may be put in place to promote the general adoption of preventative measures like donning face masks, avoiding crowds, and washing your hands properly?
5. How can cooperation with medical professionals be improved to encourage routine physical examinations and support physicians' recommendations for flu shots among high-risk individuals?

Objectives

1. Analyze how well various communication tactics work to debunk vaccine misconceptions and spread factual information about getting the flu shot.
2. Determine which high-risk groups to target with messaging and interventions, especially those who are less educated and show little concern for the H1N1.
3. Create a predictive model that makes the best use of behavioral, epidemiological, and demographic data to allocate resources for flu vaccine campaigns.
4. Create and put into action policies and initiatives that will increase people's adherence to preventive measures like hand hygiene, avoiding crowds, and wearing face masks.
5. Create efficient channels of communication with medical professionals to enable routine physical examinations and improve physician recommendations for influenza shots in high-risk patients. nts.

Data Description

- 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.resented as short

Import Libraries

In [1]:
```
import pandas as pd
import numpy as np
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer
from sklearn.preprocessing import OneHotEncoder, StandardScaler, MinMaxScaler
from sklearn.model_selection import train_test_split, cross_validate
from imblearn.over_sampling import SMOTE
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_curve, roc_auc_score
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn.naive_bayes import GaussianNB
from sklearn.feature_selection import RFE, mutual_info_classif
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV
from xgboost import XGBClassifier
from sklearn.ensemble import BaggingClassifier, RandomForestClassifier
from sklearn.pipeline import Pipeline
import matplotlib.pyplot as plt
import seaborn as sns
plt.style.use('seaborn-v0_8-darkgrid')
```

Load Data

In [2]:
```
# Loading data [original data from 'DrivenData' was seperated into training and testing sets]
train_features_df = pd.read_csv("Data/training_set_features.csv", index_col="respondent_id")

train_labels_df = pd.read_csv("Data/training_set_labels.csv",index_col="respondent_id")

test_validation_df = pd.read_csv("Data/test_set_features.csv",index_col="respondent_id")
```

In [3]:
```
# check first 5 rows of features_df
train_features_df.head()
```

Out[3]:

	h1n1_concern	h1n1_knowledge	behavioral_antiviral_meds	behavioral_avoidance	behavioral_face_mask	behavioral_wash_hands	bel
respondent_id							
0	1.0	0.0	0.0	0.0	0.0	0.0	
1	3.0	2.0	0.0	1.0	0.0	1.0	
2	1.0	1.0	0.0	1.0	0.0	0.0	
3	1.0	1.0	0.0	1.0	0.0	1.0	
4	2.0	1.0	0.0	1.0	0.0	1.0	

5 rows × 35 columns

In [4]:
```
# check first 5 rows of labels_df
train_labels_df.head()
```

Out[4]:

	h1n1_vaccine	seasonal_vaccine
respondent_id		
0	0	0
1	0	1
2	0	0
3	0	1
4	0	0

In [5]:
```
# check first 5 rows of test_validation_df
test_validation_df.head()
```

Out[5]:

	h1n1_concern	h1n1_knowledge	behavioral_antiviral_meds	behavioral_avoidance	behavioral_face_mask	behavioral_wash_hands	bel
respondent_id							
26707	2.0	2.0	0.0	1.0	0.0	1.0	
26708	1.0	1.0	0.0	0.0	0.0	0.0	
26709	2.0	2.0	0.0	0.0	1.0	1.0	
26710	1.0	1.0	0.0	0.0	0.0	0.0	
26711	3.0	1.0	1.0	1.0	0.0	1.0	

5 rows × 35 columns

In [6]:
```
# Double-check that the rows between the features and the labels match up
np.testing.assert_array_equal(train_features_df.index.values, train_labels_df.index.values)
```

In [7]:
```
## Merge the features and labels data
merged_df = pd.concat([train_features_df, train_labels_df], axis =1)
merged_df
```

Out[7]:

	h1n1_concern	h1n1_knowledge	behavioral_antiviral_meds	behavioral_avoidance	behavioral_face_mask	behavioral_wash_hands	bel
respondent_id							
0	1.0	0.0	0.0	0.0	0.0	0.0	
1	3.0	2.0	0.0	1.0	0.0	1.0	
2	1.0	1.0	0.0	1.0	0.0	0.0	
3	1.0	1.0	0.0	1.0	0.0	1.0	
4	2.0	1.0	0.0	1.0	0.0	1.0	
...	...	...	...	...	...	...	
26702	2.0	0.0	0.0	1.0	0.0	0.0	
26703	1.0	2.0	0.0	1.0	0.0	1.0	
26704	2.0	2.0	0.0	1.0	1.0	1.0	
26705	1.0	1.0	0.0	0.0	0.0	0.0	
26706	0.0	0.0	0.0	1.0	0.0	0.0	

26707 rows × 37 columns

In [8]:
```
merged_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 26707 entries, 0 to 26706
Data columns (total 37 columns):
 #   Column                       Non-Null Count  Dtype
---  ------                       --------------  -----
 0   h1n1_concern                 26615 non-null  float64
 1   h1n1_knowledge               26591 non-null  float64
 2   behavioral_antiviral_meds    26636 non-null  float64
 3   behavioral_avoidance         26499 non-null  float64
 4   behavioral_face_mask         26688 non-null  float64
 5   behavioral_wash_hands        26665 non-null  float64
 6   behavioral_large_gatherings  26620 non-null  float64
 7   behavioral_outside_home      26625 non-null  float64
 8   behavioral_touch_face        26579 non-null  float64
 9   doctor_recc_h1n1             24547 non-null  float64
 10  doctor_recc_seasonal         24547 non-null  float64
 11  chronic_med_condition        25736 non-null  float64
 12  child_under_6_months         25887 non-null  float64
 13  health_worker                25903 non-null  float64
 14  health_insurance             14433 non-null  float64
 15  opinion_h1n1_vacc_effective  26316 non-null  float64
 16  opinion_h1n1_risk            26319 non-null  float64
 17  opinion_h1n1_sick_from_vacc  26312 non-null  float64
 18  opinion_seas_vacc_effective  26245 non-null  float64
 19  opinion_seas_risk            26193 non-null  float64
 20  opinion_seas_sick_from_vacc  26170 non-null  float64
 21  age_group                    26707 non-null  object
 22  education                    25300 non-null  object
 23  race                         26707 non-null  object
 24  sex                          26707 non-null  object
 25  income_poverty               22284 non-null  object
 26  marital_status               25299 non-null  object
 27  rent_or_own                  24665 non-null  object
 28  employment_status            25244 non-null  object
 29  hhs_geo_region               26707 non-null  object
 30  census_msa                   26707 non-null  object
 31  household_adults             26458 non-null  float64
 32  household_children           26458 non-null  float64
 33  employment_industry          13377 non-null  object
 34  employment_occupation        13237 non-null  object
 35  h1n1_vaccine                 26707 non-null  int64
```

```
 36  seasonal_vaccine            26707 non-null  int64
dtypes: float64(23), int64(2), object(12)
memory usage: 7.7+ MB
```

Additionally the datatypes consist of 23 columns of float type, 2 interger(int64) columns and 12 columns of object datatype.

In [9]:
```
merged_df.describe()
```

Out[9]:

	h1n1_concern	h1n1_knowledge	behavioral_antiviral_meds	behavioral_avoidance	behavioral_face_mask	behavioral_wash_hands	behavioral_
count	26615.000000	26591.000000	26636.000000	26499.000000	26688.000000	26665.000000	
mean	1.618486	1.262532	0.048844	0.725612	0.068982	0.825614	
std	0.910311	0.618149	0.215545	0.446214	0.253429	0.379448	
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	1.000000	1.000000	0.000000	0.000000	0.000000	1.000000	
50%	2.000000	1.000000	0.000000	1.000000	0.000000	1.000000	
75%	2.000000	2.000000	0.000000	1.000000	0.000000	1.000000	
max	3.000000	2.000000	1.000000	1.000000	1.000000	1.000000	

8 rows × 25 columns

Inference

Target Variable

- `h1n1_vaccine` : The sample popluation show that 21.24% received the vacccine, therefore 78.76% during the survey year (2009)

Data Cleaning

In [10]:
```
merged_df.isna().sum()
```

Out[10]:
```
h1n1_concern                      92
h1n1_knowledge                   116
behavioral_antiviral_meds         71
behavioral_avoidance             208
behavioral_face_mask              19
behavioral_wash_hands             42
behavioral_large_gatherings       87
behavioral_outside_home           82
behavioral_touch_face            128
doctor_recc_h1n1                2160
doctor_recc_seasonal            2160
chronic_med_condition            971
child_under_6_months             820
health_worker                    804
health_insurance               12274
opinion_h1n1_vacc_effective      391
opinion_h1n1_risk                388
opinion_h1n1_sick_from_vacc      395
opinion_seas_vacc_effective      462
opinion_seas_risk                514
opinion_seas_sick_from_vacc      537
age_group                          0
education                       1407
race                               0
sex                                0
income_poverty                  4423
marital_status                  1408
rent_or_own                     2042
employment_status               1463
hhs_geo_region                     0
census_msa                         0
household_adults                 249
household_children               249
employment_industry            13330
employment_occupation          13470
h1n1_vaccine                       0
seasonal_vaccine                   0
dtype: int64
```

In [11]:
```
missing = [[column, round(sum(merged_df[column].isna())/len(merged_df[column]),2)] for column in merged_df.columns]
pd.DataFrame(missing, columns= ['column name', 'missing proportion']).sort_values(by='missing proportion', ascending=
```

Out[11]:

	column name	missing proportion
34	employment_occupation	0.50
33	employment_industry	0.50
14	health_insurance	0.46
25	income_poverty	0.17
27	rent_or_own	0.08
9	doctor_recc_h1n1	0.08
10	doctor_recc_seasonal	0.08
26	marital_status	0.05
22	education	0.05
28	employment_status	0.05
11	chronic_med_condition	0.04
13	health_worker	0.03
12	child_under_6_months	0.03
19	opinion_seas_risk	0.02
20	opinion_seas_sick_from_vacc	0.02
18	opinion_seas_vacc_effective	0.02
31	household_adults	0.01
15	opinion_h1n1_vacc_effective	0.01
16	opinion_h1n1_risk	0.01
17	opinion_h1n1_sick_from_vacc	0.01
32	household_children	0.01
3	behavioral_avoidance	0.01
30	census_msa	0.00
29	hhs_geo_region	0.00
35	h1n1_vaccine	0.00

	column name	missing proportion
0	h1n1_concern	0.00
24	sex	0.00
23	race	0.00
21	age_group	0.00
1	h1n1_knowledge	0.00
8	behavioral_touch_face	0.00
7	behavioral_outside_home	0.00
6	behavioral_large_gatherings	0.00
5	behavioral_wash_hands	0.00
4	behavioral_face_mask	0.00
2	behavioral_antiviral_meds	0.00
36	seasonal_vaccine	0.00

- We will use 40% as the threshold for null frequency in the columns to decide whether to consider the column for analysis or not.
- Sometimes columns with greater null percentage have more semantic meaning and thus they cannot be ignored.
- Here we can see `health_insurance` , `employment_industry` and `employment_occupation` columns with approximately **46%, 50% and 50%** missing values percentages respectively, and thus they cannot be used for analysis as nearly half of the data is missing in those columns.

In [12]:
```
# Drop columns with high missing values percentages
merged_df.drop(columns=['health_insurance', 'employment_industry', 'employment_occupation'], inplace=True)
```

In [13]:
```
# Out target variable is h1n1_vaccine so drop seasonal_vaccine
# Drop columns with seasonal information
merged_df.drop(columns=['doctor_recc_seasonal','opinion_seas_vacc_effective',  'opinion_seas_risk',
                        'opinion_seas_sick_from_vacc', 'seasonal_vaccine'
                       ], inplace = True)
```

In [14]:
```
X = merged_df.drop(columns= 'h1n1_vaccine')
y = merged_df['h1n1_vaccine']
```

In [15]:
```
# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state= 40)
```

Impute missing values

In [16]:
```
# Categorical columns of X_train
categorical_X_train = X_train.select_dtypes(include=['object'])
categorical_X_train.columns
```

Out[16]:
```
Index(['age_group', 'education', 'race', 'sex', 'income_poverty',
       'marital_status', 'rent_or_own', 'employment_status', 'hhs_geo_region',
       'census_msa'],
      dtype='object')
```

In [17]:
```
# Categorical columns of X_test
categorical_X_test = X_test.select_dtypes(include=['object'])
categorical_X_test.columns
```

Out[17]:
```
Index(['age_group', 'education', 'race', 'sex', 'income_poverty',
       'marital_status', 'rent_or_own', 'employment_status', 'hhs_geo_region',
       'census_msa'],
      dtype='object')
```

In [18]:
```
# Numerical columns of X_train
numerical_X_train = X_train.select_dtypes(exclude=['object'])
numerical_X_train.columns
```

Out[18]:
```
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', 'household_adults',
       'household_children'],
      dtype='object')
```

In [19]:
```
# Numerical columns of X_test
numerical_X_test = X_test.select_dtypes(exclude=['object'])
numerical_X_test.columns
```

Out[19]:
```
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', 'household_adults',
       'household_children'],
      dtype='object')
```

In [20]:
```
# Imputation of mising values on train set data [for numerical data]
# Instantiate imputer
imputer = IterativeImputer()

# Fit and tranform X_train
imputed_numerical_X_train = imputer.fit_transform(numerical_X_train)
imputed_numerical_X_train = pd.DataFrame(np.round(numerical_X_train, 0), columns = numerical_X_train.columns)
imputed_numerical_X_train
```

Out[20]:

	h1n1_concern	h1n1_knowledge	behavioral_antiviral_meds	behavioral_avoidance	behavioral_face_mask	behavioral_wash_hands	bel
respondent_id							
7191	1.0	1.0	0.0	1.0	0.0	1.0	
23784	3.0	1.0	0.0	1.0	0.0	1.0	
316	2.0	1.0	0.0	1.0	0.0	1.0	
521	2.0	1.0	0.0	1.0	0.0	1.0	
21647	2.0	2.0	0.0	1.0	0.0	1.0	
...	...	...	...	...	...	...	
21810	2.0	1.0	0.0	1.0	0.0	0.0	
23992	1.0	0.0	0.0	0.0	0.0	0.0	
14501	2.0	1.0	0.0	1.0	0.0	1.0	
14555	2.0	1.0	0.0	0.0	0.0	1.0	
11590	1.0	2.0	0.0	1.0	0.0	1.0	

20030 rows × 18 columns

In [21]:
```
# Tranform X_test
imputed_numerical_X_test = imputer.transform(numerical_X_test)
imputed_numerical_X_test = pd.DataFrame(np.round(numerical_X_test, 0), columns = numerical_X_test.columns)
imputed_numerical_X_test
```

Out[21]:

	h1n1_concern	h1n1_knowledge	behavioral_antiviral_meds	behavioral_avoidance	behavioral_face_mask	behavioral_wash_hands	bel
respondent_id							
6458	1.0	0.0	0.0	1.0	0.0	1.0	
12839	2.0	1.0	0.0	1.0	0.0	1.0	
4377	3.0	2.0	0.0	0.0	0.0	1.0	
2731	2.0	1.0	0.0	1.0	0.0	1.0	
4982	3.0	2.0	0.0	1.0	0.0	1.0	
...	...	...	...	...	...	...	
18368	1.0	1.0	0.0	1.0	0.0	1.0	
7212	2.0	0.0	0.0	1.0	0.0	1.0	
13165	2.0	0.0	0.0	0.0	0.0	1.0	
17397	3.0	2.0	1.0	1.0	1.0	1.0	
20852	3.0	2.0	0.0	1.0	0.0	1.0	

6677 rows × 18 columns

In [22]:
```
# Encode categorical variables
# Instatiate encoder class
encoder = OneHotEncoder(drop= 'first', handle_unknown = 'ignore', sparse_output= False)

# Fit and transform X_train data
encoded_categorical_X_train = encoder.fit_transform(categorical_X_train)
encoded_categorical_X_train = pd.DataFrame(encoded_categorical_X_train, columns = encoder.get_feature_names_out(),
                                           index= categorical_X_train.index)
encoded_categorical_X_train
```

Out[22]:

respondent_id	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	education_nan	race
7191	0.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0	
23784	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	
316	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	
521	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0	
21647	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	
...	...	...	...	...	...	...	...	...	
21810	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0	
23992	0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	
14501	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	
14555	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	
11590	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	

20030 rows × 33 columns

In [23]:
```
# Check for columns with all zeros to drop
pd.DataFrame([[column, encoded_categorical_X_train[column].sum() == 0] for column in encoded_categorical_X_train.colum
```

Out[23]:

	0	1
0	age_group_35 - 44 Years	False
1	age_group_45 - 54 Years	False
2	age_group_55 - 64 Years	False
3	age_group_65+ Years	False
4	education_< 12 Years	False
5	education_College Graduate	False
6	education_Some College	False
7	education_nan	False
8	race_Hispanic	False
9	race_Other or Multiple	False
10	race_White	False
11	sex_Male	False
12	income_poverty_> $75,000	False
13	income_poverty_Below Poverty	False
14	income_poverty_nan	False
15	marital_status_Not Married	False
16	marital_status_nan	False
17	rent_or_own_Rent	False
18	rent_or_own_nan	False
19	employment_status_Not in Labor Force	False
20	employment_status_Unemployed	False
21	employment_status_nan	False
22	hhs_geo_region_bhuqouqj	False
23	hhs_geo_region_dqpwygqj	False
24	hhs_geo_region_fpwskwrf	False

	0	1
25	hhs_geo_region_kbazzjca	False
26	hhs_geo_region_lrircsnp	False
27	hhs_geo_region_lzgpxyit	False
28	hhs_geo_region_mlyzmhmf	False
29	hhs_geo_region_oxchjgsf	False
30	hhs_geo_region_qufhixun	False
31	census_msa_MSA, Principle City	False
32	census_msa_Non-MSA	False

In [24]:
```
# Transform X_test data [encoder]
encoded_categorical_X_test = encoder.transform(categorical_X_test)
encoded_categorical_X_test = pd.DataFrame(encoded_categorical_X_test, columns = encoder.get_feature_names_out(),
                                         index = categorical_X_test.index)
encoded_categorical_X_test
```

Out[24]:

respondent_id	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	education_nan	race
6458	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	
12839	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0	
4377	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0	
2731	1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	
4982	1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	
...	...	...	...	...	...	...	...	...	
18368	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
7212	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
13165	0.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	
17397	0.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	
20852	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0	

6677 rows × 33 columns

In [25]:
```
# Check for columns with all zeros to drop
pd.DataFrame([[column, encoded_categorical_X_test[column].sum() == 0] for column in encoded_categorical_X_test.columns
```

Out[25]:

	0	1
0	age_group_35 - 44 Years	False
1	age_group_45 - 54 Years	False
2	age_group_55 - 64 Years	False
3	age_group_65+ Years	False
4	education_< 12 Years	False
5	education_College Graduate	False
6	education_Some College	False
7	education_nan	False
8	race_Hispanic	False
9	race_Other or Multiple	False
10	race_White	False
11	sex_Male	False
12	income_poverty_> $75,000	False
13	income_poverty_Below Poverty	False
14	income_poverty_nan	False
15	marital_status_Not Married	False
16	marital_status_nan	False
17	rent_or_own_Rent	False
18	rent_or_own_nan	False
19	employment_status_Not in Labor Force	False
20	employment_status_Unemployed	False
21	employment_status_nan	False
22	hhs_geo_region_bhuqouqj	False
23	hhs_geo_region_dqpwygqj	False
24	hhs_geo_region_fpwskwrf	False

	0	1
25	hhs_geo_region_kbazzjca	False
26	hhs_geo_region_lrircsnp	False
27	hhs_geo_region_lzgpxyit	False
28	hhs_geo_region_mlyzmhmf	False
29	hhs_geo_region_oxchjgsf	False
30	hhs_geo_region_qufhixun	False
31	census_msa_MSA, Principle City	False
32	census_msa_Non-MSA	False

In [26]:
```
# Back concatenate numerical and categorical columns for both training and test data sets
# Training data set
X_train_clean = pd.concat([imputed_numerical_X_train, encoded_categorical_X_train], axis = 1)
```

In [27]:
```
# Test data set
X_test_clean = pd.concat([imputed_numerical_X_test, encoded_categorical_X_test], axis = 1)
```

In [28]:
```
X_train_clean.isna().sum()
```

Out[28]:
```
h1n1_concern                              68
h1n1_knowledge                            88
behavioral_antiviral_meds                 52
behavioral_avoidance                     159
behavioral_face_mask                      18
behavioral_wash_hands                     35
behavioral_large_gatherings               60
behavioral_outside_home                   59
behavioral_touch_face                     99
doctor_recc_h1n1                        1627
chronic_med_condition                    706
child_under_6_months                     602
health_worker                            587
opinion_h1n1_vacc_effective              276
opinion_h1n1_risk                        269
opinion_h1n1_sick_from_vacc              278
household_adults                         171
household_children                       171
age_group_35 - 44 Years                    0
age_group_45 - 54 Years                    0
age_group_55 - 64 Years                    0
age_group_65+ Years                        0
education_< 12 Years                       0
education_College Graduate                 0
education_Some College                     0
education_nan                              0
race_Hispanic                              0
race_Other or Multiple                     0
race_White                                 0
sex_Male                                   0
income_poverty_> $75,000                   0
income_poverty_Below Poverty               0
income_poverty_nan                         0
marital_status_Not Married                 0
marital_status_nan                         0
rent_or_own_Rent                           0
rent_or_own_nan                            0
employment_status_Not in Labor Force       0
employment_status_Unemployed               0
employment_status_nan                      0
hhs_geo_region_bhuqouqj                    0
```

```
hhs_geo_region_dqpwygqj               0
hhs_geo_region_fpwskwrf               0
hhs_geo_region_kbazzjca               0
hhs_geo_region_lrircsnp               0
hhs_geo_region_lzgpxyit               0
hhs_geo_region_mlyzmhmf               0
hhs_geo_region_oxchjgsf               0
hhs_geo_region_qufhixun               0
census_msa_MSA, Principle City        0
census_msa_Non-MSA                    0
dtype: int64
```

In [29]:
```
X_test_clean.isna().sum()
```

Out[29]:
```
h1n1_concern                            24
h1n1_knowledge                          28
behavioral_antiviral_meds               19
behavioral_avoidance                    49
behavioral_face_mask                     1
behavioral_wash_hands                    7
behavioral_large_gatherings             27
behavioral_outside_home                 23
behavioral_touch_face                   29
doctor_recc_h1n1                       533
chronic_med_condition                  265
child_under_6_months                   218
health_worker                          217
opinion_h1n1_vacc_effective            115
opinion_h1n1_risk                      119
opinion_h1n1_sick_from_vacc            117
household_adults                        78
household_children                      78
age_group_35 - 44 Years                  0
age_group_45 - 54 Years                  0
age_group_55 - 64 Years                  0
age_group_65+ Years                      0
education_< 12 Years                     0
education_College Graduate               0
education_Some College                   0
education_nan                            0
race_Hispanic                            0
race_Other or Multiple                   0
race_White                               0
sex_Male                                 0
income_poverty_> $75,000                 0
income_poverty_Below Poverty             0
income_poverty_nan                       0
marital_status_Not Married               0
marital_status_nan                       0
rent_or_own_Rent                         0
rent_or_own_nan                          0
employment_status_Not in Labor Force     0
employment_status_Unemployed             0
employment_status_nan                    0
hhs_geo_region_bhuqouqj                  0
```

```
hhs_geo_region_dqpwygqj              0
hhs_geo_region_fpwskwrf              0
hhs_geo_region_kbazzjca              0
hhs_geo_region_lrircsnp              0
hhs_geo_region_lzgpxyit              0
hhs_geo_region_mlyzmhmf              0
hhs_geo_region_oxchjgsf              0
hhs_geo_region_qufhixun              0
census_msa_MSA, Principle City       0
census_msa_Non-MSA                   0
dtype: int64
```

In [30]:
```
# Impute more missing values on X_train data set
X_train = imputer.fit_transform(X_train_clean)
X_train = pd.DataFrame(np.round(X_train, 0), columns = X_train_clean.columns, index = X_train_clean.index)
X_train
```

Out[30]:

	h1n1_concern	h1n1_knowledge	behavioral_antiviral_meds	behavioral_avoidance	behavioral_face_mask	behavioral_wash_hands	bel
respondent_id							
7191	1.0	1.0	0.0	1.0	0.0	1.0	
23784	3.0	1.0	0.0	1.0	0.0	1.0	
316	2.0	1.0	0.0	1.0	0.0	1.0	
521	2.0	1.0	0.0	1.0	0.0	1.0	
21647	2.0	2.0	0.0	1.0	0.0	1.0	
...	...	...	...	...	...	...	
21810	2.0	1.0	0.0	1.0	0.0	0.0	
23992	1.0	0.0	0.0	0.0	0.0	0.0	
14501	2.0	1.0	0.0	1.0	0.0	1.0	
14555	2.0	1.0	0.0	0.0	0.0	1.0	
11590	1.0	2.0	0.0	1.0	0.0	1.0	

20030 rows × 51 columns

In [31]:
```
# Impute more missing values on X_test data set
X_test = imputer.fit_transform(X_test_clean)
X_test = pd.DataFrame(np.round(X_test, 0), columns = X_test_clean.columns, index= X_test_clean.index)
X_test
```

Out[31]:

	h1n1_concern	h1n1_knowledge	behavioral_antiviral_meds	behavioral_avoidance	behavioral_face_mask	behavioral_wash_hands	bel
respondent_id							
6458	1.0	0.0	0.0	1.0	0.0	1.0	
12839	2.0	1.0	0.0	1.0	0.0	1.0	
4377	3.0	2.0	0.0	0.0	0.0	1.0	
2731	2.0	1.0	0.0	1.0	0.0	1.0	
4982	3.0	2.0	0.0	1.0	0.0	1.0	
...	...	...	...	...	...	...	
18368	1.0	1.0	0.0	1.0	0.0	1.0	
7212	2.0	0.0	0.0	1.0	0.0	1.0	
13165	2.0	0.0	0.0	0.0	0.0	1.0	
17397	3.0	2.0	1.0	1.0	1.0	1.0	
20852	3.0	2.0	0.0	1.0	0.0	1.0	

6677 rows × 51 columns

In [32]:
```
X_train.isna().sum()
```

Out[32]:
```
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
household_adults                         0
household_children                       0
age_group_35 - 44 Years                  0
age_group_45 - 54 Years                  0
age_group_55 - 64 Years                  0
age_group_65+ Years                      0
education_< 12 Years                     0
education_College Graduate               0
education_Some College                   0
education_nan                            0
race_Hispanic                            0
race_Other or Multiple                   0
race_White                               0
sex_Male                                 0
income_poverty_> $75,000                 0
income_poverty_Below Poverty             0
income_poverty_nan                       0
marital_status_Not Married               0
marital_status_nan                       0
rent_or_own_Rent                         0
rent_or_own_nan                          0
employment_status_Not in Labor Force     0
employment_status_Unemployed             0
employment_status_nan                    0
hhs_geo_region_bhuqouqj                  0
```

```
hhs_geo_region_dqpwygqj            0
hhs_geo_region_fpwskwrf            0
hhs_geo_region_kbazzjca            0
hhs_geo_region_lrircsnp            0
hhs_geo_region_lzgpxyit            0
hhs_geo_region_mlyzmhmf            0
hhs_geo_region_oxchjgsf            0
hhs_geo_region_qufhixun            0
census_msa_MSA, Principle City     0
census_msa_Non-MSA                 0
dtype: int64
```

In [33]:
```
X_test.isna().sum()
```

Out[33]:
```
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
household_adults                       0
household_children                     0
age_group_35 - 44 Years                0
age_group_45 - 54 Years                0
age_group_55 - 64 Years                0
age_group_65+ Years                    0
education_< 12 Years                   0
education_College Graduate             0
education_Some College                 0
education_nan                          0
race_Hispanic                          0
race_Other or Multiple                 0
race_White                             0
sex_Male                               0
income_poverty_> $75,000               0
income_poverty_Below Poverty           0
income_poverty_nan                     0
marital_status_Not Married             0
marital_status_nan                     0
rent_or_own_Rent                       0
rent_or_own_nan                        0
employment_status_Not in Labor Force   0
employment_status_Unemployed           0
employment_status_nan                  0
hhs_geo_region_bhuqouqj                0
```

```
hhs_geo_region_dqpwygqj                 0
hhs_geo_region_fpwskwrf                 0
hhs_geo_region_kbazzjca                 0
hhs_geo_region_lrircsnp                 0
hhs_geo_region_lzgpxyit                 0
hhs_geo_region_mlyzmhmf                 0
hhs_geo_region_oxchjgsf                 0
hhs_geo_region_qufhixun                 0
census_msa_MSA, Principle City          0
census_msa_Non-MSA                      0
dtype: int64
```

Feature Engineering

i) `behavior_score`

- By adding up all the behavioral variables, create a variable that shows how much a person has done behaviorally to avoid the virus, aside from getting vaccinated. These are all binary columns, where **1** denotes **YES**, indicating that the individual has taken a step to lower their risk of getting the flu. A higher score, calculated by adding the values of these columns, indicates a more circumspect and flu-conscious person.

In [34]:
```
# Get the columns with `behavior` attributes
behavior_cols = [col for col in X_train.columns if 'behavioral' in col]
behavior_cols
```

Out[34]:
```
['behavioral_antiviral_meds',
 'behavioral_avoidance',
 'behavioral_face_mask',
 'behavioral_wash_hands',
 'behavioral_large_gatherings',
 'behavioral_outside_home',
 'behavioral_touch_face']
```

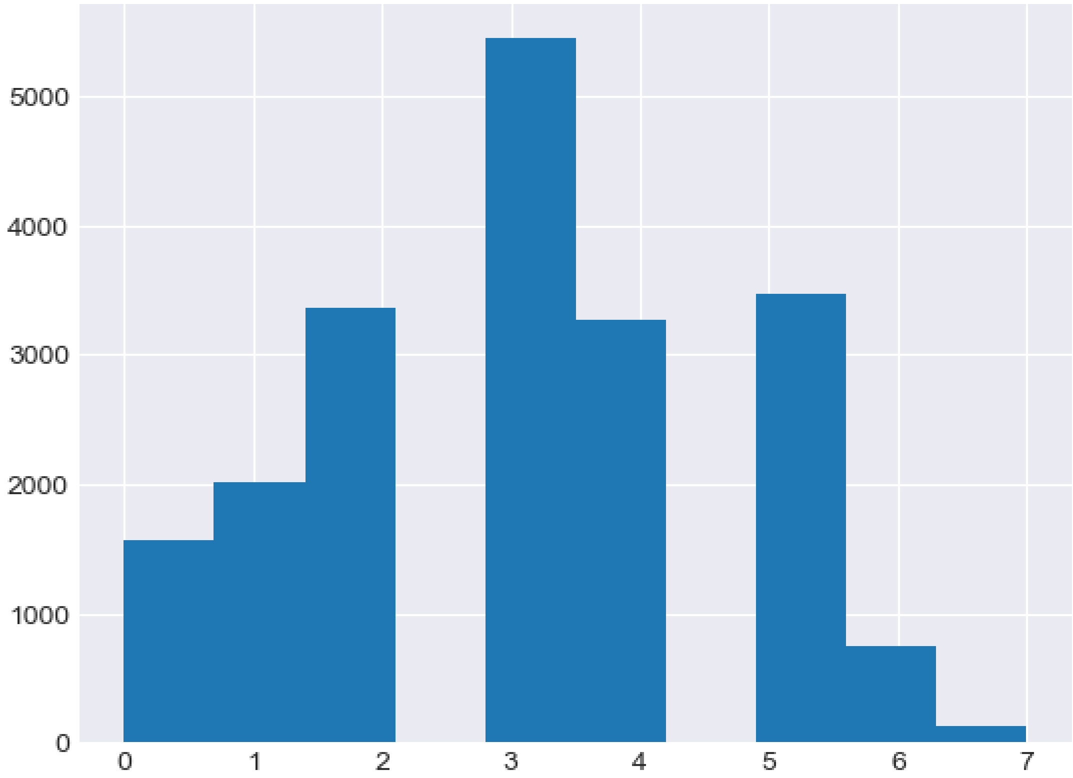
In [35]:
```
# Create the `behavior_score` variable for X_train data set
X_train['behavior_score'] = X_train[behavior_cols].sum(axis=1)
```

In [36]:
```
# Create the `behavior_score` variable for X_test data set
X_test['behavior_score'] = X_test[behavior_cols].sum(axis=1)
```

In [37]:
```
X_test.columns
```

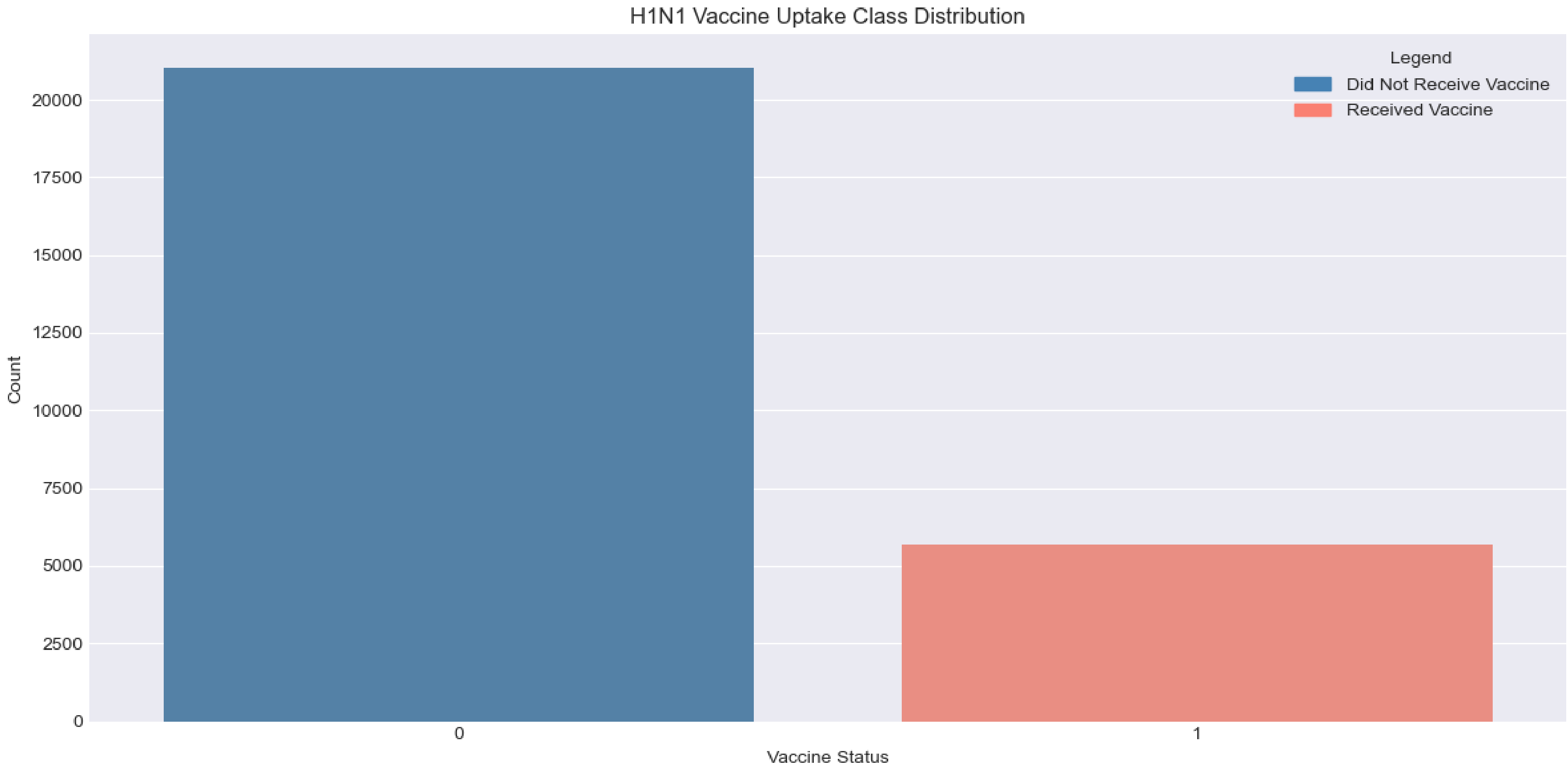
Out[37]:
```
Index(['h1n1_concern', 'h1n1_knowledge', 'behavioral_antiviral_meds',
       'behavioral_avoidance', 'behavioral_face_mask', 'behavioral_wash_hands',
       'behavioral_large_gatherings', 'behavioral_outside_home',
       'behavioral_touch_face', 'doctor_recc_h1n1', 'chronic_med_condition',
       'child_under_6_months', 'health_worker', 'opinion_h1n1_vacc_effective',
       'opinion_h1n1_risk', 'opinion_h1n1_sick_from_vacc', 'household_adults',
       'household_children', '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', 'education_nan',
       'race_Hispanic', 'race_Other or Multiple', 'race_White', 'sex_Male',
       'income_poverty_> $75,000', 'income_poverty_Below Poverty',
       'income_poverty_nan', 'marital_status_Not Married',
       'marital_status_nan', 'rent_or_own_Rent', 'rent_or_own_nan',
       'employment_status_Not in Labor Force', 'employment_status_Unemployed',
       'employment_status_nan', 'hhs_geo_region_bhuqouqj',
       'hhs_geo_region_dqpwygqj', 'hhs_geo_region_fpwskwrf',
       'hhs_geo_region_kbazzjca', 'hhs_geo_region_lrircsnp',
       'hhs_geo_region_lzgpxyit', 'hhs_geo_region_mlyzmhmf',
       'hhs_geo_region_oxchjgsf', 'hhs_geo_region_qufhixun',
       'census_msa_MSA, Principle City', 'census_msa_Non-MSA',
       'behavior_score'],
      dtype='object')
```

In [38]:
```
# Plot the distribution of behavior score variable we created
plt.hist(X_train['behavior_score']);
```



Class Imbalance

In [39]:
```
# Checking for class distribution
fig, ax = plt.subplots(figsize=(12, 6))
sns.countplot(x='h1n1_vaccine', data=merged_df, ax=ax, palette=["steelblue", "salmon"])

ax.set_title('H1N1 Vaccine Uptake Class Distribution')
ax.set_xlabel('Vaccine Status')
ax.set_ylabel('Count')

# Add separate legend labels for each class
legend_labels = ['Did Not Receive Vaccine', "Received Vaccine"]
colors = ["steelblue", "salmon"]
legend_handles = [plt.Rectangle((0, 0), 1, 1, color=color) for color in colors]
ax.legend(legend_handles, legend_labels, title='Legend')

plt.tight_layout()
plt.savefig('Visualization1')
```



- This plot shows the uptake distribution between those who received the vaccine and those who did not receive the vaccine

In [40]:
```
# Checking class imbalance for h1n1_vaccine [target variable]
y.value_counts()
```

Out[40]:
```
h1n1_vaccine
0    21033
1     5674
Name: count, dtype: int64
```

Presence of class imbalance

In [41]:
```
# Address class imbalance [only uses training set]
smote_df = pd.concat([X_train, y_train], axis = 1)

# Instantiate SMOTE class
smote = SMOTE()

# Fit and transform data
X_train_resample, y_train_resample = smote.fit_resample(X_train, y_train)
y_train_resample.value_counts()
```

Out[41]:
```
h1n1_vaccine
1    15779
0    15779
Name: count, dtype: int64
```

Exploratory Data Analysis

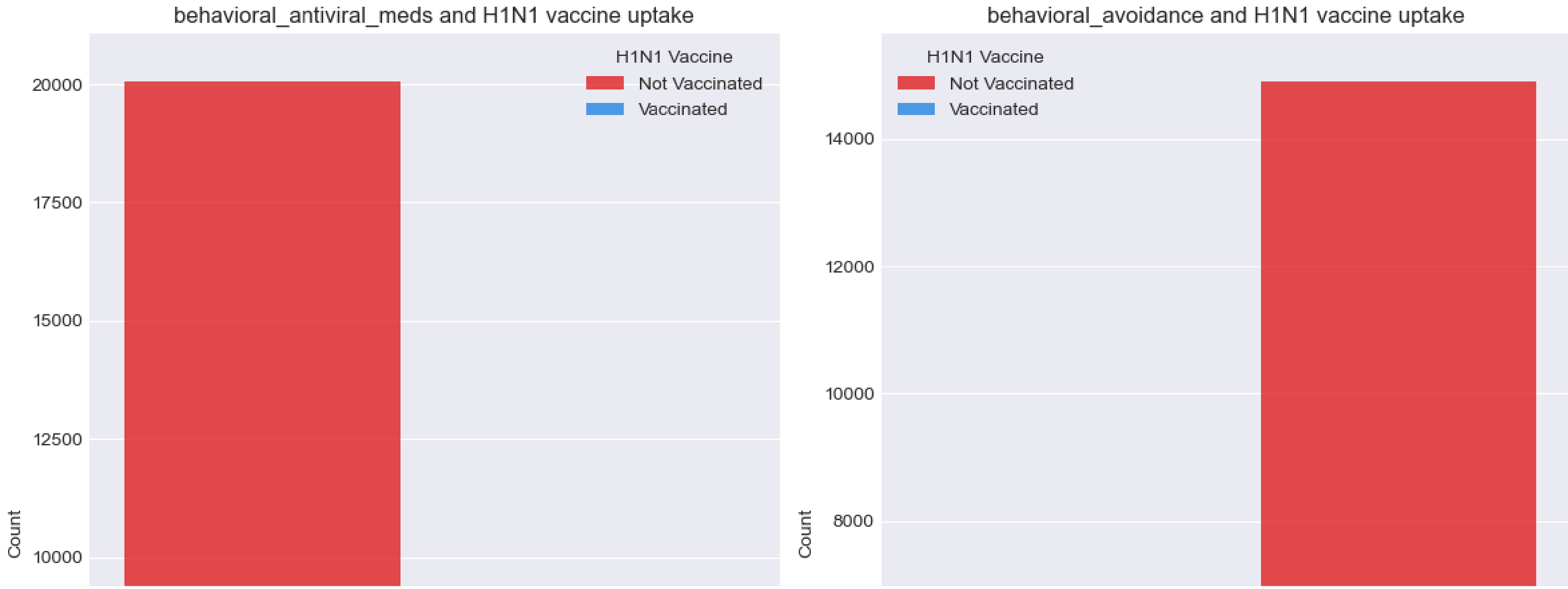
Checking the relationshop between our target variable **h1n1_vaccine** and the features.

In [42]:
```
print(merged_df['h1n1_vaccine'].unique())
```

```
[0 1]
```

In [43]:
```
# List of columns to plot
columns = ['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', 'marital_status',
           'rent_or_own', 'employment_status', 'household_adults', 'household_children']

# Select columns for plotting
columns_to_plot = columns[:25]
# Determine the number of rows and columns for subplots
num_rows = (len(columns_to_plot) + 1) // 2
num_cols = 2
# Create subplots
fig, axes = plt.subplots(num_rows, num_cols, figsize=(12, 8 * num_rows))
axes = axes.flatten()
# Iterate over the columns and create count plots
for i, column in enumerate(columns_to_plot):
    # Check if the 'h1n1_vaccine' column is present in the DataFrame
    if 'h1n1_vaccine' in merged_df.columns:
        unique_values = merged_df[column].unique()
        if len(unique_values) > 1:  # Proceed only if there are multiple unique values
            sns.countplot(x=column, data=merged_df[merged_df['h1n1_vaccine'] == 0], ax=axes[i], color='red', label='No
            sns.countplot(x=column, data=merged_df[merged_df['h1n1_vaccine'] == 1], ax=axes[i], color='dodgerblue', lal
            axes[i].set_title(f"{column} and H1N1 vaccine uptake")
            axes[i].set_xlabel('')
            axes[i].set_ylabel('Count')
            axes[i].legend(title='H1N1 Vaccine', loc='best')
        else:
            axes[i].set_title(f"{column} (only one value)")
            axes[i].set_xlabel('')
            axes[i].set_ylabel('Count')
# Remove any empty subplots
if len(columns_to_plot) % 2 == 1:
    fig.delaxes(axes[-1])
# Adjust spacing between subplots
plt.tight_layout()
# Display the plot
```

```
plt.show()
```



Modeling

Logistic Regression

In [44]:
```
# Baseline model

# Instantiate classifier
baseline_logreg = LogisticRegression(C= 1e12, random_state=40, solver='liblinear')

# Fit model to train set
baseline_logreg.fit(X_train_resample, y_train_resample)
```

Out[44]:
```
LogisticRegression
LogisticRegression(C=1000000000000.0, random_state=40, solver='liblinear')
```

In [45]:
```
# Accuracy on training data set
baseline_preds = baseline_logreg.predict(X_train_resample)

print(f"accuracy score: {accuracy_score(y_train_resample, baseline_preds)}")
print(f"f1 score: {f1_score(y_train_resample, baseline_preds)}")
print(f"precision score: {precision_score(y_train_resample, baseline_preds)}")
print(f"recall score: {recall_score(y_train_resample, baseline_preds)}")
```

```
accuracy score: 0.7676658850370746
f1 score: 0.7612970438859226
precision score: 0.7827542344513624
recall score: 0.740984853286013
```

In [46]:
```
# Accuracy on test data set
baseline_preds = baseline_logreg.predict(X_test)

print(f"accuracy score: {accuracy_score(y_test, baseline_preds)}")
print(f"f1 score: {f1_score(y_test, baseline_preds)}")
print(f"precision score: {precision_score(y_test, baseline_preds)}")
print(f"recall score: {recall_score(y_test, baseline_preds)}")
```

```
accuracy score: 0.7753482102740752
f1 score: 0.5774647887323944
precision score: 0.48189938881053124
recall score: 0.7203092059030218
```

In [47]:
```
# Model2

# Instantiate classifier
logreg_2 = LogisticRegression(penalty = 'l1', C= 1e12, random_state=40, solver='liblinear')

# Fit model to train set
logreg_2.fit(X_train_resample, y_train_resample)
```

Out[47]:
```
LogisticRegression
LogisticRegression(C=1000000000000.0, penalty='l1', random_state=40,
                   solver='liblinear')
```

In [48]:
```
# Accuracy on training data set
logreg2_preds = logreg_2.predict(X_train_resample)

print(f"accuracy score: {accuracy_score(y_train_resample, logreg2_preds)}")
print(f"f1 score: {f1_score(y_train_resample, logreg2_preds)}")
print(f"precision score: {precision_score(y_train_resample, logreg2_preds)}")
print(f"recall score: {recall_score(y_train_resample, logreg2_preds)}")
```

```
accuracy score: 0.7676975727232398
f1 score: 0.76132182972489
precision score: 0.7828066416711301
recall score: 0.740984853286013
```

In [49]:
```
# Accuracy on test data set
logreg2_preds = logreg_2.predict(X_test)

print(f"accuracy score: {accuracy_score(y_test, logreg2_preds)}")
print(f"f1 score: {f1_score(y_test, logreg2_preds)}")
print(f"precision score: {precision_score(y_test, logreg2_preds)}")
print(f"recall score: {recall_score(y_test, logreg2_preds)}")
```

```
accuracy score: 0.775198442414258
f1 score: 0.5773021684032668
precision score: 0.4816729323308271
recall score: 0.7203092059030218
```

In [50]:
```
# Model 3

# Instantiate classifier
logreg_3 = LogisticRegression(penalty = 'l2', random_state=40, solver='lbfgs')

# Fit model to train set
logreg_3.fit(X_train_resample, y_train_resample)
```

```
C:\Users\Richard.LAPTOP-2023AAHO\anaconda3\Lib\site-packages\sklearn\linear_model\_logistic.py:458: ConvergenceWarnin
g: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scikit-learn.org/stable/modules/preprocessin
g.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression (https://scikit-learn.org/stable/mo
dules/linear_model.html#logistic-regression)
  n_iter_i = _check_optimize_result(
```

Out[50]:
```
▾ LogisticRegression
LogisticRegression(random_state=40)
```

In [51]:
```
# Accuracy on training data set
logreg3_preds = logreg_3.predict(X_train_resample)

print(f"accuracy score: {accuracy_score(y_train_resample, logreg3_preds)}")
print(f"f1 score: {f1_score(y_train_resample, logreg3_preds)}")
print(f"precision score: {precision_score(y_train_resample, logreg3_preds)}")
print(f"recall score: {recall_score(y_train_resample, logreg3_preds)}")
```

```
accuracy score: 0.7677926357817352
f1 score: 0.7613651165819982
precision score: 0.7830397213477125
recall score: 0.7408581025413524
```

In [52]:
```
# Accuracy on test data set
logreg3_preds = logreg_3.predict(X_test)

print(f"accuracy score: {accuracy_score(y_test, logreg3_preds)}")
print(f"f1 score: {f1_score(y_test, logreg3_preds)}")
print(f"precision score: {precision_score(y_test, logreg3_preds)}")
print(f"recall score: {recall_score(y_test, logreg3_preds)}")
```

```
accuracy score: 0.7754979781338924
f1 score: 0.5776275007044238
precision score: 0.4821260583254939
recall score: 0.7203092059030218
```

In [53]:
```
# ROC curve for the best Logistic regression model
fpr_logreg, tpr_logreg, _ = roc_curve(y_test, logreg_3.predict_proba(X_test)[:,1])
```

K-Nearest Neighbours

In [54]:
```
# Baseline model

# Instantiate classifier
baseline_KNN_clf = KNeighborsClassifier()

# Fit model to train set
baseline_KNN_clf.fit(X_train_resample, y_train_resample)
```

Out[54]:
```
▾ KNeighborsClassifier
KNeighborsClassifier()
```

In [55]:
```
# Accuracy on training data set
baseline_preds = baseline_KNN_clf.predict(X_train_resample)

print(f"accuracy score: {accuracy_score(y_train_resample, baseline_preds)}")
print(f"f1 score: {f1_score(y_train_resample, baseline_preds)}")
print(f"precision score: {precision_score(y_train_resample, baseline_preds)}")
print(f"recall score: {recall_score(y_train_resample, baseline_preds)}")
```

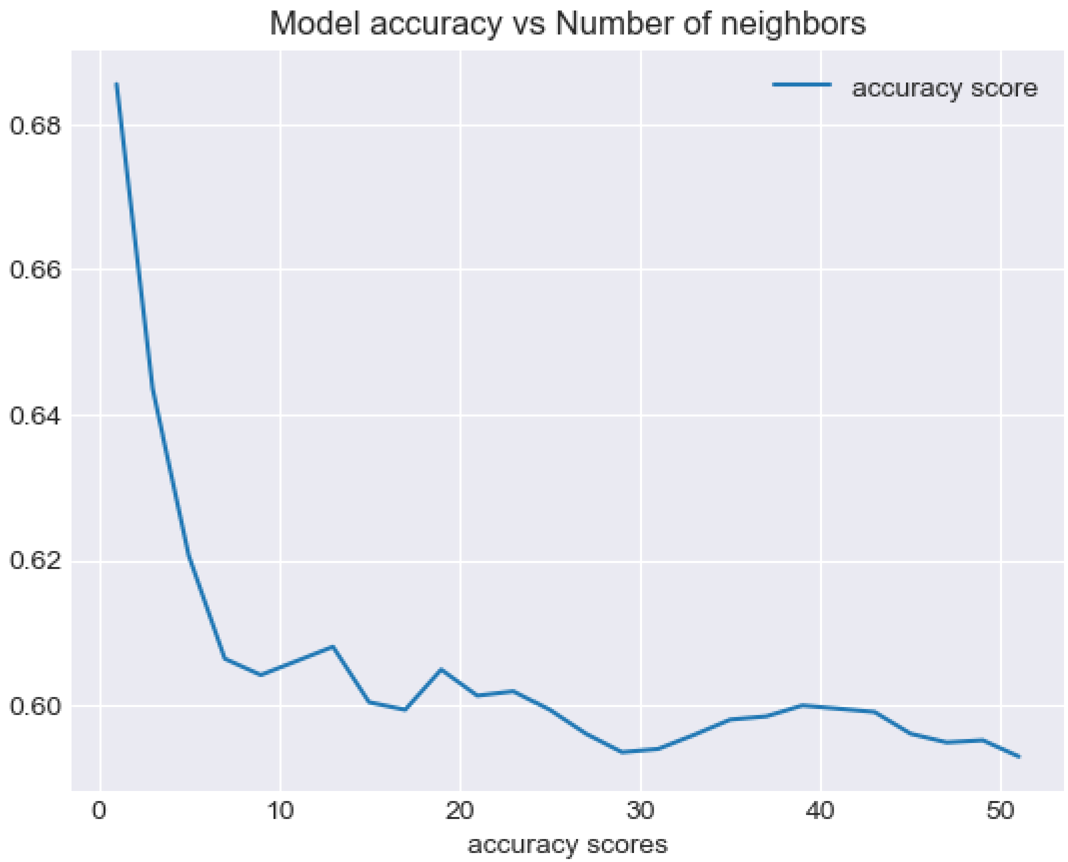
```
accuracy score: 0.8445085239875785
f1 score: 0.8642787996127783
precision score: 0.7667844522968198
recall score: 0.9901768172888016
```

In [56]:
```
# Accuracy on test data set
baseline_preds = baseline_KNN_clf.predict(X_test)

print(f"accuracy score: {accuracy_score(y_test, baseline_preds)}")
print(f"f1 score: {f1_score(y_test, baseline_preds)}")
print(f"precision score: {precision_score(y_test, baseline_preds)}")
print(f"recall score: {recall_score(y_test, baseline_preds)}")
```

```
accuracy score: 0.6206380110828216
f1 score: 0.4577178334403768
precision score: 0.3291256157635468
recall score: 0.7512297962052003
```

In [57]:
```
# Find best n_neighbors parameter
n_neighbors_params = []
accuracy_scores = []

for n in np.arange(1, 52, 2):
    n_neighbors_params.append(n)

    clf_model = KNeighborsClassifier(n_neighbors = n)
    clf_model.fit(X_train_resample, y_train_resample)
    acc = accuracy_score(y_test, clf_model.predict(X_test))
    accuracy_scores.append(acc)
```

In [58]:
```
plt.plot(n_neighbors_params, accuracy_scores, label ='accuracy score')
plt.xlabel('n_neighbors params values')
plt.xlabel('accuracy scores')
plt.title('Model accuracy vs Number of neighbors')
plt.legend();
```



In [59]:
```
# Scale training and test data sets
scaler = StandardScaler()

X_train_resample_scaled = scaler.fit_transform(X_train_resample)
X_test_scaled = scaler.transform(X_test)
```

In [60]:
```
# Model Iteration

# Instantiate classifier
KNN_clf2 = KNeighborsClassifier()

# Fit model to train set
KNN_clf2.fit(X_train_resample_scaled, y_train_resample)

# Accuracy of model on train set
print(f"accuracy score: {accuracy_score(y_train_resample, KNN_clf2.predict(X_train_resample_scaled))}")
print(f"f1 score: {f1_score(y_train_resample, KNN_clf2.predict(X_train_resample_scaled))}")
print(f"precision score: {precision_score(y_train_resample, KNN_clf2.predict(X_train_resample_scaled))}")
print(f"recall score: {recall_score(y_train_resample, KNN_clf2.predict(X_train_resample_scaled))}")
```

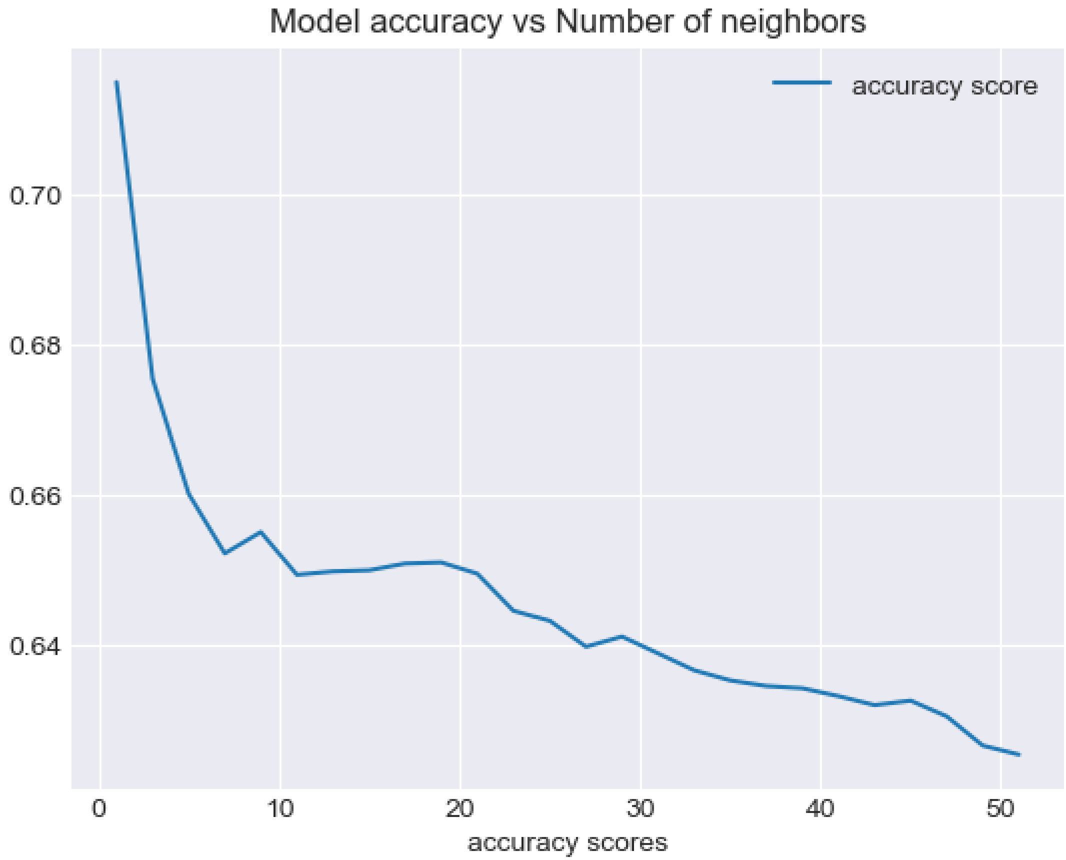
```
accuracy score: 0.8812979276253248
f1 score: 0.8923129994825505
precision score: 0.8165412742673752
recall score: 0.9835857785664491
```

In [61]:
```
# Accuracy of model on test set
print(f"accuracy score: {accuracy_score(y_test, KNN_clf2.predict(X_test_scaled))}")
print(f"f1 score: {f1_score(y_test, KNN_clf2.predict(X_test_scaled))}")
print(f"precision score: {precision_score(y_test, KNN_clf2.predict(X_test_scaled))}")
print(f"recall score: {recall_score(y_test, KNN_clf2.predict(X_test_scaled))}")
```

```
accuracy score: 0.6601767260745844
f1 score: 0.43175557225143996
precision score: 0.335408560311284
recall score: 0.6057624736472241
```

In [62]:
```
# Find best n_neighbors parameter
n_neighbors_params = []
accuracy_scores = []

for n in np.arange(1, 52, 2):
    n_neighbors_params.append(n)

    clf_model = KNeighborsClassifier(n_neighbors = n)
    clf_model.fit(X_train_resample_scaled, y_train_resample)
    acc = accuracy_score(y_test, clf_model.predict(X_test_scaled))
    accuracy_scores.append(acc)
```

In [63]:
```
plt.plot(n_neighbors_params, accuracy_scores, label ='accuracy score')
plt.xlabel('n_neighbors params values')
plt.xlabel('accuracy scores')
plt.title('Model accuracy vs Number of neighbors')
plt.legend();
```



Feature selection

In [64]:
```
# Define a custom function to compute feature importances
def custom_importance_getter_knn(estimator):
    # Extract the distances to the nearest neighbors
    distances, indices = estimator.kneighbors()
    # Compute the mean distance for each feature
    feature_importances = distances.mean(axis=0)
    return feature_importances

# Initialize RFE with the classifier and desired number of features to select
rfe = RFE(estimator= KNN_clf2, n_features_to_select= .8, step= 1, importance_getter= custom_importance_getter_knn)  # 

# Fit RFE on train data set
rfe.fit(X_train_resample_scaled, y_train_resample)

print(X_train_resample.columns)
print(rfe.ranking_)
print(rfe.support_)
```

```
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', 'household_adults',
       'household_children', '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', 'education_nan',
       'race_Hispanic', 'race_Other or Multiple', 'race_White', 'sex_Male',
       'income_poverty_> $75,000', 'income_poverty_Below Poverty',
       'income_poverty_nan', 'marital_status_Not Married',
       'marital_status_nan', 'rent_or_own_Rent', 'rent_or_own_nan',
       'employment_status_Not in Labor Force', 'employment_status_Unemployed',
       'employment_status_nan', 'hhs_geo_region_bhuqouqj',
       'hhs_geo_region_dqpwygqj', 'hhs_geo_region_fpwskwrf',
       'hhs_geo_region_kbazzjca', 'hhs_geo_region_lrircsnp',
       'hhs_geo_region_lzgpxyit', 'hhs_geo_region_mlyzmhmf',
       'hhs_geo_region_oxchjgsf', 'hhs_geo_region_qufhixun',
       'census_msa_MSA, Principle City', 'census_msa_Non-MSA',
       'behavior_score'],
      dtype='object')
[12 11 10  9  8  7  6  5  4  3  2  1  1  1  1  1  1  1  1  1  1  1  1  1
  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
  1  1  1  1]
[False False False False False False False False False False False  True
  True  True  True  True  True  True  True  True  True  True  True  True
  True  True  True  True  True  True  True  True  True  True  True  True
  True  True  True  True  True  True  True  True  True  True  True  True
  True  True  True  True]
```

In [65]:
```
masked_columns = X_train_resample.columns[rfe.get_support(indices= True)]
```

In [66]:
```
X_train_rfe = X_train_resample_scaled[:,0:11]
X_test_rfe = X_test_scaled[:,0:11]
```

In [67]:
```
# Instantiate classifier
KNN_clf3 = KNeighborsClassifier()

# Fit model to train set
KNN_clf3.fit(X_train_rfe, y_train_resample)
```

Out[67]:
```
▾ KNeighborsClassifier
KNeighborsClassifier()
```

In [68]:
```
# Accuracy of model on train set
print(f"accuracy score: {accuracy_score(y_train_resample, KNN_clf3.predict(X_train_rfe))}")
print(f"f1 score: {f1_score(y_train_resample, KNN_clf3.predict(X_train_rfe))}")
print(f"precision score: {precision_score(y_train_resample, KNN_clf3.predict(X_train_rfe))}")
print(f"recall score: {recall_score(y_train_resample, KNN_clf3.predict(X_train_rfe))}")
```

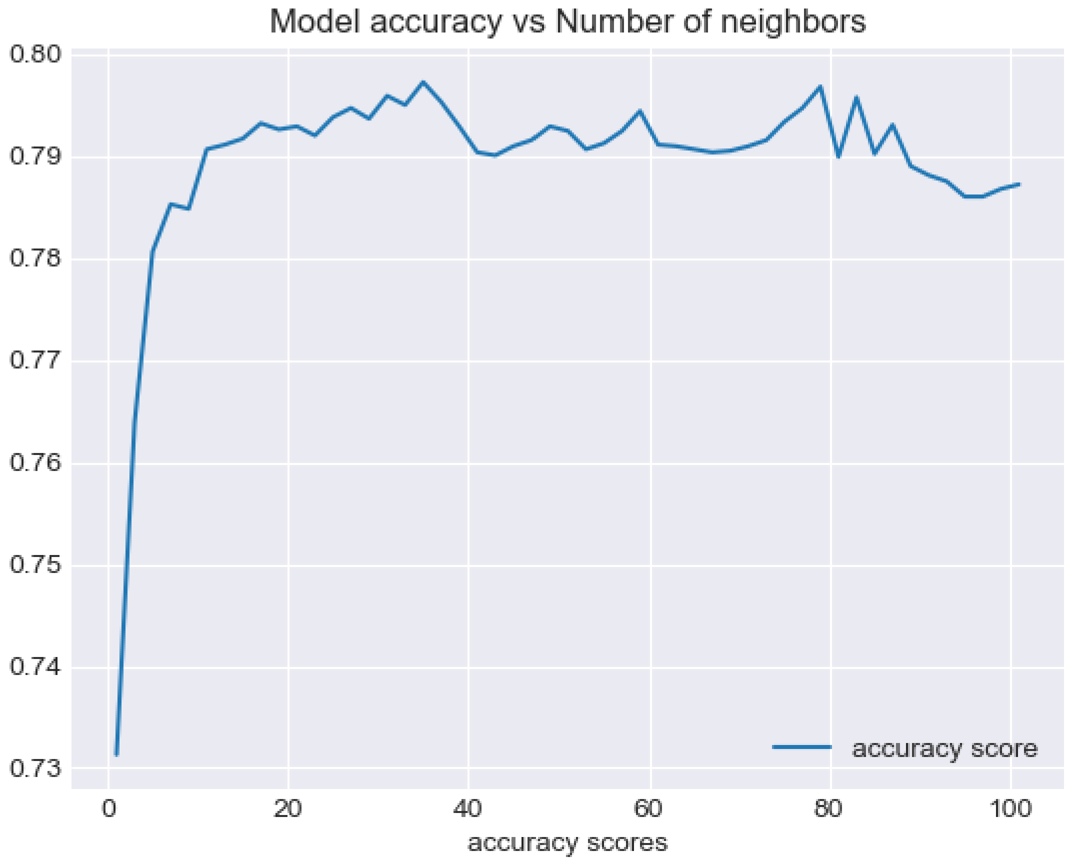
```
accuracy score: 0.8234045250015843
f1 score: 0.8061497791227521
precision score: 0.8934464148033925
recall score: 0.7343938145636606
```

In [69]:
```
# Accuracy of model on test set
print(f"accuracy score: {accuracy_score(y_test, KNN_clf3.predict(X_test_rfe))}")
print(f"f1 score: {f1_score(y_test, KNN_clf3.predict(X_test_rfe))}")
print(f"precision score: {precision_score(y_test, KNN_clf3.predict(X_test_rfe))}")
print(f"recall score: {recall_score(y_test, KNN_clf3.predict(X_test_rfe))}")
```

```
accuracy score: 0.78059008536768
f1 score: 0.4104627766599598
precision score: 0.480225988700565
recall score: 0.3583977512297962
```

In [70]:
```
# Find best n_neighbors parameter
n_neighbors_params = []
accuracy_scores = []

for n in np.arange(1, 102, 2):
    n_neighbors_params.append(n)

    clf_model = KNeighborsClassifier(n_neighbors = n)
    clf_model.fit(X_train_rfe, y_train_resample)
    acc = accuracy_score(y_test, clf_model.predict(X_test_rfe))
    accuracy_scores.append(acc)
```

In [71]:
```
plt.plot(n_neighbors_params, accuracy_scores, label ='accuracy score')
plt.xlabel('n_neighbors params values')
plt.xlabel('accuracy scores')
plt.title('Model accuracy vs Number of neighbors')
plt.legend();
```



In [72]:
```
# Best n_neighbors parameter is 51
index = accuracy_scores.index(max(accuracy_scores))
n_neighbors_params[index]
```

Out[72]: 35

In [73]:
```
# Instantiate classifier
KNN_clf_final = KNeighborsClassifier(n_neighbors = 51)

# Fit model to train set
KNN_clf_final.fit(X_train_rfe, y_train_resample)
```

Out[73]:
```
KNeighborsClassifier

KNeighborsClassifier(n_neighbors=51)
```

In [74]:
```
# Accuracy of model on train set
print(f"accuracy score on train set: {accuracy_score(y_train_resample, KNN_clf_final.predict(X_train_rfe))}")
print(f"f1 score on train set: {f1_score(y_train_resample, KNN_clf_final.predict(X_train_rfe))}")
print(f"precision score on train set: {precision_score(y_train_resample, KNN_clf_final.predict(X_train_rfe))}")
print(f"recall score on train set: {recall_score(y_train_resample, KNN_clf_final.predict(X_train_rfe))}")
print("\n------------------------------------------------------------\n")

# Accuracy of model on test set
print(f"accuracy score on test set: {accuracy_score(y_test, KNN_clf_final.predict(X_test_rfe))}")
print(f"f1 score on test set: {f1_score(y_test, KNN_clf_final.predict(X_test_rfe))}")
print(f"precision score on test set: {precision_score(y_test, KNN_clf_final.predict(X_test_rfe))}")
print(f"recall score on test set: {recall_score(y_test, KNN_clf_final.predict(X_test_rfe))}")
```

```
accuracy score on train set: 0.7502059699600735
f1 score on train set: 0.7136890277121999
precision score on train set: 0.8358856559469117
recall score on train set: 0.6226630331453197

------------------------------------------------------------

accuracy score on test set: 0.7924217462932455
f1 score on test set: 0.49416058394160584
precision score on test set: 0.5140470766894457
recall score on test set: 0.47575544624033733
```

Cross validation

In [75]:
```
results = cross_validate(KNeighborsClassifier(n_neighbors = 51), X_train_rfe, y_train_resample,
              scoring = ['f1', 'accuracy', 'precision', 'recall'])
```

In [76]:
```
print(f"test f1 score with cross validation: {np.mean(results['test_f1'])}")
print(f"test accuracy with cross validation: {np.mean(results['test_accuracy'])}")
print(f"test precision with cross validation: {np.mean(results['test_precision'])}")
print(f"test recall with cross validation: {np.mean(results['test_recall'])}")
```

```
test f1 score with cross validation: 0.6912165493582576
test accuracy with cross validation: 0.7353142881873421
test precision with cross validation: 0.8197542477475629
test recall with cross validation: 0.6014356072703315
```

In [77]:
```
# ROC curve for the best KNN model
fpr_knn, tpr_knn, _ = roc_curve(y_test, KNN_clf_final.predict_proba(X_test_rfe)[:,1])
```

Naive Bayes

In [78]:
```
# Instantiate NB classifier
baseline_nbayes_clf = GaussianNB()

# Fit model
baseline_nbayes_clf.fit(X_train_resample_scaled, y_train_resample)
```

Out[78]:
```
▾ GaussianNB
GaussianNB()
```

In [79]:
```
# Accuracy in training data
print(f"accuracy score: {accuracy_score(y_train_resample, baseline_nbayes_clf.predict(X_train_resample_scaled))}")
print(f"f1 score: {f1_score(y_train_resample, baseline_nbayes_clf.predict(X_train_resample_scaled))}")
print(f"precision score: {precision_score(y_train_resample, baseline_nbayes_clf.predict(X_train_resample_scaled))}")
print(f"recall score: {recall_score(y_train_resample, baseline_nbayes_clf.predict(X_train_resample_scaled))}")
```

```
accuracy score: 0.6946257684263895
f1 score: 0.7131759873805769
precision score: 0.672334455667789
recall score: 0.7593003358894733
```

In [80]:
```
# Accuracy in testing data
print(f"accuracy score: {accuracy_score(y_test, baseline_nbayes_clf.predict(X_test_scaled))}")
print(f"f1 score: {f1_score(y_test, baseline_nbayes_clf.predict(X_test_scaled))}")
print(f"precision score: {precision_score(y_test, baseline_nbayes_clf.predict(X_test_scaled))}")
print(f"recall score: {recall_score(y_test, baseline_nbayes_clf.predict(X_test_scaled))}")
```

```
accuracy score: 0.6350157256252809
f1 score: 0.44169530355097364
precision score: 0.32766825288919105
recall score: 0.6774420238931834
```

Feature Selection

In [81]:
```
def custom_importance_getter_gnb(estimator, X_train, y_train):
    # Fit the Gaussian Naive Bayes estimator
    estimator.fit(X_train, y_train)
    # Compute mutual information between each feature and the target variable
    feature_importances = mutual_info_classif(X_train, y_train)
    return feature_importances
```

In [82]:
```
n_bayes_clf = GaussianNB()
custom_importance_getter_gnb(n_bayes_clf, X_train_resample_scaled, y_train_resample)
```

Out[82]:
```
array([0.12125528, 0.10002706, 0.01802037, 0.03889712, 0.02572912,
       0.03155556, 0.03373206, 0.02918205, 0.04302464, 0.15632653,
       0.0678264 , 0.03016352, 0.05197852, 0.16991583, 0.14978895,
       0.08353543, 0.07620233, 0.05775088, 0.02567743, 0.03681407,
       0.03846622, 0.03220682, 0.02865216, 0.05737631, 0.04988383,
       0.00493569, 0.01687951, 0.01893362, 0.04288077, 0.07417283,
       0.05685349, 0.0336548 , 0.0388832 , 0.05023619, 0.00447644,
       0.053643  , 0.01371701, 0.05083981, 0.0215129 , 0.00444403,
       0.0404511 , 0.01993295, 0.03854789, 0.03956454, 0.03089162,
       0.05137103, 0.02643983, 0.03920495, 0.04051411, 0.06028132,
       0.06171391, 0.10447909])
```

In [83]:
```
# Fit the Gaussian Naive Bayes estimator
estimator = GaussianNB()

# Compute feature importances using the custom function
feature_importances = custom_importance_getter_gnb(estimator, X_train_resample_scaled, y_train_resample)

# Get the indices of features sorted by importance
sorted_indices = feature_importances.argsort()[::-1]

# Get feature names
feature_names = X_train_resample.columns

# Get feature names and their rankings
feature_rankings = [[feature_names[i], rank + 1] for rank, i in enumerate(sorted_indices)]

print("Feature rankings:")
pd.DataFrame(feature_rankings, columns=['Feature', 'Rank'])
```

```
Feature rankings:
```

Out[83]:

	Feature	Rank
0	opinion_h1n1_vacc_effective	1
1	doctor_recc_h1n1	2
2	opinion_h1n1_risk	3
3	h1n1_concern	4
4	behavior_score	5
5	h1n1_knowledge	6
6	opinion_h1n1_sick_from_vacc	7
7	sex_Male	8
8	household_adults	9
9	chronic_med_condition	10
10	census_msa_Non-MSA	11
11	education_Some College	12
12	education_College Graduate	13
13	census_msa_MSA, Principle City	14
14	income_poverty_> $75,000	15
15	employment_status_Not in Labor Force	16
16	household_children	17
17	health_worker	18
18	marital_status_Not Married	19
19	hhs_geo_region_lzgpxyit	20
20	rent_or_own_Rent	21
21	race_White	22
22	age_group_45 - 54 Years	23
23	behavioral_touch_face	24
24	age_group_55 - 64 Years	25

	Feature	Rank
25	behavioral_avoidance	26
26	hhs_geo_region_qufhixun	27
27	hhs_geo_region_bhuqouqj	28
28	behavioral_large_gatherings	29
29	behavioral_wash_hands	30
30	hhs_geo_region_fpwskwrf	31
31	behavioral_outside_home	32
32	child_under_6_months	33
33	hhs_geo_region_kbazzjca	34
34	age_group_65+ Years	35
35	hhs_geo_region_oxchjgsf	36
36	hhs_geo_region_mlyzmhmf	37
37	income_poverty_nan	38
38	income_poverty_Below Poverty	39
39	age_group_35 - 44 Years	40
40	hhs_geo_region_lrircsnp	41
41	education_< 12 Years	42
42	behavioral_face_mask	43
43	race_Hispanic	44
44	race_Other or Multiple	45
45	employment_status_Unemployed	46
46	behavioral_antiviral_meds	47
47	hhs_geo_region_dqpwygqj	48
48	rent_or_own_nan	49
49	education_nan	50
50	employment_status_nan	51

	Feature	Rank
51	marital_status_nan	52

In [84]:
```
feature_rankings = pd.DataFrame(feature_rankings, columns=['Feature', 'Rank'])

select_columns = feature_rankings['Feature'][0:10]
X_train_resample[select_columns]
```

Out[84]:

	opinion_h1n1_vacc_effective	doctor_recc_h1n1	opinion_h1n1_risk	h1n1_concern	behavior_score	h1n1_knowledge	opinion_h1n1_sick_from_
0	4.000000	0.000000	2.0	1.000000	5.000000	1.000000	4.0(
1	4.000000	1.000000	4.0	3.000000	3.000000	1.000000	4.0(
2	3.000000	0.000000	3.0	2.000000	4.000000	1.000000	2.0(
3	4.000000	0.000000	2.0	2.000000	4.000000	1.000000	4.0(
4	5.000000	0.000000	2.0	2.000000	3.000000	2.000000	2.0(
...	...	...	...	...	...	...	
31553	5.000000	0.165731	4.0	2.000000	4.000000	1.834269	1.1(
31554	4.000000	0.000000	4.0	2.324656	6.000000	2.000000	5.0(
31555	4.640571	0.000000	4.0	2.000000	2.640571	2.000000	2.0(
31556	4.562608	0.000000	2.0	2.000000	2.437392	2.000000	1.5(
31557	5.000000	0.000000	4.0	2.000000	4.000000	2.000000	1.0(

31558 rows × 10 columns

In [85]:
```
X_train_resample_scaled_select_columns = scaler.fit_transform(X_train_resample[select_columns])
X_test_scaled_select_columns = scaler.transform(X_test[select_columns])
```

In [86]:
```
n_bayes_clf2 = GaussianNB()

# Fit model
n_bayes_clf2.fit(X_train_resample_scaled_select_columns, y_train_resample)
```

Out[86]:
```
▾ GaussianNB
GaussianNB()
```

In [87]:
```
# Accuracy in training data
print(f"accuracy score: {accuracy_score(y_train_resample, n_bayes_clf2.predict(X_train_resample_scaled_select_columns)
print(f"f1 score: {f1_score(y_train_resample, n_bayes_clf2.predict(X_train_resample_scaled_select_columns))}")
print(f"precision score: {precision_score(y_train_resample, n_bayes_clf2.predict(X_train_resample_scaled_select_column
print(f"recall score: {recall_score(y_train_resample, n_bayes_clf2.predict(X_train_resample_scaled_select_columns))}")
```

```
accuracy score: 0.7465301983649154
f1 score: 0.7415425377233514
precision score: 0.7564271588661833
recall score: 0.7272323974903353
```

In [88]:
```
# Accuracy in testing data
print(f"accuracy score: {accuracy_score(y_test, n_bayes_clf2.predict(X_test_scaled_select_columns))}")
print(f"f1 score: {f1_score(y_test, n_bayes_clf2.predict(X_test_scaled_select_columns))}")
print(f"precision score: {precision_score(y_test, n_bayes_clf2.predict(X_test_scaled_select_columns))}")
print(f"recall score: {recall_score(y_test, n_bayes_clf2.predict(X_test_scaled_select_columns))}")
```

```
accuracy score: 0.7516848884229445
f1 score: 0.553821313240043
precision score: 0.44875708678587006
recall score: 0.7231201686577653
```

Cross validate

In [89]:
```
results = cross_validate(GaussianNB(), X_train_resample_scaled_select_columns, y_train_resample,
              scoring = ['f1', 'accuracy', 'precision', 'recall'])
```

In [90]:
```
print(f"test f1 score with cross validation: {np.mean(results['test_f1'])}")
print(f"test accuracy with cross validation: {np.mean(results['test_accuracy'])}")
print(f"test precision with cross validation: {np.mean(results['test_precision'])}")
print(f"test recall with cross validation: {np.mean(results['test_recall'])}")
```

```
test f1 score with cross validation: 0.7394549040486306
test accuracy with cross validation: 0.7445341979391407
test precision with cross validation: 0.7543987747120069
test recall with cross validation: 0.72520555016581
```

In [91]:
```
# ROC curve for the best Gausssian Naive Bayes model
fpr_gnbayes, tpr_gnbayes, _ = roc_curve(y_test, n_bayes_clf2.predict_proba(X_test_scaled_select_columns)[:,1])
```

Decision Tree

In [92]:
```
# Initialize Decision Tree classifier
dt_classifier = DecisionTreeClassifier(random_state=40)

# Train the model
dt_classifier.fit(X_train_resample, y_train_resample)

# Make predictions on the testing set
y_pred = dt_classifier.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

# Classification report
print("Classification Report:")
print(classification_report(y_test, y_pred))

# Confusion matrix
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
```

```
Accuracy: 0.7397034596375618
Classification Report:
              precision    recall  f1-score   support

           0       0.85      0.82      0.83      5254
           1       0.40      0.45      0.43      1423

    accuracy                           0.74      6677
   macro avg       0.62      0.64      0.63      6677
weighted avg       0.75      0.74      0.75      6677

Confusion Matrix:
[[4295  959]
 [ 779  644]]
```

In [93]:
```
# Define the hyperparameters 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
dt_classifier = DecisionTreeClassifier(random_state=40)

# Initialize GridSearchCV
grid_search = GridSearchCV(estimator=dt_classifier, param_grid=param_grid, cv=5, scoring='accuracy')

# Perform grid search
grid_search.fit(X_train_resample, y_train_resample)

# 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 Score:", best_score)
```

```
Best Parameters: {'criterion': 'entropy', 'max_depth': 20, 'min_samples_leaf': 4, 'min_samples_split': 2}
Best Score: 0.8422947921819166
```

In [94]:
```
# Initialize the decision tree classifier with best parameters
best_dt_classifier = DecisionTreeClassifier(criterion='entropy', max_depth=20, min_samples_leaf=4, min_samples_split=2

# Train the model with best parameters
best_dt_classifier.fit(X_train_resample, y_train_resample)

# Make predictions on the testing set
y_pred_best = best_dt_classifier.predict(X_test)

# Evaluate the model with best parameters
accuracy_best = accuracy_score(y_test, y_pred_best)
print("Accuracy with best parameters:", accuracy_best)

# Classification report
print("Classification Report with best parameters:")
print(classification_report(y_test, y_pred_best))

# Confusion matrix
print("Confusion Matrix with best parameters:")
print(confusion_matrix(y_test, y_pred_best))
```

```
Accuracy with best parameters: 0.7795417103489591
Classification Report with best parameters:
              precision    recall  f1-score   support

           0       0.85      0.88      0.86      5254
           1       0.48      0.42      0.45      1423

    accuracy                           0.78      6677
   macro avg       0.66      0.65      0.65      6677
weighted avg       0.77      0.78      0.77      6677

Confusion Matrix with best parameters:
[[4611  643]
 [ 829  594]]
```

Feature selection

In [95]:
```
# Initialize RFE with the classifier and desired number of features to select
rfe = RFE(estimator= best_dt_classifier, n_features_to_select= .8, step= 1)  # Select top 80% features

# Fit RFE on train data set
rfe.fit(X_train_resample, y_train_resample)
print(X_train_resample.columns)
print(rfe.ranking_)
print(rfe.support_)
```

```
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', 'household_adults',
       'household_children', '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', 'education_nan',
       'race_Hispanic', 'race_Other or Multiple', 'race_White', 'sex_Male',
       'income_poverty_> $75,000', 'income_poverty_Below Poverty',
       'income_poverty_nan', 'marital_status_Not Married',
       'marital_status_nan', 'rent_or_own_Rent', 'rent_or_own_nan',
       'employment_status_Not in Labor Force', 'employment_status_Unemployed',
       'employment_status_nan', 'hhs_geo_region_bhuqouqj',
       'hhs_geo_region_dqpwygqj', 'hhs_geo_region_fpwskwrf',
       'hhs_geo_region_kbazzjca', 'hhs_geo_region_lrircsnp',
       'hhs_geo_region_lzgpxyit', 'hhs_geo_region_mlyzmhmf',
       'hhs_geo_region_oxchjgsf', 'hhs_geo_region_qufhixun',
       'census_msa_MSA, Principle City', 'census_msa_Non-MSA',
       'behavior_score'],
      dtype='object')
[ 1  1  1  1  4  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  8  1
  1  9  2  3  1  1  1  5  1  1 11  1 10  1  7 12  1  6  1  1  1  1  1  1
  1  1  1  1]
[ True  True  True  True False  True  True  True  True  True  True  True
  True  True  True  True  True  True  True  True  True  True False  True
  True False False False  True  True  True False  True  True False  True
 False  True False False  True False  True  True  True  True  True  True
  True  True  True  True]
```

In [96]:
```
masked_columns = X_train_resample.columns[rfe.get_support(indices= True)]
masked_columns
```

Out[96]:
```
Index(['h1n1_concern', 'h1n1_knowledge', 'behavioral_antiviral_meds',
       'behavioral_avoidance', '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', 'household_adults',
       'household_children', 'age_group_35 - 44 Years',
       'age_group_45 - 54 Years', 'age_group_55 - 64 Years',
       'age_group_65+ Years', 'education_College Graduate',
       'education_Some College', 'race_White', 'sex_Male',
       'income_poverty_> $75,000', 'income_poverty_nan',
       'marital_status_Not Married', 'rent_or_own_Rent',
       'employment_status_Not in Labor Force', 'hhs_geo_region_bhuqouqj',
       'hhs_geo_region_fpwskwrf', 'hhs_geo_region_kbazzjca',
       'hhs_geo_region_lrircsnp', 'hhs_geo_region_lzgpxyit',
       'hhs_geo_region_mlyzmhmf', 'hhs_geo_region_oxchjgsf',
       'hhs_geo_region_qufhixun', 'census_msa_MSA, Principle City',
       'census_msa_Non-MSA', 'behavior_score'],
      dtype='object')
```

In [97]:
```
select_colums = []
for column in X_train_resample.columns:
    if column not in masked_columns:
        select_colums.append(column)

select_colums
```

Out[97]:
```
['behavioral_face_mask',
 'education_< 12 Years',
 'education_nan',
 'race_Hispanic',
 'race_Other or Multiple',
 'income_poverty_Below Poverty',
 'marital_status_nan',
 'rent_or_own_nan',
 'employment_status_Unemployed',
 'employment_status_nan',
 'hhs_geo_region_dqpwygqj']
```

In [98]:
```
X_train_rfe = X_train_resample[select_colums]
X_test_rfe = X_test[select_colums]
```

In [99]:
```
# Initialize the decision tree classifier with best parameters
best_dt_classifier_with_rfe = DecisionTreeClassifier(criterion='entropy', max_depth=20, min_samples_leaf=4, min_sample:

# Train the model with best parameters
best_dt_classifier_with_rfe.fit(X_train_rfe, y_train_resample)

# Make predictions on the testing set
y_pred_best = best_dt_classifier_with_rfe.predict(X_test_rfe)

# Evaluate the model with best parameters
accuracy_best = accuracy_score(y_test, y_pred_best)
print("Accuracy with best parameters:", accuracy_best)

# Classification report
print("Classification Report with best parameters:")
print(classification_report(y_test, y_pred_best))

# Confusion matrix
print("Confusion Matrix with best parameters:")
print(confusion_matrix(y_test, y_pred_best))
```

```
Accuracy with best parameters: 0.7867305676201887
Classification Report with best parameters:
              precision    recall  f1-score   support

           0       0.79      1.00      0.88      5254
           1       0.44      0.00      0.01      1423

    accuracy                           0.79      6677
   macro avg       0.62      0.50      0.44      6677
weighted avg       0.71      0.79      0.69      6677

Confusion Matrix with best parameters:
[[5249    5]
 [1419    4]]
```

In [100]:
```
# ROC curve for the best Decision tree model
fpr_dtree, tpr_dtree, _ = roc_curve(y_test, best_dt_classifier_with_rfe.predict_proba(X_test_rfe)[:,1])
```

Random Forest

In [118]:
```
def create_models(seed=40):
    models =[]
    models.append(('random_forest', RandomForestClassifier(random_state=seed)))
    return models
models= create_models()
```

In [102]:
```
# results using default parameters
results= []
names=[]
scoring = 'accuracy'
for name, model in models:
    # fit model with training data
    model.fit(X_train_resample, y_train_resample).predict(X_test)
    # make predictions with testing data
    predictions=model.predict(X_test)
    # calculating accuracy
    accuracy = accuracy_score(y_test, predictions)
    # append model name and accuracy to the lists
    results.append(accuracy)
    names.append(name)
    #print classifier accuracy
    print('classifier:{}, Accuracy score: {})'.format(name, accuracy))
    print(classification_report(y_test, predictions))
```

```
classifier:random_forest, Accuracy score: 0.8352553542009885)
              precision    recall  f1-score   support

           0       0.86      0.94      0.90      5254
           1       0.66      0.46      0.54      1423

    accuracy                           0.84      6677
   macro avg       0.76      0.70      0.72      6677
weighted avg       0.82      0.84      0.82      6677
```

In [103]:
```
def perform_grid_search(classifer, param_grid):
    pipe = Pipeline([
        ('scaler', StandardScaler()),
        ('classifier', RandomForestClassifier())
    ])

    random_search = GridSearchCV(estimator = pipe,
                       param_grid = param_grid,
                       scoring = 'accuracy',
                       cv = 3,
                       verbose =1)
    random_search.fit(X_train_resample, y_train_resample)

    best_params = random_search.best_params_
    print("Best Parameters:",  best_params)

    # Evaluate the model on the test set
    y_pred = random_search.predict(X_test)


    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)



    print("Test Accuracy:", accuracy)
    print("Precision:", precision)
    print("Recall:", recall)
    print("F1 score:", f1)

    # classifiaction reports
    print("Classification Report:")
    print(classification_report(y_test, y_pred))

    return best_params, accuracy
```

In [104]:
```
classifier = RandomForestClassifier()
param_grid = [{'classifier__max_depth': [5,10],
        'classifier__min_samples_split': [4,6]}]
perform_grid_search(classifier, param_grid)
```

```
Fitting 3 folds for each of 4 candidates, totalling 12 fits
Best Parameters: {'classifier__max_depth': 10, 'classifier__min_samples_split': 4}
Test Accuracy: 0.8301632469672008
Precision: 0.6040316774658028
Recall: 0.5895994378074491
F1 score: 0.5967283072546231
Classification Report:
              precision    recall  f1-score   support

           0       0.89      0.90      0.89      5254
           1       0.60      0.59      0.60      1423

    accuracy                           0.83      6677
   macro avg       0.75      0.74      0.74      6677
weighted avg       0.83      0.83      0.83      6677
```

Out[104]:
```
({'classifier__max_depth': 10, 'classifier__min_samples_split': 4},
 0.8301632469672008)
```

In [105]:
```
# Final random forest model
# Instantiate classifier
random_forest_final = RandomForestClassifier(max_depth= 10, min_samples_split= 4)

# Fit model
random_forest_final.fit(X_train_resample, y_train_resample)
```

Out[105]:
```
RandomForestClassifier

RandomForestClassifier(max_depth=10, min_samples_split=4)
```

In [106]:
```
# ROC curve for the best Decision tree model
fpr_rforest, tpr_rforest, _ = roc_curve(y_test, random_forest_final.predict_proba(X_test)[:,1])
```

XGBoost

In [107]:
```
# Clean feature names
clean_feature_names = [name.replace('[', '').replace(']', '').replace('<', '') for name in X_train_resample.columns]
X_train_resample.columns = clean_feature_names

X_test.columns = [name.replace('education_< ', 'education_ ') for name in X_test.columns]

# Initialize the XGBoost model
xgboost = XGBClassifier()

# Train the XGBoost model
xgboost.fit(X_train_resample, y_train_resample)

# Make predictions on the training set
y_train_pred = xgboost.predict(X_train_resample)

# Calculate evaluation metrics for training set
train_accuracy = accuracy_score(y_train_resample, y_train_pred)
train_precision = precision_score(y_train_resample, y_train_pred)
train_recall = recall_score(y_train_resample, y_train_pred)
train_f1_score = f1_score(y_train_resample, y_train_pred)

# Make predictions on the test set
y_test_pred = xgboost.predict(X_test)

print("confusion_matrix for XGBoost\n",confusion_matrix(y_test,y_test_pred))


print(classification_report(y_test, y_test_pred))


print("Test Accuracy Score:", accuracy_score(y_test, y_test_pred))
```

```
confusion_matrix for XGBoost
 [[4890  364]
 [ 755  668]]
              precision    recall  f1-score   support

           0       0.87      0.93      0.90      5254
           1       0.65      0.47      0.54      1423

    accuracy                           0.83      6677
   macro avg       0.76      0.70      0.72      6677
weighted avg       0.82      0.83      0.82      6677

Test Accuracy Score: 0.8324097648644601
```

In [108]:
```
# Define the hyperparameters to tune

param_grid = {
    'max_depth': [ 5, 7, 9],
    'learning_rate': [0.1, 0.01],
    'n_estimators': [100, 300],
    'subsample': [0.8, 1.0],
    'colsample_bytree': [0.8, 1.0]
}

grid_clf = GridSearchCV(xgboost, param_grid, scoring='accuracy', cv=None, n_jobs=1)
grid_clf.fit(X_train_resample, y_train_resample)

best_parameters = grid_clf.best_params_

print('Grid Search found the following optimal parameters: ')
for param_name in sorted(best_parameters.keys()):
    print('%s: %r' % (param_name, best_parameters[param_name]))

training_preds = grid_clf.predict(X_train_resample)
test_preds = grid_clf.predict(X_test)
training_accuracy = accuracy_score(y_train_resample, training_preds)
test_accuracy = accuracy_score(y_test, test_preds)

print('')
print('Training Accuracy: {:.4}%'.format(training_accuracy * 100))
print('Validation accuracy: {:.4}%'.format(test_accuracy * 100))
```

```
Grid Search found the following optimal parameters:
colsample_bytree: 0.8
learning_rate: 0.01
max_depth: 9
n_estimators: 300
subsample: 0.8

Training Accuracy: 90.91%
Validation accuracy: 83.8%
```

In [109]:
```
# Final XGBoost Model
# Instantiate Classifer
xgboost_final = XGBClassifier(colsample_bytree= 0.8, learning_rate= 0.01, max_depth= 9, n_estimators= 300, subsample= (

# Fit model
xgboost_final.fit(X_train_resample, y_train_resample)
```

Out[109]:
```
XGBClassifier

XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=0.8, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=0.01, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=9, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              multi_strategy=None, n_estimators=300, n_jobs=None,
```

In [110]:
```
# ROC curve for the best Decision tree model
fpr_xgboost, tpr_xgboost, _ = roc_curve(y_test, xgboost_final.predict_proba(X_test)[:,1])
```

Summary of all the tuned models using an Accuracy Scores:

1. Logistic Regression - **0.7735**
2. Decision Tree - **0.7814**
3. KNN - **0.7811**
4. Naive Bayes - **0.7528**
5. Random Forest - **0.8288**
6. XGBoost - **0.8405**

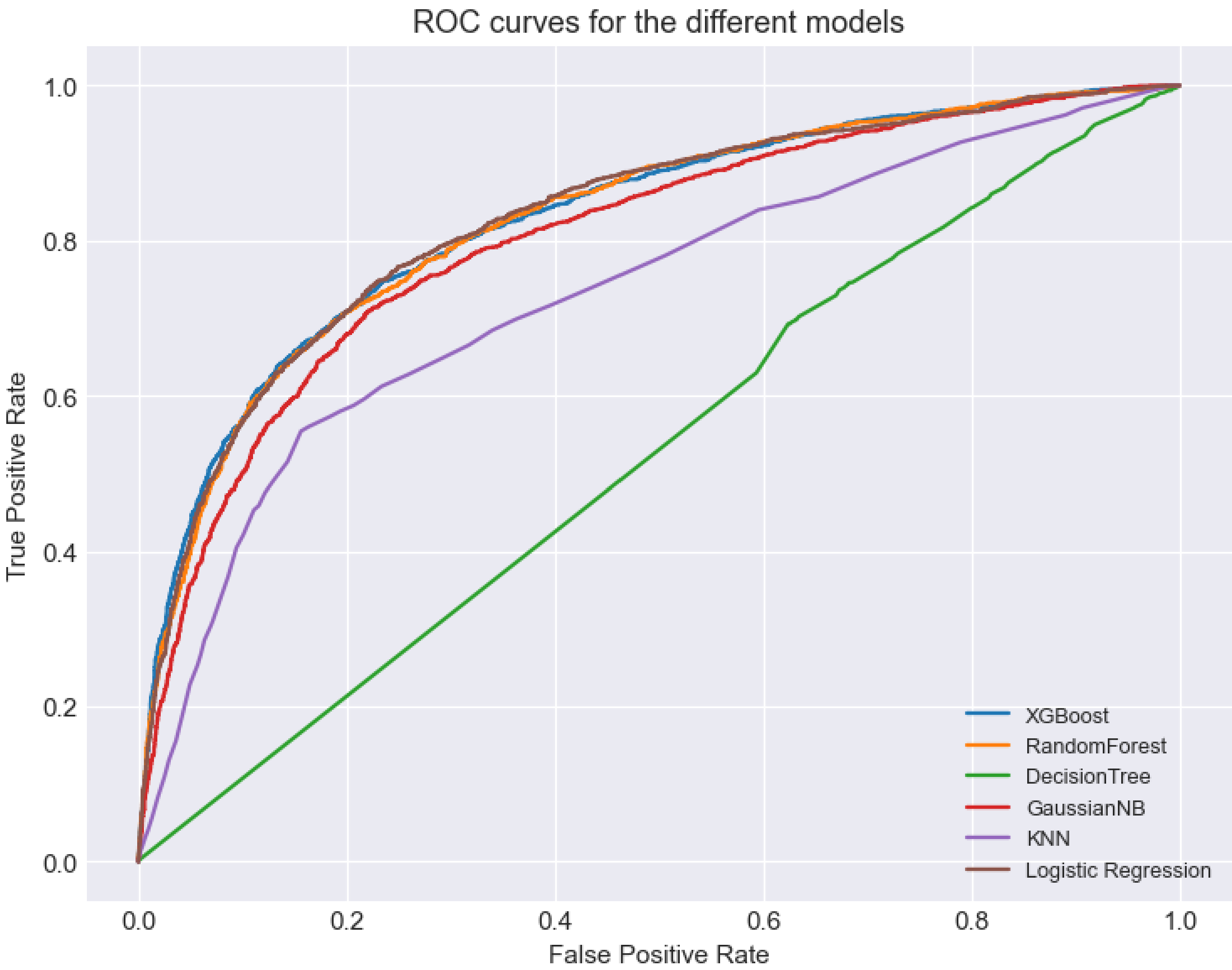
The best model for predicting vaccine uptake:

- XGBoost is therefore our **best and final model** with the best Accuracy of **0.8405**, as compared to all other models

ROC Curve

Summary of all the tuned models using an ROC curve:

In [111]:
```
plt.figure(figsize=(8,6), dpi=110)
plt.plot(fpr_xgboost, tpr_xgboost, label='XGBoost')
plt.plot(fpr_rforest, tpr_rforest, label='RandomForest')
plt.plot(fpr_dtree, tpr_dtree, label='DecisionTree')
plt.plot(fpr_gnbayes, tpr_gnbayes, label='GaussianNB')
plt.plot(fpr_knn, tpr_knn, label='KNN')
plt.plot(fpr_logreg, tpr_logreg, label='Logistic Regression')
plt.title('ROC curves for the different models')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend(fontsize= 'small', loc='lower right')
plt.show()
```



Conclusion

Features that were most important in predicting H1N1 vaccine uptake:

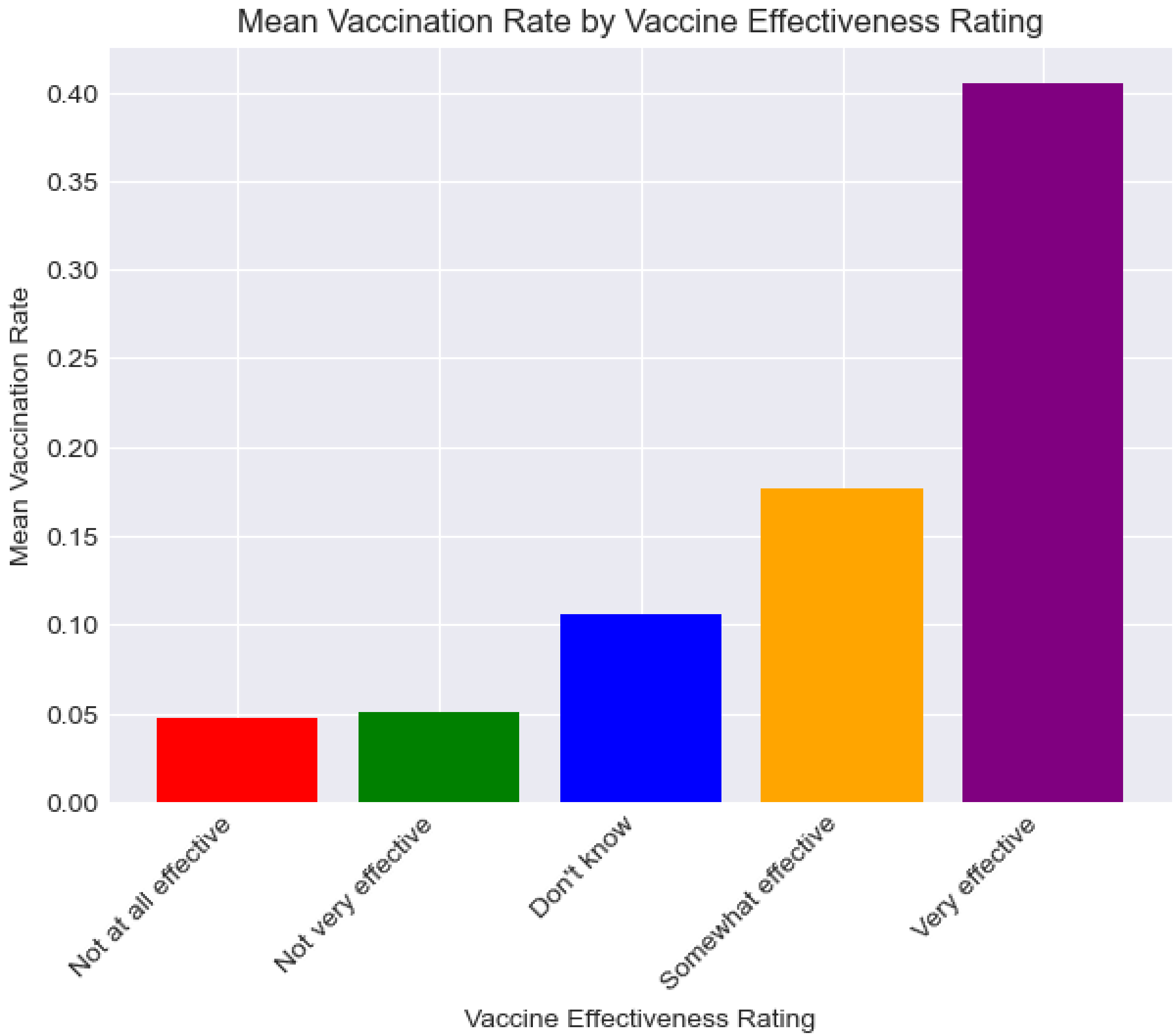
1. Opinion on H1N1 vaccine effectiveness.
2. Doctor's recommendation.
3. H1N1 concern.
4. Gender.
5. H1N1 Knowledge.
6. Perceived side-effects from H1N1 vaccine.
7. Chronic medical condition.
8. Number of adults in an household.

In [112]:
```
# Define a function to generate the plots for the important features

def create_vaccination_rate_plot(feature, target, data, x_label, y_label, plot_title,
                                 x_tick_labels=None, plot_width=7, plot_height=5):
    mean_vaccination_rate = data.groupby(feature)[target].mean().reset_index()

    # Define a list of colors to assign to each category
    colors = ['red', 'green', 'blue', 'orange', 'purple', 'black']

    fig, ax = plt.subplots(figsize=(plot_width, plot_height))
    ax.bar(mean_vaccination_rate[feature], mean_vaccination_rate[target],
           color=colors[:len(mean_vaccination_rate[feature])])

    ax.set_title(plot_title)
    ax.set_xlabel(x_label)
    ax.set_ylabel(y_label)
    if x_tick_labels:
        ax.set_xticks(mean_vaccination_rate[feature])
        ax.set_xticklabels(x_tick_labels, rotation=45, ha='right')

    plt.show()
```

Plot of the top 5 most important features in predicting H1N1 vaccine uptake

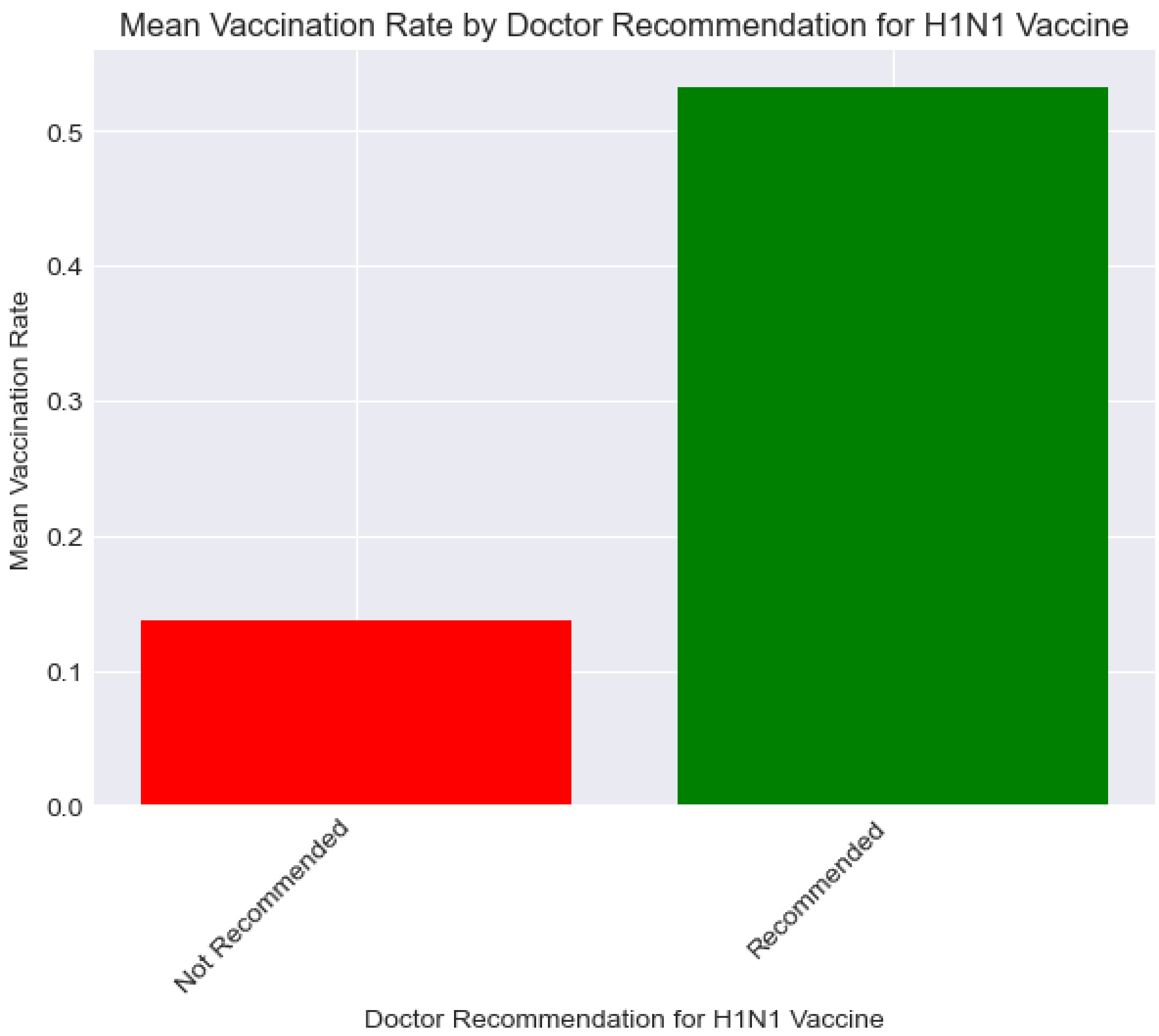
1. Respondent's opinion about H1N1 flu vaccine effectiveness.

In [113]:
```
fig = create_vaccination_rate_plot('opinion_h1n1_vacc_effective', 'h1n1_vaccine', merged_df,
                                   'Vaccine Effectiveness Rating',
                                   'Mean Vaccination Rate',
                                   'Mean Vaccination Rate by Vaccine Effectiveness Rating',
                                    x_tick_labels=['Not at all effective', 'Not very effective', "Don't know",
                                                   'Somewhat effective', 'Very effective'],
                                    plot_width=7, plot_height=5)

plt.show()
```



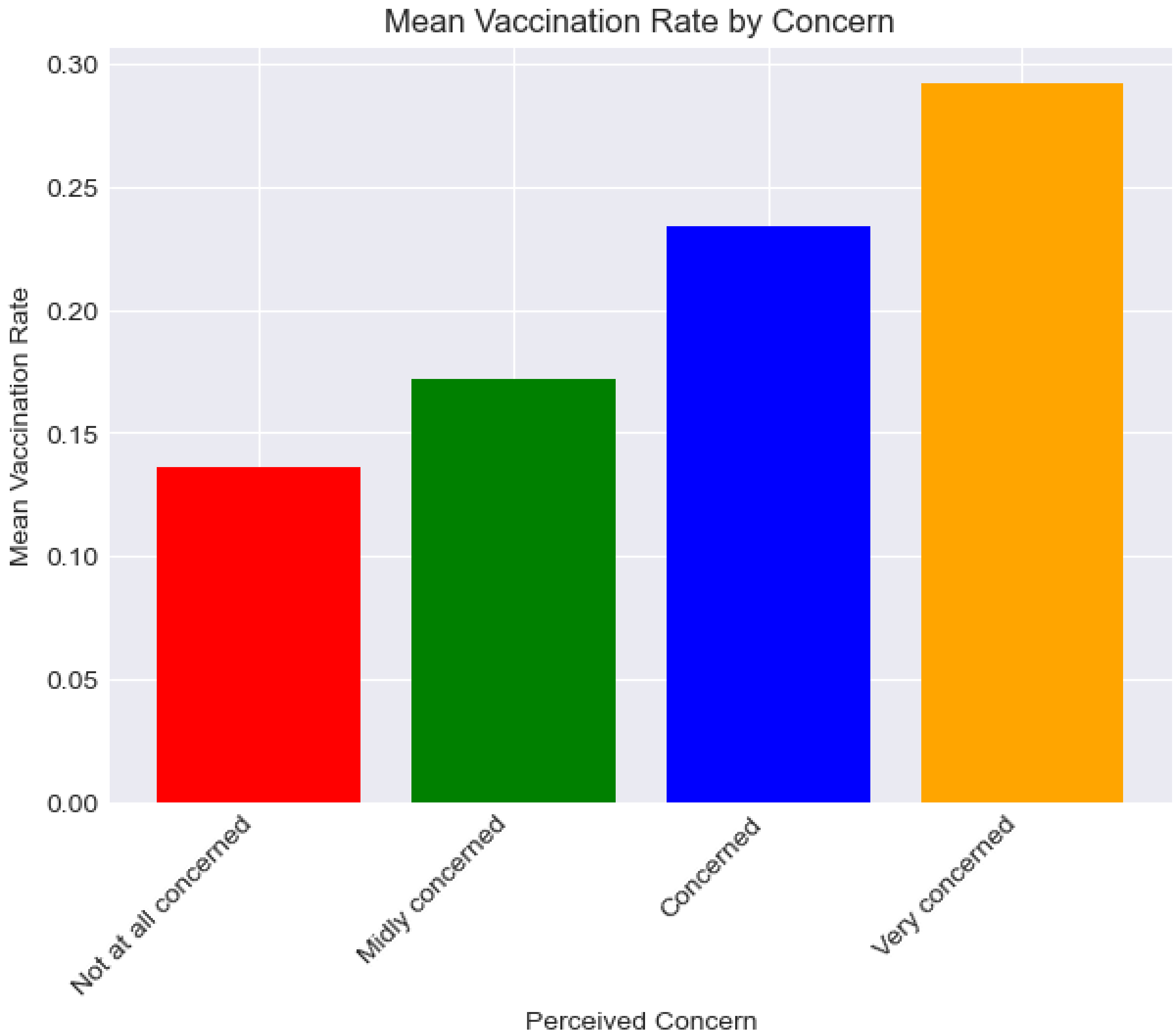
2. Doctor's recommendation.

In [114]:
```
fig = create_vaccination_rate_plot('doctor_recc_h1n1', 'h1n1_vaccine', merged_df,
                                   'Doctor Recommendation for H1N1 Vaccine',
                                   'Mean Vaccination Rate',
                                   'Mean Vaccination Rate by Doctor Recommendation for H1N1 Vaccine',
                                    x_tick_labels=['Not Recommended', 'Recommended'],
                                    plot_width=7, plot_height=5)

plt.show()
```



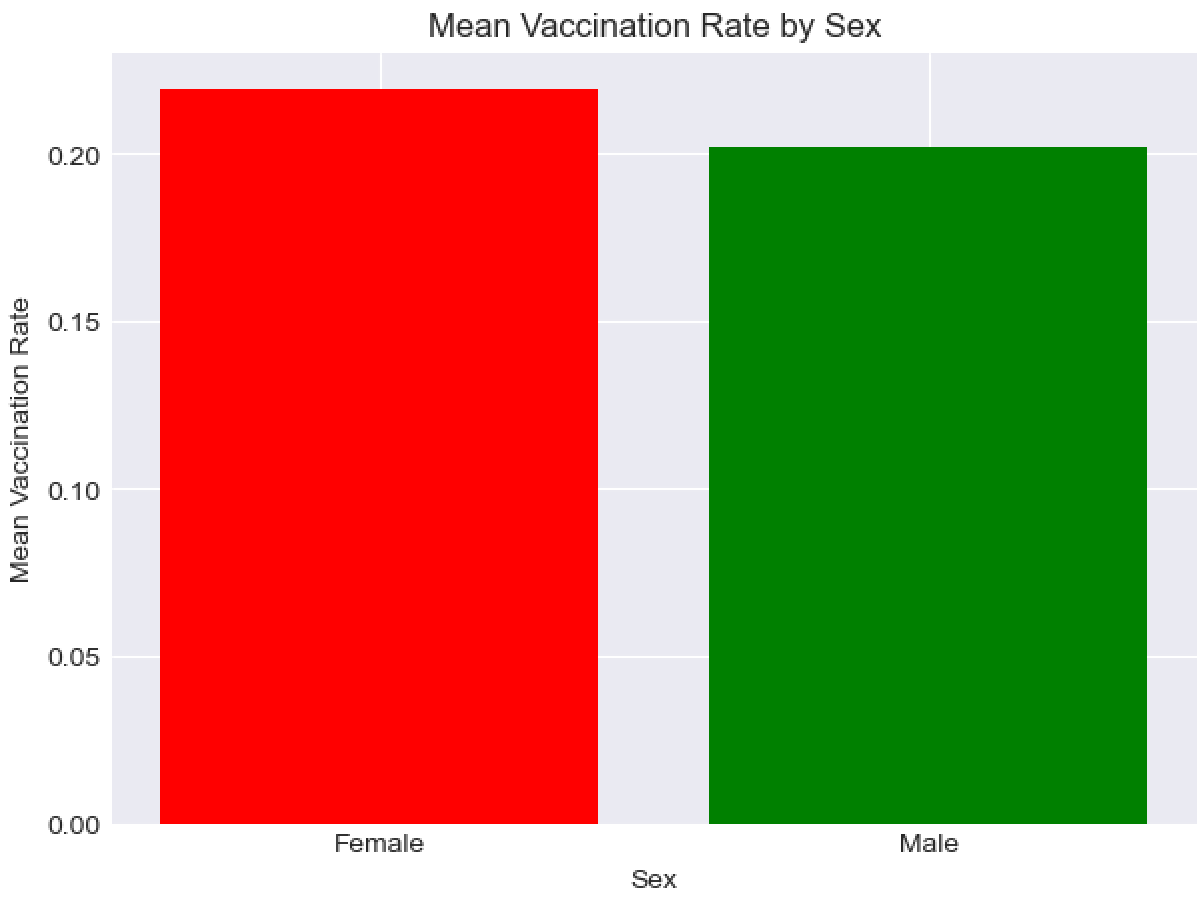
3. H1N1 concern

In [115]:
```
fig = create_vaccination_rate_plot('h1n1_concern', 'h1n1_vaccine', merged_df,
                                   'Perceived Concern',
                                   'Mean Vaccination Rate',
                                   'Mean Vaccination Rate by Concern',
                                    x_tick_labels=['Not at all concerned', 'Midly concerned',"Concerned",
                                                   'Very concerned'],
                                    plot_width=7, plot_height=5)

plt.show()
```


Mean Vaccination Rate by Concern
0.30
0.25
0.20
0.15
0.10
0.05
0.00
Mean Vaccination Rate
Not at all concerned
Midly concerned
Concerned
Very concerned
Perceived Concern

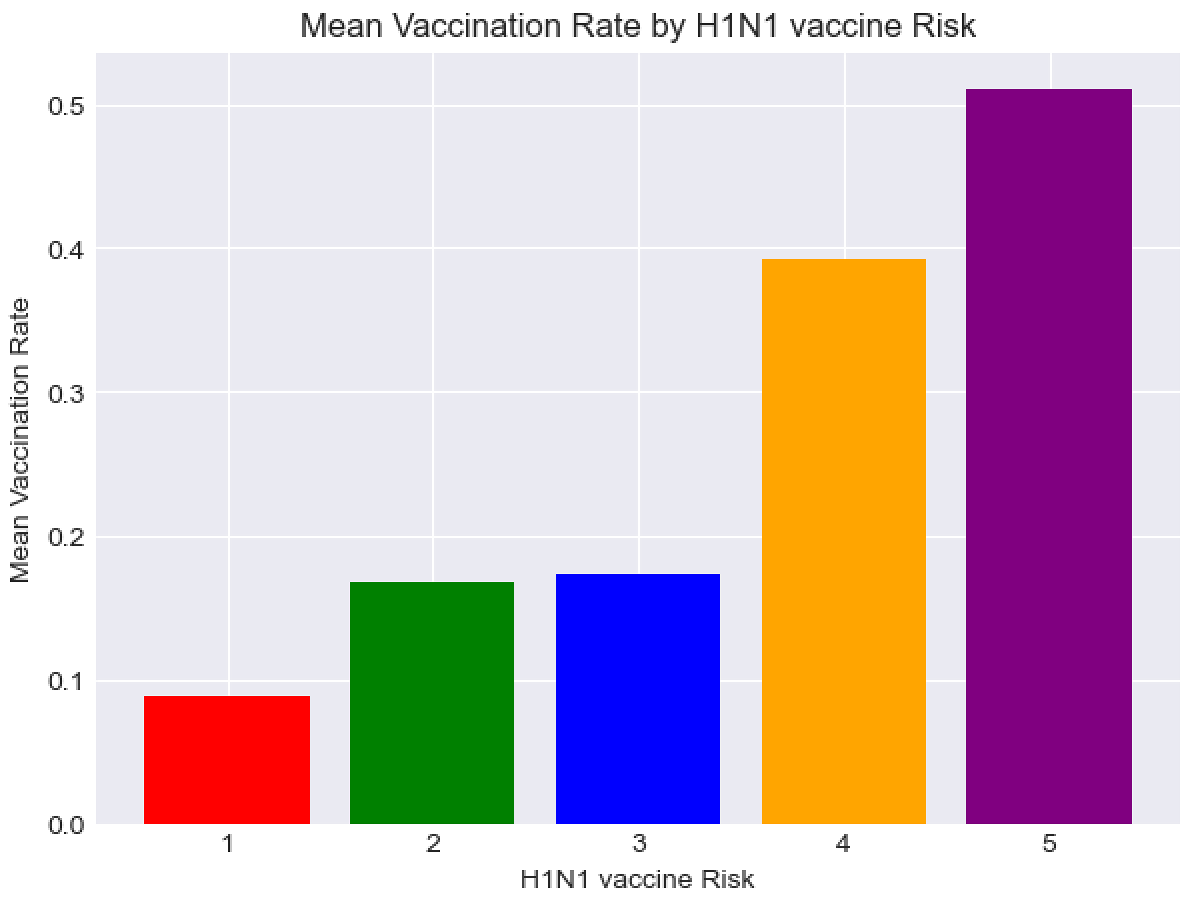
4. Gender

In [116]:
```
fig = create_vaccination_rate_plot('sex', 'h1n1_vaccine', merged_df,
                                   'Sex',
                                   'Mean Vaccination Rate',
                                   'Mean Vaccination Rate by Sex',
                                    x_tick_labels=[],
                                    plot_width=7, plot_height=5)

plt.show()
```



5. Perceived side effects from H1N1 vaccine

In [117]:
```
fig = create_vaccination_rate_plot('opinion_h1n1_risk', 'h1n1_vaccine', merged_df,
                                   'H1N1 vaccine Risk',
                                   'Mean Vaccination Rate',
                                   'Mean Vaccination Rate by H1N1 vaccine Risk',
                                    x_tick_labels=[],
                                    plot_width=7, plot_height=5)

plt.show()
```



Recommendations

1. Dispel vaccine myths and promote preventative measures against the flu.
2. Identify high risk groups partcularly those with lower levels of education and indviduals expressing low concern for H1N1, to implement targeted messaging to them.
3. Utilize predictive model for efficent resource allocation on the vaccination campaign.
4. Implement policies that emphasize on face masks, minimizing gatherings and hand washing.
5. Collaboration with health experts to encouraage medical checkups and doctor's recommendation. tion

Next Steps

1. Monitoring and Assessment: Keep a close eye on the results of the interventions and communication tactics used. Compare the vaccination uptake rates before and after the interventions to assess the efficacy of various strategies.

2. Update the predictive model often with fresh information to enhance future interventions and targeting tactics.